Problem Set 9

In this final problem set of the quarter, you will explore the limits of what can be computed efficiently. What problems do we believe are computationally intractable? What do they look like? And are they purely theoretical, or might you might bump into one some day?

Start this problem set early. It contains three problems (plus one survey question and one extra credit problem), each of which requires thought. As much as you possibly can, please try to work on this problem set individually. That said, if you do work with others, please be sure to cite who you are working with and on what problems. For more details, see the section on the honor code in the course information handout.

In any question that asks for a proof, you **must** provide a rigorous mathematical proof.

As always, please feel free to drop by office hours or send us emails if you have any questions. We'd be happy to help out.

This problem set has 125 possible points. It is weighted at 6% of your total grade (note that this is slightly less than usual). The first two questions serve as a warm-up for the last problem, so be aware that the difficulty of the problems increases over the course of this problem set.

Good luck, and have fun!

Due Friday, December 9th at 2:15 PM

Problem One: Nondeterministic Algorithms (30 points)

Nondeterministic Turing machines make it possible to solve many problems that might otherwise seem unrecognizably or undecidably hard. To wrap up our tour of computability theory, let's see how we can use the NTM to tackle tricky TM problems.

i. Prove that the RE languages are closed under concatenation. That is, if L_1 and L_2 are RE, then L_1L_2 is RE as well. *(Hint: If you are given a string w in L_1L_2 and knew how to cut it into two pieces x and y such that $x \in L_1$ and $y \in L_2$, could you confirm that you had guessed correctly?)*

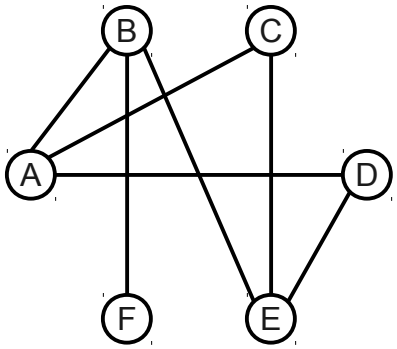
ii. Consider the language

$$L_{ne} = \{ \langle M \rangle \mid L(M) \neq \emptyset \}$$

Using Rice's theorem it is possible to show that this language is undecidable (you don't need to prove this, but be sure that you understand why!) However, L_{ne} is indeed recognizable. Design an NTM that recognizes L_{ne}, then prove why your NTM is correct.

Problem Two: The Long Path Problem (30 points)

Given an undirected graph G = (V, E), a *simple path* in a G is a path between two nodes u, v ∈ V such that no node is repeated on the path. For example, given this graph:



A → C → E is a simple path from A to E, but A → B → E → C → A → D is not a simple path because node A is visited twice.*

Consider the following language:

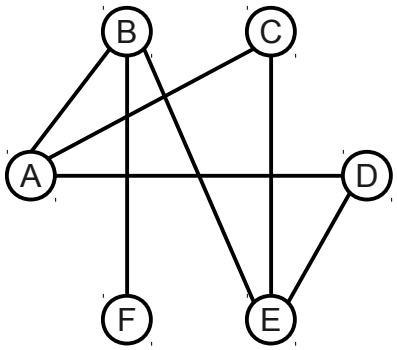
ULONGPATH = { ⟨G, u, v, k⟩ | G is an undirected graph,
u and v are nodes in the graph, and
there exists a simple path from u to v containing k nodes. }

* Although we have defined a path in a graph as a series of edges, it is often easier to reason about the path as the series of nodes that it passes through. Throughout this problem set, we will adopt this convention.

For example, if G is the above graph, then ⟨G, D, F, 6⟩ ∈ *ULONGPATH* because there is a simple path of six nodes from D to F (namely, D → A → C → E → B → F), but
⟨G, A, C, 5⟩ ∉ *ULONGPATH* because there is no simple path of 5 nodes from A to C.

i. Prove that *ULONGPATH* is in NP by designing a polynomial-time verifier for it.

In an undirected graph G = (V, E), a *Hamiltonian path* is a simple path between two nodes u and v that visits every node in the graph exactly once. For example, in this graph:



The path F → B → E → C → A → D is a Hamiltonian path from F to D, but F → B → E → D is not (because it doesn't visit every node), nor is F → B → E → D → A → C → E → D (because it is not a simple path).

The language *UHAMPATH* is defined as follows:

UHAMPATH = {⟨G, u, v⟩ | G is an undirected graph and
there is a Hamiltonian path from u to v.}

UHAMPATH is known to be NP-complete by a fairly clever series of reductions from *SAT*; see Sipser, page 291 for more details.

ii. Prove that *ULONGPATH* is NP-complete by finding a polynomial-time reduction from *UHAMPATH* to it. That is, show how to turn an input ⟨G, u, v⟩ to *UHAMPATH* into an input ⟨G, u, v, k⟩ to *ULONGPATH* in polynomial time such that ⟨G, u, v⟩ ∈ *UHAMPATH* iff ⟨G, u, v, k⟩ ∈ *ULONGPATH*.

Problem Three: Word Ladders (60 points)

A *word ladder* is a sequence of distinct words such that each word differs from the previous word in the sequence by one letter. For example, the series of words

prove → prose → prese → prest → wrest → weest → geest → guest → guess

is a word ladder from "prove" to "guess,"* and

imply → amply → ample → amole → anole → anile
→ anise → arise → prise → paise → pause → cause

is a ladder from "imply" to "cause."† However,

cat → cot → dot → rot → cot → dot → dog

is **not** a word ladder, because the word "cot" is duplicated, and

power → mower → moves → moves → moles →
molts → melts → meats → seats → stats → stays

is **not** a word ladder, because the words "mower" and "moves" differ by more than one letter.

Suppose that we are interested in finding long word ladders that connect two different words. For example, while both

cat → cot → cog → log → leg → peg → pig → pin → pen → den → don → dog

and

cat → cot → cog → dog

are word ladders connecting "cat" and "dog," the latter is much longer than the former. We will say that the length of a word ladder is the number of words in it, so the first ladder from cat to dog has length 12, while the second has length 4. A word ladder consisting of a single word has length one.

* "Prese" is plural of "presa," the time at which a singer should begin singing. "Prest" means "ready" or "prepared." "Weest" means "the most wee," or the smallest. "Geest" is a type of dry soil found in northern Germany.

† "Amole" is the bulb of certain types of plants. "Anole" is another word for "chameleon." "Anile" means "like an old woman." "Paise" is a type of Indian currency.

As you may have guessed, these ladders were generated by a computer. My vocabulary is nowhere near this good. :-)

Consider the following language:

LONGLADDER = { ⟨D, s, t, k⟩ | D is a list of words, s and t are words in D, and there is a ladder of length at least k between s and t }

It turns out that this problem is NP-complete, and in this problem you'll explore the reduction that proves this. But first, we need to establish that *LONGLADDER* is in NP in the first place.

i. Prove that *LONGLADDER* ∈ NP by designing a polynomial-time verifier for it.

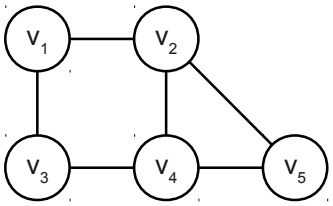
In order to show that *LONGLADDER* is NP-complete, we'll reduce *ULONGPATH* from the previous problem to *LONGLADDER*. The idea behind the proof will be as follows. Given a graph, a start node, an end node, and a length, we will build a dictionary out of the graph such that simple paths in the graph correspond to word ladders in the dictionary. That way, there will be a sufficiently long simple path between the original pair of nodes if and only if there is a sufficiently long word ladder between two corresponding words in the dictionary.

Given an undirected graph G = (V, E), we will construct a dictionary of words from V and E. Order the vertices in the graph as V = $\{v_1, v_2, \ldots, v_n\}$. Then consider the dictionary of words over the alphabet {0, 1} defined like this:

- For each node v_i, there is a word of length n (where n is the number of nodes in the graph) that is 0 everywhere, except at index i, at which it is a 1. We will call these **node words** and will let w_i denote the node word for v_i.

- For each edge $\{v_i, v_j\}$, there is a word of length n that is 0 everywhere, except that the ith and jth characters are 1s. We will call these **edge words** and denote the edge word for edge $\{v_i, v_j\}$ as $e_{i,j}$. Since G is undirected, let $e_{j,i} = e_{i,j}$ be another name for this edge word.

For example, given this undirected graph



We would construct the dictionary

{10000, 01000, 00100, 00010, 00001, *(node words)*
11000, 10100, 01010, 01001, 00110, 00011 } *(edge words)*

Where w_1 = 10000, w_2 = 01000, w_3 = 00100, etc. and $e_{1,2}$ = 11000, $e_{2,4}$ = 01010, etc.

This dictionary has the special property that a simple path in the original graph corresponds precisely to a word ladder in the new dictionary and vice-versa. For example, consider the path

$$v_1 \rightarrow v_2 \rightarrow v_5 \rightarrow v_4 \rightarrow v_3$$

Notice that if we list off the node words for the nodes in this path in order, we get the following:

w_1	→	w_2	→	w_5	→	w_4	→	w_3
10000	→	**01000**	→	**00001**	→	**00010**	→	**00100**

Although this is not a word ladder, we can make it a word ladder by inserting the corresponding edge words in-between these node words:

w_1	→	$e_{1,2}$	→	**w_2**	→	$e_{2,5}$	→	**w_5**	→	$e_{5,4}$	→	**w_4**	→	$e_{4,3}$	→	**w_3**
10000	→	11000	→	**01000**	→	01001	→	**00001**	→	00011	→	**00010**	→	00110	→	**00100**

ii. Prove that if $v_{i_1} \rightarrow v_{i_2} \rightarrow \ldots \rightarrow v_{i_k}$ is a simple path in G, then

$$w_{i_1} \rightarrow e_{i_1,i_2} \rightarrow w_{i_2} \rightarrow \ldots \rightarrow w_{i_{k-1}} \rightarrow e_{i_{k-1},i_k} \rightarrow w_{i_k}$$

is a word ladder. To prove this is a word ladder, show that all adjacent words differ by exactly one letter and that no word appears twice.

Your proof in (ii) shows that if there is a simple path of k nodes in the original graph between v_i and v_j, there is a word ladder in our new dictionary of length 2k – 1 from w_i to w_j. The reverse is also true: if there is a word ladder of length 2k – 1 from w_i to w_j, then there is a simple path of k nodes from v_i to v_j. Intuitively, this statement is true because in a word ladder of length 2k – 1 that starts and ends at a node word, every other word in the word ladder is an node word. Those nodes, in sequence, give a path of k nodes in the original graph.

iii. Prove that in a word ladder of length 2k – 1 that begins at a word node w_i and ends at a word node w_j, the words alternate between node words and edge words.

iv. Prove that if $w_{i_1} \rightarrow e_{i_1,i_2} \rightarrow w_{i_2} \rightarrow \ldots \rightarrow w_{i_{k-1}} \rightarrow e_{i_{k-1},i_k} \rightarrow w_{i_k}$ is a word ladder of length 2k – 1 that begins and ends at a node word, then $v_{i_1} \rightarrow v_{i_2} \rightarrow \ldots \rightarrow v_{i_k}$ is a simple path from v_{i_1} to v_{i_k} of k nodes in graph G. To do so, prove that this sequence of nodes is a simple path by showing that there is an edge between each adjacent pair of nodes and that no node appears twice.

You have just proven the following:

- If there is a simple path in G from v_i to v_j of k nodes, then there is a word ladder of length $2k - 1$ from w_i to w_j.

- If there is a word ladder of length $2k - 1$ from w_i to w_j, then there is a simple path in G from v_i to v_j of k nodes.

In other words, we have a reduction from *ULONGPATH* to *LONGLADDER*! However, to finish everything off, we need to prove that this reduction can be done in polynomial time.

v. Prove that the reduction from *ULONGPATH* to *LONGLADDER* is a polynomial-time reduction. *(Hint: Prove that there are a polynomial number of words generated, each of which has polynomial size).*

Putting this all together, we have the following:

Theorem: LONGLADDER is NP-complete.
Proof: We have proven that *LONGLADDER* is in NP in part (i), so all we need to do is show that any problem in NP is polynomial-time reducible to it. We do this by reducing *ULONGPATH* , which is NP-complete, to *LONGLADDER*. Given input $\langle G, v_i, v_j, k \rangle$, construct the dictionary D using the indicated construction. Then, construct the input $\langle D, w_i, w_j, 2k - 1 \rangle$ to *LONGLADDER*. As proven in (ii) and (iv), there is a ladder of length $2k - 1$ from w_i to w_j using dictionary D iff there is a simple path of k nodes in G from v_i to v_j. As proven in (v), this reduction works in polynomial time. Thus we have a polynomial-time reduction from *ULONGPATH,* which is NP-complete, to *LONGLADDER,* so *LONGLADDER* is NP-complete. ■

Problem Four: Course Feedback (5 Points)

We want this course to be as good as it can be, and we'd really appreciate your feedback on how we're doing. For a free five points, please answer the following questions. We'll give you full credit no matter what you write (as long as you write something!), but we'd appreciate it if you're honest about how we're doing.

i. How hard did you find this problem set? How long did it take you to finish?

ii. Does that seem unreasonably difficult or time-consuming for a five-unit class?

iii. Did you attend Monday's problem session? If so, did you find it useful?

iv. **If you could change any one thing about this course, what would it be?**

v. **If we should keep any one thing about this course the same in future offerings, what would it be?**

Submission Instructions

There are four ways to submit this assignment:

1. Hand in a physical copy of your answers at the start of class.

2. Submit a physical copy of your answers in the filing cabinet in the open space near the handout hangout in the Gates building. If you haven't been there before, it's right inside the entrance labeled "Stanford Venture Fund Laboratories." There will be a clearly-labeled filing cabinet into which you can submit your homework.

3. Submit a physical copy of your answers in the drop-off box in the Gates basement. We'll send out an email announcement about the location of this box on the night of Friday, December 9.

4. Send an email with an electronic copy of your answers to **cs103@cs.stanford.edu**

Extra Credit Problem: P and NP (Worth an A+ in CS103, $1,000,000, and a Stanford Ph.D)

Prove or disprove: P = NP.

Congratulations! You have just finished the final problem set of the quarter!