Kazakh-British Technical University

Algorithms and Data Structures, Spring 2011

Lecture 5: Problem Cases for trees data structure

1 Map Demonstration

```
#include <iostream>
#include <map>

using namespace std;

int main(){
  map<string, int> age; // declaration of empty map
  // insertion of several elements in map
  age["Ali"] = 31;
  age["Alibek"] = 15;
  age["Alizhan"] = 12;
  age["Alisher"] = 20;
  age["Aliamir"] = 17;
  age["Ivan"] = 35;
  age["Alibaba"] = 70;

  // change the age of Alibaba
  age["Alibaba"] += 10;
  cout << "Alibaba: " << age["Alibaba"] << endl;

  // output of element which is not in our map
  cout << "neAli: " << age["neAli"] << endl;
  return 0;
}
```

```
Alibaba: 80
neAli: 0
```

```
#include <iostream>
#include <map>

using namespace std;

int main(){
  map<string, int> age; // declaration of empty map
  // insertion of several elements in map
  age["Ali"] = 31;
  age["Alibek"] = 15;
  age["Alizhan"] = 12;
  age["Alisher"] = 20;
  age["Aliamir"] = 17;
  age["Ivan"] = 35;
  age["Alibaba"] = 70;

  // change all elements of map
  for(map<string, int>::iterator it = age.begin(); it != age.end(); ++it){
    pair<string, int> tmp = *it;
    cout << tmp.first << " is " << tmp.second << " years old" << endl;
  }
  return 0;
}
```

```
Ali is 31 years old
Aliamir is 17 years old
Alibaba is 70 years old
Alibek is 15 years old
Alisher is 20 years old
Alizhan is 12 years old
Ivan is 35 years old
```

```cpp
#include <iostream>
#include <map>

using namespace std;

int main(){
  map<string, int> age; // declaration of empty map
  // insertion of several elements in map
  age["Ali"] = 31;
  age["Alibek"] = 15;
  age["Alizhan"] = 12;
  age["Alisher"] = 20;
  age["Aliamir"] = 17;
  age["Ivan"] = 35;
  age["Alibaba"] = 70;

  // output of all elements
  map<string, int>::iterator it;
  for(it = age.begin(); it != age.end(); ++it){
    cout << it -> first << " is " << it -> second << " years old" << endl;
  }

  // result after one year
  for(it = age.begin(); it != age.end(); ++it){
    it -> second ++;
  }

  //output all elements after one year
  for(it = age.begin(); it != age.end(); ++it){
    cout << "Next year " << it -> first << " will be " << it -> second << " years old" << endl;
  }
  return 0;
}
```

```
Ali is 31 years old
Aliamir is 17 years old
Alibaba is 70 years old
Alibek is 15 years old
Alisher is 20 years old
Alizhan is 12 years old
Ivan is 35 years old
Next year Ali will be 32 years old
Next year Aliamir will be 18 years old
Next year Alibaba will be 71 years old
Next year Alibek will be 16 years old
Next year Alisher will be 21 years old
Next year Alizhan will be 13 years old
Next year Ivan will be 36 years old
```

2 Binary Search Tree Implementation

```
#include <iostream>

using namespace std;

struct node{ // each element of tree
  int key; // value
  int left; // link to index of left subtree
  int right; // link to index of right subtree
  int parent; // link to index of parent
};

node tree[1000]; // tree is array of nodes
int size = 0; // real size of tree2
int root = -1; // index of root

// walk to output in sorted order
void inorder_tree_walk(int x){
  if (x != -1){
    inorder_tree_walk(tree[x].left); // recursive function
    cout << tree[x].key << " "; // output current minimal element
    inorder_tree_walk(tree[x].right);
  }
};

// search key in subtree with node index x
int tree_search(int x, int key){
  if (x == -1 || key == tree[x].key)
    return x; // return index
  if (key < tree[x].key)
    return tree_search(tree[x].left, key);
  else
    return tree_search(tree[x].right, key);
};

// find index of minimal element
int tree_min(int x){
  while (tree[x].left != -1)
    x = tree[x].left;
  return x;
};

// find index of maximal element
int tree_max(int x){
  while (tree[x].right != -1)
    x = tree[x].right;
  return x;
};

//find index of successor -- i.e.
//index of prev. element which less that key[x]
int tree_successor(int x){
  if (tree[x].right != -1)
    return tree_min(tree[x].right);
  int y = tree[x].parent;
  while (y != -1 && x == tree[y].right){
    x = y;
    y = tree[y].parent;
  }
  return y;
};

// insert new element into index pos with value key
void tree_insert(int key, int pos){
  tree[pos].key = key;
  int y = -1;
  int x = root;
  while (x != -1){
    y = x;
    if (key < tree[x].key)
      x = tree[x].left;
    else
      x = tree[x].right;
  }
  tree[pos].parent = y;
  if (y == -1)
    root = pos;// tree was empty
  else
    if (key < tree[y].key)
      tree[y].left = pos;
    else
      tree[y].right = pos;

};
```

```
//delete element from index pos
void tree_delete(int pos){
  int y, x;
  if (tree[pos].left == -1 || tree[pos].right == -1)
    y = pos;
  else
    y = tree_successor(pos);
  if (tree[y].left != -1)
    x = tree[y].left;
  else
    x = tree[y].right;
  if (x != -1)
    tree[x].parent = tree[y].parent;
  if (tree[y].parent == -1)
    root = x;
  else
    if (y == tree[tree[y].parent].left)
      tree[tree[y].parent].left = x;
    else
      tree[tree[y].parent].right = x;
  if (y != pos){
    tree[pos].key = tree[y].key;
  }
};

int main(){
  freopen("input.txt", "r", stdin);
  freopen("output.txt", "w", stdout);
  // initialize all values of tree by -1;
  for(int i=0; i<1000; i++){
    tree[i].left = -1;
    tree[i].right = -1;
    tree[i].parent = -1;
  }

  int n; cin >> n; // number of elements
  int x;
  for(int i=0; i<n; i++){
    cin >> x; size++; // read new key and increase size
    tree_insert(x, size - 1); // inset this element
    inorder_tree_walk(root); // output elements in sorted order
    cout << endl;
  }
  cout << "----------------" << endl;
  cin >> x; // read new element to delete
  tree_delete(tree_search(root, x)); // delete element with key=x
  inorder_tree_walk(root);//output elements in sorted order
  return 0;
}
```

Input:

```
9
12 18 5 19 9 2 15 13 17
12
```

Output:

```
12
12 18
5 12 18
5 12 18 19
5 9 12 18 19
2 5 9 12 18 19
2 5 9 12 15 18 19
2 5 9 12 13 15 18 19
2 5 9 12 13 15 17 18 19
----------------
2 5 9 13 15 17 18 19
```

References

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