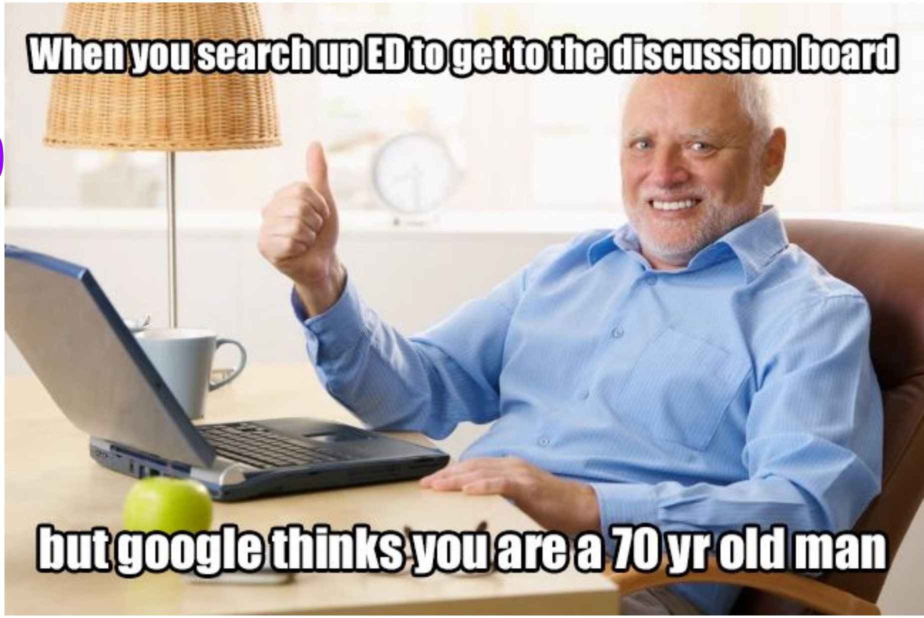
CSE 373 SP1057 Section 10

Final Review


When you search up ED to get to the discussion board
but google thinks you are a 70 yr old man

Meme Credit: Michael Yung

Q1: Reductions

Reduction: Shortest Paths

- Design an efficient algorithm for the following problem: Given a weighted, directed graph G where the weights of every edge in G are all integers between 1 and 10, and a starting vertex s in G, find the distance from s to every other vertex in the graph (where the distance between two vertices is defined as the weight of the shortest path connecting them, or infinity if no such path exists).

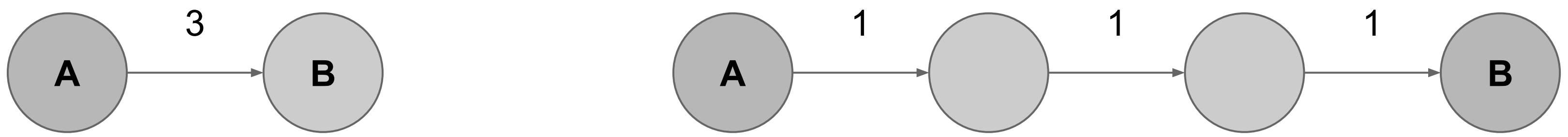
  Your algorithm **must run faster (asymptotically) than Dijkstra's**.

- You **must** show the running time of your algorithm in terms of V and E (the number of vertices and edges in the *original* graph).

Reduction: Shortest Paths

- Observation: Dijkstra and BFS process nodes in the same order if all edge weights are identical.

- Solution: For every edge e in the graph, replace e with a chain of w-1 vertices (where w is the weight of e) where the two ends of the chain are the endpoints of e. Pictorially:



Then run BFS on the modified graph, and update the dist and pred fields just like in Dijkstra. This works because we still visit the nodes in the same order, without the need for a PQ!

Reduction: Shortest Paths

- Runtime:

  The new graph has **at most 10 edges** for every edge in the original graph, and at most |V| + 9|E| vertices. So BFS runs in Θ(|V| + 9|E| + 10|E|) = **Θ (|V| + |E|)** time.

  Faster (asymptotically) than Dijkstra!

  ...since Dijkstra is **(|V| + |E|)logV** due to the need for a **PQ.**

Q2B: Design Decisions

Design Decisions: B i

Q: You are writing a program to manage a todo list with a very specific approach to tasks. This program will order tasks for someone to tackle so that the most recent task is addressed first. How would you store the transactions in appropriate order?

Main idea: Most recent task should be handled first, also known as…

L.I.F.O: Last in, first out

A: Stack -> Our go-to LIFO data structure!

Design Decisions: B ii

Q: You are writing a study support program that creates flash cards where each flash card has two pieces of crucial information: the question to be asked and the correct answer. How would you store these flash card objects?

Main idea: Questions have an associated correct answer

A: Dictionary (or map) -> We can use the question to look up the answer!

Design Decisions: B iii

Q: You are writing a program to store the history of transactions for a small business. You need to maintain the order in which the transactions occurred and be able to access entries based on the order in which they were received

Main idea: Keep the relative order, and be able to access elements

A: List -> We can get elements without manipulating the data!

Design Decisions: B iv

Q: You are writing a program to surface candidates for a job. As candidates are met and evaluated, they are added to a collection from which hiring managers can request the next best applicant whenever they have an open job position. How would you store the candidates and their evaluations for hiring managers to be able to quickly get the next best applicant?

Main idea: Quickly access the best candidate

A: Heap -> Priority = goodness of the candidate. We can return the best applicant quickly!

Design Decisions: B v

Q: You are writing a program to schedule jobs sent to a laser printer. The laser printer should process these jobs in the order in which the requests were received. How would you store the jobs in appropriate order?

Main idea: Process jobs in order received. Also known as...

F.I.F.O: First in, first out

A: Queue-> Our goto FIFO data structure

Design Decisions: B vi

Q: You are developing a video game where you play as the Roman Empire. Every time you conquer a city-state, that city-state and everything they own is added to your empire.

Main idea: Easily add everything from a city-state to belong under us.

A: Disjoint Sets -> Each city-state and the Roman Empire are sets. Whenever a city-state is taken over, add it to the Roman Empire set!

Q3: Sorting Design Decisions

Sorting Design Decisions: a

Q: **Suppose we want to sort an array containing 50 strings. Which of the following five algorithms is the best choice?**

insertion sort, merge sort, quick sort, selection sort, heap sort

Main idea:

- Only 50 strings
- When it comes to computer programs, this is not a very large input size

A: **Insertion Sort**

- Fast on small inputs
- May be worst-case Θ(n^2) but it also has a low constant factor
- Merge, quick, and heap sort may also be reasonable choices!

Sorting Design Decisions: b

Q: **Suppose we have an array containing a few hundred elements that is almost entirely sorted, apart from one or two elements that were swapped with the previous item in the list. Which of the following algorithms is the best way of sorting this data?**

insertion sort, merge sort, quick sort, selection sort, heap sort

Main idea:

- Array is already almost entirely sorted
- Only have to move a few elements, and swap them with their neighbor

A: **Insertion Sort**

- Runs in $\Theta(n)$ time for inputs that are already or close to completely sorted

Sorting Design Decisions: c

Q: **Suppose we want to sort an array of ints, but also want to minimize the amount of extra memory we want to use as much as possible. Which of the following algorithms is the best choice?**

insertion sort, merge sort, quick sort, selection sort, heap sort

Main idea:

- We don't know anything about the input
- Minimize memory!!

A: **Quick Sort**

- In place so it uses no extra memory
- Could use in-place heap sort for the same reason

Sorting Design Decisions: d

Q: **Suppose we have a version of quick sort which selects pivots randomly to create partitions. Explain how you would build an input array that would cause this version of quick sort to always run in O(n^2) time.**

What would the array look like? What happens when you try running quick sort on it?

Main idea:

- To get good performance time with quick sort, we partition the input array into roughly halves each time. For bad, O(n^2) performance we would need to do the opposite.

A: **Array with all duplicate elements**

- Each time you partition, all elements will fall on the left side of the partition, since they are <= to. We will always get O(n^2)

Sorting Design Decisions: e

Q: **How can you modify both versions from quicksort so that they no longer display O(n^2) behavior given the same inputs?**

Main idea:

- An array with all duplicate elements will always be sorted in O(n^2) time because the elements will always fall on the left of the pivot

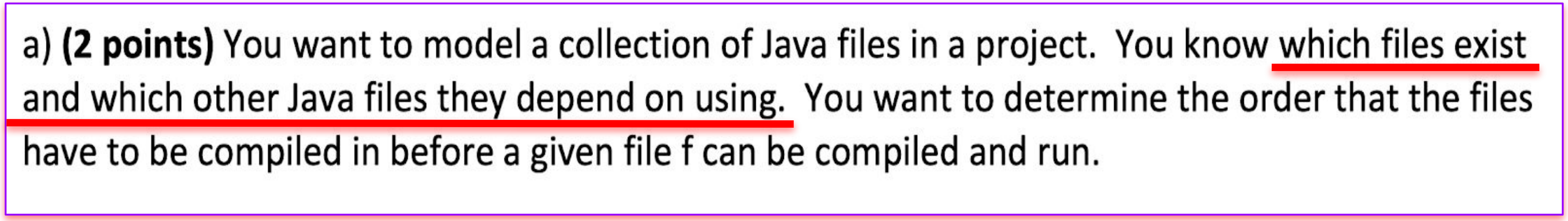
A: **Partition the array into three groups instead of two. Have the middle group represent duplicates and do not perform further quicksort on this group.**

- There would be no more work left to do for the duplicate group
- Our modified algorithm would sort an all duplicate array in O(n) time instead of O(n^2)!

Q6A: Graph Design Decisions

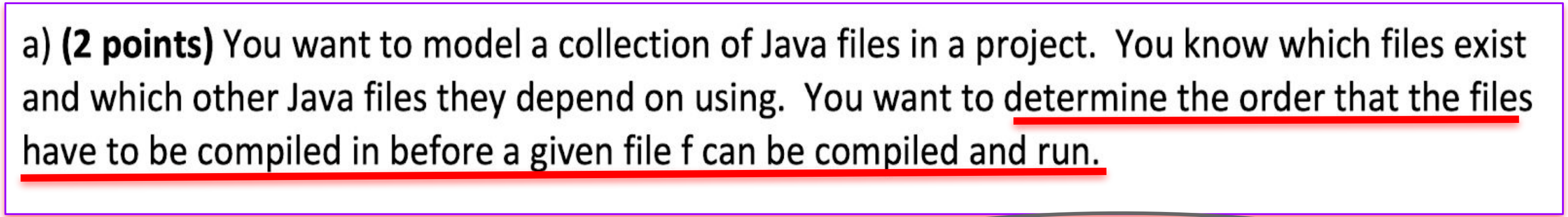
Graph Design Decisions: a

Some files depend on others. We need to figure out the order of files which need to be compiled!

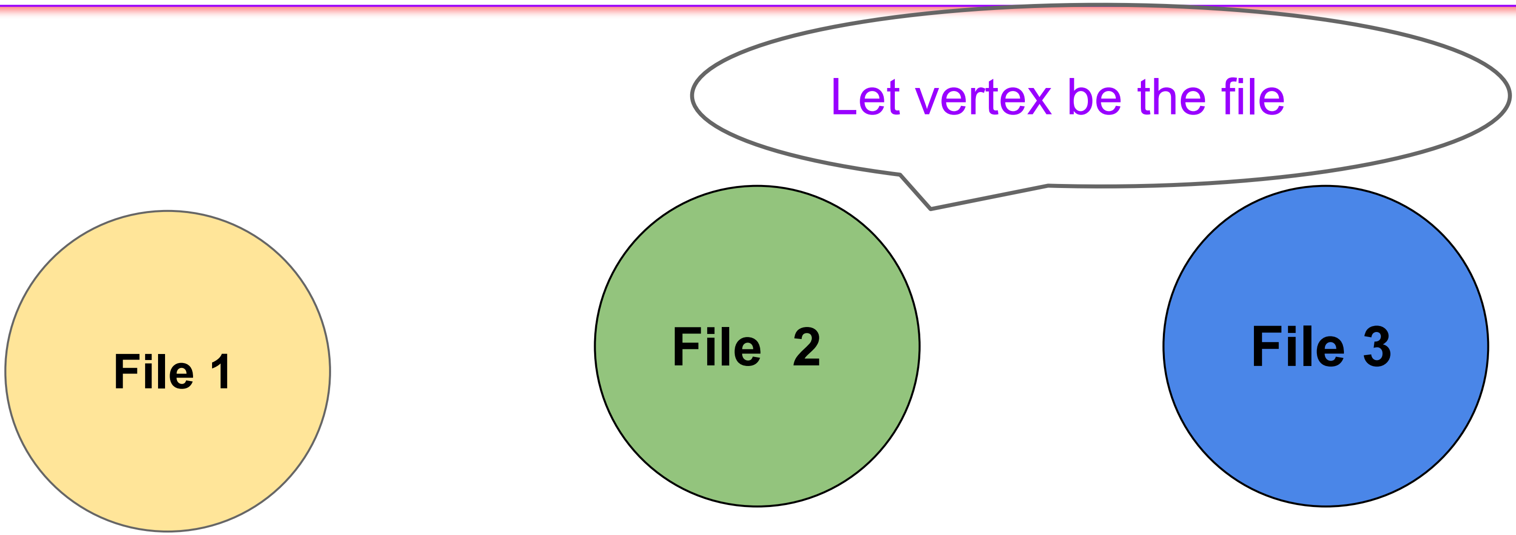


a) **(2 points)** You want to model a collection of Java files in a project. You know which files exist and which other Java files they depend on using. You want to determine the order that the files have to be compiled in before a given file f can be compiled and run.

Graph Design Decisions: a

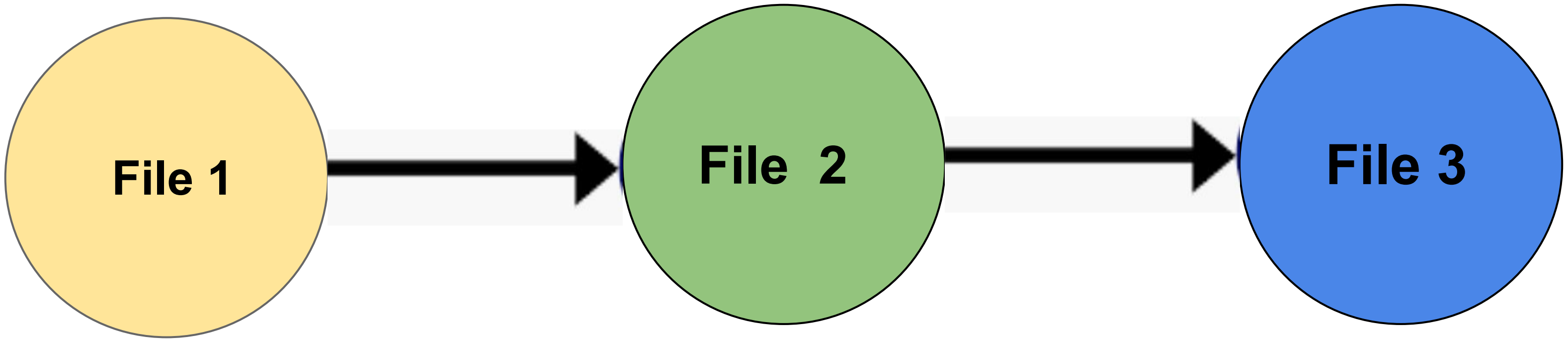


a) **(2 points)** You want to model a collection of Java files in a project. You know which files exist and which other Java files they depend on using. You want to determine the order that the files have to be compiled in before a given file f can be compiled and run.

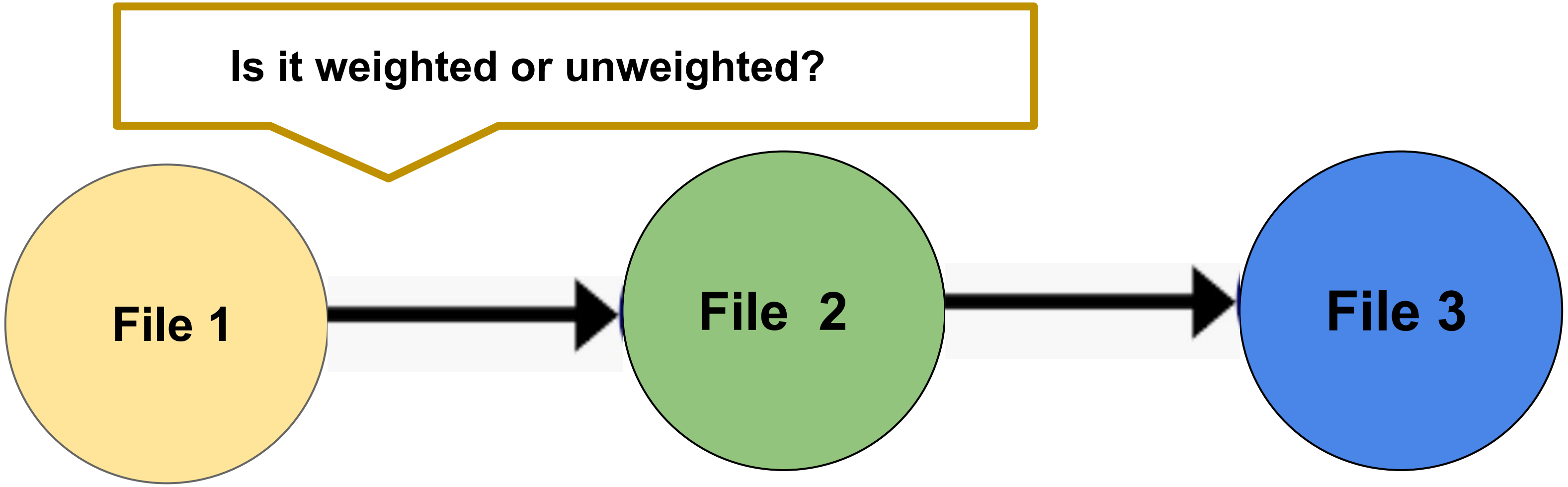


Graph Design Decisions: a

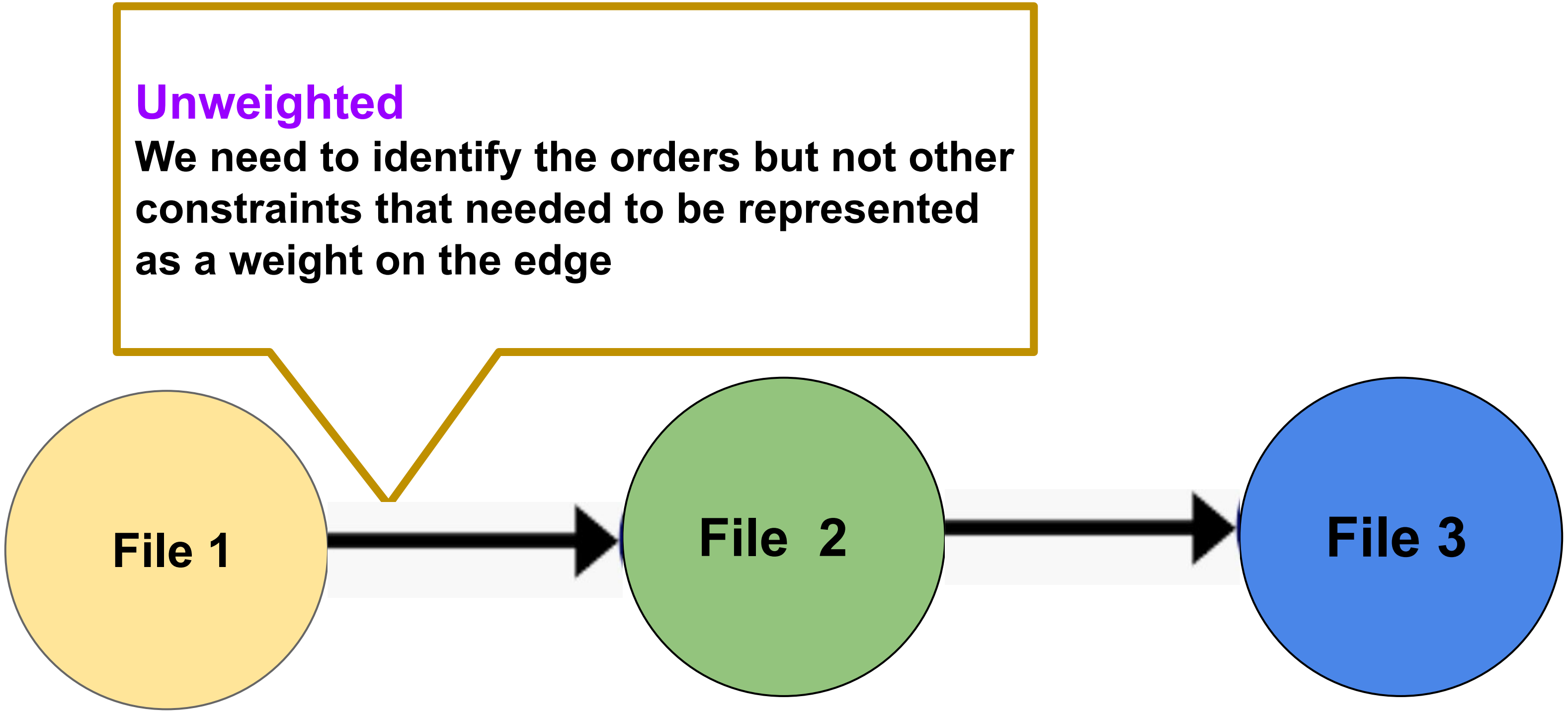
Edges are directed because we have a set of order we need to follow



Graph Design Decisions: a



Graph Design Decisions: a

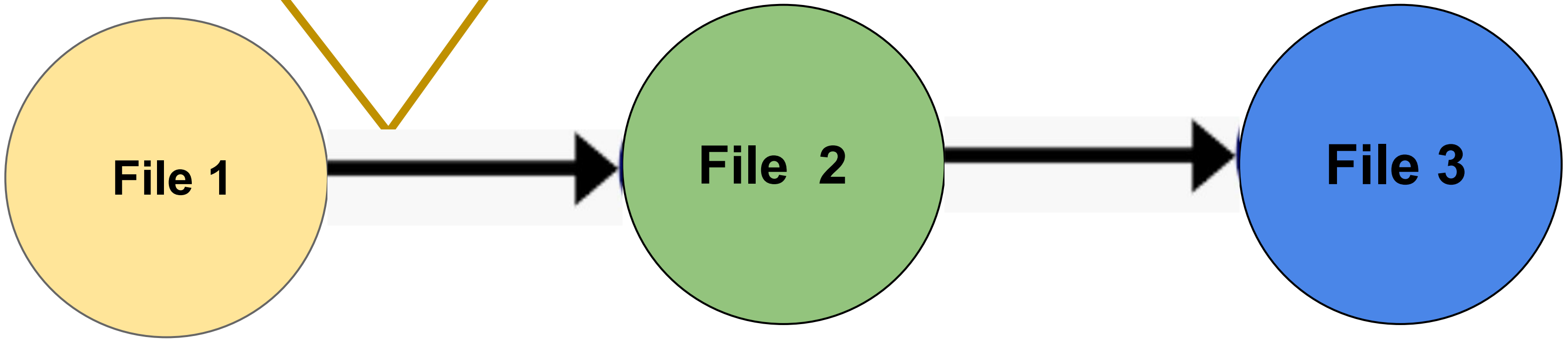


Unweighted

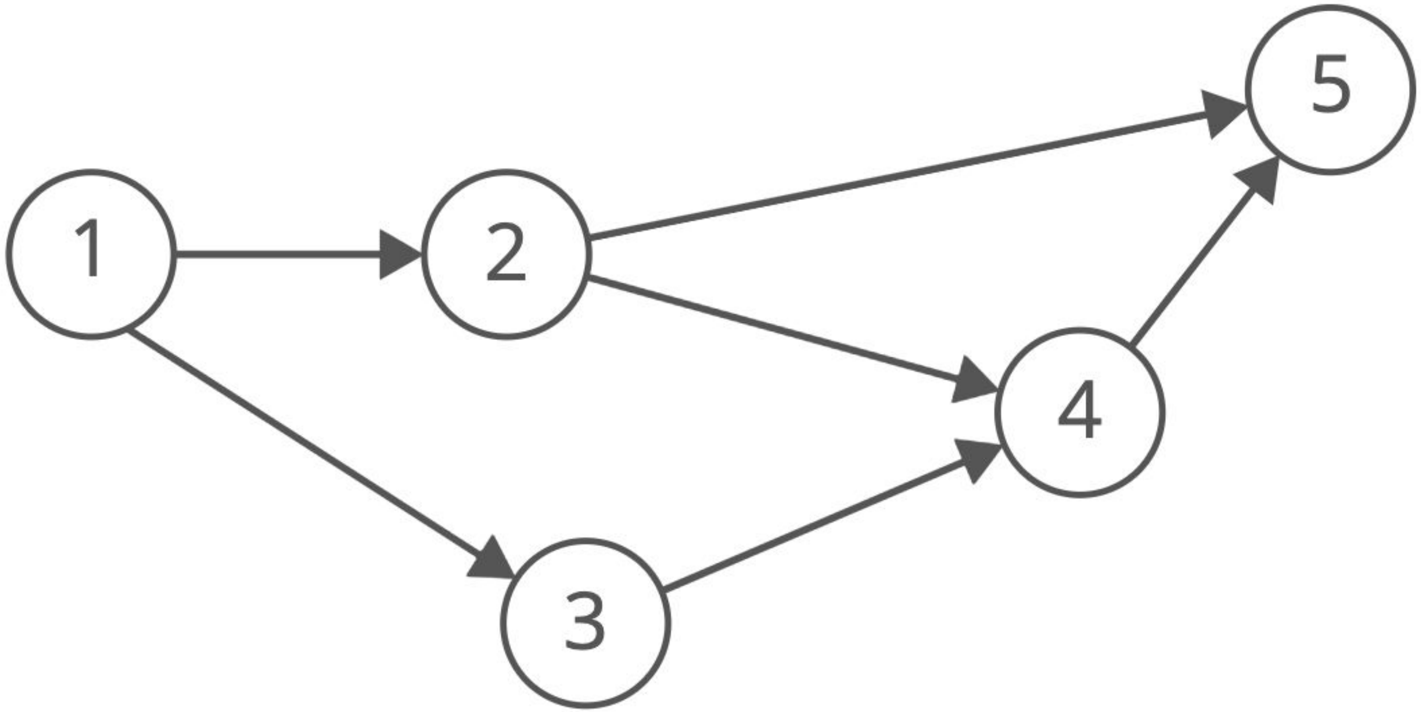
We need to identify the orders but not other constraints that needed to be represented as a weight on the edge

Graph Design Decisions: a

Is there an algorithm that we can use to identify the order that the file needs to be compiled?



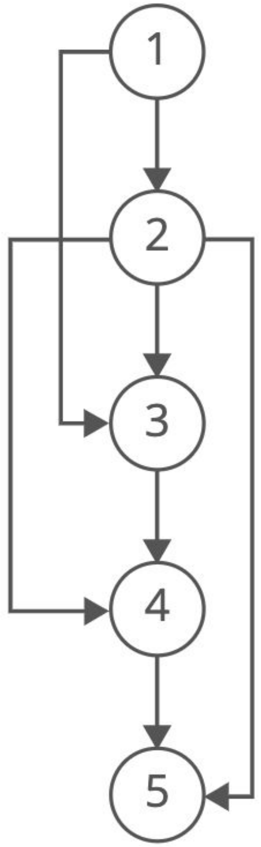
Bigger Example Graph



What We Want

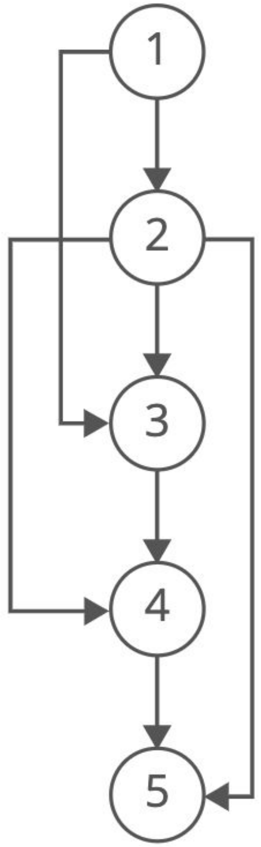
Idea: Order files (nodes) so that the edges only go in **one direction**.

This means that every file already has it's dependencies compiled.



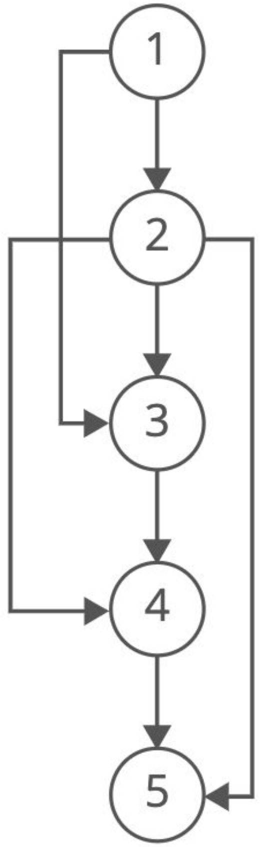
What We Want

What is this???



What We Want

Topological Sort!



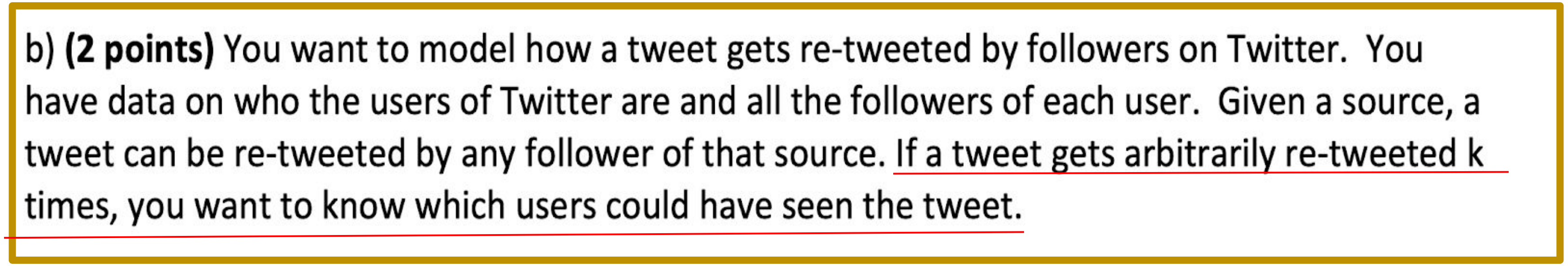
Q6B: Graph Design Decisions

Graph Design Decisions:b

b) **(2 points)** You want to model how a tweet gets re-tweeted by followers on Twitter. You have data on who the users of Twitter are and all the followers of each user. Given a source, a tweet can be re-tweeted by any follower of that source. If a tweet gets arbitrarily re-tweeted k times, you want to know which users could have seen the tweet.

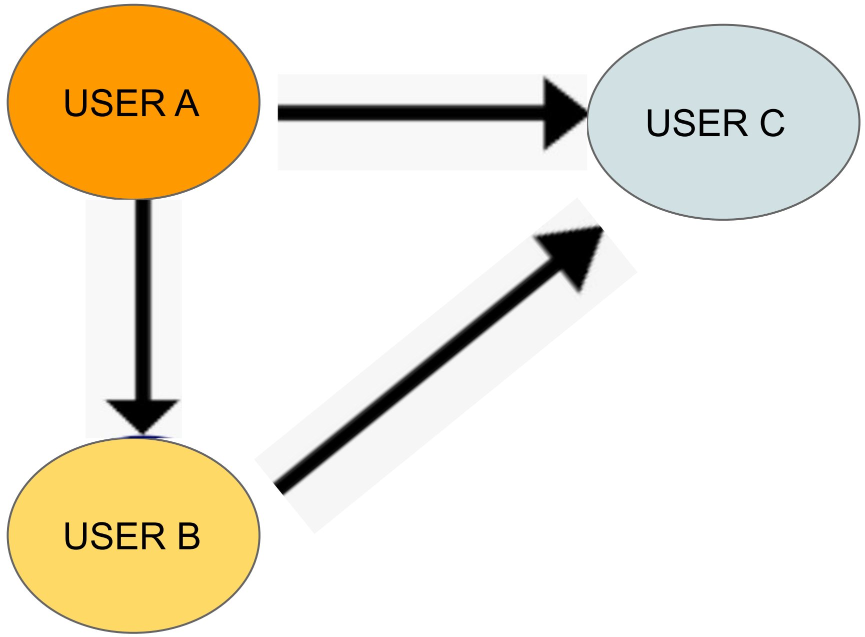
Graph Design Decisions:b

We need to remember who follow who



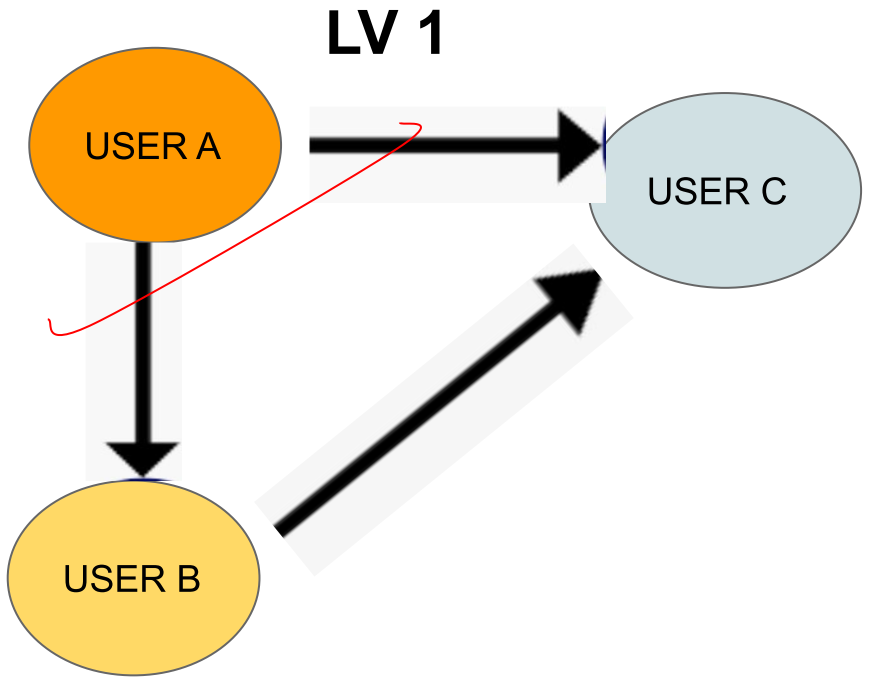
b) **(2 points)** You want to model how a tweet gets re-tweeted by followers on Twitter. You have data on who the users of Twitter are and all the followers of each user. Given a source, a tweet can be re-tweeted by any follower of that source. If a tweet gets arbitrarily re-tweeted k times, you want to know which users could have seen the tweet.

Graph Design Decisions:b



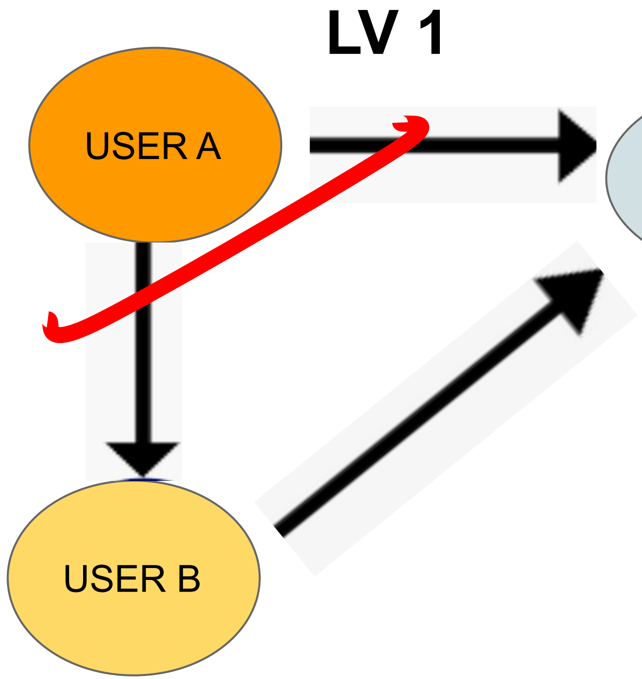
Directed Edge representing followers

Graph Design Decisions:b



Directed Edge representing followers

Graph Design Decisions:b



We can use an algorithm that are utilize neighbors!

EX) BFS, DFS

Graph Design Decisions Solution:b

Directed and unweighted graph. BFS or DFS to a level / depth of k.

Q8: Graph Modeling

Graph Modeling: a

Q: **Frodo and Sam are on their way to Mordor to destroy the ring, and along their path they have to pass through several human villages, some of which are now deserted and likely Orc territory. They learn that some paths are being heavily guarded by Orcs, and they want to avoid those paths at all costs. Friendly spies tell Frodo and Sam that they should avoid the road between two deserted villages, as it is more likely to be monitored by Orcs.**

How would you represent a graph to capture this problem?

- Vertices?
- Directed?
- Self-loops?
- Edges?
- Weighted?
- Parallel edges?

Thinking ahead: Frodo and Sam will need to find the shortest and safest path to Mordor (in part b).

Our graph model needs to support that!

Graph Modeling: a

How would you represent a graph to capture this problem?

Break down situation into key information:

1) Traveling to Mordor. Need to pass through villages.

2) Villages could be occupied or deserted.

3) Certain roads to villages could be monitored by dangerous Orcs.

4) Avoid roads between two deserted villages. Likely to have Orcs.

Vertices can be **villages**, edges can be **roads between villages**. Vertices can store information about **whether that village is occupied or deserted**.

Undirected. Frodo and Sam should be able to walk on a road in either direction.

Unweighted. No additional information needed on edges.

Allow parallel edges since there may be different roads connecting two villages.

No self-loops since they aren't useful in any way here.

Graph Modeling: b

Q: **How should Frodo and Sam find the shortest and safest path to Mordor? State the runtime of your solution.**

Graph Model

Vertices: villages and occupation status.

Edges: roads between villages.

Undirected, unweighted.

Parallel edges, no self-loops.

Shortest path algorithms?

BFS and Dijkstra's! We don't have edge weights here so BFS will work just fine. Let's pick it over Dijkstra's because it's more efficient.

Safest path algorithms?

Uhhh….

What does "safe" mean *in the context of our graph*?

Remember that friendly spies told Frodo and Sam to avoid roads between two deserted villages, since they're likely to be monitored by Orcs!

Graph Modeling: b

Q: **How should Frodo and Sam find the shortest and safest path to Mordor? State the runtime of your solution.**

Shortest path algorithm: BFS!

Safest path algorithms?

How can we ensure that Frodo and Sam avoid roads between two deserted villages?

Graph Model

- Vertices: villages and occupation status.
- Edges: roads between villages.
- Undirected, unweighted.
- Parallel edges, no self-loops.

Do some pre-processing on our graph and remove those edges! This way, when we find the shortest path, Frodo and Sam won’t be able to take dangerous roads. Run BFS (or DFS) to traverse the graph and delete all edges that connect two deserted villages.

Graph Modeling: b

Q: **How should Frodo and Sam find the shortest and safest path to Mordor? State the runtime of your solution.**

Shortest path algorithm: BFS!

Safest path algorithm: Modified BFS!

Graph Model

Vertices: villages and occupation status.

Edges: roads between villages.

Undirected, unweighted.

Parallel edges, no self-loops.

Solution: Run BFS once to remove all unsafe roads. Run BFS again to find the shortest path.

Runtime: Θ(V + E)

Q11: Find Me a Cycle

Find Me a Cycle

- Given a directed graph **G** and a node **V** in the graph, devise an algorithm that returns whether or not **V** is part of a cycle in **G.**

Find Me a Cycle: Solution

- Given a directed graph **G** and a node **V** in the graph, devise an algorithm that returns whether or not **V** is part of a cycle in **G.**

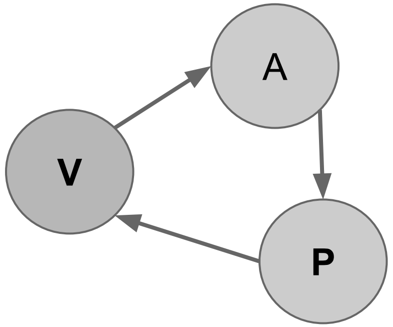
  Run BFS with **V** as the start node. When examining the neighbors of another node during BFS, if **V** is encountered again return **true**. Return **false** once BFS terminates without this happening.

Find Me a Cycle: Solution

- Run BFS with **V** as the start node. When examining the neighbors of a node during BFS, if **V** is encountered again return true. Return false once BFS terminates without this happening.

  Why?
  Any cycle involving **V** has a node **P** that comes right before **V** in the cycle. Because this is a cycle, there must also be a path from **V** to **P** in the graph.

  Because this path exists, BFS starting at **V** will visit **P**. BFS will then look at **P's** neighbors, and see that **V** is one of them!



Q12: Repairing Dijkstra

Repairing Dijkstra

- Assume a connected graph that we have run Dijkstra on, with start and goal **S** and **G**. This means each node has its **predecessor** set.

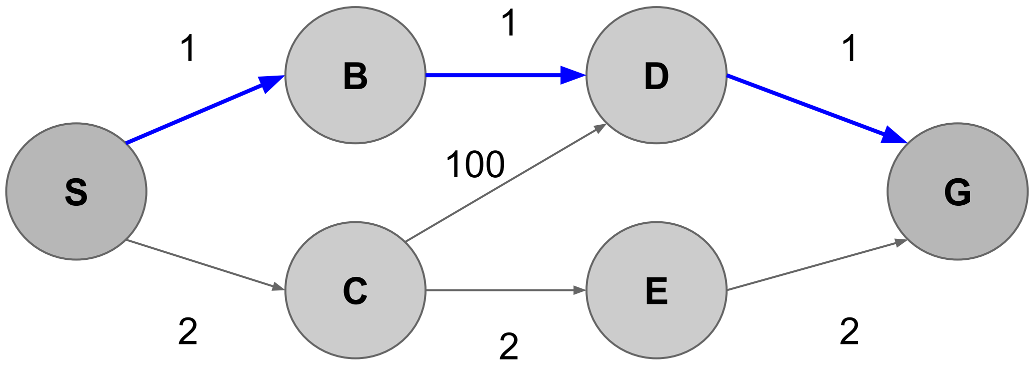
  After this, I increase the weight of some edge **U->V** on the shortest path between **S** and **G** (you don't know by how much). Although my shortest path may no longer be valid, I can look at all of **V's** incoming edges, select the cheapest one, and set **V's** new predecessor to be the other endpoint of that cheapest edge.

  This sets all of the predecessors correctly, and I have repaired my shortest path despite the edge change.

  If this works, give a (short) explanation why. If False, give a counterexample.

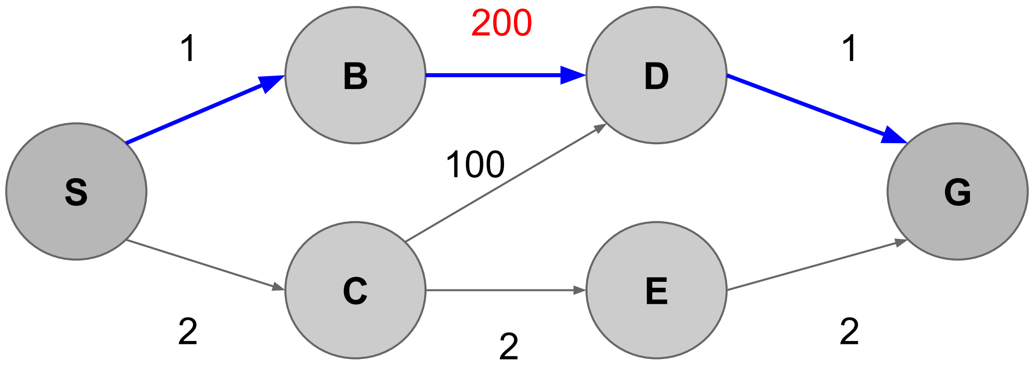
Repairing Dijkstra: Solution

- This **does not work.** It sounds like it should, but consider this graph:



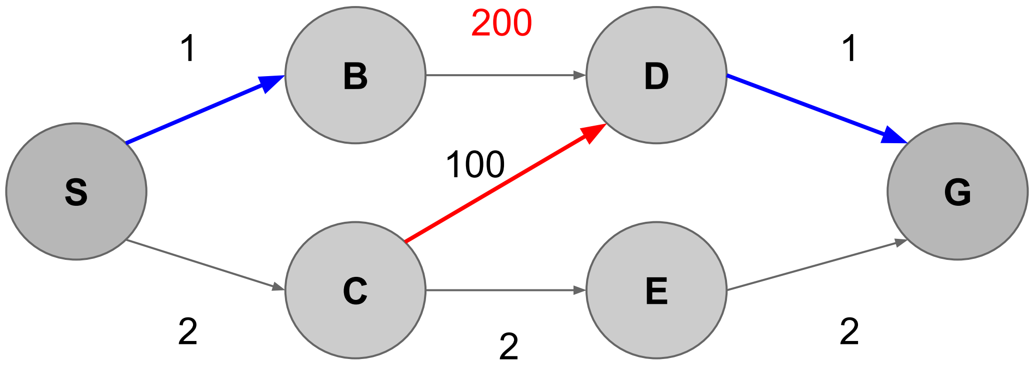
Repairing Dijkstra: Solution

- This **does not work.** It sounds like it should, but consider this graph:



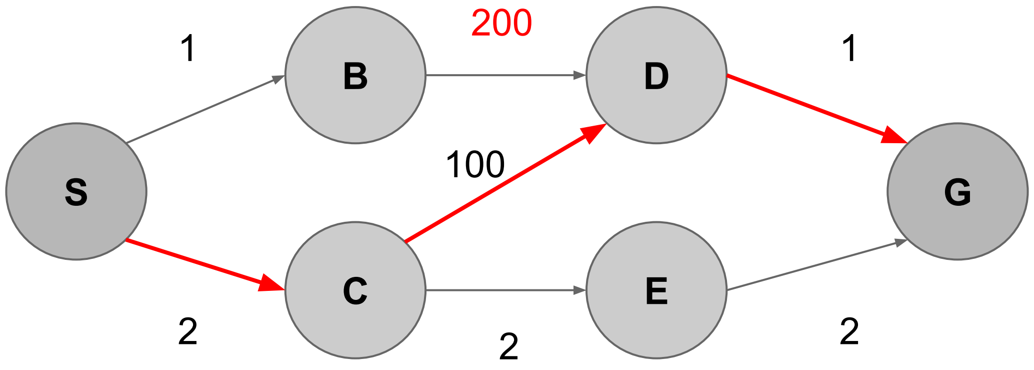
Repairing Dijkstra: Solution

- This **does not work.** It sounds like it should, but consider this graph:



Repairing Dijkstra: Solution

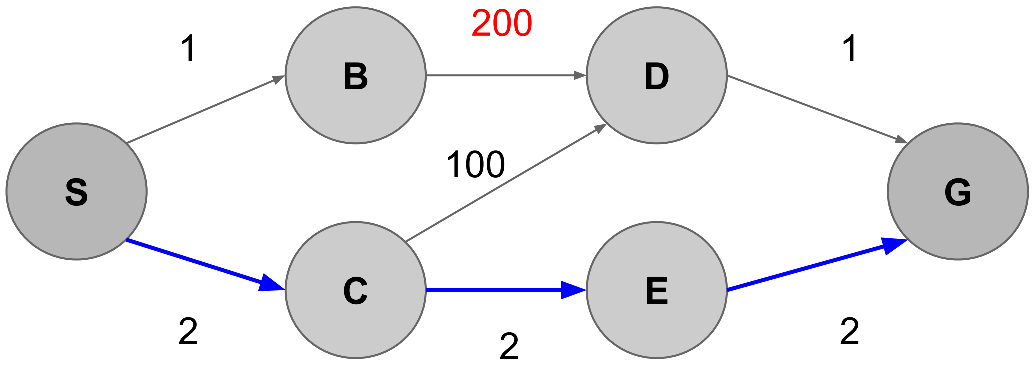
- This **does not work.** It sounds like it should, but consider this graph:



This is my new path. This is wrong!

Repairing Dijkstra: Solution

- True shortest path after the edge change:



Extra: If you're curious, look up "Lifelong Planning Dijkstra" for an algorithm that actually works in this scenario.