HPPS
Informe 4 - Algoritmo de Dijkstra

Juan Braun

4 de junio de 2013

Introducción

El objetivo de este informe es comentar los resultados obtenidos para las diferentes versiones del algorimto de Dijkstra implementadas. El algoritmo de Dijkstra es un algoritmo de búsqueda en grafos. Dado un grafo $G = (E,V)$ donde E son aristas con costos asociados y V los vértices, el algoritmo calcula los caminos de menor costo desde un vértice inicial $s \in V$ hasta todos los demás vértices $v \in V - \{s\}$. A continuación se muestra el pseudo-código del algoritmo para el caso en que se usa la cola de prioridad.

```
dijkstra(G,s)
dist[s] = 0;  (distancia al vertice inicial)
for (todos los demas vertices)
   dist[v] = inf; (Las distancias a los nodos desconocidos es infinita)

S vacio; (En S voy guardando los nodos visitados)
build_heap(Q,V); (creo un min-heap Q con los vertices, se ordenan según el costo)

while (Q no vacio)'
   u = extractMin(Q) (saco el siguinte nodo con costo mas bajo)
   Agrego u a la lista de visitados S'

      for (todos los vecinos de u)

         if dist[v]>dist[u]+costo(u,v) (si hay un nuevo camino de menor costo)
            d[v] = d[u]+costo(u,v) (actualizo costo)
            move_up(Q,v) (Reordeno el heap)
         end if

      end for

return dist
```

El caso en el que no se usa la cola de prioridad es muy similar, la diferencia esta en que cada vez que se quiere conseguir el siguiente nodo de menor costo se ordena el arreglo nuevamente el arreglo.

El pseudo-código que se muestra sirve para calcular los menores costos de un nodo inicial hacía todos los demás, lo que se necesita en este caso es encontrar el camino de menor costo entre dos nodos dados.

Para calcular el camino hacia un nodo en particular se necesita tener un arreglo en el que se almacenan los nodos previos con el costo mínimo para llegar a ellos. Con este arreglo y sabiendo que cualquier tramo incluido en un camino de costo mínimo es un camino de costo mínimo para un nodo en particular, es posible calcular el camino mínimo.

Por ejemplo para el grafo de la Figura 1, cuando se busca el camino entre el vértice A y el vértice F, se obtiene los siguientes arreglo con costos mínimos y nodos previos.

V	A	B	C	D	E	F	G	Z
Costos	0	25	20	25	25	20	10	30
Previo	nil	D	A	G	D	G	A	G

Cuadro 1: Costos y nodos previos

A partir del Cuadro 1 se puede calcular el menor camino de A a F. Se mira el nodo previo de F, G. Ahora se mira el nodo previo de G, A. El camino de menor costo es $A \rightarrow G \rightarrow F$

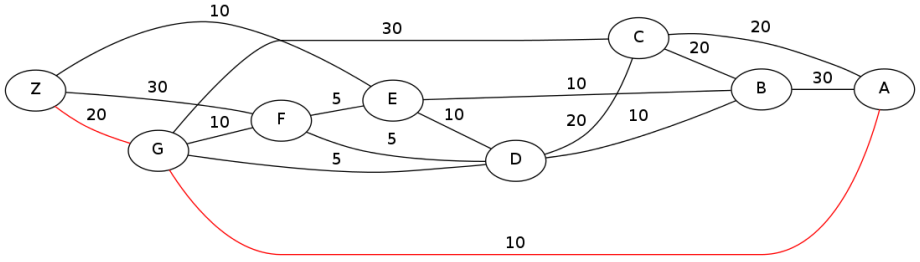


Figura 1: Grafo de prueba

La complejidad teórica del algoritmo depende de que versión se utilice. Si se utiliza la versión con *heap* se tiene que la complejidad esperada es de $O(E+V^2)$. En el caso particular en que el grafo tiene pocas aristas, $E = o(V^2 log(V))$, la complejidad del algoritmo baja a $O((V+E)\log(V))$.

En el caso en que no se utiliza el *min heap* la complejidad del algoritmo sube a $O(V^3)$ ya que esta versión ordena V veces el array de nodos y la función `sort` que ordena es $O(V^2)$.

1. Resultados

Se midieron los ciclos de reloj para diferentes entradas. Los resultados obtenidos se muestran el Cuadro 2.

	# nodos	# ciclos con heap	# ciclos sin heap
E1	8	21 260	198 015
E2	50	154 892	2 917 770
E3	75	279 164	7 780 416
E4	100	538 346	17 863 372
E5	250	2 718 002	204 572 291
E6	500	9 950 835	1 442 783 202
E7	750	21 601 507	4 652 164 219
E8	1000	37 042 054	10 718 859 780

Cuadro 2: Resultados

La matrices de adyacencia que se utilizaron para obtener los resultados del Cuadro 2, fueron matrices densas.En todos los casos requiere menos ciclos la versión que utiliza el *min heap*.

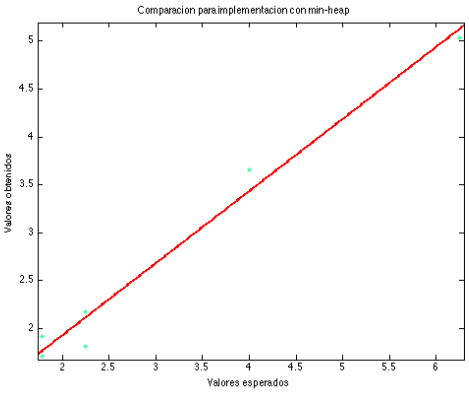
	# relacion teorica	# relacion obtenida
E3/E2	2.25	1.81
E4/E3	1.78	1.92
E5/E4	6.25	5.04
E6/E5	4	3.66
E7/E6	2.25	2.17
E8/E7	1.78	1.71

Cuadro 3: Comparación resultados implementación con heap

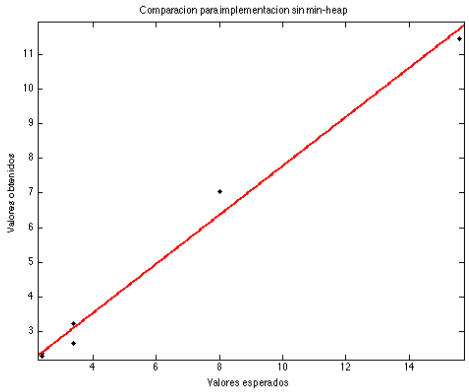
	# relacion teorica	# relacion obtenida
E3/E2	3.375	2.67
E4/E3	2.37	2.29
E5/E4	15.6	11.45
E6/E5	8	7.05
E7/E6	3.375	3.22
E8/E7	2.37	2.3

Cuadro 4: Comparación resultados implementación sin heap

En los Cuadros 6 y 4 se puede ver la comparación entre los resultados obtenidos y los resultados esperados para cada caso. También se pude ver gráficamente en las Figuras 2(a) y 2(b). Si bien difieren entre si, es razonable considerar que los resultados obtenidos son coeherentes con la complejidad teorica del algoritmo.



(a) Comparación implementación con min-heap



(b) Comparación implementación sin min-heap

Figura 2: Comparaciones de los resultados esperados y los obtenidos

Se utilizaron matrices de adyacencia esparzas con una densidad de 40%. Los resultados obtenidos se muestran en el Cuadro 5

Se observó que los la cantidad de ciclos no bajó demasiado para la versión que utiliza *min-heap*, además las relaciones obtenidas para los valores teóricos y lo obtenidos no son coherentes con lo esperado. La relación teórica se calculó como

$$(V_x + E_x)\log(V_x)/(V_y + E_y)\log(V_y) = (1{,}4)\dot{V}_x\log(V_x)/(1{,}4)\dot{V}_y\log(V_y)$$

donde V_x y V_y son la cantidad de nodos de las entradas utilizadas.

	# nodos	# ciclos con heap	# ciclos sin heap
E1	8	19 247	181 410
E2	50	126 910	3 001 624
E3	75	248 509	7 990 348
E4	100	413 025	16 694 398
E5	250	2 327 314	196 689 026
E6	500	9 012 278	1 411 192 186
E7	750	20 078 698	4 581 173 954
E8	1000	35 513 522	10 644 106 747

Cuadro 5: Resultados para matriz esparza con densidad 40 %

	# relacion teorica	# relacion obtenida
E3/E2	1.65	1.95
E4/E3	1.42	1.66
E5/E4	2.99	5.6
E6/E5	2.25	3.87
E7/E6	1.6	2..22
E8/E7	1.39	1.76

Cuadro 6: Comparación resultados implementación con heap para matriz con densidad 40 %

2. Conclusiones

Se implementaron dos versiones de la algoritmo de Dijkstra, una que utiliza una cola de prioridad y otra que no. Para todas las pruebas realizadas funcionó mejor la versión que utiliza la cola de prioridad.

Se vio que los resultados obtenidos para matrices densas fueron de acuerdo a lo esperado. Para el caso en que la matriz es esparza, el algoritmo no se comportó como se esperaba. Se hicieron pruebas con diferentes valores de densidad de la matriz de adyacencia y los resultados fueron similares.