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

October 1, 2017

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 and algorithm. Remember, "giving and algorithm" requires the three steps mentioned above.

Solution. Consider the following algorithm.

Algorithm 1.1: Shortest s-t path

```
1  Initialize $V[u] \leftarrow 0$ for every $u \in V(G)$.
2  Set $V[u] \leftarrow 1$.
3  Let $Q$ be a queue data structure (initialized to be empty).
4  Push $(s)$ to the end of $Q$.
5  while $Q$ is not empty do
6      Pop the first element $(v_0, \ldots, v_k)$ from $Q$.
7      if $v_k = t$ then return $(v_0, \ldots, v_k)$.
8      for $u \in V(G)$ such that $(v_k, u) \in E(G)$ do
9          if $V[u] = 0$ then
10             Set $V[u] \leftarrow 1$.
11             Push $(v_0, \ldots, v_k, u)$ to the end of $Q$.
12 return "NOT CONNECTED".
```

Let us first analyze the time complexity Algorithm 1.1.

First note that the vector V gets initialized to 0 (which takes time $O(n)$) and each of its entries can only be set to 1 later. Furthermore, note that whenever the "if" of line 9 tests true, one entry of V gets set to 1. This immediately implies that the "if" of line 9 tests true at most $n = |V(G)|$ times.

Note also that the only places where some element is pushed to the end of the queue Q are either on line 4 or on line 11 inside the "if" of line 9. Since the first instruction of the "while" of line 5 pops an element of Q, this "while" can execute at most $n + 1$ times.

Each time a new sequence is pushed to the end of Q on line 11, it consists of the previously popped sequence plus one additional vertex appended. This implies that the maximum length of any sequence in Q is n. Hence, each push or pop operation takes time at most $O(n \log n)$ (the $\log n$ factor accounts for the bit-length of the names of the vertices, which are assumed to be $V(G) = \{1, \ldots, n\}$).

Finally, the "for" of line 8 clearly executes at most n times each time (depending on the representation of the graph, we can prove that it executes a total of $m = |E(G)|$ times throughout the whole algorithm).

Putting everything together the time complexity of Algorithm 1.1 is

$$O(n) + (n+1) \cdot O(n \log n + n) + n \cdot O(n \log n) = O(n^2 \log n),$$

where the first summand accounts for initialization, the second summand for the cost of the "while" loop except for the inside of the "if" of line 9, which is accounted by the last summand. This is clearly polynomial as $O(n^2 \log n) \leqslant O(n^3)$.

We remark that with small adjustments to this algorithm (namely, being more efficient with what to store in the queue), this time complexity can be improved.

Let us now prove the correctness of Algorithm 1.1.

First, we claim that every sequence pushed on Q is a path in G starting at s.

Suppose not and let $(v_0, \ldots, v_k)$ be the first sequence pushed on Q that is not a path in G. Since (s) is a path in G, the violating sequence must have been pushed on line 11, hence the sequence $(v_0, \ldots, v_{k-1})$ must have been previously in Q and $(v_{k-1}, v_k) \in E(G)$. By our choice of $(v_0, \ldots, v_k)$, we know that $(v_0, \ldots, v_{k-1})$ is a path in G starting at s, which implies that $(v_0, \ldots, v_k)$ is a path in G starting at s, which is a contradiction.

Therefore all sequences pushed on Q are paths of G. Since the only place of Algorithm 1.1 that returns a path $(v_0, \ldots, v_k)$ is on line 4, and this requires that $v_k = t$, it follows that if Algorithm 1.1 returns a path, then there exists an s-t path in G and the returned path is one of them.

It remains to prove that if G contains an s-t path, then Algorithm 1.1 returns a path.

Let then $(w_0, \ldots, w_k)$ be an s-t path in G and let us prove by induction in $i \in \{0, \ldots, k\}$ that there is some $(v_0, \ldots, v_r)$ pushed on Q with $v_r = w_i$.

For $i = 0$, since $w_0 = s$, this is trivial as (s) is pushed on Q.

Suppose then that $i \in \{1, \ldots, k\}$ and that the assertion holds for $i - 1$.

Since every time some $(v_0, \ldots, v_r)$ is pushed on Q, the algorithm sets $V[v_r]$ to 1, it is enough to show that $V[w_i]$ is eventually set to 1.

Let $(v_0, \ldots, v_r)$ be the first sequence with $v_r = w_{i-1}$ that is pushed on Q and consider the iteration of the "while" of line 5 in which $(v_0, \ldots, v_r)$ is popped from Q. Since $(w_{k-1}, w_k) \in E(G)$ (as $(w_0, \ldots, w_k)$ is a path of G), we know that the "for" of line 8 takes $u = w_k$ during such iteration. Then either we already had $V[w_k] = 1$ or the "if" of line 9 tests true and sets $V[w_k] = 1$. In any case we are done.

Therefore, by induction it follows that for every $i \in \{0, \ldots, k\}$, there is some $(v_0, \ldots, v_r)$ pushed on Q with $v_r = w_i$.

Considering the case of $i = k$, when this path $(v_0, \ldots, v_r)$ is popped from Q, the "if" of line 7 tests true and the algorithm returns a path as desired. $\triangleleft$

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.

Solution. Let us prove that Algorithm 1.1 of Exercise 1 in fact returns a shortest s-t path.

First, let us prove that paths are pushed onto Q in a non-decreasing order of length and at any moment, there exists $k \geqslant 0$ such that all paths in the queue Q have length either k or $k+1$.

Suppose not, and let $(v_0, \ldots, v_r)$ be the first path pushed on Q that makes it violate one of these properties. Clearly $(v_0, \ldots, v_r)$ must have been pushed on line 11.

By our choice of $(v_0, \ldots, v_k)$, the queue Q satisfied the properties before pushing this element and since $(v_0, \ldots, v_{k-1})$ must have been popped from Q to push $(v_0, \ldots, v_k)$ later and between these two operations only paths of length k can have been pushed, it follows all paths of Q must have had length either $k-1$ or k, contradicting the choice of $(v_0, \ldots, v_k)$.

Let us now prove that every $(v_0, \ldots, v_k)$ that is pushed on Q is a shortest s-v_k path (hence the returned path must be a shortest s-t path since we have already proved it must be an s-t path).

The proof is by induction in the distance $d_s(v_k)$ of v_k from s (i.e., the length of one shortest s-v_k path). For $d_s(v_k) = 0$ this is trivial since the only path ending on s pushed on Q is (s).

Suppose then that $d_s(v_k) > 0$ and that the result holds for smaller values and let us prove that if $(v_0, \ldots, v_k)$ is pushed on Q, then it is a shortest s-v_k path (i.e., we have $d_s(v_k) = k$).

Let $(w_1, \ldots, w_r)$ be a shortest s-v_k path (hence $r \leqslant k$). Clearly $(w_1, \ldots, w_{r-1})$ is a shortest s-w_{r-1} path (otherwise taking one such shortest s-w_{r-1} path and appending w_r would yield an s-w_r path shorter than $(w_1, \ldots, w_r)$), hence $d_s(w_{r-1}) = d_s(v_k) - 1$.

By what we proved in Exercise 1, we know that there exists some s-w_{r-1} path $(x_1, \ldots, x_\ell)$ that is pushed on Q. Since $d_s(x_\ell) = d_s(w_{r-1}) < d_s(v_k)$, we know that $(x_1, \ldots, x_\ell)$ is a shortest s-w_{r-1} path, hence $\ell = r-1$. But then, since $(w_{r-1}, w_r) \in E(G)$, the path $(v_0, \ldots, v_k)$ must be pushed on Q either in the iteration when $(x_1, \ldots, x_\ell)$ is popped or before (since $(v_0, \ldots, v_k)$ is the only s-v_k-path pushed on Q), which implies $k \leqslant \ell + 1 = r$.

Therefore $k = r = d_s(v_k)$ as desired. ◁

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.

Solution. Consider the following algorithm.

Algorithm 3.1: Strongly connected

```
1 for $u, v \in V(G)$ do
2     Run Algorithm 1.1 with input $(G, u, v)$.
3     if Algorithm 1.1 returns "NOT CONNECTED" then
4         return "NOT STRONGLY CONNECTED".
5 return "STRONGLY CONNECTED".
```

Clearly the time complexity of Algorithm 3.1 is n^2 times the time complexity of Algorithm 1.1, hence polynomial (where $n = |V(G)|$).

Furthermore, assuming the correctness of Algorithm 1.1, it is easy to see that for Algorithm 3.1 to return "STRONGLY CONNECTED", there must exist an u-v path for all $u, v \in V(G)$, i.e., the graph G must be strongly connected.

On the other hand, for Algorithm 3.1 to return "NOT STRONGLY CONNECTED", there must exist $u, v \in V(G)$ such that there is no u-v path, hence the graph G is not strongly connected. ◁

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.

Solution. Consider the following algorithm.

Algorithm 4.1: Weakly connected

```
1 Let $A$ be the adjacency matrix of $G$ and $V(G) = \{1, \dots, n\}$.
2 for $i \leftarrow 1$ to $n$ do
3     for $j \leftarrow 1$ to $n$ do
4         if $A[i, j] = 1$ then Set $A[j, i] \leftarrow 1$.
5 Run Algorithm 3.1 on the graph given by the adjacency matrix $A$ and return its result.
```

Clearly the time complexity of Algorithm 4.1 is $O(n^2)$ plus the time complexity of Algorithm 3.1, hence polynomial.

Furthermore, if H is the graph passed as input to Algorithm 3.1, then (u, v) is an edge of H if and only if at least one of (u, v) or (v, u) is an edge of G. This implies that a sequence $(v_0, \dots, v_k)$ is a (directed) path in H if and only if $(v_0, \dots, v_k)$ is an undirected path in G. In turn, this implies that H is strongly connected if and only if G is weakly connected.

Therefore, Algorithm 4.1 correctly solves the weak connectivity problem. ◁

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 $i = 1, \ldots, k$ and all $t = 1, \ldots, T$.
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?

Solution. We start by solving item (a).

First note that the table $A[i,t]$ (after initialized) is filled in non-decreasing order of i (and each entry can only change from 0 to 1).

Let us prove by induction in i that $A[i,t]$ changes to 1 if and only if there exists $S \subseteq \{1,\ldots,i\}$ such that $\sum_{j\in S} x_j = t$ (from this correctness immediately follows since Algorithm 5.1 returns "YES" if and only if $A[k,T] = 1$).

For $i = 1$ this is trivially true as the loop in line 2 sets $A[1,t]$ to 1 if and only if $x_1 = t$.

Suppose then that $i > 1$ and that the result holds for $i-1$. Then we know that there exists $S \subseteq \{1,\ldots,i\}$ such that $\sum_{j\in S} x_j = t$ if and only if at least one of the following occurs.

- We have $x_i = t$.
- There exists $S \subseteq \{1,\ldots,i-1\}$ such that $\sum_{j\in S} x_j = t$.
- There exists $S \subseteq \{1,\ldots,i-1\}$ such that $\sum_{j\in S} x_j = t - x_i$.

Each of the items above is captured by each of the conditions in the loop 6, where the entries $A[i-1,u]$ are assumed to be correct by inductive hypothesis and the fact that the table $A[i,t]$ is filled in non-decreasing order of i.

Therefore $A[i,t]$ changes to 1 if and only if there exists $S \subseteq \{1,\ldots,i\}$ such that $\sum_{j\in S} x_j = t$ as desired.

For item (b), clearly the loop in line 2 executes T times, the loop in line 5 executes k times and the (inner) loop in line 6 executes T times, hence the time complexity of Algorithm 5.1 is

$$T \cdot O(1) + k \cdot T \cdot O(1) = O(k \cdot T).$$

Finally, for item (c), note that the input is a list of k numbers and the number T, so the bit-length of the input is $O(k \cdot \log M \cdot \log T)$, where $M = \max\{|x_i| : i \in \{1,\ldots,k\}\}$, hence Algorithm 5.1 does not run in polynomial time in the size of the input, but in exponential time (as $T = \Omega(2^{\log T})$). ◁