CS101A(H) Homework 8

Due date: December 30, 2025, at 20:00 p.m.

Your grade for this homework will be computed as

$$\min\big(\text{Single Choices} + \text{Multiple Choices} + \text{Others},\ 30\ \text{pts}\big).$$

Notes: Unless otherwise stated, always express your final asymptotic bounds using $\boldsymbol{\Theta(\cdot)}$ notation for tight complexity, not just $O(\cdot)$ or $\Omega(\cdot)$. All answers must be written **inside the provided answer boxes** and **in English**. When submitting, match your solutions to the problems correctly in Gradescope. No late submission will be accepted. Failure to follow these rule may result in partial or full loss of credit.

Usage of AI. The use of AI tools for searching information or obtaining assistance on this homework is strictly prohibited. All solutions must be derived from your own understanding and effort. Submissions will be reviewed carefully, and any indication of reliance on AI-generated content will result in severe penalties.

Academic Integrity. This course upholds the highest standards of academic integrity. Any form of academic dishonesty, including plagiarism, unauthorized collaboration, or the use of prohibited resources such as AI tools, will be treated as a serious violation. Such actions not only undermine your own learning but also violate university policies.

Table 1: Fill answers here

question	1	2	3	4
answer				

1. (4 points) Single Choices

Each question has **exactly one** correct answer. Select the best answer.

(a) (2') We run the Floyd-Warshall algorithm on a simple graph without negative cycles with $3n$ vertices $v_1, \ldots, v_n, \ldots, v_{2n}, \ldots, v_{3n}$. Suppose all three loops (k, i, j) are iterated from 1 to $3n$. After running at least how many iterations of the out-most loop k, it is ensured to find the shortest path between v_{2n} and v_{3n}?

A. $2n - 1$

B. $2n$

C. $3n - 1$

D. $3n$

(b) (2') Which of the following statements is **TRUE**?

A. Bellman-Ford algorithm can find the shortest path for negative-weighted directed graphs, while the Dijkstra algorithm may fail.

B. The run time of the Bellman-Ford algorithm is $O(|V||E|)$, which is more time-consuming than the Dijkstra algorithm with heap implementation.

C. The A* graph search algorithm with an admissible heuristic will always return an optimal solution if it exists.

D. Floyd-Warshall algorithm can always give the correct shortest distance between any two vertices in directed graphs with negative weights.

2. (6 points) Multiple Choices

Each question has **one or more** correct answers. Select **all** correct answers. You will receive **no credit** if you select any wrong choice for a question.

(a) (3') Which of the following statements is/are **TRUE**?

A. Dijkstra algorithm can find the shortest path in any DAG.

B. If we use the Dijkstra algorithm, whether the graph is directed or undirected does not matter.

C. At each iteration of the Dijkstra algorithm, we pop the vertex with the shortest current distance to the start vertex, while in the Prim algorithm, we pop the vertex with the shortest distance to the current minimum spanning tree.

D. In directed graph $G = (V, E)$, if $(s, v_1, v_2, v_3, v_4, t)$, where $s, v_i, t \in V$, is the shortest path from s to t in G, then (v_1, v_2, v_3, v_4) must be the shortest path from v_1 to v_4 in G.

(b) (3') Which of the following is/are **TRUE**?

A. Dijkstra algorithm can be viewed as a special case of the A* Graph Search algorithm where the heuristic function from any vertex u to the terminal z is $h(u, z) = 0$.

B. For two graphs $G_1 = (V_1, E_1), G_2 = (V_2, E_2)$ such that $|V_1| = |V_2|$ but $|E_1| > |E_2|$, the Floyd-Warshall algorithm costs more time on G_1 than G_2.

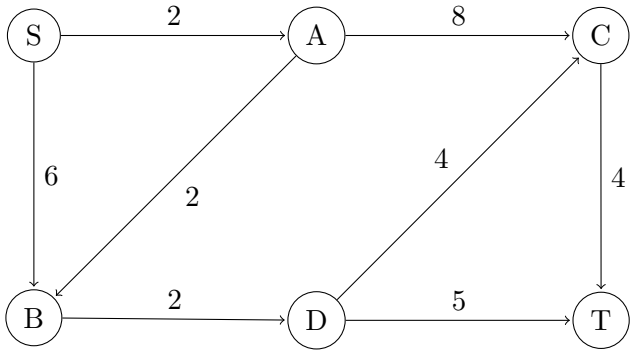
C. In A* graph search algorithm with a consistent heuristic function, if vertex u is marked visited before v, then $d(u)+h(u) \leq d(v)+h(v)$, where $d(u)$ is the distance from the start vertex to u.

D. In a DAG with probably negative edge weights, Bellman-Ford algorithm is guaranteed to find the shortest path from source s to any vertex if it can be reached from s.

3. (10 points) A star

Logan is doing an A* **graph search** from S to T on the graph below. Edges are labeled with weights.

The exploration order follows in their lexicographical order. (For example, $S \to X \to A$ would be expanded before $S \to X \to B$, and $S \to A \to Z$ would be expanded before $S \to B \to A$.



Node	S	A	B	C	D	T
Heuristic	11	9	12	4	5	0

Table 2: The initial heuristic value

Node	S	A	B	C	D	T
Heuristic	10	6	4	2	3	0

Table 3: The new heuristic value

(a) (3') He first uses the heuristic value in the Table 2 but finds it does not work well. That's because the given heuristic values are:

A. Admissible but not consistent

B. Consistent but not admissible

C. Neither admissible nor consistent

(b) Logan decides to modify the heuristic.

i. (3') He first gives out a new heuristic as in Table 3 above, please check whether the heuristic meets both admissibility and consistency (Write 'Yes' or 'No' for this).
If so, write down the path from S to T returned by A^* graph search (in the form of nodes, e.g. $S \to A \to C \to T$ should be written as $SACT$).
Otherwise, give out a contradiction that will lead admissibility or consistency to fail (in the form of 'Admissibility/Consistency' and its corresponding inequality.
e.g. If consistency fails on $A \to C$, write 'Consistency, $h(C) + 8 < h(A)$', here 8 corresponds to $w(A, C)$.).

Solution: No. Consistency, $h(A) + 2 < h(S)$

ii. (4') He finds that he can modify the heuristic of **only one node** to meet admissibility and consistency from the initial heuristic values in Table 2. Find the node and corresponding heuristic value for the chosen node. Justify your answer by stating the equalities obtained from admissibility and consistency.

Solution: Choose B with new value 7.
Admissibility: $h(B) \leq 7$. Consistency: $h(B) \geq 7.(A \to B \to D \to E)$.

4. (10 points) Negative Cycle Detection

The **inf(∞)** mentioned in this problem could be regarded as a pre-defined large enough constant. The graph mentioned in this problem is a simple directed graph. Please write down your codes in **standard C++**.

(a) Consider the following implementation of the Floyd-Warshall algorithm. Suppose W is the adjacency matrix of the graph, and assume that $W_{ij} = \infty$ where there is no edge between vertex i and vertex j, and assume $W_{ii} = 0$ for every vertex i. And other W_{ij} are the weights of the edge between vertex i and vertex j. The array 'graph' passed into the function is the adjacency matrix W.

```
bool DetectNegCycle_Floyd(const int graph[][V])
{
    int dist[V][V];

    for (int i = 0; i < V; ++i)
        for (int j = 0; j < V; ++j)
            dist[i][j] = graph[i][j];

    for (int k = 0; k < V; ++k)
        for (int i = 0; i < V; ++i)
            for (int j = 0; j < V; ++j)
                if (dist[i][j] > dist[i][k] + dist[k][j])
                        dist[i][j] = dist[i][k] + dist[k][j];

    ------------------------------------
    ------------------------------------
    ------------------------------------
    ------------------------------------

    return false;
}
```

i. (2') Consider the three loop lines in the Floyd-Warshall algorithm: lines 9, 10, and 11. Which pair(s) of these lines can be swapped without affecting the correctness of the algorithm? List all possible pairs of lines that can be swapped.

Solution: 10, 11.

ii. (4') Add some codes in the blank lines to detect whether there are negative cycles in the graph. (You may not use all blank lines, or you can add more lines.)

Solution:

```
for (int i = 0; i < V; ++i)__________
    if (dist[i][i] < 0)______________
        return true;_________________
____________________________________
```

(b) (4') Consider the following implementation of the Bellman-Ford algorithm. The 'edge' structure is used to represent the edges of the graph, with its elements u, v, and w denoting an edge from node u to node v and its corresponding weight w.

```
struct edge
{
    int u, v, w;
};

bool detectNegCycle_BellmanFord(const std::vector<edge>& Edge, int s)
{
    int dist[V];

    for (int i = 0; i < V; ++i)
        dist[i] = inf;
    dist[s] = 0;

    for(i = 1; i <= V - 1; ++i)
    {
        for(const auto& e : Edge)
            if (dist[e.v] > dist[e.u] + e.w)
                dist[e.v] = dist[e.u] + e.w;
    }

    ____________________________________
    ____________________________________
    ____________________________________
    ____________________________________

    return false;
}
```

Add some codes in the blank lines to detect whether there are negative cycles in the graph. (You may not use all blank lines, or you can add more lines.)

Solution:

```
for(const auto& e : Edge)____________
    if (dist[e.v] > dist[e.u] + e.w)_
        return true;_________________
____________________________________
```