CS172, Spring 2001
Midterm 2 solutions

This midterm is open-book. There are 3 problems and 7 true/false questions. You have 80 minutes; the number of points assigned to each problem reflects the number of minutes expected to be spent on it, so there's a total of 80 points. We are not looking for rigidly formal construction proofs unless otherwise specified, but you should give sufficient detail to demonstrate that you can make the necessary construction based on constructions presented in class and/or in the textbook. Good luck!

1. (20 pts) Show that, if $P = NP$, there would exist a polynomial time algorithm that, given a graph G and a number k on input, outputs a simple path of length k in G if such a path exists.

   This problem requests that we find an *algorithm* that actually *finds* a path of a certain length (as opposed to just deciding whether one exists, which does not necessarily require finding one). Thus, although we may note that the decision problem k-PATH=$\{\langle G,k\rangle | G$ has a simple path of length $k\}$ is clearly in NP, since a certificate (the path itself) is easily verified to be correct in polynomial time, and thus in P given the P=NP assumption, this is not sufficient to obtain the algorithm requested. Rather, we want the algorithm to repeatedly use deterministic polytime deciders for NP decision problems (which exist by the assumption) to progressively discover the certificate, part by part.

   Define the problem:

   k-PATH-HEAD=$\{\langle G,k,v_1,\ldots,v_m\rangle | G$ has a simple path of length k whose first m vertices are $v_1,\ldots,v_m\}$

   which asks for the existence of a path with a specific "head." Clearly, this problem is still in NP since a path satisfying the conditions can still serve as a certificate and be verified quickly. By the P=NP assumption, this problem is also in P. Consider the following algorithm:

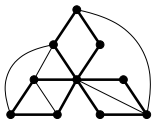
```
check whether ⟨G,k⟩ ∈ k-PATH and return ''no path'' if not
path={}
for i=1 to k
    for all vertices v of G
        check whether ⟨G,k,path⟩ ∈k-PATH-HEAD
        if so, append v to path and skip to next i
return path
```

   Since the k-PATH-HEAD decider is only called $k|G|$ times (and if $k > |G|$, the first line clearly rejects), and the input to it is linear in the length of the original input, $\langle G,k\rangle$, the above algorithm runs in polynomial time. If there exist *any* paths of length k, the above algorithm is guaranteed to discover at least one of them by continually finding larger "heads" that are at the beginning of a valid k-path. □

2. We say that an undirected graph $G = (V, E)$ with $kn + 1$ nodes is a "k-daisy" if there is a collection of k "petals", $P_1, \ldots, P_n$, such that:

   (1) $\forall i, P_i \subseteq V$ and $|P_i| = n + 1$ (each petal is a set of $n + 1$ nodes)
   (2) $\exists c \in V$ s.t. $\forall i \neq j, P_i \cap P_j = \{c\}$ (there's a specific vertex c shared by all petals)
   (3) $\forall i$, there is a simple cycle in G through all vertices of P_i

   Note that the above conditions implicitly force all vertices in the graph other than c to be included in exactly 1 petal (since, in addition to c, each of the k petals has n other nodes that it doesn't share with any other petal, and there's a total of $kn + 1$ nodes). For instance, the following graph is a 3-daisy (with the cycles through the 3 petals highlighted):

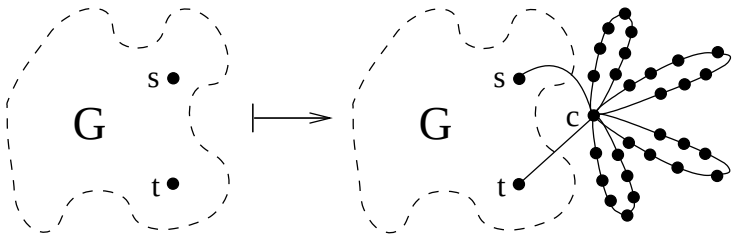


   a. (10 pts) Prove that the language DAISY $= \{\langle G, k\rangle | \text{the graph } G \text{ is a } k\text{-daisy}\}$ is NP-complete. You may not use the result from part (b).

   To see that DAISY is in NP, it is sufficient to note that an assignment of the center c, the subsets of vertices $P_1, \ldots, P_k$ corresponding to the petals, and a path through each petal comprise a certificate for the graph being a daisy that can be easily verified in polynomial time. Now, to prove that DAISY is NP-complete, just note that a 1-daisy corresponds to the graph being a single "petal" with a cycle through all of its vertices, i.e. a Hamiltonian cycle, and, conversely, a graph with a Hamiltonian cycle is immediately a 1-daisy. Thus, a polytime reduction from HAM-CYCLE to DAISY may be obtained by using the function $f(\langle G\rangle) = \langle G, 1\rangle$. A similar reduction may also be obtained from UHAMPATH by adding a single extra "center" node to the graph.

   b. (12 pts) Prove that the language 5-DAISY $= \{\langle G\rangle | \text{the graph } G \text{ is a 5-daisy}\}$ is NP-complete.

   5-DAISY is clearly in NP by an argument similar to the one above – a combination of the center, the 5 petals, and the paths through them are verifiable in polytime. While there is no simple reduction **from** DAISY **to** 5-DAISY, we can still reduce from, for instance, UHAMPATH by adding "vestigial" vertices. Given a graph G with k vertices and desired-endpoint vertices s and t, add an extra vertex c (connected to s and t), and 4 groups of k vertices each with just a cycle going through each of the 4 groups and c:



   (The figure assumes G has 7 vertices). If there's an undirected Hamiltonian path through G from s to t, G clearly forms 1 petal, and the other 4 groups form petals as well, making G a 5-daisy. Conversely, if G is a 5-daisy, it can be immediately seen that (1) none of the $4k$ new vertices other than c can be the center, so that forces c to be the center, (2) thus each of the 4 groups must form a separate petal, (3) G must've formed the last petal, and there must thus be a u. ham. path through it, from s to t. Thus the above is a reduction from UHAMPATH to 5-DAISY. It is polynomial time since the input size is only increased by a constant factor, and the conversion procedure is trivially polytime.

3. Define $L = \{\langle M \rangle | M \text{ is a TM that accepts at least 2 distinct strings}\}$.

   a. (8 pts) Show that A_{TM} is mapping-reducible to L.

   Given an input $\langle M, w \rangle$ to A_{TM}, we construct a machine M' that ignores its own input, simulates M on w, and accepts iff M accepted. If $\langle M, w \rangle \in A_{TM}$, M accepts w, so M' accepts Σ^*, and thus $\langle M' \rangle \in L$. Conversely, if $\langle M, w \rangle \notin A_{TM}$, M does not accept w, so $L(M') = \emptyset$, and thus $\langle M' \rangle \notin L$. Therefore, $f(\langle M, w \rangle) = \langle M' \rangle$ is a mapping reduction. $\square$

   b. (10 pts) Use the Recursion Theorem to show that L is undecidable. You may not use your result from part (a).

   Suppose a Turing machine, D, decides L. Then construct a new machine as follows:

   ```
   M = Ignore input
   Obtain its own description, ⟨M⟩, via the Recursion Theorem
        Run D on M
        If D accepts, reject
        Else, accept
   ```

   Since D is a decider, the third step always terminates, and thus so does M. If $\langle M \rangle \in L$, it is accepted by D and thus rejects all strings, so it can't be in L. On the other hand, if $\langle M \rangle \notin L$, it's rejected by D and thus accepts everything, putting it in L. This is a contradiction, and thus neither M nor D can exist, so L is undecidable. $\square$

   c. (8 pts) Show that L is Turing-recognizable.

   Given a machine M as input, we can recognize L by running M "on Σ^*, and terminating and accepting whenever we find 2 strings on which M terminates and accepts. Since Σ^* is infinite, we implement this notion by ordering Σ^* in some order (say lexicographic), and run M for 1 step on the 1st string, then for 1 step on the 1st and 2nd strings, and so on (producing an "interleaving" pattern similar to that used in the classical proof of the countability of rationals). Clearly, whenever M terminates and accepts after a certain number of steps on some 2 strings, we'll eventually reach the right step on each of those 2 strings, and accept as well; otherwise, we'll never have a reason to terminate and accept. Thus this procedure recognizes L. $\square$

4. (12 pts) True/False:

True If COMPOSITE $= \{\langle N \rangle | N$ is a composite integer$\}$ is not NP-complete, then $P \neq NP$. *If $P = NP$, all non-trivial problems in NP are NP-complete. So if* COMPOSITE *isn't, we know $P \neq NP$.*

False If a function $f : A \to B$ is a mapping reduction from A to B *and* is one-to-one, then B is mapping-reducible to A as well. *We aren't guaranteed that f is invertible (we don't know if it's onto). Counterexample: $\{0\} \leq_m A_{TM}$, but not vice versa.*

True If A and B are in NP, then so is $A \cup B$. *We can create a certificate for w being in $A \cup B$ by providing either its certificate as a member of A or of B, and indicating which one we're providing.*

False If A and B are NP-complete, then so is $A \cap B$. $0 \circ SAT \cap 1 \circ SAT = \emptyset$, *which is not NP-complete.*

True $CFL \subsetneq P$ (where CFL is the class of context-free languages). *We know $CFL \subseteq P$ since we demonstrated an $O(n^3)$ universal parser for CFLs; clearly, $\{0^n 1^n 2^n\} \in P$, but it's not in CFL. Thus the inclusion is proper.*

False The set of problems *to* which A_{TM} is mapping-reducible is uncountable (note that all such problems are necessarily undecidable). *A mapping reduction is a computable function, which is defined by a TM, and there's only a countable number of distinct TMs.*