VISVESVARAYA TECHNOLOGICAL UNIVERSITY

"JnanaSangama", Belgaum -590014, Karnataka.

LAB REPORT on

Analysis and Design of Algorithms

Submitted by

CHANCHAL BHATI(1BM21CS042)

in partial fulfillment for the award of the degree of

BACHELOR OF ENGINEERING

in

COMPUTER SCIENCE AND ENGINEERING

B.M.S. COLLEGE OF ENGINEERING BENGALURU-560019

May-2023 to July-2023

(Autonomous Institution under VTU)

B. M. S. College of Engineering,

Bull Temple Road, Bangalore 560019

(Affiliated To Visvesvaraya Technological University, Belgaum)

Department of Computer Science and Engineering

CERTIFICATE

This is to certify that the Lab work entitled "**Analysis and Design of Algorithms**" carried out by **CHANCHAL BHATI(1BM21CS042),** who is bonafide student of **B.M.S. College of Engineering.** It is in partial fulfillment for the award of **Bachelor of Engineering in Computer Science and Engineering** of the Visvesvaraya Technological University, Belgaum during the academic semester May-2023 to July-2023. The Lab report has been approved as it satisfies the academic requirements in respect of a **Analysis and Design of Algorithms (22CS4PCADA)** work prescribed for the said degree.

Dr.Rajeshwari B S
Assistant Professor
Department of CSE
BMSCE, Bengaluru

Dr. Jyothi S Nayak
Professor and Head
Department of CSE
BMSCE, Bengaluru

INDEX SHEET

Course Outcome

CO1	Analyze time complexity of Recursive and Non-recursive algorithms using asymptotic notations.
CO2	Apply various design techniques for the given problem.
CO3	Apply the knowledge of complexity classes P, NP, and NP-Complete and prove certain problems are NP-Complete
CO4	Design efficient algorithms and conduct practical experiments to solve problems.

PROGRAM 1

Write program to do the following:
a. Print all the nodes reachable from a given starting node in a digraph using the BFS method.
b. Check whether a given graph is connected or not using the DFS method.

CODE:

BFS:

```
#include<stdio.h>
#include<conio.h>
void insert_rear(int q[],int *r, int item, int size)
{ if(*r==size)
   printf("Queue overflow!\n");
   else
   {
      *r=*r+1; q[*r]=item;
   }
}
int delete_front(int q[],int *r, int *f)
{ int del_item=-1;
   *f=*f+1;
   del_item=q[*f];


   return del_item;
}
int isEmpty(int q[], int *r, int *f)
{ if(*r==-1 || *r==*f)
   return     1;     else
   return 0;
}
void main()
{ int n,i,j,r=-1,f=-1;
   printf("Enter the
   number of
   vertices:\n");
   scanf("%d",&n);
   printf("Enter the adjacency matrix representing the graph:\n");
   int graph[n][n]; int vis[n],q[n]; for(int i=0;i<n;i++)
   { for(int j=0;j<n;j++)
```

```
        { scanf("%d",&graph[i][j]);
        } }
    for(int i=0;i<n;i++)
    {   vis[i]=0;   }   int   k=0;
    printf("%d   ",k);   vis[k]=1;
    insert_rear(q,&r,k,n);
    while(isEmpty(q,&r,&f)==0
    )
    { int node=delete_front(q,&r,&f);
      for(j=0;j<n;j++)
      { if(graph[node][j]==1 && vis[j]==0)
        { printf("%d ",j);
          vis[j]=1;
          insert_rear(q,&r,j,n)
          ;
        }
      }
    }
}
```

OUTPUT:

```
Enter the number of vertices:
6
Enter the adjacency matrix representing the graph:
0 1 1 0 0 0
0 0 0 1 1 0
0 0 0 0 0 1
0 1 0 0 0 0
0 1 0 0 0 0
0 0 1 0 0 0
0 1 2 3 4 5
```

DFS:

```
int graph[20][20]; void
  DFS(int i,int vis[],int n)
  {
     int j; printf("%d
     ",i);      vis[i]=1;
     for(j=0;j<n;j++)
     { if(graph[i][j]==1 && vis[j]==0)
        {
           DFS(j,vis,n);
        }
     }
  }         void
  main()
  { int n,i,j,top=-1;
     printf("Enter the number of vertices:\n");
     scanf("%d",&n);
     printf("Enter the adjacency matrix representing the graph:\n");
int  vis[n],st[n];  for(int
     i=0;i<n;i++)
     { for(int j=0;j<n;j++)
        { scanf("%d",&graph[i][j]);
        } }
     for(int i=0;i<n;i++)
     { vis[i]=0;
     }
     DFS(0,vis,n); }
```

OUTPUT:

```
Enter the number of vertices:
5
Enter the adjacency matrix representing the graph:
0 1 1 0 0
1 0 1 1 0
1 1 0 0 1
0 1 0 0 1
0 0 1 1 0
0 1 2 4 3
```

PROGRAM 2

Write a program to obtain the Topological ordering of vertices in a given Digraph.

CODE:

```
#include<stdio.h>
#include<conio.h>

void dfs(int);
int a[10][10],vis[10],exp[10],n,j,m;

void main()
{ int i,x,y;
   printf("enter the number of vertices\n");
   scanf("%d",&n);
   for(i=1;i<=n;i++)
   {
     for(j=1;j<=n;j++
     )
     { a[i][j]=0;
     } vis[i]=0;
   }
   printf("enter the number of edges\n");
   scanf("%d",&m); for(i=1;i<=m;i++)
   { printf("enter an edge\n");
     scanf("%d %d",&x,&y);
     a[x][y]=1;
   } j=0;
   for(i=1;i<=n;i++)
   { if(vis[i]==0)
     dfs(i);
   }
   printf("topological sort\n");
   for(i=n-1;i>=0;i--)
   { printf("%d",exp[i]);
   } getch();
}

void dfs(int v)
{ int i;
   vis[v]=1;
   for(i=1;i<=n;i++)
```

```
    { if(a[v][i]==1 &&
    vis[i]==0) dfs(i); }
    exp[j++]=v;
}
```

OUTPUT:

```
enter the number of vertices
5
enter the number of edges
6
enter an edge
1 2
enter an edge
2 3
enter an edge
1 3
enter an edge
3 4
enter an edge
4 5
enter an edge
3 5
topological sort
12345
```

PROGRAM 3

Implement Johnson Trotter algorithm to generate permutations.

CODE:

```
#include <stdio.h>

#define RIGHT_TO_LEFT 0
#define LEFT_TO_RIGHT 1
void swap(int *a, int *b) {
    int temp = *a; *a
    = *b;
    *b = temp; }
int searchArr(int a[], int n, int mobile) {
    for (int i = 0; i < n; i++) {
        if (a[i] == mobile) {
            return i + 1;
```

```
        } }
    return -1; // Mobile not found
}

int getMobile(int a[], int dir[], int n) {
    int mobile_prev = 0, mobile = 0; for
    (int i = 0; i < n; i++) {
        // Direction 0 represents RIGHT TO LEFT.
        if (dir[a[i] - 1] == RIGHT_TO_LEFT && i != 0) {
            if (a[i] > a[i - 1] && a[i] > mobile_prev) {
                mobile = a[i]; mobile_prev
                = mobile;
            }
        }

        if (dir[a[i] - 1] == LEFT_TO_RIGHT && i != n - 1) {
            if (a[i] > a[i + 1] && a[i] > mobile_prev) {
                mobile = a[i];
                mobile_prev = mobile;
            }
        }
    }

    if (mobile == 0 && mobile_prev == 0) {
        return 0; // No mobile element found
    } else { return
        mobile;
    }
}

void printOnePerm(int a[], int dir[], int n) {
    int mobile = getMobile(a, dir, n); int
    pos = searchArr(a, n, mobile);

    if (dir[a[pos - 1] - 1] == RIGHT_TO_LEFT) {
        swap(&a[pos - 1], &a[pos - 2]);
    } else if (dir[a[pos - 1] - 1] == LEFT_TO_RIGHT) { swap(&a[pos],
        &a[pos - 1]);
    }
for (int i = 0; i < n; i++) {
        if (a[i] > mobile) {
```

```
            if (dir[a[i] - 1] == LEFT_TO_RIGHT) {
                dir[a[i] - 1] = RIGHT_TO_LEFT;
            } else if (dir[a[i] - 1] == RIGHT_TO_LEFT) { dir[a[i]
                - 1] = LEFT_TO_RIGHT;
            }
        }
    }

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

int factorial(int n) {
    int res = 1;
```

```
    for (int i = 1; i <= n; i++) {
        res = res * i;3
    }
    return res;
}

void printPermutation(int n) {
    int a[n]; int dir[n];
for (int i = 0; i < n; i++) { a[i]
    = i + 1;
        printf("%d ", a[i]);
    }
    printf("\n");
for (int i = 0; i < n; i++) {
        dir[i] = RIGHT_TO_LEFT;
    }
for (int i = 1; i < factorial(n); i++) {
    printOnePerm(a, dir, n);
    }
}

int main() { int
    n;
    printf("Enter the value of n: ");
    scanf("%d", &n);

    printPermutation(n);

    return 0;
}
```

OUTPUT:

PROGRAM 4

Sort a given set of N integer elements using Merge Sort technique and compute its time taken. Run the program for different values of N and record the time taken to sort.

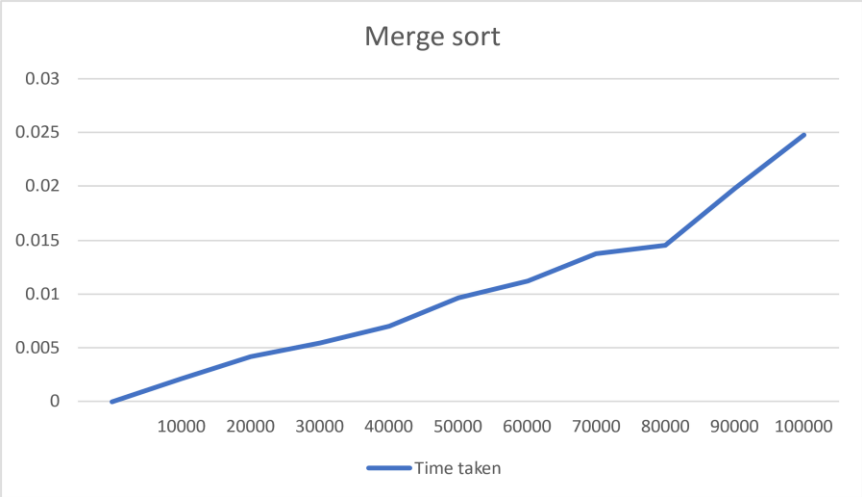
CODE:

```
#include<stdio.h>
#include<time.h>
void conquor(int arr[], int s,int mid,int e)
{ int merged[e-s+1];
   int   i1=s;   int
   i2=mid+1; int
   x=0;
   while(i1<=mid && i2<=e)
   { if(arr[i1]<=arr[i2])
      { merged[x]=arr[i1];
      x++; i1++; } else
      { merged[x]=arr[i2];
         x++; i2++;
      } }
   while(i1<=mid)
```

```
    { merged[x]=arr[i1];
    x++; i1++; }
    while(i2<=e)
    { merged[x]=arr[i2];
       x++;
       i2++; }
    for(int i=0;i<x;i++)
    { arr[i+s]=merged[i];
    }

}
void divide(int arr[], int s, int e)
{ if(s>=e) return; int
    mid=s+(e-s)/2;
    divide(arr,s,mid);
    divide(arr,mid+1,e);
    conquor(arr,s,mid,e)
    ;
}        void
main()
```

```
{ clock_t st,et;
  double ts; int
  n;
  printf("Enter the number of elements in the array:\n");
  scanf("%d", &n); int
  arr[n];
  printf("Enter %d elements of array:\n", n);
  for (int i = 0; i < n; i++) {
     scanf("%d", &arr[i]);
  }  st=clock();
divide(arr,0,n-1);
  et=clock(); ts=(double)((et-st)/CLOCKS_PER_SEC);
  for(int i=0;i<n;i++)
  { printf("%d ",arr[i]);
  }
  printf("\nThe time taken for merge sort is: %f\n",ts);}
```

GRAPH:



OUTPUT:

```
Enter the number of elements in the array:
5
Enter 5 elements of array:
34
65
12
87
16
12 16 34 65 87
The time taken for merge sort is: 0.000000
```

PROGRAM 5

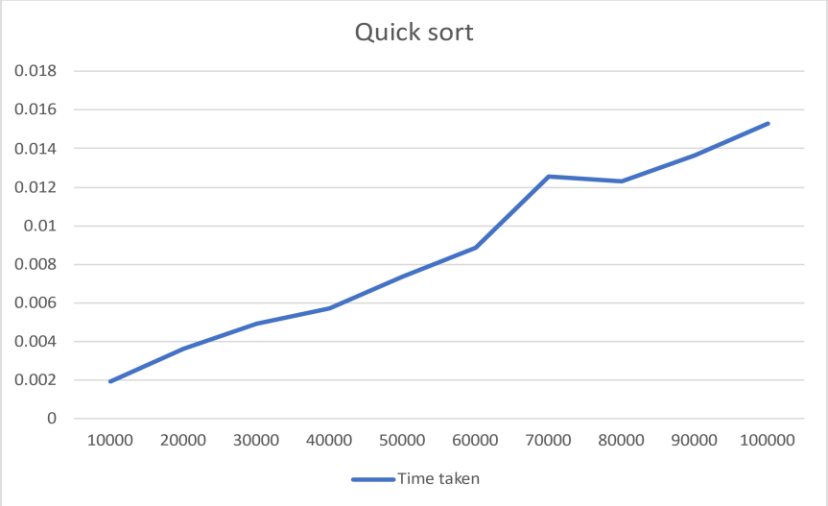
Sort a given set of N integer elements using Quick Sort technique and compute its time taken.

CODE:

```
#include<stdio.h>
#include<time.h>
int partition(int arr[], int low, int high) {
    int pivot = arr[low]; int i = low; int j
    = high + 1;
    int temp;

    while (1) {
       do       {
       i++;
       } while (arr[i] <= pivot && i <= high);
do {
          j--;
       } while (arr[j] > pivot && j >= low);

       if (i >= j)
       { break; }



       temp = arr[i];
    arr[i]   =   arr[j];
    arr[j] = temp; }



    temp = arr[low];
    arr[low] = arr[j];
    arr[j] = temp;
return j;
 }
void quickSort(int arr[], int low, int high) {
    if (low < high) {
       int pidx = partition(arr, low, high);
       quickSort(arr,   low,   pidx   -   1);
       quickSort(arr, pidx + 1, high);
    }
 }

 int main() {
```

```
    int n; clock_t
    st,et;  double
    ts;
    printf("Enter the number of elements in the array:\n");
    scanf("%d", &n); int
    arr[n];
    printf("Enter %d elements of array:\n", n);
    for (int i = 0; i < n; i++) {
        scanf("%d", &arr[i]);
    } st=clock();
    quickSort(arr, 0, n - 1);
    et=clock(); ts=(double)((et-
    st)/CLOCKS_PER_SEC); printf("The
    sorted array is:\n"); for (int i = 0; i < n;
    i++) {
        printf("%d ", arr[i]);
    } printf("\n");
    printf("Time taken for quick sort:%f\n",ts);
    return 0;
}
```

GRAPH:



OUTPUT:

```
Enter the number of elements in the array:
5
Enter 5 elements of array:
43
75
12
32
69
The sorted array is:
12 32 43 69 75
Time taken for quick sort:0.000000
```

PROGRAM 6

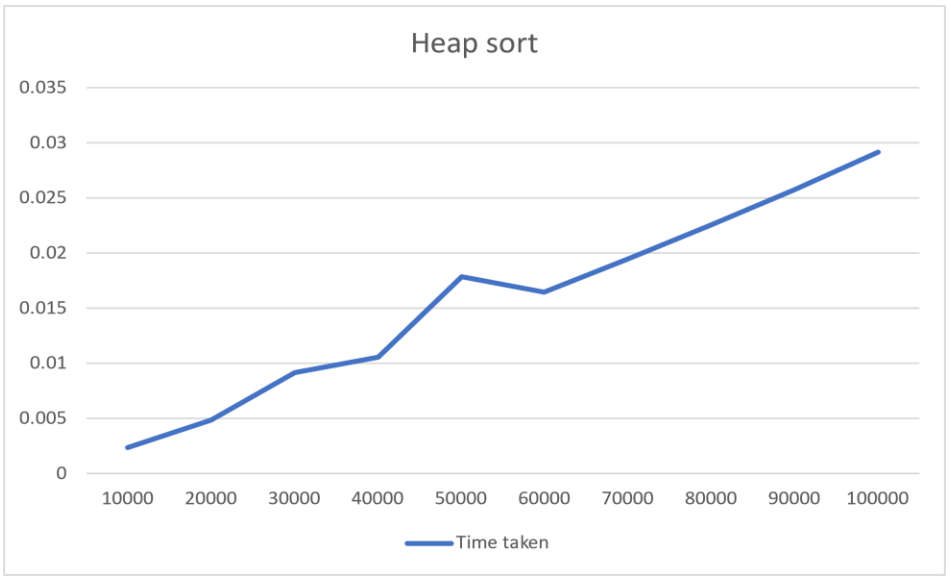
Sort a given set of N integer elements using Heap Sort technique and compute its time taken.

CODE:

```
#include <stdio.h>
#include <time.h>
void heapify(int arr[], int n, int i) {
  int largest = i; int left = 2 * i + 1;
    int right = 2 * i + 2;

    if (left < n && arr[left] > arr[largest]) largest
       = left;

    if (right < n && arr[right] > arr[largest]) largest
       = right;
if (largest != i) { int temp
    =  arr[i];  arr[i]  =
    arr[largest];
    arr[largest] = temp;

       heapify(arr, n, largest);
    }
  }

  void heapSort(int arr[], int n) {
    for (int i = n / 2 - 1; i >= 0; i--)
    heapify(arr, n, i);
for (int i = n - 1; i > 0; i--) { int
    temp  =  arr[0];  arr[0]  =
    arr[i]; arr[i] = temp;
```

```
        heapify(arr, i, 0);
    }
  }

  int main() { int
    n;

    printf("Enter the number of elements: "); scanf("%d",
    &n);
int arr[n];
printf("Enter the elements:\n"); for
    (int i = 0; i < n; i++)
    scanf("%d", &arr[i]);
printf("Original array: "); for
    (int i = 0; i < n; i++)
    printf("%d ", arr[i]);
clock_t start_time = clock(); // Record the start time

    heapSort(arr, n); clock_t end_time = clock(); //

    Record the end time

printf("\nSorted array: "); for
    (int i = 0; i < n; i++)
    printf("%d ", arr[i]);

    double time_taken = (double)(end_time - start_time) / CLOCKS_PER_SEC; printf("\nTime
    taken: %f seconds\n", time_taken);


    return 0;
  }
```

GRAPH:



OUTPUT:

```
Enter the number of elements: 5
Enter the elements:
54 12 34 60 23
Original array: 54 12 34 60 23
Sorted array: 12 23 34 54 60
Time taken: 0.000000 seconds
```

7

Implement 0/1 Knapsack problem using dynamic programming. **CODE:**

```
#include<stdio.h>
void main()
{     int      i,j,w[10],pft[10],x[10],n,cap;
   printf("Enter the number of items\n");
   scanf("%d",&n);
   printf("enter the weight and profit of each item\n"); for(i=1;i<=n;i++)
   { scanf("%d %d",&w[i],&pft[i]);
   }
   printf("enter the knapsack capacity\n"); scanf("%d",&cap);

   int arr[n + 1][cap + 1]; // Corrected the size of the 'arr' matrix

   for (i = 0; i <= n; i++)
   { for (j = 0; j <= cap; j++)
      { if (i == 0 || j == 0)
         arr[i][j] = 0;
         else if (j < w[i]) // Changed 'w[i-1]' to 'w[i]' arr[i][j]
            = arr[i - 1][j];
         else if (j >= w[i]) // Changed 'w[i-1]' to 'w[i]'
         { if (arr[i - 1][j] >= arr[i - 1][j - w[i]] + pft[i]) // Corrected the condition
            arr[i][j] = arr[i - 1][j];
            else arr[i][j] = arr[i - 1][j - w[i]] + pft[i]; // Added 'else'
         block }
      }
   }
   printf("\n Knapsack table\n"); for(i=0;i<=n;i++)
   { for(j=0;j<=cap;j++)
      { printf("%d\t",arr[i][j]);
      }
      printf("\n");
   }
   for(i=n;i>=1;i--)
   { if(arr[i][cap]!=arr[i-1][cap])
      { x[i]=1; cap
         -= w[i];
      } else
```

PROGRAM

```
        { x[i]=0;
        } }
    printf("\nitems selected are designated 1\n"); for(i=1;i<=n;i++)
    { printf("%d ",x[i]);
    }
}
```

OUTPUT:

```
Enter the number of elements: 5
Enter the elements:
54 12 34 60 23
Original array: 54 12 34 60 23
Sorted array: 12 23 34 54 60
Time taken: 0.000000 seconds
```

8

Implement All Pair Shortest paths problem using Floyd's algorithm.

CODE:

```
#include<stdio.h>
void main()
{ int adj[10][10],n,i,j,k; int result[10][10];
    printf("Floyd's                algorithm\n");
    printf("enter the number of vertices\n");
    scanf("%d",&n);
    printf("Enter the distance matrix for %d vertices\n",n); for(i=0;i<n;i++)
    { for(j=0;j<n;j++)
        { scanf("%d",&adj[i][j]);
          result[i][j]=adj[i][j];
        } }
    for(k=0;k<n;k++)
    { for(j=0;j<n;j++)
        { for(i=0;i<n;i++)
            { result[i][j]=result[i][j]<(result[i][k]+result[k][j])?result[i][j]:(result[i][k]+result[k][j]);
            }
```

```
        } }
    printf("\nResult\n")
    ; for(i=0;i<n;i++)
    { for(j=0;j<n;j++)
        { printf("%d\t",result[i][j]);
        }
        printf("\n");
    }
}
```

OUTPUT:

PROGRAM

```
enter the number of vertices
4
Enter the distance matrix for 4 vertices
0 999 3 999
2 0 999 999
999 7 0 1
6 999 999 0

Result
0       10      3       4
2       0       5       6
7       7       0       1
6       16      9       0
```

9

Find Minimum Cost Spanning Tree of a given undirected graph using Prim/Kruskal's algorithm.

KRUSKALS ALGORITHM

CODE:

```
#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()
{ printf("Kruskal's algorithm in C\n");
 printf("========================\n");

 printf("Enter 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("The edges of Minimum Cost Spanning Tree are\n"); while
 (ne < n)
 {
  for (i = 1, min = 999; i <= n; i++)
  { for (j = 1; j <= n; j++)
   {  if  (cost[i][j]  <
    min)
```

PROGRAM

```
        {       min       =
         cost[i][j]; a = u
         = i; b = v = j;
        }
       }
      }

      u = find(u); v
      = find(v);
if (uni(u, v))
      { printf("%d edge (%d,%d) =%d\n", ne++, a, b, min);
        mincost += min;
      }

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

    printf("\nMinimum cost = %d\n", mincost);
    getch();
  }

  int find(int i)
  { while
    (parent[i]) i =
    parent[i]; return
    i;
  }

  int uni(int i, int j)
  { if (i  !=
    j)
     {
```

```
    parent[j] = i; return
     1;
   }

  return 0;
}
```

OUTPUT:

```
Enter the no. of vertices:
4

Enter the cost adjacency matrix:
0 1 3 4
1 0 2 999
3 2 0 5
4 999 5 0
The edges of Minimum Cost Spanning Tree are
1 edge (1,2) =1
2 edge (2,3) =2
3 edge (1,4) =4

Minimum cost = 7
```

PRIMS ALGORITHM

CODE:

```
#include<stdio.h>
#include<conio.h>

int cost[10][10],vt[10],et[10][10],vis[10],j,n;
int sum=0; int x=1; int e=0;
void prims();

void main()
{ int i;
printf("enter the number of vertices\n");
   scanf("%d",&n);
   printf("enter the cost adjacency matrix\n");
   for(i=1;i<=n;i++)
   { for(j=1;j<=n;j++)
      { scanf("%d",&cost[i][j]);
      }
      vis[i]=0;
   } prims();
   printf("edges of spanning tree\n");
   for(i=1;i<=e;i++)
   { printf("%d,%d\t",et[i][0],et[i][1]);
   }
   printf("weight=%d\n",sum);
   getch();
}

void prims() {
```

```
int s,min,m,k,u,v;
vt[x]=1; vis[x]=1;
for(s=1;s<n;s++)
{ j=x;
   min=999;
   while(j>0)
   { k=vt[j];
        for(m=2;m<=n;m++)
        { if(vis[m]==0)
          { if(cost[k][m]<min) {
                min=cost[k][m];
                u=k;
                v=m;
              }
          }
        }
        j--;
  } vt[++x]=v;
  et[s][0]=u;
  et[s][1]=v;
  e++;
  vis[v]=1;
  sum=sum+min;
 }
}
```

OUTPUT:

```
enter the number of vertices
4
enter the cost adjacency matrix
0 1 3 4
1 0 2 999
3 2 0 5
4 999 5 0
edges of spanning tree
1,2     2,3     1,4     weight=7
```

v

PROGRAM 10

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

CODE:

```
#include <stdio.h>
#define INFINITY 999 #define
MAX 10
void Dijkstra(int Graph[MAX][MAX], int n, int start);
void Dijkstra(int Graph[MAX][MAX], int n, int start) {
  int cost[MAX][MAX], distance[MAX], pred[MAX];
  int visited[MAX], count, mindistance, nextnode, i, j;

   // Creating cost matrix for
   (i = 0; i < n; i++) for (j =
   0; j < n; j++) if
   (Graph[i][j] == 0)
   cost[i][j] = INFINITY;
   else
       cost[i][j] = Graph[i][j];
for (i = 0; i < n; i++) {
    distance[i] = cost[start][i];
    pred[i] = start; visited[i] =
    0;
   }
distance[start] = 0;
   visited[start] = 1;
   count = 1;
while (count < n - 1) {
    mindistance = INFINITY;
for (i = 0; i < n; i++)
      if (distance[i] < mindistance && !visited[i]) {
        mindistance = distance[i]; nextnode
        = i;
       }

     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++;
    }

    // Printing the distance
    for (i = 0; i < n; i++) if
    (i != start) {
       printf("\nDistance from source to %d: %d", i, distance[i]); }
  } int main()
  {
     int Graph[MAX][MAX], i, j, n, u;
printf("Enter the number of nodes \n");
     scanf("%d", &n);

     printf("Enter the adjacency matrix of the graph \n");
     for (i = 0; i < n; i++) {
        for (j = 0; j < n; j++) {
           scanf("%d", &Graph[i][j]);
        }
     }

     printf("Enter the source vertex \n"); scanf("%d",
     &u);

     Dijkstra(Graph, n, u); return

     0;
```

OUTPUT:

```
Enter the number of nodes
6
Enter the adjacency matrix of the graph
0 25 100 35 999 999
999 0 999 27 14 999
999 999 0 50 999 999
999 999 999 0 29 999
999 999 999 999 0 21
999 999 48 999 999 0
Enter the source vertex
0

Distance from source to 1: 25
Distance from source to 2: 100
Distance from source to 3: 35
Distance from source to 4: 39
Distance from source to 5: 60
```

PROGRAM 11

Implement “N-Queens Problem” using Backtracking.

CODE:

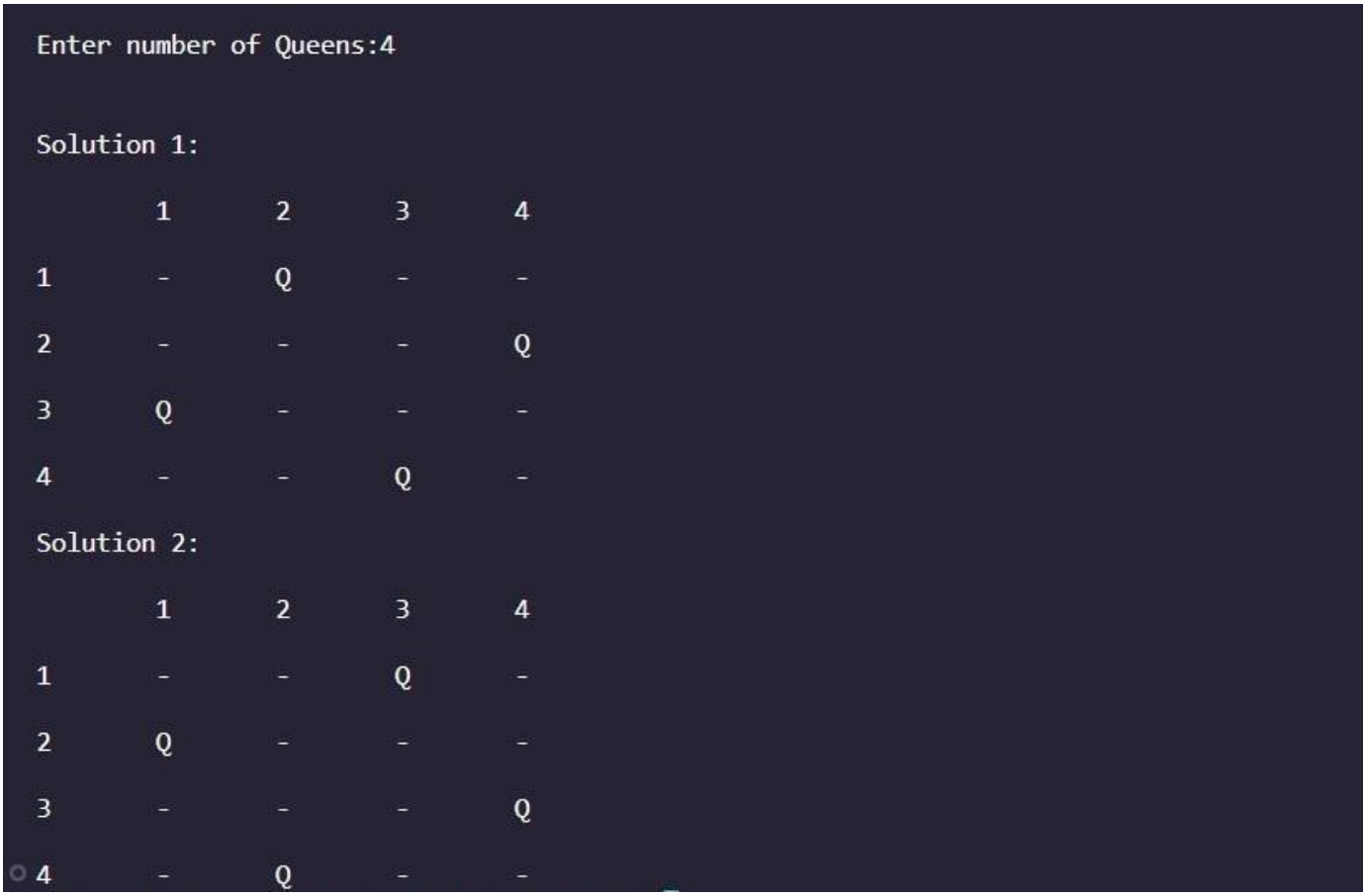
```
#include<stdio.h>

#include<math.h>

int board[20],count;

int main()
{ int n,i,j;
void queen(int row,int n);

printf(" - N Queens Problem Using Backtracking -"); printf("\n\nEnter
number of Queens:");
```

scanf("%d",&n)

```
  ; queen(1,n);
  return 0; }
void print(int n)
  {
  int i,j;
  printf("\n\nSolution %d:\n\n",++count);
for(i=1;i<=n;++i)
   printf("\t%d",i);

  for(i=1;i<=n;++i)
  { printf("\n\n%d",i);
    for(j=1;j<=n;++j) //for nxn
    board
    { if(board[i]==j)
      printf("\tQ"); //queen at i,j position
      else
      printf("\t-"); //empty slot
    }
  }
  }

  int place(int row,int column)
  {
  int i;
  for(i=1;i<=row-1;++i)
  {
   if(board[i]==column)
     return 0;
   else
     if(abs(board[i]-column)==abs(i-row))
     return
  0; } return
  1; }
void queen(int row,int n)
  {           int
  column;
  for(column=1;column<=n;++column)
  { if(place(row,column))
    {  board[row]=column;  //no  conflicts  so  place
    queen if(row==n) //dead end
     print(n); //printing the board configuration
     else //try queen with next position queen(row+1,n);
    }
```

```
}
}
```

OUTPUT:


Enter number of Queens:4
Solution 1:
1 2 3 4
1 - Q - -
2 - - - Q
3 Q - - -
4 - - Q -
Solution 2:
1 2 3 4
1 - - Q -
2 Q - - -
3 - - - Q
4 - Q - -