# Análisis del algoritmo Dijkstra usando Binomial y Fibonacci Heap

Luigy Machaca Arcana
Gabriel Aparicio Tony
Universidad Nacional de San Agustín
Escuela Profesional de Ciencia de la Computación
Análisis y diseño de algoritmos

June 8, 2015

**Abstract**

Los heaps son estructuras de datos de prioridad de cola, en dónde la extracción del elemento (máximo o mínimo) es siempre $\theta(1)$. Existen distintas formas de implementar este tipo de estructuras, algunas de estas son binomial y los fibonacci heap. En el artículo se analizará algunas definiciones preliminares sobre dijkstra, binomial heap y fibonacci heap, asi como se experimentará acelerando el algoritmo dijkstra con binomial y fibonacci heap respectivamente. También incluye los resultados y conclusiones de un experimento realizado para reforzar el análisis previo.

## 1 Introducción

Muchas veces necesitamos usar estructuras parcialmente ordenadas para recurrir al o a los elementos con mayor prioridad. Por ejemplo para el algoritmo de Dijkstra es necesario tener acceso inmediato a la arista con menor peso, por lo que las aristas deben estar almacenadas en una estructura parcialmente ordenada. Otro caso se da cuando queremos rankear los elementos, de tal forma que podemos obtener un top 5, top 10, etc. ya que solo nos importaría los primeros elementos.
Así como estas aplicaciones, existen muchas otras que necesitan de estructuras parcialmente ordenadas. Entre este tipo de estructuras se encuentran las priority queues, tales como listas enlazadas, binary heaps, binomial heaps, fibonacci heaps, relaxed heaps, etc. [3].

Por otra parte Dijkstra es un algoritmo el cual partiendo de un nodo inicial, encuentra todos los caminos mas cortos, hacialos demás nodos.

El objetivo de este articulo es analizar los algortimos de dijkstra (1) sin mejora alguna, segundo dijkstra con mejora de un heap binomial (2) y tercero con mejora de un heap fibonacci (2).

# 2 Conceptos preliminares

## 2.1 Priority Queue

*Priority Queue es una estructura de datos fundamental que permite el acceso solo al elemento mínimo o máximo* [1] (según sea el orden). Una implementación de las priority queue es el Heap. Deben soportar por lo menos las siguientes operaciones: Insertar, Encontrar menor o mayor, Eliminar menor o mayor.

## 2.2 Heap

Un Heap es una estructura de datos especializada basada en el concepto de árbol, que satisface la siguiente propiedad: Para todo nodo A cuyo padre es P, A debe estar ordenado respecto a P. Entre algunos tipos de Heap estan: Binary Heap, **Binomial Heap**, **Fibonacci Heap**, Radix Heap, d-ary Heap, Soft Heap, etc. El Min-Max Heap es un caso especial, ya que deja de ser un árbol.

## 2.3 Binomial Heap

BINOMIAL TREE: Un árbol binomial es un árbol ordenado definido recursivamente, como sigue: Sea $B_k$ un árbol binomial

- $B_0$ es un sólo nodo
- Para $k > 0$ , $B_k$ consiste en dos $B_(k-1)$'s que están unidos entre sí de manera que la raíz de uno es el hijo izquierdo de la raíz de la otra

Un Binomial Heap es un conjunto de árboles binomiales tales que:

- Cada árbol binomial es un árbol parcialmente ordenado, es decir la clave de todo nodo es mayor(menor) o igual que la de su padre.
- Contiene no más de un árbol binomial $B_i$ para cada grado $i$

La mayoría de operaciones del binomial heap están basadas en la operación de unión. Este algoritmo utiliza dos funciones: Unión de dos árboles binomiales de un mismo grado y una operación mezclar, que se encarga de enlazar los nodos del primer nivel de dos heaps de acuerdo al orden de sus grados.
La operación de inserción utiliza la operación de unión, simplemente aplicándola al heap y a un nodo creado a partir de la llave. Un ejemplo es la figura 1.

Figure 1: Ejemplo de Árboles binomiales

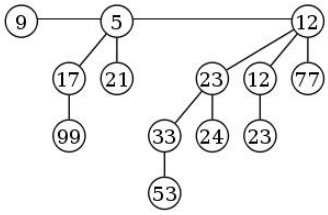
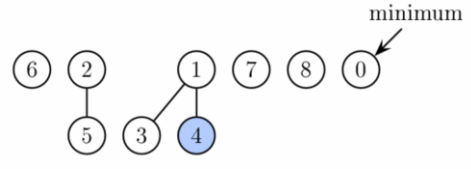



Figure 2: Ejemplo de Árboles binomiales




## 2.4 Fibonacci heap

Es una coleccion de árboles que satisfacen la propiedad del del min-heap. [4]

- es similar a un Binomial-Heap, pero es una estructura menos rigida
- Un Binomial-Heap al insertar ordena tus nodos, un Fibonacci-Heap no lo hace
- Fibonacci solo ordena sus nodos, cuando delete-min es llamado
- Tiene una funcion potencial: $\varphi(H) = tree(H) + 2 * marks(H)$

A diferencia del Binomial-Heap, la mayoría de operaciones no están basadas en la operación de unión. Este algoritmo solo usa la operacion Union-heap cuando Delete-min es llamado.
Un ejemplo es la figura 2.

## 2.5 Comparación Heap

En la siguiente tabla se compara el costo computacional en el peor caso de las principales funciones en cada. Heap [1]

| Función | Fibonacci Heap (Peor caso) | Binomial Heap (Peor Caso) |
|---|---|---|
| Insert | (1) | $\log(n)$ |
| Decrease key | (1) | $(\log(n))$ |
| Encontrar mínimo | (1)relajado | (1)relajado |
| Extraer mínimo | $\log(n)$ | $\log(n)$ |
| Union | (1) | $(\log(n))$ |

## 2.6 Dijkstra

También llamado **algoritmo de caminos minimos**, este algoritmo sirve para determinar el camino mas corto hacia todos los demas vertices, en otras palabras dado los vertices $(i)\epsilon G$, donde $G$ representa los vertices del grafo, $!E$ un camino minimo $c$, de $i$ hacia $f$, para todo $(f)\epsilon G$-{i}.
Siendo la idea principal hallar todos lo caminos minimos de $i$ hacia todos los demas elemnento del grafo $G$.
En el algoritmo [**1**] se expone el pseudocódigo.

**Algorithm 1** Algoritmo Dijkstra

**procedure** DIJKSTRA($G, a$ : G es un grafo bidireccional, $a$ es nodo incial ) ▷ todos los pesos de las aristas son positivos
  **for** $i = 1$ **to** n **do**
    $L(v_i) = \inf$
  $L(a) = 0$
  $S = \emptyset$
  **while** $z \notin S$ **do** ▷ z es un nodo ya visitado
    $u = a$ ▷ $a$ no es vertice de $S$ y es minimo $L(u)$
    $S = S \bigcup \{u\}$
    **for** todos los vertices $v \notin S$ **do** ▷ **extracción del minimo**
      **if** $L(u) + W(u, v) < L(v)$ **then**
        $L(v) = L(u) + W(u, v)$
  **return** $L(z)$ ▷ retorna los caminos mas cortos

Podemos notar que en el algoritmo [**1**] en la linea 8, que la extracción del minimo es $O(v)$,siendo esta una busqueda secuencial. Es posible mejorar esta extracción usando algun **heap**, este algoritmo es mostrado en [**2**], teniendo en este caso un tiempo del **extract-min** del heap usado.

### 2.6.1 compraciones

- **Dijkstra con busqueda secuencial**
  extract-min demora en $v$ vertices $O(v)$, sabiendo que extract-min se encuentra contenido en **while** convierte este tiempo en $O(v^2)$, como existen $E$ adyacentes, entonces el bucle itera $E$ veces con $O(1)$, osea el tiempo tomado por el algoritmo es $O(v^2 + E)$, osea $O(V^2)$.

- **Dijkstra con binomial heap**
  insert en un binomial en $v$ vertices $O(log(v))$, como existen $E$ adyacentes, la inserción demora un total de $O(E * lg(v))$.

- **Dijkstra con fibonacci heap**
  insert en un fibonacci en $v$ vertices $O(1)$, como existen $E$ adyacentes, la inserción demora un total de $O(E * (1))$, osea $O(E)$.

**Algorithm 2** Algoritmo Dijkstra

**procedure** DIJKSTRA($G, a$ : G es un grafo bidireccional, $a$ es nodo incial ) ▷ todos los pesos de las aristas son positivos
  **for** $i = 1$ **to** n **do**
    $L(v_i) = \inf$
  $L(a) = 0$
  $S = \emptyset$
  **while** $z \notin S$ **do** ▷ z es un nodo ya visitado
    $u = a$ ▷ $a$ no es vertice de $S$ y es minimo $L(u)$
    $S = S \bigcup \{u\}$
    $L(v) = extraer_m inimodev \notin S$ ▷ **extracción del minimo**
    **if** $L(u) + W(u, v) < L(v)$ **then**
      $L(v) = L(u) + W(u, v)$
  **return** $L(z)$ ▷ retorna los caminos mas cortos

# 3 Experimentos

## 3.1 Especificación de Entorno

Los experimentos se realizaron en una computadora con las siguientes características: Memoria de 7.8 GiB. Procesador: Intel Core i7-3770 CPU @ 3.40GHz x 8
Los algoritmos se implementaron en el lenguaje

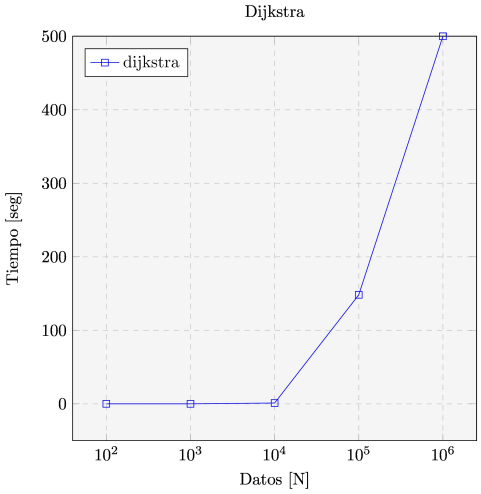
de programación C++, sobre el Sistema Operativo Linux en la distribución Ubuntu 14.04 LTS 64-bit

### 3.2 Resultados

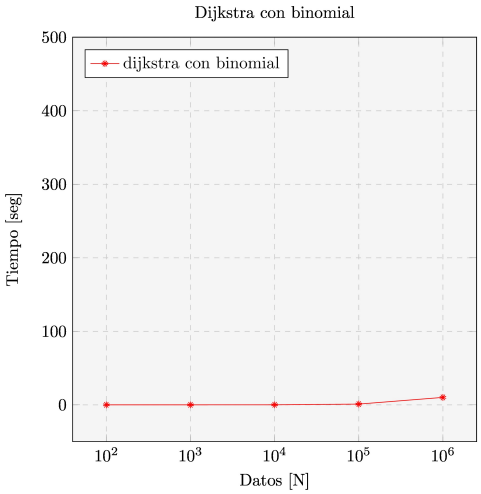
Los datos obtenidos sobre un grafo, son resultado de la inserción de $n$ vertices con 8 vertices aleatorios cado uno. Se utilizó la función *gettimeofday* para calcular los tiempos de ejecución. Es necesario aclarar que se insertaron random pesos en las aristas. Los tiempos de ejecución fueron calculados en microsegundos.

- En la Figura (3) se puede observar la gráfica de estos resultados de **Dijkstra sin cambios**.
- En la Figura (3) se puede observar la gráfica de estos resultados de **Dijkstra con Binomial**.
- En la Figura (3) se puede observar la gráfica de estos resultados de **Dijkstra con Fibonacci**.
- En la Figura (4) se puede observar la gráfica de estos resultados de **Comparacion de resultado**.

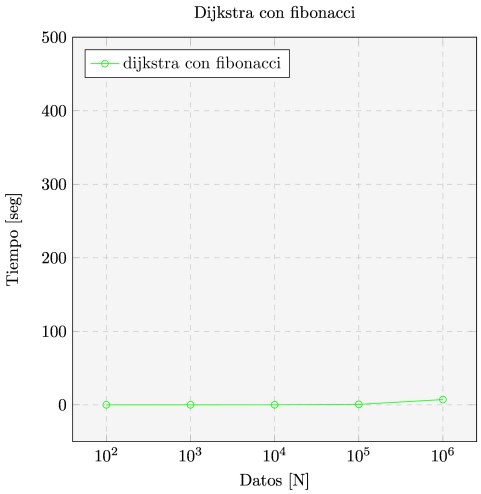
Figure 3: Resultados




(a) dijkstra




(b) dijkstrabinomial




(c) dijkstrafibonacci

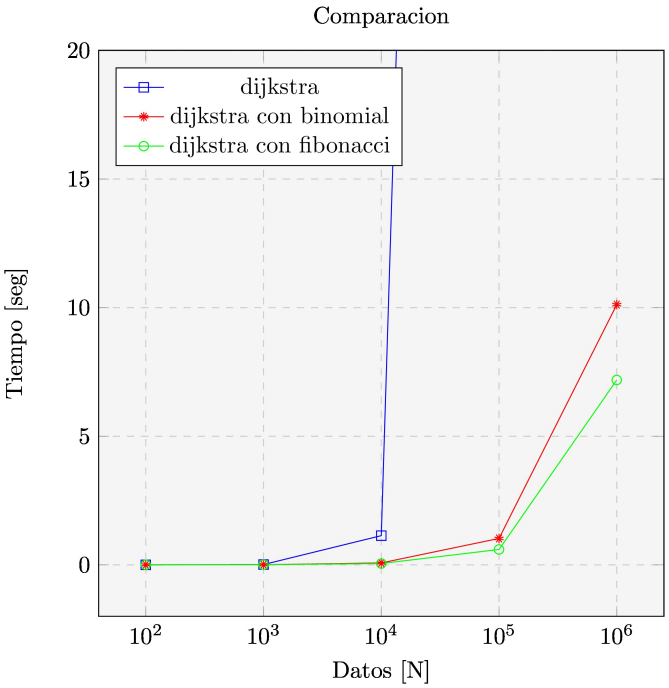
## 4 Conclusiones

Las tres variaciones del algoritmo dijkstra poseen similar comportamiento para un grafo menor a $10^3$ nodos. Con un grafo mayor $10^3$ nodos la diferencianse hace notar, entre el dijkstra tradicional y los acelerados por algun heap, Las variacones entre un Dijkstra+Binomial y un Diksjtra+Fibonacci, cobran mayor importancia en la inserción, siento el primero de orden $0(\log(n))$ y la segunda (1) en cada inserción de los nodos adyacentes en cada iteracion del *while*. Razón por la cual podemos concluir que para valores mayores a $10^6$ un dijkstra+Fibonacci hara notar su rapidez.

## References


[1] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, Cliford Stein. *Introduction to Algorithms*. The MIT Press, Cambridge, Massachusetts, 3rd Edition, 2009.

Figure 4: compracion de resultados





[2] Kevin Wayne, *Robert Sedgewick - Algorithms - Fourth edition*. Princeton University, 2012.

[3] Kevin Wayne, *Binary and Binomial Heaps - Theory of Algorithms*. Princeton University, 2002.

[4] Rezaul A. Chowdhury, *Lectures 13 y 13 (dijkstra and fibonacci heap)*. Department of Computer Science - SUNY Stony Brook 2012.