Design and Analysis of Algorithms
NP-Completeness

Some Slides from: Haidong Xue

Provided by: Dr. Atif Tahir, Waheed Ahmed

Presented by: Farrukh Salim Shaikh

Week 13-14:

P, NP, NP-Completeness, Approximation Algorithms and Reduction

What is polynomial-time?

- Polynomial-time: running time is $O(n^k)$, where k is a constant.
- Are they polynomial-time running time?
  - $T(n) = 3$
    - yes
  - $T(n) = n$
    - yes
  - $T(n) = nlg(n)$
    - yes
  - $T(n) = n^3$
    - yes

What is polynomial-time?

- Are the polynomial-time?
  - $T(n) = 5^n$
    - No
  - $T(n) = n!$
    - No
- Problems with polynomial-time algorithms are considered as tractable
- With polynomial-time, we can define **P** problems, and **NP** problems

What are P and NP?

- P problems
  - (The original definition) Problems that can be solved by **deterministic Turing machine** in polynomial-time.
  - (A equivalent definition) Problems that are solvable in polynomial time.
- NP problems
  - (The original definition) Problems that can be solved by **non-deterministic Turing machine** in polynomial-time.
  - (A equivalent definition) Problems that are **verifiable** in polynomial time.
    - Given a solution, there is a polynomial-time algorithm to tell if this solution is correct.

What are P and NP?

- Based on the definition of P and NP, which statements are correct?
  - "NP means non-polynomial"
    - No!
  - $\mathrm{P} \cap \mathrm{NP} = \emptyset$
    - No. $\mathrm{P} \subseteq \mathrm{NP}$
  - A P problem is also a NP problem
    - Yes. $\mathrm{P} \subseteq \mathrm{NP}$

What are P and NP?

- any problem solvable by a deterministic Turing machine in polynomial time is also solvable by a nondeterministic Turing machine in polynomial time. Thus, $\mathbf{P} \subseteq \mathbf{NP}$

P = NP means whether an NP problem can belong to class P problem. In other words whether every problem whose solution can be verified by a computer in polynomial time can also be solved by a computer in polynomial time

What are P and NP?

In order to prove that **P** ≠ **NP**, we would need to prove that there exists a set of problems *X* such that:

- *X* falls in **NP**. There exists an algorithm with which a nondeterministic Turing machine could solve problems in *X* in polynomial time
- *X* does *not* fall in **P**. There exists *no algorithm whatsoever* with which a deterministic Turing machine could solve problems in *X* in polynomial time

N

Tractable/Intractable Problems

- Problems in P are also called **tractable**
- Problems **not** in P are **intractable or unsolvable**
  - Can be solved in reasonable time only for small inputs
  - Or, can not be solved at all
- Are non-polynomial algorithms always worst than polynomial algorithms?

  - $n^{1,000,000}$ is *technically* tractable, but really impossible

  - $n^{\log \log \log n}$ is *technically* intractable, but easy

Decision Problem = Yes or No

Example of Unsolvable Problem

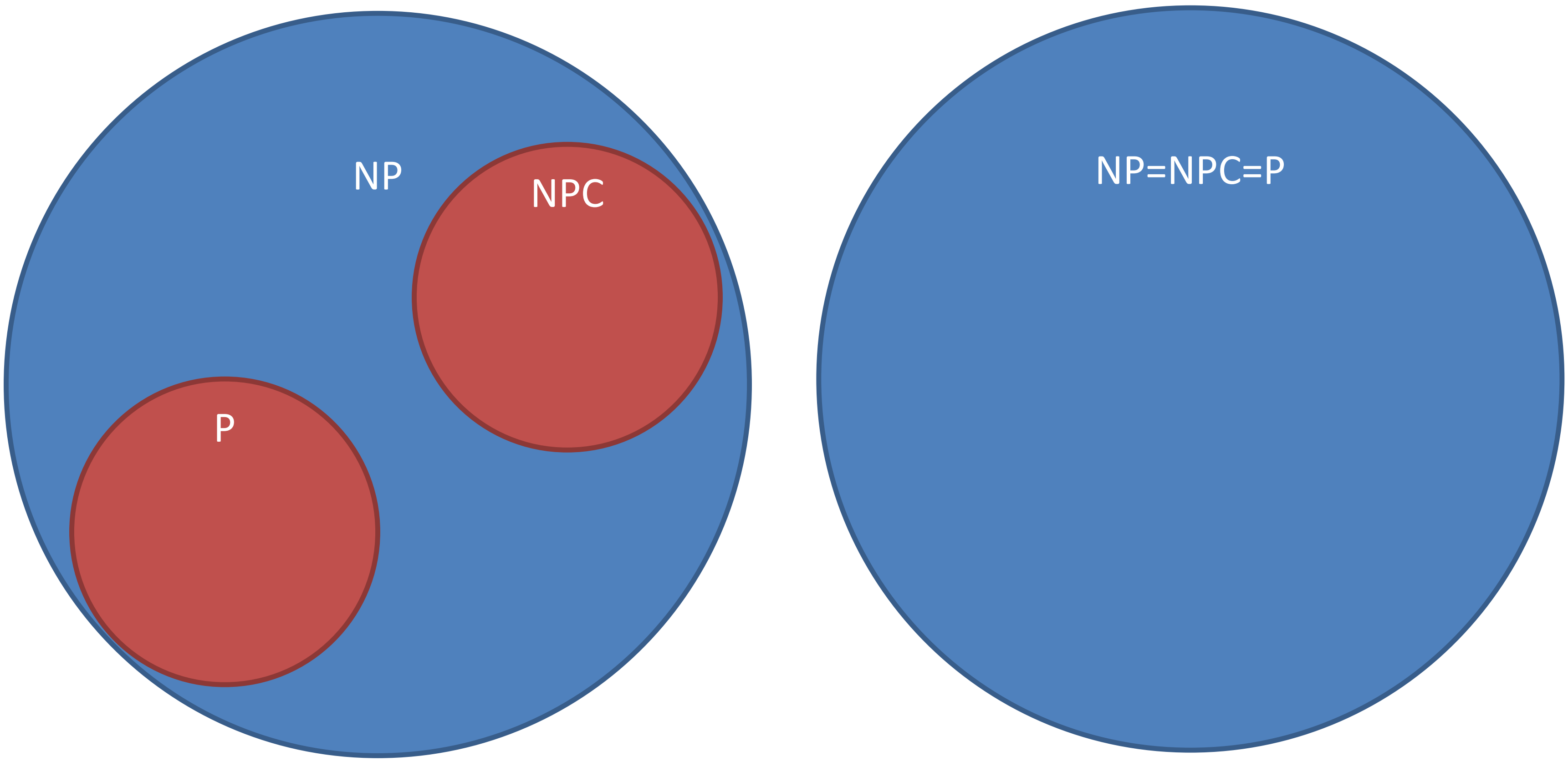
- Turing discovered in the 1930's that there are problems **unsolvable** by *any* algorithm.
- The most famous of them is the ***halting problem***
  - Given an arbitrary algorithm and its input, will that algorithm eventually halt, or will it continue forever in an "*infinite loop*?"

What are NP-complete problems?

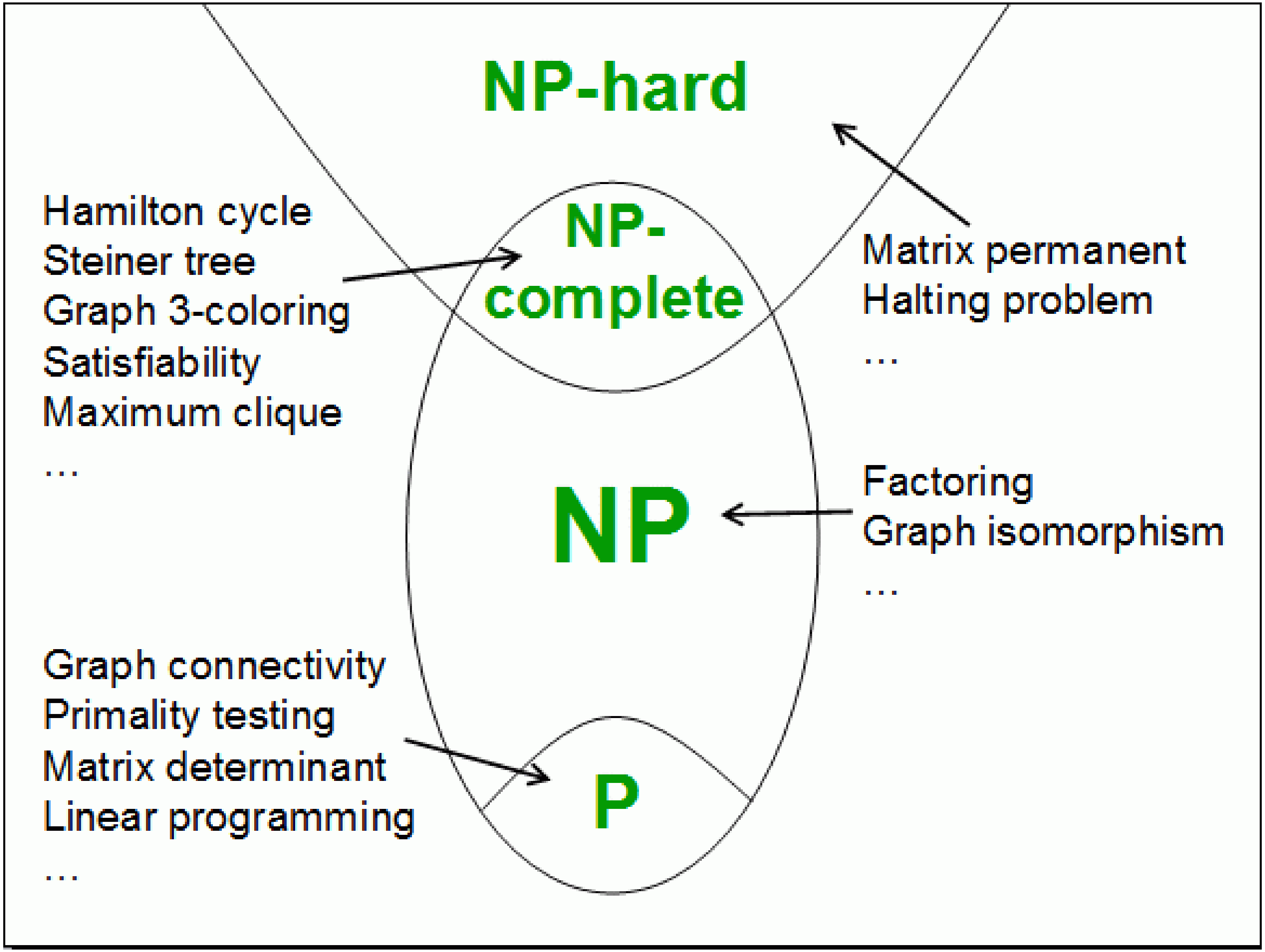
- A NP-complete problem(NPC) is
  - a NP problem
  - harder than all equal to all NP problems
- In other words, NPC problems are the hardest NP problems
- **So far**, no polynomial time algorithms are found for any of NPC problems

What are NP-complete problems?

Most scientists believe: However, it is not ruled out that:



NP-Hard problems: problems harder than or equal to NPC problems


NP-hard
NP-
complete
NP
P
Hamilton cycle
Steiner tree
Graph 3-coloring
Satisfiability
Maximum clique
...
Matrix permanent
Halting problem
...
Factoring
Graph isomorphism
...
Graph connectivity
Primality testing
Matrix determinant
Linear programming
...

Why we study NPC?

- One of the most important reasons is:
  - If you see a problem is NPC, you can stop from spending time and energy to develop a fast polynomial-time algorithm to solve it.
- Just tell your boss it is a NPC problem
- How to prove a problem is a NPC problem?

How to prove a problem is a NPC problem?

- A common method is to prove that it is not easier than a known NPC problem.
- To prove problem A is a NPC problem
  - Choose a NPC problem B
  - Develop a **polynomial-time algorithm translate A to B**
- A **reduction algorithm**
- If A can be solved in polynomial time, then B can be solved in polynomial time. It is contradicted with B is a NPC problem.
- So, A cannot be solved in polynomial time, it is also a NPC problem.

NP Hard & NP Completeness

- A problem X is NP-complete if X $\in$NP and X is NP-hard.
- A problem X is NP-hard if every problem Y $\in$NP reduces to X.

A language $L \subseteq \{0, 1\}^*$ is ***NP-complete*** if

1. $L \in$ NP, and
2. $L' \leq_P L$ for every $L' \in$ NP.

If a language L satisfies property 2, but not necessarily property 1, we say that L is ***NP-hard***. We also define NPC to be the class of NP-complete languages.

What if a NPC problem needs to be solved?

- Buy a more expensive machine and wait
  - (could be 1000 years)
- Turn to approximation algorithms
  - Algorithms that produce near optimal solutions

Approximation algorithms for NP-complete problems

Approximation algorithms for NPC problems

- If a problem is NP-complete, there is very likely no polynomial-time algorithm to find an optimal solution
- The idea of approximation algorithms is to develop polynomial-time algorithms to find a near optimal solution

Approximation algorithms for NPC problems

- E.g.: develop a greedy algorithm without proving the greedy choice property and optimal substructure.
- Are those solution found near-optimal?
- How near are they?

Approximation algorithms for NPC problems

- **Approximation ratio** $\boldsymbol{\rho(n)}$
  - Define the cost of the optimal solution as C*
  - The cost of the solution produced by a approximation algorithm is C
  - $\boldsymbol{\rho(n)} \geq max(\frac{C}{C^*}, \frac{C^*}{C})$
- The approximation algorithm is then called a $\boldsymbol{\rho(n)}$-approximation algorithm.

Approximation algorithms for NPC problems

- E.g.:
  - If the total weigh of a MST of graph G is 20
  - A algorithm can produce some spanning trees, and they are not MSTs, but their total weights are always smaller than 25
  - What is the approximation ratio?
    - 25/20 = 1.25
  - This algorithm is called?
    - A 1.25-approximation algorithm

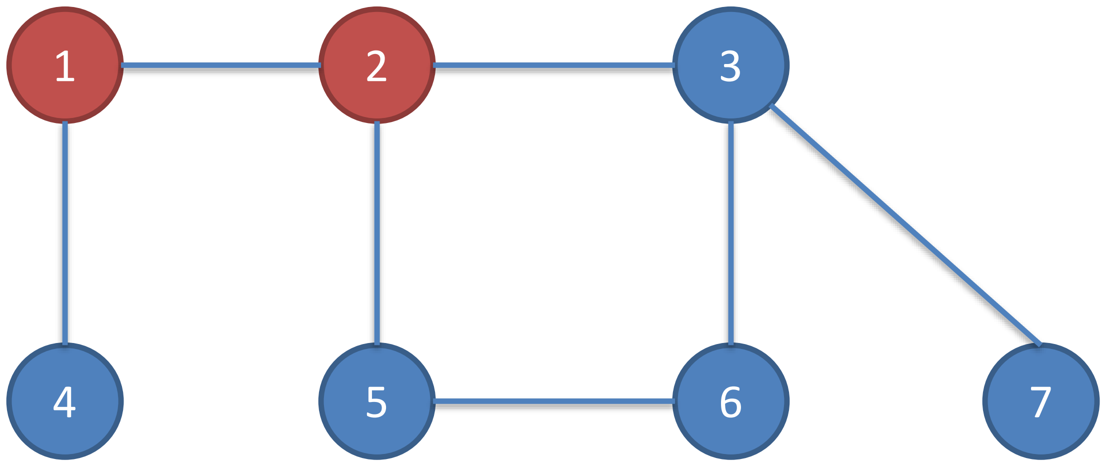
Approximation algorithms for NPC problems

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

Vertex-cover problem and a 2-approximation algorithm

- 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 and a 2-approximation algorithm

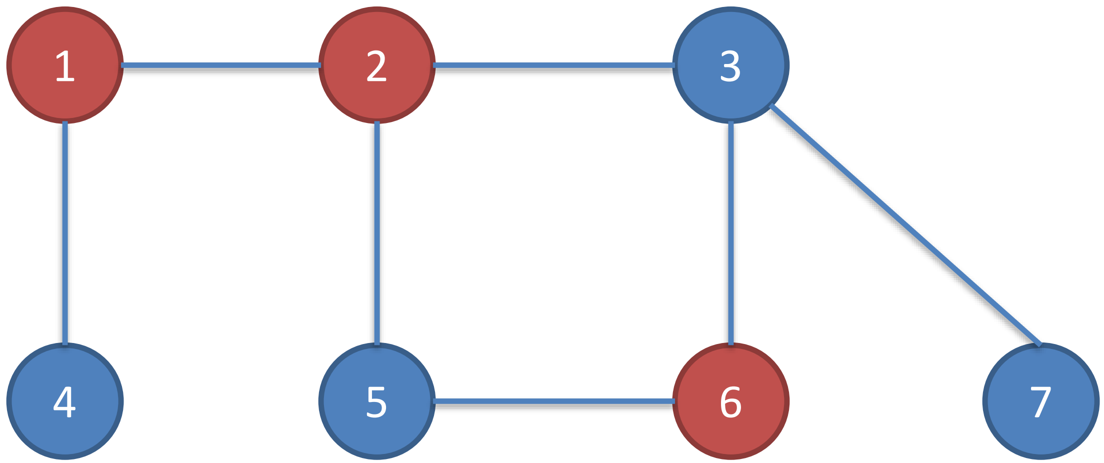


Are the red vertices a vertex-cover?

No. why?

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

Vertex-cover problem and a 2-approximation algorithm

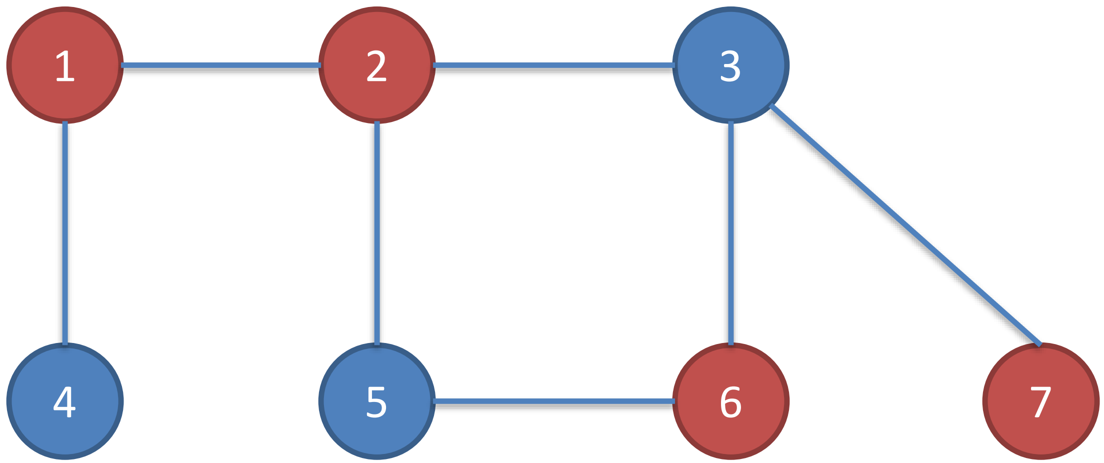


Are the red vertices a vertex-cover?

No. why?

Edge (3, 7) is not covered by it

Vertex-cover problem and a 2-approximation algorithm



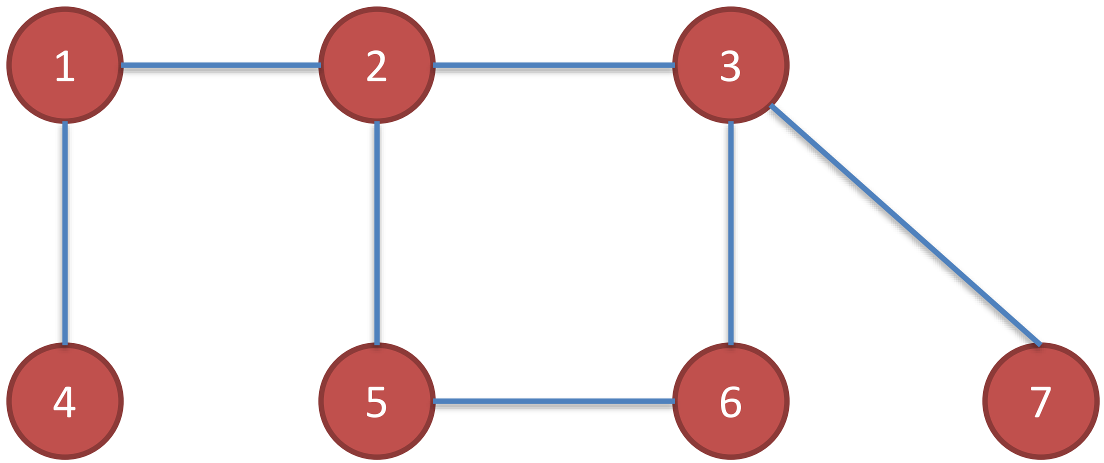
Are the red vertices a vertex-cover?

Yes

What is the size?

4

Vertex-cover problem and a 2-approximation algorithm



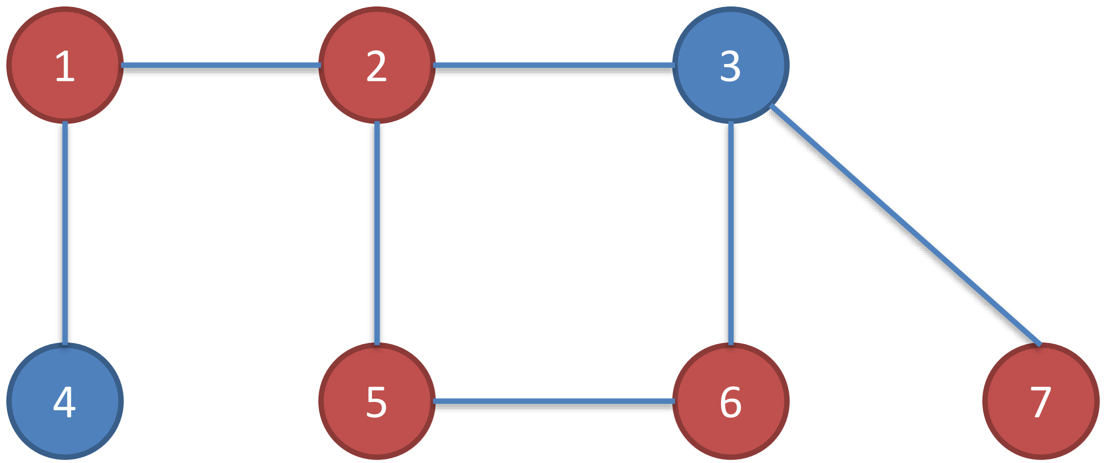
Are the red vertices a vertex-cover?

Yes

What is the size?

7

Vertex-cover problem and a 2-approximation algorithm



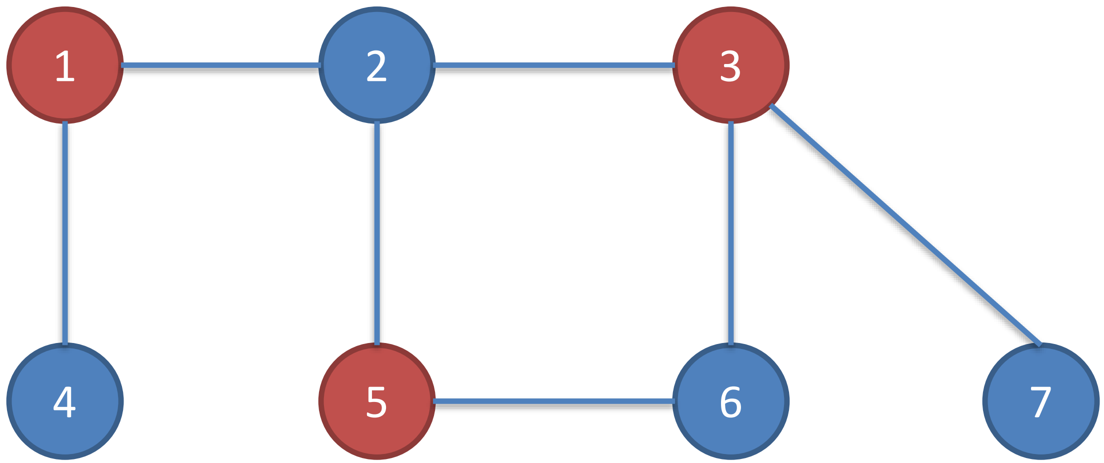
Are the red vertices a vertex-cover?

Yes

What is the size?

5

Vertex-cover problem and a 2-approximation algorithm



Are the red vertices a vertex-cover?

Yes

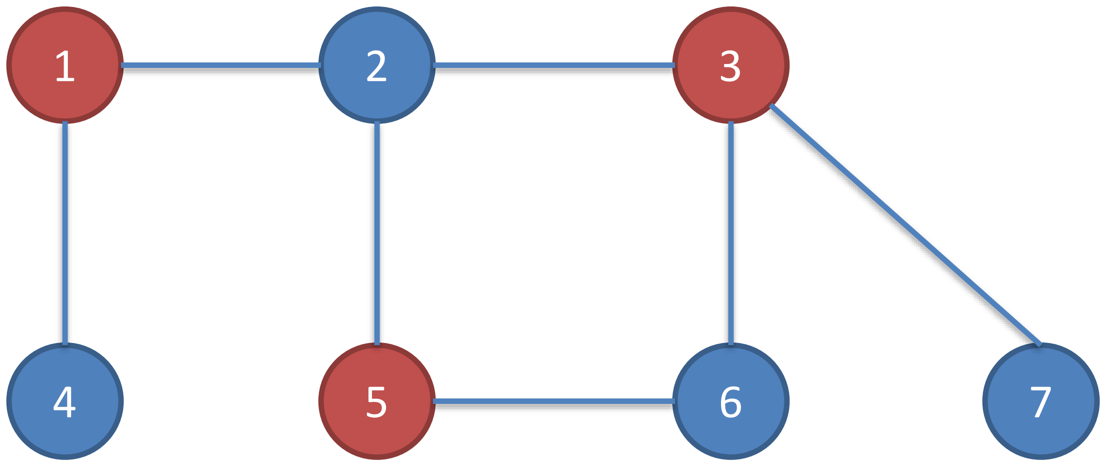
What is the size?

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

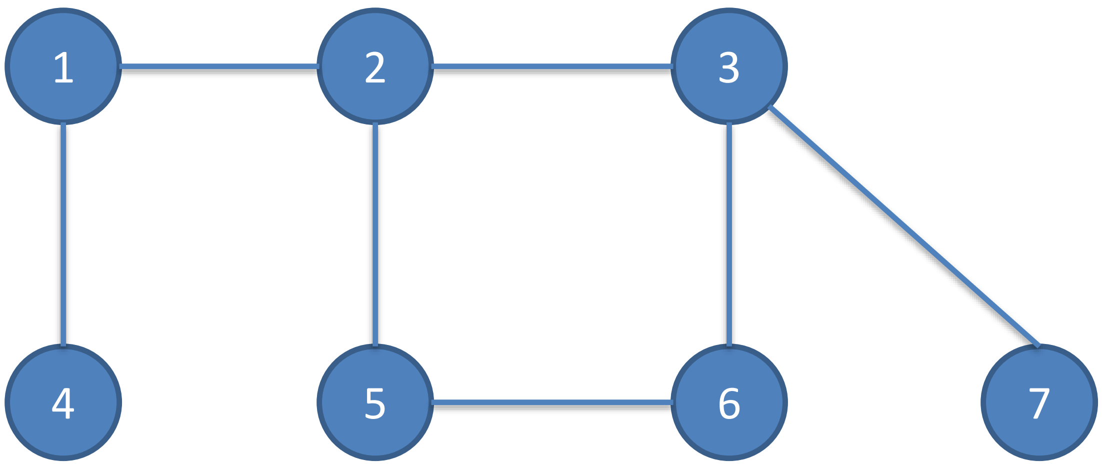
- Vertex-cover problem is **NP-complete**
- A 2-approximation polynomial time algorithm is as the following:
- **APPROX-VERTEX-COVER**(G)

```
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)

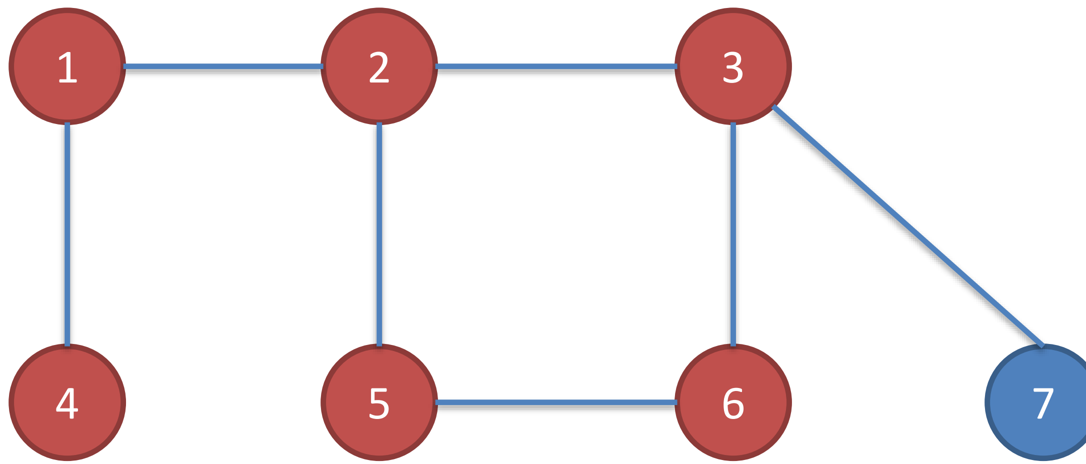
    C = ∅;
    E'=G.E;
    while(E' ≠ ∅){
        Randomly choose a edge
        (u,v) in E', put u and v
        into C;
        Remove all the edges
        that covered by u or v
        from E'
    }
    Return C;



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

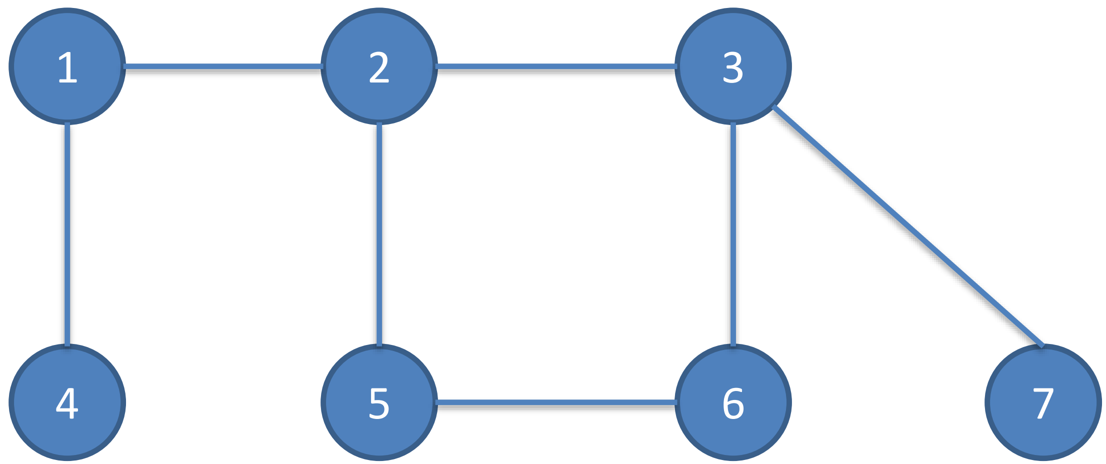
Size? 6

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

Vertex-cover problem and a 2-approximation algorithm

APPROX-VERTEX-COVER(G)

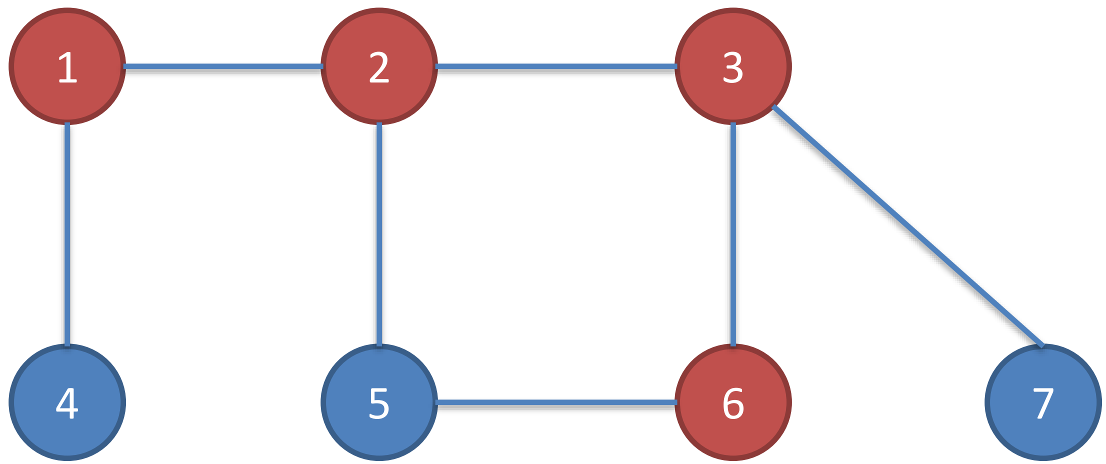
    C = ∅;
    E'=G.E;
    while(E' ≠ ∅){
        Randomly choose a edge (u,v) in E', put u and v into C;
        Remove all the edges that covered by u or v from E'
    }
    Return C;



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;



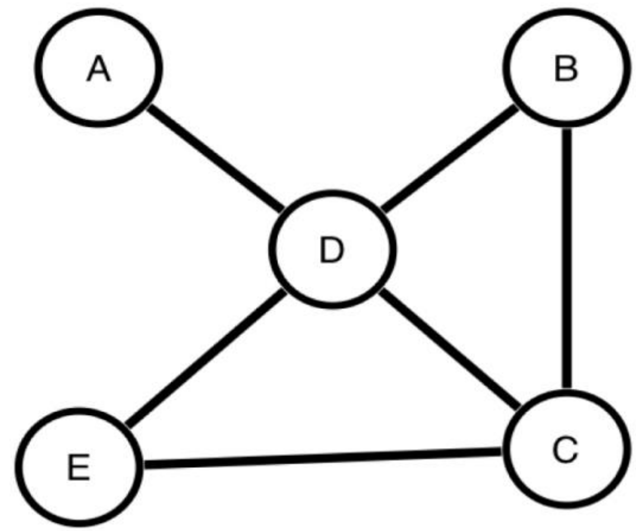
It is then a vertex cover

Size? 4

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

Applications of Vertex Cover Problem (VC)

- Minimum Vertex Cover (MVC) problem comes into play in scheduling problems.
- A scheduling problem can be modeled as a graph, where the vertices represent tasks or times, and an edge between vertices means that a conflict exists between those times or tasks.
- Finding the minimum number of tasks that needs to be removed in order to resolve all conflicts is equivalent to finding a minimum vertex cover.
- [Carruthers, Sarah, Ulrike Stege, and Michael Masson. "Human performance on hard non-Euclidean graph problems: Vertex cover." (2012).]



Vertex-cover problem and a 2-approximation algorithm

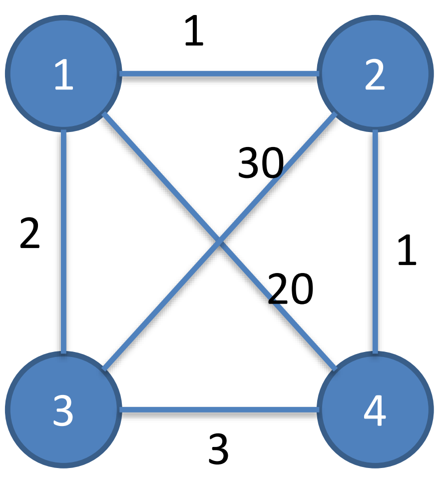
- **APPROX-VERTEX-COVER**(G) is a 2-approximation algorithm
- When the size of minimum vertex-cover is *s*
- The vertex-cover produced by **APPROX-VERTEX-COVER** is at most 2*s*

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 $\frac{|\mathrm{U}|}{|\mathrm{U}^*|} \leq 2$

Traveling-salesman problem

- **Traveling-salesman problem (TSP):**
  - Given a weighted, undirected graph with V >= 3, start from certain vertex, find a **minimum** route visit each vertices once, and return to the original vertex.



Traveling-salesman problem

- TSP is a NP-complete problem
- There is **no polynomial-time approximation** algorithm with a **constant approximation ratio**
- Another strategy to solve NPC problem:
  - **Solve a special case**

Traveling-salesman problem

- **Triangle inequality**:
  - Weight(u, v) <= Weight(u, w) + Weight(w, v)
- E.g.:
  - If all the edges are defined as the distance on a 2D map, the triangle inequality is true
- For the TSPs where the triangle inequality is true:
  - There is a 2-approximation polynomial time algorithm

Metric TSP

- Metric TSP is an special case of the TSP that satisfies the triangle inequality.
- Each vertex should be connected with every other vertex.

Traveling-salesman problem

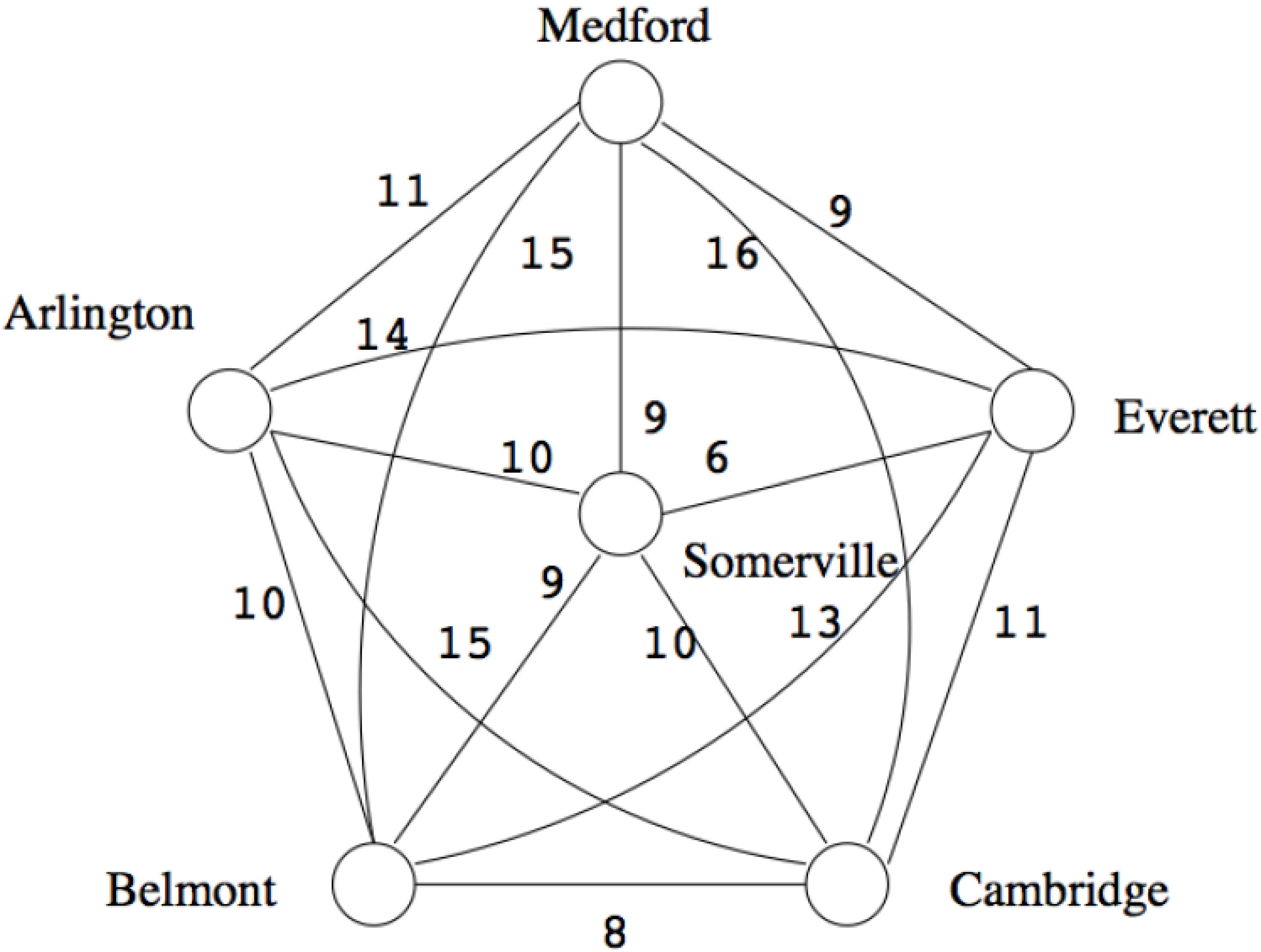
APPROX-TSP-TOUR(G)

Minimum Spanning Tree;

Creating a Cycle;

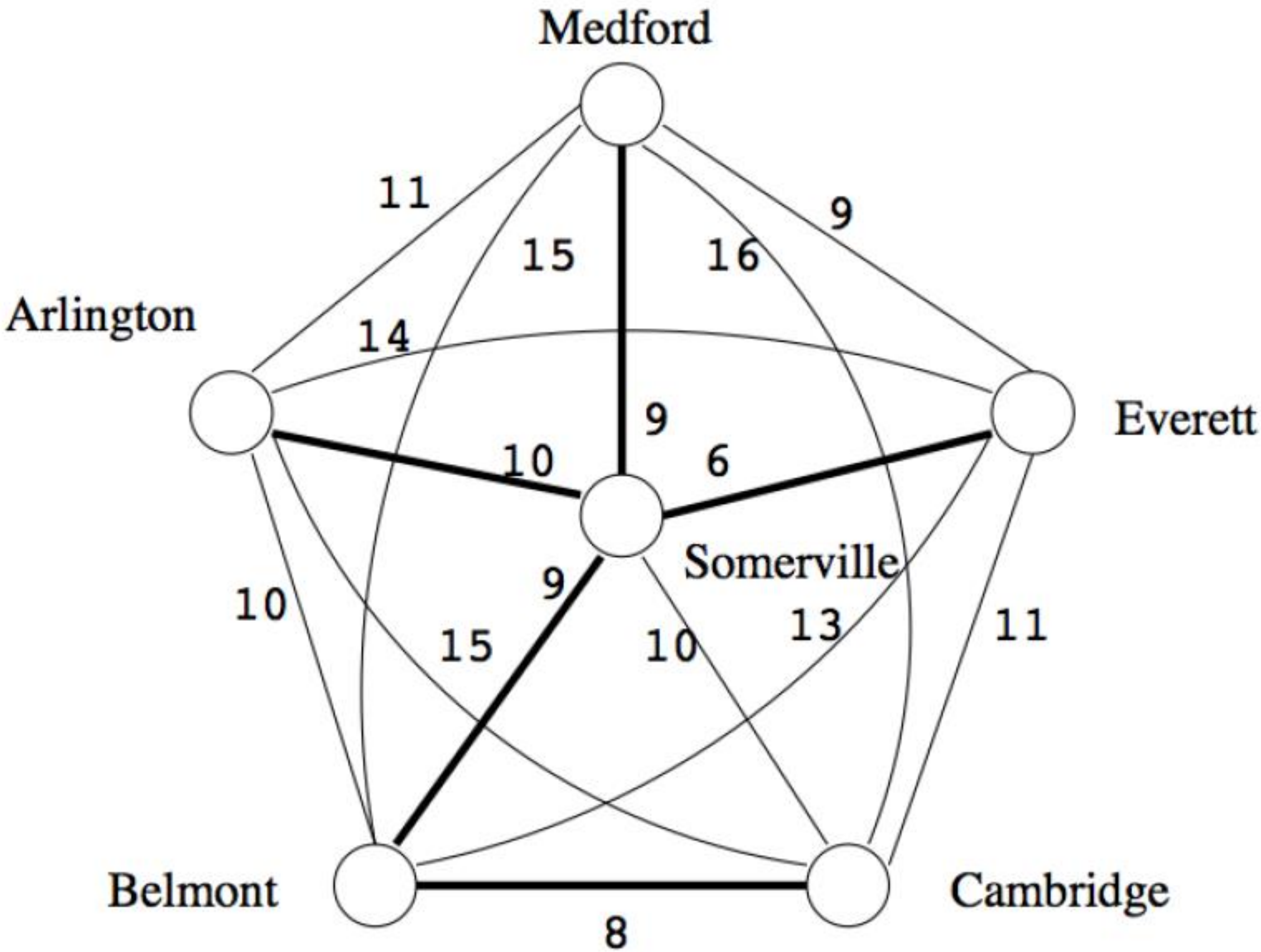
Removing Redundant Visits;

Metric TSP



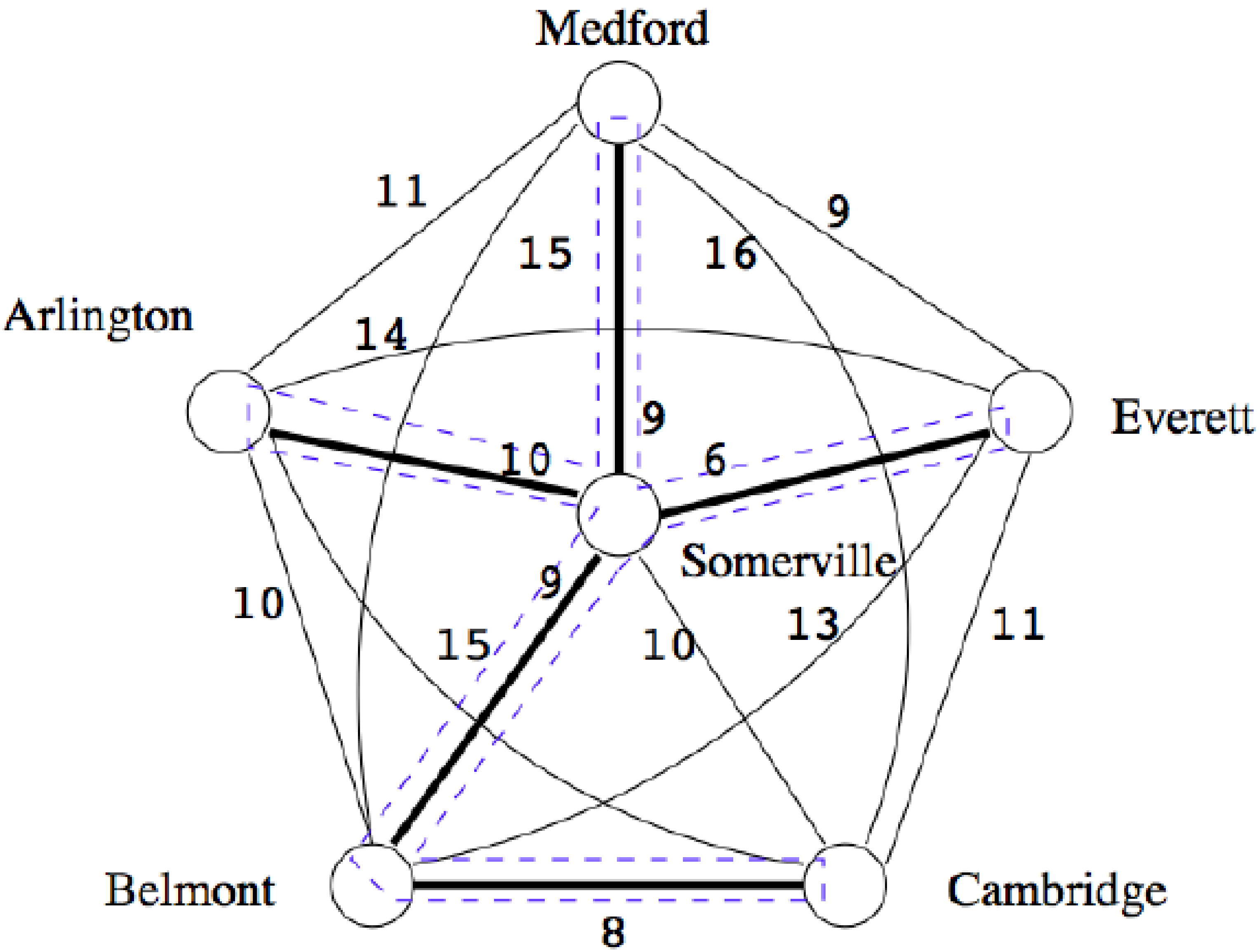
Medford
Arlington
Everett
Somerville
Belmont
Cambridge
11
15
16
9
14
9
10
6
10
9
15
10
13
11
8

Step 1: MST

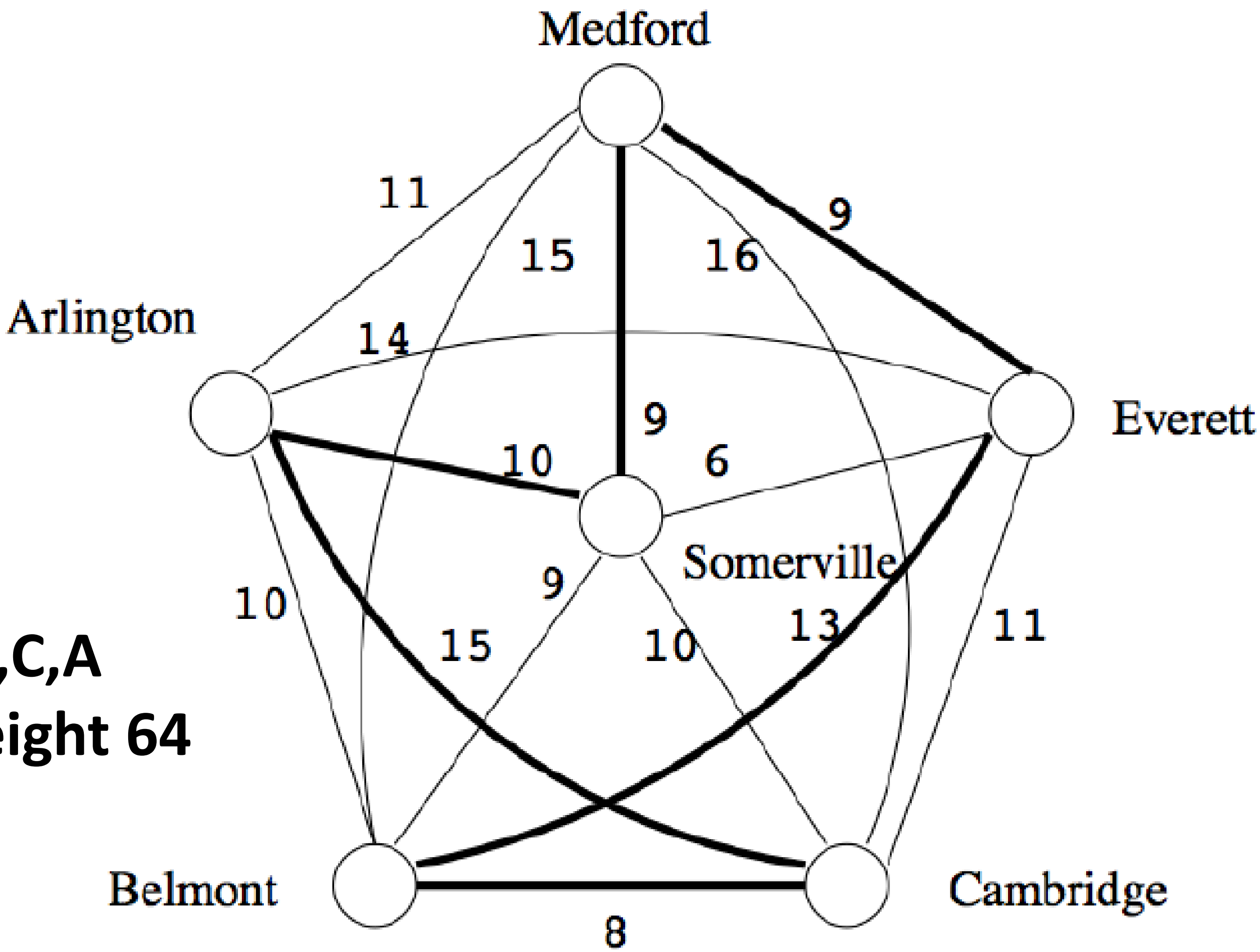


Medford
11
15
16
9
Arlington
14
Everett
9
10
6
Somerville
10
9
15
10
13
11
Belmont
8
Cambridge

Step 2: Creating a cycle - using DFS



Step 3: Removing Redundant Visits of DFS



S,M,E,B,C,A with weight 64

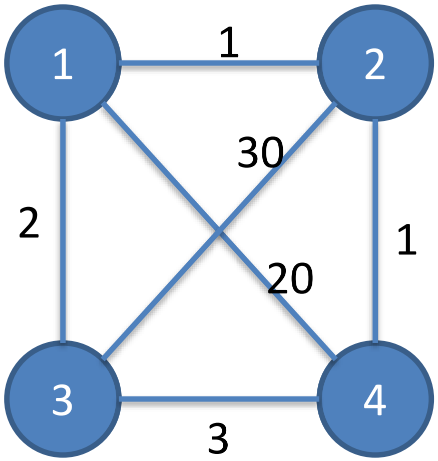
2 Approximation Algorithm

- **Claim:** The weight of the MST M is less than OPT, the weight of the TSP solution T.
- Take T and remove an edge e. T is now a spanning tree.
- Because M is the MST,
  - $w(M) \leq w(T-e)$
  - $w(M) \leq w(T)-w(e)$
  - $w(M) \leq OPT-w(e)$
- Therefore, $w(M) < OPT$.
- Because each edge is used exactly twice during a depth first search on a tree (once descending, once ascending)
- $w(W) = 2 * w(M) < 2 * OPT$.

≤

Traveling-salesman problem

Can we apply the approximation algorithm on this one?



No. The triangle inequality is violated.

Traveling-salesman problem (Method 2)

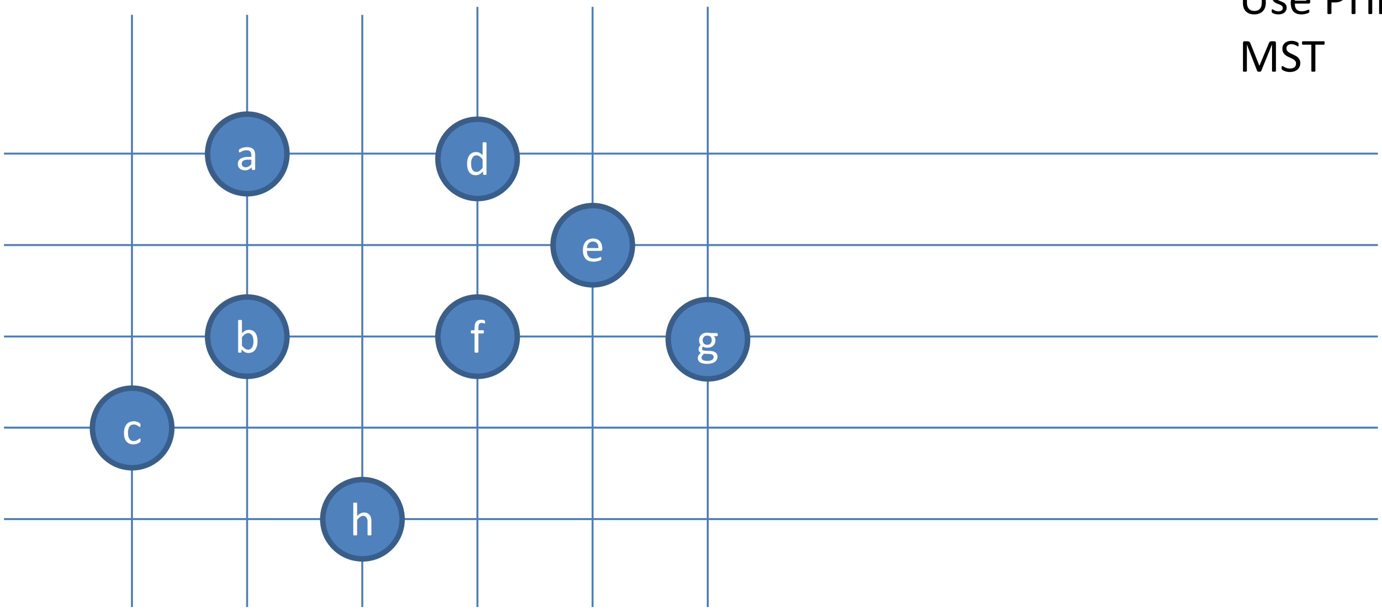
APPROX-TSP-TOUR(G)

Find a MST m;

Choose a vertex as root r;

return preorderTreeWalk(m, r);

Traveling-salesman problem (Method 2)

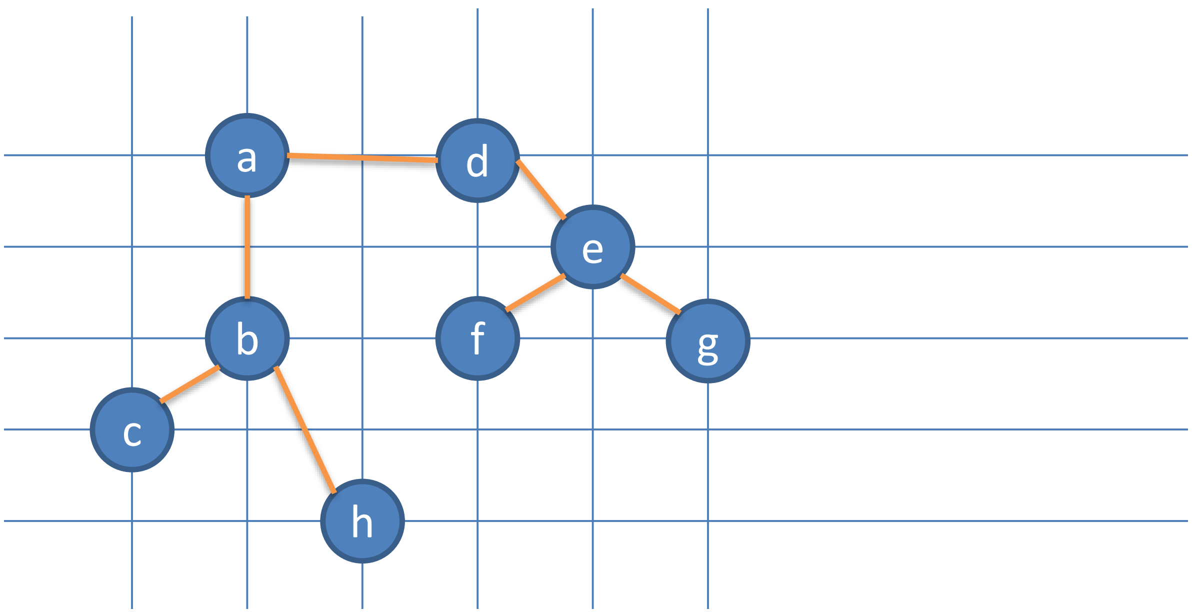


Use Prim's algorithm to get a MST

For any pair of vertices, there is a edge and the weight is the Euclidean distance

Triangle inequality is true, we can apply the approximation algorithm

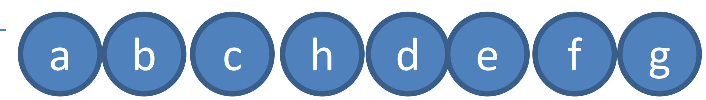
Traveling-salesman problem (Method 2)



Use Prim's algorithm to get a MST

Choose "a" as the root

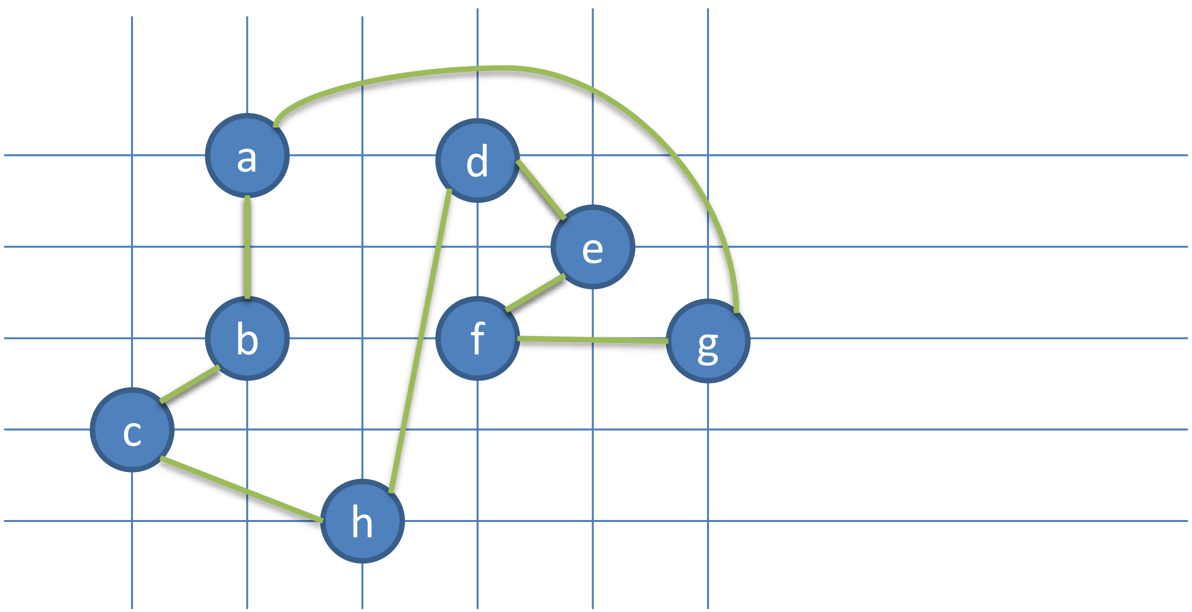
Preorder tree walk



For any pair of vertices, there is a edge and the weight is the Euclidean distance

Triangle inequality is true, we can apply the approximation algorithm

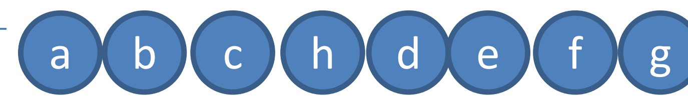
Traveling-salesman problem (Method 2)



Use Prim's algorithm to get a MST

Choose "a" as the root

Preorder tree walk



The route is then…

Because it is a 2-approximation algorithm

A TSP solution is found, and the total weight is at most twice as much as the optimal one

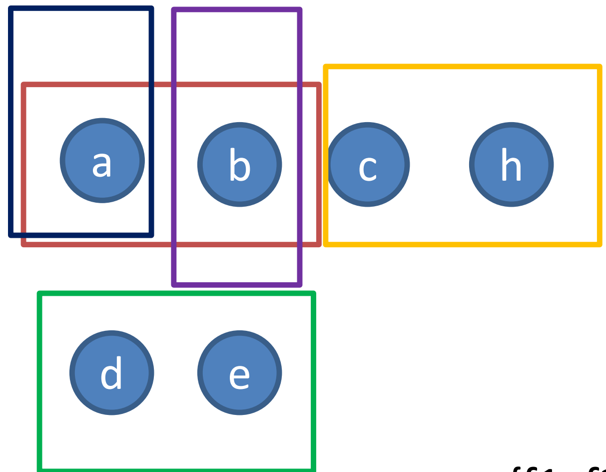
The set-covering problem

Set-covering problem

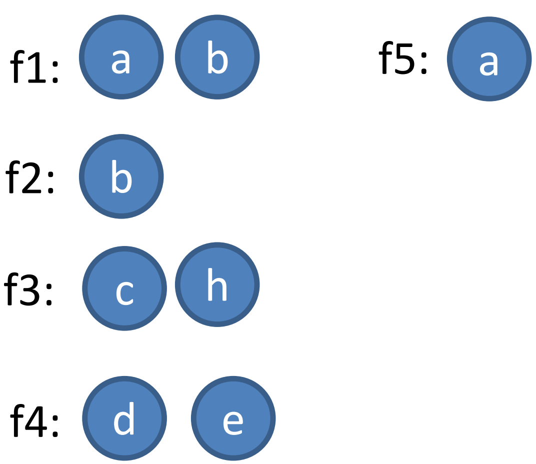
- Given a set X, and a family F of subsets of X, where F covers X, i.e. $\mathrm{X} = \bigcup_{S \in \mathrm{F}} S$.
- Find a subset of F that covers X and with minimum size

The set-covering problem

X:



F:



{f1, f3, f4} is a subset of F covering X

{f1, f2, f3, f4} is a subset of F covering X

{f2, f3, f4, f5} is a subset of F covering X

Here, {f1, f3, f4} is a minimum cover set

The set-covering problem

- **Set-covering problem** is NP-complete.
- If the size of the largest set in F is m, there is a $\sum_{i=1}^{m} 1/i$ - approximation polynomial time algorithm to solve it.

The set-covering problem

- **GREEDY-SET-COVER**(X, F)

    U=X;

    C=∅;

    While(U≠ ∅){

        Select S∈F that maximizes |S∩U|;

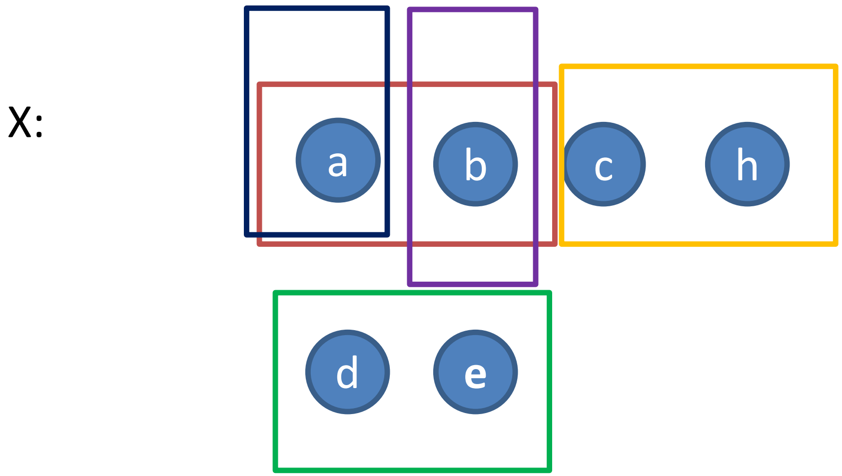
        U=U-S;

        C=C∪{S};

    }

    return C;

The set-covering problem



We can choose from f1, f3 and f4

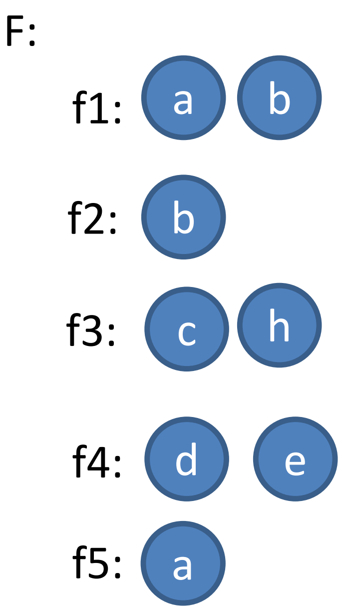
Choose f1

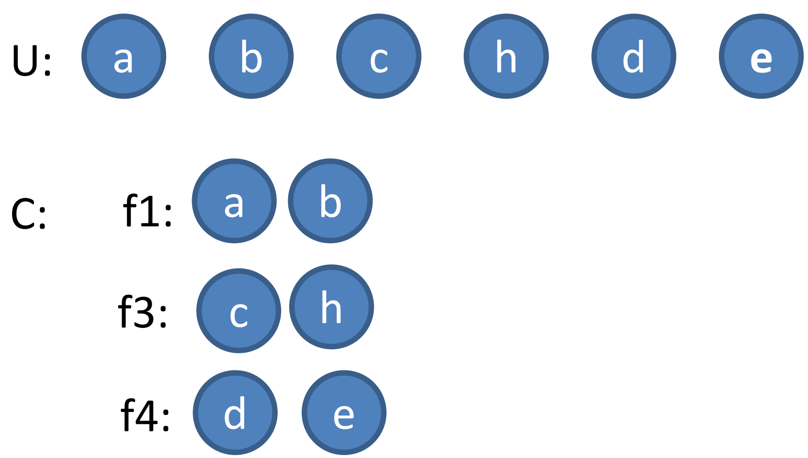
We can choose from f3 and f4

Choose f3

We can choose from f4

Choose f4





Set Cover and its generalizations and variants are fundamental problems with numerous applications. Examples include:

- selecting a small number of nodes in a network to store a file so that all nodes have a nearby copy,
- selecting a small number of sentences to be uttered to tune all features in a speech-recognition model [11],
- selecting a small number of telescope snapshots to be taken to capture light from all galaxies in the night sky,
- finding a short string having each string in a given set as a contiguous sub-string.

Summary NP-naming convention

- **NP-complete -** means problems that are 'complete' in NP, i.e. the most difficult to solve in NP
- **NP-hard** - stands for 'at least' as hard as NP (but not necessarily **in** NP);
- **NP-easy** - stands for 'at most' as hard as NP (but not necessarily **in** NP);
- **NP-equivalent** - means equally difficult as NP, (but not necessarily **in** NP);

Examples NP-complete and NP-hard problems

Hamiltonian Paths NP-complete

Optimization Problem: Given a graph, find a path that passes through every vertex exactly once

Decision Problem: Does a given graph have a Hamiltonian Path ?

Traveling Salesman NP-hard

Optimization Problem: Find a minimum weight Hamiltonian Path

Decision Problem: Given a graph and an integer k, is there a Hamiltonian Path with a total weight at most k ?

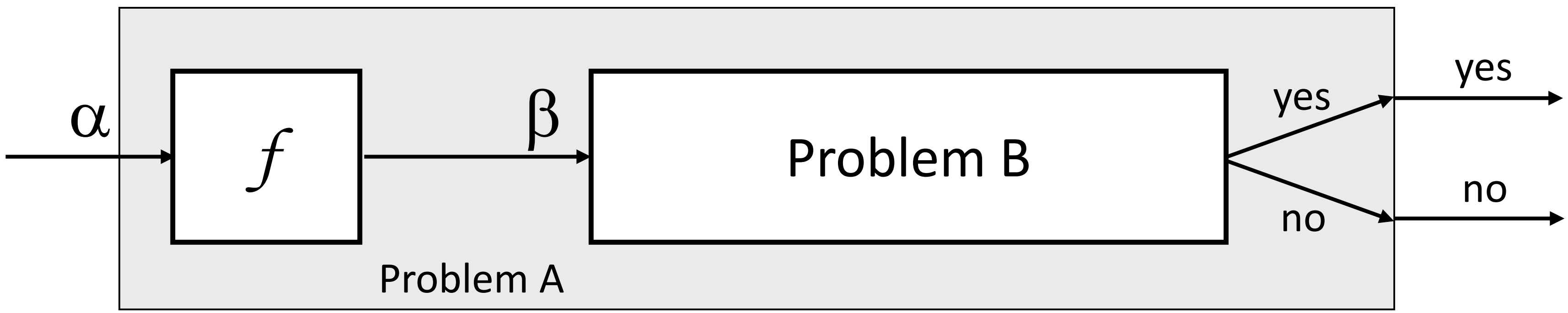
NP-Completeness Reduction

Reductions

- Reduction is a way of saying that one problem is **"easier"** than another.
- We say that problem A is easier than problem B,

  (i.e., we write **"A $\leq$ B"**)

  if we can solve A using the algorithm that solves B.
- **Idea:** transform the inputs of A to inputs of B

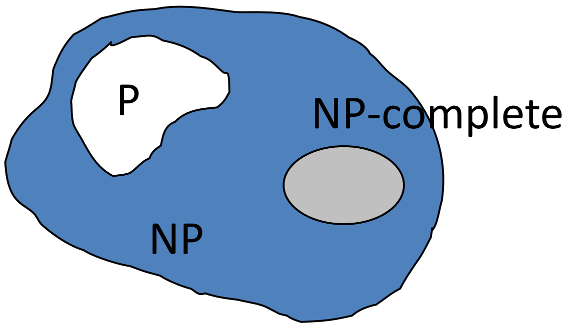


Polynomial Reductions

- Given two problems A, B, we say that A is polynomially **reducible** to B ($A \leq_p B$) if:

1. There exists a function f that converts the input of A to inputs of B in polynomial time
2. $A(i) = YES \Leftrightarrow B(f(i)) = YES$

NP-Completeness (formally)



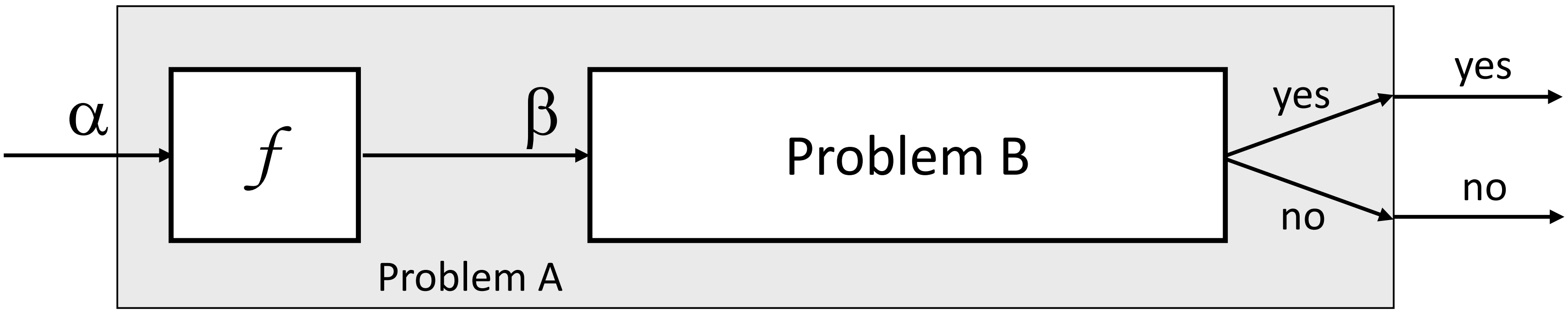
- A problem B is **NP-complete** if:

  (1) B $\in$ **NP**

  (2) A $\leq_p$ B for all A $\in$ **NP**

- If B satisfies only property (2) we say that B is **NP-hard**
- No polynomial time algorithm has been discovered for an **NP-Complete** problem
- No one has ever proven that no polynomial time algorithm can exist for any **NP-Complete** problem
- if any NP-complete problem can be solved in polynomial time, then every problem in NP has a polynomial-time solution.

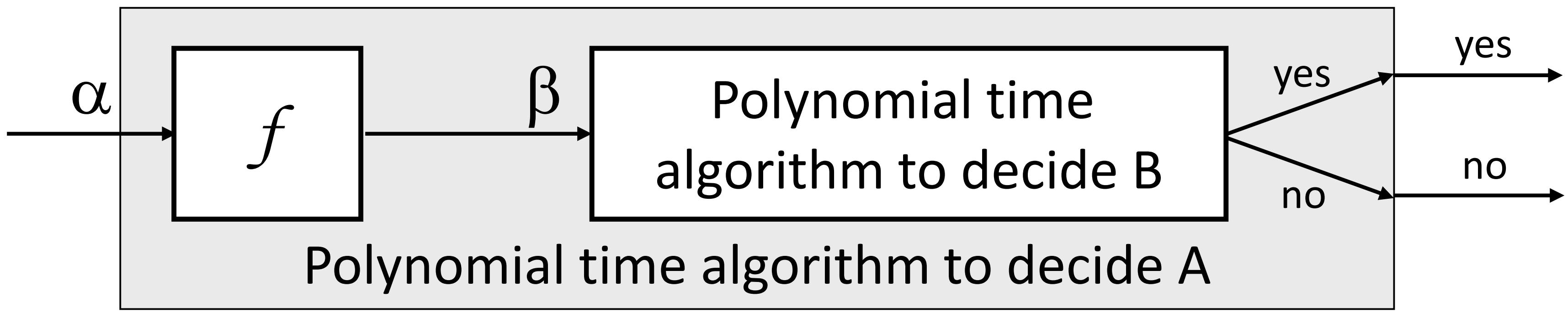
Implications of Reduction



- If $A \leq_p B$ and $B \in P$, then $A \in P$

- if $A \leq_p B$ and $A \notin P$, then $B \notin P$

Proving Polynomial Time

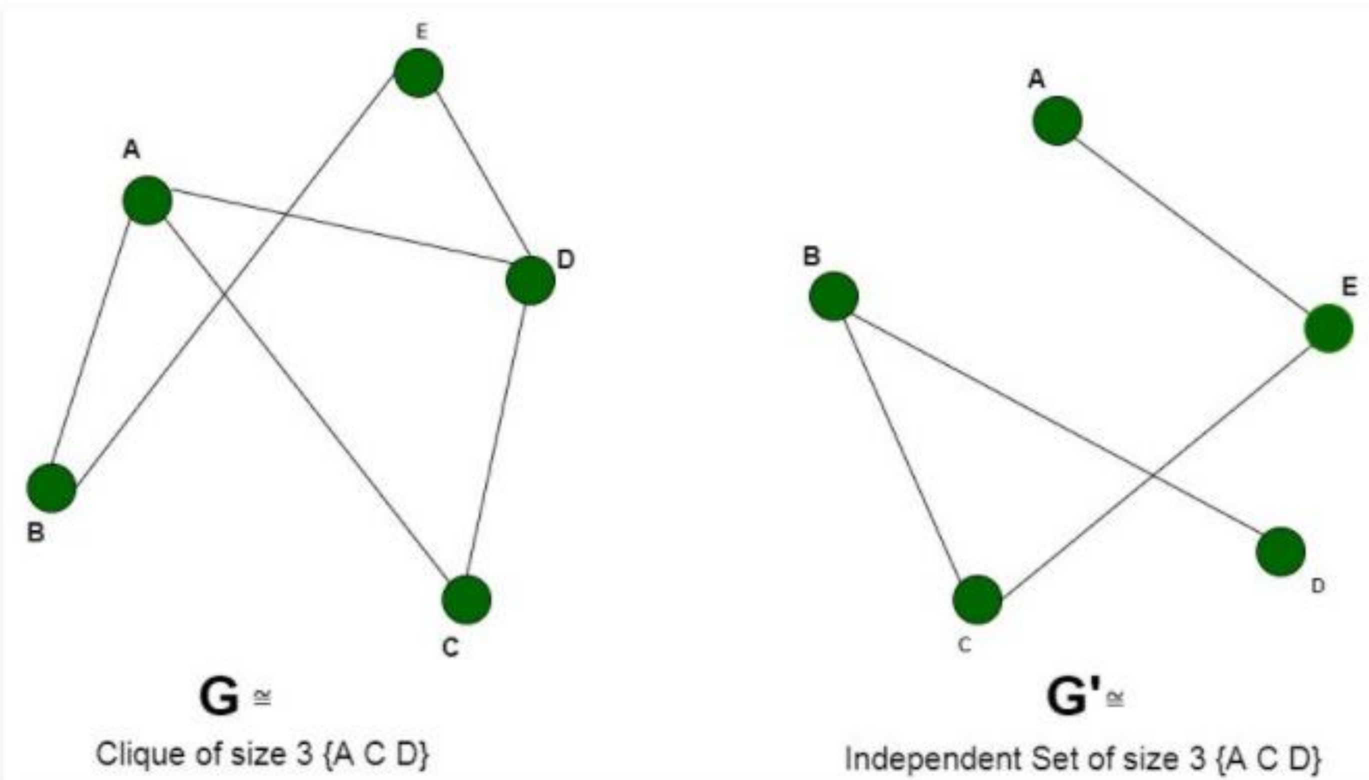


1. Use a **polynomial time** reduction algorithm to transform A into B
2. Run a known **polynomial time** algorithm for B
3. Use the answer for B as the answer for A

Proving NP-Completeness In Practice

- Prove that the problem B is in NP
  - A randomly generated string can be checked in polynomial time to determine if it represents a solution
- Show that **one known** NP-Complete problem can be transformed to B in polynomial time
  - No need to check that **all** NP-Complete problems are reducible to B

Independent Set & Clique Problem



Reduction: Clique to Independent Set

Clique and Independent Set problems

For a given graph $G = (V, E)$ and integer k, the Clique problem is to find whether G contains a clique of size $>= k$.

For a given graph $G' = (V', E')$ and integer k', the Independent Set problem is to find whether G' contains an Independent Set of size $>= k'$.

Reduction of Clique to Independent Set

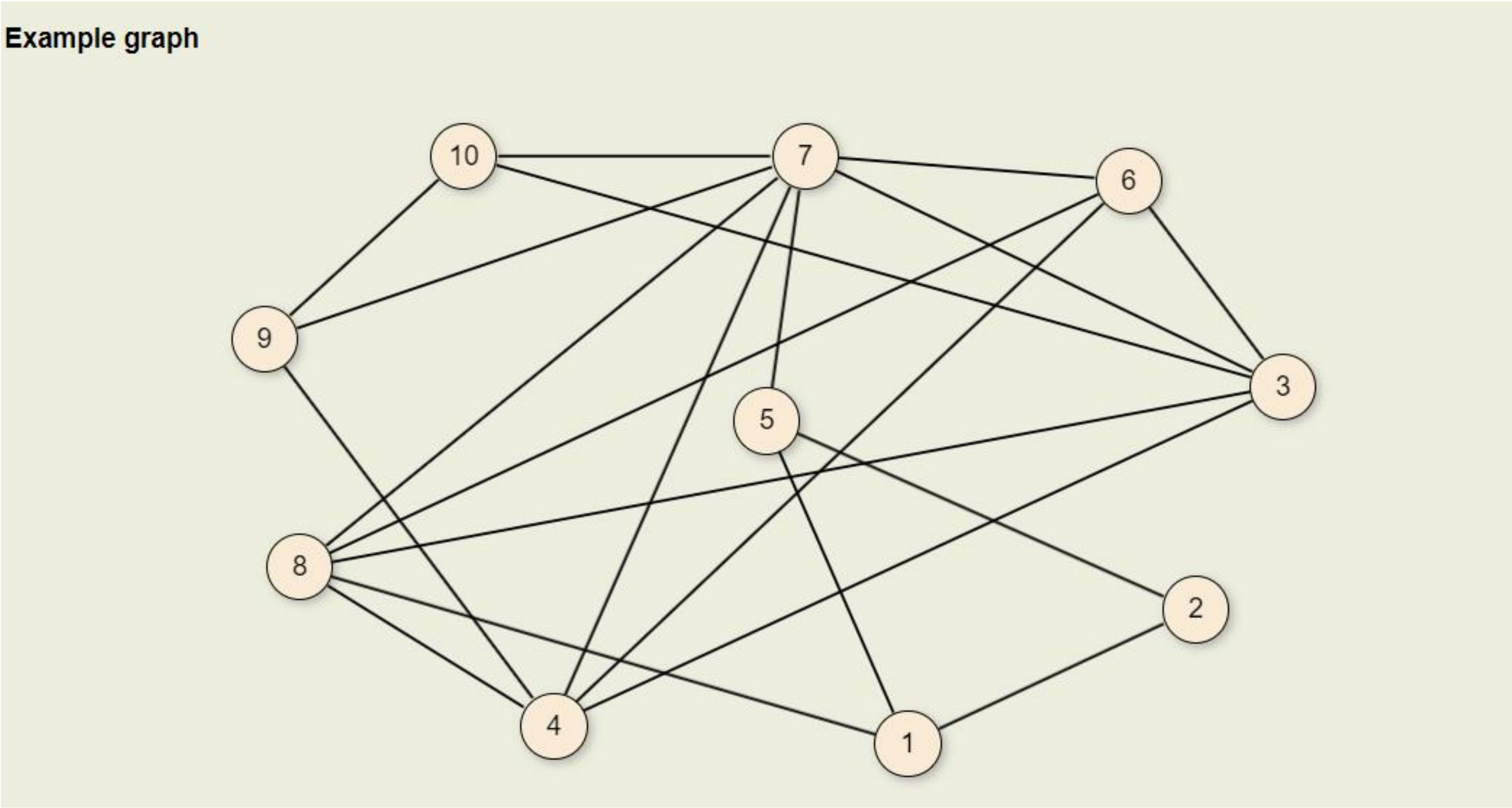
To reduce a Clique Problem to an Independent Set problem for a given graph $G = (V, E)$, construct a complimentary graph $G' = (V', E')$ such that

1. $V = V'$, that is the compliment graph will have the same vertices as the original graph

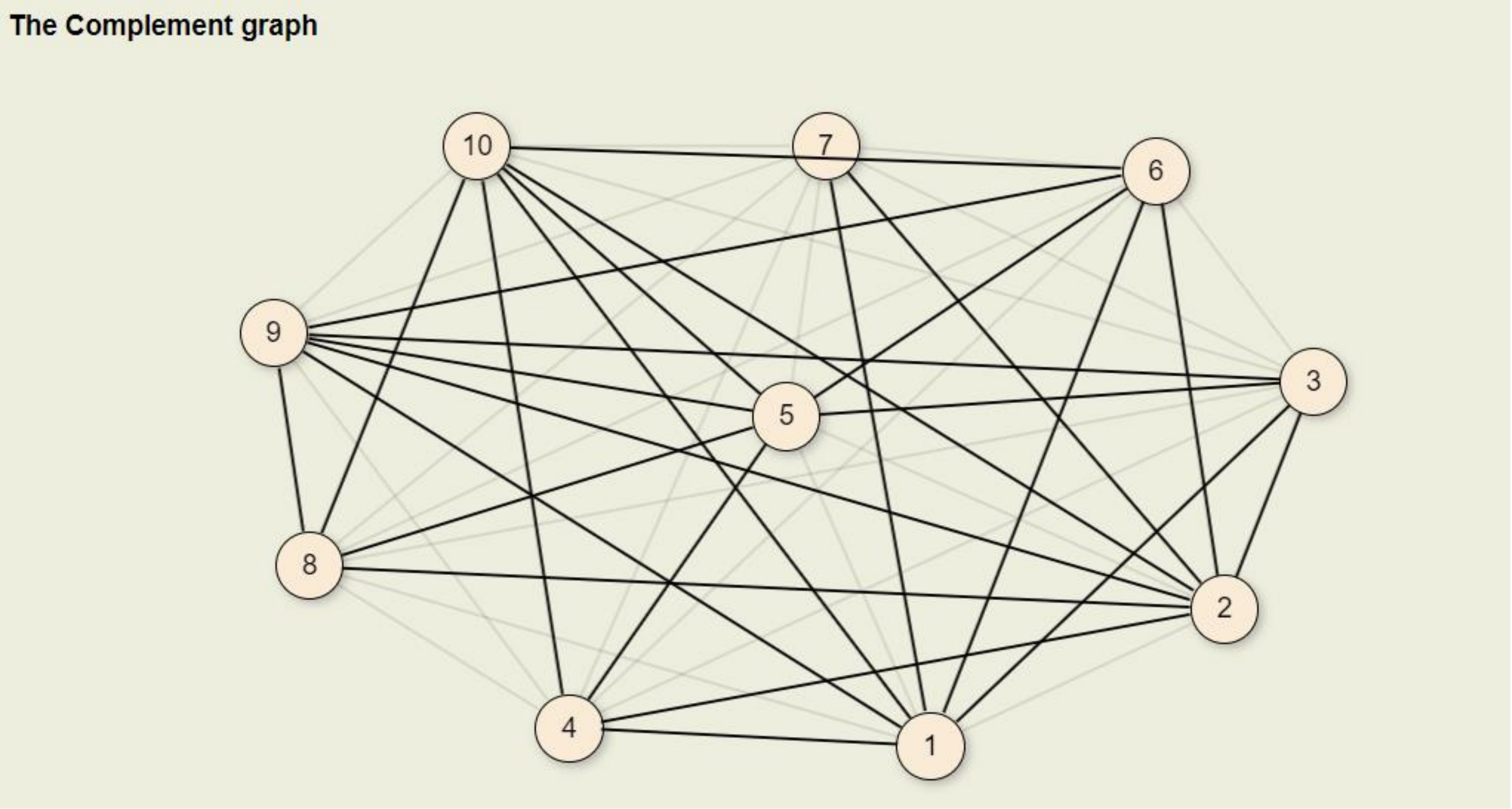
2. E' is the compliment of E that is G' has all the edges that is **not** present in G

Note: Construction of the complimentary graph can be done in polynomial time

Reduction: Clique to Independent Set



Reduction: Clique to Independent Set



Reduction: Clique to Independent Set

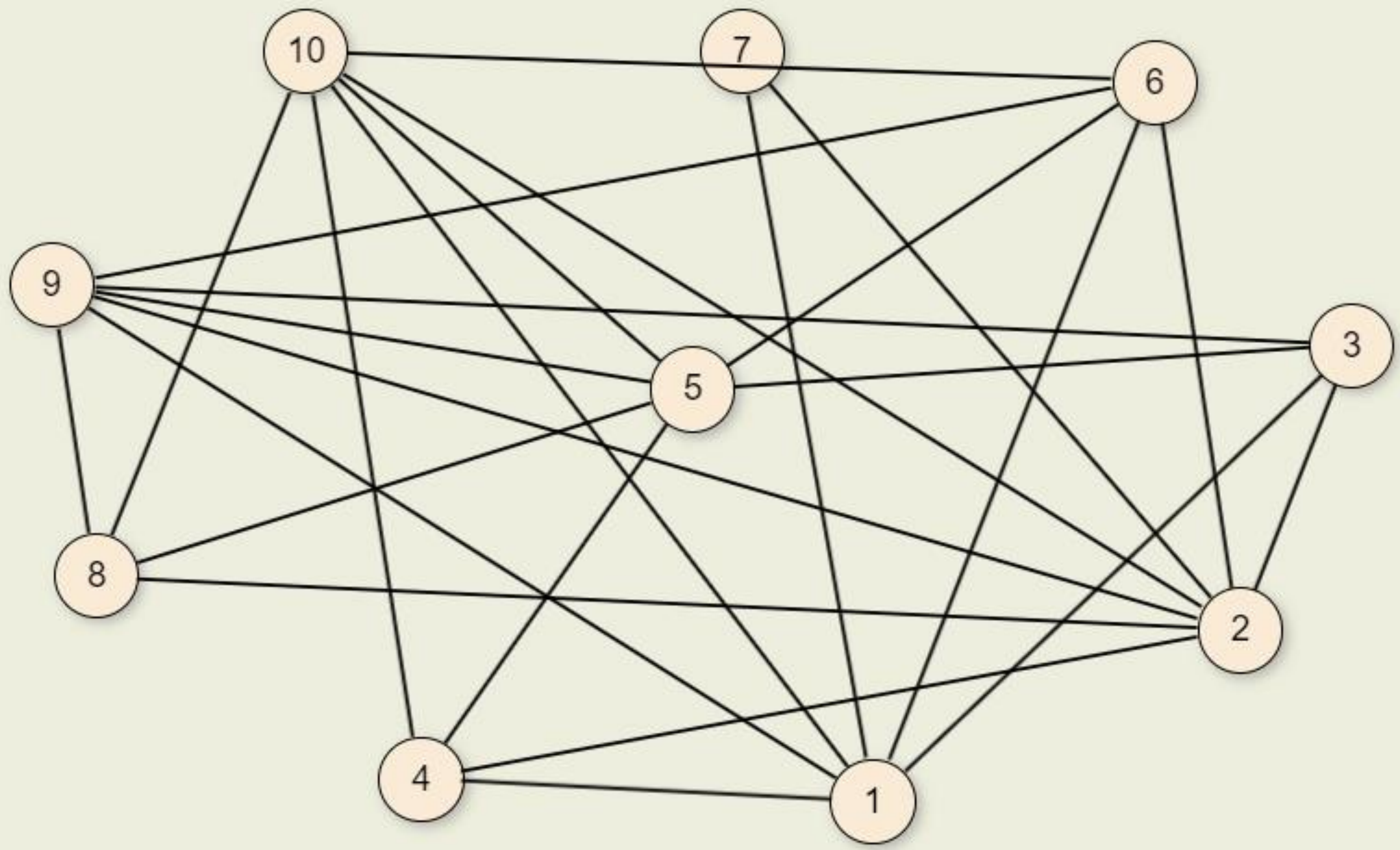
Clique problem reduced to Independent Set

1. **If there is an independent set of size k in the complement graph** G', it implies no two vertices share an edge in G' which further implies all of those vertices share an edge with all others in G forming a clique. that is **there exists a clique of size k in** G

2. **If there is a clique of size** k **in the graph** G, it implies all vertices share an edge with all others in G which further implies no two of these vertices share an edge in G' forming an Independent Set. that is **there exists an independent set of size k in** G'

Reduction: Clique to Independent Set

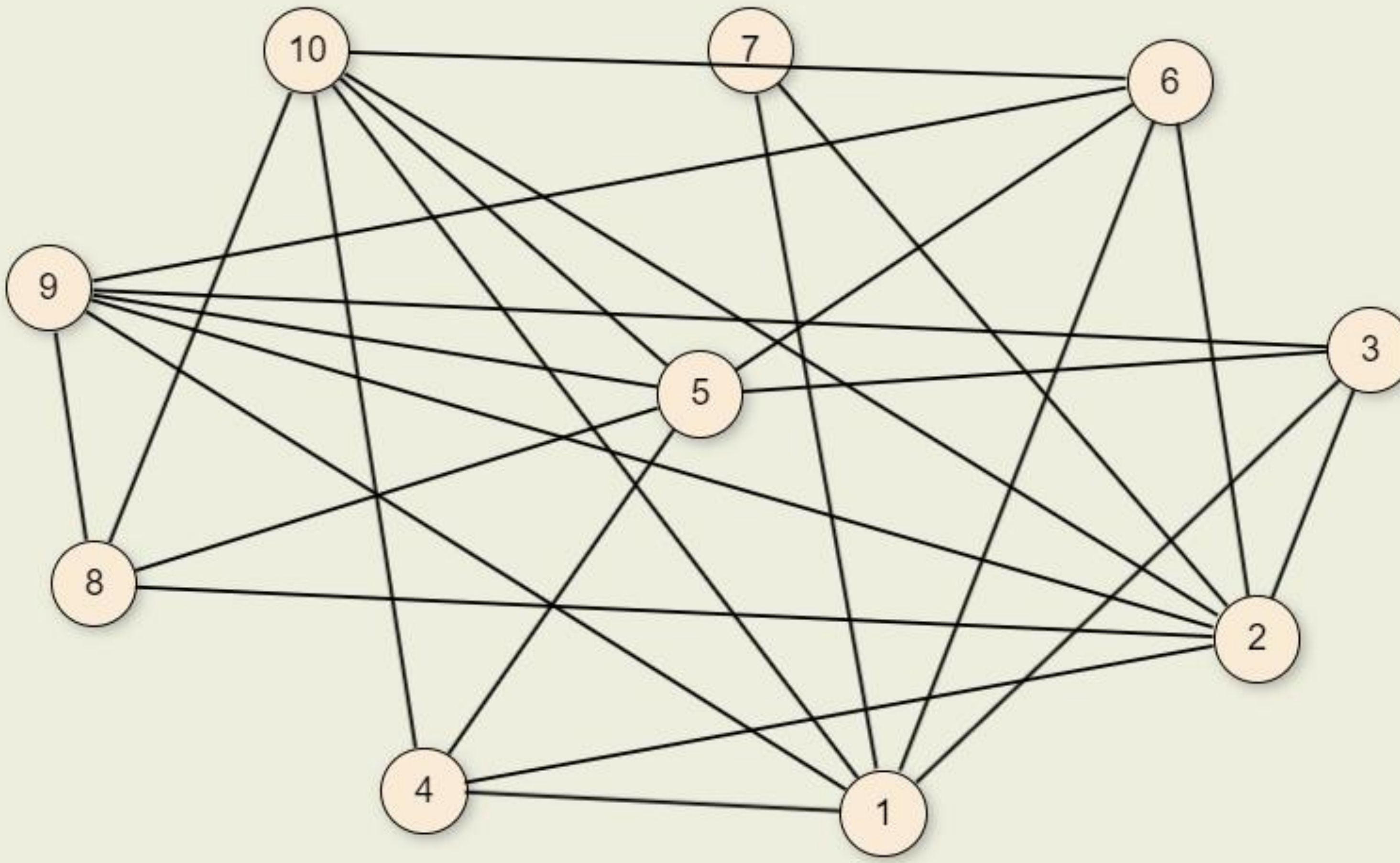
Does G' below have an independent set of size 8?



Reduction: Clique to Independent Set

Does G' below have an independent set of size 8?

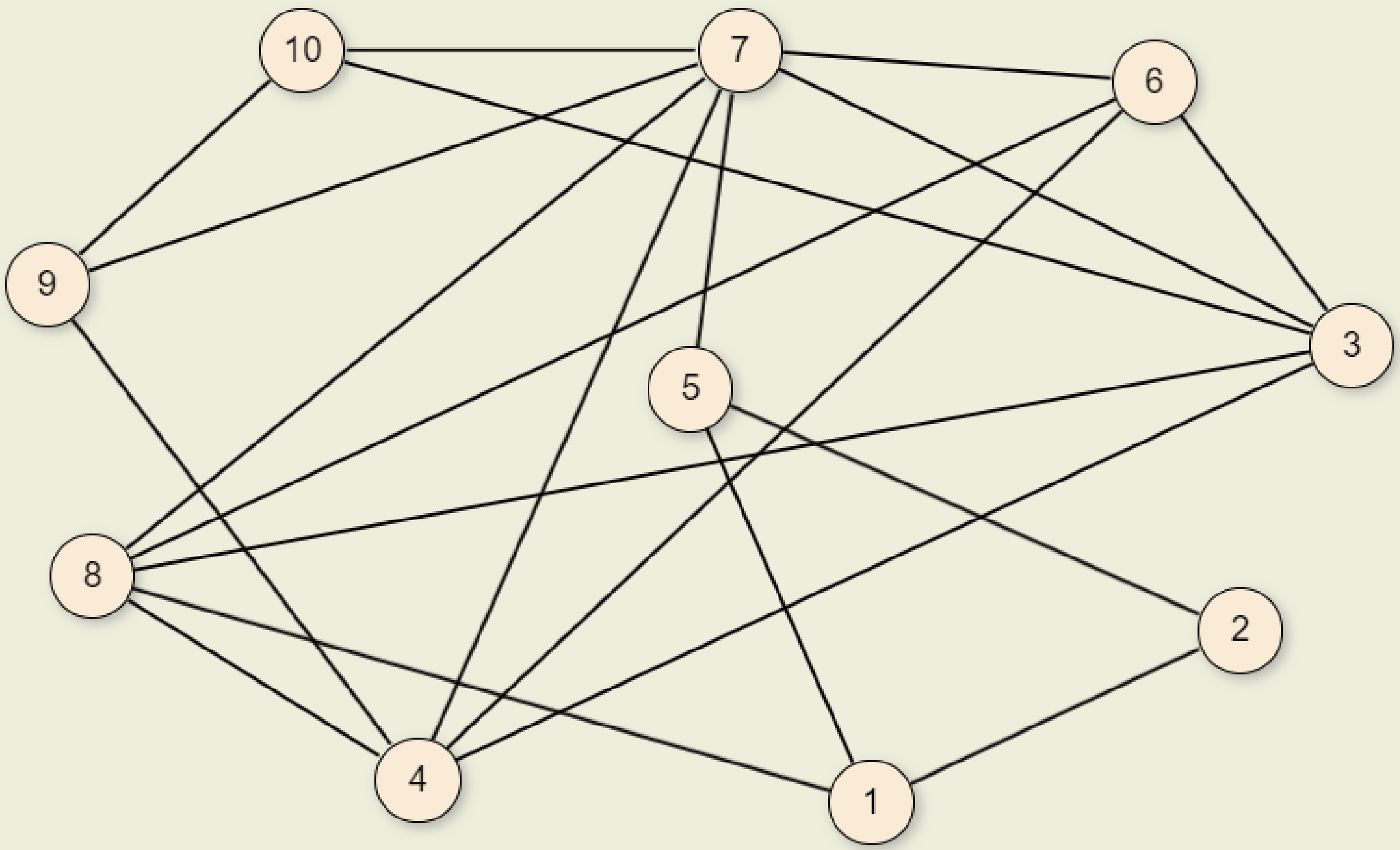
NO



Reduction: Clique to Independent Set

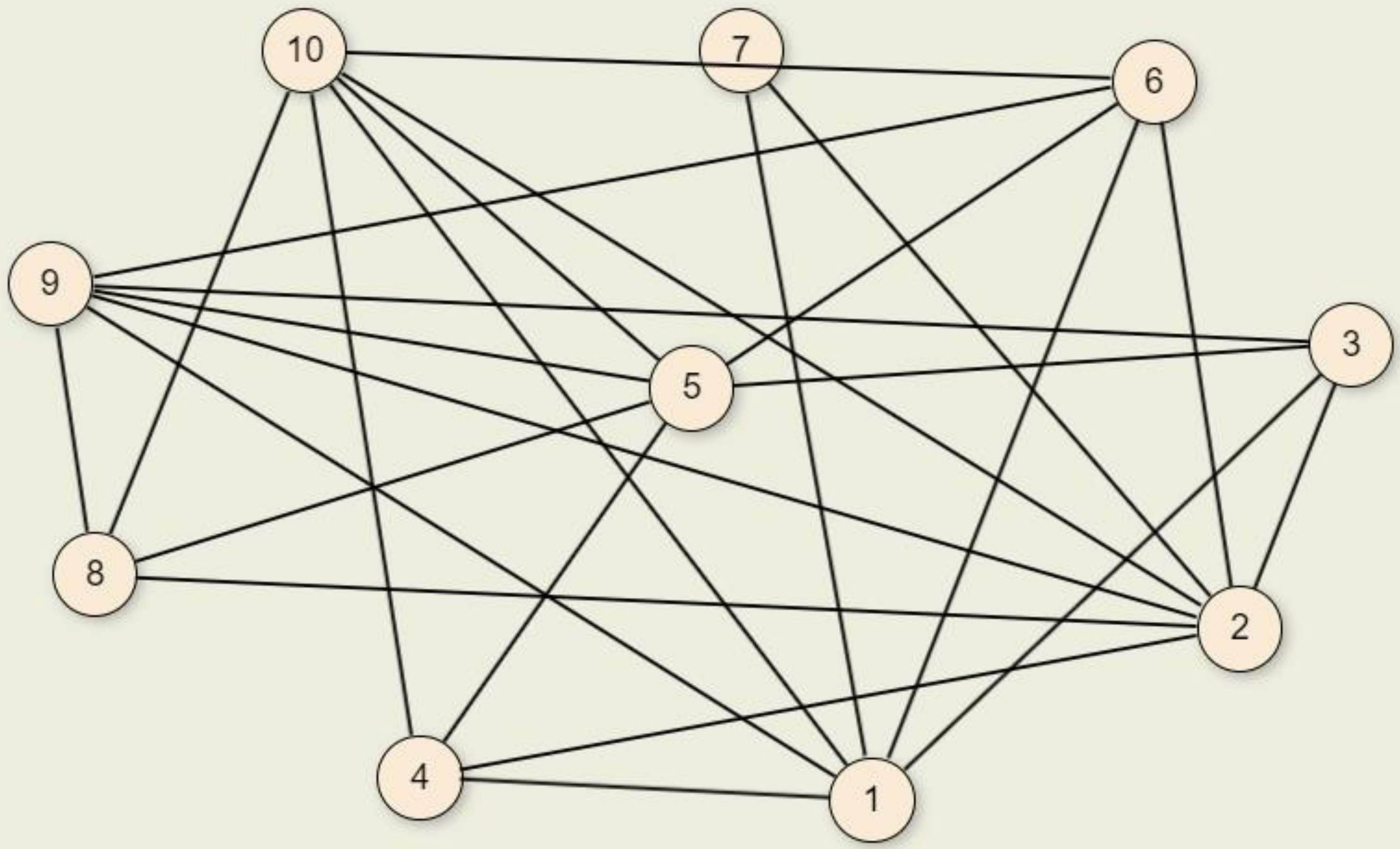
Does G below have a clique of size 8?

NO



Reduction: Clique to Independent Set

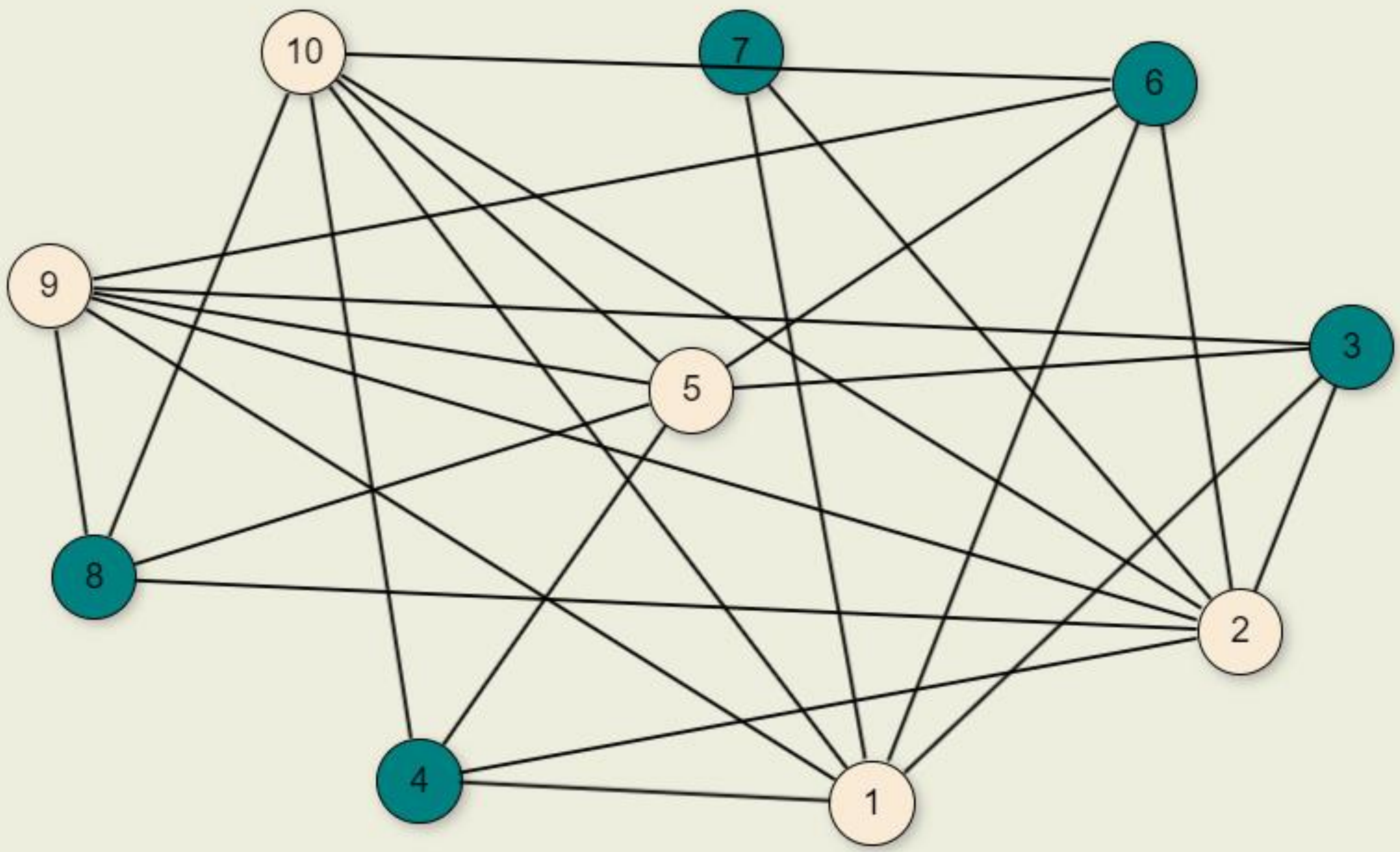
Does G' below have an independent set of size 5?



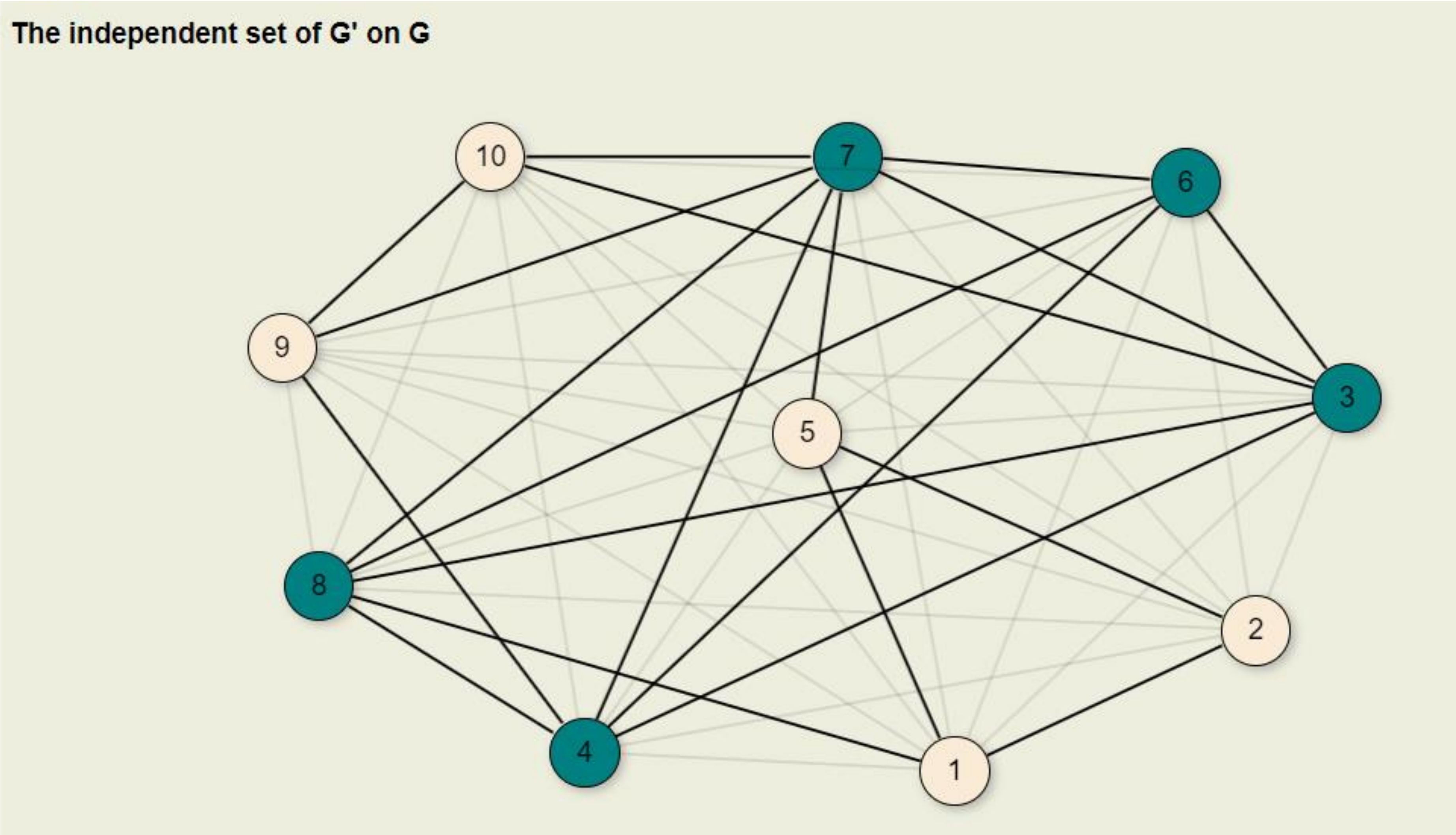
Reduction: Clique to Independent Set

Does G' below have an independent set of size 5?

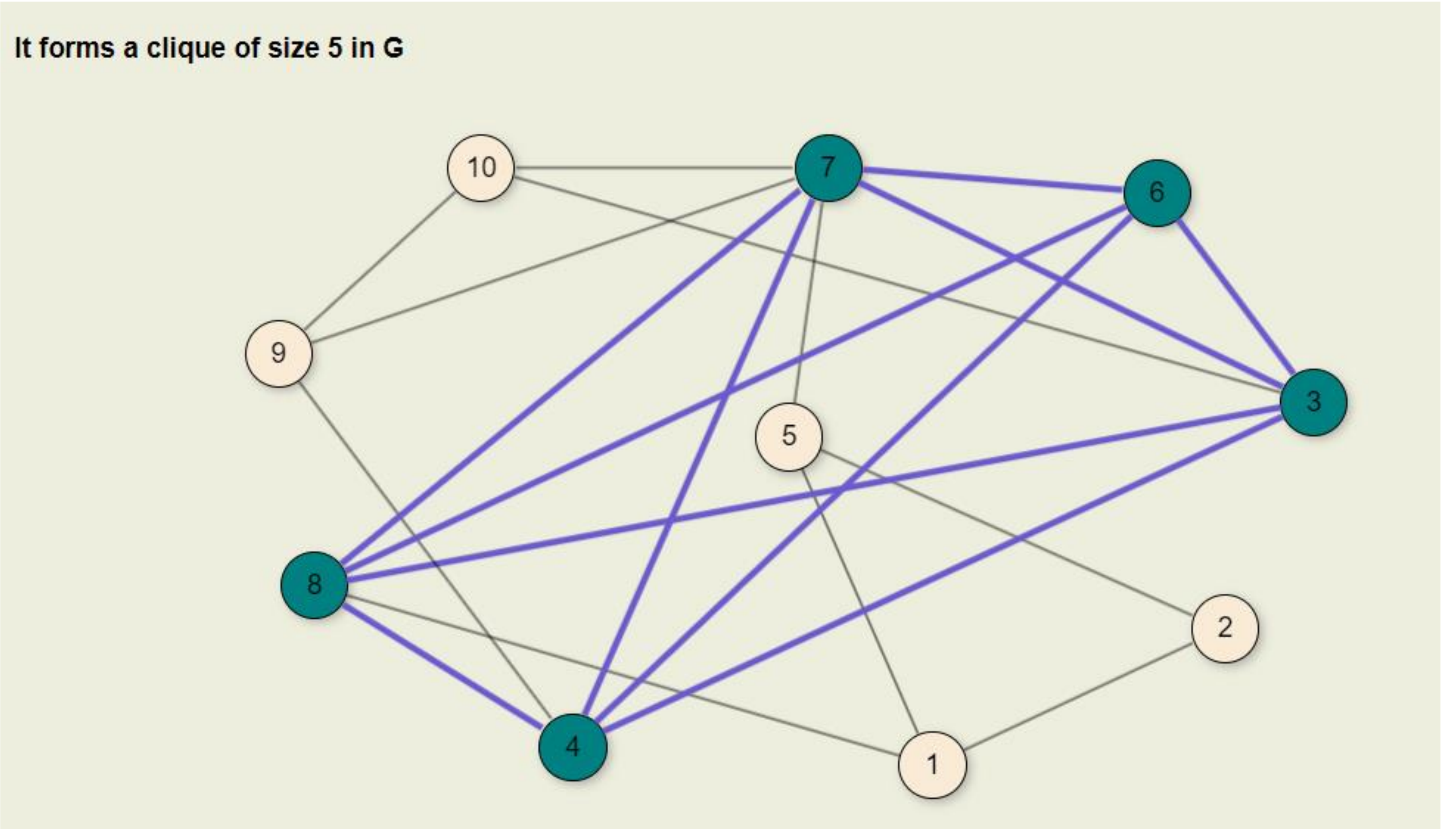
YES



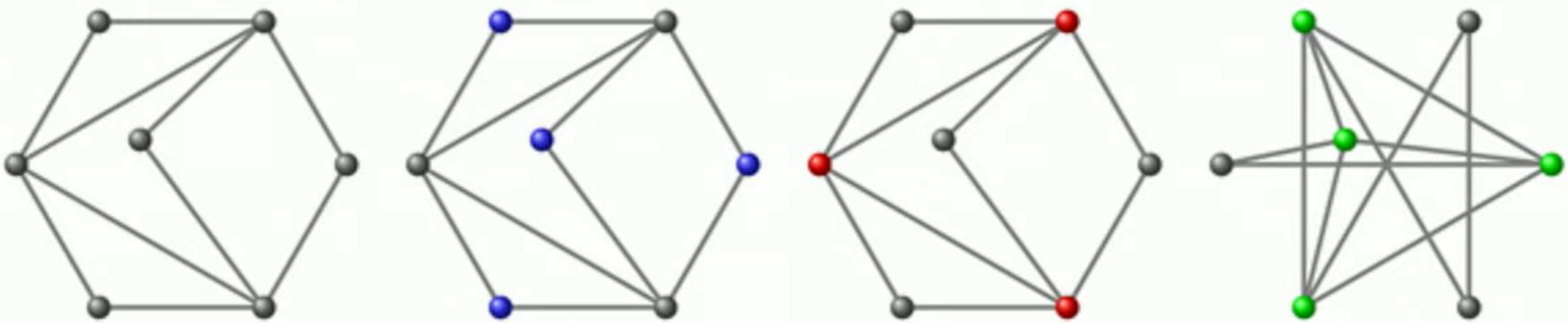
Reduction: Clique to Independent Set



Reduction: Clique to Independent Set



Reduction: Independent Set, Vertex Cover, and Clique



(a) Original Graph (b) Independent Set (c) Vertex Cover (d) Clique

Figure 1: Relations among Independent Set, Vertex Cover, and Clique