گزارش پروژه **Dynamic Programming**

گروه **PI Radian**

اعضا: **علیرضا پیرهادی، محمد میغانی، علی نظری**

الف) توضیح تئوری الگوریتم:

۱. شکستن مسئه تبدیل آن به یک مسئله بازگشتی

$$\mathbf{M[i,\ j] = Cost\ [i,\ j] + \min_{1 \le k \le numberOfCities} (M[i-1, k] + RelocationCost[k, j])}$$

تابع بازگشتی به صورت فوق است که در آن:

M[i, j]:
مینیموم هزینه از ماه اول تا ماه i ام در صورتی که ماه i ام در شهر j ام باشیم.

Cost [i, j]:
مقدار هزینه مربوط به ماه i ام در شهر j ام.

RelocationCost:
هزینه جابجایی از شهر k ام به شهر j ام. اگر k با j برابر باشد به این معنی است که در شهر فعلی می‌مانیم که هزینه انتقال آن صفر است.

تابع بازگشتی فوق به این صورت عمل می کند که ابتدا مسئله را برای ماه اول به ازای شهر های مختلف حل می کند سپس برای دو ماه اول و همین روند را برای j ماه اول انجام می دهد تا به جواب برسیم.

pseudo code .۲

```
//get input
Define dp[n][k]
Define path[n][k]

for  i = 0 to n(number of month)
    for  j = 0 to k(number of cities)
        min = Integer.MAX_VALUE
         if i == 0
             min = 0
         else
             for l = 0 to k
                 if dp[i-1][l] + f[l][j] < min
                     min = dp[i - 1][l] + f[l][j]
                     path[i][j] = l
         dp[i][j] = c[j][i] + min

min = dp[n-1][0]
finalCity = 0

for  i = 1 to k
    if dp[n-1][i] < min
        min = dp[n - 1][i]
        finalCity = i

print min(minimum_cost)

city = finalCity
Define bestPath[n]
bestPath[n-1] = finalCity

for  i = 0 to n -1
     bestPath[n-2-i] = path[n-1-i][city]
     city = path[n-1-i][city]

//print Order of cities in this way:
for  i = 0 to n -1
    print (bestPath[i] + 1) + " -> "
print bestPath[n - 1] + 1
```

۳. توضیح در مورد الگوریتمی استفاده شده

با توجه تابع بازگشتی قسمت اول به این صورت عمل می کنیم که ابتدا مینیموم هزینه در یک ماه به ازای شهر های مختلف را به دست می آوریم سپس برای دو ماه اول با توجه به رابطه بازگشتی و جواب مرحله قبل ، برای هر شهر دو حالت رخ خواهد داد یا هزینه ماندن در شهر از انتقال به شهر دیگر کم تر است که در این صورت در همان شهر می مانیم یا هزینه انتقال به شهر دیگر از هزینه ماندن در شهر فعلی کم تر است که در این صورت به شهر مورد نظر منتقل می شویم واضح است برای این کار باید هزینه ماندن در شهر فعلی را با هزینه جابجایی تمام شهرها مقایسه شود.
سپس برای سه ماه اول مسئله را حل می کنیم و همین روند را برای j ماه اول انجام می دهیم تا به جواب مسئله برسیم.

ب) پیچیدگی الگوریتم

پیچیدگی این الگوریتم به صورت $O(n \times k^2)$ است. همانطور که در قطعه کد زیر می بینیم، ما ۳ حلقه for تو در تو داریم که ثابت اولی n و ثابت دومی و سومی هم k است و در نهایت پیچیدگی ای که در بالا به آن اشاره شد را می سازند؛ به عبارت دیگر قسمت اصلی الگوریتم یعنی بلوک زیر است که $n \times k^2$ بار اجرا می شود.

```
if (dp[i - 1][l] + f[l][j] < min)
{
    min = dp[i - 1][l] + f[l][j];
    path[i][j] = l;
}
```

```
for (int i = 0; i < n; i++)
{
    for (int j = 0; j < k; j++)
    {
        int min = Integer.MAX_VALUE;
        if (i == 0)
        {
            min = 0;
        }
        else
        {
            for (int l = 0; l < k; l++)
            {
                if (dp[i - 1][l] + f[l][j] < min)
                {
                    min = dp[i - 1][l] + f[l][j];
                    path[i][j] = l;
                }
            }
        }
        dp[i][j] = c[j][i] + min;
    }
}
```

پ) پیاده سازی الگوریتم

الگوریتمی که در قسمت الف آن را توضیح دادیم در قالب قطعه کد زیر پیاده سازی شده که فایل آن به پیوست ارسال گردیده است و تصویری از آن را نیز در اینجا مشاهده می کنیم:

```
import java.util.Scanner;

public class Main
{
    public static void main(String[] args)
    {
        Scanner input = new Scanner(System.in);
        int k, n;
        System.out.print("Enter number of cities: ");
        k = input.nextInt();
        System.out.print("Enter number of months: ");
        n = input.nextInt();
        int c[][] = new int[k][n];
        for (int i = 0; i < k; i++)
        {
            System.out.print("Enter costs of operation in city " + (i + 1) + ":
");
            for (int j = 0; j < n; j++)
            {
                c[i][j] = input.nextInt();
            }
        }
        int f[][] = new int[k][k];
        System.out.println("Enter relocation costs: ");
        for (int i = 0; i < k; i++)
        {
            for (int j = 0; j < k; j++)
            {
                if (i != j)
                {
                    System.out.print("f[" + (i + 1) + "," + (j + 1) + "] = ");
                    f[i][j] = input.nextInt();
                }
                else
                {
                    f[i][j] = 0;
                }
            }
        }
        int dp[][] = new int[n][k];
        int path[][] = new int[n][k];
        for (int i = 0; i < n; i++)
        {
            for (int j = 0; j < k; j++)
            {
                int min = Integer.MAX_VALUE;
```

```
                if (i == 0)
                {
                    min = 0;
                }
                else
                {
                    for (int l = 0; l < k; l++)
                    {
                        if (dp[i - 1][l] + f[l][j] < min)
                        {
                            min = dp[i - 1][l] + f[l][j];
                            path[i][j] = l;
                        }
                    }
                }
                dp[i][j] = c[j][i] + min;
            }
        }
        int min = dp[n - 1][0];
        int finalCity = 0;
        for (int i = 1; i < k; i++)
        {
            if (dp[n - 1][i] < min)
            {
                min = dp[n - 1][i];
                finalCity = i;
            }
        }
        System.out.println("Minimum cost = " + min);
        int city = finalCity;
        int bestPath[] = new int[n];
        bestPath[n - 1] = finalCity;
        for (int i = 0; i < n - 1; i++)
        {
            bestPath[n - 2 - i] = path[n - 1 - i][city];
            city = path[n - 1 - i][city];
        }
        System.out.print("Order of cities : ");
        for (int i = 0; i < n - 1; i++)
        {
            System.out.print((bestPath[i] + 1) + " -> ");
        }
        System.out.println(bestPath[n - 1] + 1);
    }
}
```

ت) خروجی الگوریتم پیاده سازی شده به اضافه ورودی داده شده

ورودی داده شده در صورت پروژه به صورت زیر بوده است:

Operating Costs:

	Months										
City	1	2	3	4	5	6	7	8	9	10	11	12
1 (NY)	8	3	10	43	15	48	5	40	20	30	28	24
2 (LA)	18	1	35	18	10	19	18	10	8	5	8	20
3 (DEN)	40	5	8	13	21	12	4	27	25	10	5	15

Relocation costs:

City	1 (NY)	2 (LA)	3 (DEN)
1 (NY)	0	20	15
2 (LA)	20	0	10
3 (DEN)	15	10	0

خروجی الگوریتم پیاده سازی شده توسط ما نیز به شکل زیر است که آن را مشاهده میکنیم: