CS101 Algorithms and Data Structures
Fall 2023
Homework 9

Due date: December 18, 2023, at 23:59

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 problems correctly.
6. No late submission will be accepted.
7. Violations to any of the above may result in zero points.

1. (6 points) Multiple Choices

Each question has **one or more** correct answer(s). Select all the correct answer(s). For each question, you will get 0 points if you select one or more wrong answers, but you will get 1 point if you select a non-empty subset of the correct answers.

Write your answers in the following table.

(a)	(b)	(c)

(a) (2') Which of the following statements about topological sort is/are true?

A. The implementation of topological sort requires $O(|V|)$ extra space.

B. Any sub-graph of a DAG has a topological sorting.

C. A DAG can have more than one topological sorting.

D. Since we have to scan all vertices to find those with zero in-degree in each iteration, the run time of topological sort is $\Omega(|V|^2)$.

(b) (2') Which of the following statements about topological sort and critical path is/are true?

A. Create a graph from a rooted tree by assigning arbitrary directions to the edges. The graph is guaranteed to have a valid topological order.

B. A critical path in a DAG is a path from the source to the sink with the minimum total weights.

C. A DAG with all different weighted edges has one unique critical path.

D. Let $c(G)$ be the run time of finding a critical path and $t(G)$ be the run time of finding an arbitrary topological sort in a DAG G, then $c(G) = \Theta(t(G))$.

(c) (2') For the coin changing problem, which of the following statements is/are true? Denote

- $1 = c_1 < c_2 < \cdots < c_k$: the denominations of coins;
- $g^*(n)$: the optimal solution, i.e., the minimum number of coins needed to make change for n;
- $g(n)$: the greedy solution, written as

$$g(n) = \begin{cases} 0, & n = 0 \\ 1 + g(n - \max\limits_{c_i \le n} c_i), & n \ge 1 \end{cases}$$

A. If $\forall i \in [2, n]$, $\dfrac{c_i}{c_{i-1}}$ is an integer, then $\forall n, g^*(n) = g(n)$.

B. If $\forall i \in [3, n], c_i = c_{i-1} + c_{i-2}$, then $\forall n, g^*(n) = g(n)$.

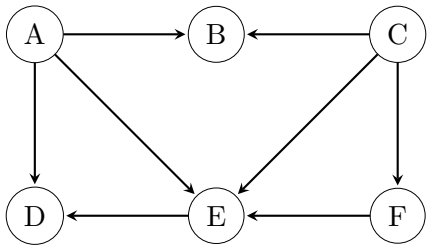
C. If $\forall i, 2c_i \le c_{i+1}$, then $\forall n, g^*(n) = g(n)$.

D. If $\exists i, 2c_i > c_{i+1}$, then $\exists n, g^*(n) \ne g(n)$.

2. (8 points) Topological Sort

Given the following DAG, run topological sort with a queue. Write down the vertex you select and update the in-degree `ind[i]` of all vertices in each iteration.

Note: When pushing several vertices into the queue at the same time, push them alphabetically. You are NOT required to show your queue at each step.



	vertex	ind[A]	ind[B]	ind[C]	ind[D]	ind[E]	ind[F]
initial	/	0	2	0	2	3	1
iteration 1	A		1	0	1	2	1
iteration 2	C		0		1	1	0
iteration 3	B				1	1	0
iteration 4	F				1	0	
iteration 5	E				0		
iteration 6	D						

(a) (3') Fill in the table above.

(b) (2') What is the topological order that you obtain?

Solution: A, C, B, F, E, D

(c) (3') How many different topological orders starting with A does this graph have? Write them down.

Solution:
2
A, C, B, F, E, D
A, C, F, B, E, D

3. (9 points) Array Section

Given an upper bound M and a sequence of positive integers $A = \langle a_1, \cdots, a_n \rangle$ where $\forall i \in [1, n], a_i \leq M$, we want to divide it into several consecutive sections so that the sum of each section is less than or equal to the upper bound M, and the number of sections is minimized.

For example, if $M = 6$ and $A = \langle 4, 2, 4, 5, 1 \rangle$, the minimum number of sections is 3, and there are two ways to divide the sequence A into 3 sections: $\langle 4 \rangle, \langle 2, 4 \rangle, \langle 5, 1 \rangle$ and $\langle 4, 2 \rangle, \langle 4 \rangle, \langle 5, 1 \rangle$.

Design a greedy algorithm to find the minimum number of sections in $\Theta(n)$ time, and prove its correctness.

(a) (2') Describe your algorithm in **pseudocode**.

Solution:

```
function MinimizeSections(A, M)
    n ← length of A
    sections ← 1
    currentSum ← 0
    for i ← 1 to n do
        if currentSum + A[i] > M then
            sections ← sections + 1
            currentSum ← A[i]
        else
            currentSum ← currentSum + A[i]
    return sections
```

(b) (2') Analyse the time complexity based on your **pseudocode**.

Solution: The time complexity of the algorithm is $\Theta(n)$ since we only need to iterate through the sequence once.

(c) (1') How to define the sub-problem $g(i)$ in your algorithm?

Solution: Define the sub-problem $g(i)$ as the minimum number of sections required for the subsequence $A[1...i]$, where $1 \leq i \leq n$.

(d) (2') How do you solve $g(i)$ by calling $g(i-1)$ recursively?

Solution:

```
if i = 1 then
    g(i) ← 1
else
    g(i) ← g(i − 1)
    if sum of elements in the last section of g(i − 1) + A[i] > M then
        g(i) ← g(i) + 1
```

(e) (2') Prove the correctness of solving $g(i)$ by calling $g(i-1)$.

Solution: To prove the correctness of the greedy algorithm solving $g(i)$ by using $g(i-1)$, we argue that if there were a better solution with fewer sections, it would imply that we could add an element to a section or merge sections without exceeding M, contradicting the assumption that our solution wasn't optimal. Since the greedy algorithm makes the best local choice at each step, and since these choices are independent, the overall solution is optimal. Thus, by induction, our algorithm finds the minimum number of sections for sequence A.

4. (13 points) Minimum Refueling

A vehicle is driving from city A to city B on a highway. The distance between A and B is d kilometers. The vehicle departs with f_0 units of fuel. Each unit of fuel makes the vehicle travel one kilometer. There are n gas stations along the way. Station i, denoted as S_i, is situated p_i kilometers away from city A. If the vehicle chooses to refuel at S_i, f_i units would be added to the fuel tank whose capacity is unlimited.

Your job is to design a **greedy** algorithm that returns **the minimum number of refueling** to make sure the vehicle reaches the destination.

For example, if $d = 100, f_0 = 20, (f_1, f_2, f_3, f_4) = (30, 60, 10, 20), (p_1, p_2, p_3, p_4) = (10, 20, 30, 60)$, the algorithm should return 2. There are two ways of refueling 2 times. The first one is:

- Start from city A with 20 units of fuel.
- Drive to station 1 with 10 units of fuel left, and refuel to 40 units.
- Drive to station 2 with 30 units of fuel left, and refuel to 90 units.
- Drive to city B with 10 units of fuel left.

And the second one is:

- Start from city A with 20 units of fuel.
- Drive to station 2 with 0 units of fuel left, and refuel to 60 units.
- Drive to station 4 with 20 units of fuel left, and refuel to 40 units.
- Drive to city B with 0 units of fuel left.

Note that:

1. $0 < p_1 < p_2 < \cdots < p_n < d$, and $\forall i \in [0, n], f_i \geq 1$.
2. If the vehicle cannot reach the target, return `-1`.
3. The time complexity of your algorithm should be $O(n \log n)$. You should use a max heap, and simply write `heap.push(var)`, `var = heap.pop()` in your **pseudocode**.

(a) (2') Define the sub-problem $g(i)$ as the indices of an n-choose-i permutation of stations $P = (P_1, P_2, \cdots, P_i) \in \text{Per}(n, i)$ satisfying the following conditions:

$$g(i) = \{P \in \text{Per}(n, i) : \forall k \in [1, n], (P_1, P_2, \cdots, P_k) \in \arg\max_{Q \in C_k} \sum_{j=1}^{i} f_{Q_j}\}$$

$$C_k = \left\{ Q \in \text{Per}(n, k) : \forall l \in [1, k], f_0 + \sum_{j=1}^{l-1} f_{Q_j} \geq p_{Q_l} \right\}$$

Then what is $g(1)$ and $g(2)$ in the example above?

Solution: $g(1) = \{1\}$, $g(2) = \{2, 1\}$.

(b) (2') How do you find one of the solutions in $g(i+1)$ by using one of the solutions in $g(i)$? And when does $g(i+1) = \emptyset$?

Solution: To find a solution in $g(i+1)$ using a solution in $g(i)$:

1. Let $P = (P_1, ..., P_i)$ be a solution in $g(i)$.
2. Calculate available fuel: fuel $= f_0 + \sum_{j=1}^{i} f_{P_j}$ − distance to P_i.
3. Identify stations reachable from P_i with this fuel.
4. If no stations are reachable, set $g(i+1) = \emptyset$.
5. If reachable, select S_x, the station with the most fuel.
6. Update P for $g(i+1)$ as $(P_1, ..., P_i, x)$.
7. Repeat until the destination is reached or $g(i+1) = \emptyset$ if unreachable.

$g(i+1) = \emptyset$ signifies that the no any destination can be reached with the available fuel.

(c) (2') Prove the correctness of solving $g(i+1)$ by calling $g(i)$ when $g(i+1) \neq \emptyset$.

Solution:
Theorem:
Extending an optimal solution for $g(i)$ with a greedy choice yields an optimal solution for $g(i+1)$ when $g(i+1) \neq \emptyset$.

Optimal Substructure:
An optimal solution to $g(i+1)$ contains within it an optimal solution to $g(i)$. If $P = (P_1, \ldots, P_i)$ is an optimal solution for $g(i)$, then any extension of P to include one additional station must preserve optimality for $g(i+1)$, provided that the extension is done optimally.

Greedy-Choice Property:
The greedy choice for extending an optimal solution P to solve $g(i+1)$ is selecting the station S_g that is reachable from P_i with the available fuel and offers the maximum additional fuel. This choice is locally optimal and is made without considering the entire set of remaining stations.

Proof:
Assume for contradiction that there is a better station S_b to extend P than the greedy choice S_g. If S_b were indeed better, then P followed by S_b would lead to a more optimal solution than P followed by S_g. This would contradict the optimality of P for $g(i)$ since we could have reached S_b from P_i and had a better outcome.

Since no such S_b can exist, the greedy choice S_g must be part of an optimal solution for $g(i+1)$. Therefore, extending P with S_g maintains optimality, and the solution for $g(i+1)$ constructed in this manner is optimal.

(d) (2') Define the problem $h(i)$ as the maximal distance that the vehicle can drive from city A:

$$h(i) = f_0 + \max_{Q \in E_i \cap C_i} \sum_{j=1}^{i} f_{Q_j}$$

$$E_i = \{Q \in \text{Per}(n, i) : Q_1 < Q_2 < \cdots < Q_i\}$$

$$C_i = \left\{ Q \in \text{Per}(n, i) : \forall l \in [1, i], f_0 + \sum_{j=1}^{l-1} f_{Q_j} \geq p_{Q_l} \right\}$$

Prove that $\forall P \in g(i), f_0 + \sum_{j=1}^{i} f_{P_j} = h(i)$.

Solution:

Let $P \in g(i)$ be an optimal solution for $g(i)$. By definition, P is a permutation of i distinct stations such that the vehicle can reach station P_j from the starting point, or the previous station, with the available fuel.

Since P is an optimal solution for $g(i)$, the vehicle can reach the i-th station, which implies that P satisfies the fuel constraints defined in C_i. Therefore, $P \in C_i$.

Now, consider the definition of $h(i)$, which is the maximal distance that can be driven from city A using any permutation in $E_i \cap C_i$. The optimality of P for $g(i)$ implies that it maximizes the distance traveled with the available fuel, which means that the sum of the fuel amounts at the stations in P, plus the initial fuel f_0, must be at least as great as for any other permutation in $E_i \cap C_i$.

Therefore, we have that $f_0 + \sum_{j=1}^{i} f_{P_j} \geq h(i)$. However, since $h(i)$ is the maximum value by definition, it must also hold that $f_0 + \sum_{j=1}^{i} f_{P_j} \leq h(i)$. Combining both inequalities, we conclude that $f_0 + \sum_{j=1}^{i} f_{P_j} = h(i)$ for all $P \in g(i)$, which completes the proof.

(e) (3') What is the relationship between $h(i)$ and the minimum number of refueling? And under what condition does the vehicle cannot reach the target? Based on your analysis above, describe your algorithm in **pseudocode**.

Solution:

Require: f_0 (initial fuel), d (distance to target), *stations* (array of fuel stations)
Ensure: *minStops* (minimum number of refueling stops to reach the target)

```
currentFuel ← f_0
currentPosition ← 0
minStops ← 0
while currentPosition < d do
    maxReachable ← currentPosition + currentFuel
    if maxReachable ≥ d then
        return minStops
    nextStation ← null
    for each station in stations not yet visited do
        if station.position ≤ maxReachable and station.position > currentPosition then
            if nextStation is null or station.fuel > nextStation.fuel then
                nextStation ← station
    if nextStation is null then
        return "Cannot reach the target"
    currentFuel ← currentFuel − (nextStation.position − currentPosition) + nextStation.fuel
    currentPosition ← nextStation.position
    minStops ← minStops + 1
return minStops
```

(f) (2') Analyse the time complexity based on your **pseudocode**.

Solution: The time complexity of the algorithm can be analyzed as follows:

1. Initializing variables takes constant time, $O(1)$.
2. The while loop iterates at most n times, which is the number of stations.
3. Within the while loop, finding the next best station to refuel involves iterating over the remaining stations. In the worst case, this results in a series of $n + (n-1) + (n-2) + \ldots + 1$, which has a time complexity of $O(n^2)$.
4. Updating the current position and fuel takes constant time for each iteration of the while loop, $O(1)$.

Therefore, the overall time complexity of the algorithm is dominated by the nested iteration for selecting the next station, resulting in $O(n^2)$.