Colas con priopidad (Heaps)

Luigy Alex Machaca Arcana
Luigy.mach.arc@gmail.com
Universidad Nacional de San Agustín
Escuela Profesional de Ciencia de la Computación
Análisis y Estructura de Datos

April 9, 2014

Abstract

Los heaps son estructuras frecuentemente usadas para implementar colas de prioridad, en dicha estructura la extracción del menor dato es mas rápida que en otras estructuras. Existen diferentes formas de implementar este tipo de estructuras, algunas de ellas son *binary heap, binomial heap, fibonacci heap*. Se explicará conceptos y algunas funciones sobre estas estructuras.

1 Introducción

A veces necesitamos usar estructuras parcialmente ordenadas para recurrir al elemento con mayor prioridad. Por ejemplo para el algoritmo de Kruskal y Prim para el cálculo del árbol de expansión mínimo de un grafo etiquetado, ó para el algoritmo de Dijkstra para el cálculo de caminos mínimos en un grafo etiquetado, por lo que las aristas deben estar almacenadas en una estructura parcialmente ordenada. Otro caso se da cuando queremos establecer orden de prioridad en un grupo de elementos, de tal forma que podemos obtener un cierto numero de elementos con la mayor prioridad. Entre este tipo de estructuras se encuentran las *priority queues* [1], tales como listas enlazadas, binary heaps, binomial heaps, fibonacci heaps.[2] Las estructuras a analizar en este artículo son: Binary Heap, la cuales estan basadas en un árbol binario; y Binomial Heap, que está basada en un arboles binomiales. Un dato extra es que es posible ordenar elementos haciendo uso de las funciones de heaps.

2 Conceptos previos

2.1 Priority queue

Priority Queue es una estructura de datos fundamental que permite el acceso solo al elemento mínimo. [1] Una implementación de las priority queue es el Heap, que debe soportar inserción, consultar menor y extraer menor.

[1] **Una cola de prioridades** que es una estructura de datos en la que los elementos se atienden en el orden indicado por una prioridad asociada a cada uno

2.2 Heaps

Un Heap es una estructura de datos basada en árboles (*se utilizan para el ordenamiento de datos HeapSort y para implementar colas de prioridades*), estas satisface la siguiente propiedad: Para todo nodo A cuyo padre es P, P debe tener mayor prioridad que A.

2.3 Binary Heap

Un binary heaps es un caso particular y bastante sencillo, dicha estructura esta basada en un árbol binario parcialmente ordenado, que puede verse como un árbol binario con dos propiedades(Restricciones) adicionales[3] :

- Cada nodo tiene un valor mayor o menor (orden ascendente u orden descendente) que cualquiera de sus hijos.
- Este árbol binario tiene que tiene que ser completo, es decir debe tener todos sus nodos, con excepción de que tal ves no existan todos los nodos en el último nivel.

Cumpliendo estas anteriores propiedades el Binary Heap se puede representar a través de un vector como se muestra en la Figura 1. A partir de esto se puede deducir que:

- Para cada elemento en la posicion i [2] .
- Sus hijo deben de estar(dentro del array), en las posiciones **2k** y **2k+1**, siendo el primero para hijo izquierdo y el segundo para el derecho respectivamente.
- para encontrar el padre de k solo se debe realizar una división entera del índice k/2.

Para el binary heap una de las operaciones más importantes es **Heapify**, que se encarga de hacer que se cumpla la propiedad de los Heaps En el Algoritmo (1) se expone el pseudocódigo. Para extraer la cabeza del heap, simplemente se intercambia, el último elemento del heap con el primero para luego eliminar este último y aplicar Heapify(1) y todo el árbol.

Otra de las operaciones importantes es la interseción, la cual añade el elemento al final del árbol para luego hacerlo flotar [3] hasta que llegue a la posición correcta.

En el Algoritmo (2) se expone el pseudocódigo.

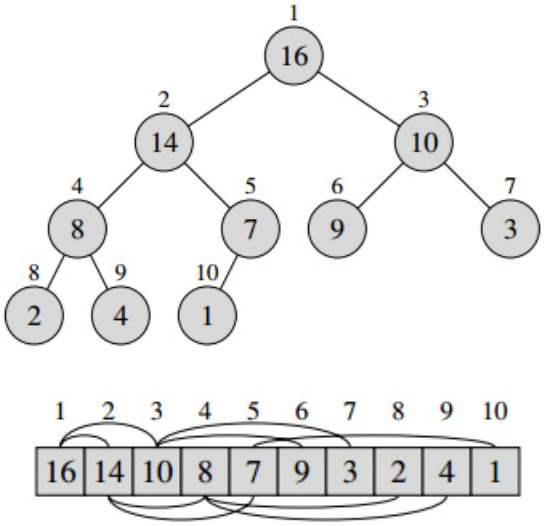
3 Binomial Heap

Estan basados en arboles binomiales, los cuales son árboles definidos recursivamente. Si B_k es un árbol binomial, B_0 será un solo nodo y para $k > 0$, B_k tendrá dos $B_{(k-1)}s$ que estarán unidos entre sí de tal manera que la raíz de uno es el hijo izquierdo de la raíz de la otra. Algunos ejemplos en la figura 2.

[2] como **i** es la poscion siempre sera un entero.

[3] por flotar, revisar que arbol cumple las condiciones necesarias

Figure 1: Max Heap, representado a través de un array



Algorithm 1 Heapity

Require: *data* (Vector de datos) , *i* (Posición del nodo a calcular) , *cmp* (Función de Comparación)

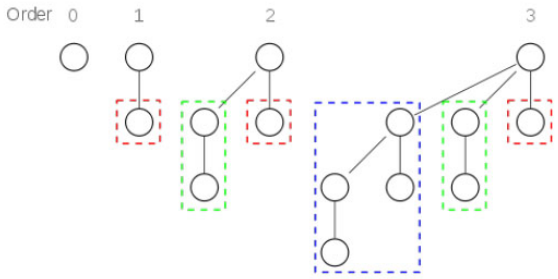
$l \leftarrow 2 * i$
$r \leftarrow 2 * i + 1$
$temp \leftarrow i$
if $l < n$ AND $cmp(data[l], data[temp])$ **then**
  $temp \leftarrow l$
end if
if $r < n$ AND $cmp(data[r], data[temp])$ **then**
  $temp \leftarrow r$
end if
if $temp = i$ **then**
  $return$
end if
$swap(data[i], data[temp])$
$Heapify(data, temp, cmp)$

Algorithm 2 insert

Require: *data* (Vector de datos) , *key* (Dato a insertar) , *cmp* (Función de Comparación)

$data.push_back(key)$
for $j = data.size() - 1; j > 0$ AND $cmp(data[j], data[j/2]); j = j/2$ **do**
  $swap(data[j], data[j/2])$
end for

Figure 2: Arboles binomiales



El binomial heap es un conjunto de arboles binomiales parcialmente ordenados, y contiene no ,ás de un árbol binomial para cada grado del heap. Las operaciones de este heap estan basadas en la operación de unión. La inserción es la unión de un heap con otro heap con un solo nodo. Al momento de enlazar los árboles se debe mantener las prioridades.

4 Comparacíon

Ambos Heaps tienen el mismo costo computacional de todas sus funciones con un excepcion de la unión. A continuación se presenta la tabla de comparación de costos computacionales en el peor y mejor caso entre binary y binomial heap.

Función	Binary Heap (Peor caso)	Binomial Heap (Peor Caso)
Insert	$\log(n)$	$\log(n)$
Encontrar mínimo	1	$\log(n)$
Extraer mínimo	$\log(n)$	$\log(n)$
Union	n	$\log(n)$

References

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

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

[3] Antonio Garrido Carrillo y Joaquin Valdivia , Antonio Garrido Carrillo, Joaquín Fernández Valdivia *Abstracción y Estructuras de datos en C++* Delta Publicaciones, 2006 .

[4] Alonso Ramirez Manzanares *Colas de prioridad (priority queues)* - Computación y Algoritmos, 2012.