Tutorial 3

Data Structures 1, 7. 3. 2025 https://iuuk.mff.cuni.cz/~chmel/2425/ds1en/

Leftovers from last time

Exercise 1 (Flexible arrays have potential!)
In the lecture, you have seen the analysis of the doubling flexible arrays. There, you used the aggregate method to argue that doubling the array when full and halving the array when at the quarter of its capacity gives amortized constant time.
Prove the same result using a potential defined as $\phi(\text{array}) = |\frac{\text{capacity of array}}{2} - \text{number of elements in array}|$.

Solution
Any non-resizing operation has constant cost and the potential changes by one at most. If we are resizing, then the potential is either $\frac{\text{capacity of array}}{2}$ if we are growing the array or $\frac{\text{capacity of array}}{4}$ if we are shrinking. Either way, we have a potential that is a constant multiple of the real cost of the operation. Thus, we can multiply the potential definition by a constant and the calculation works out in the end.

Splay trees

Exercise 2 (Splay trees might not have logarithmic depth)
Create a sequence of operations that builds a splay tree with linear depth that works for both naive and standard splay operation.

Solution
We run the operations Splay(1), Splay(2),...,Splay(n) in this order, or we can use finds instead of splays. Alternatively, we can insert in this order into an empty tree.

Exercise 3 (Why isn't naive splay enough?)
What breaks when the Splay operation is implemented naively, that is by only rotating the node up using only single rotations on the node? It is best if you construct a sequence of $m \geq n$ operations that has total cost larger than $m \log n$, but just saying what breaks in the proof of the amortized complexity also works.

Solution
Calling Splay(1),...,Splay(n) twice in a row leads to first building a left path, and then building a second left path above the first, which yields total cost $\mathcal{O}(n^2)$.

Exercise 4 (Splay trees, potentially)
Ensure that you understand the definition of the potential in splay trees and that you understand the main ideas of the amortized analysis of the splay operation.

a) What is the definition of the potential of a splay tree?

b) What is the amortized cost of a rotation (a single and a double)?

c) What is the amortized cost of the full splay operation and how does it follow from the amortized costs of rotations?

d) What is the *real* cost of the splay operation (and what units do we measure it in)?

e) What is the potential of a perfectly balanced BST? (A reasonable upper and lower bound are sufficient, assume the tree has $n = 2^k - 1$ nodes.)

f) What is the potential of a path? (Again, reasonable bounds are sufficient.)

Solution a) Let $T(v)$ be the subtree with root v, its size is $s(v) = |T(v)|$, its rank is $r(v) = \log_2(s(v))$, and the potential is $\Phi = \sum_{v \in V} r(v)$, and we (traditionally) use n to use the number of nodes in the tree.

b) Double $3r'(x) - 3r(x)$, where r' is rank after the operation, r is rank before, and a simple costs $3r'(x) - 3r(x) + 1$.

c) At most $3r'(x) - 3r(x) + 1 \in \mathcal{O}(\log n)$ - this works out, because the sums of rotation costs telescope..

d) The cost is the number of performed rotations, and we measure it in rotations.

e) $\sum_{i=0}^{k-1} 2^i \cdot (\log_2(2^{k-i}-1)) \leq \sum_{i=0}^{k-1} 2^i \cdot (\log_2(2^{k-i})) = \sum_{i=0}^{k-1} 2^i \cdot (k-i) = \sum_{\ell=1}^{k} \sum_{i=0}^{\ell-1} 2^i = \sum_{\ell=1}^{k} 2^\ell - 1 = 2^{k+1} - 1 - (k+1) = 2^{k+1} - k - 2 = 2(2^k - 1) - k = 2n - \log n$. We also have a trivial lower bound $n/4$, as a complete binary tree has $2^{k-2} \approx n/4$ nodes on the layer just above the leaves.

f) $\sum_{i=1}^{n} \log(i) = \log(n!) \geq \frac{n}{2} \log\left(\frac{n}{2}\right)$, and we can also bound $\log(n!) \leq n \log n$, protože $n! \leq n^n$.

Bonus exercises

Exercise 5 (Statically optimal BST)
We have a set of n keys $\{k_1, \ldots, k_n\}$ from which we want to build a static BST. At the same time, we know the probabilities of accessing each key $w_1, \ldots, w_n$ with $w_i > 0$ for all i.
Create an algorithm for building a statically optimal BST, that is a BST that minimizes the sum $\sum_{i=1}^{n} d(i) \cdot w_i$, where $d(i)$ is the depth of the key k_i and the root has depth 1. (That is, we are looking for such a tree such that the expected length of accessing a key is as small as possible.)
Hint: dynamic programming might be useful.

Solution
We use dynamic programming depending on the root: we will build a table $T[a, b]$ for $a \leq b$, which will be the minimal expected weight for a tree with keys $k_a, \ldots, k_b$. Immediately we get $T[a, a] = 0$, as then k_a must be in the root. Next, $T[a, b] = \min_{a \leq c \leq b}(T[a, c-1] + T[c+1, b] + \sum_{i=a}^{b} w_i)$.
This yields $\mathcal{O}(n^3)$ cost, as there are $\mathcal{O}(n^2)$ cells in the array and we need at most linear time per the computation for each array (provided we know all subinterval evaluations).

Exercise 6 (Analysing the successor using a potential)
During the first exercise, we showed that using $n-1$ successor operations on any BST, when starting in the node with minimum key, has total cost $\mathcal{O}(n)$. Prove the same result using a potential.

Solution
We define the potential $\Phi(x) := d + c_L(x) - c_R(x)$ where x is the current node, d is the maximum depth of the tree, $c_L(x)$ is the number of left children on the path from the root to x, and $c_R(x)$ is the number of right children on the path from the root to x.
In particular, the cost of finding the leftmost node costs its depth even amortized, but that is not an issue.
Let us consider some cases of applying the successor operation.

1. If x has a right child and y is the successor of x, then $c_R(x) - c_R(y) = -1, c_L(y) - c_L(x) =$ the depth of y under the right c and thus $\Phi(y) - \Phi(x)$ is on the order of the real cost of the operation.

2. If x does not have a right child, we go upwards until we first move into an ancestor from its left child. In that case, for y that is a successor of x, we have $c_R(x) - c_R(y) =$ the number of edges we have climbed up $- 1$ and $c_L(y) - c_L(x) = -1$, which again yields that $\Phi(y) - \Phi(x)$ is on the order of the real cost of operation.

We can also immediately note that the cost of potential is at least zero and at most $2h$.
Thus, except for the first operation running in time h we have all other operations amortized in $\mathcal{O}(1)$, and thus the total complexity is $\mathcal{O}(n + h) = \mathcal{O}(n)$ as $h \leq n$.