Act 3.2 - Árbol Heap: Implementando una fila priorizada

Programación de estructuras de datos y algoritmos fundamentales (Gpo 4)

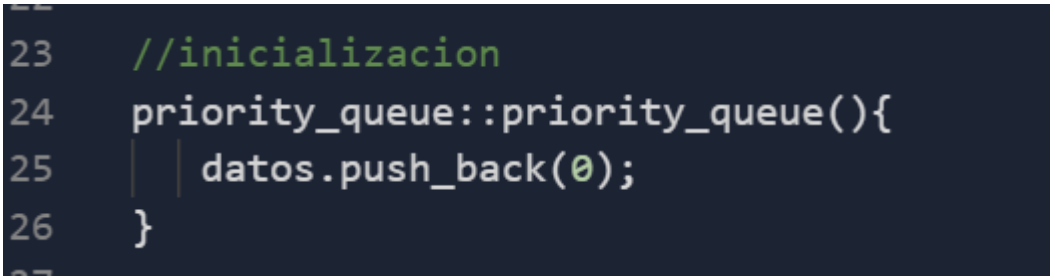
Alumnos:

Thomas Freund Paternostro //A00831997

Fecha de entrega:

22/10/2021

Repelit: https://replit.com/join/xflbusvduk-thomasfreund1



Línea	Costo	Repeticiones (peor caso)
24	C1	1

$$T(n) = C1 = C$$
$$T(n) = C$$

$$Complejidad: O\ (1))$$

```
28   //Agregue un dato a la fila priorizada (insert)
29   void priority_queue::push(int dato){
30     //insertar el dato en el queue
31     datos.push_back(dato);
32     //establecer un index
33     int index = datos.size()-1;
34     //seguier mientras no sea 1
35     while(index != 1)
36     {
37       //si el elemento es mayor que dato con ese index, swap
38     if(datos[index] > datos[index >> 1])
39     {
40     swap(datos[index], datos[index >> 1]);
41     index >>= 1;
42     }
43     else break;
44   }}
```

Línea	Costo	Repeticiones (peor caso)
31	C1	1
33	C2	1
34	C3	1
38	C4	n
40	C5	nlogn
41	C6	n
43	C7	n

$$T(n) = C1 + C2 + C3(n) + C4(n) + C5(nlogn) + C6(n) + C7(n)$$
$$T(n) = (C3 + C4 + C6 + C7)n + (C1 + C2) + nlognC5$$
$$a = C3 + C4 + C5 + C6 + C7,\ b = C1 + C2$$
$$T(n) = an + b + nlogn$$

Dado que se evalúa el peor caso y se lleva al límite donde $\lim_{n \to \infty}$, b se vuelve insignificante.

$$T(n) = an + nlogn$$
$$Complejidad: O\ (an + nlogn) = O\ (nlogn)$$

```
47  //Regresa el valor del dato que esta con mayor prioridad en la
    fila priorizada.
48  int priority_queue::top(){
49    return datos[1];
50  }
```

Línea	Costo	Repeticiones (peor caso)
48	C1	1

$$T(n) = C1 = C$$
$$T(n) = C$$

$$Complejidad: O\ (1))$$

```
52   //Regresa un valor boleando diciendo si la fila priorizada esta
     vacía o tiene datos.
53   bool priority_queue::empty(){
54     //si el size regresa un valor que no es cero, no esta vacio
55     if (datos.size() == 0){
56       return true;
57     }else{
58     return false;
59     }
60   }
```

Linea	Comp	Caso
55	1	C1
56	1	C2
58	1	C3

$$T(n) = C1 + C2 + C3 = C$$
$$T(n) = C$$

$$Complejidad: O\ (1))$$

```
62    //Regresa la cantidad de datos que tiene la fila priorizada
63    int priority_queue::size(){
64      return datos.size()-1;
65    }
```

Línea	Costo	Repeticiones (peor caso)
48	C1	1

$$T(n) = C1 = C$$
$$T(n) = C$$
$$Complejidad: O\ (1))$$

```
75  //Saca de la fila priorizada el dato que tiene mayor prioridad
76  void priority_queue::pop(){
77
78    if(datos.size() == 1){
79      return;
80    }
81    swap(datos[1], datos[datos.size()-1]);
82    datos.pop_back();
83    maxHeapify(1);//llama al actual top y determina la estructura
      para tener maxHeap
```

Linea Comp Caso

78	1	C1
79	1	C2
81	1	C3
82	1	C4
83	1	C5

$$T(n) = C1 + C2 + C3 + C4 + C5 = C$$

Dado que se evalúa el peor caso y se lleva al límite donde $\lim_{n \to \infty}$, b se vuelve insignificante.

$$T(n) = C$$

$$Complejidad: O\,(an) = O\,(1)$$

Nota: Pop debe ser O(nlog) pero como esta llama a maxHeapify el ordenamiento para tener un Max Heap que en el peor caso es O(nlogn) donde cada swap es O(1).

```
86   //explicado en clase
87   void priority_queue::maxHeapify(int index){
88     while(index < datos.size())
89           {
90       int largest = index;
91       if((index << 1) < datos.size() && datos[index << 1] > datos
         [largest]){
92       largest = index << 1;
93       }
94       if((index << 1) + 1 < datos.size() && datos[(index << 1) + 1]
          > datos[largest]){
95       largest = (index << 1) + 1;
96       }
97       if(largest != index)
98       {
99       swap(datos[index], datos[largest]);
100      index = largest;
101      }
102      else break;
103    }
104  }
```

Línea	Costo	Repeticiones (peor caso)
87	C1	n
88	C2	n
90	C3	n
91	C4	n
92	C5	n
93	C6	n
94	C7	n
95	C8	n
97	C9	n
99	C10	nlogn
100	C11	n
102	C12	n

$$T(n) = C1(n) + C2(n) + C3(n) + C4(n) + C5(n) + C6(n) + C7(n) + C8(n) + C9(n) + C10(nlogn) + C$$

$$T(n) = (C1 + C2 + C3 + C4 + C5 + C7 + C8 + C9 + C11 + C12)n + C10(nlogn)$$

$$a = C4 + C5 + C7 + C8 + C9 + C10 + C11,\ b = C10$$

$$T(n) = an + b(nlogn)$$

Dado que se evalúa el peor caso y se lleva al límite donde $\lim_{n \to \infty}$,la an se vuelve insignificante.

$$T(n) = b(nlogn)$$

Esta complejidad se le pudiera atribuir al pop si se lo escribiera en la otra función.

Reflexión:
En esta actividad se trabajó con una lista de prioridad heap. Dónde se pudo entender un método de ordenamiento y también una estructura de datos que permite establecer una lista " tipo árbol" dónde está ordenado por niveles dónde mayor nivel en este caso es el número más significativo o priorizado en la lista. Esta actividad me permitió poder entender cómo usar un heap y entender su estructura y su funcionamiento. Donde cuando uso un push e insertar un número estos se ordenan acorde a mean o maxheap y cuando realizo un pop que lleva en este caso a una función maxHeapify este realiza los swaps correspondientes para mantener la integridad correspondiente.