Algoritmos y estructuras de datos

Problemas y algoritmos en gráficas

Francisco Javier Zaragoza Martínez

Universidad Autónoma Metropolitana Unidad Azcapotzalco
Departamento de Sistemas

29 de junio de 2020

Contenido

1. Aplicaciones de búsqueda en profundidad
   - Componentes conexas y biconexas
   - Unión y pertenencia
   - Ordenamiento topológico

2. Gráficas con costos

Componentes conexas

- Tanto la búsqueda en profundidad como en amplitud se pueden usar para encontrar las componentes conexas de una gráfica.
- 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).

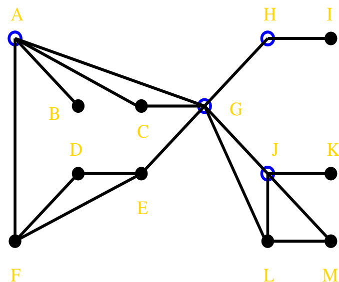
Implementación de componentes

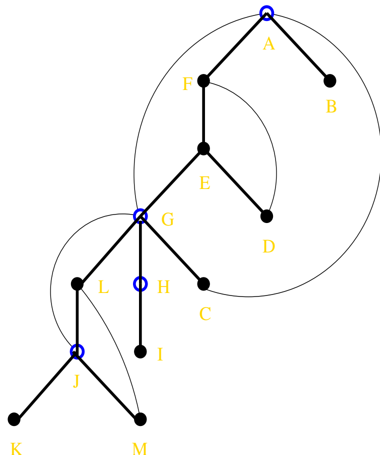
```
orden = 0;
for (k = 0; k < n; k++)
  visto[k] = 0;
for (k = 0; k < n; k++)
  if (visto[k] == 0) {
    visita(k);
    inver[visto[k]] = -inver[visto[k]];
  }

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 una gráfica no tiene cuellos de botella o puntos de fallo únicos.
- Un punto de articulación es un vértice de una gráfica que al borrarlo aumenta el número de componentes conexas de la gráfica.
- Una gráfica que no tiene puntos de articulación se llama biconexa.
- En una gráfica biconexa cada pareja de vértices distintos están conectados por al menos dos caminos disjuntos.

Ejemplo de componentes biconexas





Componentes biconexas de una gráfica.

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

1. Aplicaciones de búsqueda en profundidad
   - Componentes conexas y biconexas
   - Unión y pertenencia
   - Ordenamiento topológico
2. Gráficas 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.

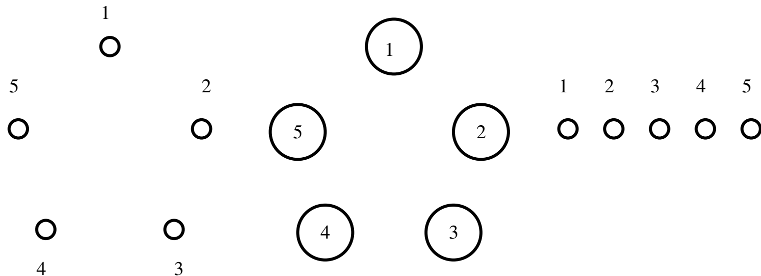
Gráficas dinámicas

- 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

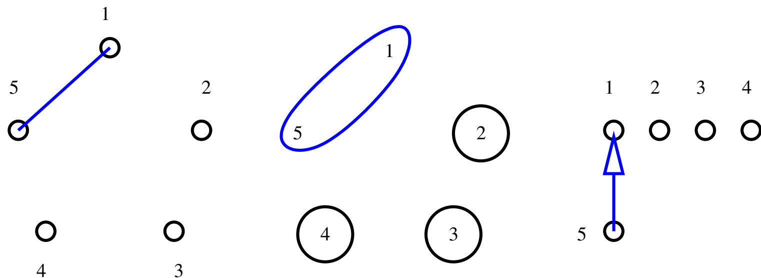
- La estructura que usaremos para representar a los subconjuntos es un bosque de árboles.
- Al principio cada elemento forma su propio subconjunto y queda representado por un árbol con un solo vértice.
- Cuando se unen dos subconjuntos, sus árboles se juntan en un solo árbol.
- Para verificar la pertenencia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

Ejemplo de unión y pertenencia 1



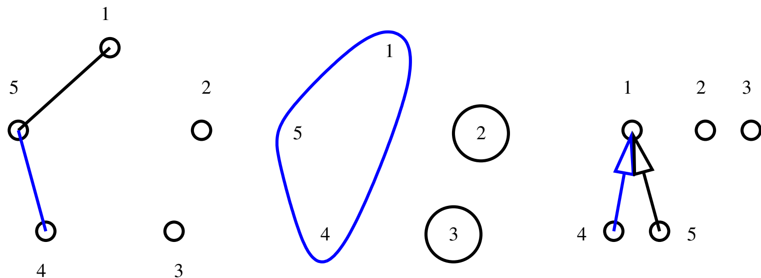
Gráfica sin aristas.

Ejemplo de unión y pertenencia 2



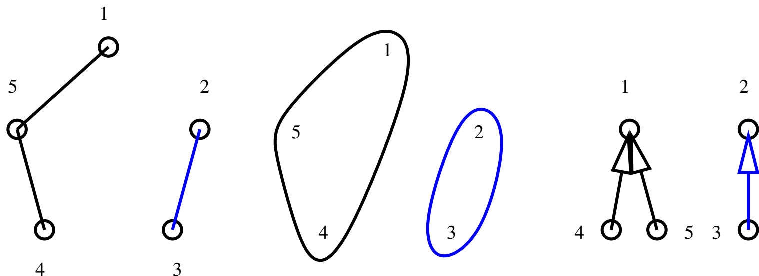
Se añade la arista $(1, 5)$.

Ejemplo de unión y pertenencia 3



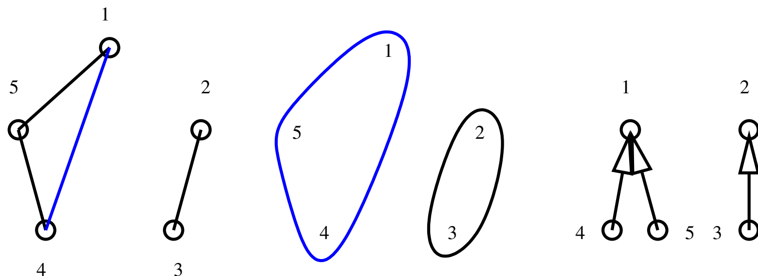
Se añade la arista $(4, 5)$.

Ejemplo de unión y pertenencia 4



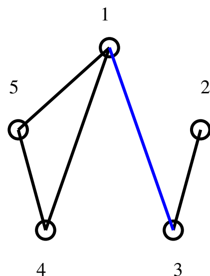
Se añade la arista $(3, 2)$.

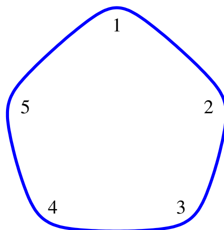
Ejemplo de unión y pertenencia 5

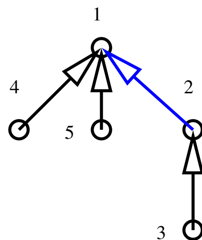


Se añade la arista $(4, 1)$.

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`.

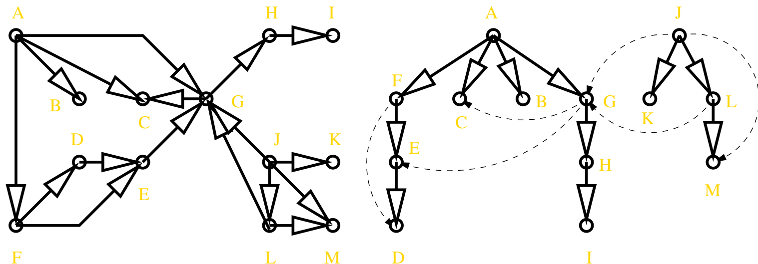
Contenido

1. Aplicaciones de búsqueda en profundidad
   - Componentes conexas y biconexas
   - Unión y pertenencia
   - Ordenamiento topológico
2. Gráficas con costos

Gráficas dirigidas acíclicas

- En muchas aplicaciones es importante que una gráfica dirigida 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 gráficas dirigidas acíclicas (en inglés directed acyclic graph o DAG).

Ejemplo de gráfica acíclica



Búsqueda en profundidad en una gráfica acíclica.

Ordenamiento topológico

- Los vértices de una gráfica acíclica 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?

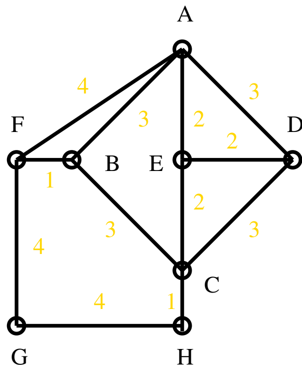
Contenido

1. Aplicaciones de búsqueda en profundidad
2. Gráficas con costos
   - Búsqueda por prioridad
   - Árboles abarcadores de costo mínimo
   - Caminos más cortos

Gráficas con costos

- En muchas aplicaciones se desea elegir algunas aristas de una gráfica 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 gráficas.

Ejemplo de gráfica con costos

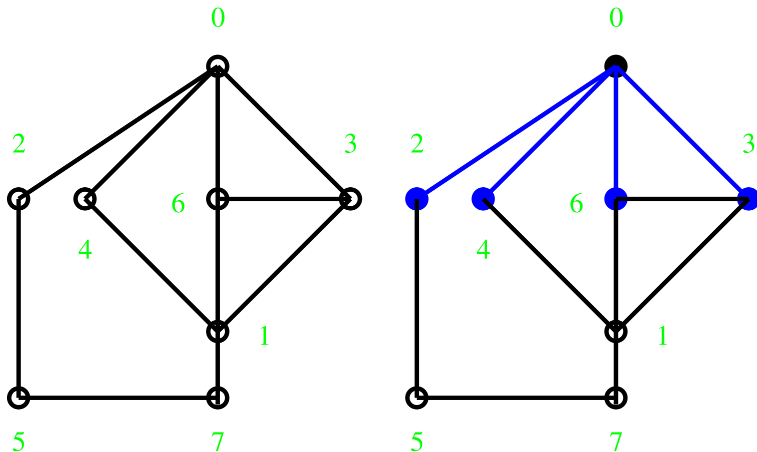


c	A	B	C	D	E	F	G	H
A	-	3	-	3	2	4	-	-
B	3	-	3	-	-	1	-	-
C	-	3	-	3	2	-	-	1
D	3	-	3	-	2	-	-	-
E	2	-	2	2	-	-	-	-
F	4	1	-	-	-	-	4	-
G	-	-	-	-	-	4	-	4
H	-	-	1	-	-	-	4	-

Búsqueda por prioridad

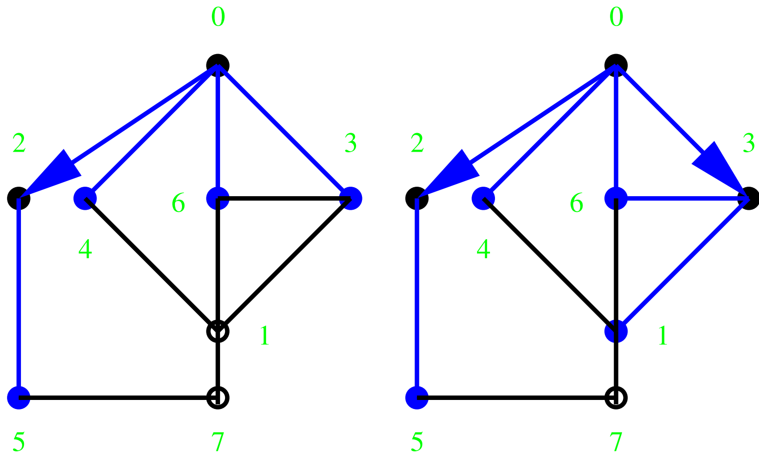
- Si en el método de búsqueda en profundidad no recursivo se reemplaza la pila por una cola de prioridad se obtiene un recorrido distinto.
- A éste se le llama búsqueda por prioridad.
- La prioridad se le asigna a los vértices.
- Recordemos que una cola de prioridad se puede implementar con un montículo.

Ejemplo de búsqueda por prioridad 1



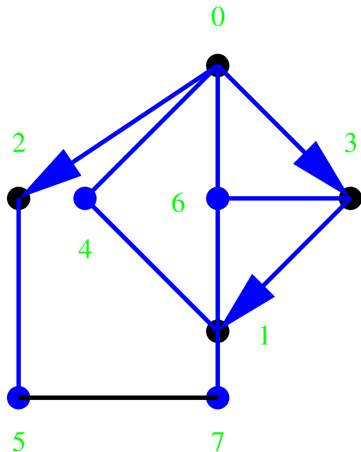
Cola de prioridad $[(0,-)] \rightarrow [(2,0),(3,0),(4,0),(6,0)]$.

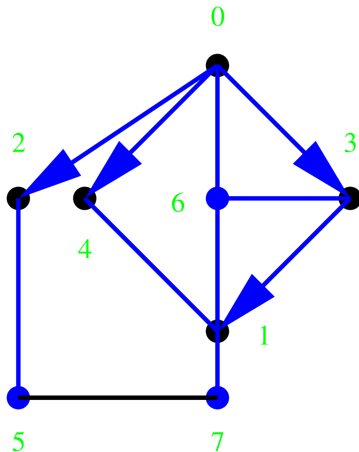
Ejemplo de búsqueda por prioridad 2



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

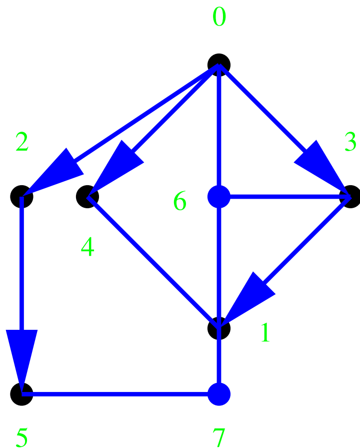
Ejemplo de búsqueda por prioridad 3

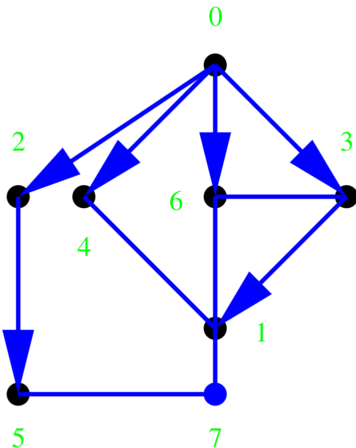




$[(1,3),(4,0),(5,2),(6,0)] \rightarrow [(4,0),(5,2),(6,0),(7,1)] \rightarrow$
$[(5,2),(6,0),(7,1)]$.

Ejemplo de búsqueda por prioridad 4





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

Contenido

1 Aplicaciones de búsqueda en profundidad

2 Gráficas con costos
- Búsqueda por prioridad
- Árboles abarcadores de costo mínimo
- Caminos más cortos

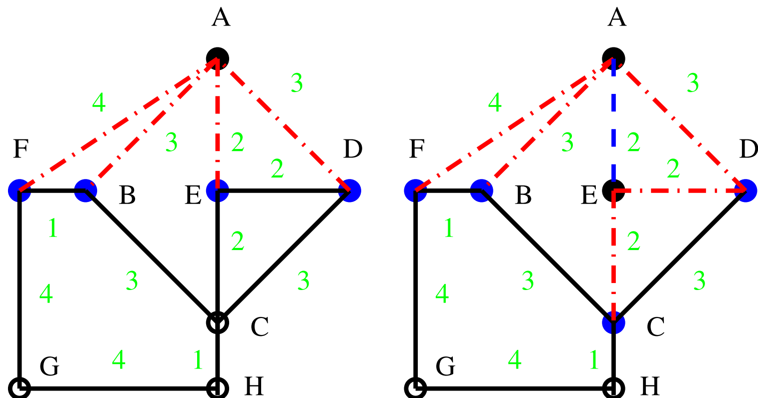
Árboles abarcadores mínimos

- Recordemos que un árbol abarcador es un árbol que usa todos los vértices de una gráfica.
- Si las aristas tienen costos le asignaremos a cada árbol abarcador como costo la suma de los costos de las aristas que lo forman.
- Una gráfica conexa 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

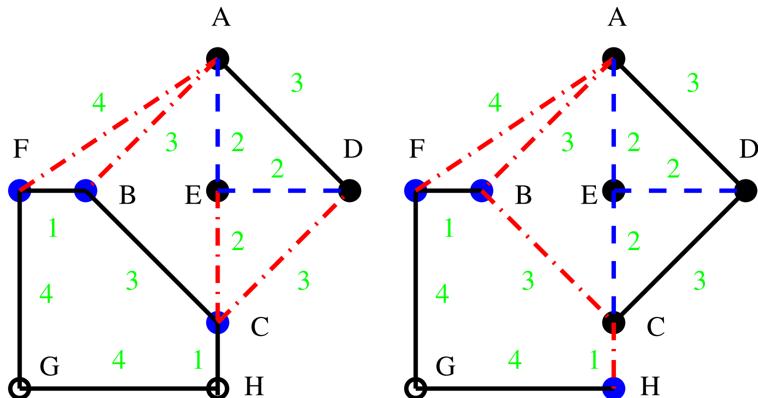
- El algoritmo de Prim construye un árbol abarcador de costo mínimo usando la búsqueda por prioridad.
- La prioridad de cada vértice viene dada por el costo mínimo de las aristas que lo unan a los vértices ya explorados.
- La prioridad de los vértices puede cambiar a lo largo de la ejecución del algoritmo.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

Ejemplo del algoritmo de Prim 1



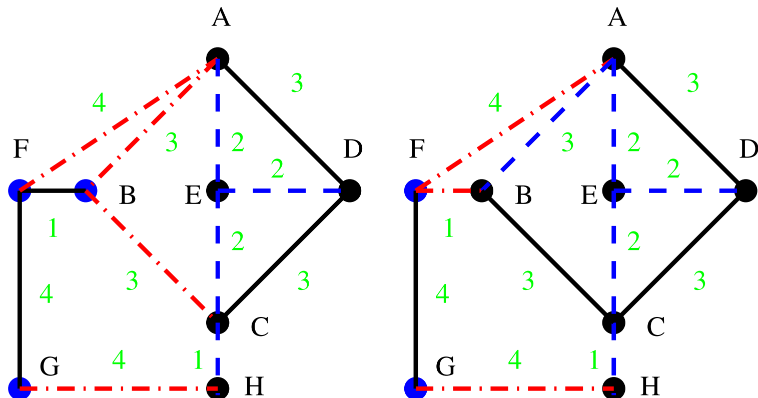
$[(A, -, 0)] \rightarrow [(E, A, 2), (B, A, 3), (D, A, 3), (F, A, 4)] \rightarrow$
$[(D, E, 2), (C, E, 2), (B, A, 3), (F, A, 4)]$.

Ejemplo del algoritmo de Prim 2



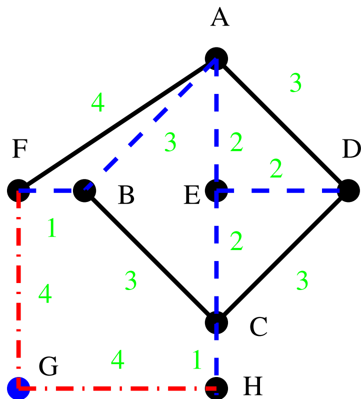
$[(D, E, 2), (C, E, 2), (B, A, 3), (F, A, 4)] \rightarrow$
$[(C, E, 2), (B, A, 3), (F, A, 4)] \rightarrow [(H, C, 1), (B, A, 3), (F, A, 4)].$

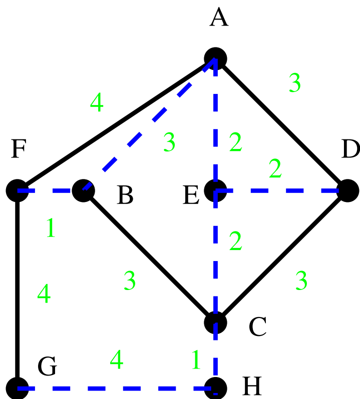
Ejemplo del algoritmo de Prim 3



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

Ejemplo del algoritmo de Prim 4



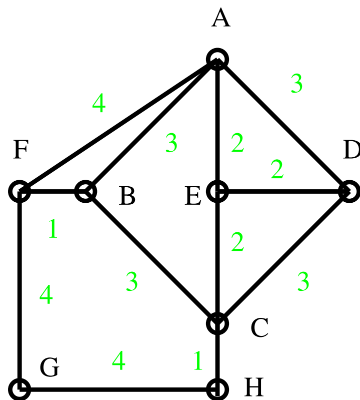


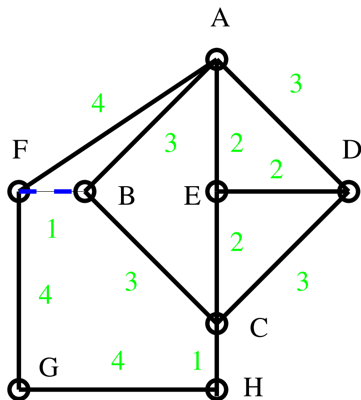
$[(F, B, 1), (G, H, 4)] \rightarrow [(G, H, 4)] \rightarrow []$.

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.

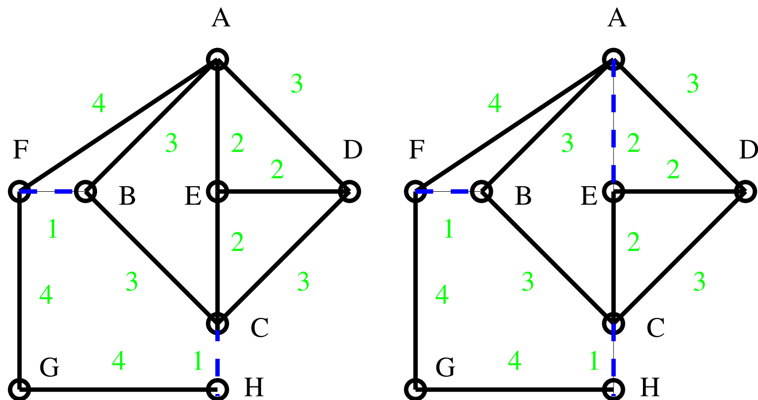
Ejemplo del algoritmo de Kruskal 1





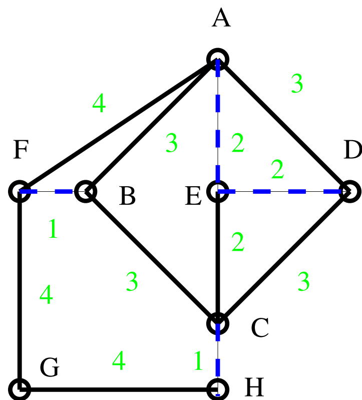
FB, CH, AE, ED, EC, AD, AB, BC, DC, FA, GH, FG.

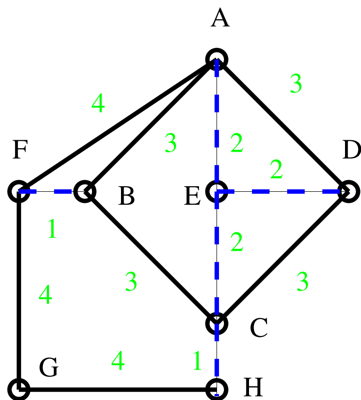
Ejemplo del algoritmo de Kruskal 2



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

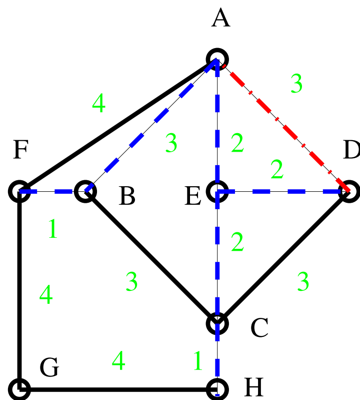
Ejemplo del algoritmo de Kruskal 3

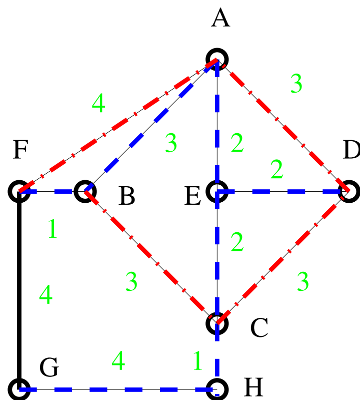




ED, EC, AD, AB, BC, DC, FA, GH, FG.

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

1. Aplicaciones de búsqueda en profundidad
2. Gráficas con costos
   - Búsqueda por prioridad
   - Árboles abarcadores de costo mínimo
   - Caminos más cortos

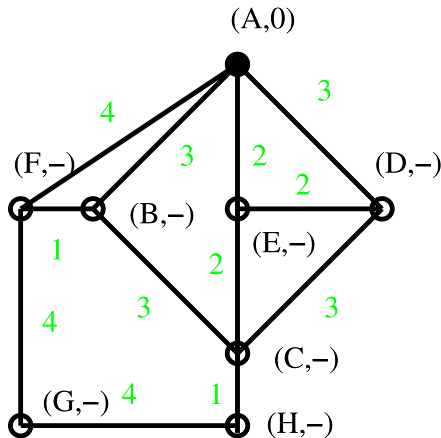
Caminos más cortos

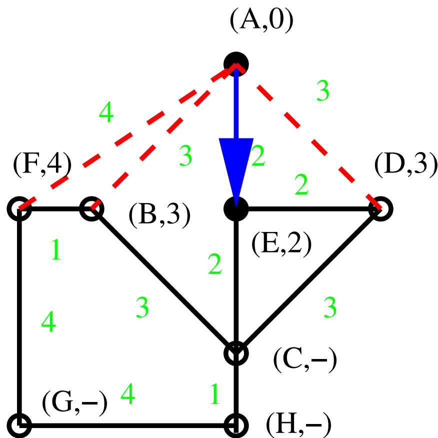
- Si una gráfica es conexa entonces todos sus vértices están conectados por caminos.
- Es probable que cada pareja de vértices esté conectada por más de un camino.
- Si a cada camino le asignamos un costo igual a la suma de los costos de las aristas que lo forman nos interesa encontrar un camino de costo 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.

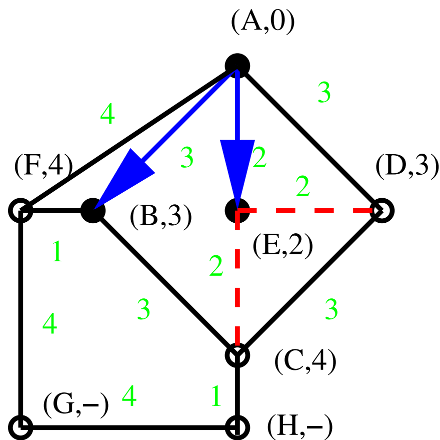
Ejemplo del algoritmo de Dijkstra 1

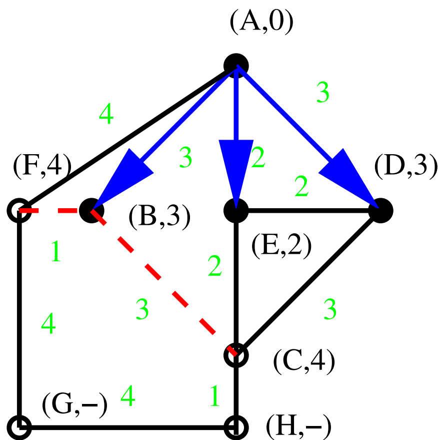




$[(A, -, 0)] \rightarrow [(E, A, 2), (B, A, 3), (D, A, 3), (F, A, 4)] \rightarrow$
$[(B, A, 3), (D, A, 3), (C, E, 4), (F, A, 4)]$.

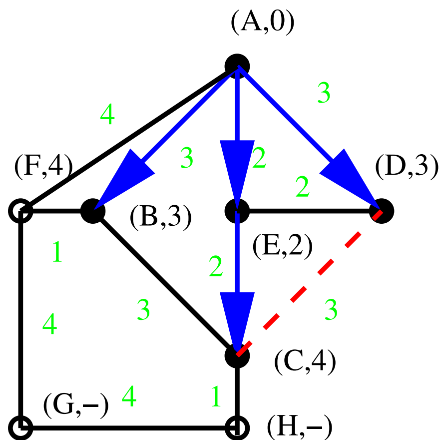
Ejemplo del algoritmo de Dijkstra 2

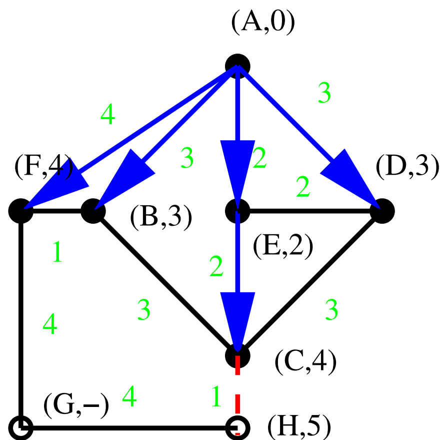




$[(B, A, 3), (D, A, 3), (C, E, 4), (F, A, 4)] \to$
$[(D, A, 3), (C, E, 4), (F, A, 4)] \to [(C, E, 4), (F, A, 4)].$

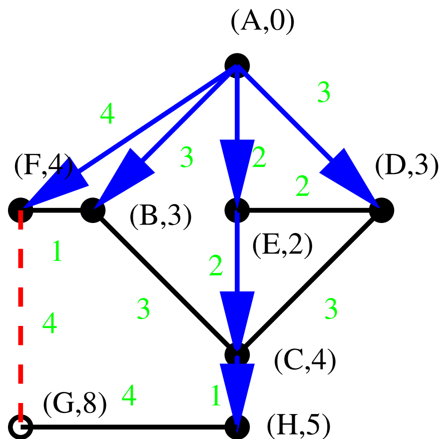
Ejemplo del algoritmo de Dijkstra 3

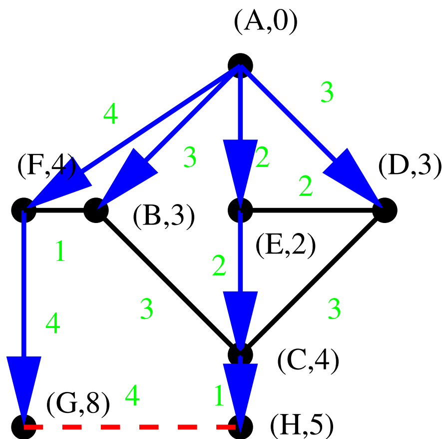




$[(C,E,4),(F,A,4)] \rightarrow [(F,A,4),(H,C,5)] \rightarrow [(H,C,5),(G,F,8)].$

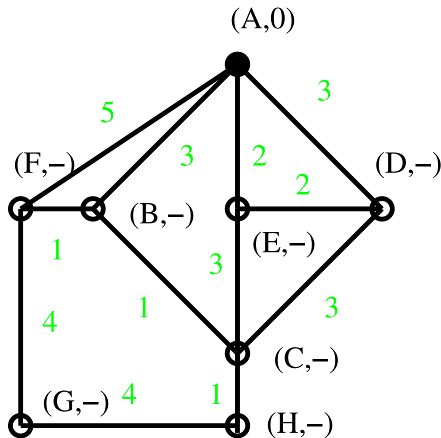
Ejemplo del algoritmo de Dijkstra 4

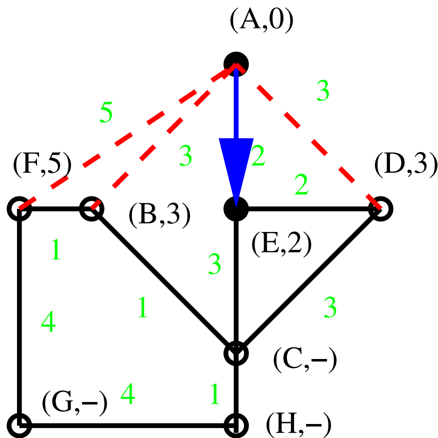




$[(H, C, 5), (G, F, 8)] \to [(G, F, 8)] \to []$.

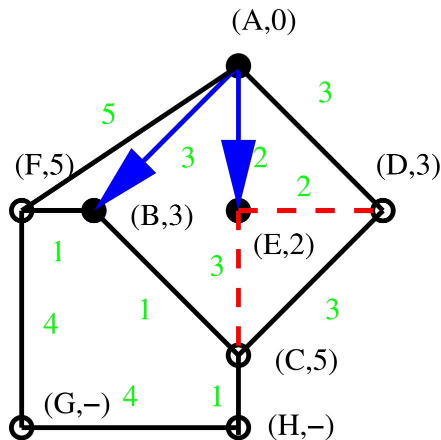
Otro ejemplo de Dijkstra 1

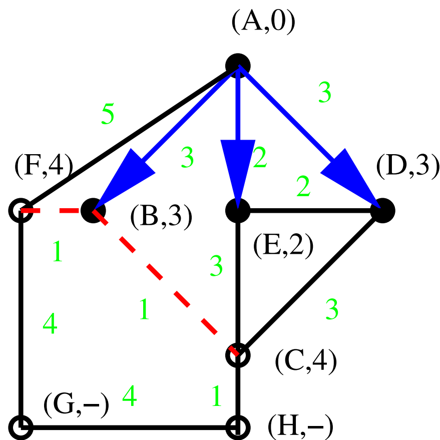




$[(A, -, 0)] \rightarrow [(E, A, 2), (B, A, 3), (D, A, 3), (F, A, 5)] \rightarrow$
$[(B, A, 3), (D, A, 3), (C, E, 5), (F, A, 5)]$.

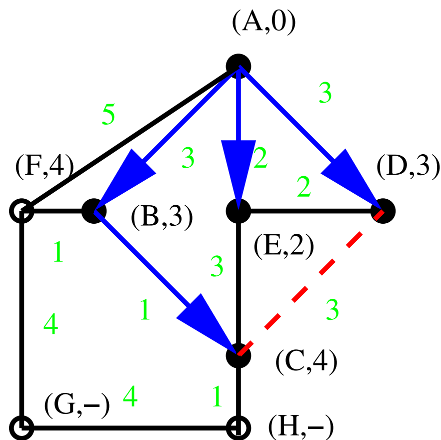
Otro ejemplo de Dijkstra 2

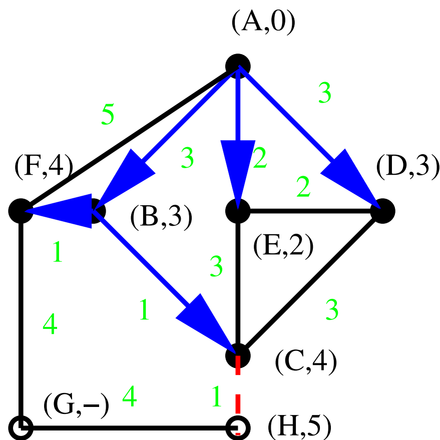




$[(B, A, 3), (D, A, 3), (C, E, 5), (F, A, 5)] \rightarrow$
$[(D, A, 3), (C, B, 4), (F, B, 4)] \rightarrow [(C, B, 4), (F, B, 4)].$

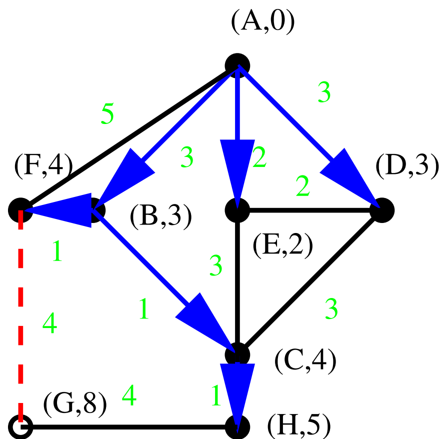
Otro ejemplo de Dijkstra 3

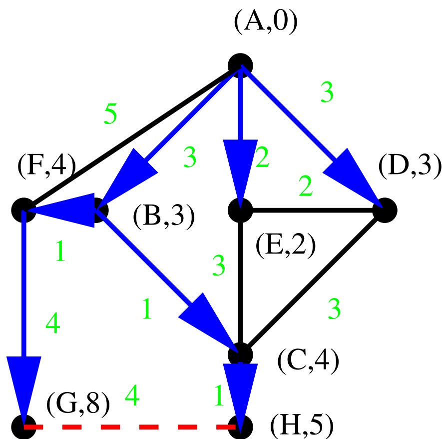




$[(C, B, 4), (F, B, 4)] \to [(F, B, 4), (H, C, 5)] \to [(H, C, 5), (G, F, 8)].$

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 una gráfica.
- 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