Heap. Cola de Prioridad

Objetivos

- Profundizar en el conocimiento de la estructura de datos `heap`, sus algortimos, propiedades y aplicaciones.
- Analizar y desarrollar especificaciones e implementaciones del TAD *Cola de prioridad*.
- Implementar variantes del TAD *Cola de prioridad* para la resolución de problemas simples.

Se asume definido el tipo `uint` que representa a los enteros no negativos.

Sin pérdida de generalidad se asume, a menos que se diga de manera explícita lo contrario, que en los heaps el primer elemento del arreglo es menor o igual al resto y que en las colas de prioridad el elemento prioritario tiene asociado un valor de prioridad menor o igual que el resto.

Heap

Ejercicio 1 Similar a Segundo Parcial 2018

(a) Describa las dos propiedades que debe cumplir un árbol binario para ser un `heap` (montículo binario).

(b) Dibuje el `heap` resultante de insertar los números 6, 7, 8, 9, 1, 2, 3, 2, 5, 14, 12 en ese orden y a partir de un heap vacío. Asuma que el orden de los elementos está establecido por la relación $\leq$.

(c) Dibuje el `heap` resultante de eliminar el mínimo elemento 2 veces a partir del `heap` resultante de la parte (b).

Ejercicio 2 Estructura de los árboles completos

(a) Demuestre que si a un nodo le corresponde en el arreglo la posición i, entonces, en caso de existir, a sus hijos les corresponden las posiciones $2i$ y $2i + 1$ y a su padre la posición $\lfloor i/2 \rfloor$.

(b) Suponga que la correspondencia entre los nodos y las posiciones del arreglo empiezan en la posición 0, en vez de en la 1. Esto es, a la raíz del árbol le corresponde la posición 0. Si a un nodo le corresponde la posición i, ¿qué posiciones, en caso de existir, les corresponden a sus hijos y a su padre?

(c) Demuestre que si el árbol tiene n nodos la cantidad de hojas es $\lceil n/2 \rceil$.

Ejercicio 3 Filtrado ascendente y filtrado descendente

Suponemos que tratamos con árboles completos de n elementos que se implementan en las posiciones $[1..n]$ de un arreglo. Los elementos del árbol son de un tipo genérico `T` para el cual están definidos los operadores relacionales.

(a) Escriba versiones recursiva e iterativa del algoritmo *filtrado ascendente* aplicado en una posición que pertenece al rango $[1..n]$.

(b) Escriba versiones recursiva e iterativa del algoritmo *filtrado descendente* aplicado en una posición que pertenece al rango $[1..n]$.

(c) Determine el orden de crecimiento del tiempo de ejecución de los algoritmos.

(d) Escriba un algoritmo que inserte un elemento y otro que elimine el mínimo de un `heap`.

(e) Suponga que el mismo elemento se inserta más de una vez. Demuestre que tras sucesivos llamados a eliminar el mínimo esos elementos se eliminan en el mismo orden en que se insertaron o muestre un contraejemplo de que eso no se cumple.

Ejercicio 4 Construir montículo

(a) Diseñe algoritmos que dado un arreglo dispuesto de manera genérica lo transforme en un `heap`. Escriba una versión basada en *filtradoAscendente* y otra basada en *filtradoDescendente*. Se puede demostrar que para la segunda de ellas se puede diseñar una implementación con orden de tiempo de ejecución $O(n)$.

(b) Implemente los algoritmos anteriores y compare sus tiempos de ejecución.

Sugerencia: use la función `clock` de la biblioteca `time.h` para medir tiempos como en `principal.cpp` de la Tarea 3, y las funciones `srand` y `rand` de la biblioteca `stdlib.h` para generar números seudo-aleatorios.

(c) Usando alguno de los algoritmos de la parte (a) diseñe un algoritmo que dado un arreglo lo deje ordenado de manera descendente.

(d) Diseñe un algoritmo que dado un arreglo de n elementos encuentra el k-ésimo mayor. El orden de crecimiento del tiempo de ejecución debe ser $O(n \log k)$, siendo n la cantidad de elementos del `heap`.

Ejercicio 5 Buscar

(a) Diseñe un algoritmo que busque un elemento dado de tipo `T` en un `heap`. ¿Se puede utilizar la propiedad de orden del `heap` para disminuir la cantidad de comparaciones que realiza el algoritmo? En caso afirmativo, utilícela en el algoritmo.

(b) Analice el tiempo de ejecución del algoritmo. Describa cuándo se da el peor caso.

Ejercicio 6 Decrementar e Incrementar

Suponga que los elementos de un `heap` tienen dos campos. Uno de ellos, `dato`, toma valores en el rango $[0..M-1]$, siendo M una constante conocida, y se puede asumir que en el `heap` no hay dos elementos con el mismo valor de `dato`. El otro campo, `orden`, es el que determina el orden del `heap`. A las operaciones de `heap` se quiere agregar otra que, a un elemento identificado por su valor `dato` le modifica su valor `orden` y restablece la condición de orden del `heap`.

Adapte las estructuras y algoritmos para que el orden de crecimiento del tiempo de ejecución en el peor caso de la nueva operación sea, igual que el de las operaciones que insertan y eliminan elementos, $O(\log n)$.

Cola de Prioridad

Ejercicio 7 Especificación

Desarrolle una especificación funcional/procedural para el TAD **Cola de Prioridad** de elementos de tipo `T`, donde las prioridades son números enteros.

(a) Escriba una especificación con mínima cantidad de operaciones para las colas de prioridad no acotadas (sin restricciones de capacidad).

(b) ¿Qué modificaciones deberían hacerse a la especificación anterior para obtener una especificación del TAD **Cola de Prioridad acotada**?

Ejercicio 8 Implementaciones

(a) Desarrolle implementaciones completas del TAD **Cola de Prioridad**, utilizando:

I. Listas ordenadas de nodos enlazados para la versión no acotada del TAD.

II. Árboles Parcialmente Ordenados implementados sobre un arreglo (Montículo o Heap) para la versión acotada del TAD.

(b) Explique como implementar colas de prioridad no acotadas para que se cumplan los siguientes tiempos de ejecución en el peor caso.

I. Insertar y obtener el prioritario en $O(1)$.

II. Insertar y eliminar el prioritario en $O(\log n)$; obtener el prioritario en $O(1)$.

Justifique por qué se cumplen esos órdenes y exlique cuáles son los órdenes de las otras operaciones.

(c) Compare las implementaciones según el tiempo de ejecución de **todas** las operaciones.

Ejercicio 9 Extensiones

Asuma que a la especificación de Cola de Prioridad vista en el Ejercicio 7 se agrega la siguiente operación:

```
/* Decrementa el valor 'restar' a las prioridades de todos los
   elementos de 'cp'. */
void decTodas(uint restar, ColaPrio cp);
```

¿Es posible adaptar alguna de las implementaciones de Cola de Prioridad del Ejercicio 8 para que se pueda implementar `decTodas` sin recorrer todos los elementos? En caso negativo demuestre que no es posible. En caso afirmativo explique que impacto tiene sobre la implementación y sobre el resto de las operaciones y analice el tiempo de ejecución de la operación.

Ejercicio 10 Sistema de Impresión

Considere un sistema de administración de impresión que procesa documentos, de tipo `string`, con prioridades en el rango $[1 : K]$, con $K = 10$, constante que se puede usar en el código. Cuando llega un documento a imprimirse, se agrega a la colección de documentos que esperan ser impresos (en particular, la colección puede estar vacía). En el momento de imprimir se selecciona el documento prioritario, se devuelve y se elimina de la colección. Dadas dos strings s y t con prioridades i y j respectivamente, diremos que s tiene prioridad frente a t si y solo si $i > j$, o si $i = j$ y s es más antiguo que t.
Asuma que el `string` es un tipo primitivo, para el cual pueden usarse los operadores habituales de comparación y asignación.

Se pide:

(a) Especifique el TAD `ColaImpresora` de elementos de tipo `string` y con prioridades en el rango $[1 : K]$ que permita resolver adecuadamente el sistema de administración de impresión previamente descrito.

(b) Proponga una implementación del TAD `ColaImpresora` en la cual las operaciones de inserción, recuperación y eliminación de elementos sean $O(1)$ peor caso. Desarrolle la representación del TAD e impleméntelo completamente (puede usar otros TADs). Explique gráficamente la representación usada antes de implementar las operaciones.