Analísis de Floyd-Warshall

Diseño y análisis de algoritmos

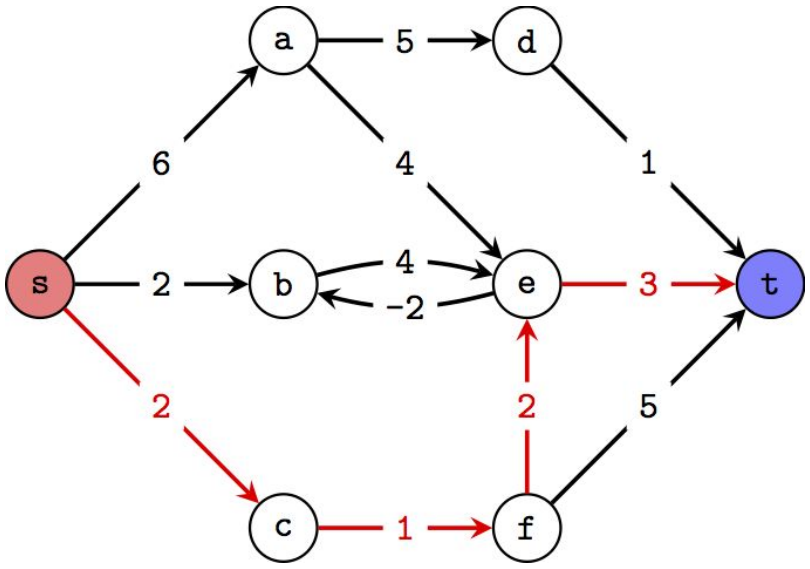
Carlos Troyano Carmona
Miguel Bravo Arvelo

Indice

Introducción

Vamos analizar dos algoritmos que resuelven el mismo problema y con un análisis empírico de tiempo de ejecución y comparándolo con sus curvas teóricas al calcular su complejidad.
El problemas es el de encontrar el camino más corto entre uno o todos los nodos del grafo y el resto. Veremos los pro y los contras de cada uno y que diferencias hay en la resolución del problema.

Imagen 1.1 problema del camino más corto



Floyd-Warshall

Floyd-Warshall. Es un algoritmo que trata el problema del "camino más corto" para un grafo cualquiera dando como solución una matriz con todas las distancias mínimas de dos pares de nodos. Con capacidad de detectar los ciclos negativos y además calcula la distancia entre todos los nodos del grafo(siendo infinito los que no están en componentes conexas comunes).

El algoritmo se basa en la técnica de programación dinámica. La cual en grandes pinceladas es una técnica que subdivide el problema en subproblemas del mismo tipo. También almacena las soluciones que obtiene para evitar recalcular las soluciones ya calculadas. Con esto mejora la curva de tiempo asintótica del algoritmo.

Aún así este algoritmo no es nada eficiente pero si es más eficiente que una búsqueda lineal que tendría un tiempo exponencial y no polinómico. Vamos a proceder a ver pseudocódigo del algoritmo, Además implementaremos su gran rival **Dijkstra's** que resuelve el mismo problema pero solo para un nodo a todos los demás del grafo. Eso sí sin tener en cuenta los ciclos negativos que pudiese haber.

Pseudocodigo

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

Análisis

Como se puede apreciar fácilmente en el pseudocódigo el algoritmo tiene tres iteraciones anidadas desde 1 hasta N si asumimos como N los V vértices. Por lo tanto la complejidad es **$O(n^3)$**. Esto es fácilmente demostrable con lo siguiente :

Si utilizamos matrices booleanas, para encontrar todos los $n2$ de WK desde WK-1 se necesita hacer $2n2$ operaciones binarias. Debido a que empezamos con W0 = WK y computamos la secuencia de n matrices booleanas W1, W2, ... Wn = MR*, el número total de operaciones binarias es de n x $2n2 = 2n3$. Por lo tanto, la complejidad del algoritmo es O(n3) y puede ser resuelto por una máquina determinista de Turing en tiempo polinómico.

Reconstrucción del camino

```
1 procedure Path(u, v)
2   if next[u][v] = null then
3       return []
4   path = [u]
5   while u ≠ v
6       u ← next[u][v]
7       path.append(u)
8   return path
```

Dijkstra's

Dijkstra's . Es un algoritmo que resuelve el camino más corto de un único nodo de origen a uno destino sin haber arcos sin valor negativo que no hace 'exploración' directa al nodo destino, más bien la única consideración en determinar el siguiente nodo es la distancia al nodo de inicio. Por lo tanto este algoritmo se expande hacia fuera del nodo de inicio de manera interactiva considerando cada nodo de menor distancia hasta que llega a el destino.

Cuando se entiende el algoritmo de esta manera se puede llegar a la conclusión de que el camino encontrado sea el de menor recorrido. Sin embargo, en topologías específicas este algoritmo puede llegar a ser relativamente lento.

```
1  function Dijkstra(Graph, source):
2
3      create vertex set Q
4
5      for each vertex v in Graph:
6          dist[v] ← INFINITY
7          prev[v] ← UNDEFINED
8          add v to Q
10     dist[source] ← 0
11
12     while Q is not empty:
13         u ← vertex in Q with min dist[u]
14
15         remove u from Q
16
17         for each neighbor v of u:
18             alt ← dist[u] + length(u, v)
19             if alt < dist[v]:
20                 dist[v] ← alt
21                 prev[v] ← u
22
23     return dist[], prev[]
```

Analisis

La complejidad computacional del algoritmo de Dijkstra se puede calcular contando las operaciones realizadas:

El algoritmo consiste en n-1 iteraciones, como máximo. En cada iteración, se añade un vértice al conjunto distinguido.
En cada iteración, se identifica el vértice con la menor etiqueta entre los que no están en Sk. El número de estas operaciones está acotado por n-1.
Además, se realizan una suma y una comparación para actualizar la etiqueta de cada uno de los vértices que no están en Sk.
Luego, en cada iteración se realizan a lo sumo 2(n-1) operaciones.
Entonces:
Teorema: El algoritmo de Dijkstra realiza $O(n^2)$ operaciones (sumas y comparaciones) para determinar la longitud del camino más corto entre dos vértices de un grafo ponderado simple, conexo y no dirigido con n vértice

Resultados experimentales

Vamos analizar los algoritmos con los tiempos recogidos con el código desarrollado en Java. Cómo podemos ver comparándolo con la curva teórica de la complejidad de los algoritmos coincide bien aunque tengamos menos datos de entrada para describir bien la curva. Podemos apreciar la diferencia de eficiencia entre uno y el otro.

Imagen 2.1 Tiempos experimentales

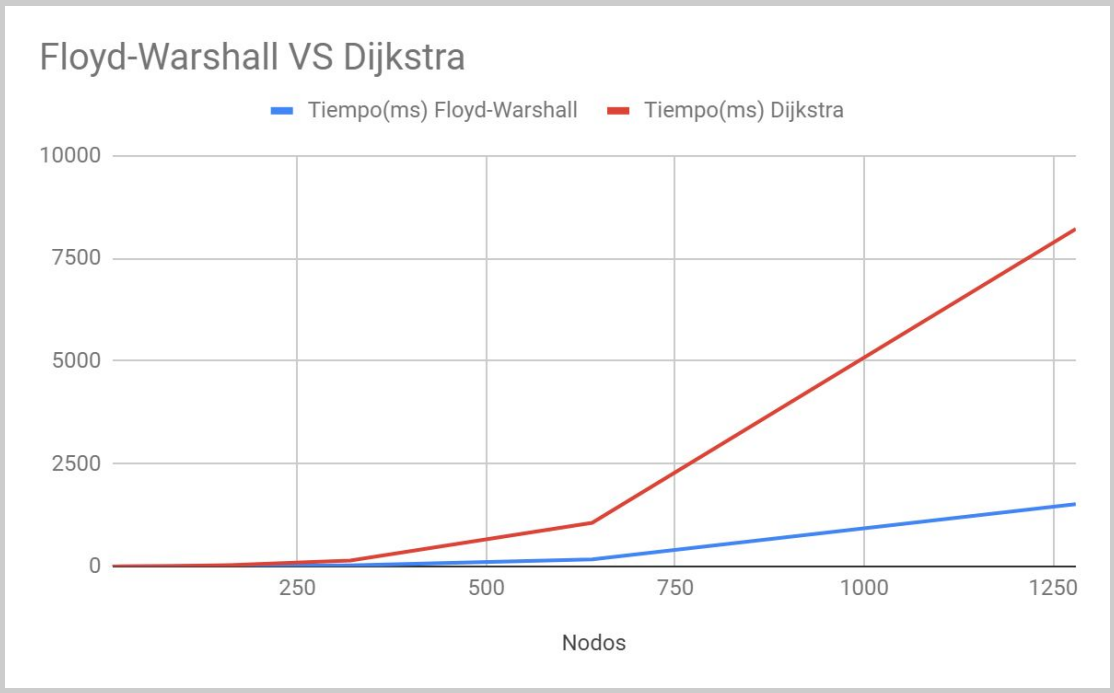
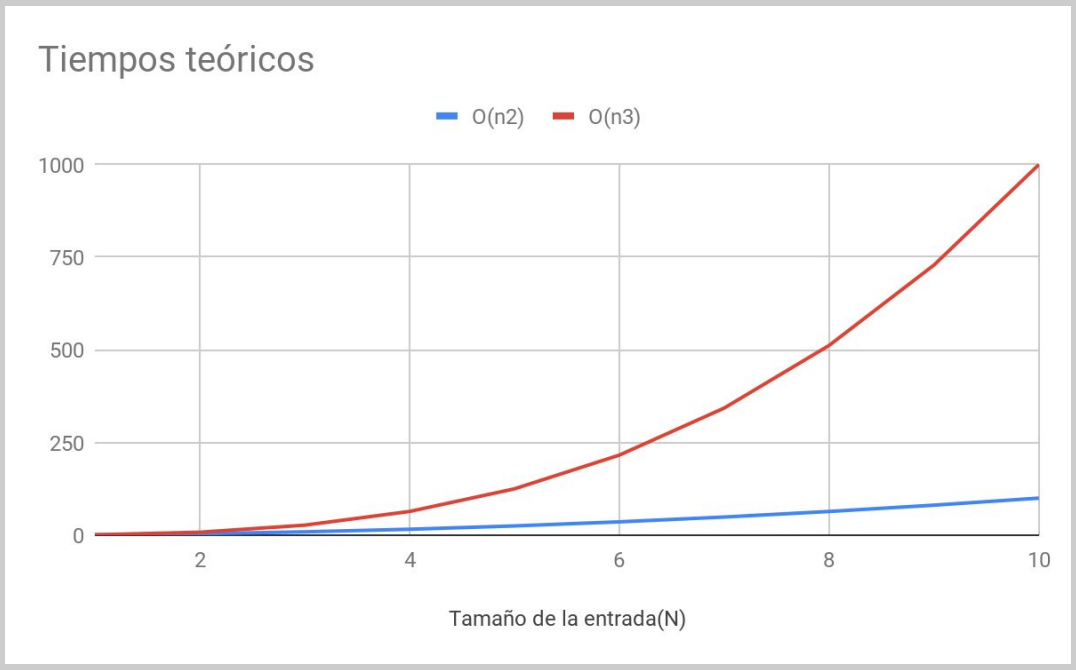


Imagen 2.2 Tiempos teóricos



Conclusiones

El área de conocimiento de teoría de grafos estudiada en este trabajo hemos podido apreciar la utilidad de los algoritmos de recorridos mínimos como solución simple a problemas planteados como un grafo

Además podemos inferir de los resultados que del análisis que si tenemos que obtener el camino más corto en un grafo en el que sabemos que no tenemos ciclos negativos debemos usar Dijkstra que es una alternativa más eficiente. Si no los sabemos deberemos sacrificar el tiempo para comprobarlo con Floyd-Warshall.

Cabe destacar que hoy en día hay algoritmos más eficientes para resolver el mismo problema.

Referencias

Shortest path problem, (s. f). En Wikipedia. Recuperado el 24 de Marzo de 2017 de https://en.wikipedia.org/wiki/Shortest_path_problem

Graph theory, (s. f). En Wikipedia. Recuperado el 22 de Marzo de 2017 de https://en.wikipedia.org/wiki/Graph_theory#Computer_science

Dijkstra's algorithm, (s. f). En Wikipedia. Recuperado el 22 de Marzo de 2017 de https://en.wikipedia.org/wiki/Dijkstra%27s_algorithm

Floyd-Warshall algorithm, (s. f). En Wikipedia. Recuperado el 21 de Marzo de 2017 de https://en.wikipedia.org/wiki/Floyd%E2%80%93Warshall_algorithm

MIT OpenCourseWare. (31 de diciembre de 2017). [Video de Youtube]. Recuperado de https://www.youtube.com/watch?v=NzgFUwOaolw