Real-World Analysis + Project

CS/MCS 401: Computer Algorithms I
Lecture 4
Professor: Jonathan Liu

Sit next to someone! If you don't do it now, you'll have to move later.

Introduce yourself!
Icebreaker: Who is someone you want to be more like, and why?

Lesson Goals

Today we're exploring two interesting problems.

- First, we'll introduce one of the most influential algorithms of the modern era: PageRank.
- Next, we'll introduce the problem underlying our course project.

Search Engines and PageRank

Early Searching

In the early days of the internet (before 1996), search engines did one thing: find webpages that were most similar to the content you searched for.

If you searched “ice cream”, it scoured its database for all web pages that contain the words “ice cream” and returned them to you. It ranked them by how often the words “ice cream” appeared.

What do you think happened?

Early Searching

In the early days of the internet (before 1996), search engines did one thing: find webpages that were most similar to the content you searched for.

If you searched “ice cream”, it scoured its database for all web pages that contain the words “ice cream” and returned them to you. It ranked them by how often the words “ice cream” appeared.

What do you think happened?

Websites would write “ice cream ice cream ice cream ice cream” in tiny white font so that machines would rank it higher (even on non-“ice cream” pages!)

How do you find “good” websites?

There needed to be a better way to determine which sites were “good.” But companies couldn’t exactly go through and manually rank every website.

Enter:

- Robin Li (Baidu)
- Larry Page (Google)

Key Idea: Let the pages rank each other! If my page links to Wikipedia, I’m endorsing Wikipedia as a good, useful website. So we should rank pages by how many pages link to them.

What do you think happened?

People started making a bunch of fake pages linking to their own (real) page.

So then what?

How do we stop fake links from getting counted?

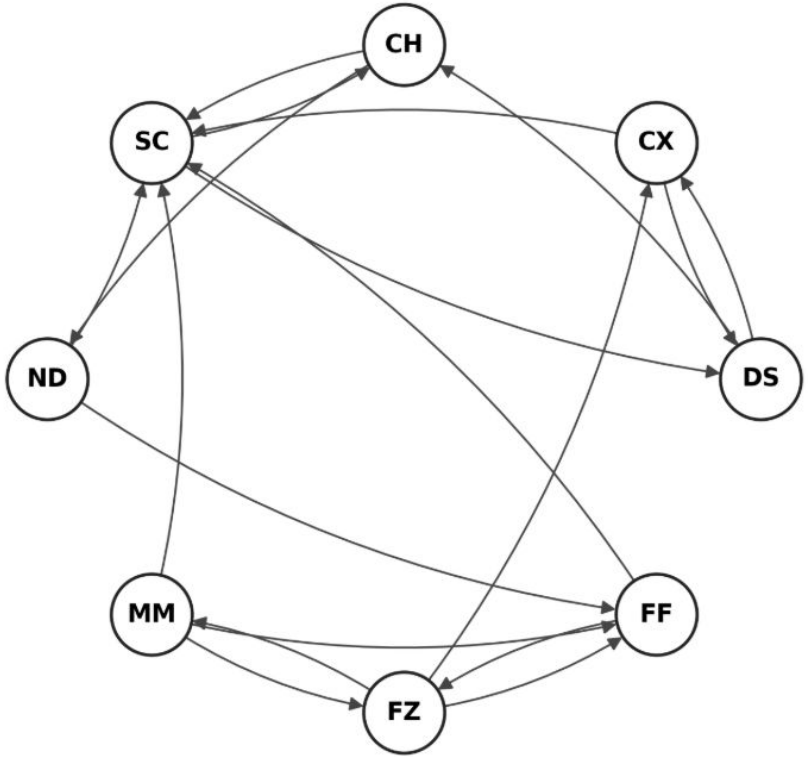
Idea 2: Links from “good” websites should be worth more!

If a good site links to me, that means more than if a random site links to me.

Google 1998

Let’s look at a real graph! (Problem 1a)

Code	Site	What it is
SC	The Scoop	ice cream review blog, running since 1998
DS	Dairy Science Digest	food-science site, rigorous, posts rarely
CH	Churn	independent review site
ND	Nondairy Now	non-dairy ice cream, new
FF	FlavorFeed	aggregator; reposts everyone’s content
FZ	Freezer Notes	home-churning hobbyist blog
MM	Meltdown	ice cream memes
CX	The Codex	volunteer reference wiki



PageRank Pseudocode

```
set quality[p] = 1/n for all n pages.

while forever:
    set new_quality[p]=0 for all pages
    for every page p:
        for every link from p to q:
            add quality[p]*(1/links on p) to new_quality[q]

    for every page p:
        new_quality[p] = 0.85 * new_quality[p] + 0.15 * (1/n)

    if quality = new_quality, exit the loop
    else set quality = new_quality
return pages, sorted by quality
```

PageRank Pseudocode

```
set quality[p] = 1/n for all n pages.

while forever:
    set new_quality[p]=0 for all pages
    for every page p:
        for every link from p to q:
            add quality[p]*(1/links on p) to new_quality[q]

    for every page p:
        new_quality[p] = 0.85 * new_quality[p] + 0.15 * (1/n)

    if quality = new_quality, exit the loop
    else set quality = new_quality
return pages, sorted by quality
```

Start with all websites being equal.

PageRank Pseudocode

```
set quality[p] = 1/n for all n pages.

while forever:
    set new_quality[p]=0 for all pages
    for every page p:
        for every link from p to q:
            add quality[p]*(1/links on p) to new_quality[q]

    for every page p:
        new_quality[p] = 0.85 * new_quality[p] + 0.15 * (1/n)

    if quality = new_quality, exit the loop
    else set quality = new_quality
return pages, sorted by quality
```

Get each link's quality in terms of the quality of sites that link to it.

PageRank Pseudocode

```
set quality[p] = 1/n for all n pages.

while forever:
    set new_quality[p]=0 for all pages
    for every page p:
        for every link from p to q:
            add quality[p]*(1/links on p) to new_quality[q]

    for every page p:
        new_quality[p] = 0.85 * new_quality[p] + 0.15 * (1/n)

    if quality = new_quality, exit the loop
    else set quality = new_quality
return pages, sorted by quality
```

Bring each value a bit closer to the average, so none are too strong or too weak.

PageRank Pseudocode

```
set quality[p] = 1/n for all n pages.

while forever:
    set new_quality[p]=0 for all pages
    for every page p:
        for every link from p to q:
            add quality[p]*(1/links on p) to new_quality[q]

    for every page p:
        new_quality[p] = 0.85 * new_quality[p] + 0.15 * (1/n)

    if quality = new_quality, exit the loop
    else set quality = new_quality
return pages, sorted by quality
```

Repeat until the quality rankings stop changing.

PageRank Pseudocode (Problem 1b, 1c)

```
set quality[p] = 1/n for all n pages.

while forever:
    set new_quality[p]=0 for all pages
    for every page p:
        for every link from p to q:
            add quality[p]*(1/links on p) to new_quality[q]

    for every page p:
        new_quality[p] = 0.85 * new_quality[p] + 0.15 * (1/n)

    if quality = new_quality, exit the loop
    else set quality = new_quality
return pages, sorted by quality
```

Potential Changes to PageRank

```
set quality[p] = 1/n for all n pages.

while forever:
    set new_quality[p]=0 for all pages
    for every page p:
        for every link from p to q:
            add quality[p]*(1/links on p) to new_quality[q]

    for every page p:
        new_quality[p] = 0.85 * new_quality[p] + 0.15 * (1/n)

    if quality = new_quality, exit the loop
    else set quality = new_quality
return pages, sorted by quality
```

These values could be anything!
In fact, if you change .85 to 1 then ND > CX. If you make it 0.5, then FF jumps to 4th.

Potential Changes to PageRank

```
set quality[p] = 1/n for all n pages.

while forever:
    set new_quality[p]=0 for all pages
    for every page p:
        for every link from p to q:
            add quality[p]*(1/links on p) to new_quality[q]

    for every page p:
        new_quality[p] = 0.85 * new_quality[p] + 0.15 * (1/n)

    if quality = new_quality, exit the loop
    else set quality = new_quality
return pages, sorted by quality
```

This value can be page-specific! This could be weighted in other ways, like favoring newer sites or more popular ones.

Takeaways

It doesn’t take a crazy algorithm to change the world – just the right application. (The underlying idea was proposed in 1895 to determine the winner of a chess tournament.)

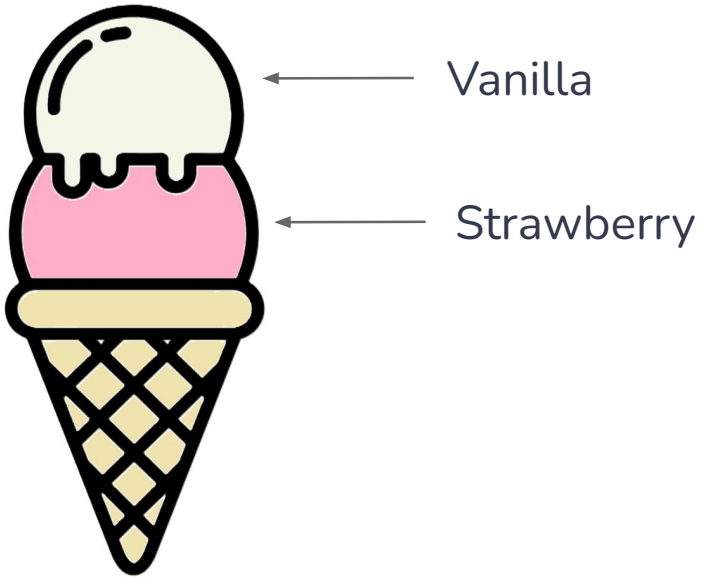
Algorithms change human behavior, which may require us to change algorithm behavior!

Simple decisions in algorithms can have really strong real-world consequences. For example, they may either allow newer websites to get traction, or prevent them from doing so!

Project: Cone-flict of Interest

The Story

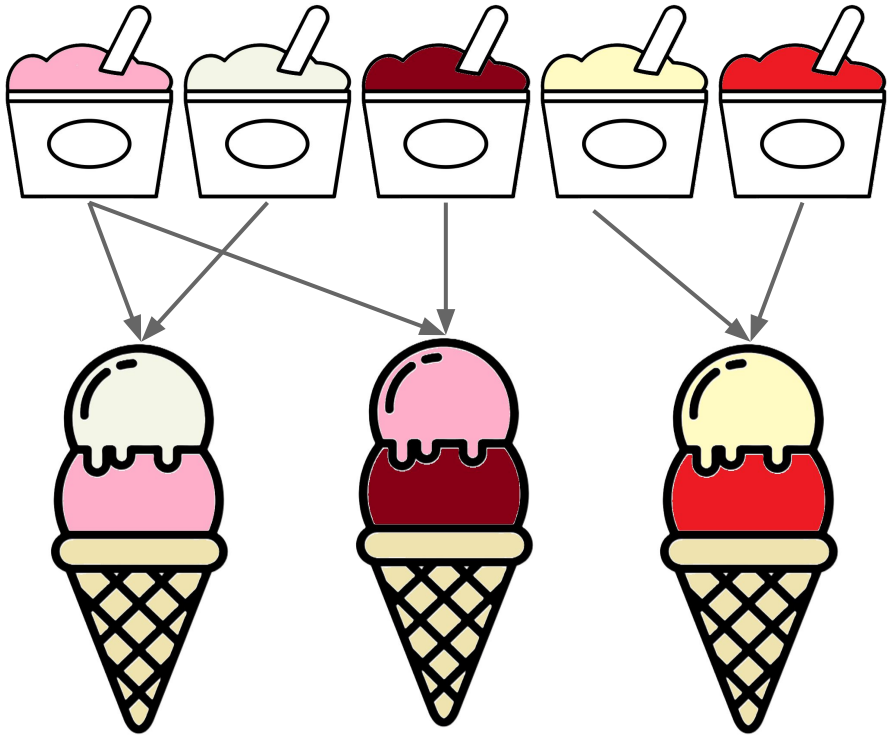
Your ice cream shop specializes in **flavor combination cones**: cones with two different flavors that go well together.



The Story

Your ice cream shop specializes in **flavor combination cones**: cones with two different flavors that go well together.

You have 2n flavors in your inventory, and you'd like to use them to offer a menu with as many **good** pairings as you can, without repeating any flavors.

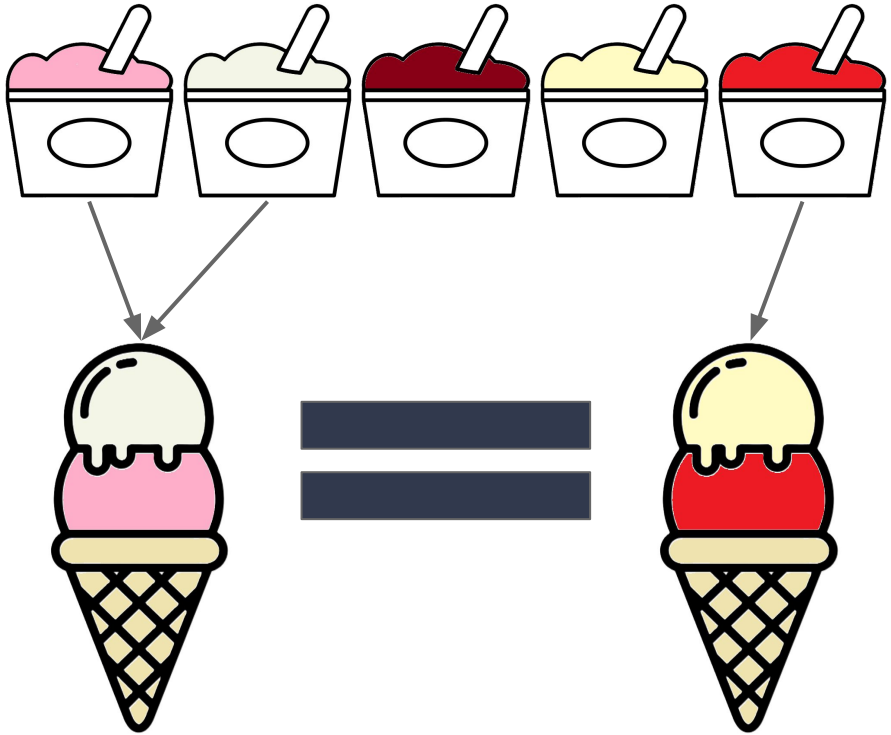


The Story

Your ice cream shop specializes in **flavor combination cones**: cones with two different flavors that go well together.

You have 2n flavors in your inventory, and you'd like to use them to offer a menu with as many **good** pairings as you can, without repeating any flavors.

However, some pairings are too similar to one another. Each similar pairing will get a "similarity penalty."



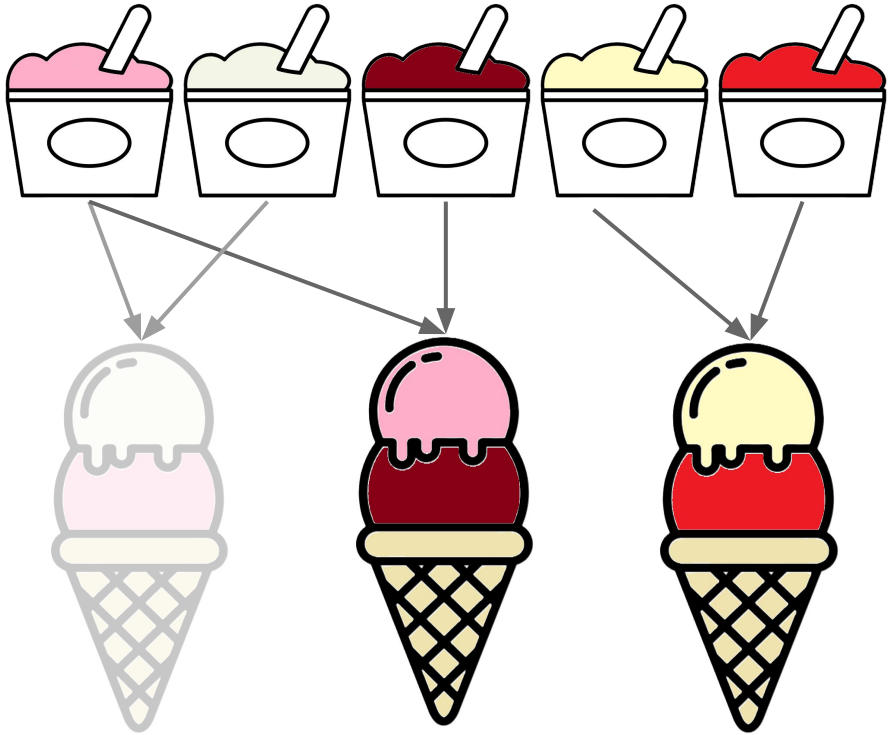
The Story

Your ice cream shop specializes in **flavor combination cones**: cones with two different flavors that go well together.

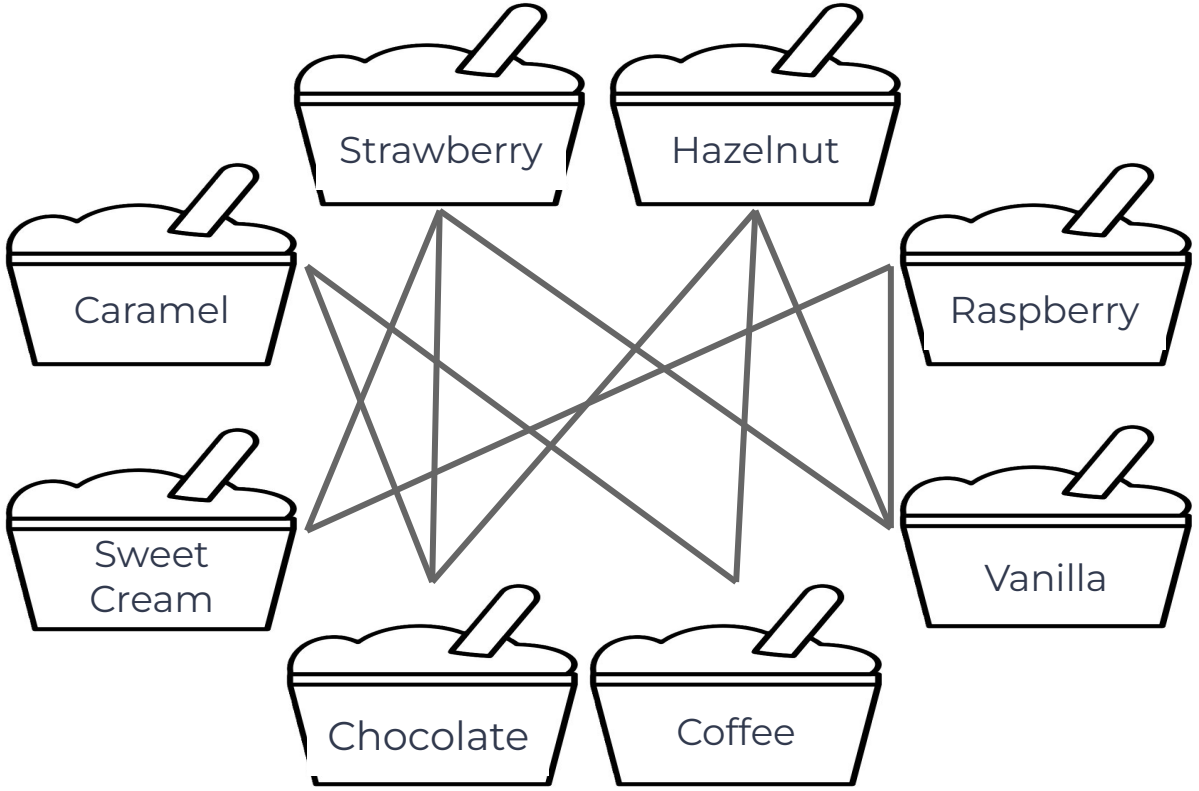
You have 2n flavors in your inventory, and you'd like to use them to offer a menu with as many **good** pairings as you can, without repeating any flavors.

However, some pairings are too similar to one another. Each similar pairing will get a "similarity penalty."

Goal: Find the menu that offers as many flavors as possible, while trying to also minimize similarity between pairs.

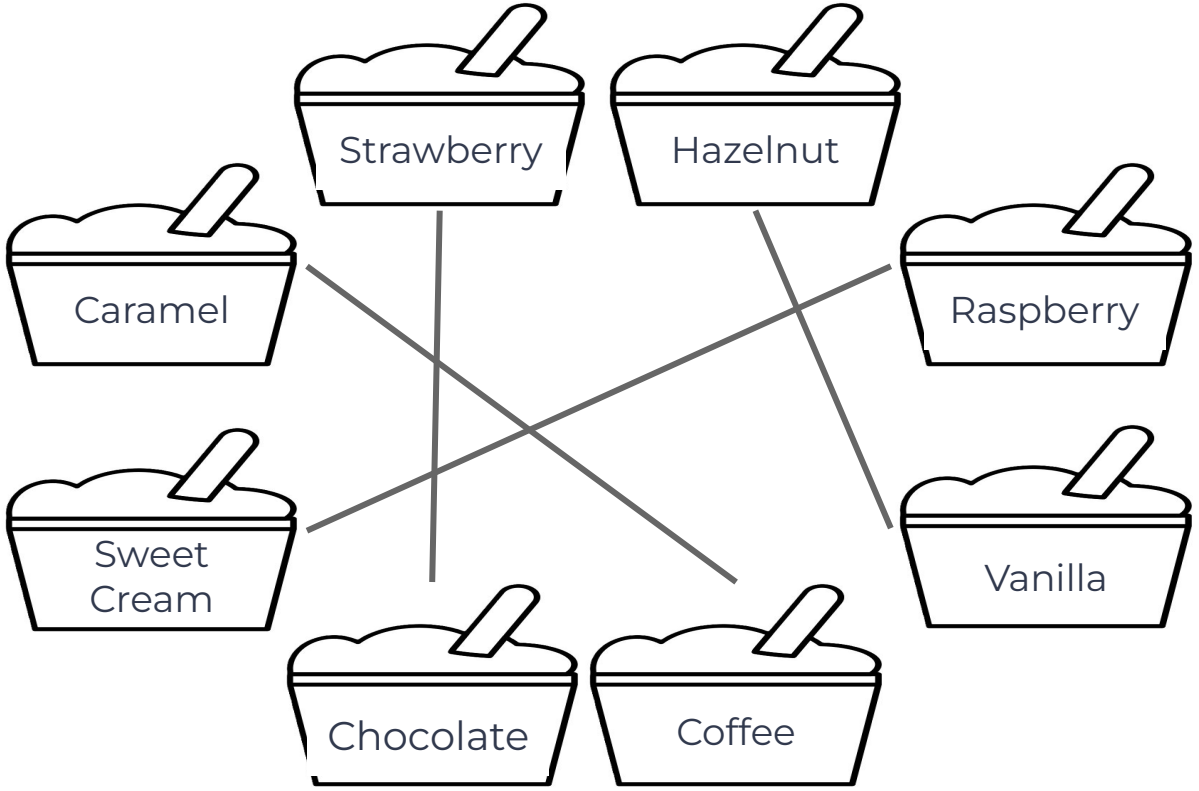


Try it yourself! (Problem 2)



Pair 1	Pair 2	Score
Strawberry+ Vanilla	Raspberry+ Sweet Cream	9
Strawberry+ Sweet Cream	Raspberry+ Vanilla	8
Caramel + Coffee	Chocolate + Hazelnut	4
Chocolate + Caramel	Coffee + Hazelnut	2
Strawberry + Chocolate	Caramel + Coffee	2
Vanilla + Hazelnut	Caramel + Coffee	2
Vanilla + Hazelnut	Raspberry + Sweet Cream	2
Strawberry + Chocolate	Raspberry + Sweet Cream	1

A menu with 7



Pair 1	Pair 2	Score
Strawberry+ Vanilla	Raspberry+ Sweet Cream	9
Strawberry+ Sweet Cream	Raspberry+ Vanilla	8
Caramel + Coffee	Chocolate + Hazelnut	4
Chocolate + Caramel	Coffee + Hazelnut	2
Strawberry + Chocolate	Caramel + Coffee	2
Vanilla + Hazelnut	Caramel + Coffee	2
Vanilla + Hazelnut	Raspberry + Sweet Cream	2
Strawberry + Chocolate	Raspberry + Sweet Cream	1

Project Overview

We have good reason to believe that **there is no efficient way to solve this problem**. (You'll see this in Unit 6).

Project Goal: Write a solver for this problem. Your solver will not need to output the best possible solution, but as good of one as you can find. It also does not need to follow any runtime constraints.

Learning Goal: See, in practice, how hard algorithms problems can really be, and explore what to do about it.

Project Logistics

Deliverables:

You will be given many different inputs to this problem. They will range in size and difficulty.

- You will need to submit the best menu you found for each input.
- You will need to submit the code you ran to find them.
- You will give small presentations on 12/3 to a few classmates who will grade you. You will also present to me.

You will be graded on menu score and your ability to explain your approach.

If you are an undergraduate, you may work with a partner on this project. You will be asked to name your partner shortly after the midterm.

If you are a graduate student, you must work alone.

Unit Overview

Over the last two weeks, we have learn about many tools and skills that will be useful throughout the entire course.

We learned:

- How to convert real-world problems to algorithms problems
- How to write pseudocode
- How to write proofs of correctness
- How to talk about runtime
- How to think about real-world consequences of algorithm decisions

All of these skills will be used repeatedly throughout the whole course.

The algorithms (stable matching, pagerank) will not.