FIRST SEMESTER (2020 SCHEME)

PRACTICAL EXAMINATION JUNE - JULY 2021

20MCA135 DATA STRUCTURES LAB

THRESIAMMA MATHEW
ICE20MCA - 2042
Date - 30 - June - 2021

Time : 1 pm to

BATCH - 3

1) To implement linked stack:

Program

```
#include <stdio.h>
#include <conio.h>
#include <stdlib.h>
#include<limits.h>
#define CAPACITY 10000
struct stack
{
int data;
struct stack *next;
}
*top;
int size=0
void push(int element);
```

```
int pop();
void main()
{
int choice, data;
clrscr();
printf("----------\n");
printf("LINKED STACK IMPLEMENTATION PROGRAM\n");
printf("----------\n");
printf("1. push\n");
printf("2. pop\n");
printf("3. size\n");
printf("4. Exit\n");
printf("----------\n");
while(1)
{
printf("Enter your choice:");
scanf("%d", &choice);
switch(choice)
{
case 1:
        printf("Enter data to push into stack:");
        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 from app\n");
        exit(0);
        break;

default:
        printf("invalid choice, please try again\n");
 }

printf("\n\n");
getch();
  }
}

void push(int element)
{
struct Stack * newNode;
if(size >= CAPACITY)
{
printf("Stack overflow, can't add more elements
```

```
return;
}
newNode = (struct stack*) malloc(sizeof(struct stack));
newNode->data = element;
newNode->next = top;
top = newNode;
size++;
printf("Data pushed to stack\n");
}
int pop()
{
int data=0;
struct stack *topNode;
if(size<=0||!top)
{
printf("stack is empty\n");
return INT_MIN;
}
top node = top;
data = top->data;
top = top->next;
free(topnode);
size--;
return data;
}
```

Output

STACK IMPLEMENTATION PROGRAM

1. push
2. pop
3. size
4. Exit

Enter your choice: 1
Enter data to push into stack: 10
Data pushed to stack.

2) To implement Kruskal's Algorithm

```
#include <stdio.h>
#include <conio.h>
# define MAX 30
typedef struct edge
{
    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("Elements of the graph are\n");
    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;
}
int
{
}
void
{
}
```

```
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;
for(i=1; i<elist.n; i++)
        for(j=0; j<elist.n-1; j++)
        if(elist.data[j].w > elist.data[j+1].w)
          {
             temp = elist.data[j];
             elist.data[j] = elist.data[j+1];
             elist.data[j+1] = temp;
        } }
void print()
   {
   int i, cost = 0;
   for(i=0; i<spanlist.n; i++)
          {
          printf("\n%d-%d : %d", spanlist.data[i].u
          spanlist.data[i].v, spanlist.data[i].w);
          cost = cost + spanlist.data[i].w;
          }
     printf("\nSpanning tree cost : %d", cost);
   }
```

```
void main()
{
  int i, j, total_cost;
  clrscr();
  n=6;
Graph[0][0]=0;
Graph[0][1]=4;
Graph[0][2]=4;
Graph[0][3]=0;
Graph[0][4]=0;
Graph[0][5]=0;
Graph[0][6]=0;

Graph[1][0]=4;
Graph[1][2]=0;
Graph[1][3]=2;
Graph[1][4]=0;
Graph[1][5]=0;
Graph[1][6]=0;

Graph[2][0]=4;
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] = 4;
Graph[4][3] = 3;
Graph[4][4] = 0;
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;

KruskalAlgo();
print();
getch();
}
```

output

elements of the graph are

2-1 : 2
5-2 : 2
3-2 : 3
4-3 : 3
1-0 : 4

Spanning tree cost : 14