Practice Problem Set – Dynamic Programming

1. There is a reality show where there are a collection of n songs. You know all the songs, all the judges, and your own dancing ability pretty well. For every $k \in \{1, 2, \ldots, n\}$, you know that if you dance to the k-th song, then you will earn $Score[k]$ points, but then you will be physically unable to dance at the next $Wait[k]$ songs, i.e. $(k+1)$-th song to $(k + Wait[k])$-th songs. The dancer with the highest total score at the night wins the contest. So you want your total score to be as high as possible.

   Design a polynomial-time algorithm that computes the maximum total score you can achieve.

2. Suppose you are managing the construction of billboards on the Stephen Daedalus Memorial Highway, a heavily travelled stretch of road that runs west-east for M miles. The possible sites for billboards are given by numbers $x_1, \ldots, x_n$ each in the interval $[0, M]$. More specifically, x_i specifies the position of the i-th site along this highway. If you place a billboard at location x_i, you receive a revenue of $r_i > 0$.

   Regulations required by the countries highway department enforces that no two of the billboards can be within less than or equal to 5 miles of each other. You would like to place billboards at a subset of the sites so as to maximize your total revenue. Design an algorithm that takes the input instance of this problem as input and returns the maximum total revenue that can be obtained from any valid subset of sites. The running time of the algorithm should be polynomial in n.

3. You are going on a long trip. You start on the road at mile post 0. Along the way there are n hotels, at mile posts $a_1 < a_2 < \cdots < a_n$, where each ai is measured from the starting point. The only places you are allowed to stop are at these hotels, but you can choose which of the hotels you stop at. You must stop at the final hotel (at distance a_n), which is your destination. You'd ideally like to travel 250 miles a day, but this may not be possible (depending on the spacing of the hotels). If you travel x miles during a day, the penalty for that day is $(250 - x)^2$. You want to plan your trip so as to minimize the total penalty – that is, the sum, over all travel days, of the daily penalties. Give an efficient algorithm that determines the minimum cost of a sequence of hotels at which to stop.

4. Suppose you are working on a lightweight consulting business, just you – two associates, and some rented equipment. Your clients are distributed between north part, and south part of the country. Each month, you can either run your business from an office in Delhi (DEL), or from an office in Mumbai (BOM). In the i-th month, you will incur an operating cost of d_i if you run from DEL; you will incur an operating cost of m_i if you run from BOM. However, if you run the business in one city in the i-th month and move it to another city in the $(i+1)$-th month, then you incur a fixed moving cost of M to switch base offices.

   Given a sequence of n months, a plan is a sequence of n locations – each one equal to either DEL or BOM – such that the i-th location indicates a city in which you will be based in the

i-th month. The cost of a plan is the sum of the operating costs for each of the n months, plus a moving cost of M for each time you switch cities. The plan can begin in any of the two cities.

Example: Suppose that $n = 4$, $M = 10$, and the operating costs are given by the following example. The values are $d_1 = 1, d_2 = 3, d_3 = 20, d_4 = 30$ and $m_1 = 50, m_2 = 20, m_3 = 2, m_4 = 2$. The plan of a minimum cost would be the sequence of locations (DEL, DEL, BOM, BOM) with a total cost $1 + 3 + 10 + 2 + 4$ while the term 10 stands for moving cost M that was conducted between the second and the third month.

Design and analyze a polynomial-time algorithm that takes the values n, M, the values d_i's and m_i's and returns the cost of an optimal plan.

5. After graduating from Sham-Poobanana University, you decide to interview for a position at the Wall Street bank Long Live Boole. The managing director of the bank, Eloob Egroeg, poses a 'solve-or-die' problems to each new employee, which they must solve within hours. Those who fail to solve the problem are fired immediately! Entering the bank for the first time, you notice that the employee offices are organized in a straight row, with a large T or F (i.e. T or F in the upper case) printed on the door of each office. Furthermore, between each adjacent pair of offices, there is a board marked by one of the symbols $\wedge, \vee, \oplus$. When you ask about these arcane symbols, Eloob confirms that T and F represent the boolean values TRUE and FALSE respectively. And the symbols on the boards represent the standard boolean operators AND, OR, XOR. He also explains that these letters and symbols describe whether certain combinations of employees can work together successfully. At the start of any new project, Eloob hierarchically clusters his employees by adding parentheses to the sequence of symbols, to obtain an unambiguous boolean expression. The project is successful if this parenthesized boolean expression evaluates to T.

   For example, if the bank has three employees, and the sequence of symbols on and between their doors is $T \wedge F \oplus T$, there is exactly one successful parenthesization scheme: $(T \wedge (F \oplus T))$. However, if the list of door symbols is $F \wedge T \oplus F$, there is no way to add parentheses to make the project successful.

   Eloob finally poses your **solve-or-die** interview question: Describe an algorithm to decide whether a given sequence of symbols can be parenthesized so that the resulting boolean expression evaluates to T . Your input is an array $S[0, 1, \ldots, 2n]$, where $S[k] \in \{T, F\}$ when k is even, and $S[k] \in \{\vee, \wedge, \oplus\}$ when k is odd.

6. Given a vertex weighted rooted binary tree T whose weight function $\mathrm{wt} : T \to \mathbb{R}^+$. A set of vertices $S \subseteq V(T)$ is called a *dominating set* if for every $u \in V(T)$, either $u \in S$ or there is $v \in S$ such that $(u, v) \in E(T)$. Informally speaking, a set S of vertices is a dominating set if every vertex u is either in S or has some neighbor v that is in S. The weight of a vertex subset S is denoted by $\mathrm{wt}(S) = \sum_{u \in S} \mathrm{wt}(u)$. A dominating set S of T is of *minimum weight* if it has smaller weight than any other dominating set of T.

   Design a dynamic programming based polynomial-time algorithm that computes the total weight of a minimum weighted dominating set of a binary tree.

7. Suppose that you are given an array $A[1, 2, \ldots, n]$ of positive integers. An *increasing back and fourth sequence* is a sequence of indices $J[1, 2, \ldots, \ell]$ satisfying the following properties.

   - $1 \leq J[i] \leq n$ for all j.

- $A[J[i]] < A[J[i+1]]$ for all $i < \ell$.
- If $J[i]$ is even, then $J[i+1] > J[i]$.
- If $J[i]$ is odd, then $J[i+1] < J[i]$.

For example, if the numbers in the input array A appear in this order $1, 1, 8, 7, 5, 6, 3, 6, 4, 4, 8, 3, 9, 1, 2, 2, 3, 9, 4, 0$, then a longest increasing back and fourth sequence is $0(20), 1(1), 2(15), 3(18), 4(10), 6(6), 7(4), 8(3), 9(13)$.

Describe an algorithm that computes the length of a longest increasing back and forth sequence.

8. Given a set A of n elements and an integer k, design an algorithm to count the number of subsets of A of size exactly k. Your solution will receive zero marks if it uses an algorithm that computes $\frac{n!}{k!(n-k)!}$ just by using simple algorithms that compute $n!$, $(n-k)!$, and $k!$. Your algorithm must be either a recursive algorithm or should run in time polynomial in n and k.

9. If there is a team S of n players, then design an algorithm that counts the number of tuples (A, x) such that $A \subseteq S$, $x \in A$ and $|A| = k$. Same as previous question, if your algorithm uses $\frac{n!}{k!(n-k)!}$ to compute the number of teams having exactly k players, then you will be awarded zero marks.

10. You and your eight-year-old nephew Elmo decide to play a simple card game. At the beginning of the game, the cards are dealt face up in a long row. Each card is worth a dierent number of points. After all the cards are dealt, you and Elmo take turns removing either the leftmost or rightmost card from the row, until all the cards are gone. At each turn, you can decide which of the two cards to take. The winner of the game is the player that has collected the most points when the game ends. Having never taken an algorithms class, Elmo follows the obvious greedy strategy—when it's his turn, Elmo always takes the card with the higher point value. Your task is to find a strategy that will beat Elmo whenever possible. (It might seem mean to beat up on a little kid like this, but Elmo absolutely hates it when grown-ups let him win.)