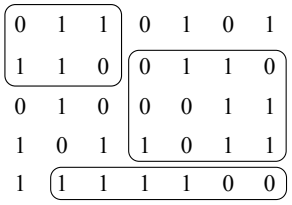
CSC 495 – Problem Set 7 – Due Tuesday, February 28

Problem 7.R1: Star Census

Required Problem
Points: 50 points

Background

Astronomers have run several large "sky survey" projects, mapping locations of stars in the sky. Imagine storing a map divided into cells, so that each cell either contains a star or does not — this map is represented by a two-dimensional array in which each element contains either a zero (no star) or a 1 (a star is in the cell). Your job is to answer queries about star density efficiently. Consider the following small example, as a 7 column by 5 row map, where three different queries are marked.



Each query is given by specifying the upper-left corner (column number followed by row number, where these numbers start at 0) and the width and height of the rectangle. The three queries illustrated above, and the answer for each (the count of the number of 1's in the rectangle) is given in the table below.

Corner	Width	Height	Count
0,0	3	2	4
3,1	4	3	7
1,4	6	1	4

The Problem to Solve

You will be given a map as described above, followed by a list of queries. The map can be as large as 5000×5000, and there can be as many as 50,000 queries. You should be able to answer all queries in less than one minute.

Hints and Techniques

Just storing the map and counting stars for each query will not be fast enough. With clever pre-processing you can answer each query in $O(1)$ time, regardless of the size of the query rectangle. This is a two-dimensional variant of one of the problems at the end of Column 8 of *Programming Pearls*. Make sure you read through those problems and understand their solutions, and then think about how you can solve this problem.

Input and Output

The input consists of two integers giving the number of columns (m) and the number of rows (n) in the map, followed by the map data row-by-row. The map data will consist of nm bits (0 or 1), but there is no specified line format (the data could be all on one line, or each row could be on a line, or each individual cell could be on a line by itself). The map data is followed by an integer q giving the number of queries, followed by q sets of 4 integers, each representing a query. A query consists of the column and row number of one corner of the query rectangle, followed by the width and height of the rectangle. For each query, you should output a line containing the number of 1's in the rectangle. The sample input and output below shows the example given in the "Background" section.

Sample Input

```
7 5
0 1 1 0 1 0 1
1 1 0 0 1 1 0
0 1 0 0 0 1 1
1 0 1 1 0 1 1
1 1 1 1 1 0 0
3
0 0 3 2
3 1 4 3
1 4 6 1
```

Sample Output

```
4
7
4
```

Problem 7.R2: Road Wear-y

Required Problem
Points: 50 points
Extra Ninja Points: 15 extra ninja points if your code can handle values of n up to 1,000,000 in under 60 seconds. Submit your code as A7R2big if you have an algorithm that is efficient enough to handle this.

Background

Consider the problem of maintaining a stretch of highway, where the amount of wear on pavement is proportional to the number of cars that have driven on it. On a controlled-access highway there are limited places where cars can enter and leave the highway, and in this problem we assume that each car is tracked and reported (the "Big Brother Highway System"). For simplicity, we assume every highway entrance is also an exit, and vice versa, so we refer to these generically as "Exits," which are numbered from 1 to n for some maximum exit number n. We are given the statistics that are gathered at regular intervals, that are reported in the form "m cars entered at Exit s and left at Exit e." Your task is to collect these statistics and find k mostly highly used highway segments (between consecutive exits), where k is an input parameter.

For example, consider a highway with 10 exits, and the following statistics:

> 12 cars entered at Exit 3 and left at Exit 8
> 6 cars entered at Exit 5 and left at Exit 2
> 3 cars entered at Exit 1 and left at Exit 10
> 3 cars entered at Exit 6 and left at Exit 7
> 4 cars entered at Exit 10 and left at Exit 1

The top 3 heaviest-used highway segments are then between Exits 3 and 4 (traveled by 25 cars), Exits 4 and 5 (traveled by 25 cars), and Exits 6 and 7 (traveled by 22 cars).

The Problem to Solve

You are given numbers n (representing the number of exits on the highway) and k (representing how many most-used segments should be reported), and a sequence of 3-tuples representing statistics (number of cars, and exit numbers for entry and exit). For the basic version of this problem, n satisfies $1 \leq n \leq 1000$ — if that sounds like a lot of exits, note that I-40 has 833 exits between North Carolina and California. There will be up to 1,000,000 individual statistics lines, with each reporting on at most 1000 cars. There are no restrictions on k other than the obvious one, that $1 \leq k \leq n - 1$. Your code should be able to process any such input in under 60 seconds.

To get the 15 extra ninja points referred to at the top of the problem, you should be able to handle values of n up to 1,000,000, where all of the other restrictions are the same (up to 1,000,000 individual statistics, etc.).

The output should be given in order of non-increasing road wear, and if there are multiple segments with the same wear always output the segments with lower exit numbers before higher

exit numbers. So, in particular, if $k = 1$ in the example above, you should output only the Exit 3/4 segment.

Hints and Techniques

Like Problem 7.R1, this problem is also similar to a problem given at the end of Column 8 of *Programming Pearls*. Note that individually adding up the number of car trips for each segment might work for the smaller version of the problem (as long as you make a relatively efficient implementation), but that naive solution has no hope for handling the "big" version of the problem.

Input and Output

The input will consist of a line containing the integers n (the number of exits), k (the number of segments that should be reported), and m (the number of measurements that have been collected). This is followed by m lines that each contain 3 integers: The number of cars represented by this line, and the exits at which these cars enter and leave the highway. You are to output k lines representing the k most heavily used highway segments as follows: Each line contains 3 numbers, the first two being exit numbers (these should be consecutive integers, with the smaller exit number given first) and the last one being the number of times that highway segment was traversed. The sample input/output below represents the problem stated in the "Background" section.

Sample Input

```
10 3 5
12 3 8
6 5 2
3 1 10
3 6 7
4 10 1
```

Sample Output

```
3 4 25
4 5 25
6 7 22
```

Problem 7.C1: Max, Max on the Range

Challenge Problem
Ninja Points: This challenge problem is worth up to 20 base ninja points

Background

Problem 7.R1 was an example of a "range query" problem: A user can make queries about ranges of elements in the input. For Problem 7.R1 the problem was counting, and addition over the integers has some nice properties that could be taken advantage of in that problem. What about other functions over ranges of values, such as "max"? In other words, what if you have a large array `x[]` of values, and want to be able to efficiently answer queries such as "What is the largest element in the range `x[12..145]` (the elements in positions 12 through 145)?" On one extreme of solutions, you could just store the array values and compute the max over each subarray when asked (so this could require $\Theta(n)$ time for a query). At the other extreme, you could pre-compute the answer to all possible queries — there are $\Theta(n^2)$ such queries, so this would take a table of size $\Theta(n^2)$. If the input array has 1,000,000 elements, how large would this be? Have you ever seen a computer with that much RAM?

What we'd like is a nice middle ground, and it turns out that is possible. You can pre-process the data in $O(n)$ time and create a data structure of size $O(n)$ such that queries can be answered in time $O(\log n)$. That's what you should do for this problem.

The Problem to Solve

You are give an array of n integer values (positive and negative), where $1 \leq n \leq 500,000$. This is followed by up to 500,000 queries, where queries are given by the starting and ending indices in the sub-array of interest (indices start at 0). You should answer each query with the value of the largest element in that sub-array. You should be able to handle any such sequence of queries in under 60 seconds.

Hints and Techniques

Trees are nice....

Input and Output

Input consists of a line containing the number n, the size of the array of data values, followed by n lines each containing an integer for the array. This data is followed by a number m, the number of queries, followed by m lines each containing a query with two indices (both in the range $0, \ldots, n-1$). The output should consist of m lines, each containing the largest value in the sub-array corresponding to the query. The following sample shows a 10 item array, with 5 different queries.

Sample Input

```
10
4
9
1
-3
5
8
2
10
1
0
5
0 9
0 5
3 3
4 6
7 8
```

Sample Output

```
10
9
-3
8
10
```

Problem 7.C2: Burning Rectangles

Challenge Problem
Ninja Points: This challenge problem is worth up to 20 base ninja points

Background

Consider a computer screen that displays solid rectangles over time. Unfortunatley, the pixels on the screen are susceptibale to burning out if they have been lit for a long time. We want to track how long each pixel has been lit, so we can determine which have been lit the longest (so are at highest risk for wearing out). For example, consider a 6×6 screen that has displayed 4 rectangles for different amounts of time, as given in the following table:

Minutes	Corner	Width	Height
3	3,3	3	3
5	4,4	1	1
7	1,3	3	4
7	4,3	3	1

Drawing a picture of this screen, where each pixel is marked with the total number of minutes that it has been on (and marking each of the four rectangles from the table above):

0	0	0	0	0	0
0	0	0	0	0	0
7	7	10	10	10	7
7	7	10	8	3	0
7	7	10	3	3	0
7	7	7	0	0	0

If asked for the 7 pixels that have been lit the longest, you would output the 5 pixels that have been lit for 10 minutes, the pixel that has been lit for 8 minutes, and one of the 7-minute pixels.

The Problem to Solve

You are to read the description of the screen size, followed by a list of rectangles and times lit, as described above. The screen can be as large as 1000×1000 and there can be as many as 1,000,000 rectangles given, each lit for at most 1000 minutes. You should be able to process any such input in under 60 seconds.

Hints and Techniques

This is a two-dimensional version of problem 7.R2, although you must use the efficient algorithm (there is no "small" version).

Input and Output

The input consists of one line that contains the size of the screen (width then height), the number of pixels requested, and the number of rectangles in the input. Then each rectangle is presented on a line using 5 integers, in the same order as the table shown in "Background" (the number of minutes lit, the column and row of the upper-left corner, and the width and height of the rectangle). You should output descriptions of the requested number of pixels, sorted in non-increasing order of pixel wear. For each pixel, give the coordinates (column then row, separated by a comma), a colon and a space, and finally the total number of minutes that pixel has been lit. In ordering pixels, ties should be broken by pixel position: output pixels left-to-right, and then top-to-bottom within a column. The sample input and output below reflect the problem given in the "Background" section.

Sample Input

```
6 6 7 4
3 3 3 3 3
5 4 4 1 1
7 1 3 3 4
7 4 3 3 1
```

Sample Output

```
3,3: 10
3,4: 10
3,5: 10
4,3: 10
5,3: 10
4,4: 8
1,3: 7
```