fsmk.org/labmanual

IV
SEMESTER
FOR CSE & ISE
DESIGN AND ANALYSIS OF ALGORITHAMS
LABORATORY
[10CSL47]

LAB MANUAL

VTU SYLLABUS 2010

FREE SOFTWARE MOVEMENT KARNATAKA

Content

About Free Software Movement Karnataka (FSMK)

Free Software Movement Karnataka (FSMK) is a nonprofit organization formed in March 2009 to spread the ideals of free software. We try to increase the understanding of the philosophy behind free software and encourage its use in educational institutions, our very own community center and other organisations.

The phrase "free and open source software" can be used to collectively describe a set of operating systems and standalone applications that are free from the clutches of large corporate organisations which produce software only for commercial purposes. Free software is often freelyavailable and is created by a thriving community of programmers, designers and writers. The source code(i.e, the computer program) that underlies an application or an operating system is also open to the perusal of the common individual, which allows every curious tinkerer, student orotherwise, to play around with the source code.Most free software organisations welcome the general public to be part of their community and to contribute in any of the three spheres mentioned above. In the event of you not being a programmer, designer or writer, you can also become a member of the community by actively using free software applications, thereby reporting any issues that you notice, and also by spreading awareness about free software among your friends and family.At FSMK, we believe that it is unfortunate that schools, colleges and other small organisations that can use Linux and related free software for no cost, instead invest in using proprietary and commercial software, thereby spending large amounts annually on licensing and other fees, which can instead be used for the betterment of education or related services, and thereby, the betterment of society.

I want to be involved Website: http://fsmk.org/

Mailing list: http://www.fsmk.org/?q=mailinglistsubscribe

About Spoken Tutorial Project:

The Spoken Tutorial project is an initiative of National Mission on Education through ICT, Government of India, to promote IT literacy through Free and Open Source Software. The project is being developed and coordinated by IIT-Bombay and led by Dr. Kannan M. Moudgalya. The project aims at building a repository of self learning courses through video tutorials of various open source softwares. These courses are then used to organize 2 hour workshops in government organizations, NGOs, SMEs and School and Colleges in India completely free of cost for the participants. These tutorials are not only available in English but also in various regional languages for the learner to be able to learn in the language he/she is comfortable in. Currently, Spoken Tutorials are available for free software tools like Blender, GIMP, Latex, Scilab, LibreOffice Suite, Ubuntu Linux, Mozilla Firefox, Thunderbird, MySQL and also programming languages and scripts like C, C++, Python, Ruby, Perl, PHP, Java. All the spoken tutorials which are released under Creative Commons License are available for download free of cost at their website http://spoken-tutorial.org.

About Jnana Vikas Institute of Technology, Bidadi

Jnana Vikas Institute of Technology was established in the year 2001 by JNANA VIKAS VIDYA SANGHA with a

mission to not just provide a solid educational foundation to students but to build their careers, to make them eminent personalities in the society and to make the industry doors open to them. It is approved by AICTE, New Delhi and affiliated to VTU, Belgaum. It has a residential campus with nearly 73 faculties, 28 technical and non-technical supporting staff, 27 administrative and supporting staff and 590 students and is a self-contained campus located in a beautiful green land of about 25 acres. The institute has four academic departments in various disciplines of engineering and three departments in general science with nearly 19 laboratories all together, organized in a unique pattern. There is a separate department for management discipline. The campus is located at Bidadi, in southern part of city of Bengaluru. More information about the college is provided at their website, http://www.jvitedu.in/

Contributors

Following is the list of all the volunteers who contributed to the making of the Lab Manual.

- Ananda Kumar H N, Faculty, A.T.M.E College of Engineering, Mysore
- Arun Chavan L, Faculty, The Oxford College of Engineering, Bangalore
- Bhavya D, Faculty, P.E.S. College of Engineering, Mandya
- Bindu Madavi K P, Faculty, The Oxford College of Engineering, Bangalore
- Byregowda B K, Faculty, Sir M. Visvesvaraya Institute of Technology, Banglore
- Deepika, Faculty, P.E.S. College of Engineering, Mandya
- Kiran B, Faculty, A.T.M.E College of Engineering, Mysore
- Manjunatha H C, Faculty, Sir M. Visvesvaraya Institute of Technology, Banglore
- Neeta Ann Jacob, Faculty, The Oxford College of Engineering, Bangalore
- Rajesh N, Faculty, Sir M. Visvesvaraya Institute of Technology, Banglore
- Shashidhar S, Faculty, A.T.M.E College of Engineering, Mysore
- Shwetha M K, Faculty, P.E.S. College of Engineering, Mandya
- Abdul Nooran, Student, Vivekananda College of Engineering and Technology, Puttur
- Abhiram R., Student, P.E.S Institute of Technology, Bangalore South Campus
- Ajith K.S., Student, The Oxford College of Engineering, Bangalore
- Akshay Gudi, Student, P.E.S. College of Engineering, Mandya
- Arjun M.Y., Student, P.E.S. College of Engineering, Mandya
- Chaitra Kulkarani, Student, Government Engineering College, Hassan
- John Paul S., Student, Jnana Vikas Institute of Technology, Bidadi
- Karan Jain, Student, P.E.S Institute of Technology, Bangalore South Campus
- Kuna Sharathchandra, Student, P.E.S. College of Engineering, Mandya
- Manas J.K., Student, Dr. Ambedkar Institute of Technology, Bangalore
- Manu H., Student, P.E.S. College of Engineering, Mandya
- Meghana S., Student, Government Engineering College, Hassan
- Nagaraj, Student, Vivekananda College of Engineering and Technology, Puttur
- Nandan Hegde, Student, Vivekananda College of Engineering and Technology, Puttur
- Narmada B., Student, P.E.S Institute of Technology, Bangalore South Campus
- Nawaf Abdul, Student, P.A. College of Engineering, Mangalore
- Nitesh A. Jain, Student, B.M.S. Institute of Technology, Bangalore
- Nitin R., Student, The Oxford College of Engineering, Bangalore
- Padmavathi K.,Student, B.M.S. Institute of Technology, Bangalore
- Poojitha Koneti, Student, B.M.S. Institute of Technology, Bangalore
- Rahul Kondi, Student, S.J.B. Institute of Technology, Bangalore
- Rohit G.S., Student,Dr. Ambedkar Institute of Technology, Bangalore
- Samruda, Student, Student, Government Engineering College, Hassan
- Samyama H.M., Student, Government Engineering College, Hassan
- Santosh Kumar, Student,Dr. Ambedkar Institute of Technology, Bangalore
- Shashank M C., Student, S.J.B. Institute of Technology, Bangalore
- Soheb Mohammed, Student, P.E.S Institute of Technology, Bangalore South Campus
- Indra Priyadarshini, Student, P.E.S Institute of Technology, Bangalore South Campus
- Suhas, Student, P.A. College of Engineering, Mangalore
- Vamsikrishna G., Student, P.E.S Institute of Technology, Bangalore South Campus
- Vikram S., Student, P.E.S Institute of Technology, Bangalore South Campus
- Vivek Basavraj, Student, Jnana Vikas Institute of Technology, Bidadi
- Aruna S, Core member of FSMK
- HariPrasad, Core member of FSMK
- Isham, Core member of FSMK
- Jayakumar , Core member of FSMK
- Jeeva J, Core member of FSMK
- Jickson, Core member of FSMK

- Karthik Bhat, Core member of FSMK
- Prabodh C P, Core member of FSMK
- RaghuRam N, Core member of FSMK
- Rameez Thonnakkal, Core member of FSMK
- Sarath M S, Core member of FSMK
- Shijil TV, Core member of FSMK
- Vignesh Prabhu, Core member of FSMK
- Vijay Kulkarni, Core member of FSMK
- Yajnesh, Core member of FSMK

Foreword

"Free Software is the future, future is ours" is the motto with which Free Software Movement Karnataka has been spreading Free Software to all parts of society, mainly amongst engineering college faculty and students. However our efforts were limited due to number of volunteers who could visit different colleges physically and explain what is Free Software and why colleges and students should use Free Software. Hence we were looking for ways to reach out to colleges and students in a much larger way. In the year 2013, we conducted two major Free Software camps which were attended by close to 160 students from 25 different colleges. But with more than 150engineering colleges in Karnataka, we still wanted to find more avenues to reach out to studentsand take the idea of free software in a much larger way to them.

Lab Manual running on Free Software idea was initially suggested by Dr. Ganesh Aithal, Head of Department, CSE, P.A. Engineering College and Dr. Swarnajyothi L., Principal, Jnana Vikas Institute of Technology during our various interactions with them individually. FSMK took their suggestions and decided to create a lab manual which will help colleges to migrate their labs to Free Software, help faculty members to get access to good documentation on how to conduct various labs in Free Software and also help students by providing good and clear explanations of various lab programs specified by the university. We were very clear on the idea that this lab manual should be produced also from the students and faculty members of the colleges as they knew the right way to explain the problems to a large audience with varying level knowledge in the subject. FSMK promotes freedom of knowledge in all respects and hence we were also very clear that the development and release of this lab manual should under Creative Commons License so that colleges can adopt the manual and share,print, distribute it to their students and there by helping us in spreading free software.

Based on this ideology, we decided to conduct a documentation workshop for college faculty members where they all could come together and help us produce this lab manual. As this was a first attempt for even FSMK,we decided to conduct a mock documentation workshop for one day at Indian Institute of Science, Bangalore on 12 Jan, 2014. Close to 40 participants attended it,mainly our students from various colleges and we tried documenting various labs specified by VTU. Based on this experience, we conducted a 3 day residential documentation workshop jointly organized with Jnana Vikas Institute of Technology, Bidadi at their campus from 23 January, 2014. It was attended by 16 faculty members of different colleges and 40 volunteers from FSMK. The documentation workshop was sponsored by Spoken Tutorial Project, an initiative by Government of India to promote IT literacy through Open Source software. Spoken Tutorials are very good learning material to learn about various Free Software tools and hence the videos are excellent companion to this Lab Manual. The videos themselves are released under Creative Commons license, so students can easily download them and share it with others. We would highly recommend our students to go through the Spoken Tutorials while using this Lab Manual and web links to the respective spoken tutorials are shared within the lab manual also.

Finally, we are glad that efforts and support by close to 60 people for around 3 months has lead to creation of this Lab Manual. However like any Free Software project, the lab manual will go through constant improvement and we would like the faculty members and students to send us regular feedback on how we can improve the quality of the lab manual. We are also interested to extend the lab manual project to cover MCA departments and ECE departments and are looking for volunteers who can put the effort in this direction. Please contact us if you are interested to support us.

GCC- GNU C Compiler

GCC is an alternative to the Turbo C compiler. It provides a variety of features and supports many languages apart from C itself.

When you compile a program , "gcc" checks the source code for errors and creates a binary object file of that code (if no errors exist). It then calls the linker to link your code's object file with other pre-compiled object files residing in libraries. These linked object binaries are saved as your newly compiled program.

Options can be provided to gcc to dictate the way a process is performed. For example, you could tell "gcc" to just create the object file and skip the linking specially when developing large programs or building your own libraries.

Options used in GCC:

The important options commonly used in gcc are-
-Wall -L{directory_name}
-l{library} -o{file_name}
where:
{library} denotes a library file
{file_name} denotes the name of a Unix file
{directory_name} name of the directory

Example: main.c

```
#include<stdio.h>
int main(void)
    {
       printf("FSMK");
       return 0;
    }
```

Compilation

```
 gcc main.c
```

Explanation of the options:

-Wall

> This option enables all the warnings in GCC.

-o

> This is to specify the output file name for the executable.
> ex: **gcc main.c -o main**

-l

> Tells the linker to search a standard list of directories for the library (i.e,used to link with shared libraries). The linker then uses this file as if it had been specified precisely by name.
> ex: **gcc main.c -lccp**

-lm

> To use the library math.h, use the -lm option during compilation.
> ex: **gcc main.c -lm**

-L

> Tells the linker to search standard system directories plus user specified directories.

-g

> Generates additional symbolic debuggging information for use with gdb debugger.

-C

> Produce only the compiled code (without any linking)
> ex: **gcc -C main.c**

-D

> The compiler option -D can be used to define the macro MY_MACRO from command line.
> ex: **gcc -DMY_MACRO main.c**

-fopenmp

> This option is used to enable the different OpenMP directives (#pragma omp). This option along with -static is used to link OpenMP.
> ex: **gcc main.c -fopenmp -o main**

Syntatic differences between Turbo C and GCC:

- The library conio.h is not used in GCC. Hence, getch() and other functions using conio.h do not work.
- The function clrscr() does not work.
- Since GNU/Linux environment always expects a running process to return an exit status when the process is completed, the main() function in C Programs should always return an integer instead of returning void.

Advantages of GCC:

1. GCC is free.
2. Supports multiple languages (C,C++,Java etc)
3. GCC is portable. Runs on almost all platforms.
4. Generates backend code.

Resources

- Please go through the video tutorials on C Programming and GCC developed and released by **Spoken Tutorial Project**, an initiative of National Mission on Education through ICT, Government of India, to promote IT literacy through Open Source Software. Students can go through these video tutorials to get better understanding of the subject. The tutorials can be downloaded from here. More info about the project can be found here

Lab Programs list for Analysis and Design of Algorithms Lab as specified by VTU for 4th Semester students:

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. Using OpenMP, 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 a subset of a given set S = {sl, s2,.....,sn} of n positive integers whose sum is equal to a given positive integer d. For example, if S= {1, 2, 5, 6, 8} and d = 9 there are two solutions{1,2,6}and{1,8}. A suitable message is to be displayed if the given problem instance doesn't have a solution.
9. Implement any scheme to find the optimal solution for the Traveling Salesperson problem and then solve the same problem instance using any approximation algorithm and determine the error in the approximation.
10. Find Minimum Cost Spanning Tree of a given undirected graph using Prim's algorithm.
11. Implement All-Pairs Shortest Paths Problem using Floyd's algorithm. Parallelize this algorithm, implement it using OpenMP and determine the speed-up achieved.
12. Implement N Queen's problem using Back Tracking.

Aim:

To 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.

Description:

> The program is based on the Quicksort algorithm which is an instatiation of divide and conquer method of solving the problem. Here the given array is partitioned every time and the sub-array is sorted.Dividing is based on an element called pivot. A divide and conquer algorithm works by recursively breaking down a problem into two or more sub-problems of the same (or related) type, until these become simple enough to be solved directly. The solutions to the sub-problems are then combined to give a solution to the original problem.

Algorithm:

1. Pick an element, called a pivot, from the list.
2. Reorder the list so that all elements with values less than the pivot come before the pivot, while all elements with values greater than the pivot come after it (equal values can go either way). After this partitioning, the pivot is in its final position. This is called the partition operation.
3. Recursively apply the above steps to the sub-list of elements with smaller values and separately the sub-list of elements with greater values.

Code:

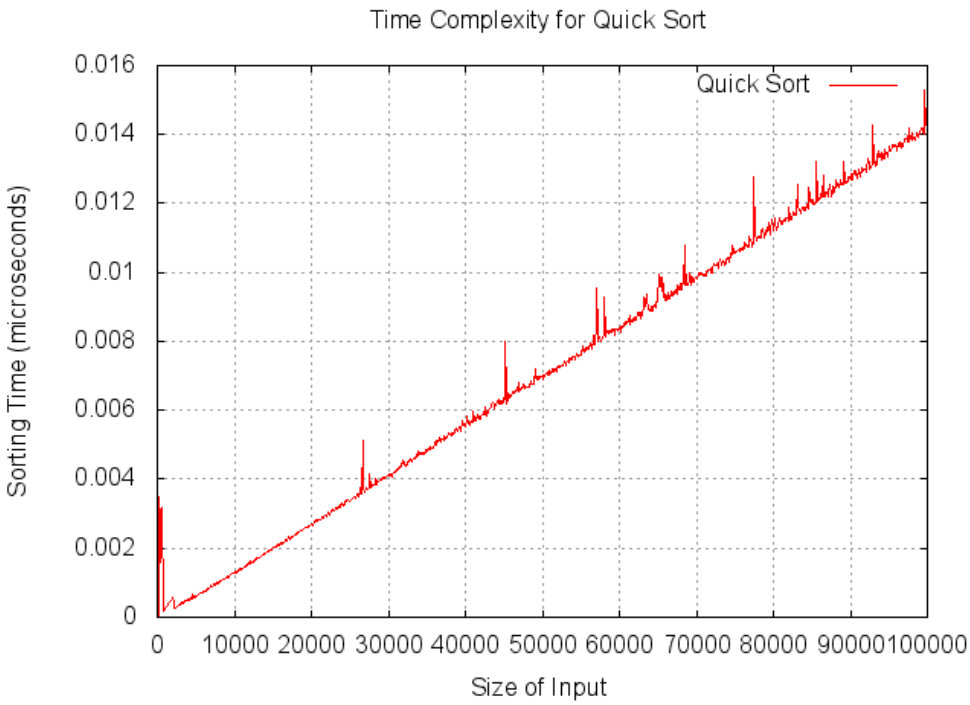
```
#include <stdio.h>
#include <stdlib.h>
#include <sys/time.h>
#include <time.h>

void fnGenRandInput(int [], int);
void fnDispArray( int [], int);
int fnPartition(int [], int , int );
void fnQuickSort(int [], int , int );
inline void fnSwap(int*, int*);


inline void fnSwap(int *a, int *b)
{
    int t = *a; *a = *b; *b = t;
}

/*************************************************************************
*Function    : main
*Input parameters:
*    int argc - no of commamd line arguments
*    char **argv - vector to store command line argumennts
*RETURNS    :    0 on success
*************************************************************************/

int main( int argc, char **argv)
{

    FILE *fp;
    struct timeval tv;
```

```
    double dStart,dEnd;
    int iaArr[500000],iNum,iPos,iKey,i,iChoice;

    for(;;)
    {
    printf("\n1.Plot the Graph\n2.QuickSort\n3.Exit");
    printf("\nEnter your choice\n");
    scanf("%d",&iChoice);

    switch(iChoice)
    {
        case 1:
            fp = fopen("QuickPlot.dat","w");

            for(i=100;i<100000;i+=100)
            {
                fnGenRandInput(iaArr,i);

                gettimeofday(&tv,NULL);
                dStart = tv.tv_sec + (tv.tv_usec/1000000.0);

                fnQuickSort(iaArr,0,i-1);

                gettimeofday(&tv,NULL);
                dEnd = tv.tv_sec + (tv.tv_usec/1000000.0);

                fprintf(fp,"%d\t%lf\n",i,dEnd-dStart);

            }
            fclose(fp);

            printf("\nData File generated and stored in file < QuickPlot.dat >.\n Use a plottin
g utility\n");
        break;

        case 2:
            printf("\nEnter the number of elements to sort\n");
            scanf("%d",&iNum);
            printf("\nUnsorted Array\n");
            fnGenRandInput(iaArr,iNum);
            fnDispArray(iaArr,iNum);
            fnQuickSort(iaArr,0,iNum-1);
            printf("\nSorted Array\n");
            fnDispArray(iaArr,iNum);
        break;

        case 3:
            exit(0);
    }
    }

    return 0;
}


/**************************************************************************
*Function    : fnPartition
*Description    : Function to partition an iaArray using First element as Pivot
*Input parameters:
*    int a[] - iaArray to hold integers
*    int l    - start index of the subiaArray to be sorted
*    int r    - end index of the subiaArray to be sorted
```

```
*RETURNS    : integer value specifying the location of partition
*****************************************************************************/

int fnPartition(int a[], int l, int r)
{
    int i,j,temp;
    int p;

    p = a[l];
    i = l;
    j = r+1;

    do
    {
        do { i++; } while (a[i] < p);
        do { j--; } while (a[j] > p);

        fnSwap(&a[i], &a[j]);
    }
    while (i<j);

    fnSwap(&a[i], &a[j]);
    fnSwap(&a[l], &a[j]);

    return j;
}

/*****************************************************************************
*Function    : fnQuickSort
*Description    : Function to sort elements in an iaArray using Quick Sort
*Input parameters:
*    int a[] - iaArray to hold integers
*    int l    - start index of the subiaArray to be sorted
*    int r    - end index of the subiaArray to be sorted
*RETURNS    : no value
*****************************************************************************/

void fnQuickSort(int a[], int l, int r)
{
    int s;

    if (l < r)
    {
        s = fnPartition(a, l, r);
        fnQuickSort(a, l, s-1);
        fnQuickSort(a, s+1, r);
    }
}

/*****************************************************************************
*Function    : GenRandInput
*Description    : Function to generate a fixed number of random elements
*Input parameters:
*    int X[] - array to hold integers
*    int n    - no of elements in the array
*RETURNS    :no return value
*****************************************************************************/

void fnGenRandInput(int X[], int n)
{
    int i;
```

```
    srand(time(NULL));
    for(i=0;i<n;i++)
    {
        X[i] = rand()%10000;
    }

}

/*****************************************************************************
*Function    : DispArray
*Description    : Function to display elements of an array
*Input parameters:
*    int X[] - array to hold integers
*    int n    - no of elements in the array
*RETURNS    : no return value
*****************************************************************************/

void fnDispArray( int X[], int n)
{
    int i;

    for(i=0;i<n;i++)
        printf(" %5d \n",X[i]);

}
```

OUTPUT:



AIM:

Program to sort an array using Merge Sort

DESCRIPTION:

> Merge sort is an O(n log n) comparison-based sorting algorithm. Most implementations produce a stable sort, meaning that the implementation preserves the input order of equal elements in the sorted output. It is a divide and conquer algorithm.

ALGORITHM:

1. Mergesort(A[O .. n - 1])
2. Sorts array A[O .. n - 1] by recursive mergesort
3. Input: An array A[O .. n - 1] of orderable elements
4. Output: Array A[O .. n - 1] sorted in nondecreasing order
5. Merge(B[O .. p- 1], C[O .. q -1], A[O.. p + q -1])
6. Merges two sorted arrays into one sorted array
7. Input: Arrays B[O .. p -1] and C[O .. q -1] both sorted
8. Output: Sorted array A[O .. p + q -1] of the elements of Band C

CODE:

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


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

*Function    : simplemerge
*Description    : Function to merge two sorted arrays
*Input parameters:
*    int a[] - iaArray to hold integers
*    int low    - start index of the subiaArray to be sorted
*    int mid    - mid index of the subiaArray to be sorted
*    int right    - end index of the subiaArray to be sorted
*RETURNS    : no value

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

void simplemerge(int a[], int low, int mid, int high)
{
    int i,j,k,c[20000];
    i=low;
    j=mid+1;
    k=low;
    int tid;
    omp_set_num_threads(10);
    {
        tid=omp_get_thread_num();
        while(i<=mid&&j<=high)
        {
            if(a[i] < a[j])
            {
```

```
                c[k]=a[i];
                //printf("%d%d",tid,c[k]);
                i++;
                k++;
            }
            else
            {
                c[k]=a[j];
                //printf("%d%d", tid, c[k]);
                j++;
                k++;
            }
        }
    }
    while(i<=mid)
    {
        c[k]=a[i];
        i++;
        k++;
    }
    while(j<=high)
    {
        c[k]=a[j];
        j++;
        k++;
    }
    for(k=low;k<=high;k++)
    a[k]=c[k];
}

/**************************************************************************
*Function    : merge
*Description    : Function to sort elements in an iaArray using Quick Sort
*Input parameters:
    int a[] - iaArray to hold integers
    int low    - start index of the array to be sorted
    int high- end index of the array to be sorted
*RETURNS    : no value
**************************************************************************/

void merge(int a[],int low,int high)
{
    int mid;
    if(low < high)
    {
        mid=(low+high)/2;
        merge(a,low,mid);
        merge(a,mid+1,high);
        simplemerge(a,low,mid,high);
    }
}

void getnumber(int a[], int n)
{
    int i;
    for(i=0;i < n;i++)
        a[i]=rand()%100;
}

/**************************************************************************
*Function    : main
*Input parameters: no
```

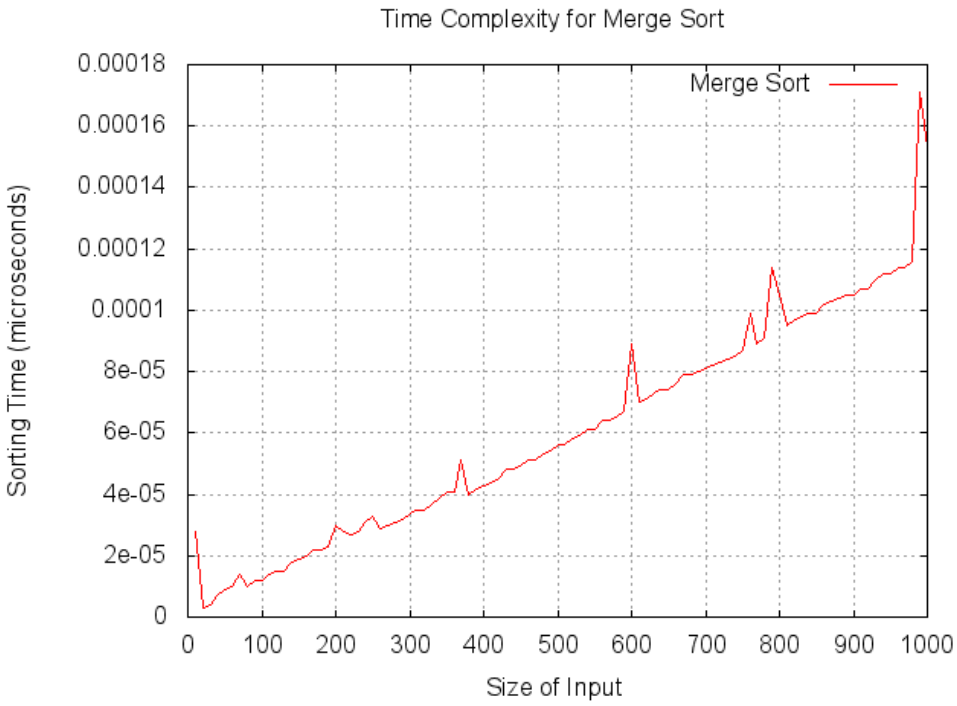
```
*RETURNS    :    0 on success
*****************************************************************************/
int main()
{
    FILE *fp;
    int a[2000],i;
    struct timeval tv;
    double start, end, elapse;
    fp=fopen("mergesort.txt","w");
    for(i=10;i<=1000;i+=10)
    {
        getnumber(a,i);
        gettimeofday(&tv,NULL);
        start=tv.tv_sec+(tv.tv_usec/1000000.0);
        merge(a,0,i-1);
        gettimeofday(&tv,NULL);
        end=tv.tv_sec+(tv.tv_usec/1000000.0);
        elapse=end-start;
        fprintf(fp,"%d\t%lf\n",i,elapse);
    }
    fclose(fp);
    system("gnuplot");
    return 0;
}
```

mergesort.gpl

Gnuplot script file for plotting data in file "mergesort.txt" This file is called mergesort.gpl

```
set terminal png font arial
set title "Time Complexity for Merge Sort"
set autoscale
set xlabel "Size of Input"
set ylabel "Sorting Time (microseconds)"
set grid
set output "mergesort.png"
plot "mergesort.txt" t "Merge Sort" with lines
```

OUTPUT


Time Complexity for Merge Sort
Merge Sort
Sorting Time (microseconds)
Size of Input
0.00018
0.00016
0.00014
0.00012
0.0001
8e-05
6e-05
4e-05
2e-05
0
0
100
200
300
400
500
600
700
800
900
1000

AIM : To obtain the Topological ordering of vertices in a given digraph.

DESCRIPTION :

> Topological sorting for Directed Acyclic Graph (DAG) is a linear ordering of vertices such that for every directed edge uv, vertex u comes before v in the ordering.Topological Sorting for a graph is not possible if the graph is not a DAG. Input parameters: int a[MAX][MAX] - adjacency matrix of the input graph int n - no of vertices in the graph

ALGORITHM :

L Empty list that will contain the sorted elements S Set of all nodes with no incoming edges while S is non-empty do remove a node n from S add n to tail of L for each node m with an edge e from n to m do remove edge e from the graph if m has no other incoming edges then insert m into S if graph has edges then return error (graph has at least one cycle) else return L (a topologically sorted order)

CODE :

```
#include <stdio.h>

const int MAX = 10;
void fnTopological(int a[MAX][MAX], int n);
int main(void)
{
    int a[MAX][MAX],n;
    int i,j;

    printf("Topological Sorting Algorithm -\n");
    printf("\nEnter the number of vertices : ");
     scanf("%d",&n);

    printf("Enter the adjacency matrix -\n");
    for (i=0; i<n; i++)
        for (j=0; j<n; j++)
            scanf("%d",&a[i][j]);

    fnTopological(a,n);
    printf("\n");
    return 0;
}

void fnTopological(int a[MAX][MAX], int n)
{
    int in[MAX], out[MAX], stack[MAX], top=-1;
    int i,j,k=0;

    for (i=0;i<n;i++)
    {
        in[i] = 0;
        for (j=0; j<n; j++)
            if (a[j][i] == 1)
                in[i]++;
    }
```

```
    while(1)
    {
        for (i=0;i<n;i++)
        {
            if (in[i] == 0)
            {
                stack[++top] = i;
                in[i] = -1;
            }
        }

        if (top == -1)
            break;

        out[k] = stack[top--];

        for (i=0;i<n;i++)
        {
            if (a[out[k]][i] == 1)
                in[i]--;
        }
        k++;
    }

    printf("Topological Sorting (JOB SEQUENCE) is:- \n");
    for (i=0;i<k;i++)
        printf("%d ",out[i] + 1);
}
```

OUTPUT

Input Graph : 5 vertices 0 0 1 0 0 0 0 1 0 0 0 0 0 1 1 0 0 0 0 1 0 0 0 0 0

Topological Sorting (JOB SEQUENCE) is:- 2 1 3 4 5

AIM : Compute the transitive closure of a given directed graph using Warshall's algorithm.

DESCRIPTION :

> Warshall's algorithm determines whether there is a path between any two nodes in the graph. It does not give the number of the paths between two nodes. According to Warshall's algorith,a path exists between two vertices i, j, iff there is a path from i to j or there is a path from i to j through 1,..,k intermadiate nodes.

ALGORITHM:

```
    n = |V|
    t(0) = the adjacency matrix for G

    for i in 1..n do
        t(0)[i,i] = True
    end for

    for k in 1..n do
        for i in 1..n do
            for j in 1..n do
                t(k)[i,j] = t(k-1)[i,j] OR
                    (t(k-1)[i,k] AND t(k-1)[k,j])
            end for
        end for
    end for
    return t(n)
```

CODE :

```
#include<stdio.h>
const int MAX = 100;

void WarshallTransitiveClosure(int graph[MAX][MAX], int numVert);
int main(void)
{
    int i, j, numVert;
    int graph[MAX][MAX];

    printf("Warshall's Transitive Closure\n");
    printf("Enter the number of vertices : ");
    scanf("%d",&numVert);

    printf("Enter the adjacency matrix :-\n");
    for (i=0; i<numVert; i++)
        for (j=0; j<numVert; j++)
            scanf("%d",&graph[i][j]);

    WarshallTransitiveClosure(graph, numVert);

    printf("\nThe transitive closure for the given graph is :-\n");
    for (i=0; i<numVert; i++)
    {
```

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

    return 0;
}

void WarshallTransitiveClosure(int graph[MAX][MAX], int numVert)
{
    int i,j,k;

    for (k=0; k<numVert; k++)
    {
        for (i=0; i<numVert; i++)
        {
            for (j=0; j<numVert; j++)
            {
                if (graph[i][j] || (graph[i][k] && graph[k][j]))
                    graph[i][j] = 1;
            }
        }
    }
}
```

OUTPUT

Enter the number of vertices : 4 Enter the adjacency matrix :- 0 0 1 0 0 0 0 1 1 0 0 0 0 1 0 0

The transitive closure for the given graph is :- 1 0 1 0
0 1 0 1
1 0 1 0
0 1 0 1

Warshall's Transitive Closure Enter the number of vertices : 4 Enter the adjacency matrix :-

0 1 1 0 1 0 0 1 1 0 0 1 0 1 1 0

The transitive closure for the given graph is :- 1 1 1 1
1 1 1 1
1 1 1 1
1 1 1 1

AIM:

To solve 0/1 Knapsack problem using Dynamic Programming.

DESCRIPTION:

> The Knapsack problem is probably one of the most interesting and most popular in computer science, especially when we talk about dynamic programming.The knapsack problem is a problem in combinatorial optimization. Given a set of items, each with a weight and a value, determine the number of each item to include in a collection so that the total weight is less than or equal to a given limit and the total value is as large as possible. It derives its name from the problem faced by someone who is constrained by a fixed-size knapsack and must fill it with the most valuable items.

Basic Algorithm:

```
Input:
a set of items with weights and values
output:
the greatest combined value of a subset
partition the set {1...n} into two sets A and B of approximately equal size
compute the weights and values of all subsets of each set
for each subset of A
find the subset of B of greatest value such that the combined weight is less than W
keep track of the greatest combined value seen so far
```

ALGORITHM:

```
Input:
 Values (stored in array v or profit)
 Weights (stored in array w or weight)
 Number of distinct items (n)
 Knapsack capacity (W)
for j from 0 to W do
  m[0, j] = 0
end for
for i from 1 to n do
  for j from 0 to W do
    if w[i] <= j then
      m[i, j] = max(m[i-1, j], m[i-1, j-w[i]] + v[i])
    else
      m[i, j] = m[i-1, j]
    end if
  end for
end for
```

CODE:

```
#include <iostream>
#include <cstdlib>
using namespace std;

const int MAX = 10;
inline int max(int a, int b);
void fnProfitTable(int w[MAX], int p[MAX], int n, int c, int t[MAX][MAX]);
```

```
void fnSelectItems(int n,int c, int t[MAX][MAX], int w[MAX], int l[MAX]);<\pre>

/*************************************************************************
*Function    : main
*Input parameters: no parameters
*RETURNS    :    0 on success
*************************************************************************/<\pre>
    int main(void)
    {
        int i, j, totalProfit;
        int weight[MAX];
        int profit[MAX];
        int capacity;
        int num;
        int loaded[MAX];
        int table[MAX][MAX];
        cout<<"Enter the maxium number of objects : ";
     cin >> num;

        cout << "Enter the weights : \n";
        for (i=1; i<=num; i++)
    {
        cout << "\nWeight " << i << ": ";
        cin >> weight[i];
    }
    cout << "\nEnter the profits : \n";
    for (i=1; i<=num; i++)
    {
        cout << "\nProfit " << i << ": ";
        cin >> profit[i];
    }
    cout << "\nEnter the maximum capacity : ";
    cin >> capacity;


    totalProfit = 0;

    for( i=1; i<=num; i++)
        loaded[i] = 0;

    fnProfitTable(weight,profit,num,capacity,table);
    fnSelectItems(num,capacity,table,weight,loaded);
    cout << "Profit Matrix\n";
    for (i=0; i<=num; i++)
    {
        for(j=0; j<=capacity; j++)
        {
        cout <<"\t"<<table[i][j];
        }
        cout << endl;
    }

    cout << "\nItem numbers which are loaded : \n{ ";
    for (i=1; i<=num; i++)
    {
        if (loaded[i])
        {
            cout <<i << " ";
            totalProfit += profit[i];
        }
    }
    cout << "}" << endl;
```

```
    cout << "\nTotal Profit : " << totalProfit << endl;
    return 0;
}

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

/*****************************************************************************
*Function    : fnProfitTable
*Description    : Function to construct the profit table
*Input parameters:
*    int w[MAX]    -    weight vector
*    int p[MAX]    -    profit vector
*    int n    - no of items
*    int c    - knapsack capacity
*    int t[MAX][MAX] - profit table
*RETURNS    : no value
*****************************************************************************/

void fnProfitTable(int w[MAX], int p[MAX], int n, int c, int t[MAX][MAX])
{
    int i,j;

    for (j=0; j<=c; j++)
        t[0][j] = 0;

    for (i=0; i<=n; i++)
        t[i][0] = 0;

    for (i=1; i<=n; i++)
    {
        for (j=1; j<=c; j++)
        {
            if (j-w[i] < 0)
                t[i][j] = t[i-1][j];
            else
                t[i][j] = max( t[i-1][j], p[i] + t[i-1][j-w[i]]);
        }
    }
}

/*****************************************************************************
*Function    : fnSelectItems
*Description: Function to determine optimal subset that fits into the knapsack
*Input parameters:
*    int n    - no of items
*    int c    - knapsack capacity
*    int t[MAX][MAX] - profit table
*    int w[MAX]    -    weight vector
*    int l[MAX]    -    bit vector representing the optimal subset
*RETURNS    : no value
*****************************************************************************/

void fnSelectItems(int n,int c, int t[MAX][MAX], int w[MAX], int l[MAX])
{
    int i,j;

    i = n;
    j = c;
```

```
    while (i >= 1 && j >= 1)
    {
            if (t[i][j] != t[i-1][j])
            {
                l[i] = 1;
                j = j - w[i];
                i--;
            }
            else
                i--;
    }
}
```

OUTPUT:

Enter the maxium number of objects : 4 Enter the weights :

Weight 1: 2

Weight 2: 1

Weight 3: 3

Weight 4: 2

Enter the profits :

Profit 1: 12

Profit 2: 10

Profit 3: 20

Profit 4: 15

Enter the maximum capacity : 5 Profit Matrix 0 0 0 0 0 0 0 0 12 12 12 12 0 10 12 22 22 22 0 10 12 22 30 32 0 10 15 25 30 37

Item numbers which are loaded : { 1 2 4 }

Total Profit : 37

AIM:Program 5. From a given vertex in a weighted connected graph, find shortest paths to other vertices using Dijkstra's algorithm.

DESCRIPTION:

1. To find the shortest path between points, the weight or length of a path is calculated as the sum of the weights of the edges in the path.
2. A path is a shortest path is there is no path from x to y with lower weight.
3. Dijkstra's algorithm finds the shortest path from x to y in order of increasing distance from x. That is, it chooses the first minimum edge, stores this value and adds the next minimum value from the next edge it selects.
4. It starts out at one vertex and branches out by selecting certain edges that lead to new vertices.
5. It is similar to the minimum spanning tree algorithm, in that it is "greedy", always choosing the closest edge in hopes of an optimal solution.

ALGORITHM:

```
function Dijkstra(Graph, source):
      for each vertex v in Graph:                                    // Initializations
      dist[v]  := infinity ;                                        // Unknown distance function from
                                                                    // source to v
      previous[v]  := undefined ;                                   // Previous node in optimal path
      end for                                                       // from source

      dist[source]  := 0 ;                                          // Distance from source to sou
rce
      Q := the set of all nodes in Graph ;                          // All nodes in the graph are
                                                                    // unoptimized - thus are in Q
      while Q is not empty:                                         // The main loop
      u := vertex in Q with smallest distance in dist[] ;           // Source node in first case
      remove u from Q ;
      if dist[u] = infinity:
         break ;                                                    // all remaining vertices are
      end if                                                        // inaccessible from source

      for each neighbor v of u:                                     // where v has not yet been
                                                                    // removed from Q.
          alt := dist[u] + dist_between(u, v) ;
          if alt < dist[v]:                                         // Relax (u,v,a)
              dist[v]  := alt ;
              previous[v]  := u ;
              decrease-key v in Q;                                  // Reorder v in the Queue
          end if
      end for
      end while
      return dist;
end function
```

CODE:

```
/*****************************************************************************
*File        : Dijkstra.cpp
*Description    : Program to find shortest paths to other vertices
```

```
              using Dijkstra's algorithm.
*Author        : Prabodh C P
*Compiler    : gcc compiler 4.6.3, Ubuntu 12.04
*Date        : Friday 22 November 2013

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


 #include<ostream>
 #include<cstdio>
using namespace std;

const int MAXNODES = 10,INF = 9999;

void fnDijkstra(int [][MAXNODES], int [], int [], int[], int, int, int);

/*******************************************************************************
*Function    : main
*Input parameters: no parameters
*RETURNS    :    0 on success
********************************************************************************/

int main(void)
{
    int n,cost[MAXNODES][MAXNODES],dist[MAXNODES],visited[MAXNODES],path[MAXNODES],i,j,source,d
est;

    cout << "\nEnter the number of nodes\n";
    cin >> n;
    cout << "Enter the Cost Matrix\n" << endl;
    for (i=0;i<n;i++)
    for (j=0;j<n;j++)
        cin >> cost[i][j];


    for (source = 0; source < n; source++)
    {
    getchar();
    cout << "\n//For Source Vertex : " << source  << " shortest path to other vertices//"<< end
l;
    for (dest=0; dest < n; dest++)
    {
        fnDijkstra(cost,dist,path,visited,source,dest,n);


        if (dist[dest] == INF)
            cout << dest << " not reachable" << endl;
        else
        {
            cout << endl;
            i = dest;
        do
            {
                cout << i << "<--";
                i = path[i];
            }while (i!= source);
            cout << i << " = " << dist[dest] << endl;
        }
    }
    cout << "Press Enter to continue...";
    }
```

```
    return 0;
}



/**************************************************************************
*Function    : fnDijkstra
*Description    : Function to find shortest paths to other vertices
              using Dijkstra's algorithm.
*Input parameters:
*    int c[][] - cost adjacency matrix of the graph
*    int d[] - distance vector
*    int p[] - path vector
*    int s[] - vector to store visited information
*    int so    - source vertex
*    int de    - destination vertex
*    int n    - no of vertices in the graph
*RETURNS    : no value
**************************************************************************/

void fnDijkstra(int c[MAXNODES][MAXNODES], int d[MAXNODES], int p[MAXNODES], int s[MAXNODES], i
nt so, int de, int n)
{
    int i,j,a,b,min;

    for (i=0;i<n;i++)
    {
    s[i] = 0;
    d[i] = c[so][i];
    p[i] = so;
    }

    s[so] = 1;

    for (i=1;i<n;i++)
    {
    min = INF;
    a = -1;
    for (j=0;j<n;j++)
    {
        if (s[j] == 0)
        {
            if (d[j] < min)
            {
                min = d[j];
                a = j;
            }
        }
    }

    if (a == -1) return;

    s[a] = 1;

    if (a == de) return;

    for (b=0;b<n;b++)
    {
        if (s[b] == 0)
        {
            if (d[a] + c[a][b] <d[b])
            {
```

```
                d[b] = d[a] + c[a][b];
                p[b] = a;
            }
        }
    }
    }
}
```

Output

Enter the number of nodes 5 Enter the Cost Matrix

0 3 9999 7 9999 3 0 4 2 9999 9999 4 0 5 6 7 2 5 0 4 9999 9999 6 4 0

//For Source Vertex : 0 shortest path to other vertices//

0<--0 = 0

1<--0 = 3

2<--1<--0 = 7

3<--1<--0 = 5

4<--3<--1<--0 = 9 Press Enter to continue...

//For Source Vertex : 1 shortest path to other vertices//

0<--1 = 3

1<--1 = 0

2<--1 = 4

3<--1 = 2

4<--3<--1 = 6 Press Enter to continue...

//For Source Vertex : 2 shortest path to other vertices//

0<--1<--2 = 7

1<--2 = 4

2<--2 = 0

3<--2 = 5

4<--2 = 6 Press Enter to continue...

//For Source Vertex : 3 shortest path to other vertices//

0<--1<--3 = 5

1<--3 = 2

2<--3 = 5

3<--3 = 0

4<--3 = 4 Press Enter to continue...

//For Source Vertex : 4 shortest path to other vertices//

0<--1<--3<--4 = 9

1<--3<--4 = 6

2<--4 = 6

3<--4 = 4

4<--4 = 0 Press Enter to continue...

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

DESCRIPTION:

> Kruskal's algorithm is an algorithm in graph theory that finds a minimum spanning tree for a connectedweighted graph. This means it finds a subset of the edges that forms a tree that includes every vertex, where the total weight of all the edges in the tree is minimized. If the graph is not connected, then it finds a minimum spanning forest (a minimum spanning tree for each connected component). Kruskal'salgorithm is an example of a greedy algorithm

ALGORITHM:

```
Let G = (V, E) be the given graph, with | V| = n
{
     Start with a graph T = (V,$ \phi$) consisting of only the
     vertices of G and no edges; /* This can be viewed as n connected components, each vertex b
eing one connected component */
       Arrange E in the order of increasing costs;
    for (i = 1, i$ \leq$n - 1, i + +)
    {
     Select the next smallest cost edge;
     if (the edge connects two different connected components)
     add the edge to T;
    }
}
```

CODE:

```
/*****************************************************************************
*File          : Kruskal.cpp
*Description   : Program to find Minimum Cost Spanning Tree of a given undirected graph using K
ruskal's algorithm.
*Author        : Prabodh C P
*Compiler      : gcc compiler 4.6.3, Ubuntu 12.04
*Date          : Friday 22 November 2013
*****************************************************************************/
#include <iostream>

using namespace std;

const int MAXNODES = 10;

const int INF = 9999;

// Structure to represent an edge

struct edge

{

   int u, v, cost;

};
```

```
int fnFindParent(int v, int parent[]);

void fnUnion_ij(int i, int j, int parent[]);

void fnInputGraph(int m, edge e[]);

int fnGetMinEdge(edge e[], int n);

void kruskal(int n, edge e[], int m);

/*****************************************************************************
*Function          : main
*Input parameters  :
    *int argc - no of commamd line arguments
    *char **argv - vector to store command line argumennts
*RETURNS           : 0 on success
*****************************************************************************/
int main( int argc, char **argv)

{

    int n = 6, m = 10;
    edge e[2*MAXNODES] = {{0,1,6},{1,4,3},{4,5,6},{5,3,2},{3,0,5},{0,2,1},{1,2,5},{3,2,5},{4,2,
6},{5,2,4}};

cout << "Enter the number of nodes : ";

cin >> n;

cout << "Enter the number of edges : ";

cin >> m;

fnInputGraph(m, e);

kruskal(n, e, m);

return 0;

}

/*****************************************************************************
*Function         : fnFindParent
*Description      : Function to find parent of a given vertex
*Input parameters :
*    int v    - vertex for whom parent has to be found
*    int parent[] - parent vector
*RETURNS          : parent vertex
*****************************************************************************/

int fnFindParent(int v, int parent[])

{

    while (parent[v] != v)
        v = parent[v];

    return v;

}
```

```
/*****************************************************************************
*Function    : fnUnion_ij
*Description    : Function to merge two trees
*Input parameters:
*    int i, j - vertices to be merged
*    int parent[] - parent vector
*RETURNS    : no value
*****************************************************************************/
void fnUnion_ij(int i, int j, int parent[])

{

    if(i < j)
        parent[j] = i;
    else
        parent[i] = j;

}

/*****************************************************************************
*Function         : fnInputGraph
*Description      : Function to read a graph
*Input parameters :
*    int m    - no of edges in the graph
*    edge e[] - set of edges in the graph
*RETURNS          : no value
*****************************************************************************/
void fnInputGraph(int m, edge e[])

{

    int i, j, k, cost;

    for(k=0; k<m; k++)
    {
        cout << "Enter edge and cost in the form u, v, w : \n";
        cin >> i >> j >> cost;

        e[k].u = i;
        e[k].v = j;
        e[k].cost = cost;
    }
}

/*****************************************************************************
*Function         : fnGetMinEdge(
*Description      : Function to find the least cost edge in the edge set
*Input parameters :
*    edge e[] - set of edges in the graph
*    int n    - no of vertices in the graph
*RETURNS          : index of least cost edge in the edge set
*****************************************************************************/

int fnGetMinEdge(edge e[], int n)

{

    int i, small, pos;
    small = INF;
    pos = -1;

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

```
    {
        if(e[i].cost < small)
        {
            small = e[i].cost;
            pos = i;
        }
    }

    return pos;
}

void kruskal(int n, edge e[], int m)

{

    int i, j, count, k, sum, u, v, t[MAXNODES][2], pos;
    int parent[MAXNODES];
    count = 0;
    k = 0;
    sum = 0;

    for(i=0; i<n; i++)
    {
        parent[i] = i;
    }

    while(count != n-1)
    {
        pos = fnGetMinEdge(e,m);
        if(pos == -1)
        {
            break;
        }
        u = e[pos].u;
        v = e[pos].v;
        i = fnFindParent(u,parent);
        j = fnFindParent(v,parent);

        if(i != j)
        {
            t[k][0] = u;
            t[k][1] = v;
            k++;
            count++;
            sum += e[pos].cost;
            fnUnion_ij(i, j, parent);
        }
        e[pos].cost = INF;
    }

    if(count == n-1)
    {
        cout << "\nSpanning tree exists";
        cout << "\nThe Spanning tree is shown below\n";
        for(i=0; i<n-1; i++)
            cout << t[i][0] << " " << t[i][1] << endl;

        cout << "\nCost of the spanning tree : " << sum;
    }
    else
        cout << "\nThe spanning tree does not exist";
}
```

OUTPUT:

Enter the number of nodes : 6
Enter the number of edges : 10
Enter edge and cost in the form u, v, w :
0 1 6
Enter edge and cost in the form u, v, w :
1 4 3
Enter edge and cost in the form u, v, w :
4 5 6
Enter edge and cost in the form u, v, w :
5 3 2
Enter edge and cost in the form u, v, w :
3 0 5
Enter edge and cost in the form u, v, w :
0 2 1
Enter edge and cost in the form u, v, w :
1 2 5
Enter edge and cost in the form u, v, w :
3 2 5
Enter edge and cost in the form u, v, w :
4 2 6
Enter edge and cost in the form u, v, w :
5 2 4
Spanning tree exists
The Spanning tree is shown below
0 2
5 3
1 4
5 2
1 2
Cost of the spanning tree : 15

AIM:

Program to find all nodes reachable from a given node using BFS

DESCRIPTION:

In graph theory, breadth-first search (BFS) is a strategy for searching in a graph when search is limited to essentially two operations:

- visit and inspect a node of a graph;
- gain access to visit the nodes that neighbor the currently visited node.

The BFS begins at a root node and inspects all the neighboring nodes. Then for each of those neighbor nodes in turn, it inspects their neighbor ##nodes which were unvisited, and so on.

ALGORITHM:

Input: A graph G and a root v of G 1 procedure BFS(G,v) is 2 create a queue Q 3 create a set V 4 enqueue v onto Q 5 add v to V 6 while Q is not empty loop 7 t â† Q.dequeue() 8 if t is what we are looking for then 9 return t 10 end if 11 for all edges e in G.adjacentEdges(t) loop 12 u â† G.adjacentVertex(t,e) 13 if u is not in V then 14 add u to V 15 enqueue u onto Q 16 end if 17 end loop 18 end loop 19 return none 20 end BFS

CODE

```
#include <iostream>
#include <cstdlib>

using namespace std;

const int MAX = 100;
void fnBreadthFirstSearchReach(int vertex, int g[MAX][MAX], int v[MAX], int n);

class Queue

{

   private:

       int cQueue[MAX];
       int front, rear;

   public:

       Queue();
       ~Queue();
       int enqueue(int data);
       int dequeue();
       int empty() { return front == -1 ? 1 : 0;  };
};

/*************************************************************************
*Function    : main
*Input parameters: no parameters
*RETURNS    :    0 on success
*************************************************************************/
int main(void)
{
    int i,j;
```

```
    int graph[MAX][MAX];
    int visited[MAX];
    int numVert;
    int startVert;

    cout << "Enter the number of vertices : ";
    cin >> numVert;

    cout << "Enter the adjacency matrix :\n";
    for (i=0; i < numVert; i++)
        visited[i] = 0;

    for (i=0; i<numVert; i++)
        for (j=0; j<numVert; j++)
            cin >> graph[i][j];

    cout << "Enter the starting vertex : ";
    cin >> startVert;

    fnBreadthFirstSearchReach(startVert-1,graph,visited,numVert);

    cout << "Vertices which can be reached from vertex " << startVert << " are :-" << endl;
    for (i=0; i<numVert; i++)
        if (visited[i])
            cout << i+1 << ", ";
    cout << endl;
    return 0;
}

/*Constructor*/
Queue::Queue()
{
    front = rear = -1;
}

/*Destructor*/
Queue::~Queue()
{
}


/*****************************************************************************
*Function    : enqueue
*Description    : Function to insert an element at the rear of a Queue
*Input parameters:
*    int data    - element to be inserted into the queue
*RETURNS        : returns 1 on success and 0 if queue is full
*****************************************************************************/

int Queue::enqueue(int data)

{
    if (front == (rear+1)%MAX)
        return 0;

    if (rear == -1)
        front = rear = 0;
    else
        rear = (rear+1)%MAX;

    cQueue[rear] = data;
    return 1;
```

```
}
/*****************************************************************************
*Function    : dequeue
*Description    : Function to delete an element from the front of a Queue
*Input parameters    : no parameters
*RETURNS        : returns element deleted on success and -1 if queue is empty
*****************************************************************************/

int Queue::dequeue()

{
    int data;

    if (front == -1)
        return -1;

    data = cQueue[front];

    if (front == rear)
        front = rear = -1;
    else
        front = (front+1)%MAX;

    return data;
}
/*****************************************************************************
*Function    : fnBreadthFirstSearchReach
*Description    : Function to perform BFS traversal and mark visited vertices
*Input parameters:
*    int vertex    - source vertex
*    int g[][]    - adjacency matrix of the graph
*    int v[]        - vector to store visited information
    int n        - no of vertices
RETURNS        : void
*****************************************************************************/

void fnBreadthFirstSearchReach(int vertex, int g[MAX][MAX], int v[MAX], int n)
{
    Queue verticesVisited;
    int frontVertex;
    int i;

    v[vertex] = 1;
    verticesVisited.enqueue(vertex);

    while (!verticesVisited.empty())
    {
        frontVertex = verticesVisited.dequeue();
        for (i=0; i<n; i++)
        {
            if (g[frontVertex][i] && !v[i])
            {
                v[i] = 1;
                verticesVisited.enqueue(i);
            }
        }
    }
}
```

OUTPUT

SAMPLE 1

Enter the number of vertices : 4
Enter the adjacency matrix :
0 1 1 0
1 0 0 1
1 0 0 1
0 1 1 0
Enter the starting vertex : 1
Vertices which can be reached from vertex 1 are :-
1, 2, 3, 4,
SAMPLE 2
Enter the number of vertices : 4
Enter the adjacency matrix :
0 1 0 0
1 0 0 0
0 0 0 1
0 0 1 0
Enter the starting vertex : 1
Vertices which can be reached from vertex 1 are :-
1, 2,
SAMPLE 3
Enter the number of vertices : 4
Enter the adjacency matrix :
0 1 0 0
0 0 1 0
0 0 0 1
0 0 0 0
Enter the starting vertex : 2
Vertices which can be reached from vertex 2 are :-
2, 3, 4,
SAMPLE 4
Enter the number of vertices : 4
Enter the adjacency matrix :
0 1 0 0
0 0 1 0
0 0 0 1
1 0 0 0
Enter the starting vertex : 2
Vertices which can be reached from vertex 2 are :-
1, 2, 3, 4,

AIM:Program 7.b. Check whether a given graph is connected or not using DFS method.

DESCRIPTION:

> Depth-first search is a graph traversal algorithm, which has a very wide application area. This algorithm may be used for finding out number of components of a graph, topological order of its nodes or detection of cycles.Itis an algorithm for traversing or searching a tree, tree structure, or graph. One starts at the root (selecting some node as the root in the graph case) and explores as far as possible along each branch before backtracking.

ALGORITHM:

```
DFS(G,v)   ( v is the vertex where the search starts )
        Stack S := {};   ( start with an empty stack )
        for each vertex u, set visited[u] := false;
        push S, v;
        while (S is not empty) do
           u := pop S;
           if (not visited[u]) then
              visited[u] := true;
              for each unvisited neighbour w of u
                 push S, w;
           end if
        end while
     END DFS()
```

CODE:

```
#include <iostream>
#include <cstdlib>
using namespace std;

const int MAX = 100;


void DepthFirstSearch(int currentVertex, int v[MAX], int g[MAX][MAX], int n)
{

    int i;

    v[currentVertex] = 1;

    for (i=0; i<n; i++)
        if (g[currentVertex][i] && !v[i])
            DepthFirstSearch(i,v,g,n);
}

int main()
{

    int i,j,k;
    int visited[MAX];
    int graph[MAX][MAX];
```

```
    int numVert;

    cout << "Enter the number of vertices : ";
    cin >> numVert;

    for (i=0; i<numVert; i++)
        visited[i] = 0;

    cout << "Enter the adjacency matrix :\n";

    for (i=0; i<numVert; i++)
        for (j=0; j<numVert; j++)
            cin >> graph[i][j];

    for (i=0; i<numVert; i++)
    {
        for (k=0; k<numVert; k++)
            visited[k] = 0;

        DepthFirstSearch(i,visited,graph,numVert);

        for (k=0; k<numVert; k++)
        {
            if (!visited[k])
            {
            cout << "\nGraph is not connected since there is no path between " << i << " and "
<< k << endl;

            exit(0);
            }
        }
    }

    cout << "\nGraph is connected."<< endl;
    return 0;
}
```

OUTPUTS

Enter the number of vertices : 4

Enter the adjacency matrix :

0 1 0 0

0 0 1 0

0 0 0 1

1 0 0 0

Graph is connected.

AIM:Program to solve Subset sum problem.

Desrcription:

> In computer science, the subset sum problem is an important problem in complexity theory and cryptography. The problem is this: Given a set of integers, is there a non-empty subset whose sum is zero? For example, given the set {-7, -3, -2, 5, 8}, the answer is yes because the subset {-3, -2, 5} sums to zero. The problem is NP-complete. Input: The number of elements. The Weights of each element. Total Required Weight. Output:Subsets in which the sum of elements is equal to the given required weight(input).

CODE:

```
#include <iostream>
using namespace std;

// Constant definitions
const int MAX = 100;

// class definitions
class SubSet
{
    int stk[MAX], set[MAX];
    int size, top, count;
    public:
    SubSet()
    {
        top = -1;
        count = 0;
    }
    void getInfo(void);
    void push(int data);
    void pop(void);
    void display(void);
    int fnFindSubset(int pos, int sum);
};

/*****************************************************************************
*Function    : getInfo
*Description: Function to read input
*Input parameters: no parameters
*RETURNS    : no value
*****************************************************************************/

void SubSet :: getInfo(void)
{
    int i;
    cout << "Enter the maximum number of elements : ";
    cin >> size;

    cout << "Enter the weights of the elements : \n";
    for (i=1; i<=size; i++)
        cin >> set[i];

}

/*****************************************************************************
```

```
*Function    : push
*Description: Function to push an element on to the stack
*Input parameters:
*int data    - value to be pushed on to the stack
*RETURNS    : no value
*****************************************************************************/
void SubSet :: push(int data)
{
    stk[++top] = data;
}

/*****************************************************************************
*Function    : pop
*Description: Function to pop an element from the stack
*Input parameters: no parameters
*RETURNS    : no value
*****************************************************************************/

void SubSet :: pop(void)
{
    top--;
}

/*****************************************************************************
*Function    : display
*Description: Function to display solution to sub set sum problem
*Input parameters: no parameters
*RETURNS    : no value
*****************************************************************************/
void SubSet :: display()
{
    int i;
    cout << "\nSOLUTION #"<< ++count <<" IS\n{ ";
    for (i=0; i<=top; i++)
        cout << stk[i] << " ";

    cout << "}" << endl;
}

/*****************************************************************************
*Function    : fnFindSubset
*Description    : Function to solve Subset sum problem.
*Input parameters:
*    int pos    - position
*    int sum    - sum of elements
*RETURNS    : returns 1 if solution exists or zero otherwise
*****************************************************************************/
int SubSet :: fnFindSubset(int pos, int sum)
{
    int i;
    static int foundSoln = 0;

    if (sum>0)
    {
        for (i=pos; i<=size; i++)
        {
            push(set[i]);
            fnFindSubset(i+1, sum - set[i]);
            pop();
        }
    }
```

```
    if (sum == 0)
    {
        display();
        foundSoln = 1;
    }

    return foundSoln;
}

/*************************************************************************
*Function    : main
*Input parameters: no parameters
*RETURNS    :    0 on success
*************************************************************************/
int main(void)
{
    int i,sum;

    SubSet set1;

    set1.getInfo();
    cout << "Enter the total required weight : ";
    cin >> sum;

    cout << endl;

    if (!set1.fnFindSubset(1, sum))
        cout << "\n\nThe given problem instance doesnt have any solution." << endl;
    else
        cout << "\n\nThe above-mentioned sets are the required solution to the given instance."
 << endl;

    return 0;
}
```

OUTPUT

SAMPLE 1

Enter the maximum number of elements : 5

Enter the weights of the elements :

1 2 3 4 5

Enter the total required weight : 5

SOLUTION #1 IS

{ 1 4 }

SOLUTION #2 IS

{ 2 3 }

SOLUTION #3 IS

{ 5 }

The above-mentioned sets are the required solution to the given instance.

SAMPLE 2

Enter the maximum number of elements : 4

Enter the weights of the elements :

1 2 3 4

Enter the total required weight : 11

The given problem instance doesnt have any solution.

AIM:

Implement any scheme to find the optimal solution for the TRAVELLING SALESMAN PROBLEM and then solve the same problem instance using any approximation algorithm and determine the error in the approximation.

DESCRIPTION:

> The travelling salesman problem (TSP) asks the following question: Given a list of cities and the distances between each pair of cities, what is the shortest possible route that visits each city exactly once and returns to the origin city? It is an NP-hard problem in combinatorial optimization, important in operations research and theoretical computer science.
> The TSP has several applications even in its purest formulation, such as planning, logistics, and the manufacture of microchips. Slightly modified, it appears as a sub-problem in many areas, such as DNA sequencing. In these applications, the concept city represents, for example, customers, soldering points, or DNA fragments, and the concept distance represents travelling times or cost, or a similarity measure between DNA fragments. In many applications, additional constraints such as limited resources or time windows make the problem considerably harder. TSP is a special case of the travelling purchaser problem.
> In the theory of computational complexity, the decision version of the TSP (where, given a length L, the task is to decide whether the graph has any tour shorter than L) belongs to the class of NP-complete problems. Thus, it is possible that the worst-case running time for any algorithm for the TSP increases superpolynomially (or perhaps exponentially) with the number of cities.

CODE:

```
#include <iostream>
#include <iomanip>

using namespace std;

const int MAX = 10;
int path[MAX];
static int k=0;
int count = 0;
int perm[120][7];
int tourcost[120];

void swap (int *x, int *y)
{
    int temp;
    temp = *x;
    *x = *y;
    *y = temp;
}

void DepthFirstSearch(int currentVertex, int v[MAX], int g[MAX][MAX], int n)
{
     int i;
     v[currentVertex]=1;
     path[k++]=currentVertex;
     for (i=0; i<n; i++)
        if (g[currentVertex][i] && !v[i])
            DepthFirstSearch(i,v,g,n);
}

void permute(int *a, int i, int n)
{
   int j,k;
```

```
   if (i == n)
   {
       for(k=0;k<=n;k++)
       {
       //start from 2nd column
       perm[count][k+1] = a[k];
       }
       count++;
   }
   else
   {
    for (j = i; j <= n; j++)
       {
      swap((a+i), (a+j));
      permute(a, i+1, n);
      swap((a+i), (a+j)); //backtrack
       }
   }
}

int AppTSP(int n, int cost[MAX][MAX])
{
    int i, j, u, v, min,Excost=0;
    int sum, k, t[MAX][2], p[MAX], d[MAX], s[MAX],tree[MAX][MAX];
    int source, count;
    int visited[MAX];
    for (i=0; i<n; i++)
        visited[i] = 0;

    min = 9999;
    source = 0;

    for(i=0; i<n; i++)
    {
        for(j=0; j<n; j++)
        {
            if(cost[i][j] != 0 && cost[i][j] <= min)
            {
                min = cost[i][j];
                source = i;
            }
        }
    }

    for(i=0; i<n; i++)
    {
        d[i] = cost[source][i];
        s[i] = 0;
        p[i] = source;
    }
    s[source] = 1;
    sum = 0;
    k = 0;
    count = 0;

    while (count != n-1)
    {
        min = 9999;
        u = -1;
        for(j=0; j<n; j++)
        {
            if(s[j] == 0)
```

```
            {
                if(d[j] <= min)
                {
                     min = d[j];
                     u = j;
                }
            }
        }

        t[k][0] = u;
        t[k][1] = p[u];
        k++;
        count++;
        sum += cost[u][p[u]];
        s[u] = 1;

        for(v=0; v<n; v++)
        {
            if(s[v]==0 && cost[u][v]<d[v])
            {
                d[v] = cost[u][v];
                p[v] = u;
            }
        }
    }

    for(i=0; i<n; i++)
    {
        for(j=0; j<n; j++)
        {
        tree[i][j]=0;
        }
    }

    if(sum >= 9999)
    cout << "\nSpanning tree does not exist";
    else
    {
    for(i=0; i<k; i++)
    {
        tree[t[i][0]][t[i][1]] = tree[t[i][1]][t[i][0]] =1;
    }
    }

    DepthFirstSearch(0,visited,tree,n);

    cout << "\n The Approximate Minimum Cost tour is" << endl;
    for(i=0;i<=k;i++)
    {
    cout << path[i] << "->";
    Excost += cost[path[i]][path[i+1]];
    }
    cout << path[0];
    Excost += cost[path[i]][path[0]];
    cout << "\n The Approximate Minimum Cost of the tour is" << Excost << endl;
    return Excost;
}

int main(void)
{
    int a[MAX][MAX] = { { 0,  4,  8,  9, 12},
                { 4,  0,  6,  8,  9},
```

```
                { 8,  6,  0, 10, 11},
                { 9,  8, 10,  0,  7},
                {12,  9, 11,  7,  0}};

    int NumOfCity = 5;
    int interCities = 4,i,j;
    int mct=999,mctIndex,Appmct;

    //Source and destination is 0 remaining are intermediary cities
    int city[4] = {1,2,3,4};
    permute(city, 0, interCities-1);
    for(i=0;i<24;i++)
    {
    for(j=0;j<5;j++)
    {
        tourcost[i]+= a[perm[i][j]][perm[i][j+1]];
    }
    if( mct > tourcost[i])
    {
        mct = tourcost[i];
        mctIndex = i;
    }
    }

    cout << "\n The Exact Minimum Cost tour is" << endl;
    for(i=0;i<NumOfCity;i++)
    cout << perm[mctIndex][i] << "->";
    cout << perm[mctIndex][i];
    cout << "\n The Exact Minimum Cost of the tour is" << mct << endl;

    Appmct = AppTSP(NumOfCity,a);
    cout << "\n The error in Approximation is " << Appmct - mct << " units" << endl;
    cout << "\n The Accuracy ratio is " << (float)Appmct / mct << endl;
    cout << "\n The Approximate tour is "<<(((float)Appmct / mct) - 1)*100<< " percent longer t
han the optimal tour" << endl;

    return 0;
}
```

OUTPUT:

The Exact Minimum Cost tour is

0->1->2->4->3->0

The Exact Minimum Cost of the tour is37

The Approximate Minimum Cost tour is

0->1->2->3->4->0

The Approximate Minimum Cost of the tour is39

The error in Approximation is 2 units

The Accuracy ratio is 1.05405

The Approximate tour is 5.40541 percent longer than the optimal tour

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

Description:

> The program uses prim's algorithm which is based on minimum spanning tree for a connected undirected graph.A predefinined cost adjecency matrix is the input.To find the minimum spanning tree, we choose the source node at random and in every step we find the node which is closest as well as having the least cost from the previously selected node.And also the cost of selected edge is being added to variable sum.Based on the value of sum, the presence of the minimum spanning tree is found.

Algorithm:

```
   Input: A non-empty connected weighted graph with vertices V and edges E (the weights can be
negative).
   Initialize: Vnew = {x}, where x is an arbitrary node (starting point) from V, Enew = {}
   Repeat until Vnew = V:
       Choose an edge {u, v} with minimal weight such that u is in Vnew and v is not (if there
are multiple edges with the same weight, any of them may be picked)
       Add v to Vnew, and {u, v} to Enew
   Output: Vnew and Enew describe a minimal spanning tree
```

Code:

```
/*****************************************************************************
*File        : Prim.cpp
*Description    : Program to find Minimum Cost Spanning Tree of a given
*                undirected graph using Prim's algorithm.
*Author        : Prabodh C P
*Compiler    : gcc compiler 4.6.3, Ubuntu 12.04
*Date        : Friday 22 November 2013
*****************************************************************************/

#include<iostream>
using namespace std;

const int MAXNODES = 10;
void fnPrims(int n, int cost[MAXNODES][MAXNODES]);
void fnGetMatrix(int n,int a[MAXNODES][MAXNODES]);

/*****************************************************************************
*Function    : main
*Input parameters:
*    int argc - no of commamd line arguments
*    char **argv - vector to store command line argumennts
*RETURNS    :    0 on success
*****************************************************************************/
int main( int argc, char **argv)

{
    int a[MAXNODES][MAXNODES] = {{0, 3, 9999, 7, 9999},
                {3, 0, 4, 2, 9999},
                {9999, 4, 0, 5, 6},
                {7, 2, 5, 0, 4},
```

```
                {9999, 9999, 6, 4, 0}};

    int n = 5;

    cout << "Enter the number of vertices : ";
    cin >> n;

    fnGetMatrix(n,a);
    fnPrims(n,a);

    return 0;
}

/*****************************************************************************
*Function    : fnPrims
*Description    : Function to find Minimum Cost Spanning Tree of a given
*                undirected graph using Prims algorithm.
*Input parameters:
*    int n    - no of vertices in the graph
*    int cost[][] - cost adjacency matrix of the graph
*RETURNS    : no value
*****************************************************************************/
void fnPrims(int n, int cost[MAXNODES][MAXNODES])
{
    int i, j, u, v, min;
    int sum, k, t[MAXNODES][2], p[MAXNODES], d[MAXNODES], s[MAXNODES];
    int source, count;

    min = 9999;
    source = 0;

    for(i=0; i<n; i++)  //finding the node with minimum cost
    {
        for(j=0; j<n; j++)
        {
            if(cost[i][j] != 0 && cost[i][j] <= min)
            {
                min = cost[i][j];
                source = i;
            }
        }
    }

    for(i=0; i<n; i++)
    {
        d[i] = cost[source][i]; //initializing the array with th cost of all the nodes from sou
rce.
        s[i] = 0;
        p[i] = source;
    }
    s[source] = 1;
    sum = 0;
    k = 0;
    count = 0;

    while (count != n-1)
    {
        min = 9999;
        u = -1;
        for(j=0; j<n; j++)
        {
            if(s[j] == 0)
```

```
            {
                if(d[j] <= min)
                {
                     min = d[j];
                     u = j;
                }
            }
        }

        t[k][0] = u;
        t[k][1] = p[u];
        k++;
        count++;
        sum += cost[u][p[u]];
        s[u] = 1;

        for(v=0; v<n; v++)
        {
            if(s[v]==0 && cost[u][v]<d[v])
            {
                d[v] = cost[u][v];
                p[v] = u;
            }
        }
    }

    if(sum >= 9999)
        cout << "\nSpanning tree does not exist\n";
    else
    {
        cout << "\nThe spanning tree exists and minimum cost spanning tree is \n";
        for(i=0; i<k; i++)
            cout << t[i][1] << " " << t[i][0] << endl;

        cout << "\nThe cost of the minimum cost spanning tree is " << sum << endl;
    }
}

/*************************************************************************
*Function    : fnGetMatrix
*Description    : Function to read cost adjacency matrix of the graph
*Input parameters:
*    int n    - no of vertices in the graph
*    int a[][] - cost adjacency matrix of the graph
*RETURNS    : no value
*************************************************************************/
void fnGetMatrix(int n,int a[MAXNODES][MAXNODES])
{
    int i, j;

    cout << "Enter the Cost Adjacency Matrix" << endl;
    for(i=0; i<n; i++)
        for(j=0; j<n; j++)
            cin >> a[i][j];
}
```

Output Sample 2:

Enter the number of vertices : 5 Enter the Cost Adjacency Matrix

0 3 9999 7 9999

3 0 4 2 9999

9999 4 0 5 6

7 2 5 0 4

9999 9999 6 4 0

The spanning tree exists and minimum cost spanning tree is

3 1

1 0

3 4

1 2

The cost of the minimum cost spanning tree is 13

Output Sample 2:

Enter the number of vertices : 5

Enter the Cost Adjacency Matrix

0 3 9999 7 9999

3 0 9999 2 9999

9999 9999 0 9999 9999

7 2 9999 0 4

9999 9999 9999 4 0

Spanning tree does not exist

AIM:

Implement All-Pairs Shortest Paths Problem using Floyd's algorithm. Parallelize this algorithm, implement it using OpenMP and determine the speed-up achieved.

DESCRIPTION:

> The Floyd's algorithm is a graph analysis algorithm for finding shortest paths in a weighted graph with positive or negative edge weights (but with no negative cycles, see below) and also for finding transitive closure of a relation R. A single execution of the algorithm will find the lengths (summed weights) of the shortest paths between all pairs of vertices, though it does not return details of the paths themselves.

ALGORITHM:

```
let dist be a |V| x |V| array of minimum distances initialized to infinity
for each vertex v
   dist[v][v] <- 0
for each edge (u,v)
   dist[u][v] <- w(u,v)  // the weight of the edge (u,v)
for k from 1 to |V|
   for i from 1 to |V|
      for j from 1 to |V|
         if dist[i][j] > dist[i][k] + dist[k][j]
            dist[i][j] <- dist[i][k] + dist[k][j]
        end if
```

CODE:

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

int min(int,int);
int main()
{
    int n,k,i,j,c[10][10];
    int tid;
    omp_set_num_threads(0);
    {
        tid=omp_get_thread_num();
        printf("Enter the number of nodes:");
        scanf("%d",&n);
        printf("Enter the cost matrix:\n");
        for(i=0;i<n;i++)
        for(j=0;j<n;j++)
              scanf("%d",&c[i][j]);
        for(k=0;k<n;k++)
        {
                for(i=0;i<n;i++)
                   for(j=0;j<n;j++)
                           c[i][j]=min(c[i][j],c[i][k]+c[k][j]);
        }
        printf("\n All pairs shortest path\n");
        for(i=0;i<n;i++)
```

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

int min(int a,int b)

{

  return(a<b?a:b);

}
```

OUTPUT:

Enter the number of nodes:3

Enter the cost matrix:

5 6 7

8 9 1

2 3 4

All pairs shortest path

5 6 7

3 4 1

2 3 4

AIM: Program to solve N Queens problem using backtracking.

DESCRIPTION:

> Given a CHESS BOARD of size N*N,we are supposed to place N QUEEN's such that no QUEEN is in an attacking position.
> BACKTRACKING:Backtracking is a general algorithm for finding all (or some) solutions to some computational problem, that incrementally builds candidates to the solutions, and abandons each partial candidate c ("backtracks") as soon as it determines that c cannot possibly be completed to a valid solution.

ALGORITHM:

```
1) Start in the leftmost column
2) If all queens are placed
    return true
3) Try all rows in the current column.  Do following for every tried row.
    a) If the queen can be placed safely in this row then mark this [row,
    column] as part of the solution and recursively check if placing
    queen here leads to a solution.
    b) If placing queen in [row, column] leads to a solution then return
    true.
    c) If placing queen doesn't lead to a solution then umark this [row,
    column] (Backtrack) and go to step (a) to try other rows.
3) If all rows have been tried and nothing worked, return false to trigger
    backtracking.
```

CODE

```
#include <iostream>
#include <cstdlib>

using namespace std;

const int MAX = 10;

int SolnCount =0;

void fnChessBoardShow(int n, int row[MAX]);
bool fnCheckPlace(int KthQueen, int ColNum, int row[MAX]);
int NQueen(int k,int n, int row[MAX]);

/*************************************************************************
*Function    : main
*Input parameters: no parameters
*RETURNS    :    0 on success
*************************************************************************/
int main(void)

{

    int n;
    int row[MAX];
    cout << "Enter the number of queens : ";
```

```
    cin >> n;
     if (!NQueen(0,n,row))
        cout << "No solution exists for the given problem instance." << endl;
    else
        cout << "Number of solution for the given problem instance is : " << SolnCount << endl;
     return 0;
}
/**************************************************************************
*Function    : NQueen
*Description    : Function to place n queens on a nxn chess board without any
*                    queen attacking any other queen
*Input parameters:
*    int k    -    kth queen
*    int n    - no of queens
*    int row[MAX] - vector containing column numbers of each queen
*RETURNS    : returns 1 if solution exists or zero otherwise
**************************************************************************/

int NQueen(int k,int n, int row[MAX])

{
    static int flag;
    for(int i=0;i<n;i++)
    {
        if(fnCheckPlace(k,i,row) == true)
        {
            row[k] = i;
            if(k == n-1)
            {
                fnChessBoardShow(n,row);
                SolnCount++;
                flag = 1;
                return flag;
            }
            NQueen(k+1, n, row);
        }
    }
    return flag;
}
/**************************************************************************
*Function    : fnCheckPlace
*Description: Function to check whether a kth queen can be palced in a specific
*                   column or not
*Input parameters:
*    int KthQueen    -    kth queen
*    int ColNum        - columnn number
*    int row[MAX]     - vector containing column numbers of each queen
*RETURNS    : returns true if the queen can be palced or false otherwise
**************************************************************************/
bool fnCheckPlace(int KthQueen, int ColNum, int row[MAX])

{

    for(int i=0; i<KthQueen; i++)
    {
        if(row[i] == ColNum || abs(row[i]-ColNum) == abs(i-KthQueen))
            return false;
    }

    return true;
}
```

```
/*************************************************************************
*Function    : fnChessBoardShow
*Description: Function to graphically display solution to n queens problem
*Input parameters:
*    int n    - no of queens
*    int row[MAX]     - vector containing column numbers of each queen
*RETURNS    : no value
*************************************************************************/

void fnChessBoardShow(int n, int row[MAX])

{
    cout << "\nSolution #" << SolnCount+1 << endl << endl;

    for (int i=0; i<n; i++)
    {
        for (int j=0; j<n; j++)
        {
            if (j == row[i])
                cout << "Q ";
           else
                cout << "# ";
        }
        cout << endl;
    }
    cout << endl;

}
```

OUTPUT

SAMPLE 1

Enter the number of queens : 4

Solution #1

Q #

Q

Q # # #

Q

Solution #2

Q

Q # # #

Q

Q #

Number of solution for the given problem instance is : 2

SAMPLE 2

Enter the number of queens : 3 No solution exists for the given problem instance.

Tools Required

ubuntu

This book is done using free softwares only