Little Shop of Flowers [1]

February 2, 2002

1 Problem Statement

1.1 The Problem

You want to arrange the window of your flower shop in a most pleasant way. You have F bunches of flowers, each being of a different kind, and at least as many vases ordered in a row. The vases are glued onto the shelf and are numbered consecutively 1 through V, where V is the number of vases, from left to right so that the vase 1 is the leftmost, and the vase V is the rightmost vase. The bunches are moveable and are uniquely identified by integers between 1 and F. These id-numbers have a significance: They determine the required order of appearance of the flower bunches in the row of vases so that the bunch i must be in a vase to the left of the vase containing bunch j whenever $i < j$. Suppose, for example, you have bunch of azaleas (id-number=1), a bunch of begonias (id-number=2) and a bunch of carnations (id-number=3). Now, all the bunches must be put into the vases keeping their id-numbers in order. The bunch of azaleas must be in a vase to the left of begonias, and the bunch of begonias must be in a vase to the left of carnations. If there are more vases than bunches of flowers then the excess will be left empty. A vase can hold only one bunch of flowers.

Each vase has a distinct characteristic (just like flowers do). Hence, putting a bunch of flowers in a vase results in a certain aesthetic value, expressed by an integer. The aesthetic values are presented in a table as shown below. Leaving a vase empty has an aesthetic value of 0.

	Vases				
Flowers	1	2	3	4	5
1	7	23	-5	-24	16
2	5	21	-4	10	23
3	-21	5	-4	-20	20

According to the table, azaleas, for example, would look great in vase 2, but they would look awful in vase 4.

To achieve the most pleasant effect you have to maximize the sum of aesthetic values for the arrangement while keeping the required ordering of the flowers.

If more than one arrangement has the maximal sum value, any one of them will be acceptable. You have to produce exactly one arrangement.

1.2 Assumption

- $1 \leq F \leq 100$ where F is the number of the bunches of flowers. The bunches are numbered 1 through F.
- $F \leq V \leq 100$ where V is the number of vases.
- $-50 \leq A_{ij} \leq 50$ where A_{ij} is the aesthetic value obtained by putting the flower bunch i into the vase j.

1.3 Input

- The first line contains two numbers: F, V.
- The following F lines: Each of the these lines contains V integers, so that A_{ij} is given as the j^{th} number on the $(i+1)^{st}$ line of the input.

1.4 Output

The output consists of two lines:

- The first line will contain the sum of aesthetic values for your arrangement
- The second line must present the arrangement as a list of F numbers, so that the k'th number on this line identifies the vase in which the bunch k is put.

1.5 Sample Input

```
3 5
7 23 -5 -24 16
5 21 -4 10 23
-21 5 -4 -20 20
```

1.6 Sample Output

```
53
2 4 5
```

2 Solution

2.1 Dynamic Programming Properties

The following discussion is based on the sample input.

Let's first try to place the first flower. If we place it in Vase 1, it will result in a smaller problem of placing 2 flowers (Flower 2, 3) in 4 vases (Vase 2, 3, 4, 5). If we place it in Vase 2, it will result in a smaller problem of placing 2 flowers (Flower 2, 3) in 3 vases (Vase 3, 4, 5). And so on. From here, we can discover the optimal substructure of the problem.

If we place Flower 1 in Vase 1, and Flower 2 in Vase 3, it will result in a smaller problem of placing 1 flower (Flower 3) in 2 vases (Vase 4, 5). If we place Flower 1 in Vase 2, and Flower 2 in Vase 3, it will still result in a smaller problem of placing 1 flower (Flower 3) in 2 vases (Vase 4, 5). This provides a clue that we need to store the result of placing 1 flower (Flower 3) in 2 vases (Vase 4, 5). Otherwise we will do some redundant computations.

2.2 Solution Algorithm

With the two properties above, we can proceed to write a DP algorithm to solve the problem. While the whole C++ program can be found in the Appendix, the following is the core procedure of the DP algorithm.

```
int ref=INT_MIN;
int arr[100][100], result[100][100], pos[100][100];
int F, V;
int compute(int x, int y) {
    if(x>=F || y>=V) return 0;
    if(result[x][y]>ref) return result[x][y];
    int m=ref;
    int last=V-F+x;
    for(int i=y;i<=last;i++) {
        int temp=arr[x][i]+compute(x+1,i+1);
        if(temp>=m) {
            m=temp;
            pos[x][y]=i;  }   }
    result[x][y]=m;
    return m;
}//method compute
```

Let's examine the code in detail. In the code, array *arr* is the input table which stores the profit values for each flower placed in different vases. For the sample, *arr* would be a matrix like:

7	23	-5	24	16
5	21	-4	10	23
-21	5	-4	-20	20

Array *result* is to store the intermediate results for future usage. The value of the cell $result[x][y]$ is the maximum profit value that can be achieved by placing Flower x to F in Vase y to V.

Array *pos* is to store the detailed placements that achieve the maximum profit. It will be used to construct the optimal placements.

2.3 Illustration of the Algorithm on the Sample Data

Let's take a close look at the example that shows how the values of array *result* and array *pos* are changed. In the following tables *NA* means Not Available.

53	NA	NA	NA	NA
NA	41	30	30	NA
NA	NA	20	20	20

The Final Values of Array *result*

2	NA	NA	NA	NA
NA	2	4	4	NA
NA	NA	5	5	5

The Final Values of Array *pos*

Let's trace how these two tables are computed. Initially,

NA	NA	NA	NA	NA
NA	NA	NA	NA	NA
NA	NA	NA	NA	NA

The Values of Array *result*

NA	NA	NA	NA	NA
NA	NA	NA	NA	NA
NA	NA	NA	NA	NA

The Values of Array *pos*

According to the sample, the recursive calling relation of method *compute(int, int)* is:

1. *compute(1,1)* calls *compute(2,2)*, which calls *compute(3,3)*, *compute(3,4)* and *compute(3,5)*.
2. *compute(1,1)* calls *compute(2,3)*, which calls *compute(3,4)* and *compute(3,5)*.
3. *compute(1,1)* calls *compute(2,4)*, which calls *compute(3,5)*.

The values of arrays *result* and *pos* will be changed as follows:

1. After the first call to *compute(3,3)*,

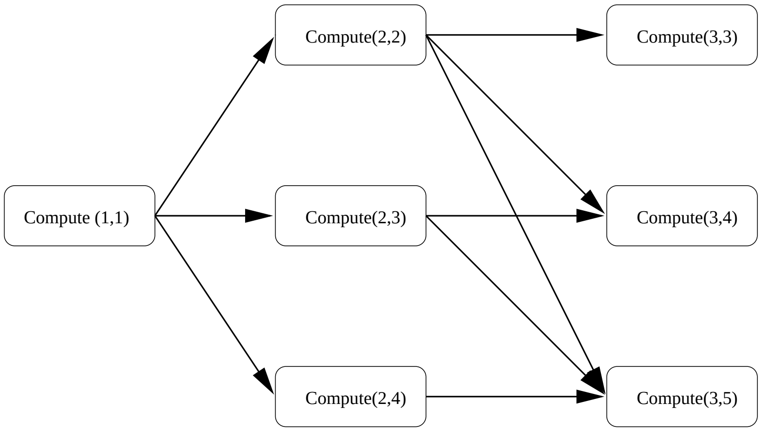

Figure 1: Recursive Function Call Diagram

NA	NA	NA	NA	NA
NA	NA	NA	NA	NA
NA	NA	20	NA	NA

The Values of Array *result*

NA	NA	NA	NA	NA
NA	NA	NA	NA	NA
NA	NA	5	NA	NA

The Values of Array *pos*

2. After the first call to *compute(3,4)*,

NA	NA	NA	NA	NA
NA	NA	NA	NA	NA
NA	NA	20	20	NA

The Values of Array *result*

NA	NA	NA	NA	NA
NA	NA	NA	NA	NA
NA	NA	5	5	NA

The Values of Array *pos*

3. After the first call to *compute(3,5)*,

NA	NA	NA	NA	NA
NA	NA	NA	NA	NA
NA	NA	20	20	20

The Values of Array *result*

NA	NA	NA	NA	NA
NA	NA	NA	NA	NA
NA	NA	5	5	5

The Values of Array *pos*

4. After the first call to *compute(2,2)*,

NA	NA	NA	NA	NA
NA	41	NA	NA	NA
NA	NA	20	20	20

The Values of Array *result*

NA	NA	NA	NA	NA
NA	2	NA	NA	NA
NA	NA	5	5	5

The Values of Array *pos*

5. After the first call to *compute(2,3)*,

NA	NA	NA	NA	NA
NA	41	30	NA	NA
NA	NA	20	20	20

The Values of Array *result*

NA	NA	NA	NA	NA
NA	2	4	NA	NA
NA	NA	5	5	5

The Values of Array *pos*

6. After the first call to *compute(2,4)*,

NA	NA	NA	NA	NA
NA	41	30	30	NA
NA	NA	20	20	20

The Values of Array *result*

NA	NA	NA	NA	NA
NA	2	4	4	NA
NA	NA	5	5	5

The Values of Array *pos*

7. After the first call to *compute(1,1)*,

53	NA	NA	NA	NA
NA	41	30	30	NA
NA	NA	20	20	20

The Values of Array *result*

2	NA	NA	NA	NA
NA	2	4	4	NA
NA	NA	5	5	5

The Values of Array *pos*

From the final values of Array *result* and Array *pos*, we will be able to get the maximum profit and generate the placement that achieves the maximum profit. From the cell result[1][1] = 53, we know that the maximum profit is 53. Then $pos[1][1] = 2$ indicates that Flower 1 is placed in Vase 2, $pos[2][2+1] = 4$ tells that Flower 2 is placed in Vase 4, and pos[3][4+1] = 5 shows that Flower 3 is placed in Vase 5. This completes the re-construction of the optimal process. In general, to re-construct an optimal solution, the program needs to record down information about the path. After the DP process ends, we should be will be able to construct an optimal solution based on those information.

References

[1] IOI'99. See *http://www.ioi99.org.tr/tasks/index.html*

Appendix: C++ Code for the Flower Problem

The program requires the input to be formatted as: F, V followed by the matrix decrypting the profit values for each flower placed in different vases. For the sample, the input would be:

```
3    5
7  23  -5  -24  16
5  21  -4  10  23
-21  5  -4  -20 20
```

```
#include <iostream>
using namespace std;

int ref=-1000000;
int arr[100][100];
int result[100][100];
int pos[100][100];
int F, V;
int compute(int x, int y) {
    if(x>=F || y>=V) return 0;
    if(result[x][y]>ref) return result[x][y];
    int m=ref;
    int temp;
    int last=V-F+x;
    for(int i=y;i<=last;i++) {
        temp=arr[x][i]+compute(x+1,i+1);
        if(temp>=m) {
            m=temp;
            pos[x][y]=i;  }   }
    result[x][y]=m;
    return m;
}

int main() {
    cin>>F>>V;
    for(int i=0;i<F;i++)
        for(int j=0;j<V;j++)
            cin>>arr[i][j];
    for(int i=0;i<F;i++)
        for(int j=0;j<V;j++)
```

```
            result[i][j]=ref;
    int max=compute(0,0);
    cout<<max<<endl;
    int temp=pos[0][0];
    cout<<temp+1;
    for(int i=1;i<F;i++) {
        temp=pos[i][temp+1];
        cout<<' '<<temp+1; }
    cout<<endl;
}//method main
```