Documento Taller 5

1. MaxColaPrioridad: La manera más eficiente de implementar esta estructura es mediante el uso de Heaps, ya que así se tiene una complejidad de O(log N) para insertar y eliminar elementos en la cola de prioridad. También es una buena implementación cuando se hace con árboles de izquierda, así (y con heap) se tiene la siguiente complejidad para los métodos:

isEmpty(): En tiempo O(1).

max(): En tiempo O(1).

darNumeroElementos(): En tiempo O(1) (size()).

darElementoPos(int n): En tiempo O(1) (get()).

agregar(T elemento): En tiempo O(log N) en el mejor caso y en caso promedio(N es el tamaño de la lista de prioridad, mismo método put()).

agregar(T elemento): En tiempo O(N log N) en el peor caso.

delMax(): En tiempo O(log N) en el mejor caso y caso promedio (N es el tamaño de la lista de prioridad, mismo método remove()).

delMax(): En tiempo O(N log N) en el peor caso.

1. MaxHeapCP: Como es una estructura que debe estar siempre ordenada de mayor a menor, su complejidad al insertar elementos es de O(log N) debido al número de comparaciones y de intercambios que se deben realizar.

Al construirla su complejidad es de O(N), es decir, el tiempo de construcción del Heap es directamente proporcional con el número de datos que se van a insertar en él.

Se tiene la misma complejidad en los mismos casos que en el numeral anterior.

Conclusión: Es mejor utilizar la cola de prioridad basándose en la estructura de Heap, ya que la complejidad computacional es mucho más baja que cuando se hace con base en una Queue, ya que con ésta la complejidad de inserción se convierte en O(N) y la de ordenamiento para mantener la cola de prioridad cambia a O(N log N).

Tabla de tiempos de ejecución según los datos de entrada.

|  |  |
| --- | --- |
| MUESTRA | TIEMPO PROMEDIO  DE EJECUCIÓN |
| Enero: 86685 datos | 4,9379 |
| Febrero: 74985 datos | 4,8749 |
| Marzo: 104218 datos | 5,0179 |
| Abril: 107655 datos | 5,0320 |

Gráfica comparativa: