

VISVESVARAYA TECHNOLOGICAL UNIVERSITY

“JnanaSangama”, Belgaum -590014, Karnataka.



LAB REPORT on

Analysis and Design of Algorithms

Submitted by

SONY MAHATO (1BM21CS217)

in partial fulfillment for the award of the degree of

BACHELOR OF ENGINEERING

in

COMPUTER SCIENCE AND ENGINEERING



B.M.S. COLLEGE OF ENGINEERING

(Autonomous Institution under VTU)

BENGALURU-560019

June-2023 to Sep-2023

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 **SONY MAHATO(1BM21CS217)**, who is a 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 June-2023 to Sep-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.

Sunayana S

Assistant Professor
Department of CSE
BMSCE, Bengaluru

Dr. Jyothi S Nayak

Professor and Head
Department of CSE
BMSCE, Bengaluru

Index Sheet

Lab Program No.	Program Details	Page No.
1	Write program to do the following: 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.	1-6
2	Write program to obtain the Topological ordering of vertices in a given digraph.	7-9
3	Implement Johnson Trotter algorithm to generate permutations.	10-14
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.	15-21
5	Sort a given set of N integer elements using Quick Sort technique and compute its time taken.	22-27
6	Sort a given set of N integer elements using Heap Sort technique and compute its time taken.	28-33
7	Implement 0/1 Knapsack problem using dynamic programming.	34-37
8	Implement All Pair Shortest paths problem using Floyd's algorithm.	38-41
9	Find Minimum Cost Spanning Tree of a given undirected graph using Prim's and Kruskal's algorithm.	42-50
10	From a given vertex in a weighted connected graph, find shortest paths to other vertices using Dijkstra's algorithm.	51-54
11	Implement "N-Queens Problem" using Backtracking.	55-58
12	LeetCode Problems	59-62

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

DFS

```
#include <stdio.h>

void DFS(int);
int isConnected();
int A[10][10], vis[10], n;
int main()
{
    printf("Enter the number of vertices: ");
    scanf("%d", &n);
    printf("Enter Adjacency Matrix\n");
    for (int i = 1; i <= n; i++)
    {
        for (int j = 1; j <= n; j++)
        {
            scanf("%d", &A[i][j]);
        }
    }
    printf("DFS Traversal\n");
    for (int i = 1; i <= n; i++)
    {
        vis[i] = 0;
    }
    DFS(1);
    if(isConnected()==1){
```

```

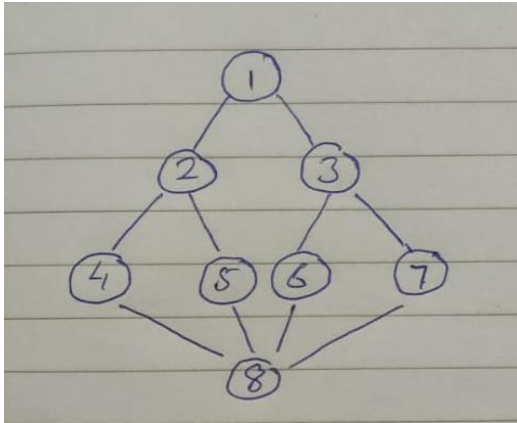
        printf("\nGraph is Conncted.");
    }
    else{
        printf("\nGraph is Not Conncted.");
    }
    return 0;
}

void DFS(int v)
{
    vis[v] = 1;
    printf("%d ", v);
    for (int i = 1; i <= n; i++)
    {
        if (A[v][i] == 1 && vis[i] == 0)
        {
            DFS(i);
        }
    }
}

int isConnected()
{
    for (int i = 1; i <= n; i++)
    {
        if (vis[i] == 0)
            return 0;
    }
    return 1;
}

```

GRAPH:



OUTPUT:

```
Enter the number of vertices: 8
```

```
Enter Adjacency Matrix
```

```
0 1 1 0 0 0 0 0
```

```
0 0 0 1 1 0 0 0
```

```
0 0 0 0 0 1 1 0
```

```
0 0 0 0 0 0 0 1
```

```
0 0 0 0 0 0 0 1
```

```
0 0 0 0 0 0 0 1
```

```
0 0 0 0 0 0 0 1
```

```
0 0 0 0 0 0 0 0
```

```
DFS Traversal:
```

```
>> 1 2 4 8 5 3 6 7
```

```
Graph is Connctd.
```

BFS

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

```
void BFS(int);
```

```
int Q[10],F=-1,R=-1;
```

```
int A[10][10],vis[10];
```

```
int n,m;
```

```
int main(){
```

```
    int v,u,st;
```

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

```
    scanf("%d",&n);
```

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

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

```
            A[i][j]=0;
```

```
        }
```

```
    }
```

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

```
    scanf("%d",&m);
```

```
    printf("Enter the edges\n");
```

```
    for(int i=1;i<=m;i++){
```

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

```
        A[u][v]=1;
```

```
    }
```

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

```
        vis[i]=0;
```



```

    }

    printf("Enter the starting Node\n");
    scanf("%d",&st);
    printf("Nodes reachable from %d\n",st);

    BFS(st);
    return 0;
}

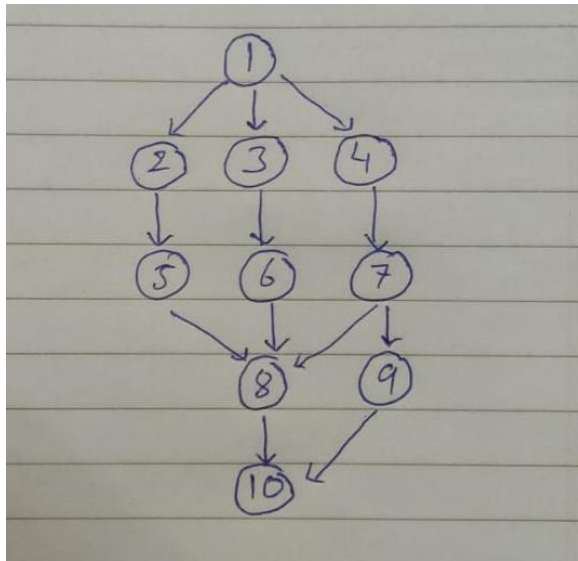
void BFS(int v){
    int u;
    vis[v]=1;
    Q[++R]=v;

    while(F<=R){
        u=Q[++F];
        printf("%d ",u);
        for(int i=1;i<=n;i++){
            if(A[u][i]==1 && vis[i]==0){

                Q[++R]=i;
                vis[i]=1;
            }
        }
    }
}

```

GRAPH:



OUTPUT:

```
Enter the number of vertices: 10
Enter the number of edges: 12
Enter the edges:
1 2
1 3
1 4
2 5
3 6
4 7
5 8
6 8
7 9
7 8
8 10
9 1
Enter the starting Node: 1

Nodes reachable from 1:
1 2 3 4 5 6 7 8 9 10
```

PROGRAM-2

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

```
#include <stdio.h>

void DFS(int);

int A[10][10], vis[10], EXP[10], J = 0;

int n, m;

int main()
{
    int v, u;
    printf("Enter the number of vertices\n");
    scanf("%d", &n);
    for (int i = 1; i <= n; i++)
    {
        for (int j = 1; j <= n; j++)
        {
            A[i][j] = 0;
        }
    }
    printf("Enter the number of edges\n");
    scanf("%d", &m);
    printf("Enter the edges\n");
    for (int i = 1; i <= m; i++)
    {
        scanf("%d %d", &u, &v);
        A[u][v] = 1;
    }
    for (int i = 1; i <= n; i++)
```

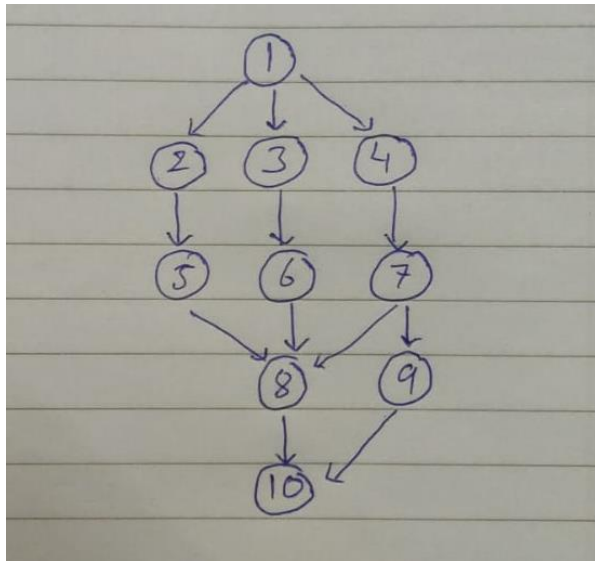
```

    vis[i] = 0;
for (int i = 1; i <= n; i++)
{
    if (vis[i] == 0)
    {
        DFS(i);
    }
}
printf("Topological traversal\n");
for (int i = n - 1; i >= 0; i--)
{
    printf("%d ", EXP[i]);
}
}

void DFS(int v)
{
    int i;
    vis[v] = 1;
    for (int i = 1; i <= n; i++)
    {
        if (A[v][i] == 1 && vis[i] == 0)
        {
            DFS(i);
        }
    }
    EXP[J++] = v;
}

```

GRAPH:



OUTPUT:

```
Enter the number of vertices: 7
Enter the number of edges: 12
Enter the edges:
1 2
1 6
6 5
4 5
2 3
4 3
2 4
7 1
7 6
7 5
7 4
7 2
Topological traversal:
>> 7 1 6 2 4 5 3
```

PROGRAM-3

Implement Johnson Trotter algorithm to generate permutations.

```
#include <stdio.h>

#include <conio.h>

int LEFT_TO_RIGHT = 1;
int RIGHT_TO_LEFT = 0;
int searchArr(int a[], int n, int mobile)
{
    for (int i = 0; i < n; i++)
        if (a[i] == mobile)
            return i + 1;
}

int getMobile(int a[], int dir[], int n)
{
    int mobile_prev = 0, mobile = 0;
    for (int i = 0; i < n; i++)
    {

        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;
else
    return mobile;
}

int 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)
    {
        printf("\n");
        int temp;
        temp = a[pos - 1];

```

```

        a[pos - 1] = a[pos - 2];
        a[pos - 2] = temp;
    }

    else if (dir[a[pos - 1] - 1] == LEFT_TO_RIGHT)
    {
        printf("\n");
        int temp;
        temp = a[pos];
        a[pos] = a[pos - 1];
        a[pos - 1] = temp;
    }

    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]);
}

```



```

int fact(int n)
{
    int res = 1;
    int i;
    for (i = 1; i <= n; i++)
        res = res * i;
    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]);
    }
    for (int i = 0; i < n; i++)
        dir[i] = RIGHT_TO_LEFT;
    for (int i = 1; i < fact(n); i++)
        printOnePerm(a, dir, n);
    printf("\n");
}

int main()

```

```
{  
    int n;  
    printf("\nEnter the value of n: ");  
    scanf("%d", &n);  
    printf("\n");  
    printPermutation(n);  
    printf("\n");  
    return 0;  
}
```

OUTPUT:

```
Enter the value of n: 4
```

```
1 2 3 4  
1 2 4 3  
1 4 2 3  
4 1 2 3  
4 1 3 2  
1 4 3 2  
1 3 4 2  
1 3 2 4  
3 1 2 4  
3 1 4 2  
3 4 1 2  
4 3 1 2  
4 3 2 1  
3 4 2 1  
3 2 4 1  
3 2 1 4  
2 3 1 4  
2 3 4 1  
2 4 3 1  
4 2 3 1  
4 2 1 3  
2 4 1 3  
2 1 4 3  
2 1 3 4
```

PROGRAM-4

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

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

void merge(int arr[], int l, int m, int r)
{
    int i, j, k;
    int n1 = m - l + 1;
    int n2 = r - m;

    int L[n1], R[n2];

    for (i = 0; i < n1; i++)
        L[i] = arr[l + i];
    for (j = 0; j < n2; j++)
        R[j] = arr[m + 1 + j];

    i = 0;
    j = 0;
    k = l;
    while (i < n1 && j < n2)
    {
        if (L[i] <= R[j])
        {
            arr[k] = L[i];
```

```

        i++;
    }
    else
    {
        arr[k] = R[j];
        j++;
    }
    k++;
}

while (i < n1)
{
    arr[k] = L[i];
    i++;
    k++;
}

while (j < n2)
{
    arr[k] = R[j];
    j++;
    k++;
}
}

void mergeSort(int arr[], int l, int r)
{
    if (l < r)

```

```

    {
        int m = 1 + (r - l) / 2;

        mergeSort(arr, l, m);
        mergeSort(arr, m + 1, r);
        merge(arr, l, m, r);
    }
}

int main()
{

    int ch;
    int n;
    int A[100];
    clock_t start_time, end_time;

    printf("\n1.For manual entry of N value and array elements\n2.For Random Values of
N\n3.Exit");

    while (1)
    {
        printf("\nEnter your choice: ");
        scanf("%d", &ch);
        switch (ch)
        {
            case 1:

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

```

```
printf("Enter array elements\n");
for (int i = 0; i < n; i++)
{
    scanf("%d", &A[i]);
}

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

start_time = clock();

mergeSort(A, 0, n - 1);

end_time = clock();
double taken_time = (double)(end_time - start_time) / CLOCKS_PER_SEC;

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

printf("\nTime taken: %f seconds\n", taken_time);
break;
```

case 2:

```
srand(time(NULL));
```

```
int sizes[] = {10, 50, 100, 200, 500, 1000, 2000, 5000, 10000, 20000, 50000, 100000};
```

```
int num_sizes = sizeof(sizes) / sizeof(sizes[0]);
```

```
for (int i = 0; i < num_sizes; i++)
```

```
{
```

```
    int N = sizes[i];
```

```
    int arr[N];
```

```
    for (int j = 0; j < N; j++)
```

```
    {
```

```
        arr[j] = rand() % 1000;
```

```
    }
```

```
    clock_t start = clock();
```

```
    mergeSort(arr, 0, N - 1);
```

```
    clock_t end = clock();
```

```
    double time_taken = ((double)(end - start)) / CLOCKS_PER_SEC;
```

```
    printf("Time taken to sort array of size %d: %lf seconds\n", N, time_taken);
```

```
}
```

```
break;
```

case 3:

```
printf("Exiting the program.\n");
```

```
        exit(0);

    default:
        printf("Invalid choice");
        break;
    }
}

return 0;
}
```


OUTPUT:

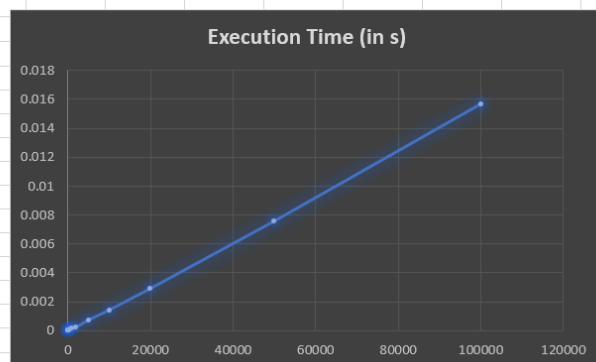
```
1. For manual entry of N value and array elements
2. For Random Values of N
3. Exit
Enter your choice: 1

Enter the number of elements: 6
Enter array elements
1 6 -8 4 2 -3
Array Elements:
1 6 -8 4 2 -3
Sorted Array:
-8 -3 1 2 4 6
Time taken: 0.000003 seconds

Enter your choice: 2
Time taken to sort array of size 10: 0.000003 seconds
Time taken to sort array of size 50: 0.000006 seconds
Time taken to sort array of size 100: 0.000012 seconds
Time taken to sort array of size 200: 0.000025 seconds
Time taken to sort array of size 500: 0.000067 seconds
Time taken to sort array of size 1000: 0.000128 seconds
Time taken to sort array of size 2000: 0.000259 seconds
Time taken to sort array of size 5000: 0.000713 seconds
Time taken to sort array of size 10000: 0.001422 seconds
Time taken to sort array of size 20000: 0.002932 seconds
Time taken to sort array of size 50000: 0.007578 seconds
Time taken to sort array of size 100000: 0.015673 seconds

Enter your choice: 3
Exiting the program.
```

N	Execution Time (in s)
10	0.000003
50	0.000006
100	0.000012
200	0.000025
500	0.000067
1000	0.000128
2000	0.000259
5000	0.000713
10000	0.001422
20000	0.002932
50000	0.007578
100000	0.015673



PROGRAM-5

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

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

int partition(int arr[], int low, int high)
{
    int pivot = arr[high];
    int i = (low - 1);

    for (int j = low; j <= high - 1; j++)
    {
        if (arr[j] < pivot)
        {
            i++;
            int temp = arr[i];
            arr[i] = arr[j];
            arr[j] = temp;
        }
    }

    int temp = arr[i + 1];
    arr[i + 1] = arr[high];
    arr[high] = temp;

    return (i + 1);
}
```

```

}

void quickSort(int arr[], int low, int high)
{
    if (low < high)
    {
        int pi = partition(arr, low, high);

        quickSort(arr, low, pi - 1);
        quickSort(arr, pi + 1, high);
    }
}

int main()
{
    int ch;
    int n;
    int A[100];
    clock_t start_time, end_time;

    printf("\n1.For manual entry of N value and array elements\n2.For Random Values of
N\n3.Exit");

    while (1)
    {
        printf("\nEnter your choice: ");
        scanf("%d", &ch);

        switch (ch)

```

```

{
case 1:

    printf("\nEnter the number of elements: ");
    scanf("%d", &n);
    printf("Enter array elements\n");
    for (int i = 0; i < n; i++)
    {
        scanf("%d", &A[i]);
    }

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

    start_time = clock();
    quickSort(A, 0, n - 1);
    end_time = clock();

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

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

```

```

        break;

case 2:
    srand(time(NULL));

    int sizes[] = {10, 50, 100, 200, 500, 1000, 2000, 5000, 10000, 20000, 50000, 100000};
    int num_sizes = sizeof(sizes) / sizeof(sizes[0]);

    for (int i = 0; i < num_sizes; i++)
    {
        int N = sizes[i];
        int arr[N];

        for (int j = 0; j < N; j++)
        {
            arr[j] = rand() % 1000;
        }

        clock_t start = clock();
        quickSort(arr, 0, N - 1);
        clock_t end = clock();

        printf("Time taken to sort array of size %d: %lf seconds\n", N, ((double)(end - start)) /
CLOCKS_PER_SEC);
    }
    break;

case 3:
    printf("Exiting the program.\n");

```

```
        exit(0);

    default:
        printf("Invalid choice\n");
        break;
    }
}

return 0;
}
```

OUTPUT:

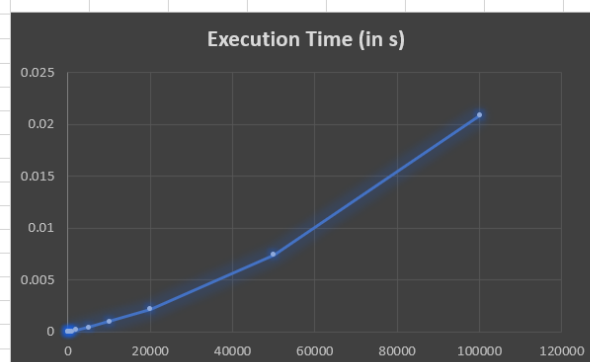
```
1. For manual entry of N value and array elements
2. For Random Values of N
3. Exit
Enter your choice: 1

Enter the number of elements: 7
Enter array elements
6 2 1 5 -4 3 4
Array Elements:
6 2 1 5 -4 3 4
Sorted Array:
-4 1 2 3 4 5 6
Time taken: 0.000001 seconds

Enter your choice: 2
Time taken to sort array of size 10: 0.000001 seconds
Time taken to sort array of size 50: 0.000004 seconds
Time taken to sort array of size 100: 0.000008 seconds
Time taken to sort array of size 200: 0.000015 seconds
Time taken to sort array of size 500: 0.000044 seconds
Time taken to sort array of size 1000: 0.000079 seconds
Time taken to sort array of size 2000: 0.000177 seconds
Time taken to sort array of size 5000: 0.000456 seconds
Time taken to sort array of size 10000: 0.000993 seconds
Time taken to sort array of size 20000: 0.002222 seconds
Time taken to sort array of size 50000: 0.007444 seconds
Time taken to sort array of size 100000: 0.020886 seconds

Enter your choice: 3
Exiting the program.
```

N	Execution Time (in s)
10	0.000001
50	0.000004
100	0.000008
200	0.000015
500	0.000044
1000	0.000079
2000	0.000177
5000	0.000456
10000	0.000993
20000	0.002222
50000	0.007444
100000	0.020886



PROGRAM-6

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

```
#include <stdio.h>
#include <stdlib.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)
    {
        // Swap arr[i] and arr[largest]
        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--)
    {
        // Swap arr[0] and arr[i]
        int temp = arr[0];
        arr[0] = arr[i];
        arr[i] = temp;

        heapify(arr, i, 0);
    }
}

int main()
{
    int ch;
    int n;

```

```

int arr[100];
clock_t start_time, end_time;

printf("\n1.For manual entry of N value and array elements\n2.For Random Values of
N\n3.Exit");

while (1)
{
    printf("\nEnter your choice: ");
    scanf("%d", &ch);

    switch (ch)
    {
        case 1:

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

            printf("Enter array elements: ");
            for (int i = 0; i < n; i++)
            {
                scanf("%d", &arr[i]);
            }

            start_time = clock();
            heapSort(arr, n);
            end_time = clock();

            printf("\nSorted Array: \n");

```

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

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

case 2:

```
srand(time(NULL));
```

```
int sizes[] = {10, 50, 100, 200, 500, 1000, 2000, 5000, 10000, 20000, 50000, 100000};  
int num_sizes = sizeof(sizes) / sizeof(sizes[0]);
```

```
for (int i = 0; i < num_sizes; i++)  
{  
    int N = sizes[i];  
    int arr[N];  
  
    for (int j = 0; j < N; j++)  
    {  
        arr[j] = rand() % 1000;  
    }  
}
```

```
clock_t start = clock();  
heapSort(arr, N);  
clock_t end = clock();
```

```
        printf("Time taken to sort array of size %d: %lf seconds\n", N, ((double)(end - start)) /  
CLOCKS_PER_SEC);  
    }  
    break;  
  
    case 3:  
        printf("Exiting the program.\n");  
        exit(0);  
  
    default:  
        printf("Invalid choice\n");  
        break;  
    }  
}  
  
return 0;  
}
```

OUTPUT:

```
1.For manual entry of N value and array elements
2.For Random Values of N
3.Exit
Enter your choice: 1

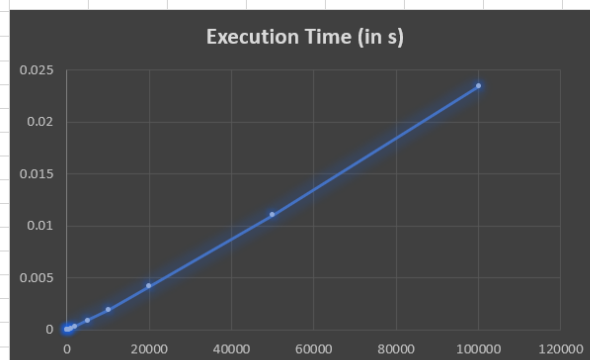
Enter the number of elements: 7
Enter array elements: 2 4 1 -2 5 3 6

Sorted Array:
-2 1 2 3 4 5 6
Time taken: 0.000002 seconds

Enter your choice: 2
Time taken to sort array of size 10: 0.000002 seconds
Time taken to sort array of size 50: 0.000006 seconds
Time taken to sort array of size 100: 0.000012 seconds
Time taken to sort array of size 200: 0.000028 seconds
Time taken to sort array of size 500: 0.000067 seconds
Time taken to sort array of size 1000: 0.000145 seconds
Time taken to sort array of size 2000: 0.000322 seconds
Time taken to sort array of size 5000: 0.000890 seconds
Time taken to sort array of size 10000: 0.001920 seconds
Time taken to sort array of size 20000: 0.004192 seconds
Time taken to sort array of size 50000: 0.011076 seconds
Time taken to sort array of size 100000: 0.023422 seconds

Enter your choice: 3
Exiting the program.
```

N	Execution Time (in s)
10	0.000002
50	0.000006
100	0.000012
200	0.000028
500	0.000067
1000	0.000145
2000	0.000322
5000	0.00089
10000	0.00192
20000	0.004192
50000	0.011076
100000	0.023422



PROGRAM-7

Implement 0/1 Knapsack problem using dynamic programming.

```
#include <stdio.h>

#include <stdlib.h>

int V[100][100];

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

int knapsack(int W, int N, int val[], int wt[])
{
    for (int i = 0; i <= N; i++)
    {
        for (int j = 0; j <= W; j++)
        {
            if (i == 0 || j == 0)
            {
                V[i][j] = 0;
            }
            else if (wt[i - 1] > j)
            {
                V[i][j] = V[i - 1][j];
            }
            else
            {
                V[i][j] = max(V[i - 1][j], V[i - 1][j - wt[i - 1]] + val[i - 1]);
            }
        }
    }
}
```

```

    }
}
return V[N][W];
}
void object_selecetd(int N, int W, int wt[])
{
    int X[N + 1];
    for (int i = 1; i <= N; i++)
    {
        X[i] = 0;
    }
    int i = N;
    int j = W;
    while (i != 0 && j != 0)
    {
        if (V[i][j] != V[i - 1][j])
        {
            X[i] = 1;
            j = j - wt[i - 1];
        }
        i--;
    }

    printf("\n");

    for (int i = 1; i <= N; i++)
    {
        if (X[i] == 1)

```

```

        {
            printf("Object %d Selected\n", i);
        }
    }
}

int main()
{
    int W, N;

    printf("\nEnter number of items: ");
    scanf("%d", &N);
    printf("Enter the Capacity of bag: ");
    scanf("%d", &W);

    int val[W], wt[N];

    for (int i = 0; i < N; i++)
    {
        printf("Enter profit and weight of item %d: ", i + 1);
        scanf("%d%d", &val[i], &wt[i]);
    }

    int result = knapsack(W, N, val, wt);
    object_selectd(N, W, wt);
    printf("\nMaximum profit is: %d", result);
}

```


OUTPUT:

```
Enter number of items: 4
Enter the Capacity of bag: 5
Enter profit and weight of item 1: 12 2
Enter profit and weight of item 2: 10 1
Enter profit and weight of item 3: 20 3
Enter profit and weight of item 4: 15 2
```

```
Object 1 Selected
Object 2 Selected
Object 4 Selected
```

```
Maximum profit is: 37
```

PROGRAM-8

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

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

```
int min(int, int);
```

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

```
{
```

```
    int i, j, k;
```

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

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

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

```
                if (i == j)
```

```
                    p[i][j] = 0;
```

```
                else
```

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

```
}
```

```
int min(int a, int b)
```

```
{
```

```
    if (a < b)
```

```
        return (a);
```

```
    else
```

```
        return (b);
```

```
}
```

```
void main()
```

```
{
```

```

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

printf("\nEnter the number of vertices: ");
scanf("%d", &n);
printf("Enter the number of edges: ");
scanf("%d", &e);

for (i = 1; i <= n; i++)
{
    for (j = 1; j <= n; j++)
        p[i][j] = 999;
}

for (i = 1; i <= e; i++)
{
    printf("\nEnter the end vertices of edge %d: ", i);
    scanf("%d%d", &u, &v);
    printf("Enter Weight: ");
    scanf("%d",&w);
    p[u][v] = w;
}

printf("\nAdjacency Matrix: \n");
for (i = 1; i <= n; i++)
{
    for (j = 1; j <= n; j++)
        printf("%d \t", p[i][j]);
    printf("\n");
}

```

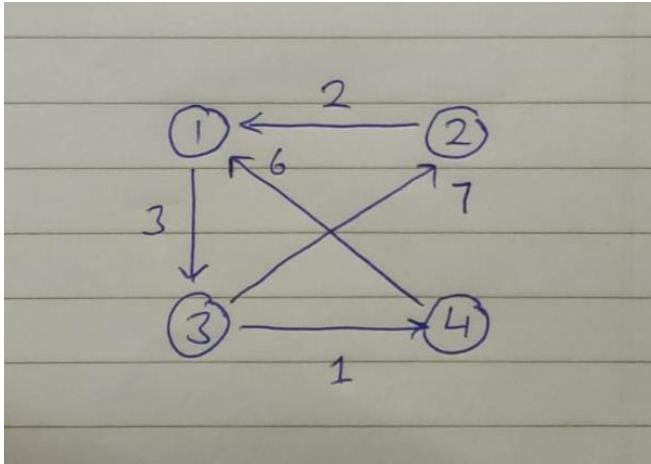
```
}

floyds(p, n);

printf("\nPath Matrix: \n");
for (i = 1; i <= n; i++)
{
    for (j = 1; j <= n; j++)
        printf("%d \t", p[i][j]);
    printf("\n");
}

}
```

GRAPH:



OUTPUT:

```
Enter the number of vertices: 4
Enter the number of edges: 5

Enter the end vertices of edge 1: 1 3
Enter Weight: 3

Enter the end vertices of edge 2: 2 1
Enter Weight: 2

Enter the end vertices of edge 3: 3 4
Enter Weight: 1

Enter the end vertices of edge 4: 3 2
Enter Weight: 7

Enter the end vertices of edge 5: 4 1
Enter Weight: 6
```

Adjacency Matrix:

999	999	3	999
2	999	999	999
999	7	999	1
6	999	999	999

Path Matrix:

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

PROGRAM-9

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

Prim's algorithm

```
#include <stdio.h>

int n, m, e = 0;

float sum = 0;

float costs[100][100];

int VT[100], ET[100][2], vis[20];

void prims()
{
    int u, v;
    int x = 1, j, K, min;
    VT[x] = 1;
    vis[x] = 1;
    for (int i = 1; i < n; i++)
    {
        j = x;
        min = 999;
        while (j > 0)
        {
            K = VT[j];
            for (int m = 2; m <= n; m++)
            {
                if (costs[K][m] < min && vis[m] == 0)
                {
```

```

        min = costs[K][m];
        u = K;
        v = m;
    }
}
j--;
}
VT[++x] = v;
ET[i][0] = u;
ET[i][1] = v;
e++;
vis[v] = 1;
sum += costs[u][v];
}
}

void main()
{
    printf("\n    Prim's Algorithm\n");
    printf("    -----");
    int u, v;
    float w;
    printf("\nEnter the number of vertices: ");
    scanf("%d", &n);

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

```

```

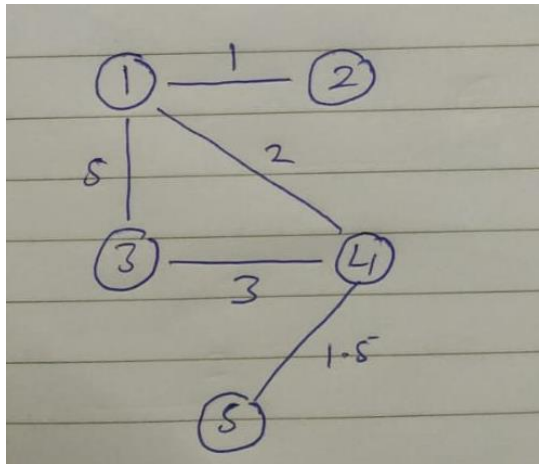
    {
        if (i == j)
            costs[i][j] = 0;
        else
            costs[i][j] = 999;
    }
}

printf("Enter the number of egdes: ");
scanf("%d", &m);

printf("Enter vertices of edge with its weight: \n");
for (int i = 1; i <= m; i++)
{
    scanf("%d%d%f", &u, &v, &w);
    costs[u][v] = costs[v][u] = w;
}
for (int i = 1; i <= n; i++)
{
    vis[i] = 0;
}
prims();
printf("\nMinimum Cost: %.2f\n", sum);
printf("\nEdges of Minimum spanning tree\n");
for (int i = 1; i <= e; i++)
{
    printf("%d-->%d\n", ET[i][0], ET[i][1]);
}
}

```


GRAPH:



OUTPUT:

```
Prim's Algorithm
-----
Enter the number of vertices: 5
Enter the number of egdes: 5
Enter vertices of edge with its weight:
1 2 1
1 3 5
1 4 2
3 4 3
4 5 1.5

Minimum Cost: 7.50

Edges of Minimum spanning tree
1-->2
1-->4
4-->5
4-->3
```

Kruskal's algorithm

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

int n, m, parent[100];
int count = 0;
int ET[100][2];
int cost[100][100];
int sum = 0;

void unionn(int a, int b)
{
    if (a < b)
        parent[b] = a;
    else
        parent[a] = b;
}

int find(int a)
{
    while (parent[a] != a)
    {
        a = parent[a];
    }
    return a;
}
```

```

void kruskal()
{
    int k = 0;
    for (int i = 1; i <= n; i++)
    {
        parent[i] = i;
    }
    while (count != n - 1)
    {
        int min = 999;
        int u, v;
        for (int i = 1; i <= n; i++)
        {
            for (int j = 1; j <= n; j++)
            {
                if (cost[i][j] < min && cost[i][j] != 0)
                {
                    min = cost[i][j];
                    u = i;
                    v = j;
                }
            }
        }

        int x = find(u);
        int y = find(v);

        if (x != y)

```

```

    {
        ET[k][0] = u;
        ET[k][1] = v;
        k++;
        count++;
        sum += cost[u][v];
        unionn(x, y);
    }

    cost[u][v] = cost[v][u] = 999;
}
}

int main()
{
    printf("\n    Kruskal's algorithm\n");
    printf("    -----");
    int u, v, w;
    printf("\nEnter the number of vertices: ");
    scanf("%d", &n);

    for (int i = 1; i <= n; i++)
    {
        for (int j = 1; j <= n; j++)
        {
            if (i == j)
                cost[i][j] = 0;
            else

```

```

        cost[i][j] = 999;
    }
}

printf("Enter the number of edges: ");
scanf("%d", &m);

printf("Enter the egde with its weight: \n");
for (int i = 1; i <= m; i++)
{
    scanf("%d%d%d", &u, &v, &w);
    cost[u][v] = cost[v][u] = w;
}

kruskal();

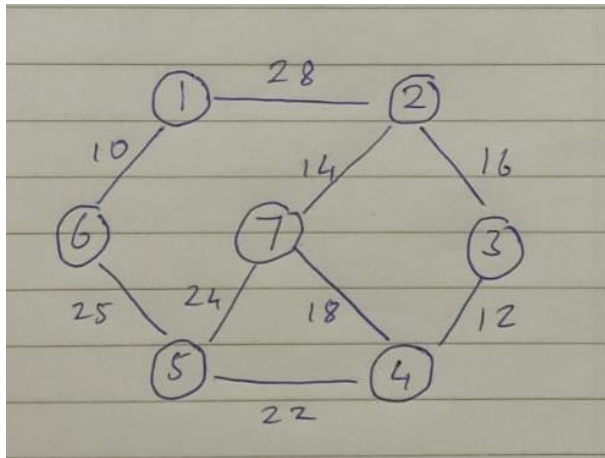
printf("\nMinimum cost = %d\n", sum);

printf("Minimum spanning tree:\n");
for (int i = 1; i < count; i++)
{
    printf("%d -> %d\n", ET[i][0], ET[i][1]);
}

return 0;
}

```

GRAPH:



OUTPUT:

Kruskal's algorithm

Enter the number of vertices: 7

Enter the number of edges: 9

Enter the edge with its weight:

1 2 28

1 6 10

2 7 14

2 3 16

3 4 12

4 7 18

4 5 22

5 7 24

5 6 25

Minimum cost = 99

Minimum spanning tree:

3 -> 4

2 -> 7

2 -> 3

4 -> 5

5 -> 6

PROGRAM-10

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

```
#include <stdio.h>

int dist[10], cost[100][100], n, vis[10], src;

void dijkstra()
{
    int count, min, u;
    for (int i = 1; i <= n; i++)
    {
        dist[i] = cost[src][i];
        vis[src] = 1;
    }
    count = 1;
    while (count < n)
    {
        min = 9999;
        for (int i = 1; i <= n; i++)
        {
            if (dist[i] < min && vis[i] == 0)
            {
                min = dist[i];
                u = i;
            }
        }
        vis[u] = 1;
        for (int i = 1; i <= n; i++)
        {
```

```

        if (dist[u] + cost[u][i] < dist[i] && vis[i] == 0)
        {
            dist[i] = dist[u] + cost[u][i];
        }
    }
    count++;
}
}

```

```

void main()
{
    int m, u, v, w;
    printf("\n    Dijkstra's Algorithm\n");
    printf("    -----");
    printf("\nEnter the number of vertices: ");
    scanf("%d", &n);
    for (int i = 1; i <= n; i++)
    {
        for (int j = 1; j <= n; j++)
        {
            if (i == j)
            {
                cost[i][j] = 0;
            }
            else
            {
                cost[i][j] = 9999;
            }
        }
    }
}

```



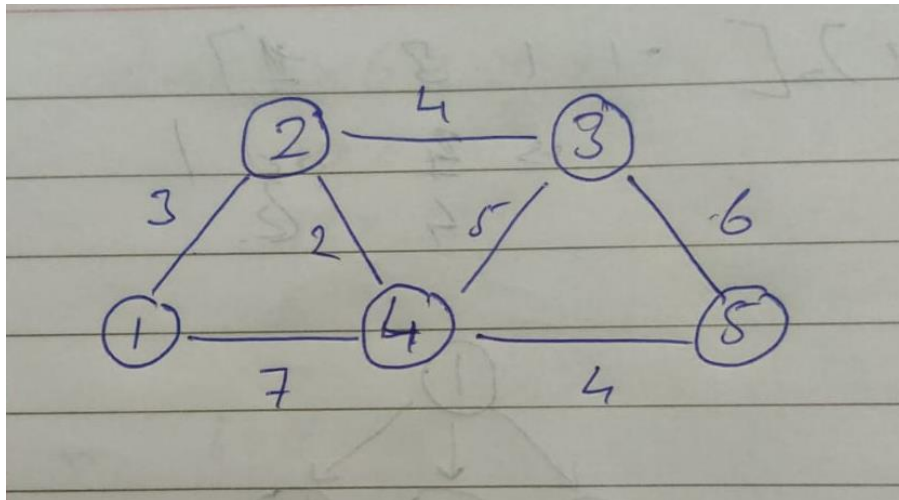
```

    }
}
printf("Enter the number of edges: ");
scanf("%d", &m);
printf("Enter the edge with its weight\n");
for (int i = 1; i <= m; i++)
{
    scanf("%d%d%d", &u, &v, &w);
    cost[v][u] = cost[u][v] = w;
}
printf("Enter the source\n");
scanf("%d", &src);
dijkstra();

printf("\n");
for (int i = 2; i <= n; i++)
    printf("The distance from %d --> %d is %d\n", src, i, dist[i]);
}

```

GRAPH:



OUTPUT:

```
Dijkstra's Algorithm
-----
Enter the number of vertices: 5
Enter the number of edges: 7
Enter the edge with its weight
1 2 3
1 4 7
2 3 4
2 4 2
3 4 5
3 5 6
4 5 4
Enter the source
1

The distance from 1 --> 2 is 3
The distance from 1 --> 3 is 7
The distance from 1 --> 4 is 5
The distance from 1 --> 5 is 9
```

PROGRAM-11

Implement “N-Queens Problem” using Backtracking.

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

```
int n, count=0;
```

```
int isSafe(char board[n][n], int row, int col)
```

```
{  
    for (int i = row - 1; i >= 0; i--)  
    {  
        if (board[i][col] == 'Q')  
        {  
            return 0;  
        }  
    }  
}
```

```
for (int i = row - 1, j = col - 1; i >= 0 && j >= 0; i--, j--)  
{  
    if (board[i][j] == 'Q')  
    {  
        return 0;  
    }  
}
```

```
for (int i = row - 1, j = col + 1; i >= 0 && j < n; i--, j++)  
{  
    if (board[i][j] == 'Q')
```

```

        {
            return 0;
        }
    }
    return 1;
}

void printBoard(char board[][n])
{
    printf("\n---Chess Board---\n");

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

void nQueens(char board[n][n], int row)
{
    if (row == n)
    {
        printBoard(board);
        count++;
        return;
    }
}

```

```

    }
    for (int j = 0; j < n; j++)
    {
        if (isSafe(board, row, j) == 1)
        {
            board[row][j] = 'Q';
            nQueens(board, row + 1);
            board[row][j] = 'X';
        }
    }
}

int main()
{
    printf("Enter the size of the board: ");
    scanf("%d", &n);
    char board[n][n];
    for (int i = 0; i < n; i++)
    {
        for (int j = 0; j < n; j++)
        {
            board[i][j] = 'X';
        }
    }
    nQueens(board, 0);
    printf("\nTotal Possible Solution: %d ",count);
}

```

OUTPUT:

```
Enter the size of the board: 4
```

```
---Chess Board---
```

```
X Q X X
```

```
X X X Q
```

```
Q X X X
```

```
X X Q X
```

```
---Chess Board---
```

```
X X Q X
```

```
Q X X X
```

```
X X X Q
```

```
X Q X X
```

```
Total Possible Solution: 2
```

LeetCode Problems

I.

The screenshot shows the LeetCode problem page for "73. Set Matrix Zeroes". The problem description states: "Given an $m \times n$ integer matrix `matrix`, if an element is 0, set its entire row and column to 0's. You must do it in place." An example is provided with a 3x3 matrix:

1	1	1
1	0	1
1	1	1

 which is transformed into

1	0	1
0	0	0
1	0	1

. The input is `matrix = [[1,1,1],[1,0,1],[1,1,1]]` and the output is `[[1,0,1],[0,0,0],[1,0,1]]`. The solution editor shows a Java implementation using dummy rows and columns to track zeros.

```
1 class Solution {
2     public void setZeroes(int[][] matrix) {
3         int row = matrix.length;
4         int col = matrix[0].length;
5         int[] dummyRow = new int[row];
6         int[] dummyCol = new int[col];
7         Arrays.fill(dummyRow, -1);
8         Arrays.fill(dummyCol, -1);
9         for(int i=0; i<row; i++){
10             for(int j=0; j<col; j++){
11                 if(matrix[i][j]==0){
12                     dummyRow[i] = 0;
13                     dummyCol[j] = 0;
14                 }
15             }
16         }
17         for(int i=0; i<row; i++){
18             for(int j=0; j<col; j++){
19                 if(dummyRow[i] == 0 || dummyCol[j] == 0){
20                     matrix[i][j] = 0;
21                 }
22             }
23         }
24     }
25 }
```

The screenshot shows the submission page for the "Set Matrix Zeroes" problem. The submission status is "Accepted". The table below shows the submission details:

Status	Language	Runtime	Memory	Time	Notes
Accepted	Java	1 ms	44.9 MB	Jun 13, 2023	

The solution editor shows the same Java code as in the previous screenshot.

II.

Problem List

1971. Find if Path Exists in Graph

Easy

3.2K

155

☆

🔗

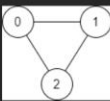
Companies

There is a **bi-directional** graph with n vertices, where each vertex is labeled from 0 to $n - 1$ (**inclusive**). The edges in the graph are represented as a 2D integer array `edges`, where each `edges[i] = [ui, vi]` denotes a bi-directional edge between vertex `ui` and vertex `vi`. Every vertex pair is connected by **at most one** edge, and no vertex has an edge to itself.

You want to determine if there is a **valid path** that exists from vertex `source` to vertex `destination`.


Given `edges` and the integers `n`, `source`, and `destination`, return `true` if there is a **valid path** from `source` to `destination`, or `false` otherwise.

Example 1:



Input: `n = 3, edges = [[0,1],[1,2],[2,0]], source = 0, destination = 2`
Output: `true`
Explanation: There are two paths from vertex 0 to vertex 2:
 - `0 → 1 → 2`
 - `0 → 2`

Example 2:



```

1 class Solution {
2     public boolean validPath(int n, int[][] edges, int source, int destination) {
3         DS d = new DS(n);
4         for (int i = 0; i < edges.length; i++) {
5             d.union(edges[i][0], edges[i][1]);
6         }
7         return d.areTheyConnected(source, destination);
8     }
9 }
10 class DS {
11     int P[];
12
13     public DS(int n) {
14         P = new int[n];
15
16         for (int i = 0; i < n; i++) {
17             P[i] = i;
18         }
19     }
20
21     public boolean areTheyConnected(int u, int v) {
22         return find(u) == find(v);
23     }
24
25     private int find(int x) {
26         if (x == P[x]) {
27             return x;
28         } else {
29             return find(P[x]);
30         }
31     }
32
33     private void union(int x, int y) {
34         int X = find(x);
35         int Y = find(y);
36
37         if (X == Y) {
38             return;
39         } else {
40             P[Y] = X;
41         }
42     }
43 }

```

4th Sem ADA June-Sep 2023 D

LeetCode - The World's Leading

+

← → ↺

leetcode.com/problems/find-if-path-exists-in-graph/submissions/

🔍 📌 ☆ ⚙️ 🏠 👤

Problem List

1971. Find if Path Exists in Graph

Accepted

Java

33 ms

100.2 MB

Jun 20, 2023

Solutions (2K)

Submissions

```

1 class Solution {
2     public boolean validPath(int n, int[][] edges, int source, int destination) {
3         DS d = new DS(n);
4         for (int i = 0; i < edges.length; i++) {
5             d.union(edges[i][0], edges[i][1]);
6         }
7         return d.areTheyConnected(source, destination);
8     }
9 }
10 class DS {
11     int P[];
12
13     public DS(int n) {
14         P = new int[n];
15
16         for (int i = 0; i < n; i++) {
17             P[i] = i;
18         }
19     }
20
21     public boolean areTheyConnected(int u, int v) {
22         return find(u) == find(v);
23     }
24
25     private int find(int x) {
26         if (x == P[x]) {
27             return x;
28         } else {
29             return find(P[x]);
30         }
31     }
32
33     private void union(int x, int y) {
34         int X = find(x);
35         int Y = find(y);
36
37         if (X == Y) {
38             return;
39         } else {
40             P[Y] = X;
41         }
42     }
43 }

```


III.

Problem List

112. Path Sum

Easy

8.9K

998

Companies

Given the `root` of a binary tree and an integer `targetSum`, return `true` if the tree has a **root-to-leaf** path such that adding up all the values along the path equals `targetSum`.

A **leaf** is a node with no children.

Example 1:

```
graph TD; 5((5)) --- 4L((4)); 5 --- 8((8)); 4L --- 11((11)); 4L --- 13((13)); 11 --- 7((7)); 11 --- 2((2)); 8 --- 4R((4)); 8 --- 1((1));
```

Input: `root = [5,4,8,11,null,13,4,7,2,null,null,null,1]`, `targetSum = 22`
Output: `true`
Explanation: The root-to-leaf path with the target sum is shown.

Console

Run Submit

C

Auto

1

2

3

4

5

6 `bool hasPathSum(struct TreeNode *root, int targetSum)`

7 `{`

8 `if(!root)`

9 `return 0;`

10 `targetSum -= root->val;`

11 `return (!targetSum && !root->right && !root->left) ||`

12 `hasPathSum(root->left, targetSum) || hasPathSum(root->right, targetSum);`

13 `}`

Ln 13, Col 2

4th Sem ADA June-Sep 2023 D

LeetCode - The World's Leading

+

← → ↺

leetcode.com/problems/path-sum/submissions/

🔍 📄 ⭐ ⚙️ 🏠 👤

Problem List

112. Path Sum

Easy

8.9K

998

Companies

Given the `root` of a binary tree and an integer `targetSum`, return `true` if the tree has a **root-to-leaf** path such that adding up all the values along the path equals `targetSum`.

A **leaf** is a node with no children.

Example 1:

```
graph TD; 5((5)) --- 4L((4)); 5 --- 8((8)); 4L --- 11((11)); 4L --- 13((13)); 11 --- 7((7)); 11 --- 2((2)); 8 --- 4R((4)); 8 --- 1((1));
```

Input: `root = [5,4,8,11,null,13,4,7,2,null,null,null,1]`, `targetSum = 22`
Output: `true`
Explanation: The root-to-leaf path with the target sum is shown.

Console

Run Submit

C

Auto

1

2

3

4

5

6 `bool hasPathSum(struct TreeNode *root, int targetSum)`

7 `{`

8 `if(!root)`

9 `return 0;`

10 `targetSum -= root->val;`

11 `return (!targetSum && !root->right && !root->left) ||`

12 `hasPathSum(root->left, targetSum) || hasPathSum(root->right, targetSum);`

13 `}`

Ln 13, Col 2

Status	Language	Runtime	Memory	Time	Notes
Accepted	C	8 ms	8.1 MB	Jul 11, 2023	

IV.

221. Maximal Square

Medium

Given an $m \times n$ binary matrix filled with 0's and 1's, find the largest square containing only 1's and return its area.

Example 1:

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

Input: matrix = `[["1","0","1","0","0"],["1","0","1","1","1"],["1","1","1","1","1"],["1","0","0","1","0"]]`
Output: 4

Example 2:

0	1
---	---

```
1 class Solution
2 {
3     public int maximalSquare(char[][] a) {
4         if (a == null || a.length == 0 || a[0].length == 0)
5             return 0;
6         int max = 0, n = a.length, m = a[0].length;
7         int[][] dp = new int[n + 1][m + 1];
8         for (int i = 1; i <= n; i++)
9         {
10             for (int j = 1; j <= m; j++)
11             {
12                 if (a[i - 1][j - 1] == '1')
13                 {
14                     dp[i][j] = Math.min(dp[i - 1][j - 1], Math.min(dp[i - 1][j], dp[i][j - 1])) + 1;
15                     max = Math.max(max, dp[i][j]);
16                 }
17             }
18         }
19         return max * max;
20     }
21 }
```

4th Sem ADA June-Sep 2023 D... x LeetCode - The World's Leading x +

leetcode.com/problems/maximal-square/submissions/

Submissions

Status	Language	Runtime	Memory	Time	Notes
Accepted	Java	6 ms	54.4 MB	Jul 25, 2023	

```
1 class Solution
2 {
3     public int maximalSquare(char[][] a) {
4         if (a == null || a.length == 0 || a[0].length == 0)
5             return 0;
6         int max = 0, n = a.length, m = a[0].length;
7         int[][] dp = new int[n + 1][m + 1];
8         for (int i = 1; i <= n; i++)
9         {
10             for (int j = 1; j <= m; j++)
11             {
12                 if (a[i - 1][j - 1] == '1')
13                 {
14                     dp[i][j] = Math.min(dp[i - 1][j - 1], Math.min(dp[i - 1][j], dp[i][j - 1])) + 1;
15                     max = Math.max(max, dp[i][j]);
16                 }
17             }
18         }
19         return max * max;
20     }
21 }
```