First semester MCA (2020 scheme)
Practical examination June-July 2021

Date: 30/06/2021 Time: 9-30-12-30. Page: 1

Ashitha Rasheed
ICE20MCA-2018

Batch - C

1. Implement linked stack?
2. Implement Kruskal's algorithm?.

1. Implement linked stacks?

step 1 : start.
step 2 : Create a node new and declare variable top.
step 3 : set new data part to be Null
step 4 : Read the node to be inserted.
step 5 : Check if the node is null, then print "overflow".
step 6 : If node is not null, assign the item to new data part of new and assign top to link.
step 7 : Check if the top is null, then print "stack underflow"
step 8 : If top is not null, assign the top's link part to ptr and assign ptr to stack.
step 9 : stop.

Program

```
#include <stdio.h>
#include <conio.h>
#include <stdlib.h>
#include <limits.h>
#define CAPACITY 10000
struct stack
{
 int data;
struct stack *next;
}
*top;
// stack size
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 element to stack.\n");
return;}
}
New node = (struct stack *) malloc (size of (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;
}
topNode=top;
data=top->data;
top=top->next;
free(top Node);
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.

Stack Implementation program.

1. Push.
2. POP
3. size
4. exit

enter your choice : 1

enter data to push into stack : 20

data pushed to stack.

Stack Implementation program.

1. Push
2. POP
3. size
4. exit

Enter your choice : 1

enter data to push into stack : 30

data pushed to stack.

Stack implementation program.

1. Push
2. POP
3. size
4. exit.

enter your choice : 2

Data : => 30.

Stack implementation program.

1. Push.
2. POP.

3. size
4. exit

enter your choice : 3

Stack size : 2.

Stack implementation program.

1. push
2. Pop
3. size
4. exit

enter your choice : 4.

2. Implement Kruskal's algorithm ?

Algorithm

step1 : start with a weighted graph.

step2 : Choose the edge with the least weight, if there are more than 1, choose anyone.

step3 : Choose the next shortest edge and add it.

step4 : Choose the next shortest edge that doesn't create a cycle and add it.

step5 : Repeat until you have a spanning tree.

Program.

```
#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 vertexc no);
void apply union(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;
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++)
If (elist.data[j].w>elist.data[j+1].w]
{
 temp=elist.data[j];
 elist.data[j]=elist.data[j+1];
 elist.dat[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("\n Spanning 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][1] = 0;
Graph[1][2] = 2;
Graph[1][3] = 0;
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;

Kruskal Algo();
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.