Kazakh-British Technical University

Algorithms and Data Structures, Spring 2011

Lecture 4: Trees

1 Introduction

During this lecture we will consider trees as some data structure in spite of there exists a definition from the graph theory. Generally, tree in terms data structure is a some hierarchy with parents and children. In computer science such approach can be helpful and even crucial in some related problems.

Suppose that we write an application which needs to store a family tree. It is obvious that we have to organize data in the way where parent and child are related to each other. There are several solutions may occur such as adjust links from each parent to his children or conversely adjust child to his father.

As for tree data structure we consider a set of nodes where each node has a link to his father. For example,

```
struct node{
  string name;
  int father;
};
```

And the whole tree is a set of nodes:

```
node tree[1000];
int size = 0; // size of tree
```

We also need one main element – root of the tree.

```
int root = -1; // by default there is no index of root;
```

Let us firstly write a function which by given `string` name can find its index in `tree`.

```
int find_index(string name){
  for(int i=0; i<size; i++)
    if (tree[i].name == name)
      return i;
  return -1;
}
```

Now suppose that we have the following queries from input:

```
father1 son1
father2 son2
...
fathern sonn
```

Then we can store given tree in our implementation as:

```
string father, son;
while (cin >> father >> son){
  if (find_index(father) == -1){ // father is not stored in our tree yet
    size++;
    tree[size-1].name = father;
    tree[size-1].father = -1; // new father at this moment has no parent;
  }
  if (find_index(son) == -1){ // son is not stored in our tree yet
    size++;
    tree[size-1].name = son;
    tree[size-1].father = find_index(father);
  }
  else
    tree[find_index(son)].father = find_index(father);
}
```

Next function we will implement is a function which output all the predecessor of given person.

```
void print_all_predecessors(string name){
  int x = find_index(name);
  while (tree[x].father!=-1){
    cout << tree[tree[x].father].name << endl;
    x = tree[x].father;
  }
}
```

For example, input is the following:

```
Alibek Ali
Alibek Vali
Ali Alik
Ivan Alexey
Ivan Alexandr
Ivan Alibaba
Alibaba Alibek
```

And the whole program:

```
#include <iostream>
using namespace std;

struct node{
  string name;
  int father; // by default he has no father
};
node tree[1000];
int size = 0; // size of tree

int find_index(string name){
  for(int i=0; i<size; i++)
    if (tree[i].name == name)
      return i;
  return -1;
}

void print_all_predecessors(string name){
  int x = find_index(name);
  while (tree[x].father!=-1){
    cout << tree[tree[x].father].name << endl;
    x = tree[x].father;
  }
}

int main(){
  freopen("in", "r", stdin);
  string father, son;
  while (cin >> father >> son){
    if (find_index(father) == -1){ // father is not stored in our tree yet
      size++;
      tree[size-1].name = father;
      tree[size-1].father = -1; // new father at this moment has no parent;
    }
    if (find_index(son) == -1){ // son is not stored in our tree yet
      size++;
      tree[size-1].name = son;
      tree[size-1].father = find_index(father);
    }
    else
      tree[find_index(son)].father = find_index(father);
  }
  print_all_predecessors("Alik");
  return 0;
}
```

The output is:

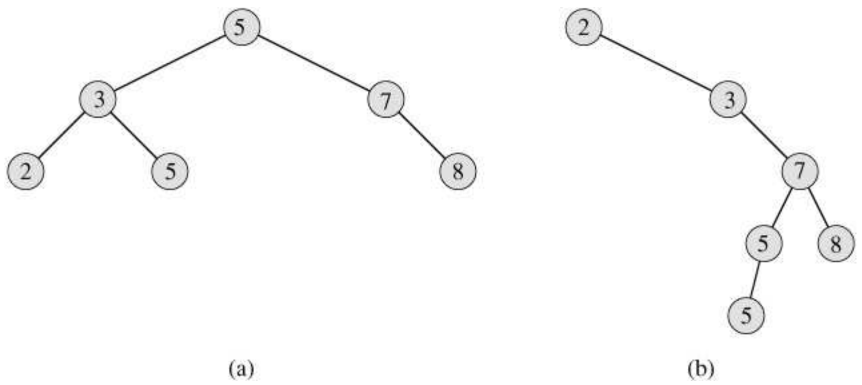
```
Ali
Alibek
Alibaba
Ivan
```

2 Binary Search Trees

Binary Search Tree is a data structure which is based on binary tree, a tree where each node has at most two children. So, each node represented with its key value, left and right child, and link to the parent.

```
struct node{
  int key;
  int left;
  int right;
  int parent;
}
```

The root of the tree is a node without parent (or parent = -1).



All keys in binary search tree organized in the way so that:
Let x be a node. Then

- all elements from left subtree of x have keys less or equal $\leq$ than $key[x]$ and
- all elements from right subtree of x have keys greater or equal $\geq$ than $key[x]$.

Traversal of binary search trees

Suppose that we want to print all the elements of given binary search tree in sorted order. Here we present an inorder tree walk – a recursive algorithm of needed traversal.

```
void inorder_tree_walk(int x){
  if (x != -1){
    inorder_tree_walk(tree[x].left);
    cout << tree[x].key << endl;
    inorder_tree_walk(tree[x].right);
  }
}
```

Tree search

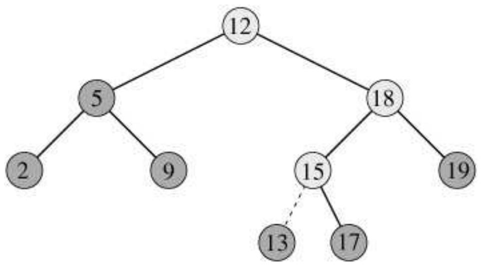
```
int tree_search(int x, int k){
  if (x == -1 || k == tree[x].key)
    return x;
  if (k < tree[x].key)
    return tree_search(tree[x].left, k);
  else
    return tree_search(tree[x].right, k);
}
```

Min and Max

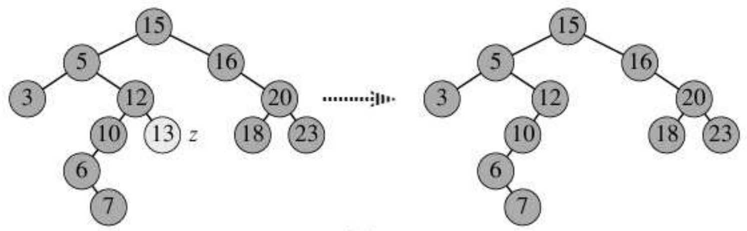
```
int tree_min(int x){
  while (tree[x].left != -1)
    x = tree[x].left;
  return x;
}

int tree_max(int x){
  while (tree[x].right != -1)
    x = tree[x].right;
  return x;
}
```

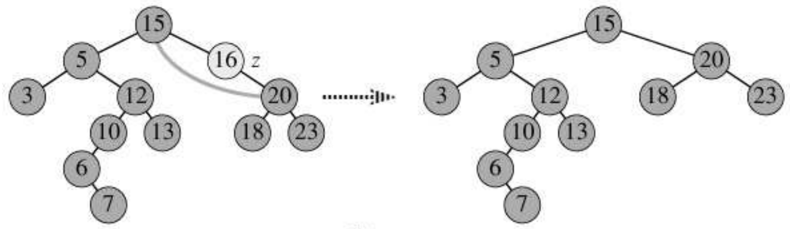
Insertion and Deletion



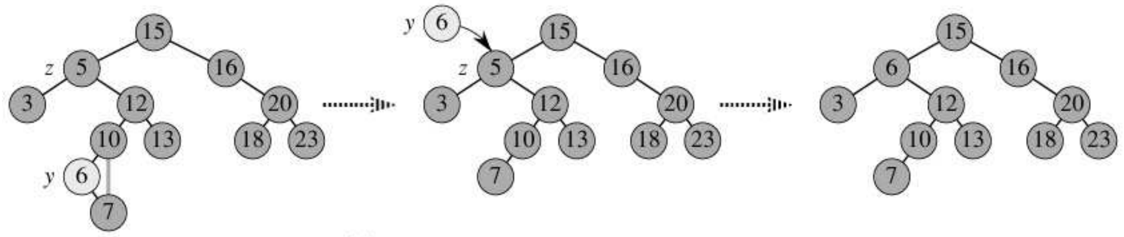
```
void tree_insert(int key){
  size++;
  tree[size-1].key = key;
  int y = -1;
  int x = root;
  while (x != -1){
    y = x;
    if (tree[size-1].key < tree[x].key)
      x = tree[x].left;
    else
      x = tree[x].right;
  }
  tree[size-1].parent = y;
  if (y == -1)
    root = size-1;
  else
    if (tree[size-1].key < tree[y].key)
      tree[y].left = size - 1;
    else
      tree[y].right = size - 1;
}
```



(a)



(b)



(c)

References

[1] [chapter 12] Thomas H. Cormen, Charles E. Leiserson. *Introduction to algorithms – 2-nd edition.* – USA : MIT Press, 2001. – 1180p.