CS 170 Homework 3

Due **2/14/2022, at 10:00 pm (grace period until 11:59pm)**

1 Study Group

List the names and SIDs of the members in your study group. If you have no collaborators, you must explicitly write "none".

2 Preorder, Postorder

Suppose we just ran DFS on a directed (not necessarily strongly connected) graph G starting from vertex r, and have the pre-visit and post-visit numbers $pre(v), post(v)$ for every vertex. We now delete vertex r and all edges adjacent to it to get a new graph G'. Given *just* the arrays $pre(v), post(v)$, describe how to modify them to arrive at new arrays $pre'(v), post'(v)$ such that $pre'(v), post'(v)$ are a valid pre-visit and post-visit ordering for some DFS of G'.

Solution: For all v such that $pre(r) < pre(v) < post(v) < post(r)$, set $pre'(v) = pre(v) - 1, post'(v) = post(v) - 1$. For all other v in G', $pre'(v) = pre(v) - 2, post'(v) = post(v) - 2$.

One valid DFS on G' is: Run DFS, whenever we need to pick a new vertex to explore from, or whenever we choose a neighbor of the "current vertex" to explore, choose the unvisited vertex with the smallest value of $pre(v)$. This will visit all vertices in G' in the same order as the DFS on G. For example, notice that vertices adjacent to r have lower previsit numbers than vertices that can't be reached from r. So this DFS on G' will explore the vertices reachable from r in G first, and then vertices not reachable from r, just like the DFS on G.

For vertices with $pre(r) < pre(v) < post(v) < post(r)$, their pre/post-visit number decreases by 1 in this DFS since we no long pre-visit r before them. For all other vertices, their pre/post-visit number decreases by 2 since we no long pre-visit or post-visit r before them.

3 Where's the Graph?

Each of the following problems can be solved with techniques taught in lecture. Construct a simple directed graph and write an algorithm for each problem by black-boxing algorithms taught in lecture and in the textbook.

(a) Sarah wants to do an extra credit problem for her math class. She is given three numbers: 1, x, and y. Starting from x, she needs to find the shortest sequence of additions, subtractions, and divisions (only possible when the number is divisble by y) using 1 and y to get to 2021. If there are multiple sequences with the shortest length, return any one of them. She can use 1 and y multiple times. Give an algorithm that Sarah can query to get this sequence of arithmetic operations.

Solution: We can view this as a BFS problem, where the nodes in the graph are numbers that have been calculated. There are at most five edges from each node: $+1$, -1, $+y$, $-y$, and $/y$. Our algorithm starts from x and run BFS on this graph until the node 2021 is reached.

(b) There are n different species of Gem Berry, all descended from the original Stone Berry. For any species of Gem Berry, Emily knows all of the species *directly* descended from it. Emily wants to write a program. There would be two inputs to her program: a and b, which represent two different species of Gem Berries. Her program will then output one of three options in constant time (the time complexity cannot rely on n):

(1) a is descended from b.

(2) b is descended from a.

(3) a and b share a common ancestor, but neither are descended from each other.

Unfortunately, Emily is very limited in space and cannot store all of the descendants of a given species for all of the species. Give an algorithm that Emily's program could use to solve the problem above given the constraints. Emily can run some algorithm on her data and store the outputs of the algorithm for her program.

Solution: This is a directed graph, with each node representing some species and an edge from x to y indicating that y descended from x. We run DFS on this graph, storing the post-numbers and pre-numbers for each node. When the program is queried, it checks whether the edge (a, b) is a back edge (a is descended from b), a tree or forward edge (b is descended from a), or a cross edge (a and b share a common ancestor but are not descended from each other)

(c) Bob has n different boxes. He wants to send the famous "Blue Roses' Unicorn" figurine from his glass menagerie to his crush. To protect it, he will put it in a sequence of boxes. Each box has a weight w and size s; with advances in technology, some boxes have negative weight. A box a inside a box b cannot be more than 15% smaller than the size of box b; otherwise, the box will move, and the precious figurine will shatter. The figurine needs to be placed in the smallest box x of Bob's box collection.

Bob (and Bob's computer) can ask his digital home assistant Falexa to give him a list of all boxes less than 15% smaller (but not necessarily lighter) than a given box c. Bob will need to pay postage for each unit of weight. Find an algorithm that will find the lightest sequence of boxes that can fit in each other in linear time (in terms of the graph).

Hint: The time complexity should be no more than quadratic in number of boxes.

Solution: We can view each box as a node in a directed graph, with an edge from a to b indicating that a fits in b. Since Falexa only gives the edges in the opposite direction (in that it tells Bob the boxes that can fit into b), we have to construct a graph and then reverse the edges, which is doable in linear time. This graph is a DAG. If box a can fit in box b and box c can fit in box a, box b cannot fit into box c. We set incoming edge weights to a node a to be the weight w of the corresponding box. We can linearize this DAG and then find the shortest path from the smallest box x to any other box in linear time by running DFS.

4 The Greatest Roads in America

Arguably, one of the best things to do in America is to take a great American road trip. And in America there are some amazing roads to drive on (think Pacific Crest Highway, Route 66

etc). An intrepid traveler has chosen to set course across America in search of some amazing driving. What is the length of the shortest path that hits at least k of these amazing roads?

Assume that the roads in America can be expressed as a directed weighted graph $G = (V, E, d)$, and that our traveler wishes to drive across at least k roads from the subset $R \subseteq E$ of "amazing" roads. Furthermore, assume that the traveler starts and ends at her home $h \in V$. You may also assume that the traveler is fine with repeating roads from R, i.e. the k roads chosen from R need not be unique.

Design an efficient algorithm to solve this problem. Provide a 3-part solution with runtime in terms of $n = |V|$, $m = |E|$, k.

Hint: Create a new graph G' based on G such that for some s', t' in G', each path from s' to t' in G' corresponds to a path of the same length from h to itself in G containing at least k roads in R. It may be easier to start by trying to solve the problem for $k = 1$.

Solution: Main idea:

We want to build a new graph G' such that we can apply Dijkstra's algorithm on G' to solve the problem.

We'll start by creating $k+1$ copies of G. Call these $G_0, G_1, \ldots G_k$. These copies include all the edges and vertices in G, as well as the same weights on edges. Let the copy of v in G_i be denoted by v_i. For each road $(u, v) \in R$, we also add the edges $(u_0, v_1), (u_1, v_2), \ldots (u_{k-1}, v_k)$, with the same weight as (u, v). The intuition behind creating these copies is that each time we use an edge in G' corresponding to an edge in R, we can advance from one copy of G to the next, and this is the only way to advance to the next copy. So if we've reached v_i from h_0, we know we must have used (at least) i edges in R so far.

Now, consider any path p' from h_0 to h_k in G'. If we take each edge (u_i, v_i) or (u_i, v_{i+1}) and replace it with the corresponding edge (u, v) in G, we get a path p in G from h to itself. Furthermore, since the path goes from h_0 to h_k, it contains k edges of the form (u_i, v_{i+1}), where (u, v) is an edge in R. So, p will contain at least k edges in R.

Our algorithm is now just to create G' as described above, and find the shortest path p' from h_0 to h_k using Dijkstra's, and then output the corresponding path p in G.

Correctness:

Assume there is a valid path p in G that is shorter than the one produced by this algorithm. Consider the equivalent path p' in G' formed by modifying the path to go to the next copy of G whenever an edge of R is crossed. Since p is valid, p' must go from h in G_0 to h in G_k. But then p' would be a shorter path in G' than the one produced by Dijkstra's, which is a contradiction.

Runtime:

Since G' includes $k+1$ copies of G, Dijkstra's algorithm will run in time $O((km+kn)\log(kn))$. Since $k \leq m$ and $\log m = O(\log n)$, the runtime can be simplified to $O(k(m+n)\log n)$.

5 Pattern Matching

Consider the following string matching problem:

Input:

- A string g of length n made of 0s and 1s. Let us call g, the "pattern".
- A string s of length m made of 0s and 1s. Let us call s the "sequence".
- Integer k

Goal: Find the (starting) location of all length n-substrings of s which match g in at least $n-k$ positions.
Example: Using 0-indexing, if $g = 0111$, $s = 01010110111$, and $k = 1$ your algorithm should output 0,2,4 and 7.

(a) Give a $O(nm)$ time algorithm for this problem.

Solution:

For each of $i \in \{0, 1, \ldots, m-n\}$ starting points in s, check if the substring $s[i : i+n-1]$ differs from g in at most k positions. The check takes $O(n)$ time at each of $O(m)$ starting points, so the time complexity is $O(mn)$.

We will now design an $O(m \log m)$ time algorithm for the problem using FFT. *Pause a moment here to contemplate how strange this is. What does matching strings have to do with roots of unity and complex numbers?*

(b) Devise an FFT based algorithm for the problem that runs in time $O(m \log m)$. Write down the algorithm, prove its correctness and show a runtime bound.

Hint: On the example strings g and s, the first step of the algorithm is to construct the following polynomials

$$0111 \to 1 + x + x^2 - x^3$$
$$01010110111 \to -1 + x - x^2 + x^3 - x^4 + x^5 + x^6 - x^7 + x^8 + x^9 + x^{10}$$

Solution:
Main Idea:
Let g' be g with 0's replaced by -1's. Let $p_1(x) = a_0 + a_1 x + \cdots + a_{n-1}x^{n-1}$ where $a_d = g'(n-d-1)$ for all $d \in \{0, 1, \ldots, n-1\}$. Similarly, let $p_2(x) = b_0 + b_1 x + \cdots + b_{m-1}x^{m-1}$ where $b_d = s'(d)$ for all $d \in \{0, 1, \ldots, m-1\}$, where again s' is just s with 0's replaced by -1's. Notice $p_1(x)$ is reversed in the sense that the coefficients are in opposite order of the bits in g.

Now consider $p_3(x) = p_1(x) \times p_2(x) = c_0 + c_1 x + \cdots$, the coefficient of x^{n-1+j} in $p_3(x)$ is $c_{n-1+j} = \sum_{i=0}^{n-1} a_{n-1-i} b_{j+i} = \sum_{i=0}^{n-1} g'(i)s'(j+i)$ for any $j \in \{0, 1, \ldots, m-n\}$, which is exactly the dot product of the substring in s' starting at index j and the string g'. If these strings differ in at most k positions, then this dot product will be at least $n - 2k$. Thus all we need is to compute $p_3(x)$, and output all the j's between 0 and $m-n$ such that $c_{n-1+j} \geq n - 2k$.

Proof of Correctness:
Consider the mapping $\Phi : \{0, 1\} \to \{+1, -1\}$ defined as:

$$0 \to -1 \qquad\qquad 1 \to +1$$

Going along with the hint, construct the polynomial

$$G(x) = \sum_i G_i x^i$$

from the pattern $g[0 : n-1]$, by setting $G[i] = \Phi(g[n-1-i])$.

Similarly, construct the polynomial

$$S(x) = \sum_i S_i x^i$$

from the pattern $s[0 : m-1]$ by setting $S[i] = \Phi(s[i])$.

Now consider the product of these two polynomials $G(x)$ and $S(x)$,

$$\begin{aligned} G(x) \cdot S(x) &= \left(\sum_{i=0}^{n-1} G[i]x^i\right) \cdot \left(\sum_{j=0}^{m-1} S[j]x^j\right) \\ &= \sum_{r=0}^{m+n-2} x^r \cdot \left(\sum_{\ell=0}^{r} G[\ell] \cdot S[r-\ell]\right) \end{aligned}$$

In the last step, we are just writing out the coefficients of polynomial multiplication. Now, let us look at the coefficients a little bit more carefully. Let us start with coefficient of x^{n-1}, i.e., $r = n-1$,

$$\begin{aligned} \text{Coefficient of } x^r &= \sum_{\ell=0}^{n-1} G[\ell] \cdot S[r-\ell] \\ &= G \cdot S[r-n+1 : r]. \end{aligned}$$

This is the dot product of $\Phi(g)$ and $\Phi(s[: n])$ when $r = n-1$. More generally, the coefficient of x^r is the dot product of $\Phi(g)$ and $\Phi(s[: r+1])$. Note that if these strings differ in at most k positions, then this dot product will be at least $n - 2k$. Thus all we need is to compute $p_3(x)$, and output all the j's between 0 and $m-n$ such that $c_{n-1+j} \geq n - 2k$.

Runtime Analysis: The computation takes a FFT, a point-wise product, an inverse FFT, and a linear scan of the coefficients. The running time of this algorithm, $O(m \log m)$, is dominated by the FFT and inverse FFT steps, which each take $O(m \log m)$ time. The point-wise product and search for $c(i) \geq n - 2k$ each take $O(m)$ time.

(c) (Extra Credit) Often times in biology, we would like to locate the existence of a gene in a species' DNA. Of course, due to genetic mutations, there can be many similar but not identical genes that serve the same function, and genes often appear multiple times in one DNA sequence. So a more practical problem is to find all genes in a DNA sequence that are similar to a known gene.

This problem is very similar to the one we solved earlier, the string s is complete sequence and the pattern g is a specific gene. We would like to find all locations in the complete sequence s, where the gene g appears, but for k modifications.

Except in genetics, the strings g and s consist of one of four alphabets $\{A, C, T, G\}$ (not 0s and 1s). Can you devise an $O(m \log m)$ time algorithm for this modified problem?
Solution:

Solution 1: Map A to 0000, B to 1100, C to 1010 and G to 1001. Note that every pair of letters are different by two indices. Convert the bitstrings into the appropriate polynomials in vector form and run the algorithm from part (b). Instead of using a $n-2k$ threshold to find the appropriate starting indices, however, we will find all indices with values greater than or equal to $4n - 4k$.

If there is a mismatch between two letters, the dot product between the vector representations will be 0. Otherwise, the dot product between the vector representations of the same numbers are 4. Hence, for a substring with k mismatches from the pattern, the resulting dot product will be $4n - 4k$.

Solution 2: For both s and g, choose a single letter, say "A", to be 1. Convert all letters other than "A" to be 0. Reverse g, and run fast polynomial multiplication on these new vectors.

Repeat the above steps, choosing a different letter in the first step. Now, we have four vectors. For the vector where we chose "A" in the first step, the number at index i represents the number of matches for only "A". That also follows for the three remaining vectors.

Now, we can perform elementwise addition on the four vectors. Call the final vector sum v. The value at index i of v represents the number of total matches. Hence, we output all the i's between 0 and $m - n$ such that $c_{n-1+j} \geq n - k$.