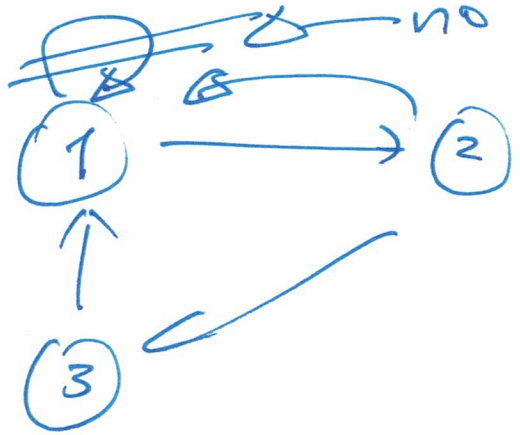
Cpts 515. 11/9/2020

Laplacian. a technique to translate a graph to a "nice" matrix.

Let G be a directed graph.

Exmple:



$$A = \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 1 \\ 1 & 0 & 0 \end{bmatrix}$$

$$D = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

this "2" means: out degree of node 2 is "2".

$$L = D - A = \begin{bmatrix} 1 & -1 & 0 \\ -1 & 2 & -1 \\ -1 & 0 & 1 \end{bmatrix}$$

Good properties:

all eigenvalues of L is ≥ 0. (all reals.) These n (# of nodes) eigenvalues form the spectrum of G.

We can use the spectrum to test whether two graphs are isomorphic.

Random walk on a graph. ← We have many approaches to do the same, with different results.

Applications: we want to create a "random" sample from some data. Our approach: ① create an "model" of the data

② the model is a graph.

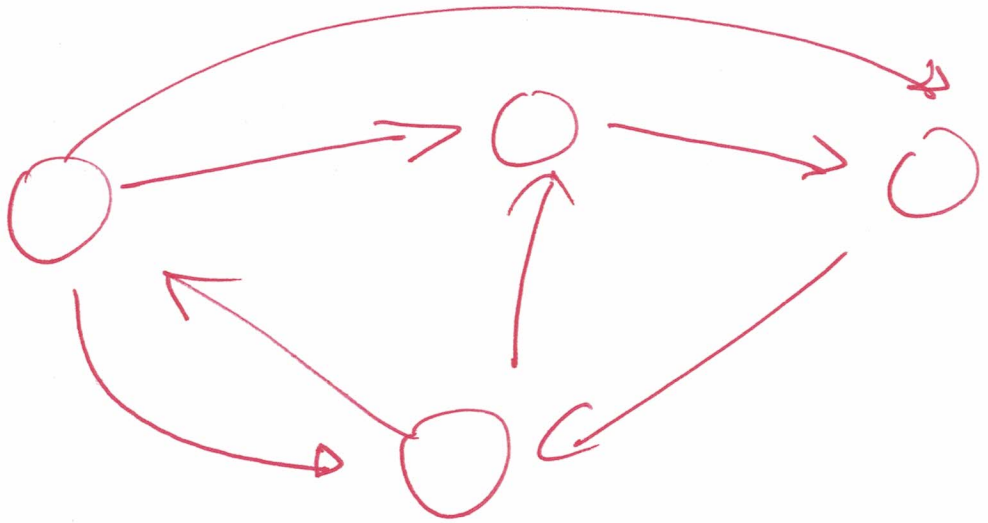
③. random walks on the model.

↳ sample.

Exaple. (most common way) — software design like UML.

Given a graph (model of data). ← This graph doesn't have any probability !!!

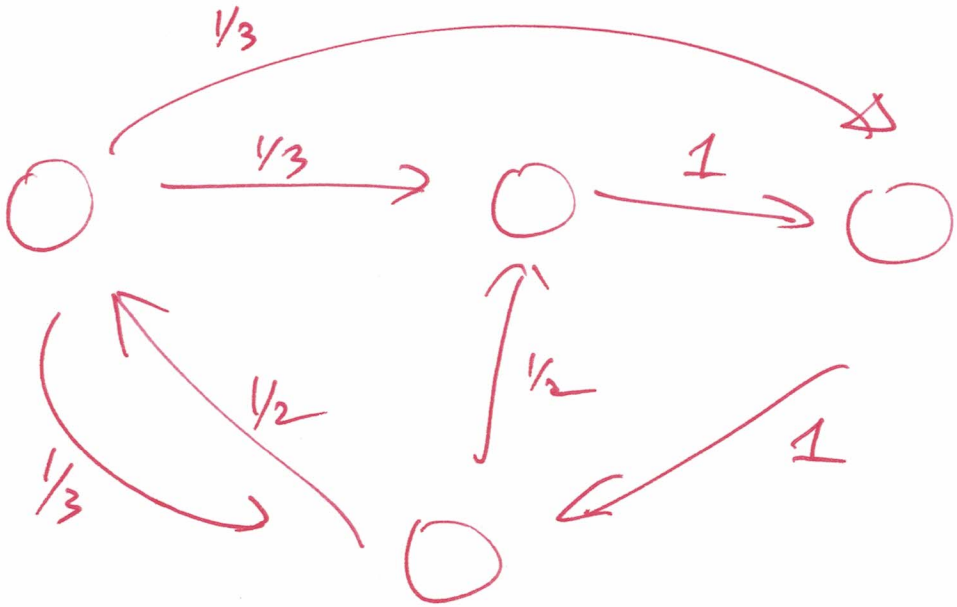
(Note: recall Bayesian Netwk ← trouble: who is going to give you the probabilty assgnmets ?)



Assume that G is connected.

⇒ How to assign "transition" probabilities on edges ?

$$P_{ij} = \frac{1}{\text{outdegree}(i)}$$



Remark: ① It's stationary dist is unique and can be efficiently computed.

② It's arguable that a sequence of nodes (it's a walk) genated from this graph or marker chain is "random".

③. "local" randomness but NOT global randomness.

④. Do you have other ways to assign these probabilities?

yes. roughly:

$G \longrightarrow$ adjacent matrix A

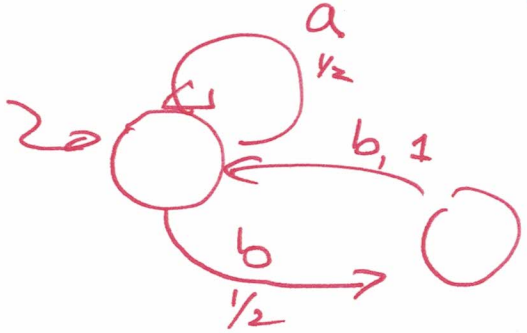
$(\cdots\rightarrow$ transform on this matrix. (e.g. e^A).

$\longrightarrow$ Perron number's left & right eigenvector.

$\longrightarrow$ multiply these two eigenvectors

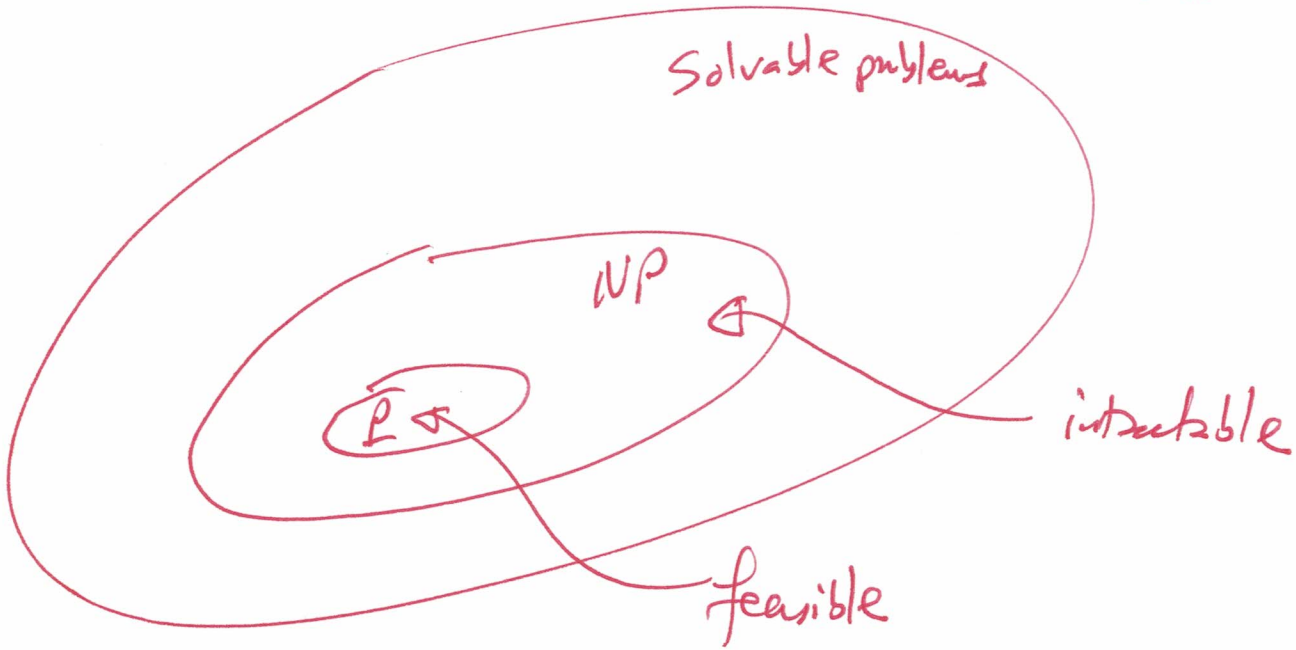
$\longrightarrow \{P_{ij}\}$. (true random walk).

Example: $\mathcal{L} = (a+bb)^*$. Alg to generate a random string in $\mathcal{L}$.



won't give you "random" string in $\mathcal{L}$.

unsolvable problems.



Algs can only solve a small portion of problems, within reasonable resources. (time and/or space).

P, NP, #P. etc.

Recall: problem = a set of positive instances

= a set of strings

= a language.

Example problem: Given: a graph G

Question: Is G connected?

$\{\langle G \rangle : G \text{ is connected}\}$.

string encoding of G.

Complexity class: a class of problems that can be solved by algorithms with a given complexity.

We focus on "time" complexity.

poly-time: $O(n^3)$, $O(n^4)$, ...

exp-time: $O(2^n)$.

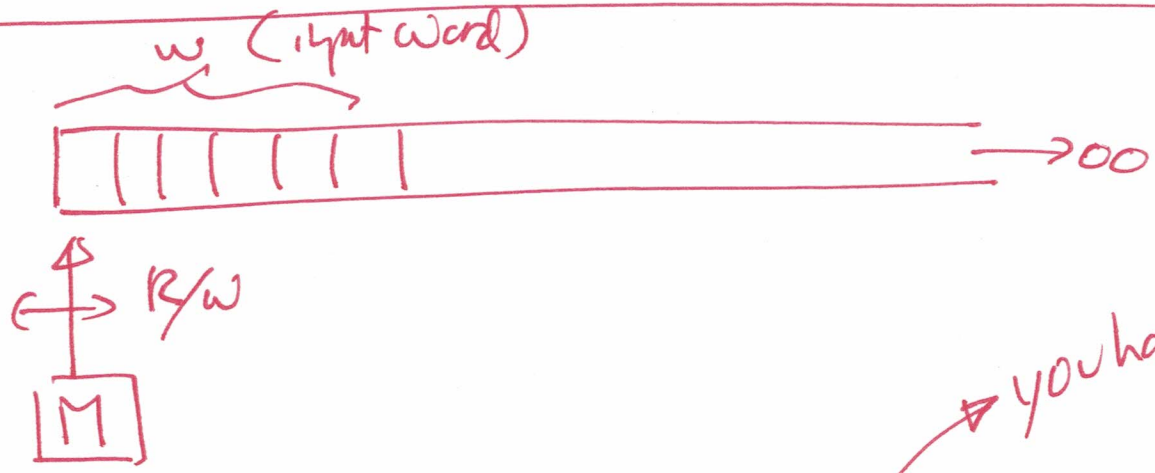
$$\lim_{n \to \infty} \frac{2^n}{n^4} = \infty .$$

"feasible", "tractable" ——— a problem that has poly-time alg to solve;

"infeasible", "intractable" ——— "inherently" difficult.

NP-complete.

Recall: { det. Alg
nondet. Alg } ⟶ "Algorithms" we usually see.



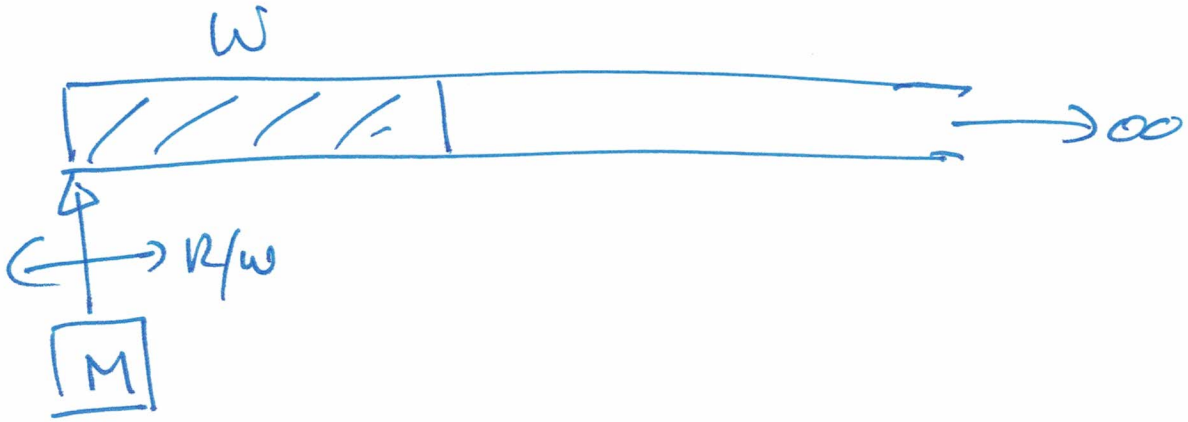
M is a (det) alg if M is a det. TM and M halts on all input.

→ you have at most one next move.

P = the class of problems, each of which can be solved by a det. Alg running in poly-time.

roughly means "efficient"

P is a small subset of all solvable problems.



M is a nondet. TM.

M has nondet. time complexity $T(n)$ if, for any input w of size n, M accepts w, then M accepts w on a run whose length is at most $T(n)$.

In this case, we call M a nondet. Alg with running time $T(n)$.

NP: The class of problems, each of which can be solved by a nondet. Alg whose running time is polynomial.

$\Updownarrow$

be accepted by a NTM with nondet. poly-time.

NP is a small subset of nondet. Algs.

↳ problems solvable by

So far, we have two classes of problems.

P and NP.

Remark: ① they are classes of problems; NOT classes of algs. You can NOT say my alg is in P, you can NOT say my alg is NP.

② No.1 open problem in CS:

$$P == NP ?$$

(We don't know whether a problem solvable in nondet. poly-time can also be solvable in det. poly-time.)

③ This is why in the entire history of CS, we study "nondet." Throughout. (Recall midterm).

④. In Quantum computing world, $P = NP$.