Project 2: Dynamic programming

COT 4400, Spring 2024

Due March 31, 2024

1 Overview

For this project, you will develop an algorithm to locate an optimal alignment of a sequence in a matrix. Designing and implementing this solution will require you to model the problem using dynamic programming, then understand and implement your model.

You are only allowed to consult the class slides, the textbook, the TAs, and the professor. In particular, you are not allowed to use the Internet. This is a group project. The only people you can work with on this project are your group members. This policy is strictly enforced.

In addition to the group submission, you will also evaluate your teammates' cooperation and contribution. These evaluations will form a major part of your grade on this project, so be sure that you respond to messages promptly, communicate effectively, and contribute substantially to your group's solution. Details for your team evaluations are in Section 6.2. You will submit the peer evaluations to another assignment on Canvas, labelled "Project 2 (peer eval)."

2 Problem Description

In this project, we will evaluate an alignment optimization problem in which a sequence is being overlaid on top of a matrix, where the sequence may start at any position in the matrix, subsequent values in the sequence may move in any of 4 directions (up, down, left, or right), and we are trying to minimize the absolute difference between the sequence and the matrix values it overlays. Formally, given a sequence s of size k and a matrix A of size m by n, we are looking for a sequence of matrix indexes (r_1, c_1), (r_2, c_2), $\ldots$, (r_k, c_k) such that

- All matrix indexes are valid (row indexes between 1 and m, column indexes between 1 and n)
- Consecutive indexes are one step horizontal or vertical ($r_{i+1} = r_i$ and $c_{i+1} = c_i \pm 1$ or vice versa)
- The alignment minimizes the absolute difference score D between the sequence s and the matrix A it overlays:

$$D = \sum_{i=1}^{k} |s[i] - A[r_i, c_i]|;$$

As an example application, this problem could be used to solve a *zig-zag word search*, in which we are given a list of words to search for in a grid and the words may move up, down, left, or right

in the matrix and change direction mid-word. For example, consider the zig-zag search where we are searching for the word *adapt* in the grid below:

d	a	t	p	a
a	e	p	e	h
a	h	a	d	s
d	t	e	a	p
n	e	d	d	a

This puzzle can be modeled by aligning the sequence $[1, 4, 1, 16, 20]$ in the matrix

$$\begin{bmatrix} 4 & 1 & 20 & 16 & 1 \\ 1 & 5 & 16 & 5 & 8 \\ 1 & 8 & 1 & 4 & 19 \\ 4 & 20 & 5 & 1 & 16 \\ 14 & 5 & 4 & 4 & 1 \end{bmatrix},$$

which has two unique solutions with absolute difference 0, one of which is highlighted below:

$$\begin{bmatrix} 4 & 1 & 20 & 16 & 1 \\ 1 & 5 & 16 & 5 & 8 \\ 1 & 8 & 1 & 4 & 19 \\ 4 & 20 & 5 & 1 & 16 \\ 14 & 5 & 4 & 4 & 1 \end{bmatrix}$$

Note that in this version of the alignment problem, the indexes are *not* required to be distinct. For example, consider the problem where we are searching for the sequence *banana* in the grid:

b	a	n

This problem has an optimal solution (difference 0) that starts on the left and moves right, right, left, right, and left.

The second optimal solution for the *adapt* problem is to start at row 3, column 3, and move right, left, up, and up.

3 Project report

You will need to prepare a report in which you answer the 9 questions below. *Note that questions 1–7 refer to the problem of calculating the minimum absolute difference rather than actually computing the alignment coordinates.* You must use dynamic programming to solve this problem; other solutions will not receive significant credit.

1. How you can break down a large problem instance into one or more smaller instances? Your answer should include how the minimum absolute difference for the original problem is constructed from the minimum solutions to the subproblems and why this breakdown makes sense.

   Hint: consider a restriction of the problem where we are given the coordinates for the first row and column of the alignment, and you can return the minimum difference among all possible starting points.

2. What recurrence can you use to model this problem using dynamic programming?
3. What are the base cases of this recurrence?
4. Give pseudocode for a memoized dynamic programming algorithm for the problem of identifying the minimum possible alignment difference.
5. What is the time complexity of your memoized algorithm? Justify your answer.
6. Describe a valid iteration order you would recommend for an iterative algorithm for this problem in pseudocode.
7. Give pseudocode for an iterative algorithm for the problem.
8. How could you modify your solution to problem of identifying the minimum absolute difference to actually locate the alignment in the grid? You may answer this question by giving pseudocode for an algorithm that computes and returns the alignment, or you may describe (in English) how to modify the iterative or memoized algorithm.
9. Can the space complexity of your iterative solution for finding the minimum score or alignment be reduced? Why or why not?

4 Coding your solutions

In addition to your report, you should implement a dynamic programming algorithm to solve the problem of finding the optimal sequence alignment. Your code may be an iterative or memoized dynamic programming algorithm, and it may be written in C or C++, but it must compile and run on the C4 Linux Lab machines. If compiling your code cannot be accomplished by the command

```
g++ -o align *.cpp
```

you should include a Makefile that capable of compiling the code via the `make` command.

Your source code may be split into any number of files. Your program will not need to handle invalid input nor problem instances with a play area with more than 100 rows or columns.

4.1 Input format

Your program should read its input from the file `input.txt`, in the following format. The first line contains the length of the sequence k and the number of rows and columns in the matrix (m and n), separated by spaces. The second line contains the sequence, which will be k integers, separated by spaces. The following m lines will each contain a row of the matrix (n integers separated by spaces).

For example, the input file for the *adapt* problem described earlier would be:

```
5 5 5
1 4 1 16 20
4 1 20 16 1
1 5 16 5 8
1 8 1 4 19
4 20 5 1 16
14 5 4 4 1
```

As another example, the input file for the *banana* problem would be:

```
6 1 3
2 1 14 1 14 1
2 1 14
```

4.2 Output format

Your program should write its output to the file `output.txt`. This file should have three lines. The first line should consist of the minimum alignment score. The second line should contain the initial row and column for the first element of the sequence, separated by spaces. The row and column indexes should be given with a base of 1 (i.e., the upper-left corner of the matrix is $(1, 1)$, not $(0, 0)$). The final line should consist of $k - 1$ letters separated by spaces, indicating how the optimum alignment should proceed (`U`, `D`, `L`, or `R`, representing up, down, left, or right).

For example, the solution to the *adapt* problem would be:

```
0
4 4
U L U U
```

or:

```
0
3 3
R L U U
```

and the solution to the *banana* problem would be:

```
0
1 1
R R L R L
```

5 Implementation hints

1. Start by solving the version of the problem described in the hint for question 1, where you are not only given the sequence and matrix, but also the starting row and column for the alignment *and* you are only trying to return the minimum difference instead of the alignment itself.

2. The test cases will not be any longer than a sequence of length 1000 and a matrix with 100 rows and 100 columns. The minimum absolute difference might not be 0, though.

6 Submission

Your submission for this project will be in two parts, the group submission and your individual peer evaluations.

6.1 Group submission

The submission for your group should be a zip archive containing 1) your report (described in Section 3) as a PDF document, and 2) your code (described in Section 4). If your code requires more than a simple command to compile and run then you must also provide a Makefile and/or shell script. You should submit this zip archive to the "Project 2 (group)" assignment on Canvas.

Be aware that your project report and code will be checked for plagiarism.

6.2 Teamwork evaluation

The second part of your project grade will be determined by a peer evaluation. Your peer evaluation should be a text file that includes 1) the names of all of your teammates (including yourself), 2) the team member responsibilities, 3) whether or not your teammates were cooperative, 4) a numeric rating indicating the proportional amount of effort each of you put into the project, and 5) other issues we should be aware of when evaluating your and your teammates' relative contribution. The numeric ratings must be integers that sum to 60.

It's important that you be honest in your evaluation of your peers. In addition to letting your team members whether they do (or do not) need to work on their teamwork and communication skills, we will also evaluate your group submission in light of your team evaluations. For example, a team in which one member did not contribute would be assessed differently than a team with three functioning members.

You should submit your peer evaluation to the "Project 2 (peer eval)" assignment on Canvas.

7 Grading

Report	**50 points**
Questions 4 and 6	8 each
Questions 1 and 8	7 each
Questions 2, 3, 5, 7, and 9	4 each
Code	**20 points**
Compiles and is correct	15
Good coding style	5
Teamwork	**30 points**

Note that if your algorithm is inefficient, you may lose points for both your pseudocode and your submission.

Also, in extreme cases, the teamwork portion of your grade may become negative or greater than 30. In particular, if you do not contribute to your group's solution at all, you can expect to receive a total grade of 0.