Report of assignment 2 of algorithm and data structure

Zhiqing Zhong(CID:01333269)

Introduction：

In this assignment, the task is to achieve the task of reducing nodes of the binary trees, which the main purpose is to get the value of the base of the trees.

Building trees:

First what must do is to build a binary tree and the method of using is the recursion. The rules of building trees pass left when the number is 0 and pass right when the number is 1. And at the bottom of the tree, which is the leaves, should be filled with 0. Then according to the input of the strings what should do is change the base value with into 1, and all others remain 0.

```
bdt constree ( std::string n , bdt l,bdt r,int counter) {
    bdt tmp = new bdnode ;
    tmp->val = n+std::to_string(counter);
    tmp->left =l;
    tmp->right =r;
    return tmp ;
}
bdt instree (bdt t ,int c,int n ,int counter) {
    if ( c==n) {
        bdt t = new bdnode ;
        t->val = "0";
        t->left =NULL;
        t->right =NULL;
        return t ;
    }
    else {
        t->left=constree("x", NULL, NULL,counter);
        t->right=constree("x", NULL, NULL,counter);
        t->left = instree (t->left,c+1,n,counter+1) ;
        t->right = instree (t->right,c+1 ,n,counter+1) ;
    }
    return t;
}
```

And the change function has been shown:

```
bdt changeone(std::string s,int x,int n,int m,bdt tmp){
    bdt tmpp=new bdnode;
    tmpp=tmp;
    for (int i=0; i<n; i++) {
        if (s[i]==49) {
            tmp=tmp->right;
        }
        else{
            tmp=tmp->left;
        }
    }
    tmp->val="1";
    return tmpp;
}
```

These functions could be used to build the tree, they were very similar to the function of the first assignment, the code should comply with c++11. Due to the function of the std::to_string.

Function of cutting the node:

And the algorithm of reducing node is using the pass-through to pass all elements of the trees and chose every node from the base to the top which means the subtree would become large each time and the function to achieve is also using the recursion

The next step is store into two different vectors, then we need to compare to different vector elements. Basically, if each element is the same which means that these to subtrees are totally the same. In addition, the reconnect function could be used to delete the elements and connect the first code with the left one.

Due to the function of compareall is easily to use the recursion to compare each node and subtree, and the compare function is to judge whether these two sub trees are same, so and pass the two pointer of function, so using the void function might be much more useful.

```
bdt compareall(bdt t,std::vector<std::string>v1,std::vector<std::string>v2){
    if (t->left!=NULL&&t->right!=NULL) {
        t->right=compareall(t->right, v1, v2);
        t->left=compareall(t->left, v1, v2);
        compare(t->right,t,v1,v2);
        compare(t->left,t,v1,v2);
        return t;
    }
    else{
        return t;
    }
}
void compare(bdt& t,bdt& pt,std::vector<std::string>v1,std::vector<std::string>v2){
    if (t->left!=NULL&&t->right!=NULL) {
        int c=0;
        int total1=0;
        int total2=0;
        pass(t->left,v1,total1);
        pass(t->right,v2,total2);
        if (v1.size()==v2.size()) {
            for (int i=0; i<v1.size(); i++) {
                if(v1[i]==v2[i]){
                    c++;
                }
            }
        }
        if (c==v1.size()&&c!=total1) {
            reconnect(t, pt);
        }
    }
}
```

Alternatively, when I was trying to put the normal compare function with the compareroot function together, it might cause some problems, which is that the normal compare function might connect the root to the subtree, and the compareroot function makes the pointer point to the right subtree, and it would be very hard to find the whether it is the root or not. Additionally, the function of compareroot need only return the pointer so, it could be written into the bdt function.

```
bdt compareroot(bdt t,std::vector<std::string>v1,std::vector<std::string>v2){
    int total1=0;
    int total2=0;
    pass(t->left,v1,total1);
    pass(t->right,v2,total2);
    int c=0;
    if (v1.size()==v2.size()) {
        for (int i=0; i<v1.size(); i++) {
            if(v1[i]==v2[i]){
                c++;
            }
        }
    }
    if (c==v1.size()) {
        reconnectroot(t);
        return t;
    }
    else{
        return t;
    }
}
```

The reconnect functions are used to delete the node/subtrees and connect the pervious node with the new node together , the bdt function could be used for the recursion, and the same reason why the reconnect function of using void but not the bdt function is there were two values which might pass by referemce.

```
void reconnect(bdt& t,bdt& pt){
    reconnection(t->right);
    if(pt->left==t){
        bdt x=new bdnode;
        x=t;
        pt->left=t->left;
        delete x;
    }
    else{
        bdt x=new bdnode;
        x=t;
        pt->right=t->left;
        delete x;
    }
}
bdt reconnection(bdt t){
    if (t!=NULL) {
        reconnection(t->right);
        reconnection(t->left);
        delete t;
        return t;
    }
    return t;
}
bdt reconnectroot(bdt t){
    reconnection(t->right);
    bdt x=new bdnode;
    x=t;
    t=t->left;
    delete x;
    return t;
}
```

Test function:

```
std::string evalcompactbdt(bdt t,const std::string& input){
    std::string value ;
    t=passnew(t, input, value);
    return value;
}
bdt passnew(bdt t,const std::string input,std::string& value){
    std::string sameinput1;
    std::string counter;
    if (t->left==NULL&&t->right==NULL) {
        value=t->val;
        return t;
    }
    else{
        sameinput1=t->val;
        sameinput1.erase(0,1);
        int i=std::stoi(sameinput1,nullptr,0);
        counter=input[i-1];
        if (counter=="0"&&t->left!=NULL&&t->right!=NULL) {
            return passnew(t->left, input, value);
        }
        if (counter=="1"&&t->left!=NULL&&t->right!=NULL) {
            return passnew(t->right, input, value);
        }
        else{
            return t;
        }
    }
}
```

The test function is using the function called the erase and std::stoi the elements which has been cut, and the strand of using the function uses the nullptr, which is the empty pointer. Also, the recursion function has been used into the test function which might get the true value.

Conclusion:

As the result, the final number of the node should be decreased, and it should be worked well for most of the situations.

Reference: notes from lectures, data structure algorithms and applications for c++(Sartaj Sahni)