Act 3.2 - Árbol Heap: Implementando una fila priorizada

Daniel Esparza Arizpe - A01637076

Diego Alberto Cisneros Fajardo - A01643454

Luis Fernando Li Chen - A00836174

Sadrac Aramburo Enciso - A01643639

Diego Michell Villa Durán - A00836723

Programación de estructuras de datos y algoritmos fundamentales

Grupo 605

Tecnológico de Monterrey

Viernes 3 de noviembre 2023

Act 3.2 - Árbol Heap: Implementando una fila priorizada

```
//DC, DE, LL, SA, DV
#include <iostream>
#include <vector>

using namespace std;

class priority_queue {
private:
int size;
public:
priority_queue(){//inicializamos el valor en 0
size = 0;
};
vector<int> heap;

//cuando insertamos un nuevo elemento
void heapify_i(int index) {
if (index == 0) {
return;
}
int parent_index = (index-1)/2;

if (heap[parent_index] < heap[index]) { //cambiamos si es mas grande
que el padre
swap(heap[parent_index], heap[index]);
heapify_i(parent_index);
}
}
//cuando sacamos un elemento
void heapify_del(int index) {
int leftChild = 2*index+1;
int rightChild = 2*index+2;
int largestChild = index;

if (leftChild < heap.size() && heap[leftChild] > heap[largestChild]) {
//compara el elemento actual con sus hijos para encontrar el hijo más
grande
```

```
largestChild = leftChild;
}

if (rightChild < heap.size() && heap[rightChild] > heap[largestChild])
{
largestChild = rightChild;
}

if (largestChild != index) {
swap(heap[index], heap[largestChild]);
heapify_del(largestChild);
}
}

void push(int value) {
heap.push_back(value);
heapify_i(heap.size() -1);
}

int top() {
if (heap.empty()) {
return -1;
}
return heap[0];
}

int sizePq() {
return heap.size();
}

int pop() {
if (heap.empty()) {
return -1;
}

int root = heap[0];
heap[0] = heap.back();
heap.pop_back();
heapify_del(0);
```

```
return root;
}

bool empty() {
return heap.empty();
}
};

int main() {
priority_queue heap;
//insertamos los valores
heap.push(55);
heap.push(9);
heap.push(21);
heap.push(67);
heap.push(43);
heap.push(1);

//iteramos para mostrarlo
for(int i=0; i<heap.heap.size(); i++){
cout << heap.heap[i]<<",";
}
heap.pop();
cout <<endl;
//checamos si despues del pop ya lo borro
//checamos si el metodo top funciona
//iteramos para mostrarlo
for(int i=0; i<heap.heap.size(); i++){
cout << heap.heap[i]<<",";
}
//probamos los metodos
cout <<endl;
cout<< heap.sizePq();
cout <<endl;
cout<< heap.empty();
cout <<endl;
cout<< heap.top();
```

```
return 0;
}
```

Complejidad

Inserción (push):
La inserción en un montículo binario tiene una complejidad de O(log N), donde N es el número de elementos en el montículo. En este caso, después de agregar un elemento, se llama a heapify_i, que tiene una complejidad de O(log N) en el peor caso.

Extracción (pop):
La extracción de un elemento del montículo también tiene una complejidad de O(log N) en el peor caso. Después de eliminar el elemento, se llama a heapify_del, que también tiene una complejidad de O(log N) en el peor caso.

Acceso al elemento máximo (top):
Obtener el elemento máximo del montículo (sin eliminarlo) tiene una complejidad de O(1), ya que el elemento máximo siempre está en la raíz del montículo.

Tamaño del queue (sizePq):
Obtener el tamaño del montículo (número de elementos) tiene una complejidad de O(1), ya que el tamaño se almacena en una variable y se actualiza con cada inserción y eliminación.

Verificar si el montículo está vacío (empty):
Verificar si el montículo está vacío tiene una complejidad de O(1), ya que simplemente verifica si el tamaño del montículo es 0.

Casos de prueba

```
83
84  int main() {
85      priority_queue heap;
86      //insertamos los valores
87      heap.push(55);
88      heap.push(9);
89      heap.push(21);
90      heap.push(67);
91      heap.push(43);
92      heap.push(1);
93
94      //iteramos para mostrarlo
95      for(int i=0; i<heap.heap.size(); i++){
96          cout << heap.heap[i]<<",";
97      }
98      heap.pop();
99      cout <<endl;
100     //checamos si despues del pop ya lo borro
101     //checamos si el metodo top funciona
102     //iteramos para mostrarlo
103     for(int i=0; i<heap.heap.size(); i++){
104         cout << heap.heap[i]<<",";
105     }
106     //probamos los metodos
107     cout <<endl;
108     cout<< heap.sizePq();
109     cout <<endl;
110     cout<< heap.empty();
111     cout <<endl;
112     cout<< heap.top();
113
114     return 0;
115 }
116
117
```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS  POLYGLOT NOTEBOOK

PS D:\TEC\algortimos> ./PAm
67,55,21,9,43,1,
55,43,21,9,1,
5
0
55
```

```cpp

int main() {
    priority_queue heap;
    //insertamos los valores
    heap.push(3);
    heap.push(43);
    heap.push(1);

    //iteramos para mostrarlo
    for(int i=0; i<heap.heap.size(); i++){
        cout << heap.heap[i]<<",";
    }
    heap.pop();
    cout <<endl;
    //checamos si despues del pop ya lo borro
    //checamos si el metodo top funciona
    //iteramos para mostrarlo
    for(int i=0; i<heap.heap.size(); i++){
        cout << heap.heap[i]<<",";
    }
    //probamos los metodos
    cout <<endl;
    cout<< heap.sizePq();
    cout <<endl;
    cout<< heap.empty();
    cout <<endl;
    cout<< heap.top();

    return 0;
}
```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS  POLYGLOT NOTEBOOK

PS D:\TEC\algortimos> g++ ./heap.cpp -o PAm
PS D:\TEC\algortimos> ./PAm
43,3,1,
3,1,
2
0
3
```

```
    }

    bool empty() {
        return heap.empty();
    }
};

int main() {
    priority_queue heap;
    //insertamos los valores
    heap.push(3);
    heap.push(43);
    heap.push(1);
    heap.push(2);
    heap.push(5);

    //iteramos para mostrarlo
    for(int i=0; i<heap.heap.size(); i++){
        cout << heap.heap[i]<<",";
    }
    heap.pop();
    cout <<endl;
    //checamos si despues del pop ya lo borro
    //checamos si el metodo top funciona
    //iteramos para mostrarlo
    for(int i=0; i<heap.heap.size(); i++){
        cout << heap.heap[i]<<",";
    }
    //probamos los metodos
    cout <<endl;
    cout<< heap.sizePq();
    cout <<endl;
    cout<< heap.empty();
    cout <<endl;
    cout<< heap.top();

    return 0;
}
```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS  POLYGLOT NOTEBOOK

PS D:\TEC\algortimos> g++ ./heap.cpp -o PAm
PS D:\TEC\algortimos> ./PAm
43,5,1,2,3,
5,3,1,2,
4
0
5
PS D:\TEC\algortimos>
```

valores de size, empty y top cuando no insertamos ningún valor.