**CS8381 - DATA STRUCTURES LABORATORY**

**Guidelines**

Aim

Algorithm

Program

Sample Input Output

Conclusion

**List of Exercises**

Ex.No.1. Array implementation of List ADT

Ex.No.2. Linked list implementation of List ADT

Ex.No.3. Array implementation of Stack ADT

Ex.No.4. Linked list implementation of Stack ADT

Ex.No.5. Array implementation of Queue ADT

Ex.No.6. Linked list implementation of Queue ADT

Ex.No.7. Applications of List - Polynomial Addition

Ex.No.8 Implementation of Infix to postfix conversion

Ex.No.9. Implementation of Postfix Evaluation

Ex.No.10 Implementation of balancing parenthesis

Ex:No.11 Implementation of binary search tree ADT

Ex.No.12 Implementation of Graph Representation

Ex.No.13 Implementation of Breadth First Traversal and Depth First Traversal

Ex.No.14. Implementation of Prim’s Algorithm and Dijkstra’s Algorithm

Ex.No.15. Implementation of Linear and binary search.

Ex.No.16. Implementation of Sorting Techniques

Ex.No.17. Implementation of Hashing Techniques

**Ex.No.1. Array implementation of List ADT**

| #include <stdio.h>  **int maxsize 10 int a[maxsize];**  **int currentsize=0; void insert(int,int); void del(int);**  **int find(int); void printlist();**  int main()  {  int choice=0,p,x; while(choice<=4)  {  clrscr();  printf("1.insert \n2.delete \n3.find \n4.display  \n5.exit\nEnter your choice"); scanf("%d",&choice); switch(choice)  {  case 1: printf("Enter the possition and value "); scanf("%d %d",&p,&x);  insert(p,x); break;  case 2: printf("Enter the Element to delete"); scanf("%d",&x);  del(x); break;  case 3: printf("Enter the Element to find"); scanf("%d",&x);  p=find(x);  printf("Element found at %d",p); break;  case 4: printlist();  break;  }  getch();  }  } |  | **void insert(int po,int x)**  {  int i=currentsize; while(po<=i)  {a[i]=a[i-1]; i--;  }  a[i]=x; currentsize=currentsize+1;  }  **void del(int x)**  {  int i,po; po=find(x); if(po!=-1)  {  i=po; while(i<currentsize)  {  a[i-1]=a[i]; i++;  }  currentsize=currentsize-1;  }  else  printf("Element not found");  }  **int find(int x)**  {  int i=0;  while(i<currentsize && a[i]!=x)  {i=i+1;}  if(i==currentsize)  {return -1;} else  {return (i+1);}  }  **void printlist()**  { int i; for(i=0;i<currentsize;i++) printf("%d\n",a[i]);  } |
| --- | --- | --- |

**Ex.No.2. Linked list implementation of List ADT**

| #include<stdio.h> #include<conio.h> **struct node**  **{**  **int element; struct node \*next;**  **};**  **typedef struct node\* list; list insert(list,int,int); list del(list,int);**  **list find(list,int);**  **list findprevious(list,int); void print(list);**  void main()  {  list l=NULL,t=NULL; int val,pos,choice=0; while(choice<5)  {  clrscr(); printf("\n1.Insert\n2.delete\n3.find\n4.Printlist\n 5.Exit\nEnter your choice"); scanf("%d",&choice);  switch(choice)  {  case 1:  printf("Enter position and element"); scanf("%d%d",&pos,&val); l=insert(l,pos,val);  break; case 2:  printf("Enter element to delete"); scanf("%d",&val);  l=del(l,val); break;  case 3:  printf("Enter element to find"); scanf("%d",&val); t=find(l,val);  printf("element found%d",t->element); break;  case 4:  print(l);  getch(); break;  }}} |  | **list insert(list l1,int pos,int v)**  {  list newnode,l2; int i;  l2=l1;  newnode=malloc(sizeof(struct node)); newnode->element=v;  if(pos==1)  {  newnode->next=l1; return newnode;  }  else  {  for(i=1;i<pos-1&&l1->next!=NULL;i++)  {  l1=l1->next;  }  newnode->next=l1->next; l1->next=newnode; return l2;  }  }  **list del(list l1,int v)**  {  list tmp,fp;  if(l1->element==v)  {  tmp=l1; l1=l1->next; free(tmp); return l1;  }  else  {  fp=findprevious(l1,v); tmp=fp->next;  fp->next=fp->next->next; free(tmp);  return l1;  }  }  **void print(list l1)**  { while(l1!=NULL)  { printf("%d->",l1->element); l1=l1->next;}} |
| --- | --- | --- |
| **list find(list l1,int v)**  {  while(l1!=NULL)  { if(l1->element==v) return l1;  l1=l1->next;  }  return NULL;  } | | **list findprevious(list l1,int v)**  {  while(l1->next!=NULL)  { if(l1->next->element==v) return l1;  l1=l1->next;  }  return NULL;  } |

**Ex.No.3. Array implementation of Stack ADT**

#include <stdio.h>

int maxsize=10;

int stack[10];

int sp=-1;

void push(int);

int pop();

void printstack();

int main()

{

int choice=0,p,x;

while(choice<4)

{

clrscr();

printf("1.push \n2.pop \n3.print \n4.exit

\nEnter your choice");

scanf("%d",&choice);

switch(choice)

{

case 1:

printf("Enter the value ");

scanf("%d",&x);

push(x);

break;

case 2:

x=pop();

if(x!=NULL)

printf("The popped element is %d",x);

break;

case 3:

printstack();

break;

}

getch();

}

}

**void push(int x)**

{

if(sp<maxsize-1)

stack[++sp]=x;

else

{

printf("Stack Full");

getch();

}

}

**int pop()**

{

int x;

if(sp!=-1)

{

x=stack[sp];

sp--;

return x;

}

else

{

printf("Stack Empty");

return NULL;

}

}

**void printstack()**

{

int i;

for(i=0;i<=sp;i++)

printf("%d\n",stack[i]);

}

**Ex.No.4. Linked list implementation of Stack ADT**

| include <stdio.h> struct node  {  int element; struct node \*next;  };  typedef struct node\* list; list push(list,int);  list top(list); list pop(list);  void printstack(list); int main()  { list l=NULL,tmp; int choice=0,p,x; while(choice<=4)  {  clrscr();  printf("1.push \n2.top \n3.Pop \n4.Print  \n5.exit\nEnter your choice"); scanf("%d",&choice); switch(choice)  {  case 1: printf("Enter the value "); scanf("%d",&x); l=push(l,x);  break;  case 2: tmp=top(l);  if(tmp!=NULL) printf("The poped  element is %d",tmp->element);  break; case 3: l=pop(l);  break;  case 4: printstack(l); break;  }  getch();  }  } | **list push(list l1,int v)**  {  list newnode; newnode=malloc(sizeof(struct node)); newnode->element=v;  newnode->next=l1; return newnode;  }  list top(list l1)  {  list tmp; if(l1!=NULL)  return tmp; else  {  printf("Stack Empty"); return NULL;  }  }  **list pop(list l1)**  {  list tmp; if(l1!=NULL)  {tmp=l1; l1=l1->next; free(tmp); return l1;  }  else  {  printf("Stack Empty"); return NULL;  }  }  **void printstack(list l1)**  {  while(l1!=NULL)  { printf("%d\n",l1->element); l1=l1->next;  }  } |
| --- | --- |

**Ex.No.5. Array implementation of Queue ADT**

| #include <stdio.h> int maxsize=10; int queue[10];  int front=-1; int rear=-1;  void enqueue(int); int dequeue(); void printqueue();  int main()  {  int choice=0,p,x; while(choice<4)  {  clrscr();  printf("1.Enqueue \n2.Dequeue \n3.Print \n4.exit\nEnter your choice");  scanf("%d",&choice); switch(choice)  {  case 1: printf("Enter the value "); | **void enqueue(int x)**  {  if(rear<maxsize-1)  { rear=rear+1; queue[rear]=x; if(front==-1) front=0;  }  else  {  printf("Queue Full");  }  }  **int dequeue()**  { int x;  if(front>-1 && front<=rear)  {  x=queue[front];  front=front+1; return x; |
| --- | --- |
| scanf("%d",&x); enqueue(x); break;  case 2: x=dequeue();  if(x!=NULL)  printf("The popped element is %d",x);  break;  case 3: printqueue(); break;  }  getch();  }  } | **}**  **else**  **{**  **printf("Queue Empty"); return NULL;**  **}**  **}**  **void printqueue()**  **{ int i;**  **for(i=front;i<=rear;i++) printf("%d\n",queue[i]);**  **}** |

**Ex.No.6. Linked list implementation of Queue ADT**

| #include <stdio.h> struct node  {  int element; struct node \*next;  };  typedef struct node\* list; list enqueue(list,int);  list top(list);  list dequeue(list); void printqueue(list); int main()  { list l=NULL,tmp=NULL; int choice=0,p,x; while(choice<=4)  {  clrscr();  printf("1.Enqueue \n2.top \n3.Dequeue  \n4.Print \n5.exit\nEnter your choice"); scanf("%d",&choice); switch(choice)  {  case 1: printf("Enter the value "); scanf("%d",&x); l=enqueue(l,x);  break;  case 2: tmp=top(l);  if(tmp!=NULL)  printf("The poped element is  %d",tmp->element);  break;  case 3: l=dequeue(l); break;  case 4: printqueue(l); break;  }  getch();  }  }  **void printqueue(list l1)**  {  while(l1!=NULL)  { printf("%d\n",l1->element); l1=l1->next;  }} | **list enqueue(list l1,int v)**  {  list newnode,l2=l1; newnode=malloc(sizeof(struct node)); newnode->element=v;  newnode->next=NULL; if(l1==NULL)  return newnode; else  {  while(l1->next!=NULL) l1=l1->next;  l1->next=newnode; return l2;  }  }  **list top(list l1)**  {  if(l1!=NULL)  return l1; else  {  printf("Queue Empty"); return NULL;  }  }  **list dequeue(list l1)**  {  list tmp; if(l1!=NULL)  { tmp=l1;  l1=l1->next; free(tmp); return l1;  }  else  {  printf("Queue Empty"); return NULL;  }  } |
| --- | --- |

**Ex.No.7. Applications of List - Polynomial Addition**

| #include<stdio.h> #include<conio.h> struct node  {  int coef; int exp;  struct node \*next;  };  typedef struct node\* list; list create(list);  list add(list,list,list); void printlist(list); void main()  {  list p1,p2,p3; int coef,exp; p1=create(p1); p2=create(p2);  printf("\nPolinomial one  :");  printlist(p1); printf("\nPolinomial two  :");  printlist(p2); p3=add(p3,p1,p2); printf("\nSum of polinomial:"); printlist(p3); getch();  } | **list create(list l1)**  {  int i=0,c,e,n; list newnode,l2;  printf("Enter no of node "); scanf("%d",&n); while(i<n)  { printf("Enter Coef and exp"); scanf("%d %d",&c,&e); newnode=malloc(sizeof(struct  node));  newnode->coef=c; newnode->exp=e; newnode->next=NULL; if(i==0)  { l1=newnode; l2=newnode;  }  else  { l1->next=newnode; l1=newnode;  } i++;  }  return l2;  }  **void printlist(list l1)**  {  while(l1!=NULL)  {  printf("%d %d ->",l1->coef,l1-  >exp);  l1=l1->next;  }  } | **list add(list l,list l1,list l2)**  {  list l3,newnode; int i=0;  while(l1!=NULL && l2!=NULL)  {newnode=malloc(sizeof(struct node));  newnode->next=NULL; if(l1->exp>l2->exp)  {newnode=l1; l1=l1->next;  }  else if(l2->exp>l1->exp)  { newnode=l2; l2=l2->next;  }  else  { newnode->coef=l1-  >coef+l2->coef;  newnode->exp=l1->exp; l1=l1->next;  l2=l2->next;  }  if(i==0)  { l=newnode; l3=l;  i++;  }  else  { l->next=newnode; l=newnode;  }  }  if(l1!=NULL) l->next=l1; if(l2!=NULL) l->next=l2; return l3;  } |
| --- | --- | --- |

**Ex.No.8 Infix to postfix conversion**

#include <stdio.h>

#include <string.h>

int maxsize=10;

char stack[10];

int sp=-1;

void push(char);

char pop();

int priority(char);

void main()

{

char expr[20],t; int x,i;

printf("Enter infix expression :: ");

scanf("%s",expr);

for(i=0;i<strlen(expr);i++)

{

if(isdigit(expr[i])

printf("%c",expr[i]);

else if(expr[i]=='(')

push(expr[i]);

else if(expr[i]==')')

{

while(stack[sp]!='(')

{

printf("%c",pop());

}

t=pop();

}

else

{

while(priority(stack[sp]) >= priority(expr[i]))

printf("%c",pop());

push(expr[i]);

}

}

while(sp != -1)

{

printf("%c",pop());

}

getch();

}

**void push(char x)**

{

if(sp<maxsize-1)

stack[++sp]=x;

else

{

printf("Stack Full");

getch();

}

}

**char pop()**

{

int x;

if(sp!=-1)

{

x=stack[sp];

sp--;

return x;

}

else

{

printf("Stack Empty");

return NULL;

}

}

**int priority(char x)**

{

if(x == '(') return 0;

if(x == '+' || x == '-') return 1;

if(x == '\*' || x == '/') return 2;

}

**Ex.No.9. Postfix Evaluation**

#include <stdio.h>

#include <string.h>

#include <ctype.h>

int maxsize=10;

int stack[10]; int sp=-1;

void push(int); int pop();

void main()

{

char expr[20];

int i,a,b;

printf("Enter post fix expression :: ");

scanf("%s",expr);

for(i=0;i<strlen(expr);i++)

{

if(isdigit(expr[i]))

push(expr[i]-48);

else

{

b=pop();

a=pop();

switch(expr[i])

{

case '+':

push(a+b); break;

case '-':

push(a-b); break;

case '\*':

push(a\*b); break;

case '/':

push(a/b); break;

}

}

}

printf("%d",pop());

}

**void push(int x)**

{

if(sp<maxsize-1)

stack[++sp]=x;

else

{

printf("Stack Full");

getch();

}

}

**int pop()**

{

int x;

if(sp!=-1)

{

x=stack[sp];

sp--;

return x;

}

else

{

printf("Stack Empty");

return NULL;

}

}

**Ex.No.10 Balancing parenthesis**

#include <stdio.h>

#include <string.h>

#include <stdio.h>

int maxsize=10;

int stack[10];

int sp=-1;

void push(int);

int pop();

int main()

{

int i;

char expr[10];

clrscr();

printf("Enter the expression : ");

scanf("%s",expr);

for(i=0;i<strlen(expr);i++)

{

if(expr[i]=='(' || expr[i]=='[' || expr[i]=='{')

push(expr[i]);

else if(expr[i]==')' && pop()=='(');

else if(expr[i]==']' && pop()=='[');

else if(expr[i]=='}' && pop()=='{');

else if(expr[i]!=')' && expr[i]!=']' && expr[i]!='}')

{}

else

{

printf("Unbalanced ");

getch();

exit(0);

}

}

if(sp == -1)

printf("Balanced");

else

printf("Unbalanced");

getch();

}

**void push(int x)**

{

if(sp<maxsize-1)

stack[++sp]=x;

else

{

printf("Stack Full");

getch();

}

}

**int pop( )**

{

int x;

if(sp!=-1)

{

x=stack[sp];

sp--;

return x;

}

else

{

printf("Stack Empty");

return NULL;

}

}

**Ex:No.11 Implementation of binary search tree ADT**

#include<stdio.h>

#include<stdlib.h>

struct node

{

int data;

struct node\* left;

struct node\* right;

};

typedef struct node \* tree;

tree createnode(int);

tree insert(tree,int);

tree find(tree,int);

tree findmin(tree);

tree findmax(tree);

tree deletenode(tree,int);

void inorder(tree);

int main()

{

tree root = NULL;

tree temp;

int ch,element;

clrscr();

while(ch<7)

{

clrscr();

printf("1. Insert\n2. Display\n3. Find\n4. Findmin\n5. Findmax\n6. Delete\n7.Exit");

printf("\nEnter your choice :");

scanf("%d",&ch);

switch(ch)

{

case 1:

printf("Enter element to insert ");

scanf("%d",&element);

root=insert(root,element);

break;

case 2:

printf("List of element:");

inorder(root);

break;

case 3:

printf("Enter element to find ");

scanf("%d",&element);

temp=find(root,element);

if(temp!=NULL)

printf("\nThe element is = %d",temp->data);

else

printf("\nElement not found");

break;

case 4: temp=findmin(root);

printf("the minimum element is :%d",temp->data);

break;

case 5: temp=findmax(root);

printf("the maximum element is :%d",temp->data);

break;

case 6:

printf("Enter the element to delete ");

scanf("%d",&element);

root=deletenode(root,element);

break;

}

getch();

}

}

**tree createnode(int value)**{

tree newNode = malloc(sizeof(struct node));

newNode->data = value;

newNode->left = NULL;

newNode->right = NULL;

return newNode;

}

**tree insert(tree root, int data)**

{

if (root == NULL)

return

createnode(data);

if (data < root->data)

root->left = insert(root->left, data);

else if (data > root->data)

root->right = insert(root->right, data);

return root;

}

**void inorder(tree root)**

{

if(root == NULL)

return;

inorder(root->left);

printf("%d ->", root->data);

inorder(root->right);

}

**tree find(tree t,int key)**

{

tree temp=t;

if(t==NULL)

return NULL;

if(key<t->data)

temp=find(t->left,key);

else if(key>t->data)

temp=find(t->right,key);

return temp;

}

**tree findmin(tree t)**

{

tree minnode= t;

while (minnode->left != NULL)

minnode= minnode->left;

return minnode;

}

**tree findmax(tree t)**

{

tree maxnode= t;

while (maxnode->right != NULL)

maxnode= maxnode->right;

return maxnode;

}

**tree deletenode(tree root, int key)**

{

tree temp;

if (root == NULL) return root;

if (key < root->data)

root->left = deletenode(root->left, key);

else if (key > root->data)

root->right = deletenode(root->right, key);

else

{

// node with only one child or no child

if (root->left == NULL)

{

tree temp = root->right; free(root);

return temp;

}

else if (root->right == NULL)

{

tree temp = root->left; free(root);

return temp;

}

//node with two children:

temp = findmin(root->right);

root->data = temp->data;

root->right = deletenode(root->right, temp->data);

}

return root;

}

**Ex.No.12 Implementation of Graph Representation**

#include<stdio.h>

#define MAX 100

int adj[MAX][MAX];

int n;

int main()

{

int max\_edges,i,j,origin,destin;

int graph\_type;

printf("\nEnter 1 for Undirected graph\nEnter 2 for Directed graph\n");

printf("\nEnter your choice :: ");

scanf("%d",&graph\_type);

printf("\nEnter number of vertices :: ");

scanf("%d",&n);

if(graph\_type==1)

max\_edges = n\*(n-1)/2;

else

max\_edges = n\*(n-1);

for(i=1; i<=max\_edges; i++)

{

printf("\nEnter edge [ %d ] ( -1 -1 to quit ) : ",i);

scanf("%d %d",&origin,&destin);

if( (origin == -1) && (destin == -1) )

break;

if( origin>=n || destin>=n || origin<0 || destin<0 )

{

printf("\nInvalid vertex!\n");

i--;

}

else

{

adj[origin][destin] = 1;

if( graph\_type == 1)

adj[destin][origin] = 1;

}

}

printf("\nThe adjacency matrix is :: \n");

for(i=0; i<=n-1; i++)

{

for(j=0; j<=n-1; j++)

printf("%4d",adj[i][j]);

printf("\n");

}

}

**Ex.No. 13 Implementation of Breadth First Traversal and Depth first traversal**

**PROBLEM STATEMENT:**

To implement Breadth first traversal in graphs.

**Algorithm:**

1. Construct a graph.

2. To traverse a graph in breadth first technique ,label vertex v as reached.

3. Initialize Q to be a queue with only v in it.

4. While Q is not empty, do the following steps

5. Delete a vertex W from the queue

6. Let u be a vertex adjacent from w.

7. While u, if u has not been labeled then add u to the queue label u as reached.

8. Set u = next vertex, that is adjacent from w

9. To traverse a graph in DFS label vertex v as reached.

10. While u is adjacent to v, if u is not reached call DFS recursively

11. Set u as next adjacent vertex of v. Repeat from step 9 till all the nodes are visited.

**Breadth first traversal**

/\* Breadth first traversal\*/

#include<stdio.h>

#include<conio.h>

#define MAX 10

int adj[MAX][MAX];

int visited[MAX],n,e,v1,v2,ch,i,j;

void read()

{

printf("1.Directed graph\n2.Undirected graph");

scanf("%d",&ch);

printf("Enter the no.of nodes");

scanf("%d",&n);

for(i=1;i<=n;i++)

for(j=1;j<=n;j++)

adj[i][j]=0;

printf("Enter the no of edges");

scanf("%d",&e);

while(e>0)

{

scanf("%d %d",&v1,&v2);

if(ch==1)

{

adj[v1][v2]=1;

}

else

{

adj[v1][v2]=1;

adj[v2][v1]=1;

}

e--;

}

for (i = 1; i <= n; i++)

visited[i] = 0;

}

void bfs(int source)

{

int queue[MAX];

int i, front, rear, root;

front = rear = 0;

visited[source] = 1;

queue[rear++] = source;

printf("%d\t",source );

while (front != rear)

{

root = queue[front];

for (i = 1; i <= n; i++)

if (adj[root][i] && !visited[i])

{

visited[i] = 1;

queue[rear++] = i;

printf("%d \t ", i);

}

front++;

}

}

void main()

{

int source;

clrscr();

read();

printf("\nEnter the Source : ");

scanf("%d",&source);

printf("\nThe nodes visited in the BFS order is : ");

bfs(source);

getch();

}

**Output for Breadth first search:**

1.Directed graph

2.Undirected graph

1

Enter the no.of nodes 4

Enter the no of edges 5

1 2

2 4

3 2

1 3

1 4

Enter the Source : 1

**The nodes visited in the BFS order is : 1 2 3 4**

1.Directed graph

2.Undirected graph2

Enter the no.of nodes4

Enter the no of edges5

1 2

2 3

3 1

1 4

4 2

Enter the Source : 2

**The nodes visited in the BFS order is : 2 1 3 4**

**DEPTH FIRST TRAVERSAL**

**PROBLEM STATEMENT:**

To implement Depth first traversal in graphs.

**Algorithm:**

1. Construct a graph.

2. To traverse a graph in depth first technique ,label vertex v as reached.

3. Let u be a vertex adjacent from w.

4. To traverse a graph in DFS label vertex v as reached.

5. While u is adjacent to v, if u is not reached call DFS recursively

6. Set u as next adjacent vertex of v.

7. Repeat from step 9 till all the nodes are visited.

**Depth first Traversal:**

/\* Depth first traversal \*/

#include<stdio.h>

#include<conio.h>

#define MAX 10

int adj[MAX][MAX];

int visited[MAX],n,e,v1,v2,ch,i,j;

void read()

{

printf("1.Directed graph\n2.Undirected graph");

scanf("%d",&ch);

printf("Enter the no.of nodes");

scanf("%d",&n);

for(i=1;i<=n;i++)

for(j=1;j<=n;j++)

adj[i][j]=0;

printf("Enter the no of edges");

scanf("%d",&e);

while(e>0)

{

scanf("%d %d",&v1,&v2);

if(ch==1)

{

adj[v1][v2]=1;

}

else

{

adj[v1][v2]=1;

adj[v2][v1]=1;

}

e--;

}

for (i = 1; i <= n; i++)

visited[i] = 0;

}

void dfs(int source)

{

visited[source] = 1;

printf(" %d",source);

for (i = 1; i <= n; i++)

if (adj[source][i] && !visited[i])

dfs(i);

}

void main()

{

int source;

read();

printf("\nEnter the Source : ");

scanf("%d",&source);

printf("\nThe nodes visited in the DFS order is :\n ");

dfs(source);

getch();

}

**Output for Depth first search:**

1.Directed graph

2.Undirected graph

1

Enter the no.of nodes4

Enter the no of edges5

1 2

2 4

2 3

3 4

4 1

Enter the Source : 2

The nodes visited in the DFS order is :

2 3 4 1

1.Directed graph

2.Undirected graph

2

Enter the no.of nodes3

Enter the no of edges4

1 2

2 3

3 4

4 1

Enter the Source : 3

The nodes visited in the DFS order is :

3 2 1

**Ex.No.14. Implementation of Prim’s Algorithm and Dijkstra’s Algorithm**

**Prims Algorithm**

#include<stdio.h>

#include<conio.h>

int a,b,u,v,n,i,j,ne=1;

int visited[10]={0},min, mincost=0,cost[10][10];

void main( )

{

clrscr();

printf("\n Enter the number of nodes:");

scanf("%d",&n);

printf("\n Enter the adjacency matrix:\n");

for(i=1;i<=n;i++)

for(j=1;j<=n;j++)

{

scanf("%d",&cost[i][j]);

if(cost[i][j]==0)

cost[i][j]=999;

}

visited[1]=1;

printf("\n");

while(ne<n)

{

for(i=1,min=999;i<=n;i++)

for(j=1;j<=n;j++)

if(cost[i][j]<min)

if(visited[i]!=0)

{

min=cost[i][j];

a=u=i;

b=v=j;

}

if(visited[u]==0||visited[v]==0)

{

printf("\nEdge %d:(%d%d) cost:%d",ne++,a,b,min);

mincost+=min;

visited[b]=1;

}

cost[a][b]=cost[b][a]=999;

}

printf("\n Minimum cost=%d",mincost);

for(i=1;i<=n;i++)

for(j=1;j<=n;j++)

printf("\n%d\t",cost[i][j]);

getch();

}

**Dijkstra’s Algorithm**

#include <stdio.h>

#include <conio.h>

#define infinity 999

void dij(int n,int v,int cost[10][10],int dist[])

{

int i,u,count,w,flag[10],min;

for(i=1;i<=n;i++)

flag[i]=0,dist[i]=cost[v][i];

count=2;

while(count<=n)

{

min=99;

for(w=1;w<=n;w++)

if(dist[w]<min && !flag[w])

min=dist[w],u=w;

flag[u]=1;

count++;

for(w=1;w<=n;w++)

if((dist[u]+cost[u][w]<dist[w]) && !flag[w])

dist[w]=dist[u]+cost[u][w];

}

}

void main()

{

int n,v,i,j,cost[10][10],dist[10];

clrscr();

printf("\n Enter the number of nodes:");

scanf("%d",&n);

printf("\n Enter the cost matrix:\n");

for(i=1;i<=n;i++)

for(j=1;j<=n;j++)

{

scanf("%d",&cost[i][j]);

if(cost[i][j]==0)

cost[i][j]=infinity;

}

printf("\n Enter the source node:");

scanf("%d",&v);

dij(n,v,cost,dist);

printf("\n Shortest path:\n");

for(i=1;i<=n;i++)

if(i!=v)

printf("%d->%d,cost=%dn",v,i,dist[i]);

getch();

}