Priority Queue

Frank Roger Salas Ticona

12 de diciembre de 2022

Definición de Priority Queue

Frank Roger
Salas Ticona

Es una estructura de datos similar a las colas y pilas, donde los elementos llevan cierta prioridad asociada, el elemento con la mayor prioridad es el que es utilizado primero.

Priority Queue		Queue
57		32
32		12
19		57
12		19

¿Por qué una priority queue?

Las colas de prioridad son usadas ampliamente en diversos algoritmos tales como el algoritmo de Dijkstra, Huffman, best-first search, entre otros. Lo cual la hace una estructura de datos idónea para resolver distintos tipos de problemas.

¿Cómo es una priority queue?

En este caso para nuestra priority queue usaremos una estructura de datos conocida como SkipList.

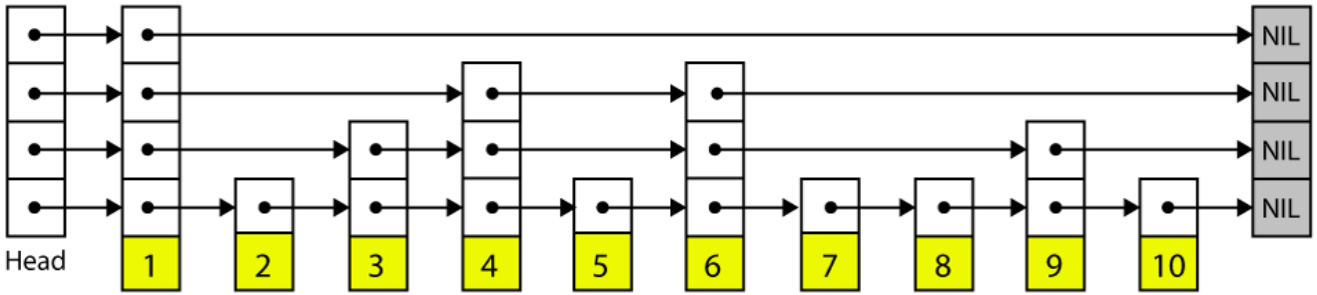

Figura: Skiplist.

¿Cómo es una priority queue?

¿Cómo obtenemos el dato con mayor prioridad? Simplemente es el primero de nuestra lista.

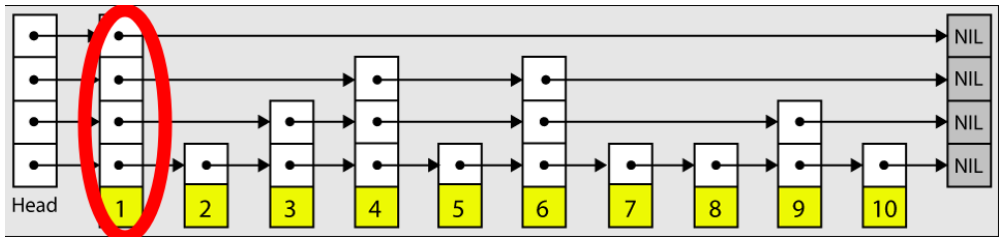

Figura: Top de skiplist.

¿Cómo es una priority queue?

La eliminación es muy sencilla pues tengamos en cuanta que nuestra priority queue solo elimina el primer elemento. Es decir lo único que se debe hacer es que el *header* de la skiplist apunte al siguiente elemento de esta manera, el *pop* se hace con complejidad de $O(1)$.

¿Cómo es una priority queue?

Frank Roger
Salas Ticona

La inserción tiene como complejidad esperada de $O(logn)$ debido a que la construcción de un skiplist es aleatoria al momento de escoger los niveles. Lo primero que se debe de hacer es buscar el lugar donde debe de ser insertado, lo cual es $O(logn)$ y solo queda reajustar los punteros como en una lista enlazada, cuya complejidad es $O(1)$.

Paralelismo

Como es explicado en el paper de Håkan Sundell y Philippas Tsigas, usaremos operaciones atómicas las cuales permitan hacer modificaciones en paralelo a nuestra skiplist, estas operaciones atómicas se hacen en las operaciones principales, es decir en el *push*, *pop* y *top*.

```
function Insert(key:integer, value:pointer to word):boolean
I1   ⟨TraverseTimeStamps();⟩
I2   level:=randomLevel();
I3   newNode:=CreateNode(level,key,value);
I4   COPY_NODE(newNode);
I5   node1:=COPY_NODE(head);
I6   for i:=maxLevel-1 to 1 step -1 do
I7       node2:=ScanKey(&node1,i,key);
I8       RELEASE_NODE(node2);
I9       if i<level then savedNodes[i]:=COPY_NODE(node1);
I10  while true do
I11      node2:=ScanKey(&node1,0,key);
I12      value2:=node2.value;
I13      if not IS_MARKED(value2) and node2.key=key then
I14          if CAS(&node2.value,value2,value) then
I15              RELEASE_NODE(node1);
I16              RELEASE_NODE(node2);
I17              for i:=1 to level-1 do
I18                  RELEASE_NODE(savedNodes[i]);
I19              RELEASE_NODE(newNode);
I20              RELEASE_NODE(newNode);
I21              return true_2;
I22          else
```

Figura: Pseudo código de inserción.

Paralelismo

Implementación en C++.

```
template<typename T>
void
PriorityQueue<T>::insertElement(T key)
{
...
    for (int i = level; i >= 0; i--) {
        while (current->forward[i] != NULL &&
                current->forward[i]->key < key)
            current = current->forward[i];
        update[i] = current;
    }
...
    for (int i = 0; i <= rlevel; i++) {
        n->forward[i] = update[i]->forward[i];
        update[i]->forward[i] = n;
```

Paralelismo

La eliminación en la skiplist para nuestra priority queue, tiene que también ser modificada para un correcto funcionamiento en el paralelismo.

```
function DeleteMin():pointer to Node
D1    ⟨TraverseTimeStamps();⟩
D2    ⟨time:=getNextTimeStamp();⟩
D3    prev:=COPY_NODE(head);
D4    while true do
D5        node1:=ReadNext(&prev,0);
D6        if node1=tail then
D7            RELEASE_NODE(prev);
D8            RELEASE_NODE(node1);
D9            return NULL;
      retry:
D10       value:=node1.value;
D11       if not IS_MARKED(value) ⟨and
            compareTimeStamp(time,node1.timeInsert)>0⟩ then
D12           if CAS(&node1.value,value,
               GET_MARKED(value)) then
D13               node1.prev:=prev;
D14               break;
D15           else goto retry;
D16       else if IS_MARKED(value) then
D17           node1:=HelpDelete(node1,0);
D18       RELEASE_NODE(prev);
D19       prev:=node1;
D20   for i:=0 to node1.level-1 do
```

Figura: Pseudo código de eliminación.

Paralelismo

Implementación en C++.

```
template<typename T>
void
PriorityQueue<T>::deleteElement(T key)
{
...
    for(int i = level; i >= 0; i--)
    {
        while(current->forward[i] != NULL  &&
              current->forward[i]->key < key)
            current = current->forward[i];
        update[i] = current;
    }
...
    update[i]->forward[i] = current->forward[i];
...
}
```

Comparación entre implementaciones.

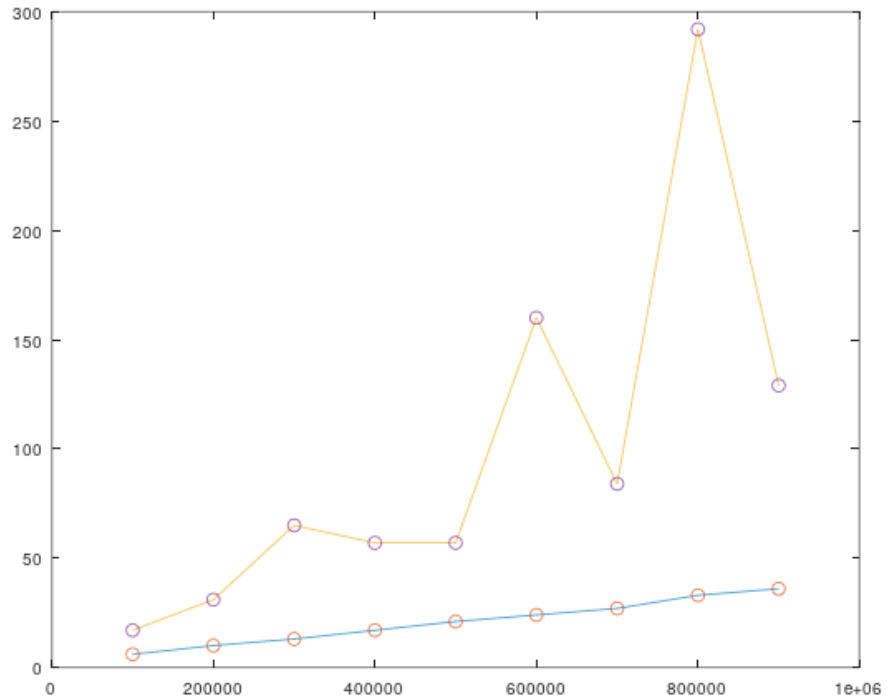

Figura: Comparación con distintas cantidades de datos entre priority queue sin y con paralelismo.

Bibliográfia

http://www.non-blocking.com/download/SunT03_PQueue_TR.pdf
https://en.cppreference.com/w/cpp/container/priority_queue