* NP hard & NP Complete

Satisfiability problem

Reduced NP hard

$O(1)$

polynomial time

Exponential time

poly. time

linear Search → $O(n)$

binary Search → $O(\log n)$

Insertion Sort → $O(n^2)$

Merge Sort → $O(n \log n)$

Matrix Multiplicat^n → $O(n^3)$

0/1 Knapsack → 2^n

Traveling S P → 2^n

Sum of subset → 2^n

Graph Coloring → 2^n

Hamiltonian cycle → 2^n

Comparatively faster in execution

Research study is going on to find polynomial time solution to these problem

* Or atleast can we show that all such problems are related

→ ie

can we design polynomia

deterministic algo for suc

problems

magical algo

polynomial time Soln but using non deterministic algo

NP

exponential time algo

NP- hard

polynomial time using deterministic algo

(1) P ⊆ NP

NP- complete problems (satisfiability)

② $P = NP$ ✓ (polynomial time using deterministic algo)

③ $P \neq NP$ ✓

---

* Satisfiability problem (common base problem)

- for e.g 3 variables

$x_i = (x_1, x_2, x_3) \longrightarrow$ 3 var.

- draft calculus formula (CNF)

$$x_i = (x_1 \vee \bar{x}_2 \vee x_3) \wedge (\bar{x}_1 \vee x_2 \vee \bar{x}_3)$$

✓

soln $\Rightarrow 2^3 = 8$ options

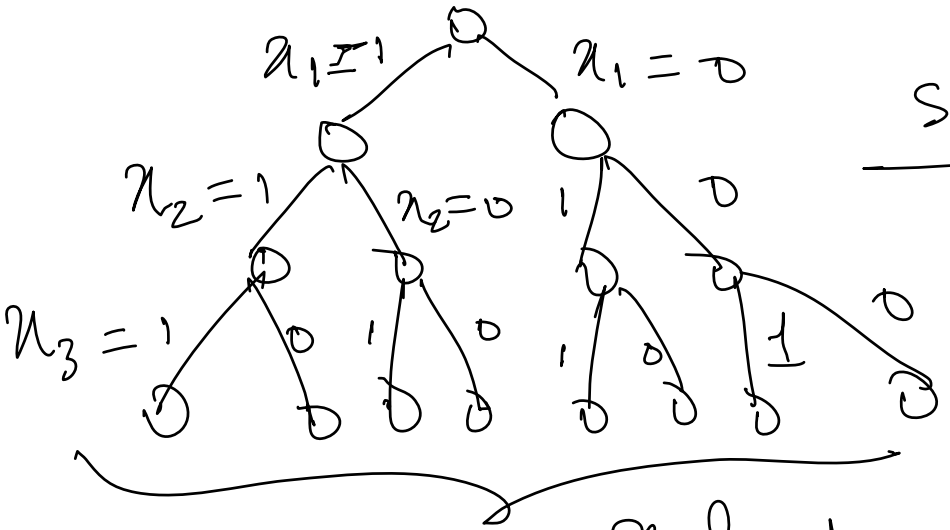
$2^3 \approx 2^n$ → exponential time

(x_1, x_2, x_3)

✓ 1 1 0

✓ 0 0 0

⇓ SOM can be rep as state space tree



Same as 0/1 Knapsack problem

8 leaf nodes ≈ 8 options

- if we can solve this state space tree in polynomial time then all the exponential time algo can also be solved in polynomial time

Reduction ($\propto$)

(relate expo-time algo to satisfiability problem)

Statisfiability problem $\propto$ exponential time algo (0/1 Knapsack)

$I_1 \propto I_2$

* Reduction problem (Transitive nature) → polynomial time

$L_1 \propto L_2$, $L_2 \propto L_3$

⇓

$L_1 \propto L_3$

* Cook's Theorem

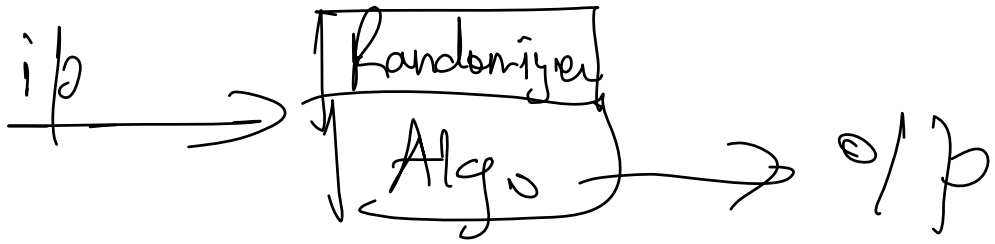
- If Satisfiability is in P iff $P = NP$

↓

We have well known deterministic polynomial time

* Randomized Algo

(uses randomizer / random number generator)



2 types of R.A

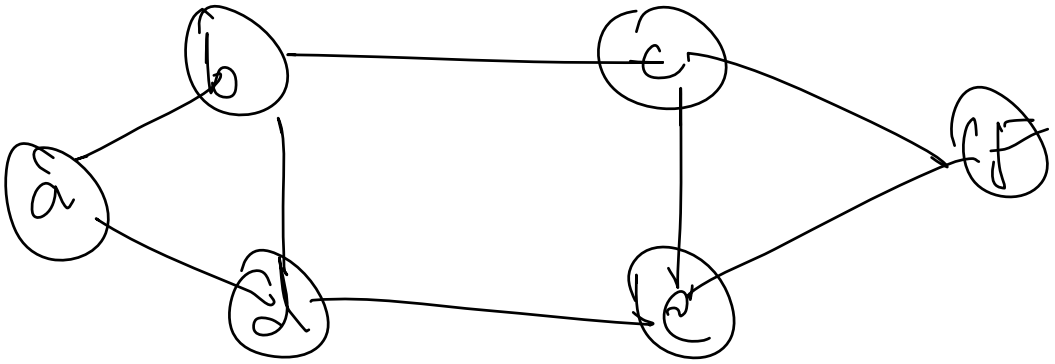
Las Vegas	Monte Carlo Algo
- o/p is always correct - efficient - Ex: Randomized Quick Sort	- o/p may be incorrect - not so efficient Ex: Randomized median finding problem

Ex: of R-A to find minimum cut in a graph

(Kanger's Min cut Algorithm)

Random_min_cut(G)

1. Repeat step 2 to 4 until only 2 vertices are left in G
2. pick an edge $\langle u, v \rangle$ at random
3. Merge u & v
4. Remove self loops from E
5. Return |E|



Achieve just 2 vertices

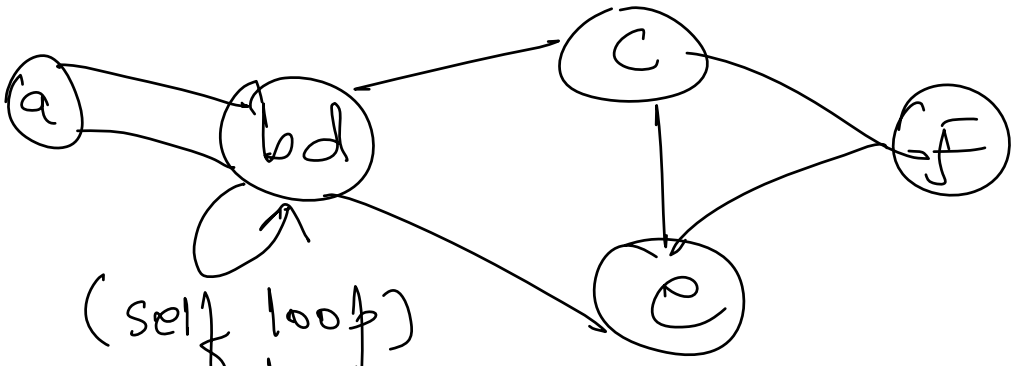
$V = 6$

$V = \{ \quad \}$

$E = \{ \langle a, b \rangle, \langle b, d \rangle \ldots \quad \ldots \}$

Step 2 I take b – d

Step 3

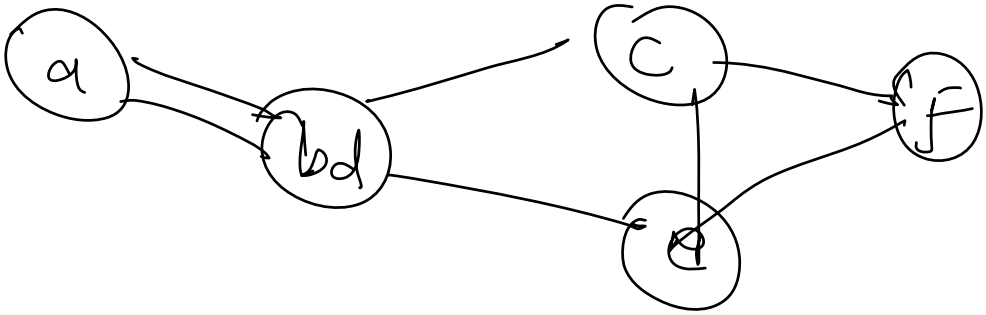


(self loop)

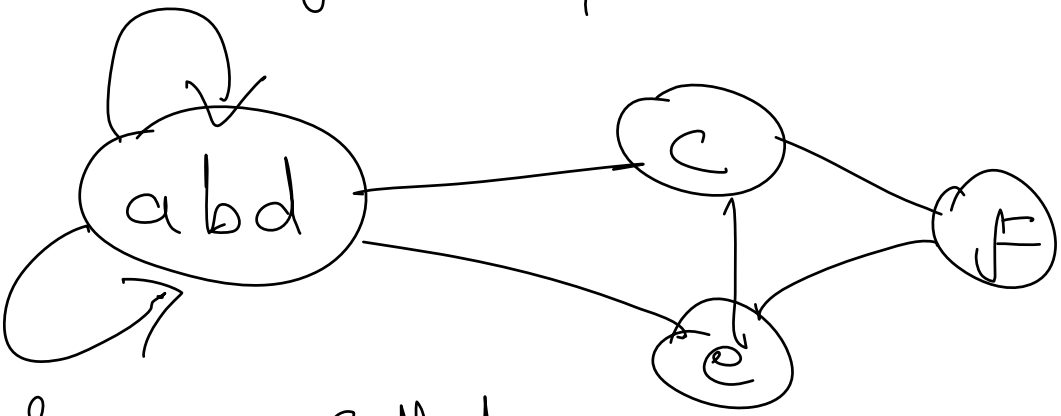
↓

remove it

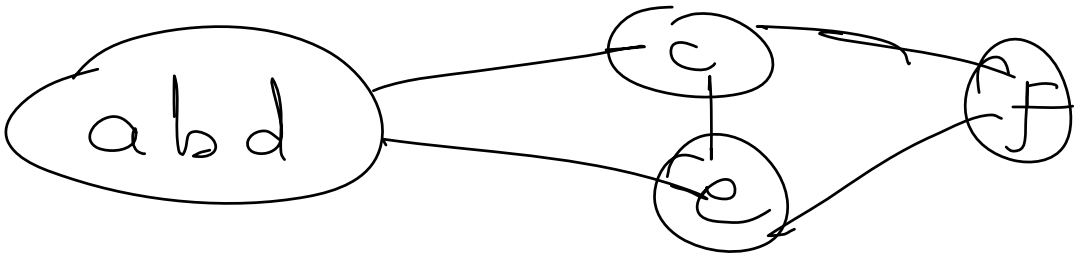
⇓

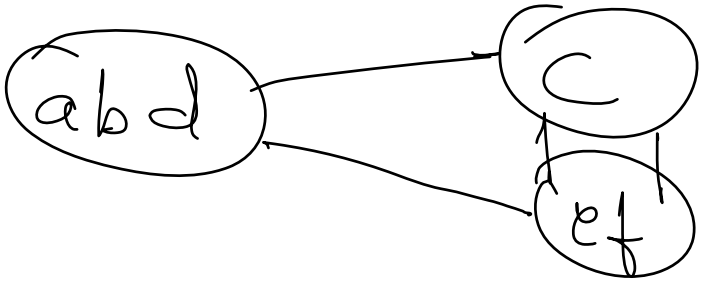


* Taking a & bd



remove self loop





* Merging c with ef

remove self