PONTIFICIA UNIVERSIDAD CATOLICA DE CHILE
ESCUELA DE INGENIERIA
DEPARTAMENTO DE CIENCIA DE LA COMPUTACION

Complexity Theory - IIC3242
Homework 1
Deadline: Monday, 25th of April

1 Programming Turing machines [1 point]

The web page `https://turingmachinesimulator.com/` provides a programmable Turing machine. Here you can determine the number of tapes used, the vocabulary, etc. In this task you are asked to provide a Turing machine that decides the problem:

$$A = \{0^k 1^k \mid k \geq 0\},$$

and that *runs in logarithmic space*. You are allowed to use any number of tapes and any alphabet. However, do make sure that you: a) decide the said language; and b) that the space used on work tapes is logarithmic in the size of the input.

2 An easy reduction [2 points]

Consider the following problem:

$$\text{AUTOMATA} = \{\langle N, w\rangle \mid N \text{ is a non deterministic finite state automaton that accepts the word } w\}.$$

a) Show that AUTOMATA is NLOGSPACE-hard by reducing PATH to AUTOMATA. Recall:

$$\text{PATH} = \{\langle G, s, t\rangle \mid G \text{ a graph with a path from } s \text{ to } t\}$$

b) Program the above reduction using the Turing machine from `https://turingmachinesimulator.com/`. Be careful how you represent graphs and automata in order to make the programs simpler.

3 A bit more challenging reduction (but still easy) [3 points]

In the CONNECTING PATHS problem we are given a directed graph G and a set of pairs $\{(s_1, t_1, \ldots, s_k, t_k)\}$. We are asked if there are paths $P_1, \ldots, P_k$ in our graph G such that:

- P_i is a path from s_i to t_i; and
- P_i and P_j do not have any nodes in common (apart from possibly the starting/ending ones), when $i \neq j$.

Show that the CONNECTING PATHS problem is NP-complete. (Hint: a reduction from 3SAT is rather straightforward. Just think how many paths do you need to code one variable (or its negation).)

4 Some easy thinking [6 points]

Web databases. A *web database* is directed graph where each edge is labelled by a label from some finite alphabet Σ. Formally, a web database over an alphabet Σ can be defined as a tuple $G = (V, E)$, where V is a finite set of nodes and $E \subseteq V \times \Sigma \times V$ is a finite set of edges labelled by a letter from Σ. An example of a web database is given in the following image:

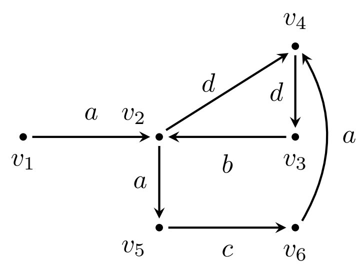

Figure 1: Here the nodes are $V = \{v_1, v_2, v_3, v_4, v_5\}$ and the edges are $E = \{(v_1, a, v_2), (v_2, a, v_5), \ldots\}$.

Web queries. In order to explore a web database we will use the language of *web queries* that allows us to see how the nodes of the graph are connected. Formally, a *web query* q is an expression of the form

$$q = x \xrightarrow{e} y,$$

where e is a regular expression over Σ. The intended meaning of a web query q is to extract all pairs (x, y) of nodes in G that are connected by a path P such that, when we read the word formed by the edge labels along this path belongs to the language of e. We formalize these concepts below.

Semantics of web queries. A path in a web database $G = (V, E)$ is a sequence

$$P = v_1, a_1, v_2, a_2, v_3 \ldots, a_k, v_{k+1},$$

where each v_i is a node in V and $a_i \in \Sigma$ such that $(v_i, a_i, v_{i+1}) \in E$ is an edge of this web database. The label of the path P above is defined as $\lambda(P) = a_1 \cdots a_k$, that is, as the word that is obtained by concatenating the letters on the edges of P. Let $q = x \xrightarrow{e} y$ be a web query. We define the *evaluation of* q *over the web graph* G, denoted by $\text{EVAL}(q, G)$ as the set of all pairs (v, v') of nodes in G such that there is some path P starting with v and ending with v' such that $\lambda(P) \in L(e)$, where $L(e)$ denotes the language defined by the regular expression e.

A quick example. Consider the web database G from the above figure. If we define our web query as $q = x \xrightarrow{ac^*} y$ it is easy to see that $\text{EVAL}(q, G) = \{(v_1, v_2), (v_2, v_5), (v_2, v_6), (v_6, v_4)\}$. For example, (v_2, v_6) is in the answer to our query because for $P = v_2 a v_5 c v_6$ we have that $\lambda(P) = ac$ is in the language of the expression ac^*.

Computing the answer. In order to make web queries useful we need to be able to compute them. One way of doing this would be to design an algorithm that, when given q and G as its input, computes the set $\text{EVAL}(q, G)$. Since we are dealing with decision problems we will be interested in the following language (which can then be used to compute the entire answer):

$$\text{WEBCOMP} = \{\langle q, G, (v, v') \rangle \mid (v, v') \in \text{EVAL}(q, G)\}.$$

In this assignment you are asked to prove the following:

a) [2 point] Show that the language WEBCOMP is NLOGSPACE-complete. Here you might want to think how can you view graphs and regular expressions as the same object. For some inspiration you might want to take a look at Section 2.3 in [1] (but do not expect a complete solution).

b) [1 point] Assume that when defining the semantics of web queries we only allow paths P that never repeat a node (i.e. they are simple paths). Call this problem SIMPLEWEBCOMP. Show that the problem SIMPLEWEBCOMP is NP-complete. You might want to use the fact that the following problem, called EVEN, is NP-complete:

$$\text{EVEN} = \{\langle G, s, t\rangle \mid G \text{ has a } \textbf{simple} \text{ path from } s \text{ to } t \text{ of even length (number of edges) }\}.$$

c) [1 point] Assume that when defining the semantics of web queries we only allow paths P that never repeat an edge (but they may repeat a node). Call this problem EDGEWEBCOMP. Show that the problem EDGEWEBCOMP is also NP-complete.

More expressive web queries. Web queries allow us to search web databases using paths, but we are often interested in more complicated patterns occurring in the database. For instance, we might want to check if there are two different paths connecting two nodes, or if there are three nodes connected into a clique labelled a certain way. For this we use *conjunctive web queries*. Formally, conjunctive web queries are expressions of the form:

$$\varphi(x_1, \dots, x_n) = \exists y_1 \dots \exists y_m \bigwedge_{i=1\dots k} z_i \xrightarrow{e_i} u_i,$$

where $Var = \{x_1, \dots, x_n, y_1, \dots, y_m\}$ are all the variables appearing in the expression (and are pairwise distinct), $u_i, z_i \in Var$ and e_i is a regular expressions. Intuitively, conjunctive web queries are the closure of web queries under conjunction and existential quantification, since we are claiming that there exist some nodes $(y_1, \dots, y_m)$, for which a conjunction of web queries holds true (with the appropriate values for $x_1, \dots, x_n$). Formally, for the query φ above, we define the *evaluation of φ over a web database G*, denoted by $\text{EVAL}(\varphi, G)$, as the set of all n-tuples of nodes $(a_1, \dots, a_n)$, where:

1. there exists some m-tuple of nodes $b_1, \dots, b_m$ such that:
2. when we replace x_i with a_i, for $i \leq n$, and we replace y_i with b_i, for $i \leq m$,
3. we have that $(v_i, v_i') \in \text{EVAL}(z_i \xrightarrow{e_i} u_i, G)$, where v_i replaces z_i and v_i' replaces u_i, for $i \leq k$ (recall that all z_i, u_i are amongst $x_i s$ and y_is).

Some examples. Consider the following conjunctive web query:

$$\varphi(x_1) = \exists y_1 \exists y_2 (x_1 \xrightarrow{ad^*} y_1) \wedge (x_1 \xrightarrow{a^*c} y_2) \wedge (y_2 \xrightarrow{a} y_1).$$

The query asks for all the nodes x_1 such that there exist two nodes, y_1 and y_2, with the following three properties:

1. x_1 is connected with y_1 by a path labelled by (a word in) ad^*,
2. x_1 is connected with y_2 by a path labelled by (a word in) a^*c,
3. y_2 is connected to y_1 by a path labelled a.

When evaluated over the web database G from Figure 1, this query will return the node v_1 and nothing else. This holds because we can substitute v_1 for x_1 and witness y_1 by v_4 and y_2 by v_6.

The evaluation problem. Similarly as for ordinary web queries, here we are interested in the following problem:

$$\text{CONJWEBCOMP} = \{\langle \varphi(x_1, \dots, x_n), G, (a_1, \dots, a_n)\rangle \mid (a_1, \dots, a_n) \in \text{EVAL}(\varphi, G)\}.$$

For the final part of this assignment you are asked to show the following:

d) [2 point] Prove that the problem CONJWEBCOMP is NP-complete. For the lower bound you might want to think about the 3-colorability problem (but make sure to define it properly if you use this reduction). SAT and 3SAT are also feasible candidates.

References

[1] Moshe Y. Vardi, *An Automata-Theoretic Approach to Linear Temporal Logic*, `http://people.na.infn.it/~bene/TSV/LTL-readings/Vardi_automata-theoretic-and-LTL.pdf`