2022-2023 Fall CMPE480 HW1

Deadline: Check out Moodle
Late-submission policy: -10*day upto 3 days

Submit 2 files (not zipped!): last-name.py (not ipynb!) and last-name.pdf

- last-name.py file contents will be copied to Google Colab notebook. Make sure your program runs with version Python 3.7 in Google Colab environment (run `!python --version`).
    - I will not wait too long for the execution to finish. Please implement your code efficiently.
- last-name.pdf:
    - Name LastName.
    - Which search algorithms are implemented (see the list below).
    - If A*2 is implemented:
        - Clearly explain your heuristic
        - Clearly explain why it is consistent and admissible.
        - ~~Clearly explain why h2(n) dominates h1(n).~~

Instructions related to code

Copy and paste the following code segment to the **beginning** of your program.

```
import requests
search_list=["BFS","DFS","UCS","GS","A*","A*2"]

search="BFS"
target_url="https://www.cmpe.boun.edu.tr/~emre/courses/cmpe480/hw1/input.txt"

txt = requests.get(target_url).text
```

- `search` and `target_url` will be set to different values while evaluating your program.

Input file and board configuration

You will need to read a board configuration. Always a rectangular board will be provided. Example:

```
........
..dh....
..cg....
..bf....
..ae....
........
```

Here, the dots represent empty holes and the letters represent pegs in the holes. The pegs might be placed arbitrarily in this configuration.

Rules, actions, costs

Please read the SoloTest rules from https://en.wikipedia.org/wiki/Peg_solitaire

The goal is to leave one piece only. For the initial configuration shown above, there are several solutions:

- One solution:
  - a up (peg a jumps up, pegs b,c,d are removed)
  - e up (peg e jumps up, pegs f,g,h are removed)
  - a right (peg a jumps right, peg e is removed, goal achieved)
- Another solution (with less explanation):
  - a right, b right, c right, d right, a up

As noticed, different from the standard solitaire game,

- A peg is allowed to jump over multiple pegs (until an empty hole), removing all the pegs being jumped over.
- The pegs and empty holes can be placed arbitrarily in the beginning of the game.

The costs of actions:

- Up actions cost 1, right actions cost 2, down actions cost 3, left actions cost 4
- The amount of jump does not affect action costs.

You can assume there is always a solution from the initial board to the goal.

Search Algorithms to Implement

You will implement:

- `"BFS":` Breadth-first-search
- `"DFS":` Depth-first-search
- `"UCS":` Uniform-cost-search
- `"GS":` Greedy search with the heuristic function h1(n)
- `"A*":` A* search with the following heuristic function h1(n).
- `"A*2":` You will implement your own heuristic function h2(n) which should be admissible and consistent ~~and should dominate h1(n)~~.

Here h1(n)= min(nRows,nCols) - 1
where nRows/nCols correspond to the number of rows/column with pegs.

Output

Your program should output (for example):

```
Number of nodes removed from the fringe: 45
Path cost: 9
Path: a right, b right, c right, d right, a up
```

All of these fields should be correct in order to receive points. If one of them is incorrect, it means the corresponding search strategy is not correctly implemented.

!!! Important Note on Tie-Breaking!!!

1. In the Expand function, in the `for each action, result in Successor-Fn` line, you need to implement the following ordering for actions:
    - The pegs should be considered following alphabetical ordering.
    - For each peg, the following ordering should be applied: left, down, right, up
2. In `"UCS"`, `"GS"`, `"A*"`, and `"A*2"`, the above tie-breaker rules are applied if the nodes have the same costs/f-values.