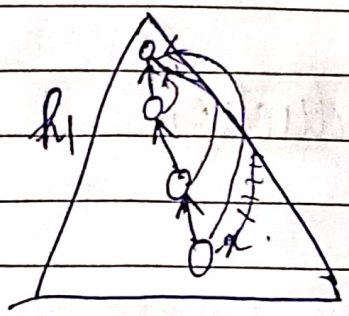
Path compression



$O(m\,\alpha(n))$ → very slow growing function

union by rank + path compression

- update all same find → then cost less

find → take that much time only

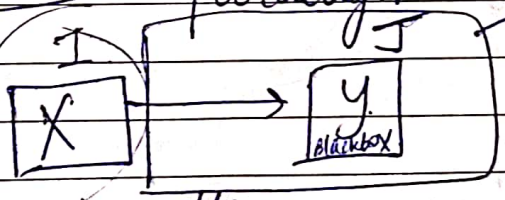
4/9

1. Dynamic programming
2. Network flows
3. NP-hardness

Fractal-2 → Portion

Recursion:

Reductions - it is a very powerful algorithm paradigm.

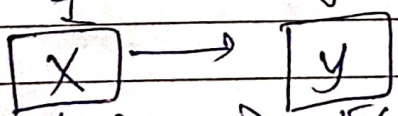


problem x → converted problem y

using blackbox → solve it.

↓ algm for y.

decision warshow problem → yes/no

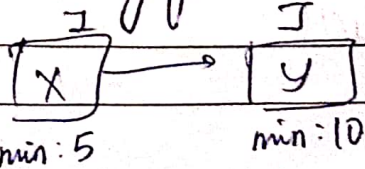


YES ⟹ YES

NO → NO ⟩ show using contrapositive.

YES ⟸ YES

- tool can be used not only for decision, but also optimisation.



- design algorithm
- lower Bound } used for

→ Recursion is a type of reduction

Vertex coloring problem

i/p :- a graph G, and integer k

Q can we color vertices of graph using atmost k colors such that for every edge $uv \in E(G)$, color of u and v is not same?

$\chi : V(G) \to [k]$ — 1 to k integers 1...k

s.t. $u,v \in E(G)$, $\chi(u) \neq \chi(v)$

↳ chromatic no

proper coloring

Q $\min^m$ no of colors.

$f(n) \times n$.

Binary search

$\frac{n}{2}$

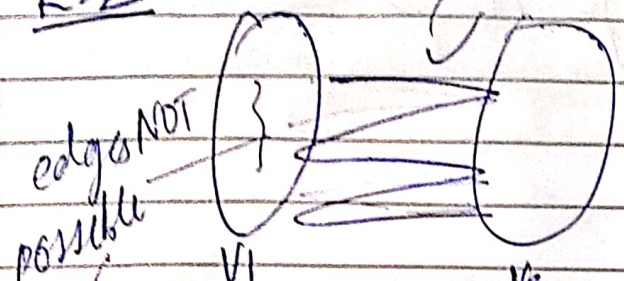
$\frac{n}{2}-1$, $\frac{n}{2}+1$

k=1

if G is edgeless then yes
trivial else No

Bipartite graph.

K=2



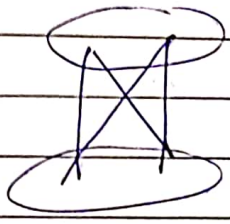
vertex set colored 1

vertex set coloured 2.

edges inside sets are not possible.

Visualise op.

If you have a soln for these 2 partitions



n, - vertices.
m - edges

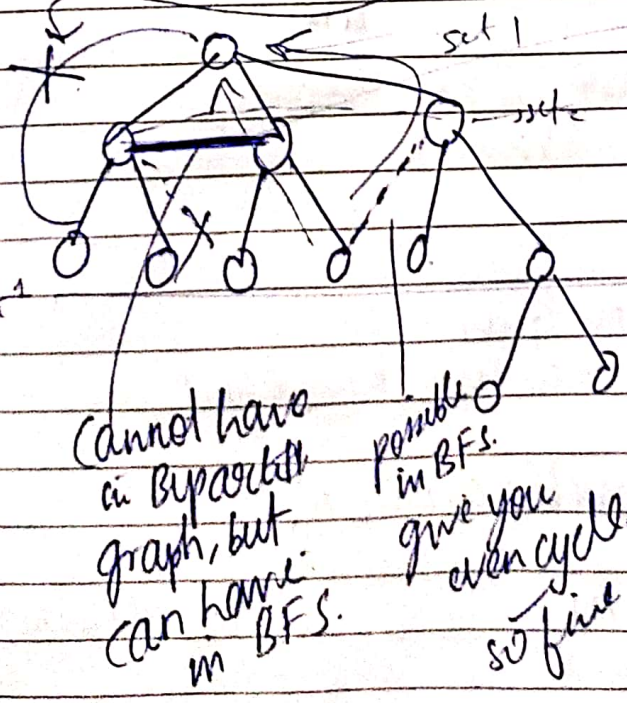
Can you vertex color in 2 colors → is reduced to checking if graph is Bipartite graph → BFS → polynomial time

go over vertices 2^n polynomial

Assign

Why does BFS work?

If you have don't have this edge in BFS then Bipartite



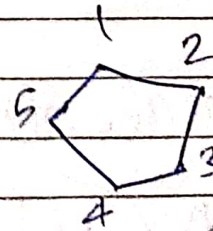
Cannot have in Bipartite graph, but can have in BFS.

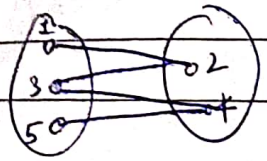
possible in BFS. give you even cycle. so fine

Property of Bipartite graph.

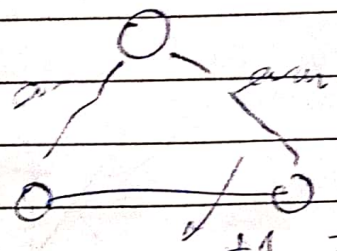
⇓

do not contain odd cycle.



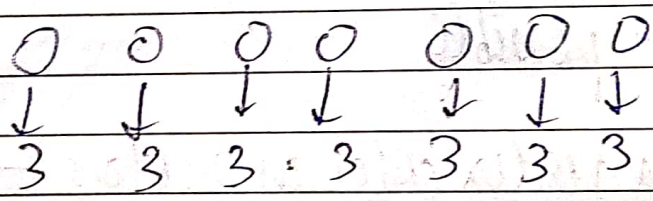


BFS → check edges at alternate levels.

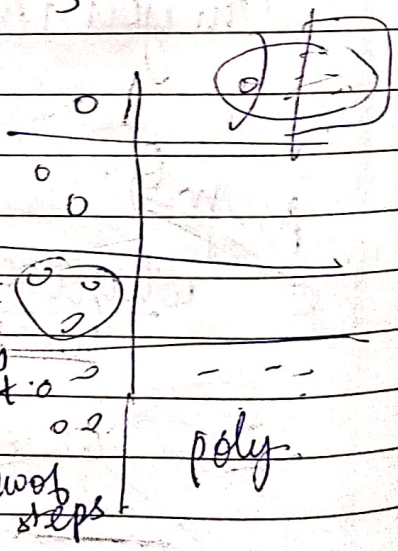


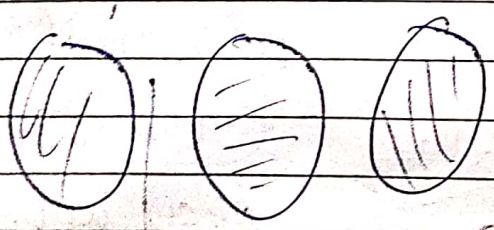
k=2 polynomial time

k=3



3^n → Brute force





given yellow color

poly.

now G-V1.

now use k=2 algm

-enumerate all possible choices of V1

$2^n \times pol(n,m)$

V1-edgeless

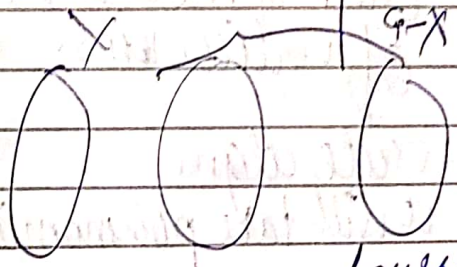
$2^n \times O(n+m)$ O(1).

3-coloring (Assume) → guess → 2-coloring
guess → 2-coloring
Some color 3rd
2^n

$X \subseteq V(G)$
$G(X)$ - edges
What guarantee $G-X$ is 2-colorable?

Now try to generalise for k-coloring

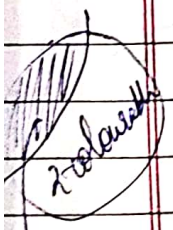
guess → try all possible options.
2^n all possible subsets.



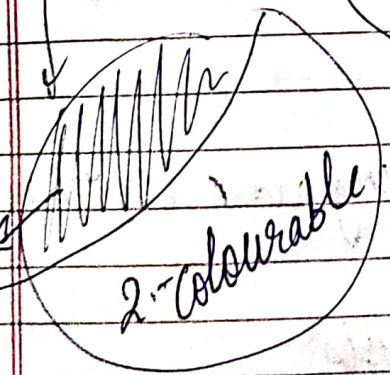
12/9 Vertex coloring

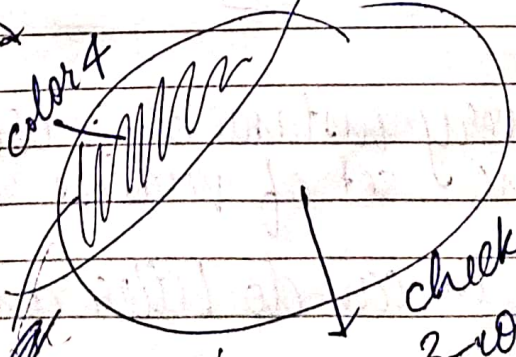
$O(k^n \times poly)$
$O^*(2^n)$
2-colourable
polynomial $\times 3^n$

What we learnt so far?:
① 2-coloring → $(n+m)^{O(1)}$ — polynomial time
Bipartite graph or not
② 3-coloring — Brute force — $O^*(3^n)$
Reduction to 2-coloring $O^*(2^n)$
③ 4-coloring — Brute force : $O^*(4^n)$.
Reduction to 3-coloring $O^*(2^{2n})$



does independent set





enumerate all possible options.

$2^n \times 2^n = 2^{2n}$

check if it is 3-colourable?

Enumerate

- Orange set - just an subset of yours whole graphs, the s.t there is no edge b/w them.

time req?

(4th) 3rd 3rd

2^n 2^n 2^n $O(2^{2n})$

$S \subseteq V(G)$

Q check if there is no edge in G b/w any pair of vertices in S.

- Write algm.
- It will take polynomial time.

$5^n \rightarrow O(2^{3n})$

(4) K-coloring - Brute force - $O^*(k^n)$

Reduction to k-1 coloring

$= O^*(2^{(k-2)n})$

convert to Base 2:

$k^n \rightarrow 2^{\log_2 k^n}$

$= 2^{n \log k}$ ← i.e. This is better.

Vs

$2^{(k-2)n}$ Brute force.

→ For every instance of 3-coloring we don't have same set of vertices.

∴ we can do better analysis

3-coloring

$\sum_{i=0}^{n} \binom{n}{i} T(n-i, 3)$

orange part — 3-colourable, white part

$\sum_{i=0}^{n} \binom{n}{i} 2^n = 3^n$

$(n+1)^n$ Binomial formula.

$T(n,k)$ - time taken to test if the graph on n vertices is k-colourable

5-coloring — n choose i

$\sum_{i=0}^{n} \binom{n}{i} T(n-i, 4)$

$\sum_{i=0}^{n} \binom{n}{i} 3^{n-i} = 4^n$

For k-coloring, brute force is better.

For 4-coloring, 5-coloring

becomes $O^*((k-1)^n)$ ∴ this is better than brute force.

Improvement over $(k-1)^n$ algorithm.

DP - ① what do we want to compute? ② how do we compute?

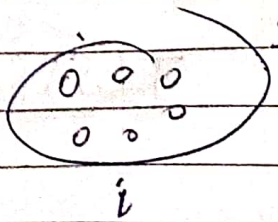
subproblem → problem

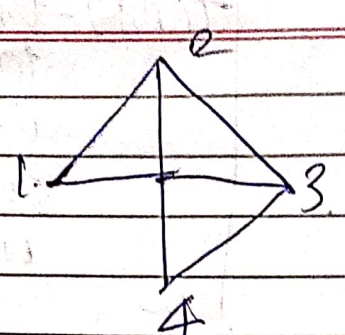
problem → subproblem

For $S \subseteq V(G)$, $i \in [K] = \{1, \ldots, K\}$ (no of colors)

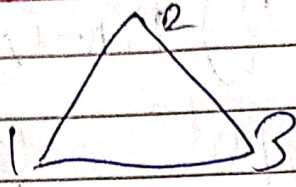
$T[S, i] = 1$ if $G[S]$ is i-colourable (induced subgraph: all edges between those vertices are present)

$= 0$ otherwise

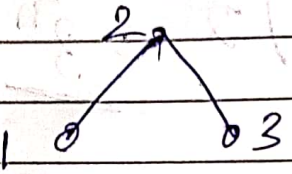




$S = \{1,2,3\}$



induced subgraph



subgraph only

For $S \subseteq V(G), i \in [K] = \{1, \ldots K\}$

$T[S,i] = 1$ if $G[S]$ is i-colourable

$= 0$ otherwise.

Base case

$\forall\ S \subseteq V(G), i = 1$

$T[S,1] = 1$ if $G[S]$ is edgeless

$= 0$ otherwise

$\forall\ S \subseteq V(G), 1 < i \leq K$

$T[S,i] = \bigvee_{S' \subseteq S} T[S \setminus S', i-1]$ — $S - S'$

$G[S']$ - edgeless ***

remaining part check colorable or not

OR

if any one subset is i-colorable

the S' is colorable.

$S_1, S_2, S_3, S_4 \subseteq S$

$S - S_1$

$T[S \setminus S_1, i-1] \vee T[S \setminus S_2, i-1] \vee T[S \setminus S_3, i-1] \vee T[S \setminus S_4, i-1]$

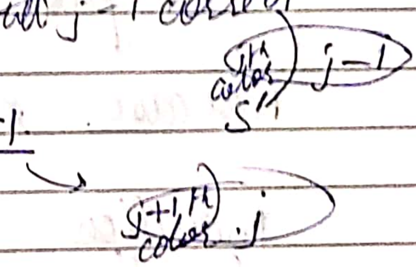
- computing for 2^n subsets, k times — i - size of S.
  - First choose S' → $\sum_{i=0}^{n} \binom{n}{i} 2^i = 3^n$

- Correctness.
  ↓
  Why is this algm correct? — mathematical induction.

goal → $T[V(G), K]$ → store 1 if graph is K-colourable

~~T[S/V]~~

i = 1 ✓
i = j — all j - 1 correct
i = j + 1



4/4.

Vertex coloring

Dynamic programming (DP).

$G = (V, E)$, K
$\forall i \in [K]$, $S \subseteq V(G)$

$$T[S, i] = \begin{cases} 1 & \text{if } G[S] \text{ is } i\text{-colourable} \\ 0 & \text{otherwise} \end{cases}$$

$\forall S \subseteq V(G)$,
$$T[S, 1] = \begin{cases} 1 & \text{if } G[S] \text{ is edgeless} \\ 0 & \text{otherwise} \end{cases}$$

$\forall S \subseteq V(G)$, $1 < i \leq k$
$$T[S, i] = \bigvee_{\substack{S' \subseteq S \\ G[S'] - \text{edgeless}}} T[S \setminus S', i-1]$$

Correctness -

$T[V(G), K] = 1$ or 0.

- if each entry in table is correct then $T[V(G), K]$ is also correct.

Lemma :- For all $S \subseteq V(G)$, $i \in [K]$, $T[S, i]$ is computed correctly.

We will prove using induction on i

Base case: $i = 1 \Rightarrow$ trivial. (write complete). Because if $i=1$, then graph is edgeless.

Induction hypothesis True for $i < j$

If $(i = j)$

→ we want to show that if $\bigvee_{S' \subseteq S} T[S \setminus S', j-1] = 1$ $G[S']$-edgeless.

then $G[S]$ is j-colourable.

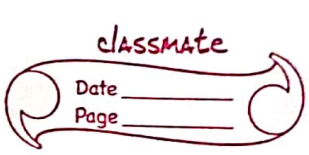
If $\bigvee_{S' \subseteq S} T[S \setminus S', j-1] = 0$, then $G[S]$ is not j-colourable

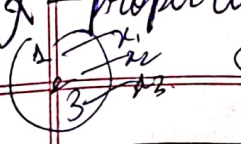
OR

→ If $G[S]$ is j-colourable, then $\bigvee_{S' \subseteq S} T[S \setminus S', j-1] = 1$ $G[S]$-edgeless

x — no of colors
χ — proper coloring set

$a=b$ ($a \le b$, $b \le a$





→ If $\bigvee_{\hat{S} \subseteq S}$ $T[S \setminus \hat{S}, j-1] = 1$, then $G[S]$ is j-colourable is true then

$G[S]$-edgeless

Proof ⇒ ∃ a subset $\hat{S} \subseteq S$ s.t $G[\hat{S}]$ is edgeless and $G[S \setminus \hat{S}]$ is $(j-1)$ colourable.

$\chi \leftarrow (j-1)$ colouring for $G[S \setminus \hat{S}]$ extend χ to χ'

Construct χ' as follows:

$\forall\, v \in S \setminus \hat{S}$, $\chi'(v) = \chi(v)$

$\forall\, v \in \hat{S}$, $\chi'(v) = j$

claim:- χ' is proper coloring for $G[S]$

→ proving this

⇐ $G[S]$ - j-colourable.

There exists a proper j-coloring χ for $G[S]$

let $S' = \chi^{-1}(j)$ vertices colored using jth color.

$G[S \setminus S']$ is $j-1$ colourable.

$O(3^n)$ algm for vertex coloring

- we have considered S' : and $S' \subseteq S$

and $G[S']$ is edgeless because it is colored using jth color.

∴ proved.

any one subset needs to be $j-1$ colourable.

2^n - known

little o $\rightarrow$ $2^{o(n)}$ - no hope of such an algm for vertex coloring

1.99^n - ?

open question.

SAT — satisfiability assignment set to 0 or 1.

$V = \{x_1, x_2, \ldots x_n\}$.

ℓ = a set of clauses.

CNF formula $\begin{cases} (x_1 \vee x_4 \vee \underbrace{x_5}_{\text{+ve literal}}) \text{ - clause of size 3} \\ (x_1 \vee \underbrace{\bar{x}_2}_{\text{-ve literal}} \vee x_3) + \end{cases}$

- Does there exist an assignment $\psi : V \rightarrow \{0, 1\}$ s.t given CNF formula is satisfied?

- value of assignment to variables - result to true

$\psi(x_2) = 1$

3 SAT - atmost 3 literals in every clause

cell 10.

no hope for $2^{o(n+m)}$

SAT —poly. time→ Vertex coloring

n - no of vertices.

$|V(G)| \leq n+m$. no of vertices bounded by $n+m$.

clauses

NP hardness.

$2^{o(|V(G)|)}$ — we cannot take this as no hope for $2^{o(n+m)}$.

↓

such algm does not exist.

15/9

Vertex coloring

Lower Bound -

3-SAT problem.

↳ ip set of variables $X = \{x_1, \ldots, x_n\}$
set of clauses $C = \{C_1, \ldots, C_m\}$
such that size of each clause is 3.

Q Does there exist an assignment $\psi : X \to \{0,1\}$ s.t each clause is satisfied.

Assumptions

① We cannot hope to design polynomial time algm for 3-SAT unless P ≠ NP.

② Under exponential Time hypothesis (ETH) we cannot

Hope to obtain $2^{o(n)}$ (little o) algorithm for 3-SAT.

clause example — variable x_1 → +ve literal x_1, -ve literal $\bar{x}_1$

↳ $C_1 = (x_1 \vee \bar{x}_2 \vee x_3)$ no of literals = size of clause

$C_1 = (x_1 \vee x_2 \vee x_3)$
$C_2 = (\bar{x}_1 \vee \bar{x}_2 \vee x_4)$
$C_3 = (x_3 \vee \bar{x}_4 \vee x_2)$

variables ↳ x_1, x_2, x_3, x_4 — $C_1 \wedge C_2 \wedge C_3$

assignment
$x_1 = 1, x_2 = 0, x_3 = 0, x_4 = 1$

(if all satisfied) { C_1 ✓, C_2 ✓ } satisfied, C_3 X ↳ NOT satisfied — Just this assignment

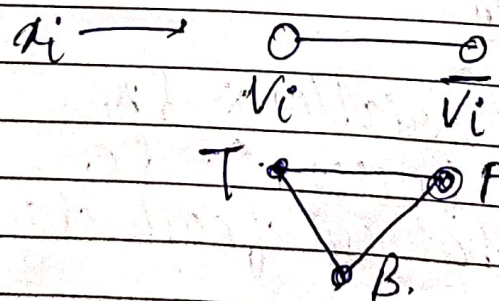
∴ satisfying assignment

↓

eg $x_1 = 1, x_2 = 0, x_3 = 0, x_4 = 0$

variable/literal. vertices.

→ Corresponding to x_i, I create 2 vertices



Reducing to 3-colorable

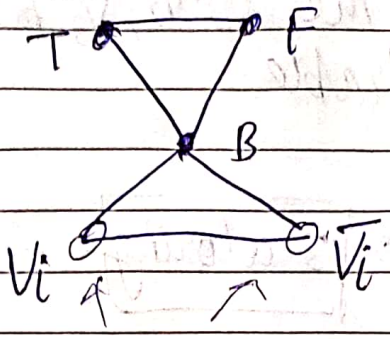
— we need 3 colors.

T F B

color → True
— False
— Base

} associate colors with assignment

we have this △ triangle in our graph.

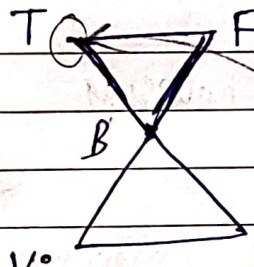


① For each $x_i \in X$, we add vertices v_i & $\bar{v}_i$ in G and add edge $v_i \bar{v}_i$

② We add a triangle {T, F, B}

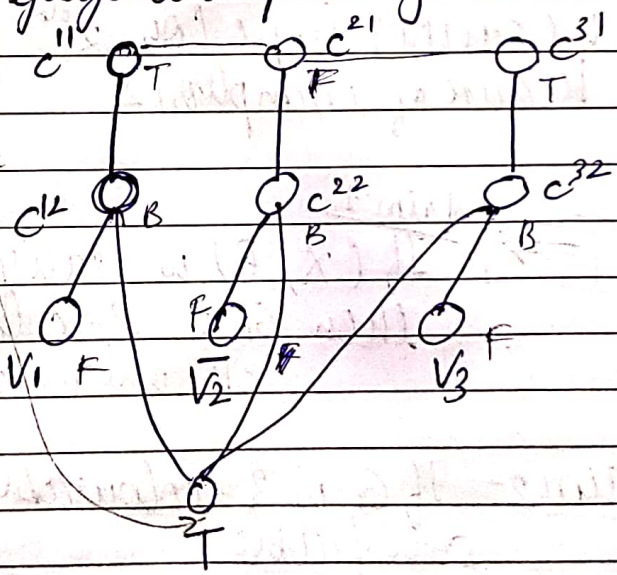
③ For all $i \in [n]$, $v_i B$ and $\bar{v}_i B$ are edges in G.

can only take T or F but either $v_i \to T, \bar{v}_i \to F$ OR $v_i \to F, \bar{v}_i \to T$.



same

gadget corresponding to clause C



6 vertices

3 copies

because 3 set

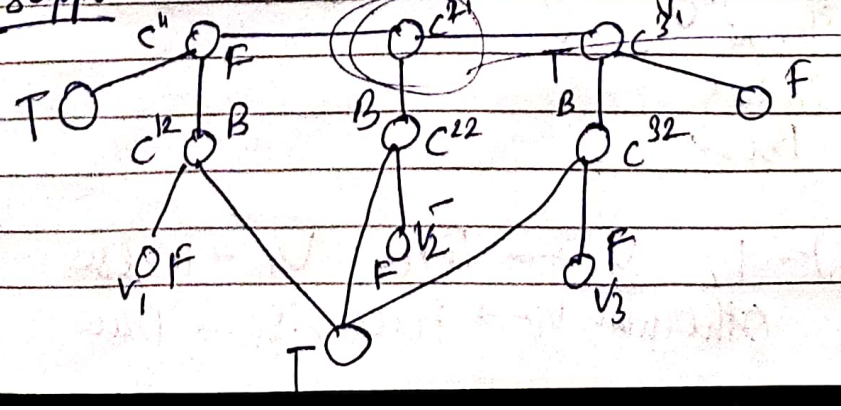
let $C = (x_1 \vee \bar{x}_2 \vee x_3)$

for every clause, add 6 vertices.

connect them correspondingly to vertices according to these literals

either v_1 or $\bar{v}_2$ or v_3 is colored True.

Suppose $v_1, \bar{v}_2, v_3$ → all are taking color F



no color available

∴ NOT 3-colorable

- we should force one of the v_1, v_2, v_3 to be True for it to 3-colorable.

3-SAT $\xrightarrow{\text{poly time}}$ 3-coloring

suppose $2^{o(n)}$

yes $\Leftarrow$ $\Rightarrow$ yes.

No. $\Leftarrow$ $\Rightarrow$ No

poly. time + $2^{o(n)} \simeq 2^{o(n)}$.

we cannot hope for $2^{o(n)}$ for 3-coloring because of assumption 2

proof $\Rightarrow$ claim 1

If $(X, \mathcal{C})$ is satisfiable then G is 3-colourable

> the one we constructed.

claim 2 – If G is 3-colourable, then $(X, \mathcal{C})$ is satisfiable

contra +ve

proof of claim 1 :-

Let ψ be satisfying assignment to $(X, \mathcal{C})$

vertex → one color

T → True

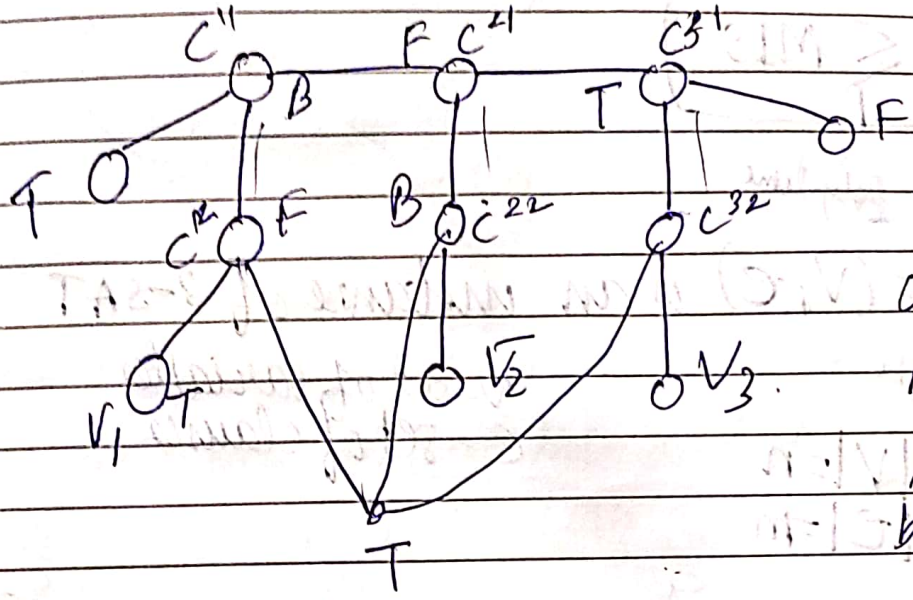
F → False

B → Base

if $\psi(x_i) = 1$, $v_i \rightarrow$ True, $\overline{v_i} \rightarrow$ False

Otherwise, $v_i \rightarrow$ False, $\overline{v_i} \rightarrow$ True

let $V_1 \rightarrow$ True ∴ (any one).



Coloring is proper b/w Type of edge / b/w clauses

for claim 2:
↳ not all V_1, V_2, V_3 are taking F. ← prove this

19/9

* Maximum Independent set (MIS).

- Independent set :- Let $G=(V,E)$ be a graph. A set $S \subseteq V(G)$ is called an (independent set) of G if for every pair $u,v \in S$, $uv \notin E(G)$.

Maximum independent set

ip. - a graph G

o/p - find a maximum sized independent set in G.

at most k size
↳ singleton set

any algm?

① Test all possible subsets of $V(G)$ → $O^*(2^n)$.

Can we design better algm? - in terms of running time.

Can we design $n^{O(1)}$ time algm?

3-SAT $\leq_P$ MIS

poly. time

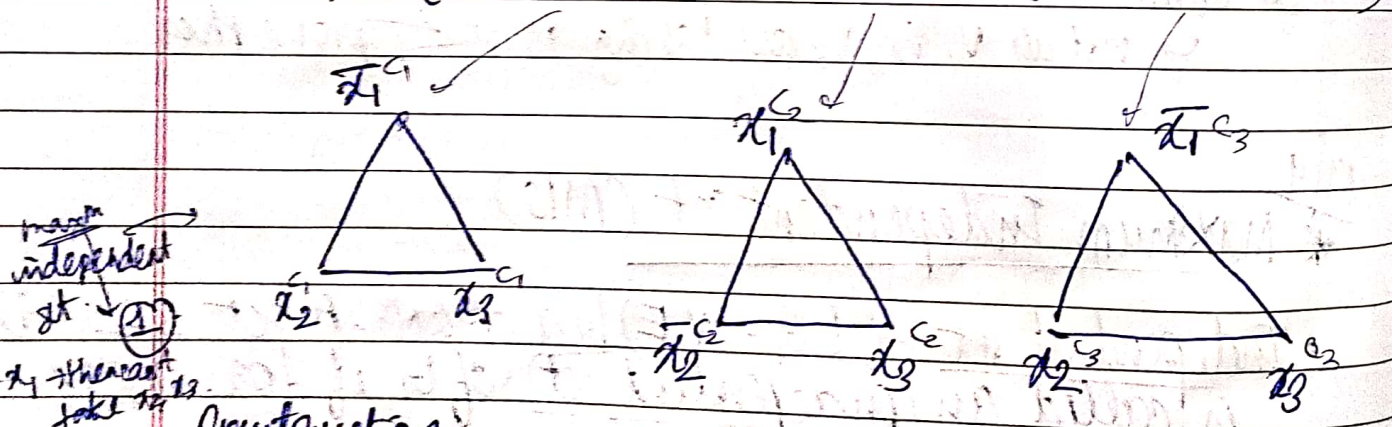
$\psi = (V, C)$ is an instance of 3-SAT

↓ CNF formula

V - set of variables
C - set of clauses

$|V| = n$
$|C| = m$

$\psi = (\bar{x}_1 \vee x_2 \vee x_3)^{C_1} \wedge (x_1 \vee \bar{x}_2 \vee x_3)^{C_2} \wedge (\bar{x}_1 \vee x_2 \vee \bar{x}_3)^{C_3}$



max independent set → (1) x_1 then can't take $\bar{x}_1, x_2, x_3$

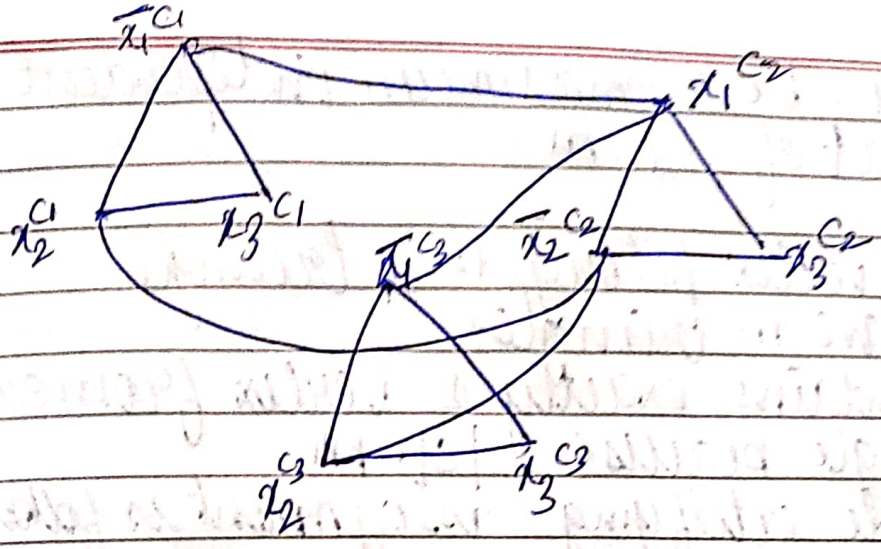
Construction

Construct graph G_ψ as follows,

(1) Corresponding to every literal in a clause, we have a vertex in G_ψ. Suppose $C = (l_1 \vee l_2 \vee l_3)$ be a clause in ψ. Then we have vertices l_1^C, l_2^C, l_3^C in G_ψ and form a triangle using these literals.

(2) Let $x, \bar{x} \in V(G)$ s.t x is the vertex corresponding +ve literal of x and $\bar{x}$ " " -ve " " "

Add edge $x\bar{x}$



either $\bar{x}_1$, x_1 both cannot take 1 together ∴ draw edge b/w +ve, -ve literal

→ -ve, +ve of same variable then add edge.
reason → we cannot take both x_i and $\bar{x}_i$ in independent set

m clauses. $x_1 x_2 x_3$ $\bar{x}_1 \bar{x}_2 \bar{x}_3$

max^m independent set → m. (1 from each Δ)

lemma ψ is satisfiable iff G_ψ has a max^m independent set of size m.

Proof $\Rightarrow$ Let ϕ be a satisfying assignment for ψ.

Let l^* be a literal in the clause that is responsible to satisfy C.

$P(l^*) = 1$

Construct a set S as follows:- add l^* to S.
$|S| = m$. (1 literal for each clause)

∴ maximum → cannot be more than m.
is it independent? → yes, because you are picking either x_i or $\bar{x}_i$ not both.

∴

$\Leftarrow$ Let S be a maximum independent set of size m.

1. each vertex belongs to a triangle
2. we have m triangles.
3. S contains exactly 1 vertex from each triangle because $|S| = m$.
4. Create satisfying assignment as follows.
   if $x \in S$, $\phi(x) = 1$

corresponding to each Δ, 1 literal is chosen
$\therefore$ satisfying assignment.

Vertices = 3m

NP-Hardness. not P $\rightarrow$ does not imply NP.

Under ETH, 3-SAT cannot be solved in $O^*(2^{o(n)})$ time.

Can we contradict $2^{o(n)}$ algm. for MIS where $\hat{n}$ is the no. of vertices in the graph?

$\hat{n}$ = vertices

$\hat{n} = 3m$; $m \leq \binom{2n}{3}$

each clause is distinct

$= O(n^3)$

$O(n^3)$.

$2^{o(\hat{n})}$ — not possible $\leftarrow$ this statement not correct

$2^{o(\hat{n}^{1/3})}$

$2^{o((n^3)^{1/3})} = 2^{o(n)}$.

Sparsification Lemma

Under ETH, 3-SAT can not be solved in $O^*(2^{o(n+m)})$ time

$\Rightarrow 2^{o(n)}$ not possible for MIS

→ Using that conjecture we proved that.

ETH-1

2^n - algm for MIS.

$2^{o(n)}$ - not possible for MIS.

$(2-\varepsilon)^n$ - ? for $\varepsilon > 0$.

21/9

Maximum independent set

① $O^*(2^n)$ algm.

② no hope to obtain an algm for MIS that runs in $O^*(2^{o(n)})$ time.

Exercise1:- give a reduction from SAT to MIS
- arbitrary size of clause
- for SAT we don't know if there is an algm $O(2^n)$.

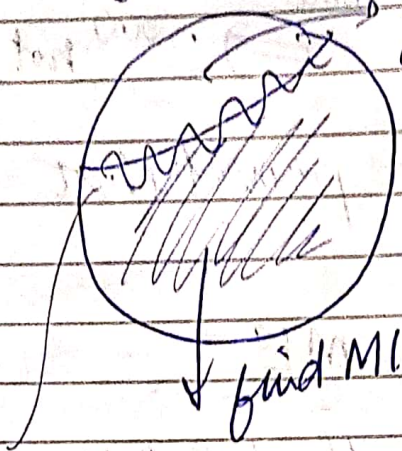
Exercise2 counter such algm for MIS using reduction from SAT.

We cannot counter - observe that ←

Goal:- design an algm for MIS that runs in $O^*((2-\varepsilon)^n)$ time when $0 < \varepsilon < 1$

- mimic 3-coloring, k-coloring ideas.

TRICK



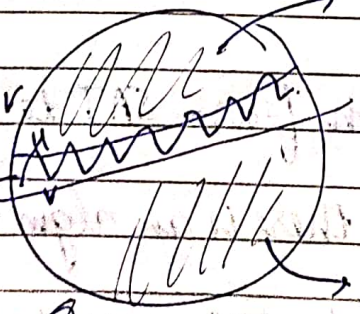
guess a subset that is an independent set

find MIS in this part

If we merge them → will it be IS?

Feasibility? No. The union of these 2 subsets may not be independent set

Because we haven't taken care of those edges b/w 1st & 2nd subset.



→ subset is an IS → 2^n

Take u, remove v.

neighbours of blue part

remove them from this subset

→ find MIS here

this idea does not give us any benefit

subset.

all possible subsets

Approach 2 - Dynamic programming

for all $S \subseteq V(G)$

$T[S]$ = size of MIS in $G[S]$

For $S \subseteq V(G)$ s.t $|S| = 1$ singleton.

$T[S] = 1$

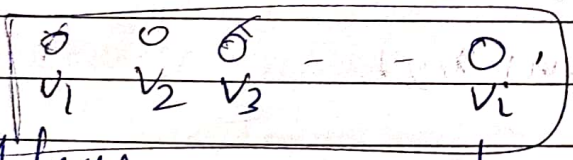
For $S \subseteq V(G)$ s.t $|S| > 1$

$$T[S] = \max_{S' \subseteq S} \left([T[S \setminus N[S']] + |S'| \right)$$

$N[S']$ closed neighbourhood

S' - independent set.

$\hookrightarrow O^*(3^n)$



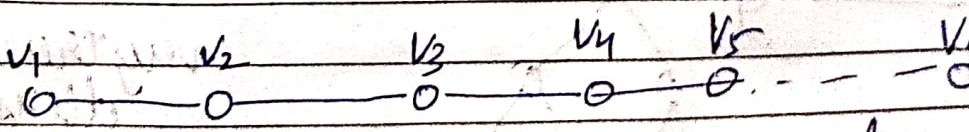
neighbours scattered

it can be here

we can't take v_i or not.

If we take v_i then we have to delete neighbours of v_i

Path:



possible to order vertices such that there is edge only b/w consecutive vertices.

every vertex v_i is neighbour of v_{i-1} and v_{i+1}

$\forall i \in [n]$

$T[i]$ = size of maximum ind. set in $G[\{v_1, \ldots, v_i\}]$ induced on

Base case

$T[1] = 1$

$i > 1$ $T[i] = \max\{T[i-2]+1, T[i-1]\}$

2 options.

v_i in set → $T[i-2]+1$

v_i not in set → $T[i-1]$

Exercise 2:-

Suppose you have weights on graph.

Find max weight independent set in path.

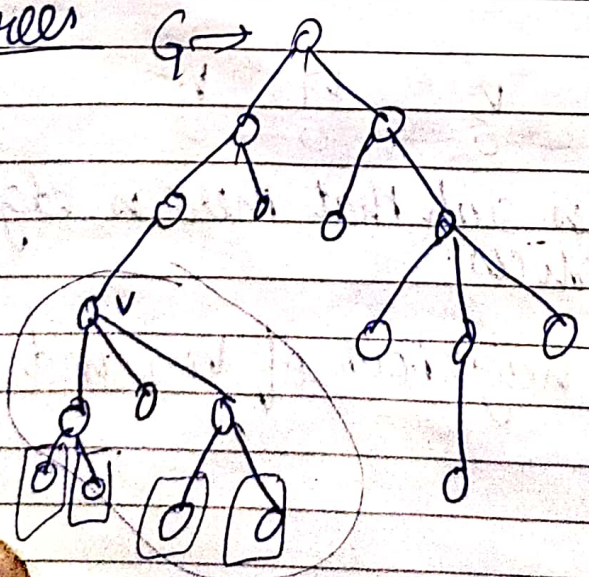
size of DP & time taken by 1. } to find T.C. ⇒ $O(n)$

const → max(x, y).

path → poly. time ✓

Trees

forest



every tree contains atleast 2 degrees.

DP - always. Imp is to understand subproblem

classmate Date Page

$\forall v \in V(G)$

$T[v]$ = size of Maximum independent set in the tree rooted at v

Bottom-up DP

Base case :-

for leaf vertex v

$T[v] = 1$

notation for $v \in V(G)$, $C(v)$ = set of children of v

$GC(v)$ = set of grandchildren of v

(contd)

for non-leaf vertex v

$$T[v] = \max \left\{ 1 + \sum_{x \in GC(v)} T[x] \;,\; \sum_{x \in C(v)} T[x] \right\}$$

prove correctness of algm

↳ quiz, minor 2

atleast poly time

DP didn't work.

max → n children

exercise 3 look for — FS

22/9

Maximum independent set

S - solution.

$v \in S$ | $v \notin S$

$G = N[v]$ | $G - v$

neighbourhood of v and v and neighbours of v deleted from

```
Recursive MIS(G)
if ΔG = 0, then return V(G)
let v ∈ V
a = Recursive MIS(G-v)
b = Recursive MIS(G-N[v])
return max{a,b}
```

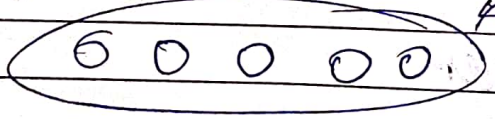
$T(n) \leq T(n-1) + T(n-1-\deg(v)) + n^{O(1)}$

2^n

if $\deg(v) = 0$

$T(n) = 2^n$

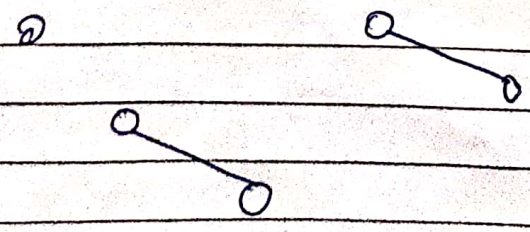
$T(n) = T(n-1) + T(n-2)$

$G =$  ← zero degree vertices

$T(n) = x^n$ ⇒ $x^n = x^{n-1} + x^{n-2}$ — divide by x^{n-2} throughout

$x^2 = x + 1$ | roots are $x = 1.168$.

$\Delta G \leq 1$ — disjoint edges in graph



1 degree vertices.

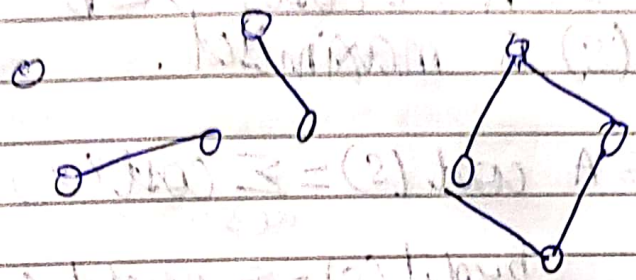
isolated vertices + pick any 1 in that edges.

if $\Delta G \leq 1$ solve in poly. time

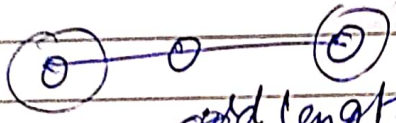
$x^3 = 1 + x^2 \Rightarrow 1.1656$.

if $\Delta G \geq 1$.

Now if $\Delta G \geq 2$. max^m degree of graph almost 2

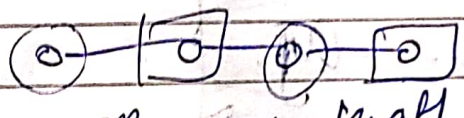


deg 0
or paths
or cycles.

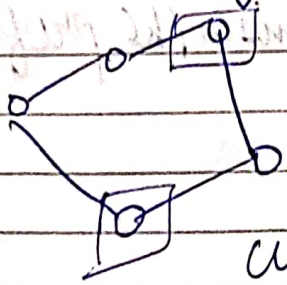


odd length

OR



even length. (path)



$\left\lfloor \frac{n}{2} \right\rfloor$.

cycle

poly time solvable ✓ $x^4 = 1 + x^3 \Rightarrow 1.3803$.

$2^n \rightarrow 1.168^n \rightarrow 1.4656^n \rightarrow 1.3803^n \rightarrow \dots$

1.1996^n.

(2019 yr.)

improving 3rd and and decimal.

2^n.

Exercise

Design efficient algm for 3-SAT using trick we learned today

(need in fractal-3 as well)

* knapsack

Set of items $A = \{a_1, \ldots, a_n\}$

fns { cost: $A \to \mathbb{N}$, profit: $A \to \mathbb{N}$, budget: B } functions are additive

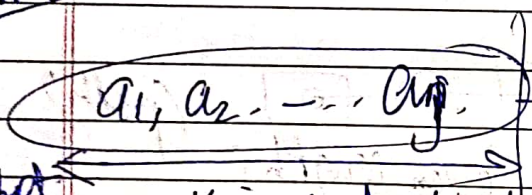
$\mathbb{N}$ natural no

Q find a subset $S \subseteq A$ s.t $cost(S) \leq B$ and $Profit(S)$ is maximised.

For a subset $S \subseteq A$ $cost(S) = \sum_{\alpha \in S} cost(\alpha)$

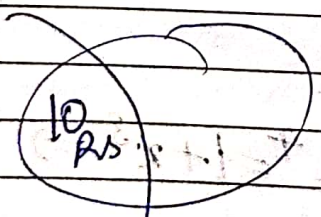
$profit(S) = \sum_{\alpha \in S} profit(\alpha)$

NP hard



within the budget what maximises the profit

- Budget for subproblem (vimp)



we'll try all possible budgets.

1 Rs, 2 Rs, - - -

∴ NOT a poly-time algm.

$a_1, a_2, \ldots a_n$.

for all $i \in [n]$, $b \in [B]$

$T[i, b]$ = max profit of a subset of $a_1, \ldots a_i$ within budget b. of cost b.

i^{th} item.

$T[n, B]$ = max profit of a subset of $a_1, \ldots a_n$ within budget B.

↓ we can compute this

$a_1, \ldots a_{i-1}, a_i, a_{i+1} \ldots a_n$

$\sum a_i$

Base case

$i = 1$, $b \in [B]$

$T[1, b]$ = profit(a_i), if cost(a_i) $\leq b$.

$= -\infty$, otherwise

For $i > 1$

a_i belongs to soln or not (2 choices)

$$T[i, b] = \max \begin{cases} T[i-1, b], & a_i \text{ does not belong to soln} \\ T[i-1, b - \text{cost}(a_i)] + \text{profit}(a_i) & \text{max of these} \end{cases}$$

Need to prove correct of this ↑

DP for knapsack

becomes poly if b is polynomially bounded by instance size

If $B = n^{O(1)}$ (↑ w) then it is poly. time algm.

details of knapsack → DP

"Pseudo polynomial time" algms. these are called!

size → log[B] x n

no of entries: n x B

Why is it not polynomial in size of i/p?
It is because: n x log n.
i/p.

It is not B x log B → ✗
∴ not poly time

26/9 Text segmentation.

BOTHEARTH ANDSATURNSPIN

↓

String of characters → segment into meaningful english words.

BOTH EARTH AND SATURN SPIN
BOT HEART HAND SATURN SPIN
BOT HE ART " " "
" " " SAT URN " "

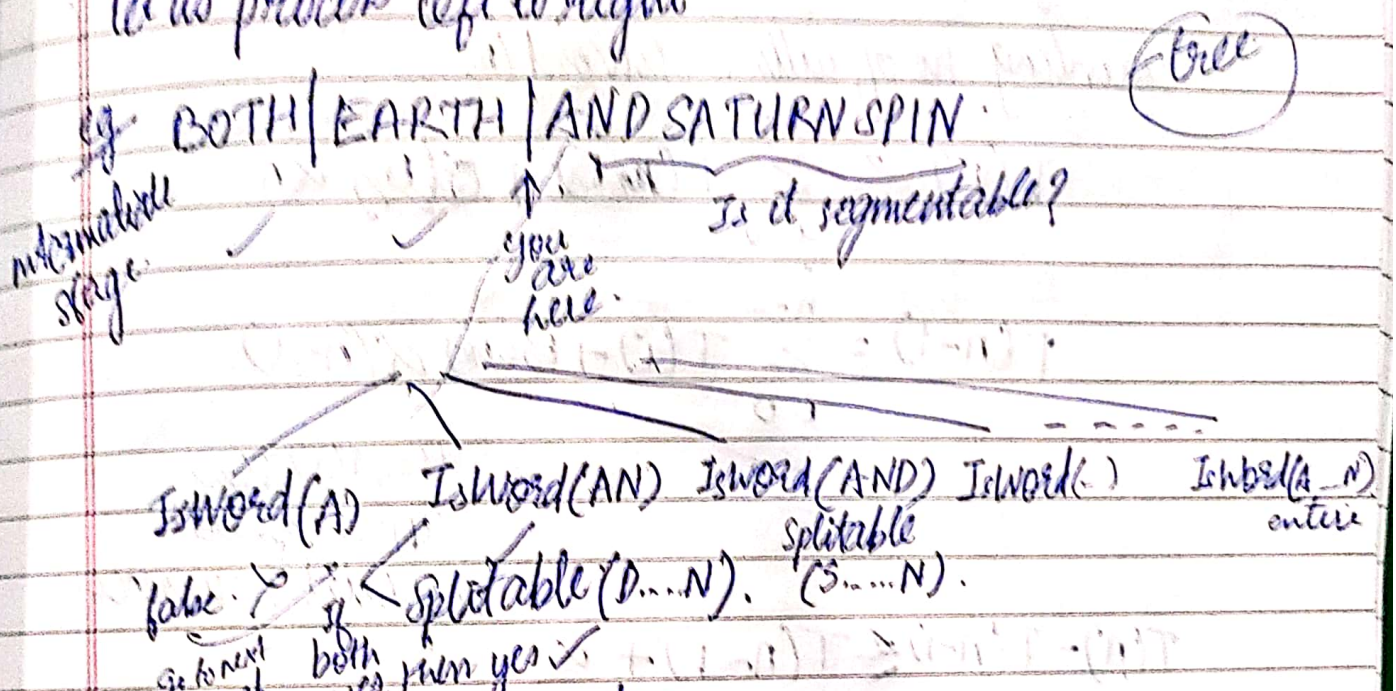
I/p - a string of characters A
Q - Can A be segmented into English words?

Blackbox Dictionary

IsWord(W) → it takes W and returns TRUE iff W is an English word.

Let us process left to right

eg BOTH|EARTH|AND SATURN SPIN



Guess → means enumeration here.

```
SPLITTABLE(A[1,...n]).
    if n=0
    return true
    for i=1 to n
    {  if IsWord(A[1,...i])
       {  if SPLITTABLE(A[i+1,...,n])
          return true
    }
    return false.
```

A=BOTHEAR.
i=1 to 7.
IsWord ✓
Splitable(HEAR)
i=1 to 4.
HE → word.
Splitable(AR).
↳ return false.
∴ He not taken as word.

Correctness. – by induction.

Running time →

counting no. of calls to IsWord(·).

$$T(n) \leq \sum_{i=0}^{n-1} T(n-i) + \alpha n$$

$$T(n-1) = \sum_{i=0}^{n-2} T(i) + \alpha(n-1)$$

T(0), T(1), T(2) appear in both — repeated occurrence.

$$T(n) - T(n-1) \leq T(n-1) + \alpha$$

$$T(n) \leq 2T(n-1) + \alpha.$$

$$T(n) = O(2^n)$$

- not a good idea to pass arrays.
- better to pass indices.

→ We only call splitable in suffix part. ∴ we only need i. — i to n

→ IsWord can be asked anywhere ∴ need 2 indices i & j

SPLITTABLE(i) = 1 iff A[i...n] is splittable

IsWord(i,j) = 1 iff A[i,...j] is a word

SPLITTABLE(i) = { 1 if i > n
$\bigvee_{j=1}^{n}$ IsWord(i,j) ∧ Splitable(j+1). otherwise
(const) (const)

- you can store value of splitable(i) — use DP.
  n entries — you are computing for each i → 1 to n.
  ↳ each entry can take n time summation ... n.
  ∴ total n^2 time

surprise Quiz — correctness of algm.

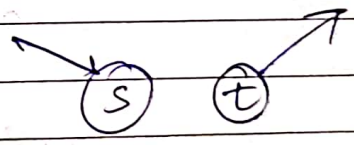
29/9

Lec23 Network flows.

Jeff Herron → N/w flows. topic

maximum flow and minimum cut

Input → a directed graph G = (V, E).



→ 2 special vertices s, t.
- s - source (there is no arc coming to s)
- t - sink (" " " " going out of t).
- source and sink are unique.

→ every edge e has +ve capacity $c_e > 0$
→ no isolated vertex (deg = 0) indegree = outdegree = 0.
→ all nodes other than s, t are internal nodes

u → v : arc from u to v
(u, v) : " " " "

directed { undirected uv — (u, v) u → v are from u to v.

Flow - function on edges.

s-t flow is a function $f: E(G) \to \mathbb{R}^+$ satisfying following properties

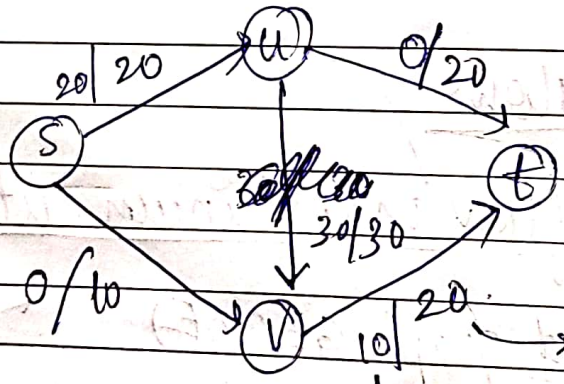
- capacity constraint : $0 \le f(u \to v) \le c(u \to v)$
- conservation " : $\forall u \in V(G), u \notin \{s, t\}$

for internal nodes

$$\sum_{(u \to v) \in E(G)} f(u \to v) = \sum_{(v \to u) \in E(G)} f(v \to u)$$

outgoing flow = incoming flow

eg



→ capacity

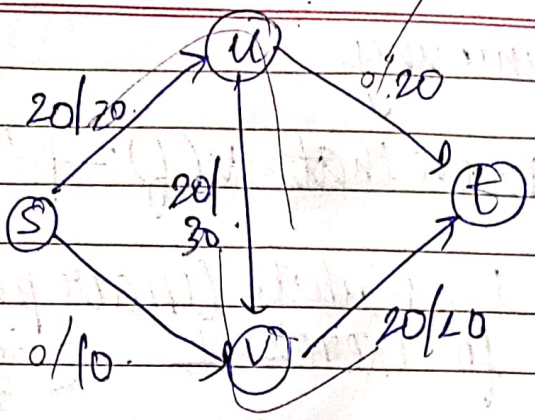
→ actual flow.

Is it a flow?

- capacity constraint → all met.
- conservation " → check for cut nodes u, v

for u, incoming 20
outgoing 30) ∴ not ✗.

0 only if nothing mentioned



it is a flow

u → 20 → 20 ✓

v → 20 → 20 ✓

$f^{out}(u) = \sum_{(u \to v) \in E(G)} f(u \to v)$ total flow going out of u

$f^{in}(u) = \sum_{(v \to u) \in E(G)} f(v \to u)$ " " coming into u.

value of flow f is $v(f) = f^{out}(s)$

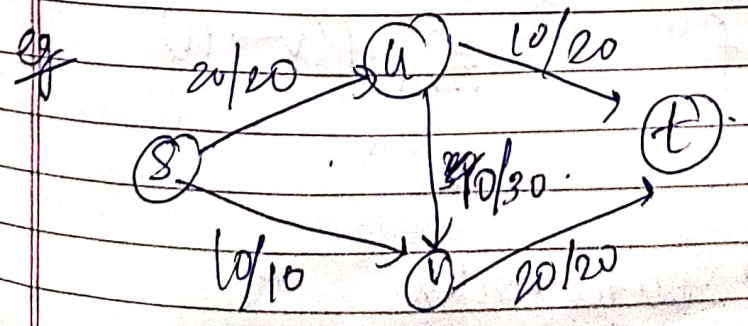
goal → find max^m flow.

find ↓ $f : E(G) \to \mathbb{R}^+$ such that v(f) is maximised

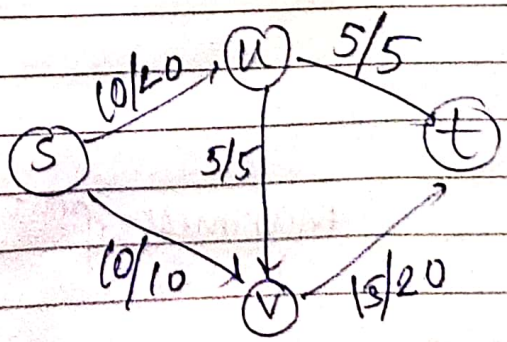
optimisation constraint

satisfying capacity & conservation constraint (feasibility constraint)

eg



max m is 30.



Linear programming

exercise1 prove that $v(f) = f^{in}(t)$

11 2 Formulate linear programming for exercise 1 — text — objfn.

Q. can we solve LP in poly-time?

A. Yes.

simplest method — is NOT poly time

MIS → ILP → not poly time

- N/w flow

$f: E(G) \to \mathbb{R}_{\geq 0}$

capacity constraint : $f(e) \leq c(e) \quad \forall e \in E(G)$

conservation " : $\forall u \in V(G), f^{out}(u) = f^{in}(u)$

$$f^{out}(u) = \sum_{\substack{u \to v \\ \in E(G)}} f(u \to v)$$

$$f^{in}(u) = \sum_{\substack{(v \to u) \\ \in E(G)}} f(v \to u)$$

maximise $f^{out}(s)$

↳ source vertex.

LP formulation

- edges take values.

→ variable :. x_{ij} corresponding to arc $i \to j$

→ constraint : $x_{ij} \leq c_{ij} \quad \forall (i \to j) \in E(G)$

→ $\sum_{(i \to j) \in E(G)} x_{ij} = \sum_{(j \to i) \in E(G)} x_{ji} \quad \forall i \in V(G)$

→ $x_{ij} \geq 0$

→ we want to maximise flow

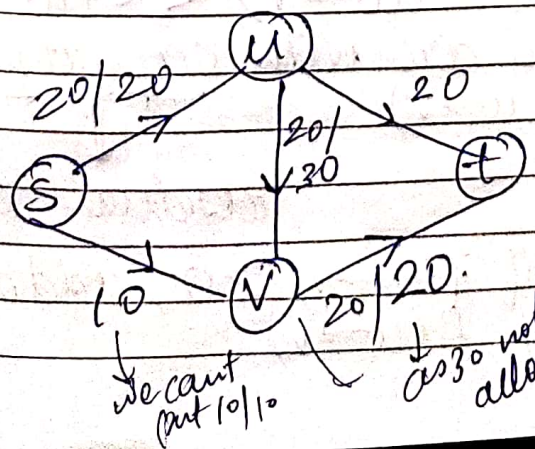
$$\max \sum_{s \to j \in E(G)} x_{sj}$$

→ This is a polynomial time reduction.

→ This is an optimisation problem :. no need of if

N/W flow is NP-solvable in poly. time ✓

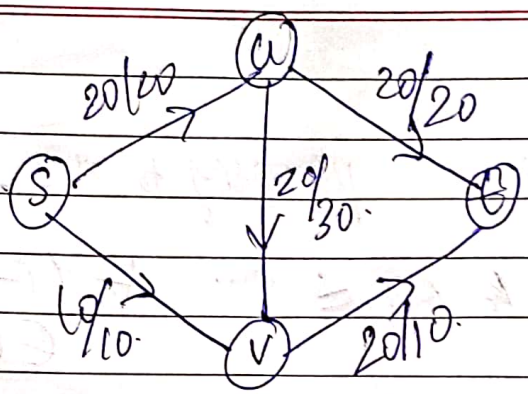
→ Simplex algorithm. → classical algm to solve linear programming.
— not a poly time algm, just an exponential time algm.



Not max flow.

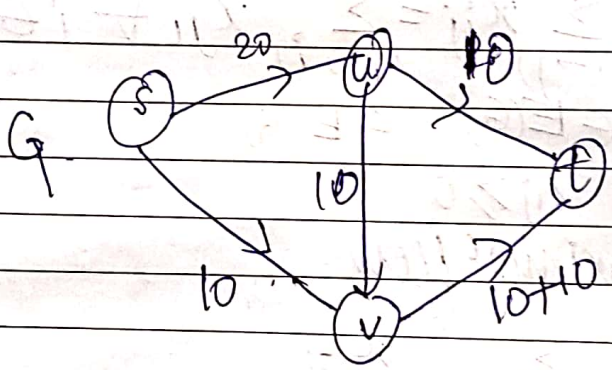
Ford Fulkerson algm

→ push back and undo some flow



greedy

max flow possible → 30 from source



→ 0 flow is valid.

Idea:

1) Push forward an edge that has leftover capacity
2) Push backward an edge that are already carrying some flow in the direction opposite to it

Given a flow, we create a residual graph G_f as follows:

$V(G_f) = V(G)$ - same as graph.

add 2 type of edges

① Forward edge: if $f(u \to v) < c(u \to v)$ then add an arc with capacity $c(u \to v) - f(u \to v)$

↓ residual capacity

② Backward edge: if $f(u \to v) > 0$, add arc $v \to u$ with capacity $f(u \to v)$

use DFS ∴ we can find in poly time

Suppose that P is an s-t path in G_f

$b(P, f)$ ⟹ min^m capacity of an edge in P

bottleneck capacity

constructing a flow

Construct a function f' as follows.

$$f'(u \to v) = f(u \to v) \quad \text{if } (u \to v) \notin P$$

edge not in P.

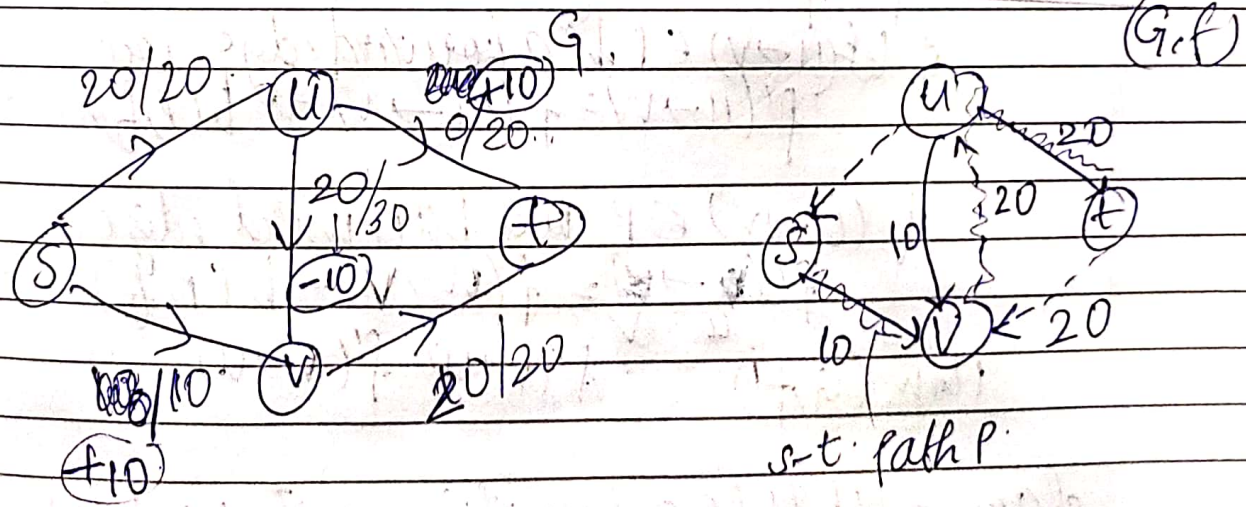
If $(u \to v)$ is a forward edge in P, then

$$f'(u \to v) = f(u \to v) + b(P, f)$$

If $f(u \to v)$ is a backward edge in P, then

$$f'(u \to v) = f(u \to v) - b(P, f)$$

Lemma → f' is a flow.



$b(P, f) = 10$.

- at every step, we'll atleast increment a flow by 1.
- Iteratively repeat algm, create graph, repeat.
  ↳ till no. of paths

$\Sigma \rightarrow$ no. of iterations.

→ Not a poly time. It is a pseudo-poly time algm.

6/12 Network Flow

Initially $f = 0$ ← zero flow

Flow (G, f)
  construct a residual graph G_f.
  if there is no s-t path in G_f,
    then return f.
  else
    let P be a s-t path in G_f.
    compute b(P, f) ▷ bottleneck capacity
    f' = f
    if $(u \rightarrow v) \in P$ is a forward edge arc
      $f'(u \rightarrow v) = f(u \rightarrow v) + b(P, f)$

    If $(u \rightarrow v) \in P$ is a backward edge.
      $f'(u \rightarrow v) = f(u \rightarrow v) - b(P, f)$
  Flow (G, f') → $f'(v \rightarrow u) = f'(v \rightarrow u)$.

claim:- let f' be a function returned by algm. f' is a flow

Proof Capacity constraint.

To show:- for all $(u \to v) \in E(G)$

$0 \le f'(u \to v) \le c'(u \to v)$ (in prime)

case 1:-

if $(u \to v) \in E(G)$ but $(u \to v) \notin p$

then $f'(u \to v) = f(u \to v)$

since $f(u \to v)$ is a flow, $\therefore$ it satisfies constraints

$\therefore 0 \le f(u \to v) \le c(u \to v)$

$0 \le f'(u \to v) \le c(u \to v)$

case 2:- $u \to v \in p$

case A:- $u \to v$ is a forward edge

(by construction) $f'(u \to v) = f(u \to v) + b(p, f)$

$\therefore 0 \le f(u \to v) \le f'(u \to v) = f(u \to v) + b(p, f)$

$b(p, f) \le c(u \to v) - f(u \to v)$

residual capacity

$f'(u \to v) \le c(u \to v)$

case B:- $u \to v$ is a backward edge

$f'(u \to v) = f(u \to v) - b(p, f)$

$f'(u \to v) \le f(u \to v) \le c(u \to v)$

$-b(p,f) \ge -f(u \to v)$ $\quad b(p, f) \le f(u \to v)$

$f'(u \to v) \ge f(u \to v) - f(u \to v) = 0$

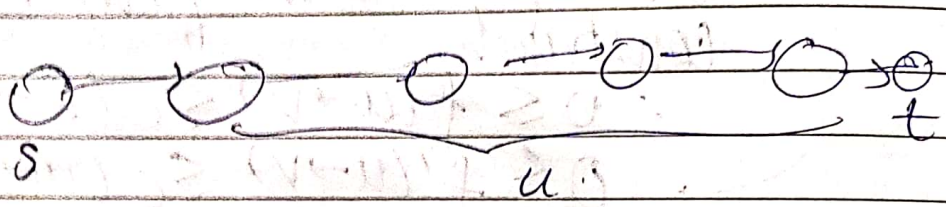
≥ 0.

edges.

capacity constraint proved

Conservation constraint

- we are not changing for vertices which is not part of P as flow is same as before
∴ conservation constraints met



∴ ∀

let $u \in P$

$u \notin \{s, t\}$

u has exactly one ~~in-neighbor~~ in-neighbor in P and exactly one out-neighbor in P.

case 1:- suppose edge going into u is forward edge

subcase 1:- suppose edge going out of u is forward edge (both fwd edge) ∈ P

Contradiction

In N/W flows source and sink are unique

$$f'^{out}(u) = \sum_{(u\to x)\in E(G)} f'(u\to x)$$

$f'(u\to x) = f(u\to x)$

$$= \sum_{\substack{(u\to x)\in E(G)\\ x\neq w}} f(u\to x) + f'(u\to w)$$

$$= \quad \| \quad + f(u\to w) + b(P,f)$$

$$= f^{out}(u) + b(P,f)$$

$$f'^{in}(u) = \sum_{\substack{(x\to u)\in E(G)\\ x\neq v}} f(x\to u) + f'(v\to u)$$

$f(x\to u) = f'(x\to u)$

$$= \quad \| \quad + f(v\to u) + b(P,f)$$

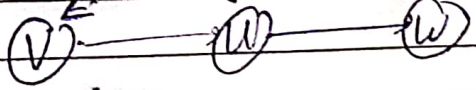
fwd edge.

$$= f^{IN}(u) + b(P,f)$$

$$f^{out}(u) = f^{IN}(u)$$

$$\Rightarrow f'^{out}(u) = f'^{IN}(u)$$

subcase B :- edge going out of u in P is backward edge.



$f'^{IN}(u) = f^{IN}(u)$

no incoming edges, both are outgoing edges

$$f'^{out}(u) = \sum_{\substack{(u\to x)\in E(G)\\ x\notin\{v,w\}}} f'(u\to x) + f'(u\to v) + f'(u\to w)$$

$$= \sum_{\substack{u \to x \\ \in E(G) \\ x \notin \{v,w\}}} f'(u \to x) + f(u \to v) - b(P,f) + b(P,f) - f(u \to w) + b(P,f)$$

$f'^{out}(u) = f^{out}(u)$.

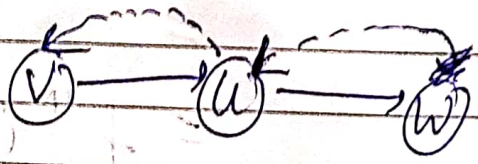
Case 2:- Suppose edge going inbou in P is backward edge.

Case A:- edge going out of u in P is forward.

same as case 1B

Case B:- edge going out of u in P is backward

same as case 1A.



~~$f'^{IN}(u) = f^{IN}(u)$~~ no incoming edge ∴ same as before.

$$f'^{IN}(w) = \sum_{\substack{(x \to u) \in E(G) \\ x \neq w}} f^{IN}(u) + f'(w \to u)$$

backward

$$= f^{IN}(u) + f(w \to u) - b(P,f)$$

$$= f^{IN}(w) - b(P,f)$$

$f'^{out}(w) = f^{out}(w) - b(P,f)$.

∴ conservation constraint

in every iteration flow is increasing (fractional).

classmate Date Page

claim:- Let f be a flow in G.
Let P be an s-t path in G_f
Then $v(f') = v(f) + b(P,f)$
and $v(f') > v(f)$.

$v(f)$ → value of flow f out of s.
$|v(f) = f^{out}(s)|$

Proof $v(f') = \sum_{(s\to x)\in E(G)} f'(s\to x)$

s - source
$\therefore s\to u$ has to be fwd edge

Let $(s\to u) \in P$.

$s\to u$ is a forward edge as s is a source

$f'(s\to u) = f(s\to u) + b(P,f)$

$b(P,f) > 0$

fwd edge.
$c(u\to v) - f(u\to v)$
$\geq$
strictly > 0
fwd edge
capacity is flow +ve

Running time

$C = \sum_{(s\to x)\in E(G)} c(s\to x)$

total capacity going out of source.

- capacities are integral.
- flow ~~integral~~ non-integral.

increasing by residual capacity [loop invariant]

If flow is integral, then

Running time $O(mC)$

termination depends on C, cannot be more than that.

10/10

Lemma 1 :- f' is a flow in each recursive call.

Lemma 2 :- $f'(s) > f(s)$.

Lemma 3 :- In each recursive call, f is integral and residual capacity of each edge in G_f is integral.

$$\#\text{recursive calls} \leq \sum_{e \text{ out of } s} c(e) = C$$

Running time is $O(Cm)$.

- so far it is not clear that the flow is max^m or not.

* s-t Cut — we define another problem for this

2 partitions.
disjoint

minimisation problem

$$cut(A,B) = \sum_{e \text{ out of } A} c(e).$$

Lemma 4 → let f be any flow in G and (A,B) be any s-t cut. Then $v(f) = f^{out}(A) - f^{IN}(A)$

value of flow

value of flow = flow going out of s

Proof $V(f) = f^{out}(s)$

$f^{IN}(s) = 0$

$V(f) = f^{out}(s) - f^{IN}(s)$.

Because f is a flow, flow coming into a node is same going out of it

$\therefore$ for all $v \in A$, $v \neq s$

$f^{out}(v) = f^{IN}(v)$, $f^{out}(v) - f^{IN}(v) = 0$

$V(f) = \sum_{v \in A} f^{out}(v) - f^{IN}(v)$

$= f^{out}(s)$.

$v \in S$ then diff $= 0$.
$v \notin S$ then $\neq f^{IN}$.

Let $(u \to v)$ be an edge s.t $u, v \in A$.

$f(u \to v)$ appears once as +ve term and once as -ve term in summation.

Let $(u \to v)$ be an edge s.t $u \in A$, $v \in B$.
so, $f(u \to v)$ will appear only once as +ve term in summation

Let $(u \to v)$ be an edge s.t $u \in B$, $v \in A$
so, $f(u \to v)$ will appear only once as -ve term in summation.

$$v(f) = f^{out}(A) - f^{IN}(A).$$

Lemma 5

Let f be any flow in G and (A, B) be any s-t cut

Then $v(f) \leq cut(A, B)$

Proof $v(f) = f^{out}(A) - f^{IN}(A)$

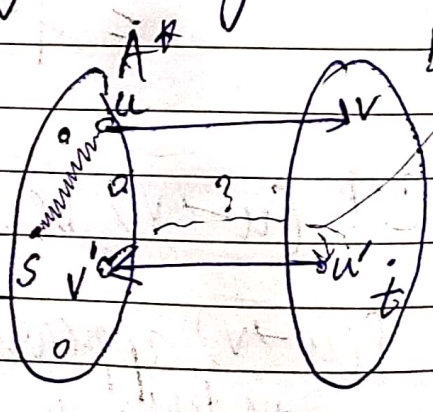
$\leq f^{out}(A)$ — $f^{IN}(A) \geq 0$ flow is never -ve

$\leq cut(A, B)$

Let f be a flow returned by our algm.

G_f does not have s-t path.

$A^* = \{ v \in V(G)$ such that s-v is a path in $G_f \}$



add s to A^*

$B^* = V(G) \setminus A^*$ — whatever not in A^* will be in B^*

$t \in B^*$

$\therefore (A^*, B^*)$ is a cut.

$$v(f) = f^{out}(A^*) - f^{IN}(A^*)$$

I didn't worry about edges inside A.

Let $u \to v$ be an edge in G s.t $u \in A^*$ and $v \in B^*$

s to u have path, then $u \to v$ edge, then $s \to v$ also edge but that defn is false

By defn of A^* and B^*, $u \to v$ is not ~~in~~ in G_f.

$f(u \to v) = c(u \to v)$. (flow = capacity)

(that is why you didn't add the forward edge.)

(we don't have this edge.
- as it is like adding a backward edge
- there was no flow on edge $u' \to v'$ edge.)

Let $u' \to v'$ be an edge in G s.t $u' \in B^*$, $v' \in A^*$
By defn of A^*, B^* we know that $v' \to u'$ is not in G_f

This implies that $f(u' \to v') = 0$.

$v(f) = f^{out}(A^*) - f^{IN}(A^*)$ capacity of e = 0. (using lemma)

$= \sum_{e \text{ out of } A^*} c(e) - \sum_{e \text{ into } A^*} f(e)$

$= \sum_{e \text{ out of } A^*} c(e)$.

$= cut(A^*, B^*)$

flow going out of $A \to \sum_{\text{edges out of } A} f(u \to v)$

$cut(A, B) \geq v(f)$.

$\to v(\bar{f}) = cut(A^*, B^*)$

$\therefore$ f greater than equal to cut always.

max flow min cut problem.

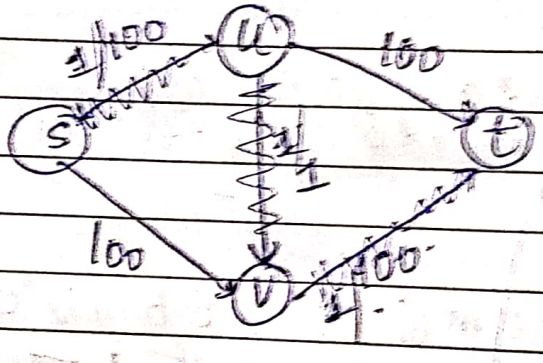
not poly now

12/10

Syllabus - N/w flows, NP-hardness, DP - for minors. - till here syllabus.

Network Flows -

$O(mC)$ → not a poly time algm.



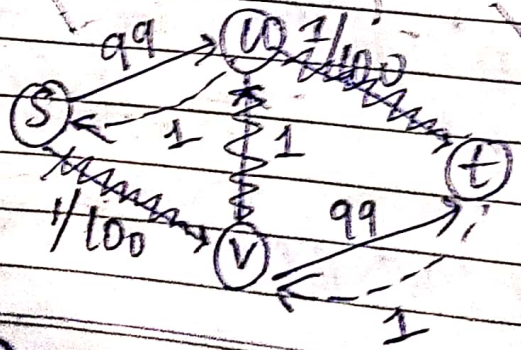
$f(s \to u) = 1$
$f(s \to v) = 1$
$f(u \to v) = 1$
$f(u \to t) = 1$
$f(v \to t) = 1$

→ augmenting is s-t path.

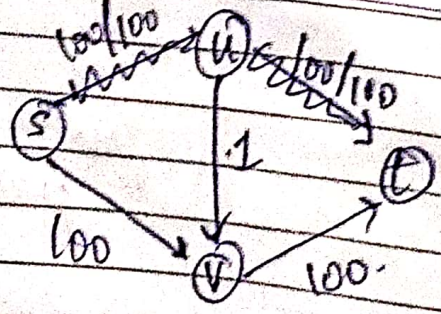
residual graph

initially 0 flow
bottleneck capacity = 1.
all forward edges capacity becomes 99 now

200 iterations.



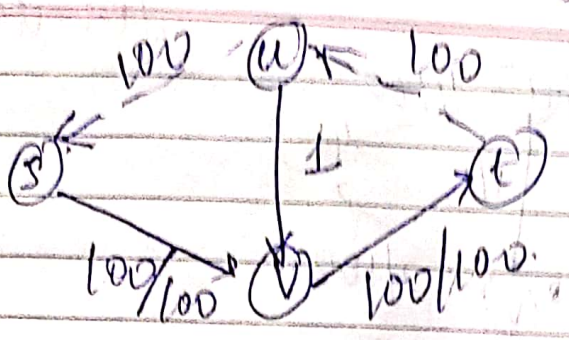
flow ↑ by 1 everytime



1st iteration of graph

bottleneck capacity → 100.

finding widest path
Dijkstra.



2
now 2 deviations only

now no more s-t path remains

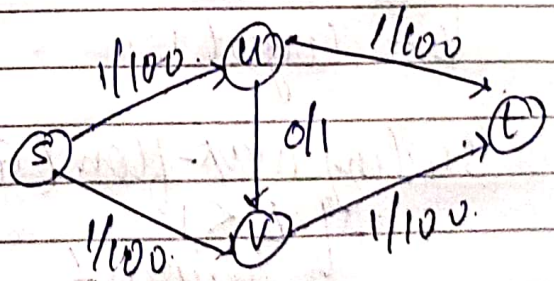
→ ∴ Our algm was slow because of our choice of s-t path. Therefore, choose path with the max^m bottleneck capacity. Then we could terminate possible. fastor. → no s-t path.

This could be expensive → does not mean exponential time

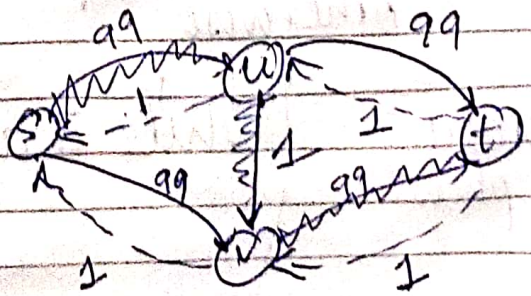
still poly-time only ✓

(read notes in dropbox)

→ Atleast, take big jumps. Try to increase atleast that much
threshold



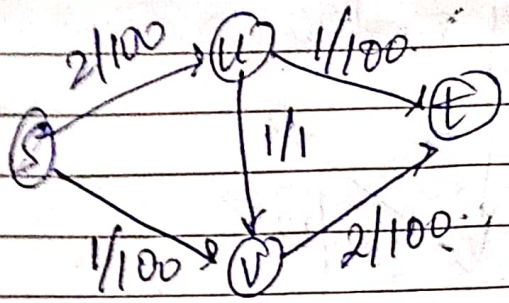
f(s→u) = 1
f(s→v) = 1
f(u→v) = 0
f(u→t) = 1
f(v→t) = 1



s-t path all forward edges.

poly in time of bits.

after iterations;



Δ - largest power of 2 that is smaller than largest capacity of edge incident on s.

Δ = 64.

- delete the lower capacity edges. ↓whose residual capacity is < 64. i.e Δ

∴ atleast Δ.

Δ → Δ/2 → Δ/4 ----

Algorithm:

f = 0

Δ - largest ".

scaling Max-flow (G, f, Δ).

If Δ < 1

return f.

otherwise

f' = f

while ∃ an s-t path in $G_f(\Delta)$

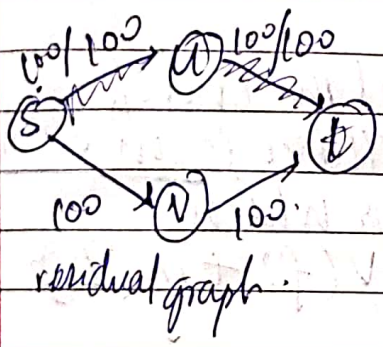
residual graph having edges ≥ Δ only.

keep edges with residual capacity at least Δ

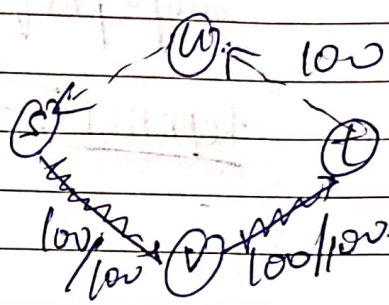
Δ - scaling phase

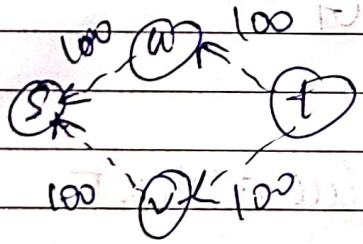
fwd edge → +1
bck edge → −1
consists of
Date
Page
process

Let P be an s-t path in $G_f(\Delta)$
augment (P, f) — increase flow by at least Δ
update f & $G_f(\Delta)$.
end
$\Delta = \Delta/2$
scaling Max-flow (G, f, Δ). — updated flow



$f(s \to u) = 0 + 100 = 100$
$f(s \to t) = 0 + 100 = 100$.
$f(s \to v) = 0$
$f(v \to t) = 0$.





now, no s-t path there.
$\therefore \Delta = \Delta/2$.
max flow ( ).
runs 5-6 times but while loop doesn't run!

when $\Delta = 1$; then $G_f = G_f(\Delta)$
then flow will be max^m

no of iterations = 200.
↓
1.
$= \log_2 2$.

$O(mC)$.
↓
now only $\log C$ iterations
earlier it was C !!
poly ✓

no change in correctness as such.
for $\Delta = 1$, $G_f(\Delta) = G_f$

Running Time

$\Delta \leq C = \sum_{e \text{ out of } s} c_e$.

claim :→ The number of recursive calls is atmost $\lceil \log_2 C \rceil$

simple proof - check for yourself

Lemma :- Let f be the flow at the end of Δ-scaling phase. There is an s-cut (A, B) in G for which $\text{cut}(A, B) \leq v(f) + m(\Delta)$

m - # of edges in G.

Consequently max flow has value atmost $v(f) + m\Delta$.

Lemma: The no. of augmentations in a scaling phase is $\leq 2m$.

Proof: First scaling phase, we augment on each edge coming out of s atmost once. So, m times.

Consider a later scaling phase Δ,

In the previous scaling phase for algm with $\Delta' = 2\Delta$

$f' \longrightarrow$ max flow

cap - flow $< \Delta$.
flow $< \Delta$

Let f_p be the flow returned in Δ's scaling phase.

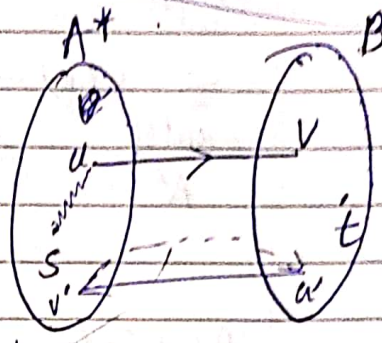
$$V(f^*) \leq V(f_p) + m2\Delta$$ current one.

each iteration ↑ by Δ.

proof left ;

Proof —

$A^* = \{ v \in V(G) :$ there is a s-v path in $G_f(\Delta) \}$



add s to A^*.

$B^* = V(G) \setminus A^*$

u, v such edge exists ∴ is a cut $s \in A^*$, $t \in B^*$

let $u \to v$ be an edge in G s.t $u \in A^*$ and $v \in B^*$ $u \to v \notin G_f(\Delta)$ otherwise there is a path from s to v in $G_f(\Delta)$, a contradiction.

Thus, $c(u \to v) - f(u \to v) < \Delta$ —

$f(u \to v) > c(u \to v) - \Delta$

$u' \in B^*$, $v' \in A^*$

$f(u' \to v') < \Delta$

$V(f) = f^{out}(A^*) - f^{IN}(A^*)$

$= \sum_{e \text{ out of } A^*} f(e) - \sum_{e \text{ into } A^*} f(e)$

$\geq \sum_{e \text{ out of } A^*} (c(e) - \Delta) - \sum_{e \text{ into } A^*} \Delta$

$= cut(A^*, B^*) - m\Delta$

$cut(A^*, B^*) \leq v(f) + m\Delta$.

hence proved.

$\sum_{e \text{ out of } A^*} (c(e) - \Delta) - \sum_{e \text{ out of } A^*} \Delta$.

$\sum_{e \text{ out of } A^*} c(e) - \sum_{e \text{ out of } A^*} \Delta - \sum_{e \text{ out of } A^*} \Delta = \sum_{e \text{ out of } A^*} c(e) - \Delta \sum_{\substack{\text{out of } A^* \text{ or} \\ \text{into } A^*}} 1$

$\leq m$.

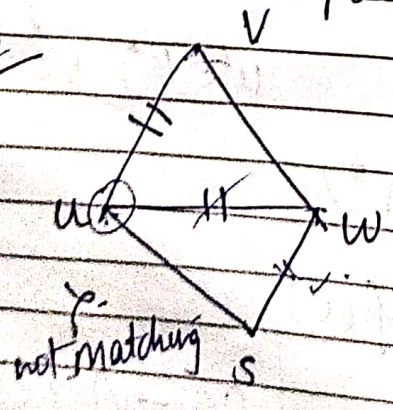
Applications of N/w flows.

① Matching

$G = (V, E)$ — undirected graph.

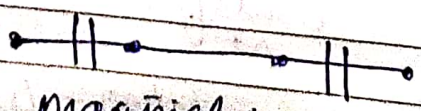
$M \subseteq E(G)$ is a matching if no two edges share an endpoint

eg



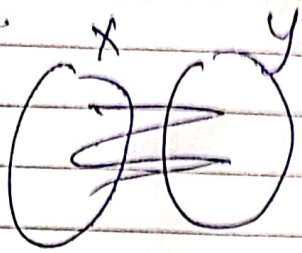
max^m matching

maximal matching

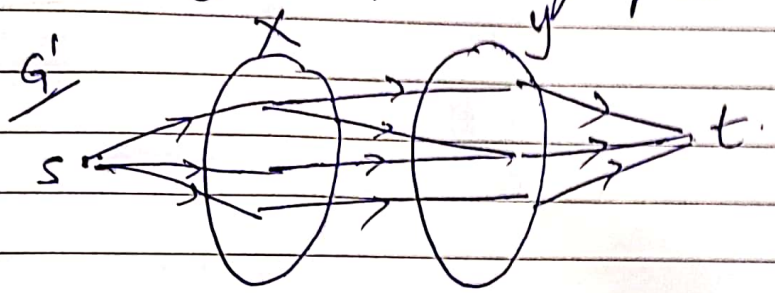


maximal & maximum matching

Bipartite graph.



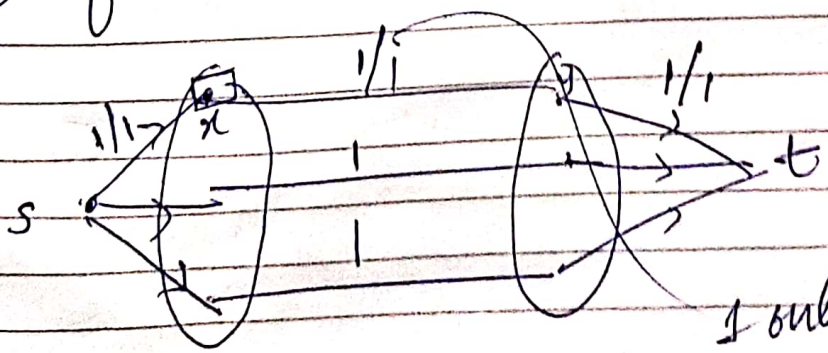
see matching in Bipartite graph using N/W flow



- whatever edge here.
- simply direct from $x \to y$ - every edge, no new edge added.
- capacity of all edges $\to 1$

Let $G = ((X, Y), E)$ be a bipartite graph construct G' as follows

1. $V(G') = V(G) \cup \{s, t\}$
2. if $xy \in E(G)$ s.t $x \in X, y \in Y$ add arc, $x \to y$
3. for every $x \in X$, add $s \to x$
4. for " $y \in Y$, add $y \to t$
5. for all $e \in E(G')$, $c(e) = 1$



integral flow

flow either 0 or 1

1 only if that edge is in ~~maximum~~ matching