Course: CSC707, Automata, Computability and Computational Theory
EXAM 1: Countability, closure properties of countable sets. Complexity theory, NP-completeness, polynomial-time reducibility, self-reduction, approximability.
Submission: Home-take
FULL NAME: Xusheng Xiao

Due Date: February 25, 11:59 PM Mid-Term Exam, Spring, 2010

---

1. Provide a solution to **ONE** problem in **EACH** category; a total of **FOUR** categories.
2. Solutions to the other problems in each category will **NOT** be graded.
3. This is an open-textbook (main course book), open-notes, open-homeworks, BUT **closed-internet** exam. You may **NOT** discuss the exam with any one. The exam is an **INDIVIDUAL** effort.

---

1. Prove or disprove the countability of each of the following sets (**25 points**: identifying which class (5 points), proof idea (5 points), complete proof (15 points)):
   (a) The set of all regular languages over $\{0, 1\}$.
   (b) The set of all languages over the alphabet $\{0, 1\}$
   (c) The set of all infinite length strings over the three letter alphabet $\{0, 1, 2\}$

   I would prove that the set of all infinite length strings over the three letter alphabet $\{0, 1, 2\}$ is uncountable.

   Theorem 1 *The set S of all infinite length strings over the three letter alphabet $\{0, 1, 2\}$ is uncountable.*

   Proof. Prove by contradiction using diagonalization. Assume that S is countable. By defnition of a countable set, there exists a bijection function $f : N \to S$. Therefore, the elements of S can be enumerated as:

$$S = \{s_1, s_2, \ldots, s_n, \ldots\}$$

   Each element of S can be represented as follows:

$$s_j = a_{j1}a_{j2}a_{j3} \ldots a_{jn} \ldots | a_{jn} \in \{0, 1, 2\}, \forall n \in N$$

$n \in N$	s_n	1	2	3	...	j	...
$1 \rightarrow$	s_1	a_{11}	a_{12}	a_{13}	...	a_{1j}	...
$2 \rightarrow$	s_2	a_{21}	a_{22}	a_{23}	...	a_{2j}	...
$3 \rightarrow$	s_3	a_{31}	a_{32}	a_{33}	...	a_{3j}	...
...	...	...	...	...	...	...	...
$j \rightarrow$	s_j	a_{j1}	a_{j2}	a_{j3}	...	a_{jj}	...
...	...	...	...	...	...	...	...

The elements of S can be represented as a matrix, where each row j corresponds to a infinite length of string s_j and each element in row j and column k is an alphabet $a_{jk} \in \{0,1,2\}$

Construct a new element s_{new} from S as follows:

$$s_{new} = a_{k1}a_{k2}a_{k3} \dots a_{kn} \dots | a_{kn} \in \{0,1,2\}, a_{kn} \neq a_{nn}, \forall n \in N$$

By definition of S, $s_{new} \in S$. It means that $\exists j \in N$ such that $s_{new} = s_j$. But by definition of s_{new}, $a_{kj} \neq a_{jj}$. Thus, $s_{new} \neq s_j \Rightarrow s_{new} \notin S$. Contradiction.

■

2. NP**-completeness, 25 points**: Verification Step (5 points), Reduction Step (10 points), Correctness Step (10 points): Prove that the problem is NP-complete.

   (a) BIGGER-CLIQUE = $\{\langle G_1, G_2 \rangle \mid$ the largest clique of graph G_1 is larger than every clique of graph $G_2\}$

   (b) NP-PATH = $< G, s, t, k >$ — G is an undirected graph containing a simple path of length at least k from s to t. (Hint: *Use the fact that the Hamiltonian Path problem for undirected graphs is NP-complete.*)

   (c) HALF-CLIQUE = $< G >$ — G is an undirected graph having a complete subgraph with $\lfloor n/2 \rfloor$ nodes, where n is the number of nodes in G.

I would prove that NP-PATH is NP-complete.

INSTANCE: A graph $G = (V, E)$, a starting vertex $s \in V$, an ending vertex $t \in V$ and a positive integer k QUESTION: Does G have a simple path $P = \{e_1, e_2, \dots, e_k | e_n \in E, 1 \leq n \leq k\}$ of length at least k from s to t?

Theorem 2 *NP-PATH is NP-complete.*

Proof.

Step 1: Verification

$NP - PATH$ can be verified as follows:

(a) Count the number of the edges of the given path, this takes $O(|P|)$.

(b) If $|P|$ is less than k, then return "no".

(c) Else, create an integer array C of size $|V|$, and assign 0 to each element in the array, which requires $O(|V|)$. In this way, each vertex can be mapped to one element of the array. For each edge in the path P, we can know the two endpoints of the edge and increase the value of the corresponding elements in the array C, which takes tiem of $O(|P|)$. Since the path starts from s and ends at t, the value of the corresponding elements in the array must be 1. All the other vertices in the path must be mapped to the elements whose value is 2.

(d) All the egdesAfter all the edges are counted, we can check each element in the array C, if the element that mapped to s and t are not 1, then return "no". If there are $|P| - 1$ elements whose count is 2, then return "yes". Otherwise, return "no". This takes time $O(|V|)$.

Since all these steps can be completed in polynomial time, the total time required for verifying NP-PATH can be completed in polynomial time. Thus, $NP - PATH$ is in NP.

Step 2: Reduction

We now show that $NP - PATH$ is $NP - Hard$. Let $< G, s, t >$ be ANY instance of $HamiltonianPath$, let $n = V(G)$. We produce SOME instance $< G', s', t', k' >$ of $NP - PATH$ as follows:

Construct $G' = G$,$s' = s$,$t' = t$,$k = n - 1$, Return $< G, s, t, k >$. This construction is in $O(|V| + |E|)$, which is polynomial.

Step 3: Correctness

If there is a $HamiltonianPath$ from s to t, then there is a simple of length $n-1$, which is a $NP-PATH$ of length at least $n-1$ from s to t. Similarly, if there is a $NP-PATH$ of length at least $n-1$ from s to t, then there is a simple path of length $n-1$ that visits every vertiex in the graph, which means that it is a $HamiltonianPath$ from s to t.

■

3. **Self-reducibility, 25 points**:

(a) Find an isomorphism between graphs G_1 and G_2 and provide time complexity, or state that none exists. (An isomorphism is a bijection $\phi : V(G_1) \to V(G_2)$ such that $(v_1, v_2) \in E(G_1)$ iff $(\phi(v_1), \phi(v_2)) \in E(G_2)$.) Assume that you have a decision algorithm $D(G, G')$ that decides whether G and G' are isomorphic in $O(f(|V(G)| + |V(G')|))$ time.

(b) Given a set of integers A, find a subset of A that sums to zero and provide time complexity, or state that none exists. Assume that you have a decision algorithm $D(S)$ that decides whether such a subset exists in $O(f(|S|))$ time.

I would solve the problem (b).

(a) **Given**: A set of integers A, and decision algorithm $D(S)$ that decides whether a subset that sums to zero exists in $O(f(|S|))$ time.

(b) **Return**: If A has a subset that sums to zero, return the instance of it, else return empty.

(c) **Procedure**:

   i. Mark every integer in A white.

   ii. Call $D(A)$

   iii. If return "no", then return empty. Otherwise, go to step iv

   iv. Construct A' by removing an arbitrary integer i that is marked white.

   v. Call $D(A')$, if return "yes", $A = A'$ and repeat step iv until there is no integer marked white.

   vi. If return "no", mark i black, put i back to A and repeat step iv until there is no integer marked white.

After these steps, all the integers left in A is a subset that sums to zero. We could call $D(A)$ for $|A|$ times, thus the time complexity is $O(|A| * f(|S|))$

4. **Approximability, 25 points**:

(a) If $P \neq NP$, then Vertex Cover does not allow any absolute approximation. (There is an absolute approximation algorithm A if $A(G) \leq OPT(G) + C$ for some constant integer C and any instance, G of the vertex cover.)

(b) The VERTEX-COVER problem and the NP-complete CLIQUE problem are complementary in the sense that an optimal vertex cover is the complement of a maximum-size clique in the complement graph. Does this relationship imply that there is a polynomial-time approximation algorithm with a constant approximation ratio for the CLIQUE problem? Justify your answer.

I would solve problem (a).

Theorem 3 *If $P \neq NP$, then Vertex Cover does not allow any absolute approximation.*

Proof. Prove by contradiction.

Suppose that given any graph G, there is an absolute approximation A such that $A(G) \leq OPT(G) + C$ for some constant C. For any graph G, we can construct a graph G' by making $C + 1$ copies of G and no two copies of G are connected to each other. Then a vertex cover in G' consists of a vertex cover in each copy of G and $OPT(G') = (C + 1)OPT(G)$. Call

$A(G')$, we can get an approximation to a vertex cover of G' whose size is at most $OPT(G') + C$. This vertex cover contains a vertex cover for each copy of G in G'. By the defintion of A, we have:

$$A(G') \leq OPT(G') + C$$
$$\Rightarrow A(G') \leq (C+1)OPT(G) + C$$
$$\Rightarrow (C+1)A(G) - (C+1)OPT(G) \leq C$$
$$\Rightarrow \frac{A(G')}{C+1} - OPT(G) \leq \frac{C}{C+1}$$

Since the solution to vertex cover can only be integer, we can have:

$$\frac{A(G')}{C+1} - OPT(G) \leq \frac{C}{C+1} \Rightarrow A(G) - OPT(G) \leq 0 \Rightarrow A(G) - OPT(G) = 0$$

In this way, we can obtain an exact solution to vertex cover in polynomial time. Since vertex cover is a NP-complete problem, if it can be solved in polynomial time, $P = NP$, which contradicts with our assumption that $P \neq NP$.

■