COL 351:
Analysis and Design of Algorithms

Lecture 34

All Problems covered till now

- Have polynomial time solutions
  - Example:
    Max-flows, MST, Closest-pair, Matrix-multiplication, Sorting, etc.

There are large class of problems for which

- No polynomial time (eg. $O(n^c)$) solution is known till now:
  - Example: Vertex-cover, Independent-set, Longest-Path, etc.
- They have no known "super-polynomial" lower bound on running-time.
- They all are ALL <u>closely related to each other.</u>

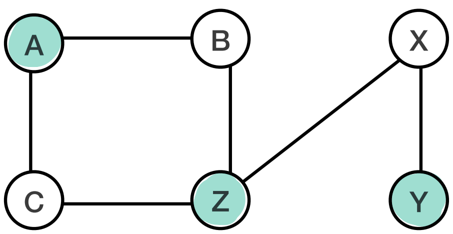
Vertex Cover and Dominating Set

Vertex Cover

Given: A graph $G = (V, E)$ with n vertices.

Def: A subset $W \subseteq V$ such that for each edge $(a, b) \in E$, either a or b lies in W.

Decision Version:
Find if there is a vertex-cover of size $\leq k$.



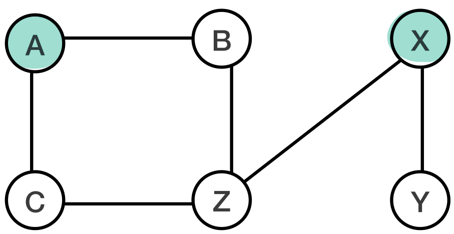
(G, 1) — NO
(G, 2) — NO
(G, 3) — YES

Dominating Set

Given: A graph $G = (V, E)$ with n vertices.

Def: A subset $D \subseteq V$ such that for each $v \notin D$, a neighbour of v lies in set "D".

Decision Version:
Find if there is a dominating-set of size $\leq k$.

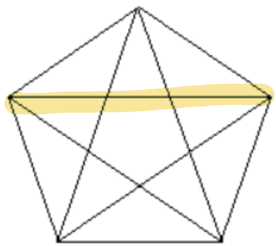


(G, 1) — NO
(G, 2) — YES

More Examples: Graph Instances

Class: Vertex Cover

G



Instances

- $(G, 1)$ — NO
- $(G, 2)$ — NO
- $(G, 3)$ — NO
- $(G, 4)$ — YES
- $(G, 5)$ — YES

If $G = K_n$, then

$(G, n-2)$ — NO

$(G, n-1)$ — YES

Vertex Cover

Given: A graph $G = (V, E)$ with n vertices.

Def: A subset $W \subseteq V$ such that for each edge $(a, b) \in E$, either a or b lies in W.

Decision Version:
Find if there is a vertex-cover of size $\leq k$.

Dominating Set

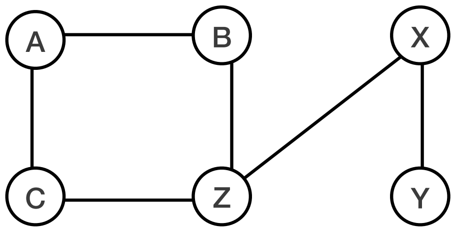
Given: A graph $G = (V, E)$ with n vertices.

Def: A subset $D \subseteq V$ such that for each $v \notin D$, a neighbour of v lies in set "D".

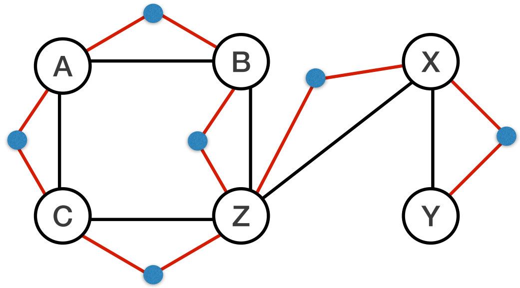
Decision Version:
Find if there is a dominating-set of size $\leq k$.

Which problem is easier to solve?

Lemma: If dominating-set can be solved in polynomial time, then the vertex-cover problem also has a poly-time algorithm.

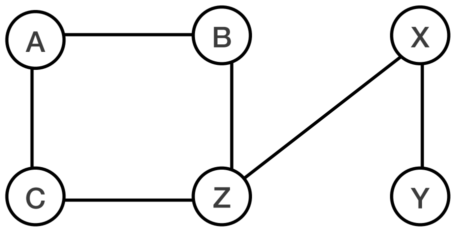


(G, k), an instance of VC

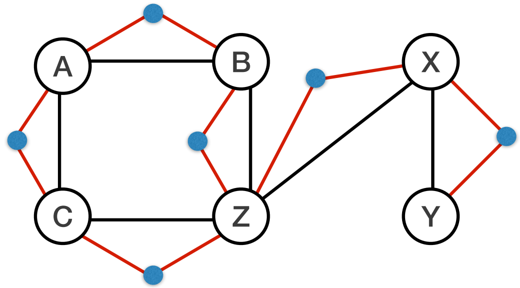


(H, k), an instance of DS

Lemma: If dominating-set can be solved in polynomial time, then the vertex-cover problem also has a poly-time algorithm.



(G, k), an instance of VC



(H, k), an instance of DS

Proof: Let (X_V, X_E) be a partition of vertices in H, where $X_V = V(G)$, and X_E correspond to $E(G)$.

(G, k) is Yes $\Rightarrow$ (H, k) is Yes

- Let W be a vertex-cover of G of size at most k.
- Then, W is a dominating-set of G.
- Morover, each vertex in X_E is adjacent to a vertex in W.
- So, W is a dominating-set of H.

(H, k) is Yes $\Rightarrow$ (G, k) is Yes

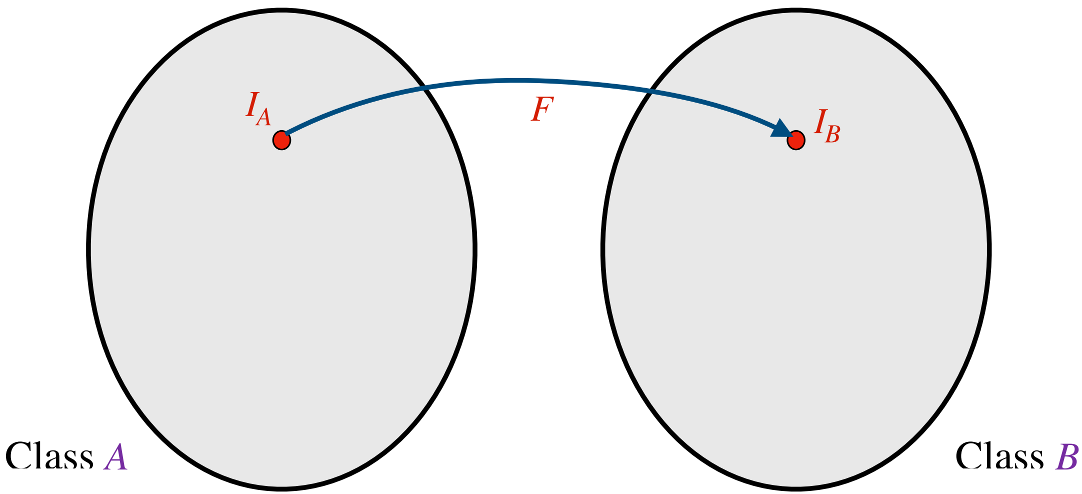
- Let D be a dominating-set of H of size at most k.
- By exchange argument, we can ensure $D \subseteq X_V$.
- Each vertex in X_E is adjacent to a vertex in D.
- Thus, D is a vertex-cover of G.

Polynomial Time Reductions

Definition: A problem A is said to be reducible in polynomial time to problem B (denoted by $A \leq_P B$) iff

There is a polynomial-time algorithm F that

- Maps an instance I_A of A to an instance $I_B = F(I_A)$ of B.
- I_A is YES-instance of $A \iff I_B$ is YES-instance of B.

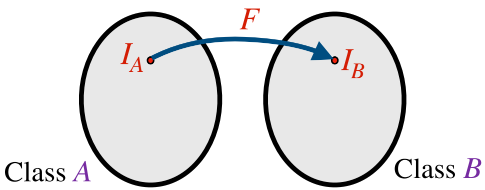


Polynomial Time Reductions

Definition: A problem A is said to be reducible in polynomial time to problem B (denoted by $A_{\leq P}B$) iff

There is a polynomial-time algorithm F that

- Maps an instance I_A of A to an instance $I_B = F(I_A)$ of B.
- I_A is YES-instance of $A \iff I_B$ is YES-instance of B.



Consequence 1: If B is polynomial time solvable, then A is also polynomial time solvable.

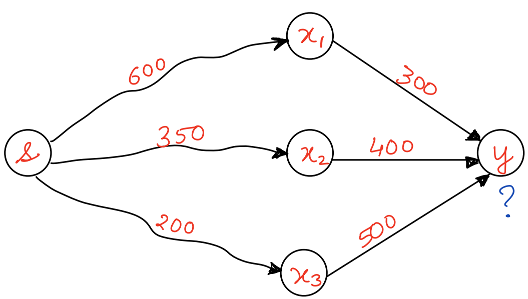
Consequence 2: If there is NO polynomial time algorithm for A, then there is NO polynomial time algorithm for B.

From last class: Computing path of maximum capacity

Claim: Given any s in flow-graph $G = (V, E, c)$, we can compute for each vertex v, an (s, v)-path of maximum capacity in $O(m \log n)$ total time.

If $x_1, \ldots, x_t$ are in-neighbors of y, then

$$\mathrm{MaxCap}(s, y) = \max_{1 \leqslant i \leqslant t} \left(\min \left(\mathrm{MaxCap}(s, x_i), c(x_i, y) \right) \right)$$



Max capacity using

(x_1, y) edge	300	
(x_2, y) edge	350	← ANSWER
(x_3, y) edge	200	

From last class: Computing path of maximum capacity

1. Initialise an empty Q, and an array max-cap[ ] of size n

2. For each vertex $v \in V$:
    max-cap[v] ← 0
    Add v to queue Q.

3. max-cap[s] ← $\cancel{0}$ ∞

4. While Q is not empty:
    x ← vertex in Q with largest max-cap[x]
    Remove x from Q
    For each out-neighbor y of x still in Q:
        temp ← min(max-cap[x] , c(x, y))
        If *temp* > max-cap[y]:
            max-cap[x] ← *temp*

5. Return max-cap[ ]