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)
ABC	AD	ABCD

(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

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

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 $\forall n, g^*(n) = g(n)$.

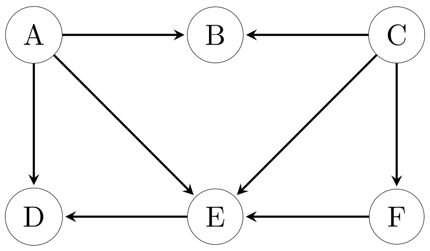
C. If $\forall i, 2c_i \le c_{i+1}$, then $\forall n, g^*(n) = g(n)$.

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

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?

ACBFED

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

4.

ACBFED, ACFBED, ACFEBD, ACFEDB

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**.

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

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

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

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

(a)

```
function minSections(M, A)
    $n \leftarrow len(A)$
    $cnt \leftarrow 1$
    $sum \leftarrow A[0]$
    for $i \leftarrow 1$ to $n$ do
        if $sum + A[i] > M$ then
            $cnt \leftarrow cnt + 1$
            $sum \leftarrow A[i]$
        else
            $sum \leftarrow sum + A[i]$
        end if
    end for
    return $cnt$
end function
```

(b) The time complexity of the algorithm is $\Theta(n)$ because it only involves a single pass through the array.

(c) The sub-problem $g(i)$ is the one with the same M and $A_i = A[0 : i-1]$.

(d) If $A[i-1]$ can be added to the last section of $g(i-1)$, then $g(i) = g(i-1)$.
Otherwise, $g(i) = g(i-1) + 1$.
The base case is $g(1) = 1$.

(e) The algorithm is correct because at each step, it makes the optimal choice of either adding the current element to the existing section or starting a new section. This local optimality leads to a globally optimal solution because once a section is closed, its sum cannot be altered without increasing the total number of sections. Thus, the greedy choice property is satisfied, and the algorithm is correct.

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?

(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 $g(i+1) = \emptyset$?

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

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

(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**.

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

(a) $g(1)$ is the station with the maximum f_i which can be reached after departing from A.
$g(2)$ is the station in $g(1)$ and another distinct station with maximum f_i which can be reached after refueling in $g(1)$.

(b) A solution in $g(i+1)$ can be found by adding a station to the solutions in $g(i)$, where the added station is a distinct station with the maximum f_i which can be reached after refueling in every station in $g(i)$.
$g(i+1)$ will be empty, if no more station can be reached after $g(i)$.

(c) The correctness of solving $g(i+1)$ by calling $g(i)$ can be proven by induction. Refueling in every station in $g(i)$ can result in the maximum distance after i refueling. Then adding such a station to the solution in $g(i)$ can then result in the maximum distance after $i+1$ refueling, which is the solution of $g(i+1)$.

(d) Since $h(i)$ is the maximal distance that the vehicle can drive from city A, so $\forall P \in g(i), f_0 + \sum_{j=1}^{i} f_{P_j} = h(i)$ which means the maximum distance that can be reached is the initial fuel plus the fuel obtained at the stations in the solution.

(e) The relationship between $h(i)$ and the minimum number of refueling is that the minimum i which makes $h(i) \geq d$ is the minimum number of refueling.
The vehicle cannot reach the target if $g(k) = \emptyset$, but $h(i) < d$ for any $i = 1, 2, \ldots, k-1$.
The pseudocode for the algorithm is as following

```
function MINREFUEL($d, f_0, stations$)
    $heap \leftarrow new\ MaxHeap()$ \\heap ranked by fuel
    $totalFuel \leftarrow f_0$
    $curr \leftarrow 0$
    for $i \leftarrow 1$ to $stations.size()$ do
        if $p_i <= totalFuel$ then
            $heap.push(f_i)$
            $curr \leftarrow curr + 1$
        else
            $break$
        end if
    end for
    for $i \leftarrow 0$ to $stations.size()$ do
        if $totalFuel >= d$ then
            return $i$
        end if
        if $heap.empty()$ then
            return $-1$
        end if
        $totalFuel \leftarrow totalFuel + heap.pop()$
        for $j \leftarrow curr$ to $stations.size()$ do
            if $p_j <= totalFuel$ then
                $heap.push(f_j)$
                $curr \leftarrow curr + 1$
            else
                $break$
            end if
        end for
    end for
end function
```

(f) The time complexity of the algorithm is $O(nlogn)$ due to the operations on the max heap (push and pop), which take $O(logn)$ time each, and the fact that each station is visited only once.