CS 125 Algorithms & Complexity — Fall 2016

Problem Set 3

Due: 11:59pm, Friday, September 23rd

See homework submission instructions at `http://seas.harvard.edu/~cs125/fall16/schedule.htm`

Problem 5 is worth one-third of this problem set, and problems 1-4 constitute the remaining two-thirds.

When asked to give an algorithm, you should prove correctness and analyze running time unless the problem states otherwise.

Problem 1

Consider the following text search problem. There is an alphabet Σ (which you can think of as just the set of integers $1, 2, \ldots, |\Sigma|$) and two strings $P \in \Sigma^m$, $T \in \Sigma^n$, $n \geq m$ (i.e. strings of length m, n, respectively, made up of characters in Σ). We would like to output a list of all indices i such that $T[i, i+1, \ldots, i+m-1] = P$, i.e. $t_{i+j-1} = p_j$ for $j = 1, \ldots, m$. At the end of class, whe we sketched how to use the FFT to solve this problem in $O(n \log n)$ time when $\Sigma = \{0, 1\}$, assuming a computer supporting infinite precision complex arithmetic (the full details are in the FFT lecture notes). For the rest of this problem, you will make the same precision assumptions.

(a) (1 point) Consider the "successive dot products" problem in which you are given a vector $(x_1, \ldots, x_m)$ and $(y_1, \ldots, y_n)$ with $n \geq m$ and you want to compute the list of numbers $\sum_{i=1}^{m} x_i y_{j+i}$ for $j = 0, \ldots, n-m$. Show how to use the FFT to perform this task in time $O(n \log n)$.

(b) (1 point) Improve your time from part (a) to $O(n \log m)$.

(c) (2 points) Back to the pattern matching problem: deal with the case when P (but not T) has "don't care" symbols *. A * symbol can match either a 0 or a 1.

(d) (2 points) Show how to deal with the case $|\Sigma| > 2$, without don't care symbols.

(e) (4 points) Give a solution for $|\Sigma| > 2$ **with** don't care symbols, which can match any character in $\Sigma = \{1, 2, \ldots, |\Sigma|\}$. An $O(n \log m)$ solution (which is possible) naturally implies you don't have to separately do parts (c) and (d).

Problem 2

You are given an n-digit positive integer x written in base-2. Give an efficient algorithm to return its representation in base-10 and analyze its running time. Assume you have black-box access to an integer multiplication algorithm which can multiply two n-digit numbers in time $M(n)$ for some $M(n)$ which is $\Omega(n^{1.01})$ and $O(n^2)$.

Each of the two problems below is a dynamic programming problem and should be viewed as having 3 parts. You should find a function f which can be computed recursively so that evaluation of f on a certain input gives the answer to the stated problem. Part (a) is to define f *in words* (without mention of how to compute it recursively). You should clearly state how many parameters f has, what those parameters represent, what f evaluated on those parameters represents, and what parameters you should feed into f to get the answer to the stated problem. Part (b) is to give a recurrence relation showing how to compute f recursively. In part (c) you should give the running time **and space** for solving the original problem using computation of f via memoization or bottom-up dynamic programming. If you need to use certain data structures to make computation of f faster, you should say so. **Note:** if there are multiple solutions to solve the stated dynamic programming problem, you should describe the most time-efficient one you know. If there are multiple solutions with the same asymptotic time complexity, you should describe the implementation that gives the best asymptotic space complexity.

As an example, suppose the stated problem is that you are given a positive integer n and a list `L[1..m]` of coin denominations available in some currency, and it is guaranteed that `L[1]` $= 1$. Then you are asked for the minimum number of coins needed to make change for n cents. For example, when $n = 8$ and `L` $= [1, 4, 5]$ then the answer is 2, since we can make change as $8 = 4+4$. Then for (a) we could define $f(i, j)$ to be the minimum number of coins needed to make change for i cents using only the coins in `L[1..j]`. The result we want is then $f(n, m)$. For (b), the base case is $j = 1$, in which case the answer is i since `L[1]` is 1. When $j > 1$, we can choose to use the jth coin (if its value is at most i) or not. Thus we obtain the recurrence relation

$$f(i,j) = \begin{cases} i, & \text{if } j = 1 \\ f(i, j-1), & \text{if } \texttt{L[j]} > i \\ \min\{1 + f(i - \texttt{L[j]}, j), f(i, j-1)\}, & \text{otherwise} \end{cases}$$

For (c), the running time is $O(nm)$ since there are nm possible pairs of input parameters to f, and for each we do constant work in the recurrence relation. The space naively is $O(nm)$ using memoization, but observe that $f(\cdot, j)$ values only depend on $f(\cdot, j-1)$ values. Thus we can improve the space to $O(n)$ using bottom-up dynamic programming.

Problem 3

You are given a sequence of n words $w_1, \ldots, w_n$ that must be displayed on the screen with a single space in between each word. Word w_i is s_i characters long; all characters (including spaces) consume 1 pixel width on the screen. The screen is only L pixels wide. Thus you have to insert line breaks so that no words spill off the edge of the screen. In doing so, each line j that is non-empty will have some number of unused pixels ℓ_j at the end. The "ugliness" of a formatting into T lines is then $\sum_{j=1}^{T} \ell_j^3$. Given the s_i and L, calculate the ugliness of the least ugly formatting attainable over all choices of where to insert line breaks. For example, if $L = 10$ and the sequence of words is "the quick brown fox jumped over the lazy dog", then

```
the quick
brown fox
jumped
over
the
lazy dog
```

has ugliness $1^3 + 1^3 + 4^3 + 6^3 + 7^3 + 2^3 = 633$. A less ugly choice of where to place line breaks is

```
the quick
brown
fox jumped
over the
lazy dog
```

which has ugliness $1^3 + 5^3 + 0^3 + 2^3 + 2^3 = 142$.

Problem 4

A trie, or prefix tree, is a certain kind of tree that helps when searching through string data. Unlike, e.g., binary search trees, the actual structure of the tree of a trie carries information about the data, so tries cannot be rebalanced to increase performance. Internal nodes are not necessarily binary, and can have any number of children (even 1).

You are given the tree structure of a trie, and your job is to pack nodes of the tree into blocks on disk. Disk has an infinite number of blocks, each of size B. Each node must be placed in exactly one block, but there is no restriction on which block you place a node in.

Given a packing of nodes into blocks, the cost of querying x (where x is a node in the tree) is the number of blocks that must be touched when traversing the unique path from the root of the tree to x. You should assume that the machine servicing the queries has enough memory to hold only a single block at a time. For example, suppose that in order to visit node x from the root, the order of nodes you visit is t, u, v, w, x, where t, w, x are in block 0, and u, v are in block 1. The cost of this query is then 3. You pay once to access block 0 in order to visit t, then you pay once again to access block 1 to visit u and v. In order to access block 1, you had to evict block 0, so you have to pay again to access block 0 in order to visit w then x.

There are n nodes in the tree, and you are given an array $T[1 \dots n]$ where $T[i]$ is the number of times node i was queried last week. Given this information, you want to find a packing of nodes into blocks on disk that would have minimized the sum of all costs of queries last week. Your algorithm does not need to output the actual packing, but just needs to return what sum of all costs of queries it achieves (a single number).

You should assume that you are given the tree in a format so that: (1) the root is vertex 1, (2) there is an array $p[1 \dots n]$ such that $p[i]$ is the parent of node i ($p[1]$ is 0), (3) there is an array $deg[1 \dots n]$ such that $deg[i]$ is the number of children of node i, and (4) for any i and $1 \leq j \leq deg[i]$, you can find the jth child of node i in constant time.

Problem 5 (Programming Problem)

Solve "PROP" on the programming server `https://cs125.seas.harvard.edu`.
(under "Problem Set 3").