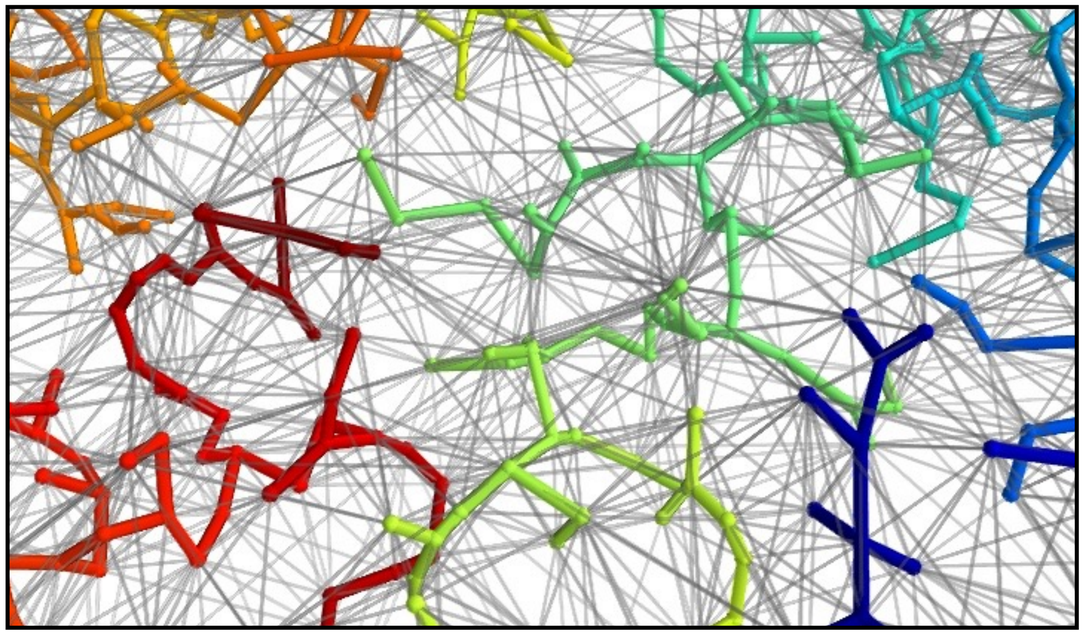
CS251

Great Ideas in *Theoretical* Computer Science

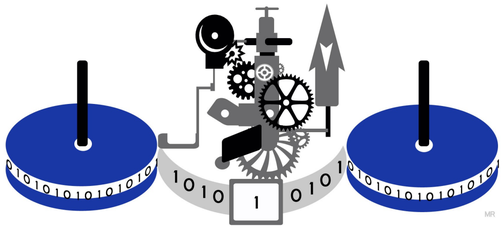


Maximum Matchings

Some real-world examples

matching machines and jobs

each machine is capable of doing a subset of the jobs



Job 1

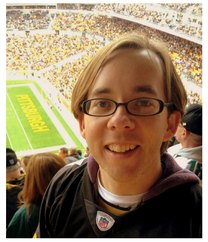
Job 2

⋮

Job n

Some real-world examples

matching professors and courses



⋮

15-110

15-112

15-122

15-150

15-251

15-259

⋮

Some real-world examples

matching rooms and courses

GHC 4401	15-110
DH 2210	15-112
GHC 5222	15-122
WEH 7500	15-150
DH 2315	15-251
⋮	⋮

Some real-world examples

matching students and internships

Macrosoft

Moogle

UMBRELLA CORPORATION

"Our business is life itself..."

How do you solve a problem like this?

1. Formulate the problem

2. **Ask**: Is there a trivial algorithm? Find and analyze.

3. **Ask**: Is there a better algorithm? Find and analyze.

Step 1: Formulate the problem

Purpose:

- Get rid of distractions, identify the crux.

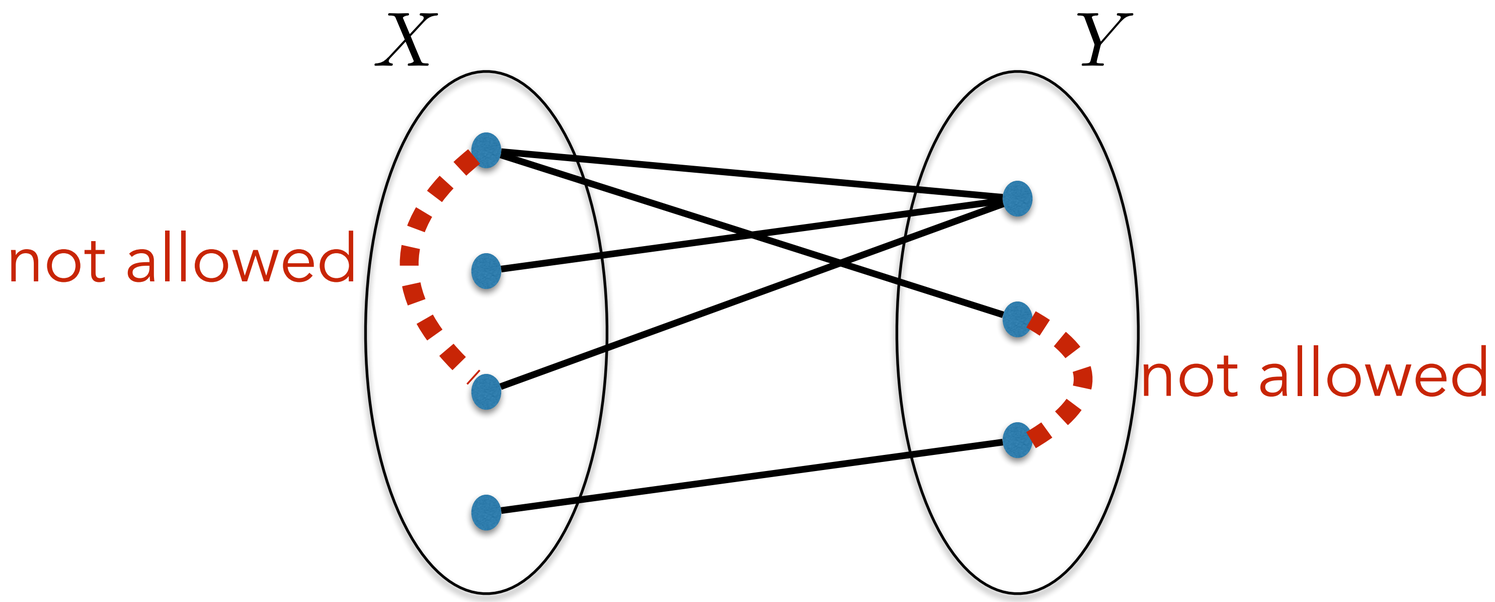
- Get a clean mathematical model (easier to reason about).

- Solutions often generalize to other settings.

CS Life Lesson

If your problem has a graph, great!!!
If not, try to make it have a graph.

Bipartite Graphs

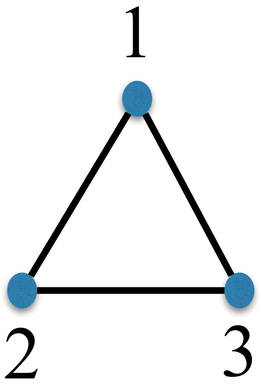


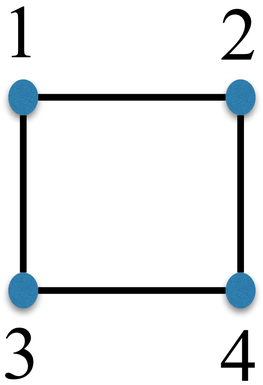
$G = (V, E)$ is **bipartite** if:

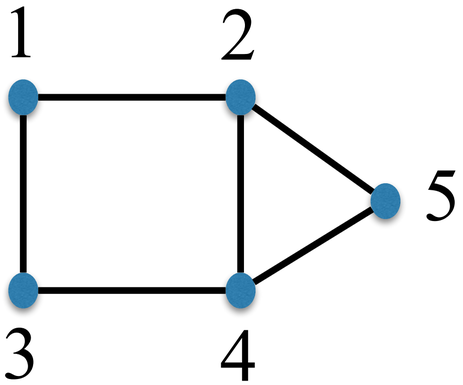
- there exists a bipartition of V into X and Y,
- each edge connects a vertex in X to a vertex in Y.

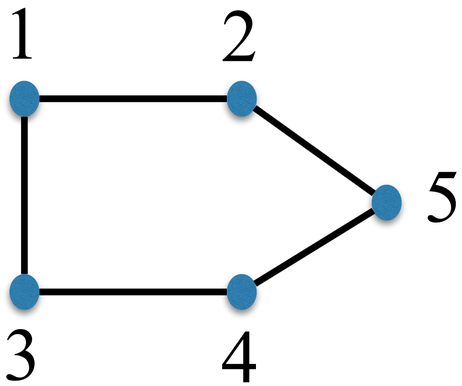
Bipartite Graphs

Given a graph $G = (V, E)$, we can ask, is it bipartite?



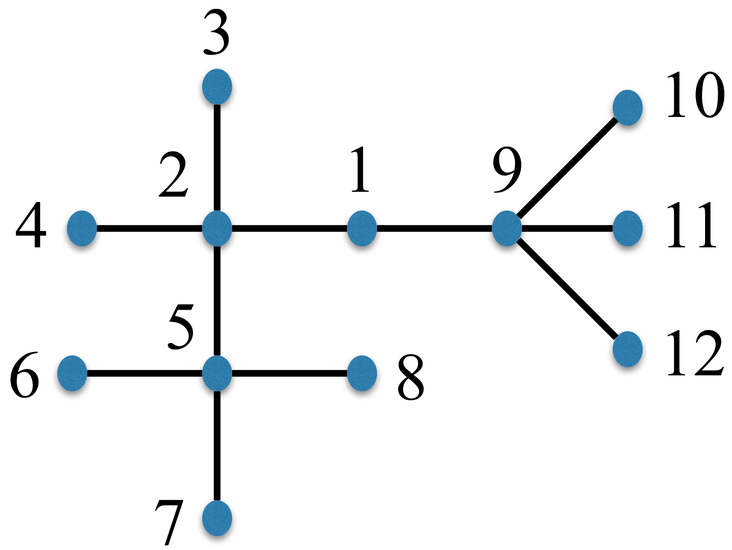




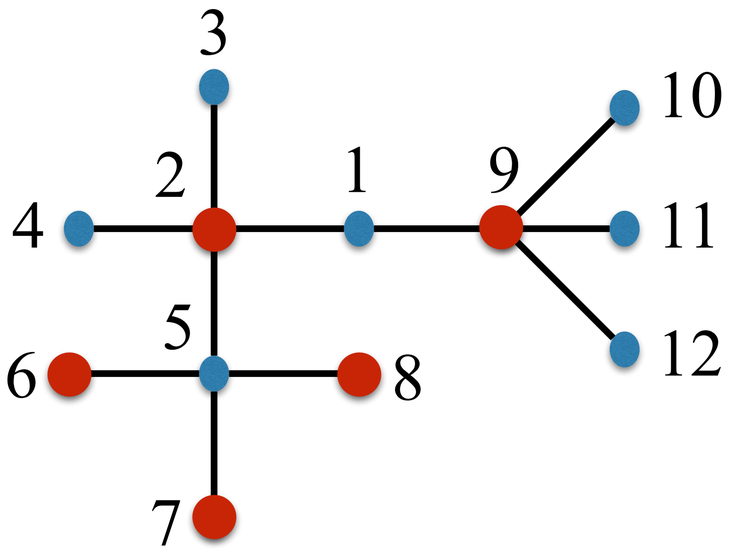


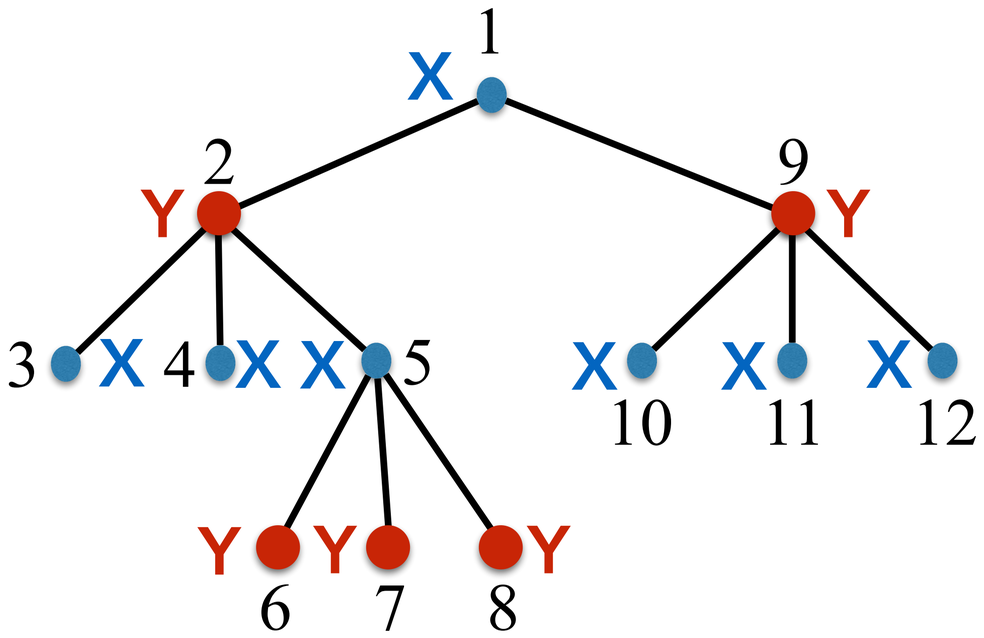
poll.cs251.com

Is this bipartite?



Poll Answer





bipartite = 2-colorable

k-colorable graph: Can color the vertices with **k colors** so that no edge's endpoints get the same color.

Characterization of Bipartite Graphs

An obstruction for being bipartite:

Contains a cycle of odd length.

Is this the only type of obstruction?

Theorem: A graph $G = (V, E)$ is bipartite
if and only if it contains no cycles of odd length.

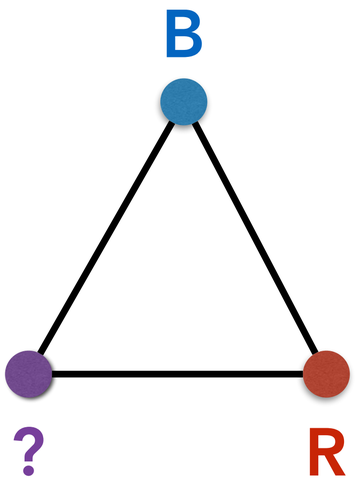
Proof

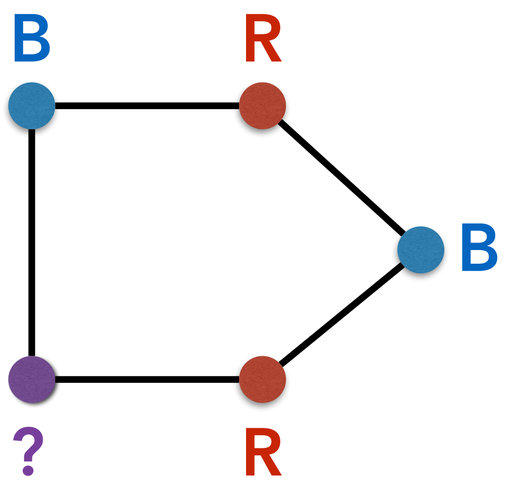
G is bipartite **iff** G contains no odd-length cycles.

Proof

G is 2-colorable **iff** G contains no odd-length cycles.

If G has an odd-length cycle, then it is not 2-colorable.





Proof

G is 2-colorable **iff** G contains no odd-length cycles.

If G has no odd-length cycle, then it is 2-colorable.

Strategy:

- Present a proper 2-coloring of the graph.
- Can assume the graph is connected.
  (we can 2-color each connected component separately)

Proof

G is 2-colorable **iff** G contains no odd-length cycles.

If G has no odd-length cycle, then it is 2-colorable.

Distance from root

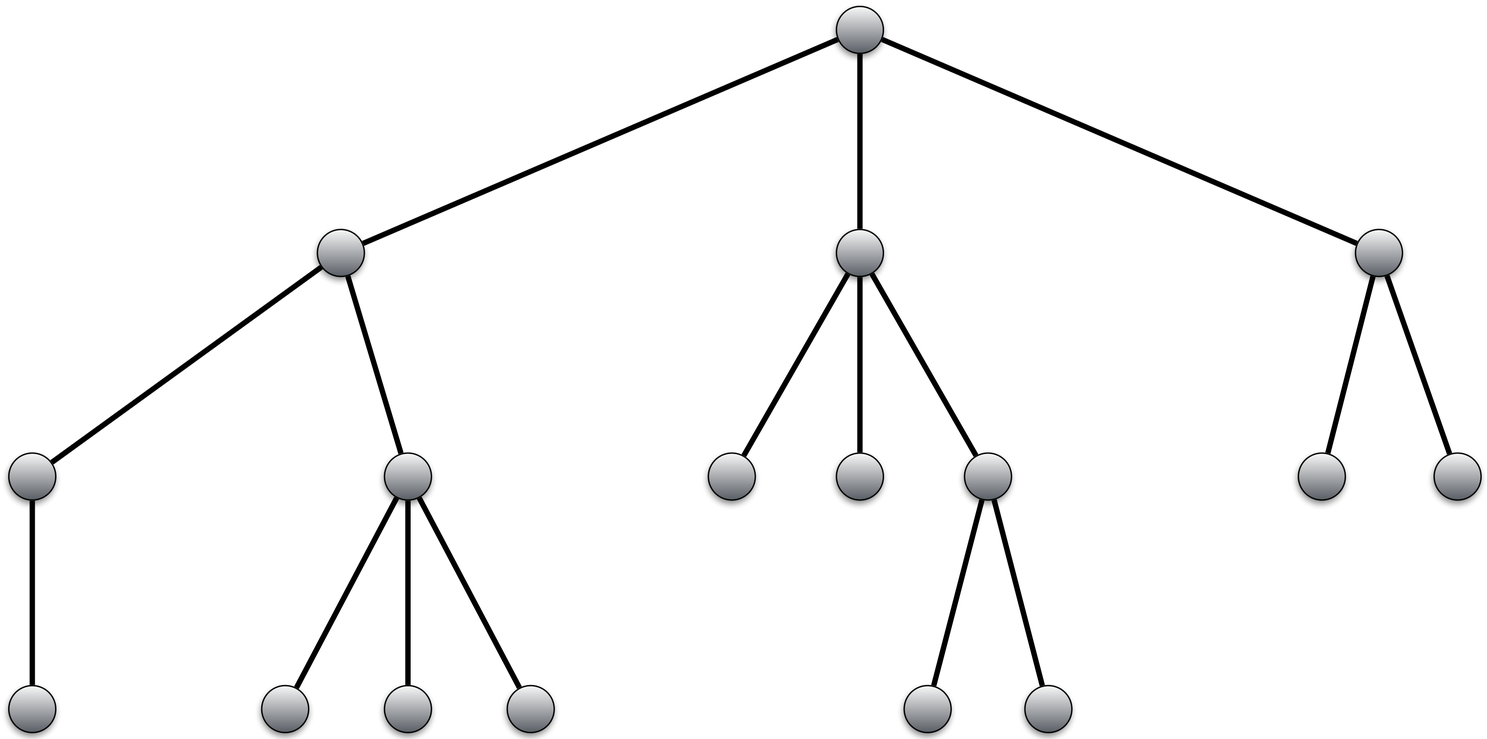
0

1

2

3

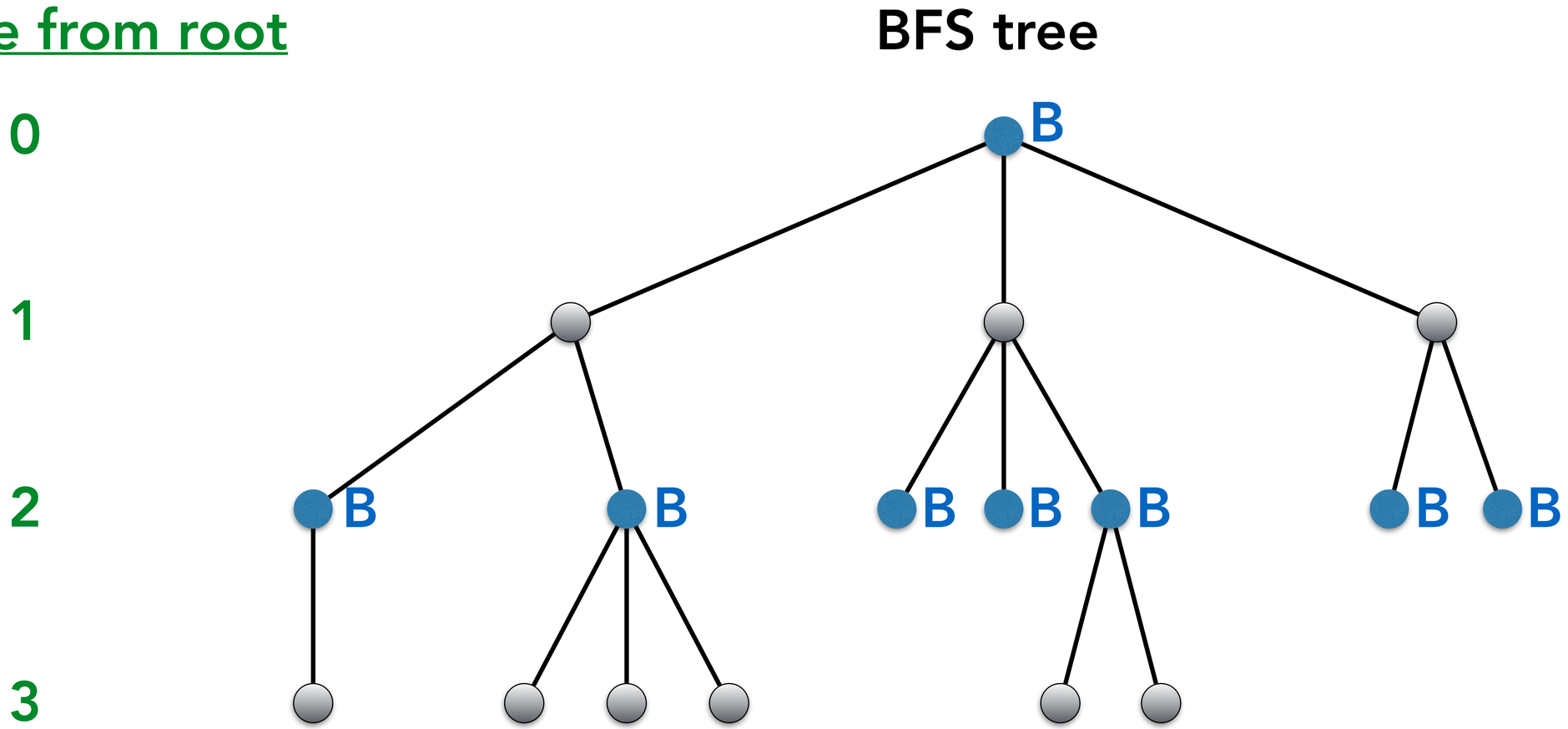
BFS tree



Proof

G is 2-colorable **iff** G contains no odd-length cycles.

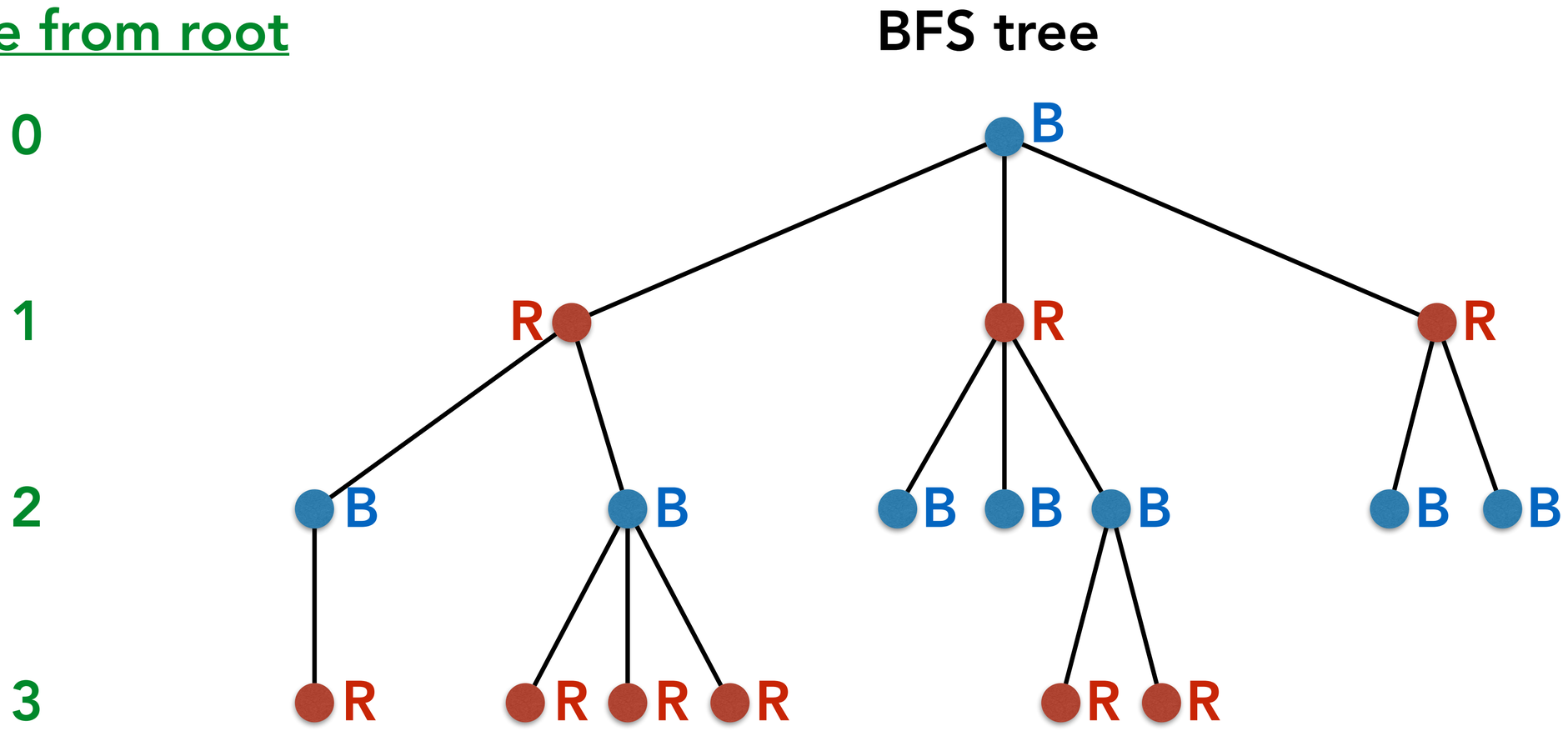
If G has no odd-length cycle, then it is 2-colorable.



Proof

G is 2-colorable **iff** G contains no odd-length cycles.

If G has no odd-length cycle, then it is 2-colorable.

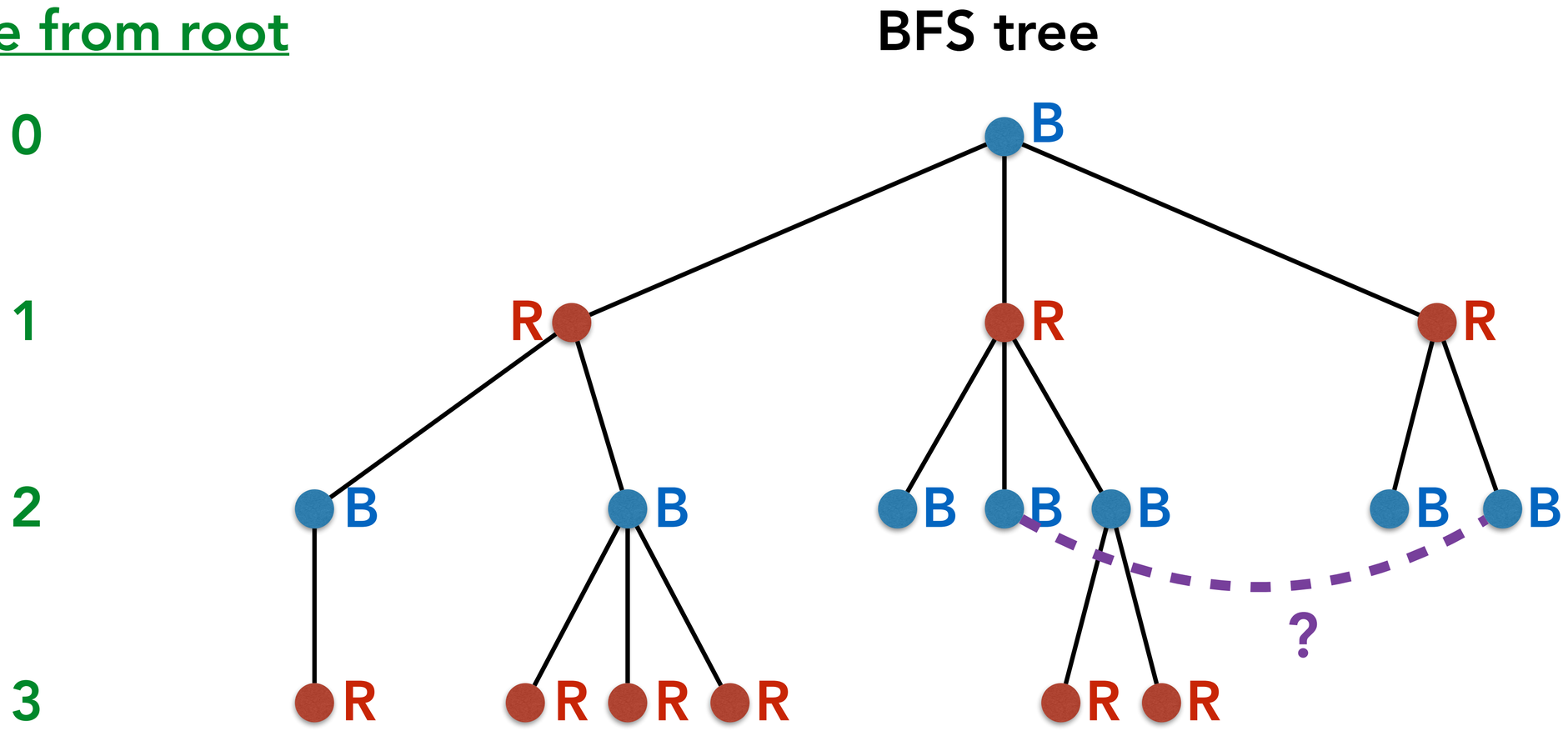


Can there be an edge whose endpoints are colored the same?

Proof

G is 2-colorable **iff** G contains no odd-length cycles.

If G has no odd-length cycle, then it is 2-colorable.

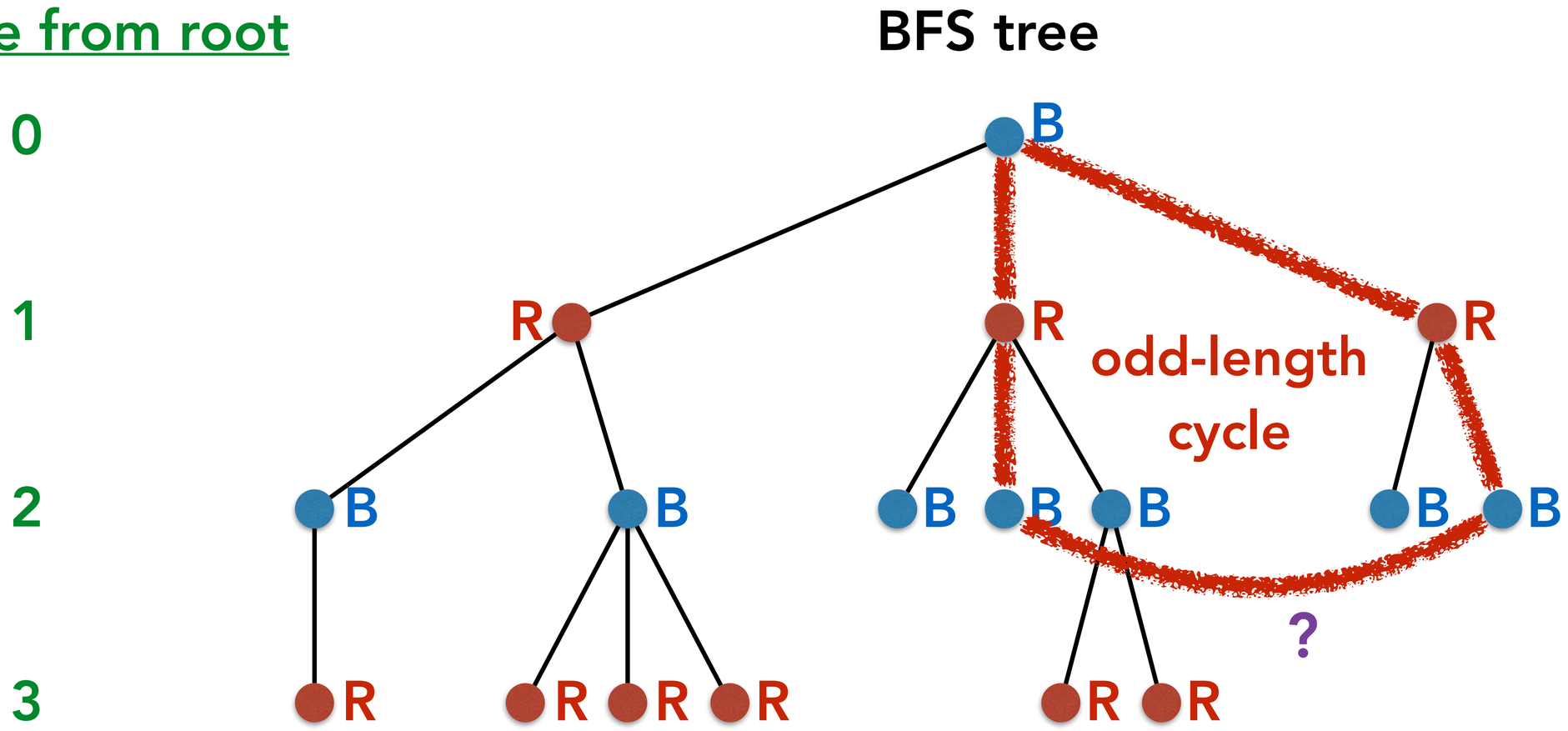


Can there be an edge whose endpoints are colored the same?

Proof

G is 2-colorable **iff** G contains no odd-length cycles.

If G has no odd-length cycle, then it is 2-colorable.

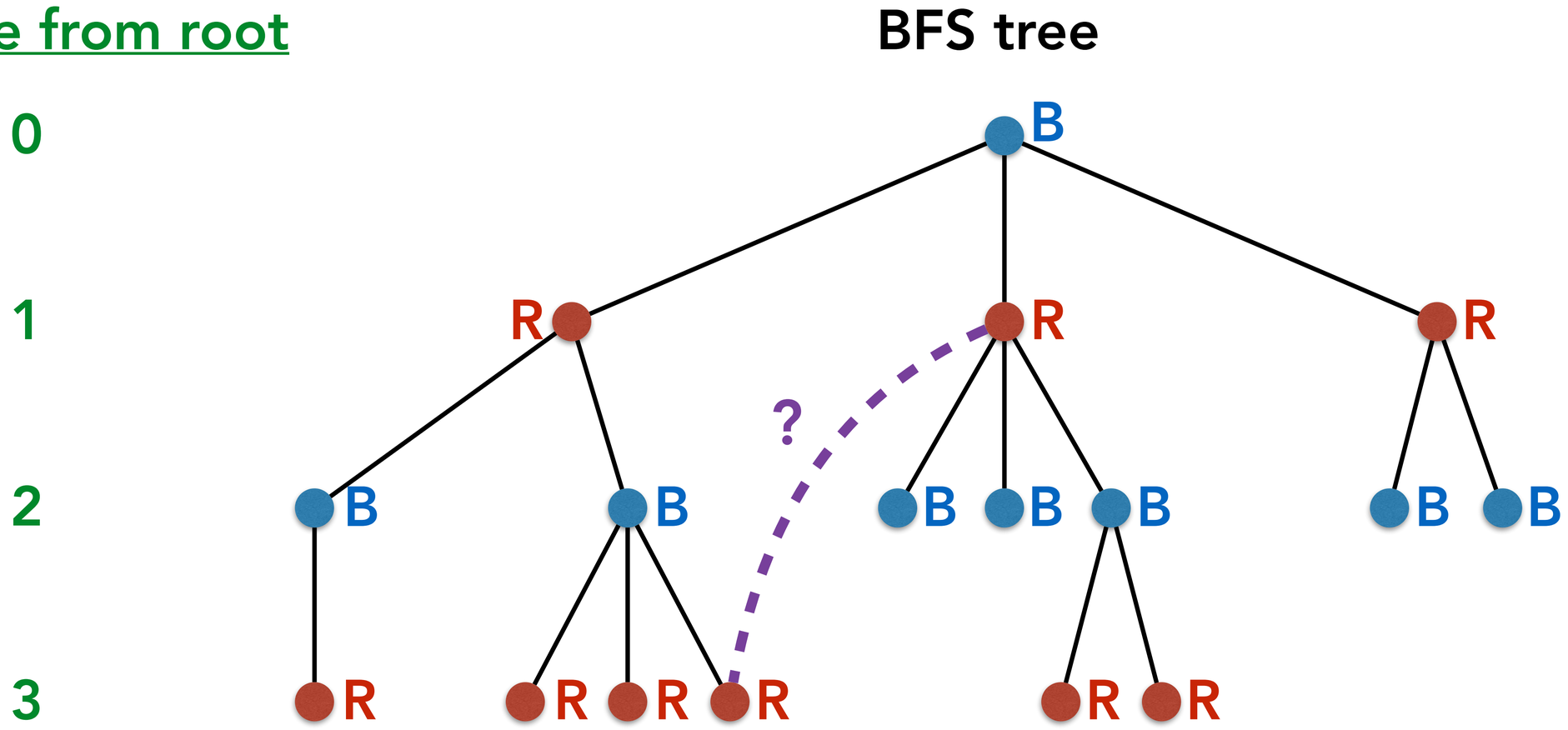


Can there be an edge whose endpoints are colored the same?

Proof

G is 2-colorable **iff** G contains no odd-length cycles.

If G has no odd-length cycle, then it is 2-colorable.

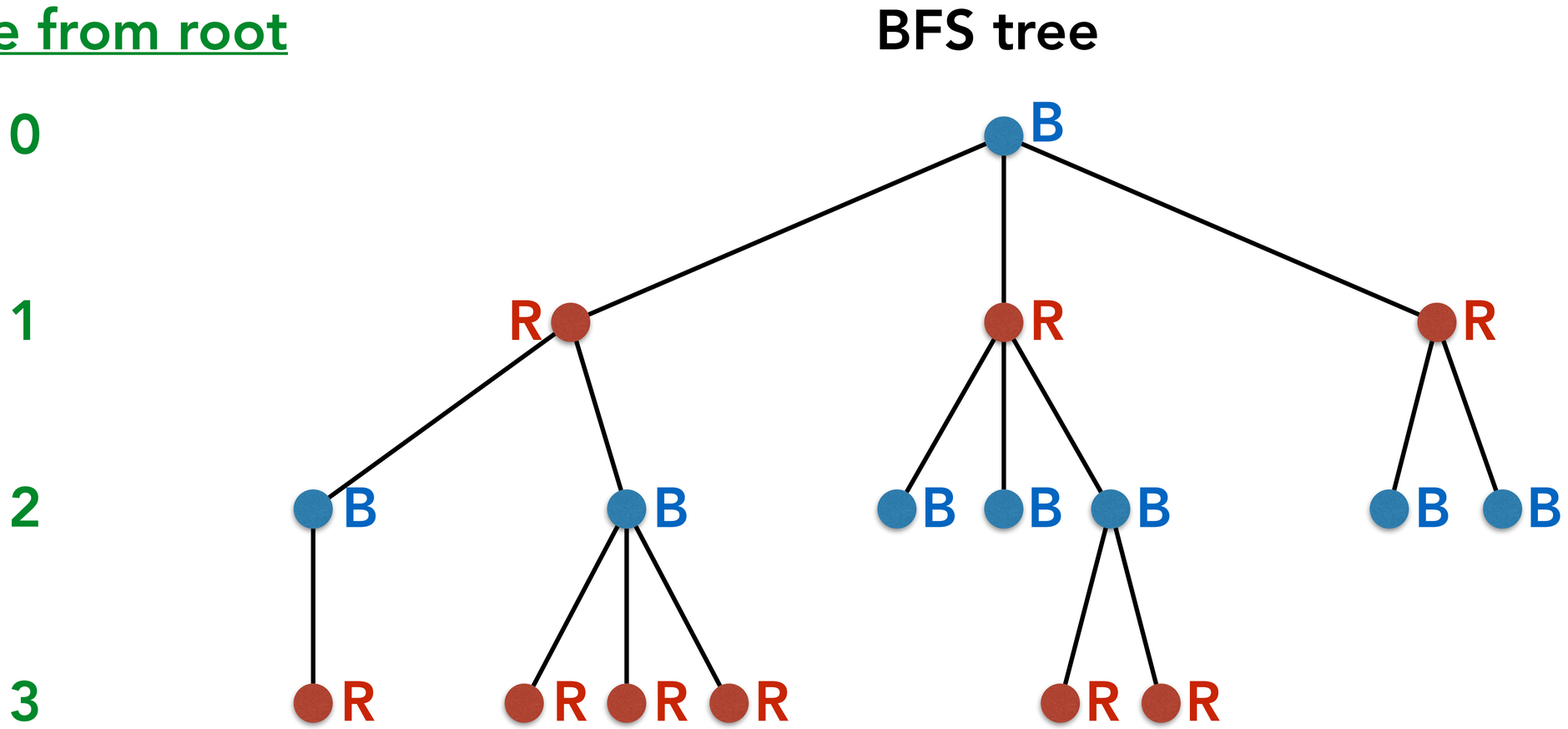


Can there be an edge whose endpoints are colored the same?

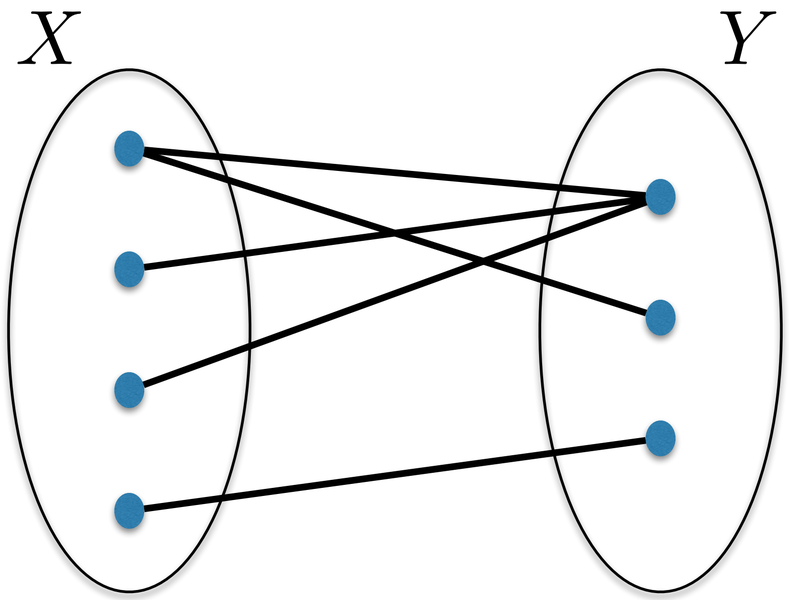
Proof

G is 2-colorable **iff** G contains no odd-length cycles.

If G has no odd-length cycle, then it is 2-colorable.



Bipartite Graphs



Often we write the bipartition explicitly:

$$G = (X, Y, E)$$

Bipartite Graphs

Great at modeling relations between two distinct classes of objects.

Examples:

X = **machines**, Y = **jobs**

An edge $\{x, y\}$ means x is capable of doing y.

X = **professors**, Y = **courses**

An edge $\{x, y\}$ means x can teach y.

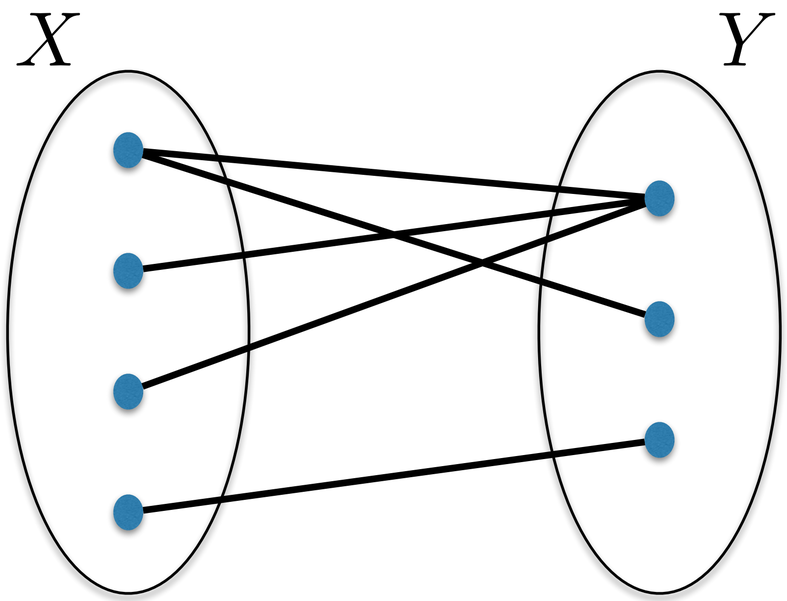
X = **students**, Y = **jobs**

An edge $\{x, y\}$ means x and y are interested in each other.

⋮

Matchings in graphs

Often, we are interested in finding a **matching** in a graph.

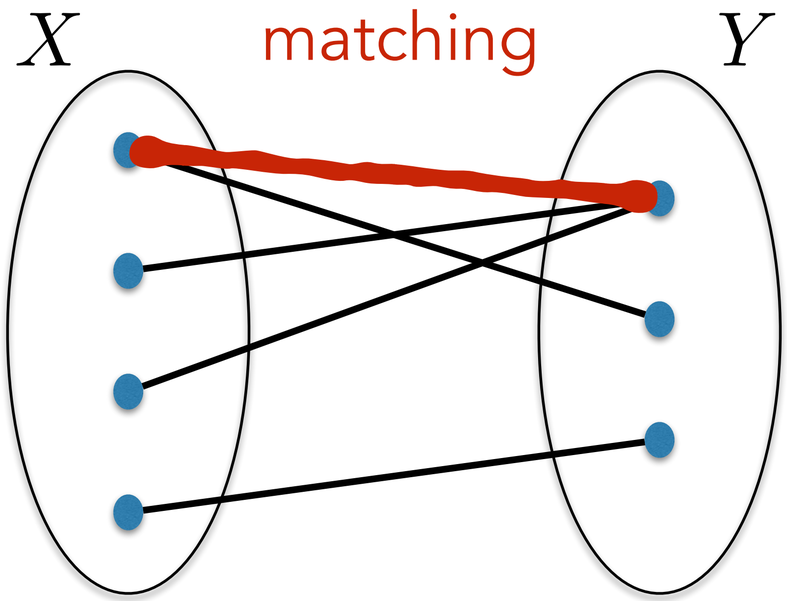


Matching:

A subset of the edges that do not share an endpoint.

Matchings in graphs

Often, we are interested in finding a **matching** in a graph.

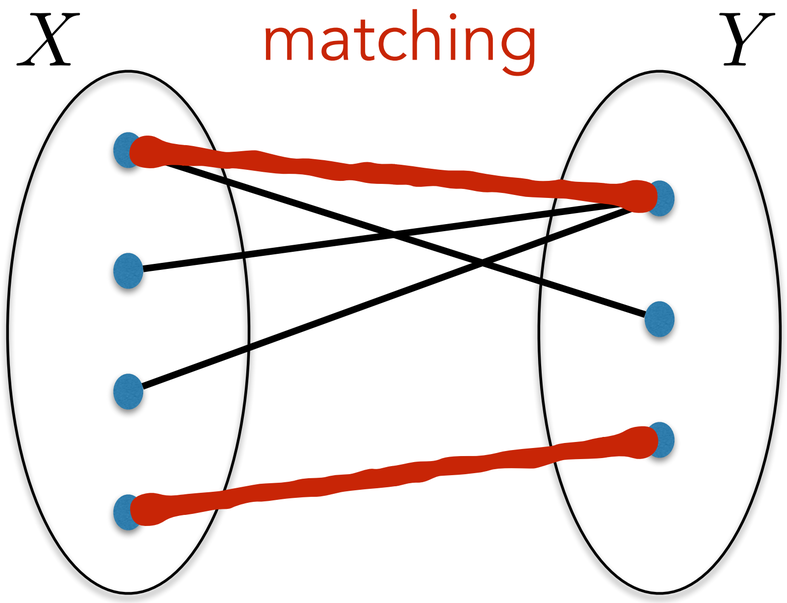


Matching:

A subset of the edges that do not share an endpoint.

Matchings in graphs

Often, we are interested in finding a **matching** in a graph.

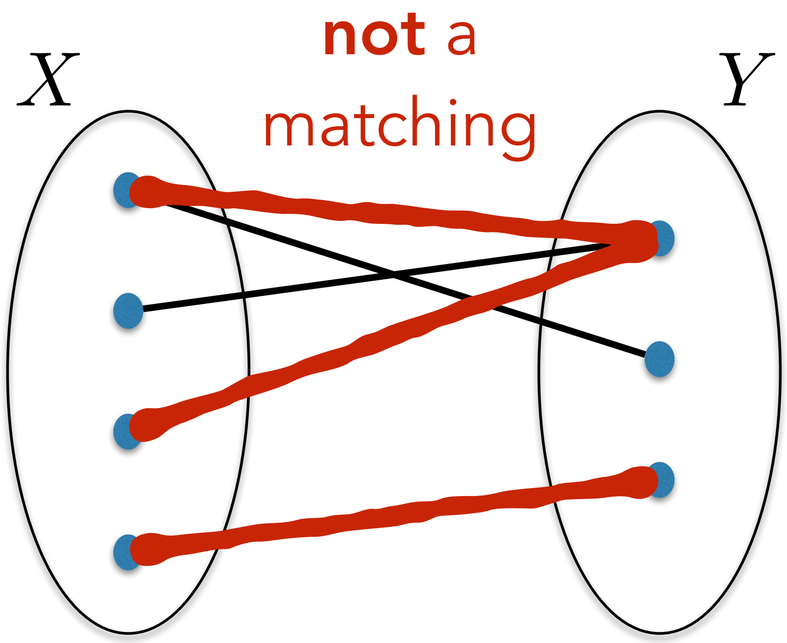


Matching:

A subset of the edges that do not share an endpoint.

Matchings in graphs

Often, we are interested in finding a **matching** in a graph.

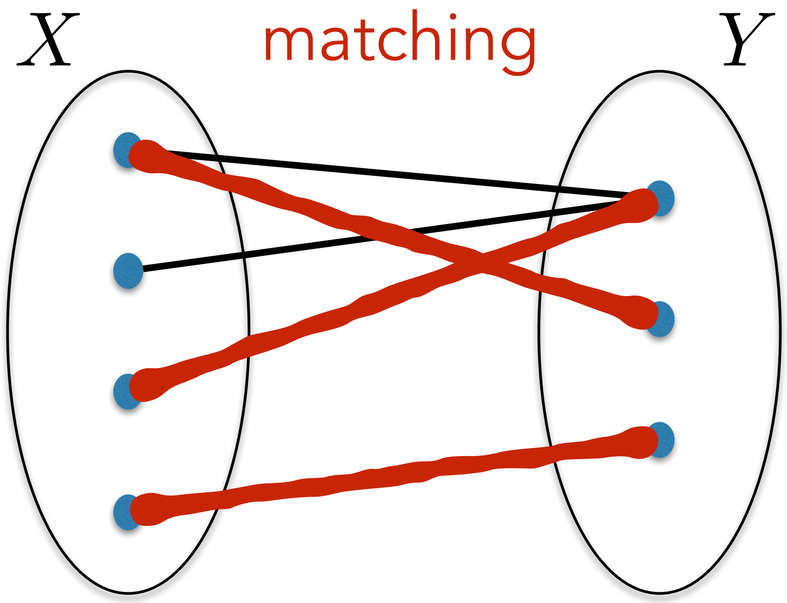


Matching:

A <u>subset of the edges</u> that do not share an endpoint.

Matchings in graphs

Often, we are interested in finding a **matching** in a graph.

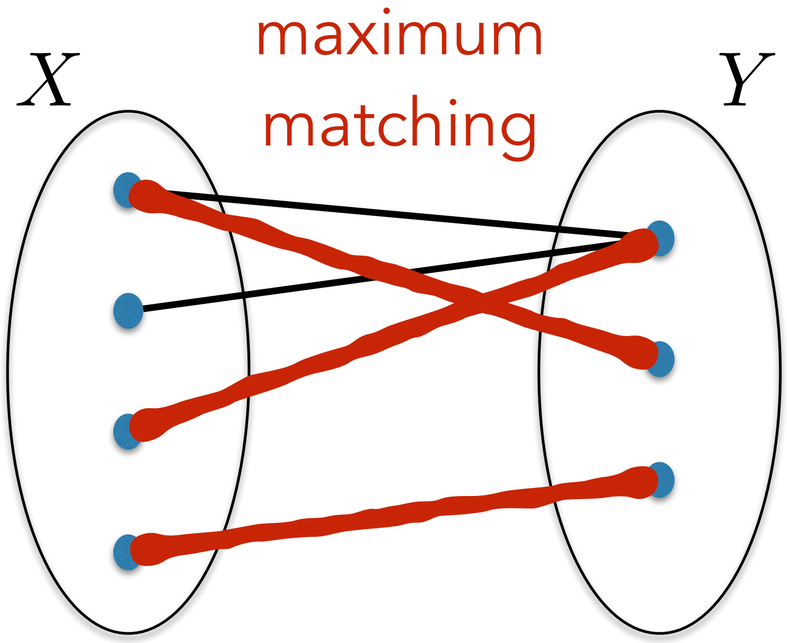


Matching:

A subset of the edges that do not share an endpoint.

Matchings in graphs

Often, we are interested in finding a **matching** in a graph.



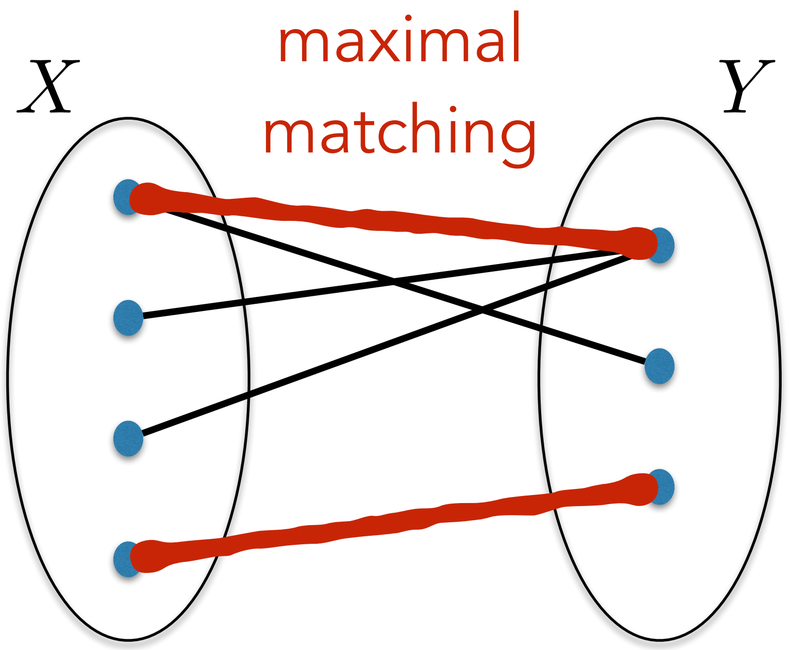
Maximum matching:

A matching with largest number of edges.

(among all possible matchings)

Matchings in graphs

Often, we are interested in finding a **matching** in a graph.



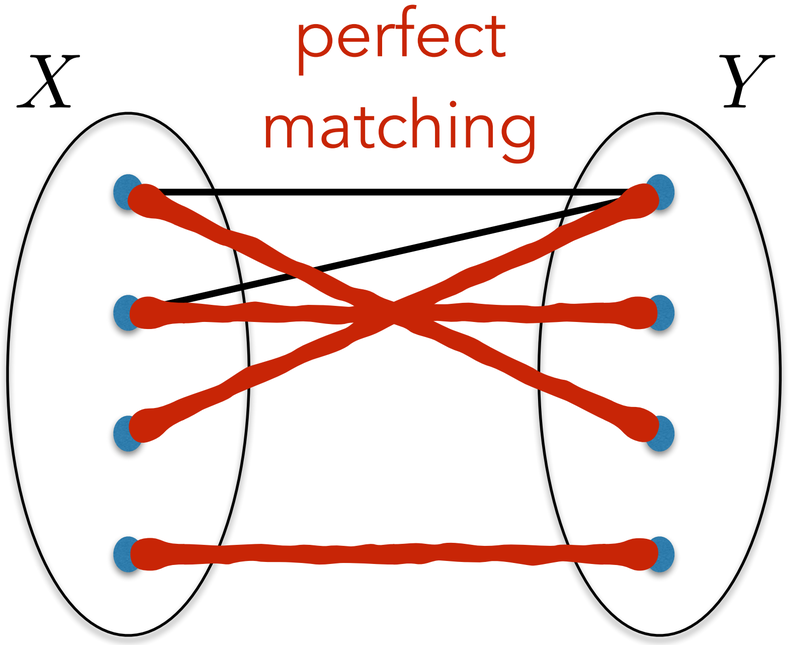
local optimum

Maximal matching:

A matching which cannot contain any more edges.

Matchings in graphs

Often, we are interested in finding a **matching** in a graph.



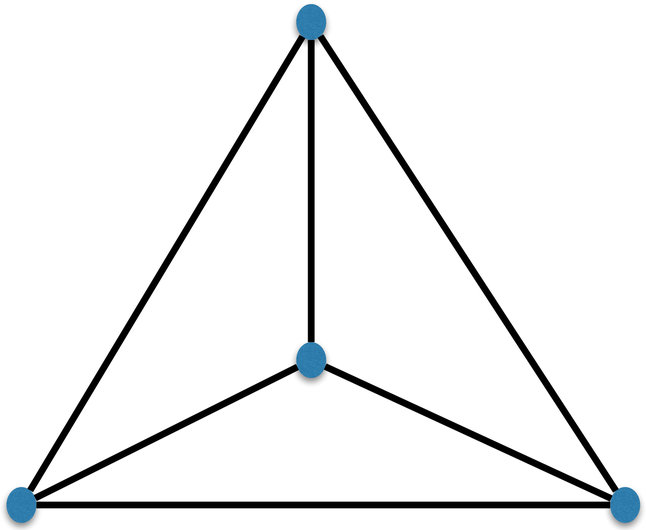
necessary condition for
having a perfect matching
in a bipartite graph:

$$|X| = |Y|$$

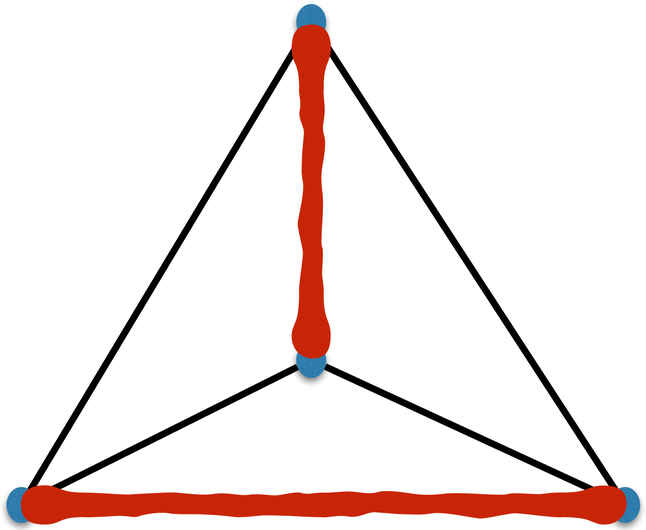
Perfect matching:

A matching that covers all vertices.

Matchings are defined for all graphs (not just bipartite).



Matchings are defined for all graphs (not just bipartite).



Maximum matching problem

The problem we want to solve is:

Maximum matching problem

Input: A graph $G = (V, E)$.

Output: A maximum matching in G.

Maximum matching problem

Actually, we want to solve the following restriction:

Bipartite maximum matching problem

Input: A bipartite graph $G = (X, Y, E)$.

Output: A maximum matching in G.

How do you solve a problem like this?

1. Formulate the problem 

2. **Ask**: Is there a trivial algorithm? Find and analyze.

3. **Ask**: Is there a better algorithm? Find and analyze.

Step 2: Is there a trivial algorithm?

Bipartite maximum matching problem

Input: A bipartite graph $G = (X, Y, E)$.

Output: A maximum matching in G.

Is there a (trivial) algorithm to solve this problem?

Try all possible subsets of the edges.

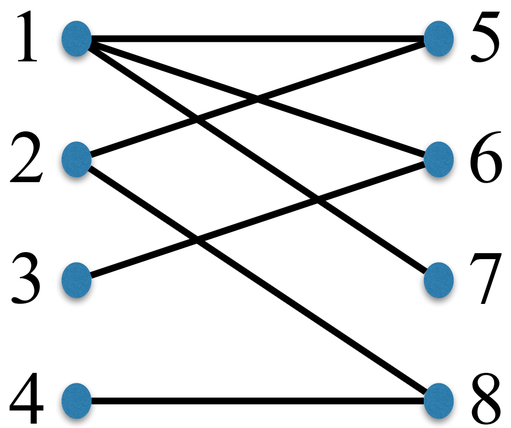
Running time: $\Omega(2^m)$

How do you solve a problem like this?

1. Formulate the problem 
2. **Ask**: Is there a trivial algorithm? Find and analyze. 
3. **Ask**: Is there a better algorithm? Find and analyze.

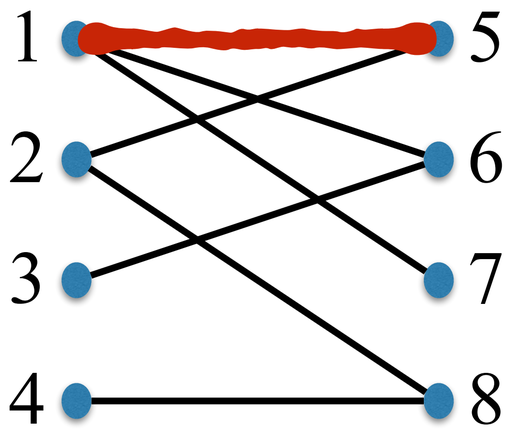
Trying to find a maximum matching

1st Attempt: What if we picked edges "greedily"?



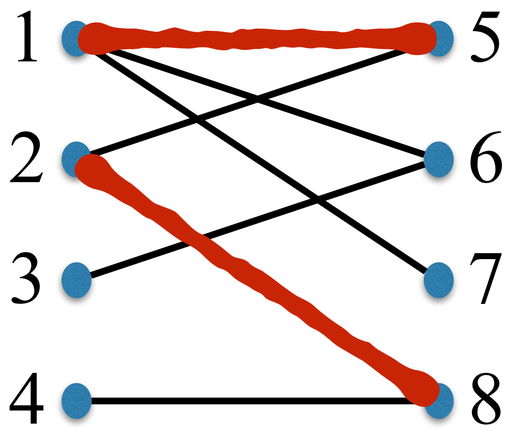
Trying to find a maximum matching

1st Attempt: What if we picked edges "greedily"?



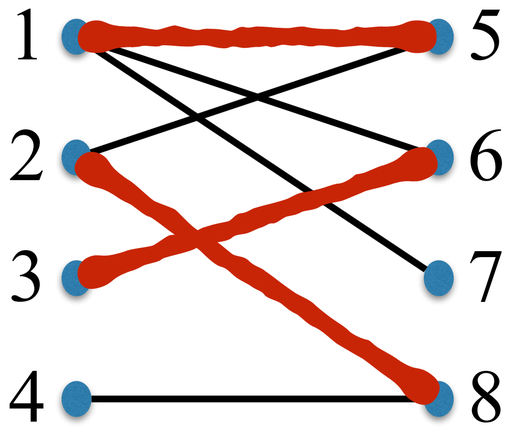
Trying to find a maximum matching

1st Attempt: What if we picked edges "greedily"?



Trying to find a maximum matching

1st Attempt: What if we picked edges "greedily"?



maximal matching

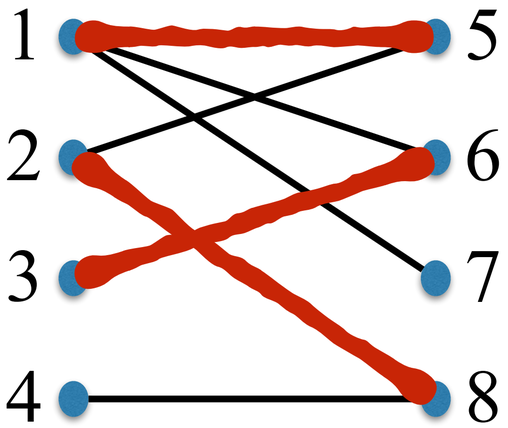
but **not** **maximum**

? Is there a way to get out of this *local optimum*?

What is interesting about the path 4 – 8 – 2 – 5 – 1 – 7 ?

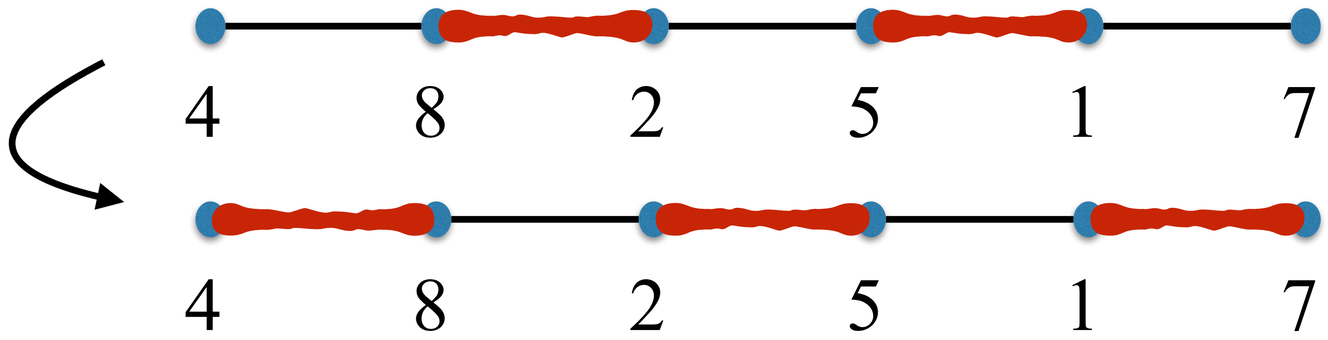
Trying to find a maximum matching

1st Attempt: What if we picked edges "greedily"?



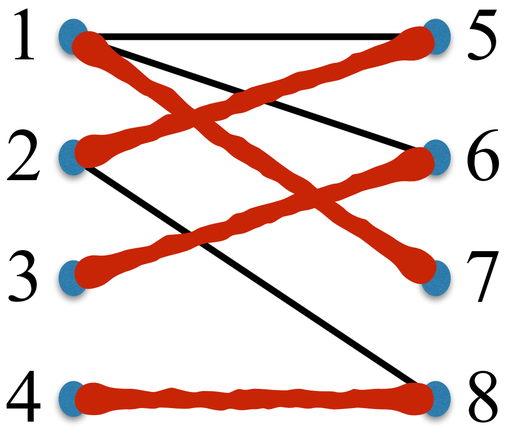
maximal matching

but **not maximum**

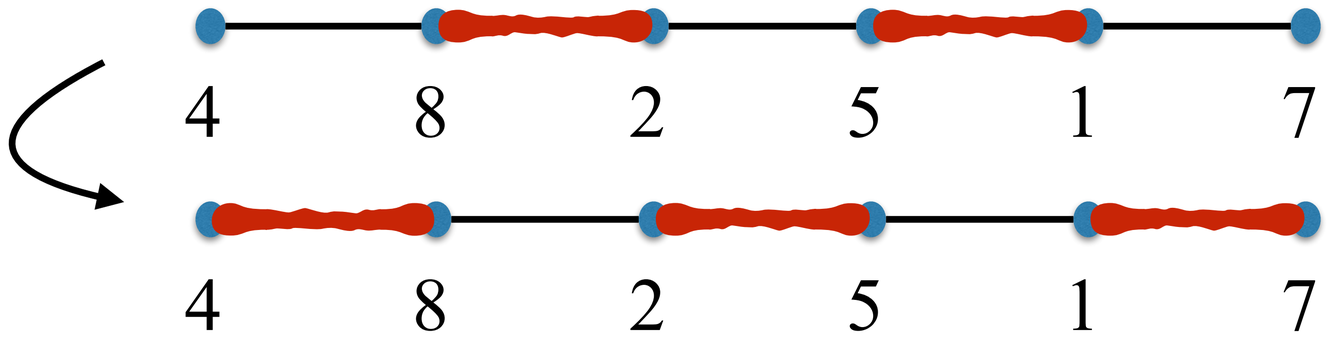


Trying to find a maximum matching

1st Attempt: What if we picked edges "greedily"?

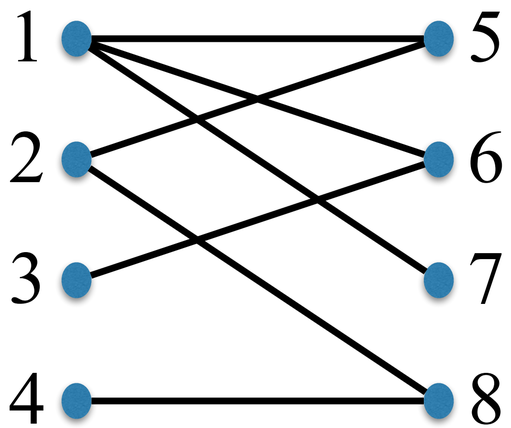


now **maximum**



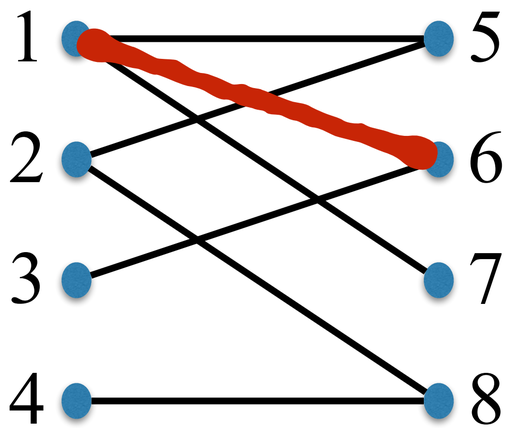
Trying to find a maximum matching

1st Attempt: What if we picked edges "greedily"?



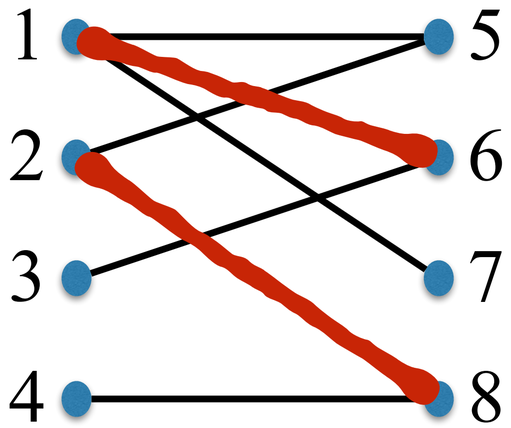
Trying to find a maximum matching

1st Attempt: What if we picked edges "greedily"?



Trying to find a maximum matching

1st Attempt: What if we picked edges "greedily"?



maximal matching

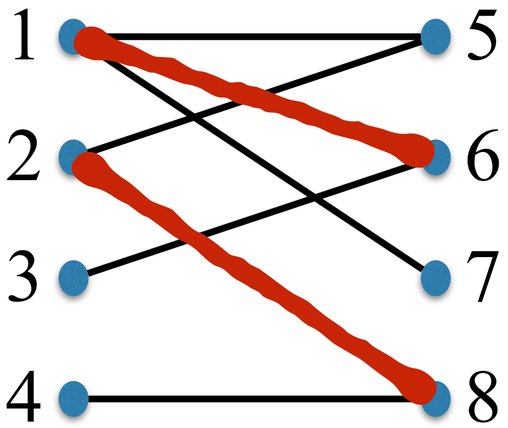
but **not** **maximum**

? Is there a way to get out of this *local optimum*?

What is interesting about the path $5 - 1 - 6 - 3$?

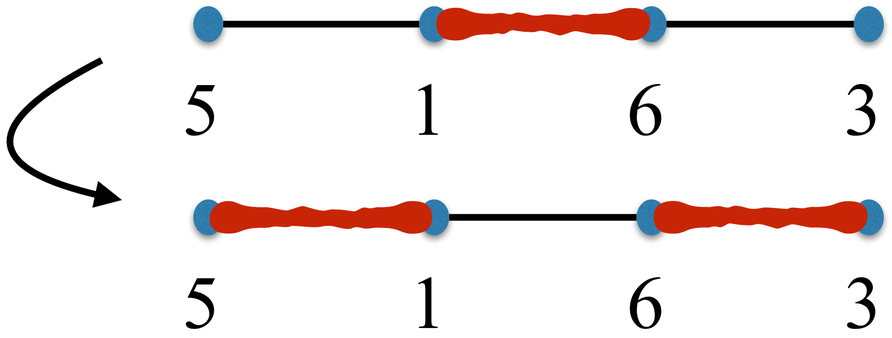
Trying to find a maximum matching

1st Attempt: What if we picked edges "greedily"?



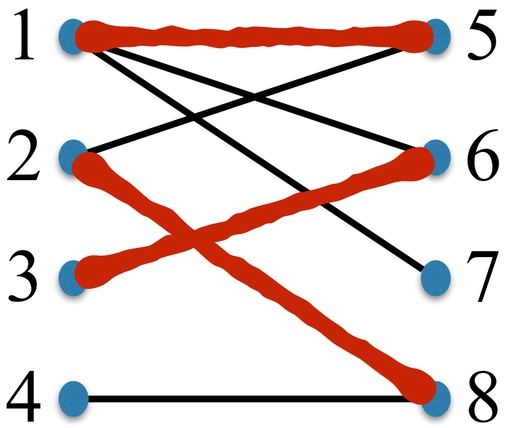
maximal matching

but **not** **maximum**



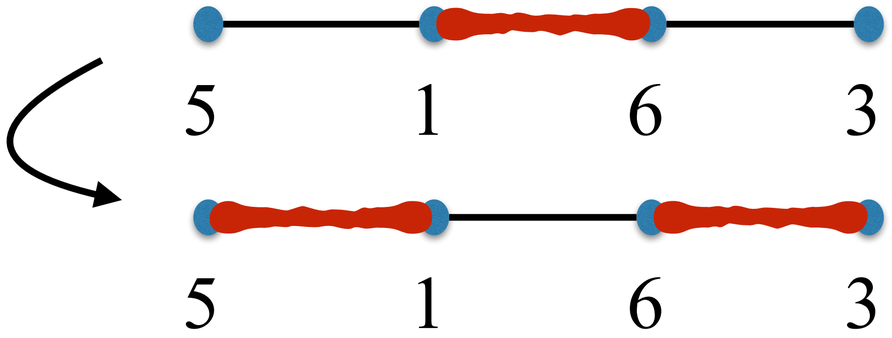
Trying to find a maximum matching

1st Attempt: What if we picked edges "greedily"?



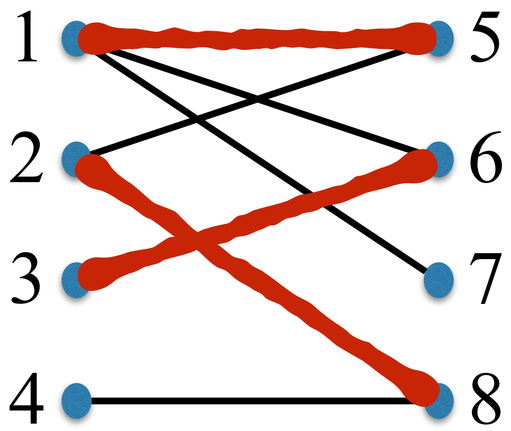
maximal matching

but **not** **maximum**



Trying to find a maximum matching

1st Attempt: What if we picked edges "greedily"?



maximal matching

but **not maximum**

Now same as previous example!

Do the same to find the maximum matching.

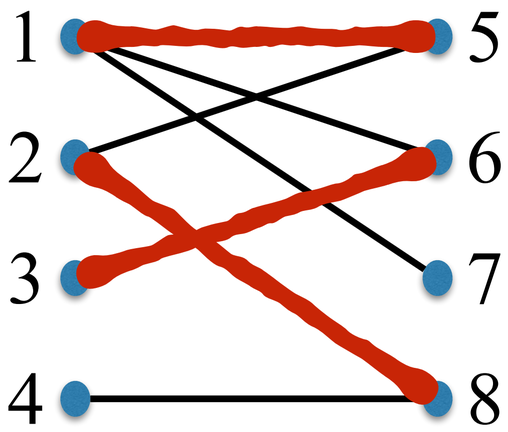
Augmenting path

Let **M** be some matching.

Definition: An **alternating path** with respect to **M** is a path such that edges in the path alternate between being in **M** and not being in **M**.

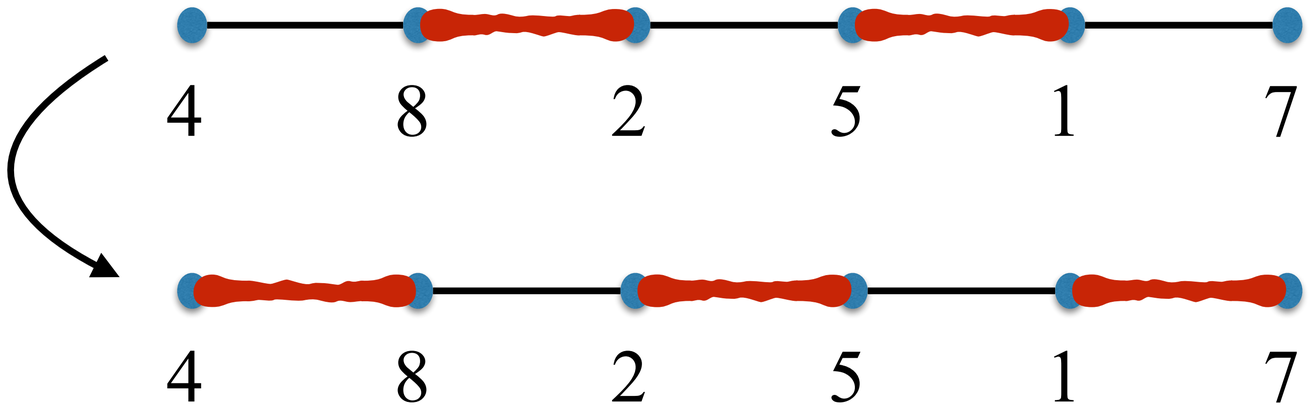
Definition: An **augmenting path** with respect to **M** is an **alternating path** such that the first and last vertices are **not** matched by **M**.

Augmenting path



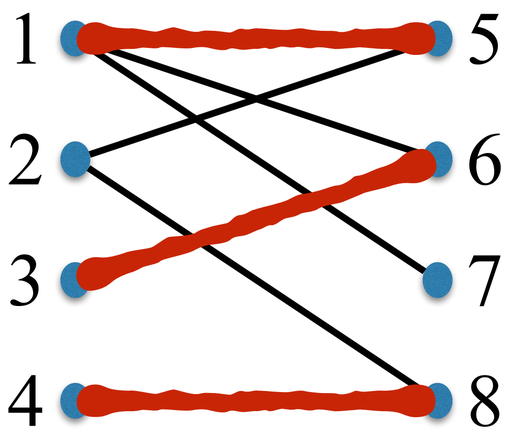
Augmenting path:

4 – 8 – 2 – 5 – 1 – 7



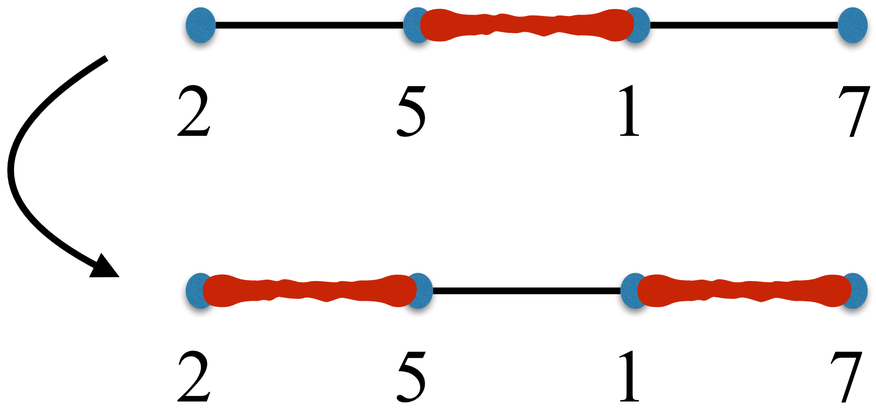
augmenting path $\Longrightarrow$ can obtain a bigger matching.

Augmenting path



Augmenting path:

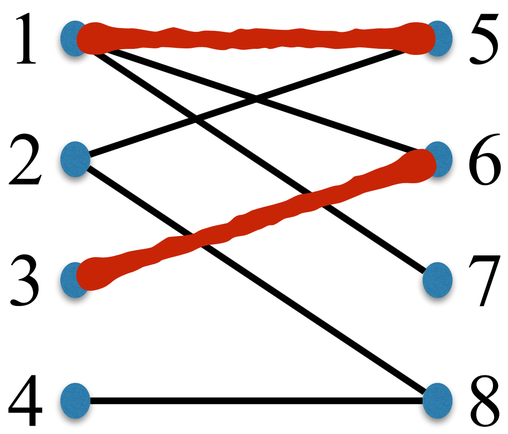
2 – 5 – 1 – 7



An augmenting path
need **not** contain
all the edges of the matching.

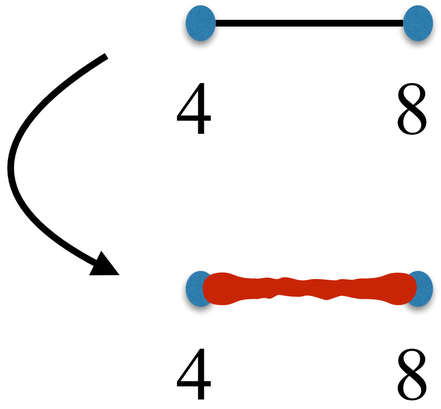
augmenting path $\Longrightarrow$ can obtain a bigger matching.

Augmenting path



Augmenting path:

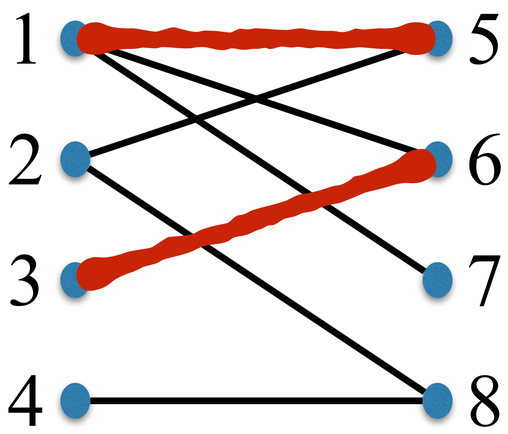
4 – 8



An augmenting path
need **not** contain
any of the edges in the matching.

augmenting path $\Longrightarrow$ can obtain a bigger matching.

Augmenting path



Not (!) an augmenting path:

2 – 5

5 is matched!

Characterization of Maximum Matchings

An obstruction for being a maximum matching:

There is an augmenting path.

Is this the only type of obstruction?

Theorem: A matching **M** is maximum **if and only if** there is **no** augmenting path with respect to **M**.

Algorithm to find maximum matching

Theorem: A matching **M** is maximum **if and only if** there is **no** augmenting path with respect to **M**.

Algorithm to find max matching:

- Start with a single edge as your matching **M**.
- Repeat until there is no augmenting path w.r.t. **M**:
  - Find an augmenting path with respect to **M**.
  - Update **M** according to the augmenting path.

? OK, but how do you find an augmenting path?

Proof of theorem

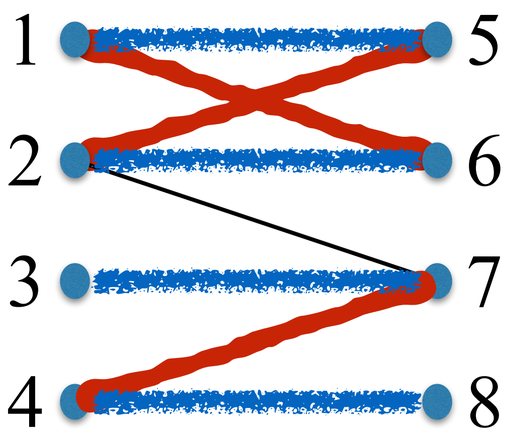
M is not maximum **iff** ∃ augmenting path w.r.t. **M**

If ∃ an augmenting path w.r.t. **M**,
can find a bigger matching (i.e. **M** is not maximum).

Goal:

If **M** is not maximum, find an augmenting path w.r.t. **M**.

Let **M*** be a matching such that |**M***| > |**M**|.



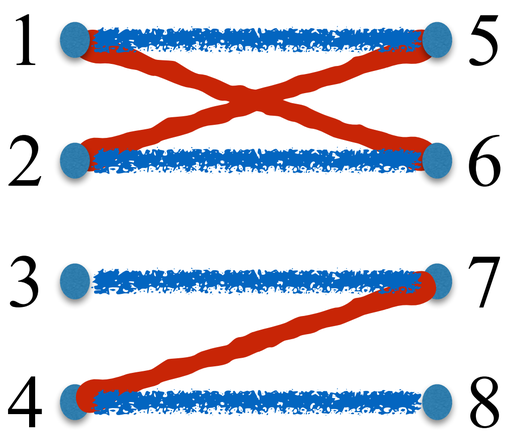
red = **M**

blue = **M***
(horizontal edges)

S = (**M*** ∪ **M**) - (**M** ∩ **M***)

(will find an **augmenting path** in **S**)

Proof of theorem



red = **M**

blue = **M***
(horizontal edges)

S = (**M*** ∪ **M**) - (**M** ∩ **M***)

(will find an **augmenting path** in **S**)

? What does **S** look like?

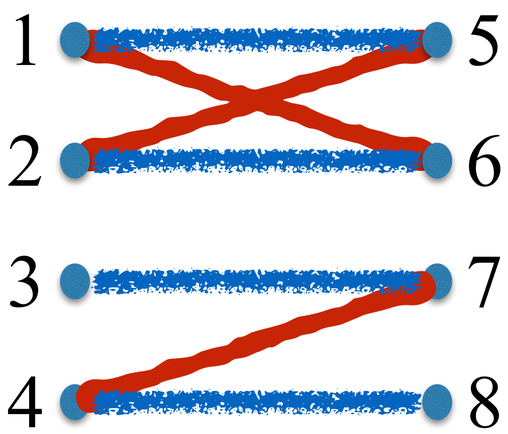
(i) Each vertex has degree 1 or 2. (why?)

(ii) So **S** is a collection of disjoint **cycles** and **paths**. (exercise)

(iii) The edges alternate **red** and **blue**.

(so cycles must have even length)

Proof of theorem



red = **M**

blue = **M***
(horizontal edges)

S = (**M*** ∪ **M**) - (**M** ∩ **M***)

(will find an **augmenting path** in **S**)

So **S** is a collection of disjoint **cycles** (even length) and **paths**.

The edges alternate **red** and **blue**.

blue > # red in **S**

blue = # red in **cycles**

So ∃ a **path** with # blue > # red.

This is an **augmenting path** with respect to **M**. ■

Theorem: A matching **M** is maximum **if and only if** there is **no** augmenting path with respect to **M**.

Summary of proof:

$\Longrightarrow$

If there is an augmenting path, not a max matching.

(can find a bigger matching)

$\Longleftarrow$

If **M** is not maximum, $\exists$ **M*** s.t. $|M^*| > |M|$.

Can find an augmenting path w.r.t. **M**

in the "symmetric difference" of **M*** and **M**.

Theorem: A matching **M** is maximum **if and only if** there is **no** augmenting path with respect to **M**.

Algorithm to find max matching:

- Start with a single edge as your matching **M**.
- Repeat until there is no augmenting path w.r.t. **M**:
    - Find an augmenting path with respect to **M**.
    - Update **M** according to the augmenting path.

How do you solve a problem like this?

1. Formulate the problem 
2. **Ask**: Is there a trivial algorithm? Find and analyze. 
3. **Ask**: Is there a better algorithm? Find and analyze. 

Next Time: Stable Matchings