Problem Solving Strategies

CSE 417 Winter 21
Lecture 18

Updates

Most of you already used three or all of your late days, and we've still got almost half of the course left.

We're going to give everyone 2 extra late days.

I'm still getting some feedback that homeworks are taking many of you a long time.

And I've gotten requests to talk about approaching problems in interviews.

We're going to spend today talking about how to approach problems in general so that you can use the time you're spending more effectively.

More Updates

We have to rearrange the schedule as a result

HW6 will be 1.5 times the length of the last few homeworks with twice as long to do it, due on March 5th We'll release it on Friday, but much of it will only make sense partway through next week.

We'll only have seven homeworks instead of eight.

An analogy

When you were first programming, debugging took a long time.

Even for “just” off-by-one errors.

Because you didn’t have a lot of practice debugging.

And if you got frustrated it was easy to start randomly changing < to <= one at a time in all combinations until one of them worked.

Or until you realized the bug was in some other part of the code.

You might have gotten the answer at the end of three hours of changing < to <=. But if you stopped and tried a different strategy

Pandemic-Specific Advice

It’s hard to focus on work right now.

Designing algorithms needs focus.

If you’re feeling frustrated, take a break.

Spread out the work you’re doing to give yourself time to digest new ideas.

It really helps to bounce ideas off of others – come to office hours!

General Advice

Spend extra time making sure you understand the problem statement at the start.

Make examples and solve them yourself early on.

Start with a baseline/brute force solution, even if it’s hopelessly inefficient.

General Advice

Find a homework buddy – human, or [stuffed] animal.

When you’re banging your head against a wall, explain to your buddy what you think is going wrong.

When you’re stuck, ask questions!

“Have I seen a similar problem before?” “Does this remind me of anything?”

“If I make the problem **easier**, Can I solve it?”

A Process

TECHNICAL INTERVIEWS

Talk out the problem statement

Examples

Brute force

Optimize

Walk through

Implement

Test/Debug

HOMEWORK

Understand the statement

Do some examples

Is there a starting algorithm?

Common Questions

Can I simplify the problem? Have I seen this before?

Write pseudocode

Analyze and Justify

Example Problem

Let $G = (V, E)$ be an undirected graph. Let $e = \{u, v\}$ be an edge in G. Give an $O(n + m)$ time algorithm that finds the shortest cycle in G which contains the edge e. Explain why your algorithm is correct.

Adapted from the Autumn 417 midterm.

Talk

Let $G = (V, E)$ be an undirected graph. Let $e = \{u, v\}$ be an edge in G. Give an $O(n + m)$ time algorithm that finds the shortest cycle in G which contains the edge e. Explain why your algorithm is correct.

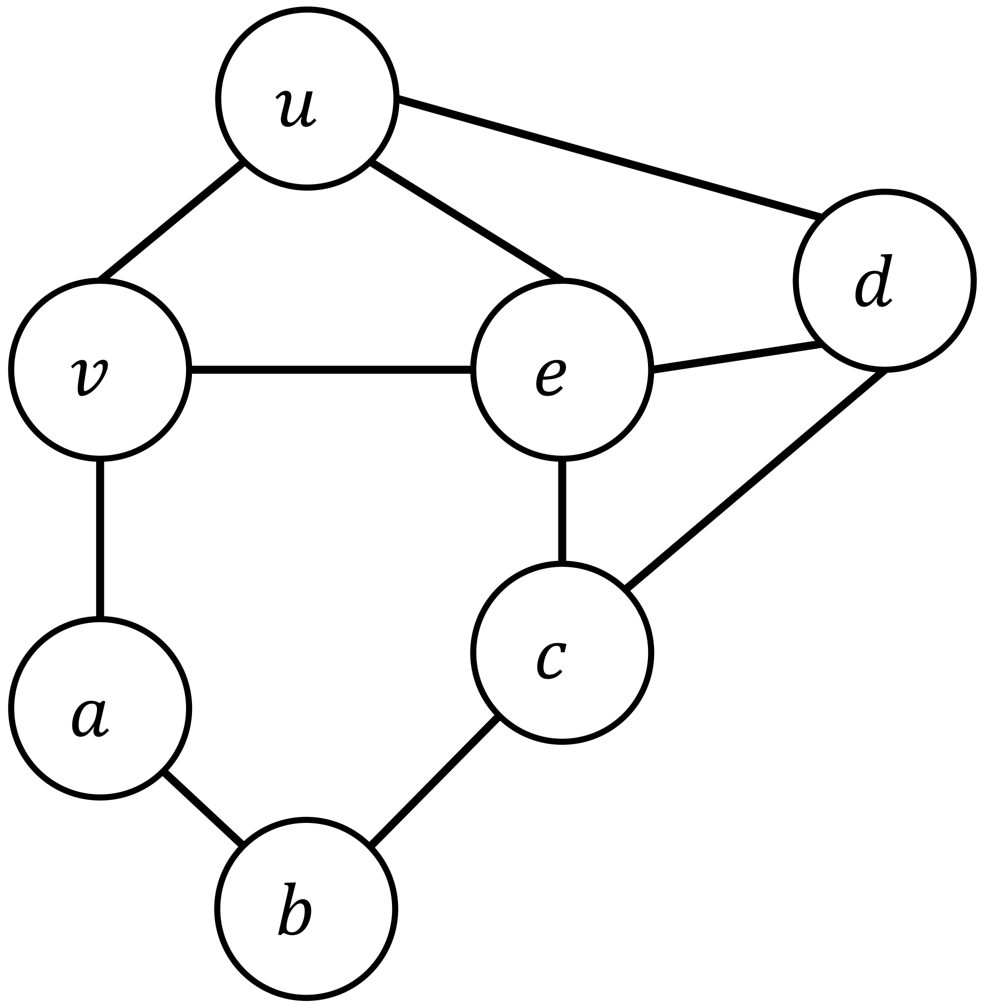
Do you understand each individual word?

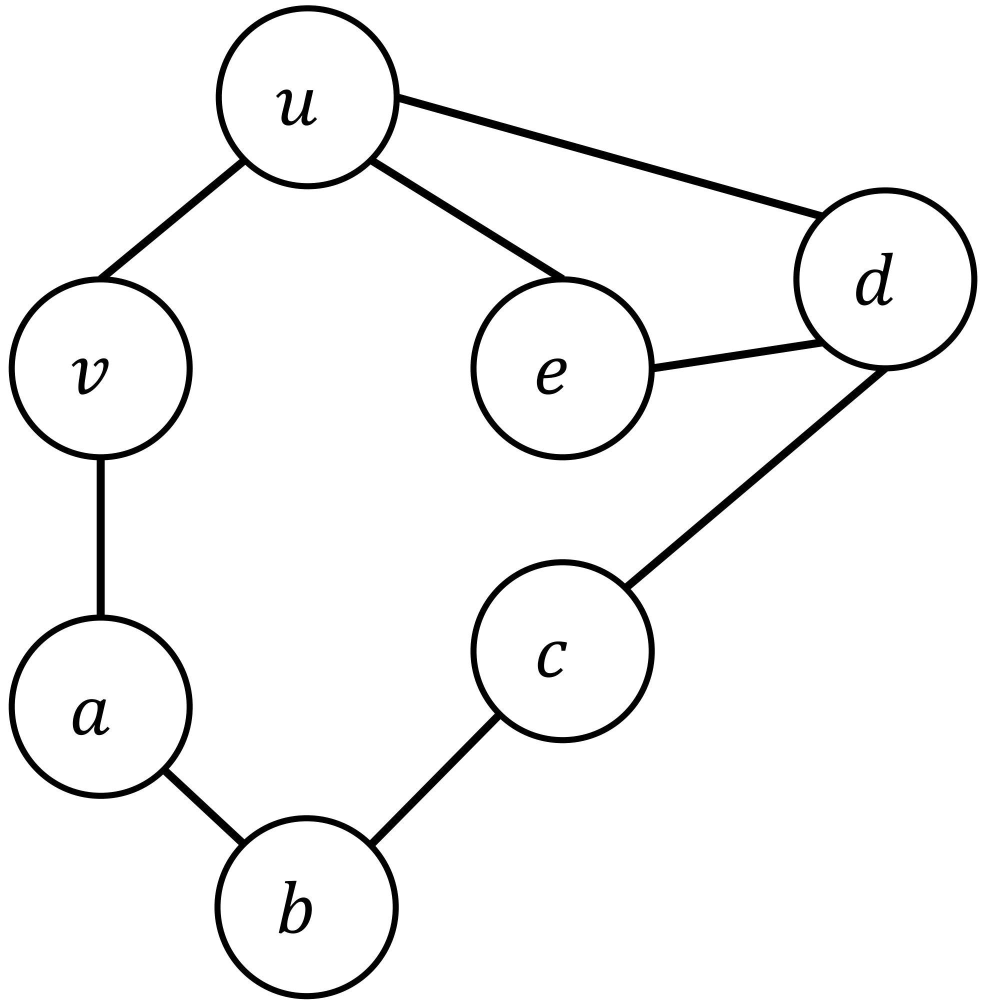
Do you understand the problem as a whole?

What would the method signature be (return type, parameters)?

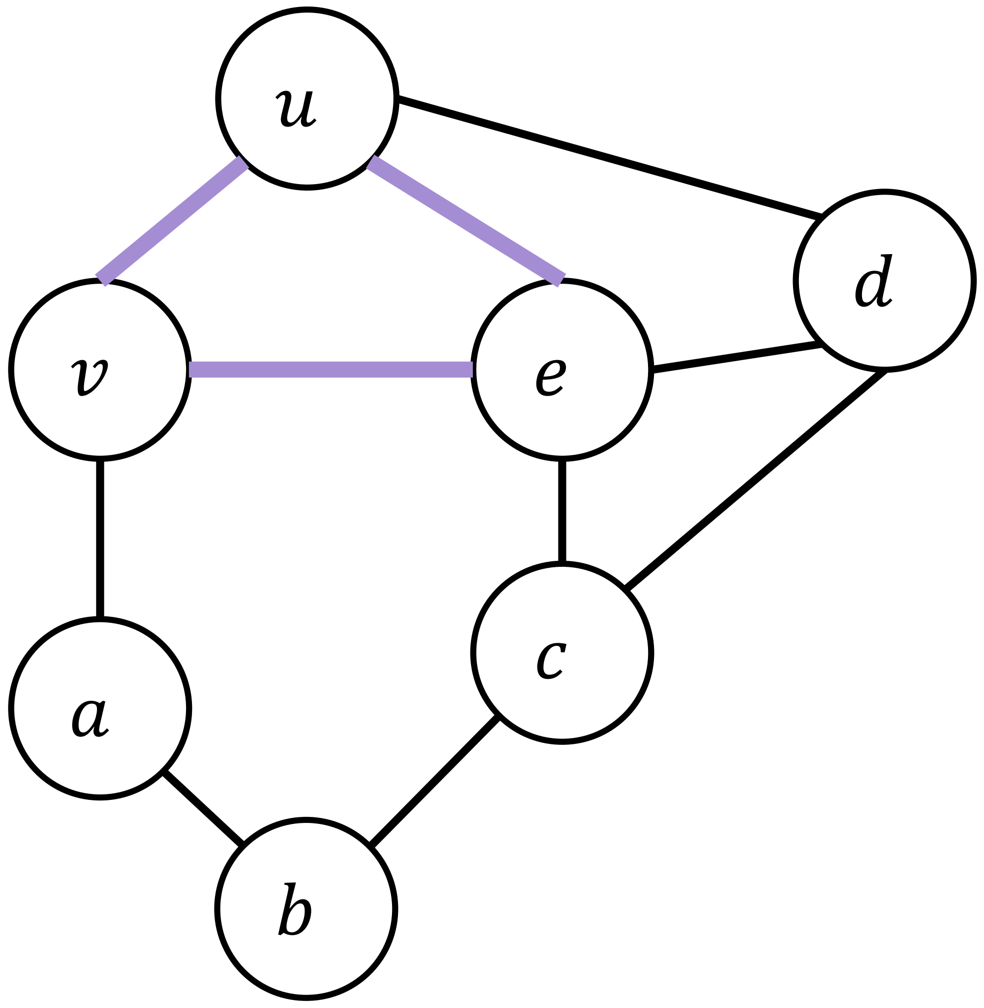
Fill out the poll everywhere for Activity Credit!
Go to pollev.com/cse417 and login with your UW identity

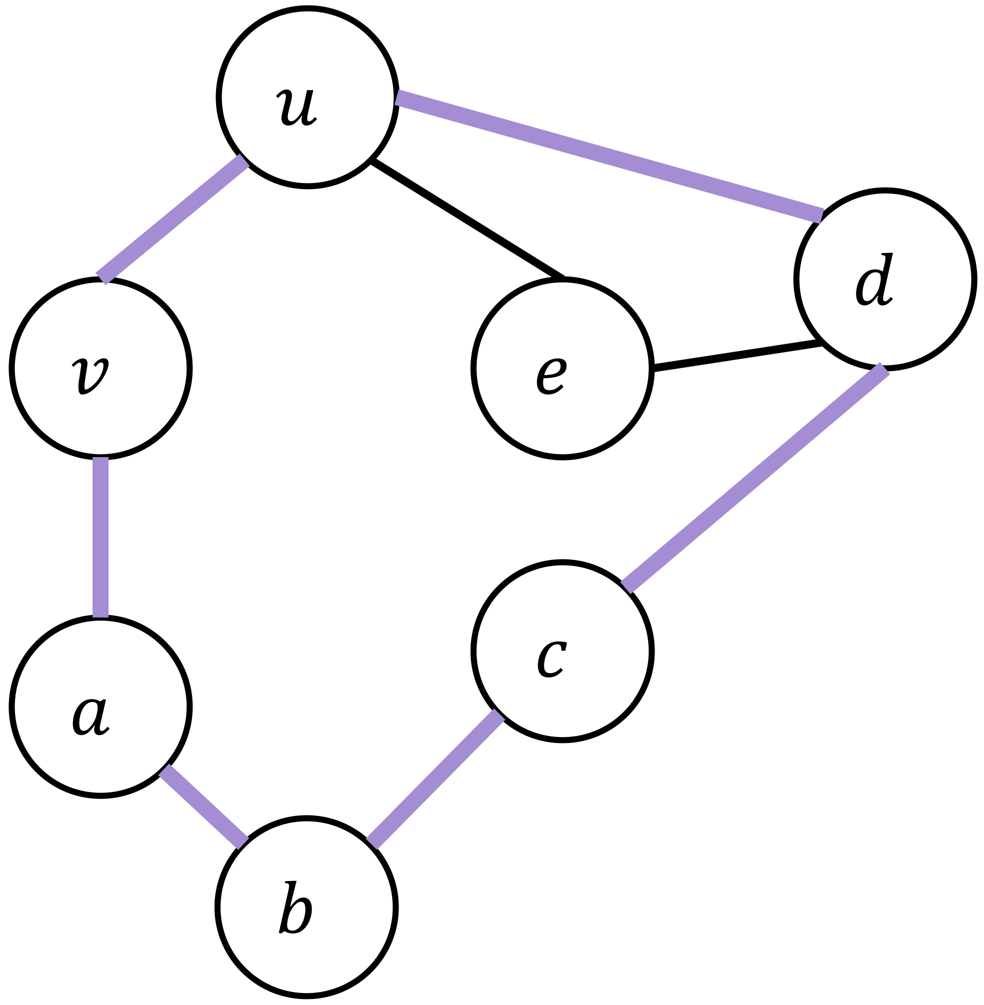
Examples





Examples





Brute Force/Baseline

Come up with an algorithm **any algorithm** that works.

Benefit 1: Sometimes you can improve the efficiency until you get a working algorithm (especially the case with DP!)

Benefit 2: Stress reduction.

You have something. Put it in your back pocket. Worst case, you’re writing something down.

Brute Force/Baseline

Uhhhh…

I guess check the 3-cycles to see if they include e.

Then the 4-cycles

Then the 5-cycles,…

How do we get all the 3-cycles involving (u, v)? Check all possible third vertices. So $O(n)$ already.

4 cycles, now check all possible pairs $O(n^2)$ ….

This is gonna be slow.

Optimize

That brute force algorithm is bad

Hopelessly bad. We aren't going to speed it up.

Let's try to come up with something better…

Good questions to ask:

does this problem remind you of anything?

Is there a similar problem we've already studied?

Similar Problems

We detected cycles in **directed** graphs with depth-first-search.

In undirected graphs, we accidentally found **odd-length** cycles with BFS.

When we were 2-coloring, if the 2-coloring failed it was because of an intralayer edge that corresponded to an odd-cycle.

We found short **paths** in graphs.

Any of these might look like a reasonable starting point

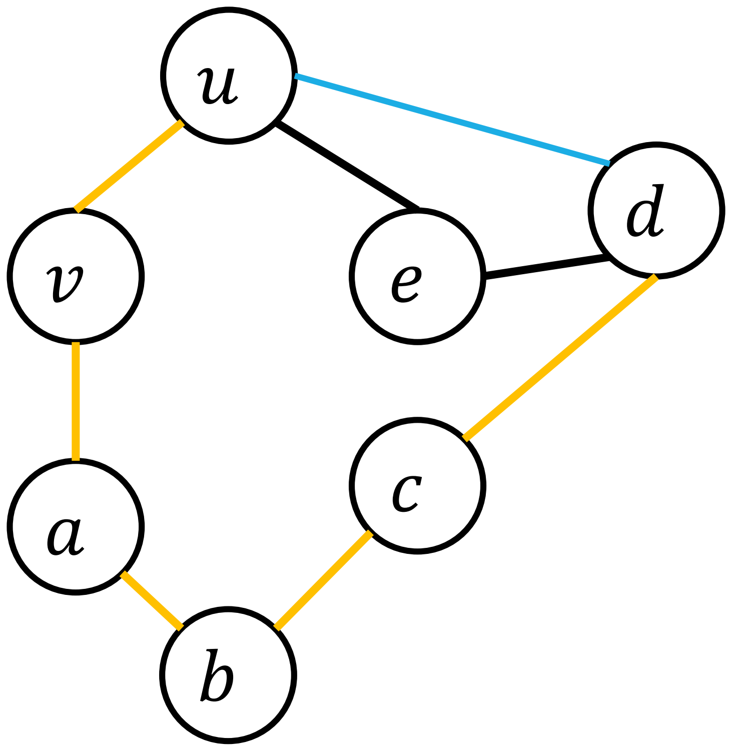
Choose Your Own Adventure

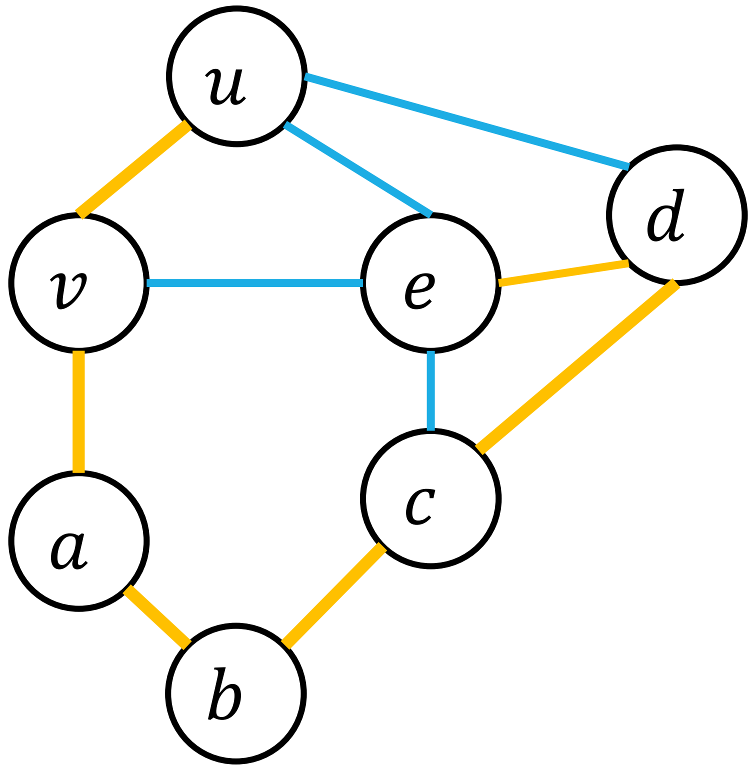
Depth First Search

Depth First Search?

Probably want to start by processing u, v
Sometimes it's the first back-edge we find!

But sometimes it's not…





Why isn’t DFS working?

Keep asking questions – when your algorithm isn’t working, try to get a high-level description of why.

We can find a cycle with a back edge. But it’s hard to tell which back edge(s) are part of the shortest cycle.

We could try to keep track of distances…like how fast you could get from w back to u, v

Those sound like distances….wait that’s what BFS does!

Choose Your Own Adventure

Modify BFS

Let’s try BFS instead

Ask a simpler question.

We know BFS finds odd cycles. What if the shortest cycle in the whole graph goes through u. AND it’s odd-length.
Can it find the shortest odd-cycle going through the **starting vertex**?

Starting from u, let the shortest cycle be $u, v_1, v_2, \ldots, v_k, w_k, w_{k-1}, \ldots, w_1$

Start at u

Then v_1 and w_1 MUST go on the queue. And none of the other vertices on that short cycle (that would mean there’s an edge from u to those other vertices, but then there’s a shorter cycle going through u!)

Let’s Try BFS Instead

That pattern continues

u_i w_i both go on the queue. We can’t have any intralayer edges, as those would give us odd-cycles shorter than the shortest one!

And then we get an intra-layer edge u_k, w_k

Woohoo!

Making the question harder.

We’ve answered the easy question:

We can find the shortest cycle in the whole graph if it has odd-length and goes through u.

Let’s make it a little harder. Now we want to go through the edge (u, v)

Making it an edge

For going through u, we just put u on the queue first.

Can we put both u and v on the queue right at the start?

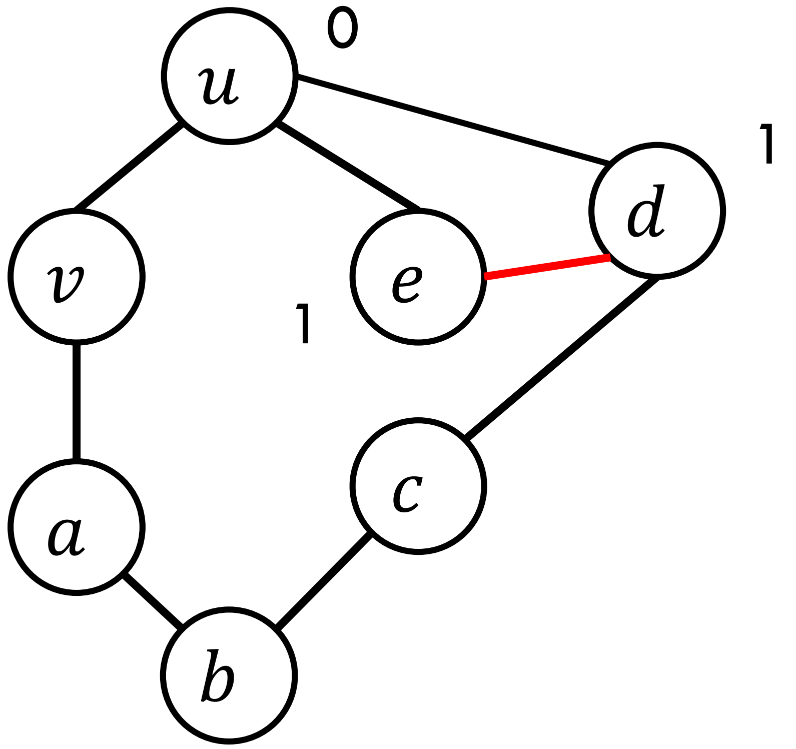
Yes! Make that the first layer.

Another way to think of it – what if we could "combine" the edge into a vertex ("squash it down")

That's exactly what putting both u and v on the queue to start does.

Almost There…

What if the cycle isn't the shortest one overall?

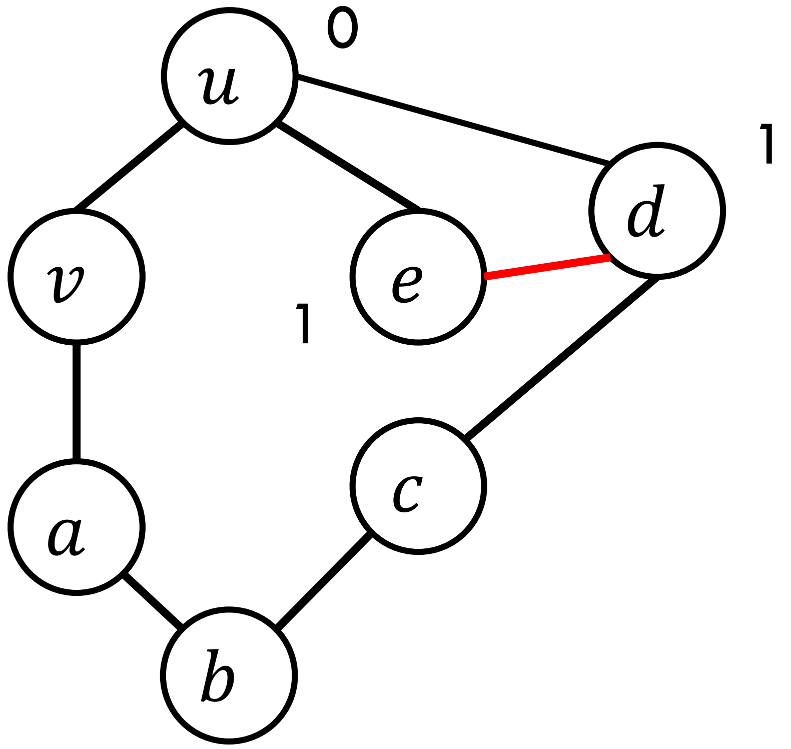


(d, e) will be an intralayer edge, but it's associated with a useless cycle…

What's the problem?

Almost There…

What if the cycle isn't the shortest one overall?



(d, e) will be an intralayer edge, but it's associated with a useless cycle…

What's the problem?
Both d and e come from u, we need one to trace back to v.
Keep track of who we trace back to!

Almost There…

What happens in the length is odd?

Instead of an intralayer edge, we'll have a single vertex discovered twice…

Once from u and once from v. We can check for that too!

Modified BFS

```
CycleFinder(G,u,v)
  u.side=u, v.side=v, u.layer=0, v.layer=0
  delete edge (u,v)
  Place u and v on queue
  Place divider on queue
  while(queue is not empty)
    curr = queue.dequeue()
    if(curr==divider)
        layer++
      for(each edge (curr,w))
             if(w is not seen)
                    w.layer=curr.layer+1
                    queue.enqueue(w)
                    w.seen=true
                    w.pred=curr
                    w.side=curr.side
             else
                    if(w.side!=curr.side) //a real cycle
                           findCycle(u,v,curr,w)
                    //otherwise do nothing. Not a real cycle and w already on queue.
```

Helper

```
//finds cycle by backtracking from x,y until reaching u,v
findCycle(u,v,x,y)
list cycle
cycle.insertFront(x)
cycle.insertFront(y)
while(x != u && x != v)
    cycle.insertFront(x)
    x=x.pred
cycle.insertFront(x)
while(y != 6 && y != v)
    cycle.insertBack(y)
    y=y.pred
cycle.insertBack(y)
return cycle
```

Choose Your own Adventure

Shortest Paths

What is a cycle

A cycle is just a path from u to v (not using the edge (u, v)) with the edge (u, v) added on.

So we know the edge…

We just need to find the path. Which path do we want?

The shortest one that doesn't use (u, v)

Algorithm

```
FindCycle(G,u,v)
    Delete (u,v) from G
    Run BFS-Shortest-Path(G, u) //i.e. shortest paths starting from u.
    List cycle
    curr = v
    while(curr != u)
        cycle.insertBack(curr)
        curr=curr.pred
    cycle.insertBack(curr) // i.e. insert u
    return cycle
```

Summary

Talk/Read carefully – understand individual words of question, then question as a whole.

Do **E**xamples – confirms you understand the problem, get more intuition.

Brute Force Solution – sometimes can be optimized to a final answer, sometimes not…but a backup plan can take the pressure off.

Optimize – find a better algorithm. Ask yourself questions:

Have I seen a similar problem before?

Can I make the problem easier? Can I handle a special case?

Summary

Walk Through your idea on an example. Just to see that it's not way off.

Implement – write the pseudocode (or real code)

Test – you already wrote some examples – do they work? Did you think of other edge cases when you were implementing? Check those too.

An Extra Example

Sledding

Lenny and Bill are going to go sledding! They've set up n ramps on a hill, and want to plan the ultimate sledding trip.

Whenever Lenny or Bill reaches a ramp while on the ground, they can either use that ramp to jump through the air, possibly flying over one or more ramps, or sled past that ramp and stay on the ground. Obviously, if someone flies over a ramp, they cannot use that ramp to extend their jump.

Suppose you are given a pair of arrays Ramp[1 .. n] and Length[1 .. n], where Ramp[i] is the distance from the top of the hill to the ith ramp, and Length[i] is the distance that any sledder who takes the ith ramp will travel through the air. Describe and analyze an algorithm to determine the maximum total distance that Lenny or Bill can spend in the air

Adapted from http://jeffe.cs.illinois.edu/teaching/algorithms/book/03-dynprog.pdf, problem 15

Step 1: Talk – Ask Questions

The goal of this step is to make sure you understand the problem statement.

In technical interviews, interviewers (at least in the past…) would sometimes leave questions somewhat ambiguous intentionally to see how you ask questions.

Robbie’s usual self-checks:

Do you understand all the **words** in the problem individually?

Do you understand the statement as a whole?

What’s our return type?

Are there any edge cases that immediately stick out?

Edge Case

If we take a ramp at 5 and we're in the air for 3, can we take the ramp at 8?

Not totally clear in the problem – we'll say yes!

Step 2: Examples

Good way to ensure you've understood the question

And to get started on solving it.

If we give you examples on a homework, use those. Cover up the "answers" and try to find them yourself.

If you aren't given them, just generate you own instances and solve them.

Step 2: Examples

Ramp[]

1	2	3	4	5
0	15	16	20	25

Length[]

1	2	3	4	5
10	2	4	6	1

We can take ramp 1 (flying through the air from 0 to 10)	10 units so far
Take ramp 2 (flying through the air from 15 to 17)	12 units so far
We are not allowed to take 3 (in the air)	12 units so far
We choose to skip ramp 4.	12 units so far
Take ramp 5 (in the air from 25 to 26)	13 units total.

Step 2: Examples

Ramp[]

1	2	3	4	5
0	15	16	20	25

Length[]

1	2	3	4	5
10	2	4	6	1

We can take ramp 1 (flying through the air from 0 to 10)	10 units so far
Skip ramp 2	10 units so far
Take ramp 3 (flying through the air from 16 to 20)	14 units so far
Take ramp 4 (flying through the air from 20 to 26)	20 units so far
We cannot take ramp 5 (in the air)	20 units total.

Step 2: More Examples

Ramp[]

1	2	3	4	5
0	5	10	15	20

Length[]

1	2	3	4	5
6	4	3	6	1

Step 2: More Examples

Ramp[]

1	2	3	4	5
0	5	10	15	20

Length[]

1	2	3	4	5
6	4	3	6	1

We can skip ramp 1	0 units so far
Take ramp 2 (flying through the air from 5 to 9)	4 units so far
Take ramp 3 (flying through the air from 10 to 13)	7 units so far
Take ramp 4 (flying through the air from 15 to 21)	13 units so far
We cannot take ramp 5 (in the air)	13 units total.

Step 3: Brute Force (or just any algorithm)

In a technical interview: it’s a good idea to just state the first algorithm idea you think of. Even if it’s slow.

In our context: sometimes skippable, but often a good idea.

It takes pressure off. Worst case scenario you write that one up.

You’ll also often see more about the problem/see the optimization to get to a better solution.

Step 3: Brute Force

Just try every possible plan.

Spin off a method to be take this ramp and another skip this ramp.

(Need a check to make sure ramp is valid)

Step 3: Brute Force

```
BruteForceRamp(int score, int index, ramps, length)
    //base case

    //skip

    //take ramp
```

Step 3: Brute Force

```
BruteForceRamp(int score, int index, ramps, length)
    //base case
    if(index > n) return 0
    //skip ramp
    int s=(score, index+1, ramps,length)
    //take ramp
    int newInd=findInd(ramps[index]+length[index])
    int t=(score+length[index],newInd,ramps,length)
    return max(s,t)
```

Step 4: Optimize

Turn that code into a more efficient version. Can we use DP instead?

What parameters do we really need?

```
int s=(score, index+1, ramps,length)
int t=(score+length[index],newInd,ramps,length)
```

Score made things easier when we weren't sure what order to evaluate the recurrence. But we don't need it.

Step 4: Optimize

Turn that code into a more efficient version. Can we use DP instead?

What parameters do we really need?

```
int s=(score, index+1, ramps,length)
int t=(score+length[index],newInd,ramps,length)
```

Ramps, length would be needed in java code, but aren't changing.

Optimized

Define OPT(i) as the maximum time in the air from i to end, starting from the ground at ramp i.

$$OPT(i) = \begin{cases} 0 & \text{if } i > n \\ \max\{OPT(i+1), \text{length}[i] + OPT(j)\} & \text{otherwise} \end{cases}$$

Where $j = \min_{k}\{\text{ramps}[k] \leq \text{ramps}[i] + \text{length}[i]\}$

Try This one on your own

The university lawyers heard about Lenny and Bill’s little bet and immediately objected. To protect the university from either lawsuits or sky-rocketing insurance rates, they impose an upper bound on the number of jumps that either sledder can take. Describe and analyze an algorithm to determine the maximum total distance that Lenny or Bill can spend in the air with at most k jumps, given the original arrays Ramp[1 .. n] and Length[1 .. n] and the integer k as input.

Do all the steps – Start by altering the examples and making sure you understand the change!