CSE 421 Section 2

Graph Search

Administrivia

Announcements & Reminders

- HW1
  - Was due yesterday, 1/11
  - Remember, this quarter we have a LATE PROBLEMS policy, instead of a late assignments policy
    - Can turn in up to **6 problems late**
    - Can turn in a late problem up to **48 hours late**

- HW2
  - Due Wednesday 1/18 @ 11:59pm

Proof by Contradiction

Contradiction

- In addition to induction, proof by contradiction is another really common technique we will use to prove that algorithms are correct in this class.

- Reminder: proof by contradiction is the 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, so therefore 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!

Problem 2 - Write it Better: Proof by Contradiction

Claim: For every simple graph G, if every node of G has degree at least 2, then G has a cycle.

a) Prove the claim using proof by contradiction.

Starting point:

Suppose, for the sake of contradiction, there is a graph G such that every node of G has degree at least 2, but G has no cycle.

Problem 2 - Write it Better: Proof by Contradiction

It's common in proofs by 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! But not particularly elegant. And you have to write down enough steps that your reader knows what the pattern is, which could be a lot.

The more elegant version is to use ***proof by contradiction with extremality***. 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 A and jump right to Option B.

Problem 2 - Write it Better: Proof by Contradiction

Claim: For every simple graph G, if every node of G has degree at least 2, then G has a cycle.

b) Rewrite the proof, using the proof by contradiction with extremality technique.

NEW, EXTREME Starting point:

Suppose, for the sake of contradiction, there is a graph G such that every node of G has degree at least 2, but G has no cycle. Let $P = v_0, v_1, \ldots, v_k$ be a longest simple path in G.

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

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

(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 the bucket

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 the bucket

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 the bucket

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 $c_1, \ldots, c_{10}$, ranging from 1 to C. Your goal is to determine whether it is possible to get exactly t gallons into the bucket

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 the bucket

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 $f(n)$ is eventually always $< c \cdot g(n)$
  - $f(n)$ is $\boldsymbol{\Omega(g(n))}$ iff there is a constant $c > 0$ so that $f(n)$ 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 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.

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

That's All, Folks!

Thanks for coming to section this week!
Any questions?