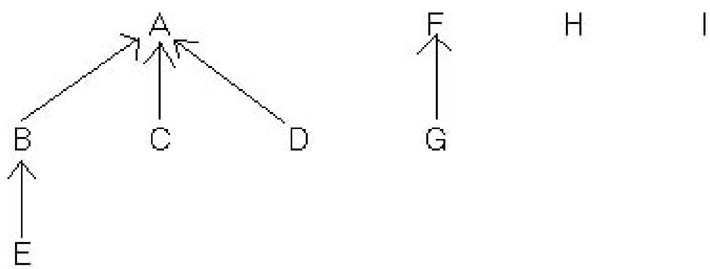
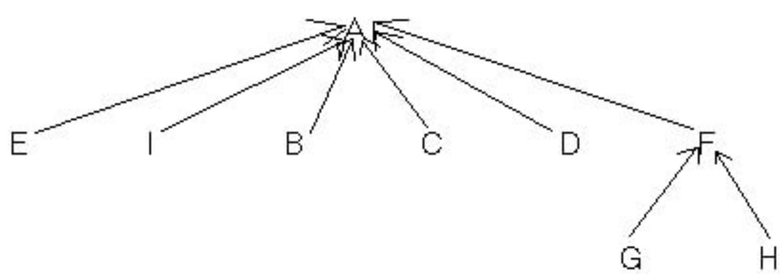
CS 170, Fall 1999
Midterm 2 with Solutions
Professor Demmel

Problem #1

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:
Answer:



(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 a tie during union, assume that UNION will put the lexicographically first letter as a root.
Answer: One solution is
UNION (F,G), UNION (A,C), UNION (B,E), UNION (B,D), UNION (D,A)

Problem #2

2) (25 points) Let p(x) = SUM_FROM_i=0_to_n (p_sub_i*x^i) and q(x) = SUM_FROM_i=0_to_m (q_sub_i*x^i) be polynomials of degrees n and m, respectively, where n and m can be any integers such that n>=m.

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

Answer: (1) Round up n+m+1 to the nearest power of 2, ie find the smallest k such that 2^k>=n+m+1: k = CEILING_OF(LOGbase2(n + m + 1)). (2) Pad the vectors [p_sub_0,...,p_sub_n] and [q_sub_0,...,q_sub_n] with enough zeroes to make vectors p_prime and q_prime of length 2^k. (3) Compute p_hat = FFT(p+prime)

and q_hat = FFT (q_prime). The cost is 3*k*2^k complex operations, or 10*k*2^k real operations. (4) Multiply (r_hat)_sub_i = ((p_hat)_sub_i)* ((q_hat)_sub_i)for i = 0, ..., (2^k)-1. The cost is 2^k complex operations, or 6*(2^k) real operations. (5) Compute r_prime = invFFT(r_hat) and extract the leading n+m+1 entries. The cost is 1.5*k*2^k complex operations or 5*k*2^k real operations.
The total cost is (4.5k + 1)2^k complex arithmetic operations, or (15k+6)2^k real arithmetic operations, or more simply O(n*log n) operations.

(b) Give an algorithm NOT using the FFT that computes the coefficients of r(x) = p(x)DOTq(x). How many arithmetic operations does it perfrom as a function of m and n?
Answer: For j = 0 to m+n compute r_sub_j = SUM_FROM_i=(max(0,j-m))_to_(min(j,n)) [p_sub_i*q_sub_j-i]. The cost is about 2mn complex operations, or 8mn real operations, or more simply, O(mn) operations.

(c) Combine teh above algorithms to give the fastest possible algorithm depending on m and n. How many arithmetic operations does it perform? Roughly how small (in a O() sense) does m have to be for the non-FFT algorithm to be at least as fast as the FFT algorithm?
Answer: If (15k + 6)2^k <= 8mn use the FFT based algorithm, else the non-FFT based algorithm. Or more roughly, if log_base2_of_n < m, then use the FFT based algorithm.)

Problem #3

3) (25 points) Given a set S = {s_sub_1, ... , s_sub_n} of n nonnegative intergers, 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 of 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 cound on its running time in tersm of n and T and (4) give a short justification of both the correctness of the algorithm and its running time.

Answer: Define AddUp(T_prime,i) to be True is a subset of {s_sub_1, ... , s_sub_n} adds up to T_prime <= T, and False otherwise. Clearly AddUp(T_prime,1) = True if s_sub_1 = T_primt and False otherwise, and for larger i AddUp(T_prime,i) = AddUp(T_prime,i-1) v AddUp(T_prime - s_sub_i,i-1). AddUp can be computed by filling in a T-by-n table of all possible values of AddUp(T_prime,i) for 1<= T_prime <= T and 1<=i<=n, first filling in all values of AddUp(T_prime,1) and then AddUp(T_prime,i) for i = 2 to n, at a cost of O(1) per table entry, and O(Tn) overall. Finally,one inspects AddUp(T,n), which is true if and only if the problem can be solved. Another T-by-n table Set where Set(T_prime, i) records which of AddUp(T_prime,i-1) or AddUp(T_prime - s_sub_i,i-1) is true (pick arbitrarily if both are true) will let the actual set adding up to T be reconstructed.

Problem #4

4) (15 points) True or False?? No explanation 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>=2, then we can multiply any two n-by-n matrices in O(n^d) time
Answer: TRUE
(b) If the frequencies of the individual characters in a file are unique, the file's Huffman code is unqiue.
Answer: FALSE
(c) Huffman coding can compress any file
Answer: FALSE
(d) The solution to the recurrance T(n)=2T(n/2)+O(n*log_n) is T(n) = Theta(n(log_n)^2).
Answer: TRUE

(e) log* log_n = O(loglog* n)
Answer: FALSE
(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.
Answer: FALSE
(g) In Union-Find data structure with union-by-rank but no path compression, m union and finds takes O(m log m) time.
Answer: TRUE
(h) If the compression is not used, but union-by-rank is used, it is possible to arrange m LINK and FIND operation so that is takes Omega(m log m) time.
Answer: TRUE
(i) If w is a complex n-th root of unity, then |w| = 1, where |w| is the absolute value of w.
Answer: TRUE
(j) If we want to ise FFT to multiply two polynomials of degree n = 2^m, we need to run the FF on vectors of length 2n.
Answer: FALSE
(k) The value of a degree n polynomials at n+2 distinct points determines its coefficients uniquely.
Answer: TRUE
(l) To find a 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 result.
Answer: FALSE
(m) Floyd-Warhsall algorithm works with negative edge weights when there are no neagtive cycles.
Answer: TRUE
(n) Floyd-Warshall algorithm is always asymptotically faster than running Dijkstra n times, where n is te number of vertices
Answer: FALSE
(o) You wrote your name and your TA's name on the first page
