Programación Dinámica: Algoritmo de Floyd para hallar caminos mínimos en un grafo

---

Diseño y Análisis de Algoritmos

Jesús Eduardo Plasencia Pimentel
Pamela Jiménez Rebenaque
Alberto Antonio Sarabia Suarez

1. Introducción.

El objetivo de este informe es comparar los algoritmos Floyd-Warshall y Dijkstra para la resolución del problema del camino mínimo.

Este problema consiste en encontrar un camino entre dos vértices (o nodos) de tal manera que la suma de los pesos de las aristas que lo constituyen es mínima. Un ejemplo de esto es encontrar el camino más rápido para ir de una ciudad a otra en un mapa. En este caso, los vértices representan las ciudades y las aristas las carreteras que las unen, cuya ponderación viene dada por el tiempo que se emplea en atravesarlas.

2. Funcionamiento de Dijkstra.

Algoritmo de Dijkstra. También llamado algoritmo de caminos mínimos, es un algoritmo para la determinación del camino más corto dado un vértice origen al resto de vértices en un grafo con pesos en cada arista. Su nombre se refiere a Edsger Dijkstra, quien lo describió por primera vez en 1959.

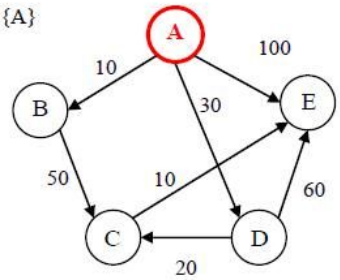
- Es un algoritmo greedy.
- Trabaja por etapas, y toma en cada etapa la mejor solución sin considerar consecuencias futuras.

El funcionamiento de este algoritmo es el siguiente:

- Sea V un conjunto de vértices de un grafo.
- Sea C una matriz de costos de las aristas del grafo, donde en C[u,v] se almacena el costo de la arista entre u y v.
- Sea S un conjunto que contendrá los vértices para los cuales ya se tiene determinado el camino mínimo.
- Sea D un arreglo unidimensional tal que D[v] es el costo del camino mínimo del vértice origen al vértice v.
- Sea P un arreglo unidimensional tal que P[v] es el vértice predecesor de v en el camino mínimo que se tiene construido.
- Sea vinicial el vértice origen. Recordar que el Algoritmo Dijkstra determina los caminos mínimos que existen partiendo de un vértice origen al resto de los vértices.

Paso 1: inicialización.

S = {A}



Matriz de Costos : C

	A	B	C	D	E
A	∞	10	∞	30	100
B	∞	∞	50	∞	∞
C	∞	∞	∞	∞	10
D	∞	∞	20	∞	60
E	∞	∞	∞	∞	∞

`Para cada v ≠ A hacer: D[v] ← C[A, v]` ⟹

D[B]	D[C]	D[D]	D[E]
10	∞	30	100

`Para cada v ≠ A hacer: P[v] ← A`
El predecesor de los vértices de V es A ⟹

P[B]	P[C]	P[D]	P[E]
A	A	A	A

Paso 2: Elegir un vértice w ∈ V - {A} tal que D[w] sea mínimo, y agregar w al conjunto solución S

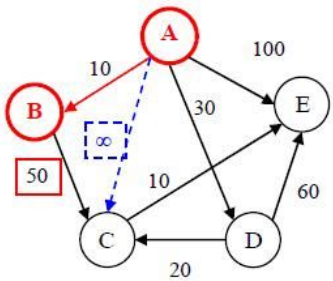
Paso 3: cada v ∈ {C, D, E} hacer D[v] ← min(D[v], D[w]+C[w, v])

Para v = C

```
D[C] ← min (D[C], D[B] + C[B, C])
D[C] ← min (∞, 10 + 50) = 60
```

⟹

D[B]	D[C]	D[D]	D[E]
10	~~∞~~ 60	30	100



Matriz de Costos : C

	A	B	C	D	E
A	∞	10	∞	30	100
B	∞	∞	50	∞	∞
C	∞	∞	∞	∞	10
D	∞	∞	20	∞	60
E	∞	∞	∞	∞	∞

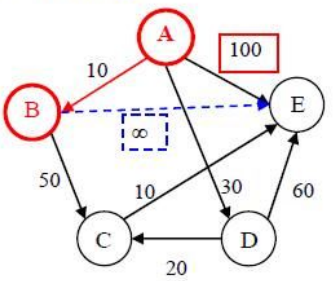
```
P[C] ← B
```
El predecesor de C ahora es B ⟹

P[B]	P[C]	P[D]	P[E]
A	~~A~~ B	A	A

Para v = E

```
D[E] ← min (D[E], D[B] + C[B, E])
D[E] ← min (100, 10 + ∞) = 100
```
El costo del camino mínimo hacia E sigue siendo 100 ⟹

D[B]	D[C]	D[D]	D[E]
10	~~∞~~ 60	30	100



Matriz de Costos : C

	A	B	C	D	E
A	∞	10	∞	30	100
B	∞	∞	50	∞	∞
C	∞	∞	∞	∞	10
D	∞	∞	20	∞	60
E	∞	∞	∞	∞	∞

```
P[E] ← A
```
El predecesor de E sigue siendo A ⟹

P[B]	P[C]	P[D]	P[E]
A	~~A~~ B	A	A

Y así sucesivamente hasta alcanzar la solución.

3. Funcionamiento de Floyd-Warshall.

En informática, el algoritmo de Floyd-Warshall, descrito en 1959 por Bernard Roy, es un algoritmo de análisis sobre grafos para encontrar el camino mínimo en grafos dirigidos ponderados. El algoritmo encuentra el camino entre todos los pares de vértices en una única ejecución. El algoritmo de Floyd-Warshall es un ejemplo de programación dinámica.

El algoritmo de Floyd-Warshall es más general que el de Dijkstra, ya que determina la ruta más corta entre dos nodos cualquiera de la red.

- El algoritmo representa una red de *n* nodos como una matriz cuadrada de orden *n*, la llamaremos matriz *C*. De esta forma, el valor *Cij* representa el coste de ir desde el nodo *i* al nodo *j*, inicialmente en caso de no existir un arco entre ambos, el valor *Cij* será infinito.
- Definiremos otra matriz *D*, también cuadrada de orden *n*, cuyos elementos van a ser los nodos predecesores en el camino hacia el nodo origen, es decir, el valor *Dij* representará el nodo predecesor a *j* en el camino mínimo desde *i* hasta *j*. Inicialmente se comienza con caminos de longitud *1*, por lo que *Dij = i*.
- Las diagonales de ambas matrices representan el coste y el nodo predecesor para ir de un nodo a sí mismo, por lo que no sirven para nada, estarán bloqueadas.

Los pasos a dar en la aplicación del algoritmo de Floyd-Warshall son los siguientes:

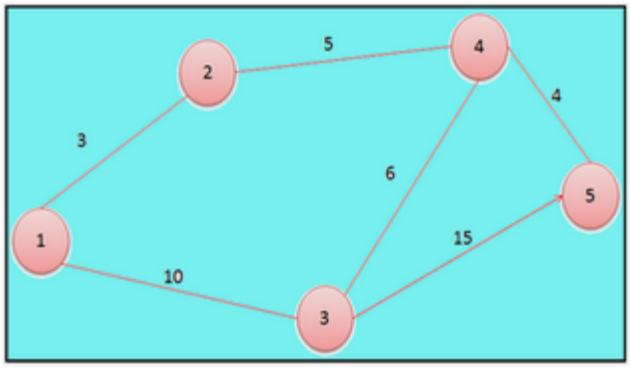
- Formar las matrices iniciales *C* y *D*.
- Se toma *k=1*.
- Se selecciona la fila y la columna *k* de la matriz *C* y entonces, para *i* y *j*, con $i \neq k$, $j \neq k$ e $i \neq j$, hacemos:

  Si $(Cik + Ckj) < Cij \rightarrow Dij = Dkj$ y $Cij = Cik + Ckj$

En caso contrario, dejamos las matrices como están.

- Si $k \leq n$, aumentamos *k* en una unidad y repetimos el paso anterior, en caso contrario terminamos las iteraciones.
- La matriz final *C* contiene los costes óptimos para ir de un vértice a otro, mientras que la matriz *D* contiene los penúltimos vértices de los caminos óptimos que unen dos vértices, lo cual permite reconstruir cualquier camino óptimo para ir de un vértice a otro.

A continuación, veremos un ejemplo de su funcionamiento:



Matriz C

	1	2	3	4	5
1	-	3	10	8	8
2	3	-	8	5	8
3	10	8	-	6	15
4	8	5	6	-	4
5	8	8	8	4	-

Matriz D

	1	2	3	4	5
1	-	1	1	1	1
2	2	-	2	2	2
3	3	2	-	3	3
4	4	4	4	-	4
5	5	5	5	5	-

Para k = 1:

Matriz C

	1	2	3	4	5
1	-	3	10	8	8
2	3	-	13	5	8
3	10	13	-	6	15
4	8	5	6	-	4
5	8	8	8	4	-

Matriz D

	1	2	3	4	5
1	-	1	1	1	1
2	2	-	1	2	2
3	3	1	-	3	3
4	4	4	4	-	4
5	5	5	5	5	-

Saltamos a k = 3:

Matriz C

	1	2	3	4	5
1	-	3	10	8	25
2	3	-	13	5	28
3	10	13	-	6	15
4	8	5	6	-	4
5	8	8	8	4	-

Matriz D

	1	2	3	4	5
1	-	1	1	2	3
2	2	-	1	2	3
3	3	1	-	3	3
4	2	4	4	-	4
5	5	5	5	5	-

Y así sucesivamente hasta obtener las matrices:

Matriz C

	1	2	3	4	5
1	-	3	10	8	12
2	3	-	11	5	9
3	10	11	-	6	10
4	8	5	6	-	4
5	12	9	10	4	-

Matriz D

	1	2	3	4	5
1	-	1	1	2	4
2	2	-	4	2	4
3	3	4	-	3	4
4	2	4	4	-	4
5	2	4	4	5	-

4. Pseudocódigo y complejidad de Dijkstra.

El tiempo de ejecución del algoritmo de Dijkstra depende tanto del número de nodos E y el número de aristas V como de la implementación del conjunto de vértices por visitar. Las implementaciones más eficientes utilizan una estructura de datos que se mantiene organizada y por tanto tiene un tiempo de consulta de O(log n). De aquí podemos calcular la complejidad del algoritmo de Dijkstra:

Para cada nodo en E se debe revisar todas sus aristas conectadas, que será como máximo |V|-1. O(|E|+|V|)

Conseguir y actualizar el peso de cada vértice es O(log|V|).

Por tanto el tiempo total es la suma de nodos y aristas a revisar por el tiempo que toma actualizar cada arista adyacente: O((|E|+|V|)log|V|)=O(|E|log|V|)

```
DIJKSTRA (Grafo G, nodo_fuente s)
    para u ∈ V[G] hacer
        distancia[u] = INFINITO
        padre[u] = NULL
    distancia[s] = 0
    Encolar (cola, grafo)
    mientras que cola no es vacía hacer
        u = extraer_minimo(cola)
        para v ∈ adyacencia[u] hacer
            si distancia[v] > distancia[u] + peso (u, v) hacer
                distancia[v] = distancia[u] + peso (u, v)
                padre[v] = u
                Actualizar(cola,distancia,v)
```

Para poder compararlo correctamente con Floyd-Warshall hay también que repetir esto con cada nodo como nodo inicial por tanto multiplicamos esta complejidad por el número de nodos quedando O(|E|^2log|V|).

Para comprobar esto de manera práctica, se han realizado mediciones del tiempo de ejecución sobre distintos grafos con diferentes números de nodos y de aristas. El resultado se puede observar en la siguiente tabla:

Número de nodos (n)	Tiempo de ejecución (ms)
3	1,283660
5	2,780796
8	3,853728
11	11,350863

Y el gráfico correspondiente:

5. Pseudocódigo y complejidad de Floyd.

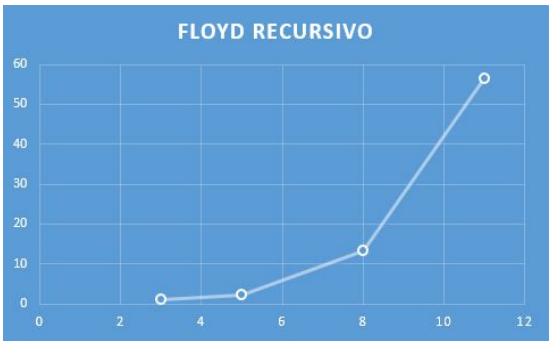
Siendo n = |V|, debemos conseguir los 2n^2 que resultan de conseguir el camino mínimo entre todo par de nodos, esto a través de un número de iteraciones igual al número de nodos, n. El número total de operaciones resulta n*2n^2=2n^3, por tanto el tiempo de ejecución resulta ser O(n^3).

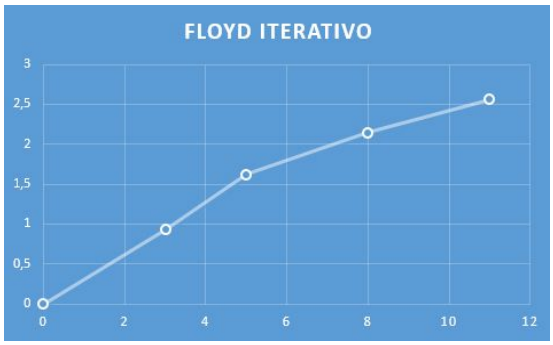
```
let dist be a |V| × |V| array of minimum distances initialized to ∞ (infinity)
for each vertex v
   dist[v][v] ← 0
for each edge (u,v)
   dist[u][v] ← w(u,v)  // the weight of the edge (u,v)
for k from 1 to |V|
   for i from 1 to |V|
      for j from 1 to |V|
         if dist[i][j] > dist[i][k] + dist[k][j]
             dist[i][j] ← dist[i][k] + dist[k][j]
         end if
```

Para este algoritmo también se han realizado mediciones sobre distintos grafos con diferentes números de nodos y de aristas. Las mediciones se han realizado sobre dos variaciones del mismo algoritmo, una recursiva y otra iterativa (ambas versiones se encuentran disponibles en la carpeta correspondiente del repositorio). La versión recursiva es mucho más ineficiente que la iterativa, debido a las llamadas recursivas. Los resultados se pueden observar en la siguiente tabla:

Número de nodos (n)	Floyd recursivo	Floyd iterativo
3	1,105734	0,926862
5	2,233947	1,622145
8	13,326303	2,145761
11	56,636881	2,558855

Y los gráficos correspondientes:





6. Conclusión: Dijkstra vs. Floyd.

Al comparar Dijkstra y Floyd un detalle importante es que la complejidad de Dijkstra no depende únicamente de la cantidad de vértices que posea, sino también del número de aristas, por tanto no es posible decir que uno sea más eficiente que otro de manera general. Dijkstra será más rápido en grafos donde la cantidad de aristas sea pequeña relativa al número de nodos que tenga, mientras Floyd ganará en grafos más densos. Para resaltar las ventajas de Floyd, que trabaja con Programación Dinámica, hemos utilizado grafos densos para las mediciones.

Cabe notar que debido al funcionamiento de Dijkstra, este no puede ser utilizado en grafos con costes negativos, a diferencia de Floyd, que sí puede ser utilizado con la condición de que no haya bucles negativos.

En la siguiente tabla se pueden observar las diferencias entre los distintos algoritmos:

Número de nodos (n)	Dijkstra (ms)	Floyd recursivo (ms)	Floyd iterativo (ms)
3	1,283660	1,105734	0,926862
5	2,780796	2,233947	1,622145
8	3,853728	13,326303	2,145761
11	11,350863	56,636881	2,558855

7. Referencias.

https://es.wikipedia.org/wiki/Problema_del_camino_m%C3%A1s_corto
https://www.youtube.com/watch?v=4OQeCuLYj-4
http://es.stackoverflow.com/questions/28688/de-qu%C3%A9-trata-la-t%C3%A9cnica-bottom-up
https://en.wikipedia.org/wiki/Floyd%E2%80%93Warshall_algorithm#Algorithm
https://es.wikipedia.org/wiki/Algoritmo_de_Dijkstra
https://www.ecured.cu/Algoritmo_de_Dijkstra
http://algoritmodefloyd.blogspot.com.es/