

Algorithms and More

Blog about programming!

CAMINO MAS CORTO: ALGORITMO DE DIJKSTRA

Publicado el [marzo 19, 2012](#) | [59 comentarios](#)

Para el problema de la ruta corta tenemos varios algoritmos, en esta oportunidad se explicará el algoritmo de dijkstra el cual usa una técnica voraz (greedy). Al final del artículo se encuentran adjuntas las implementaciones en C++ y JAVA.

Descripción

El algoritmo de dijkstra determina la ruta más corta desde un nodo origen hacia los demás nodos para ello es requerido como entrada un grafo cuyas aristas posean pesos. Algunas consideraciones:

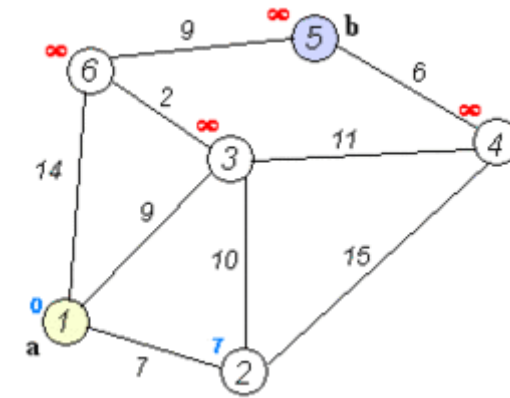
- Si los pesos de mis aristas son de valor 1, entonces bastará con usar el [algoritmo de BFS](#).
- Si los pesos de mis aristas son negativos no puedo usar el algoritmo de dijkstra, para pesos negativos tenemos otro algoritmo llamado Algoritmo de Bellman-Ford.

Como trabaja

Primero marcamos todos los vértices como no utilizados. El algoritmo parte de un vértice origen que será ingresado, a partir de ese vértice evaluaremos sus adyacentes, como dijkstra usa una técnica greedy – *La técnica greedy utiliza el principio de que para que un camino sea óptimo, todos los caminos que contiene también deben ser óptimos*- entre todos los vértices adyacentes, buscamos el que esté más cerca de nuestro punto origen, lo tomamos como punto intermedio y vemos si podemos llegar más rápido a través de este vértice a los demás. Después escogemos al siguiente más cercano (con las distancias ya actualizadas) y repetimos el proceso. Esto lo hacemos hasta que el vértice no utilizado más cercano sea nuestro destino. Al proceso de actualizar las distancias tomando como punto intermedio al nuevo vértice se le conoce como **relajación (relaxation)**.

Dijkstra es muy similar a [BFS](#), si recordamos [BFS](#) usaba una Cola para el recorrido para el caso de Dijkstra usaremos una Cola de Prioridad o Heap, este Heap debe tener la propiedad de Min-Heap es decir cada vez que extraiga un elemento del Heap me debe devolver el de menor valor, en nuestro caso dicho valor será el peso acumulado en los nodos.

Tanto java como C++ cuentan con una cola de prioridad ambos implementan un Binary Heap aunque con un Fibonacci Heap la complejidad de dijkstra se reduce haciéndolo mas eficiente, pero en un concurso mas vale usar la librería que intentar programar una nueva estructura como un Fibonacci Heap, claro que particularmente uno puede investigar y programarlo para saber como funciona internamente.



Algoritmo en pseudocódigo

Considerar distancia[i] como la distancia mas corta del vértice origen ingresado al vértice i.

```

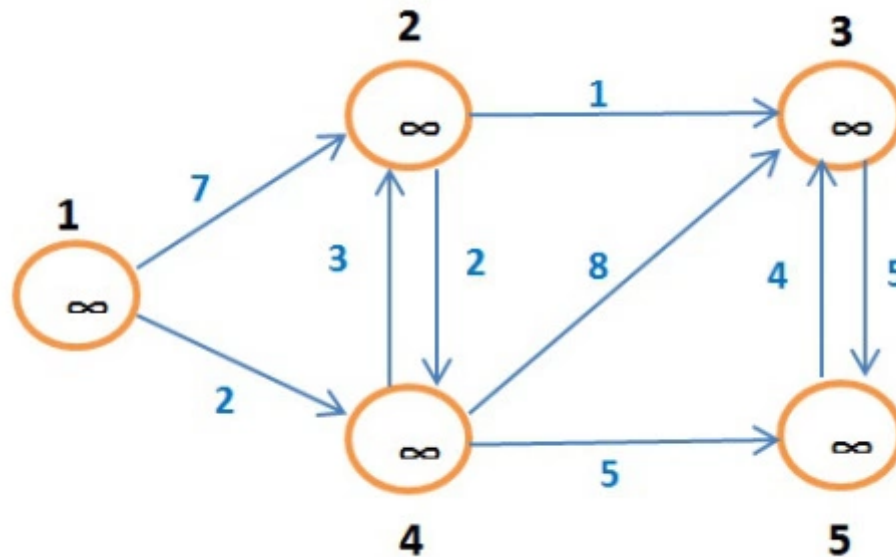
1  método Dijkstra( Grafo, origen ):
2      creamos una cola de prioridad Q
3      agregamos origen a la cola de prioridad Q
4      mientras Q no este vacío:
5          sacamos un elemento de la cola Q llamado u
6          si u ya fue visitado continuo sacando elementos de Q
7          marcamos como visitado u
8          para cada vértice v adyacente a u en el Grafo:
9              sea w el peso entre vértices ( u , v )
10             si v no ah sido visitado:
11                 Relajacion( u , v , w )

1  método Relajacion( actual , adyacente , peso ):
2      si distancia[ actual ] + peso < distancia[ adyacente ]
3          distancia[ adyacente ] = distancia[ actual ] + peso
4          agregamos adyacente a la cola de prioridad Q

```

Ejemplo y Código paso a paso

Tengamos el siguiente grafo, cuyos ID están en color negro encima de cada vértice, los pesos esta en color azul y la distancia inicial en cada vértice es infinito



Algunas consideraciones para entender el código que se explicara junto con el funcionamiento del algoritmo.

first , second) donde first es el vertice adyacente y second el peso de la arista
 distancia infinita inicial, basta conque sea superior al maximo valor del peso en alguna de las aristas

Inicializamos los valores de nuestros arreglos

Vértices	1	2	3	4	5
Distancia[u]	∞	∞	∞	∞	∞
Visitado[u]	0	0	0	0	0
Previo[u]	-1	-1	-1	-1	-1

Donde:

- **Vértices:** ID de los vértices.
- **Distancia[u]:** Distancia mas corta desde vértice inicial a vértice con ID = u.
- **Visitado[u]:** 0 si el vértice con ID = u no fue visitado y 1 si ya fue visitado.
- **Previo[u]:** Almacenara el ID del vértice anterior al vértice con ID = u, me servirá para impresión del camino mas corto.

En cuanto al código los declaramos de la siguiente manera:

```

1  vector< Node > ady[ MAX ]; //lista de adyacencia
2  int distancia[ MAX ];      //distancia[ u ] distancia de vértice inicial a vértice con ID = u
3  bool visitado[ MAX ];      //para vértices visitados
4  int previo[ MAX ];         //para la impresion de caminos
5  priority_queue< Node , vector<Node> , cmp > Q; //priority queue propia del c++, usamos el comparado
6  int V;                     //numero de vertices

```

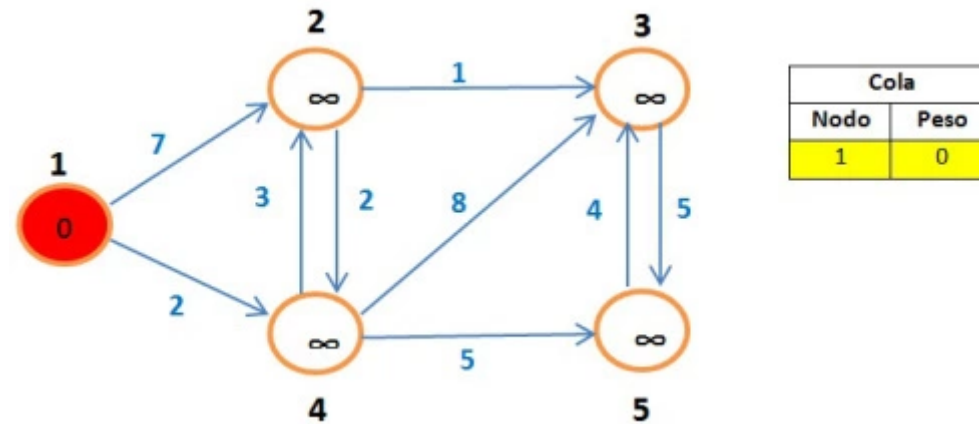
Y la función de inicialización será simplemente lo siguiente

```

1  //función de inicialización
2  void init(){
3      for( int i = 0 ; i <= V ; ++i ){
4          distancia[ i ] = INF; //inicializamos todas las distancias con valor infinito
5          visitado[ i ] = false; //inicializamos todos los vértices como no visitado
6          previo[ i ] = -1;      //inicializamos el previo del vértice i con -1
7      }
8  }

```

De acuerdo al vértice inicial que elijamos cambiara la distancia inicial, por ejemplo la ruta más corta partiendo del vértice 1 a todos los demás vértices:



El vértice 1 es visitado, la distancia de vértice 1 -> vértice 1 es 0 por estar en el mismo lugar.

```
1 | Q.push( Node( inicial , 0 ) ); //Insertamos el vértice inicial en la Cola de Prioridad
2 | distancia[ inicial ] = 0;      //Este paso es importante, inicializamos la distancia del inicial co
```

Extraemos el tope de la cola de prioridad

```
( ) {
= Q.top().first; //Mientras cola no este vacia
;               //Obtengo de la cola el nodo con menor peso, en un comienzo será el inicial
;               //Sacamos el elemento de la cola
```

Si el tope ya fue visitado entonces no tengo necesidad de evaluarlo, por ello continuaría extrayendo elementos de la cola:

```
actual ] ) continue; //Si el vértice actual ya fue visitado entonces sigo sacando elementos de la cola
```

En este caso al ser el tope el inicial no está visitado por lo tanto marcamos como visitado.

```
1 | visitado[ actual ] = true; //Marco como visitado el vértice actual
```

Hasta este momento la tabla quedaría de la siguiente manera

Vértices	1	2	3	4	5
Distancia[u]	0	∞	∞	∞	∞
Visitado[u]	1	0	0	0	0
Previo[u]	-1	-1	-1	-1	-1

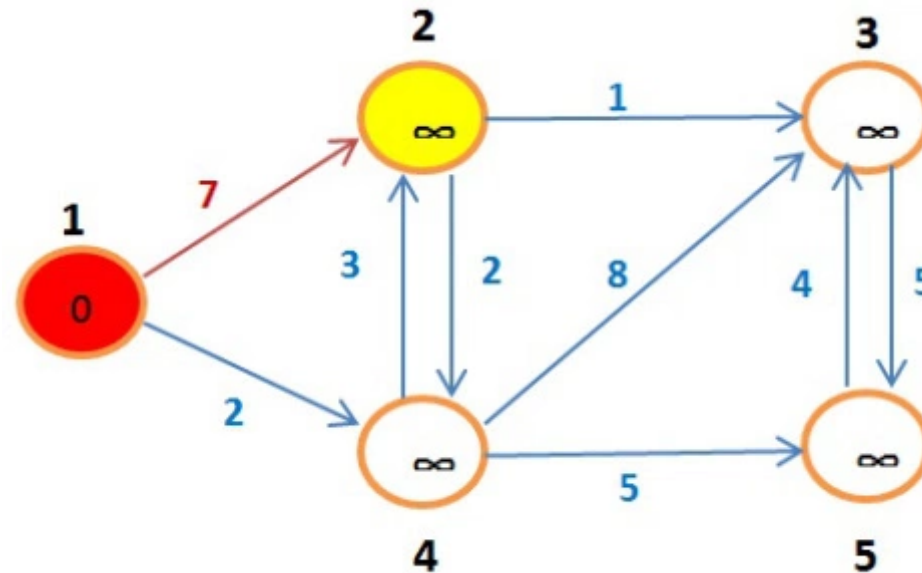
Ahora vemos sus adyacentes que no hayan sido visitados. Tendríamos 2 y 4.

```

0 ; i < ady[ actual ].size() ; ++i ){ //reviso sus adyacentes del vertice actual
:= ady[ actual ][ i ].first; //id del vertice adyacente
ly[ actual ][ i ].second; //peso de la arista que une actual con adyacente ( actual , adyacente )
tado[ adyacente ] ){ //si el vertice adyacente no fue visitado

```

Evaluamos primero para vértice 2

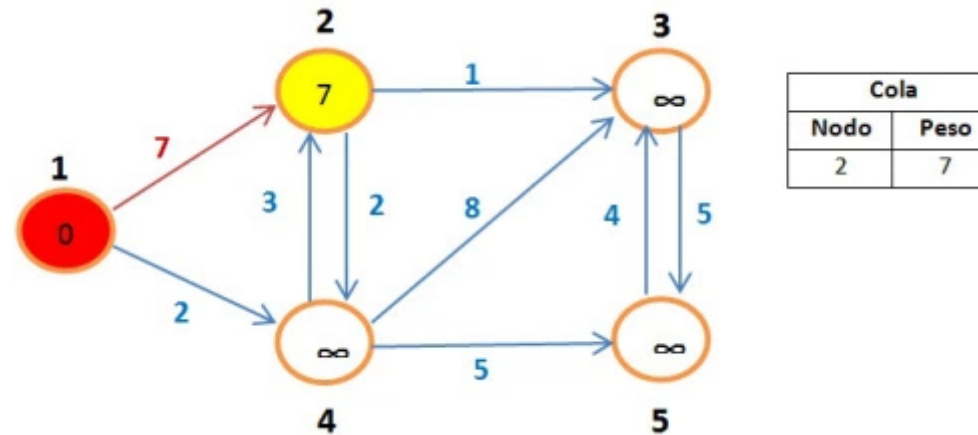


Vemos que la distancia actual desde el vértice inicial a 2 es ∞ , verifiquemos el paso de relajación:

$$\text{distancia}[1] + 7 < \text{distancia}[2] \quad \rightarrow \quad 0 + 7 < \infty \quad \rightarrow \quad 7 < \infty$$

El paso de relajación es posible realizarlo entonces actualizamos la distancia en el vértice 2 y agregando el vértice en la cola de prioridad con nueva distancia quedando:

Vértices	1	2	3	4	5
Distancia[u]	0	7	∞	∞	∞
Visitado[u]	1	0	0	0	0
Previo[u]	-1	1	-1	-1	-1



El paso de relajación se daría en la siguiente parte:

```

1  for( int i = 0 ; i < ady[ actual ].size() ; ++i ){ //reviso sus adyacentes del vertice actual
2      adyacente = ady[ actual ][ i ].first;    //id del vertice adyacente
3      peso = ady[ actual ][ i ].second;        //peso de la arista que une actual con adyacente ( act
4      if( !visitado[ adyacente ] ){           //si el vertice adyacente no fue visitado
5          relajacion( actual , adyacente , peso ); //realizamos el paso de relajacion
6      }
7  }
```

Donde la función de relajación sería

```

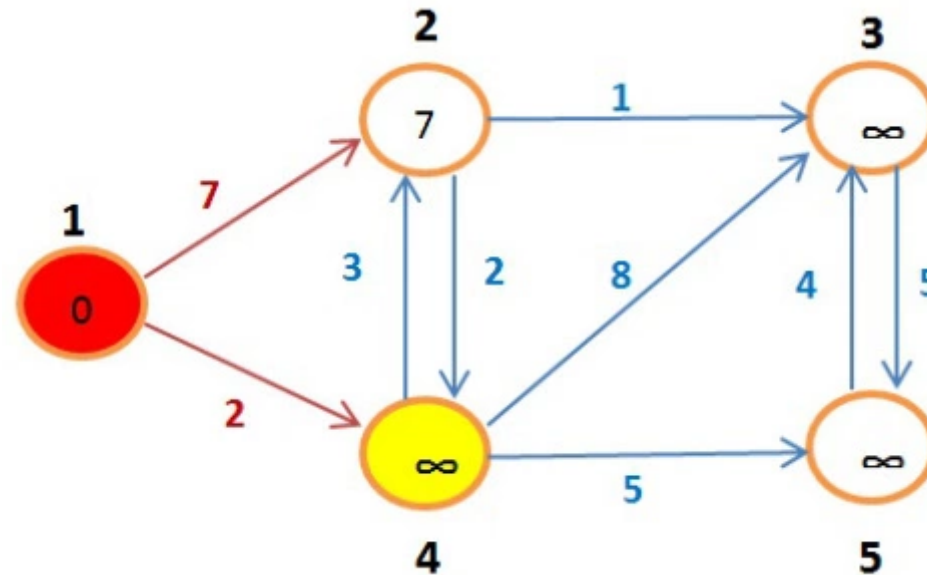
1  //Paso de relajacion
2  void relajacion( int actual , int adyacente , int peso ){
3      //Si la distancia del origen al vertice actual + peso de su arista es menor a la distancia del
4      if( distancia[ actual ] + peso < distancia[ adyacente ] ){
5          distancia[ adyacente ] = distancia[ actual ] + peso; //relajamos el vertice actualizando l
6          previo[ adyacente ] = actual;                        //a su vez actualizamos el vertice pr
```

```

7 |      Q.push( Node( adyacente , distancia[ adyacente ] ) ); //agregamos adyacente a la cola de pr
8 |      }
9 |  }

```

Ahora evaluamos al siguiente adyacente que es el vértice 4:



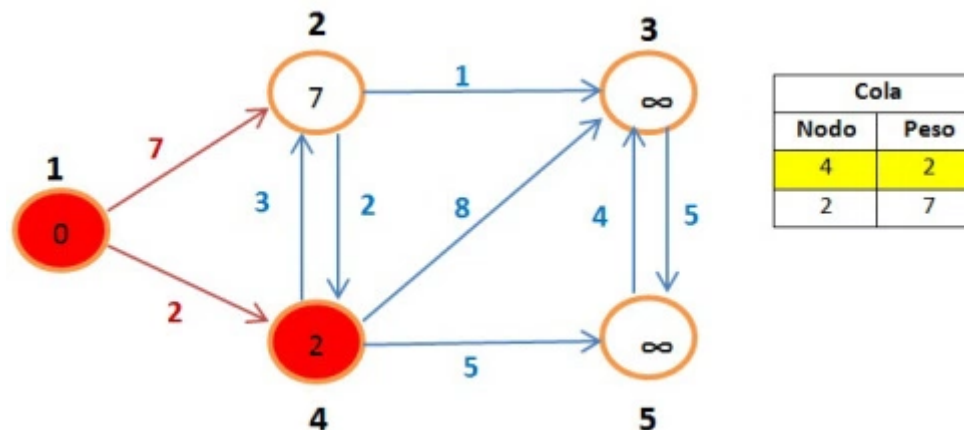
De manera similar al anterior vemos que la distancia actual desde el vértice inicial a 4 es ∞ , verifiquemos el paso de relajación:

$$\text{distancia}[1] + 2 < \text{distancia}[4] \quad \rightarrow \quad 0 + 2 < \infty \quad \rightarrow \quad 2 < \infty$$

El paso de relajación es posible realizarlo entonces actualizamos la distancia en el vértice 4 quedando:

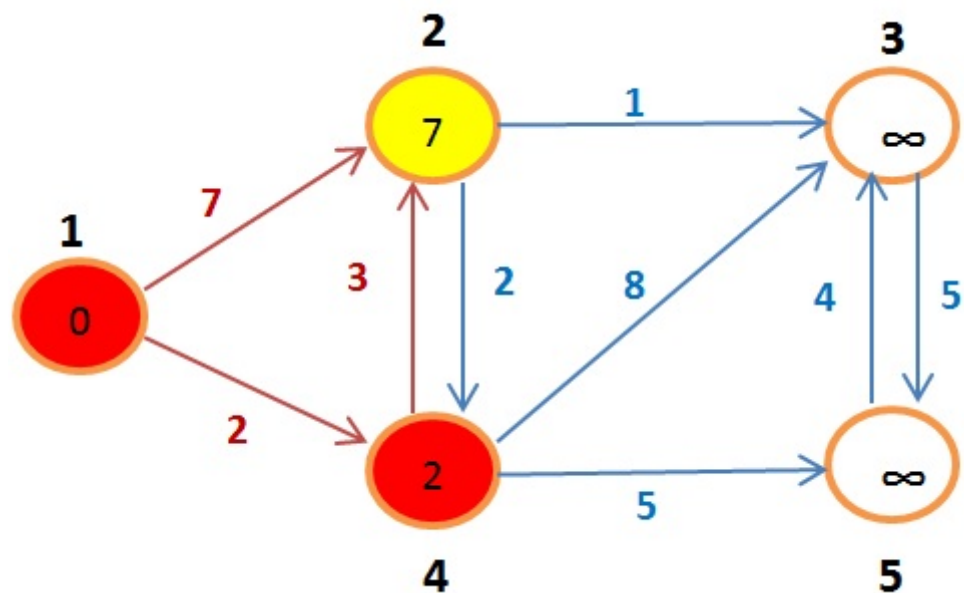
Vértices	1	2	3	4	5
Distancia[u]	0	7	∞	2	∞
Visitado[u]	1	0	0	0	0
Previo[u]	-1	1	-1	1	-1

En cuanto a la cola de prioridad como tenemos un vértice con menor peso este nuevo vértice ira en el tope de la cola:



Revisamos sus adyacentes no visitados que serian vértices 2, 3 y 5.

Empecemos por el vértice 2:



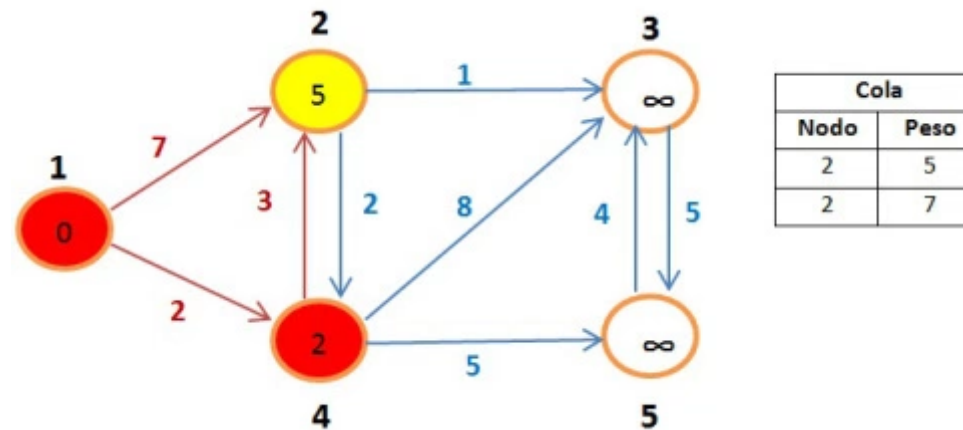
Ahora vemos que la distancia actual del vértice inicial al vértice 2 es 7, verifiquemos el paso de relajación:

$$\text{distancia}[4] + 3 < \text{distancia}[2] \quad \rightarrow \quad 2 + 3 < 7 \quad \rightarrow \quad 5 < 7$$

En esta oportunidad hemos encontrado una ruta mas corta partiendo desde el vértice inicial al vértice 2, actualizamos la distancia en el vértice 2 y actualizamos el vértice previo al actual quedando:

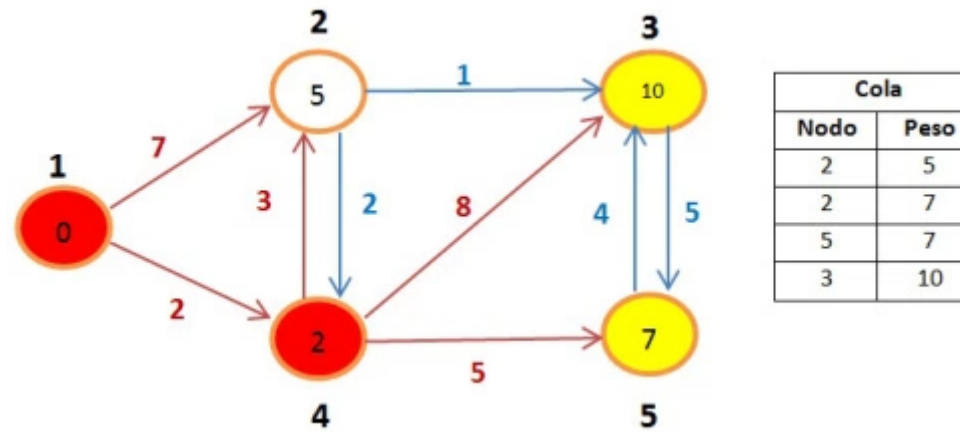
Vértices	1	2	3	4	5
Distancia[u]	0	5	∞	2	∞
Visitado[u]	1	0	0	1	0
Previo[u]	-1	4	-1	1	-1

En cuanto a la cola de prioridad como tenemos un vértice con menor peso este nuevo vértice ira en el tope de la cola, podemos ver que tenemos 2 veces el mismo vértice pero como usamos una técnica greedy siempre usaremos el valor óptimo:

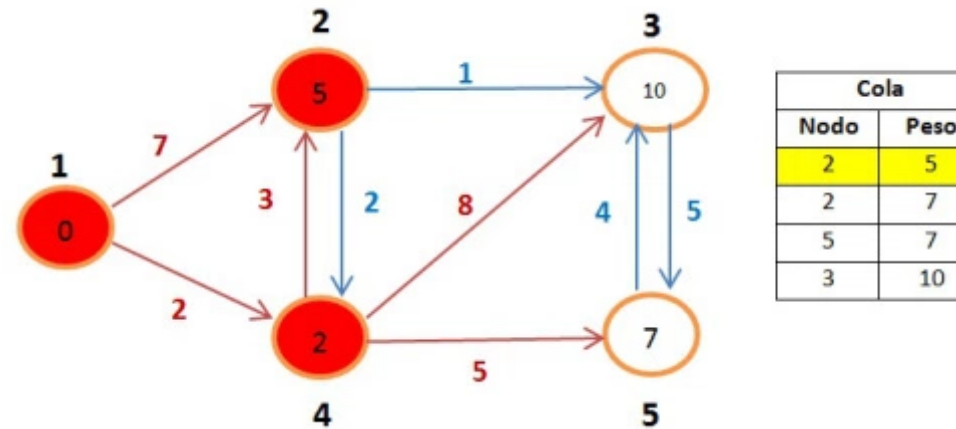


Continuamos con los Vértices 3 y 5 como tienen valor ∞ si será posible relajarlos por lo que sería similar a los pasos iniciales solo que en los pasos iniciales distancia[1] era 0 en este caso distancia[4] es 2, quedando:

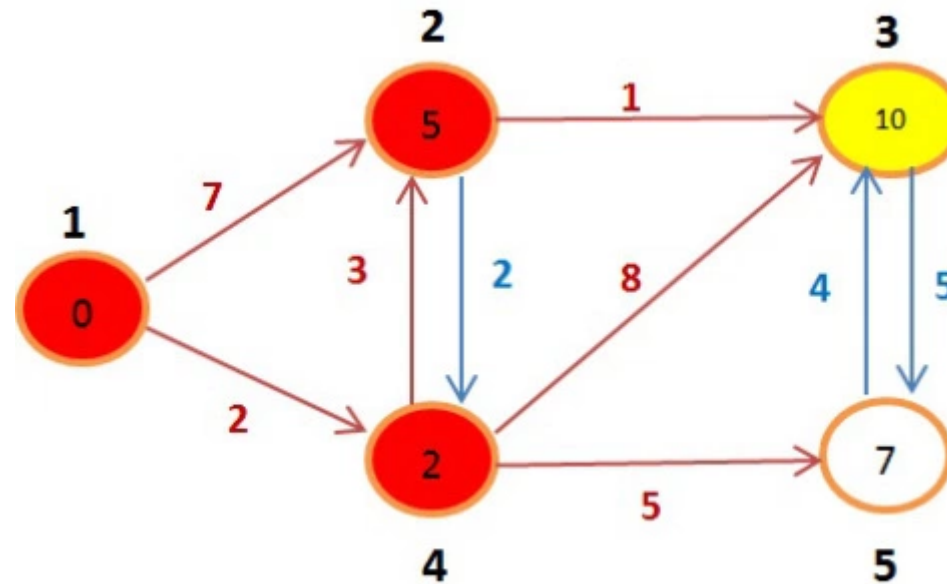
Vértices	1	2	3	4	5
Distancia[u]	0	5	10	2	7
Visitado[u]	1	0	0	1	0
Previo[u]	-1	4	4	1	4



Hemos terminado de evaluar al vértice 4, continuamos con el tope de la cola que es vértice 2, el cual marcamos como visitado.



Los adyacentes no visitados del vértice 2 son los vértices 3 y 5. Comencemos con el vértice 3

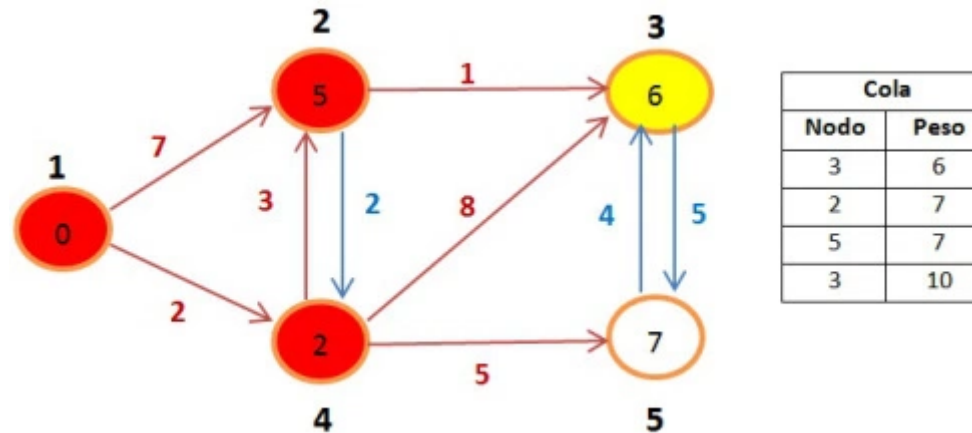


Ahora vemos que la distancia actual del vértice inicial al vértice 3 es 10, verifiquemos el paso de relajación:

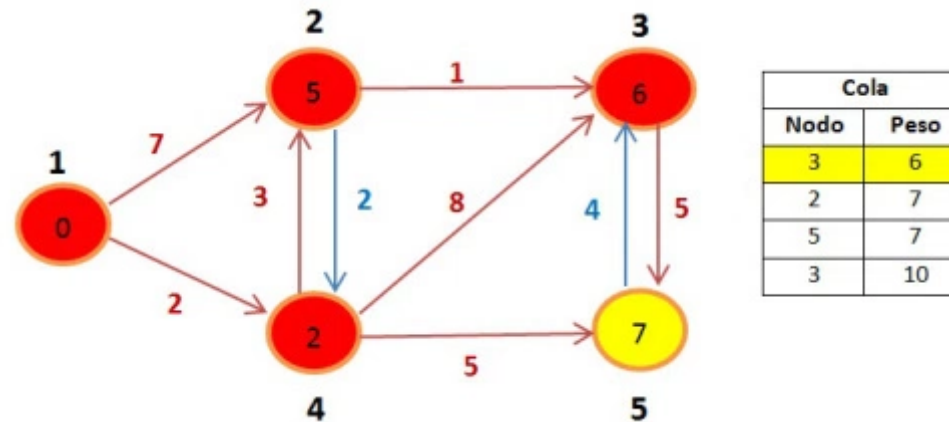
$$\text{distancia}[2] + 1 < \text{distancia}[3] \quad \rightarrow \quad 5 + 1 < 10 \quad \rightarrow \quad 6 < 10$$

En esta oportunidad hemos encontrado una ruta mas corta partiendo desde el vértice inicial al vértice 3, dicha ruta sería 1 -> 4 -> 2 -> 3 cuyo peso es 6 que es mucho menor que la ruta 1 -> 4 -> 3 cuyo peso es 10, actualizamos la distancia en el vértice 3 quedando:

Vértices	1	2	3	4	5
Distancia[u]	0	5	6	2	7
Visitado[u]	1	1	0	1	0
Previo[u]	-1	4	2	1	4



El siguiente vértice de la cola de prioridad es el vértice 3 y su único adyacente no visitado es el vértice 5.



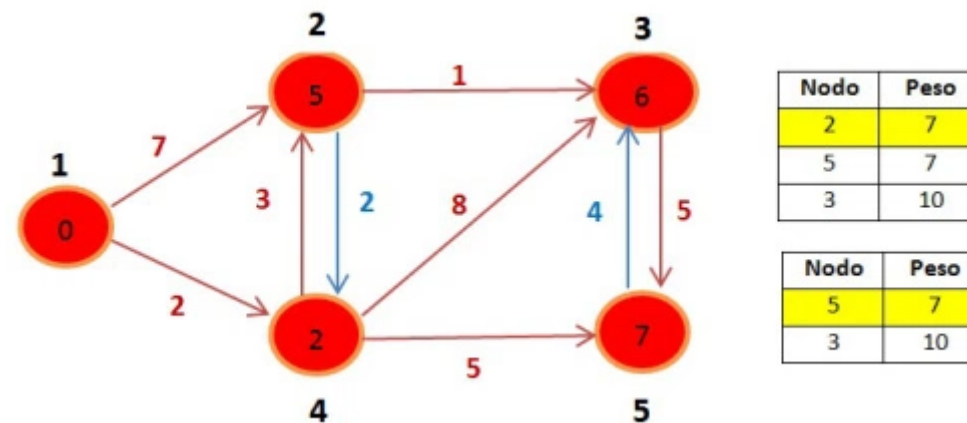
Vemos que la distancia actual del vértice inicial al vértice 5 es 7, verifiquemos el paso de relajación:

$$\text{distancia}[3] + 5 < \text{distancia}[5] \quad \rightarrow \quad 6 + 5 < 7 \quad \rightarrow \quad 11 < 7$$

En esta oportunidad se no cumple por lo que no relajamos el vértice 5, por lo que la tabla en cuanto a distancias no sufre modificaciones y no agregamos vértices a la cola:

Vértices	1	2	3	4	5
Distancia[u]	0	5	6	2	7
Visitado[u]	1	1	1	1	0
Previo[u]	-1	4	2	1	4

Ahora tocaría el vértice 2 pero como ya fue visitado seguimos extrayendo elementos de la cola, el siguiente vértice será el 5.



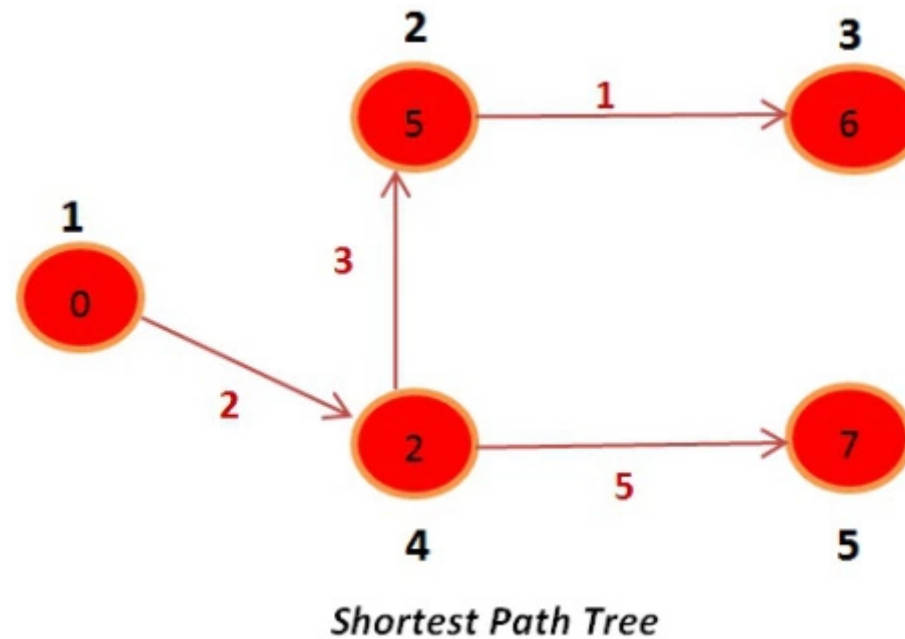
Al ser el ultimo vértice a evaluar no posee adyacentes sin visitar por lo tanto hemos terminado el algoritmo. En el grafico anterior observamos que 2 aristas no fueron usadas para la relajación, las demás si fueron usadas. La tabla final quedaría de la siguiente manera:

Vértices	1	2	3	4	5
Distancia[u]	0	5	6	2	7
Visitado[u]	1	1	1	1	1
Previo[u]	-1	4	2	1	4

De la tabla si deseo saber la distancia mas corta del vértice 1 al vértice 5, solo tengo que acceder al valor del arreglo en su índice respectivo (distancia[5]).

Shortest Path Tree

En el proceso anterior usábamos el arreglo previo[u] para almacenar el ID del vértice previo al vértice con ID = u, ello me sirve para formar el árbol de la ruta mas corta y además me sirve para imprimir caminos de la ruta mas corta.



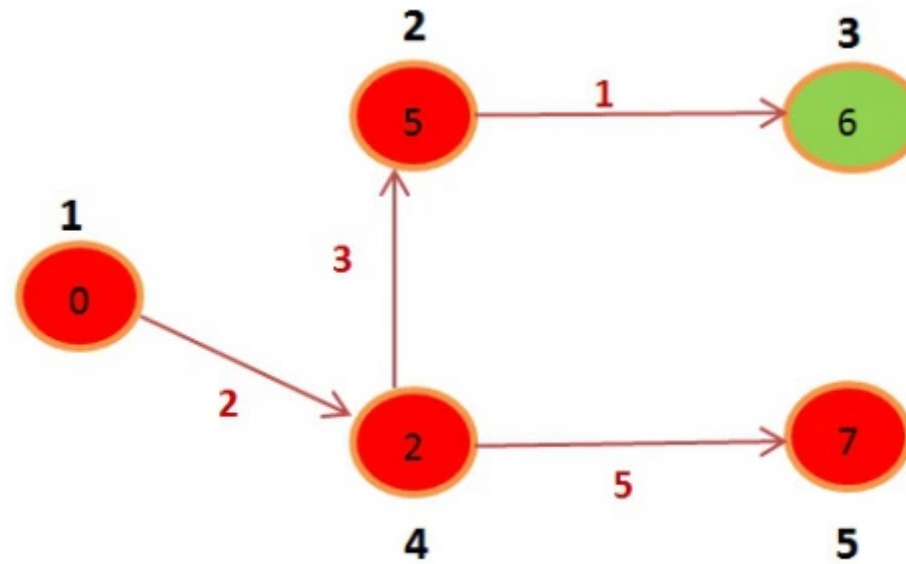
Impresión del camino encontrado.

Para imprimir el camino mas corto deseado usamos el arreglo `previo[u]`, donde `u` tendrá el ID del vértice destino, o sea si quiero imprimir el camino mas corto desde vértice 1 -> vértice 3 partiré desde `previo[3]` hasta el `previo[1]`. De manera similar a lo que se explico en el algoritmo BFS, en este caso se realizara de manera recursiva:

```

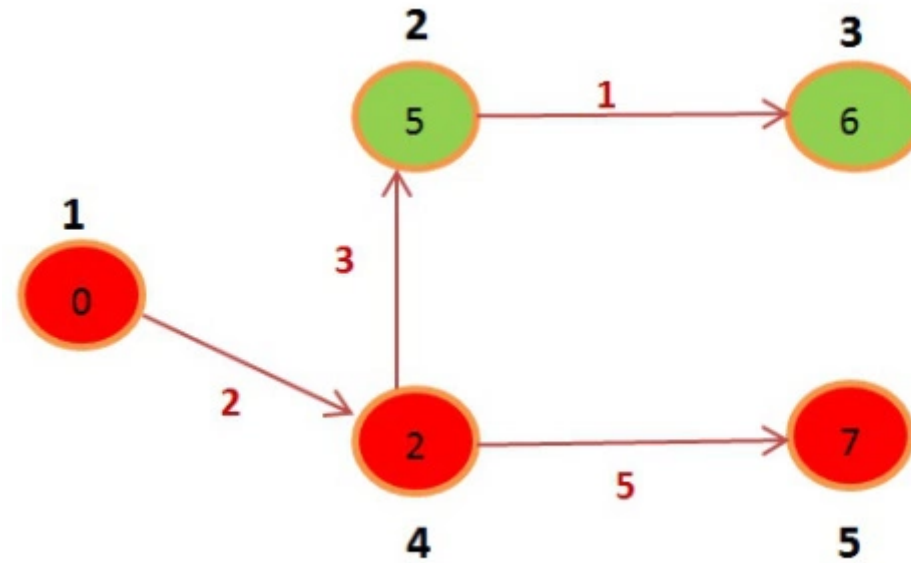
1 //Impresion del camino mas corto desde el vertice inicial y final ingresados
2 void print( int destino ){
3     if( previo[ destino ] != -1 )    //si aun poseo un vertice previo
4         print( previo[ destino ] ); //recursivamente sigo explorando
5     printf( "%d " , destino );      //terminada la recursion imprimo los vertices recorridos
6 }
```

Veamos gráficamente el funcionamiento, desde el grafo comenzamos en 3



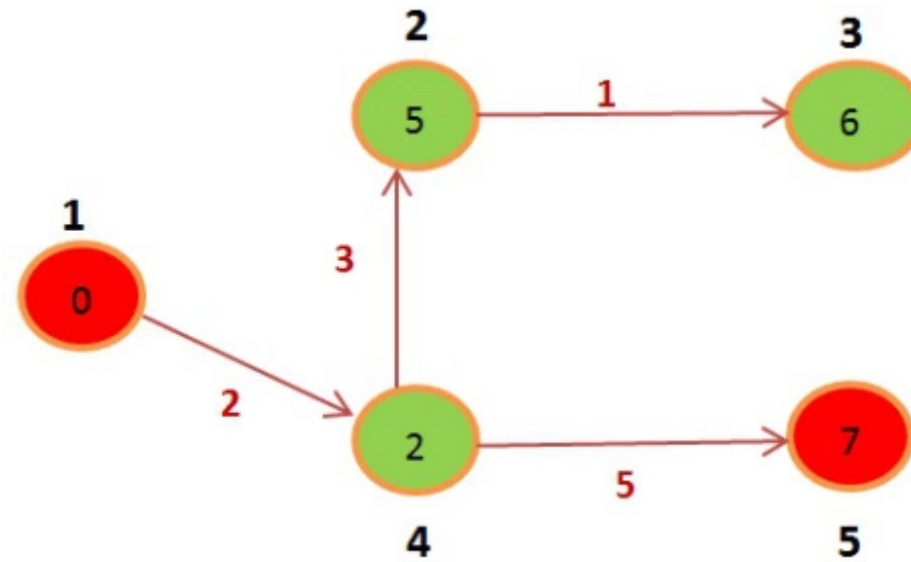
Vértices	1	2	3	4	5
Previo[u]	-1	4	2	1	4

El previo de 3 es el vértice 2, por lo tanto ahora evaluó 2:



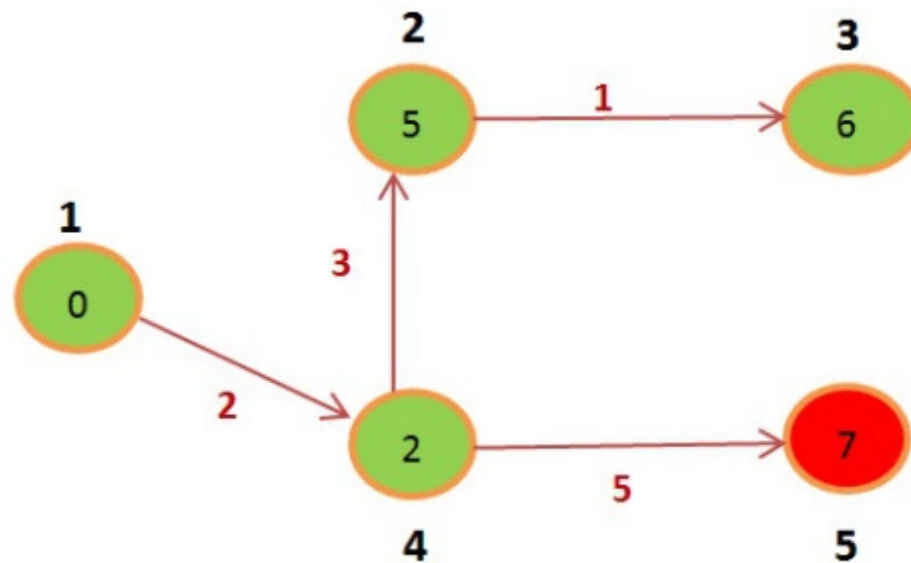
Vértices	1	2	3	4	5
Previo[u]	-1	4	2	1	4

Ahora el previo de 2 es el vértice 4:



Vértices	1	2	3	4	5
Previo[u]	-1	4	2	1	4

El previo de 4 es el vértice inicial 1



Vértices	1	2	3	4	5
Previo[u]	-1	4	2	1	4

Como el previo de 1 es -1 terminamos el recorrido, ahora en el retorno de las llamadas recursivas imprimo el camino: 1 4 2 3

Problemas de diferentes Jueces

UVA

[341 – Non-Stop Travel](#)

[423 – MPI Maelstrom](#)

[10278 – Fire Station](#)

[10801 – Lift Hopping](#)

[12144 – Almost Shortest Path](#)

TJU

[1325 – Fire Station](#)

SPOJ

[15 – The Shortest Path \(SHPATH\)](#)

[50 – Invitation Cards \(INCARDS\)](#)

[440 – The Turtle's Shortest Path \(TSPATH\)](#)

[3405 – Almost Shortest Path \(SAMERo8A\)](#)

POJ

[1511 – Invitation Cards](#)

HDU

[1535 – Invitation Cards](#)

ICPC

[4210 – Almost Shortest Path](#)

COJ

[1659 – Checking an Alibi](#)





Códigos:

Implementación del algoritmo en C++: [Algoritmo de Dijkstra](#)

Implementación del algoritmo en JAVA: [Algoritmo de Dijkstra](#)

Por Jhosimar George Arias Figueroa

Anuncios

	
3,45 \$	28,59 \$
	
32,46 \$	3,89 \$

GearBest

	
3,45 \$	28,59 \$
	
32,46 \$	3,89 \$

GearBest

SHARE THIS:



3 bloggers like this.

Esta entrada fue publicada en [Algorithms](#), [Main](#) y etiquetada [dijkstra](#), [graph](#), [shortest path](#), [sssp](#). Guarda el [enlace permanente](#).

59 RESPUESTAS A “CAMINO MAS CORTO: ALGORITMO DE DIJKSTRA”

Horacio | [septiembre 7, 2012 en 6:38 am](#) | [Responder](#)

Muy bueno, lo mejor que encontré sobre dijkstra, gracias



Misael | [octubre 28, 2012 en 3:56 pm](#) | [Responder](#)

Excelente información, todo muy claro y con mucha pedagogía



omar | [abril 6, 2013 en 2:13 am](#) | [Responder](#)

hola pero como se corre el programa se cicla, alguien me puede decir lo mas pronto posible, gracias



carlos andres (@andres8822) | [mayo 14, 2013 en 2:28 pm](#) | [Responder](#)

aqui les dejo una aplicacion echa en java gracias jarias

<http://andres2288.webcindario.com/mis%20jar/rutasaereas/airport.zip>



pfallasro | [julio 14, 2013 en 3:02 pm](#) | [Responder](#)

Buenisimo!



Luis | [noviembre 30, 2013 en 2:00 am](#) | [Responder](#)

Disculpa, què es el cmp que esta incluido en la priority_queue?



Luis | [noviembre 30, 2013 en 3:20 am](#) | [Responder](#)

Disculpa

Lo de la priority_queue ya lo resolvi, pero como llenas la lista de adyacencia

vector ady[Max]; ???

se llena sola o como...? es la parte que no entiendo.

Saludos! (:



jariasf | [noviembre 30, 2013 en 10:31 am](#) | [Responder](#)

En el codigo puedes ver como es el llenado, pero basicamente cuando tengas un vertice A y B y deseas agregar una arista entre ambos, si usas listas de adyacencia como es este caso entonces seria:

```
ady[ A ].push_back( B ); // A -> B
```

Ahora eso es para grafos dirigidos, si deseas no dirigidos entonces agregas una arista adicional de B hacia A:

```
ady[ B ].push_back( A ); // B -> A
```

En este caso tenemos un valor adicional que es el peso de la arista entonces mediante una estructura puedes almacenar tanto el vertice destino como el peso:

```
ady[ A ].push_back( Nodo( B , W ) ); // A -> B con peso igual a W
```

```
ady[ B ].push_back( Nodo( A , W ) ); // B -> A con peso igual a W
```



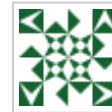
En otros problemas puede ser que requieras mas valores adicionales, esos los agregas a la estructura y funcionaria de forma similar. Y asi es como se forma el grafo con listas de adyacencia para aplicar los algoritmos.

[José Luis Uc Lozano](#) | diciembre 3, 2013 en 11:20 am | [Responder](#)



Hola, tengo una duda, cuando corro el programa en dev c++, la pantalla me aparece en negro, supongo que debo insertar los valores de de los nodos y el de los pesos, pero no se en que orden, me podrias auxiliar, gracias y buen aporte.

[jariasf](#) | diciembre 3, 2013 en 11:54 am | [Responder](#)



En la parte superior del codigo hay ejemplo de ingreso comentado, pero bueno la entrada se da asi:

NumeroVertices NumeroAristas

Luego acorde al NumeroAristas se ingresa:

VerticeOrigen VerticeDestino Peso

....

Finalmente:

VerticeInicial que es con el cual se comenzara el algoritmo de Dijkstra.

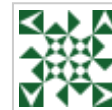
VerticeFinal para imprimir los vertices que conforman la ruta mas corta entre VerticeInicial y VerticeFinal.

[José Luis Uc Lozano](#) | diciembre 3, 2013 en 12:36 pm | [Responder](#)



Ohhh, y bueno, tengo una duda mas, si quisiera añadir una gran cantidad de datos, como por ejemplo, la ruta mas corta entre las estaciones del metro de una ciudad, como meteria esos datos? (tanto las estaciones de la linea como las distancias entre las estaciones del metro), tengo la idea de meter los datos en un archivo de tipo .TXT o en un lista en excel, pero eso como lo podria meter para que me lo leyera el programa, gracias 😊

[jariasf](#) | diciembre 3, 2013 en 12:57 pm | [Responder](#)



Puede ser un .txt, la lectura de un archivo encuentras en internet pero por ejemplo puedes poner algo como esto:

C/C++ -> freopen("input.txt", "r", stdin);

Java -> BufferedReader br = new BufferedReader(new InputStreamReader(new FileInputStream("input.txt")));

en el archivo input.txt pondrias los valores con el formato que te mencione, para el caso de Java tendrias que hacer modificaciones al codigo para que reciba la entrada correctamente, en el caso de C++ no hay cambios que hacer en la entrada. Ahora si son ciudades las distancias no seran enteros sino valores decimales, tendrias que cambiar el tipo de dato para el peso. Y si vas a ingresar nombres de ciudades por ejemplo: ciudadA ciudadB Peso donde ciudadA y

ciudadB son cadenas entonces podrias hacer un mapeo de esos valores, reemplazandolos con numeros y despues aplicar el algoritmo.

[José Luis Uc Lozano](#) | diciembre 3, 2013 en 1:30 pm |

Gracias, pensaba hacer eso, solo necesitaba que alguien me lo confirmara.



uxmar | enero 20, 2014 en 9:48 pm | [Responder](#)

Hola, gracias por el post. Una pregunta, como agrego los datos de las aristas y los vértices al código de java que colocaste?



[jariasf](#) | enero 20, 2014 en 9:52 pm | [Responder](#)

Es similar que el codigo de C++. En la parte superior del codigo hay ejemplo de ingreso comentado. La entrada se da asi:

NumeroVertices NumeroAristas

Luego acorde al NumeroAristas se ingresa:

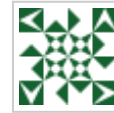
VerticeOrigen VerticeDestino Peso

....

Finalmente:

VerticeInicial que es con el cual se comenzara el algoritmo de Dijkstra.

VerticeFinal para imprimir los vertices que conforman la ruta mas corta entre VerticeInicial y VerticeFinal.



uxmar | enero 20, 2014 en 10:07 pm |

Lo siento, pero no comprendo del todo, debido a que soy nueva en java y quiero implementar este algoritmo. Me podrías decir como se debe agregar directamente en el código y a que nivel del código debo agregarlo. Muchas gracias por tu ayuda, realmente la necesito.



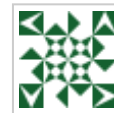
[jariasf](#) | enero 20, 2014 en 10:25 pm |

Ejecuta el codigo en un IDE o mediante comandos eh ingresa lo que esta comentado, puedes hacer un copy/paste:

5 9

1 2 7

1 4 2




```

2 3 1
2 4 2
3 5 4
4 2 3
4 3 8
4 5 5
5 3 5
1 3

```

uxmar | [enero 21, 2014 en 12:32 am](#) | [Responder](#)

Excelente! Ya lo hice, muchas gracias por tu valiosa ayuda.



uxmar | [marzo 18, 2014 en 6:55 pm](#) | [Responder](#)

Hola jariasf, estuve estudiando detenidamente tu código y me surgen las siguientes preguntas:



- 1.- Creo saber la respuesta pero prefiero preguntar al experto. En lugar de introducir los nodos como números se podrían utilizar letras?
- 2.- Cómo se podría hacer para imprimir las rutas posibles para llegar por ejemplo del nodo 1 al nodo 5, sin importar cual es la más corta, solo las rutas posibles.

Muchas Gracias por tu atención.

jariasf | [marzo 18, 2014 en 9:03 pm](#) | [Responder](#)

1. Si es posible incluso nombres pero tienes que mapearlos, puedes usar un map en C++ o Map en JAVA. Ingresarias los Strings y le pondrias un valor entero a cada cadena y con esos valores hacer es Dijkstra normal.



2. Tambien es posible, pero en ese caso es mejor usar otro algoritmo como este -> <https://jariasf.wordpress.com/2012/03/02/algoritmo-de-busqueda-depth-first-search-parte-1/> que seria como un backtracking para mostrar las rutas posibles. Claro que con backtracking se hace mas lento por la complejidad que conlleva.

uxmar | [marzo 18, 2014 en 9:23 pm](#) | [Responder](#)

Excelente, muchas gracias jariasf.



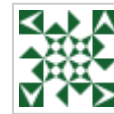
Yo estoy trabajando en java porque estoy intentando hacer una app libre que proporcionará rutas de trenes, es mi 1era app, estoy aprendiendo aun, es por eso que estoy investigando estos algoritmos y es por esto que ya leí y entendí cada línea de tu código.

- 1.- Acabo de ver el link sobre DEPTH FIRST SEARCH, mi pregunta es, estará disponible en java?
- 2.- Buscando de no re-inventar la rueda, sabes de algún algoritmo de estos que ya este adaptado para android?

jariasf | [marzo 18, 2014 en 9:49 pm](#) | [Responder](#)

1. Por el momento no esta en Java pero no es muy complicado pasarlo de C++ a Java. Es mas si entendiste el Dijkstra entonces podras hacer el DFS(Depth First Search) en Java. Tomalo como para practicar tambien 😊
2. Google Maps es lo mas conocido y usado.

Dependiendo de la cantidad de datos que manejes es lo que definira tu algoritmo. Si con Dijkstra resulta que obtienes resultados lentos, puedes usar otro algoritmo llamado A*(es bastante similar a Dijkstra) que usa heurísticas. El problema con el A* es que al usar heurísticas no te garantiza la solución óptima siempre sino un aproximado en algunos casos. Lo de hallar todas las rutas posibles es un tema mas complejo si tienes bastantes datos.



uxmar | [marzo 19, 2014 en 6:57 pm](#) | [Responder](#)

Ok jariasf muy agradecida con tu ayuda.



Procederé a analizar los datos para definir entonces el método a usar.

Gracias de nuevo.

Ted Carrasco | [abril 20, 2014 en 11:12 pm](#) | [Responder](#)

una pregunta en un caso específico copiando tu entrada de datos pero en vez de poner como vertice inicial el 1 puse el 2 y se supone q el camino mas corto al vertice 5 del 2 es 6 pero tu programa de 5, podrias explicarme porq?? gracias de antemano



[Ted Carrasco](#) | abril 20, 2014 en 11:15 pm | [Responder](#)

en si comenzando del vertice 2 (v inicial) si quiero viajar al vertice 5 usando tu entrada de datos(cambiando el vertice inicial) la respuesta tendria q dar 6 por q va al vertice 3 y luego al vertice 5 sumando $1 + 5 = 6$, pero el programa me da 5, es un error de comprension, podrias explicarme porfavor?



[Ted Carrasco](#) | abril 20, 2014 en 11:49 pm |

ya se cual es el error q la entrada de datos no es la misma q la de tu ejemplo por eso me confundi, buenísimo tu tutorial y gracias



java | mayo 6, 2014 en 1:53 am | [Responder](#)

hola jariasf esta muy bueno tu programa pero cuando lo corro en el modo de grafo no dirigido no entiendo muy bien como obtiene los caminos mas cortos con el ejemplo que pones de tus datos me da esto



Ingrese el numero de vertices: 5

Ingrese el numero de aristas: 9

1 2 7

1 4 2

2 3 1

2 4 2

3 5 4

4 2 3

4 3 8

4 5 5

5 3 5

Ingrese el vertice inicial: 1

Distancias mas cortas iniciando en vertice 1

Vertice 1 , distancia mas corta = 0

Vertice 2 , distancia mas corta = 1073741824

Vertice 3 , distancia mas corta = 1073741824

Vertice 4 , distancia mas corta = 1073741824

Vertice 5 , distancia mas corta = 1073741824

[jariasf](#) | mayo 6, 2014 en 10:58 am | [Responder](#)



Hola, un grafo no dirigido es aquel donde existe arista de A -> B asi como de B -> A entonces estas dos lineas de code deben estar presentes:

```
ady[ origen ].push_back( Node( destino , peso ) );
ady[ destino ].push_back( Node( origen , peso ) );
```



java | [mayo 6, 2014 en 12:34 pm](#) | [Responder](#)

ok ya lo hize si me resulto gracias

otra pregunta

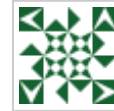
si quiero añadir la matriz de adyascencia para poder imprimirla tendría que implementarla yo o ya en el programa hay un método que la realiza?

gracias por la ayuda



jariasf | [mayo 6, 2014 en 1:14 pm](#) | [Responder](#)

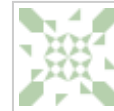
El codigo usa Listas de Adyacencia, si deseas usar Matriz de Adyacencia tendrias que hacer cambios al codigo.



Juan | [junio 14, 2014 en 11:01 am](#) | [Responder](#)

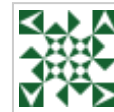
Hola, gracias por tus explicaciones, es un excelente post.

Quería saber si para el ejemplo concreto de recorrer una intersección de ciudades por tramos (CiudadDestino,CiudadOrigen) usando una matriz en java, este método me sirve?



jariasf | [junio 15, 2014 en 12:56 pm](#) | [Responder](#)

Este algoritmo sirve para hallar la ruta mas corta de un vertice a todos los demas, usando matriz de adyancencia tambien funciona. Si solo deseas recorrer un grafo puedes usar un algoritmos de busqueda: BFS o DFS.



Luis | [agosto 26, 2014 en 2:32 pm](#) | [Responder](#)

hola al copiar el codigo en netbeans ide8 y al ejecutarlo sale este error:

java.lang.ExceptionInInitializerError

Caused by: java.lang.RuntimeException: Uncompilable source code – cannot find symbol

symbol: class Scanner

location: class dijkstra.Dijkstra

at dijkstra.Dijkstra.(Dijkstra.java:38)

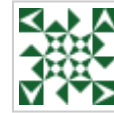


Exception in thread "main" Java Result: 1
BUILD SUCCESSFUL (total time: 0 seconds)

jariasf | agosto 26, 2014 en 9:10 pm | Responder

No tengo Netbeans, pero el código ejecuta bien, de todos modos prueba con esta versión:

<https://github.com/jariasf/Online-Judges-Solutions/blob/master/Algorithms/JAVA/Graph%20Theory/DijkstraAlgorithm.java>



Eliana | septiembre 25, 2014 en 3:13 pm | Responder

estoy mirando tu material y esta muy interesante, yo debo hacer una tarea que es realizar un programa en C++ para ingresar los datos de varios cines y que el usuario pueda hacer sus compras de entradas por internet pero que a su vez se le ofrezca el cine mas cercano. como hago no se ni por donde empezar



Christian Avalos | octubre 31, 2014 en 1:37 pm | Responder

Men yo lo ejecuto en c++ y me sale un error en la biblioteca que es para grafos esta—> #include "grafo_heap.h" porque?? sera que esta mal escrita o le falta otra biblioteca para para trabajar con grafos?? porfavor necesito tu ayuda urgente antes de las 6 de la tarde de hoy. gracias!!



jariasf | octubre 31, 2014 en 1:55 pm | Responder

grafo_heap.h? ese error debe ser de tu biblioteca. El código que esta para descargar funciona correctamente, si vas agregar otras cosas, agregalas sobre ese código.



Christian Avalos | octubre 31, 2014 en 2:04 pm |

que tal men?? lo que pasa esque probe tu código y compila bien pero no muestra nada en pantalla pues algun resultado, no me pide nada para ingresaaar tampoko para que haga el calculo y busque otro código y confundí con el tuyo, y ya que me respondiste queria aprovechar para pedirte ayuda y que me digas por favor como hago para que tome datos del usuario y que encontrado el camino mas corto, me muestre en pantalla??? por favor si no es mucha molestia!!



mego | junio 13, 2015 en 3:41 pm | Responder



Hola buenas tardes, yo lo unik que quisiera ers saber como obtengo estos numeros

1 2 7

1 4 2

2 3 1

2 4 2

3 5 4

4 2 3

4 3 8

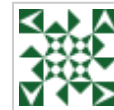
4 5 5

5 3 5

si es aleatorio o como los sacos, ya el programa me corre pero esa es mi gran duda, lo que pasa es q mi programa es sobre un cartero q tiene q recorrer uvarias distancias y pasar por la misma calle

[jariasf](#) | junio 13, 2015 en 9:14 pm | [Responder](#)

Hola, son aristas con la forma -> (vertice inicial, vertice final, peso de arista). Por ejemplo 1 2 7 quiere decir que existe una arista entre los vertices 1 y 2 con peso 7. Dependiendo de la implementacion puede ser una arista en una direccion o en dos direcciones dependiendo si el grafo es dirigido, no dirigido o que tenga ambos tipos de aristas.



[meo](#) | junio 16, 2015 en 3:27 pm |

m... ok, muchas gracias, y el proyecto q no mandaron a realizar esde un cartero que tiene q recorrer varias calles sin pasar por el mismo vertice, es ciclo euleriano



[jariasf](#) | junio 19, 2015 en 8:21 am |

Ok para el ciclo euleriano se hace con DFS, puedes revisar el Algoritmo de Fleury que va eliminando aristas al momento de recorrer el grafo, pero previamente tienes que hacer una revision como si el grafo es conexo, asi como otras propiedades como si el grafo es no dirigido entonces todos los vertices tienen que tener grado par, etc.



[Rey Fernando Salcedo Padilla](#) | octubre 15, 2015 en 12:50 am | [Responder](#)

Aporto una aplicacion en java <http://usandojava.blogspot.com.co/2012/05/implementacion-en-java-del-algoritmo-de.html>



[Jorge Lopez](#) | noviembre 10, 2015 en 4:12 pm | [Responder](#)



Alguien podría subir la solución del ejercicio “50 – Invitation Cards (INCARDS)” del juez SPOJ. Porque la mía me da un error de tiempo excedido.



[Pablo Fonseca](#) | noviembre 24, 2016 en 11:46 pm | [Responder](#)

Hola muy buena la explicacion, una duda con Dijkstra:

En todos los ejemplos que veo, siempre comienza el algoritmo con un vertice con grados de entrada = 0 esto es una condicion obligatoria.

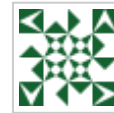
O puedo lanzar el algoritmo desde cualquier vertice???

Gracias



[Jhosimar](#) | noviembre 25, 2016 en 8:39 am | [Responder](#)

Puedes comenzar en cualquier vertice



michelromero | enero 6, 2017 en 1:02 pm | [Responder](#)

No entiendo como hacer el cmp



[eduardo perez](#) | febrero 7, 2017 en 9:50 am | [Responder](#)

cuan eficiente puede ser este grafo de camino cortos(dijkstra) si tengo 10.000 vertices, con relacion de vertice a vertice de por lo menos 3 aristas,seria n cuadrado ? la busqueda ? por que supongo que tendria que ir a buscar a los 10.000 vertices



[Jhosimar](#) | febrero 7, 2017 en 10:10 am | [Responder](#)

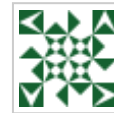
Te refieres a un grafo completo? donde cada vertice esta conectado a otro por una arista. Si ese es el caso entonces tendríamos esto:

La complejidad de Dijkstra es $O(|E| + |V|\log(|V|))$

En grafos completos tenemos $|E| = (|V| * (|V| - 1))/2 \rightarrow O(|V|^2)$

Entonces la complejidad total seria $O(|V|^2 + |V|\log(|V|)) = O(|V|^2)$

Ahora eso de 3 aristas te refieres a que de un vertice a otro existen 3 aristas? si es asi, solo considera la de menor peso y tendrías una sola arista. EL algoritmo no de vera afectado.



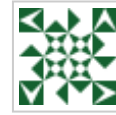
miguel | [marzo 22, 2017 en 10:11 pm](#) | [Responder](#)

el pseudocodigo no funciona correctamente con la linea 10, es incorrecta la condición en ese paso



Jhosimar | [marzo 22, 2017 en 10:17 pm](#) | [Responder](#)

Por que es incorrecto?



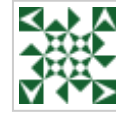
SpiritoftheLaw | [mayo 16, 2017 en 11:25 am](#) | [Responder](#)

Su codigo es excelente, Gracias por compartirlo, Como podria hacer para que desvolviera el peso total recorrido?



Jhosimar | [mayo 16, 2017 en 11:32 am](#) | [Responder](#)

El arreglo `distancia[destino]` contiene la mínima distancia total recorrida hacia el vértice *destino* partiendo de un vértice inicialmente especificado. A eso te refieres con el peso total recorrido?



arteypintura484 | [mayo 24, 2017 en 6:44 pm](#) | [Responder](#)

Buenas tardes, Jhosimar.

Te quería consultar algo. Primeramente decirte que es un excelente código.

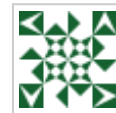
Cómo podría hacer para ponerle una restricción, es decir, que me encuentre el camino mas corto hacia todos los nodos pero, que no supere `X_distancia`, por ejemplo 300 metros. Y claro, los nodos que no entren en ese análisis quedan fuera.



Te agradecería tu respuesta.

Jhosimar | [mayo 24, 2017 en 7:10 pm](#) | [Responder](#)

Te refieres a que la ruta mas corta entre un nodo y cualquier otro nodo no supere la distancia `X`? si es asi, es solo modificar el paso de relajación: `if(distancia[actual] + peso < distancia[adyacente])`. En esa parte adicionarias la condición de que la nueva distancia no sea mayor a `X`.



arteypintura484 | [mayo 24, 2017 en 8:22 pm](#) | [Responder](#)

Si, exactamente. Eso quería intentar.

Si hago tal y como me lo dices, no se modifica nada del enfoque como tal del algoritmo Dijkstra?



[arteypintura484](#) | mayo 24, 2017 en 8:26 pm | [Responder](#)

Lo hago como una nueva condición? o modificando la condición actual que tiene el código?



Matias | agosto 2, 2017 en 8:32 pm | [Responder](#)

Jhosimar, ¿tenés pensado escribir sobre otros algoritmos?, están muy bien explicados. Muchas gracias.



[Jhosimar](#) | agosto 3, 2017 en 9:17 pm | [Responder](#)

la verdad que si me gustaria realizar otros tutoriales, aunque por ahora no creo que sea posible pero estoy viendo que otros algoritmos podria presentar, tu que algoritmos crees sean una buena opción?

