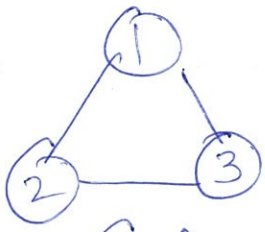
Dec 4

k-colorability

$G = (V, E)$

Def: A k-coloring of G is $c: V \to \{1, \ldots, k\}$ s.t $\forall (u,w) \in E$, $c(u) \neq c(w)$



Def: G is k-colorable if & only if $\exists$ a k-coloring for it.

Def: (k-coloring / colorability)

I/p: G, k

O/p: T if G is k-colorable
F if G is NOT k-colorable

Claim: k-colorability $\in NP$

THM: 3-SAT $\leq_p$ 3-coloring $\leq_p$ k-colorability ($k \geq 3$)

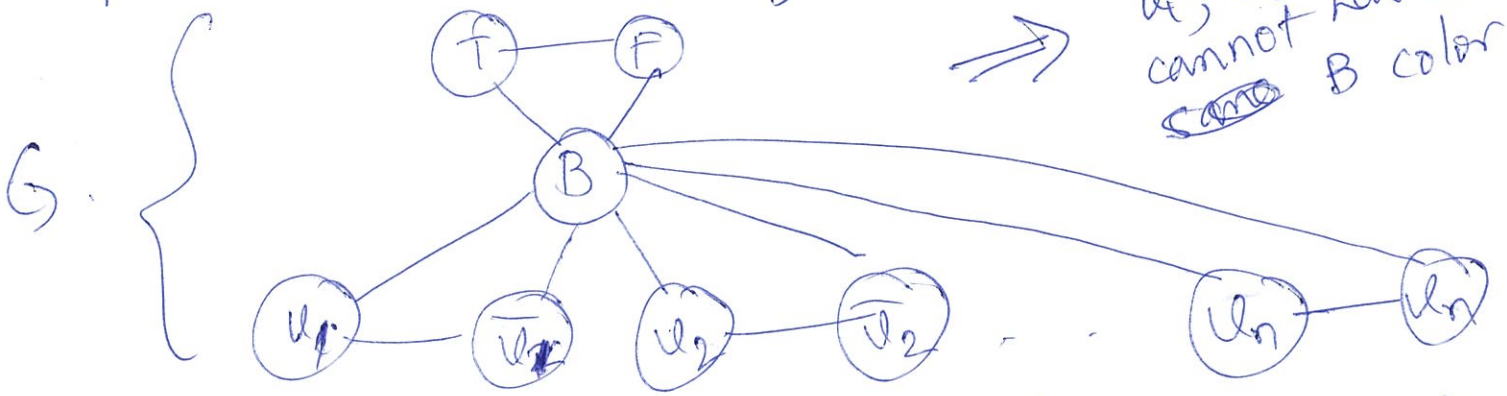
Goal: given any 3-SAT formula $C_1, \ldots, C_m$ or $X = \{x_1, \ldots, x_n\}$ compute a graph G (s.t & $|G| = \text{poly}(n,m)$) s.t G is 3-colorable $\iff$ $C_1, \ldots, C_m$ is satisfiable

Step 1: One node $v_i \equiv x_i$
$\overline{v_i} \equiv \overline{x_i}$

$\forall i$: 

$\Rightarrow$ v_i & $\overline{v_i}$ need to be assigned diff colors

3 special nodes: T, F, B



$\Rightarrow$ v_i, $\overline{v_i}$ cannot have B color

Claim: If $c(v_i) = c(T)$ $\Rightarrow c(\overline{x_i}) = c(F)$ | $c(v_i) = c(F)$ $\Rightarrow c(\overline{v_i}) = c(T)$

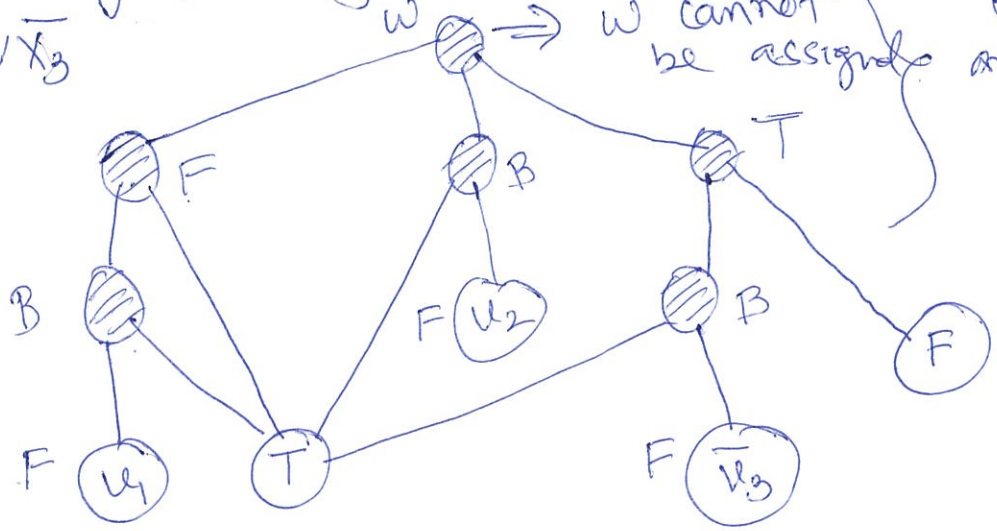
Claim: 3-coloring of G $\Leftrightarrow$ valid assignment to $x_1, \ldots, x_n$

→ Encode the clauses $C_1, \ldots, C_m$ in this graph (adding more vertices & edges in G).

eg. $x_1 \vee x_2 \vee \overline{x_3}$

(Find a gadget by trial & error)

$x_1 \vee x_2 \vee \overline{x_3}$ = C_i



w $\Rightarrow$ w cannot be assigned any of c(T), c(F) or c(B)

Gadget

Claim: 3-coloring of $Gad_i \Rightarrow$ at least one of the literals in Gad_i is assigned $c(T)$

Pf (idea): Assume v_1, v_2, $\bar{v}_3$ are all assigned $c(F)$

$\Rightarrow$ w cannot be assigned any of $c(T)$, $c(F)$ or $c(B)$

$\Rightarrow$ we do not have a valid 3-coloring!

Final redux: Given $C_1, \ldots, C_m$ on X

(1) Compute G from X

(2) Add Gad_i to G $\forall$ clauses C_i $\Rightarrow$ G'

(3) Output G' / Feed G' to an algo for 3-coloring.

Claim: $C_1, \ldots, C_m$ is satisfiable $\Leftrightarrow$ G' is 3-colorable.

Show: Problems that are harder to solve than NP-complete.

HALTING PROBLEM:

Input: Program P, input I for P — (P, I) a pair of strings.

Output: Yes if P terminates on I in any finite time

No o/w

Q: Does there $\exists$ an algo to solve the Halting problem

THM: NO!

Pf: By contradiction

Assume $\exists$ algo h s.t $h(P,I) = \begin{cases} \text{Yes} & \text{if } P(I) \text{ terminat} \\ \text{No} & \end{cases}$

```
def c(x):
    if h(x,x) = Yes:
        loop forever
    else
        return.
```

Consider the call c(c):

Case 1: $h(c,c) = \text{Yes} \Rightarrow$ c(c) loops forever

Case 2: $h(c,c) = \text{No} \Rightarrow$ c(c) terminates

$\Rightarrow$ contradicts h solving the halting problem