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	AD

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

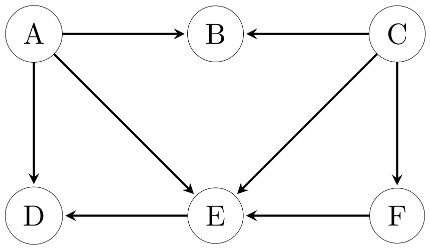
C. If $\forall i, 2c_i \le c_{i+1}$, then $\forall n, g^*(n) = g(n)$.

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.



$$\begin{tikzcd}
A \arrow[r] \arrow[d] \arrow[dr] & B & C \arrow[l] \arrow[dl] \arrow[d] \\
D & E \arrow[l] & F \arrow[l]
\end{tikzcd}$$

	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?

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

Solution:

1. ACBFED
2. 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 $g(i-1)$.

Solution:

1. **Pseudocode:**
   Define two iterators it which starts at the first element of the array. Answer is the number of sections we need. Length is the length of array A. Current is the sum of elements in the array which is less than or equal to M.

```
function FIND_MINIMUM_SECTIONS(array A, M)
    it ← 0
    answer ← 0
    length ← A.size()
    current ← 0
    while it != length do
        if current + A[it] < M then
            current += A[it]
            it++
        else if current + A[it] = M then
            current ← 0
            answer += 1
            it++
        else
            current ← A[it]
            answer += 1
            it++
        end if
    end while
end function
```

Solution:

1. **Analysis:** Outside the loop the time complexity is $\theta(n)$.
   Inside the loop the whole time complexity is $\frac{5}{12}\theta(2) + \frac{1}{6}\theta(3) + \frac{5}{12}\theta(3)$, which is $\theta(1)$.
   Thus the whole time complexity is $\theta(n) + n \cdot \theta(1) = \theta(n)$.

2. The sub-problem g(i) is the minimun number of section in A from the first element to the ith element.

3. If after adding A[i] to the current in the pseudocode, current is larger than or equal to M, then g(i) = g(i-1)+1; otherwise, g(i) = g(i-1).

4. Knowing that g(i-1) is the best way of dividing, and we assume that there exist a better solution for g(i), called h(i). Then there must be a inequation that the number of sections in h(i) is less than or equal to those in g(i). We know that g(i) equals to g(i-1) or g(i-1)+1, thus h(i) is sometimes g(i-1) when g(i) is g(i-1)+1. When g(i) equals to g(i-1)+1, it means the ith element can not be put into the last section, while in h(i) the ith can. Since the order will not change, there must be some elements in the last section of g(i) not belonging to the last section in h(i). For these elements, they can be regarded as new elements which should be added into g(i-2). Repeat the operations above until g(1)(h(1) should add new elements to g(1)), then we can know that h(1) is obviously larger than or equal to g(1), contradicting to the condition that h(i) is less than or equal to those in g(i). Thus solving g(i) by calling g(i-1) 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:

$$h(i) = f_0 + \max_{Q \in E_i \cap C_i} \sum_{j=1}^{i} f_{Q_j}$$
$$E_i = \{Q \in \mathrm{Per}(n,i) : Q_1 < Q_2 < \cdots < Q_i\}$$
$$C_i = \left\{ Q \in \mathrm{Per}(n,i) : \forall l \in [1,i], f_0 + \sum_{j=1}^{l-1} f_{Q_j} \geq p_{Q_l} \right\}$$

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

(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**.

Solution:

1. g(1) = (2) and g(2) = (2,1)
2. For every set in g(i), choose the gas station that the vehicle haven't reached with max f_j. If the vehicle cannot reach that gas station P_j, which in other hand P_j does not belong to C_k, we find the second max f_j. If the vehicle can reach, we push it back to this set. Repeat these operations, if in all sets all gas stations don't fit the requirement above, g(i+1) = $\varnothing$.
3. For every solution from g(i) to g(i+1), we choose a gas station which can be reached and it's fuel is the max, thus it's obvious that if g(i+1) $\neq \varnothing$ solving g(i+1) by g(i) is correct.
4. Knowing that each unit of fuel drives the vehicle one km, h(i) needs to choose the gas station that can reach with max f_j. g(i) choose the gas station that can reach with max f_j. It's obvious that g(i) and h(i) does the same operations. h(i) contains f_0 while g(i) doesn't. Thus the equation is correct.
5. If h(i) is larger than or equal to the distance between two cities, then return i. If for all i, h(i) is less than the distance, then the vehicle cannot reach.

Pseudocode:
S is an array of the distances between the city A and gas stations. F is an array of the amount of fuel the gas stations can give. V is a vector which restore S and F in pair, like S_i, F_i and it is sorted according to f_j in descending order.

```
function MINIMUM_FUELING(V)
    answer ← ($f_0$,0)
    goal ← (distance,0)
    maxheap.push(goal)
    for int i=0;i<V.size();++i do
        if answer[0] > V[i][0] then
            answer[0] += V[i][0]
            answer[1]++
        end if
        maxheap.push(answer)
        if maxheap[0][1] != 0 then return
        end if
    end for
end function
```

6. In my pseudocode, the construction of max heap and the sort of vector is both $\theta(n\log n)$, and the loop will run n times with $\theta(\log n)$ operation. Thus the whole time complexity is $\theta(n\log n)$.