**LAB MANUAL**

**Lab Name :**  Analysis of Algorithms Lab

**Lab Code :** 5CS4-23

**Branch :** Computer Science Engineering

**Year :** 3rd Year (2021-22)

****

Department of Computer Science Engineering

**Jaipur Engineering College and Research Centre, Jaipur**

(Affiliated to RTU, Kota)

**INDEX**

|  |  |  |  |
| --- | --- | --- | --- |
| **S. No.** | **Contents** | | **Page No.** |
| 1 | Vision and Mission of the Institute | | iv |
| 2 | Vision and Mission of the Department | | v |
| 3 | Program Educational Objectives (PEOs) | | vi |
| 4 | Program Outcomes (POs) | | vii |
| 5 | PSO of the Department (PSOs) | | viii |
| 6 | RTU Syllabus with List of Experiments | | ix-x |
| 7 | Course Outcomes (COs) | | xi-xii |
| 8 | CO/PO-PSO mapping | | xiii |
| 9 | Introduction about Lab &its Applications | | xiv |
| 10 | Instructions Sheet | | xv |
| **Experiment List (As per RTU, Kota Syllabus)** | | | |
| Experiment 1 | | Sort a given set of elements using the Quicksort method and determine the time required to sort the elements. Repeat the experiment for different values of n, the number of elements in the list to be sorted and 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. |  |
| Experiment 2 | | Implement a parallelized Merge Sort algorithm to sort a given set of elements and determine the time required to sort the elements. Repeat the experiment for different values of n, the number of elements in the list to be sorted and 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. |  |
| Experiment 3 | | a. Obtain the Topological ordering of vertices in a given digraph. b. Compute the transitive closure of a given directed graph using Warshall's algorithm. |  |
| Experiment 4 | | Implement Fractional Knapsack problem Greedy Approach. |  |
| Experiment 5 | | Implement 0/1 Knapsack problem using Dynamic Programming. |  |
| Experiment 6 | | From a given vertex in a weighted connected graph, find shortest paths to other vertices using Dijkstra's algorithm. |  |
| Experiment 7 | | Find Minimum Cost Spanning Tree of a given undirected graph using Kruskal's algorithm. |  |
| Experiment 8 | | a. Print all the nodes reachable from a given starting node in a digraph using BFS method. b. Check whether a given graph is connected or not using DFS method. |  |
| Experiment 9 | | Find Minimum Cost Spanning Tree of a given undirected graph using Prim’s algorithm. |  |
| Experiment 10 | | Implement All-Pairs Shortest Paths Problem using Floyd's algorithm. |  |
| Experiment 11 | | Implement N Queen's problem using Back Tracking. |  |
| Experiment 12 | | Implement Bubble Sort algorithm to sort a given set of elements and determine the time required to sort the elements. The elements can be read from a file or can be generated using the random number generator. |  |
| Experiment 13 | | Implement Linear Search algorithm to search a value from a given set of elements. The elements can be read from a file or can be generated using the random number generator. |  |
| Experiment 14 | | Implement Binary Search algorithm to search a value from a given set of elements. The elements can be read from a file or can be generated using the random number generator. |  |
| Experiment 15 | | Compute two 2 by 2 matrix multiplication to find multiplication matrix using Strassen’s Matrix Multiplication. |  |
| Experiment 16 | | Implement Longest Common Subsequence problem using Dynamic Programming. |  |
| Experiment 17 | | Implement Matrix Chain Multiplication using Dynamic Programming. |  |
| **Content Beyond Syllabus** | | | |
| Experiment 18 | | Implement Graph Coloring Problem using backtracking. |  |
| Experiment 19 | | Implement Sum of Subset Problem using backtracking. |  |

**Vision of the Institute**

To become a renowned centre of outcome based learning, and work towards academic, professional, cultural and social enrichment of the lives of individuals and communities.

**Mission of the Institute**

M1: Focus on evaluation of learning outcomes and motivate students to inculcate research aptitude by project based learning.

M2: Identify, based on informed perception of Indian, regional and global needs, areas of focus and provide platform to gain knowledge and solutions.

M3: Offer opportunities for interaction between academia and industry.

M4: Develop human potential to its fullest extent so that intellectually capable and imaginatively gifted leaders can emerge in a range of professions.

**Vision of the Department**

To become renowned Centre of excellence in Computer Science and Engineering and make competent engineers & professionals with high ethical values prepared for lifelong learning.

**Mission of the Department**

**M1:** To impart outcome based education for emerging technologies in the field of Computer Science and Engineering.

**M2:** To provide opportunities for interaction between academia and industry.

**M3:** To provide platform for lifelong learning by accepting the change in technologies

**M4:** To develop aptitude of fulfilling social responsibilities

**PROGRAM EDUCATIONAL OBJECTIVES (PEOs)**

**PEO1:** To provide students with the fundamentals of Engineering Sciences with more emphasis in Computer Science & Engineering by way of analysing and exploiting engineering challenges.

**PEO2:** To train students with good scientific and engineering knowledge so as to comprehend, analyse, design, and create novel products and solutions for the real life problems in Computer Science and Engineering

**PEO3:** To inculcate professional and ethical attitude, effective communication skills, teamwork skills, multidisciplinary approach, entrepreneurial thinking and an ability to relate engineering issues with social issues for Computer Science & Engineering.

**PEO4:** To provide students with an academic environment aware of excellence, leadership, written ethical codes and guidelines, and the self-motivated life-long learning needed for a successful professional career in Computer Science & Engineering.

**PEO5:** To prepare students to excel in Industry and Higher education by Educating Students along with High moral values and Knowledge in Computer Science & Engineering.

**PROGRAM OUTCOMES (POs)**

1. Engineering knowledge: Apply the knowledge of mathematics, science, engineering fundamentals, and Computer Science & Engineering specialization to the solution of complex Computer Science & Engineering problems.

2. Problem analysis: Identify, formulate, research literature, and analyse complex Computer Science and Engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.

3. Design/development of solutions: Design solutions for complex Computer Science and Engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations.

4. Conduct investigations of complex problems: Use research-based knowledge and research methods including design of Computer Science and Engineering experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.

5. Modern tool usage: Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modelling to complex Computer Science Engineering activities with an understanding of the limitations.

6. The engineer and society: Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional Computer Science and Engineering practice.

7. Environment and sustainability: Understand the impact of the professional Computer Science and Engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.

8. Ethics: Apply ethical principles and commit to professional ethics and responsibilities and norms of the Computer Science and Engineering practice.

9. Individual and team work: Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings in Computer Science and Engineering.

10. Communication: Communicate effectively on complex Computer Science and Engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.

11. Project management and finance: Demonstrate knowledge and understanding of the Computer Science and Engineering and management principles and apply these to one’s own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.

12. Life-long learning: Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change in Computer Science and Engineering.

**PROGRAM SPECIFIC OUTCOMES (PSOs)**

**PSO1**. Ability to interpret and analyze network specific and cyber security issues, automation in real word environment.

**PSO2**. Ability to Design and Develop Mobile and Web-based applications under realistic constraints.

**RTU Syllabus with List of Experiments**

**5CS4-23: DESIGN AND ANALYSIS OF ALGORITHMS LAB**

|  |  |
| --- | --- |
| **Class: V Sem. B.Tech.** | **Evaluation** |
| **Branch: Computer Science & Engineering** | **Examination Time = Two (2) Hours** |
| **Schedule (Per Week Practical Hrs.): Two (2)** | **Maximum Marks = 50** |
|  | **[Sessional /Mid-term (30) & End-term (20)** |

5CS4-23: Analysis of Algorithms Lab

Credit: 1 Max. Marks:50 (IA:30, ETE:20)

0L+0T+2P End Term Exam: 2 Hours

|  |  |
| --- | --- |
| **SN** | **List of Experiments** |
| **1** | Sort a given set of elements using the Quicksort method and determine the time required to sort the elements. Repeat the experiment for different values of n, the number of elements in the list to be sorted and 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. |
| **2** | Implement a parallelized Merge Sort algorithm to sort a given set of elements and determine the time required to sort the elements. Repeat the experiment for different values of n, the number of elements in the list to be sorted and 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. |
| **3** | a. Obtain the Topological ordering of vertices in a given digraph. b. Compute the transitive closure of a given directed graph using Warshall's algorithm. |
| **4** | Implement 0/1 Knapsack problem using Dynamic Programming. |
| **5** | From a given vertex in a weighted connected graph, find shortest paths to other vertices using Dijkstra's algorithm. |
| **6** | Find Minimum Cost Spanning Tree of a given undirected graph using Kruskal's algorithm. |
| **7** | a. Print all the nodes reachable from a given starting node in a digraph using BFS method. b. Check whether a given graph is connected or not using DFS method. |
| **8.** | Find Minimum Cost Spanning Tree of a given undirected graph using Prim’s algorithm. |
| **9.** | Implement All-Pairs Shortest Paths Problem using Floyd's algorithm. |
| **10.** | Implement N Queen's problem using Back Tracking. |

**Course Outcomes**

Graduates would be able:

CO1. To implement various searching and sorting techniques on linear/nonlinear data structures to solve various computing problems.

CO2. Design and implement efficient algorithms for a specified application.

CO3. Strengthen the ability to identify and apply the suitable algorithms for the given real problem.

**Mapping of Experiments with Cos & BT Level**

|  |  |  |  |
| --- | --- | --- | --- |
| **SN** | **List of Experiments** | COs | BT Level |
| **1** | Sort a given set of elements using the Quicksort method and determine the time required to sort the elements. Repeat the experiment for different values of n, the number of elements in the list to be sorted and 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. | CO1 | BT3 |
| **2** | Implement a parallelized Merge Sort algorithm to sort a given set of elements and determine the time required to sort the elements. Repeat the experiment for different values of n, the number of elements in the list to be sorted and 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. | CO1 | BT3 |
| **3** | a. Obtain the Topological ordering of vertices in a given digraph. b. Compute the transitive closure of a given directed graph using Warshall's algorithm. | CO1 | BT3 |
| **4** | Implement Fractional Knapsack problem using Greedy Method. | CO2 | BT3 |
| **5** | Implement 0/1 Knapsack problem using Dynamic Programming. | CO2 | BT3 |
| **6** | From a given vertex in a weighted connected graph, find shortest paths to other vertices using Dijkstra's algorithm. | CO2 | BT3 |
| **7** | Find Minimum Cost Spanning Tree of a given undirected graph using Kruskal's algorithm. | CO2 | BT3 |
| **8** | a. Print all the nodes reachable from a given starting node in a digraph using BFS method. b. Check whether a given graph is connected or not using DFS method. | CO3 | BT3 |
|  |
| **9** | Find Minimum Cost Spanning Tree of a given undirected graph using Prim’s algorithm. | CO3 | BT3 |
| **10** | Implement All-Pairs Shortest Paths Problem using Floyd's algorithm. | CO3 | BT3 |
| **11** | Implement N Queen's problem using Back Tracking. | CO3 | BT3 |

\* BT - Bloom's Taxonomy

**Mapping of Course Outcomes & POs/PSOs**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | **Engineering Knowledge** | **Problem analysis** | **Design/Development of Solution** | **Conduct Invest. of complex problems** | **Modern Tool Usage** | **The engineer and society** | **Environment and Sustainability** | **Ethics** | **Individual and Team Work** | **Communication** | **Project Management and Finance** | **Life-long Learning** | **Concepts of Vehicle Mechanism** | **3-D printing Technology** |
|  | **PO1** | **PO2** | **PO3** | **PO4** | **PO5** | **PO6** | **PO7** | **PO8** | **PO9** | **PO 10** | **PO 11** | **PO 12** | **PSO1** | **PSO2** |
| CO-1 | 3 | 3 | 3 | 3 | - | - | - | - | - | - | - | 2 | 1 | 2 |
| CO-2 | 3 | 3 | 3 | 3 | 3 | 1 | - | - | - | - | - | 2 | 2 | 2 |
| CO-3 | 3 | 3 | 3 | 3 | 3 | 1 | - | - | - | - | - | 2 | 2 | 2 |

**INTRODUCTION ABOUT LABORATORY& APPLICATIONS**

This laboratory is focused towards reinforcing the learning of students in the field of developing and analyzing algorithms and applying those learning in practical environments. The experiments in laboratory cover concepts on the developing and analyzing algorithms including, but not limited to, Identify and evaluate complex problems using principles of mathematics and engineering science.

Experiments performed in this course will provide students a better understanding about

Solve problems by applying appropriate algorithms.

• Analyze the efficiency of various algorithms.

• Apply techniques of stacks and queues to solve problems.

• Develop a program that can be solved in many ways using different techniques.

• Identify and evaluate complex problems using principles of mathematics and engineering science.

The understanding of Identify and evaluate complex problems developed in this laboratory, along-with theoretical concepts, will help students in design and analysis of algorithms. This will enhance the employability of students in Software development, testing and research industry.

**INSTRUCTIONS SHEET**

**We need your full support and cooperation for smooth functioning of the lab.**

**DO’s**

* Please switch off the Mobile/Cell phone before entering Lab.
* Enter the Lab with complete source code and data.
* Check whether all peripheral are available at desktop before proceeding for program.
* Intimate the Lab in Charge whenever you are incompatible in using the system or in case software get corrupted/ infected by virus.
* Arrange all the peripheral and seats before leaving the lab.
* Properly shutdown the system before leaving the lab.
* Keep the bag outside in the racks.
* Enter the lab on time and leave at proper time.
* Maintain the decorum of the lab.
* Utilize lab hours in the corresponding experiment.
* Get your Cd / Pen drive checked by Lab in Charge before using it in the lab.

**DON’Ts**

* Don’t mishandle the system.
* Don’t leave the system on standing for long.
* Don’t bring any external material in the lab.
* Don’t make noise in the lab.
* Don’t bring the mobile in the lab. If extremely necessary then keep ringers off.
* Don’t enter in the lab without permission of Lab in Charge.
* Don’t litter in the lab.
* Don’t delete or make any modification in system files.
* Don’t carry any lab equipment outside the lab.

**BEFORE ENTERING IN THE LAB**

* All the students are supposed to prepare the theory regarding the next program.
* Students are supposed to bring the practical file and the lab copy.
* Previous programs should be written in the practical file.
* Any student not following these instructions will be denied entry in the lab.

**WHILE WORKING IN THE LAB**

* Adhere to experimental schedule as instructed by the Lab in Charge.
* Get the previously executed program signed by the instructor.
* Get the output of the current program checked by the instructor in the lab copy.
* Each student should work on his/her assigned computer at each turn of the lab.
* Take responsibility of valuable accessories.
* Concentrate on the assigned practical and do not play games.
* If anyone caught red handed carrying any equipment of the lab, then he will have to face serious consequences.

**Experiment No. 1**

**AIM:**

**Experiment-1**

**Aim** - **Sort a given set of elements using the Quick Sort method and determine the time required to sort the elements. The elements can be read from a file or can be generated using the random number generator.**

/\* program to sort elements of an array using Quick Sort \*/

#include<stdio.h>

#include<conio.h>

void quicksort(int[ ],int,int);

void main( )

{

int low, high, pivot, t, n, i, j, a[10];

clrscr( );

printf("\nHow many elements you want to sort ? ");

scanf("%d",&n);

printf("\Enter elements for an array:");

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

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

low=0;

high=n-1;

quicksort(a,low,high);

printf("\After Sorting the elements are:");

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

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

getch();

}

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

{

int pivot,t,i,j;

if(low<high)

{

pivot=a[low];

i=low+1;

j=high;

while(1)

{

while(pivot>a[i]&&i<=high)

i++;

while(pivot<a[j]&&j>=low)

j--;

if(i<j)

{

t=a[i];

a[i]=a[j];

a[j]=t;

}

else

break;

}

a[low]=a[j];

a[j]=pivot;

quicksort(a,low,j-1);

quicksort(a,j+1,high);

}

}

**Experiment – 2**

**Aim** - **Implement Merge Sort algorithm to sort a given set of elements and determine the time required to sort the elements. The elements can be read from a file or can be generated using the random number generator.**

#include<stdio.h>

#include<conio.h>

void disp();

void mergesort(int,int,int);

void msortdiv(int,int);

int a[50],n;

void main()

{

int i;

clrscr();

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

scanf("%d",&n);

printf("\nEnter elements for an array:");

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

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

printf("\nBefore Sorting the elements are:");

disp();

msortdiv(0,n-1);

printf("\nAfter Sorting the elements are:");

disp();

getch( );

}

void disp( )

{

int i;

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

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

}

void mergesort(int low,int mid,int high)

{

int t[50],i,j,k;

i=low;

j=mid+1;

k=low;

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

{

if(a[i]>=a[j])

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

else

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

}

while(i<=mid)

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

while(j<=high)

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

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

a[i]=t[i];

}

void msortdiv(int low,int high)

{

int mid;

if(low!=high)

{

mid=((low+high)/2);

msortdiv(low,mid);

msortdiv(mid+1,high);

mergesort(low,mid,high);

}

}

**Experiment – 3**

**a. Obtain the Topological ordering of vertices in a given digraph.**

**b. Compute the transitive closure of a given directed graph using Warshall's algorithm.**

**a. Obtain the Topological ordering of vertices in a given digraph.**

#include <stdio.h>

int main(){

int i,j,k,n,a[10][10],indeg[10],flag[10],count=0;

printf("Enter the no of vertices:\n");

scanf("%d",&n);

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

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

printf("Enter row %d\n",i+1);

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

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

}

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

        indeg[i]=0;

        flag[i]=0;

    }

    for(i=0;i<n;i++)

        for(j=0;j<n;j++)

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

    printf("\nThe topological order is:");

    while(count<n){

        for(k=0;k<n;k++){

            if((indeg[k]==0) && (flag[k]==0)){

                printf("%d ",(k+1));

                flag [k]=1;

            }

            for(i=0;i<n;i++){

                if(a[i][k]==1)

                    indeg[k]--;

            }

        }

        count++;

    }

    return 0;

}

**b. Compute the transitive closure of a given directed graph using Warshall's algorithm.**

//Compute the transitive closure of a given directed graph using Warshall's algorithm.  
# include <stdio.h>  
# include <conio.h>  
int n,a[10][10],p[10][10];  
void path()  
{  
int i,j,k;  
for(i=0;i<n;i++)  
for(j=0;j<n;j++)  
p[i][j]=a[i][j];  
for(k=0;k<n;k++)  
for(i=0;i<n;i++)  
for(j=0;j<n;j++)  
if(p[i][k]==1&&p[k][j]==1) p[i][j]=1;  
}  
void main()  
{  
int i,j;  
clrscr();  
printf("Enter the number of nodes:");  
scanf("%d",&n);  
printf("\nEnter the adjacency matrix:\n");  
for(i=0;i<n;i++)  
for(j=0;j<n;j++)  
scanf("%d",&a[i][j]);  
path();  
printf("\nThe path matrix is showm below\n");  
for(i=0;i<n;i++)  
{  
for(j=0;j<n;j++)  
printf("%d ",p[i][j]);  
printf("\n");  
}  
getch();  
}

**Experiment – 4**

**Aim - Implement Knapsack problem using Greedy Method.**

# include<stdio.h>

#include<conio.h>

void knapsack(int n, float weight[], float profit[], float capacity)

{

float x[20], tp = 0;

int i, j, u;

u = capacity;

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

x[i] = 0.0;

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

{

if(weight[i] > u)

break;

else

{

x[i] = 1.0;

tp = tp + profit[i];

u = u - weight[i];

}

}

if(i < n)

x[i] = u / weight[i];

tp = tp + (x[i] \* profit[i]);

printf("\nThe result vector is:- ");

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

printf("%.2f\t", x[i]);

printf("\nMaximum profit is:- %.2f", tp);

}

void main()

{

float weight[20], profit[20], capacity;

int num, i, j;

float ratio[20], temp;

clrscr();

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

scanf("%d", &num);

printf("\nEnter the profits and wts of %d object ",num);

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

{

printf("\nEnter the profits and wts of %d object",i+1);

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

}

printf("\nEnter the capacityacity of knapsack:- ");

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;

}

}

  }

   knapsack(num, weight, profit, capacity);

   getch();

}

**Experiment – 5**

**Aim - Implement 0/1 Knapsack problem using Dynamic Programming.**

#include <conio.h>

#include <stdio.h>

void knapsack01(int v[],int w[],int n,int capacity)

{

int x=0,b[100][100],i=0;

for(x=0;x<=capacity;x++)

b[0][x]=0;

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

b[i][0]=0;

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

{

for(x=1;x<=capacity;x++)

{

if(w[i]<=x)

{

if( (v[i]+b[i-1][x-w[i]] ) > b[i-1][x] )

b[i][x]= (v[i]+b[i-1][x-w[i]] ) ;

else

b[i][x]=b[i-1][x];

}

else

b[i][x]=b[i-1][x];

}

}

printf("the matix is\n");

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

{

for(x=0;x<=capacity;x++)

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

printf("\n");

}

printf("the benifits is \n%d",b[n][capacity]);

printf("\nthe selected items is\n");

i=n,x=capacity;

while(i>0 && x>0)

{

if(b[i][x]!=b[i-1][x])

{

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

b[i][x]=b[i-1][x-w[i]];

x=x-w[i];

i=i-1;

}

else

{

b[i][x]=b[i-1][x];

i=i-1;

}

}

}

void main()

{

int i,n,capacity,v[100],w[100];

clrscr();

printf("enter the total capacity of knapsack \n");

scanf("%d",&capacity);

printf("enter the no of items\n");

scanf("%d",&n);

//value of each item's

printf("enter the weight and value  of items\n");

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

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

knapsack01(v,w,n,capacity);

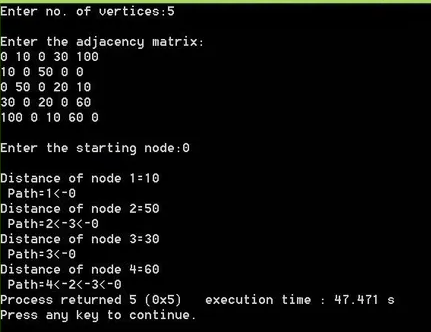
getch();

}

**Experiment – 6**

**From a given vertex in a weighted connected graph, find shortest paths to other vertices using Dijkstra's algorithm.**

|  |
| --- |
| #include<stdio.h>  #include<conio.h>  #define INFINITY 9999  #define MAX 10    void dijkstra(int G[MAX][MAX],int n,int startnode);    int main()  {  int G[MAX][MAX],i,j,n,u;  printf("Enter no. of vertices:");  scanf("%d",&n);  printf("\nEnter the adjacency matrix:\n");  for(i=0;i<n;i++)  for(j=0;j<n;j++)  scanf("%d",&G[i][j]);  printf("\nEnter the starting node:");  scanf("%d",&u);  dijkstra(G,n,u);  return 0;  }    void dijkstra(int G[MAX][MAX],int n,int startnode)  {    int cost[MAX][MAX],distance[MAX],pred[MAX];  int visited[MAX],count,mindistance,nextnode,i,j;  //pred[] stores the predecessor of each node  //count gives the number of nodes seen so far  //create the cost matrix  for(i=0;i<n;i++)  for(j=0;j<n;j++)  if(G[i][j]==0)  cost[i][j]=INFINITY;  else  cost[i][j]=G[i][j];  //initialize pred[],distance[] and visited[]  for(i=0;i<n;i++)  {  distance[i]=cost[startnode][i];  pred[i]=startnode;  visited[i]=0;  }  distance[startnode]=0;  visited[startnode]=1;  count=1;  while(count<n-1)  {  mindistance=INFINITY;  //nextnode gives the node at minimum distance  for(i=0;i<n;i++)  if(distance[i]<mindistance&&!visited[i])  {  mindistance=distance[i];  nextnode=i;  }  //check if a better path exists through nextnode  visited[nextnode]=1;  for(i=0;i<n;i++)  if(!visited[i])  if(mindistance+cost[nextnode][i]<distance[i])  {  distance[i]=mindistance+cost[nextnode][i];  pred[i]=nextnode;  }  count++;  }    //print the path and distance of each node  for(i=0;i<n;i++)  if(i!=startnode)  {  printf("\nDistance of node%d=%d",i,distance[i]);  printf("\nPath=%d",i);  j=i;  do  {  j=pred[j];  printf("<-%d",j);  }while(j!=startnode);  }  } |

****

**Experiment – 7**

**Aim - Find Minimum Cost Spanning Tree of a given undirected graph using Kruskal's algorithm**.

#include<stdio.h>

#include<conio.h>

#include<stdlib.h>

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

int min,mincost=0,cost[9][9],parent[9];

int find(int);

int uni(int,int);

void main()

{

clrscr();

printf("\n\n\tImplementation of Kruskal's algorithm\n\n");

printf("\nEnter the no. of vertices\n");

scanf("%d",&n);

printf("\nEnter the cost 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;

}

}

printf("\nThe edges of Minimum Cost Spanning Tree are\n\n");

while(ne<n)

{

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

{

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

{

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

{

min=cost[i][j];

a=u=i;

b=v=j;

}

}

}

printf("The min value is %d",min);

u=find(u);

v=find(v);

if(uni(u,v))

{

printf("\n%d edge (%d,%d) =%d\n",ne++,a,b,min);

mincost +=min;

}

cost[a][b]=cost[b][a]=999;    //remove the minimum edge

}

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

getch();

}

int find(int i)

{

while(parent[i])

i=parent[i];

return i;

}

int uni(int i,int j)     //for cycles

{

if(i!=j)

{

parent[j]=i;

return 1;

}

return 0;

}

**Experiment – 8**

**a. Print all the nodes reachable from a given starting node in a digraph using BFS method.**

**b. Check whether a given graph is connected or not using DFS method.**

**a. Print all the nodes reachable from a given starting node in a digraph using BFS method.**

#include<stdio.h>

#include<conio.h>

int a[20][20],q[20],visited[20],n,i,j,f=0,r=-1;

void bfs(int v)

{

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

if(a[v][i] && !visited[i])

q[++r]=i;

if(f<=r)

{

visited[q[f]]=1;

bfs(q[f++]);

}

}

void main()

{

int v;

clrscr();

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

scanf("%d",&n);

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

{

q[i]=0;

visited[i]=0;

}

printf("\n Enter graph data in matrix form:\n");

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

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

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

printf("\n Enter the starting vertex:");

scanf("%d",&v);

bfs(v);

printf("\n The node which are reachable are:\n");

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

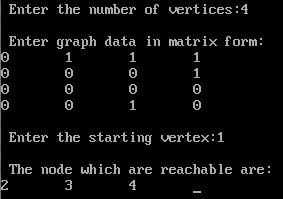
if(visited[i])

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

getch();

}

Output:



**b. Check whether a given graph is connected or not using DFS method.**

#include<stdio.h>

#include<conio.h>

int a[20][20],reach[20],n;

void dfs(int v)

{

int i;

reach[v]=1;

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

if(a[v][i] && !reach[i])

{

printf("\n %d->%d",v,i);

dfs(i);

}

}

void main()

{

int i,j,count=0;

clrscr();

printf("\n Enter number of vertices:");

scanf("%d",&n);

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

{

reach[i]=0;

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

a[i][j]=0;

}

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

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

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

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

dfs(1);

printf("\n");

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

{

if(reach[i])

count++;

}

if(count==n)

printf("\n Graph is connected");

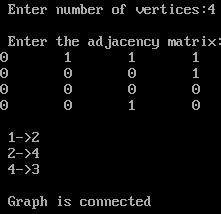
else

printf("\n Graph is not connected");

getch();

}

Output



**Experiment – 9**

**Aim - Find Minimum Cost Spanning Tree of a given undirected graph using Prim’s algorithm.**

#include<stdio.h>

#include<conio.h>

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

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

}

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

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

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

{

idx=i;

printf("the value of i is %d\t",i);

}

visited[idx]=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("\n Edge %d:(%d %d) cost:%d",ne++,a,b,min);

mincost+=min;

visited[b]=1;

}

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

}

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

getch();

}

**Experiment – 10**

**Aim: Implement All-Pairs Shortest Paths Problem using Floyd's algorithm.**

#include<stdio.h>

int min(int,int);

void floyds(int p[10][10],int n)

{

int i,j,k;

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

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

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

if(i==j)

p[i][j]=0;

else

p[i][j]=min(p[i][j],p[i][k]+p[k][j]);

}

int min(int a,int b)

{

if(a<b)

return(a);

else

return(b);

}

void main()

{

int p[10][10],w,n,e,u,v,i,j;;

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

scanf("%d",&n);

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

scanf("%d",&e);

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

{

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

p[i][j]=999;

}

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

{

printf("\n Enter the end vertices of edge%d with its weight \n",i);

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

p[u][v]=w;

}

printf("\n Matrix of input data:\n");

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

{

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

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

printf("\n");

}

floyds(p,n);

printf("\n Transitive closure:\n");

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

{

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

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

printf("\n");

}

printf("\n The shortest paths are:\n");

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

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

{

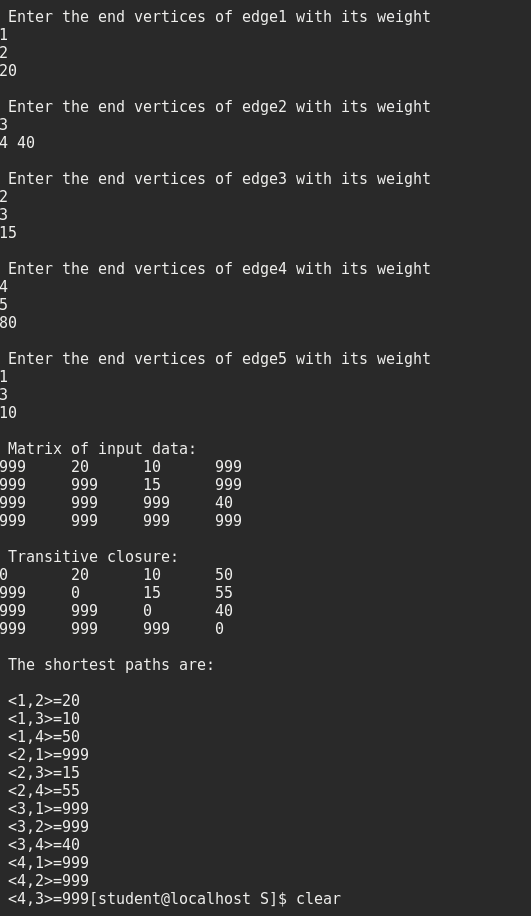
if(i!=j)

printf("\n <%d,%d>=%d",i,j,p[i][j]);

}

}

Output



**Experiment – 11**

**Aim - Implement N Queen's problem using Back Tracking.**

#include <stdio.h>

#include<conio.h>

int row[8],s=0;

int safe(int,int);

void putboard();

void queen(int);

int safe(int x, int y)

{

int i;

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

if( row[y-i]==x || row[y-i]==x-i || row[y-i]==x+i)

return 0;

return 1;

}

void putboard()

{

int x,y;

printf("\nSolution # %d",++s);

printf(":\n---------------------------------\n");

for(y=0;y<8; y++)

{

for (x=0;x<8;x++)

if(x==row[y])

printf("| Q ");

else

printf("|   ");

printf("|\n---------------------------------\n");

}

getch();

}

void queen(int y)

{

int x;

for(x=0;x<8;x++)

{

row[y-1]=x;

if( safe(x,y-1) )

if (y<8)

queen(y+1);

else

putboard();

}

}

void main()

{

clrscr();

queen(1);

}

**Experiment – 12**

**Aim - Implement Bubble Sort algorithm to sort a given set of elements and determine the time required to sort the elements. The elements can be read from a file or can be generated using the random number generator.**

#include<stdio.h>

#include<conio.h>

void main()

{

int i,n,temp,j,arr[25];

clrscr();

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

scanf("%d",&n);

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

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

{

printf(" Array[%d] = ",i);

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

}

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

{

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

{

if(arr[j]>arr[j+1]) //Swapping Condition is Checked

{

temp=arr[j];

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

arr[j+1]=temp;

}

}

}

printf("\nThe Sorted Array is:\n\n");

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

{

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

}

getch();

}

**Experiment – 13**

**Aim - Implement Linear Search algorithm to search a value from a given set of elements. The elements can be read from a file or can be generated using the random number generator.**

//Write a C program that searches a value in a stored array using linear search.

#include<stdio.h>

#include<conio.h>

int linear(int [ ],int,int);

void main( )

{

int a[20], pos = -1, n, k, i;

clrscr( );

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

scanf("%d",&n);

printf("\nEnter elements for an array:");

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

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

printf("\nEnter the element to be searched:");

scanf("%d",&k);

pos=linear(a,n,k);

if(pos != -1)

printf("\n Search successful element found at position %d",pos);

else

printf("\n Search unsuccessful, element not found");

getch( );

}

int linear(int a[ ],int n,int k)

{

int i;

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

{

if(a[i]==k)

return(i);

}

return -1;

}

**////////////////////////////////////////////////////////////////////**

//Write a C program that searches a value in a stored array using recursive linear search.

/\* recursive program for Linear Search\*/

#include<stdio.h>

#include<conio.h>

int linear(int [ ],int,int);

void main( )

{

int a[20],pos=-1,n,k,i;

clrscr();

printf("\nEnter n value:");

scanf("%d",&n);

printf("\nEnter elements for an array:");

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

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

printf("\n Enter the element to be searched:");

scanf("%d",&k);

pos=linear(a,n,k);

if(pos!=-1)

printf("\n Search successful, Element found at Position %d",pos);

else

printf("Search unsuccessful, element not found ");

getch( );

}

int linear(int a[ ],int n,int k)

{

int i;

for(i=n-1;i>=0;i--)

{

if(a[i]==k)

return(i);

else

{

n=n-1;

return(linear(a,n,k));

}

}

return -1;

}

**Experiment – 14**

**Aim - Implement Binary Search algorithm to search a value from a given set of elements. The elements can be read from a file or can be generated using the random number generator.**

//Write a C program that searches a value in a stored array using non recursive binary search.

#include<stdio.h>

#include<conio.h>

int bsearch(int [ ],int,int);

void main( )

{

int a[20],pos,n,k,i;

clrscr();

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

scanf("%d",&n);

printf("\nEnter elements for an array:");

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

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

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

scanf("%d",&k);

pos=bsearch(a,n,k);

if(pos!= -1)

printf("Search successful, element found at position %d",pos);

else

printf("Search unsuccessful, element not found");

getch( );

}

int bsearch(int a[ ],int n, int k)

{

int lb,ub,mid;

lb=0;

ub=n-1;

while(ub>=lb)

{

mid=(lb+ub)/2;

if(k<a[mid])

ub=mid-1;

else if(k>a[mid])

lb=mid+1;

else if(k==a[mid])

return(mid);

}

return -1;

 }

**///////////////////////////////////////////////////////////////////////////////////////**

// Write a program for recursive binary search to find the given element within array

#include<stdio.h>

#include<conio.h>

int bsearch(int [ ],int, int, int);

void main( )

{

int a[20],pos,n,k,i,lb,ub;

clrscr( );

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

scanf("%d",&n);

printf("\nEnter elements for an array:");

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

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

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

scanf("%d",&k);

lb=0;

ub=n-1;

pos=bsearch(a,k,lb,ub);

if(pos!=-1)

printf("Search successful, element found at position %d",pos);

else

printf("Search unsuccessful, element not found");

getch( );

}

int bsearch(int a[ ], int k, int lb, int ub)

{

int mid;

while(ub>=lb)

{

mid=(lb+ub)/2;

if(k<a[mid])

{

ub=mid-1;

}

else if(k>a[mid])

{

lb=mid+1;

}

else if(k==a[mid])

{

return(mid);

}

return(bsearch(a,k,lb,ub));

}

 return -1;

 }

**Experiment – 15**

**Aim - Compute two 2 by 2 matrix multiplication to find multiplication matrix using Strassen’s Matrix Multiplication.**

//C code of two 2 by 2 matrix multiplication using Strassen algorithm:

#include<stdio.h>

#include<conio.h>

void main()

{

int a[2][2],b[2][2],c[2][2],i,j;

int m1,m2,m3,m4,m5,m6,m7;

clrscr();

printf("Enter the 4 elements of first matrix: ");

for(i=0;i<2;i++)

for(j=0;j<2;j++)

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

printf("Enter the 4 elements of second matrix: ");

for(i=0;i<2;i++)

for(j=0;j<2;j++)

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

printf("\nThe first matrix is\n");

for(i=0;i<2;i++)

{

printf("\n");

for(j=0;j<2;j++)

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

}

printf("\nThe second matrix is\n");

for(i=0;i<2;i++)

{

printf("\n");

for(j=0;j<2;j++)

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

}

m1= (a[0][0] + a[1][1])\*(b[0][0]+b[1][1]);

m2= (a[1][0]+a[1][1])\*b[0][0];

m3= a[0][0]\*(b[0][1]-b[1][1]);

m4= a[1][1]\*(b[1][0]-b[0][0]);

m5= (a[0][0]+a[0][1])\*b[1][1];

m6= (a[1][0]-a[0][0])\*(b[0][0]+b[0][1]);

m7= (a[0][1]-a[1][1])\*(b[1][0]+b[1][1]);

c[0][0]=m1+m4-m5+m7;

c[0][1]=m3+m5;

c[1][0]=m2+m4;

c[1][1]=m1-m2+m3+m6;

printf("\nAfter multiplication using \n");

for(i=0;i<2;i++)

{

printf("\n");

for(j=0;j<2;j++)

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

}

   getch();

}

**Experiment – 16**

**Aim - Implement Longest Common Subsequence problem using Dynamic Programming.**

#include<stdio.h>

#include<conio.h>

#include<string.h>

int i,j,m,n,a,c[20][20];

char x[15],y[15],b[20][20];

void print\_lcs(int i,int j)

{

                if(i==0 || j==0)

                                return;

                if(b[i][j]=='c')

                {

                                print\_lcs(i-1,j-1);

                                printf(" %c",x[i-1]);

                }

                else if(b[i][j]=='u')

                                print\_lcs(i-1,j);

                else

                                print\_lcs(i,j-1);

}

void lcs\_length(void)

{

                m=strlen(x);

                n=strlen(y);

                for(i=0;i<=m;i++)

                                c[i][0]=0;

                for(i=0;i<=n;i++)

                                c[0][i]=0;

                for(i=1;i<=m;i++)

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

                                {

                                                if(x[i-1]==y[j-1])

                                                {

                                                                c[i][j]=c[i-1][j-1]+1;

                                                                b[i][j]='c';

                                                }

                                                else if(c[i-1][j]>=c[i][j-1])

                                                {

                                                                c[i][j]=c[i-1][j];

                                                                b[i][j]='u';

                                                }

                                                else

                                                {

                                                                c[i][j]=c[i][j-1];

                                                                b[i][j]='l';

                                                }

                                }

                 print\_lcs(m,n);

 }

void main()

{

                printf("Enter 1st sequence : ");

                gets(x);

                printf("Enter 2nd sequence : ");

                gets(y);

                printf("\nlongest common subsequence is : ");

                lcs\_length();

                getch();

}

**Experiment – 17**

**Aim - Implement Matrix Chain Multiplication problem using Dynamic Programming.**

#include<stdio.h>

#include<conio.h>

void prin(int,int);

void matrixchain(int [],int);

int m[100][100],s[100][100];

void main()

{

int i,j,p[100],n;

clrscr();

printf("enter the no. of matrix chian");

scanf("%d",&n);

printf("enter the order of chain matrix");

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

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

matrixchain(p,n);

printf("cost of matrix \n");

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

{

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

{

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

}

printf("\n");

}

printf("sequence matrix\n");

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

{

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

{

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

}

printf("\n");

}

prin(1,n);

printf("\ncost of matrix chain multiplication %d", m[1][n]);

getch();

}

void matrixchain(int p[],int n)

{

int i,j,k,l,q;

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

{

m[i][i]=0;

}

for(l=2;l<=n;l++)

{

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

{

j=i+l-1;

m[i][j]=9999;

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

{

q=m[i][k]+m[k+1][j]+p[i-1]\*p[k]\*p[j];

if(q<m[i][j])

{

m[i][j]=q;

s[i][j]=k;

}

}

}

}

}

void prin(int i,int j)

{

if(i==j)

{

printf("a%d",i);

}

else

{

printf("(");

prin(i,s[i][j]);

prin((s[i][j]+1),j);

printf(")");

}

}

**Experiment – 18**

**Aim: Implement Graph Coloring Problem using backtracking.**

#include<stdio.h>  
int G[50][50],x[50];  //G:adjacency matrix,x:colors  
void next\_color(int k){  
   int i,j;  
   x[k]=1;  //coloring vertex with color1  
   for(i=0;i<k;i++){ //checking all k-1 vertices-backtracking  
     if(G[i][k]!=0 && x[k]==x[i])  //if connected and has same color  
       x[k]=x[i]+1;  //assign higher color than x[i]  
   }  
}

int main(){  
  int n,e,i,j,k,l;  
  printf("Enter no. of vertices : ");  
  scanf("%d",&n);  //total vertices  
  printf("Enter no. of edges : ");  
  scanf("%d",&e);  //total edges  
   
  for(i=0;i<n;i++)  
    for(j=0;j<n;j++)  
      G[i][j]=0;  //assign 0 to all index of adjacency matrix  
       
  printf("Enter indexes where value is 1-->\n");  
  for(i=0;i<e;i++){  
    scanf("%d %d",&k,&l);  
    G[k][l]=1;  
    G[l][k]=1;  
  }  
   
  for(i=0;i<n;i++)  
    next\_color(i);  //coloring each vertex

  printf("Colors of vertices -->\n");  
  for(i=0;i<n;i++)  //displaying color of each vertex  
    printf("Vertex[%d] : %d\n",i+1,x[i]);  
  
  return 0;  
}  
NOTE : This code is written,compiled and run with GCC compiler under Linux environment Ubuntu12.04LTS Precise Pangolin.

--------------------------------------------------------------------------------------------------------------------------

*OUTPUT*

--------------------------------------------------------------------------------------------------------------------------

no. of vertices : 4

Enter no. of edges : 5  
Enter indexes where value is 1-->  
0 1  
1 2  
1 3  
2 3  
3 0  
Colors of vertices -->  
Vertex[1] : 1  
Vertex[2] : 2  
Vertex[3] : 1  
Vertex[4] : 3

**Experiment – 19**

**Aim: Implement Sum of Subset Problem using backtracking.**

#include<stdio.h>

#include<conio.h>

#define TRUE 1

#define FALSE 0

int inc[50],w[50],sum,n;

int promising(int i,int wt,int total) {

return(((wt+total)>=sum)&&((wt==sum)||(wt+w[i+1]<=sum)));

}

void main() {

int i,j,n,temp,total=0;

clrscr();

printf("\n Enter how many numbers:\n");

scanf("%d",&n);

printf("\n Enter %d numbers to th set:\n",n);

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

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

total+=w[i];

}

printf("\n Input the sum value to create sub set:\n");

scanf("%d",&sum);

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

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

if(w[j]>w[j+1]) {

temp=w[j];

w[j]=w[j+1];

w[j+1]=temp;

}

printf("\n The given %d numbers in ascending order:\n",n);

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

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

if((total<sum))

printf("\n Subset construction is not possible"); else {

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

inc[i]=0;

printf("\n The solution using backtracking is:\n");

sumset(-1,0,total);

}

getch();

}

void sumset(int i,int wt,int total) {

int j;

if(promising(i,wt,total)) {

if(wt==sum) {

printf("\n{\t");

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

if(inc[j])

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

printf("}\n");

} else {

inc[i+1]=TRUE;

sumset(i+1,wt+w[i+1],total-w[i+1]);

inc[i+1]=FALSE;

sumset(i+1,wt,total-w[i+1]);

}

}

}