

# CS172: Week 12.5 Homework Solutions

Allison Coates

- 1a. Show that if  $P = NP$ , then given a graph  $G$  that is in HAMPATH, we can determine a Hamiltonian path in this  $G$  in polynomial time. This is very similar to the self reducibility problem for solving SAT.

For simplicity, let's pick the standard definition of HAMPATH: Given a graph  $G = (V, E)$  and endpoints  $a$  and  $b$ , where  $G$  is in HAMPATH, select an edge  $e \in E$  such that  $e$  is an edge of the form  $(a, v), v \in V$ . Build a modified graph  $G' = (V', E')$  where you remove all other edges of the form  $(a, x)$  from the graph. Since  $P = NP$  then the decision problem for HAMPATH on  $G'$  can be decided in poly-time. If the answer is yes, continue in this fashion, examine the node  $v$  in  $G'$  and pick one edge of the form  $(v, v')$  as before, et cetera. When you reach an edge of the form  $(v, b)$  and the graph is in HAMPATH, then stop and output the set of selected edges; if the answer is no, then select a different edge of the form  $(a, v)$  until one works. (It must work, since the original problem is in HAMPATH.)

At worst, there are  $|E|$  edges looked at for every node  $\in V$ . Hence, the time to process this is  $|V||E|$  which is polynomial in the size of the description of  $G$ .

- 7.8 Show that primality testing is solvable in polynomial time if we use a unary encoding rather than a binary encoding for numbers. In other words, show that the language

$$\text{UNARY-PRIMES} = \{1^n \mid n \text{ is prime}\}$$

is in P.

We can test if a number  $p$  is prime by dividing it by all numbers from 1 to  $\sqrt{n}$ , and checking if any is a factor of  $p$ . Division can be performed in polynomial time; the number of numbers that must be tried is clearly polynomial in the length of the input. Note how different this is from the case where the numbers are represented in binary:  $\sqrt{n}$  is not polynomial in the length of the input is the input is of size  $\log n$ .

- 7.15 Let UNARY-SSUM be the subset sum problem in which all numbers are represented in unary. Why does the NP-completeness proof for SUBSET-SUM fail to show UNARY-SSUM is NP-complete? Show that UNARY-SSUM is in P.

The 3SAT to UNARY-SSUM reduction gives (cf. Thm 7.37): For  $\phi$  a  $k$ -clause,  $l$ -variable 3CNF formula, the reduction of Thm 7.37 gives a set of  $(l + k)$  numbers, and a target  $t$  that has length  $(l + k)$  decimals. Thus  $t > 10^{(l+k)}$ , and hence the length of  $t$  in unary is exponential in  $(l, k)$ . This makes the 3SAT  $\rightarrow$  UNARY-SSUM reduction non-polytime.

Show that UNARY-SSUM is in P. Consider an instances  $(S, t)$  of UNARY-SSUM. If  $L$  is the length of the input  $(S, t)$  in unary, then we know that  $t \leq L$ . Let  $S = (s_1, \dots, s_n)$ , then we can bound the 'reach' of  $S$  by  $\sum_i s_i = R$ , where again we know (because of unarity)  $R, n \leq L$ . We now construct an index-table of length  $R + 1$  that indicates which targets can be made with  $S$ , and we do this in  $O(L^2)$  time. The table is called  $X$  and has  $R + 1$  Boolean entries. We start with  $X[0] = \text{True}$ ,  $X[1] = X[2] = \dots = X[R] = \text{False}$  (reflecting that with no  $S$  entries we can only make the target  $t = 0$ ). We can then write the following code, which solves the problem (the table  $Y$  functions as a copy of  $X$ ):

- For  $i = 1$  to  $n$ 
  - $Y := X$ ;
  - For  $j = 0$  to  $R$ 
    - \* IF  $X[j] = \text{True}$  THEN  $Y[j + s_i] := \text{True}$ ;
  - Next  $j$ ;
  - $X := Y$ ;
- Next  $i$ ;
- IF  $X[t] = \text{True}$  THEN print(“Target can be made.”)
  - ELSE print(“Target cannot be made.”)

7.16 Let  $G$  represent an undirected graph and let

SPATH =  $\{\langle G, a, b, k \rangle \mid G \text{ contains a simple path of length at most } k \text{ from } a \text{ to } b\}$ ,

and

LPATH =  $\{\langle G, a, b, k \rangle \mid G \text{ contains a simple path of length at least } k \text{ from } a \text{ to } b\}$ .

(a) Show that SPATH  $\in$  P.

On instance  $\langle G, a, b, k \rangle$ , perform breadth-first search beginning at  $a$  for  $k$  steps:

- begin by marking vertex  $a$ .
- Set  $i = 1$ .
- Then, mark every vertex reachable in  $i$  steps from  $a$ .
- If  $i = k$ , check that  $b$  is marked; if so, output yes and halt, otherwise, output no and halt.
- If  $i \neq k$ , increment  $i$  and continue with breadth-first search.

This algorithm runs in polynomial time since finding  $a$  takes  $O(n)$  time; marking every vertex reachable from a vertex  $v$  requires  $O(n^2)$  time; the loop runs in at most  $n$  times, so that the total time is  $O(n^3)$ .

(b) Show that LPATH is NP-Complete. You may assume the NP-completeness of UHAMPATH, the Hamiltonian path problem for undirected graphs.

We begin by showing that LPATH  $\in$  NP. Let the path  $p$  be the certificate for the LPATH. To verify  $p$  is an instance of LPATH in polynomial time, simply walk the graph, verifying that each edge in  $p$  is valid—that is, that the path begins at  $a$ , each edge walked is in the graph, and the final edge ends at  $b$ . Maintain a counter to determine the number of edges that are traversed. If the path is valid, and the value of the counter is  $\geq k$  then accept, else reject. Traversing the path can take at most  $O(n^2)$  time, and the counter is surely less than this. Hence, this certificate can be verified in polynomial time.

To show that LPATH is NP-complete, we reduce from HAMPATH by restriction. We wish to show a mapping  $f$  such that a problem  $x \in \text{HAMPATH} \Leftrightarrow f(x) \in \text{LPATH}$ . Define HAMPATH in the standard way:  $\text{HAMPATH} = \{\langle G, a, b \rangle \mid G \text{ contains a path from } a \text{ to } b \text{ that passes through every node in } G \text{ exactly once}\}$ . Given a graph  $G$  with  $n$  nodes, the mapping  $f$  is as follows:  $f(\langle G, a, b \rangle) = \langle G, a, b, n \rangle$ . That is, we simply take  $k$  to be the number of nodes in  $G$  such that LPATH is simply an equivalent definition for HAMPATH. Obviously this mapping is polynomial in the size of the input. Note that this is a solution by restriction: we simply showed that a restricted set of problem instances of LPATH is already hard enough so that overall, LPATH must be hard.