Grafos II

Pablo Blanc

Buen Kilo de Pan Flauta
(con diapos bastante robadas a Leopoldo Taravilse)

Training Camp 2017

Contenidos

Cosas que vimos ayer

¿Cómo representamos un grafo por ahora?

- Lista de adyacencia
- Matriz de adyacencia

Lista de incidencia

Definición

La lista de incidencia de un grafo es una lista de ejes de un grafo, en donde cada eje se representa por sus nodos incidentes como pares (ordenados si el grafo es dirigido), más el costo del eje si el grafo es ponderado.

Lista de incidencia

Definición

La lista de incidencia de un grafo es una lista de ejes de un grafo, en donde cada eje se representa por sus nodos incidentes como pares (ordenados si el grafo es dirigido), más el costo del eje si el grafo es ponderado.

Ventajas de la lista de incidencia

Cuando sólo tenemos que iterar sobre los ejes sin importar si lo hacemos en un orden particular según qué ejes inciden en qué nodos, suele ser más eficiente que revisar listas de adyacencias con muchos nodos sin ejes que son iteraciones que consumen tiempo y no hacen nada.

Cosas que vimos ayer

- BFS
- DFS
- Dijkstra

Los algoritmos vistos no calculan la distancia mínima cuando hay aristas negativas.

Cosas que vimos ayer

- BFS
- DFS
- Dijkstra

Los algoritmos vistos no calculan la distancia mínima cuando hay aristas negativas.

Grafos dirigidos con ciclos negativos

- En un grafo no dirigido conexo, si hay un ciclo negativo podemos movernos por ese ciclo y llegar a cualquier nodo con un costo arbitrariamente bajo.

Grafos dirigidos con ciclos negativos

- En un grafo no dirigido conexo, si hay un ciclo negativo podemos movernos por ese ciclo y llegar a cualquier nodo con un costo arbitrariamente bajo.
- Es por eso que tiene sentido ver si en un grafo hay ciclos negativos cuando el grafo es dirigido.

Grafos dirigidos con ciclos negativos

- En un grafo no dirigido conexo, si hay un ciclo negativo podemos movernos por ese ciclo y llegar a cualquier nodo con un costo arbitrariamente bajo.
- Es por eso que tiene sentido ver si en un grafo hay ciclos negativos cuando el grafo es dirigido.
- Para esto vamos a usar el algoritmo de Bellman Ford.

Contenidos

Bellman-Ford

- El algoritmo de Bellman Ford itera hasta $n - 1$ veces sobre cada eje. Si moverse por ese eje disminuye la distancia desde el nodo inicial hasta el nodo hacia el cuál nos movemos por ese eje, entonces actualiza la distancia a la que obtenemos moviendonos por ese eje.

Bellman-Ford

- El algoritmo de Bellman Ford itera hasta $n-1$ veces sobre cada eje. Si moverse por ese eje disminuye la distancia desde el nodo inicial hasta el nodo hacia el cuál nos movemos por ese eje, entonces actualiza la distancia a la que obtenemos moviendonos por ese eje.
- Esto funciona porque un camino mínimo tiene como máximo $n-1$ ejes, entonces el i-ésimo paso actualizamos la distancia al $(i+1)$-ésimo nodo del camino mínimo.

Bellman-Ford

- El algoritmo de Bellman Ford itera hasta $n-1$ veces sobre cada eje. Si moverse por ese eje disminuye la distancia desde el nodo inicial hasta el nodo hacia el cuál nos movemos por ese eje, entonces actualiza la distancia a la que obtenemos moviendonos por ese eje.
- Esto funciona porque un camino mínimo tiene como máximo $n-1$ ejes, entonces el i-ésimo paso actualizamos la distancia al $(i+1)$-ésimo nodo del camino mínimo.
- Si luego de los $n-1$ pasos podemos seguir disminuyendo la distancia a un nodo quiere decir que hay un camino de longitud al menos n que asigna una menor distancia que todos los caminos más chicos, pero este camino contiene un ciclo, luego hay un ciclo negativo.

Bellman-Ford

- Si hay un ciclo negativo entonces el camino mínimo a los nodos a los que se puede llegar desde un nodo del ciclo no existe porque hay caminos arbitrariamente chicos.

Bellman-Ford

- Si hay un ciclo negativo entonces el camino mínimo a los nodos a los que se puede llegar desde un nodo del ciclo no existe porque hay caminos arbitrariamente chicos.
- Vamos a ver una versión del algoritmo que sólo nos dice si hay o no ciclos negativos

Bellman-Ford

- Si hay un ciclo negativo entonces el camino mínimo a los nodos a los que se puede llegar desde un nodo del ciclo no existe porque hay caminos arbitrariamente chicos.
- Vamos a ver una versión del algoritmo que sólo nos dice si hay o no ciclos negativos
- Cuando guardamos los padres de cada nodo, lo hacemos para poder luego modificar el algoritmo para que nos diga para qué nodos está definida la distancia desde el nodo inicial y para esos nodos podemos obtener la distancia.

Implementación de Bellman-Ford

```
1  struct arista{
2      int f,s;
3      double p;
4  };
5  int padresBF[1000]; int distBF[1000];
6  bool bFord(vector<arista> lista, int n, int source){
7      int m = lista.size();
8      for(int i=0;i<n;i++){
9          distBF[i] = INF; padresBF[i] = i;
10     distBF[source] = 0;
11     for(int i=0;i<n-1;i++)for(int j=0;j<m;j++)
12     if(distBF[lista[j].s] > distBF[lista[j].f]+ lista[j].p){
13         distBF[lista[j].s] = distBF[lista[j].f]+ lista[j].p;
14         padresBF[lista[j].s] = lista[j].f;
15     for(int j=0;j<m;j++)
16     if(distBF[lista[j].s] > distBF[lista[j].f]+ lista[j].p)
17         return false;
18     return true;
19 }
```

Contenidos

Algoritmo de Floyd-Warshall

- Hasta ahora vimos cómo calcular el camino mínimo desde un nodo hasta todos los demás.

Algoritmo de Floyd-Warshall

- Hasta ahora vimos cómo calcular el camino mínimo desde un nodo hasta todos los demás.
- ¿Qué pasa si queremos calcular la distancia de todos los nodos a todos los nodos?

Algoritmo de Floyd-Warshall

- Hasta ahora vimos cómo calcular el camino mínimo desde un nodo hasta todos los demás.
- ¿Qué pasa si queremos calcular la distancia de todos los nodos a todos los nodos?
- Si el grafo es no ponderado, podemos hacer un BFS para cada nodo, esto funciona tanto con grafos dirigidos como con grafos no dirigidos, si en cambio el grafo es ponderado, el algoritmo de Bellman Ford funciona en ambos casos pero es muy costoso, para esto existe el algoritmo de Floyd Warshal, que incluso funciona con ejes de peso negativo.

Algoritmo de Floyd-Warshall

- El algoritmo de Floyd-Warshall (también conocido como algoritmo de Floyd) va compactando aristas. En el i-ésimo paso compacta las que unen los vértices v con el i-ésimo y el i-ésimo con w generando, si la distancia es más chica que la que hay hasta el momento, una arista entre v y w.

Algoritmo de Floyd-Warshall

- El algoritmo de Floyd-Warshall (también conocido como algoritmo de Floyd) va compactando aristas. En el i-ésimo paso compacta las que unen los vértices v con el i-ésimo y el i-ésimo con w generando, si la distancia es más chica que la que hay hasta el momento, una arista entre v y w.
- Su complejidad es $O(n^3)$ y se implementa con una matriz de adyacencia. Su implementación es mucho más sencila que la de los algoritmos que vimos anteriormente.

Algoritmo de Floyd-Warshall

- El algoritmo de Floyd-Warshall (también conocido como algoritmo de Floyd) va compactando aristas. En el i-ésimo paso compacta las que unen los vértices v con el i-ésimo y el i-ésimo con w generando, si la distancia es más chica que la que hay hasta el momento, una arista entre v y w.
- Su complejidad es $O(n^3)$ y se implementa con una matriz de adyacencia. Su implementación es mucho más sencila que la de los algoritmos que vimos anteriormente.
- Los nodos i tales que el camino mínimo de i a i tienen peso negativo son los que participan de un ciclo negativo.

Implementación del algoritmo de Floyd

```
1 void floyd(vector<vector<int> > &matriz){
2     for(int k=0;k<n;k++)
3     for(int i=0;i<n;i++)
4     for(int j=0;j<n;j++)
5         matriz[i][j] = min(matriz[i][j],matriz[i][k]+matriz[k][j]);
6 }
```

Implementación del algoritmo de Floyd

```
1 void floyd(vector<vector<int> > &matriz){
2     for(int k=0;k<n;k++)
3     for(int i=0;i<n;i++)
4     for(int j=0;j<n;j++)
5         matriz[i][j] = min(matriz[i][j],matriz[i][k]+matriz[k][j]);
6 }
```

- La matriz empieza inicializada con las distancias dadas por los ejes en donde hay ejes y con infinito en todas las demás posiciones, además, la diagonal principal empieza inicializada en cero.

Contenidos

Árbol generador mínimo

Definición

Sea $G = (V, E)$ un grafo ponderado conexo. Un árbol generador mínimo de G es un árbol $G' = (V, E')$ tal que la suma de los costos de las aristas de G' es mínima.

Árbol generador mínimo

Definición

Sea $G = (V, E)$ un grafo ponderado conexo. Un árbol generador mínimo de G es un árbol $G' = (V, E')$ tal que la suma de los costos de las aristas de G' es mínima.

- Existen dos algoritmos conocidos que resuelven este problema. Uno de ello es el algoritmo de Kruskal, y el otro el algoritmo de Prim.

Árbol generador mínimo

Definición

Sea $G = (V, E)$ un grafo ponderado conexo. Un árbol generador mínimo de G es un árbol $G' = (V, E')$ tal que la suma de los costos de las aristas de G' es mínima.

- Existen dos algoritmos conocidos que resuelven este problema. Uno de ello es el algoritmo de Kruskal, y el otro el algoritmo de Prim.
- Vamos a ver el algoritmo de Kruskal en detalle y contaremos cómo funciona el algoritmo de Prim sin entrar en detalles ni dar el código.

Union-Find

- Antes de ver cómo funciona y saber qué hace el algoritmo de Kruskal hay que conocer una estructura de datos llamada Union-Find.

Union-Find

- Antes de ver cómo funciona y saber qué hace el algoritmo de Kruskal hay que conocer una estructura de datos llamada Union-Find.
- Esta estructura se utiliza para trabajar con componentes conexas, comienza con todos los nodos del grafo como componentes conexas separadas y en cada paso de unión junta dos componentes en una sóla componente. Las consultas consisten en dar dos nodos y preguntar si están en una misma componente.

Union-Find

- Antes de ver cómo funciona y saber qué hace el algoritmo de Kruskal hay que conocer una estructura de datos llamada Union-Find.
- Esta estructura se utiliza para trabajar con componentes conexas, comienza con todos los nodos del grafo como componentes conexas separadas y en cada paso de unión junta dos componentes en una sóla componente. Las consultas consisten en dar dos nodos y preguntar si están en una misma componente.
- Una implementación eficiente como la que daremos a continuación tiene como complejidad, y no lo vamos a dar ni a demostrar porque es muy difícil, una función que crece muy lento, y es casi lineal.

Implementación de Union-Find

```
1  int padres[1000];
2  int prof[1000];
3  void initUF(int n){
4      for(int i=0;i<n;i++){ padres[i] = i; prof[i] = 0;}
5  }
6  int find(int x){
7      if(padres[x] == x) return x;
8      padres[x] = find(padres[x]);
9      return padres[x];
10 }
11 void join(int x, int y){
12     int xRaiz = find(x), yRaiz = find(y);
13     if(prof[xRaiz]<prof[yRaiz]) padres[xRaiz] = yRaiz;
14     else if(prof[xRaiz]>prof[yRaiz]) padres[yRaiz] = xRaiz;
15     else{
16         padres[yRaiz] = xRaiz;
17         prof[xRaiz]++;
18     }
19 }
```

Implementación de Union-Find

```
1  int padres[1000];
2  int sz[1000];
3  void initUF(int n){
4      for(int i=0;i<n;i++){ padres[i] = i; sz[i] = 1;}
5  }
6  int find(int x){
7      if(padres[x] == x) return x;
8      padres[x] = find(padres[x]);
9      return padres[x];
10 }
11 void join(int x, int y){
12     int xRaiz = find(x), yRaiz = find(y);
13     if(sz[xRaiz]<sz[yRaiz]){
14         padres[xRaiz] = yRaiz;
15         sz[xRaiz]+= sz[yRaiz];
16     }else{
17         padres[yRaiz] = xRaiz;
18         sz[yRaiz]+= sz[xRaiz];
19       }
20 }
```

Algoritmo de Kruskal

- El algoritmo de Kruskal ordena las aristas por peso y va agregando desde la arista de menor peso hasta la de mayor peso, evitando agregar las aristas que forman ciclos.

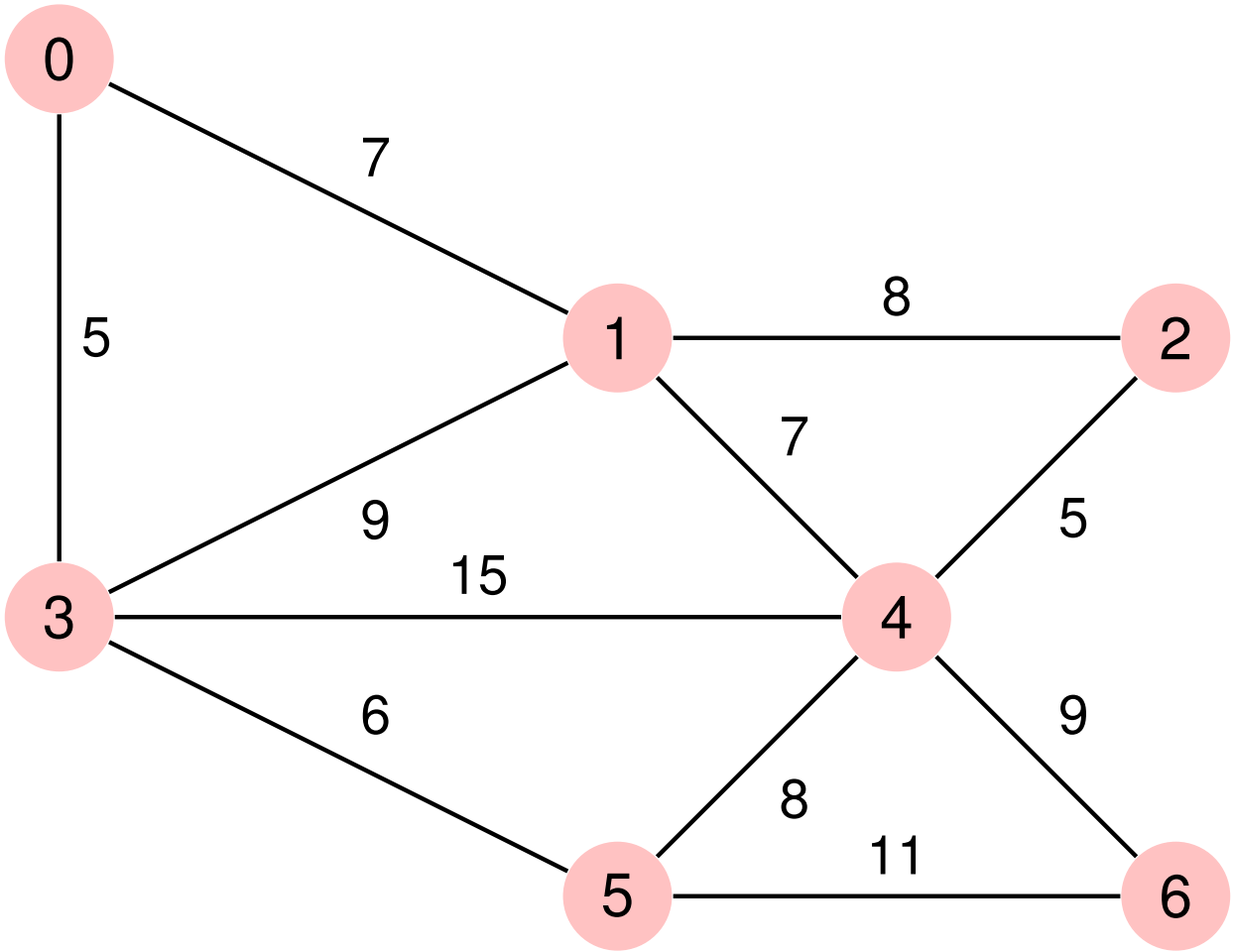
Algoritmo de Kruskal

- El algoritmo de Kruskal ordena las aristas por peso y va agregando desde la arista de menor peso hasta la de mayor peso, evitando agregar las aristas que forman ciclos.
- El árbol generador mínimo no es único y Kruskal puede encontrar todos los AGM según como se ordenen las aristas de igual peso.

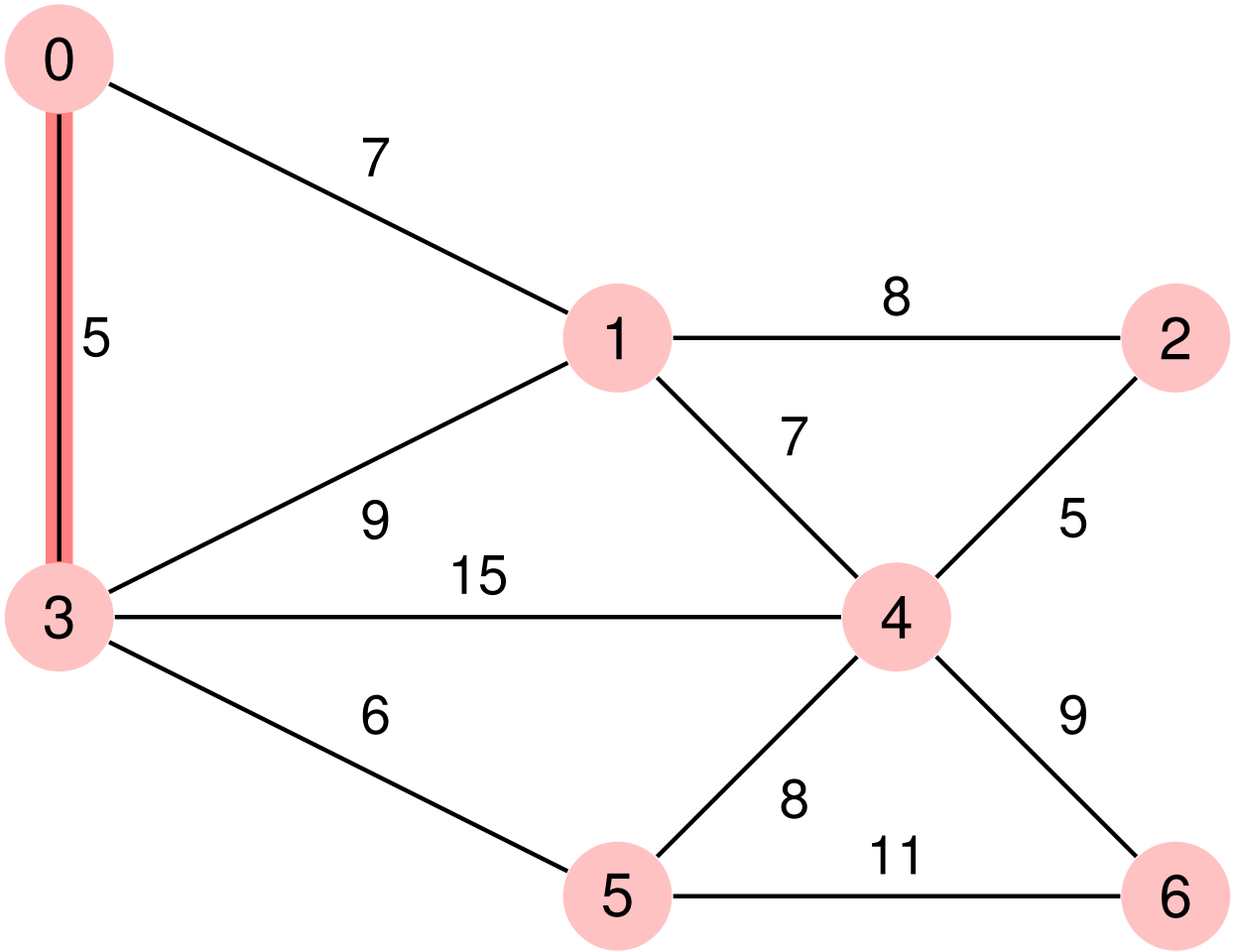
Algoritmo de Kruskal

- El algoritmo de Kruskal ordena las aristas por peso y va agregando desde la arista de menor peso hasta la de mayor peso, evitando agregar las aristas que forman ciclos.
- El árbol generador mínimo no es único y Kruskal puede encontrar todos los AGM según como se ordenen las aristas de igual peso.
- Ahora es fácil ver que la definición de AGM es equivalente a decir que la arista de mayor peso entre todos los árboles generadores tenga peso mínimo.

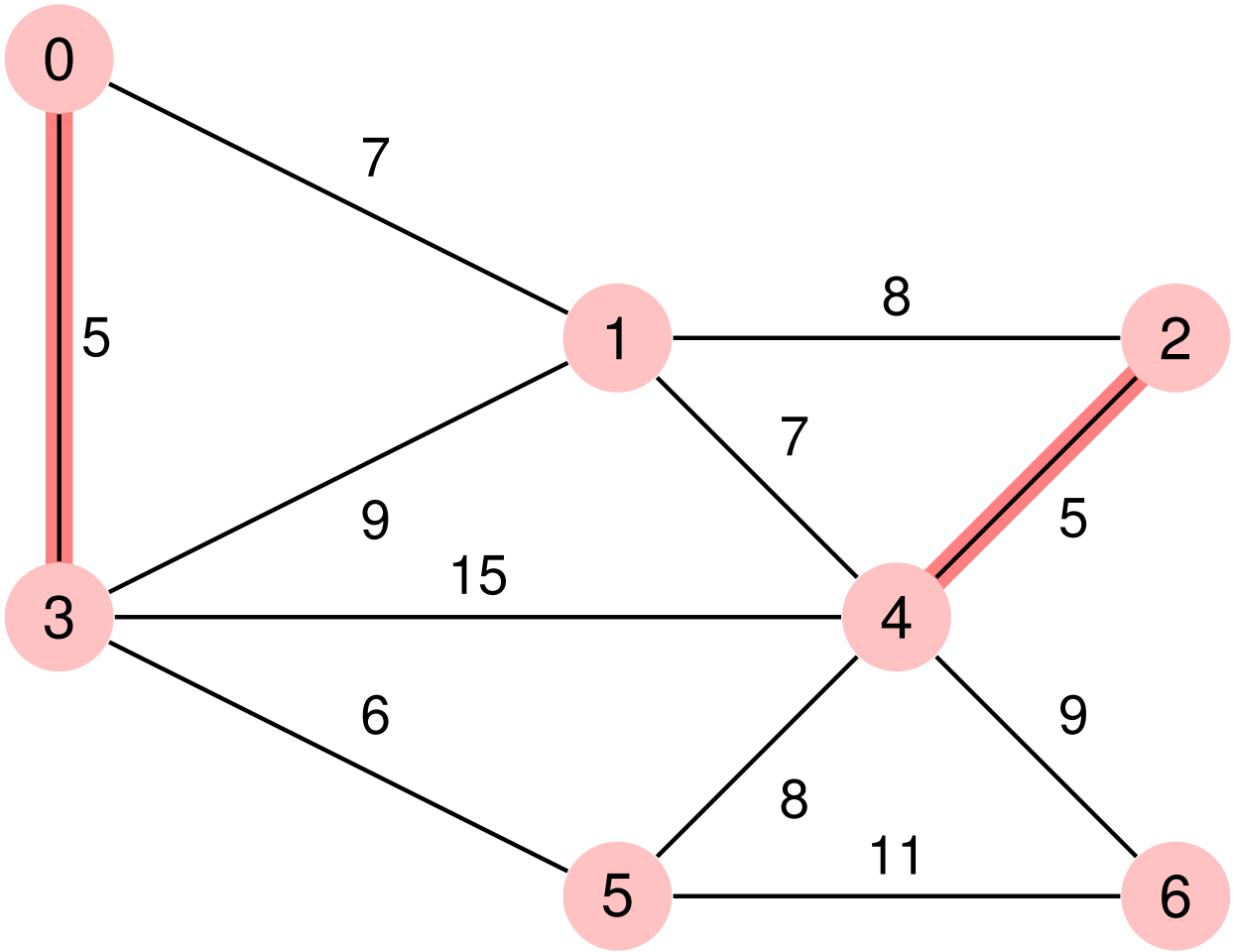
Ejemplo



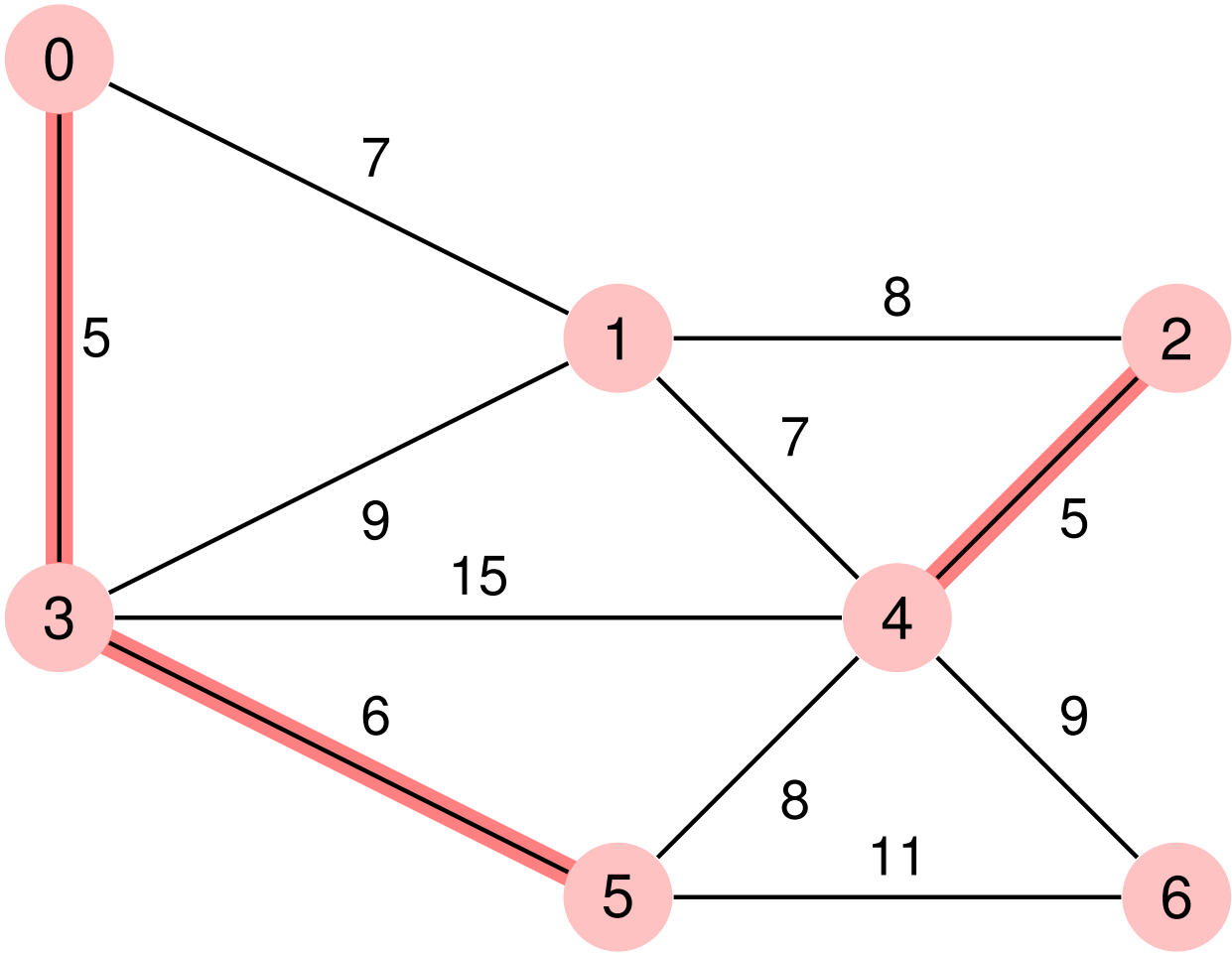
Ejemplo



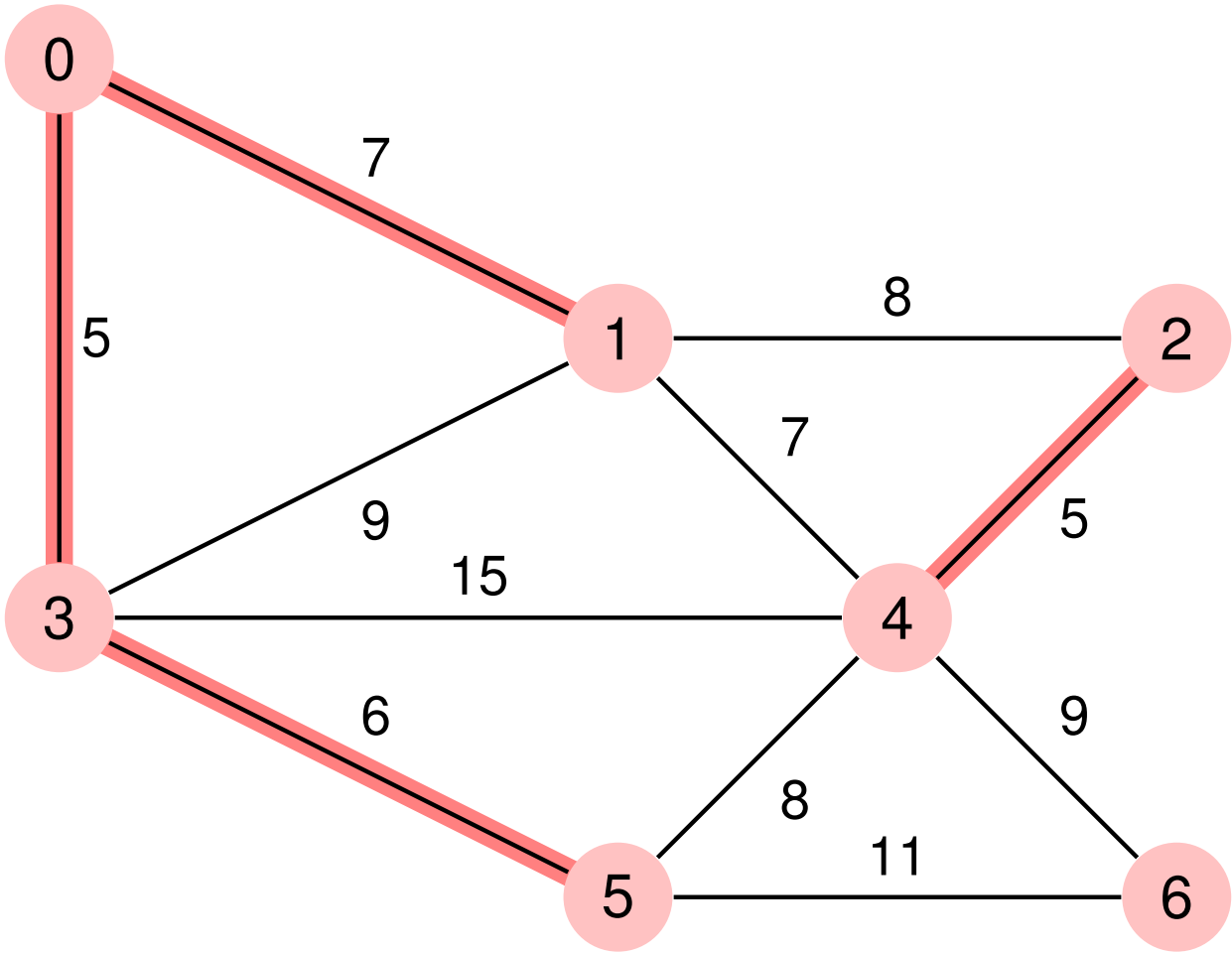
Ejemplo



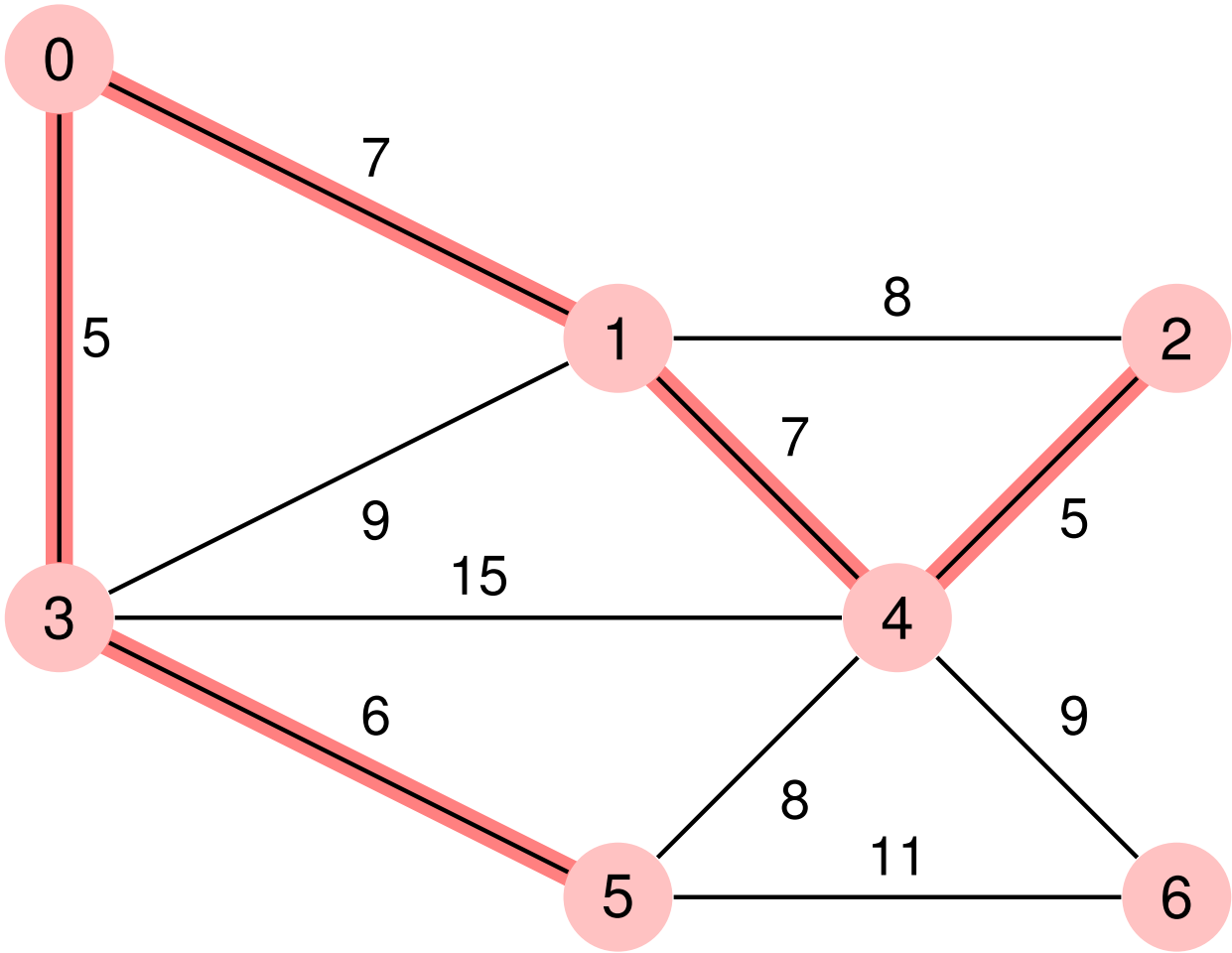
Ejemplo



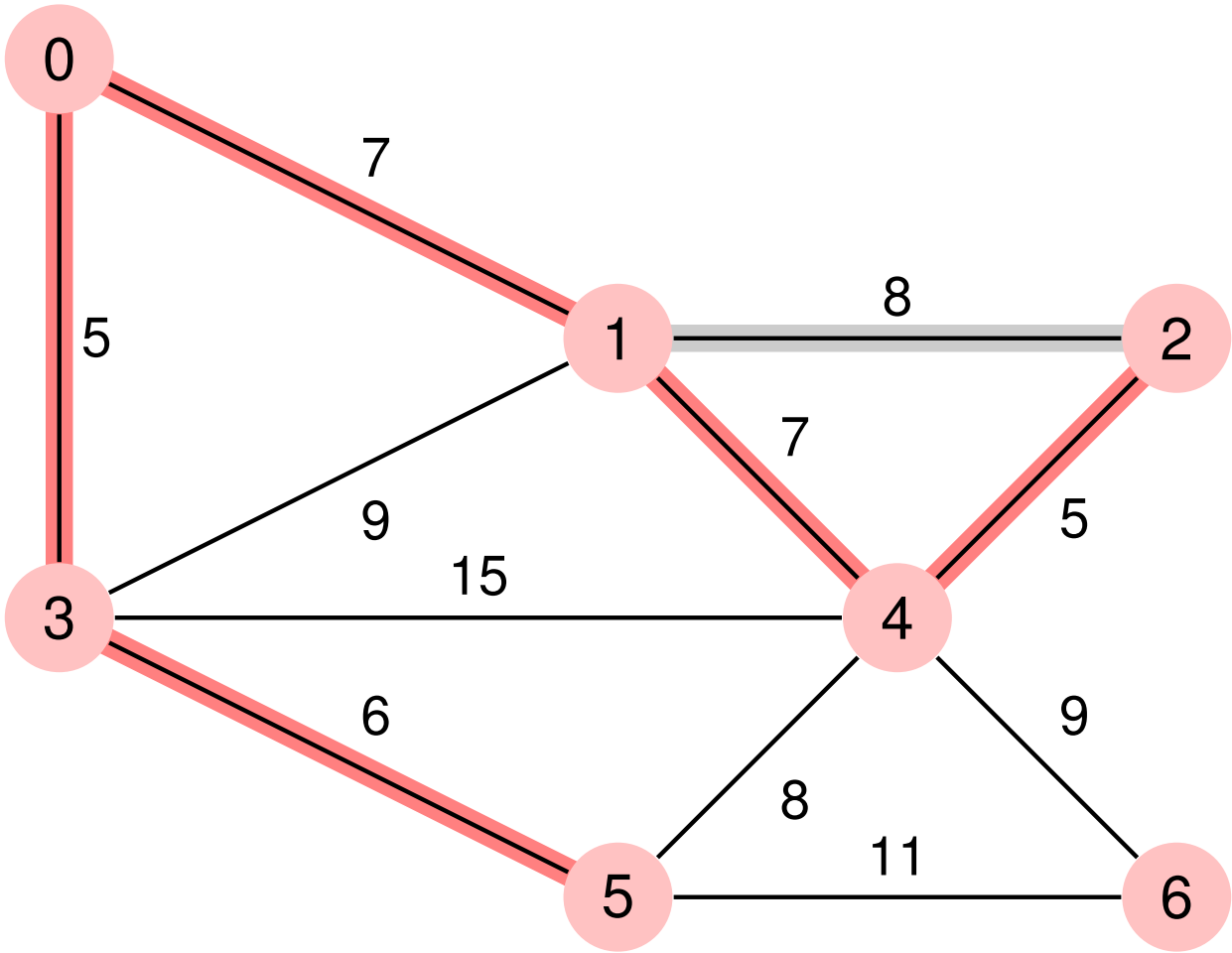
Ejemplo



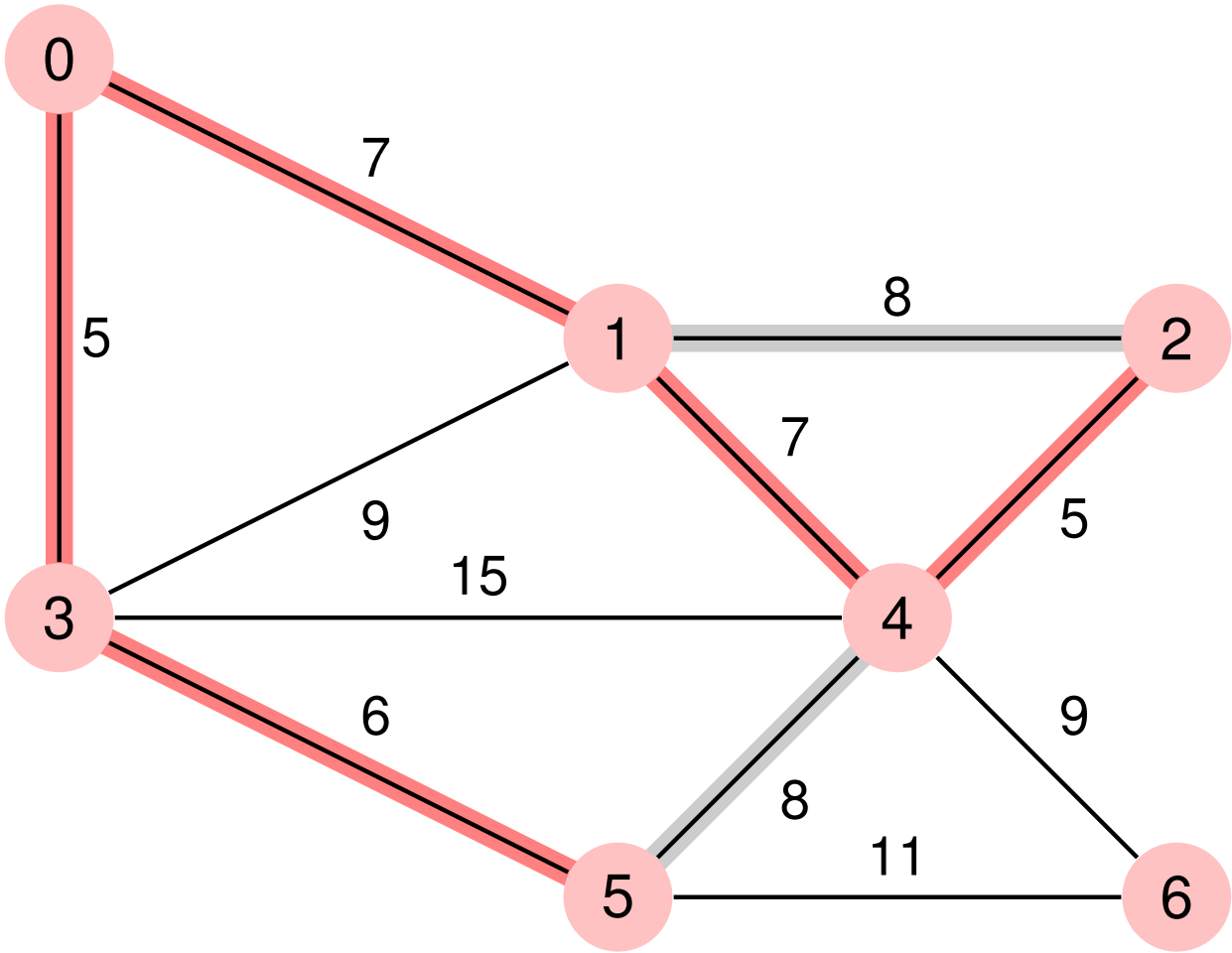
Ejemplo



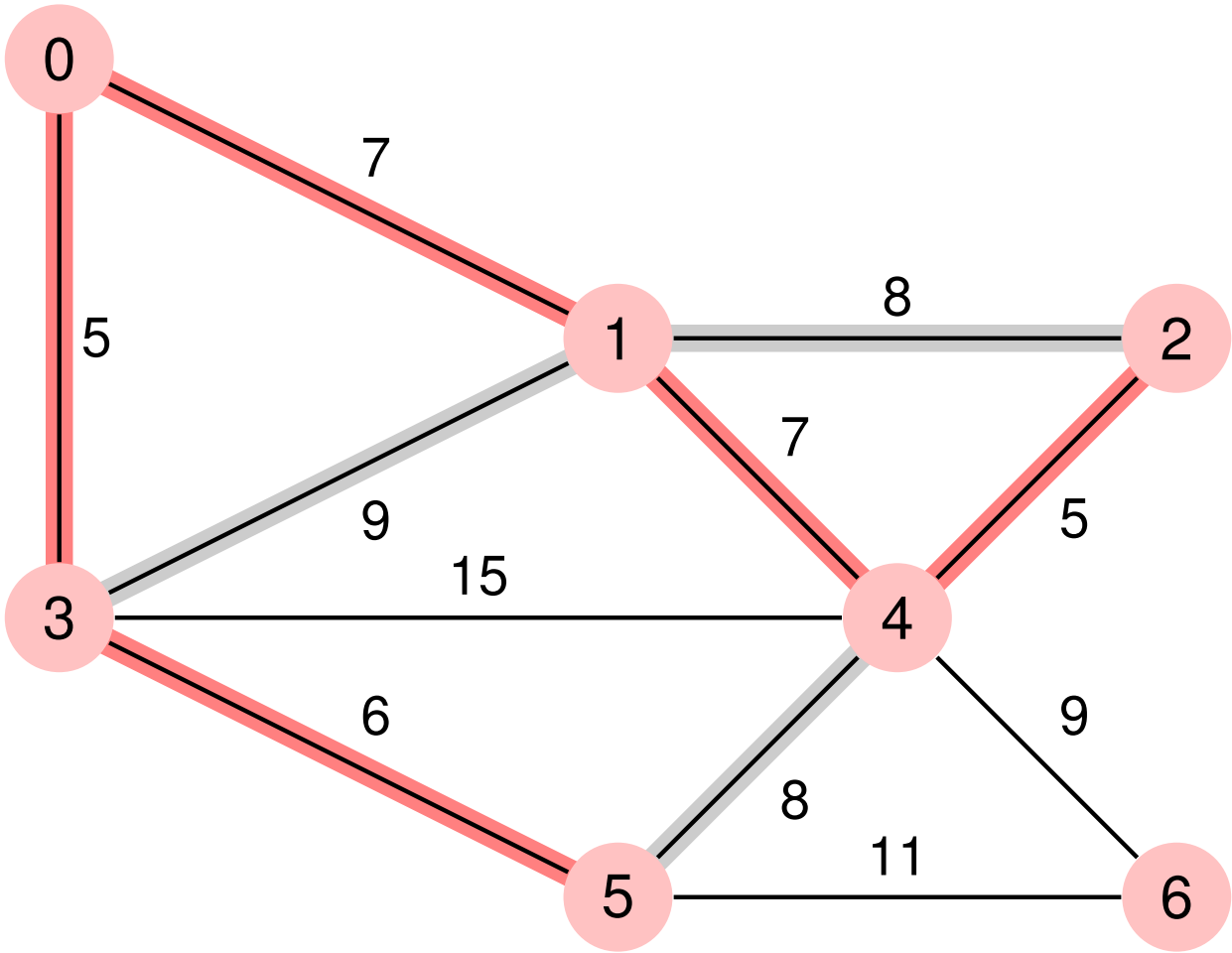
Ejemplo



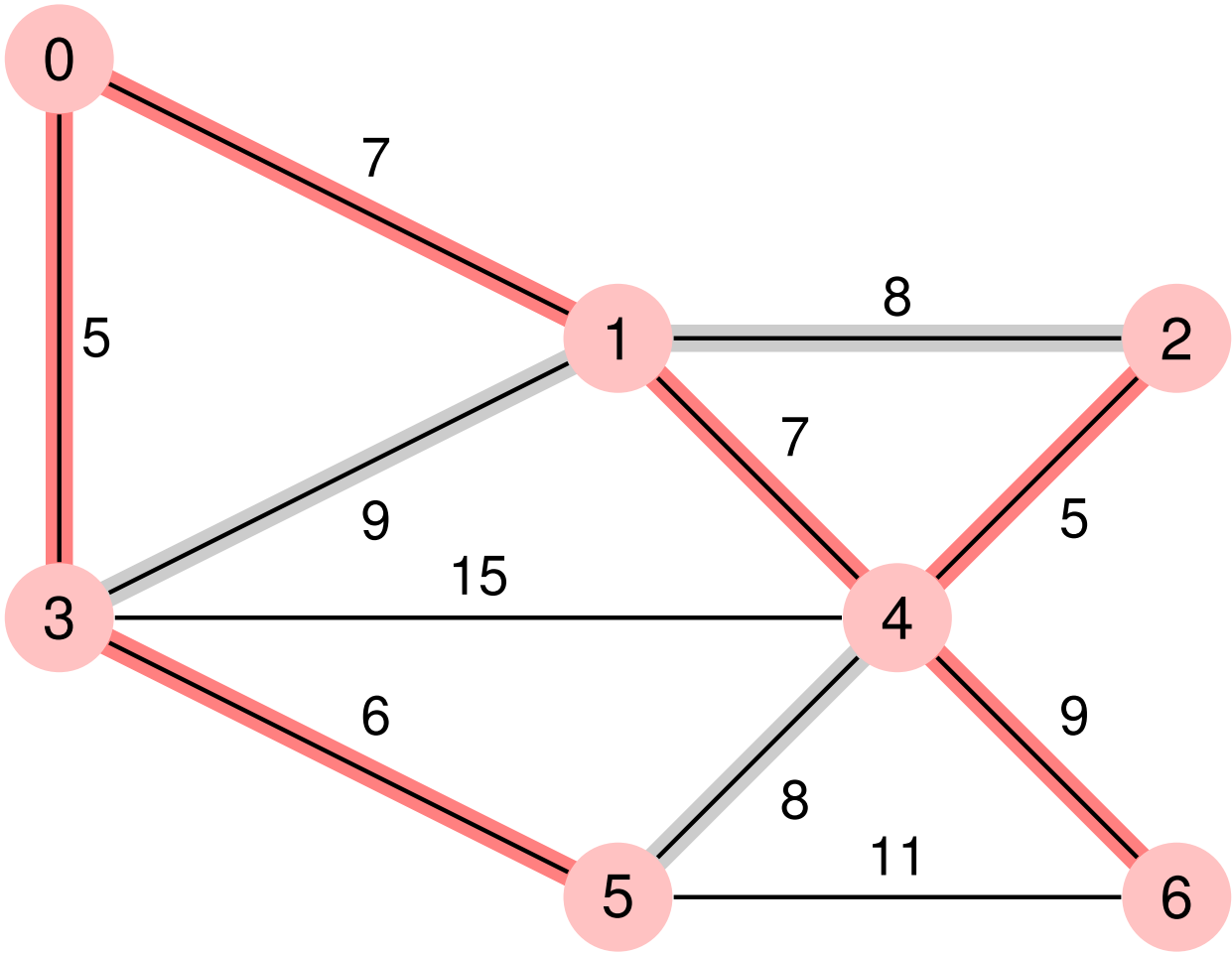
Ejemplo



Ejemplo



Ejemplo



Implementación de Kruskal

```
1  int kruskal(vector<pair<int,pair<int,int> > > ejes, int n)
2  {
3      sort(ejes.begin(),ejes.end());
4      initUF(n);
5      int u = 0;
6      long long t = 0;
7      for(int i=0;i<ejes.size();i++)
8      if(find(ejes[i].second.first)!=find(ejes[i].second.second)){
9          u++;
10         t+=ejes[i].first;
11         if(u==n-1)
12             return t;
13         join(ejes[i].second.first,ejes[i].second.second);
14     }
15     return -1;
16 }
```

Análisis de Kruskal

- En ejes recibimos los ejes como pares $(p, (v_1, v_2))$ que representa un eje de peso p entre los nodos v_1 y v_2, al ordenarlos se ordenan por peso.

Análisis de Kruskal

- En ejes recibimos los ejes como pares $(p, (v_1, v_2))$ que representa un eje de peso p entre los nodos v_1 y v_2, al ordenarlos se ordenan por peso.
- Ahora empezamos a recorrer todos los ejes, cada vez que podemos agregar un eje sumamos su peso, si recorrimos todos los ejes y no conectamos el grafo es porque este no era conexo y devolvemos -1.

Análisis de Kruskal

- En ejes recibimos los ejes como pares $(p, (v_1, v_2))$ que representa un eje de peso p entre los nodos v_1 y v_2, al ordenarlos se ordenan por peso.
- Ahora empezamos a recorrer todos los ejes, cada vez que podemos agregar un eje sumamos su peso, si recorrimos todos los ejes y no conectamos el grafo es porque este no era conexo y devolvemos -1.
- Si llegamos a $n - 1$ aristas donde $n = \#V$ entonces ya tenemos el árbol y devolvemos la suma de sus pesos.

Análisis de Kruskal

- En ejes recibimos los ejes como pares $(p, (v_1, v_2))$ que representa un eje de peso p entre los nodos v_1 y v_2, al ordenarlos se ordenan por peso.
- Ahora empezamos a recorrer todos los ejes, cada vez que podemos agregar un eje sumamos su peso, si recorrimos todos los ejes y no conectamos el grafo es porque este no era conexo y devolvemos -1.
- Si llegamos a $n - 1$ aristas donde $n = \#V$ entonces ya tenemos el árbol y devolvemos la suma de sus pesos.
- Podemos modificar levemente el algoritmo para que devuelva los ejes en lugar de la suma de sus pesos pero buscar la suma de los pesos del AGM suele ser un problema bastante frecuente y por eso dimos este algoritmo.

Contenidos

Algoritmo de Prim

Descripción del Algoritmo

El algoritmo de Prim es parecido al algoritmo de Dijkstra. Se empieza por un nodo como AGM y se le va agregando siempre el vértice más cercano al AGM actual, hasta que el AGM tiene los n vértices. Para esto se guardan las distancias al AGM de todos los vértices aún no agregados al mismo.

Algoritmo de Prim

Descripción del Algoritmo

El algoritmo de Prim es parecido al algoritmo de Dijkstra. Se empieza por un nodo como AGM y se le va agregando siempre el vértice más cercano al AGM actual, hasta que el AGM tiene los n vértices. Para esto se guardan las distancias al AGM de todos los vértices aún no agregados al mismo.

- Al igual que Dijkstra, Prim se puede implementar con o sin cola de prioridad, y tiene las mismas complejidades que Dijkstra. Implementandolo con cola de prioridad con un set de C++ la complejidad es $O((m+n)\log n)$.

Comparaciones Kruskal vs. Prim

Implementación

Por lo general es conveniente implementar uno u otro algorimo según cómo venga la entrada, ya que Kruskal toma lista de ejes y Prim, por lo general, lista de adyacencia. También, hay variantes del problema, que pueden ser resueltos por uno de los algoritmos y no por el otro.

Comparaciones Kruskal vs. Prim

Implementación

Por lo general es conveniente implementar uno u otro algorimo según cómo venga la entrada, ya que Kruskal toma lista de ejes y Prim, por lo general, lista de adyacencia. También, hay variantes del problema, que pueden ser resueltos por uno de los algoritmos y no por el otro.

Complejidades

La complejdiad de Kruskal es $O(m \log m)$ mientras que la de Prim es $O(n^2)$ u $O((m+n) \log n)$ según como se implemente. Para grafos con pocos ejes es conveniente Kruskal ya que la complejidad depende de m que es muy parecido a n, para grafos con muchos ejes suele ser más conveniente Prim.

Contenidos

Cajas y herramientas

Tenemos n ($1 \leq n \leq 10^5$) herramientas numeradas de 1 a n y n cajas numeradas de 1 a n. Inicialmente la herramienta i empieza en la caja i. Hay q ($1 \leq q \leq 10^5$) queries, en cada una hay que mover todas las herramientas de la caja donde esta la herramienta s a la caja donde esta la herramienta t. Debemos decir cuantas herramientas hay en la caja donde quedaron todas juntas y si ya estaban juntas o no.

Otro problema: codeforces.com/gym/100571/problem/F

Cajas y herramientas

Tenemos n ($1 \leq n \leq 10^5$) herramientas numeradas de 1 a n y n cajas numeradas de 1 a n. Inicialmente la herramienta i empieza en la caja i. Hay q ($1 \leq q \leq 10^5$) queries, en cada una hay que mover todas las herramientas de la caja donde esta la herramienta s a la caja donde esta la herramienta t. Debemos decir cuantas herramientas hay en la caja donde quedaron todas juntas y si ya estaban juntas o no.
Otro problema: codeforces.com/gym/100571/problem/F

Engranajes

Tenemos *n* discos (dados con centro y radio) en el plano. El 1 está conectado a un motor y rota. Queremos conectarlos con bandas de manera que todos se muevan. ¿Cuál es la menor longitud total de las bandas necesaria?

Sponge Island

Deep in the Carribean, there is an island even stranger than the Monkey Island, dwelled by Horatio Torquemada Marley. Not only it has a rectangular shape, but is also divided into an $N \times M$ grid. Each grid field has a certain height. Unfortunately, the sea level started to raise and in year i, the level is i meters. Another strange feature of the island is that it is made of sponge, and the water can freely flow through it. Thus, a grid field whose height is at most the current sea level is considered flooded. Adjacent unflooded fields (i.e., sharing common edge) create unflooded areas. Sailors are interested in the number of unflooded areas in a given year.
$(1 \leq n, m \leq 300, 1 \leq T \leq 10^5, 0 \leq t_1 \leq t_2 \leq \cdots \leq t_T \leq 10^9)$.

Desigualdades

Tenemos variables $x_1, x_2, \ldots, x_n$ $(1 \leq n \leq 1000)$.
Y desigualdades de la forma $x_{a_i} - x_{b_i} \geq k_i$ (1000 de estas).
¿Es compatible?

Ultimo problema

Queremos un árbol que minimice la suma de las distancias al nodo 0.