Algoritmos y estructuras de datos

Árboles abarcadores de costo mínimo

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

Fernando de Rojas

Contentémonos con lo razonable, no sea que por querer más lo perdamos todo, que quien mucho abarca, poco aprieta.

Paul Graham

Los arquitectos saben que algunos tipos de problemas de diseño son más personales que otros. Uno de los más limpios y abstractos problemas de diseño es diseñar puentes. Allí tu trabajo es mayormente un asunto de abarcar una distancia con el mínimo de material.

David Hartley Coleridge

Dulce sería el cambio ¡aún si fuera cambio de tortura! Pero crecer como roca inamovible, como prisión, era tras era. ¡Hasta que los soles que giran superen a las arenas de una playa desgastada por la marea! Sin esperanza, o esa triste imitación de esperanza que engaña con aburrida desesperación. ¡Abarcando el infinito! ¡Tormento sin medida!

Redes y redes dirigidas

Definiciones

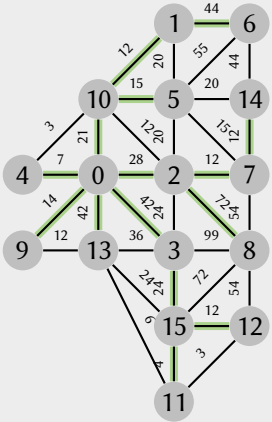
Las gráficas y digráficas que aparecen en la práctica vienen usualmente acompañadas de información numérica en sus vértices (importancia, prioridad), en sus aristas (longitud, duración) y en sus arcos (costo, capacidad). En general, lo llamaremos vector de costos.

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.

Los problemas de diseño de redes se tratan de encontrar una subred H de G (o de D) que sea de costo mínimo (o máximo) y que al mismo tiempo satisfaga alguna propiedad deseada. El costo de H está dado por la suma de los costos de sus vértices, aristas o arcos.

Gráficas

Árboles abarcadores



Un bosque de G es una subgráfica de G que no tiene ciclos, un árbol de G es un bosque de G conexo y un árbol abarcador de G es un árbol con todos los vértices de G.

Si G forma parte de una red con costos en las aristas, entonces el costo de cualquier bosque de G es la suma de los costos de sus aristas. Si todos los costos son positivos, entonces un bosque de costo mínimo no tiene aristas.

El problema se vuelve más interesante cuando G es conexa y se busca un árbol abarcador de costo mínimo.

Árboles abarcadores de costo mínimo

Algoritmo de Prim

El algoritmo de Prim es un tipo de búsqueda por prioridad variable. Al principio, el vértice inicial u tendrá prioridad $p_u = 0$ y los otros vértices $v \neq u$ tendrán prioridad $p_v = +\infty$. Cada que visitemos un vértice u actualizaremos la prioridad de sus vecinos v con el mínimo de p_v y c_{uv}. Cada vértice elige la arista que lo une al último que actualizó su prioridad. Un árbol abarcador de costo mínimo se encuentra en el arreglo `arista`.

```
void actPrim(red r, int u, int p[], int visto[], int arista[]) {
  for (int v = 0; v < r.n; v++)
    if (r.a[u][v] == 1 && visto[v] == 0 && r.c[u][v] < p[v]) {
      p[v] = r.c[u][v];
      arista[v] = u; // ultimo vertice en actualizar a v
    }
}
```

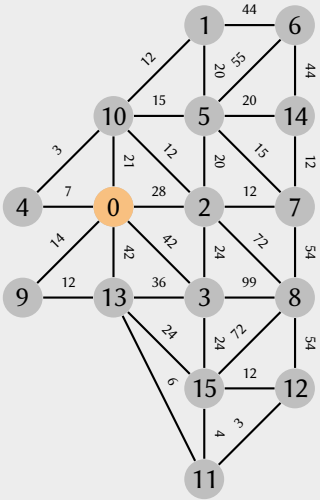
Árboles abarcadores de costo mínimo

Implementación del algoritmo de Prim con matriz de adyacencia

```
void prim(red r, int u, int arista[]) {
  int *p = (int *) malloc(r.n*sizeof(int));
  int *visto = (int *) calloc(r.n, sizeof(int));
  arista[u] = u;                    // u no genera una arista
  for (int v = 0; v < r.n; v++)
    p[v] = (v == u) ? 0 : INT_MAX;   // prioridad inicial
  for (int orden = 1; orden <= r.n; orden++) {
    visto[u] = orden;
    actPrim(r, u, p, visto, arista);
    u = minPrioridad(r, p, visto);
  }
  free(visto);
  free(p);
}
```

Árboles abarcadores de costo mínimo

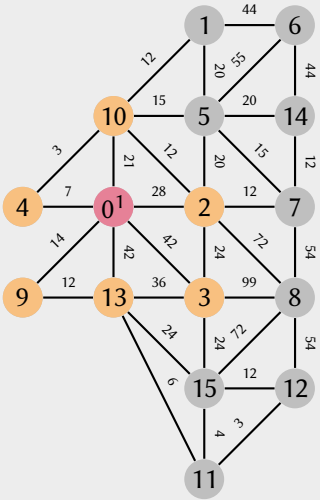
Ejemplo del algoritmo de Prim



u	v[u]	p[u]	a[u]
0	0	0	0
1	0	$+\infty$	×
2	0	$+\infty$	×
3	0	$+\infty$	×
4	0	$+\infty$	×
5	0	$+\infty$	×
6	0	$+\infty$	×
7	0	$+\infty$	×
8	0	$+\infty$	×
9	0	$+\infty$	×
10	0	$+\infty$	×
11	0	$+\infty$	×
12	0	$+\infty$	×
13	0	$+\infty$	×
14	0	$+\infty$	×
15	0	$+\infty$	×

Árboles abarcadores de costo mínimo

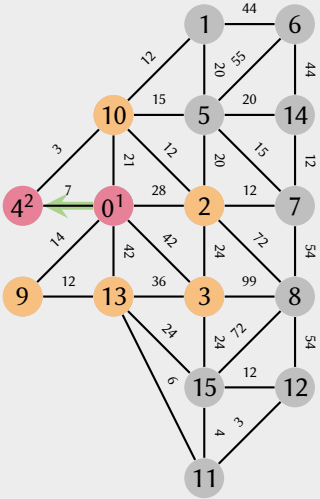
Ejemplo del algoritmo de Prim



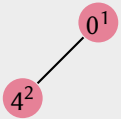
u	v[u]	p[u]	a[u]
0	1	0	0
1	0	$+\infty$	$\times$
2	0	28	0
3	0	42	0
4	0	7	0
5	0	$+\infty$	$\times$
6	0	$+\infty$	$\times$
7	0	$+\infty$	$\times$
8	0	$+\infty$	$\times$
9	0	14	0
10	0	21	0
11	0	$+\infty$	$\times$
12	0	$+\infty$	$\times$
13	0	42	0
14	0	$+\infty$	$\times$
15	0	$+\infty$	$\times$

Árboles abarcadores de costo mínimo

Ejemplo del algoritmo de Prim

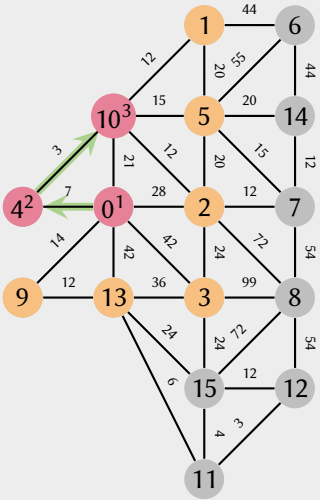


u	v[u]	p[u]	a[u]
0	1	0	0
1	0	$+\infty$	×
2	0	28	0
3	0	42	0
4	2	7	0
5	0	$+\infty$	×
6	0	$+\infty$	×
7	0	$+\infty$	×
8	0	$+\infty$	×
9	0	14	0
10	0	3	4
11	0	$+\infty$	×
12	0	$+\infty$	×
13	0	42	0
14	0	$+\infty$	×
15	0	$+\infty$	×

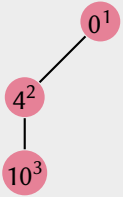


Árboles abarcadores de costo mínimo

Ejemplo del algoritmo de Prim

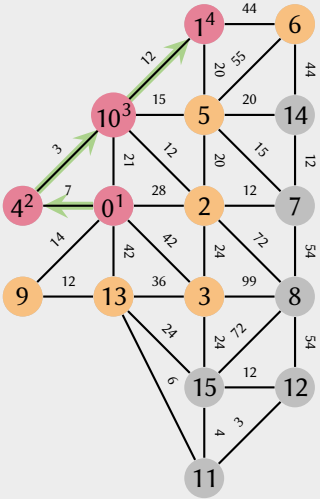


u	v[u]	p[u]	a[u]
0	1	0	0
1	0	12	10
2	0	12	10
3	0	42	0
4	2	7	0
5	0	15	10
6	0	$+\infty$	$\times$
7	0	$+\infty$	$\times$
8	0	$+\infty$	$\times$
9	0	14	0
10	3	3	4
11	0	$+\infty$	$\times$
12	0	$+\infty$	$\times$
13	0	42	0
14	0	$+\infty$	$\times$
15	0	$+\infty$	$\times$

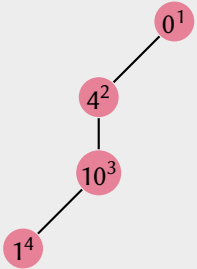


Árboles abarcadores de costo mínimo

Ejemplo del algoritmo de Prim

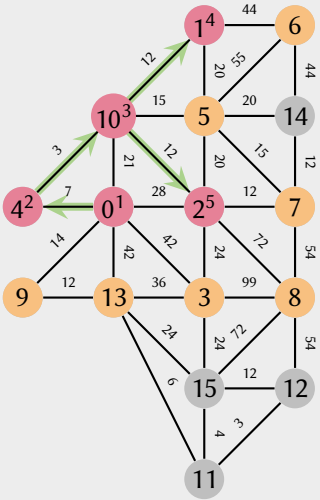


u	v[u]	p[u]	a[u]
0	1	0	0
1	4	12	10
2	0	12	10
3	0	42	0
4	2	7	0
5	0	15	10
6	0	44	1
7	0	$+\infty$	$\times$
8	0	$+\infty$	$\times$
9	0	14	0
10	3	3	4
11	0	$+\infty$	$\times$
12	0	$+\infty$	$\times$
13	0	42	0
14	0	$+\infty$	$\times$
15	0	$+\infty$	$\times$

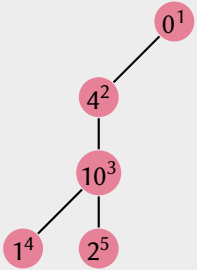


Árboles abarcadores de costo mínimo

Ejemplo del algoritmo de Prim

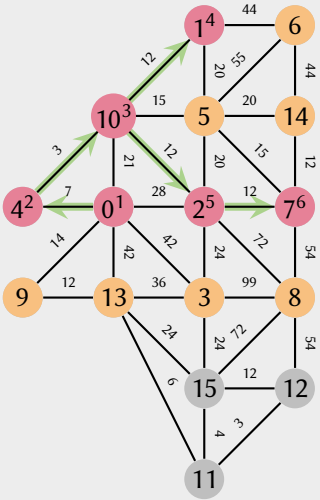


u	v[u]	p[u]	a[u]
0	1	0	0
1	4	12	10
2	5	12	10
3	0	24	2
4	2	7	0
5	0	15	10
6	0	44	1
7	0	12	2
8	0	72	2
9	0	14	0
10	3	3	4
11	0	$+\infty$	$\times$
12	0	$+\infty$	$\times$
13	0	42	0
14	0	$+\infty$	$\times$
15	0	$+\infty$	$\times$

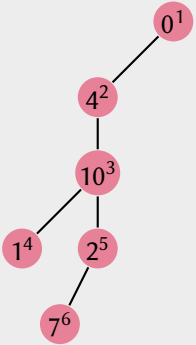


Árboles abarcadores de costo mínimo

Ejemplo del algoritmo de Prim

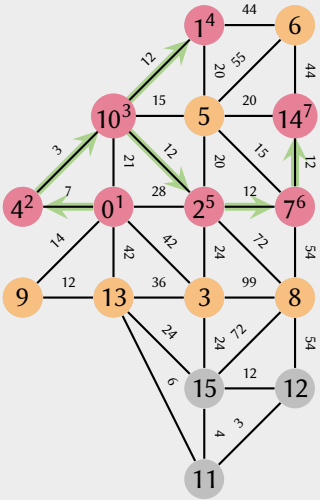


u	v[u]	p[u]	a[u]
0	1	0	0
1	4	12	10
2	5	12	10
3	0	24	2
4	2	7	0
5	0	15	10
6	0	44	1
7	6	12	2
8	0	54	7
9	0	14	0
10	3	3	4
11	0	$+\infty$	×
12	0	$+\infty$	×
13	0	42	0
14	0	12	7
15	0	$+\infty$	×

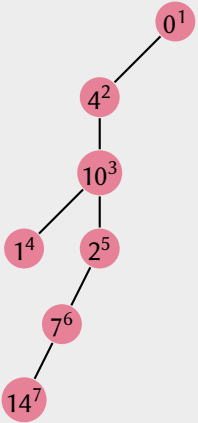


Árboles abarcadores de costo mínimo

Ejemplo del algoritmo de Prim

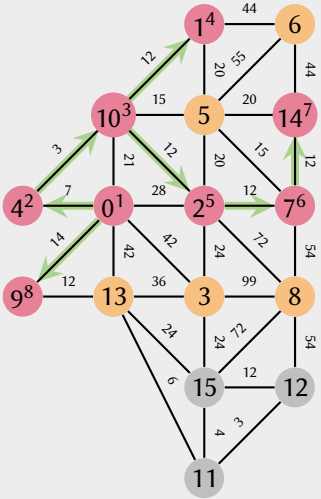


u	v[u]	p[u]	a[u]
0	1	0	0
1	4	12	10
2	5	12	10
3	0	24	2
4	2	7	0
5	0	15	10
6	0	44	1
7	6	12	2
8	0	54	7
9	0	14	0
10	3	3	4
11	0	$+\infty$	$\times$
12	0	$+\infty$	$\times$
13	0	42	0
14	7	12	7
15	0	$+\infty$	$\times$

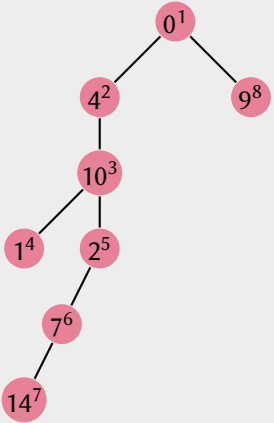


Árboles abarcadores de costo mínimo

Ejemplo del algoritmo de Prim

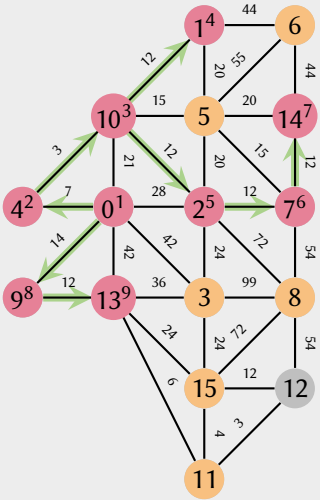


u	v[u]	p[u]	a[u]
0	1	0	0
1	4	12	10
2	5	12	10
3	0	24	2
4	2	7	0
5	0	15	10
6	0	44	1
7	6	12	2
8	0	54	7
9	8	14	0
10	3	3	4
11	0	$+\infty$	$\times$
12	0	$+\infty$	$\times$
13	0	12	9
14	7	12	7
15	0	$+\infty$	$\times$

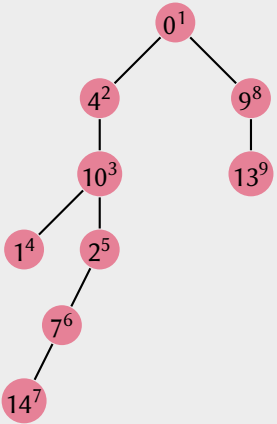


Árboles abarcadores de costo mínimo

Ejemplo del algoritmo de Prim

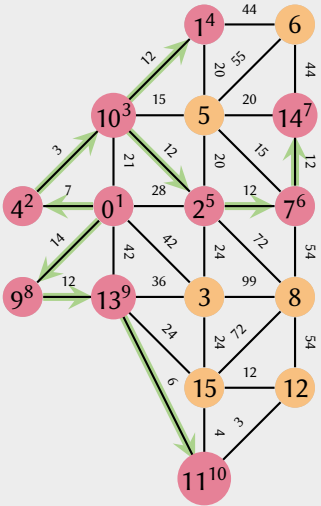


u	v[u]	p[u]	a[u]
0	1	0	0
1	4	12	10
2	5	12	10
3	0	24	2
4	2	7	0
5	0	15	10
6	0	44	1
7	6	12	2
8	0	54	7
9	8	14	0
10	3	3	4
11	0	6	13
12	0	$+\infty$	$\times$
13	9	12	9
14	7	12	7
15	0	24	13

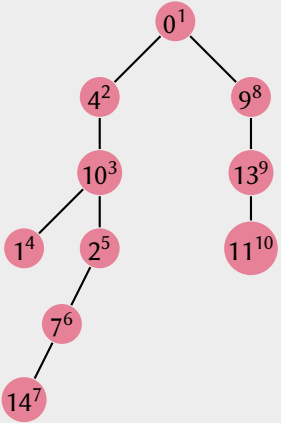


Árboles abarcadores de costo mínimo

Ejemplo del algoritmo de Prim

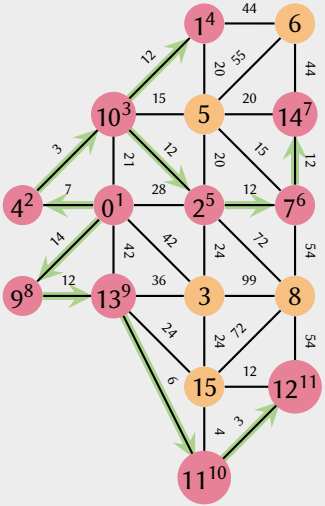


u	v[u]	p[u]	a[u]
0	1	0	0
1	4	12	10
2	5	12	10
3	0	24	2
4	2	7	0
5	0	15	10
6	0	44	1
7	6	12	2
8	0	54	7
9	8	14	0
10	3	3	4
11	10	6	13
12	0	3	11
13	9	12	9
14	7	12	7
15	0	4	11

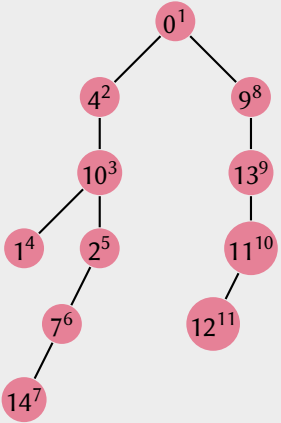


Árboles abarcadores de costo mínimo

Ejemplo del algoritmo de Prim

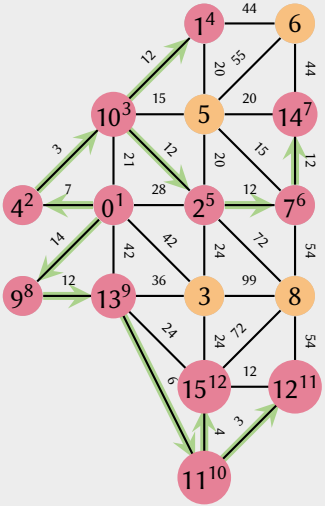


u	v[u]	p[u]	a[u]
0	1	0	0
1	4	12	10
2	5	12	10
3	0	24	2
4	2	7	0
5	0	15	10
6	0	44	1
7	6	12	2
8	0	54	7
9	8	14	0
10	3	3	4
11	10	6	13
12	11	3	11
13	9	12	9
14	7	12	7
15	0	4	11

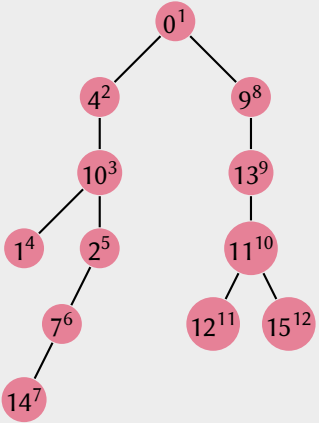


Árboles abarcadores de costo mínimo

Ejemplo del algoritmo de Prim

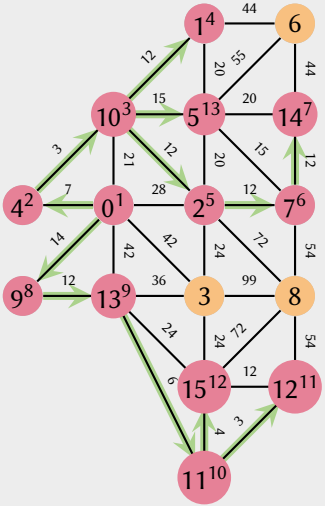


u	v[u]	p[u]	a[u]
0	1	0	0
1	4	12	10
2	5	12	10
3	0	24	2
4	2	7	0
5	0	15	10
6	0	44	1
7	6	12	2
8	0	54	7
9	8	14	0
10	3	3	4
11	10	6	13
12	11	3	11
13	9	12	9
14	7	12	7
15	12	4	11

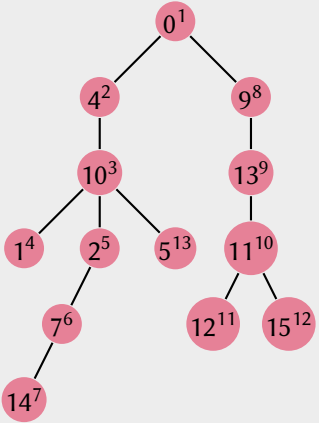


Árboles abarcadores de costo mínimo

Ejemplo del algoritmo de Prim

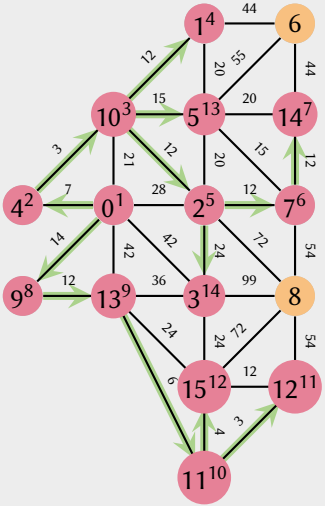


u	v[u]	p[u]	a[u]
0	1	0	0
1	4	12	10
2	5	12	10
3	0	24	2
4	2	7	0
5	13	15	10
6	0	44	1
7	6	12	2
8	0	54	7
9	8	14	0
10	3	3	4
11	10	6	13
12	11	3	11
13	9	12	9
14	7	12	7
15	12	4	11

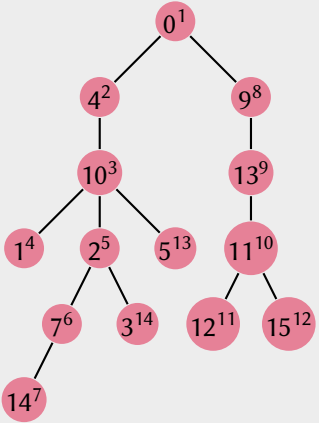


Árboles abarcadores de costo mínimo

Ejemplo del algoritmo de Prim

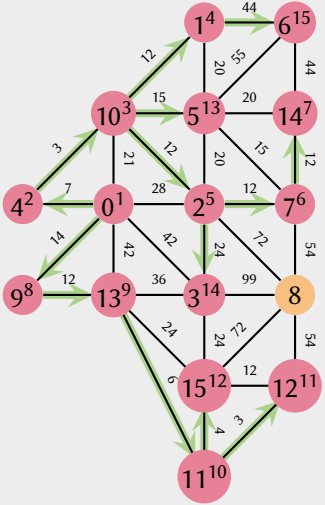


u	v[u]	p[u]	a[u]
0	1	0	0
1	4	12	10
2	5	12	10
3	14	24	2
4	2	7	0
5	13	15	10
6	0	44	1
7	6	12	2
8	0	54	7
9	8	14	0
10	3	3	4
11	10	6	13
12	11	3	11
13	9	12	9
14	7	12	7
15	12	4	11

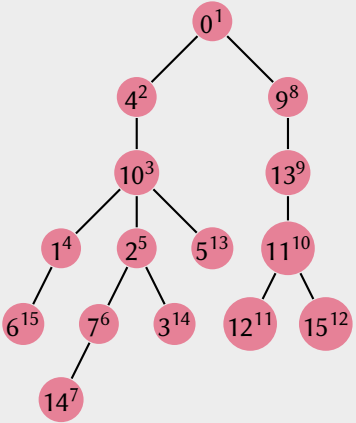


Árboles abarcadores de costo mínimo

Ejemplo del algoritmo de Prim

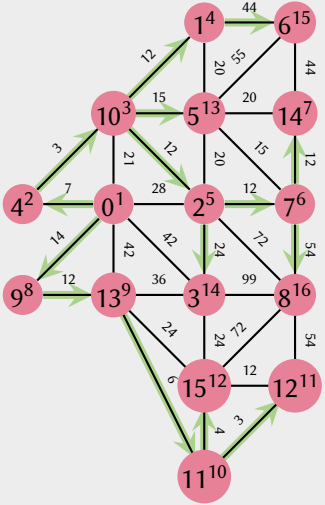


u	v[u]	p[u]	a[u]
0	1	0	0
1	4	12	10
2	5	12	10
3	14	24	2
4	2	7	0
5	13	15	10
6	15	44	1
7	6	12	2
8	0	54	7
9	8	14	0
10	3	3	4
11	10	6	13
12	11	3	11
13	9	12	9
14	7	12	7
15	12	4	11

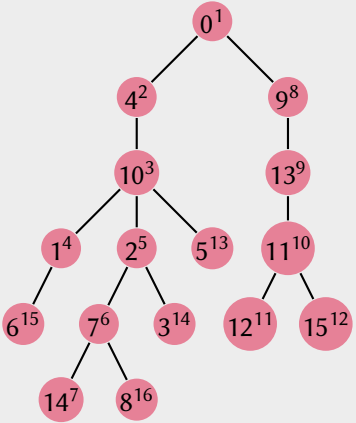


Árboles abarcadores de costo mínimo

Ejemplo del algoritmo de Prim



u	v[u]	p[u]	a[u]
0	1	0	0
1	4	12	10
2	5	12	10
3	14	24	2
4	2	7	0
5	13	15	10
6	15	44	1
7	6	12	2
8	16	54	7
9	8	14	0
10	3	3	4
11	10	6	13
12	11	3	11
13	9	12	9
14	7	12	7
15	12	4	11



Árboles abarcadores de costo mínimo

Algoritmo de Kruskal

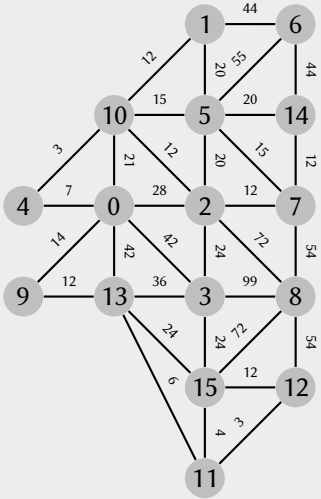
El algoritmo de Kruskal es un algoritmo glotón que mantiene un bosque de aristas baratas hasta que ese bosque se vuelva conexo. Más precisamente:

1. Se ordenan las aristas de G por costo ascendente $c_{e_1} \leq \ldots \leq c_{e_m}$.
2. Sea $B = \emptyset$ el conjunto de aristas elegidas y sea $i = 1$.
3. Mientras el bosque (V, B) no sea conexo:
    1. Si $B \cup \{e_i\}$ es un bosque, entonces se agrega e_i a B.
    2. Incrementa i.

Preguntar si $B \cup \{e_i\}$ es un bosque es equivalente a preguntar si $B \cup \{e_i\}$ no tiene ciclos.
Preguntar si (V, B) es conexo es equivalente a preguntar si $|B| = n - 1$.

Árboles abarcadores de costo mínimo

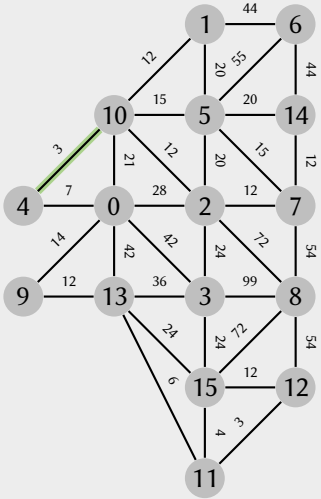
Ejemplo del algoritmo de Kruskal



(u, v), $c_{u,v}$	(4,10), 3	(11,12), 3	(11,15), 4	(11,13), 6
(0,4), 7	(1,10), 12	(2,7), 12	(2,10), 12	(7,14), 12
(9,13), 12	(12,15), 12	(0,9), 14	(5,7), 15	(5,10), 15
(1,5), 20	(2,5), 20	(5,14), 20	(0,10), 21	(2,3), 24
(3,15), 24	(13,15), 24	(0,2), 28	(3,13), 36	(0,3), 42
(0,13), 42	(1,6), 44	(6,14), 44	(7,8), 54	(8,12), 54
(5,6), 55	(2,8), 72	(8,15), 72	(3,8), 99	

Árboles abarcadores de costo mínimo

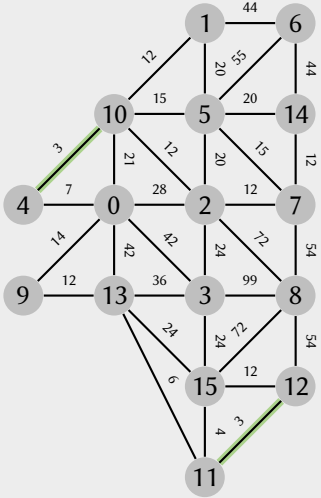
Ejemplo del algoritmo de Kruskal



$(u, v), c_{u,v}$	(4,10), 3	(11,12), 3	(11,15), 4	(11,13), 6
(0,4), 7	(1,10), 12	(2,7), 12	(2,10), 12	(7,14), 12
(9,13), 12	(12,15), 12	(0,9), 14	(5,7), 15	(5,10), 15
(1,5), 20	(2,5), 20	(5,14), 20	(0,10), 21	(2,3), 24
(3,15), 24	(13,15), 24	(0,2), 28	(3,13), 36	(0,3), 42
(0,13), 42	(1,6), 44	(6,14), 44	(7,8), 54	(8,12), 54
(5,6), 55	(2,8), 72	(8,15), 72	(3,8), 99	

Árboles abarcadores de costo mínimo

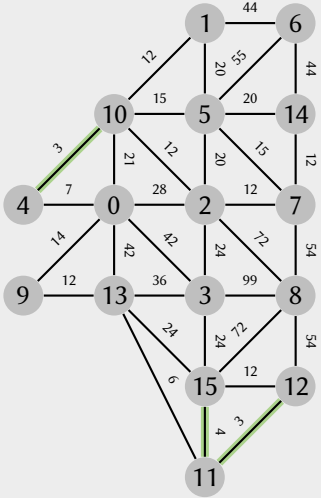
Ejemplo del algoritmo de Kruskal



(u, v), $c_{u,v}$	(4,10), 3	(11,12), 3	(11,15), 4	(11,13), 6
(0,4), 7	(1,10), 12	(2,7), 12	(2,10), 12	(7,14), 12
(9,13), 12	(12,15), 12	(0,9), 14	(5,7), 15	(5,10), 15
(1,5), 20	(2,5), 20	(5,14), 20	(0,10), 21	(2,3), 24
(3,15), 24	(13,15), 24	(0,2), 28	(3,13), 36	(0,3), 42
(0,13), 42	(1,6), 44	(6,14), 44	(7,8), 54	(8,12), 54
(5,6), 55	(2,8), 72	(8,15), 72	(3,8), 99	

Árboles abarcadores de costo mínimo

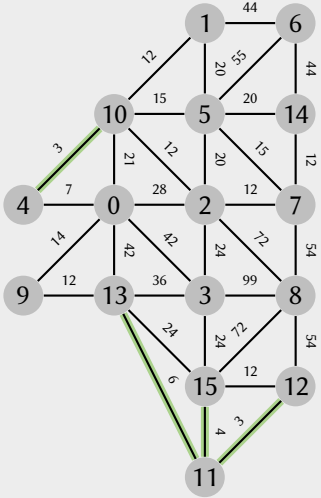
Ejemplo del algoritmo de Kruskal



(u, v), $c_{u,v}$	(4,10), 3	(11,12), 3	(11,15), 4	(11,13), 6
(0,4), 7	(1,10), 12	(2,7), 12	(2,10), 12	(7,14), 12
(9,13), 12	(12,15), 12	(0,9), 14	(5,7), 15	(5,10), 15
(1,5), 20	(2,5), 20	(5,14), 20	(0,10), 21	(2,3), 24
(3,15), 24	(13,15), 24	(0,2), 28	(3,13), 36	(0,3), 42
(0,13), 42	(1,6), 44	(6,14), 44	(7,8), 54	(8,12), 54
(5,6), 55	(2,8), 72	(8,15), 72	(3,8), 99	

Árboles abarcadores de costo mínimo

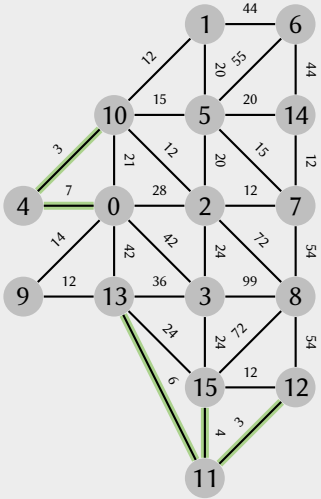
Ejemplo del algoritmo de Kruskal



(u, v), $c_{u,v}$	(4,10), 3	(11,12), 3	(11,15), 4	(11,13), 6
(0,4), 7	(1,10), 12	(2,7), 12	(2,10), 12	(7,14), 12
(9,13), 12	(12,15), 12	(0,9), 14	(5,7), 15	(5,10), 15
(1,5), 20	(2,5), 20	(5,14), 20	(0,10), 21	(2,3), 24
(3,15), 24	(13,15), 24	(0,2), 28	(3,13), 36	(0,3), 42
(0,13), 42	(1,6), 44	(6,14), 44	(7,8), 54	(8,12), 54
(5,6), 55	(2,8), 72	(8,15), 72	(3,8), 99	

Árboles abarcadores de costo mínimo

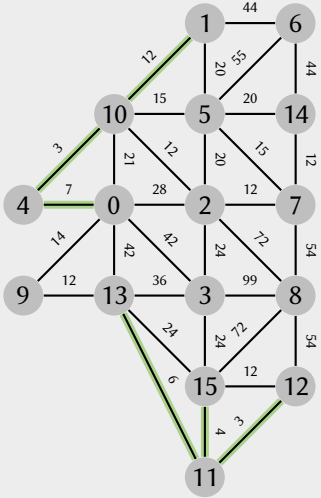
Ejemplo del algoritmo de Kruskal



(u, v), $c_{u,v}$	(4,10), 3	(11,12), 3	(11,15), 4	(11,13), 6
(0,4), 7	(1,10), 12	(2,7), 12	(2,10), 12	(7,14), 12
(9,13), 12	(12,15), 12	(0,9), 14	(5,7), 15	(5,10), 15
(1,5), 20	(2,5), 20	(5,14), 20	(0,10), 21	(2,3), 24
(3,15), 24	(13,15), 24	(0,2), 28	(3,13), 36	(0,3), 42
(0,13), 42	(1,6), 44	(6,14), 44	(7,8), 54	(8,12), 54
(5,6), 55	(2,8), 72	(8,15), 72	(3,8), 99	

Árboles abarcadores de costo mínimo

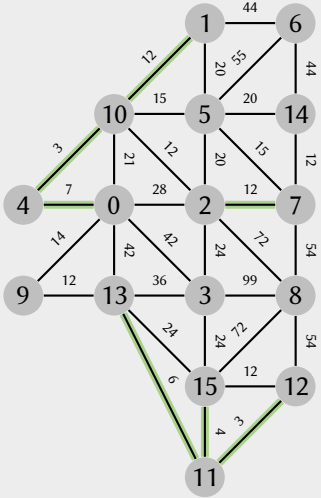
Ejemplo del algoritmo de Kruskal



(u, v), $c_{u,v}$	(4,10), 3	(11,12), 3	(11,15), 4	(11,13), 6
(0,4), 7	(1,10), 12	(2,7), 12	(2,10), 12	(7,14), 12
(9,13), 12	(12,15), 12	(0,9), 14	(5,7), 15	(5,10), 15
(1,5), 20	(2,5), 20	(5,14), 20	(0,10), 21	(2,3), 24
(3,15), 24	(13,15), 24	(0,2), 28	(3,13), 36	(0,3), 42
(0,13), 42	(1,6), 44	(6,14), 44	(7,8), 54	(8,12), 54
(5,6), 55	(2,8), 72	(8,15), 72	(3,8), 99	

Árboles abarcadores de costo mínimo

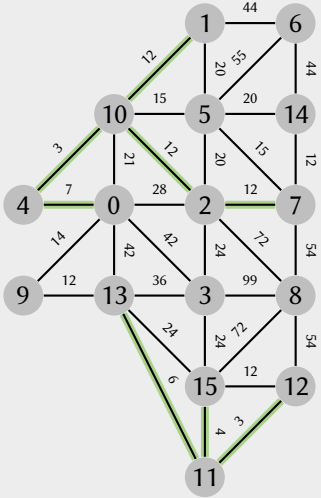
Ejemplo del algoritmo de Kruskal



(u, v), $c_{u,v}$	(4,10), 3	(11,12), 3	(11,15), 4	(11,13), 6
(0,4), 7	(1,10), 12	(2,7), 12	(2,10), 12	(7,14), 12
(9,13), 12	(12,15), 12	(0,9), 14	(5,7), 15	(5,10), 15
(1,5), 20	(2,5), 20	(5,14), 20	(0,10), 21	(2,3), 24
(3,15), 24	(13,15), 24	(0,2), 28	(3,13), 36	(0,3), 42
(0,13), 42	(1,6), 44	(6,14), 44	(7,8), 54	(8,12), 54
(5,6), 55	(2,8), 72	(8,15), 72	(3,8), 99	

Árboles abarcadores de costo mínimo

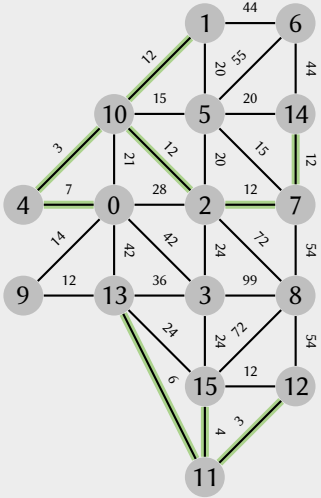
Ejemplo del algoritmo de Kruskal



(u, v), $c_{u,v}$	(4,10), 3	(11,12), 3	(11,15), 4	(11,13), 6
(0,4), 7	(1,10), 12	(2,7), 12	(2,10), 12	(7,14), 12
(9,13), 12	(12,15), 12	(0,9), 14	(5,7), 15	(5,10), 15
(1,5), 20	(2,5), 20	(5,14), 20	(0,10), 21	(2,3), 24
(3,15), 24	(13,15), 24	(0,2), 28	(3,13), 36	(0,3), 42
(0,13), 42	(1,6), 44	(6,14), 44	(7,8), 54	(8,12), 54
(5,6), 55	(2,8), 72	(8,15), 72	(3,8), 99	

Árboles abarcadores de costo mínimo

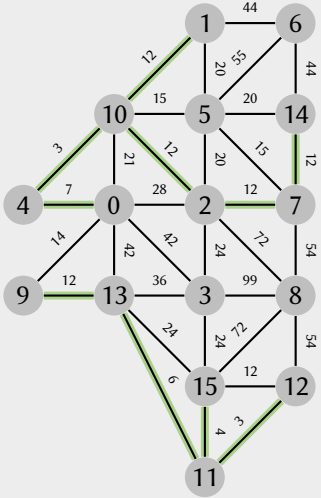
Ejemplo del algoritmo de Kruskal



(u, v), $c_{u,v}$	(4,10), 3	(11,12), 3	(11,15), 4	(11,13), 6
(0,4), 7	(1,10), 12	(2,7), 12	(2,10), 12	(7,14), 12
(9,13), 12	(12,15), 12	(0,9), 14	(5,7), 15	(5,10), 15
(1,5), 20	(2,5), 20	(5,14), 20	(0,10), 21	(2,3), 24
(3,15), 24	(13,15), 24	(0,2), 28	(3,13), 36	(0,3), 42
(0,13), 42	(1,6), 44	(6,14), 44	(7,8), 54	(8,12), 54
(5,6), 55	(2,8), 72	(8,15), 72	(3,8), 99	

Árboles abarcadores de costo mínimo

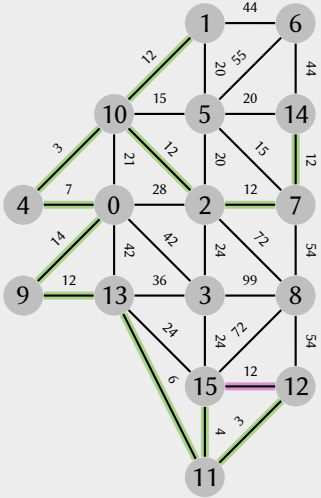
Ejemplo del algoritmo de Kruskal



(u, v), $c_{u,v}$	(4,10), 3	(11,12), 3	(11,15), 4	(11,13), 6
(0,4), 7	(1,10), 12	(2,7), 12	(2,10), 12	(7,14), 12
(9,13), 12	(12,15), 12	(0,9), 14	(5,7), 15	(5,10), 15
(1,5), 20	(2,5), 20	(5,14), 20	(0,10), 21	(2,3), 24
(3,15), 24	(13,15), 24	(0,2), 28	(3,13), 36	(0,3), 42
(0,13), 42	(1,6), 44	(6,14), 44	(7,8), 54	(8,12), 54
(5,6), 55	(2,8), 72	(8,15), 72	(3,8), 99	

Árboles abarcadores de costo mínimo

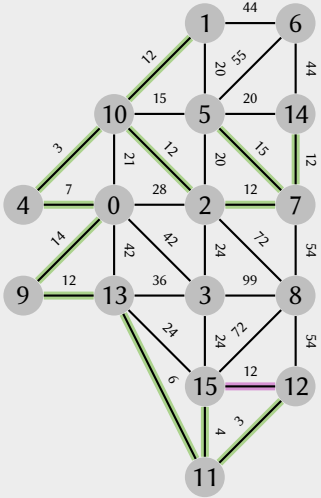
Ejemplo del algoritmo de Kruskal



(u, v), $c_{u,v}$	(4,10), 3	(11,12), 3	(11,15), 4	(11,13), 6
(0,4), 7	(1,10), 12	(2,7), 12	(2,10), 12	(7,14), 12
(9,13), 12	(12,15), 12	(0,9), 14	(5,7), 15	(5,10), 15
(1,5), 20	(2,5), 20	(5,14), 20	(0,10), 21	(2,3), 24
(3,15), 24	(13,15), 24	(0,2), 28	(3,13), 36	(0,3), 42
(0,13), 42	(1,6), 44	(6,14), 44	(7,8), 54	(8,12), 54
(5,6), 55	(2,8), 72	(8,15), 72	(3,8), 99	

Árboles abarcadores de costo mínimo

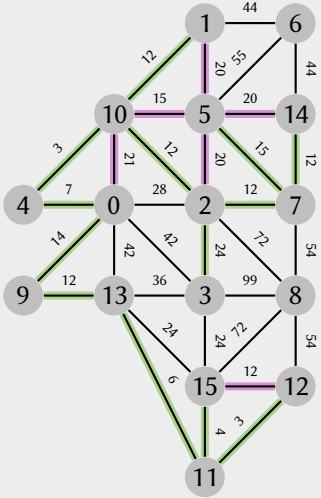
Ejemplo del algoritmo de Kruskal



(u, v), $c_{u,v}$	(4,10), 3	(11,12), 3	(11,15), 4	(11,13), 6
(0,4), 7	(1,10), 12	(2,7), 12	(2,10), 12	(7,14), 12
(9,13), 12	(12,15), 12	(0,9), 14	(5,7), 15	(5,10), 15
(1,5), 20	(2,5), 20	(5,14), 20	(0,10), 21	(2,3), 24
(3,15), 24	(13,15), 24	(0,2), 28	(3,13), 36	(0,3), 42
(0,13), 42	(1,6), 44	(6,14), 44	(7,8), 54	(8,12), 54
(5,6), 55	(2,8), 72	(8,15), 72	(3,8), 99	

Árboles abarcadores de costo mínimo

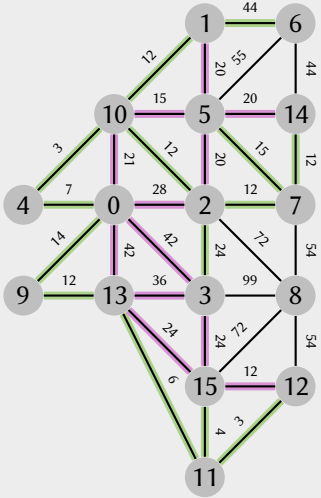
Ejemplo del algoritmo de Kruskal



(u, v), $c_{u,v}$	(4,10), 3	(11,12), 3	(11,15), 4	(11,13), 6
(0,4), 7	(1,10), 12	(2,7), 12	(2,10), 12	(7,14), 12
(9,13), 12	(12,15), 12	(0,9), 14	(5,7), 15	(5,10), 15
(1,5), 20	(2,5), 20	(5,14), 20	(0,10), 21	(2,3), 24
(3,15), 24	(13,15), 24	(0,2), 28	(3,13), 36	(0,3), 42
(0,13), 42	(1,6), 44	(6,14), 44	(7,8), 54	(8,12), 54
(5,6), 55	(2,8), 72	(8,15), 72	(3,8), 99	

Árboles abarcadores de costo mínimo

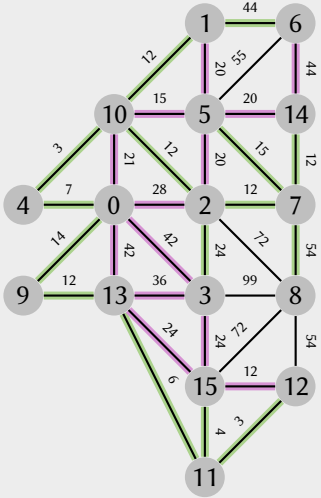
Ejemplo del algoritmo de Kruskal



(u, v), $c_{u,v}$	(4,10), 3	(11,12), 3	(11,15), 4	(11,13), 6
(0,4), 7	(1,10), 12	(2,7), 12	(2,10), 12	(7,14), 12
(9,13), 12	(12,15), 12	(0,9), 14	(5,7), 15	(5,10), 15
(1,5), 20	(2,5), 20	(5,14), 20	(0,10), 21	(2,3), 24
(3,15), 24	(13,15), 24	(0,2), 28	(3,13), 36	(0,3), 42
(0,13), 42	(1,6), 44	(6,14), 44	(7,8), 54	(8,12), 54
(5,6), 55	(2,8), 72	(8,15), 72	(3,8), 99	

Árboles abarcadores de costo mínimo

Ejemplo del algoritmo de Kruskal



(u, v), $c_{u,v}$	(4,10), 3	(11,12), 3	(11,15), 4	(11,13), 6
(0,4), 7	(1,10), 12	(2,7), 12	(2,10), 12	(7,14), 12
(9,13), 12	(12,15), 12	(0,9), 14	(5,7), 15	(5,10), 15
(1,5), 20	(2,5), 20	(5,14), 20	(0,10), 21	(2,3), 24
(3,15), 24	(13,15), 24	(0,2), 28	(3,13), 36	(0,3), 42
(0,13), 42	(1,6), 44	(6,14), 44	(7,8), 54	(8,12), 54
(5,6), 55	(2,8), 72	(8,15), 72	(3,8), 99	

Árboles abarcadores de costo mínimo

Detalles de implementación del algoritmo de Kruskal

En este caso será conveniente que la red esté representada por sus listas de adyacencia. De este modo se puede generar un vector de sus m aristas (u, v) con costos $c_{u,v}$ en $n + m$ pasos. Para evitar duplicaciones, sólo almacenaremos el caso $u < v$.

Para ordenar este vector por costo ascendente usaremos un algoritmo de ordenamiento eficiente que sólo requiera $m \log_2 m$ pasos (ordenamiento por mezcla o por montículo).

Para saber si agregar la arista (u, v) forma un ciclo con B, se podría usar búsqueda en profundidad desde u para ver si se puede llegar a v en B. Esto puede requerir hasta n pasos por iteración. Como puede haber hasta m iteraciones, esto es demasiado lento.

Veamos una estructura de datos que permite hacer esto en $\approx \log_2 n$ pasos por iteración.

Unión y búsqueda

Particiones

Sea $V = \{0, 1, \ldots, n-1\}$ y sea $V_0, \ldots, V_{k-1}$ una partición de V en $k \geq 1$ subconjuntos disjuntos no vacíos. Se desea poder hacer dos operaciones:

Búsqueda Dado un $u \in V$ ¿en cuál $V_i \equiv V(u)$ se encuentra u?

Unión Dados $u, v \in V$, reemplazar $V(u)$ y $V(v)$ por $V(u) \cup V(v)$.

Como identificación, cada V_i contendrá un elemento representante v_i. Al inicio, $k = n$ y $V_i = \{i\}$, es decir, $v_i = i$. Esto irá cambiando conforme se hagan operaciones de unión.

Aplicación en el algoritmo de Kruskal

Al principio hay n componentes conexas formadas por un solo vértice. Después, cuando se procese la arista uv, pueden pasar dos cosas: Si u y v están en la misma componente conexa entonces añadirla formaría un ciclo. Si u y v no están en la misma componente conexa entonces añadirla conecta dos componentes conexas para formar una sola.

Unión y búsqueda

Implementación de la partición

Representaremos las particiones con un bosque invertido, formado por dos arreglos de n elementos: un arreglo rep de representantes y un arreglo car de cardinalidades.

```
typedef struct {
  int n;
  int *rep;
  int *car;
} particion;
```

Unión y búsqueda

Implementación de la inicialización

Al inicio, cada elemento i de V es el representante de $V_i = \{i\}$ de cardinalidad 1. En otras palabras, i es la raíz de su propio árbol invertido.

```
particion creaParticion(int n) {
  particion p;
  p.n = n;
  p.rep = (int *) malloc(p.n*sizeof(int));
  p.car = (int *) malloc(p.n*sizeof(int));
  for (int i = 0; i < n; i++) {
    p.rep[i] = i;
    p.car[i] = 1;
  }
  return p;
}
```

Unión y búsqueda

Implementación de la búsqueda

Dado un elemento u, su precursor es el elemento p.rep[u]. Cuando estos dos sean iguales, entonces u será la raíz de su propio árbol y, por lo tanto, será su representante.

```
int buscaParte(int u, particion p) {
  while (u != p.rep[u])
    u = p.rep[u];
  return u;
}
```

Esta función hará, en el peor de los casos, tantos pasos como la altura del árbol de u. Aquí la clave será evitar que esa altura se vuelva muy grande.

Unión y búsqueda

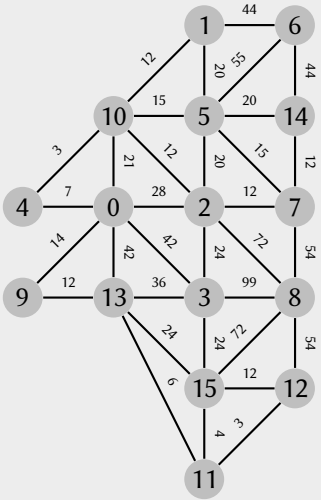
Implementación de la unión

Dados dos elementos u y v, buscaremos sus representantes. Si son distintos, la raíz del árbol con más elementos será la nueva precursora de la raíz del otro árbol y se actualiza su cuenta de elementos. Esta simple regla garantiza que la altura no exceda $1 + \lceil \log_2 n \rceil$.

```
void unePartes(int u, int v, particion p) {
  u = buscaParte(u, p);
  v = buscaParte(v, p);
  if (u != v) {
    if (p.car[u] < p.car[v])
      intercambia(&u, &v);
    p.rep[v] = u;
    p.car[u] += p.car[v];
  }
}
```

Unión y búsqueda

Ejemplo del algoritmo de Kruskal



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

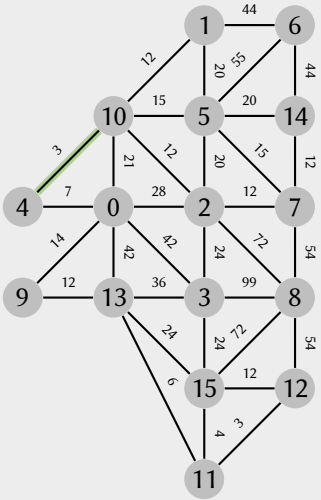
15^1 14^1 13^1 12^1 11^1 10^1 9^1 8^1

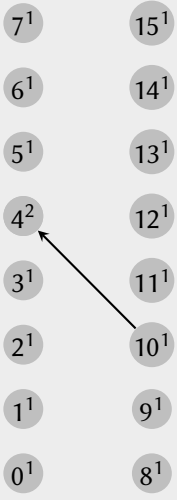
$(u, v), c_{u,v}$

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

Unión y búsqueda

Ejemplo del algoritmo de Kruskal



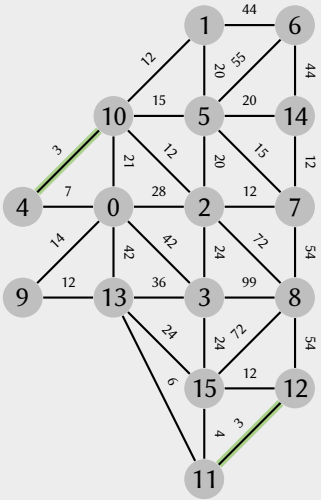


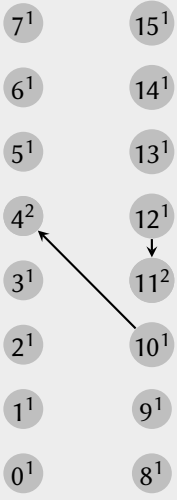
$(u, v), c_{u,v}$

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

Unión y búsqueda

Ejemplo del algoritmo de Kruskal



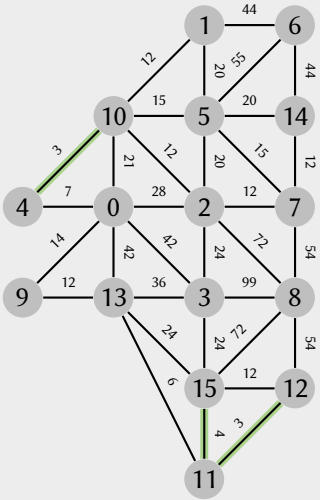


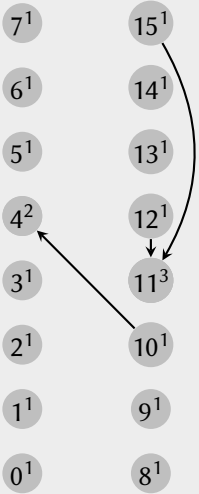
(u, v), $c_{u,v}$

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

Unión y búsqueda

Ejemplo del algoritmo de Kruskal



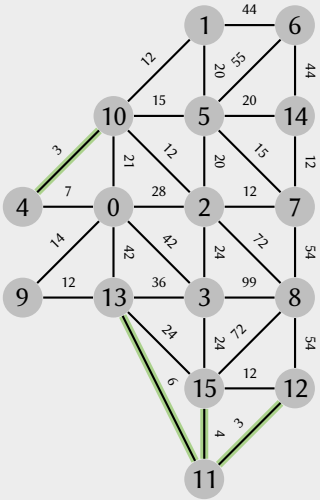


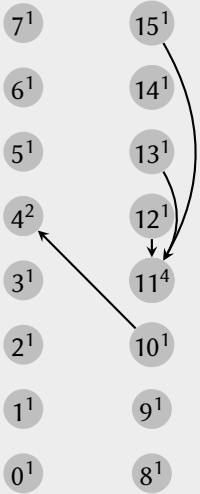
$(u, v), c_{u,v}$

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

Unión y búsqueda

Ejemplo del algoritmo de Kruskal



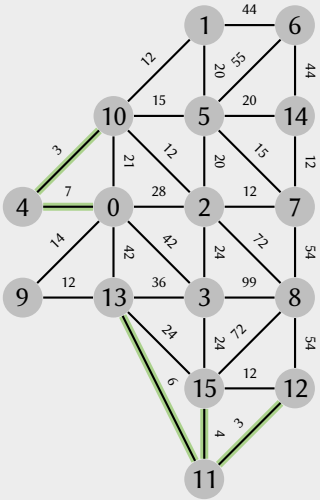


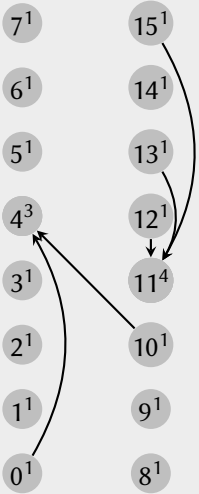
$(u, v), c_{u,v}$

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

Unión y búsqueda

Ejemplo del algoritmo de Kruskal



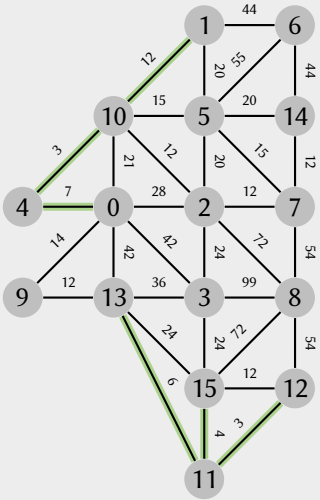


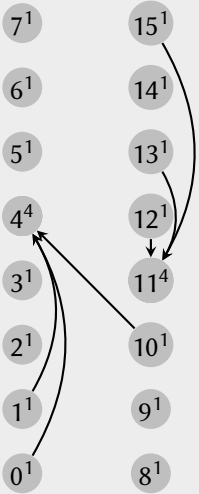
(u, v), $c_{u,v}$

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

Unión y búsqueda

Ejemplo del algoritmo de Kruskal



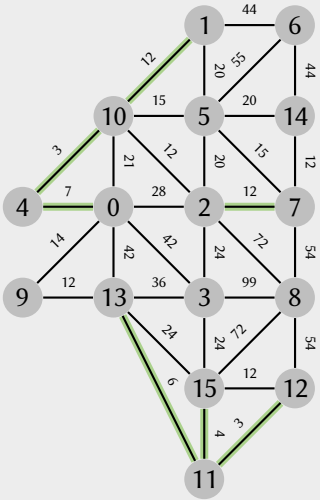


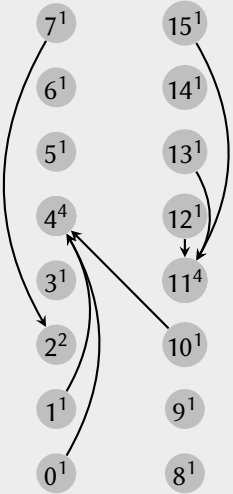
(u, v), $c_{u,v}$

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

Unión y búsqueda

Ejemplo del algoritmo de Kruskal



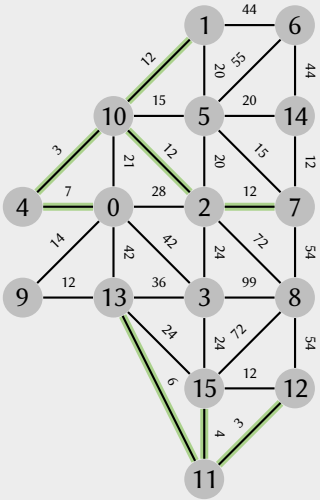


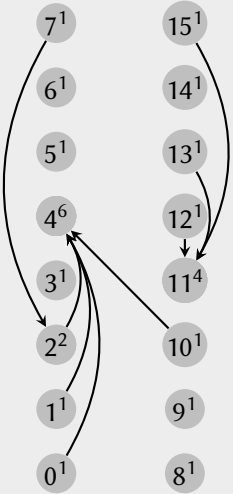
(u, v), $c_{u,v}$

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

Unión y búsqueda

Ejemplo del algoritmo de Kruskal



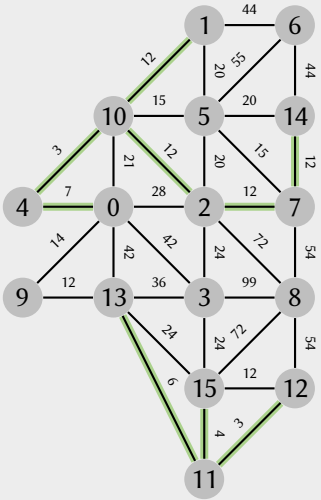


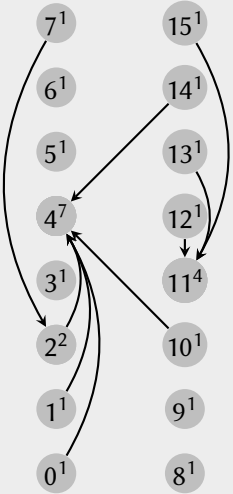
(u, v), $c_{u,v}$

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

Unión y búsqueda

Ejemplo del algoritmo de Kruskal



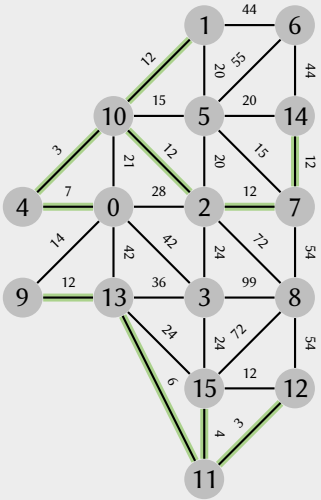


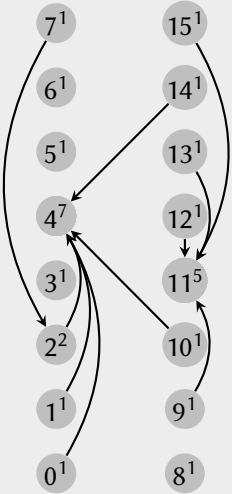
(u, v), $c_{u,v}$

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

Unión y búsqueda

Ejemplo del algoritmo de Kruskal



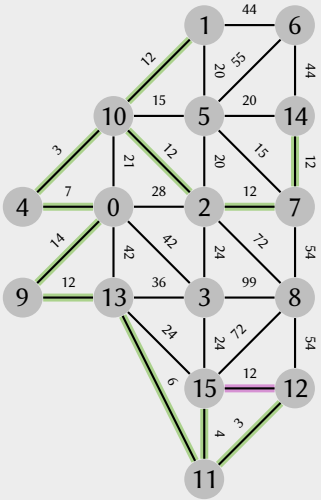


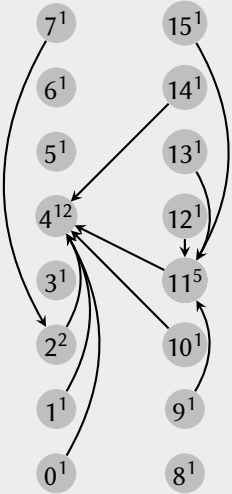
(u, v), $c_{u,v}$

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

Unión y búsqueda

Ejemplo del algoritmo de Kruskal



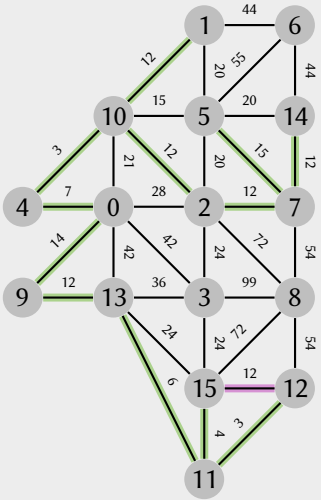


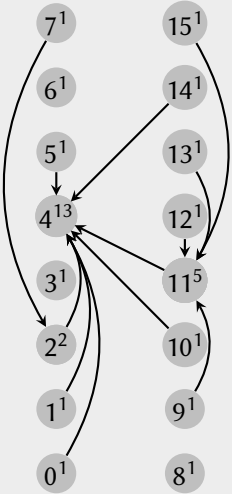
(u, v), $c_{u,v}$

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

Unión y búsqueda

Ejemplo del algoritmo de Kruskal



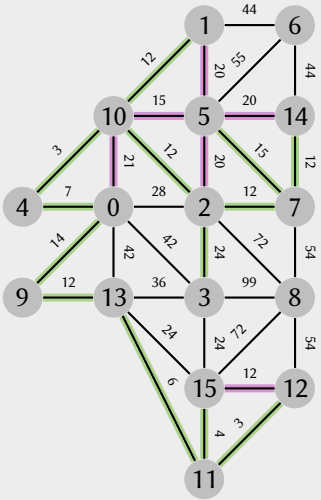


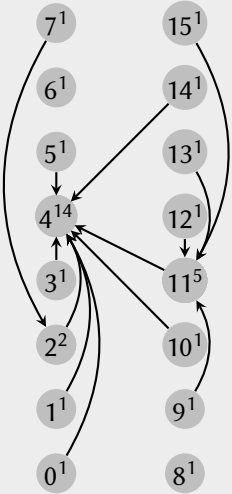
(u, v), $c_{u,v}$

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

Unión y búsqueda

Ejemplo del algoritmo de Kruskal



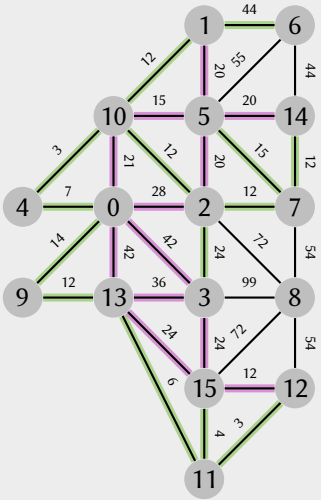


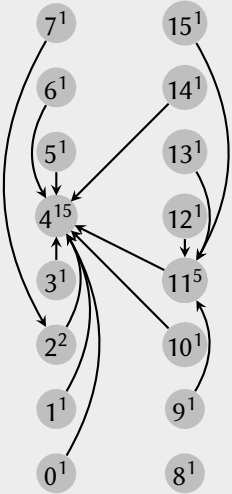
$(u, v), c_{u,v}$

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

Unión y búsqueda

Ejemplo del algoritmo de Kruskal



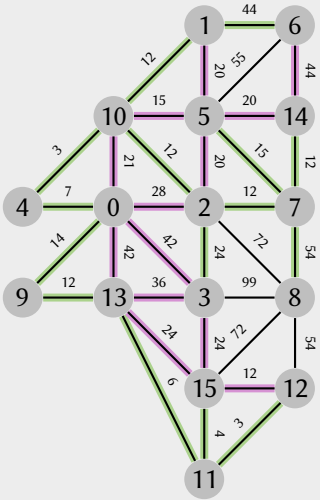


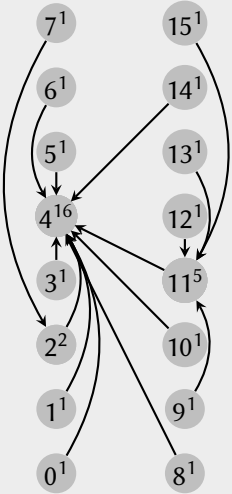
$(u, v), c_{u,v}$

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

Unión y búsqueda

Ejemplo del algoritmo de Kruskal





$(u, v), c_{u,v}$

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

Árboles abarcadores de costo mínimo

Observaciones

1. Tanto el algoritmo de Prim como el algoritmo de Kruskal se pueden adaptar fácilmente para obtener árboles abarcadores de costo máximo.
2. El algoritmo de Prim se puede implementar también con listas de adyacencia, pero para obtener el vértice no visto de menor prioridad la búsqueda lineal sería muy lenta. En este caso se requiere una implementación de montículos que permita disminuir la prioridad de un vértice. Esto se puede hacer en $\approx \log_2 n$ pasos, por lo que el tiempo de ejecución del algoritmo de Prim es proporcional a $(n + m) \log_2 n$.
3. En la estructura de unión y búsqueda se puede hacer que el rango sea directamente la altura del árbol. La raíz del árbol más alto (altura h_u) se vuelve el precursor de la raíz del otro árbol (altura h_v). La nueva altura de u será el máximo de h_u y $h_v + 1$.
4. Hay estructuras de unión y búsqueda más eficientes, pero no mejoran en gran medida el tiempo de ejecución del algoritmo de Kruskal.