Assignment 1: Backtracking Tipover Solver

Objectives

- to use backtracking to solve a puzzle

Introduction

Tipover is a puzzle played on a rectangular grid (6x6 in the commercial version of the game) with stacks of crates of various heights placed on it. The player has a tipper who starts atop one of the stacks. On each turn, the player can move the tipper to an adjacent crate or stack of crates (the tipper can climb up and down to or from the top of a stack, so any difference in height does not matter), or the player can tip over the stack of two or more crates the tipper is currently on, as long as there is room for the crates to fall – a stack of n crates falls to fill the n spaces in the direction it is tipped, starting with the space adjacent to where it started, and those spaces must all be empty (not contain another stack, previously tipped crates, or the goal) and on the board. After a tipping move, the tipper stands on the crate that used to be at the top of the stack. (The tipper has the incredible agility required to pull off that move!) The object is to move the tipper to the target crate.

Assignment

Write a program that outputs a solution to a Tipover puzzle, or determines that no solution exists.

The input will be a text file containing the dimensions of the board, the initial position of the tipper, and a grid showing where the stacked crates are. Specifically:

- the first line will contain two positive integers separated by a single space giving the height and width of the board;
- the second line will contain two nonnegative integers separated by a single space giving the initial position of the tipper, where the first is the row index (counting from 0 at the top) and the second is the column index (counting from 0 at the left);
- there are `height` more lines of input, each containing a single string of `width` characters, each of which is either a period (`.`) to denote an empty space, an integer

between 2 and 9 inclusive to indicate a stack of crates of the corresponding height, or an asterisk (*) to denote the location of the goal.

The output should consist of a list of the tipping moves to make that will solve the puzzle. The tipping moves should be given in order with the moves having the tipper walking over or climbing to adjacent crates omitted. The output should have one tip per line, where each line gives the row and column of the stack to tip followed by the direction to tip the stack given as Δr and Δc (so, for example, up is `-1 0` and right is `0 1`). Each value on a line should be separated by a single space. If there is more than one solution then it does not matter which one you output (instead of checking for an exact match with a specific output, the test script verifies that your output represents a series of legal moves that gets the tipper to the goal).

If there is no solution then there should be no output (and there is no need to distinguish this from the case where no moves are required to complete the puzzle).

Your code should be efficient – the asymptotic running time should be $O(nk)$ where n is the number of nodes in the game tree and k is the size of the maximal possible connected component of adjacent crates for the particular input.

The focus on this class is on algorithms and not testing edge cases, and while there will be private test cases, those test cases will not include inputs that do not conform to this specification.

Example

If `tipover.dat` contains

```
6 6
3 1
.3...4
......
......
.2....
......
.....*
```

Then executing your program should produce

```
(base) [jrg94@turtle code]$ ./Tipover
3 1 -1 0
0 1 0 1
0 5 1 0
```

Submissions

So that the submission system can use the same build and execute commands regardless of language, you must supply a makefile that creates an executable called `Tipover` when called with that as the target. For Python and Java, the executable that your makefile creates can be a `bash` script that compiles (for Java) and runs your program. For example,

```
Tipover:
        echo "#!/bin/bash" > Tipover
        echo "pypy3 tipover.py \"\$$@\"" >> Tipover
        chmod u+x Tipover
```

You would then submit your source code, makefile, and log as assignment 1.

```
(base) [jrg94@turtle code]$ /c/cs474/bin/submit 1 *.java makefile log
Copying TipoverModel.java
...
Copying makefile
Copying log
(base) [jrg94@turtle code]$ /c/cs474/bin/testit 1 Tipover
/home/classes/cs474/Hwk1/test.Tipover
Executing /home/classes/cs474/Hwk1/test.Tipover

Public test script for Tipover (09/04/2020)

***** Checking for warning messages *****
Making -B ./Tipover
javac *.java
echo "#!/bin/bash" > Tipover
echo "java TipoverModel \"\$@\"" >> Tipover
chmod u+x Tipover

Each test is either passed or failed; there is no partial credit.

To execute the test labelled IJ, type one of the following commands
depending on whether the file /c/cs474/hw1/Tests/tIJ is executable or not
     /c/cs474/hw1/Tests/tIJ
     ./Tipover < /c/cs474/hw1/Tests/tIJ
The answer expected is in /c/cs474/hw1/Tests/tIJ.out.


           Small puzzles
PASSED  001. Beginner
PASSED  002. Novice
PASSED  003. Intermediate
PASSED  004. Expert

           Small puzzles: 4 of 4 tests passed
```

```
               Deductions for Violating Specification (0 => no violation)

End of Public Script

  4 of 4 Total tests passed for Tipover

           Possible Deductions (assessed later as appropriate)
               -10 Deficient style (comments, identifiers, formatting, ...)
                -5 Does not make
                -5 Makefile missing
                -5 Makefile incorrect
                -1 Log file incorrectly named
                -1 Log file lacks estimated time
                -1 Log file lacks total time
                -1 Log file lacks statement of major difficulties

***** Checking log file *****
Estimate: ESTIMATE: 2:00
Total: TOTAL: 3:00

***** Checking makefile *****
```