

NAME: NAGAVENI L G

SRN: PES2UG21CS315

SECTION: F

Course Outcomes

At the end of the course, the student will be able to:

- CO1. Identify the design technique used in an algorithm.
- CO2. Analyse algorithms using quantitative evaluation.
- CO3. Design and implement efficient algorithms for practical and unseen problems.
- CO4. Analyse time efficiency over trading space.
- CO5. Understand the limits of algorithms and the ways to cope with the limitations.

Question	Course Outcome	Topic
Q1		Fibonacci
Q2		Floyd-Warshall
Q3		Floyd

Ticking Session 1

Q1. A mathematician tries to find a new series by using the logic of the existing fibonacci series. To get the next number in the sequence He multiplies the previous number that occurred in the sequence with the 3rd last term which was in the sequence before the new term joined the sequence. He uses the first 3 natural numbers as the first 3 terms in the sequence. Find the nth number in the sequence. (Use Dynamic Programming approach).
 $1 \leq n \leq 13$

Sample Input 1:
6 // nth term

Expected Output 1:
18

Boilerplate/Skeleton Code:

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

```
long long operation(int n)
{
    // Write your code here
}
```

```
// Driver's code
int main()
```

```

{
    int n;
    scanf("%d",&n);
    printf("%lld\n", operation(n));
    return 0;
}

```

```

#include <stdio.h>

Long Long operation(int n)
{
    // Write your code here
    Long Long array[n];
    array[0] = 1, array[1] = 2, array[2] = 3;
    for(int i = 3; i < n; i++){
        array[i] = array[i - 1] * array[i - 3];
    }
    return array[n - 1];
}

// Driver's code
int main()
{
    int n;
    scanf("%d",&n);
    printf("%lld\n", operation(n));
    return 0;
}

```

Test cases:

1.
2

Output -

2

2.

13

Output -

187406683791040512

3.

10

Output -

314928

```

PS C:\Users\Praka\OneDrive\Documents\DAA> gcc sample.c
PS C:\Users\Praka\OneDrive\Documents\DAA> ./a
6
18
PS C:\Users\Praka\OneDrive\Documents\DAA> ./a
2
2
PS C:\Users\Praka\OneDrive\Documents\DAA> ./a
10
314928
PS C:\Users\Praka\OneDrive\Documents\DAA> ./a
13
187406683791040512
PS C:\Users\Praka\OneDrive\Documents\DAA> 

```

Q2. N number of cities in a state is represented using an adjacency matrix "path" where each element $\text{path}[i][j]$ (i and j are the index city i and city j) represents the distance from city i to city j. Ram wants to know the total number of intermediate cities he has to go through to reach any city if he is starting from city "src" and the distance travelled is not more than 500. Print the total number of intermediate cities he has to go through to reach each of the destination cities from src, print -1 if it's not possible to reach a city.

Sample Input 1:

```

4      //Number of cities N
0      //src
0 500 400 10    //adjacency matrix "path"
500 0 1000 1000
400 1000 0 20
10 1000 20 0

```

Expected Output 1:

```

0
-1
1
0

```

```
PS C:\Users\Praka\OneDrive\Documents\DAA> ./a
4
0
0 500 400 10
500 0 1000 1000
400 1000 0 20
10 1000 20 0
0
-1
1
0
PS C:\Users\Praka\OneDrive\Documents\DAA> █
```

Boilerplate/Skeleton Code:

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

```
void solution(int src,int n,
              int path[n][n])
{
    //Write your solution here
}
```

```
void main(){
```

```
//driver's code
```

```

int n;int src;
scanf("%d",&n);
scanf("%d",&src);
int (*adj)[n]=malloc(sizeof(int)*n*n);
for(int i=0;i<n;i++){
    for(int j=0;j<n;j++){
        scanf("%d",&adj[i][j]);
    }
}

solution(src,n,adj);
}

```

```

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

#define INF 500

void solution(int src, int n, int path[n][n]) {
    int dist[n][n];
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            dist[i][j] = path[i][j];
        }
    }

    for (int k = 0; k < n; k++) {
        for (int i = 0; i < n; i++) {
            for (int j = 0; j < n; j++) {
                if (dist[i][k] != INF && dist[k][j] != INF && dist[i][k] + dist[k][j] <
dist[i][j]) {
                    dist[i][j] = dist[i][k] + dist[k][j];
                }
            }
        }
    }

    for (int i = 0; i < n; i++) {
        if (i == src) {
            printf("0\n");
        } else if (dist[src][i] == INF) {
            printf("-1\n");
        } else if (dist[src][i] <= 500) {
            int intermediate = 0;
            for (int j = 0; j < n; j++) {
                if (j != src && j != i && dist[src][j] < INF && dist[j][i] < INF &&
dist[src][j] + dist[j][i] == dist[src][i]) {
                    intermediate++;
                }
            }
            printf("%d\n", intermediate);
        } else {
            printf("-1\n");
        }
    }
}

```

```

    }
}

void main(){
    //driver's code
    int n;int src;
    scanf("%d",&n);
    scanf("%d",&src);
    int (*adj)[n]=malloc(sizeof(int)*n*n);
    for(int i=0;i<n;i++){
        for(int j=0;j<n;j++){
            scanf("%d",&adj[i][j]);
        }
    }

    solution(src,n,adj);
}

```

Test cases:

```

1.
5
0
0 500 400 10 1000
500 0 1000 1000 1000
400 1000 0 20 200
10 1000 20 0 1000
1000 1000 200 1000 0

```

Output -

```

0
-1
1
0
2
PS C:\Users\Praka\OneDrive\Documents\DAA> ./a
5
0
0 500 400 10 1000
500 0 1000 1000 1000
400 1000 0 20 200
10 1000 20 0 1000
1000 1000 200 1000 0
0
-1
1
0
2
PS C:\Users\Praka\OneDrive\Documents\DAA>

```

2.

```
5
1
0 500 400 10 1000
500 0 1000 1000 1000
400 1000 0 20 200
10 1000 20 0 1000
1000 1000 200 1000 0
```

Output -

```
-1
0
```

```
-1
-1
-1
```

```
PS C:\Users\Praka\OneDrive\Documents\DAA> ./a
5
1
0 500 400 10 1000
500 0 1000 1000 1000
400 1000 0 20 200
10 1000 20 0 1000
1000 1000 200 1000 0
-1
0
-1
-1
-1
PS C:\Users\Praka\OneDrive\Documents\DAA> █
```

3.

5

2

0 500 400 10 1000

500 0 1000 1000 1000

400 1000 0 20 200

10 1000 20 0 1000

1000 1000 200 1000 0

Output -

1

-1

0

0

0

```
PS C:\Users\Praka\OneDrive\Documents\DAA> ./a
```

5

2

0 500 400 10 1000

500 0 1000 1000 1000

400 1000 0 20 200

10 1000 20 0 1000

1000 1000 200 1000 0

1

-1

0

0

0

```
PS C:\Users\Praka\OneDrive\Documents\DAA> 
```

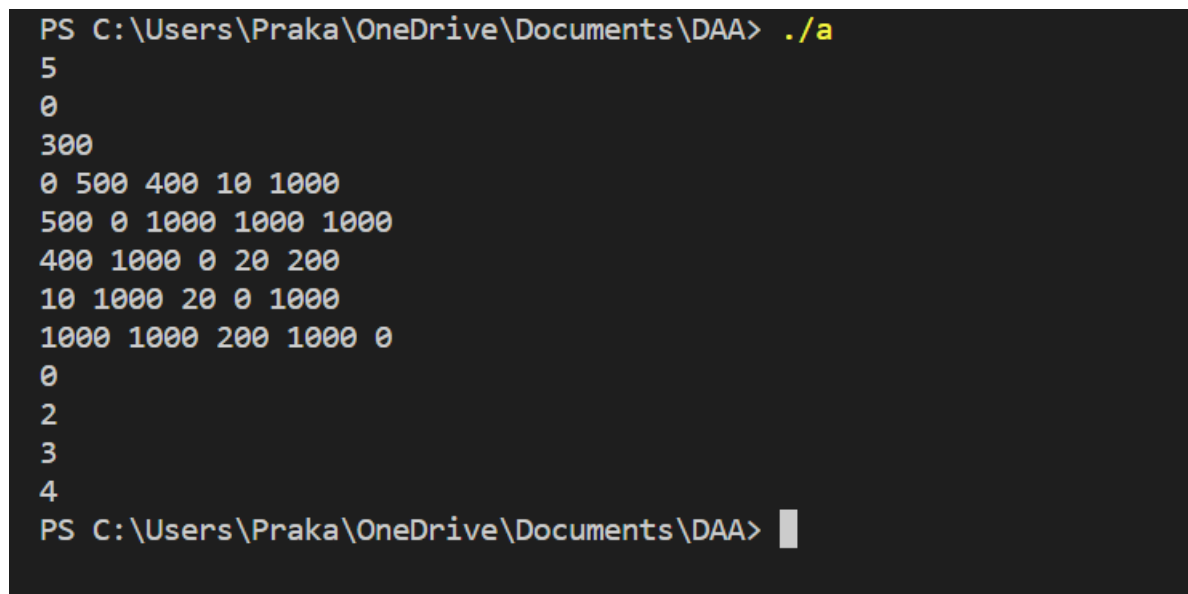

Q3. N number of cities in a state is represented using an adjacency matrix "path" where each element path[i][j](i and j are the index city i and city j) represents the distance from city i to city j. Print the indexes of cities which can be reached from city "src" in less than "dist" distance.

Sample Input 1:

```
5 //N : number of cities
0 // src
300 //dist
0 500 400 10 1000 //adjacency matrix representing cities
500 0 1000 1000 1000
400 1000 0 20 200
10 1000 20 0 1000
1000 1000 200 1000 0
```

Expected Output 1:

```
0
2
3
4
```



```
PS C:\Users\Praka\OneDrive\Documents\DAA> ./a
5
0
300
0 500 400 10 1000
500 0 1000 1000 1000
400 1000 0 20 200
10 1000 20 0 1000
1000 1000 200 1000 0
0
2
3
4
PS C:\Users\Praka\OneDrive\Documents\DAA> █
```

Boilerplate/Skeleton Code:

```
void main(){
    //Driver's Code
    int n;int src;int dist;
    scanf("%d",&n);
    scanf("%d",&src);
    scanf("%d",&dist);
```

```
int (*adj)[n]=malloc(sizeof(int)*n*n);
for(int i=0;i<n;i++){
    for(int j=0;j<n;j++){
        scanf("%d",&adj[i][j]);
    }
}
solution(src,n,adj,dist);
}
```

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

void solution(int src, int n, int adj[][n], int dist) {
    // Use Floyd Warshall algorithm to find the shortest distance between all pairs of cities
    for (int k = 0; k < n; k++) {
        for (int i = 0; i < n; i++) {
```

```

        for (int j = 0; j < n; j++) {
            if (adj[i][j] > adj[i][k] + adj[k][j]) {
                adj[i][j] = adj[i][k] + adj[k][j];
            }
        }
    }
}

// Print the indices of cities that can be reached from the source city within the given
distance
for (int i = 0; i < n; i++) {
    if (adj[src][i] <= dist) {
        printf("%d\n", i);
    }
}
}

void main(){
    //Driver's Code
    int n;int src;int dist;
    scanf("%d",&n);
    scanf("%d",&src);
    scanf("%d",&dist);
    int (*adj)[n]=malloc(sizeof(int)*n*n);
    for(int i=0;i<n;i++){
        for(int j=0;j<n;j++){
            scanf("%d",&adj[i][j]);
        }
    }
    solution(src,n,adj,dist);
}

```

Test cases:

```

1.
5
0
175
0 200 300 150 1000
200 0 1000 1000 1000
300 1000 0 20 200
150 1000 20 0 1000
1000 1000 200 1000 0

```

Output -

```

0
2
3

```

```

PS C:\Users\Praka\OneDrive\Documents\DAA> ./a
5
0
175
0 200 300 150 1000
200 0 1000 1000 1000
300 1000 0 20 200
150 1000 20 0 1000
1000 1000 200 1000 0
0
2
3
PS C:\Users\Praka\OneDrive\Documents\DAA>

```

2.

4

0

200

0 500 400 10

500 0 1000 1000

400 1000 0 20

10 1000 20 0

Output -

0

2

3

```

PS C:\Users\Praka\OneDrive\Documents\DAA> ./a
4
0
200
0 500 400 10
500 0 1000 1000
400 1000 0 20
10 1000 20 0
0
2
3
PS C:\Users\Praka\OneDrive\Documents\DAA>

```

3.
5
0
150
0 200 300 150 1000
200 0 1000 1000 1000
300 1000 0 20 200
150 1000 20 0 1000
1000 1000 200 1000 0

Output -

0
3

```
PS C:\Users\Praka\OneDrive\Documents\DAA> ./a
5
0
150
0 200 300 150 1000
200 0 1000 1000 1000
300 1000 0 20 200
150 1000 20 0 1000
1000 1000 200 1000 0
0
3
PS C:\Users\Praka\OneDrive\Documents\DAA>
```

