①

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

20MCA135 DATA STRUCTURES LAB

(Batch 3)

Aparnna Bimal
ICE20MCA-2016

Date : 30 June 2021

Time : 9:30-12:30

1. Implement Linked Stack

Algorithm

Push() operation

Step 1 : start
~~Step 2 : Set New data part to be null~~
Step 2 : ~~Set new data to~~ Create a node and declare variable
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, the point
Step 6: If node is not null, assign the item to data part of new and assign top to link part of new and also point stack head to new.

pop() · Operation

Step 1 : Start.
Step 2 : check if the top is NULL, then point "stack underflow";
Step 3: If top is not null, assign the top's link part to pointer to stack_head's link part
Step 4 : Stop.

Size() Operation / Peek() operation

Step 1 : start
Step 2 : point or store the node pointed by top variable
Step 3 : stop

(2)

Program code:

```
#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;
clrscr();
while(1) {
Printf("\n----\n");
Printf("Stack Implimentation Program\n");
Printf("\n-----");
Printf("\n1. Push\n");
Printf("\n2. Pop\n");
Printf("\n3. size\n");
Printf("\n4. Exit\n");
Printf("\n-------");
Scanf("\nEnter Your choice :");
Scanf("%d", &choice);
Switch(choice) {

case 1:
  Printf("\nEnter data to push into stack:");
  Scanf("%d", &data);
  push(data);
  break;
```

③

```
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");
    break;
default:
    printf("Invalid choice, Please try again, \n"); }
printf("\n\n");
 }}
Void push(int element) {
Struct stack *newnode = (struct stack *) malloc(size of(struct
                                                    stack));
if(size >= CAPACITY)
{ printf("Stack Overflow, cant add more element to stack, \n");
return; }
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");
```

④

~~Implement Linked Stack~~

~~Implement Linked Operation~~

```
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 your choice: 22
~~Enter~~ Data pushed to stack

Enter your choice: 1
Enter data to pushed into stack: 22
Data pushed to stack

Enter your choice: 21
Enter data to pushed into stak: 21
Data pushed to stack.

Enter your choice: 1
Enter data to pushed into stack: 23
Data pushed to stack

Enter your choice: 3
stack size => 3

2. Impliment Kauskal's Algoorithm.

Algoorithm

Step 1: Sort all the edges in non-decreasing ordeo of their weight.

Step 2: Pick the smallest edge. Check if it forms a cycle with the spanning toee formed so faa. If cycle is not formed, include this edge. Else, discard it.

Step 3: Repeat Step ② until there are (V-1) edges in the spanning toee.

Programme code:

```
#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 kauskalAlgo();
int findC(int belongs[], int vertexno);
void applyunion(int belongs[], int c1, int c2);
```

(6)

```
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=0; i<n; i++)
for(j=0; j<i; j++)
{
if (Graph[i][j]!=0)
{
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++)
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].v, 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;

7

```
Graph[4][5] = 0;
Graph[5][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 of the graph are

2-1: 2
5-2: 2
3-2: 3
4-3: 3
1-0: 4
Spanningtree cost: 14