Árboles de cubrimiento mínimo

La propiedad de corte, Kruskal y Prim

Carlos A Delgado S

Pontificia Universidad Javeriana Cali

Contenido

1. De dónde venimos
2. Conectar al menor costo
3. La propiedad de corte
4. El algoritmo de Kruskal
5. El algoritmo de Prim
6. Cuál de los dos
7. Ejercicios
8. Lo que sigue
9. Referencias

Contenido

1 De dónde venimos

2 Conectar al menor costo

3 La propiedad de corte

4 El algoritmo de Kruskal

5 El algoritmo de Prim

6 Cuál de los dos

7 Ejercicios

8 Lo que sigue

9 Referencias

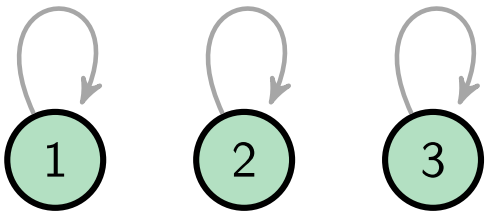
Lo que dejaron los conjuntos disjuntos

operación	qué hace	cuánto cuesta
`make_set(x)`	arma el conjunto $\{x\}$, con x de raíz	$\Theta(1)$
`find_set(x)`	la raíz del conjunto donde está x	$O(\alpha(n))$
`union(x, y)`	cuelga un conjunto del otro	$O(\alpha(n))$

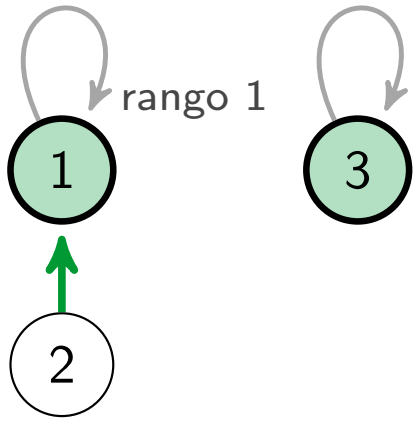
Las dos heurísticas y para qué sirven hoy

La unión por rango cuelga el árbol bajo del alto. La compresión de caminos deja apuntando a la raíz todo lo que encontró de paso. Dos elementos tienen la misma raíz si están en el mismo conjunto, que es la pregunta de si dos ciudades ya quedaron conectadas.

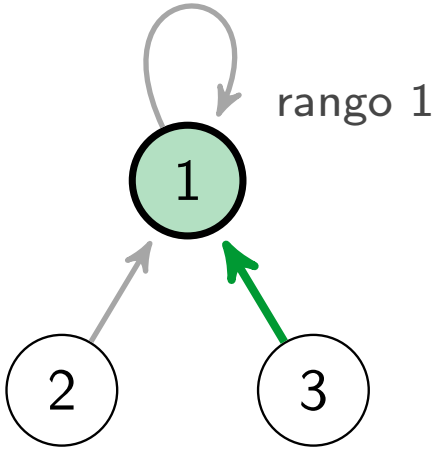
Las tres operaciones, en tres elementos



`make_set` de 1, 2 y 3



`union(1, 2)`



`union(2, 3)`

Qué dejó cada llamada

`make_set` deja a cada uno apuntándose a sí mismo, que es ser raíz, con rango 0. La primera unión encuentra las raíces 1 y 2, iguales de rango, así que cuelga la segunda de la primera y el rango de 1 sube a 1. La segunda pide unir 2 con 3, y lo que se cuelga es la raíz de 2, que ya es 1. Al final `find_set` devuelve 1 para los tres.

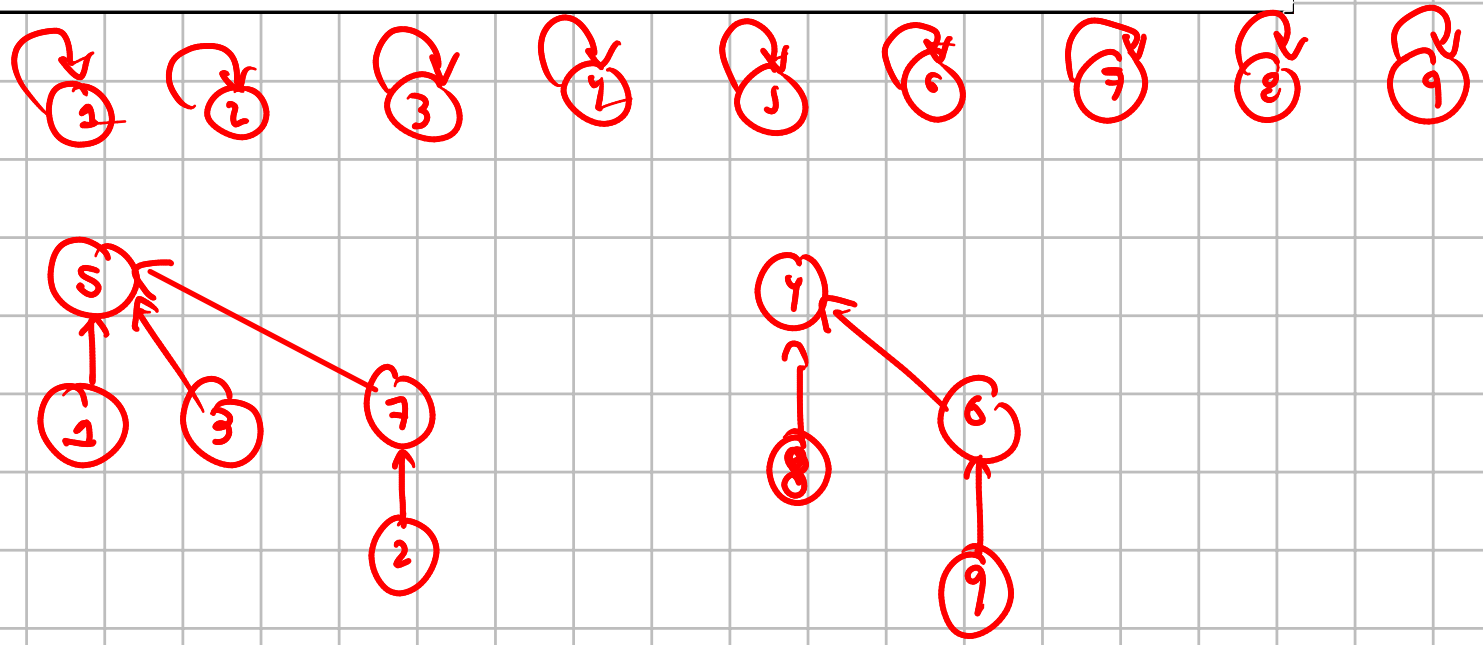
Las siete uniones, una por una

	llamado	raíces	rangos	qué cambia
1	`union(5, 1)`	5, 1	$0 = 0$	$p[1] = 5$ y el rango de 5 sube a 1
2	`union(1, 3)`	5, 3	$1 > 0$	$p[3] = 5$ y el rango se queda
3	`union(7, 2)`	7, 2	$0 = 0$	$p[2] = 7$ y el rango de 7 sube a 1
4	`union(4, 8)`	4, 8	$0 = 0$	$p[8] = 4$ y el rango de 4 sube a 1
5	`union(6, 9)`	6, 9	$0 = 0$	$p[9] = 6$ y el rango de 6 sube a 1
6	`union(3, 2)`	5, 7	$1 = 1$	$p[7] = 5$ y el rango de 5 sube a 2
7	`union(8, 9)`	4, 6	$1 = 1$	$p[6] = 4$ y el rango de 4 sube a 2

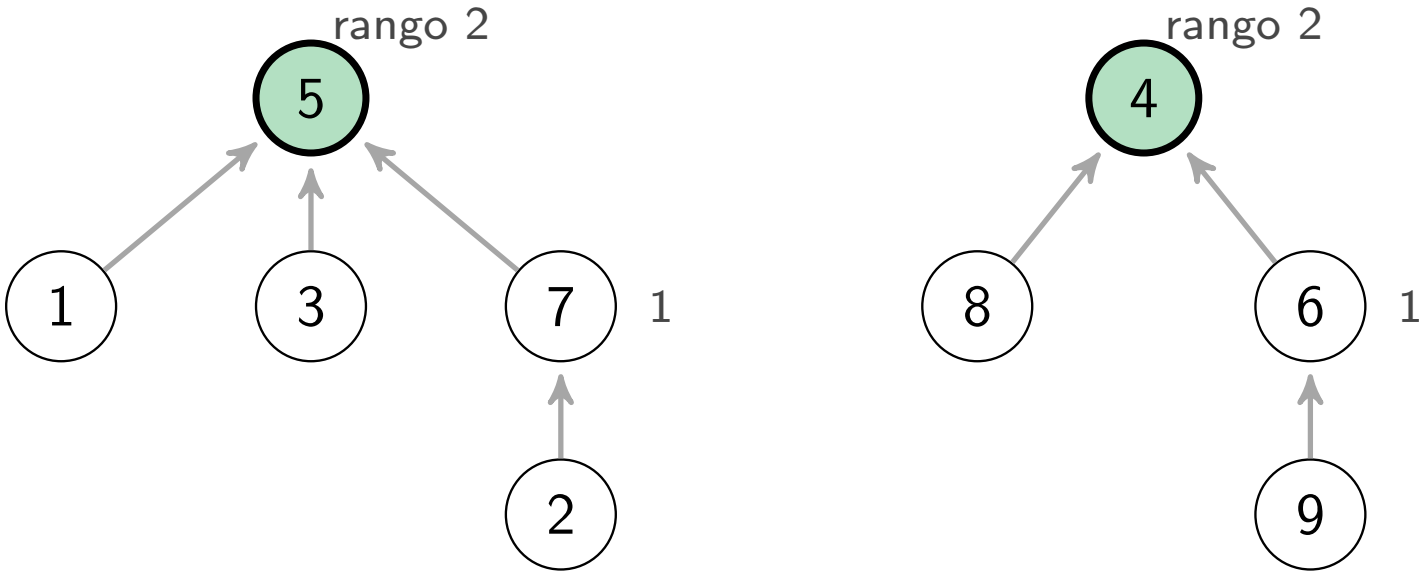
Las dos columnas del medio

Lo que se cuelga son las raíces y no los argumentos: la sexta línea pide unir 3 con 2, y lo que se mueve es 7 debajo de 5. El rango sube solo cuando los dos llegan iguales, y la segunda línea dice por qué: ahí 3 se cuelga de un árbol que ya era más alto, así que la altura no cambió y el rango tampoco.

	llamado	raíces	rangos	qué cambia
1	`union(5, 1)`	5, 1	$0 = 0$	$p[1] = 5$ y el rango de 5 sube a 1
2	`union(1, 3)`	5, 3	$1 > 0$	$p[3] = 5$ y el rango se queda
3	`union(7, 2)`	7, 2	$0 = 0$	$p[2] = 7$ y el rango de 7 sube a 1
4	`union(4, 8)`	4, 8	$0 = 0$	$p[8] = 4$ y el rango de 4 sube a 1
5	`union(6, 9)`	6, 9	$0 = 0$	$p[9] = 6$ y el rango de 6 sube a 1
6	`union(3, 2)`	5, 7	$1 = 1$	$p[7] = 5$ y el rango de 5 sube a 2
7	`union(8, 9)`	4, 6	$1 = 1$	$p[6] = 4$ y el rango de 4 sube a 2



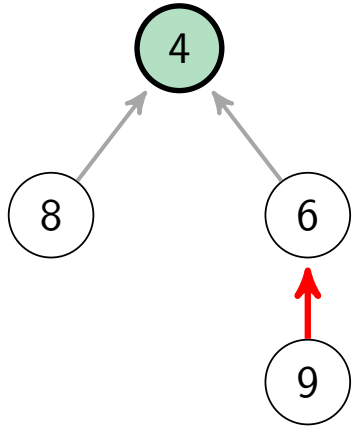
El bosque, tras siete uniones



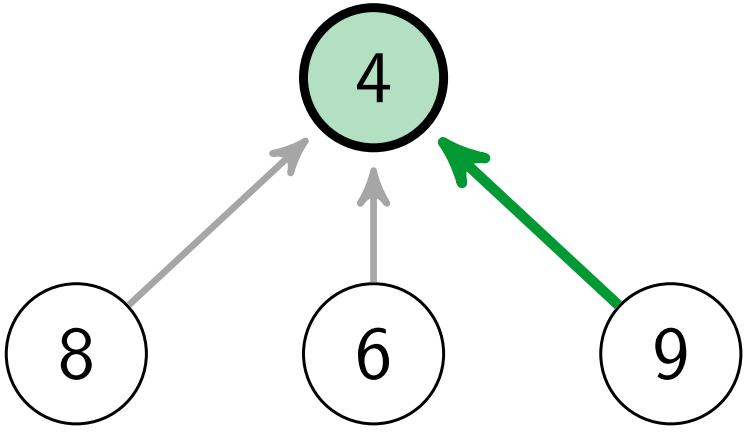
Cómo se lee

Cada flecha va del hijo al padre y la raíz es quien se apunta a sí mismo. Los dos verdes son los representantes: preguntar si 2 y 3 están juntos es subir desde cada uno hasta su raíz y comparar, y los dos llegan a 5. El rango es una cota de la altura, no el tamaño: el de 5 es 2 aunque cuelguen cuatro.

Buscar aplana el camino



antes de `find_set(9)`



después

Las dos heurísticas, en una frase cada una

La unión cuelga la raíz de menor rango de la de mayor, y así el árbol no se alarga sin necesidad. La búsqueda aprovecha el viaje: al volver, reengancha a la raíz todo lo que encontró de paso. Con las dos, una operación cuesta $O(\alpha(n))$, que para cualquier n real no pasa de cuatro.

Para qué más sirven los conjuntos disjuntos

La pregunta que contestan barato

"¿Estos dos ya se alcanzan?", sobre una red a la que le van llegando enlaces. Con un recorrido cada pregunta cuesta $O(V + E)$ y hay que recorrer otra vez en cuanto entre un enlace nuevo; con conjuntos disjuntos la pregunta cuesta $O(\alpha(n))$ y el enlace nuevo es una unión.

Tres sitios donde están

- Segmentación de imágenes: un elemento por píxel, se unen los vecinos de color parecido y cada conjunto queda una región.
- Percolación: se unen las celdas abiertas de un material poroso y se pregunta si el borde de arriba quedó con el de abajo.
- Un laberinto se genera tumbando paredes mientras los dos lados estén en conjuntos distintos.

Contenido

1 De dónde venimos

2 Conectar al menor costo

3 La propiedad de corte

4 El algoritmo de Kruskal

5 El algoritmo de Prim

6 Cuál de los dos

7 Ejercicios

8 Lo que sigue

9 Referencias

Siete barrios y un presupuesto

El encargo

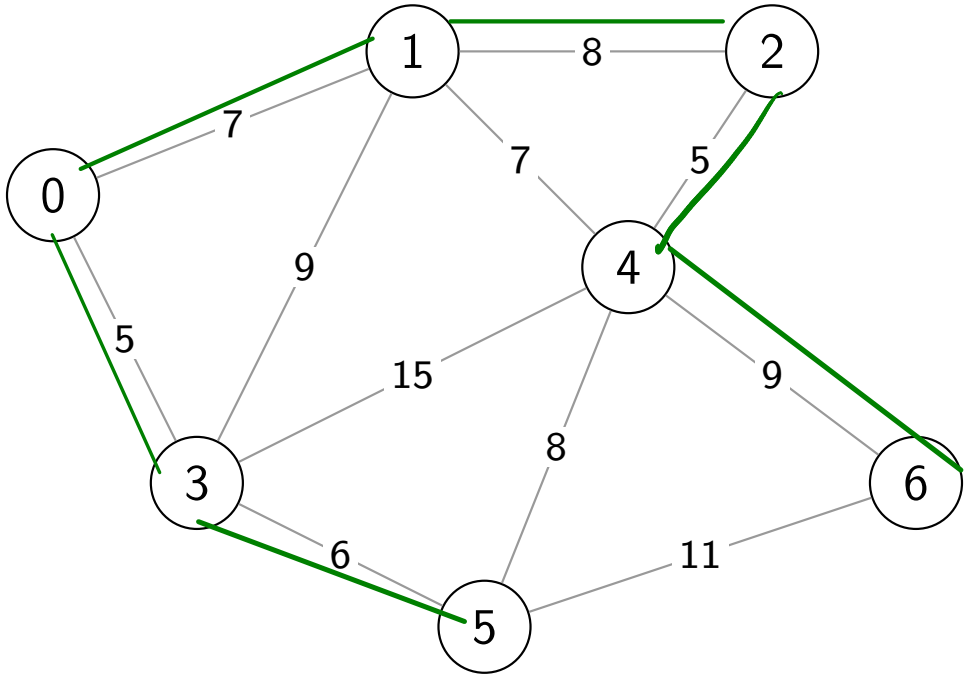
Una empresa tiene que dejar comunicados siete barrios con fibra. No todos los pares se pueden tender, y cada tendido posible tiene su costo en millones. Basta con que cada barrio alcance a los demás, sea directo o pasando por otros.

Antes de mirar el grafo

¿Cuántos tendidos va a tener la respuesta? ¿Puede sobrar alguno?

El grafo de la sesión

0-3 5
0-1 7
1-2 8
2-4 5
3-5 6
4-6 9
40



Pruebe a mano

Escoja seis tendidos que dejen los siete barrios conectados y sume.
Después intente otra selección y compare las dos sumas.

La respuesta es un árbol

Por qué no sobra ninguno

Si la selección deja un ciclo, se quita cualquier arista de ese ciclo y todo sigue conectado, porque sus extremos se alcanzan por el otro lado. La respuesta no tiene ciclos y toca todos los vértices: es un árbol, y tiene $|V| - 1$ aristas.

Definición (CLRS, sección 23, página 624)

Sea $G = (V, E)$ no dirigido y conexo con pesos $w : E \to \mathbb{R}$. Un árbol de cubrimiento mínimo es un subconjunto $E' \subseteq E$ tal que (V, E') es un árbol y $w(E') = \sum_{(u,v)\in E'} w(u, v)$ es el menor posible.

Dónde aparece

Redes que se construyen una vez

Acueductos, fibra, tendido eléctrico, las pistas de una placa de circuito impreso. El primer algoritmo conocido, el de Otakar Borůvka en 1926, salió de llevarle luz eléctrica a los pueblos de Moravia.

Agrupar datos

El agrupamiento por enlace simple es el árbol mínimo sobre las distancias entre los puntos: quitarle las $k - 1$ aristas más caras deja los k grupos. Así se segmentan imágenes, con un vértice por píxel. Y recorrer el árbol de ida y vuelta da un circuito por todas las ciudades que cuesta a lo sumo el doble del mejor, que es la aproximación clásica para el vendedor viajero.

Probar todos no sirve

Cuántos árboles de cubrimiento hay

Este grafo, con siete vértices y once aristas, tiene 141. Si se pudieran tender todos los pares serían $7^5 = 16\,807$, que es la fórmula de Cayley n^{n-2} para el grafo completo.

barrios	árboles de cubrimiento del grafo completo
7	16 807
10	10^8
20	$2{,}6 \times 10^{23}$

Entonces

Enumerar queda descartado desde veinte barrios. Lo que sigue es un algoritmo que va escogiendo aristas de una en una, y un teorema que garantiza que ninguna de esas decisiones hay que deshacerla.

Contenido

1 De dónde venimos

2 Conectar al menor costo

3 La propiedad de corte

4 El algoritmo de Kruskal

5 El algoritmo de Prim

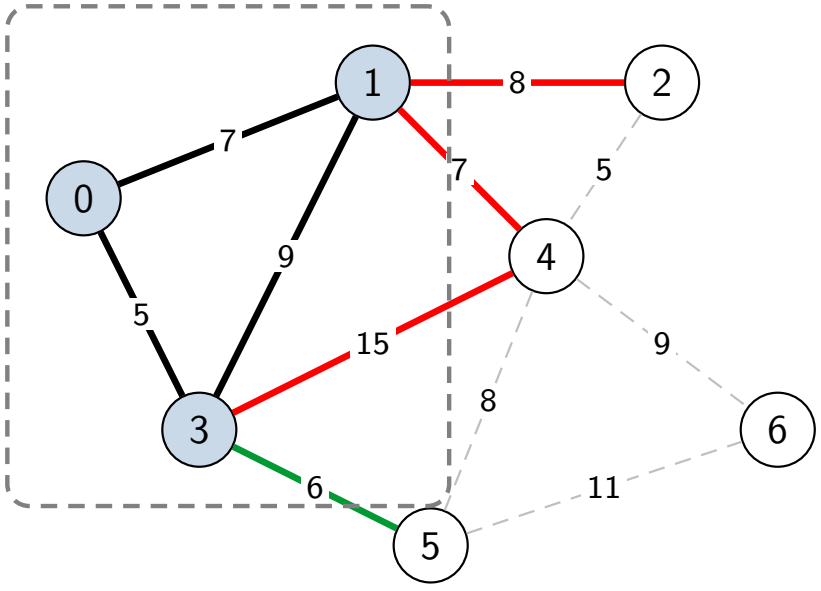
6 Cuál de los dos

7 Ejercicios

8 Lo que sigue

9 Referencias

Corte, cruzar, respetar, ligera



Cuatro palabras

corte	parte V en dos montones; aquí $S = \{0, 1, 3\}$
cruza	tiene una punta en cada lado: las cuatro de color
respeta	el corte respeta a A si ninguna de A lo cruza
ligera	la más barata de las que cruzan: $(3, 5)$, con 6

La propiedad de corte

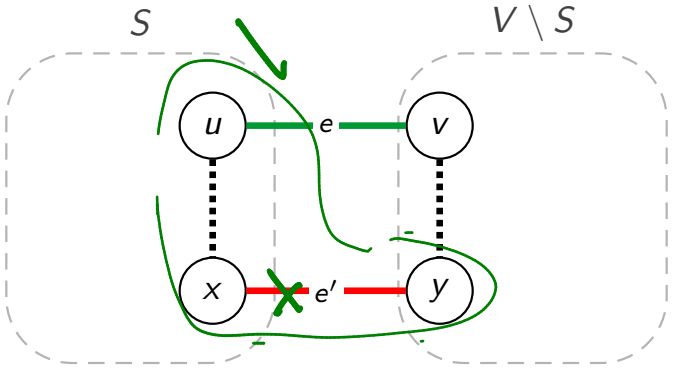
Qué es una arista segura

Una arista $e \notin A$ es segura si $A \cup \{e\}$ todavía cabe dentro de algún árbol de cubrimiento mínimo.

Teorema (CLRS, teorema 23.1, página 626)

Sea A un subconjunto de algún árbol de cubrimiento mínimo T, sea $(S, V \setminus S)$ un corte que respeta a A y sea e una arista ligera de ese corte. Entonces e es segura para A.

$w(T') = w(T) - w(e') + w(e)$



El intercambio

Demostración

Se procede de forma directa. Si $e \in T$ no hay nada que hacer, así que se supone $e = (u, v) \notin T$.
En T hay un único camino de u a v, y agregar e lo cierra en un ciclo. Ese camino arranca en S y termina afuera, de modo que en algún paso salta de lado: sea $e' = (x, y)$ esa arista. Cruza el corte, y como el corte respeta a A, no está en A. Como e es ligera, $w(e) \leq w(e')$.
Sea $T' = T \cup \{e\} \setminus \{e'\}$. Quitar e' del ciclo deja todo conectado, así que T' es árbol de cubrimiento, contiene a A y pesa $w(T) + w(e) - w(e')$.

Conclusión

Ese peso no supera a $w(T)$ y T era mínimo, así que los dos pesan lo mismo y T' también es mínimo. Entonces $A \cup \{e\}$ cabe en un árbol mínimo, que es lo que significa que e sea segura.

Contenido

1 De dónde venimos

2 Conectar al menor costo

3 La propiedad de corte

4 El algoritmo de Kruskal

5 El algoritmo de Prim

6 Cuál de los dos

7 Ejercicios

8 Lo que sigue

9 Referencias

La idea

Las aristas en orden de precio

Se recorren de la más barata a la más cara. Si los extremos están en pedazos distintos, la arista entra; si ya estaban juntos, cerraría un ciclo y se descarta.

Por qué la que entra es ligera de un corte

Sea S el pedazo de u al mirar (u, v). Ninguna arista aceptada cruza ese corte, porque viven dentro de un pedazo. Y si alguna del corte pesara menos, ya se habría mirado y u y v estarían juntos.

El código

```
32 def kruskal(V, aristas):
33     ordenadas = sorted(aristas, key=lambda e: e[2])
34     p = {}
35     rango = {}
36     for v in V:
37         make_set(p, rango, v)
38     mst = []
39     peso = 0
40     i = 0
41     while i < len(ordenadas) and len(mst) < len(V) - 1:
42         u, v, w = ordenadas[i]
43         if find_set(p, u) != find_set(p, v):
44             union(p, rango, u, v)
45             mst.append((u, v, w))
46             peso = peso + w
47         i = i + 1
48     return mst, peso
```

Lo que cuesta

Ordenar es $O(|E| \log |E|)$ y las $|E|$ consultas al conjunto disjunto son casi lineales. El ordenamiento domina.

Antes de la tabla, prediga

Las once aristas en orden

$$(0,3,5),\ (2,4,5),\ (3,5,6),\ (0,1,7),\ (1,4,7),\ (1,2,8),$$
$$(4,5,8),\ (1,3,9),\ (4,6,9),\ (5,6,11),\ (3,4,15)$$

Dos preguntas

Recorriendo esa lista de izquierda a derecha, ¿cuáles entran y cuáles se descartan por cerrar ciclo? ¿En cuál se completa el árbol?

La traza

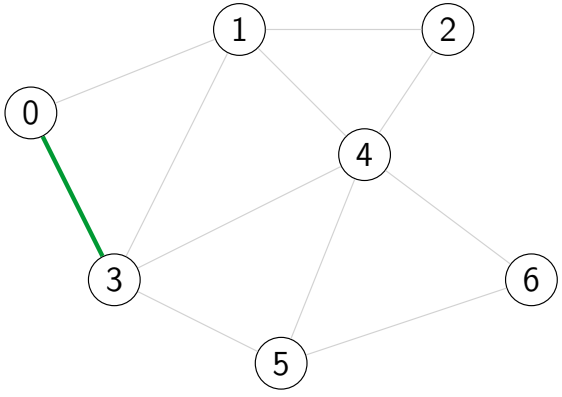
make_set {0} {1} ... {6}

	arista	lo que compara `find_set`	decisión	pedazos después
1	$(0, 3, 5)$	$\{0\} \neq \{3\}$	entra	$\{0, 3\}, \{1\}, \{2\}, \{4\}, \{5\}, \{6\}$
2	$(2, 4, 5)$	$\{2\} \neq \{4\}$	entra	$\{0, 3\}, \{1\}, \{2, 4\}, \{5\}, \{6\}$
3	$(3, 5, 6)$	$\{0, 3\} \neq \{5\}$	entra	$\{0, 3, 5\}, \{1\}, \{2, 4\}, \{6\}$
4	$(0, 1, 7)$	$\{0, 3, 5\} \neq \{1\}$	entra	$\{0, 1, 3, 5\}, \{2, 4\}, \{6\}$
5	$(1, 4, 7)$	$\{0, 1, 3, 5\} \neq \{2, 4\}$	entra	$\{0, 1, 2, 3, 4, 5\}, \{6\}$
6	$(1, 2, 8)$	el mismo pedazo	descartar	sin cambio
7	$(4, 5, 8)$	el mismo pedazo	descartar	sin cambio
8	$(1, 3, 9)$	el mismo pedazo	descartar	sin cambio
9	$(4, 6, 9)$	$\{0, \ldots, 5\} \neq \{6\}$	entra	$\{0, 1, 2, 3, 4, 5, 6\}$

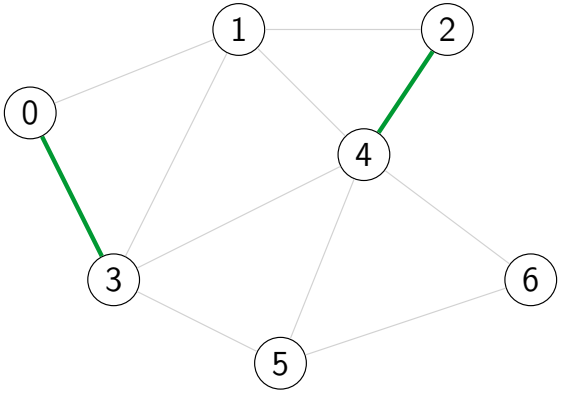
Dónde para

En la novena ya hay seis aristas, que son $|V| - 1$, y el ciclo no vuelve a entrar: las dos que faltaban ni se miran. El peso es 39, el único de ese peso entre los 141 árboles de cubrimiento del grafo.

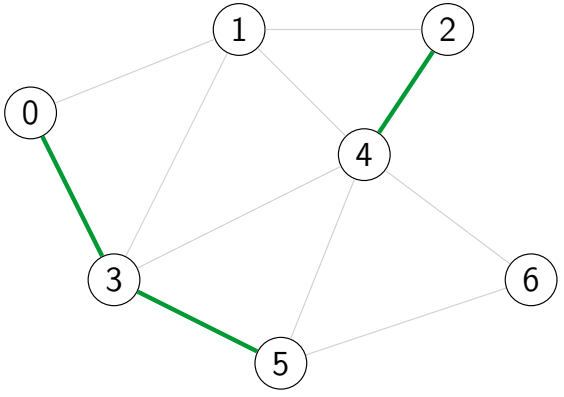
Kruskal, dibujo por dibujo



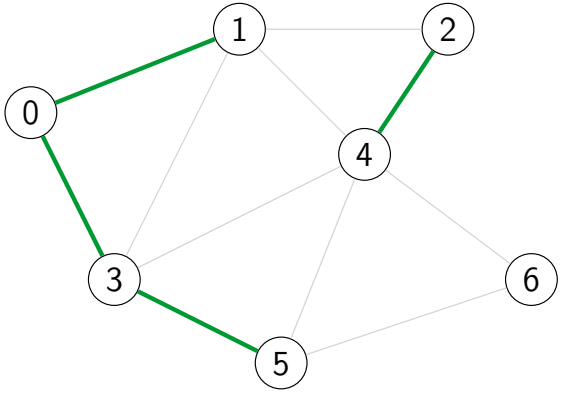
1. $(0, 3)$



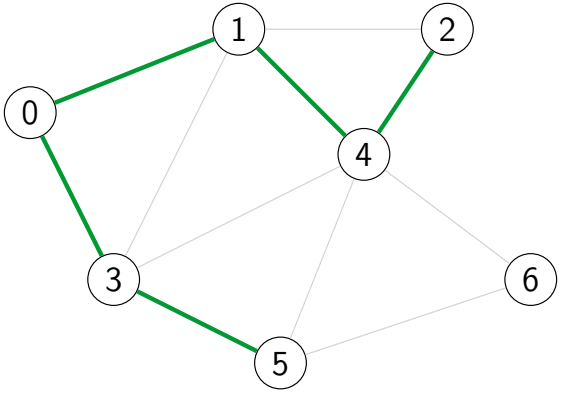
2. $(2, 4)$



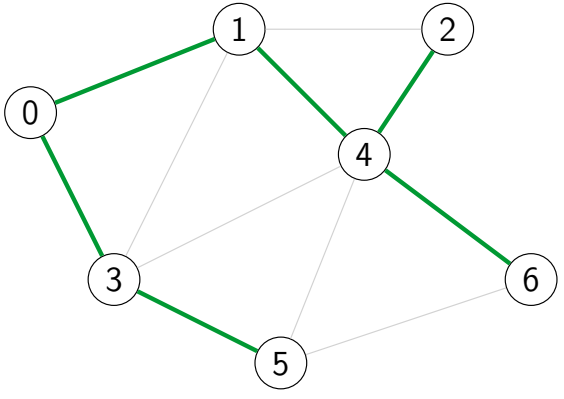
3. $(3, 5)$



4. $(0, 1)$

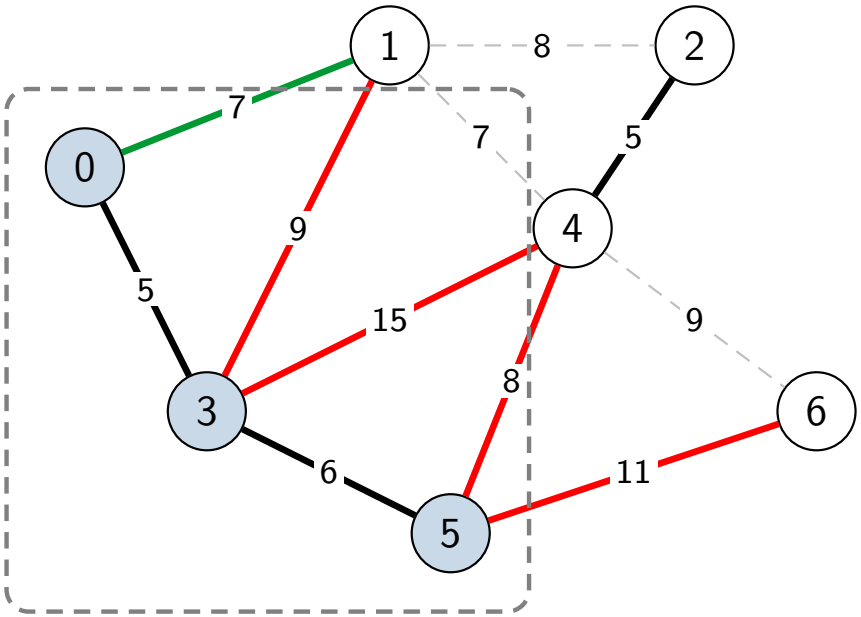


5. $(1, 4)$



6. $(4, 6)$

El corte detrás del cuarto paso



Qué cruza

Al llegar a $(0, 1)$ el pedazo de 0 es $\{0, 3, 5\}$. Cruzan el corte cinco aristas: $(0, 1)$ con 7, $(4, 5)$ con 8, $(1, 3)$ con 9, $(5, 6)$ con 11 y $(3, 4)$ con 15. La más barata es $(0, 1)$, y no podía ser otra: una arista del corte más barata que 7 ya se habría mirado y 0 y 1 estarían juntos. Las tres aristas negras no cruzan, así que el corte las respeta.

Contenido

1 De dónde venimos

2 Conectar al menor costo

3 La propiedad de corte

4 El algoritmo de Kruskal

5 El algoritmo de Prim

6 Cuál de los dos

7 Ejercicios

8 Lo que sigue

9 Referencias

La idea

Un solo árbol que crece

Se arranca de una raíz y se mantiene V_T, lo que ya entró. En cada paso se agrega la arista más barata que va de V_T a algún vértice de afuera. El corte no hay que buscarlo: es siempre $(V_T, V \setminus V_T)$, el borde del árbol.

Qué se guarda

Para cada vértice de afuera, $clave[v]$ es el peso de la arista más barata que lo conecta con el árbol y $padre[v]$ es de dónde sale. Al entrar u, sus vecinos revisan si les mejora la clave.

Sacar el más barato de los de afuera

```
12 def extraer_minimo(clave, en_mst):
13     minimo = None ; mejor = INFTY
14     for v in clave:
15         if not en_mst[v] and clave[v] < mejor:
16             mejor = clave[v] ; minimo = v
17     return minimo
```

Lo que cuesta

Recorre todos los vértices y devuelve el de menor clave entre los que siguen afuera. Son $|V|$ pasos y se llama $|V|$ veces. `INFTY` es `float('inf')`, que es mayor que cualquier peso y así la primera comparación siempre mejora.

El código con búsqueda lineal

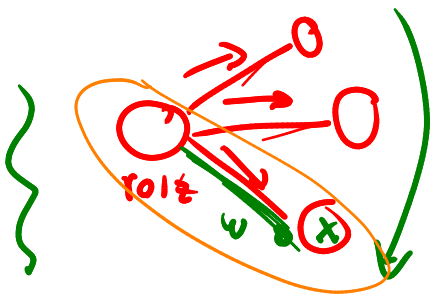
```
20 def prim(grafo, raiz):
21     clave = {} ; padre = {} ; en_mst = {}
22     for v in grafo:
23         clave[v] = INFTY ; padre[v] = None ; en_mst[v] = False
24     clave[raiz] = 0
25     peso = 0
26     restantes = len(grafo)
27     while restantes > 0:
28         u = extraer_minimo(clave, en_mst)
29         if u is None:
30             restantes = 0
31         else:
32             en_mst[u] = True
33             if padre[u] is not None:
34                 peso = peso + clave[u]
35             for (v, w) in grafo[u]:
36                 if not en_mst[v] and w < clave[v]:
37                     clave[v] = w ; padre[v] = u
38             restantes = restantes - 1
39     return padre, peso
```



Antes de la traza, prediga

Prim desde la raíz 0

¿En qué orden entran los siete vértices al árbol?
¿Cuál de ellos cambia de padre más de una vez antes de que le llegue el turno?



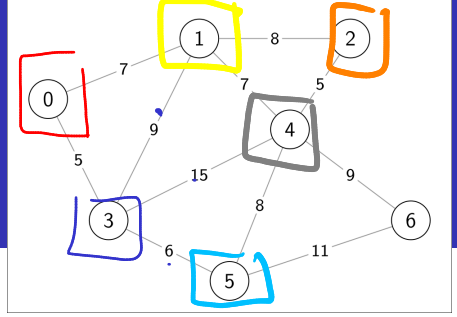
La traza desde la raíz 0

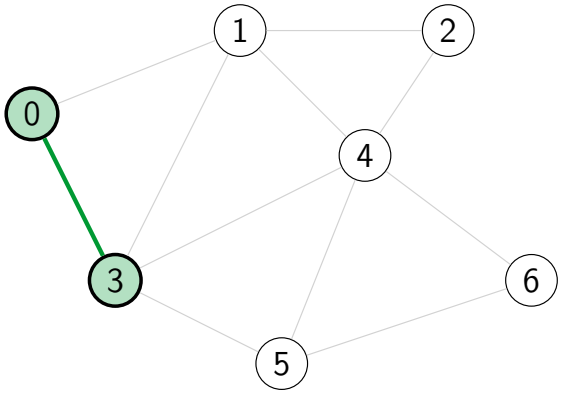
iter	0	1	2	3	4	5	6	extrae	arista
0	**0**	∞	∞	∞	∞	∞	∞	0	
1		7	∞	**5**	∞	∞	∞	3	$(0,3,5)$
2		**7**	∞		15	6	∞	5	$(3,5,6)$
3		**7**	∞		8		11	1	$(0,1,7)$
4			8		**7**		11	4	$(1,4,7)$
5			**5**				9	2	$(4,2,5)$
6							**9**	6	$(4,6,9)$

Cómo se lee

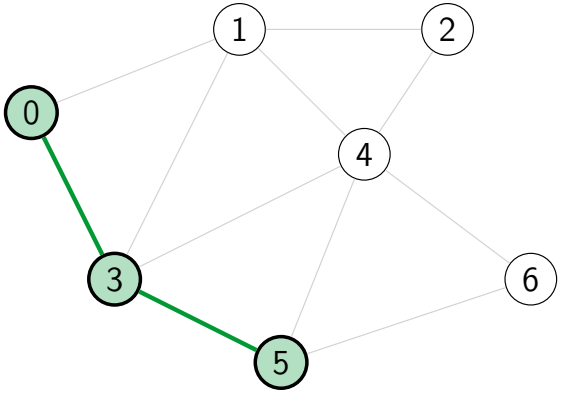
Cada fila trae las claves de los que están afuera, con el mínimo en negrita y vacía la casilla del que ya entró. La clave de 4 baja de 15 a 8 y después a 7, y con ella su padre pasa de 3 a 5 y a 1. Las seis aristas suman 39, las mismas de Kruskal.

Prim, dibujo por dibujo

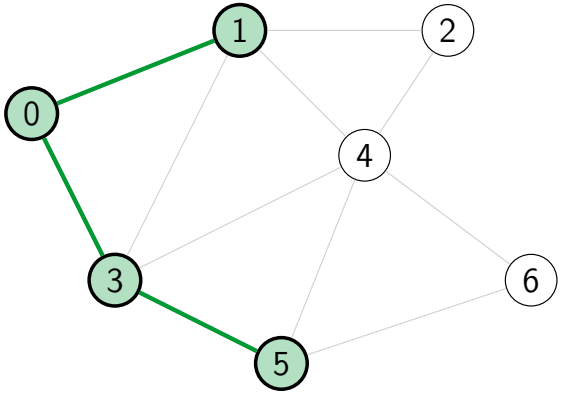




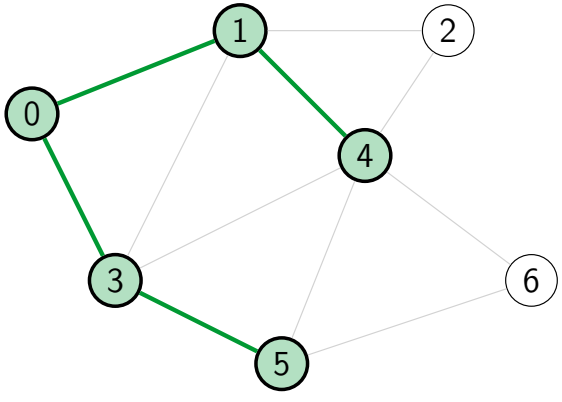
1. entra 3



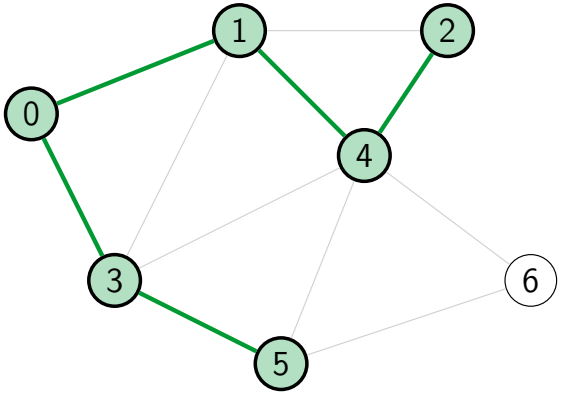
2. entra 5



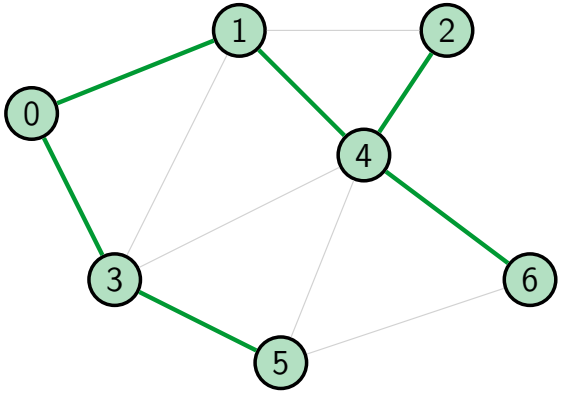
3. entra 1



4. entra 4

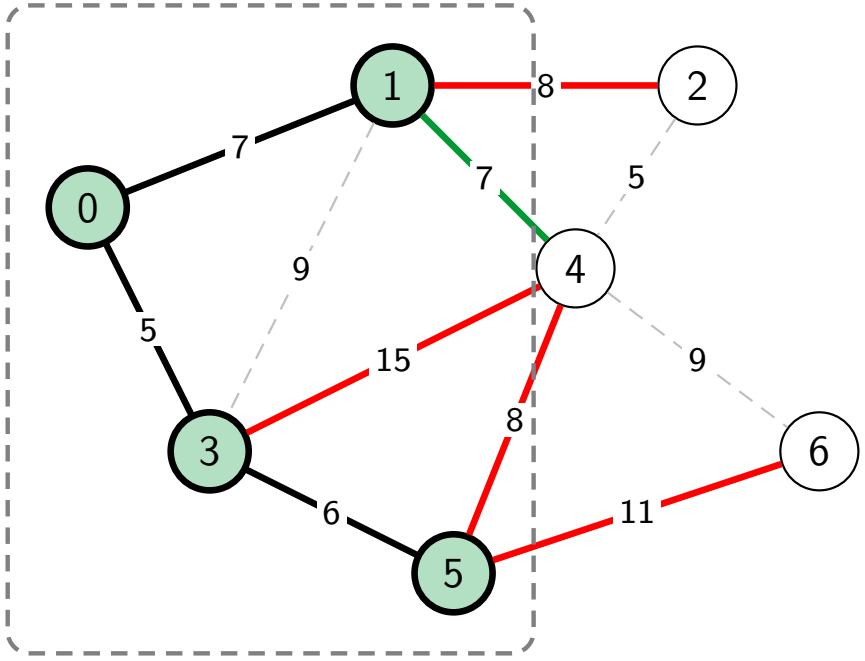


5. entra 2



6. entra 6

El corte de la cuarta iteración



Qué cruza

Con $V_T = \{0, 1, 3, 5\}$ cruzan $(1, 4)$ con 7, $(1, 2)$ con 8, $(4, 5)$ con 8, $(5, 6)$ con 11 y $(3, 4)$ con 15. Entra $(1, 4)$. La diferencia con Kruskal es de punto de vista: allá el corte aparece alrededor del pedazo que se acaba de formar, aquí está fijo en el borde del árbol.

La misma respuesta desde otra raíz

iter	0	1	2	3	4	5	6	extrae	arista
0	∞	∞	∞	∞	∞	∞	0	6	
1	∞	∞	∞	∞	9	11		4	$(6, 4, 9)$
2	∞	7	5	15		8		2	$(4, 2, 5)$
3	∞	7		15		8		1	$(4, 1, 7)$
4	7			9		8		0	$(1, 0, 7)$
5				5		8		3	$(0, 3, 5)$
6						6		5	$(3, 5, 6)$

Qué cambió y qué no

Las seis aristas son las mismas y suman 39, pero el orden en que entran se dio vuelta: la de 9 abre y la de 5 no entra hasta la quinta. Prim no toma las aristas de menor a mayor, toma la más barata del corte que tiene en ese momento. Kruskal sí va en orden y no tiene raíz, y entre pesos iguales el orden es arbitrario: los dieciséis de este grafo dan el mismo árbol.

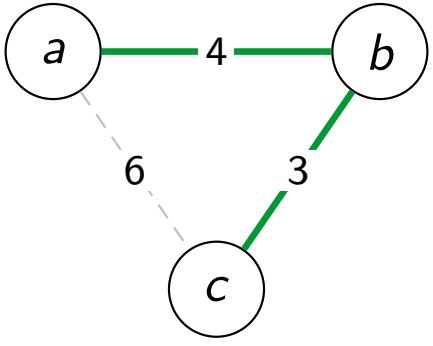
Prim con cola de prioridad

```
42 def prim_cola(grafo, raiz):
43     clave = {} ; padre = {} ; en_mst = {}
44     for v in grafo:
45         clave[v] = INFTY ; padre[v] = None ; en_mst[v] = False
46     clave[raiz] = 0
47     cola = list() ; heappush(cola, (clave[raiz], raiz))
48     peso = 0
49     while len(cola) != 0:
50         cu, u = heappop(cola)
51         # si u ya entro al arbol, la pareja es vieja y se descarta
52         if not en_mst[u]:
53             en_mst[u] = True
54             peso = peso + cu
55             for (v, w) in grafo[u]:
56                 if not en_mst[v] and w < clave[v]:
57                     clave[v] = w ; padre[v] = u
58                     heappush(cola, (clave[v], v))
59     return padre, peso
```

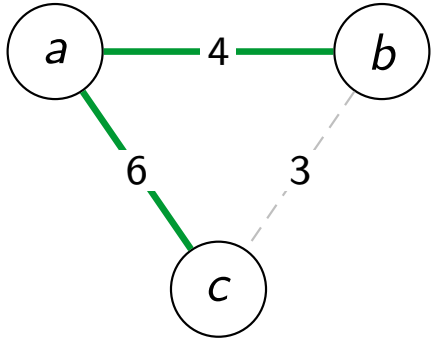
Una línea de diferencia con Dijkstra

Allá, $d_u + w(u, v) < dist[v]$. Aquí, $w < clave[v]$.

El árbol de caminos más cortos es otro



Prim desde a: peso 7



Dijkstra desde a: peso 10

Dónde se separan

Al mirar c, Prim compara el 3 que cuesta engancharlo desde b. Dijkstra compara $4 + 3$ contra el 6 de la arista directa y se queda con el 6. En el árbol mínimo, ir de a a c cuesta 7.

Contenido

1 De dónde venimos

2 Conectar al menor costo

3 La propiedad de corte

4 El algoritmo de Kruskal

5 El algoritmo de Prim

6 Cuál de los dos

7 Ejercicios

8 Lo que sigue

9 Referencias

Kruskal y Prim, lado a lado

	Kruskal	Prim
qué recorre	las aristas, ordenadas	el borde del árbol
estructura	conjuntos disjuntos	cola de prioridad
cómo llega el grafo	lista de aristas	listas de adyacencia
costo	$O(\|E\| \log \|E\|)$	$O(\|E\| \log \|V\|)$ con cola $O(\|V\|^2)$ con búsqueda lineal
le conviene	grafos ralos	grafos densos

Cuándo gana cada uno

En un grafo denso, con $|E|$ cerca de $|V|^2$, la búsqueda lineal de Prim cuesta $|V|^2$ sin logaritmo ni estructura de por medio, y ordenar $|V|^2$ aristas sale más caro. En un grafo ralo pasa lo contrario, y además Kruskal no necesita tener el grafo armado: le basta la lista de aristas tal como viene de la entrada.

Cuál se usa en la práctica

Kruskal, donde las aristas ya son el dato

No necesita el grafo armado: le basta la lista de aristas, y si llega ordenada se ahorra el Ig. Por eso corre dentro del agrupamiento por enlace simple y de la segmentación de imágenes, donde el dato son las diferencias entre pares y lo que se busca son los pedazos que quedan al dejar de unir.

Prim, donde el grafo es denso o no está escrito

Nunca mira una arista con los dos extremos afuera, así que con búsqueda lineal cuesta $O(V^2)$ sin guardar ni una arista. Eso sirve cuando el peso se calcula al momento, como la distancia entre n puntos del plano: ordenar las $n(n-1)/2$ aristas saldría más caro. Y el árbol crece desde un punto, que es el orden en que un tendido se instala.

Cuatro formas de equivocarse

Creer que el árbol es único

Con pesos repetidos puede haber varios, todos del mismo peso total, que es lo que suele pedir un juez.

No ver que el grafo no es conexo

Los dos terminan con menos de $|V| - 1$ aristas. Eso se reporta, en vez de imprimir la suma de los pedazos.

Pensar que los pesos negativos lo rompen

La propiedad de corte compara pesos y nunca los suma, así que los dos algoritmos siguen sirviendo. Es Dijkstra el que no los soporta.

Dejar la suma en un entero de 32 bits

Con doscientas mil aristas el total se desborda sin avisar.

Contenido

1 De dónde venimos

2 Conectar al menor costo

3 La propiedad de corte

4 El algoritmo de Kruskal

5 El algoritmo de Prim

6 Cuál de los dos

7 Ejercicios

8 Lo que sigue

9 Referencias

Ejercicio 1: en parejas

Sobre el grafo de la sesión

1. Baje el peso de $(3, 4)$ de 15 a 4 y vuelva a correr Kruskal. ¿Cambia el árbol? ¿Cuál arista sale?
2. Suba el peso de $(2, 4)$ de 5 a 7, con lo que empata con $(0, 1)$ y $(1, 4)$, y muestre dos árboles del mismo peso.
3. Corra Prim desde la raíz 6 y compare las seis aristas con las de la traza desde 0.

Después intercambien con la pareja de al lado y verifiquen con el algoritmo que no usaron.

Cinco ejercicios más, en el navegador

`https://cardel.github.io/notasUniversidad/actividades/ayg-puj/arboles-cubrimiento-minimo/`

Ejercicio 2: apagar faroles

UVa 11631, Dark roads

Alumbrar un metro de calle cuesta un dólar por noche. El gobierno quiere apagar calles para gastar menos, con la condición de que entre dos cruces cualesquiera siga quedando un camino iluminado. Se pide cuánto es lo máximo que puede ahorrar por noche.

El caso del enunciado es el grafo de hoy

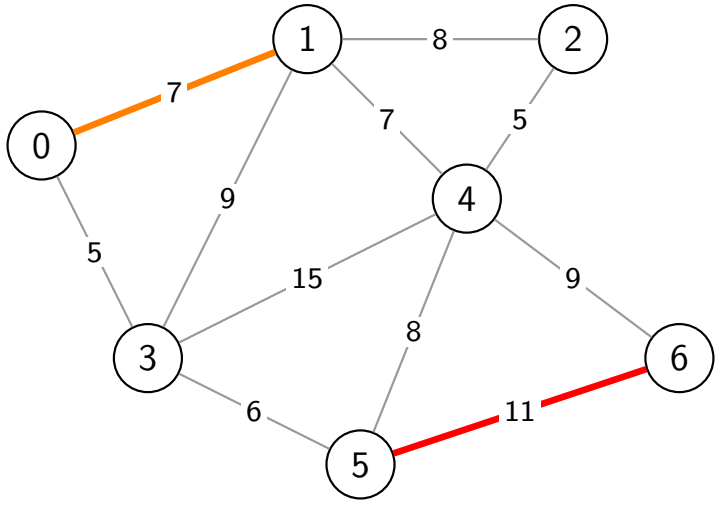
Las once calles suman 90 y el árbol de cubrimiento mínimo pesa 39, así que la respuesta es $90 - 39 = 51$. Lo que se ahorra es todo lo que no entra al árbol.

Dónde está

Enunciado y envío: `https://vjudge.net/problem/UVA-11631`

La entrada, línea por línea

```
7 11
0 1 7
0 3 5
1 2 8
1 3 9
1 4 7
2 4 5
3 4 15
3 5 6
4 5 8
4 6 9
5 6 11
0 0
```



Qué dice cada línea

La primera, cuántos cruces y cuántas calles: siete y once. Las once que siguen son una calle cada una, con sus dos cruces y su longitud, de modo que `0 1 7` es la naranja y `5 6 11` la roja. Los cruces van numerados desde 0. La línea `0 0` cierra la entrada, y antes puede venir otro caso completo. La salida es una línea por caso: aquí, 51.

Cómo leer la entrada

```
int m, n;
scanf("%d %d", &m, &n);
while (m != 0) {
  vector<pair<int, pair<int, int> > > aristas(n);
  long long total = 0;
  for (int i = 0; i < n; i++) {
    int x, y, z;
    scanf("%d %d %d", &x, &y, &z);
    aristas[i] = make_pair(z, make_pair(x, y));
    total = total + z;
  }
  /* aqui se corre Kruskal y se imprime total - peso del MST */
  scanf("%d %d", &m, &n);
}
```

Tres detalles

Cada caso empieza con el número de cruces y el de calles, y la entrada termina cuando los dos llegan en cero. Los cruces vienen numerados desde 0. Con 200 000 calles y pesos de hasta 2^{31} la suma de todas no cabe en un `int`.

Contenido

1 De dónde venimos

2 Conectar al menor costo

3 La propiedad de corte

4 El algoritmo de Kruskal

5 El algoritmo de Prim

6 Cuál de los dos

7 Ejercicios

8 Lo que sigue

9 Referencias

Lo que sigue

Preguntarle a un rango

Lo que viene usa un árbol para responder preguntas sobre pedazos de un arreglo: la suma de las posiciones de i a j, o el mínimo, con el arreglo cambiando entre pregunta y pregunta.

Para investigar

Con el arreglo quieto, las sumas acumuladas contestan cualquier rango en una resta. ¿Qué pasa con ellas cuando una posición cambia? ¿Y si en lugar de guardar la suma de cada prefijo se guarda la de cada mitad, y de cada mitad de la mitad?

Contenido

1 De dónde venimos

2 Conectar al menor costo

3 La propiedad de corte

4 El algoritmo de Kruskal

5 El algoritmo de Prim

6 Cuál de los dos

7 Ejercicios

8 Lo que sigue

9 Referencias

Referencias


Cormen, T. H., Leiserson, C. E., Rivest, R. L., Stein, C. *Introduction to Algorithms*, 3rd ed. MIT Press, 2009. Capítulo 23, páginas 624 a 642.


Borůvka, O. *O jistém problému minimálním*. Práce Moravské přírodovědecké společnosti, 3, 1926, páginas 37 a 58.


Kruskal, J. B. *On the shortest spanning subtree of a graph and the traveling salesman problem*. Proceedings of the American Mathematical Society, 7(1), 1956, páginas 48 a 50.


Prim, R. C. *Shortest connection networks and some generalizations*. Bell System Technical Journal, 36(6), 1957, páginas 1389 a 1401.


Halim, S., Halim, F., Effendy, S. *Competitive Programming 4*. Lulu, 2020. Capítulo 4: árboles de cubrimiento mínimo.