CSE 373: Data Structures and Algorithms

Memory and Locality

Autumn 2018

Shrirang (Shri) Mare
shri@cs.washington.edu

Thanks to Kasey Champion, Ben Jones, Adam Blank, Michael Lee, Evan McCarty, Robbie Weber, Whitaker Brand, Zora Fung, Stuart Reges, Justin Hsia, Ruth Anderson, and many others for sample slides and materials ...

Announcements

Homework 6 is out. Individual assignment. Due Nov 30 at noon.

My office hours are today 5-6pm (moved from 4-5pm)

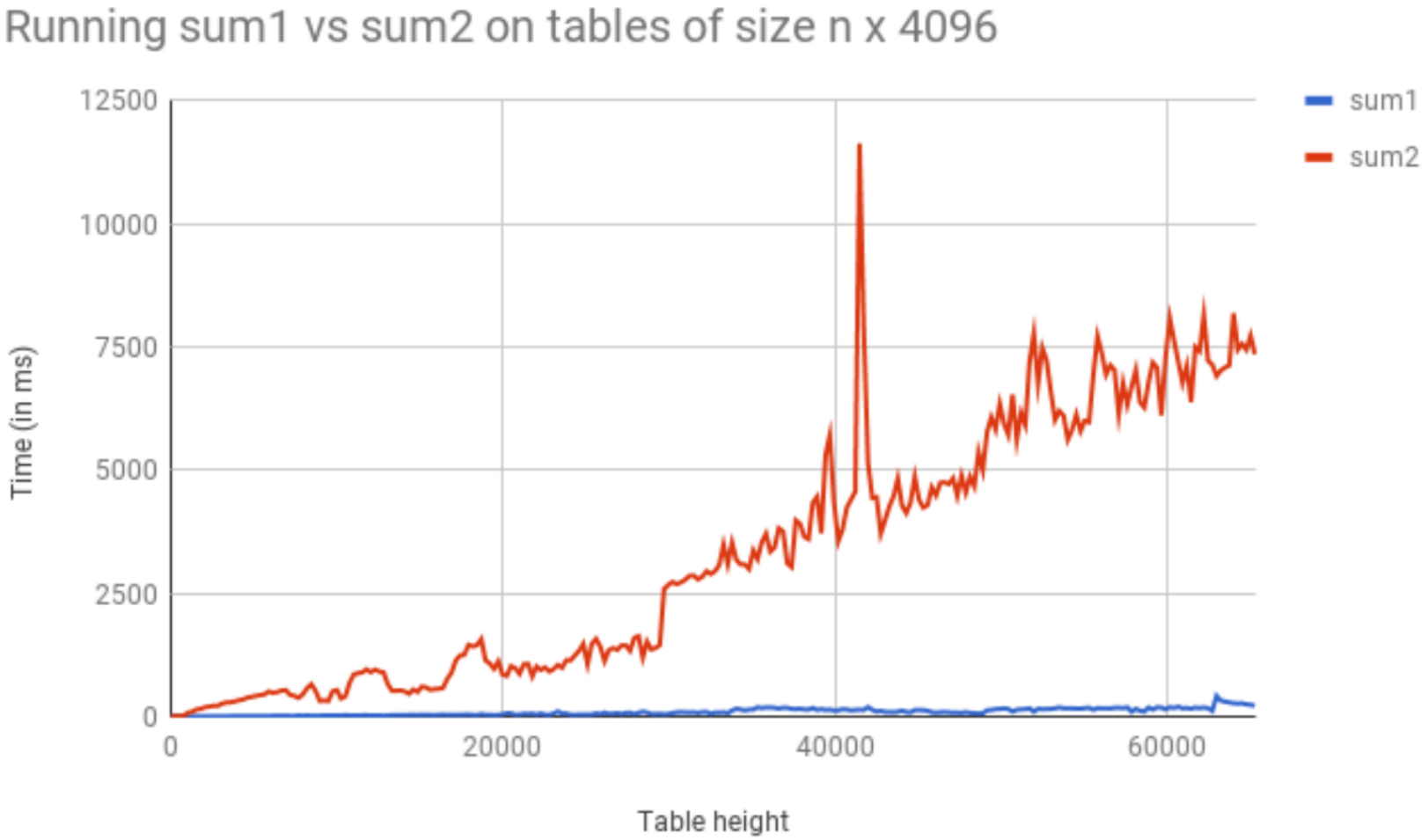
Question

What is the big-theta bound for each of these methods? $\Theta(m \cdot n)$

```
1  public int sum1(int n, int m, int[][] table) {
2      int output = 0;
3      for (int i = 0; i < n; i++) {
4          for (int j = 0; j < m; j++) {
5              output += table[i][j];
6          }
7      }
8      return output;
9  }
```

```
1  public int sum2(int n, int m, int[][] table) {
2      int output = 0;
3      for (int i = 0; i < m; i++) {
4          for (int j = 0; j < n; j++) {
5              output += table[j][i];
6          }
7      }
8      return output;
9  }
```

Comparing sum1 vs. sum2



Assumptions

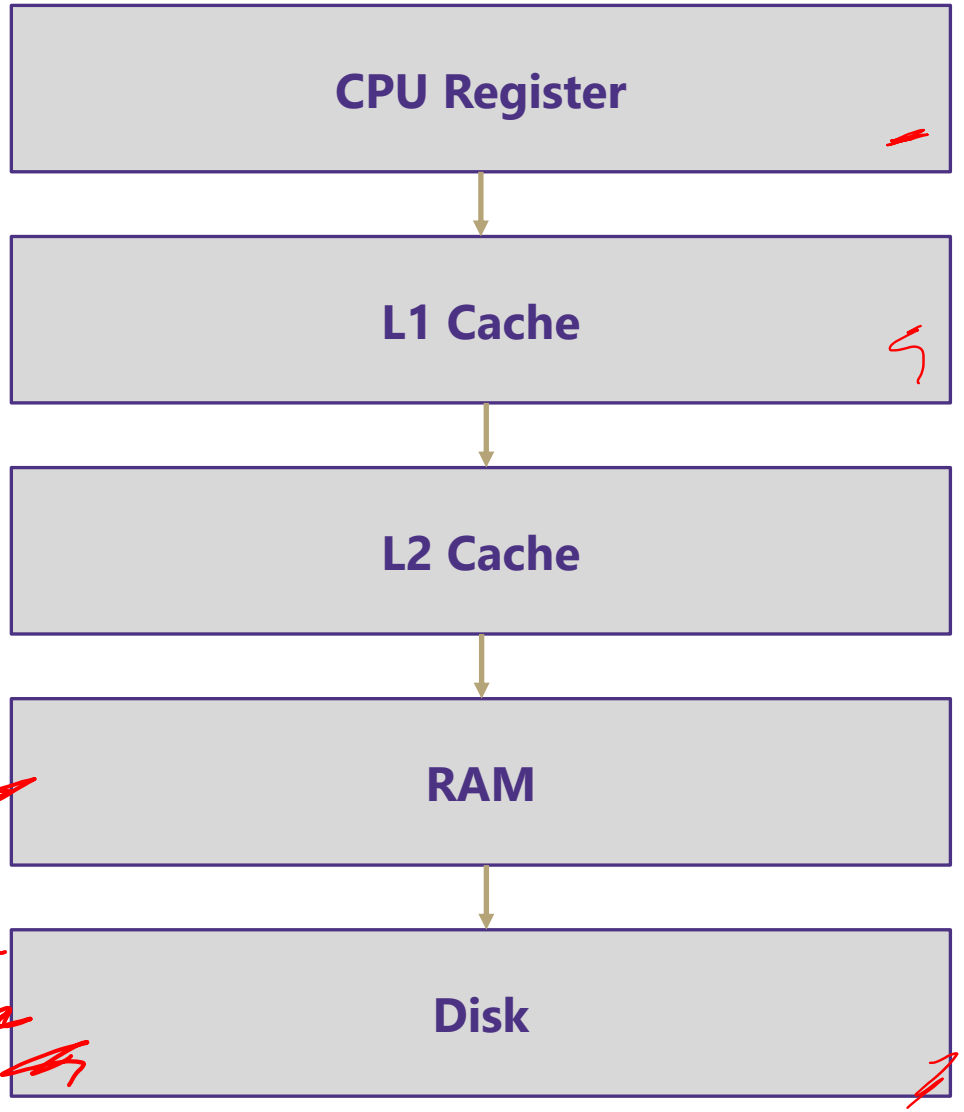
Accessing memory is a quick and constant-time operation

- Not quite true

Sometimes accessing memory is cheaper and easier than at other times

Sometimes accessing memory is very slow

Memory Architecture



What is it?	Typical Size	Time
The brain of the computer!	32 bits	≈free
Extra memory to make accessing it faster	128KB	0.5 ns
Extra memory to make accessing it faster	2MB	7 ns
Working memory, what your programs need	8GB	100 ns
Large, longtime storage	1 TB	8,000,000 ns

Memory Architecture Takeaways

The more memory a layer can store, the slower it is (generally)

Accessing the disk is **very** slow

It is much faster to do:	Than:
- `5 million arithmetic operations`	`1 disk access`
- `2500 L2 cache accesses`	`1 disk access`
- `400 main memory accesses`	`1 disk access`

Why are computers built this way?

- Physical realities (speed of light, closeness to CPU)
- Cost (price per byte of different technologies)
- Disks get much bigger not much faster

How does data move up the hierarchy?

Moving data up the memory hierarchy is slow because of latency (think distance-to-travel)

How can we make things efficient here?

- Move as much data as we can during each trip (think carpooling)
- Cost of bringing 1 byte vs several bytes is the same
- Which additional data to move? Move nearby data, because it is easy to find and likely to be asked soon.

- Keep data in cache as long as possible.
- When a value is accessed, it is more likely to be accessed again in the near future (more likely than some random value)

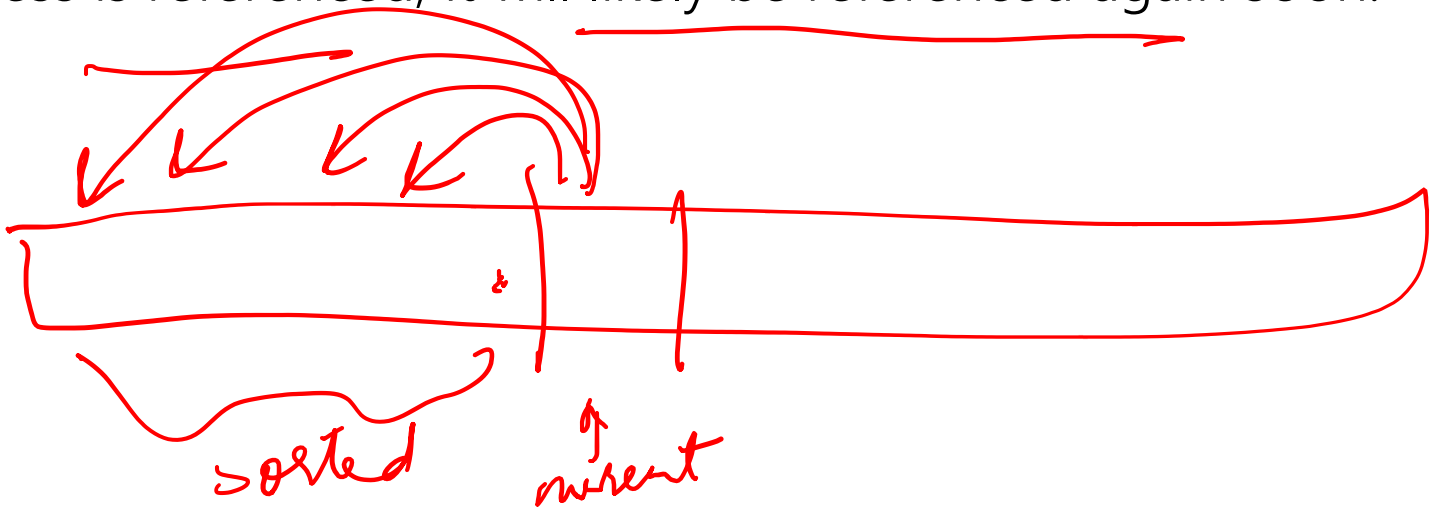
Locality

Spatial Locality (locality in **space**)

If an address is referenced, addresses that are close by will tend to be referenced soon

Temporal Locality (locality in **time**)

If an address is referenced, it will likely be referenced again soon.



Moving Memory

Amount of memory moved from **disk** to **RAM**

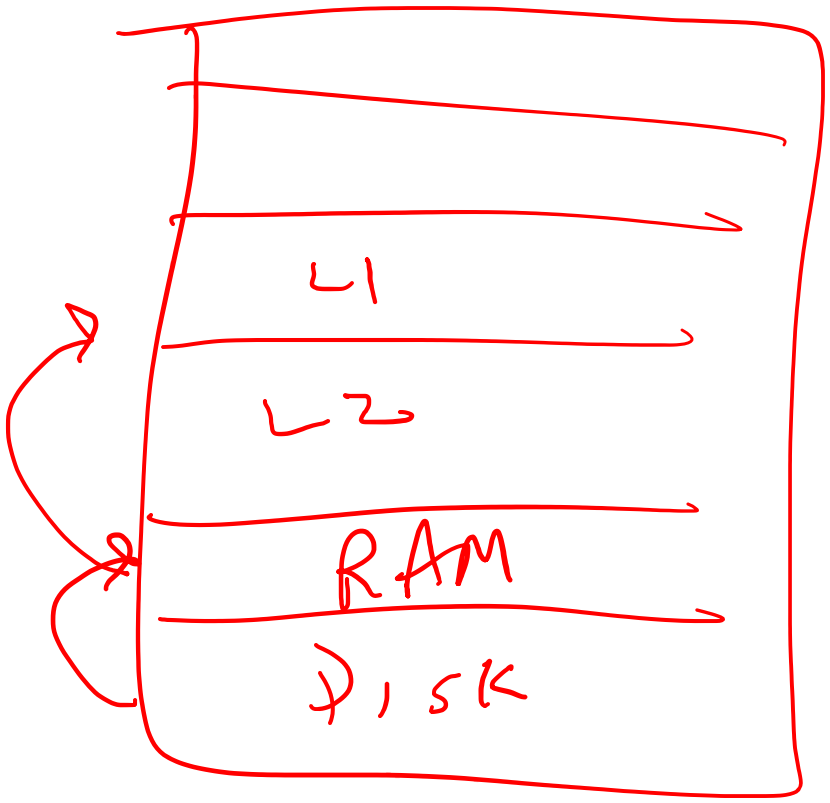
- Called a "**block**" or "**page**"
  - ≈ 4KB (kilo bytes)
  - Smallest unit of data on disk

Amount of memory moved from **RAM** to **Cache**

- called a "**cache line**"
  - ≈ 64 B (bytes)

Operating System is the Memory Boss

- controls page and cache line size
- decides when to move data to cache or evict



Revisiting our warm up question

```
1  public int sum1(int n, int m, int[][] table) {
2     int output = 0;
3     for (int i = 0; i < n; i++) {
4        for (int j = 0; j < m; j++) {
5           output += table[i][j];
6        }
7     }
8     return output;
9  }
```

```
1  public int sum2(int n, int m, int[][] table) {
2     int output = 0;
3     for (int i = 0; i < m; i++) {
4        for (int j = 0; j < n; j++) {
5           output += table[j][i];
6        }
7     }
8     return output;
9  }
```

Why does sum1 run so much faster than sum2?
sum1 takes advantage of spatial and temporal locality

10	21	2	3	61	72	1	0	41			

Memory representation of 2D arrays

Java and Memory

What happens when you use the "**new**" keyword in Java?

- Your program asks the **J**ava **V**irtual **M**achine for more memory from the "heap"
  - Pile of recently used memory
- If necessary the JVM asks Operating System for more memory
  - Hardware can only allocate in units of page
  - If you want 100 bytes you get 4kb
  - Each page is contiguous

What happens when you create a new array?

- Program asks JVM for one long, contiguous chunk of memory

What happens when you create a new object?

- Program asks the JVM for any random place in memory

What happens when you read an array index?

- Program asks JVM for the address, JVM hands off to OS
- OS checks the L1 caches, the L2 caches then RAM then disk to find it
- If data is found, OS loads it into caches to speed up future lookups

What happens when we open and read data from a file?

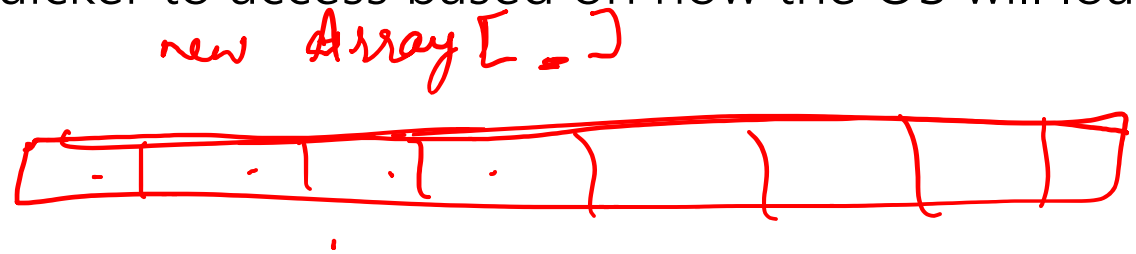
- Files are always stored on disk, must make a disk access

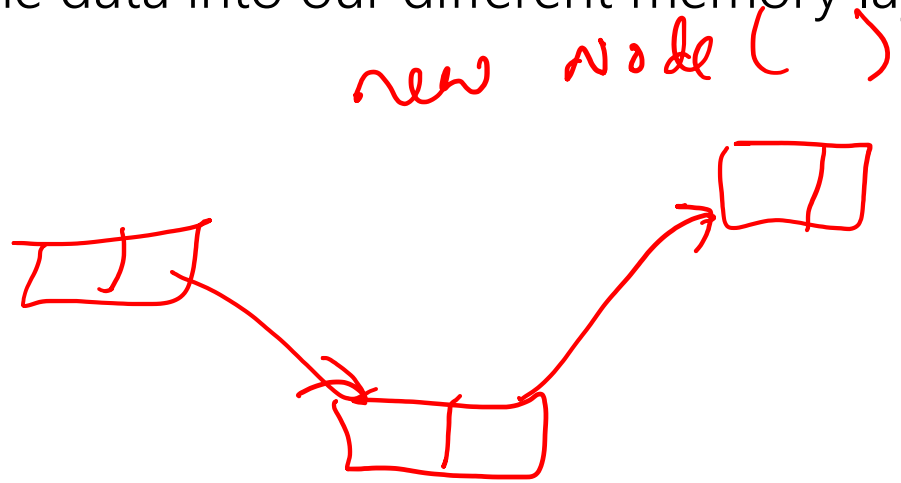
Array v Linked List

Is iterating over an ArrayList faster than iterating over a LinkedList?

Answer:

LinkedList nodes can be stored in memory, which means the don't have spatial locality. The ArrayList is more likely to be stored in contiguous regions of memory, so it should be quicker to access based on how the OS will load the data into our different memory layers.





Worksheet questions 1-2

Framing/modeling problems as graphs

General framework to approach problems

1. Does this problem look like any other problem I know how to solve

2. If answer to (1) isn't clear, can I change the problem so it becomes a problem I know how to solve ("reduction")

Applying this in the context of graphs:

1. Is the question asking about a map or a social network. If so, represent as a graph.

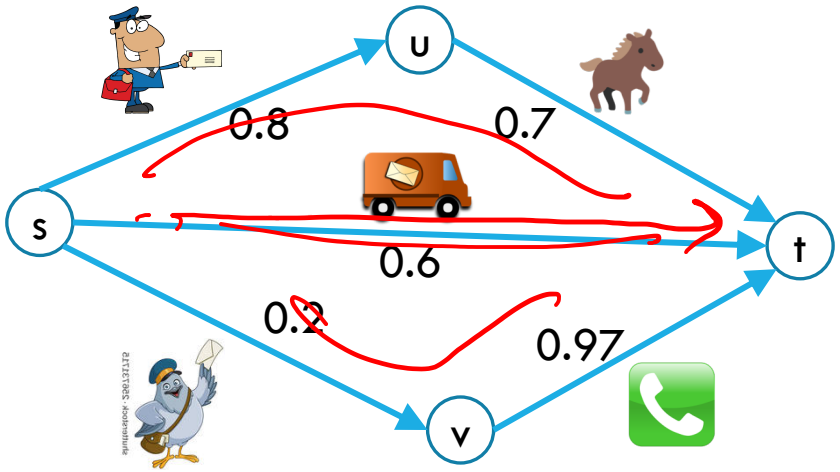
2. Otherwise, can convert the problem to a graph problem (what are vertices? How do I represent/encode relationships as edges?)

3. Is the problem "encoded" in the graph? If not, go back to (2). If yes, is this one of the 4 classic graph problems we know. Can we apply one of the graph algorithms we know?

4. Can I optimize my graph at all? (e.g., add dummy nodes, add edges, reverse edges)

Problem 1: Maximum probability path

I have a message I need to get from point s to point t.
But the connections are unreliable.
What path should I send the message along so it has the best chance of arriving?



$0.8 \times 0.7 = 0.56$

0.6

$0.2 \quad 0.97 \quad \approx 0.2$

Maximum Probability Path

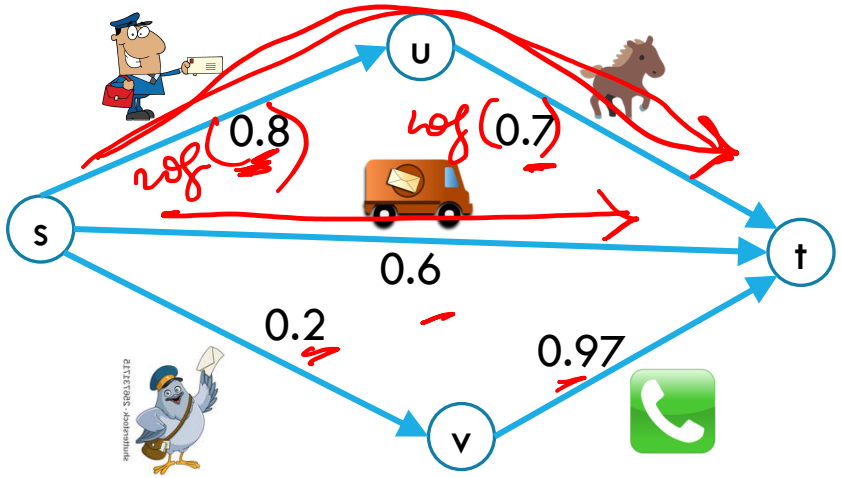
Given: a directed graph G, where each edge weight is the probability of successfully transmitting a message across that edge
Find: the path from s to t with maximum probability of message transmission

Looks like an application of shortest path

Let each edge's weight be the probability a message is sent successfully across the edge.

What's the probability we get our message all the way across a path?

- It's the product of the edge weights.

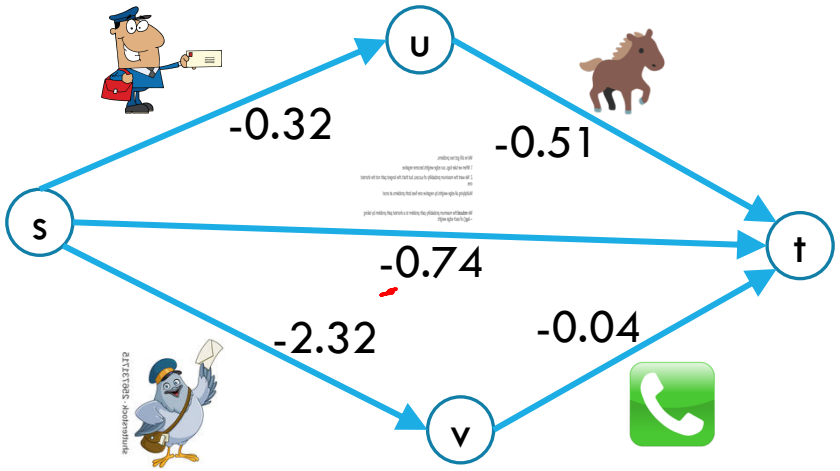


We only know how to handle sums of edge weights.

Is there a way to turn products into sums?

$$\log(ab) = \log a + \log b$$

Application of shortest path



We've still got two problems.

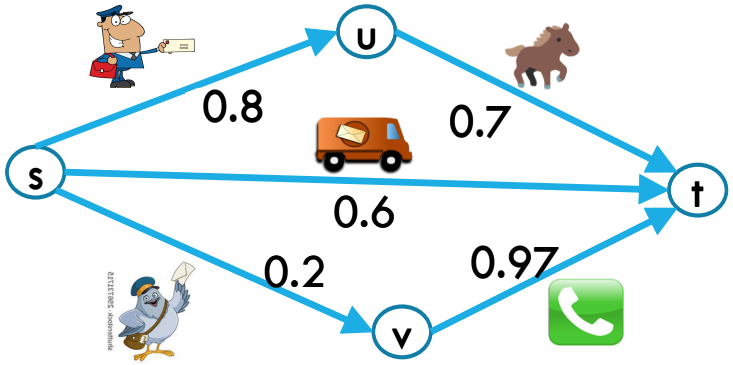
1. When we take logs, our edge weights become negative.

2. We want the *maximum* probability of success, but that's the longest path not the shortest one.

Multiplying all edge weights by negative one fixes both problems at once!

We **reduced** the maximum probability path problem to a shortest path problem by taking $-\log()$ of each edge weight.

Maximum Probability Path Reduction



Transform Input

Weighted Shortest Paths

Transform Output

Topological sort and SCC takeaways

Review

If (and only if) your graph is a DAG, you can find a topological sort of your graph.

Finding SCCs lets you **collapse** your graph to the meta-structure.

The collapsed graph will always be a DAG.

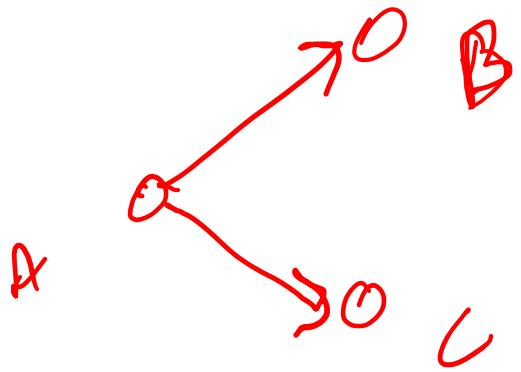
Both of these algorithms run in linear time.

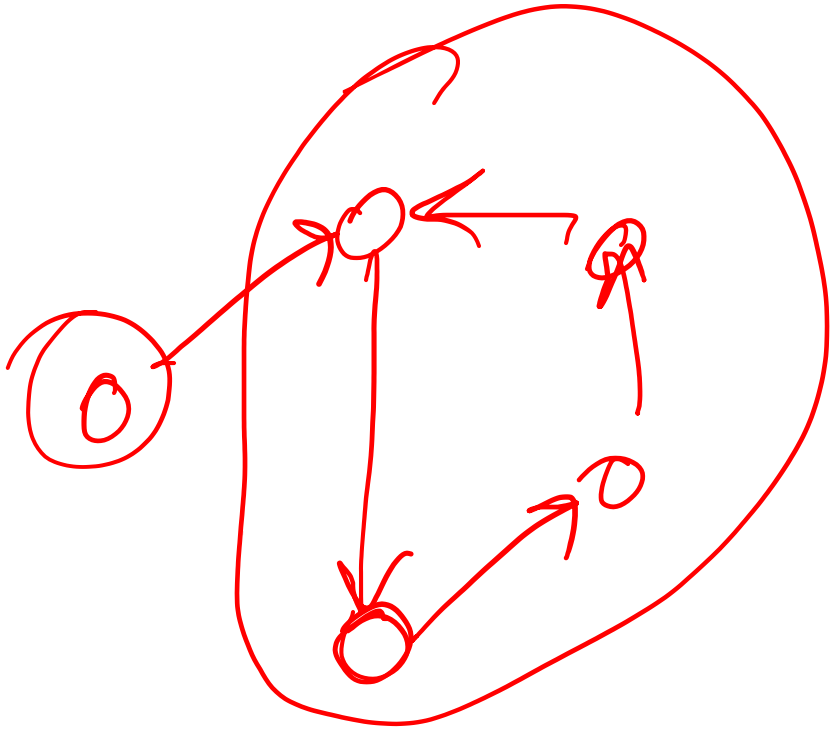
Just about everything you could want to do with your graph will take at least as long.

You should think of these as **"almost free" preprocessing** of your graph.

- Your other graph algorithms only need to work on
  - topologically sorted graphs and
  - strongly connected graphs.

Worksheet questions 3-4





Problem 2: Creating review session

We have a long list of types of problems we might want to cover in the final review session.

- Heap insertion problem, big-O problems, finding closed forms of recurrences, testing…

To try to make you all happy, we might ask for your preferences. Each of you gives us two preferences of the form "I [do/don't] want a [] problem in the review"

We'll assume you'll be happy if you get at least one of your two preferences.

Review Creation Problem

Given: A list of 2 preferences per student.

Find: A set of questions so every student gets at least one of their preferences (or accurately report no such question set exists).

Review Creation: Take 1

We have Q kinds of questions and S students.

What if we try every possible combination of questions.

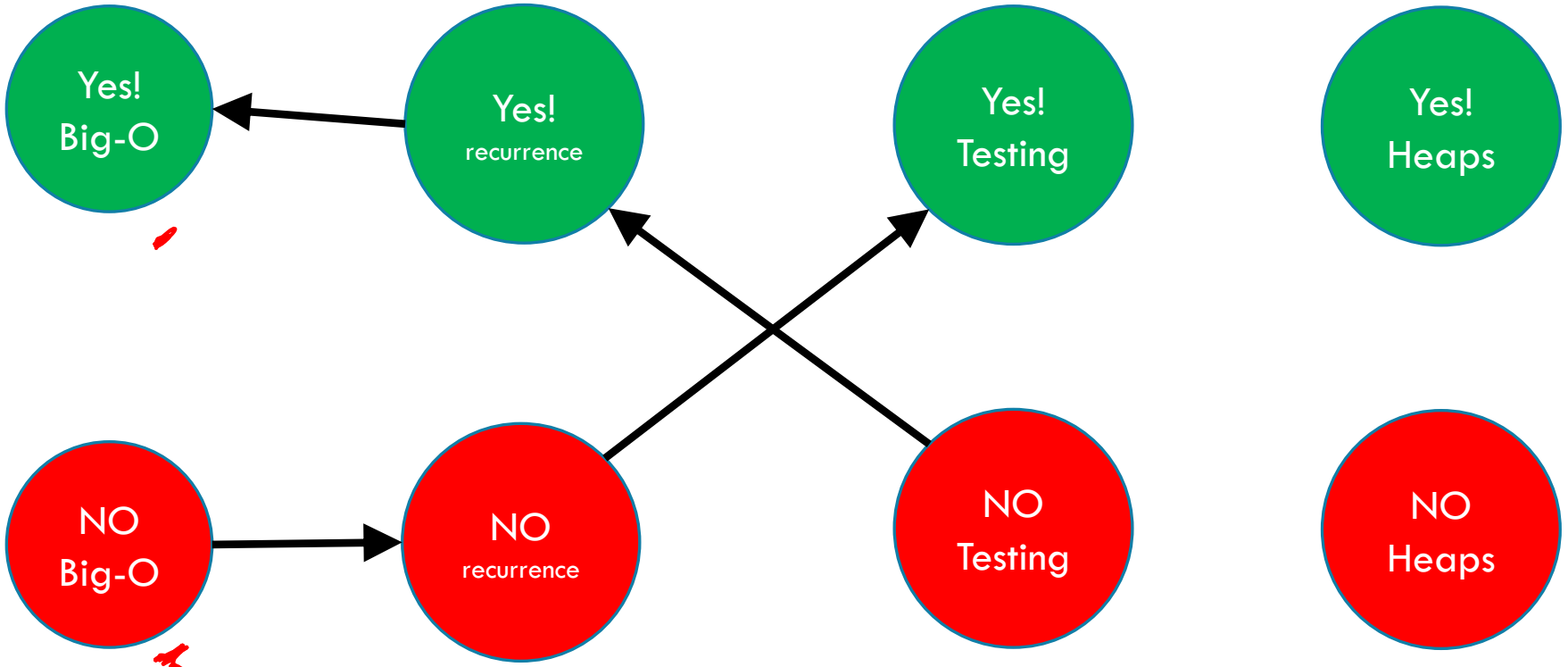
How long does this take? $O(2^Q S)$

If we have a lot of questions, that's **really** slow.

Review Creation: Take 2

Each student introduces new relationships for data:

Let's say your preferences are represented by this table:



Problem	YES	NO
Big-O	X	
Recurrence		X
Testing		
Heaps		

Problem	YES	NO
Big-O		
Recurrence	X	
Testing	X	
Heaps		

If we don't include a big-O proof, can you still be happy?

If we do include a recurrence can you still be happy?

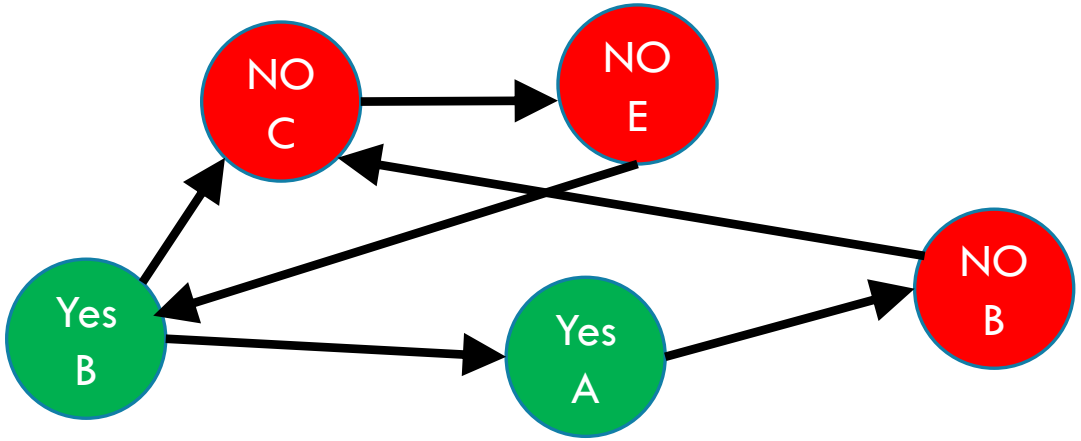
Review Creation: Take 2

Hey we made a graph!

What do the edges mean?

- We need to avoid an edge that goes TRUE THING → FALSE THING

Let's think about a single SCC of the graph.



Can we have a true and false statement in the same SCC?

What happens now that Yes B and NO B are in the same SCC?

Review Creation: SCCs

The vertices of a SCC must either be all true or all false.

Algorithm Step 1: Run SCC on the graph. Check that each question-type-pair are in different SCC.

Now what? Every SCC gets the same value.

- Treat it as a single object!

We want to avoid edges from true things to false things.

- "Trues" seem more useful for us at the end.

Is there some way to start from the end?

YES! Topological Sort

Making the review problems

Algorithm:
Make the requirements graph.

Find the SCCs.

If any SCC has including and not including a problem, we can't make the review session.

Run topological sort on the graph of SCC.

Starting from the end:

- if everything in a component is unassigned, set them to true, and set their opposites to false.
- Else If one thing in a component is assigned, assign the same value to the rest of the nodes in the component and the opposite value to their opposites.

This works!!

How fast is it?

O(Q + S). That's a HUGE improvement.

Some More Context

The Final Making Problem was a type of "Satisfiability" (SAT) problem.

We had a bunch of variables (include/exclude this question), and needed to satisfy everything in a list of requirements.

SAT is a general way to encode lots of hard problems.

Because every requirement was "do at least one of these 2" this was a 2-SAT instance.

If we change the 2 into a 3, no one knows an algorithm that runs efficiently.

And finding one (or proving one doesn't exist) has a $1,000,000 prize.

If we get to P vs. NP at the end of the quarter we'll discuss this more.

Reference slides

Binary, Bits and Bytes

Review

binary

A base-2 system of representing numbers using only 1s and 0s

- vs decimal, base 10, which has 9 symbols

bit

The smallest unit of computer memory represented as a single binary value either 0 or 1

byte

The most commonly referred to unit of memory, a grouping of 8 bits

Can represent 265 different numbers (28)

1 Kilobyte = 1 thousand bytes (KB)

1 Megabyte = 1 million bytes (MB)

1 Gigabyte = 1 billion bytes (GB)

Decimal	Decimal Break Down	Binary	Binary Break Down
0	$(0 * 10^0)$	0	$(0 * 2^0)$
1	$(1 * 10^0)$	1	$(1 * 2^0)$
10	$(1 * 10^1) + (0 * 10^0)$	1010	$(1 * 2^3) + (0 * 2^2) + (1 * 2^1) + (0 * 2^0)$
12	$(1 * 10^1) + (2 * 10^0)$	1100	$(1 * 2^3) + (1 * 2^2) + (0 * 2^1) + (0 * 2^0)$
127	$(1 * 10^2) + (1 * 10^1) + (2 * 10^0)$	01111111	$(0 * 2^7) + (1 * 2^6) + (1 * 2^5) + (1 * 2^4)(1 * 2^3) + (1 * 2^2) + (1 * 2^1) + (1 * 2^0)$