ALGORITHMS EXAMINATION
Department of Computer Science
New York University
December 17, 2007

- This examination is a *three hour* exam.
- All questions carry the same weight.
- Answer all of the following *six* questions. Please be aware that to pass this exam you need to provide good answers to several questions; it is not sufficient to obtain partial credit on each question.
- Please print your name and SID on the front of the envelope only (not on the exam booklets).
- Please answer each question in a *separate* booklet, and *number* each booklet according to the question.
- Read the questions carefully. Keep your answers legible, and brief but precise.
- Assume standard results, except where asked to prove them.

Good luck!

Question 1

Let $G = (V, E)$ be an undirected graph with vertex set V and edge set E. A partition of a graph is a partition of V into two disjoint sets, i.e., $V = V' \cup V''$ and $V' \cap V'' = \emptyset$. An edge is said to be *cut* by a partition, if one of its endpoint lies in V' and the other in V''. Show that there always exists a partition that cuts at least $|E|/2$ edges (i.e. half of the total number of edges in the graph).

Solution. A probabilistic argument runs as follows. Choose a random cut, i.e., for each vertex, assign it to V' with probability $1/2$. Now calculate the expected number of edges that cross the cut. For each edge, it is easy to calculate that it crosses the cut with probability $1/2$, and so by linearity of expectation, the expected number of edges that cross the cut is $|E|/2$. It follows that some partition must attain the bound $|E|/2$.

Go To the Next Page

Question 2

Let $\{x_1, x_2, \ldots, x_n\}$ be boolean variables. A *literal* is either a variable or its negation, i.e. x_i or $\overline{x}_i$. A *clause* is logical OR of one or more distinct literals. The size of a clause is the number of literals in it. A 2CNF formula ϕ is a collection of m clauses (possibly with repetition),

$$\phi = (C_1, C_2, \ldots, C_m),$$

where each C_i is of size at most two.

Let MAX-2SAT be the following decision problem: Given a pair (ϕ, k), where ϕ is a 2CNF formula with n variables and m clauses, and k is a positive integer such that $k \leq m$, decide whether there exists an assignment to the n boolean variables that satisfies at least k clauses.

Show that MAX-2SAT is NP-complete, by giving a polynomial time reduction from VERTEX COVER. Recall that VERTEX COVER is a problem where, given an undirected graph $G = (V, E)$, with $|V| = n$, and given a positive integer $\ell \leq n$, one needs to decide whether G has a vertex cover of size at most ℓ. A vertex cover is a subset $V' \subseteq V$ such that for every edge in E, at least one of its endpoints is included in V'.

Hint: To every vertex in the graph, assign a boolean variable which is intended to be TRUE if and only if the vertex is included in the vertex cover. Add clauses of size two corresponding to the edges, and clauses of size one corresponding to the vertices. The clauses corresponding to edges may need to be repeated a number of times.

Solution. Let $(G = (V, E), \ell)$ be an instance of the VERTEX COVER problem, with $V = \{1, 2, \ldots, n\}$, and $m = |E|$. Introduce a boolean variable x_i corresponding to vertex i. The intention is that x_i is TRUE iff vertex i is in the vertex cover. For an edge $e = (i, j) \in E$, let C_e denote the clause

$$C_e := x_i \vee x_j.$$

Note that C_e is satisfied iff either x_i or x_j is TRUE, i.e. iff e is covered by the intended vertex cover.

Let $C_e^1, C_e^2, \ldots, C_e^{2n}$ be clauses that are copies of (the same) clause C_e. Let ϕ be be the 2SAT formula defined as

$$\phi := \bigcup_{i=1}^{n} \{\overline{x}_i\} \qquad \bigcup_{e \in E} \{C_e^1, C_e^2, \ldots, C_e^{2n}\}.$$

Note that ϕ has a total of $n + 2mn$ clauses. Let $k := n + 2mn - \ell$. We will show that there exists a vertex cover of size at most ℓ iff there is an assignment to the n boolean variables that satisfies at least k clauses.

For the forward direction, let V' be a vertex cover of size at most ℓ. Consider the boolean assignment where x_i is TRUE iff $i \in V'$. As observed before, since V' is a vertex cover, all the clauses C_e^j are satisfied for $e \in E, 1 \leq j \leq 2n$. A clause $\overline{x}_i$ is satisfied unless $i \in V'$ and there are at most ℓ such i's. Therefore this assignment satisfies at least $(n - \ell) + 2mn$ clauses.

For the reverse direction, assume that there is a boolean assignment that satisfies at least $n + 2mn - \ell$ clauses. First we claim that all the clauses C_e^j must be satisfied. This is because otherwise, since these clauses occur as $2n$ copies of the same clause, at least $2n$ clauses will be unsatisfied and $2n > n \geq \ell$. Now assume that all the edge clauses are satisfied, which implies that at most ℓ vertex clauses are not satisfied. Let V' be the set of vertices i for which $x_i =$ TRUE. Clearly $|V'| \leq \ell$, and V' is a vertex cover since all the edge clauses are satisfied.

Go To the Next Page

Question 3

Let $T = (V, E)$ be an arbitrary (not necessarily binary) tree of n vertices where each vertex v contains a number $v.val$.

Let the root of T have depth zero, and define

$$Goodness(T) = \sum_{v \in V} 2^{depth(v)} v.val,$$

so that $Goodness(T)$ is the sum of weighted vertex values where the weighting for $v.val$ is 2 raised to the exponent that is the depth of v in T.

Now let $A[1..n]$ be an array of n numbers that could be positive or negative.

For any tree T of n vertices, let $T(A)$ assign the values in $A[1..n]$ to V so that $v.val$ is $A[j]$, where v is the jth vertex reached in a preorder traversal of T. That is, $T(A)$ is formed by the code:

```
global A[1..n];
global preid ← 1;
procedure DFS(T) ;
    T.val ← A[preid];
    preid ← preid + 1;
    for each child v of T do
        DFS(v)
    endfor;
```

Of course, there are many trees T of n vertices. Present a recursive dynamic programming formulation that computes the greatest *Goodess* value for trees with data A. Code is unnecessary as is the use of lookup tables, which would be used to make the solution efficient.
Hint: Observe that T can be viewed as a tree U with the same root as T plus a rightmost subtree W that hangs off of that root. So U is the entirety of T, apart from the rightmost subtree of the root. In this case, the elements of W comprise a suffix of A, which is to say that if W has k vertices, then its data elements are the sequence $A[n-k+1]$, $A[n-k+2]$, ..., $A[n]$.

Solution. Let $Good(i, j)$ be the greatest goodness among all trees built with the preorder values in $A[i..j]$. Then

$$Good(i,j) = \begin{cases} A[i] & \text{if } i = j; \\ \max_{i<k\leq j}\{Good(i,k-1) + 2Good(k,j)\} & \text{otherwise.} \end{cases}$$

Go To the Next Page

Question 4

You are to design an algorithm that determines whether or not the language of a given deterministic finite automaton (DFA) is infinite.

The alphabet for the DFA is $\{0, 1\}$. If the DFA has n states, it is encoded using three arrays: $D_0[1..n]$, $D_1[1..n]$, and $F[1..n]$. Here, it is assumed that the states are $1, \ldots n$, and that state 1 is the start (or initial) state. The arrays D_0 and D_1 encode the transition function: if the machine reads a 0 in state i and moves to state j, then $D_0[i] = j$, and if the machine reads a 1 in state i and moves to state k, then $D_1[i] = k$. The array F encodes the set of final (or accepting) states: if i is a final state, then $F[i] = 1$, and otherwise, $F[i] = 0$.

Design an efficient algorithm that determines if the language of bit strings accepted by such a DFA is infinite. It should be as efficient as possible (ignoring constant factors in the running time).

Solution. (Sketch) Covert the DFA to a graph, find SCC's, and look for a path from the SCC containing the start state top the SCC containing a finial state which goes through some non-trivial SCC, which consists of either two or more nodes, or a single node with a self loop (DFS/BFS + graph cloning easily takes care of all this).

Comment: since this is already pretty easy, I don't tell them to shoot for a $O(n)$ running time.

Go To the Next Page

Question 5

Let $G = (V, E)$ be directed graphs with edge weights $w : E \to \mathbb{R}$. For vertices u and v, the *distance* from u to v, denoted $\delta(u, v)$, is the weight of the least-weight path from u to v, if such a path exists. There are two exceptional cases:

- if there is no path from u to v, then $\delta(u, v) := +\infty$;
- if there is a path from u to v that contains a negative-weight cycle, then $\delta(u, v) := -\infty$.

Design and analyze an algorithm that takes G and w as input, and computes $\delta(u, v)$ for all pairs $u, v \in V$. Your algorithm should run in time $O(|V|^3)$.

HINT: the standard Floyd-Warshall algorithm assumes that the graph contains no negative-weight cycles; show how to modify this algorithm appropriately.

Solution. Number the nodes $1..n$. For $k = 0..n$, define $\delta_k(i, j)$ to be the distance from i to j via paths with intermediate nodes in $\{1..k\}$. Assume $w(i, j) := \infty$ if (i, j) is not an edge.

By inspection, we have:

$$\delta_0(i, i) = \min(0, w(i, i))$$

and

$$\delta_0(i, j) = w(i, j) \text{ for } i \neq j$$

Also, for $k = 1..n$, we have

$$\delta_k(i, j) = \min\Big\{\delta_{k-1}(i, j), \quad \delta_{k-1}(i, k) + (\delta_{k-1}(k, k))^* + \delta_{k-1}(k, j)\Big\},$$

where $x^* := -\infty$ if $x < 0$, and $x^* := 0$, otherwise; moreover, the following convention applies to addition with ∞:

$$(+\infty) + (-\infty) = +\infty.$$

From this, one can just read off an $O(n^3)$-time algorithm.

Go To the Next Page

Question 6

An *addition gate* is a boolean gate with 3 input bits and 2 output bits, representing the sum (modulo 2) of the input bits, and the carry bit. An *addition circuit* is a boolean circuit built out of addition gates, with "fan out" restricted to one. The circuit may have several input bits and several output bits. You are to prove that when such a circuit is evaluated, the number of carries generated by the addition gates is equal to the number of 1-bits in the input minus the number of 1-bits in the output, regardless of the structure of the circuit.

To make the problem more precise, the truth table for an addition gate is as follows, where c is the carry, and s the sum:

			c	s
0	0	0	0	0
0	1	0	0	1
0	0	1	0	1
0	1	1	1	0
1	0	0	0	1
1	1	0	1	0
1	0	1	1	0
1	1	1	1	1

An addition circuit can be viewed as a directed acyclic graph, where each edge (i.e., "wire") is labeled i (an "input wire"), c (a "carry wire"), or s (a "sum wire"), and where vertices (i.e., "gates") are one of three types:

Input: no incoming edges, and one outgoing edge labeled i;

Addition: three incoming edges, and two outgoing edges, labeled c and s;

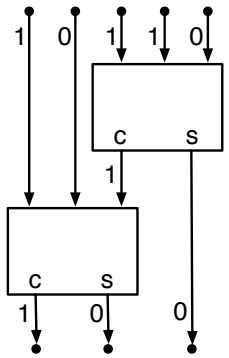
Output: one incoming edge, and no outgoing edges.

The incoming edge of an output gate is called an "output wire" (but it is also either an input, carry, or sum wire).

If we assign bit values to each input wire, then since the graph is acyclic, we may evaluate the circuit, assigning to each wire a bit value, using the above truth table to determine the values assigned to the outgoing carry and sum wires on each addition gate, computed as a function of the values assigned to the incoming wires. If I be the number of 1-bits assigned to input wires, C is the number of 1-bits assigned to carry wires, and O is the number of 1-bits assigned to output wires, then the statement you are asked to prove is: $C = I - O$.

See the following page for an example.

Example. In the following circuit, we have $I = 3$, $C = 2$, and $O = 1$.



Solution. Place \$1.00 on each input bit that is a one, and no money on each zero. Then there are enough dollars so that the dollars can be moved to the outputs with each one covered by a dollar, each zero covered by nothing, and an extra dollar left over whenever a carry is generated. So the number of leftover dollars, which is the difference between the number of ones in the inputs and the number of ones in the answer is exactly the number of carries generated by the adder circuits.