```
Exercise 1

Find the minimum cost to reach a destination traveling by train

There are N stations in a train route. The train goes from station 0 up to N
The cost of a ticket is given when j is greater than i. We need to find the
cost to reach the destination from the initial station.

INPUT
    ST0 ST1 ST2 ST3
ST0 0   15  80  90
ST1 INF 0   40  50
ST2 INF INF 0   70
ST3 INF INF INF 0

Cost[i,j] is the cost to reach j from i
Cost[I,j] = 0 if i = j
Cost[i,j] = INF if j > i


A recursive definition of the problem:

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] }


Recursive Algorithm

Recursive-Min-Cost(cost, s, d)
{
    if s == d or s+1 == d
        return cost[s][d]

    min = cost[s][d]  // Initialize with cost of direct ticket

    for (i = s + 1, i < d, i++)
    {
        c = Recursive-Min-Cost(cost, s, i ) + Recursive-Min-Cost(cost, i, d
        if (c < min)
            min = c;
    }
    return min;
}
```

```
1.  Optimal substructure?
We want the optimal path, which requires optimal sub-paths
    As in the shortest path problem in graphs that we will see later

2.  Overlapped sub-problems?

Cost 0-1 + cost 1-d
        Cost 1-2 + Cost 2-d
                Cost 2-3 + Cost 3-d
                        Cost 3-4 + Cost 4-d
                                …
Cost 0-2 + cost 2-d
        Cost 2-3 + Cost 3-d
                Cost 3-4 + Cost 4-d
                        Cost 4-5 + Cost 5-d
                                …
Cost 0-3 + cost 3-d
            Cost 3-4 + Cost 4-d
                    Cost 4-5 + Cost 5-d
                            Cost 5-6 + Cost 6-d
                                    …
    …


We can use dynamic programming

How can we use dynamic programming
- Considering the routes as a graph
- We perform a topological sort on the vertices of the graph: 0, 1, 2, …, N-
    - Compute the minimum cost to each station and store them in an array di
    - Min cost for station 0 is 0, then distance[0] = cost[0][0] = 0
    - Min cost for station 1 is cost[0][1], then distance[1] = cost[0][1]
        - dist[0] + cost[0][1]
    - Min cost for station 2 is the minimum of:
        - dist[0] + cost[0][2]
        - dist[1] + cost[1][2]
    - Min cost for station 3 is the minimum of:
        - dist[0] + cost[0][3]
        - dist[1] + cost[1][3]
        - dist[2] + cost[2][3]
    - and so on…

Dynamic Programming Algorithm:
```

```
DP-Min-Cost(cost)
{
    Create array dist[ N] to store the minimum cost to reach station i
    Initialize dist[i] to a very large value, INF
    dist[0] = 0

    // For each station, verify if using it as intermediate station gives us
    for (int i= 0, i < N, i++
        for (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]
}
```

Exercise 2

Find the minimum number of coins that make a given value

Given a value V, we want to make chance for V cents. We assume that we have infinite supply of each of the C = {C1, C2, …, Cm} valued coins. We are aske make the change with the minimum number of coins.

Examples

Input: coins: 25, 10, 5, V = 30
Output: Minimum 2 coins required: 1 x 25, 1 x 5

Input: coins: 9, 6, 5, 1, V = 11
Output: Minimum 2 coins required: 1 x 6, 1 x 5

A recursive definition of the problem:

```
    If V == 0, then 0 coins

    If V > 0: minCoin(coins[0…m-1], V) =
                          min { 1 + minCoins(V - coin[i] ) }
```

Recursive Algorithm:

```
Recursive-Min-Coins(coins, m, V)
{
    If (V == 0) return 0
```

```

    res = INT_MAX  // Initialization of the result variable
    for (i, i<m, i++)
    {
        if (coins[i] <= V)
        {
            int sub_res = Recursive-Min-Coins(coins, m, V - coins[i])

            if (sub_res != INT_MAX && sub_res + 1 < res)
                res = sub_res + 1
        }
    }
    return res
}


1.  Optimal substructure?
We want the minimum number of coins to make change for V (then optimal).
An optimal decision on V requires optimal decisions for values for each of
the different coins values Vi


2.  Overlapped sub-problems?
For coins = 5, 10, 25 and V = 90
    minCoins(coins, 90) =min{ 1 + minCoins(90 – 5)
                    1 + minCoins(90 – 10)
                    1 + minCoins(90 – 25)}

    minCoins(coins,85) = min{ 1 + minCoins(85 – 5)
                    1 + minCoins(85 – 10)
                    1 + minCoins(85 – 25)

    minCoins(coins,80) = min{ 1 + minCoins(80 – 5)
                    1 + minCoins(80 – 10)
                    1 + minCoins(80 – 25) }


We can use dynamic programming

-   We avoid solving sub- problems over and over again
-   We can use a bottom up approach if we store the solutions to sub-problem

Dynamic Programming Algorithm:

DP-Min-Coins(coins, m, V)
{
```

```
    Create table[V+1]  //minimum number of coins for value i
    table[0] = 0
    for (i=1, i<=V, i++)
        table[i]=INT_MAX

    // Get the minimum coins required for values from 1 to V
    for (i=1, i<=V, i++)
    {
        // For coins smaller than i
        for (j=0, j<m, j++)
        {
            if (coins[j] < i)
            {
                sub_res = table[i-coins[j]]
                if (sub_res != INT_MAX && sub_res + 1 < table[i])
                    table[i] = sub_res + 1
            }
        }
    }
    return table[V]
}
```