CS70: Discrete Math and Probability

Fan Ye

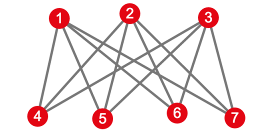
June 28, 2016

Planar non-planar

A finite graph is planar iff it does not contain a subgraph that is (a subdivision of) K_5 or $K_{3,3}$

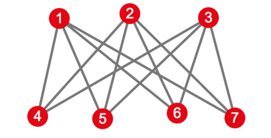
Planar non-planar

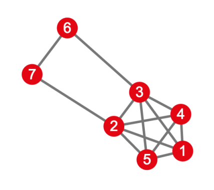
A finite graph is planar iff it does not contain a subgraph that is (a subdivision of) K_5 or $K_{3,3}$



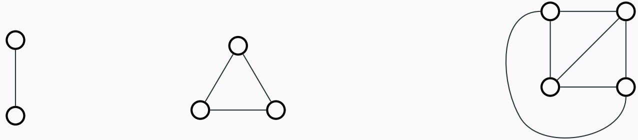
Planar non-planar

A finite graph is planar iff it does not contain a subgraph that is (a subdivision of) K_5 or $K_{3,3}$



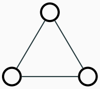


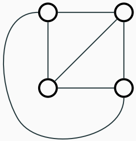
Complete Graph.



K_n complete graph on n vertices.

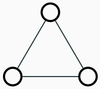
Complete Graph.

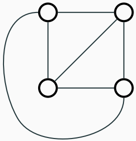




K_n complete graph on n vertices.
All edges are present.

Complete Graph.



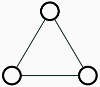
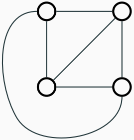


K_n complete graph on n vertices.

All edges are present.

Everyone is my neighbor.

Complete Graph.

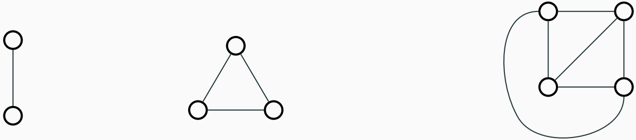
K_n complete graph on n vertices.

All edges are present.

Everyone is my neighbor.

Each vertex is adjacent to every other vertex.

Complete Graph.



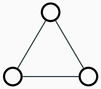
K_n complete graph on n vertices.

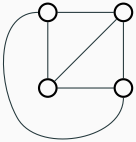
All edges are present.

Everyone is my neighbor.

Each vertex is adjacent to every other vertex.

Complete Graph.





K_n complete graph on n vertices.

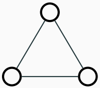
All edges are present.

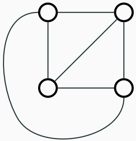
Everyone is my neighbor.

Each vertex is adjacent to every other vertex.

How many edges?

Complete Graph.





K_n complete graph on n vertices.

All edges are present.

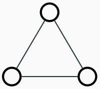
Everyone is my neighbor.

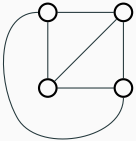
Each vertex is adjacent to every other vertex.

How many edges?

Each vertex is incident to $n-1$ edges.

Complete Graph.





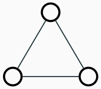
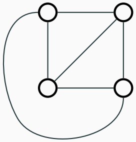
K_n complete graph on n vertices.

- All edges are present.
- Everyone is my neighbor.
- Each vertex is adjacent to every other vertex.

How many edges?

- Each vertex is incident to $n-1$ edges.
- Sum of degrees is $n(n-1)$.

Complete Graph.

K_n complete graph on n vertices.

All edges are present.

Everyone is my neighbor.

Each vertex is adjacent to every other vertex.

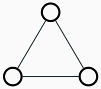
How many edges?

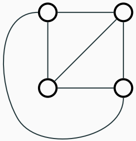
Each vertex is incident to $n-1$ edges.

Sum of degrees is $n(n-1)$.

$\implies$ Number of edges is $n(n-1)/2$.

Complete Graph.





K_n complete graph on n vertices.

All edges are present.

Everyone is my neighbor.

Each vertex is adjacent to every other vertex.

How many edges?

Each vertex is incident to $n-1$ edges.

Sum of degrees is $n(n-1)$.

$\implies$ Number of edges is $n(n-1)/2$.

Remember sum of degree is $2|E|$.

Trees.

Definitions:

Trees.

Definitions:

A connected graph without a cycle.

Trees.

Definitions:

- A connected graph without a cycle.
- A connected graph with $|V| - 1$ edges.

Trees.

Definitions:

- A connected graph without a cycle.
- A connected graph with $|V|-1$ edges.
- A connected graph where any edge removal disconnects it.

Trees.

Definitions:

- A connected graph without a cycle.
- A connected graph with $|V| - 1$ edges.
- A connected graph where any edge removal disconnects it.
- A connected graph where any edge addition creates a cycle.

Trees.

Definitions:

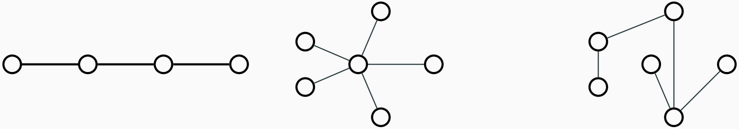
- A connected graph without a cycle.
- A connected graph with $|V| - 1$ edges.
- A connected graph where any edge removal disconnects it.
- A connected graph where any edge addition creates a cycle.

Trees.

Definitions:

A connected graph without a cycle.
A connected graph with $|V|-1$ edges.
A connected graph where any edge removal disconnects it.
A connected graph where any edge addition creates a cycle.

Some trees.



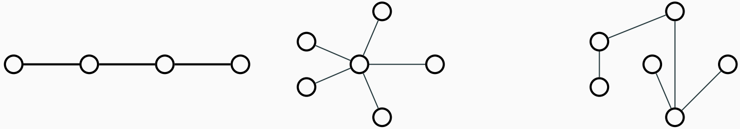
no cycle and connected?

Trees.

Definitions:

A connected graph without a cycle.
A connected graph with $|V|-1$ edges.
A connected graph where any edge removal disconnects it.
A connected graph where any edge addition creates a cycle.

Some trees.



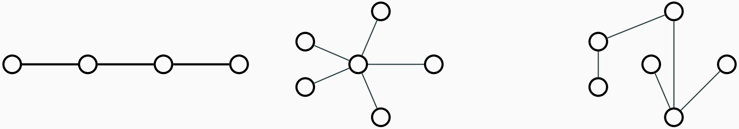
no cycle and connected? Yes.

Trees.

Definitions:

A connected graph without a cycle.
A connected graph with $|V|-1$ edges.
A connected graph where any edge removal disconnects it.
A connected graph where any edge addition creates a cycle.

Some trees.



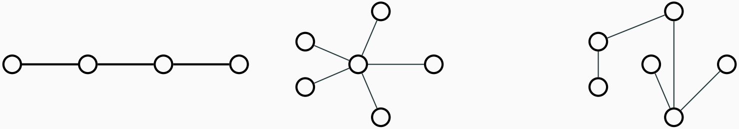
no cycle and connected? Yes.
$|V|-1$ edges and connected?

Trees.

Definitions:

A connected graph without a cycle.
A connected graph with $|V|-1$ edges.
A connected graph where any edge removal disconnects it.
A connected graph where any edge addition creates a cycle.

Some trees.



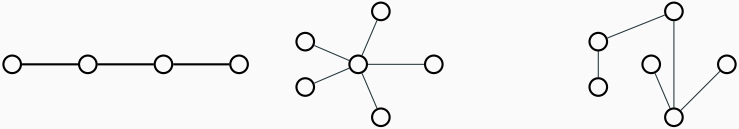
no cycle and connected? Yes.
$|V|-1$ edges and connected? Yes.

Trees.

Definitions:

- A connected graph without a cycle.
- A connected graph with $|V|-1$ edges.
- A connected graph where any edge removal disconnects it.
- A connected graph where any edge addition creates a cycle.

Some trees.



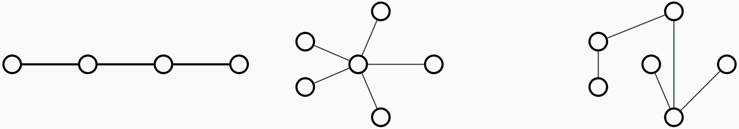
no cycle and connected? Yes.
$|V|-1$ edges and connected? Yes.
removing any edge disconnects it.

Trees.

Definitions:

- A connected graph without a cycle.
- A connected graph with $|V|-1$ edges.
- A connected graph where any edge removal disconnects it.
- A connected graph where any edge addition creates a cycle.

Some trees.



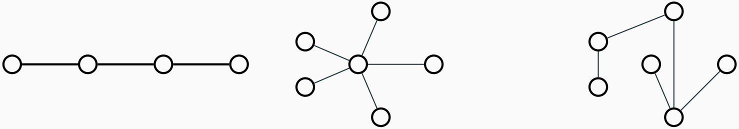
no cycle and connected? Yes.
$|V|-1$ edges and connected? Yes.
removing any edge disconnects it. Harder to check.

Trees.

Definitions:

A connected graph without a cycle.
A connected graph with $|V|-1$ edges.
A connected graph where any edge removal disconnects it.
A connected graph where any edge addition creates a cycle.

Some trees.



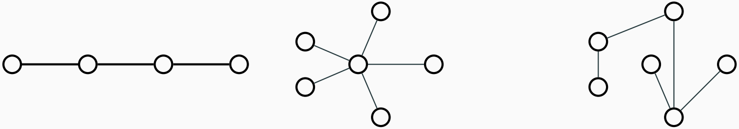
no cycle and connected? Yes.
$|V|-1$ edges and connected? Yes.
removing any edge disconnects it. Harder to check. but yes.

Trees.

Definitions:

- A connected graph without a cycle.
- A connected graph with $|V|-1$ edges.
- A connected graph where any edge removal disconnects it.
- A connected graph where any edge addition creates a cycle.

Some trees.



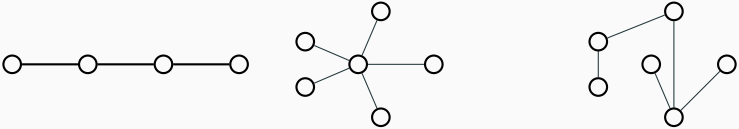
no cycle and connected? Yes.
$|V|-1$ edges and connected? Yes.
removing any edge disconnects it. Harder to check. but yes.
Adding any edge creates cycle.

Trees.

Definitions:

A connected graph without a cycle.
A connected graph with $|V|-1$ edges.
A connected graph where any edge removal disconnects it.
A connected graph where any edge addition creates a cycle.

Some trees.



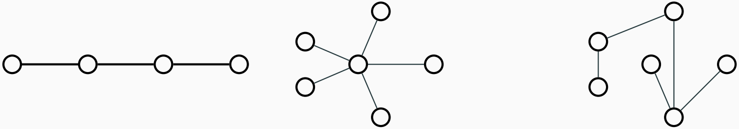
no cycle and connected? Yes.
$|V|-1$ edges and connected? Yes.
removing any edge disconnects it. Harder to check. but yes.
Adding any edge creates cycle. Harder to check.

Trees.

Definitions:

- A connected graph without a cycle.
- A connected graph with $|V|-1$ edges.
- A connected graph where any edge removal disconnects it.
- A connected graph where any edge addition creates a cycle.

Some trees.



no cycle and connected? Yes.
$|V|-1$ edges and connected? Yes.
removing any edge disconnects it. Harder to check. but yes.
Adding any edge creates cycle. Harder to check. but yes.

Equivalence of Definitions.

Theorem:
"G connected and has $|V| - 1$ edges" $\equiv$
"G is connected and has no cycles."

Equivalence of Definitions.

Theorem:
"G connected and has $|V|-1$ edges" $\equiv$
"G is connected and has no cycles."

Lemma: If v is a degree 1 in connected graph G, $G-v$ is connected.

Proof:

For $x \neq v, y \neq v \in V$,

Equivalence of Definitions.

Theorem:
"G connected and has $|V|-1$ edges" $\equiv$
"G is connected and has no cycles."

Lemma: If v is a degree 1 in connected graph G, $G-v$ is connected.

Proof:

For $x \neq v, y \neq v \in V$,
there is path between x and y in G since connected.

Equivalence of Definitions.

Theorem:
"G connected and has $|V|-1$ edges" $\equiv$
"G is connected and has no cycles."

Lemma: If v is a degree 1 in connected graph G, $G-v$ is connected.

Proof:

For $x \neq v, y \neq v \in V$,
there is path between x and y in G since connected.
and does not use v (degree 1)

Equivalence of Definitions.

Theorem:
"G connected and has $|V|-1$ edges" $\equiv$
"G is connected and has no cycles."

Lemma: If v is a degree 1 in connected graph G, $G-v$ is connected.

Proof:

For $x \neq v, y \neq v \in V$,
there is path between x and y in G since connected.
and does not use v (degree 1)
$\implies G-v$ is connected.

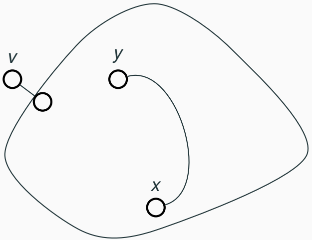
Equivalence of Definitions.

Theorem:
"G connected and has $|V|-1$ edges" $\equiv$
"G is connected and has no cycles."

Lemma: If v is a degree 1 in connected graph G, $G-v$ is connected.

Proof:

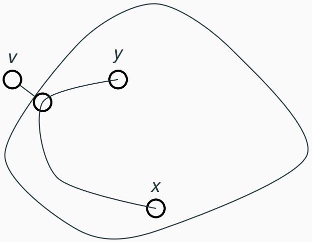
For $x \neq v, y \neq v \in V$,
there is path between x and y in G since connected.
and does not use v (degree 1)
$\implies$ $G-v$ is connected. $\square$



Equivalence of Definitions.

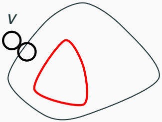
Theorem:
"G connected and has $|V|-1$ edges" $\equiv$
"G is connected and has no cycles."

Lemma: If v is a degree 1 in connected graph G, $G-v$ is connected.
Proof:
For $x \neq v, y \neq v \in V$,
there is path between x and y in G since connected.
and does not use v (degree 1)
$\implies G-v$ is connected. $\square$



Proof of only if.

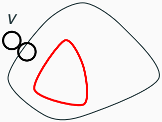
Thm:
"G connected and has $|V| - 1$ edges" $\equiv$
"G is connected and has no cycles."



Proof of $\implies$:

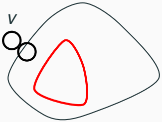
Proof of only if.

Thm:
"G connected and has $|V|-1$ edges" $\equiv$
"G is connected and has no cycles."



Proof of $\implies$: By induction on $|V|$.

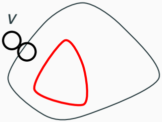
Proof of only if.



Thm:
"G connected and has $|V|-1$ edges" $\equiv$
"G is connected and has no cycles."

Proof of $\implies$: By induction on $|V|$.
Base Case: $|V|=1$. $0=|V|-1$ edges and has no cycles.

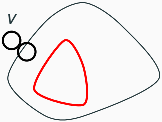
Proof of only if.



Thm:
"G connected and has $|V|-1$ edges" $\equiv$
"G is connected and has no cycles."

Proof of $\implies$: By induction on $|V|$.
Base Case: $|V|=1$. $0=|V|-1$ edges and has no cycles.

Proof of only if.

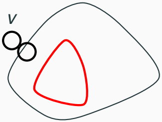


Thm:
"G connected and has $|V|-1$ edges" $\equiv$
"G is connected and has no cycles."

Proof of $\implies$: By induction on $|V|$.
Base Case: $|V| = 1$. $0 = |V| - 1$ edges and has no cycles.

Induction Step:

Proof of only if.

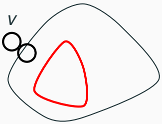


Thm:
"G connected and has $|V|-1$ edges" $\equiv$
"G is connected and has no cycles."

Proof of $\implies$: By induction on $|V|$.
Base Case: $|V|=1$. $0=|V|-1$ edges and has no cycles.

Induction Step:
Claim: There is a degree 1 node.

Proof of only if.

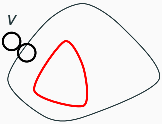


Thm:
"G connected and has $|V|-1$ edges" $\equiv$
"G is connected and has no cycles."

Proof of $\implies$: By induction on $|V|$.
Base Case: $|V|=1$. $0=|V|-1$ edges and has no cycles.

Induction Step:
Claim: There is a degree 1 node.
Proof: First, connected $\implies$ every vertex degree ≥ 1.

Proof of only if.

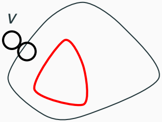


Thm:
"G connected and has $|V|-1$ edges" $\equiv$
"G is connected and has no cycles."

Proof of $\implies$: By induction on $|V|$.
Base Case: $|V|=1$. $0=|V|-1$ edges and has no cycles.

Induction Step:
Claim: There is a degree 1 node.
Proof: First, connected $\implies$ every vertex degree ≥ 1.
Sum of degrees is $2|V|-2$

Proof of only if.

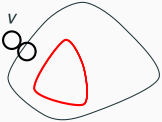


Thm:
"G connected and has $|V|-1$ edges" $\equiv$
"G is connected and has no cycles."

Proof of $\implies$: By induction on $|V|$.
Base Case: $|V|=1$. $0=|V|-1$ edges and has no cycles.

Induction Step:
Claim: There is a degree 1 node.
Proof: First, connected $\implies$ every vertex degree ≥ 1.
Sum of degrees is $2|V|-2$
Average degree $2-2/|V|$

Proof of only if.

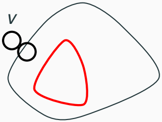


Thm:
"G connected and has $|V|-1$ edges" $\equiv$
"G is connected and has no cycles."

Proof of $\implies$: By induction on $|V|$.
Base Case: $|V| = 1$. $0 = |V| - 1$ edges and has no cycles.

Induction Step:
Claim: There is a degree 1 node.
Proof: First, connected $\implies$ every vertex degree ≥ 1.
Sum of degrees is $2|V| - 2$
Average degree $2 - 2/|V|$
Not everyone is bigger than average!

Proof of only if.

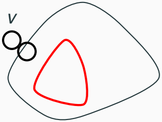


Thm:
"G connected and has $|V|-1$ edges" $\equiv$
"G is connected and has no cycles."

Proof of $\implies$: By induction on $|V|$.
Base Case: $|V|=1$. $0=|V|-1$ edges and has no cycles.

Induction Step:
Claim: There is a degree 1 node.
Proof: First, connected $\implies$ every vertex degree ≥ 1.
Sum of degrees is $2|V|-2$
Average degree $2-2/|V|$
Not everyone is bigger than average! $\square$
By degree 1 removal lemma, $G-v$ is connected.

Proof of only if.

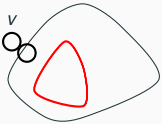


Thm:
"G connected and has $|V|-1$ edges" $\equiv$
"G is connected and has no cycles."

Proof of $\implies$: By induction on $|V|$.
Base Case: $|V|=1$. $0=|V|-1$ edges and has no cycles.

Induction Step:
Claim: There is a degree 1 node.
Proof: First, connected $\implies$ every vertex degree ≥ 1.
Sum of degrees is $2|V|-2$
Average degree $2-2/|V|$
Not everyone is bigger than average! $\square$
By degree 1 removal lemma, $G-v$ is connected.
$G-v$ has $|V|-1$ vertices and $|V|-2$ edges so by induction

Proof of only if.



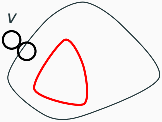
Thm:
"G connected and has $|V|-1$ edges" $\equiv$
"G is connected and has no cycles."

Proof of $\implies$: By induction on $|V|$.
Base Case: $|V|=1$. $0=|V|-1$ edges and has no cycles.

Induction Step:
Claim: There is a degree 1 node.
Proof: First, connected $\implies$ every vertex degree ≥ 1.
Sum of degrees is $2|V|-2$
Average degree $2-2/|V|$
Not everyone is bigger than average! $\square$

By degree 1 removal lemma, $G-v$ is connected.
$G-v$ has $|V|-1$ vertices and $|V|-2$ edges so by induction
$\implies$ no cycle in $G-v$.

Proof of only if.



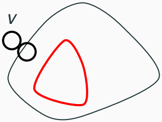
Thm:
"G connected and has $|V|-1$ edges" $\equiv$
"G is connected and has no cycles."

Proof of $\implies$: By induction on $|V|$.
Base Case: $|V|=1$. $0=|V|-1$ edges and has no cycles.

Induction Step:
Claim: There is a degree 1 node.
Proof: First, connected $\implies$ every vertex degree ≥ 1.
Sum of degrees is $2|V|-2$
Average degree $2-2/|V|$
Not everyone is bigger than average! $\square$

By degree 1 removal lemma, $G-v$ is connected.
$G-v$ has $|V|-1$ vertices and $|V|-2$ edges so by induction
$\implies$ no cycle in $G-v$.
And no cycle in G since degree 1 cannot participate in cycle.

Proof of only if.



Thm:
"G connected and has $|V|-1$ edges" $\equiv$
"G is connected and has no cycles."

Proof of $\implies$: By induction on $|V|$.
Base Case: $|V|=1$. $0=|V|-1$ edges and has no cycles.

Induction Step:
Claim: There is a degree 1 node.
Proof: First, connected $\implies$ every vertex degree ≥ 1.
Sum of degrees is $2|V|-2$
Average degree $2-2/|V|$
Not everyone is bigger than average! $\square$
By degree 1 removal lemma, $G-v$ is connected.
$G-v$ has $|V|-1$ vertices and $|V|-2$ edges so by induction
$\implies$ no cycle in $G-v$.
And no cycle in G since degree 1 cannot participate in cycle. $\square$

Proof of if

Thm:
"G is connected and has no cycles" $\implies$ "G connected and has $|V|-1$ edges"

Proof:

Proof of if

Thm:
"G is connected and has no cycles" $\implies$ "G connected and has $|V|-1$ edges"

Proof:
Walk from a vertex using untraversed edges.

Proof of if

Thm:
"G is connected and has no cycles" $\implies$ "G connected and has $|V|-1$ edges"

Proof:
Walk from a vertex using untraversed edges.
Until get stuck.

Proof of if

Thm:
"G is connected and has no cycles" $\implies$ "G connected and has $|V| - 1$ edges"

Proof:
Walk from a vertex using untraversed edges.
Until get stuck.
Claim: Must stuck at a degree 1 vertex.

Proof of if

Thm:
"G is connected and has no cycles" $\implies$ "G connected and has $|V| - 1$ edges"

Proof:
Walk from a vertex using untraversed edges.
Until get stuck.
Claim: Must stuck at a degree 1 vertex.
Proof of Claim:
Can't visit any vertex more than once since no cycle.

Proof of if

Thm:
"G is connected and has no cycles" $\implies$ "G connected and has $|V| - 1$ edges"

Proof:
Walk from a vertex using untraversed edges.
Until get stuck.
Claim: Must stuck at a degree 1 vertex.
Proof of Claim:
Can't visit any vertex more than once since no cycle.
Entered.

Proof of if

Thm:
"G is connected and has no cycles" $\implies$ "G connected and has $|V| - 1$ edges"

Proof:
Walk from a vertex using untraversed edges.
Until get stuck.
Claim: Must stuck at a degree 1 vertex.
Proof of Claim:
Can't visit any vertex more than once since no cycle.
Entered. Didn't leave.

Proof of if

Thm:
"G is connected and has no cycles" $\Longrightarrow$ "G connected and has $|V|-1$ edges"

Proof:
Walk from a vertex using untraversed edges.
Until get stuck.

Claim: Must stuck at a degree 1 vertex.

Proof of Claim:
Can't visit any vertex more than once since no cycle.
Entered. Didn't leave. Only one incident edge.

Proof of if

Thm:
"G is connected and has no cycles" $\implies$ "G connected and has $|V|-1$ edges"

Proof:
Walk from a vertex using untraversed edges.
Until get stuck.
Claim: Must stuck at a degree 1 vertex.
Proof of Claim:
Can't visit any vertex more than once since no cycle.
Entered. Didn't leave. Only one incident edge. $\square$
Removing node doesn't create cycle.

Proof of if

Thm:
"G is connected and has no cycles" $\implies$ "G connected and has $|V| - 1$ edges"

Proof:
Walk from a vertex using untraversed edges.
Until get stuck.
Claim: Must stuck at a degree 1 vertex.
Proof of Claim:
Can't visit any vertex more than once since no cycle.
Entered. Didn't leave. Only one incident edge. $\square$
Removing node doesn't create cycle.
New graph is connected.

Proof of if

Thm:
"G is connected and has no cycles" $\implies$ "G connected and has $|V|-1$ edges"

Proof:
Walk from a vertex using untraversed edges.
Until get stuck.

Claim: Must stuck at a degree 1 vertex.

Proof of Claim:
Can't visit any vertex more than once since no cycle.
Entered. Didn't leave. Only one incident edge. $\square$

Removing node doesn't create cycle.
New graph is connected.
Removing degree 1 node doesn't disconnect from Degree 1 lemma.

Proof of if

Thm:
"G is connected and has no cycles" $\implies$ "G connected and has $|V|-1$ edges"

Proof:
Walk from a vertex using untraversed edges.
Until get stuck.
Claim: Must stuck at a degree 1 vertex.
Proof of Claim:
Can't visit any vertex more than once since no cycle.
Entered. Didn't leave. Only one incident edge. $\square$
Removing node doesn't create cycle.
New graph is connected.
Removing degree 1 node doesn't disconnect from Degree 1 lemma.
By induction $G - v$ has $|V| - 2$ edges.

Proof of if

Thm:
"G is connected and has no cycles" $\implies$ "G connected and has $|V|-1$ edges"

Proof:
Walk from a vertex using untraversed edges.
Until get stuck.

Claim: Must stuck at a degree 1 vertex.

Proof of Claim:
Can't visit any vertex more than once since no cycle.
Entered. Didn't leave. Only one incident edge. $\square$

Removing node doesn't create cycle.
New graph is connected.
Removing degree 1 node doesn't disconnect from Degree 1 lemma.
By induction $G-v$ has $|V|-2$ edges.
G has one more or $|V|-1$ edges.

Proof of if

Thm:
"G is connected and has no cycles" $\implies$ "G connected and has $|V|-1$ edges"

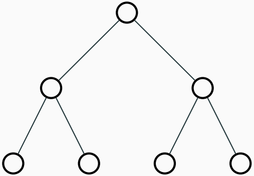
Proof:
Walk from a vertex using untraversed edges.
Until get stuck.
Claim: Must stuck at a degree 1 vertex.
Proof of Claim:
Can't visit any vertex more than once since no cycle.
Entered. Didn't leave. Only one incident edge. $\square$
Removing node doesn't create cycle.
New graph is connected.
Removing degree 1 node doesn't disconnect from Degree 1 lemma.
By induction $G-v$ has $|V|-2$ edges.
G has one more or $|V|-1$ edges. $\square$

Tree's fall apart.

Thm: Can always find a node such that the largest connected component we get by removing it has size at most $|V|/2$

Tree's fall apart.

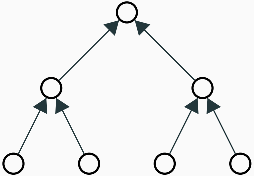
Thm: Can always find a node such that the largest connected component we get by removing it has size at most $|V|/2$



Idea of proof.

Tree's fall apart.

Thm: Can always find a node such that the largest connected component we get by removing it has size at most $|V|/2$

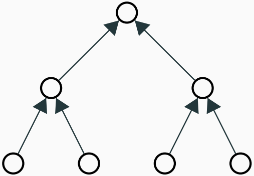


Idea of proof.

Point edge toward bigger side.

Tree's fall apart.

Thm: Can always find a node such that the largest connected component we get by removing it has size at most $|V|/2$

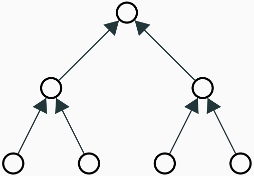


Idea of proof.

Point edge toward bigger side.
    Remove center node.

Tree's fall apart.

Thm: Can always find a node such that the largest connected component we get by removing it has size at most $|V|/2$

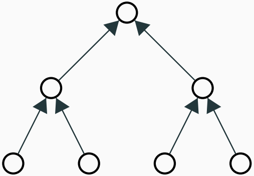


Idea of proof.

Point edge toward bigger side.
  Remove center node.

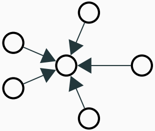
Tree's fall apart.

Thm: Can always find a node such that the largest connected component we get by removing it has size at most $|V|/2$



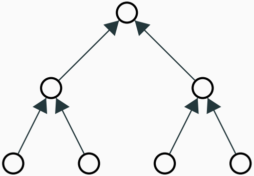
Idea of proof.

Point edge toward bigger side.
Remove center node.



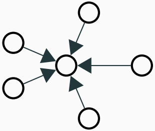
Tree's fall apart.

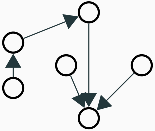
Thm: Can always find a node such that the largest connected component we get by removing it has size at most $|V|/2$



Idea of proof.

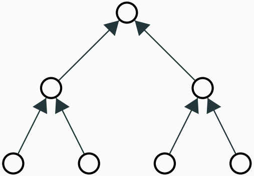
Point edge toward bigger side.
Remove center node.





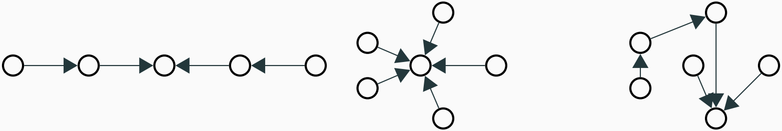
Tree's fall apart.

Thm: Can always find a node such that the largest connected component we get by removing it has size at most $|V|/2$



Idea of proof.

Point edge toward bigger side.
Remove center node.



Hypercubes.

Complete graphs, really connected!

Hypercubes.

Complete graphs, really connected! But lots of edges.

$|V|(|V|-1)/2$

Hypercubes.

Complete graphs, really connected! But lots of edges.

$|V|(|V|-1)/2$

Trees,

Hypercubes.

Complete graphs, really connected! But lots of edges.

$|V|(|V|-1)/2$

Trees, But few edges. $(|V|-1)$

Hypercubes.

Complete graphs, really connected! But lots of edges.

$|V|(|V|-1)/2$

Trees, But few edges. $(|V|-1)$

just falls apart!

Hypercubes.

Complete graphs, really connected! But lots of edges.

$|V|(|V|-1)/2$

Trees, But few edges. $(|V|-1)$

just falls apart!

Hypercubes.

Complete graphs, really connected! But lots of edges.

$|V|(|V|-1)/2$

Trees, But few edges. $(|V|-1)$

just falls apart!

Hypercubes.

Hypercubes.

Complete graphs, really connected! But lots of edges.
    $|V|(|V|-1)/2$
Trees, But few edges. $(|V|-1)$
    just falls apart!

Hypercubes. Really connected.

Hypercubes.

Complete graphs, really connected! But lots of edges.

$|V|(|V|-1)/2$

Trees, But few edges. $(|V|-1)$

just falls apart!

Hypercubes. Really connected.

Also represents bit-strings nicely.

Hypercubes.

Complete graphs, really connected! But lots of edges.

$|V|(|V|-1)/2$

Trees, But few edges. $(|V|-1)$

just falls apart!

Hypercubes. Really connected.

Also represents bit-strings nicely.

Hypercubes.

Complete graphs, really connected! But lots of edges.

$|V|(|V|-1)/2$

Trees, But few edges. $(|V|-1)$

just falls apart!

Hypercubes. Really connected.

Also represents bit-strings nicely.

$G=(V,E)$

Hypercubes.

Complete graphs, really connected! But lots of edges.

$|V|(|V|-1)/2$

Trees, But few edges. $(|V|-1)$

just falls apart!

Hypercubes. Really connected.

Also represents bit-strings nicely.

$G = (V,E)$

$|V| = \{0,1\}^n$,

Hypercubes.

Complete graphs, really connected! But lots of edges.

$|V|(|V|-1)/2$

Trees, But few edges. $(|V|-1)$

just falls apart!

Hypercubes. Really connected.

Also represents bit-strings nicely.

$G=(V,E)$

$|V|=\{0,1\}^n$,

$|E|=\{(x,y)|x$ and y differ in one bit position.$\}$

Hypercubes.

Complete graphs, really connected! But lots of edges.

$|V|(|V|-1)/2$

Trees, But few edges. $(|V|-1)$

just falls apart!

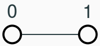
Hypercubes. Really connected.

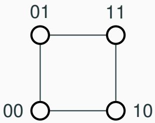
Also represents bit-strings nicely.

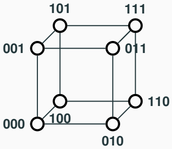
$G = (V, E)$

$|V| = \{0,1\}^n$,

$|E| = \{(x,y) | x$ and y differ in one bit position.$\}$







Hypercubes.

Complete graphs, really connected! But lots of edges.

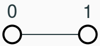
$|V|(|V|-1)/2$

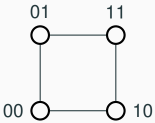
Trees, But few edges. $(|V|-1)$

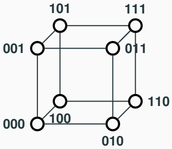
just falls apart!

Hypercubes. Really connected.
Also represents bit-strings nicely.

$G = (V,E)$
$|V| = \{0,1\}^n$,
$|E| = \{(x,y)|x$ and y differ in one bit position.$\}$







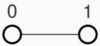
2^n vertices.

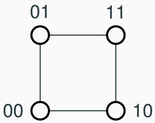
Hypercubes.

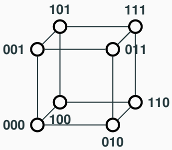
Complete graphs, really connected! But lots of edges.
$|V|(|V|-1)/2$
Trees, But few edges. $(|V|-1)$
just falls apart!

Hypercubes. Really connected.
Also represents bit-strings nicely.

$G=(V,E)$
$|V|=\{0,1\}^n$,
$|E|=\{(x,y)|x$ and y differ in one bit position.$\}$







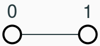
2^n vertices. number of n-bit strings!

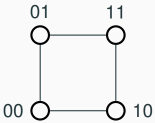
Hypercubes.

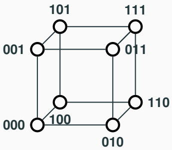
Complete graphs, really connected! But lots of edges.
$|V|(|V|-1)/2$
Trees, But few edges. $(|V|-1)$
just falls apart!

Hypercubes. Really connected.
Also represents bit-strings nicely.

$G=(V,E)$
$|V|=\{0,1\}^n$,
$|E|=\{(x,y)|x$ and y differ in one bit position.$\}$







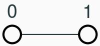
2^n vertices. number of n-bit strings!
$n2^{n-1}$ edges.

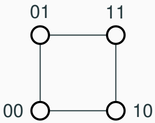
Hypercubes.

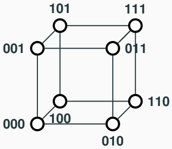
Complete graphs, really connected! But lots of edges.
  $|V|(|V|-1)/2$
Trees, But few edges. $(|V|-1)$
  just falls apart!

Hypercubes. Really connected.
 Also represents bit-strings nicely.

$G=(V,E)$
 $|V|=\{0,1\}^n$,
 $|E|=\{(x,y)|x$ and y differ in one bit position.$\}$







2^n vertices. number of n-bit strings!
$n2^{n-1}$ edges.
  2^n vertices each of degree n

Hypercubes.

Complete graphs, really connected! But lots of edges.

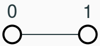
$|V|(|V|-1)/2$

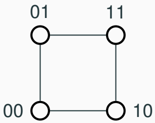
Trees, But few edges. $(|V|-1)$

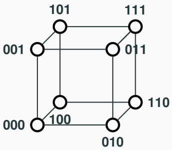
just falls apart!

Hypercubes. Really connected.
Also represents bit-strings nicely.

$G=(V,E)$
$|V|=\{0,1\}^n$,
$|E|=\{(x,y)|x$ and y differ in one bit position.$\}$







2^n vertices. number of n-bit strings!
$n2^{n-1}$ edges.

2^n vertices each of degree n

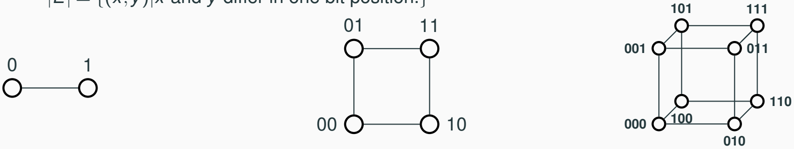
total degree is $n2^n$

Hypercubes.

Complete graphs, really connected! But lots of edges.
  $|V|(|V|-1)/2$
Trees, But few edges. $(|V|-1)$
  just falls apart!

Hypercubes. Really connected.
 Also represents bit-strings nicely.

$G=(V,E)$
 $|V|=\{0,1\}^n$,
 $|E|=\{(x,y)|x$ and y differ in one bit position.$\}$



2^n vertices. number of n-bit strings!
$n2^{n-1}$ edges.
  2^n vertices each of degree n
   total degree is $n2^n$ and half as many edges!

Hypercubes.

Complete graphs, really connected! But lots of edges.

$|V|(|V|-1)/2$

Trees, But few edges. $(|V|-1)$

just falls apart!

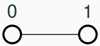
Hypercubes. Really connected.

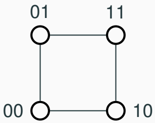
Also represents bit-strings nicely.

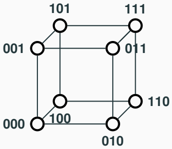
$G = (V, E)$

$|V| = \{0,1\}^n$,

$|E| = \{(x,y) | x$ and y differ in one bit position.$\}$







2^n vertices. number of n-bit strings!

$n2^{n-1}$ edges.

2^n vertices each of degree n

total degree is $n2^n$ and half as many edges!

Recursive Definition.

A 0-dimensional hypercube is a node labelled with the empty string of bits.

Recursive Definition.

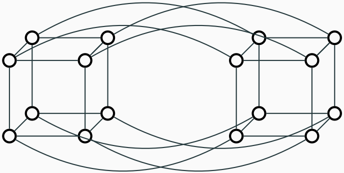
A 0-dimensional hypercube is a node labelled with the empty string of bits.

An n-dimensional hypercube consists of a 0-subcube (1-subcube) which is a $n-1$-dimensional hypercube with nodes labelled $0x$ ($1x$) with the additional edges $(0x, 1x)$.

Recursive Definition.

A 0-dimensional hypercube is a node labelled with the empty string of bits.

An n-dimensional hypercube consists of a 0-subcube (1-subcube) which is a $n-1$-dimensional hypercube with nodes labelled $0x$ ($1x$) with the additional edges $(0x, 1x)$.



Hypercube: Can't cut me!

Hypercube: Can't cut me!

Thm: Any subset S of the hypercube where $|S| \leq |V|/2$ has $\geq |S|$ edges connecting it to $V - S$;

Hypercube: Can't cut me!

Thm: Any subset S of the hypercube where $|S| \leq |V|/2$ has $\geq |S|$ edges connecting it to $V - S$; $|E \cap S \times (V - S)| \geq |S|$

Hypercube: Can't cut me!

Thm: Any subset S of the hypercube where $|S| \leq |V|/2$ has $\geq |S|$ edges connecting it to $V - S$; $|E \cap S \times (V - S)| \geq |S|$

Terminology:

Hypercube: Can't cut me!

Thm: Any subset S of the hypercube where $|S| \leq |V|/2$ has $\geq |S|$ edges connecting it to $V - S$; $|E \cap S \times (V - S)| \geq |S|$

Terminology:
$(S, V - S)$ is cut.

Hypercube: Can't cut me!

Thm: Any subset S of the hypercube where $|S| \leq |V|/2$ has $\geq |S|$ edges connecting it to $V - S$; $|E \cap S \times (V - S)| \geq |S|$

Terminology:
  $(S, V - S)$ is cut.
    a partition of the vertices of a graph into two disjoint subsets.

Hypercube: Can't cut me!

Thm: Any subset S of the hypercube where $|S| \leq |V|/2$ has $\geq |S|$ edges connecting it to $V - S$; $|E \cap S \times (V - S)| \geq |S|$

Terminology:

$(S, V - S)$ is cut.

a partition of the vertices of a graph into two disjoint subsets.

$(E \cap S \times (V - S))$ - cut edges.

Hypercube: Can't cut me!

Thm: Any subset S of the hypercube where $|S| \leq |V|/2$ has $\geq |S|$ edges connecting it to $V - S$; $|E \cap S \times (V - S)| \geq |S|$

Terminology:
  $(S, V - S)$ is cut.
    a partition of the vertices of a graph into two disjoint subsets.
  $(E \cap S \times (V - S))$ - cut edges.

Hypercube: Can't cut me!

Thm: Any subset S of the hypercube where $|S| \leq |V|/2$ has $\geq |S|$ edges connecting it to $V - S$; $|E \cap S \times (V - S)| \geq |S|$

Terminology:

$(S, V - S)$ is cut.

a partition of the vertices of a graph into two disjoint subsets.

$(E \cap S \times (V - S))$ - cut edges.

Restatement: for any cut in the hypercube, the number of cut edges is at least the size of the small side.

Proof of Large Cuts.

Thm: For any cut $(S, V - S)$ in the hypercube, the number of cut edges is at least the size of the small side.

Proof:

Proof of Large Cuts.

Thm: For any cut $(S, V - S)$ in the hypercube, the number of cut edges is at least the size of the small side.

Proof:
Base Case: $n = 1$

Proof of Large Cuts.

Thm: For any cut $(S, V-S)$ in the hypercube, the number of cut edges is at least the size of the small side.

Proof:
Base Case: $n = 1$ V= $\{0,1\}$.

Proof of Large Cuts.

Thm: For any cut $(S, V-S)$ in the hypercube, the number of cut edges is at least the size of the small side.

Proof:
Base Case: $n=1$ V= $\{0,1\}$.

$S=\{0\}, |\text{ cut edges }|=1$ $\qquad$ $S=\emptyset, |\text{ cut edges }|=0$

Proof of Large Cuts.

Thm: For any cut $(S, V-S)$ in the hypercube, the number of cut edges is at least the size of the small side.

Proof:

Base Case: $n=1$ V= $\{0,1\}$.

$S=\{0\}, |\text{ cut edges }|=1$ $\qquad$ $S=\emptyset, |\text{ cut edges }|=0$

Induction Step Idea

Thm: For any cut $(S, V - S)$ in the hypercube, the number of cut edges is at least the size of the small side.

Induction Step Idea

Thm: For any cut $(S, V - S)$ in the hypercube, the number of cut edges is at least the size of the small side.

Use recursive definition into two subcubes.

Induction Step Idea

Thm: For any cut $(S, V - S)$ in the hypercube, the number of cut edges is at least the size of the small side.

Use recursive definition into two subcubes.

Two cubes connected by edges.

Induction Step Idea

Thm: For any cut $(S, V - S)$ in the hypercube, the number of cut edges is at least the size of the small side.

Use recursive definition into two subcubes.

Two cubes connected by edges.

Case 1: Count edges inside subcube inductively.

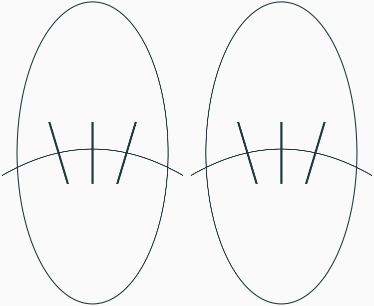
Induction Step Idea

Thm: For any cut $(S, V - S)$ in the hypercube, the number of cut edges is at least the size of the small side.

Use recursive definition into two subcubes.

Two cubes connected by edges.

Case 1: Count edges inside subcube inductively.



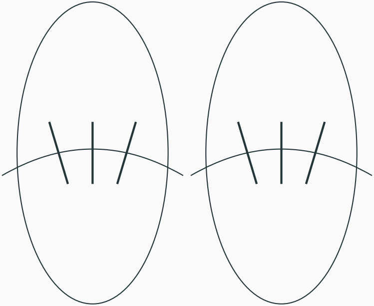
Induction Step Idea

Thm: For any cut $(S, V - S)$ in the hypercube, the number of cut edges is at least the size of the small side.

Use recursive definition into two subcubes.

Two cubes connected by edges.

Case 1: Count edges inside subcube inductively.



Case 2: Count inside and across.

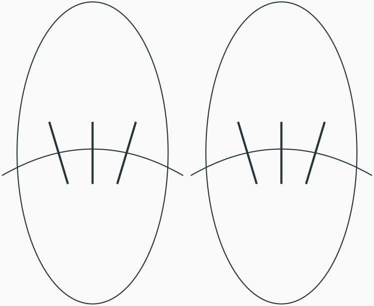
Induction Step Idea

Thm: For any cut $(S, V - S)$ in the hypercube, the number of cut edges is at least the size of the small side.

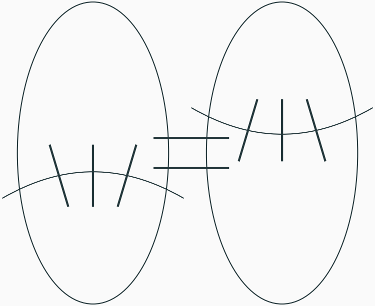
Use recursive definition into two subcubes.

Two cubes connected by edges.

Case 1: Count edges inside subcube inductively.



Case 2: Count inside and across.



Induction Step

Thm: For any cut $(S, V - S)$ in the hypercube, the number of cut edges is at least the size of the small side, $|S|$.

Induction Step

Thm: For any cut $(S, V - S)$ in the hypercube, the number of cut edges is at least the size of the small side, $|S|$.

Proof: Induction Step.

Induction Step

Thm: For any cut $(S, V - S)$ in the hypercube, the number of cut edges is at least the size of the small side, $|S|$.

Proof: Induction Step.
Recursive definition:

Induction Step

Thm: For any cut $(S, V - S)$ in the hypercube, the number of cut edges is at least the size of the small side, $|S|$.

Proof: Induction Step.
Recursive definition:
$H_0 = (V_0, E_0)$,$H_1 = (V_1, E_1)$, edges E_x that connect them.

Induction Step

Thm: For any cut $(S, V - S)$ in the hypercube, the number of cut edges is at least the size of the small side, $|S|$.

Proof: Induction Step.
Recursive definition:

$H_0 = (V_0, E_0)$,$H_1 = (V_1, E_1)$, edges E_x that connect them.
$H = (V_0 \cup V_1, E_0 \cup E_1 \cup E_x)$

Induction Step

Thm: For any cut $(S, V - S)$ in the hypercube, the number of cut edges is at least the size of the small side, $|S|$.

Proof: Induction Step.
Recursive definition:

$H_0 = (V_0, E_0)$,$H_1 = (V_1, E_1)$, edges E_x that connect them.
$H = (V_0 \cup V_1, E_0 \cup E_1 \cup E_x)$
$S = S_0 \cup S_1$ where S_0 in first, and S_1 in other.

Induction Step

Thm: For any cut $(S, V - S)$ in the hypercube, the number of cut edges is at least the size of the small side, $|S|$.

Proof: Induction Step.

Recursive definition:

$H_0 = (V_0, E_0)$,$H_1 = (V_1, E_1)$, edges E_x that connect them.

$H = (V_0 \cup V_1, E_0 \cup E_1 \cup E_x)$

$S = S_0 \cup S_1$ where S_0 in first, and S_1 in other.

Case 1: $|S_0| \le |V_0|/2, |S_1| \le |V_1|/2$

Induction Step

Thm: For any cut $(S, V-S)$ in the hypercube, the number of cut edges is at least the size of the small side, $|S|$.

Proof: Induction Step.

Recursive definition:

$H_0 = (V_0, E_0)$,$H_1 = (V_1, E_1)$, edges E_x that connect them.

$H = (V_0 \cup V_1, E_0 \cup E_1 \cup E_x)$

$S = S_0 \cup S_1$ where S_0 in first, and S_1 in other.

Case 1: $|S_0| \leq |V_0|/2, |S_1| \leq |V_1|/2$

Both S_0 and S_1 are small sides.

Induction Step

Thm: For any cut $(S, V - S)$ in the hypercube, the number of cut edges is at least the size of the small side, $|S|$.

Proof: Induction Step.
Recursive definition:

$H_0 = (V_0, E_0)$,$H_1 = (V_1, E_1)$, edges E_x that connect them.
$H = (V_0 \cup V_1, E_0 \cup E_1 \cup E_x)$
$S = S_0 \cup S_1$ where S_0 in first, and S_1 in other.

Case 1: $|S_0| \leq |V_0|/2, |S_1| \leq |V_1|/2$
Both S_0 and S_1 are small sides. So by induction.

Induction Step

Thm: For any cut $(S, V - S)$ in the hypercube, the number of cut edges is at least the size of the small side, $|S|$.

Proof: Induction Step.

Recursive definition:

$H_0 = (V_0, E_0)$,$H_1 = (V_1, E_1)$, edges E_x that connect them.

$H = (V_0 \cup V_1, E_0 \cup E_1 \cup E_x)$

$S = S_0 \cup S_1$ where S_0 in first, and S_1 in other.

Case 1: $|S_0| \leq |V_0|/2, |S_1| \leq |V_1|/2$

Both S_0 and S_1 are small sides. So by induction.

Edges cut in $H_0 \geq |S_0|$.

Induction Step

Thm: For any cut $(S, V - S)$ in the hypercube, the number of cut edges is at least the size of the small side, $|S|$.

Proof: Induction Step.
Recursive definition:
$H_0 = (V_0, E_0)$,$H_1 = (V_1, E_1)$, edges E_x that connect them.
$H = (V_0 \cup V_1, E_0 \cup E_1 \cup E_x)$
$S = S_0 \cup S_1$ where S_0 in first, and S_1 in other.

Case 1: $|S_0| \leq |V_0|/2, |S_1| \leq |V_1|/2$
Both S_0 and S_1 are small sides. So by induction.
Edges cut in $H_0 \geq |S_0|$.
Edges cut in $H_1 \geq |S_1|$.

Induction Step

Thm: For any cut $(S, V-S)$ in the hypercube, the number of cut edges is at least the size of the small side, $|S|$.

Proof: Induction Step.

Recursive definition:

$H_0 = (V_0, E_0)$,$H_1 = (V_1, E_1)$, edges E_x that connect them.

$H = (V_0 \cup V_1, E_0 \cup E_1 \cup E_x)$

$S = S_0 \cup S_1$ where S_0 in first, and S_1 in other.

Case 1: $|S_0| \leq |V_0|/2, |S_1| \leq |V_1|/2$

Both S_0 and S_1 are small sides. So by induction.

Edges cut in $H_0 \geq |S_0|$.

Edges cut in $H_1 \geq |S_1|$.

Induction Step

Thm: For any cut $(S, V - S)$ in the hypercube, the number of cut edges is at least the size of the small side, $|S|$.

Proof: Induction Step.

Recursive definition:

$H_0 = (V_0, E_0)$,$H_1 = (V_1, E_1)$, edges E_x that connect them.

$H = (V_0 \cup V_1, E_0 \cup E_1 \cup E_x)$

$S = S_0 \cup S_1$ where S_0 in first, and S_1 in other.

Case 1: $|S_0| \leq |V_0|/2, |S_1| \leq |V_1|/2$

Both S_0 and S_1 are small sides. So by induction.

Edges cut in $H_0 \geq |S_0|$.

Edges cut in $H_1 \geq |S_1|$.

Total cut edges $\geq |S_0| + |S_1| = |S|$.

Induction Step

Thm: For any cut $(S, V - S)$ in the hypercube, the number of cut edges is at least the size of the small side, $|S|$.

Proof: Induction Step.
Recursive definition:
$H_0 = (V_0, E_0)$,$H_1 = (V_1, E_1)$, edges E_x that connect them.
$H = (V_0 \cup V_1, E_0 \cup E_1 \cup E_x)$
$S = S_0 \cup S_1$ where S_0 in first, and S_1 in other.

Case 1: $|S_0| \leq |V_0|/2, |S_1| \leq |V_1|/2$
Both S_0 and S_1 are small sides. So by induction.
Edges cut in $H_0 \geq |S_0|$.
Edges cut in $H_1 \geq |S_1|$.

Total cut edges $\geq |S_0| + |S_1| = |S|$. □

Thm: For any cut $(S, V - S)$ in the hypercube, the number of cut edges is at least the size of the small side, $|S|$.

Proof: Induction Step. Case 2. $|S_0| \geq |V_0|/2$.

Induction Step. Case 2.

Thm: For any cut $(S, V - S)$ in the hypercube, the number of cut edges is at least the size of the small side, $|S|$.

Proof: Induction Step. Case 2. $|S_0| \geq |V_0|/2$.

Recall Case 1: $|S_0|, |S_1| \leq |V|/2$

$|S_1| \leq |V_1|/2$ since $|S| \leq |V|/2$.

Induction Step. Case 2.

Thm: For any cut $(S, V - S)$ in the hypercube, the number of cut edges is at least the size of the small side, $|S|$.

Proof: Induction Step. Case 2. $|S_0| \geq |V_0|/2$.

Recall Case 1: $|S_0|, |S_1| \leq |V|/2$

$|S_1| \leq |V_1|/2$ since $|S| \leq |V|/2$.

$\implies \geq |S_1|$ edges cut in E_1.

Induction Step. Case 2.

Thm: For any cut $(S, V-S)$ in the hypercube, the number of cut edges is at least the size of the small side, $|S|$.

Proof: Induction Step. Case 2. $|S_0| \geq |V_0|/2$.

Recall Case 1: $|S_0|, |S_1| \leq |V|/2$

$|S_1| \leq |V_1|/2$ since $|S| \leq |V|/2$.

$\implies \geq |S_1|$ edges cut in E_1.

$|S_0| \geq |V_0|/2 \implies |V_0 - S_0| \leq |V_0|/2$

Induction Step. Case 2.

Thm: For any cut $(S, V-S)$ in the hypercube, the number of cut edges is at least the size of the small side, $|S|$.

Proof: Induction Step. Case 2. $|S_0| \geq |V_0|/2$.

Recall Case 1: $|S_0|, |S_1| \leq |V|/2$

$|S_1| \leq |V_1|/2$ since $|S| \leq |V|/2$.

$\implies \geq |S_1|$ edges cut in E_1.

$|S_0| \geq |V_0|/2 \implies |V_0 - S_0| \leq |V_0|/2$

$\implies \geq |V_0| - |S_0|$ edges cut in E_0.

Induction Step. Case 2.

Thm: For any cut $(S, V - S)$ in the hypercube, the number of cut edges is at least the size of the small side, $|S|$.

Proof: Induction Step. Case 2. $|S_0| \geq |V_0|/2$.

Recall Case 1: $|S_0|, |S_1| \leq |V|/2$

$|S_1| \leq |V_1|/2$ since $|S| \leq |V|/2$.

$\implies \geq |S_1|$ edges cut in E_1.

$|S_0| \geq |V_0|/2 \implies |V_0 - S_0| \leq |V_0|/2$

$\implies \geq |V_0| - |S_0|$ edges cut in E_0.

Edges in E_x connect corresponding nodes.

Induction Step. Case 2.

Thm: For any cut $(S, V-S)$ in the hypercube, the number of cut edges is at least the size of the small side, $|S|$.

Proof: Induction Step. Case 2. $|S_0| \geq |V_0|/2$.

Recall Case 1: $|S_0|, |S_1| \leq |V|/2$

$|S_1| \leq |V_1|/2$ since $|S| \leq |V|/2$.

$\implies \geq |S_1|$ edges cut in E_1.

$|S_0| \geq |V_0|/2 \implies |V_0 - S_0| \leq |V_0|/2$

$\implies \geq |V_0| - |S_0|$ edges cut in E_0.

Edges in E_x connect corresponding nodes.

$\implies \geq |S_0| - |S_1|$ edges cut in E_x.

Induction Step. Case 2.

Thm: For any cut $(S, V-S)$ in the hypercube, the number of cut edges is at least the size of the small side, $|S|$.

Proof: Induction Step. Case 2. $|S_0| \geq |V_0|/2$.

Recall Case 1: $|S_0|, |S_1| \leq |V|/2$

$|S_1| \leq |V_1|/2$ since $|S| \leq |V|/2$.

$\implies \geq |S_1|$ edges cut in E_1.

$|S_0| \geq |V_0|/2 \implies |V_0 - S_0| \leq |V_0|/2$

$\implies \geq |V_0| - |S_0|$ edges cut in E_0.

Edges in E_x connect corresponding nodes.

$\implies \geq |S_0| - |S_1|$ edges cut in E_x.

Induction Step. Case 2.

Thm: For any cut $(S, V - S)$ in the hypercube, the number of cut edges is at least the size of the small side, $|S|$.

Proof: Induction Step. Case 2. $|S_0| \geq |V_0|/2$.

Recall Case 1: $|S_0|, |S_1| \leq |V|/2$

$|S_1| \leq |V_1|/2$ since $|S| \leq |V|/2$.

$\implies \geq |S_1|$ edges cut in E_1.

$|S_0| \geq |V_0|/2 \implies |V_0 - S_0| \leq |V_0|/2$

$\implies \geq |V_0| - |S_0|$ edges cut in E_0.

Edges in E_x connect corresponding nodes.

$\implies \geq |S_0| - |S_1|$ edges cut in E_x.

Total edges cut:

Induction Step. Case 2.

Thm: For any cut $(S, V - S)$ in the hypercube, the number of cut edges is at least the size of the small side, $|S|$.

Proof: Induction Step. Case 2. $|S_0| \geq |V_0|/2$.

Recall Case 1: $|S_0|, |S_1| \leq |V|/2$

$|S_1| \leq |V_1|/2$ since $|S| \leq |V|/2$.

$\implies \geq |S_1|$ edges cut in E_1.

$|S_0| \geq |V_0|/2 \implies |V_0 - S_0| \leq |V_0|/2$

$\implies \geq |V_0| - |S_0|$ edges cut in E_0.

Edges in E_x connect corresponding nodes.

$\implies \geq |S_0| - |S_1|$ edges cut in E_x.

Total edges cut:

$\geq$

Induction Step. Case 2.

Thm: For any cut $(S, V - S)$ in the hypercube, the number of cut edges is at least the size of the small side, $|S|$.

Proof: Induction Step. Case 2. $|S_0| \geq |V_0|/2$.

Recall Case 1: $|S_0|, |S_1| \leq |V|/2$

$|S_1| \leq |V_1|/2$ since $|S| \leq |V|/2$.

$\implies \geq |S_1|$ edges cut in E_1.

$|S_0| \geq |V_0|/2 \implies |V_0 - S_0| \leq |V_0|/2$

$\implies \geq |V_0| - |S_0|$ edges cut in E_0.

Edges in E_x connect corresponding nodes.

$\implies \geq |S_0| - |S_1|$ edges cut in E_x.

Total edges cut:

$\geq \ |S_1|$

Induction Step. Case 2.

Thm: For any cut $(S, V - S)$ in the hypercube, the number of cut edges is at least the size of the small side, $|S|$.

Proof: Induction Step. Case 2. $|S_0| \geq |V_0|/2$.

Recall Case 1: $|S_0|, |S_1| \leq |V|/2$

$|S_1| \leq |V_1|/2$ since $|S| \leq |V|/2$.

$\implies \geq |S_1|$ edges cut in E_1.

$|S_0| \geq |V_0|/2 \implies |V_0 - S_0| \leq |V_0|/2$

$\implies \geq |V_0| - |S_0|$ edges cut in E_0.

Edges in E_x connect corresponding nodes.

$\implies \geq |S_0| - |S_1|$ edges cut in E_x.

Total edges cut:

$\geq \; |S_1| + |V_0| - |S_0|$

Induction Step. Case 2.

Thm: For any cut $(S, V - S)$ in the hypercube, the number of cut edges is at least the size of the small side, $|S|$.

Proof: Induction Step. Case 2. $|S_0| \geq |V_0|/2$.

Recall Case 1: $|S_0|, |S_1| \leq |V|/2$

$|S_1| \leq |V_1|/2$ since $|S| \leq |V|/2$.

$\implies \geq |S_1|$ edges cut in E_1.

$|S_0| \geq |V_0|/2 \implies |V_0 - S_0| \leq |V_0|/2$

$\implies \geq |V_0| - |S_0|$ edges cut in E_0.

Edges in E_x connect corresponding nodes.

$\implies \geq |S_0| - |S_1|$ edges cut in E_x.

Total edges cut:

$\geq \; |S_1| + |V_0| - |S_0| + |S_0| - |S_1|$

Induction Step. Case 2.

Thm: For any cut $(S, V-S)$ in the hypercube, the number of cut edges is at least the size of the small side, $|S|$.

Proof: Induction Step. Case 2. $|S_0| \geq |V_0|/2$.

Recall Case 1: $|S_0|, |S_1| \leq |V|/2$

$|S_1| \leq |V_1|/2$ since $|S| \leq |V|/2$.

$\implies \geq |S_1|$ edges cut in E_1.

$|S_0| \geq |V_0|/2 \implies |V_0 - S_0| \leq |V_0|/2$

$\implies \geq |V_0| - |S_0|$ edges cut in E_0.

Edges in E_x connect corresponding nodes.

$\implies \geq |S_0| - |S_1|$ edges cut in E_x.

Total edges cut:

$\geq \; |S_1| + |V_0| - |S_0| + |S_0| - |S_1| = |V_0|$

Induction Step. Case 2.

Thm: For any cut $(S, V-S)$ in the hypercube, the number of cut edges is at least the size of the small side, $|S|$.

Proof: Induction Step. Case 2. $|S_0| \geq |V_0|/2$.

Recall Case 1: $|S_0|,|S_1| \leq |V|/2$

$|S_1| \leq |V_1|/2$ since $|S| \leq |V|/2$.

$\implies \geq |S_1|$ edges cut in E_1.

$|S_0| \geq |V_0|/2 \implies |V_0 - S_0| \leq |V_0|/2$

$\implies \geq |V_0| - |S_0|$ edges cut in E_0.

Edges in E_x connect corresponding nodes.

$\implies \geq |S_0| - |S_1|$ edges cut in E_x.

Total edges cut:

$\geq \ |S_1| + |V_0| - |S_0| + |S_0| - |S_1| = |V_0|$

$|V_0|$

Induction Step. Case 2.

Thm: For any cut $(S, V-S)$ in the hypercube, the number of cut edges is at least the size of the small side, $|S|$.

Proof: Induction Step. Case 2. $|S_0| \geq |V_0|/2$.

Recall Case 1: $|S_0|, |S_1| \leq |V|/2$

$|S_1| \leq |V_1|/2$ since $|S| \leq |V|/2$.

$\implies \geq |S_1|$ edges cut in E_1.

$|S_0| \geq |V_0|/2 \implies |V_0 - S_0| \leq |V_0|/2$

$\implies \geq |V_0| - |S_0|$ edges cut in E_0.

Edges in E_x connect corresponding nodes.

$\implies \geq |S_0| - |S_1|$ edges cut in E_x.

Total edges cut:

$\geq |S_1| + |V_0| - |S_0| + |S_0| - |S_1| = |V_0|$

$|V_0| = |V|/2 \geq |S|$.

Induction Step. Case 2.

Thm: For any cut $(S, V - S)$ in the hypercube, the number of cut edges is at least the size of the small side, $|S|$.

Proof: Induction Step. Case 2. $|S_0| \geq |V_0|/2$.

Recall Case 1: $|S_0|, |S_1| \leq |V|/2$

$|S_1| \leq |V_1|/2$ since $|S| \leq |V|/2$.

$\implies \geq |S_1|$ edges cut in E_1.

$|S_0| \geq |V_0|/2 \implies |V_0 - S_0| \leq |V_0|/2$

$\implies \geq |V_0| - |S_0|$ edges cut in E_0.

Edges in E_x connect corresponding nodes.

$\implies \geq |S_0| - |S_1|$ edges cut in E_x.

Total edges cut:

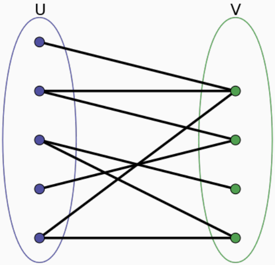
$\geq \ |S_1| + |V_0| - |S_0| + |S_0| - |S_1| = |V_0|$

$|V_0| = |V|/2 \geq |S|$. $\square$

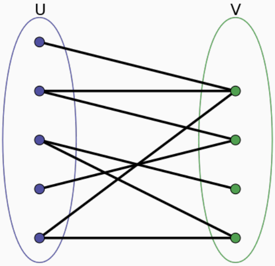
Also, case 3 where $|S_1| \geq |V|/2$ is symmetric.

Bipartite graph

Bipartite graph

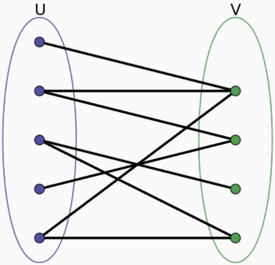


Bipartite graph



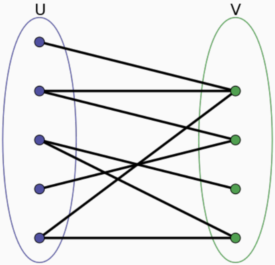
Bipartite graph:

Bipartite graph



Bipartite graph: a bipartite graph is a graph whose vertices can be divided into two disjoint sets U and V such that every edge connects a vertex in U to one in V.

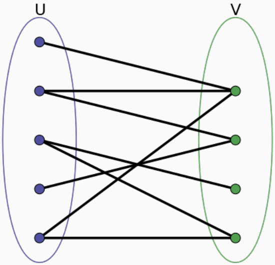
Bipartite graph



Bipartite graph: a bipartite graph is a graph whose vertices can be divided into two disjoint sets U and V such that every edge connects a vertex in U to one in V.

U and V are sometimes called the parts of the graph.

Bipartite graph

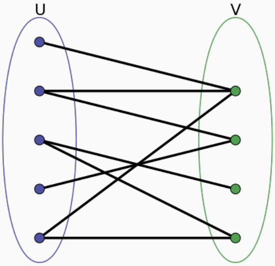


Bipartite graph: a bipartite graph is a graph whose vertices can be divided into two disjoint sets U and V such that every edge connects a vertex in U to one in V.

U and V are sometimes called the parts of the graph.

Coloring?

Bipartite graph

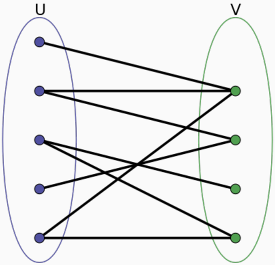


Bipartite graph: a bipartite graph is a graph whose vertices can be divided into two disjoint sets U and V such that every edge connects a vertex in U to one in V.

U and V are sometimes called the parts of the graph.

Coloring? How many colors do we need?

Bipartite graph



Bipartite graph: a bipartite graph is a graph whose vertices can be divided into two disjoint sets U and V such that every edge connects a vertex in U to one in V.

U and V are sometimes called the parts of the graph.

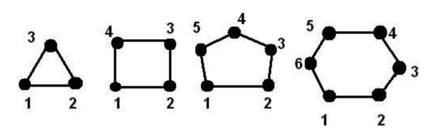
Coloring? How many colors do we need? 2!

Bipartite?

Which of the following graphs are bipartite?

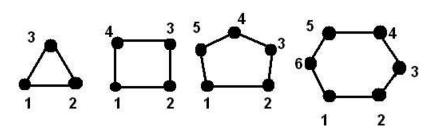
Bipartite?

Which of the following graphs are bipartite?



Bipartite?

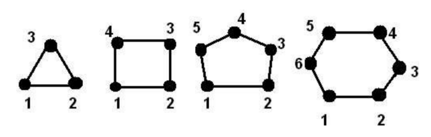
Which of the following graphs are bipartite?



No

Bipartite?

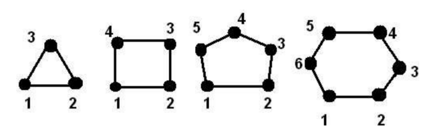
Which of the following graphs are bipartite?



No Yes

Bipartite?

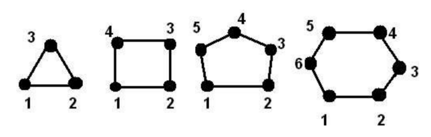
Which of the following graphs are bipartite?



No Yes No

Bipartite?

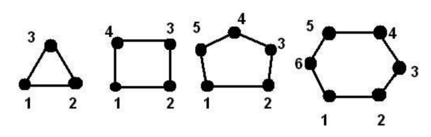
Which of the following graphs are bipartite?



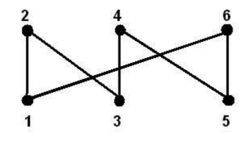
No Yes No Yes

Bipartite?

Which of the following graphs are bipartite?

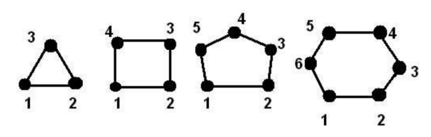


No Yes No Yes

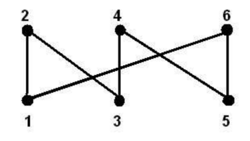


Bipartite?

Which of the following graphs are bipartite?



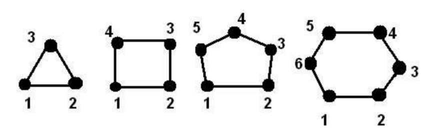
No Yes No Yes



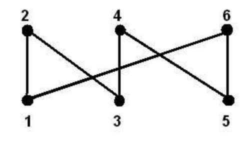
A graph is a bipartite graph if and only if it does not contain any odd-length cycles.

Bipartite?

Which of the following graphs are bipartite?



No Yes No Yes



A graph is a bipartite graph if and only if it does not contain any odd-length cycles.

Proof

Proof

Only if: trivial

Proof

Only if: trivial

Start at a node v in one part, say V, the cycle must be like leaving V, entering V, . . .

Proof

Only if: trivial

Start at a node v in one part, say V, the cycle must be like leaving V, entering V, ...
Also the cycle must end at v, so the cycle must end with ”entering V”.

Proof

Only if: trivial

Start at a node v in one part, say V, the cycle must be like leaving V, entering V, ...
Also the cycle must end at v, so the cycle must end with "entering V".All paired up, even length.

Proof

Only if: trivial

Start at a node v in one part, say V, the cycle must be like leaving V, entering V, ...
Also the cycle must end at v, so the cycle must end with "entering V".All paired up, even length.

No odd-length cycle $\implies$ bipartite:

Proof

Only if: trivial

Start at a node v in one part, say V, the cycle must be like leaving V, entering V, ...
Also the cycle must end at v, so the cycle must end with ”entering V”.All paired up, even length.

No odd-length cycle $\implies$ bipartite:

Different connected components does not influence each other,

Proof

Only if: trivial

Start at a node v in one part, say V, the cycle must be like leaving V, entering V, ...
Also the cycle must end at v, so the cycle must end with "entering V".All paired up, even length.

No odd-length cycle $\implies$ bipartite:

Different connected components does not influence each other, just look at one first

Proof

Only if: trivial

Start at a node v in one part, say V, the cycle must be like leaving V, entering V, ...
Also the cycle must end at v, so the cycle must end with ”entering V”.All paired up, even length.

No odd-length cycle $\implies$ bipartite:

Different connected components does not influence each other, just look at one first

Pick one arbitrary vertex v, split all vertices into two groups

Proof

Only if: trivial

Start at a node v in one part, say V, the cycle must be like leaving V, entering V, ...
Also the cycle must end at v, so the cycle must end with "entering V".All paired up, even length.

No odd-length cycle $\implies$ bipartite:

Different connected components does not influence each other, just look at one first

Pick one arbitrary vertex v, split all vertices into two groups
$A = \{u \in V | \exists$ odd length path from v to $u\}$

Proof

Only if: trivial

Start at a node v in one part, say V, the cycle must be like leaving V, entering V, ...
Also the cycle must end at v, so the cycle must end with ”entering V”.All paired up, even length.

No odd-length cycle $\implies$ bipartite:

Different connected components does not influence each other, just look at one first

Pick one arbitrary vertex v, split all vertices into two groups
$A = \{u \in V | \exists \text{ odd length path from } v \text{ to } u\}$
$B = \{u \in V | \exists \text{ even length path from } v \text{ to } u\}$

Proof

Only if: trivial

Start at a node v in one part, say V, the cycle must be like leaving V, entering V, ...
Also the cycle must end at v, so the cycle must end with "entering V".All paired up, even length.

No odd-length cycle $\implies$ bipartite:

Different connected components does not influence each other, just look at one first

Pick one arbitrary vertex v, split all vertices into two groups
$A = \{u \in V | \exists$ odd length path from v to $u\}$
$B = \{u \in V | \exists$ even length path from v to $u\}$

We have a bipartite graph if A and B are disjoint.

Proof

Only if: trivial

Start at a node v in one part, say V, the cycle must be like leaving V, entering V, ...
Also the cycle must end at v, so the cycle must end with "entering V".All paired up, even length.

No odd-length cycle $\implies$ bipartite:

Different connected components does not influence each other, just look at one first

Pick one arbitrary vertex v, split all vertices into two groups
$A = \{u \in V | \exists$ odd length path from v to $u\}$
$B = \{u \in V | \exists$ even length path from v to $u\}$

We have a bipartite graph if A and B are disjoint.
What if a vertex in both sets?

Proof

Only if: trivial

Start at a node v in one part, say V, the cycle must be like leaving V, entering V, ...
Also the cycle must end at v, so the cycle must end with "entering V".All paired up, even length.

No odd-length cycle $\implies$ bipartite:

Different connected components does not influence each other, just look at one first

Pick one arbitrary vertex v, split all vertices into two groups
$A = \{u \in V | \exists \text{ odd length path from } v \text{ to } u\}$
$B = \{u \in V | \exists \text{ even length path from } v \text{ to } u\}$

We have a bipartite graph if A and B are disjoint.
What if a vertex in both sets? Odd length cycle!

Proof

Only if: trivial

Start at a node v in one part, say V, the cycle must be like leaving V, entering V, ...
Also the cycle must end at v, so the cycle must end with "entering V".All paired up, even length.

No odd-length cycle $\implies$ bipartite:

Different connected components does not influence each other, just look at one first

Pick one arbitrary vertex v, split all vertices into two groups
$A = \{u \in V | \exists$ odd length path from v to $u\}$
$B = \{u \in V | \exists$ even length path from v to $u\}$

We have a bipartite graph if A and B are disjoint.
What if a vertex in both sets? Odd length cycle! Contradiction

What have we done?!

Graphs!

What have we done?!

Graphs!

Eulerian tour:

What have we done?!

Graphs!

Eulerian tour: DNA sequence reconstructing

What have we done?!

Graphs!

Eulerian tour: DNA sequence reconstructing

Coloring:

What have we done?!

Graphs!

Eulerian tour: DNA sequence reconstructing

Coloring: Cellular tower frequency assignment

What have we done?!

Graphs!

Eulerian tour: DNA sequence reconstructing

Coloring: Cellular tower frequency assignment

Trees:

What have we done?!

Graphs!

Eulerian tour: DNA sequence reconstructing

Coloring: Cellular tower frequency assignment

Trees: Immense applications.........

What have we done?!

Graphs!

Eulerian tour: DNA sequence reconstructing

Coloring: Cellular tower frequency assignment

Trees: Immense applications.........

Modeling reality:

What have we done?!

Graphs!

Eulerian tour: DNA sequence reconstructing

Coloring: Cellular tower frequency assignment

Trees: Immense applications.........

Modeling reality:

Internet?

What have we done?!

Graphs!

Eulerian tour: DNA sequence reconstructing

Coloring: Cellular tower frequency assignment

Trees: Immense applications.........

Modeling reality:

Internet? Giant directed graph

What have we done?!

Graphs!

Eulerian tour: DNA sequence reconstructing

Coloring: Cellular tower frequency assignment

Trees: Immense applications.........

Modeling reality:

- Internet? Giant directed graph
- Dark net?

What have we done?!

Graphs!

Eulerian tour: DNA sequence reconstructing

Coloring: Cellular tower frequency assignment

Trees: Immense applications.........

Modeling reality:

Internet? Giant directed graph
Dark net? A separate connect component!

What have we done?!

Graphs!

Eulerian tour: DNA sequence reconstructing

Coloring: Cellular tower frequency assignment

Trees: Immense applications.........

Modeling reality:

- Internet? Giant directed graph
- Dark net? A separate connect component!
- ......