Data Structures

Heap Homework 2

Mostafa S. Ibrahim
Teaching, Training and Coaching since more than a decade!

Artificial Intelligence & Computer Vision Researcher
PhD from Simon Fraser University - Canada
Bachelor / Msc from Cairo University - Egypt
Ex-(Software Engineer / ICPC World Finalist)

Problem #: Kth Smallest number (stream)

```
110 public:
111     KthNumberProcessor(int k): k(k){
112     }
113
114     int next(int new_num) {...
124 };
125
126 int main() {
127     KthNumberProcessor processor(4);
128
129     int num;
130     while (cin >> num)
131         cout << processor.next(num) << "\n";
132
```

- This class receives a **infinite stream** of numbers, each time return the kth smallest number
  - Or largest if elements < k
- In other words, if numbers are sorted, it is arr[k-1]

Problem #1: Kth Smallest number (stream)

- Assume input: 9 8 7 6 5 4 10 8 3 5 15
- Let’s simulate
  - 9 ⇒ 9
  - 8 9 ⇒ 9
  - 7 8 9 ⇒ 9
  - 6 7 8 9 ⇒ 9
  - 5 6 7 **8** 9 ⇒ 8
  - 4 5 6 **7** 8 9 ⇒ 7
  - 4 5 6 **7** 8 9 10 ⇒ 7
  - 4 5 6 **7** 8 8 9 10 ⇒ 7
  - 3 4 5 **6** 7 8 8 9 10 ⇒ 6
  - 3 4 5 **5** 6 7 8 8 9 10 ⇒ 5
  - 3 4 5 **5** 6 7 8 8 9 10 15 ⇒ 5

Problem #2: Priority Queue

- Priority queue is a queue in which each element has a "**priority**" associated with it. Elements with **high priority** are **served first** before low priority.
- Assume, in an OS, we have tasks each with priority [and positive value]
  - Assume we enqueued as following:
  - Enqueue (task_id = 1131, priority = 1)
  - Enqueue (task_id = 3111, priority = 3)
  - Enqueue (task_id = 2211, priority = 2)
  - Enqueue (task_id = 3161, priority = 3)
  - Let's print tasks in order: 3111 3161 2211 1131
- Implement a priority queue based on max-heap code
  - Max Heap by definition always has the element of a max value as top
  - So max heap is a perfect **underlying implementation** to the priority queue **ADT**

Problem #2: Priority Queue

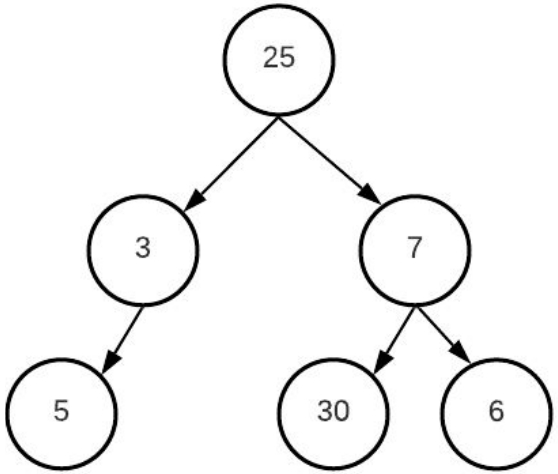
```
PriorityQueue tasks;

tasks.enqueue(1131, 1);
tasks.enqueue(3111, 3);
tasks.enqueue(2211, 2);
tasks.enqueue(3161, 3);
tasks.enqueue(7761, 7);

cout << tasks.dequeue() << "\n";
cout << tasks.dequeue() << "\n";

tasks.enqueue(1535, 1);
tasks.enqueue(2815, 2);
tasks.enqueue(3845, 3);
tasks.enqueue(3145, 3);

while (!tasks.isempty())
    cout << tasks.dequeue() << " ";
```

```
7761
3111
3145 3161 3845 2815 2211 1131 1535
```

- Observe
  - Value 3161 is added before 3145
  - Value 3161 is printed after 3145
  - This is valid: the constraint is tasks with higher priority are printed first
  - If same priority = print in any order
- Study this example to know WHY max-heap couldn't **preserve input order**

Problem #3: Binary Tree Special Traversal

- Going back to the Binary Tree section: Add member function
- void level_order_traversal_sorted()
- It does level order traversal, but at each level values are printed for large to small.
- Use **STL prioirty_queue** to do the task
- Output for this tree
  - 25
  - 7 3
  - 30 6 5



Optional Mini-Project - No solution from me

- Design a data structure that provides **find_min and find_max** in O(1)
  - Other operations: insert(value), delete_min, delete_max
- One mentality: let's **reuse** the common data-structures
  - You will use several ones in the same time
  - E.g. min-heap + max-heap + doubly linked list
  - Interesting: min and max heaps store addresses of nodes
- Another mentality: let's invent a new data-structure
  - Min-max heap is an interesting DS
  - Even level acts like a min heap (e.g. node smaller than descendants)
  - Odd level acts like a max heap
  - Code is like a merge of the 2 heaps
  - Interesting read

"Acquire knowledge and impart it to the people."

"Seek knowledge from the Cradle to the Grave."