0.1 Priority queues

- *Algorithm:* Maintain a sequence according to the priority of each element.
- *Input:* A sequence that each of its element has a priority value
- *Complexity:* $\mathcal{O}(\log n)$ for maintenance
- *Data structure compatibility:* Heap
- *Common applications:* Optimize Dijkstra's algorithm, optimize Prim's algorithm for minimum spanning tree

Priority queues

A priority queue is a data structure whose all elements are sorted by their priority. There are algorithms that maintains the priority queue when elements are added or removed.

Description

Problem Clarification

Suppose we have an array and an algorithm that compares priority between elements, we wish to maintain the array all time with such principle: whatever the order in which elements are enqueued, the element with high priority is at the front of the one with low priority.

Algorithm Clarification

A priority queue is a data structure for maintaining a set S of elements. Priority is measured by a value of each element named k. For the priority queue discussed here,it is implemented by heap. A heap is a data structure based on complete binary tree and has the property that for any element i other than the root in heap A,

$$A[\lfloor i/2 \rfloor] \leq A[i], \text{ for min-heap}$$

Or

$$A[\lfloor i/2 \rfloor] \geq A[i], \text{ for max-heap}$$

A priority queue should support operations as followed: [1]

1. $insert(S, x)$ which inserts the element x into the set S.
2. $pop(S)$ which removes and returns the element of S with the highest priority.
3. $top(S)$ which returns the element of S with the highest priority.

Complexity

Since a heap of n elements is based on a complete binary tree, we can know that its height is $\Theta(\log n)$. All the operations on the priority queue or on the heap have a running time that is at most the height of the heap.
The time complexity for $insert(S, x)$ is $\mathcal{O}(\log n)$.
The time complexity for $pop(S)$ is $\mathcal{O}(\log n)$. [2] The time complexity for $top(S)$ is $\mathcal{O}(1)$.

Algorithm 1: $insert(S, x)$

Input : The array to add into A, the element to be added x
Output: None

```
A.size++
A[A.size] ← x
i ← A.size
while i>1 and A[⌊i/2⌋].key<A[i].key do
    swap A[i] with A[⌊i/2⌋]
    i ← ⌊i/2⌋
end while
return
```

Algorithm 2: $pop(S)$

Input : The array to add into A,the function that compares priority fn
Output: The element in A with the highest priority

```
Function maintain(A, i):
    l ← 2i
    r ← 2i + 1
    if l ≤ A.size and A[l] > A[i] then
        largest ← l
    end if
    else
        largest ← i
    end if
    if r ≤ A.size and A[r] > A[largest] then
        largest ← r
    end if
    if largest ≠ i then
        swap A[i] with A[largest]
        maintain(A, largest)
    end if
end
max ← A[1]
A[1] ← A[A.size]
A.size–
maintain(A, 1)
return max;
```

Algorithm 3: $top(S)$

Input : The array to A
Output: The element in A with the highest priority

1 **return** A[1]

References.

[1] Leiserson Charles E. Cormen Thomas H. and etc. *Introduction to Algorithms – Third Edition*. The MIT Press, 2009 (cit. on p. 1).

[2] Manuel. *VE477 – Introduction to Algorithms (lecture slides)*. 2019 (cit. on p. 2).