Lecture 20

- HW 3 Due tomorrow (Tues Nov 21, 11:59 am)

- Today : Wrapup on TMs and computability
  Start of Last Topic : Complexity Theory, P, NP

Computability Wrapup

1. TMs : general model of computation (Church-Turing Thesis)

   stronger versions : Multi-tape, Nondeterministic

2. Decidable (Recursive) Languages

   Examples: all Regular Languages, all CFL's

   Recognizable (R.e.) Languages

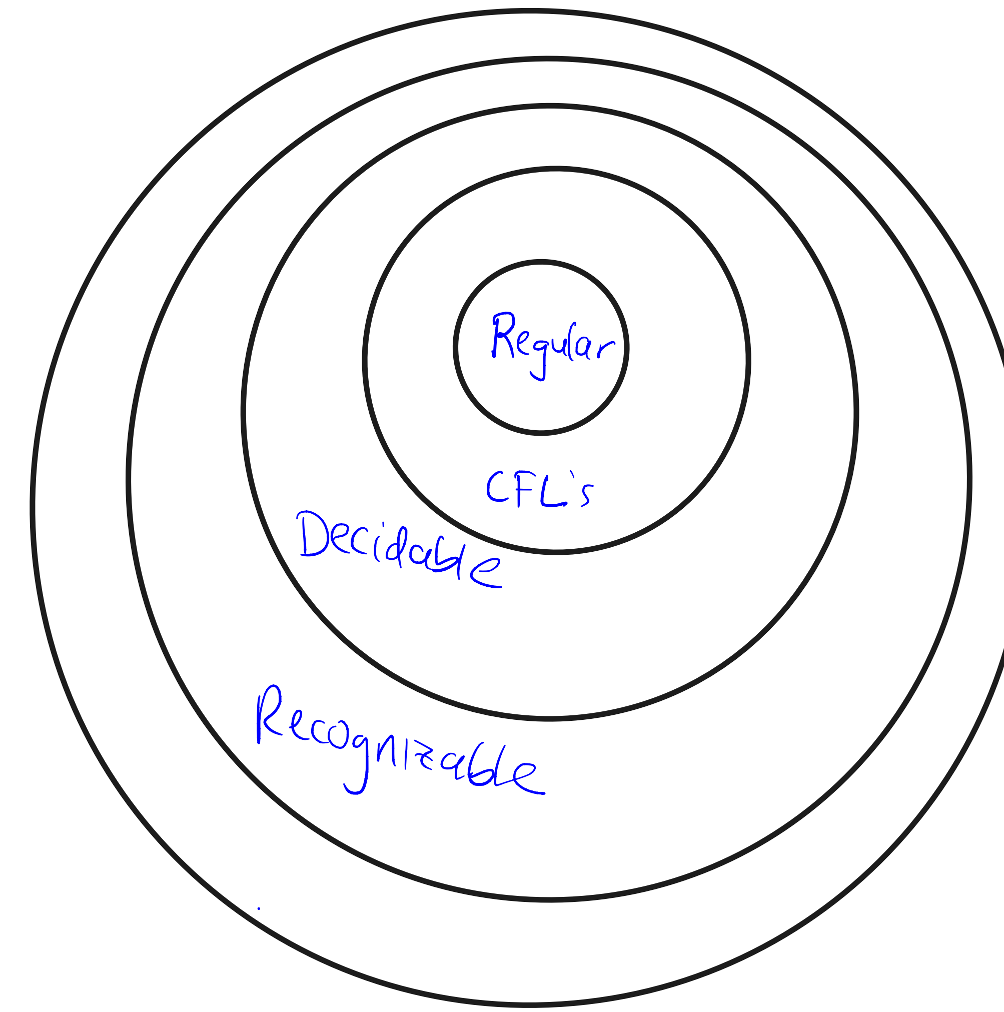
   Examples: All decidable languages HALT, A_{TM}

3. Closure properties of decidable/recognizable L's

4. Undecidable/unrecognizable Languages

   Method of Diagonalization (D not r.e.)

   Reductions



The Languages we showed are undecidable were all about properties of TMs. What about other more natural functions?

Here is a sample of some other (famous) undecidable problems:

1. Same questions (HALT, A_{TM}) are also undecidable in any other model of computation, e.g. Python programs, quantum computers, etc.

2. Hilbert's Tenth problem is undecidable (1900)

   Input: a diophantine equation (polynomial equation with integer coeff's)

   Example: $3x^2 - 2xy - z^3 + 5x^2y^2 = 0$

   Output: a soln over integers, or "unsolvable"

3. Undecidability of First order Logic (Hilbert's Entscheidungsproblem)

(4) Data compression

given a string $s \in \{0,1\}^*$, find shortest program that outputs s

(5) group theory

given a (finitely presented) group G

Is G finite?
Is G simple?
Is G commutative

} all undecidable

(6) Physics - Spectral gap (2015)

(difference between ground state + first excited state)

Sci. American Oct 2018)

Many subsequent undecidable problems in quantum physics

Complexity Theory

We saw that certain languages are undecidable --
even with unbounded resources (time, memory)
we cant solve these problems in the worst case

But even if a problem is decidable it may take
an enormous amount of time/memory, so it
still may not be solvable in practice

Complexity Theory : the study of important/central
problems and the amount of resources required
to solve them.

time, space, randomness, parallel computation,
quantum computer

Some Examples

① Matrix Multiplication: given 2 $n \times n$ matrices M_1, M_2
How much time (elementary plus/times operations) to output $M_1 \cdot M_2$?

② Prime: given a number x in binary, is it prime?

③ Factoring: given x in binary, output prime factorization

④ Clique: given a graph G on n vertices, and a number k,
does G contain a clique of size k?

⑤ Sudoku: input $n \times n$ puzzle, output a solution

Example: $n = 3$

$$M_1 = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} \quad M_2 = \begin{bmatrix} b_{11} & b_{12} & b_{13} \\ b_{21} & b_{22} & b_{23} \\ b_{31} & b_{32} & b_{33} \end{bmatrix}$$

n n

Entry (i,j) of $M_1 \times M_2$:

$$= \left. a_{i1} b_{1i} + a_{i2} b_{2i} + a_{i3} b_{3i} \right\} O(n) \text{ operations}$$

Runtime of this obvious alg $\approx \underbrace{n^2}_{n^2 \text{ entries}} \cdot n$

$O(n)$ operations each

Q. Is there an alg running in n^2 time?

Best known $\approx n^{2.2}$ time

Primality

Say $x = \underbrace{23}$ decimal

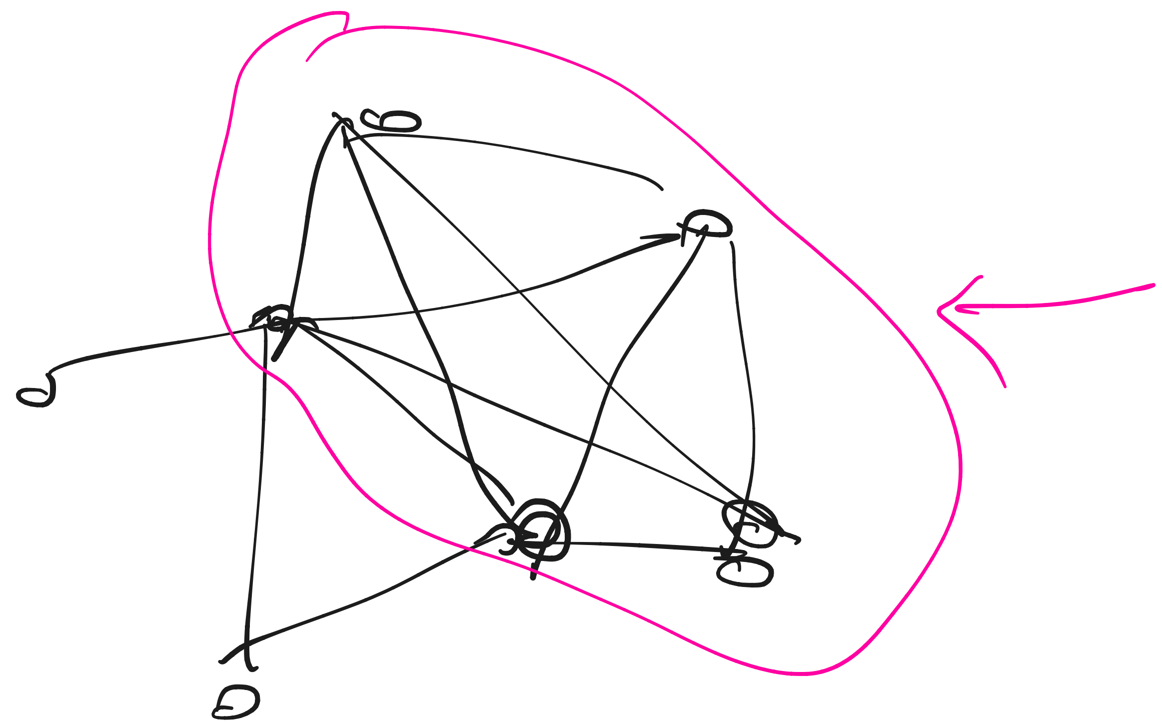
binary

There is a randomized alg. that runs very fast

Open for a long time: is there a fast deterministic alg (runtime n^2, n = length of x)

Yes!

$\mathcal{G} =$



Brute Force: Say $k=3$ then

$\mathcal{G}$ has a 3-clique $\equiv$ $\mathcal{G}$ containing a Δ

to solve: try all possible subsets $S \subseteq V$, $|S| = 3$

$$\binom{n}{3} \sim n^3$$

k-clique: $\binom{n}{k}$

Time Complexity

A step of a TM is a single transition of the TM on an input

The time complexity of a TM is a function (denoted $t(n)$ or $f(n)$) that measures the worst-case number of steps M takes (before halting) on any input of length n

Time complexity, Big-O notation

Big-O : ignore everything except the dominant growth term, including constant factors

Def'n For any 2 functions $f(n)$, $g(n)$
$f(n) = O(g(n))$ if $\exists c, n_0$ s.t. $\forall n \geq n_0$ $f(n) \leq c \cdot g(n)$.

Examples

1. $4n + 10 = O(n)$ ⟵ $f(n) = 4n+10$ $g(n) = n$
   $f(n) = O(g(n))$
2. $n^2 + 3n + 4 = O(n^2)$
3. $2^n + n^3 = O(2^n)$
4. $n^2 \log n + \log\log n = O(n^2 \log n)$

Time complexity, Big-O notation

Why do we care about asymptotic (Big-O) growth?

- We want to estimate the runtime of an algorithm.
  However differences in hardware / implementation can lead to differences in runtime.

  Example: register size, caching, etc.

- Analyzing Runtime can be cumbersome — big-O hides a lot of unnecessary details

Example of how Big-O Makes things Easier

M on input w

Scan across tape until we see a 0 or 1 — $O(n)$ steps

If none found → halt and accept — $O(1)$ steps

If one found, continue scanning until a matching 0 or 1 found — $O(n)$ steps

If none found reject — $O(1)$ steps

OW cross off that symbol and repeat — $O(n)$ steps

$O(n)$ loops

So total worst case runtime on w, $|w| = n$ is

$$\big(O(n) + O(1) + O(n) + O(1) + O(n)\big) \cdot O(n) = O(n) \cdot O(n) = O(n^2)$$

The Famous Class "P"

Defn Let $t : \mathbb{N} \to \mathbb{N}$ ($t \approx$ runtime)

$\mathrm{TIME}(t(n)) = \{ L \mid L$ is a language decided by a $O(t(n))$-time TM $\}$

Defn $t : \mathbb{N} \to \mathbb{N}$ is polynomial if $t(n) = O(n^k)$ for some $k \geq 0$

Ex $t(n) = n^2$, $t(n) = n$, $t(n) = n \log n$ are polynomial
($t(n) = n^{\log n}$ or $t(n) = 2^n$ are not polynomial.)

Defn $P = \{ L \mid L$ is a language decided by a TM running in polynomial time $\}$

* We think of problems in P as those that have relatively efficient algorithms.

The Famous Class "P" : Discussion / Motivation

Q1: Why polynomial time? Why not linear time or quadratic time?

Valid point.
If some program runs in n^{1000} time that certainly isn't feasibly solvable

If some problem $\notin P$ then we can say for sure that it is infeasible to solve in the worst case

Typical polytime algs actually run in time n or n^2 or n^3
so placing a problem in P usually means it is hopeful that it can be solved fairly efficiently

Still, it is important after placing a problem in P, to find a truly fast (ie. $O(n)$ or $O(n \log n)$ time) algorithm

The Famous Class "P" : Discussion / Motivation

Q2: TMs are so slow. why don't we define "P" for a better model of computation

Also good point. Really we want to consider a more realistic model like multitape TMs, or random-access machines

But the simulation of these by ordinary TMs is polynomial time, so if some problem has a polytime alg in some other model it will usually also have a polytime TM algorithm

one big exception: quantum computers

The Famous Class "P" : Discussion / Motivation

Q3 : Why worst-case runtime ?

Another good point.
Just because a problem is hard on some inputs, this isn't the whole story.

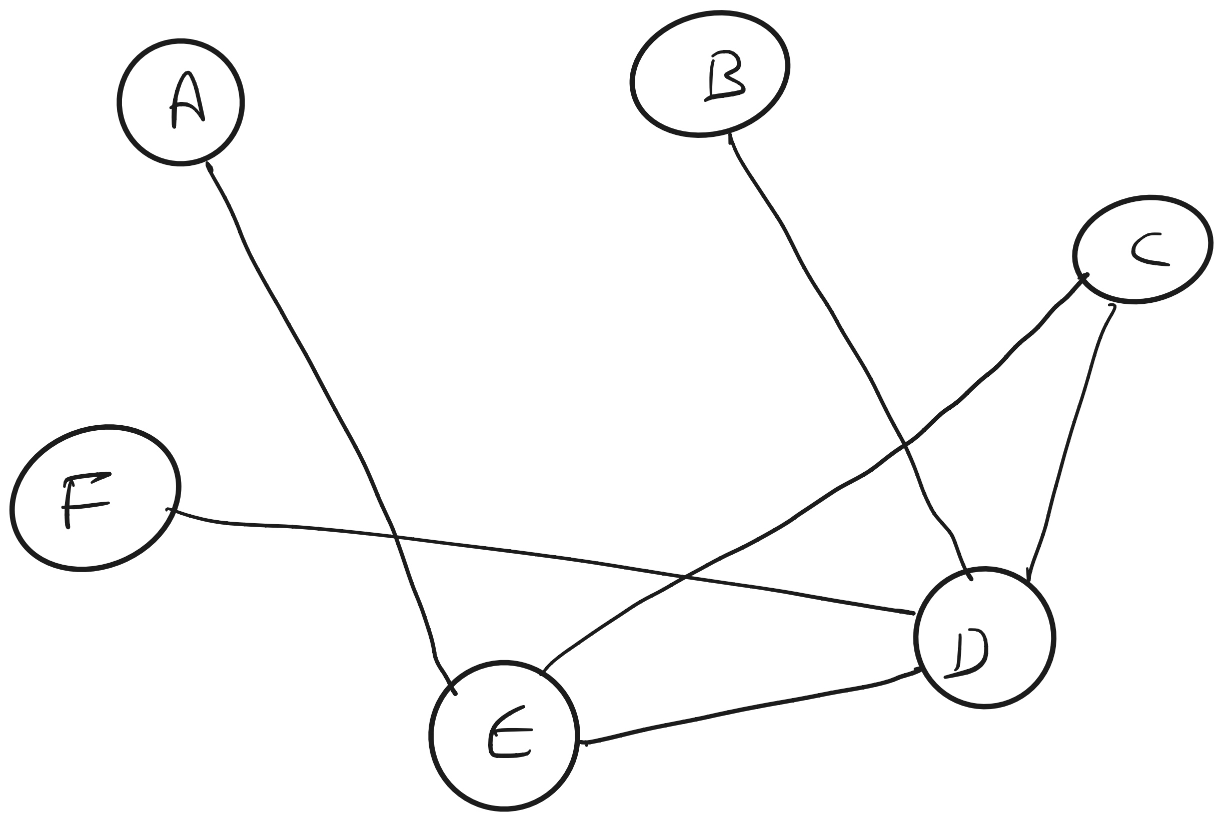
- It may be very easy to solve on 'typical' inputs
  Example: whole field of machine learning, chatgpt

- It may be easy to get a very good approximation in polytime even if solving optimally is not in P

Again, understanding worst case complexity is a starting point
ideal: solve exactly in polytime. If not possible, see if efficient on avg, or easy to approximate.

Some Problems in P

(i) S-t connectivity : given graph G, find length of shortest path from A to F



Naive soln : try all possible paths ~ $n!$ paths
runtime ~ $n! > 2^n$

Better soln : $O(n+m)$ $n = \#$ vertices
$m = \#$ edges

Some Problems in P

(2) Primes : given x in binary, is x prime?

Naive alg : try to divide x by $2, 3, 4, \dots, x-1$

Runtime : $\sim x$ steps

= exponential in $|x|$

Highly Nontrivial alg : Primes $\in P$

Some Problems in P

③ All Regular, CFL's are in P

④ graph connectivity:

given G, is there a path between <u>every</u> pair of vertices in G?

⑤ Linear Programming:

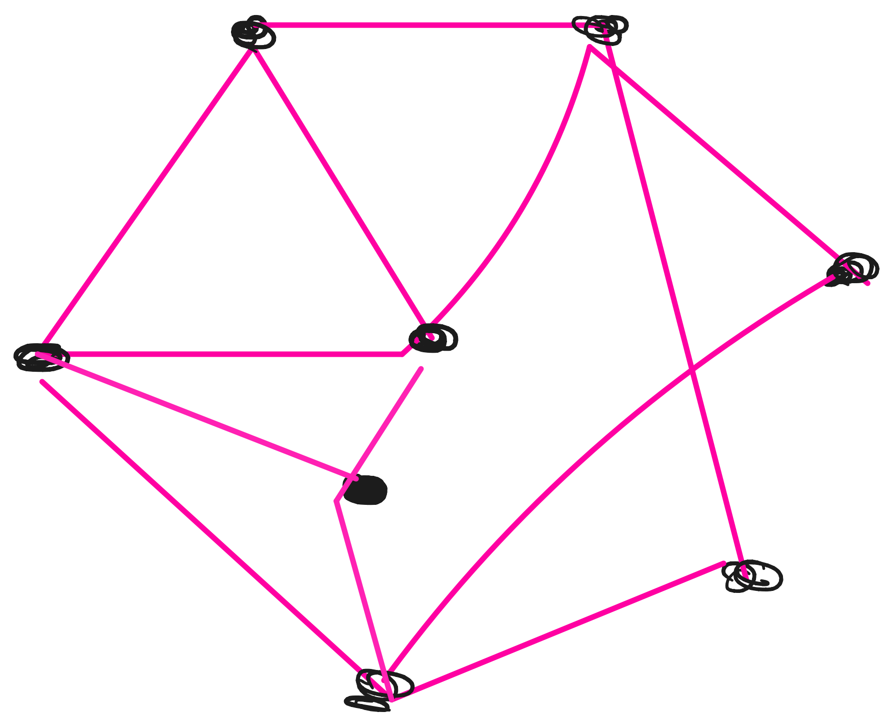
given linear set of constraints and linear objective function, Find optimal solution (over $\mathbb{R}$)

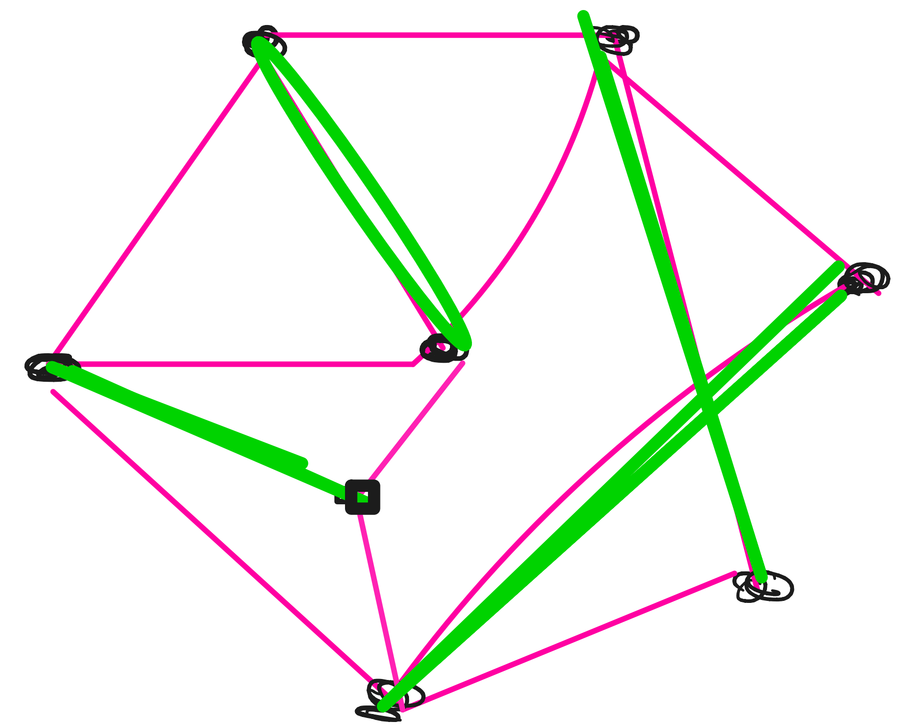
⑥ perfect matching: given G,

does $\exists$ a perfect matching in G?

Perfect matching cont'd

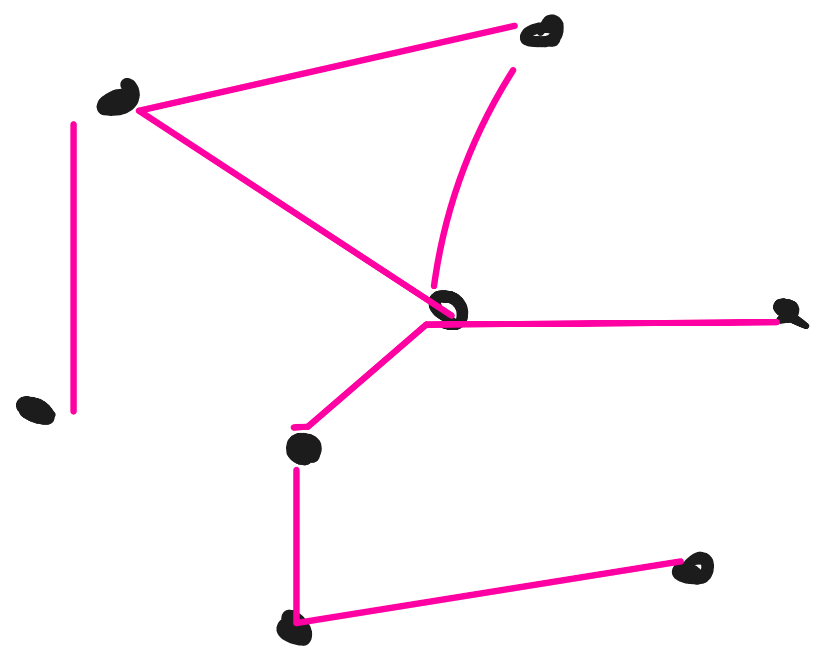
$\mathcal{G}$:





So $\mathcal{G}$ has a PM

Perfect matching cont'd



this G has no PM.