Assignment 12

21118

Problem Statement:

Write a program to create a priority queue in C++ using inorder-list to store the items in the queue. Create a class that includes the data items (which should be template) & the priority (which should be int). The inorder list should contain these objects, with <= operator overloaded so that the items with highest priority appear at the start of the list (which will make it relatively easy to retrive the highest item.)

Objectives

1) Understand the priority queue data structure & it's implementation details.
2) ~~Implement~~ understand operator overloading concept to compare two user defined data structures.

Outcomes

1) Implement priority queue data structure using arrays.
2) Implement operator overloading to compare user defined data types.

Hardware requirement:

Manufacturer & Model: Acer Swift-3
Processor: Intel core-i5 8th gen (8265U @ 1.6GHz)
Installed memory: 8GB RAM, 512 GB SSD
Architecture: 64-bit

Software requirement:

Operating system: Ubuntu 20.04 LTS on oracle virtual machine (3 processors & 4096MB base memory is allocated)
C++ version: C++ 14
Compiler for C++: g++ (version: 10.1.0)
Code editor: sublime text (Build 2011)

Theory

A priority queue is an abstract data type similar to a regular queue / stack data structure in which each element additional has a "priority" associated with it.

In a priority queue, an element with high priority is served before an element with low priority.

There are three operations defined on priority queue.

insert: insert element in priority queue.

remove: remove the element with hightest priority

~~get~~ front: get the element with the highest priority which is present in the queue.

Pseudo Code:

```
Operations of priority queue: ~~arr~~
  Algorithm front(){ // returns element with highest priority
     return arr[0];
  }

  Algorithm insert(T x){ // insert element in the queue
     if (list is full) return;
     else {
        1. add x to end of arr/list.
        2. shift ~~it~~ to it's appropriate position based on
           it's priority.
        3. increase the size of list.
     }
  }

  Algorithm remove(){ // removes highest priority
                                element in queue.
if (list is empty) return.
else {
    //first element of list is of highest priority
    ~~1. shift all elements to their previous position except~~ 1st.
    ~~2. decrease the size of list.~~
    1. increase the value of front by 1 in circular list.
    }
}
```

ADT of classes:

```
1) class Job {    // simulates job in operating system.
    private:
      string data;
      int priority;
    public:
      Job (string data = "", int priority = 0 {    // constructor
         //code
      }
      void setData() {  // sets value of data & priority
         //code
      }
      string getData() {   // return data of job
         // code
      }
      int getPriority() {   // return priority of current job
         //code
      }
      friend bool operator <= (Job, Job); // overloaded <=
                                 operator to compare to jobs.
}

2) class PriorityQueue {   // template class of priority queue
```

```
private:
  T arr[n];        //array to store elements of type T.
  int n;
public:
  PriorityQueue() {     //constructor function.
    //code
  }
  int getSize() {       //returns size of / total no. of
    //code                elements in queue
  }
  bool isEmpty() {      //checks if queue is empty or not
      //code
  }
  bool isFull() {       //checks if capacity of queue
      //code              allows any ~~other~~ new item
  }
  T front() {           //return element of highest
      //code              priority present in the queue
  }
  void insert(T x) {    //insert x in the queue.
      //code
  }
  void remove() {       //removes ~~to~~ element of highest
      //code              priority from queue.
  }
  void print() {        //prints all elements present in
      //code              queue.
  }
}
```

* Analysis of algorithms:

operations of priority queue:

① inserting new element	in worst case O(n) time is required. n is total elements in queue.	constant space is required.
② removing element	time complexity of algorithm is O(1) ~~where n is total elements in queue.~~	Constant space is required.
③ getting highest priority element	The algorithm runs in constant time i.e. O(1)	No extra space is required.

Applications

The priorities queues are very helpful data structure & they have diverse applications.

1. Bandwidth management
2. Discrete event simulation.
3. Dijkstras Algorithm - single source shortest path.
4. Huffman coding
5. Prim's algorithm.

Conclusions

Priority queue is very useful data structure & also auxiliary data structure in many of algorithms of computer science. The assignment gives complete / hight level idea of working of priority queue data structure.