CompSci 260P Spring 2023 Project 2

Due: April 24, 2023 at 6:00 PM. This project is eligible for late submissions.

Requirements

You must implement the function in gridynp.cpp, which solves the problem given. The function is called solve, returns an unsigned integer, and takes as a parameter a 2D grid of std::string. You are guaranteed that this grid is a square, with dimensions N by N, for some integer N in the range [2, 100].

What is the problem? You are walking along a grid, starting at the top-left element (0,0). Your goal is to get to the bottom-right corner, element (N-1, N-1). Every time you move, you may move either right one or down one. You may not move up, nor left, nor may you move more than one spot at once. You may not stand still at any iteration.

You begin with some positive integer number of hit points.

Every square of the grid has one of four types of marking.

- A string consisting of a positive integer x. Walking into this square increases your health by the given value. This will never be a value higher than 1000.
- A string consisting of a negative integer x. Walking into this square decreases your health by the absolute value of x. For example, if the string is “-5” then you lose 5 HP by walking into that square.
- The string “P”. Your hit points don’t change upon walking into this square, but the next time you enter a square that decreases your health, its effect is canceled instead. The effect of this does not stack: if you walk through two such squares and then two squares with negative integers, you take damage from the second one.
- The string “D”. Your hit points don’t change upon walking into this square, but the next time you enter a square that increases your health, its effect is doubled. The effect of this does not stack: if you walk through two such squares and then two squares with positive integers, the second one only heals the stated value.

It is possible to be under the influence of a “P” effect and a “D” effect at the same time. If you are under the effect of both, and you walk into a square that increases your health, the effect is doubled (use of the “D” effect), and you are still under the influence of the “P” effect.

The return value of the function is an unsigned integer representing the answer to the following question: what is the fewest hit points you could begin with, such that you can walk from the top-left to the bottom-right corner while never having fewer than one HP?

If you are stuck, here is a hint. I suggest not reading this hint until you are stuck.

Regardless, continue at the next page. How would you solve it if it were in 1D instead of 2D?
For this project, you have a few requirements:

- You must implement the function in gridynp.cpp. You are welcome to introduce helper functions as well if you so choose. If you use additional files, remember to make sure they end up on git.

- Each test case must run in under ten seconds\(^1\) on the grader’s computer. Test cases that take longer than this to run may be deemed to be incorrect. Note that this means you may need to think a little about efficiency in your program. It also imposes a restriction on me; in this case, the largest input grid is 100 x 100. For an N by N grid, it is possible to solve the problem in O(N\(^3\)).
  - Be very careful if you are solving this recursively without memoization.
  - When you pass parameters, be very careful you do not pass them by value unless you are sure you intend to do so. If you do not know what that warning means, read about passing by value and reference in C++.
  - Remember there is a chance your computer is faster than the grader’s

- Your solution must be based on dynamic programming.

- You may assume all inputs are valid

You may use standard libraries as appropriate, unless there is one that makes solving this problem trivial. I am unaware of any such part of the library.

You are explicitly permitted to use C++ standard library container classes (std::vector, std::set, etc). You are welcome to ask anything you want about these libraries, or to look up material about them online. Information about how to use an explicitly-permitted library may always be shared among classmates, but refrain from telling one another how you solved a problem in the assignment with them. For example, answering “how do I check if an element is in a std::set?” is great and encouraged, while answering “what did you use std::set for in your project?” is not.

A good reference for the STL container classes (such as those listed above, including std::map) is [http://www.cplusplus.com/reference/map/map/](http://www.cplusplus.com/reference/map/map/).

---

\(^1\) Technically, I use processor minutes for this, so (a) if I have something else running, it doesn’t count against you when it’s on the processor, and (b) someone cannot get around this by spinning up extra threads. You can use `timeout 10s ./run gtest --gtest_filter=SampleTests.Sample1` to test that particular test case, with a timeout of 10 seconds, on your own machine if you want. I do run test cases when I don’t have other things running though.
Your grade on this project

There are 10 points possible on this project; they are only available by test cases. We will run test cases with the code you submit; each test case is worth some fraction of the grade. Each test case is graded independently, other than that a single instance of non-compiling code will prevent our program from grading at all, resulting in a score of zero.

Unlike other projects this quarter, there are no “required” test cases to be graded. There are some sample cases provided, and failing those cases would be a strong sign that you will probably not like your grade on this project, but there are no a priori must-pass-to-be-graded test cases for this project this quarter.

Test cases that take longer than ten seconds to run on the grader’s computer may be deemed incorrect runs, even if a longer amount of time available to them would cause a correct answer. Note that the grader’s computer may be slower than yours, especially if you have a very new computer, such as an M1/M2. Do not assume that the speed on your computer will match that on the grader’s!

There is also a style concern for this quarter: under no conditions may your code contain the following declaration: using namespace std; Projects that contain that declaration are worth zero points, regardless of other considerations. This rule is in effect for all projects in CompSci 260P this quarter.

For more information about grading, including how to submit a late project, see the relevant section of the lab manual.

Remember that the purpose of this project isn’t to find a program that solves this problem, but rather to code it yourself and to figure out the algorithm you need for yourself. Submitting work that isn’t yours (for any reason) is a decidedly bad idea. If I find that you submitted code you found online, or that you shared a solution with a fellow student, your grade will be worse than if you hadn’t turned in this project at all. Similarly, do not post your solution online -- you can get in trouble if this assignment is given in a future quarter and a student submits your work.

Remember that this is not a group project.