CompSci 162
Spring 2023 Lecture 20: Introduction to Computational Complexity

TM M_1 for $L = \{a^k b^k : k \geq 0\}$

n = size of input

1. Scan across and reject if any a right of a b } $O(n)$
2. Repeat while any a, b still on tape: } $O(n^2)$
3. Scan across, cross off one a, one b } $O(n)$
4. If only one letter type remains, reject
   Otherwise, accept if neither a nor b remain } $O(n)$

$O(n^2)$

TM M_2 for $L = \{a^k b^k : k \geq 0\}$

1. Scan across and reject if any a right of a b } $O(n)$
2. Repeat as long as some of each remain: } $O(n \lg n)$
3. If total $a + b$ is odd, reject. } $O(n)$
4. Cross off every other a, every other b } $O(n)$
5. If none of each remain, accept. } $O(n)$

$O(n \lg n)$

TM M_3 for $L = \{a^k b^k : k \geq 0\}$

1. Scan across and reject if any a right of a b } $O(n)$
2. Scan across the as on tape 1 until first b. While doing so, copy the as onto tape 2. } $O(n)$
3. Cross off a and b, 1 : 1 via two tapes If all a crossed off after and b remain, reject } $O(n)$
4. If all a crossed off, accept. Else reject. } $O(n)$

$O(n)$

Non-deterministic Running Time

- Running time is maximum steps, any branch
- Not intended as model of real world computation
- Relationship deterministic and non-deterministic:

  Every $t(n)$ time nondeterministic single-tape Turing Machine has an equivalent $2^{\mathcal{O}(t(n))}$ time deterministic single-tape Turing Machine

The Class $\mathcal{P}$

$\mathcal{P}$: languages that are decidable in polynomial time on a deterministic single-tape Turing machine.

$$\mathcal{P} = \bigcup_k \mathsf{TIME}(n^k)$$

- Polynomial differences are not important?

The Class $\mathcal{P}$

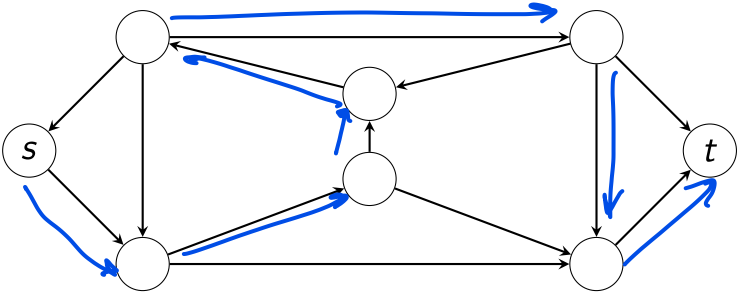
$\mathcal{P}$: languages that are decidable in polynomial time on a deterministic single-tape Turing machine.

$$\mathcal{P} = \bigcup_k \mathsf{TIME}(n^k)$$

- Polynomial differences are not important?

1. $\mathcal{P}$ is invariant for all equivalent models
2. $\mathcal{P} \approx$ realistically solvable on a computer.

HAMPATH



1. Best currently known algorithm's running time? exp
2. Anything related polynomial time solvable?

verifier

COMPOSITES

$$\text{COMPOSITES} = \{x : x = pq,\ p, q > 1,\ p, q \in \mathbb{Z}\}$$

- Needed for verifier?

  [alleged] divisor

- Polynomially verifiable?

  do the division

- When was PRIMES known to be in $\mathcal{P}$?

  $O(\log^{12} x) \longrightarrow O(\log^{6} x)$

What is a Verifier?

A **verifier** for language A is an algorithm V such that $A = \{w : V \text{ accepts } \langle w, c \rangle \text{ for some string } c\}$. We measure the running time of a verifier as a function of the length of w.

- c is the *certificate*

The Class $\mathcal{NP}$

- $\mathcal{NP}$ languages with polynomial time verifiers.
- $\mathcal{NP}$: "non-deterministic polynomial"

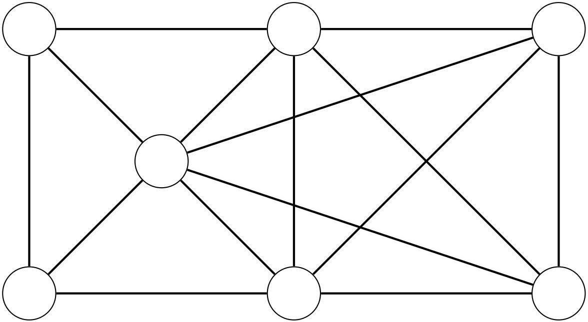
HAMPATH is in $\mathcal{NP}$

via non-deterministic Turing Machine.

Input: $\langle G, s, t, \rangle$, where G is a directed graph with nodes s and t

1. Write a list of m numbers, $p_1, \ldots, p_m$, where m is the number of vertices in G.
   Each number in the list is nondeterministically selected to be between 1 and m.
2. Check for repetitions in the list. If any are found, reject.
3. Check whether $s = p_1$ and $t = p_m$. If either fail, reject.
4. For each i between 1 and $m - 1$, check whether (p_i, p_{i+1}) is an edge of G. If any are not, reject.
5. If we reach this line, all tests have passed, so accept.

The CLIQUE Problem



Boolean Satisfiability

$$\phi = (x_2 \vee x_3 \vee x_4)(\overline{x_2} \vee x_3 \vee \overline{x_4})(\overline{x_1} \vee x_3 \vee x_5)$$
$$(\overline{x_1} \vee x_2 \vee x_5)(\overline{x_3} \vee x_4 \vee x_5)(\overline{x_2} \vee \overline{x_4} \vee \overline{x_5})$$
$$(x_1 \vee \overline{x_2} \vee x_5)(x_3 \vee \overline{x_4} \vee x_5)(x_1 \vee \overline{x_3} \vee \overline{x_5})$$
$$(\overline{x_2} \vee x_4 \vee \overline{x_5})(x_1 \vee x_2 \vee \overline{x_4})(\overline{x_1} \vee x_2 \vee x_3)$$
$$(\overline{x_1} \vee \overline{x_2} \vee x_5)(\overline{x_1} \vee x_2 \vee \overline{x_4})$$

x_1	True	False
x_2	True	False
x_3	True	False
x_4	True	False
x_5	True	False

Why care about Boolean Satisfiability?

- Some problems have an efficient solution
- Some problems are literally impossible
- Some do not have an efficient solution
  - i.e., list all subsets of a set
- Some do not have a *known* efficient solution

Why care about Boolean Satisfiability?

- Some problems have an efficient solution
- Some problems are literally impossible
- Some do not have an efficient solution
  - i.e., list all subsets of a set
- Some do not have a *known* efficient solution

- For every problem in $\mathcal{NP}$:
  - You *can* convert it to SAT
  - Size of ϕ is polynomial of original
    - Details in "handout 4.2" (reading)
  - Therefore, a poly solution to SAT gives you ...

- This was discovered by Cook and Levin in 1971