koutecky@iuuk.mff.cuni.cz

9. tutorial

1. Data Structure 1. Construct a (composite) data structure which can handle the following operations in the required time:

- `Init()` – initializes the data structure– $\mathcal{O}(1)$.
- `INSERT(`X`)` – inserts element X, if it is not yet in the structure – $\mathcal{O}(\log n)$.
- `DELETE(`X`)` – deletes X, if it is in the structure – $\mathcal{O}(\log n)$.
- `DELETE_IN_PLACE(`I`)` – deletes element which was the I-th added – $\mathcal{O}(\log n)$.
- `GET_PLACE(`X`)` – returns a number I such that X was the I-th added element – $\mathcal{O}(\log n)$.

2. Data Structure 2. An electrician wants to maintain a list of clients indexed by their IDs together with a record of whether they are male or female *(bonus task: handle more genders)*. Design a data structure which handles the following operations in the time $\mathcal{O}(\log n)$:

- `INSERT(`K`, `C`)` – inserts a new client C with `ID`$=K$, designates them female.
- `UPDATE(`K`)` – designates client with `ID`$=K$ as male.
- `FINDDIFF(`K`)` – finds the difference between the numbers of male and female clients among those with `ID` $\leq K$.

3. Subsequence. We are given a sequence of n numbers and we want to find the longest increasing subsequence (doesn't have to be contiguous) in time $\mathcal{O}(n \log n)$. (We have already seen this task in our first tutorial, and we could only solve it in time $\mathcal{O}(n^2)$ by finding the longest path in a DAG.)

4. Window. Numbers are arriving on input. Whenever a new number arrives, report the median and average of the last k numbers. Try to attain $\mathcal{O}(\log k)$ complexity per report.

5. (a, b) in one direction. Modify the `INSERT` and `DELETE` operations in (a, b)-trees so that they only make modifications on the way down.

6. List. Design a data structure for storing a list such that we can quickly find the k-the element and move it to the beginning of the list.

7. Sum. Say we have a set of natural numbers and a number x. We want to find out as quickly as possible whether our set contains a pair of elements which sum up to x.
What if I had a fixed x, but wanted my set to be dynamic, that is, I can `INSERT` and `DELETE` elements, and I want to be able to quickly check whether there is or isn't a pair of elements summing up to x. In what time is this possible?

8. Convenience Store. Frank's convenience store has customers come in and add orders into a queue; an order is a triple (item, quantitity, name of customer). Frank would like to have a good overview of whether he has enough goods of each kind in his storage.
Design a data structure for his convenience store, which will be able to execute the following operations in $\mathcal{O}(1)$ time:

(1) `ENQUEUE(`R`)` — enqueues the order R
(2) `DEQUEUE()` — prints the next order and removes it from the queueu
(3) `QUERY(`P`)` — for item P reports the total quantity of orders of this product.

(I'm assuming you know the FIFO queue data structure.) You are guaranteed that the queue will never contain more than m orders, and you know that there are n types of items in the store. Can you find a solution in space $\mathcal{O}(n)$? What about space $\mathcal{O}(m)$, in case that $m \ll n$?
You can assume that you can implement a Dictionary data structure such that `INSERT, FIND, DELETE` run in time $\mathcal{O}(1)$, even though we'll only see this later in the lectures.