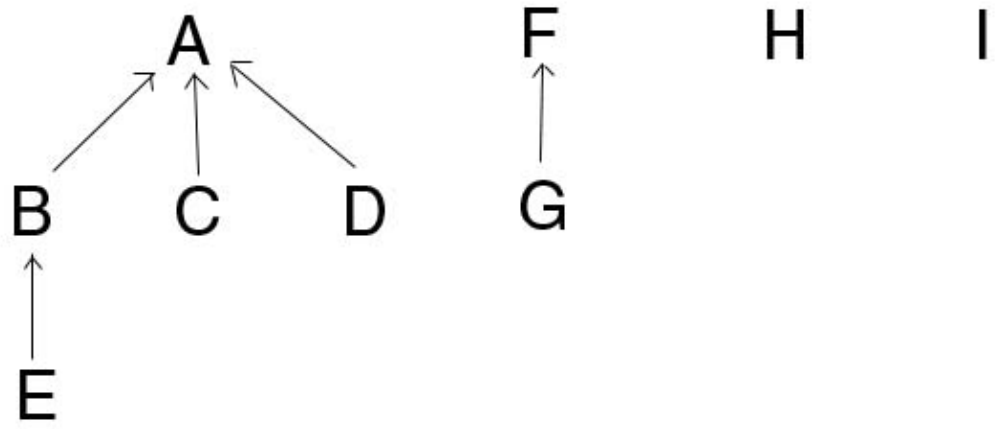
CS 170, Fall 1999
Midterm II

1. (15 points) The following is a forest formed after some number of UNIONs and FINDs starting with the disjoint sets A, B, C, D, E, F, G, H, and I. Both union-by-rank and path-compression were



used.

(a) Starting with the forest above, we now call the following routines in order:

FIND(B), UNION (G, H), UNION(A, G), UNION(E,I)

Draw the resulting forest using both union-by-rank and path-compression. In case of tie during UNION, assume that UNION will put the lexicographically first letter as root.

(b) Starting with the disjoint sets A, B, C, D, E, F, G, H, and I, give a sequence of UNIONs and FINDs that results in the forest shown at the top of the page. In case of tie during union, assume that UNION will put the lexicographically first letter as root.

2. (25 points) Let

$$p(x)=\sum_{i=0}^{n} p_i x^i \quad \text{and} \quad q(x)=\sum_{i=0}^{m} q_i x^i$$

be polynomials of degrees n and m, respectively, where n and m can be integers such that $n \geq m$

(a) Give an algorithm using the FFT that computes the coefficients of $r(x) = p(x) \cdot q(x)$. How many arithmetic operations does it perform, as a function of m and n? Your answer can use $O()$ notation.

(b) Give an algorithm *not* using the FFT that computes the coefficients of $r(x) = p(x) \cdot q(x)$. How many arithmetic operations does it perform, as a function of m and n?

(c) Combine the above algorithms to give the fastest possible algorithm depending on m and n. How many arithmatic operations does it perform? Roughly how small (in an O() sense) does m have to be for the non-FFT algorithm to be at least as fast as the FFT algroithm?

3. (25 points) Giaven a set $S = \{s_1, s_2, ..., s_n\}$ of n nonnegative integers, and a positive integer T, find a subset of S that adds up to T. Use dynamic programming; your solution should not have a cost growing like 2^n.

You should (1) Formulate your algorithm recursively, (2) describe how it would be implemented in a bottom-up iterative manner, (3) give a bound on its running time in terms of n and T, and (4) give a short justification of both the correctness of the algorithm and its running time.

4. (15 points) True or False? No explination required, except for partial credit. Each correct answer is worth 1 point, but 1 point will be *subtracted* for each wrong answer, so answer only if you are reasonably certain.

(a) If we can square a general n-by-n matrix in $O(n^d)$ time, where $d \geq 2$, then we can multiply any two n-by-n matrices in $O(n^d)$time.

(b) If the frequencies of the individual characters in a file are unique, the file's Huffman code is unique.

(c) Huffman coding can compress any file.

(d) The solution to the recurrence $T(n) = 2T(n/2) + O(n \log n)$ is $T(n) = \Theta(n(\log n)^2)$.

(e) log*log n = O(loglog*n)

(f)In Union-Find (with union-by-rank and path compression, any union only takes $O(\log^* n)$ time, where n is the number of nodes.

(g) In Union-Find data structure with union-by-rank but no path compression, m union and finds takes $O(m \log m)$ time.

(h) If path compression is not used, but union-by-rank is used, it is possible to arrange m LINK and FIND operations so that it takes *(m log m) time.

(i) If w is a complex n-th root of unity, then $|w| = 1$ where $|w|$ is the absolute value of w.

(j) If we want to use FFT to multiply two polynomials of degree $n = 2^m$, we need to run the FFT on vectors of length $2n$.

(k) The values of a degree n polynomial at n+2 distinct points determines its coefficients uniquely.

(l) To find an optimal way to multiply 6 matrices A1*A2*...*A6, we can find an optimal way to multiply A1*A2*A3, and to multiply A4*A5*A6 and combine the results.

(m) Floyd-Warshall algorithm works with negative edge weights when there are no negative cycles.

(n) Floyd-Warshall algorithm is always asymptotically faster than running Dijkstra n times where n is the number of verticies.

(o) You wrote your name and your TA's name on the first page.