Instituto Politécnico Nacional
Escuela Superior de Cómputo

Análisis de Algoritmos, Sem: 2022-2, 3CV11, Práctica Extra, 07 de junio de 2022

Práctica Extra: Algoritmo de Dijkstra

Luis Francisco Renteria Cedillo, Denzel Omar Vazquez Perez.

*lrenteriac*1400@*alumno.ipn.mx*, *dvazquezp*1600@*alumno.ipn.mx*

Resumen: En el presente documento se muestra el análisis *a Priori* y *a Posteriori* a el algoritmo de Dijkstra para determinar su complejidad temporal a partir de su implementación en el lenguaje de programación C con matrices de adyacencia.

Palabras Clave: Dijkstra, Matriz, Greedy, Solución optima, Lenguaje C.

1 Introducción

Siempre ha habido problemas donde se requiere encontrar el camino más corto entre un conjunto de puntos incluso en la informática. Hay muchos más ejemplos y comunes, desde la navegación por satélite hasta el enrutamiento de paquetes de Internet; incluso encontrar la longitud de cable más corta para conectar los pines en una placa de circuito.

Al tener mucha recurrencia este tipo de problemas surgen algunos algoritmos estándar que se pueden utilizar para encontrar una solución, uno de ellos se conoce como algoritmo de Dijkstra, diseñado por el físico holandés Edsger Dijkstra en 1956, cuando pensó en cómo podría calcular la ruta más corta de Rotterdam a Groningen.
Este algoritmo tiene bastas aplicaciones para resolver una amplia variedad de problemas complejos, ya que esta escrito en el contexto de grafos ponderados, por lo que todo el lenguaje lo refleja pues esta compuesto de vértices y aristas. Por tanto, este algoritmo funcionará en cualquier cosa que pueda abstraerse a una serie de vértices que estén conectados y tienen valores adjuntos a esas conexiones.

Así el algoritmo funciona bien para la búsqueda de rutas para un conjunto completo de vértices pero, cuando se usa para buscar la ruta más corta a un objetivo específico, puede ser ineficiente y no se podrían obtener soluciones al problema.

2 Conceptos Básicos

2.1 Matrices de adyacencia

Una matriz de adyacencia representa un grafo con una matriz nxn donde n es $|V|$. Así, si $M = [E(i,j)]$ donde cada elemento $E(i,j)$ contiene los atributos de la arista. Si las aristas no tienen un costo, el grafo se puede representar mediante una matriz booleana para ahorrar espacio en la memoria véase Figura 1.

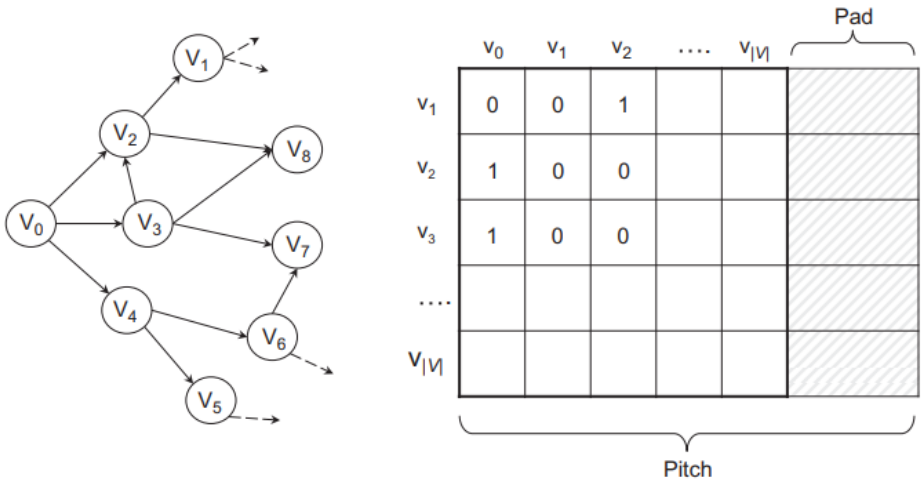


Figura 1: Matrices de adyacencia

Los mas comunes algoritmos que usan esta representación son la ruta más corta de todos los pares, donde si el gráfico es ponderado, cada valor de $E(i,j)$ se define de la siguiente manera:

$$M[i,j] = \begin{cases} 0 & si \quad i = j \\ w(i,j) & si \quad i = j \ y \ (i,j) \in E \\ \infty & si \quad i \neq j \ y \ (i,j) \notin E \end{cases}$$

2.2 Algoritmos Greedy

Los algoritmos Greedy emplean un procedimiento de resolución de problemas para construir progresivamente soluciones candidadtas óptimas, para aproximarse al óptimo global, obteniendo mejores soluciones óptimas localmente en

cada etapa. Así con esta estrategia de solución de problemas no se puede producir una solución óptima global, pero si buenas soluciones óptimas localmente en un tiempo razonable y con menos esfuerzo computacional, véase **Algorithm 1** para ver la estructura de la implementación general de la estrategia.

Algorithm 1 FuntionGreedy(set C)

```
1: S=∅
2: while ¬isSolution(S) && C≠ ∅ do
3:     x=select_best_option(C)
4:     C=C-{x}
5:     if isOptimum(S∪{x} then
6:         S=S∪{x}
7:     end if
8: end while
9: if isSolution(S∪{x} then
10:    return S
11: end if
```

2.3 Algoritmo de Dijkstra

El algoritmo de Dijkstra es el método elegido para encontrar las rutas más cortas en un grafo ponderado por aristas y/o vértices. Dado un vértice de inicio particular, encuentra la ruta mas corta a cualquier otro vértice en el grafo, incluido el vértice destino, véase Figura 2.

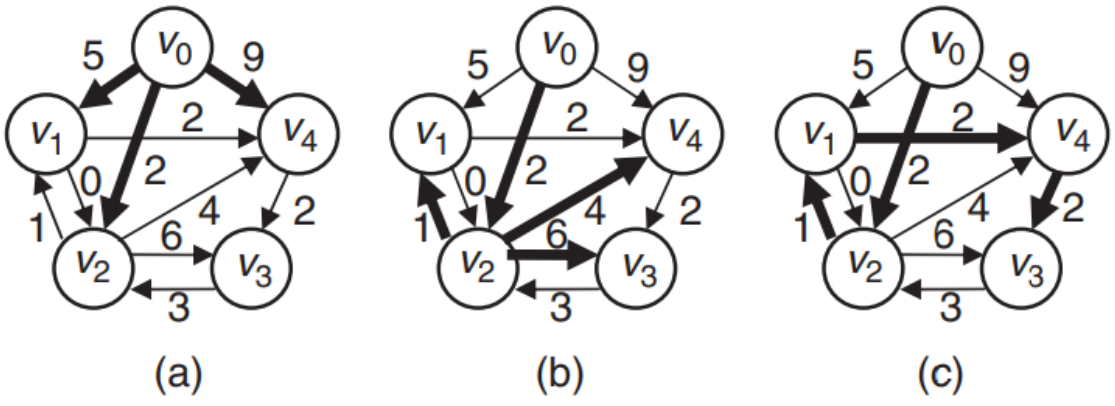


Figura 2: Ejemplo del algoritmo de Dijkstra

2.4 Pseudocódigo del algoritmo de Dijkstra

El pseudocódigo del **Algorithm 2** muestra el algoritmo de Dijkstra que a partir de un grafo G, haciendo uso de matrices de adyacencia y al inicializar los arreglo S, D y P, repetidamente realiza la diferencia entre los conjuntos de V y S, para tomar el menor peso w de V-S e insertando este en S, luego se comparan los vértices que están en V-S, pasando por cada uno de estos para para encontrar el valor mínino de D[v] y D[w] G[w,v], para su inserción en de w en P[v], terminando si V-S en $\emptyset$.

Algorithm 2 Dijkstra(G:Graph, initial:node,V:Vértex)

```
1: Inicializar(G,V)
2: S={0, ..., |V| − 1}
3: D={0, ..., |V| − 1}
4: P={0, ..., |V| − 1}
5: insert(initial,S)
6: for v∈V, v≠initial  do
7:     D[v]= G[initial,v]
8:     P[v]=initial
9: end for
10: while V-S≠ ∅ do
11:     getMin(w∈V-S)
12:     S=S∪{w}
13:     for v∈V-S do
14:         D[v]=min(D[v], D[w]+G[w,v])
15:         if min(D[v], D[w]+G[w,v])=D[w]+G[w,v] then
16:             P[v]=w
17:         end if
18:     end for
19: end while
```

3 Experimentación y Resultados

3.1 Análisis a Priori

Dada la implementación del **Algorithm 2** en el lenguaje de programación C se tiene la siguiente codificación. Donde a partir de un grafo se genera una matriz adyacente para su posterior ejecución, entonces dado cada bloque de sentencia del algoritmo, se muestra la obtención del orden de complejidad para el peor caso de este por medio de un análisis de segmentos de código, véase Figura 3, donde se concluye que $T(n) \in \Theta(n^2)$

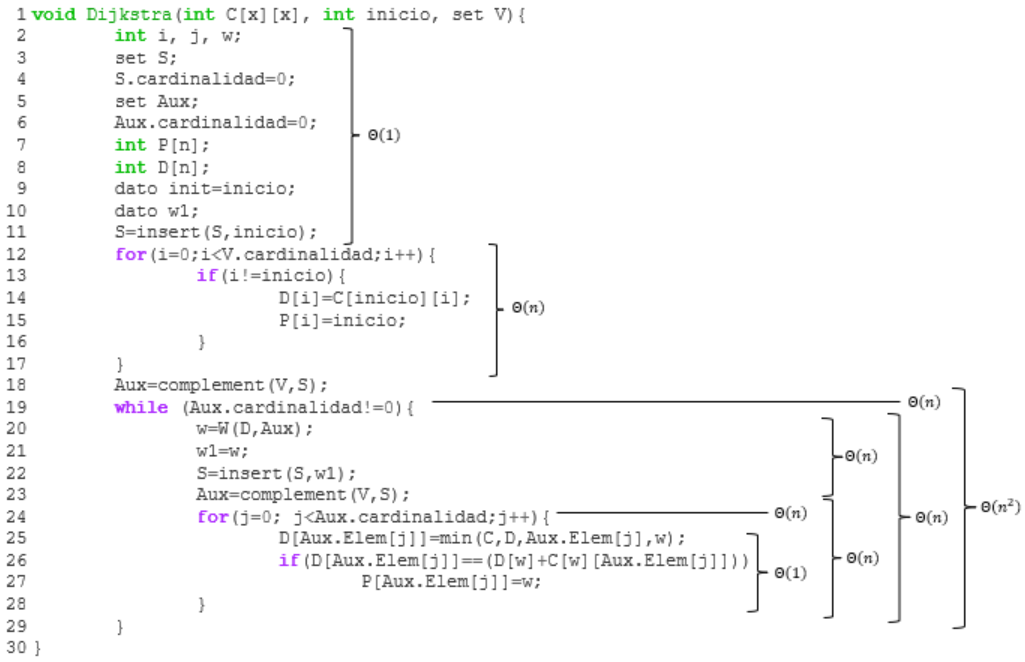


Figura 3: Implementación del algoritmo de Dijkstra en lenguaje C

Así el algoritmo mantienen invariante el $Aux = V - S$ al comienzo de cada iteración del bucle while de las lineas 19-30. En la linea 11 se inicializa el arreglo S para contenga el vértice de partida hasta contener todos los vértices de V, desde insert(initial,S) en ese momento se cumple la condición de la linea 19 si V-S$\neq \emptyset$.

Por lo que cada vez que pasa por el bucle while, en la linea 20 se extrae el vértice con la distancia menor cumpliendo que D[v]$\in$Aux, así se coloca este en el arreglo S, obteniendo de nueva cuenta $Aux = V - S$ para pasar por cada uno de los elementos del conjunto Aux, para obtener la mínima distancia para cada $v \in Aux$ tal que $D[v] = min(D[v], D[w] + G[w, v])$, donde si $D[v] = D[w] + G[w, v]$ se inserta el vértice w en el arreglo P. Por tanto hay que observar que el algoritmo nunca inserta vértices de V después de la linea 18, y que cada vértice se extrae de V y se inserta en S exactamente una vez de modo que el ciclo while de la lineas 19-30 itera exactamente $|V|$ veces, donde $n = |V|$.

Ante esto se puede decir que el algoritmo de Dijkstra al siempre elegir el vértice con menor costo en V-S para agregarlo al conjunto S, se puede afirmar que este usa una estrategia Greevy.

3.2 Análisis a Posteriori

Al ejecutar la función con el nombre *Dijkstra* , con la asignación de valores de entrada por una matriz de adyacencia C, un nodo inicial y el conjunto de todos los vértices, se muestran los siguientes ejemplos de la ejecución, donde los arreglo S representa la solución al problema, el arreglo D las distancias desde el nodo inicial a cada uno de los nodos y el arreglo P la contrucción de los caminos correspondientes.
El primer grafo implementado tiene |V|=4 entonces, si se escoge como variable inicial=0 se tiene los siguientes resultados, véase Figura 4.

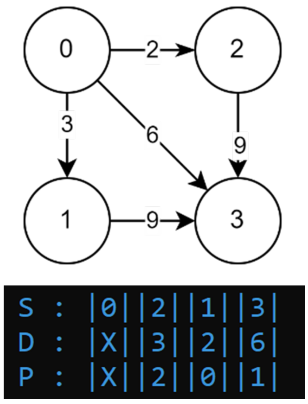


Figura 4: Ejemplo 1 del algoritmo de Dijkstra

El segundo grafo implementado tiene |V|=5 entonces, si se escoge como variable inicial=2 se tiene los siguientes resultados, véase Figura 5.

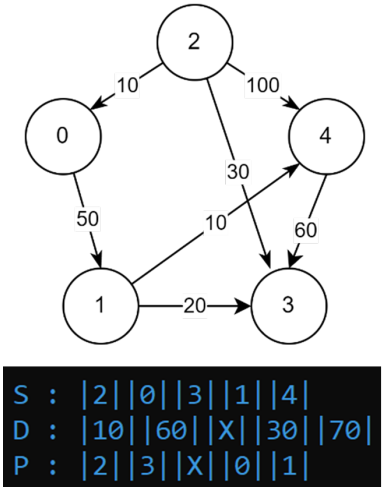
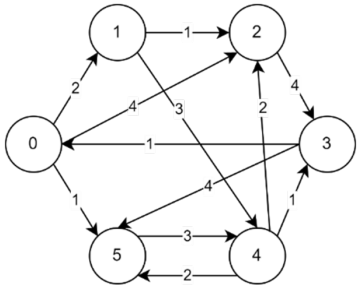


Figura 5: Ejemplo 2 del algoritmo de Dijkstra

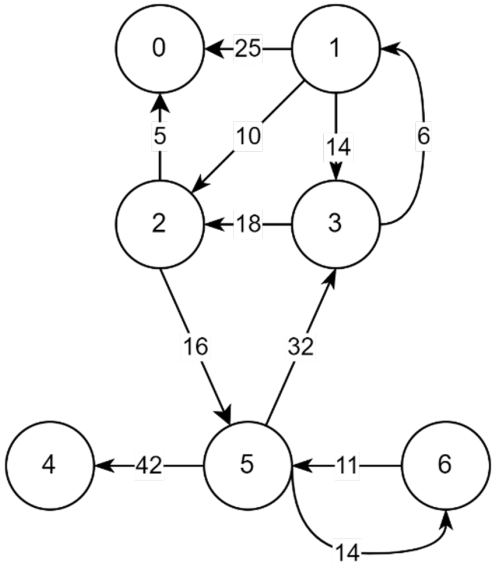
El tercer grafo implementado tiene |V|=6 entonces, si se escoge como variable inicial=0 se tiene los siguientes resultados, véase Figura 6.




```
S : |0||5||1||4||2||3|
D : |X||2||3||4||3||1|
P : |X||5||4||2||1||0|
```

Figura 6: Ejemplo 3 del algoritmo de Dijkstra

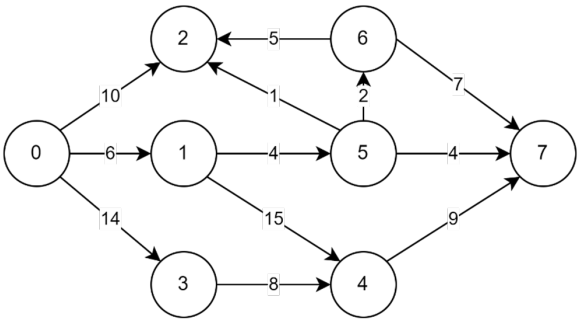
El cuarto grafo implementado tiene |V|=7 entonces, si se escoge como variable inicial=2 se tiene los siguientes resultados, véase Figura 7.




```
S : |2||0||5||6||3||1||4|
D : |5||54||X||48||58||16||30|
P : |2||3||X||6||1||0||5|
```

Figura 7: Ejemplo 4 del algoritmo de Dijkstra

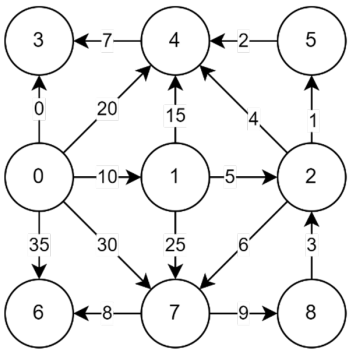
El quinto grafo implementado tiene |V|=8 entonces, si se escoge como variable inicial=0 se tiene los siguientes resultados, véase Figura 8.




```
S : |0||1||5||2||6||7||3||4|
D : |X||6||10||14||21||10||12||14|
P : |X||0||5||7||3||1||2||6|
```

Figura 8: Ejemplo 5 del algoritmo de Dijkstra

El sexto grafo implementado tiene |V|=9 entonces, si se escoge como variable inicial=0 se tiene los siguientes resultados, véase Figura 9.




```
S : |0||3||1||2||5||4||7||6||8|
D : |X||10||15||0||18||16||29||21||30|
P : |X||3||1||0||5||2||7||4||6|
```

Figura 9: Ejemplo 6 del algoritmo de Dijkstra

Ya que el dato de entrada es un número natural (incluido el 0) n que es la $|V|$, se tiene que la salida del programa da el número de operaciones hechas por cada iteración, los resultados registrados se muestran en la Tabla 1.

Tabla 1: Complejidad temporal del algoritmo de Dijkstra

n	**T(n)**
0	13
1	16
2	28
3	45
4	67
5	96
6	129
7	167
8	209
9	262

Dado el comportamiento de los puntos de muestra, se aprecia que este es cuadrático, no habiendo diferencia entre el mejor y peor caso, y conociendo que Θ se define como:

$$\Theta(g(n)) = \{f(n) \mid \exists_n\ C_1,\ C_2 > 0\ \&\ n_0 > 0\ tal\ que$$
$$0 \leq\ C_1 g(n)\ \leq\ f(n)\ \leq\ C_2 g(n)\ \forall\ n \geq\ n_0\}$$

Se propone $C_1 = \frac{20}{7}$, $C_2 = \frac{17}{4}$ y $n_0 = 4$, es posible obsevar en la Figura 10 que dados estos valores se acotan la parte superior como la inferior así como el punto de cruce en el que se cumple la inecuación, por tanto se puede decir que $T(n) \in \Theta(n^2)$.

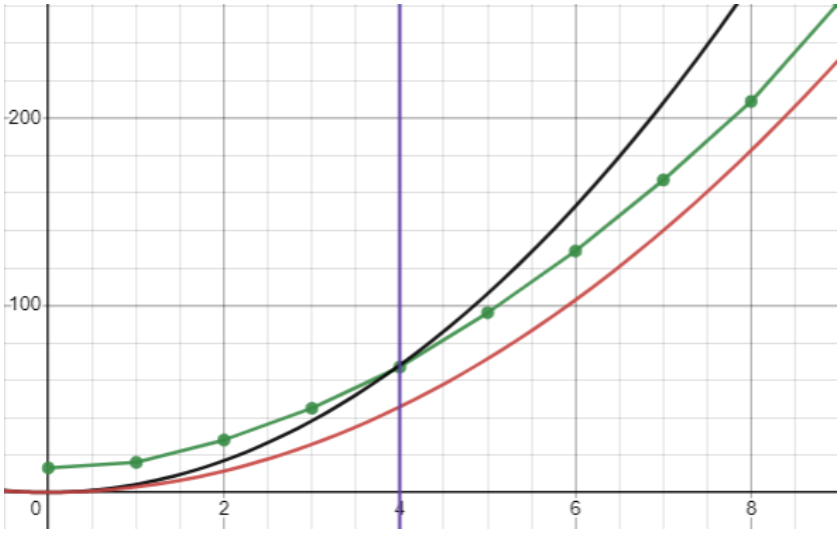


Figura 10: Gráfica del orden de complejidad del algoritmo de Dijkstra

4 Conclusiones

Luis Francisco Renteria Cedillo

La importancia de obtener el camino más corto reside en alcanzar una ruta de punto A a punto B en el menor costo posible. El Algoritmo de Djikstra contribuye a solucionar este tipo de problemas en múltiples campos como la telemática, inteligencia artificial y logística para transportes. Gracias a este algoritmo se es posible resolver grafos con múltiples nodos, donde se observa lo importante y crucial que son los algoritmos Greedy.

Denzel Omar Vazquez Perez

En la práctica se pudo analizar el algoritmo de Dijkstra por medio de matrices de adyacencia, donde se muestra las ventajas de usar una estrategia de programación Greedy y es que este algoritmo es muy similar al algoritmo de Prim, dado que en cada iteración, se agrega exactamente un vértice al arreglo

solición, donde haciendo el seguimiento de la mejor ruta vista hasta la fecha para todos los vértices fuera de S y se agrega en orden de costo creciente. La diferencia entre los algoritmos de Dijkstra y Prim es cómo califican la conveniencia de cada vértice.

Finalmente, es posible decir que Dijkstra funciona bien como un algoritmo de búsqueda de rutas para un conjunto completo de nodos pero, cuando se usa para buscar la ruta más corta a un vértice específico, puede ser ineficiente, por lo que no se obtendrá una solución óptima al problema en las primeras iteraciones, ya que algoritmo sigue siempre el camino con menor costo de V-S.

5 Bibliografía

Busato, F., & Bombieri, N. (2017).*Graph algorithms on GPUs. Advances in GPU Research and Practice*(pp.163–198).

Brassard, G. (1997). *Fundamentos de Algoritmia*. España: Ed. Prentice Hall.

Cormen, E. A. (2022). *Introduction To Algorithms*, 3Rd Ed. Phi.

Huang, C.-Y. (Ric), Lai, C.-Y., & Cheng, K.-T. (Tim). (2009).*Fundamentals of algorithms. Electronic Design Automation*(pp.173–234).