1 Solutions to HW3

Solution 1.1

Let $T$ be a tree, let the initial vertex is $x$ and let us call the sequence of vertices we visit when finding $k$ successors traverse of $T$. We call vertices which are among $k$ succesors the interval. Vertices which are smaller then any vertex from interval are small vertices and vertices bigger than any vertex from interval are big vertices, see Fig. 1. We also assume $u = \text{key}(u)$ for every vertex $u$.

![Figure 1: Small, big and interval vertices and visualization of $v, v_s$ and $v_b$.](image)

We first prove that traverse of $T$ never use an edge twice in the same direction, that already prove that we never visit any vertex in $T$ more than constant number of times, because degree of every vertex is bounded by 3. For contradiction assume we choose the same edge twice, it means, that in the moment we are going the second time we are in the same part of program as we were in the first time, which means that we return same successor vertex the second time, which means we will not find $k$ successors, which is contradiction.

Now we want to prove that when traversing $T$ we pass through at most $O(h)$ vertices outside interval. Let first prove that we visit at most $h$ small vertices.

For contradiction we assume there are two small vertices $v_s < v_b$ which have same height and both of them are on traverse of $T$. Assume the nearest common ancestor $v$ of $v_s$ and $v_b$, see Fig. 1. If we first visit $v_b$ then we cannot visit $v_s$, because if we get in any time to $v$ we either going up or to left child, and because $v$ is nerest common ancestor then $v_s$ is in left subtree. Assume we first visited $v_s$. Notice $v$ has to be small vertex, if not then $v_b$ is not small. Anytime after we visit $v_s$ we get to $v$ we never choose to go down because $v$ is not in interval, therefore $v_s$ and $v_b$ cannot be visited in same traverse of $T$ which proves we visit only at most $h$ small elements. An analogical proof applies for big elements.

Solution 1.2

Let start with a simple question. If we have a complete binary tree on $n$ vertices, when we color all vertices black, it is proper RB-tree and ratio is $0/n = 0$, and the ratio of any tree cannot be negative, so 0 is smallest such ratio (you can notice that such a tree is not possible to create by using insert operations,
because we proved in previous homework that such a tree on at last 2 vertices contains red vertex).

Because the problem is about internal nodes, we do not count the black NIL nodes. We will prove that ration can get as big as 2, but not more. First we show that there is RB-tree where the ration 2. Assume complete binary tree of even height, so that all vertices of even heights are red, where we immediately get that there is twice as many red vertices than black vertices, see Fig. ?? . For contradiction assume there is a tree $T$ which has the ration more than 2. For every red vertex has exactly one black parent. So assume there is ration more than 2, then by pigeonhole principle there is one black vertex which is parent of at least 3 vertices, which is in contradiction that $T$ is a binary tree.

**Solution 1.3**

Our original example contains operation INCREMENT and now we add decrement. We would first make $2^k$ operations of INCREMENT and the rest of the operations would be pairs of operations DECREMENT and INCREMENT. we can assume that first $2^k$ operations have to flip $\Theta(n)$ bits because of the proof from the class. And the rest $n - 2^k$ operations costs $k$ bit flips, for $n$ large enough is $2^k$ negligible (for example if $k \leq \log(n)$). So the operations costs us $2^k + (n - k)k = 2^k + kn - k^2 = \Omega(kn)$, opposite is trivial we get upper bound by assuming, that in each step we do not flip more then $k$ bits which gets us $O(kn)$.

**Solution 1.4**

Let us assume our stack operations are PUSH, POP, MULTIPOP (but we can assume also just PUSH and POP). We assign prize of 2 dollars to every operation of PUSH and 0 to POP and MULTIPOP. If we push element we pay one dollar for getting element into stack, and all elements in the stack has saved one dollar. We use this one extra dollar for operations POP and MULTIPOP. This proves that stack operations costs only constant amortize prize. Now we would like to introduce operation BACKUP, which takes place after every k operations and
costs the size of the stack. Now we change prizes of our operations, operation BACKUP costs 0 dollars, operations POP and multipop costs 1 dollar and operations PUSH costs 3 dollars. We proceed same way as previously so each element in stack have 1 dollar. But we also pay 1 dollar for every operations we made. After $k$ steps we have $k$ dollars for each operation. But because our stack has size at most $k$, we save enough to copy (at most) $k$ stack elements. That already conclude the proof.

Solution 1.5

Let $T$ be a binary tree and assume a path $p$ in $T$ from root to vertex $v$ (notice $p$ is uniquely defined), we now define weight $w$ of vertex $v$, it is the number of left edges (which are going to the left from parents to child) in the path $p$. We now assume sum $S = \sum_{v \in V} w(v)$, we can notice that $w(v)$ is at most $n$ for every $v$, so the sum $S$ is at most $n^2$. We can notice that after every right rotation the sum decreases at least by one, and the sum $S$ is positive for every binary tree. It already conclude that we cannot do more than $O(n^2)$ right rotations.