COMP 482 Project 3: Generalize Binary/Bisection Search

Due: Wednesday November 30 at 2355

Points: 30 points possible

Overview: Binary search finds whether a given target item is in a sorted array by checking the middle entry. If the middle entry is the value, it has been found. If the middle entry is smaller than the target it continues the search on the right half. If the middle entry is larger than the target then it continues on the left half.

What if you had an m by n array where each row and each column was sorted and you wanted to find a given target item. There are obvious slow methods (just searching row by row and column by column would be $O(m \cdot n)$, searching each row using binary search would be $O(\lg m \cdot n))$, but there is a divide and conquer method that check the item in the middle row and middle column. If the value in the middle row and middle column is the target, it has been found. If the value in the middle row and column is too small, we can rule out the top left portion of the array. If the value in the middle row and middle column is too large we can rule out the bottom right portion of the array. In either case where the value is not found, we have 3 cases each approximately 1/4 the original size (or if you prefer 2 cases: 1 approximately 1/4 of the original size and one approximately 1/2 the original size).

Details: The input will come from a file called input.txt which will be placed in the same directory as your java file. The first line of the file will have a two integer values M and N which will be the number of rows and columns. The second line of the file will contain a single value T, the target. The next M lines will each contain N values in increasing order (for any $1 \leq i \leq N$ the ith item in the next M rows will also be increasing).

Program the divide and conquer algorithm described above to find the location of the target (if it is in the array). If found, print out the row and column. If not found, print out NOT FOUND. I will be testing the algorithm on large enough examples that I'll notice how slow they run.

See the sample input below for an example.

You can discuss ideas for the algorithm to be used with anyone and consult any source (books, internet, etc). However, for this project, you are expected to write the code on your own with limited or no assistance from the professor, no assistance from others, and limited or no assistance from other sources (books, internet, etc). To clarify, you can seek assistance in understanding the task and how it can be solved, but "your code" should be written by you: not written by others, not copied from others, not copied from books/internet.

Picky, but required specifications: Your project must:

- be submitted via canvas.
- consist of 1 or more dot-java files (no class files, zip files, input files or other files should be submitted). Each file must have your name and which project you are submitting as comments on the first 2 lines.
- not be placed into any package (for the java pedants, it must be in the default package).
- have one file called Project3.java.
- compile using the command 'javac Project3.java'.
- run using the command 'java Project3',
- accept input from a file called input.txt in the same directory as the java file(s) formatted precisely as described above.
- be submitted on time (early and multiple times is fine).

If your project fails any of the above, you will receive a zero (recall that each of you may replace 1 and only 1 project that receives a zero this semester). If your project meets the requirements above then it will be graded on whether it:

- is designed and formatted reasonably (correct indentation, no excessively long lines, no excessively long methods, has useful method/variable names, etc) and

- accomplishes the goal of the project. In other words, the output should be the correct answer, computed in a valid way, formatted correctly.

Sample execution: If input.txt contains

```
6 5
9
 1  3   5  9 12
 2  7   8 11 13
 4  19 21 25 26
 6  20 24 37 39
40  41 45 47 48
42  43 50 55 57
```

then the output should be

```
1 4
```

because you checked position (3,3) = 21 (too large) which rules out the sub array

```
21 25 26
24 37 39
45 47 48
50 55 57
```

and leaves the 3 sub-arrays

```
 1  3
 2  7
```

```
5  9 12
8 11 13
```

and

```
 4  19
 6  20
40  41
42  43
```

Now you check each of these 3 recursively.

In the first you check position (1,1) = 1 (too small) and yields 3 subarrays of size 1x1 (base case).

In the second you check position (1,4) = 9 (correct) print out.

In the third you check position (4,1) = 6 (too small) and yields 3 sub arrays of size 2x1 (problem reduces to ordinary binary search.

Stray Thoughts:
I suggest you finish and submit your project at least several days in advance. This way you have time and opportunity to ask any last questions and verify that what you upload satisfies the requirements. There is nothing wrong with working on the project for a day and uploading your code, working for another day and uploading your improved code, ..., working for another day and uploading your final version. In fact there are advantages: you have a fairly reliable place that keeps your versions and even if you get busy at the last moment you have still uploaded your best version.
Your project should be written and understood by you. Helping or receiving help from others to figure out what is allowed/required is fine, but copying code is not. Significant shared source code indicates that you either did not write/understand what you submitted or you provided code to another (allowing them to submit code they did not write/understand). Both are academic dishonesty.