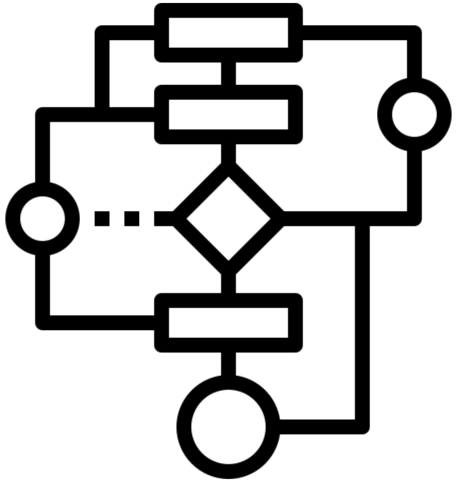


CS 310: Algorithms

Lecture 23

Instructor: Naveed Anwar Bhatti

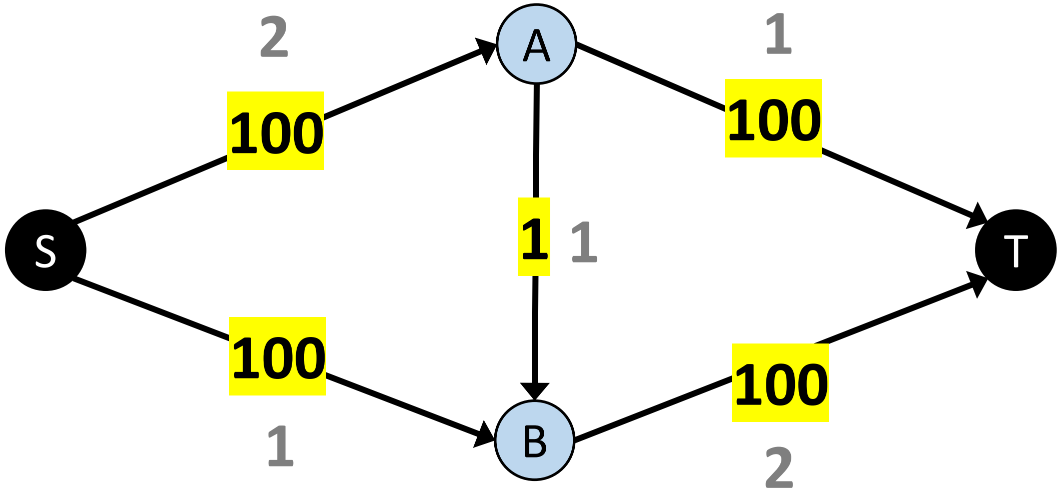
Few Slides taken from Dr. Imdad’s CS 510 course

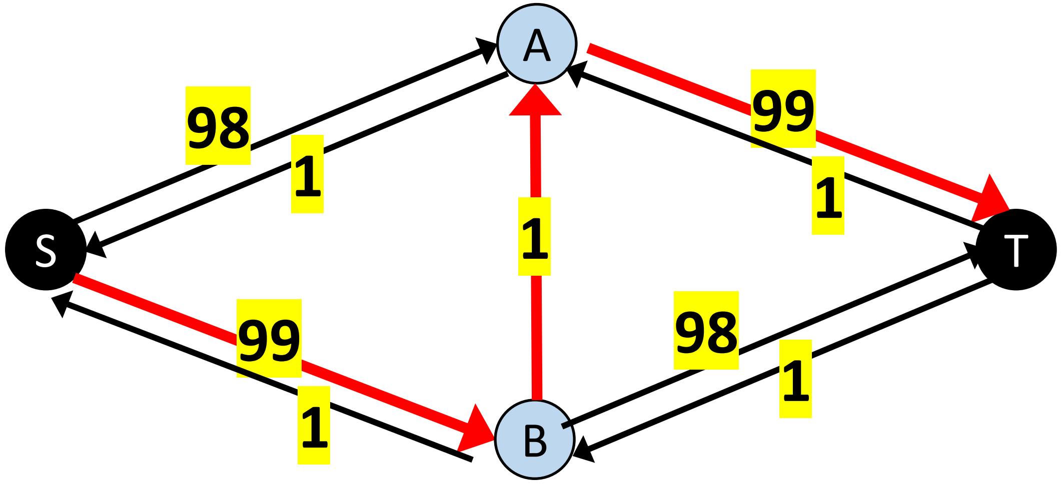

THE
END
IS
NEAR

Administrivia

- The last class will be on **4th of December**
- The last quiz will be on **4th of December**
- Final exam will be on **15th of December**

How O(f E)? Why O(f E)? When O(f E)?





Waiting for Ford Fulkerson algorithm to complete on 4 Vertices and 5 Edges

Ford-Fulkerson Algorithm

How can we improve (resolve) this?

We need to choose the augmenting path **wisely** to fix the problem

Max Flow : The Edmond-Karp Algorithm

Choose ***shortest*** augmenting paths (in terms on number of edges)

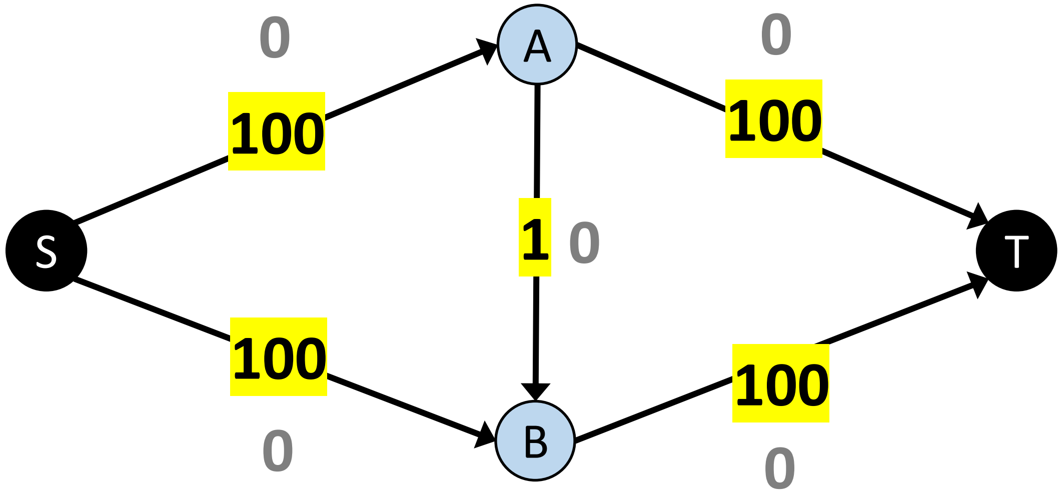
Max Flow : The Edmond-Karp Algorithm

Choose ***shortest*** augmenting paths in terms on number of edges

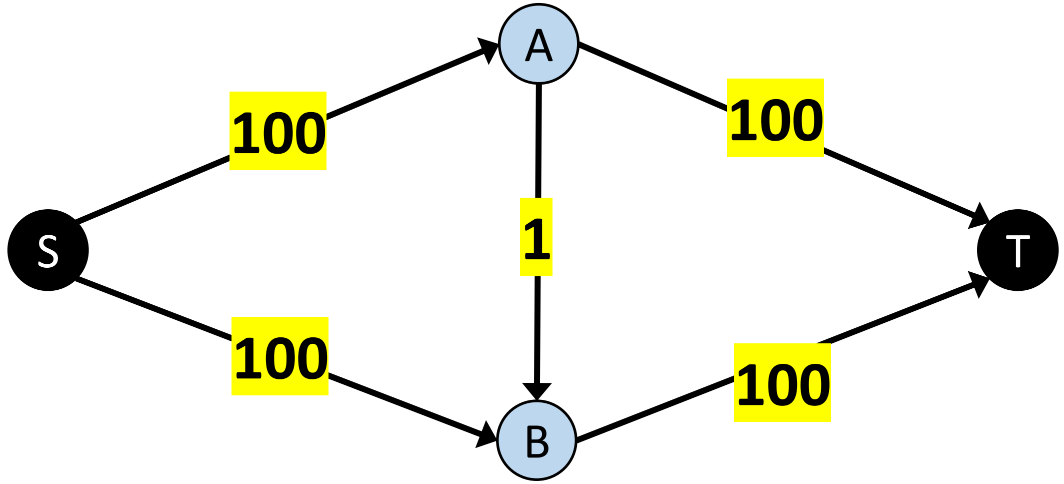
Algorithm Ford-Fulkerson Algorithm (G) with Shortest Paths

$f \leftarrow 0$ ▷ Initialize to a (valid) flow of size 0 (on every edge)
while TRUE **do**
  Compute G_f
  Find a shortest $s - t$ path P in G_f ▷ Using BFS
  if no such path **then**
    return f
  else
    $f \leftarrow \textsc{Augment}(P, f)$

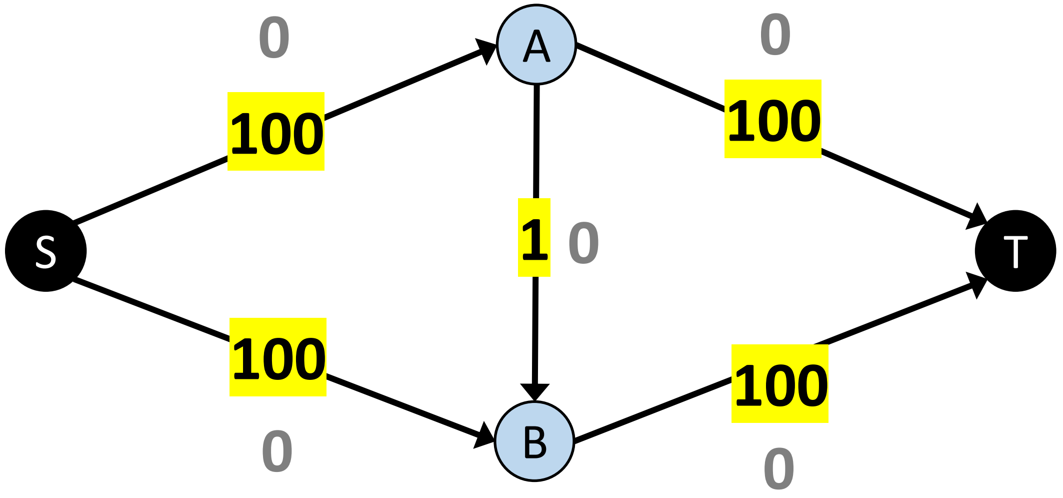
Again (Same example)



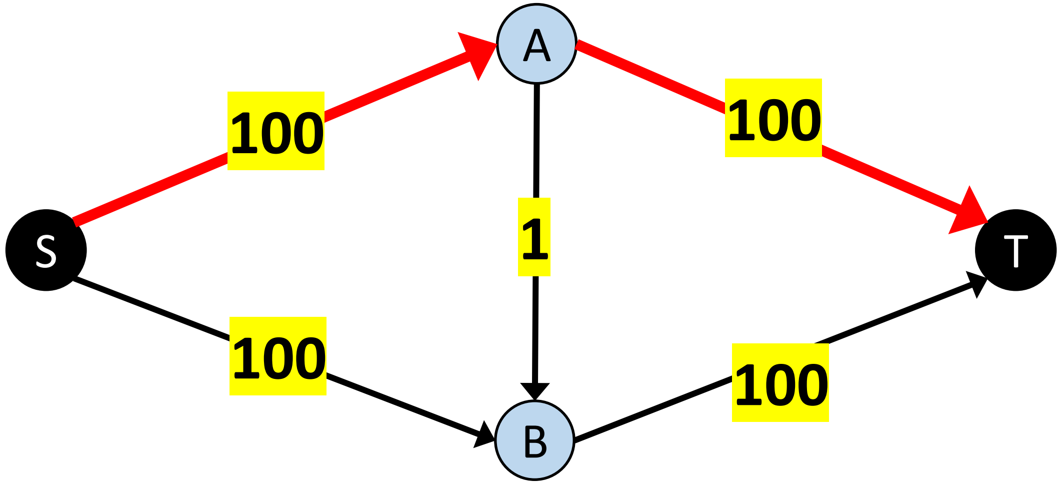
Residual Network



Again (Same example)

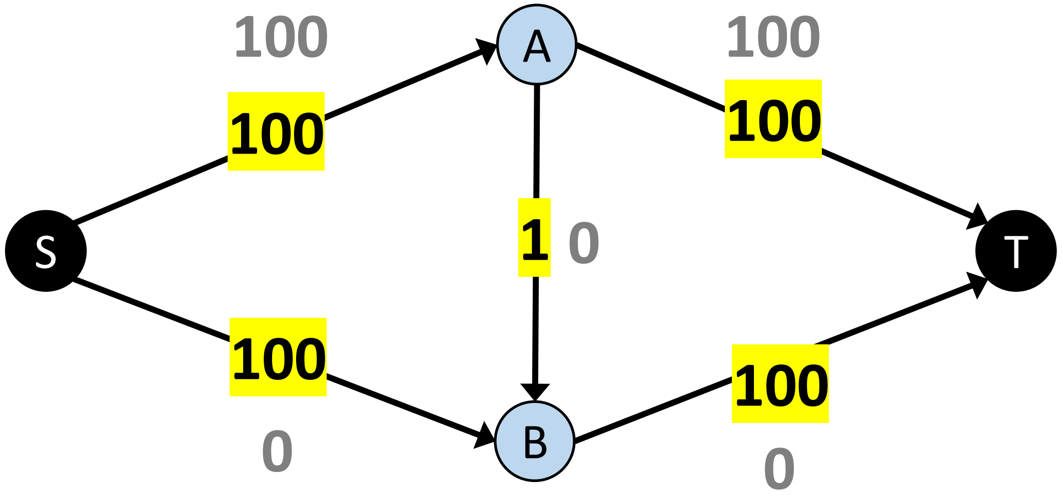


Residual Network



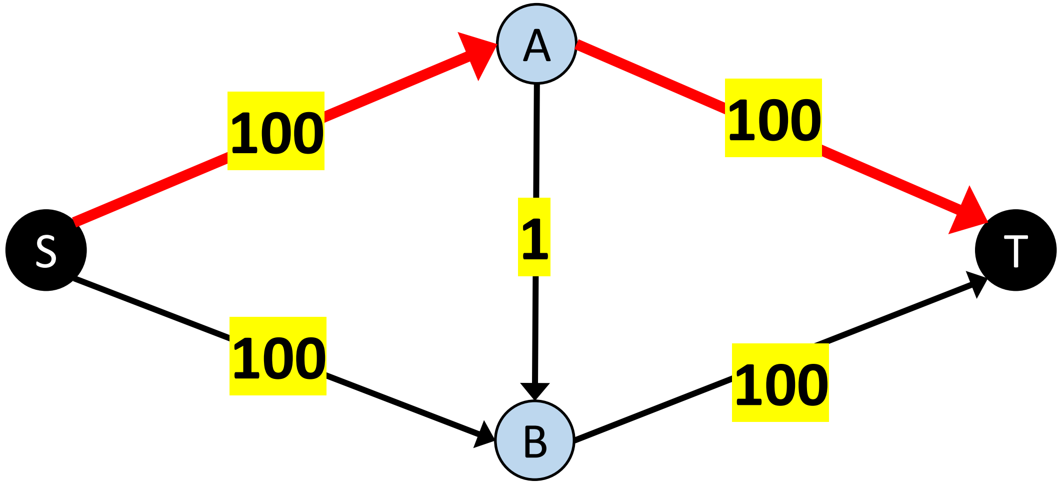
Bottleneck (P) = 100

Again (Same example)



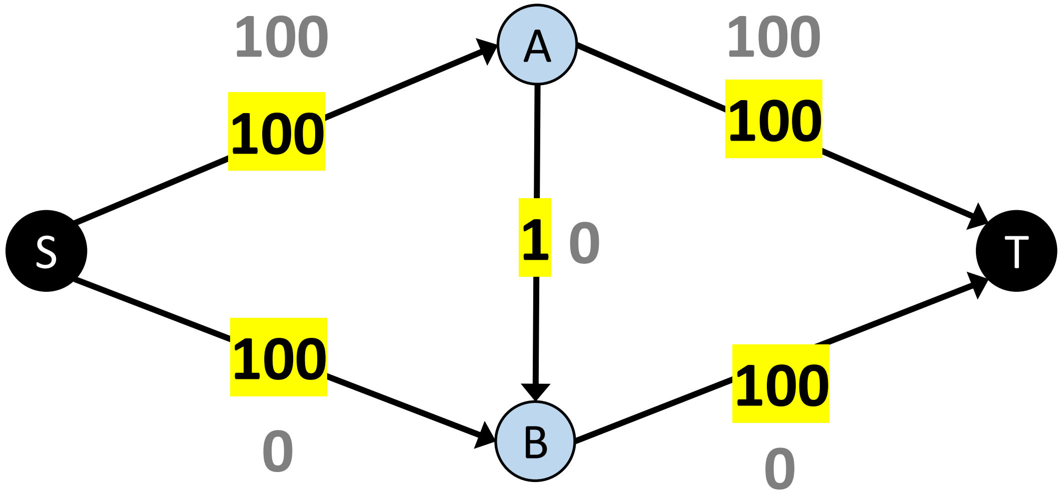
Flow Value = 100

Residual Network



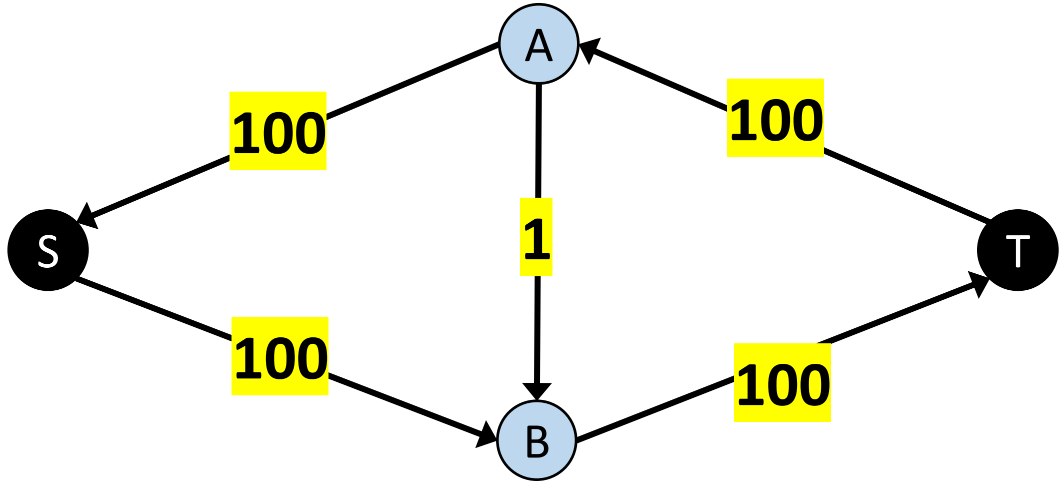
Bottleneck (P) = 100

Again (Same example)

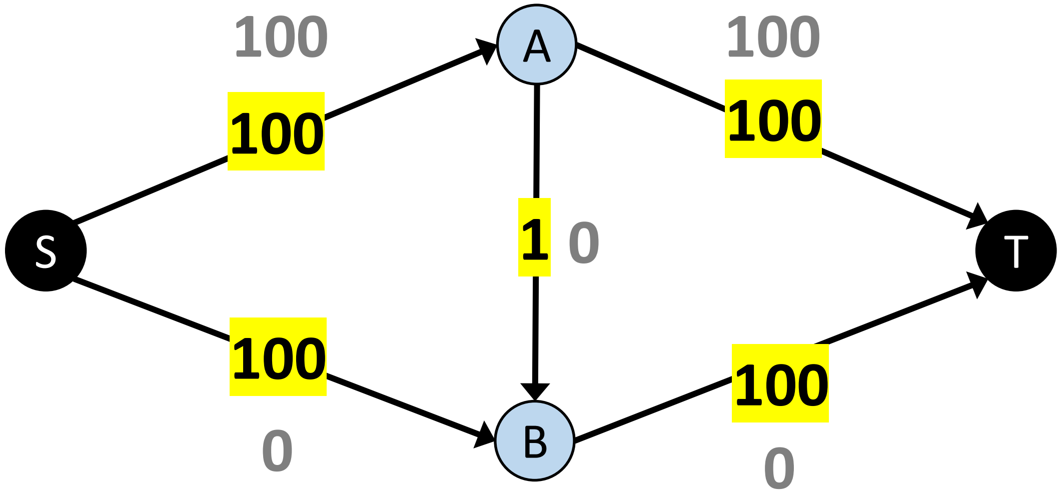


Flow Value = 100

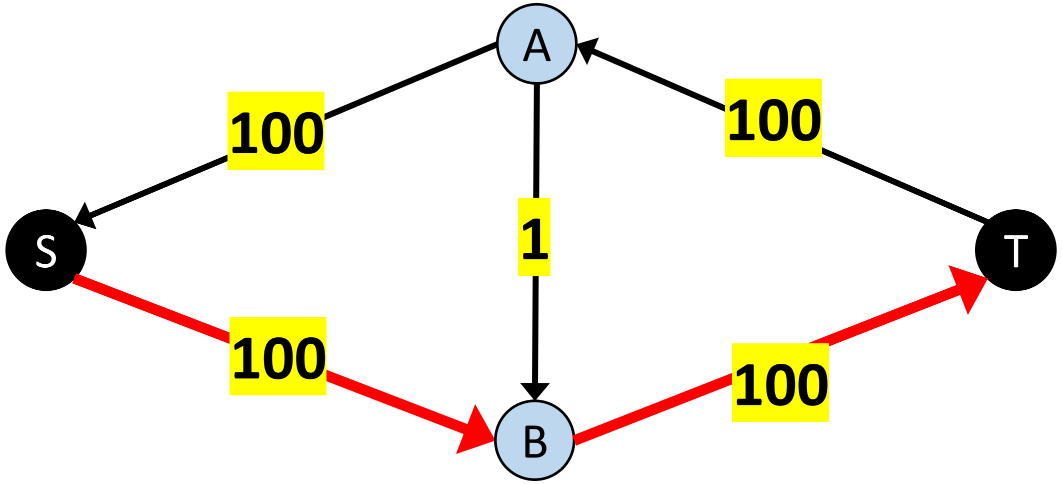
Residual Network



Again (Same example)



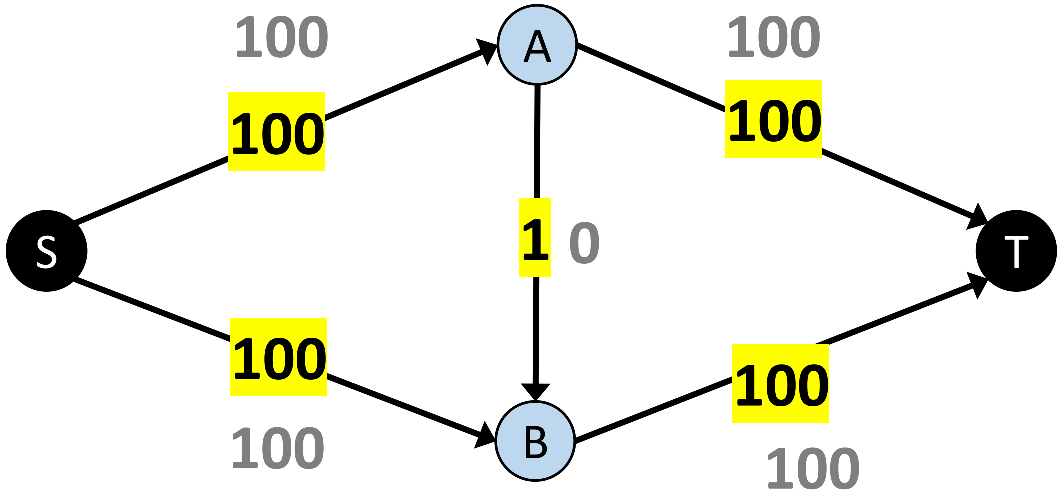
Flow Value = 100



Residual Network

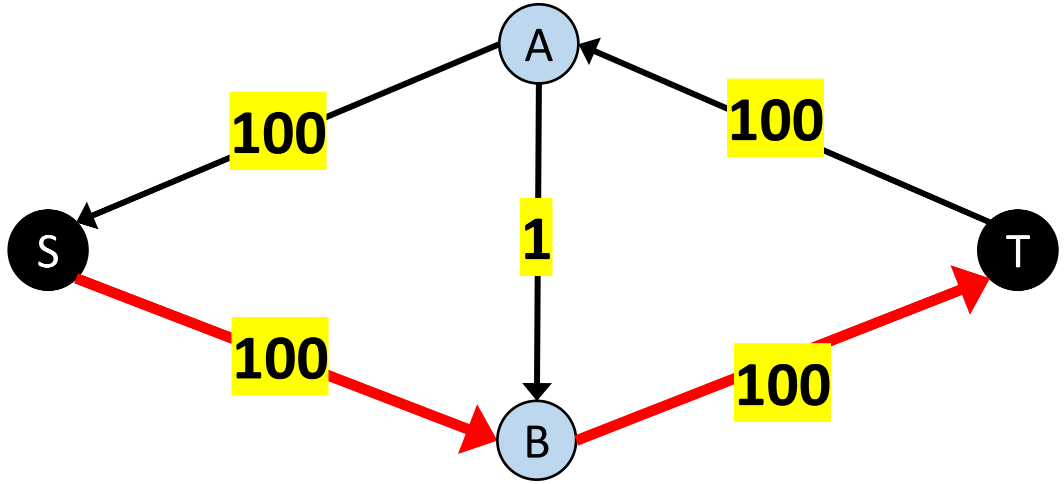
Bottleneck (P) = 100

Again (Same example)



Flow Value = 200

Residual Network



Bottleneck (P) = 100

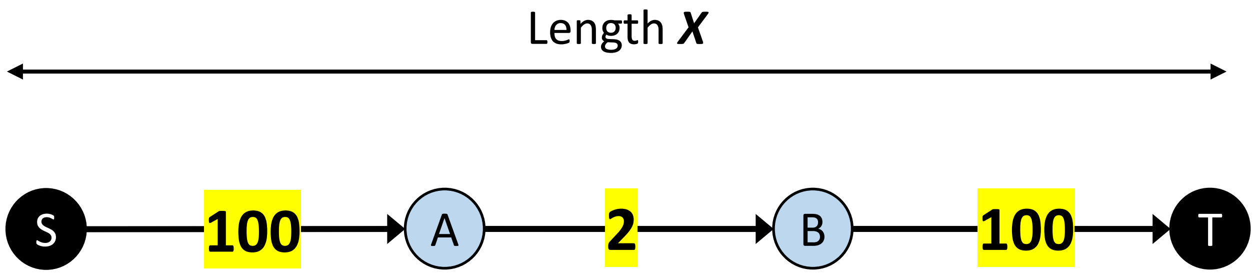
The Edmond-Karp Algorithm (Time Complexity)

Choose ***shortest*** augmenting paths in terms on number of edges

Algorithm Ford-Fulkerson Algorithm (G) with Shortest Paths

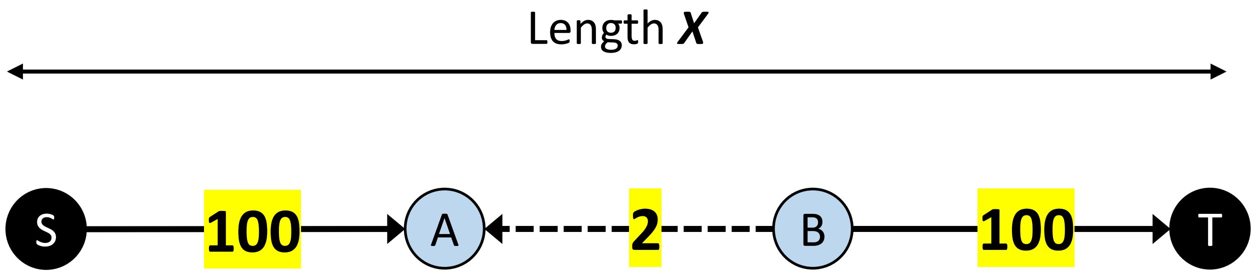
$f \leftarrow 0$ ▷ Initialize to a (valid) flow of size 0 (on every edge)
while TRUE **do**
  Compute G_f **O(V+E)**
  Find a shortest $s - t$ path P in G_f **O(E)** ▷ Using BFS
  if no such path **then**
    return f
  else
    $f \leftarrow \text{AUGMENT}(P, f)$ **O(E)**

The Edmond-Karp Algorithm (Time Complexity)



In **shortest *Augmented*** path, at least one bottleneck edge gets **saturated**

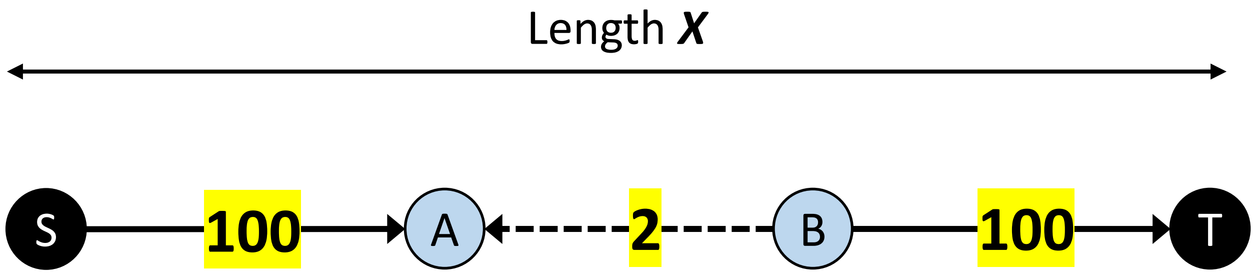
The Edmond-Karp Algorithm (Time Complexity)



In **shortest *Augmented*** path, at least one bottleneck edge gets **saturated**

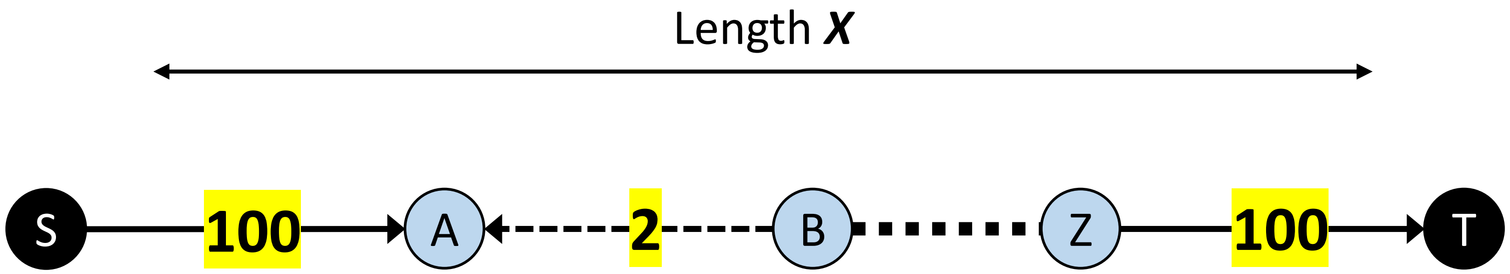
How many time each edge can be **saturated?**

The Edmond-Karp Algorithm (Time Complexity)



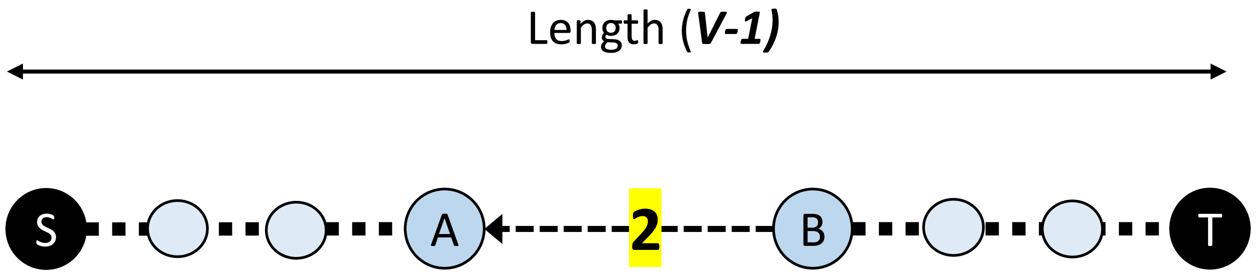
Next time when this saturated edge will get selected, the total length will be at least >= ***X+1***

The Edmond-Karp Algorithm (Time Complexity)



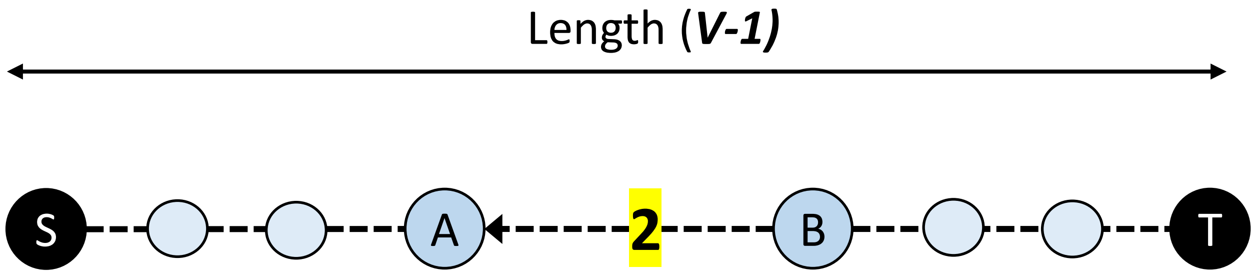
Next time when this saturated edge will get selected, the total length will be at least >= ***X+1***

The Edmond-Karp Algorithm (Time Complexity)



Next time when this saturated edge will get selected, the total length will be at least >= ***X+1***

The Edmond-Karp Algorithm (Time Complexity)



Each edge can be re-selected **V-1** times **MAX**

E edges can be re-selected **(E)*(V-1)** times **MAX**

The Edmond-Karp Algorithm (Time Complexity)

Choose ***shortest*** augmenting paths in terms on number of edges

---

Algorithm Ford-Fulkerson Algorithm (G) with Shortest Paths

$f \leftarrow 0$ ▷ Initialize to a (valid) flow of size 0 (on every edge)

while TRUE **do** O(EV)

  Compute G_f O(V+E)

  Find a shortest $s - t$ path P in G_f O(E) ▷ Using BFS

  if no such path **then**

    return f

  else

    $f \leftarrow \text{AUGMENT}(P, f)$ O(E)

The Edmond-Karp Algorithm (Time Complexity)

Choose ***shortest*** augmenting paths in terms on number of edges

---

Algorithm Ford-Fulkerson Algorithm (G) with Shortest Paths

$f \leftarrow 0$ ▷ Initialize to a (valid) flow of size 0 (on every edge)

while TRUE **do** O(EV)

  Compute G_f O(V+E)

  Find a shortest $s - t$ path P in G_f O(E) ▷ Using BFS

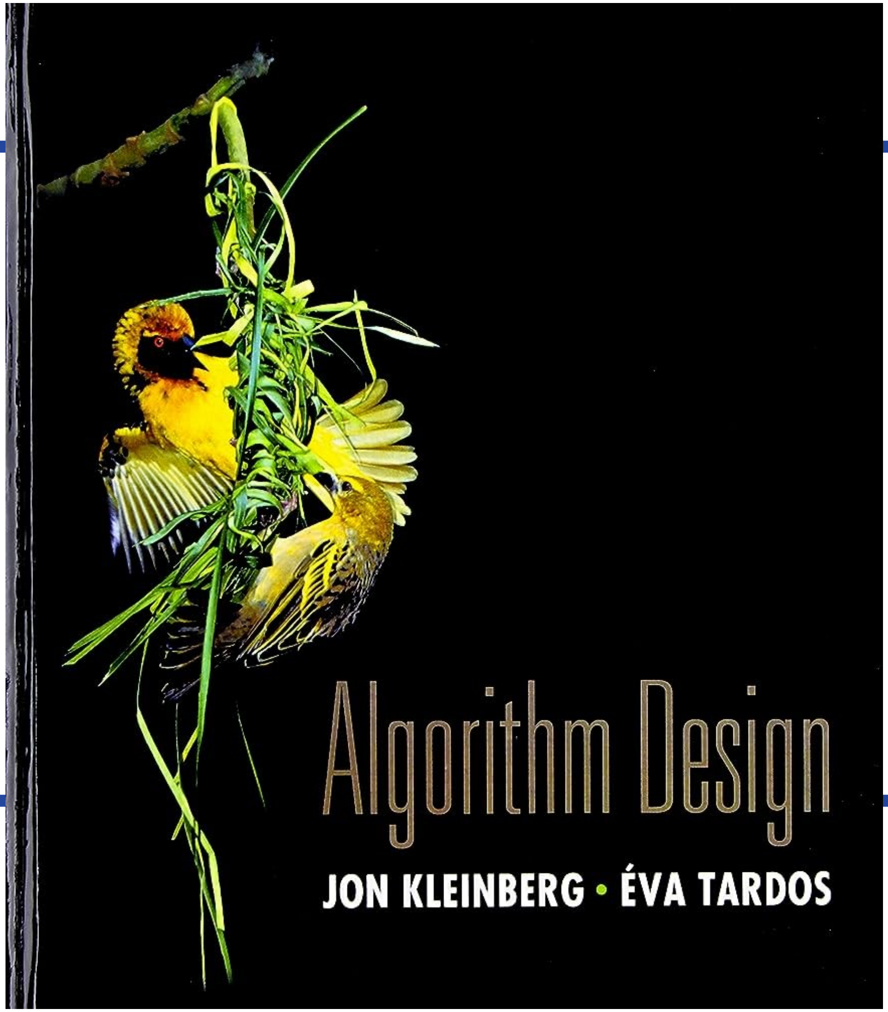
  if no such path **then**

    return f

  else

    $f \leftarrow$ AUGMENT(P, f) O(E)

$O(VE^2)$ when E >= V



Chapter 8:
NP and Computational Intractability

Section 8.1 :
Polynomial Time Reduction

Efficiently Solvable Problems

So far, we dealt with problems like:

- Sorting ***n*** numbers
- Connected components in a graph
- Shortest path between two points (s-t path),
- Minimum Spanning Tree (MST),
- Best alignment (possibly in sequence alignment)
- Maximum flow

We devised ***efficient*** algorithms for them

Efficient in what sense?

Efficiently Solvable Problems

So far, we dealt with problems like:

- Sorting ***n*** numbers ----- $n!$
- Connected components in a graph ----- 2^E
- Shortest path between two points (s-t path), ----- 2^V
- Minimum Spanning Tree (MST), ----- n^n
- Best alignment (possibly in sequence alignment)
- Maximum flow

Search space for solutions is typically exponential in these problems

Efficiently Solvable Problems

So far, we dealt with problems like:

- Sorting $\boldsymbol{n}$ numbers ----- $\boldsymbol{n}!$
- Connected components in a graph ----- $\mathbf{2}^{\boldsymbol{E}}$
- Shortest path between two points (s-t path), ----- $\mathbf{2}^{\boldsymbol{V}}$
- Minimum Spanning Tree (MST), ----- $\boldsymbol{n}^{\boldsymbol{n}}$
- Best alignment (possibly in sequence alignment)
- Maximum flow

Efficiently Solvable Problem = Polynomial Time complexity

$\exists$ an $O(n^k)$ worst case time algorithm for instances of size n, constant k

Intractable Problems

Dictionary

Definitions from Oxford Languages · Learn more

intractable

/ɪnˈtraktəbl/

adjective

hard to control or deal with.
"intractable economic problems"

Similar: unmanageable | uncontrollable | ungovernable | out of control | out of hand

- (of a person) difficult or stubborn.

Similar: stubborn | obstinate | obdurate | inflexible | unadaptable | unmalleable

Hard (Intractable) Problems

- No known $O(n^k)$ algorithm
- Exponential time is needed $O(n^n), O(n!), O(k^n)$

Hard (Intractable) Problems

Hard (Intractable) Problems

- No known $O(n^k)$ algorithm
- Exponential time is needed $O(n^n), O(n!), O(k^n)$

Cannot say they are not efficiently solvable (just don't know yet)

We establish that These "hard problems" in some sense are equivalent

Classifying Problem Types

Decision Problem: Output Yes/No

Connectivity: can we get from s to t in a graph G?

Optimization Problem: Find the best numerical value

Distance: what is the length of shortest path from s to t in a graph G?

Max Flow-Min Cut: what is the maximum flow that can pass in a flow network?

Search Problem: Find a particular object

Shortest Path: find the shortest path with a lowest cost

Polynomial Time Reduction

- To explore the class of computational hard problems, we define a notion of comparing the hardness of two problems
- Measures the relative difficulty of two problems

Problem A is polynomial time reducible to **Problem B**, $A \leq_p B$

If any instance of problem ***A*** can be transform using a polynomial amount of computation to instance of ***B*** plus polynomial amount of computation to solve problem ***B***

Polynomial Time Reduction

- To explore the class of computational hard problems, we define a notion of comparing the hardness of two problems
- Measures the relative difficulty of two problems

Problem A is polynomial time reducible to Problem B, $A \leq_p B$

If any instance of problem ***A*** can be transform using a polynomial amount of computation to instance of ***B*** plus polynomial amount of computation to solve problem ***B***

Problem A is polynomial time reducible to Problem B, $A \leq_p B$

If any subroutine (C + + function) for problem ***B*** can be used (called (once or more) with clever legal inputs) to solve any instance of problem A

Polynomial Time Reduction

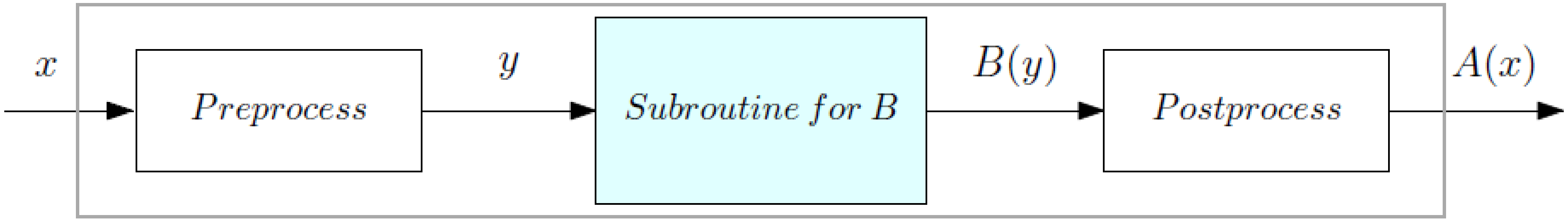
Problem A is polynomial time reducible to Problem B, $A \leq_p B$

If any instance of problem ***A*** can be transform using a polynomial amount of computation to instance of ***B*** plus polynomial amount of computation to solve problem ***B***

Problem A is polynomial time reducible to Problem B, $A \leq_p B$

If any subroutine (C + + function) for problem ***B*** can be used (called (once or more) with clever legal inputs) to solve any instance of problem A

Subroutine for B takes an instance y of B and return the solution $B(y)$



Algorithm for A transform an instance x of A to an instance y of B. Then transform $B(y)$ to $A(x)$

Polynomial Time Reduction

Problem A is polynomial time reducible to Problem B, $A \leq_p B$

If any instance of problem ***A*** can be transform using a polynomial amount of computation to instance of ***B*** plus polynomial amount of computation to solve problem ***B***

1. *If there is a polynomial time algorithm for **B**, then there is a polynomial time algorithm for **A***

2. *If there is no polynomial time algorithm for **A**, then there is no polynomial time algorithm for **B***

Polynomial Time Reduction

Problem A is polynomial time reducible to Problem B, $A \leq_p B$

If any instance of problem ***A*** can be transform using a polynomial amount of computation to instance of ***B*** plus polynomial amount of computation to solve problem ***B***

$$FindMin \leq_p Sort$$

$$Sort \leq_p FindMin$$

Polynomial Time Reduction

Reductions by Equivalence

Reduction from special case to general case

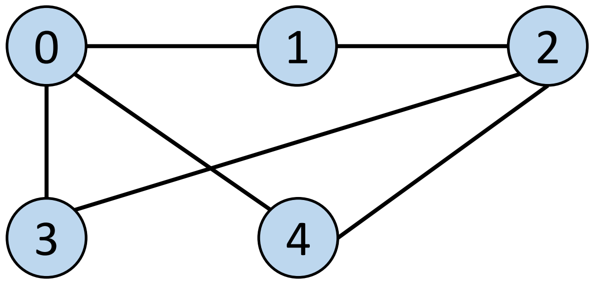
Reduction by encoding with gadgets

Polynomial Time Reduction – *by equivalence*

Minimum Vertex Cover $\leq_p$ Independent Set

Find minimum number of vertices '**K**' such that for every edge (u, v) of the undirected graph '**G**', either 'u' or 'v' is in the vertex cover.

Problem: Is there a size '**K'** vertex cover in graph '**G**'?



Polynomial Time Reduction – *by equivalence*

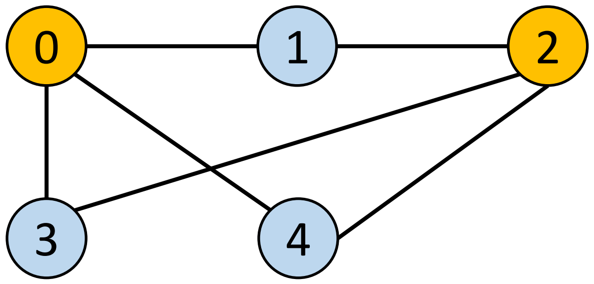
Minimum Vertex Cover

$\leq_p$

Independent Set

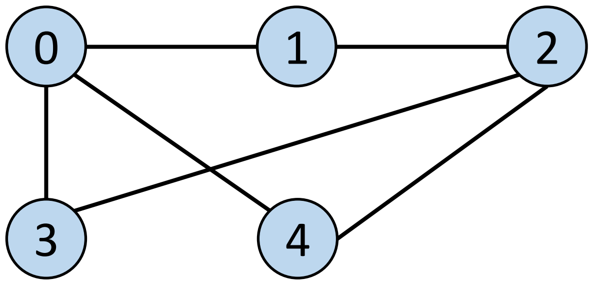
Find minimum number of vertices '**K**' such that for every edge (u, v) of the undirected graph '**G**', either 'u' or 'v' is in the vertex cover.

Problem: Is there a size '**K'** vertex cover in graph '**G**'?



Finding the largest independent set of size '**K**' in a undirected graph '**G**', where an independent set is a set of vertices such that no two vertices are adjacent

Problem: Is there a size '**K'** independent set in graph '**G**'?



Polynomial Time Reduction – *by equivalence*

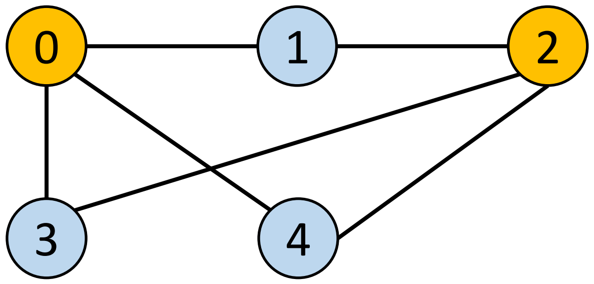
Minimum Vertex Cover

$\leq_p$

Independent Set

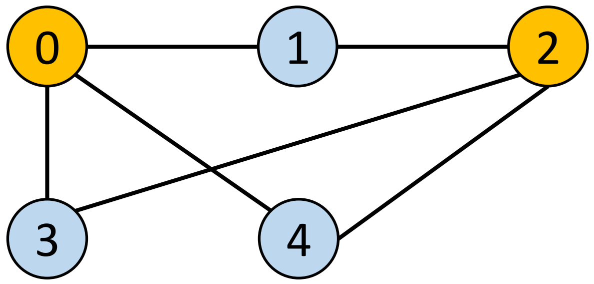
Find minimum number of vertices '**K**' such that for every edge (u, v) of the undirected graph '**G**', either 'u' or 'v' is in the vertex cover.

Problem: Is there a size '**K'** vertex cover in graph '**G**'?



Finding the largest independent set of size '**K**' in a undirected graph '**G**', where an independent set is a set of vertices such that no two vertices are adjacent

Problem: Is there a size '**K'** independent set in graph '**G**'?



Polynomial Time Reduction – *by equivalence*

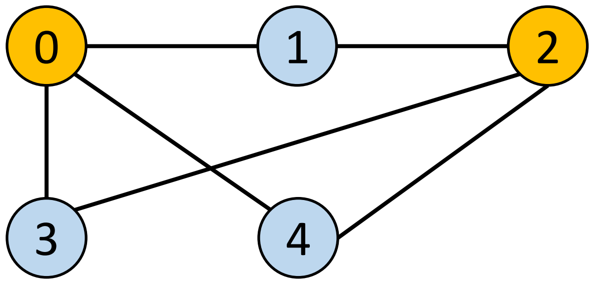
Minimum Vertex Cover

$\leq_p$

Independent Set

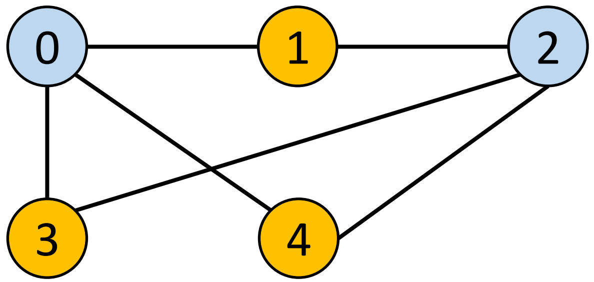
Find minimum number of vertices ‘**K**’ such that for every edge (u, v) of the undirected graph ‘**G**’, either ‘u’ or ‘v’ is in the vertex cover.

Problem: Is there a size ‘**K’** vertex cover in graph ‘**G**’?



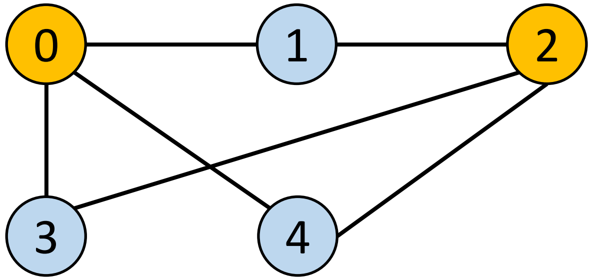
Finding the largest independent set of size ‘**K**’ in a undirected graph ‘**G**’, where an independent set is a set of vertices such that no two vertices are adjacent

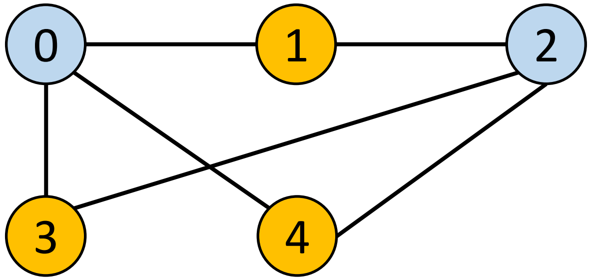
Problem: Is there a size ‘**K’** independent set in graph ‘**G**’?



Polynomial Time Reduction – *by equivalence*

Minimum Vertex Cover $\leq_p$ Independent Set





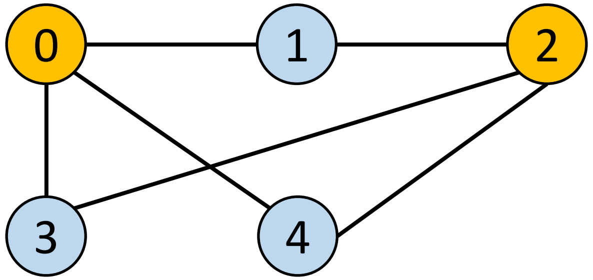
Step 1: Translation Algorithm

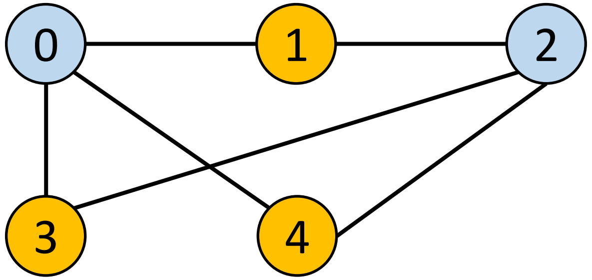
Step 2: Prove that Translation Algorithm runs in polynomial time

Step 3: Prove that the original instance will have output of “**YES**” *iff* translated instance have output of “**YES**”

Polynomial Time Reduction – *by equivalence*

Minimum Vertex Cover $\leq_p$ Independent Set





Step 1: Translation Algorithm

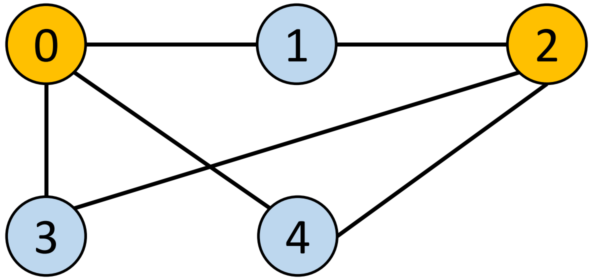
Given **G=(V,E)** and **K** for VC, build **G`=G, K`= V-K** for IS

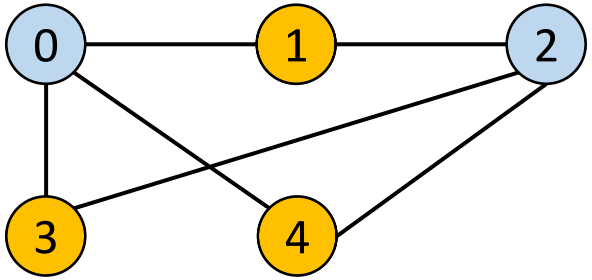
Step 2: Prove that Translation Algorithm runs in polynomial time

Step 3: Prove that the original instance will have output of "**YES**" *iff* translated instance have output of "**YES**"

Polynomial Time Reduction – *by equivalence*

Minimum Vertex Cover $\leq_p$ Independent Set





Step 1: Translation Algorithm

Given **G=(V,E)** and **K** for VC, build **G`=G, K`= V-K** for IS

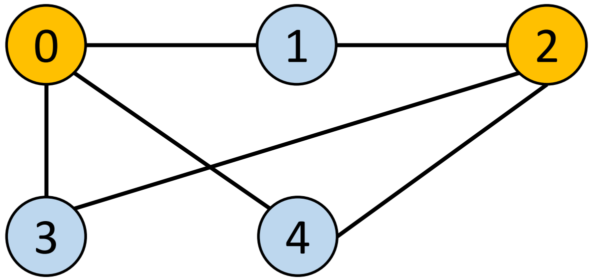
Step 2: Prove that Translation Algorithm runs in polynomial time

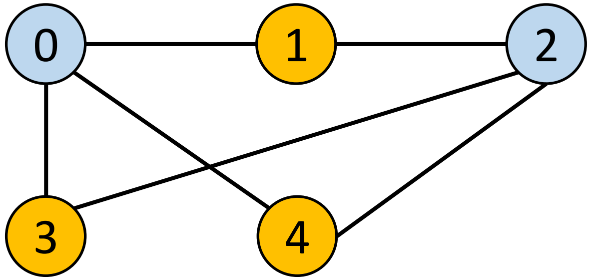
This conversion takes constant time

Step 3: Prove that the original instance will have output of "**YES**" *iff* translated instance have output of "**YES**"

Polynomial Time Reduction – *by equivalence*

Minimum Vertex Cover $\leq_p$ Independent Set





Step 3: Prove that the original instance will have output of "**YES**" *iff* translated instance have output of "**YES**"

IS (G`, K`) = yes **=>** **VC** (G, K) = yes

- Suppose that **S** is an independent set.
- Consider an arbitrary **edge e = (u, v)**.
- Since **S** is independent set, it cannot be the case that both **u** and **v** are in **S**; so, one of them must be in **V - S**.
- It follows that every edge has at least one end in **V - S**, and so **V - S** is a vertex cover.

VC (G, K) = yes **=>** **IS** (G`, K`) = yes

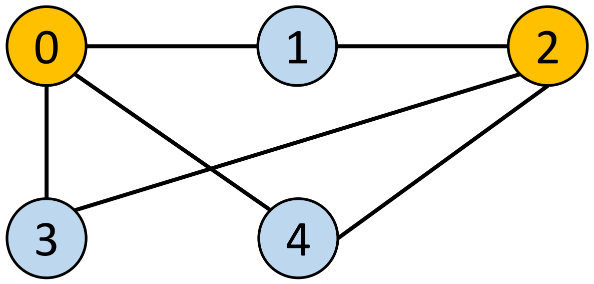
- Suppose that **S** is Vertex Cover set of size **K** of graph **G**.
- At least one end of all edges should be in **S**
- Any vertex in **V-S** must be connected to vertices in **S**
- So, **V-S** vertices cannot be adjacent to each other
- Thus **V-S** is Independent Set

Polynomial Time Reduction - *from Special Cases to General Case*

Minimum Vertex Cover $\leq_p$ Set Cover

Find minimum number of vertices ‘**K**’ such that for every edge (u, v) of the undirected graph ‘**G**’, either ‘u’ or ‘v’ is in the vertex cover.

Problem: Is there a size **‘K’** vertex cover in graph ‘**G**’?



Given a set of elements **U = {1, 2, ..., n}** (called the universe) and a collection of subsets of **U**, **S = {**S_1**,** S_1**, ...,** S_m**}**, and an integer **‘K’.** Input is (**U**,**S**,**K**)

Problem: Does there exist **‘K’** or fewer subsets such that their union is equal to **U**?

U = {1, 2, 3, 4, 5, 6, 7} **K=2**

S_1= {3,7}

S_2= {3,4,5,6}

S_3= {1}

S_4= {2,4}

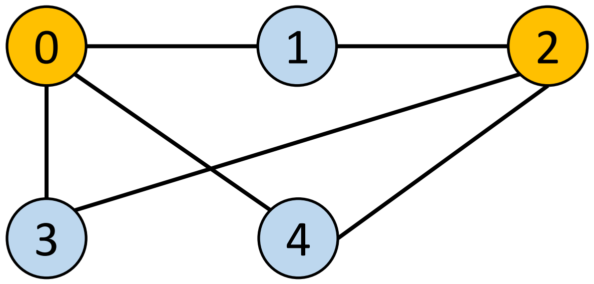
S_5= {1,2,6,7}

Polynomial Time Reduction - *from Special Cases to General Case*

Minimum Vertex Cover $\leq_p$ Set Cover

Find minimum number of vertices '**K**' such that for every edge (u, v) of the undirected graph '**G**', either 'u' or 'v' is in the vertex cover.

Problem: Is there a size **'K'** vertex cover in graph '**G**'?



Given a set of elements **U = {1, 2, ..., n}** (called the universe) and a collection of subsets of **U**, **S = {**S_1**,** S_1**, ...,** S_m**}**, and an integer **'K'.** Input is (**U**,**S**,**K**)

Problem: Does there exist **'K'** or fewer subsets such that their union is equal to **U**?

U = {1, 2, 3, 4, 5, 6, 7} **K=2**

S_1= {3,7}

S_2= **{3,4,5,6}**

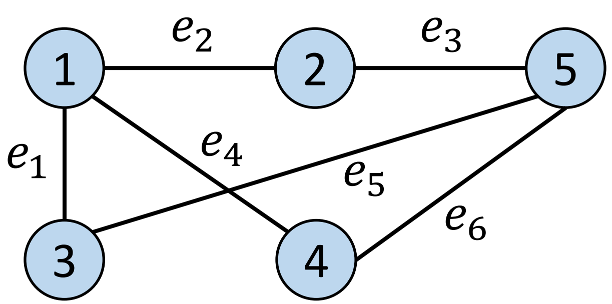
S_3= {1}

S_4= {2,4}

S_5= **{1,2,6,7}**

Polynomial Time Reduction - *from Special Cases to General Case*

Minimum Vertex Cover $\leq_p$ Set Cover



Step 1: Translation Algorithm

Given **G=(V,E)** and **K** for VC, we build **Set Cover (SC)** input:

U = $\{e_1, e_2, e_3, e_4, e_5, e_6\}$

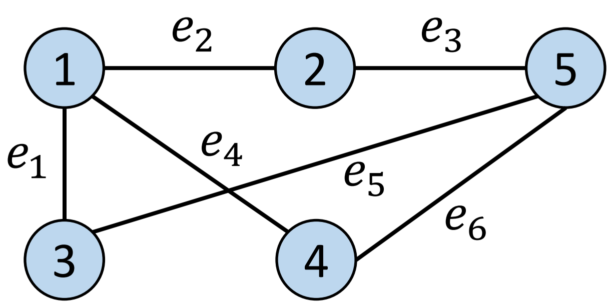
$\boldsymbol{S_1} = \{e_1, e_2, e_4\}$ $\quad$ $\boldsymbol{S_4} = \{e_4, e_6\}$

$\boldsymbol{S_2} = \{e_2, e_3\}$ $\quad$ $\boldsymbol{S_5} = \{e_3, e_5, e_6\}$

$\boldsymbol{S_3} = \{e_1, e_4\}$ $\quad$ **K`= K**

Polynomial Time Reduction - *from Special Cases to General Case*

Minimum Vertex Cover $\leq_p$ Set Cover



Step 1: Translation Algorithm

Given **G=(V,E)** and **K** for VC, we build Set Cover input:

U = $\{e_1, e_2, e_3, e_4, e_5, e_6\}$

$S_1 = \{e_1, e_2, e_4\}$

$\boldsymbol{S_2} = \{e_2, e_3\}$

$\boldsymbol{S_3} = \{e_1, e_4\}$

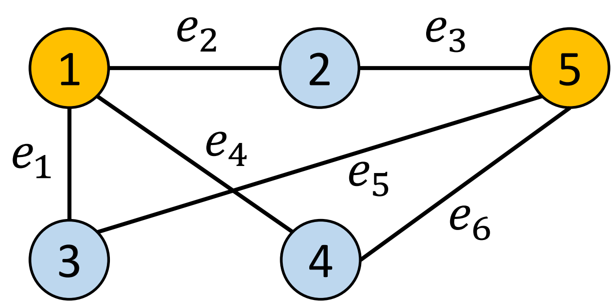
$\boldsymbol{S_4} = \{e_4, e_6\}$

$S_5 = \{e_3, e_5, e_6\}$

K\`= K

Polynomial Time Reduction - *from Special Cases to General Case*

Minimum Vertex Cover $\leq_p$ Set Cover



Step 1: Translation Algorithm

Given **G=(V,E)** and **K** for VC, we build Set Cover input:

U = $\{e_1, e_2, e_3, e_4, e_5, e_6\}$

$S_1 = \{e_1, e_2, e_4\}$

$S_2 = \{e_2, e_3\}$

$S_3 = \{e_1, e_4\}$

$S_4 = \{e_4, e_6\}$

$S_5 = \{e_3, e_5, e_6\}$

K\`= K

Step 2 and **Step 3** are straight forward

Polynomial Time Reduction – *by encoding gadgets*

3-Sat $\leq_p$ Independent Set

$$f = (x_{11} \vee x_{12} \vee x_{13}) \wedge (x_{21} \vee x_{22} \vee x_{23}) \wedge \dots \quad \dots \wedge (x_{m1} \vee x_{m2} \vee x_{m3})$$

We need to set each of $x_1, \dots, x_n$ to $0/1$ so as $f = 1$

Alternatively,

1. We need to pick a literal from each clause and set it to 1
2. But we cannot make conflicting settings

Polynomial Time Reduction – *by encoding gadets*

3-Sat $\leq_{\boldsymbol{p}}$ Independent Set

$$f = (x_{11} \vee x_{12} \vee x_{13}) \wedge (x_{21} \vee x_{22} \vee x_{23}) \wedge \dots \quad \dots \wedge (x_{m1} \vee x_{m2} \vee x_{m3})$$

Step 1: Translation Algorithm

- Given f on n variables and m clauses - Make a graph G
- For each clause make a triangle with nodes labeled with literals
- Connect nodes of each triangle with edges
- Make edges between literals appearing in different clauses as complements

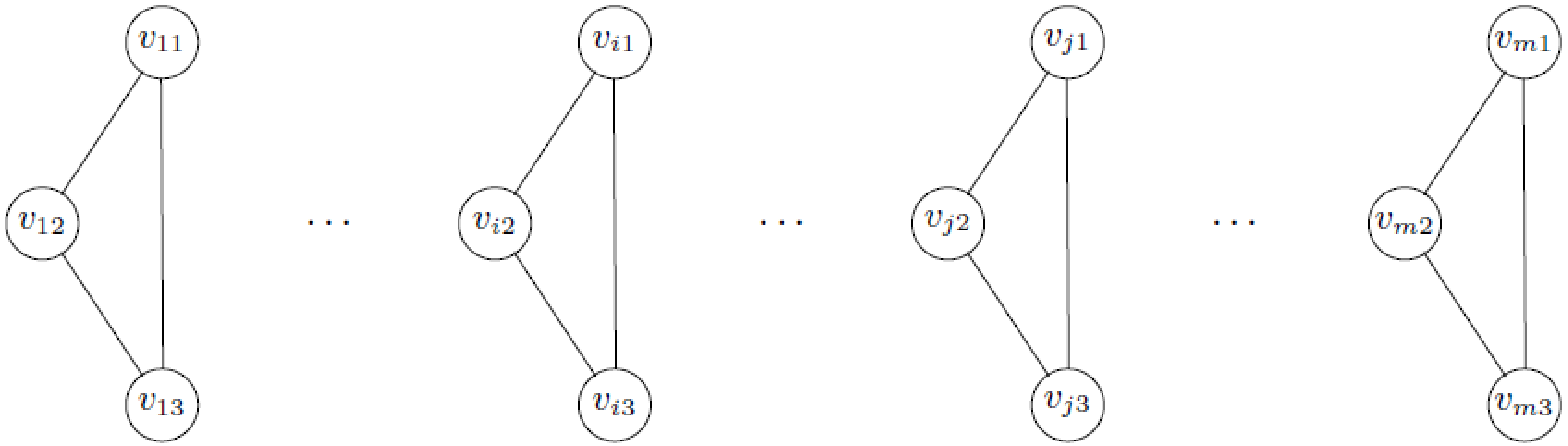
Polynomial Time Reduction – *by encoding gadets*

3-Sat $\leq_p$ Independent Set

Step 1: Translation Algorithm

- Given f on n variables and m clauses - Make a graph G
- For each clause make a triangle with nodes labeled with literals
- Connect nodes of each triangle with edges
- Make edges between literals appearing in different clauses as complements

$$(x_{11} \vee x_{12} \vee x_{13}) \wedge \ \ldots \ \wedge (x_{i1} \vee x_{i2} \vee x_{i3}) \wedge \ \ldots \ \wedge (x_{j1} \vee x_{j2} \vee x_{j3}) \wedge \ \ldots \ \wedge (x_{m1} \vee x_{m2} \vee x_{m3})$$



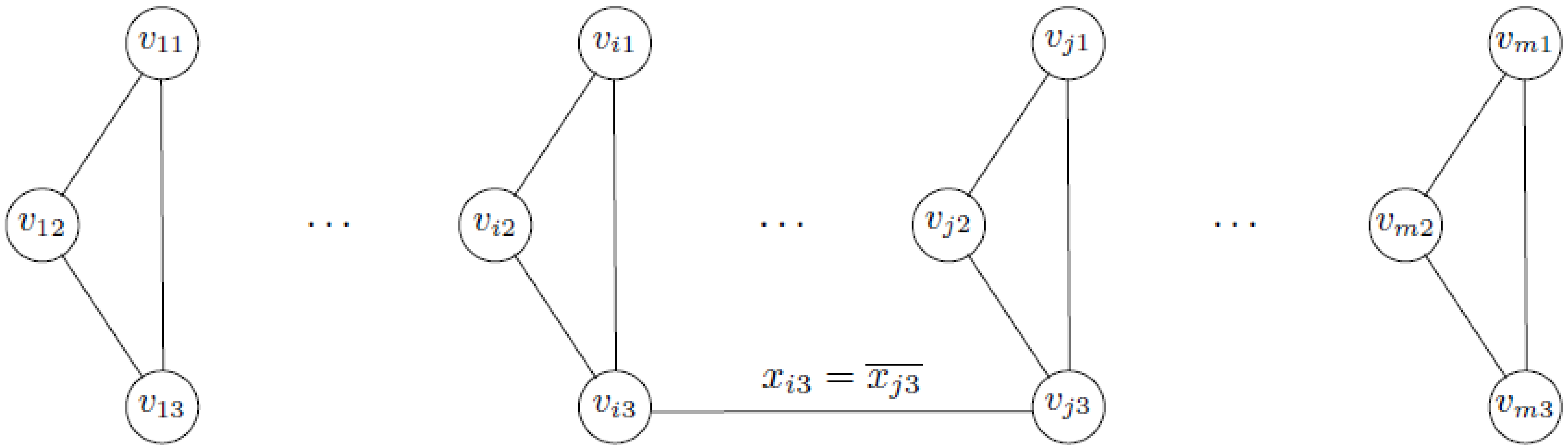
Polynomial Time Reduction – *by encoding gadets*

3-Sat $\leq_p$ Independent Set

Step 1: Translation Algorithm

- Given f on n variables and m clauses - Make a graph G
- For each clause make a triangle with nodes labeled with literals
- Connect nodes of each triangle with edges
- Make edges between literals appearing in different clauses as complements

$$(x_{11} \vee x_{12} \vee x_{13}) \wedge \ \ldots \ \wedge (x_{i1} \vee x_{i2} \vee x_{i3}) \wedge \ \ldots \ \wedge (x_{j1} \vee x_{j2} \vee x_{j3}) \wedge \ \ldots \ \wedge (x_{m1} \vee x_{m2} \vee x_{m3})$$



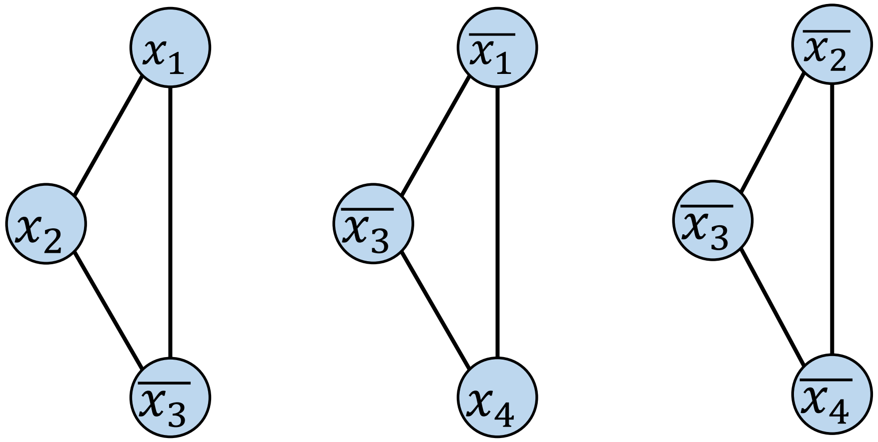
Polynomial Time Reduction – *by encoding gadets*

3-Sat $\leq_p$ Independent Set

Step 1: Translation Algorithm

- Given f on n variables and m clauses - Make a graph G
- For each clause make a triangle with nodes labeled with literals
- Connect nodes of each triangle with edges
- Make edges between literals appearing in different clauses as complements

$$(x_1 \vee x_2 \vee \overline{x_3}) \quad \wedge \quad (\overline{x_1} \vee \overline{x_3} \vee x_4) \quad \wedge \quad (\overline{x_2} \vee \overline{x_3} \vee \overline{x_4})$$



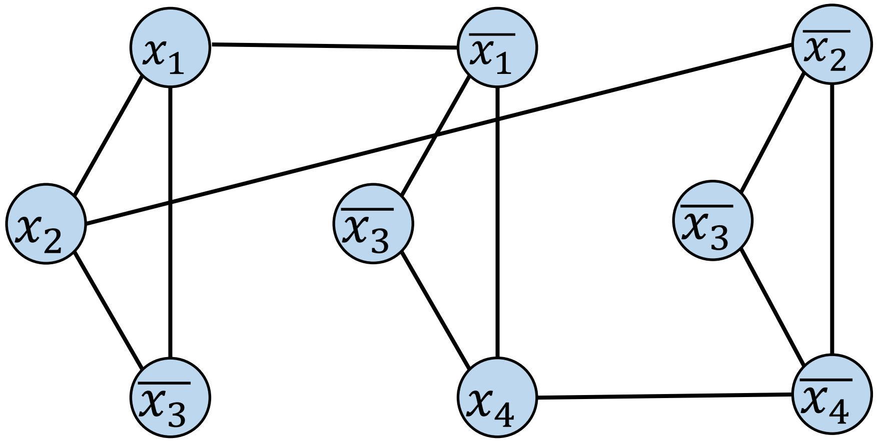
Polynomial Time Reduction – *by encoding gadets*

3-Sat $\leq_p$ Independent Set

Step 1: Translation Algorithm

- Given f on n variables and m clauses - Make a graph G
- For each clause make a triangle with nodes labeled with literals
- Connect nodes of each triangle with edges
- Make edges between literals appearing in different clauses as complements

$$(x_1 \vee x_2 \vee \overline{x_3}) \quad \wedge \quad (\overline{x_1} \vee \overline{x_3} \vee x_4) \quad \wedge \quad (\overline{x_2} \vee \overline{x_3} \vee \overline{x_4})$$



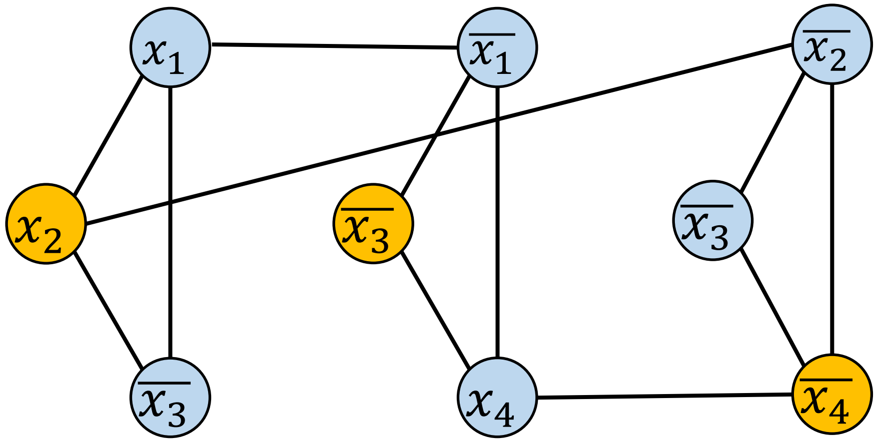
Polynomial Time Reduction – *by encoding gadets*

3-Sat $\leq_p$ Independent Set

Step 1: Translation Algorithm

- Given f on n variables and m clauses - Make a graph G
- For each clause make a triangle with nodes labeled with literals
- Connect nodes of each triangle with edges
- Make edges between literals appearing in different clauses as complements

$$(x_1 \vee x_2 \vee \overline{x_3}) \quad \wedge \quad (\overline{x_1} \vee \overline{x_3} \vee x_4) \quad \wedge \quad (\overline{x_2} \vee \overline{x_3} \vee \overline{x_4})$$



Polynomial Time Reduction – *by encoding gadets*

3-Sat $\leq_p$ Independent Set

Theorem: f is satisfiable iff G has an independent set of size m

The reduction is as follows:

- Let $\mathcal{A}$ be an algorithm for the $\text{INDEPENDENT-SET}(G, k)$ problem
- We will use $\mathcal{A}$ to solve the $3\text{-SAT}(f)$ problem
- Given any instance f of $3\text{-SAT}(f)$ on n variables and m clauses
- Construct the graph as outlined above
- Call $\mathcal{A}$ on $[G, m]$
- if $\mathcal{A}$ returns **Yes**, declare f satisfiable and vice-versa
- G can be constructed in time polynomial in n and m
- Hence, this is a polynomial time reduction

Transitivity of Reduction

We used the following techniques for reduction

- Simple Equivalence
- Special Case to General Case
- Encoding with Gadgets

A very powerful technique is to exploit transitivity of reductions

Theorem: If $X \leq_p Y$ and $Y \leq_p Z$, then $X \leq_p Z$

3-Sat $\leq_p$ Independent Set $\leq_p$ Minimum Vertex Cover $\leq_p$ Set Cover

Thanks a lot

If you are taking a Nap, **wake up**........Lecture Over