Teoría de Grafos

Camino más corto

B.S. Rodolfo Mercado Gonzales
Universidad Nacional de Ingeniería

Peso de un Camino

Es igual a la suma de los pesos de las aristas que constituyen el camino.

Sea un camino: $\boldsymbol{p = \{ v_0, v_1, \dots, v_k \}}$

$$\boldsymbol{w(p) = \sum_{i=1}^{k} w(v_{i-1}, v_i)}$$

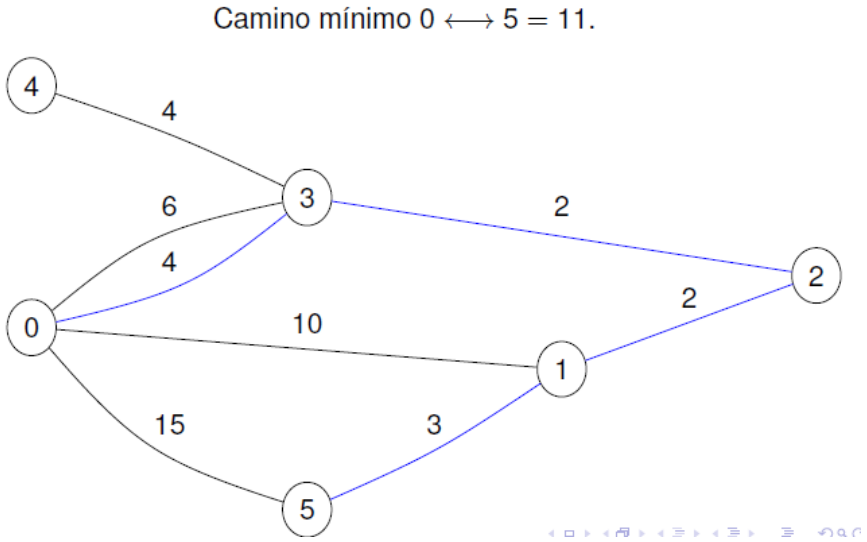
Camino más Corto

El camino más corto desde un vértice $\boldsymbol{u}$ hacia un vértice $\boldsymbol{v}$, esta definido como el camino $\boldsymbol{p} = \{\boldsymbol{u}, \dots, \boldsymbol{v}\}$ que tiene menor peso y es denotado por $\boldsymbol{d}(\boldsymbol{u}, \boldsymbol{v})$.

$$\boldsymbol{d}(\boldsymbol{u}, \boldsymbol{v}) = \begin{cases} \boldsymbol{min}(\, \boldsymbol{w}(\boldsymbol{p}) \;\; / \, \boldsymbol{p} = \{\boldsymbol{u}, \dots, \boldsymbol{v}\}\,) & , si\ existe\ camino\ de\ \boldsymbol{u}\ a\ \boldsymbol{v} \\ \boldsymbol{\inf} \end{cases}$$

Camino más Corto

Cuando se tiene un grafo con pesos, el BFS no da necesariamente la respuesta esperada.



Definición

$\boldsymbol{dis}[\,\boldsymbol{u}\,]$: cota superior del peso del camino mínimo desde el origen hacia $\boldsymbol{u}$.

$\boldsymbol{d}[\,\boldsymbol{u}\,]$: peso del camino mínimo desde el origen hacia $\boldsymbol{u}$.

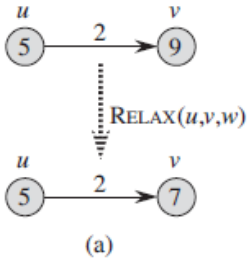
Inicialización

En todos los algoritmos de “camino más corto desde un origen” empezaremos **inicializando** $dis[u]$ para cada vértice del grafo (el origen posee peso 0 y el resto peso infinito)

```
for( int i = 0; i < n; ++i ) dis[ i ] = inf;
dis[ s ] = 0;
```

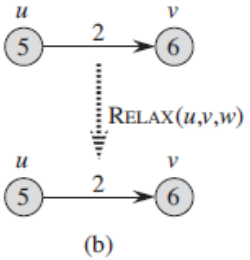
Relajación

Relajar una arista $(\boldsymbol{u}, \boldsymbol{v})$ consiste en evaluar si es posible mejorar el camino más corto hacia $\boldsymbol{v}$ ($\boldsymbol{dis}[\,\boldsymbol{v}\,]$) usando dicha arista.



(a)

```
void relax( int u, int v, int w_uv ){
    if( dis[ v ] > dis[ u ] + w_uv ){
        dis[ v ] = dis[ u ] + w_uv;
    }
}
```



(b)

Propiedades

Propiedad de Relajación de la arista

Luego de relajar una arista $(\boldsymbol{u}, \boldsymbol{v})$ se cumple : $\boldsymbol{dis}[\boldsymbol{v}] \leq \boldsymbol{dis}[\boldsymbol{u}] + \boldsymbol{w}(\boldsymbol{u}, \boldsymbol{v})$

Desigualdad triangular

Para cualquier arista $(\boldsymbol{u}, \boldsymbol{v})$ se cumple : $\boldsymbol{d}[v] \leq \boldsymbol{d}[u] + \boldsymbol{w}(\boldsymbol{u}, \boldsymbol{v})$

Propiedad de la Cota Superior

Para cualquier vértice $\boldsymbol{v}$ se cumple $\boldsymbol{dis}[\boldsymbol{v}] \geq \boldsymbol{d}[\boldsymbol{v}]$ sin importar el número de relajaciones que se hagan. Una vez que $\boldsymbol{dis}[\boldsymbol{v}]$ alcanza el valor de $\boldsymbol{d}[\boldsymbol{v}]$ ya nunca cambia.

Propiedades

Propiedad de la convergencia

Sea $\boldsymbol{p} = \{\boldsymbol{s}, \ldots., \boldsymbol{u}, \boldsymbol{v}\}$ un camino mínimo hacia $\boldsymbol{v}$. Si $\boldsymbol{dis}[\boldsymbol{u}] = \boldsymbol{d}[\boldsymbol{u}]$ antes de relajar la arista $(\boldsymbol{u}, \boldsymbol{v})$, entonces $\boldsymbol{dis}[\boldsymbol{v}] = \boldsymbol{d}[\boldsymbol{v}]$ luego de relajarla.

Propiedad de Relajación del Camino

Sea $\boldsymbol{p} = \{\boldsymbol{v_0}, \boldsymbol{v_1}, \ldots, \boldsymbol{v_k}\}$ el camino más corto de $\boldsymbol{v_0}$ *a* $\boldsymbol{v_k}$.
Si relajamos todas las aristas de $\boldsymbol{p}$ en orden $(\boldsymbol{v_0}, \boldsymbol{v_1})$, $(\boldsymbol{v_1}, \boldsymbol{v_2}), \ldots, (\boldsymbol{v_{k-1}}, \boldsymbol{v_k})$, entonces $\boldsymbol{dis}[\boldsymbol{k}] = \boldsymbol{d}[\boldsymbol{k}]$

Propiedades

Los subcaminos de un “camino más corto” también son “caminos más cortos”

Sea $\boldsymbol{p} = \{\boldsymbol{v_0}, \boldsymbol{v_1}, \dots, \boldsymbol{v_k}\}$ el camino más corto de $\boldsymbol{v_0}$ *a* $\boldsymbol{v_k}$ y $\boldsymbol{p_{ij}} = \{\boldsymbol{v_i}, \boldsymbol{v_{i+1}}, \dots, \boldsymbol{v_j}\}$ un subcamino de $\boldsymbol{p}$ $(\boldsymbol{0 \leq i \leq j \leq k})$, entonces $\boldsymbol{p_{ij}}$ es el camino más corto desde $\boldsymbol{v_i}$ a $\boldsymbol{v_j}$

Algoritmo de Bellman-Ford

Resuelve el problema del **“camino más corto desde un origen”** de forma general, es decir pueden existir aristas con pesos negativos.

Algoritmo de Bellman-Ford

Caso 1 : No hay ciclos negativos alcanzables desde el origen

- En un camino más corto es absurdo que existan ciclos positivos.
- En el peor de los casos un camino más corto tendrá $\boldsymbol{V}$ nodos y $\boldsymbol{V-1}$ aristas.
- Por la propiedad de “relajación del camino”, basta con relajar $\mathbf{V-1}$ veces todas la aristas y obtendremos todos los caminos más cortos.

Algoritmo de Bellman-Ford

Paso 1 : Inicializamos los **dis[u]**

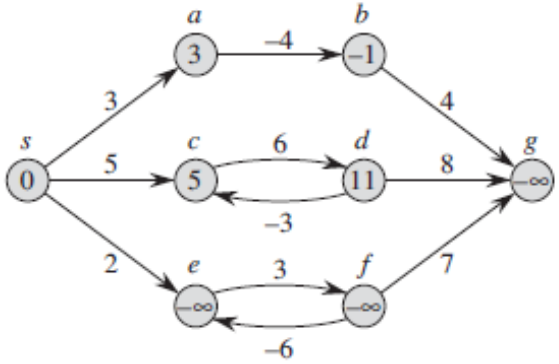
Paso 2 : Relajamos $\mathbf{V} - \mathbf{1}$ veces todas la aristas

$$O(VE)$$

Algoritmo de Bellman-Ford

Caso 2 : Existen ciclos negativos alcanzables desde el origen

Si un grafo tiene un ciclo de peso negativo alcanzable desde el origen, entonces no todos los nodos tendrán el camino más corto bien definido.



Algoritmo de Bellman-Ford

¿Cómo detectamos si hay ciclos de peso negativo alcanzable desde el origen?

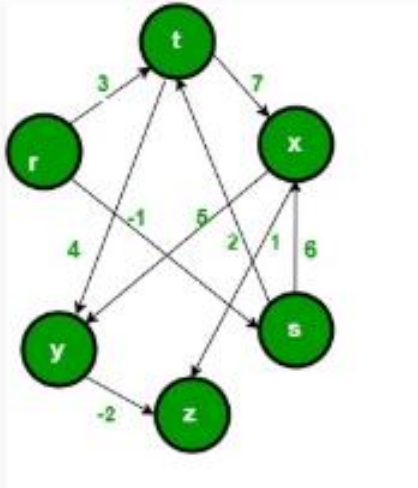
Luego de relajar $\mathbf{V} - \mathbf{1}$ veces todas las aristas todos los caminos más cortos han sido obtenidos. Si luego de ello todavía es posible relajar alguna arista entonces existe un ciclo de peso negativo.

Problemas

UVA 558 – Wormholes

UVA 10557 - XYZZY

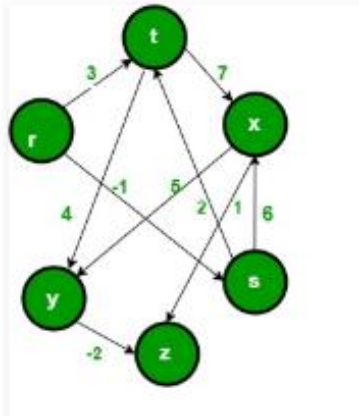
Camino más corto en un DAG

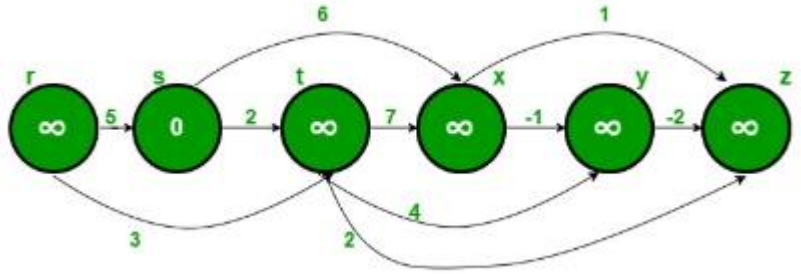


Los caminos más cortos están bien definidos ya que no existen ciclos de peso negativo.

Camino más corto en un DAG

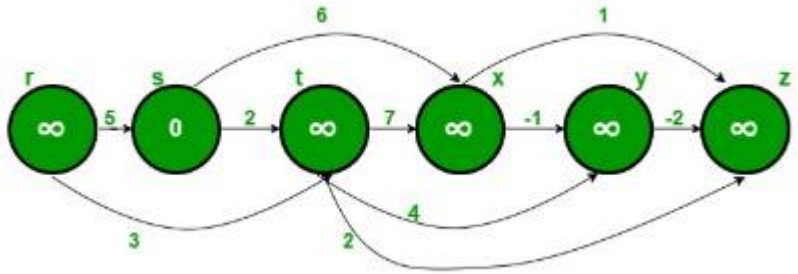
¿Qué pasa si hacemos un ordenamiento topológico?





Camino más corto en un DAG

¿Qué pasa si hacemos un ordenamiento topológico?



Basta con relajar las aristas en el orden topológico (por la propiedad de relajación del camino)

O(V+ E)

Difference Constraints

En total tenemos $\mathbf{n}$ incógnitas por descubrir y $\mathbf{m}$ restricciones, donde cada restricción es una desigualdad lineal de la forma:

$$\boldsymbol{x_j - x_i \leq b_k}\,, \qquad donde\ 1 \leq i, j \leq n, \qquad i \neq j\ \ , 1 \leq k \leq m$$

Difference Constraints

Para **n** = 3 y **m** = 3

$$
\begin{aligned}
x_1 - x_2 &\leq 10 \\
x_2 - x_3 &\leq 20 \\
x_3 - x_1 &\leq 5
\end{aligned}
$$

$$
\begin{pmatrix} 1 & -1 & 0 \\ 0 & 1 & -1 \\ -1 & 0 & 1 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} \leq \begin{pmatrix} 10 \\ 20 \\ 5 \end{pmatrix}
$$

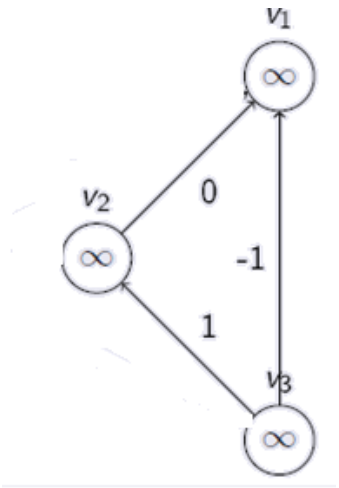
Grafo de Constraints

Construimos un grafo $G(V, E)$, donde

- $V = \{v_1, v_2, \dots, v_n\}$
- $\mathrm{E} = \{(v_i, v_j) \mid (x_j - x_i) \leq b_k \; es \; una \; constraint\}$
- $w(v_i, v_j) = \begin{cases} b_k, & (x_j - x_i) \leq b_k \; es \; una \; constraint \end{cases}$

Grafo de Constraints

$$x_1 - x_2 \leq 0$$
$$x_1 - x_3 \leq -1$$
$$x_2 - x_3 \leq 1$$



Grafo de Constraints

Teorema

Si un grafo de constraints tiene un ciclo de pesos negativos, entonces el sistema de difference constraints no se puede satisfacer.

Grafo de Constraints

Demostración

Sea el ciclo negativo $v_1 \rightarrow v_2, \rightarrow \cdots \rightarrow v_k \rightarrow v_1$

$$x_2 - x_1 \leq w_{1\,2}$$

$$x_3 - x_2 \leq w_{2\,3}$$

....

$$x_k - x_{k-1} \leq w_{k-1\,k}$$

$$x_1 - x_k \leq w_{k1}$$

Por lo tanto 0 <= W(ciclo negativo) , contradicción!!!

Grafo de Constraints

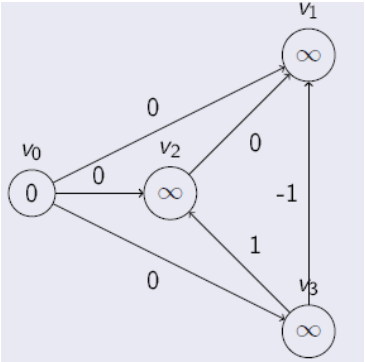
Teorema

Si un grafo de constraints no tiene un ciclo de pesos negativos, entonces el sistema de difference constraints se puede satisfacer.

Grafo de Constraints

Demostración

Agreguemos un nodo **origen** al grafo, con aristas de peso **0** desde el origen hacia el resto de nodos.



El nuevo grafo no tiene ciclos de peso negativo y existe camino del origen a los demás nodos.

Grafo de Constraints

¿ Qué pasa si hallamos los caminos más cortos y hacemos que $\mathbf{x_i = d[v_i]}$?

$x_j - x_i \leq w_{ij} \leftrightarrow \boldsymbol{d[v_j]} - \boldsymbol{d[v_i]} \leq \boldsymbol{w(v_i, v_j)} \leftrightarrow \boldsymbol{d[v_j]} \leq \boldsymbol{d[v_i]} + \boldsymbol{w(v_i, v_j)}$

Y por desigualdad triangular sabemos que en un grafo , para cada arista $(\boldsymbol{v_i, v_j})$

$\boldsymbol{d[v_j]} \leq \boldsymbol{d[v_i]} + \boldsymbol{w(v_i, v_j)}$

Grafo de Constraints

El sistema de difference constraints no tiene solución si el grafo posee algún ciclo de peso negativo, caso contrario las soluciones son $\boldsymbol{x_i = d[v_i]}$

Problemas

UVA 515 - King

Minimum Mean Cycle

Dado un grafo dirigido con pesos no negativos, encontrar un ciclo con el menor peso promedio

Minimum Mean Cycle

- Sea un ciclo con **n** aristas el de menor peso promedio, su promedio es :

$\frac{w_1+w_2+\cdots+w_n}{n} = p \rightarrow w_1 + w_2 + \cdots + w_n = p * n$

- Si ha cada arista le quitamos **c**, el promedio también se reduce en **c**.

$(p * n - c * n) / n = p - c$

Minimum Mean Cycle

- Cuando $c <= p$, no habrán ciclos negativos, caso contrario habrán ciclos negativos.

Problemas

UVA 11090 – Going in Cycle!!

LiveArchive 3531 – Word Rings

Algoritmo de Dijkstra

Resuelve el problema del **“camino más corto desde un origen”**, siempre y cuando las aristas tienen pesos no negativos.

Algoritmo de Dijkstra

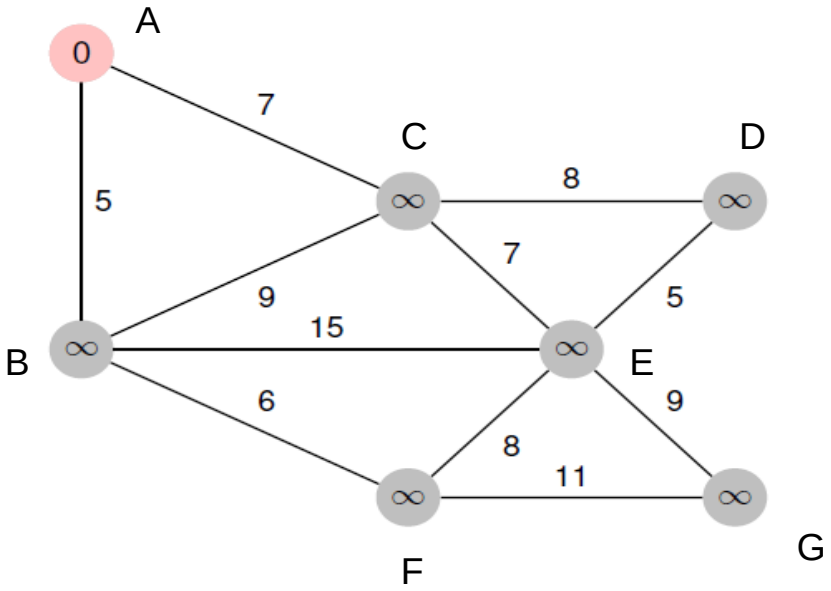
Paso 1 : Realizamos la inicialización.

Paso 2 : Realizamos n iteraciones

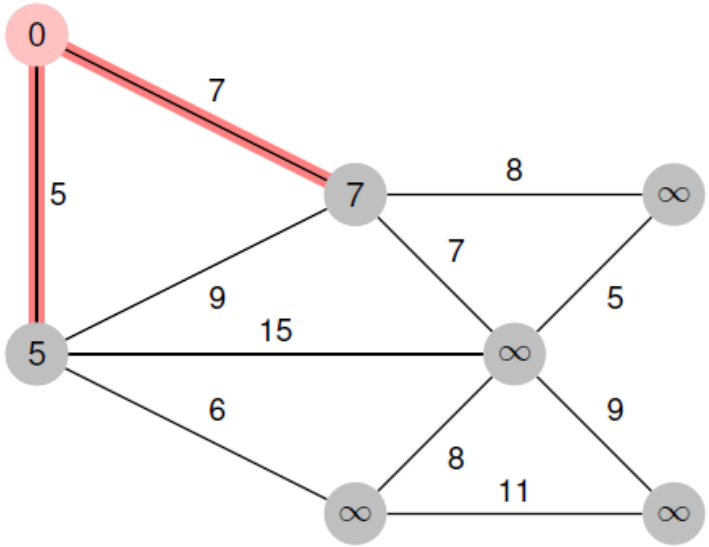
En cada iteración procesamos un vértice (su cota superior se convierte en su camino mínimo) y relajamos todas las aristas de dicho vértice.

El vértice escogido será el más cercano al origen en ese momento y que no haya sido procesado aún.

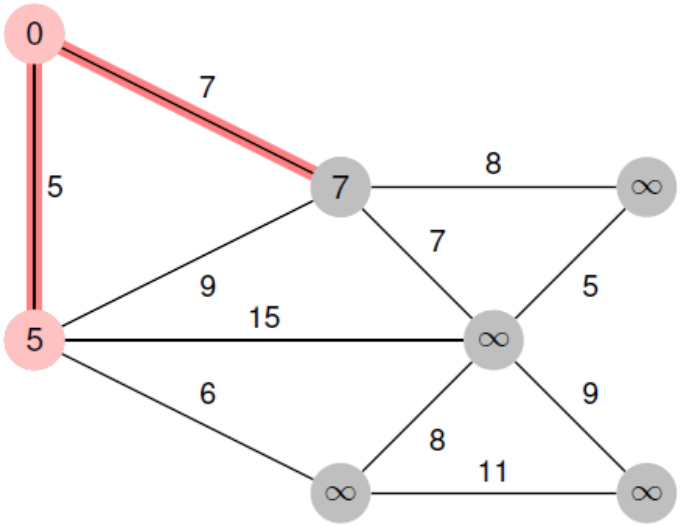
Algoritmo de Dijkstra



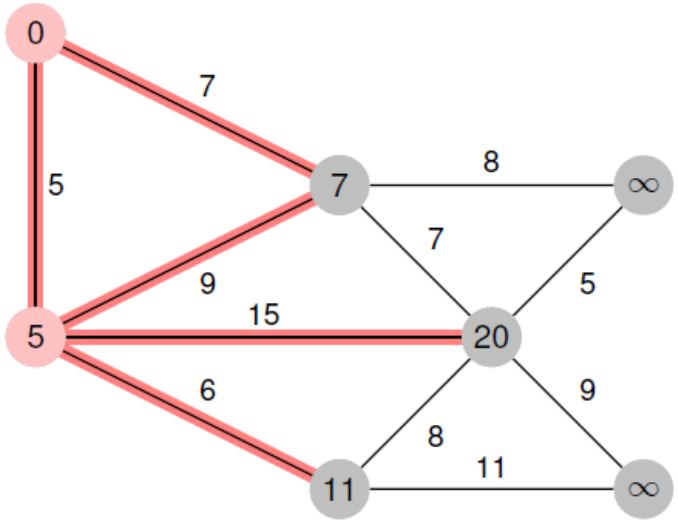
Algoritmo de Dijkstra



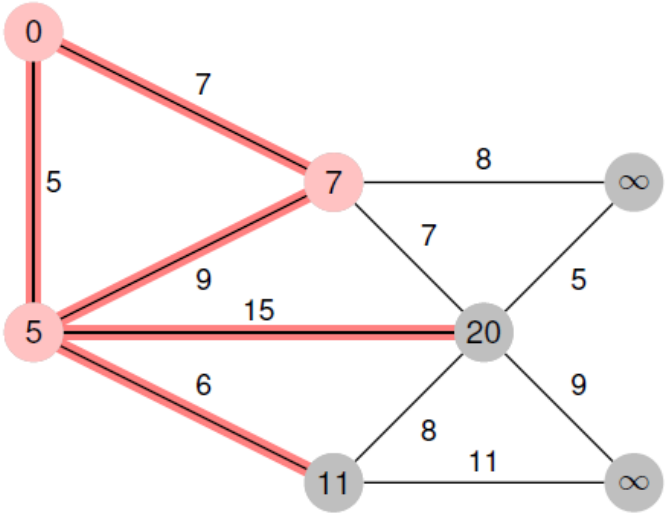
Algoritmo de Dijkstra



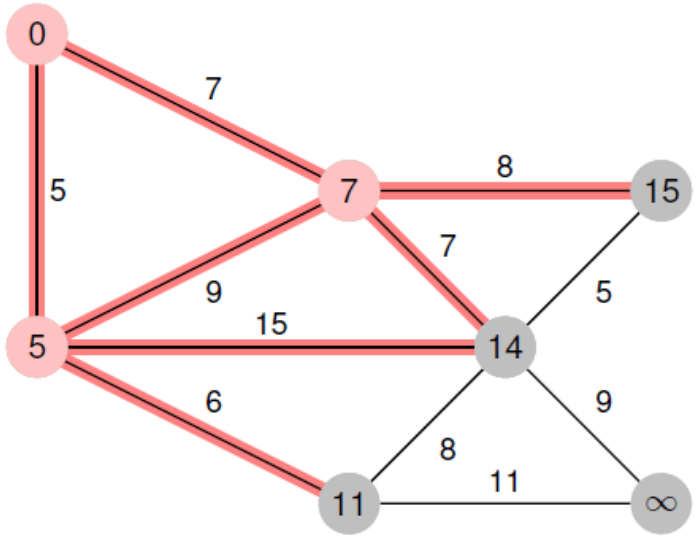
Algoritmo de Dijkstra



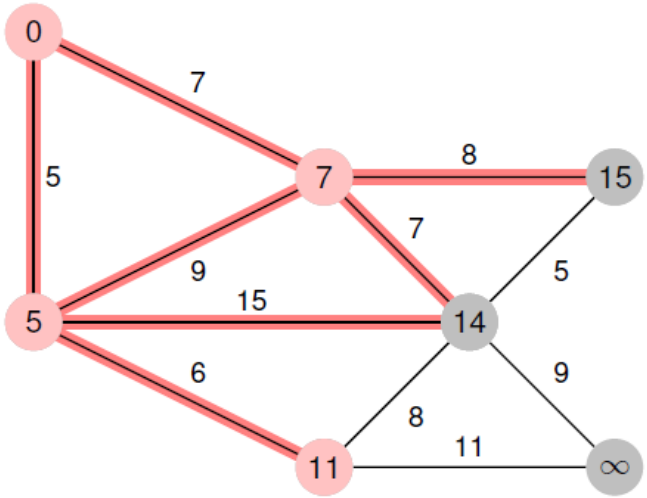
Algoritmo de Dijkstra



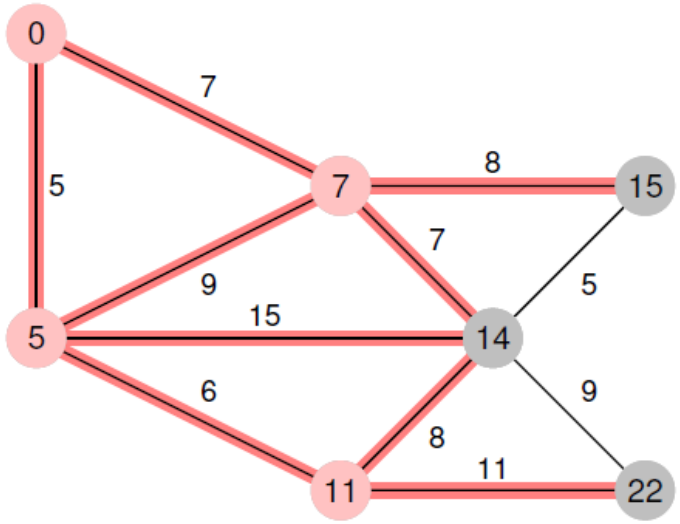
Algoritmo de Dijkstra



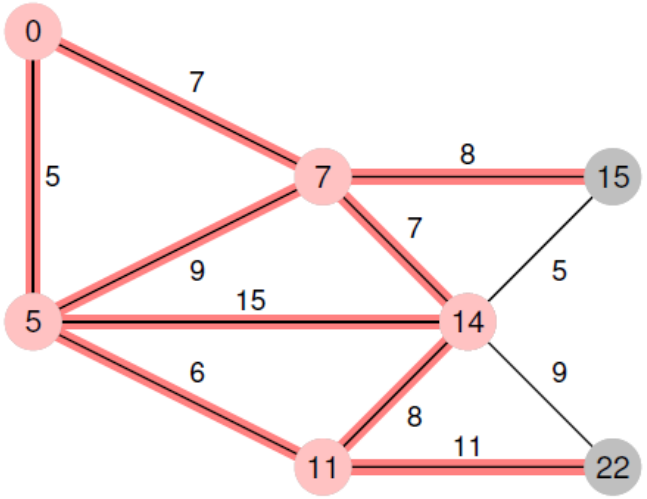
Algoritmo de Dijkstra



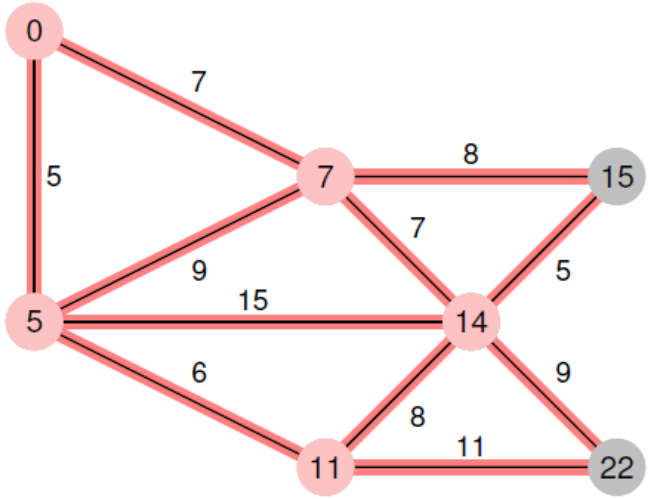
Algoritmo de Dijkstra



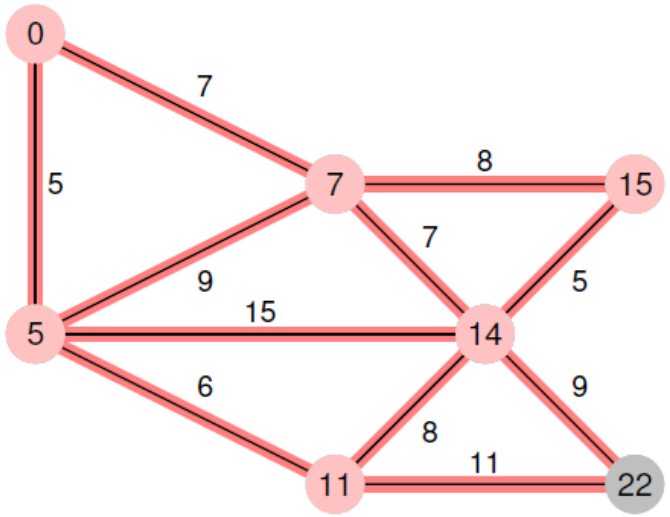
Algoritmo de Dijkstra



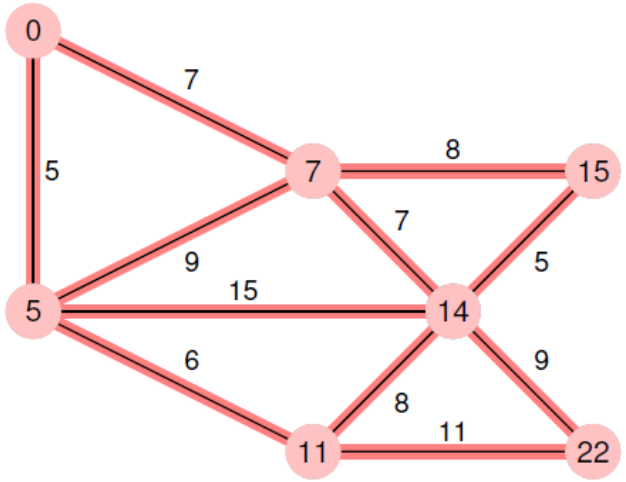
Algoritmo de Dijkstra



Algoritmo de Dijkstra



Algoritmo de Dijkstra



Algoritmo de Dijkstra

```
vector<int> dijkstra(int s){
  vector<int> dis( n, inf );
  vector<bool> used( n, 0 );
  dis[ s ] = 0;
  for( int i = 0; i < n; ++i ){
      int minDist = inf, u = -1 ;
      for( int j = 0; j < n; ++j ){
          if( !used[ j ] && dis[ j ] < minDist ){
              minDist = dis[ j ];
              u = j;
           }
     }
     if( u == -1 ) break;
     used[ u ] = 1;
        for( int j = 0; j < adj[ u ].size(); ++j ){
      int v = adj[ u ][ j ];
      int w = cost[ u ][ j ];
          int temp = dis[ u ] + w ;
          if( temp < dis[ v ]) dis[ v ] = temp;
      }
   }
   return dis;
}
```

$$O(V^2)$$

Algoritmo de Dijkstra

```
struct nodo{
  int u,d;
  nodo(int a,int b){ u = a, d = b; }
  bool operator < (nodo X)const{
    return d > X.d;
  }
};
```

```
vector<int> dijkstra( int s ){
  vector<int> dis( n, inf );
  vector<bool> used( n, 0 );
  dis[ s ] = 0;
  priority_queue<nodo> Q;
  Q.push ( nodo( s, 0 ) );
  while( !Q.empty() ){
    nodo p = Q.top();
    int u = p.u;
    Q.pop();
    if( used[ u ] ) continue;
    used[ u ] = 1;
    for(int i = 0; i < adj[ u ].size(); ++i ){
      int v = adj[ u ][ i ];
      int w = cost[ u ][ i ];
      int temp = dis[ u ] + w;
      if( temp < dis[ v ] ){
        Q.push( nodo( v, temp ) );
        dis[ v ] = temp;
      }
    }
  }
  return dis;
}
```

$$O(E\ log\mathrm{V})$$

Algoritmo de Dijkstra

Invariante

Para cada uno de los vértices procesados $\boldsymbol{u}$, $\boldsymbol{dis}[\,\boldsymbol{u}\,]$ es la distancia mínima desde el origen hacia el vértice $\boldsymbol{u}$ y para los vértices no procesados $\boldsymbol{v}$, $\boldsymbol{dis}[\,\boldsymbol{v}\,]$ es la distancia mínima desde el origen hacia el vértice $\boldsymbol{v}$ pero usando solamente vértices procesados (cota superior).

Algoritmo de Dijkstra (demostración)

Supongamos que ya hemos procesado $k-1$ vértices y queremos procesar el k-ésimo

Ahora si escogemos una arista (u, v) , donde v tiene el menor $dis[\,v\,]$ (u representa procesados y v no procesados) y se cumple $dis[\,v\,] = dis[\,u\,] + peso(\,u, v\,)$, entonces $dis[\,v\,]$ ahora es la menor distancia del origen hacia el vértice v ($d[\,v\,] = dis[\,v\,]$).

Si existiera otro camino más corto entonces debería pasar por algún vértices no procesado w, lo que haría que $dis[\,w\,]$ debería ser menor que $dis[\,v\,]$ y caeríamos en contradicción.

Entonces el nuevo vértice a procesar es v y la invariante se mantiene, ya que para cualquier vértice no procesado w el menor camino si pasa por v será actualizado (con la relajación).

Referencias

- T. Cormen, Introduction to Algorithms
- S. Halim, Competitive Programming 3
- MIT Open Courseware, Introduction to Algorithms (SMA 5503)

¡ Good luck and have fun !