**V.Venkateswarlu Gupta**

**1602-19-733-119**

**CSE-B**

**Data Structures**

Assignment – 3

1.You have been hired by the government for their latest project. The country consists of N cities and M roads. Roads are directional. There may be more than one roads between some cities. The cities are numbered from 1 to N.You are given Q queries, each query has two integers U and V. Your task is to print the shortest path between city U and city V. If it is not possible to go from city U to city V, then print -1. InputFirst line consists of three integers N, M and Q.Next M lines each consists of three integers u, v, and w denoting a bidirectional road between city u and city v with length w.Then Q lines follow, each line having two integers U and V.OutputPrint Q lines of output. For each query print the length of the shortest path between U and VNotes1 ≤ N ≤ 200 1 ≤ M, Q, ≤ N × N

#define INF 987654

#include<stdio.h>

#include<stdlib.h>

*struct* node{

*int* to;

*int* weight;

*struct* node \*next;

};

*struct* graph{

*int* noOfVertices;

*struct* node \*\*lists;

};

*struct* node \*createNode(*int* *t*,*int* *w*){

*struct* node \*new = (*struct* node \*)malloc(sizeof(*struct* node));

    new->to = *t*;

    new->weight=*w*;

    new->next = NULL;

    return new;

}

*struct* graph \*createGraph(*int* *vertices*){

*struct* graph \*g = (*struct* graph \*)malloc(sizeof(*struct* graph));

    g->noOfVertices = *vertices*;

    g->lists = (*struct* node \*\*)malloc(sizeof(*struct* node \*) \* *vertices*);

    while (*vertices* > 0){

        g->lists[--*vertices*] = NULL;

    }

    return g;

}

*void* addEdgeForward(*struct* graph \**g*, *int* *from*, *int* *to*,*int* *w*){

*struct* node \*new = createNode(*to*,*w*);

    new->next = *g*->lists[*from*];

*g*->lists[*from*] = new;

}

*void* addEdgeBackward(*struct* graph \**g*, *int* *from*, *int* *to*,*int* *w*){

*struct* node \*new = createNode(*from*, *w*);

    new->next = *g*->lists[*to*];

*g*->lists[*to*] = new;

}

*void* addEdge(*struct* graph \**g*, *int* *from*, *int* *to*, *int* *isUndirected*,*int* *w*){

    addEdgeForward(*g*, *from*, *to*,*w*);

    if (*isUndirected*){

        addEdgeBackward(*g*, *from*, *to*,*w*);

    }

}

*void* print(*struct* graph \**g*)                                 {

    for (*int* v = 0; v < *g*->noOfVertices; v++)               {

*struct* node \*t = *g*->lists[v]                        ;

        printf("\n Vertex %d : ", v)                        ;

        while (t)                                           {

            printf("%d,%d -> ", t->to,t->weight)            ;

        t = t->next                                         ;

                                                            }

                                                            }

                                                            }

*void* dijkstrallsd(*struct* graph \**g*,*int* *n*, *int* *source*, *int* *d*) {

    //this function is dijkstra using List Source Destination

*int* i,j,max=0;

*int* \*visited=(*int* \*)calloc(*n*,sizeof(*int*));

*int* \*distance=(*int* \*)calloc(*n*,sizeof(*int*));

    for(i=0;i<*n*;i++){

        distance[i]=INF;

    }

    distance[*source*]=0;

    for(i=0;i<*n*;i++){

*int* uu=INF,u=-1;

        for(j=0;j<*n*;j++){

            if(uu>=distance[j]&&visited[j]==0){

                uu=distance[j];

                u=j;

            }

        }

        visited[u]=1;

*struct* node \*t = *g*->lists[u];

        while(t){

            max+=t->weight;

            if(visited[t->to]==0){

                if(distance[t->to]>distance[u]+t->weight){

                    distance[t->to]=distance[u]+t->weight;

                }

            }

            t=t->next;

        }

    }

    if(distance[*d*]>max){

        printf("-1\n");

    }else{

        printf("%d\n",distance[*d*]);

    }

}

*void* main(){

*int* i,n,m,q,u,v,w,s,d;

    FILE \*fp;

    fp=fopen("graphInput.txt","r");

    fscanf(fp,"%d %d %d",&n,&m,&q);

*struct* graph\* g=createGraph(n);

    //function signature

    //addEdge(struct graph \*g, int from, int to, int isUndirected,int weight)

    for(i=0;i<m;i++){

        fscanf(fp,"%d %d %d",&u,&v,&w);

        addEdge(g,u-1,v-1,0,w);

    }

    //print(g);

    while(q){

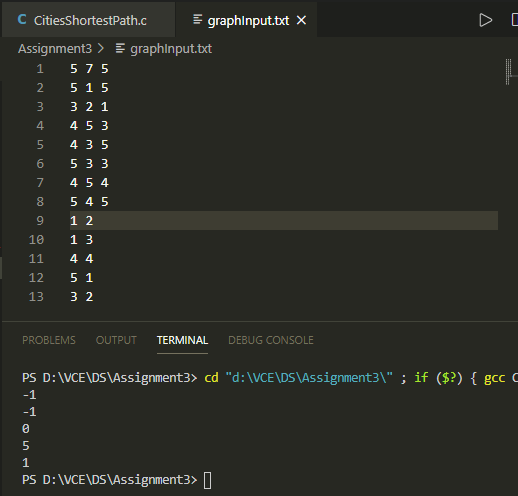
        fscanf(fp,"%d%d",&s,&d);

        dijkstrallsd(g,n,s-1,d-1);

        q--;

    }

}



2. Akshay is a new to programming and while learning the programming language he came to know the following rules:- Each program may start with '{' and end with '}'.- Each program must contain only one main function. Main function must start with '<' and end with '>'.- A program may or may not contain user defined function(s). There is no limitation on the number of user defined functions in the program. User defined function must start with '(' and end with ')'.- Loops are allowed only inside the functions (this function can be either main function or user defined function(s)). Every loop must startwith '{' and end with '}'.- User defined function(s) are not allowed to be defined inside main function or other user defined function(s).- Nested loops are allowed. - Instructions can be anywhere inside the program.- Number of instructions inside any user defined function must not be more than 100. If any of the above conditions is not satisfied, then the program will generate compilation errors. Today Akshay has written a few programs,but he is not sure about the correctness of the programs. Your task is to help him to find out whether his program will compile without any errors or not. Input Format:First line starts with T, number of test cases. Each test case will contain a single line L, where L is a program written by Akshay. Output Format:Print "No Compilation Errors" if there are no compilation errors, else print "Compilation Errors".Constraints:1<=T<=100L is a text and can be composed of any of the characters {, }, (, ), <, >and P, where P will represents the instruction. L, comprised of characters mentioned above should be single spaced delimited. Number of characters in the text, |L| < = 10000

#include <stdio.h>

#include<string.h>

#include<ctype.h>

*char* arr[50];

*int* top=-1;

*void* push(*char* *ch*){

    arr[top++]=*ch*;

}

*char* pop(){

    return arr[--top];

}

*int* check(*char* *c1*,*char* *c2*){

*char* c3;

    if(*c1*=='<'&&*c2*=='>')

    return 1;

    else if(*c1*=='('&&*c2*==')')

    return 1;

    else if(*c1*=='{'){

        if(*c2*!='}')

        return 0;

        else if(top!=-1){

        c3=pop();

        push(c3);

        if(c3!='{'||c3!='('||c3!='<'){

        return 0;

        }

        }

        else if(top==-1)

        return 0;

    }

    else if(*c1*=='{'&&*c2*=='}')

    return 1;

    else

    return 0;

}

*int* main()

{

*char* str[50],ch;

*int* t,i,count1,count2;

    scanf("%d",&t);

    for(i=0;i<t;i++){

        scanf("%s",str);

*int* j,flag=0;

        count1=0,count2=0;

        if(str[0]!='{'||str[strlen(str)-1]!='}'){

                printf("Compilation Errors\n");

        }

        else{

        j=0;

        while(j<strlen(str)){

            if(str[j]=='<')

            count1++;

            if(str[j]=='>')

            count2++;

            j++;

        }

        if(count1!=1||count2!=1)

        printf("Compilation Errors\n");

        else{

            j=1;

        while(j<strlen(str)-1){

            if(str[j]=='<'){

                if(str[j+1]=='('||str[j+1]==')'){

                printf("Complilation Errors\n");

                break;

                }

                else

                push(str[j]);

            }

            else if(str[j]=='('){

                if(str[j+1]=='<'||str[j+1]=='>'){

                    printf("Compilation Errors\n");

                    break;

                }

                else

                push(str[j]);

            }

            else if(str[j]=='{'){

                if(str[j+1]=='('||str[j+1]==')'||str[j+1]=='>'||str[j+1]=='<'){

                    printf("Compilation Errors\n");

                    flag=0;

                    break;

                }

                else

                push(str[j]);

            }

            else if(str[j]=='>'||str[j]==')'||str[j]=='}'){

                ch=pop();

                if(check(ch,str[j])==0){

                    printf("Compilation Errors\n");

                    flag=0;

                    break;

                }

                else if(check(ch,str[j])==1){

                    flag=1;

                }

            }

            else if(str[j]!='P'){

                printf("Compilation Errors\n");

                flag=0;

                break;

            }

            j++;

        }

        if(flag==1)

        printf("No compilation errors\n");

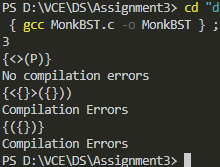
        }

        }

    }

    return 0;

}

3.Around 25 teams participated in a paper presentation contest. The Evaluators have awarded marks to teams based on 3 criteria - depth of content, presentation and communications skills. Each criteria allotted with 10 marks each. Display the evaluation list in the decreasing order of their total marks in all the 3 criteria. If the teams have same total mark then which team has registered first will be placed first than the other one. Accept the input details as described in the scenario and design a C program to display the evaluation list. Use Quick Sort

#include<stdio.h>

#include<stdlib.h>

typedef *struct* skillset{

*int* content,presentation,communication,total;

}skill;

typedef *struct* list{

    skill s;

*int* teamnum;

*struct* list \*next;

}node;

 node\* q;

node\* createll(node \**head*,*int* *team*){

    node \*temp=(node \*)malloc(sizeof(node));

    scanf("%d%d%d",&temp->s.content,&temp->s.presentation,&temp->s.communication);

    temp->s.total=temp->s.content+temp->s.presentation+temp->s.communication;

    temp->next=NULL;

    temp->teamnum=team;

    if(head==NULL){

        head=temp;

        q=temp;

    }

    else{

        q->next=temp;

        q=temp;

    }

    return head;

}

*void* display(node \**head*){

    if(head==NULL){

        printf("No teams participated\n");

    }

    else{

        node \*q=head;

*int* i=1;

        while(q!=NULL){

            printf("Team %d scores: %d %d %d Total: %d\n",q->teamnum,q->s.content,q->s.presentation,q->s.communication,q->s.total);

            i++;

            q=q->next;

        }

    }

}

node \*gettail(node \**q*){

   while(q!=NULL&&q->next!=NULL){

       q=q->next;

   }

   return q;

}

node\* partition(node \**head*,node \**end*,node \*\**newhead*,node \*\**newend*){

    node \*pivot=end;

    node \*prev=NULL, \*cur=head,\*tail=pivot;

    while(cur!=pivot){

        if(cur->s.total>=pivot->s.total){

            if((\*newhead)==NULL){

                (\*newhead)=cur;

            }

            prev=cur;

            cur=cur->next;

        }

        else{

            if(prev){

                prev->next=cur->next;}

                node \*temp=cur->next;

                cur->next=NULL;

                tail->next=cur;

                tail=cur;

                cur=temp;

            }

    }

    if(\*newhead==NULL){

        \*newhead=pivot;

    }

    \*newend=tail;

    return pivot;

}

node \*quicksortrecur(node \**head*,node \**end*){

    if(!head||head==end)return head;

    node \*newhead=NULL,\*newend=NULL;

    node \*pivot=partition(head,end,&newhead,&newend);

    if(newhead!=pivot){

        node \*temp=newhead;

        while(temp->next!=pivot){

            temp=temp->next;

        }

        temp->next=NULL;

        newhead=quicksortrecur(newhead,temp);

        temp=gettail(newhead);

        temp->next=pivot;

    }

    pivot->next=quicksortrecur(pivot->next,newend);

    return newhead;

}

node\* quicksort(node \*\**head*){

    \*head=quicksortrecur(\*head,gettail(\*head));

    return \*head;

}

*int* main(){

    node\* head=NULL;

    for(*int* i=0;i<3;i++){

        printf("Enter the scores of team %d  in 3 respective fields out of 10\n",i+1);

       head= createll(head,i+1);

    }

    display(head);

   head= quicksort(&head);

   printf("List of teams after sorting based of total scores\n");

    display(head);

}

4. Monk is standing at the door of his classroom. There are currently N students in the class, i 'th student got Ai candies.There are still M more students to come. At every instant, a student enters the class and wishes to be seated with a student who has exactlythe same number of candies. For each student, Monk shouts YES if such a student is found, NO otherwise. (Use BST)Input:First line contains an integer T . T test cases follow.First line of each case contains two space-separated integers N and M. Second line contains N + M space-separated integers, the candies of the students.Output:For each test case, output Mnew line, Monk's answer to the M students. Print "YES" (without the quotes) or "NO" (without the quotes) pertaining to the Monk's answer.Constraints:1 ≤ T ≤ 101 ≤ N , M≤ 1050 ≤ Ai≤ 1012

#include<stdio.h>

#include<stdlib.h>

typedef *struct* node{

*int* data;

*struct* node \*leftChild,\*rightChild;

}BST;

BST \*createNode(*int* *value*){

    BST \*newNode;

    newNode=(BST \*)malloc(sizeof(BST));

    newNode->leftChild=NULL;

    newNode->data=*value*;

    newNode->rightChild=NULL;

    return newNode;

}

BST \*insertNode(BST \**root*, *int* *data*){

    BST \*parent;

    BST \*current;

    if(*root*==NULL){

*root*=createNode(*data*);

    }else{

        current=*root*;

        while(1){

            parent=current;

            if(*data*<current->data){

                current=current->leftChild;

                if(current==NULL){

                    parent->leftChild=createNode(*data*);

                    break;

                }

            }else{

                current=current->rightChild;

                if(current==NULL){

                    parent->rightChild=createNode(*data*);

                    break;

                }

            }

        }

    }

    return *root*;

}

BST \*createBST(BST \**root*,*int* *n*){

*int* i,data;

    for(i=0;i<*n*;i++){

        scanf("%d",&data);

*root*=insertNode(*root*, data);

    }

    return *root*;

}

*void* monkShouting(BST \**root*, *int* *key*){

    BST \*cur=*root*;

    while(1){

        if(cur->data==*key*){

            printf("YES\n");

            break;

        }else if(cur->data>*key*){

            cur=cur->leftChild;

            if(cur==NULL){

                printf("NO\n");

                break;

            }

        }else{

            cur=cur->rightChild;

            if(cur==NULL){

                printf("NO\n");

                break;

            }

        }

    }

}

*void* main(){

*int* t;

    scanf("%d",&t);

    while(t--){

*int* n,m,enteredStudent;

        BST \*root=NULL;

        scanf("%d%d",&n,&m);

        root=createBST(root, n);

        while(m--){

            scanf("%d",&enteredStudent);

            monkShouting(root,enteredStudent);

        }

    }

}

