Problem Set 8

Name: Thomas Ren

Collaborators: None

Problem 8-1.

We will solve this problem following the ***SRT BOT*** *template.*

- **Subproblem**
  - To make this problem runs in $O(nm)$-time, we need two parameters to recursively call on **the length of the list of orders** n **and the number of oil barrels** m.
  - Define a subproblem $x(i, j)$: the maximum profits Ron can get giving the suffixes list $L[i:]$ with j barrels of oil to sell.
- **Relation**
  - For the i_{th} barrels price pair (a_i, p_i), we don't know whether Ron would take this order. **Guess!**
  - Locally burte-force all possibilities, if $a_i < j$, then we can sold the oils with price p_i. Otherwise, we don't sell this order and recursively look for $x(i + 1, j)$.
  -
$$x(i,j) = max \begin{cases} p_i + x(i+1, j - a_i), & \text{if} a_i < j \\ x(i+1, j), & \text{always} \end{cases}$$
- **Topological Order**
  - The subproblem $x(i, j)$ depends on strictly smaller $j - i$, so **acyclic**
- **Base Case**
  - $x(n, m) = -m \cdot s$, for Ron doesn't have any order, so he paid for all m barrels' storing fee.
  - $x(n, 0) = 0$, for Ron doesn't have any barrels left, so he cannot make any profit and doesn't need to pay the storing fee.
- **Original Problem**
  - $x(0, m)$, when Ron has the full list of orders and all barrels of oil to sell.
- **Time**
  - Number of subproblems: $O(nm)$
  - Work per subproblem: $O(1)$
  - Total running time: $O(nm)$
  - **Pseudopolynomial** if m fits in a word-RAM.

Problem 8-2.

We will solve this problem following the ***SRT BOT*** *template.*

- **Subproblem**
  - Remember that when we solve the bowling problem in *Lecture 15*, we define a subproblem $x(i, j)$ to represent the maximum value we can get in range i to j.
  - Here, we will add a new parameter to represent the vacuum in range i to j.
  - Suppose that the values of bowling form a list V, where $V[i] = v_i$ represents the i_{th} value of a bowling pin.
  - subproblem $x(i, j, v)$: the maximum value player can get in $L[i : j+1]$, and v represents whether this range is vacuum. If $v = 1$, then all pins in range i to j have been knocked down, and $v = 0$ otherwise.
  - for $i, j \in \{1, 2, \cdots, n\}, \quad v \in \{0, 1\}$
- **Relation**
  - In this part, our bidirectional subproblem is actually related similar to suffixes, but sometime we need the j parameter to locate the vacuum range.
  - for the i_{th} pin, if $v = 0$, we can knock it or not. **Guess!**
    - $*$**Locally brute-force what to do with the i_{th} pin.**
    - $*$don't knock down pin i, we go the subproblem $x(i + 1, j, v)$, assuming $x(i, j, 0)$
    - $*$knock down pin i by itself, we go to $x(i + 1, j, v) + v_i$
    - $*$knock down pin i with pin $i + 1$, we go to $x(i + 2, j, v) + v_i \cdot v_{i+1}$
    - $*$knock down pin i with pin k, assuming vacuum exists in range $i + 1$ to $k - 1$, then we go to $x(k + 1, j, v) + x(i + 1, k - 1, 1) + v_i \cdot v_k$
  -

$$x(i, j, v) = max \left\{ \begin{array}{l} x(i + 1, j, v) \\ x(i + 1, j, v) + v_i \\ x(i + 2, j, v) + v_i \cdot v_{i+1} \\ max\{v_i \cdot v_k + x(i + 1, k - 1, 1) + x(k + 1, j, v), \quad k = i + 2, \cdots, j - 1\} \end{array} \right\}$$

- **Topological Order**
  - The dependency requires strictly smaller $j - i$, so **acyclic**.
- **Base Case**
  - when $j - i$ reduces to zero, there is no more pins to hit, so: $x(i, i, v) = 0$
  - for $i \in \{0, 1, \cdots, n - 1\}, \quad k \in \{0, 1\}$
- **Original Problem**
  - when we have the whole list of pins to hit and don't need to vacuum the whole list.
  - $x(0, n - 1, 0)$

- **Time**
  - Number of subproblems: $O(n^2)$
  - Work per subproblem: $O(n)$
  - Total running time: $O(n^3)$
  - **Polynomial** in all circumstances.

Problem 8-3.

We will solve this problem following the ***SRT BOT*** *template.*

- **Subproblem**
  - This problem may sound like an optimization problem, for we need to find out the **minimum of the maximum** individual sum. However, we only know how to solve a **decision problem involving partition**.
  - So, we will solve a *decision problem* using dynamic programming, then apply this solved decision problem to solve an optimization problem.
  - $x(i, s_1, s_2, s_3)$: for suffixes i of list $A[i:]$, return **True** if suffixes can be partitioned into four subsets, with $s_j = \sum_{a_k \in A_j} a_k | j \in \{1, 2, 3\}$, and **False** otherwise.
  - Note that we only need s_1, s_2, s_3 to represent our partitions sum, cause the last partition's sum can be easily computed as $s_4 = m - s_1 - s_2 - s_3$.
- **Relation**
  - Since the i_{th} integer a_i can be put in any partition, $x(i, s_1, s_2, s_3)$ will be True if a_i can be put in any of the four partitions:
  - $$x(i, s_1, s_2, s_3) = OR \left\{ \begin{array}{l} x(i+1, s_1 - a_i, s_2, s_3), \\ x(i+1, s_1, s_2 - a_i, s_3), \\ x(i+1, s_1, s_2, s_3 - a_i), \\ x(i+1, s_1, s_2, s_3) \end{array} \right\}$$
- **Topological Order**
  - The dependency requires strictly larger i, so **acyclic**.
- **Base Case**
  - Since i is always incrementing, the first parameter has to be n.
  - $x(n, 0, 0, 0)$ is True. Since we can always put zero number to zero sums.
  - $x(n, s_1, s_2, s_3)$ is False. Since we we cannot split zero to positive numbers.
  - for $s_1, s_2, s_3 \in (0, m]$
- **Original Problem**
  - If $x(0, s_1, s_2, s_3)$ is True, then we can try to find the minimum of the maximum of $s_1, s_2, s_3, m - s_1 - s_2 - s_3$
  - Assume $y(s_1, s_2, s_3) = max\{s_1, s_2, s_3, m - s_1 - s_2 - s_3\}$ if $x(0, s_1, s_2, s_3)$ is True.
  - for $s_1, s_2, s_3 \in (0, m]$
  - The original problem is $min\{y(s_1, s_2, s_3)\}$, for $s_1, s_2, s_3 \in (0, m]$.
- **Time**
  - Number of subproblems: $O(nm^3)$
  - Work per subproblem: $O(1)$

–Work to solve the original problem using the decision subproblem: $O(m^3)$
–Total running time: $O(nm^3 + m^3) = O(nm^3)$
–**Pseudopolynomial** if m fits in a word-RAM model.

Problem 8-4. For those uncorrupted logs, we need to store those in a data structure that can be efficient to look up.
So, we can build a **Hash Table** H to achieve the lower bound:

1. Build an empty hash table.
2. Enumerate through the *word* w in a *log*.
3. If w is already in H, then increment the mapping number $f(w)$, and set to 1 otherwise.
4. The number of words is $O(m^3n)$, to insert each word, takes $O(1)$-time. Sums up to $O(m^3n)$-time.

Then, we can solve the maximized **Restoration** by **dynamic programming**. *We will solve this problem following the* ***SRT BOT*** *template*.

- **Subproblem**
  - $x(i)$: the maximized **Restoration** for suffixes $l_j[i:]$ while keep the parent pointers.
- **Relation**
  - $x(i) = max\{f(l_j[i:k]) + x(k) \big| k \in \{i+1, \cdots, |l_j|\}\}$
  - $f(l_j[i:k])$ returns the number of times word $l_j[i:k]$ appears, and zero if word $l_j[i:k]$ doesn't exist in H.
- **Topological Order**
  - Dependency relies on strictly larger i, so **acyclic**.
- **Base Case**
  - $x(|l_j|) = 0$
  - for $j \in \{0, 1, \cdots, n-1\}$
- **Original Problem**
  - $x(0)$ and return the restoration R_j using parent pointers.
- **Time**
  - Number of subproblems: $O(m)$
  - Work per subproblem: $O(m^2)$
    ($O(m)$ for searching the maximum value, $O(m)$ for a single lookup in hash table when, worst case, word $l_j[i:k]$ is not in H.)
  - Total running time per log: $O(m^3)$
  - For n corrupted logs, takes $O(m^3n)$-tim