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 breifly **explain each term in the equation** if necessary, which might improve the readability of your solution and help TAs grade it.
4. Analyse 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: $OPT(i)$ = the maximum sum of subarrays of A ending with A_i.

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

Solution:

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

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:

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

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

Solution: $\Theta(n)$

2. (20 points) Having a Buffet

You plans 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 food as possible, you decide to taste at most one plate of each kind of food. Please design a dynmaic 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: (It is a classical knapsack problem.)
$OPT(i, w)$ = the maximum total value of a subset of the food $1, \ldots, i$ with weight limit w.

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

Solution:

$$OPT(i,w) = \begin{cases} 0 & \text{if } i = 0 \\ OPT(i-1,w) & \text{if } w_i > w \\ \max\{OPT(i-1,w),\ v_i + OPT(i-1,w-w_i)\} & \text{otherwise} \end{cases}$$

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

Solution: You can get your money's worth if $OPT(n, W) \geq T$ and cannot otherwise.

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') Desribe 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: Any proper greedy strategy is acceptable. For example, we apply a *Greedy by Ratio* strategy, where we repeatedly eat food with maximum $\frac{v_i}{w_i}$ whose w_i is smaller than the rest capacity.

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

Solution: Assume we apply *Greedy by Ratio* with $W = 10$. There are 2 kinds of food: food 1 with $v_1 = 3$ and $w_1 = 6$, food 2 with $v_2 = 2$ and $w_2 = 5$. Then the greedy strategy will only pick 1 plate of food 1 (worth 3) since $\frac{v_1}{w_1} > \frac{v_2}{w_2}$. However, the optimal solution is to pick 2 plates of food 2 (worth 4 in total).

(c) **Time for Dynamic Programming**

Again, you eat as many plates of each kind of food as you want. Please design a dynmaic 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: (It is a so-called the naïve solution of complete knapsack problem.)

$$OPT(i,w) = \begin{cases} 0 & \text{if } i = 0 \\ OPT(i-1,w) & \text{if } w_i > w \\ \max\limits_{0 \le k \le \frac{w}{w_i}} \{kv_i + OPT(i-1, w-kw_i)\} & \text{otherwise} \end{cases}$$

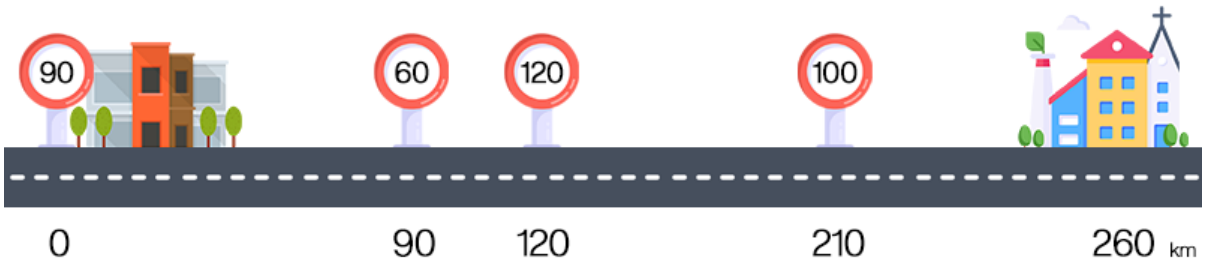
ii. (4') Try to solve this question by only modifying the Bellman equation in (a) without changing its runtime complexity. Give your modified Bellman equation.

Solution: (It is a so-called complete knapsack problem.)

$$OPT(i,w) = \begin{cases} 0 & \text{if } i = 0 \\ OPT(i-1,w) & \text{if } w_i > w \\ \max\{OPT(i-1,w),\ v_i + OPT(i, w-w_i)\} & \text{otherwise} \end{cases}$$

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 orgin 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 dynmaic 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: $OPT(i,k)$ = the minimum time of travelling from the origin to the i-th sign with k signs are already removed.

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

Solution:

$$OPT(i,k) = \begin{cases} 0 & \text{if } i = 1 \\ \min\limits_{\max\{0,\ i-k-1\} \le j \le i-1} \{\frac{x_i - x_j}{v_j} + OPT(j, k-(i-j-1))\} & \text{otherwise} \end{cases}$$

Explanation: (NOT Required)

- j-th term in min : removing signs between x_j and x_i (from x_{j+1} to x_{i-1}, in total $i-j-1$), following the limit v_j

Or equivalently,

$$OPT(i,k) = \begin{cases} 0 & \text{if } i = 1 \\ \min\limits_{0 \le j \le \min\{i-2,\ k\}} \{\frac{x_i - x_{i-j-1}}{v_{i-j-1}} + OPT(i-j-1, k-j)\} & \text{otherwise} \end{cases}$$

Explanation: (NOT Required)

- j-th term in min : removing j signs before x_i (so from x_{i-j} to x_{i-1}), following the limit v_{i-j-1}

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

Solution: For convenience, let $x_{n+1} = L$ and compute $OPT(n+1,k)$ for each k:

$$\min_{k \in \{0,1,\ldots,K\}} OPT(n+1,k)$$

Or equivalently,

$$\min_{k \in \{0,1,\ldots,K\}} \min_{n-k \le j \le n} \{\frac{L - x_j}{v_j} + OPT(j, k-n+j)\}$$

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

Solution: $\Theta(nK^2)$ (You will get only 0.5 credits if your answer is $O(n^3)$)

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. 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 dynmaic 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: $OPT(i, j)$ = the maximum score of aligning the first i nucleotides of Q and the first j nucleotides of S.

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

Solution:

$$OPT(i,j) = \begin{cases} j \cdot \text{Penalty}(-, Y) & \text{if } i = 0 \\ i \cdot \text{Penalty}(X, -) & \text{if } j = 0 \\ \max \begin{cases} OPT(i-1, j-1) + \text{Score}(S_i, Q_j) \\ OPT(i-1, j) + \text{Penalty}(S_i, -) \\ OPT(i, j-1) + \text{Penalty}(-, Q_j) \end{cases} & \text{otherwise} \end{cases}$$

Explanation: (NOT Required)

- The 1st term in max : align $S_{1\ldots i}$ with $Q_{1\ldots j}$ and match or mismatch S_i with Q_j
- The 2nd term in max : align $S_{1\ldots i-1}$ with $Q_{1\ldots j}$ and insert a gap to Q to align S_i
- The 3rd term in max : align $S_{1\ldots i}$ with $Q_{1\ldots j-1}$ and insert a gap to S to align Q_j

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

Solution: $OPT(m, n)$

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

Solution: $\Theta(mn)$