There are N stations on route of a train. The train goes from station 0 to N-1. The ticket cost for all pair of stations (i, j) is given where j is greater than i. Find the minimum cost to reach the destination.

Consider the following example:

```
Input:
cost[N][N] = { {0, 15, 80, 90},
               {INF, 0, 40, 50},
               {INF, INF, 0, 70},
               {INF, INF, INF, 0}
             };
There are 4 stations and cost[i][j] indicates cost to reach j
from i. The entries where j < i are meaningless.

Output:
The minimum cost is 65
The minimum cost can be obtained by first going to station 1
from 0. Then from station 1 to station 3.
```

We strongly recommend to minimize your browser and try this yourself first.

The minimum cost to reach N-1 from 0 can be recursively written as following:

```
minCost(0, N-1) = MIN { cost[0][n-1],
                        cost[0][1] + minCost(1, N-1),
                        minCost(0, 2) + minCost(2, N-1),
                        ........,
                        minCost(0, N-2) + cost[N-2][n-1] }
```

The following is C++ implementation of above recursive formula.

```
// A naive recursive solution to find min cost path from station 0
// to station N-1
#include<iostream>
#include<climits>
using namespace std;

// infinite value
#define INF INT_MAX

// Number of stations
#define N 4

// A recursive function to find the shortest path from
// source 's' to destination 'd'.
int minCostRec(int cost[][N], int s, int d)
{
    // If source is same as destination
    // or destination is next to source
    if (s == d || s+1 == d)
      return cost[s][d];

    // Initialize min cost as direct ticket from
    // source 's' to destination 'd'.
    int min = cost[s][d];

    // Try every intermediate vertex to find minimum
    for (int i = s+1; i<d; i++)
    {
        int c = minCostRec(cost, s, i) +
```

```
                minCostRec(cost, i, d);
            if (c < min)
               min = c;
        }
        return min;
}

// This function returns the smallest possible cost to
// reach station N-1 from station 0. This function mainly
// uses minCostRec().
int minCost(int cost[][N])
{
        return minCostRec(cost, 0, N-1);
}

// Driver program to test above function
int main()
{
        int cost[N][N] = { {0, 15, 80, 90},
                           {INF, 0, 40, 50},
                           {INF, INF, 0, 70},
                           {INF, INF, INF, 0}
                         };
        cout << "The Minimum cost to reach station "
             << N << " is " << minCost(cost);
        return 0;
}
```

Output:

```
The Minimum cost to reach station 4 is 65
```

Time complexity of the above implementation is exponential as it tries every possible path from 0 to N-1. The above solution solves same subrpoblems multiple times (it can be seen by drawing recursion tree for minCostPathRec(0, 5).
Since this problem has both properties of dynamic programming problems ((see this and this). Like other typical Dynamic Programming(DP) problems, re-computations of same subproblems can be avoided by storing the solutions to subproblems and solving problems in bottom up manner.

One dynamic programming solution is to create a 2D table and fill the table using above given recursive formula. The extra space required in this solution would be $O(N^2)$ and time complexity would be $O(N^3)$

We can solve this problem using O(N) extra space and $O(N^2)$ time. The idea is based on the fact that given input matrix is a Directed Acyclic Graph (DAG). The shortest path in DAG can be calculated using the approach discussed in below post.

Shortest Path in Directed Acyclic Graph

We need to do less work here compared to above mentioned post as we know topological sortingof the graph. The topological sorting of vertices here is 0, 1, ..., N-1. Following is the idea once topological sorting is known.

The idea in below code is to first calculate min cost for station 1, then for station 2, and so on. These costs are stored in an array dist[0...N-1].

1) The min cost for station 0 is 0, i.e., dist[0] = 0

2) The min cost for station 1 is cost[0][1], i.e., dist[1] = cost[0][1]

3) The min cost for station 2 is minimum of following two.
   a) dist[0] + cost[0][2]
   b) dist[1] + cost[1][2]

3) The min cost for station 3 is minimum of following three.

a) dist[0] + cost[0][3]

b) dist[1] + cost[1][3]

c) dist[2] + cost[2][3]

Similarly, dist[4], dist[5], ... dist[N-1] are calculated.

Below is C++ implementation of above idea.

```
// A Dynamic Programming based solution to find min cost
// to reach station N-1 from station 0.
#include<iostream>
#include<climits>
using namespace std;

#define INF INT_MAX
#define N 4

// This function returns the smallest possible cost to
// reach station N-1 from station 0.
int minCost(int cost[][N])
{
    // dist[i] stores minimum cost to reach station i
    // from station 0.
    int dist[N];
    for (int i=0; i<N; i++)
       dist[i] = INF;
    dist[0] = 0;

    // Go through every station and check if using it
    // as an intermediate station gives better path
    for (int i=0; i<N; i++)
       for (int j=i+1; j<N; j++)
          if (dist[j] > dist[i] + cost[i][j])
             dist[j] = dist[i] + cost[i][j];

    return dist[N-1];
}

// Driver program to test above function
int main()
{
    int cost[N][N] = { {0, 15, 80, 90},
                       {INF, 0, 40, 50},
                       {INF, INF, 0, 70},
                       {INF, INF, INF, 0}
                     };
    cout << "The Minimum cost to reach station "
          << N << " is " << minCost(cost);
    return 0;
}
```

Output:

```
The Minimum cost to reach station 4 is 65
```