CS311 Homework 7

100 points, 20 points per problem

The NO-THREE-IN-LINE Problem

Consider an $n \times n$ grid of points in the Euclidean plane. The NO-THREE-IN-LINE problem involves marking points in this grid under the constraint that no three marked points may be collinear - that is, it is impossible to draw a straight line through three marked points. The decision variant of this problem asks if it is possible to mark m points in an $n \times n$ grid, while the optimization variant of this problem asks for the greatest number of points that can be marked in an $n \times n$ grid.

Problem 1

For this problem you will use a simulated annealing algorithm to find approximate solutions to the optimization variant of NO-THREE-IN-LINE.

(a) Specify in pseudocode a fitness function that evaluates potential solutions to NO-THREE-IN-LINE.

(b) In the context of simulated annealing, *neighbor solutions* are generated from a given solution S by applying a *move* to S. The available moves therefore determine which solutions are near to which other solutions, and have a fundamental effect on the efficacy of the algorithm. Specify an appropriate set of moves for the NO-THREE-IN-LINE problem.

(c) Discuss the strengths and weaknesses of your fitness function and set of moves. Are potential solutions that the set of moves label as 'near' actually similar in a useful sense? Why or why not? How does this affect the effectiveness of the simulated annealing algorithm?

Problem 2

For this problem you will use backtracking to find exact solutions to the decision variant of NO-THREE-IN-LINE. Backtracking involves enumerating a set of *partial candidates*, each of which could potentially be extended by some series of *candidate extension steps* into a complete and valid solution to the problem.

(a) Specify a set of partial candidates and a method for generating candidate extension steps for NO-THREE-IN-LINE.

(b) If a partial candidate can demonstrably *not* be extended into a complete solution, it should be abandoned, or *pruned*, as soon as possible. Specify in pseudocode a method that efficiently determines whether a partial candidate should be pruned. This method should be as aggressive as possible while not sacrificing significant computational effort or the correctness of the algorithm.

(c) Given your answers to (a) and (b), would a backtracking algorithm correctly solve the decision variant of NO-THREE-IN-LINE for every n?

The SEQUENCE-SEGMENTATION Problem

Consider the following problem:

> *Problem:* SEQUENCE-SEGMENTATION
> *Input:* A sentence S and an error function err.
> *Output:* A partition of S into a list W of words such that $err(W)$ is minimized.

For example, the sentence *monstercookie* could be partitioned into *monst* and *ercookie*, but (if err 'knows' English) *monster* and *cookie* might be a better partition.

Problem 3

Let $P(w)$ denote the probability of word w (determined, perhaps, by counting the number of times w occurs in all the books in Parks Library). Let UNIGRAM-SEGMENTATION be a variant of SEQUENCE-SEGMENTATION where:

$$err(W) = 1 - \prod_{i=0}^{|W|-1} P(W[i])$$

Specify in pseudocode a dynamic programming algorithm that efficiently solves UNIGRAM-SEGMENTATION. Hint: $\prod_{i=0}^{|W|-1} P(W[i]) = P(W[0]) \prod_{i=1}^{|W|-1} P(W[i])$

Problem 4

Let $P(w_n \mid w_1, w_2, \ldots, w_{n-1})$ denote the probability of the word w_n occurring immediately after the word sequence $w_1, w_2, \ldots, w_{n-1}$. In this context, even if *banana* and *cookie* are equally common words, *monster banana* could be less likely than *monster cookie* if *monster cookie* occurs more frequently. Let NGRAM-SEGMENTATION be a variant of SEQUENCE-SEGMENTATION where:

$$err(W) = 1 - \prod_{i=n}^{|W|-1} P(W[i] \mid W[i-n+1], \ldots, W[i-1])$$

Specify in pseudocode a dynamic programming algorithm that efficiently solves NGRAM-SEGMENTATION. As a function of n and $|S|$, what is the asymptotic worst-case time complexity of your solution? Hint: It may be useful to first solve BIGRAM-SEGMENTATION (i.e. NGRAM-SEGMENTATION where n=2) and then generalize your solution.

Problem 5

Consider the following problem:

Problem: COUNT-BOOLEAN-PARENTHESIZATIONS
Input: A sequence e of *true, false,* $\wedge$, $\vee$ and $\oplus$ symbols. This sequence encodes a boolean expression (with parentheses removed). $true{\wedge}false{\oplus}false{\vee}true$, for example, is a possible value for e. $truefalse\wedge$, however, is not, as it does not encode a valid boolean expression.
Output: The number of ways to parenthesize e that cause e to evaluate to *true*. All five parenthesizations of the above example, including $(true \wedge false) \oplus (false \vee true)$ and $true \wedge (false \oplus (false \vee true))$, evaluate to true, so the solution to the above example is 5.

Write an efficient dynamic programming algorithm that solves COUNT-BOOLEAN-PARENTHESIZATIONS. Hint: how would you insert parentheses to force a particular operator to be evaluated last?

Problem 6

Suppose you have n unusually durable eggs and live in an h-story apartment building. In order to understand just how durable the eggs are, you resolve to find the highest floor c from which the eggs can be dropped without breaking. A given egg can be dropped multiple times, but once it breaks, it naturally cannot be dropped again. You can't get any more of these special eggs, and of course simply *must* know the value of c, so you can't drop your last egg unless you would discover the value of c if it broke.

Problem: EGG-DROPPINGS
Input: The values of n and h.
Output: The minimum number of drops necessary to *guarantee* that the value of c will be discovered before you run out of eggs.

1. Let $n = 1$. What egg-dropping strategy will guarantee that you find the value of c in the minimal number of drops? What is the largest number of drops your strategy could require, and what value(s) of c cause this situation to arise?

2. Let $n = 2$. What egg-dropping strategy will guarantee that you find the value of c in the minimal number of drops? What is the largest number of drops your strategy could require, and what value(s) of c cause this situation to arise?

3. Let $n \in \mathbb{Z}^+$. Write a dynamic programming algorithm that efficiently solves EGG-DROPPINGS.