Priority Queue implementation

OS lab

UoG

Yiting Jin

Priority queue ADT

- Queue
  - Contains a collection of items that are waiting to be processed.
  - FIFO
- Priority queue
  - Items in the queue are selected by priority, not the time which item is added.s

Opeartion

- Insert(x,p)
  - Adds an item of any BaseType x and its priority p.
- Remove()
  - Removes the items has currently highest priority in the priority queue.
- makeEmpty()
  - Removes all items from the priority queue.
- IsEmpty()
  - Returns a value of type Boolean, which is true if the priority queue contains no items and returns false if there is at least one item.

Implementation

- List: (linked list or array)
  - If we keep the items in the list sorted
    - i.e. items in the list is sorted in the non-descending order according to priority .
      - Next item to be removed is always the last element,
        - » O(1)
      - Inserting a new item into the right place, if we have n items in the list.
        - » O(n)

Implementation

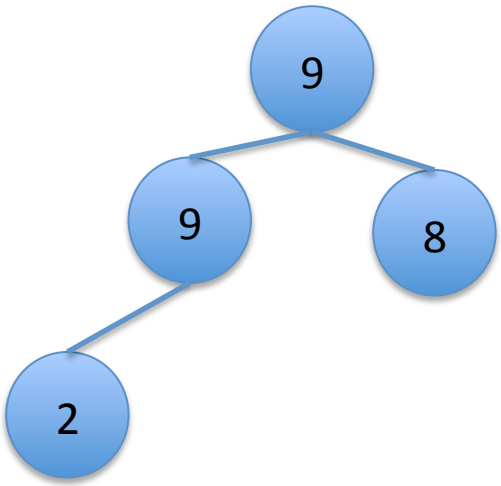
- List: (linked list or array)
  - If items in the list not sorted
    - Finding the next item to remove in a list of n items
      - O(n)
    - Inserting a new item
      - O(1)

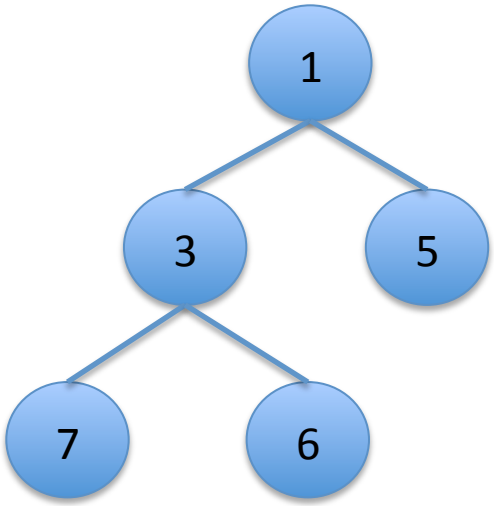
Heap

- Heap(binary)
  - Shape property:
    - A binary heap is a (almost) complete binary tree.
      - All levels of the tree(except possibly the last one), is fully filled and if the last level of the tree is not complete, the nodes of that level are filled from left to right.
  - Heap property:
    - All nodes are either *greater than or equal to (Max heap)* or *less than or equal to (Min heap)* each of its children.

Heap

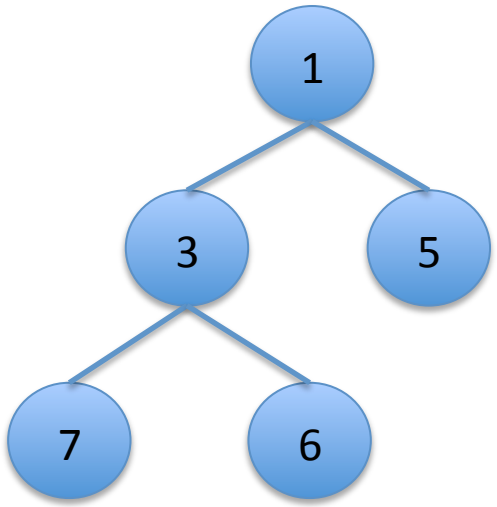
- Example:





Priority queue implementation by heap

- Heap can be physically represented as an array, due to the fact that it is a full binary tree.



0	1	2	3	4
1	3	5	7	6

The children of node at index k in the array are located at 2*k+1 and 2*k+2

Priority queue implementation by heap

- Let's implement priority queue by heap.

```
Struct HeapItem
{
        double priority;
        BaseType item; //assume the basetype is the type of items to be  `
                                //stored
}

HeapItem heap[MAX_SIZE];  //the heap(array of type heapitem)
Int itemCount;  // number of items on the heap
```

Priority queue implementation by heap

- Main functions to be implemented
  - Insert(x,p)
  - Remove()

Insert

```
//Add the new item in the next available position in the array:
heap[itemCount].item = x;
heap[itemCount].priority = p;
itemCount++;
```

Insert

```
//Put the new item into a proper position:
 int position = itemCount -1;   //Current position of item
 while(true)
{       if(position==0){break;} //add at the root, nothing else to do
        int parent = (position-1)/2; //index in array of parent node
        if(heap[position].priority <= heap[parent].priority)
                {break;} //parent has higher priority (current in the proper position)
        else //item added has higher priority than its parent, swap position
        {  HeapItem temp = heap[position];
           heap[position] = heap[parent];
           heap[parent] = temp;
           position  = parent; //Item now in its parent's position
        }
}
```

Remove

```
//Get the item that is being removed
BaseType next = heap[0].item;
//Move the last item from end of the array to root position and decrease the size of the heap by one (maintaining shape property)
 heap[0] = heap[itemCount-1];
 itemCount --;
```

Remove

```
//Perform 'heapify' operation on the root node (maintaining heap property)
 int position = 0; //position of possibly misplaced item.
 while(true)
{ int child1 = 2*position +1 ; int child2 = 2*position+2;
  if(child1>= itemCount){break;}  //child position out of bounds of heap
  int biggerChild; //Index of child with higher priority
  else if(child2>=itemCount){biggerChild = child1;} //node has only left child
  else {biggerChild = child2;}
  if(heap[biggerChild].priority <= heap[position].priority) {break;} //child has
                                    //lower priority ,current node in the right position
  //If not we swap the node with its biggerChild;
  …
}
Return next
```

Time complexity of operation

- Depth of heap
  - Log(n)
- Insertion
  - Log(n)
- Remove
  - Log(n)
- Better than log(n)

Q&A

- Thank you!
- Assignment 1 Remarking