Class starts after the music

Jacques Offenbach, Giovanni Sollima, Andrea Noferini – Duos for 2 Cellos, Op. 54 No.1: III(2023)

requested by Ian Zhang (TA-of-CM6)

I love listening to classical music and performing with others on the piano and cello. I enjoy competing in various algorithmic programming competitions.

Duke

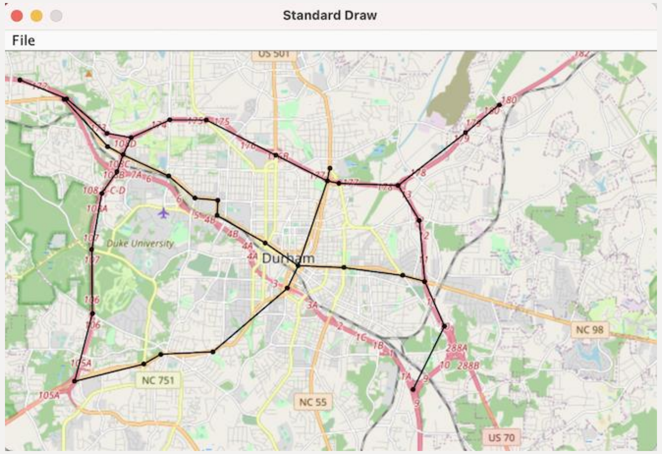
Logistic Bulletin Board

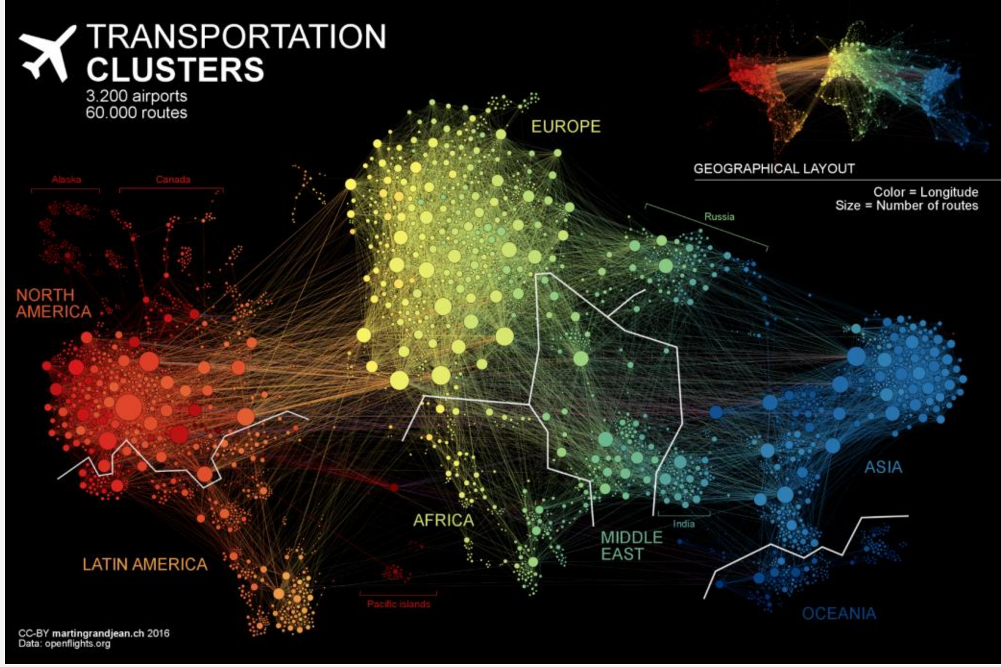
- Mid-semester survey due 2/29 midnight

CS230 Spring 2024
Module 06: Graph Fundamentals
(Induction on) Graphs

Duke

Why graphs?





- Models just about anything in the world
  - Road network: navigation
  - Social network: (mis-)information spread
  - Electoral districts: redistricting
  - Matching workers to jobs
- Here we starts the “fun” part of CS230

Graphs in CS201/230

- CS201 focuses on **trees/graphs as data structures**
  - detailed implementations in a Java context
  - simple, step-by-step algorithms that operate on the data structures (tree traversal, DFS, BFS…)
- CS230 focuses on **trees/graphs as abstract ideas**
  - directed, undirected, self-loops… don't care about how to implement
  - reasoning about properties of trees/graphs

Focus of CM6

- In CM6, we will devote most of our time and energy on:
  - Graph/tree properties (as mathematical facts, not as algorithms)
  - How to formally reason about graph/tree properties

Is Dijkstra's algorithm guaranteed to be correct? (Informal)

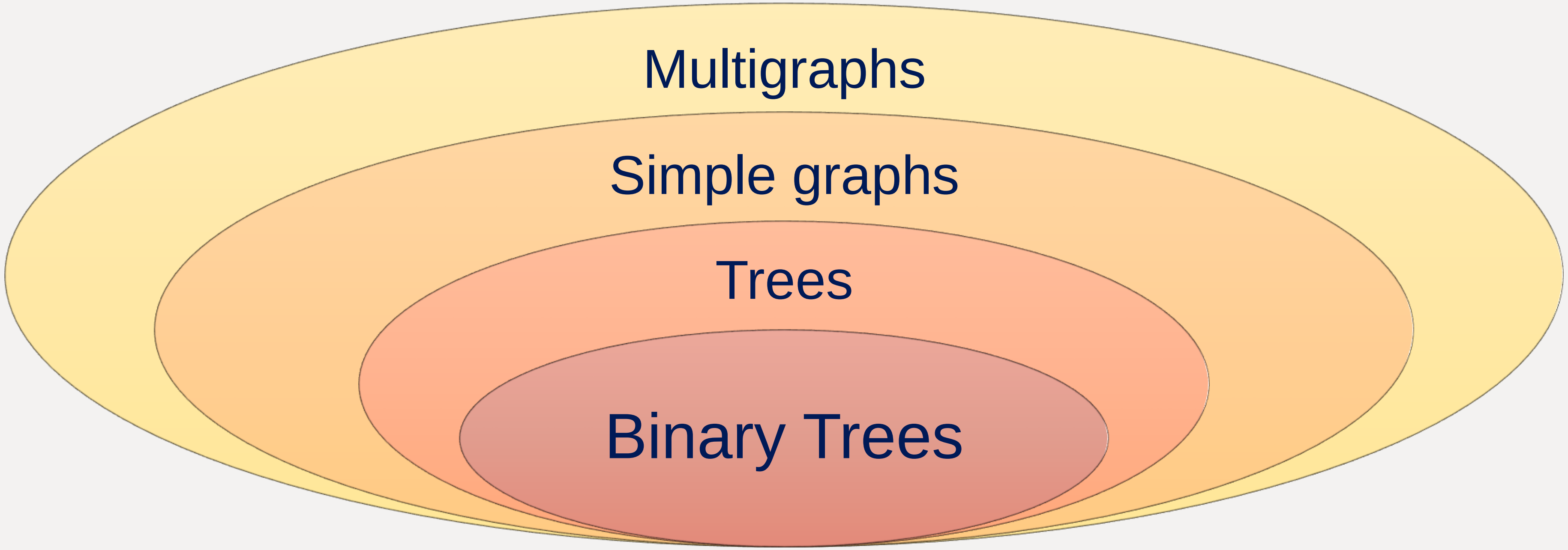
- **Claim.** Distance is correct shortest path distance for all nodes *explored* so far, and shortest path distance *through explored nodes* for all others.

- Formal proof is *by induction,* see Compsci 230.
  - Assume the property is true up to some point in the algorithm, then…
  - Consider the next node we explore:

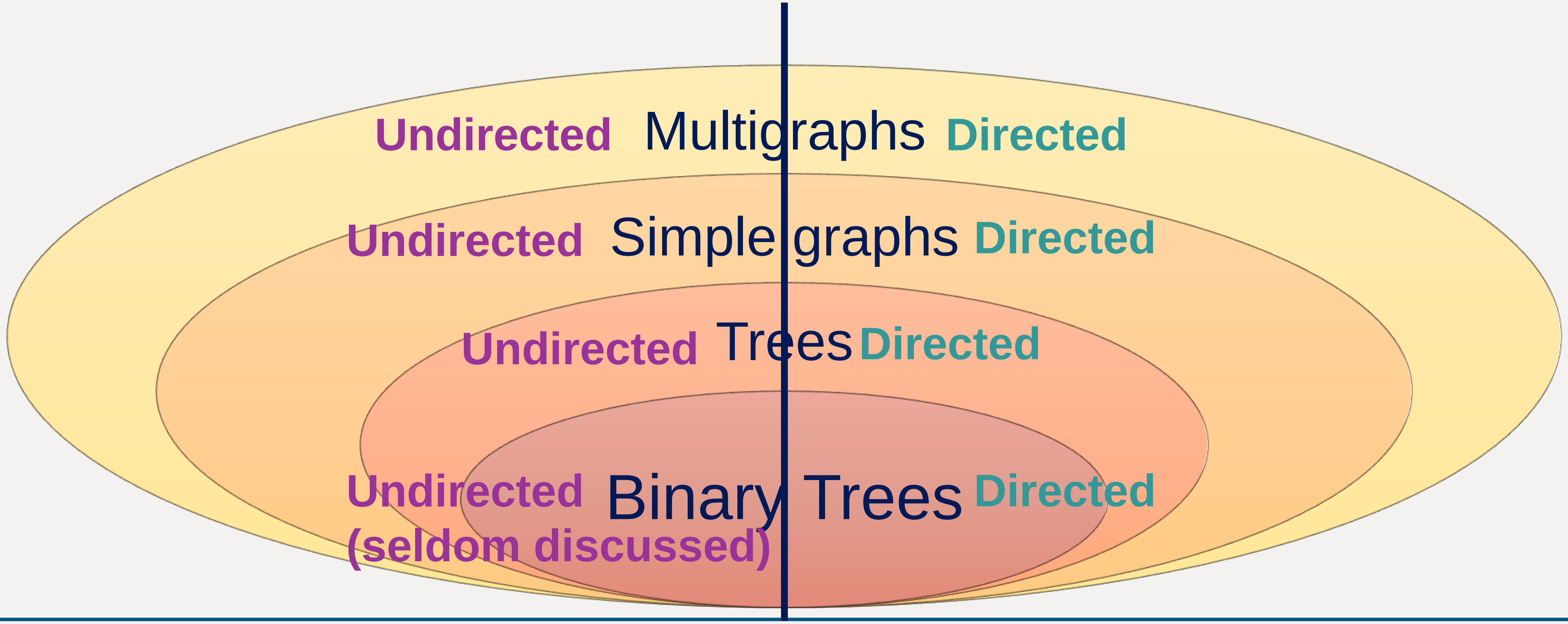
4/12/23 Compsci 201, Spring 2023, L24: Shortest Paths 35

Terminology Musing

- Think about these terms as **sets**.
- **Graphs with stronger properties** are subsets of graphs with weaker properties
- **Trees** are also multigraphs
- **Binary Trees** are also simple graphs



Undirected vs. directed



1 1 point

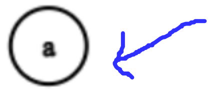
Consider a graph G with just one edge l

What kind of graph is G? (Select all that

- [ ] undirected multigraph
- [x] directed multigraph
- [ ] simple undirected graph
- [x] directed graph
- [ ] undirected tree
- [x] directed tree
- [ ] undirected binary tree
- [x] directed binary tree

2 1 point

Consider an even simpler graph H with just one vert



What kind of graph is H? (Select all that apply)

- [x] undirected multigraph
- [x] directed multigraph
- [x] simple undirected graph
- [x] directed graph
- [x] undirected tree
- [x] directed tree
- [x] undirected binary tree
- [x] directed binary tree

Downward closed graph properties

- Many graph properties are "downward closed":
  - Given that graph G has a property X
  - Then all subgraphs $G' \subseteq G$ retain the property X
  - When we write $G' \subseteq G$ we actually mean $G' = (V', E'), V' \subseteq V, E' \subseteq E$

  - Examples of downward closed graph properties:
    - *Simple, Forest, Planar, Acyclic*

$\{v\} = V_1 \qquad V_2 = \emptyset$

Downward closed graph properties

Definition 10.43. *A simple graph G is called* **bipartite** *if the vertex set V can be partitioned into two disjoint ~~nonempty~~ sets V_1 and V_2 such that every edge connects a vertex in V_1 to a vertex in V_2.*

Put another way, no vertices in V_1 are connected to each other, and no vertices in V_2 are connected to each other.

- Bipartiteness is downward closed… until things become weird
- For this reason, we will allow graphs of 0 or 1 vertices to be bipartite *(although AIDMA doesn't agree)*

“Upward closed” graph properties

- Some other graph properties are instead “upward closed” provided that *we only add edges and not vertices:*
  - Given that graph $G = (V, E)$ has a property X
  - Then all $G' = (V, E')$ s.t. $E \subseteq E'$ retain the property X

  - Examples of upward closed graph properties:
    - *Connected, Hamiltonian, Cyclic*

Usefulness of closed graph properties

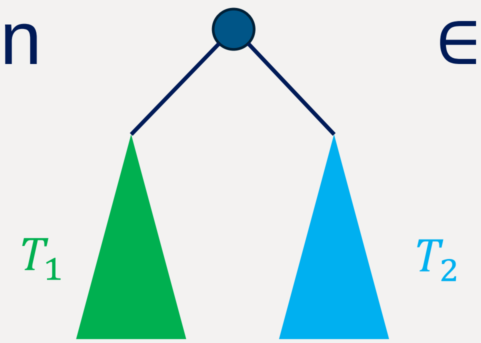
- Knowing certain graph properties are closed is useful for proving theorems *for the whole family of graphs*
- **Theorem.** Any graph with property X satisfies statement Y.
- **Proof sketch:**
  - **Base Case(s).** The “smallest” graphs with property X satisfies statement Y.
  - **Induction Step.** Consider now an arbitrary graph G with property X.
    - Remove one vertex (or one edge) from G.
    - The resulting graph is a “smaller” graph G' that satisfies statement X.
    - We assume (implicit hypothesis) that G' satisfies statement Y.
    - We then prove that adding such vertex/edge back retains statement Y.

Wait, we can do *that*?

- The "proof sketch" in the previous slide is a template of **structural induction** (in the context of graphs).

- If weak/strong inductions are driven by the set of natural numbers, structural inductions are driven by **recursively defined structures**.

Recursively defined structures

- Same idea, but the set now contains objects, not just numbers
- Example 3. The set of binary trees, $\mathcal{T}$, can be defined as:
  - **Base Case:** $T = (\emptyset, \emptyset) \in \mathcal{T}$ (the "empty tree")
  - **Constructor Case:** If $T_1, T_2 \in \mathcal{T}$, then  $\in \mathcal{T}$

Inducting on recursively defined structures

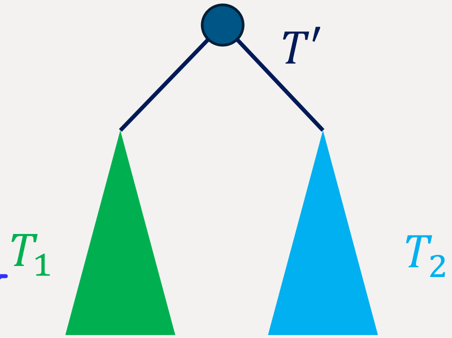
- (Structural induction template) Given a recursively defined set S
- **Goal.** We want to show $\forall(s \in S)[P(s)]$ for a predicate P
- **Proof:**
  - **Base Case(s).** Prove $P(s)$ for all base cases in the definition of S.
  - **Inductive Hypothesis (USUALLY IMPLICIT).**
    Assume $P(s)$ for all elements of S in the constructor case.
  - **Induction Step.** Prove $P(s)$ for each of subcases of the constructor case.
  - Sometimes called Basis Step and Recursive Step

Inducting on recursively defined structures

- Consider again the set of binary trees, $\mathcal{T}$
- **Theorem.** $\forall T \in \mathcal{T}\ [n(T) \leq 2^{h(T)+1} - 1]$ $n(T)$= # of vertices in T $h(T)$=height of T
- **Proof** (contains informal language):
  - **Base Case(s).** For $T = (\emptyset, \emptyset)$ $\text{LHS} = 0$ $\text{RHS} = 1$
  - **Induction Step.** Consider two binary trees T_1 and T_2 such that $n(T_1) \leq 2^{h(T_1)+1} - 1$ and $n(T_2) \leq 2^{h(T_2)+1} - 1$. Then for the new tree T':

$$\begin{aligned}\text{LHS} = n(T_1) + n(T_2) + 1 &\leq (2^{h(T_1)+1} - 1) + (2^{h(T_2)+1} - 1) + 1 \\ &\leq 2^{\max(h(T_1)+1, h(T_2)+1)} + 2^{\max(h(T_1)+1, h(T_2)+1)} - 1 \\ &= 2^{\max(h(T_1)+1, h(T_2)+1)+1} - 1 \\ &\leq 2^{h(T')+1} - 1 = \text{RHS}\end{aligned}$$

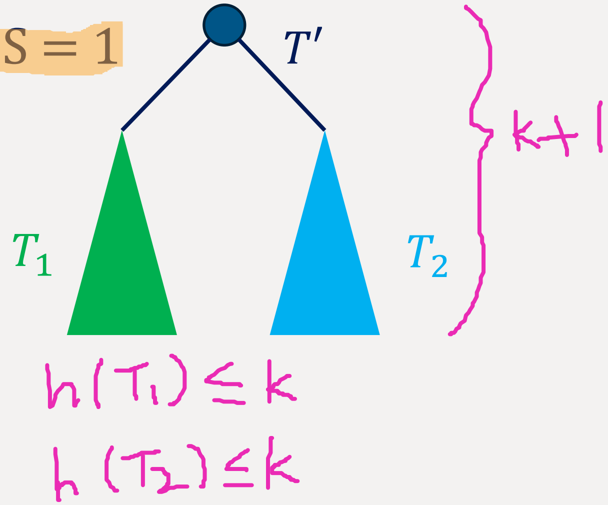
This inequality holds as long as the two subtrees T_1 and T_2 are not both empty. But if the two subtrees are both empty, then the entire tree is just one vertex - we can manually verify that LHS=RHS=1 for that case.



Inducting on recursively defined structures

- The same proof can instead "induct on $h(T)$"
- **Theorem.** $\forall T \in \mathcal{T}\ [n(T) \leq 2^{h(T)+1} - 1]$ $n(T)$= # of vertices in T $h(T)$=height of T
- **Proof** (contains informal language):
  - **Base Case(s).** For **all trees T with $h(T) = 0$** we have $\text{LHS} \leq 1$, $\text{RHS} = 1$
  - **(Strong) Inductive Hypothesis.** Assume the result holds for **all trees T with $h(T) \leq k$.**
  - **Induction Step.** Consider an arbitrary tree T' with height $k+1$. It can be written as the root plus two binary subtrees T_1 and T_2.
  - Then for the new tree T':

$$\text{LHS} = n(T_1) + n(T_2) + 1 \leq (2^{k+1} - 1) + (2^{k+1} - 1) + 1$$
$$= 2^{k+2} - 1 \leq 2^{h(T')+1} - 1 = \text{RHS}$$



Inducting on recursively defined structures

- The proof on the previous slide was just a “regular strong induction”:
- **Theorem.** $\forall \boldsymbol{n} \in \mathbb{N}\ [\boldsymbol{P}(\boldsymbol{n})]$
  where $\boldsymbol{P}(\boldsymbol{n}) \coloneqq \forall T \in \mathcal{T}\ [h(T) \leq n \rightarrow n(T) \leq 2^{h(T)+1} - 1]$

- Neither approach is “strictly better” than the other
- The takeaway here is that we can directly induct on the structure (like in the first proof) and not rely on any variable (like in the second proof)

weak induction is structural induction

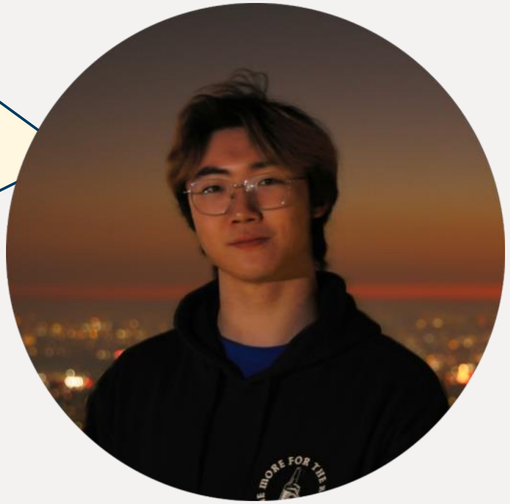
- Example 6. The set of nonnegative integers, $\mathbb{N}$, can be “defined” as:
  - **Base Case:** $0 \in \mathbb{N}$
  - **Constructor Case:** If $x \in \mathbb{N}$, then $x + 1 \in \mathbb{N}$.
- This is somewhat like circular reasoning: addition does not really have a meaning without defining $\mathbb{N}$ first
- But this shows weak induction is a special case of structural induction on the recursively defined set $\mathbb{N}$.
  - So anything achievable by weak induction is also achievable by structural induction
  - Is the opposite true? We will revisit this next week

Class starts after this song

Song Dongye – Anhe Bridge (2013)

requested by Shawn Ma (TA-of-CM6)

I am a Computer Science and Biology double major with a minor in Asian an Middle Eastern Studies. Outside of coursework, I am currently a student consultant at a startup by Duke alum and am involved in Lambda Phi Epsilon. Feel free to reach out to me about poker and/or anime!



Duke

Logistic Bulletin Board

- Mid-semester survey:
  - Some started but “did not finish” according to Qualtrics+Violet
  - Please complete it by end of Sunday if that’s the case
- Elective modules:
  - Completely async EMs (E and F) released in Canvas
  - So you have 2 full months to play with them
  - Hybrid ones (A and D) next Friday and then in Canvas
  - Sync ones (B and C) in April

4	5	6	7	8	9	10
CM6:Graph Fundamentals (Recitation I)	CM6 reading/Canvas quiz Part II due	CM6:Graph Fundamentals (3)		CM6 Assignment Due ON PAPER EMa/d: Graph Applications in AI and Robotics	Asynchronous EMs Release	
11	**12**	**13**	**14**	**15**	**16**	**17**
						CM6 Gradescope Assignment Actually Due
18	**19**	**20**	**21**	**22**	**23**	**24**
	CM7 reading/Canvas quiz due			CM7:Combinatorics (1) ONLINE		

No recitations
Recitations are converted into consulting hours
Some will be online
Graders start grading CM6 on 3/18 evening

Duke

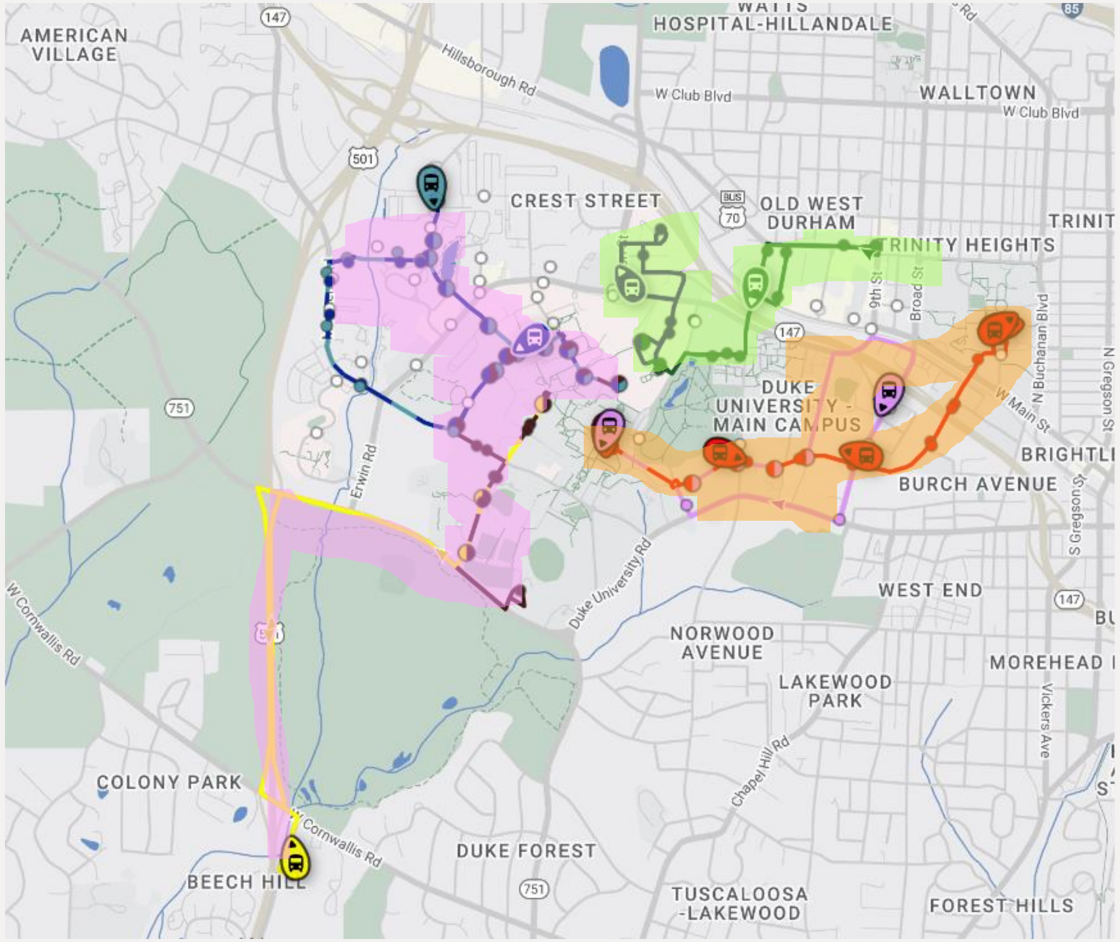
CS230 Spring 2024
Module 06: Graph Fundamentals
Graph Topics
(connectivity, colorability, matching)

Duke

Connectivity

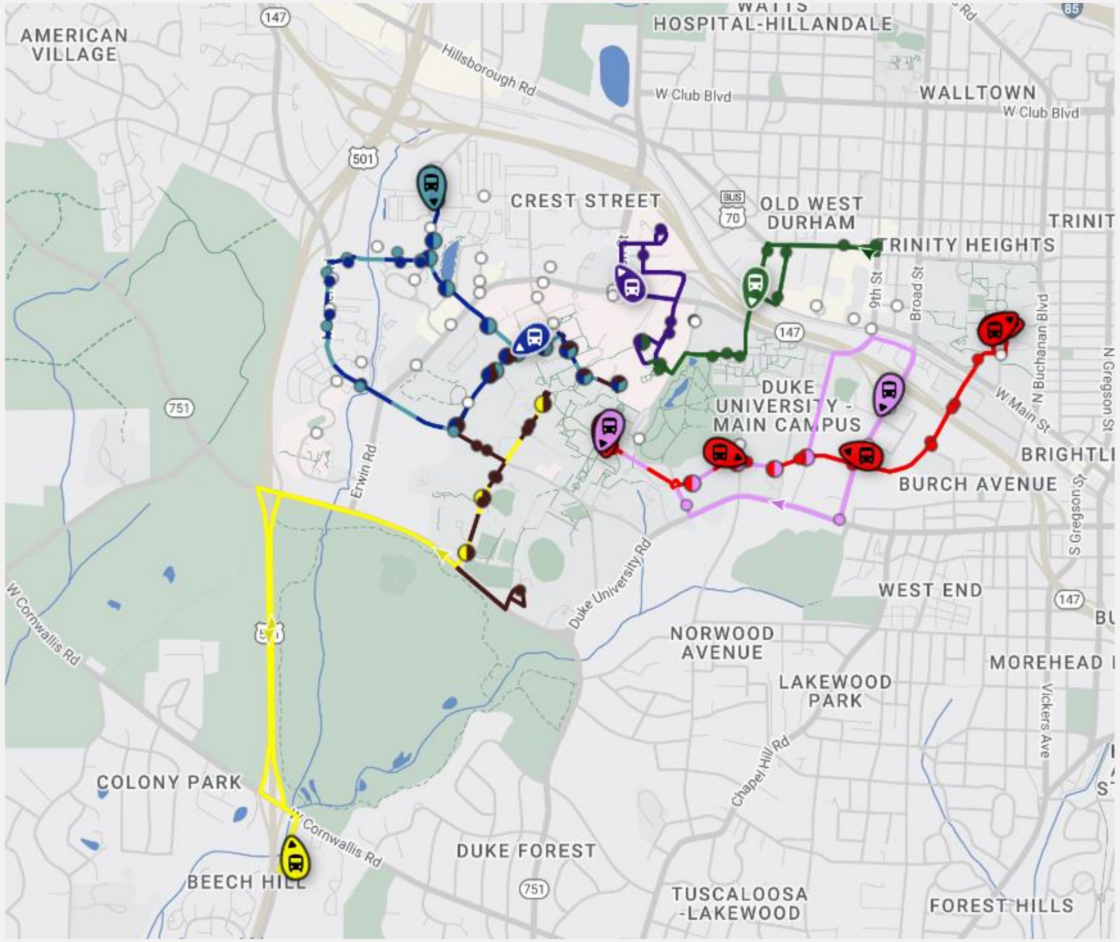
- “Graph of Duke bus network” during non-peak time of a regular weekday

- one vertex for each bus stop
- (undirected) edges between two consecutive stops on a route



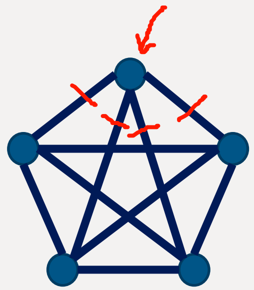
Modeling the world

- What does the graph and its connectivity really capture?
- Should we model the graph differently?
    - Depend on what we care about



Duke

k –connectivity



The complete graph K_n with n vertices is $(n-1)$ –edge connected and also $(n-1)$ –vertex connected

- A graph G is said to be k –edge connected if G remains connected after the removal of (any) $k-1$ edges.
  - It takes at least k removals to disconnect the graph
- A graph G is said to be k –vertex connected if G remains connected after the removal of (any) $k-1$ vertices.
  - Remember edges can only exist between pairs of vertices, so removing a vertex also removes all edges incident to the vertex.

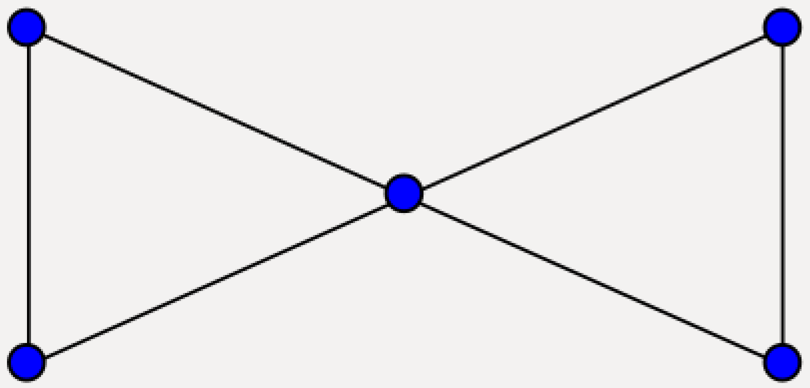
PI: k-connectivity

1 1 point

Which of the following implications are true for all $k \geq 2$?

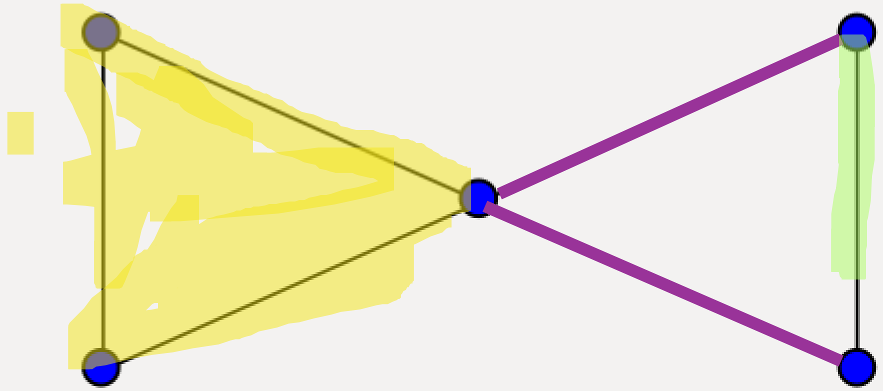
- [ ] If a graph is k-edge connected, then it is also k-vertex connected
- [x] If a graph is k-edge connected, then it is also $(k-1)$-edge connected
- [x] If a graph is k-vertex connected, then it is also $(k-1)$-vertex connected

The butterfly graph



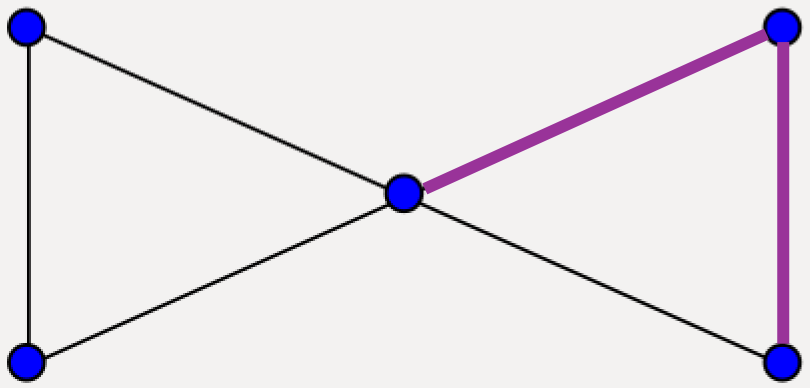
- 2 –edge connected
- 1 –vertex connected
- Not 2 –vertex connected
- Continued in recitation

Edge cut and vertex cut



- In a graph $G = (V, E)$,
  a subset of edges $E' \subseteq E$
  is an edge cut
  if $G' = (V, E \backslash E')$ is disconnected.
- Can define vertex cuts similarly

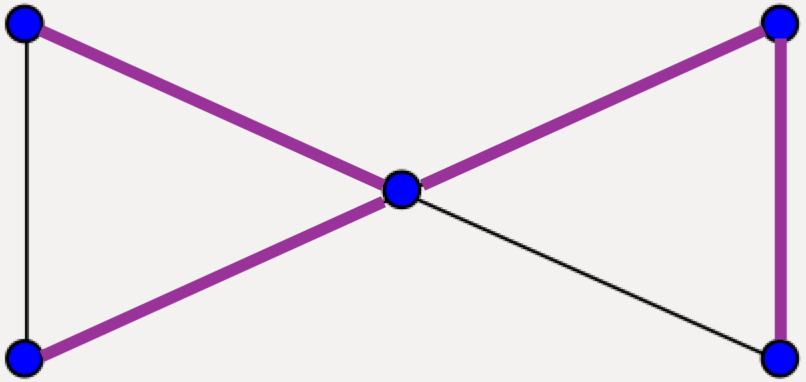
Edge cut and vertex cut



- In a graph $G = (V, E)$,
  a subset of edges $E' \subseteq E$
  is an edge cut
  if $G' = (V, E \backslash E')$ is disconnected.

- There are multiple minimum edge cuts (in terms of $|E'|$)

Edge cut and vertex cut

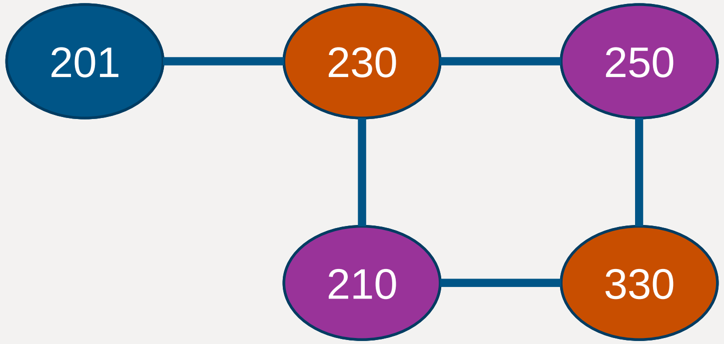


- In a graph $G = (V, E)$,
  a subset of edges $E' \subseteq E$
  is an edge cut
  if $G' = (V, E \backslash E')$ is disconnected.

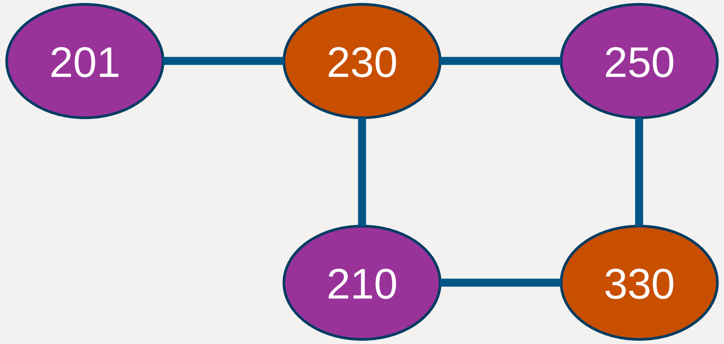
- Non-minimum edge cut

Coloring

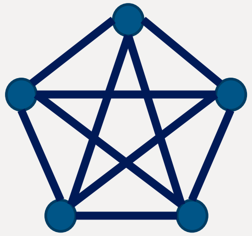
- “Final exam scheduling problem”



- one vertex for each class
- an edge between two vertices if there are students taking both classes
- each “color” is a final exam slot



k –colorability



The complete graph K_n with n vertices is n –colorable but not $(n-1)$ –colorable, so $\chi(K_n) = n$

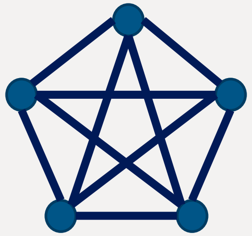
- A graph $G = (V, E)$ is said to be k –(vertex) colorable if there is a function $f: V \to \{1,2,\ldots,k\}$ such that for every edge $(u, v) \in E$ we have $f(u) \neq f(v)$.
- In words: we can color the vertices using k colors, such that the two endpoints of each edge have different colors.
- The minimum such k is called the chromatic number $\chi(G)$
  - Can define k –edge colorable similarly (swap vertices and edges)

PI: k-colorability

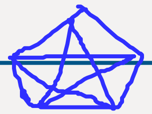
1 1 point

Which of the following implications are true for all k?

- [x] If a graph is k-colorable, then it is also $(k+1)$-colorable
- [ ] If a graph is k-colorable, then it is also $(k-1)$-colorable
- [ ] If a graph is k-vertex connected, then it is also k-colorable
- [ ] If a graph is k-colorable, then it is also k-vertex connected



The complete graph K_n with $n > 2$ vertices is 2 –vertex connected but not 2 –colorable



Inducting on graphs

- Consider again the set of undirected (simple) graphs, $\mathcal{G}$
- **Theorem.** Define the maximum degree of a graph $G \in \mathcal{G}$ as $\Delta(G) \coloneqq \max_{v \in V} \deg(v)$. Then every G is $(\Delta(G) + 1)$-colorable.
- **Proof**:
  - **Base Case(s).** For $G = (\emptyset, \emptyset)$, $\Delta(G) = 0$ and G is indeed 1-colorable
  - **Induction Step. What should we do here?**

Never attempt a structural induction without thinking about the recursive definition first

Duke

Recursively defined structures

- Example 4. The set of (undirected) graphs, $\mathcal{G}$, can be defined as:
  - **Base Case:** $G = (\emptyset, \emptyset) \in \mathcal{G}$ (the "empty graph")
  - **Constructor Case:** If $G = (V, E) \in \mathcal{G}$, then:
    - $G' = (V \cup \{v\}, E) \in \mathcal{G}$ ("add a vertex")
    - $G' = (V, E \cup \{e\}) \in \mathcal{G}$ where $e = (u, v), u, v \in V$ ("add an edge")

An alternative recursive definition

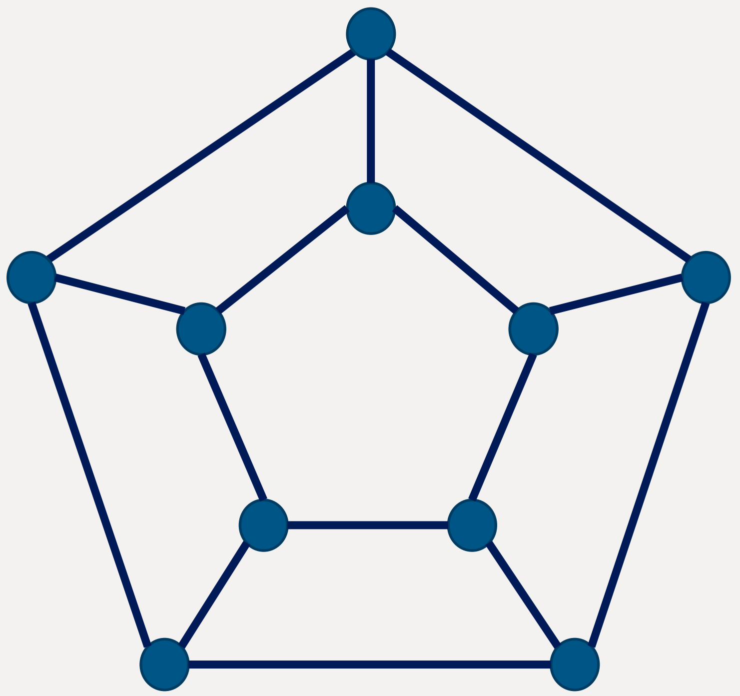
- The set of (undirected) simple graphs, $\mathcal{G}$, can be alternatively defined as:
  - **Base Case:** $G = (\emptyset, \emptyset) \in \mathcal{G}$ (the “empty graph”)
  - **Constructor Case:** If $G = (V, E) \in \mathcal{G}$, then:
    - $G' = (V \cup \{v\}, E \cup E_v) \in \mathcal{G}$ (“add a vertex and all its incident edges”)
      - Every edge in E_v must be between v and some existing vertex

Inducting on graphs

- **Theorem.** Define the maximum degree of a graph $G \in \mathcal{G}$ as $\Delta(G) \coloneqq \max_{v \in V} \deg(v)$. Then every G is $(\Delta(G) + 1)$-colorable.
- **Induction Step.** (contains informal language):
  - Assume G is $(\Delta(G) + 1)$-colorable. Consider $G' = (V \cup \{v\}, E \cup E_v)$ where $|E_v| \leq \Delta(G')$.
  - Consider any proper $(\Delta(G) + 1)$-coloring of G (let's call it $\mathcal{C}(G)$). By definition we have $\Delta(G) \leq \Delta(G')$, which means $\mathcal{C}(G)$ uses at most $\Delta(G') + 1$ colors.
  - Every neighbor of v has a color in $\mathcal{C}(G)$. Since there are $|E_v| \leq \Delta(G')$ neighbors of v, they use up at most $|E_v| \leq \Delta(G')$ colors. Since we have $\Delta(G') + 1$ colors, there is at least one spare color for v.
  - This in combination with $\mathcal{C}(G)$ gives a proper $(\Delta(G') + 1)$-coloring of G'.

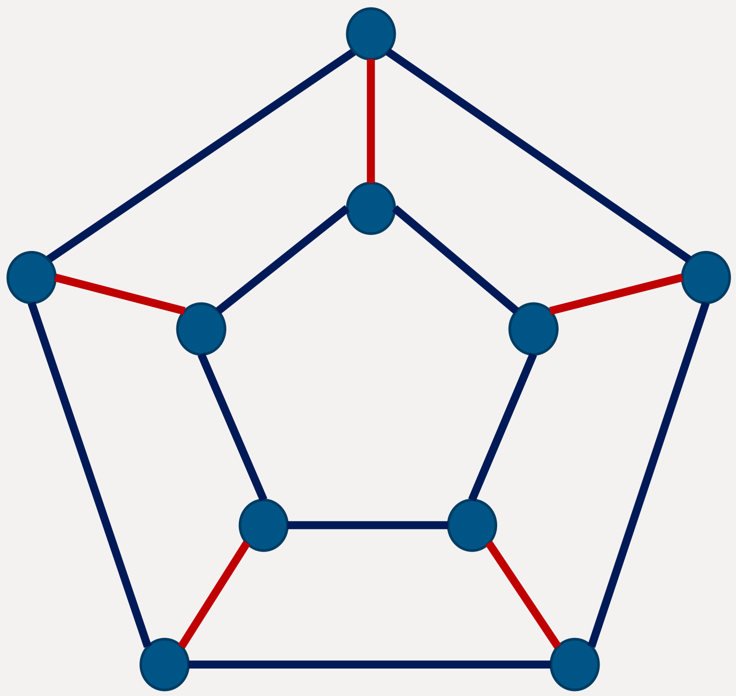
Matching

- A matching of a graph $G = (V, E)$ is an edge subset $M \subseteq E$ such that in the subgraph $G' = (V, M)$ we have $\deg(v) \leq 1$ for all $v \in V$.
- In words, each vertex is either matched to another vertex or unmatched.
- Perfect matching if $\deg(v) = 1\ \forall v \in V$



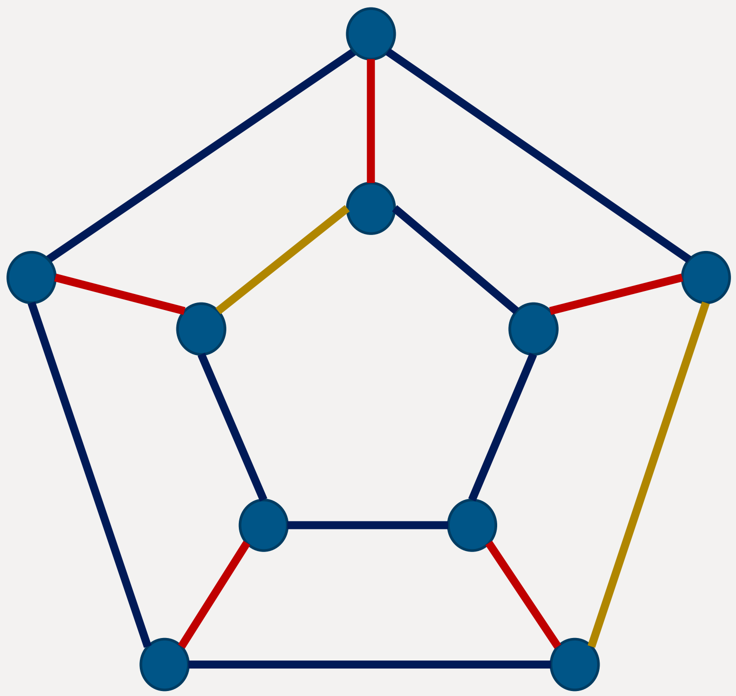
Matching

- {Red edges} is a perfect matching



Matching

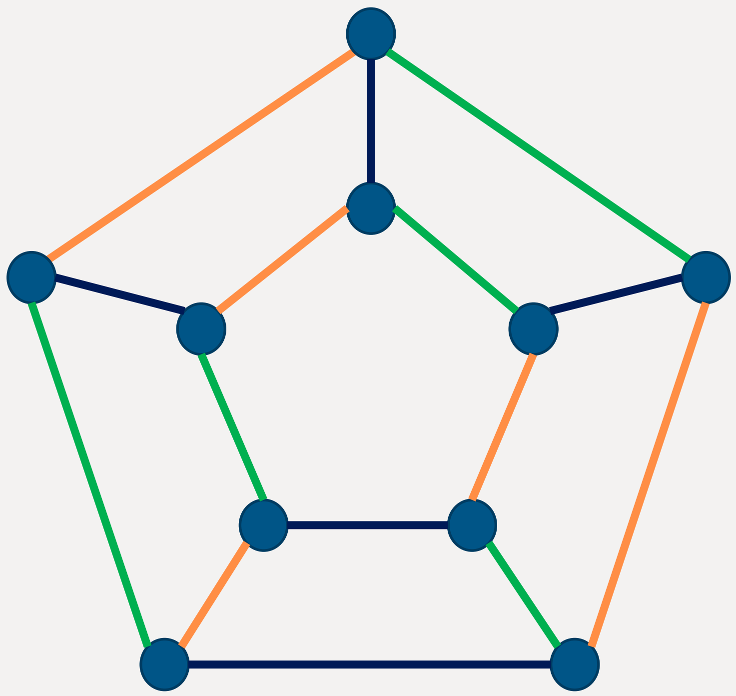
- {Red edges} is a perfect matching
- {Gold edges} is a matching but not perfect
- {Purple edges} is a matching as well
  - what, not seeing the edges? There's no edge in this matching at all



Duke

Matching

- Perfect matching
- Yet another perfect matching that is **disjoint** with the orange one
- The uncolored edges form a third disjoint perfect matching



Bipartite Matching

- Matching, but with the underlying graph already bipartite:
$$G = (L \cup R, E)$$
- Matching TAs/residents/interns with positions
- Matching jobs with machine cycles/slots
- … but **NOT**: ~~matching men with women~~
- These applications often involves preferences and some additional ideas of what an *optimal* matching looks like (stable, fair, …)
- We will just discuss whether a perfect matching exists

Duke

Hall's theorem (for the special case $|L| = |R|$)

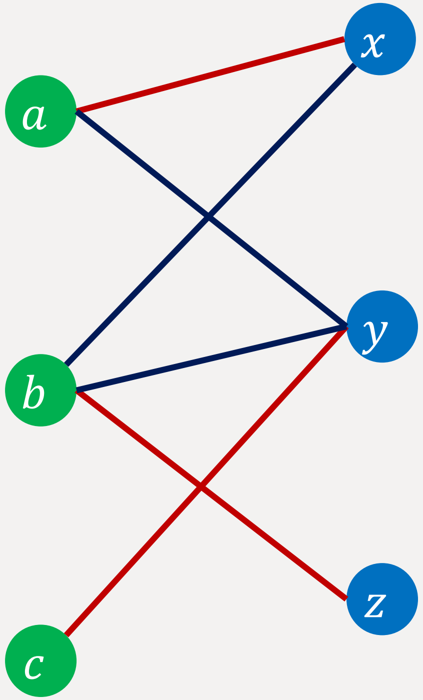
- Theorem.
  A bipartite graph $G = (L \cup R, E)$ has a perfect matching
  if and only if

  Hall's condition

  for any subset $S \subseteq L$, the size of the image of S w.r.t. E, $N(S)$, is at least as large as $|S|$.
  - $N(S)$ is just the set of all vertices in R that are connected to some vertices in L by edges in E

Terminology Practice

S	$\lvert S\rvert$	$N(S)$	$\lvert N(S)\rvert$	Hall's condition satisfied?
$\{a\}$	1	$\{x, y\}$	2	Yes
$\{b\}$	1	$\{x, y, z\}$	3	Yes
$\{c\}$	1	$\{y\}$	1	Yes
$\{a, b\}$	2	$\{x, y, z\}$	3	Yes
$\{a, c\}$	2	$\{x, y\}$	2	Yes
$\{b, c\}$	2	$\{x, y, z\}$	3	Yes
$\{a, b, c\}$	3	$\{x, y, z\}$	3	Yes



Proof of Hall's theorem (1)

- Part 1. perfect matching exists $\Rightarrow$ Hall's condition
  - Simple prove by contradiction: assume Hall's condition doesn't hold
  - Then there exists a subset $S \subseteq L$ such that $|N(S)| < |S|$
  - But if a perfect matching exists, every vertex in S needs to match with a distinct vertex in $N(S)$, which means $|N(S)| \geq |S|$, a contradiction

Proof of Hall's theorem (2)

- Part 2. Hall's condition ⇒ perfect matching exists
- The set of balanced bipartite graphs, $\mathcal{B}$, can be alternatively defined as:
  - **Base Case:** $G = (\emptyset \cup \emptyset, \emptyset) \in \mathcal{B}$ (the "empty graph")
  - **Constructor Case:** If $G = (L \cup R, E) \in \mathcal{B}$, then:
    - $G = ((L \cup \{l\}) \cup (R \cup \{r\}), E \cup E') \in \mathcal{B}$
      ("adding one vertex to each side, then add some edges")
    - … this is hard to work with! We will instead resort to a (weak) induction

Proof of Hall's theorem (2)

- Part 2. Hall's condition ⇒ perfect matching exists for all $|L| = |R| = n$
- **Proof**:
  - **Base Case.** $n = 0$. For $G = (\emptyset \cup \emptyset, \emptyset)$, $\emptyset$ is a perfect matching (yes it is)
  - **Inductive Hypothesis.** Assume the theorem holds for $n = k$ for $k \geq 0$.
  - **Induction Step.** Consider a graph $G = (L \cup R, E)$ with $|L| = |R| = k + 1$.
    - Case 1. Hall's condition is loosely satisfied (equation never holds).
    - Case 2. Hall's condition is tightly satisfied (equation holds for some S).

Proof of Hall's theorem (2-1)

- Part 2-1. Hall's condition loosely satisfied ⇒ perfect matching exists
- **Induction Step.** Consider a graph $G = (L \cup R, E)$ with $|L| = |R| = k + 1$ such that for all $S \subseteq L$ we have $|N(S)| > |S|$.
  - Pick an arbitrary left vertex $l \in L$ and match l with an arbitrary right vertex $r \in N(\{l\})$
  - Consider the remainder graph $G' = (L - \{l\} \cup R - \{r\}, E')$ with k vertices each side.
  - Now for all $S' \subseteq L - \{l\}$, we have $|N'(S')| \geq |N(S')| - 1 \geq |S'|$.
  - In other words, Hall's condition is still satisfied (not necessarily loosely) for G'.
  - Any perfect matching of G' plus (l, r) is a perfect matching for G!

Proof of Hall's theorem (2-2 - *skeleton*)

- Part 2-2. Hall's condition strictly satisfied ⇒ perfect matching exists
- **Induction Step.** Consider a graph $G = (L \cup R, E)$ with $|L| = |R| = k + 1$ such that for all $S \subseteq L$ we have $|N(S)| \geq |S|$ and the equality holds for at least one S.
  - Find a perfect matching between $|S|$ and $|N(S)|$
  - Consider the rest of the graph without $|S|$ and $|N(S)|$
    - We can show Hall's condition still holds for this remainder graph
    - Therefore there is a perfect matching for this remainder graph; combine the two parts gives a perfect matching for the entire graph