CSE 421 Section 2

Graph Search

Administrivia

Announcements & Reminders

- HW1
  - Was due yesterday, 1/10
  - Total of up to **5 late days**
  - At most **2 late days per assignment**

- HW2
  - Due Wednesday 1/17 @ 11:59pm
  - Don’t wait to start

Graph Modeling

Modeling a Problem

- In order to write an algorithm for a word problem, first we need to translate that word problem into a form that we can interact with more easily.
- Often, that means figuring out how to encode it into data structures and identifying what type of algorithm might work for solving it
- A common form this will take is **graph modeling**, turning the problem into a graph. This will let us use graph algorithms to help us find our solution.

Graph Modeling Steps

1. Ask **"what are my fundamental objects?"** (These are usually your vertices)
2. Ask **"how are they related to each other?"** (These are usually your edges)
    - Be sure you can answer these questions:
        - Are the edges directed or undirected?
        - Are the edges weighted? If so, what are the weights? Are negative edges possible?
        - The prior two usually warrant explicit discussion in a solution. You should also be able to answer, "are there self-loops and multi-edges", though often it doesn't need explicit mention in the solution.
3. Ask **"What am I looking for, is it encoded in the graph?"** Are you looking for a path in the graph? A short(-est) one or long(-est) one or any one? Or maybe an MST or something else?
4. Ask **"How do I find the object from 3?"** If you know how, great! Choose an algorithm you know. If not, can you design an algorithm?
5. If stuck on step 4, you might need to go back to step 1! Sometimes a different graph representation leads to better results, and you'll only realize it around step 3 or 4.

Writing Algorithms Using Existing Algorithms

- Often, a problem can be solved by using an existing algorithm in one of two ways:
    - **Reduction / Calling the existing algorithm** (like a library function) with some additional work before and/or after the call
    - **Modifying the existing algorithm** slightly
- Both are valid approaches, and which one you choose depends on the problem
- Whenever possible, it's a good idea to use ideas that you know work! You don't need to start from scratch to reinvent the wheel

Big-O

Big-O Review

- Big-O lets us analyze the runtime of algorithms as a function of the size of the input, usually denoted as n

- Super important for understanding algorithms and comparing them!
(so, you will be doing this analysis for every algorithm you write)

- Given two functions f and g:
    - $f(n)$ is $\boldsymbol{\mathcal{O}(g(n))}$ iff there is a constant $c > 0$ so that $f(n)/g(n)$is eventually always $\leq c$
    - $f(n)$ is $\boldsymbol{o(g(n))}$ iff $\lim_{n\to\infty} f(n)/g(n) = 0.$
    - $f(n)$ is $\boldsymbol{\Omega(g(n))}$ iff there is a constant $c > 0$ so that $f(n)/g(n)$ is eventually always $> c \cdot g(n)$
    - $f(n)$ is $\boldsymbol{\Theta(g(n))}$ iff there are constants $c_1, c_2 > 0$ so that eventually always $c_1 \cdot g(n) < f(n) < c_2 \cdot g(n)$

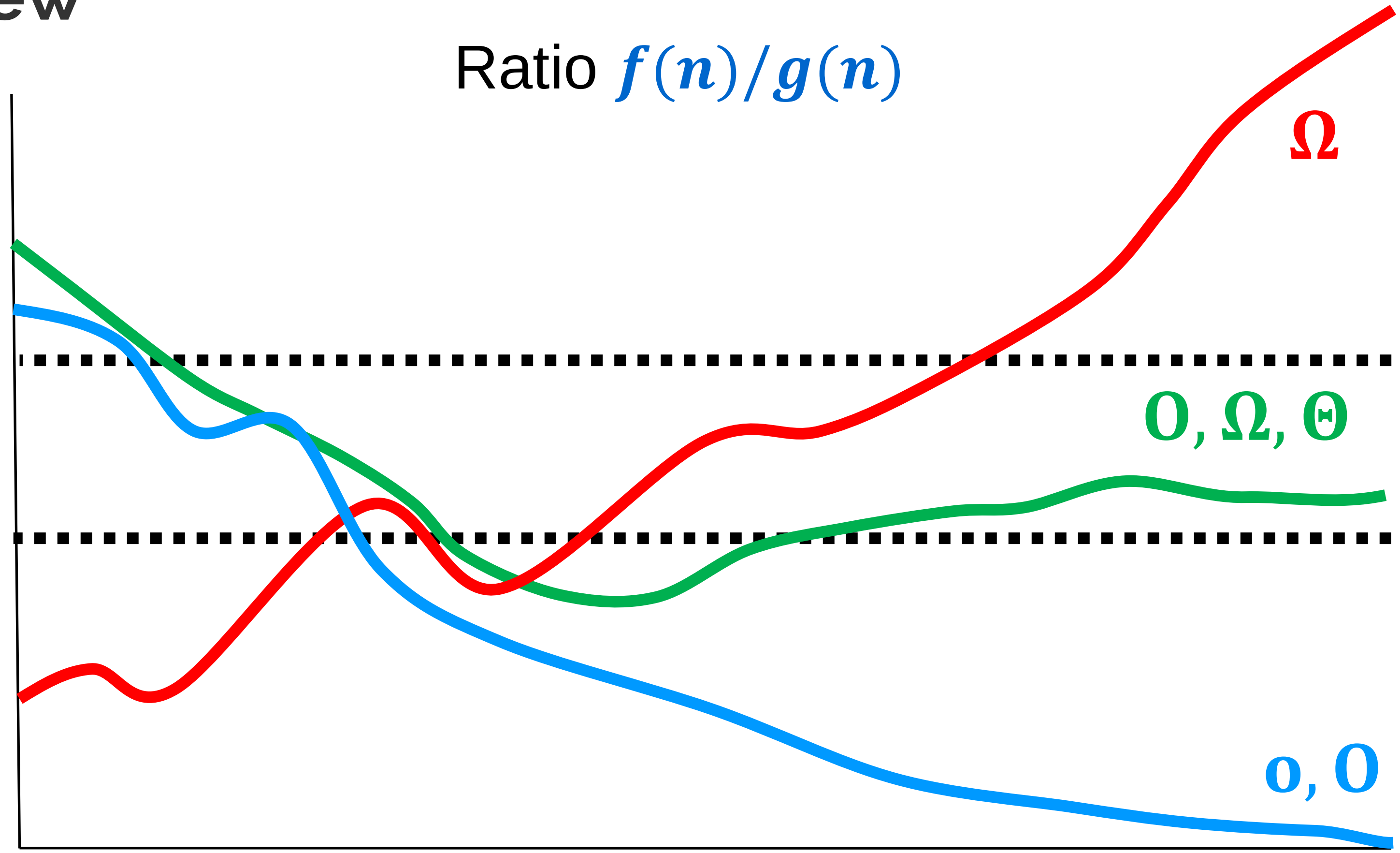
$\boldsymbol{\mathcal{O}(g(n))}$ is fancy $\leq$

$\boldsymbol{\Omega(g(n))}$ is fancy $\geq$

$\boldsymbol{\Theta(g(n))}$ is fancy $\approx$

Big-O Review

Ratio $f(n)/g(n)$



Big-O Tips for Comparing

- We're looking for asymptotic comparison, so just testing values won't necessarily give you a good idea

  - **Exponentials:** 2^n and 3^n are different, which means 2^n and $2^{n/2}$ are different!
    [constant factors IN EXPONENTS are not constant factors]

  - **Exponentials vs Polynomials:** for all $r > 1$ and all $d > 0$, $n^d = O(r^n)$
    [in other words, every exponential grows faster than every polynomial]

  - **Logs vs Polynomials:** $\log^a(n)$ is asymptotically less than n^b for any positive constants a, b

- Key strategy: rewriting functions as $2^{f(n)}$ or $\log(f(n))$ will often make it easier to find the correct order for functions

Problem 1 – Big-O-No

Put these functions in increasing order. That is, if f comes before g in the list, it must be the case that $f(n)$ is $\mathcal{O}(g(n))$. Additionally, if there are any pairs such that $f(n)$ is $\Theta(g(n))$, mark those pairs.

- $2^{\log(n)}$
- $2^{n \log(n)}$
- $\log(\log(n))$
- $2^{\sqrt{n}}$
- $3^{\sqrt{n}}$
- $\log(n)$
- $\log(n^2)$
- $\sqrt{n}$
- $(\log(n))^2$

Hint: A useful trick in these problems is to know that since $\log(\cdot)$ is an increasing function, if $f(n)$ is $\mathcal{O}(g(n))$, then $\log(f(n))$ is $\mathcal{O}(\log(g(n))$. But be careful! Since $\log(\cdot)$ makes functions much smaller it can obscure differences between functions. For example, even though n^3 is less than n^4 , $\log(n^3)$ and $\log(n^4)$ are big-Θ of each other.

Work on this problem with the people around you, and then we'll go over it together!

Problem 1 – Big-O-No

Put these functions in increasing order. That is, if f comes before g in the list, it must be the case that $f(n)$ is $\mathcal{O}(g(n))$. Additionally, if there are any pairs such that $f(n)$ is $\Theta(g(n))$, mark those pairs.

- $2^{\log(n)}$
- $2^{n\log(n)}$
- $\log(\log(n))$
- $2^{\sqrt{n}}$
- $3^{\sqrt{n}}$
- $\log(n)$
- $\log(n^2)$
- $\sqrt{n}$
- $(\log(n))^2$

Problem 1 – Big-O-No

Put these functions in increasing order. That is, if f comes before g in the list, it must be the case that $f(n)$ is $\mathcal{O}(g(n))$. Additionally, if there are any pairs such that $f(n)$ is $\Theta(g(n))$, mark those pairs.

- $2^{\log(n)}$
- $2^{n\log(n)}$
- $\log(\log(n))$
- $2^{\sqrt{n}}$
- $3^{\sqrt{n}}$
- $\log(n)$
- $\log(n^2)$
- $\sqrt{n}$
- $(\log(n))^2$

a) $\log(\log(n))$

Problem 1 – Big-O-No

Put these functions in increasing order. That is, if f comes before g in the list, it must be the case that $f(n)$ is $\mathcal{O}(g(n))$. Additionally, if there are any pairs such that $f(n)$ is $\Theta(g(n))$, mark those pairs.

- $2^{\log(n)}$
- $2^{n \log(n)}$
- $\log(\log(n))$
- $2^{\sqrt{n}}$
- $3^{\sqrt{n}}$
- $\log(n)$
- $\log(n^2)$
- $\sqrt{n}$
- $(\log(n))^2$

a) $\log(\log(n))$
b) $\log(n)$

Problem 1 – Big-O-No

Put these functions in increasing order. That is, if f comes before g in the list, it must be the case that $f(n)$ is $\mathcal{O}(g(n))$. Additionally, if there are any pairs such that $f(n)$ is $\Theta(g(n))$, mark those pairs.

- $2^{\log(n)}$
- $2^{n\log(n)}$
- $\log(\log(n))$
- $2^{\sqrt{n}}$
- $3^{\sqrt{n}}$
- $\log(n)$
- $\log(n^2)$
- $\sqrt{n}$
- $(\log(n))^2$

a) $\log(\log(n))$
b) $\log(n)$
c) $\log(n^2)$
this function is Θ(log(n)); they differ only by the constant factor 2
d) $(\log(n))^2$

Problem 1 – Big-O-No

Put these functions in increasing order. That is, if f comes before g in the list, it must be the case that $f(n)$ is $\mathcal{O}(g(n))$. Additionally, if there are any pairs such that $f(n)$ is $\Theta(g(n))$, mark those pairs.

- $2^{\log(n)}$
- $2^{n\log(n)}$
- $\log(\log(n))$
- $2^{\sqrt{n}}$
- $3^{\sqrt{n}}$
- $\log(n)$
- $\log(n^2)$
- $\sqrt{n}$
- $(\log(n))^2$

a) $\log(\log(n))$
b) $\log(n)$
c) $\log(n^2)$
this function is Θ(log(n)); they differ only by the constant factor 2
d) $(\log(n))^2$
e) $\sqrt{n}$

Problem 1 – Big-O-No

Put these functions in increasing order. That is, if f comes before g in the list, it must be the case that $f(n)$ is $\mathcal{O}(g(n))$. Additionally, if there are any pairs such that $f(n)$ is $\Theta(g(n))$, mark those pairs.

- $2^{\log(n)}$
- $2^{n\log(n)}$
- $\log(\log(n))$
- $2^{\sqrt{n}}$
- $3^{\sqrt{n}}$
- $\log(n)$
- $\log(n^2)$
- $\sqrt{n}$
- $(\log(n))^2$

a) $\log(\log(n))$

b) $\log(n)$

c) $\log(n^2)$

this function is Θ(log(n)); they differ only by the constant factor 2

d) $(\log(n))^2$

e) $\sqrt{n}$

f) $2^{\log(n)}$

note: this is just n

Problem 1 – Big-O-No

Put these functions in increasing order. That is, if f comes before g in the list, it must be the case that $f(n)$ is $\mathcal{O}(g(n))$. Additionally, if there are any pairs such that $f(n)$ is $\Theta(g(n))$, mark those pairs.

- $2^{\log(n)}$
- $2^{n\log(n)}$
- $\log(\log(n))$
- $2^{\sqrt{n}}$
- $3^{\sqrt{n}}$
- $\log(n)$
- $\log(n^2)$
- $\sqrt{n}$
- $(\log(n))^2$

a) $\log(\log(n))$
b) $\log(n)$
c) $\log(n^2)$
this function is Θ(log(n)); they differ only by the constant factor 2
d) $(\log(n))^2$
e) $\sqrt{n}$
f) $2^{\log(n)}$
note: this is just n
g) $2^{\sqrt{n}}$

Problem 1 – Big-O-No

Put these functions in increasing order. That is, if f comes before g in the list, it must be the case that $f(n)$ is $\mathcal{O}(g(n))$. Additionally, if there are any pairs such that $f(n)$ is $\Theta(g(n))$, mark those pairs.

- $2^{\log(n)}$
- $2^{n\log(n)}$
- $\log(\log(n))$
- $2^{\sqrt{n}}$
- $3^{\sqrt{n}}$
- $\log(n)$
- $\log(n^2)$
- $\sqrt{n}$
- $(\log(n))^2$

a) $\log(\log(n))$
b) $\log(n)$
c) $\log(n^2)$
this function is Θ(log(n)); they differ only by the constant factor 2
d) $(\log(n))^2$
e) $\sqrt{n}$
f) $2^{\log(n)}$
note: this is just n
g) $2^{\sqrt{n}}$
h) $3^{\sqrt{n}}$

Problem 1 – Big-O-No

Put these functions in increasing order. That is, if f comes before g in the list, it must be the case that $f(n)$ is $\mathcal{O}(g(n))$. Additionally, if there are any pairs such that $f(n)$ is $\Theta(g(n))$, mark those pairs.

- $2^{\log(n)}$
- $2^{n\log(n)}$
- $\log(\log(n))$
- $2^{\sqrt{n}}$
- $3^{\sqrt{n}}$
- $\log(n)$
- $\log(n^2)$
- $\sqrt{n}$
- $(\log(n))^2$

a) $\log(\log(n))$
b) $\log(n)$
c) $\log(n^2)$
this function is Θ(log(n)); they differ only by the constant factor 2
d) $(\log(n))^2$
e) $\sqrt{n}$
f) $2^{\log(n)}$
note: this is just n
g) $2^{\sqrt{n}}$
h) $3^{\sqrt{n}}$
i) $2^{n\log(n)}$

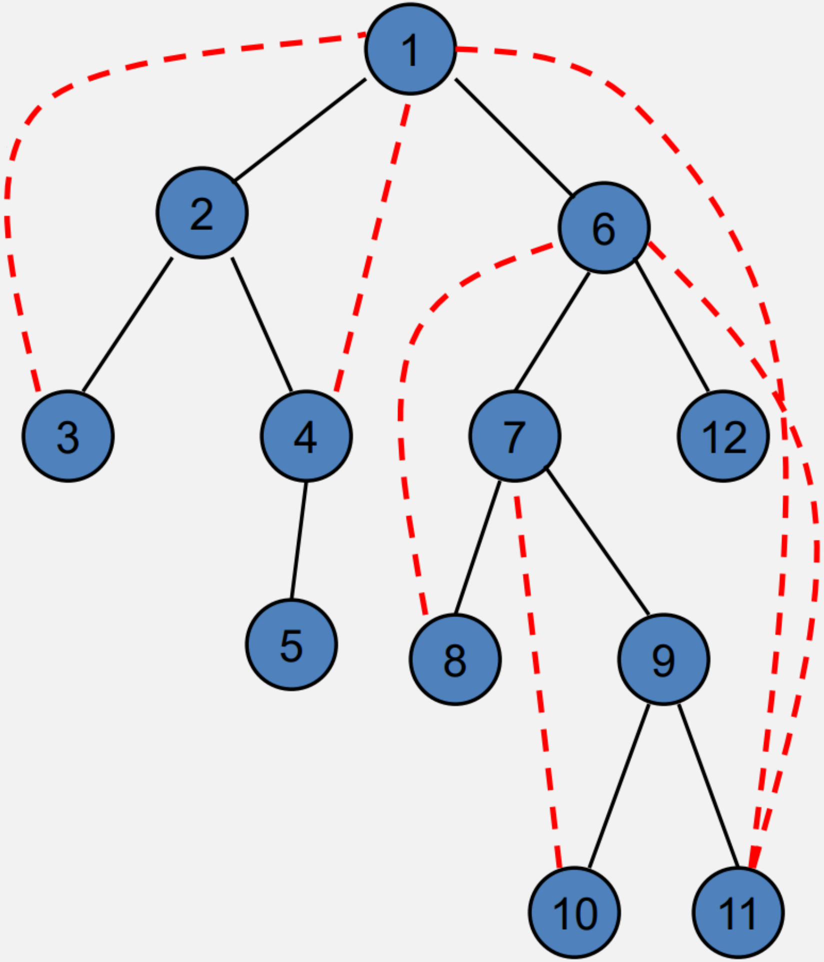
Graph

DFS Review

```
S = {s}
while S is not empty
        u = Pop(S)
        if u is unvisited
                visit u
                foreach v in N(u)
                        Push(S, v)
```

DFS Review

- Each edge goes between vertices on the same branch
- No cross edges

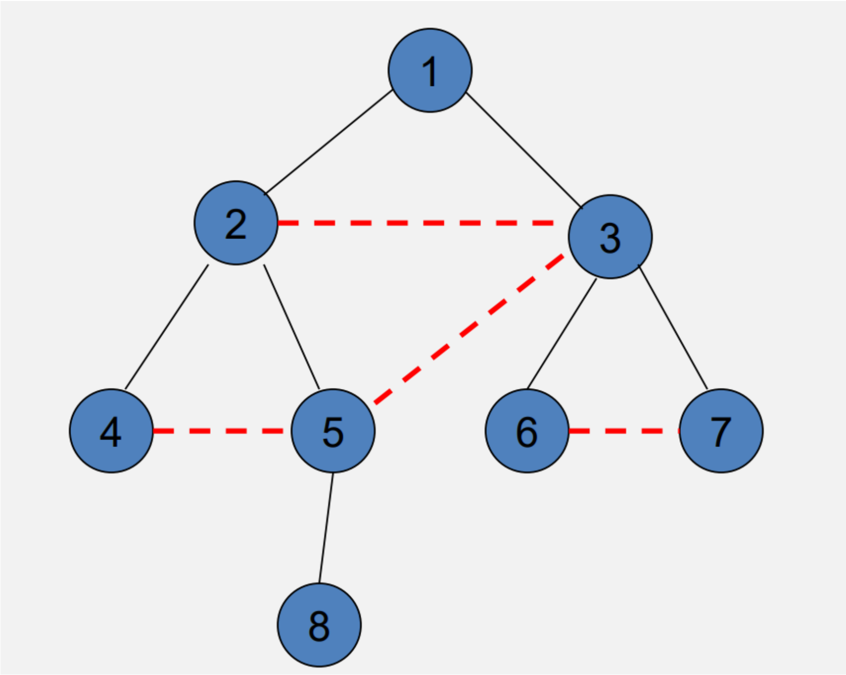


BFS Review

```
S = {s}
while S is not empty
        u = Dequeue(S)
        if u is unvisited
                visit u
                foreach v in N(u)
                        Enqueue(S, v)
```

BFS Review

- All edges go between vertices on the same layer or adjacent layers



Topological Sort

- Given a set of tasks with precedence constraints, find a linear order of the tasks.
- If a graph has a cycle, there is no topological sort

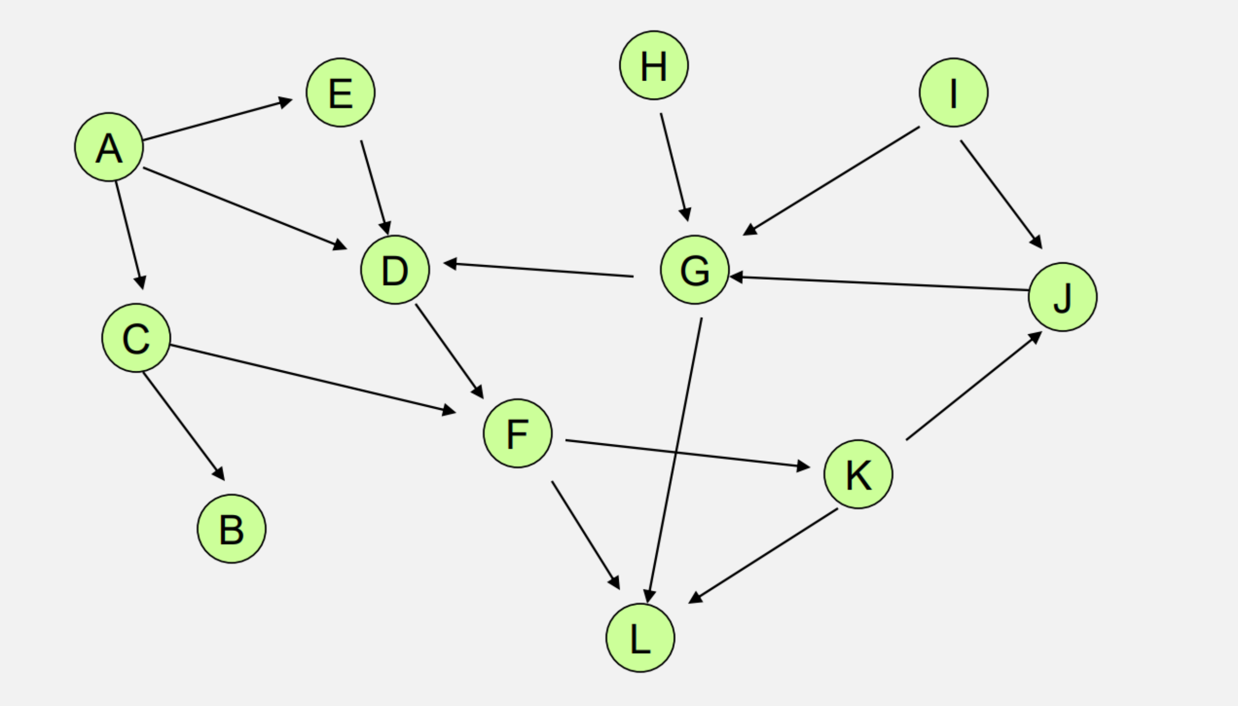
While there exists a vertex v with in-degree 0

    Output vertex v

    Delete the vertex v and all out going edges

- O(n+m)

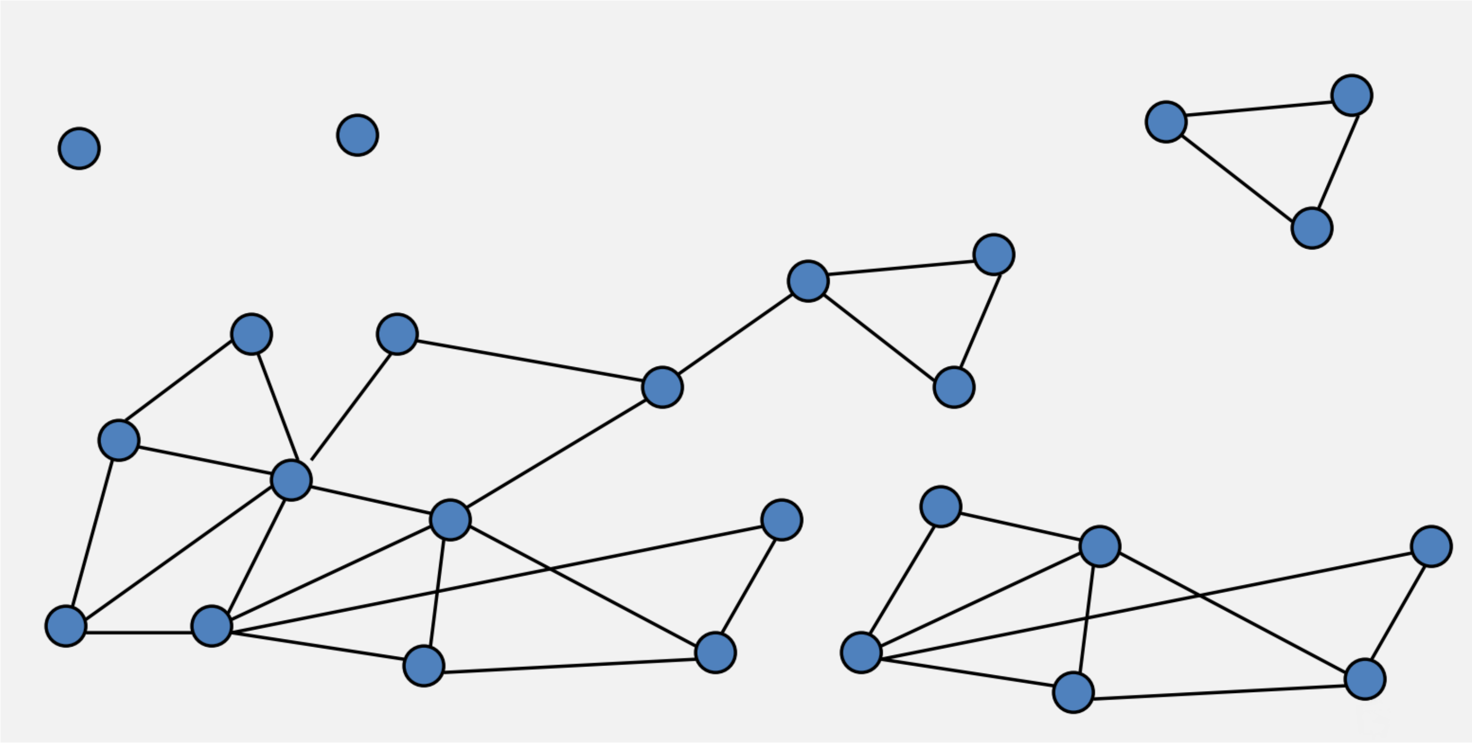
Try to find some different topological orders

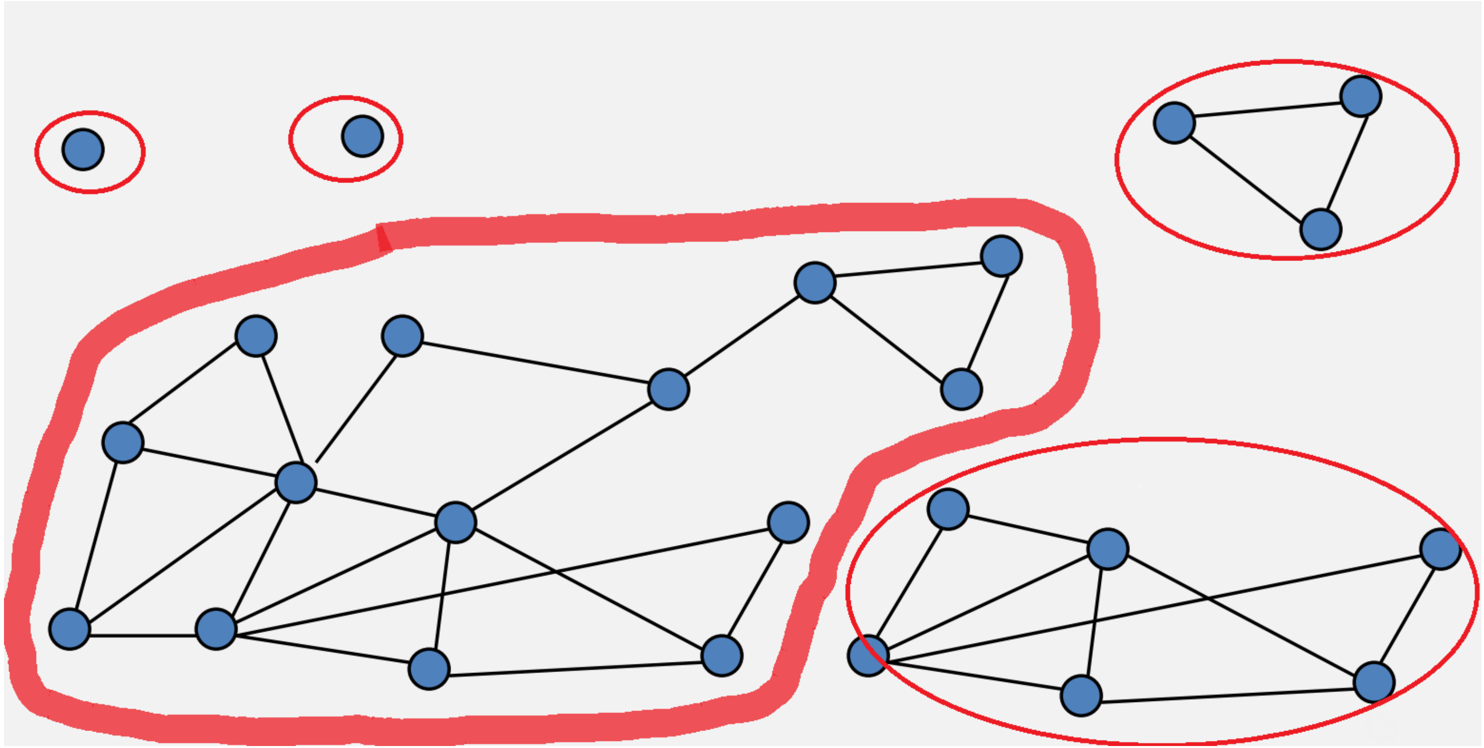


Connected Components

- A connected subgraph that is not part of any larger connected subgraph.
- *For undirected graph

Find all connected components

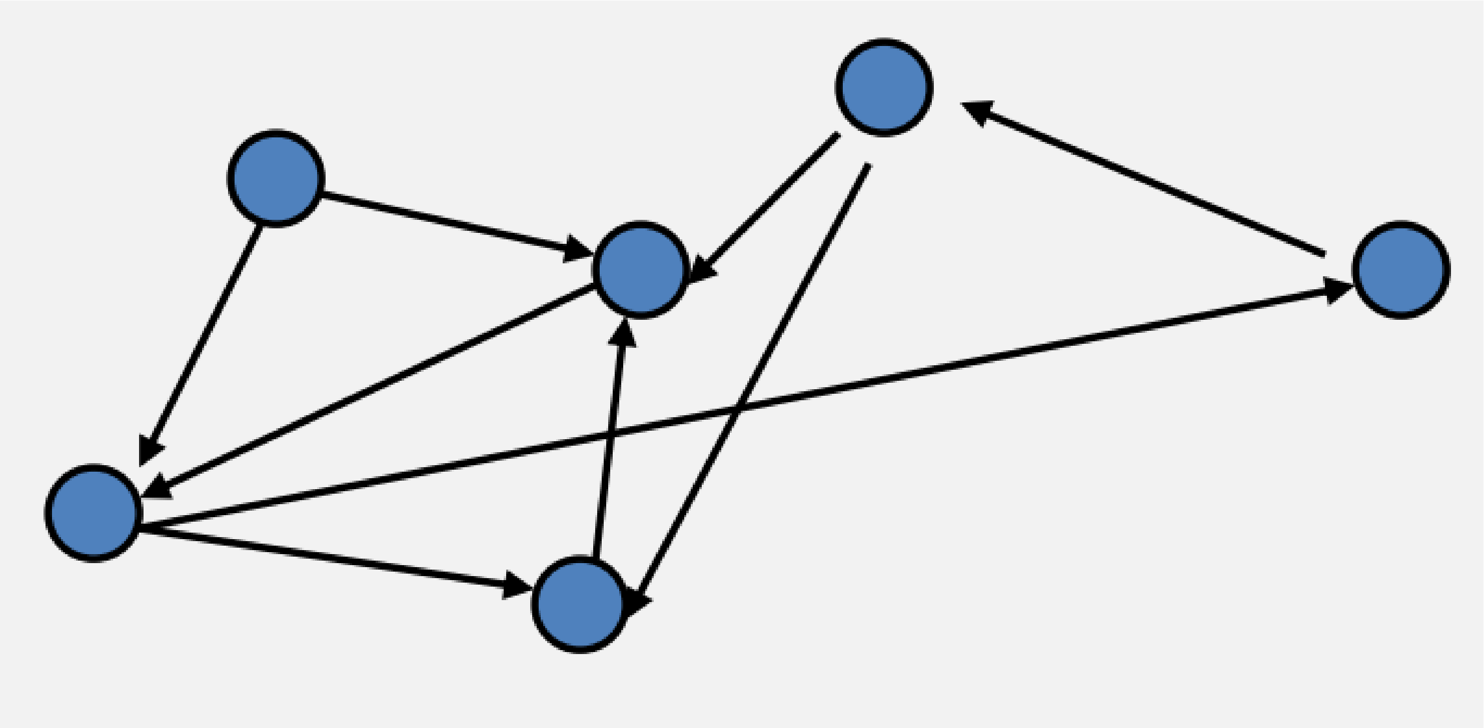


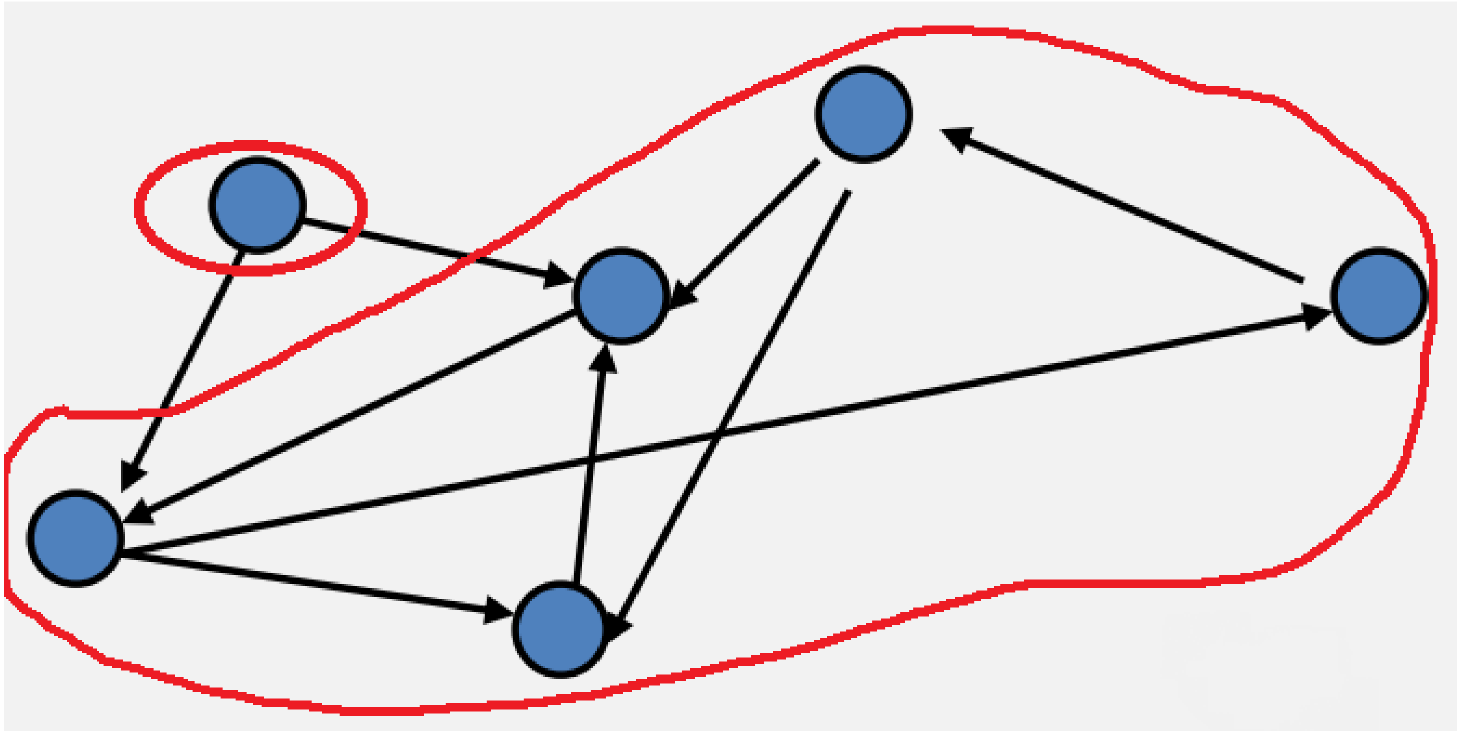


Strongly Connected Components

- A subset of the vertices with paths between every pair of vertices
- *For directed graph

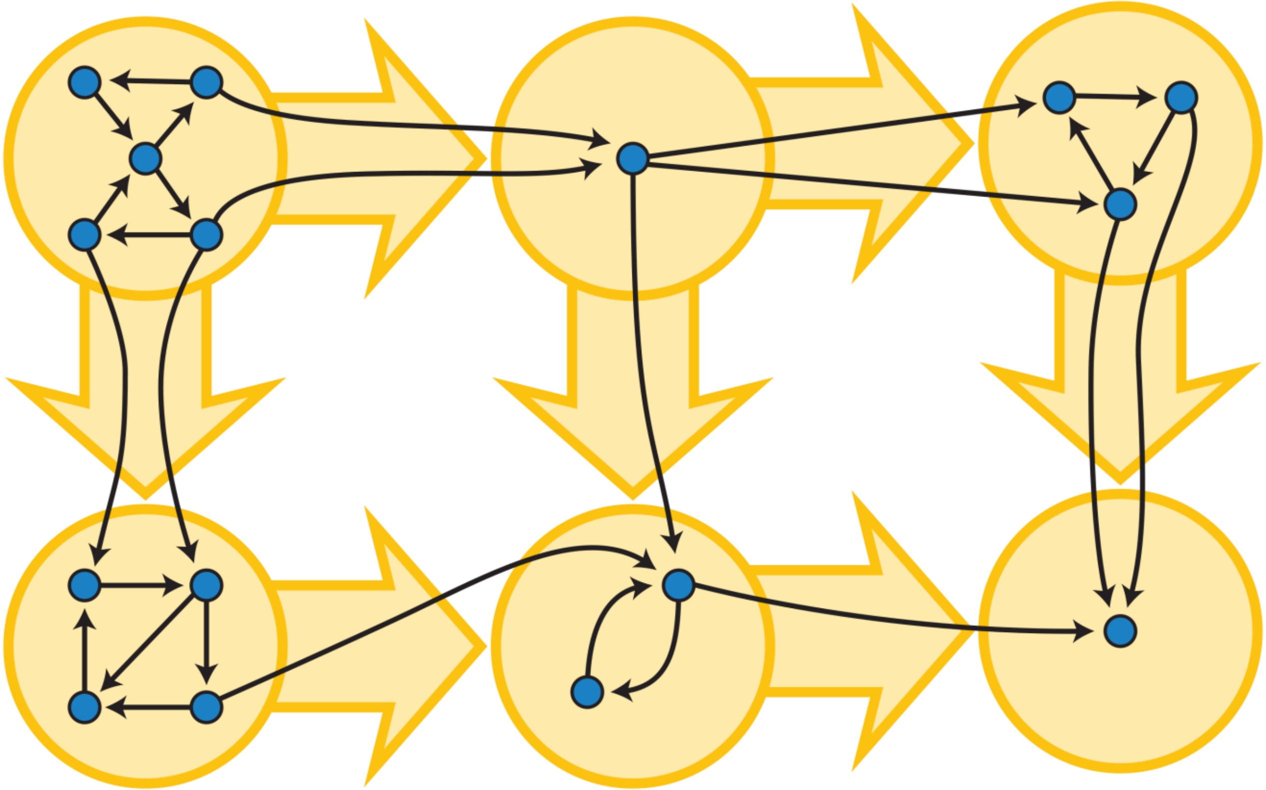
Try to separate the graph into SCCs





Fun Fact

- If each strongly connected component is ”contracted” to a single vertex, the resulting graph is a directed acyclic graph (DAG)



the following slides are provided for reference and may not have been covered in every section

Problem 4 – Graph Modeling

In this problem we're going to solve a classic riddle.

(a) First, you should solve the classic riddle yourself to get a feel for the problem. 2 You are on the beach with a jug that holds exactly 5 gallons, a jug that holds exactly 3 gallons, and a large bucket. Your goal is to put **exactly** 4 gallons of water into the bucket. Unfortunately, the jugs are not graduated (e.g., you can't just fill the larger jug 4/5 full). What you can do are the following operations.

- Completely fill any of your jugs.
- Pour from one of your containers into another until the first container is empty or the second is full.
- Pour out all the remaining water in a container.

How do you get 4 gallons of water into the bucket?

Work on this problem with the people around you, and then we'll go over it together!

Problem 4 – Graph Modeling

(a) How do you get 4 gallons of water into the bucket?

Problem 4 – Graph Modeling

(a) How do you get 4 gallons of water into the bucket?

Fill the 5 gallon jug, pour into the 3 gallon jug until it is full (the jugs now contain 2 and 3 gallons respectively). Pour from the larger jug into the large bucket (it now has 2 gallons). Empty the 3 gallon jug, and repeat all these steps to get the desired 4 gallons.

Problem 4 – Graph Modeling

(a) How do you get 4 gallons of water into the bucket?

Fill the 5 gallon jug, pour into the 3 gallon jug until it is full (the jugs now contain 2 and 3 gallons respectively). Pour from the larger jug into the large bucket (it now has 2 gallons). Empty the 3 gallon jug, and repeat all these steps to get the desired 4 gallons.

Alternatively, Fill the 3 gallon jug, pour into the 5 gallon jug. Fill the 3 gallon jug again, pour until the 5 gallon jug is full (they now contain 5 gallons and 1 gallon respectively) . Pour the 1 gallon from the 3 gallon jug into the bucket. Refill the 3 gallon jug and pour into the bucket to bring the total to 4 gallons.

There may be other solutions.

Problem 4 – Graph Modeling

(b) Now, write an algorithm to solve any instance of this puzzle. You are given a list of 10 jugs with (positive integer) capacities $c_1, \ldots, c_{10}$, ranging from 1 to C. Your goal is to determine whether it is possible to get exactly t gallons into a bucket whose capacity is at least t and at most B.

Hint: Think about how you can relate possible "states" of the puzzle to nodes of some graph. What would be a good way to define edges for this graph? How could you think of solutions to the puzzle in terms of the graph?

Problem 4 – Graph Modeling

(b) Now, write an algorithm to solve any instance of this puzzle. You are given a list of 10 jugs with (positive integer) capacities $c_1, \dots, c_{10}$, ranging from 1 to C. Your goal is to determine whether it is possible to get exactly t gallons into a bucket whose capacity is at least t and at most B.

Hint: Think about how you can relate possible "states" of the puzzle to nodes of some graph. What would be a good way to define edges for this graph? How could you think of solutions to the puzzle in terms of the graph?

Intuition:

The "state" of the puzzle can be represented as the number of gallons in each of the jugs and the bucket.

We encode the rules of the puzzle such that each possible step is an edge.

Problem 4 – Graph Modeling

(b) Now, write an algorithm to solve any instance of this puzzle. You are given a list of 10 jugs with (positive integer) capacities $c_1, \ldots, c_{10}$, ranging from 1 to C. Your goal is to determine whether it is possible to get exactly t gallons into a bucket whose capacity is is at least t and at most B.

Intuition:

The "state" of the puzzle can be represented as the number of gallons in each of the jugs and the bucket.

We encode the rules of the puzzle such that each possible step is an edge.

Work on this problem with the people around you. First see if you can model it as a graph, and then think about how you could use that graph in an algorithm. Then we'll go over it together!

Problem 4 – Graph Modeling

(b) Now, write an algorithm to solve any instance of this puzzle. You are given a list of 10 jugs with (positive integer) capacities $c_1, \dots, c_{10}$, ranging from 1 to C. Your goal is to determine whether it is possible to get exactly t gallons into a bucket with capacity at least t and most B.

What are my fundamental objects?:

How are they related to each other?:

What am I looking for, is it encoded in the graph?:

How do I find the object from 3:

Problem 4 – Graph Modeling

(b) Now, write an algorithm to solve any instance of this puzzle. You are given a list of 10 jugs with (positive integer) capacities $c_1, \ldots, c_{10}$, ranging from 1 to C. Your goal is to determine whether it is possible to get exactly t gallons into a bucket with capacity at least t and most B.

What are my fundamental objects?: How much water is in each jug and how much is in the final bucket. We need a vertex for each possible state the jugs and bucket can be in.

How are they related to each other?:

What am I looking for, is it encoded in the graph?:

How do I find the object from 3:

Problem 4 – Graph Modeling

(b) Now, write an algorithm to solve any instance of this puzzle. You are given a list of 10 jugs with (positive integer) capacities $c_1, \ldots, c_{10}$, ranging from 1 to C. Your goal is to determine whether it is possible to get exactly t gallons into a bucket with capacity at least t and most B.

What are my fundamental objects?: How much water is in each jug and how much is in the final bucket. We need a vertex for each possible state the jugs and bucket can be in.

How are they related to each other?: We can fill a jug, empty a jug, or pour one jug into another or the bucket. So, we need edges that connect the different states for each of these valid operations.

What am I looking for, is it encoded in the graph?:

How do I find the object from 3:

Problem 4 – Graph Modeling

(b) Now, write an algorithm to solve any instance of this puzzle. You are given a list of 10 jugs with (positive integer) capacities $c_1, \ldots, c_{10}$, ranging from 1 to C. Your goal is to determine whether it is possible to get exactly t gallons into a bucket with capacity at least t and most B.

What are my fundamental objects?: How much water is in each jug and how much is in the final bucket. We need a vertex for each possible state the jugs and bucket can be in.

How are they related to each other?: We can fill a jug, empty a jug, or pour one jug into another or the bucket. So, we need edges that connect the different states for each of these valid operations.

What am I looking for, is it encoded in the graph?: The series of pours that will get t gallons of water into the bucket. So, any path that gets us from 0 gallons in all the jugs and the bucket, to one of the states where the amount of water in the bucket is equal to t, if one exists, is the solution. To make things easier, we could add an extra “end” vertex that connects to every such state where there are t gallons in the bucket. Then, any path starting from the 0 vertex ending at this “end” vertex is a possible solution.

How do I find the object from 3:

Problem 4 – Graph Modeling

(b) Now, write an algorithm to solve any instance of this puzzle. You are given a list of 10 jugs with (positive integer) capacities $c_1, \ldots, c_{10}$, ranging from 1 to C. Your goal is to determine whether it is possible to get exactly t gallons into a bucket with capacity at least t and most B.

What are my fundamental objects?: How much water is in each jug and how much is in the final bucket. We need a vertex for each possible state the jugs and bucket can be in.

How are they related to each other?: We can fill a jug, empty a jug, or pour one jug into another or the bucket. So, we need edges that connect the different states for each of these valid operations.

What am I looking for, is it encoded in the graph?: The series of pours that will get t gallons of water into the bucket. So, any path that gets us from 0 gallons in all the jugs and the bucket, to one of the states where the amount of water in the bucket is equal to t, if one exists, is the solution. To make things easier, we could add an extra "end" vertex that connects to every such state where there are t gallons in the bucket. Then, any path starting from the 0 vertex ending at this "end" vertex is a possible solution.

How do I find the object from 3: Work on this part with the people around you, see if you can put it together into an algorithm, and we'll go over it together!

Problem 4 – Graph Modeling

(b) Now, write an algorithm to solve any instance of this puzzle. You are given a list of 10 jugs with (positive integer) capacities $c_1, \ldots, c_{10}$, ranging from 1 to C. Your goal is to determine whether it is possible to get exactly t gallons into a bucket with capacity at least t and most B.

Problem 4 – Graph Modeling

(b) Now, write an algorithm to solve any instance of this puzzle. You are given a list of 10 jugs with (positive integer) capacities $c_1, \ldots, c_{10}$, ranging from 1 to C. Your goal is to determine whether it is possible to get exactly t gallons into a bucket with capacity at least t and most B.

Let S be the set of all 11-tuples, where for the first 10 entries, the entry is an integer between 0 and c_i , and the final entry for the bucket is an integer between 0 and B. There are $(C+1)^{10} \cdot (B+1)$ such states.

We make a graph with a vertex for every element of S. And add an edge from u to v if and only if the states meet one of these conditions.

Problem 4 – Graph Modeling

(b) Now, write an algorithm to solve any instance of this puzzle. You are given a list of 10 jugs with (positive integer) capacities $c_1, \dots, c_{10}$, ranging from 1 to C. Your goal is to determine whether it is possible to get exactly t gallons into a bucket with capacity at least t and most B.

Let S be the set of all 11-tuples, where for the first 10 entries, the entry is an integer between 0 and c_i , and the final entry for the bucket is an integer between 0 and B. There are $(C+1)^{10} \cdot (B+1)$ such states.

We make a graph with a vertex for every element of S. And add an edge from u to v if and only if the states meet one of these conditions.

From a given state $(j_1, j_2, \dots, j_{10}, j_{11})$, you can move to another state $(j'_1, j'_2, \dots, j'_{10}, j'_{11})$ if and only if one of the following hold:

Problem 4 – Graph Modeling

(b) Now, write an algorithm to solve any instance of this puzzle. You are given a list of 10 jugs with (positive integer) capacities $c_1, \dots, c_{10}$, ranging from 1 to C. Your goal is to determine whether it is possible to get exactly t gallons into a bucket with capacity at least t and most B.

Let S be the set of all 11-tuples, where for the first 10 entries, the entry is an integer between 0 and c_i , and the final entry for the bucket is an integer between 0 and B. There are $(C+1)^{10} \cdot (B+1)$ such states.

We make a graph with a vertex for every element of S. And add an edge from u to v if and only if the states meet one of these conditions.

From a given state $(j_1, j_2, \dots, j_{10}, j_{11})$, you can move to another state $(j'_1, j'_2, \dots, j'_{10}, j'_{11})$ if and only if one of the following hold:

- There is only one index, k, where the tuples differ, and $j'_k = 0$. (we emptied a jug or the bucket)
- There is only one index, k, $(k \leq 10)$ where the tuples differ, and $j_k = c_k$. (we filled a jug)
- There are two indices, k, l where the tuples differ:
  - $j'_k = 0$ or $j'_l = c_l$
  - $j_k + j_l = j'_k + j'_l$

Problem 4 – Graph Modeling

(b) Now, write an algorithm to solve any instance of this puzzle. You are given a list of 10 jugs with (positive integer) capacities $c_1, \ldots, c_{10}$, ranging from 1 to C. Your goal is to determine whether it is possible to get exactly t gallons into a bucket with capacity at least t and most B.

Let S be the set of all 11-tuples, where for the first 10 entries, the entry is an integer between 0 and c_i , and the final entry for the bucket is an integer between 0 and B. There are $(C+1)^{10} \cdot (B+1)$ such states.

We make a graph with a vertex for every element of S. And add an edge from u to v if and only if the states meet one of these conditions.

From a given state $(j_1, j_2, \ldots, j_{10}, j_{11})$, you can move to another state $(j'_1, j'_2, \ldots, j'_{10}, j'_{11})$ if and only if one of the following hold:

- There is only one index, k, where the tuples differ, and $j'_k = 0$. (we emptied a jug or the bucket)
- There is only one index, k, $(k \leq 10)$ where the tuples differ, and $j_k = c_k$. (we filled a jug)
- There are two indices, k, l where the tuples differ:
  – $j'_k = 0$ or $j'_l = c_l$
  – $j_k + j_l = j'_k + j'_l$

Finally, we add a target vertex z, to the graph. Add an edge from every tuple where $j_{11} = t$ to z.

Problem 4 – Graph Modeling

(b) Now, write an algorithm to solve any instance of this puzzle. You are given a list of 10 jugs with (positive integer) capacities $c_1, \dots, c_{10}$, ranging from 1 to C. Your goal is to determine whether it is possible to get exactly t gallons into a bucket with capacity at least t and most B.

Let S be the set of all 11-tuples, where for the first 10 entries, the entry is an integer between 0 and c_i , and the final entry for the bucket is an integer between 0 and B. There are $(C+1)^{10} \cdot (B+1)$ such states.

We make a graph with a vertex for every element of S. And add an edge from u to v if and only if the states meet one of these conditions.

From a given state $(j_1, j_2, \dots, j_{10}, j_{11})$, you can move to another state $(j'_1, j'_2, \dots, j'_{10}, j'_{11})$ if and only if one of the following hold:

- There is only one index, k, where the tuples differ, and $j'_k = 0$. (we emptied a jug or the bucket)
- There is only one index, k, $(k \leq 10)$ where the tuples differ, and $j_k = c_k$. (we filled a jug)
- There are two indices, k, l where the tuples differ:
  - $j'_k = 0$ or $j'_l = c_l$
  - $j_k + j_l = j'_k + j'_l$

Finally, we add a target vertex z, to the graph. Add an edge from every tuple where $j_{11} = t$ to z.

We then use [B/D]FS, starting from the all 0's tuple, and searching to see if z is reachable. If it is, we can return true (and predecessor edges will show the steps to take). If t is not reachable, then return false.

Problem 4 – Graph Modeling

(b) Now, write an algorithm to solve any instance of this puzzle. You are given a list of 10 jugs with (positive integer) capacities $c_1, \ldots, c_{10}$, ranging from 1 to C. Your goal is to determine whether it is possible to get exactly t gallons into a bucket with capacity at least t and most B.

Correctness

Problem 4 – Graph Modeling

(b) Now, write an algorithm to solve any instance of this puzzle. You are given a list of 10 jugs with (positive integer) capacities $c_1, \ldots, c_{10}$, ranging from 1 to C. Your goal is to determine whether it is possible to get exactly t gallons into a bucket with capacity at least t and most B.

Correctness

Suppose our algorithm returns true. Then [B/D]FS found a walk from all 0's to z. By construction of the graph, each edge corresponds to a valid rule we can apply in the original puzzle. Since the only edges going into t are from states where $j_{11} = t$, the walk must reach such a state, thus the puzzle can be solved by doing the steps on the edges of that walk.

Conversely, suppose the puzzle is solvable. Then there is a series of steps that can be taken to end up with t gallons into the bucket. Each legal step has a corresponding edge in the graph to the next state by construction, so there is a path to a valid stopping state (i.e., a tuple where $j_{11} = t$), we added an edge from all such vertices to z, so there is a path from all 0's to z. [B/D]FS will discover this path, so we will return true.

Problem 4 – Graph Modeling

(b) Now, write an algorithm to solve any instance of this puzzle. You are given a list of 10 jugs with (positive integer) capacities $c_1, \ldots, c_{10}$, ranging from 1 to C. Your goal is to determine whether it is possible to get exactly t gallons into a bucket with capacity at least t and most B.

Running Time

Problem 4 – Graph Modeling

(b) Now, write an algorithm to solve any instance of this puzzle. You are given a list of 10 jugs with (positive integer) capacities $c_1, \dots, c_{10}$, ranging from 1 to C. Your goal is to determine whether it is possible to get exactly t gallons into a bucket with capacity at least t and most B.

Running Time

Let $n = (C+1)^{10}(B+1)$+1. Note that this is the number of vertices in our graph. Each vertex has a constant number of edges leaving it (as you are performing one of three operations (dump, pour, fill) among a constant number of jugs. So the graph can be has $O(n)$ edges and can be constructed in $O(n)$ time. Running [B/D]FS in a graph with $O(n)$ vertices and edges takes $O(n)$ time, so the overall running time is $O(n)$.

Proof by Contradiction using Extremes

Contradiction

- Writing inductive arguments concisely using proofs by contradiction is a really common technique we will use to prove that algorithms are correct in this class
  - We already did this for the proof of proposer-optimality when we considered the first bad event, but it isn't usually as complicated as that argument
  - In the Section 1 handout Problem 5 there is another example involved in Proving Code Correct.

- Reminder: This isn't the only way to use proof by contradiction. Proof by contradiction is the general technique where we assume the opposite of the conclusion we want to prove, and show that somewhere there will be a logical inconsistency if this is the case,
  - the assumption must be false (and therefore the original claim must be true)

- Explicitly state that you are doing proof by contradiction in the introduction of your proof!

- Explicitly identify what it is that you are supposing!

- And explicitly state at the end how you got the contradiction!

Problem 2 - Write it Slicker: Proof by Contradiction

Claim: For every directed graph G, if every node of G has out-degree at least 1, then G has a directed cycle.

a) Prove the claim using proof by contradiction.

Starting point:

Problem 2 - Write it Slicker: Proof by Contradiction

Claim: For every directed graph G, if every node of G has out-degree at least 1, then G has a directed cycle.

a) Prove the claim using proof by contradiction.

Starting point:

Suppose, for the sake of contradiction, that there is a finite directed graph G such that every node of G has out-degree at least 1, but G has no directed cycle.

Problem 2 - Write it Slicker: Proof by Contradiction

Claim: For every directed graph G, if every node of G has out-degree at least 1, then G has a directed cycle.

a) Prove the claim using proof by contradiction.

Starting point:

Suppose, for the sake of contradiction, that there is a finite directed graph G such that every node of G has out-degree at least 1, but G has no directed cycle.

Now finish the proof with the people around you, and we'll go over it together.

Problem 2 - Write it Slicker: Proof by Contradiction

<u>Claim</u>: For every directed graph G, if every node of G has out-degree at least 1, then G has a directed cycle.

a) Prove the claim using proof by contradiction.

Your proof probably looked something like this:

Suppose, for the sake of contradiction, that there is a finite directed graph G such that every node of G has out-degree at least 1, but G has no directed cycle.

We construct long and longer simple paths in G: Start at some node v_0, of G. Its out-degree is at least 1, so we can move from v_0 along an out-edge. If this leads to v_0 that would be a cycle of length 1, a contradiction! Therefore we have a new node v_1, which (like every other node) has out-degree at least 1, so there is an out-edge pointing to v_2. If v_2 is a vertex we have already visited (i.e., v_0 or v_1), then we have found a cycle, a contradiction! Otherwise, from v_2, we may repeat the same argument and continue finding $v_3, v_4, \ldots$. The graph is finite, so we cannot continue this process forever. Eventually we find a repeated vertex, which means we have a cycle, a contradiction!

Problem 2 - Write it Slicker: Proof by Contradiction

It’s common in inductive proofs written using contradiction to have cases like we’ve seen here;

“*Option A*: we’re done with our proof!

Option B: do something else.”

Here, though, that “do something else” has us basically where we started (a new end-vertex on our path where we’ve used one edge), and it’s tempting to say “repeat indefinitely, eventually you hit the other case.” That’s mathematically correct… really using induction under the hood! But not particularly elegant. Even to do things informally you have to write down enough steps that your reader knows what the pattern is, which could be a lot in more complicated situations

A more elegant version is to use ***proof by contradiction using extremes***. Instead of slowly building an object (here, the path), just start with the most *extreme* version of the object at the beginning (usually the biggest one or the first one). Starting with the right object lets us eliminate Option B and jump right to Option A.

Problem 2 - Write it Slicker: Proof by Contradiction

<u>Claim:</u> For every directed graph G, if every node of G has out-degree at least 1, then G has a directed cycle.

b) Rewrite the proof, using the proof by contradiction using extremes technique.

NEW, EXTREME Starting point:

Problem 2 - Write it Slicker: Proof by Contradiction

Claim: For every directed graph G, if every node of G has out-degree at least 1, then G has a directed cycle.

b) Rewrite the proof, using the proof by contradiction using extremes technique.

NEW, EXTREME Starting point:

Suppose, for the sake of contradiction, that there is a directed graph G such that every node of G has out-degree at least 1, but G has no cycle. Let $P = v_0, v_1, \ldots, v_k$ be a longest simple path in G.

Problem 2 - Write it Slicker: Proof by Contradiction

Claim: For every directed graph G, if every node of G has out-degree at least 1, then G has a directed cycle.

b) Rewrite the proof, using the proof by contradiction using extremes technique.

NEW, EXTREME Starting point:

Suppose, for the sake of contradiction, that there is a directed graph G such that every node of G has out-degree at least 1, but G has no directed cycle. Let $P = v_0, v_1, \dots, v_k$ be a longest simple path in G.

Now finish the proof with the people around you, and we'll go over it together.

Problem 2 - Write it Slicker: Proof by Contradiction

<u>Claim:</u> For every directed graph G, if every node of G has out-degree at least 1, then G has a directed cycle.

b) Rewrite the proof, using the proof by contradiction using extremes technique.

Suppose, for the sake of contradiction, that there is a directed graph G such that every node of G has out-degree at least 1, but G has no directed cycle. Let $P = v_0, v_1, \ldots, v_k$ be a longest simple directed path in G.

Since v_k has out-degree at-least 1, it must have an out-edge to a vertex w.

Since P is a *longest* simple directed path, w must be a repeat of a vertex among $v_1, \ldots, v_k$ (otherwise $v_0, v_1, \ldots, v_k, w$ would be a longer simple directed path). Let v_j be the vertex = w.

But then adding the edge (v_k, v_j) to the part of P from v_j to v_k gives a directed cycle in G:

$$v_j, \ldots, v_k, v_j$$

But that contradicts the assumption that G has no directed cycle!

That’s All, Folks!

Thanks for coming to section this week!
Any questions?