COMP 472: Assignment 1

Explore heuristic search.

Submitted by: Samuel RRJ Masuy - 26590624

I certify that this submission is my original work and meets the Faculty's Expectations of Originality

File structure

In `go.py`: Main program, and generic search.

In `fringe.py`: Fringe defintion and Fringe specific definition for each algorithm.

In `puzzle.py`: State definition, and specific puzzle state definition for 8-Puzzle particularly. Definition of heuristics for 8-puzzle.

Note: The name of the heuritics defined below, match the one found in `puzzle.py`

Usage

```
python go.py
         [-h]
         [-a {astar,best,bfs,dfs}]
         [-t {manhattan,displaced,invalid,max, linear}]
         [--start START START START START START START START START START]
         [-d {easy,medium,hard,worst}] [-b BENCHMARK] [-v]

optional arguments:
  -h, --help
                        show this help message and exit

  -a {astar,best,bfs,dfs}
                        Select the algorithm you want to run. Chose between
                        astar, best, bfs or dfs.

  -t {manhattan,displaced,invalid,max, linear}
                        Select the heuristic you want to apply. Chose between
                        manhattan, displaced, invalid, linear or max.

  -d {easy,medium,hard,worst}
                        Select a predifined start state with specific
                        difficulty. Chose between easy, medium, hard or worst.
                        Note: if this option is selected, --start will be
```

```
                      ignored, also goal state is set to:
                      1   2   3
                      8   4
                      7   6   5

  --start START START START START START START START START START
                      Write the 8-Puzzle start state you would like to
                      solve. Integers between 1 and 8, blank represented
                      with B. Default start state:
                      1   2   3
                      4       5
                      7   6   8

  -b BENCHMARK, --benchmark BENCHMARK
                      Select the number of loops, you want to average on. It
                      will run all algorithms, and output results to a file.

  -v, --verbose
                      Increase output verbosity. Will show the path to the
                      goal.
```

Test Runs

Start state is: `(5, 6, 7, 4, 8, B, 3, 2, 1)`

Goal state is: `(1, 2, 3, 8, 4, B, 7, 6, 5)`

Algorithm	Heuristic	Avg. Time (s)	Node to goal	Node visited
astar	manhattan	0.9343	31	52495
astar	displaced	4.2816	31	162764
astar	max	1.0431	31	52504
astar	linear	4.4095	31	125207
astar	invalid	5.3976	31	166829
best	manhattan	0.0016	45	129
best	displaced	0.0007	31	79
best	max	0.0020	45	129
best	linear	0.0107	77	441
best	invalid	0.0051	39	203
dfs	N/A	0.9809	64835	149154
bfs	N/A	1.6932	31	181438

Algorithm	Heuristic	Avg. Time (s)	Node to goal	Node visited

Note: We chose specific start and goal state to have significant differences in terms of *time*, *node to goal* and *node visited* between the different Algorithms and Heuristics.

Note: Test runs used was possible using `--benchmark 500` parameter.

Note: To determine the *Average time*, each row in the table were ran 500 times. In total, 12 * 500 = **600** experiments ran to get theses results. It took `4:25:58`, to finish these experiments, on a remote machine.

Analysis and Explanation

We see that `A*`, no matter the heuristic chosen is always finding the optimal path. But, we see that `A*` takes in general the most time compared to all other Algorithms, and visits more node then `Best-First Search`.

`Best-First Search` only finds the optimal path when given the *displace tiles* heuristic. We see that `Best-First Search` takes the least amount of time, no matter the heuristic chosen.

`Depth-First Search` is definitly far from the optimal path, it visits a lot of nodes. But is faster then `A*` or `Breadth-First Search`.

`Breadth-First Search` finds the optimal path, but is the one that visits the most node.

Regarding heuristics, for *Manhattan* and *displaced tiles*, the results are quite unexpected. `Best-First Search` performs at its best using *displaced tiles*, while `A*` is performing really poorly using the same heuristic. Its time to execute and the number of visited node is really poor. We see that `A*` performs the best using the *Maximum* heuristic (combination of *Manhattan* and *displaced tiles*).

The *invalid* heuristic is not very conclusive in this case, other that, it is `A*` worst heuristic. And `Best-First Search` gets its second best result in terms of *node to goal*.

Sample run using provided goal state

```
Solving using: astar
Using Heuristic: manhattan
Start state is: (1, 2, 3, 4, 0, 5, 7, 6, 8)
Goal state is: (1, 2, 3, 8, 0, 4, 7, 6, 5)
```

```
astar - Solved in 0.0025 seconds

1.
 1 | 2 | 3
 4 | B | 5
 7 | 6 | 8

 cost: 8 total_cost: 0

2.
 1 | 2 | 3
 B | 4 | 5
 7 | 6 | 8

 cost: 8 total_cost: 8

3.
 1 | 2 | 3
 7 | 4 | 5
 B | 6 | 8

 cost: 9 total_cost: 17

4.
 1 | 2 | 3
 7 | 4 | 5
 6 | B | 8

cost: 10 total_cost: 27

5.
 1 | 2 | 3
 7 | 4 | 5
 6 | 8 | B

cost: 9 total_cost: 36

6.
 1 | 2 | 3
 7 | 4 | B
 6 | 8 | 5

cost: 8 total_cost: 44

7.
 1 | 2 | 3
 7 | B | 4
 6 | 8 | 5

cost: 4 total_cost: 48

8.
 1 | 2 | 3
 7 | 8 | 4
 6 | B | 5

 cost: 10 total_cost: 58

9.
 1 | 2 | 3
 7 | 8 | 4
 B | 6 | 5

 cost: 9 total_cost: 67

10.
 1 | 2 | 3
 B | 8 | 4
 7 | 6 | 5

 cost: 8 total_cost: 75

11.
 1 | 2 | 3
 8 | B | 4
 7 | 6 | 5

 cost: 0 total_cost: 75

  Total nodes to goal: 11
  Total nodes visited: 186
```