Almacenamiento y Recuperación de la Información

Notas de clase basadas en los libros File Structures de Folk y Algorithms de Sedgewick

Dr. Francisco Javier Zaragoza Martínez
franz@correo.azc.uam.mx

UAM Azcapotzalco
Departamento de Sistemas

Trimestre 2009 Invierno

Contenido

1. Estructuras de archivos
2. Grafos y sus aplicaciones
3. Árboles AVL
4. Ordenamiento externo
5. Índices
6. Árboles B y B^+
7. Dispersión

Evaluación

- Habrá 3 exámenes (E) y 6 tareas (T).
- Cada examen valdrá 25 puntos.
- Cada tarea valdrá 5 puntos.
- S requiere $E \geq 35$, $T \geq 15$ y $E + T \geq 60$.
- B requiere $E \geq 40$, $T \geq 20$ y $E + T \geq 73$.
- MB requiere $E \geq 45$, $T \geq 25$ y $E + T \geq 87$.

Exámenes y tareas

- El primer examen consistirá únicamente del tema Estructuras de archivos (dos tareas).
- El segundo examen consistirá de los temas Grafos y Árboles AVL (dos tareas).
- El tercer examen consistirá de los demás temas (dos tareas).

Part I

Estructuras de archivos

Contenido

1. Diseño y especificación de estructuras de archivos
   - Objetivos de las estructuras de archivos
2. Operaciones fundamentales de procesamiento de archivos
3. Almacenamiento secundario
4. Conceptos fundamentales de estructuras de archivos
5. Manipulando archivos con registros
6. Organización de archivos para mejora del desempeño

Discos y memoria

- Los discos son muy lentos comparados con la memoria.
- Se puede obtener un dato de la memoria en aproximadamente 10 ns.
- Para obtener un dato del disco se necesitan aproximadamente 10 ms.
- Por lo tanto, los discos son unas 10^6 veces más lentos que la memoria.

Estructuras de datos y de archivos

- La diferencia de velocidad entre los discos y la memoria hace que las estructuras de datos en memoria no sirvan para resolver los problemas de datos almacenados en disco.
- Por lo tanto, debemos establecer nuevos objetivos para el diseño y especificación de estructuras de archivos.

Objetivos I

- Se desea obtener la información buscada con un solo acceso al disco.
- Si eso no es posible, se desean estructuras de archivos que nos permitan encontrar la información con tan pocos accesos al disco como sea posible.
- Queremos que la información relacionada quede agrupada, de modo que podamos obtenerla con un solo acceso al disco.

Objetivos II

- Se deben mantener todas estas propiedades aún cuando los archivos cambien, crezcan o se hagan más pequeños cuando se agregue o borre información.
- Las estructuras de datos deben aprovechar los dispositivos de almacenamiento, ya sean secuenciales o de acceso arbitrario: cintas, discos, discos compactos, etc.

Contenido

1. Diseño y especificación de estructuras de archivos
2. Operaciones fundamentales de procesamiento de archivos
   - Archivos físicos y lógicos
   - Apertura de archivos
   - Cerrado de archivos
   - Lectura y escritura de archivos
   - Tipos de acceso y búsqueda
   - Consideraciones especiales
3. Almacenamiento secundario
4. Conceptos fundamentales de estructuras de archivos
5. Manipulando archivos con registros

Archivos físicos y lógicos

- Un archivo físico es una colección de bytes almacenada en algún dispositivo (cinta, disco, disco compacto, etc.)
- Un archivo lógico es la visión de un programa de un archivo: el programa puede leer o escribir datos sin saber cómo ni dónde.

Contenido

1. Diseño y especificación de estructuras de archivos
2. Operaciones fundamentales de procesamiento de archivos
   - Archivos físicos y lógicos
   - Apertura de archivos
   - Cerrado de archivos
   - Lectura y escritura de archivos
   - Tipos de acceso y búsqueda
   - Consideraciones especiales
3. Almacenamiento secundario
4. Conceptos fundamentales de estructuras de archivos
5. Manipulando archivos con registros

Apertura de un archivo

- Antes de que un programa pueda usar un archivo lógico, éste se debe relacionar con un archivo físico.
- A esta operación se le llama apertura.
- Un archivo se puede abrir para escritura, lectura, sobreescritura, agregado y combinaciones de éstas.
- Puede tener acceso secuencial o arbitrario.
- En algunos sistemas operativos se pueden indicar los modos de uso y permisos.

Apertura de un archivo en C

- Abajo `fis` es el archivo físico, `fp` es el archivo lógico y `modo` es alguno de `w` (escritura) `r`, (lectura) o `a` (agregado) seguido posiblemente de + (actualización) o `b` (binario).

```
#include <stdio.h>

FILE *fp;

if ((fp = fopen(fis, modo)) == NULL) {
  /* no se pudo abrir el archivo */
} else {
  /* todo bien */
}
```

Contenido

1. Diseño y especificación de estructuras de archivos
2. Operaciones fundamentales de procesamiento de archivos
   - Archivos físicos y lógicos
   - Apertura de archivos
   - Cerrado de archivos
   - Lectura y escritura de archivos
   - Tipos de acceso y búsqueda
   - Consideraciones especiales
3. Almacenamiento secundario
4. Conceptos fundamentales de estructuras de archivos
5. Manipulando archivos con registros

Cerrado de un archivo

- La operación contraria se llama cerrado.
- Esto libera un archivo lógico en el programa que se puede reusar para otro archivo físico.
- También garantiza que toda la información enviada al archivo lógico quede almacenada en el archivo físico.
- Normalmente, el sistema operativo se encarga de cerrar todos los archivos que hayan quedado abiertos al finalizar la ejecución de un programa (aunque lo mejor es que lo haga el propio programa).

Cerrado de un archivo en C

- Abajo `fp` es el archivo lógico.

```
if (fclose(fp) == EOF) {
  /* no se pudo cerrar el archivo */
} else {
  /* todo bien */
}
```

Contenido

1. Diseño y especificación de estructuras de archivos
2. Operaciones fundamentales de procesamiento de archivos
   - Archivos físicos y lógicos
   - Apertura de archivos
   - Cerrado de archivos
   - Lectura y escritura de archivos
   - Tipos de acceso y búsqueda
   - Consideraciones especiales
3. Almacenamiento secundario
4. Conceptos fundamentales de estructuras de archivos
5. Manipulando archivos con registros

Lectura de archivos

- Al nivel más bajo se tiene al menos la operación

  `lee(archivo, destino, cuenta)`

  que lee `cuenta` bytes del `archivo` lógico colocándolos en la dirección `destino`.
- A niveles más altos, se pueden tener varias operaciones de lectura de caracteres, enteros, flotantes, cadenas, estructuras, etc.

Lectura de archivos en C

- Abajo `n` es el número de objetos de cierto `tipo` que se desean leer del archivo lógico `fp` en el arreglo `a` y `r` es el número de objetos que se leyeron en realidad.

```
size_t r, n;
tipo a[MAX];

r = fread(a, sizeof(tipo), n, fp);
```

Escritura de archivos

- Al nivel más bajo se tiene al menos la operación

  `escribe(archivo, fuente, cuenta)`

  que escribe `cuenta` bytes en el `archivo` lógico tomados a partir de la dirección `fuente`.
- A niveles más altos, se pueden tener varias operaciones de escritura de caracteres, enteros, flotantes, cadenas, estructuras, etc.

Escritura de archivos en C

- Abajo `n` es el número de objetos de cierto `tipo` que se desean escribir en el archivo lógico `fp` del arreglo `a` y `r` es el número de objetos que se escribieron en realidad.

```
size_t r, n;
tipo a[MAX];

r = fwrite(a, sizeof(tipo), n, fp);
```

Fin de archivo

- Normalmente se tiene una operación que nos indica si ya hemos leído el último dato o byte de un archivo lógico.
- En C, esto se hace de la siguiente forma:

```
if (feof(fp)) {
  /* llegamos al fin de archivo */
} else {
  /* aun quedan datos por leer */
}
```

Contenido

1. Diseño y especificación de estructuras de archivos
2. Operaciones fundamentales de procesamiento de archivos
   - Archivos físicos y lógicos
   - Apertura de archivos
   - Cerrado de archivos
   - Lectura y escritura de archivos
   - Tipos de acceso y búsqueda
   - Consideraciones especiales
3. Almacenamiento secundario
4. Conceptos fundamentales de estructuras de archivos
5. Manipulando archivos con registros

Acceso secuencial y arbitrario

- Cada vez que hacemos una lectura o escritura avanzamos una o más posiciones en el archivo.
- A este tipo de acceso se le llama secuencial.
- Pero en ocasiones queremos brincar a alguna posición específica del archivo, por ejemplo, porque sabemos que allí está el siguiente dato que necesitamos.
- A este tipo de acceso se le llama arbitrario.

Operación de búsqueda

- A la operación que nos permite brincar de una posición a otra de un archivo se le llama búsqueda (seek en inglés).
- Esta operación es de la forma

  `busqueda(archivo, posición)`

  donde se indica la `posición` del `archivo` lógico a la que se quiere brincar.

Búsqueda de archivos en C

- Abajo `pos` es la posición en el archivo lógico `fp` a la que queremos brincar con respecto al `origen` que puede ser `SEEK_SET` (principio), `SEEK_CUR` (actual) o `SEEK_END` (fin).

```
int origen;
long pos;

fseek(fp, pos, origen);
```

Contenido

1. Diseño y especificación de estructuras de archivos
2. Operaciones fundamentales de procesamiento de archivos
   - Archivos físicos y lógicos
   - Apertura de archivos
   - Cerrado de archivos
   - Lectura y escritura de archivos
   - Tipos de acceso y búsqueda
   - Consideraciones especiales
3. Almacenamiento secundario
4. Conceptos fundamentales de estructuras de archivos
5. Manipulando archivos con registros

Cuidados al usar archivos

- Algunos sistemas cambian unos caracteres por otros sin preguntar.
- Otros agregan información innecesaria.
- La mayoría de los sistemas organizan los archivos en directorios, pero sus estructuras son distintas de sistema a sistema.
- Algunos sistemas tienen tipos especiales de archivos (dispositivos físicos, tuberías, redirecciones, etc.).

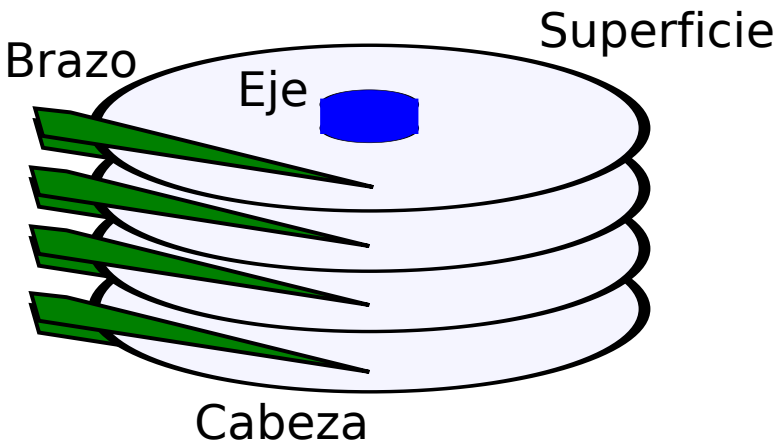
Contenido

1. Diseño y especificación de estructuras de archivos
2. Operaciones fundamentales de procesamiento de archivos
3. Almacenamiento secundario
   - Discos
   - Cintas magnéticas
   - Tipos de almacenamiento
   - Archivos y el sistema operativo
   - Buffers
4. Conceptos fundamentales de estructuras de archivos
5. Manipulando archivos con registros

Hardware de discos

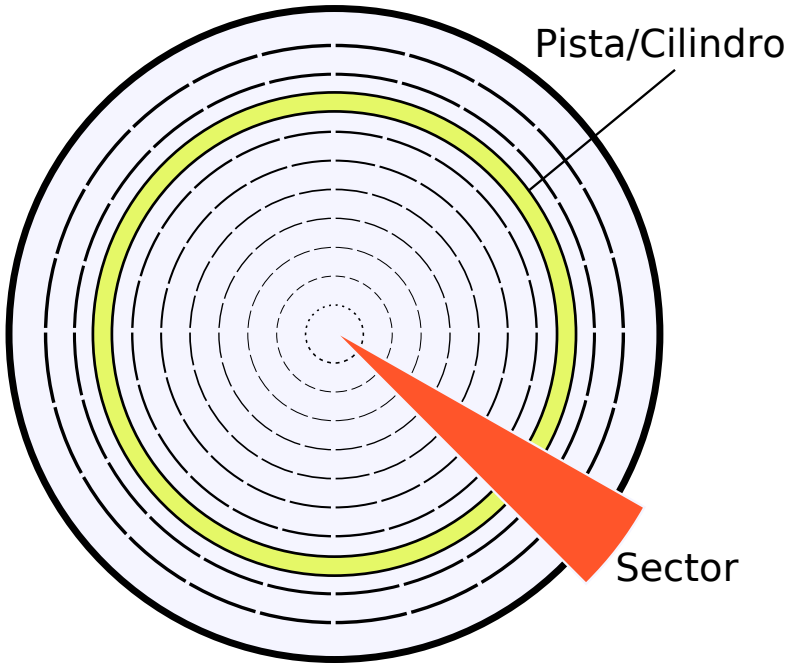
- Los accesos al disco siempre son más lentos que los accesos a la memoria.
- Recuerde que los discos son unas 10^6 veces más lentos que la memoria.
- Pero no todos los accesos al disco son igual de lentos.
- Esto se debe a la forma en la que trabajan los discos.

Vista lateral de un disco



- La información está almacenada en N_{sup} superficies que giran alrededor de un eje.
- Ejemplo: $N_{\text{sup}} = 4$.
- Cada una tiene una cabeza de lectura y escritura que está unida a un brazo.

Vista superior de un disco



- Cada superficie está dividida en N_{pis} pistas concéntricas.
- Ejemplo: $N_{\text{pis}} = 10$.
- Cada pista está dividida en N_{sec} sectores separados por espacios.
- Ejemplo: $N_{\text{sec}} = 24$.

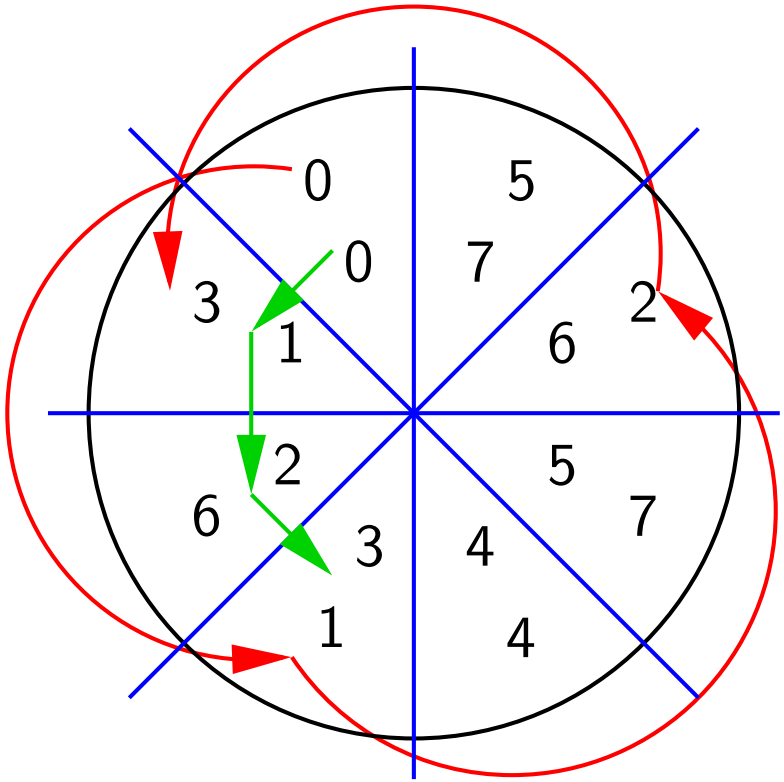
Sectores y cilindros

- Un sector es la unidad más pequeña que se puede direccionar en un disco.
- Al conjunto de pistas que quedan una arriba de otra en las diferentes superficies se le llama cilindro.
- Se pueden leer todos los datos almacenados en un cilindro sin mover las cabezas.
- A este movimiento se le llama búsqueda.

Capacidad de un disco

- Cada uno de los sectores de un disco tiene una cierta capacidad de C_{sec} bytes.
- La capacidad de pistas, superficies y disco pueden calcularse en términos de C_{sec}:
  - $C_{pis} = N_{sec} \times C_{sec}$.
  - $C_{sup} = N_{pis} \times C_{pis}$.
  - $C_{dis} = N_{sup} \times C_{sup}$.
- ¿Cómo se calcula la capacidad de un cilindro?

Compaginación



- Se pueden numerar los sectores de una pista como están colocados de forma física, pero es posible que no se puedan leer en ese orden.
- Por eso a veces se compaginan para no tener que esperar una vuelta completa para leer dos sectores consecutivos.

Grupos y alcance

- A veces los sectores se organizan en grupos con un número fijo de sectores contiguos.
- Así, una vez que se localiza un grupo en el disco se pueden leer todos sus sectores sin realizar ninguna búsqueda.
- Los grupos suelen usarse en las tablas de asignación de archivos (FAT).
- Una colección de grupos adyacentes se llama un alcance.
- En inglés: cluster y extent.

Fragmentación interna

- Todos los sectores de un disco debieran contener el mismo número de bytes, pero a veces eso no es posible.
- Ejemplo: grabar 300 bytes en un sector de 512 bytes.
- A esto se le llama fragmentación interna.
- También puede ocurrir al nivel de grupos.

Pérdida de espacio

- Existen otras fuentes de pérdida de espacio.
- Por ejemplo, la debida a información que necesite estar en el disco pero que no está relacionada con los datos.
- Ejemplo: la información de formato.

Tipos de espera

- El tiempo de búsqueda es el requerido para mover el brazo al cilindro correcto. En promedio es $\approx \frac{1}{3}$ del tiempo de lado a lado.
- La espera rotacional es el tiempo que necesita el disco para girar de modo que el sector deseado se encuentre bajo la cabeza. En promedio es $\approx \frac{1}{2}$ del tiempo de revolución.
- El tiempo de transferencia se puede calcular

$$\frac{\text{bytes transferidos}}{\text{bytes en una pista}} \times \text{tiempo de revolución}$$

Minimizar efectos de espera

- Hay muchas técnicas para tratar de minimizar los efectos de estas esperas:
  1. Cambiar el tamaño de los grupos.
  2. Usar múltiples discos.
  3. Usar paralelismo.
  4. Usar discos en memoria.
  5. Usar caches, buffers, etc.

Contenido

1. Diseño y especificación de estructuras de archivos
2. Operaciones fundamentales de procesamiento de archivos
3. Almacenamiento secundario
   - Discos
   - Cintas magnéticas
   - Tipos de almacenamiento
   - Archivos y el sistema operativo
   - Buffers
4. Conceptos fundamentales de estructuras de archivos
5. Manipulando archivos con registros

Hardware de cintas

- A diferencia de los discos, las cintas no proveen de ninguna facilidad para el acceso directo (o arbitrario) de datos.
- Sin embargo, proveen de un acceso secuencial muy rápido.
- Las cintas son pequeñas y económicas comparadas con los discos de la misma capacidad.

Vista superior de una cinta

- Una cinta se puede pensar como dividida en pistas. Cada pista es una sucesión de bits.
- Un marco son los bits en la misma posición de todas las pistas.

Marcos y paridad

- El último bit del marco no suele formar parte de un dato, sino que se suele usar para verificar su validez.
- Un método común es el de asegurar que la cantidad de unos en un marco siempre tenga la misma paridad.
- Así se habla de paridad par o impar, siendo esta última la más común.

Marcos y espacios

- Los marcos están agrupados en bloques de datos.
- Los bloques están separados por espacios que no contienen datos.
- El propósito de los espacios es el de sincronizar el flujo de datos.

Parámetros de una cinta

- Las cintas se suelen clasificar según tres de sus parámetros.
  - Su densidad d (de 800 a 30000 bit/in).
  - Su velocidad v (de 30 a 200 ft/s).
  - Su tamaño de espacio s (de 0.3 a 0.75 in).
- Además se considera la longitud ℓ de la cinta.
- Observe que todos estos parámetros son constantes.

Relaciones entre los parámetros

- Si la cinta tiene n bloques de datos y cada uno de ellos mide b in, entonces su longitud queda dada por $\ell = n(b + s)$.
- Observe que b y n son variables.
- La elección adecuada de estos parámetros impacta la cantidad de información que se puede almacenar en la cinta.
- ¿Qué pasa si $n = 1$? ¿Y si $b = 0$?

Tasas de transmisión de datos

- La tasa nominal de transmisión de datos es la máxima velocidad a la que se pueden transmitir datos de una cinta y queda dada por $12vd$. ¿De dónde salió el 12?
- La tasa efectiva de transmisión de datos es la velocidad real a la que se transmiten datos de una cinta y queda dada por $12vd\frac{b}{b+s}$.
- En ambos casos se debe multiplicar por el número de bits de datos en un marco.

Contenido

1. Diseño y especificación de estructuras de archivos
2. Operaciones fundamentales de procesamiento de archivos
3. Almacenamiento secundario
   - Discos
   - Cintas magnéticas
   - Tipos de almacenamiento
   - Archivos y el sistema operativo
   - Buffers
4. Conceptos fundamentales de estructuras de archivos
5. Manipulando archivos con registros

Discos y cintas

- Los discos se usan para tener acceso arbitrario a archivos y para almacenar archivos cuando se desea acceso inmediato.
- Las cintas se usan para procesar archivos secuencialmente y para almacenar archivos por periodos de tiempo muy largos.
- Estos roles han cambiado un poco gracias a la disminución del costo de los discos.
- Las cintas siguen siendo, en ocasiones, el método más eficiente de almacenamiento.

Almacenamiento primario

- Tipos: registros, memoria, disco en memoria y memoria caché.
- Medio: semiconductores.
- Tiempo de acceso: 10^{-9} a 10^{-5} s.
- Capacidad: 1 a 10^{9} bytes.
- Costo: 10^{-4} a 10^{-1} pesos/bit.

Almacenamiento secundario

- Tipos: acceso directo y secuencial.
- Medio: cintas y discos magnéticos.
- Tiempo de acceso: 10^{-3} a 10^{2} s.
- Capacidad: 10^{4} a 10^{11} bytes.
- Costo: 10^{-7} a 10^{-2} pesos/byte.

Almacenamiento terciario

- Tipos: archivo y copia de seguridad.
- Medio: cintas y discos ópticos.
- Tiempo de acceso: 1 a 10^2 s.
- Capacidad: 10^4 a 10^{12} bytes.
- Costo: 10^{-7} a 10^{-5} pesos/byte.

Contenido

1. Diseño y especificación de estructuras de archivos
2. Operaciones fundamentales de procesamiento de archivos
3. Almacenamiento secundario
   - Discos
   - Cintas magnéticas
   - Tipos de almacenamiento
   - Archivos y el sistema operativo
   - Buffers
4. Conceptos fundamentales de estructuras de archivos
5. Manipulando archivos con registros

El camino de un byte

- ¿Qué pasa cuando un programa escribe un byte a un archivo en un disco?
- Sabemos que el programa llama a una función de escritura.
- También sabemos que eventualmente el byte quedará escrito de alguna forma en el disco.
- Pero lo que pasa entre la llamada del programa y la escritura en el disco es moderadamente complicado.

El programa hace la llamada

- Supongamos que queremos agregar un byte que representa al caracter `P` almacenado en la variable `ch` en cierto `archivo`.
- Llamamos a `escribe(archivo, ch, 1)`.
- Esto resulta en una solicitud al sistema operativo, el cual se encargará de supervisar que la llamada se efectúe exitosamente.

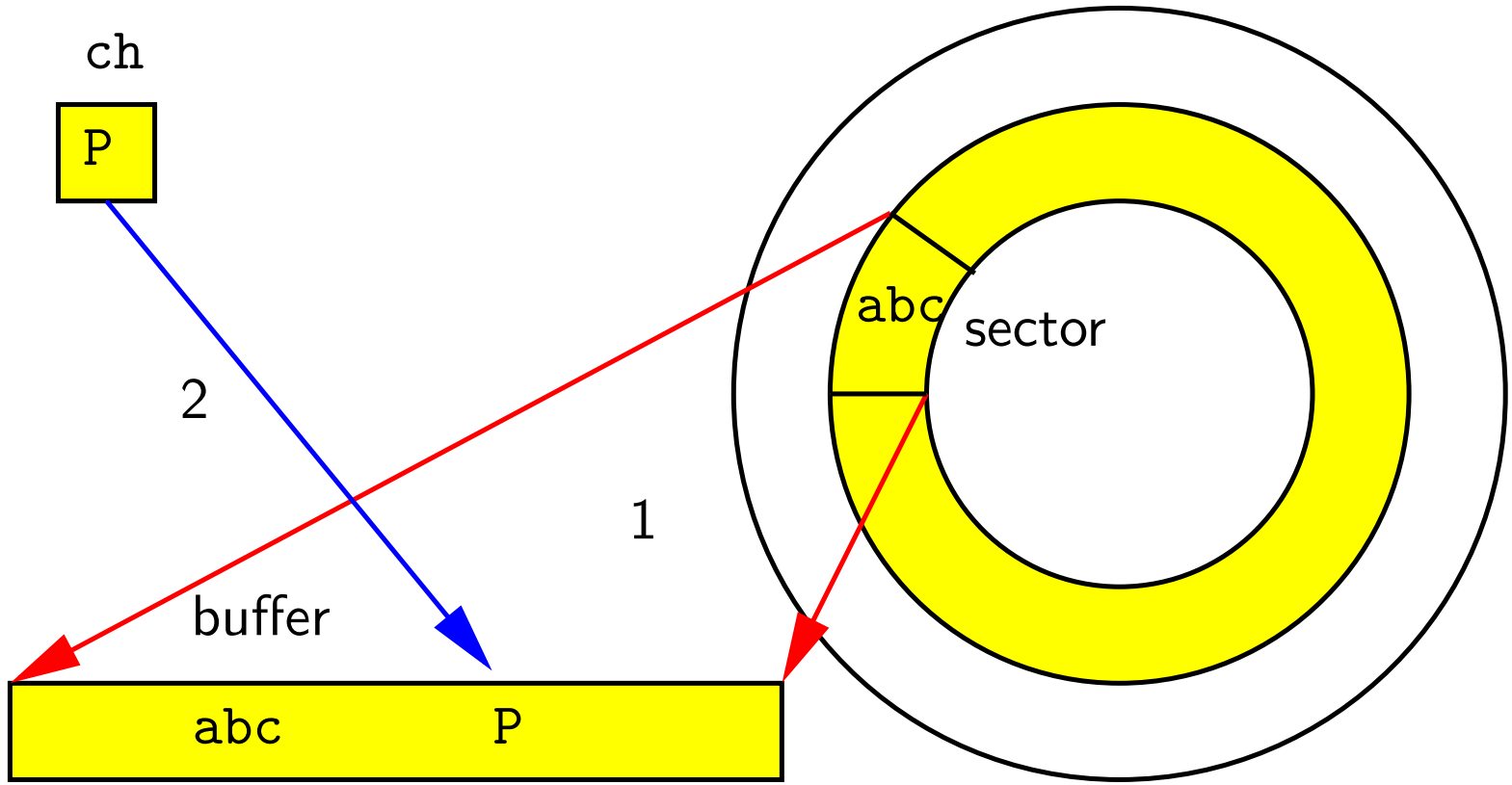
El manejador de archivos

- El sistema operativo, siendo suficientemente complejo, contiene un subsistema llamado manejador de archivos que se encarga de todo lo relacionado a la entrada, salida y dispositivos de almacenamiento.
- El manejador de archivos verifica que la solicitud se pueda llevar a cabo (archivo abierto para escritura, etc.).
- Si todo va bien, lee de la FAT en que disco, cilindro, pista y sector se debe colocar el byte.

El buffer de entrada y salida

- El manejador de archivos debe determinar si ese sector ya está en la memoria o no.
- En el segundo caso, se debe encontrar un buffer de entrada y salida para poder leer el sector en ese espacio.
- El buffer permite que se puedan hacer los accesos de entrada y salida al disco en unidades de un sector o un bloque.
- El buffer se copiará al disco cuando se llene o cuando se cierre el archivo correspondiente.

Actualización del buffer



- Si fuera necesario se lee el sector en un buffer y luego se copia la `P` a la posición correspondiente.

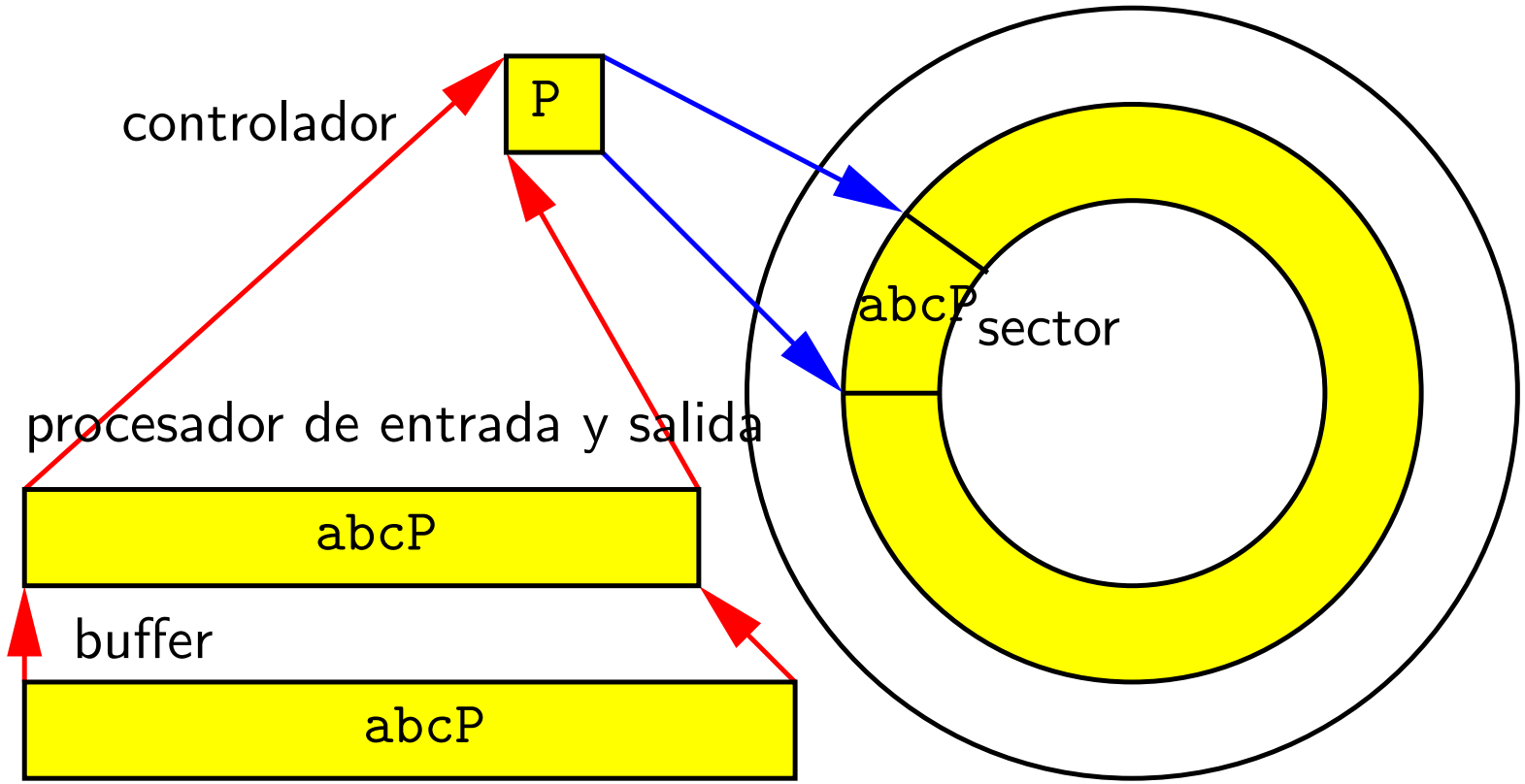
El byte deja la memoria

- Hasta este momento, todo el movimiento del byte ha ocurrido en la memoria principal.
- Ahora el contenido del buffer se pasará a un procesador de entrada y salida junto con la información de en qué parte del disco deberá ser almacenado.
- Este procesador ahora deberá esperar hasta que el controlador del disco le indique que está disponible para escritura.

Y finalmente llega al disco

- Lo que pasa ahora es mecánico y muy lento.
- El disco debe mover su cabeza a la pista pedida (a menos que ya esté allí) y debe esperar a que el sector deseado pase debajo de la cabeza.
- Cuando esto ocurre, se envían al disco uno por uno los bytes del buffer (incluyendo nuestra `P`) y sus bits se escriben uno por uno a la superficie del disco.
- Y la `P` queda allí, girando plácidamente.

Actualización del sector



- El buffer se copia al procesador de entrada y salida, de allí va al controlador del disco y finalmente al sector.

Contenido

1. Diseño y especificación de estructuras de archivos
2. Operaciones fundamentales de procesamiento de archivos
3. Almacenamiento secundario
   - Discos
   - Cintas magnéticas
   - Tipos de almacenamiento
   - Archivos y el sistema operativo
   - Buffers
4. Conceptos fundamentales de estructuras de archivos
5. Manipulando archivos con registros

Manejo de buffers

- La idea de tener un buffer es la de trabajar con grandes cantidades de información antes de hacer una operación de entrada o salida.
- La cantidad y tamaño de los buffers puede afectar bastante el desempeño del programa.
- El caso más sencillo es el de mantener un buffer en memoria para todas las operaciones de entrada y salida.
- ¿Qué pasa si sólo se tiene un buffer y se hacen lecturas y escrituras alternadas?

Dos o más buffers

- Para evitar ese problema se suelen mantener al menos dos buffers cambiando sus roles.
- La idea es que se puedan procesar los datos de un buffer mientras se están transfiriendo datos entre el otro buffer y el disco.
- Por supuesto, se pueden mantener más de dos buffers estableciendo políticas más o menos complicadas para su uso (usado menos recientemente, etc.).
- En cualquier caso, se debe experimentar.

Contenido

1. Diseño y especificación de estructuras de archivos
2. Operaciones fundamentales de procesamiento de archivos
3. Almacenamiento secundario
4. Conceptos fundamentales de estructuras de archivos
   - Organización por campos y registros
5. Manipulando archivos con registros
6. Organización de archivos para mejora del desempeño

Persistencia de datos

- La intención de escribir archivos es la de volver persistentes los datos.
- Los datos deben sobrevivir a la vida del programa que los crea.
- Además debe ser posible que otros programas los puedan usar.
- Esto implica que se debe poder recrear su estructura original.

Campos y registros

- La unidad básica de información es el campo, el cual contiene un solo valor. Los campos se pueden organizar de al menos dos formas.
- Un arreglo está formado por varios campos del mismo tipo.
- Un registro está formado por diferentes tipos de campos.
- Esto corresponde con los conceptos de miembro, vector y estructura almacenados en la memoria principal.

Ejemplos de arreglos

```
cero|uno|dos|tres|cuatro
cero..uno...dos...tres..cuatro
```

Ejemplos de registros

```
File Structures|Folk|1998|Addison Wesley|0-201-87401-6
Algoritmos en C++|Sedgewick|1995|Pearson|968-444-401-X
```

Necesidad de la estructura de archivos

- Imagine que tenemos algunas estructuras con varios miembros.
- ¿Qué pasa si las escribimos a un archivo?
- Si no tenemos cuidado no podremos distinguir entre los varios campos o registros y será imposible recuperar las estructuras originales.
- Es por eso que se necesita organizar los campos en un archivo de una forma más inteligente que la simple concatenación.

Formas de organización de campos

- Existen al menos cuatro formas de organización:
  1. Hacer que los campos tengan longitud predecible.
  2. Comenzar cada campo con un indicador de longitud.
  3. Colocar un separador al final de cada campo.
  4. Utilizar expresiones de la forma `dato=valor` para identificar cada campo y su contenido.

Campos de longitud predecible

- Es muy fácil recuperar la información del archivo.
- Sabemos cuánto mide cada campo y cada registro.
- Pero esto tiene sus inconvenientes:
- Si el espacio destinado a un campo es muy grande se desperdicia.
- Y si es muy corto es posible que algunos datos no quepan.

Ejemplos

```
File Structures  Folk      1998Addison Wesley0-201-87401-6
Algoritmos en C++Sedgewick1995Pearson        968-444-401-X
```

Campos con indicadores de longitud

- Si agregamos la longitud, sigue siendo fácil recuperar la información (siempre y cuando se coloque la cuenta al inicio del campo).
- Incluso se puede saber con cierta precisión cuánto espacio adicional se necesita para incluir esta cuenta.
- Observe que el subcampo de longitud es a su vez de longitud fija.

Ejemplos

```
15 4 41413File StructuresFolk1998Addison Wesley0-201-87401-6
17 9 4 713Algoritmos en C++Sedgewick1995Pearson968-444-401-X
```

Campos con separadores

- Si agregamos separadores, estos se deben poder distinguir fácilmente de los datos.
- Por lo tanto, la selección del separador es importante.
- Por ejemplo, no podemos usar espacios como separadores si los datos son frases de más de una palabra.

Ejemplos

```
File Structures|Folk|1998|Addison Wesley|0-201-87401-6
Algoritmos en C++|Sedgewick|1995|Pearson|968-444-401-X
```

Campos con descriptores

- A diferencia de las anteriores formas, ahora es posible que un campo provea información acerca de sí mismo.
- Esto permite tener campos adicionales a los previstos o campos ausentes.
- Normalmente se usan en conjunto con los separadores.
- Hay que cuidar que el descriptor `dato` no sea muy largo con respecto a `valor`.

Ejemplos

```
TITULO=File Structures
AUTOR=Folk
FECHA=1998
EDITORIAL=Addison Wesley
ISBN=0-201-87401-6
```

```
TITULO=Algoritmos en C++
FECHA=1995
EDITORIAL=Pearson
AUTOR=Sedgewick
ISBN=968-444-401-X
```

Formas de organización de registros

- De la misma manera, se necesitan organizar los registros en un archivo. Existen al menos cinco formas de hacer esto.
  1. Hacer que tengan longitud predecible.
  2. Hacer que los registros tengan un número de campos predecible.
  3. Comenzar con un indicador de longitud.
  4. Colocar un separador al final del registro.
  5. Usar un segundo archivo para almacenar el lugar de inicio de cada registro.

Observaciones I

- La primera forma es muy común. Observe que no es equivalente a que todos los campos tengan longitud fija.
- La segunda forma es muy útil cuando los registros tengan esa propiedad.
- La tercera forma es muy común cuando se trabaja con registros de longitud variable.

Observaciones II

- En la cuarta forma se mantienen las reglas sobre delimitadores y normalmente se usa uno distinto al que separa los campos.
- Al segundo archivo de la quinta forma se le llama índice.
- En general, ningún método es mejor que los demás para una situación cualquiera.

Indicadores de longitud y buffers

- Si queremos colocar un indicador de longitud al principio de cada registro debemos conocer su valor de antemano.
- Es por eso que primero se debe construir el registro para después escribirlo al archivo.
- A esto también se le llama buffer.
- Los buffers también se pueden usar para leer un registro completo a la vez.
- Las operaciones de escritura y lectura se deben poder coordinar entre sí.

Representación de la longitud

- Hay al menos dos formas en las que se puede escribir la longitud: binario o ASCII.
- Sin importar cuál se use, debe ser posible distinguirla del resto del archivo.
- Ejemplo: 65 en ASCII produce `0x36 0x35` mientras que en binario produce `0x41`.
- No todos los sistemas representan los números binarios de la misma forma. La pareja de bytes `0x01 0x02` puede significar 258 en un sistema y 513 en otro.

Contenido

1. Diseño y especificación de estructuras de archivos
2. Operaciones fundamentales de procesamiento de archivos
3. Almacenamiento secundario
4. Conceptos fundamentales de estructuras de archivos
5. Manipulando archivos con registros
   - Acceso a registros
   - Estructura de registros
6. Organización de archivos para mejora del desempeño

Llaves o claves

- Nuestra estructura de archivos se enfoca en el registro como unidad de información.
- Tiene sentido pensar en cómo leer un registro específico sin leer todo el archivo.
- Como primera medida resulta conveniente identificar un registro con una llave o clave.
- En inglés key.

Forma canónica

- Es más fácil pensar en el registro de `Juan` que en el primer registro.
- Como esta clave será proporcionada por un usuario que puede cometer errores, se debe establecer una forma canónica.
- Esto quiere decir que algunas claves que parecen distintas (como `Juan`, `jUaN` y `JUAN`) en realidad deben representar a la misma clave (por ejemplo `juan`).

Clave primaria

- Normalmente se desea que las claves de todos los registros presentes sean distintas.
- Si esto no ocurre podemos tener una confusión acerca de cuál es el registro buscado.
- Lo más fácil es prevenir que el problema aparezca.
- A esta clave única se le llama clave primaria.
- Esto abre la puerta a que haya claves secudarias, terciarias, etc.

Ejemplos

RFC, CURP, ISBN, ISSN, etc.

Selección de la clave

- Se debe tener cuidado en la selección de la clave primaria.
- Lo mejor es que la clave primaria no dependa de los campos del registro.
- ¿Cuántos registros con clave `juan` hay?
- La clave primaria no debe cambiar cuando cambie el registro (recuerde que una persona puede cambiar sus datos personales, pero sigue siendo la misma persona).

Búsqueda secuencial por clave

- Una vez que tenemos claves es muy sencillo escribir un programa que lea el archivo secuencialmente hasta que encuentre los registros con las claves deseadas.
- ¿Qué tan eficiente es esto?
- Recuerde que nuestra principal preocupación es la cantidad de lecturas que se hacen del disco.

Eficiencia de la búsqueda secuencial

- Si tenemos n registros y hacemos una lectura por registro entonces haremos un máximo de n y un promedio de $n/2$ lecturas.
- Por supuesto, si agrupamos los registros en grupos de k entonces haremos un máximo de n/k y un promedio de $n/(2k)$ lecturas.
- Como k es constante, de todas formas el número de lecturas es proporcional a n.

Aplicaciones de la búsqueda secuencial

- Aunque la búsqueda secuencial parece demasiado lenta, tiene sus aplicaciones:
- Búsqueda de patrones.
- Archivos con pocos registros.
- Archivos en los que casi nunca tiene que realizarse una búsqueda.
- Archivos almacenados en cinta.
- Búsquedas en las que se espera obtener muchos resultados.

Acceso directo a registros

- Una alternativa es el acceso directo.
- Se puede tener acceso directo a un registro si podemos buscar (seek) directamente el principio del registro para leerlo inmediatamente.
- De esta manera no importa que tan grande sea el archivo, se puede localizar cualquier registro en un tiempo constante.

Búsqueda de registros de longitud fija

- ¿Cómo sabemos dónde inicia un registro?
- Hay varias formas de resolver este problema.
- La más fácil es cuando todos los registros tienen la misma longitud y sabemos el número de registro relativo que buscamos.
- Si sabemos que cada registro mide r bytes y que el registro buscado tiene número relativo n entonces éste comienza a partir del byte rn.
- Los bytes y registros se numeran desde el 0.

Contenido

1. Diseño y especificación de estructuras de archivos
2. Operaciones fundamentales de procesamiento de archivos
3. Almacenamiento secundario
4. Conceptos fundamentales de estructuras de archivos
5. Manipulando archivos con registros
   - Acceso a registros
   - Estructura de registros
6. Organización de archivos para mejora del desempeño

Longitud de un registro

- Si ya decidimos tener registros de longitud fija es necesario pensar en cómo se puede escoger esa longitud.
- Imagine que un sector tiene 512 bytes y que un registro debe medir al menos 30 bytes.
- Entonces tiene sentido escoger que cada uno de los registros mida 32 bytes para asegurar que cada sector contiene un número entero de registros.

Registro de cabecera

- En ocasiones es necesario o conveniente agregar cierta información adicional a un archivo que sirva para su uso posterior.
- Frecuentemente se usa un registro de cabecera para incluir información como el número de registros, el tamaño de los registros, el momento de su última actualización, el nombre del archivo y la longitud de la cabecera.

Uso del registro de cabecera

- Seguramente, el registro de cabecera tendrá una estructura distinta a la de los demás registros del archivo.
- El propósito del registro de cabecera es que el archivo se autodescriba.
- Esto sirve para que los programas que usen este archivo no necesiten saber con anticipación todos los detalles de la estructura del archivo.

Contenido

1. Diseño y especificación de estructuras de archivos
2. Operaciones fundamentales de procesamiento de archivos
3. Almacenamiento secundario
4. Conceptos fundamentales de estructuras de archivos
5. Manipulando archivos con registros
6. Organización de archivos para mejora del desempeño
   - Compresión de datos
   - Compactación de archivos
   - Búsqueda y ordenamiento internos
   - Ordenamiento por claves

Compresión de archivos

- Hay muchas razones por las cuales es buena idea comprimir archivos:
- Uso de menos espacio.
- Transmisión más rápida.
- Transmisión con menor ancho de banda.
- Procesamiento secuencial más rápido.

Técnicas de compresión de datos

- Existen varias técnicas de compresión de datos:
- Usar una notación distinta (abreviaturas).
- Suprimir información repetida.
- Uso de códigos de longitud variable.
- Eliminación de datos.

Reducción de redundancia

- Se logra codificando los datos en campos de longitud fija y casi siempre resulta en datos binarios (es decir, no en ASCII).
- Tiene varios problemas:
- El archivo ya no podrá ser leído por una persona común y corriente.
- Hay un costo de codificación y decodificación.
- Todos los programas que usen el archivo deberán de conocer la codificación.

¿Cuándo usarla?

- La respuesta depende del contexto.
- Si el archivo ya es muy pequeño, si lo usan muchos programas distintos o si alguno de ellos no puede lidiar con datos en binario entonces es una mala idea.
- Si el archivo es muy grande y sólo lo usa un programa entonces es una buena idea.

Codificación por corridas

- Se usa en archivos que contienen pocos datos y mucho espacio desperdiciado (como las imágenes con pocos detalles).
- Se transforman las secuencias de datos consecutivos idénticos (llamadas corridas) en secuencias especiales.
- Estas consisten de un byte indicador de corrida, el dato que se repite y la cantidad de veces que se repite.

Ejemplo de corridas

- Suponga que se quiere comprimir la cadena

  $$0012121111100000222111100000001000$$

  usando al 2 como indicador de corrida y los dígitos del 1 al 9 para indicar la cuenta.
- Una corrida debe tener longitud al menos cuatro, excepto para el 2. ¿Porqué?
- La cadena comprimida resultante es

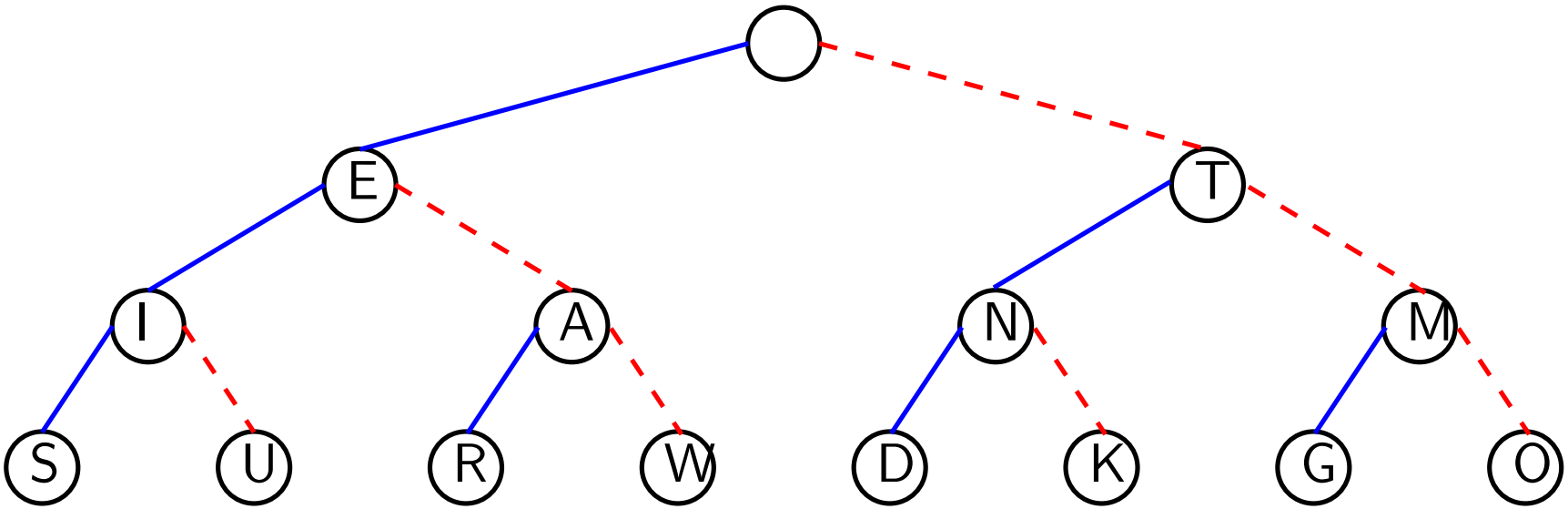
  $$00122112212152052232142081000.$$

- ¿Será cierto que siempre se comprime?

Códigos de longitud variable

- Si un dato aparece muchas más veces en un archivo que otro dato en el mismo archivo tiene sentido asignarle un código más corto al primero que al segundo.
- El código Morse es un ejemplo muy conocido de este tipo de código.
- Otro ejemplo (donde la tabla de codificación puede cambiar de archivo a archivo) es la codificación de Huffman.

Código Morse



- Árbol binario de decisión que contiene parte del código Morse
- La palabra `MORSE` se codifica como `--` `---` `.-.` `...` `.` en Morse.

Código de Huffman

- A diferencia del código Morse, este tipo de códigos no necesita separadores para saber donde termina una letra.
- Se obtienen a partir de las probabilidades de aparición de las letras en un archivo.
- Ejemplo: si las letras `a, b, c, d, e, f` aparecen con probabilidades 0.1, 0.23, 0.45, 0.06, 0.07, 0.09 entonces sus códigos son `0000, 01, 1, 0011, 0010, 0001`.
- ¿Qué dice en `100101110101100101010`?

Compresión con pérdida

- A veces no es necesario preservar la información hasta el último detalle.
- Archivos de sonido, imagen o video.
- Se pueden lograr grandes ganancias descartando la información que resulte irrelevante.
- Ejemplos: `mp3`, `jpeg` y `mpeg`.

Contenido

1. Diseño y especificación de estructuras de archivos
2. Operaciones fundamentales de procesamiento de archivos
3. Almacenamiento secundario
4. Conceptos fundamentales de estructuras de archivos
5. Manipulando archivos con registros
6. Organización de archivos para mejora del desempeño
   - Compresión de datos
   - Compactación de archivos
   - Búsqueda y ordenamiento internos
   - Ordenamiento por claves

Recuperando espacio en archivos

- Si el único tipo de operación que se realiza en un archivo es la de agregar registros entonces nunca se desperdiciará espacio.
- La situación cambia si se permiten borrados o modificaciones de registros.
- Una modificación se puede ver como un borrado seguido de una adición, por lo que no la estudiaremos a detalle.

Compactación de archivos

- Un primer mecanismo para recuperar el espacio perdido por operaciones de borrado se llama compactación.
- Éste busca a través del archivo aquel espacio en el que no haya datos y lo recupera.
- Para ello se necesita una forma de saber si cierto espacio contiene o no datos.
- Una forma de hacer esto es colocando un indicador o campo de borrado en el registro.

Programas y registros borrados

- La compactación no se hace con frecuencia.
- Los programas que usen estos archivos deben ignorar los registros borrados.
- Una ventaja es que se puede recuperar un registro borrado siempre y cuando no se haya realizado una compactación.
- Si hay espacio disponible, la forma más fácil de hacer la compactación es usando un segundo archivo (aunque no es necesario).

Problemas de la compactación

- La compactación tiene varios problemas.
- Si el archivo se modifica con gran frecuencia entonces la compactación también deberá ocurrir frecuentemente.
- Si el programa que usa el archivo es interactivo entonces el usuario tendrá que esperar una cantidad considerable de tiempo para que se efectúe esta operación.

Registros de longitud fija

- Suponga que los registros son de longitud fija. Entonces cualquier registro cabe en el espacio liberado por uno borrado.
- Por lo tanto, si sabemos dónde hay un espacio libre lo podemos reutilizar.
- Una forma simple es la de buscar secuencialmente en el archivo hasta que encontremos un registro borrado.
- Si el archivo es muy grande este proceso es demasiado lento.

Lista de registros disponibles

- Se necesita un método para decidir inmediatamente si hay o no espacio reutilizable y, en su caso, saber dónde está.
- Una forma de lograr esto es creando una lista ligada de registros disponibles.
- Como el orden en el que se encuentren estos registros es indistinto conviene utilizar el tipo de lista ligada más sencillo de implementar.
- ¿Cuál es?

Pila de registros disponibles

- Se necesita una forma de indicar dónde está el tope de la pila y cuándo no hay más elementos en la pila.
- No podemos usar apuntadores ¿porqué?
- Pero podemos usar los números de registro relativos para indicar dónde está el siguiente elemento libre y el -1 para indicar que no hay más elementos.
- Este indicador se puede guardar en un campo del registro (y el tope en la cabecera).

Registros de longitud variable

- Si los registros son de longitud variable:
- Ya no podemos usar el número de registro relativo, así que debemos usar en su lugar la posición en bytes del registro en el archivo.
- Además, necesitamos saber de qué longitud es cada registro (aunque esa información podría ya existir en el registro).
- Tampoco podemos usar cualquier registro libre sino sólo los que sean suficientemente grandes: la pila ya no nos será útil.

Fragmentación externa

- Un problema nuevo es ¿qué hacer con el espacio que sobra al usar un registro disponible más grande de lo necesario?
- ¡La idea de usar registros de longitud variable era la de desaparecer este problema!
- Una opción es colocar ese espacio extra de nuevo en la lista de espacio disponible.
- Puede ocurrir que ese espacio sea tan pequeño que no sea utilizable.
- Este es un caso de fragmentación externa.

Políticas de reuso de registros

- ¿Cómo decidimos cuál espacio libre utilizar para grabar un registro?
- Esto ya no es ciencia, sino más bien arte.
- Algunas estrategias comunes son:
  1. En el primero que quepa (first fit).
  2. En el de menor tamaño (best fit).
  3. En el de mayor tamaño (worst fit).
- Ninguna de ellas es mejor que las demás para una situación arbitraria.

Contenido

1. Diseño y especificación de estructuras de archivos
2. Operaciones fundamentales de procesamiento de archivos
3. Almacenamiento secundario
4. Conceptos fundamentales de estructuras de archivos
5. Manipulando archivos con registros
6. Organización de archivos para mejora del desempeño
   - Compresión de datos
   - Compactación de archivos
   - Búsqueda y ordenamiento internos
   - Ordenamiento por claves

Búsqueda y ordenamiento

- Suponga que un archivo tiene n registros.
- La búsqueda secuencial tarda $\propto n$.
- El ordenamiento secuencial tarda $\propto n^2$.
- La búsqueda binaria tarda $\propto \log n$.
- El ordenamiento eficiente tarda $\propto n \log n$.
- Todos son muy lentos con archivos.
- Estas operaciones debieran hacerse solamente en la memoria.

Limitaciones

- Los métodos eficientes de búsqueda y ordenamiento tienen serias limitaciones.
- La búsqueda binaria requiere de más de dos accesos al disco.
- Mantener un archivo ordenado es muy caro.
- El ordenamiento en memoria sólo sirve para archivos pequeños.

Condiciones para mejores métodos

- Se debe cumplir al menos una de las siguientes:
  1. No debe ser necesario reorganizar los registros en un archivo cada vez que se agregue un nuevo registro (índices y técnica de dispersión).
  2. Debe estar asociado con una estructura que nos permita una mejora sustancial en la velocidad de reorganización de un archivo (árboles B, B^+, B^*, etc).
- Estudiaremos estos métodos más adelante.

Contenido

1. Diseño y especificación de estructuras de archivos
2. Operaciones fundamentales de procesamiento de archivos
3. Almacenamiento secundario
4. Conceptos fundamentales de estructuras de archivos
5. Manipulando archivos con registros
6. Organización de archivos para mejora del desempeño
   - Compresión de datos
   - Compactación de archivos
   - Búsqueda y ordenamiento internos
   - Ordenamiento por claves

Ordenamiento por claves

- Observe que no se necesita tener todo el registro para poder ordenarlo.
- Basta tener las llaves o claves.
- Un primer método de mejora es:
  1. Leer las llaves junto con los números relativos de cada registro.
  2. Ordenar las llaves en memoria.
  3. Reescribir los registros en el nuevo orden.

Primer método

- Archivo original

pez	ola	uno	non	voz	res	ana	sol

- Copia en la memoria

pez	ola	uno	non	voz	res	ana	sol
0	1	2	3	4	5	6	7

- Ordenado en memoria

ana	non	ola	pez	res	sol	uno	voz
6	3	1	0	5	7	2	4

- Archivo final

ana	non	ola	pez	res	sol	uno	voz

Problemas de éste método

- Al principio parece que el método funciona.
- Podemos ordenar archivos mucho más grandes en la misma cantidad de memoria.
- Pero los registros se deben leer dos veces.
- La primera vez es en orden secuencial.
- Pero la segunda vez es en orden arbitrario.
- Así que no funciona como esperábamos.
- ¿Cómo resolver este problema?

Solución para éste método

- No reescribamos el archivo, sino que generemos un segundo archivo índice.
- Éste archivo contendrá la estructura generada en memoria. Cada uno de sus registros contendrá dos campos: la clave y el número relativo del registro original.
- Esto se puede hacer con una lectura secuencial del archivo original seguida de una escritura secuencial del archivo índice.
- Esto sí es más rápido que la solución original.

Archivo e índice

- Archivo original

0	1	2	3	4	5	6	7
pez	ola	uno	non	voz	res	ana	sol

- Archivo índice

0	1	2	3	4	5	6	7
ana	non	ola	pez	res	sol	uno	voz
6	3	1	0	5	7	2	4

Efectos secundarios

- Observe que con este método los registros no se mueven de su lugar.
- ¿Qué pasaría si en algún registro del archivo hubiera un indicador del número relativo de otro registro?
- Al mover los registros el indicador sería inútil.
- Cuando un archivo contiene este tipo de referencias se dice que los registros están fijos (pinned) y si éstas se pierden se dice que quedan colgando (dangling).

Part II

Grafos y sus aplicaciones

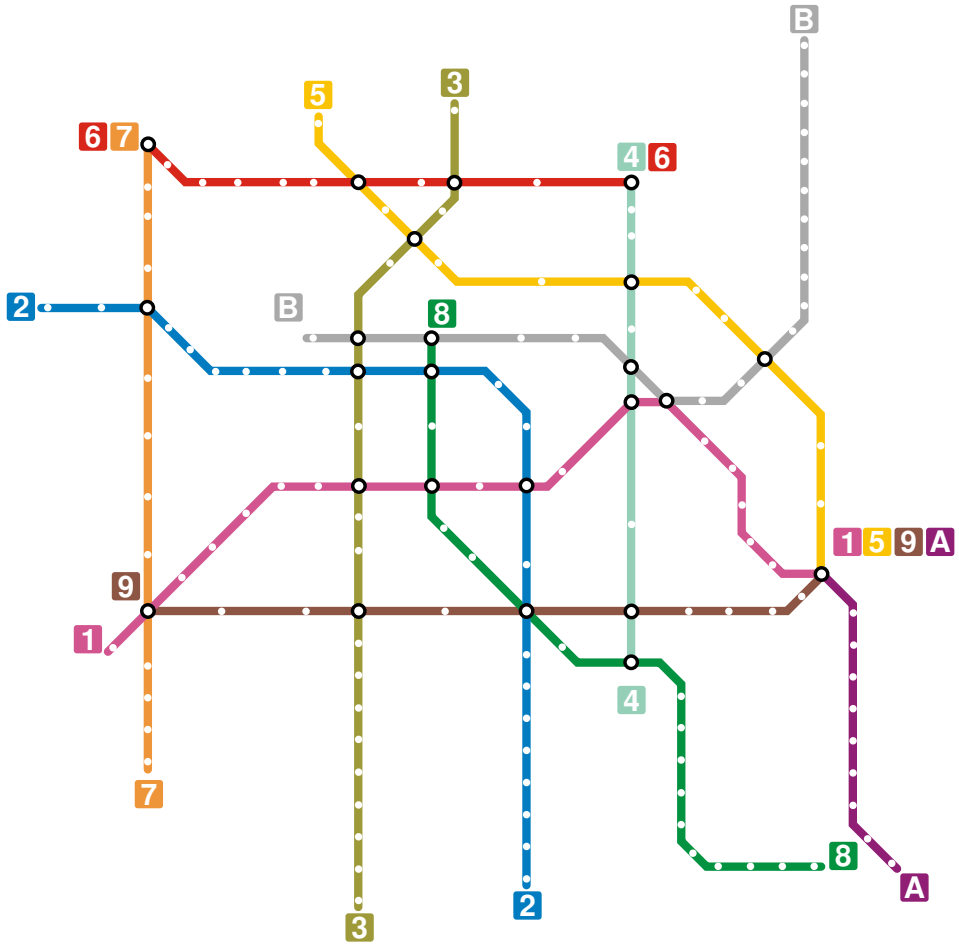
Contenido

7. Representación de grafos y aplicaciones
   - Grafos (no dirigidos)
   - Grafos dirigidos
8. Recorridos de un grafo
9. Grafos con costos

Objetos y conexiones

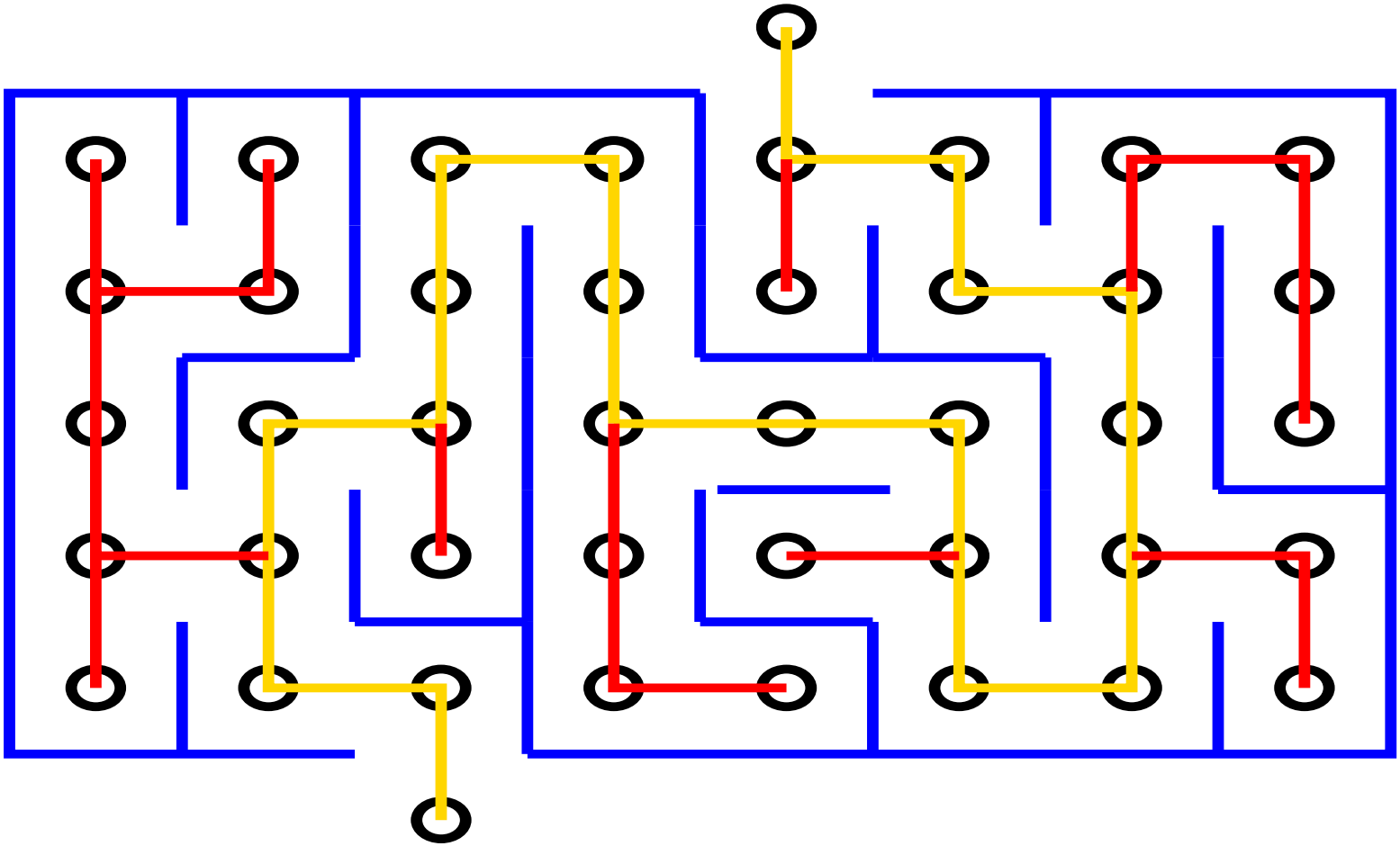
- Muchos problemas cotidianos se formulan de manera natural por medio de objetos y las conexiones que haya entre ellos.
- Mapas y caminos.
- Laberintos.
- Circuitos eléctricos.
- Redes de computadoras.
- Redes sociales.

Mapas y caminos



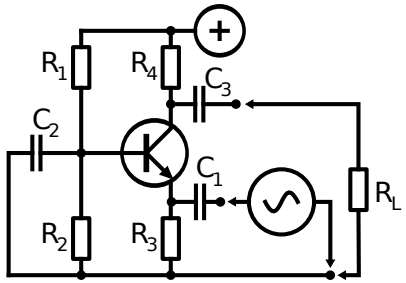
La red del Metro

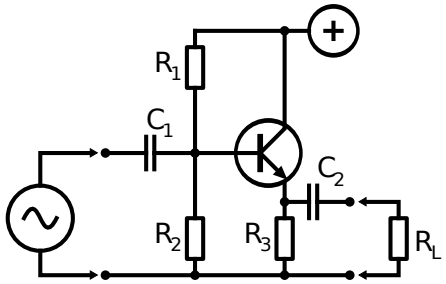
Laberintos

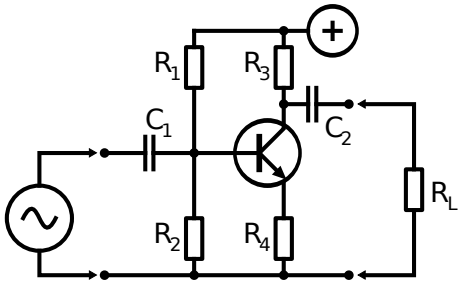


Un laberinto resuelto

Circuitos eléctricos

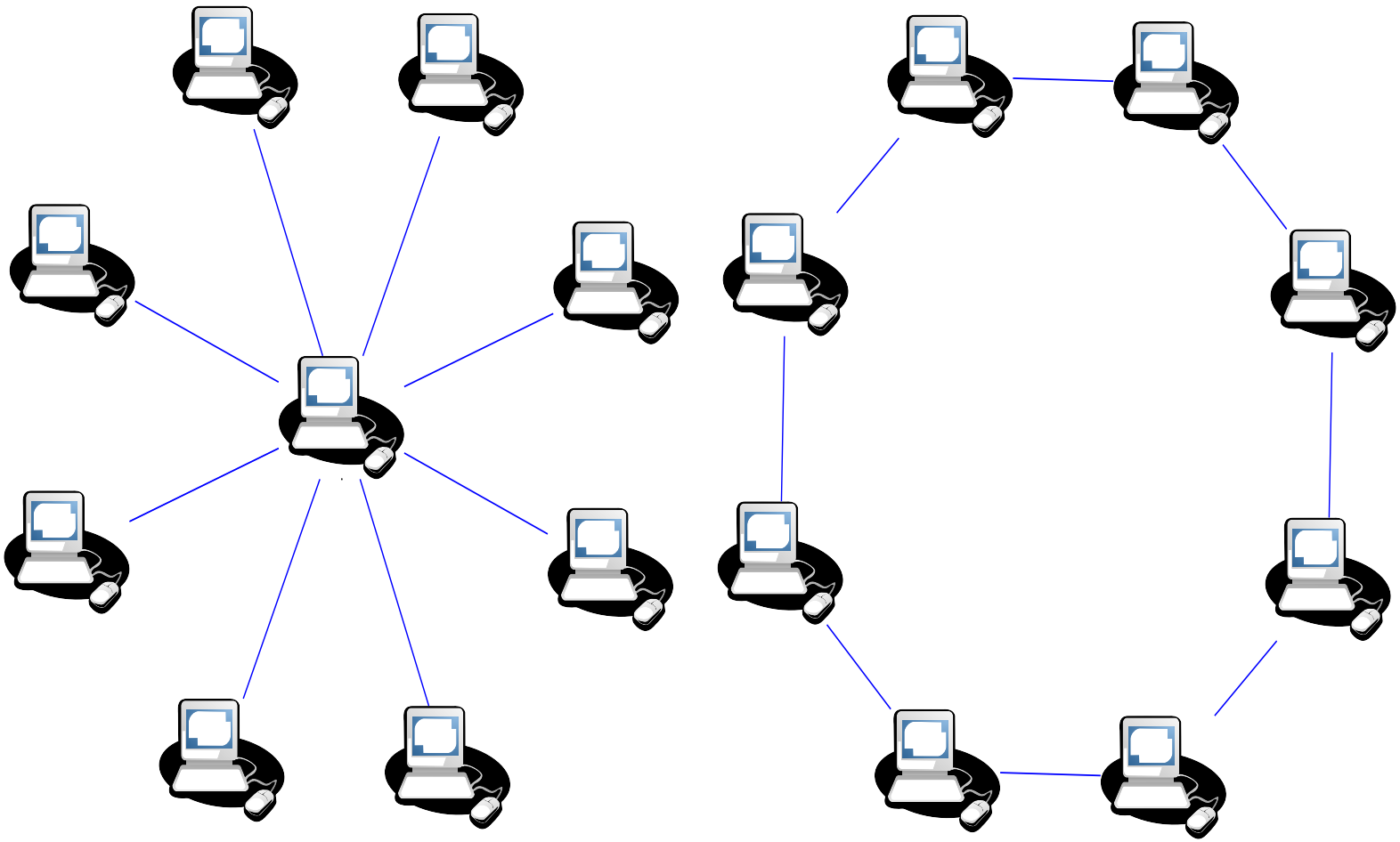






Tres amplificadores con transistores.

Redes de computadoras



Redes tipo estrella y anillo

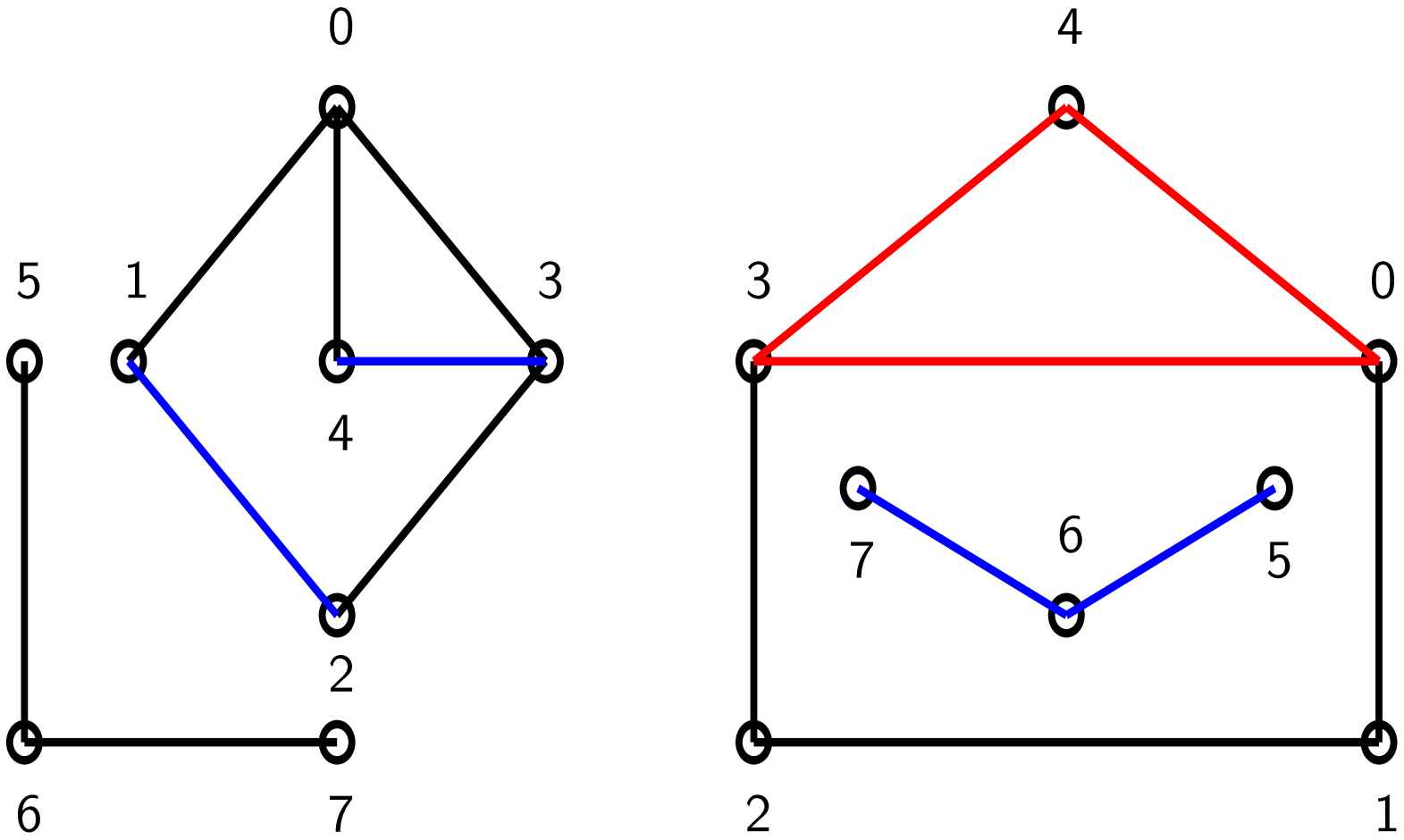
Grafos, vértices y aristas

- Un grafo $G = (V, E)$ es una pareja ordenada de conjuntos de vértices y aristas.
- Todos los vértices y aristas de G deben ser distintos (es decir, tener nombres distintos).
- Cada arista une a dos vértices.
- A veces una arista une a un vértice consigo mismo (lazo) o más de una arista une a los mismos vértices (paralelas).
- Y a veces las aristas tienen costos.

Dibujos de grafos

- Generalmente se representa a un grafo con un dibujo donde cada vértice es un punto y cada arista es una línea que une dos puntos.
- La ubicación de los puntos y la forma de las líneas suele no importar, pues solamente representan relaciones.
- Pero a veces un grafo representa a un objeto geométrico y en ese caso ambas cosas resultan ser importantes.

Dos dibujos de un grafo



Un bosque, un camino y un ciclo.

Caminos y conexidad

- Un camino es una secuencia de vértices en la que cada dos vértices consecutivos están unidos por una arista.
- Un grafo es conexo si hay caminos entre cada pareja de vértices.
- Si un grafo no es conexo entonces tiene más de una componente conexa.
- Un camino que no repite vértices se llama camino simple.

Ciclos y árboles

- Un camino que comienza y termina en el mismo vértice es un circuito.
- Un circuito que no repite vértices es un ciclo.
- Un grafo que no tiene ciclos se llama bosque.
- Un bosque conexo se llama árbol.
- Un árbol que pasa por todos los vértices de un grafo se llama abarcador.

Grafos y el número de aristas

- Sean $n = |V|$ y $m = |E|$.
- Un grafo (sin aristas paralelas ni lazos) que tiene todas las aristas posibles (es decir, si $m = \frac{1}{2}n(n-1)$) se llama completo.
- Un grafo que tiene relativamente pocas aristas (es decir, si $m \propto n$) se llama disperso.
- Un grafo que tiene relativamente muchas aristas (es decir, si $m \propto n^2$) se llama denso.

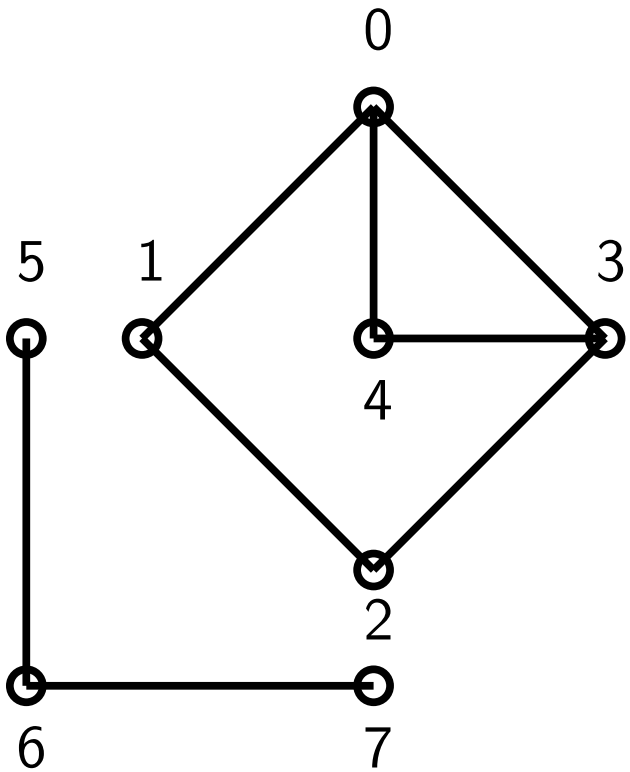
Teorema

Un grafo es un árbol si y sólo si es conexo y tiene $m = n - 1$.

Representaciones de un grafo

- Hay al menos tres formas distintas de representar un grafo en un programa.
- La matriz de adyacencia se utiliza para representar grafos densos.
- La lista de adyacencia se utiliza para representar grafos dispersos.
- En ocasiones se usa la matriz de incidencia.
- Las tres se pueden usar para representar los costos de las aristas.

Matriz de adyacencia



A	0	1	2	3	4	5	6	7
0	0	1	0	1	1	0	0	0
1	1	0	1	0	0	0	0	0
2	0	1	0	1	0	0	0	0
3	1	0	1	0	1	0	0	0
4	1	0	0	1	0	0	0	0
5	0	0	0	0	0	0	1	0
6	0	0	0	0	0	1	0	1
7	0	0	0	0	0	0	1	0

Lectura de una matriz cuadrada

```
int n;
int a[MAXV][MAXV];
int i, j;

scanf("%d", &n);
for(i = 0; i < n; i++)
 for(j = 0; j < n; j++)
  scanf("%d", &a[i][j]);
```

```
8
0 1 0 1 1 0 0 0
1 0 1 0 0 0 0 0
0 1 0 1 0 0 0 0
1 0 1 0 1 0 0 0
1 0 0 1 0 0 0 0
0 0 0 0 0 0 1 0
0 0 0 0 0 1 0 1
0 0 0 0 0 0 1 0
```

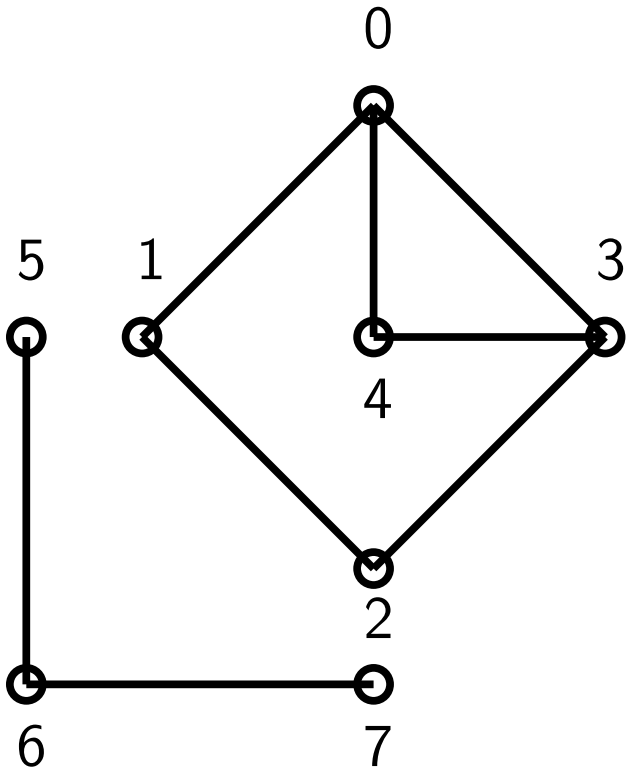
Lectura de una matriz triangular

```
int n, i, j;
int a[MAXV][MAXV];

scanf("%d", &n);
for(i = 0; i < n; i++)
 for(j = 0; j <= i; j++)
  if (i == j)
   a[i][j] = 0;
  else {
   scanf("%d", &a[i][j]);
   a[j][i] = a[i][j];
  }
```

```
8
1
0 1
1 0 1
1 0 0 1
0 0 0 0 0
0 0 0 0 0 1
0 0 0 0 0 0 1
```

Listas de adyacencia



- $0 : 1 \longrightarrow 3 \longrightarrow 4.$
- $1 : 0 \longrightarrow 2.$
- $2 : 1 \longrightarrow 3.$
- $3 : 0 \longrightarrow 2 \longrightarrow 4.$
- $4 : 0 \longrightarrow 3.$
- $5 : 6.$
- $6 : 5 \longrightarrow 7.$
- $7 : 6.$

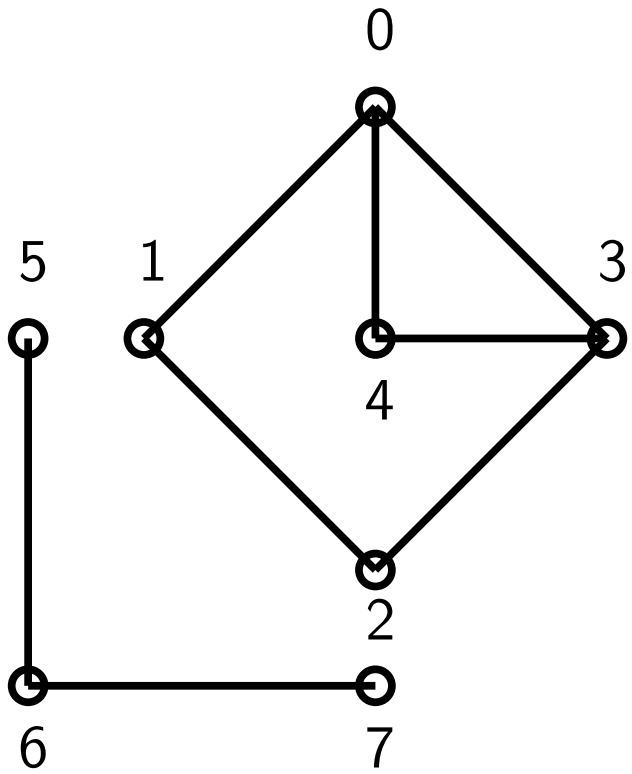
Lectura de una lista

```
int n, m, u, v, i, j;
lista a[MAXV];

scanf("%d%d", &n, &m);
for (i = 0; i < n; i++)
 inicializa(&a[i]);
for (j = 0; j < m; j++) {
 scanf("%d%d", &u, &v);
 inserta(u, &a[v]);
 inserta(v, &a[u]);
}
```

```
8 8
0 1
1 2
2 3
3 4
4 0
0 3
5 6
6 7
```

Matriz de incidencia



I	01	12	23	34	40	03	56	67
0	1	0	0	0	1	1	0	0
1	1	1	0	0	0	0	0	0
2	0	1	1	0	0	0	0	0
3	0	0	1	1	0	1	0	0
4	0	0	0	1	1	0	0	0
5	0	0	0	0	0	0	1	0
6	0	0	0	0	0	0	1	1
7	0	0	0	0	0	0	0	1

Lectura de una matriz rectangular

```
int n, m, u, v, i, j;
int a[MAXV][MAXE];

scanf("%d%d", &n, &m);
for (j = 0; j < m; j++) {
 scanf("%d%d", &u, &v);
 for (i = 0; i < n; i++) {
  if (i == u || i == v)
   a[i][j] = 1;
  else
   a[i][j] = 0;
  }
}
```

```
8 8
0 1
1 2
2 3
3 4
4 0
0 3
5 6
6 7
```

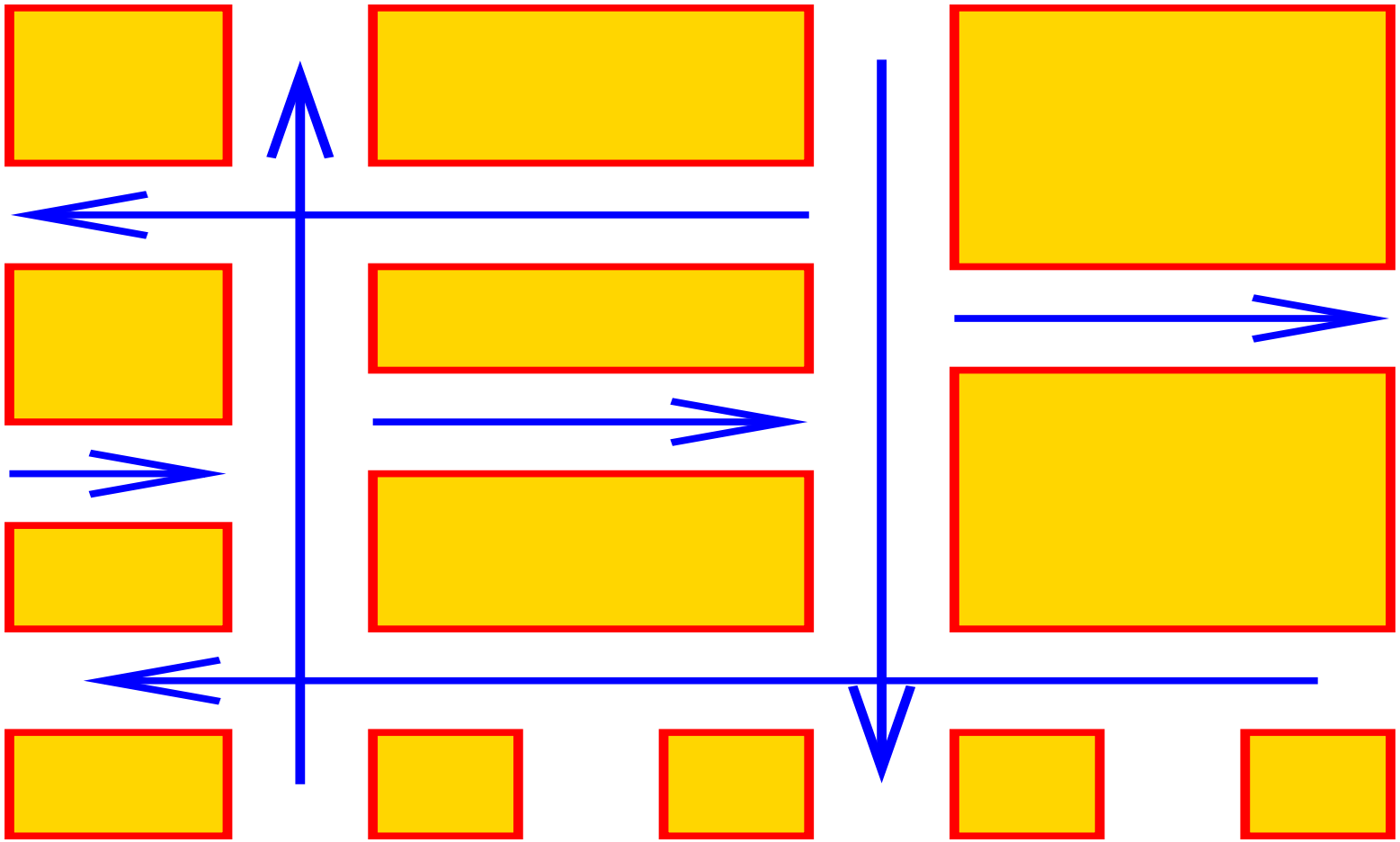
Contenido

7. Representación de grafos y aplicaciones
   - Grafos (no dirigidos)
   - Grafos dirigidos
8. Recorridos de un grafo
9. Grafos con costos

Objetos y precedencias

- Otros problemas cotidianos se formulan por medio de objetos y las relaciones de precedencia que haya entre ellos.
- Mapas con calles de un sentido.
- Secuenciación de tareas.
- Redes sociales jerárquicas.
- Flujos o tráfico de objetos.
- Relaciones asimétricas.

Calles de un sentido



Mapa del centro de la ciudad

Secuenciación de tareas

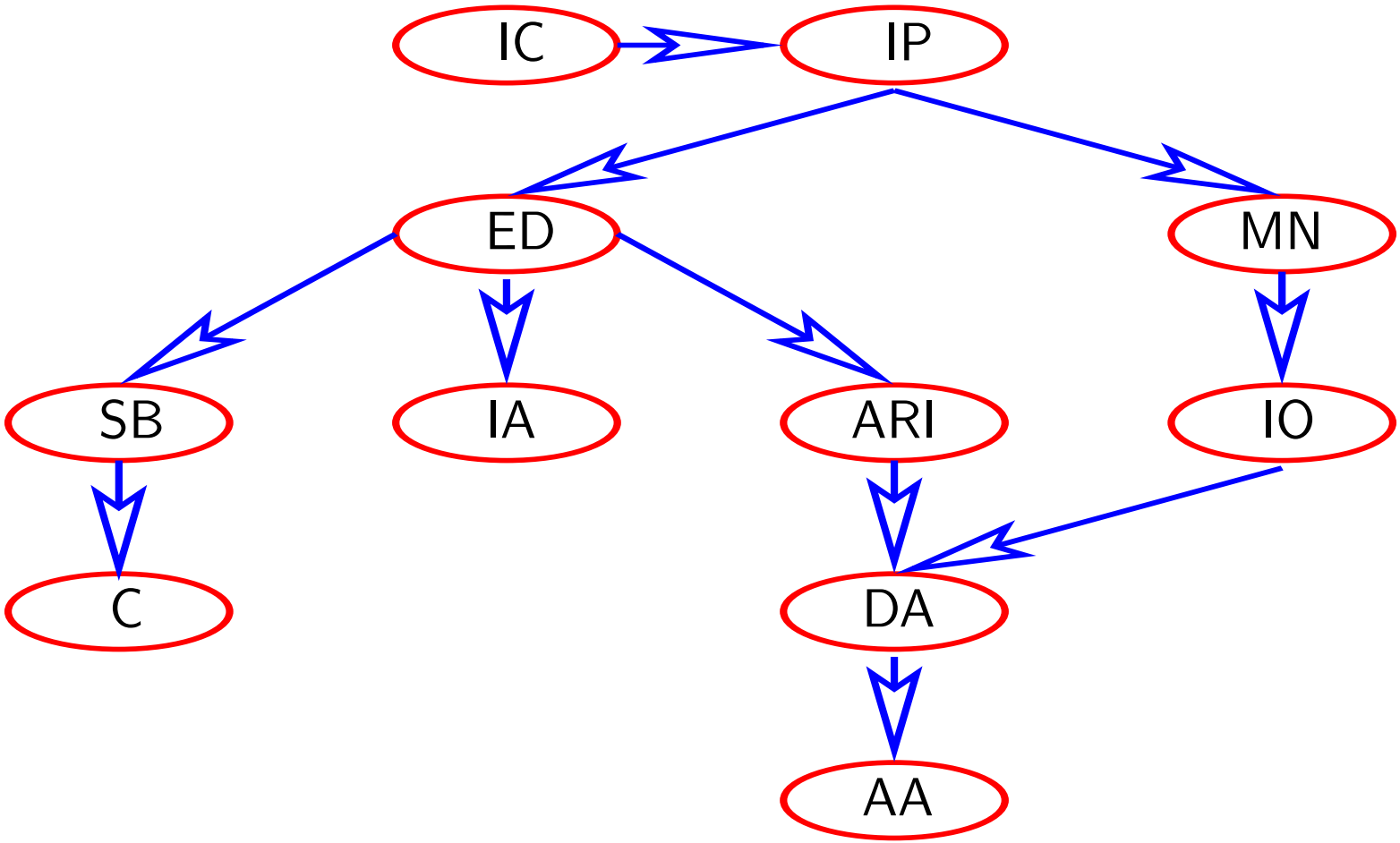
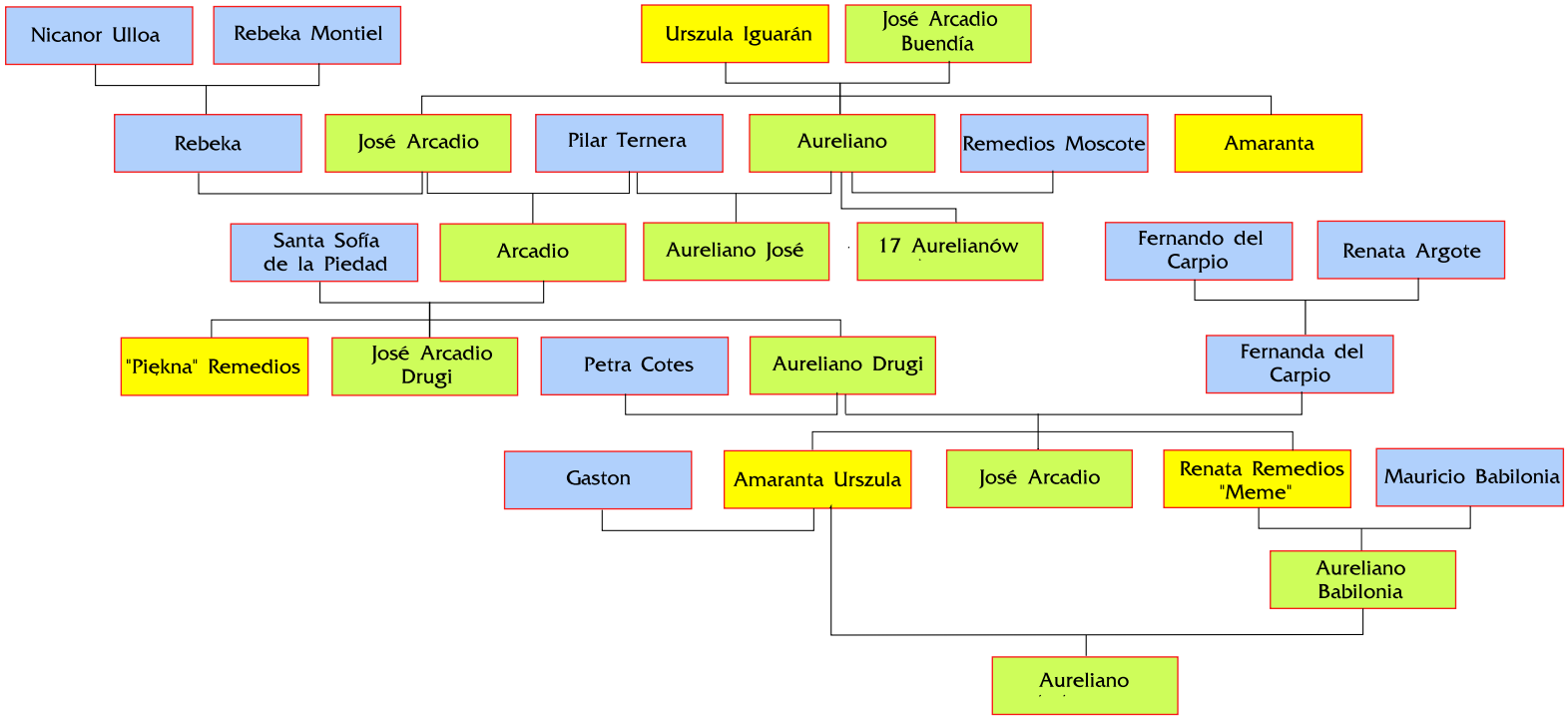


Diagrama de seriación de UEA

Redes sociales jerárquicas

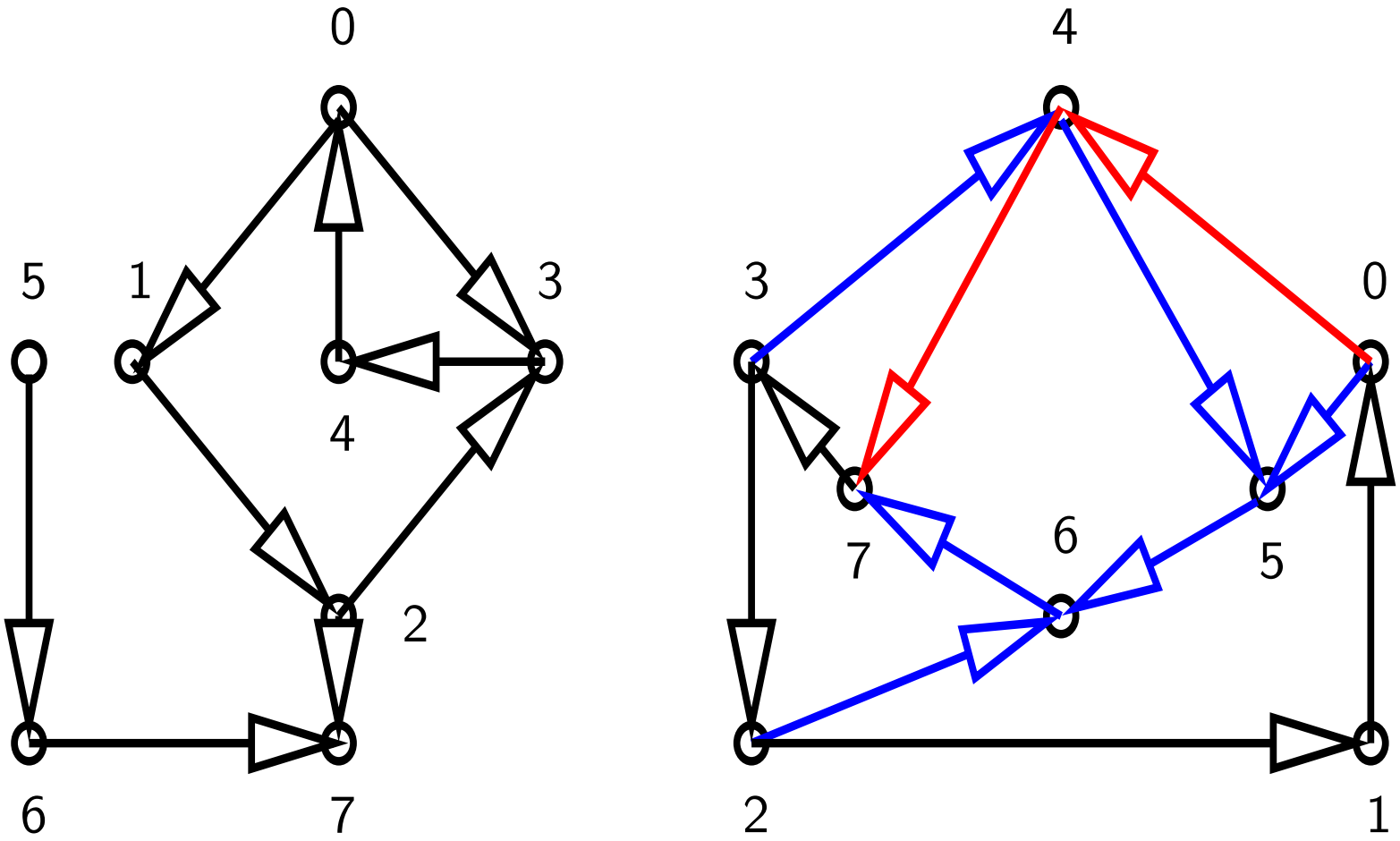


Árbol genealógico de los Buendía

Grafos dirigidos y arcos

- Un grafo dirigido $D = (V, A)$ es una pareja ordenada de conjuntos de vértices y arcos.
- Todos los vértices y arcos son distintos.
- Cada arco va de un vértice a otro.
- A veces un arco va de un vértice a sí mismo (lazo) o más de un arco va de un mismo vértice a otro (arcos paralelos).
- Y a veces los arcos tienen capacidades.

Dos grafos dirigidos



Uno fuertemente conexo y el otro no.

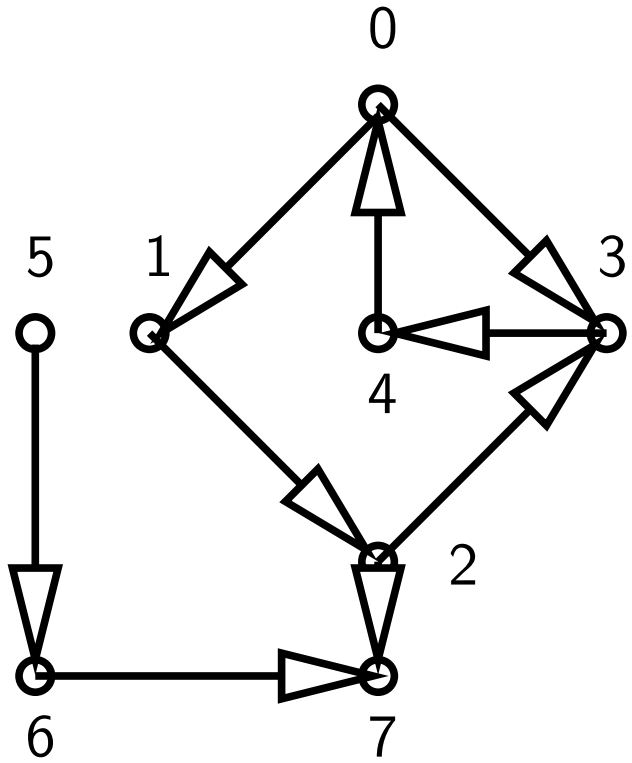
Caminos dirigidos

- Un camino dirigido es una secuencia de vértices en la que hay un arco de cada vértice al siguiente.
- Un grafo dirigido es fuertemente conexo si hay caminos entre cada pareja de vértices.
- Si un grafo dirigido no es fuertemente conexo entonces tiene más de una componente fuertemente conexa.

Representación de grafos dirigidos

- Hay al menos tres formas de representar un grafo dirigido en un programa.
- La matriz de adyacencia se usa para representar grafos dirigidos densos.
- La lista de adyacencia se usa para representar grafos dirigidos dispersos.
- En ocasiones se usa la matriz de incidencia.
- Las tres se pueden usar para representar las capacidades de los arcos.

Matriz de adyacencia



A	0	1	2	3	4	5	6	7
0	0	1	0	1	0	0	0	0
1	0	0	1	0	0	0	0	0
2	0	0	0	1	0	0	0	1
3	0	0	0	0	1	0	0	0
4	1	0	0	0	0	0	0	0
5	0	0	0	0	0	0	1	0
6	0	0	0	0	0	0	0	1
7	0	0	0	0	0	0	0	0

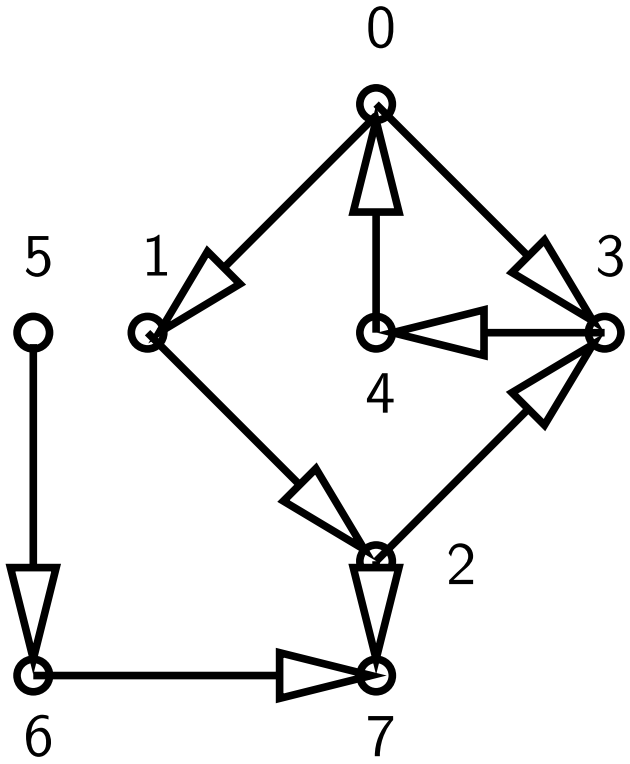
Lectura de una matriz cuadrada

```
int n;
int a[MAXV][MAXV];
int i, j

scanf("%d", &n);
for (i = 0; i < n; i++)
 for (j = 0; j < n; j++)
  scanf("%d", &a[i][j]);
```

```
8
0 1 0 1 0 0 0 0
0 0 1 0 0 0 0 0
0 0 0 1 0 0 0 1
0 0 0 0 1 0 0 0
1 0 0 0 0 0 0 0
0 0 0 0 0 0 1 0
0 0 0 0 0 0 0 1
0 0 0 0 0 0 0 0
```

Lista de adyacencia



- $0 : 1 \longrightarrow 3.$
- $1 : 2.$
- $2 : 3 \longrightarrow 7.$
- $3 : 4.$
- $4 : 0.$
- $5 : 6.$
- $6 : 7.$
- $7 : \text{nil}.$

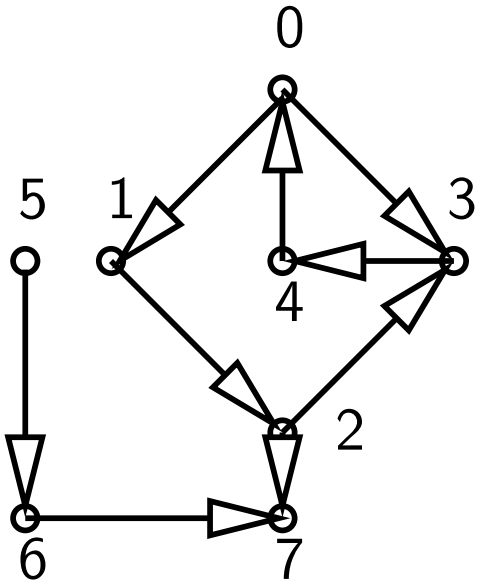
Lectura de una lista

```
int n, m, u, v, i, j;
lista a[MAXV];

scanf("%d%d", &n, &m);
for (i = 0; i < n; i++)
 inicializa(&a[i]);
for (j = 0; j < m; j++) {
 scanf("%d%d", &u, &v);
 inserta(u, &a[v]);
}
```

```
8 9
0 1
1 2
2 3
3 4
4 0
0 3
5 6
6 7
2 7
```

Matriz de incidencia



I	01	12	23	34	40	03	56	67	27
0	-1	0	0	0	+1	-1	0	0	0
1	+1	-1	0	0	0	0	0	0	0
2	0	+1	-1	0	0	0	0	0	-1
3	0	0	+1	-1	0	+1	0	0	0
4	0	0	0	+1	-1	0	0	0	0
5	0	0	0	0	0	0	-1	0	0
6	0	0	0	0	0	0	+1	-1	0
7	0	0	0	0	0	0	0	+1	+1

Lectura de una matriz rectangular

```
int n, m, u, v, i, j;
int a[MAXV][MAXA];

scanf("%d%d", &n, &m);
for (j = 0; j < m; j++) {
 scanf("%d%d", &u, &v);
 for (i = 0; i < n; i++)
  a[i][j] = 0;
  a[u][j] = -1;
  a[v][j] = +1;
}
```

```
8 9
0 1
1 2
2 3
3 4
4 0
0 3
5 6
6 7
2 7
```

Contenido

7 Representación de grafos y aplicaciones

8 Recorridos de un grafo
- Búsqueda en profundidad
- Búsqueda en amplitud
- Componentes conexas y biconexas
- Unión y pertenencia
- Ordenamiento topológico

9 Grafos con costos

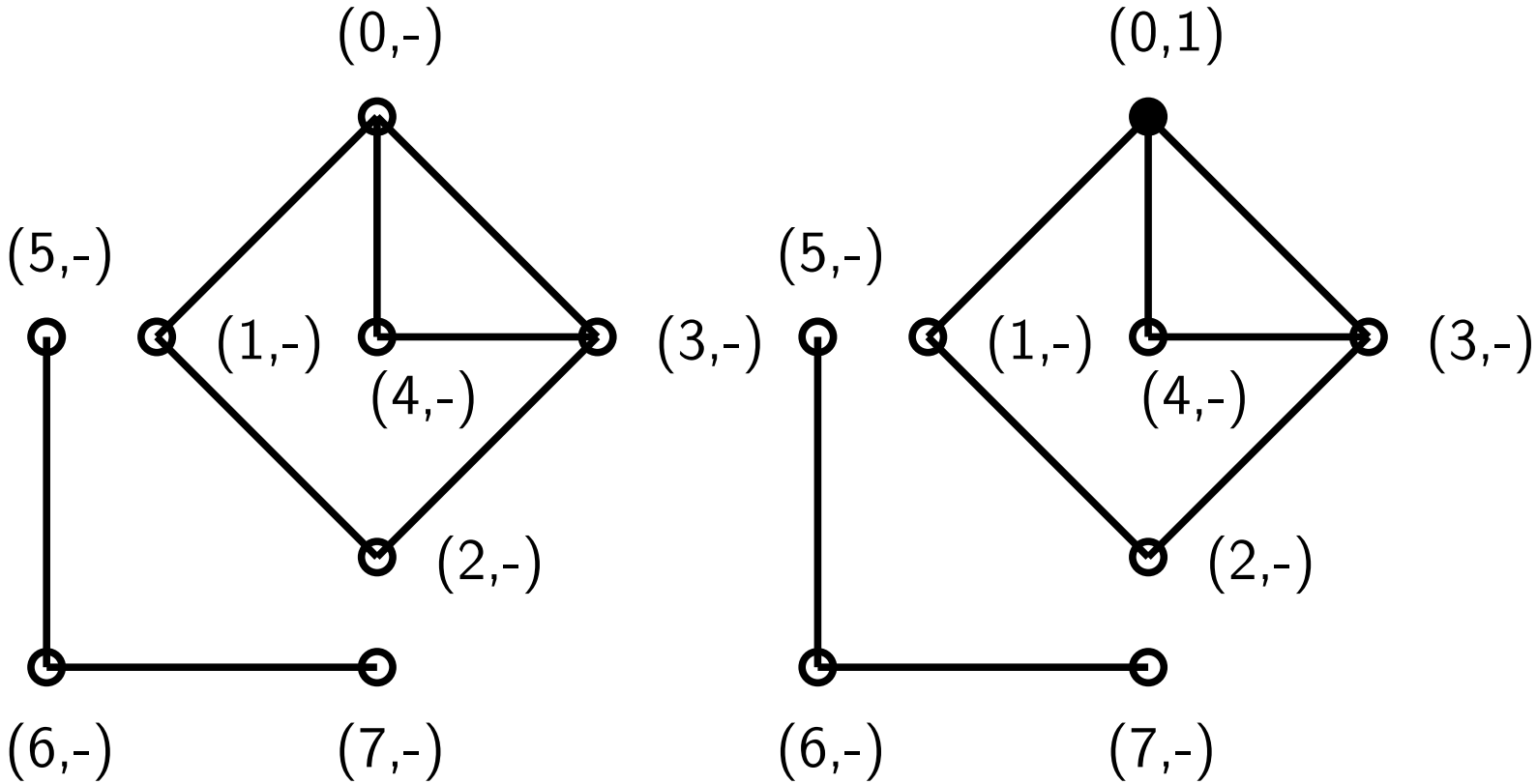
Recorrido de un grafo

- Cuando se trabaja con grafos uno se encuentra con preguntas comunes.
- ¿Tiene ciclos? ¿Es conexo? ¿Cuáles son sus componentes conexas?
- Para responder a estas preguntas se debe poder recorrer un grafo de forma metódica, de modo que se visiten todos sus vértices y aristas (de preferencia una sola vez).
- Estos algoritmos pueden ser recursivos o iterativos y sólo dependen un poco de la representación.

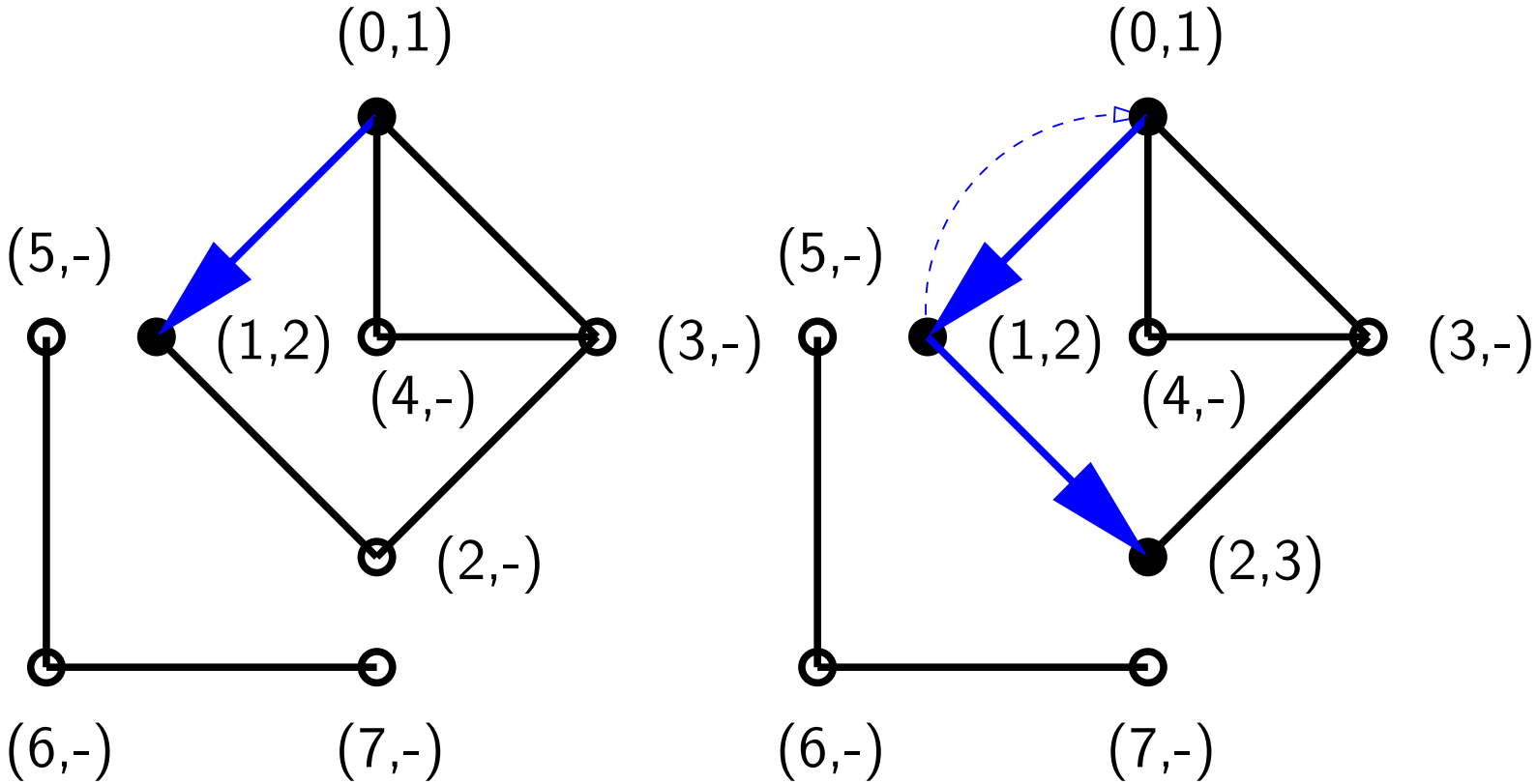
Búsqueda en profundidad

- El primer método se puede describir fácilmente de forma recursiva.
- Al inicio se marcan todos los vértices del grafo como no vistos.
- Luego, para cada vértice no visto se le marca como ya visto y se visita recursivamente a todos sus vecinos no vistos.

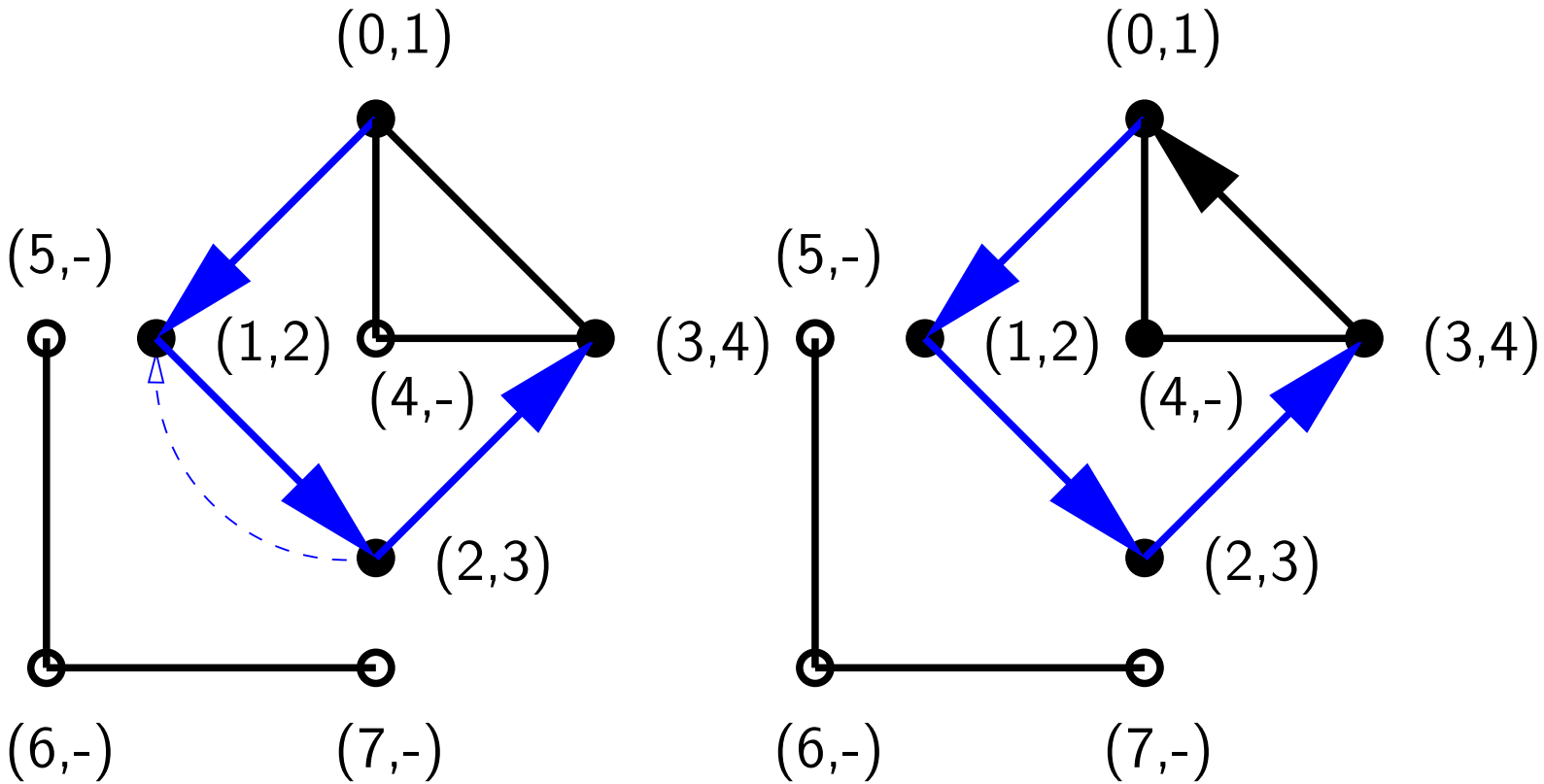
Ejemplo de profundidad (1)



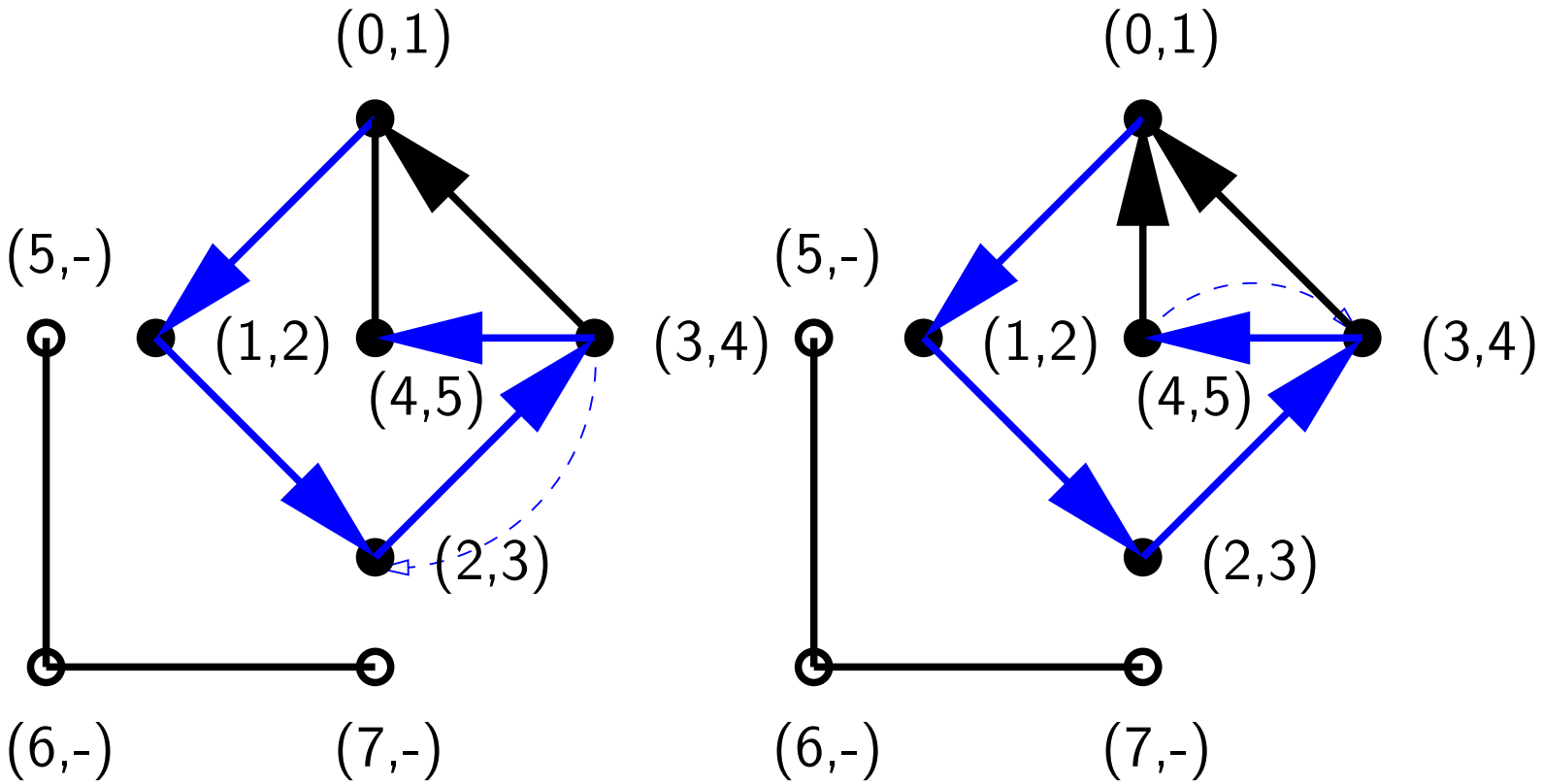
Ejemplo de profundidad (2)



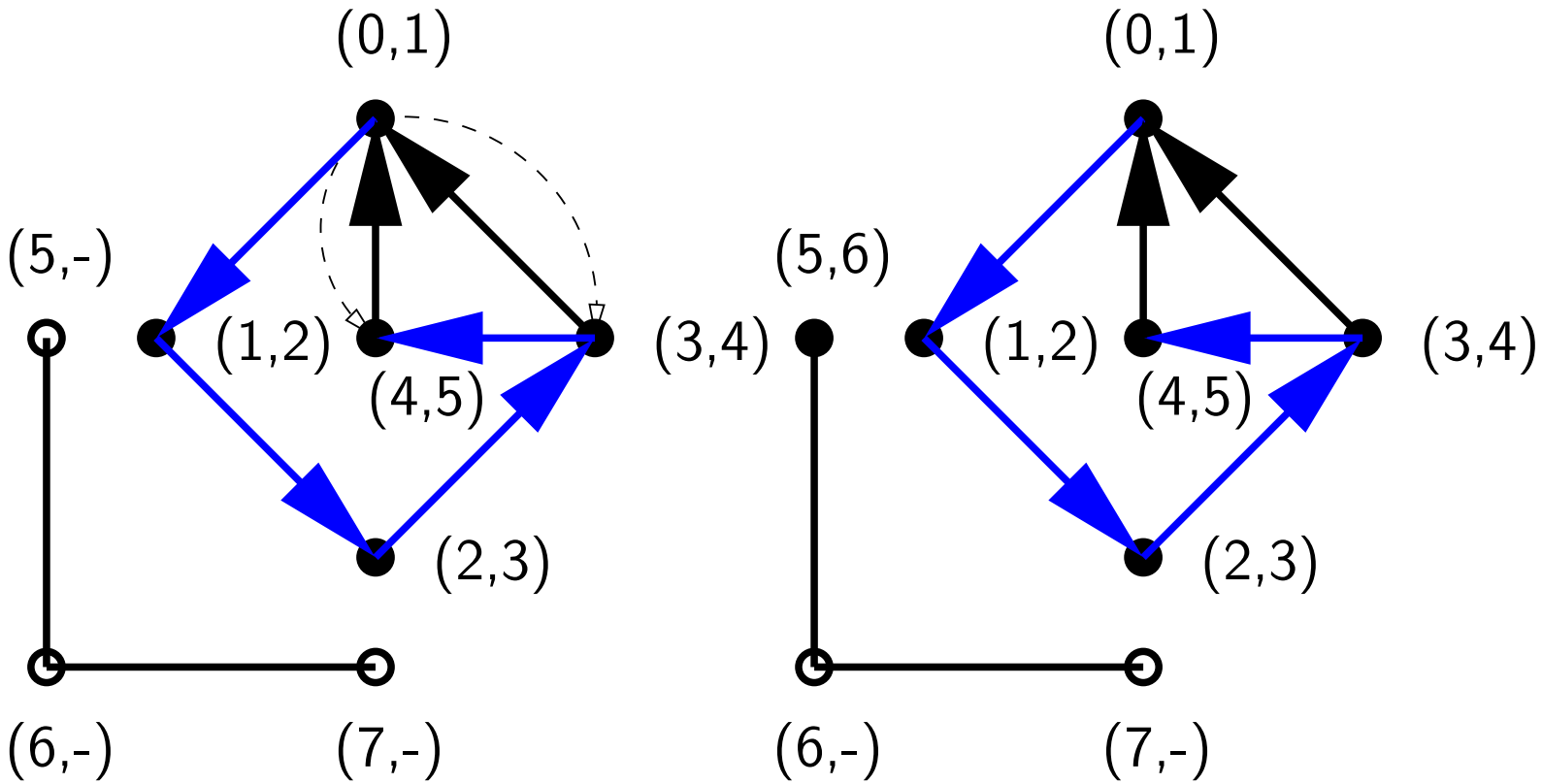
Ejemplo de profundidad (3)



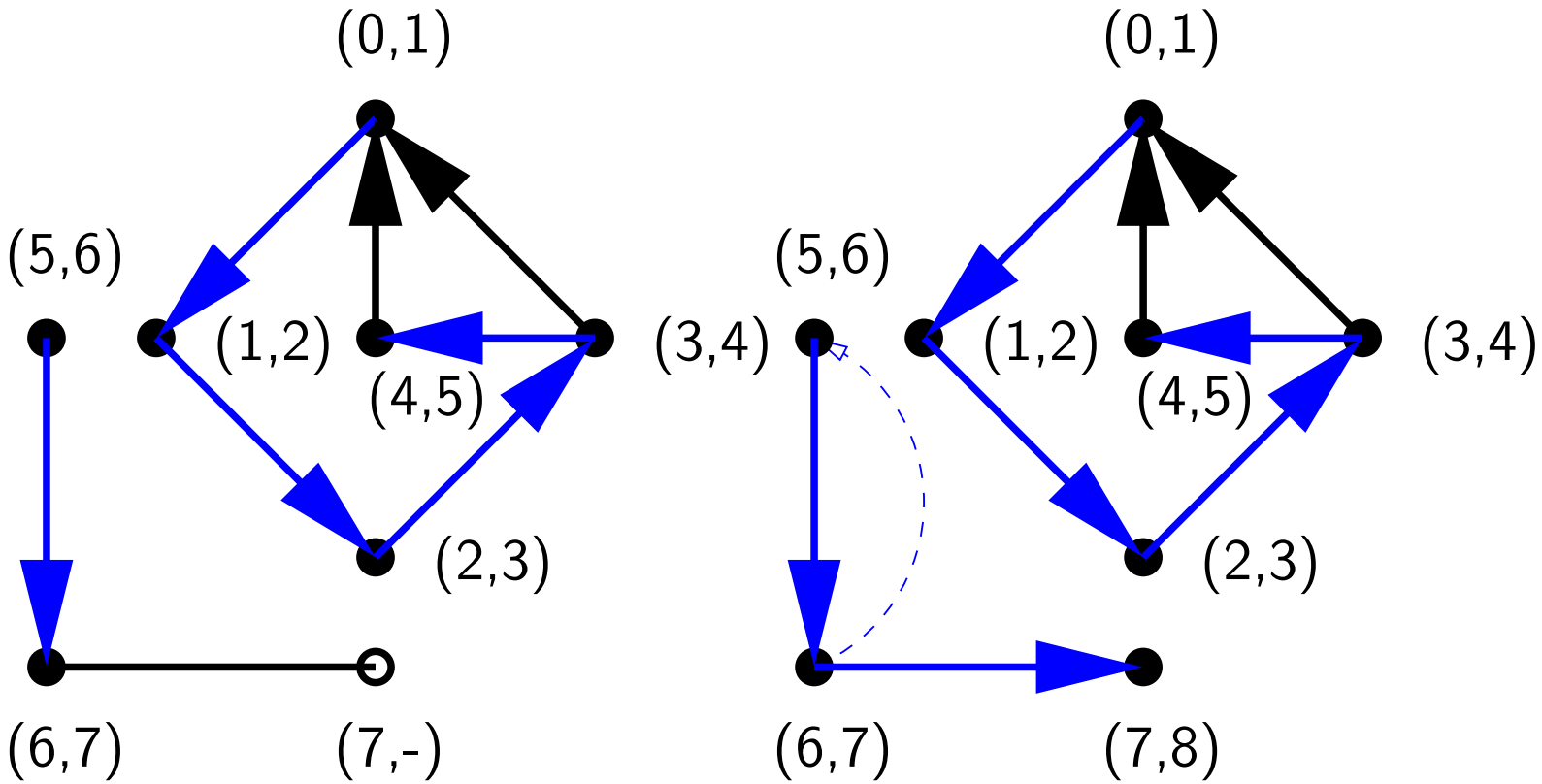
Ejemplo de profundidad (4)



Ejemplo de profundidad (5)



Ejemplo de profundidad (6)



Implementación de profundidad

- Se necesita un arreglo `visto` que indique si el vértice correspondiente ha sido visto o no.
- El valor de no visto puede ser 0 mientras que el de visto puede ser el del orden de visita.
- El resto depende de la representación.

```
orden = 0;
for (k = 0; k < n; k++)
  visto[k] = 0;
for (k = 0; k < n; k++)
  if (visto[k] == 0)
    visita(k);
```

Profundidad con listas

- En este caso la búsqueda en profundidad toma tiempo $\propto m + n$.

```
void visita(int k)
{
  nodo *t;

  visto[k] = ++orden;
  for (t = a[k]; t != NULL; t = t->sig)
    if (visto[t->v] == 0)
      visita(t->v);
}
```

Profundidad con matriz

- En este caso la búsqueda en profundidad toma tiempo $\propto n^2$.

```
void visita(int k)
{
  int t;

  visto[k] = ++orden;
  for (t = 0; t < n; t++)
    if (a[k][t] != 0)
      if (visto[t] == 0)
        visita(t);
}
```

Profundidad no recursiva

- Se necesita una `pila` (cualquier representación).

```
void visita(int k)
{
  nodo *t;
  mete(k, pila);
  while (!vacia(pila)) {
    k = saca(pila);
    visto[k] = ++orden;
    for (t = a[k]; t != NULL; t = t->sig)
      if (visto[t->v] == 0) {
        mete(t->v, pila);
        visto[t->v] = -1; /* en pila */
      }
  }
}
```

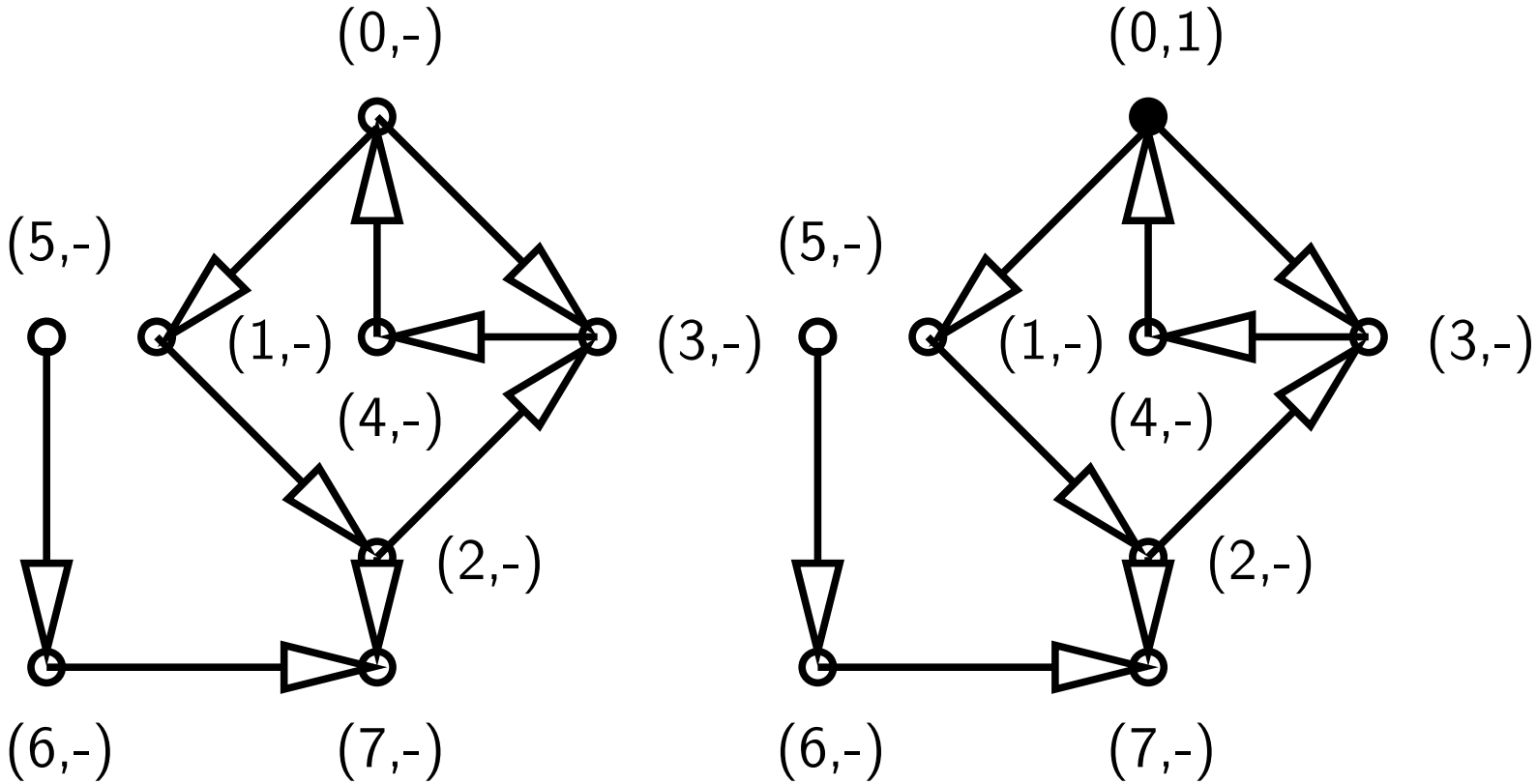
Observaciones de profundidad

- Con este método se visita cada vértice una vez y cada arista dos veces.
- El orden de visita de los vértices y aristas depende de la representación.
- Las aristas que visitan por primera vez vértices no vistos forman el bosque de búsqueda en profundidad.
- Las demás aristas apuntan hacia arriba.
- Este método es una generalización del recorrido en preorden de un árbol.

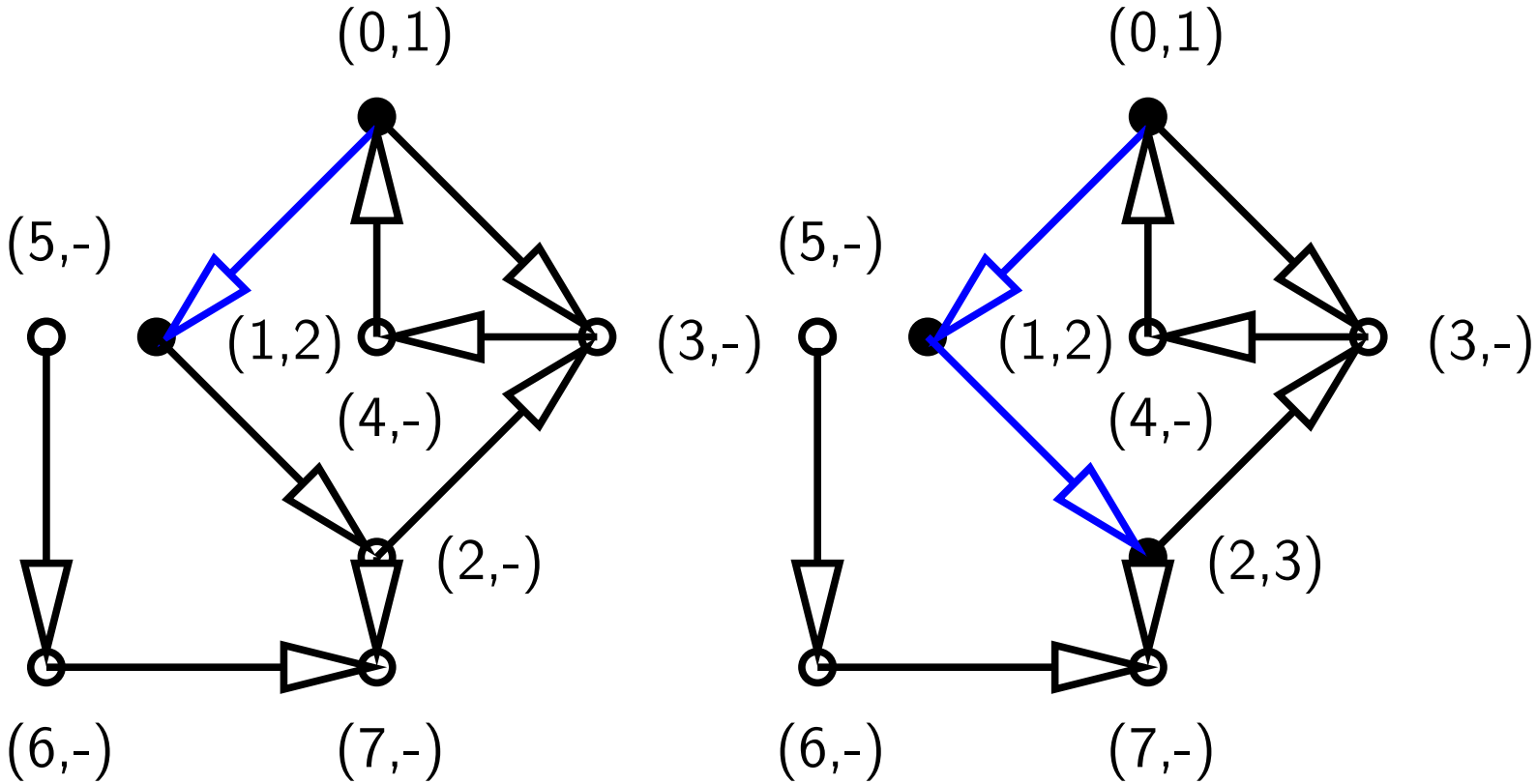
Profundidad en grafos dirigidos

- Los algoritmos de búsqueda en profundidad para grafos se pueden usar sin cambios en grafos dirigidos.
- Lo que cambia es la estructura generada.
- Los arcos solamente se visitan una vez.
- Los arcos que no pertenecen al bosque de búsqueda en profundidad pueden apuntar hacia arriba, hacia abajo o ser transversales.

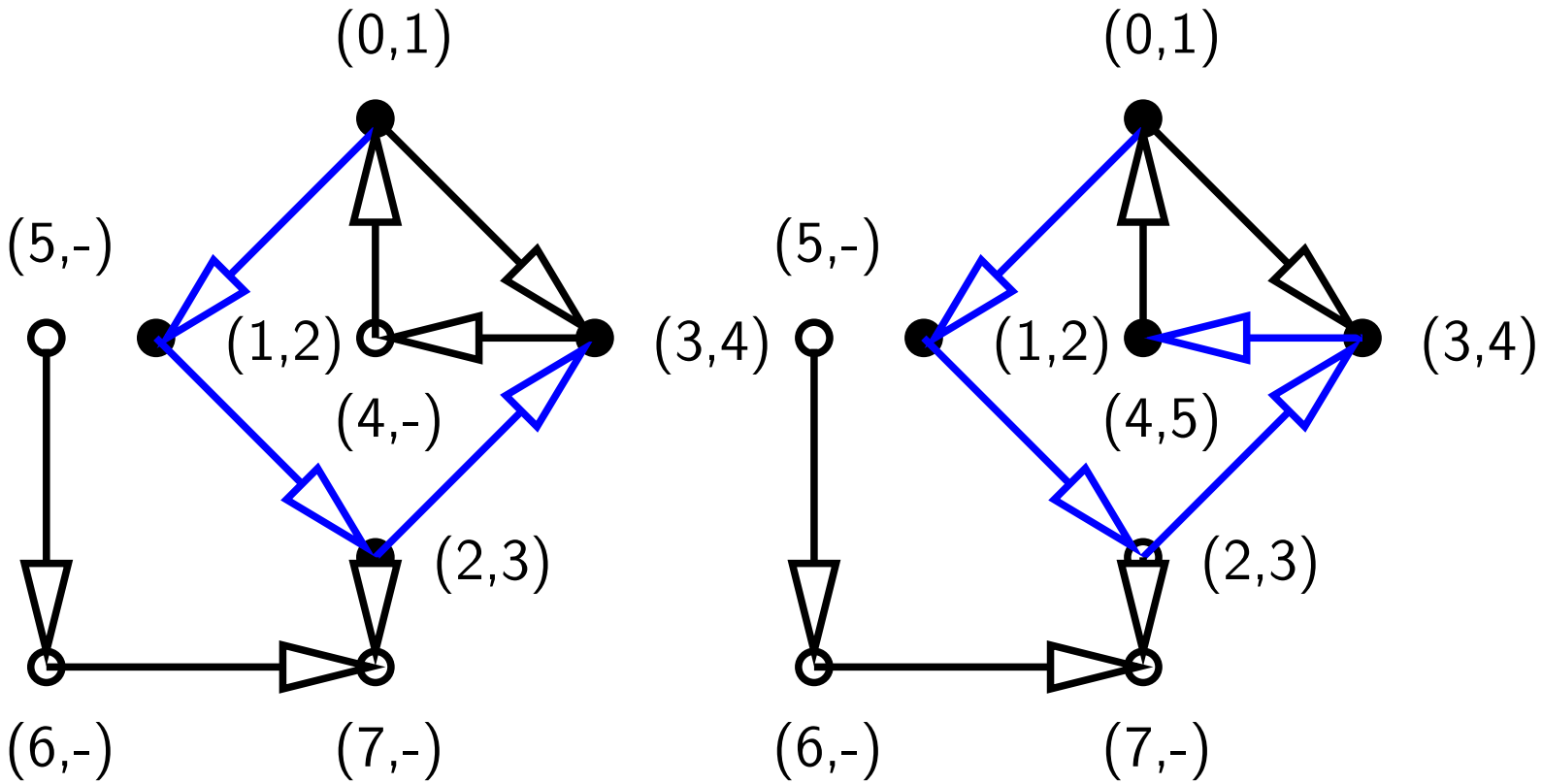
Ejemplo de profundidad dirigida 1



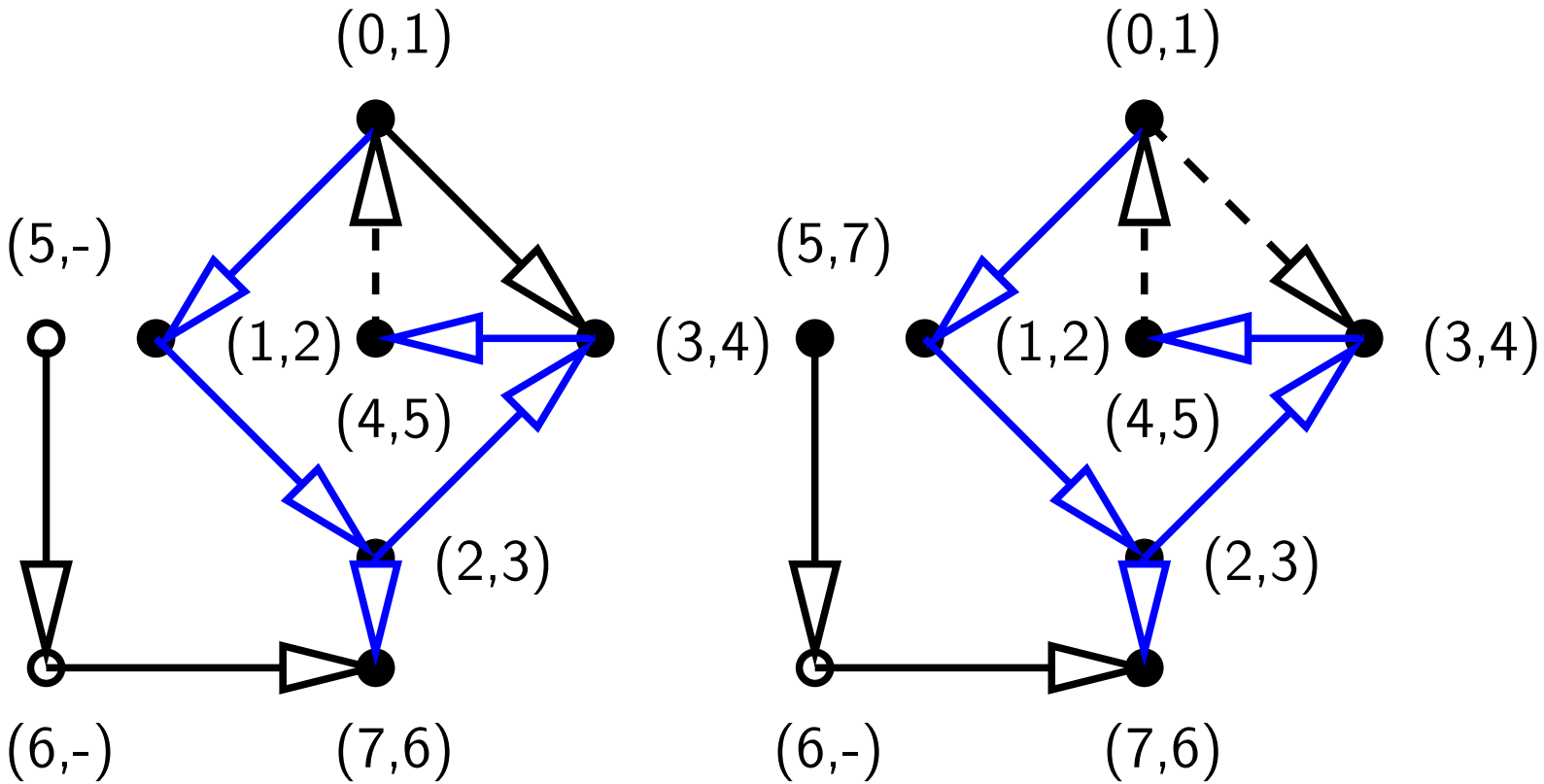
Ejemplo de profundidad dirigida 2



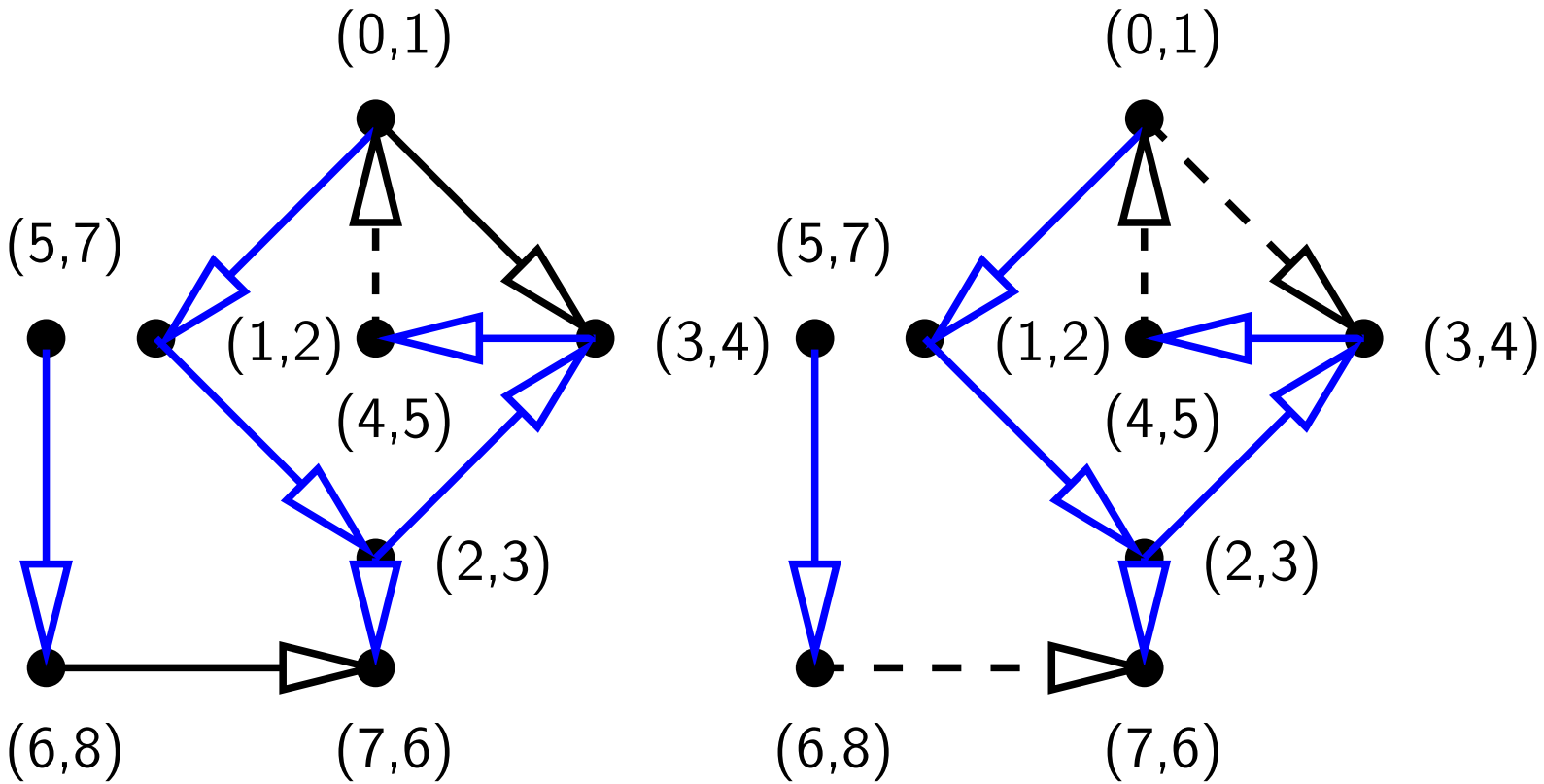
Ejemplo de profundidad dirigida 3



Ejemplo de profundidad dirigida 4



Ejemplo de profundidad dirigida 5



Contenido

7 Representación de grafos y aplicaciones

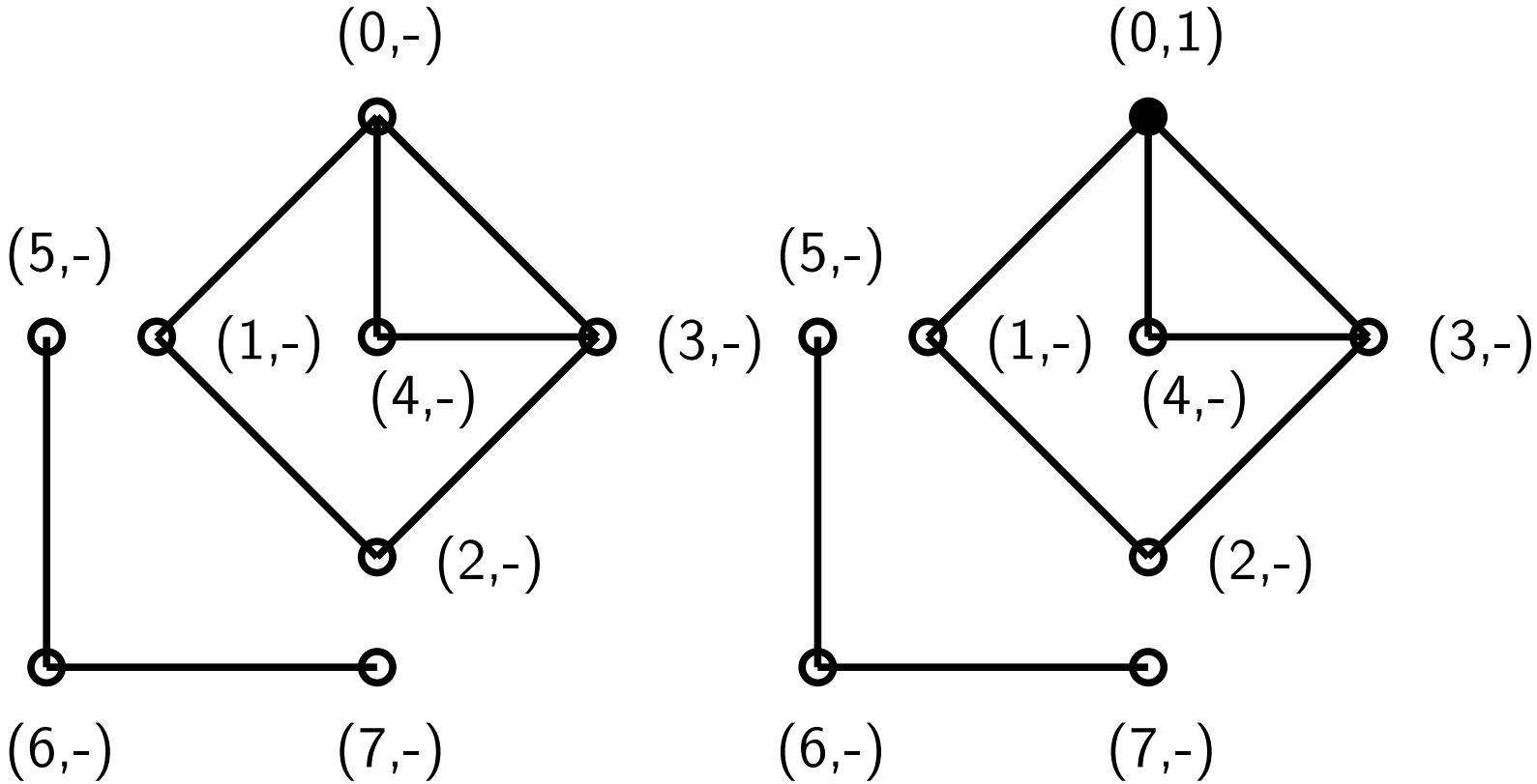
8 Recorridos de un grafo
- Búsqueda en profundidad
- Búsqueda en amplitud
- Componentes conexas y biconexas
- Unión y pertenencia
- Ordenamiento topológico

9 Grafos con costos

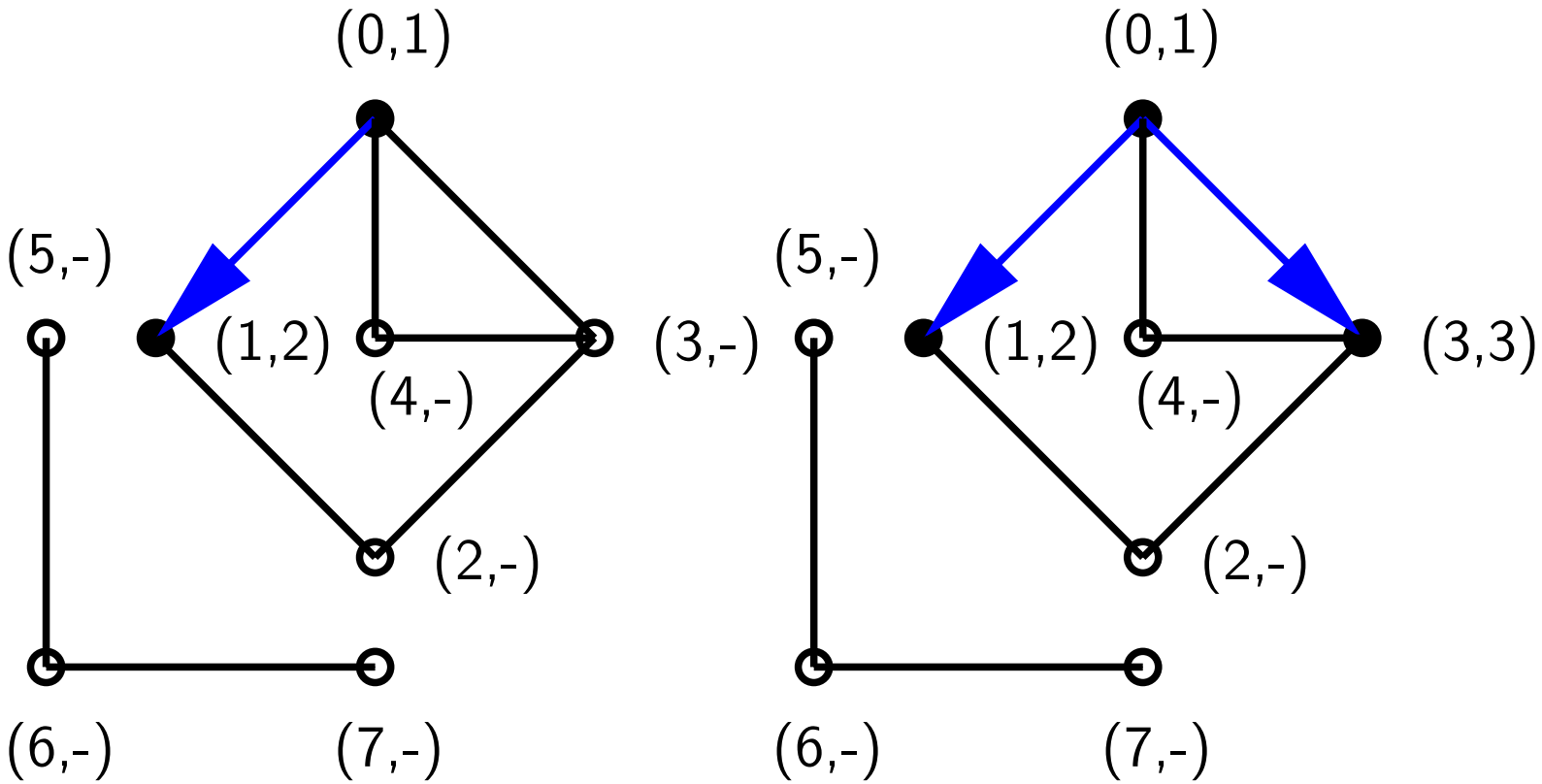
Búsqueda en amplitud

- Si en el método de búsqueda en profundidad no recursivo se reemplaza la pila por una cola se obtiene un recorrido distinto.
- A éste se le llama búsqueda en amplitud.
- Este método es parecido al de recorrer un árbol por niveles a partir de la raíz.

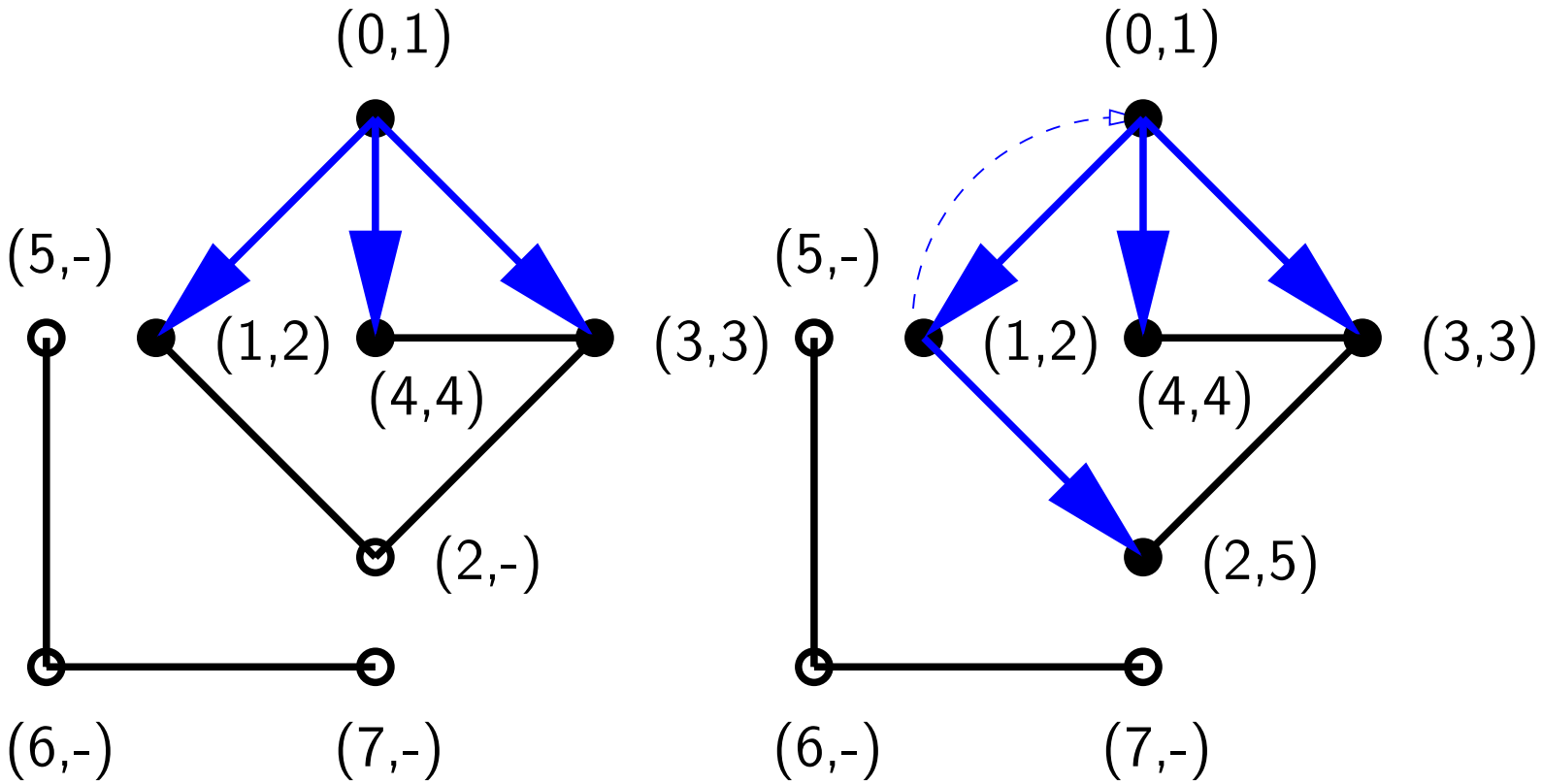
Ejemplo de amplitud 1



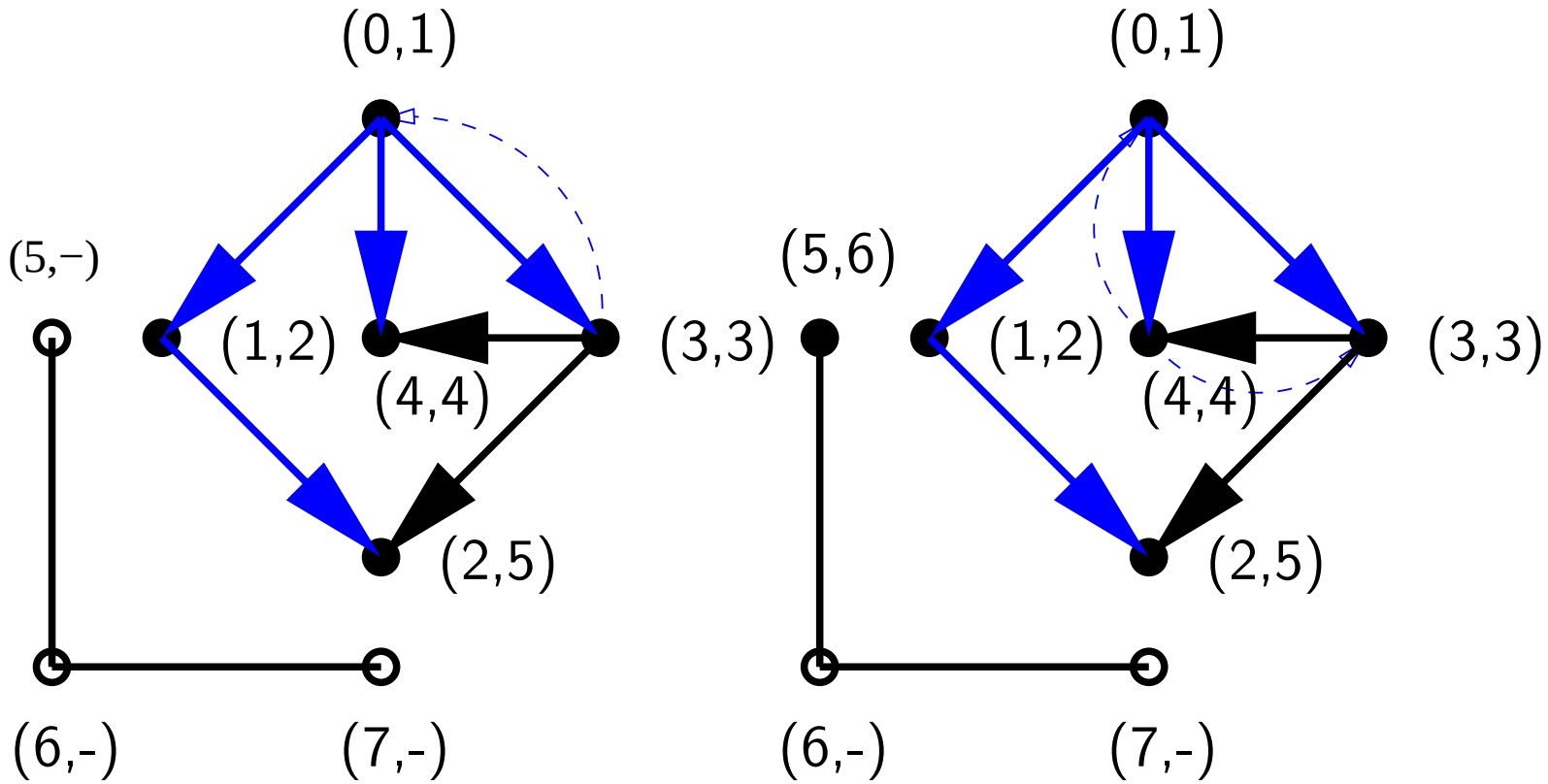
Ejemplo de amplitud 2



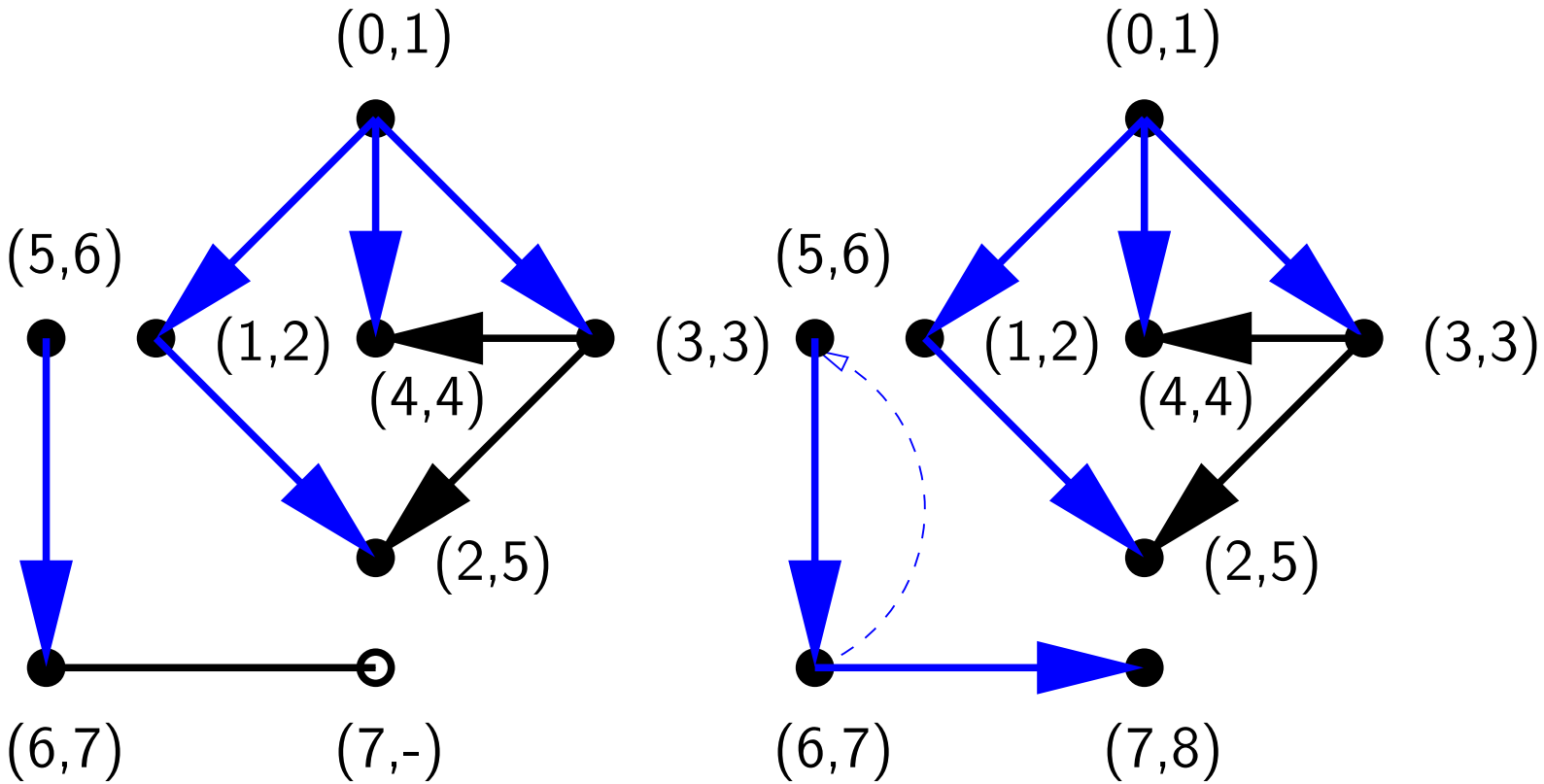
Ejemplo de amplitud 3



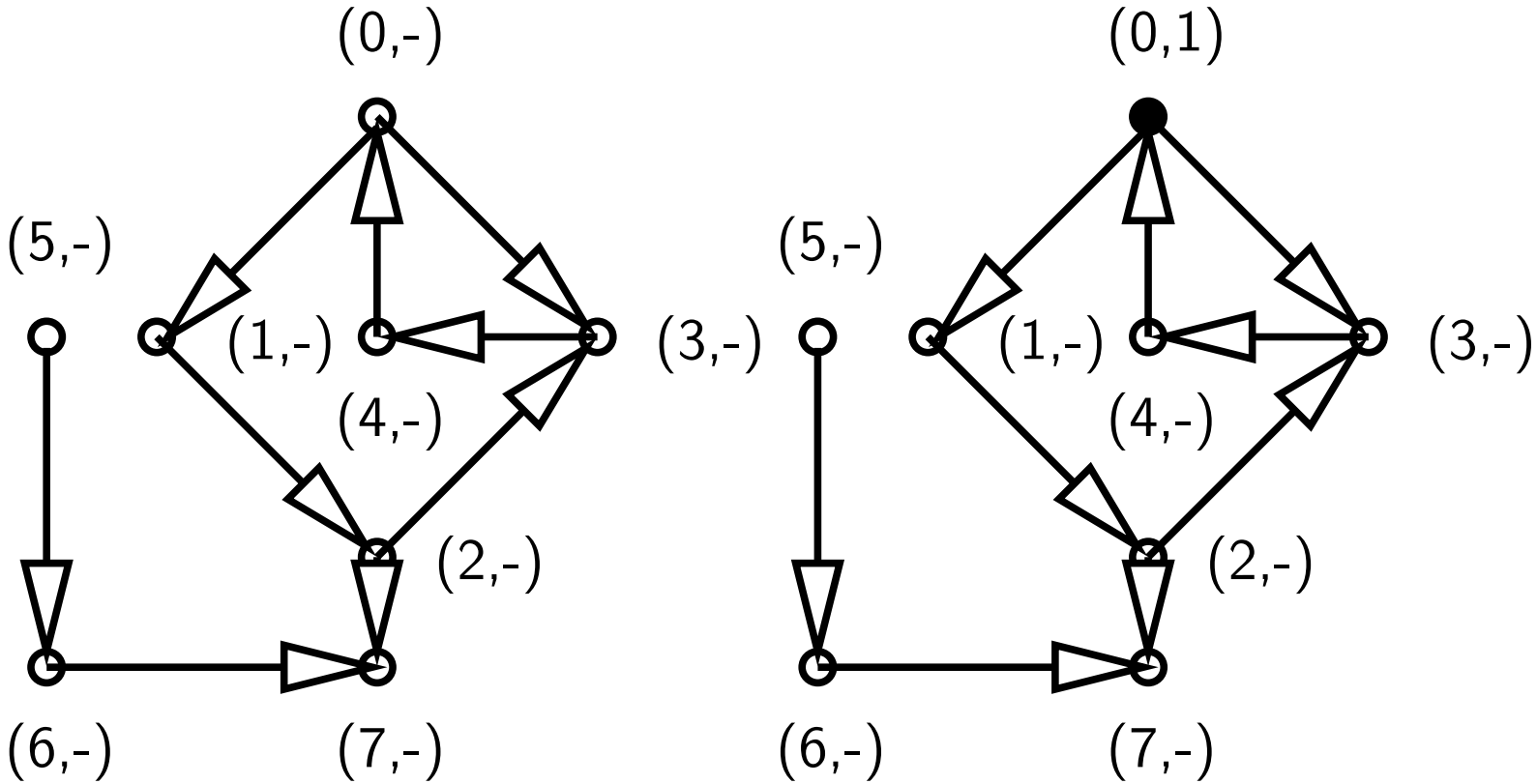
Ejemplo de amplitud 4



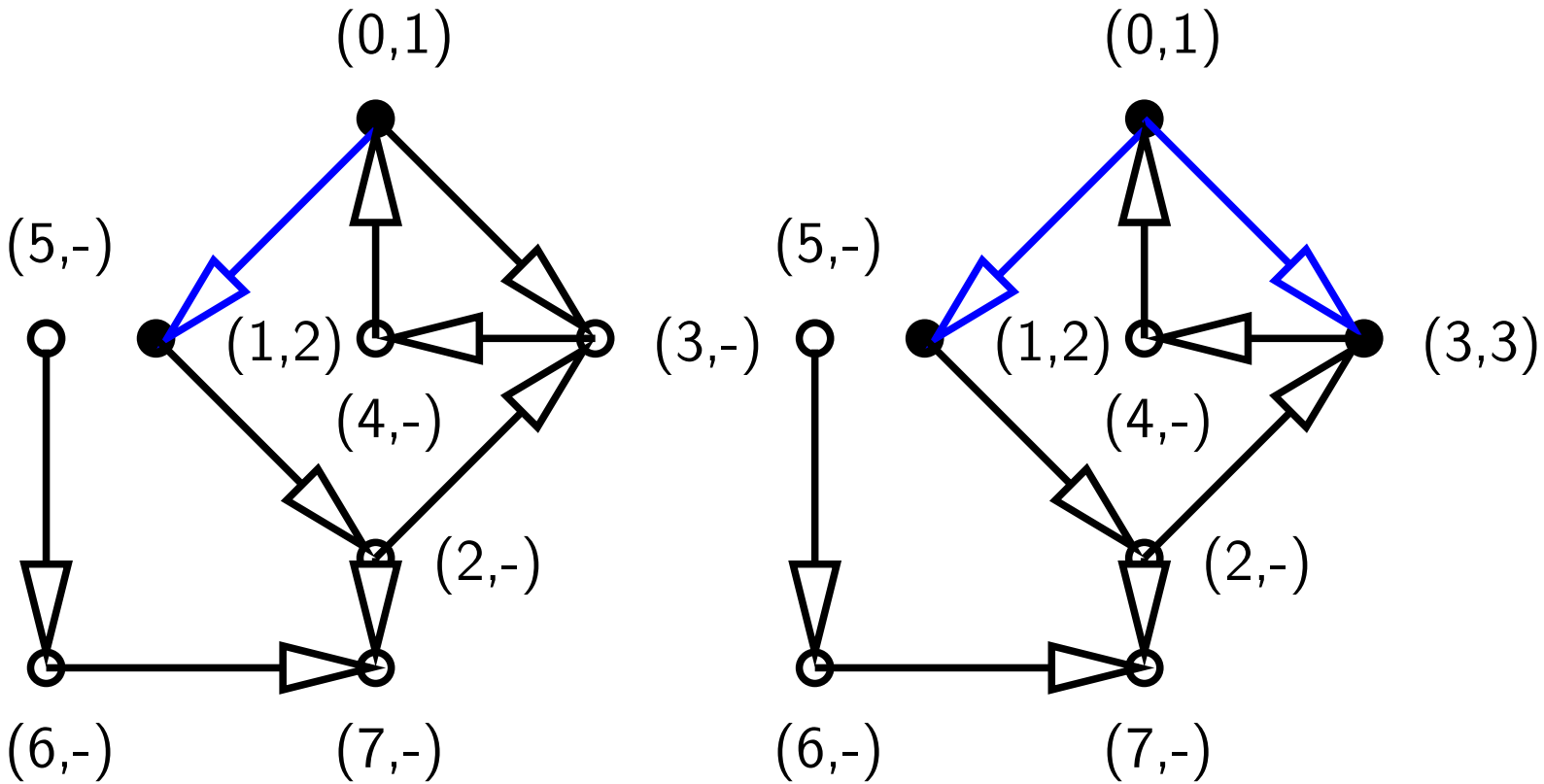
Ejemplo de amplitud 5



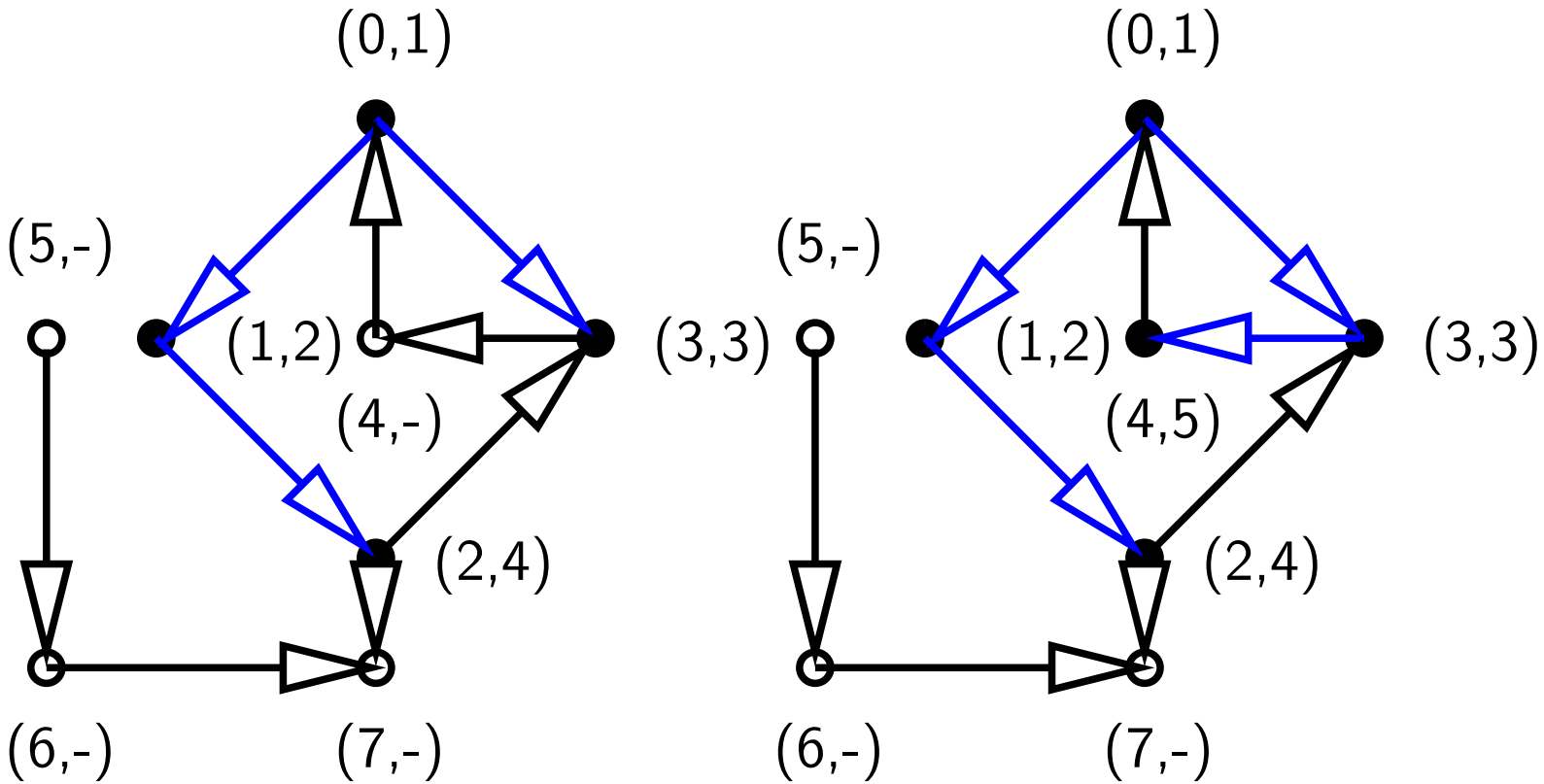
Ejemplo de amplitud dirigida 1



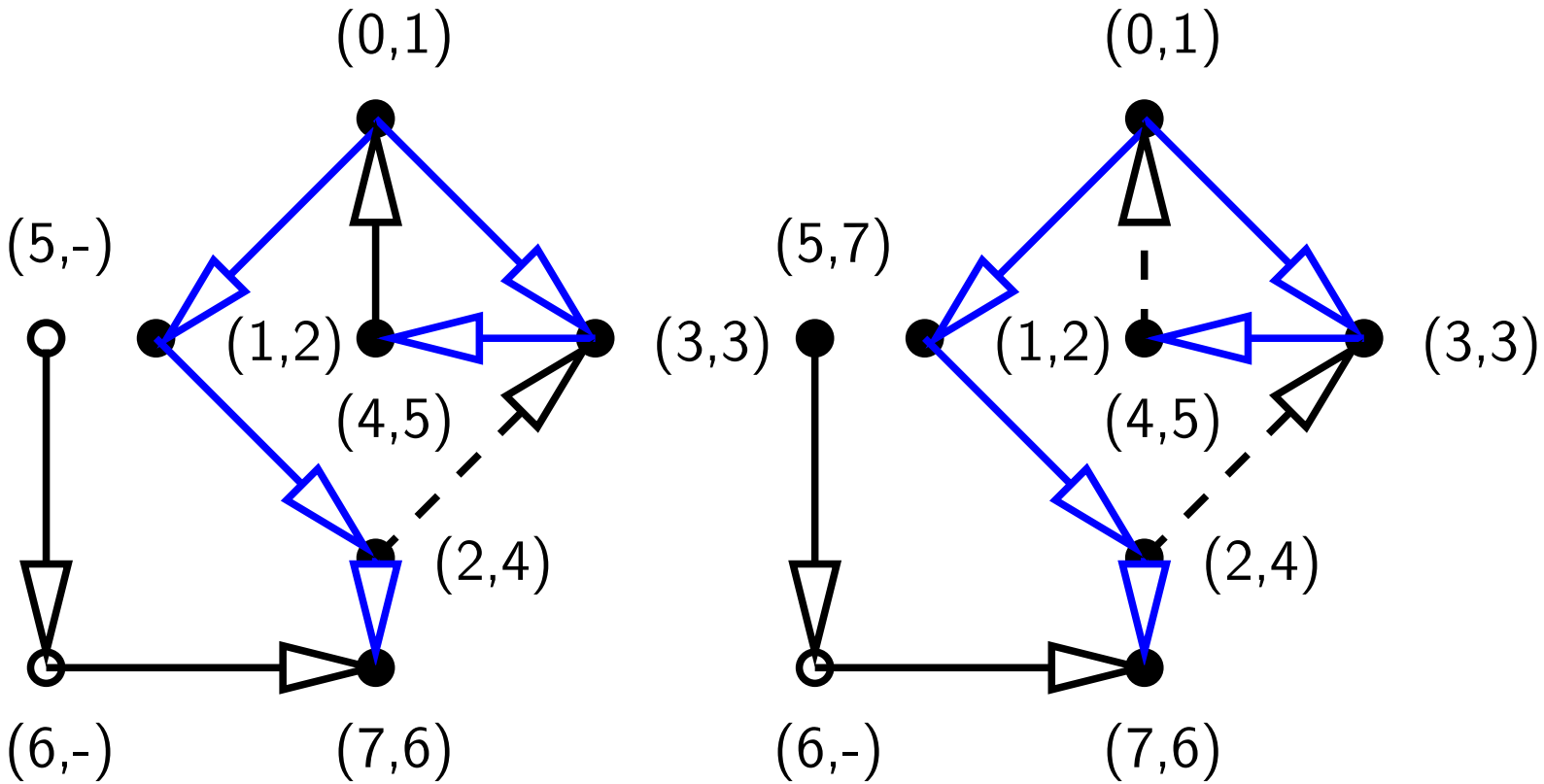
Ejemplo de amplitud dirigida 2



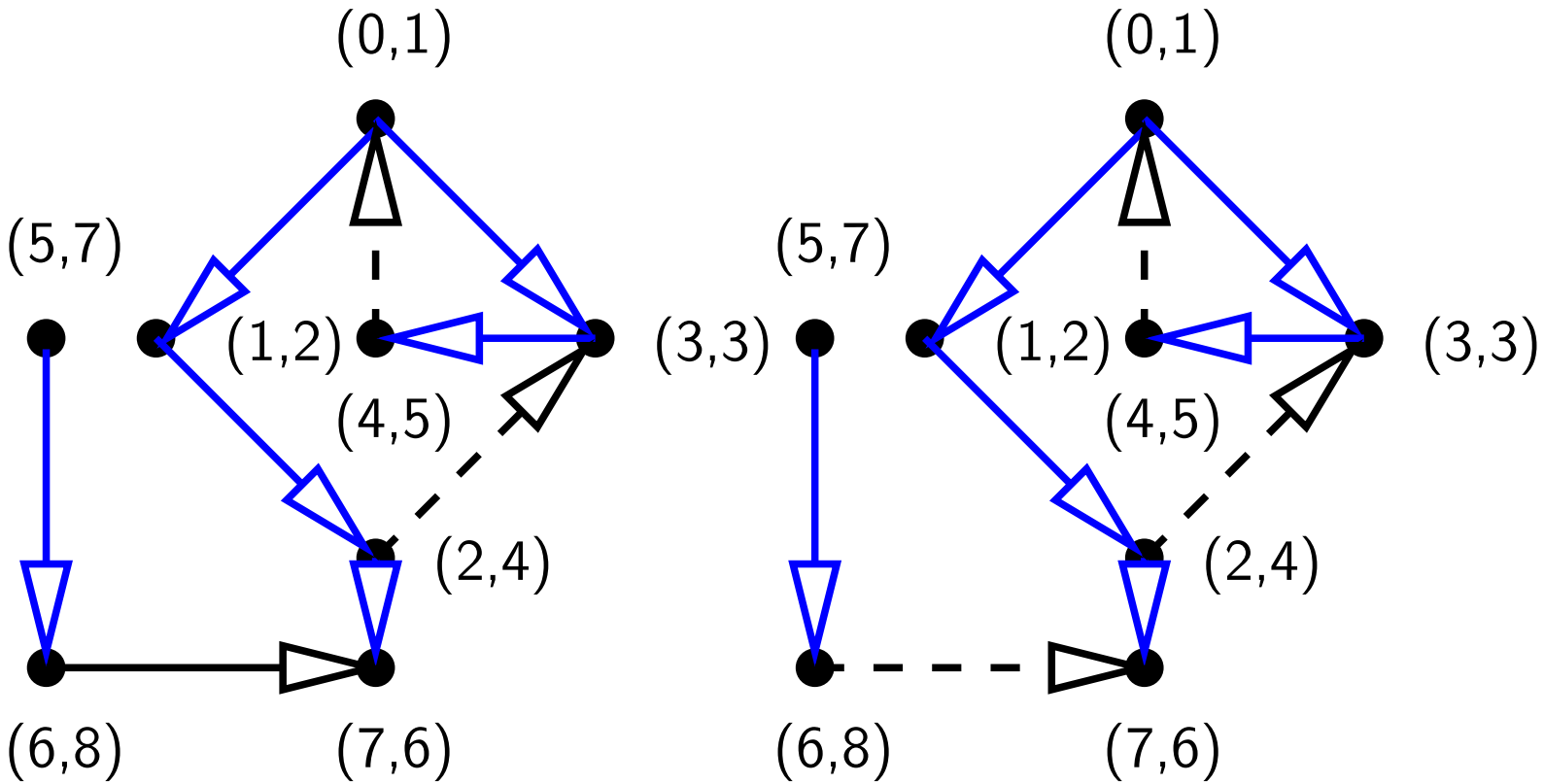
Ejemplo de amplitud dirigida 3



Ejemplo de amplitud dirigida 4



Ejemplo de amplitud dirigida 5



Contenido

7 Representación de grafos y aplicaciones

8 Recorridos de un grafo
- Búsqueda en profundidad
- Búsqueda en amplitud
- Componentes conexas y biconexas
- Unión y pertenencia
- Ordenamiento topológico

9 Grafos con costos

Componentes conexas

- Tanto la búsqueda en profundidad como en amplitud se pueden usar para encontrar las componentes conexas de un grafo.
- Las únicas modificaciones que se necesitan son que `visita` enumere el vértice que se acaba de visitar y que la llamada no recursiva a `visita` separe las listas de vértices.
- Esto se puede hacer con un arreglo `inver` que tome los valores `inver[orden] = k` (el negativo en la llamada no recursiva).

Implementación de componentes

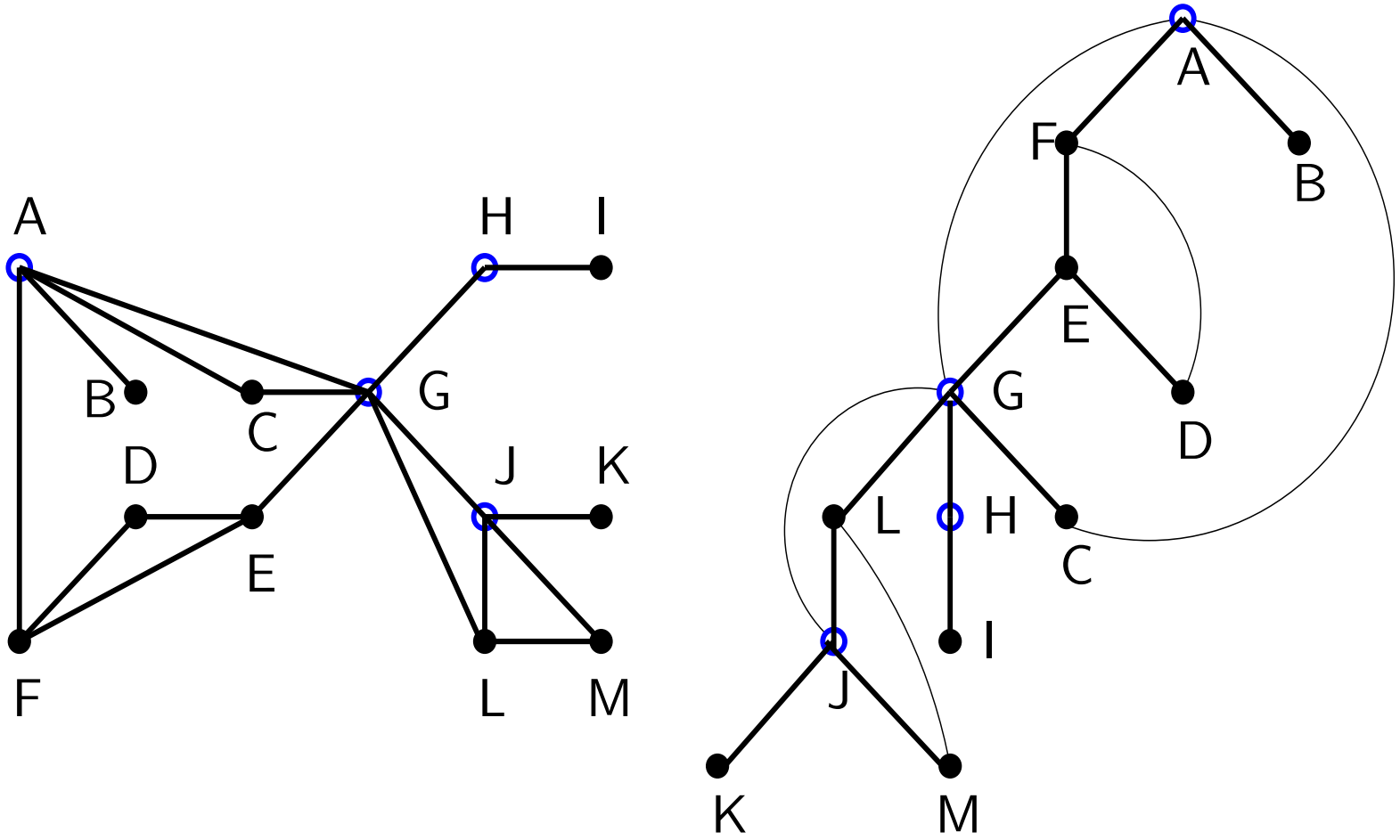
```
orden = 0;
for (k = 0; k < n; k++)
  visto[k] = 0;
for (k = 0; k < n; k++)
  if (visto[k] == 0) {
    visita(k);
    inver[visto[k]] = -inver[visto[k]];
  }

void visita(int k)
{
  nodo *t;
  visto[k] = ++orden;
  inver[orden] = k;
  for (t = a[k]; t != NULL; t = t->sig)
    if (visto[t->v] == 0)
      visita(t->v);
}
```

Componentes biconexas

- A veces es útil revisar que un grafo no tiene cuellos de botella o puntos de fallo únicos.
- Un punto de articulación es un vértice de un grafo que al borrarlo aumenta el número de componentes conexas del grafo.
- Un grafo que no tiene puntos de articulación se llama biconexo.
- En un grafo biconexo cada pareja de vértices distintos están conectados por al menos dos caminos disjuntos.

Ejemplo de componentes biconexas



Componentes biconexas de un grafo.

Algoritmo de componentes biconexas

- Modifiquemos la búsqueda en profundidad para calcular las componentes biconexas.
- Un vértice x no es de articulación si cada uno de sus hijos y tiene algún descendiente conectado a un punto más alto que x.
- Excepto si es la raíz, que es un punto de articulación si tiene dos o más hijos.
- Hagamos que `visita` devuelva el vértice más alto del árbol encontrado.

Implementación de biconexas

```
int visita(int k)
{
  nodo *t;
  int temp, alto;

  visto[k] = ++orden;
  alto = orden;
  for (t = a[k]; t != NULL; t = t->sig)
    if (visto[t->v] == 0) {
      temp = visita(t->v);
      if (temp < alto)
        alto = temp;
      if (temp >= visto[k])
        articulacion(k);
    } else if (visto[t->v] < alto)
      alto = visto[t->v];
  return alto;
}
```

Contenido

7 Representación de grafos y aplicaciones

8 Recorridos de un grafo
- Búsqueda en profundidad
- Búsqueda en amplitud
- Componentes conexas y biconexas
- Unión y pertenencia
- Ordenamiento topológico

9 Grafos con costos

Unión y pertenencia

- En algunas variantes del problema de conexidad lo único que se quiere saber es si un cierto vértice está conectado a otro.
- Otro problema idéntico es el siguiente: se tiene una familia de subconjuntos disjuntos y se quiere saber si dos elementos están en el mismo subconjunto.
- En el problema de gráficas los subconjuntos corresponden con las componentes conexas.

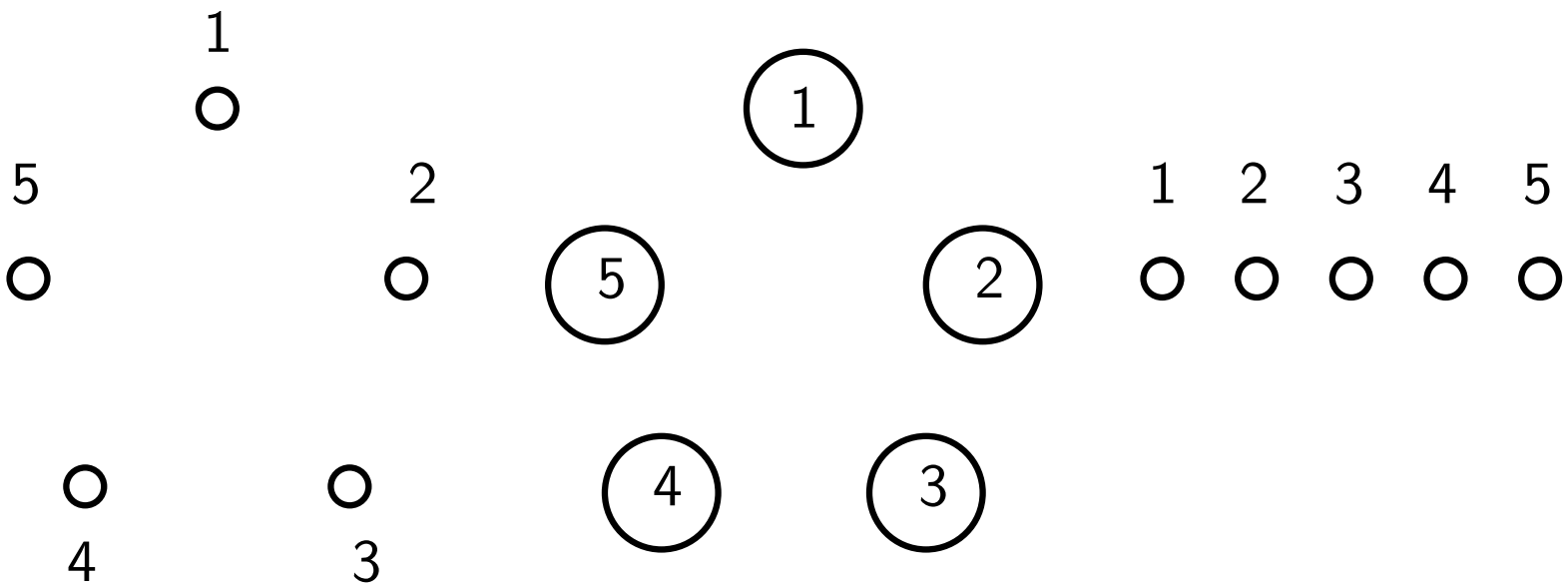
Gráficas dinámicas

- Hasta ahora todas las gráficas que hemos considerado han sido estáticas (no cambian).
- El algoritmo que estudiaremos funciona con gráficas a las que se les agregan aristas.
- A la operación de agregar una arista la llamaremos unión y a la pregunta de conexidad la llamaremos pertenencia.
- Estos nombres vienen de la versión del problema con conjuntos.

Bosque de árboles

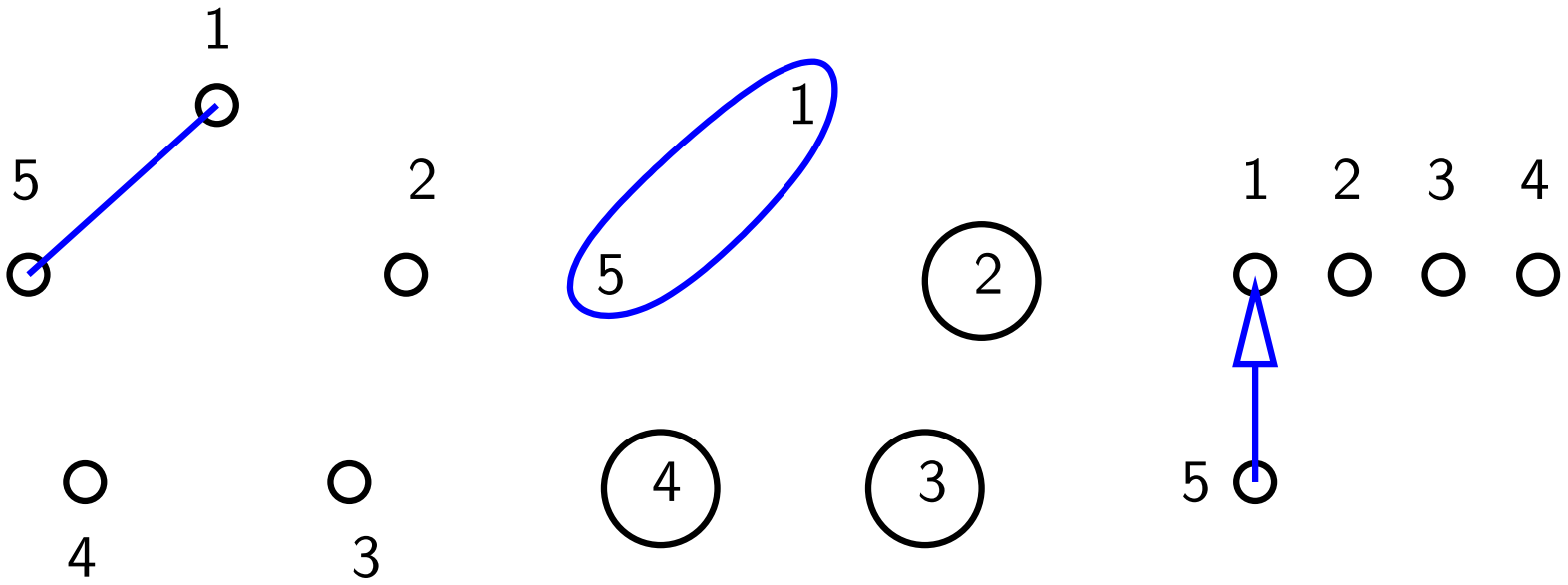
- La estructura que usaremos para representar a los subconjuntos es un bosque de árboles.
- Al principio cada elemento forma su propio subconjunto y queda representado por un árbol con un solo vértice.
- Cuando se unen dos subconjuntos, sus árboles se juntan en un solo árbol.
- Para verificar la pertenencia se revisa si los dos elementos están en el mismo árbol.

Ejemplo de unión y pertenencia 1



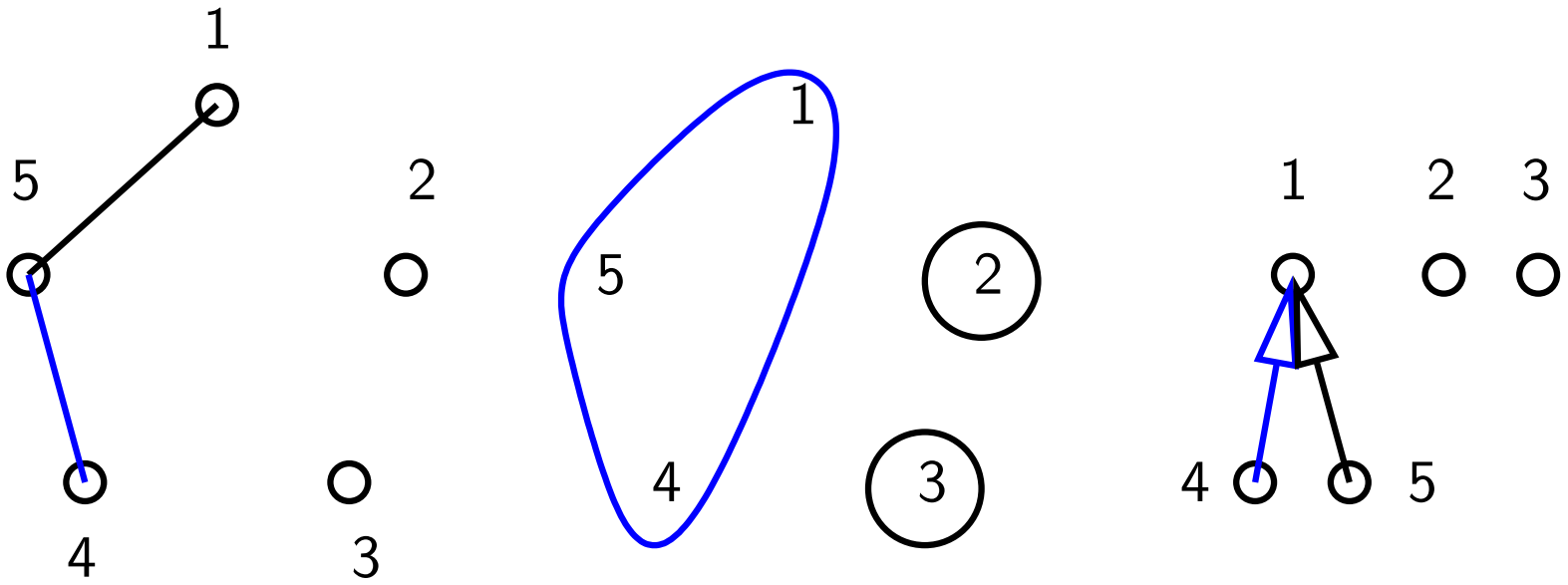
Gráfica sin aristas.

Ejemplo de unión y pertenencia 2



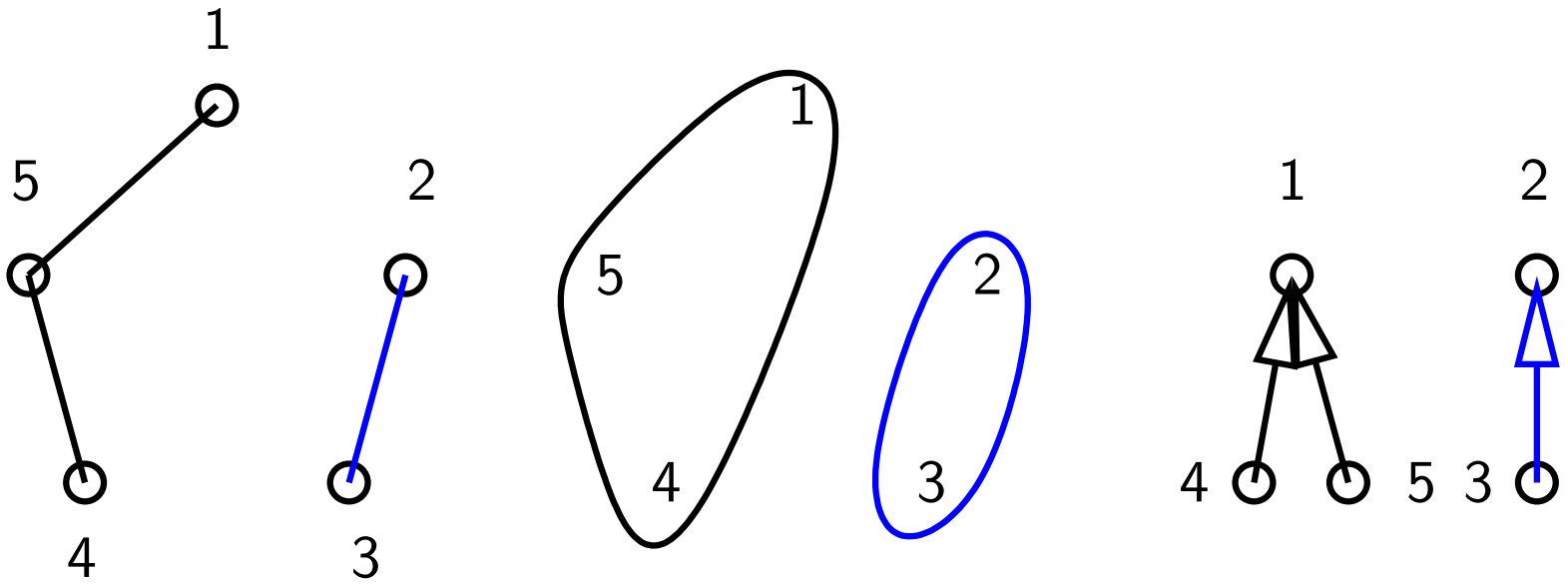
Se añade la arista $(1, 5)$.

Ejemplo de unión y pertenencia 3



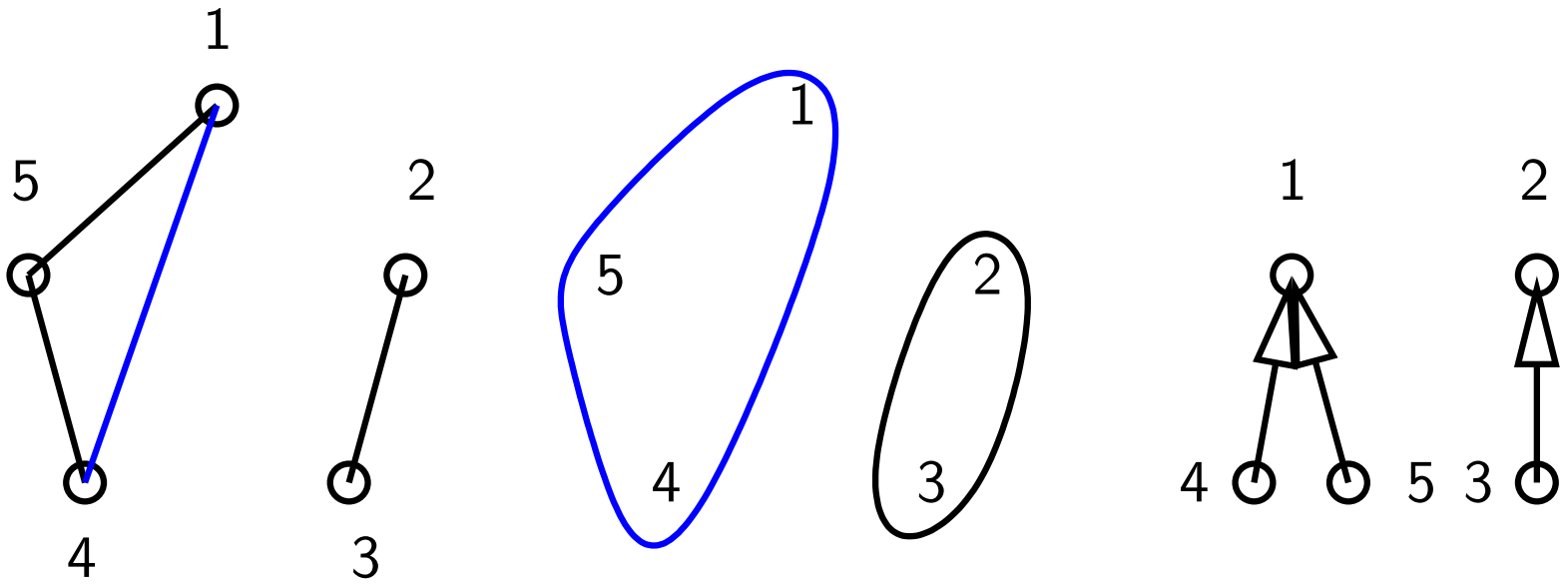
Se añade la arista $(4, 5)$.

Ejemplo de unión y pertenencia 4



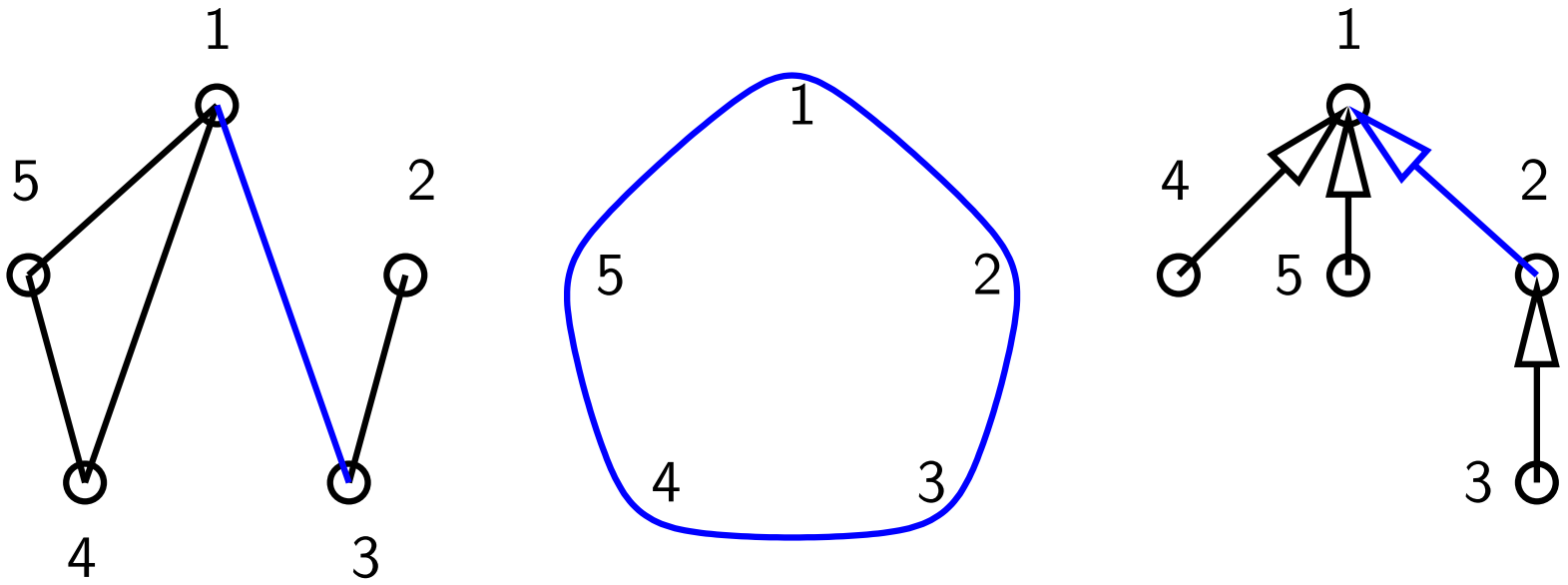
Se añade la arista $(3, 2)$.

Ejemplo de unión y pertenencia 5



Se añade la arista $(4, 1)$.

Ejemplo de unión y pertenencia 6



Se añade la arista $(1, 3)$.

Representación de los árboles

- Usaremos un arreglo `padre` donde cada entrada señala al padre.
- Al inicio el arreglo satisface `padre[i] = i`.
- La raíz del árbol que contiene a un elemento `i` se encuentra iterando `i = padre[i]`.
- Dos elementos están en el mismo árbol si tienen la misma raíz.
- Para hacer la unión de dos elementos buscamos sus raíces `i, j` y si son distintas hacemos `padre[i] = j`.

Contenido

7 Representación de grafos y aplicaciones

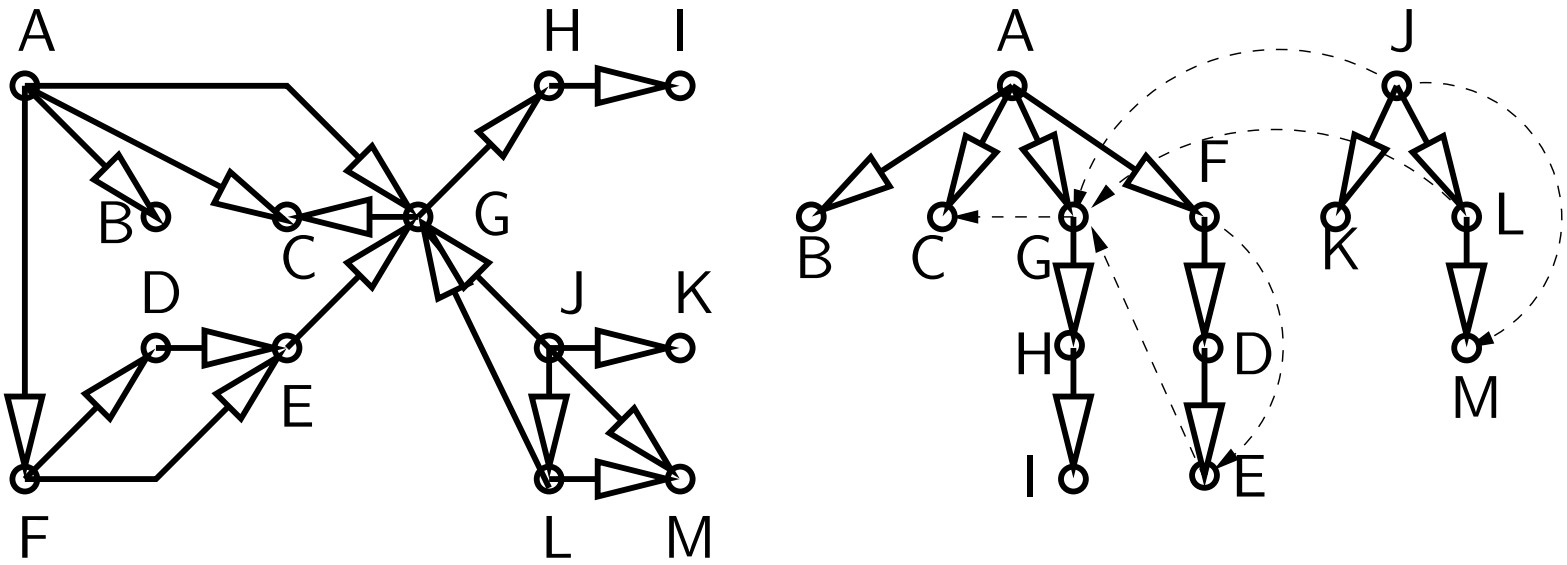
8 Recorridos de un grafo
- Búsqueda en profundidad
- Búsqueda en amplitud
- Componentes conexas y biconexas
- Unión y pertenencia
- Ordenamiento topológico

9 Grafos con costos

Grafos dirigidos acíclicos

- En muchas aplicaciones es importante que un grafo dirigido no contenga ciclos dirigidos.
- Por ejemplo, si los vértices representan actividades y los arcos precedencias entonces la presencia de un ciclo dirigido implica una inconsistencia.
- Se les llama grafos dirigidos acíclicos (en inglés directed acyclic graph o DAG).

Ejemplo de grafo acíclico



Búsqueda en profundidad en un grafo acíclico.

Ordenamiento topológico

- Los vértices de un grafo acíclico pueden procesarse de modo que ningún vértice se procese antes que otro que apunte a él.
- A esto se le llama un orden topológico (y generalmente hay más de uno de ellos).
- A veces lo que interesa es un orden topológico inverso.
- La búsqueda en profundidad recursiva encuentra un orden topológico inverso.
- ¿Cómo se halla un orden normal?

Contenido

7 Representación de grafos y aplicaciones

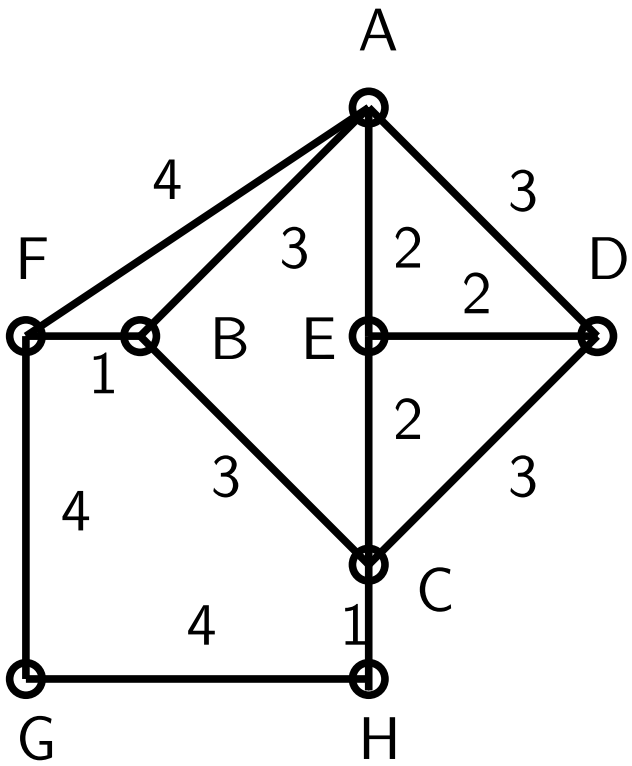
8 Recorridos de un grafo

9 Grafos con costos
- Búsqueda por prioridad
- Árboles abarcadores de costo mínimo
- Caminos más cortos

Grafos con costos

- En muchas aplicaciones se desea elegir algunas aristas de un grafo para satisfacer algunas restricciones.
- El factor de decisión suele estar asociado con un cierto costo de las aristas.
- Este costo puede representar distancia, tiempo, beneficio, etc.
- Es muy fácil almacenar el costo en cualquiera de las representaciones vistas de grafos.

Ejemplo de grafo con costos

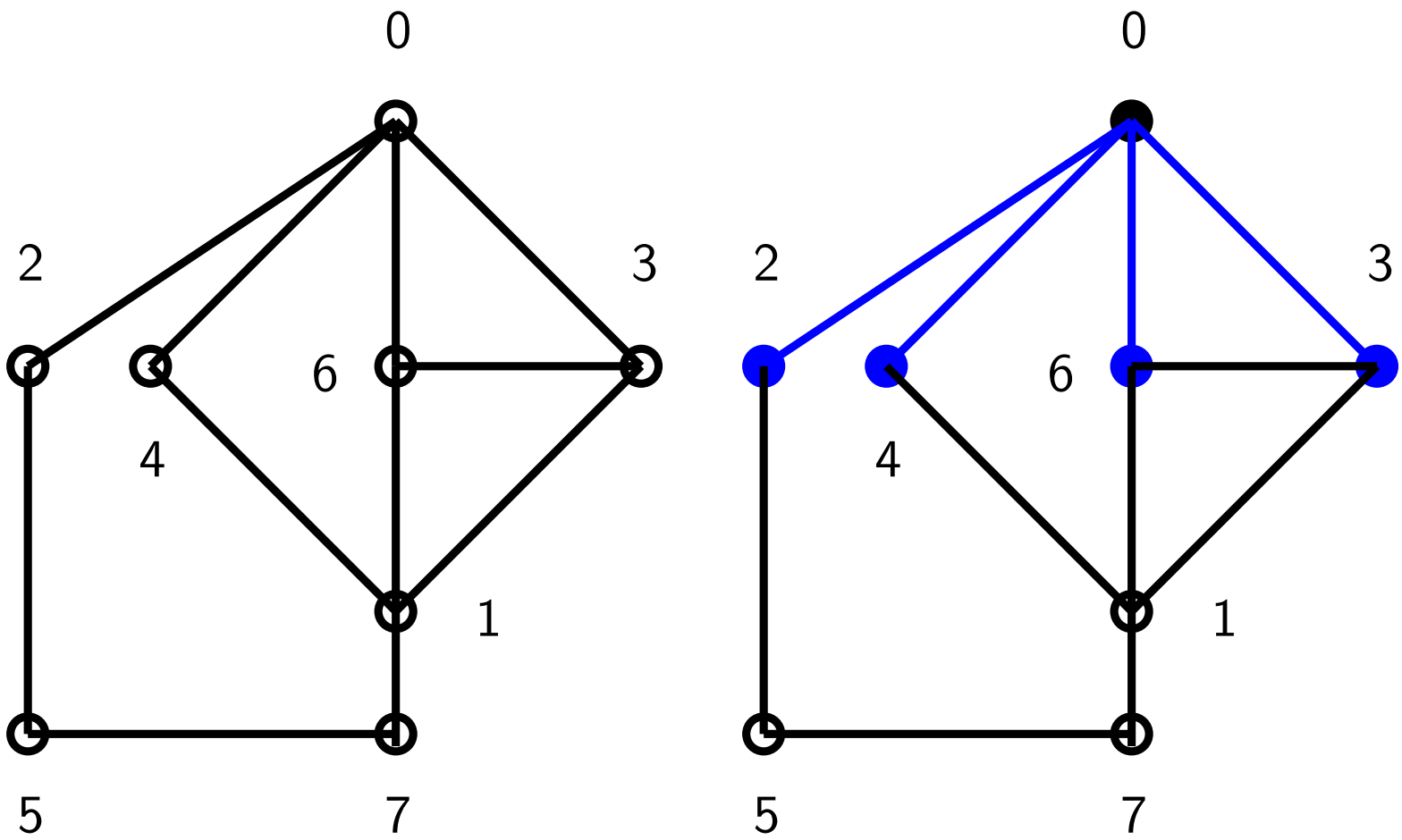


c	A	B	C	D	E	F	G	H
A	-	3	-	3	2	4	-	-
B	3	-	3	-	-	1	-	-
C	-	3	-	3	2	-	-	1
D	3	-	3	-	2	-	-	-
E	2	-	2	2	-	-	-	-
F	4	1	-	-	-	-	4	-
G	-	-	-	-	-	4	-	4
H	-	-	1	-	-	-	4	-

Búsqueda por prioridad

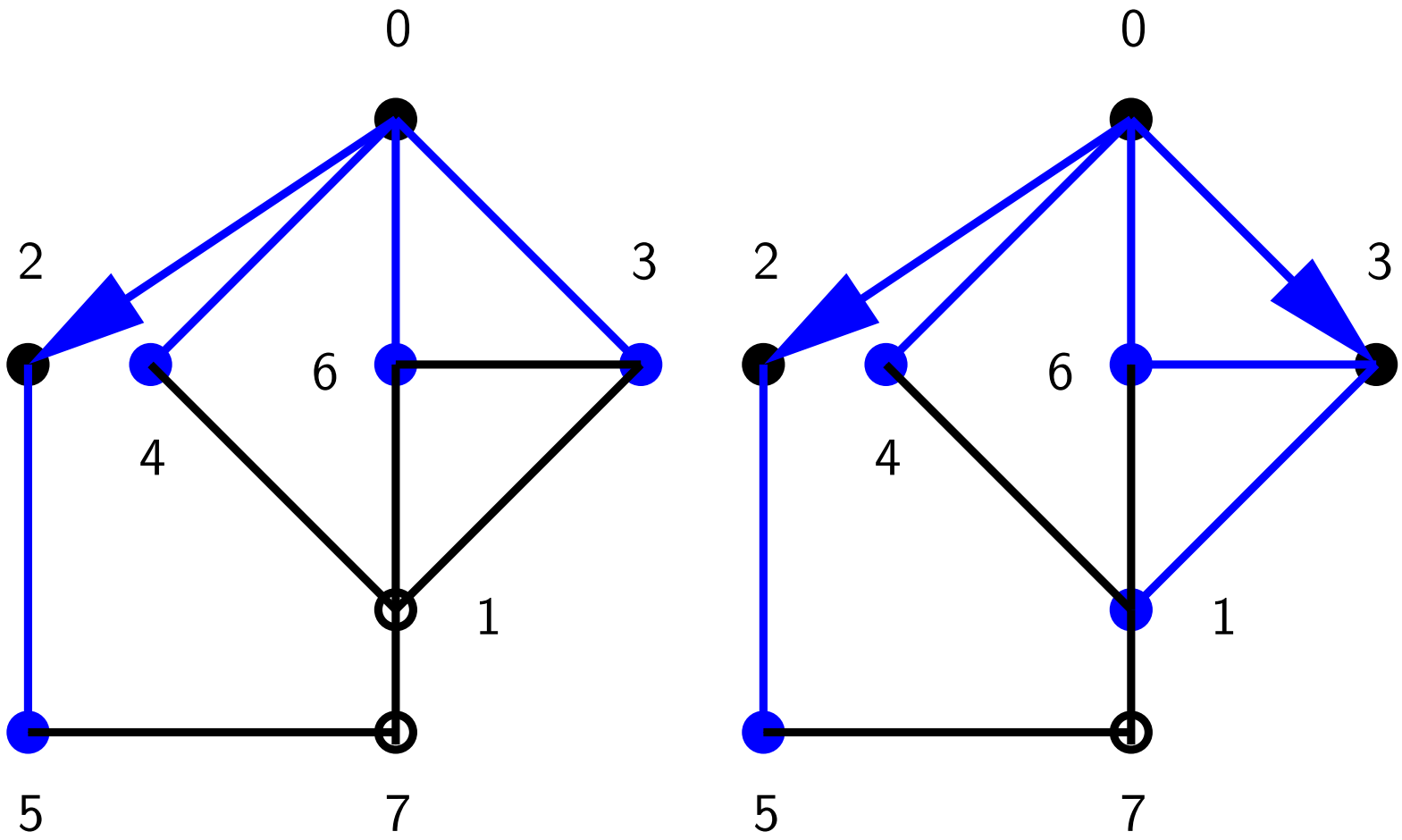
- Si en el método de búsqueda en profundidad no recursivo se reemplaza la pila por una cola de prioridad se obtiene un recorrido distinto.
- A éste se le llama búsqueda por prioridad.
- La prioridad se le asigna a los vértices.
- Recordemos que una cola de prioridad se puede implementar con un montículo.

Ejemplo de búsqueda por prioridad 1



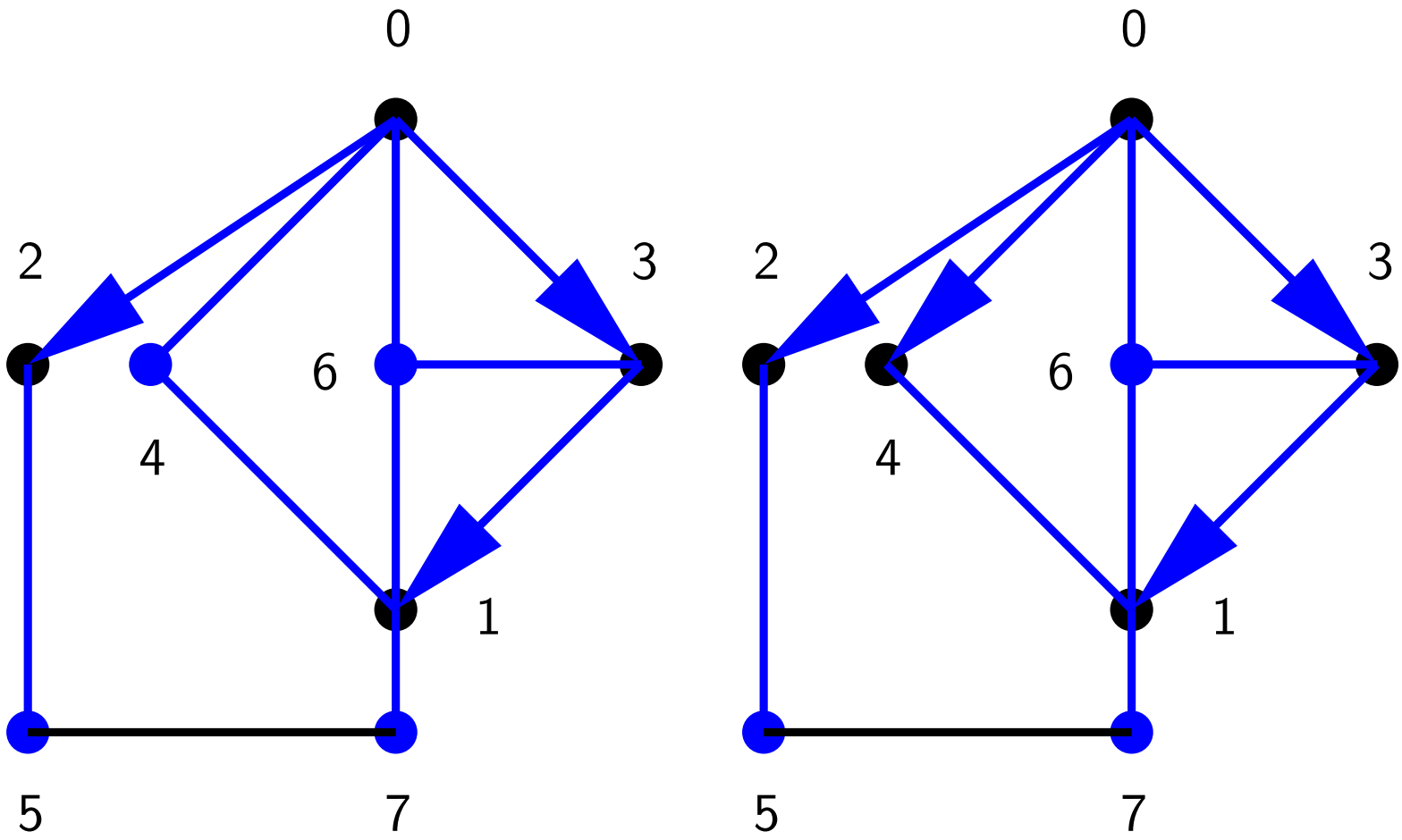
Cola de prioridad $[(0,-)] \rightarrow [(2,0),(3,0),(4,0),(6,0)]$.

Ejemplo de búsqueda por prioridad 2



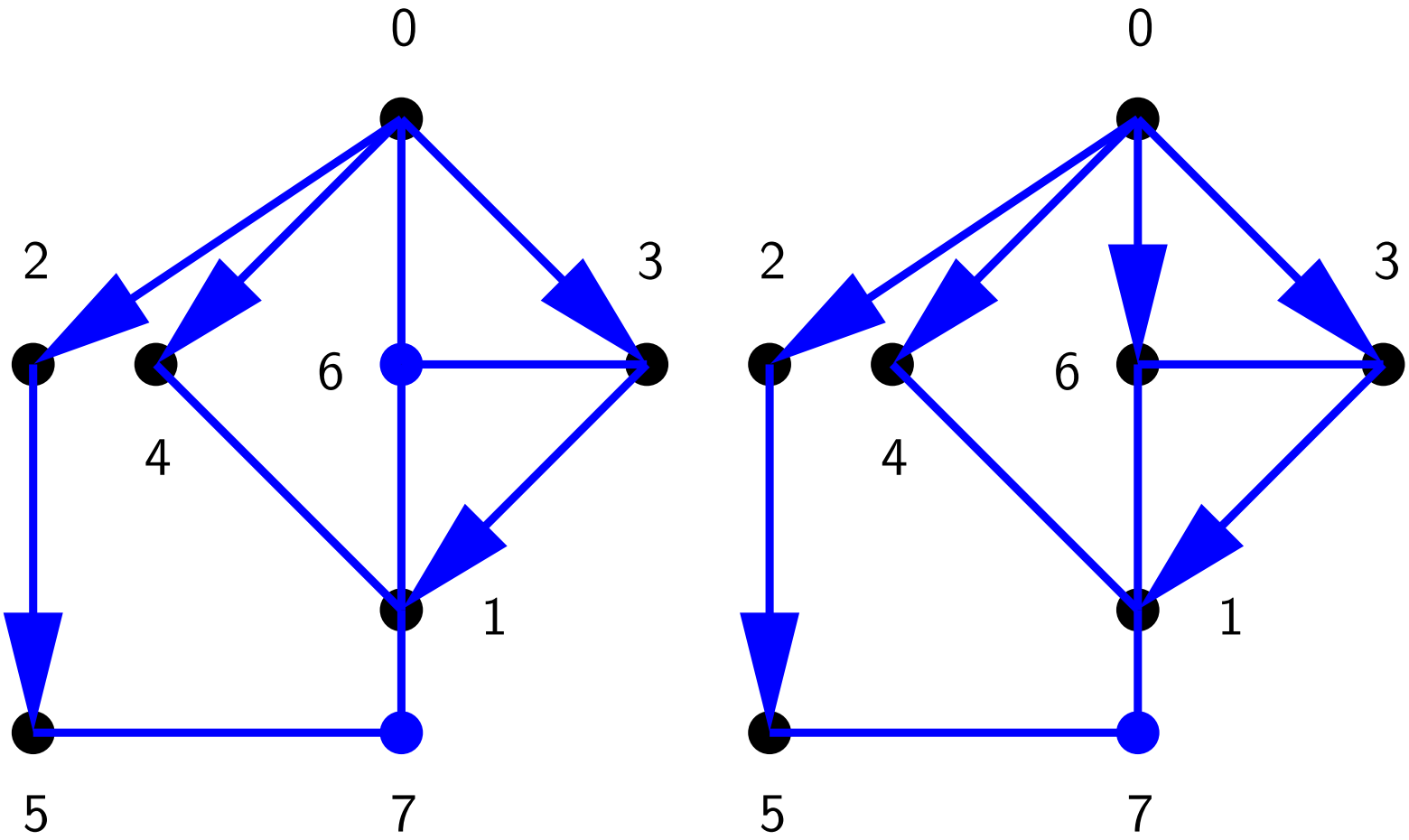
$[(2,0),(3,0),(4,0),(6,0)] \rightarrow [(3,0),(4,0),(5,2),(6,0)] \rightarrow$
$[(1,3),(4,0),(5,2),(6,0)]$.

Ejemplo de búsqueda por prioridad 3



$[(1,3),(4,0),(5,2),(6,0)] \rightarrow [(4,0),(5,2),(6,0),(7,1)] \rightarrow$
$[(5,2),(6,0),(7,1)]$.

Ejemplo de búsqueda por prioridad 4



Cola de prioridad $[(5,2),(6,0),(7,1)] \rightarrow [(6,0),(7,1)] \rightarrow [(7,1)]$.

Contenido

7 Representación de grafos y aplicaciones

8 Recorridos de un grafo

9 Grafos con costos
- Búsqueda por prioridad
- Árboles abarcadores de costo mínimo
- Caminos más cortos

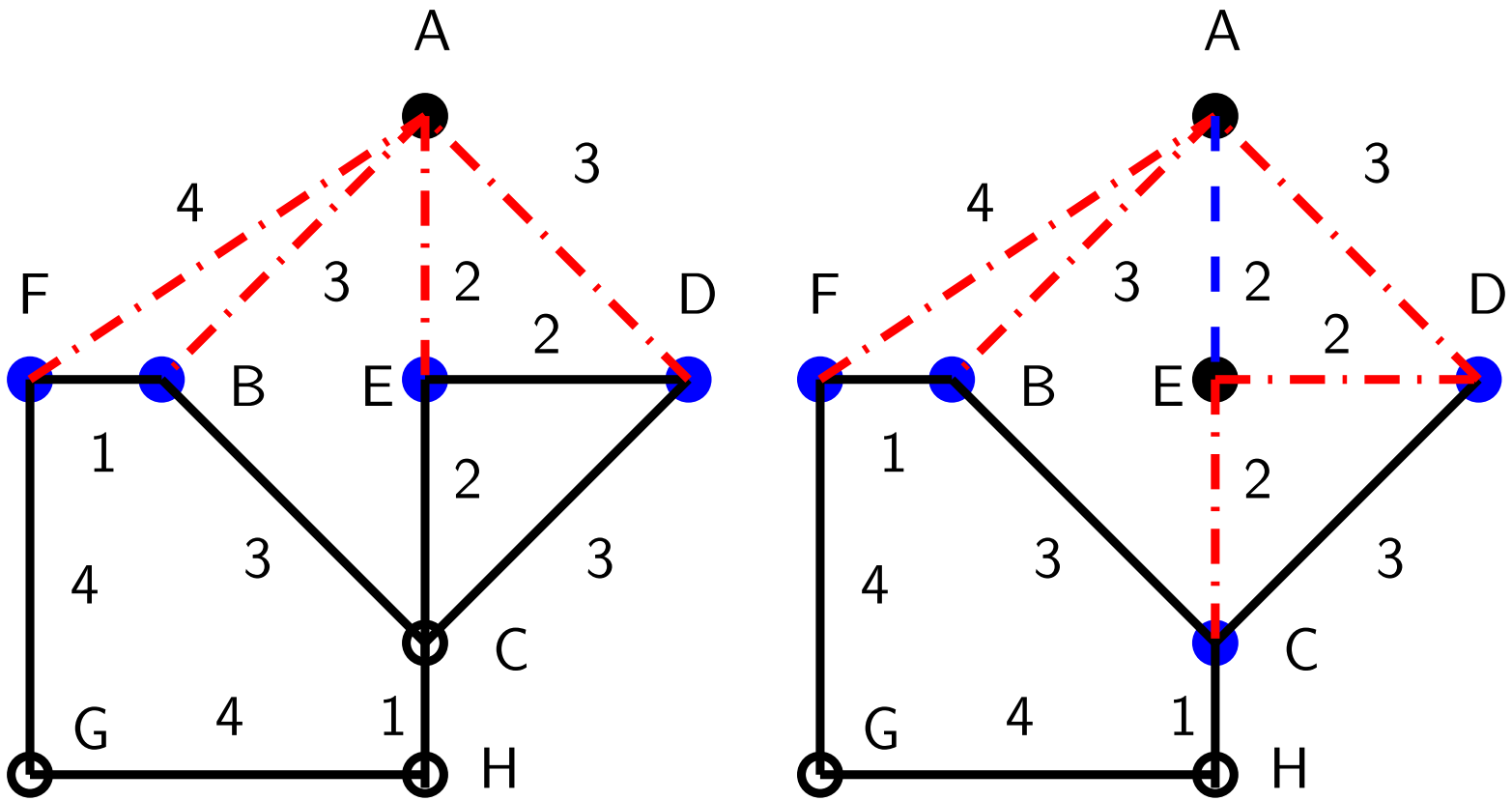
Árboles abarcadores mínimos

- Recordemos que un árbol abarcador es un árbol que usa todos los vértices de un grafo.
- Si las aristas tienen costos le asignaremos a cada árbol abarcador como costo la suma de los costos de las aristas que lo forman.
- Un grafo conexo suele tener más de un árbol abarcador. Nos interesa encontrar un árbol abarcador de costo mínimo.
- Aplicación: Conexión a costo mínimo.
- Dos algoritmos para resolver este problema.

Algoritmo de Prim

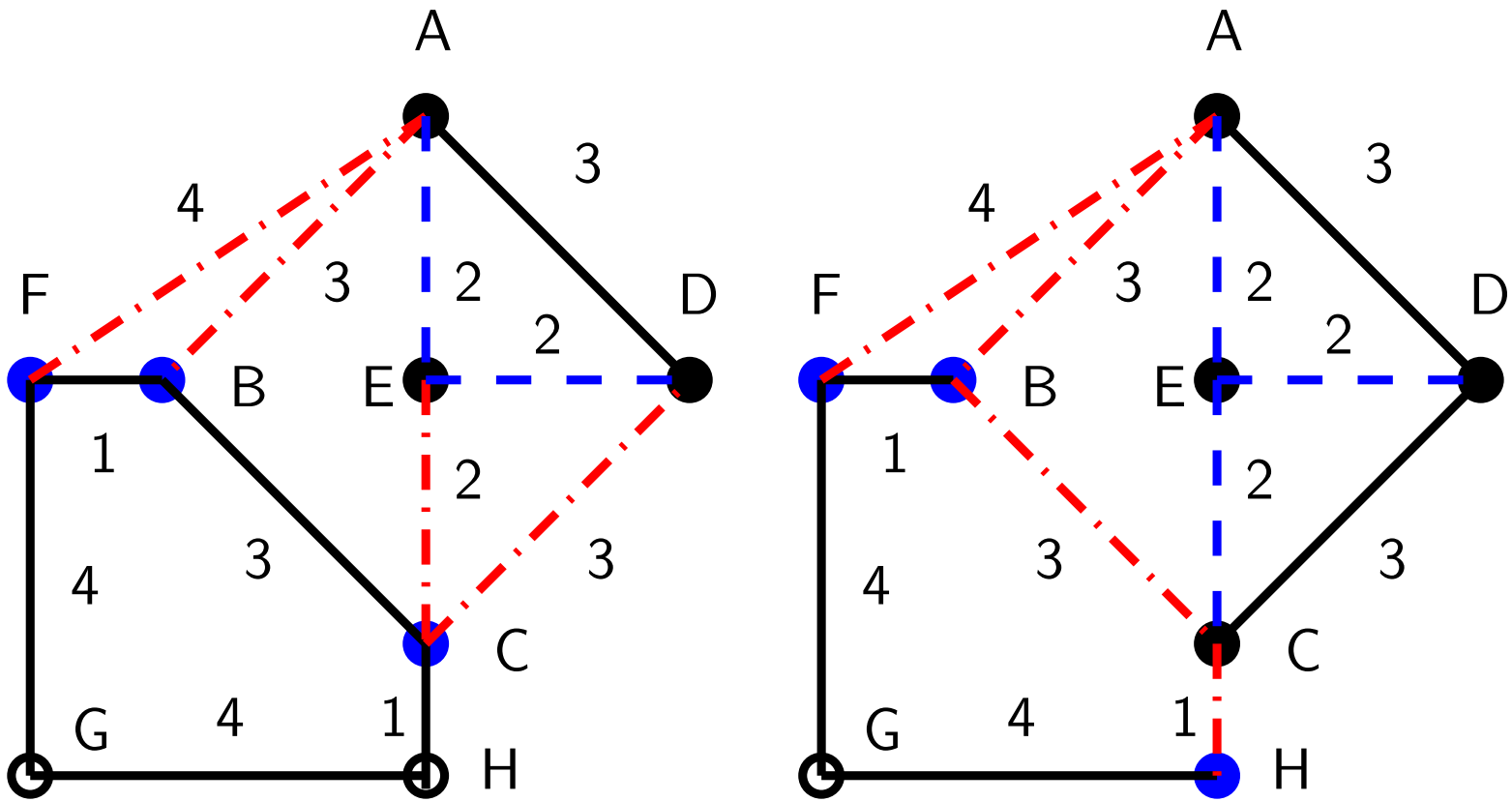
- El algoritmo de Prim construye un árbol abarcador de costo mínimo usando la búsqueda por prioridad.
- La prioridad de cada vértice viene dada por el costo mínimo de las aristas que lo unan a los vértices ya explorados.
- La prioridad de los vértices puede cambiar a lo largo de la ejecución del algoritmo.
- Esto necesita una estructura de datos que pueda actualizar la prioridad.

Ejemplo del algoritmo de Prim 1



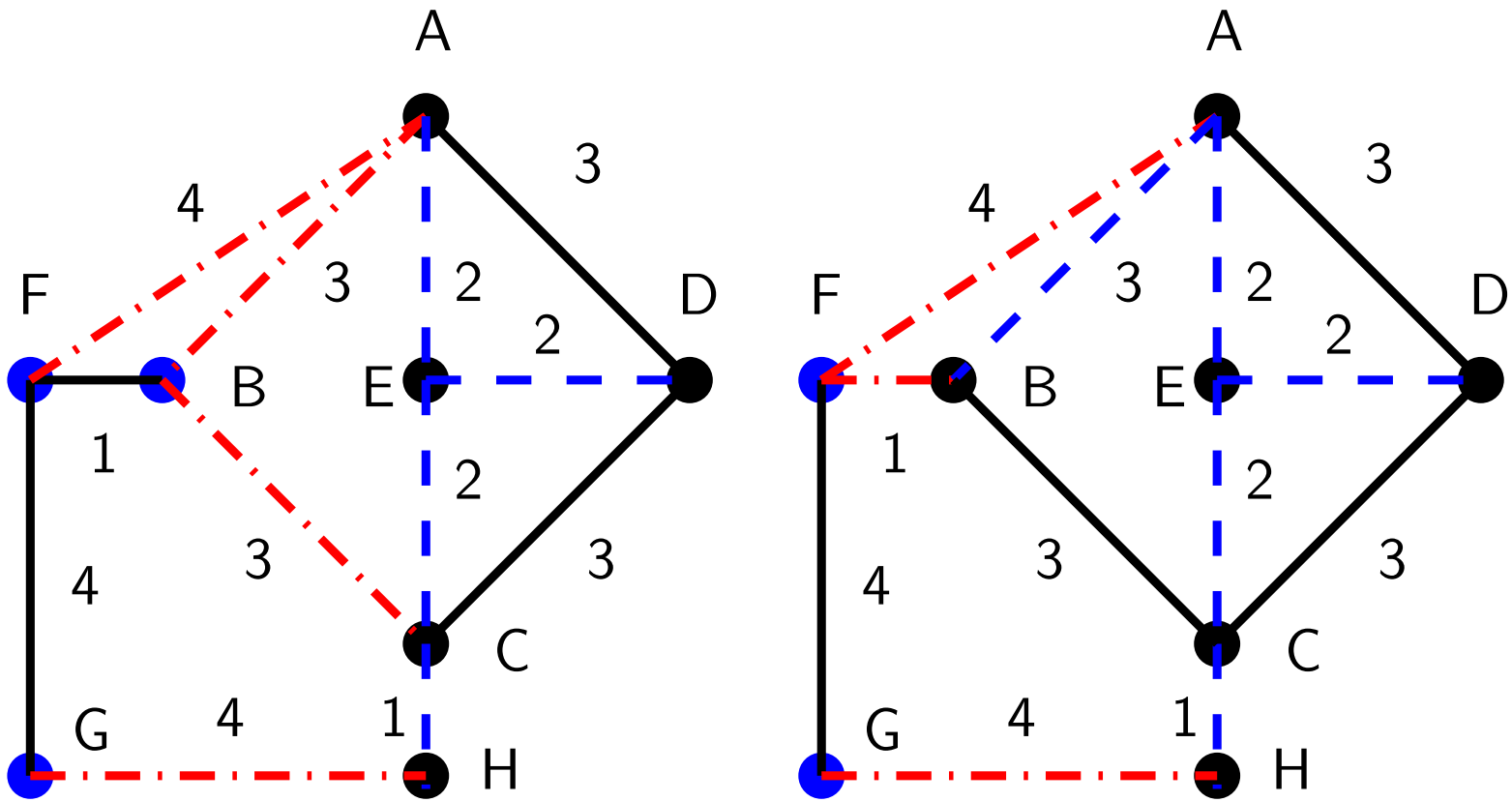
$[(A,-,0)] \rightarrow [(E,A,2),(B,A,3),(D,A,3),(F,A,4)] \rightarrow$
$[(D,E,2),(C,E,2),(B,A,3),(F,A,4)]$.

Ejemplo del algoritmo de Prim 2



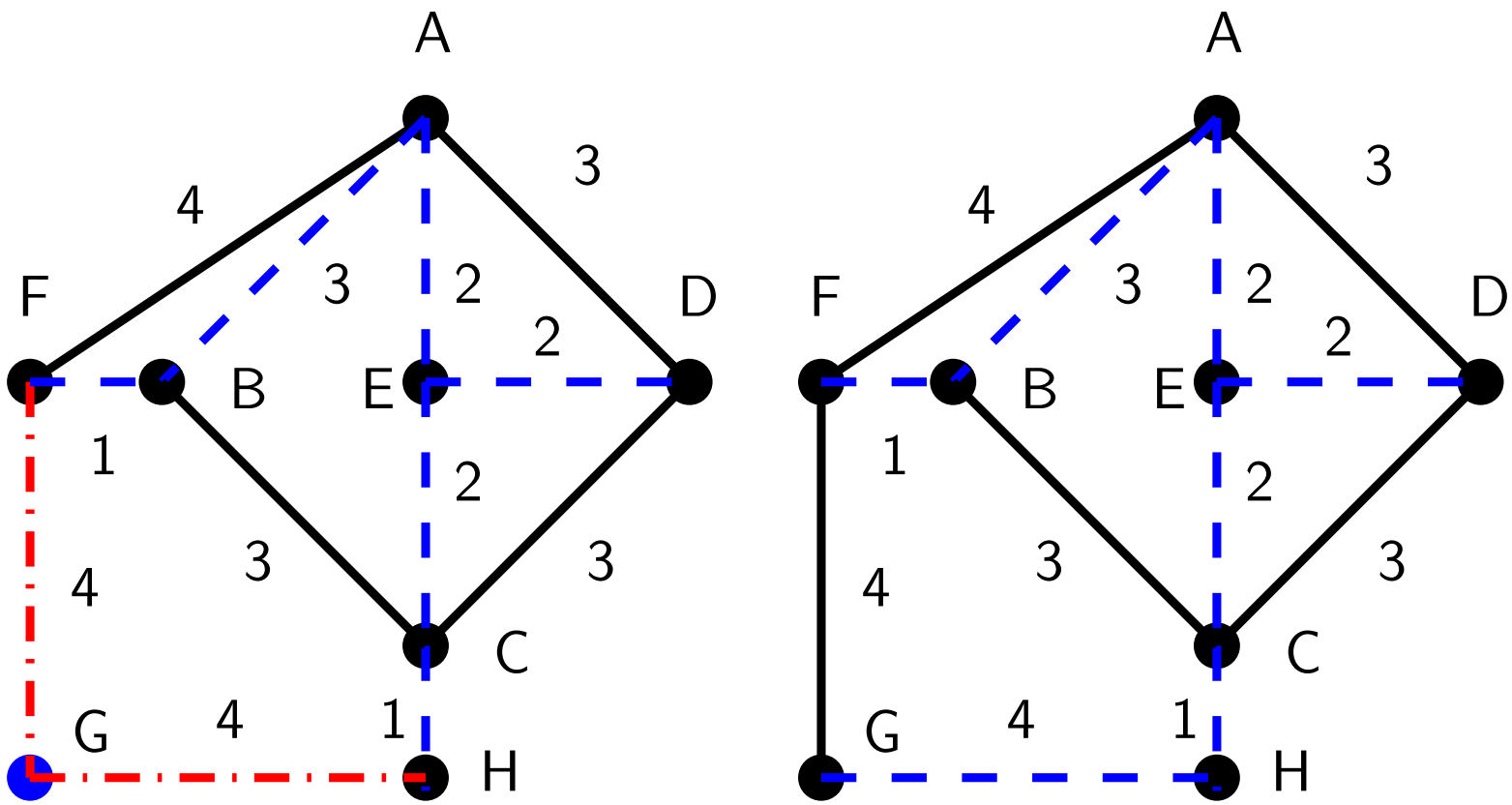
$[(D, E, 2), (C, E, 2), (B, A, 3), (F, A, 4)] \rightarrow$
$[(C, E, 2), (B, A, 3), (F, A, 4)] \rightarrow [(H, C, 1), (B, A, 3), (F, A, 4)].$

Ejemplo del algoritmo de Prim 3



$[(H, C, 1), (B, A, 3), (F, A, 4)] \rightarrow [(B, A, 3), (F, A, 4), (G, H, 4)] \rightarrow [(F, B, 1), (G, H, 4)]$.

Ejemplo del algoritmo de Prim 4

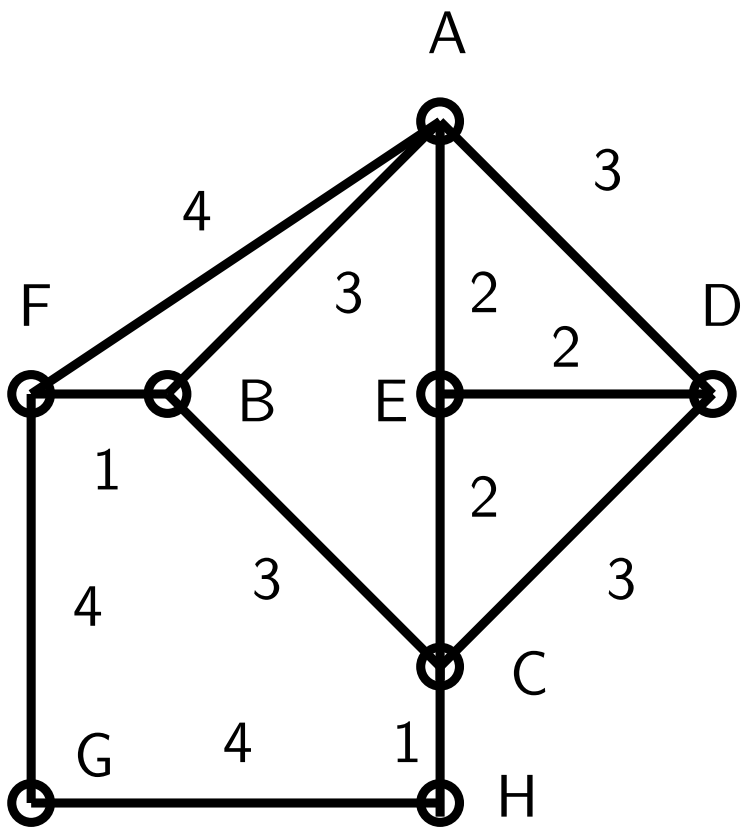


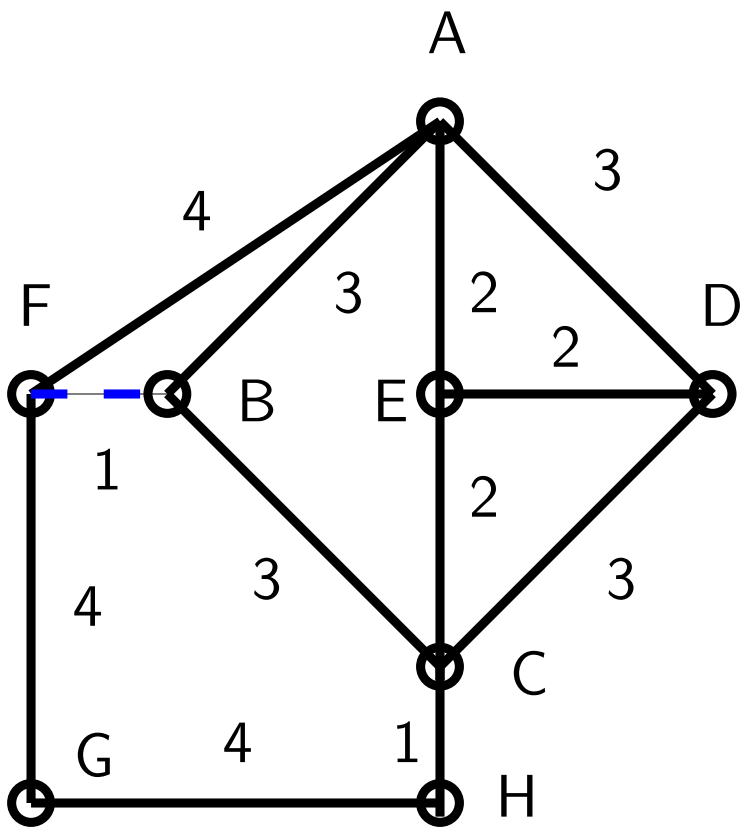
$[(F, B, 1), (G, H, 4)] \rightarrow [(G, H, 4)] \rightarrow []$.

Algoritmo de Kruskal

- El algoritmo de Kruskal construye un árbol abarcador mínimo con unión y pertenencia.
- Al principio cada vértice forma su propia componente conexa.
- Las aristas se ordenan crecientemente por costo y se consideran en ese orden.
- Si los dos vértices de una arista están en diferentes componentes conexas se hace la unión y se agrega la arista al árbol abarcador, en caso contrario se ignora.

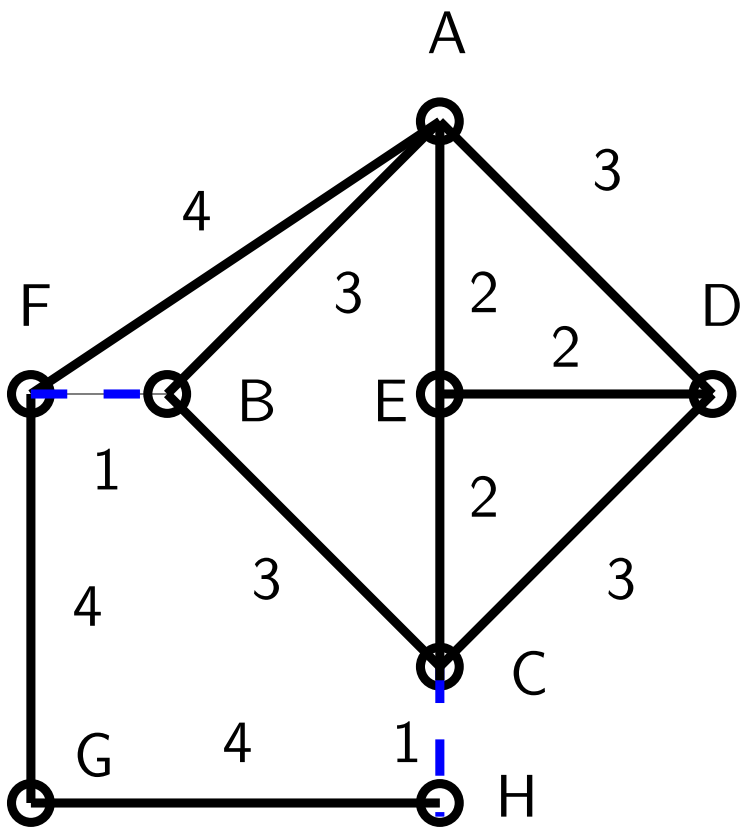
Ejemplo del algoritmo de Kruskal 1

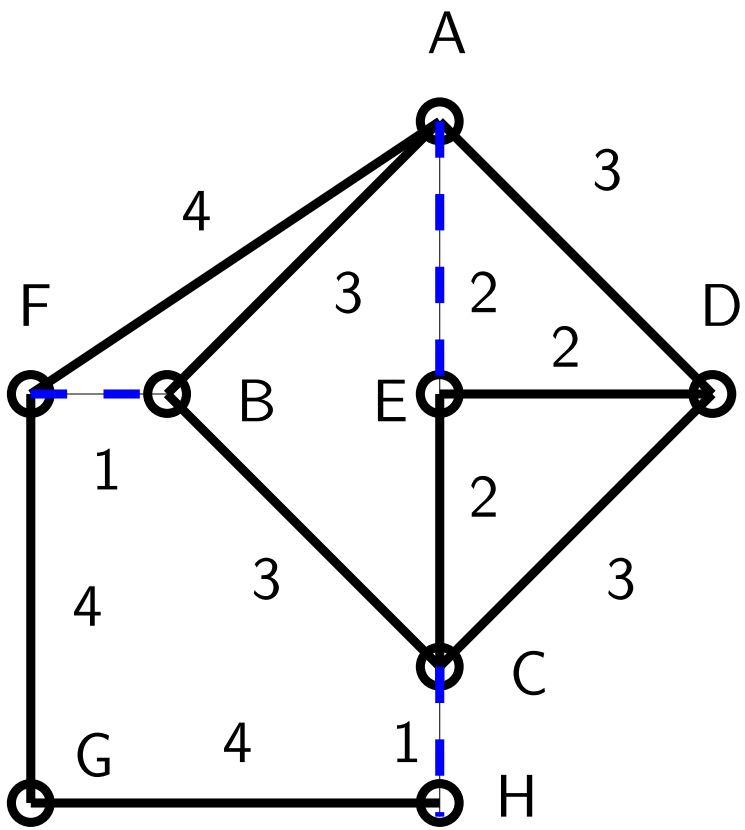




FB, CH, AE, ED, EC, AD, AB, BC, DC, FA, GH, FG.

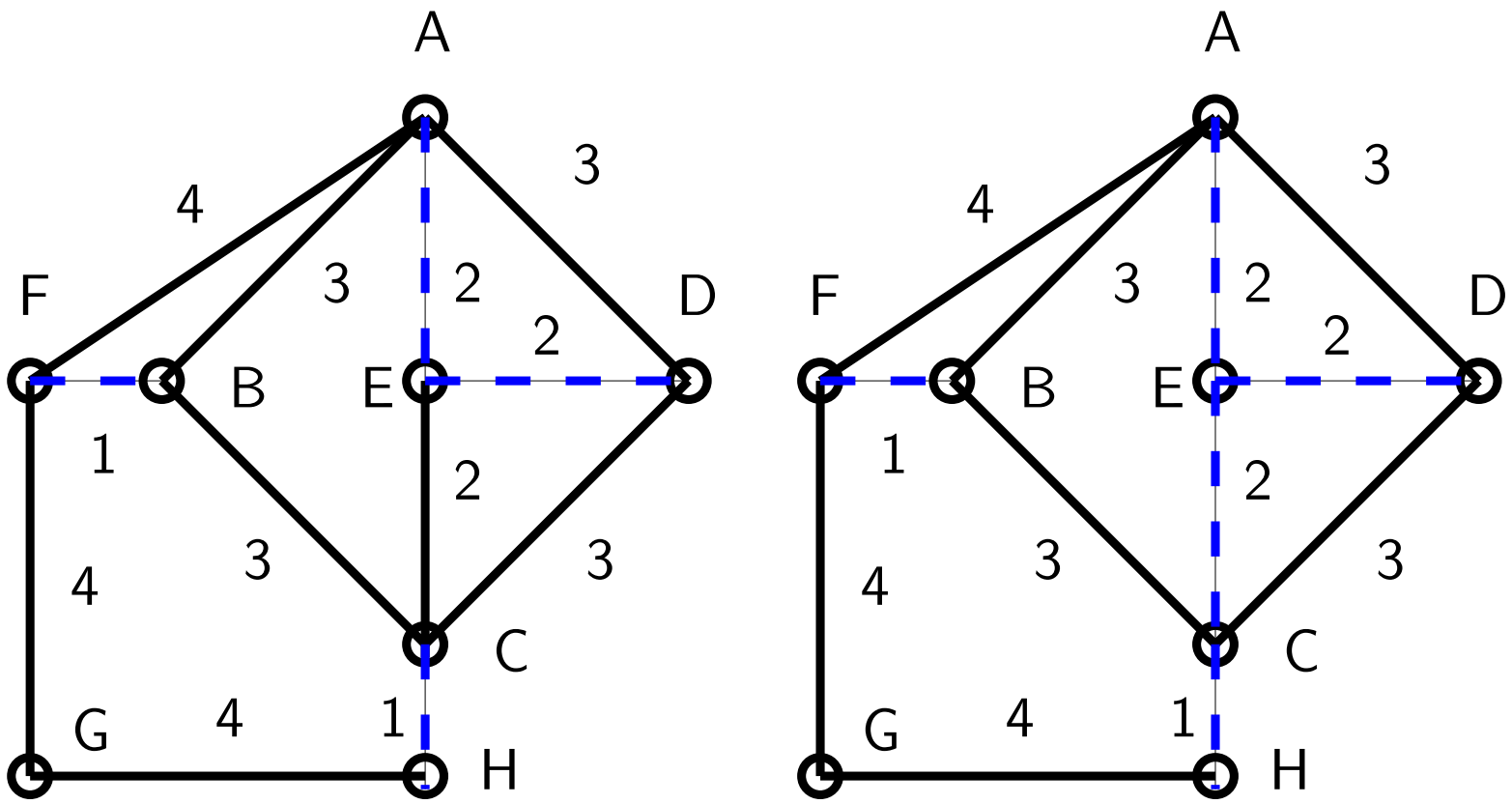
Ejemplo del algoritmo de Kruskal 2





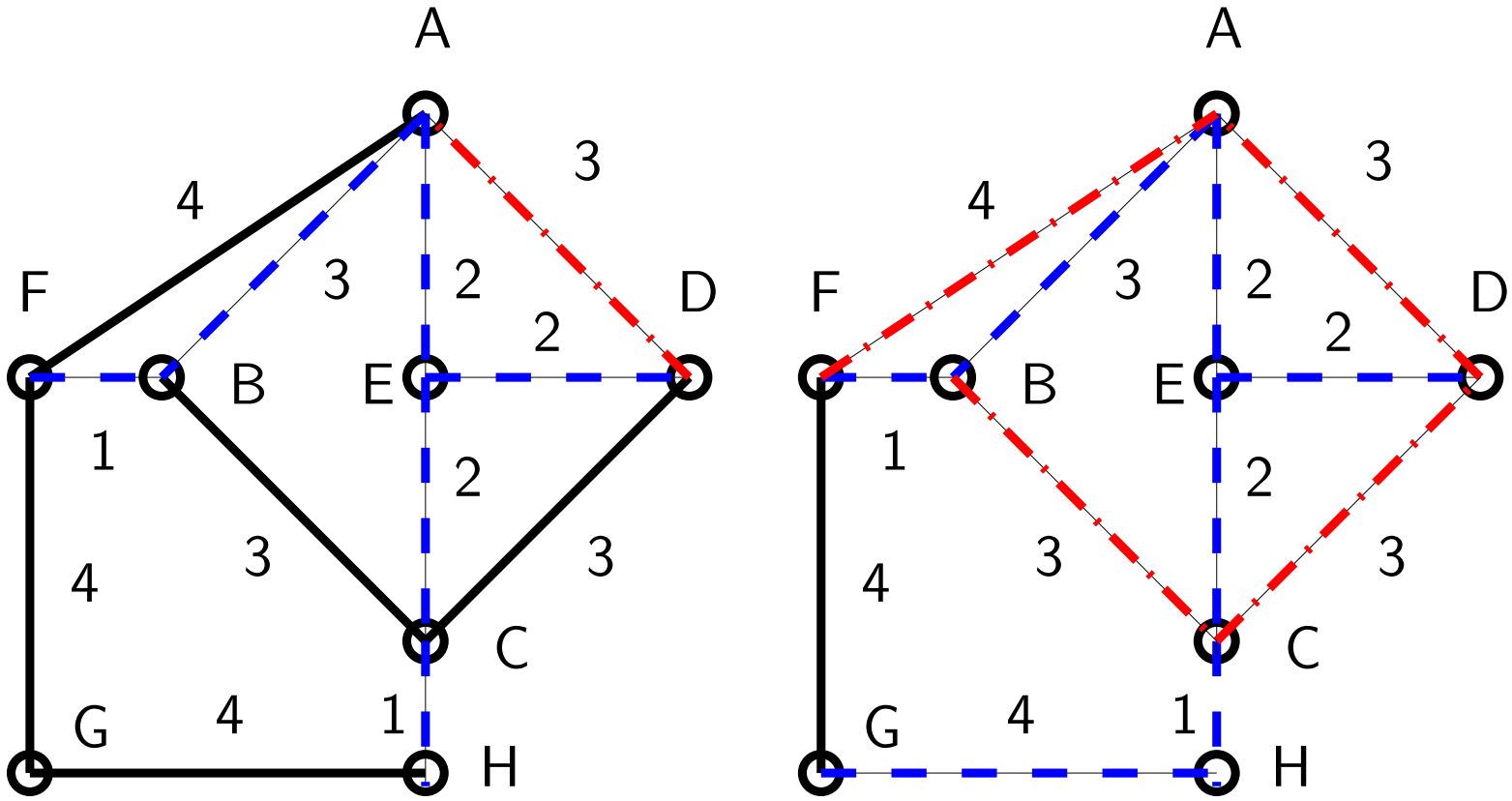
CH, AE, ED, EC, AD, AB, BC, DC, FA, GH, FG.

Ejemplo del algoritmo de Kruskal 3



ED, EC, AD, AB, BC, DC, FA, GH, FG.

Ejemplo del algoritmo de Kruskal 4



AB, BC, DC, FA, GH, FG.

Observaciones de ambos algoritmos

- En ambos algoritmos puede haber empates.
- Los empates que ocurran se pueden resolver de cualquier forma y producirán diversos árboles abarcadores mínimos.
- El algoritmo de Prim se ejecuta en tiempo $\propto (m+n)\log n$.
- El algoritmo de Kruskal se ejecuta en tiempo $\propto m\log m + n\log n$.
- Ambos algoritmos pueden encontrar árboles abarcadores máximos.

Contenido

7 Representación de grafos y aplicaciones

8 Recorridos de un grafo

9 Grafos con costos
- Búsqueda por prioridad
- Árboles abarcadores de costo mínimo
- Caminos más cortos

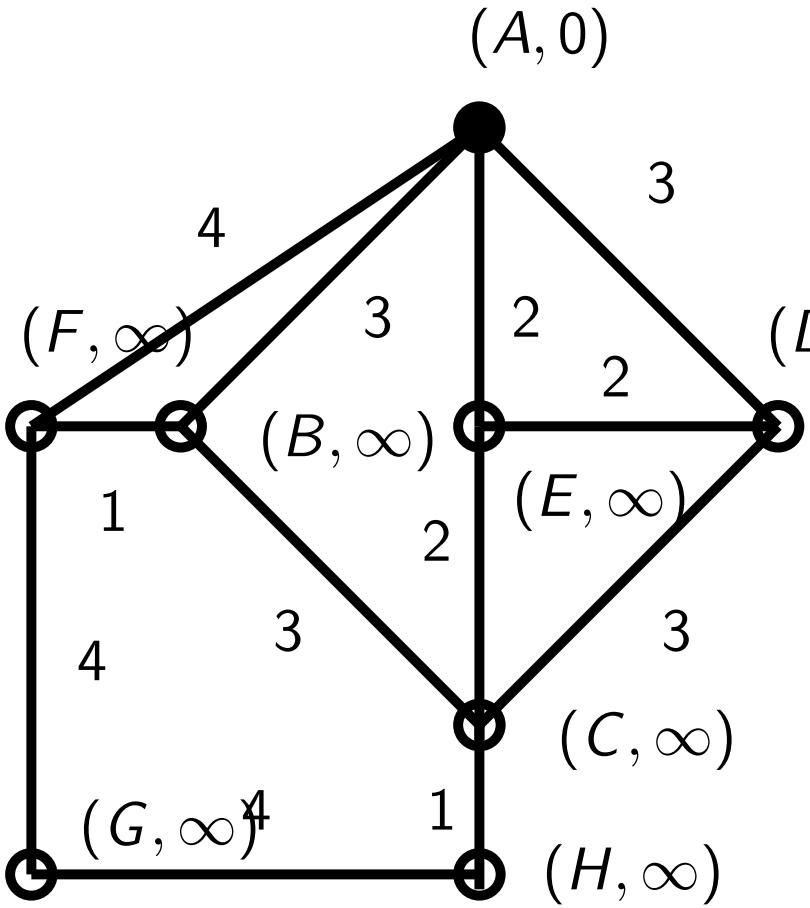
Caminos más cortos

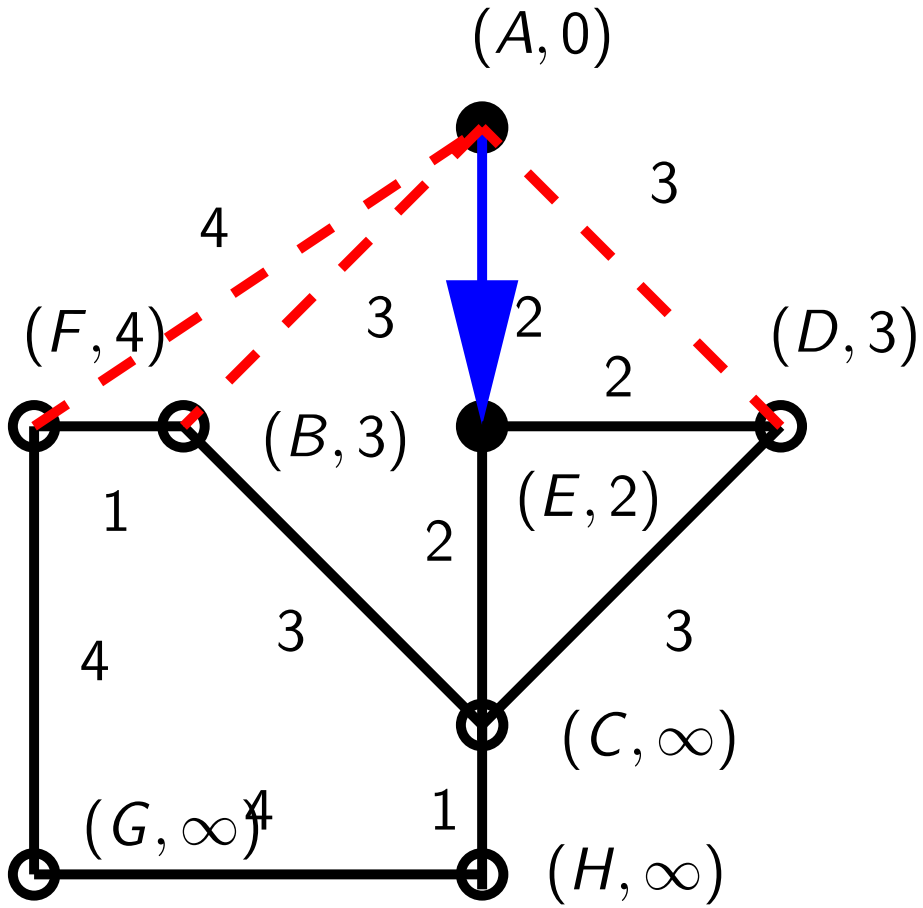
- Si un grafo es conexo entonces todos sus vértices están conectadas por caminos.
- Es probable que cada pareja de vértices esté conectada por más de un camino.
- Si a cada camino le asignamos un costo igual a la suma de los costos de las aristas que lo forman nos interesa encontrar un camino de costo mínimo o camino más corto.
- Si todos los costos son iguales a 1 entonces ya sabemos como resolver este problema.

Algoritmo de Dijkstra

- El algoritmo de Dijkstra genera los caminos más cortos desde un vértice inicial a todos los demás usando la búsqueda por prioridad.
- La prioridad de cada vértice es el costo mínimo de un camino a él desde el vértice inicial usando sólo los vértices ya explorados.
- Al principio el vértice inicial tiene prioridad 0 y todos los demás tienen prioridad $+\infty$.
- Cada que se explora un vértice se actualizan las prioridades de sus vecinos no explorados.

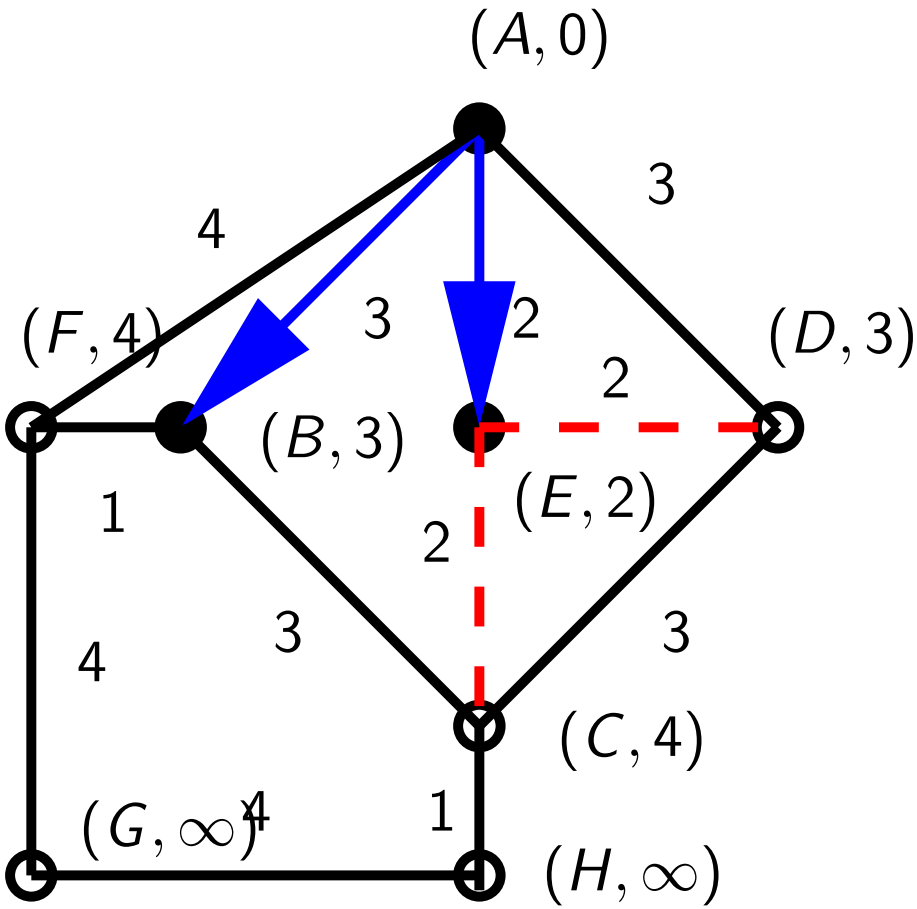
Ejemplo del algoritmo de Dijkstra 1

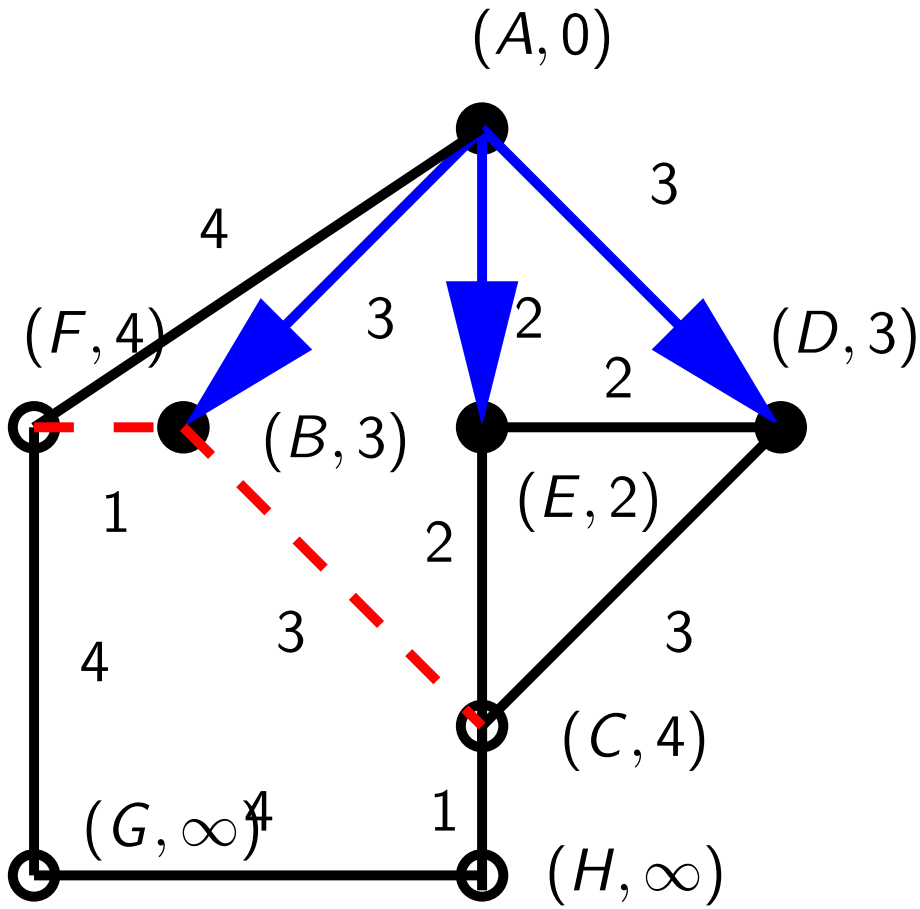




$[(A,-,0)] \rightarrow [(E,A,2),(B,A,3),(D,A,3),(F,A,4)] \rightarrow$
$[(B,A,3),(D,A,3),(C,E,4),(F,A,4)]$.

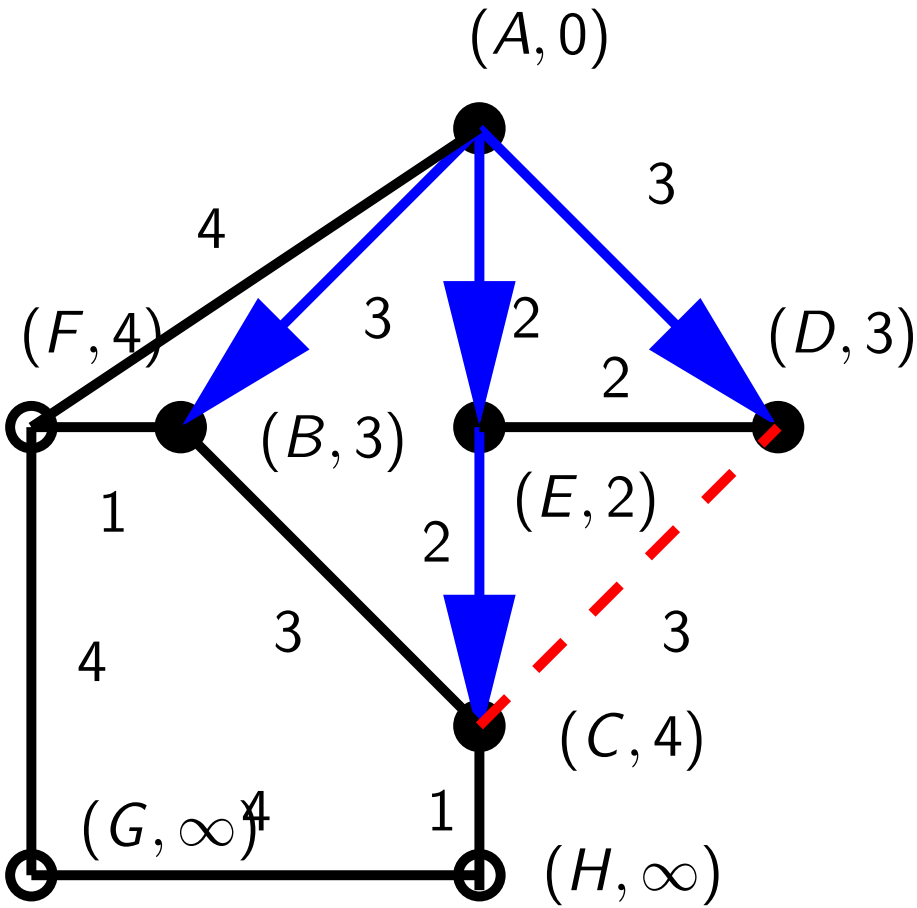
Ejemplo del algoritmo de Dijkstra 2

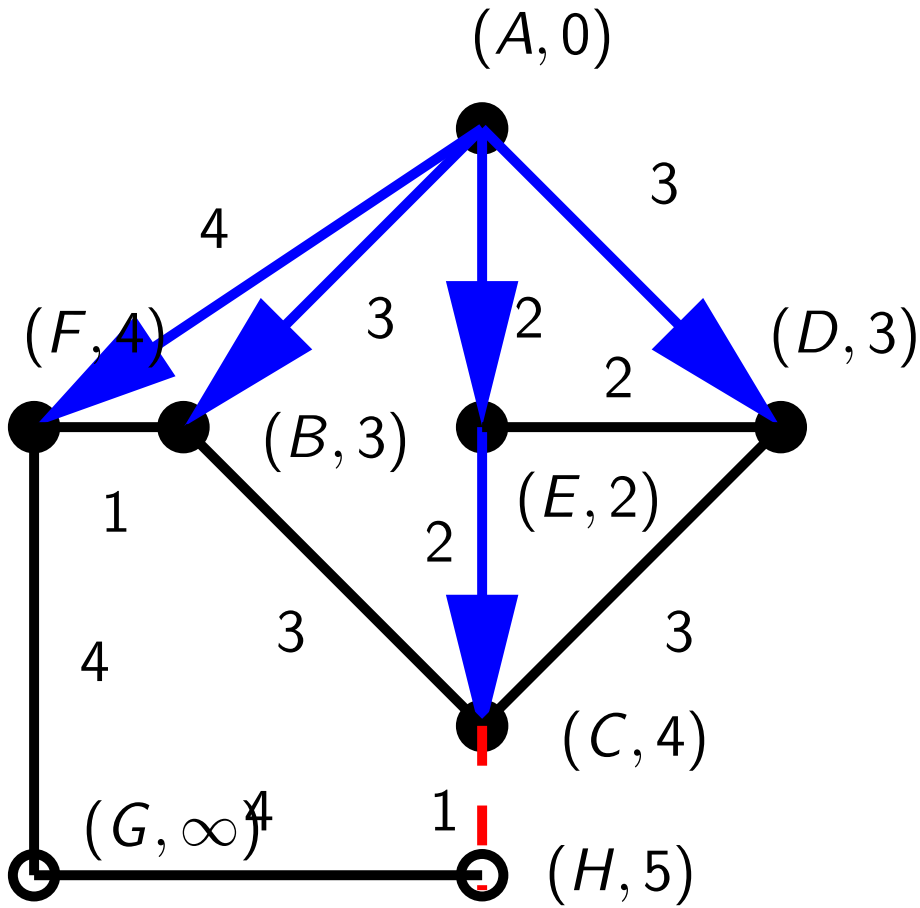




$[(B, A, 3), (D, A, 3), (C, E, 4), (F, A, 4)] \rightarrow$
$[(D, A, 3), (C, E, 4), (F, A, 4)] \rightarrow [(C, E, 4), (F, A, 4)].$

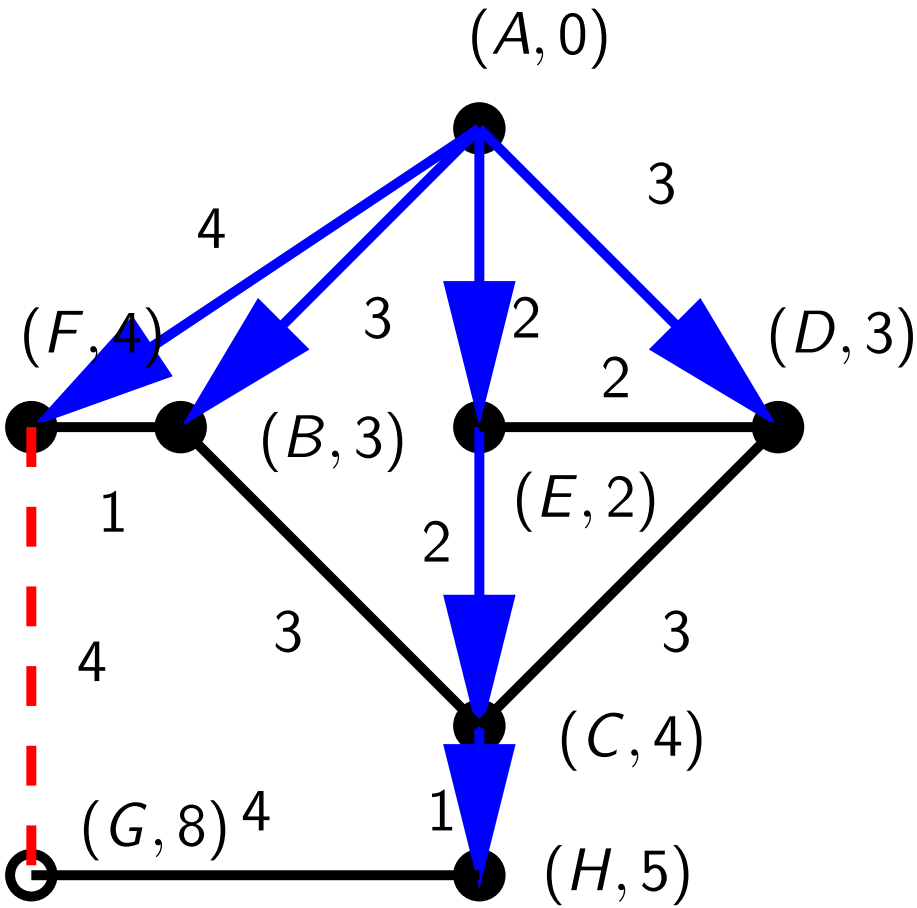
Ejemplo del algoritmo de Dijkstra 3

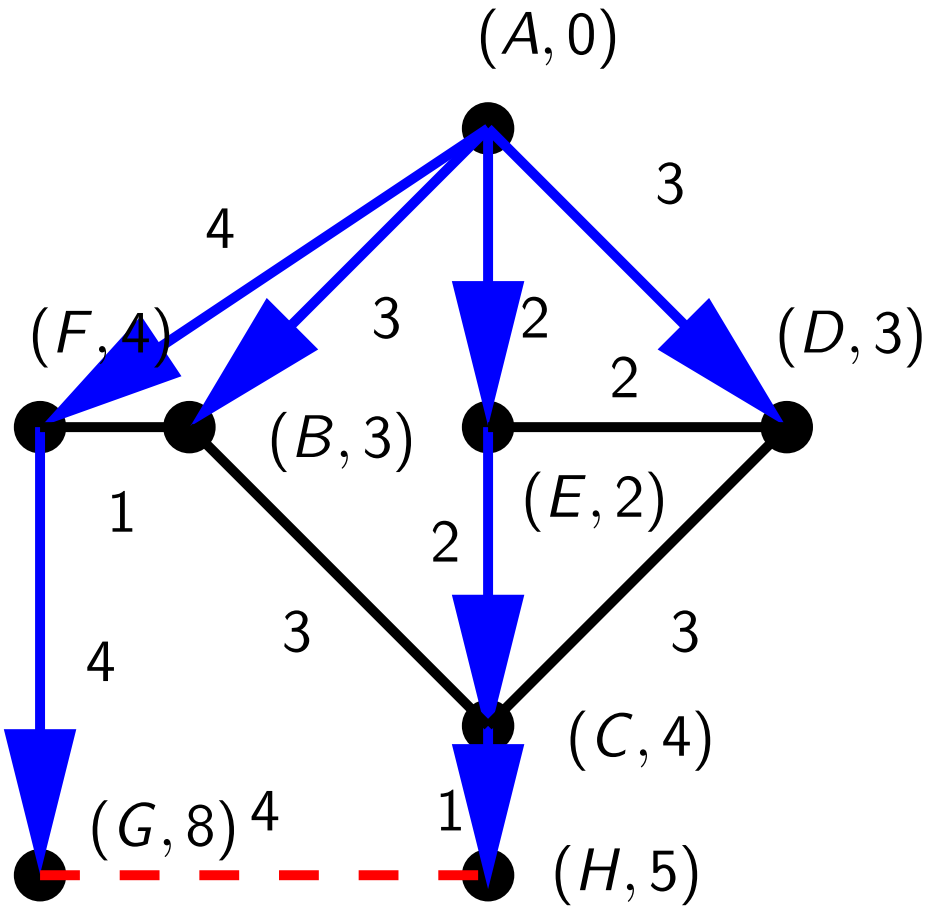




$[(C, E, 4), (F, A, 4)] \rightarrow [(F, A, 4), (H, C, 5)] \rightarrow [(H, C, 5), (G, F, 8)].$

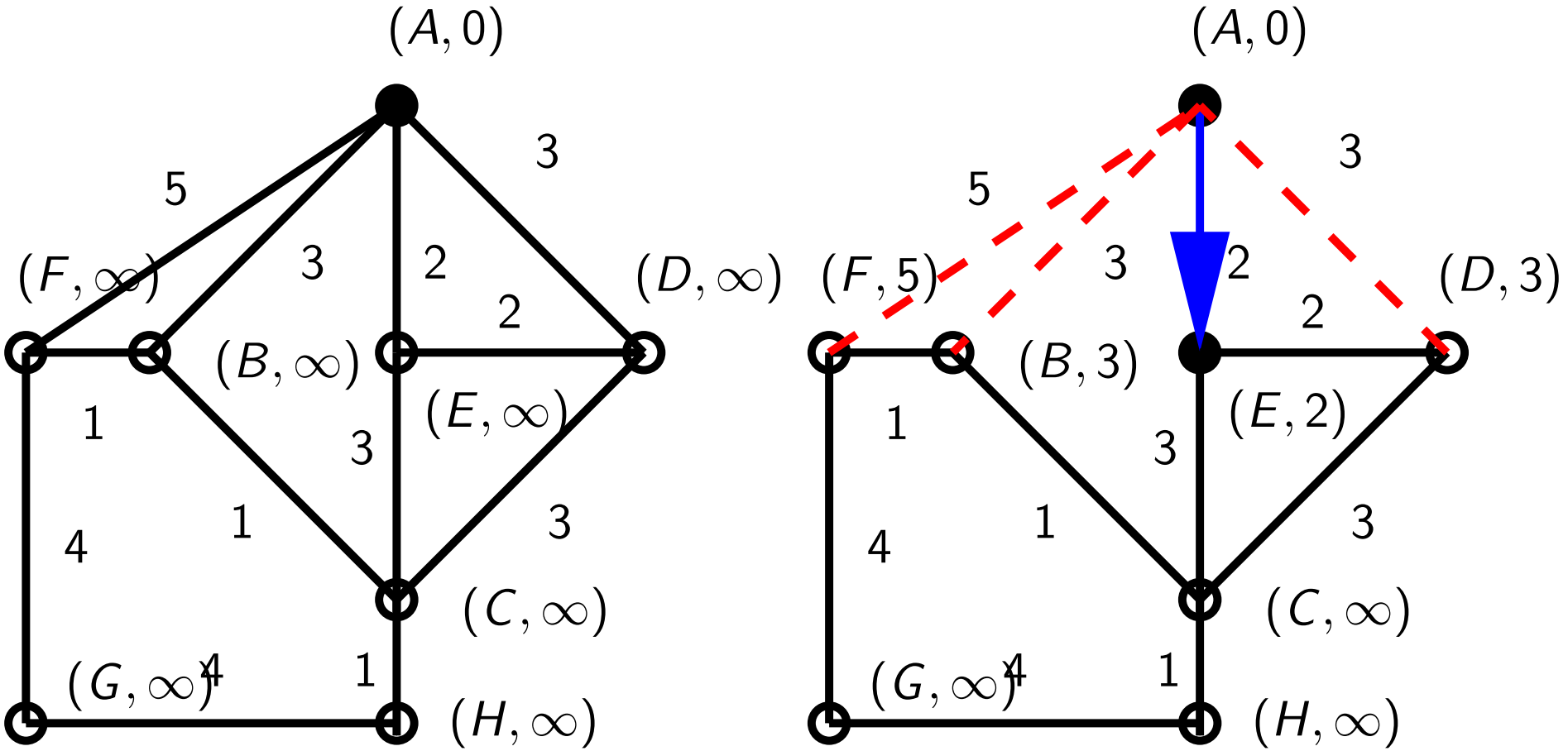
Ejemplo del algoritmo de Dijkstra 4





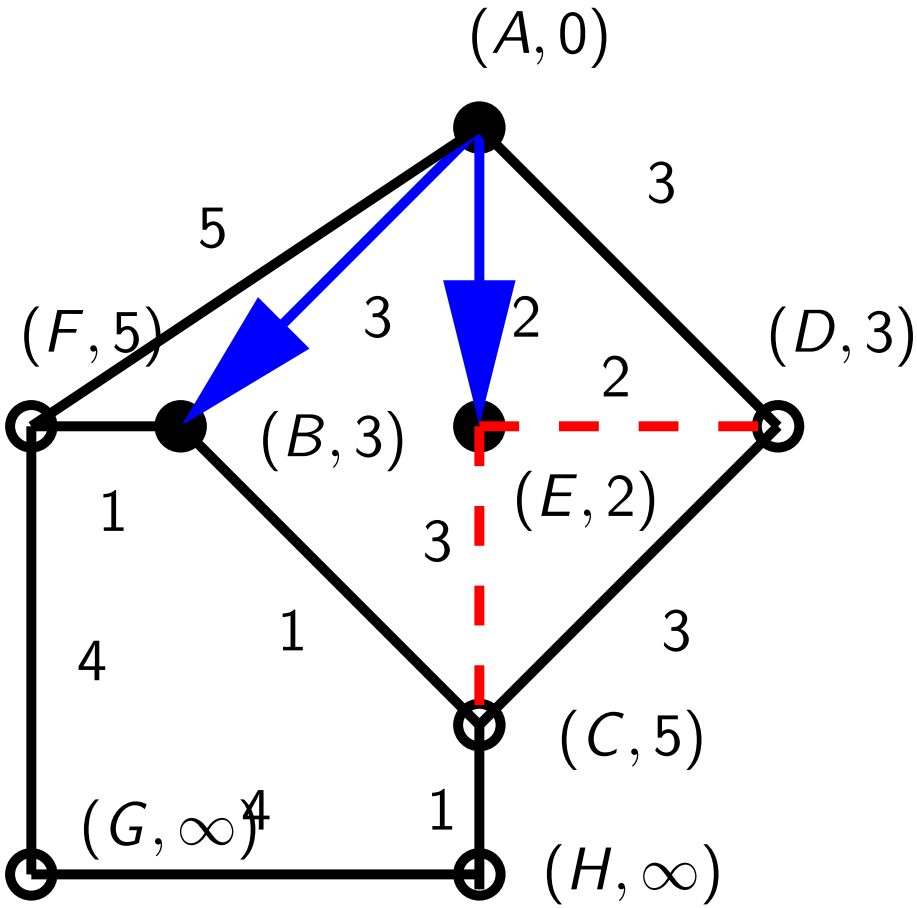
$[(H, C, 5), (G, F, 8)] \rightarrow [(G, F, 8)] \rightarrow [].$

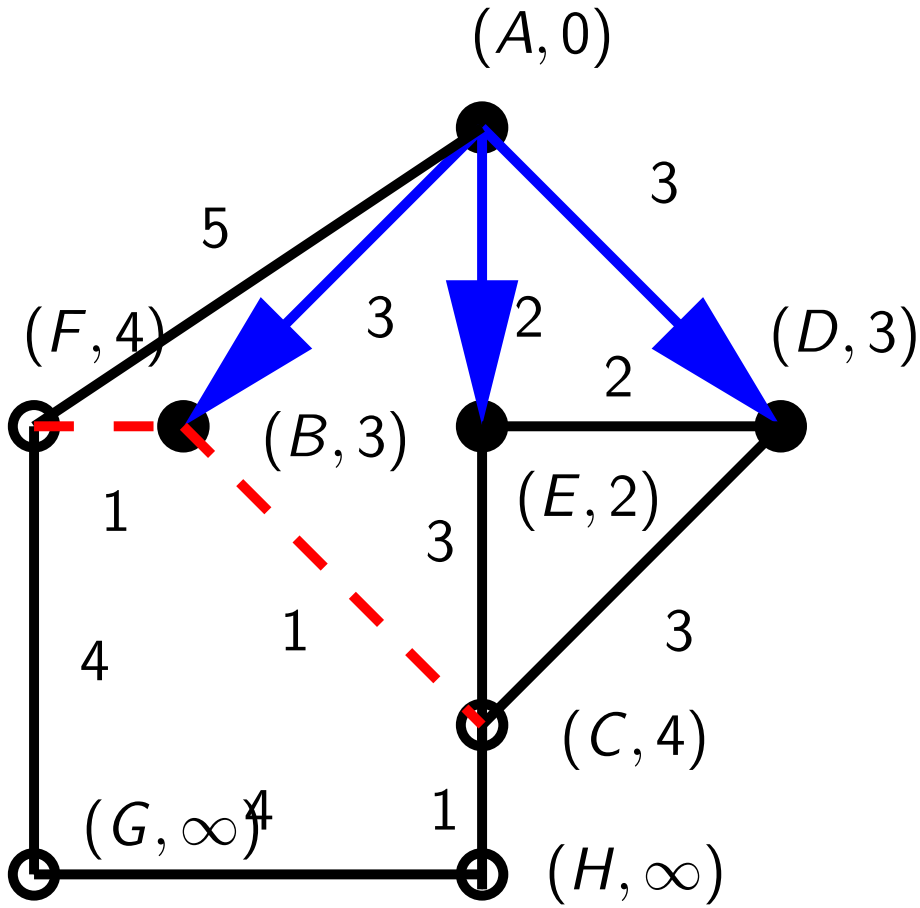
Otro ejemplo de Dijkstra 1



$[(A,-,0)] \rightarrow [(E,A,2),(B,A,3),(D,A,3),(F,A,5)] \rightarrow$
$[(B,A,3),(D,A,3),(C,E,5),(F,A,5)]$.

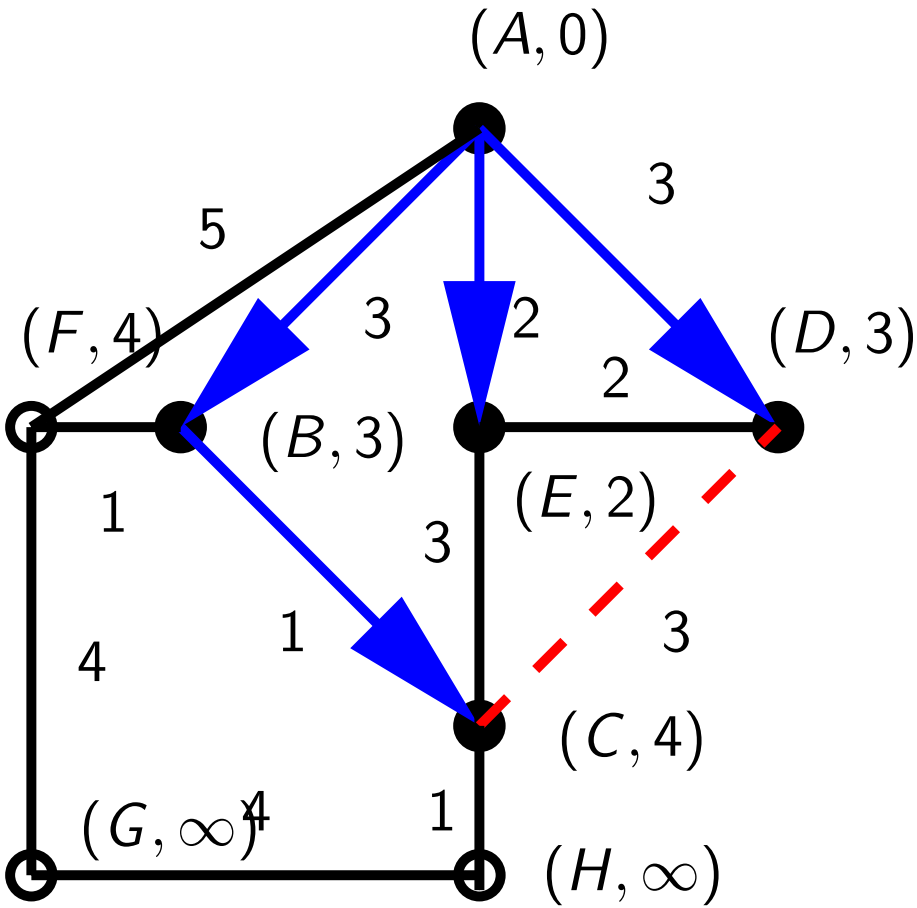
Otro ejemplo de Dijkstra 2

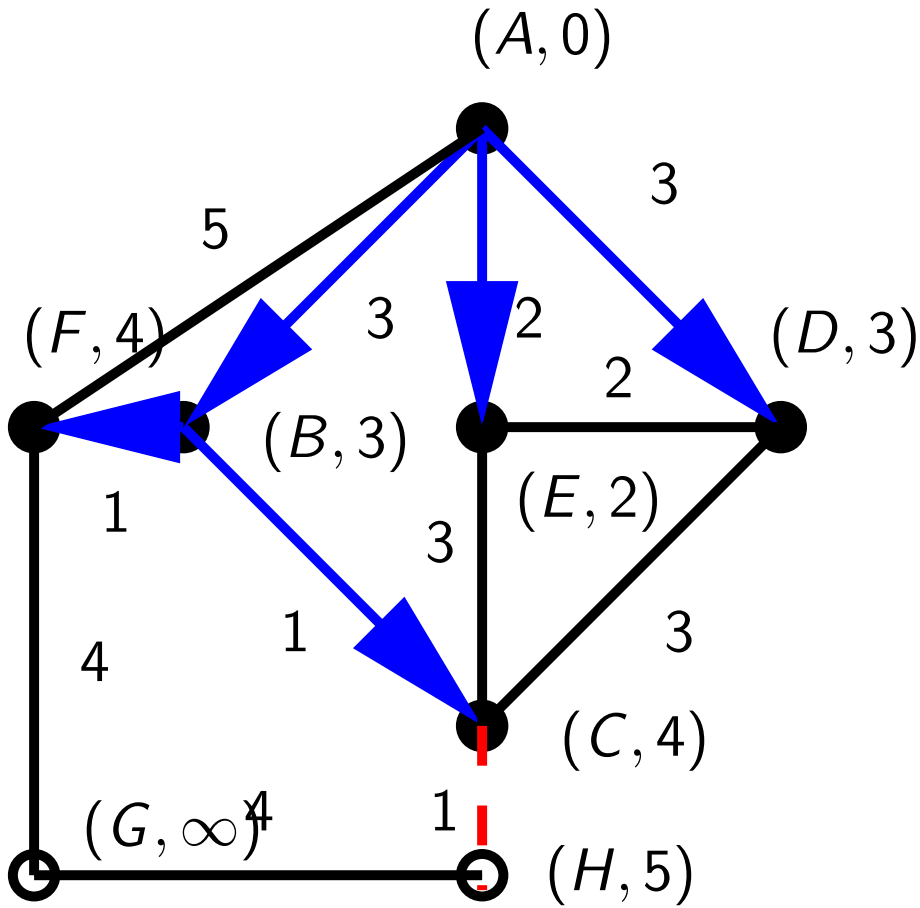




$[(B, A, 3), (D, A, 3), (C, E, 5), (F, A, 5)] \rightarrow$
$[(D, A, 3), (C, B, 4), (F, B, 4)] \rightarrow [(C, B, 4), (F, B, 4)].$

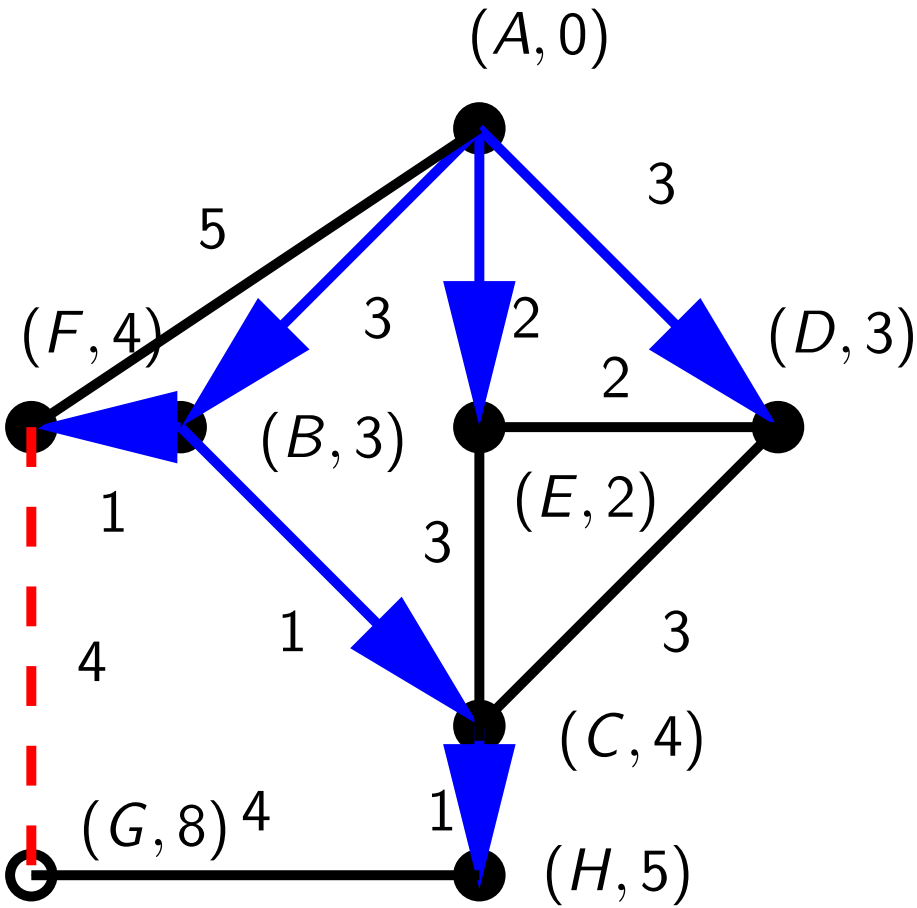
Otro ejemplo de Dijkstra 3

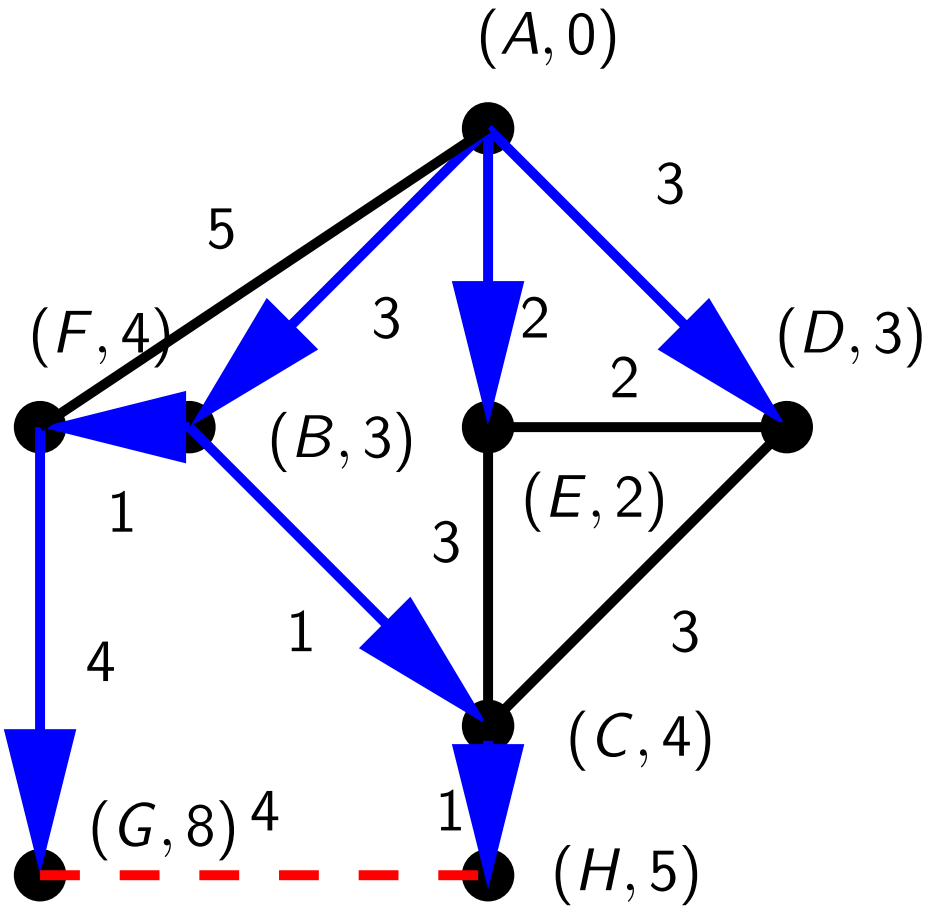




$[(C, B, 4), (F, B, 4)] \rightarrow [(F, B, 4), (H, C, 5)] \rightarrow [(H, C, 5), (G, F, 8)].$

Otro ejemplo de Dijkstra 4





$[(H, C, 5), (G, F, 8)] \rightarrow [(G, F, 8)] \rightarrow [].$

Observaciones de Dijkstra

- El algoritmo de Dijkstra construye un árbol de caminos más cortos con raíz en el vértice inicial. Este árbol no es único.
- El algoritmo de Dijkstra se ejecuta en tiempo $\propto (m+n)\log n$ usando listas de adyacencia y en tiempo $\propto n^2$ usando matrices de adyacencia.

Todos los caminos más cortos

- A veces queremos encontrar las longitudes (o costos) de los caminos más cortos entre todas las parejas de vértices de un grafo.
- Podemos hacerlo usando el algoritmo de Dijkstra n veces, una desde cada vértice.
- Esto se ejecuta en tiempo $\propto n(m+n)\log n$ usando listas de adyacencia y en tiempo $\propto n^3$ usando matrices de adyacencia.
- Hay una forma más sencilla.

Algoritmo de Floyd

- Suponga que los vértices están numerados del 1 al n y queremos responder la siguiente pregunta:
- ¿Cuál es la longitud $a(u, v, i)$ del camino más corto del vértice u al v usando sólo vértices intermedios con números $\leq i$?
- Observe que $a(u, v, 0)$ es el costo de la arista uv si u y v son adyacentes o bien $+\infty$.
- Si $i > 0$ entonces $a(u, v, i)$ es el menor de $a(u, v, i-1)$ y $a(u, i, i-1) + a(i, v, i-1)$.
- ¿Porqué?

Implementación de Floyd

- Se comienza con una matriz de adyacencia a con los costos de las aristas (o números muy grandes donde no las haya).
- Se termina con las longitudes de los caminos más cortos.

```
for (i = 0; i < n; i++)
  for (u = 0; u < n; u++)
    for (v = 0; v < n; v++) {
      t = a[u][i] + a[i][v];
      if (t < a[u][v])
        a[u][v] = t;
    }
```

Ejemplo del algoritmo de Floyd 1

Paso 0

0	3	-	3	2	4	-	-
3	0	3	-	-	1	-	-
-	3	0	3	2	-	-	1
3	-	3	0	2	-	-	-
2	-	2	2	0	-	-	-
4	1	-	-	-	0	4	-
-	-	-	-	-	4	0	4
-	-	1	-	-	-	4	0

Paso 1

0	3	-	3	2	4	-	-
3	0	3	6	5	1	-	-
-	3	0	3	2	-	-	1
3	6	3	0	2	7	-	-
2	5	2	2	0	6	-	-
4	1	-	7	6	0	4	-
-	-	-	-	-	4	0	4
-	-	1	-	-	-	4	0

Paso 2

0	3	6	3	2	4	-	-
3	0	3	6	5	1	-	-
6	3	0	3	2	4	-	1
3	6	3	0	2	7	-	-
2	5	2	2	0	6	-	-
4	1	4	7	6	0	4	-
-	-	-	-	-	4	0	4
-	-	1	-	-	-	4	0

Ejemplo del algoritmo de Floyd 2

Paso 6

0	3	4	3	2	4	8	5
3	0	3	6	5	1	5	4
4	3	0	3	2	4	8	1
3	6	3	0	2	7	+	4
2	5	2	2	0	6	+	3
4	1	4	7	6	0	4	5
8	5	8	+	+	4	0	4
5	4	1	4	3	5	4	0

Paso 7

0	3	4	3	2	4	8	5
3	0	3	6	5	1	5	4
4	3	0	3	2	4	8	1
3	6	3	0	2	7	+	4
2	5	2	2	0	6	+	3
4	1	4	7	6	0	4	5
8	5	8	+	+	4	0	4
5	4	1	4	3	5	4	0

Paso 8

0	3	4	3	2	4	8	5
3	0	3	6	5	1	5	4
4	3	0	3	2	4	5	1
3	6	3	0	2	7	8	4
2	5	2	2	0	6	7	3
4	1	4	7	6	0	4	5
8	5	5	8	7	4	0	4
5	4	1	4	3	5	4	0

Part III

Árboles AVL

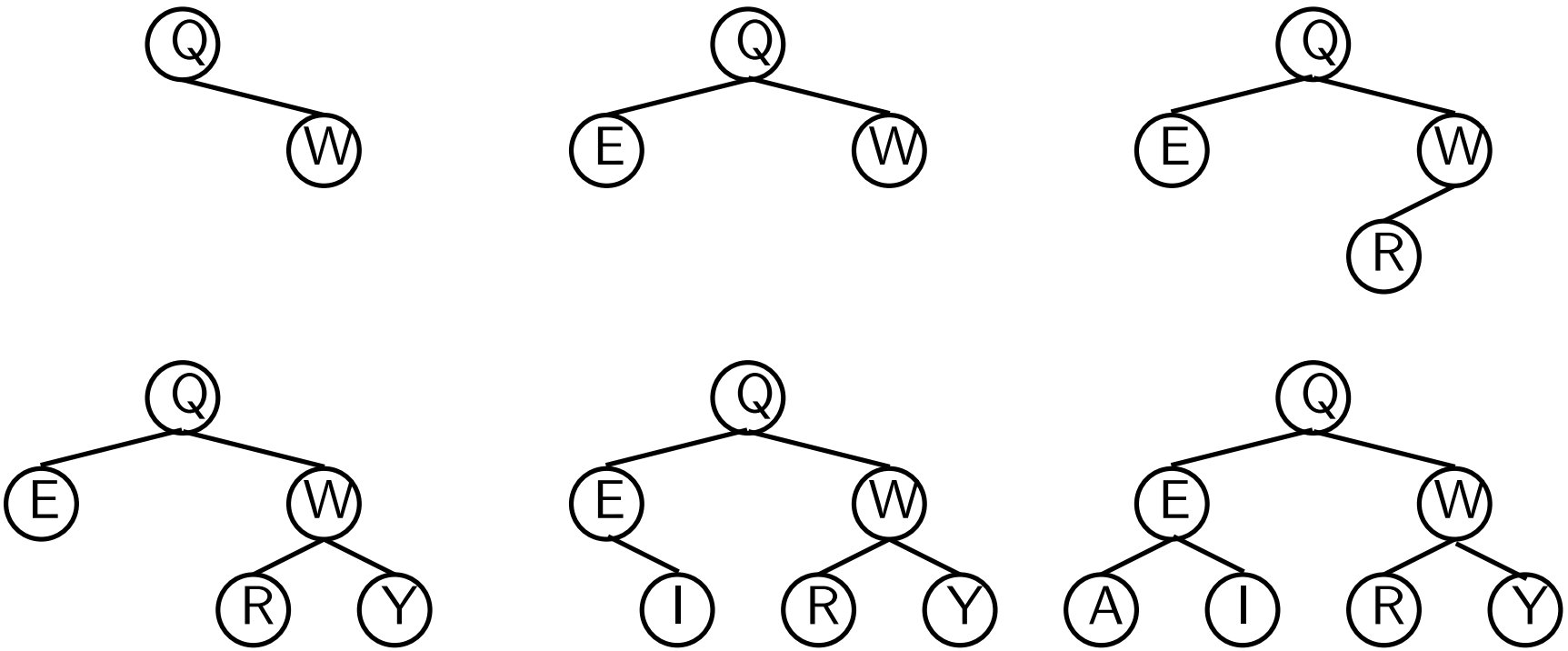
Contenido

10 Árboles balanceados
- Árboles binarios de búsqueda
- Árboles AVL
- Árboles 2-3-4
- Árboles rojinegros

Árboles binarios de búsqueda

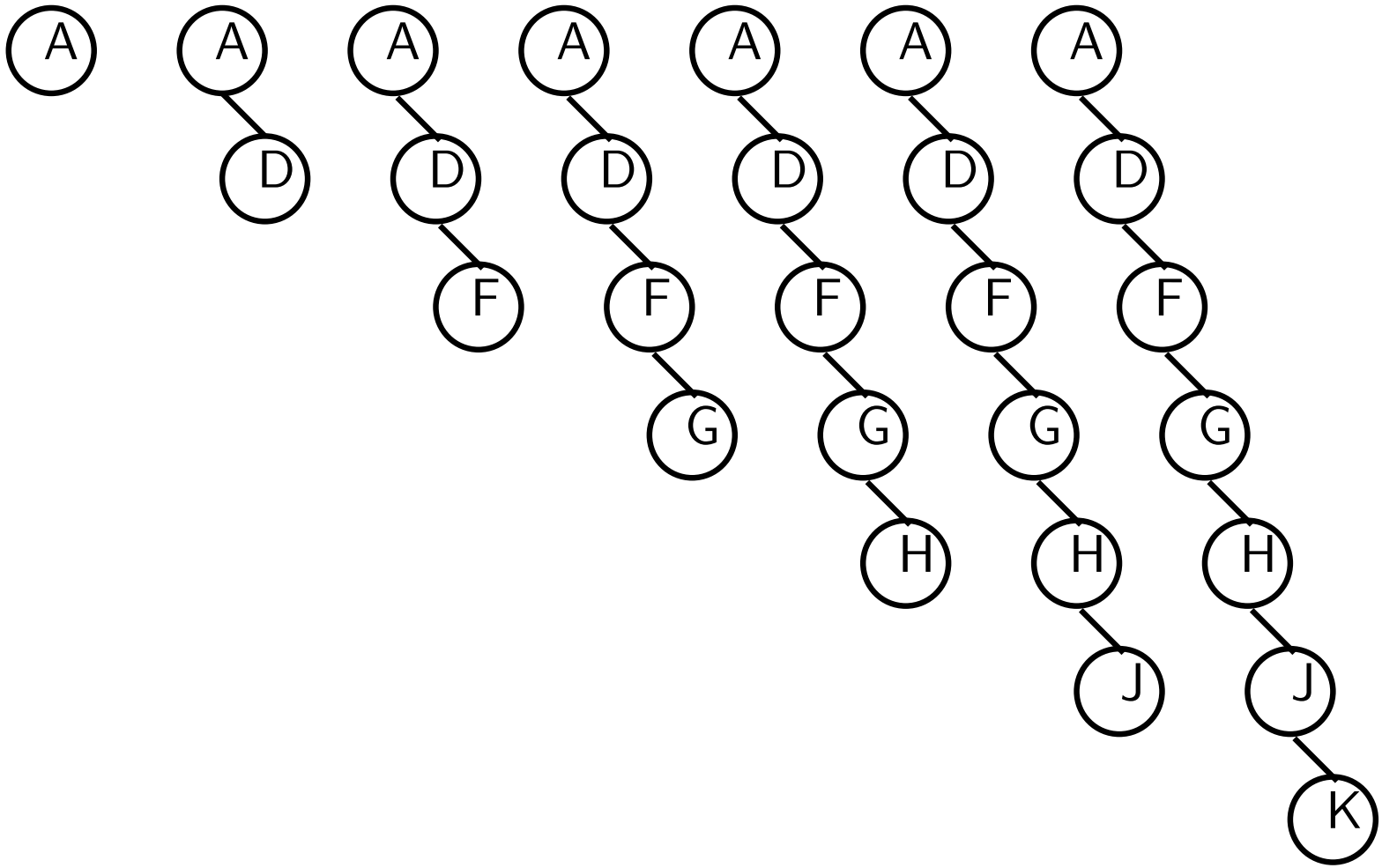
- Recordemos que un árbol binario de búsqueda es un árbol binario donde cada nodo tiene una clave.
- La clave de un nodo es mayor que la clave de su hijo izquierdo y es menor que la de su hijo derecho.
- Las operaciones de búsqueda e inserción son muy sencillas.
- El problema es que un árbol binario de búsqueda puede quedar desbalanceado.

Un ejemplo de árbol binario de búsqueda



Insertando Q, W, E, R, Y, I, A

Otro ejemplo de árbol binario de búsqueda



Insertando A, D, F, G, H, J, K

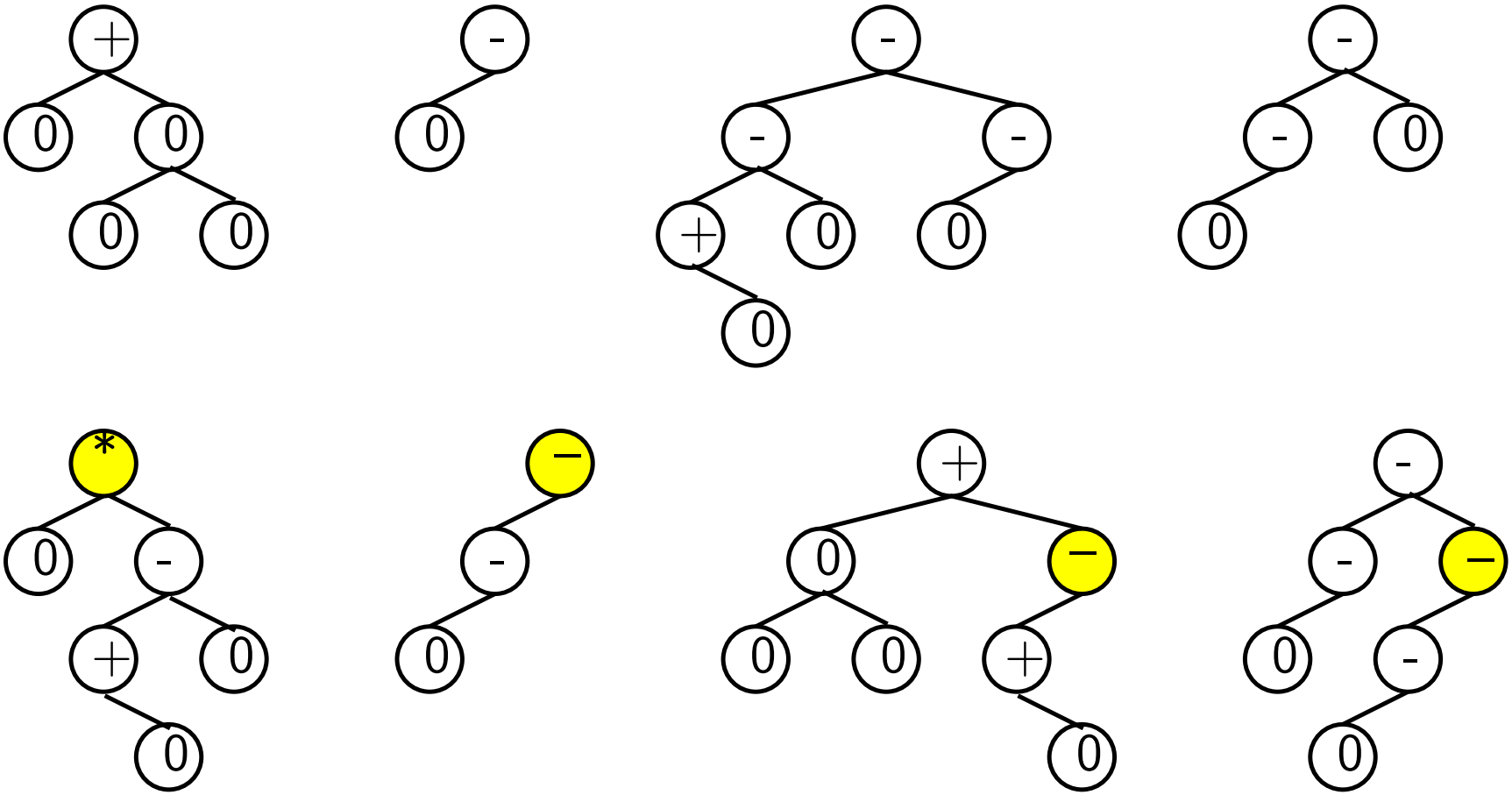
Contenido

10 Árboles balanceados
- Árboles binarios de búsqueda
- Árboles AVL
- Árboles 2-3-4
- Árboles rojinegros

Árboles AVL

- Una solución al problema de creación de árboles binarios degenerados es la de reorganizar los nodos de un árbol conforme se van haciendo operaciones.
- Los árboles AVL cumplen la propiedad de que las alturas de los dos subárboles que comparten una raíz difieren a lo mucho en 1.
- Estos árboles y sus operaciones fueron inventados por Adelson-Velsky y Landis.
- Árboles balanceados de tipo 1.

Ejemplos de árboles AVL



Árboles que sí y no son AVL

Observaciones sobre árboles AVL

- La búsqueda, inserción y el borrado en árboles AVL inician de la misma forma que los árboles binarios de búsqueda.
- La diferencia es que al descubrir un desbalance se debe realizar una o más rotaciones para recuperar el balance.
- La altura máxima de un árbol AVL con n nodos es de aproximadamente $1.44 \log n$.
- Se suelen implementar como árboles rojinegros.

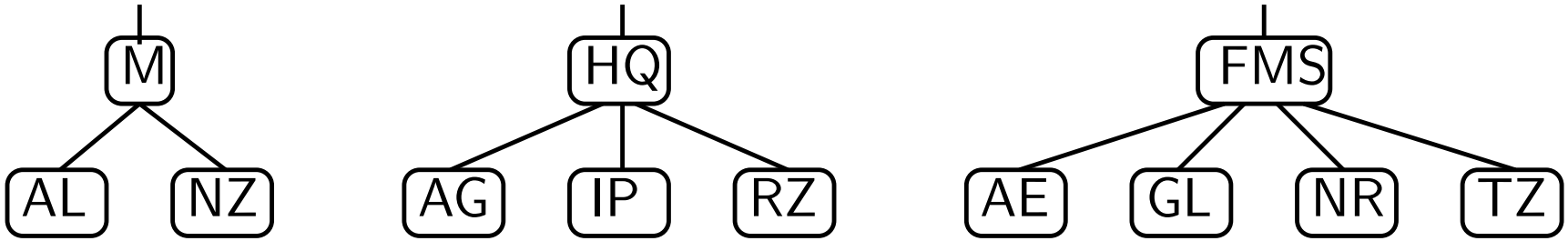
Contenido

10 Árboles balanceados
- Árboles binarios de búsqueda
- Árboles AVL
- Árboles 2-3-4
- Árboles rojinegros

Árboles 2-3-4

- Para entender mejor los árboles rojinegros, primero estudiaremos los árboles 2-3-4.
- Cada nodo de un árbol 2-3-4 puede tener una, dos o tres claves.
- Esto significa dos, tres o cuatro hijos, respectivamente.
- Un nodo con k claves define $k+1$ intervalos (es por eso que tiene $k+1$ hijos).
- La búsqueda en estos árboles es muy parecida a la de los árboles binarios.

Tres tipos de nodos



Un 2-nodo, un 3-nodo y un 4-nodo

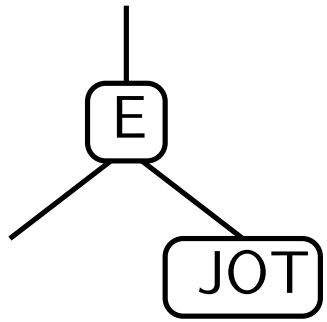
Inserción en árboles 2-3-4

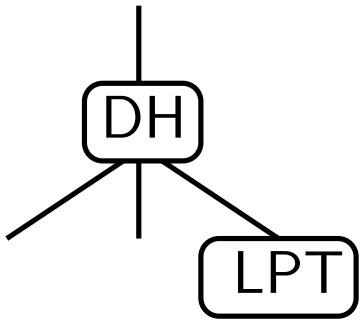
- La inserción es un poco más complicada.
- Primero se hace una búsqueda (posiblemente infructuosa).
- El caso sencillo es cuando debemos insertar en un 2-nodo o en un 3-nodo: simplemente se transforma en un 3-nodo o en un 4-nodo.
- El problema es cuando debemos insertar en un 4-nodo.

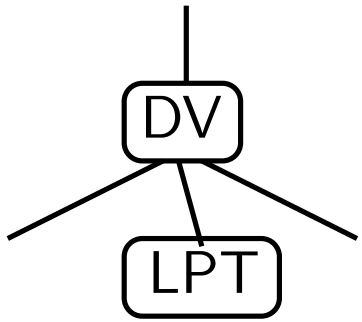
Inserción en un 4-nodo

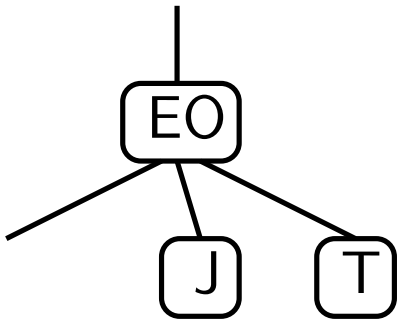
- Una primera opción sería insertar como hijo de ese 4-nodo.
- Pero resulta que una mejor opción es dividir el 4-nodo en dos 2-nodos.
- Se pasa una de las claves al nodo padre.
- Se inserta la nueva clave en uno de los dos 2-nodos recién creados.

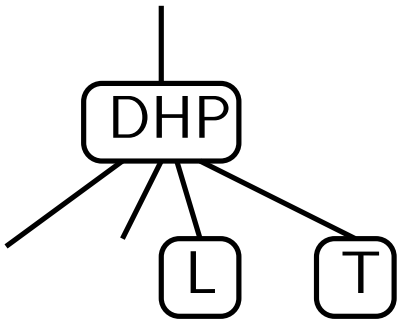
Ejemplos de inserción en un 4-nodo

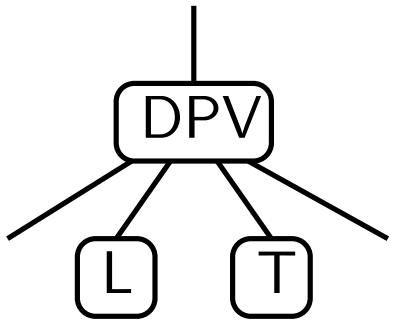












Cuando el padre es un 2-nodo o un 3-nodo

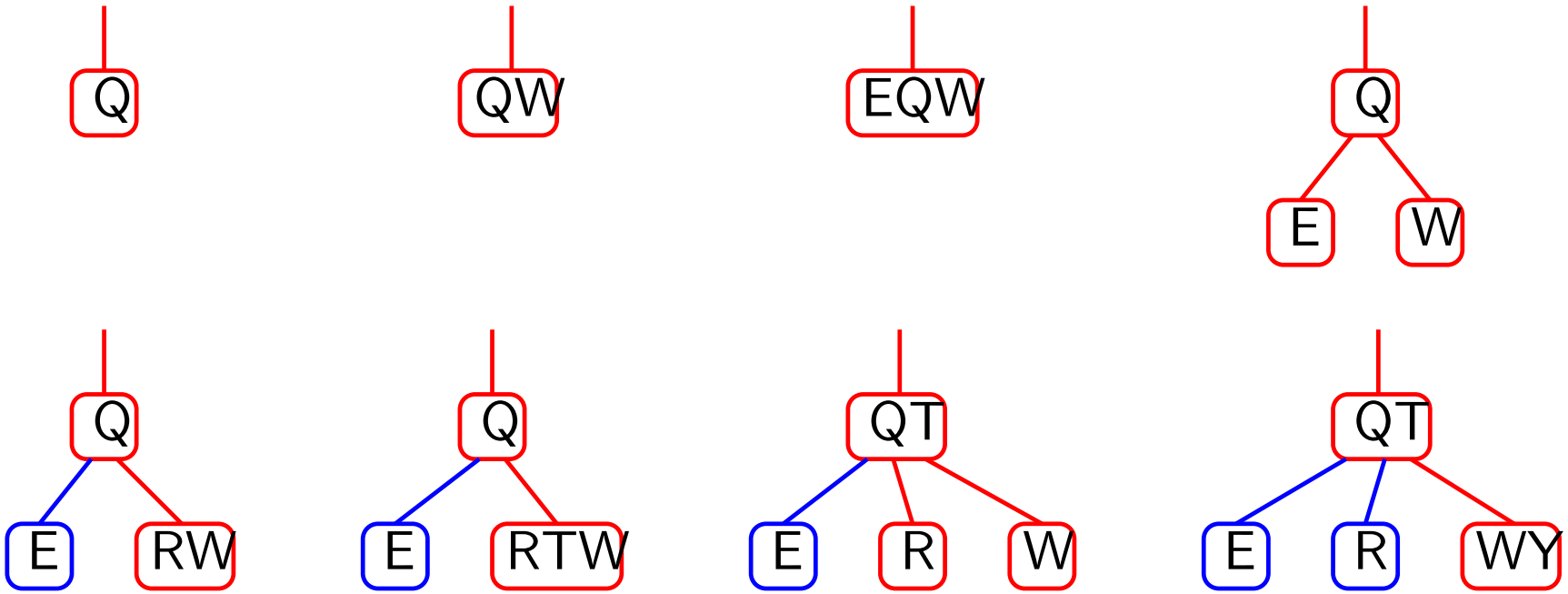
Casos especiales

- ¿Qué pasa si el padre del 4-nodo es un 4-nodo?
- Esto se puede evitar si durante la búsqueda se divide cada 4-nodo que se vea.
- Esto garantiza que ningún 4-nodo tenga un 4-nodo como padre.
- ¿Qué pasa si el 4-nodo es la raíz del árbol?
- En este caso simplemente se divide en tres 2-nodos y uno de ellos se vuelve la nueva raíz del árbol.

Observaciones de árboles 2-3-4

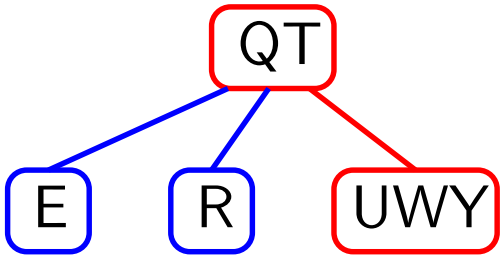
- Las búsquedas y las inserciones no necesitan más de $1 + \log_2 n$ pasos.
- Todos los caminos de la raíz a las hojas miden lo mismo.
- Los árboles 2-3-4 quedan balanceados sin mayor esfuerzo.
- Las implementaciones suelen ser lentas al tener que trabajar con diferentes tipos de nodos.

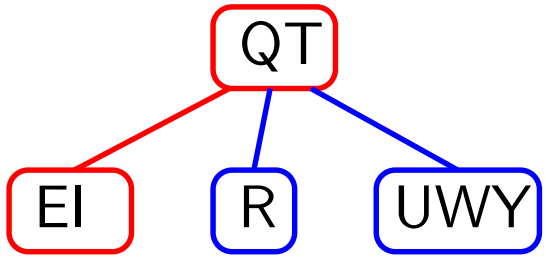
Ejemplo de construcción I

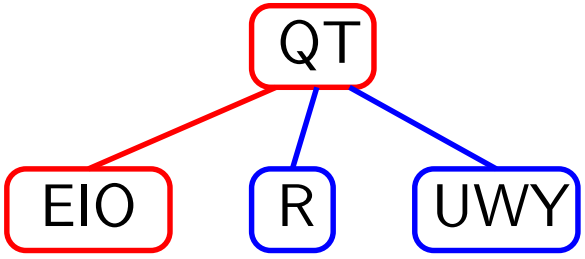


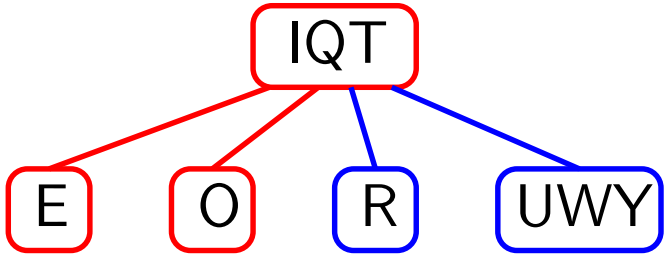
Inserción de Q, W, E, R, T, Y

Ejemplo de construcción II



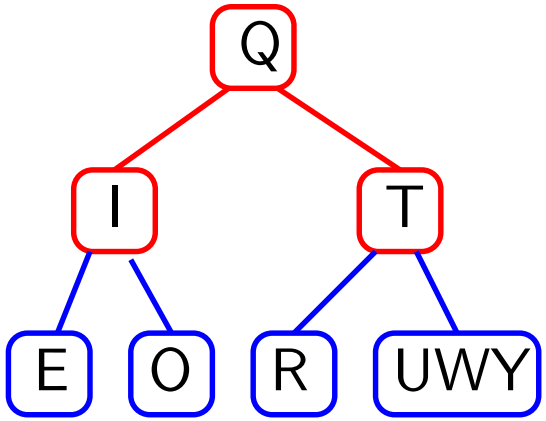


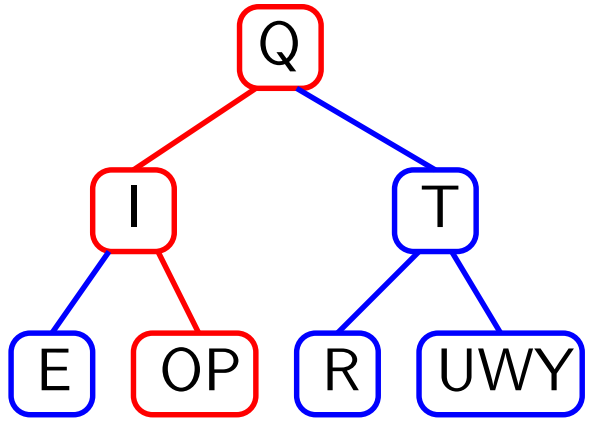


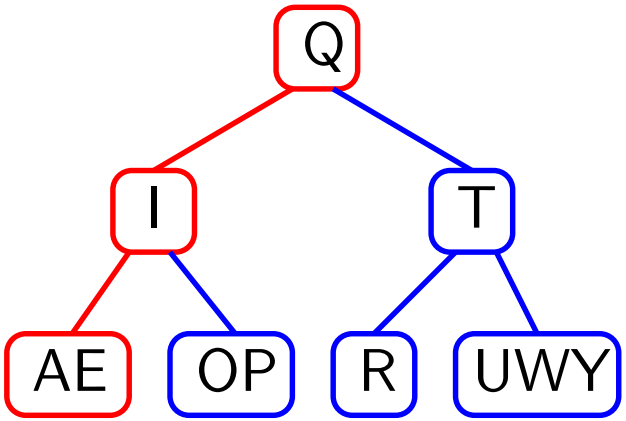


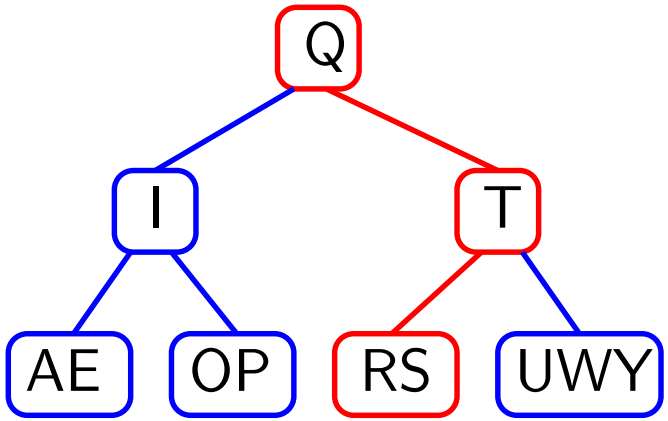
Inserción de U, I, O

Ejemplo de construcción III









Inserción de P, A, S

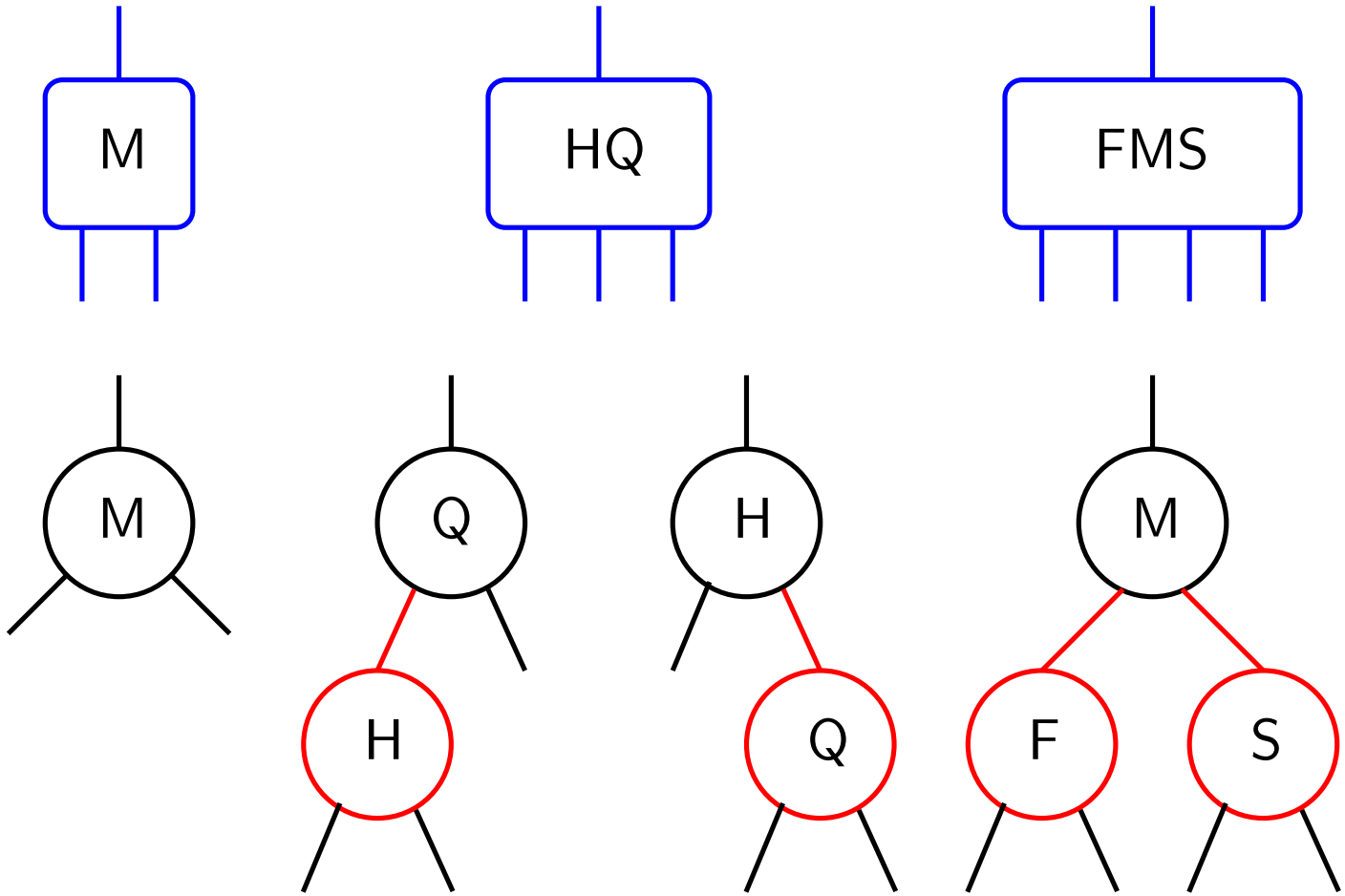
Contenido

10 Árboles balanceados
- Árboles binarios de búsqueda
- Árboles AVL
- Árboles 2-3-4
- Árboles rojinegros

Árboles rojinegros

- Curiosamente, los árboles 2-3-4 se pueden representar como árboles binarios con un bit adicional por liga.
- A este bit se le llama color y éste puede ser rojo o negro.
- A estos árboles se les llama rojinegros.
- Algunas veces se asignan colores a los nodos, pero es lo mismo que asignarle colores a las ligas.

Representación de nodos



2-nodo, 3-nodo y 4-nodo en rojinegro

Observaciones de árboles rojinegros

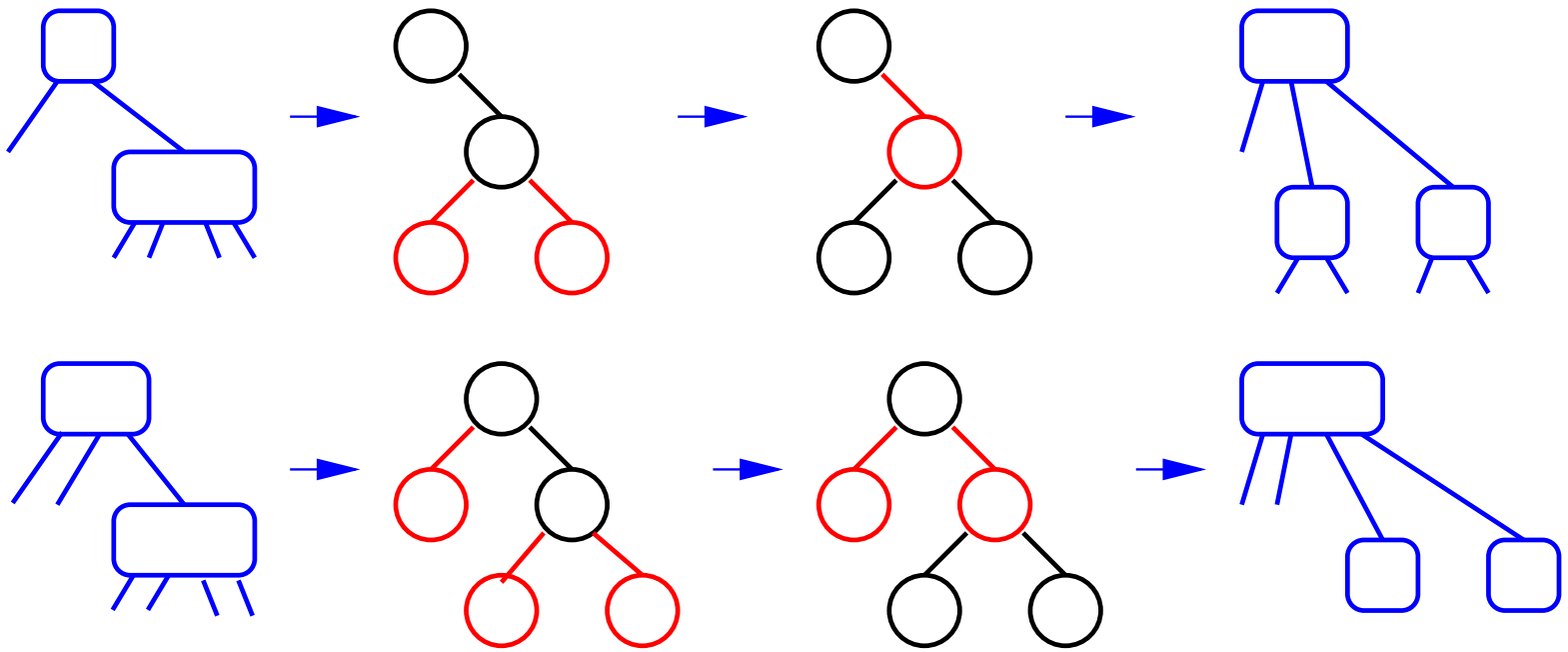
- Observe que si se contraen las ligas rojas se obtiene un árbol 2-3-4.
- Esto implica que nunca hay dos ligas rojas consecutivas.
- Todos los caminos de la raíz a las hojas tienen el mismo número de ligas negras.
- La altura de un árbol rojinegro es

$$\leq 1 + 2 \log_2 n.$$

Operaciones en árboles rojinegros

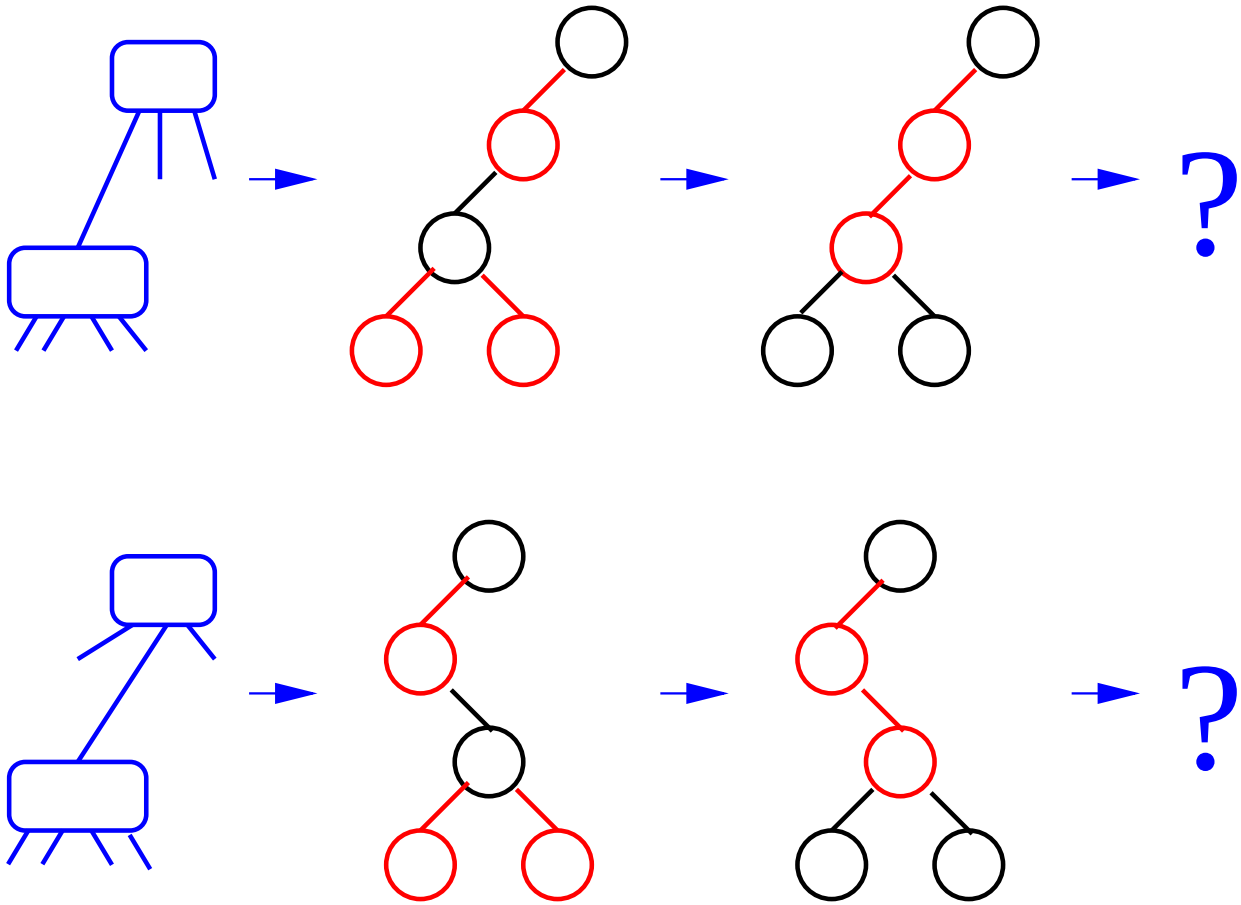
- Cada una de las operaciones con los 4-nodos se puede traducir a las operaciones correspondientes en un árbol rojinegro.
- Estas operaciones se llaman cambios de color y rotaciones.
- Los cambios de color ocurren en los casos sencillos del árbol 2-3-4.
- Las rotaciones ocurren en los casos complicados del árbol 2-3-4.

Cambios de color



División de 4-nodos con un cambio de colores

Necesidad de rotaciones

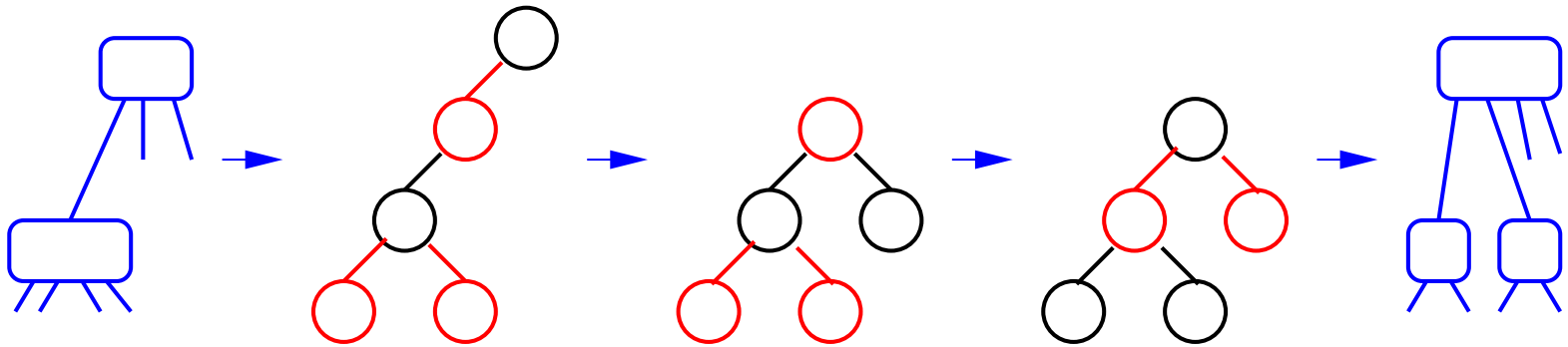


División de 4-nodos con un cambio de colores

¿Qué fue lo que pasó?

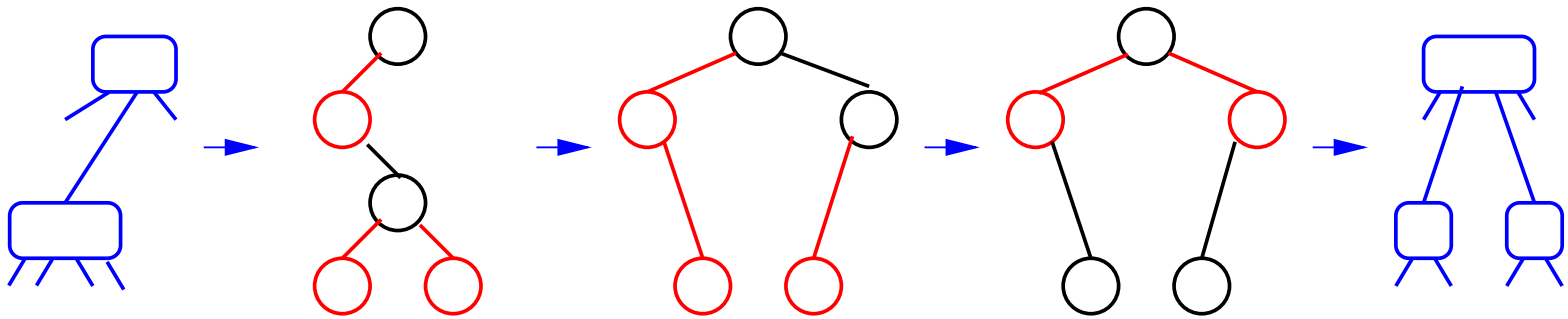
- El problema se detecta cuando se encuentran dos ligas rojas consecutivas.
- Esto sólo pudo ocurrir porque el 3-nodo quedó orientado en la dirección equivocada.
- Hay dos orientaciones para un 3-nodo.
- El problema se resuelve reestructurando el árbol rojinegro con rotaciones.

Rotaciones simples



Rotación simple izquierda izquierda

Rotaciones dobles

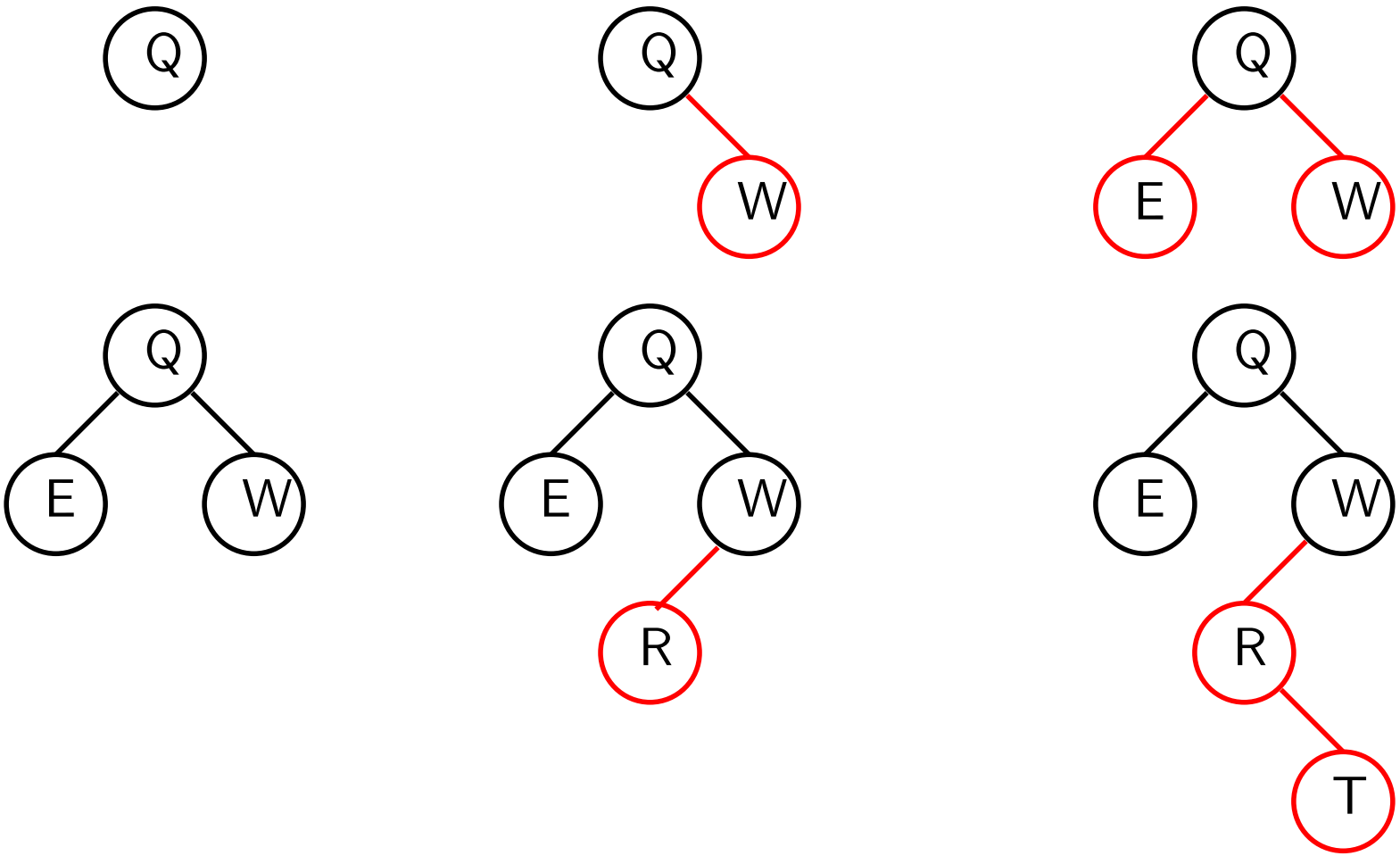


Rotación doble izquierda derecha

Rotaciones rojinegras

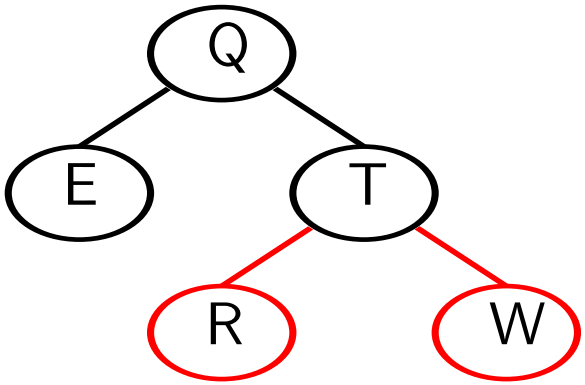
- Hay dos tipos de rotaciones simples (II y DD).
- Y dos tipos de rotaciones dobles (ID y DI).
- Todas ellas se pueden deducir de manera sencilla de las operaciones correspondientes en árboles 2-3-4.

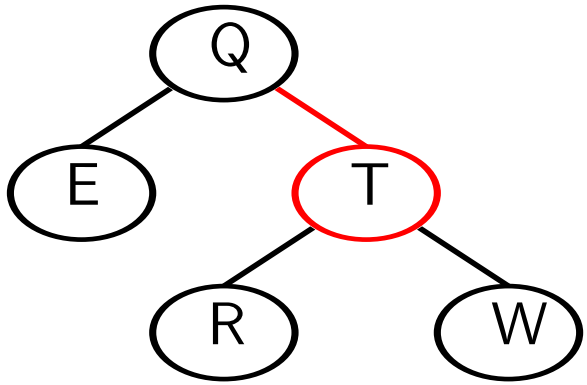
Ejemplo de construcción I

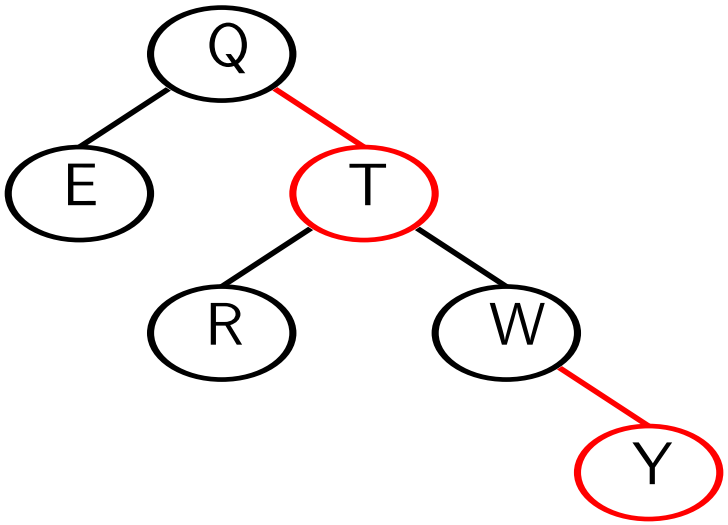


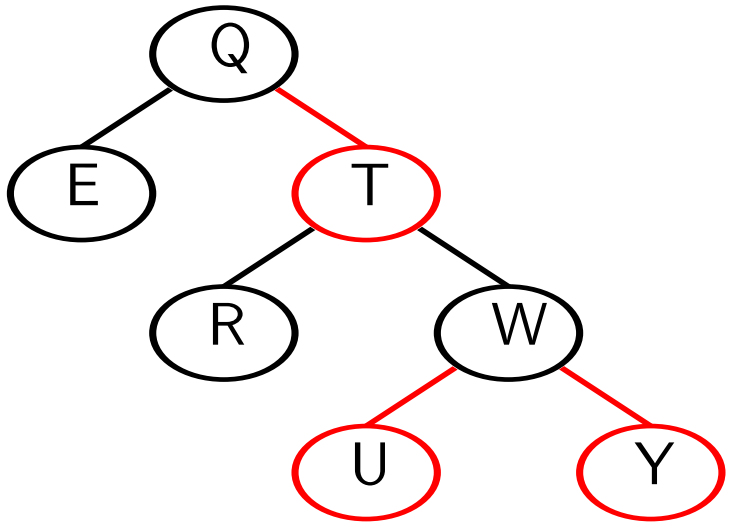
Inserción de Q, W, E, R, T

Ejemplo de construcción II



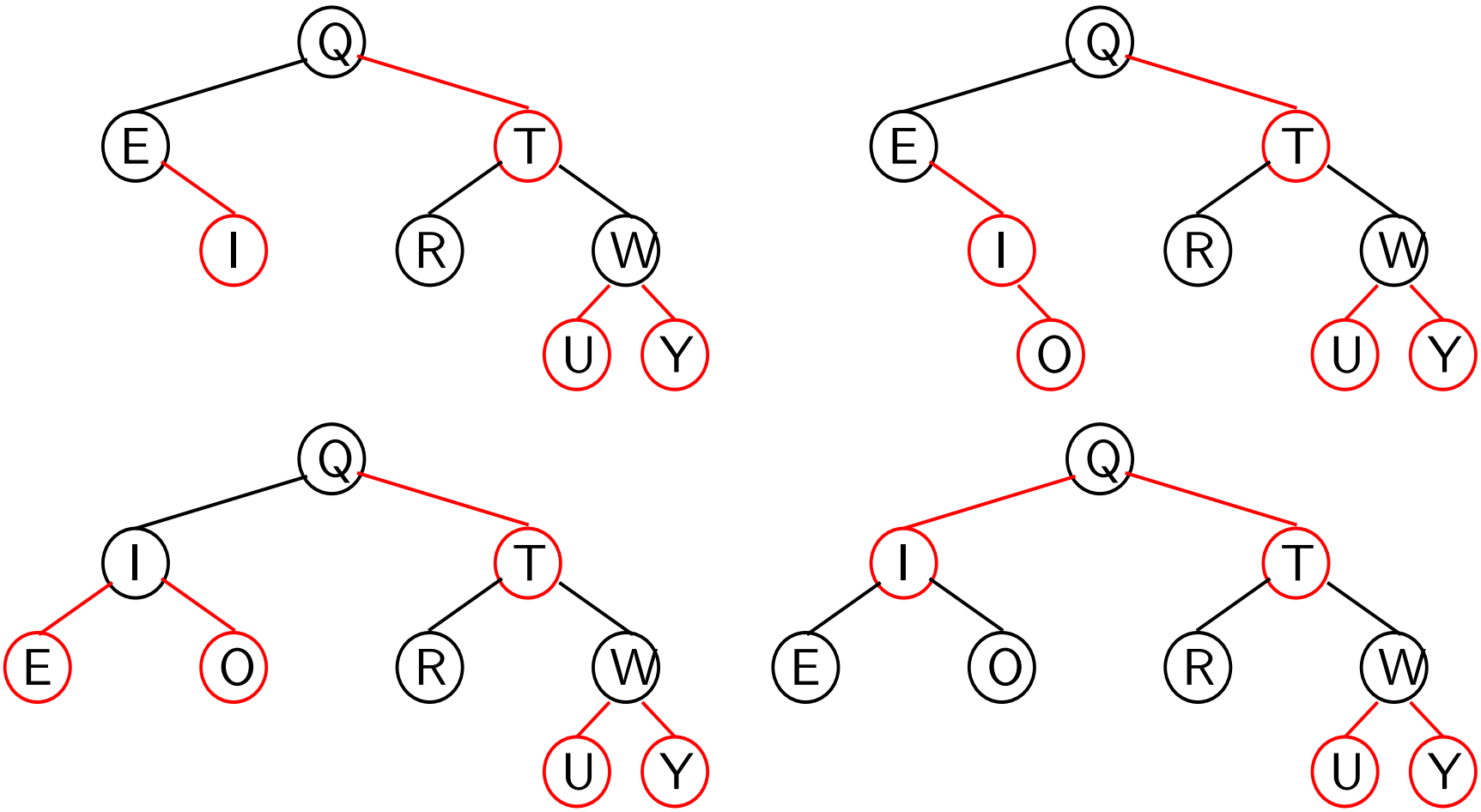






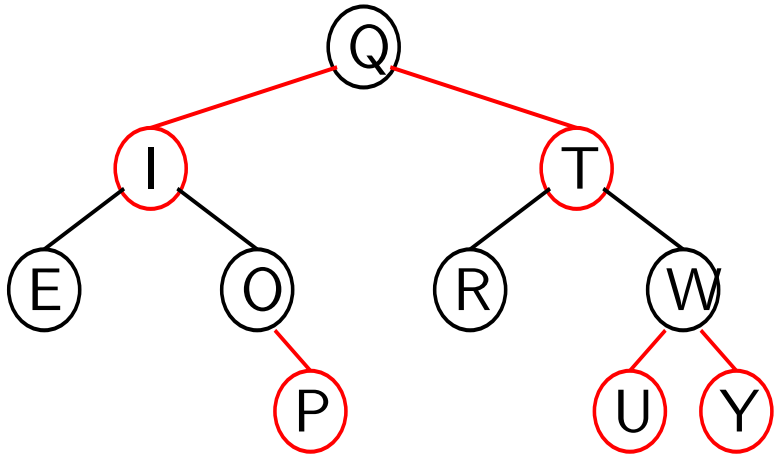
Inserción de T, Y, U

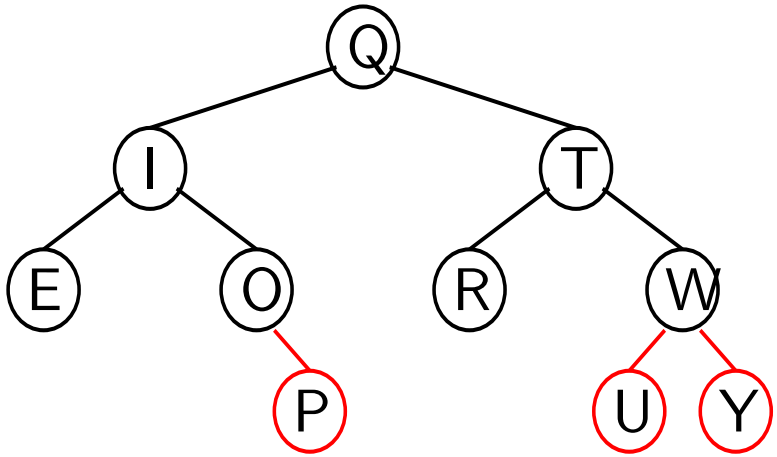
Ejemplo de construcción III

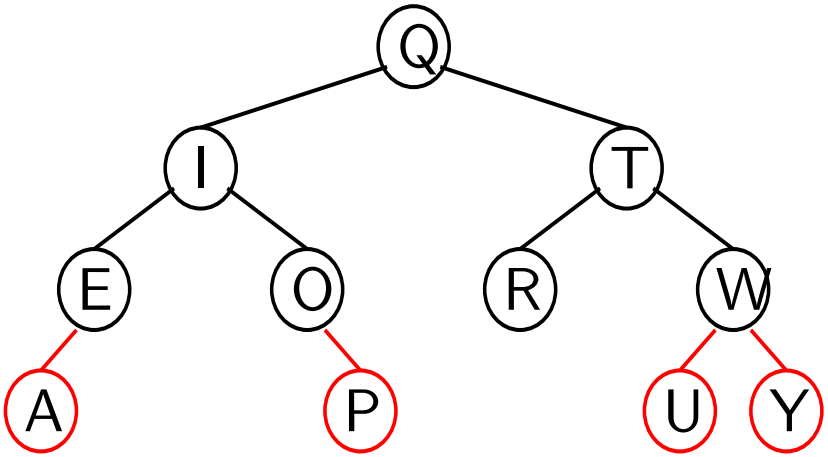


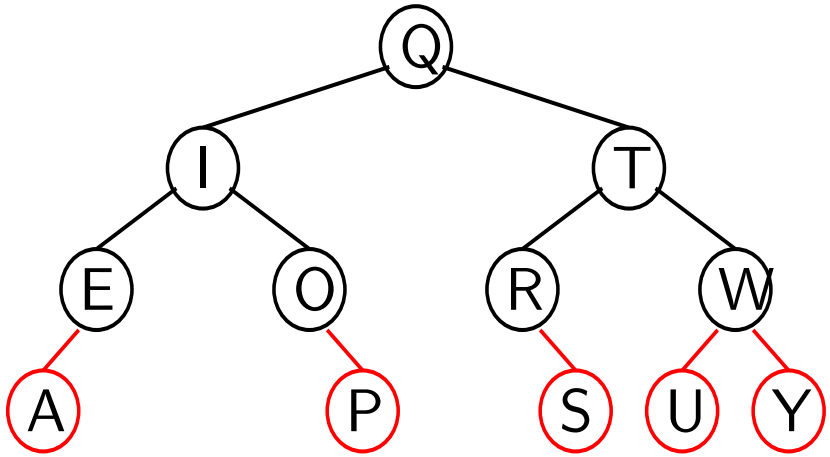
Inserción de I, O

Ejemplo de construcción IV







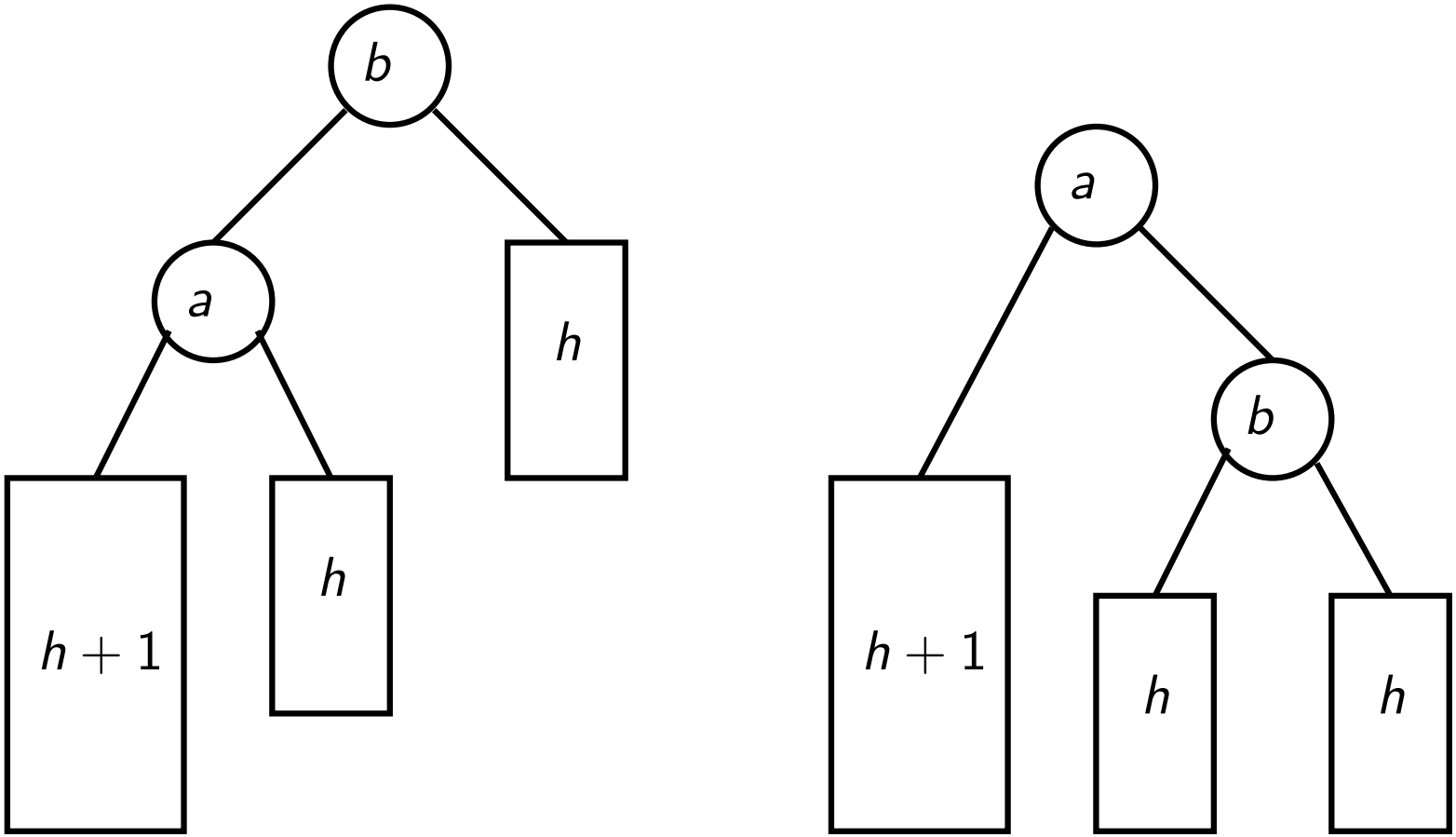


Inserción de P, A, S

Rotaciones AVL

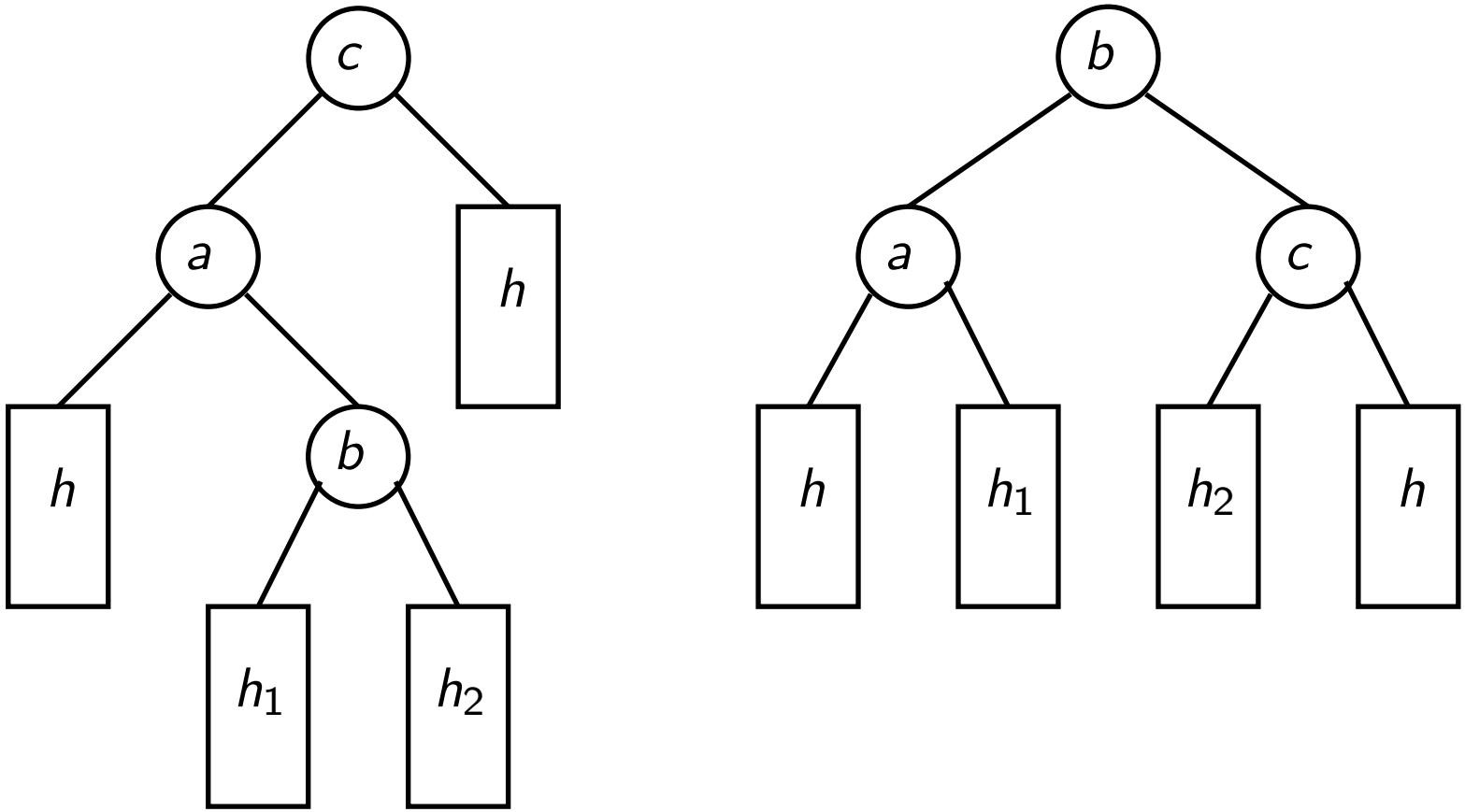
- Recordemos que en un árbol AVL las alturas de los dos subárboles de cada nodo difieren a lo mucho en 1.
- Cuando se hacen inserciones o borrados se puede perder este balance.
- Para recuperar el balance se hace tiene que hacer uno de cuatro tipos de rotaciones.

Rotaciones II y DD



Rotación izquierda izquierda (la rotación derecha derecha es simétrica)

Rotaciones ID y DI



Rotación izquierda derecha con $h = \max\{h_1, h_2\}$ y $h - 1 \leq \min\{h_1, h_2\}$ (la rotación derecha izquierda es simétrica)

Part IV

Índices

Contenido

11 Indexación
- Índices sencillos
- Índices secundarios
- Atado

¿Qué es un índice?

- Las últimas páginas de la mayoría de los libros contienen un índice.
- Tal índice es una tabla que contiene una lista de temas (claves) y los números de página (referencias) donde se pueden encontrar esos temas.
- Todos los índices están basados en los mismos dos conceptos básicos: claves y referencias.

Primera aplicación de índices

- El índice de un libro provee de una manera rápida de encontrar un tema (en lugar de buscar secuencialmente por todo el libro).
- ¿Porqué no podemos usar la búsqueda binaria?
- Si ordenáramos alfabéticamente las palabras de un libro sería muy fácil encontrarlas.
- Dice difícil el entender libro lo más pero poco que sería un.

Índices y registros fijos

- Las palabras en un libro son como los registros fijos (pinned).
- Los índices funcionan de manera indirecta.
- Permiten imponer un orden en un archivo sin ordenar el archivo.
- Esto permite el uso de registros fijos.
- También hace que agregar registros sea mucho más rápido que si mantuvieramos el archivo ordenado.

Segunda aplicación de índices

- Considere el problema de buscar libros en la biblioteca.
- Normalmente uno quiere poder localizarlos por autor, título o tema.
- Una forma de lograr esto es con tres bibliotecas: una organizada por autor, otra por título y la última por tema.
- En la realidad, una biblioteca mantiene tres catálogos: uno organizado por autor, otro por título y el último por tema.

Índices múltiples

- Los tres catálogos son en realidad tres índices distintos.
- Cada uno usa una clave distinta.
- Pero todos usan las mismas referencias.
- De este modo podemos tener distintas formas de buscar registros a través del uso de índices múltiples.

Índices sencillos

- Existen varios tipos de índices dependiendo de la estructura de datos que se utilice para representarlos.
- Primero estudiaremos los índices sencillos.
- Estos se llaman así porque la estructura que los representa es un arreglo sencillo.
- Más adelante estudiaremos índices con estructuras más complejas, en particular con la estructura de un árbol.

Ejemplo de índices y registros

La clave primaria se formó concatenando los dos primeros campos de los registros

```
Claves   Ref  Dir Registros
ANG3795  152   17 LON|2312|Romeo and Juliet|Prokofiev|Maazel
COL31809 338   62 RCA|2626|Quartet in C Sharp Minor|Beethoven|Julliard
COL38358 196  117 WAR|23699|Touchstone|Corea|Corea
DG139201 382  152 ANG|3795|Symphony No. 9|Beethoven|Giulini
DG18807  241  196 COL|38358|Nebraska|Springsteen|Springsteen
FF245    427  241 DG|18807|Symphony No. 9|Beethoven|Karajan
LON2312   17  285 MER|75016|Coq d'Or Suite|Rimsky-Korsakov|Leinsdof
MER75016 285  338 COL|31809|Symphony No. 9|Dvorak|Bernstein
RCA2626   62  382 DG|139201|Violin Concerto|Beethoven|Ferras
WAR23699 117  427 FF|245|Good News|Sweet Honey in the Rock|Sweet Honey
```

Índices y registros de tamaño variable

- En este caso no podemos ordenar el archivo y usar búsqueda binaria en él. ¿Porqué?
- Una alternativa es formar un índice.
- En nuestro caso, el índice es un arreglo que contiene las claves primarias en orden junto con las referencias a los registros.
- Podemos usar búsqueda binaria en el índice.

Observaciones sobre índices sencillos

- El índice está ordenado por clave primaria mientras que el archivo está ordenada por orden de llegada.
- Para poder hacer búsqueda binaria el índice debe estar almacenado en la memoria.
- Una vez que se encuentra la clave primaria en el índice basta hacer una búsqueda en el disco para recuperar el registro.
- Para poder usar el índice varias veces, éste se debe volver persistente.

Operaciones en un archivo indexado

- Creación de los archivos de datos y de índice.
- Carga del índice a la memoria.
- Escritura del índice al disco.
- Agregar registros al archivo de datos.
- Eliminar registros del archivo de datos.
- Actualizar registros en el archivo de datos.
- Actualización del índice.

Escritura del índice al disco

- ¿Qué ocurre si no hacemos o no se completa la escritura del índice al disco?
- El archivo de índice no reflejará el estado actual del archivo de datos.
- Es muy importante que el programa contenga al menos las siguientes dos previsiones:
- Se debe poder saber que el índice no es correcto (mediante una bandera de estado).
- Se debe poder reconstruir el índice.

Actualización de registros

- Agregar o eliminar registros es muy sencillo.
- Recordemos que se debe actualizar el índice.
- Hay dos tipos de actualización de registros:
- Si la actualización cambia el valor del campo de clave se necesita una reorganización del índice (y posiblemente de los datos).
- Si la actualización no cambia el valor del campo de clave no se necesita reorganizar el índice (pero posiblemente sí los datos).

Índices demasiado grandes

- ¿Qué pasa si el índice es tan grande que no se puede almacenar en memoria?
- En este caso debemos pensar en otras organizaciones del índice.
- Una opción es usar una estructura de árbol (por ejemplo los árboles B y B^+ que veremos más adelante).
- Otra opción es usar alguna técnica de dispersión (como las que estudiaremos al final del curso).

Contenido

11 Indexación
- Índices sencillos
- Índices secundarios
- Atado

Acceso por claves múltiples

- Es difícil pensar que haremos búsquedas usando las claves primarias.
- Casi siempre haremos búsquedas usando combinaciones de claves secundarias.
- ¿Cómo lograr encontrar un dato usando una clave secundaria?
- Podríamos mantener un segundo índice ordenado por la clave secundaria y con una referencia al registro que la contiene.

Claves secundarias duplicadas

- El primer problema que surge es que las claves secundarias no son únicas.
- Por lo tanto puede haber varios registros distintos que la contengan.
- Esto lo resolveremos ordenando las diferentes apariciones de una clave secundaria según su referencia.

Uso de referencias

- Dijimos que podríamos pensar en usar como referencia el lugar de inicio del registro correspondiente.
- En vez de eso usaremos como referencia la clave primaria.
- La ventaja de esto es que si un registro cambia de lugar sólo se debe actualizar el índice primario.

Operaciones en índices secundarios

- Agregar un registro: muy parecida a como se hace en un índice primario.
- Borrar un registro: sólo se arregla el índice primario, dejando que la operación falle al ir del índice secundario al primario.
- Actualizar un registro: hay tres casos, según se afecten las claves primarias, secundarias o ninguna de las dos.

Ejemplo de índices secundarios

Índice primario y dos índices secundarios

Primaria	Ref	Secundaria	Primaria	Secundaria	Primaria
ANG3795	152	BEETHOVEN	ANG3795	COQ D'OR SU	MER75016
COL31809	338	BEETHOVEN	DG139201	GOOD NEWS	FF245
COL38358	196	BEETHOVEN	DG18807	NEBRASKA	COL38358
DG139201	382	BEETHOVEN	RCA2626	QUARTET IN	RCA2626
DG18807	241	COREA	WAR23699	ROMEO AND J	LON2312
FF245	427	DVORAK	COL31809	SYMPHONY NO	ANG3795
LON2312	17	PROKOFIEV	LON2312	SYMPHONY NO	COL31809
MER75016	285	RIMSKY-KORS	MER75016	SYMPHONY NO	DG18807
RCA2626	62	SPRINGSTEEN	COL38358	TOUCHSTONE	WAR23699
WAR23699	117	SWEET HONEY	FF245	VIOLIN CONC	DG139201

Acceso usando combinaciones de claves

- Es sencillo implementar búsquedas con uniones o intersecciones (OR y AND).
- Primero encontramos la primera entrada de cada índice secundario que contiene las palabras de nuestra búsqueda.
- A partir de este momento se procesan las dos listas secuencialmente.
- Notemos que a cada paso es fácil saber si esa entrada nos interesa o no ya que vienen ordenadas por clave primaria.

Mejoras a la estructura

- Observe que es poco eficiente insertar varias veces la misma clave secundaria.
- Una posibilidad es dejar que cada clave secundaria se refiera a un vector de tamaño fijo de claves primarias.

```
Secundaria  Primarias

BEETHOVEN   ANG3795  DG139201 DG18807  RCA2626
COREA       WAR23699
...
```

Una mejor solución

- Otra posibilidad es la de separar el índice secundario en dos archivos.
- Uno conteniendo sólo las claves secundarias y una referencia a la primera clave primaria en el segundo archivo.
- El segundo archivo tendrá la estructura de una lista ligada ordenada.

Ejemplo de índice en dos archivos

Secundaria	Ref	#	Primaria	Sig
BEETHOVEN	0	0	ANG3795	3
COREA	6	1	COL31809	-1
DVORAK	1	2	DG18807	7
PROKOFIEV	8	3	DG139201	2
RIMSKY-KORS	5	4	FF245	-1
SPRINGSTEEN	9	5	MER75016	-1
SWEET HONEY	4	6	WAR23699	-1
		7	RCA2626	-1
		8	LON2312	-1
		9	COL38358	-1

Ventajas y desventajas

- El primer archivo sólo debe ordenarse cuando se agregue una nueva clave secundaria.
- El segundo archivo no debe reordenarse y se puede hacer con registros de tamaño fijo.
- El segundo archivo no tiene la propiedad de localidad y se requiere hacer búsquedas en el disco para moverse en la lista ordenada.

Índices selectivos

- Estos son índices que no contienen entradas para todos los registros de un archivo.
- Los índices selectivos son útiles cuando el contenido de un archivo se puede dividir natural y lógicamente en diversas categorías.

Contenido

11 Indexación
- Índices sencillos
- Índices secundarios
- Atado

Atado (binding)

- ¿En qué momento queda atada una clave a la dirección física del registro asociado?
- En nuestro índice primario en el momento de construcción mientras que en nuestro índice secundario en el momento de uso.
- El primer momento resulta en un acceso más rápido pero tiene un costo de reorganización muy alto.
- El segundo momento resulta en un acceso más lento pero más seguro.

Atado pronto y tardío

- El atado tardío permite que los índices secundarios contengan errores sin que esto sea un gran problema.
- Al final es mejor hacer los cambios importantes en un solo lugar (el índice primario) que en varios.
- Hay al menos una aplicación del atado pronto: en sistemas de archivos grabados en medios no reescribibles (nunca cambian).

Part V

Ordenamiento externo

Contenido

12 Operaciones cosecuenciales
- Procesos cosecuenciales
- Aplicación de procesos cosecuenciales
- Ordenamiento interno revisitado
- Ordenamiento externo en discos
- Ordenamiento externo en cintas

Operaciones cosecuenciales

- Estas son operaciones que consisten en el procesamiento coordinado de dos o más listas secuenciales para producir una sola lista de salida.
- Algunos ejemplos de estas operaciones:
  - Mezcla o unión.
  - Apareamiento o intersección.
- Se requiere poner atención a varios detalles.
- Que son sencillos bajo ciertas suposiciones.

Detalles

- Inicialización: acomodar las cosas de modo que podamos comenzar.
- Obtención y acceso del siguiente elemento de la lista: se requiere de un método sencillo.
- Sincronización: debemos asegurar que estamos procesando los elementos en las listas en el orden apropiado de modo que no dejemos de considerar a ninguno.
- Manejo de fin de archivo: saber qué hacer en este caso.
- Reconocimiento de errores: si ocurre algún error en los datos (por ejemplo datos duplicados o fuera de orden) queremos poder detectarlo y tomar alguna acción.
- Algoritmo: eficiente, sencillo y fácil de modificar.

Suposiciones

- Se desean procesar dos o más archivos de entrada en forma paralela para producir uno o más archivos de salida.
- Cada archivo está ordenado por una o más claves y todos los archivos están ordenados de la misma forma.
- De ser necesario, deben existir dos valores especiales (centinelas) uno menor y otro mayor que todas las claves posibles.
- Los registros se deben procesar en el orden lógico (no físico).
- Para cada archivo debe haber un único registro actual y éste es el registro que se debe procesar.
- Los registros sólo se pueden manipular en memoria interna.

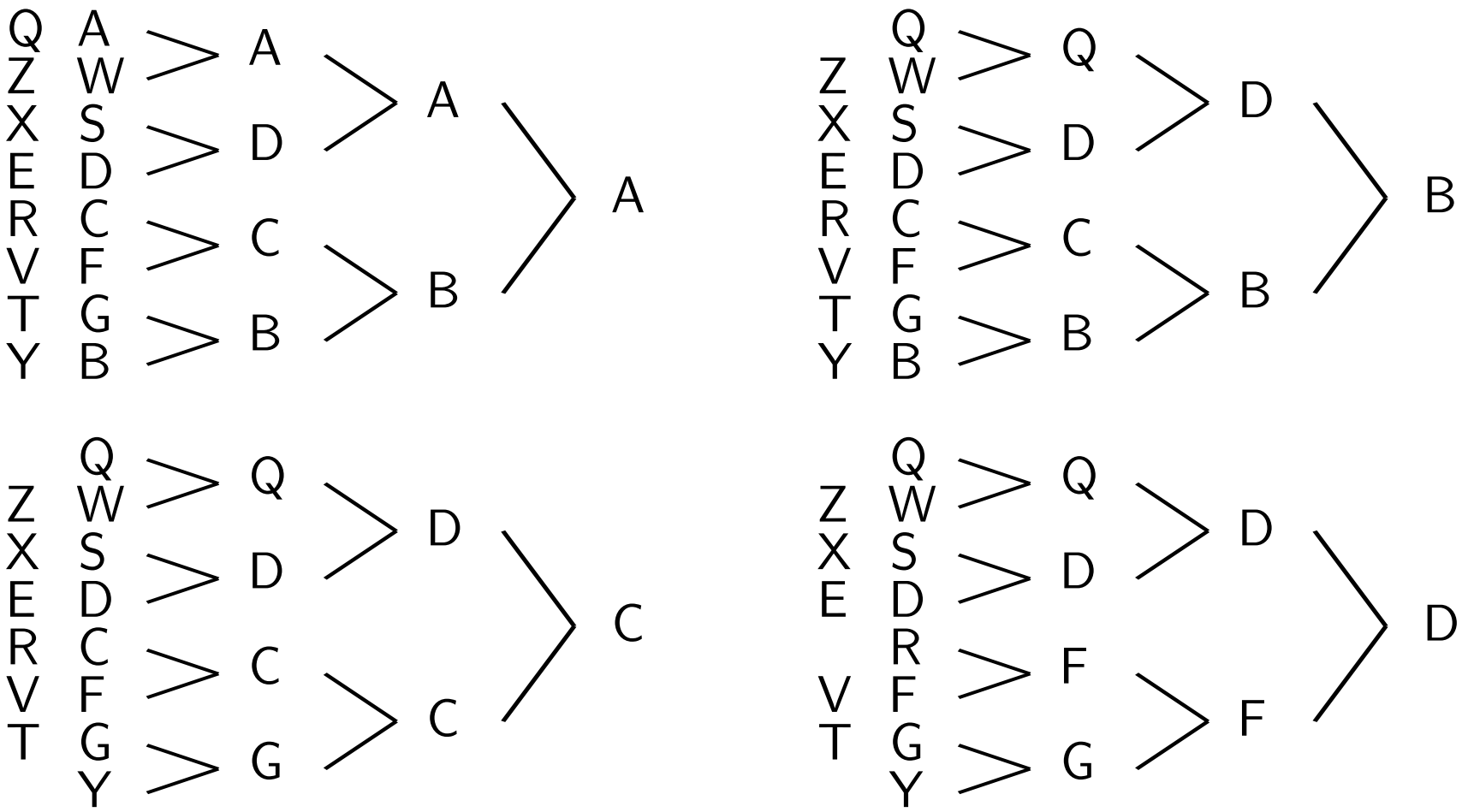
Contenido

12 Operaciones cosecuenciales
- Procesos cosecuenciales
- Aplicación de procesos cosecuenciales
- Ordenamiento interno revisitado
- Ordenamiento externo en discos
- Ordenamiento externo en cintas

Mezcla de k vías

- Considere el problema de mezclar k listas ordenadas de entrada en una sola lista ordenada de salida.
- La forma más sencilla es a través de un ciclo que revise todas las listas para averiguar cuál es el siguiente elemento a procesar.
- Esto es una generalización simple de cómo se hace con dos listas.
- No es la forma más eficiente si k es grande.

Árbol de selección o torneo



Torneo con ocho listas (tres niveles)

Contenido

12 Operaciones cosecuenciales
- Procesos cosecuenciales
- Aplicación de procesos cosecuenciales
- Ordenamiento interno revisitado
- Ordenamiento externo en discos
- Ordenamiento externo en cintas

Ordenamiento interno

- Si los datos a ordenar caben en memoria, la forma más simple de ordenarlos consta de tres pasos:
  - Leer los datos.
  - Ordenar los datos.
  - Escribir los datos.
- De esta forma parece que el tiempo necesario para estas tres operaciones es la suma de los tres tiempos de cada paso.

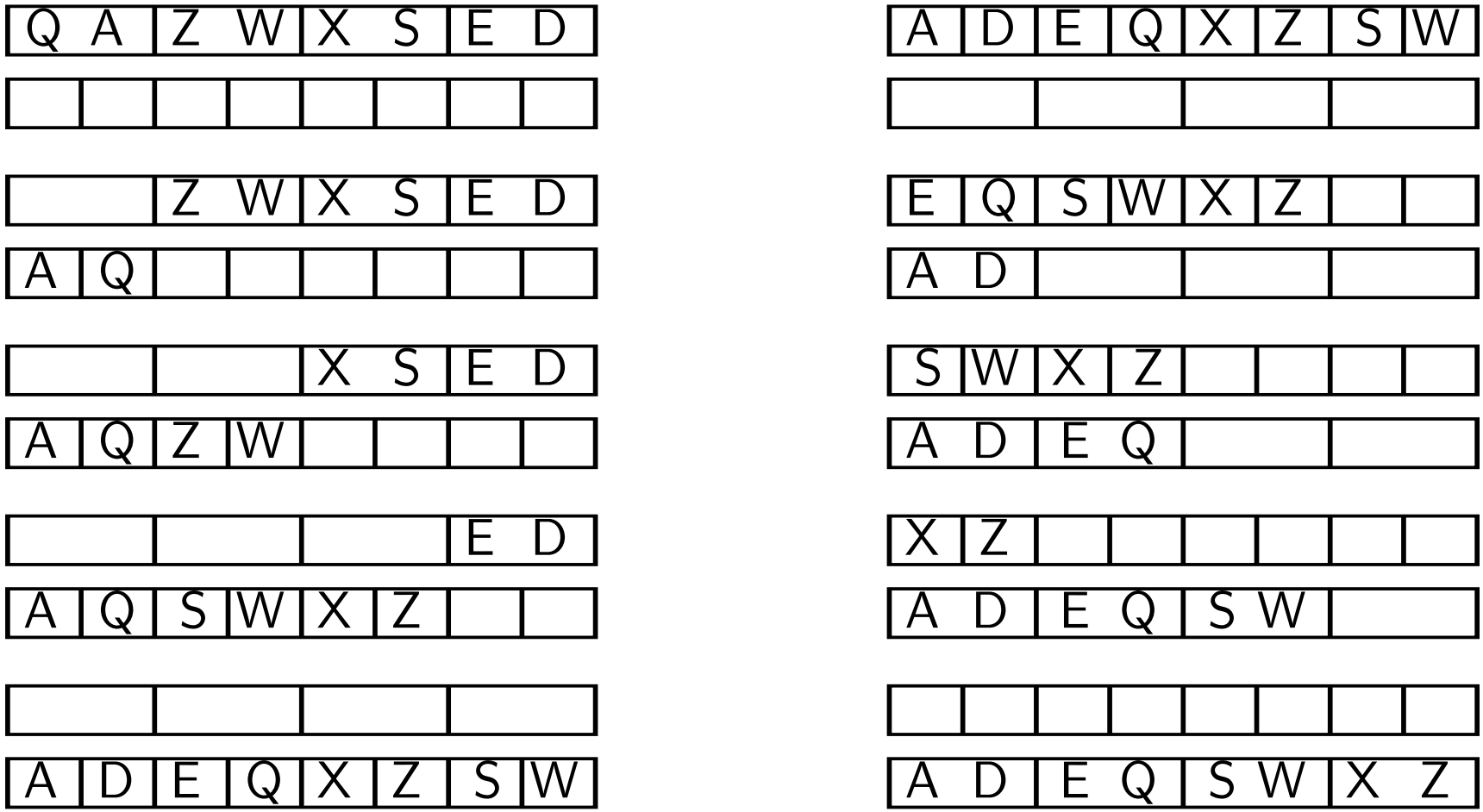
Ahorrando tiempo

- Existen formas de lograr que algunos de estos pasos se empalmen en el tiempo.
- Si hubiera dos discos la lectura y la escritura se podrían hacer en paralelo.
- Si sólo hubiera un disco se podría empalmar el ordenamiento con la lectura y escritura.
- La idea es usar ordenamiento por montículo.

Ordenamiento por montículo

- En la primera etapa se construye el montículo al mismo tiempo que se leen los datos.
- En la segunda etapa vamos ordenando los datos al mismo tiempo que los escribimos.
- La clave es el uso de buffers múltiples.
- En la primera etapa se lee un buffer mientras el anterior se agrega al montículo.
- En la segunda etapa se escribe un buffer mientras el siguiente se borra del montículo.

Montículo en dos pasos



Primera y segunda etapa del ordenamiento

Contenido

12 Operaciones cosecuenciales
- Procesos cosecuenciales
- Aplicación de procesos cosecuenciales
- Ordenamiento interno revisitado
- Ordenamiento externo en discos
- Ordenamiento externo en cintas

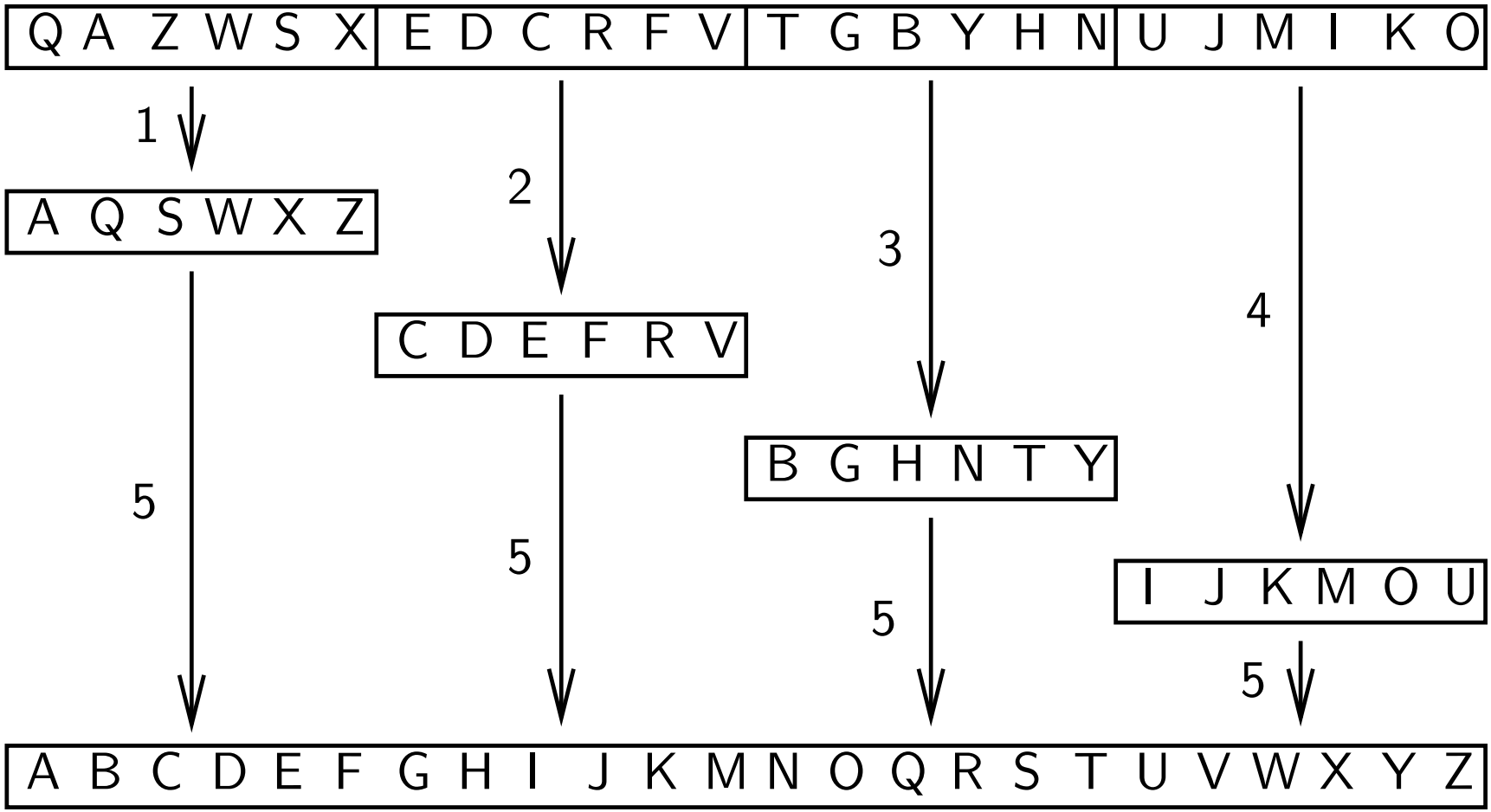
Ordenamiento externo con mezclas

- Aun no tenemos un algoritmo para ordenar archivos que no quepan en memoria.
- Sin embargo, el algoritmo de mezcla de k vías es un buen principio.
- Recordemos que existen algoritmos que pueden ordenar un vector sin necesitar almacenamiento adicional.
- Un ejemplo eficiente es el de ordenamiento por montículo.

Corridas

- Podemos leer una parte del archivo, ordenarla y grabarla en un archivo.
- Luego leer una segunda parte, ordenarla y grabarla en otro archivo, etc.
- A cada uno de estos subarchivos ordenados se le llama una corrida.
- Una vez que el archivo original se ha ordenado parcialmente en varias corridas podemos usar el algoritmo de mezcla de k vías para producir un solo archivo ordenado.

Ejemplo de corridas



Cuatro corridas y mezcla de 4 vías

Características

- Ordenar archivos de cualquier tamaño.
- La lectura de la entrada es secuencial.
- La lectura de cada corrida es secuencial.
- La escritura de la salida es secuencial.
- Búsquedas sólo al cambiar de corrida.
- Si se usan montículos se puede traslapar la entrada y salida con el procesamiento.
- También funciona con cintas.

¿Cuánto se tarda este proceso?

- Se tarda más de lo deseado al cambiar de corrida pues se requiere una búsqueda.
- Esto se puede minimizar leyendo en un buffer tanto como se pueda de cada corrida.
- Sea M la memoria disponible.
- k corridas implica buffer de tamaño M/k.
- Como cada corrida mide M entonces se hacen $\frac{M}{M/k} = k$ búsquedas por corrida.
- El número total de búsquedas es k^2.

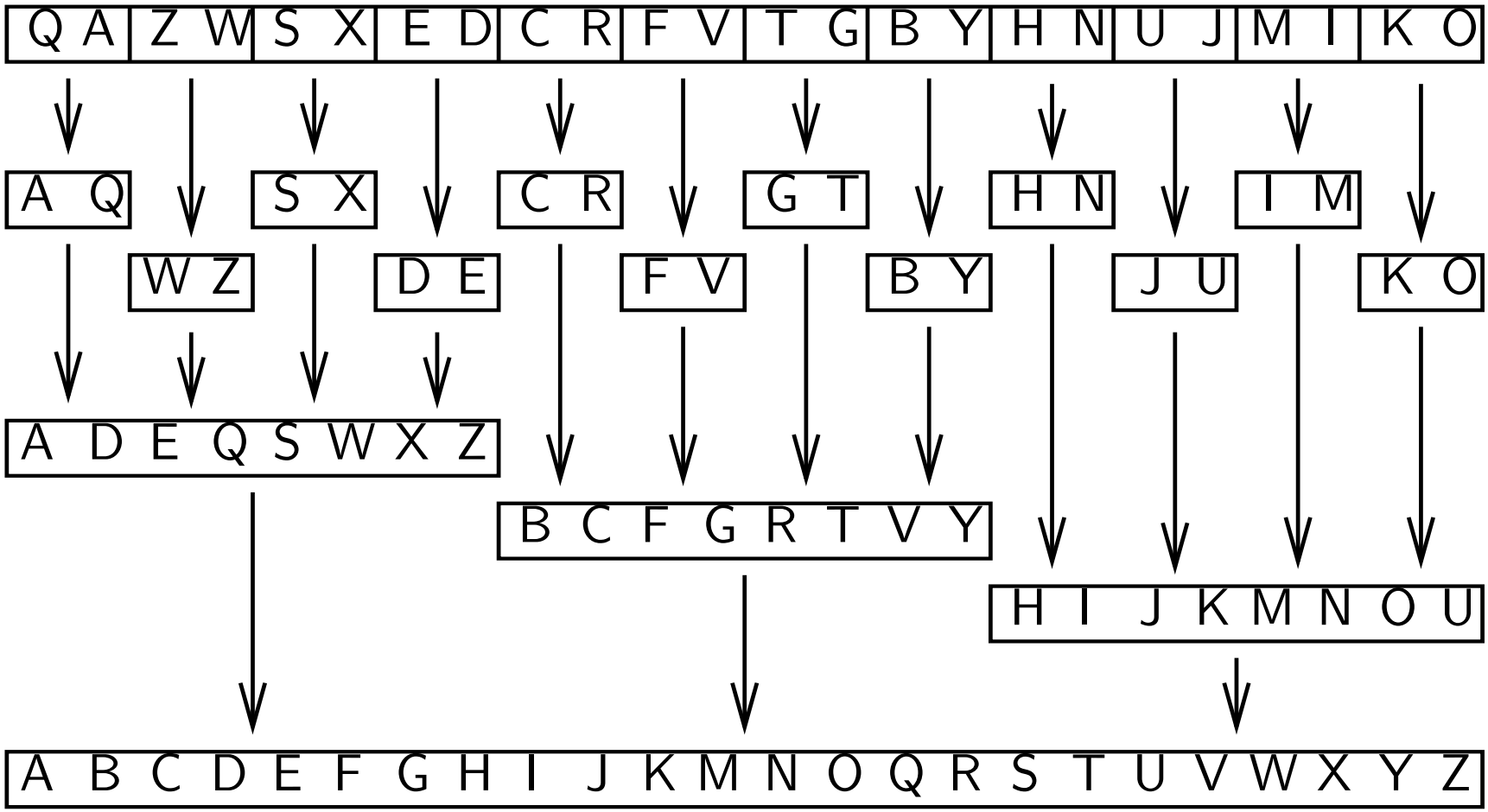
¿Cómo disminuir este costo?

- Agregar más memoria o discos (que normalmente no se puede hacer).
- Realizar la mezcla en más de un paso.
- Incrementar de alguna forma el tamaño inicial de las corridas (usando reemplazo directo).
- Encontrar otras formas de traslapar la entrada y salida con el proceso.

Mezcla en dos etapas

- Suponga que tenemos $k = k_1 k_2$ corridas de tamaño M cada una.
- Entonces podemos realizar k_1 veces el proceso de mezclar k_2 corridas para obtener k_1 corridas de tamaño $k_2 M$.
- Finalmente realizamos una mezcla de estas k_1 corridas.

Ejemplo de mezcla en dos etapas



Con $k = k_1 k_2 = 3 \cdot 4 = 12$ corridas

¿Cuál es el número total de búsquedas?

- La primera etapa hará $k_1 k_2^2$ búsquedas.
- Cada corrida de la segunda etapa mide $k_2 M$ y tendrá un buffer de tamaño M/k_1 por lo que hará $\frac{k_2 M}{M/k_1} = k_1 k_2$ búsquedas.
- Entre las dos etapas se harán $k_1 k_2 (k_1 + k_2)$ búsquedas.
- Esto es menos de k^2 búsquedas si $k_1, k_2 > 1$.

Observaciones

- Cada registro se lee dos veces (pero esto se hace de forma secuencial).
- ¿Cuál es el mejor valor para k_1 y k_2?
- Es fácil ver que deben ser iguales a $\sqrt{k}$.
- Sólo es posible si k es un cuadrado perfecto.
- ¿Qué hacer si k no es un cuadrado perfecto?
- Se puede hacer un análisis similar para mezcla en m etapas.

Contenido

12 Operaciones cosecuenciales
- Procesos cosecuenciales
- Aplicación de procesos cosecuenciales
- Ordenamiento interno revisitado
- Ordenamiento externo en discos
- Ordenamiento externo en cintas

Ordenamiento en cintas

- Los métodos para ordenar en cintas son parecidos a los métodos para ordenar en discos.
- El archivo original se distribuye en corridas ordenadas y luego se mezclan las corridas para construir el archivo completo.
- Una diferencia fundamental es que el método de ordenamiento usado para crear las corridas puede ser el reemplazo directo.

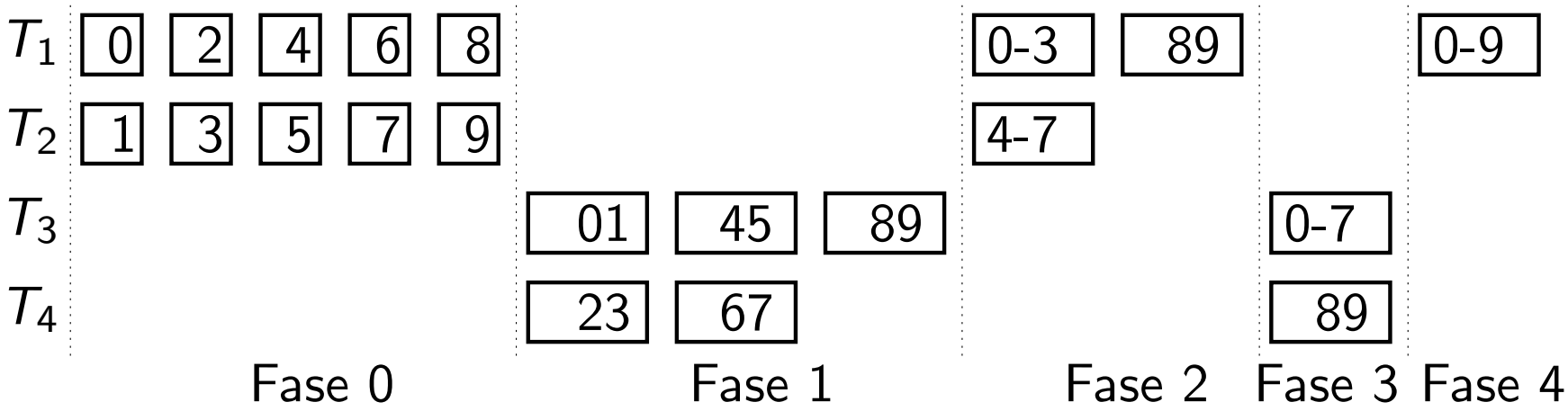
Reemplazo directo

- El reemplazo directo tiende a crear corridas más largas que las que se pueden crear con ordenamiento por montículo.
- El reemplazo directo no es muy bueno en discos porque incrementa el número de búsquedas.
- En cintas eso no es problema porque siempre suponemos que tenemos dos o más de ellas.
- Supondremos que ya hemos creado las corridas.

Mezcla balanceada de dos vías

- Este método requiere que las corridas estén distribuidas en dos cintas.
- A cada paso de la mezcla (excepto el último) la salida estará distribuida en dos cintas.
- Se necesitan cuatro cintas para poder aplicarlo.

Ejemplo de mezcla de dos vías



Diez corridas y cuatro fases

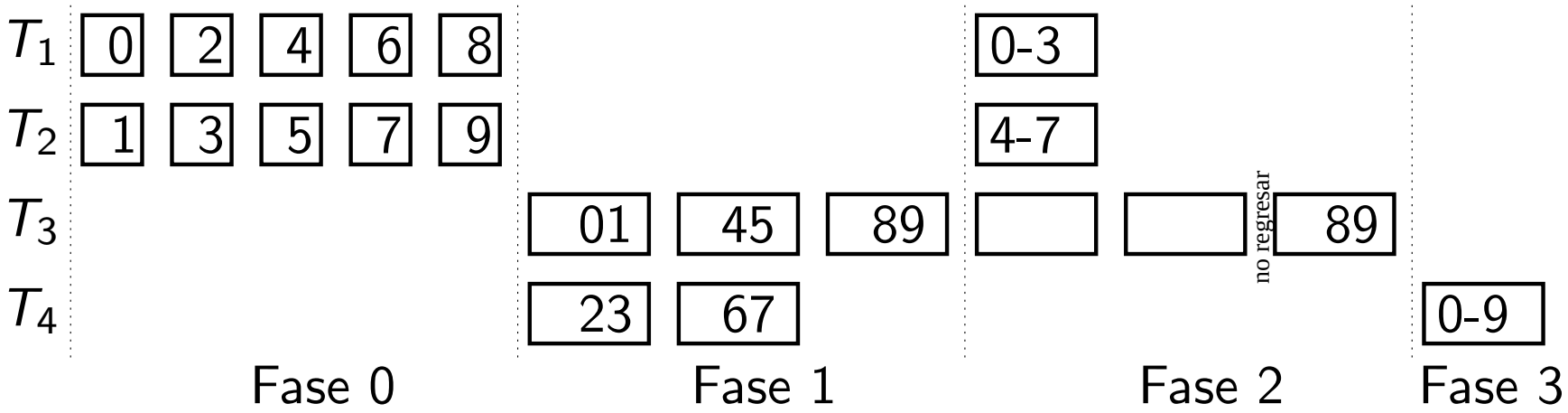
Número de fases

- Debido a que no hay búsquedas, el tiempo se mide en términos de cuántas veces leemos y escribimos los datos, así como cuántas veces rebobinamos las cintas.
- En el ejemplo esto lo hicimos cuatro veces.
- Si hay k corridas se requieren $\lceil \log_2 k \rceil$ fases.

Mezcla balanceada de n vías

- Este método es similar al anterior, excepto que se tienen n cintas de entrada y n cintas de salida en cada fase.
- En este caso se requieren $\lceil \log_n k \rceil$ fases.
- Con este método podremos ver fácilmente cómo se puede mejorar.

Ejemplo de mezcla de n vías



Diez corridas y tres fases

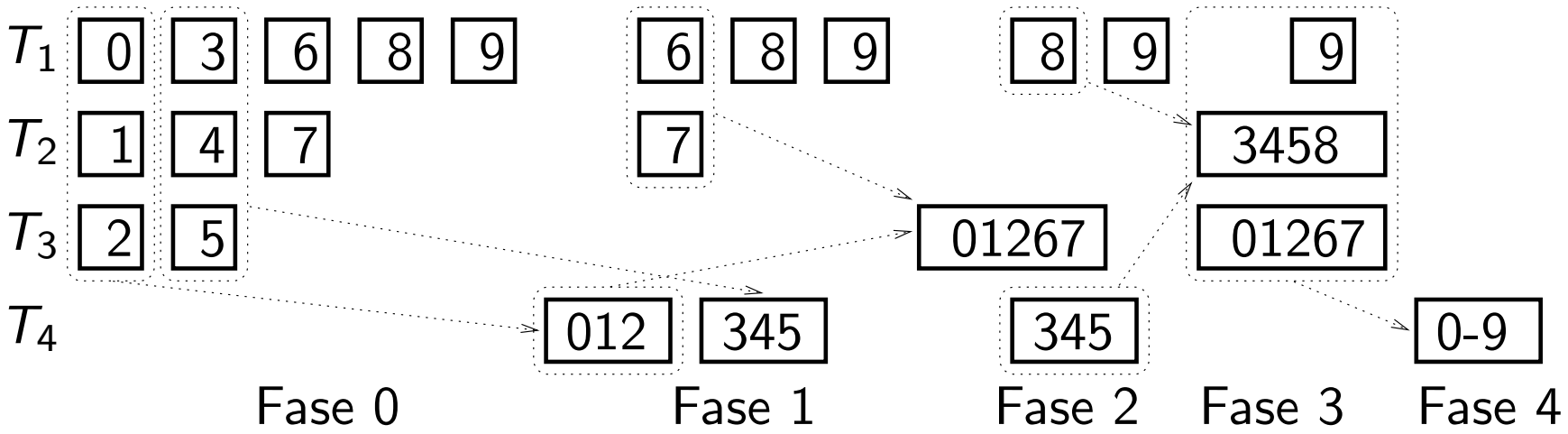
Mezcla polifásica

- La idea es tratar de evitar al máximo la simple copia de datos.
- Esto logrará también evitar rebobinar las cintas.
- Existen dos métodos que usan esta idea llamados mezcla polifásica y en cascada.

Características

- La distribución inicial de las corridas es tal que al menos la primera mezcla es de $n-1$ vías si se tienen n cintas.
- La distribución de las corridas es tal que las cintas suelen contener un número distinto de corridas en cada fase.

Ejemplo de mezcla polifásica



Al inicio hay 5, 3 y 2 corridas

Preguntas

- ¿Cómo se debe escoger la distribución inicial de las corridas que nos lleve a un patrón de mezcla eficiente?
- ¿Existirá algún algoritmo que encuentre estos patrones así como las distribuciones iniciales?
- Si tenemos k corridas y n cintas ¿existirá alguna manera de calcular la forma óptima de mezclar de modo que se pueda comparar nuestro algoritmo contra lo óptimo?

Part VI

Árboles B y B^+

Contenido

13 Árboles B
- Indexación con árboles binarios
- Árboles binarios paginados
- Indexación multinivel
- Operaciones en árboles B
- Propiedades de un árbol B
- Borrado y otros detalles en árboles B
- Árboles B^*

14 Árboles B^+

Índices demasiado grandes

- ¿Qué hacer si un índice es demasiado grande como para caber en memoria?
- Es obvio que parte del índice deberá estar en el disco y, por lo tanto, el índice será lento.
- Si queremos resolver este problema:
  1. La búsqueda en el índice debe ser más rápida que la búsqueda binaria.
  2. Insertar y borrar una clave debe ser tan rápido como la búsqueda.

Insuficiencia de los árboles binarios

- Una estructura de árbol balanceado es fundamental para resolver la segunda parte.
- Un árbol binario de búsqueda no sería suficiente pues su estructura puede degenerar en una lista.
- Los árboles balanceados AVL, rojinegros y 2-3-4 no tienen este problema, pero no resuelven la primera parte.

Problemas de las estructuras binarias

- El uso del disco es demasiado ineficiente.
- Cada lectura de un nodo puede producir una búsqueda en el disco.
- La lectura sólo trae tres cosas útiles: la clave actual y las direcciones de los subárboles izquierdo y derecho.
- Como una lectura del disco lee al menos un sector es muy importante que en ese sector almacenemos tanta información útil como sea posible.

Contenido

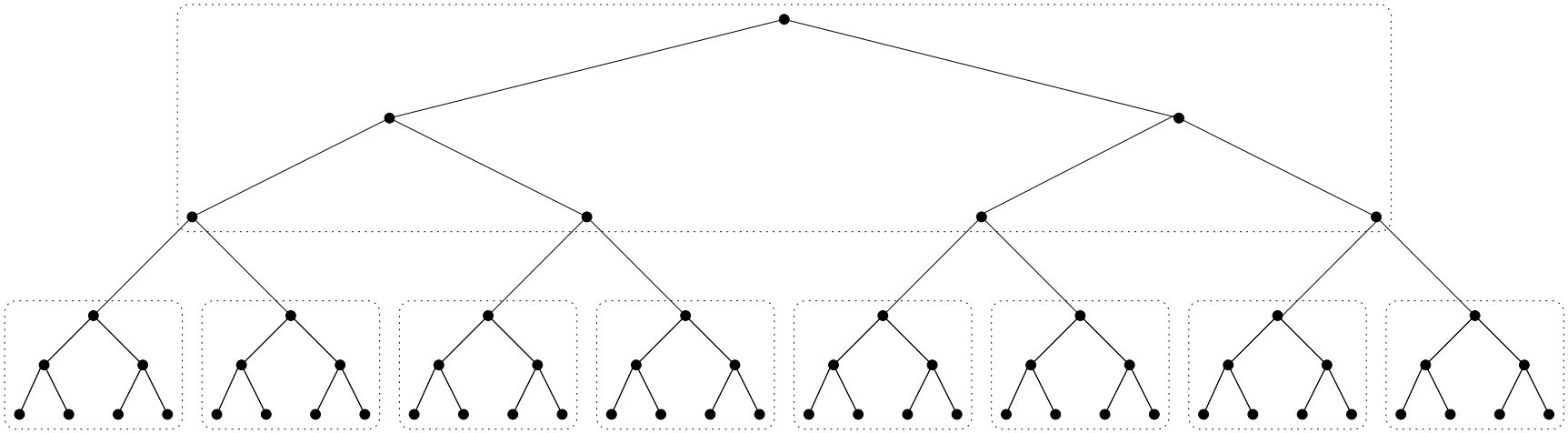
13 Árboles B
- Indexación con árboles binarios
- Árboles binarios paginados
- Indexación multinivel
- Operaciones en árboles B
- Propiedades de un árbol B
- Borrado y otros detalles en árboles B
- Árboles B^*

14 Árboles B^+

Árboles binarios paginados

- Los árboles binarios paginados intentan resolver esta situación colocando tantos nodos adyacentes de un árbol binario como sea posible dentro de un sector.
- Por ejemplo, si se pueden escribir 7 nodos en un sector entonces será posible escribir todos los nodos de un árbol balanceado con 63 nodos en 9 sectores.
- La búsqueda de cualquier nodo sólo requerirá 2 lecturas del disco (en vez de 6).

Un árbol binario paginado



63 nodos en 9 sectores

Continuación del ejemplo

- Si agregaramos otros dos niveles de sectores podríamos encontrar cualquiera de 4095 nodos con sólo 4 búsquedas.
- Uno esperaría que cupieran más de 7 nodos en un sector.
- Por ejemplo, si en cada lectura del disco podemos leer 511 nodos entonces podemos encontrar cualquiera de 134217727 claves con sólo 3 lecturas.
- ¡Éste es el desempeño que buscamos!

Desempeño de árboles paginados

- Si podemos leer k nodos en una lectura del disco entonces podremos encontrar cualquiera de n claves en un máximo de $\log_{k+1}(n+1)$ lecturas de disco.
- Sin embargo, nada es gratis.
- Aun puede ser que nuestro árbol binario quede desbalanceado, por lo que la búsqueda se puede volver secuencial.

Problemas

- ¿Cómo aseguramos que las claves en el sector raíz distribuyen de forma adecuada a las demás claves?
- ¿Cómo evitamos que en un mismo sector queden claves que no distribuyen bien a las claves debajo de ellos?
- ¿Cómo garantizamos que cada sector contenga al menos un número mínimo de claves?

Contenido

13 Árboles B
- Indexación con árboles binarios
- Árboles binarios paginados
- Indexación multinivel
- Operaciones en árboles B
- Propiedades de un árbol B
- Borrado y otros detalles en árboles B
- Árboles B^*

14 Árboles B^+

Índices en árbol

- Podemos pensar en índices estructurados como árboles.
- Esto nos puede proveer de la capacidad de encontrar cualquiera de una gran cantidad de registros con pocas lecturas del disco.
- Sin embargo, esto nos lleva a la posibilidad de requerir tiempo lineal para la inserción de una clave.
- Esto resulta fatal si además lo queremos hacer en el disco.

Contenido

13 Árboles B
- Indexación con árboles binarios
- Árboles binarios paginados
- Indexación multinivel
- Operaciones en árboles B
- Propiedades de un árbol B
- Borrado y otros detalles en árboles B
- Árboles B^*

14 Árboles B^+

Árboles B

- Los árboles B resuelven el problema de la inserción y borrado lineal.
- Se han vuelto la forma estándar de representar un índice.
- La solución requiere de dos ideas:
  - No se requiere que los registros del índice estén llenos.
  - Las claves que van a un registro lleno no se envían a otro registro, sino que éste se divide en dos registros llenos a la mitad.

Orden del árbol B

- Cada nodo de un árbol B es un registro del índice y puede contener hasta un número máximo de claves.
- Este número es el orden del árbol B.
- También un número mínimo de claves.
- Este número suele ser la mitad del orden.
- Excepto la raíz que puede llegar a tener una clave.
- El borrado mezcla dos registros en uno cuando sea necesario.

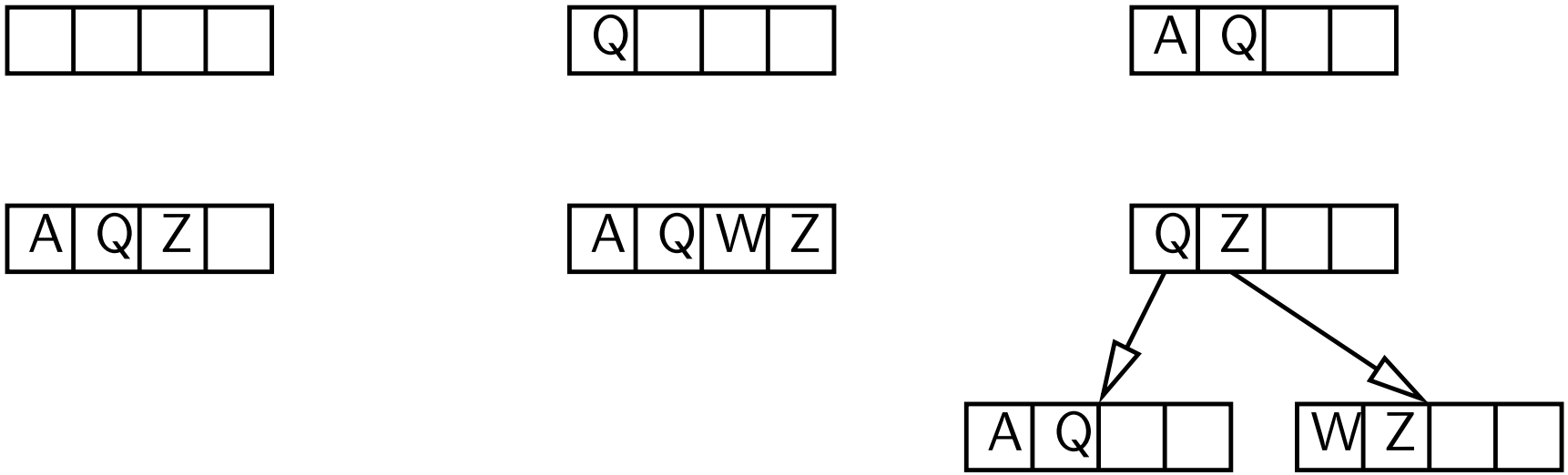
Inserción en un árbol B

- Se busca el registro donde debe ir la clave.
- Si se introduce una clave en un registro que no está lleno sólo se actualiza ese registro.
- A menos que la nueva clave resulte ser la más grande del registro.
- En este caso se deben actualizar también los niveles superiores del árbol.
- Esto tiene un costo acotado por la altura del árbol.

Inserción en un registro lleno

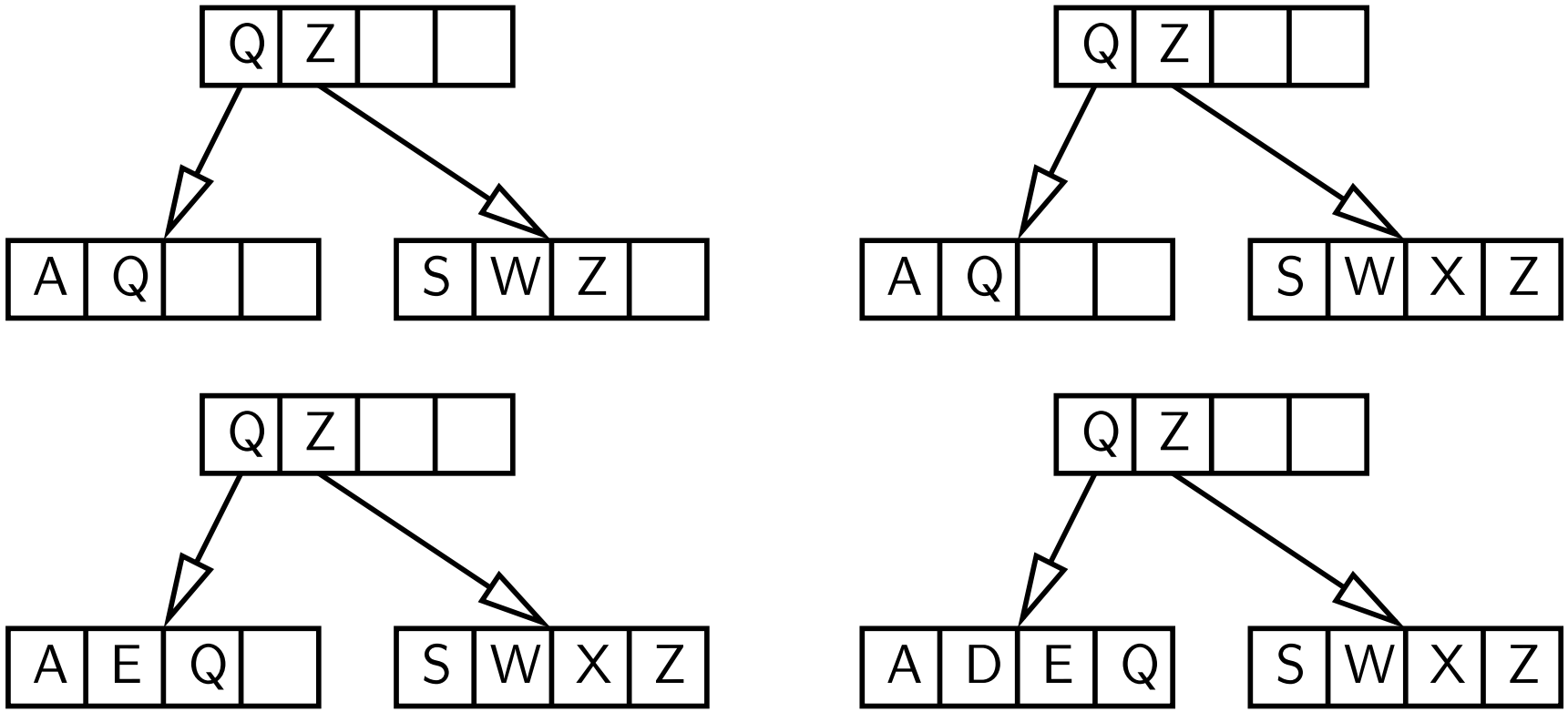
- En este caso el registro se divide en dos, cada parte con la mitad de las claves.
- Como se ha creado un nodo su clave más grande se debe insertar en el nodo superior.
- A esto se le llama la promoción de una clave.
- Esto puede causar que otros nodos se subdividan.
- Si se subdivide la raíz se agrega un nuevo nivel al árbol.
- El costo sigue acotado por la altura del árbol.

Ejemplo de inserción en un árbol B



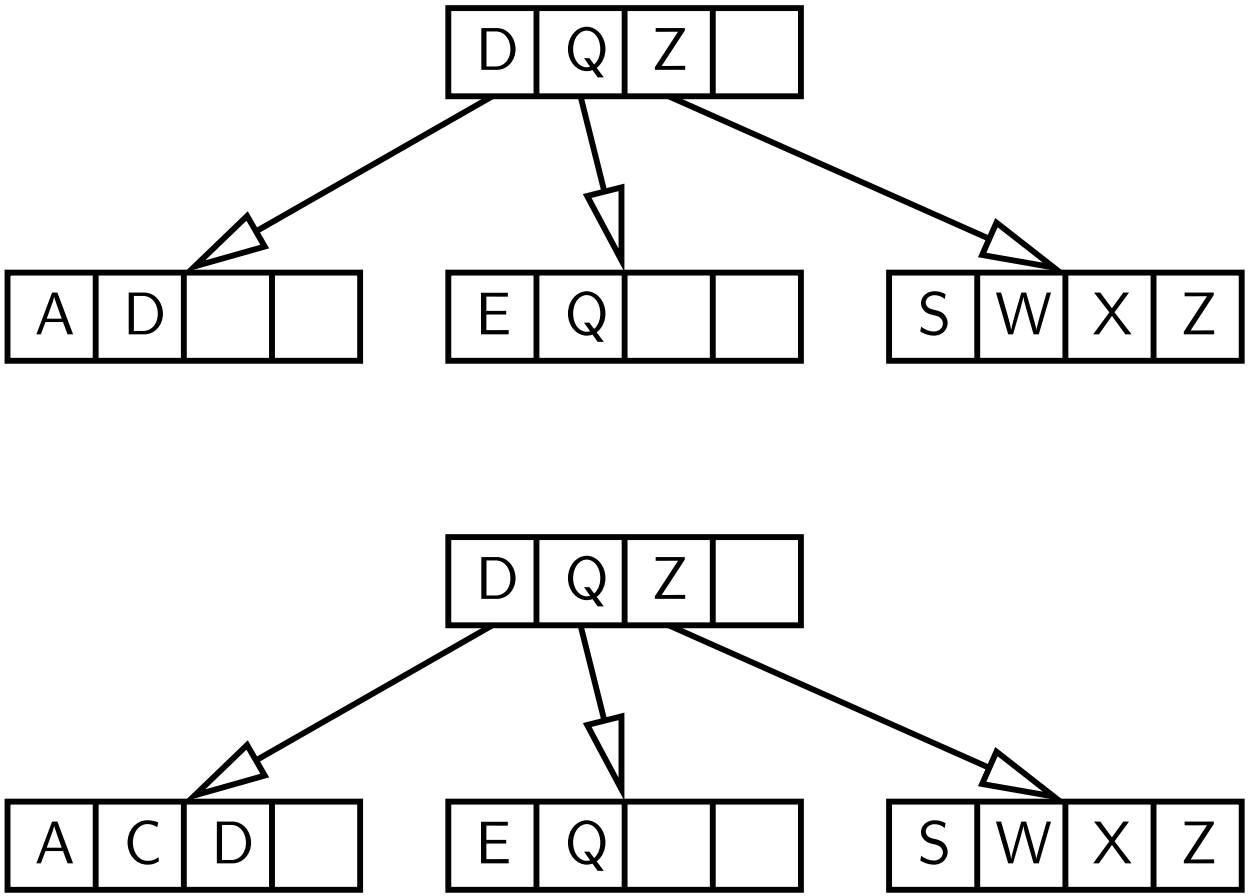
Inserción de Q, A, Z y W en un árbol B de orden 4

Ejemplo de inserción en un árbol B



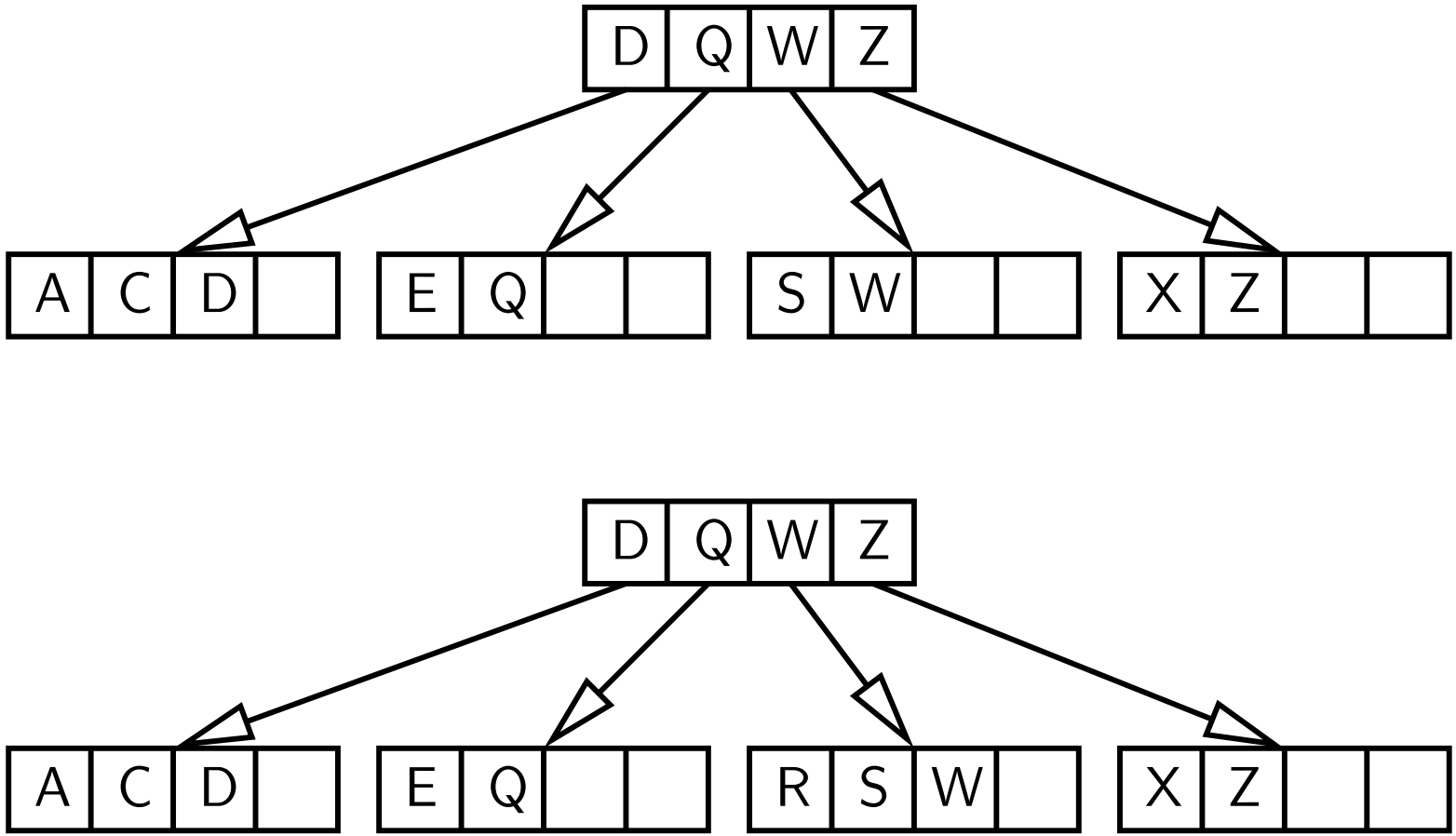
Inserción de S, X, E y D en un árbol B de orden 4

Ejemplo de inserción en un árbol B



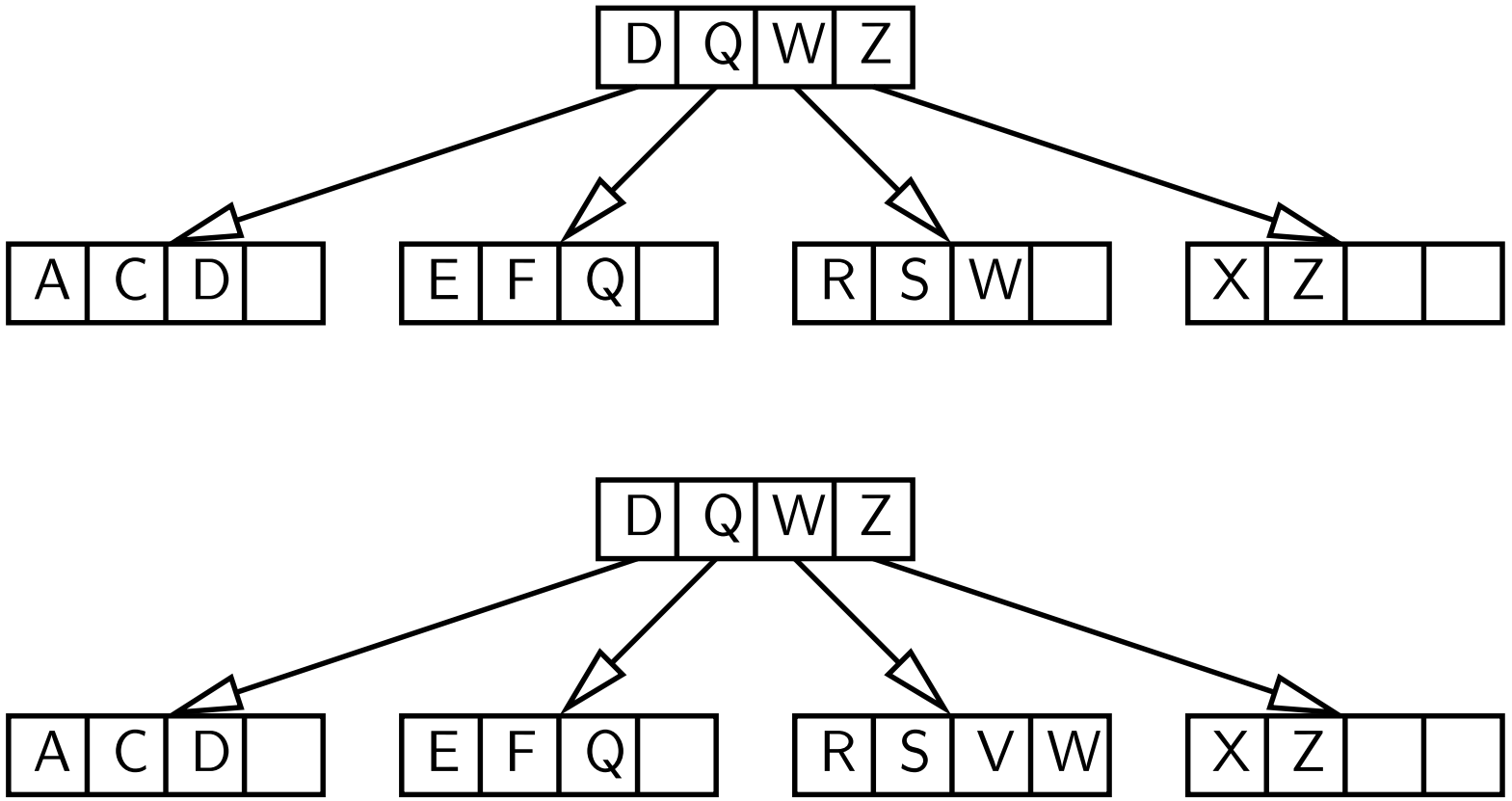
Inserción de C en un árbol B de orden 4

Ejemplo de inserción en un árbol B



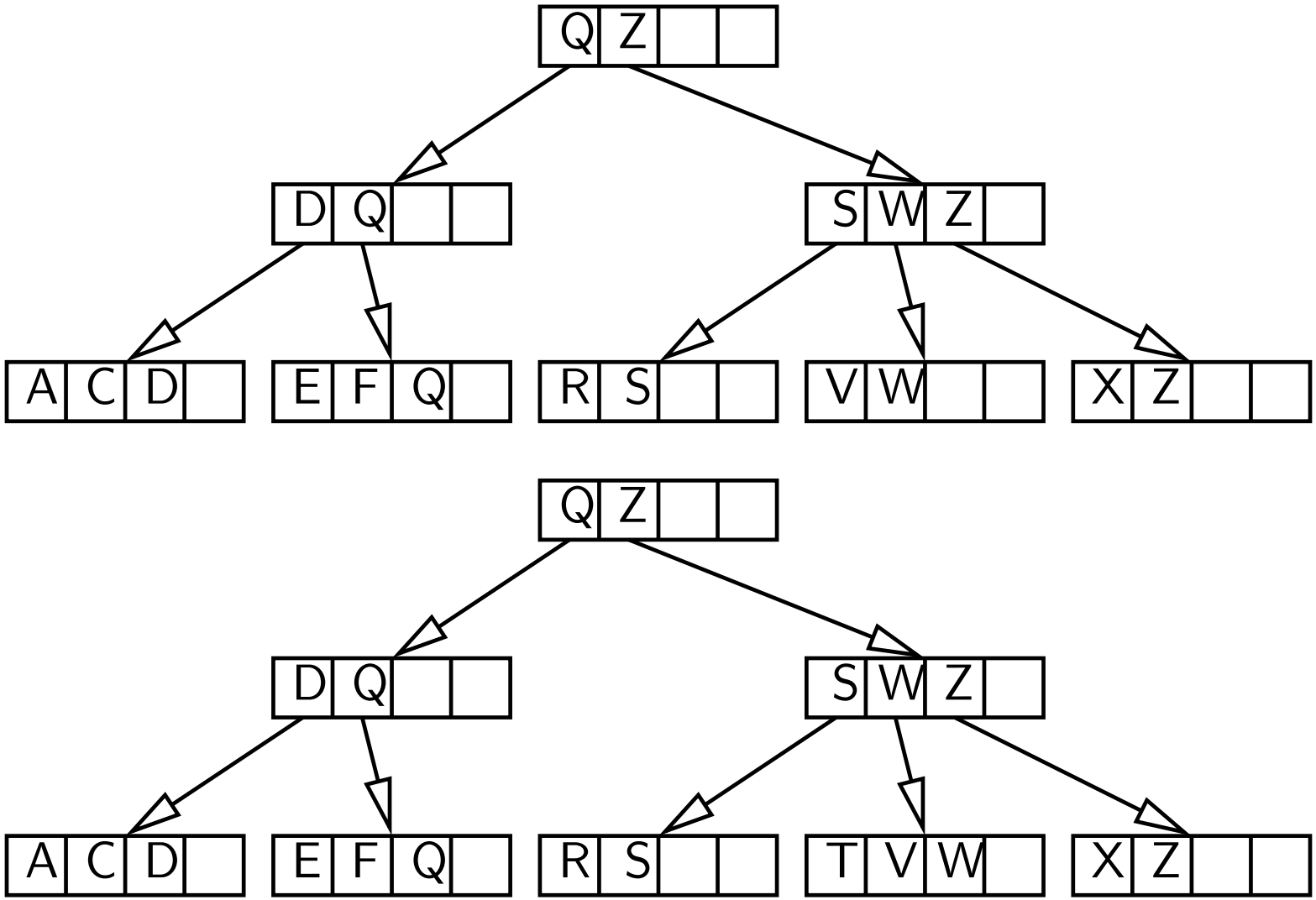
Inserción de R en un árbol B de orden 4

Ejemplo de inserción en un árbol B



Inserción de F y V en un árbol B de orden 4

Ejemplo de inserción en un árbol B



Inserción de T en un árbol B de orden 4

Contenido

13 Árboles B
- Indexación con árboles binarios
- Árboles binarios paginados
- Indexación multinivel
- Operaciones en árboles B
- Propiedades de un árbol B
- Borrado y otros detalles en árboles B
- Árboles B^*

14 Árboles B^+

Propiedades de un árbol B

- Cada registro índice tiene un máximo de m hijos (m es el orden).
- Cada registro índice, excepto por la raíz y las hojas, tiene al menos $m/2$ hijos.
- La raíz tiene cero o al menos dos hijos.
- Todas las hojas aparecen en el mismo nivel.
- El nivel de las hojas forma un índice completo y ordenado.

Búsqueda en el peor de los casos

- Note que el nivel d del árbol tiene al menos $2\lceil m/2 \rceil^{d-1}$ nodos.
- Si $n \geq 2\lceil m/2 \rceil^{d-1}$ y d es la altura del árbol entonces $d \leq 1 + \log_{\lceil m/2 \rceil}(n/2)$.
- Por lo tanto se requerirá un máximo de $\lfloor 1 + \log_{\lceil m/2 \rceil}(n/2) \rfloor$ búsquedas.
- Si $m = 2^9$ y $n = 2^{24}$ entonces $d \leq 3$.

Contenido

13 Árboles B
- Indexación con árboles binarios
- Árboles binarios paginados
- Indexación multinivel
- Operaciones en árboles B
- Propiedades de un árbol B
- Borrado y otros detalles en árboles B
- Árboles B^*

14 Árboles B^+

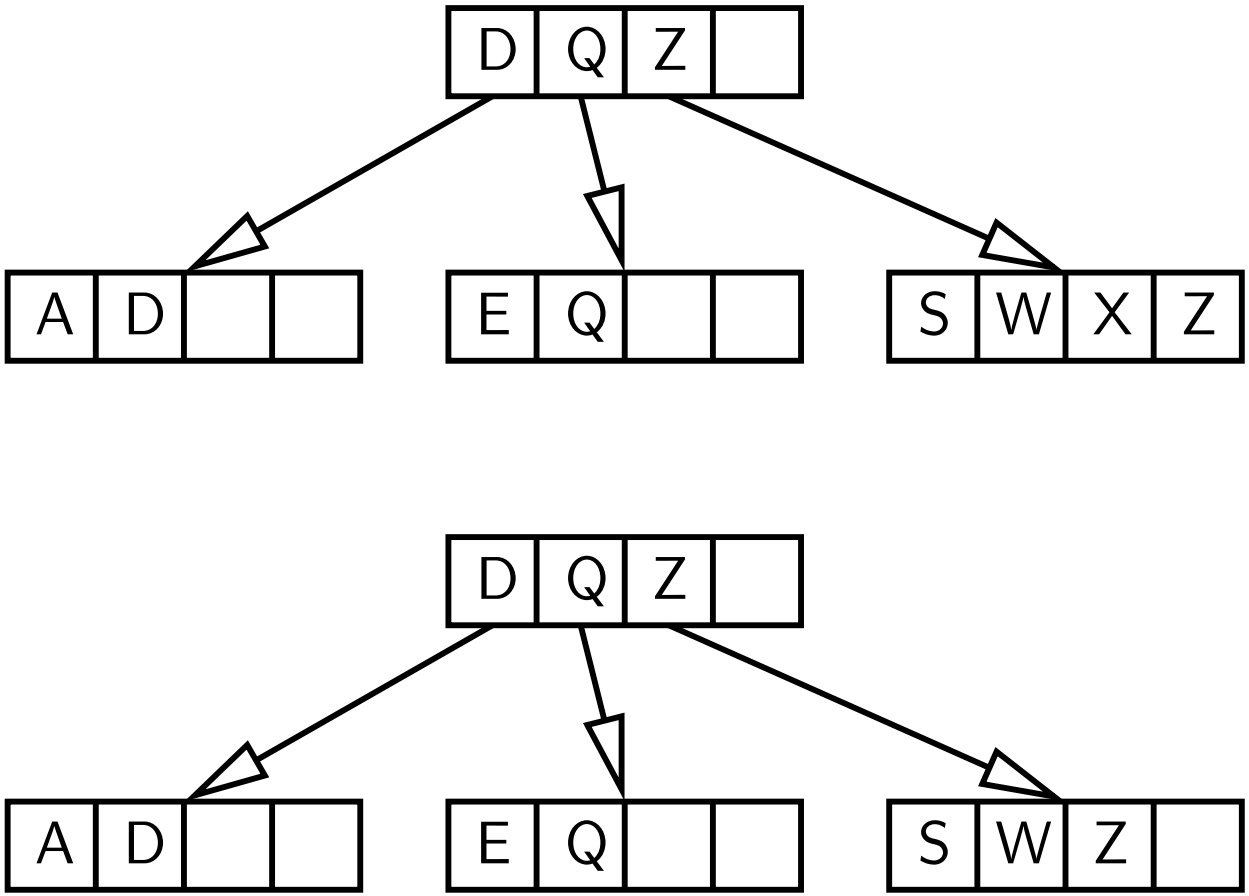
Borrado en árboles B

- Debemos buscar una forma de borrar claves que mantenga todas las propiedades de un árbol B mencionadas anteriormente.
- Es obvio que esas reglas van a depender de cuántas claves haya en el nodo del que se quiera borrar.
- También van a depender de cuál es la posición de la clave que se borra.

Borrando una clave de un nodo

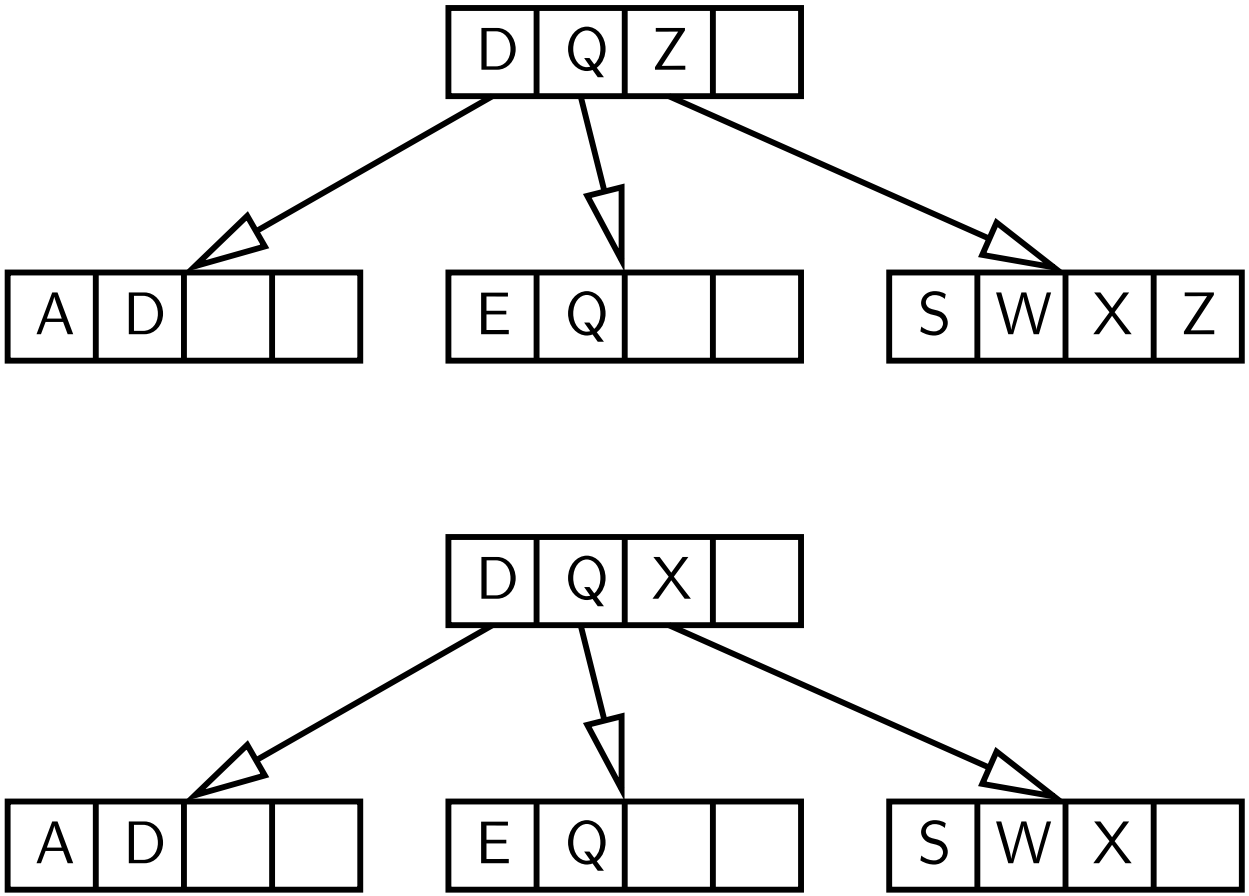
- Se desea borrar la clave C del nodo N.
- Si N tiene más claves que el número mínimo y C no es la clave más grande de N, entonces simplemente se borra C.
- Si N tiene más claves que el número mínimo pero C es la clave más grande de N, entonces se borra C y se modifican los niveles superiores del índice para reflejar la nueva clave más grande en N.

Ejemplo de borrado en un árbol B



Borrado de X

Ejemplo de borrado en un árbol B

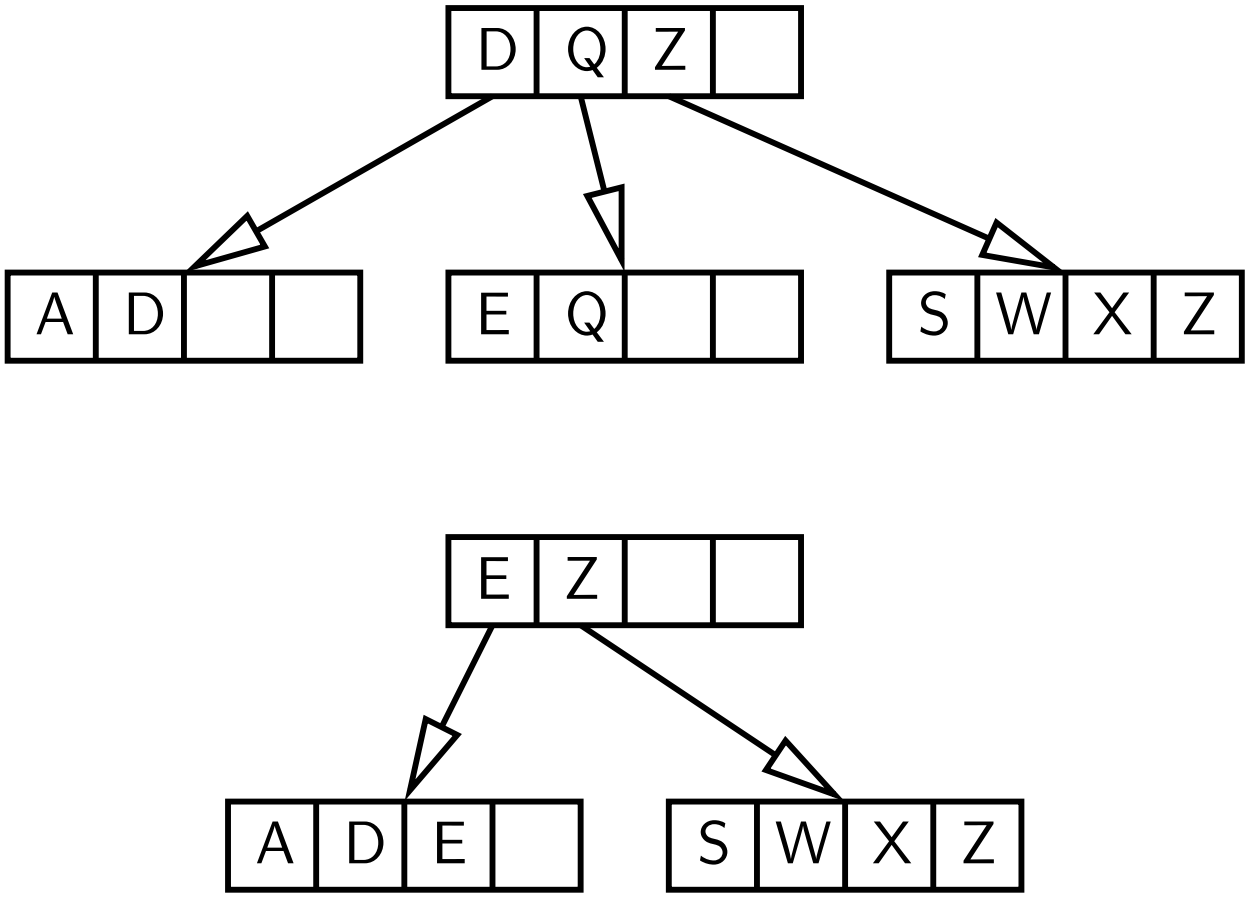


Borrado de Z

Borrando una clave con mezcla

- Suponga que N tiene exactamente el número mínimo de claves y uno de los hermanos de N tiene pocas claves.
- Entonces se mezcla a N con ese hermano y se borra una clave del nodo padre de N.
- ¿Qué quiere decir pocas claves?

Ejemplo de borrado con mezcla

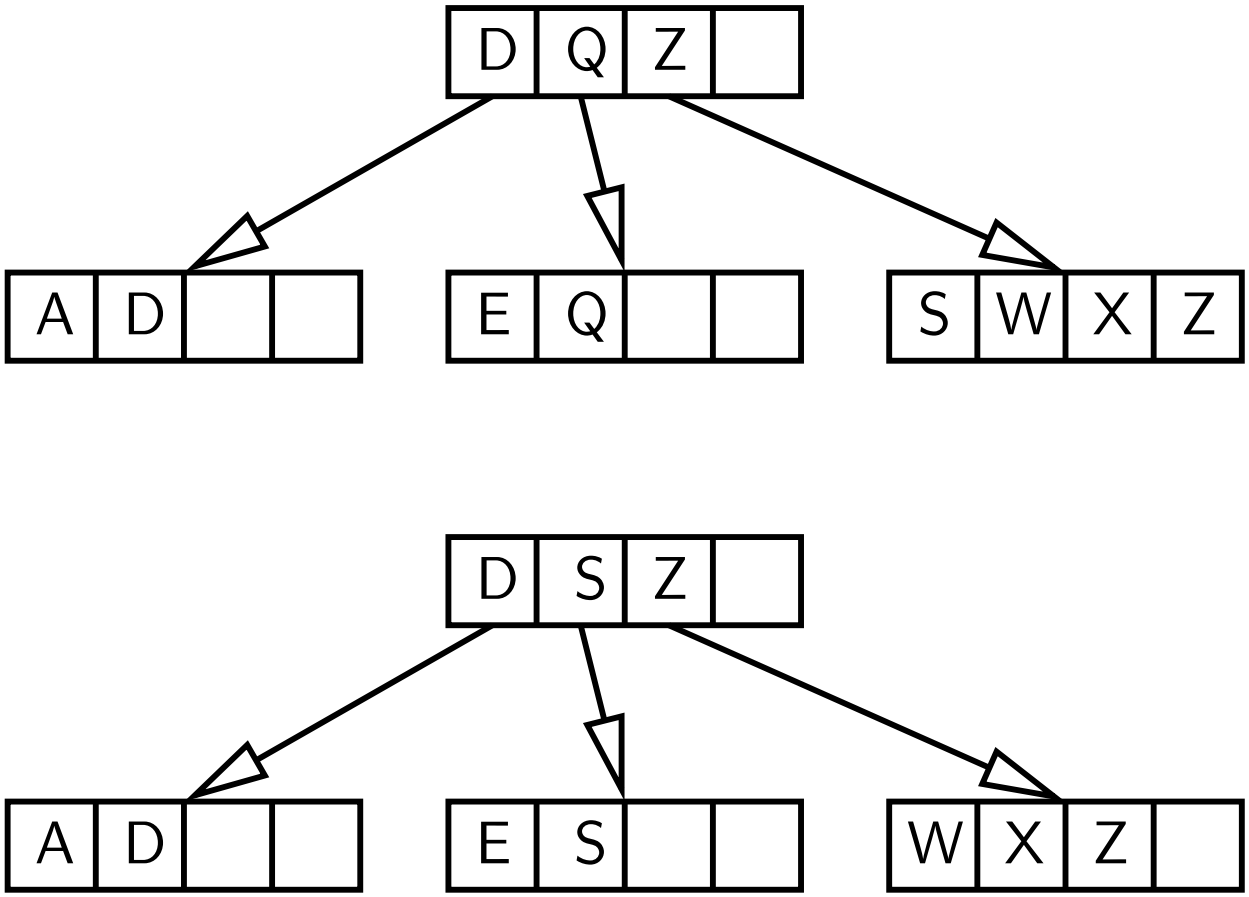


Borrado de Q

Borrando una clave con redistribución

- Suponga que N tiene exactamente el número mínimo de claves y uno de los hermanos de N tiene claves extras.
- Entonces se mueven algunas de las claves del hermano a N y se modifican los niveles superiores del índice para reflejar las claves más grandes de los nodos afectados.
- ¿Qué quiere decir claves extras?

Ejemplo de borrado con redistribución

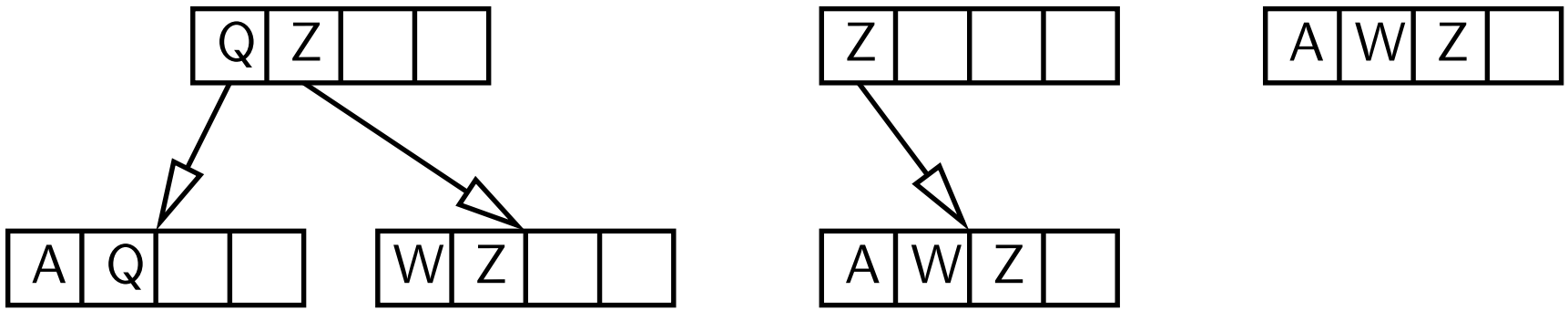


Borrado de Q

Casos especiales

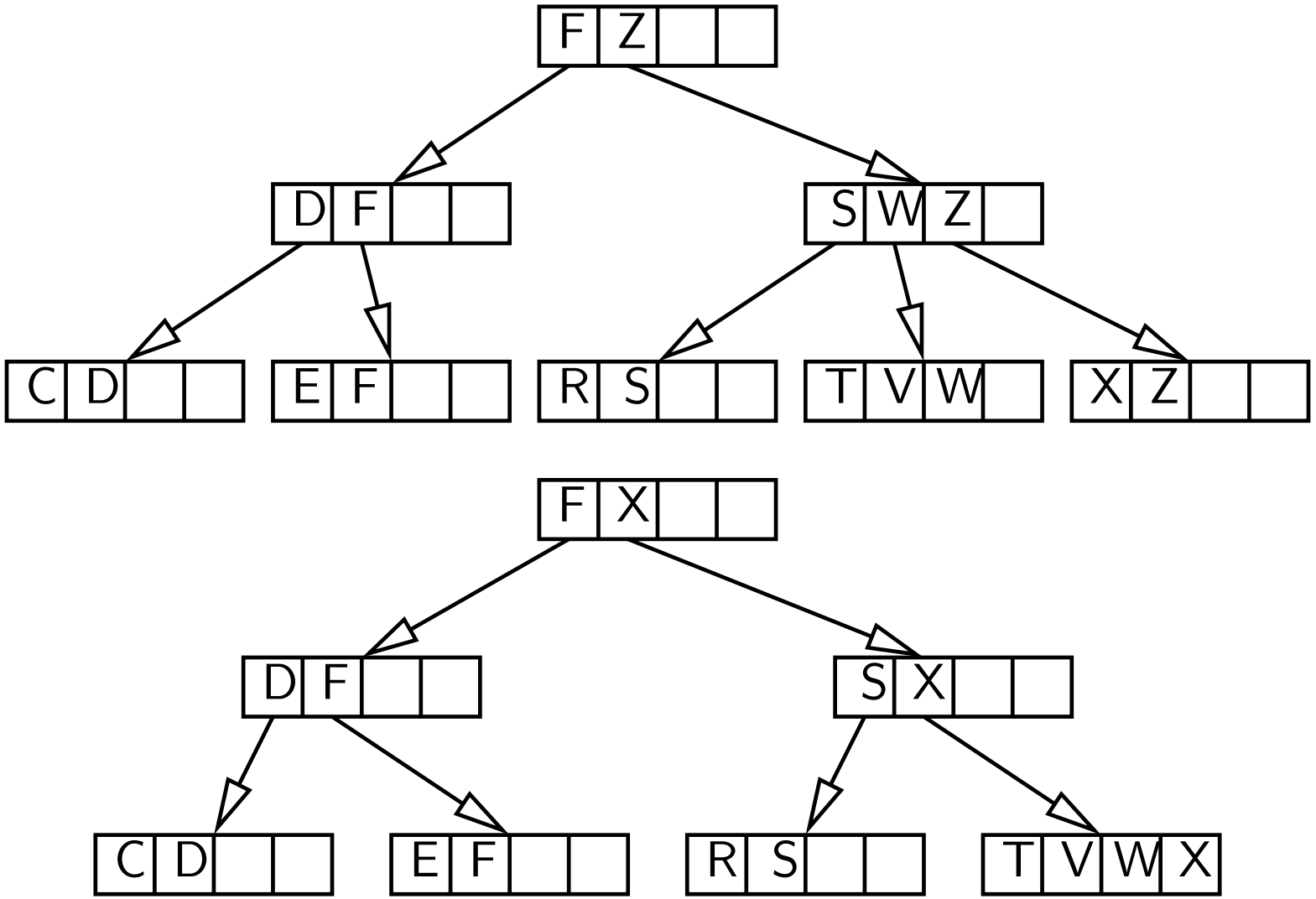
- Si la raíz termina con sólo una clave y un hijo, ésta se puede eliminar y su único hijo se vuelve la nueva raíz.
- En este caso el árbol decrece de altura.
- Es posible que se pueda aplicar la mezcla y la redistribución de forma simultanea.
- Cuando éste sea el caso, se puede aplicar cualquiera de las dos reglas.

Ejemplo de borrado de la raíz



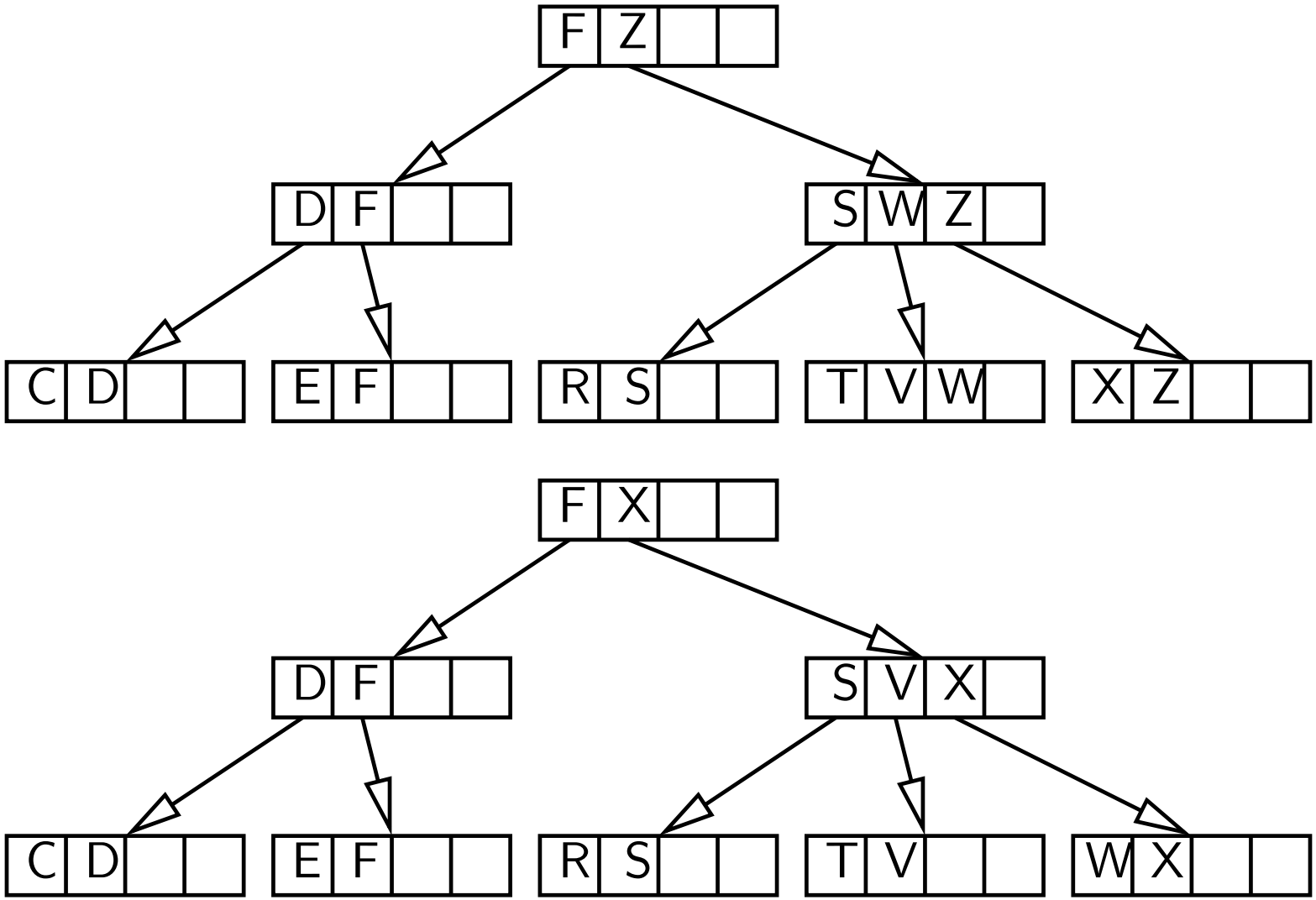
Borrado de Q

Ejemplo de borrado en un árbol B



Borrado de Z con mezcla

Ejemplo de borrado en un árbol B



Borrado de Z con redistribución

Redistribución durante la inserción

- Uno podría imaginar usar la redistribución al insertar una clave.
- Esto evitaría crear nodos nuevos.
- Como consecuencia mejoraría el uso del espacio en el árbol B.
- Algunos estudios empíricos sugieren redistribuir a menos que ambos hermanos del nodo estén llenos.

Contenido

13 Árboles B
- Indexación con árboles binarios
- Árboles binarios paginados
- Indexación multinivel
- Operaciones en árboles B
- Propiedades de un árbol B
- Borrado y otros detalles en árboles B
- Árboles B^*

14 Árboles B^+

Árboles B^*

- Knuth sugirió una modificación de los árboles B basada en la regla de redistribuir cuando se pueda.
- La idea es redistribuir las claves de dos hermanos en tres (uno de ellos nuevo) cuando no se pueda redistribuir.
- Se les llama árboles B^*.

Propiedades de los árboles B^*

- Cada nodo tiene un máximo de m hijos.
- Cada nodo (excepto la raíz) tiene al menos $\lceil \frac{2m-1}{3} \rceil$ hijos.
- La raíz tiene al menos dos hijos (a menos que sea hoja).
- Todas las hojas aparecen al mismo nivel.

Observaciones

- Los árboles B^* tienen algoritmos un poco más complicados que los árboles B por el trato especial a la raíz (que no tiene hermanos).
- Existen otras estrategias para disminuir el número de accesos al disco.
- Por ejemplo, los árboles B virtuales mantienen un buffer de nodos del árbol B.
- Como de costumbre, la eficiencia de este mecanismo dependerá de la estrategia de reutilización de buffers (LRU, etc.).

Contenido

13 Árboles B

14 Árboles B^+
- Conjuntos secuenciales y bloques
- Agregar un índice al conjunto secuencial
- Separadores
- Operaciones con árboles B^+

Acceso secuencial e indexado

- Hemos visto cómo estructurar un archivo para realizar operaciones secuenciales o para realizar búsquedas con índices.
- Sin embargo, nuestras soluciones no son capaces de resolver eficientemente el otro tipo de acceso.
- Existen muchos casos reales en los que se necesitan los dos tipos de acceso sobre los mismos archivos: sistemas de registro de estudiantes, de pago de servicios, etc.

Conjuntos secuenciales

- Ataquemos primero el problema de mantener un conjunto de registros en orden físico por clave mientras se agregan o eliminan registros.
- A este conjunto lo llamaremos secuencial.

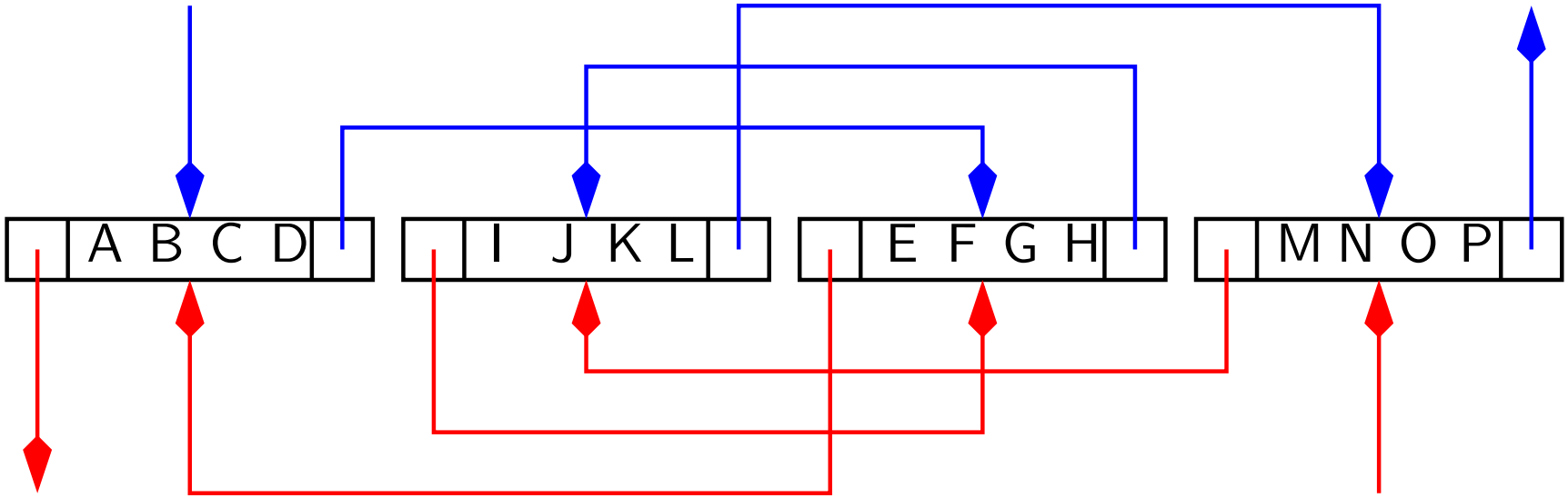
Bloques

- Una forma de localizar las modificaciones a un archivo que contiene varios conjuntos secuenciales es la de grabar cada uno de ellos en un bloque.
- Debido a que podemos leer y escribir bloques en una sola operación, se volverán nuestra unidad de entrada y salida.
- Una vez que se lee un bloque, todos los registros contenidos en él están en memoria, donde se les puede modificar rápidamente.

Lista de bloques

- Como los bloques lógicamente consecutivos no son necesariamente físicamente consecutivos también contendrán apuntadores a sus bloques adyacentes.
- Insertar o borrar registros de esta estructura se hace de forma similar a como se hace en un árbol B (excepto que ésta es una lista).
- La inserción puede causar sobrecupo.
- El borrado puede dejar un bloque muy vacío.

Ejemplo de lista de bloques



Bloques consecutivos físicamente

Observaciones

- Esta estructura nos permite mantener los registros ordenados sin la necesidad de reordenar todo el archivo.
- Esto no es gratuito:
  - Nuestro nuevo archivo usa más espacio que si todos los registros estuvieran ordenados.
  - El orden de los registros no sigue una secuencia física.
- El tamaño del bloque es crucial: no debe ser ni muy grande ni muy pequeño.

Contenido

13 Árboles B

14 Árboles B^+
- Conjuntos secuenciales y bloques
- Agregar un índice al conjunto secuencial
- Separadores
- Operaciones con árboles B^+

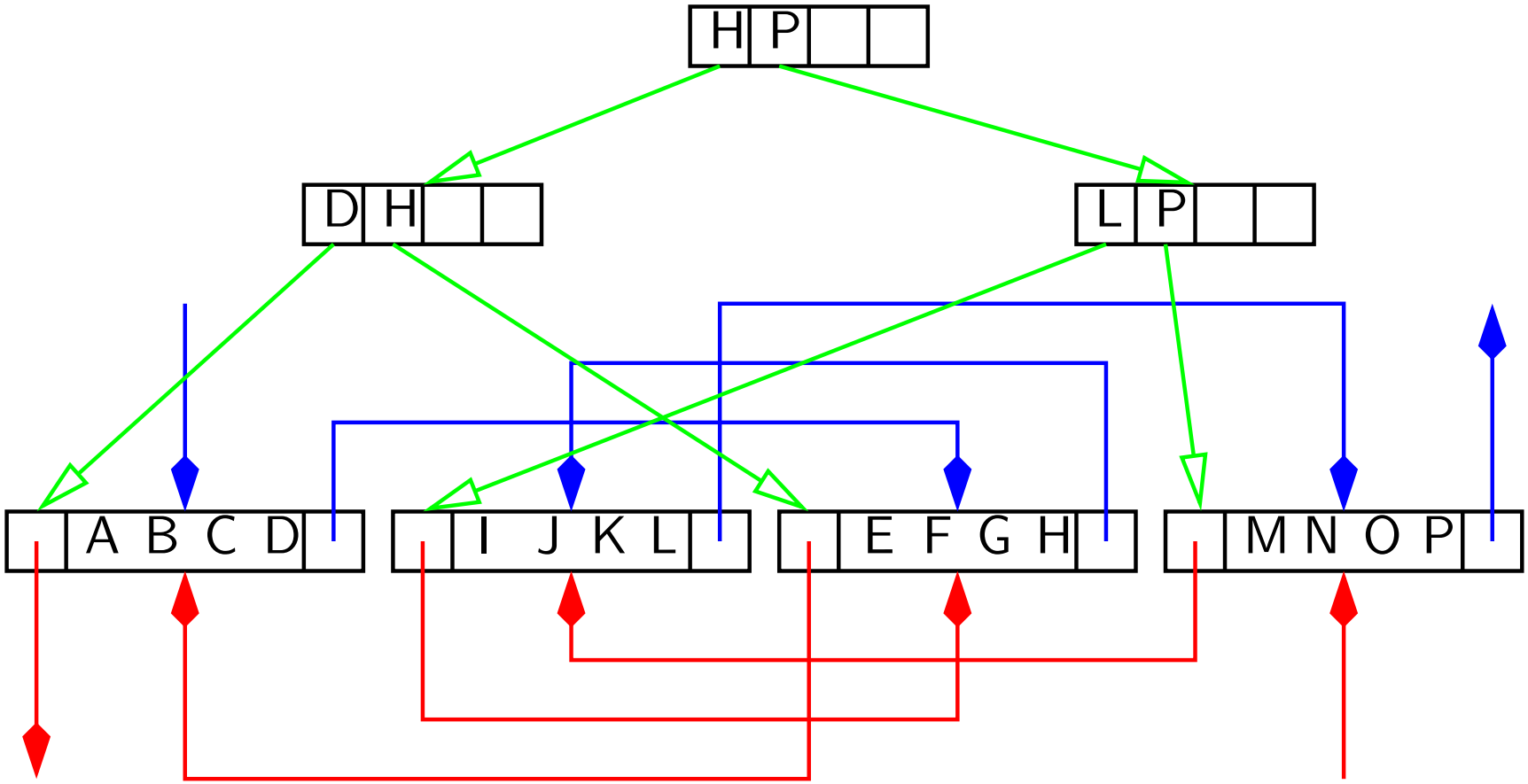
Agregar un índice simple

- Podemos notar que cada bloque contiene un rango de las claves.
- Si podemos separar esos rangos entonces podemos saber dónde debiera estar una clave dada.
- Por ejemplo, podemos usar la clave más grande en cada bloque para identificar al bloque completo de un índice.

Ventajas y desventajas

- Esto resulta en un método simple si todo el índice cabe en memoria.
- Por otro lado, notemos que las inserciones y los borrados pueden requerir modificar el índice de forma secuencial, lo que puede ser demasiado caro.
- Esto, junto con la posibilidad de que el índice no quepa en memoria, nos lleva a pensar en una estructura de tipo árbol para el índice.

Ejemplo de árbol B^+



Lista de bloques y árbol índice

Contenido

13 Árboles B

14 Árboles B^+
- Conjuntos secuenciales y bloques
- Agregar un índice al conjunto secuencial
- Separadores
- Operaciones con árboles B^+

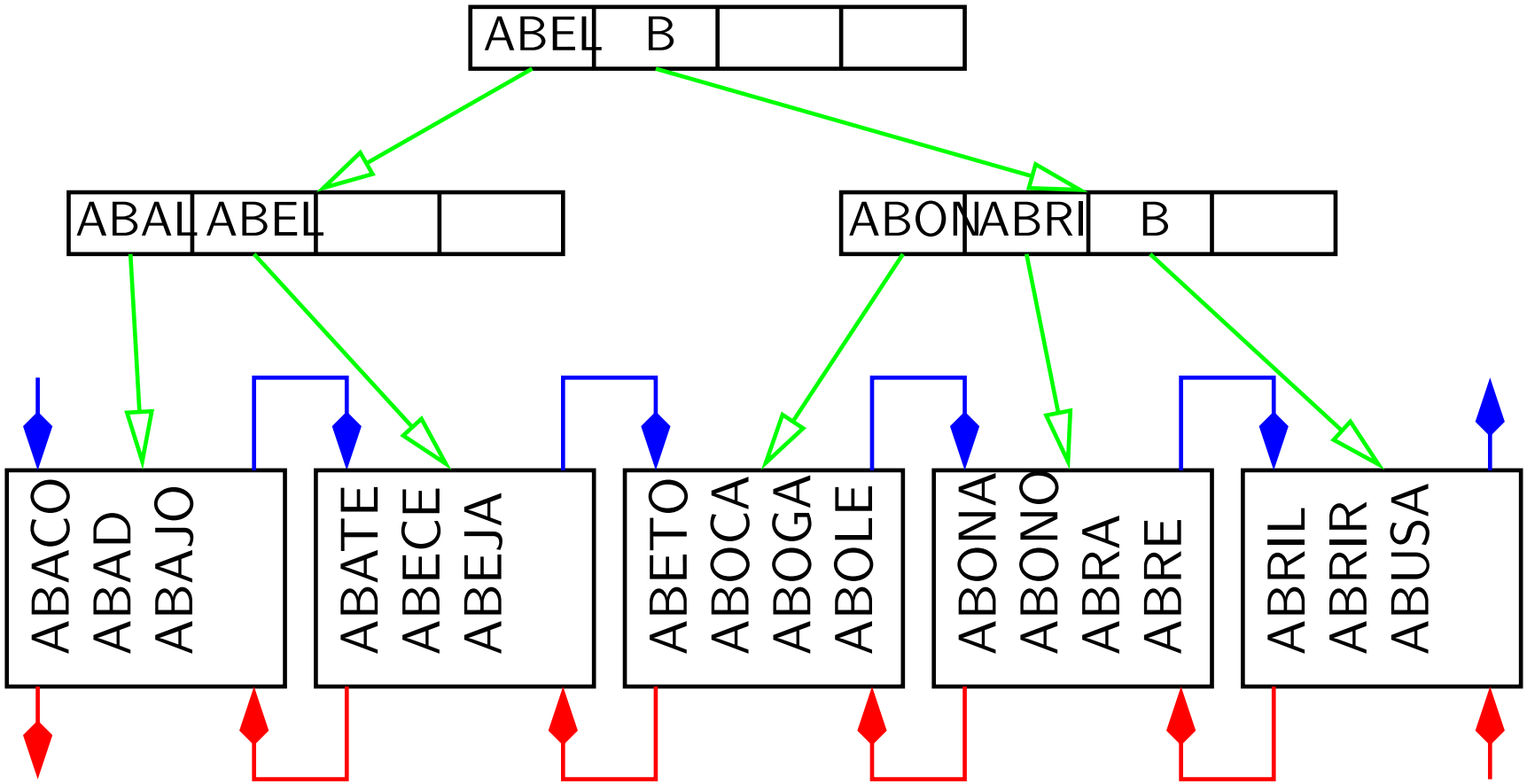
Separadores en lugar de claves

- Observe que el índice sólo nos sirve para llegar al bloque correcto.
- Para eso no se necesitan las claves, sino simples separadores.
- Una idea es la de usar lo que se conoce como separador más corto.
- Éste es una cadena de longitud mínima tal que es mayor que todas las cadenas del bloque izquierdo pero menor o igual a todas las cadenas del bloque derecho.

Separadores más cortos

- Ejemplo: `abrasivo` y `absoluto`.
- Observe que los separadores más cortos no son necesariamente únicos, pero cualquiera con esa propiedad nos podrá guiar al tomar decisiones.
- Ejemplo: `abajo` y `abeja`.
- A un árbol B de separadores más cortos junto con el conjunto de secuencias se le llama un árbol B^+ de prefijos simples.

Árbol B^+ de prefijos simples



Lista de bloques y árbol B de separadores

Contenido

13 Árboles B

14 Árboles B^+
- Conjuntos secuenciales y bloques
- Agregar un índice al conjunto secuencial
- Separadores
- Operaciones con árboles B^+

Borrado e inserción simples

- Los cambios más sencillos que podemos hacer en un árbol B^+ son los de borrado e inserción de claves que no resulten en ninguna mezcla ni redistribución de bloques.
- En estos casos no se necesita cambiar el contenido del árbol B puesto que los separadores que éste contiene siguen siendo válidos.

Cambio en el número de bloques

- La situación es un poco más complicada cuando los cambios en el árbol B^+ cambian el número de bloques.
- Si tenemos más bloques necesitamos más separadores y viceversa.
- Por lo tanto este tipo de cambios sí alteran el contenido del árbol B.

Tres casos

- Si se divide algún bloque en el conjunto de secuencias se debe insertar un nuevo separador en el conjunto índice.
- Si se mezclan dos bloques en el conjunto de secuencias se debe eliminar un separador del conjunto índice.
- Si se redistribuyen registros entre dos bloques en el conjunto de secuencias se debe cambiar el valor de uno de los separadores del conjunto índice.

Recordatorios

- No sobra hacer los siguientes recordatorios:
- Toda inserción o borrado modifica el conjunto de secuencias.
- Pero sólo algunas de estas operaciones modifican el conjunto índice.
- Y en este caso lo hacen como si fuera un simple árbol B.

Tamaño de los nodos

- Aunque no es necesario, el tamaño del nodo del árbol B suele ser el mismo que el tamaño del nodo del conjunto de secuencias.
- Esto suele ser buena idea por varias razones:
- Se puede usar el mismo conjunto de buffers para ambos tipos de nodos.
- Ambos tipos de bloques pueden aparecer en el mismo archivo.
- El tamaño se escogió de acuerdo a las características del disco y la memoria.

¿Porqué separadores más cortos?

- La razón fue la posibilidad de que cupieran tantos separadores como fuera posible en un nodo del árbol B.
- Pero entonces no es cierto que todos los nodos de este árbol tienen el mismo número de separadores.
- Por lo tanto necesitamos que la estructura del nodo nos permita tener un número variable de registros de tamaño variable.

Estructura del nodo índice

```
NNLLS1S2...SNI1I2...INR1R2...RN
```

- `NN` es el número de separadores (f).
- `LL` es la longitud total de los separadores (f).
- `S1, ..., SN` son los `NN` separadores (v).
- `I1, ..., IN` forman un índice de los separadores en el nodo (f).
- `R1, ..., RN` son los números de bloque relativo de los nodos hijo (f).

Conclusiones

- No es trivial determinar cuándo se debe dividir, mezclar o redistribuir.
- Los árboles B^+ simples difieren de los de prefijos en que los separadores son claves.
- Esto obliga a usar un poco más de espacio en los nodos y de tiempo de procesamiento.
- Pero es más fácil que trabajar con la estructura variable de los nodos.
- También se usan si los separadores no son muy cortos.

Part VII

Dispersión

Contenido

15 Introducción a la dispersión
- Dispersión y colisiones
- Una función de dispersión sencilla
- Otras funciones de dispersión
- Memoria y densidad de empacado

16 Técnicas de dispersión

Tiempo de acceso

- Hasta ahora hemos logrado acceso a un archivo con n claves en tiempo:
- $O(n)$ con búsqueda secuencial.
- $O(\log_k n)$ con árboles B.
- Lo que realmente queremos es acceso en tiempo constante $O(1)$.
- Recuerde que el tiempo se mide en términos del número de búsquedas en el disco.

Dispersión

- Una función de dispersión es una función que transforma una clave en una dirección.
- A diferencia de un índice, es posible que dos claves se asocien con la misma dirección, a lo que se le llama colisión.
- Dos claves a las que se les asocia la misma dirección se llaman sinónimas.

Dispersión y colisiones

- Es extremadamente difícil diseñar funciones de dispersión que no produzcan colisiones.
- Por lo que en general debemos preocuparnos por cómo resolverlas.
- Ya sea escogiendo funciones que produzcan pocas colisiones.
- O jugando con la forma en la que se almacenan los sinónimos.

Resolución de colisiones

- Algunas formas de lograr esto son:
- Lograr una buena distribución de las claves. De preferencia uniforme.
- Usar más memoria.
- Poner más de una clave en una dirección. A esto se le llama una cubeta.

Contenido

15 Introducción a la dispersión
- Dispersión y colisiones
- Una función de dispersión sencilla
- Otras funciones de dispersión
- Memoria y densidad de empacado

16 Técnicas de dispersión

Funciones de dispersión sencillas

- Una función de dispersión sencilla trabaja así:
- Representa la clave en forma numérica.
- La divide en partes iguales y las suma.
- Divide el resultado entre un primo p y usa el residuo como dirección.
- Las sumas se pueden hacer módulo p.
- Las direcciones estarán en el rango 0 a $p - 1$.

Ejemplo de dispersión sencilla

- Considere la clave `EJEMPLO`.
- En ASCII es $69, 74, 69, 77, 80, 76, 79$.
- Sumamos las partes para obtener 524.
- Escogemos el número primo $p = 97$.
- Y obtenemos la dirección $524 \bmod 97 = 39$.

Ejemplo de colisiones

- Considere las siete claves `CERO`, `UNO`, `DOS`, `TRES`, `CUATRO`, `CINCO` y `SEIS`.
- Utilizando la función de dispersión sencilla con $p = 7$ obtenemos las siete direcciones $3, 4, 6, 3, 0, 0, 0$.
- Hay una colisión entre las claves `CERO` y `TRES` en la dirección 3.
- Hay otra colisión entre las claves `CUATRO`, `CINCO` y `SEIS` en la dirección 0.

Contenido

15 Introducción a la dispersión
- Dispersión y colisiones
- Una función de dispersión sencilla
- Otras funciones de dispersión
- Memoria y densidad de empacado

16 Técnicas de dispersión

Otros métodos de dispersión

- Usar una parte de la clave como dirección.
- Dividir y sumar sólo una parte de la clave.
- Dividir la clave completa entre un primo.
- Elevar la clave al cuadrado y tomar el centro.
- Cambiar de base la clave.

Otros ejemplos de dispersión

- Usar la `E` y obtener 69.
- Usar `EEPO` y obtener 6.
- Usar `EJEMPLO` y obtener 95.
- El centro del cuadrado es 170 y se obtiene 73.
- En base 10 se obtiene 57.

Probabilidad discreta

- Supongamos que la probabilidad de que una clave c sea asignada a una posición de memoria i es uniforme.
- Es decir, si hay n posiciones de memoria entonces $p(i) = \frac{1}{n}$.
- Suponga que se han asignado r claves.
- ¿Cuál es la probabilidad de que no le toquen claves a la posición de memoria i?
- ¿Cuál es la probabilidad de que le toque una clave a la posición de memoria i?

Distribución de Poisson

- La probabilidad de que le toquen k claves a la posición de memoria i después de asignar r claves es $p(i,k) = \binom{r}{k}(1-\frac{1}{n})^{r-k}\frac{1}{n^k}$.
- Esta distribución binomial es difícil de calcular exactamente si r y n son grandes.
- Sin embargo, la distribución de Poisson es una buena aproximación en este caso.
- $P(i,k) = \frac{1}{k!}(\frac{r}{n})^k e^{-r/n}$.

Comparación entre las dos fórmulas

- Como ejemplo supongamos que $r = n = 10$.
- $p(i, 0) = 0.3487$ y $P(i, 0) = 0.3679$.
- $p(i, 1) = 0.3874$ y $P(i, 1) = 0.3679$.
- $p(i, 2) = 0.1937$ y $P(i, 2) = 0.1839$.
- La probabilidad de que i tenga dos o más claves es $1 - p(i, 0) - p(i, 1) = 0.2639$ o $1 - P(i, 0) - P(i, 1) = 0.2642$.

Contenido

15 Introducción a la dispersión
- Dispersión y colisiones
- Una función de dispersión sencilla
- Otras funciones de dispersión
- Memoria y densidad de empacado

16 Técnicas de dispersión

Efecto de usar más memoria

- Si $r = 10$ y $n = 20$ entonces tenemos que $1 - p(i, 0) - p(i, 1) = 0.0862$ (se obtiene 0.0902 con Poisson).
- Si $r = 10$ y $n = 30$ entonces tenemos que $1 - p(i, 0) - p(i, 1) = 0.0418$ (se obtiene 0.0447 con Poisson).
- A la razón $\frac{r}{n}$ se le llama densidad de empacado y de ella depende la cantidad de colisiones esperadas.
- Los ejemplos vistos tienen $\frac{r}{n} = 1$, $\frac{1}{2}$ y $\frac{1}{3}$.

Contenido

15 Introducción a la dispersión

16 Técnicas de dispersión
- Sobreflujo progresivo
- Cubetas
- Borrado de registros
- Otras formas de resolver colisiones

Sobreflujo progresivo

- Este método es muy simple.
- Si una clave nueva tiene la misma dirección que una clave que ya está almacenada entonces se prueba en las siguientes direcciones hasta que se encuentre una dirección vacía.
- En caso de que se llegue al final del archivo se continúa desde el principio.

Ejemplo de sobreflujo progresivo

Inserción de CERO, UNO, DOS, TRES, CUATRO, CINCO y SEIS con claves 3, 4, 6, 3, 0, 0, 0

DIR	CERO	UNO	DOS	TRES	CUATRO	CINCO	SEIS
0	-	-	-	-	CUATRO	CUATRO	CUATRO
1	-	-	-	-	-	CINCO	CINCO
2	-	-	-	-	-	-	SEIS
3	CERO	CERO	CERO	CERO	CERO	CERO	CERO
4	-	UNO	UNO	UNO	UNO	UNO	UNO
5	-	-	-	TRES	TRES	TRES	TRES
6	-	-	DOS	DOS	DOS	DOS	DOS

Registros ausentes

- ¿Qué pasa cuando se busca un registro que no está en el archivo?
- Si se encuentra una dirección vacía ya se sabe que el registro nunca se ha puesto allí.
- Si el archivo está lleno la búsqueda regresará al punto inicial, momento en el que se sabe que el registro no está allí.

Longitud de la búsqueda

- Si el archivo está (casi) lleno la búsqueda puede ser intolerablemente lenta.
- Aunque no analizaremos esto, la longitud promedio de una búsqueda aumenta muy rápidamente con respecto a la densidad de empacado.
- Se considera que si la longitud promedio de una búsqueda es mayor que 2 entonces es inaceptable.

Contenido

15 Introducción a la dispersión

16 Técnicas de dispersión
- Sobreflujo progresivo
- Cubetas
- Borrado de registros
- Otras formas de resolver colisiones

Cubetas

- Recordemos que es igual de costoso leer un registro del disco que leer todo el bloque que lo contiene (al que llamaremos cubeta).
- Así, una idea es almacenar varios registros en una sola dirección de bloque, la cual se obtiene por dispersión.
- Cuando la cubeta se llene aun debemos preocuparnos por el sobreflujo, pero esto ocurrirá con mucha menor frecuencia que en el caso anterior.

Ejemplo de cubetas

Inserción de CERO, UNO, DOS, TRES, CUATRO, CINCO y SEIS con claves 3, 4, 6, 3, 0, 0, 0

D							
0	-	-	-	-	CUATRO	-	CUATRO CINCO	CUATRO CINC
1	-	-	-	-	-	-	-	-	SEIS	-
2	-	-	-	-	-	-	-	-	-	-
3	CERO	-	CERO	-	CERO	-	CERO TRES	CERO TRES	CERO TRES	CERO TRES
4	-	-	UNO	-	UNO	-	UNO	-	UNO	-	UNO	-
5	-	-	-	-	-	-	-	-	-	-	-	-
6	-	-	-	-	DOS	-	DOS	-	DOS	-	DOS	-

Tamaño de una cubeta

- Se puede analizar esta situación y descubrir que habrá menos sobreflujos si las cubetas son más grandes.
- Esto considerando igualdad de espacio total disponible.
- Sin embargo, queda claro que una cubeta no debe ser más grande que una pista, pues se tendría que mover el brazo para terminar de leer la cubeta completa.

Contenido

15 Introducción a la dispersión

16 Técnicas de dispersión
- Sobreflujo progresivo
- Cubetas
- Borrado de registros
- Otras formas de resolver colisiones

Borrado de registros

- Borrar un registro de un archivo de dispersión es mucho más complicado que agregar un registro por dos razones:
- El espacio liberado por el borrado no debe obstruir búsquedas futuras.
- Debiera ser posible reutilizar el espacio liberado para futuras inserciones.
- Esto es particularmente importante si se usa un método de resolución de colisiones por sobreflujo progresivo.

Lápidas

- Una forma sencilla de resolver ambos problemas es a través del uso de lápidas, es decir, marcas especiales que indiquen que allí había un registro pero ya no está.
- Es importante observar que sólo se debe poner una lápida donde se necesite.
- Por ejemplo, no se necesita si el siguiente espacio está vacío.
- También se debe notar que las lápidas crean nuevos problemas que no existían.

Ejemplo de lápidas

Borrado de CERO, CUATRO, CINCO, DOS, UNO y TRES con claves 3, 0, 0, 6, 4, 3.

DIR	CERO	CUATRO	CINCO	DOS	UNO	TRES
0	CUATRO	CUATRO	+	-	-	-	-
1	CINCO	CINCO	CINCO	-	-	-	-
2	-	-	-	-	-	-	-
3	CERO	+	+	+	+	+	-
4	UNO	UNO	UNO	UNO	UNO	+	-
5	TRES	TRES	TRES	TRES	TRES	TRES	-
6	DOS	DOS	DOS	DOS	-	-	-

Contenido

15 Introducción a la dispersión

16 Técnicas de dispersión
- Sobreflujo progresivo
- Cubetas
- Borrado de registros
- Otras formas de resolver colisiones

Otras formas de resolver colisiones I

- Dispersión doble: si ocurre una colisión se usa una segunda función de dispersión para obtener una nueva dirección que se suma a la anterior hasta que se encuentre una dirección vacía.
- Sobreflujo encadenado: cada registro contendrá una dirección donde se pueda encontrar al siguiente registro que obtuvo la misma dirección al aplicar la función de dispersión.

Otras formas de resolver colisiones II

- Encadenamiento con área de sobreflujo: se crea una segunda área donde se coloquen de forma encadenada los registros que han causado sobreflujo (suele ser más lento).
- Tablas de búsqueda: se podría tener un archivo de dispersión que no contuviera registros, sino sólo apuntadores a los mismos. Esto es equivalente a un índice que se accede por dispersión.

Esto es todo

Fin