NP-Completeness

Story so far

For a wide range of Problems we have developed **Polynomial time** algorithms.

↓

On input of size n, their worst-case running time is $O(n^c)$ for some constant $c > 0$

Q: Whether **all Problems** can be solved in Polynomial time?

Ans: **No,**

For example **"Halting Problem"** Can not be Solved by any Computer, no matter how much time we allow.

There are also Problems that can be solved but not in time $O(n^k)$ for any constant k.

"Generally, we think of Problems that are Solvable in Polynomial time algorithms as being tractable, or easy and Problems that require Superpolynomial time as being intractable or hard."

Next we look at few graph theoretic Problems.

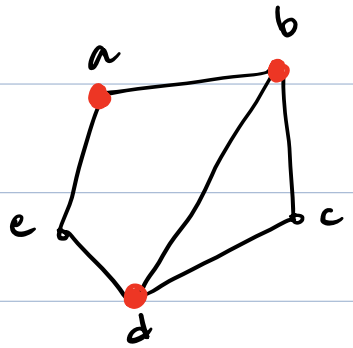
Graph theoretic Problems

Vertex cover:

A **Vertex cover** of a graph G is a set $X \subseteq V(G)$ that contains **at least one endpoint of every edge**.

Examples:

①



Ⓐ $\{a, b, d\}$ is a Vertex Cover of G.

Ⓑ $\{c, b, d\}$ is NOT a Vertex cover (edge ae is not covered)

Ⓒ $\{a, b, c, d, e\}$ is a trivial V.C

The Vertex cover problem is to find the Smallest Vertex Cover in a graph G.

The size of the **minimum size Vertex Cover** of G is denoted by $\beta(G)$. or $MVC(G)$.

VERTEX COVER (Decision version)

Input: A graph G and an integer k

Question: Does G contain a vertex cover of size at most k?

Brute-force Algorithm for V.C:

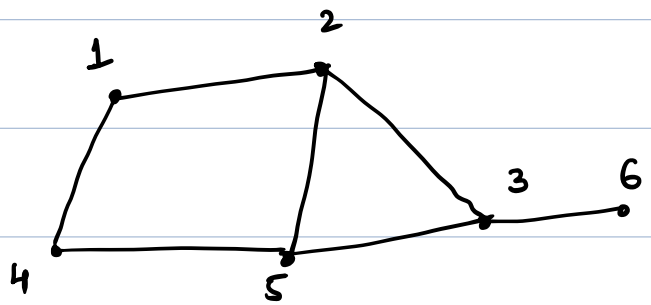
```
VC(G)
1.  MVC = |V(G)|
2.  For each X ⊆ V(G)
3.      For all e = uv ∈ E(G)
4.          if both u and v ∉ E(G)
5.              Return False
6.      if |X| < MVC
7.          MVC = |X|
8.  Return MVC
```

Running time : $O(2^n m)$

Independent set :

In a graph $G=(V,E)$, we say subset $I \subseteq V$ is independent if no two vertices in I are adjacent.

The independent set problem is to find the largest independent set in a graph G.



The largest size independent set $= \{2, 4, 6\}$

Size $= 3$.

The size of a maximum independent set in G is denoted by $\alpha(G)$.

For the above example $\alpha(G) = 3$.

Independent set (Decision version)

Input: A graph G and an integer k

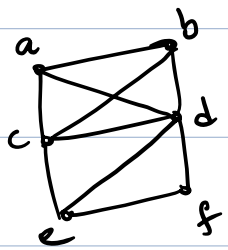
Question: Does G contain an independent set of size at least k?

CLIQUE in a graph:

A subset $C \subseteq V(G)$ is called a clique in a graph G if for all $u, v \in C$, $uv \in E(G)$ (i.e., C is a complete subgraph of G).

The CLIQUE problem is to find the size of a largest clique in a given graph (optimization version).

Ex:-



$C = \{a, b, c, d\}$ is a clique of size 4 (also largest)

The size of a largest clique in a graph G is denoted by $\omega(G)$. (also called clique number)

For the above example $\omega(G) = 4$

CLIQUE Problem [Decision version]

Input: Graph G, and an integer k

Question: Decide whether G has a clique of size k?

Back to Vertex cover, Clique and Independent set Problems.

For many of the fundamental Problems,

we do not know of Polynomial-time algorithms for

these Problems (even today).

We cannot Prove that no Polynomial-time algorithm

exists.

The good thing is :

Researchers working in this area have found a large class of problems with the following property.

"A Polynomial time algorithm for any one of them would imply the existence of a Polynomial time algorithm for all of them"

[These are NP-Complete Problems, formally we define it later]

Complexity classes P and NP

P = Set of all decision Problems that can be solved in Polynomial time.

(decision → YES/NO type questions)

Eg: Sorting, Searching, MST etc

NP = Set of all decision Problems for which a solution can be verified in Polynomial time.

Eg: Vertex Cover, Hamiltonian cycle, SAT etc

Clearly $P \subseteq NP$

(Q) Is P equal to NP? (Unsolved Problem)

belief: $P \neq NP$

Means there are Problems in NP that are harder to Compute than to verify: they could not be solved in Polynomial time, but the answer could be verified in Polynomial time.

Decision Problems vs Optimization Problems

Many Problems of interest are optimization Problems in which each feasible Solution has an associated value and we wish to find a feasible Solution with the best value.

Eg: Given a graph G and two vertices S and t find the Shortest S-t Path in G.

Decision Problem is a Problem that can be Posted as YES/NO question of the input values.

Eg: ① Given a graph G, $k \in \mathbb{Z}^+$ & two vertices S and t find the Shortest S-t Path in G.

Eg: ② Given two numbers P and q, does P divides q?

For Convenience we work with decision Problems (that have yes/no answer only).

Examples of Problems in NP

① Vertex Cover is in NP

VERTEX COVER

Input: A graph G, an integer k and a subset $X \subseteq V(G)$

Question: Check whether the X is a VertexCover of size k ?

(G, X, k)

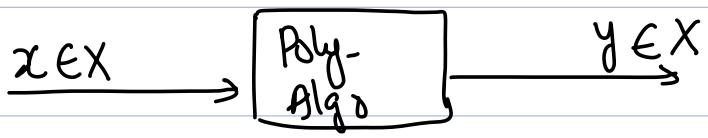
```
if |X| > K
    return NO
else
    For each vertex v ∈ X
        remove all the edges adjacent to v
if G has no edges
    return TRUE
else
    return FALSE
```

Algorithm runs in Polynomial time, $O(n+m)$

Similarly we can show that Independset and Clique Problems are in NP. (Exercise)

Polynomial-time reductions

We say that a problem X is polynomial-time reducible to a problem Y, denoted $X \leq_p Y$ if there exists a polynomial time algorithm such that given any instance x of X, the algorithm constructs an instance y of Y such that x is a YES instance of X iff y is a YES instance of Y.



Examples:

① Vertex Cover $\leq_p$ Independent Set

② Independent set $\leq_p$ Clique

We don't know how to solve either Ind set or Vertex Cover in Polynomial time.

We show that they are equally hard.

Lemma: let $G = (V, E)$ be a graph. Then I is an independent set if and only if $V - I$ is a Vertex Cover.

Proof: ($\Rightarrow$)

Suppose I is an Ind. set. Consider any edge $e = uv$. Since I is an Ind-set it cannot be the case that both u and v are in I. So one of them must be in $V - I$. That is every edge has at least one end in $V - S$. hence $V - S$ is a vedexcover.

($\Leftarrow$) Suppose that $V - I$ is a vertexcover.

Consider any two vertices u and v in I.

If they are adjacent then neither u nor v in $V - I$

Contradicting our assumption $V - I$ is a vertex Cover.

$\therefore$ I is an Ind. set.

Lemma: Independent set $\leq_p$ Vertex cover

Proof: Given an instance (G, k) of Independent set, we construct an instance (H, k') of Vertex cover.

where $H = G$, $k' = n - k$

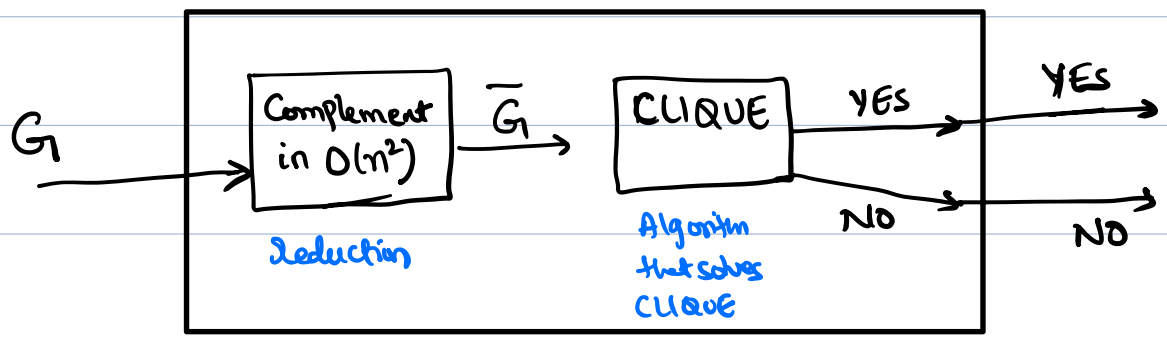
From the above lemma we have

(G, k) is an YES instance of Ind-set $\Leftrightarrow$

(H, k') is an YES " " Vertex cover.

Eg ②

IND SET $\leq_p$ CLIQUE



Set Cover :

Given a set U of n elements, a collection $S_1, S_2, \ldots S_m$ of subsets of U and number k, does there exist a collection of at most k of these sets whose union is equal to all of U ?

Ex: $U = \{1, 2, 3, 4, 5, 6\}$

$S_1 = \{1, 3, 5\}$, $S_2 = \{2, 3, 4\}$, $S_3 = \{2, 4, 6\}$,

$k = 2$

Then $S_1 \cup S_3 = U$, hence it is a YES instance.

Vertex Cover $\leq_p$ Set Cover

Given an instance (G, k) of Vertex Cover.

We will construct an instance of the Set Cover Problem.

Let $U = E(G)$ and let S_i be the set of edges that incident to vertex i.

Clearly $S_i \subseteq U$ for all i.

Also, this construction can be done in time that is Polynomial in the size of the Vertex Cover instance.

We show that the original instance of Vertex Cover is a YES instance if and only if the Set Cover instance we created is also a YES instance.

($\Rightarrow$)

Suppose G has a VertexCover of size at most k.

Let S be a Vertex Cover of size at most k.

define $T = \bigcup_{i \in S} S_i$

Claim: $T = U$ and # of sets in T are $\leq k$

↓ Easy as $|S| \leq k$

Consider any element of U.

Such an element is an edge e in G and since S is a vertex cover for G, at least one of e's endpoints is in S.

Therefore T contains at least one of the sets associated with the endpoints of e and by defn these both contain e.

$C = \{S_{i_1}, S_{i_2} .. S_{i_k}\}$

(⇐) Suppose there is a set cover of size k.

Since each set in C is associated with a vertex in G, let S be the set of these vertices.

Clearly $|S| \leq k$.

Consider any edge e.

as C is a setCover C must contain at least one set that includes e.

Thus, S must contain at least one of the end points of e. $\therefore$ S is a vertexCover of G.

The satisfiability Problem (SAT)

An instance of SAT is a boolean formula ϕ composed of

(a) n boolean variables: $x_1, x_2, \ldots x_n$

(b) m boolean connectives; such as AND ($\wedge$), OR ($\vee$), NOT ($-$), IMPLICATION ($\rightarrow$) if and only if ($\leftrightarrow$).

(c) Parantheses.

Eg $\phi = ((x_1 \vee x_2) \wedge (\bar{x}_3 \vee x_4) \vee \bar{x}_5) \wedge \bar{x}_1$

A truth assignment for a boolean formula ϕ is a set of values for the variables of ϕ and a satisfying assignment is a truth assignment that causes it to evaluate to 1.

A formula with a satisfying assignment is a satisfiable formula.

Satisfiability Problem

I/P: A Boolean formula ϕ

Q: Is ϕ satisfiable?

$$SAT = \{ \phi : \phi \text{ is a satisfiable boolean formula} \}$$

Eg: $\phi = (x_1 \vee x_2) \wedge \overline{x_2}$

has the satisfying assignment $x_1 = 1, x_2 = 0$.

$(1 \vee 0) \wedge 1$

$= 1$

A literal is a either a Propositional Variable or the negation of a Propositional Variable.

Negated Variables such as $\neg x$, $\neg y$ are called negative literals.

Some times we write $\bar{x}$, $\bar{y}$

A Clause is a disjunction (OR) of one or more literals.

Eg: $(x \vee \bar{y} \vee z)$, $(\bar{x} \vee \bar{z})$

A Boolean formula is in conjuctive normal form or CNF, if it is expressed as an AND of clauses, each of which is the OR of one or more literals.

A Boolean formula is in 3-CNF if each clause has exactly three distinct literals.

Eg: $\phi = (x_1 \vee x_2 \vee x_3) \wedge (x_1 \vee \bar{x}_1 \vee \bar{x}_2) \wedge (\bar{x}_1 \vee \bar{x}_3 \vee \bar{x}_4)$

3-CNF-SAT $\leq_p$ CLIQUE

Let $\phi = C_1 \wedge C_2 \ldots \wedge C_k$ be a boolean formula in 3-CNF with k clauses.

For $r = 1, 2, \ldots k$, each clause C_r has exactly three distinct literals l_1^r, l_2^r, l_3^r.

We shall construct a graph G such that ϕ is satisfiable iff G has a clique of size k.

Graph Construction: $G = (V, E)$

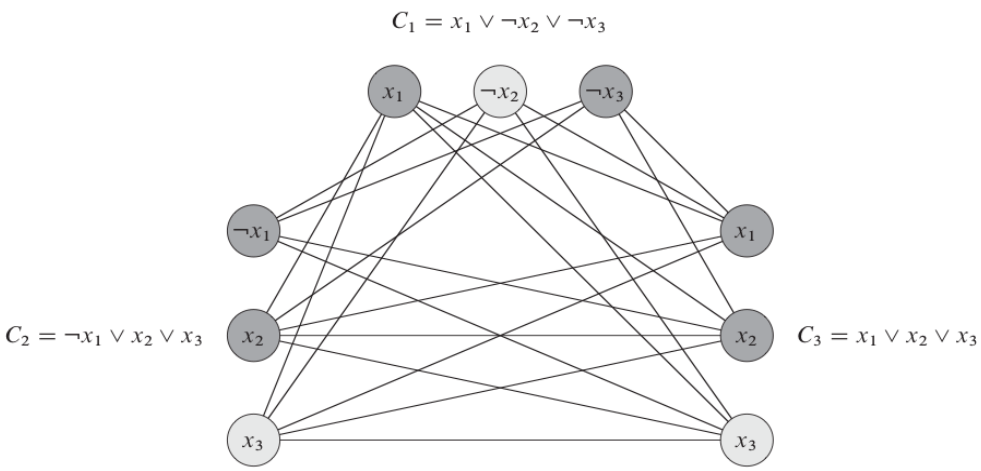
Vertex set: For each clause $C_r = (l_1^r \vee l_2^r \vee l_3^r)$ in ϕ, we add three vertices v_1^r, v_2^r, v_3^r into V.

Edges: We put an edge b/w two vertices v_i^r & v_j^s if

- v_i^r & v_j^s are in different triples, ie, $r \neq s$ &
- Their corresponding literals are consistent ie, l_i^r is not the negation of l_j^s.

We can build G from ϕ in polynomial time.

Eg $\phi = (x_1 \vee \overline{x}_2 \vee \overline{x}_3) \wedge (\overline{x}_1 \vee x_2 \vee x_3) \wedge (x_1 \vee x_2 \vee x_3)$



Forward:

Suppose that ϕ has a satisfying assignment. then each clause C_r contains at least one literal l_i^r that is assigned 1 and such literal corresponds to a vertex v_i^r.

Picking one such "true" literal from each clause yields a set V' of k vertices.

we prove that V' is a clique.

For any two vertices $v_i^r, v_j^s \in V'$, $r \neq s$ both l_i^r, l_j^s map to 1 by the given satisfying assignment and thus the literals cannot be complements. Thus by the construction of G, the edge $v_i^r v_j^s \in E$.

Reverse:

Suppose that G has a clique V' of size k. No edges in G connect vertices in the same triple and so V' contains exactly one vertex per triple.

We can assign 1 to each literal l_i^r such that $v_i^r \in V'$.

Since G doesn't contain no edges b/w inconsistent literals, we will not assign 1 to each literal & its complement.

∴ Each clause satisfied and so ϕ is satisfied.

NP-Complete :- A Problem X is NP-Complete if

① X is in NP

② $Y \leq_p X$ for every Problem $Y \in NP$

A Problem X satisfies Property ② but not necessarily Property ① we say that X is NP-hard

ie, A Problem X is NPC

if $X \in NP$ and X is NP-hard.

First NP-Complete Problem

SAT is the first Problem that was Proven to be NP-Complete (See Cook-Levin theorem)

- There is no known algorithm that efficiently solves each SAT Problem and it is generally belived that no such algorithm exist, yet this belief has not been proven mathematically, and resolving the question of whether SAT has a Polynomial-time algorithm is equivalent to the P versus NP Problem, which is a famous open Problem in theory of Computing.

- NP-Completeness only refers to the run-time of the worst case instances. Many of the instances that occur in Practical applications can be solved much more quickly.

Lemma: Suppose $Y \leq_p X$. If Y cannot be solved in Polynomial time, then X cannot be solved in Polynomial time.

Proof: Suppose X can be solved in Polynomial time

Since $Y \leq_p X$,

Y can also be solved in Polynomial time.

Lemma: Suppose X is an NP-Complete Problem.

Then X is Solvable in Polynomial time if and only if $P = NP$.

Proof Reverse Direction:

if $P = NP$, as $X \in NP \Rightarrow X \in P$.

forward:

X is NPC & $X \in P$

ie, $\forall Z \in NP$, $Z \leq_p X$

then $Z \in P$

$\Rightarrow NP \subseteq P$

$\Rightarrow P = NP$

Lemma: If Y is an NP-Complete Problem and X is a Problem in NP with the Property that $Y \leq_p X$, then X is NP-Complete.

Proof: We need to show: X is NP-Complete $\Leftarrow$ $X \in NP$ (given) (✓) and $\forall Z \in NP,\ Z \leq_p X$

Let Z be any Problem in NP

we have $Z \leq_p Y$, ($\because$ Y is NPC)

we know $Y \leq_p X$

$\therefore$ $Z \leq_p X$.

—x—

We know 3-SAT $\leq_p$ CLIQUE & 3-SAT is NP-Complete

So from the above lemma CLIQUE is NP-Complete.

Similarly we can show that VERTEX COVER and INDSET are NP-Complete.

General Strategy for Proving New Problem NP-Complete

Given a new Problem X, here is the basic Strategy for Proving it is NP-Complete

1. Prove that $X \in NP$
2. Choose a Problem Y that is known to be NP-complete.
3. Prove that $Y \leq_p X$.

Graph coloring

(Also called as Proper coloring)

Formal Definition:

Given a Graph G, the GRAPH COLORING Problem is to color the vertices of the graph such that adjacent vertices have different colors.

"The minimum number of colors need to color a graph G is called Chromatic number denoted $\chi(G)$."

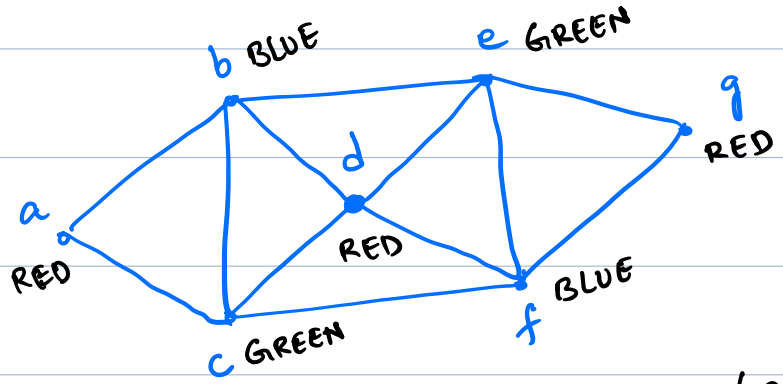
For $k \in \mathbb{Z}^+$, a graph G is k-colorable iff $\chi(G) \leq k$.

ie, $\exists f : V(G) \to \{1, 2, \ldots k\}$ such that

$f(u) \neq f(v)$ for all $uv \in E(G)$.

(f is also called a Proper coloring of G)

Eg ①

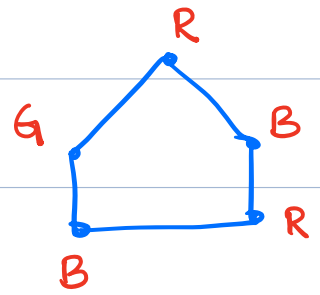


Graph G.

$\chi(G) = 3.$

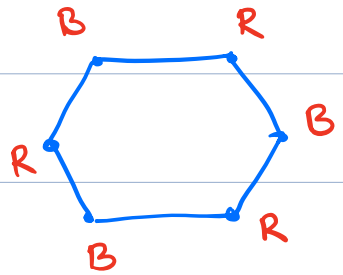
Examples

① C_5 (cycle) =



$\chi(C_5) = 3$

② C_6 (cycle) =



$\chi(C_6) = 2$

In general $\chi(C_{even}) = 2$ and

$\chi(C_{odd}) = 3$

③ Complete Graph (K_n): $\chi(K_n) = n$

k-coloring (or) Proper k-coloring Problem

I/P: A graph G

Q: Does G have a k-coloring?

We show that 2-coloring can be solved in polynomial time, however k-coloring for $k \geq 3$ is NP-complete.

Lemma: A graph G is bipartite iff it is 2-colorable.

Proof ($\Rightarrow$) $G = (A \cup B, E)$ is bipartite

define $f: V(G) \to \{1, 2\}$ as

$$f(v) = \begin{cases} 1 & \text{if } v \in A \\ 2 & \text{if } v \in B \end{cases}$$

Clearly f is a 2-coloring of G.

($\Leftarrow$) if G is a 2-colorable graph.

let $f: V(G) \to \{1, 2\}$ be a 2-coloring of G

let $A = \{ v \mid f(v) = 1 \}$

$B = \{ v \mid f(v) = 2 \}$

Clearly A and B are independent sets

hence $G = (A \cup B, E)$ is bipartite.

3-coloring Problem

Input: A graph G.

Q: Does G have 3-coloring?

Theorem: 3-coloring is NP-Complete.

Proof:

[I] 3-coloring is in NP.

Given a graph G and 3-coloring $f: V(G) \to \{1,2,3\}$, we can verify in Polynomial time that at most 3 colors are used and that no pair of vertices joined by an edge receive the same color.

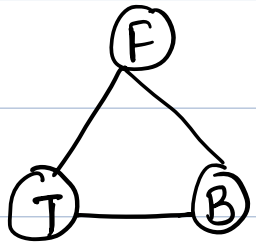
Next we show

[II] 3-SAT $\leq_p$ 3-coloring

On input ϕ, Construct G_ϕ as follows.

- Add 3 special nodes

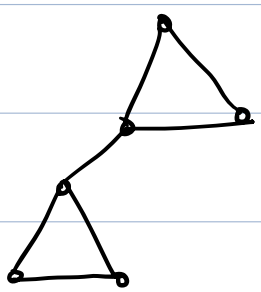
  T = "true", F = "false", B = "base"



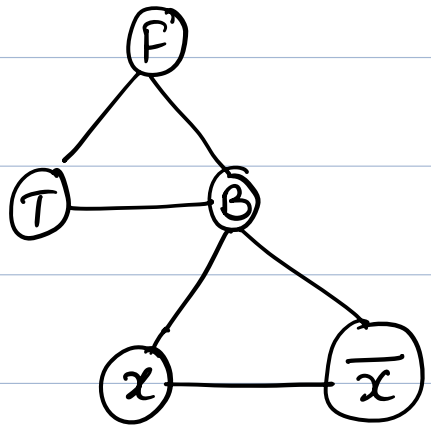
- For each variable

  add 2 literal nodes

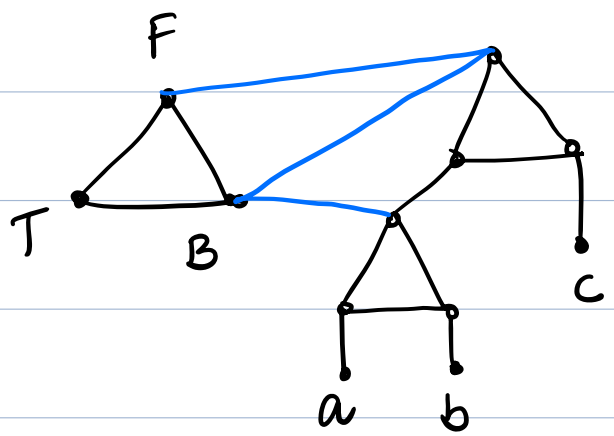
- For each clause add 6 clause nodes



- For each variable x connect

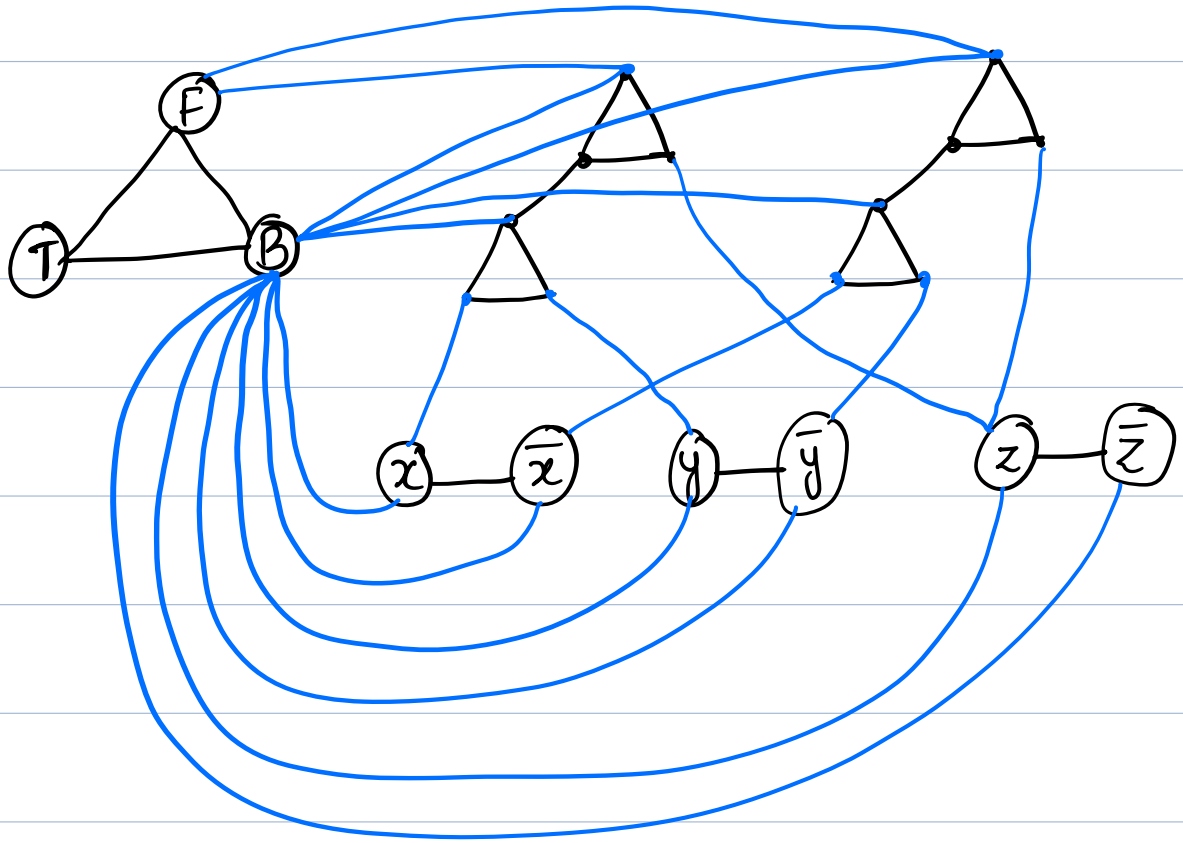


- For each clause $(a \vee b \vee c)$ connect



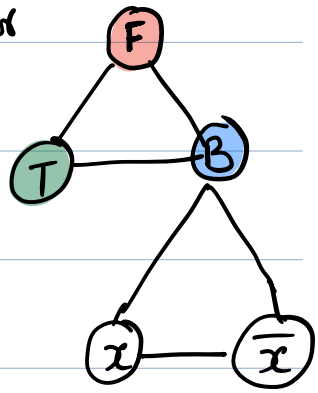
Clause - gadget

Example: $\phi = (x \vee y \vee z) \wedge (\bar{x} \vee \bar{y} \vee z)$



Some Remarks

In any 3-coloring of G_1, the nodes True, False and Base must get all three colors in some permutation. Thus we refer to the 3 colors as the True color, False color and the Base color based on which of these nodes gets which color.



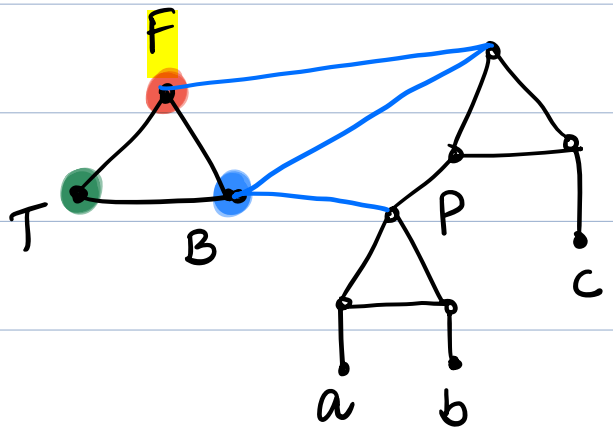
- In any 3-coloring, all variable nodes must be colored T or F, because they are adjacent to B.

Also x and $\bar{x}$ must have different colors as $x\bar{x} \in E(G_\phi)$.

- So we can convert a 3-coloring of G_ϕ into a true/false assignment to variables of ϕ.

Clause-gadgets

Fact: Graph below 3-colorable $\Leftrightarrow$ one of a, b, c colored T (ie, green)



Proof by contradiction

Suppose that a, b, c and all colored F then

P is colored F

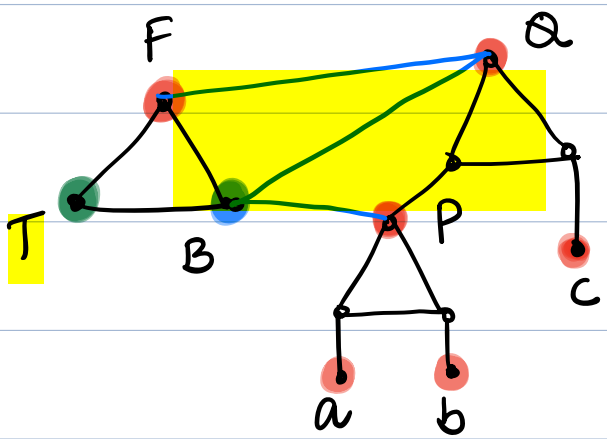
and Q is colored F

then this not a

valid 3-coloring

as F and Q adjacent

with same color.

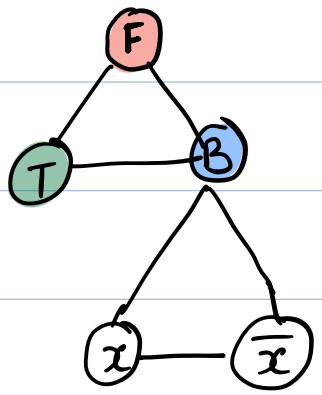


Claim: ϕ is Satisfiable $\Leftrightarrow$ G_ϕ is 3-colorable

Forward:

Suppose ϕ has a satisfying assignment

color literal nodes T or F accordingly



For each i, assign x_i the true color if $x_i = 1$ and False color if $x_i = 0$ and then $\overline{x_i}$ is assigned to only available color

As argued above, it is now possible to extend this 3-coloring into clause gadgets resulting in a 3-coloring of G_ϕ.

Reverse:

Suppose G_ϕ has a 3-coloring,

In this 3-coloring, each node x_i is assigned either True color or False color.

then we set the variable x_i in ϕ correspondigly.

Claim: In Each clause of the 3-SAT instance, at least one of the terms in the clause has truth value 1.

If not, then all three of the corresponding nodes has the False color in the 3-coloring of G_ϕ and we have seen above there is no 3-coloring of the corresponding clause gadget consistent with this, which is a contradiction.

– 3-Coloring $\leq_p$ k-coloring ($k > 3$)

Given an instance G of 3-Coloring add k-3 new nodes, and join these new nodes to each other and to every node in G.

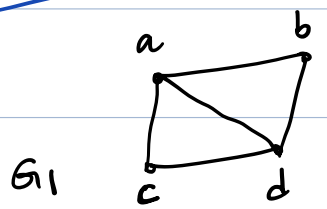
It is easy to see that the resulting graph is k-colorable iff the original graph G is 3-colorable.

Thus k-Coloring for any $k \geq 3$ is NP-Complete.

Def :- A **Simple Path (cycle)** in a graph containing all its vertices is **called a HAMILTON PATH** (respt. HAMILTON CYCLE).

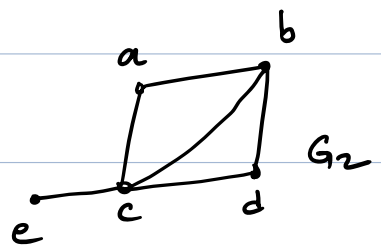
A graph is called a Hamiltonian if it contains a **Hamilton cycle**.

Examples :



$C = a\ b\ d\ c\ a$ is a Hamiton cycle

$\therefore$ G_1 is Hamiltonian



G_2 is **NOT** Hamiltonian

But G_2 has a **Hamilton Path**

$P = a\ b\ d\ c\ e$

Hamiltonian Cycle Problem

Given a graph G, check whether G is Hamiltonian or not.

Theorem:

The Hamiltonian Cycle Problem is NP-Complete.

Proof: Refer Theorem 34.13 on Page 1091 in Coreman et al. text book.

Theorem:

The Hamiltonian PATH Problem is NP-Complete.

Proof:

H.Cycle $\leq_p$ H.Path

Given an instance $G=(V,E)$ of H.cycle Problem, we will construct an instance $H=(V,E)$ of H.Path Problem as follows.

let $v \in V(G)$ be a vertex of G, and

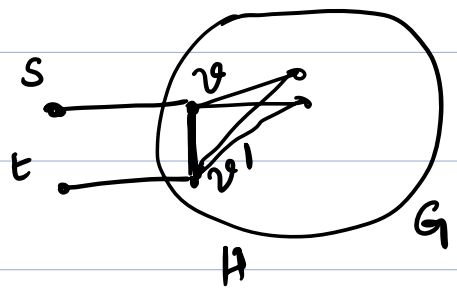
Add three new vertices v', s, t, where v' is a copy of v.

ie, $V(H) = V(G) \cup \{v', s, t\}$

$E(H) = E(G) \cup \{v'w \mid vw \in E(G)\} \cup \{sv, tv', vv'\}$

(both v & v' have same neighbors)

Clearly H can be constructed in Polynomial time.



Claim: G has a H-cycle $\iff$ H has a H-Path

($\Rightarrow$) let $C = v, \ldots u, v$ be a H-cycle then

$P = s, v, \ldots u, v', t$ is a H-Path in H.

($\Leftarrow$) let $P = s, v, \ldots u\, v', t$ be a H-Path in H

(Note that P can start/end at only at s & t)

then $C = v, \ldots u, v$ is H-cycle in G.

Longest Path Problem

Longest Path Problem is the Problem of finding a Simple Path of maximum length in a given graph.

↓

doesn't have any repeated vertices.

" In contrast to the Shortest Path Problem, which can be solved in Polynomial time in graphs without negative-weight cycles, the longest Path Problem (decision version) is NP-Complete"

↓

Input : A graph $G=(V,E)$, an integer k.

Question : Is there a Path of length at least k in G?

This Problem is NP-Complete, as there is an obvious reduction from Hamiltonian Path Problem.

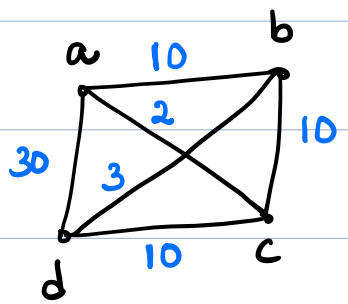
Travelling Salesman Problem

We are given a undirected complete graph G that has a non-negative integer cost C_{uv} associated with each edge $uv \in E(G)$, and we must find a hamiltonian cycle (a tour) of G with minimum cost.

For $A \subseteq E(G)$

$$c(A) = \sum_{uv \in A} C_{uv}$$

Example:



Hamiltonian cycle of minimum cost is

$a - b - d - c - a$ of cost $10 + 3 + 10 + 2 = 25$

Decision Version of TSP

I/P: Complete graph G, $c : V \times V \to \mathbb{Z}$, $k \in \mathbb{Z}$

Q: Does G has a Hamiltonian cycle with cost at most k.

Remark: TSP is NP-Complete

Proof Idea: Hamiltonian cycle $\leq_p$ TSP

Let $G = (V, E)$ be an instance of Hamiltonian cycle Problem. We construct an instance of TSP as follows.

We form a Complete graph $G' = (V, E')$, where $E' = \{ uv \mid u \in V, v \in V \}$. Cost function c is defined as

$$c_{uv} = \begin{cases} 0 & \text{if } uv \in E \\ 1 & \text{otherwise} \end{cases}$$

The instance of TSP is then $(G', c, 0)$ which can be constructed in Polynomial time.

Claim: The graph G has a hamiltonian cycle if and only if graph G' has tour of cost at most zero.

Proof: Exercise

We will revisit TSP in approximation algorithms Part.

Continuing along these lines we can show that many problems are NP-Complete, for example

Hamiltonian cycle/path

Travelling Salesman

Subset Sum

Setcover

Hitting set

- - - - -

Summary

No polynomial time algorithm is known for any NP-Complete Problem. If a polynomial time algorithm were found for any one of the NP-Complete Problem, we would able to construct polynomial time algorithms for all of them.

Researchers belive that none of the NP-Complete Problems can be solved in polynomial time. However this has not proven mathematically.