CS 170 Spring 2016 — Efficient Algorithms and Intractable Problems — Alessandro Chiesa and Umesh Vazirani

HW 3

Due February 8, 11:59pm

1. (15 pts.) Inverse FFT

The fast fourier transform can be used to efficiently multiply polynomials in the following way:

Input: Coefficients of two polynomials $A(x)$ and $B(x)$ each of degree at most d

Output: The coefficients of their product $C(x) = A(x)B(x)$

1. Find some power of two $n \geq 2d+1$
2. Use the FFT to evaluate A and B at the n^{th} roots of unity $1, \omega, \omega^2, \ldots, \omega^{n-1}$
3. Evaluate C at each of the n^{th} roots of unity: $C(1) = A(1)B(1), C(\omega) = A(\omega)B(\omega), \ldots, C(\omega^{n-1}) = A(\omega^{n-1})B(\omega^{n-1})$
4. Recover the coefficients of C from its evaluations on each of the n^{th} roots of unity

In class we saw how to do the second step efficiently using the FFT algorithm. In this question, we will see how to do the last step efficiently. For the remainder of this problem, ω will denote the first n^{th} root of unity. Recall that in class we defined M_n to be the following matrix:

$$M_n = \begin{bmatrix} 1 & 1 & 1 & \ldots & 1 \\ 1 & \omega & \omega^2 & \ldots & \omega^{n-1} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \omega^{n-1} & \omega^{2(n-1)} & \ldots & \omega^{(n-1)(n-1)} \end{bmatrix}$$

For the rest of this problem we will refer to this matrix as $M_n(\omega)$ rather than M_n.

(a) Define

$$M_n(\omega^{-1}) = \begin{bmatrix} 1 & 1 & 1 & \ldots & 1 \\ 1 & \omega^{-1} & \omega^{-2} & \ldots & \omega^{-(n-1)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \omega^{-(n-1)} & \omega^{-2(n-1)} & \ldots & \omega^{-(n-1)(n-1)} \end{bmatrix}$$

Show that $(1/n)M_n(\omega^{-1})$ is the inverse of $M_n(\omega)$, i.e. show that

$$(1/n)M_n(\omega^{-1})M_n(\omega) = I$$

where I is the $n \times n$ identity matrix– the matrix with all ones on the diagonal and zeros everywhere else.

(b) Suppose we have a polynomial $C(x)$ of degree at most $n-1$ and we know the values of $C(1), C(\omega), \ldots, C(\omega^{n-1})$. Explain how we can use $M_n(\omega^{-1})$ to find the coefficients of $C(x)$.

(c) Show that $M_n(\omega^{-1})$ can be broken up into four $n/2 \times n/2$ matrices in almost the same way as $M_n(\omega)$. Specifically, suppose we rearrange columns of M_n so that the columns with an even index are on the left side of the matrix and the columns with an odd index are on the right side of the matrix (where the indexing starts from 0). Show that after this rearrangement, $M_n(\omega^{-1})$ has the form:

$$\begin{bmatrix} M_{n/2}(\omega^{-1}) & \omega^{-j}M_{n/2}(\omega^{-1}) \\ M_{n/2}(\omega^{-1}) & -\omega^{-j}M_{n/2}(\omega^{-1}) \end{bmatrix}$$

As in class, the notation $\omega^{-j}M_{n/2}(\omega^{-1})$ is used to mean the matrix obtained from $M_{n/2}(\omega^{-1})$ by multiplying the j^{th} row of this matrix by ω^{-j} (where the rows are indexed starting from 0).

As we saw in class, the result from part (c) means that we can efficiently multiply vectors by the matrix $M_n(\omega^{-1})$.

2. (15 pts.) Triple Sum

Design an efficient algorithm for the following problem. We are given an array $A[0..n-1]$ with n elements, where each element of A is an integer in the range $0 \le A[i] \le 10n$. We are also given an integer t. The algorithm must answer the following yes-or-no question: do there exist indices i, j, k, not necessarily distinct, such that $A[i] + A[j] + A[k] = t$?

Design an $O(n \log n)$ time algorithm for this problem.

Hint: Consider the polynomial $p(x) = x^{A[0]} + x^{A[1]} + \cdots + x^{A[n-1]}$.

3. (10 pts.) Depth

Let G be a connected undirected graph. You perform a depth-first search starting from some vertex $r \in V$. For each node v in the DFS tree, let $d(v)$ denote the depth of v in the DFS tree. In particular, the root r has depth 0; r's children have depth 1; r's grandchildren have depth 2; and so on.

Describe how to compute $d(v)$ for each vertex $v \in V$, in linear time. (You can assume the vertices are numbered $0..n-1$, so your goal is to build and initialize an array $\mathtt{d}[0..n-1]$ so that $\mathtt{d}[v]$ holds the depth of v.)

4. (20 pts.) Disrupting a network of spies

Let $G = (V,E)$ denote the "social network" of a group of spies. In other words, G is an undirected graph where each vertex $v \in V$ corresponds to a spy, and we introduce the edge $\{u,v\}$ if spies u and v have had contact with each other. The police would like to determine which spy they should try to capture, to disrupt the coordination of the group of spies as much as possible. More precisely, the goal is to find a single vertex $v \in V$ whose removal from the graph splits the graph into as many different connected components as possible. This problem will walk you through the design of a linear-time algorithm to solve this problem. In other words, the running time will be $O(|V| + |E|)$.

In the following, let $f(v)$ denote the number of connected components in the graph obtained after deleting vertex v from G. Also, assume that initial graph G is connected (before any vertex is deleted) and is represented in adjacency list format.

Prove your answer to each part (some parts are simple enough that the proof can be a brief justification; others will be more involved).

(a) Perform a depth-first search starting from some vertex $r \in V$. How could you calculate $f(r)$ from the resulting depth-first search tree in an efficient way?

(b) Suppose $v \in V$ is a node in the resulting DFS tree, but v is not the root of the DFS tree (i.e., $v \neq r$). Suppose further that no descendant of v has any non-tree edge to any ancestor of v. How could you calculate $f(v)$ from the DFS tree in an efficient way?

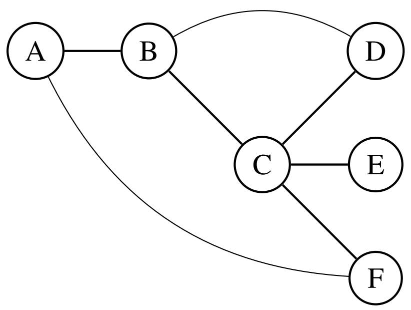
(c) Definition: If w is a node in the DFS tree, let $up(w)$ denote the depth of the shallowest node y such that there is some graph edge $\{x,y\} \in E$ where either x is a descendant of w or $x = w$.

Now suppose v is an arbitrary non-root node in the DFS tree, with children $w_1,\dots,w_k$. Describe how to compute $f(v)$ as a function of k, $up(w_1),\dots,up(w_k)$, and $d(v)$. (Recall that we have an algorithm for $d(v)$ from Question 3, so in this question you can simply call $d(v)$ as a black-box.)

Hint: Think about what happened in part (b); think about what changes when we can have non-tree edges that go up from one of v's descendants to one of v's ancestors; and think about how you can detect it from the information provided.

(d) Design an algorithm that runs in linear time and computes $up(v)$ for all vertices $v \in V$.

(e) Describe how to compute $f(v)$ for each vertex $v \in V$, in linear time.



This is a sample group of spies, with the A the root vertex and the DFS proceeding in alphabetical order (the DFS tree has bolder straight edges).

In this group, $up(A) = up(B) = up(c) = up(F) = 0$.
$up(D) = 1$; $up(E) = 2$.

5. (20 pts.) Satisfying assignment

In the following problem, if x is a boolean variable, then $\bar{x}$ indicates its negation (i.e. if x is true then $\bar{x}$ is false and vice-versa). The negation of the negation of a variable is equivalent to the variable. A boolean variable and its negation are both referred to as literals.

Suppose we are given a list of boolean variables $x_1, \ldots, x_n$ and a list of implications between literals: i.e. constraints of the form $x_i \Rightarrow x_j$ (which means that if x_i is assigned the value `true` then x_j must be assigned the value `true`), $\overline{x_i} \Rightarrow x_j$ (which means that if x_i is assigned the value `false` then x_j must be assigned the value `true`) and $x_i \Rightarrow \overline{x_j}$ (which means that if x_i is assigned the value `true` then x_j must be assigned the value `false`). We want to either return an assignment of `true` or `false` to each variable so that all the constraints are satisfied or report that no such assignment is possible. (The usual name for this problem is 2SAT. Later in the course we will see a related problem called 3SAT that is believed to be much harder.)

In this problem we will find an efficient way of solving this problem by reducing it to the problem of finding the strongly connected components of a directed graph. If there are n variables and m constraints then we will construct a directed graph $G = (V, E)$ as follows:

- G has $2n$ vertices: one for each variable and its negation (i.e. one for x_i and one for $\overline{x_i}$)
- G has $2m$ edges: for every implication between literals, $\alpha \Rightarrow \beta$, we introduce two edges, one from α to β and one from the negation of β to the negation of α. So for the constraint $\overline{x_i} \Rightarrow x_j$ we would add an edge from $\overline{x_i}$ to x_j and an edge from $\overline{x_j}$ to x_i. (It is a good idea to think about why both edges are necessary)

(a) Show that if G has a strongly connected component containing both x_i and $\bar{x}_i$ for some x_i then there is no satisfying assignment.

(b) Now show the converse of part (a): namely that if none of G's strongly connected components contain both a literal and its negation then there must be a satisfying assignment. (Hint: Assign values to the variables as follows: repeatedly pick a sink strongly connected component in G. Assign `true` to all literals in the sink and `false` to all of their negations and then delete all of these. Show that this ends up discovering a satisfying assignment.)

(c) Conclude that there exists a linear time algorithm for finding a satisfying assignment.

6. (20 pts.) Travel planning

You are given a set of n cities and an $n \times n$ matrix M, where $M(i, j)$ is the cost of the cheapest direct flight from city i to city j, or ∞ if there is no such flight. (Flights cost a nonnegative integer number of dollars.) You live in city A and want to find the cheapest and most convenient route to city B. In particular, if there are multiple routings you could take with the same total cost, you'd like the routing with the fewest connections (ie, the smallest number of flights). Design an efficient algorithm to solve this problem. You can assume that the best route will require fewer than 1000 flights.

7. (5 pts.) Optional bonus problem: Filling in matrix entries

(This is an *optional* challenge problem. Only solve it if you want an extra challenge.)

Consider the following problem: We are given an $n \times n$ matrix where some of the entries are blank. We would like to fill in the blanks so that all pairs of columns of the matrix are linearly dependent (two vectors v and u are linearly dependent if there exists some constant c such that $cv = u$) or report that there is no such way to fill in the blanks. Formulate this as a graph problem and design an $O(n^2)$ algorithm to solve it. You may assume that all the non-blank entries in the matrix are nonzero.