CS 170 **First Midterm** **4 Oct 2011**

Name (1 pt): _____

SID (1 pt): _____

Section Number, e.g. 101 (1 pt): _____

Name of Neighbor to your left (1 pt): _____

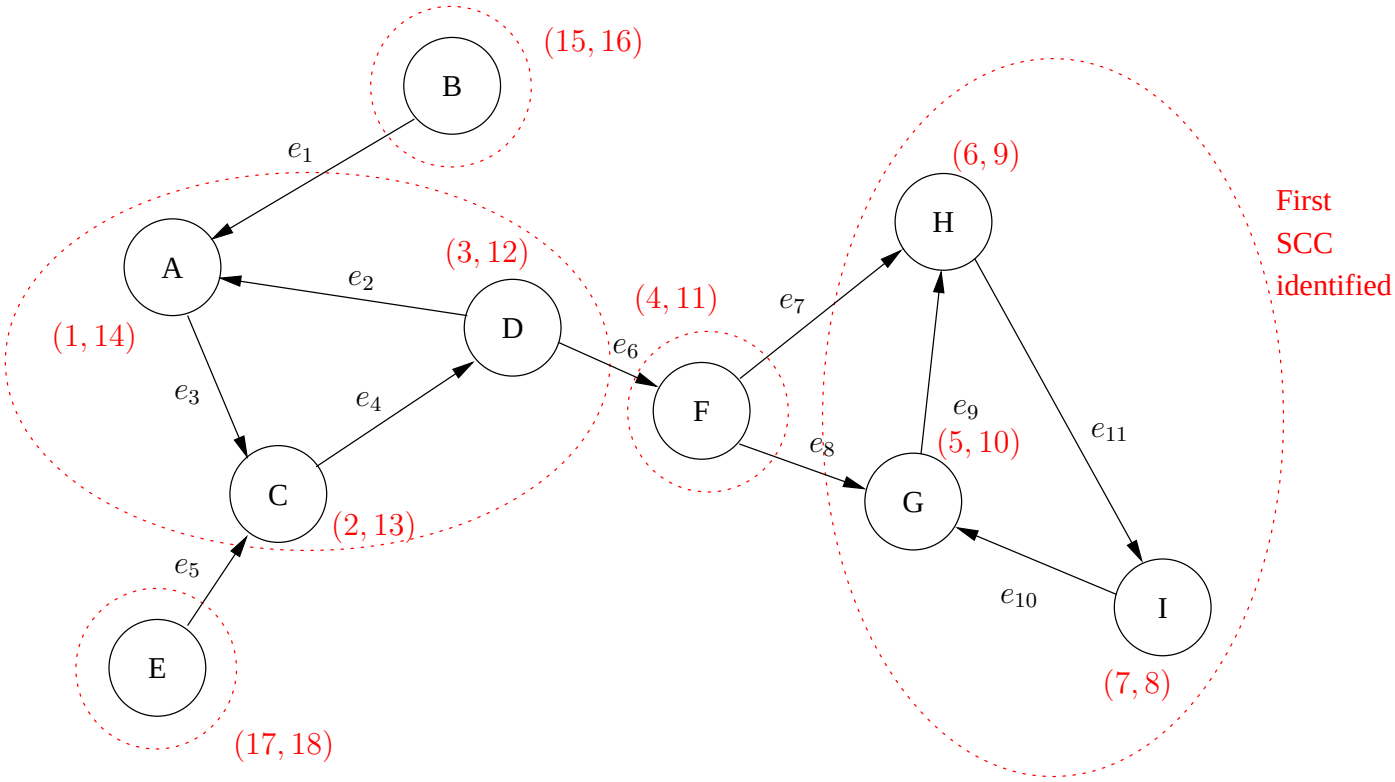
Name of Neighbor to your right (1 pt): _____

Instructions: This is a closed book, closed calculator, closed computer, closed network, open brain exam, but you are permited a 1 page, double-sided set of notes, large enough to read without a magnifying glass.
You get one point each for filling in the 5 lines at the top of this page.
Write all your answers on this exam. If you need scratch paper, ask for it, write your name on each sheet, and attach it when you turn it in (we have a stapler).

1	
2	
3	
4	
5	
Total	

1 , solutions

Question 1: Directed Graph (15 points). Given is the following graph



(a) [3 points] Add pre and post numbers to the graph as generated during a depth-first-search. Whenever the depth-first-search has a choice of vertices to explore next, always pick the one that is alphabetically first.

(b) [4 points] Classify the edges in the following categories. Use the edge labels from the graph.

forward edges	back edges	cross edges	tree edges
e_7	e_2	e_1	e_3
	e_{10}	e_5	e_4
			e_6
			e_8
			e_9
			e_{11}

(c) [3 points] Circle all strongly connected components (SCCs) in the graph.

(d) [3 points] Indicate in the graph which SCC is detected first if you run the strongly connected components algorithm on the graph.

(e) [2 points] Write out all linearizations of the DAG of SCCs, naming each vertex by listing the names of the members of the corresponding SCC.
{B},{E},{A,C,D},{F},{G,H,I}
{E},{B},{A,C,D},{F},{G,H,I}

Question 2: Divide and Conquer (20 points).

Alice and Bob play a game where Alice picks a number $x \in \{1, 2, 3, \ldots, n\}$, and Bob's goal is to figure out what x is by making guesses. After Bob's first guess, Alice says nothing, but after his second (and subsequent) guesses, Alice tells him whether his new guess is closer to x than his previous guess (hotter), further from x than his previous guess (colder), or the same distance from x. More precisely, if Bob's $(i-1)^{\text{st}}$ guess is g_{i-1} and his i^{th} guess is g_i, then Alice says "hotter" if $|g_i - x| < |g_{i-1} - x|$, "colder" if $|g_i - x| > |g_{i-1} - x|$, or "same distance" if $|g_i - x| = |g_{i-1} - x|$. Bob's guesses *do not* need to be in $\{1, 2, 3, \ldots, n\}$.

Complete the pseudocode below to get an algorithm that allows Bob to discover x using at most $\log_2(n) + O(1)$ guesses.

Hint: The idea is to maintain variables ℓ and u such that x always lies in the interval $\ell \le x \le u$ (this is a loop invariant), and to always choose our next guess so the size of the interval (namely $u - \ell + 1$) gets roughly halved. Initially, $\ell = 1$ and $u = n$. We'll use $g_1 = 0$ as the first guess. Note that then $g_2 = n+1$ would be a good choice for the second guess, because "hotter" would mean x is in the top half, and "colder" would mean x is in the bottom half. You can choose to halt the algorithm at any time if you discover what x is. In the code it might be helpful to distinguish when i is odd versus even, because the guesses should alternate between below the interval and above the interval.

(a) [5 points] Fill in the 1st box, which should compute the new guess g_i in terms of ℓ, u, and the previous guess g_{i-1}.

The idea is to make the new guess be the same distance from the center of the interval as the previous guess, but on the other side. We also maintain the invariant that when i is odd, g_i is below the interval, and when i is even, g_i is above the interval.

```
if i is odd then g_i = ℓ − (g_{i−1} − u)
if i is even then g_i = u + (ℓ − g_{i−1})
```

(b) [5 points] Fill in the 2nd box, which should update ℓ and/or u.

```
if i is odd then u = ⌈(ℓ+u)/2⌉ − 1        // ⌊(ℓ+u)/2⌋ also acceptable
if i is even then ℓ = ⌊(ℓ+u)/2⌋ + 1       // ⌈(ℓ+u)/2⌉ also acceptable
```

(c) [5 points] Fill in the 3rd box, which should update ℓ and/or u.

Same as part (b), but the two updates are swapped.

```
if i is odd then ℓ = ⌊(ℓ+u)/2⌋ + 1       // ⌈(ℓ+u)/2⌉ also acceptable
if i is even then u = ⌈(ℓ+u)/2⌉ − 1       // ⌊(ℓ+u)/2⌋ also acceptable
```

(d) [5 points] Fill in the 4th box.

If g_{i-1} and g_i are the same distance from x, then x is exactly halfway inbetween them, provided $g_{i-1} \neq g_i$ (which always holds since one of them is below the interval and the other is above the interval).

```
output (g_{i−1}+g_i)/2          // (ℓ+u)/2 is the same thing
halt
```

```
ℓ = 1
u = n
i = 1
g_1 = 0                 // the initial guess
while (ℓ < u) {
    i++
    [                                                    ]
    if "hotter" then
        [                                                ]
    elseif "colder" then
        [                                                ]
    elseif "same distance" then
        [                                                ]
}
output ℓ
```

Question 3: Fast Fourier Transform (15 points)
Let $p(x)$ and $q(x)$ be polynomials of degree $2n$ and $2m+1$, respectively. $p(x)$ only has terms with even exponents (x^0, x^2, x^4 , ...), and $q(x)$ only has terms with odd exponents. Show how to multiply these polynomials efficiently using only FFTs of size $n+m+1$.

Your answer should detail any pre- and post-processing steps you take before and after calling the normal FFT. You do not need to reprove the correctness of FFT. You should justify that any work you do besides the FFTs does not take longer (asymptotically) than the FFTs.

Say

$$p(x) = p_0 + p_1x^2 + p_2x^4 + \cdots + p_nx^{2n} \cdots$$
$$q(x) = q_0x + q_1x^3 + q_2x^5 + \cdots + q_{m+1}x^{2m+1} \cdots$$

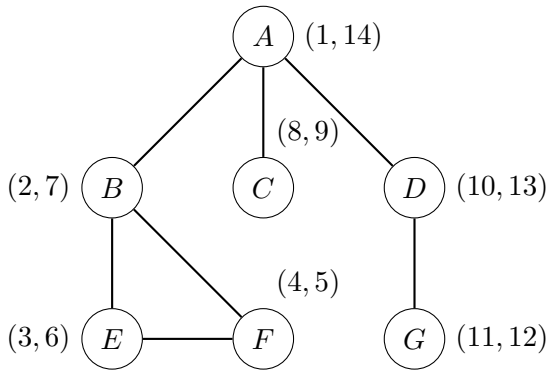
Let $q'(x) = \frac{1}{x}q(x)$. Then $p(x)q(x) = xp(x)q'(x)$. With respect to x^2, $p(x)$ and $q'(x)$ are polynomials of degree n and m, respectively, so we can use the normal FFT method to find their product. Their product, with respect to x^2, has degree $n+m$, so we take FFT of size $n+m+1$ of each polynomial, do a pairwise multiplication, and take the inverse FFT on the resulting value vector (also of size $n+m+1$) to get the coefficients of $p(x)q'(x)$ relative to x^2. We also still have to put the extra x term we factored out earlier, we can accomplish this in $O(n+m)$ time by iterating over polynomial and doubling and adding one to the degree of each term.

The preprocessing can be accomplished in linear time by simply copying the coefficient vectors into new vectors of half the size, skipping every other entry. FFT takes $O((n+m)\log(n+m))$ time, so this work does not add to the overall running time.

Question 4: Undirected Graphs (15 points).

(a) [5 points] Assume that the `pre` and `post` numbers (found by running a DFS) are represented as two arrays `vertex`$[t]$ and `start`$[t]$ for $1 \leq t \leq 2|V|$ (t for time), such that if `pre`$[v] = t$, then `vertex`$[t] = v$ and `start`$[t]$ = YES, and if `post`$[v] = t$, then `vertex`$[t] = v$ and `start`$[t]$ = NO.

For example, consider the following undirected graph, labelled with `pre` and `post` numbers:



Fill in the following table of `vertex` and `start`. You may write `Y` for YES and `N` for NO.

t	1	2	3	4	5	6	7	8	9	10	11	12	13	14
`vertex`	A	B	E	F	F	E	B	C	C	D	G	G	D	A
`start`	Y	Y	Y	Y	N	N	N	Y	N	Y	Y	N	N	N

(b) [10 points] Assume that the graph is connected and undirected, so there is only one DFS tree, whose root we call r. Design an efficient algorithm that takes arrays `vertex`[] and `start`[] as inputs, and returns a list of (or prints) all children of the root r. Note that the only information about the graph available to the algorithm are the arrays `vertex`[] and `start`[] and $|V|$.

Please state the main idea*, and write a* pseudocode*. Analysis of correctness or runtime is not required.*

Main Idea: Note that `pre`$[r] = 1$ and `post`$[r] = 2|V|$. If $|V|$ is not 1, then r has children, and its first child c_1 has `pre`$[c_1]$ = `pre`$[r] + 1$. In general, the i^{th} child has its `pre`$[c_i]$ = `post`$[c_{i-1}]+1$. Hence locate the first child at `vertex`$[2]$, with $2 =$ `pre`$[c_1]$. Once c_i is located at `pre`$[c_i]$ of `vertex`, iterate until c_i appears again on `vertex` at `post`$[c_i]$, then `post`$[c_i] + 1 =$ `pre`$[c_{i+1}]$ and so on, unless `post`$[c_i] + 1 =$ `post`$[r]$.

Pseudocode:

Algorithm 1: Children of Root

Data: numbers `vertex`$[t]$ and `start`$[t]$ for $1 \leq t \leq 2|V|$

```
1 last := 2;
2 t := 3;
3 while t ≤ 2|V| − 1 do
4     if vertex[t] = vertex[last] then
5         Report vertex[t];
6         t := t + 1;
7         last := t;
8     t := t + 1;
```

Correctness (not required): Say a range $[a, b]$ is a proper subrange of $[c, d]$ if $c < a < b < d$, and say the range of u is $\mathit{range}(u) := [\texttt{pre}[u], \texttt{post}[u]]$, then u is a children of r iff $\mathit{range}(u)$ is a proper subrange of $\mathit{range}(r)$, but $\mathit{range}(u)$ is not a proper subrange of $\mathit{range}(v)$ unless $v = r$.

Question 5: True/False (18 points). Circle your answer.

T or F: Suppose you write a parallel algorithm for the FFT by modifying the sequential algorithm discussed in class as follows: the two recursive calls to FFT are done in parallel, and all the iterations in the subsequent loop are done in parallel. Assuming you have enough processors, this algorithm runs in time $\Theta(\log n)$, where n is the input size.
True. The recurrence is $T(n) = T(n/2) + O(1) = O(\log n)$.

T or F: $2^{(2^{(3^n)})} = O(10^{(10^{(2^n)})})$.
False. Take $\log_2$ of both sides to get $2^{(3^n)}$ and $\log_2 10 \cdot 10^{(2^n)}$. Take $\log_2$ of both sides again to get 3^n and $\log_2 \log_2 10 + \log_2 10 \cdot 2^n$. The first expression can clearly be arbitrarily larger than the second expression.

T or F: Let F be the n-by-n Fourier transform matrix. Let $X = F \cdot F$ (matrix multiplication). Then $X(j,k) = n$ if $n|(j+k)$ and 0 otherwise. (We index X from $X(0,0)$ to $X(n-1, n-1)$.)
True. $X(j,k) = \sum_{m=0}^{n-1} F(j,m)F(m,k) = \sum_{m=0}^{n-1} \omega^{jm}\omega^{km} = \sum_{m=0}^{n-1} \omega^{(j+k)m} = n$ if $n|(j+k)$ so that $\omega^{(j+k)m} = 1$, and $= (1-\omega^{(j+k)n})/(1-\omega^{(j+k)}) = (1-1)/(1-\omega^{(j+k)}) = 0$ otherwise.

T or F: If $T(n) = T(n/2) + 3T(n/3) + n^2$ and $T(1) = 1$, then $T(n) = O(n^2 \log n)$.
True. Apply the Master Theorem to the upper bound $T(n) \le T(n/2) + 3T(n/2) + n^2 = 4T(n/2) + n^2$.

T or F: A directed graph of n vertices and m edges (with at most one edge connecting any pair of distinct vertices) has at least $\min(n, n-m+1)$ Strongly Connected Components.
True. With $m = 0$ or $m = 1$ edges, there are clearly n SCCs as the formula says. The question is asking how few SCCs you can have with $m > 1$ edges, or how many vertices can be put in a single SCC with $m > 1$ edges. The answer is to connect m vertices in a ring (one SCC), leaving $n - m$ other isolated vertices as SCCs, for a total of $n - m + 1$ SCCs when $m > 1$. The answer is vacuously true when $m > n$.

T or F: When doing DFS on an undirected graph, it is possible for some edge $\{w, v\}$ that $pre[v] < post[v] < pre[w] < post[w]$.
False. This is only possible for cross edges in directed graphs, which don't exist in undirected graphs. In particular, if v is visited before w and $\{w, v\}$ is an edge, then $pre[v] < pre[w] < post[w] < post[v]$, i.e. $\{w, v\}$ is a tree or back edge.