A) FORWARD GRAPH

DATE:

CODE:

```
#include <stdio.h>
#include <stdlib.h>
#include <limits.h>
#define IN 10000

// Function prototypes
void Fgraph(int n, int k, int **c);
int forward_graph(int **c, int d[], int cost[], int stages[], int n, int k, int path[]);
void print_forward_table(int d[], int cost[], int stages[], int n, int **c, int path[], int k);

void Fgraph(int n, int k, int **c)
{
    int cost[n], j, r, i, d[n], stage[n], mini[n], p[n], path[n];
    stage[n - 1] = k;
    for (i = 0; i < n; i++) {
        cost[i] = 0;
        mini[i] = IN;
    }
    for (j = n - 2; j >= 0; j--) {
        int min = INT_MAX;
        for (i = j + 1; i < n; i++) {
            if (c[j][i] != IN) {
                if (c[j][i] + cost[i] < mini[i]) {
                    mini[i] = c[j][i] + cost[i];
                    stage[j] = stage[i] - 1;
                }
            }
            if (c[j][i] != IN && c[j][i] + cost[i] < min) {
                r = i;
                min = c[j][i] + cost[i];
            }
        }
        if (stage[j] != stage[j + 1])
            printf("Stage %d:\n", stage[j]);
        printf("%-10s", " ");
        printf("cost[%d, %d] = min( ", stage[j], j + 1);
        for (i = j + 1; i < n; i++) {
            if (mini[i] != INT_MAX)
                printf("%d ", mini[i]);
        }
        printf(")\n");
        for (i = 0; i < n; i++)
            mini[i] = INT_MAX;
        cost[j] = c[j][r] + cost[r];
        d[j] = r;
        printf("%-10s", " ");
        printf("cost[%d, %d] = %d", stage[j], j + 1, cost[j]);
        printf("%-10s", " ");
        printf("d[%d,%d] = %d",stage[j], j + 1, d[j] + 1);
        printf("\n\n");
    }
    p[0] = 0;
    p[k - 1] = n - 1;
    for (j = 1; j <= k - 2; j++)
        p[j] = d[p[j - 1]];
    printf("Minimum distance is : %d\n", cost[0]);
    printf("Path : ");
    for (i = 0; i < k; i++) {
        printf("%d ", p[i] + 1);
        if (i != k - 1)
            printf(" -> ");
    }
    printf("\n");
}

int forward_graph(int **c, int d[], int cost[], int stages[], int n, int k, int path[])
{
    Fgraph(n, k, c);
    for (int i = 0; i < n; i++) {
        cost[i] = IN;
        stages[i] = -1; // Initialize stages to -1 (indicating not visited)
        path[i] = 0; // Initialize path to 0
    }
    cost[n - 1] = 0;
    stages[n - 1] = k; // The last node is in the last stage
    for (int i = n - 2; i >= 0; i--) {
        for (int j = i + 1; j < n; j++) {
            if ((c[i][j] + cost[j]) < cost[i]) {
                cost[i] = c[i][j] + cost[j];
                d[i] = j;
                stages[i] = stages[j] - 1; // Assign the stage of vertex i
            }
        }
    }
    int ptr = d[0];
    path[0] = 1; // Mark the first node as part of the path
    for (int i = 0; i < k-1; i++) {
        path[ptr] = 1; // Mark the nodes on the path
        ptr = d[ptr];
    }
    return cost[0];
}

void print_forward_table(int d[], int cost[], int stages[], int n, int **c, int path[], int k)
{
    printf("\nVertex\t\t\tCost\t\t\tMinimum Values Considered\n");
    for (int i = n - 2; i >= 0; i--) {
        if(path[i]==1)
        printf("|d[%d,%d]|=|%d|\tcost[%d,%d]=%d\t\t min(",stages[i], i + 1, d[i] +
1,stages[i],i+1, cost[i]);
        else
        printf("d[%d,%d]=%d\t\tcost[%d,%d]=%d\t\t\tmin(",stages[i], i + 1, d[i] +
1,stages[i],i+1, cost[i]);
        int first = 1; // Flag to handle printing comma
        for (int j = i + 1; j < n; j++) {
            if (c[i][j] + cost[j] == cost[i]) {
                if (!first) {
                    printf(", ");
                }
                printf("%d", c[i][j]);
                first = 0;
            }
        }
        printf(")\t"); // Added tab
        if (path[i] == 1) { // Check if d value is part of the path
            printf(" [Path]"); // Print [Path] if it is part of the path
        }
        printf("\n");
    }
}

int main()
{
    int n; // Number of nodes
    int k; // Number of stages
    printf("Enter the number of nodes: ");
    scanf("%d", &n);
    int **c = (int **)malloc(n * sizeof(int *));
    for (int i = 0; i < n; i++)
        c[i] = (int *)malloc(n * sizeof(int));
    // Initialize the graph with IN
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            c[i][j] = IN;
        }
    }
    printf("Enter the number of stages: ");
    scanf("%d", &k);
    // Input the edges and weights until -1 -1 -1 is entered
    printf("Enter edges and weights (source destination weight), enter -1 -1 -1 to stop:\n");
    int source, destination, weight;
    while (1) {
        scanf("%d %d %d", &source, &destination, &weight);
        if (source == -1 && destination == -1 && weight == -1)
            break;
        c[source - 1][destination - 1] = weight; // Adjusting for 0-based indexing
    }
    int *d = (int *)malloc(n * sizeof(int));
    int *cost = (int *)malloc(n * sizeof(int));
    int *stages = (int *)malloc(n * sizeof(int)); // Array to hold stages of each vertex
    int *path = (int *)malloc(n * sizeof(int)); // Array to track path
    printf("Cost of shortest path from 1 to %d: %d\n", n, forward_graph(c, d, cost, stages, n, k,
path));
    // Print the forward graph table
    printf("\nForward Graph Table:\n");
    print_forward_table(d, cost, stages, n, c, path, k);
    // Free dynamically allocated memory
    for (int i = 0; i < n; i++)
        free(c[i]);
    free(c);
    free(d);
    free(cost);
    free(stages);
    free(path);
    return 0;
}
```

OUTPUT:

```
Enter the number of nodes: 13
Enter the number of stages: 5
Enter edges and weights (source destination weight), enter -1 -1 -1 to stop:
1 2 13
1 3 12
1 4 18
1 5 17
2 6 16
2 8 15
3 7 11
3 9 12
4 6 11
4 8 13
5 7 11
5 9 12
6 10 14
6 11 15
6 12 11
7 10 10
7 11 8
7 12 12
8 11 11
8 12 10
9 11  8
9 12 10
10 13 9
11 13 8
12 13 7
-1-1-1
```

```
Stage 4:
        cost[4, 12] = min( 7 )
        cost[4, 12] = 7         d[4,12] = 13

        cost[4, 11] = min( 8 )
        cost[4, 11] = 8         d[4,11] = 13

        cost[4, 10] = min( 9 )
        cost[4, 10] = 9         d[4,10] = 13

Stage 3:
        cost[3, 9] = min( 16 17 )
        cost[3, 9] = 16         d[3,9] = 11

        cost[3, 8] = min( 19 17 )
        cost[3, 8] = 17         d[3,8] = 12

        cost[3, 7] = min( 19 16 19 )
        cost[3, 7] = 16         d[3,7] = 11

        cost[3, 6] = min( 23 23 18 )
        cost[3, 6] = 18         d[3,6] = 12

Stage 2:
        cost[2, 5] = min( 27 28 )
        cost[2, 5] = 27         d[2,5] = 7

        cost[2, 4] = min( 29 30 )
        cost[2, 4] = 29         d[2,4] = 6

        cost[2, 3] = min( 27 28 )
        cost[2, 3] = 27         d[2,3] = 7

        cost[2, 2] = min( 34 32 )
        cost[2, 2] = 32         d[2,2] = 8

Stage 1:
        cost[1, 1] = min( 45 39 47 44 )
        cost[1, 1] = 39         d[1,1] = 3

Minimum distance is : 39
Path : 1  -> 3  -> 7  -> 11  -> 13
Cost of shortest path from 1 to 13: 39
```

Forward Graph Table:

Vertex	Cost	Minimum Values Considered
d[4,12]=13	cost[4,12]=7	min(7)
|d[4,11]|=|13|	cost[4,11]=8	min(8) [Path]
d[4,10]=13	cost[4,10]=9	min(9)
d[3,9]=11	cost[3,9]=16	min(8)
d[3,8]=12	cost[3,8]=17	min(10)
|d[3,7]|=|11|	cost[3,7]=16	min(8) [Path]
d[3,6]=12	cost[3,6]=18	min(11)
d[2,5]=7	cost[2,5]=27	min(11)
d[2,4]=6	cost[2,4]=29	min(11)
|d[2,3]|=|7|	cost[2,3]=27	min(11) [Path]
d[2,2]=8	cost[2,2]=32	min(15)
|d[1,1]|=|3|	cost[1,1]=39	min(12) [Path]