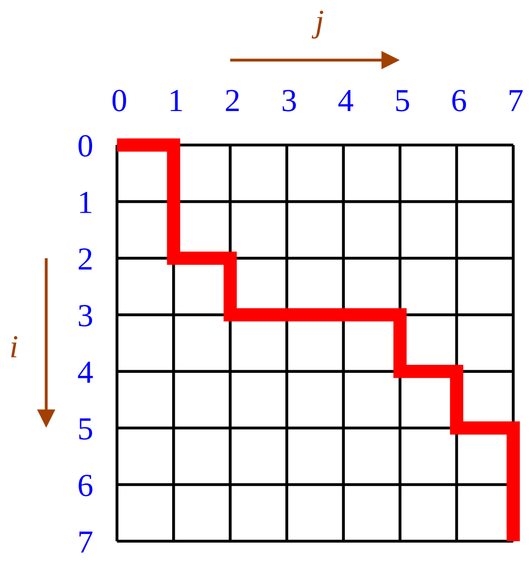
CS29003 Algorithms Laboratory

LAB TEST: EVEN PC

Last date of submission: 27–March–2018

Shunyajit is playing a video game. He enters an $n \times n$ mesh of tunnels at $(0,0)$, and the exit is at the point $(n-1, n-1)$. At every point of time, Shunyajit can move one unit right or one unit down through the mesh. That is, if he is at location (i, j) of the mesh, then his next position can be either $(i, j+1)$ or $(i+1, j)$ (unless he is at one of the boundaries). The following figure demonstrates an allowed sequence of moves of Shunyajit in an 8×8 mesh of tunnels (see the thick path).



Shunyajit starts at $(0,0)$ with energy level 1 (a positive floating-point number). While crossing a vertical segment of the mesh, Shunyajit has to fight with demons, and his energy level drops. The factor by which the energy level changes is supplied in an $(n-1) \times n$ matrix EF (exhaustion factor). That is, if Shunyajit moves from (i, j) to $(i+1, j)$, his energy level is multiplied by $EF[i][j]$.

The horizontal segments of the mesh do not contain any demons. Some (not all) of the horizontal segments contain magic potions. If Shunyajit drinks the potion at a horizontal segment, his energy level increases by 25% (that is, is multiplied by 1.25). Notice that the energy level can grow to values larger than 1. The presence/absence of the magic potion in the horizontal segments is presented by an $n \times (n-1)$ matrix PP (potion position) with entries from $\{0,1\}$. If $PP[i][j] = 0$, then the horizontal segment (i, j) to $(i, j+1)$ does not contain the potion. If $PP[i][j] = 1$, this segment contains the potion.

The goal of Shunyajit is to exit the mesh at $(n-1, n-1)$ with energy level as high as possible.

Part 1: Implement a greedy strategy for Shunyajit. Suppose that at some point of time, Shunyajit is at position $(i, j) \neq (n-1, n-1)$. Consider the three cases. First, if $i = n-1$, then a vertical movement cannot be made, so the next position of Shunyajit must be $(i, j+1)$. Second, if $j = n-1$, then a horizontal movement is not possible, so Shunyajit must move to $(i+1, j)$. Finally, consider the case that $i < n-1$ and $j < n-1$. Shunyajit makes a greedy decision for moving to the $(i+1)$-st row. He considers all values of $k \in \{j, j+1, \ldots, n-1\}$. For each k, he moves from (i, j) to (i, k) horizontally (drinking all available potions), and then makes the vertical movement from (i, k) to $(i+1, k)$ (after fighting the demons in that segment). Shunyajit chooses that k in the above range, for which his energy level at $(i+1, k)$ is maximized. Write a function **`greedy(EF,PP,n)`** realizing this greedy strategy. The function should print the movements of Shunyajit in each of the above three cases, and his energy level after the movements following each decision.

Part 2: The greedy strategy of Part 1 is not guaranteed to produce an optimal solution. Use dynamic programming to generate an optimal solution. Build an $n \times n$ table T such that $T[i][j]$ is meant to store the maximum possible energy level of Shunyajit when he is at (i, j). If both i and j are positive, then $T[i][j]$ can be related to $T[i-1][j]$ and $T[i][j-1]$ easily by the rules of the game. Handle the boundary conditions ($i = 0$ or $j = 0$) appropriately. Write a function **`DP(EF,PP,n)`** to implement this dynamic-programming algorithm.

The function should return (or print) the maximum energy level with which Shunyajit can leave the mesh at position $(n-1,n-1)$. The running time of your algorithm should be $O(n^2)$. ***Do not use memoization.***

Part 3: Copy the function `DP` to a function `DPsol`. The purpose of this function is to print an optimal path that Shunyajit should follow in the mesh. The function should print all of the horizontal and vertical movements of Shunyajit, and his energy level after each movement. The running time of `DPsol` should again be $O(n^2)$.

The *main*() function

- Read n from the user.
- Read the two matrices EF and PP from the user in row-major order (that is, row by row, and in each row, from left to right). Notice again that EF is an $(n-1) \times n$ matrix of floating-point numbers in the open interval $(0,1)$, whereas PP is an $n \times (n-1)$ matrix with entries from $\{0,1\}$.
- Call `greedy` to print the movements and the final energy level.
- Call `DP` to obtain and print the optimal final energy level.
- Call `DPsol` to print an optimal path in the mesh.

Sample output

```
n = 5

+++ Exhausting factors:
    0.8  0.8  0.9  0.8  0.9
    0.7  0.8  0.7  0.5  0.6
    0.7  0.9  0.7  0.8  0.8
    0.7  0.7  0.7  0.8  0.9

+++ Potion positions:
    1  0  0  0
    0  1  1  0
    1  1  1  1
    1  0  0  1
    0  0  0  0

+++ Part 1: Greedy algorithm
    At (0,0) with energy level = 1.000000
    At (1,2) with energy level = 1.125000
    At (2,4) with energy level = 0.843750
    At (3,4) with energy level = 0.675000
    At (4,4) with energy level = 0.607500
--- Energy level during exit   = 0.607500

+++ Part 2: Dynamic-programming algorithm
--- Energy level during exit   = 1.125000

+++ Part 3: Dynamic-programming algorithm with solution
    At (0,0) with energy level = 1.000000
    At (0,1) with energy level = 1.250000
    At (1,1) with energy level = 1.000000
    At (2,1) with energy level = 0.800000
    At (2,2) with energy level = 1.000000
    At (2,3) with energy level = 1.250000
    At (2,4) with energy level = 1.562500
    At (3,4) with energy level = 1.250000
    At (4,4) with energy level = 1.125000
```

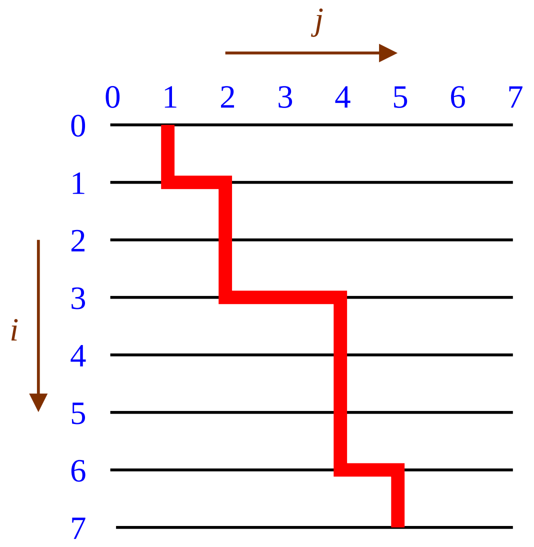
Submit a single C/C++ source file. Do not use global/static variables.

CS29003 Algorithms Laboratory

LAB TEST: ODD PC

Last date of submission: 27–March–2018

The city of Ekone has n parallel (east-west or horizontal) roads. The distance between two consecutive roads is 1 km. The mayor of Ekone wants to build a cross road that runs north-to-south (or vertically) and uses some portions of the existing horizontal roads. The cross road consists of vertical flyovers connecting two consecutive roads. Assume that the positions of the flyovers are integers in the range $0, 1, 2, \ldots, n-1$ (the unit of the distance is 1 km). This means the horizontal segments of the cross road run for integral numbers of kilometers (zero is allowed). The following figure demonstrates a construction for $n = 8$ (the cross road is the thick line).



Suppose that the flyover connecting Road i with Road $i+1$ is at position j_i. The mayor decides to obey the constraints: $0 \leqslant j_0 \leqslant j_1 \leqslant j_2 \leqslant \cdots \leqslant j_{n-2} \leqslant n-1$. The estimated cost of building a flyover from location (i, j) to $(i+1, j)$ is specified in an $(n-1) \times n$ matrix C. Assume that all $C[i][j]$ are integers (the unit may be millions of Ekonian pounds). In order to complete the construction of the cross road, the horizontal portions used by the cross road also need to be upgraded. The cost of upgrading each kilometer of horizontal portion is 5 units (irrespective of which horizontal roads the segments belong to). The total construction cost is the sum of the costs of building the $n-1$ flyovers and of the costs of upgrading the horizontal segments.

Notice that the topmost flyover can be at $j_0 > 0$. In that case, you do not include the cost of the horizontal segment $(0, 0)$ to $(0, j_0)$. Likewise, if the the bottommost flyover is at $j_{n-2} < n-1$, the cost of the horizontal segment $(n-1, j_{n-2})$ to $(n-1, n-1)$ is not to be included in the total construction cost.

The mayor calls you to work out the locations of the flyovers so that the total construction cost is minimized.

Part 1: In this part, you implement the following greedy strategy. Suppose that you have decided the locations of the first i flyovers, and are at a position (i, j_{i-1}) on the $(i+1)$-st horizontal road (the road with index i). If $i > 0$, the allowed locations of the next flyover are $j_{i-1}, j_{i-1}+1, \ldots, n-1$. For each j in this range, compute the cost of extending the cross road to use the horizontal segment from (i, j_{i-1}) to (i, j) and then to use a flyover from (i, j) to $(i+1, j)$. Among the above possibilities of j, the one which gives the minimum extension cost is taken as j_i. If $i = 0$ (that is, you are deciding the location of the topmost flyover), the search space for j is $0, 1, 2, \ldots, n-1$, and you do not include the cost of any horizontal segment. Implement this strategy in a function `greedy(C,n)`. The function should return (or print) the minimum cost computed. It should also print the locations of the flyovers, and the total cost for the horizontal segments.

Part 2: The greedy algorithm of Part 1 may fail to produce an optimal solution. Use dynamic programming to compute an optimal solution. Build an $n \times n$ table T such that $T[i][j]$ stores the minimum cost of building the part of the flyover from the top to the position (i, j). If $i > 0$ and $j > 0$, then $T[i][j]$ can be easily derived from $T[i-1][j]$ and $T[i][j-1]$. Handle the boundary conditions ($i = 0$ or $j = 0$) appropriately.

Write a function **DP(C,n)** to compute and return the minimum total construction cost of the cross road. The running time of your algorithm should be $O(n^2)$. ***Do not use memoization.***

Part 3: Copy the function **DP** of Part 2 to a function **DPsol**. The purpose of this function is to construct an optimal cross road. That is, this function should print the optimal locations of the flyovers, and also the total cost of upgrading the horizontal segments. The running time of **DPsol** should again be $O(n^2)$.

The *main*() function

- Read n from the user.
- Read from the user the array C storing the flyover-building costs, in the row-major order (that is, row by row, and in each row, from left to right). Recall that C is an $(n-1) \times n$ matrix of positive integers.
- Call **greedy** to obtain and print a greedy solution.
- Call **DP** to obtain and print the optimal (minimum) total construction cost.
- Call **DPsol** to print the flyover locations and the total cost of upgrading the horizontal segments, as recommended by an optimal solution.

Sample output

```
n = 5

+++ Flyover building costs
     7    8   13   14   20
    16    6   13   17   19
    20   13   17   10   12
    16   19   14   11   19

+++ Part 1: Greedy algorithm
    Building flyover from (0,0) to (1,0): Cost =  7
    Building flyover from (1,1) to (2,1): Cost =  6
    Building flyover from (2,1) to (3,1): Cost = 13
    Building flyover from (3,1) to (4,1): Cost = 19
    Total cost of the horizontal segments      =  5
--- Total cost = 50

+++ Part 2: Dynamic programming algorithm
--- Total cost = 45

+++ Part 3: Dynamic programming algorithm with solution
    Building flyover from (0,1) to (1,1): Cost =  8
    Building flyover from (1,1) to (2,1): Cost =  6
    Building flyover from (2,3) to (3,3): Cost = 10
    Building flyover from (3,3) to (4,3): Cost = 11
    Total cost of the horizontal segments      = 10
```

Submit a single C/C++ source file. Do not use global/static variables.