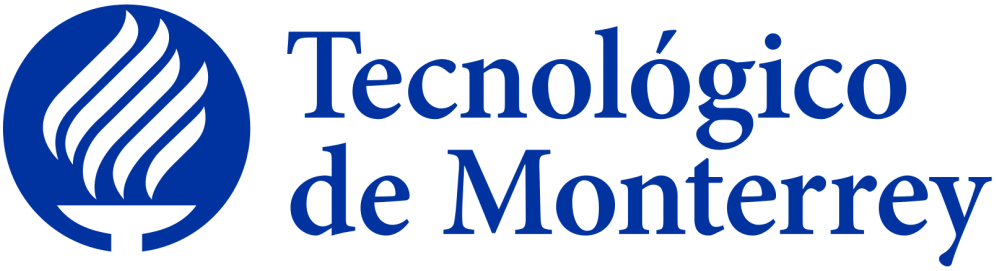
Instituto Tecnológico de Estudios Superiores de Monterrey



Campus Monterrey

Act 3.2 - Árbol Heap: Implementando una fila priorizada

Programación de estructuras de datos y algoritmos fundamentales (Gpo 610)

Eduardo López Benítez

Fidel Morales Briones A01198630

Monterrey, Nuevo León, a 16 de octubre de 2023

Casos Prueba:

Heap 1

```
217     // heap 1
218     priorityQueue *listaPrioridadHeap1 = new priorityQueue();
219     cout << "HEAP 1:" << endl;
220
221     // empty
222     cout << "empty: " << listaPrioridadHeap1->empty() << endl;
223
224     // push
225     listaPrioridadHeap1->push(5);
226     listaPrioridadHeap1->push(3);
227     listaPrioridadHeap1->push(26);
228     listaPrioridadHeap1->push(15);
229     listaPrioridadHeap1->push(25);
230     listaPrioridadHeap1->push(25);
231
232
233     // pop
234     listaPrioridadHeap1->pop();
235
236     // top
237     cout << "top: " << listaPrioridadHeap1->top() << endl;
238
239     // empty
240     cout << "empty: " << listaPrioridadHeap1->empty() << endl;
241
242     // size
243     cout << "size: " << listaPrioridadHeap1->Size() << endl;
244
245     listaPrioridadHeap1->print();
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
HEAP 1:
empty: 1
top: 25
empty: 0
size: 5
25 15 25 3 5
```

Heap 2

```
247     // heap 2
248     priorityQueue *listaPrioridadHeap2 = new priorityQueue();
249     cout << "\nHEAP 2:" << endl;
250
251     // empty
252     cout << "empty: " << listaPrioridadHeap2->empty() << endl;
253
254     // push
255     listaPrioridadHeap2->push(11);
256     listaPrioridadHeap2->push(-12);
257     listaPrioridadHeap2->push(-12);
258     listaPrioridadHeap2->push(13);
259     listaPrioridadHeap2->push(14);
260     listaPrioridadHeap2->push(3);
261     listaPrioridadHeap2->push(13);
262
263     // pop
264     listaPrioridadHeap2->pop();
265
266     // top
267     cout << "top: " << listaPrioridadHeap2->top() << endl;
268
269     // empty
270     cout << "empty: " << listaPrioridadHeap2->empty() << endl;
271
272     // size
273     cout << "size: " << listaPrioridadHeap2->Size() << endl;
274
275     listaPrioridadHeap2->print();
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
HEAP 2:
empty: 1
top: 13
empty: 0
size: 6
13 11 13 -12 3 -12
```

Heap 3

C: > Users > T480 > Desktop > tec > 3ro > algoritmos > arboles > tarea3_2.cpp > main()

```
278     priorityQueue *listaPrioridadHeap3 = new priorityQueue();
279     cout << "\nHEAP 3:" << endl;
280
281     // empty
282     cout << "empty: " << listaPrioridadHeap3->empty() << endl;
283
284     // push
285     listaPrioridadHeap3->push(1);
286     listaPrioridadHeap3->push(1);
287     listaPrioridadHeap3->push(2);
288     listaPrioridadHeap3->push(2);
289     listaPrioridadHeap3->push(3);
290     listaPrioridadHeap3->push(3);
291     listaPrioridadHeap3->push(0);
292     listaPrioridadHeap3->push(0);
293
294     // pop
295     listaPrioridadHeap3->pop();
296
297     // top
298     cout << "top: " << listaPrioridadHeap3->top() << endl;
299
300     // empty
301     cout << "empty: " << listaPrioridadHeap3->empty() << endl;
302
303     // size
304     cout << "size: " << listaPrioridadHeap3->Size() << endl;
305
306     listaPrioridadHeap3->print();
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
HEAP 3:
empty: 1
top: 3
empty: 0
size: 7
3 2 1 1 2 0 0
```

Heap 4

```
308     // heap 4
309     priorityQueue *listaPrioridadHeap4 = new priorityQueue();
310     cout << "\nHEAP 4:" << endl;
311
312     // empty
313     cout << "empty: " << listaPrioridadHeap4->empty() << endl;
314
315     // push
316     listaPrioridadHeap4->push(111);
317     listaPrioridadHeap4->push(222);
318     listaPrioridadHeap4->push(56);
319     listaPrioridadHeap4->push(78);
320     listaPrioridadHeap4->push(0);
321
322     // pop
323     listaPrioridadHeap4->pop();
324
325     // top
326     cout << "top: " << listaPrioridadHeap4->top() << endl;
327
328     // empty
329     cout << "empty: " << listaPrioridadHeap4->empty() << endl;
330
331     // size
332     cout << "size: " << listaPrioridadHeap4->Size() << endl;
333
334     listaPrioridadHeap4->print();
335
336 }
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
HEAP 4:
empty: 1
top: 111
empty: 0
size: 4
111 78 56 0
```

Programa:

```
/*
Act 3.2 - Árbol Heap: Implementando una fila priorizada
Fidel Morales Briones A01198630
16 de octubre de 2023
*/
#include <iostream>
using namespace std;

class priorityQueue {
public:
    int heapArray[15];
    int size = 0;

    priorityQueue() {
    }

    /*
    * push, inserta un nuevo valor dentro del arreglo heap
    *
    * Se agrega un valor dentro del arreglo heap, y se utiliza el
      método swiftUp para acomodar el arreglo
    * para que el valor recién agregado se encuentre en la posición
      correcta
    *
    * @param value: dato a insertar
    * @return: no tiene
    * Complejidad tiempo: O(log(n))
    * Complejidad espacio: O(n)
    */
    void push(int value) {
            heapArray[size] = value;
            size++;
            if (size > 1) {
                swiftUp((size / 2) - 1);
            }
    }

    /*
    * pop, elimina el valor con mayor prioridad dentro del arreglo heap
    *
```

```
    * El valor de mayor prioridad del heap se va hasta el final del
      arreglo, y se
    * utiliza el método swiftDown para acomodar el arreglo
    *
    * @param: no tiene
    * @return: no tiene
    * Complejidad tiempo: O(log(n))
    * Complejidad espacio: O(n)
    */
    void pop() {

        int temp = heapArray[size - 1];
        heapArray[size-1] = heapArray[0];
        heapArray[0] = temp;
        size--;
        if (size > 1) {
            swiftDown(0);
        }
    }

    /*
    * top, regresa el valor con mayor prioridad dentro del arreglo heap
    *
    * regresa el valor en la posición 0 del arreglo
    *
    * @param: no tiene
    * @return: el entero que está en la posición 0 del arreglo
    * Complejidad tiempo: O(1)
    * Complejidad espacio: O(1)
    */
    int top() {
        return heapArray[0];
    }

    /*
    * empty, regresa si el arreglo heap está vacío o no
    *
    * regresa true si el arreglo está vacío, false si no
    *
    * @param: no tiene
    * @return: valor booleano que indica si el arreglo está vacío o no
    * Complejidad tiempo: O(1)
    * Complejidad espacio: O(1)
```

```
*/
bool empty() {
    return size == 0;
}

/*
* size, regresa el tamaño del arreglo heap
*
* regresa el atributo size de la clase, que indica el tamaño del
  arreglo heap
*
* @param: no tiene
* @return: entero que indica el tamaño del arreglo heap
* Complejidad tiempo: O(1)
* Complejidad espacio: O(1)
*/
int Size() {
    return size;
}

/*
* swap, intercambia dos valores dentro del arreglo heap
*
* dos valores se interacambian dentro del arreglo heap, se utiliza
  dentro de los métodos
* swiftUp y swiftDown. Se crea una variable temporal para completar
  el intercambio
*
* @param y: entero que indica la posición del primer valor a
  intercambiar
* @param x: entero que indica la posición del segundo valor a
  intercambiar
* @return: no tiene
* Complejidad tiempo: O(1)
* Complejidad espacio: O(1)
*/
void swap(int x, int y) {
    int temp = heapArray[x];
    heapArray[x] = heapArray[y];
    heapArray[y] = temp;

}
```

```
/*
* swiftUp, ordena el arreglo cuando un valor se agrega al final del
  arreglo
*
* revisa si el valor recién agregado es mayor que su padre, si es
  así, se intercambian,
* así hasta que todos los valores dentor del arreglo estén
  ordenados y cumplan con la
* condicion de un heap
*
* @param rootIndex: índice del último nodo que tiene por lo menos
  un hijo
* @return: no tiene
* Complejidad tiempo: O(log(n))
* Complejidad espacio: O(n)
*/
void swiftUp(int rootIndex) {
    int max = rootIndex;
    int left = (2 * rootIndex) + 1;
    int right = (2 * rootIndex) + 2;

    if (left < size && heapArray[max] < heapArray[left]) {
        max = left;
    }
    if (right < size && heapArray[max] < heapArray[right]) {
        max = right;
    }

    if (max != rootIndex) {
        swap(max, rootIndex);
        swiftUp((rootIndex-1)/2);
    }
}

/*
* swiftDown, ordena el arreglo cuando un valor se elimina del
  inicio del arreglo y el último valor se coloca en el inicio
*
* revisa si el valor recién agregado es menor que su hijo, si es
  así, se intercambian,
* así hasta que todos los valores dentor del arreglo estén
  ordenados y cumplan con la
* condicion de un heap
```

```
*
* @param rootIndex: índice del nodo con mayor prioridad
* @return: no tiene
* Complejidad tiempo: O(log(n))
* Complejidad espacio: O(n)
*/
void swiftDown(int rootIndex) {
    int max = rootIndex;
    int left = (2 * rootIndex) + 1;
    int right = (2 * rootIndex) + 2;

    if (left < size && heapArray[max] < heapArray[left]) {
        max = left;
    }
    if (right < size && heapArray[max] < heapArray[right]) {
        max = right;
    }

    if (max != rootIndex) {
        swap(max, rootIndex);
        swiftDown(max);
    }
}

/*
* print, imprime el arreglo heap
*
* imprime todos los valores dentro del heap
*
* @param: no tiene
* @return: se imprimen los valores en la terminal
* Complejidad tiempo: O(n)
* Complejidad espacio: O(1)
*/
void print() {
    for (int i = 0; i < size; i++) {
        cout << heapArray[i] << " ";
    }
    cout << endl;
}

/*
* print, imprime sola una parte del arreglo heap
```

```
    *
    * dependiendo del parámetro fullSize, imprime esa cantidad de
      valores del arreglo heap
    *
    * @param fullSize: cantidad de valores que se quieren imprimir
    * @return: se imprimen los valores en la terminal
    * Complejidad tiempo: O(n)
    * Complejidad espacio: O(1)
    */
    void print(int fullSize) {
        for (int i = 0; i < fullSize; i++) {
            cout << heapArray[i] << " ";
        }
        cout << endl;
    }

};

int main()
{
    // heap 1
    priorityQueue *listaPrioridadHeap1 = new priorityQueue();
    cout << "HEAP 1:" << endl;

    // empty
    cout << "empty: " << listaPrioridadHeap1->empty() << endl;

    // push
    listaPrioridadHeap1->push(5);
    listaPrioridadHeap1->push(3);
    listaPrioridadHeap1->push(26);
    listaPrioridadHeap1->push(15);
    listaPrioridadHeap1->push(25);
    listaPrioridadHeap1->push(25);


    // pop
    listaPrioridadHeap1->pop();

    // top
    cout << "top: " << listaPrioridadHeap1->top() << endl;

    // empty
```

```
    cout << "empty: " << listaPrioridadHeap1->empty() << endl;

    // size
    cout << "size: " << listaPrioridadHeap1->Size() << endl;

    listaPrioridadHeap1->print();

    // heap 2
    priorityQueue *listaPrioridadHeap2 = new priorityQueue();
    cout << "\nHEAP 2:" << endl;

    // empty
    cout << "empty: " << listaPrioridadHeap2->empty() << endl;

    // push
    listaPrioridadHeap2->push(11);
    listaPrioridadHeap2->push(-12);
    listaPrioridadHeap2->push(-12);
    listaPrioridadHeap2->push(13);
    listaPrioridadHeap2->push(14);
    listaPrioridadHeap2->push(3);
    listaPrioridadHeap2->push(13);

    // pop
    listaPrioridadHeap2->pop();

    // top
    cout << "top: " << listaPrioridadHeap2->top() << endl;

    // empty
    cout << "empty: " << listaPrioridadHeap2->empty() << endl;

    // size
    cout << "size: " << listaPrioridadHeap2->Size() << endl;

    listaPrioridadHeap2->print();

    // heap 3
    priorityQueue *listaPrioridadHeap3 = new priorityQueue();
    cout << "\nHEAP 3:" << endl;

    // empty
    cout << "empty: " << listaPrioridadHeap3->empty() << endl;
```

```
// push
listaPrioridadHeap3->push(1);
listaPrioridadHeap3->push(1);
listaPrioridadHeap3->push(2);
listaPrioridadHeap3->push(2);
listaPrioridadHeap3->push(3);
listaPrioridadHeap3->push(3);
listaPrioridadHeap3->push(0);
listaPrioridadHeap3->push(0);

// pop
listaPrioridadHeap3->pop();

// top
cout << "top: " << listaPrioridadHeap3->top() << endl;

// empty
cout << "empty: " << listaPrioridadHeap3->empty() << endl;

// size
cout << "size: " << listaPrioridadHeap3->Size() << endl;

listaPrioridadHeap3->print();

// heap 4
priorityQueue *listaPrioridadHeap4 = new priorityQueue();
cout << "\nHEAP 4:" << endl;

// empty
cout << "empty: " << listaPrioridadHeap4->empty() << endl;

// push
listaPrioridadHeap4->push(111);
listaPrioridadHeap4->push(222);
listaPrioridadHeap4->push(56);
listaPrioridadHeap4->push(78);
listaPrioridadHeap4->push(0);

// pop
listaPrioridadHeap4->pop();

// top
```

```
    cout << "top: " << listaPrioridadHeap4->top() << endl;

    // empty
    cout << "empty: " << listaPrioridadHeap4->empty() << endl;

    // size
    cout << "size: " << listaPrioridadHeap4->Size() << endl;

    listaPrioridadHeap4->print();

}
```

Bibliografía:

GeeksforGeeks. (2023, 17 abril). *Binary heap*. https://www.geeksforgeeks.org/binary-heap/