

Data Structures and Algorithms

Exercise 6

Markus Innerebner, Romans Kasperovics
dsa(a]inf.unibz.it

Free University of Bozen - Bolzano, Italy

Thursday, April 9th, 2009

Solutions 06

Task 1a 2/2

```
NODE* TreeMinimum(NODE* x){
    while(x->left!=NULL) x=x->left;
    return x;
}

NODE* TreeSuccessor(NODE* x){
    if(x->right!=NULL) return TreeMinimum(x->right);
    NODE* y = x->parent;
    while(y!=NULL && x==y->right){
        x=y; y=y->parent;
    }
    return y;
}
```

Output: #./task1 → 1 3 5 5 7

Solutions 06

Task 1a 1/2

Implement an inorder tree walk by finding the minimum element and then invoking $n - 1$ the tree successor.

```
struct node {
    int key; struct node* left;
    struct node* right; struct node* parent;
};
typedef struct node NODE;

void InorderTreeWalk(NODE* x){
    if(x!=NULL){
        NODE* min = (NODE*)TreeMinimum(x);
        printf("%d \t",min->key);
        NODE* current;
        while((current = (NODE*)TreeSuccessor(min)) != NULL){
            printf("%d \t",current->key);
            min = current;
        }
    }
}
```

Solutions 06

Task 1b 1/3

Algorithm INORDER-TREEWALK runs in $\Theta(n)$ time because:

- requires $\Omega(n)$ time to do the n procedure calls
- traverses each of the $n - 1$ tree edges at most twice ($O(n)$)

To prove: each edge is traversed at most twice:

Consider edge (u, v) between any node u and either of its children v

Downward traversing of the tree only in TREE-MINIMUM

Upward traversing of the tree only in TREE-SUCCESSOR if a node has no right subtree

Solutions 06

Task 1b 2/3

Suppose v is u 's left child:

Before printing u , first all nodes in left subtree must be printed (rooted at v). Ensures that $\rightarrow (u, v)$ is traversed downward

After all nodes in u 's left subtree are printed, u must be printed next. TREE-SUCCESSOR traverses an upward path from the maximum element(no right subtree) to u . Path includes (u, v) .

Since all nodes in u 's left subtree are printed (u, v) is never traversed again

Solutions 06

Task 2a

Structure of radix tree:

```
# include <stdio.h>
# include <stdlib.h>
typedef enum { FALSE, TRUE } bool;

struct rnode {
    bool empty;
    struct rnode* left;
    struct rnode* right;
};
typedef struct rnode RNODE;

int A[100]; //global array to put the radix
```

Solutions 06

Task 1b 3/3

Suppose v is u 's right child:

- After u is printed, TREE-SUCCESSOR(u) is called. To get minimum element in u 's right subtree (rooted at v) (u, v) must be traversed downward

After all nodes in u 's right subtree are printed, TREE-SUCCESSOR is called on maximum element(no subtrees) in subtree rooted at v .

TREE-SUCCESSOR traversed a path up to tree from the maximum element to an element after u , since u was already printed.

Since all nodes in u 's right subtree are printed (u, v) is never traversed again

Hence, no edge is traversed twice in same direction.

Therefore algorithm INORDER-TREEWALK runs in $\theta(n)$ time.

Solutions 06

Task 2b 1/2

Print the radix tree in lexicographically sorted order:

```
void PrintPreorder(RNODE* x, int depth){
    if(x!=NULL){
        if(!x->empty) {
            int i;
            for(i=1;i<=depth;i++) printf("%d",A[i]);
            printf("\n");
        }
        A[depth+1]=0; PrintPreorder(x->left,depth+1);
        A[depth+1]=1; PrintPreorder(x->right,depth+1);
    }
}

int main() {
    RNODE* root = Init(); //fills tree with nodes
    PrintPreorder(root, 0);
}
```

Solutions 06

Task 2b 2/2

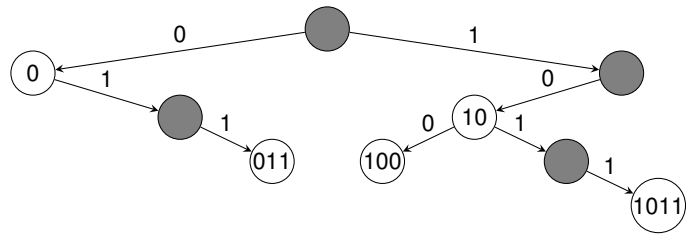


Figure: Radix Tree

Output: #./task2 → 0, 011, 10, 100, 1011