

CMSC 28100-1 / MATH 28100-1
Introduction to Complexity Theory
Fall 2017 – Homework 1

September 28, 2017

Important note about writing algorithms.

You may write algorithms in (pseudo-)code or English, but they should be completely specified. For example, saying

“Let (i, j) be the best edge satisfying this complicated criteria.”

is not acceptable. Saying

“Let j_0 be the least j such that $A[i, j] = 1$.”

is acceptable. Whenever we say “give an algorithm” we require the following three things.

1. Describe the algorithm completely, as mentioned above.
2. Prove that the algorithm correctly solves the problem it is supposed to. For example, for Exercise 1 below, you must prove that your algorithm returns a sequence of vertices that is a valid path from s to t , if and only if there is such a path.
3. Analyze the runtime of your algorithm. You may report the runtime using big-Oh $O(\cdot)$ notation.

The problems in this homework are about graphs. Report the runtime as a function of the number of vertices V and the number of edges E . For example $O(E \log E + V^2)$.

Definition. A directed graph G consists of a set of vertices $V(G)$ and a set of edges $E(G) \subseteq V \times V$. If there is an edge $(u, v) \in E(G)$, we say that u is the source of the edge and v is the target of the edge, and that u is a predecessor of v and v is a successor of u . A path in G is a sequence of distinct vertices v_0, \dots, v_n such that (v_i, v_{i+1}) is an edge in $E(G)$ for all $i = 0, \dots, n-1$. Such a path is called a path from v_0 to v_n . The length of such a path is the number of edges in it, that is, n (note that we started numbering vertices from 0). Unless explicitly stated otherwise, all graphs here are directed. Graphs are given as input to algorithms by their adjacency matrices: if the graph has n vertices then its vertices are labeled by the integers $1, \dots, n$ and the adjacency matrix of a graph G is the matrix A such that $A[i, j] = 1$ if $(i, j) \in E(G)$ and $A[i, j] = 0$ otherwise.

Homework Questions

Exercise 1. Consider the following problem.

Problem: s - t PATH.

Input: A directed graph G , and two vertices $s, t \in V(G)$.

Output: A path from s to t or “NOT CONNECTED” if there is no path from s to t in G .

Show that this problem can be solved in polynomial time by giving an algorithm. Remember, “giving an algorithm” requires the three steps mentioned above.

Exercise 2. A shortest path from s to t is a path from s to t that is the shortest among all paths from s to t (i.e., has least length). Consider the following variant of the problem above.

Problem: SHORTEST s - t PATH.

Input: A directed graph G , and two vertices $s, t \in V(G)$.

Output: A shortest path from s to t or “NOT CONNECTED” if there is no path from s to t in G .

Show that this problem can be solved in polynomial time by giving an algorithm. If there is more than one shortest path, your algorithm may return any shortest path. If your algorithm from the previous exercise already solves this problem, you do not have to repeat it, but you do have to prove its correctness, that is, you still need to show that your algorithm always returns a shortest path from s to t whenever there is any path from s to t .

Exercise 3. A graph is strongly connected if for any two vertices $u, v \in V(G)$ there is a path from u to v . Show that the problem of deciding whether a graph is strongly connected can be solved in polynomial time, as in the previous questions.

If you wish, you may use algorithms from the previous questions as sub-routines in this algorithm, you do not have to rewrite them or re-prove their correctness, but you still have to prove the correctness of your algorithm for this problem assuming the correctness of the algorithms for the previous exercises.

Exercise 4. If there is an edge $(u, v) \in E(G)$ in a directed graph G , then u and v are said to be neighbors or adjacent to one another. An undirected path in a (directed) graph G is a sequence of distinct vertices v_0, \dots, v_n such that for each $i = 0, \dots, n - 1$, the vertices v_i and v_{i+1} are adjacent, in other words, at least one of (v_i, v_{i+1}) and (v_{i+1}, v_i) is an edge for each i .

A graph is (weakly) connected if for any pair of vertices $u, v \in V(G)$, there is an undirected path from u to v . Show that the problem of determining whether a graph is (weakly) connected can be solved in polynomial time by giving an algorithm. You may use any algorithm from previous questions as subroutines, as before.

Exercise 5. Consider the following problem and algorithm to solve it.

Problem: SUBSET-SUM.

Input: Numbers x_1, \dots, x_k and a target number T .

Output: “YES” if there is a subset $S \subseteq \{1, \dots, k\}$ such that $\sum_{i \in S} x_i = T$, and “NO” otherwise.

Algorithm 5.1: SUBSET-SUM

```
1 Initialize  $A[i, t] \leftarrow 0$  for all  $i = 1, \dots, k$  and all  $t = 1, \dots, T$ .
2 for  $t \leftarrow 1$  to  $T$  do
3   if  $x_1 = t$  then  $A[1, t] \leftarrow 1$ 
4   else  $A[1, t] \leftarrow 0$ 
5 for  $i \leftarrow 2$  to  $k$  do
6   for  $t \leftarrow 1$  to  $T$  do
7     if  $x_i = t$  then  $A[i, t] \leftarrow 1$ 
8     if  $A[i - 1, t] = 1$  then  $A[i, t] \leftarrow 1$ 
9     if  $A[i - 1, t - x_i] = 1$  then  $A[i, t] \leftarrow 1$ 
10 if  $A[k, T] = 1$  then return "YES"
11 else return "NO"
```

- (a) Prove that the algorithm above correctly solves SUBSET-SUM.
- (b) Analyze the running time of the algorithm above in terms of k and T .
- (c) Why does the algorithm above not show that SUBSET-SUM can be solved in polynomial time? Hint: what is the size, in bits, of the input as a function of k and T ? Then what is the running time in terms of the input size?