University of Sussex
Informatics

Spring 2020

Limits of Computation
Answers for Exercises 8
Dr Bernhard Reus

NP and NP-complete
(Lectures 17–19)

1. Show that the Graph Colouring Problem is in **NP**. To do so:

   (a) Sketch a `WHILE`-verifier for the Graph Colouring Problem. Explain how you would represent the data needed as input and sketch roughly how the program would work.

   Answer:
   The verifier takes as input an encoding of the graph $G = (V, E)$ and of the number K of colours, and additionally a certificate. There are various ways to represent the graph. Let's assume here that it is represented as two lists: a list of vertices (nodes), all of which are encoded as numbers, and a list of edges, which are represented as pairs (v_1, v_2). Note that the data of $((V, E), K)$ has to be encoded uniquely as a binary sequence but we assume we can encode and decode pairs and lists of numbers. In particular, all numbers are represented in binary. The verifier then must check:

   i. whether the certificate C is a list of pairs of numbers that are assumed to be a name of a node and its colour encoded as a number – (vertex,colour).
   ii. if (i) is false return false else check whether the colours used in C (in the second position of the pairs) are allowed, i.e. one of $1, \ldots, K$.
   iii. if (ii) is false then return false else for each vertex $v \in V$ collect a list L of all the colours of nodes that appear in a edge in E that contains v. The colours can be found in C. If a node is not in C return false immediately. Otherwise check whether the colours in L are different from the colour of v.

iv. if (ii) is false return false else return true.

(b) Show that your verifier above runs in polynomial time in the size of the problem instance.

Answer:
Consider the runtime of each step:

i. This can be done in *constant* time in the size of the problem instance, i.e. in $((V, E), K)$.

ii. This can be done in time $\mathcal{O}(|K|)$ for the comparisons with K. Note that size of K is logarithmic in K.

iii. The collection of L can be done in time $\mathcal{O}(|E|)$. Checking whether the colours are different for each vertex can be checked in $\mathcal{O}(|V|^2)$

iv. This can be done in constant time.

So if we add all those time requirements up we get $\mathcal{O}(|V| + |K|+|E|+|V|^2)$ and thus we have a polynomial runtime in the size of the problem instance $((V, E), K)$ which is $|((V, E), K)| = |V| + |E| + |K|$.

Note that you can implement the checking da bit differently, also more efficiently. It's important that your program "sketch" is clear enough and that you get a good description of the runtime and argue it's polynomial.

2. Explain, by sketching the algorithm and its runtime, why checking whether a graph can be coloured using only <u>two</u> colours can be easily done in polynomial time.

Answer: One only has to check whether the graph $G = (V, E)$ is bipartite, i.e. whether V can be divided into two (disjoint) sets S and T, such that all edges in E are between a node in S and a node in T. Then clearly G can be coloured correctly by one colour for nodes in S and a second one for nodes in T.
This check con be done e.g. by a 'greedy' graph traversal. Start with a random vertex (colour it red). Look at all its adjacent vertices (connected by an edge in E). Colour blue. For each of those vertices (that you have not already coloured) repeat this action until you hit a vertex that you already coloured with a different colour. In this case you cannot use just 2-colours. When you have eventually visited all vertices without such a contradiction

you know that two colours suffice. This algorithm clearly is at worst $\mathcal{O}(|V|^2)$ and thus polynomial in the size of the input, i.e. the graph.

3. Consider the following decision problem PARTITION:
Given a finite set S of n elements that each have weight s_i for $1 \leq i \leq n$, can S be partitioned into two equally weighted subsets, i.e. is there s subset $T \subseteq S$ such that $\sum_{i \in T} s_i = \sum_{i \in S \setminus T} s_i$?
Show that PARTITION is polynomially reducible to 0-1 KNAPSACK, i.e. PARTITION $\leq_P$ 0-1 KNAPSACK. To do this:

   (a) Define the reduction function f such that $f(x) \in$ 0-1 KNAPSACK iff $x \in$ PARTITION for all problem instances x of PARTITION. *This is the part where you need to think!*

   Answer: The instance of PARTITION is represented as S where S is a list of length n containing weights s_i that are natural numbers. We need to map S to an instance of 0-1 KNAPSACK which is represented as (w, p, W, K) where w is a list of weights w_i for m items where w_i are all natural numbers, p is a list of profits p_i for the m items where p_i are all natural numbers, W is the total size of the knapsack, and K is the threshold for the total profit of the chosen elements, both natural numbers. We need to define $f(S) = (w, p, W, K)$ such that

   $$S \in \text{PARTITION iff } f(S) \in \text{0-1 KNAPSACK} \qquad (*)$$

   We observe that for a solution T of the PARTITION problem it holds $\sum_{i \in T} s_i = \sum_{i \notin T} s_i$ which is equivalent to $\sum_{i \in T} s_i = \sum_{i \notin T} s_i = \frac{1}{2} \sum_{1 \leq i \leq n} s_i$ as $\sum_{1 \leq i \leq n} s_i = \sum_{i \in T} s_i + \sum_{i \notin T} s_i = \sum_{i \in T} s_i + \sum_{i \in T} s_i = 2 \sum_{i \in T} s_i$. This observation is all we need to find our reduction function f:

   $$f(S) = (S, S, \frac{1}{2} \sum_{1 \leq i \leq n} s_i, \frac{1}{2} \sum_{1 \leq i \leq n} s_i)$$

   Now we check that $(*)$ is met:
   $S \in$ PARTITION
   iff there is a $T \subseteq \{1, \ldots, n\}$ s.t. $\sum_{i \in T} s_i = \sum_{i \notin T} s_i$
   iff there is a $T \subseteq \{1, \ldots, n\}$ s.t. $\sum_{i \in T} s_i = \frac{1}{2} \sum_{1 \leq i \leq n} s_i$ (see

above)
iff there is a $T \subseteq \{1, \ldots, n\}$ s.t. $\sum_{i \in T} s_i \leq \frac{1}{2} \sum_{1 \leq i \leq n} s_i$ and $\sum_{i \in T} s_i \geq \frac{1}{2} \sum_{1 \leq i \leq n} s_i$
iff $(S, S, \frac{1}{2} \sum_{1 \leq i \leq n} s_i, \frac{1}{2} \sum_{1 \leq i \leq n} s_i) \in$ 0-1 KNAPSACK .

(b) Check that f is total and computable.
Answer:
f is total by definition. It is also clearly computable as finite sums are computable.

(c) Check that the program implementing f runs in polynomial time.
Answer:
Computing

$$f(S) = (S, S, \frac{1}{2} \sum_{1 \leq i \leq n} s_i, \frac{1}{2} \sum_{1 \leq i \leq n} s_i)$$

can clearly be done in time $\mathcal{O}(|S|)$ as the sums are computed in time $\mathcal{O}(|S|)$, given that $n = |S|$.