CSCI E-124 Data Structures and Algorithms — Spring 2015

Problem Set 5

Due: 11:59pm, ~~Monday, March 23th~~ Tuesday, March 24th

See homework submission instructions at
`http://sites.fas.harvard.edu/~cs124/e124/problem_sets.html`

Problem 5 is worth 40% of this problem set, and problems 1-4 constitute the remaining 60%.

Each of the 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 weighted directed graph with n vertices $\{1, \ldots, n\}$, m edges, and positive weights. The problem then might ask you to print an array of shortest-path distances $d_{u,v}$ for all vertices u, v. This can be solved with the Floyd-Warshall algorithm. A solution to part (a) could say that $f(u, v, k)$ is a function that takes as input vertex ID's u, v, k, $f(u, v, k)$ represents the length of the shortest path from u to v in which all intermediate vertices should be from the set $\{1, \ldots, k\}$, and finally the answer $d_{u,v}$ is $f(u, v, n)$. Then part (b) should give a recurrence relation to compute f (see the top of page 3 of "Notes 5" on the lecture notes website). Part (c) in this case should argue why the running time is $\Theta(n^3)$ and how to get space $\Theta(n^2)$ using bottom-up dynamic programming.

Upon completing this problem set, if you have roughly two minutes to spare then please fill out this survey: `http://goo.gl/forms/3emr4H4Edd`

1 Problem 1

You are given an undirected tree (a connected and acyclic graph) in adjacency list form where each vertex v has a weight $w(v)$. For a subset S of vertices, we define its weight $w(S)$ to be $\sum_{v \in S} w(v)$. A set S is called *separated* if for all $u, v \in S$, the edge (u, v) does not appear in the graph. Give an algorithm to find the largest $w(S)$ achievable over all separated sets (you need not find the S achieving it, just $w(S)$).

2 Problem 2

There are four types of brackets: (,), <, and >. We define what it means for a string made up of these four characters to be *well-nested* in the following way:

1. The empty string is well-nested.
2. If A is well-nested, then so are <A> and (A).
3. If S, T are both well-nested, then so is their concatenation ST.

For example, (), <>, (()), (<>), ()<>, and ()<()> are all well-nested. Meanwhile, (, <,),)(, (<)>, and (<(> are not well-nested.

Devise an algorithm that takes as input a string s of length n made up of these four types of characters. The output should be the length of the shortest well-nested string that contains s as a subsequence. For example, if (<(> is the input, then the answer is 6; a shortest string containing s as a subsequence is ()<()>.

3 Problem 3

For a string s, we define s^k as concatenating s with itself k times. For example if $s = aba$, then $s^3 = abaabaaba$ and s^0 is the empty string.

We say x is a *powering* of s if x is a prefix of s^k for some k. We say y is a *mixing* of s, t if the characters of y can be partitioned into two (not necessarily contiguous) subsequences such that the first subsequence is a powering of s, and the second subsequence is a powering of t. For example $y = abcadcdba$ is a mixing of $s = ab, t = cd$ since $ababa$ is a prefix of s^3 and $cdcd$ is a prefix of t^2.

Describe an algorithm which, given y, s, t determines whether y is a mixing of s, t. In describing your solution, let k be the length of s, m be the length of t, and n be the length of y.

4 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.

5 Problem 5

Solve "Problem A - Break the String" on the programming server; see the "Problem Sets" part of the course web page for the link.