Problem Set 8

Please write your solutions in the LaTeX template provided. Aim for concise solutions; convoluted and obtuse descriptions might receive low marks, even when they are correct. **There is no coding part to submit.**

Please solve each of the following problems using **dynamic programming**. For each problem, be sure to define a set of subproblems, relate the subproblems recursively, argue the relation is acyclic, provide base cases, construct a solution from the subproblems, and analyze running time. Correct but inefficient dynamic programs will be awarded significant partial credit.

For each problem below, please indicate whether the requested running time is either:
(1) **polynomial**, (2) **pseudopolynomial**, or (3) **exponential** in the size of the input.
This categorization will be worth **3 points per problem**.

Problem 8-1. [25 points] **Oil Well that Ends Well**

The oil wells of tycoon Ron Jockefeller will produce m oil barrels this month. Ron has a list of n orders from potential buyers, where the ith order states a willingness to buy a_i barrels for a total price of p_i (not per barrel), which may be negative.[1] Each order must be filled completely or not at all, and can only be filled once. Ron does not have to sell all of his oil, but he must pay s dollars per unsold barrel in storage costs. Describe an $O(nm)$-time algorithm to determine which orders to fill so that Ron can maximize his profit (which may be negative).

Solution:

1. **Subproblems**
   - $x(i, j)$: maximum possible profit by selling j barrels to the suffix of buyers i through n
   - for $i \in \{1, \dots, n+1\}, j \in \{0, \dots, m\}$
2. **Relate**
   - Ron can either sell or not sell to buyer i
     - If he sells, can't sell to them again but must sell remaining barrels
     - If not sells, no change to barrels to be sold
   - $x(i, j) = \max \left\{ \begin{array}{ll} p_i + x(i+1, j-a_i) & \text{if } j \geq a_i \\ x(i+1, j) & \text{always} \end{array} \right\}$
3. **Topo. Order**
   - $x(i, j)$ only depends on subproblems with strictly larger i, so acyclic

[1] Earlier this year, oil futures contract prices went negative: people were paying money to not accept delivery of oil because demand for oil had fallen dramatically and there was a shortage of places to store oil.

4. **Base**
   - If no more buyers remain, Ron must pay s dollars per unsold barrel
   - $x(n+1, j) = -sj$ for all $j \in \{0, \ldots, m\}$
5. **Original**
   - Solve subproblems via recursive top down or iterative bottom up
   - $x(1, m)$ is the maximum profit allowing sales of m barrels to all buyers
   - Store parent pointers to reconstruct which sales fulfill an optimal order
6. **Time**
   - # subproblems: $(n+1)(m+1) = O(nm)$
   - Work per subproblem: $O(1)$
   - $O(nm)$ running time, which is **pseudopolynomial** in the size of the input

Rubric: (Same for all problems in PS7)

- S: 5 points for a correct subproblem description
- R: 5 points for a correct recursive relation
- T: 2 points for indication that relation is acyclic
- B: 2 point for correct base cases in terms of subproblems
- O: 2 point for correct original solution in terms of subproblems
- T: 2 points for correct analysis of running time
- 4 points if a correct dynamic program is efficient (meets requested bound)
- 3 points for correctly labeling running time polynomial or pseudopolynomial
- Partial credit may be awarded

Problem 8-2. [25 points] **Splits Bowling**

In Lecture 15, we introduced **Bowling**: a one-player game played on a sequence of n pins, where pin i has integer value v_i (possibly negative). The player repeatedly knocks down pins in two ways:

- knock down a single pin, providing v_i points; or
- knock down two adjacent pins i and $i+1$, providing $v_i \cdot v_{i+1}$ points.

Pins may be knocked down at most once, though the player may choose not to knock down some pins. A Bowling variant, **Split Bowling**, adds a third way the player can knock down two pins forming a **split**, specifically:

- knock down two pins i and $j > i+1$ if all pins in $\{i+1, \ldots, j-1\}$ between them have already been previously knocked down, providing $v_i \cdot v_j$ points.

Describe an $O(n^3)$-time algorithm to determine the maximum score possible playing Split Bowling on a given input sequence of n pins.

Solution:

1. **Subproblems**
   - Assume pins are zero-indexed in array $V = (v_0, \dots, v_{n-1})$
   - $x(i, j, c)$: maximum possible score playing Split Bowling on substring of pins $V[i : j]$, where all pins in range must be knocked down if $c = 1$, but unconstrained if $c = 0$
   - for $i \in \{0, \dots, n\}, j \in \{i, \dots, n\}, c \in \{0, 1\}$
2. **Relate**
   - Guess what to do with pin i
     - If $c = 0$, can choose not to knock down pin i
     - Otherwise, can knock down pin i in one of three ways:
       * knock down by itself getting v_i points,
       * knock down with pin $i + 1$ for $v_i \cdot v_{i+1}$ points, or
       * knock down with pin k for $v_i \cdot v_k$ points for some $k \in \{i + 2, \dots, j - 1\}$
     - In last case, need max value to knock down all pins between and pins after
   - $x(i, j, c) = \max \left\{ \begin{array}{ll} x(i + 1, j, c) & \text{if } c = 0 \\ v_i + x(i + 1, j, c) & \text{if } i < j \\ \displaystyle\max_{k \in \{i+1, \dots, j-1\}} v_i \cdot v_k + x(i + 1, k - 1, 1) + x(k + 1, j, c) & \text{if } i + 1 < j \end{array} \right\}$
3. **Topo. Order**
   - $x(i, j)$ only depends on subproblems with strictly smaller $j - i$, so acyclic
4. **Base**
   - $x(i, i, k) = 0$ (no more pins gives zero value)
   - for $i \in \{0, \dots, n\}, k \in \{0, 1\}$
5. **Original**
   - Solve subproblems via recursive top down or iterative bottom up
   - $x(0, n, 0)$: maximum score possible playing on all pins, unconstrained
6. **Time**
   - # subproblems: $O(n^2)$
   - Work per subproblem: $O(n)$
   - $O(n^3)$ running time, which is **polynomial** in the size of the input

Problem 8-3. [25 points] **Quarter Partition**

Given a set $A = \{a_0, \dots, a_{n-1}\}$ containing n distinct positive integers where $m = \sum_{a_i \in A} a_i$, describe an $O(m^3 n)$-time algorithm to return a **partition** of A into four subsets $A_1, A_2, A_3, A_4 \subsetneq A$ (where $A_1 \cup A_2 \cup A_3 \cup A_4 = A$) such that the maximum of their individual sums is as small as possible, i.e., such that $\max\left\{\sum_{a_i \in A_j} a_i \mid j \in \{1, 2, 3, 4\}\right\}$ is minimized.

Solution:

1. **Subproblems**
   - $x(k, s_1, s_2, s_3)$: True if it is possible to partition suffix of items $A[k:]$ into four subsets A_1, A_2, A_3, A_4, where $s_j = \sum_{a_i \in A_j} a_i$ for all $j \in \{1, 2, 3\}$, and false otherwise
   - for $k \in \{0, \dots, n\}$ and $s_1, s_2, s_3 \in \{0, \dots, m\}$
2. **Relate**
   - Integer a_k must be placed in some partition. Guess!
   - $x(k, s_1, s_2, s_3) = \text{OR}\left\{\begin{array}{ll} x(k+1, s_1 - a_k, s_2, s_3) & \text{if } a_k \le s_1 \\ x(k+1, s_1, s_2 - a_k, s_3) & \text{if } a_k \le s_2 \\ x(k+1, s_1, s_2, s_3 - a_k) & \text{if } a_k \le s_3 \\ x(k+1, s_1, s_2, s_3) & \text{always} \end{array}\right\}$
3. **Topo. Order**
   - $x(k, s_1, s_2, s_3)$ only depends on subproblems with strictly larger k, so acyclic
4. **Base**
   - $x(n, 0, 0, 0) =$ True (can partition zero integers into zero sum subsets)
   - $x(n, s_1, s_2, s_3) =$ False for any $s_1, s_2, s_3 > 0$
   - (cannot partition zero integers into any subset with positive sum)
   - for $i \in \{0, \dots, n\}, k \in \{0, 1\}$
5. **Original**
   - Solve subproblems via recursive top down or iterative bottom up
   - Let $m(0, s_1, s_2, s_3) = \begin{cases} \max\{s_1, s_2, s_3, m - s_1 - s_2 - s_3\} & \text{if } x(0, s_1, s_2, s_3) \text{ is True} \\ \infty & \text{otherwise} \end{cases}$
   - Solution to original problem is then given by:
   - $\min\{m(0, s_1, s_2, s_3) \mid s_1, s_2, s_3 \in \{0, \dots, m\}\}$
   - Store parent pointers to reconstruct each subset
6. **Time**
   - # subproblems: $O(m^3 n)$
   - Work per subproblem: $O(1)$
   - Work to compute original: $O(m^3)$
   - $O(m^3 n)$ running time, which is **pseudopolynomial** in the size of the input

Problem 8-4. [25 points] **Corrupt Chronicles**

Kimmy Jerk is the captain of the USS Exitcost, a starship charged with exploring new worlds. Each day, Capt. Jerk uploads a **captain's log** to the ship's computer: a string of at most m lowercase English letters and spaces, where a **word** in a log is any maximal substring not containing a space.

One day, Capt. Jerk is abducted, and Communications Officer Uhota Nyura goes to the captain's logs looking for evidence. Unfortunately, the log upload system has malfunctioned, and has **corrupted** each of the last n logs by dropping all spaces. Officer Nyura wants to restore the spaces based on Capt. Jerk's speech patterns in previous logs. Given a list L_c of the n corrupted logs, as well as a list L_u of $O(m^2 n)$ uncorrupted logs from before the malfunction, Officer Nyura wants to:

- for each word w appearing in any log in L_u, compute $f(w)$: the positive integer number of times word w appears in L_u (note, $f(w)$ is zero for any word w not appearing in L_u); and
- for each log $\ell_i \in L_c$, return a **restoration** R_i of ℓ_i (i.e, a sequence of words R_i whose ordered concatenation equals ℓ_i), such that $\sum_{w \in R_i} f(w)$ is maximized over all possible restorations.

Describe an $O(m^3 n)$-time algorithm to restore Capt. Jerk's logs based on the above protocol.

Solution: Observe that if we can process L_u in $O(m^2 n)$ time, and then compute the restoration the n corrupted logs, each in $O(m^3)$ time, then we will be within the desired bound.

First, scan through each of the $O(m^2 n)$ at most length-m uncorrupted logs, and insert each word w into a hash table H mapping to frequency $f(w)$. Specifically, if w does not appear in H, add it to H mapping to 1, and if w does appear in H, increase the mapped value by 1. This process computes all $f(w)$ for words appearing in any log in L_u directly in expected $O(m^3 n)$ time, since the time to hash each word is linear in its length.

Second, we compute a restoration for each log ℓ_i in L_c via dynamic programming in expected $O(m^3)$ time, leading to an expected $O(m^3 n)$ running time in total, which is **polynomial** in the size of the input (there are at least $\Omega(nm)$ characters in the input).

1. **Subproblems**
   - $x(j)$: maximum $\sum_{w \in R_{i,j}} f(w)$ for any restoration $R_{i,j}$ of suffix $\ell_i[j:]$
   - for $j \in \{0, \ldots, |\ell_i| \le m\}$
2. **Relate**
   - Guess the first word in $\ell_i[j:]$ and recurse on the remainder
   - $x(j) = \max\{f(\ell_i[j:k]) + x(k) \mid k \in \{j+1, \ldots, |\ell_i|\}\}$
   - Where $f(w) = \begin{cases} H(w) & \text{if } w \in H \\ 0 & \text{otherwise} \end{cases}$
3. **Topo. Order**
   - $x(j)$ only depends on strictly larger j, so acyclic
4. **Base**

- $x(|\ell_i|) = 0$ (no log left to restore)

5. **Original**
   - Solve subproblems via recursive top down or iterative bottom up
   - Solution to original problem is given by $x(0)$
   - Store parent pointers to reconstruct a restoration achieving value $x(0)$

6. **Time**
   - # subproblems: $O(m)$
   - Work per subproblem: $O(m^2)$
   - ($O(m)$ choices, and each hash table lookup costs expected $O(m)$ time)
   - Expected $O(m^3)$ running time per restoration