BATCH III

FIRST SEMESTER (2020 SCHEME)

PRACTICAL EXAMINATION JUNE-JULY 2021

20MCA135 DATA STRUCTURES LAB

TIME: 9:30 AM - 12:30 PM

Date : 30.06.2021

NAME: ARYA PRADEEP

REG.NO : ICE20MCA 2017

1. Implement Linked Stack

Algorithm

Push operation

Step 1 :- Start.

Step 2 : Create a new node and declare the variable at the top of the stack.

Step 3 : Set the new data part to be null

Step 4 : insert the node.

Step 5 : If node = NULL
then print ("insufficient Memory");

Step 6 : If node ≠ NULL
Assign the value to the data part.
and assign to top of the ~~first link~~ stack.

Pop operation

Step 1 : Start

Step 2 : If top = NULL then,
print "Stack underflow";

Step 3 :. If top ≠ NULL, then
create a temporary node. and ~~set~~ pop it to the Top.

Step 4 : print the data of Top.

Step 5 : Top to point to the next node. and delete the Temporary node

Step 6 : stop.

Output

Stack implementation program.

1. push
2. posp
3. size
4. exist

Enter your choice : 1

enter data to push into stack
15
data pushed into stack.

Stack implementation program.

1. Push
2. Pop
3. Size
4. exist

Enter your choice. 1
Enter data to push into stack

11

data ~~ph~~ pushed into stack.

Stack implementation pgm.

1. push
2. pop
3. Size
4. exist

enter your choice. 3
Stack size 2.

stack implementation Program.

1. Push
2. Pop
3. Size
4 exist

enter your choice 2
Data => 3.

Stack implementation ~~of~~ program.

1. Push.
2. PoP
3 Size
4 exist
enter your choice 4.

Program

```
#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 (1)
{
Printf ("STACK IMPLEMENTATION");
Printf ("1. push");
~~Printf ("2. pb~~
Printf ("2 stack
printf (" 2. pop");
Printf ("3. size");
printf ("4. exit");
Printf ("enter your choice");
```

```
~~Sca~~
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 ", size);
break;          -> case 3:
                   printf("stack size %d"), size);
                   break;
case 4:
printf ("exiting");
break;
~~default;~~
default :
printf(" Invalid choice, please try ");
}
printf ("\n \n");
}
}
void push (int element)
{
struct stack * newNode = (struct stack * ) malloc (size of
                                              (struct stack));
```

```
if (size >= CAPACITY).
{
printf (" stack overflow");
return;
}
new node => data = element
new node => next = top;
top = newNode;
size ++;
printf ("data pushed into stack");
}
int pop ()
{
int data = 0;
struct stack * top node;
if (size <= 0 || ! top)
{
printf ( " empty stack ");
return main;
}
top Node = top;
data = top -> data;
top = top => next;
free (topnode);
size --;
return data;
}
```

Implementation of Kruskal Algorithm.

2. Algorithm

step 1: Start

step 2: All the edges in decreasing order of the weight.

step 3: ~~take~~ the smallest edge and ~~check~~ ~~the~~ check if forms a cycle with spanning tree. formed so far. if cycle is not formed.

then include this edge

Else

Discard.

step 3: Repeat step (2) until there are (v-1) edges in the spanning tree.

step 4: ends.

Output

Elements of graph are

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

spanning tree cost 14.

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 vertexno);
```

~~void applyunion (int belongs [], int vertexno);~~

~~void applyUnion(int~~

```
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 \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[e]~~
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 = spanslist.n + 1;
apply Union (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 ("%d %d", spanlist.data[i].u, spanlist.data[i].v,
        spanlist.data[i].w);
cost = cost + spanlist.data[i].w;
}
printf (" spanning tree cost %d", cost);
}
void main()
{
int i, j, total cost;
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 ();
}
```