Complexity theory: Supervision 1

Daniel Chatfield

May 3, 2015

1. In the lectures, a proof was sketched showing a $\omega(nlogn)$ lower bound on the complexity of the sorting problem. It was also stated that a similar analysis could be used to establish the same bound for the Travelling Salesman Problem. Give a detailed sketch of such an argument. Can you think of a way to improve the lower bound?

   Not sure how to adapt the proof to apply to TSP

2. Consider the language *Unary-Prime* in the one letter alphabet $\{a\}$ defined by:

   $$Unary - Prime = \{a^n | n \text{ is prime } \}$$

   Show that this language is in P.

   To show that Unary-Prime is in P it is sufficient to demonstrate that an algorithm of polynomial running time exists that decides Unary-Prime.

   Such an algorithm is:

   ```
   // On input $\{ 0,1 \}^n$:

   for k = 2, ..., n-1:
       if k divides n, reject
   accept
   ```

   The division can be carried in time $\mathcal{O}(n^2)$ for each value of k, thus since k ranges from 2 to n-1 the total running time is $\mathcal{O}(n^3)$ which is polynomial in time.

3. We say that a propositional formula ϕ is in 2CNF if it is a conjunction of clauses, each of which contains exactly 2 literals. The point of this problem is to show that the satisfiability problem for formulas in 2CNF can be solved by a polynomial time algorithm.

   First note that any clause with 2 literals can be written as an implication in

exactly two ways. For instance $(p \vee \neg q)$ is equivalent to $(q \implies p)$ and $(\neg p \to \neg q)$, and $(p \vee q)$ is equivalent to $(\neg p \to q)$ and $(\neg q \to p)$.

For any formula ϕ, define the directed graph G_ϕ to be the graph whose set of vertices is the set of all literals that occur in ϕ, and in which there is an edge from literal x to literal y if, and only if, the implication $(x \implies y)$ is equivalent to one of the clauses in ϕ.

(a) If ϕ has n variables and m clauses, give an upper bound on the number of vertices and edges in G_ϕ.

> An upper bound on the number of vertices is $2n$ (one for each variable and its negation). The upper bound on the number of edges is $2m$ (two for each clause).

(b) Show that ϕ is unsatisfiable if, and only if, there is a literal x such that there is a path in G_ϕ from x to $\neg x$ and a path from $\neg x$ to x.

> Since if $x \to y$ and $y \to z$ then $x \to z$, a path from x to $\neg x$ is equivalent to $\neg x \vee \neg x$. A path from $\neg x$ to x is equivalent to $x \vee x$.
>
> This is trivially unsatisfiable.
>
> *Not sure about reverse implication*

(c) Give an algorithm for verifying that a graph G_ϕ satisfies the property stated in (b) above. What is the complexity of your algorithm?

> Reachability algorithm:
>
> 1. mark node a, leaving other nodes unmarked and initialise set S to $\{a\}$.
> 2. while S is not empty, choose node i in S: remove i from S and for all j such that there is an edge (i, j) and j is unmarked, mark j and add to S.
> 3. if b is marked, accept else reject.
>
> The complexity of the algorithm that determines whether there is a path between two nodes is $\mathcal{O}(V + E)$, since this might need to be computed for each variable, the total complexity is $\mathcal{O}(n \times (2n + 2m))$ = $\mathcal{O}(n^2)$.

(d) From (c) deduce that there is a polynomial time algorithm for testing whether or not a 2CNF propositional formula is satisfiable.

> The algorithm in (c) determines whether a 2CNF formula is unsatisfiable, and thus can be used to determine whether it is satisfiable as this is simply the opposite. The running time is polynomial.

(e) Why does this idea not work if we have 3 literals per clause?

> It is no longer possible to convert each clause to a simple implication.

4. A clause (i.e. a disjunction of literals) is called a *Horn clause*, if it contains at most one positive literal. Such a clause can be written as an implication: $(x \vee (\neg y) \vee (\neg w) \vee (\neg z))$ is equivalent to $((y \wedge w \wedge z) \rightarrow x))$. HORNSAT is the problem of deciding whether a given Boolean expression that is a conjunction of Horn clauses is satisfiable.

   (a) Show that there is a polynomial time algorithm for solving HORNSAT.

   (Hint: if a variable is the only literal in a clause, it must be set to true; if all the negative variables in a clause have been set to true, then the positive one must also be set to true. Continue this procedure until a contradiction is reached or a satisfying truth assignment is found).

   > Each clause corresponds to an implication with a conjunction of zero or more positive literals on the left and zero or one positive literals on the right.
   >
   > Set all variables to false.
   >
   > While there exists a clause, c in ϕ that is $false$ under the current truth assignment then if the clause has no variables on the right side of the implication then return "UNSAT", otherwise set the variable on the right side of the implication to *true*.

   (b) In the proof of the NP-completeness of SAT it was shown how to construct, for every nondeterministic machine M, integer k and string x a Boolean expression ϕ which is satisfiable if, and only if, M accepts x within nk steps. Show that, if M is deterministic, then ϕ can be chosen to be a conjunction of Horn clauses.

   > *Not sure*

5. In general *k-colourability* is the problem of deciding, given a graph $G = (V, E)$, whether there is a colouring $\mathcal{X} : V \rightarrow 1, \ldots, k$ of the vertices such that if $(u, v) \in E$, then $\mathcal{X}(u) \neq \mathcal{X}(v)$. That is, adjacent vertices do not have the same colour.

   (a) Show that there is a polynomial time algorithm for solving 2-colourability.

To determine whether a graph can be coloured with 2 colours it is sufficient to determine whether the graph contains any odd loops.

To do this, start at a vertex then travel along the edges, assigning either "odd" or "even" to each vertex alternately. An odd cycle can be detected if two vertices with the same index are adjacent.

The running time of this is $\mathcal{O}(V^2)$.

(b) Show that, for each k, k-colourability is reducible to $k+1$-colourability.

Construct graph G' from G by adding a new vertex adjacent to all the old vertices, then G is *k-colourable* iff G' is *$k+1$-colourable*.

What can you conclude from this about the complexity of 4-colourability?

It is NP-complete, since 3-colourability is.