P & NP Problems

"Hard Problems" → a problem that takes exponential time to compute → (runtime) where it is not practical to obtain a solution for a problem of modest size n.

Tower OF Hanoi

$M_1 = 1$

$M_n = 2M_{n-1} + 1$

soln : $M_n = 2^n - 1$ ← real cost too high!!!

↑

why is it so lengthy?

decision problem : algo prob that has 2 possible answers

optimization problem : finding optimal soln from all feasible solns

every decision problem has a corr. optimization problem

if we provide evidence that the optimization problem is hard ⇒ optimization problem is also tough.

easy to solve opt prob ⇒ easy to solve decision

NOT HARD - P

Polynomially bounded algo : worst case complexity is bounded by poly function of input size.

poly bound prob ⇒ poly bound algo

P problems = class of decision problems that are polynomially bounded.

Dijk ($O(|V|^2)$) : worst case

eg. 1. Can we travel from city A to city B in k hours?

2. possible to supple electricity to all homes (n) using a powerline of k kilometers?

MST ($O(N^2)$)

only if there's a cycle in the graph → fully connected → A . . . . . . A permutate. →

3. Can we start from A, visit every other city once, and then return to A in k hours?

⇓ opt prob.

travelling salesman problem

consider all tours, find one that costs no more than k. for n cities → thats n! cases

not polynomial

a. correct tour?
b. total length?

for 1 soln takes $O(n^2)$ time

give a soln & check → P problem

0/1 Knapsack problem

is there a subset of objects that fits in the knapsack and returns a profit of at least K?

no known poly time : no. of subsets $= 2^n$

worst case check all : $O(2^n)$

given a subset $\rightarrow O(n)$ to check

dp takes $O(nC)$ time : psuedo-polynomial

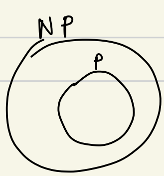
(size & value of input)

—

nondeterministic algorithm : solves problem in 2 phases : ① guess & ② verify + output step

NP : nondeterministic polynomially bounded :
a class of decision problems for which there is a polynomially bounded nondeterministic algo
or
a class of decision probs whose answers can be verified in polynomial time algorithm

$P \subseteq NP$



NP eg : travelling salesman
knapsack

· non NP ($\Rightarrow$ not P) : variant of travelling salesman :

does every tour have cost less than k

$n!$

each verification : $O(n)$

$\therefore$ total verification : $O(n \cdot n!)$

basically if the 'guess' is 'all solutions' then it won't work.

Subclass of NP
NP - Completeness
Problem Reduction

- How do we know a problem is hard ← takes $T >$ poly time
- How to <u>show</u> a problem is hard

① Reduce a known hard problem to given problem

P_1 (known) $\xrightarrow[\text{map to}]{}$ P_2

- if P_2 has a solution, reduction gives a way to solve P_1
- $\Rightarrow$ P_1 is as hard as P_2

eg. FactorAll(n) $\rightsquigarrow$ Factor1(n)
- use Factor1 to get 1 factor 'm' of n. divide n/m
- call factorAll(n/m) ⟶ keep repeating

Hamiltonian path problem: given graph G. is there any path that passes through all the vertices of the graph exactly once

travelling salesmen problem ← TSP graph

copy graph : edge: $w=1$; no edge: add, $w=2$

transformed problem: is there a path through the graph to all nodes wrt weight $\leq (n+1)$ ← travelling salesman problem

takes polynomial time (traverse edges: max $: V(V-1)$)

NP - Completeness

Prob D : $D \in NP$, every prob Q in NP is reducible to D in polynomial time.
then D is NP- complete.

$D \equiv P_1$

eg. Circuit Satisfyability problem : CIRCUIT-SAT

given a circuit, is there a set of inputs : output is true

n inputs : 2 options each : 2^n possibilities

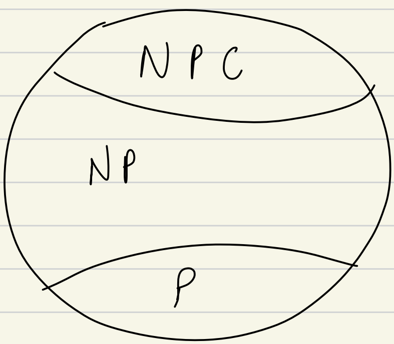
guess + verify can happen in polynomial time

· SAT : satisfiable boolean formula
given a bool formula, ... same thing

Cook + Levin proved that circuit-SAT is NP-complete
$\Rightarrow$ every NP prob can be reduced to circuit-SAT
$\because$ CIRCUIT-SAT can be reduced to SAT.
SAT is in NP
$\Rightarrow$ SAT is NP - Complete.

redusibility is transitive.

- everything in NP can reduce to NP-Complete
- NP-complete are the "hardest" soln to them is soln to all
- once soln to NP-Complete is found, all probs in NP can be solved in polynomial time
- if D is NPC, Q is NP, $D \xrightarrow{\text{red.}} Q$
  $\Rightarrow$ Q is NPC



Solving NP

- small problem size
- solve special instance
- heuristic algorithms (greedy heuristic)

locally optimal solns

picks least current cost.

1. fast
2. not guaranteed to give best soln
3. will give close to optimal solution in most cases
4. can give horrible soln

Greedy Heuristic for TSP : ① nearest neighbour

heuristic greedy

in greedy, if the last edge is very heavy, it will fuck up.

wont happen in practice

② shortest link

loop till we have n-1 edges.

pick from edges → least weight.
pick next cheapest edge that doesn't create a cycle and the edge is not the third edge incident on the vertice

→ ensures we visit vertice only once.

then link the 2 end points

greedy heuristic for knapscack

by DP : psuedo polyno midl.

sort obj acc to profit/weight (descending)

for 1 → n

    if (weight + w[i] ≤ c)

        value += v[i]

        weight += W[i]

time to calc P/S → O(n)

sort → O(nlogn) ← TC.

choose → O(n)

3 a) $\log_b a = 1$

$f(n) = n-1 = O(n) = O(n^{\log_b a})$

$\theta(n \log n) = W(n)$