Implementación

Pregunta 01

Implementación del heap-montículo ternario: Parte asignada a Diego, se implementa una primera versión y José revisa y pone test al programa.

La estuctura contiene los siguientes datos:

- vec : [int] — colección de enteros
- size : int — tamaño actual del vector
- capacity : int — capacidad total de memoria para el vector

Inicializar la estructura THeap como THeap *h = NULL; y después agregar un elemento cualquiera.

Pregunta 02

Complejidad O(log_3(n)): (n = size = tamaño actual del arreglo)

Se accede trivialmente al máximo. Para el insert, se coloca el nuevo elemento en O(1) y se realiza el heapify que en el peor caso es hasta el principio, el cual está a distancia log_3(n). Similarmente para el remove, se reemplaza el máximo por el último en el arreglo en O(1) y se realiza otro heapify hacia abajo que de igual manera, el peor caso se alcanza en log_3(n).

Comparación con montículo binario: A priori, se supera la complejidad respecto al binario debido a que se necesitan menos pasos para realizar el heapify. Es una ventaja contar con menos niveles de profundidad por lo anterior. Sin embargo, se ocupa más memoria para tener niveles profundos en el árbol.

Aplicación

Pregunta 03

Aplicación para mónticulos: Cálculo de la mediana en streaming. Parte asiganada a José, se implementa una primera versión y Diego revisa y pone test al programa. La siguiente aplicación tiene las siguientes dependencias:

"heap.c" → se implementa la estructura de montículo ternario "reverseheap.c" → modificación de las funciones en heap.c

El cálculo de el valor mediano toma como entrada 2 montículos: thmin y thmax, la variable de la mediana "median" está definida como global.

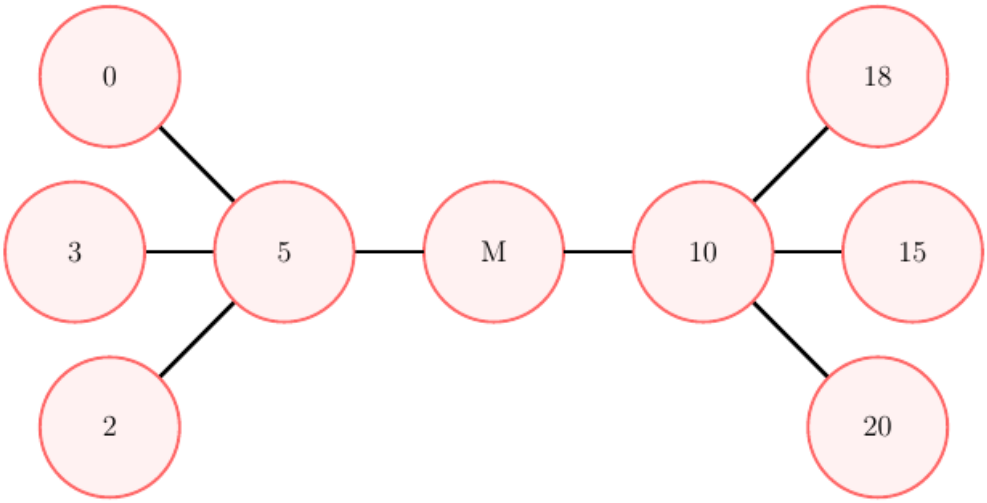
Pregunta 04

Para la implementación de este algoritmo, se requieren dos montículos diferentes: un montículo thmax que almacenará la mitad inferior de los datos (menores), y thmin que guardará la mitad superior de los datos (mayores).

Por la forma de implementación, thmax funciona como una cola de prioridad tomando como key los valores más grandes, es decir, el dato almacenado cuyo valor numérico es el más alto se encuentra sobre la raíz. Similarmente, thmin toma como key los valores más pequeños, siendo el de menor valor numérico el que se encuentra sobre la raíz.

Si alguno de los dos montículos es más grande que el otro, la mediana se encontrrará en la raíz de mayor tamaño (pues representa el valor que se encuentra a la mitad ordénando los datos en orden creciente). Caso contrario, la mediana será el valor promedio entre las dos raíces.

La siguiente figura muestra como se organizan los datos en cada montículo; cuando ambos montículos son del mismo tamaño y se añade un nuevo nodo, este se reacomodará en su heap correspondiente (si es mayor a la mediana anterior en se sitúa en el thmin, si es menor en el thmax, los detalles se encuentran en el documento de la tarea original), la raíz del montículo más grande corresponderá a la nueva mediana. Si después de añadir un nuevo dato, ambos heaps tienen el mismo tamaño, la mediana corresponderá al promedio entre las dos raíces, Nótese que no importa cual sea el orden de las hojas o nodos hijos, los datos de interés son los que se encuentran en la parte media.



Pregunta 05

Análisis de complejidad:

En el peor de los casos, se debe remover la raíz de uno de los montículos para colocarlo sobre el otro, y se debe agregar el dato de entrada en el montículo con menos elementos.

Suponiendo que el tamaño de thmax es mayor al de thmin, y el dato es menor a la mediana actual, se debe colocar el valor máximo de thmax en thmin, para ello, se utilizan las funciones de remover e insertar.

La acción de remover el valor máximo de thmax, en el peor caso, deberá realizar la función heapify en T(n) = log_3(n). (recordar que el último dato del arreglo se intercambia por el primero y a partir de ahi, se trata de encontrar el lugar para este elemento).

El nuevo valor insertado en thmin tendrá que estar en la raíz, la cual se encuentra a una distacia de log_3(n). La función heapify tiene un tiempo de ejecución de T(n) = log_3(n).

Se deberá incluír el nuevo dato de entrada en el montículo de menor tamaño (en este caso thmax), por lo que, si este corresponde a la raíz, se encontrará a una distancia de log_3(n), de igual manera T(n) = log_3(n).

Las operaciones para la asignación de variables auxiliares y la modificación de el valor mediano, se realizan en tiempo constante (T(n) = c).

En total, el tiempo de ejecución de este algoritmo es:

T(n) = 3log_{3}(n) + c

Por lo que:

T(n) = O(log(n)).

Resultados obtenidos

La siguiente figura muestra la salida en la terminal utilizando como flujo de entrada los datos que se encuentran en input1.txt e input2.txt respectivamente

```
~/Documentos/cimat/Programacion/tarea8/pai-2020-tarea-08/DiegoM-JGonzaloP (6aed90e ✔) ▷ make all
rm -rf *.app *.tmp
gcc -g -Wall -Wpedantic -lm main.c -o main.app
./main.app < input.txt > output.tmp
diff --side-by-side --report-identical-files output.txt output.tmp
1.000                                                           1.000
2.500                                                           2.500
4.000                                                           4.000
3.500                                                           3.500
Files output.txt and output.tmp are identical
./main.app < input2.txt > output2.tmp
diff --side-by-side --report-identical-files output2.txt output2.tmp
1.000                                                           1.000
3.000                                                           3.000
5.000                                                           5.000
33.000                                                          33.000
61.000                                                          61.000
36.500                                                          36.500
12.000                                                          12.000
8.500                                                           8.500
5.000                                                           5.000
4.000                                                           4.000
3.000                                                           3.000
3.000                                                           3.000
3.000                                                           3.000
2.500                                                           2.500
3.000                                                           3.000
2.500                                                           2.500
2.000                                                           2.000
```

de igual manera, se presenta la captura de los resultados de los tests propuestos en specs.c

```
Tamaño heap1: 1
Tamaño heap2: 2
Tamaño heap3: 3
Valor despues de la liberacion: 1492595139
Valor despues de la liberacion: 736822115
Valor despues de la liberacion: 736822019
Liberacion OK

Prueba de eliminación/inserción  de elementos
Estás tratando de remover de un montículo vacío
Estás tratando de remover de un montículo vacío
Esto debe imprimir un 5 -> 5
Prueba exitosa

Al probar un elemento se debe mantener la propiedad del heap:
Prueba exitosa: se mantiene la propiedad del heap

Total de pruebas realizadas: 3
Total de pruebas exitosas: 3
Total de pruebas fallidas: 0
gcc -g -Wall -Wpedantic -lm specsApp.c -o specsapp.app
./specsapp.app

Pruebas para la aplicación de la mediana:

Total de pruebas realizadas: 10
Total de pruebas exitosas: 10
Total de pruebas fallidas: 0
```