FIRST LECTURE RECAP

- Practicalities
- Discussion about computation and computational complexity
- Basic definitions
- Computational model: $\boxed{\text{TURING MACHINE}}$ (TM)

  Q set of states

  Σ finite-size alphabet

  Tapes Read-only input tape
  Read-write work tape
  (Add more tapes if convenient)

- TMs used to solve $\boxed{\text{DECISION PROBLEMS}}$

$$f : \Sigma^* \to \{0, 1\}$$

think of 0 = no
1 = yes

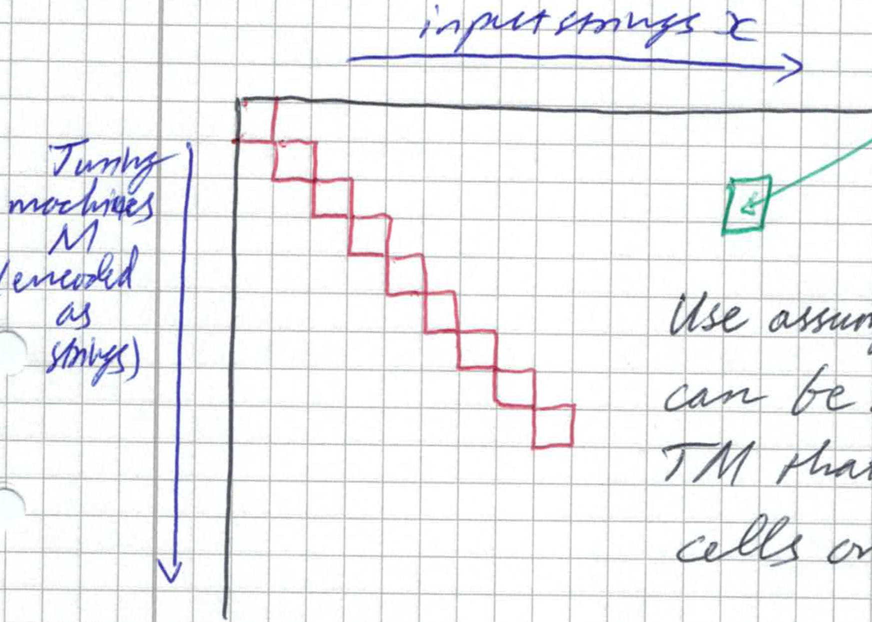
$$\boxed{\begin{array}{c}\text{Solve decision problem}\\ f : \Sigma^* \to \{0,1\}\end{array}} \iff \boxed{\begin{array}{c}\text{DECIDE LANGUAGE}\\ L = \{x \in \Sigma^* \mid f(x) = 1\}\end{array}}$$

There is a UNIVERSAL TURING MACHINE that can simulate any other TM efficiently given its description as a string

It is UNDECIDABLE whether a given Turing machine M halts on a given input x.

DETOUR: What happened in the proof of undecidability of the Halting problem?

DIAGONALIZATION



Use assumption that Halting problem can be decided to build TM that disagrees with all cells on diagonal of table

$\Downarrow$

Cannot be one of TMs listed in table, but we are listing all of them ↯

Diagonalization is an important proof technique in computational complexity theory — will see more examples

Even if a problem is computable, it can be infeasible in practice

FROM NOW ON: Focus on computable problems classify how hard they are

COMPLEXCITY CLASS set of functions that can be computed within some given resource

Resource we care most about: $\boxed{\text{TIME}}$

Let $T: \mathbb{N} \to \mathbb{N}$ be a function

Definition (DTIME)

A language L is in $\boxed{DTIME(T(n))}$ if there is a Turing machine M_L that

a) decides L

b) runs in time $O(T(n))$

Definition (P)

$$\boxed{P} = \bigcup_{k=1}^{\infty} DTIME(n^k)$$

$\boxed{P}$ = the set of languages that can be decided in POLYNOMIAL TIME, i.e. time $O(n^k)$ for some constant k (arbitrary but fixed

Note
- P defined for decision problems
- Running time measured in size of input

CHURCH - TURING THESIS
Every physically realizable computational device can be simulated by a Turing machine

Not a theorem — it couldn't be — but consistent with what we currently know about nature

STRONG/EXTENDED CHURCH-TURING THESIS
Anything efficiently computable on any computational device can be efficiently simulated by a Turing machine (i.e., with at most polynomial overhead)

Might not be true if quantum computers can be built
But we think of P as capturing what is efficiently computable

Is P a reasonable model of efficiently solvable problems?

Pros

- Composes well: efficient programs can call efficient subroutines and stay efficient
- Exponents of polynomials in running times are often small
- Reasonable agreement between theoretical definition and what we see in practice

Cons

- Worst-case complexity — have to have polynomial running time for all problem instances — is too strict! What if difficulty due to some pathological instance never seen in practice?!
  - Not quite clear what "in practice" should mean mathematically
  - There have been attempts at average-case complexity
- Polynomial time is too slow! The small exponents we observe are due to that this is the kind of algorithms we can discover and understand. And for huge data sets, quadratic or sometimes even linear time is impractical
  - There is research into such scenarios
  - But P is still a relevant class
- Focusing on decision problems is too limited a framework!
  - Yes, sometimes. But surprisingly often not! Definitely not for the problems we are discussing here.

Cons, continued...

- What about computation in other physical models that might ~~be~~
  - (a) be CONTINUOUS rather than discrete?
    - if so, we still need to measure, and to deal with noise
  - (b) use randomness (obtained, say, from radioactive decay)?
    - randomness can be useful in practice but does not seem to matter for our theoretical definition
  - (c) use effects from quantum mechanics?
    - yes, that might make a difference — not for computability but for efficiency

---

We want to study different computational problems and understand how hard they are

In particular, is a given problem / language in P or not?

Turn out to be challenging to decide for many problems that we care about.

INTERLUDE: Given language / problem L, would like to

a) Give algorithm A deciding L.

b) Prove that no algorithm can do (asymptotically) better than A

Many successes for (a)
(not least here in the AC Section at DIKU)

Task (b) seems much, much harder! Because how could you prove that no algorithm, however crazy, cannot possibly do better?

What to do?

1. RESTRICT COMPUTATIONAL MODEL, e.g., focus on what "non-crazy" algorithms do and prove that no such algorithm can do better

2. RELATE HOW HARD DIFFERENT PROBLEMS ARE compared to each other

   Can use REDUCTIONS to translate between different problems to understand how hardness is related

   - Shows nontrivial connections
   - Helps us to expand intuition about what to expect

REDUCTIONS

L_1 reduces to L_2, written $\boxed{L_1 \leq L_2}$ if exists efficient algorithm computing some function $g : \Sigma^* \rightarrow \Sigma^*$ such that

$$\boxed{\begin{array}{lll} x \in L_1 & \Rightarrow & g(x) \in L_2 \\ x \notin L_1 & \Rightarrow & g(x) \notin L_2 \end{array}}$$

Positive use case:

Have efficient algorithm A for L_2
Encounter a new problem L_1
If we can find a reduction from L_1 to L_2 then we can solve L_1 efficiently by computing $A(g(x))$ for input x

"Solving L_1 is at least as easy as solving L_2"

Ex
- Reduce bipartite matching to max flow
- Encode problem as propositional logic formula and solve formula

Negative use case:

Believe (or know) that L_1 is a hard problem
Encounter a new problem L_2
If we can find a reduction g from L_1 to L_2, then L_2 must be at least as hard as L_1

"Solving L_2 is at least as hard as solving L_1"

SOLVING A PROBLEM VS. VERIFYING SOLUTIONS

Doing an exam requires coming up with solutions — can be hard

Grading the exam just involves verifying correctness — much easier (hopefully)

Complexity class P

Class of efficiently solvable (decision) problems (i.e., in polynomial time)

Complexity class NP

Class of problems for which proposed solutions can be verified efficiently

Except we have decision problems, so what do we mean by a "solution"

Formally, some kind of auxiliary string (called CERTIFICATE or WITNESS) that helps to verify that yes-instances are yes-instances.

For us, this will often be the solution to the search problem that the decision problem came from

Examples of certificates:

1. s-t-PATH: An ordered list of vertices forming the path.
2. FACTORING: The prime factorization of N
3. SATISFIABILITY: A satisfying assignment
4. SUBSET SUM: A subset summing to the target T

DEFINITION Language L is in $\boxed{NP}$ if
- exist polynomial p
- exist polynomial-time Turing machine M (verifier) taking two arguments x, y

such that

$$x \in L \iff \boxed{\begin{array}{l}\text{Exists } y \in \Sigma^* \text{ of length} \\ |y| \leq p(|x|) \text{ such that} \\ M(x,y) = 1\end{array}}$$

Again, y is called a $\boxed{\text{CERTIFICATE}}$ or $\boxed{\text{WITNESS}}$ for x

(Arora-Barak wants witness of size EXACTLY $p(|x|)$ — does not matter)

Why is NP an interesting problem class? Because for most practical problems that we want to solve by constructing some object, it is possible to check if a proposed solution meets the requirements

Is it possible to solve all problems in NP efficiently? We don't know.

One of the big open $\boxed{\text{MILLENNIUM PRIZE PROBLEMS}}$ in modern mathematics

We will next take a closer look at NP and study the hardest and most interesting problems in this class — the $\boxed{NP\text{-complete problems}}$

QUICK RECAP

REDUCTIONS

(polynomial time in $|x|$)

$L_1 \leq_p L_2$ if exists efficiently computable g such that

sometimes called KARP REDUCTION

$$x \in L_1 \implies g(x) \in L_2$$
$$x \notin L_2 \implies g(x) \notin L_2$$

L_1 is not harder than L_2 — use to solve problem

L_2 is not easier than L_1 — use to prove hardness

NP

Class of problems / languages with "efficiently verifiable solutions"

$L \in NP$ if exist

- polynomial p
- polynomial-time Turing machine $M(x,y)$

such that

$$\boxed{x \in L \iff \text{Exist } y \text{ ~~off~~ of length} \leq p(|x|) \text{ such that } M(x,y) = 1}$$

$$\boxed{EXP = \bigcup_{k=1}^{\infty} DTIME\left(2^{n^k}\right)}$$

EXP

All languages L that can be decided in exponential time $O(2^{n^k})$ for some constant k (depending on L)

PROPOSITION 1 $P \subseteq NP \subseteq EXP$

Proof $P \subseteq NP$. Choose witness of length 0. Pick TM M that just solves the problem

$NP \subseteq EXP$ At most exponentially many witness candidates. Each can be checked in polynomial time

This simple proposition is state of the art (sadly)
One of the MILLENNIUM PRIZE PROBLEMS $P \overset{?}{=} NP$

Most researchers (but not all) believe $\boxed{P \neq NP}$

SOME EXAMPLE PROBLEMS AND WITNESSES

(1) Is there a $\boxed{\text{PATH}}$ from s to t in G?

(2) Is given positive integer N $\boxed{\text{COMPOSITE}}$? (i.e., not prime)

(3) Does integer N have $\boxed{\text{PRIME FACTOR}}$ $\leq u$?

(4) $\boxed{\text{LINEAR PROGRAMMING}}$

m linear inequalities $a_1 \cdot u_1 + \ldots + a_n \cdot u_n \leq b$, $a_i, b \in \mathbb{Q}$. Is there an assignment to the u_i satisfying all inequalities?

(5) $\boxed{\text{0-1 INTEGER LINEAR PROGRAMMING}}$

Same as (4), ~~by~~ but u_i have to be in $\{0,1\}$

(6) Is a given propositional logic formula $\boxed{\text{SATISFIABLE}}$?

$$(x_1 \lor x_2 \lor x_3) \land (\neg x_1 \lor \neg x_2) \land (\neg x_1 \lor \neg x_3) \land (\neg x_2 \lor \neg x_3)$$

(7) Is a given propositional logic formula a $\boxed{\text{TAUTOLOGY}}$?
(i.e., always true)

$$(\neg x_1 \land \neg x_2 \land \neg x_3) \lor (x_1 \land x_2) \lor (x_1 \land x_3) \lor (x_2 \land x_3)$$

(8) Given integers $S = \{A_1, \ldots, A_n\}$ and target T is it possible to construct a $\boxed{\text{SUBSET SUM}}$ $S' \subseteq S$ such that $\sum_{A_i \in S'} A_i = T$

$\{2, 3, 5, 7\}$
$T = 11$

(1) In NP Witness: vertices in path
In fact, in P Do, e.g., breadth first search

(2) In NP Witness: Factor f with $1 < f < N$
In fact, in P Efficient randomized algorithms known since [Miller '76] and [Rabin '80]
Poly-time algorithm (without randomness) in [AKS '04]

(3) In NP Witness: Prime factor
Not believed to be in P (RSA crypto would break)
But also not believed to be among hardest problems in NP

(4) In NP Witness: assignment
For long time, best LP algorithm had exponential worst-case complexity (Simplex) though very efficient in practice
Khachiyan '79: in P
Karmarkar '84: Practical algorithm

(5) In NP Witness: assignment
This problem is NP-complete — one of the hardest in NP

(6) In NP Witness: assignment
Also NP-complete

(7) ??? What would be a short witness?
Not believed to be in NP
But note that there are short counter-examples for non-members

(8) In NP Witness: subset S'
NP-complete What does this mean? Up next...

ORIGINAL DEFINITION OF NP

Uses nondeterminism (the "N" in NP)

NONDETERMINISTIC TURING MACHINE (NTM)

Each "line in the program" has two variants (the TM has two transition functions)
At each step, TM arbitrarily chooses one

One way of thinking about this is as flipping a "golden coin" that always comes up "the right way"

A nondeterministic Turing machine accepts x if <u>at least one</u> possible sequence of choices / coin flips leads to accept; otherwise, if all sequences fail, the TM rejects

A nondeterministic Turing machine runs in time T if <u>all</u> possible sequences of choices terminate within time T

<u>Definition (NTIME)</u>

L is in $NTIME(T(n))$ if there exists an NTM M_L that

a) runs in time $O(T(n))$

b) satisfies

$$x \in L \iff M_L \text{ accepts } x$$

(in the sense of the definition above)

LEMMA $\boxed{NP = \bigcup_{k=1}^{\infty} NTIME(n^k)}$

Proof Need to prove

(1) $L \in NP \Rightarrow L \in \bigcup_{k=1}^{\infty} NTIME(n^k)$

(2) $L \in \bigcup_{k=1}^{\infty} NTIME(n^k) \Rightarrow L \in NP$

(1) There is a TM $M(x,y)$ such that for $x \in L$ $\exists$ y of size $poly(|x|)$ such that $M(x,y) = 1$

Let M' be the NTM that

a) nondeterministically guesses y

b) then runs $M(x,y)$

$x \in L \Rightarrow \exists$ good witness $y \Rightarrow M'$ accepts x

$x \notin L \Rightarrow$ All witnesses fail $\Rightarrow M'$ rejects x

M' runs in poly time, since y poly size and M runs in poly time in size of input

(2) There is an NTM M' that accepts precisely $x \in L$.

Let the witness y be the sequence of coin flips.

Let M be the (deterministic) TM that given sequence of coin flips y simulates M' with these choices.

Wouldn't it be great to have a nondeterministic Turing machine on your desk? Not physically realizable (as far as we know). But useful theoretical model, e.g., for EXHAUSTIVE SEARCH