Name: Yasmine Hosny Hassan Mahmoud

ID: 20203616445

Report about Priority Queue

Description:

a priority queue is an abstract data-type similar to stack data structure in which each element additionally has a "priority" associated with it. In a priority queue, an element with
high priority is served before an element with low priority. In some implementations, if two elements have the same priority, they are served according to the order in which they were enqueued;
in other implementations ordering of elements with the same priority remains undefined.
A priority queue is a concept like "a list" or "a map"; just as a list can be implemented with a linked list or with an array, a priority queue can be implemented with a heap or with
a variety of other methods such as an unordered array.

Example:

1. we are adding books to queue and printing all the books. The elements in PriorityQueue must be of Comparable type. String and Wrapper classes are Comparable by default.
To add user-defined objects in PriorityQueue, you need to implement Comparable interface.
The following are some of the most popular applications of the priority queue:
2. Dijkstra’s Shortest Path algorithm: Priority queue can be used in Dijkstra’s Shortest Path algorithm when the graph is stored in the form of the adjacency list.
3. Prim’s Algorithm: Prim’s algorithm uses the priority queue to the values or keys of nodes and draws out the minimum of these values at every step.
4. Data Compression: Huffman codes use the priority queue to compress the data.
5. Operating Systems: The priority queue is highly useful for operating systems in various processes such as load balancing and interruption handling.

Implementation:

```
class PriorityQueue
   {
        private int [] priqueue;
        private int  maxSize;
        private int  nItems;

        public PriorityQueue(int maxsize) {
            this.maxSize = maxSize;
            priqueue = new int[maxSize];
            nItems = 0;
        }

        public void insert(int item) {
            if(nItems == 0)
                priqueue[nItems++] = item;
            else
            {
                int k;
                for(k = nItems-1 ; k>=0 ; k--)
                {
                    if(item > priqueue[k])
                        priqueue[k+1] = priqueue[k];
                    else
                        break;
                }
                priqueue[k+1] = item;
                nItems++;
            }
        }

        public int remove() {
            return priqueue[--nItems];
        }

        public boolean isEmpty() {
            return (nItems == 0);
        }
   }
```