Problem Set 9

Assigned: July 23
Due: July 30

Problem 1.

Suppose that you have a function $f(L)$ that maps a list of integers L to an integer. Write a dynamic programming algorithm to do the following: given a list of integers L, partition it into sublists $s_1, s_2 \ldots s_k$ such that $\sum_{i=1}^{k} f(s_i)$ is maximal.

For instance, suppose the list L is $[28, 14, 1, 16]$. Then you need to choose the maximum of the following:

$$
\begin{array}{lll}
f([28,14,1,16]), & & \\
f([28]) + f([14,1,16]), & f([28,14]) + f([1,16]), & f([28,14,1]) + f([16]), \\
f([28,14]) + f([1]) + f([16]), & f([28]) + f([14,1]) + f([16]), & f([28]) + f([14]) + f([1,16]), \\
f([28]) + f([14]) + f([1]) + f[16]). & &
\end{array}
$$

Problem 2.

Consider the following generalization of the longest common subsequence problem, called "most similar subsequence". As in the LCS problem, you are given two strings and you want to find the subsequences that match best. However, in this version, the elements don't have to match perfectly; you can get partial credit for imperfect matches.

Specifically: Assume that these are strings of characters, and that you have a function $f(x, y)$ which assigns a numeric score to a match between characters x and y. You may assume that this is always equal to 1 if $x = y$ and always between 0 and 1 if $x \neq y$. Now, given two strings of characters U and V, find two subsequences of U and V with the highest total match score. Modify the LCS algorithm to solve this problem.

For example: Suppose that the characters are 'a', 'e', 'm', and 'n', and suppose that f is given by the following table:

f	a	e	m	n
a	1.0	0.7	0	0
e	0.7	1.0	0	0
m	0	0	1.0	0.2
n	0	0	0.2	1.0

Then if U="aaamenn" and V="eenmnm" then the following match has score 0.7+0.7+1+1+0.2 = 3.6

```
a   a   a   m   e   n   n
|   |       |       |   |
e   e   n   m       n   m
```

Problem 3

Suppose that you are given a fixed collection of integers, which you wish to arrange in a binary search tree. You are also given the frequency with which various elements are searched for. You want to construct the tree that minimizes the average time for a search. Assume that the time to search for an element is just equal to its depth in the tree (thus, the root takes time 0; its children take time 1, and so on.) Then the average search time for search tree T, $E(T) = \sum_{x \in S} p(x) \cdot d(x, T)$ where S is the set of elements, $p(x)$ is the frequency of searches for x, and $d(x, T)$ is the depth of x in tree T.

Construct a dynamic programming algorithm that finds the best binary search tree given this data.

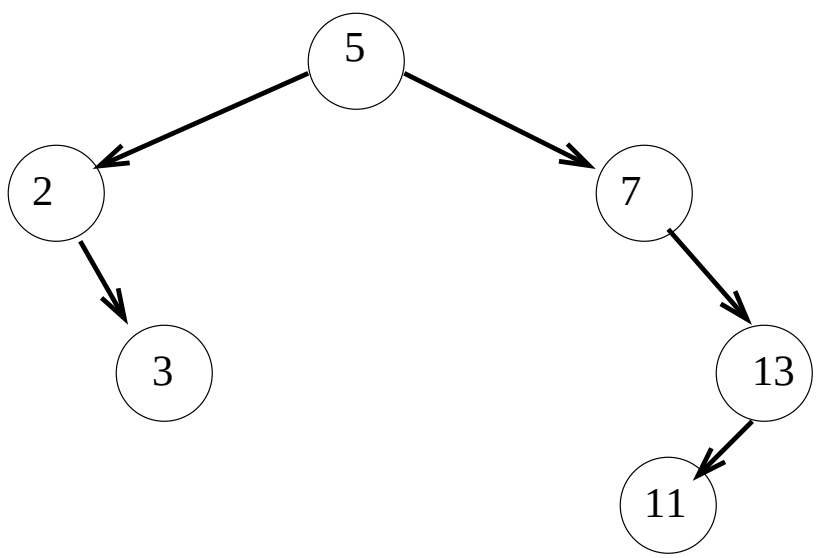
For example: suppose that you are given the following data:

Item	2	3	5	7	11	13
Frequency	0.1	0.05	0.3	0.2	0.15	0.2

Then for the tree below, the average search time is

$p(2) \cdot d(2, T) + p(3) \cdot d(3, T) + p(5) \cdot d(5, T) + p(7) \cdot d(7, T) + p(11) \cdot d(11, T) + p(13) \cdot d(13, T) =$
$0.1 \cdot 1 + 0.05 \cdot 2 + 0.3 \cdot 0 + 0.2 \cdot 1 + 0.15 \cdot 3 + 0.2 \cdot 2 = 1.25.$

(I don't know whether this is the optimal tree.)



Hint: Try all possible elements as the root; for each choice, recursively solve for the subtrees. Memoize the partial solutions using two two-dimensional arrays: $R[i, j]$ is the root of the optimal tree over elements i through j and $C[i, j]$ is the expected search time in the optimal tree down from its root. It is easily shown if you have a tree T with root R and with subtrees U and V then

$$E(T) = E(U) + E(V) + \sum_{x \in U} p(x) + \sum_{x \in V} p(x),$$

since each element in U and in V is pushed down one level of depth by being placed into T. Therefore, the choice of optimal binary search tree for items i through j does not depend on the depth of the root.