Računska zahtevnost

Zapiski s predavanj prof. Alexandra Simpsona

Domen Vogrin

Ljubljana, jesen 2023

Kazalo

1 Basics

1.1 Determinant of a matrix

Determinant of matrix

$$A = \begin{bmatrix} a_{11} & \cdots & a_{1n} \\ \vdots & \ddots & \vdots \\ a_{n1} & \cdots & a_{nn} \end{bmatrix}$$

is calculated via formula

$$det(A) = \sum_{\sigma \in perm(n)} sgn(\sigma) \cdot \prod_{i=1}^{n} a_{i,\sigma(i)}$$

Naive algorithm requires $n! \cdot (n-1)$ multiplications, which is worse than exponantional. However, there are better algorithms for the permanent. Use Gaussian elimination to factorise A

$$A = L \cdot U \cdot P$$

where L is lower triangle matrix, U is upper triangle matrix, and P is a permutation. This helps, because

$$\begin{aligned} det(A) =& det(L) \cdot det(U) \cdot det(P) \\ =& det(L) \cdot det(U) \end{aligned}$$

We can compute determinants of L and U by just multiplying diagonal elements.

Time complexity od LU decomposition is $\mathcal{O}(n^3)$, which is also time complexity of our algorithm.

Fastest known time for determinant is $\mathcal{O}(n^{2,376})$.

1.2 Primality testing

For given n, is n prime?

Naive algorithm: Count through the odd nimbers from 3 up to $\sqrt{n}$ and check that none divides n. This gives us naive time complexity of $\mathcal{O}(\sqrt{n})$. But the appropriate size measure is $|n|$, which is the length of n. This means, that naive algorithm is exponential in $|n|$.

There are clever more efficient algorithms:

- The algorithm used in practice (Miller-Rabin) uses randomness, and has a positive probability of error. (1970s)
- Breakthrough in 2002 the AKS algorithm: a deterministic polynomial-time algorithm for primality testing.

1.3 Prime factorisation

For given n, compute the list (with multiples) of its prime numbers.

For given n, return two numbers smaller than n whose product is n (if such exist), 0 otherwise. It is an open problem if exist an efficient deterministic/algorithm algorithm to solve this problem.

However, Shor's algorithm (1994) is an efficient quantum algorithm.

1.4 K-colourability

E.g. 3-colourability.

It is an open question if there exists an efficient deterministic (random) algorithm for 3-colourability (or for k-colourabilits if $k \geq 3$).

Equivalent statement: P = NP

These statements are equivalent, because 3-colourability is NP-complete.

Related questions:

- Decision problem: Is a given graph 3-colourable (Does there exists a solution)
- Search problem: Given a graph, find a 3-colouring (if one exists, otherwise return "I can't")
- Counting problem: Count the number of 3-colourings
- Sampling problem: Find a random (w.s.t. uniform distribution) 3-colouring.

1.5 Perfect matchings

Given a bipartite graph (two sided)

A perfect matching is a bijection netween the two sides that follows the edges of the graph.

- Decision problem: efficient algorithm exsists, because
- Search problem: effician algorithm exsists (Ford-Falkerson for network flow ...)
- Counting problem: Open question ($FP \neq \#P$)

We can represent the bipartite graph as an adjacency matrix.

Number of perfect matchings is

$$\sum_{\sigma=perm(n)} \prod_{i=1}^{n} a_{i,\sigma(i)} = perm(A)$$

which is a permanent of A.

2 Asymptotic notation

measures rate of growth.

Typically we want to relate a function

$$T : \mathbb{N} \to \mathbb{R}$$

to a more mathematically natural

$$f : \mathbb{N} \to \mathbb{R}$$

2.1 Big o notation

Main definition:

$$T(n) = \mathcal{O}(f(n)) \iff \exists N \, \exists c > 0 \text{ s.t. } \forall n \geq N \; T(n) \leq c \cdot f(n)$$

This is big $\mathcal{O}$ notation.

We are really saying $T \in \mathcal{O}(f)$.

2.2 Little o notation

Main definition:

$$T(n) = o(f(n)) \iff \forall c > 0 \, \exists N \text{ s.t. } \forall n \geq N \; T(n) \leq c \cdot f(n)$$

We are really saying $T \in o(f)$.

How hard is it to solve computational problems?

We are interested in the resources used, in particular time and space. We need a mathematical model of what an algorithm is that supports an analysis of time and space.

3 Turing machines (1936)

Turing machine is algorithm by definition, sort of computer that runs a fixed program.

- An algorithm should be a finite description of a computational process.
- The algorithm should specify the computational process in its entirety.
- The process of following algorithm should be carried out without any creative input.
- So it is a process that can be carried out by a suitable machine.
- Machine will perform steps in time and need enough working space to carry out calculations.
- We do not bound time and space in advance.

Definition 3.1. A (deterministic) Turing machine (TM) is specified by:

- A finite tape alphabet Γ with $\square \in \Gamma$ ($\square$ is a blank symbol)
- A finite set Q of states with start $\in Q$.
- A partial function $\delta : Q \times \Gamma \to Q \times \Gamma \times \{-1, 0, +1\}$.

A machine configuration is a triple (q, t, i) where

- $q \in Q$
- t is a tape configuration
- i is the position of the head.

A tape configuration is a $t : \mathbb{Z} \to \Gamma$ so that $t(m) \neq \square$ for only finitely many m.

3.1 Turing machine execution

We start the machine in a machine configuration (q, t, i), where $q = start$ and i points at the left-most non $\square$ symbol on the tape

$$i = min\{m \in \mathbb{Z} : t(m) \neq \square\}$$

(if the tape is empty, i can be arbitrary.)

One can define mathematically what it means that δ takes us from machine configuration (q, t, i) to (q', t', i') and what it means that δ halts machine configuration (q, t, i).

3.2 Using a TM as a language recogniser

We assume an input alphabet $\Sigma \subseteq \Gamma \setminus \{\square\}$.

(A language is a subset $L \subseteq \Sigma^*$)

We also assume distinguisged halting states accept, reject $\in Q$.

A TM M decides the language L if for any input word $w \in \Sigma^*$

- if $w \in L$ then M halts in the accept state on input w.
- if $w \notin L$ then M halts in the reject state on input w.

A language is decideable, if there exist a TM M that decides it.

3.3 Using TMs to compute functions

Assume an input alphabet Σ_1 and optput alphabet Σ_2 with $\Sigma_1, \Sigma_2 \subseteq \Gamma \setminus \{\square\}$. A TM M computes a function $f : \Sigma_1^* \to \Sigma_2^*$ if

- for any $q \in \Sigma_1^*$, when given w as input M eventually halts with $f(w)$ on the tape.
- f is computable if there exists a TM that computes f.

3.4 Variant Turing Machines

- Machines in which the tape is infinite in only one direction
- Machines with K tapes (one head per tape)
- Machines with K tapes and many heads per tape, allowing heads to move between tapes

- Machines with multidimensional workspaces instead of one dimentional tapes
- etcetera ...

All such variants are equivalent.

In general a K-tape machine has K tapes each with its own head. So a machine configuration is of the form $(q, t_1, \ldots, t_k, i_1, \ldots, i_k)$ (k tape configurations and k head positions). Transition function has form

$$\delta : Q \times \Gamma^k \to Q \times \Gamma^k \times \{-1, 0, +1\}^k$$

We execute the machine in the "natural way" as in the example (not written).

Definition 3.2. If M is a (k-tape) language recognising TM we define $time_M : \mathbb{N} \to \mathbb{N}$:

$$time_M(n) = max\{\text{no. of steps taken by input } w : w \in \Sigma^*, |w| = n\}$$

For a language $L \subseteq \Sigma^*$ and a function $T : \mathbb{N} \to \mathbb{N}$ we say that $L \in DTime(T(n))$ if there exist $k - tape$ TM M for some $k \geq 1$ that decides L such that $time_M(n) = \mathcal{O}(T(n))$.

Definition 3.3 (Fundemental def.)**.**

$$P := \bigcup_{d \geq 1} DTime(n^d)$$

Equivalent between machine models proved by translations between models. Such translations do not preserve $DTime(T(n))$. But all translations are polynomially bounded in space and time. So the class P is invarient under choice of computational model.

4 Nondeterminism

An nondeterministic TM (NTM) is:

- Γ as before
- Q as before
- A subset $\Delta \subseteq (Q \times \Gamma) \times (Q \times \Gamma \times \{-1, 0, +1\})$

The transition relation: for every $(q, a) \in Q \times \Gamma$ there is a finite set of (q', a', d'), so that $((q, a), (q', a', d')) \in \Delta$. Notation: $(q, a) \to (q', a', d')$

Machine configurations are as before.

A run of the machine is a sequence of configurations starting in the starting configuration and respecting the 'rules' of the transition relation, whose last configuration, if it exists, is a halting/terminating configuration.

NTMs potentially have many different runs from the same starting configuration.

4.1 NTMs as language recognisers

Assume input alphabet $\Sigma \subseteq \Gamma \setminus \{\square\}$ and halting states $accept, reject \in \mathbb{Q}$

An NTM M accepts a word $w \in \Sigma^*$ if there exists a run of M on input w that terminates in accept.

An NTM M rejects a word $w \in \Sigma^*$ if every run of input w terminates in reject.

An NTM M decides the language $L \subseteq \Sigma^*$ if for every $q \in \Sigma^*$:

- if $w \in L$ then M accepts w
- if $w \notin L$ then M rejects w

If M is a NTM, define $\text{time}_M : \mathbb{N} \to \mathbb{N} \cup \{\infty\}$

$$\begin{aligned}\text{time}_M(n) = max\{&\text{no. of steps in the run } r|w \\ &\in \Sigma^*, |w| = n, r \text{ is a run of M on input } w\}\end{aligned}$$

For a language $L \subseteq \Sigma^*$ and function $T(N) : \mathbb{N} \to \mathbb{N}$, we say that $L \in \text{NTime}(T(n))$ if there exists a NTM M that decides L such that $\text{time}_M(n) = \mathcal{O}(T(n))$.

$$NP := \bigcup_k NTime(n^k)$$

$$P \subseteq NP$$

because every deterministic TM is nondeterministic.

Big question is:

$$P = NP$$

Example 4.1. Independent set problem - INDSET.

Input: A graph G and a natural number $k \leq |G|$ - number of vertices of G.

Question: Does G have an independent set of size k.

An independant set or anticlique is a subset I of the vertices of G such that no two elements of I have an edge between them.

We code this up as a decision problem for a language over $\{0, 1, ,, ;\}$

Example encoding (not here) -> Represent a graph as an adjecent matrix. Our input word is 010,101,010;10 (before; is matrix, after; is number of vertices)

If G has n vertices, then our input size is $m^2 + m + \lfloor log_2 k \rfloor + 1$.

We write $\ulcorner$(G, k)$\urcorner$ for the encoding in Σ^* representing the input G, k

INDSET:

$\{w \in \{0, 1, , , ; \}$
$*|w$ is of the form $\ulcorner(G, k)\urcorner$ where G is a graph with an independent set of size k$\}$

We now argue that INDSET belongs to NP.

Given an input word

$$a_1 \dots a_M, \dots$$

Scan to the first comma and count number of characters to obtain M as we go along write a nondeterministic sequence of binary bits on second tape. $\mathcal{O}(m)$

Test to see that there are exactly k 1s on the second tape (if not -> reject). $\mathcal{O}(m^2)$

For each pair of 1s on the second tape, check that there is a 0 in the relevant entry in the adjacency matrix on the first tape. $\mathcal{O}(m^4)$

If so, we accept (otherwise reject)

The time bound is $\mathcal{O}(m^4)$, ie $\mathcal{O}(n^2)$ (because of input is m^2)

Theorem 4.2 (Certificate chatacterisation of NP)**.** T.f.a.e. for $L \subseteq \Sigma^*$

1. $L \in NP$
2. There is a polynomial $p(n) : \mathbb{N} \to \mathbb{N}$ and a polynomial time bounded deterministic TM such that for any input $w \in \Sigma^*$ t.f.a.e.
   (a) $w \in L$
   (b) there exists certificate $v \in \Sigma^{p(|w|)}$ such that M accepts w, v

Dokaz. (2) -> (1) Nondeterministically generate $v \in \Sigma^{p(|w|)}$. Then deterministically check if it is a solution. (1) -> (2) Use the sequence of nondeterministic choices as the certificate. ■

P problems solvable in deterministic polynomial time, NP problems solvable in nondetrministic polynomial time.

$$L \in P \iff L \in DTime(p(n))$$

Problems we can efficiently solve.

$$L \in NP \iff L \in NTime(p(n))$$

Problems for which we can efficiently verify the correctness of a solution.

This intuition is embodied in the certificate characterisation of LP.

5 Meet some NP problems

All these problems are in fact NP-complete.

Main theorem today: Levin theorem: SAT is NP-complete.

Important today: the notions of NP hardness, polynomial time reducibility.

5.1 SAT

A propositional formula is

- a boolean variable: $u, u', u'', u''', u'''', ...$
- a truth value: 1 (true), 0 (false)
- if ϕ, ψ formulas then so are $\phi \cup \psi, \phi \cap \psi, \neg\phi$

If we assign truth values $\{0, 1\}$ to propositional variables

$$v : \text{variables} \to \{0, 1\}$$

then we compute $v(\phi)$ using truth tables.

ϕ is satisfiable, if there exists an assignment v : variable $\to \{0, 1\}$, such that $v(\phi) = \neg$. We call v a satisfying assignment for ϕ.

Example 5.1. $(u \wedge u') \vee (u \wedge u'') \vee (u' \wedge u'')$ This is satisfiable. v is a satisfying assignment $\equiv$ if maps at least two variables to 1.

Example 5.2. $u \wedge \neg u$ This is not satisfiable.

The SAT problem:

- Input: a propositional formula ϕ
- Question: is ϕ satisfiable?

We represent SAT as a language over the aplhabet $\sigma_{SAT} = \{0, 1, \wedge, \vee, \neg, u, '\}$

$$u \wedge (u'' \vee u''')\phi$$

represent as the

$$\wedge u \vee u''u'''\phi'$$

$$SAT = \{\phi' | \phi \text{ is a satisfiable formula}\}$$

SAT $\in$ NP

A certificate for the satisfiability of a formula ϕ containing variables up to u_k is simply a word $v \in \{0, 1\}^*$ representing a stisfying assignment.

We can verify in deterministic polynomial time given input ϕ', v wheter or not v is a satisfying assignment for ϕ.

SAT $\in$ NP by the certificate definition of NP.

Polytime reducibility

We say that a language $L_1 \subseteq \Sigma_1^*$ is polynomial-time (ptime) reducible to a language $L_2 \subseteq \Sigma_2^*$ if there exists a deterministic TM M that compute a function $f : \Sigma_1^* \to \Sigma_2^*$, such that

- for all $w \in \Sigma_1^*$:
$$f(w) \in L_2 \iff w \in L_1$$
and
- M is polynomial time bounded
$$\exists K \text{ s.t. } time_M(n) = \mathcal{O}(n^K)$$

Observations

- $L \leq_p L$
- If $L_1 \leq_p L2$ and $L_2 \leq_p L_3$, then $L_1 \leq_p L_3$
- If $L' \leq_p L \in P$ then $L' \in P$
- If $L' \leq_p L \in NP$ then $L' \in NP$
- If L is NP hard and $L \leq_p L'$, then L' is also NP hard

NP hardness

L is NP hard, if for any NP language L' we have $L' \leq_p L$.

NP completness

L is NP complete, if $L \in NP$ and L is NP hard.

Theorem 5.3 (Cook/Levin)**.** SAT is NP complete.

5.2 3-SAT

A literal is a propositional vaiable u_i or its negation $\neg u_i$.

A clause is a disjunction $l_1 \vee \ldots \vee l_k$ of literals.

A formula is in conjunctive normal form (CNF) if it is a conjunction $D_1 \wedge \ldots \wedge D_m$ where each D_i is a clause.

A 3-clause is a disjunction $l_1 \vee l_2 \vee l_3$ of exactly 3 (not necessarily distinct) literals.

A formula is in 3-CNF if it is a conjunction $D_1 \wedge \ldots \wedge D_m$ of 3-clauses.

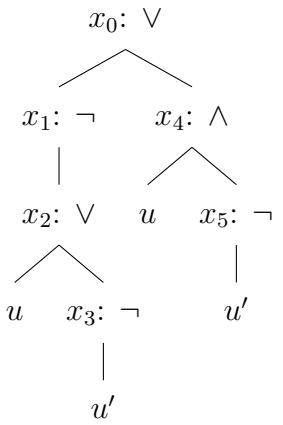
3-SAT:

- Input: a propositional formula in 3-CNF
- Question: is the formula satisfiable?

SAT $\leq_p$ **3-SAT** - The Tseytin transformation.

Example 5.4. Input: $\neg(u \vee \neg u') \vee (u \wedge \neg u')$

From this, we build a tree:



Idea behind this tree:

$$
\begin{aligned}
& x_0 \\
& \wedge (x_0 \leftrightarrow x_1 \vee x_2) \\
& \wedge (x_1 \leftrightarrow \neg x_2) \\
& \wedge (x_2 \leftrightarrow u \vee x_3) \\
& \wedge (x_3 \leftrightarrow \neg u') \\
& \wedge (x_4 \leftrightarrow u \wedge x_5) \\
& \wedge (x_5 \leftrightarrow \neg u')
\end{aligned}
$$

Output:

$(x_0 \vee x_0 \vee x_0) \wedge (\neg x_0 \vee x_1 \vee x_4) \wedge (\neg x_1 \vee x_0 \vee x_0) \wedge (\neg x_4 \vee x_0 \vee x_0) \wedge$
$(\neg x_1 \vee x_2 \vee x_2) \wedge (x_2 \vee x_1 \vee x_1) \wedge (\neg x_2 \vee u \vee x_3) \wedge (\neg u \vee x_2 \vee x_2) \wedge$
$(\neg x_3 \vee x_2 \vee x_2) \wedge (\neg x_3 \vee \neg u' \vee \neg u') \wedge (u' \vee x_3 \vee x_3) \wedge (\neg x_4 \vee u \vee u) \wedge$
$(\neg x_4 \vee x_5 \vee x_5) \wedge (\neg u \vee \neg x_5 \vee x_4) \wedge (\neg x_5 \vee \neg u' \vee \neg u') \wedge (u' \vee x_5 \vee x_5)$

5.3 3-COL

Given a cojnunction of 3-clauses ϕ. Convert it to a graph that is 3-colourable iff ϕ is satisfiable.

Suppose ϕ has variables up to u_k.

Dokaz. (that SAT is NP hard)

Suppose $L \in NP$. We show $L \leq_p SAT$.

Let M be a single-tape ND TM that decides L with time bound $p(n) \geq time_M(n)$. (p a polynomial).

If we run M on input w.

The machine can only during its execution look at squares between $-p(n)$ and $p(n)$.

Modify M so that from any halting state it makes a dummy step (does nothing). We will call it M'.

Given an input word $w \in \Sigma$ we construct a propositional formula Φ_w such that

$$\begin{aligned} \Phi_w \text{ satisfiable} &\iff \text{there exists a run of M' on input w} \\ &\qquad \text{that is in the accept state after } p(|w|) \text{ step} \\ &\iff w \in L \end{aligned}$$

Φ_w will use $(p(|w|)+1) \times |Q| + (p(|w|)+1) \cdot (2p(|w|)+1) \cdot (|\Gamma|+1)$ variables. This is polynomial in $|w|$ $\mathcal{O}((p(n))^2)$.

Mnemonic names for the variables:

- $(\text{state}_t = q)$ for every $t = 0 \ldots p(|w|)$, $q \in Q$
- $(\text{symbol}_{t,i} = a)$ for every $t = 0 \ldots p(|w|)$, $i = -p(|w|), \ldots, p(|w|)$, $a \in \Gamma$
- $(\text{head}_t = i)$ fpr every $t = 0 \ldots p(|w|)$, $i = -p(|w|) \ldots p(|w|)$

Define Φ_w in such a way that its satiyfying assignments are in correspondence with runs of length (time steps) $p(|w|)$ that end in accept state.

We break its definition into parts.

$$\begin{aligned} Q_{init-w} := \vee_{o \le i < |w|} (Symbol_{0,i} = w_i) \wedge (head_0 = 0) \\ \wedge (state_0 = start) \\ \wedge (\vee)_{i=-p(|w|)\cdots-1} (symbol_{0,i} = \square) \end{aligned}$$

■

6 Several time and space complexity classes.

$L \in \text{DTime}(T(n))$:

$\exists$ deterministic TM M that decides out such that $time_M(n) = \mathcal{O}(T(n))$

$L \in \text{NTime}(T(n))$:

$\exists$ nondeterministic TM M that decides out such that $time_M(n) = \mathcal{O}(T(n))$

$$P := \bigcup_{\mathbb{P}:\mathbb{N}\to\mathbb{N} \text{ (polynomial)}} DTime(p(n))$$

$$NP := \bigcup_{\mathbb{P}:\mathbb{N}\to\mathbb{N} \text{ (polynomial)}} NTime(p(n))$$

Proposition: P is closed under complement.

If $L \in P$ then $\overline{\text{L}} \in P$, $\overline{\text{L}} = \Sigma^* \setminus L$

Proof: Switch the accept and reject states.

(Open)Question: Is NP closed under complement? (It is not known if $\neg SAT \in NP$)

The same proof does not work, because of the quantification over runs in how a nondeterministic machine decides a language.

$$coNP := \{\overline{\text{L}} | L \in NP\}$$

What is a complement of SAT?

UNSAT = $\{\Phi | \Phi$ is an unsatisfiable prop formula$\}$

$\overline{\text{SAT}}$ = UNSAT $\cup+$ the set of all words that are not ... of formulas. (Junk)

UNSAT $\in$ coNP! UNSAT is a coNP-complete problem.

coUNSAT = SAT U Junk

Known inclusions:

$$P \subseteq NP \cap coNP \begin{cases} \subsetneq \text{NP} \\ \subsetneq \text{coNP} \end{cases}$$

It si not known if any of the above includes are strict. We dont know if SAT $\in$ coNP and UNSAT $\in$ NP.

The problem of determining the winner of a parity game is in NP $\cap$ coNP, but not known to be in P.

6.1 Space complexity

Measure how much working space a TM needs to use (not including space taken by the input).

Assume TMs have a separate input tape with a read-only head and K working tapes with read-write heads.

$\text{SPACE}_M(n) = \max\{$total number of spaces vidited by working tape heads on run $r|$ $w \in \Sigma^*, |w| = n, e$ is a run of M on input $w\}$

$L \in DSpace(T(n))$: deterministic TM M that decides for language such that

$$space_M(n) = O(T(n))$$

$L \in NSpace(T(n))$: nondeterministic TM M that decides for language such that

$$space_M(n) = O(T(n))$$

$$PSPACE = \bigcup_{\substack{\mathbb{P}:\mathbb{N}\to\mathbb{N} \\ \text{(polynomial)}}} DSpace(p(n))$$

$$NPSPACE = \bigcup_{\substack{\mathbb{P}:\mathbb{N}\to\mathbb{N} \\ \text{(polynomial)}}} NSpace(p(n))$$

Logaritmic space:

$$L = DSPACE(Log(n))$$

Nondeterministic logaritmic space:

$$NP = NSPACE(Log(n))$$

Proposition: NP $\subseteq$ PSPACE

Dokaz. Suppose $L \in NP$. By the certificate characterisation of NP there exists a deterministic PTime TM M and a polynomial $p(n) : \mathbb{N} \to \mathbb{N}$, s.t. $w \in L \iff \exists v \Sigma^{p(n)}$ M accepts w.v.

We adapt M to a deterministic polyspace TM for L.

Adapted machine: If $w \in \Sigma^n$ is on input tape repeat for every $v \in \Sigma^{p(n)}$ on working tape we write w.v run M on input on working tape, accept w if M accepts w.v. If no certificate is found, at end reject.

Takes exponential time but only polynomial space. ■

$$EXP = \bigcup_{\substack{\mathbb{P}:\mathbb{N}\to\mathbb{N} \\ \text{(polynomial)}}} DTime(2^{p(n)})$$

$$NEXP = \bigcup_{\substack{\mathbb{P}:\mathbb{N}\to\mathbb{N} \\ \text{(polynomial)}}} NTime(2^{p(n)})$$

Theorem 6.1. NPSPACE $\subseteq$ EXP

Dokaz. idea: Look at all possible configurations of a nondeterministic TM that uses only polynomial space. There are only exponentially many such configurations. Using exponential time we can search the graph of configurations to see if there is a path to an accepting state. ■

We now have:

$$L \subseteq NL \subseteq P \subseteq NP \subseteq PSPACE \subseteq EXP \subseteq NEXP$$

The only inclusions known to be struct are:

$$L \subset PSPACE \text{ and } P \subset EXP$$

The only known to be not strict is:

$$PSPACE = NPSPACE$$

EXP $\subseteq$ NEXP is not known to be strict. In fact:

Theorem 6.2. $EXP \neq NEXP => P \neq NP$

Dokaz. Suppose $L \in NEXP \setminus EXP$. Let $p(n)$ be a polynomial such that L is decided by nondet TM M that runs in $2^{p(n)}$ steps. Consider the language $(L' = \{0^{2^{p(|w|)}}, w | w \in L\})$. We claim $L' \in NP \setminus P$.

$L' \in NP$

Take an input v. Deterministically check if it of the form $0^{2^{p(|w)}}, w$ (othervise reject). Then run the nondet. M on input w. This is a polytime nondeterministic machine.

$L' \notin P$ Suppose there is a deterministic TM M' that decides L' in polynomial time $q(n) : \mathbb{N} \to \mathbb{N}$.

We use this to decide L by Given input wl write $0^{2^{p(|w|)}}, w$ onto the tape. Run M' on the above input.

This takes time: $\mathcal{O}(2^{p(|w|)})$ for setting np. $\mathcal{O}(q(c \cdot 2^{p(|w|)}))$ for running M' $= \mathcal{O}(2^{poly(|w|)})$ ■

7 PSPACE Completeness

Todays content:

- The TQBF problem
- TQBF $\in$ PSPACE
- * TQBF is NSPACE complete
- hence PSPACE = NPSPACE

7.1 QBF

[quantified boolean formulas]

A quantified boolean formula is:

- a boolean variable: $u, b, \ldots$
- a boolean constant: 0 (false), 1 (true)
- if Φ, Ψ are quantified boolean formulas, then so are: $\Phi \vee \Psi, \Phi \wedge \Psi, \neg\Phi$

Every propositional formula is a qbf (with no quantifiers).

If Φ is a propositional formula with variables $u_1, \ldots, u_k$, then

- $\exists u_1 \ldots \exists u_k \Phi$ is true $\iff$ Φ is satisfiable.
- $\forall u_1 \ldots \forall u_k \Phi$ is true $\iff$ Φ is a tautology (valid).

A variable in a qbf is free if it does not accur inside the scope of a quantifier. bound otherwise.

A qbf Φ is closed if it has no free variables. A closed qbf has a determined truth value.

7.2 TQBF problem

Input: a closed qbf Φ

Question: Is Φ true?

As a language

$$TQBF = \{\Phi | \Phi \text{ is a closed qbf formula and } \Phi \text{ is true}\}$$

Proposition: TQBF $\in$ PSPACE

Proof idea illustrated using one example.

Input: $\forall u \exists v (u \cap v) \cup (\neg u \cap \neg v)$

L is PSPACE hard, if for every PSPACE language L' $L' \leq_p L$. L is NPSPACE hard, if for every NPSPACE language L' $L' \leq_p L$.

Theorem 7.1 (Meyer & Stockmeyer 1973)**.** TQBF is NPSPACE hard. (hence also PSPACE hard).

Corollary: PSPACE = NPSPACE

Dokaz. Suppose $L \in$NPSPACE. Let f be a ptime reduction from L to TQBF. Then we have the following deterministic polynomial space algorithm for L. Given input w

1. compute $f(w)$
2. decide if $f(w)$ is true.

■

Suppose L is decided by a nondeterministic TM M where space is bounded by $p(n)$ polynomial.

For convenience assume that the machine M loops on the accept, and only ever accepts after erasing all data on the working tapes and reset head positions.

We define the configuration graph of M run on an input word w. Vertices are configurations:

- a state $q \in Q$
- w on the input tape + input tape head position
- $v_1, \ldots, v_k \in \Gamma^{p(|w|)}$ representing the relevant data on the K working tape + K head positions.

Such a configuration c can be encoded by word "c" $\in \{0,1\}^{ap(|w|)}$

The edges in the graph $c \to c'$ if M can make one step of computation from configuration c to configuration c'.

We construct a propositional formula $\Phi_{M,w}(u_1, \ldots, u_N, v_1, \ldots, v_N)$, where $N = a \cdot p(|w|)$ such that

- The size of $\Phi_{M,w}$ is $\mathcal{O}(n)$.

- For $\overrightarrow{a}, \overrightarrow{b} \in \{0,1\}^N \ldots$

  $\Phi_{M,w}(\overrightarrow{a}, \overrightarrow{b}) = 1 \iff \overrightarrow{a} = \ulcorner c \urcorner$ and $\overrightarrow{b} = \ulcorner c' \urcorner$ for 2 node c, c' in the configuration graph and $c \to c'$.

The formula $\Phi_{M,w}$ is defined in an analogous why to the formula constructed in the proof of Cook-Levin theorem.

We construct qbformulas $\Psi_i(u_1, \ldots, u_N, v_1, \ldots, v_N)$. Satisfying

- The size of Ψ_i is polynomial in N.
- For c, c' in the config graph $\Psi(\ulcorner c \urcorner, \ulcorner c' \urcorner) = 1 \iff$ there is a directed path of length 2^i from c to c'.

Given this we are done, because the required ptime reduction from L to TQBF is:

- given the input word w, construct $\Psi_{M,w}(\vec{u}, \vec{v})$ as above, and hence $\Psi_N(\vec{u}, \vec{v})$
- Then $\Psi_N(\ulcorner start \urcorner, \ulcorner accept \urcorner) = 1$
  $\iff \exists$ path of length 2^N from start to accept configuration
  $\iff \exists$ path from start to accept configuration
  $\iff w \in L$
- The closed formula $\Psi_N(\ulcorner start \urcorner, \ulcorner accept \urcorner)$ has size polynomial in $N = cp(|w|)$ hence polynomial in $|w|$

8 Logarithmic space

Recall:

- L = DSPACE(log(n))
- NL = NSPACE(log(n))

Two examples in L

$$DIV3 = \{w \in \{0,1\}^* | \exists n \in \mathbb{N}\ w = bin(n) \text{ and } 3|n\}$$
$$MULT = \{bin(m), bin(n), bin(l) | m, n, l \in \mathbb{N} \text{ and } m \cdot n = l\}$$

8.1 The PATH problem

Input: A directed graph G and two vertices $s, t \in G$. Question: Is there a directed path from s to t?

As a language

$$PATH = \{(G', s', t') | \text{G is a graph given as a adjacent matrix } G', s'$$
the binary representation of vertex s, t' the binary representation of vertex t$\}$

For a graph G with k vertices such a truple has size $\mathcal{O}(k^2)$.

Vertices are represented by words of size $\mathcal{O}(logk)$. This is also $\mathcal{O}(logn)$ when $n = \mathcal{O}(k^2)$.

Proposition: PATH $\in$ NL.

Dokaz. Observation: If there is a path from s to t in G with k vertices, then there is a path of length $\leq k$.

Algorithm (nondeterministic):

- Use two working tapes (A and B).
- Start of algorithm given input G', s', t'
- Write s' on tape A and bin(1) on tape B.

*: At every successible step check tape A to see if t' is on tape A => if so accept.

Check tape B to see if $bin(k)$ (no. of vertices in G) is on B => if so accept.

Otherwise consider the vertex v with $bin(v)$ on tape (A).

Nondeterministically choose a vertex v' s.t. $v \to v'$ in G. Write w'' on tape A. increment counter on tape B. Go back to *

We have PATH $\in$ NL.

We will show PATH is NL-hard. Hence, PATH is NL-complete. ■

L is NL-hard if for every NL language L', we have a deterministic logarithmic space reduction from L' to L.

Reduction uses a TM with read only input tape and write only output tape and k working tapes. We only count the space on working tapes.

Notation:

$$L' \leq_L L$$

Theorem 8.1. PATH is NL-complete.

Dokaz. Suppose L is decided by a nondeterministic k tape TM M, whose space is bounded by $c \cdot log(n)$.

(Assume that M erases its working tapes and resets head position of the input tape before entering the accept state).

As last week consider the configuration graph of M on an input word w. Configurations have:

- a state $q \in q$
- w on input tape + head position on input tape
- $v_1, \ldots, v_k \in \Gamma^{c \cdot log|w|}$ configurations of working tapes + k head positions.

The space we need to represent an invividual configuration is $\mathcal{O}(log|w|)$. Any configuration c can be represented by $c' \in \{0,1\}^{a \cdot log|w|}$ for some a.

The required logspace reduction $w \to G, s, t$ where:

- G is the adjacent matrix of the configuration graph of M on W
- s is the start configuration
- t is the accept configuration

This reduction is only logarithmic space by the following algorithm: Construct the adjacency matrix of G by a nested for loop.

```
1: for i ∈ {0,1}^{a·log|w|} do
2:     for j ∈ {0,1}^{a·log|w|} do
3:         if i = c' and j = c' and c → c' then
4:             add 1 to the accept tape
5:         else
6:             put 0 otherwise
7:         end if
8:     end for
9: end for
```

■

9 Probabilistic TMs and class BPP

A simple probabilistic TM

$$(start, \_) \to (start, \_, 0)$$
$$(start, \_) \to (accept, \_, 0)$$

The probabilistic TM chooses the transition by tossing a fair coin.

- The worst case run time is ∞
- The probability of termination is 1
- The expected run time is $\frac{1}{2} \cdot 1 + \frac{1}{4} \cdot 2 + \frac{1}{8} \cdot 3 \cdots = \sum_{n=1}^{\infty} \frac{n}{2^n} = 2$

A probabilistic TM is specified by (Q, Γ, Δ), exactly the same data as a nondeterministic TM. The difference is in the execution model (how we run the machine).

- Start in the start state
- Whenever the machine is in a configuration (q, t, h) and there are K possible transitions to K potential successor configurations, the machine chooses one of these successors with uniform probability $\frac{1}{k}$.

When the machine is executed on an input word w we either get

- a finite run $c_1 \to \ldots \to c_N$ where c_N is a terminal configuration (always with a positive probability)
- an infinite run $c_1 \to c_2 \to \ldots$ that never reaches a terminal configuration (might happen with positive probability, or might happen with probability 0)

If M is a PTM:

Worst-case time: ($\equiv$ time for M as a NTM)

$wtime_M(n) = \max\{$no. steps in the run r | r is a run of M on input w,$|w| = n\}$

This is not appropriate for probabilistic computation.

More relevant is Expected time:

$$xtime_M(N) = max\{E[\text{number of steps in a run } r \text{ of } M \text{ on } w] || w| = n\}$$

Example 9.1. Finding the k-th smallest element in an unsorted array.

Input: n numbers, k

Output: find the k-th smallest a_i, ie the $k-th$ element if we sort $a_1, \ldots, a_n$

An easy algorithm: sort the array, then select the k-th element from the array. Time complexity is $\mathcal{O}(n)$.

A probabilistic algorithm with expected time $O(n)$:

A recursive probabilistic algorithm Find (k; $a_1, \ldots, a_n$)

- Randomly choose $i \in \{1, \ldots, n\}$
- Go through $a_1, \ldots, a_n$ in turn and we return a permutation of $a_1, \ldots, a_n$ of the form $b_1, \ldots, b_l, c_{l+1}, \ldots, c_m, d_{m+1}, \ldots, d_n$ where $0 \leq l < m \leq n$
- if $l + 1 \leq k \leq m$ return a_i
- if $k \leq k$ recursively compute Find $(k; b_1, \ldots, b_l)$
- if $k > n$ recursively compute Find $(k - m; d_{m+1}, \ldots, d_n)$

Proposition: $wtime_{Find}(n) = \mathcal{O}(n^2)$

Proposition: $xtime_{Find}(n) = \mathcal{O}(n)$

Dokaz. We assume for convenience that all $a_1, \ldots, a_n$ are distinct. So always $m = l + 1 =$ the order rank of a_i in $a_1, \ldots, a_n$. From the algorithm we have for some constant c $T(n) \leq c \cdot n + \frac{1}{n}[\sum_{m=1}^{k-1} + \sum_{m=k+1}^{n} T(m-1)]$ $T(1) \leq c$

We prove by induction the $T(n) \leq 4cn$

$$\begin{aligned} T(n) \leq & cn + \frac{1}{n}[\sum_{m=1}^{k-1} T(n-m) + \sum_{m=k+1}^{n} T(m-1)] \\ \leq & cn + \frac{4c}{n}[\sum_{m=1}^{k-1}(n-m) + \sum_{m=k+1}^{n}(m-1)] \\ \leq & cn + \frac{4c}{n} \cdot \frac{3}{4}n^2 \\ = & 4cn \end{aligned}$$

■

(visual proof v profesorjevih zapiskih)

10 Primality testing

The Miller-Rabin property for prime numbers.

If $p = 2^s \cdot d + 1$ is prime, where d is odd, then for every number $a \in \{1, \ldots, p-1\}$ one of the following holds;

a $a^d \equiv 1 mod n$

b $a^{2^r \cdot d} \equiv -1 mod n$ for some $r \in \{0, \ldots, s-1\}$

If $n = 2^s \cdot d + 1$ is composite (d odd, $s \geq 1$), then at most $\frac{n}{4}$ of $a \in \{1, \ldots, n'1\}$ satisfy one of (a) or (b) holds.

Miller-Rabin algorithm for large odd n $= 2^s \cdot d + 1$ in

- Randomly choose $a \in \{1, \ldots, n-1\}$
- check if one of (a) or (b) holds

if so return **PROBABLY Prime** if not return **COMPOSITE**.

If repeated k times the algorithm returns always **PROBABLY Prime**, then there is only a probability of $\frac{1}{4^k}$ of error.

11 BPP

(Bounded Probabilistic Polynomial)

A PTM runs in expected polynomial time if there is a polynomial $p(n)$ such that $xtime_M(n) = \mathcal{O}(p(n))$.

A PTM decides L with bounded error

- if on any input $w \in \Sigma^*$, M halts with probability 1 in either accept or reject.
- if $w \in L$, P(M on w halts in accept) $\geq \frac{2}{3}$
- if $w \notin L$, then P(M on w halts in reject) $\geq \frac{2}{3}$

BPP = $\{L|$ there exists an expected ptime probabilistic TM that decides L with bounded error$\}$

12 The graph isomorphism problem (GI)

Input: G_1, G_2, two graphs with the same number of vertices given as adjecency matrices.

Question: Are G_1 and G_2 isomorphic?

As a language:

$$GI = \{\ulcorner G_1 \urcorner, \ulcorner G_2 \urcorner | G_1 \text{ and } G_2 \text{ are isomorphic } (G_1 \cong G_2)\}$$

$$GNI = \{\ulcorner G_1 \urcorner, \ulcorner G_2 \urcorner | G_1 \text{ and } G_2 \text{ are not isomorphic } (G_1 \ncong G_2)\}$$

GI and GNI are not known to be in P.

Babai: GI is quasipolynomial, in fact decidable in $DTime(2^{(\log n)^3})$.

GI is clearly in NP (take the isomorphism itself as the certificate).

GNI is clearly in coNP but not known to be in NP.

How can we "certify" that two graphs are not isomorphic?

By considering interactive protocols it is possible for a prover that knows that 2 graphs are not isomorphic to convince a verifier (that is polynomially time bounded but can use probability) of this fact with a probability of error that can be made arbitrarily small.

This is an example of an interactive proof.

An interactive proof showing that $G_1 \ncong G_2$

Assuming the 2 graphs are not isomorphic.

1. **V** picks a random $i \in \{1, 2\}$ by tossing a fair coin and a random permutation σ on $\{1, 2, \ldots, n\}$ with the uniform distribution.

   V communicates $G_3 = \sigma(G_i)$ to the prover.

2. **P** chooses j where $j = 1$ if $G_3 = G_1$ else 2 and communicates j to **V**.

3. **V accept** if $j = i$ else **reject**.

In total:

- If the two graphs are not isomorphic then **V accept**s with probability 1.

- If the two graphs are isomorphic irrespective of the algorithm that **P** uses for its part of the interaction.
- $P(\textbf{accept}) = \frac{1}{2}$

Completeness

There exists a behaviour of P such that if $G_1 \ncong G_2$ then $P(\textbf{accept}) = 1$.

Soundness

For any P that conforms to the protocol (returns $j \in \{1, 2\}$) If $G_1 \cong G_2$ then $P(\textbf{reject}) = \frac{1}{2}$

Argument

The input to P is a graph uniformly at random from the collection of graphs isomorphic to G_1 and G_2. This gives no information about i.

The chance of j is independent of i. So $Pr(i \neq j) = \frac{1}{2}$.

12.1 Interactive protocols between V and P

V is given by a probabilistic TM.

P is any function $\Sigma^* \to \Sigma^*$.

A K-round interaction on input word $w \in \Sigma^*$ is (k can depend on w):

- **V** computes $r_1, v_1 = V(w)$

  P computes $p_1 = P(w, v)$
- ...
- **V** computes $r_k, v_k = V(w, r_1, v_1, p_1, \ldots, r_{k-1}, v_{k-1}, p_{k-1})$

  P computes $p_k = P(w, v_1, p_1, \ldots, v_{k-1}, p_{k-1}, v_k)$
- **V** halts in state $V(w, r_1, v_1, p_1, \ldots, r_k, v_k, p_k) \in \{\textbf{accept}, \textbf{reject}\}$

Definition 12.1. IP (languages with interactive proofs)

$L \in IP$ if there exists an expected polytime probabilistic TM V such that:

- completeness:

$\exists$ proover that interacts with V such that

$$\forall w \in L : Pr(Out < v, p, w >= \textbf{accept}) \geq \frac{2}{3}$$

- soundness:

  $\forall$ proovers P that interact with V and

$$\forall w \notin L : Pr(out < v, p, w >= \textbf{reject}) \geq \frac{2}{3}$$

Where **P** interacts with **V** means

- $\forall w, v_1, p_1, v_2, \ldots, v_i \; |P(w, v1, p1 \ldots)| \leq p(|w|)$
- $\forall w \in \Sigma^*$ with probability 1 the **P**, **V** interaction on w halts in **accept** or **reject** in a number of rounds polynomial in the length of w.

13 Zero-knowledge proofs

Recall

$$GI = \{\ulcorner G_1 \urcorner, \ulcorner G_2 \urcorner | G_1 \cong G_2\}$$

Suppose **P** knows an isomorphism $\pi : G_1 \to G_2$. **P** can convince **V** that the graphs are isomorphic by communicating π to **V**.

We will show, that **P** can convince **V** that G_1 and G_2 are isomorphic without giving any information about the isomorphism π (or indeed any other isomorphism).

13.1 ZKP for GI

Assume that **P** knows an isomorphism $\pi : [1, \ldots, n] \to [1, \ldots, n]$, $\pi(G_1) = G_2$.

1. **P** generates a random permutation $\pi' : [1, \ldots, n] \to [1, \ldots, n]$

   P communicates the graph $G_3 = \pi'(G_2)$ to **V**.

2. **V** tosses a coin $i \in \{1, 2\}$ with half-half probability and communicates i to **P**.

3. **P** returns to **V** $\pi'' =$(π' if $i = 2$ or $\pi' \circ \pi$ if $i = 1$).

4. **V** checks that $\pi''(G_i) = G_3$ and if so **accept**s else **reject**s.

- Completeness:

  If π is a graph isomorphism with $G_2 = \pi(G_1)$ then **V** accepts with probability 1.

- Soundness:

  If $G_1 \ncong G_2$ then for any proover **P** interacting with **V** probability of reject $\geq \frac{1}{2}$.

- Zero-knowledge:

  If $G_1 \cong G_2$ amd **P** behaves following the rules then no $\mathbf{V}^*$ learns anything about π from the interaction.

For any polytime probabilistic $\mathbf{V}^*$ that interacts with **P**, there exists a ptime probabilistic $\mathbf{S}^*$ that simulates the result of $\mathbf{V}^*$, **P** discourse without any recourse to π. I.e.,

$$\forall(\ulcorner G_1 \urcorner, \ulcorner G_2 \urcorner) \in GI \ldots$$

$\mathbf{S}^*$ we have $G_1 \cong G_2$

1. Chooses a random $j \in \{1, 2\}$ (via fair coin) and random permutation π' and generates $G_3 = \pi'(G_1)$
2. $\mathbf{S}^*$ uses $\mathbf{V}^*$ to do what it does to return $i \in \{1, 2\}$
3. if $i \neq j$ go back to step 1
4. simulate the second round of $\mathbf{V}^*$ on input π''

Suppose P knows a 3-colouring C of a graph G

1. **P** generates a random permutation $\sigma : \{R, G, B\} \to \{R, G, B\}$.
2. **V** chooses random edge of a graph.
3. **P** opens those envelopes (connected vertices to this edge).
4. **V** accepts if the two colours in the envelopes are different, reject if same.

- Completeness:

  If G is 3-colourable and C is a 3-colouring then $P(\textbf{accept}) = 1$

- Soundness:

  If there is no 3-colouring then for any $\mathbf{P}^*$ we have $P(\textbf{reject}) \geq \frac{1}{n^2}$

- Zero-knowledge:

  If C is a 3-colouring of G and **P** is behaving as its supposed to then no probabilistic ptime verifier learns anything about C from the interaction.

The sealed envelopes are implemented using bit commitment.

A one-way function is a ptime computable $f : \Sigma^* \to \Sigma^*$ such that for any ptime A there is a negligible function Σ such that for all n:

$$Pr_{x \in \Sigma^n}[A(f(x)) = x's \cdot tf(x') = f(x)] < \Sigma(n)$$

Proposition: If 1-way function exists then $P \neq NP$

A one-way permutation is a one-way function with two additional properties:

- f is one-to-one (injective)

- $\forall w \in \Sigma^* |f(w)| = |w|$

Follows that f is a permutation on Σ^n

Assumption for next part of lecture: a 1-way permutation exists.

Theorem 13.1 (Goldreich-Levin theorem)**.** Suppose f is a 1-way permutation. Then for every probabilistic ptime algorithm there is a negligible function Σ s.t.

$$Pr_{x \in \{0,1\}^n, r \in \{0,1\}^n}[A(f(x), r) = x \odot r] \leq \frac{1}{2} + \Sigma(n)$$

where

$$[x \odot r = \sum_{i=1}^{n} x_i r_i mod 2]$$

To commit a bit b:

- Pick a sufficiently safe n
- Randomly choose x, r $\in \Sigma^n$ such that xor = b
- Then (f(x), r) is my 'sealed envelope' and can be publicly revealed
- To reveal the information, reveal x

14 $IP = PSPACE$

This section is not completed. This topic is required only for challenge exam.

15 Complexity classes

Decision problems are represented by functions $\Sigma^* \to \{0,1\}$. We have lots of complexity classes for such functions. Today some complexity classes for more general functions $\Sigma_1^* \to \Sigma_2^*$.

15.1 FP (Function polytime)

A function $f : \Sigma_1^* \to \Sigma_2^*$ is in FP if there exsists a multitape TM M that computes f and a polynomial $p(n)$ such that

$$\forall w \in \Sigma_1^* time_M(w) \leq p(|w|)$$

Remark. If $f \in FP$ then there exsists a polynomial $p(n)$ such that $\forall w \in \Sigma_1^*$ $|f(w)| \leq p(|w|)$.

(We already used this in polytime reducibility.)

Remark.

$$f : \mathbb{N} \to \mathbb{N}$$
$$f : \Sigma^* \to \mathbb{N}$$
$$f : \mathbb{N} \to \Sigma^*$$

We say f is in FP if function that maps binary representations of n to binary frpresentations of $f(n)$ is in FP.

15.2 Counting problems

Counting problems provide an interesting class of computationally interesting functions.

15.3 Some examples

- $\#SAT : \Sigma_{PROP}^* \to \mathbb{N}$

  $$\#SAT(\Phi) = \text{number of satisfying assignments for } \Phi$$

- $\#3COL : \Sigma_{Graph}^* \to \mathbb{N}$

  $$\#3COL(G) = \text{number of 3-colourings of } G$$

15.3.1 #P

"Sharp P"

A function $f : \Sigma^* \to \mathbb{N}$ belongs to #P if there exsists a deterministic polytime M and polynomial p s.t.

$$f(w) = |\{v \in \Sigma^{p(|w|)} | M(w, v) \textbf{ accept}\text{s}\}|$$

Recall: $L \in NP$ iff $\exists$ deterministic polytime TM M and polynomial p s.t.

$$w \in L \iff \exists v \in \Sigma^{p(|w|)}.M(u, v) \textbf{ accept}\text{s}$$

(the certificate definition of NP)

Remark. Obvious that #SAT $\in$ #P, #3COL $\in$ #P ...

Fact: FP $\subseteq \#P$

Proposition: if $FP = \#P$ then $P = NP$.

Dokaz. Suppose #SAT $\in$ FP. Then we have a P decision procedure for SAT.

Input: Φ Output: #SAT(Φ) $\neq 0$

It is atleast as hard to count certificates as is it to check if a certificate exsists. ■

(In words:)

Dokaz. If $FP = \#P$ then we have a deterministic polytime algorithm for counting the number of satisfying assignments of a propositional formula Φ. By accepting if this number $\neq 0$ and rejecting if this number $= 0$ we obtain a deterministic polytime decision procedure for SAT. ■

More interesting: sometimes it is strictly harder to count certificates then to check if a certificate exsists. (assuming $P \neq NP$)

15.4 Cycles in directed graphs

Input is a directed graph G.

The decision problem CYCLE:

- does G have a (directed) cycle?

This is in P (actually solvable in linear time).

The counting problem #CYCLE:

- #CYCLE(G) = the number of simple cycles in G.

(a simple directed cycle of length n is a sequence $v_0 \to v_1 \to ... \to v_{n-1} \to v_0$ in which no vertex appears twice)

Proposition:

1. $CYCLE \in P$
2. If $\#CYCLE \in FP$ then $P = NP$

Dokaz. We will proove each part individually.

1. By depth-first search
2. We prove that if #CYCLE $\in$ FP then HAM $\in P$ Since HAM is NP-complete, $P = NP$.

   Suppose we have an FP algorithm for #CYCLE. We'll define a P procedure for HAM.

   Let G with n vertices be an input graph. Construct a graph G' by replacing every vertex w in G with the following gadget: (not included, see images on moodle)

   Resulting graph has $2n(n\lceil log_2 n \rceil + 1)$ vertices.

   Important fact: There are $m = 2^{n\lceil log_2 n \rceil}$ paths from u to v in G'.

   So for each cycle of length l in G there are 2^{ml} corresponding cycles in G'. (Relies on cycles being identified up to rotation equivalence.)

   A Hamiltonian cycle in G is a simple cycle of length n.

   If G has a Hamiltonian cycle then G' has at least $2^{n^2\lceil log_2 n \rceil}$ simple cycles.

   If G does not have a Ham. cycle, then G has at most n^{n-1} simple cycles.

   So G' has at most $n^{n-1} \cdot 2^{m(n-1)}$ simple cycles.

$$\begin{aligned} n^{n-1} \cdot 2^{m(n-1)} =& n^{n-1} \cdot 2^{n\lceil log_2 n \rceil (n-1)} \\ =& n^{n-1} \bar{n}^{n(n-1)} \\ \leq& \bar{n}^{(n-1)+n(n-1)} \\ =& \bar{n}^{n^2-1} \\ <& \bar{n}^{n^2} \\ =& 2^{n^2 \lceil log_2 n \rceil} \end{aligned}$$

Decide "G has a Hamiltonian cycle" by:

$$\#CYCLE(G') \geq 2^{n^2 \lceil log_2 n \rceil}$$

■

Given a function $f : \Sigma^* \to \Sigma^*$

A deterministic TM with an oracle for f is a TM with an additional tape (the oracle tape) and a more general transition (partial) function

$$\delta : Q + \Gamma^{k+1} \to Q((\Gamma^{k+1} \times \{-1, 0, 1\}^{k+1}) \cup \{call - f\})$$

a transition $(q, a_1, \ldots, a_{k+1}) \to (q', call - f)$ change the state to q' and replace the word w on the oracle tape with the word $f(w)$.

A function $g : \Sigma_1^* \to \Sigma_2^*$ is in FP^f if there exists an oracle TM that computes g in polynomial time.

Lemma 15.1. If $f \in FP$ and $g \in FP^f$. Then $g \in FP$.

A function f is #P-hard if, for every $g \in \#P, g \in FP^f$.

f is #P-complete if $f \in \#P$ and f is #P-hard.

Theorem 15.2. $\#SAT$ is $\#P$-complete.

(no proof)

Its not that much material for today.

16 Synthesis based on the permanent

Reminder

If A is an $(n \times n)$-matrix

$$det(A) = \sum_{\sigma \in perm(n)} \prod_{i=1}^{n} sgn(\sigma) \cdot A_{i,\sigma(i)}$$

$$perm(A) = \sum_{\sigma \in perm(n)} \prod_{i=1}^{n} A_{i,\sigma(i)}$$

det(A) is important in linear algebra, while perm(A) is important in combinatorics.

The permanent of a $\{0,1\}$-matrix counts the number of perfect matchings in the bipartite graph whose bipartite-adjacency matrix is A.

(Bipartite-adjacency matrix shows vertices of one side as rows and vertices of other as columns.)

A perfect matching is a collection of edges such that each vertex has exactly one edge in the collection.

The determinant and perminant are both downwards self-recuvible, i.e. they can be computed in polynomial time given an oracle solving smaller instances.

$$det(\begin{bmatrix} 3 & 1 & 4 \\ 1 & 5 & 9 \\ 2 & 6 & 5 \end{bmatrix}) = 3 \cdot det(\begin{bmatrix} 5 & 9 \\ 6 & 5 \end{bmatrix}) - 1 \cdot det(\begin{bmatrix} 1 & 9 \\ 2 & 5 \end{bmatrix}) + 4 \cdot det(\begin{bmatrix} 1 & 5 \\ 2 & 6 \end{bmatrix})$$

$$perm(\begin{bmatrix} 3 & 1 & 4 \\ 1 & 5 & 9 \\ 2 & 6 & 5 \end{bmatrix}) = 3 \cdot perm(\begin{bmatrix} 5 & 9 \\ 6 & 5 \end{bmatrix}) + 1 \cdot perm(\begin{bmatrix} 1 & 9 \\ 2 & 5 \end{bmatrix}) + 4 \cdot perm(\begin{bmatrix} 1 & 5 \\ 2 & 6 \end{bmatrix})$$

Downwards self reduciblility doesnt provide a polytime algorithm. But it is useful in giving IP for the permanent.

Determinant is polynomial time reducible using Gaussian elimination.

I.e. $Det \in FP$

In contrast.

Theorem 16.1. Computing the permanent of a $\{0,1\}$-matrix is #P-complete.

A word about problems:

- The decision problem: "does a perfect matching exist" is in P.
- The counting problem: "how many perfect matchings are there" is #P-complete.
- The decision problem: "given B and N does B have N perfect matchings" is in PSPACE and has a nice IP.

If *perm* is in FP then P = NP.

Theorem 16.2. (Toda) PH (polynomial heirarchy) $\subseteq P^{\#P}$

Corollary 16.3. If $perm \in FP$ then $PH = P$.

Theorem 16.4. (Lipton) Random self-reducibility of perm.

Suppose we have an oracle that computes *perm* correctly on a $(1 - \frac{1}{3n})$ fraction of all $(n \times n)$ matrices over a finite field F with $|F| > 3n$. Then we can compute *perm* in polytime (using oracle) on all $(n \times n)$-matrices by a probabilistic algorithmthat, for any input matrix, returns the correct result with probability $\geq \frac{2}{3} - \frac{1}{3n}$.

The modern version replaces $(1 - \frac{1}{3n})$ with any constant $c > \frac{1}{2}$.

Dokaz. Let A be the input matrix. Choose (uniformly) a random matrix $R \in F^{n \times n}$. Define $B(x) = A - x \cdot R$. Then:

- $Perm(B(x))$ is a degree n univariate polynomial
- For any $a \neq 0$, $B(a)$ is random, hence the oracle computes $perm(B(a))$ correctly with probability $\geq 1 - \frac{1}{3n}$.

  Choose $n+1$ distinct values $a_1, \ldots, a_{n+1}$. Use the oracle $(n+1)$ times on inputs $B(a_1), \ldots, B(a_{n+1})$ to get values $p_1, \ldots, p_{n+1}$. So with probability $\geq 1 - \frac{n+1}{3n} = \frac{2}{3} - \frac{1}{3n}$ all values of $p_i = perm(B_(a_i))$.

  Find (using interpolation) the unique degree n polynomial $q(x)$ that enjoys $q(a_i) = p_1, \ldots, q(a_{n+1}) = p_{n+1}$. Return $q(0)$, which $= perm(A)$ in the case that al $p_i = perm(B(a_i))$.

■

»*I find my daughter's mathematics really difficult.*
I wish they did some proofs.«
– prof. dr. Alexander Keith Simpson,
15. 1. 2024

If we have an algorithm M for $perm$

$$Avgtime(n) = E_{A \in F^{(n \times n)}}(\text{time taken by M on matrix A})$$

If $Avgtime(n)$ is $\mathcal{O}(p(n))$ (p a polynomial) then there exsists a ptime algorithm that correctly computes the permanent on a $\frac{2}{3}$ fraction of all inputs.

Then by (modern version of theorem) $perm \in P$.

Run the algorithm with polynomial $p(n)$ expected time for at most $4p(n)$ steps. And if it still hasn't stopped return 0.

The above algorithm is correct on $> \frac{2}{3}$ fraction of inputs.

Theorem 16.5. (Jerrum, Sinclair, Vigoda 2004) Approximating the permanent

There exists a probabilistic algorithm what given an $(n \times n)$-matrix $A \in \mathbb{N}^{n \times n}$ and ϵ, δ satisfies

$$Pr[\frac{|\text{result algorithm } - perm(A)|}{perm(A)} < \epsilon] \geq 1 - \delta$$

whose computation time is polynomial in n, $log(\delta^{-1})$, ϵ^{-1}.

Caveats:

- the result is for non-negative matrices
- polynomial in ϵ^{-1} is not really very good ($(log\epsilon^{-1})$ would be better)
- the hidden constants and order of the polynomial are large.