Monash University
Faculty of Information Technology
2nd Semester 2023

FIT2014
Exercises 10
Complexity: P, NP, Polynomial Time Reductions

ASSESSED PREPARATION: Question 2.
You must provide a serious attempt at parts (a)–(g) of this question at the start of your class.

---

1. Recall Exercise Sheet 7, Question 4, where you built a Turing machine that interchanges the first and last letters of the input string. Review the discussion in part (d) of the time complexity $t(n)$.

Justify the claim in the solutions that this time complexity satisfies $t(n) = O(n)$.

2. This question is about developing big-O bounds for the time complexity of constructing φ_G from G, in Assignment 1, Problem 2.

- Assume that, as in the Assignment, the input graph is given as an edge list.
- Assume that G has no isolated vertices.
- Assume that G is a **simple graph**, i.e., that it has no loops or multiple edges.
- Use the expression from Problem 3.
- Assume that the time taken to construct each clause is at most a constant.

(a) Give an upper bound for the number n of vertices of G in terms of the number m of edges of G.
(b) Express the time complexity in big-O as a function of m.
(c) Give an upper bound for m in terms of n.
(d) Express the time complexity in big-O as a function of n.
(e) The actual length of G, as an input to this construction, is taken to be the number of characters in its input file, ignoring all spaces. Give a lower bound for the length of G in terms of m.
(f) Express the time complexity of the algorithm in big-O notation as a function of the length of G (which we can denote by $|G|$).
(g) We can now see that this algorithm, for $G \mapsto \varphi_G$, is a mapping reduction. What language does it reduce <u>from</u> and what language does it reduce <u>to</u>? What type of mapping reduction is it?

(h) Give an upper bound for the length of G in terms of m and n.
(i) Prove that the length of G is $O(m \log m)$.

3.

You are running a program with running time an^5, where a is a constant.

(a) Assuming Moore's Law, how long do you have to wait before your program can solve inputs that are four times as large as those you can solve now, in the same time?

(b) Your friend has a program that runs in time 2^n. Give a lower bound on how long she has to wait, under the same scenario. (Assume for this part that your input size is at least 40.)

4.

Suppose you work in bioinformatics, and you have written a program that takes, as input, a DNA string, regarded as a sequence of *bases*, where each base is one of C,G,A,T. Your program runs in time $5n^3$, where n is the length of the input string.

Now, your boss insists on measuring string length in bits, where each of the four bases is represented by two bits.

(a) How would you state your program's running time, if n now represents the number of *bits* in the input string?

(b) You rush off to change the description of the program's running time in the documentation. You discover that, in your team, documentation gives all running times in big-O notation. What changes do you need to make to the big-O statement of the running time?

5.

Suppose algorithm A takes an adjacency matrix[1] of a graph as input and solves a particular graph-theoretic problem Π in polynomial time.

1. Prove that, if the graph is instead given as an edge-list representation[2], and we restrict to graphs without isolated vertices, then the problem Π can still be solved in polynomial time.

2. Does this mean that the way the input is represented does not matter? (See also Q10.)

[1] An *adjacency matrix* for a connected graph with v vertices is a $v \times v$ matrix whose rows and columns are indexed by the vertices, where the entry in row i and column j is 1 if vertices i and j are adjacent in the graph, and 0 otherwise. So the matrix consists entirely of 0s and 1s. How many bits do you need to represent a graph in this way?

[2] An *edge list* for a graph with v vertices and e edges consists of a list of e pairs, one for each edge. A pair (i, j) represents an edge between vertices i and j. If vertices i and j are not adjacent, then the pair (i, j) does not appear in the list. How many bits do you need to represent a graph in this way?

6. A *k-edge-colouring* of a graph G is a function that assigns to each edge a colour from the set $\{1, 2, \ldots, k\}$ such any two edges that have an endpoint in common receive different colours.

Consider the following problem.

EDGE-COLOURING
INPUT: Graph G, positive integer k.
QUESTION: Does G have a k-edge-colouring?

(a) Prove that EDGE-COLOURING $\in$ NP.

(b) Can you find a value of k such that, if we restrict to that particular k, the problem can be solved in polynomial time?

7.

Consider the following problem.

SHORTEST REGEXP
INPUT: Regular expression E, positive integer k.
QUESTION: Does there exist another regular expression F that is equivalent to E (i.e., they both represent the same language) and which has length $\leq k$?

Here is an *attempt* at an outline of a proof that SHORTEST REGEXP belongs to NP. Comment on what is correct and incorrect in this proof.

(1) Consider the following verifier:
   (i) Input: Regular expression E, positive integer k.
   (ii) Certificate: another regular expression, F.
   (iii) Convert E to a Finite Automaton A that recognises the language that E represents (using the methods from the proof of Kleene's Theorem).
   (iv) Convert F to a Finite Automaton B that recognises the language that F represents.
   (v) Apply the state minimisation algorithm to A. This produces a Finite Automaton $A^{\min}$ that recognises the same language as A and is the unique FA with fewest states that does this.
   (vi) Apply the state minimisation algorithm to B. This produces a Finite Automaton $B^{\min}$ that recognises the same language as B and is the unique FA with fewest states that does this.

(vii) **If** $A^{\min} = B^{\min}$ **and** $|F| \leq k$ **then** ACCEPT,

(viii) **else** REJECT.

(2) Each of the algorithms used in this verifier runs in polynomial time.

(3) To test whether $A^{\min} = B^{\min}$, just test that the state names correspond and that the transitions between state pairs also correspond. This takes polynomial time.

(4) Testing whether $|F| \leq k$ can be done in polynomial time.

(5) Therefore the entire verifier runs in polynomial time.

(6) The verifier accepts if and only if the two regular expressions E and F represent the same language (by Kleene's Theorem) *and* $|F| \leq k$. Therefore the verifier accepts if and only if $(E, k) \in$ SHORTEST REGEXP. So it is indeed a verifier for SHORTEST REGEXP.

(7) Since it is a polynomial time verifier for SHORTEST REGEXP, we conclude that SHORTEST REGEXP $\in$ NP.

8.

Consider the following problem.

NEARLY SAT
INPUT: Boolean expression φ in CNF.
QUESTION: Is there a truth assignment that satisfies all, or all except one, of the clauses of φ?

(a) Prove that NEARLY SAT belongs to NP.

(b) Give a polynomial-time reduction from SAT to NEARLY SAT and prove that it is such a polynomial-time reduction.

Supplementary exercises

9. Recall Exercise Sheet 9, Q8, about **doubling** and **halving**, and the following operation on languages.

Let L be any nonempty non-universal language that is closed under doubling and halving. Let L^{even} be the set of all strings of even length in L.

Prove that $L \in \mathrm{P}$ if and only if $L^{\text{even}} \in \mathrm{P}$.

10.

(a) Write a simple, naive algorithm which takes a positive integer x as input and outputs the smallest factor of x which is greater than 1 and less than x (if such exists).

For example, if $x = 6$, your algorithm should output 2. If $x = 7$, your algorithm should report that no such factor exists (since the only factors of 7 are 1 and 7).

For this question, you may assume (not quite realistically) that testing integer divisibility takes constant time.

(b) If the integer is given in *unary* (i.e., as a string of x 1s), what is the complexity of the algorithm, in big-O notation, in terms of the input size n?

(c) Does this algorithm run in polynomial time?

(d) If the integer is given in *binary*, what is the complexity of the algorithm, in big-O notation, in terms of the input size n?

(e) Does the algorithm run in polynomial time now?

(f) What would happen if you used some other base $b > 2$?

(g) [Advanced] Now suppose that each positive integer x is encoded by a sequence giving the successive exponents in its unique prime factorisation. So, if $x = 126$, then its prime factorisation is

$$126 = 2 \cdot 3 \cdot 3 \cdot 7 = 2^1 \cdot 3^2 \cdot 5^0 \cdot 7^1$$

so this x is represented by the sequence $(1, 2, 0, 1)$.

(i) How feasible would it be to use your algorithm with this way of encoding integers?

(ii) Write a faster algorithm, based on this representation, and determine its big-O complexity.

11.

A **short verifier** is a verifier whose certificate consists of ≤ 100 characters.

Let NP[SHORT] be the class of languages with polynomial-time *short* verifiers.

(a) Prove that P = NP[SHORT].

Challenge:

More generally, for any function $f : \mathbb{N} \cup \{0\} \to \mathbb{N} \cup \{0\}$, define NP$[f(n)]$ be the class of languages accepted by polynomial-time verifiers whose certificates consist of $\leq f(n)$ characters, where n is the input size. So, for example, NP[100] is just NP[SHORT]. In that case the function is just the constant function whose value is always 100. But think about what can happen when f is *unbounded*, i.e., there is no constant b such that $f(n) \leq b$ for all n.

(b) Can you find an *unbounded* function f such that P = NP$[f(n)]$?

(c) Can you find an *unbounded* function f such that NP = NP$[f(n)]$?