hw1_search_q11_lookahead_graph_search

Question 11: Lookahead Graph Search

0.0/4.0 points (graded)

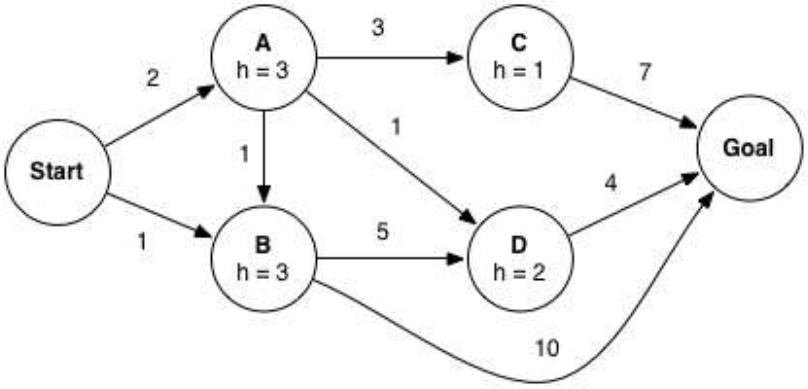
Recall from lecture the general algorithm for Graph Search reproduced below.

```
function GRAPH-SEARCH(problem, fringe, strategy) return a solution, or failure
    closed ← an empty set
    fringe ← INSERT(MAKE-NODE(INITIAL-STATE[problem]), fringe)
    loop do
        if fringe is empty then return failure
        node ← REMOVE-FRONT(fringe, strategy)
        if GOAL-TEST(problem, STATE[node]) then return node
        if STATE[node] is not in closed then
            add STATE[node] to closed
            for child-node in EXPAND(STATE[node], problem) do
                fringe ← INSERT(child-node, fringe)
            end
    end
```

Using GRAPH-SEARCH, when a node is expanded it is added to the closed set. This means that even if a node is added to the fringe multiple times it will not be expanded more than once. Consider an alternative version of GRAPH-SEARCH, LOOKAHEAD-GRAPH-SEARCH, which saves memory by using a "fringe-closed-set" keeping track of which states have been on the fringe and only adding a child node to the fringe if the state of that child node has not been added to it at some point. Concretely, we replace the highlighted block above with the highlighted block below.

```
function LOOKAHEAD-GRAPH-SEARCH(problem, fringe, strategy) return a solution, or failure
    fringe-closed ← an empty set
    fringe ← INSERT(MAKE-NODE(INITIAL-STATE[problem]), fringe)
    add INITIAL-STATE[problem] to fringe-closed
    loop do
        if fringe is empty then return failure
        node ← REMOVE-FRONT(fringe, strategy)
        if GOAL-TEST(problem, STATE[node]) then return node
        for child-node in EXPAND(node, problem) do
            if STATE[child-node] is not in fringe-closed then
                add STATE[child-node] to fringe-closed
                fringe ← INSERT(child-node, fringe)
        end
    end
```

Now, we've produced a more memory efficient graph search algorithm. However, in doing so, we might have affected some properties of the algorithm. To explore the possible differences, consider the example graph below.



If using LOOKAHEAD-GRAPH-SEARCH with an A* node expansion strategy, which path will this algorithm return? (We strongly encourage you to step through the execution of the algorithm on a scratch sheet of paper and keep track of the fringe and the search tree as nodes get added to the fringe.)

- $S \to A \to D \to G$
- $S \to B \to G$ ✔
- $S \to A \to C \to G$
- $S \to B \to D \to G$

- $S \rightarrow A \rightarrow B \rightarrow D \rightarrow G$

Fringe: [S]
Fringe-Closed: [S]

Expand S.
Fringe: [S->A (f=5), S->B (f=4)]
Fringe-Closed: [S, A, B]

Pick path with lowest f-cost. Expand S->B.
Fringe: [S->A (f=5), S->B->D (f=8), S->B->G (f=11)]
Fringe-Closed: [S, A, B, D, G]

We can keep going, but since we added G to the Fringe-Closed set, we will never add any more paths ending at G onto the fringe, so the final path returned will be the current goal-terminating path on the fringe, which is S->B->G.

Submit

Answers are displayed within the problem

problem

0.0/4.0 points (graded)
Assume you run LOOKAHEAD-GRAPH-SEARCH with the A* node expansion strategy and a consistent heuristic, select all statements that are true.

- [x] The EXPAND function can be called at most once for each state. ✔
- [x] The algorithm is complete. ✔
- [ ] The algorithm will return an optimal solution.

The first two choices are correct. The EXPAND function can be called at most once for each state because the algorithm ensures that for every state, at most one path ending at that state will ever be added to the fringe, which implies that every state gets expanded at most once.

Like GRAPH-SEARCH, the algorithm is complete. When a node, *n*, doesn't make it onto the fringe it's always the case that there is another node, *m*, which is in the fringe or the closed set with STATE[*n*] = STATE[*m*].

However, the algorithm is not guaranteed to return an optimal solution. The bug is that inserting a state into the closed list when a node containing that state is inserted into the fringe means that the first path to that state will be the only one considered. This can cause suboptimality as we can only guarantee that a state has been reached optimally when a node is popped off the fringe.

Submit

Answers are displayed within the problem