S20190010034A3

C Bhavesh Kumar

October 2020

1 Storing hierarchial structure of company

We want to maintain the list of employees in a company. We will be concerned with two quantities associated with each employee in the company – name of the employee (you can assume no two employees in the company have the same name), and the level of the employee. The level denotes where the person stands in the hierarchy. Level 1 denotes the highest post in the company (say the CEO), level 2 comes below level 1 and so on. There is only 1 person at level 1, but there can be several employees at level i ¿ 1. Each level i employee works under a level i-1 employee, which is his/her immediate boss. Given an employee A, we can form a sequence of employees A', A", A"', ... where A works under A', A' works under A", and so on. We say that each employee in A', A", A"',... is a boss of A.

(1) Data structure used for implementing the hierarchy structure.

```
    typedef struct node{
    char name[101];
    int level;
    struct node *parent;  //pointer to the boss
    struct node *list;    //pointer for the list of employees
    working directly under the employee
    struct node *next;    //pointer for coworkers
    struct node *prev;    //pointer for coworkers
}node;

```

Search used:

```
    node *search(node *root,char *name){
    if(root!=NULL){
        if(!strcmp(root->name,name)) return root;
        node *temp1=search(root->list ,name);
        if(temp1!=NULL) return temp1;
        node *temp2=search(root->next,name);
        return temp2;
    }
}

```

(2) Add employee(S,S')
We want to add a new employee with name S under employee S'.Level of S is one more than S'.
Code used is:

```
    int insert(node *root,char *name1,char *name2){
    node *insertplace=search(root,name2);
    if(insertplace==NULL)
        return 0;
    node *insertnode=newnode(name1);
    insertnode->level=insertplace->level+1;
    insertnode->parent=insertplace;
    if(insertplace->list==NULL){
        insertplace->list=insertnode;
    }
    else{
        insertnode->next=insertplace->list;
        insertplace->list->prev=insertnode;
        insertplace->list=insertnode;
    }
    return 1;
}

```

Duplicates and insertion failure written as a part of main program.

```
        if(!strcmp(name1,name2)){
                    print("Error:Both names are same");
                    break;
                }
                int success=insert(boss,name1,name2);
                if(success){
                    print("Employee successfully Added!");
                }
                else{
                    print("Could not find the boss with given
    name,please try again");
                }

```

The time complexity of the operation add employee is O(n),where n is the total number of employees in the company.
This is because the operation involves search, which searches the boss under whom S will work,this can be done in O(n) time and after finding we can insert in O(1) time as we use the linked list implementation.
Time complexity:O(n)

(3) Delete Employee(S,S')
Here we delete employee with name S and give the responsibilities of S to S'.
All the errors that might arise have been taken care of and duplicacy and asking to delete ceo has been included as a part of main program.
Code used:

```
    void delete(node *root,char *name1,char *name2){
```

```
    node *tobedeleted=search(root,name1);
    if(tobedeleted==NULL){
        printf("Employee to be deleted not found\n");
        return;
    }
    if(tobedeleted->prev==NULL && tobedeleted->next==NULL){
        printf("The given employee cannot be deleted as there is no one to take his responsibility\n");
        return;
    }
    node *copynode=searchlist(tobedeleted->parent->list,name2);
    if(copynode==NULL){
        printf("Employee who should have employees working under the deleted employee not found\n");
        printf("The two employees may not be on same level or has different bosses\nPlease try again\n");
        return;
    }
    if(copynode->level!=tobedeleted->level){
        printf("Given employees are in different levels\n");
        return;
    }
    //Copying data before delete
    node *temp=tobedeleted->list;
    node *t=copynode->list;
    if(copynode->list==NULL){
        copynode->list=temp;
    }
    else{
        while(t->next!=NULL) t=t->next;
        if(temp!=NULL){
            t->next=temp;
            temp->prev=t;
        }
    }
    while(temp!=NULL){
        temp->parent=copynode;
        temp=temp->next;
    }
    tobedeleted->list=NULL;
    /*Deleting*/
    if(tobedeleted->prev==NULL){
        tobedeleted->parent->list=tobedeleted->next;
        tobedeleted->next->prev=NULL;
        tobedeleted->next=NULL;
        free(tobedeleted);
    }
    else if(tobedeleted->next==NULL){
        tobedeleted->prev->next=NULL;
        tobedeleted->prev=NULL;
        free(tobedeleted);
    }
    else{
        node *t=tobedeleted->prev;
        t->next=tobedeleted->next;
        tobedeleted->next->prev=t;
```

```
        free(tobedeleted);
    }
    print("Successfully Deleted");
}

```

The time complexity of the above code is also O(n).We find the required employees in O(n) using search operation and perform delete in O(1) runtime.
Observe that the search used for S' is different from S.As S' is on same level of S and has same boss we can traverrse through the list of employees working under S's boss to find S' which is alot better than to traverse whole tree again to S'.
Search used for S':

```
node* searchlist(node *root,char *name){
    if(root!=NULL){
        if(!strcmp(root->name,name)){
            return root;
        }
        return searchlist(root->next,name);
    }
    return NULL;
}

```

Time complexity:O(n)

(4) Lowest common boss(S,S'):
Here S and S' are the names of employees.And we need to find the lowest common boss of these two employees in the hierarchy.
Code used:

```
node* lowestcommonboss(node *root,char *name1,char *name2){
    node *t1=search(root,name1);
    node *t2=search(root,name2);
    if(t1==NULL){
        printf("%s not found\n",name1);
        return NULL;
    }
    if(t2==NULL){
        printf("%s not found\n",name2);
        return NULL;
    }
    if(t1->level==1 || t2->level==1){
        printf("One of the given names is boss or ceo\n");
        return NULL;
    }
    while(t1->level!=t2->level){
        if(t1->level>t2->level){
            t1=t1->parent;
        }
        else if(t1->level<t2->level){
            t2=t2->parent;
        }
```

```
    }
    while(strcmp(t1->parent->name,t2->parent->name)){
        t1=t1->parent;
        t2=t2->parent;
    }
    return t1->parent;
}

```

The time complexity of the above code is atmost the height of the tree.
Which is nothing but the number of levels of the hierarchy.
This is because we climb each level up the tree from the employees given using the parent pointer and hence we can find the common boss in O(H),where H is height of tree.
First we get to the same level from both employees and compare the names of bosses at the level,if they are same we return the boss as lowest common boss or else we continue climbing up the tree and comparing until we find.
Time complexity=O(number of levels)

(5) print employee
Here we are supposed to print the employees in the hierarchy,levelwise.
We do this by using level order traversal.
This is similar to Breadth First search Algorithm(BFS) and hence follows the same time complexity.
Code used:

```
  void print_employee(node *root){
    push(root);
    int level=0;
    while(!isempty()){
        node *t=pop();
        if(level!=t->level){
            level=t->level;
            printf("\nLevel %d: %s",level,t->name);
        }
        else
        printf(", %s",t->name);
        node *temp=t->list;
        while(temp!=NULL){
            push(temp);
            temp=temp->next;
        }
    }
    printf("\n");
}

```

Queue used is a globally defined queue(head and tail pointers) and the code for queue is written in the code file.
The time complexity of the operation is O(n).This is because we visit each employee exactly once and traverse whole tree.
This can also be obtained from BFS where time complexity is $O(n+e)$,where e is number of edges, in trees $e = n-1$,Therefore O(n+n-1)==>

$O(n)$
Time complexity=O(n)