Lecture 1: Classical Complexity Review

The future is not laid out on a track, it is something that we can decide and to the extent that we do not violate any known laws of the universe we can probably make it work the way we want to.

1) Implications of Shor's Algorithm?

(i) Factoring $\in P$?

(ii) Quantum computers can solve "really hard" problems (NP-comp)

(iii) Maybe QC can solve "intermediate" problems.

↙

The problems that we know is not in P, but also don't know how to show them NP hard.

(make sure to look at complexity zoo)

One of these 3 is false:

i) Extended-Church Turing thesis is True

ii) Factoring $\notin$ P

iii) Large-scale universal QC cannot be built.

2) Notations

$\mathbb{N}$, $\mathbb{R}$, $\mathbb{Z}$ (integers), $\mathbb{C}$ (complex)

$\mathbb{Z}^+$, $\mathbb{R}^+$ non-negative

$\mathbb{N}$: natural; $\mathbb{R}$: Real

- n-bit strings $\{0,1\}^n$

- $|x|$ → if $x \in \mathbb{C}$, $|x| = \sqrt{x^* x}$

  → if $x \in \{0,1\}^n$, $|x| = n$ (length of string)

→ $[n] = \{1, \ldots, n\}$

→ $:=$ → definition

3) Classical Complexity Theory

3.1 Turing machine : 1936 Alan Turing

Def: A TM is a 7-tuple $(Q, \Sigma, \Gamma, \delta, q_0, q_{acc}, q_{rej})$

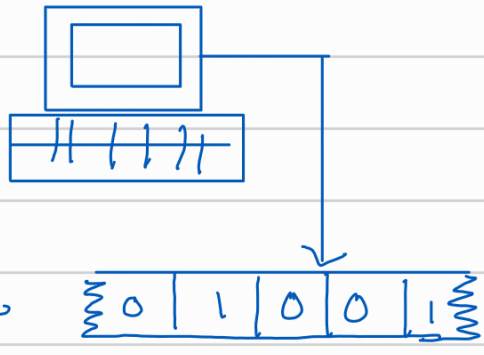
a) Q – a finite set of states a TM can be in

b) Σ – a finite set denoting the input alphabet

c) Γ – " " " the tape alphabet (includes $\sqcup \in \Gamma$)

d) $\delta : Q \times \Gamma \longrightarrow Q \times \Gamma \times \{L, R\}$

(transition function ; ($\Sigma \subseteq \Gamma$) ; (move the head right one cell)



this takes in the current state and Tape symbol and outputs the next state, tape symbol and move

e) $q_0 \in Q$ – start state of the TM

f) $q_{acc} \in Q$ ~ accept and halt state of TM

g) $q_{rej} \in Q$ ~ reject and halt state of TM

One step of computation?

i) Read current symbol ii) Update state of TM
iii) write symbol to current cell iv) move head L or R one cell.

Given input $x \in \{0,1\}^*$, TM need not halt.

In remainder of this note, we will work with Turing machine model of classical computing, one can equivantly work with the circuit model, which we will take a look at in next note.

The Turing machine model is very basic - it consists of a brain or control unit which makes the decision, an infinitely large memory stored on an infinitely long tape, and a head to read and write data from the tape.

Language and decision problems

Language → $L \subseteq \{0,1\}^*$

Decision Problem → Given $x \in \{0,1\}^*$, is $x \in L$?

A T.M decides L: $\forall$ input $x \in \{0,1\}^*$, TM halts and accepts/rejects.

(decides: says nothing about time complexity)

($x \in L$ / $x \notin L$)

recall $\exists$ undecidable languages L. There are 'yes' or 'no' problem that no T.M can solve in finite time.

3&4

EXERCISE: Given as input a description of a TM M and an input x, does M halt on input x? HALTING PROBLEM

Given input $\langle M \rangle$ - encoding and input x, does M halt on x?

There are problems that cannot be solved by any algorithm. The Halting Problem in a computability theory asks, given a computer program and an input, will the program terminate or will it run forever.

Eg.
```
x = input()
while x:
    pass
```
If the input is empty, the program will terminate, and if the input isn't empty, the program will loop forever.

Definition: The decision problem H, for Halting problem, is the set of all $\{\langle p, x\rangle \mid$ program p halts on $x\}$. A T.M. solves H if given any input $\langle p, x\rangle$, it terminates in q_{acc} if $\langle p, x\rangle \in H$ and q_{rej} otherwise. As long as it terminates, TM should accept it

Proof by contradiction

T.M A that decides H. Now a T.M B defined as follows:

it takes an input $\langle p\rangle$, runs A on input $\langle p, \langle p\rangle\rangle$ and halts iff A rejects. i.e

B takes a program p, runs the supposed TM that decides the halting problem on the input "program p, input $\langle p\rangle$" based on the outcome if it says "yes p on input $\langle p\rangle$ halts", then it loops forever, otherwise it says "no p on input $\langle p\rangle$" doesn't halt" and then it terminates.

The undecidability of the Halting Problem arises from the fact that there can be infinitely many possible inputs and program states, even with finite memory.

The halting problem is decidable for Linear Bounded automata or deterministic machines with finite memory. A machine with finite memory has a finite number of configurations, and thus any deterministic program on it must eventually halt or repeat a previous configuration.

Church - Turing Thesis

if ∃ mechanical process, for computing some function $f: \{0,1\}^* \to \{0,1\}$
there ∃ T.M computing f.

it's not a proven theorem! but proving or disproving this is confusing.

it says nothing about runtime/overhead of Turing machine.

↓

Extended Church - Turing Thesis

3.2) P and NP

Def: (Polynomial-time): A language $L \subseteq \{0,1\}^*$ is in P, if ∃ deterministic T.M and fixed polynomial $re_L: \mathbb{N} \to \mathbb{R}^+$, s.t for any input $x \in \{0,1\}^n$, M halts in at most $O(re_L(n))$ steps and:

i) (completeness/yes case) if $x \in L$, M accepts
ii) (soundness/no case) if $x \notin L$, M rejects

Eg. F.T using Brute force takes $O(n^2)$ (unusable)
but using FFT takes $O(n \log n)$ time (usable)

Exercise 7: Can the choice of re_L, depend on the input x?
In the definition, we've first fixed the polynomial and then the input. The answer is No.

Prob ∈ P: Integer multiplication: Given as input $x, y \in \mathbb{Z}$ and

threshold $t \in \mathbb{Z}$, is $xy \leq 0$

Reverse of this problem: Factorization: Given $z \in \mathbb{Z}^+$, threshold t. does z have non-trivial factors $\leq t$?

($\exists$ $1 < x \leq t$ and y st $x.y = z$)

not known to be in P (RSA / cryptosystem)

it is in NP, that means given the factors x, can try to divide z by x in poly-time.

Turing machine for Multiplication

B	1	1	1	m	1	1	B

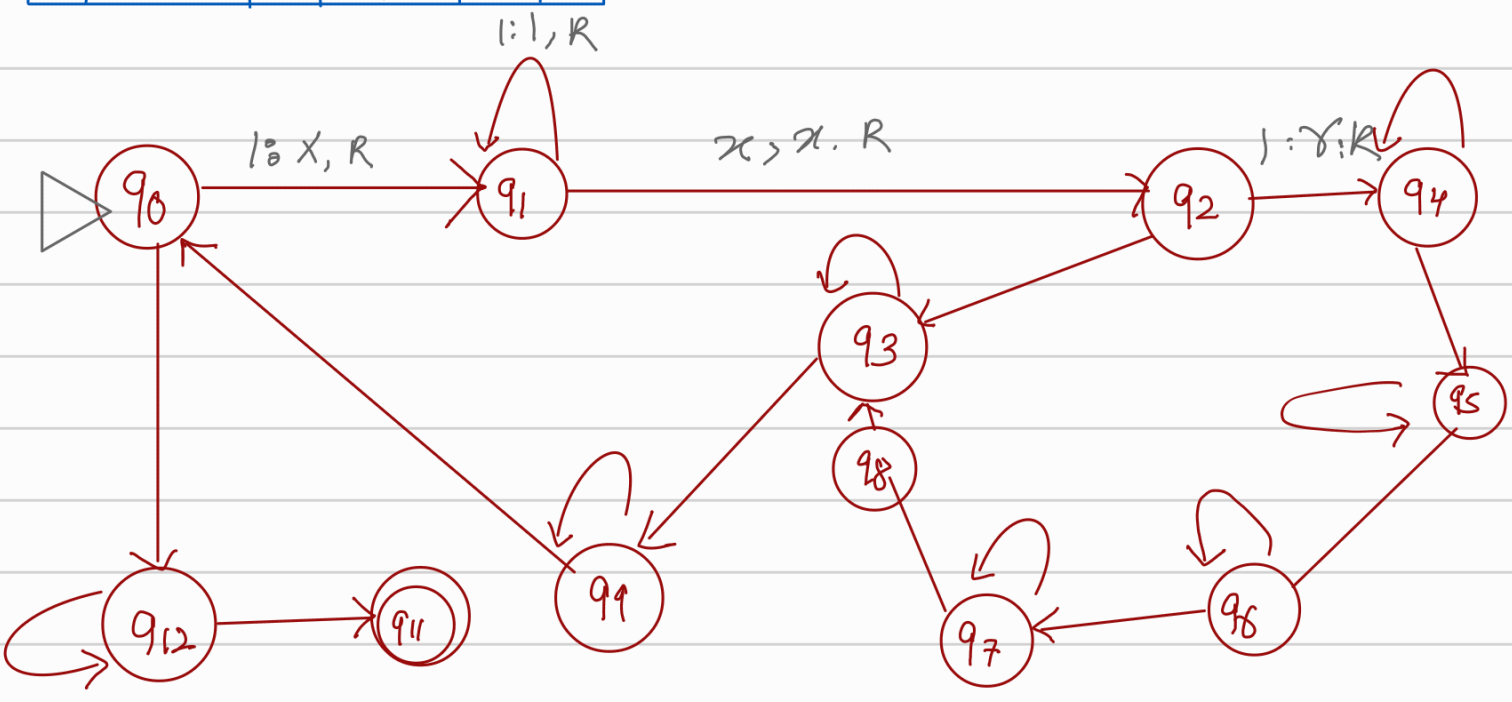
3 x 2 = 6

B is the blank

m is symbol to separate

After multiplication

B	1	1	1	1	1	1	B



Def 3: (Non-deterministic Polynomial time)

A language $L \in NP$ if $\exists$ TM, M, and fixed polynomials p_L, q_L s.t $\forall$ input $x \in \{0,1\}^n$, M takes in "proof" $y \in \{0,1\}^{p_L(n)}$ halts in $O(q_L(n))$ steps and:

$p_L(n)$

i) Completeness if $x \in L$, $\exists$ proof $y \in \{0,1\}^{p_1(n)}$ such that M accepts (x,y)

ii) soundness if $x \notin L$. $\forall$ proof $y \in \{0,1\}^{p_2(n)}$, M rejects (x,y)

FACTOR $\in$ NP : It can be verified efficiently. A language is in NP if there exists a polynomial-time algorithm to verify whether a given input belong to the language.

Here the polynomials depend on the size of the input.

The term "proof" refers to additional information or evidence that helps the deterministic Turing machine efficiently verify whether a given input 'x' belongs to Language L.

The Boolean satisfiability Problem

K-SAT: input : Boolean formula $\phi: \{0,1\}^n \mapsto \{0,1\}$ in (NP)

conjuctive normal form

eg: $\phi = (x_1 \vee \bar{x}_2 \vee x_4) \wedge (\bar{x}_2 \vee \bar{x}_5 \vee x_3) \wedge \cdots \wedge (x_7 \vee \cdot)$

literals — negation NOT — OR — AND

3-SAT problem

clause

output: $\exists\, x \in \{0,1\}^n$ s.t $\phi(x) = 1$? satisfiable

3.3 Cook-Levin theorem, NP completeness, reductions

Reductions; Intuition : Reduce problem A to problem B

A: don't know how to solve

B: know how to solve

(Helium Atom) $\longmapsto$ (Hydrogen Atom)

If the generality of P and NP is the Eiffel tower of complexity theory, then the Cook-Levin theorem is its Arc de Triomphe

Cook-Levin theorem showed that to solve every problem in NP efficiently it suffices to solve just a single problem - the Boolean satisfiability problem. Eg: 3-SAT, CLIQUE, KNAPSACK, are all the same problem as far as complexity theory is concerned.

many-one / Karp / Mapping reduction

Def: Reduction Let $A, B \subseteq \{0,1\}^*$. A reduction $A \leq B$ is a computable function $f : \{0,1\}^* \to \{0,1\}^*$ s.t $\forall$ input $x \in \{0,1\}^*$,

$$x \in A \quad \text{iff} \quad f(x) \in B$$

say reduction is polynomial-time if TM computing the reduction (f) is poly($|x|$) time.

so we write $A \leq_p B$

↳ Polynomial time reduction.

$$x \mapsto f(x) \quad \text{s.t} \quad x \in A \text{ iff } f(x) \in B$$

V/S

Turing Reductions: Imagine having an oracle O_B for deciding B. Mapping reduction: Computes $f(x)$, plugs $f(x)$ into O_B, output O_B's answer. immediately.

Turing reduction: O_B called poly many times, followed by adaptive post-processing after each call.

↳ Quantum Separability Problem: Given bipartite state ρ_{AB} (as classical matrix). is ρ_{AB} separable?

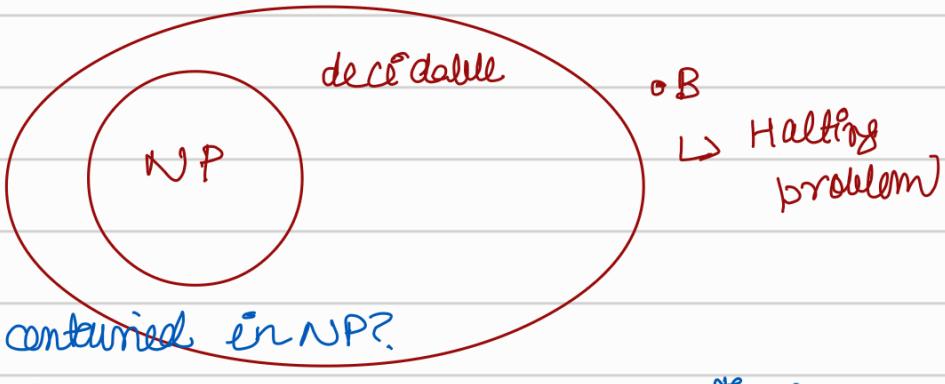
This is strongly NP-hard under Turing reduction

not known to be NP-hard under mapping reduction.

NP completeness — lower bound on difficulty of B

Def: NP-hard: A language $B \subseteq \{0,1\}^*$ is NP-hard, if $\forall A \in NP$ $\exists$ poly-time reduction to B, i.e $A \leq_p B$.

Think of it as problem with lower bounds. They are at least as hard as NP



Are you contained in NP?

Def: NP-completeness: Language $B \subseteq \{0,1\}^*$ is NP-complete, if B is NP-hard and $B \in NP$

The Cook-Levin theorem

→ SAT is NP-complete (Cook/Levin)

→ 3SAT is NP-complete (Karp's 21-NPC problem)

Theorem: SAT is NP-complete

↳ Lemma : 3-SAT

Proof : 3SAT ∈ NP (trivial)

We show $SAT \leq_p$ 3-SAT

Let $\phi : \{0,1\}^n \to \{0,1\}$ be an n-bit input to SAT, we construct a 3SAT formula $\phi' : \{0,1\}^n \longmapsto \{0,1\}$, s.t

ϕ is satisfiable iff ϕ' is satisfiable

Let 'c' be some arbitrary clause of ϕ of size $s \geq 4$, s.t

$$c = (l_{c,1} \vee l_{c,2} \vee \dots \vee l_{c,s})$$

We map 'c' to pair of new clauses, c_1 and c_2 of size $s-1$ & 3 respectively

Introduce new variable y_c and define

$$c_1 = (l_{c_1} \vee l_{c_2} \vee \dots \vee l_{x_{s-2}} \vee y_c) \quad \text{- size}(c_1) = s-1$$

$$c_2 = (l_{c_{s-1}} \vee l_{c_s} \vee \bar{y}_c) \quad \text{- size}(c_2) = 3$$

↳ polynomial time

↳ Prove the correctness:

claim $c_1 \wedge c_2$ is satisfiable iff c is satisfiable.

Prove 3SAT is NP-hard even if each variable appears at most 3 times.

3-4) The Extended Church Turing thesis

Due to shor's factoring algorithm, one of the following is false.

(i) Extended church Turing thesis is true

(ii) FACTOR $\notin$ P

(iii) Large scale (universal) QC can be built*.

Any "physically realizable" model of computing can be simulated by a TM with poly overhead.

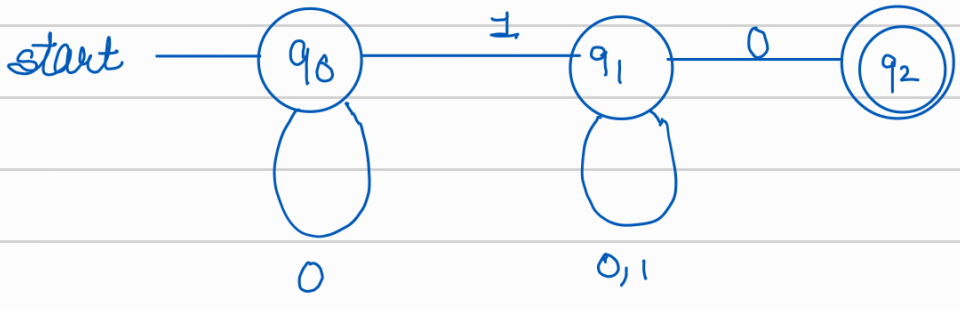
Best case: (2) and (3) $\Rightarrow$ (1) false

Worst case (1) and (2) $\Rightarrow$ (3) false.

EXERCISE 1: Sketch a TM which, given input strings $x \in \{0,1\}^n$, outputs 0 if x contains only zeros, and outputs 1 otherwise.

This TM operates in linear time with respect to the length of the input string. If no 1's are encountered, it halts after reading the entire input. Therefore the number of steps it takes is asymptotically $O(n)$

This algorithm always halts because it terminates as soon as it encounters the first 1 in the input string.



EXERCISE 2: Given as input a description of a TM, M and an input x, does M halt on input x? Can you prove that HALT is undecidable?

This involves a proof by contradiction, often using reduction from another known decidable problem.

1. Assume, for the sake of contradiction, that there exists a TM, H that can decide the Halting problem (HALT)
2. Now construct a TM, G as follows:
   i) On input $\langle M, x \rangle$, where M is description of TM and x is an input string:
      a) simulate M on input x
      b) If M halts on input x, G enters an ∞ loop
      c) If M does not halts on input x, G halts

3. Now consider what will happen when we feed $\langle G, \langle G \rangle \rangle$ to H:
   i) If 'H' returns "Yes", it implies G halts on input $\langle G \rangle$
      a) But this is a contradiction, because G halts iff G doesn't halt on $\langle G \rangle$
   ii) If 'H' returns "No" $\Rightarrow$ G does not Halts on input $\langle G \rangle$
      a) Again, this is a contradiction because G Halts iff G doesn't halt on $\langle G \rangle$.

Since, both outcomes lead to contradictions, the assumption that H exists must be false.

This proof demonstrates that there is no algorithmic procedure that can correctly determine whether an arbitrary TM halts on a given input, thereby establishing the undecidability of the Halting problem.