DAA Prac 08

Name: Vijiyant Tanaji Shejwalkar

Reg no: 2020BIT057

1. Floyd Warshall Algorithm

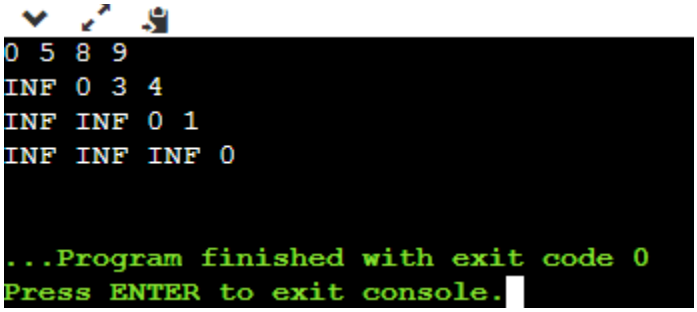
```
#include <bits/stdc++.h>
using namespace std;

// Number of vertices in the graph
#define V 4

/* Define Infinite as a large enough
value.This value will be used for
vertices not connected to each other */
#define INF 99999

// A function to print the solution matrix
void printSolution(int dist[][V]);

// Solves the all-pairs shortest path
// problem using Floyd Warshall algorithm
void floydWarshall(int dist[][V])
```

```
{

    int i, j, k;

    /* Add all vertices one by one to
    the set of intermediate vertices.
    ---> Before start of an iteration,
    we have shortest distances between all
    pairs of vertices such that the
    shortest distances consider only the
    vertices in set {0, 1, 2, .. k-1} as
    intermediate vertices.
    ----> After the end of an iteration,
    vertex no. k is added to the set of
    intermediate vertices and the set becomes {0, 1, 2, ..
    k} */
    for (k = 0; k < V; k++) {
        // Pick all vertices as source one by one
        for (i = 0; i < V; i++) {
            // Pick all vertices as destination for the
            // above picked source
            for (j = 0; j < V; j++) {
                // If vertex k is on the shortest path from
                // i to j, then update the value of
```

```
                                        // dist[i][j]
                                        if (dist[i][j] > (dist[i][k] + dist[k][j])
                                                && (dist[k][j] != INF
                                                        && dist[i][k] != INF))
                                                dist[i][j] = dist[i][k] + dist[k][j];
                                }
                        }
                }

        // Print the shortest distance matrix
        printSolution(dist);
}

/* A utility function to print solution */
void printSolution(int dist[][V])
{
        cout << "The following matrix shows the shortest "
                        "distances"
                        " between every pair of vertices \n";
        for (int i = 0; i < V; i++) {
                for (int j = 0; j < V; j++) {
                        if (dist[i][j] == INF)
                                cout << "INF"
                                        << " ";
```

```
                else
                        cout << dist[i][j] << " ";
            }
            cout << endl;
        }
}

// Driver's code
int main()
{
        /* Let us create the following weighted graph
                10
        (0)------->(3)
            |         /|\
        5 |        |
            |         | 1
        \|/       |
        (1)------->(2)
                3          */
        int graph[V][V] = { { 0, 5, INF, 10 },
                                                { INF, 0, 3, INF },
                                                { INF, INF, 0, 1 },
                                                { INF, INF, INF, 0 } };
```

```
    // Function call
    floydWarshall(graph);
    return 0;
}
```



2. Knapsack Algorithm

```
#include <stdio.h>

// A utility function that returns
// maximum of two integers
int max(int a, int b) { return (a > b) ? a : b; }

// Returns the maximum value that
// can be put in a knapsack of capacity W
int knapSack(int W, int wt[], int val[], int n)
{
   int i, w;
```

```
    int K[n + 1][W + 1];


    // Build table K[][] in bottom up manner
    for (i = 0; i <= n; i++) {
        for (w = 0; w <= W; w++) {
            if (i == 0 || w == 0)
                K[i][w] = 0;
            else if (wt[i - 1] <= w)
                K[i][w] = max(val[i - 1]
                              + K[i - 1][w - wt[i - 1]],
                          K[i - 1][w]);
            else
                K[i][w] = K[i - 1][w];
        }
    }


    return K[n][W];
}


// Driver Code
int main()
{
    int profit[] = { 60, 100, 120 };
    int weight[] = { 10, 20, 30 };
```

```
    int W = 50;
    int n = sizeof(profit) / sizeof(profit[0]);
    printf("%d", knapSack(W, weight, profit, n));
    return 0;
}
```