

public void delete(K key) {

int deleteKey = hashFunction(key); ***// O(1) (complejidad temporal y espacial)***

HNode<K,V> deleteNode = table[deleteKey]; ***// O(1) (complejidad temporal y espacial)***

while (deleteNode != null){ ***// Esto iterará a través de la lista enlazada, O(n) en el peor caso***

if(deleteNode.getKey().equals(key)){ ***// O(1) (complejidad temporal y espacial)***

HNode<K,V> prev = deleteNode.getPreviousNode(); ***// O(1) (complejidad temporal y espacial)***

HNode<K,V> next = deleteNode.getNextNode(); ***// O(1) (complejidad temporal y espacial)***

prev.setNextNode(next); ***// O(1) (complejidad temporal y espacial)***

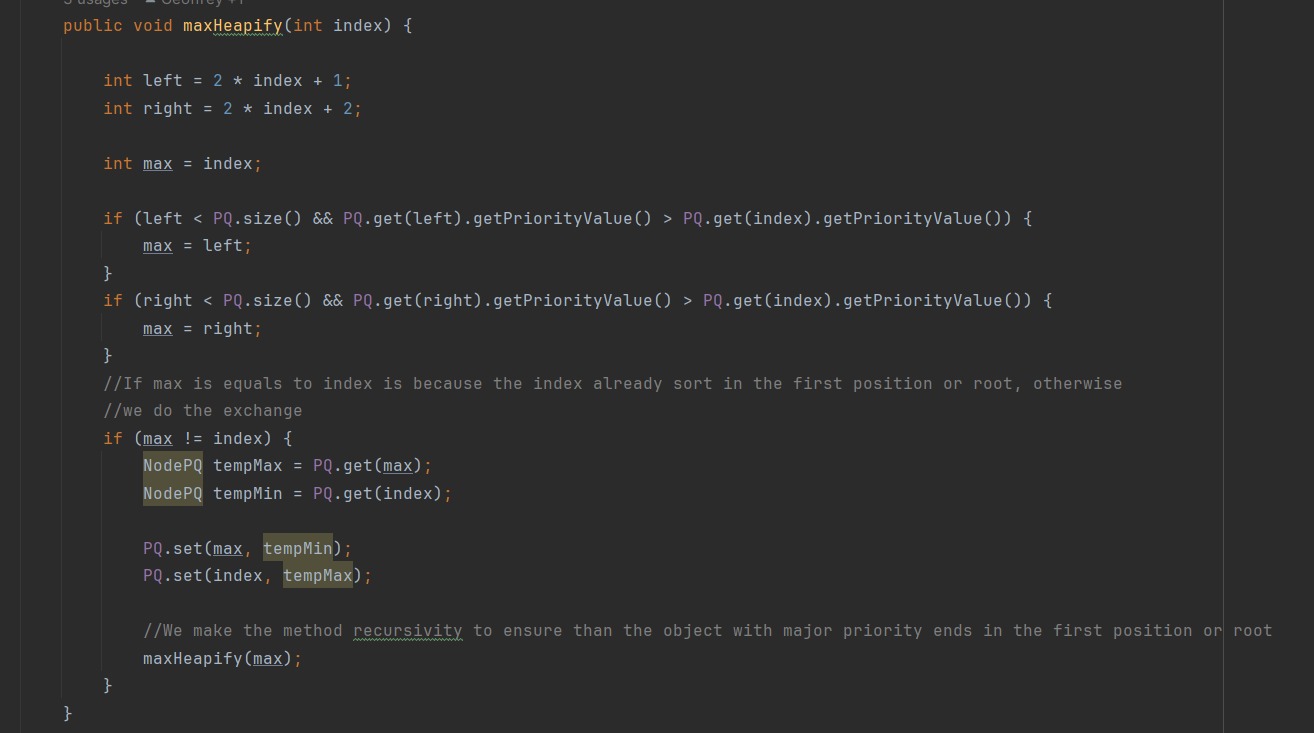
next.setPreviousNode(prev); ***// O(1) (complejidad temporal y espacial)***

}

deleteNode = deleteNode.getNextNode(); ***// O(1) (complejidad temporal y espacial)***

}

}  
  
  
El peor de los casos para complejidad temporal es O(n), debido a que usamos un ciclo while, el cual recorre toda la lista enlazada de tamaño (n). En cuanto a la espacial es constantemente O(1).



public void maxHeapify(int index) {

int left = 2 \* index + 1; ***// O(1) (complejidad temporal y espacial)***

int right = 2 \* index + 2; ***// O(1) (complejidad temporal y espacial)***

int max = index; ***// O(1) (complejidad temporal y espacial)***

if (left < PQ.size() && PQ.get(left).getPriorityValue() > PQ.get(index).getPriorityValue()) {

max = left; ***// O(1) (complejidad temporal y espacial)***

}

if (right < PQ.size() && PQ.get(right).getPriorityValue() > PQ.get(index).getPriorityValue()) {

max = right; ***// O(1) (complejidad temporal y espacial)***

}

if (max != index) {

NodePQ tempMax = PQ.get(max); ***// O(1) (complejidad temporal y espacial)***

NodePQ tempMin = PQ.get(index); ***// O(1) (complejidad temporal y espacial)***

PQ.set(max, tempMin); ***// O(1) (complejidad temporal y espacial)***

PQ.set(index, tempMax); ***// O(1) (complejidad temporal y espacial)***

maxHeapify(max); ***// Llamado recursivo; busca por lados del árbol y realiza reordenamiento a partir de la identificación del lado, es decir O(Log n)***

}

}

El peor de los casos para la complejidad temporal sería O(Log n) debido a que usa un llamado recursivo para realizar reordenamiento de nodos dentro del árbol, y no recorre todo el árbol, si no que ordena un lado solamente. En cuanto a la complejidad espacial, es constante O(1).