Algorithms

Ok-Ran Jeong

Fall, 2016

Department of Software
Gachon University

9. NP Completeness
& Approximation Algorithms

Contents

- NP Completeness
  - Polynomial-Time Algorithms
  - NP-Complete Problems
  - Decision VS. Optimization Problems
  - NP-hard Problem
- Approximation Algorithms
  - Approximation vs. Optimization

Polynomial-Time Algorithms

- Are some problems solvable in polynomial time?
  - Of course: every algorithm we've studied provides polynomial-time solution to some problem
  - We define P to be the class of problems solvable in polynomial time
- Are all problems solvable in polynomial time?
  - No: Turing's "Halting Problem" is not solvable by any computer, no matter how much time is given.
  - Such problems are clearly intractable, not in P

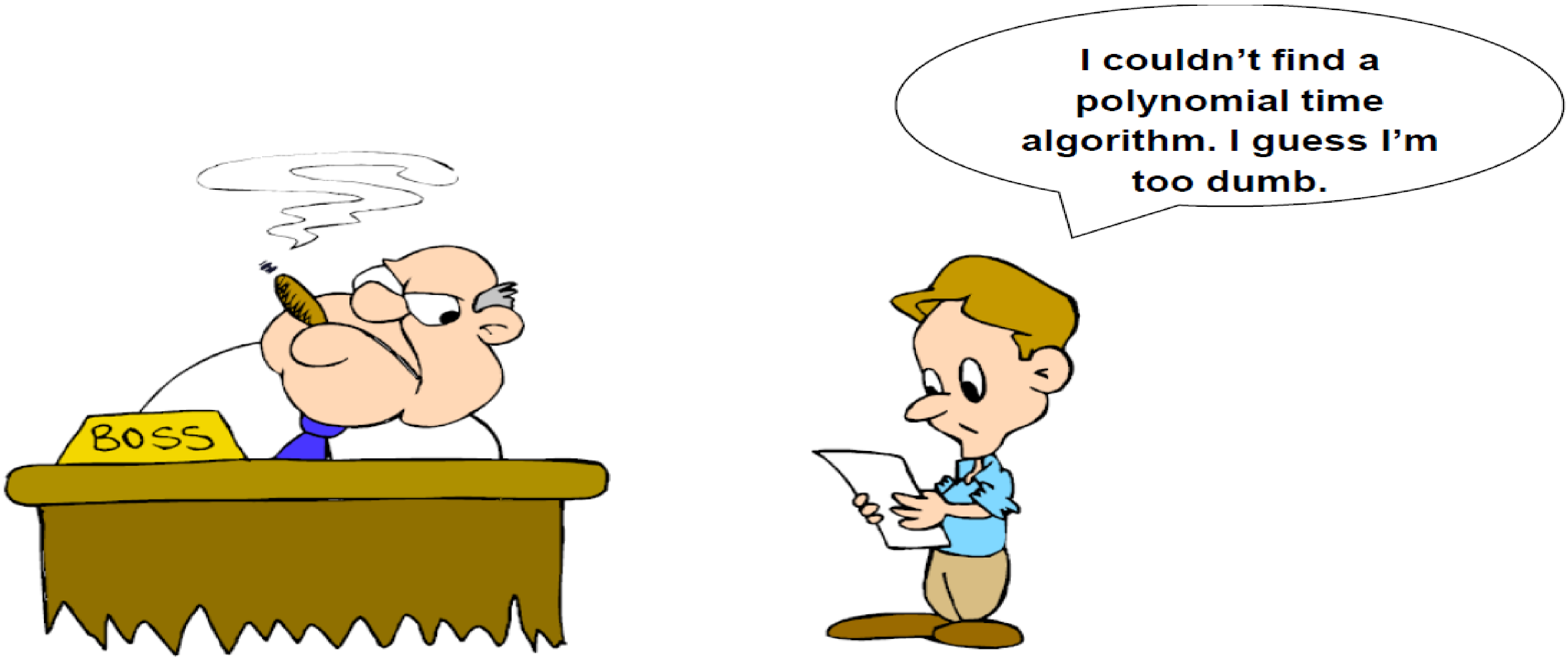
Halting Problem

- An important question of computing science is “Are there problems that cannot be solved?”
- There are, and probably the most famous of these is the halting problem described by Turing.
- Alan Turing was thinking in terms of Turing machines (there were no computers), but it is easy to extend the idea.

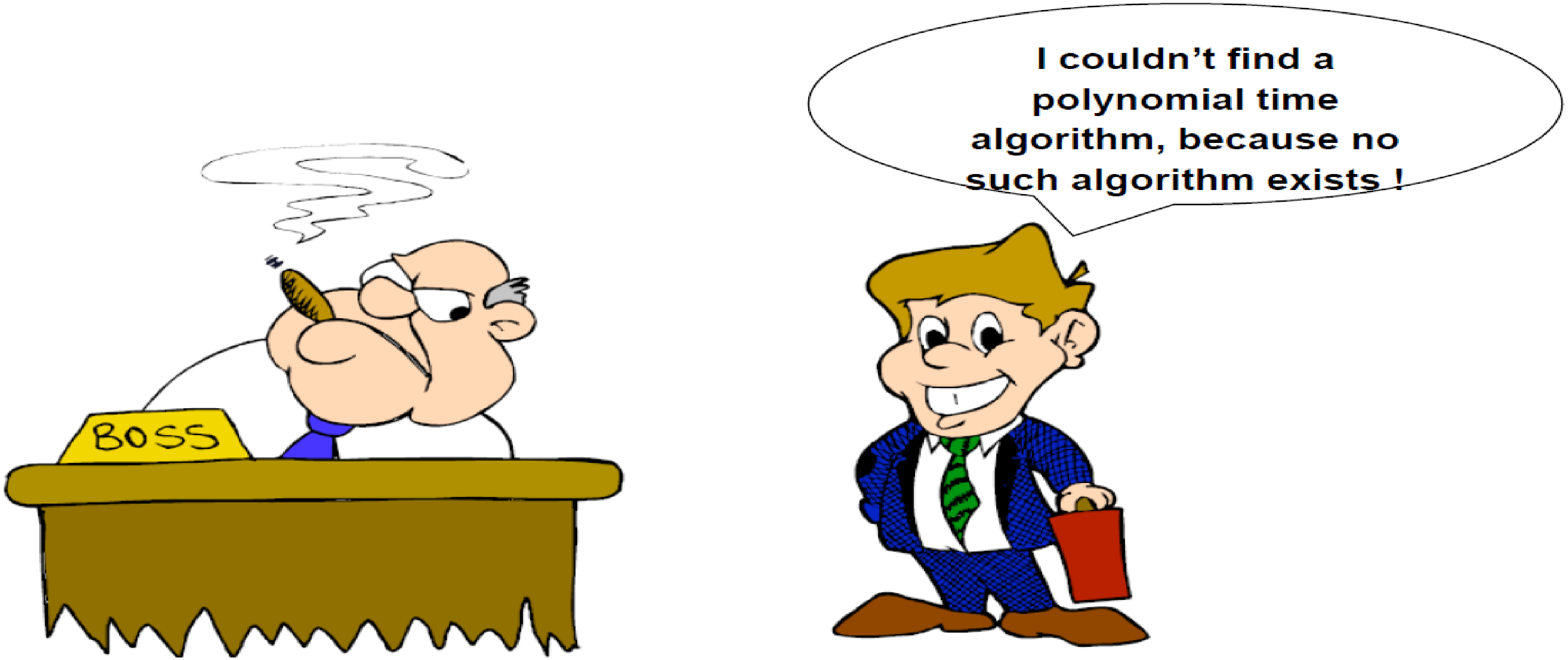
Halting Problem

- Halting problem
  - Can we write a program that will look at any computer program and its input and decide if the program will halt (not run infinitely)?
  - A practical solution might be to run the program and if it halts you have your answer. If after a given amount of time it doesn't halt, guess that it won't halt. However, you wouldn't know if the program would eventually halt.

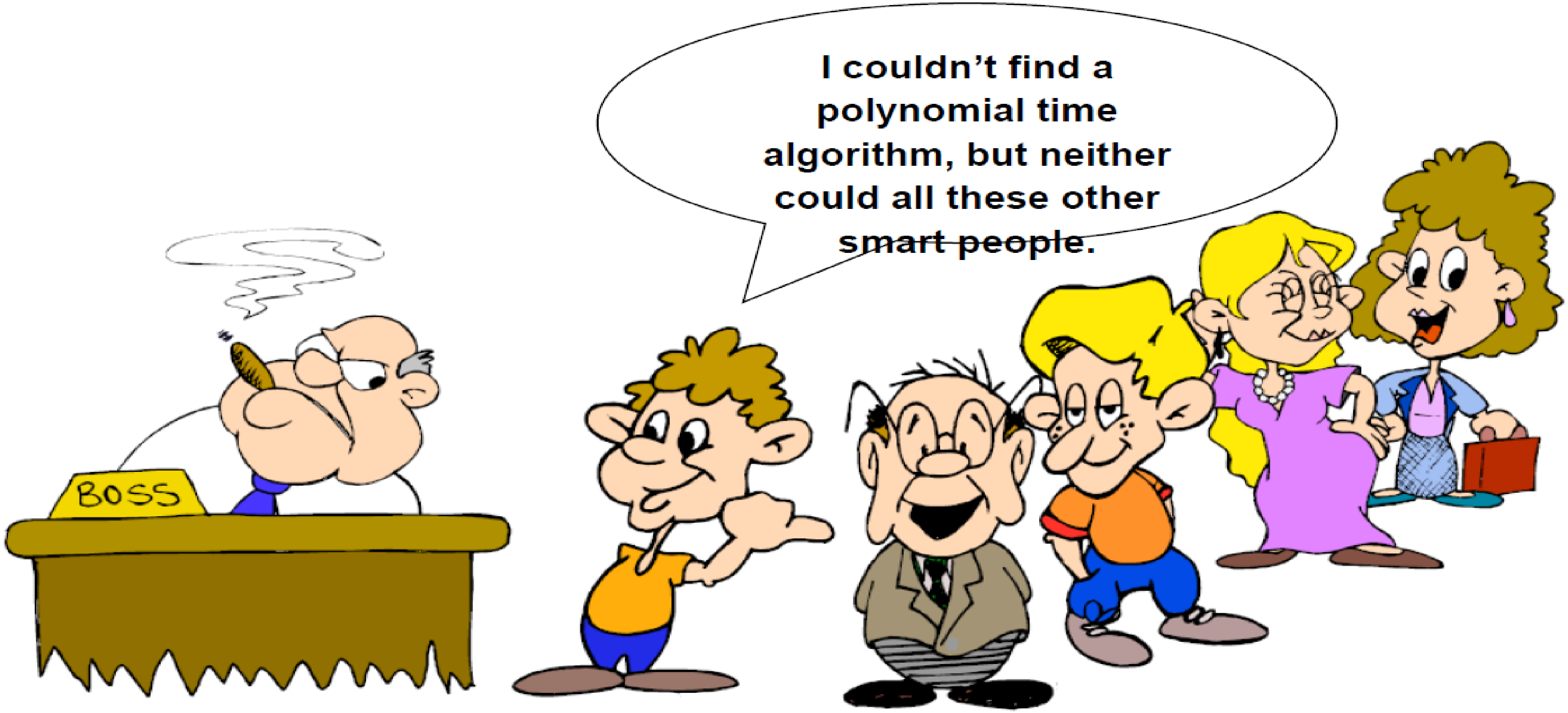
NP-Complete


BOSS
I couldn't find a polynomial time algorithm. I guess I'm too dumb.

NP-Complete


BOSS
I couldn't find a polynomial time algorithm, because no such algorithm exists !

NP-Complete


BOSS
I couldn't find a polynomial time algorithm, but neither could all these other smart people.

NP-Completeness

- Some problems are intractable
  - They grow large, we are unable to solve them in reasonable time
- What constitutes reasonable time?

  Standard working definition: polynomial time
  - On an input of size n the worst-case running time is $O(n^k)$ for some constant K
  - Polynomial time: $O(n^2)$, $O(n^3)$, $O(1)$, $O(n \lg n)$
  - Not in polynomial time: $O(2^n)$, $O(n^n)$, $O(n!)$

NP-Complete Problems

- Definition
  - The class P (or NP) denotes the family of all problems that can be solved by deterministic (nondeterministic) polynomial time algorithms
    - deterministic(unique result) vs. nondeterminimistic (choice, failure, success)
  - All Problems are decision problems only answers YES or No
- Examples of an NP-Complete problem:
  - Longest path problem
  - The Hamiltonian path problem
  - Traveling salesman problem

NP-Complete Problems

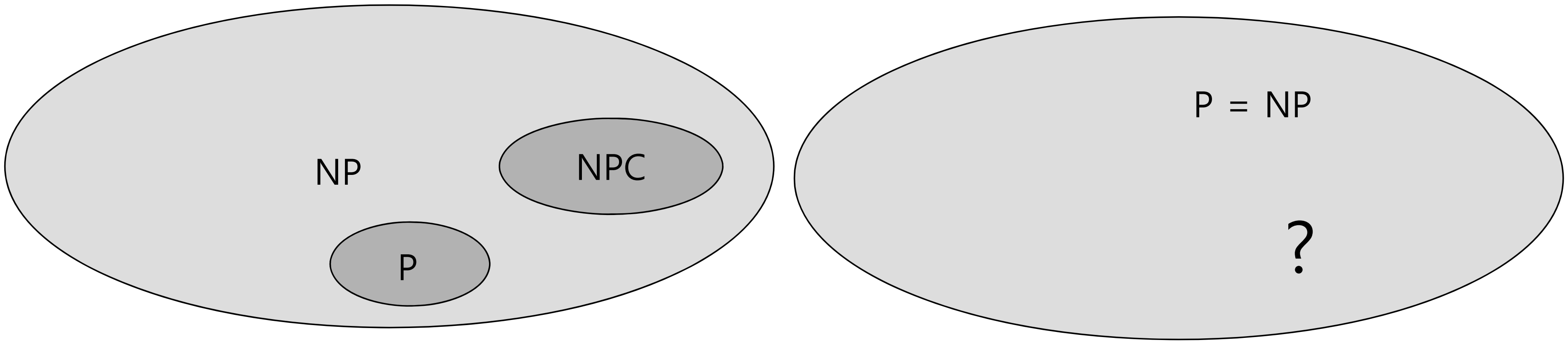
- The NP-Complete problems are an interesting class of problems whose status in unknown
  - No polynomial-time algorithm has been discovered for an NP-Complete problem
- We call this the P = NP question
  - The biggest open problem in CS

NP-Completeness

- Poly time algorithm: input size n (in some encoding), worst case running time – $O(n^c)$ for some constant c.
- Three classes of problems
  - P: problems solvable in poly time.
  - NP: problems verifiable in poly time.
  - NPC: problems in NP and as hard as any problem in NP.

Unsolvable/ Undecidable	Solvable/ Decidable
– halting problem - Hilbert 10th problem	-P -NP -NPC

View of Theoretical Computer Scientists on P, NP, NPC



$P \subset NP$, $NPC \subset NP$, $P \cap NPC = \varnothing$

NP-Completeness (verifiable)

- Verifiable in poly time: given a certificate of a solution, could verify the certificate is correct in poly time.
- Examples (their definitions come later):
  - Hamiltonian-cycle, given a certificate of a sequence $(v_1, v_2, \ldots, v_n)$, easily verified in poly time.
  - (so try each instance, and verify it, but 2^n instances)
- Why not defined as “solvable in exponential time?” or “Non Poly time”?

Decision VS. Optimization Problems

- Decision problem: solving the problem by giving an answer "YES" or "NO"
- Optimization problem: solving the problem by finding the optimal solution.
- Examples:
  - SHORTEST-PATH (optimization)
    - Given G, u,v, find a path from u to v with fewest edges.
  - PATH (decision)
    - Given G, u,v, and k, whether exist a path from u to v consisting of at most k edges.

Decision VS. Optimization Problems (Cont.)

- Decision is easier (i.e., no harder) than optimization
- If there is an algorithm for an optimization problem, the algorithm can be used to solve the corresponding decision problem.
  - Example: SHORTEST-PATH for PATH
- If optimization is easy, its corresponding decision is also easy. Or in another way, if provide evidence that decision problem is hard, then the corresponding optimization problem is also hard.
- NPC is confined to decision problem. (also applicable to optimization problem.)
  - Another reason is that: easy to define reduction between decision problems.

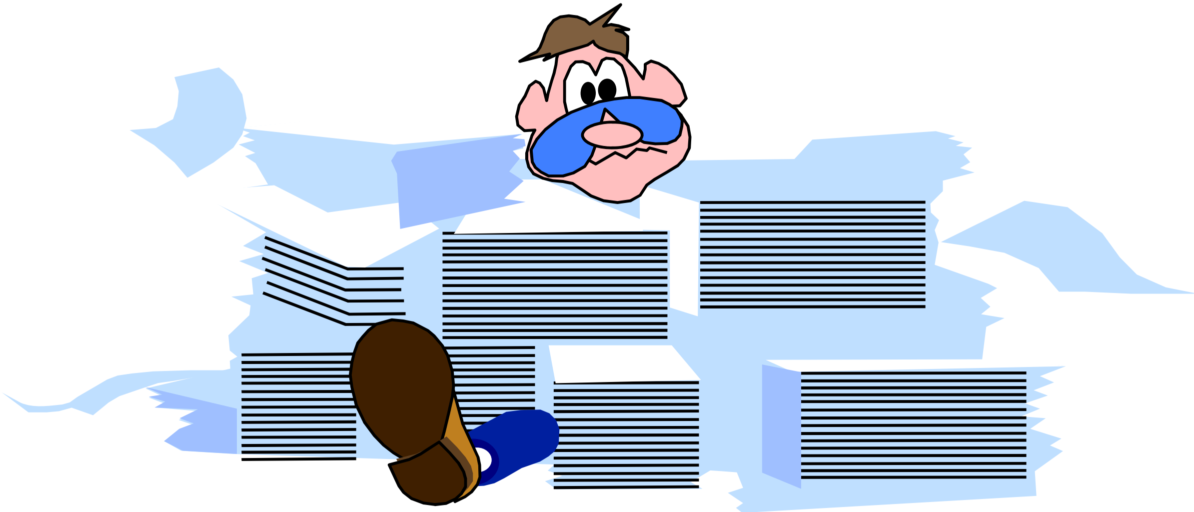
NP-hard Problem

- **NP-hard** (non-deterministic polynomial-time hard), in computational complexity theory, is a class of problems that are, informally, "at least as hard as the hardest problems in NP".

Approximation Algorithms

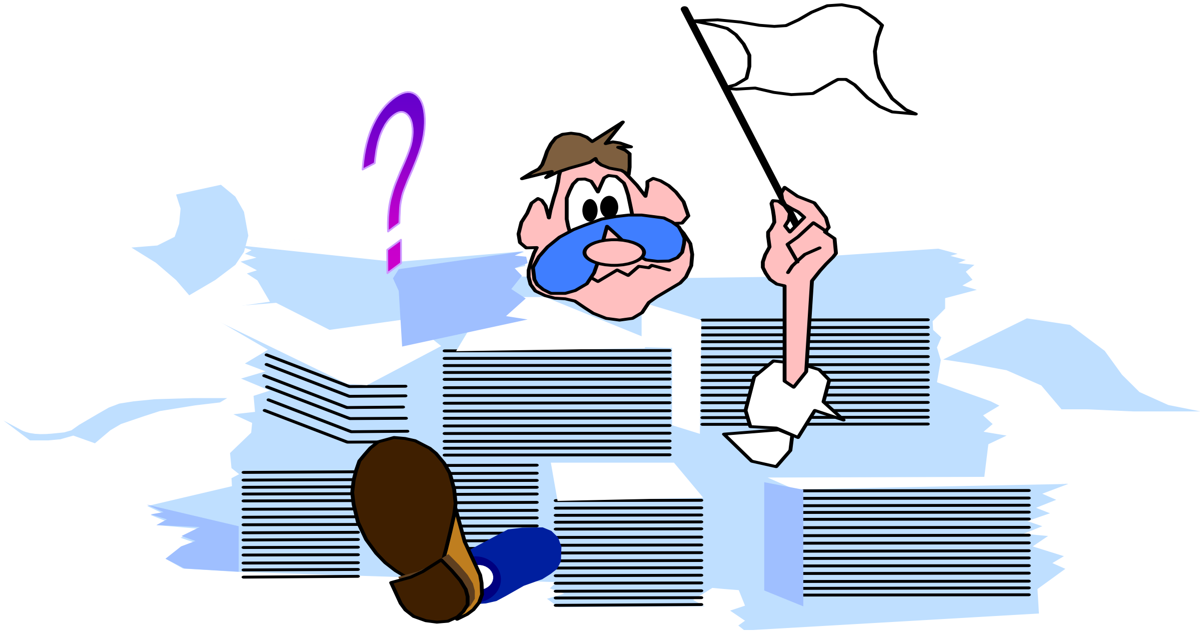
Motivation

- By now we've seen many NP-Complete problems.
- We conjecture none of them has polynomial time algorithm.



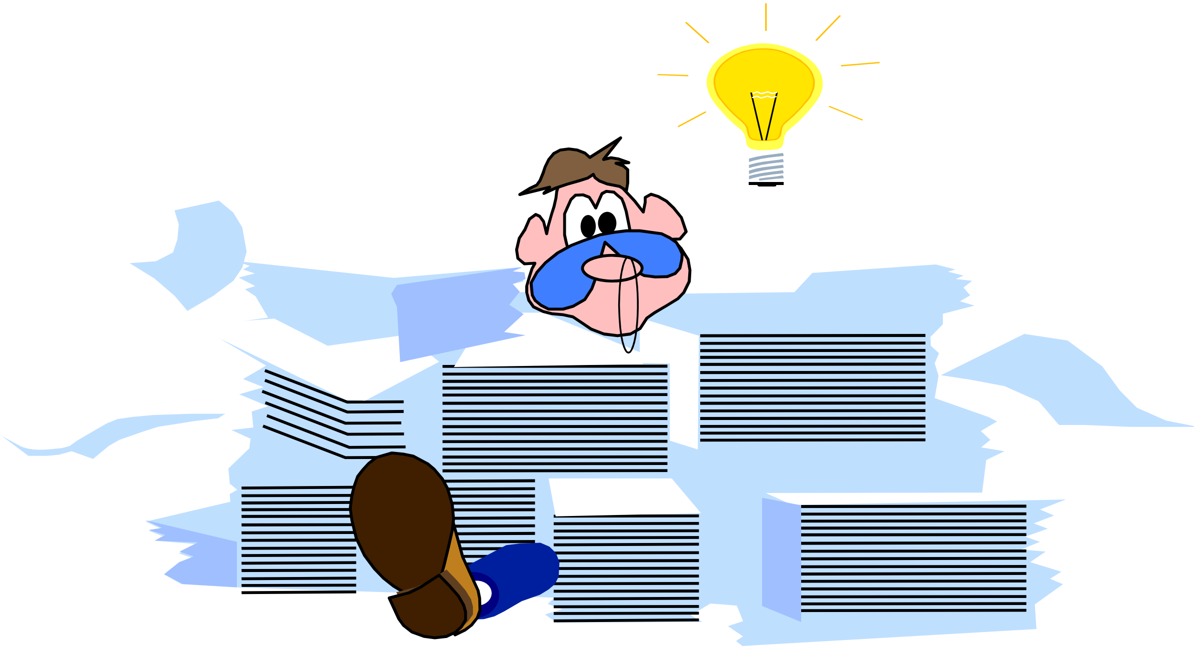
Motivation

- Is this a dead-end? Should we give up altogether?



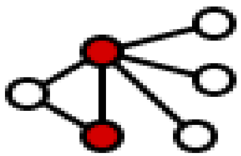
Motivation

- Or maybe we can settle for good approximation algorithms?



Introduction

- Objectives:
  - To formalize the notion of approximation.
  - To demonstrate several such algorithms.
- Overview:
  - Optimization and Approximation
  - VERTEX-COVER



size of Min vertex cover: 2

Optimization

- Many of the problems we've encountered so far are really *optimization problems*.
- The task can be naturally rephrased as finding a maximal/minimal solution.

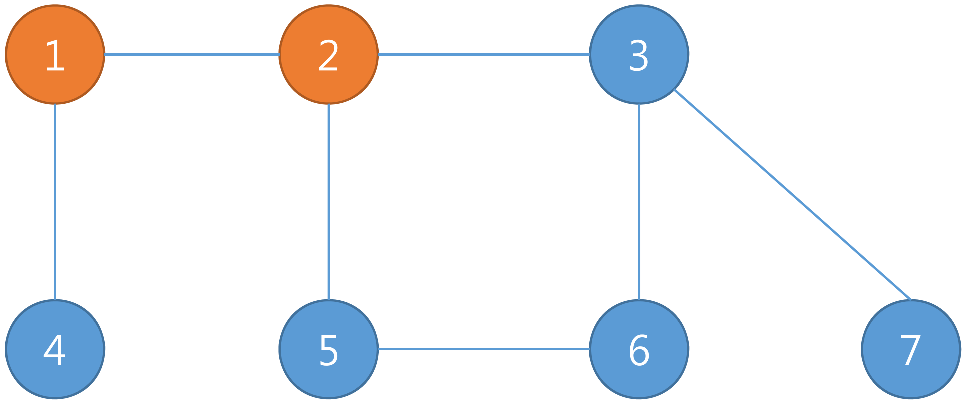
Approximation

- An algorithm which returns an answer C which is "close" to the optimal solution C* is called an *approximation algorithm*.
- "Closeness" is usually measured by the ratio bound $\rho(n)$ the algorithm produces.
- Which is a function that satisfies, for any input size n, $\max\{C/C^*, C^*/C\} \leq \rho(n)$.

Vertex-cover problem

- What is a vertex-cover?
- Given a undirected graph G=(V, E), **vertex-cover** V':
  - $V' \subseteq V$
  - for each edge (u, v) in E, either $u \in V'$ or $v \in V'$ (or both)
- The size of a vertex-cover is |V'|

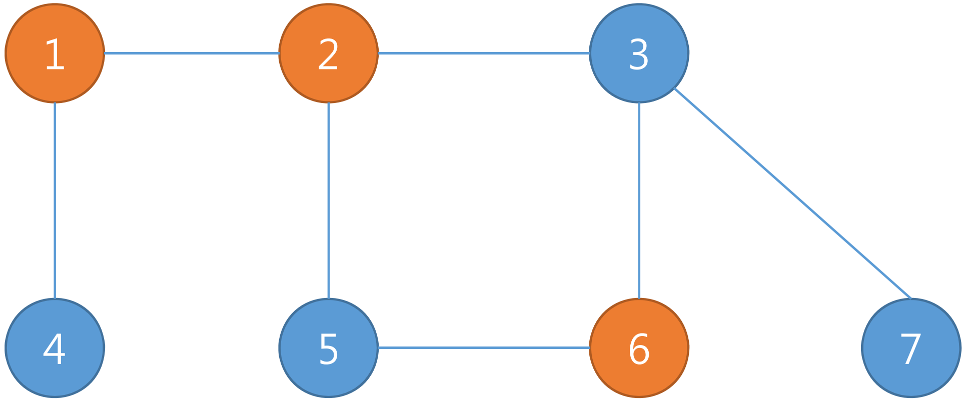
Vertex-cover problem



Are the red vertices a vertex-cover
?
No. why?

Edges (5, 6), (3, 6) and (3, 7) are not covered by it

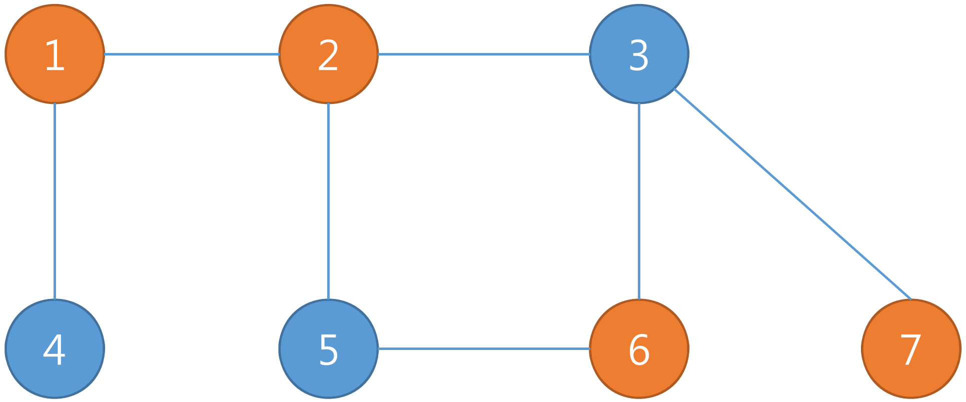
Vertex-cover problem



Are the red vertices a vertex-cover
?
No. why?

Edge (3, 7) is not covered by it

Vertex-cover problem

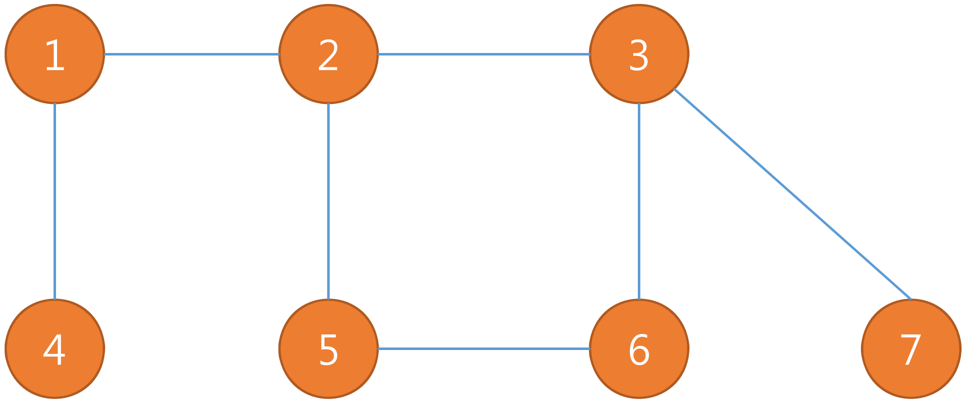


Are the red vertices a vertex-cover
?
Yes

What is the size?

4

Vertex-cover problem

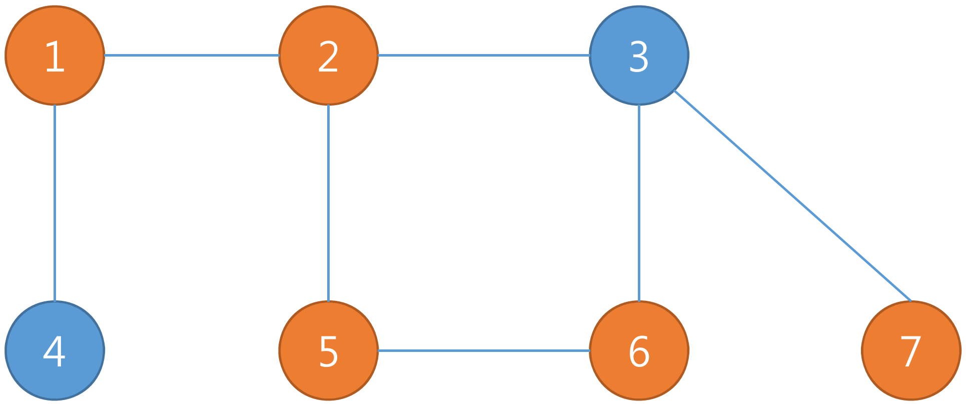


Are the red vertices a vertex-cover
?
Yes

What is the size?

7

Vertex-cover problem and a 2-approximation algorithm



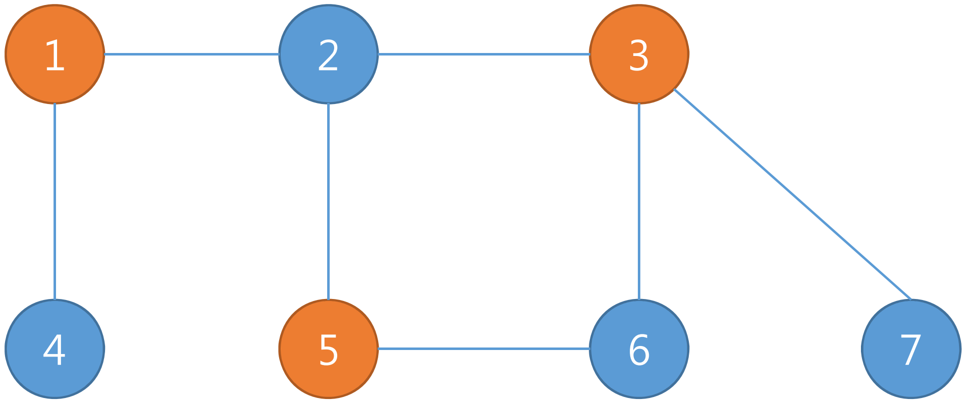
Are the red vertices a vertex-cover?

Yes

What is the size?

5

Vertex-cover problem



Are the red vertices a vertex-cover
?
Yes

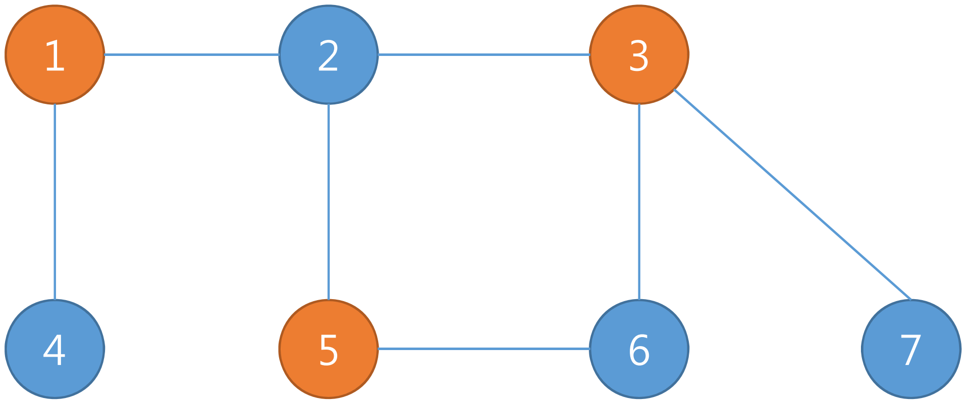
What is the size?

3

Vertex-cover problem and a 2-approximation algorithm

- **Vertex-cover problem**
  - Given a undirected graph, find a vertex cover with minimum size.

Vertex-cover problem and a 2-approximation algorithm



A minimum vertex-cover

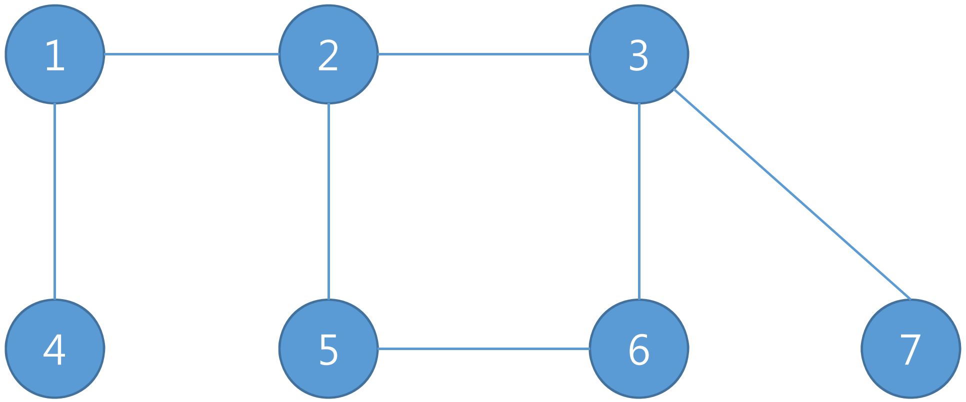
Vertex-cover problem and a 2-approximation algorithm

- Vertex-cover problem is **NP-complete**
- A 2-approximation polynomial time algorithm is as the following:
- **APPROX-VERTEX-COVER**(G)

```
C = ∅;
E'=G.E;
while(E' ≠ ∅ ){
    Randomly choose a edge (u,v) in E', put u and v into C;
    Remove all the edges that covered by u or v from E'
}
Return C;
```

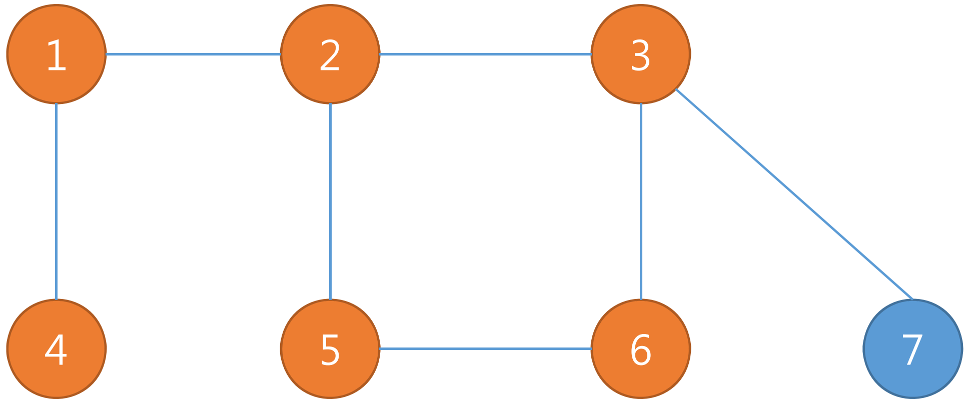
Vertex-cover problem and a 2-approximation algorithm

```
APPROX-VERTEX-COVER(G)
    C = ∅;
    E′=G.E;
    while(E′ ≠ ∅ ){
        Randomly choose a e
        dge (u,v) in E′, put u a
        nd v into C;
        Remove all the edges
        that covered by u or v
        from E′
    }
    Return C;
```



Vertex-cover problem and a 2-approximation algorithm

APPROX-VERTEX-COVER(
G)
  C = ∅;
  E′=G.E;
  while(E′ ≠ ∅){
    Randomly choose a edge (u,v) in E′, put u and v into C;
    Remove all the edges that covered by u or v from E′
  }
  Return C;



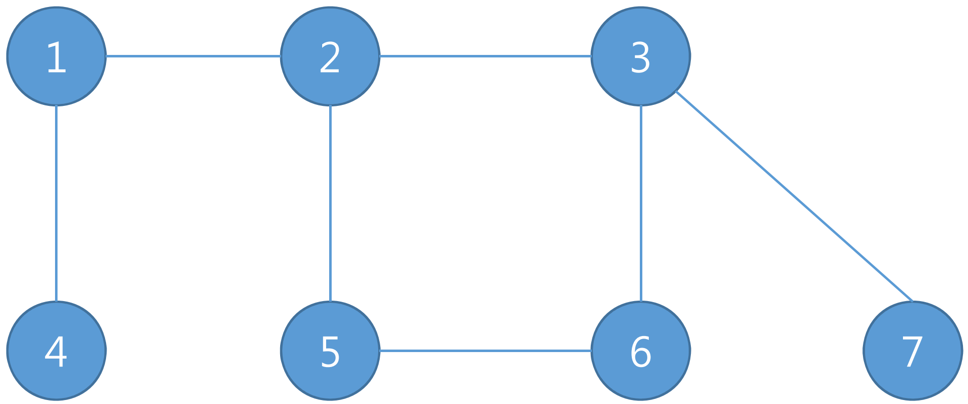
It is then a vertex cover

Size? 6

How far from optimal one? Max(6/3, 3/6) = 2

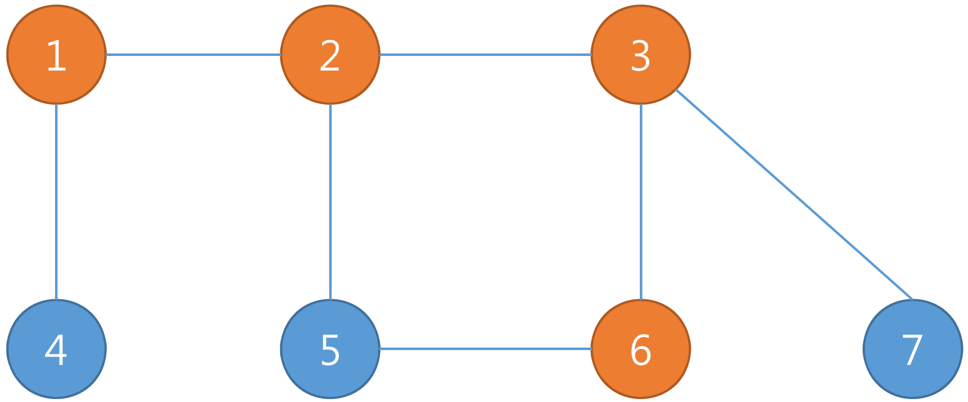
Vertex-cover problem and a 2-approximation algorithm

```
APPROX-VERTEX-COVER(G)
    C = ∅;
    E′=G.E;
    while(E′ ≠ ∅ ){
        Randomly choose a edge (u,v) in E′,
        put u and v into C;
        Remove all the edges that covered by u or v from E′
    }
    Return C;
```



Vertex-cover problem and a 2-approximation algorithm

```
APPROX-VERTEX-COVER(G)
    C = ∅;
    E′=G.E;
    while(E′ ≠ ∅ ){
        Randomly choose a edge (u,v) in E′,
        put u and v into C;
        Remove all the edges that covered by u or v from E′
    }
    Return C;
```



It is then a vertex cover

Size? 4

How far from optimal one? Max(4/3, 3/4) = 1.33

Vertex-cover problem and a 2-approximation algorithm

- **APPROX-VERTEX-COVER**(G) is a 2-approximation algorithm
- When the size of minimum vertex-cover is s
- The vertex-cover produced by **APPROX-VERTEX-COVER** is at most $2s$

Vertex-cover problem and a 2-approximation algorithm

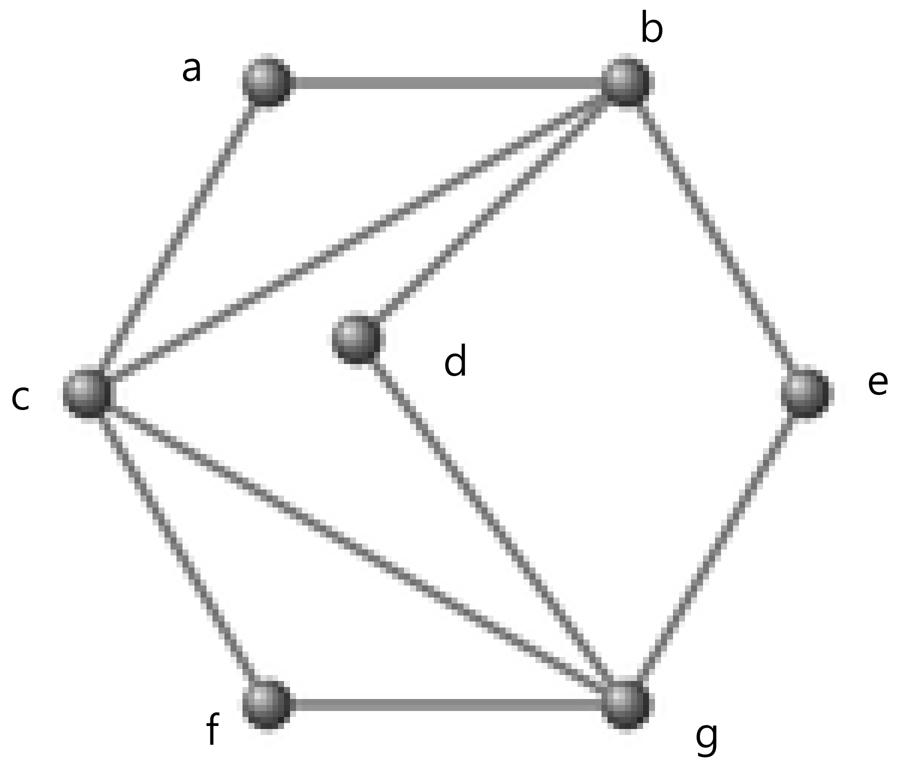
Proof:

- Assume a minimum vertex-cover is U*
- A vertex-cover produced by **APPROX-VERTEX-COVER**(G) is U
- The edges chosen in **APPROX-VERTEX-COVER**(G) is A
- A vertex in U* can only cover 1 edge in A
  - So |U*|>= |A|
- For each edge in A, there are 2 vertices in U
  - So |U|=2|A|
- So |U*|>= |U|/2
- So $\frac{|U|}{|U^*|} \leq 2$

Approximation algorithms for NPC problems

- What if $\rho(n)$=1?
- It is an algorithm that can always find a optimal solution

What is vertex cover?



THANK YOU