Welcome!

Puzzle of the Day:

Connect the dots.
Use ?? straight lines.
Don’t lift your pen from the paper.

5 x 4:
Use 7 lines.

Can you end at the same place you began?

7 x 7:
Use 12 lines
Why are we here?

Everyone here knows classical algorithms:

- Dijkstra’s, Prim’s, Red-black trees, Fibonacci Heaps, etc.

What happens when you try to use these algorithms on real-world data sets?
Imagine a graph containing:

- 4 billion nodes
- 1.2 trillion edges

E.g., $\approx 1$ TB of data

(The Facebook graph, for example, is at least that big.)

(Today, a 1TB data set is not that big.)
Imagine a graph containing:

- 4 billion nodes
- 1.2 trillion edges

Simple question:
What is the diameter of your graph?

E.g., ≈ 1 TB of data
Imagine a graph containing:
- 4 billion nodes
- 1.2 trillion edges

Simple question:
What is the diameter of your graph?

Simple answer:
Run breadth-first-search n times.

E.g., \( \approx 1 \) TB of data

\( O(m \cdot n) \)
Assume a graph of size 1TB.

- Disk scan: 200 MB/s $\rightarrow$ 83 minutes
- Disk seek: 1 MB/s $\rightarrow$ 11.5 days

Cost of simple breadth-first-search? $\approx$ 1 week

Cost of finding the diameter? $\approx$ years
Why are we here?

To answer simple questions...

• What is the diameter?
• Is the graph connected?
• How many connected components are there?
• What is the shortest path from A to B?
• Find a minimum spanning tree.

... when classical algorithms are too slow.
General strategies?

What can we do?
General strategies?

What can we do?

• **Optimize the algorithm.**
  – Leverage special structures (e.g., graph is planar).
  – E.g., Use caches more efficiently, keep your data better organized, etc.

• **Approximate the answer.**
  – E.g., Use random sampling.

• **Parallelize the computation.**
  – E.g., leverage multicore / multiprocessor / cluster computation
Example: Diameter

Step 1: Approximate

Goal:
Given graph G=(V,E), find approximate diameter D such that:

\[
\frac{1}{2} \cdot \text{diameter}(G) \leq D \leq 2 \cdot \text{diameter}(G)
\]

2-approximation
Example: Diameter

Simple approximation algorithm:
Example: Diameter

Simple approximation algorithm:
1. Choose a node u.
2. Run BFS from u.
3. Let v be farthest node from u.
4. Return \( d(u,v) \).

d(u,v) = 4
Simple approximation algorithm:
1. Choose a node u.
2. Run BFS from u.
3. Let v be farthest node from u.
4. Return d(u,v).

Proof:
For all pairs (x,y):
\[ d(x,y) \leq d(x,u) + d(u,y) \]
\[ \leq d(u,v) + d(u,v) \]
\[ \leq 2 \cdot d(u,v) \]
d(u,v) = 4

Example: Diameter
Example: Diameter

Simple approximation algorithm:
1. Choose a node \( u \).
2. Run BFS from \( u \).
3. Let \( v \) be farthest node from \( u \).
4. Return \( d(u,v) \).

Time:
\( O(m + n) \)
Example: Diameter

Simple approximation algorithm:
1. Choose a node $u$.
2. Run BFS from $u$.
3. Let $v$ be farthest node from $u$.
4. Return $d(u,v)$.

Time:
$O(m + n)$

Cost of simple BFS?
$\approx 1$ week
Oops…
Better, but still slow.
Example: Diameter

Step 2: Scan time is faster than seek time!

- Disk scan: 200 MB/s ➞ 83 minutes
- Disk seek: 1 MB/s ➞ 11.5 days

Goal:

Do BFS by scanning entire graph in any order.
Find log(n)-approximation.

Time: $O(n + m)$ scan time.

Speedup of 100x for log-approximate answer.
Example: Diameter

Step 3: Sampling + Approximation

Goal:

If the graph has diameter \( \leq D \) return true.
If the graph is FAR from a graph with diameter \( \leq 4D + 2 \) return true.

Time: \( O(1/\varepsilon^3) \)

error term / depends on how FAR is FAR
Step 4: Optimization + 2-Approximation

Goal:
Use faster BFS to get a 2-approximation.
Assume a cache of size $C$ with block-size $B$.

Time: $O(n + \frac{m}{B} \cdot \log_{\frac{C}{B}} (\frac{m}{B}))$

Speedup of 100x to 1000x when data is on disk.
Step 4: Parallelize + 2-Approximation

Goal:

Use parallel-BFS to find a 2-approximation.

Time: $O((m/p) \cdot \log^2 n + D \cdot \log^3 n)$ on $p$ cores.
or

Time: $n$ iterations on Map-Reduce cluster.
Another Ex: Minimum Spanning Tree

Algorithms 101

- Kruskal’s Algorithm
- Prim’s Algorithm

- Runs in $O(m \log n)$ time for $n$ nodes and $m$ edges.

- Fast enough?
Example: Minimum Spanning Tree

Special Structure

- Is graph planar?
  ➔ Then we can find an MST in $O(m)$ time.

- Is the graph a social network?
  ➔ Then graph has special structure too!
Randomization and Approximation

• Can we find a *faster* randomized algorithm?
• Approximate MSG?
• Estimate weight of MST?

Amazingly: \(O(dW \log(dW))\)
for a graph with degree \(d\) and max. edge weight \(W\)
No dependence on \(n\)!!
Streaming

- What if we only have limited access to data?
- We get to read each edge once in some arbitrary order: $e_1, e_2, e_3, \ldots, e_m$

- We can’t store the whole graph!
- Output an (approximate) MST?
Dynamic

• What if edges change over time?
• Edges are continually added and removed from our graph.
• After each change, find a new MST.
Caching

- Caching performance is critical.
- Each time we access part of the graph, a block of memory is loaded.
  - Expensive!
- How can we design an algorithm for finding an MST that uses cache efficiently?
Example: Minimum Spanning Tree

Parallel/GPU/Distributed

- Can we leverage a multicore machine to find an MST faster?
- Can we use GPUs to get faster performance?
- Can we use a distributed cluster (e.g., MapReduce/Hadoop) to find an MST faster?
New Challenges

Scale

• How do we deal with graphs that are big?
• Cannot store entire graph in memory.
• Processing time is large!
New Challenges

Where is the data?

- Data is no longer as easily accessible.
- Is data distributed?
- Is data streaming?
- Is data noisy?
New Challenges

Dynamic world

• Data is no longer static.
• Graphs change over time.
• Edges may be added and removed.
• Users may come and go.
New Challenges

Context matters

• Where did the data come from?
• Is it from a social network?
• Is it from a wireless network?
• Is it from a game?
• How can we leverage the structure to do better?
Why are we here?

To answer simple questions...

• What is the diameter?
• Is the graph connected?
• How many connected components are there?
• What is the shortest path from A to B?
• Find a minimum spanning tree.

... when classical algorithms are too slow by...

• Optimizing the algorithm.
  – Leverage special structure (e.g., graph is planar).
  – Use caches more efficiently, keep your data better organized, etc.
• Approximating the answer.
  – E.g., Use random sampling.
• Parallelizing the computation.
  – E.g., leverage multicore / multiprocessor / cluster computation
This is a class about algorithms.
This is a class about algorithms.
This is a class about algorithms.

The goal is to deeply understand the algorithms we are studying.

How do they work?

Why do they work?

How do you implement them?

What are the underlying techniques?

What are the trade-offs?
Previous Student feedback:

“What a great class, but so many algorithms. I wish I had known that the class was going to include so many proofs and so much math.”
-- Gil Gunderson

“I loved the class, it was amazing, the algorithms were almost like magic! But unfortunately I had no idea what you were talking about most of the time, since I had never taken an algorithms class before.”
-- Ernie Macmillan

** The above quotes are entirely fabricated and only loosely reflect real student feedback.
For example, if I say:

“To find the shortest path, first use Prim’s algorithm (with a Fibonacci Heap) to find an MST in $O(m + n \log n)$ time.”

You should have two immediate reactions:
For example, if I say:

“To find the shortest path, first use *Prim’s algorithm* (with a *Fibonacci Heap*) to find an *MST* in $O(m + n \log n)$ time.”

You should have two immediate reactions:

1. Nod your head since you understand exactly what I am saying.
For example, if I say:

“To find the shortest path, first use Prim’s algorithm (with a Fibonacci Heap) to find an MST in $O(m + n \log n)$ time.”

You should have two immediate reactions:

1. Nod your head since you understand exactly what I am saying.

2. Tell me that this claim seems like NONSENSE (because MSTs are almost never useful for finding a shortest path).
CS5234: Algorithms at Scale

Target students:
- Advanced (3rd or 4th year) undergraduates
- Master’s students
- PhD students
- Interested in algorithms
- Interested in tools for solving hard problems

Prerequisites:
- CS3230 (Analysis of Algorithms)
- Mathematical fundamentals
"If you need your software to run twice as fast, hire better programmers.

But if you need your software to run more than twice as fast, use a better algorithm."

-- Software Lead at Microsoft
“... pleasure has probably been the main goal all along.

-- D. E. Knuth
CS5234 Overview

- **Mid-term exam**
  
  October 16  In class, Week 9

- **Final exam**
  
  November 27

(Please double-check the official schedule in case it changes or in case this is wrong!)
CS5234 Overview

- Lecture
  Thursday  6:30-8:30pm

- Extra time
  Thursday  8:30-9:30pm

Extra time will be used for discussion, reviewing problem sets, answering questions, solving riddles, doing crossword puzzles, eating cookies, etc.
CS5234 Overview

- **Assessments**
  - Problem sets + Mini-Project
  - Mid-term exam
  - Final exam

- **Problem sets**
  - 5-6 sets (roughly every week)
  - Focused on algorithm design and analysis.
  - Perhaps a few will require coding.
Mini-Project

Small project

Idea: put together some of the different ideas we have used in the class.

Time scale: last 4 weeks of the semester.
CS5234 Overview

Survey: Google form.
On the web page.
What is your background?
Not more than 10 minutes.

PS1: Released tomorrow.
CS5234 Overview

Problem set grading

Simple scheme:

3 : excellent, perfect answer
2 : satisfactory, mostly right
1 : many mistakes / poorly written
0 : mostly wrong / not handed in
-1 : utter nonsense
What to submit:

Concise and precise answers:
Solutions should be rigorous, containing all necessary detail, but no more.

Algorithm descriptions consist of:
1. Summary of results/claims.
2. Description of algorithm in English.
3. Pseudocode, if helpful.
4. Worked example of algorithm.
5. Diagram / picture.
CS5234 Overview

How to draw pictures?

By hand:

Either submit hardcopy, or scan, or take a picture with your phone!

Or use a tablet / iPad...

Digitally:

1. xfig (ugh)
2. OmniGraffle (mac)
3. Powerpoint (hmmm)
4. ???
CS5234 Overview

Policy on plagiarism:

Do your work yourself:
Your submission should be unique, unlike anything else submitted, on the web, etc.

Discuss with other students:
1. Discuss general approach and techniques.
2. Do not take notes.
3. Spend 30 minutes on facebook (or equiv.).
4. Write up solution on your own.
5. List all collaborators.

Do not search for solutions on the web:
Use web to learn techniques and to review material from class.
CS5234 Overview

Policy on plagiarism:

Penalized severely:
First offense: minimum of one letter grade lost on final grade for class (or referral to SoC disciplinary committee).

Second offense: F for the class and/or referral to SoC.

Do not copy/compare solutions!
Algorithms Review

Introduction to Algorithms

– Cormen, Leiserson, Rivest, Stein
Algorithms Review

Algorithm Design

– Kleinberg and Tardos
Topics (tentative)

- Sampling and Sketching Very Big Graphs
- Efficient Algorithms for Modern Machines

A modern twist on classic problems...

BFS, DFS, MST, Shortest Path, etc.
Topics (tentative)

- Sampling and Sketching Very Big Graphs
  Part 1: Graph properties in less than linear time
  - Connectivity
  - Connected components
  - Minimum spanning tree
  - Average degree
  - Approximate diameter
  - Matching
Topics (tentative)

- Sampling and Sketching Very Big Graphs
  Part 2: Sketches and streams
  - Sampling from a stream
  - L0-samplers
  - Graph sketches
  - Connectivity
  - Minimum spanning trees
  - Triangle counting
Topics (tentative)

Efficient Algorithms for Modern Machines

Part 3: Caching

Cache-efficient algorithms

BFS

Priority queues

Shortest path

Minimum spanning trees
Topics (tentative)

Efficient Algorithms for Modern Machines

Part 4: Parallel Algorithms

- Fork-join parallelism
- Map-Reduce
- BFS / DFS
- Shortest path
CS5234 Overview

- **Webpage:**
  http://www.comp.nus.edu.sg/~gilbert/CS5234

- **Instructor:** Seth Gilbert
  Office: COM2-323
  Office hours: by appointment
Welcome!

Puzzle of the Day:

Connect the dots.
Use ?? straight lines.
Don’t lift your pen from the paper.

5 x 4:
Use 7 lines.

7 x 7:
Use 12 lines

Can you end at the same place you began?