PROJECT ASSIGNMENT 4

Issue Date : 19.12.2023 - Tuesday
Recitation Date : 20.12.2023 - Wednesday (19:00) (on Zoom)
Due Date : 30.12.2023 - Saturday (23:00)
Advisor : R.A. Görkem AKYILDIZ
Programming Language : Python 3.6.8

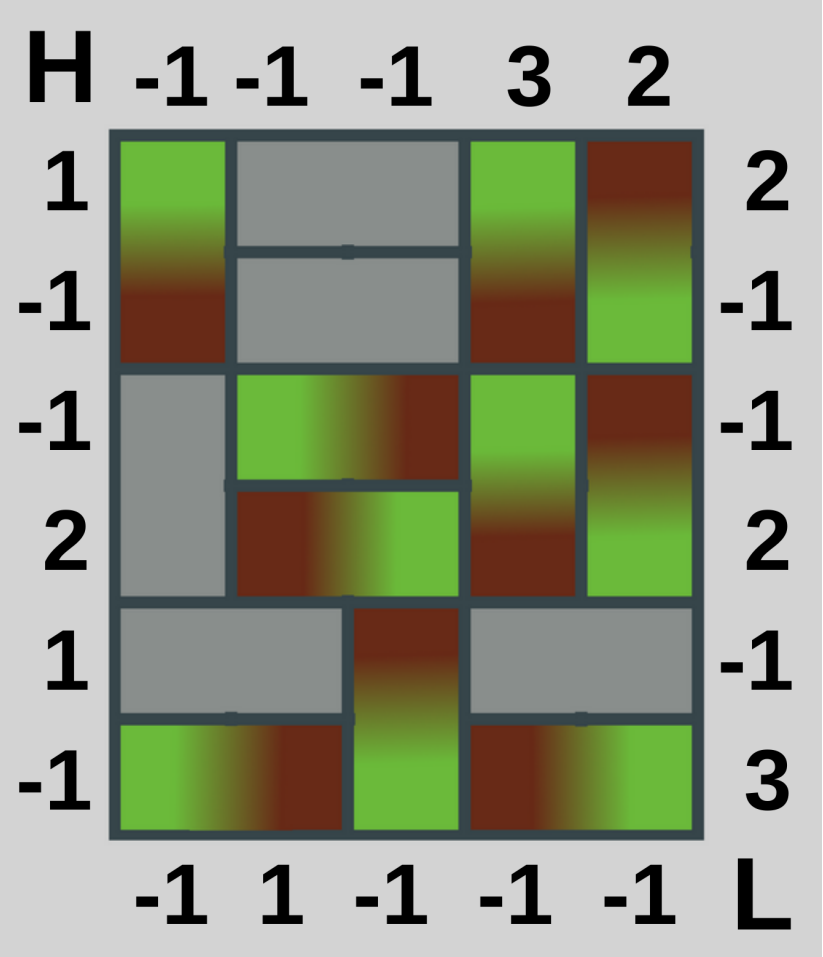

Figure 1: Visual representation of the Second I/O Pair. Green corresponds to Base, Brown corresponds to High, Gray corresponds to Neutral. Numbers at the left corresponds to High constraint of that row while the ones at the right corresponds to Low constraint; same rule is valid for the top and bottom respectively.

1 Introduction

Backtracking is a class of algorithms for finding solutions to some computational problems, notably constraint satisfaction problems, that incrementally builds candidates to the solutions, and abandons a candidate ("backtracks") as soon as it determines that the candidate cannot possibly be completed to a valid solution.[1]

If it is wanted to visualize backtracking, it can be thought as a tree structure. Say that there is a road in front of you that divides into two, and at the end of the each path, it divides into two until the finish, and you do not know which path brings you to your destination. The way that you can apply is as follows; try all possible paths one by one to discover which one brings you to the destination. Trying them randomly does not help too much as it is not a good systematic way. Instead, you can try each possible path in a more systematic way. For example, say that, firstly, you are trying to go to the leftmost path and if it reaches to your destination, then everything is good for you, but if it is not, then it would be clever to move just one step backwards and try the road at the right instead of the one at the right, and if it still does not bring you to your destination, it is clear that you have to move two steps backwards as there is no way out at the path that is leaded by turning left at two steps backwards, and you may try the road at the right and so on until you reach your destination.

On the other hand, you may think it in Sudoku way. Although at your Project Assignment 2 it was always guaranteed that there will be at least one cell with one possibility at each step, say that it was otherwise. Then your approach should be as follows: Say that there are three cells left and they have more than one probability, just call these cells as A, B, and C; say that you filled the cell A with one of its probabilities and continued with the cell B with its eliminated probabilities as you just filled A, and then continued with the cell C, but if it gets resulted with no possible probabilities for C, that means you either placed wrong number to the cell A or B, so, you moved one step backwards to try the B with its other possible probabilities, and if it still does not fit, it is obvious that the problem was A, and after you changed it to try all possibilities, it eventually brings you to the correct solution.

As it is seen on the both examples, the solution actually tries one of the candidates and backtracks if it results with an unsuccessful situation. If it is thought as a tree structure, it is clear that backtracking is a problem that is recursive inherently, because at each step, it tries to solve the same problem but with less amount of constraints (steps).

In this project, you are requested to solve a puzzle namely "Blind Valley" which is a combination of "Blind Alley" and "Valley". It is called so as there are a few true solutions where the others are just blind alley, and as it is about structuring valleys with given domino-like tiles.

At this puzzle game, you are given a template that consists of empty one by two (1x2) grids (either in horizontal or vertical direction) to place the domino-like tiles in them with given rule set. There are two kind of domino-like tiles and they can be rotated in any of the four directions; one of them is neutral, which means it does not ruin any rules, the other one is the tile that has high at the one end and base at the other hand. You can use them as much as you want but there are three rules to place them:

- No direct neighbour of a high cannot be high, and no direct neighbour of a base cannot be base. Diagonal neighbours are not important.
- The tiles must placed according to the given layout, and each grid at the layout must be filled with one of the tiles.
- There are constraints about each row and column about number of highs and bases, they must be satisfied.

2 Definition of Input

First four lines of the input are the constrains about count of highs and bases. First line consists of exact number of highs allowed for each row according to their order row by row. Second line consists of exact number of bases allowed for each row. The third and the fourth lines contains number of highs and bases allowed respectively for the columns. Note that -1 (minus one) means that there is no constraint for that row or column.

Following lines are the template for the table that is going to be filled. Each line corresponds to each row and each character corresponds to each cell. The cells are separated with space for the sake of better view. There are four characters that has meaning as follows: L corresponds to the left part of a horizontal tile place where R corresponds to the right part of that tile. U corresponds to the upper part of a vertical tile place where D corresponds to the lower (down) part of that tile.

3 Definition of Output

At the output, you will just printout the resulting table which is filled with given domino tiles. The table must be printed line by line where lines correspond to rows. The text representation of the tiles are H for High, B for Base, and N for Neutral. Note that if there is no solution for given table, your program must printout only one line that consists of "No solution!" (without quote marks) to the output.

4 Hints for the Solution

There are several ways to solve this game where some of them are iterative, some of them are recursive. Some of them are random, some of them are structured. **For this project, you must follow the recursive approach with backtracking as a structure to follow.** While solving a problem with backtracking, normally it is up to programmer to select which path to follow at the beginning as at the average it does not matter which path have been selected to start with if the problem is randomized enough, but for this project, for the sake of meeting on a common ground, you have to start from leftmost and the topmost cell of the table (left and top are according to the shape of it at the input) and continue with the cell at its right until that row ends, then you have to continue with the following row until reaching the rightmost and bottommost cell. While trying to place the tiles, following rules must be followed: If the current empty tile space is horizontal (L-R), begin with L part is filled with High while R part is filled with Base, if there is no solution for that trial, continue with L part is filled with Base while R part is filled with High, and if there is no solution either, continue with the neutral tile for both of them. If the current empty tile space is vertical (U-D), the rule is the same with the horizontal tile except the U can be assumed as L and D can be assumed as R at that explanation. In short, begin with U is High, D is Base, then continue with U is Base, D is High, and neutral for both of them at the last step.

IMPORTANT NOTE: As there may be more than one solution for a table, if the explained structure is not followed correctly, even if the code solves the problem correctly, you may face with point deductions (as getting as low as zero) as your solution may not exactly match with the given solution.

5 Restrictions

- Your code must be able to execute on our department's developer server (dev.cs.hacettepe.edu.tr).
- You must obey given submit hierarchy and get score (1 point) from the submit system.
- **You must benefit from recursion.**
- Your code must be clean, do not forget that main method is just a driver method that means it is just for making your code fragments run, not for using them as a main container, create functions in necessary situations but use them as required.
- You must use comments for this project and you must give brief information about the challenging parts of your code. Do not over comment as it is against clean code approach. Design your comments so that they make your code fully understandable and not excessive for others.
- You can benefit from Internet sources for inspiration but do not use any code that does not belong to you.
- You can discuss high-level (design) problems with your friends but do not share any code or implementation with anybody.
- Do not miss the submission deadline.
- Source code readability is a great of importance. Thus, write READABLE SOURCE CODE, comments, and clear MAIN function. This expectation will be graded as "clean code".
- Use UNDERSTANDABLE names to your variables, classes, and functions regardless of the length. The names of classes, attributes and methods should obey Python naming convention. This expectation will be graded as "coding standards".
- You can ask your questions through course's Piazza group, and you are supposed to be aware of everything discussed in the Piazza group. General discussion of the problem is allowed, but **DO NOT SHARE** answers, algorithms, source codes and reports.
- All assignments must be original, individual work. Duplicate or very similar assignments are both going to be considered as cheating.
- Submit system for this homework will be opened a few days before deadline, so please be patient.

6 Execution and Test

Your code must be executed under **Python 3.6.8** at **dev.cs.hacettepe.edu.tr**. If your code does not run at developer server during the testing stage, then you will be graded as 0 for code part even if it works on your own machine. Sample run command is as follows:

- python3 blind_valley.py input.txt output.txt

7 Grading

Task	Point
Correct Output	80
Clean Code & Comment	20*
Total	100

*** The score of the part of clean code & comment will be multiplied by your overall score (excluding clean code & comment part) divided by the maximum score that can be taken from these parts. Say that you got 60 from all parts excluding clean code & comment part and 10 from clean code & comment part, your score for clean code & comment part is going to be 10*(60/80) which is 7.5 and your overall score will be 60+7.5=67.5.**
Note that you must score one at the submit system, otherwise 20% of your grade will be deducted, moreover, you must implement a main function otherwise 10% of your grade will be deducted! There may also be other point deductions if you do not obey the given rules, such as if you do not use recursion as necessary.

8 Submit Format

File hierarchy must be zipped before submitted (Not .rar, only not compressed .zip files because the system just supports .zip files).

```
- b<StudentID>.zip
    - blind_valley.py
```

9 Late Policy

You have two days for late submission. You will lose 10 points from maximum evaluation score for each day (your submitted study will be evaluated over 90 and 80 for each late submission day). You must submit your solution in at the most two days later than submission date, otherwise it will not be evaluated. Please do not e-mail to me even if you miss the deadline for a few seconds due to your own fault as it would be unfair for your friends, e-mail submissions will not be considered if you do not have a valid issue.

References

[1] Backtracking - wikipedia. https://en.wikipedia.org/wiki/Backtracking (Last access: 18.12.2023).