

Solutions for Sample Final Problems

1. We are asked to show that the language $\text{Accept}(\Lambda)$ or the set of encodings of Turing machines that accept the empty string is undecidable. The basic idea of the proof is a reduction from the undecidable Halting Problem. Given $e(T), x$ with the question does Turing machine T halt on x , we construct $e(T')$ where T' is a Turing machine that first simulates T on x , and if this halts, checks to see if its input is the empty string. If it is, it accepts. We have that T' accepts empty string if and only if T halts on x . This shows that $\text{Accept}(\Lambda)$ is at least as hard as the Halting problem, and so is undecidable. (See proof of Theorem 12.4 on page 344 in the textbook.)
2. The decision problem of determining whether a given boolean formula is satisfiable or not is in **NP**. Thus, if $\mathbf{P} = \mathbf{NP}$, it means that it can be determined in polynomial time; in particular, there is a polynomial-time algorithm that will determine whether a given formula is satisfiable or not. Let $\text{isSat}(\phi)$ denote the algorithm, where ϕ is a formula.

Assume that the given formula ϕ is over the set of variables x_1, \dots, x_n . Then, the following algorithm finds a satisfying truth assignment (or announces that ϕ is unsatisfiable).

1. **for** $i = 1$ **to** n **do**
2. **if** $\text{isSat}(\phi|_{x_i=\text{true}})$ **then** $\phi \leftarrow \phi|_{x_i=\text{true}}$
3. **else if** $\text{isSat}(\phi|_{x_i=\text{false}})$ **then** $\phi \leftarrow \phi|_{x_i=\text{false}}$
4. **else reject end if**
5. **end for**
6. **accept**

By our supposition, each call to isSat costs $p(n)$ in time for some polynomial function p . There are at most $2n$ calls to it. Thus, the total cost is no more than $2np(n)$, which is clearly polynomial.

The identical question for **NP** can also be solved with the above algorithm. However there is a much simpler solution that exploits non-determ.:

Assume the non-determ. TM M accepts L . We construct a two-tape non-determ. TM M' for accepting L^* . M' iterates the following. If the input left on tape one is empty then it accepts:

Remove an initial segment of the input and move it to tape two. Run M on tape two. Erase tape two (keep markers to facilitate erasing tape two).

M' accepts L^* because a word lies in L^* iff it can be split into pieces that are all accepted by M .

3. We will construct an algorithm that would accept input strings that are in L^* in polynomial time. The algorithm uses dynamic programming and will iteratively compute values for a one-dimensional boolean-valued table T . Intuitively, if $T(i)$ is true, then $x_1 \cdots x_i$ is in L^* ; in particular, if $T(|x|)$ is true, then x (the input string) is in L^* . Our argument is that all the values in T can be computed in polynomial time.

Define T to be as follows:

$$T(i) = \begin{cases} \text{true} & \text{if } i = 0 \\ \bigvee_{j=0}^{i-1} (T(j) \wedge (x_{j+1} \cdots x_i \in L)) & \text{if } i > 0 \end{cases}$$

The values of T can be computed using the following algorithm:

1. $T(0) \leftarrow \text{true}$
2. **for** $i = 1$ **to** $|x|$ **do**
3. $T(i) \leftarrow \text{false}$
4. **for** $j = 0$ **to** $i - 1$ **do**
5. $T(i) \leftarrow T(i) \vee (T(j) \wedge (x_{j+1} \cdots x_i \in L))$
6. **if** $T(|x|)$ **then** **accept** **else** **reject**

To see that the algorithm runs in polynomial time, note that the membership test can be done in polynomial time. (This is because $L \in \mathbf{P}$.) Membership test need not be done any more than $|x|^2$ times (this is somewhat of an over-estimate, but it suffices for our purpose of showing that the algorithm runs in polynomial time). Thus, the overall cost is clearly polynomial.

4. HAMILTONIAN PATH (HP) is in \mathbf{NP} because a Hamiltonian path (i.e., a solution) can be guessed in polynomial time (as some permutation of the set of vertices), and it can also be verified in polynomial time (check that adjacent vertices in the path have an edge between them).

Given an instance I_1 of HAMILTONIAN CYCLE (HC) as $G_1 = (V_1, E_1)$, we construct an instance I_2 of HP as $G_2 = (V_2, E_2)$ as follows:

- (a) Initialize V_2 to be V_1 , and E_2 to be E_1 .
- (b) Arbitrarily pick a vertex from V_2 ; let this be v .
- (c) Make a copy of v , including making copies of edges that are incident on v . Let v' denote the copy. The new vertex v' is added to V_2 , and the new edges are added to E_2 .
- (d) Add two edges (v, u) and (v', u') where u and u' are new vertices. The new vertices u and v are added to V_2 , and the new edges (v, u) and (v', u') are added to E_2 .

The construction (transformation) is clearly polynomial in time.

Suppose I_1 has a Hamiltonian cycle γ . Without loss of generality, we can assume that γ begins and ends in v (the vertex that was copied during transformation). Thus, $\gamma = (v, x_2, \dots, x_{|V_1|}, v)$, where $x_2, \dots, x_{|V_1|}$ are vertices chosen from $V_1 - \{v\}$ without repetition. Then the path $(u, v, x_2, \dots, x_{|V_1|}, v', u')$ is a Hamiltonian path of G_2 .

Suppose I_2 has a Hamiltonian path π . Since u and u' are degree-one vertices, any Hamiltonian path must begin with one of the two and end in the other. Without loss of generality, suppose that π begins with u ; then the path is of the form $\pi = (u, v, y_2, \dots, y_{|V_1|}, v', u')$, where $y_2, \dots, y_{|V_1|}$ are vertices chosen from $V_2 - \{v, v', u, u'\}$ without repetition. Then, the cycle $(v, y_2, \dots, y_{|V_1|}, v)$ is a Hamiltonian cycle of G_1 because $V_2 - \{v, v', u, u'\}$ is equal to $V_1 - \{v\}$.

5. A decision problem L is \mathbf{NP} -complete if and only if the following two conditions are satisfied.
 - L is in \mathbf{NP} ; that is, given an instance of L , it can be solved using a nondeterministic Turing machine in polynomial time.
 - L is \mathbf{NP} -hard; that is, any problem Q in \mathbf{NP} can be reduced to L in polynomial time; this is denoted as $Q \leq_p L$. (Note that in practice, it is sufficient to show that L has a polynomial-time reduction from an \mathbf{NP} -hard problem.)
6. (Exercise 15.19 on page 434 in the textbook) $L = \{e(T) \mid T \text{ accepts } x \text{ in } t \text{ steps or less}\}$ is in \mathbf{NP} because given an input, a NDTM can guess a computation of T and verify whether T accepts x in t steps or less by simulating T on x . Since t is encoded in unary, the simulation can be done in polynomial time.

To show that L is **NP**-hard, we directly show that there is a reduction from every **NP** language. We show a *generic* transformation, which can further be “instantiated” for each **NP** language.

Note that an **NP** language can be characterized by a NDTM that accepts it. In the discussion below, we fix for a particular **NP** language L_1 and for a particular NDTM T that accepts L_1 , but these can be changed for other **NP** languages. Note further that every string x in L_1 can be accepted by T in $p(|x|)$ steps or less, where $p(n)$ is some polynomial function. Although we do not know what p is, we do know that it exists and it is a polynomial function. Not knowing p exactly is not a problem as we only need to show that a transformation *exists* (as opposed to actually describing how to construct a transformed problem instance).

Given a string x , we output $e(T)x1^{p(|x|)}$; let x' denote the output string. This can be done in polynomial time, because we append a copy of the encoding of T (constant length once T has been fixed) and the string $1^{p(|x|)}$ which is of polynomial length.

If x is in L_1 , then T accepts x in $p(|x|)$ steps or less. Thus, x' is in L .

Suppose x' is in L . This means that T accepts x (in polynomial time), which means that x is in L_1 .

7. TSP (Traveling Salesperson Problem) is in **NP** because a tour (i.e., a solution) can be guessed in polynomial time (as some permutation of the set of vertices), and the tour can be verified in polynomial time (compute the length of the tour and compare it against the bound).

To show that TSP is **NP**-hard, we show a polynomial-time reduction from HC (Hamiltonian Cycle).

Given an instance I_1 of HC as $G_1 = (V_1, E_1)$, construct an instance I_2 of TSP as $G_2 = (V_2, E_2, w_2)$ and B_2 as follows:

$$\begin{aligned} V_2 &= V_1 \\ E_2 &= \text{clique of size } |V_1| \\ w_2(u, v) &= \begin{cases} 1 & \text{if } (u, v) \in E_1 \\ 2 & \text{if } (u, v) \notin E_1 \end{cases} \\ B_2 &= |V_1| \end{aligned}$$

The transformation can be done in polynomial time.

Suppose I_1 has a Hamiltonian cycle. The total length of the Hamiltonian cycle is $|V_1|$, which satisfies the bound B_2 .

Suppose I_2 has a successful tour. A successful tour can only use edges whose length is 1. This means that all the edges used in the tour exists in G_1 . Thus, the tour is a Hamiltonian cycle of G_1 .

8. The difference between k -CLIQUE and CLIQUE is that the size of a clique sought is specified as part of the *problem* and not as part of an *instance*. The approach we will take to show that k -CLIQUE is in **P** is to systematically guess a subset of size k and test whether the subset is a clique. The algorithm follows. Assume that the graph is given as $G = (V, E)$.

1. **for** each subset $V' \subseteq V$ of size k **do**
2. $found \leftarrow \text{true}$
3. **for** each pair of vertices (u, v) in V' where $u \neq v$ **do**
4. **if** $(u, v) \notin E$ **then** $found \leftarrow \text{false}$
5. **if** $found$ **then** accept
6. reject

Assuming that adjacency list is used to record edges, each test to see if an edge exists in E or not can be done in $O(|E|)$ time. For a given subset, there are no more than k^2 tests, which is a constant relative to the size of the input. The number of subsets is C_k^n , the k -th binomial coefficient of n , which equals $n(n-1)(n-2)\dots(n-k+1)/k(k-1)(k-2)\dots 1$, which is $O(n^k)$. (The numbers are not necessarily “tight”, but it is sufficient for our purpose of showing that the algorithm runs in polynomial time.) Overall, the running time of the algorithm is $O(|V|^k|E|)$, which is clearly polynomial in the length of the input.

9. • **FEEDBACK VERTEX SET (FVS)** is in **NP**; this is because a subset can be guessed and verified for the desired property in polynomial time.

The transformation takes an instance I_1 of VC as $G_1 = (V_1, E_1)$ and K_1 , and constructs an instance I_2 of FVS as $G_2 = (V_2, A_2)$ and K_2 as follows:

$$\begin{aligned} V_2 &= V_1 \\ A_2 &= \{(u, v), (v, u) \mid \{u, v\} \in E_1\} \\ K_2 &= K_1 \end{aligned}$$

The transformation can clearly be done in polynomial time because we are just doubling all the edges.

Suppose I_1 has a vertex cover. Then, for each arc in G_2 , at least one of the “head” or the “tail” is covered by the vertex cover. Thus, any cycle includes a vertex in the vertex cover. This means that the vertex cover is also a feedback vertex set of G_2 .

Suppose I_2 has a feedback vertex set V' . This means that any cycle in G_2 has at least one vertex in V' . In particular, any cycle of the form (u, v, u) has either u or v (or both) in V' . This means that at least one endpoint of edge $\{u, v\}$ in G_1 is in V' . This means that V' is a vertex cover of G_1 .

- **DOMINATING SET (DS)** is in **NP**; this is because a subset can be guessed and verified for the desired property in polynomial time.

The transformation takes an instance I_1 of VC as $G_1 = (V_1, E_1)$ and K_1 , and constructs an instance I_2 of DS as $G_2 = (V_2, E_2)$ and K_2 as follows:

$$\begin{aligned} V_2 &= V_1 \cup \{x_{e_1}, \dots, x_{e_{|E_1|}}\} \\ E_2 &= \{(u, v), (u, x_e), (x_e, v) \mid e \in E_1 \text{ where } e = \{u, v\}\} \\ K_2 &= K_1 \end{aligned}$$

where $x_{e_1}, \dots, x_{e_{|E_1|}}$ are new vertices not in V_1 .

The transformation can clearly be done in polynomial time because we have essentially converted each edge in the original graph into three edges in the new graph.

Suppose I_1 has a vertex cover V' . We wish to show that every vertex in V_2 is either in V' or connected to a vertex in V' by an edge. For each vertex in V_2 that is also in V_1 , this condition is satisfied because V' is a vertex cover of G_1 . For each vertex x_{e_i} in $\{x_{e_1}, \dots, x_{e_{|E_1|}}\}$, since it is connected to both endpoints of some edge, the condition is also satisfied. Thus, V' is also a dominating set of G_2 .

Suppose I_2 has a dominating set V'' . We first show that V'' can be modified to only include vertices that came from V_1 and still be a dominating set of G_2 . Suppose V'' includes a vertex x_{e_i} . We claim in particular that replacing this vertex with one of the vertices adjacent to x_{e_i} will retain V'' as a dominating set. This is because the only vertices that (possibly) benefit from

having x_{e_i} in the dominating set is the two vertices u and v where $e_i = (u, v)$. Since there is an edge between u and v , replacing x_{e_i} with either u or v will not change the property. From this point on, we assume that V'' only includes vertices that are also in V_1 . We now claim that V'' is also a vertex cover of G_1 . For every edge (u, v) in E_1 , either u or v must be in V'' ; otherwise, x_e is not adjacent to a vertex in V'' and causes V'' to not be a dominating set of G_2 . Thus, V'' is a vertex cover of G_1 .

10. Input: A set of pairs

$$(\alpha_1, \beta_1), (\alpha_2, \beta_2), \dots, (\alpha_n, \beta_n),$$

where the α_i and β_i are words over some finite alphabet Σ .

Question: Is there a sequence of one or more integers i_1, i_2, \dots, i_k , such that each i_j is an integer between 1 and n and

$$\alpha_{i_1}, \alpha_{i_2}, \dots, \alpha_{i_k} = \beta_{i_1}, \beta_{i_2}, \dots, \beta_{i_k}.$$

This decision problem is undecidable.

11. There are uncountably infinite many reals in $(0,1)$. However the number of computable reals is only countably infinite. This is because each computable real is enumerated by a TM and there are only countably infinitely many TM's.
12. SA is Turing-acceptable and thus there is an unrestricted grammar. The language class generated by unrestricted grammars is the class of Turing acceptable languages.
13. $b(n)$ is the maximum number of ones left on the tape by any TM with n states and tape alphabet $\{0, 1\}$ that halt on input 1^n .

This function cannot be computed by a TM.

Proof: A diagonalization argument; you should know this proof!

By the Church-Turing thesis no computational device can compute this function including those that use light to send messages.

14. Design a TM that implements the following alg:

```
Ignore the input
For n=1 step 1 do
  Decide whether T halts on n using T_H
  If the answer is no then halt
```

The above machine halts on any input iff there is a number that does not go to one (i.e, Collatz's conjecture is false).

Thus by feeding the above TM and input 1 to T_H , Collatz's conjecture can be resolved.

15. See class notes.
16. See class notes.