**1.Write a program to sort a list of N elements using selection sort Technique.**

#include<stdio.h>

#include<conio.h>

void selection\_sort(int[],int n);

void main()

{

int n,a[10],i;

printf("enetr size:");

scanf("%d",&n);

printf("eneter elements of array:");

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

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

selection\_sort(a,n);

printf("after sorting element are\n");

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

printf("%d\t",a[i]);

}

void selection\_sort(int a[],int n)

{

int i,j,min,temp;

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

{

min=i;

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

{

if(a[j]<a[min])

min=j;

}

temp=a[i];

a[i]=a[min];

a[min]=temp;

}

}

**2.Write a program to perform Travelling Salesman problem.**

#include <stdio.h>

int tsp\_g[10][10] =

{

{0, 10, 15, 20},

{5, 0, 9, 10},

{6, 13, 0, 12},

{8, 8, 9, 0},

};

int visited[10], n, cost = 0;

void travellingsalesman(int i)

{

int j, adj\_vertex = 10;

int min = 10;

visited[i] = 1;

printf("%d ", i + 1);

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

{

if((tsp\_g[i][j] != 0) && (visited[j] == 0))

{

if(tsp\_g[i][j] < min)

{

min = tsp\_g[i][j];

}

adj\_vertex = j;

}

}

if(min != 10)

{

cost = cost + min;

}

if(adj\_vertex == 10)

{

adj\_vertex = 0;

printf("%d", adj\_vertex + 1);

cost = cost + tsp\_g[i][adj\_vertex];

return;

}

travellingsalesman(adj\_vertex);

}

int main()

{

int i, j;

n = 5;

for(i = 0; i < n; i++) {

visited[i] = 0;

}

printf("Shortest Path: ");

travellingsalesman(0);

printf("\nMinimum Cost: ");

printf("%d\n", cost);

return 0;

}

**3.Write a program to implement dynamic Programming algorithm for 0/1 Knapsack problrm.**

#include<stdio.h>

int main()

{

int i, n, profit[20], weight[20], W;

printf("Enter number of items:");

scanf("%d", &n);

printf("Enter profit and weight of items:\n");

for(i=0;i<n;i++){

scanf("%d%d", &profit[i], &weight[i]);

}

printf("Enter size of knapsack:");

scanf("%d", &W);

printf("%d", knapSack(W,weight, profit, n));

return 0;

}

int max(int a, int b) { return (a > b)? a : b; }

int knapSack(int W, int weight[], int profit[], int n)

{

int i, w;

int K[n+1][W+1];

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

{

for (w = 0; w <= W; w++)

{

if (i==0 || w==0)

K[i][w] = 0;

else if (weight[i-1] <= w)

K[i][w] = max(profit[i-1] + K[i-1][w-weight[i-1]], K[i-1][w]);

else

K[i][w] = K[i-1][w];

}

}

return K[n][W];

}

**4.Write a program to perform Knapsack Problem using Greedy Solution.**

#include<stdio.h>

int main()

{

float weight[10],profit[10],ratio[10],Totalvalue,temp,capacity,amount;

int i,j,num;

printf("Enter number of items :");

scanf("%d",&num);

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

{

printf("\n\nEnter Profit  and Weight for item[%d] :\n",i);

scanf("%f %f",&profit[i],&weight[i]);

}

printf("\n\nEnter capacity of knapsack :\n");

scanf("%f",&capacity);

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

ratio[i]=profit[i]/weight[i];

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

{

for (j = i + 1; j < num; j++)

{

if (ratio[i] < ratio[j])

{

temp = ratio[j];

ratio[j] = ratio[i];

ratio[i] = temp;

temp = weight[j];

weight[j] = weight[i];

weight[i] = temp;

temp = profit[j];

profit[j] = profit[i];

profit[i] = temp;

}

}

}

printf("\nKnapsack Problem using Greedy Method :\n");

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

{

if (weight[i] > capacity)

break;

else

{

Totalvalue = Totalvalue + profit[i];

capacity = capacity - weight[i];

}

}

if (i < num)

Totalvalue = Totalvalue + (ratio[i]\*capacity);

printf("\nThe maximum value is :%f\n",Totalvalue);

return 0;

}

**5.Write a program to implement the DFS and BFS algorithm for a graph.**

#include<stdio.h>  
int q[20],top=-1,front=-1,rear=-1,a[20][20],vis[20],stack[20];  
int delete();  
void add(int item);  
void bfs(int s,int n);  
void dfs(int s,int n);  
void push(int item);  
int pop();  
void main()  
{  
int n,i,s,ch,j;  
char c,useless;  
printf("ENTER THE NUMBER VERTICES ");  
scanf("%d",&n);  
for(i=1;i<=n;i++)  
{  
for(j=1;j<=n;j++)  
{  
printf("ENTER THE NODE ",i,j);  
scanf("%d",&a[i][j]);  
}  
}  
printf("THE ADJACENCY MATRIX IS\n");  
for(i=1;i<=n;i++)  
{  
for(j=1;j<=n;j++)  
{  
printf(" %d",a[i][j]);  
}  
printf("\n");  
}  
  
do  
{  
for(i=1;i<=n;i++)  
vis[i]=0;  
printf("\nMENU");

printf("\n1.BFS");  
printf("\n2.DFS");  
printf("\nENTER YOUR CHOICE");  
scanf("%d",&ch);  
printf("ENTER THE SOURCE VERTEX :");  
scanf("%d",&s);  
  
switch(ch)  
{  
case 1:bfs(s,n);  
break;  
case 2:  
dfs(s,n);  
break;  
}  
printf("DO U WANT TO CONTINUE(Y/N) ? ");  
scanf("%c",&useless);  
scanf("%c",&c);  
}  
while((c=='y')||(c=='Y'));  
}

void bfs(int s,int n)  
{  
int p,i;  
add(s);  
vis[s]=1;  
p=delete();  
if(p!=0)  
printf(" %d",p);  
while(p!=0)  
{  
for(i=1;i<=n;i++)  
if((a[p][i]!=0)&&(vis[i]==0))  
{  
add(i);  
vis[i]=1;  
}  
p=delete();  
if(p!=0)  
printf(" %d ",p);  
}  
for(i=1;i<=n;i++)  
if(vis[i]==0)  
bfs(i,n);  
}  
  
  
void add(int item)  
{  
if(rear==19)  
printf("QUEUE FULL");  
else  
{  
if(rear==-1)  
{  
q[++rear]=item;  
front++;  
}  
else  
q[++rear]=item;  
}  
}  
int delete()  
{  
int k;  
if((front>rear)||(front==-1))  
return(0);  
else  
{  
k=q[front++];  
return(k);  
}  
}  
  
  
  
void dfs(int s,int n)  
{  
int i,k;  
push(s);  
vis[s]=1;  
k=pop();  
if(k!=0)  
printf(" %d ",k);  
while(k!=0)  
{  
for(i=1;i<=n;i++)  
if((a[k][i]!=0)&&(vis[i]==0))  
{  
push(i);  
vis[i]=1;  
}  
k=pop();  
if(k!=0)  
printf(" %d ",k);  
}  
for(i=1;i<=n;i++)  
if(vis[i]==0)  
dfs(i,n);  
}  
void push(int item)  
{  
if(top==19)  
printf("Stack overflow ");  
else  
stack[++top]=item;  
}  
int pop()  
{  
int k;  
if(top==-1)  
return(0);  
else  
{  
k=stack[top--];  
return(k);  
}  
}

**6.Write program to find minimum and maximum value in an array using divide and conquer.**

#include<stdio.h>  
int max, min;  
int a[100];  
void maxmin(int i, int j)  
{  
 int max1, min1, mid;  
 if(i==j)  
 {  
  max = min = a[i];  
 }  
 else  
 {  
  if(i == j-1)  
  {  
   if(a[i] <a[j]

{  
    max = a[j];  
    min = a[i];  
   }  
   else  
   {  
    max = a[i];  
    min = a[j];  
   }  
  }  
  else  
  {  
   mid = (i+j)/2;  
   maxmin(i, mid);  
   max1 = max;  
   min1 = min;  
   maxmin(mid+1, j);  
   if(max <max1)  
    max = max1;  
   if(min > min1)  
    min = min1;  
  }  
 }  
}  
int main ()  
{  
 int i, num;  
 printf ("\nEnter the total number of numbers : ");  
 scanf ("%d",&num);  
 printf ("Enter the numbers : \n");  
 for (i=1;i<=num;i++)  
  scanf ("%d",&a[i]);  
  
 max = a[0];  
 min = a[0];  
 maxmin(1, num);  
 printf ("Minimum element in an array : %d\n", min);  
 printf ("Maximum element in an array : %d\n", max);  
 return 0;  
}

**7.Write a test program to implement Divide and Conquer strategy .Eg: Quick sort algorithm for sorting list of integers in ascending order.**

#include<stdio.h>

void quicksort();

int main()

{

int n,a[10],i;

printf("enetr size:");

scanf("%d",&n);

printf("eneter elements of array:");

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

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

quicksort(a,0,n-1);

printf("after sorting element are\n");

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

printf("%d\t",a[i]);

return 0;

}

void quicksort(int number[25],int low,int high)

{

int i, j, pivot, temp;

if(low<high)

{

pivot=low;

i=low;

j=high;

while(i<j)

{

while(number[i]<=number[pivot]&&i<high)

i++;

while(number[j]>number[pivot])

j--;

if(i<j)

{

temp=number[i];

number[i]=number[j];

number[j]=temp;

}

}

temp=number[low];

number[low]=number[j];

number[j]=temp;

quicksort(number,low,j-1);

quicksort(number,j+1,high);

}

}

**8.Write a program to implement Merge sort algorithm for sorting a list of integers in ascending order.**

#include<stdio.h>

void mergesort();

void merge();

int main()

{

int n,a[10],i;

printf("enetr size:");

scanf("%d",&n);

printf("eneter elements of array:");

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

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

mergesort(a,0,n-1);

printf("after sorting element are\n");

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

printf("%d\t",a[i]);

return 0;

}

void mergesort(int a[],int low,int high)

{

int mid;

if(low<high)

{

mid=(low+high)/2;

mergesort(a,low,mid);

mergesort(a,mid+1,high);

merge(a,low,mid,high);

}

}

void merge(int a[],int low,int mid,int high)

{

int i,j,k,b[20];

i=low;

j=mid+1;

k=low;

while(i<=mid&&j<=high)

{

if(a[i]<a[j])

{

b[k]=a[i];

i++;

}

else

{

b[k]=a[j];

j++;

}

k++;

}

if(i>mid)

{

while(j<=high)

b[k++]=a[j++];

}

else

{

while(i<=mid)

b[k++]=a[i++];

}

for(i=low;i<=high;i++)

{

a[i]=b[i];

}

}

**9.Sort a given set of n integer elements using Merge sort method and compute its time complexity. Run the program for varied values of n>5000, and record the time taken to sort.**

#include<stdio.h>  
#include<math.h>  
#include<stdlib.h>  
#include<time.h>  
void mergesort( int \* ,int,int);  
void merge(int \*,int,int,int);  
void main()  
{  
int a[20],n,i;  
double t;  
printf("eneter the number:");  
scanf("%d",&n);  
printf(" enter the elements:");  
for(i=0;i<n;i++)  
scanf("%d",&a[i]);  
printf(" original list");  
for(i=0;i<n;i++)  
printf("%4d",a[i]);  
clock\_t start =clock();  
mergesort(a,0,n-1);  
clock\_t end =clock();  
printf(" sorted list");  
for(i=0;i<n;i++)  
printf("%4d",a[i]);  
t=n\*log(n);  
printf(" time complexity is %f",t);  
double cpu\_time\_used =((double)(end - start)) / CLOCKS\_PER\_SEC;  
printf(" time taken fro storing micclenious:%f",cpu\_time\_used);  
return ;  
}  
  
void mergesort(int a[],int low,int high)  
{  
int mid;  
if(low<high)  
{  
mid=(low+high)/2;  
mergesort(a,low,mid);  
mergesort(a,mid+1,high);  
merge(a,low,mid,high);  
}  
}  
void merge(int a[],int low,int mid,int high)  
{  
int i,j,k,b[20];  
i=low;  
j=mid+1;  
k=low;  
while(i<=mid&&j<=high)  
{  
if(a[i]<a[j])  
{  
b[k]=a[i];  
i++;  
}  
else  
{  
b[k]=a[j];  
j++;  
}  
k++;  
}  
if(i>mid)  
{  
while(j<=high)  
b[k++]=a[j++];  
}  
else  
{  
while(i<=mid)  
b[k++]=a[i++];  
}  
for(i=low;i<=high;i++)  
{  
a[i]=b[i];  
}  
}

**10.Sort a given set of n integer elements using Quick sort method and compute its time complexity. Run the program for varied values of n>5000, and record the time taken to sort.**

#include<stdio.h>  
#include<math.h>  
#include<stdlib.h>  
#include<time.h>  
void quicksort( int a[],int low,int high);  
void main()  
{  
int a[20],n,i,low,high;  
double t;  
printf("eneter the number:");  
scanf("%d",&n);  
printf(" enter the elements:");  
for(i=0;i<n;i++)  
scanf("%d",&a[i]);  
printf(" original list");  
for(i=0;i<n;i++)  
printf("%4d",a[i]);  
clock\_t start =clock();  
quicksort(a,0,n-1);  
clock\_t end =clock();  
printf(" sorted list");  
for(i=0;i<n;i++)  
printf("%4d",a[i]);  
t=n\*log(n);  
printf(" time complexity is %f",t);  
double cpu\_time\_used =((double)(end - start)) / CLOCKS\_PER\_SEC;  
printf(" time taken fro storing micclenious:%f",cpu\_time\_used);  
return ;  
}  
void quicksort(int number[25],int low,int high)  
{  
  
int i, j, pivot, temp;  
  
if(low<high)  
{  
  
pivot=low;  
  
i=low;  
  
j=high;  
  
while(i<j)  
{  
  
while(number[i]<=number[pivot]&&i<high)  
  
i++;  
  
while(number[j]>number[pivot])  
  
j--;  
  
if(i<j)  
{  
  
temp=number[i];  
  
number[i]=number[j];  
  
number[j]=temp;  
  
}  
  
}  
  
temp=number[low];  
  
number[low]=number[j];  
  
number[j]=temp;  
  
quicksort(number,low,j-1);  
  
quicksort(number,j+1,high);  
  
}  
  
}

**11.Write c program that accepts the vertices and edges for a graph and stores it as an adjacency matrix.**

#include <stdio.h>

#define MAX\_VERTICES 100

int main()

{

int adjMatrix[MAX\_VERTICES][MAX\_VERTICES] = {0};

int numVertices, numEdges;

int i, j, u, v;

printf("Enter the number of vertices in the graph: ");

scanf("%d", &numVertices);

printf("Enter the number of edges in the graph: ");

scanf("%d", &numEdges);

printf("Enter the edges (u, v):\n");

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

{

scanf("%d %d", &u, &v);

adjMatrix[u][v] = 1;

adjMatrix[v][u] = 1;

}

printf("\nAdjacency Matrix:\n");

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

{

for (j = 0; j < numVertices; j++)

{

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

}

printf("\n");

}

return 0;

}

**12.Implement function to print IN-Degree ,Out-Degree and to display the adjacency matrix.**

#include<stdio.h>

#define MAX 10

void accept\_graph(int G[][MAX], int n)

{

int i,j;

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

{

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

{

printf("Edge (V%d,V%d) exists:",i,j);

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

}

}

}

void disp\_adj\_mat(int G[][MAX], int n)

{

int i,j;

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

{

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

{

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

}

printf("\n");

}

}

void calc\_out\_degree(int G[][MAX], int n)

{

int i,j,sum;

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

{

sum=0;

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

{

sum += G[i][j];

}

printf("out-deg(V%d)=%d\n",i,sum);

}

}

void calc\_in\_degree(int G[][MAX], int n)

{

int i,j,sum;

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

{

sum=0;

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

{

sum += G[j][i];

}

printf("in-deg(V%d)=%d\n",i,sum);

}

}

void main()

{

int G[MAX][MAX],n;

printf("Enter no.of vertices:");

scanf("%d",&n);

accept\_graph(G,n);

printf("Adjacency Matrix:\n");

disp\_adj\_mat(G,n);

printf("Out degree:\n");

calc\_out\_degree(G,n);

printf("In degree:\n");

calc\_in\_degree(G,n);

}

**13.Write a program to implement backtracking algorithm for solving problems like N queens.**

#include<stdio.h>

#include<conio.h>

#include<math.h>

int a[30],count=0;

int place(int pos)

{

int i;

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

{

if((a[i]==a[pos])||((abs(a[i]-a[pos])==abs(i-pos))))

return 0;

}

return 1;

}

void print\_sol(int n)

{

int i,j;

count++;

printf("\n\nSolution #%d:\n",count);

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

{

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

{

if(a[i]==j)

printf("Q\t");

else printf("\*\t");

}

printf("\n");

}

}

void queen(int n)

{

int k=1;

a[k]=0;

while(k!=0)

{

a[k]=a[k]+1;

while((a[k]<=n)&&!place(k))

a[k]++;

if(a[k]<=n)

{

if(k==n)

print\_sol(n);

else

{

k++;

a[k]=0;

}

}

else k--;

}

}

void main()

{

int i,n;

clrscr();

printf("Enter the number of Queens\n");

scanf("%d",&n);

queen(n);

printf("\nTotal solutions=%d",count);

getch();

}

**14.Write a program to implement backtracking algorithm for the sum of subsets problem.**

#include <stdio.h>

#include <stdlib.h>

static int total\_nodes;

void printValues(int A[], int size){

for (int i = 0; i < size; i++) {

printf("%\*d", 5, A[i]);

}

printf("\n");

}

void subset\_sum(int s[], int t[], int s\_size, int t\_size, int sum, int ite, int const target\_sum){

total\_nodes++;

if (target\_sum == sum) {

printValues(t, t\_size);

subset\_sum(s, t, s\_size, t\_size - 1, sum - s[ite], ite + 1, target\_sum);

return;

}

else {

for (int i = ite; i < s\_size; i++) {

t[t\_size] = s[i];

subset\_sum(s, t, s\_size, t\_size + 1, sum + s[i], i + 1, target\_sum);

}

}

}

void generateSubsets(int s[], int size, int target\_sum){

int\* tuplet\_vector = (int\*)malloc(size \* sizeof(int));

subset\_sum(s, tuplet\_vector, size, 0, 0, 0, target\_sum);

free(tuplet\_vector);

}

int main(){

int set[] = { 5, 6, 12 , 54, 2 , 20 , 15 };

int size = sizeof(set) / sizeof(set[0]);

printf("The set is ");

printValues(set , size);

generateSubsets(set, size, 25);

printf("Total Nodes generated %d\n", total\_nodes);

return 0;

}

**15.Write a program to implement greedy algorithm for job sequencing with deadlines.**

#include <stdio.h>

#include <stdlib.h>

#include <stdbool.h>

typedef struct

{

char id;

int deadline;

int profit;

}Job;

int compareJob(const Job \*a, const Job \*b)

{

return b->profit - a->profit;

}

void bestJob(Job jobs[],int sizeOfJobs)

{

char jobsToDo[5]= {'\0'};

for(int i=0, k=0; i<sizeOfJobs; i++)

{

k = jobs[i].deadline-1;

while(jobsToDo[k] != '\0' && k >= 0)

{

k--;

}

if(k != -1)

jobsToDo[k]= jobs[i].id;

}

printf("\nBest order and jobs to do is: ");

int idx=0;

while(jobsToDo[idx] != '\0'){

printf("%c ",jobsToDo[idx]);

idx++;

}

}

void display(Job jobs[], int n){

printf("Job Id: \t");

for(int i=0; i<n; i++){

printf("%c \t",jobs[i].id);

}

printf("\n");

printf("Job Deadline: \t");

for(int i=0; i<n; i++){

printf("%d \t",jobs[i].deadline);

}

printf("\n");

printf("Job Profit: \t");

for(int i=0; i<n; i++){

printf("%d \t",jobs[i].profit);

}

printf("\n");

}

int main()

{

Job jobs[]= {{'w', 1, 19}, {'v', 2, 100}, {'x', 2, 27},

{'y', 1, 25}, {'z', 3, 15}};

display(jobs,5);

qsort(jobs,5,sizeof(jobs[0]),compareJob);

bestJob(jobs,5);

return 0;

}

**16.** **Write a c program to implement binary search tree operations.**

#include <stdio.h>

#include <stdlib.h>

struct node {

  int item;

  struct node\* left;

  struct node\* right;

};

void inorderTraversal(struct node\* root) {

  if (root == NULL) return;

  inorderTraversal(root->left);

  printf("%d ->", root->item);

  inorderTraversal(root->right);

}

void preorderTraversal(struct node\* root) {

  if (root == NULL) return;

  printf("%d ->", root->item);

  preorderTraversal(root->left);

  preorderTraversal(root->right);

}

void postorderTraversal(struct node\* root) {

  if (root == NULL) return;

  postorderTraversal(root->left);

  postorderTraversal(root->right);

  printf("%d ->", root->item);

}

struct node\* createNode(value) {

  struct node\* newNode = malloc(sizeof(struct node));

  newNode->item = value;

  newNode->left = NULL;

  newNode->right = NULL;

  return newNode;

}

struct node\* insertLeft(struct node\* root, int value) {

  root->left = createNode(value);

  return root->left;

}

struct node\* insertRight(struct node\* root, int value) {

  root->right = createNode(value);

  return root->right;

}

int main() {

  struct node\* root = createNode(1);

  insertLeft(root, 12);

  insertRight(root, 9);

  insertLeft(root->left, 5);

  insertRight(root->left, 6);

  printf("Inorder traversal \n");

  inorderTraversal(root);

  printf("\nPreorder traversal \n");

  preorderTraversal(root);

  printf("\nPostorder traversal \n");

  postorderTraversal(root);

}

**17.Write a program that implements Prim’s algorithm to generate minimum cost spanning Tree.**

**#include <stdio.h>**

**#include <limits.h>**

**#define V 5**

**int minKey(int key[], int mstSet[])**

**{**

**int min = INT\_MAX, min\_index;**

**int v;**

**for (v = 0; v < V; v++)**

**if (mstSet[v] == 0 && key[v] < min)**

**min = key[v], min\_index = v;**

**return min\_index;**

**}**

**int printMST(int parent[], int n, int graph[V][V])**

**{**

**int i;**

**printf("Edge Weight\n");**

**for (i = 1; i < V; i++)**

**printf("%d - %d %d \n", parent[i], i, graph[i][parent[i]]);**

**}**

**void primMST(int graph[V][V])**

**{**

**int parent[V];**

**int key[V], i, v, count;**

**int mstSet[V];**

**for (i = 0; i < V; i++)**

**key[i] = INT\_MAX, mstSet[i] = 0;**

**key[0] = 0;**

**parent[0] = -1;**

**for (count = 0; count < V - 1; count++) {**

**int u = minKey(key, mstSet);**

**mstSet[u] = 1;**

**for (v = 0; v < V; v++)**

**if (graph[u][v] && mstSet[v] == 0 && graph[u][v] < key[v])**

**parent[v] = u, key[v] = graph[u][v];**

**}**

**printMST(parent, V, graph);**

**}**

**int main() {**

**int graph[V][V] = { { 0, 2, 0, 6, 0 }, { 2, 0, 3, 8, 5 },**

**{ 0, 3, 0, 0, 7 }, { 6, 8, 0, 0, 9 }, { 0, 5, 7, 9, 0 }, };**

**primMST(graph);**

**return 0;**

**}**

**18.write a program that implement Kruskal’s algorithm to generate minimum cost spanning tree.**

**#include <stdio.h>**

**#include <stdlib.h>**

**const int inf = 999999;**

**int k, a, b, u, v, n, ne = 1;**

**int mincost = 0;**

**int cost[3][3] = {{0, 10, 20},{12, 0,15},{16, 18, 0}};**

**int p[9] = {0};**

**int applyfind(int i)**

**{**

**while(p[i] != 0)**

**i=p[i];**

**return i;**

**}**

**int applyunion(int i,int j)**

**{**

**if(i!=j) {**

**p[j]=i;**

**return 1;**

**}**

**return 0;**

**}**

**int main()**

**{**

**n = 3;**

**int i, j;**

**for (i = 0; i < n; i++)**

**{**

**for (j = 0; j < n; j++) {**

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

**cost[i][j] = inf;**

**}**

**}**

**}**

**printf("Minimum Cost Spanning Tree: \n");**

**while(ne < n) {**

**int min\_val = inf;**

**for(i=0; i<n; i++) {**

**for(j=0; j <n; j++) {**

**if(cost[i][j] < min\_val) {**

**min\_val = cost[i][j];**

**a = u = i;**

**b = v = j;**

**}**

**}**

**}**

**u = applyfind(u);**

**v = applyfind(v);**

**if(applyunion(u, v) != 0) {**

**printf("%d -> %d\n", a, b);**

**mincost +=min\_val;**

**}**

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

**ne++;**

**}**

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

**return 0;**

**}**