Tutorial 1

Data Structures 1, 21. 2. 2025 `https://iuuk.mff.cuni.cz/~chmel/2425/ds1en/`

Exercise 1 (Heaps and Dijkstra)
Recall Dijkstra's algorithm and d-regular heaps. What is the asymptotic time complexity of Dijkstra's algorithm with a d-regular heap. Find the best d for optimizing the time complexity of Dijkstra's algorithm with a d-regular heap.

Solution
Heap complexity: insert $\mathcal{O}(\log_d n)$, min $\mathcal{O}(1)$, extractmin $\mathcal{O}(d \cdot \log_d n)$, increase $\mathcal{O}(d \cdot \log_d n)$, decrease $\mathcal{O}(\log_d n)$. Dijkstra complexity: $\mathcal{O}(n \cdot T_i + n \cdot T_x + m \cdot T_d)$, where T_i, T_x, T_d are the complexities of the operations Insert, ExtractMin, Decrease in this order. So, we get $\mathcal{O}(n \cdot (\log_d n) + n \cdot d \cdot (\log_d n) + m \cdot \log_d n)$ in total, and here we can observe that setting $d = m/n$ is the best choice, so we get $\mathcal{O}(n \cdot (\log_{m/n} n) + n \cdot m/n \cdot (\log_{m/n} n) + m \cdot \log_{m/n} n) = \mathcal{O}(n \cdot \log_{m/n} n)$.

Exercise 2 (Asymptotics)
Group the following functions into classes of functions with the same growth rate (i.e., for all f, g in one group, we have $f = \Theta(g)$), and then, compare the groups using o and ω: n, $42n + 7$, n^2, $\log n$, $\log_e n$, $\log(n^2)$, $(\log n)^2$, $\sqrt{n}$, 2^n, 2^{2n}, 4^n, $2^{n \log n}$, $2^{2 \log n}$, n^n, $n!$, $(n + 1)!$.
All logarithms which do not have a specified base are in base 2.

Solution
From the slowest-growing to the fastest-growing:

- $\log n, \log_e n, \log(n^2)$
- $(\log n)^2$
- $\sqrt{n}$
- $n, 42n + 7$
- $n^2, 2^{2 \log n}$
- 2^n
- $2^{2n}, 4^n$
- $n!$
- $(n + 1)!$
- $n^n, 2^{n \log n}$

Exercise 3 (Successor iteration)
In a BST on n nodes, we find the node with the minimum key, and then we use the successor operation $n - 1$ times. What is the total complexity of the operations?

Solution
We are essentially doing a DFS, so it is linear in n. (Another way of looking at it is that we traverse every edge in the tree exactly twice, and a tree has linearly many edges.)

Exercise 4 (Perfectly balanced, as all things should be)
Create an algorithm that, when given an ordered array, builds a perfectly balanced BST in linear time. (That is, for every node, it must hold that the number of nodes in the left subtree and the right subtree differ at most by 1.)

Solution
Divide and conquer: we consider the middle cell of the array (we either know the array beforehand, or we can calculate it by iterating through the array once) and then we recurse on the left subarray and right subarray (with the knowledge of the lengths of these two subarrays), where the middle cell will be the root of the tree.

Bonus exercises

Exercise 5 (Careful reading)
Can you find a formal error in the task Asymptotics?

Solution
$=$ should be $\in$ instead.

Exercise 6 (Interval update)
Let us have a BST as a dictionary of pairs (key, value) with numeric values. Modify the structure to also support the operation $\textsc{Add}(x, y, \delta)$ that adds δ to all values whose keys are in the interval $[x, y]$.
The operation should run in $\mathcal{O}(d)$ where d is the depth of the tree. This means that we do not have to immediately update the values corresponding to all keys in the interval. It sufficies if the operation $\textsc{Find}(k)$ returns the correct value corresponding to the key k.

Solution
TODO

Definition (Heap as a generic data structure)**.** Heap (in our case the minimum heap) is a data structure for storing a set of elements with keys that offers the following operations: INSERT(x): inserts the element x into the set
MIN(): finds the element with the minimum key
EXTRACTMIN(): removes the element with the minimum key and returns it

We can also add operations INCREASE, DECREASE - for an element (given to us by a pointer), we can either increase or decrease its key. Usually, we use $k(x)$ to denote the key of the element x.

Definition (Minimum binary heap)**.** Minimum binary heap is a data structure in the shape of a binary tree such that contains one element in each node and that also satisfies the following

1. *Heap shape*: All levels except for the lowest one are full. The lowest level is filled in left-to-right.
2. *Heap ordering*: For a node v and its child s, we have $k(v) \leq k(s)$.

The operations are perfomed as follows:
INSERT(x): Inserts x into the heap into the first free space in the lowest level, and then we "bubble" the element up until the heap ordering property is satisfied.
MIN(): Returns the root of the tree.
EXTRACTMIN(): The value in the root is swapped with the value of the right-most node v in the lowest level of the tree, we delete v and then we "bubble" the root down until the heap ordering property is satisfied.
Using the "bubbling", we can also implement the INCREASE, DECREASE operations.

Algorithm (Dijkstra with heap (a sketch))**.** Input: a graph $G = (V, E)$ with positive edge lengths $\ell : E \to \mathbb{R}^+$ and a starting vertex $v_0 \in V$.

1. insert all vertices into the heap with weights $+\infty$
2. DECREASE(v_0, 0) - decreasing the key of v_0 to zero
3. While the heap is nonempty:
   (a) v = EXTRACTMIN()
   (b) For all neighbors u of the vertex v: if $k(u) > k(v) + \ell(v, u)$, then DECREASE(u, $k(v) + \ell(v, u)$)

Theorem (Complexity of Dijkstra)**.** Dijkstra's algorithm with a heap has time complexity $\mathcal{O}(n \cdot T_i + n \cdot T_x + m \cdot T_d)$ where T_i, T_x, T_d are the complexities of operations INSERT, EXTRACTMIN, DECREASE in this order.