

Data Structures and Algorithms

Exercise 6

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

Free University of Bozen - Bolzano, Italy

Thursday, April 2nd, 2009

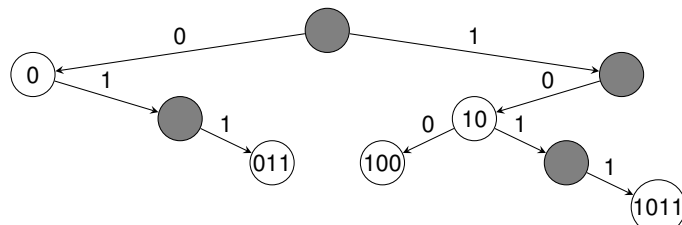
Assignment 06

Task 2

Given two bit strings $a = a_0a_1\dots a_p$ and $b = b_0b_1\dots b_q$, where each a_i and each b_j is either 0 or 1, we say that a is lexicographically less than b if either

- there exists an integer j , where $0 \leq j \leq \min(p, q)$, such that $a_i = b_i$ for all $0 \leq i \leq j - 1$ and $a_j < b_j$, or
- $p < q$ and $a_i = b_i$ for all $0 \leq i \leq p$

The above lexicographic ordering of bit strings can be used to build a radix tree that stores a set of bit string keys. For example, the radix tree data structure shown in the figure stores the bit strings 0, 10, 100, 011, and 1011.



Assignment 06

Task 1

An inorder tree walk of an n -node binary search tree can be implemented by finding the minimum element in the tree with the function `TREEMINIMUM` and then making $n-1$ calls to function `TREESUCCESSOR`. Use as example the tree of figure 2

- Implement an inorder tree walk in C with the two functions `NODE* TreeMinimum(NODE* x)` and `NODE* TreeSuccessor(NODE* x)`.
- Prove that this algorithm runs in $\Theta(n)$ time

Assignment 06

Task 2

When searching for a key we go left at a node at depth i if $a_i = 0$ and right if $a_i = 1$.

- Define a C data structure for radix trees. The nodes shall not store the actual bit string keys.
- Write a function that prints the set of binary strings represented by a radix tree in sorted order.

Hints

Defining Tree

```
struct node {
  int key;
  struct node* left;
  struct node* right;
  struct node* parent;
}; // semicolon here is important!
typedef struct node NODE; // now 'NODE' is a shorthand for 'struct node'
```

Hints

Inorder Tree Walk

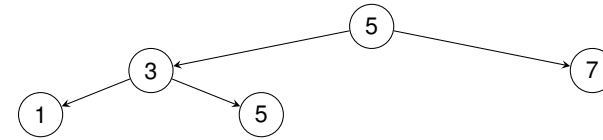


Figure: Example: Binary Search Tree

```
void InOrderTreeWalk(NODE* x){
  if(x≠NULL){
    InOrderTreeWalk(x→left);
    printf("%d", x→key);
    InOrderTreeWalk(x→right);
  }
}
```

Output: 1 → 3 → 5 → 5 → 7

Hints

Preorder Tree Walk

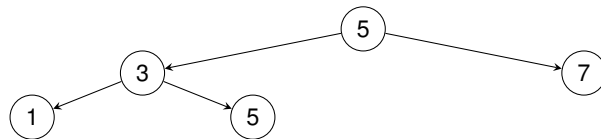


Figure: Example: Binary Search Tree

```
void PreOrderTreeWalk(NODE* x) {
  if(x≠NULL){
    printf("%d", x→key);
    PreOrderTreeWalk(x→left);
    PreOrderTreeWalk(x→right);
  }
}
```

Output: 5 → 3 → 1 → 5 → 7

Hints

Postorder Tree Walk

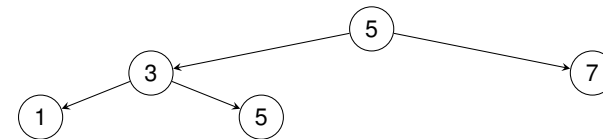


Figure: Example: Binary Search Tree

```
void PostOrderTreeWalk(NODE* x) {
  if(x≠NULL){
    PostOrderTreeWalk(x→left);
    PostOrderTreeWalk(x→right);
    printf("%d", x→key);
  }
}
```

Output: 1 → 5 → 3 → 7 → 5