Planning and Reinforcement Learning (046203)
Homework 1

Question 1 (Longest Increasing Subsequence)

In the lectures we saw the *longest increasing subsequence problem*, in which we are given a sequence A_n of n numbers $\{a_i\}_{i=1}^n$, and we defined the *longest increasing subsequence* (LIS) of A_n as the longest *strictly* increasing subset of $\{a_i\}_{i=1}^n$. For example, the LIS of $(9, 2, 6, 4, 8)$ is $(2, 6, 8)$ or $(2, 4, 8)$. In the lectures we formulated a DP approach to find the length of the LIS of A_n.

Now, we define the *longest increasing odd even subsequence* (LIOES) of a sequence A_n, as the longest increasing subsequence of A_n whose terms alternate between odd and even numbers. For example, the LIOES of $(4, 5, 8, 3, 6, 7, 9)$ is $(4, 5, 6, 7)$ or $(4, 5, 6, 9)$.

a) Efficient solution – formulate a DP approach to find the size of the LIOES of a sequence of length n. Write the algorithm down in pseudocode.
b) How does the complexity of the algorithm you suggested compare to that of an exhaustive solution (checking all possibilities)?

Question 2 (Counting)

Given an integer number X, we would like to count in how many ways it can be represented as a sum of N natural numbers (with relevance to order) marked by $\Psi_N(X)$. For example, $\Psi_2(3)=2$ since: $3=1+2=2+1$.

a. Find the following: $\Psi_1(2), \Psi_2(4), \Psi_3(2), \Psi_N(N)$ and $\Psi_1(X)$.
b. Find a recursive equation for $\Psi_N(X)$.
c. Write a code in for finding $\Psi_N(X)$ using dynamic programming.
    1. What is the time and memory complexity of your algorithm?
    2. Find $\Psi_{12}(800)$.
d. Now assume each natural number i is associated with some cost c_i. For a given X, N we are interested in finding the lowest cost combination of natural numbers $\{x_i\}_{i=1}^N$ satisfying $\sum_{i=1}^N x_i = X$.
    1. Formulate the problem as a finite horizon decision problem: Define the state space, the action space and the cumulative cost function.
    2. Bound the complexity of finding the best combination.

Question 3 (Language model)

In the city of Bokoboko the locals use a language with only 3 letters ('B','K','O'). After careful inspection of this language, researchers have reached two conclusions:

I. Every word starts with the letter 'B'.

II. Every consecutive letter is distributed only according to the previous letter as follows:

$$P(l_{t+1} \mid l_t) = \begin{matrix} B \\ K \\ O \\ - \end{matrix} \begin{bmatrix} 0.1 & 0.325 & 0.25 & 0.325 \\ 0.4 & 0 & 0.4 & 0.2 \\ 0.2 & 0.2 & 0.2 & 0.4 \\ 1 & 0 & 0 & 0 \end{bmatrix}$$

Where '-' represents the end of a word. For example, the probability of the word 'bko' is given by $0.325 \cdot 0.4 \cdot 0.4 = 0.052$.

a. Find the probability of the following words: 'Bob', 'Koko', 'B', 'Bokk','Boooo'.

b. We wish to find the most probable word in the language of length K.

1. Formulate the problem as a finite horizon decision problem: Define the state space, the action space and the **multiplicative** cost function.
2. Bound the complexity of finding the best combination.
3. Find a reduction from the given problem to an analogous problem with **additive** cost function instead.
4. Explain when each approach (multiplicative vs. additive) is preferable. Hint: Think of the consequence of applying the reduction on the memory representation of a number in a standard operating system.
5. Write a code which finds the most probable word of a given size using dynamic programming. What is the most probable word of size 5?

Question 4 (MinMax dynamic programming)

In this problem we consider an adversarial version of finite horizon dynamic programming, which is suitable for solving 2-player games. In this setting, at time $k \in \{0,1,2,\ldots,N-1\}$ the system is at state $s_k \in S_k$, the agent chooses an action $a_k \in A_k(s_k)$ according to the agent policy $\pi_k^a(s_k)$, and subsequently the opponent chooses an action b_k from the set of allowable opponent actions $B_k(s_k, a_k)$, according to the opponent policy $\pi_k^b(s_k, a_k)$.

The system then transitions to a new state according to:

$$s_{k+1} = f_k(s_k, a_k, b_k), \quad k = 0,1,2,\ldots,N-1.$$

The instantaneous reward is denoted by $r(s_k, a_k, b_k)$, and, for an N-stage path $h_N = (s_0, a_0, b_0 \ldots, s_{N-1}, a_{N-1}, b_{N-1}, s_N)$ the cumulative reward is

$$R_N(h_N) = \sum_{k=0}^{N-1} r_k(s_k, a_k, b_k) + r_N(s_N).$$

Given s_0, the agent's goal is to find a policy π_a^* that maximizes the *worst-case* cumulative reward:

$$\pi_a^* \in \arg\max_{\pi_a} \min_{\pi_b} R_N(h_N)$$

a. Formulate a dynamic programming algorithm for this problem. Explain what the value function represents in this case.
b. What is the computational complexity of your algorithm?
c. Could this approach be used to solve the game of tic-tac-toe? Explain what are the states, actions and rewards for this game.
d. Could this approach be used to solve the game of chess? Explain.

Question 5 (Two traversals maximal score)

Given a matrix of dimension $R \times C$, where every entry contains a score, how do you collect a maximal overall score using two traversals?

The two traversals are defined as follows.

1. The two traversals have to start from top row, from the left and right corners, i.e., $[1,1]$ and $[1, C]$. The first (in the left corner) has to reach bottom left corner, i.e., $[R, 1]$. The second traversal should reach bottom right corner, i.e., $[R, C]$.
2. From a point $[i, j]$, each traversal can move to $[i + 1, j + 1]$ or $[i + 1, j - 1]$ or $[i + 1, j]$.

A traversal gets all points of a particular cell through which it passes. If one traversal has already collected points of a cell, then the other traversal gets no points if goes through that cell again.

Example For the following input:

3	6	8	2
5	2	4	3
1	1	20	10
1	1	20	10
1	1	20	10

Figure 1: Example of an input matrix.

The output will be 73.
First traversal collects total score of value $3 + 2 + 20 + 1 + 1 = 27$.
Second traversal collects total score of value $2 + 4 + 10 + 20 + 10 = 46$.
Total score collected $= 27 + 46 = 73$.

a. Analyze the complexity of an enumeration-based (naïve) algorithm.
b. Suggest a dynamic-programming based algorithm to solve the problem and analyze its complexity.