CS/ECE 374 A ✧ Fall 2019
Final Exam Problem 1 Solution

For each of the following questions, indicate ***every*** correct answer by marking the "Yes" box, and indicate ***every*** incorrect answer by marking the "No" box. **Assume P ≠ NP.** If there is any other ambiguity or uncertainty, mark the "No" box.

There are **40** yes/no choices altogether. Each correct choice is worth +½ point; each incorrect choice is worth −¼ point; each checked "IDK" is worth +1/8 point.

(a) Which of the following statements is true for ***every*** language $L \subseteq \{0,1\}^*$?

[X] Yes	[ ] No	[ ] IDK	L^* is non-empty. — L^* always contains the empty string ε.
[ ] Yes	[X] No	[ ] IDK	L^* is regular. — Let $L = \{0^{n^2}1 \mid n \geq 0\}$.
[ ] Yes	[X] No	[ ] IDK	L^* is decidable. — Let $L = X2$, where X is any undecidable subset of $\{0,1\}^*$.
[X] Yes	[ ] No	[ ] IDK	If L is NP-hard, then L is not regular. — Every regular language is in P, because a DFA is a linear-time algorithm.
[ ] Yes	[X] No	[ ] IDK	If L is not regular, then L is undecidable. — The non-regular language $\{0^n1^n \mid n \geq 0\}$ is decidable.
[ ] Yes	[X] No	[ ] IDK	If L is context-free, then L is infinite. — Consider the grammar $S \to 1$
[X] Yes	[ ] No	[ ] IDK	L is the intersection of two regular languages if and only if L is regular. — Standard closure properties. In particular, $L = L \cap L$.
[ ] Yes	[X] No	[ ] IDK	L is decidable if and only if L^* is decidable. — $(\text{HALT}+0+1)^* = (0+1)^*$.
[X] Yes	[ ] No	[ ] IDK	L is decidable if and only if its reversal $L^R = \{w^R \mid w \in L\}$ is decidable. (Recall that w^R denotes the reversal of the string w.) — You can write an algorithm to reverse a string.
[ ] Yes	[X] No	[ ] IDK	L is decidable if and only if its complement $\overline{L}$ is undecidable. — In fact L is decidable if and only if $\overline{L}$ is decidable.

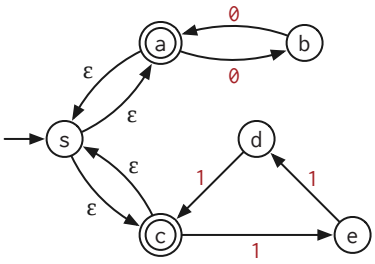
(b) Consider the following sets of undirected graphs:

- TREES is the set of all connected undirected graphs with no cycles.
- 3COLOR is the set of all undirected graphs that can be properly colored using at most 3 colors.

(For concreteness, assume that in both of these languages, graphs are represented by their adjacency matrices.) Which of the following ***must*** be true, assuming P≠NP?

[X] Yes	[ ] No	[ ] IDK	TREES $\in NP$ — To verify that a graph is a tree, just look for cycles.
[X] Yes	[ ] No	[ ] IDK	TREES $\subseteq$ 3COLOR — In fact, every graph is 2-colorable; see Midterm 2.
[X] Yes	[ ] No	[ ] IDK	There is a polynomial-time reduction from TREES to 3COLOR — Look for cycles and ignore the 3COLOR subroutine.
[ ] Yes	[X] No	[ ] IDK	There is a polynomial-time reduction from 3COLOR to TREES — That would imply that TREES is NP-hard.
[ ] Yes	[X] No	[ ] IDK	TREES is NP-hard. — And *that* would imply that P=NP!

(c) Let M be the following NFA:



Which of the following statements about M are true?

[X] Yes	[ ] No	[ ] IDK	M accepts the empty string ε — via ε-transition to either a or c.
[ ] Yes	[X] No	[ ] IDK	$\delta^*(s, \texttt{010}) = \{s, a, c\}$ — In fact, $\delta^*(s, \texttt{010}) = \emptyset$.
[X] Yes	[ ] No	[ ] IDK	ε-reach$(a) = \{s, a, c\}$
[X] Yes	[ ] No	[ ] IDK	M rejects the string 11100111000 — because $\delta^*(s, \texttt{11100111000}) = \{b\}$.
[ ] Yes	[X] No	[ ] IDK	$L(M) = (\texttt{00})^* + (\texttt{111})^*$ — Close, but not quite. In fact, $L(M) = (\texttt{00} + \texttt{111})^*$.

(d) Which of the following languages over the alphabet $\Sigma = \{0, 1\}$ are ***regular***? Recall that $\#(a, w)$ denotes the number of times symbol a appears in string w.

[X] Yes [ ] No [ ] IDK — The intersection of two regular languages — standard closure properties

[ ] Yes [X] No [ ] IDK — $\{w \in \Sigma^* \mid |w| \text{ is prime}\}$ — Intuitively, you can't detect primes without counting

[X] Yes [ ] No [ ] IDK — $\{w \in \Sigma^* \mid \#(0, w) + \#(1, w) > 374\} = (0 + 1)^{375}(0 + 1)^*$

[ ] Yes [X] No [ ] IDK — $\{w \in \Sigma^* \mid \#(0, w) - \#(1, w) > 374\}$ — via easy fooling set argument (or counting intuition)

[X] Yes [ ] No [ ] IDK — The language generated by the context-free grammar $S \to 0S \mid 10S \mid \varepsilon$ — This is the language $(0 + 10)^*$.

(e) Which of the following languages or problems are ***decidable***?

- [X] Yes [ ] No [ ] IDK — Σ^* — accept everything.
- [X] Yes [ ] No [ ] IDK — 3SAT — An exponential-time algorithm is still an algorithm.
- [ ] Yes [X] No [ ] IDK — $\{\langle M \rangle \mid M$ accepts every string whose length is prime$\}$ — Rice's theorem!
- [X] Yes [X] No [X] IDK — $\{\langle M \rangle \mid M$ accepts all strings in 0^* and rejects all strings in $1^*\}$

  Everyone gets credit for this one. We intended this to be a nontrivial question about the accept/reject behavior of programs. But as one student pointed out, no machine both accepts and rejects the empty string, so this language is actually empty!
- [X] Yes [ ] No [ ] IDK — $\{\langle M \rangle \mid M$ is a Turing machine with at least two states$\}$ — This is a question about the *syntax* of programs. Consider the similar question "Is this Python program more than one line long?"

(f) Which of the following languages or problems can be proved undecidable ***using Rice's Theorem***?

- [ ] Yes [X] No [ ] IDK — Σ^* — decidable
- [ ] Yes [X] No [ ] IDK — 3SAT — decidable
- [X] Yes [ ] No [ ] IDK — $\{\langle M \rangle \mid M$ accepts every string whose length is prime$\}$ — Let Y accept everything, and let N accept nothing.
- [ ] Yes [X] No [ ] IDK — $\{\langle M \rangle \mid M$ accepts all strings in 0^* and rejects all strings in $1^*\}$ — This language is empty, and therefore decidable. Also, Rice's theorem considers only the languages that programs *accept*.
- [ ] Yes [X] No [ ] IDK — $\{\langle M \rangle \mid M$ is a Turing machine with at least two states$\}$ — decidable

---

(g) Suppose we want to prove that the following language is undecidable.

$$\textsc{Marvin} := \{\langle M \rangle \mid M \text{ rejects an infinite number of strings}\}$$

Professor Prefect, your instructor in Vogon Poetry and Knowing Where Your Towel Is, suggests a reduction from the standard halting language

$$\textsc{Halt} := \{(\langle M \rangle, w) \mid M \text{ halts on inputs } w\}.$$

Specifically, suppose there is a program ParanoidAndroid that decides Marvin. Professor Prefect claims that the following algorithm decides Halt.

```
DecideHalt(⟨M⟩, w):
  Write code for the following algorithm:
      HoopyFrood(x):
        run M on input w
        if x = DONTPANIC
            return True
        else
            return False
  return ParanoidAndroid(⟨HoopyFrood⟩)
```

Which of the following statements is true for all inputs $(\langle M \rangle, w)$?

Yes	~~No~~	IDK	If M accepts w, then HoopyFrood accepts BEEBLEBROX. — Rejects, actually, because BEEBLEBROX $\neq$ DONTPANIC.
~~Yes~~	No	IDK	If M rejects w, then HoopyFrood rejects BEEBLEBROX. — Again, because BEEBLEBROX $\neq$ DONTPANIC.
Yes	~~No~~	IDK	If M hangs on w, then HoopyFrood rejects DONTPANIC. — In fact, if M hangs on w, then HoopyFrood hangs on every input.
Yes	~~No~~	IDK	ParanoidAndroid accepts ⟨HoopyFrood⟩. — Only if M halts on w. If M hangs on w, then HoopyFrood does not reject any strings, and therefore ParanoidAndroid rejects ⟨HoopyFrood⟩.
~~Yes~~	No	IDK	DecideHalt decides Halt; that is, Professor Prefect's proof is correct.

Rubric: +½ for each correct answer, -½ for each incorrect answer, +1/8 for each IDK. No negative scores. No explanations necessary.

CS/ECE 374 A ✧ Fall 2019
Final Exam Problem 2 Solution

You are planning a hiking trip in Jellystone National Park over winter break. You have a complete map of the park's trails; the map indicates that some trail segments have a high risk of bear encounters. All visitors to the park are required to purchase a canister of bear repellent. You can safely traverse a high-bear-risk trail segment only by *completely* using up a *full* canister of bear repellent. The park rangers have installed refilling stations at several locations around the park, where you can refill empty canisters at no cost. The canisters themselves are expensive and heavy, so you cannot carry more than one. Because the trails are narrow, each trail segment allows traffic in only one direction.

You have converted the trail map into a directed graph $G = (V, E)$, whose vertices represent trail intersections, and whose edges represent trail segments. A subset $R \subseteq V$ of the vertices indicate the locations of the *R*epellent *R*efilling stations, and a subset $B \subseteq E$ of the edges are marked as having a high risk of *B*ears. Your campsite appears on the map as a particular vertex $s \in V$, and the visitor center is another vertex $t \in V$.

(a) Describe and analyze an algorithm to decide if you can safely walk from your campsite s to the visitor center t. Assume there is a refill station at your camp site, and another refill station at the visitor center.

(b) Describe and analyze an algorithm to decide if you can walk safely from any refill station any other refill station. In other words, for *every* pair of vertices u and v in R, is there a safe path from u to v?

Solution (a): Let $G = (V, E)$ be the input graph. We construct a new directed graph $G' = (V', E')$ as follows:

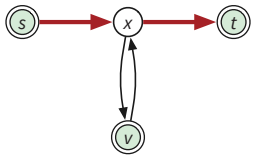
- $V' = V \times \{0, 1\}$ — Each vertex (v, i) denotes being at vertex v with i full cans of bear repellent.
- E' contains four types of edges:
  - $(u, 1) \to (v, 0)$ for each edge $u \to v \in B$ — dangerous trails with bear repellent
  - $(u, 1) \to (v, 1)$ for each edge $u \to v \notin B$ — safe trails with bear repellent
  - $(u, 0) \to (v, 0)$ for each edge $u \to v \notin B$ — safe trails without bear repellent
  - $(v, 0) \to (v, 1)$ for each vertex $v \in R$ — refill stations

Paths in G' correspond exactly to safe paths in G.

We need to decide if $(s, 1)$ can reach $(t, 1)$ in G'. We an answer this question using whatever-first search in $O(V' + E') = \boldsymbol{O(V + E)}$ ***time***. ■

Rubric: 5 points: standard graph reduction rubric. −½ for using Dijkstra instead of WFS. Max 3 points for $O(VE \log V)$ or $O(V^3)$-time algorithm; Max 2 for any slower algorithm.

Several students described algorithms that gave incorrect results on the following graph, where $R = \{s, t, v\}$ and $B = \{s \to x, x \to t\}$. Every safe walk from s to t in this graph visits x at least twice; in particular, the walk $s \to x \to v \to x \to t$ is safe.



Solution (b): First we construct the same graph G' as in part (a).

We need to determine whether there is a path in G' from $(u, 1)$ to $(v, 1)$, for every pair of vertices $u, v \in R$. In other words, we need to determine whether all nodes $(u, 1)$ such that $u \in R$ lie in the same strong component of G'.

We can compute the strong components of G' in $O(V' + E') = O(V + E)$ time, after which we can check that all refill nodes $(u, 1)$ lies in the same strong component.

Altogether, the algorithm runs in $O(V' + E') = \boldsymbol{O(V + E)}$ ***time***. ■

Solution (b): There is a safe path from any refill station to any other refill station if and only if (1) there is a safe path from s to every refill station and (2) there is a safe path from every refill station to s.

We construct the same graph G' as in part (a).

Using a single whatever-first search from $(s, 1)$, we mark all vertices that are reachable from $(s, 1)$ in G'. Then using a second whatever-first search *in the reversal of* G', we mark all vertices that can reach $(s, 1)$. Finally, we verify by brute force whether every vertex $(r, 1)$ such that $r \in R$ is marked twice.

Altogether, the algorithm runs in $O(V' + E') = \boldsymbol{O(V + E)}$ ***time***. ■

Solution (b) (4/5): Give every edge in B weight 1 and every edge in $E \setminus B$ weight 0, and then compute the shortest-path distance from every refill station to every other refill station, either by running Floyd-Warshall or by running Dijkstra's algorithm at each vertex $r \in R$.

Now construct a new graph G' whose vertices are the refill station, and whose edges $r \to r'$ indicate that the distance from r to r' is at most 1. Finally, if G' is strongly connected, return TRUE; otherwise, return FALSE.

Altogether, this algorithm runs in $\boldsymbol{O(V^3)}$ ***time*** if we use Floyd-Warshall, or $\boldsymbol{O(VE \log V)}$ ***time*** if we use Dijkstra. (We can get rid of the $\log V$ factor with more care.) ■

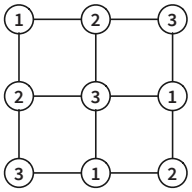
Solution (b) (3/5): Run the algorithm from part (a) for every pair $s, t \in R$. The overall running time is $O(V^2(V + E))$. ■

Rubric: 5 points: standard graph reduction rubric. I'm sure these are not the only correct solutions.

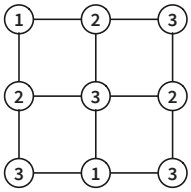
- −½ for using Dijkstra instead of WFS.
- Max 4 points for $O(V(V + E))$ or $O(V^3)$ time (for example, consider each refill station separately)
- Max 3 points for $O(V^2(V + E))$ time (for example, consider each pair of refill stations separately)

CS/ECE 374 A ✧ Fall 2019
Final Exam Problem 3 Solution

Recall that a *proper 3-coloring* of a graph G assigns each vertex of G one of three colors, so that every edge of G has endpoints with different colors. A proper 3-coloring is *balanced* if each color is assigned to *exactly* the same number of vertices.



A balanced proper 3-coloring.



A proper 3-coloring that is not balanced.

The BALANCED3COLOR problem asks, given an undirected graph G, whether G has a balanced proper 3-coloring. ***Prove*** that BALANCED3COLOR is NP-hard.

Solution: We prove this problem is NP-hard by reduction from the standard 3COLOR problem.

Let $G = (V, E)$ be a given undirected graph. We construct a new graph G' by adding $2V$ isolated vertices to G. Let W denote the set of $2V$ new isolated vertices. I claim that G has a proper 3-coloring if and only if G' has a balanced proper 3-coloring.

$\Longrightarrow$ Suppose G has a proper 3-coloring, using the colors red, green, and blue. Let r be the number of red vertices, let b be the number of blue vertices, and let g be the number of green vertices, the definition of "3-coloring" implies that $r + b + g = V$. Color $g + b$ vertices in W red, $r + b$ vertices in W green, and the remaining $r + g$ vertices in W blue. Then G' has exactly $r + b + g$ vertices of each color. Thus, G' has a balanced 3-coloring.

$\Longleftarrow$ Suppose G' has a balanced 3-coloring. Then every edge in G' has endpoints with different colors. But every edge in G' is actually an edge in G, so every edge in G has endpoints with different colors. Thus, the restriction of the balanced 3-coloring to G is a proper 3-coloring of G.

We can easily construct G' from G in polynomial time. ■

Solution: We prove this problem is NP-hard by reduction from the standard 3COLOR problem.

Let $G = (V, E)$ be a given undirected graph. We construct a new graph G' consisting of three disjoint copies of G (called G_1, G_2, and G_3) with no edges between them. For each vertex v in G, let v_i denote the copy of v in subgraph G_i. I claim that G has a proper 3-coloring if and only if G' has a balanced proper 3-coloring.

$\Longrightarrow$ Suppose G has a proper 3-coloring, using the colors red, green, and blue. For each vertex v in G, we color the corresponding nodes in G' as follows:

- If v is red, color v_1 red, v_2 green, and v_3 blue.
- If v is green, color v_1 green, v_2 blue, and v_3 red.
- If v is blue, color v_1 blue, v_2 red, and v_3 green.

Every edge in G' has endpoints with two distinct colors, because the corresponding edge in G has endpoints with two distinct colors. So we have a proper 3-coloring for G'. Moreover, if G has r red vertices, b blue vertices, and g green vertices, then G' has $r+b+g$ vertices of each color; thus, we actually have a *balanced* proper 3-coloring of G'.

$\Longleftarrow$ Suppose G' has a balanced 3-coloring. Then subgraph G_1 has a proper 3-coloring. But G_1 is just a copy of G. We conclude that G is 3-colorable.

We can easily construct G' from G in polynomial time. ■

Rubric: 10 points: standard NP-hardness rubric. These are not the only correct solutions.

CS/ECE 374 A ✧ Fall 2019
Final Exam Problem 4 Solution

For each of the following languages, state whether the language is regular or not, and then justify your answer as follows:

- If the language is regular, ***either*** give an regular expression that describes the language, ***or*** draw/describe a DFA or NFA that accepts the language. You do not need to prove that your automaton or regular expression is correct.
- If the language is not regular, ***prove*** that the language is not regular.

[Hint: Exactly one of these languages is regular.]

(a) $\{xy \mid x, y \in \Sigma^+ \text{ and } x \text{ and } y \text{ are both palindromes}\}$

(b) $\{xy \mid x, y \in \Sigma^+ \text{ and } x \text{ is not a palindrome}\}$

Solution (a): This language L is ***not regular.***

- Let F be the infinite set 0^*.
- Let x and y be two arbitrary strings in F.
- Then $x = 0^i$ and $y = 0^j$ for some non-negative integers $i \neq j$.
- Let $z = 010^{i+1}11$.
- Then $xz = 0^{i+1}10^{i+1}11 = 0^{i+1}10^{i+1} \bullet 11 \in L$.
- But $yz = 0^{j+1}10^{i+1}11 \notin L$. — The only non-empty palindrome suffixes of yz are 1 and 11, because every longer prefix starts with at most one 1. For the suffix 1, the corresponding prefix starts with 0 and ends with 1; for the suffix 11, the corresponding prefix $0^{j+1}10^{i+1}$ is not a palindrome, because $i \neq j$.

We conclude that F is a fooling set for L. Because F is infinite, L cannot be regular. ■

Solution (a): This language L is ***not regular.***

- Let F be the infinite set 0^+11.
- Let x and y be two arbitrary strings in F.
- Then $x = 0^i11$ and $y = 0^j11$ for some positive integers $i \neq j$.
- Let $z = 0^i1$.
- Then $xz = 0^i110^i1 = 0^i110^i \bullet 1 \in L$.
- But $yz = 0^j110^i1 \notin L$. — The only non-empty palindrome suffixes of yz are 1 and 10^i1. For the suffix 1, the corresponding prefix $0^{j+1}10^{i+1}$ is not a palindrome, because $i \neq j$. For the suffix 10^i1, the corresponding prefix 0^i1 is not a palindrome because it starts with 0 and ends with 1.

We conclude that F is a fooling set for L. Because F is infinite, L cannot be regular. ■

Solution (a): This language L is ***not regular.***

- Let F be the infinite set $(\texttt{100})^+$.
- Let x and y be two arbitrary strings in F.
- Then $x = (\texttt{100})^i$ and $y = (\texttt{100})^j$ for some positive integers $i \neq j$.
- Let $z = (\texttt{001})^i\texttt{00000}$.
- Then $xz = (\texttt{100})^i(\texttt{001})^i\texttt{00000} = (\texttt{100})^i(\texttt{001})^i \bullet \texttt{00000} \in L$.
- But $yz = (\texttt{100})^j(\texttt{001})^i\texttt{00000} \notin L$. — Every palindrome prefix of yz starts and ends with 1. If this prefix does not include every 1, then the corresponding suffix starts with a run of at most four 0s, but ends with a run of five 0s, so it is not a palindrome. On the other hand, the prefix $(\texttt{100})^j(\texttt{001})^i$ is not a palindrome, because $i \neq j$.

We conclude that F is a fooling set for L. Because F is infinite, L cannot be regular. ■

Rubric: 5 points = 1 for "not regular" + 4 for fooling set argument. These are not the only correct arguments for these fooling sets, and these are not the only correct fooling sets.

Solution (b): This language L is ***regular***. In particular, L is the set of all strings that contain both a 0 and a 1 *before their last character*. Thus, this language is described by the following regular expression:

$$(\texttt{0}+\texttt{1})^*(\texttt{01}+\texttt{10})(\texttt{0}+\texttt{1})^*(\texttt{0}+\texttt{1})$$

Let w be any string that contains both a 0 and a 1 before its last character. Write $w = xy$, where x is the *shortest* prefix of w that contains both 0 and 1. Then either $x \in \texttt{0}^+\texttt{1}$, or $x \in \texttt{1}^+\texttt{0}$. In both cases, x is a prefix of w that is not a palindrome. Moreover, because x excludes the last character of w, the suffix y is non-empty.

On the other hand, suppose w is a string in the complement of L. Then *every* proper prefix of a string w is a palindrome. In particular, for every integer $k < |w|$, the kth symbol in w is equal to the first symbol in w. ■

Rubric: 5 points = 2 for "regular" + 3 for regular expression/DFA. (So correctly stating "not regular" for (a) and "regular" for (b) is worth a total of 3 points.) This is not the only correct regular expression.

CS/ECE 374 A ✧ Fall 2019
Final Exam Problem 5 Solution

(a) Recall that a *palindrome* is any string that is equal to its reversal, like REDIVIDER or POOP. Describe an algorithm to find the length of the longest subsequence of a given string that is a palindrome.

(b) A *double palindrome* is the concatenation of two *non-empty* palindromes, like POOPREDIVIDER or POOPPOOP. Describe an algorithm to find the length of the longest subsequence of a given string that is a *double* palindrome. *[Hint: Use your algorithm from part (a).]*

For both algorithms, the input is an array $A[1..n]$, and the output is an integer. For example, given the string MAYBEDYNAMICPROGRAMMING as input, your algorithm for part (a) should return 7 (for the palindrome subsequence NMRORMN), and your algorithm for part (b) should return 12 (for the double palindrome subsequence MAYBYAMIRORI).

Solution (a): Let $LPS(i,j)$ denote the length of the longest palindrome subsequence of $A[i..j]$. We need to compute $LPS(1,n)$. This function obeys the following recurrence:

$$LPS(i,j) = \begin{cases} 0 & \text{if } i > j \\ 1 & \text{if } i = j \\ \max\{2 + LPS(i+1,j-1),\ LPS(i+1,j),\ LPS(i,j-1)\} & \text{if } A[i] = A[j] \\ \max\{LPS(i+1,j),\ LPS(i,j-1)\} & \text{otherwise} \end{cases}$$

We can memoize this function into a two-dimensional array $LPS[1..n, 1..n]$. We fill this array using two nested loops, increasing i in the outer loop, and decreasing j in the inner loop, in $\boldsymbol{O(n^2)}$ ***time***. ■

Rubric: 5 points: standard dynamic programming rubric. No penalty for greedy optimization in the recursive cases.

Solution (b): First fill the array $LPS[1..n, 1..n]$ using the algorithm from part (a), and then compute the maximum of $LPS[1,i] + LPS[i+1,n]$ over all i from 1 to $n-1$ in $O(n)$ additional time. The overall algorithm runs in $\boldsymbol{O(n^2)}$ ***time***. ■

Solution (b) (4/5): Let $Best(i) = LPS(1,i) + LPS(i+1,n)$. For each index i, we can compute $Best(i)$ by calling the algorithm from part (a) twice. By wrapping these calls in a for-loop, we can compute $\max_i Best(i)$ in $\boldsymbol{O(n^3)}$ ***time***. ■

Rubric: 5 points for $O(n^2)$-time algorithm = 2 for correctly invoking part (a) + 2 for considering all splits + 1 for time analysis. Max 4 points for $O(n^3)$ time, 3 points for $O(n^4)$ time, 2 points for any slower correct algorithm.

CS/ECE 374 A ✧ Fall 2019
Final Exam Problem 6 Solution

Let M be an arbitrary DFA. Describe and analyze an efficient algorithm to decide whether M rejects an infinite number of strings.

Solution: Assume the input alphabet is $\Sigma = \{0, 1\}$. We treat the DFA $M = (Q, s, A, \delta)$ as a directed graph with $|Q|$ vertices and at most $2|Q|$ edges. We need to decide whether this graph contains a cycle that is reachable from the start state s and that can reach some non-accepting state $q \notin A$.

- Find and delete any states that are unreachable from s, using whatever-first search, in $O(V + E) = O(|Q|)$ time. If every state reachable from s is an accepting state, return FALSE immediately.
- Add a new target node t, with edges from every remaining non-accepting state. Find and delete any states that cannot reach t, using whatever-first search in the reversal of M, in $O(V + E) = O(|Q|)$ time.
- If the remaining graph M contains a cycle, return FALSE; otherwise (if the remaining graph is a dag), return TRUE. We can detect cycles in $O(V + E) = O(|Q|)$ time via depth-first search/topological sort.

The overall algorithm runs in $O(V + E) = \boldsymbol{O(|Q|)}$ ***time***. ■

Solution: Assume the input alphabet is $\Sigma = \{0, 1\}$. We treat the DFA $M = (Q, s, A, \delta)$ as a directed graph with $|Q|$ vertices and at most $2|Q|$ edges.

- Add a new *terminal* vertex t to M, with edges from every non-accepting state.
- Construct the strong component graph G of M. For each vertex q in M, let $[q]$ denote both the strong component of M that contains q and the corresponding vertex of G.
- Give each edge $[p] \to [q]$ in G weight 1 if strong component $[p]$ contains a cycle (that is, either more than one vertex or a self-loop) and weight 0 otherwise.
- Finally, compute the longest path from $[s]$ to $[t]$ in G in linear time, using the linear-time algorithm described in class. This path has positive length if and only if M rejects an infinite number of strings.

The overall algorithm runs in $O(V + E) = \boldsymbol{O(|Q|)}$ ***time***. ■

Rubric: 10 points =

- 5 for (stated or implied) characterization: M contains a non-accepting state that is reachable from a cycle that is reachable from s
  - –1 for ignoring reachability from s
  - –1 for cycle *containing* (instead of *reaching*) a non-accepting state

  - – 1 for "self-loop" instead of "cycle"
  - – ½ for "strong component" instead of "cycle" — A strong component with only one vertex need not have a cycle.
  - – ½ for "strong component with more than one vertex" instead of "cycle" — DFAs can have self-loops.
  - ◦ No penalty for "strong component containing a cycle" or "strong component with either more than one state or a self-loop" instead of "cycle"
- 3 for algorithm
  - ◦ No credit for algorithm if characterization is worth less than 2½ points
- 2 for time analysis in terms of $|Q|$
  - – 1 for reporting time in terms of V and E
  - – ½ for reporting time in terms of $|Q|$ and "number of transitions"
  - ◦ No credit for time analysis if characterization or algorithm is far from correct

Maximum 8 points for $O(|Q|^2)$ time (considering each non-accept state separately, or using Bellman-Ford to detect negative cycles); scale partial credit.

This is not the only correct solution. Most common errors imply incorrect results for either or both of the following three-state DFAs. The DFA on the left rejects all strings in 0^*1; the DFA on the right does not reject any strings at all.