Algoritmos y estructuras de datos

Redes y búsqueda por prioridad

Francisco Javier Zaragoza Martínez

Universidad Autónoma Metropolitana Unidad Azcapotzalco
Departamento de Sistemas

Universidad Autónoma Metropolitana
Casa abierta al tiempo Azcapotzalco

Posgrado en Optimización

7 de junio de 2021

Citas

Frank Herbert

Debemos desarrollar una prioridad absoluta de los humanos por delante de las ganancias: cualquier humano por delante de cualquier ganancia. Así sobreviviremos.

Benjamin Creme

La primera prioridad será proveer de comida adecuada para todas las personas. La segunda será proveer de alojamiento adecuado para todas las personas. La tercera será proveer salud y educación adecuadas para todas las personas. Estos son los derechos humanos básicos que se necesitan en todas partes por todas las personas. Sin embargo, no hay país en el mundo en el cual todo esto pertenezca a los derechos universales.

Ben Carson

Si establecemos nuestra prioridad como “la eliminación de todo riesgo”, entonces pronto tendremos ambientes de aprendizaje estériles, estancados y poco estimulantes.

Definiciones

Las gráficas y digráficas que aparecen en la práctica vienen usualmente acompañadas de información numérica en sus vértices (importancia, prioridad), en sus aristas (longitud, duración) y en sus arcos (costo, capacidad). En general, lo llamaremos vector de costos.

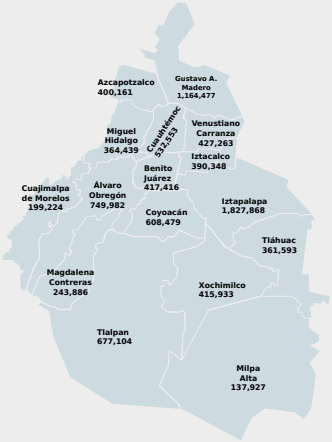
Una red consta de una gráfica $G = (V, E)$ y al menos un vector de costos en sus vértices o en sus aristas. Una red dirigida consta de una digráfica $D = (V, A)$ y al menos un vector de costos en sus vértices o en sus arcos.

Cada vértice y arista de G (o arco de D) puede venir acompañado de cierta información adicional. Un vértice puede tener nombre, ubicación, etc., mientras que una arista o arco puede tener nombre, etc. Para simplificar, los vértices estarán numerados del 0 al $n - 1$ y el resto de la información se almacenará por separado. Por ejemplo, los nombres de los vértices se podrían almacenar usando un arreglo de apuntadores a cadenas (`char *s[n]`).

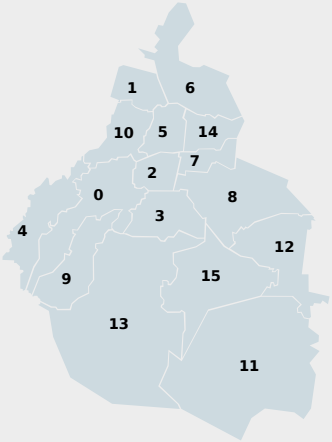
Red con costos en los vértices

Ejemplo

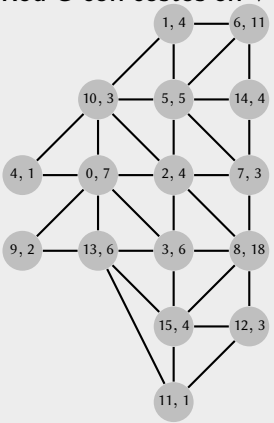
Población (INEGI 2015)



Mapa con números



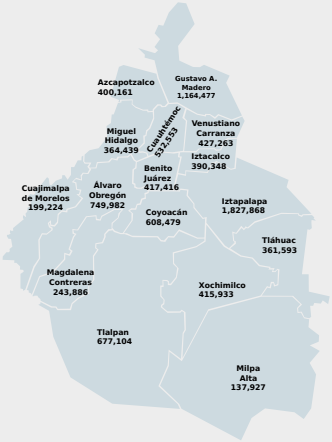
Red G con costos en V



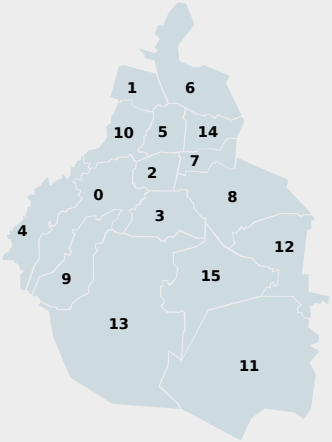
Red con costos en las aristas

Ejemplo

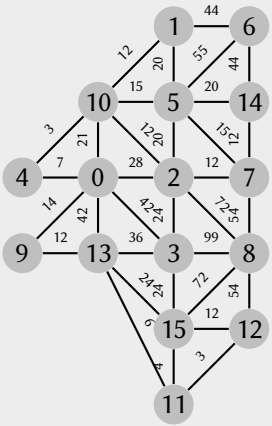
Población (INEGI 2015)



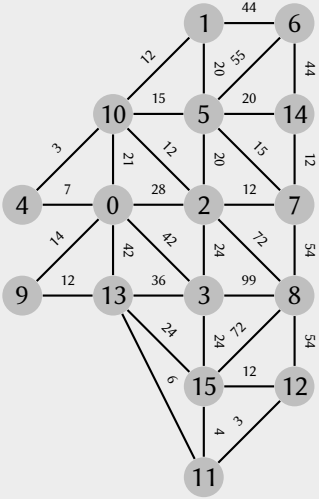
Mapa con números



Red G con costos en E



Representación con matriz de costos

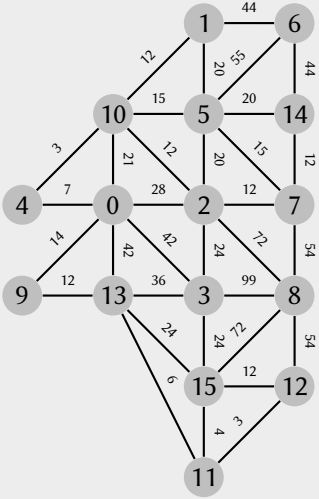


La forma más sencilla de representar una red es a través de su matriz de costos C, la cual es una matriz cuadrada simétrica de $n \times n$ en la cual C_{uv} es el costo de la arista uv cuando exista y se elegirá adecuadamente (como cero, negativo, muy grande, etc.) en caso contrario.

La matriz de costos ocupa espacio proporcional a n^2. Esto la hace ideal para representar redes completas o densas. Por otro lado, la hace completamente inadecuada para representar redes dispersas.

Redes

Representación con matriz de costos



C	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	0	28	42	7	0	0	0	0	14	21	0	0	42	0	0
1	0	0	0	0	0	20	44	0	0	0	12	0	0	0	0	0
2	28	0	0	24	0	20	0	12	72	0	12	0	0	0	0	0
3	42	0	24	0	0	0	0	0	99	0	0	0	0	36	0	24
4	7	0	0	0	0	0	0	0	0	0	3	0	0	0	0	0
5	0	20	20	0	0	0	55	15	0	0	15	0	0	0	20	0
6	0	44	0	0	0	55	0	0	0	0	0	0	0	0	44	0
7	0	0	12	0	0	15	0	0	54	0	0	0	0	0	12	0
8	0	0	72	99	0	0	0	54	0	0	0	0	54	0	0	72
9	14	0	0	0	0	0	0	0	0	0	0	0	0	12	0	0
10	21	12	12	0	3	15	0	0	0	0	0	0	0	0	0	0
11	0	0	0	0	0	0	0	0	0	0	0	0	3	6	0	4
12	0	0	0	0	0	0	0	0	54	0	0	3	0	0	0	12
13	42	0	0	36	0	0	0	0	0	12	0	6	0	0	0	24
14	0	0	0	0	0	20	44	12	0	0	0	0	0	0	0	0
15	0	0	0	24	0	0	0	0	72	0	0	4	12	24	0	0

Tipos asociados a la matriz de costos

Definiremos un tipo estructurado `red` para representar una gráfica y su matriz de costos. Este tipo consiste de la cantidad `n` de vértices y de dos matrices `a` y `c` de $n \times n$.

```
typedef struct {
  int    n; // cantidad de vertices
  int **a; // matriz de adyacencia
  int  **c; // matriz de costos
} red;
```

En realidad `a` y `c` serán apuntadores a vectores de `n` apuntadores a vectores de tamaño `n` pero posteriormente los usaremos simplemente como matrices. De ser necesario, esta estructura se podría aumentar con un vector para los nombres de los vértices, etc.

Redes

Creación de una red vacía

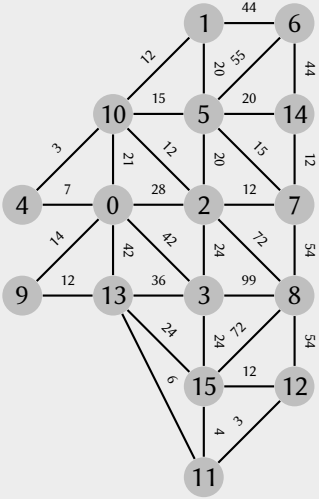
```
red creaRed(int n) {
  red r;
  r.n = n;
  r.a = (int **) malloc(r.n*sizeof(int *));
  r.c = (int **) malloc(r.n*sizeof(int *));
  for (int u = 0; u < r.n; u++) {
    r.a[u] = (int *) calloc(r.n, sizeof(int));
    r.c[u] = (int *) calloc(r.n, sizeof(int));
  }
  return r;
}
```

Destrucción de una red

Primero liberamos los renglones de a y c y luego los apuntadores a los renglones.

```
void destruyeRed(red *r) {
  for (int u = 0; u < r->n; u++) {
    free(r->a[u]);
    free(r->c[u]);
  }
  free(r->a);
  free(r->c);
  r->a = NULL;
  r->c = NULL;
  r->n = 0;
}
```

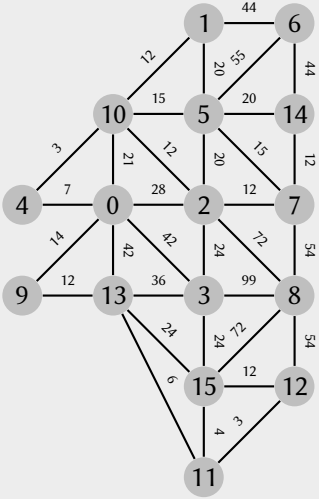
Representación con listas de adyacencia



Otra forma de representar una red es a través de sus listas de adyacencia A, las cuales son representaciones de los conjuntos A_u de los vecinos v de cada vértice u junto con el costo C_{uv} de la arista correspondiente. Observe que una arista que une a u y v queda representada dos veces: como $v \in A_u$ y como $u \in A_v$.

Las listas de adyacencia ocupan espacio proporcional a $n + m$. Esto las hace ideales para representar redes dispersas (y ligeramente inferiores para redes densas).

Representación con listas de adyacencia



A_0 = { (2,28), (3,42), (4,7), (9,14), (10,21), (13,42) }

A_1 = { (5,20), (6,44), (10,12) }

A_2 = { (0,28), (3,24), (5,20), (7,12), (8,72), (10,12) }

A_3 = { (0,42), (2,24), (8,99), (13,36), (15,24) }

A_4 = { (0,7), (10,3) }

A_5 = { (1,20), (2,20), (6,55), (7,15), (10,15), (14,20) }

A_6 = { (1,44), (5,55), (14,44) }

A_7 = { (2,12), (5,15), (8,54), (14,12) }

A_8 = { (2,72), (3,99), (7,54), (12,54), (15,72) }

A_9 = { (0,14), (13,12) }

A_{10} = { (0,21), (1,12), (2,12), (4,10), (5,15) }

A_{11} = { (12,3), (13,6), (15,4) }

A_{12} = { (8,54), (11,3), (15,12) }

A_{13} = { (0,42), (3,36), (9,12), (11,6), (15,24) }

A_{14} = { (5,20), (6,44), (7,12) }

A_{15} = { (3,24), (8,72), (11,4), (12,12), (13,24) }

Tipos asociados a las listas de adyacencia

Definiremos un tipo estructurado `redLA` para representar las listas de adyacencia de una red. Este tipo consiste de la cantidad `n` de vértices y de un vector `a` de `n` listas.

```
typedef struct {
  int    n; // cantidad de vertices
  lista *a; // listas de adyacencia
} redLA;
```

Los nodos de las listas deberán almacenar los costos de las aristas.

```
typedef struct nodoRed {
  int v;           // vecino v
  int c;           // costo c[u][v]
  struct nodoRed *sig; // siguiente
} nodoRed;
```

Esta estructura se podría aumentar con un vector para los nombres de los vértices, etc.

Búsqueda por prioridad

Un algoritmo para recorrer una red con un vector de prioridades p en sus vértices, llamado búsqueda por prioridad (PFS por las siglas en inglés de *priority first search*), se obtiene de sustituir en la búsqueda en amplitud la cola por una cola de prioridad.

1. Se marcan todos los vértices como no vistos.
2. Mientras haya algún vértice u no visto:
    1. Se forma u (con prioridad p_u).
    2. Mientras la cola de prioridad no esté vacía:
        1. Se desforma u.
        2. Se marca a u como ya visto.
        3. Se forma (con prioridad p_v) a cada vecino v no visto de u.

Con matriz de adyacencia, la cola de prioridad se puede sustituir por una búsqueda lineal sin pérdida de eficiencia. Con listas de adyacencia se requerirá un montículo.

Usaremos un arreglo en el que `visto[u]` será el `orden` de visita de `u` (`0` si no se ha visto).

```
void busquedaPrioridad(red r, int p[]) {
  int orden = 0;
  int *visto = (int *) calloc(r.n, sizeof(int));
  for (int u = 0; u < r.n; u++)
    if (visto[u] == 0)
      prioridad(r, u, visto, &orden, p);
  free(visto);
}
```

En lo que sigue, las funciones que terminan en `CPE` son de una cola de prioridad en un arreglo estático, por ejemplo, un montículo mínimo.

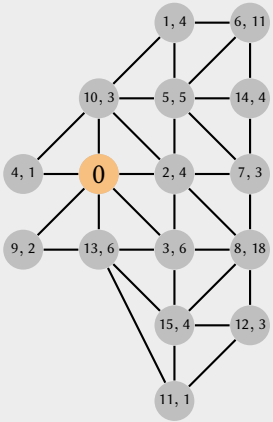
Búsqueda por prioridad

Implementación con matrices de adyacencia

```
void prioridad(red r, int u, int visto[], int *orden, int p[]) {
  int q;
  CPE s = creaCPE(r.n);
  formaCPE(&s, u, p[u]);
  while (!vaciaCPE(&s)) {
    desformaCPE(&s, &u, &q);
    visto[u] = ++(*orden);
    for (int v = 0; v < r.n; v++)
      if (r.a[u][v] == 1 && visto[v] == 0) {
        formaCPE(&s, v, p[v]);
        visto[v] = -1;
      }
  }
  destruyeCPE(&s);
}
```

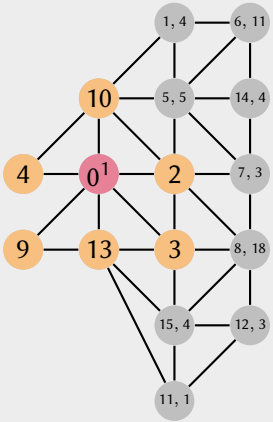
Búsqueda por prioridad

Ejemplo de red



Búsqueda por prioridad

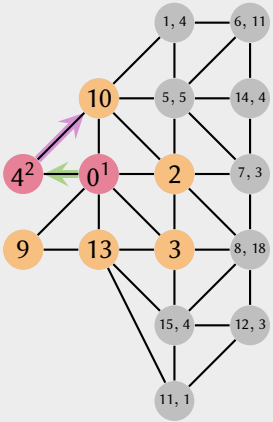
Ejemplo de red



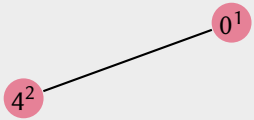
13,6
3,6
2,4
10,3
9,2
4,1
CPE s

Búsqueda por prioridad

Ejemplo de red

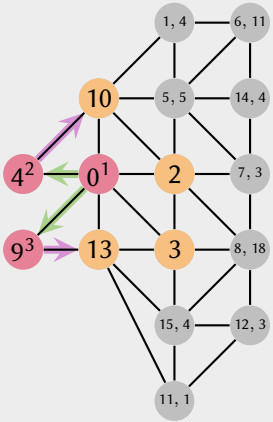


13,6
3,6
2,4
10,3
9,2
CPE s

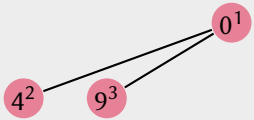


Búsqueda por prioridad

Ejemplo de red

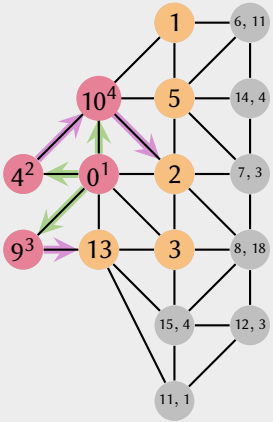


13,6
3,6
2,4
10,3
CPE s

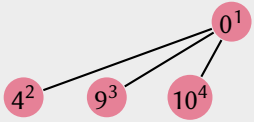


Búsqueda por prioridad

Ejemplo de red

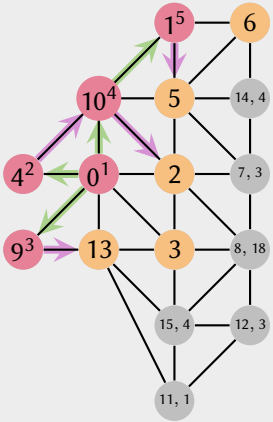


13,6
3,6
5,5
2,4
1,4
CPE s

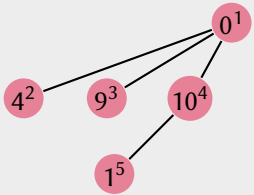


Búsqueda por prioridad

Ejemplo de red

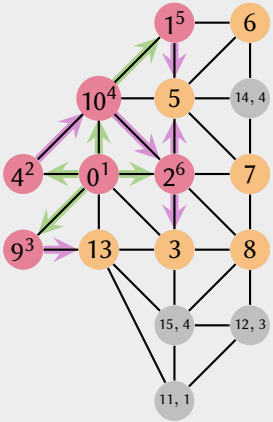


6,11
13,6
3,6
5,5
2,4
CPE s

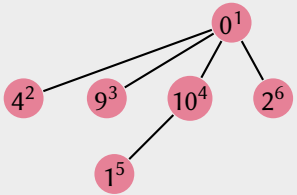


Búsqueda por prioridad

Ejemplo de red

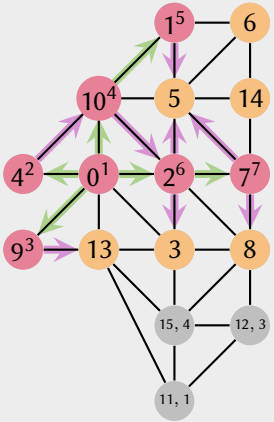


8,18
6,11
13,6
3,6
5,5
7,3
CPE s

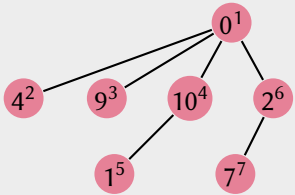


Búsqueda por prioridad

Ejemplo de red

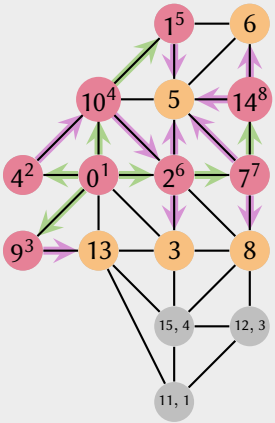


8,18
6,11
13,6
3,6
5,5
14,4
CPE s

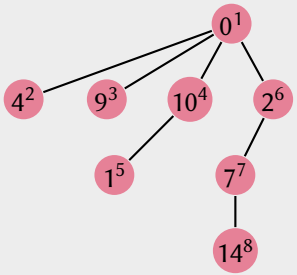


Búsqueda por prioridad

Ejemplo de red

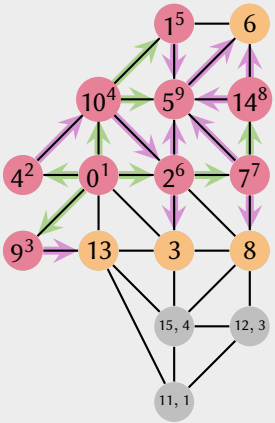


8,18
6,11
13,6
3,6
5,5
CPE s

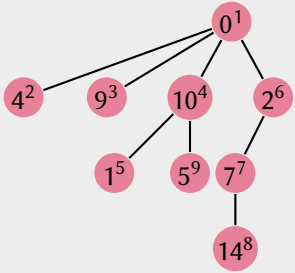


Búsqueda por prioridad

Ejemplo de red

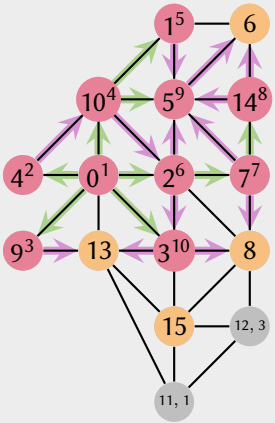


8,18
6,11
13,6
3,6
CPE s

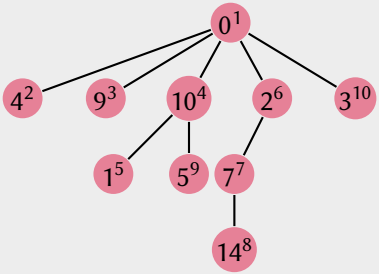


Búsqueda por prioridad

Ejemplo de red

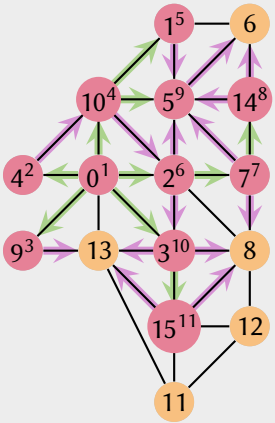


8,18
6,11
13,6
15,4
CPE s

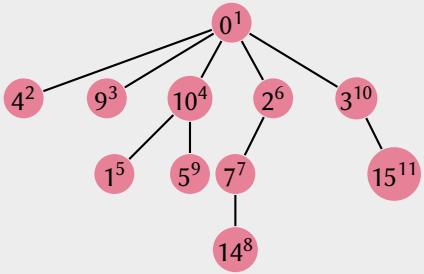


Búsqueda por prioridad

Ejemplo de red

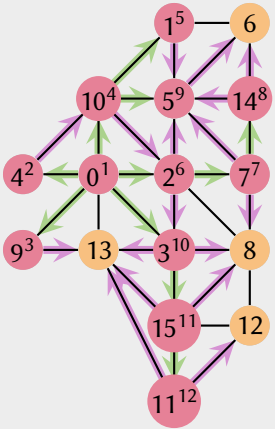


8,18
6,11
13,6
12,3
11,1
CPE s

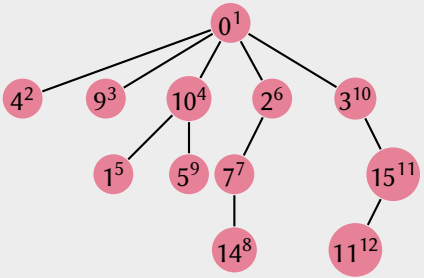


Búsqueda por prioridad

Ejemplo de red

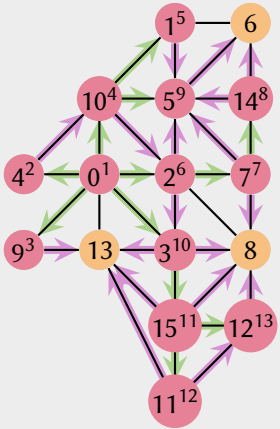


8,18
6,11
13,6
12,3
CPE s

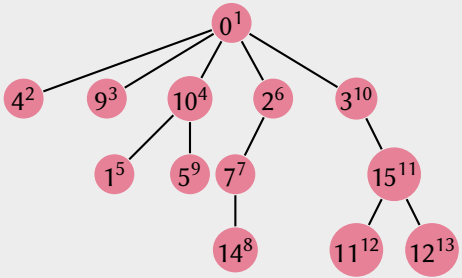


Búsqueda por prioridad

Ejemplo de red

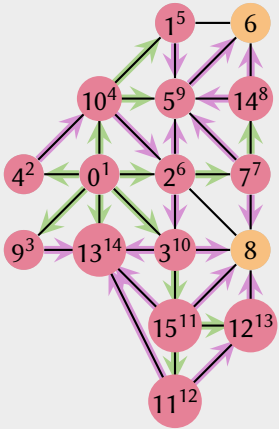


8,18
6,11
13,6
CPE s

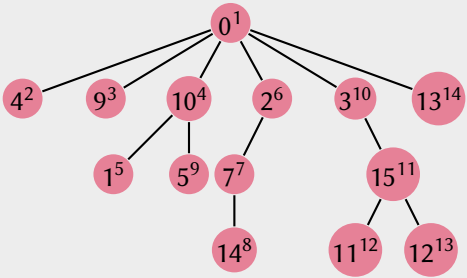


Búsqueda por prioridad

Ejemplo de red

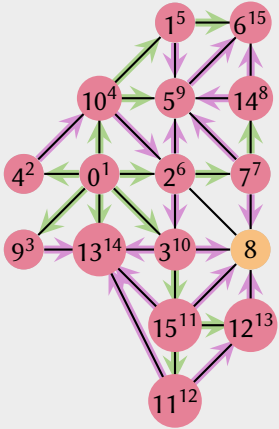


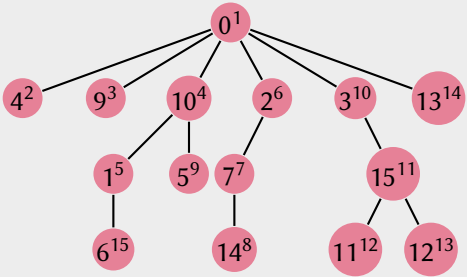
8,18
6,11
CPE s



Búsqueda por prioridad

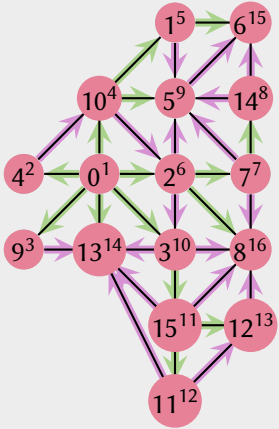
Ejemplo de red

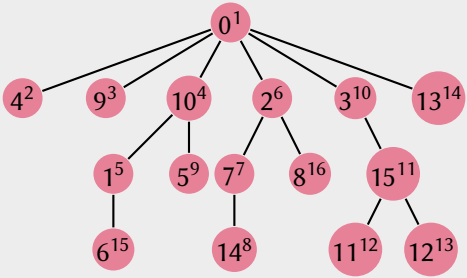




Búsqueda por prioridad

Ejemplo de red





Búsqueda por prioridad

Redes completas y búsqueda lineal

En muchas aplicaciones de redes se puede suponer que la red es completa. En este caso, el ciclo de busqueda es innecesario pues solo se llamaría a prioridad una vez. Además, si la red está representada por su matriz de adyacencia, entonces podemos hacer búsqueda lineal en p sin pérdida de eficiencia (la búsqueda de vecinos ya toma tiempo lineal).

```
int minPrioridad(red r, int p[], int visto[]) {
  int u, q = INT_MAX;     // prioridad infinita
  for (int v = 0; v < r.n; v++)
    if (visto[v] == 0 && p[v] < q) {
      u = v;              // vertice no visto
      q = p[v];           // con menor prioridad
    }
  return u;
}
```

Búsqueda por prioridad

Implementación con matrices de adyacencia (red completa, búsqueda lineal)

```
void prioridad(red r, int u, int p[]) {
  int *visto = (int *) calloc(r.n, sizeof(int));
  for (int orden = 1; orden <= r.n; orden++) {
    visto[u] = orden;
    u = minPrioridad(r, p, visto);
  }
  free(visto);
}
```

Búsqueda por prioridad

Observaciones

1. El orden de visita es de acuerdo a la prioridad de los vértices.
2. Cada vértice se visita una vez y cada arista se visita dos veces.
3. El tiempo de ejecución depende de la representación (matriz con búsqueda lineal n^2, listas de adyacencia con montículo $(n + m) \log_2 n$).
4. Las aristas que visitan por primera vez vértices no vistos forman un bosque de búsqueda por prioridad.

Este algoritmo se usa sin cambios en digráficas, aunque cambia el comportamiento.

1. Cada vértice y cada arco se visitan una vez.
2. Los arcos que no pertenecen al bosque de búsqueda por prioridad pueden apuntar hacia arriba, hacia abajo o ser transversales.