CS101 Algorithms and Data Structures
Fall 2022
Homework 11

Due date: 23:59, December 11th, 2022

1. Please write your solutions in English.
2. Submit your solutions to Gradescope.
3. If you want to submit a handwritten version, scan it clearly.
4. When submitting, match your solutions to the problems correctly.
5. No late submission will be accepted.
6. Violations to any of the above may result in zero credits.
7. You are recommended to finish the algorithm design part of this homework with LaTeX.
8. Please check your Account Settings for Gradescope when submitting! Set your FULL name to your Chinese name and your 10-digit STUDENT ID correctly.

1. (0 points) Maximum Subarray Problem

Given an array $A = \langle A_1, \ldots, A_n \rangle$ of n elements, please design a dynamic programming algorithm to find a contiguous subarray whose sum is maximum.

Notes: (MUST READ!)

1. Problems in this homework require you to design **dynamic programming** algorithms. When grading these problems, we will put more emphasis on how you define your subproblems, whether your Bellman equation is correct and correctness of your complexity analysis.
2. **Define your subproblems clearly.** Your definition should include the variables you choose for each subproblem and a brief description of your subproblem in terms of the chosen variables.
3. Your **Bellman equation** should be a recurrence relation whose **base case** is well-defined. You can briefly **explain each term in the equation** if necessary, which might improve the readability of your solution and help TAs grade it.
4. Analyze the **runtime complexity** of your algorithm in terms of $\Theta(\cdot)$ notation.
5. You only need to calculate the optimal value in each problem of this homework, and you don't have to back-track to find the optimal solution.

(a) (0') Define your subproblem for this question.

Solution: $\mathrm{OPT}(i)$ = the maximum sum of subarrays of A ending with A_i.

(b) (0') Give your Bellman equation to solve the subproblems.

Solution:

$$\mathrm{OPT}(i) = \begin{cases} A_1 & \text{if } i = 1 \\ \max\{A_i,\ A_i + \mathrm{OPT}(i-1)\} & \text{if } i > 1 \end{cases}$$

Explanation: (NOT Required)

- The 1st term in max: only take A_i
- The 2nd term in max: take A_i together with the best subarray ending with A_{i-1}

(c) (0') What is the answer to this question in terms of the subproblems?

Solution:

$$\max_{i \in \{1,2,\ldots,n\}} \mathrm{OPT}(i)$$

(d) (0') What is the runtime complexity of your algorithm?

Solution: $\Theta(n)$

2. (20 points) Having a Buffet

You plan to have a buffet at Aloft hotel on the weekend. There are n different kinds of food provided by the hotel, and you can eat at most W grams of food for the buffet. The i-th kind of food is worth v_i yuan and weighs w_i ($w_i, W \in \mathbb{Z}^+$) grams per plate. You are very frugal, so you will not waste any food and eat up each plate of food. You have paid T yuan for the buffet ticket, and you wonder whether you can get your money's worth or not.

(a) **Warm Up**

In order to enjoy as many kinds of foods as possible, you decide to taste at most one plate of each kind of food. Please design a dynamic programming algorithm to find out **whether you can get your money's worth** or not for the buffet. That is to say, it is possible for the total value of the food you eat to exceed the price you paid for the buffet ticket.

i. (2') Define your subproblem for this question.

Solution:
let dp[i][j] be the maximum value we can get after considering first i kinds of food, with the weight limit of j.

ii. (4') Give your Bellman equation to solve the subproblems.

Solution:
initially: $dp[i][j] = 0, \forall i, j$

$$dp[i][j] = \begin{cases} dp[i-1][j] & \text{if } j < w_i \\ \max\{dp[i-1][j],\ dp[i-1][j-w_i] + v_i\} & \text{if } j \geq w_i \end{cases}$$

Explanation:

- The 1st term in max: do not take the $i-th$ kind of food.
- The 2nd term in max: take the $i-th$ kind of food, so the total weight comes from $j - w_i$.

iii. (2') What is the answer to this question in terms of the subproblems?

Solution:
if $dp[n][W] \geq T$, then it can get the money worth.
otherwise, it cannot.

iv. (1') What is the runtime complexity of your algorithm?

Solution:
$\Theta(nW)$

(b) **Greedy Eater**

This time, for each kind of food, you decide to **eat as many plates** as you want. Please design a greedy algorithm **trying** to maximize the total value of the food you can eat.

i. (2') Describe your greedy strategy where you should take both the value and weight of each kind of food into account. This is an open question and your algorithm does not have to give the optimal solution.

Solution:
consider the cost performance ratio (i.e. $\frac{v_i}{w_i}$) of each food, and sort the food with their cost performance ratio in the descending order.
and consider the food from the biggest cost performance ratio to the lowest, for each kind of food, take as many as we can.

ii. (2') Provide a counterexample to show your greedy algorithm fails in finding the optimal solution to this question.

Solution:
let $W = 10$, $v_1 = 10, w_1 = 6, v_2 = 6, w_2 = 5$
the first kind of food has the cost performance ratio $\frac{10}{6} = 1.67$, while the second kind of food has the cost performance ratio $\frac{6}{5} = 1.2$
so with the greedy, we will take one first food, and get the value of 10.
but actually, we can take two second kind of food, then the value is $2*6 = 12 > 10$.
so the greedy is not correct.

(c) **Time for Dynamic Programming**

Again, you eat as many plates of each kind of food as you want. Please design a dynamic programming algorithm to find out whether you can get your money's worth or not for the buffet.

i. (3') Notice that you can achieve this goal by using the subproblem you defined in (a) and **only modifying your Bellman equation** in (a). Now, please first come up with a naïve $O(W\sum_{i=1}^{n}\frac{W}{w_i})$ dynamic programming algorithm. Give your modified Bellman equation.
Hint: Try to enumerate the number of plates you eat of each kind of food.

Solution:
initially: $dp[i][j] = 0, \forall i, j$

$$dp[i][j] = \begin{cases} dp[i-1][j] & \text{if } j < k \cdot w_i \\ \max\{dp[i-1][j],\ dp[i-1][j-k\cdot w_i] + k\cdot v_i\} & \text{if } j \geq k \cdot w_i \end{cases}$$

where $0 < k \cdot w_i \leq j$

Explanation:

- The 1st term in max: do not take the $i-th$ kind of food.
- The 2nd term in max: take the $i-th$ kind of food with k times, so the total weight comes from $j - k \cdot w_i$.

ii. (4') Try to solve this question by **only modifying your Bellman equation** in (a) **without changing its runtime complexity**. Give your modified Bellman equation.

Solution:
initially: $dp[i][j] = 0, \forall i, j$

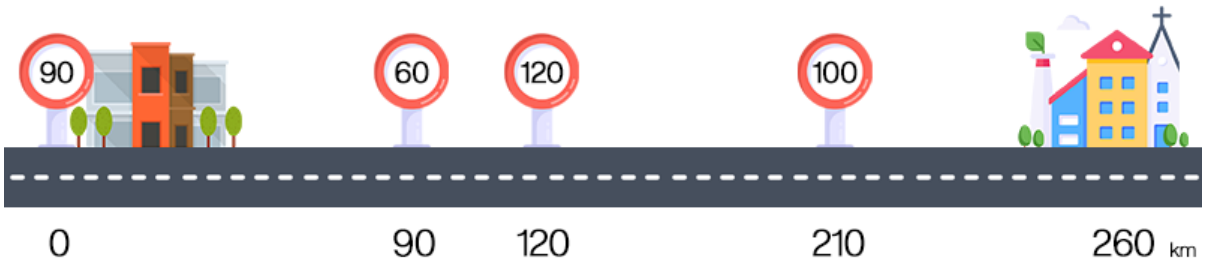
$$dp[i][j] = \begin{cases} dp[i-1][j] & \text{if } j < w_i \\ \max\{dp[i-1][j],\ dp[i][j-w_i] + v_i\} & \text{if } j \geq w_i \end{cases}$$

Explanation:

- The 1st term in max: do not take the $i-th$ kind of food.
- The 2nd term in max: take the $i-th$ kind of food, so the total weight comes from $j-w_i$, and the food can be taken of many times as we want, so we can get it from the $i-th$ level, instead of the $(i-1)-th$.

3. (10 points) Highway Speed-Limit Sign Schedule

There is a highway of length L kilometers from Alpha Town to Beta City and there are n traffic signs along the highway. Each speed-limit sign i at x_i ($0 = x_1 < x_2 < \cdots < x_n < L$) kilometers from the origin has a speed limit v_i, indicating that you can travel at a speed of at most v_i kilometers per hour until you reach the next sign or arrive at your destination. There is also a sign at Alpha Town ($x_1 = 0$) which sets the initial speed limit.



For example, assume we have $n = 4$ and $L = 260$ as the figure shown above. Then, it will take at least $\frac{90-0}{90} + \frac{120-90}{60} + \frac{210-120}{120} + \frac{260-210}{100} = 2.75$ hours to travel from Alpha Town to Beta City under these speed limits.

In order to improve the traffic efficiency, the highway administration department decides to remove some speed-limit signs along the highway. However, due to the traffic restrictions of Alpha Town, the first sign at the origin $x_1 = 0$ cannot be removed.

Please come up with a dynamic programming algorithm to **minimize the travel time** from Alpha Town to Beta City by removing no more than K speed-limit signs.

(a) (2') Define your subproblem for this question.

Solution:
let `dp[i][j]` be the minimal time we will take from $x_1 = 0$ to the x_i, and the i-th sign is remained, j speed-limit signs in the first i signs are removed.

(b) (5') Give your Bellman equation to solve the subproblems.

Solution:
let $x_{n+1} = L$
$1 \le i \le n+1; j < i; j \le K$

$dp[i][j] = min\{dp[k][j-(i-k-1)] + \frac{x_i - x_k}{v_k}\}, k = (i-j-1), \cdots, i-1$

Explanation:

- since the i-th sign is not removed, so $j < i$.
- k means that from the $k+1$-th sign to the $i-1$-th sign are all removed, and the k-th sign and the i-th sign are remained, so there are totally $i-1-k$ signs are removed, so it should transform from $dp[k][j-(i-k-1)]$, and during the period of x_k to x_i, the limit speed if v_k.
- since we removed $i-k-1$ signs this time, so $i-k-1 \le j$, i.e.$k \ge i-j-1$

(c) (2') What is the answer to this question in terms of the subproblems?

Solution:
$min\{dp[n+1][i], i = 0, 1, \cdots, K\}$

(d) (1') What is the runtime complexity of your algorithm?

Solution:
$\Theta(nk^2)$

4. (10 points) Pairwise DNA Sequence Alignment

Given two DNA sequences: a query sequence $Q = \langle Q_1, \ldots, Q_m \rangle$ of m nucleotides and a subject sequence $S = \langle S_1, \ldots, S_n \rangle$ of n nucleotides. There are 4 types of nucleotides, namely Adenine (A), Cytosine (C), Guanine (G) and Thymine (T). We provide a scoring matrix for nucleotides at the same index of these two sequences:

	A	C	G	T
A	1	-5	-1	-5
C	-5	1	-5	-1
G	-1	-5	1	-5
T	-5	-1	-5	1

where $\text{Score}(X, X)$ on the diagonal represents the score of a successful match for nucleotide X at the same index of both sequences and $\text{Score}(X, Y)$ indicates the penalty for mismatching nucleotide X by nucleotide Y.

For example, assume we have $m = n$ here and there are two sequences $Q = \langle TGGTG \rangle$ and $S = \langle ATCGT \rangle$. Then the alignment score for these two sequences is

$$\begin{aligned}
\text{Score}(Q, S) &= \sum_{i}^{n} \text{Score}(Q_i, S_i) \\
&= \text{Score}(T, A) + \text{Score}(G, T) + \text{Score}(G, C) + \text{Score}(T, G) + \text{Score}(G, T) \\
&= (-5) + (-5) + (-5) + (-5) + (-5) \\
&= -25
\end{aligned}$$

However, if $m \neq n$, we must insert several gaps '$-$' to make two sequences the same length. What's more, in order to align two sequences for higher score, we can also insert arbitrary number of gaps '$-$' to each sequence at arbitrary index. However, adding one gap will result in a gap penalty $\text{Penalty}(X, -) = \text{Penalty}(-, Y) = -2$.

Assume after inserting 4 gaps, we obtain $Q' = \langle -T - GGTG \rangle$ and $S' = \langle ATCG - T- \rangle$. Then the recomputed alignment score is:

$$\begin{aligned}
\text{Score}(Q', S') &= \text{Penalty}(-, A) + \text{Score}(T, T) + \text{Penalty}(-, C) + \text{Score}(G, G) \\
&\quad + \text{Penalty}(G, -) + \text{Score}(T, T) + \text{Penalty}(G, -) \\
&= (-2) + 1 + (-2) + 1 + (-2) + 1 + (-2) \\
&= -5
\end{aligned}$$

Notice that Q' and S' should share the same length after inserting gaps.

Given the scoring matrix and gap penalty, please come up with a dynamic programming algorithm to **maximize the pairwise alignment score** of a pair of DNA sequences by inserting gaps to these two sequences.

(a) (2') Define your subproblem for this question.

Solution: let $dp[i][j]$ means that the maximum value we can get when matching the first i nucleotides of Q, and the first j nucleotides of S.
suppose that the two sequence's index start from 1.

(b) (5') Give your Bellman equation to solve the subproblems.

Solution:
$dp[0][0] = 0$

$dp[i][j] = max\{$
$dp[i-1][j-1] + Score(Q[i], S[j]),$
$dp[i-1][j] + Penalty(Q[i], -),$
$dp[i][j-1] + Penalty(-, S[j])$
$\}(i > 0, j > 0)$

Explanation:

- The 1st term in max: do not use "$-$", just take the last untaken nucleotide, so Q_i and S_j are matched.
- The 2nd term in max: we will use the "$-$" on S, so Q_i will match "$-$".
- The 3rd term in max: we will use the "$-$" on Q, so S_j will match "$-$".

(c) (2') What is the answer to this question in terms of the subproblems?

Solution:
$dp[n][m]$
n is the length of Q, while m is the length of S.

(d) (1') What is the runtime complexity of your algorithm?

Solution:
$\Theta(nm)$
n is the length of Q, while m is the length of S.