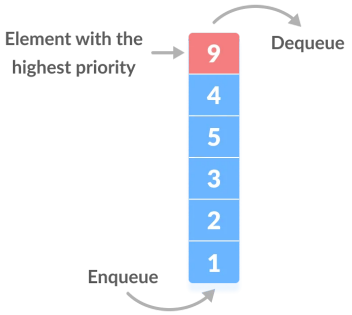
Carlos César Rodríguez García | A00344666
TC1031 Grupo 13

Análisis de Estructuras de Datos Utilizadas en la Actividad

Priority Queue

Esta es la principal estructura utilizada para la solución de este reto. Es bastante similar a un *stack* o a un *queue*, sin embargo, su mayor diferencia radica en que los elementos se insertan en una posición determinada dependiendo de un valor de prioridad. Esto es similar a la fila en un hospital, donde si llega un paciente con una urgencia (alta prioridad) será colocado al principio de la fila y atendido en primer lugar.



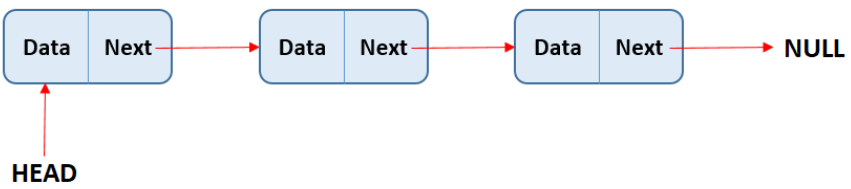
Las Priority Queues varios usos reales como en búsqueda A* en Inteligencia Artificial o aplicaciones en estadística. Incluso de manera reciente hice uso de *Priority Queues* en mi trabajo de medio tiempo, pues son importantes para la compresión de datos con el Algoritmo de Huffman; el cual en realidad está detrás de muchos algoritmos de compresión más complejos y que usamos de manera cotidiana.

Ahora bien, una *Priority Queue* puede implementarse teniendo distintas estructuras de datos como base. En esta ocasión, nosotros la implementamos usando un *Linked List.*

Linked List

Las *linked list* son estructuras de datos lineales compuestas por nodos donde cada uno de estos tiene una referencia a su posterior. De esta forma, las listas ligadas simples se recorren en una única dirección y tienen un comportamiento dinámico, pues se puede seguir agregando nodos sin ninguna restricción.

De esta última cualidad nace una de las razones por las cuales son tan importantes, pues son de ayuda cuando no sabemos cuál será el tamaño de cierto input. Así, también se pueden leer archivos u otros recursos donde no se sabe de antemano su tamaño.



Funciones

Ahora, para las *Priority Queues* implementadas con *Linked Lists*, se pueden emplear las siguientes funciones:

- **pop():** que elimina y retorna el nodo de enfrente (o tope) del queue. *Complejidad:* O(1)
- **push()/enqueue():** que agrega un nuevo nodo, recorriendo la lista y encontrando su lugar. *Complejidad:* O(n)
- **top():** que únicamente retorna el nodo de enfrente. *Complejidad:* O(1)

¿Por qué no usamos una Doubly Linked List?

Las listas doblemente ligadas tienen las mismas ventajas que una *linked list* simple, sin embargo, cada nodo tiene también una referencia a su anterior. Con esto, se puede recorrer la *Priority Queue* en dos direcciones. Sin embargo, no lo consideramos necesario para esta ocasión, puesto que la *queue* solo se llena una vez y esta acción puede ser realizada únicamente con el *push* básico de una lista ligada.

No creímos que fuera ineficaz; simplemente, innecesario.

Bibliografía

- Elgabry, O. (Enero de 2017). Priority Queues. *Medium*. Recuperado de https://medium.com/omarelgabrys-blog/priority-queues-11e469cda253
- Sharma, R. (Mayo de 2021). Priority Queue in Data Structure. *upGrad.* Recuperado de http://pages.cs.wisc.edu/~vernon/cs367/notes/9.BST.html