FIRST SEMESTER MCA (2020 SCHEME)
PRACTICAL EXAMINATION JUNE-JULY 2021

Swetha Sadanandan
ICG 20MCA-2039

20MCA135 DATA STRUCTURES LAB

Date - 30.06.2021 Batch - III

1) Implement linked stack

```
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
#include<limits.h>
#define CAPACITY 1000
struct stack
{
  int data;
struct stack *next;
}*top;
int size=0;
void push(int element);
int pop();
void main()
{
int choice, data;
while()
{
printf("-------\n");
printf("stack implementation program\n");
printf("1. push\n");
printf("2. pop\n");
printf("3. size\n");
printf("4. exit\n");
printf("enter your choice\n");
scanf("%d", &choice);
switch(choice)
{
case1: printf("enter data to push into stack\n");
scanf("%d", &data);
push(data);
break;
```

```
case 2 :
data = pop();
if (data != INT_MIN)
printf("Data => %d \n", data);
break ;
case 3 :
printf ("stack size : %d\n", size) ;
break ;
case 4 :
printf ("exiting\n");
break ;
default :
printf("invalid choice , please try again\n");
}printf("\n\n");
}
void push (int element)
{
struct stack *newNode = (struct stack*)malloc(sizeof(struct stac
if(size >= CAPACITY)
{
printf("stack overflow\n");
return ;
}
newNode -> data = element ;
newNode -> next = top ;
top = newNode ;
size++;
printf("data pushed into stack");
}
int pop()
{
int data = 0;
struct stack *topNodes;
if (size <= 0 || !top)
{
printf ("stack is empty");
return INT_MIN ;
}
topNode = top ;
data = top -> data ;
top = top -> next ;
free (topNode);
```

```
size--;
return data;
}
```

Output

1. Push
2. pop
3. Size
4. Exit

Enter choice: 1
Enter data to push into stack: 5

2) Implement Kruskal's algorithm

```
#include<stdio.h>
#include<conio.h>
#define MAX 30
typedef struct edge
{
int u,v,w;
}
edge;
typedef struct edge_list
{
edge data[MAX];
int n;
}
edge_list;
edge_list elist;
int Graph[MAX][MAX],n;
edge_list spanlist;
void kruskalAlgo();
int find(int belongs[], int vertexno);
void applyUnion(int belongs[], int c1, int c2);
void sort();
void print();
void kruskalAlgo()
{
int belongs[MAX], i, j, cno1, cno2;
elist.n=0;
printf("element of graph are");
for(i=1; i<n; i++)
for(j=0; j<i; j++)
{
if(Graph[i][j]!=0)
{
```

```
elist.data[elist.n].u=i;
elist.data[elist.n].v=j;
elist.data[elist.n].w = Graph[i][j];
elist.n++;
}
}
sort();
for(i=0;i<n;i++)
belongs[i]=i;
spanlist.n=0;
for(i=0;i<elist.n;i++)
{
cno1=find(belongs,elist.data[i].u);
cno2=find[belongs,elist.data[i].v);
if(cno1!=cno2)
{
spanlist.data[spanlist.n]=elist.data[i];
spanlist.n=spanlist.n+1;
applyUnion(belongs,cno1,cno2);
}
}
}
int find(int belongs[], int vertexno)
{
return(belongs[vertexno]);
}
void applyUnion(int belongs[],int c1,int c2)
{
int i;
for(i=0;i<n;i++)
if(belongs[i]==c2)
belongs[i]=c1;
}
void sort()
{
int i,j;
edge temp;
```

```
Graph [2] [1] = 2 ;
Graph [2] [2] = 0 ;
Graph [2] [3] = 3 ;
Graph [2] [4] = 4 ;
Graph [2] [5] = 0 ;
Graph [2] [6] = 0 ;
Graph [3] [0] = 0 ;
Graph [3] [1] = 0 ;
Graph [3] [2] = 3 ;
Graph [3] [3] = 0 ;
Graph [3] [4] = 3 ;
Graph [3] [5] = 0 ;
Graph [3] [6] = 0 ;
Graph [4] [0] = 0 ;
Graph [4] [1] = 0 ;
Graph [4] [2] = 0 ;
Graph [4] [3] = 4 ;
Graph [4] [4] = 3 ;
Graph [4] [5] = 0 ;
Graph [4] [6] = 0;
Graph [5] [0] = 0;
Graph [5] [1] = 0;
Graph [5] [2] = 2;
Graph [5] [3] = 0;
Graph [5] [4] = 3;
Graph [5] [5] = 0;
Graph [5] [6] = 0;
[illegible]
[illegible]
[illegible]
[illegible]
[illegible]
[illegible]
[illegible]
KruskalAlgo();
print();
getch();
}
```

Output

```
2 1 2
5 2 2
3 2 3
4 3 3
1 0 4
```

Spanning tree cost 14.