Report for assignment 4

Shashwat Gupta (14IE10028)

February 8, 2016

1 Maximum Sum from root to node

Given a binary tree storing integer keys, find the path from the root to a leaf node that has the maximum sum, output the keys separated by a blank and terminated by double LF.

maxSumPathRoot calls the Final function which recursively calls itself. The initial sum, node reference sum and final node reference pointer are passed. The work of Final is to recursively compute the maximum sum and also find the final leaf node which comprises the maximum sum path and this is returned back to the maxSumPathRoot function.
The function recurses from one node to the next and adds the data of each node to sum. Once a leaf node arrives, the sum counter is checked with the maximum sum counter and if it is greater then the max sum counter as well as final leaf node pointer are updated.
The recursion ends when all nodes have been traversed and the max sum and leaf node have been found.

The following is the Code of void Final

```
void Final (node *t, int *max, int sum, node **point_node)
{
    if (t == NULL)
        return;
    sum = sum + t->data;
    if (t->lChild == NULL && t->rChild == NULL)
    {
        if (sum > *max)
        {
            *max = sum;
            *point_node = t;
        }
    }
    Final (t->lChild, max, sum, point_node);
    Final (t->rChild, max, sum, point_node);
}
```

Listing 1: elimin_solve

Once Final is terminated, maxSumPathRoot has the max sum and the leaf node. These are passed to printPath for printing node to leaf node.

The following is the Code of int maxSumPathRoot

```
int maxSumPathRoot (node *t)
{
    if (t == NULL)
        return 0;
    node *point_node;
    int max_sum = -100;
    Final (t, &max_sum, 0, &point_node);
    printPath (t, point_node);
    return max_sum;
}
```

Listing 2: elimin_solve

The only difference in revmaxSumPathRoot is that it calls revPrint instead of printPath.

The following is the Code of int revmaxSumPathRoot

```
int revmaxSumPathRoot (node *t)
{
    if (t == NULL)
        return 0;
    node *point_node;
    int max_sum = -100;
    Final (t, &max_sum, 0, &point_node);
    revPrint (t, point_node);
    return max_sum;
}
```

Listing 3: elimin_solve

2 Maximum Sum from node to node

Given a binary tree storing integer keys, find the path (between any pair of nodes) in it that has the maximum sum, output the keys separated by a blank and terminated by double LF.

For each node there can be four ways that the max path goes through the node:

- Node only
- Max path through Left Child + Node
- Max path through Right Child + Node
- Max path through Left Child + Node + Max path through Right Child

This idea is being used. Ultimately the special node whos both childs comprise the maximum sum path is found. Once this node is found, we make use of the maxSumPathRoot and revmaxSumPathRoot functions undependently to get the path on the node's left child and right child. This gives us the max sum path.

The following is the Code of int maxSumFunc

```
int maxSumFunc(node *root, int *final, node **nodeBothChild)
{
    if (root == NULL)
        return 0;
    int l = maxSumFunc(root->lChild,final,nodeBothChild);
    int r = maxSumFunc(root->rChild,final,nodeBothChild);
    int max_single = MAX(MAX(l, r) + root->data, root->data);
    int maxBothChild = MAX(max_single, l + r + root->data);
    if(maxBothChild>*final)
    {
        *final=maxBothChild;
        *nodeBothChild=root;
    }
    return max_single;
}
```

Listing 4: check_print

nodeBothChild is found and is used to call maxSumPathRoot and revmaxSumPathRoot functions undependently in the int main() to print the max sum path.

3 Printing

There are two printing functions being used in the code. One is for printing nodes from root to leaf (revPrint) and the other is for printing from leaf to root (printPath).

The following is the Code of int printPath

```
int printPath (node *root, node *point)
{
    if (root == NULL)
        return 0;
    if (root==point || printPath(root->lChild, point)==1 ||
    printPath(root->rChild, point)==1)
    {
        printf("%d ", root->data);
        return 1;
    }
    return 0;
}
```

Listing 5: check_print

The following is the Code of int revPrint

```
int revPrint(node *root, node *point)
{
    if (root == NULL)
        return 0;
    if (root==point || revPrint(root->lChild, point)==1 || revPrint
    (root->rChild, point)==1)
    {
        a[k]=root->data;
        k++;
        return 1;
```

```
    }
    return 0;
}
```

Listing 6: check_print

The reverse printing is done using a global array which stores the numbers and prints them in reverse.

4 Main

The following code is the main function that runs when the program is executed.

The following is the Code of int main

```
int main()
{
    int i;
    srand(time(NULL));
    node *root = (node*)malloc(sizeof(node));
    node *nodeBothChild = (node*)malloc(sizeof(node));
    root = create(100, root);
    inorder(root);
    printf("\n\n");
    preorder(root);
    printf("\n\n");
    postorder(root);
    printf("\n\n");
    int sum = revmaxSumPathRoot(root);
    for(i=k-1;i>=0;i--)
        printf("%d ",a[i]);
    printf("\n\n");
    k=0;
    sum = -100;
    i=maxSumFunc(root, &sum, &nodeBothChild);
    maxSumPathRoot(nodeBothChild->lChild);
    printf("%d ", nodeBothChild->data);
    revmaxSumPathRoot(nodeBothChild->rChild);
    for(i=k-1;i>=0;i--)
        printf("%d ",a[i]);
    printf("\n\n");
}
```

Listing 7: main

5 Time Complexity Calculation

Time complexity has the recursive relation:

$$T(n) = T(n * p) + T(n - n * p - 1) + \theta(1)$$

where p in a randomly generated number between 0.4 and 0.6 in each iteration.

Solving this equation by recursive method yields T(n) $\in O(n)$.

where n is number of nodes in Binary Tree.