

Introduction to Algorithms and Data Structures Solutions

IT University of Copenhagen

June 11, 2004

Problem 1 (35%)

a) How many times is line 3 of MININDEX executed as a function of n and k ?

Line 3 is executed $n - (k + 1) + 1 = n - k$ times.

b) Consider a single call to procedure MINSORT. What is the total number of times line 3 of MININDEX is executed as a function of n ?

The total number of times line 3 in MININDEX is executed is given by the sum

$$\begin{aligned}\sum_{i=1}^{n-1} (n - i) &= \sum_{i=1}^{n-1} n - \sum_{i=1}^{n-1} i \\ &= (n - 1)n - \frac{n(n - 1)}{2} \\ &= \frac{n(n - 1)}{2}.\end{aligned}$$

c) What is the asymptotic time complexity of MINSORT?

We have $\frac{n(n-1)}{2} = \frac{n^2}{2} - \frac{n}{2} = O(n^2)$.

d) Use the loop invariant defined above to carefully prove correctness of procedure MININDEX.

Initialization: we have $A[m] = A[k]$ which is a minimum element of $A[k..k]$.

Maintenance: assume that the loop invariant holds at the start of some iteration of the loop. That is, $A[m]$ is a minimum element of subarray $A[k..i - 1]$. Let i' and m' denote the value of m and i at the start of the next iteration. We have, $i' = i + 1$. Due to line 3 and 4, we have

$$m' = \begin{cases} i & : A[i] < A[m] \\ m & : \text{otherwise} \end{cases}$$

Since $A[m]$ is a minimum element of $A[k..i - 1]$, we have that m' is a minimum element of $A[k..i]$. Thus as required for maintaining the loop invariant, m' is a minimum element of $A[k..i' - 1]$.

Termination: MININDEX terminates since the for loop in line 2-4 is bounded. When the loop terminates, we have $i = n + 1$. Thus, due to the loop invariant, MININDEX returns a minimum element of $A[k..n]$.

e) Write a procedure FIND(A, a, b, e) that returns *true* if $A[i] = e$ for some $1 \leq a \leq i \leq b \leq n$ and otherwise *false*. The asymptotic time complexity of FIND(A, a, b, e) should be $O(\lg n)$.

```

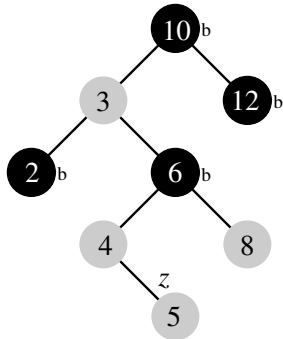
FIND( $A, a, b, e$ )
1  if  $a > b$  then return false
2  else
3       $m \leftarrow \lfloor (a + b)/2 \rfloor$ 
4      if  $e = A[m]$  then return true
5          else if  $e < A[m]$  then return FIND( $A, a, m - 1, e$ )
6          else return FIND( $A, m + 1, b, e$ )

```

The FIND procedure performs *binary search*. There are at most $O(\lg n)$ recursive calls to FIND since the subarray $A[a..b]$ is halved between each call. The complexity of the body of FIND (excluding the recursive calls) is $O(1)$, so we have that the asymptotic time complexity of FIND is $O(\lg n)$.

Problem 2 (20%)

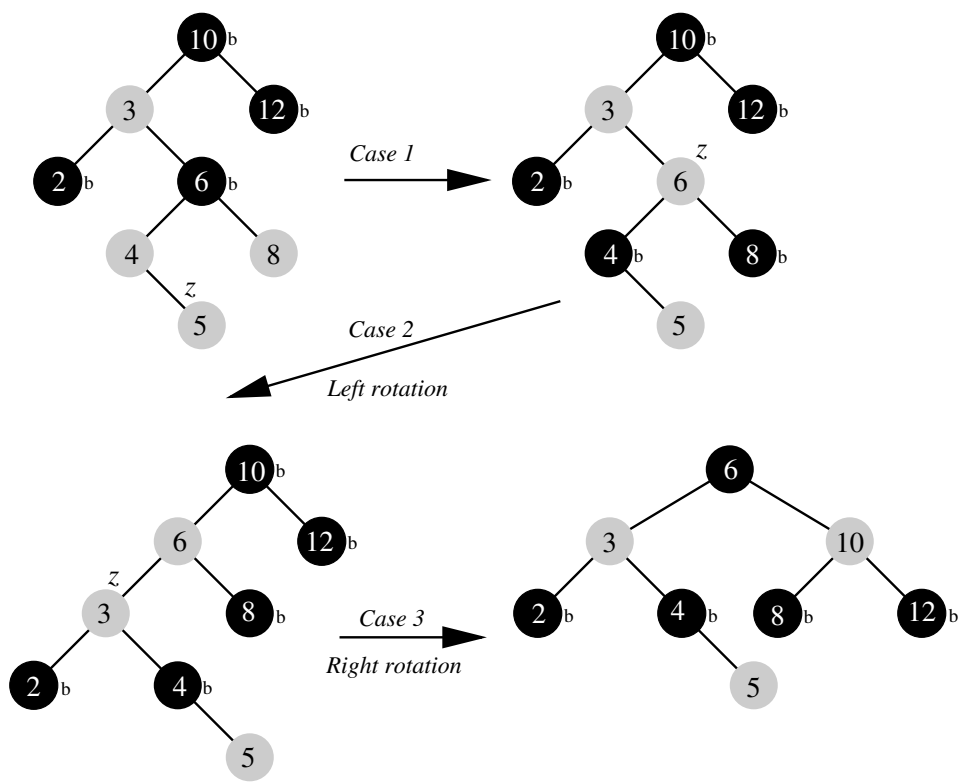
a) Rearrange the numbers such that T_{RB} satisfies the binary-search-tree property. Name the resulting tree T'_{RB} .



b) Which of the properties 1-5 on page 273 of CLRS are not satisfied by T'_{RB} ?

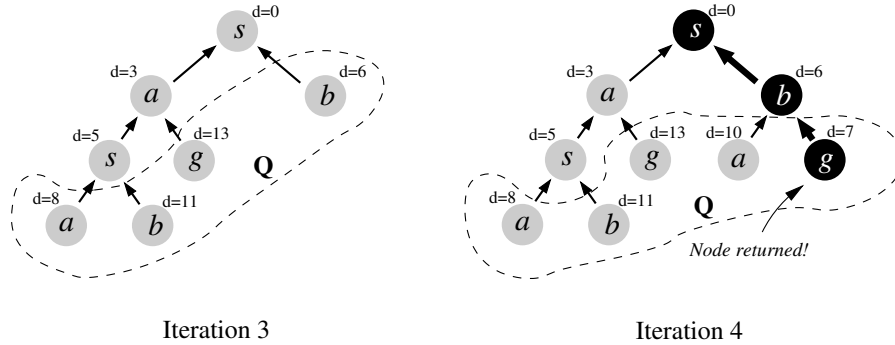
Property 4 is not satisfied. Property 1-3 and 5 are satisfied (recall that we do not draw leafs).

c) Illustrate the operations of $\text{RB-INSERT-FIXUP}(T'_{RB}, z)$ using the approach shown in Figure 13.4 on page 282 of CLRS. Make sure to clearly mark black nodes (e.g., with a small 'b' as shown for T_{RB}). Indicate for each rearrangement of the nodes, whether it was caused by a right or left rotation.

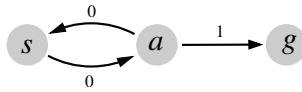


Problem 3 (20%)

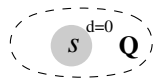
a) In the same style as above, draw the search tree after each of the remaining iterations of the while loop. Point out the node returned by SHORTESTPATH on the last search tree and mark the search nodes corresponding to the shortest path from s to g found by SHORTESTPATH.



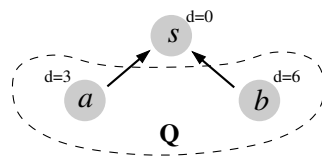
b) Give an example showing this.



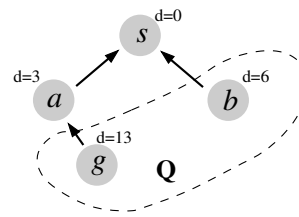
c) Consider a version of the SHORTESTPATH procedure employing this pruning rule. Draw the search tree after each iteration of this procedure given G_0 as input. Point out the node returned by the procedure on the last search tree and mark the search nodes corresponding to the shortest path from s to g found by the procedure.



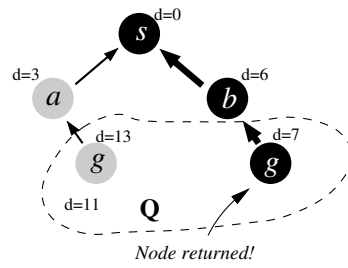
Iteration 0



Iteration 1



Iteration 2



Iteration 3

Problem 4 (25%)

a) Design a data structure (involving G and the list of evildoers) together with implementations of the two operations so that $\text{MOVE}(e, v)$ takes $O(1)$, and $\text{EXPLODE}(v)$ takes $O(\text{deg } v)$ worst-case time.

We represent vertices v as objects stored together with their adjacency lists $\text{Adj}[v]$ of (links to) vertices adjacent to v in G , and the link $\text{occ}[v]$ to the evildoer that currently occupies v . ($\text{occ}[v] = \text{NIL}$ if v is unoccupied.)

We assume that for evildoers e , their current positions in the graph are given by the field $\text{pos}[e]$.

```
MOVE( $e, v$ )
1    $u \leftarrow \text{pos}[e]$ 
2   if  $u \neq v$ 
3       then  $\text{occ}[u] \leftarrow \text{NIL}$ 
4           if  $\text{occ}[v] = \text{NIL}$ 
5               then  $\text{occ}[v] \leftarrow e$ 
6                    $\text{pos}[e] \leftarrow v$ 
7           else remove  $\text{occ}[v]$  from the list of evildoers
8                    $\text{occ}[v] \leftarrow \text{NIL}$ 
9                   remove  $e$  from the list of evildoers
```

Since removing an object from a doubly linked-list takes $O(1)$ time, each operation in the procedure above as well as the whole procedure perform in $O(1)$ time.

For an object a representing a vertex u in an adjacency list $\text{Adj}[v]$, let $\text{vertex}[a]$ be the link to u .

```
EXPLODE( $v$ )
1   if  $\text{occ}[v] \neq \text{NIL}$ 
2       then remove  $\text{occ}[v]$  from the list of evildoers
3            $\text{occ}[v] \leftarrow \text{NIL}$ 
4   for each element  $a \in \text{Adj}[v]$ 
5       do if  $\text{occ}[\text{vertex}[a]] \neq \text{NIL}$ 
6           then remove  $\text{occ}[\text{vertex}[a]]$  from the list of evildoers
7            $\text{occ}[\text{vertex}[a]] \leftarrow \text{NIL}$ 
```

We have $O(\text{deg } v)$ iterations of the loop with each iteration taking $O(1)$ time.

b) Suppose that explosions at v affect not just the evildoers, but that the vertex v itself together with all the edges incident on it disappear from the game. Adjust your data structure and algorithms so that $\text{MOVE}(e, v)$ and the new version of $\text{EXPLODE}(v)$ still only take $O(1)$ and $O(\deg v)$ time respectively.

We maintain the same data structures with the following modifications: The objects v representing vertices now have to maintain the information $\text{present}[v]$ on whether the vertex v is still present in G . An attempt to move an evildoer to a vertex absent from G should result in an error. Apart from that, the procedure $\text{MOVE}(e, v)$ is left unchanged.

Suppose the vertex v explodes. That vertex may have been connected to one or more surviving vertices u . Accordingly, one of the objects a in $\text{Adj}[u]$ must mention v : $\text{vertex}[a] = v$. If objects a corresponding to blown-up vertices were allowed to stay in $\text{Adj}[u]$ after explosions then such objects would accumulate and could eventually form the dominant population of $\text{Adj}[u]$. This would render us unable to claim that the traversal of the list $\text{Adj}[u]$ takes just $O(\deg u)$ time, $\deg u$ being the current degree of the vertex u . Hence a needs to be removed from $\text{Adj}[u]$ after the explosion.

To address this new requirement, the adjacency lists of each vertex v of the graph now have to be doubly-linked to facilitate fast removal of vertices. On top of that, with each object a in the adjacency list $\text{Adj}[v]$ of v we now store in a new field $\text{otherend}[a]$ the link to the object representing the vertex v in the adjacency list $\text{Adj}[\text{vertex}[a]]$ of $\text{vertex}[a]$. This is because $\text{otherend}[a]$ will have to be removed from $\text{Adj}[\text{vertex}[a]]$ in the event of explosion at v .

The new $\text{EXPLODE}'(v)$ now looks like this:

```
EXPLODE'(v)
1  EXPLODE(v)      ▷ use the old version to handle evildoers
2  for each element  $a \in \text{Adj}[v]$ 
3      do remove  $\text{otherend}[a]$  from  $\text{Adj}[\text{vertex}[a]]$ 
4   $\text{Adj}[v] \leftarrow \text{NIL}$ 
5   $\text{present}[v] \leftarrow \text{false}$ 
```

We have only added a loop that executes $O(\deg v)$ times, each iteration taking just $O(1)$ time.