SSN COLLEGE OF ENGINEERING
(Autonomous)

DEPARTMENT OF CSE

UCS308 Data Structures Lab

Assignment 9

Dictionary Using AVL

Register Number : 185001131

Name : Sai Charan B

Class : CSE – B

Vertex.h

```
typedef struct
Vertex{
                    char v;
                    char adj[10];
                    unsigned visited:1;
                }Vertex;


                typedef Vertex Graph[30];
```

```
Vertex getVertex(){
	Vertex vertex;
	char adj[10];
	char v;
	vertex.visited = 0;

	printf("Enter the vertex: ");
	scanf("%c",&v);
	getchar();
	printf("Enter the verices adjacent to vertex:
");
	scanf("%[^\n]",adj);
	getchar();

	vertex.v = v;
	strcpy(vertex.adj,adj);

	return vertex;
}
```

Stack.h

```
typedef Vertex
Data;

typedef struct StackNode{
	Data d;
	struct StackNode * next;
}StackNode;

typedef StackNode* Stack;
```

```
Stack createEmptyStack(){
	return 0;
}

int isEmptyStack(Stack top){
	return top == 0;
}

void push(Stack * top,Data d){
	StackNode * tmp =
(StackNode*)malloc(sizeof(StackNode));
	tmp -> d = d;
	tmp -> next = 0;
	if(!isEmptyStack(*top))
		tmp -> next = (*top);
	(*top) = tmp;
}

Data pop(Stack * top){
	Data rval;
	strcpy(rval.adj,"");

	if(isEmptyStack(*top))
		return rval;

	rval = (*top) -> d;
	StackNode * tmp = *top;
	*top = (*top) -> next;
	free(tmp);
```

```
        return rval;
}

Data peek(Stack top){
        return top -> d;
}

void displayStack(Stack top){
        StackNode * tmp = top;
        if(isEmptyStack(top)){
                printf("Empty Stack!");
                return;
        }
        while(tmp){
                printf("%c ",tmp -> d.v);
                tmp = tmp -> next;
        }
        printf("\n");
}
```

Queue.h

```
typedef struct
QueueNode{
                Data d;
                struct QueueNode * next ;
        }QueueNode;
```

```
typedef QueueNode * Queue;


int isEmptyQ(Queue front,Queue rear){
	if(front == 0 || rear == 0)
		return 1;
	return 0;
}


void enqueue(Queue *front,Queue *rear,Data d){

	QueueNode * tmp =
(QueueNode*)malloc(sizeof(QueueNode));
	tmp -> d = d;
	tmp -> next = NULL;


	if(isEmptyQ(*front,*rear))
		(*front) = (*rear) = tmp;
	else{
		(*rear) -> next = tmp;
		(*rear) = tmp;
	}


}


Data dequeue(Queue * front,Queue * rear){
	Vertex rval;
	strcpy(rval.adj,"");
```

```
        QueueNode * tmp;


        if(isEmptyQ(*front,*rear)){
                printf("Queue Empty!\n");
                return rval;
        }


        rval = (*front) -> d;
        tmp = (*front);


        if( (*front) == (*rear) )
                (*front) = (*rear) = 0;
        else
                (*front) = (*front) -> next;



        free(tmp);
        return rval;
}
```

GraphTraversal.h

```
int find(Graph g,int
size,const char C){
                for(int i = 0 ; i < size ; i++)
                        if(g[i].v == C)//Match Found
                                return i;
                return -1;
        }
```

```
void DFS(Graph g,int size,Vertex start){
	if(strcmp(start.adj,"-") == 0)
		return;

	Stack s = createEmptyStack();
	char adj[10];
	push(&s,start);
	Vertex tmp;

	while(!isEmptyStack(s)){
		tmp = pop(&s);
		if(g[find(g,size,tmp.v)].visited == 0){
			g[find(g,size,tmp.v)].visited =
1;
			printf("%c\t",tmp.v);
		}
		else
		 continue;

		strcpy(adj,tmp.adj);

		for(int i = 0 ; adj[i] ; i++)
			if(adj[i] == ' ')
				continue;
			else

	push(&s,g[find(g,size,adj[i])]);
	}
}
```

```
void BFS(Graph g,int size,Vertex start){
	if(strcmp(start.adj,"-") == 0)
		return;

	char adj[10];
	Queue front = 0, rear = 0;
	enqueue(&front,&rear,start);
	Vertex temp;

	while(!isEmptyQ(front,rear)){
		temp = dequeue(&front,&rear);

		if(g[find(g,size,temp.v)].visited == 1)
			continue;

		printf("%c\t",temp.v);
		g[find(g,size,temp.v)].visited = 1;
		strcpy(adj,temp.adj);
		if(strcmp(temp.adj,"-") == 0)
			continue;
		for(int i = 0 ; adj[i] ; i++){
			if(adj[i] == ' ')
				continue;
			if(
g[find(g,size,adj[i])].visited == 1)
					continue;
			else{
				//printf("%c\t",adj[i]);

	enqueue(&front,&rear,g[find(g,size,adj[i])]);
			}
		}
```

```
                                    }
                                }
```

Main.c

```
#include
<stdlib.h>
                    #include <stdio.h>
                    #include <string.h>



                    #include "Vertex.h"
                    #include "Queue.h"
                    #include "Stack.h"
                    #include "GraphTraversal.h"


                    char *Strrev(char *str)
                    {
                         char *p1, *p2;


                         if (! str || ! *str)
                               return str;
                         for (p1 = str, p2 = str + strlen(str) - 1; p2 > p1; ++p1,
                    --p2)
                         {
                               *p1 ^= *p2;
                               *p2 ^= *p1;
                               *p1 ^= *p2;
                         }
                         return str;
                    }
```

```c
int main(void){
	Graph g1,g2;
	int size;
	Vertex start;

	printf("ENTER THE GRAPH:\n");
	printf("Enter the number of vertices: ");
	scanf("%d",&size);
	getchar();

	for(int i = 0 ; i < size; i++){
		g1[i] = getVertex();

		if(i == 0)
			start = g1[i];
	}

	printf("BFS OUTPUT:\n");
	BFS(g1,size,start);

	for(int i = 0 ; i < size ; i++){
		g1[i].visited = 0;
		Strrev(g1[i].adj);
	}
	start = g1[0];

	printf("\nDFS OUTPUT:\n");
	DFS(g1,size,start);
```

```
    return 0;
}
```

OUTPUT:

```
ENTER THE GRAPH:
Enter the number of vertices: 5
Enter the vertex: A
Enter the verices adjacent to vertex: B C E
Enter the vertex: B
Enter the verices adjacent to vertex: A D E
Enter the vertex: C
Enter the verices adjacent to vertex: A
Enter the vertex: D
Enter the verices adjacent to vertex: B
Enter the vertex: E
Enter the verices adjacent to vertex: A B

 ADJACENCY LIST
 +--------+----------------+
 | Vertex | Adjacenct List |
 +--------+----------------+
 |   A    | B C E          |
 |   B    | A D E          |
 |   C    | A              |
 |   D    | B              |
 |   E    | A B            |
 +--------+----------------+


BFS OUTPUT:
A   B   C   E   D
DFS OUTPUT:
A   B   D   E   C

ENTER THE GRAPH:
```

```
Enter the number of vertices: 5
Enter the vertex: 0
Enter the verices adjacent to vertex: 1
Enter the vertex: 1
Enter the verices adjacent to vertex: 2
Enter the vertex: 2
Enter the verices adjacent to vertex: 3 4
Enter the vertex: 3
Enter the verices adjacent to vertex: 0
Enter the vertex: 4
Enter the verices adjacent to vertex: 2

 ADJACENCY LIST
 +--------+----------------+
 | Vertex | Adjacenct List |
 +--------+----------------+
 |   0    | 1              |
 |   1    | 2              |
 |   2    | 3 4            |
 |   3    | 0              |
 |   4    | 2              |
 +--------+----------------+
BFS OUTPUT:
0   1   2   3   4
DFS OUTPUT:
0   1   2   3   4
```