CS 101 *Data Structure and Algorithm*

Fall 2021

Final Exam

INSTRUCTOR: Dengji Zhao, Yuyao Zhang, Zhihao Jiang

TIME: Jan 5th, 2022 8:00-10:00

INSTRUCTIONS

- You have 2 hours.
- You are not allowed to bring any papers, books or electronic devices including regular calculators.
- You are not allowed to discuss or share anything with others during the exam.
- You should write the answer of every problem in the dedicated box.
- You should write **your name and your student ID** as indicated on the top of each page of the exam sheet.
- You should write your answers **clearly**.

Name	
Student ID	
ShanghaiTech Email(Before @)	
Exam Room	
Seat Number	
All the work on this exam is my own. **(please copy this and sign)**	

THIS PAGE INTENTIONALLY LEFT BLANK

THIS PAGE INTENTIONALLY LEFT BLANK

THIS PAGE INTENTIONALLY LEFT BLANK

1. (20 points) True or False

You should write you answers of this section in the table below.

(a)	(b)	(c)	(d)	(e)	(f)	(g)	(h)	(i)	(j)
F	T	F	F	T	T	F	T	T	T

(a) **(2 pt)** If union by size optimization is used but path compression optimization is not used in disjoint sets, the worst case running time of a single **find** operation is $\Theta(N)$.False.

(b) **(2 pt)** A directed graph is strongly connected if and only if a DFS started from any vertex will visit every vertex in the graph.True.

(c) **(2 pt)** When finding Minimum Spanning Tree, Prim's algorithm and Kruskal's algorithm always compute the same result. F

(d) **(2 pt)** Given coin denominations $\{\$1, \$4, \$9, \$16, \$25\}$, greedy algorithm can find the fewest coins needed to make the change for arbitrary amount of money. F

(e) **(2 pt)** The topological sorting of a graph G is unique if and only if when running topological sort on G, the maximum number of vertices in the queue at the same time is 1. T

(f) **(2 pt)** 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. T

(g) **(2 pt)** The Floyd-Warshall algorithm can return the shortest path between all pairs of nodes in a connected graph with n nodes in $O(n^3)$, while this algorithm only needs $O(n^2)$ to find the shortest path between a given pair of nodes in the graph(in the worst case). F

(h) **(2 pt)** In any connected graph without negative cycle, A* tree-search algorithm with consistent Heuristics can always find the shortest path between two nodes. T

(i) **(2 pt)** If there exists a polynomial time algorithm to solve **Vertex Cover** problem, then any problem in NP can be solved by some polynomial time algorithm. True.

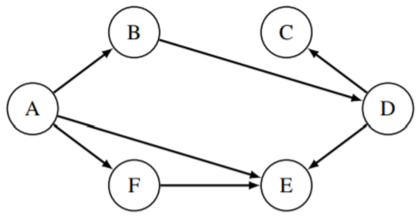
(j) **(2 pt)** Every problem in NP-complete can be solved in exponential time. True.

2. (15 points) Single Choice (Choose the ONLY ONE correct answer)

You should write you answers of this section in the table below.

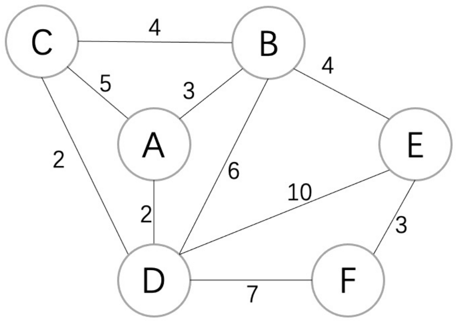
(a)	(b)	(c)	(d)	(e)
-	-	-	-	-

(a) (3 pt) Run preorder depth first search (DFS) on the following graph, starting from node A. List the order in which each node is visited. Whenever there is a choice of which node to visit next, visit nodes in alphabetical order.



(A) A, E, D, C, B, F
(B) A, B, D, E, C, F
(C) A, B, D, C, E, F
(D) A, B, D, C, F, E

(b) (3 pt) What is the last edge added to **minimum** spanning tree using Prim's algorithm. Suppose we start from vertex "A". B

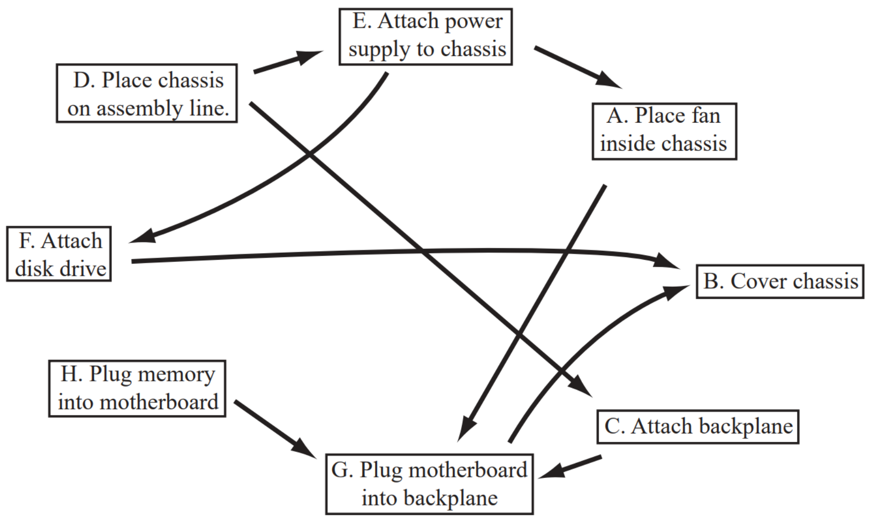


(A) (B, C)
(B) (E, F)
(C) (A, C)
(D) (D, E)

(c) (3 pt) Given an undirected graph $G = (V, E)$ with its minimum spanning tree $T \subset E$, which of the following statements is **false**? C

(A) Both Prim's algorithm and Dijkstra's algorithm can run in $O(|E| \log |V|)$ time complexity.
(B) At each iteration of Dijkstra's algorithm, we pop the vertex with the shortest current distance to the start vertex, while in Prims' algorithm, we pop the vertex with the shortest distance to the current minimum spining tree.
(C) The shortest path from any vertex s to any vertex t in all G is always the same as the shortest path from s to t only using edges in T.
(D) There might exist a vertex v in G s.t. the shortest paths from v to all other vertices in V are the same as the shortest paths only using edges in T.

(d) **(3 pt)** In the following graph, boxes represent tasks that must be performed in the assembly of a computer and arrows represent constraints that one task must be performed before another. Which of the following sequences to perform assembly tasks is **not feasible** i.e. violating some constraint? D


E. Attach power supply to chassis
D. Place chassis on assembly line.
A. Place fan inside chassis
F. Attach disk drive
B. Cover chassis
H. Plug memory into motherboard
C. Attach backplane
G. Plug motherboard into backplane

(A) D E A F H C G B
(B) H D C E A G F B
(C) D H C E A F G B
(D) H D E C G A F B

(e) **(3 pt)** We run Floyd-Warshall algorithm on a graph with n vertices $v_1, v_2, ...v_{n/2}, ...v_n$ (n is even). Suppose all three loops (k, i, j) are iterated from 1 to n. After running at least ______ iterations of the out-most loop k, it is ensured to find the shortest path between $v_{n/2}$ and v_n. Note: You can refer to Question 3(c) for the pseudo-code of the Floyd-Warshall algorithm. C

(A) $\frac{n}{2} - 1$
(B) $\frac{n}{2}$
(C) $n - 1$
(D) n

Name: ID:

3. (25 points) Multiple Choices (There will be one or multiple correct answers. Choose ALL the correct answers. You will get half credits if you only choose a subset of the correct answers.)

Note that you should write you answers of this section in the table below.

(a)	(b)	(c)	(d)	(e)
BC	ABCD	AC	BCD	AB

(a) (5 pt) $G = (V, E)$ is undirected and connected. Which of the following statements is/are correct? Do not assume the edge weights are distinct unless specifically stated. BC

(A) A minimum spanning tree of G has $|V|$ edges.

(B) Let e be any edge of minimum weight in G. Then e must be in some of the minimum spanning trees of G.

(C) If G has a cycle with unique minimum weight edge e, then e must be in every minimum spanning tree.

(D) Prim's algorithm fails to find a minimum spanning tree when there are negative-weighted edges in G.

(b) (5 pt) Which of the following statements about shortest path algorithms is/are true? ABCD

(A) If we modify the outer loop of Dijkstra's algorithm to execute $|V| - 1$ iterations instead of $|V|$ iterations (i.e. only pop $|V| - 1$ times from the heap), the algorithm can still find the shortest path on a positive-weighted graph.

(B) If we modify the outer loop of Bellman-Ford algorithm to execute $|V|$ iterations instead of $|V| - 1$ iterations (i.e. apply update rule to each edge for $|V|$ times), the algorithm can still find the shortest path on a positive-weighted graph.

(C) We can modify Floyd-Warshall algorithm to detect whether there exists a negative cycle or not in a directed graph.

(D) We can modify Bellman-Ford algorithm to detect whether there exists a negative cycle or not in a directed graph.

(c) (5 pt) Which of the following statements about the Floyd-Warshall's algorithm is/are **correct**? B

Algorithm 1 Floyd-Warshall's algorithm

```
1:  procedure FLOYD-WARSHALL($V, E$)
2:      let dist be a $|V| \times |V|$ array of minimum distances initialized to $\infty$ (infinity)
3:      for each edge$(u, v) \in E$ do
4:          dist$[u][v] \leftarrow$ w$(u, v)$ // The weight of the edge (u, v)
5:      end for
6:      for each vertex $v \in V$ do
7:          dist$[v][v] \leftarrow 0$
8:      end for
9:      for $k$ from 1 to $|V|$ do
10:         for $i$ from 1 to $|V|$ do
11:             for $j$ from 1 to $|V|$ do
12:                 if dist$[i][j]$ > dist$[i][k]$ + dist$[k][j]$ then
13:                     dist$[i][j] \leftarrow$ dist$[i][k]$ + dist$[k][j]$
14:                 end if
15:             end for
16:         end for
17:     end for
18:     return dist
19: end procedure
```

(A) After swapping the codes in line 9 and line 10, the algorithm still computes the correct output.

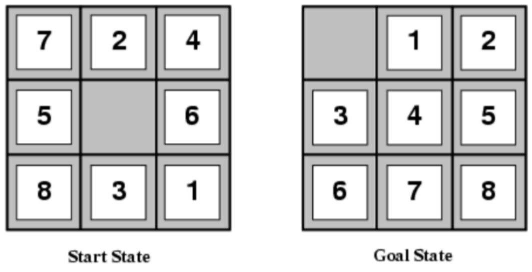
(B) After swapping the codes in line 10 and line 11, the algorithm still computes the correct output.

(C) After swapping the codes in line 9 and line 11, the algorithm still computes the correct output.

(D) None of the above.

(d) **(5 pt)** To solve the N Puzzle problems with the A* search, which of following heuristics is/are **admissible**?
BCD

(A) $h =$ if the state is the goal state, it is 1; otherwise, it is 0
(B) $h =$ number of misplaced tiles
(C) $h =$ the sum of the minimum number of moves required to put each tile in its correct location
(D) $h =$ actual minimum number of moves required to put all tile in its correct location



(e) **(5 pt)** Choose one condition for each of the two statements such that they are true, respectively.
1. "Minimum Spanning Tree Problem is in NP" is ______.
2. "Minimum Spanning Tree Problem is in NP-complete." is ______.
Note: The order of your answer matters. e.g. AB is different from BA

(A) Always True.
(B) True if and only if $P = NP$.
(C) True if and only if $P \neq NP$.
(D) Always False.

AB

4. (10 points) Shortest Path in DAG

Consider a modified version of Bellman-Ford's algorithm, called **DAG-SP**, as a new approach to find the shortest path in a directed acyclic graph G from single source vertex s by the following steps:

1. **Topological Sort:** Find a topological sorting of the given directed graph.
2. **Traversal:** Iterate vertices in topologically sorted order.
3. **Relaxation:** For each vertex, relax its outward edges and update the current distance to each of its neighbors from the source vertex, like what we do in other shortest path algorithms.

The psuedocode (with two incomplete lines) is given below.

```
function DAG-SP($G$, $s$)
    $toposorting \leftarrow$ TopologicalSort($G$)
    for each vertex $v$ in $G$ do
        $dis[v] \leftarrow \infty$
    end for
    ________(1)________
    for each vertex $u$ in $toposorting$ order do
        for each adjacent vertex $v$ of $u$ in $G$ do
            $w \leftarrow$ weight of edge $(u, v)$
            $dis[v] \leftarrow$ Min($dis[v]$, ________(2)________)
        end for
    end for
end function
```

(a) (3 pt) Fill in the blank in line 6 and line 10 of the psuedocode.

Note: Min(a, b) *returns the smaller one of* a *and* b.

(1) $dis[s] \leftarrow 0$
(2) $dis[u] + w$

(b) (3 pt) If the input $G = (V, E)$ is guaranteed to be a DAG, briefly analyse the **total runtime complexity** of DAG-SP algorithm and **compare** its runtime with the original Bellman-Ford algorithm when applying to DAG. Assume we use adjacency list implementation to store the graph.

Hint: To analyse the time complexity of step 3, consider how many relaxation operations are performed at most in total.

Time complexity:
1. Topological sort takes $O(|V| + |E|)$ time.
2. There are $|V|$ vertices to be intialized and visited.
3. There are $|E|$ edges to be relaxed and each relaxation takes $O(1)$ time.
Therefore, the total time for DAG-SP algorithm is $O(|V| + |E|)$.

$O(|V| + |E|)$ is more efficient than $O(|V||E|)$

(c) (4 pt) Please pair the **most appropriate** shortest path algorithm to each of the following problems by considering the corresponding graph. One option can be paired only once.

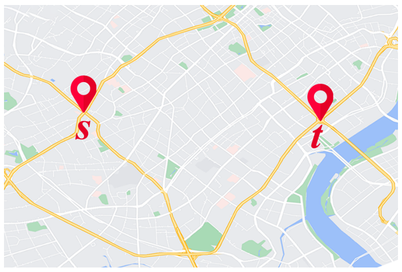
(A) DAG-SP (B) Dijkstra (C) Bellman-Ford (D) A* Search

i. (1 pt) D Find the shortest path from s to t in G_1, which is based on a real-world map, with Euclidean distance from each vertex to t is known.

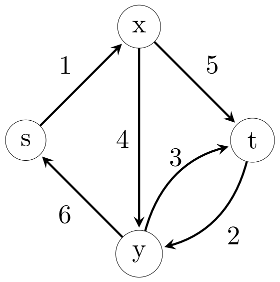
ii. (1 pt) B Find the shortest path from s to t in G_2.

iii. (1 pt) A Find the shortest path from s to t in G_3.

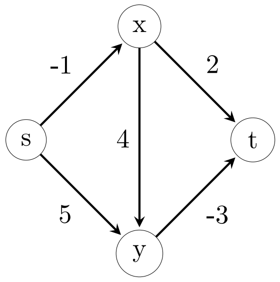
iv. (1 pt) C Among all paths from s to t in G_4 that contain at most k edges, find the shortest one.



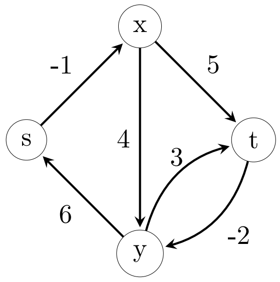
G_1



G_2



G_3



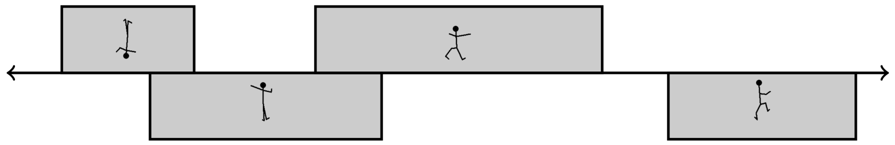
G_4

5. (10 points) Placing Bus Stations

A city is interested in designing a new bus line that runs on a straight road. However, they are unsure of where to place the bus stops. The city surveyed bus users along the route for (a) where they live and (b) how far they would be willing to walk to a bus station. Design a greedy algorithm that minimizes the number of bus stops required to cover all surveyed bus users. Give an exchange argument to prove that your algorithm is correct and analyse the run time complexity.

Assume the survey included n users with user i living at x_i and willing to walk a distance d_i(in either direction). You may assume that all elementary arithmetic operations take unit time.

For example, the following picture shows 4 users and the ranges they are willing to walk for a bus. 3 stations are needed at least.



Solution:
For each user, there is a strict range $[l_i = x_i d_i, r_i = x_i + d_i]$ of the x axis in which a stop must be put to satisfy user i. Now, if we sort all users by r in increasing order, and begin looking at the first user. Where should we choose z_1 for the first stop? In order to cover the first user, we need $z_1 \leq r_1$. However, placing it at any $z_1 < r_1$ wastes resource, as it covers no more users than placing it at $z_1 = r_1$.

Now that the first stop is placed, it covers some users. Where should we place the next router? We can simply repeat the above process and find among the uncovered users the one with the smallest r coordinate. The process goes on until we cover all the users.

Exchange argument: consider an optimal solution $y_1, y_2, \cdots, y_k$ which maximizes j such that the first j stops are the same as in greedy. i.e. $y_i = z_i$ for $1 \leq i \leq j$. Let r_k be the right side of the next (i.e., leftmost) range not covered by $y_1, \cdots, y_j$. By our greedy algorithm $z_{j+1} = r_k$. If $y_{j+1} > z_{j+1}$, then the k^{th} user isn't covered in the optimal solution (a contradiction). So $y_{j+1} \leq z_{j+1}$. Moving y_{j+1} right so it equals z_{j+1} cannot uncover some previously covered interval, since it would have to move right of some interval not covered by $y_1, \cdots, y_j$ (contradicting the definition of r_k). Therefore, we can replace y_{j+1} with z_{j+1}. By induction, we can replace each y_i with z_i without uncovering any interval, so $z_1, \cdots, z_k$ is an optimal solution.

The runtime is $O(n \log n)$ as it takes $O(n)$ time to compute the l and r values, $O(n \log n)$ to sort with respect to r, and then $O(n)$ to perform a linear sweep assigning stops.

6. (10 points) Taking Lectures

You have n lectures numbered $1, 2, \cdots, n$ in the time order. Attending lecture i will spend you t_i time. You may not be able to take all the lectures, because after you attend lecture i and spend t_i time on it, you will feel very tired and cannot continue to take any of the following y_i lectures after lecture i. For example, if you take the i-th lecture with $y_i = 2$, you will miss lecture $i + 1$ and lecture $i + 2$.

However, as a hardworking ShanghaiTech student, you want to spend more time on lectures, i.e. you want to maximize the total time you can spend on those lectures.

Suppose that you are given the t_i and y_i values for all n lectures as input. Design a **dynamic programming** algorithm to find the maximum total time you can spend on all the lectures.

Note: Try to give the most time-efficient algorithm that you can think of, and you may only receive partial credit if your algorithm is not efficient enough.

(a) (2 pt) Define your subproblems.

dp[i] as the maximum total time that can be spent considering lecture i to n.

(b) (4 pt) Describe and justify the Bellman Equation for computing the solution to subproblems.

$dp[i] = max(t[i] + dp[i + y[i] + 1], dp[i + 1])$

(c) (2 pt) Write down the base cases of your recurrence function and how can we reach the final answer using the function you have defined.

Base case: dp[i] = 0, if i > n
Output: dp[1]

(d) (2 pt) What is the time and space complexity of computing your solution?

Both O(n).

7. (10 points) Find a reduction

Given an array $A = [a_1, a_2, ..., a_n]$ of non-negative integers, consider the following problems:

1.Partition: Determine whether there is a subset $P \subseteq [n]([n] = \{1, 2, ..., n\})$ such that $\sum_{i \in P} a_i = \sum_{j \in [n] \setminus P} a_j$. For example, given $A = [2, 4, 6, 8]$, then $P = \{1, 4\}$ is a partition of A since $a_1 + a_4 = a_2 + a_3 = 10$.

2.Subset Sum: Given some integer K, determine whether there is a subset $P \subseteq [n]$ such that $\sum_{i \in P} a_i = K$. For example, given $A = [1, 3, 5, 7]$ and $K = 6$, then $P = \{1, 3\}$ gives a subset sum of $a_1 + a_3 = 6$.

Suppose we have proven that Subset Sum problem is in NP-complete, prove that Partition problem is also in NP-complete.

1. Partition problem is in NP.
Simply iterate through P and $[n]\setminus P$ to verify that whether $\sum_{i \in P} a_i = \sum_{j \in [n] \setminus P} a_j$. This process has a time complexity of $\Theta(n)$

2. We can find a linear time reduction from Subset Problem, which is proved NP-complete, to Partition Problem.
Suppose we are given some A with target sum K. Let S be the sum of all elements in A.
If $S - 2K \geq 0$, generate a new set $A_0 = A \cup \{S - 2K\}$. If A_0 can be partitioned, then there is a subset of A that sums to K. We know that the two sets in our partition must each sum to $S - K$ since the sum of all elements will be $2S - 2K$. One of these sets, must contain the element $S - 2K$. Thus the remaining elements in this set sum to K.
If $S - 2K \leq 0$, generate a new set $A_0 = A \cup \{2K - S\}$. If A_0 can be partitioned, then there is a subset of A that sums to K. We know that the two sets in our partition must each sum to t since the sum of all elements will be $2K$. The set that does not contain $2K - S$ will be our solution to subset sum.
This reduction also clearly operates in $O(n)$, as we simply need to generate a new set with a single additional element (whose value is determined by summing all the elements of A).