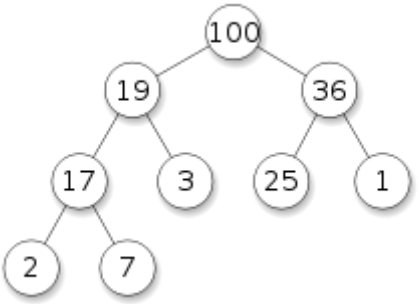
Tutorial #12. Heap

Theoretical part

Heap properties

Heap is a balanced binary tree, and it is **different from binary search tree**! Heap properties are:

- **Heap is a complete binary tree**. That means all levels are full (except, maybe the last), and last level nodes are filled *from left to right*. Property obviously leads to the idea that if you store heap as an **array-based k-ary (2-ary) tree**, data alignment will be compact (no *null*s inside the array). We need from 2^{h-1} up to 2^h-1 element array to store a tree of height **h**. The worst and the best cases for space consumption are different only by multiplier.
- Each parent node is **greater or equal than both children** (max-heap) (less or equal – min-heap). Yes, in heaps we are definitely allowed to store equal keys.



Idea of heap ADT is to preserve the smallest (biggest) element on the top to allow very fast access and deletion of this value. When you put elements to DS, and **pick the topmost element without any parameters** (using some rule), this approach is called **pool** (stacks and queues are pools). There is a data structure (container) that expects exactly this behavior for implementation – **priority queue**. This is a kind of queue, but you provide not only element, but **also a priority number** (key), that can help an element to move faster through the queue.

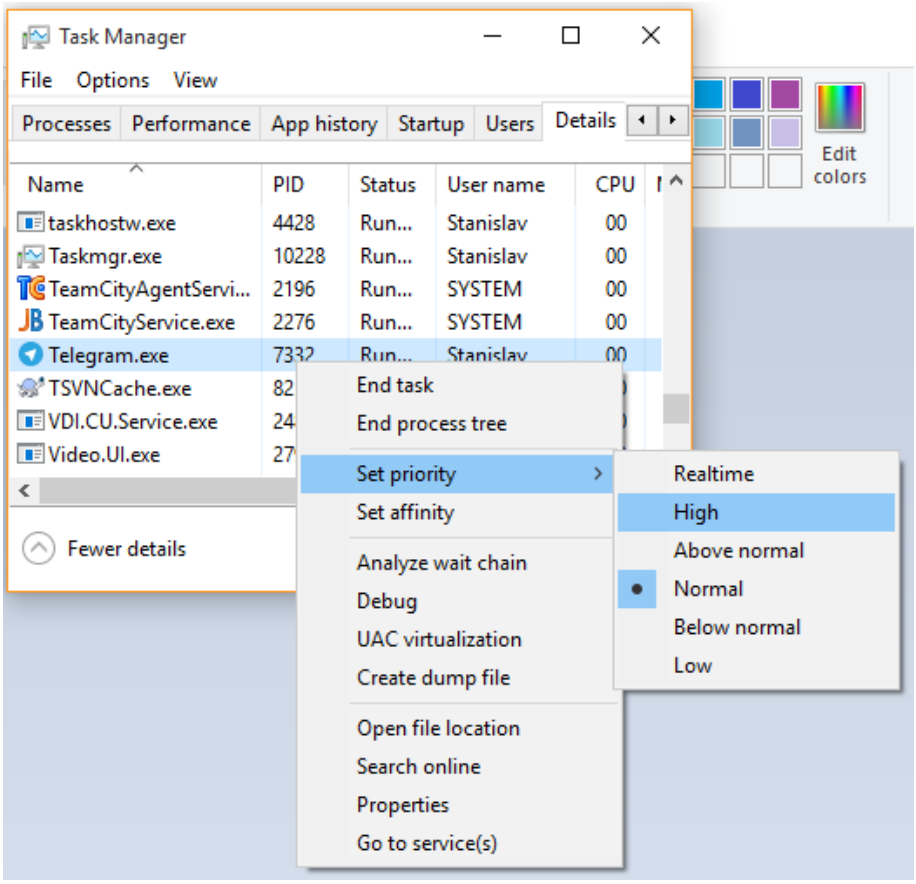
Heap operations

Upheap

When you insert a value into an array-based heap, you just add new value as a last element (according to fill property of complete tree).

What will happen if after insertion **parent value is less than inserted**? Let's become a bubble! Proceed changing values with parents. Do it either with tail recursion, or a loop. Heap height (any complete binary tree) is O(log n). This can make us confident, that loop will end in O(log n) time.

What are other upheap applications? Consider you use heap for scheduler, when you want to raise a priority of the process, it means we need to reconsider it's place in a queue.



What can we do it? Upheap!

Downheap

The main idea of the data structure is to preserve the smallest (biggest) element on the top to allow very fast **read** and **deletion** of this value. As you delete the topmost (or first element in terms of array-based heap), you will now have 2 orphan subtrees. How can we fix it? Let's put **the last** (most probably one of the smallest) element to the top. But we know this deadbeat doesn't worth with place. So, let him **sink to the bottom**! How (for max-heap)?

1) Select the biggest child.
2) Swap.
3) Repeat.

Downheap works the same way if you want to lower process priority in a queue.

Building a heap

What else can we do with the heap? We can build a heap from arbitrary data in O(n log n) time. How? There are few approaches all O(n log n). The easiest way is just start inserting elements from the first to the last. But there's a way to do it faster.

1) Take elements starting from the last one by one.
2) Proceed downheap. This will move bigger elements closer to the beginning of array.
3) Repeat until the first element.

Heap sort

Let's use heap building to build a sorted array! That's easy.

1) Build a heap from arbitrary array.
2) Loop N times
    a. Swap last element and first. Now last element in array is the biggest.
    b. Reduce heap size by one. We now do not consider last element as a part of array.
    c. Downheap.
    d. Repeat.

Practical part

1) Implement array based heap as a **Map** data structure.
    a. Downheap
    b. Upheap
    c. Heap building
    d. Reuse your k-ary tree.
2) Implement Priority Queue using your Heap.
3) Implement heap sort.
4) Implement either merge sort or quick sort.