Carlos Mc Gregor c.mcgregormuro@mail.utoronto.ca

CSC373S: Algorithm Design, Analysis & Complexity

Lecture 24

Wednesday March 8, 2017

based on notes by Denis Pankratov

Some midterm pointers: You need to know...

- Design algorithm and runtime
- Solve simplex by hand (small linear program)
- Shortest path
- Network flow
- Network flow by hand

Complexity Theory

Example Story:
Imagine that you work at a company that takes *specifications* from clients and makes microschips based on them. Your boss tells you to automate the process (*i.e.* design an algorithm to make the best cheap microchips). At the beginning, you are excited! But some time passes and you get nowhere.

Best case scenario: you tell your boss there is no algorithm to solve the problem. Unfortunately, there is no way to prove that it cannot be done.

Next best thing: you say that there is a formal theoretical framework that explains why there is no optimal algorithm. That is what complexity theory does. It finds patterns that, if you are able to solve similar problems, then you can solve this problem. The same is true for the opposite: problems that are historically impossible to solve will still be impossible to solve (unless extensive resources or a genius finds a way).

Goal: a given natural problem does not have an efficient algorithm.

Ideally: show such problem requires exponential time.

Relaxed Goal: abiity to relate hardness of one problem with hardness of another problem; identify *hardest* problems, provide theoretical justification for no *efficient* algorithms.

NP-Completeness Theory

Problem: set of instances & a task.

Example: max-flow problem
Instances: $G = (V, E), c, s, t$ - flow networks
task: find max value of a flow

Optimization problem: task - max or min some objective function
Decision problem: given an instance, answer a *yes/no* question

Example:
Input: $G = (V, E), c, s, t$ - flow networks
$\upsilon \in \mathbb{R}$
Output: yes, if there is a flow of value $\geq \upsilon$

Optimal problems often correspond to decision problems, and runtimes of optimal algorithms for these problems are *polynomially related*

Let A be an algorithm that solves max-flow optimal version
To solve *corresponding decision version*

1. run A on the input
2. compare computed max flow value with υ.

Let A' be an algorithm for max-flow decision version
To solve *max-flow optimal version*

1. $A'' =$ run A' seeral times to binary search for maximum value υ such that there is a flow of value υ
2. runtime(A'') = runtime$(A)\mathcal{O}(log\upsilon*), \upsilon* =$ value of max-flow

Formal Languages

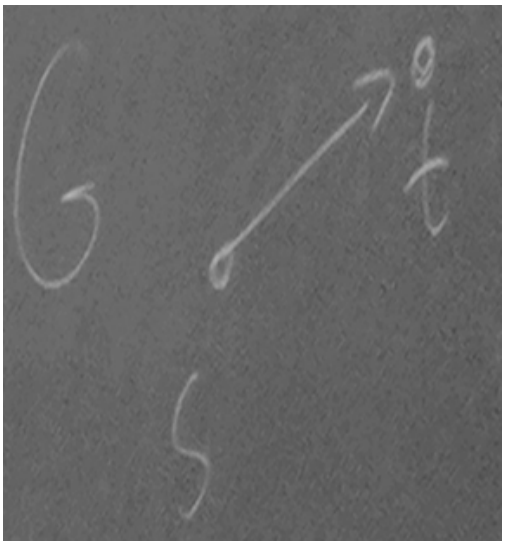
Σ - an alphabet (finite set)
$\Sigma^* =$ set of all finite strings over alphabet Σ

Definition: L is a langauge over Σ if $L \subseteq \Sigma^*$

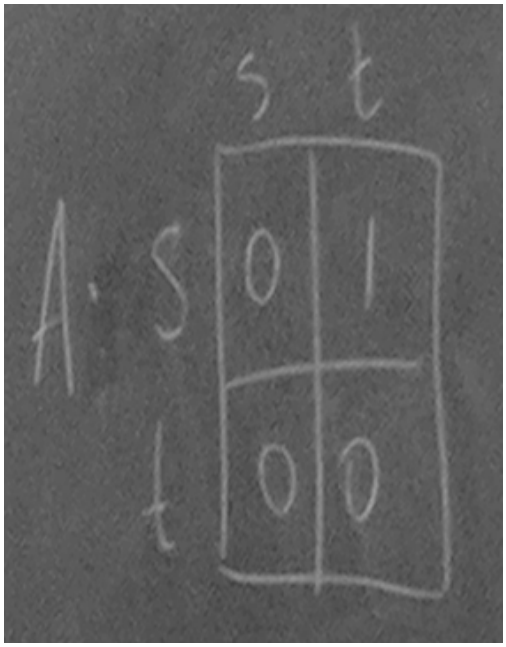
Decision Problems

$\leftrightarrow$ via encodings e to languages

Example: given G, s, t is s reachable from t
$e : (G, s, t) \mapsto < G, s, t > \leftarrow$ notation for encoding
G is encoded using adjacency matrix $s, t-$ encoded in binary
$\Sigma = \{'<', '>', '0', '1', ','\}$



(a) *General Visualization:* from s to t



(b) *Adjacency matrix* s, t*:*
encode s with 0, encode t with 1

Figure 1: Example

$$e(G, s, t) = \text{" } < 0100, 0, 1 > \text{"}$$

Most reasonable encodings have *polynomially related* sizes.

Important exception: $k \in \mathbb{N}$, unary encoding of k as a string of k 1s - length k

Binary encoding of k has length $\lfloor \log_2 k \rfloor + 1$

We will assume that $\Sigma = \{0, 1\}$, without loss of generality

Definition: $L \subseteq \{0,1\}^*$ is *polytime* solvable if there is a *polytime algorithm* A so that $(\forall x \in \{0,1\}^*)(x \in L \Leftrightarrow A(x) = 1$

Runtime of A is $\mathcal{O}(|x|^k)$ for some fixed $k > 0$
(*Note:* In practice, you do not see a value of k that is very high. This is because k represents nested loops, each of which represents an *idea* on how to solve the problem which can later be *simplified*.)

Definition: $P = \{L \subseteq \{0,1\}^* | L \text{ is polytime solvable } \}$
Translation: $P =$ class of *decision problem solvable* in polytime

Definition: $L \subseteq \{0,1\}^*$ is polytime verifiable if $\exists c > 0$ and polytime algorithm A of two inputs (x, y) so that $(\forall x \in \{0,1\})(x \in L \Leftrightarrow \exists y \in \{0,1\}^*, |y| \leq |x|^c, A(x, y) = 1)$

Definition: NP $= \{L \subseteq \{0,1\}^* | L$ is polytime verifiable $\}$
$\Uparrow$
nondeterministic polytime

Example: verifier for max-flow decision version instance is G, c, s, t flow network, υ
f - flow

1. check f satsifies flow constraints
2. check $|f| \geq \upsilon$