Department of
Computer Science and Engineering
计算机科学与工程系

CS215 DISCRETE MATH

Dr. Qi Wang

Department of Computer Science and Engineering
Office: Room413, CoE South Tower
Email: wangqi@sustech.edu.cn

NP-complete Problems

- Class **NP** vs Class **P**
  - **P**: decision problems solvable in polynomial time
  - **NP**: decision problems with certificates verifiable in polynomial time (polynomial time verification)

NP-complete Problems

- Class **NP** vs Class **P**
  - **P**: decision problems solvable in polynomial time
  - **NP**: decision problems with certificates verifiable in polynomial time (polynomial time verification)

- Some examples in Class NP, but will focus on intuition
  More reading:
  CLRS / M. Sipser: Introduction to Theory of Computation

NP-complete Problems

- Class **NP** vs Class **P**
  – **P**: decision problems solvable in polynomial time
  – **NP**: decision problems with certificates verifiable in polynomial time (polynomial time verification)

- Some examples in Class NP, but will focus on intuition
  More reading:
  CLRS / M. Sipser: Introduction to Theory of Computation

- Approximation Algorithm
  Natural idea: settle for *non-optimal* solutions for these "hard" problems, if we can find such close-to-the-optimal solutions reasonably fast.

Satisfiability Problem

- Satisfiability (*SAT*) – one of the most important **NP** problems

Satisfiability Problem

- Satisfiability (*SAT*) – one of the most important **NP** problems
- **Definition** A *Boolean formula* is a logical formula consisting of
  - Boolean variables (0 = false, 1 = true),
  - logical operations
    - $\neg x$: Negation
    - $x \vee y$: Disjunction
    - $x \wedge y$: Conjunction

  With the truth table defined by:

x	y	$\neg x$	$x \vee y$	$x \wedge y$
0	0	1	0	0
0	1		1	0
1	0	0	1	0
1	1		1	1

Satisfiable

- A given Boolean formula is *satisfiable* if there is a way to assign truth values (0 or 1) to the variables s.t. the final result is 1.

Satisfiable

- A given Boolean formula is *satisfiable* if there is a way to assign truth values (0 or 1) to the variables s.t. the final result is 1.

Example. $f(x, y, z) = (x \wedge (y \vee \neg z)) \vee (\neg y \wedge z \wedge \neg x)$

x	y	z	$(x \wedge (y \vee \neg z))$	$(\neg y \wedge z \wedge \neg x)$	$f(x, y, z)$
0	0	0	0	0	0
0	0	1	0	1	1
0	1	0	0	0	0
0	1	1	0	0	0
1	0	0	1	0	1
1	0	1	0	0	0
1	1	0	1	0	1
1	1	1	1	0	1

Satisfiable

- A given Boolean formula is *satisfiable* if there is a way to assign truth values (0 or 1) to the variables s.t. the final result is 1.

Example. $f(x, y, z) = (x \wedge (y \vee \neg z)) \vee (\neg y \wedge z \wedge \neg x)$

x	y	z	$(x \wedge (y \vee \neg z))$	$(\neg y \wedge z \wedge \neg x)$	$f(x, y, z)$
0	0	0	0	0	0
0	0	1	0	1	1
0	1	0	0	0	0
0	1	1	0	0	0
1	0	0	1	0	1
1	0	1	0	0	0
1	1	0	1	0	1
1	1	1	1	0	1

The assignment, $x = 1, y = 1, z = 0$ makes $f(x, y, z)$ true, and hence it is satisfiable.

Satisfiable

- **Example.** $f(x, y) = (x \vee y) \wedge (\neg x \vee y) \wedge (x \vee \neg y) \wedge (\neg x \vee \neg y)$

x	y	$x \vee y$	$\neg x \vee y$	$x \vee \neg y$	$\neg x \vee \neg y$	$f(x, y)$
0	0	0	1	1	1	0
0	1	1	1	0	1	0
1	0	1	0	1	1	0
1	1	1	1	1	0	0

Satisfiable

- **Example**. $f(x,y) = (x \vee y) \wedge (\neg x \vee y) \wedge (x \vee \neg y) \wedge (\neg x \vee \neg y)$

x	y	$x \vee y$	$\neg x \vee y$	$x \vee \neg y$	$\neg x \vee \neg y$	$f(x,y)$
0	0	0	1	1	1	0
0	1	1	1	0	1	0
1	0	1	0	1	1	0
1	1	1	1	1	0	0

There is no assignment that makes $f(x,y)$ true, and hence it is NOT satisfiable.

Satisfiable

- **Example**. $f(x,y) = (x \vee y) \wedge (\neg x \vee y) \wedge (x \vee \neg y) \wedge (\neg x \vee \neg y)$

x	y	$x \vee y$	$\neg x \vee y$	$x \vee \neg y$	$\neg x \vee \neg y$	$f(x,y)$
0	0	0	1	1	1	0
0	1	1	1	0	1	0
1	0	1	0	1	1	0
1	1	1	1	1	0	0

There is no assignment that makes $f(x,y)$ true, and hence it is NOT satisfiable.
Boolean formulas in the form of $f(x,y)$ are called 2-conjunctive normal form (2-CNF).

Satisfiable

- **Example**. $f(x,y) = (x \vee y) \wedge (\neg x \vee y) \wedge (x \vee \neg y) \wedge (\neg x \vee \neg y)$

x	y	$x \vee y$	$\neg x \vee y$	$x \vee \neg y$	$\neg x \vee \neg y$	$f(x,y)$
0	0	0	1	1	1	0
0	1	1	1	0	1	0
1	0	1	0	1	1	0
1	1	1	1	1	0	0

There is no assignment that makes $f(x,y)$ true, and hence it is NOT satisfiable.

Boolean formulas in the form of $f(x,y)$ are called 2-conjunctive normal form (2-CNF).

Definition For a fixed k, Boolean formulas in the following form are called *k*-conjunctive normal form (k-CNF):

$$f_1 \wedge f_2 \wedge \cdots \wedge f_n$$

where each f_i is of the form $f_i = y_{i,1} \vee y_{i,2} \vee \cdots \vee y_{i,k}$, and each $y_{i,j}$ is a variable or the negation of a variable.

2SAT Problem

- **2SAT**
  Instance: A 2-CNF formula f
  Problem: To decide whether f is *satisfiable*

2SAT Problem

- **2SAT**
  Instance: A 2-CNF formula f
  Problem: To decide whether f is *satisfiable*

  Example a 2-CNF formula

$$(\neg x \vee y) \wedge (\neg y \vee z) \wedge (x \vee \neg z) \wedge (z \vee y)$$

2SAT Problem

- **2SAT**
  Instance: A 2-CNF formula f
  Problem: To decide whether f is *satisfiable*

  Example a 2-CNF formula

$$(\neg x \vee y) \wedge (\neg y \vee z) \wedge (x \vee \neg z) \wedge (z \vee y)$$

  Theorem **2SAT** $\in$ Class **P**

2SAT Problem

- **2SAT**
  Instance: A 2-CNF formula f
  Problem: To decide whether f is *satisfiable*

  Example a 2-CNF formula

$$(\neg x \vee y) \wedge (\neg y \vee z) \wedge (x \vee \neg z) \wedge (z \vee y)$$

  Theorem **2SAT** $\in$ Class **P**
  Proof. We will show how to solve 2SAT efficiently using path searches in graphs.

Path Searching in Graphs

- **Theorem** Given a graph $G = (V, E)$ and two vertices $u, v \in V$, finding if there is a path from u to v in the graph G is polynomial-time decidable.

Path Searching in Graphs

- **Theorem** Given a graph $G = (V, E)$ and two vertices $u, v \in V$, finding if there is a path from u to v in the graph G is polynomial-time decidable.

  Proof.
  Use some basic search algorithms in graph theory (DFS/BFS).

Graph Construction from Boolean Formula

- For a Boolean formula, use vertex to represent each variable and a negation of a variable
- There is an edge $(x, y) \in E$ if and only if there exists a clause equivalent to $(\neg x \vee y)$

Graph Construction from Boolean Formula

- For a Boolean formula, use vertex to represent each variable and a negation of a variable
- There is an edge $(x, y) \in E$ if and only if there exists a clause equivalent to $(\neg x \lor y)$

Example

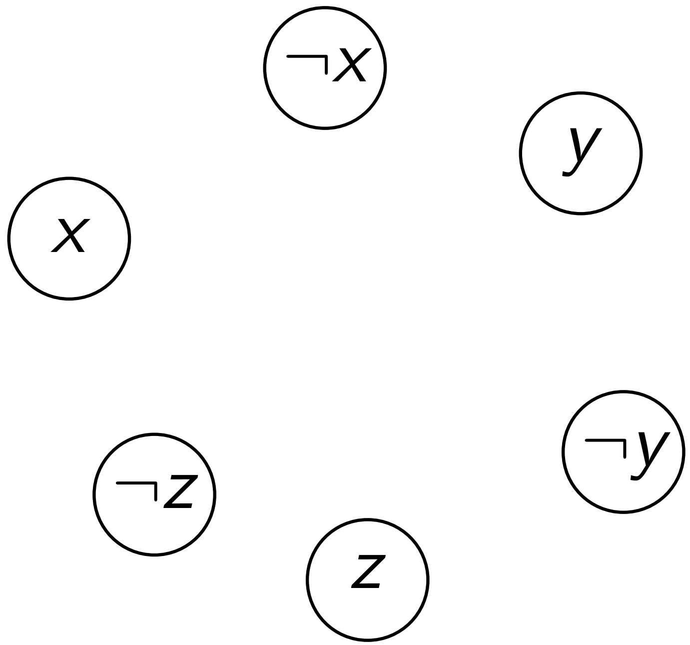
$$f = (\neg x \lor y) \land (\neg y \lor z) \land (x \lor \neg z) \land (z \lor y)$$

Graph Construction from Boolean Formula

- For a Boolean formula, use vertex to represent each variable and a negation of a variable
- There is an edge $(x, y) \in E$ if and only if there exists a clause equivalent to $(\neg x \vee y)$

Example

$$f = (\neg x \vee y) \wedge (\neg y \vee z) \wedge (x \vee \neg z) \wedge (z \vee y)$$

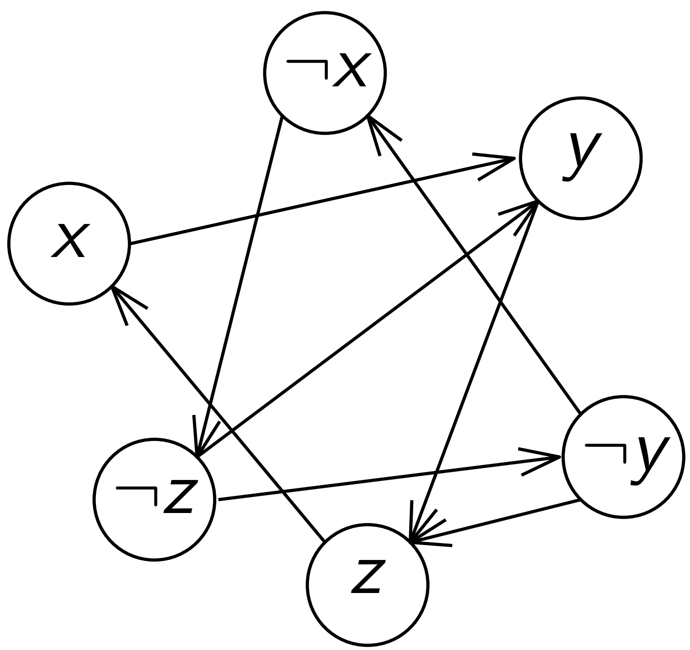


Graph Construction from Boolean Formula

- For a Boolean formula, use vertex to represent each variable and a negation of a variable
- There is an edge $(x, y) \in E$ if and only if there exists a clause equivalent to $(\neg x \vee y)$

Example

$$f = (\neg x \vee y) \wedge (\neg y \vee z) \wedge (x \vee \neg z) \wedge (z \vee y)$$



Observation

- **Claim** If the graph G contains a path from x to y, then it also contains a path from $\neg y$ to $\neg x$.

Observation

- **Claim** If the graph G contains a path from x to y, then it also contains a path from $\neg y$ to $\neg x$.

  Proof. If there is an edge $(x, y) \in E$, this means the clause $\neg x \vee y$ is included in the 2-CNF. The equivalent clause $y \vee \neg x$ implies that $(\neg y, \neg x) \in E$.

Observation

- **Claim** If the graph G contains a path from x to y, then it also contains a path from $\neg y$ to $\neg x$.

  Proof. If there is an edge $(x, y) \in E$, this means the clause $\neg x \vee y$ is included in the 2-CNF. The equivalent clause $y \vee \neg x$ implies that $(\neg y, \neg x) \in E$.

- **Claim** A 2-CNF formula f is unsatisfiable if and only if there exists a variable x such that:
  – there is a path from x to $\neg x$ in the graph G
  – there is a path from $\neg x$ to x in the graph G

Observation

- **Claim** If the graph G contains a path from x to y, then it also contains a path from $\neg y$ to $\neg x$.

  Proof. If there is an edge $(x, y) \in E$, this means the clause $\neg x \vee y$ is included in the 2-CNF. The equivalent clause $y \vee \neg x$ implies that $(\neg y, \neg x) \in E$.

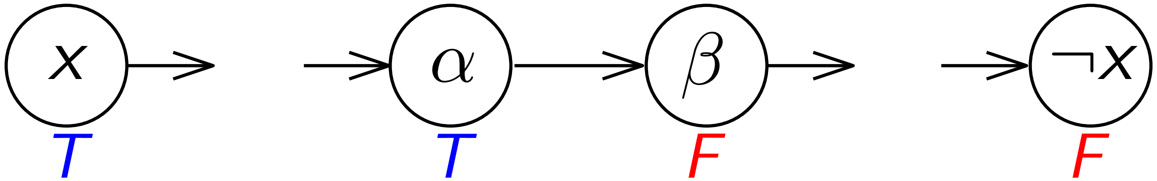
- **Claim** A 2-CNF formula f is unsatisfiable if and only if there exists a variable x such that:
  – there is a path from x to $\neg x$ in the graph G
  – there is a path from $\neg x$ to x in the graph G
  Proof. Suppose that there are paths $x \to \cdots \to \neg x$ and $\neg x \to \cdots \to x$ for some variable x. For any possible assignment ρ:
  If $\rho(x) = T$,

Observation

- **Claim** If the graph G contains a path from x to y, then it also contains a path from $\neg y$ to $\neg x$.

  Proof. If there is an edge $(x, y) \in E$, this means the clause $\neg x \vee y$ is included in the 2-CNF. The equivalent clause $y \vee \neg x$ implies that $(\neg y, \neg x) \in E$.

- **Claim** A 2-CNF formula f is unsatisfiable if and only if there exists a variable x such that:
  – there is a path from x to $\neg x$ in the graph G
  – there is a path from $\neg x$ to x in the graph G
  Proof. Suppose that there are paths $x \to \cdots \to \neg x$ and $\neg x \to \cdots \to x$ for some variable x. For any possible assignment ρ:
  If $\rho(x) = T$,

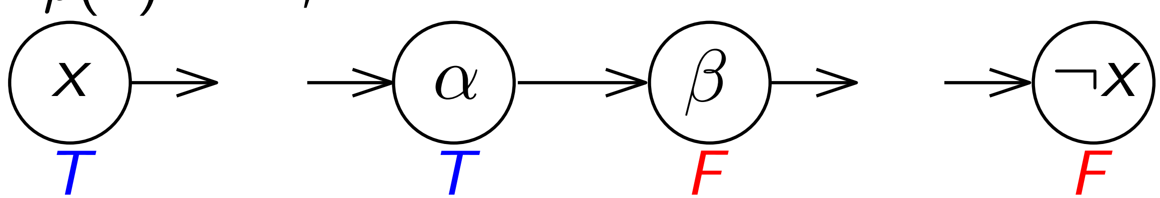


Observation

- **Claim** If the graph G contains a path from x to y, then it also contains a path from $\neg y$ to $\neg x$.

  Proof. If there is an edge $(x, y) \in E$, this means the clause $\neg x \vee y$ is included in the 2-CNF. The equivalent clause $y \vee \neg x$ implies that $(\neg y, \neg x) \in E$.

- **Claim** A 2-CNF formula f is unsatisfiable if and only if there exists a variable x such that:
  – there is a path from x to $\neg x$ in the graph G
  – there is a path from $\neg x$ to x in the graph G
  Proof. Suppose that there are paths $x \to \cdots \to \neg x$ and $\neg x \to \cdots \to x$ for some variable x. For any possible assignment ρ:
  If $\rho(x) = T$,



$(\neg\alpha \vee \beta)$ is false!

Correctness Proof II

- **Claim** A 2-CNF formula f is unsatisfiable if and only if there exists a variable x such that:
  – there is a path from x to $\neg x$ in the graph G
  – there is a path from $\neg x$ to x in the graph G

  Proof. (cont')
  Suppose that there are no such paths ($x \to \cdots \to \neg x$ and $\neg x \to \cdots \to x$). We will find a satisfiable assignment as follows.

Correctness Proof II

- **Claim** A 2-CNF formula f is unsatisfiable if and only if there exists a variable x such that:
  – there is a path from x to $\neg x$ in the graph G
  – there is a path from $\neg x$ to x in the graph G

  Proof. (cont')
  Suppose that there are no such paths ($x \to \cdots \to \neg x$ and $\neg x \to \cdots \to x$). We will find a satisfiable assignment as follows.
  – Pick an unassigned literal α, with no path from α to $\neg\alpha$, and assign it T
  – Assign T to all reachable vertices
  – Assign F to all their negations
  – Repeat until all vertices are assigned

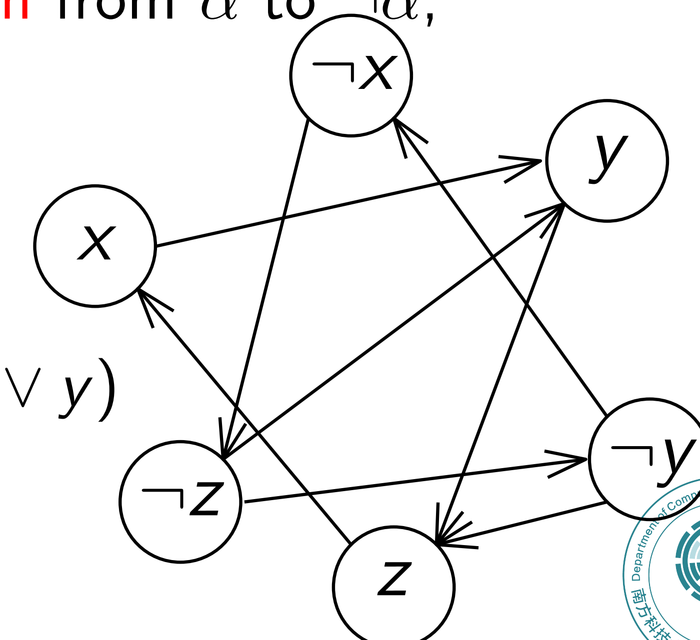
Correctness Proof II

- **Claim** A 2-CNF formula f is unsatisfiable if and only if there exists a variable x such that:
  – there is a path from x to $\neg x$ in the graph G
  – there is a path from $\neg x$ to x in the graph G

Proof. (cont')
Suppose that there are no such paths ($x \to \cdots \to \neg x$ and $\neg x \to \cdots \to x$). We will find a satisfiable assignment as follows.

– Pick an unassigned literal α, with no path from α to $\neg\alpha$, and assign it T
– Assign T to all reachable vertices
– Assign F to all their negations
– Repeat until all vertices are assigned

$f = (\neg x \vee y) \wedge (\neg y \vee z) \wedge (x \vee \neg z) \wedge (z \vee y)$



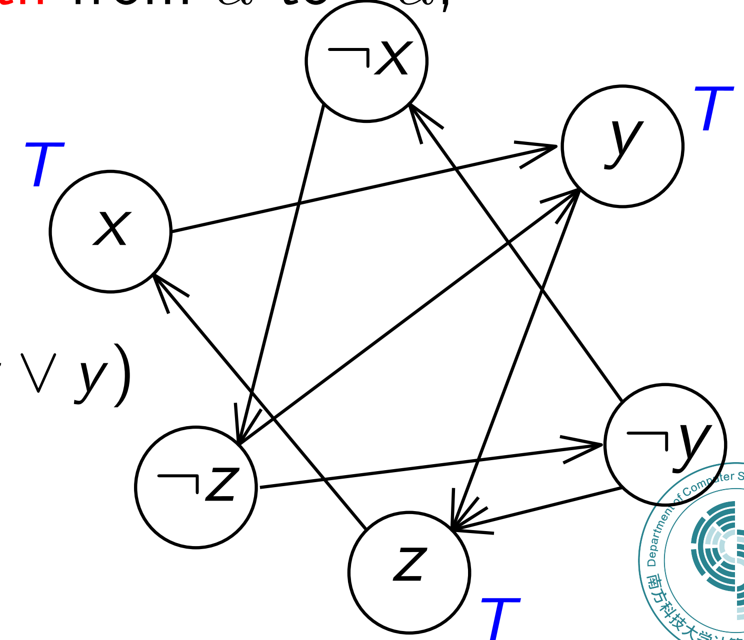
Correctness Proof II

- **Claim** A 2-CNF formula f is unsatisfiable if and only if there exists a variable x such that:
  – there is a path from x to $\neg x$ in the graph G
  – there is a path from $\neg x$ to x in the graph G

Proof. (cont')
Suppose that there are no such paths ($x \to \cdots \to \neg x$ and $\neg x \to \cdots \to x$). We will find a satisfiable assignment as follows.

– Pick an unassigned literal α, with no path from α to $\neg\alpha$, and assign it T
– Assign T to all reachable vertices
– Assign F to all their negations
– Repeat until all vertices are assigned

$$f = (\neg x \vee y) \wedge (\neg y \vee z) \wedge (x \vee \neg z) \wedge (z \vee y)$$



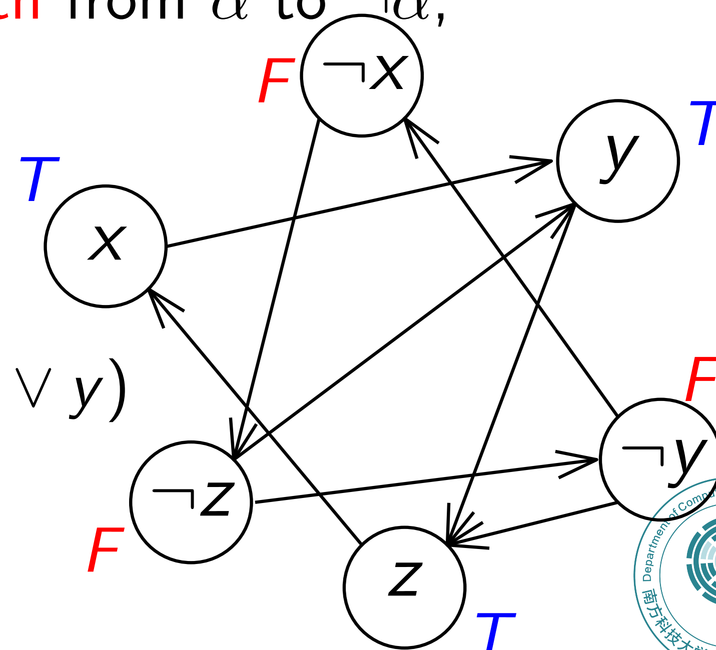
Correctness Proof II

- **Claim** A 2-CNF formula f is unsatisfiable if and only if there exists a variable x such that:
  – there is a path from x to $\neg x$ in the graph G
  – there is a path from $\neg x$ to x in the graph G

Proof. (cont')
Suppose that there are no such paths ($x \to \cdots \to \neg x$ and $\neg x \to \cdots \to x$). We will find a satisfiable assignment as follows.

– Pick an unassigned literal α, with no path from α to $\neg\alpha$, and assign it T
– Assign T to all reachable vertices
– Assign F to all their negations
– Repeat until all vertices are assigned

$f = (\neg x \vee y) \wedge (\neg y \vee z) \wedge (x \vee \neg z) \wedge (z \vee y)$



Correctness Proof II

- **Claim** The assignment algorithm is well defined.

Correctness Proof II

- **Claim** The assignment algorithm is well defined.

  Proof.
  There cannot be two paths from x to both y and $\neg y$ (this will lead to the same assignment for both y and $\neg y$, contradiction!).

Correctness Proof II

- **Claim** The assignment algorithm is well defined.

  Proof.
  There cannot be two paths from x to both y and $\neg y$ (this will lead to the same assignment for both y and $\neg y$, contradiction!).

  If so, there will be a path from x to $\neg y$, and also a path from $\neg y$ to $\neg x$ (why?). Put the two paths together, we will have a path from x to $\neg x$, contradiction!

Correctness Proof II

- **Claim** The assignment algorithm is well defined.

  Proof.
  There cannot be two paths from x to both y and $\neg y$ (this will lead to the same assignment for both y and $\neg y$, contradiction!).

  If so, there will be a path from x to $\neg y$, and also a path from $\neg y$ to $\neg x$ (why?). Put the two paths together, we will have a path from x to $\neg x$, contradiction!

- **Claim** A 2-CNF formula f is unsatisfiable if and only if there exists a variable x such that:
  – there is a path from x to $\neg x$ in the graph G
  – there is a path from $\neg x$ to x in the graph G

Correctness Proof II

- **Claim** The assignment algorithm is well defined.

  Proof.
  There cannot be two paths from x to both y and $\neg y$ (this will lead to the same assignment for both y and $\neg y$, contradiction!).

  If so, there will be a path from x to $\neg y$, and also a path from $\neg y$ to $\neg x$ (why?). Put the two paths together, we will have a path from x to $\neg x$, contradiction!

- **Claim** A 2-CNF formula f is unsatisfiable if and only if there exists a variable x such that:
  – there is a path from x to $\neg x$ in the graph G
  – there is a path from $\neg x$ to x in the graph G

  Theorem A 2-CNF formula f is satisfiable if and only if there are no paths from x to $\neg x$ or from $\neg x$ to x for any literal x.

2SAT $\in$ P

- An efficient algorithm for **2SAT** is the following.
  - In the constructed graph G, for each variable x, check whether there is a path from x to $\neg x$ and vice versa.
  - Output NO if any of these tests succeeds.
  - Output YES otherwise.

2SAT $\in$ P

- An efficient algorithm for **2SAT** is the following.
  - In the constructed graph G, for each variable x, check whether there is a path from x to $\neg x$ and vice versa.
  - Output NO if any of these tests succeeds.
  - Output YES otherwise.

- **Theorem** **2SAT** $\in$ Class **P**

Polynomial-Time Reduction

- Let L_1 and L_2 be two decision problems

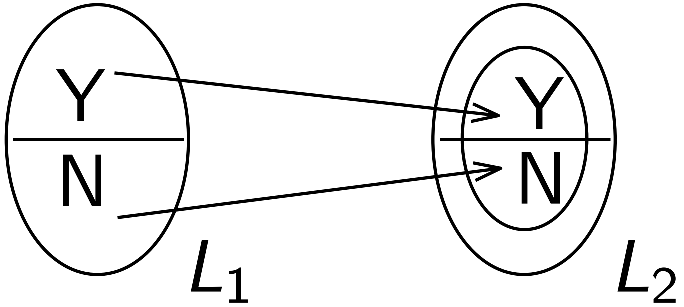
Polynomial-Time Reduction

- Let L_1 and L_2 be two decision problems
- A *polynomial-time reduction* from L_1 to L_2 is a transformation f with the following two properties:

  (1) f transforms an input x for L_1 into an input $f(x)$ for L_2 s.t.

    a yes-input of L_1 maps to a yes-input of L_2, and a no-input of L_1 maps to a no-input of L_2

  (2) f is computable in *polynomial time* in size(x)



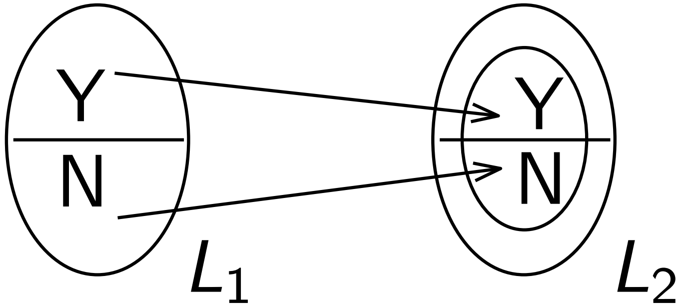
Polynomial-Time Reduction

- Let L_1 and L_2 be two decision problems
- A *polynomial-time reduction* from L_1 to L_2 is a transformation f with the following two properties:

(1) f transforms an input x for L_1 into an input $f(x)$ for L_2 s.t.

a yes-input of L_1 maps to a yes-input of L_2, and a no-input of L_1 maps to a no-input of L_2

(2) f is computable in *polynomial time* in size(x)



If such an f exists, we say that L_1 is *polynomial-time reducible* to L_2, and write $L_1 \leq_P L_2$.

Polynomial-Time Reduction

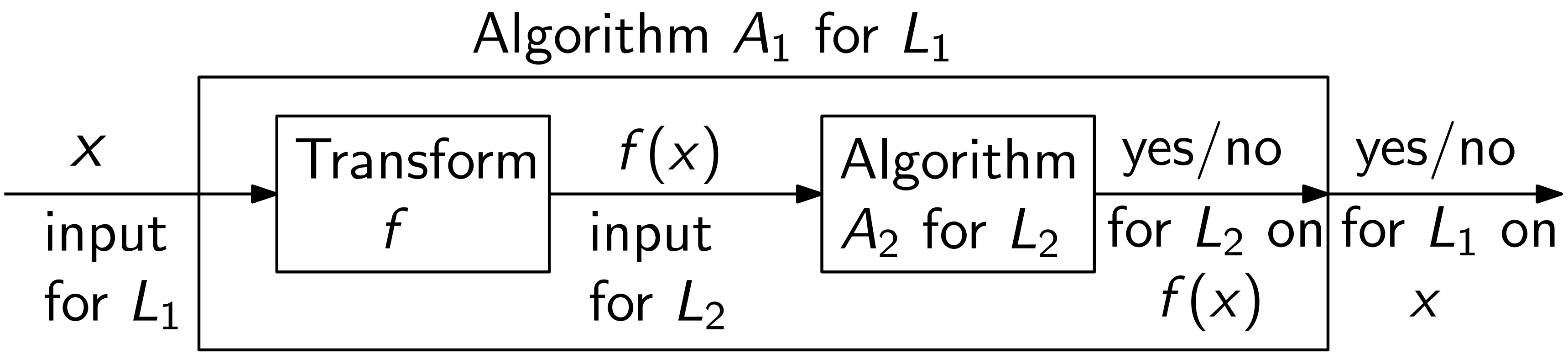
- Intuitively, $L_1 \leq_P L_2$ means that L_1 is no harder than L_2

Polynomial-Time Reduction

- Intuitively, $L_1 \leq_P L_2$ means that L_1 is no harder than L_2
- Given an algorithm A_2 for the decision problem L_2, we can develop an algorithm A_1 to solve L_1:

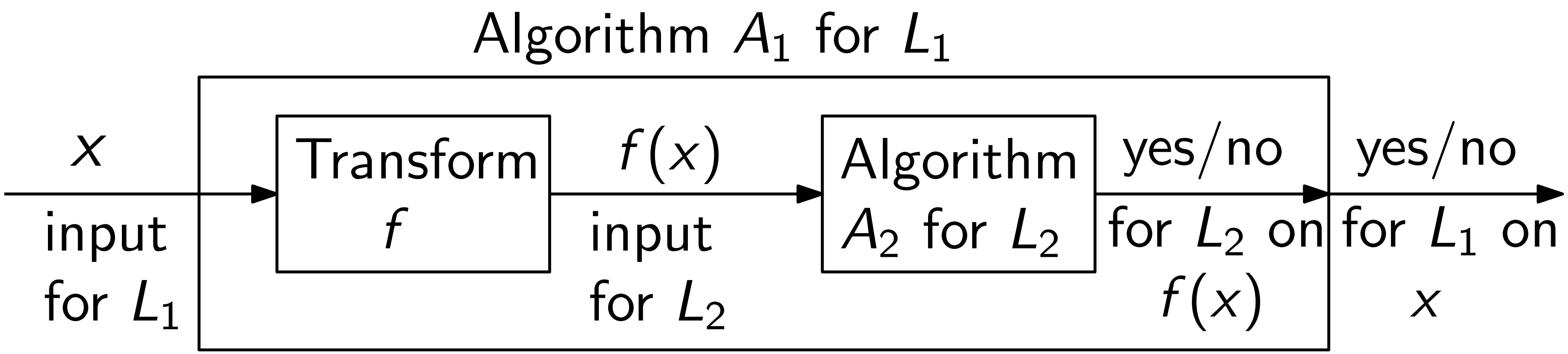
Polynomial-Time Reduction

- Intuitively, $L_1 \leq_P L_2$ means that L_1 is no harder than L_2
- Given an algorithm A_2 for the decision problem L_2, we can develop an algorithm A_1 to solve L_1:



Polynomial-Time Reduction

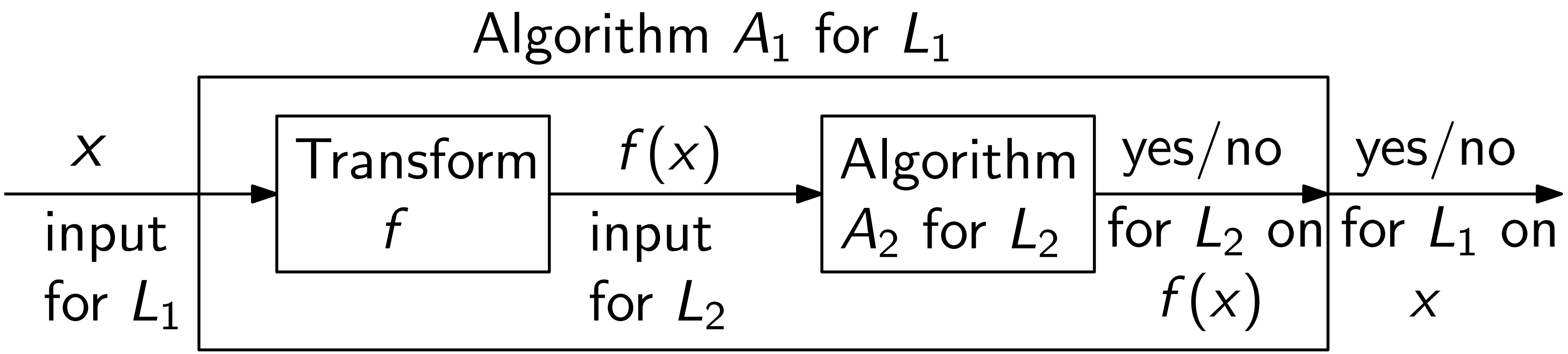
- Intuitively, $L_1 \leq_P L_2$ means that L_1 is no harder than L_2
- Given an algorithm A_2 for the decision problem L_2, we can develop an algorithm A_1 to solve L_1:



- If A_2 is polynomial-time algorithm, so is A_1

Polynomial-Time Reduction

- Intuitively, $L_1 \leq_P L_2$ means that L_1 is no harder than L_2
- Given an algorithm A_2 for the decision problem L_2, we can develop an algorithm A_1 to solve L_1:



- If A_2 is polynomial-time algorithm, so is A_1

Theorem If $L_1 \leq_P L_2$ and $L_2 \in P$, then $L_1 \in P$

Lemma If $L_1 \leq_P L_2$ and $L_2 \leq_P L_3$, then $L_1 \leq_P L_3$.

The Class NP-Complete (NPC)

- The Class *NPC* consists of all decision problems L s.t.
  (1) $L \in NP$
  (2) for every $L' \in NP$, $L' \leq_P L$

The Class NP-Complete (NPC)

- The Class *NPC* consists of all decision problems L s.t.
  (1) $L \in NP$
  (2) for every $L' \in NP$, $L' \leq_P L$

  From the definition of *NPC*, it seems impossible to prove that one decision problem $L \in NPC$.
  – By definition, it requires to show every $L' \in NP$, $L' \leq_P L$.
  – But there are infinitely many problems in NP, so how can we argue there exists a reduction from every L' to L?

The Class NP-Complete (NPC)

- The Class *NPC* consists of all decision problems L s.t.
  (1) $L \in NP$
  (2) for every $L' \in NP$, $L' \leq_P L$

  From the definition of *NPC*, it seems impossible to prove that one decision problem $L \in NPC$.
  – By definition, it requires to show every $L' \in NP$, $L' \leq_P L$.
  – But there are infinitely many problems in NP, so how can we argue there exists a reduction from every L' to L?

  However, due to the transitivity property of $\leq_P$, we can do the following to prove a decision problem $L \in NPC$:
  – prove $L \in NP$ (usually easy)
  – for some $L' \in NPC$, prove $L' \leq_P L$

The Class NP-Complete (NPC)

- The Class *NPC* consists of all decision problems L s.t.
  (1) $L \in NP$
  (2) for every $L' \in NP$, $L' \leq_P L$

From the definition of *NPC*, it seems impossible to prove that one decision problem $L \in NPC$.
– By definition, it requires to show every $L' \in NP$, $L' \leq_P L$.
– But there are infinitely many problems in NP, so how can we argue there exists a reduction from every L' to L?

However, due to the transitivity property of $\leq_P$, we can do the following to prove a decision problem $L \in NPC$:
– prove $L \in NP$ (usually easy)
– for some $L' \in NPC$, prove $L' \leq_P L$

Proof. Let L'' be any problem in NP. Since $L' \in NPC$, by definition we have $L'' \leq_P L'$. Since $L' \leq_P L$, then by transitivity, we have $L'' \leq_P L$.

SAT $\in NPC$ (Cook's Theorem)

- **Theorem** (Cook's Theorem) **SAT** $\in NPC$.

SAT $\in NPC$ (Cook's Theorem)

- **Theorem** (Cook's Theorem) **SAT** $\in NPC$.

  We will not prove this theorem, but will assume that **3SAT** $\in NPC$ as well. With this we will start to prove problems in Class **NPC**.

SAT $\in NPC$ (Cook's Theorem)

- **Theorem** (Cook's Theorem) **SAT** $\in NPC$.

  We will not prove this theorem, but will assume that **3SAT** $\in NPC$ as well. With this we will start to prove problems in Class **NPC**.

  We will prove:
  3SAT $\leq_P$ DCLIQUE
  DCLIQUE $\leq_P$ DVC

CLIQUE

- **Definition** A *clique* in an undirected graph $G = (V, E)$ is a subset $V' \subseteq V$ of vertices s.t. each pair $u, v \in V'$ is connected by an edge $(u, v) \in E$. In other words, a clique is a complete subgraph of G.

CLIQUE

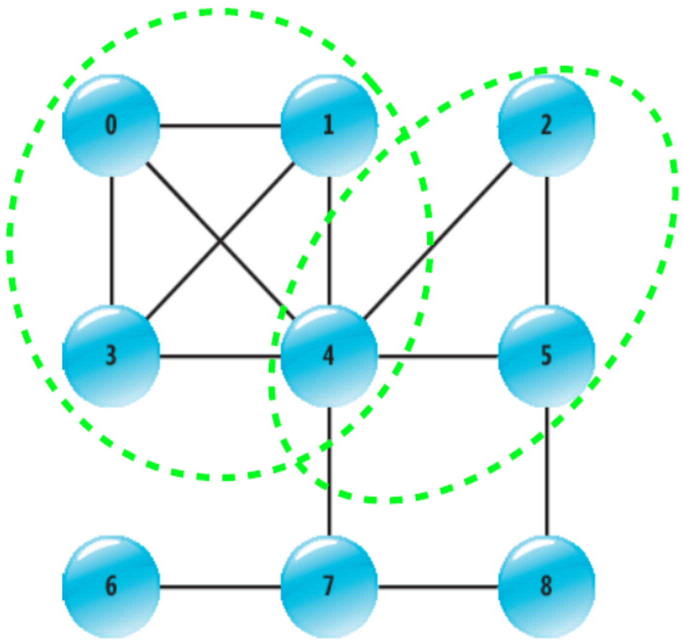
- **Definition** A *clique* in an undirected graph $G = (V, E)$ is a subset $V' \subseteq V$ of vertices s.t. each pair $u, v \in V'$ is connected by an edge $(u, v) \in E$. In other words, a clique is a complete subgraph of G.

  Example
  – a vertex is a clique of size 1
  – an edge is a clique of size 2

CLIQUE

- **Definition** A *clique* in an undirected graph $G = (V, E)$ is a subset $V' \subseteq V$ of vertices s.t. each pair $u, v \in V'$ is connected by an edge $(u, v) \in E$. In other words, a clique is a complete subgraph of G.

  Example
  – a vertex is a clique of size 1
  – an edge is a clique of size 2



CLIQUE

- The Problem CLIQUE
  Find a *clique* of maximum size in a graph G.
- The Problem DCLIQUE
  Given an undirected graph G and an integer k, determine whether G has a *clique* of size k.

CLIQUE

- The Problem CLIQUE
  Find a *clique* of maximum size in a graph G.
- The Problem DCLIQUE
  Given an undirected graph G and an integer k, determine whether G has a *clique* of size k.
- **Theorem** DCLIQUE $\in NPC$.

CLIQUE

- The Problem CLIQUE
  Find a *clique* of maximum size in a graph G.
- The Problem DCLIQUE
  Given an undirected graph G and an integer k, determine whether G has a *clique* of size k.
- **Theorem** DCLIQUE $\in NPC$.

  Proof. We need to show the following two:
  – DCLIQUE $\in NP$
  – There is some $L \in NPC$ s.t. $L \leq_P$ DCLIQUE

DCLIQUE $\in NP$

- **Claim** DCLIQUE $\in NP$.
  Proof. (easy)

DCLIQUE $\in NP$

- **Claim** DCLIQUE $\in NP$.
  Proof. (easy)
  – A *cerificate* will be a set of vertices $V' \subseteq V$ with $|V'| = k$ that is a possible *clique*.

DCLIQUE $\in NP$

- **Claim** DCLIQUE $\in NP$.
  Proof. (easy)
  - A *cerificate* will be a set of vertices $V' \subseteq V$ with $|V'| = k$ that is a possible *clique*.
  - To check that V' is a *clique*, all needed is to check that all edges (u, v) with $u \neq v$ and $u, v \in V'$, are in E.

DCLIQUE $\in NP$

- **Claim** DCLIQUE $\in NP$.
  Proof. (easy)
  - A *cerificate* will be a set of vertices $V' \subseteq V$ with $|V'| = k$ that is a possible *clique*.
  - To check that V' is a *clique*, all needed is to check that all edges (u, v) with $u \neq v$ and $u, v \in V'$, are in E.
  - This can be done in time $O(|V|^2)$, i.e., in polynomial time.

DCLIQUE $\in NP$

- **Claim** DCLIQUE $\in NP$.
  Proof. (easy)
  – A *cerificate* will be a set of vertices $V' \subseteq V$ with $|V'| = k$ that is a possible *clique*.
  – To check that V' is a *clique*, all needed is to check that all edges (u, v) with $u \neq v$ and $u, v \in V'$, are in E.
  – This can be done in time $O(|V|^2)$, i.e., in polynomial time.

- **Claim** 3SAT $\leq_P$ DCLIQUE.

DCLIQUE $\in NP$

- **Claim** DCLIQUE $\in NP$.
  Proof. (easy)
  – A *cerificate* will be a set of vertices $V' \subseteq V$ with $|V'| = k$ that is a possible *clique*.
  – To check that V' is a *clique*, all needed is to check that all edges (u, v) with $u \neq v$ and $u, v \in V'$, are in E.
  – This can be done in time $O(|V|^2)$, i.e., in polynomial time.

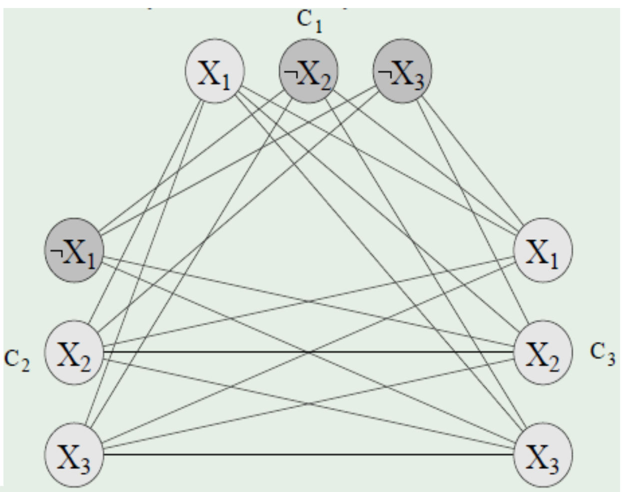
- **Claim** 3SAT $\leq_P$ DCLIQUE.

  We will define a polynomial transformation f from 3SAT to DCLIQUE $f : \phi \mapsto (G, k)$ that builds a graph G and integer k s.t. ϕ is a Yes-input to 3SAT if and only if (G, k) is a Yes-input to DCLIQUE.

- **Claim** 3SAT $\leq_P$ DCLIQUE.
  Proof.

3SAT $\leq_P$ DCLIQUE.

- **Claim** 3SAT $\leq_P$ DCLIQUE.
  Proof.
  Idea: for the k clauses input to 3SAT, draw literals as vertices, and all edges between vertices such that:
  - across clauses only (NO edges inside a clause)
  - not between x and $\neg x$

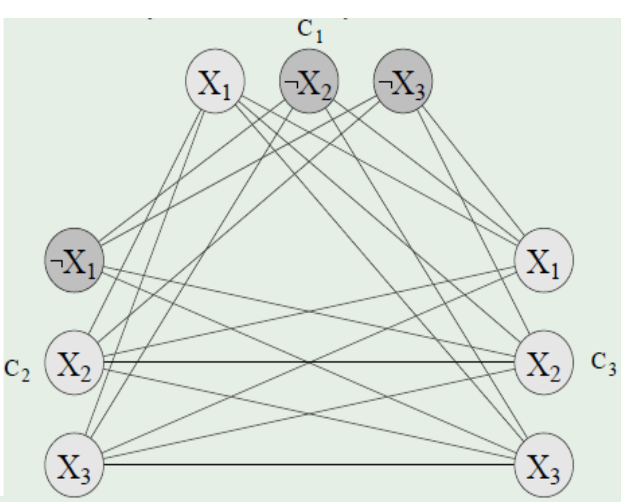


$$\phi = C_1 \wedge C_2 \wedge C_3$$

$$C_1 = (x_1 \vee \neg x_2 \vee \neg x_3)\,,\ C_2 = (\neg x_1 \vee x_2 \vee x_3)\,,\ C_3 = (x_1 \vee x_2 \vee x_3)$$

3SAT $\leq_P$ DCLIQUE.

- **Claim** 3SAT $\leq_P$ DCLIQUE.
  Proof.
  Idea: for the k clauses input to 3SAT, draw literals as vertices, and all edges between vertices such that:
  – across clauses only (NO edges inside a clause)
  – not between x and $\neg x$
  The reduction takes polynomial time
  A *satisfiable* assignment $\Rightarrow$ a *clique* of size k
  A *clique* of size k $\Rightarrow$ a *satisfiable* assignment



$$\phi = C_1 \wedge C_2 \wedge C_3$$
$$C_1 = (x_1 \vee \neg x_2 \vee \neg x_3),\ C_2 = (\neg x_1 \vee x_2 \vee x_3),\ C_3 = (x_1 \vee x_2 \vee x_3)$$

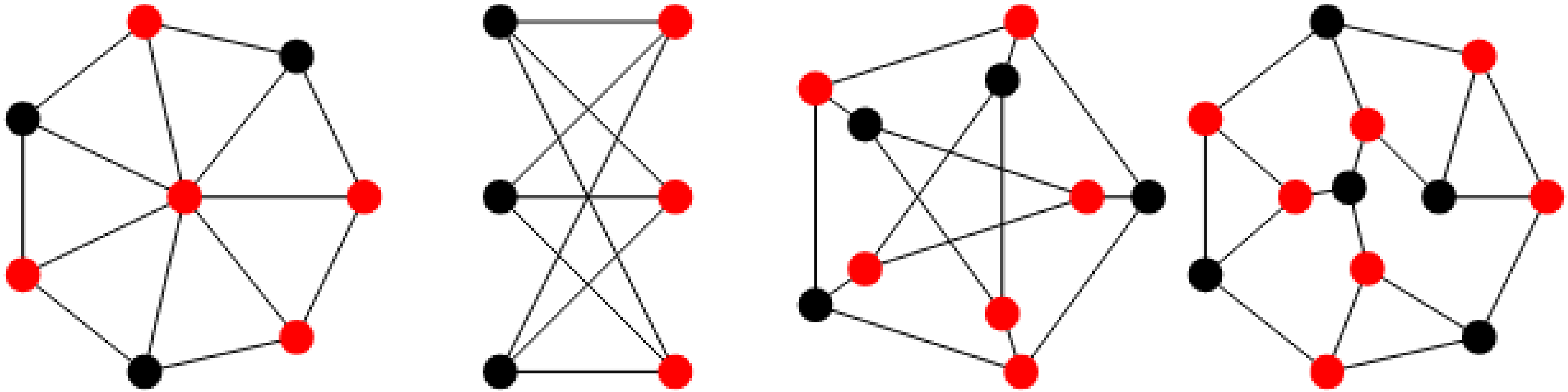
Vertex Cover

- **Definition** A *vertex cover* of G is a set of vertices such that every edge in G is incident at at least one of these vertices.

Vertex Cover

- **Definition** A *vertex cover* of G is a set of vertices such that every edge in G is incident at at least one of these vertices.

Example



Vertex Cover Problem

- The Vertex Cover Problem (VC)
  Given a graph G, find a vertex cover of G of minimum size.

- The Vertex Cover Problem (VC)
  Given a graph G, find a vertex cover of G of minimum size.
- The Problem DVC
  Given a graph G and an integer k, determine whether G has a *vertex cover* of with k vertices.

Vertex Cover Problem

- The Vertex Cover Problem (VC)
  Given a graph G, find a vertex cover of G of minimum size.
- The Problem DVC
  Given a graph G and an integer k, determine whether G has a *vertex cover* of with k vertices.
- **Theorem** DVC $\in NPC$.

Vertex Cover Problem

- The Vertex Cover Problem (VC)
  Given a graph G, find a vertex cover of G of minimum size.
- The Problem DVC
  Given a graph G and an integer k, determine whether G has a *vertex cover* of with k vertices.
- **Theorem** DVC $\in NPC$.

  Proof. We need to show the following two:
  – DVC $\in NP$
  – There is some $L \in NPC$ s.t. $L \leq_P$ DVC

DVC $\in NP$

- **Theorem** DVC $\in NP$.
  Proof. (easy)

DVC $\in NP$

- **Theorem** DVC $\in NP$.
  Proof. (easy)
  – A *cerificate* will be a set C of k vertices.

DVC $\in$ NP

- **Theorem** DVC $\in$ NP.

  Proof. (easy)

  – A *cerificate* will be a set C of k vertices.

  – The brute force method to check whether C is a vertex cover takes time $O(ke) = O((n+e)^2)$, in polynomial time.

DVC $\in NP$

- **Theorem** DVC $\in NP$.
  Proof. (easy)
  – A *cerificate* will be a set C of k vertices.
  – The brute force method to check whether C is a vertex cover takes time $O(ke) = O((n+e)^2)$, in polynomial time.

- **Claim** DCLIQUE $\leq_P$ DVC.

  We will define a polynomial transformation f from DCLIQUE to DVC.

DVC $\in NP$

- **Theorem** DVC $\in NP$.
  Proof. (easy)
  – A *cerificate* will be a set C of k vertices.
  – The brute force method to check whether C is a vertex cover takes time $O(ke) = O((n+e)^2)$, in polynomial time.

- **Claim** DCLIQUE $\leq_P$ DVC.

  We will define a polynomial transformation f from DCLIQUE to DVC.

Definition The *complement* of a graph $G = (V, E)$ is defined by $\overline{G} = (V, \overline{E})$ where

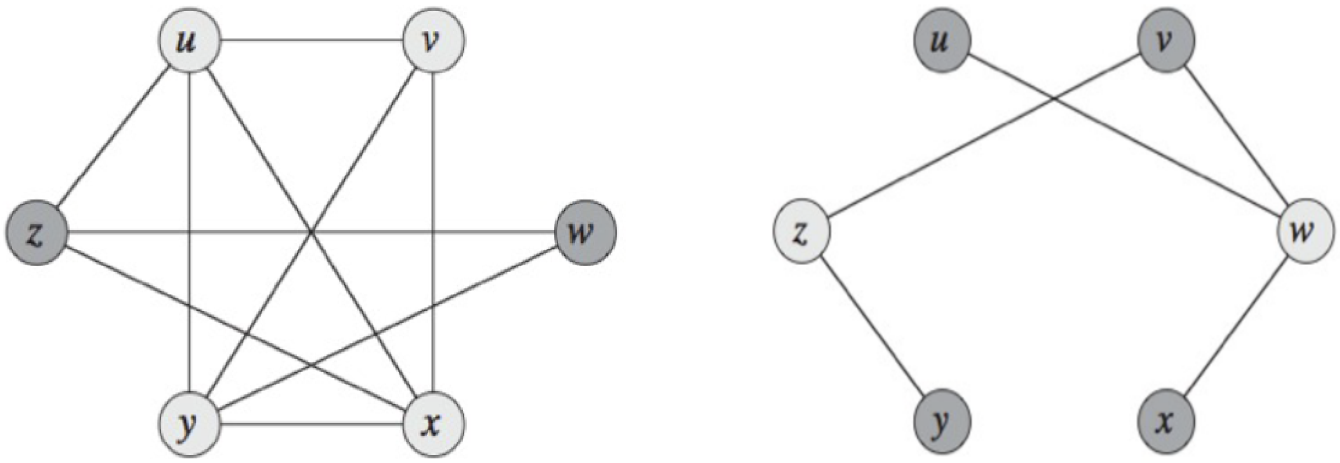
$$\overline{E} = \{(u, v) | u, v \in V, \ u \neq v, \ (u, v) \notin E\}.$$

DCLIQUE $\leq_P$ DVC

- **Theorem** DCLIQUE $\leq_P$ DVC.
  Proof.

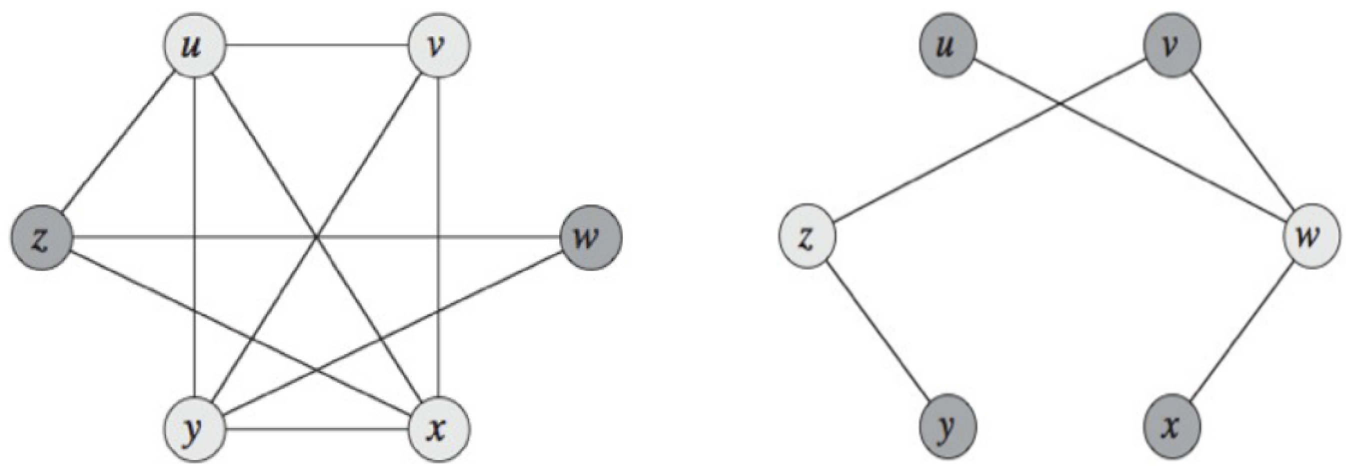
DCLIQUE $\leq_P$ DVC

- **Theorem** DCLIQUE $\leq_P$ DVC.
  Proof.
  Idea: start with the graph $G = (V, E)$ input of the DCLIQUE problem.
  – Construct the *complement graph* $\overline{G} = (V, \overline{E})$ by only considering the missing edges from E.



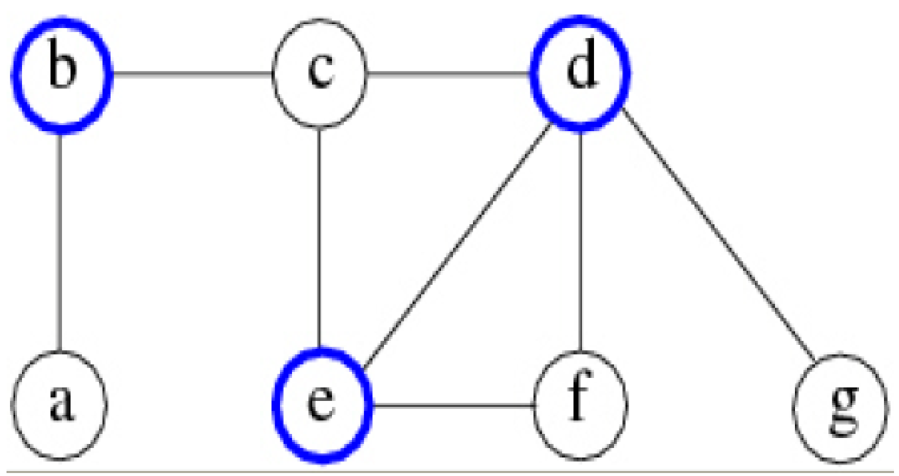
DCLIQUE $\leq_P$ DVC

- **Theorem** DCLIQUE $\leq_P$ DVC.
  Proof.
  Idea: start with the graph $G = (V, E)$ input of the DCLIQUE problem.
  – Construct the *complement graph* $\overline{G} = (V, \overline{E})$ by only considering the missing edges from E.
  The reduction takes polynomial time
  A *clique* of size k in G $\Rightarrow$ a *vertex cover* of size $|V| - k$ in $\overline{G}$
  A *vertex cover* of size k in $\overline{G}$ $\Rightarrow$ a *clique* of size $|V| - k$ in G



Approximation Algorithm Example: VC

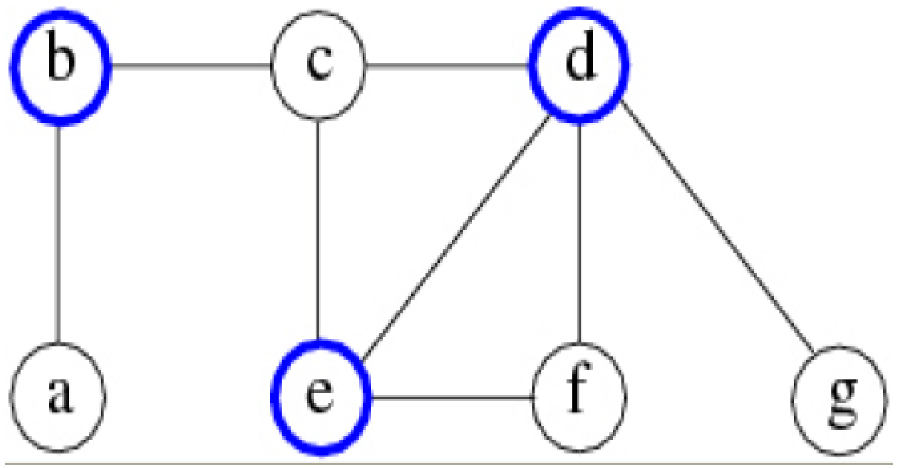
- DVC was proven NPC. Now we want to solve the *optimization version* of the *vertex cover* problem. We want to find a minimum size vertex cover of a given graph.



Approximation Algorithm Example: VC

- DVC was proven NPC. Now we want to solve the *optimization version* of the *vertex cover* problem. We want to find a minimum size vertex cover of a given graph.

  We call such a vertex cover an *optimal vertex cover* C^*.

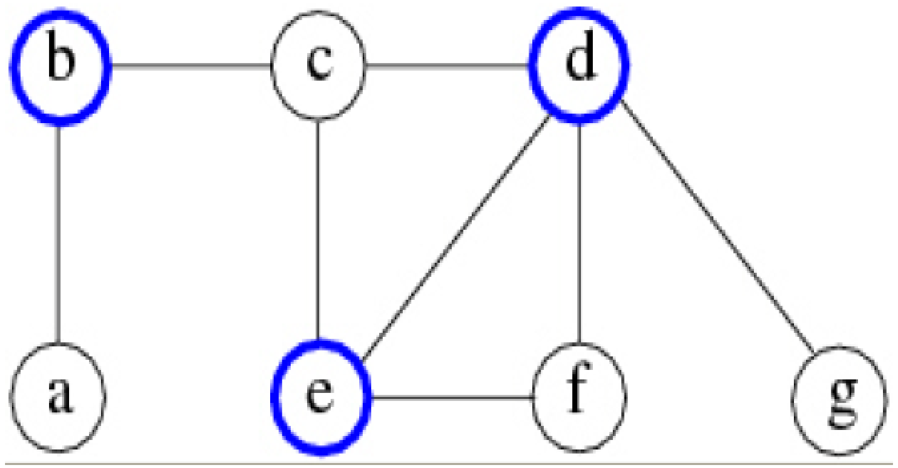


Approximation Algorithm Example: VC

- DVC was proven NPC. Now we want to solve the *optimization version* of the *vertex cover* problem. We want to find a minimum size vertex cover of a given graph.

  We call such a vertex cover an *optimal vertex cover* C^*.

  It is very unlikely to give an exact polynomial time algorithm (Why?)



An Approximation Algorithm for VC

Approx-Vertex-Cover(G=(V, E))

```
C = empty-set;
E'= E;
while  E' is not empty do do
    let (u, v) be any edge in E'          (*);
    add u and v to C;
    remove from E' all edges incident to u or v;
end
return C;
```

An Approximation Algorithm for VC

Approx-Vertex-Cover(G=(V, E))

```
C = empty-set;
E'= E;
while E' is not empty do do
    let (u, v) be any edge in E'          (*);
    add u and v to C;
    remove from E' all edges incident to u or v;
end
return C;
```

Idea: Take edges (u, v) one by one, put BOTH vertices into C, and remove all edges incident to u or v. We carry on until all edges have been removed. Obviously, C is a VC.

An Approximation Algorithm for VC

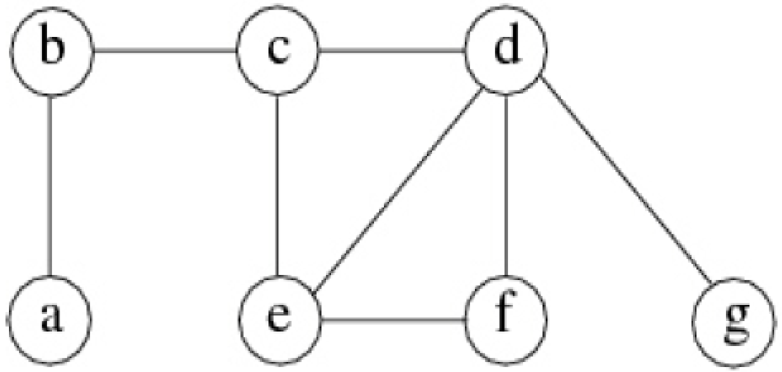
Approx-Vertex-Cover(G=(V, E))

```
C = empty-set;
E'= E;
while E' is not empty do do
    let (u, v) be any edge in E'        (*);
    add u and v to C;
    remove from E' all edges incident to u or v;
end
return C;
```

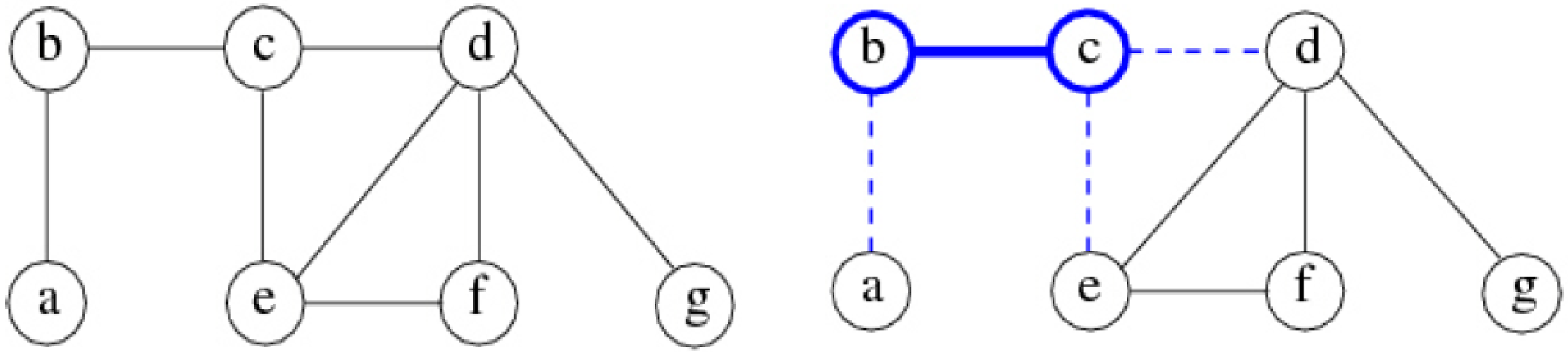
Idea: Take edges (u, v) one by one, put BOTH vertices into C, and remove all edges incident to u or v. We carry on until all edges have been removed. Obviously, C is a VC.

But, how good is C?

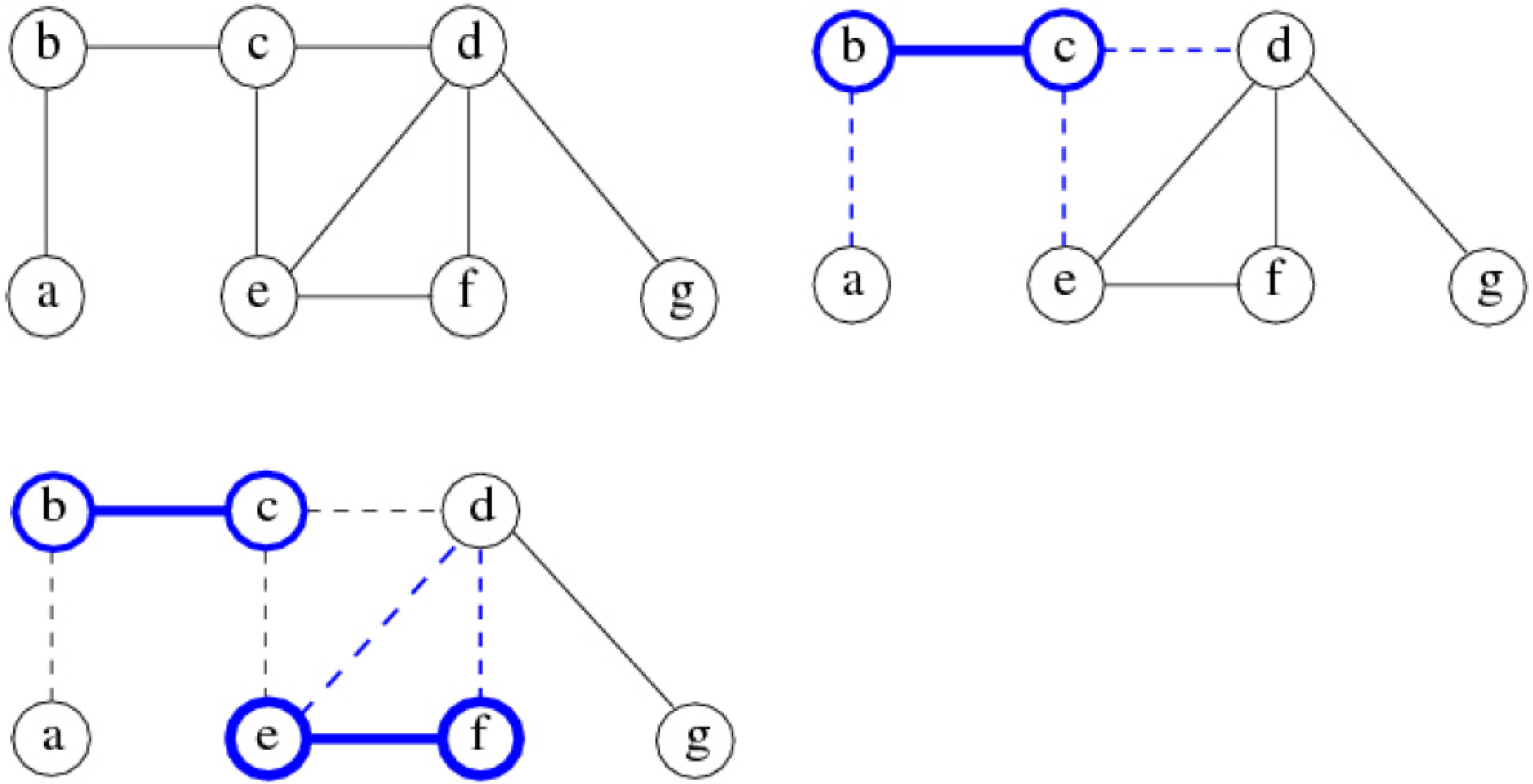
Example



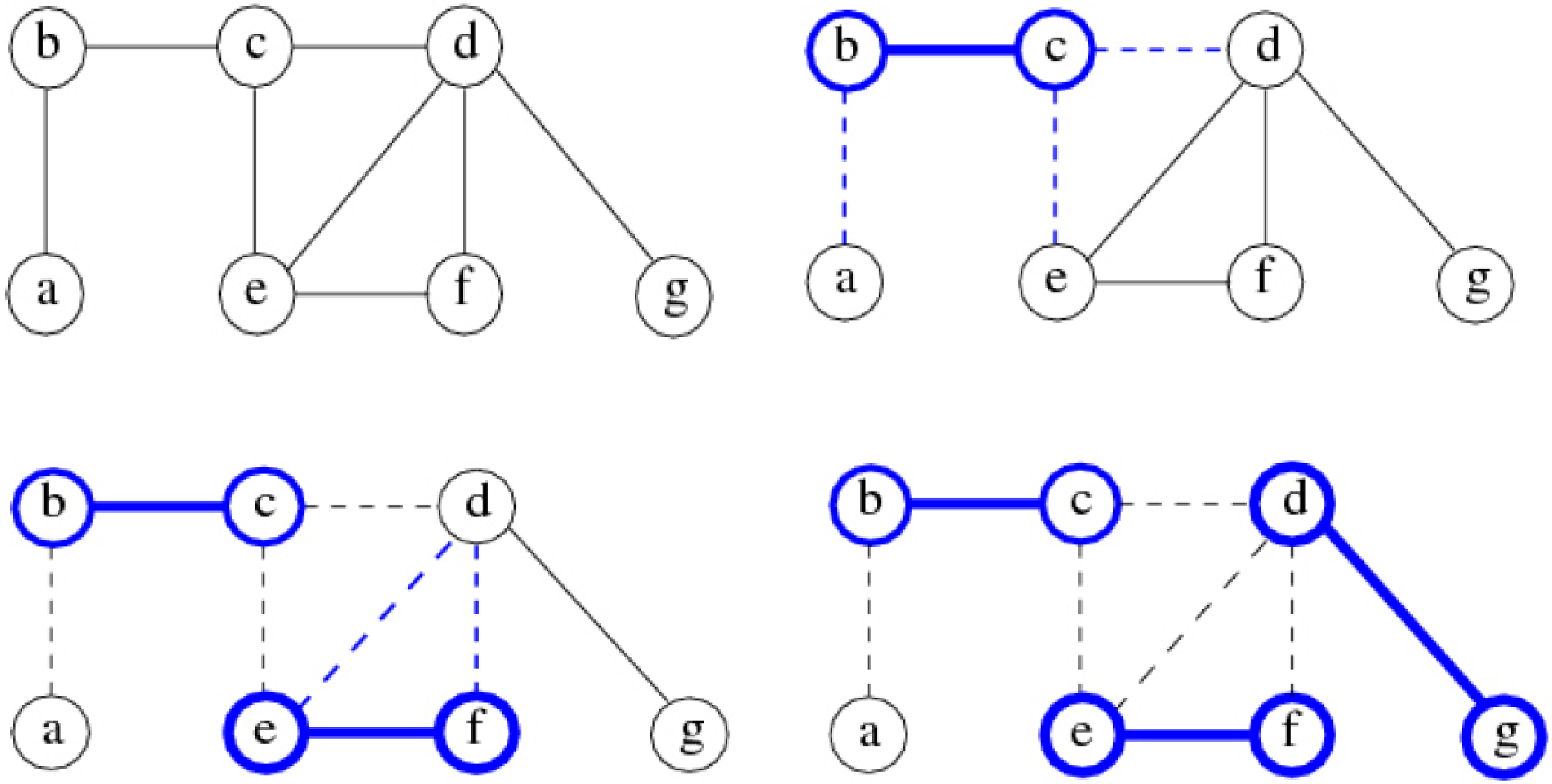
Example



Example



Example



Approximate Vertex Cover

- **Claim** Approx-Vertex-Cover is a 2-approximation algorithm, i.e.,

$$\frac{|C|}{|C^*|} \le 2.$$

Approximate Vertex Cover

- **Claim** Approx-Vertex-Cover is a 2-approximation algorithm, i.e.,

$$\frac{|C|}{|C^*|} \leq 2.$$

Proof.
Observation: The set of edges picked by this algorithm is a *maximal mathching* M: no two edges touch each other.

- **Claim** Approx-Vertex-Cover is a 2-approximation algorithm, i.e.,

$$\frac{|C|}{|C^*|} \leq 2.$$

Proof.
Observation: The set of edges picked by this algorithm is a *maximal mathching* M: no two edges touch each other.

The optimal vertex cover C^* must cover every edge in M, so $|C^*| \geq |M|$. But notice that the algorithm returns a vertex set of size $2|M|$. Therefore, we have

$$|C| = 2|M| \leq 2|C^*|.$$

Field

- A *field* is a set $\mathbb{F}$ equipped with two operations, *addition* $(+)$ and *multiplication* $(\cdot)$, and two special elements $0, 1$, s.t.:
  - $(\mathbb{F}, +)$ is an *abelian group* with identity element 0
  - $(\mathbb{F}^*, \cdot)$ is an *abelian group* with identity element 1
  - For all $a \in \mathbb{F}$, $0 \cdot a = a \cdot 0 = 0$
  - *Distributivity*: for all $a, b, c \in \mathbb{F}$, $a \cdot (b + c) = a \cdot b + a \cdot c$

Field

- A *field* is a set $\mathbb{F}$ equipped with two operations, *addition* $(+)$ and *multiplication* $(\cdot)$, and two special elements $0, 1$, s.t.:
  - $(\mathbb{F}, +)$ is an *abelian group* with identity element 0
  - $(\mathbb{F}^*, \cdot)$ is an *abelian group* with identity element 1
  - For all $a \in \mathbb{F}$, $0 \cdot a = a \cdot 0 = 0$
  - *Distributivity*: for all $a, b, c \in \mathbb{F}$, $a \cdot (b + c) = a \cdot b + a \cdot c$
- If $\mathbb{F}$ is finite, $\mathbb{F}$ is called a *finite field*.

Field

- A *field* is a set $\mathbb{F}$ equipped with two operations, *addition* $(+)$ and *multiplication* $(\cdot)$, and two special elements $0, 1$, s.t.:
  - $(\mathbb{F}, +)$ is an *abelian group* with identity element 0
  - $(\mathbb{F}^*, \cdot)$ is an *abelian group* with identity element 1
  - For all $a \in \mathbb{F}$, $0 \cdot a = a \cdot 0 = 0$
  - *Distributivity*: for all $a, b, c \in \mathbb{F}$, $a \cdot (b + c) = a \cdot b + a \cdot c$
- If $\mathbb{F}$ is finite, $\mathbb{F}$ is called a *finite field*.
- $\mathbb{F}_p = \mathbb{Z}/p\mathbb{Z} = \{0, 1, \ldots, p - 1\}$ with the operations *addition*, *multiplication* of integers modulo p, is called a *prime field*

Field

- A *field* is a set $\mathbb{F}$ equipped with two operations, *addition* $(+)$ and *multiplication* $(\cdot)$, and two special elements $0, 1$, s.t.:
  - $(\mathbb{F}, +)$ is an *abelian group* with identity element 0
  - $(\mathbb{F}^*, \cdot)$ is an *abelian group* with identity element 1
  - For all $a \in \mathbb{F}$, $0 \cdot a = a \cdot 0 = 0$
  - *Distributivity*: for all $a, b, c \in \mathbb{F}$, $a \cdot (b + c) = a \cdot b + a \cdot c$
- If $\mathbb{F}$ is finite, $\mathbb{F}$ is called a *finite field*.
- $\mathbb{F}_p = \mathbb{Z}/p\mathbb{Z} = \{0, 1, \ldots, p-1\}$ with the operations *addition*, *multiplication* of integers modulo p, is called a *prime field*
  - The properties can be verified

  Every $a \in \mathbb{F}_p^*$ has a *multiplicative inverse*: since $a \in \mathbb{F}_p^*$ and p is a prime, we have $\gcd(a, p) = 1$, and by extended Euclidean algorithm, there exist x, y s.t. $ax + py = 1$, and then $x = a^{-1} \bmod p$.

Prime Field and Characteristic

- Consider a *finite field* $\mathbb{F}$, define $S_r = 1 + 1 + \cdots + 1$ as sum of r 1's for a positive integer r

Prime Field and Characteristic

- Consider a *finite field* $\mathbb{F}$, define $S_r = 1 + 1 + \cdots + 1$ as sum of r 1's for a positive integer r
  - Let p be the smallest positive number with $S_p = 0$. If such a p exists, it must be *prime*
  - If $p = a \cdot b$ with $0 < a, b < p$, then by *distributivity*, $0 = S_p = S_a \cdot S_b$. Then one of S_a, S_b must be 0, contradicting the minimality of p.

Prime Field and Characteristic

- Consider a *finite field* $\mathbb{F}$, define
  $S_r = 1 + 1 + \cdots + 1$ as sum of r 1's for a positive integer r
  - Let p be the smallest positive number with $S_p = 0$.
    If such a p exists, it must be *prime*
  - If $p = a \cdot b$ with $0 < a, b < p$, then by *distributivity*,
    $0 = S_p = S_a \cdot S_b$. Then one of S_a, S_b must be 0,
    contradicting the minimality of p.
- This p is called the *characteristic* of the field $\mathbb{F}$.

Prime Field and Characteristic

- Consider a *finite field* $\mathbb{F}$, define $S_r = 1 + 1 + \cdots + 1$ as sum of r 1's for a positive integer r
  - Let p be the smallest positive number with $S_p = 0$. If such a p exists, it must be *prime*
  - If $p = a \cdot b$ with $0 < a, b < p$, then by *distributivity*, $0 = S_p = S_a \cdot S_b$. Then one of S_a, S_b must be 0, contradicting the minimality of p.
- This p is called the *characteristic* of the field $\mathbb{F}$.
- The subset $\{0, S_1, S_2, \ldots, S_{p-1}\} \subseteq \mathbb{F}$ is *isomorphic* to $\mathbb{F}$ (prime field)

Prime Field and Characteristic

- Consider a *finite field* $\mathbb{F}$, define $S_r = 1 + 1 + \cdots + 1$ as sum of r 1's for a positive integer r
  - Let p be the smallest positive number with $S_p = 0$. If such a p exists, it must be *prime*
  - If $p = a \cdot b$ with $0 < a, b < p$, then by *distributivity*, $0 = S_p = S_a \cdot S_b$. Then one of S_a, S_b must be 0, contradicting the minimality of p.
- This p is called the *characteristic* of the field $\mathbb{F}$.
- The subset $\{0, S_1, S_2, \ldots, S_{p-1}\} \subseteq \mathbb{F}$ is *isomorphic* to $\mathbb{F}$ (prime field)
- Any finite field $\mathbb{F}$ is a *finite dimensional vector space* over $\mathbb{F}_p$, with $n = \dim_{\mathbb{F}_p}(\mathbb{F})$, $|\mathbb{F}| = p^n$, i.e., the cardinality of $\mathbb{F}$ must be a prime power.

Finite Fields

- **Uniqueness** of finite fields:
  For any prime power q, there is essentially only one finite field of order q. Any two finite fields of order q are the same except that the labelling used to represent the field elements may be different

Finite Fields

- **Uniqueness** of finite fields:
  For any prime power q, there is essentially only one finite field of order q. Any two finite fields of order q are the same except that the labelling used to represent the field elements may be different

- *Binary field* – *characteristic-2* finite fields $\mathbb{F}_{2^m}$
  - Elements are polynomials over $\mathbb{F}_2$ of degree $\leq m-1$
  - $\mathbb{F}_{2^m} := \{a_{m-1}x^{m-1} + a_{m-2}x^{m-2} + \cdots + a_2x^2 + a_1x + a_0 : \ a_i \in \mathbb{F}_2\}$

Finite Fields

- **Uniqueness** of finite fields:
  For any prime power q, there is essentially only one finite field of order q. Any two finite fields of order q are the same except that the labelling used to represent the field elements may be different

- *Binary field* – *characteristic-2* finite fields $\mathbb{F}_{2^m}$
  – Elements are polynomials over $\mathbb{F}_2$ of degree $\leq m-1$
  – $\mathbb{F}_{2^m} := \{a_{m-1}x^{m-1} + a_{m-2}x^{m-2} + \cdots + a_2x^2 + a_1x + a_0 : \ a_i \in \mathbb{F}_2\}$

- An *irreducible polynomial* $f(x)$ of degree m is chosen:
  $f(x)$ cannot be factered as a product of binary polynomials each of degree less than m
  – *Addition*: usual
  – *Multiplication*: modulo $f(x)$

Elements of Finite Fields

- An *irreducible polynomial* $f(x)$ of degree m
  - $f(x) = x^4 + 1$ over $\mathbb{F}_2$
  - $f(x) = x^4 + x^2 + 1$ over $\mathbb{F}_2$
  - $f(x) = x^4 + x + 1$ over $\mathbb{F}_2$

Elements of Finite Fields

- An *irreducible polynomial* $f(x)$ of degree m
  - $f(x) = x^4 + 1$ over $\mathbb{F}_2$ $= (x+1)^4$
  - $f(x) = x^4 + x^2 + 1$ over $\mathbb{F}_2 = (x^2 + x + 1)^2$
  - $f(x) = x^4 + x + 1$ over $\mathbb{F}_2$

Elements of Finite Fields

- An *irreducible polynomial* $f(x)$ of degree m
  - $f(x) = x^4 + 1$ over $\mathbb{F}_2$ $= (x+1)^4$ ✗
  - $f(x) = x^4 + x^2 + 1$ over $\mathbb{F}_2 = (x^2+x+1)^2$ ✗
  - $f(x) = x^4 + x + 1$ over $\mathbb{F}_2$ ✓

Elements of Finite Fields

- An *irreducible polynomial* $f(x)$ of degree m
  - $f(x) = x^4 + 1$ over $\mathbb{F}_2$ $= (x+1)^4$ ✗
  - $f(x) = x^4 + x^2 + 1$ over $\mathbb{F}_2 = (x^2 + x + 1)^2$ ✗
  - $f(x) = x^4 + x + 1$ over $\mathbb{F}_2$ ✓
- The elements of $\mathbb{F}_{2^4}$ are the 16 polynomials of degree ≤ 3

0	z^2	z^3	$z^3 + z^2$
1	$z^2 + 1$	$z^3 + 1$	$z^3 + z^2 + 1$
z	$z^2 + z$	$z^3 + z$	$z^3 + z^2 + z$
$z + 1$	$z^2 + z + 1$	$z^3 + z + 1$	$z^3 + z^2 + z + 1$

Elements of Finite Fields

- An *irreducible polynomial* $f(x)$ of degree m
  - $f(x) = x^4 + 1$ over $\mathbb{F}_2$ $= (x+1)^4$ ✗
  - $f(x) = x^4 + x^2 + 1$ over $\mathbb{F}_2 = (x^2 + x + 1)^2$ ✗
  - $f(x) = x^4 + x + 1$ over $\mathbb{F}_2$ ✓
- The elements of $\mathbb{F}_{2^4}$ are the 16 polynomials of degree ≤ 3

$$
\begin{array}{llll}
0 & z^2 & z^3 & z^3 + z^2 \\
1 & z^2 + 1 & z^3 + 1 & z^3 + z^2 + 1 \\
z & z^2 + z & z^3 + z & z^3 + z^2 + z \\
z + 1 & z^2 + z + 1 & z^3 + z + 1 & z^3 + z^2 + z + 1
\end{array}
$$

  - *Addition*: $(z^3 + z^2 + 1) + (z^2 + z + 1) = z^3 + z$
  - *Subtraction*: $(z^3 + z^2 + 1) - (z^2 + z + 1) = z^3 + z$
  - *Multiplication*: $(z^3 + z^2 + 1) \cdot (z^2 + z + 1) = z^5 + z + 1 = z^2 + 1$
  - *Inversion*: $(z^3 + z^2 + 1)^{-1} = z^2$
    since $(z^3 + z^2 + 1) \cdot z^2 = z^5 + z^4 + z^2 = 1 \bmod z^4 + z + 1$

Elements of Finite Fields

- The elements of $\mathbb{F}_{2^4}$ can be also represented in the following:
  Let α be a root of the irreducible polynomial $f(x) = x^4 + x + 1$, i.e., $\alpha^4 + \alpha + 1 = 0$.

Elements of Finite Fields

- The elements of $\mathbb{F}_{2^4}$ can be also represented in the following:
  Let α be a root of the irreducible polynomial $f(x) = x^4 + x + 1$, i.e., $\alpha^4 + \alpha + 1 = 0$.
  The α is a *generator* of the multiplicative group $(\mathbb{F}_{2^4}^*, \cdot)$.

$\alpha^0 = 1$	$\alpha^1 = \alpha$	α^2
α^3	$\alpha^4 = \alpha + 1$	$\alpha^5 = \alpha^2 + \alpha$
$\alpha^6 = \alpha^3 + \alpha^2$	$\alpha^7 = \alpha^3 + \alpha + 1$	$\alpha^8 = \alpha^2 + 1$
$\alpha^9 = \alpha^3 + \alpha$	$\alpha^{10} = \alpha^2 + \alpha + 1$	$\alpha^{11} = \alpha^3 + \alpha^2 + \alpha$
$\alpha^{12} = \alpha^3 + \alpha^2 + \alpha + 1$	$\alpha^{13} = \alpha^3 + \alpha^2 + 1$	$\alpha^{14} = \alpha^3 + 1$
$\alpha^{15} = 1$		

Elements of Finite Fields

- The elements of $\mathbb{F}_{2^4}$ can be also represented in the following:
  Let α be a root of the irreducible polynomial $f(x) = x^4 + x + 1$, i.e., $\alpha^4 + \alpha + 1 = 0$.

  The α is a *generator* of the multiplicative group $(\mathbb{F}_{2^4}^*, \cdot)$.

$\alpha^0 = 1$	$\alpha^1 = \alpha$	α^2
α^3	$\alpha^4 = \alpha + 1$	$\alpha^5 = \alpha^2 + \alpha$
$\alpha^6 = \alpha^3 + \alpha^2$	$\alpha^7 = \alpha^3 + \alpha + 1$	$\alpha^8 = \alpha^2 + 1$
$\alpha^9 = \alpha^3 + \alpha$	$\alpha^{10} = \alpha^2 + \alpha + 1$	$\alpha^{11} = \alpha^3 + \alpha^2 + \alpha$
$\alpha^{12} = \alpha^3 + \alpha^2 + \alpha + 1$	$\alpha^{13} = \alpha^3 + \alpha^2 + 1$	$\alpha^{14} = \alpha^3 + 1$
$\alpha^{15} = 1$		

  $< \alpha^3, \alpha^2, \alpha, 1 >$ is a *basis* for $\mathbb{F}_{2^4}$ over $\mathbb{F}_2$.

Elements of Finite Fields

- The elements of $\mathbb{F}_{2^4}$ can be also represented in the following:
  Let α be a root of the irreducible polynomial $f(x) = x^4 + x + 1$, i.e., $\alpha^4 + \alpha + 1 = 0$.

  The α is a *generator* of the multiplicative group $(\mathbb{F}_{2^4}^*, \cdot)$.

  $$
  \begin{array}{lll}
  \alpha^0 = 1 & \alpha^1 = \alpha & \alpha^2 \\
  \alpha^3 & \alpha^4 = \alpha + 1 & \alpha^5 = \alpha^2 + \alpha \\
  \alpha^6 = \alpha^3 + \alpha^2 & \alpha^7 = \alpha^3 + \alpha + 1 & \alpha^8 = \alpha^2 + 1 \\
  \alpha^9 = \alpha^3 + \alpha & \alpha^{10} = \alpha^2 + \alpha + 1 & \alpha^{11} = \alpha^3 + \alpha^2 + \alpha \\
  \alpha^{12} = \alpha^3 + \alpha^2 + \alpha + 1 & \alpha^{13} = \alpha^3 + \alpha^2 + 1 & \alpha^{14} = \alpha^3 + 1 \\
  \alpha^{15} = 1 & &
  \end{array}
  $$

  $< \alpha^3, \alpha^2, \alpha, 1 >$ is a *basis* for $\mathbb{F}_{2^4}$ over $\mathbb{F}_2$.
  The finite field $\mathbb{F}_{2^4}$ can be viewed as a *vector space* over $\mathbb{F}_2$.
  The finite field $\mathbb{F}_{q^n}$ can be viewed as a *vector space* over $\mathbb{F}_q$.

Isomorphism of Finite Fields

- For a fixed q, the finite field $\mathbb{F}_q$ is unique.

Isomorphism of Finite Fields

- For a fixed q, the finite field $\mathbb{F}_q$ is unique.
  But, there are different irreducible polynomials of degree 4 over $\mathbb{F}_2$.

  $$f_1(z) = z^4 + z + 1$$
  $$f_2(z) = z^4 + z^3 + 1$$
  $$f_3(z) = z^4 + z^3 + z^2 + z + 1$$

Isomorphism of Finite Fields

- For a fixed q, the finite field $\mathbb{F}_q$ is unique.

  But, there are different irreducible polynomials of degree 4 over $\mathbb{F}_2$.

  $$f_1(z) = z^4 + z + 1 \qquad K_1$$
  $$f_2(z) = z^4 + z^3 + 1 \qquad K_2$$
  $$f_3(z) = z^4 + z^3 + z^2 + z + 1 \qquad K_3$$

  Superficially, these three fields appear to be different:

  In K_1, $z^3 \cdot z = z + 1$;
  In K_2, $z^3 \cdot z = z^3 + 1$;
  In K_3, $z^3 \cdot z = z^3 + z^2 + z + 1$.

Isomorphism of Finite Fields

- For a fixed q, the finite field $\mathbb{F}_q$ is unique.

  But, there are different irreducible polynomials of degree 4 over $\mathbb{F}_2$.

  $$f_1(z) = z^4 + z + 1 \qquad K_1$$
  $$f_2(z) = z^4 + z^3 + 1 \qquad K_2$$
  $$f_3(z) = z^4 + z^3 + z^2 + z + 1 \qquad K_3$$

  Superficially, these three fields appear to be different:

  In K_1, $z^3 \cdot z = z + 1$;
  In K_2, $z^3 \cdot z = z^3 + 1$;
  In K_3, $z^3 \cdot z = z^3 + z^2 + z + 1$.

  However, all three fields of a given order q are *isomorphic*: the difference is only in the labelling of the elements.

Isomorphism of Finite Fields

- For a fixed q, the finite field $\mathbb{F}_q$ is unique.

  But, there are different irreducible polynomials of degree 4 over $\mathbb{F}_2$.

  $f_1(z) = z^4 + z + 1$ K_1
  $f_2(z) = z^4 + z^3 + 1$ K_2
  $f_3(z) = z^4 + z^3 + z^2 + z + 1$ K_3

  Superficially, these three fields appear to be different:

  In K_1, $z^3 \cdot z = z + 1$;
  In K_2, $z^3 \cdot z = z^3 + 1$;
  In K_3, $z^3 \cdot z = z^3 + z^2 + z + 1$.

  However, all three fields of a given order q are *isomorphic*: the difference is only in the labelling of the elements.

  If $\psi : z \mapsto c$ is an *ismorphism* between K_1 and K_2, then $f_1(c) \equiv 0 \pmod{f_2}$ for some $c \in K_2$. The choices for c are $z^2 + z$, $z^2 + z + 1$, $z^3 + z^2$, and $z^3 + z^2 + 1$.

Extension Fields and Subfields

- Let p be a prime and $m \geq 2$. Let $\mathbb{F}_p[z]$ denote the set of all polynomials in the variable z with coefficients from $\mathbb{F}_p$. Let $f(z)$ be an *irreducible polynomial of degree m in $\mathbb{F}_p[z]$*.

Extension Fields and Subfields

- Let p be a prime and $m \geq 2$. Let $\mathbb{F}_p[z]$ denote the set of all polynomials in the variable z with coefficients from $\mathbb{F}_p$. Let $f(z)$ be an *irreducible polynomial of degree m in $\mathbb{F}_p[z]$*.

  The elements of $\mathbb{F}_{p^m}$ are the polynomials in $\mathbb{F}_p[z]$ of degree $\leq m-1$:

  $$\mathbb{F}_{p^m} = \{a_{m-1}z^{m-1} + a_{m-2}z^{m-2} + \cdots + a_2z^2 + a_1z + a_0 : \ a_i \in \mathbb{F}_p\}.$$

Extension Fields and Subfields

- Let p be a prime and $m \geq 2$. Let $\mathbb{F}_p[z]$ denote the set of all polynomials in the variable z with coefficients from $\mathbb{F}_p$. Let $f(z)$ be an *irreducible polynomial of degree m in $\mathbb{F}_p[z]$*.

  The elements of $\mathbb{F}_{p^m}$ are the polynomials in $\mathbb{F}_p[z]$ of degree $\leq m-1$:

  $$\mathbb{F}_{p^m} = \{a_{m-1}z^{m-1} + a_{m-2}z^{m-2} + \cdots + a_2 z^2 + a_1 z + a_0 : \ a_i \in \mathbb{F}_p\}.$$

  – *Addition*: usual addition of polynomials, with coefficients arithmetic performed in $\mathbb{F}_p$.

  – *Multiplication*: performed modulo the polynomial $f(z)$.

Extension Fields and Subfields

- Let p be a prime and $m \geq 2$. Let $\mathbb{F}_p[z]$ denote the set of all polynomials in the variable z with coefficients from $\mathbb{F}_p$. Let $f(z)$ be an *irreducible polynomial of degree m in $\mathbb{F}_p[z]$*.

  The elements of $\mathbb{F}_{p^m}$ are the polynomials in $\mathbb{F}_p[z]$ of degree $\leq m-1$:

  $$\mathbb{F}_{p^m} = \{a_{m-1}z^{m-1} + a_{m-2}z^{m-2} + \cdots + a_2z^2 + a_1z + a_0 : \ a_i \in \mathbb{F}_p\}.$$

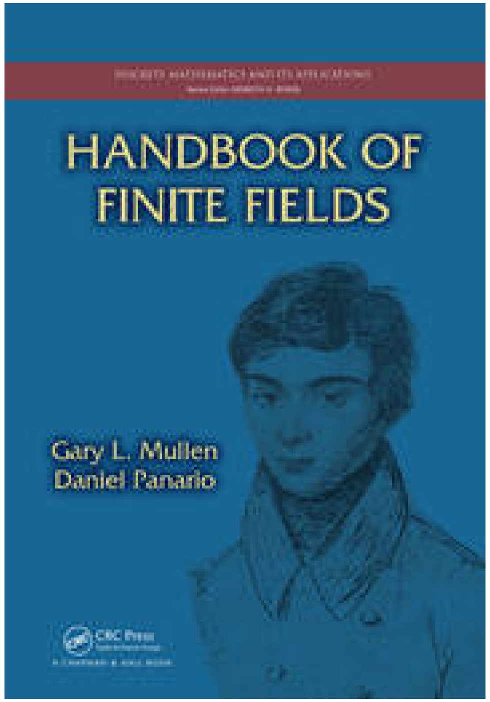
  – *Addition*: usual addition of polynomials, with coefficients arithmetic performed in $\mathbb{F}_p$.

  – *Multiplication*: performed modulo the polynomial $f(z)$.

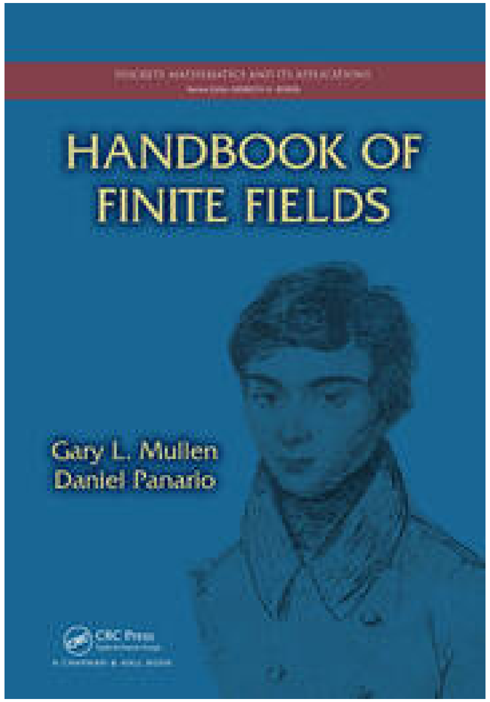
- A finite field $\mathbb{F}_{p^m}$ has precisely one subfield of order p^ℓ for each positive divisor ℓ of m.

  The elements of this subfield are the elements $a \in \mathbb{F}_{p^m}$ satisfying $a^{p^\ell} = a$; Conversely, every subfield of $\mathbb{F}_{p^m}$ has order p^ℓ for some positive divisor ℓ of m.

Applications of Finite Fields



Applications of Finite Fields



coding theory, cryptography, combinatorics, data storage systems, simulation, communications, signal design, ...

Review

01. Propositional Logic
02. Predicate Logic
03. Mathematical Proofs
04. Sets
05. Functions
06. Complexity of Algorithms
07. Number Theory
    Groups, Rings and Fields
08. Cryptography
09. Mathematical Induction
10. Recursion
11. Counting
12. Relation
13. Graphs
14. Tree

Review

01. Propositional Logic
02. Predicate Logic
03. Mathematical Proofs
04. Sets
05. Functions
06. Complexity of Algorithms
07. Number Theory
    Groups, Rings and Fields

    Discrete Probability

08. Cryptography
09. Mathematical Induction
10. Recursion
11. Counting
12. Relation
13. Graphs
14. Tree

Logic

- Logical connectives

- Logical connectives

  $\neg p,\ p \vee q,\ p \wedge q,\ p \oplus q,\ p \to q,\ p \leftrightarrow q$

Logic

- Logical connectives

  $\neg p,\ p \vee q,\ p \wedge q,\ p \oplus q,\ p \rightarrow q,\ p \leftrightarrow q$

- Logical equivalence

Logic

- Logical connectives

  $\neg p$, $p \vee q$, $p \wedge q$, $p \oplus q$, $p \rightarrow q$, $p \leftrightarrow q$

- Logical equivalence

  De Morgan's laws, communtative laws, distributive laws, ...

Logic

- Logical connectives

  $\neg p,\ p \vee q,\ p \wedge q,\ p \oplus q,\ p \rightarrow q,\ p \leftrightarrow q$

- Logical equivalence

  De Morgan's laws, communtative laws, distributive laws, ...

- Predicate logic

  contains variables

Logic

- Logical connectives

  $\neg p$, $p \vee q$, $p \wedge q$, $p \oplus q$, $p \to q$, $p \leftrightarrow q$

- Logical equivalence

  De Morgan's laws, communtative laws, distributive laws, ...

- Predicate logic

  contains variables

- Quantified statements

  universal, existential, equivalence

Methods of Proving Theorems

- Basic methods to prove theorems:
  - *direct proof*
    - $p \to q$ is proved by showing that if p is true then q follows
  - *proof by contrapositive*
    - show the contrapositive $\neg q \to \neg p$
  - *proof by contradiction*
    - show that $(p \wedge \neg q)$ contradicts the assumptions
  - *proof by cases*
    - give proofs for all possible cases
  - *proof of equivalence*
    - $p \leftrightarrow q$ is replaced with $(p \to q) \wedge (q \to p)$

- function?

Set, Function

- function?

  one-to-one (injective) function?

Set, Function

- function?

  one-to-one (injective) function?

  onto (surjective) function?

Set, Function

- function?

  one-to-one (injective) function?

  onto (surjective) function?

  bijective function (one-to-one correspondence)?

Set, Function

- function?

  one-to-one (injective) function?

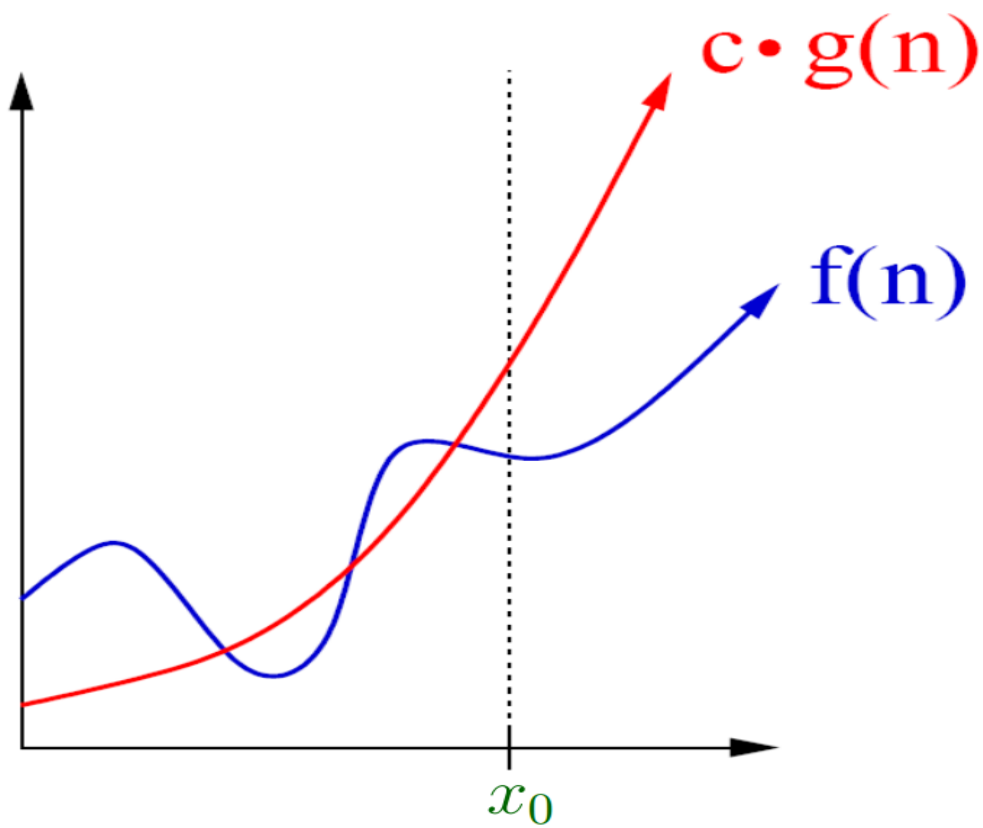
  onto (surjective) function?

  bijective function (one-to-one correspondence)?

- counting the number of such functions?

Big-O Notation

- Let f and g be functions from the set of integers or the set of real numbers to the set of real numbers. We say that $f(n) = O(g(n))$ (reads: $f(n)$ is O of $g(n)$), if there exist some positive constants C and k such that $|f(n)| \leq C|g(n)|$, whenever $n > k$.



Number Theory

- Divisibility

Number Theory

- Divisibility

  Congruence relation

Number Theory

- Divisibility

  Congruence relation

  Primes

Number Theory

- Divisibility

  Congruence relation

  Primes

  GCD and Euclidean Algorithm

Number Theory

- Divisibility

  Congruence relation

  Primes

  GCD and Euclidean Algorithm

  Modular Inverse

- Divisibility

  Congruence relation

  Primes

  GCD and Euclidean Algorithm

  Modular Inverse
  - When does an inverse of a modulo m exist?
  - How to find inverses?

Number Theory

- Divisibility

  Congruence relation

  Primes

  GCD and Euclidean Algorithm

  Modular Inverse

  When does an inverse of a modulo m exist?

  How to find inverses?

  Chinese Remainder Theorem

Number Theory

- Divisibility

  Congruence relation

  Primes

  GCD and Euclidean Algorithm

  Modular Inverse

  When does an inverse of a modulo m exist?

  How to find inverses?

  Chinese Remainder Theorem

  Back substitution

Number Theory

- Divisibility

  Congruence relation

  Primes

  GCD and Euclidean Algorithm

  Modular Inverse

  When does an inverse of a modulo m exist?

  How to find inverses?

  Chinese Remainder Theorem

  Back substitution

$$\begin{aligned} x &\equiv 2 \pmod{3} \\ x &\equiv 3 \pmod{5} \\ x &\equiv 2 \pmod{7} \end{aligned}$$

- Fermat's Little Theorem

- Fermat's Little Theorem

  Euler's Theorem

  Primitive roots, multiplicative order

- Fermat's Little Theorem

  Euler's Theorem

  Primitive roots, multiplicative order

  RSA cryptosystem

  DLP, Diffie-Hellman protocol

Mathematical Induction

- A *typical* proof by mathematical induction, showing that a statement $P(n)$ is true for all integers $n \geq b$ consists of three steps:

Mathematical Induction

- A *typical* proof by mathematical induction, showing that a statement $P(n)$ is true for all integers $n \geq b$ consists of three steps:

  1. We show that $P(b)$ is true. – Base Step

- A *typical* proof by mathematical induction, showing that a statement $P(n)$ is true for all integers $n \geq b$ consists of three steps:

  1. We show that $P(b)$ is true. – Base Step
  2. We then, $\forall n > b$, show either

$$(*) \qquad P(n-1) \rightarrow P(n)$$

or

$$(**) \qquad P(b) \wedge P(b+1) \wedge \cdots \wedge P(n-1) \rightarrow P(n)$$

Mathematical Induction

- A *typical* proof by mathematical induction, showing that a statement $P(n)$ is true for all integers $n \geq b$ consists of three steps:

  1. We show that $P(b)$ is true. – Base Step
  2. We then, $\forall n > b$, show either

  $$(*) \qquad P(n-1) \rightarrow P(n)$$

  or

  $$(**) \qquad P(b) \wedge P(b+1) \wedge \cdots \wedge P(n-1) \rightarrow P(n)$$

  We need to make the inductive hypothesis of either $P(n-1)$ or $P(b) \wedge P(b+1) \wedge \cdots \wedge P(n-1)$.We then use $(*)$ or $(**)$ to derive $P(n)$.

Mathematical Induction

- A *typical* proof by mathematical induction, showing that a statement $P(n)$ is true for all integers $n \geq b$ consists of three steps:

1. We show that $P(b)$ is true. – Base Step
2. We then, $\forall n > b$, show either

   $$(*) \qquad P(n-1) \rightarrow P(n)$$

   or

   $$(**) \qquad P(b) \wedge P(b+1) \wedge \cdots \wedge P(n-1) \rightarrow P(n)$$

   We need to make the inductive hypothesis of either $P(n-1)$ or $P(b) \wedge P(b+1) \wedge \cdots \wedge P(n-1)$.We then use $(*)$ or $(**)$ to derive $P(n)$.
3. We conclude on the basis of the principle of mathematical induction that $P(n)$ is true for all $n \geq b$.

Recurrence

- Iterating a recurrence

Recurrence

- Iterating a recurrence

  bottom up or top down

Recurrence

- Iterating a recurrence

  bottom up or top down

  prove by induction, complexity, ...

Counting

- The sum rule and product rule

Counting

- The sum rule and product rule

  The Inclusion-Exclusion Principle

Counting

- The sum rule and product rule

  The Inclusion-Exclusion Principle

  The Pigeonhole Principle

Counting

- The sum rule and product rule

  The Inclusion-Exclusion Principle

  The Pigeonhole Principle

  Theorem If N is a positive integer and k is an integer with $1 \leq k \leq n$, then there are
  $P(n,k) = n(n-1)(n-2)\cdots(n-k+1)$
  k-element permutations with n distinct elements.

- The sum rule and product rule

  The Inclusion-Exclusion Principle

  The Pigeonhole Principle

  Theorem If N is a positive integer and k is an integer with $1 \leq k \leq n$, then there are
  $P(n, k) = n(n-1)(n-2)\cdots(n-k+1)$
  k-element permutations with n distinct elements.

  $P(n, 3) = 3! \cdot C(n, 3)$

- The sum rule and product rule

  The Inclusion-Exclusion Principle

  The Pigeonhole Principle

  Theorem If N is a positive integer and k is an integer with $1 \leq k \leq n$, then there are
  $P(n,k) = n(n-1)(n-2)\cdots(n-k+1)$
  k-element permutations with n distinct elements.

  $$P(n,3) = 3! \cdot C(n,3)$$

  Pascal's Triangle, Identity

- The sum rule and product rule

  The Inclusion-Exclusion Principle

  The Pigeonhole Principle

  Theorem If N is a positive integer and k is an integer with $1 \leq k \leq n$, then there are
  $P(n,k) = n(n-1)(n-2)\cdots(n-k+1)$
  k-element permutations with n distinct elements.

  $P(n,3) = 3! \cdot C(n,3)$

  Pascal's Triangle, Identity

  The Binomial Theorem, Trinomial

- **Definition** An *r-combination* with repetition allowed, or a *multiset of size r*, chosen from a set of n elements, is an unordered selection of elements with repetition allowed.

  Example Find $\#$ multisets of size 17 from the set $\{1, 2, 3\}$.

  This is equivalent to finding the $\#$ nonnegative solutions to $x_1 + x_2 + x_3 = 17$.

- **Definition** An *r-combination* with repetition allowed, or a *multiset of size r*, chosen from a set of n elements, is an unordered selection of elements with repetition allowed.

  Example Find # multisets of size 17 from the set $\{1, 2, 3\}$.

  This is equivalent to finding the # nonnegative solutions to $x_1 + x_2 + x_3 = 17$.

- Solving linear (non)homogeneous recurrence relation

- **Definition** An *r-combination* with repetition allowed, or a *multiset of size r*, chosen from a set of n elements, is an unordered selection of elements with repetition allowed.

  Example Find # multisets of size 17 from the set $\{1, 2, 3\}$.

  This is equivalent to finding the # nonnegative solutions to $x_1 + x_2 + x_3 = 17$.

- Solving linear (non)homogeneous recurrence relation

- Combinatorial proof

Counting

- **Definition** An *r-combination* with repetition allowed, or a *multiset of size r*, chosen from a set of n elements, is an unordered selection of elements with repetition allowed.

  Example Find # multisets of size 17 from the set $\{1, 2, 3\}$.

  This is equivalent to finding the # nonnegative solutions to $x_1 + x_2 + x_3 = 17$.

- Solving linear (non)homogeneous recurrence relation

- Combinatorial proof

- Generating function

Binary Relations

- Properties of relations

Binary Relations

- Properties of relations

  Representing relations

Binary Relations

- Properties of relations

  Representing relations

  Closures on relations

- Properties of relations

  Representing relations

  Closures on relations

  Equivalence relation

  Definition A relation R on a set A is called an *equivalence relation* if it is reflexive, symmetric, and transitive.

Binary Relations

- Properties of relations

  Representing relations

  Closures on relations

  Equivalence relation

  Definition A relation R on a set A is called an *equivalence relation* if it is reflexive, symmetric, and transitive.

  Partial ordering

- Properties of relations

  Representing relations

  Closures on relations

  Equivalence relation

  Definition A relation R on a set A is called an *equivalence relation* if it is reflexive, symmetric, and transitive.

  Partial ordering

  Definition A relation R on a set A is called a *partial ordering* if it is reflexive, antisymmetric, and transitive.

- Basic concepts

- Basic concepts

  connected graph, simple graph, isomophism, chromatic number, planar graph, Euler circuit, Hamilton circuit, shortest path, bipartite graph, complete graph, special graphs (K_n, $K_{m,n}$, C_n, W_n, Q_n), m-ary tree, tree traversal, spanning tree ...

Good Luck!