Design and implement C/C++ Program to sort a given set of n integer elements using Selection Sort method and compute its time complexity. Run the program for varied values of n> 5000 and record the time taken to sort. Plot a graph of the time taken versus n. The elements can be read from a file or can be generated using the random number generator. analysis: worst case, average case and best case.

#include <stdio.h> #include <sys/time.h> #include <stdlib.h>

void selection(int a[], int n) { int i, j, min, temp;

for (i = 0; i <= n-2; i++) { min = i;

for (j = i+1; j <= n-1; j++) { if (a[j] < a[min])

min = j;

}

temp = a[min]; a[min] = a[i]; a[i] = temp;

}

}

void printArr(int a[], int n) { int i;

for (i = 0; i < n; i++) printf("%d ", a[i]);

}

void main() {

int a[10000],n, i; struct timeval t; double s, e;

printf("Enter the number of elements\n"); scanf("%d",&n);

// Code to test selection sort starts here printf("Enter the elements : "); for(i=0;i<n;i++)

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

printf("Before sorting array elements are \n"); printArr(a,n);

selection(a, n);

printf("After sorting array elements are \n"); printArr(a,n);

// Code to test selection sort ends here

/\*for(i=0;i<n;i++) a[i]=rand()%100;\*/

gettimeofday(&t,NULL); s=t.tv\_sec+(t.tv\_usec/1000000.0); selection(a, n); gettimeofday(&t,NULL); e=t.tv\_sec+(t.tv\_usec/1000000.0);

printf("\ntime taken to sort %d elements is %lf\n", n, (e-s));

}

Design and implement C/C++ Program to obtain the Topological ordering of vertices in a given digraph

#include <stdio.h>

#include <stdlib.h>

#define MAX 10

// Declare graph and in-degree array globally

int graph[MAX][MAX], inDegree[MAX];

int n; // Number of vertices

// Function to perform topological sorting

void topologicalSort() {

int i, j, k;

int count = 0; // Number of vertices processed

// Array to store the sorted result

int result[MAX];

int index = 0;

// Repeat until all vertices are processed

while (count < n) {

// Find a vertex with in-degree 0 (no incoming edges)

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

if (inDegree[i] == 0) {

// Add it to the result

result[index++] = i;

inDegree[i] = -1; // Mark this vertex as processed

count++;

// Reduce the in-degree of all neighbors

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

if (graph[i][j] == 1) {

inDegree[j]--;

}

}

break;

}

}

}

// Print the topological order

printf("Topological Sort: ");

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

printf("%d ", result[i]);

}

printf("\n");

}

int main() {

int i, j;

// Input number of vertices

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

scanf("%d", &n);

// Initialize the graph and in-degree array

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

inDegree[i] = 0; // Initially set all in-degrees to 0

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

graph[i][j] = 0; // Initialize the graph with no edges

}

}

// Input edges

printf("Enter the adjacency matrix (0 or 1):\n");

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

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

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

if (graph[i][j] == 1) {

inDegree[j]++; // Increase in-degree for vertex j

}

}

}

// Perform topological sort

topologicalSort();

return 0;

}

Design and implement C/C++ Program to 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. Plot a graph of the time taken versus n. The elements can be read from a file or can be generated using the random number generator.

Program

#include <stdio.h>

#include <stdlib.h>

#include <time.h>

// Function to merge two sorted arrays

void merge(int arr[], int left, int mid, int right)

{

int i, j, k;

int n1 = mid - left + 1;

int n2 = right - mid;

int \*L = (int \*)malloc(n1 \* sizeof(int));

int \*R = (int \*)malloc(n2 \* sizeof(int));

for (i = 0; i < n1; i++)

L[i] = arr[left + i];

for (j = 0; j < n2; j++)

R[j] = arr[mid + 1 + j];

i = 0;

j = 0;

k = left;

while (i < n1 && j < n2)

{

if (L[i] <= R[j])

{

arr[k] = L[i];

i++;

}

else

{

arr[k] = R[j];

j++;

}

k++;

}

while (i < n1)

{

arr[k] = L[i];

i++;

k++;

}

while (j < n2)

{

arr[k] = R[j];

j++;

k++;

}

free(L);

free(R);

}

// Function to implement Merge Sort

void mergeSort(int arr[], int left, int right)

{

if (left < right)

{

int mid = left + (right - left) / 2;

mergeSort(arr, left, mid);

mergeSort(arr, mid + 1, right);

merge(arr, left, mid, right);

}

}

// Function to generate random integers

void generateRandomArray(int arr[], int n)

{

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

arr[i] = rand() % 100000; // Generate random integers between 0 and 99999

}

int main()

{

int n;

struct timeval t;

double s,e;

printf("Enter the number of elements: ");

scanf("%d", &n);

if (n <= 5000)

{

printf("Please enter a value greater than 5000\n");

return 1; // Exit if the number of elements is not greater than 5000

}

int \*arr = (int \*)malloc(n \* sizeof(int));

if (arr == NULL)

{

printf("Memory allocation failed\n");

return 1; // Exit if memory allocation fails

}

generateRandomArray(arr, n);

// Repeat the sorting process multiple times to increase duration for timing

gettimeofday(&t,NULL);

s=t.tv\_sec+(t.tv\_usec/1000000.0);

for (int i = 0; i < 1000; i++)

{

mergeSort(arr, 0, n - 1);

}

gettimeofday(&t,NULL);

e=t.tv\_sec+(t.tv\_usec/1000000.0);

// Calculate and print the time taken to sort the array

printf("\ntime taken to sort %d elements is %lf\n", n, (e-s));

free(arr);

return 0;

}

Sample input and output

Enter number of elements: 6000

Time taken to sort 6000 elements: 0.000709 seconds

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Enter number of elements: 7000

Time taken to sort 7000 elements: 0.000752 seconds

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Enter number of elements: 8000

Time taken to sort 8000 elements: 0.000916 seconds

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Enter number of elements: 9000

Time taken to sort 9000 elements: 0.001493 seconds

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Enter number of elements: 10000

Time taken to sort 10000 elements: 0.001589 seconds

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Enter number of elements: 11000

Time taken to sort 11000 elements: 0.002562 seconds

Design and implement C/C++ Program to 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. Plot a graph of the time taken versus n. The elements can be read from a file or can be generated using the random number generator.

Program

#include<stdio.h>

#include<stdlib.h>

#include<time.h>

// Function to swap two elements

void swap(int\* a, int\* b)

{

int t = \*a;

\*a = \*b;

\*b = t;

}

// Partition function for Quick Sort

intpartition(intarr[], intlow, inthigh)

{

int pivot = arr[high]; // Pivot element

inti = (low - 1); // Index of smaller element

for (int j = low; j <= high - 1; j++)

{

if (arr[j] < pivot)

{

i++; // Increment index of smaller element

swap(&arr[i], &arr[j]);

}

}

swap(&arr[i + 1], &arr[high]);

return (i + 1);

}

// Quick Sort function

void quickSort(intarr[], intlow, inthigh)

{

if (low < high)

{

int pi = partition(arr, low, high);

// Recursively sort elements before and after partition

quickSort(arr, low, pi - 1);

quickSort(arr, pi + 1, high);

}

}

// Function to generate random numbers

void generateRandomNumbers(intarr[], intn)

{

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

{

arr[i] = rand() % 100000; // Generate random numbers between 0 and 99999

}

}

int main()

{

int n;

struct timeval t;

double s,e;

printf("Enter number of elements: ");

scanf("%d", &n); // Read the number of elements from the user

if (n <= 5000)

{

printf("Please enter a value greater than 5000\n");

return 1; // Exit if the number of elements is not greater than 5000

}

// Allocate memory for the array

int \*arr = (int\*)malloc(n \* sizeof(int));

if (arr == NULL)

{

printf("Memory allocation failed\n");

return 1; // Exit if memory allocation fails

}

// Generate random numbers and store them in the array

generateRandomNumbers(arr, n);

// Measure the time taken to sort the array

gettimeofday(&t,NULL);

s=t.tv\_sec+(t.tv\_usec/1000000.0);

quickSort(arr, 0, n - 1);

gettimeofday(&t,NULL);

e=t.tv\_sec+(t.tv\_usec/1000000.0);

// Calculate and print the time taken to sort the array

printf("\ntime taken to sort %d elements is %lf\n", n, (e-s));

// Free the allocated memory

free(arr);

return 0;

}

Sample input and output

Enter number of elements: 10000

Time taken to sort 10000 elements: 0.0000 seconds

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Enter number of elements: 20000

Time taken to sort 20000 elements: 0.015000 seconds

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Enter number of elements: 30000

Time taken to sort 30000 elements: 0.011000 seconds

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Enter number of elements: 35000

Time taken to sort 35000 elements: 0.003000 seconds

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Enter number of elements: 50000

Time taken to sort 50000 elements: 0.015000 seconds

Design and implement C/C++ Program to solve 0/1 Knapsack problem using Dynamic Programming method.

ALGORITHM

//(n items, W weight of sack) Input: n, wi,,, vi and W – all integers

//Output: V(n,W)

// Initialization of first column and first row elements

Repeat for i = 0 to n

set V(i,0) = 0

Repeat for j = 0 to W

Set V(0,j) = 0

//complete remaining entries row by row

Repeat for i = 1 to n

repeat for j = 1 to W

if ( wi <= j ) V(i,j)) = max{ V(i-1,j), V(i-1,j-wi) + vi }

if ( wi > j ) V(i,j) = V(i-1,j)

Print V(n,W)

PROGRAM:

#include<stdio.h>

int w[10],p[10],n;

int max(int a,int b)

{

return a>b?a:b;

}

int knap(int i,int m)

{

if(i==n) return w[i]>m?0:p[i];

if(w[i]>m) return knap(i+1,m);

return max(knap(i+1,m),knap(i+1,m-w[i])+p[i]);

}

int main()

{

int m,i,max\_profit;

printf("\nEnter the no. of objects:");

scanf("%d",&n);

printf("\nEnter the knapsack capacity:");

scanf("%d",&m);

printf("\nEnter profit followed by weight:\n");

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

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

max\_profit=knap(1,m);

printf("\nMax profit=%d",max\_profit);

return 0;

}

Input/Output:

Enter the no. of objects:4

Enter the knapsack capacity:6

Enter profit followed by weight:

78 2

45 3

92 4

71 5

Max profit=170

Design and implement C/C++ Program to solve discrete Knapsack and continuous Knapsack problems using greedy approximation method.

#include <stdio.h>

#define MAX 50

int p[MAX], w[MAX], x[MAX];

double maxprofit;

int n, m, i;

void greedyKnapsack(int n, int w[], int p[], int m) {

double ratio[MAX];

// Calculate the ratio of profit to weight for each item

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

ratio[i] = (double)p[i] / w[i];

}

// Sort items based on the ratio in non-increasing order

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

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

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

double temp = ratio[i];

ratio[i] = ratio[j];

ratio[j] = temp;

int temp2 = w[i];

w[i] = w[j];

w[j] = temp2;

temp2 = p[i];

p[i] = p[j];

p[j] = temp2;

}

}

}

int currentWeight = 0;

maxprofit = 0.0;

// Fill the knapsack with items

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

if (currentWeight + w[i] <= m) {

x[i] = 1; // Item i is selected

currentWeight += w[i];

maxprofit += p[i];

} else {

// Fractional part of item i is selected

x[i] = (m - currentWeight) / (double)w[i];

maxprofit += x[i] \* p[i];

break;

}

}

printf("Optimal solution for greedy method: %.1f\n", maxprofit);

printf("Solution vector for greedy method: ");

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

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

}

int main() {

printf("Enter the number of objects: ");

scanf("%d", &n);

printf("Enter the objects' weights: ");

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

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

printf("Enter the objects' profits: ");

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

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

printf("Enter the maximum capacity: ");

scanf("%d", &m);

greedyKnapsack(n, w, p, m);

return 0;

}

Design and implement C/C++ Program to find Minimum Cost Spanning Tree of a given connected undirected graph using Kruskal's algorithm.

Algorithm: Kruskal(G)

//Kruskal’s algorithm for constructing a minimum spanning tree

//Input: A weighted connected graph G = (V, E)

//Output: ET , the set of edges composing a minimum spanning tree of G sort E in non decreasing order of the edge weights w(ei1) ≤ . . . ≤ w(ei|E|)

ET←∅;

ecounter ←0 //initialize the set of tree edges and its size

k←0 //initialize the number of processed edges

while ecounter < |V| − 1 do

k←k + 1

if ET ∪ {eik} is acyclic

ET←ET∪ {eik};

ecounter ←ecounter + 1

return ET

Program

#include<stdio.h>

#define INF 999

#define MAX 100

int p[MAX],c[MAX][MAX],t[MAX][2];

int find(int v)

{ while(p[v])

v=p[v];

return v;

}

void union1(int i,int j)

{

p[j]=i;

}

void kruskal(int n)

{

int i,j,k,u,v,min,res1,res2,sum=0;

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

{

min=INF;

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

{

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

{

if(i==j)continue;

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

{

u=find(i);

v=find(j);

if(u!=v)

{

res1=i;

res2=j;

min=c[i][j];

}

}

}

}

union1(res1,find(res2));

t[k][1]=res1;

t[k][2]=res2;

sum=sum+min;

}

printf("\nCost of spanning tree is=%d",sum);

printf("\nEdgesof spanning tree are:\n");

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

printf("%d -> %d\n",t[i][1],t[i][2]);

}

int main()

{

int i,j,n;

printf("\nEnter the n value:");

scanf("%d",&n);

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

p[i]=0;

printf("\nEnter the graph data:\n");

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

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

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

kruskal(n);

return 0;

}

Input/Output:

Enter the n value:5

Enter the graph data:

0 10 15 9 999

10 0 999 17 15

15 999 0 20 999

9 17 20 0 18

999 15 999 18 0

Cost of spanning tree is=49

Edgesof spanning tree are:

1 -> 4

1 -> 2

1 -> 3

2 -> 5

3a. Design and implement C/C++ Program to solve All-Pairs Shortest Paths problem using Floyd's algorithm.

Algorithm:

Floyd’s Algorithm

Accept no .of vertices

Call graph function to read weighted graph // w(i,j)

Set D[ ] <- weighted graph matrix // get D {d(i,j)} for k=0

// If there is a cycle in graph, abort. How to find?

Repeat for k = 1 to n

Repeat for i = 1 to n

Repeat for j = 1 to n

D[i,j] = min {D[i,j], D[i,k] + D[k,j]}

Print D

Program

#include<stdio.h>

void main()

{

int a[100][100], i, j, k, n;

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

scanf("%d",&n);

printf("Enter the cost matrix (enter 999 if no edge exists)\n");

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

{

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

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

}

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

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

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

if(a[i][j]>a[i][k]+a[k][j])

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

printf("the shortest path matrix is:\n");

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

{

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

{

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

}

printf(“\n”);

}

}

Input/Output:

Enter the n value:4

Enter the graph data:

0 999 3 999

2 0 999 999

999 7 0 1

6 999 999 0

Shortest path matrix

0 10 3 4

2 0 5 6

7 7 0 1

6 16 9 0

3b. Design and implement C/C++ Program to find the transitive closure using Warshal's algorithm.

Algorithm

//Input: Adjacency matrix of digraph

//Output: R, transitive closure of digraph

Accept no .of vertices

Call graph function to read directed graph

Set R[ ] <- digraph matrix // get R {r(i,j)} for k=0

Print digraph

Repeat for k = 1 to n

Repeat for i = 1 to n

Repeat for j = 1 to n

R(i,j) = 1 if

{rij(k-1) = 1 OR

rik(k-1) = 1 and rkj(k-1) = 1}

Print R

Program

#include<stdio.h>

void main()

{

int a[100][100], i, j, k, n;

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

scanf("%d",&n);

printf("Enter the cost matrix (enter 0 if no edge exists)\n");

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

{

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

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

}

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

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

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

if(a[i][j]||a[i][k]&&a[k][j])

a[i][j]= 1;

printf("the shortest path matrix is:\n");

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

{

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

{

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

}

printf(“\n”);

}

}

Input/Output:

Enter the n value:4

Enter the graph data:

0 1 0 0

0 0 0 1

0 0 0 0

1 0 1 0

Resultant path matrix

1 1 1 1

1 1 1 1

0 0 0 0

1 1 1 1

Design and implement C/C++ Program to find Minimum Cost Spanning Tree of a given connected undirected graph using Prim's algorithm.

Algorithm

Prim(G)

//Prim’s algorithm for constructing a minimum spanning tree

//Input: A weighted connected graph G = (V, E)

//Output: ET , the set of edges composing a minimum spanning tree of G

VT←{v0} //the set of tree vertices can be initialized with any vertex

ET←∅

for i ←1 to |V| − 1 do

find a minimum-weight edge e∗ = (v∗, u∗) among all the edges (v, u) such that v is in VT and u is in V − VT

VT←VT∪ {u∗}

ET←ET∪ {e∗}

return ET

Program

#include <stdio.h>

#define INF 999 // Representing no edge

#define MAX\_VERTICES 11 // Maximum number of vertices

// DFS Function to Check Connectivity

void dfs(int cost[MAX\_VERTICES][MAX\_VERTICES], int n, int visited[], int node) {

visited[node] = 1;

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

if (cost[node][i] != INF && !visited[i]) { // If connected and not visited

dfs(cost, n, visited, i);

}

}

}

// Function to Check if the Graph is Connected

int isConnected(int cost[MAX\_VERTICES][MAX\_VERTICES], int n) {

int visited[MAX\_VERTICES] = {0}; // All nodes initially unvisited

dfs(cost, n, visited, 1); // Start DFS from vertex 1

// Check if all nodes were visited

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

if (!visited[i]) {

return 0; // Graph is disconnected

}

}

return 1; // Graph is connected

}

// Prim’s Algorithm to find MST

void prims(int cost[MAX\_VERTICES][MAX\_VERTICES], int n) {

int selected[MAX\_VERTICES] = {0}; // Track selected vertices

int edges = 0, min, u, v, totalCost = 0;

selected[1] = 1; // Start with first vertex (1)

printf("\nThe Minimum Spanning Tree (MST) edges are:\n");

while (edges < n - 1) {

min = INF;

u = v = -1;

// Find the minimum weight edge

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

if (selected[i]) { // Only consider already selected vertices

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

if (!selected[j] && cost[i][j] < min) { // Edge to unvisited vertex

min = cost[i][j];

u = i;

v = j;

}

}

}

}

// If no valid edge is found, break (Graph is disconnected)

if (u == -1 || v == -1) {

break;

}

// Select the new vertex and print the selected edge

selected[v] = 1;

printf("%d -> %d with cost %d\n", u, v, min);

totalCost += min;

edges++;

}

// Check if MST was formed correctly

if (edges != n - 1) {

printf("\n Spanning Tree does NOT exist (Graph is disconnected) \n");

} else {

printf("\n Total Cost of Minimum Spanning Tree: %d\n", totalCost);

}

}

int main() {

int cost[MAX\_VERTICES][MAX\_VERTICES], n;

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

scanf("%d", &n);

printf("\nEnter the cost adjacency matrix (use 0 for no edge):\n");

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

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

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

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

cost[i][j] = INF; // Replace 0 with INF except diagonal

}

}

// Check if graph is connected before running Prim’s Algorithm

if (!isConnected(cost, n)) {

printf("\n Spanning Tree does NOT exist (Graph is disconnected) \n");

return 0; // Exit early

}

prims(cost, n); // Run Prim’s algorithm

return 0;

}

OR

#include<stdio.h>

// #include<conio.h>

#define INF 999

int prim(int c[10][10],int n,int s)

{

int v[10],i,j,sum=0,ver[10],d[10],min,u;

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

{

ver[i]=s;

d[i]=c[s][i];

v[i]=0;

}

v[s]=1;

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

{

min=INF;

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

if(v[j]==0 && d[j]<min)

{

min=d[j];

u=j;

}

v[u]=1;

sum=sum+d[u];

printf("\n%d -> %d sum=%d",ver[u],u,sum);

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

if(v[j]==0 && c[u][j]<d[j])

{

d[j]=c[u][j];

ver[j]=u;

}

}

return sum;

}

void main()

{

int c[10][10],i,j,res,s,n;

clrscr();

printf("\nEnter n value:");

scanf("%d",&n);

printf("\nEnter the graph data:\n");

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

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

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

printf("\nEnter the souce node:");

scanf("%d",&s);

res=prim(c,n,s);

printf("\nCost=%d",res);

getch();

}

Input/output:

Enter n value:3

Enter the graph data:

0 10 1

10 0 6

1 6 0

Enter the souce node:1

1 -> 3 sum=1

3 -> 2 sum=7

Cost=7

Design and implement C/C++ Program to find shortest paths from a given vertex in a weighted connected graph to other vertices using Dijkstra's algorithm.

Algorithm: Dijikstra(G,s)

//Dijikstra’s algorithm for single source shortest path

//input:A weighted connected graph with non negative weights and its vertex s

//output:The length dv of a shortest path from s to v and penultimate vertex pv for every vertex v in V

Initialize(Q)

for every vertex v in V do

dv<-∞;Pv<-null

Insert(Q,v,dv)

Ds<-0; Decrease(Q,s,ds);VT<-ǿ

for i<- 0 to │V│-1 do

u\*<-DeleteMin(Q)

VT<-VT U{u\*}

For every vertex u in V-VT that is adjacent to u\* do

If du\*+w(u\*,u)<du

du<- du\*+w(u\*,u); pu<-u\*

Decrease(Q,u,du)

Program:

#include<stdio.h>

#define INF 999

void dijkstra(int c[10][10],int n,int s,int d[10])

{

int v[10],min,u,i,j;

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

{

d[i]=c[s][i];

v[i]=0;

}

v[s]=1;

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

{

min=INF;

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

if(v[j]==0 && d[j]<min)

{

min=d[j];

u=j;

}

v[u]=1;

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

if(v[j]==0 && (d[u]+c[u][j])<d[j])

d[j]=d[u]+c[u][j];

}

}

int main()

{

int c[10][10],d[10],i,j,s,sum,n;

printf("\nEnter n value:");

scanf("%d",&n);

printf("\nEnter the graph data:\n");

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

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

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

printf("\nEnter the souce node:");

scanf("%d",&s);

dijkstra(c,n,s,d);

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

printf("\nShortest distance from %d to %d is %d",s,i,d[i]);

return 0;

}

Input/Output

Enter n value:6

Enter the graph data:

0 15 10 999 45 999

999 0 15 999 20 999

20 999 0 20 999 999

999 10 999 0 35 999

999 999 999 30 0 999

999 999 999 4 999 0

Enter the souce node:2

Shortest distance from 2 to 1 is 35

Shortest distance from 2 to 2 is 0

Shortest distance from 2 to 3 is 15

Shortest distance from 2 to 4 is 35

Shortest distance from 2 to 5 is 20

Shortest distance from 2 to 6 is 999

Output:

enter the no. of nodes:

6

enter the cost adjacency matrix,'9999' for no direct path

0 15 10 9999 45 9999

9999 0 15 9999 20 9999

20 9999 0 20 9999 9999

9999 10 9999 0 35 9999

9999 9999 9999 30 0 9999

9999 9999 9999 4 9999 0

enter the starting vertex:

6

Shortest path from starting vertex to other vertices are

6->1=49

6->2=14

6->3=29

6->4=4

6->5=34

6->6=0

Design and implement C/C++ Program to find a subset of a given set S = {sl , s2,.....,sn} of n positive integers whose sum is equal to a given positive integer d.

Algorithm: SumOfSub (s, k, r)

//Values of x[ j ], 1 <= j < k, have been determined

//Node creation at level k taking place: also call for creation at level K+1 if possible

// s = sum of 1 to k-1 elements and r is sum of k to n elements

//generating left child that means including k in solution

Set x[k] = 1

If (s + s[k] = d) then subset found, print solution

If (s + s[k] + s[k+1] <=d)

then SumOfSum (s + s[k], k+1, r – s[k])

//Generate right child i.e. element k absent

If (s + r - s[k] >=d) AND (s + s[k+1] )<=d

THEN { x[k]=0;

SumOfSub(s, k+1, r – s[k])

Program:

#include<stdio.h>

// #include<conio.h>

#define MAX 10

int s[MAX],x[MAX],d;

void sumofsub(int p,int k,int r)

{

int i;

x[k]=1;

if((p+s[k])==d)

{

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

if(x[i]==1)

printf("%d ",s[i]);

printf("\n");

}

else

if(p+s[k]+s[k+1]<=d)

sumofsub(p+s[k],k+1,r-s[k]);

if((p+r-s[k]>=d) && (p+s[k+1]<=d))

{

x[k]=0;

sumofsub(p,k+1,r-s[k]);

}

}

int main()

{

int i,n,sum=0;

printf("\nEnter the n value:");

scanf("%d",&n);

printf("\nEnter the set in increasing order:");

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

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

printf("\nEnter the max subset value:");

scanf("%d",&d);

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

sum=sum+s[i];

if(sum<d || s[1]>d)

printf("\nNo subset possible");

else

sumofsub(0,1,sum);

return 0;

}

Input/output:

Enter the n value:9

Enter the set in increasing order:1 2 3 4 5 6 7 8 9

Enter the max subset value:9

1 2 6

1 3 5

1 8

2 3 4

2 7

3 6

4 5

9

Design and implement C/C++ Program for N Queen's problem using Backtracking

Algorithm:

/\* outputs all possible acceptable positions of n queens on n x n chessboard \*/

// Initialize x [ ] to zero

// Set k = 1 start with first queen

Repeat for i = 1 to n // try all columns one by one for kth queen

if Place (k, i) true then

{

x(k) = i // place kth queen in column i

if (k=n) all queens placed and hence print output (x[ ])

else NQueens(K+1,n) //try for next queen

}

Place (k,i)

/\* finds if kth queen in kth row can be placed in column i or not; returns true if queen can be placed \*/

// x[1,2, . . . k-1] have been defined

//queens at (p, q) & (r, s) attack if |p-r| = |q-s|

Repeat for j = 1 to (k-1)

if any earlier jth queen is in ith column ( x[j]= i)

or in same diagonal ( abs(x[ j] - i) = abs( j - k) )

then kth queen cannot be placed (return false)

return true (as all positions checked and no objection)

Program:

#include<stdio.h>

void queens(int);

void PrintBoard(int);

int place(int ,int[]);

int x[20],soln=1;

int main()

{

int n;

printf("Enter the no. of queens to place : ");

scanf("%d",&n);

queens(n);

return 0;

}

void queens(int n)

{

int i,k=1;

x[k]=0;

while(k!=0)

{

x[k]=x[k]+1;

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

x[k]=x[k]+1;

if(x[k]<=n)

if(k==n)

PrintBoard(n);

else

k++,x[k]=0;

else

k--;

}

printf("\nThe number of possible solutions are %d",soln-1);

}

int place(int k,int x[])

{

int i;

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

if((x[i]==x[k]) || (i-x[i]==k-x[k]) || (i+x[i]==k+x[k]))

return 0;

return 1;

}

void PrintBoard(int n)

{

int i,j;

printf("\n Solution set %d: ",soln);

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

printf(" %d ",x[i]);

printf("\n\n Placements are as shown below: \n\n");

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

{

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

(j==x[i]) ?printf(" Q%d",i) :printf(" -- ");

printf("\n");

}

soln++;

}

INPUT & OUTPUT

Enter the no. of queens to place : 4

Solution set 1: 2 4 1 3

Placements are as shown below:

-- Q1 -- --

-- -- -- Q2

Q3 -- -- --

-- -- Q4 --

Solution set 2: 3 1 4 2

Placements are as shown below:

-- -- Q1 --

Q2 -- -- --

-- -- -- Q3

-- Q4 -- --

The number of possible solutions are 2