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	A

(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 $\Omega(|V|^2)$.

Solution:

D. The run time is $\Theta(|V| + |E|)$. It can be $o(|V|^2)$ when $|E| = o(|V|^2)$.

(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 $c(G) = \Theta(t(G))$.

Solution:

A. This graph is also a DAG.

B. Maximum, not minimum.

C. Counterexample: $A \xrightarrow{1} B \xrightarrow{4} D, A \xrightarrow{2} C \xrightarrow{3} D$

D. Both of them are $\Theta(|V|)$.

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

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

Solution:

A. Let $c_t = \max_{c_i \leq n} c_i$ be the largest denomination that can be used, and we will prove at least one c_t coin is used in the optimal solution to make change for n.

Denote x_i as the number of coins of c_i used in the optimal solution, then $n = \sum_{i=1}^{t} x_i c_i$, and we will prove that $x_t > 0$.

Note that $\forall i \in [1, t-1], x_i \leq \frac{c_{i+1}}{c_i} - 1$ in the optimal solution, because if at least $\frac{c_{i+1}}{c_i}$ coins of c_i are used, then we can replace them with one c_{i+1} coin to get a better solution.

$$\begin{aligned}
x_i &\leq \frac{c_{i+1}}{c_i} - 1 \\
x_i c_i &\leq c_{i+1} - c_i \\
x_t c_t &= n - \sum_{i=1}^{t-1} x_i c_i \\
&\geq n - \sum_{i=1}^{t-1} (c_{i+1} - c_i) \\
&= n - c_t + c_1 \\
&= n - c_t + 1 \\
x_t c_t + x_t &\geq n + 1 \\
x_t + 1 &\geq \frac{n+1}{c_t} > \frac{n}{c_t} \geq 1 \\
x_t &> 0
\end{aligned}$$

B. Counterexample: $c_1 = 1, c_2 = 3, c_3 = 4, n = 6, g^*(n) = 2$ using coins $\{3, 3\}$ but $g(n) = 3$ using coins $\{4, 1, 1\}$.

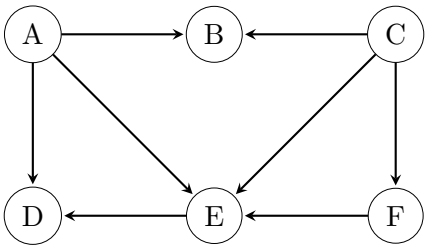
C. Counterexample: $c_1 = 1, c_2 = 4, c_3 = 9, n = 12, g^*(n) = 3$ using coins $\{4, 4, 4\}$ but $g(n) = 4$ using coins $\{9, 1, 1, 1\}$.

D. Counterexample: $c_1 = 1, c_2 = 2, c_3 = 3$, then $\forall n, g^*(n) = g(n)$.

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	0	1	0	1	2	1
iteration 2	C	0	0	0	1	1	0
iteration 3	B	0	0	0	1	1	0
iteration 4	F	0	0	0	1	0	0
iteration 5	E	0	0	0	0	0	0
iteration 6	D	0	0	0	0	0	0

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

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

Solution: ACBFED

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

Solution: 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 $g(i-1)$. You are required to give a strict proof. Prove by contradiction (suppose there is a solution which is better than $g(i)$), or prove by exchanging arguments (any solution can be exchanged to the optimal solution).

Solution: Denote a way of division as $\langle a_{s_1}, \cdots, a_{e_1} \rangle, \cdots, \langle a_{s_k}, \cdots, a_{e_k} \rangle$, where k is the number of sections and s_i, e_i means the start index and the end index of the i-th section respectively.

Method 1:

(a)
```
def ArraySection(A):
    g = 0
    for i = [1,...,n]:
        s = 0
        while s + A[g + 1] <= M:
            s = s + A[g + 1]
            g = g + 1
            if g == n:
                return i
```

(b) `g` starts with 0 and ends with n, and increases by 1 each time in the while loop. So the while loop iterates for exactly n times before the algorithm terminates, and the time complexity is $n \times \Theta(1) = \Theta(n)$.

`i` starts with 1 and ends when `g` reaches n. So the for loop iterates for not more than n times before the algorithm terminates, and the time complexity (excluding the time of running while loops) is $O(n)$.

So the time complexity is $\Theta(n)$.

(c) Sub-problem $g(i)$ means the largest end index of the rightmost section among the optimal divisions of i sections.

(d) The problem $g(i)$ can be solved greedily by calling the sub-problem $g(i-1)$ once:

$$g(i) = \begin{cases} 0, & i = 0 \\ \arg\max_{e_i \in E_i} e_i, & i \geq 1, \text{ where the valid set is } E_i = \{e_i : \sum_{j=g(i-1)+1}^{e_i} a_j \leq M\} \end{cases}$$

(e) **Lemma**: $g(i)$ is optimal if $g(i-1)$ is optimal, which implies that the greedy solution is optimal by induction.

Proof by contradiction: Suppose there is another way of division such that the end index of the rightmost section e'_i is larger than $g(i)$, i.e. $e'_i \geq g(i)+1$.

Let s'_i be the start index of the rightmost section.

Then $s'_i > g(i-1)+1$, because if $s'_i \leq g(i-1)+1$, then $\sum_{j=s'_i}^{e'_i} a_j \geq \sum_{j=g(i-1)+1}^{g(i)+1} a_j > M$, which violates the upper bound.

So the end index of the second rightmost section $e'_{i-1} = s'_i - 1$ must be greater than $g(i-1)$, which means that $g(i-1)$ is not optimal, giving a contradiction.

Method 2:

(a)
```
def ArraySection(A):
    s = A[1]
    g = 1
    for i = [2,...,n]:
        s = s + A[i]
        if s > M:
            s = A[i]
            g = g + 1
    return g
```

(b) The for loop iterates for $\Theta(n)$ times, so the time complexity is $\Theta(n) \times \Theta(1) = \Theta(n)$.

(c) Sub-problem $g(i)$ means minimizing the number of sections for $\langle a_1, \cdots, a_i \rangle$, and then maximizing the start indices of all the sections.

Note: $g(i)$ is represented as the set of start indices $\{s_1, \cdots, s_k\}$ in the following description.

(d) The problem $g(i)$ can be solved greedily by calling the sub-problem $g(i-1)$ once:

$$g(i) = \begin{cases} \{1\}, & i = 1 \\ g(i-1), & i > 1, \sum_{j=g(i-1)_k}^{i} \leq M \\ g(i-1) \cup \{i\}, & i > 1, \sum_{j=g(i-1)_k}^{i} > M \end{cases}$$

(e) **Lemma**: $g(i)$ is optimal if $g(i-1)$ is optimal, which implies that the greedy solution is optimal by induction.

Method 1: Proof by contradiction: Suppose there is a way of division $g'(i) = \{s'_1, \cdots, s'_{k'}\}$ which is better than $g(i) = \{s_1, \cdots, s_k\}$, which means either $k' < k$ or $k' = k, \exists t, s'_t > s_t$.

- If $k' = k, \exists t, s'_t > s_t$, let t be the smallest index such that $s'_t > s_t$, then $g(i-1)$ is not optimal because we can change s_t to s'_t while not violate the upper bound, which gives a contradiction.
- If $k' < k$, it is impossible that $\exists t, s'_t > s_t$ for the same reason. However, $\forall t \in [1, k'], s'_t = s_t$ is also impossible because
  - when $\sum_{j=g(i-1)_k}^{i} \leq M$, if it's possible, then $g(i-1)$ is not optimal because it's also a valid solution for the sub-problem, but $k' < k$, which contradicts.
  - when $\sum_{j=g(i-1)_k}^{i} > M$, it violates the upper bound.

Method 2: Proof by exchanging arguments: For any other way of division $g'(i) = \{s'_1, \cdots, s'_{k'}\}$ which is different from $g(i) = \{s_1, \cdots, s_k\}$, we can exchange some arguments in $g'(i)$ and finally it will not be better than $g(i)$.

Let t be the first index that $s'_1 = s_1, \cdots, s'_{t-1} = s_{t-1}, s'_t < s_t$. Then we can remove $s'_{t+1}, \cdots, s'_{t+l} \leq s_t$ from $g'(i)$ and insert s_t into $g'(i)$.

After exchanging, either the number of sections becomes smaller, or the start index becomes larger, while the upper bound won't be violated, because the section from s'_{t+l} to $s'_{t+l+1} - 1$ becomes the section from s_t to $s'_{t+l+1} - 1$ which is shorter.

And finally we can get $g'(i) = g(i)$, which means $g(i)$ is optimal. It's impossible to finally become shorter that $s'_1 = s_1, \cdots, s'_{k'} = s_{k'}, k' < k$, because

- when $\sum_{j=g(i-1)_k}^{i} \leq M$, if it's possible, then $g(i-1)$ is not optimal because it's also a valid solution for the sub-problem, but $k' < k$, which contradicts.
- when $\sum_{j=g(i-1)_k}^{i} > M$, it violates the upper bound.

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:

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

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

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

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

(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: $\forall P = (P_1, P_2, \cdots, P_i) \in g(i)$, add the t-th station so that $(P_1, P_2, \cdots, P_i, t) \in g(i+1)$, where t satisfies:

$$f_t = \max_{u \in D \setminus \bigcup_{j=1}^{i} \{P_j\}} f_u$$

$$D = \{u : f_0 + \sum_{j=1}^{i} f_{P_j} \geq p_u\}$$

(In natural language) One of the optimal solution can be found by adding a gas station to one of the sub-problem's optimal solution, where

1. The gas station should be accessible. (D)
2. Each gas station can only refuel once. $(D \setminus \bigcup_{j=1}^{i} \{P_j\})$
3. Always choose the station which provides most fuel. $(f_t = \max f_u)$

And $g(i+1) = \emptyset$ when $g(i) = \emptyset$ or $D \setminus \bigcup_{j=1}^{i} \{P_j\} = \emptyset$.

(c) (2') Prove the correctness of solving $g(i+1)$ by calling $g(i)$ when $g(i+1) \neq \emptyset$. You are required to give a strict proof. Prove by contradiction (suppose there is a solution which has a larger sum of fuel than $g(i+1)$), or prove by exchanging arguments (any solution can be exchanged to the optimal solution).

Solution:

Method 1: Proof by contradiction: Assume there is a better solution $Q = (Q_1, \ldots, Q_{i+1}) \neq (P_1, P_2, \cdots, P_i, t)$, then $\sum_{j=1}^{i+1} f_{Q_j} > f_t + \sum_{j=1}^{i} f_{P_j}$.

We have $f_{Q_{i+1}} > f_t$ because $\sum_{j=1}^{i} f_{Q_j} = \sum_{j=1}^{i} f_{P_j}$ since both $(P_1, \ldots, P_i)$ and $(Q_1, \ldots, Q_i)$ belong to $\arg\max_{Q \in C_i} \sum_{j=1}^{i} f_{Q_j}$.

If $Q_{i+1} \notin \bigcup_{j=1}^{i} \{P_j\}$, then $f_t \neq \max_{u \in D \setminus \bigcup_{j=1}^{i} \{P_j\}} f_u$ because $Q_{i+1} \in D \setminus \bigcup_{j=1}^{i} \{P_j\}$ and $f_{Q_{i+1}} > f_t$, which gives a contradiction.

If $Q_{i+1} \in \bigcup_{j=1}^{i} \{P_j\}$, let $Q_{i+1} = P_k$, then we can swap Q_k, Q_{i+1} to get another better solution because $f_{P_k} = f_{Q_k}$. Continue swapping until $Q_{i+1} \notin \bigcup_{j=1}^{i} \{P_j\}$ which gives a contradiction, and there are at most i steps of swapping.

Method 2: Proof by exchanging arguments: For any valid solution $Q = (Q_1, \ldots, Q_{i+1})$ such that $\forall k \in [1, i+1], (Q_1, Q_2, \cdots, Q_k) \in C_k$, we can exchange arguments to get $(P_1, P_2, \cdots, P_i, P_{i+1}) \in g(i+1)$ for every $(P_1, P_2, \cdots, P_i) \in g(i)$, while the sum of fuel will not go smaller.

Let t be the first index that $P_1 = Q_1, ..., P_{t-1} = Q_{t-1}, P_t \neq Q_t$.

- If $P_t \notin \bigcup_{j=t+1}^{i+1} \{Q_j\}$, then change Q_t to P_t.
- If $P_t \in \bigcup_{j=t+1}^{i+1} \{Q_j\}$, let $P_t = Q_k$ then swap Q_t, Q_k.

For $t \leq i$ we have $f_{P_t} \geq f_{Q_t}$ because $g(i)$ is optimal.

For $t = i+1$ we also have $f_{P_t} \geq f_{Q_t}$ because $f_{P_{i+1}} = \max_{u \in D \setminus \bigcup_{j=1}^{i} \{P_j\}} f_u$.

Whether changed or swapped, the sum of fuel will not go smaller, and the conditions $C_t, C_{t+1}, \cdots, C_{i+1}$ will also not be violated.

And finally we can get $Q = P$, which means P is optimal for $g(i+1)$.

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

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

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

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

Prove that $\forall P \in g(i), f_0 + \sum_{j=1}^{i} f_{P_j} = h(i)$. You should prove that $\max_{P \in C_i} \sum_{j=1}^{i} f_{P_j} = \max_{Q \in E_i \cap C_i} \sum_{j=1}^{i} f_{Q_j}$, i.e. the max sum of fuel among the two different sets are equal.

Solution:

Step1: $\max_{P \in C_i} \sum_{j=1}^{i} f_{P_j} \geq \max_{Q \in E_i \cap C_i} \sum_{j=1}^{i} f_{Q_j}$.

This is obvious because $C_i \supseteq E_i \cap C_i$.

Step2: $\max_{P \in C_i} \sum_{j=1}^{i} f_{P_j} \leq \max_{Q \in E_i \cap C_i} \sum_{j=1}^{i} f_{Q_j}$.

Lemma: $\forall P \in C_i, \exists Q \in E_i \cap C_i$ such that $\sum_{j=1}^{i} f_{P_i} = \sum_{j=1}^{i} f_{Q_i}$.

Proof: We can sort P to get Q. Use insertion sort or bubble sort, so for any adjacent inversion $P_k > P_{k+1}$, you can swap them so the permutation becomes

$(\cdots, P_{k+1}, P_k, \cdots)$, and the condition C_i won't be violated, because $f_0 + \sum_{j=1}^{k-1} f_{P_j} \geq p_{P_k} > p_{P_{k+1}}$ and $f_0 + f_{P_{k+1}} + \sum_{j=1}^{k-1} f_{P_j} > f_0 + \sum_{j=1}^{k-1} f_{P_j} \geq p_{P_k}$. Finally, the sequence is sorted, which satisfies the condition E_i.

In conclusion, $\max_{P \in C_i} \sum_{j=1}^{i} f_{P_i} = \max_{Q \in E_i \cap C_i} \sum_{j=1}^{i} f_{Q_i}$, which means $\forall P \in g(i), f_0 + \sum_{j=1}^{i} f_{P_j} = h(i)$.

(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**.

Solution: The minimum number of refueling is $\min_{h(i) \geq d} i$.

When $h\left(\max_{g(i) \neq \emptyset} i\right) < d$, the vehicle cannot reach the target.

```
def MinimumRefueling1(d,f[0,...,n],p[1,...,n]):
    h = f[0]
    j = 0
    for i = [1,...,n]:
        while j < n and h >= p[j+1]:
            j = j + 1
            heap.push(f[j])
        if heap.empty():
            return -1
        h = h + heap.pop()
        if h >= d:
            return i
    return -1

def MinimumRefueling2(d,f[0,...,n],p[1,...,n]):
    h = f[0]
    i = 0
    for j = [1,...,n]:
        while h < p[j]:
            if heap.empty():
                return -1
            h = h + heap.pop()
            i = i + 1
        heap.push(f[j])
    while h < d:
        if heap.empty():
            return -1
```

```
        h = h + heap.pop()
        i = i + 1
    return i
```

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

Solution: Brief calculation: we push some stations into the heap, and pop some of them from the heap. Every heap operation requires $O(\log n)$ time, other operations are $\Theta(1)$, and the number of operation is $\Theta(n)$.

So the overall time complexity is $O(n \log n)$.