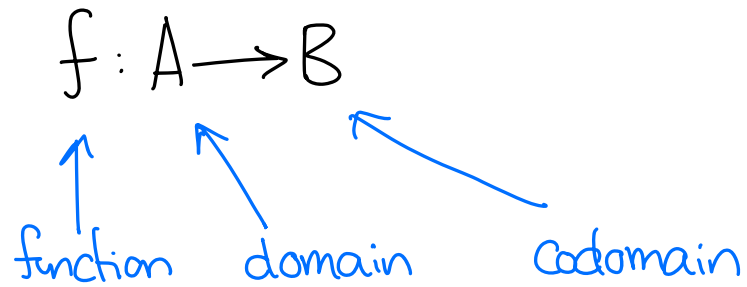


Last time...



$\forall b \in B$ b is hit

injective	≤ 1 time
surjective	≥ 1 time
bijective	$= 1$ time

Pigeonhole Principle: if there are more pigeons than pigeonholes, some hole will have at least 2 pigeons

Cantor's Theorem: \forall sets A , \nexists a surjection from A to $\mathcal{P}(A)$

→ there is a problem that computers can't solve!

CSC 236 Lecture 2: Graphs

Harry Sha

May 17, 2023

Today

Definitions

Modelling with Graphs

Problem 1. Matching

Problem 2. Shortest Path

Problem 3. The Traveling Salesman

Trees - A special type of graph

Problem 4. Minimum Spanning Tree

Annoucments

- My office hours will be in BA 2270 from 3-5 PM on Wednesday.
- A list of (optional) readings is added to the schedule on the website. Where the lecture slides and the readings differ (which won't happen often), you should follow the lecture slides.
- There are three versions of the lecture slides now
 1. Clean version.
 2. One with scribbles from class.
 3. One with more complete notes. (This file will be updated throughout the semester, eventually containing all lectures from the course).
- I will upload a preliminary version of the clean version of the slides to the website before class so you can take notes.

The goal

The goal of today's lecture is to have you see graphs everywhere in the world.

Definitions

Modelling with Graphs

Problem 1. Matching

Problem 2. Shortest Path

Problem 3. The Traveling Salesman

Trees - A special type of graph

Problem 4. Minimum Spanning Tree

Definitions

A **graph** $G = (V, E)$ is a pair of sets (V, E) , where V is a set of vertices and E is a set of pairs of vertices.

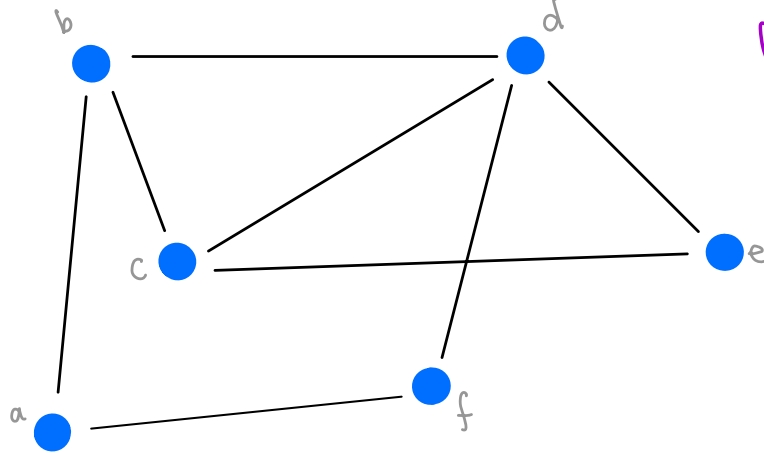
If E is a set of unordered pairs, the graph is called **undirected** and if the E is a set of ordered pairs, the graph is called **directed**.

Examples

order
doesn't
matter \rightarrow

$\{ \}$ () \leftarrow order matter

$V = \{a, b, c, d, e, f\}$

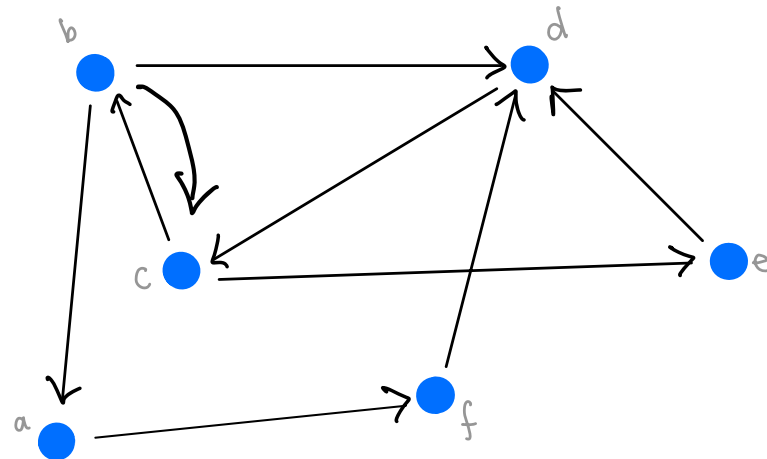


Undirected

$E = \{ \{b,d\}, \{a,b\}, \{d,c\}, \{d,e\}, \{e,c\}, \{a,f\}, \{b,c\}, \{f,d\} \}$

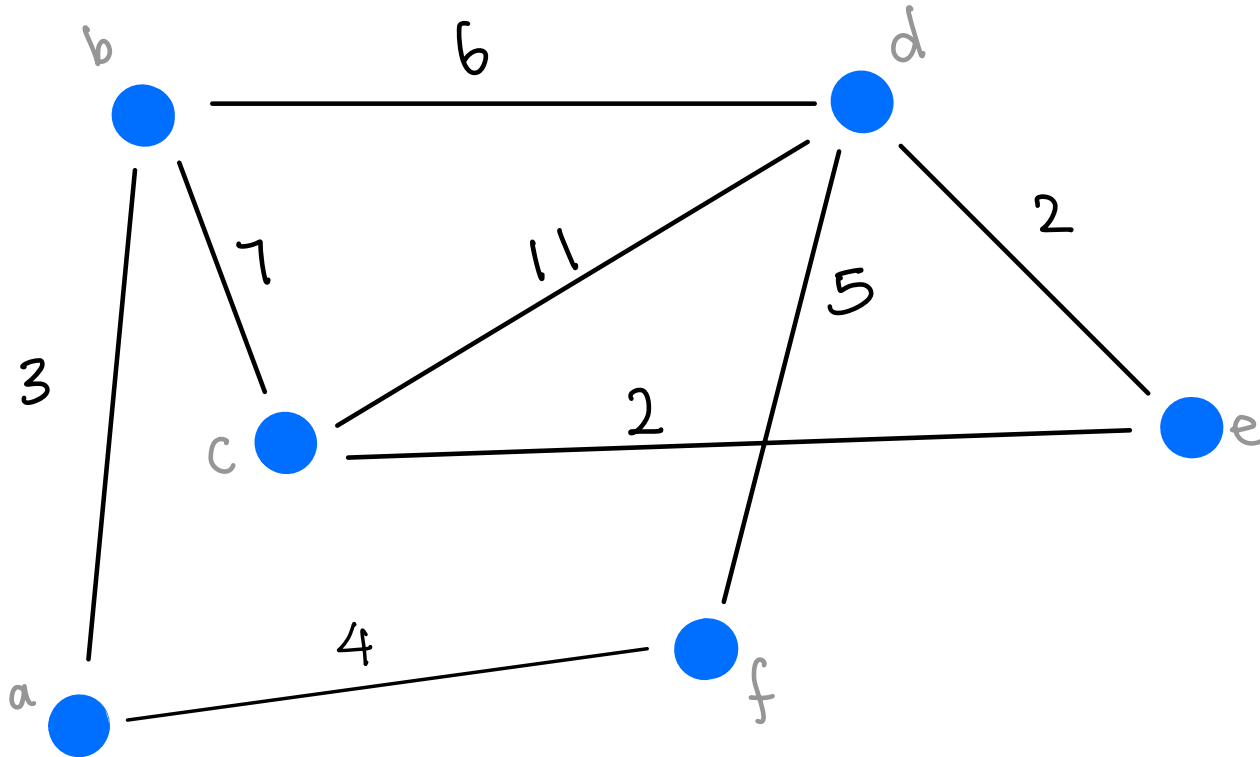
Directed

$E = \{ (b,a), (a,f), (c,b), (d,c), (b,d), (e,d), (c,e), (f,d) \}$
 (b,c)



Weights

$$w(\{b,d\}) = 6$$



Sometimes edges may have an associated weights. Formally, we can define a function $w : E \rightarrow \mathbb{R}$ where $w(\{u, v\})$ is the weight of the edge $\{u, v\}$.

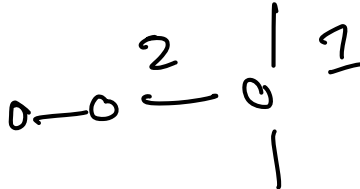
Seeing graphs everywhere

Google maps : $V = \text{places}$
 $E = \text{paths}$

Facebook : $V = \text{people}$
 $E = \text{friendships}$

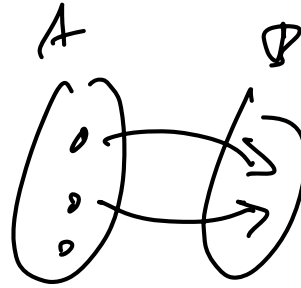
~~Bridge~~

$V = \text{cities}$
 $E = \text{Bridges}$

TTC : 

Computer network :

Seeing graphs everywhere



$$V = A \cup B$$

$$E: (a, f(a))$$

- Functions
- Binary relations
- Maps
- Web links
- Tournament brackets
- Game trees
- ...

Definitions

Modelling with Graphs

Problem 1. Matching

Problem 2. Shortest Path

Problem 3. The Traveling Salesman

Trees - A special type of graph

Problem 4. Minimum Spanning Tree

An Important Skill

The ability to model problems in real life as graph problems is super useful.

You will study algorithms to solve graph problems in CSC373.

Sometimes modelling the problem is enough since there are libraries that implement the algorithms for you!

One such library is the `networkx` (Python). We will see some examples...

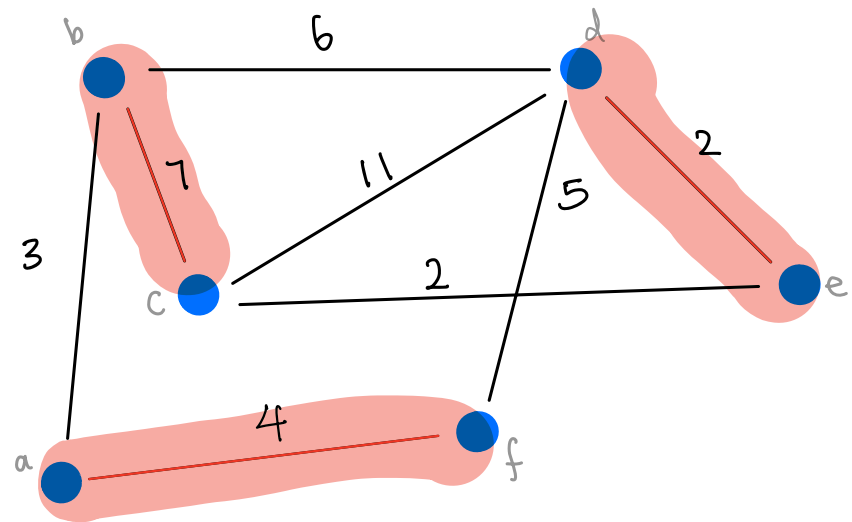
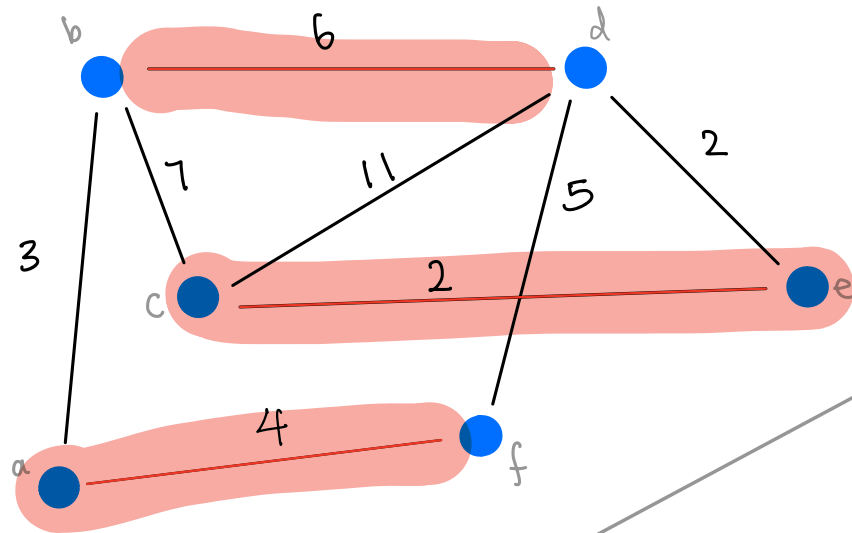
Matching

Let $G = (V, E)$ be a graph. A matching $M \subseteq E$ is a subset of edges that do not share any endpoints. I.e. every vertex appears in at most one edge in M .

A matching is **perfect** if every vertex appears exactly once in the matching.

If each edges has a weight, then the **weight of a matching** is the sum of the weights of edges in M .

Examples



Matching problem

$n \times \text{max_weight_matching}$:

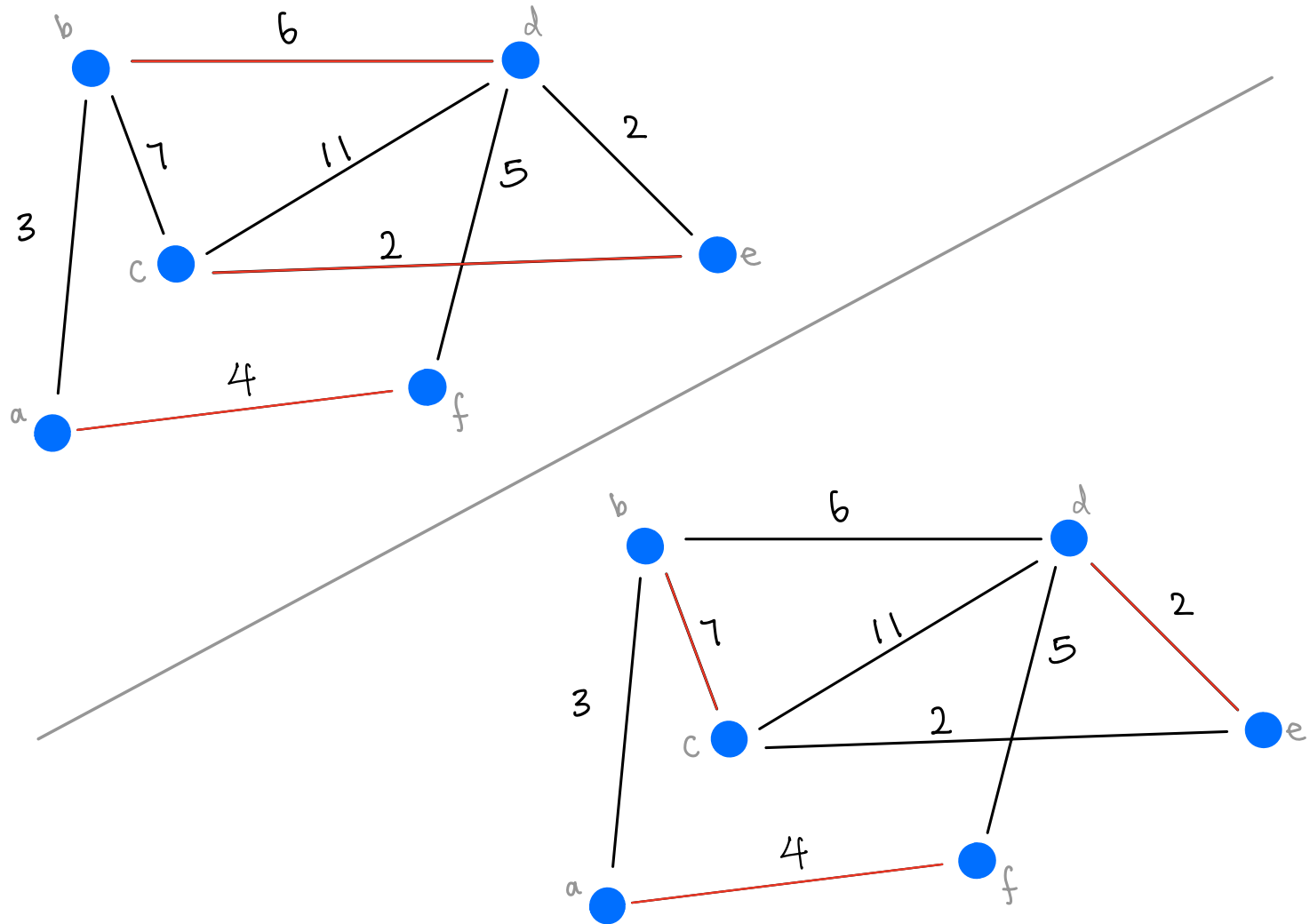
$n \times \text{Graph} \longrightarrow \mathcal{P}(E)$

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

Output: A matching $M \subseteq E$.

Usually we want $|M|$ to be as large as possible. Sometimes, we want to maximize/minimize the weight of the matching as well.

How would you find the maximum matching?



Example: Matchings in the real world

Here's the set up: There are n students in a class that need to be matched with partners. Each student fills out a form to indicate a list of times they are available to work and if they prefer to work in-person or virtually.

Let a, b be any two distinct students. a and b are **incompatible** if they don't share any available times. Otherwise, the **compatibility score** for a and b is the number of timeslots in which they overlap in their availability plus one if they additionally have the same preference to work in person or virtually.

Your task is to find a pairing with the following properties

- As many students should be matched as possible.
- Aim to have high compatibility score.

Modelling as a matching problem.

- What are the vertices of the graph? *Students.*
- What are the edges of the graph? *$u-v$ if compati*
- What are the edge weights? *$w(\{u,v\})$ is the compatibility score -*
- What properties of the matching do we want?

↪ max weight matching

Modelling as a matching problem.

- Students.
- There's an edge if they have non-zero compatibility score.
- Compatibility score.
- We want the largest matching (match as many students as possible) with the highest total compatibility!

Coding (?!)

Screenshots of Coding¹

	Virtual	Weekday afternoon	Weekend evening	Weekday evening	Weekend morning	Weekend afternoon	Weekday morning
Username							
Mildred Havercroft	0	1	1	0	0	0	0
Melody Mastroianni	1	0	1	1	1	1	0
Diana Williams	0	1	1	0	1	1	0
Kim Massaro	0	1	1	0	0	1	0
Larry Vass	1	1	1	1	0	1	0
Ethel Roberts	1	1	1	0	1	1	0
Laura Morello	0	1	1	1	0	1	0
Paula Mercado	1	0	1	1	1	1	0
Miriam Hurst	1	1	1	1	1	1	1
Mary Hutto	1	0	1	1	1	1	0

Note: Real data from this class with fake names.

¹Screenshots are from when I taught last summer, names are replaced with fake names

Screenshots of Coding¹

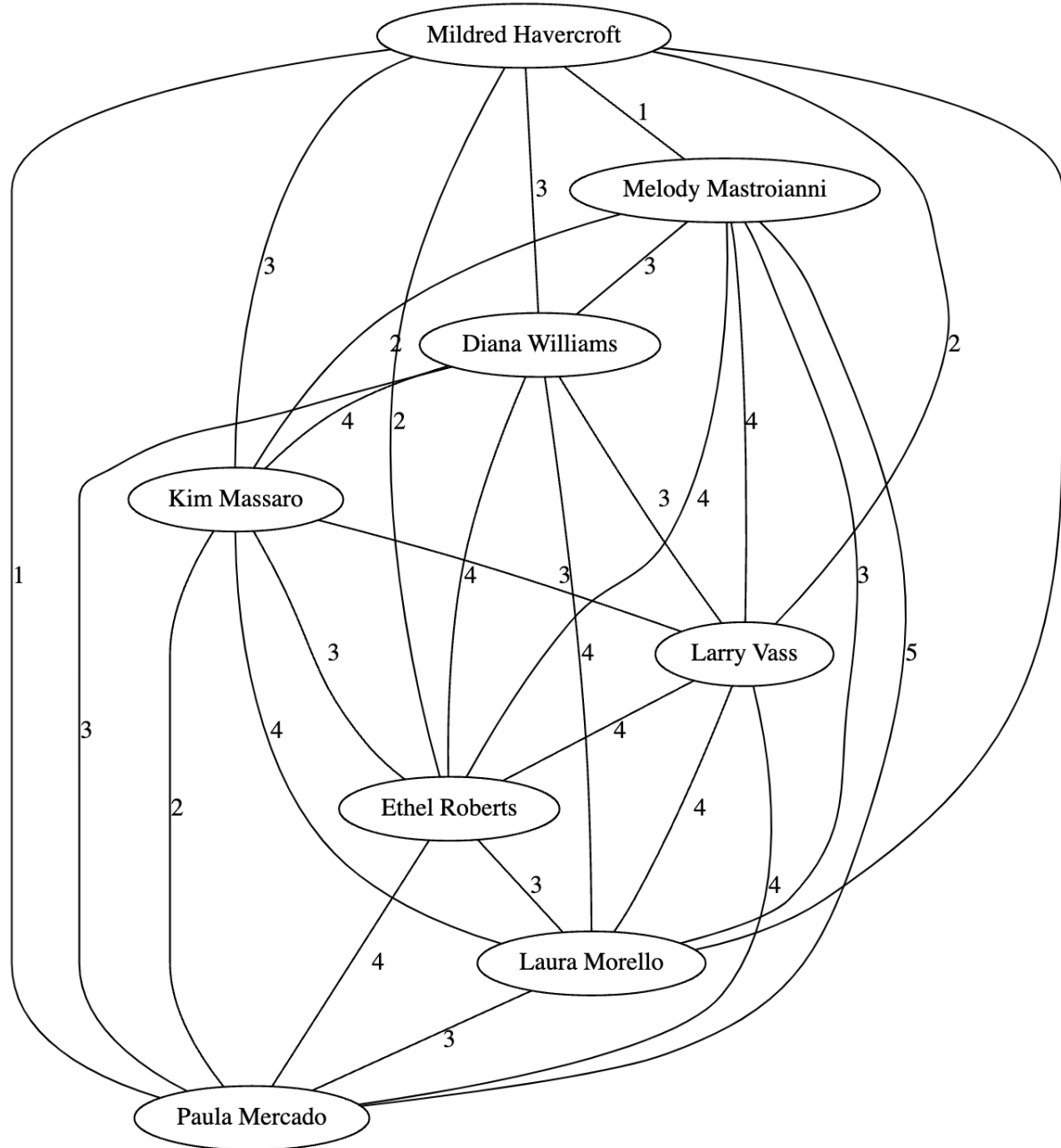
```
import networkx as nx
import itertools

def table_to_graph(data):
    data = data.head(8) # remove line for full example
    graph = nx.Graph()
    edge_list = []

    # for every pair of distinct students...
    for (user1, p1), (user2, p2) in itertools.combinations(data.iterrows(), 2):
        weight = sum(p1[TIMES].values & p2[TIMES].values)
        if weight > 0:
            weight += int(p1.Virtual == p2.Virtual)
            edge_list.append((user1, user2, weight))
    graph.add_weighted_edges_from(edge_list)
    return graph
```

¹Screenshots are from when I taught last summer, names are replaced with fake names

Screenshots of Coding¹



¹Screenshots are from when I taught last summer, names are replaced with fake names

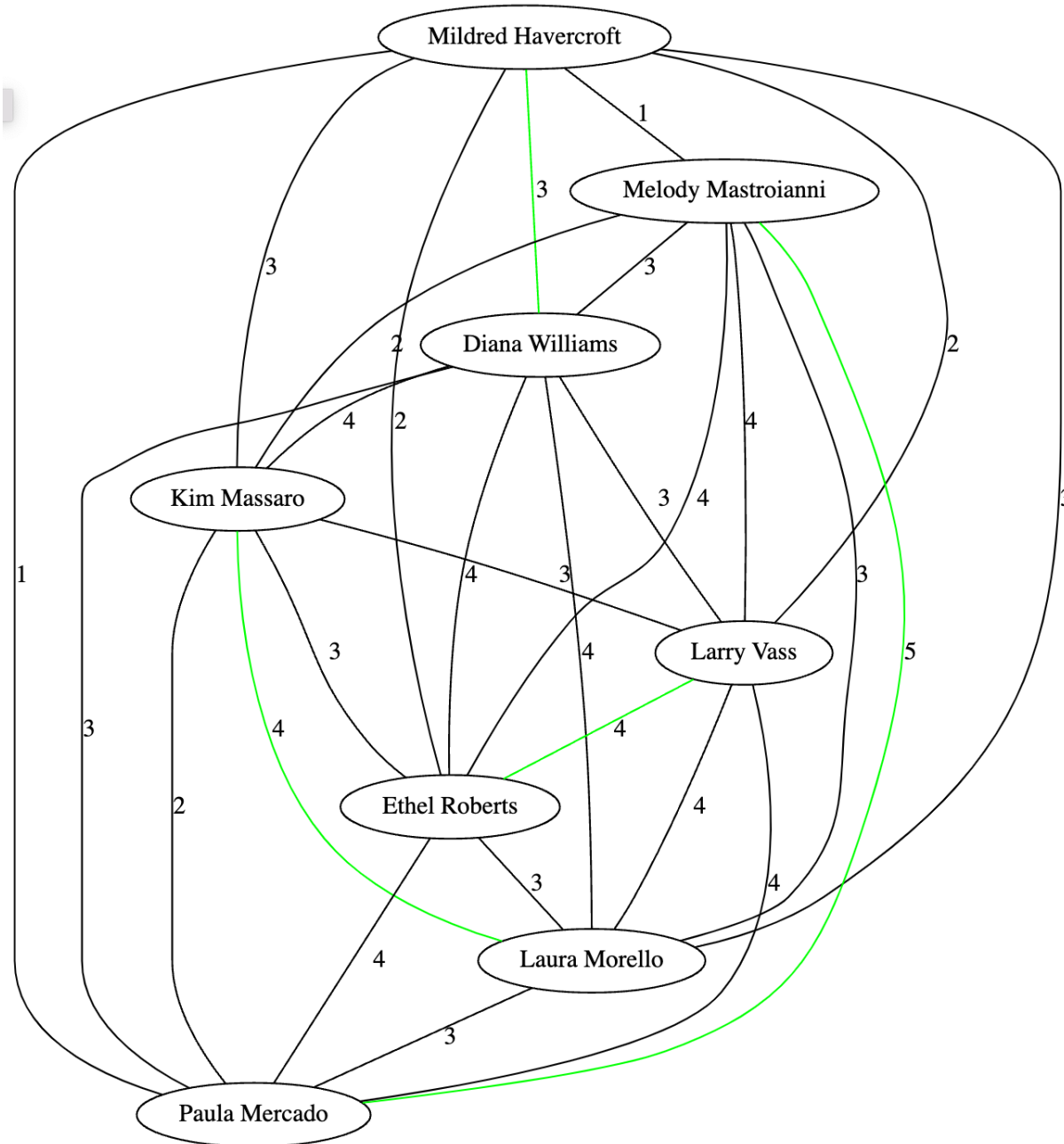
Screenshots of Coding¹

```
matching = nx.max_weight_matching(g, maxcardinality=True)

print(matching)
plot_graph_with_matching(g, matching)
```

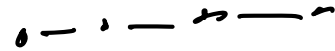
¹Screenshots are from when I taught last summer, names are replaced with fake names

Screenshots of Coding¹



¹Screenshots are from when I taught last summer, names are replaced with fake names

Paths

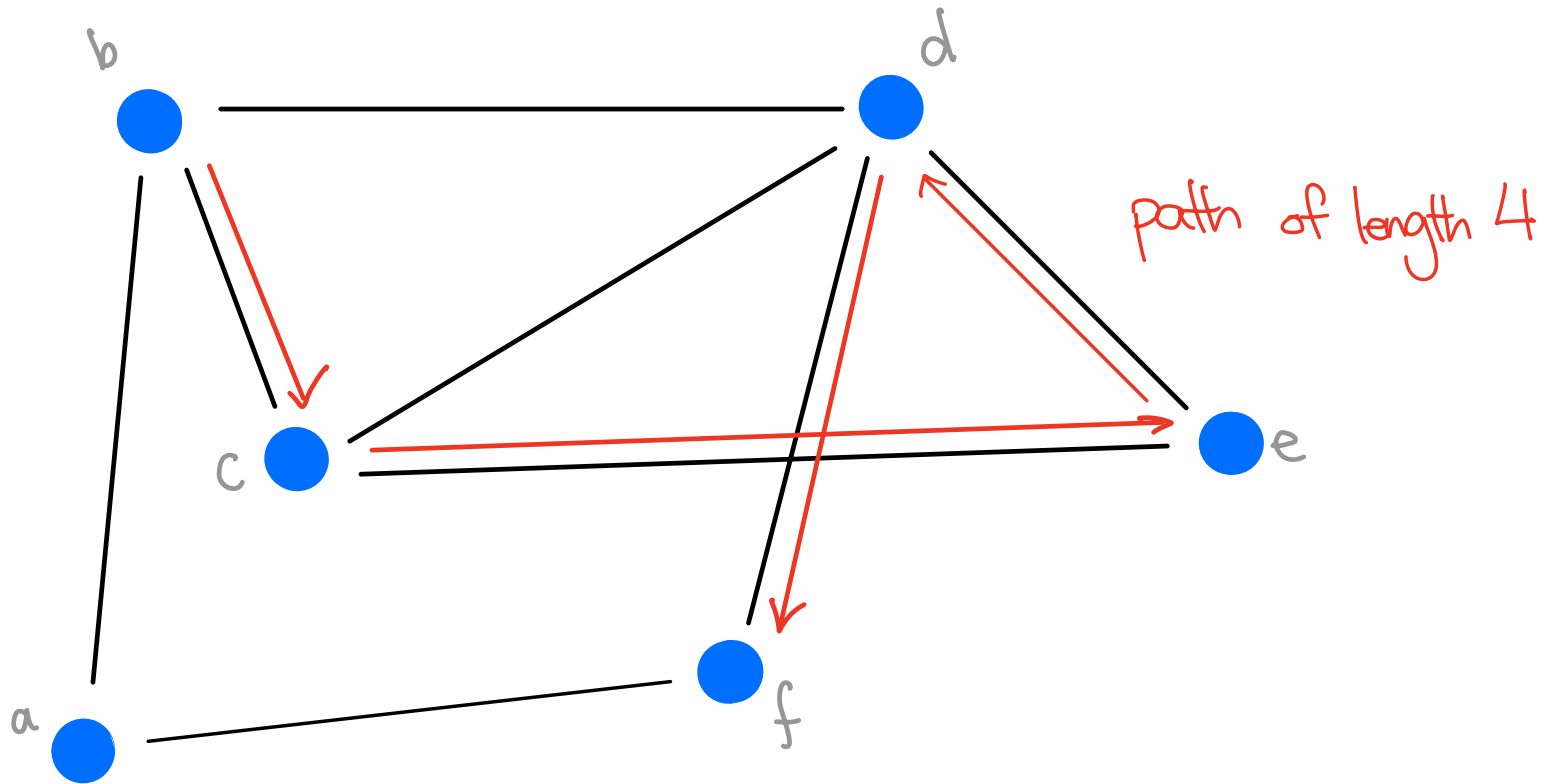


Let $G = (V, E)$ be a graph.

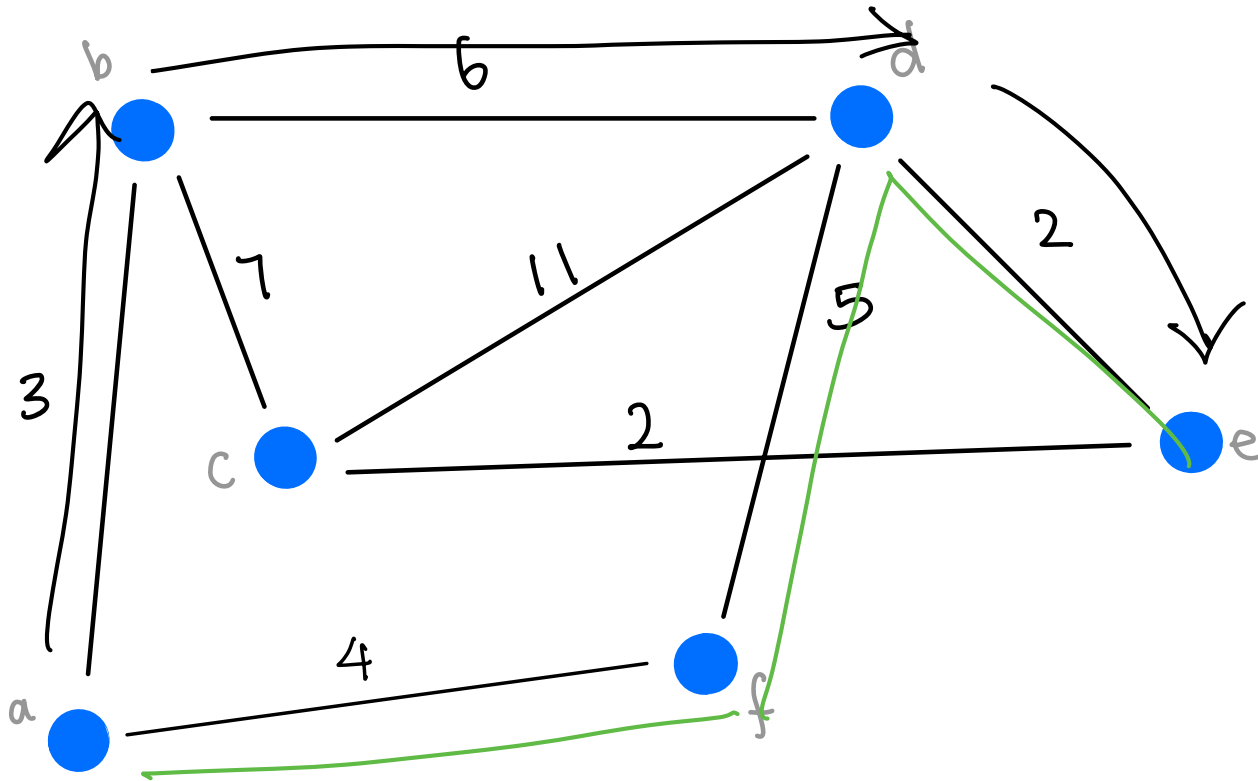
- Two vertices $u, v \in V$ are **adjacent** if $\{u, v\} \in E$
- A sequence of distinct vertices (v_1, \dots, v_n) is a **path** from v_1 to v_n if for every $i \in \{1, \dots, n-1\}$, v_i and v_{i+1} are adjacent. The **length** of the path is the number of edges in the path.

Example

b, c, e, d, f



Example



What's the shortest path from a to e?

Path Finding Problem

Input: A graph $G = (V, E)$ and two vertices $u, v \in V$.

Output: A path from u to v in G . I.e. a sequence of vertices (v_1, v_2, \dots, v_n) where $v_1 = u$ and $v_n = v$.

Typically, we want to find the path with the smallest length. If each edge has a weight we also may want to find the path with the smallest total weight.

Wikipedia Game



WIKIPEDIA
The Free Encyclopedia

[Main page](#)

[Contents](#)

[Current events](#)

[Random article](#)

[About Wikipedia](#)

[Contact us](#)

[Donate](#)

Here are the rules:

- Start with a random article
- Your goal is to find your way to the University of Toronto wiki page
- The only way you can move is by clicking on links

Let's play!

Modelling as a shortest path problem

V : articles

$E = \{(u, v) : \text{if } u \text{ contains a link to } v.\}$

$W = 1.$

Modelling as a shortest path problem

$A \times B = \text{cartesian product}.$

$\{(a, b) : a \in A, b \in B\}.$

$V \times V.$

- $V = \{\text{wiki pages}\}$
- $E = \{(u, v) \in V \times V : u \text{ links to } v\}$

Let $G = (V, E)$. Then, given a random Wikipedia page u , the shortest path from u to University of Toronto is the optimal solution for the Wikipedia Game.

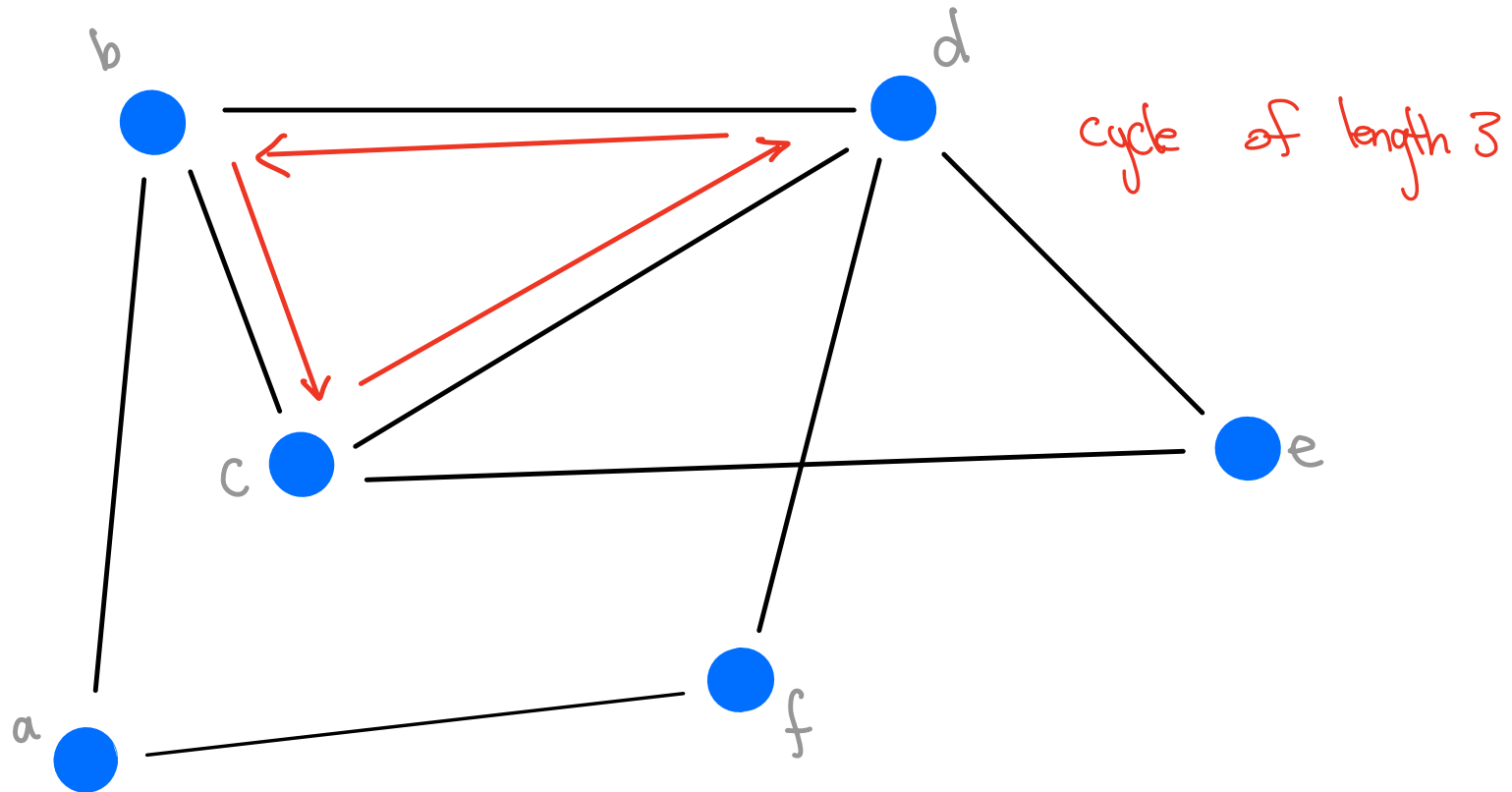
Cycles

Let $G = (V, E)$ be a graph

- A sequence of vertices (v_1, \dots, v_n) is a **cycle** if (v_1, \dots, v_{n-1}) is a path, $v_1 = v_n$, and $\{v_{n-1}, v_n\} \in E$.
- A cycle is called **Hamiltonian** if every vertex appears in the cycle exactly once (except for the start/end vertex which appears twice).

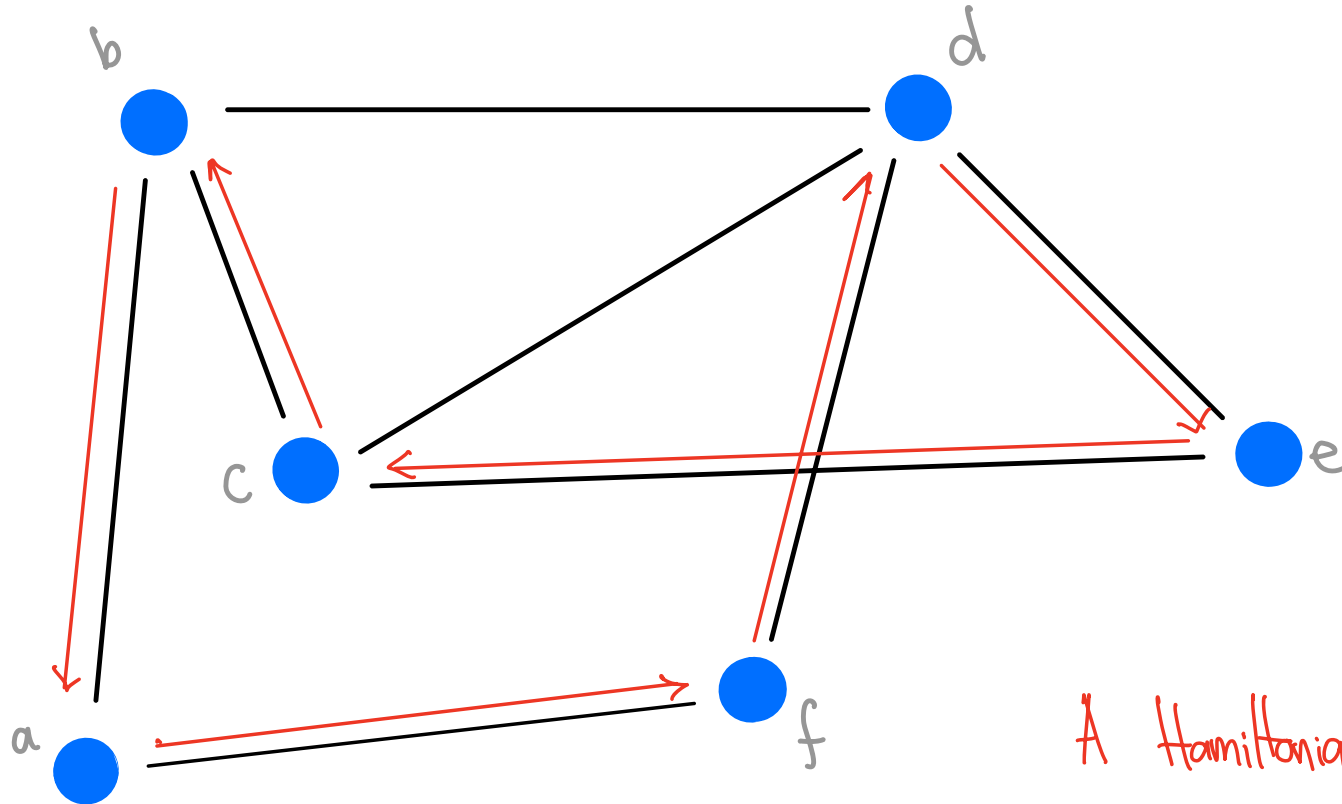
★ Length ≥ 3

Example - Cycle



Example - Hamiltonian Cycle

b a f d e c b



A Hamiltonian cycle

Traveling Salesman Problem

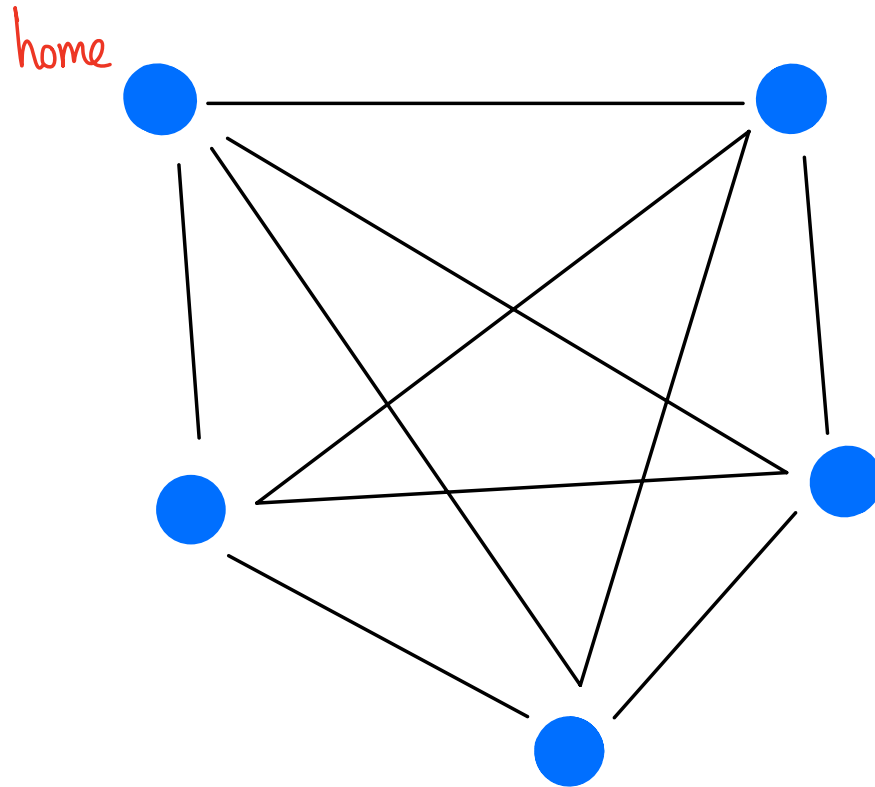
Input: A graph $G = (V, E)$ and a starting vertex h

Output: A Hamiltonian cycle in G starting from h that minimizes the total edge weights.

Selling Strawberries

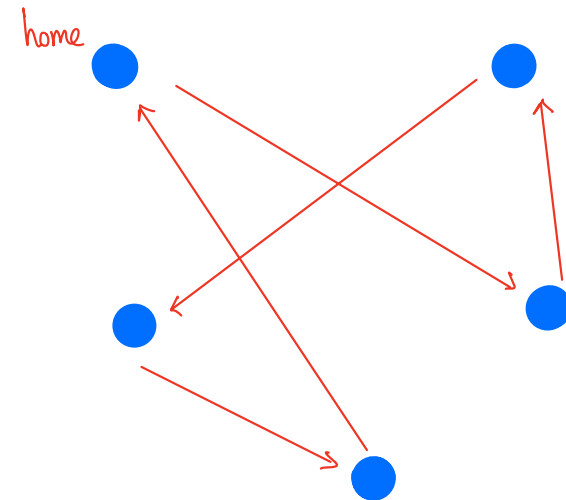
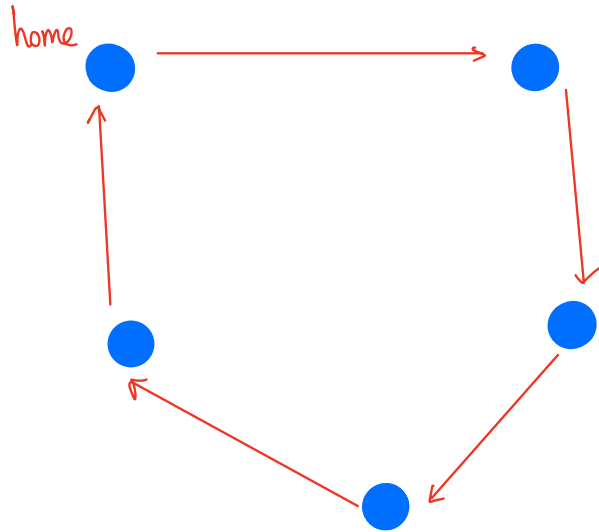
Imagine you're a door to door strawberries salesperson. Let H be the set of homes in your neighborhood. You live at $h \in H$.

Selling Strawberries

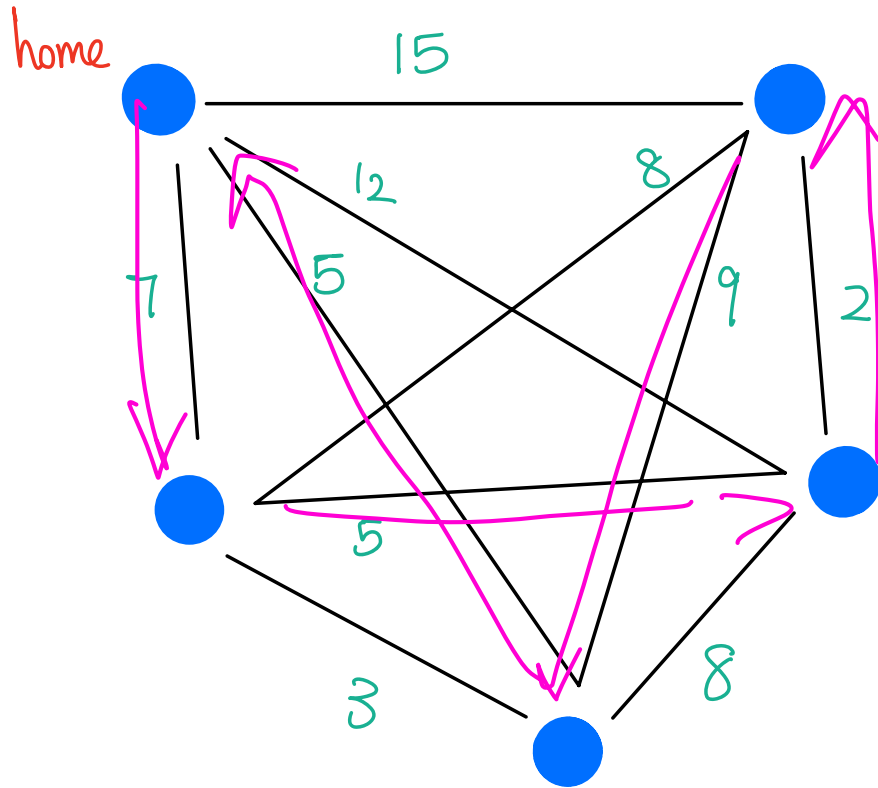


Your goal is to start at home, and visit every house in your neighborhood while walking the shortest distance possible.

Selling Strawberries



Selling Strawberries



This corresponds exactly to the solution to the Traveling Salesman Problem. I.e. we're looking for the Hamiltonian cycle that minimizes the total weight!

Definitions

Modelling with Graphs

Problem 1. Matching

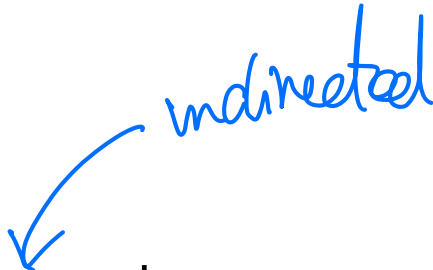
Problem 2. Shortest Path

Problem 3. The Traveling Salesman

Trees - A special type of graph

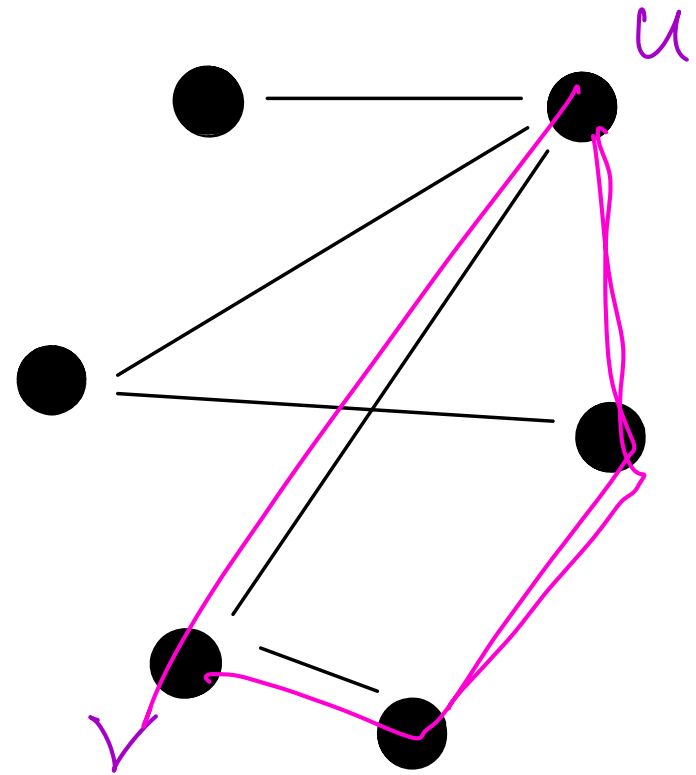
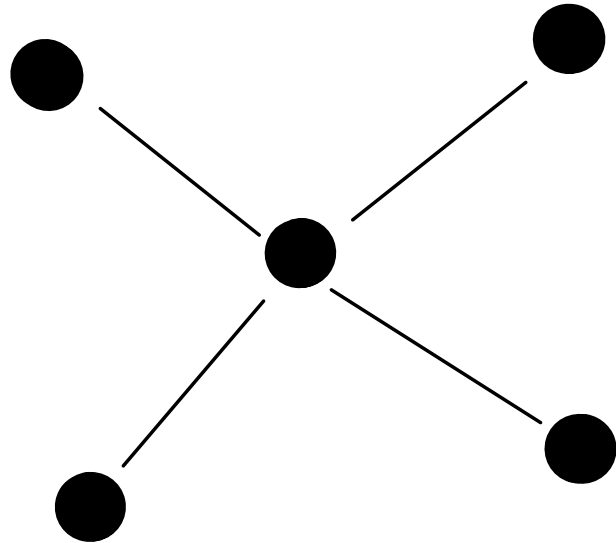
Problem 4. Minimum Spanning Tree

Trees

Let $G = (V, E)$ be any  graph.

- G is **connected** if for every pair of distinct vertices u, v , there is some path from u to v .
- G is **acyclic** if there are no cycles in G .
- G is called a **tree** if G is both connected and acyclic.

Examples



Trees - on a knife's edge

If G has many edges, it's more likely to be connected, but also more likely to have a cycle.

If G has fewer edges, it's more likely to be acyclic but less likely to be connected.

Since trees are both connected and acyclic, trees represent a perfect compromise. However, as we will see on the next slide, any addition or subtraction of an edge will destroy the balance.

Trees - on a knife's edge

Trees are **minimally connected graph**, meaning that it is connected but removing any edge causes the tree to be disconnected.

Trees - on a knife's edge

Trees are **minimally connected graph**, meaning that it is connected but removing any edge causes the tree to be disconnected.

Trees are **maximally acyclic graph**, meaning that there is no cycle but adding any edge creates a cycle.

Proof: Minimally Connected

Tree: connected and acyclic.

WTS: removing any edge causes tree to be disconnected.

Let $G = (V, E)$ be a tree, let $e = \{u, v\}$ be any edge. Consider $G' = (V, E \setminus \{e\}) \rightarrow (G - e)$.

WTS G' is not connected.

In particular, I claim, there is no path from u to v .

By contr. assume \exists path $u = v_1, \dots, v_k = v$. Then,
 $\overset{u}{v_1}, v_2, \dots, \overset{v}{v_k} = \overset{u}{v_1}$ is a cycle in G .

However G was acyclic $\Rightarrow \Leftarrow$.

Proof: Maximally Acyclic


Tree: connected and acyclic.

WTS: adding any edge creates a cycle

Let G be a tree, and let $\{u, v\}$ be an edge not already in G . Since G is connected,

\exists a path $u = v_1, v_2, v_3, \dots, v_k = v$ then

the graph $G + \{u, v\}$ has the following cycle.



$u = v_1, v_2, \dots, v_k = v, v_1 = u$

Converses

Converses

$$P \Rightarrow Q$$
$$Q \Rightarrow P -$$

Let G be a graph.

- If G is minimally connected. Then G is a tree.
- If G is maximally acyclic. Then G is a tree.

Are these also true?

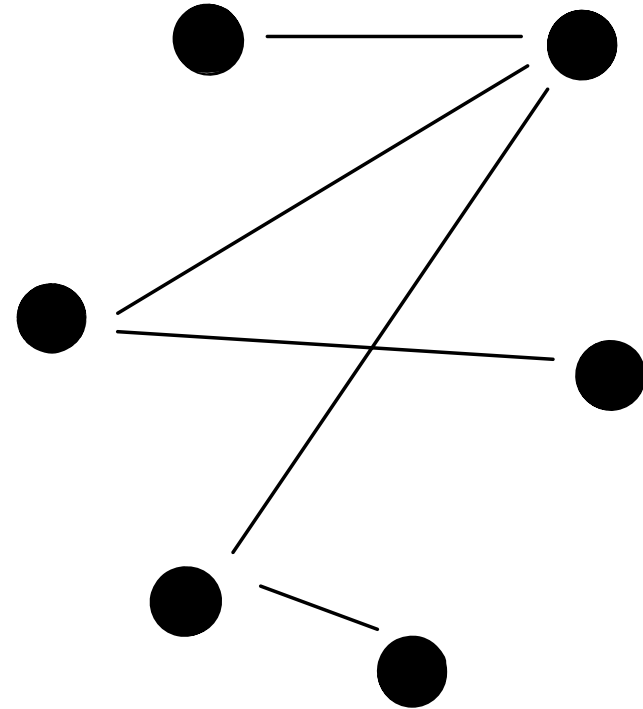
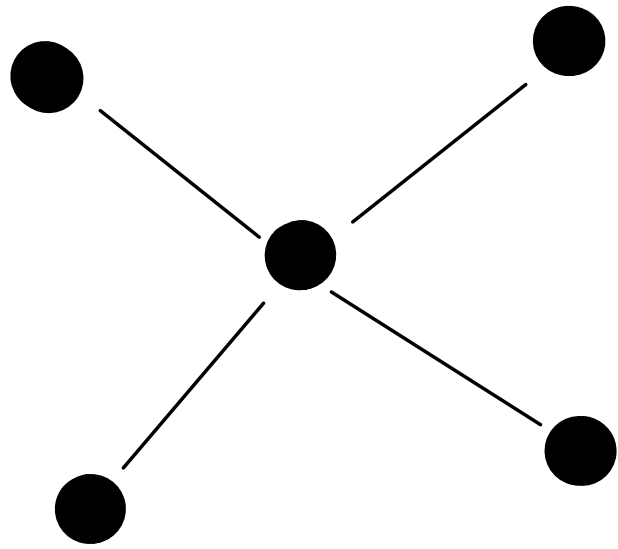
Converses

Let G be a graph.

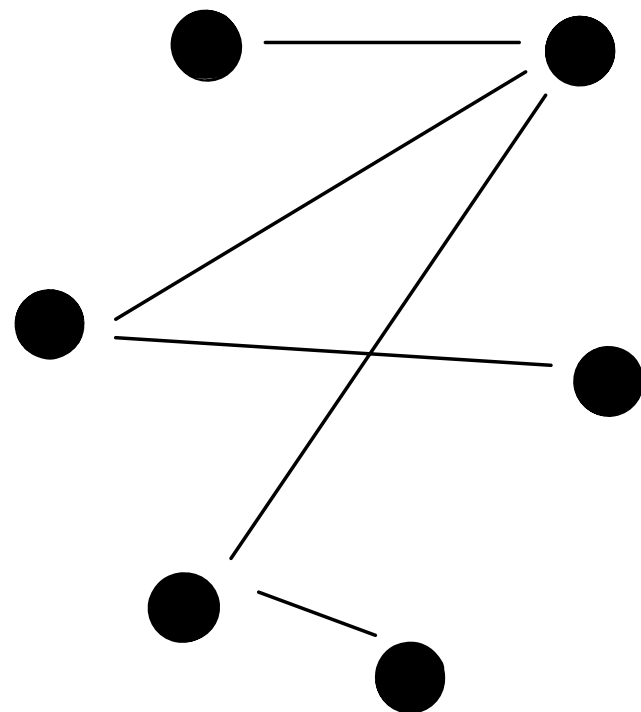
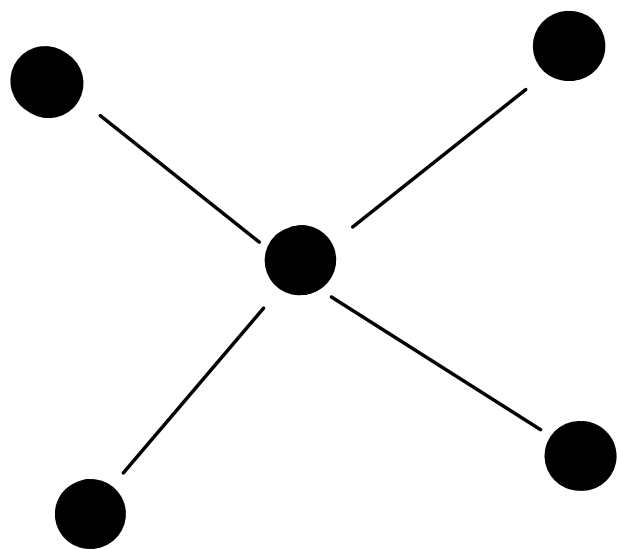
- If G is minimally connected. Then G is a tree.
- If G is maximally acyclic. Then G is a tree.

Are these also true? Yes!

How many edges does a tree have?



How many edges does a tree have?



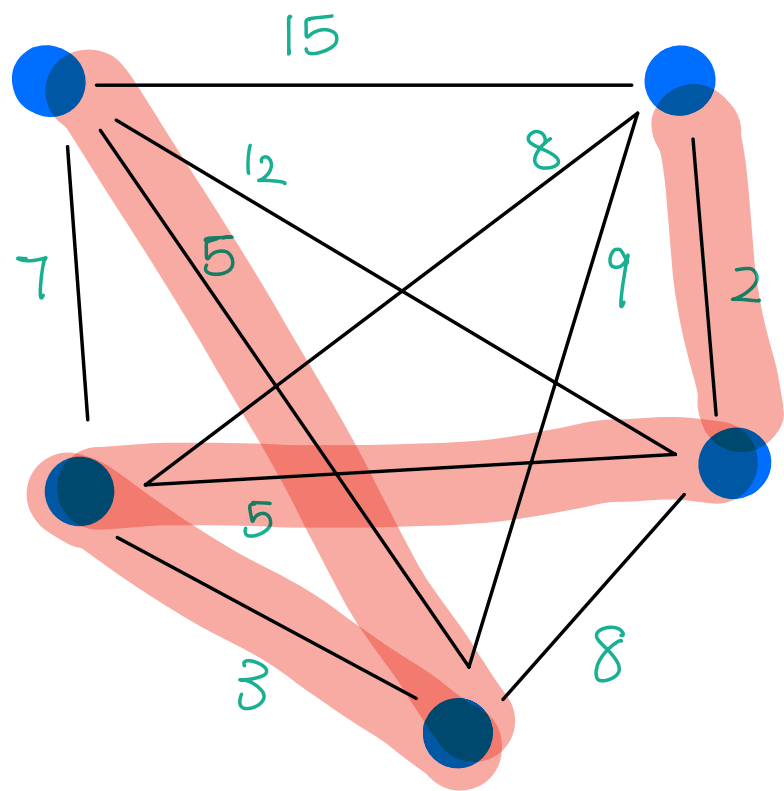
$|V| - 1$. We will prove this next time.

Electrical Grid

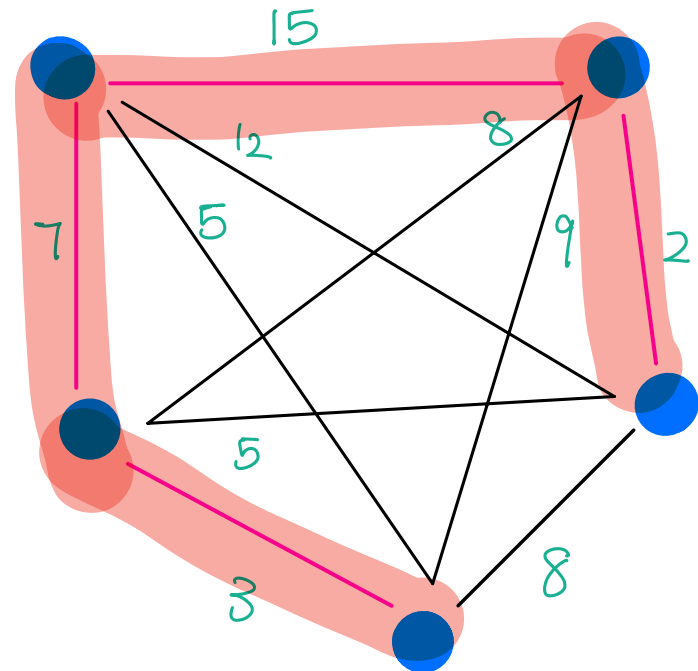
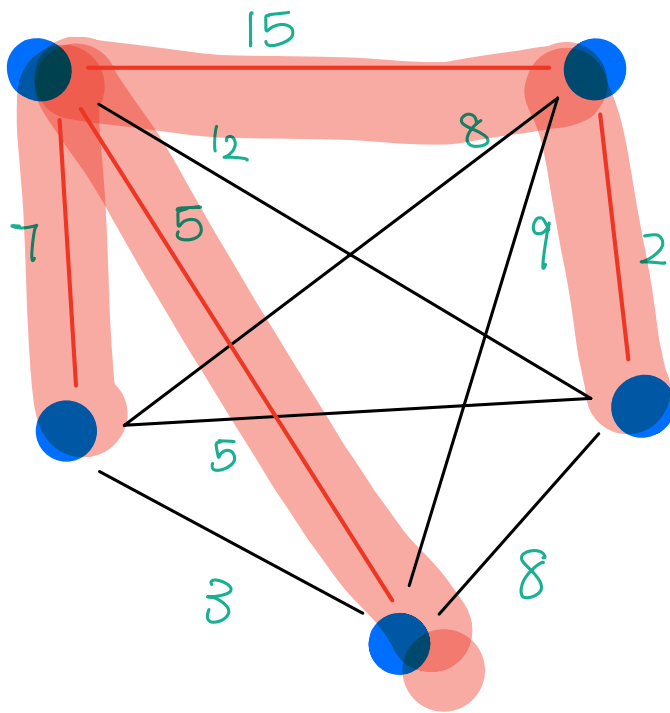
Given a set of houses, what is the most efficient way to connect an electrical grid?

The goal is to connect homes so that all of the homes are connected to each other by some path using the least amount of cable.

Electrical Grid



Electrical Grid



two spanning trees.

Minimum Spanning Tree

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

Output: A graph $T = (V, E')$, where $E' \subseteq E$, such that T is a tree that minimizes the total edge weights.

Which of these problems seem more difficult?

- Minimum Spanning Tree
- Shortest Path
- Matching
- Traveling Salesman

Which of these problems seem more difficult?

- Minimum Spanning Tree
- Shortest Path
- Matching
- Traveling Salesman

We know fast algorithms for the first three, but NOT for the last one. In fact, the Traveling Salesman problem is conjectured to have no efficient algorithm.

Additional Notes

- The skill I want you to develop here is to identify how real world problems can be translated to known problems on graphs.
- We did not study in detail HOW to solve such problems (that's the main topic of CSC373).
- Even though we don't know of a good algorithm for TSP, we do have fast approximation algorithms for it.

Additional Notes

Here are some references for some algorithms in case you are curious.

- Matching: [1 (Blossom Algorithm)], [2]
- Shortest Path: [1 (Dijkstra's Algorithm)]
- TSP: [1 (Christofides Algorithm)]
- MST: [1 (Prim's Algorithm)], [2 (Kruskal's Algorithm)]

Tutorials

Go to the same tutorial as last week!

If you are in tutorial 5101 (Room BA2165), please go to the following room instead

- If your birthday is on the 1-10th of the month, go to BA2195
- If your birthday is on the 11-20th of the month, go to BA 2159
- If your birthday is on the 21-31st of the month, go to BA2139