ShanghaiTech University

CS101 Algorithms and Data Structures

Fall 2021

Homework 10

Due date: 23:59, December 12, 2020

1. Please write your solutions in English.
2. Submit your solutions to gradescope.com.
3. Set your Full Name to your Chinese name and your STUDENT ID correctly in Account Settings.
4. If you want to submit a handwritten version, scan it clearly. Camscanner is recommended.
5. When submitting, match your solutions to the according problem numbers correctly.
6. No late submission will be accepted.
7. Violations to any of above may result in zero score.

1: (4*2') Multiple Choices

Each question has one or more correct answer(s). Select all the correct answer(s). For each question, you get 0 point if you select one or more wrong answers, but you get 1 point if you select a non-empty subset of the correct answers.
Note that you should write you answers of section 1 in the table below.

Question 1	Question 2	Question 3	Question 4

Question 1. *Which of the followings are true?*

(A) Like Dijkstra's algorithm, Floyd-Warshall algorithm cannot work with any graph with negative weight edge.

(B) ***With a simple optimization****, applying Floyd-Warshall algorithm to an* ***undirected*** *graph of N nodes just needs to take* ***about*** *half of the time of applying it to a* ***directed*** *graph of N nodes. (Do not need to be strictly 1/2)*

(C) If a graph has non-negative edges, it is always preferable to run the Floyd-Warshall algorithm to solve an all-pair shortest path problem than call |V| times Dijkstra's algorithm.

(D) In Floyd-Warshall algorithm, we always have the shortest path between any pair of vertexes which only passes through the vertex set $\{v_m | m \leq k\}$ after k iterations of the outmost loop.

Question 2. *Which of the followings are true?*

(A) For A search algorithm, if the heuristic is admissible, then it is also consistent.*

(B) For A search algorithm, if the heuristic is consistent, then it is also admissible.*

(C) For A search algorithm with admissible and consistent heuristic, heuristic $h_a(x)$ is always better than $h_b(x)$ if $\forall x : h_a(x) \geq h_b(x)$.*

(D) For A search algorithm with admissible and consistent heuristic, heuristic $h_a(x)$ is always better than $h_b(x)$ if $\forall x : h_a(x) \leq h_b(x)$.*

Question 3. *Which of the followings are true?*

(A) For A tree search algorithm, it will always return an optimal solution if it exists.*

(B) For A tree search algorithm with admissible heuristic, it will always return an optimal solution if it exists.*

(C) For A tree search algorithm with consistent heuristic, it will always return an optimal solution if it exists.*

(D) None of the above.

Question 4. *Which of the followings are true?*

(A) For A graph search algorithm, it will always return an optimal solution if it exists.*

(B) For A graph search algorithm with admissible heuristic, it will always return an optimal solution if it exists.*

(C) For A graph search algorithm with consistent heuristic, it will always return an optimal solution if it exists.*

(D) None of the above.

2: (6') Floyd-Warshall algorithm

Question 5. *Consider the following implementation of the Floyd-Warshall algorithm. Assume $w_{ij} = \infty$ where there is no edge between vertex i and vertex j, and assume $w_{ii} = 0$ for every vertex i. Add some codes in the blank lines to detect whether there is negative cycles in the graph. (You may not use all blank lines.)*

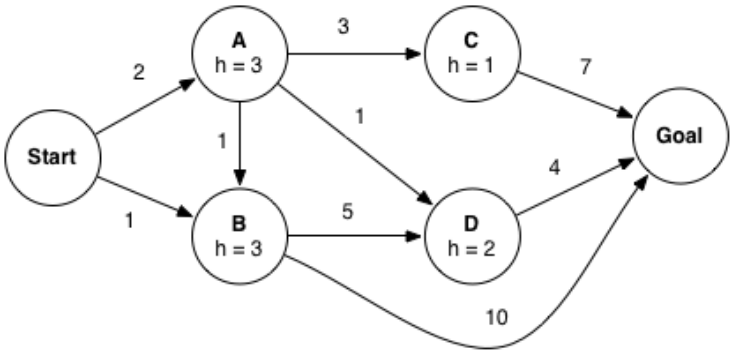
```
1  bool detectNegCycle(int graph[][V])
2  {
3      int dist[V][V], i, j, k;
4
5      for (i = 0; i < V; i++)
6          for (j = 0; j < V; j++)
7              dist[i][j] = graph[i][j];
8
9      for (k = 0; k < V; k++) {
10         for (i = 0; i < V; i++) {
11             for (j = 0; j < V; j++) {
12                 if (dist[i][k] + dist[k][j] < dist[i][j])
13                         dist[i][j] = dist[i][k] + dist[k][j];
14             }
15         }
16     }
17
18     ____________________________________
19     ____________________________________
20     ____________________________________
21     ____________________________________
22     ____________________________________
23
24     return false;
25 }
```

3: (4'+4') A* Graph Search

Here is the pseudocode of Graph Search. We can take *fringe* as the priority queue.

```
function GRAPH-SEARCH(problem, fringe) 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)
        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
```

Consider A* graph search on the graph below. Edges are labeled with costs and vertices are labeled with heuristic values. Assume that ties are broken alphabetically (so a partial plan S->X->A would be expanded before S->X->B and S->A->Z would be expanded before S->B->A).



Question 6. *In what order are vertices expanded by A* graph search? Write the expanding sequence below. (e.g. Start, A, B, C, D, Goal)*

Question 7. *What path does A* graph search return? Write the path below. (e.g. Start –> A –> C –> Goal)*

4: (4*2') A*-CSCS

After seeing the A* search algorithm, in this question we explore a new search procedure using a dictionary for the closed set, A∗ graph search with Cost Sensitive Closed Set (A*- CSCS).

```
function A*-CSCS-GRAPH-SEARCH(problem, fringe, strategy) return a solution, or failure
    closed ← an empty dictionary
    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 or COST[node] < closed[STATE[node]] then
            closed[STATE[node]] ← COST[node]
            for child-node in EXPAND(node, problem) do
                fringe ← INSERT(child-node, fringe)
            end
    end
```

Rather than just inserting the last state of a node into the closed set, we now store the last state paired with the cost of the node. Whenever A*- CSCS considers expanding a node, it checks the closed set. Only if the last state is not a key in the closed set, or the cost of the node is less than the cost associated with the state in the closed set, the node is expanded.
For the following questions, if you think the statement is correct, give a **brief** proof or explanation, if you think the statement is wrong, give a **brief** counter-example or explanation.

Question 8. *The A*-CSCS algorithm with an admissible heuristic will find an optimal solution.*

Question 9. *the A*-CSCS algorithm with a consistent heuristic will find an optimal solution.*

Question 10. *the A*-CSCS algorithm with an admissible heuristic will expand at most as many nodes as A* graph search.*

Question 11. *the A*-CSCS algorithm with a consistent heuristic will expand at most as many nodes as A* graph search.*