Homework 10

● Graded

Student

苏慧哲

Total Points

30 / 30 pts

Question 1

Multiple Choices

8 / 8 pts

- **− 1 pt** Question 1 partially correct. (BD)
- **− 1 pt** Question 2 partially correct.(BC)
- **− 1 pt** Question 3 partially correct. (BC)
- **− 1 pt** Question 4 partially correct. (C)
- **− 2 pts** Question 1 worng.
- **− 2 pts** Question 2 worng.
- **− 2 pts** Question 3 worng.
- **− 2 pts** Question 4 worng.

✔ **− 0 pts** Correct.

Question 2

Floyd-Warshall algorithm

6 / 6 pts

✔ **− 0 pts** Correct

- **− 4 pts** Wrong
- **− 2 pts** Small Mistakes.

Question 3

A* Graph Search **8** / 8 pts

3.1 **Question 6** **4** / 4 pts

✔ **– 0 pts** Correct

– 4 pts Wrong / Missing

3.2 **Question 7** **4** / 4 pts

✔ **– 0 pts** Correct

– 2 pts Partial Credit

– 4 pts Wrong / Blank

Question 4

A*-CSCS **8** / 8 pts

4.1 **Question 8** **2** / 2 pts

✔ **– 0 pts** Correct

– 1 pt no judgement

– 1 pt wrong explanation

– 2 pts totally wrong

– 2 pts blank

4.2 **Question 9** **2** / 2 pts

✔ **– 0 pts** Correct

– 1 pt wrong explanation

– 1 pt no judgement

– 2 pts totally wrong

– 2 pts blank

4.3 **Question 10** **2** / 2 pts

✔ **– 0 pts** Correct

– 1 pt Judgement wrong.

– 1 pt explanation wrong.

– 2 pts blank or wrong matching.

4.4 **Question 11** **2** / 2 pts

✔ **– 0 pts** Correct

– 1 pt Judgment wrong.

– 1 pt Explanation wrong.

– 2 pts Blank or mis-matched.

No questions assigned to the following page.

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.

Question assigned to the following page: 1

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
B D	B C	B C	C

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.*

Questions assigned to the following page: 1 and 2

(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     for (k = 0; k < V; k++) {
19         if (dist[k][k] < 0)
20             return true;
21     }
22
23     return false;
24 }
```

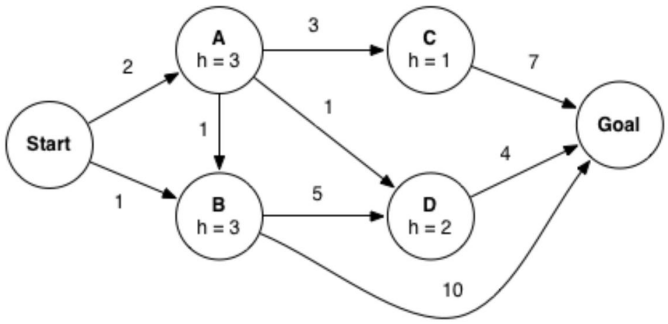
Questions assigned to the following page: 3.1 and 3.2

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)*

Start, B, A, D, C, Goal

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

Start –> A –> D –> Goal

Questions assigned to the following page: 4.4, 4.1, 4.2, and 4.3

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.

Notations:

Let $f(n)$ be the actual cost of the optimal path P start from s, end with t and pass through n.
Let $g(n)$ be the actual cost of optimal path from s to n.
Let $h(n)$ be the actual cost of optimal path from n to t.
Then, $f(n) = g(n) + h(n)$.
Let $\hat{g}(n)$ be an estimate of $g(n)$ and $\hat{h}(n)$ be an esitamte of $h(n)$.
By defination, $\hat{g}(n) \geq g(n)$.
An estimate of $f(n)$ is $\hat{f}(n) = \hat{g}(n) + \hat{h}(n)$.
$COST[node]$ equals to $\hat{g}(node)$.
A^* strategy always remove node with smallest $\hat{f}(node)$ in the fringe.
An admissible heuristic means $\hat{h}(n) \leq h(n)$ for all n in the *problem*.
An consistent heuristic means $\hat{h}(n) + h(m, n) \geq \hat{h}(m)$.
The A^* will terminate as soon as t is removed from $fringe$.

Questions assigned to the following page: 4.4, 4.1, and 4.2

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

Yes.
(Please refer to the notations on the top of this section.)
Lemma 1: If t is not in *closed*, and for any optimal path T from s to t, there exist a node n' on P in $fringe$ with $COST[n'] = \hat{g}(n') = g(n')$.
Proof:
Let $P = \{s = n_0, n_1, \cdots, n_k = t\}$. If s is not in *closed*, let $n' = s$, then the lemma is obviously true.
If s is in *closed*, let Δ be the set of all n_i in *closed* and $\hat{g}(n_i) = g(n_i)$. $\Delta \neq \varnothing$ since $\hat{g}(s) = g(s) = 0$. Let n^* be the element in Δ with the highest index, and $n^* \neq t$. Let n' be an successor of n^*. By the update of the cost in *closed* (code line 9), we know $\hat{g}(n') \leq \hat{g}(n^*) + c_{n^*,n'}$, where $c_{n^*,n'}$ is the actual cost between two directly connected node n^* and n'. $g(n') = g(n^*) + c_{n^*,n'}$, for g is the actual cost of the optimal path. Therefore $\hat{g}(n') \leq \hat{g}(n') + c_{n^*,n'} = g(n*) + c_{n^*,n'} = g(n')$. By difination, we have $\hat{g}(n') \geq g(n')$. Therefore $\hat{g}(n') = g(n')$. Since n' is a successor of n^* and n^* is in *closed*, $n*$ must have been expanded and n' is in $fringe$.
End of proof (Lemma 1)
Corollary:
Suppose the heuristic function is admissible, and suppose A^* has not terminated. For any optimal path from s to any terminal t, there exists a node n' in $fringe$ on P with $\hat{f}(n') \leq f(s)$.
Proof:
Since the heuristic function is admissible, then $\hat{h}(n) \leq h(n)$ for all node n. By the lemma, there exists a node n' on P that $\hat{g}(n') = g(n')$. Therefore, by defination of $\hat{f}$

$$\begin{aligned}\hat{f}(n') &= \hat{g}(n') + \hat{h}(n') \\ &= g(n') + \hat{h}(n') \\ &\leq g(n') + h(n') \\ &= f(n')\end{aligned}$$

Since n' is on P and P is an optimal path, $f(n') = f(s)$.
Therefore $\hat{f}(n') \leq f(s)$.
End of proof (corollary)
Next, we will prove Question 8 by contradiction.
Proof: Suppose $A^* - CSCS$ terminate with a solution which is not optimal. That means A^* terminate at t with $\hat{f}(t) = \hat{g}(t) > f(s)$. But by the corollary to **Lemma 1**, there exist a node n' on the optimal path in $fringe$ with $\hat{f}(n') \leq f(s) < \hat{f}(t)$ just before the termination. By the A^* strategy, the n' should be removed from $fringe$ before t. This is contradict with that t is removed and n' not.
Therefore, $A^* - CSCS$ with an admissible heuristic will find an optimal solution.
End of proof (Q8)

Question assigned to the following page: 4.2

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

Yes.
(Please refer to the notations on the top of this section.)
If the consistent heuristic is also admissible, then by **Q8**, we know that $A^* - CSCS$ will find an optimal solution.
We will prove that the consistent heuristic is also admissible by mathematical induction.
Proof:
For any path P from s to t. $P = \{s = N_k, N_{k-1}, \cdots, N_0 = t\}$, we have $h(N) \leq \hat{g}(N)$
Base case:
When $k = 0, h(N_k) = h(N_0) = 0$. $\hat{g}(N_0) = 0$. This is obviously true since $0 \leq 0$.
Induction hypothesis:
For any $k \leq n, h(N_k) \leq \hat{g}(N_k)$.
Inductioin step:
When $k = n + 1$. Since the heuristic function is consistent, then

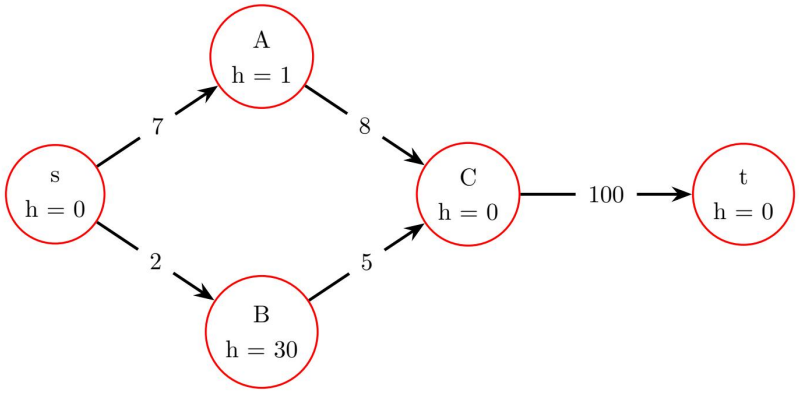
$$\begin{aligned} h(N_k) &\leq C(N_k, N_{k-1}) + h(N_{k-1}) \\ &\leq C(N_k, N_{k-1}) + C(N_{k-1}, N_{k-2}) \\ &\leq \cdots \\ &\leq C(N_k, N_{k-1}) + \cdots + C(N_1, N_0) \\ &= \sum_{i=0}^{(k-1)} C(N_i + 1, N_i) \\ &= \hat{g}(N_k) \end{aligned}$$

Therefore, the consistent heuristic is also admissible.
Hence, the *A*-CSCS* algorithm with a consistent heuristic will find an optimal solution.

Question assigned to the following page: 4.3

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

No.
(Please refer to the notations on the top of this section.)
In the following graph:



After first iteration (remove s), The node in the $fringe$ should be

$$A(g = 7, f = 8, prev = s), B(g = 2, f = 32, prev = s)$$

Therefore, in the next iteration, both *A* graph search* and *A*-CSCS* will expand A and the $fringe$ shoule be

$$C(g = 15, f = 15, prev = A), B(g = 2, f = 32, prev = s)$$

Next, both the algorithm will expand C, and the $fringe$ is

$$B(g = 2, f = 32, prev = s), t(g = 115, f = 115)$$

After that, both the alogrithm will expand B, and the $fringe$ is

$$C(g = 7, f = 7, prev = B), t(g = 115, f = 115)$$

At this point, *A* graph search* will not expand C because it's in *closed*, but *A*-CSCS* will expand C because $g = 7 < g' = 15$ in *closed*.
Therefore, *A*-CSCS* may expand more nodes than *A* graph search*

Question assigned to the following page: 4.4

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

Yes.
(Please refer to the notations on the top of this section and the proof of **Lemma 1** in Q8.)
We will prove this by showing that with a consistent heuristic, if a node n is in *closed*, then $\hat{g}(n) = g(n)$.
We will prove this by contradiction.
Lemma 2:
With a consistent heuristic, if a node n is in *closed*, then $\hat{g}(n) = g(n)$.
Proof:
Consider the graph G_s just before closing n, suppose that $COST[n] = \hat{g}(n) > g(n)$. There exist an optimal path P from s to t. By **Lemma 1 (in Q8)**, there exists a node n' on P in $fringe$ with $COST[n'] = \hat{g}(n') = g(n')$. If $n' = n$, then by contradiction, we have proved the lemma. Otherwise,

$$\begin{aligned} g(n) &= g(n') + h(n', n) \\ &= \hat{g}(n') + h(n', n) \\ \hat{g}(n) &> g(n) \\ \hat{g}(n) &> \hat{g}(n') + h(n', n) \\ \hat{g}(n) + \hat{h}(n) &> \hat{g}(n') + h(n', n) + \hat{h}(n) \end{aligned}$$

Since the heuristic is consistent

$$\hat{h}(n) + h(n', n) \geq \hat{h}(n')$$

Therefore

$$\begin{aligned} \hat{g}(n) + \hat{h}(n) &> \hat{g}(n') + \hat{h}(n') \\ \hat{f}(n) &> \hat{f}(n') \end{aligned}$$

This is contradict to the $A*$ strategy that $A*$ always select node with smaller $\hat{f}$ for expansion. The $A* - CSCS$ should expand n' rather than n at this stage.
Therefore, with a consistent heuristic, if a node n is in *closed*, then $\hat{g}(n) = g(n)$.
End of proof (Lemma 2)
By lemma 2, we know that if a node is in *closed*, no more attenption can give a cost less than the cost recorded in *closed*, since the recorded cost is already optimal.
This is means *A*-CSCS* with consistent heuristic will never expand a node again in *closed*. This is equal to *A* graph search* that will never expand a node in *closed*. Therefore, the *A*-CSCS* with a consistent heuristic will expand at most as many nodes as *A* graph search*.