Report Assignment #1: Labyrinth

Python Version: 3.4.2

Code usage: python assign1.py <path to labyrinth input file>, default: labyrinth

Answers to the questions (a)-(e):

(a) The problem state (labyrinth) is read from a file and converted into a 2D matrix (lists in a list). This matrix is then converted into a graph representation in the function *create_graph(labyrinth)*. The possible operators are (UP,DOWN,LEFT,RIGHT,PUSH) and the current state of each node is represented by the tuple (x,y,doorstate), where x,y is the field position in the matrix and doorstate defines if the door is currently open or closed. The full state space is created by iterating through all possible field and doorstates of the given labyrinth, at a switch field, the agent can go to another branch of the graph with another doorstate. The initial state is the labyrinth position with value 2 and the original doorstate. The goal states are defined by the labyrinth position(x,y) with value 3 and any doorstate. The path cost increases by 1 for each node traversed.

(b) Uniformed methods:

- Breadth First Search (BFS): It uses a FIFO queue and thus is optimal and complete in our case (path cost do not decrease with depth). Hence, the shortest path to the goal is found with that method. But: Time and Space complexity is high. Both: O(b^d), with b=branching factor and d=depth of closest goal node. In our case the Uniformed Cost Search is the same as BFS, because path costs are increasing by one for each traversal.
- Depth First Search (DFS): Implementation similar to BFS, but it uses a LIFO stack! Thus, is not complete and not optimal. (e.g. first path has infinite depth, or higher cost than optimal path). Nevertheless, it is more efficient in terms of memory O(b*m) with m=maximum length of any path.
- Iterative deepening DFS (IdDFS): It tackles the difficulties of DFS in iteratively performing limited DFS searches. The maximum depth of any path is increased until a goal state is found or the maximum depth is reached. In that way IdDFS is complete and optimal if the path costs do not decrease with depth. The time complexity is only slightly larger than DFS but it still has a smaller space complexity than BFS. O(b*d).

Informed methods:

- Greedy best-first search: Given a heuristic function h(n), this search type sorts the queue according to their heuristic value and selects the best one. It is complete if repeated nodes are not allowed, it is easy to implement, but the result is not optimal (always goes into the direction of the goal, regardless costs). Time and space complexity are O(b^m).
- A*: Improvement of Greedy approach. Evaluation function f(n)=path_cost+heuristic_estimate. The node with the minimum value is taken from the queue. If the heuristic function is consistent (also admissible) then A* is optimal and complete.
- Iterative Deepening A*(IDA): Similar to IdDFS but not the maximum depth of the paths is iteratively increased but maximum evaluation value f is changed. It is set to the smallest f that exceeded the value in the previous iteration. In that way, the memory used

is bounded.

(c) Each of the 6 search algorithms is completely independent of the labyrinth problem. The functions expect a domain-independent graph representation (dictionary), a start and goal nodes (can be more than one in a list) and maybe a cost and a heuristic function. In that way the algorithms can be used for any type of search problem.
The domain dependent part is already described in answer (a). The graph is built with the function *create_graph(labyrinth)* as described before and the cost and heuristic functions are defined with the functions *n_func(node)* and *h_func(node,goal).* These functions are also passed to the specific domain-independent search methods.

(d) The heuristic function used in the informed search methods is the L1-norm of the given labyrinth, which represents the number of steps in vertical and horizontal direction to reach the goal if there were no borders and doors. Hence the heuristic addresses a relaxed labyrinth problem and thus h(n) <= r(n), where r(n) represents the real value, or number of steps. Thus, this heuristic is admissible. This heuristic is also consistent, as the f value cannot get smaller in each iteration. h(n)<=1+h(n'). h(n') can only decrease by one in each step.
Another heuristic function h_2 could be the Euclidean L-2 norm, but h_2<=h_1 and thus the L-1 norm is the better heuristic.

(e) The following numbers and comparisons are based on the example labyrinth given in the assignment.
Computational Time (ascending): DFS(0.03s), BFS(0.04s), Greedy(0.12s), A*(0.12s), IDA(0.32s), IdDFS(1.9s).
This might lead to the fact that DFS is the best choice, but only these algorithms result in the optimal solution (50actions): BFS, A*,IDA,IdDFS.
The solution with DFS suggests 143 actions and the Greedy algorithm results in 62. Therefore, DFS may be fast but the solution is not optimal.
In this particular case, BFS outperforms the other algorithms in optimality and computational time, but it makes no use of path or heuristic costs. Therefore, A* or IDA are, given this additional information, in general the preferable algorithms to use.