

No physical handouts
but pdf on the calendar.

DP5
End of Arrays; DP on trees

CSE 421 Winter 23
Lecture 14

Midterm Information

Practice Midterm is posted, along with some other resources.

Section Thursday will be “pick what problems you want to practice”

Second-half of Friday’s lecture will be a “what have we seen so far?” review (not new problems, but at least a list of what you’ve seen).

Monday, the TAs will spend the lecture slot answering questions on the practice midterm (or anything else you want to ask about).

Please fill out the conflict form today so we can work on scheduling.

If you are sick for the exam, we’ll schedule a conflict once you’re well (we’ll have a separate mode of telling us, not that form).

Please come in-person ~~tomorrow~~

Wednesday

Ken Yasuhara (from College of Engineering's Center for Teaching and Learning) will do a mid-quarter feedback session Wednesday.

It'll take the first about 20 minutes; we'll have a normal half-of-a-lecture slot after that.

Feedback is super useful to us!

Edit Distance

More formally:

The edit distance between two strings is:

The minimum number of **deletions**, **insertions**, and **substitutions** to transform string x into string y .

Deletion: removing one character

Insertion: inserting one character (at any point in the string)

Substitution: replacing one character with one other.

Example

What's the distance between babyyodas and tastysoda?

B	A	B		Y	Y	O	D	A	S
sub		sub	ins		sub				del
T	A	S	T	Y	S	O	D	A	

Distance: 5, one point for each colored box

Quick Checks – can you explain these?

If x has length n and y has length m , the edit distance is at most $\max(x, y)$

The distance from x to y is the same as from y to x (i.e. transforming x to y and y to x are the same)

Finding a recurrence

What information would let us simplify the problem?

What would let us “take one step” toward the solution?

“Handling” one character of x or y

i.e. choosing one of insert, delete, or substitution and increasing the “distance” by 1

OR realizing the characters are the same and matching for free.

$OPT(i, j)$ is the minimum number of insertions, deletions, and substitutions to transform $x_1x_2 \cdots x_i$ into $y_1y_2 \cdots y_j$. (we’re indexing strings from 1, it’ll make things a little prettier).

The recurrence

“Handling” one character of x or y

i.e. choosing one of insert, delete, or substitution and increasing the “distance” by 1

OR realizing the characters are the same and matching for free.

What does delete look like? $OPT(i - 1, j)$ (delete character from x match the rest)

Insert $OPT(i, j - 1)$ Substitution: $OPT(i - 1, j - 1)$

Matching characters? Also $OPT(i - 1, j - 1)$ but only if $x_i = y_j$

The recurrence (v1, we'll improve soon)

"Handling" one character of x or y

i.e. choosing one of insert, delete, or substitution and increasing the "distance" by 1

OR realizing the characters are the same and matching for free.

$$OPT(i, j) = \begin{cases} \min\{ \overset{\text{Delete}}{1 + OPT(i-1, j)}, \overset{\text{Insert}}{1 + OPT(i, j-1)}, \overset{\text{Substitution}}{1 + OPT(i-1, j-1)}, \overset{\text{TODO: Just Match}}{0} \} & \text{if } i > 0 \text{ and } j > 0 \\ j & \text{if } i = 0 \\ i & \text{if } j = 0 \end{cases}$$

The recurrence (v1, we'll improve soon)

"Handling" one character of x or y

i.e. choosing one of insert, delete, or substitution and increasing the "distance" by 1

OR realizing the characters are the same and matching for free.

$$OPT(i, j) = \begin{cases} \min\left\{ \underbrace{1 + OPT(i-1, j)}_{\text{Delete}}, \underbrace{1 + OPT(i, j-1)}_{\text{Insert}}, \underbrace{1 + OPT(i-1, j-1)}_{\text{Substitution}}, \underbrace{OPT(i-1, j-1) + \infty \cdot \mathbb{I}\{x_i \neq y_j\}}_{\text{Just Match}} \right\} \\ \text{if } i = 0 \\ \text{if } j = 0 \end{cases}$$

Idea: only allow "just match" when you can just match.

Otherwise make it ∞ (will never be the min).

In code: if/else branch, probably. This is a math notation trick.

"Indicator"
like from 312

The recurrence

“Handling” one character of x or y

i.e. choosing one of insert, delete, or substitution and increasing the “distance” by 1

OR realizing the characters are the same and matching for free.

$$OPT(i, j) = \begin{cases} \min\{ \overset{\text{Delete}}{1 + OPT(i - 1, j)}, \overset{\text{Insert}}{1 + OPT(i, j - 1)}, \overset{\text{Sub and matching}}{\mathbb{I}[x_i \neq y_j] + OPT(i - 1, j - 1)} \} & \text{if } i = 0 \\ j & \text{if } j = 0 \\ i & \end{cases}$$

When we could match, we will never substitute; matching will always give us a better score! Still have to check delete, insert (those could be better).

The recurrence

“Handling” one character of x or y

i.e. choosing one of insert, delete, or substitution and increasing the “distance” by 1

OR realizing the characters are the same and matching for free.

$$OPT(i, j) = \begin{cases} \min\{ \overset{\text{Delete}}{1 + OPT(i-1, j)}, \overset{\text{Insert}}{1 + OPT(i, j-1)}, \overset{\text{Sub and matching}}{\mathbb{I}[x_i \neq y_j] + OPT(i-1, j-1)} \} & \text{if } i = 0 \\ & \text{if } j = 0 \end{cases}$$

Edit Distance

B	A	B	Y
T	A	S	

Current subproblem: edit dist
between BABY and TAS

$OPT(i, j)$	0	B, 1	A, 2	B, 3	Y, 4	Y, 5	O, 6	D, 7	A, 8	S, 9
0	0	1	2	3	4	5	6	7	8	9
T 1	1									
A 2	2									
S 3	3									
T 4	4									
Y 5	5									
S 6	6									
O 7	7									
D 8	8									
A 9	9									

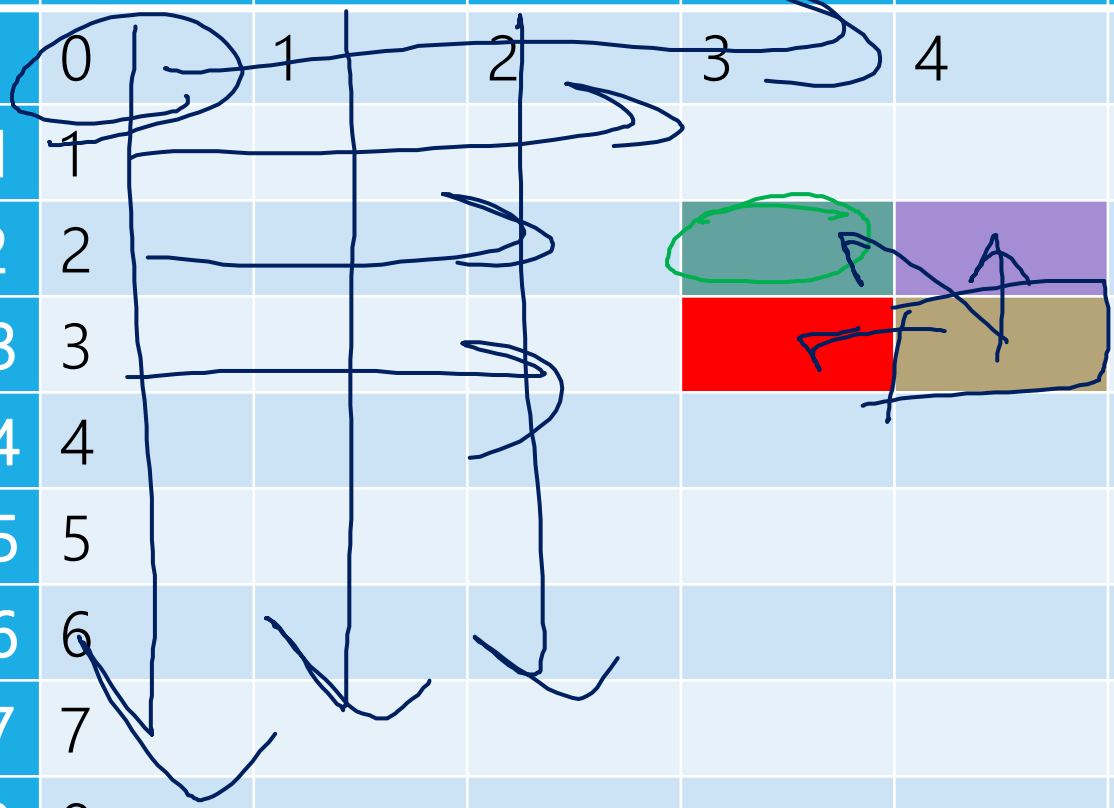
$OPT(4, 3)$

Edit Distance

B	A	B	Y
T	A	S	

Sub: Transform Y to S (cost 1)
 BAB
 TA

OPT(i,j)	0	B, 1	A, 2	B, 3	Y, 4	Y, 5	O, 6	D, 7	A, 8	S, 9
0	0	1	2	3	4	5	6	7	8	9
T	1									
A	2									
S	3									
T	4									
Y	5									
S	6									
O	7									
D	8									
A	9									



Edit Distance

BA3Y4
TAS1Y

Fill in the next two entries. Be careful with the sub/match distinction!

OPT(i,j)	0	B, 1	A, 2	B, 3	Y, 4	Y, 5	O, 6	D, 7	A, 8	S, 9
0	0	1	2	3	4	5	6	7	8	9
T 1	1	1	2	3	4	5	6	7	8	9
A 2	2	2	1	2	3	4	5	6	7	8
S 3	3	3	2	2	3	4	5	6	7	7
T 4	4	4	3	3	3	4	5	6	7	8
Y 5	5	5	4	4	3	4				
S 6										
O 7										
D 8										
A 9										

Handwritten annotations in the table: a circled '3' in row T, column Y; a circled '3' in row Y, column Y; and a circled '4' in row Y, column O.

Edit Distance

Fill in the next two entries. Be careful with the sub/match distinction!

$OPT(i, j)$	0	B, 1	A, 2	B, 3	Y, 4	Y, 5	O, 6	D, 7	A, 8	S, 9
0	0	1	2	3	4	5	6	7	8	9
T 1	1	1	2	3	4	5	6	7	8	9
A 2	2	2	1	2	3	4	5	6	7	8
S 3	3	3	2	2	3	4	5	6	7	7
T 4	4	4	3	3	3	4	5	6	7	8
Y 5	5	5	4	4	3	3	4			
S 6										
O 7										
D 8										
A 9										

Y's match, so sub is free!

Edit Distance

$OPT(i, j)$	0	B, 1	A, 2	B, 3	Y, 4	Y, 5	O, 6	D, 7	A, 8	S, 9
0	0	1	2	3	4	5	6	7	8	9
T 1	1	1	2	3	4	5	6	7	8	9
A 2	2	2	1	2	3	4	5	6	7	8
S 3	3	3	2	2	3	4	5	6	7	7
T 4	4	4	3	3	3	4	5	6	7	8
Y 5	5	5	4	4	3	3	4	5	6	7
S 6	6	6	5	5	4	4	4	5	6	6
O 7	7	7	6	6	5	5	4	5	6	7
D 8	8	8	7	7	6	6	5	4	5	6
A 9	9	9	8	8	7	7	6	6	4	5

Edit Distance – what operations?

$OPT(i, j)$	0	B, 1	A, 2	B, 3	Y, 4	Y, 5	O, 6	D, 7	A, 8	S, 9
0	0	1	2	3	4	5	6	7	8	9
T 1	1	1	2	3	4	5	6	7	8	9
A 2	2	2	1	2	3	4	5	6	7	8
S 3	3	3	2	2	3	4	5	6	7	7
T 4	4	4	3	3	3	4	5	6	7	8
Y 5	5	5	4	4	3	3	4	5	6	7
S 6	6	6	5	5	4	4	4	5	6	6
O 7	7	7	6	6	5	5	4	5	6	7
D 8	8	8	7	7	6	6	5	4	5	6
A 9	9	9	8	8	7	7	6	6	4	5

Edit Distance – what operations?

$OPT(i, j)$	0	B, 1	A, 2	B, 3	Y, 4	Y, 5	O, 6	D, 7	A, 8	S, 9
0	0	1	2	3	4	5	6	7	8	9
T 1	1	1	2	3	4	5	6	7	8	9
A 2	2	2	1	2	3	4	5	6	7	8
S 3	3	3	3	2	3	4	5	6	7	7
T 4	4	4	4	3	3	4	5	6	7	8
Y 5	5	5	5	4	4	3	4	5	6	7
S 6	6	6	6	5	5	4	4	5	6	6
O 7	7	7	7	6	6	5	5	4	5	7
D 8	8	8	8	7	7	6	6	5	4	6
A 9	9	9	9	8	8	7	7	6	6	5

Dynamic Programming Process

1. Define the object you're looking for

$OPT(i, j)$ is the minimum number of insertions, deletions, and substitutions to transform $x_1x_2 \cdots x_i$ into $y_1y_2 \cdots y_j$.

2. Write a recurrence to say how to find it



3. Design a memoization structure

$m \times n$ Array

4. Write an iterative algorithm

Outer loop: increasing j (starting from 1)

Inner loop: increasing i (starting from 1)

DP Proofs

We generally **won't** ask you for proofs of correctness on dynamic programming problems. (maybe one after the midterm)

Why?

The proofs are always inductive proofs where you say

“my recurrence checks all the possibilities” or, equivalently

“The maximum thing has to be made up of the best thing for all these other subproblems.”

The proof itself is a lot of boilerplate and hard to write clearly (you have to differentiate between your recurrence and what your recurrence intends to calculate, which can be tricky).

DP Proofs

One example in the slides, another proof will appear in section solutions sometime in the next few weeks so you know what you're (not) missing.

Instead, we're usually going to ask for your intuition on what your recurrence is doing (what do all the cases correspond to/why are they exhaustive)?

The proof is just a lot of formalism on that key idea. So we're going to have you focus on the idea, not the formalism.

Goal of DP

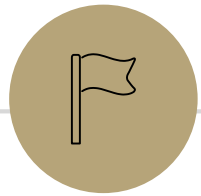
Just try all the (reasonable) possibilities.

Don't worry about greedily choosing the best, use recursion to "look ahead" for all the best options, and pick the best one.

There is a "greedy-ish" alteration to the Edit Distance recurrence...

It turns out, if the two characters match, that will always be at least as good as the insert/delete options.

But it's fine to **not** notice! And if you thought it was safe but wasn't, well....



DP on Trees



DP on Trees

Trees are recursive structures

A tree is a root node, with zero or more children

Each of which are roots of trees

Since DP is “smart recursion” (recursion where we save values)

Recursive functions/calculations are really common.

DP on Trees

Find the minimum vertex cover in a tree.

Give every **vertex** a weight, find the minimum weight vertex cover

Vertex Cover

A set S of vertices is a vertex cover if for every edge (u, v) :
 u is in S , or v is in S , (or both)

The weight of a vertex cover is just the sum of the weights of the vertices in the set.

We want to find the minimum weight vertex cover.

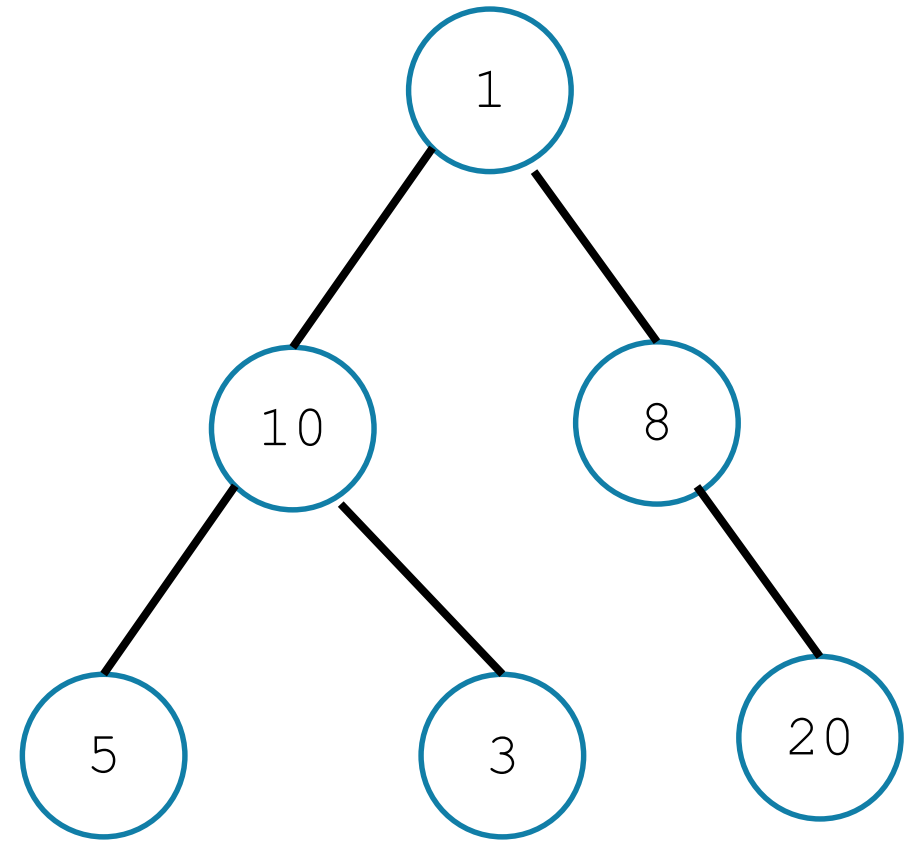
Vertex Cover

Vertex Cover

A set S of vertices is a vertex cover if for every edge (u, v) : u is in S , or v is in S , (or both)

Find the minimum vertex cover in a tree.

Give every **vertex** a weight, find the minimum weight vertex cover



Vertex Cover

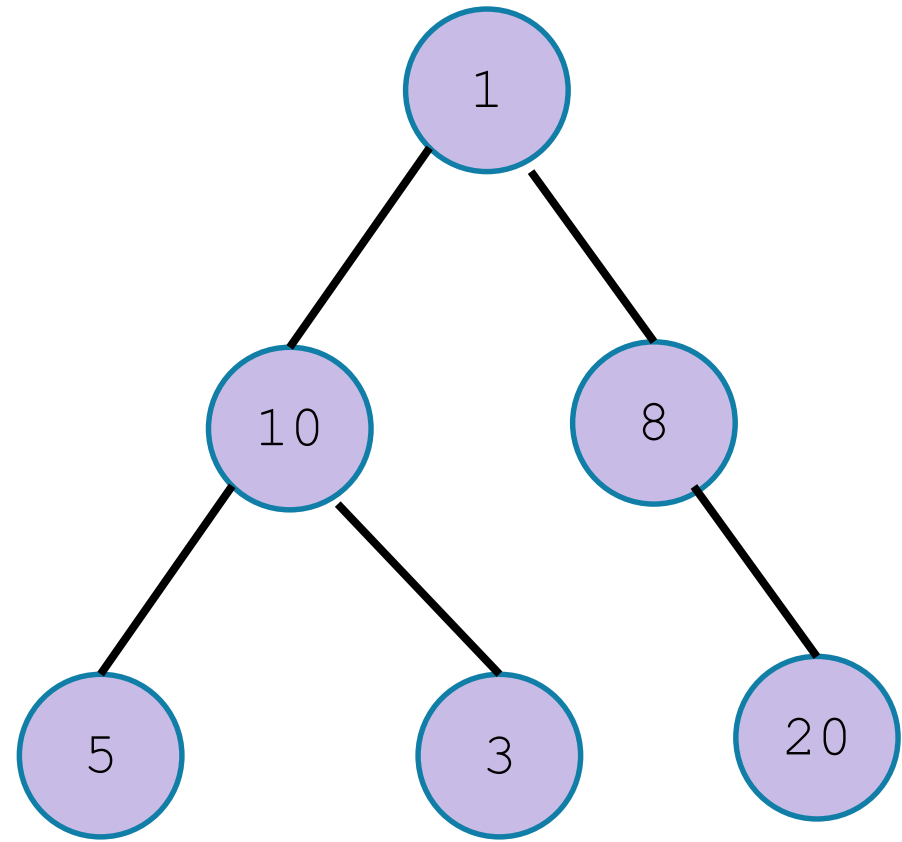
Vertex Cover

A set S of vertices is a vertex cover if for every edge (u, v) : u is in S , or v is in S , (or both)

Find the minimum vertex cover in a tree.

Give every **vertex** a weight, find the minimum weight vertex cover

A valid vertex cover! (just take everything)
Definitely not the minimum though.



Vertex Cover

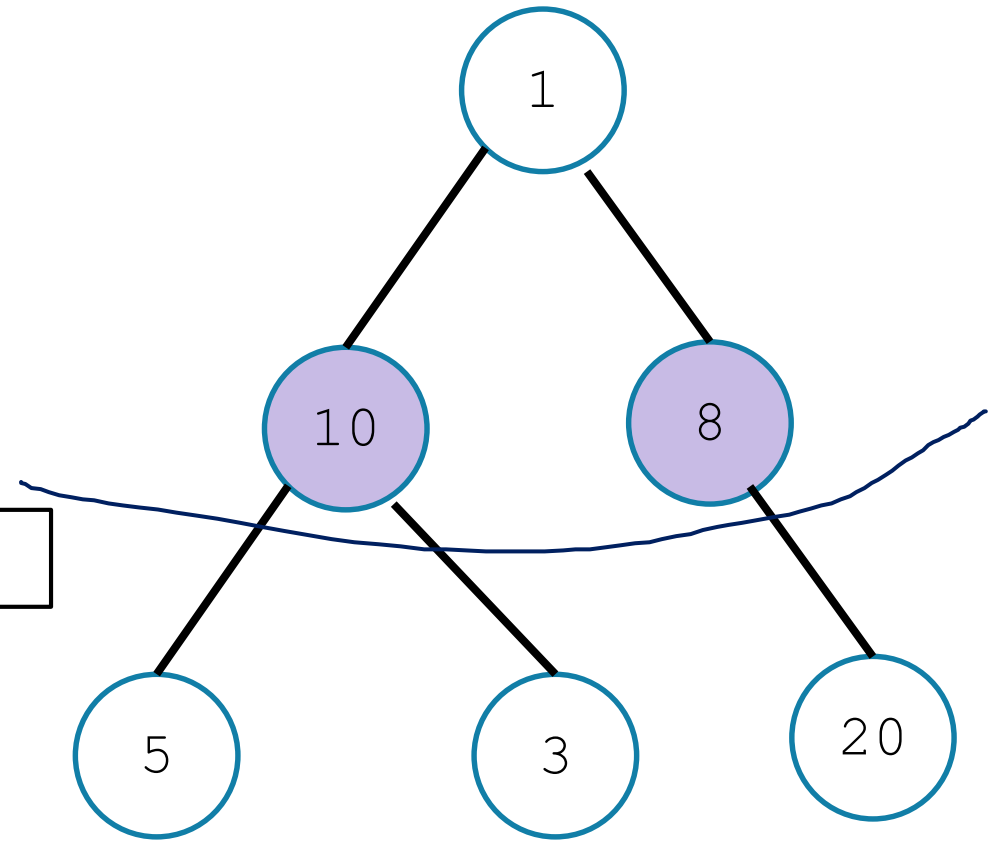
Vertex Cover

A set S of vertices is a vertex cover if for every edge (u, v) : u is in S , or v is in S , (or both)

Find the minimum vertex cover in a tree.

Give every **vertex** a weight, find the minimum weight vertex cover

A better vertex cover – weight 18



Vertex Cover

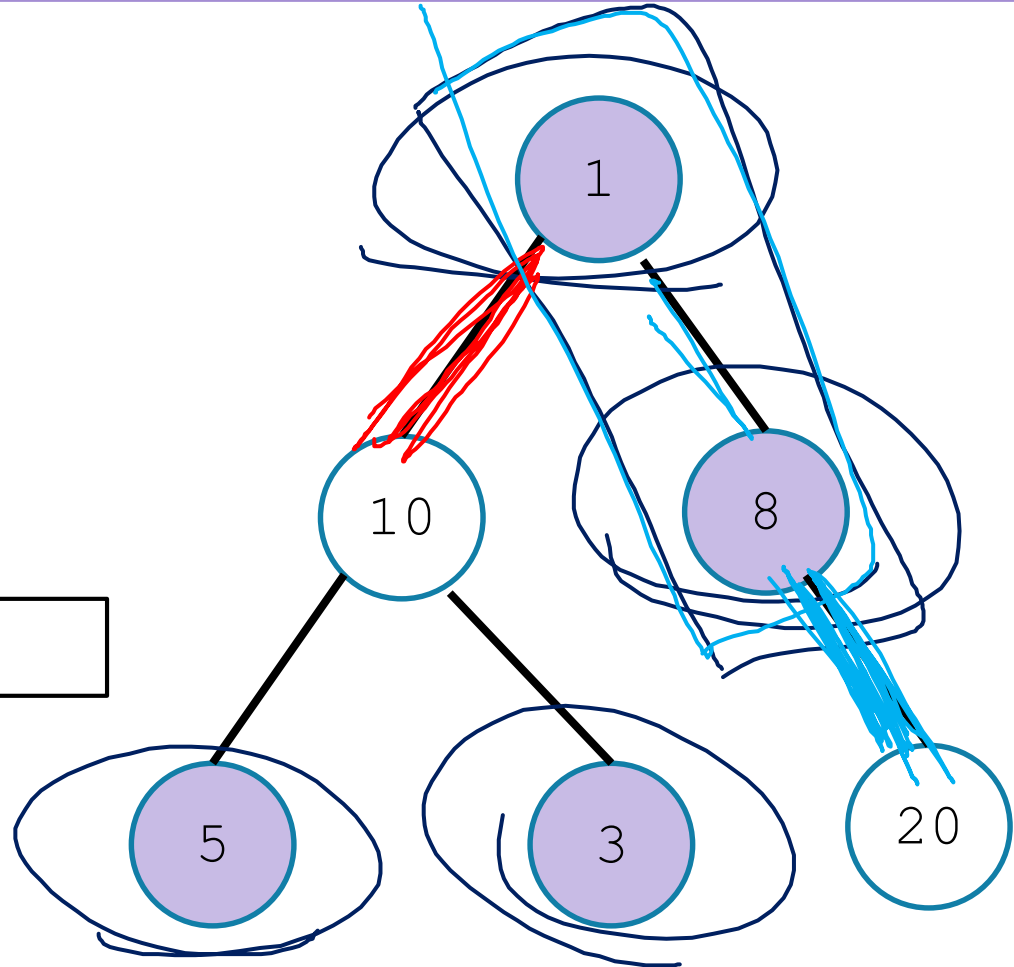
Vertex Cover

A set S of vertices is a vertex cover if for every edge (u, v) : u is in S , or v is in S , (or both)

Find the minimum vertex cover in a tree.

Give every **vertex** a weight, find the minimum weight vertex cover

The minimum vertex cover: weight 17



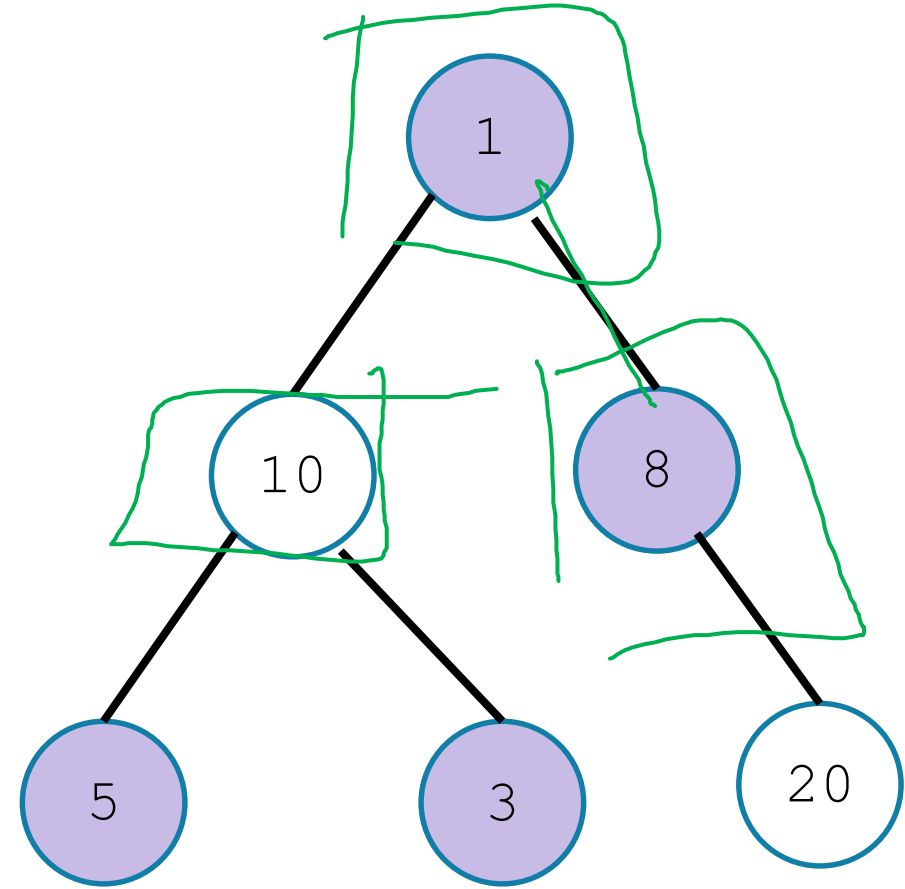
Vertex Cover

Vertex Cover

A set S of vertices is a vertex cover if for every edge (u, v) : u is in S , or v is in S , (or both)

Notice, the minimum weight vertex cover might have both endpoints of some edges

Even though only one of 1, 8 is required on the edge between them, they are both required for other edges.



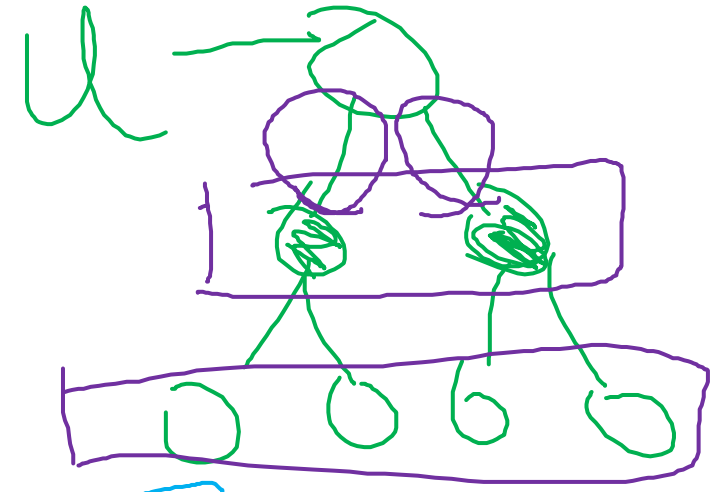
Vertex Cover – Recursively

Let's try to write a recursive algorithm first.

What information do we need to decide if we include u ?

If we don't include u then to be a valid vertex cover we need...

If we do include u then to be a valid vertex cover we need...



Vertex Cover – Recursively

Let's try to write a recursive algorithm first.

What information do we need to decide if we include u ?

If we don't include u then to be a valid vertex cover we need...

to include **all** of u 's children, and vertex covers for each subtree

If we do include u then to be a valid vertex cover we need...

just vertex covers in each subtree (whether children included or not)

Recurrence

Let $OPT(v)$ be the weight of a minimum weight vertex cover for the subtree rooted at v .

Write a recurrence for $OPT()$

Then figure out how to calculate it

Recurrence

$OPT(v)$ – the weight of the minimum weight vertex cover for the tree rooted at v (whether or not v is included).

$INCLUDE(v)$ – the weight of the minimum weight vertex cover for the tree rooted at v where v is included in the vertex cover.

$$OPT(v) = \begin{cases} \min\{\sum_{u:u \text{ is a child of } v} INCLUDE(u), \text{weight}(v) + \sum_{u:u \text{ is a child of } v} OPT(u)\} & \text{if } v \text{ is not a leaf} \\ 0 & \text{if } v \text{ is a leaf} \end{cases}$$

$$INCLUDE(v) = \text{weight}(v) + \sum_{u:u \text{ is a child of } v} OPT(u)$$

Vertex Cover Dynamic Program

What memoization structure should we use?

What code should we write?

What's the running time?

Vertex Cover Dynamic Program

What memoization structure should we use?

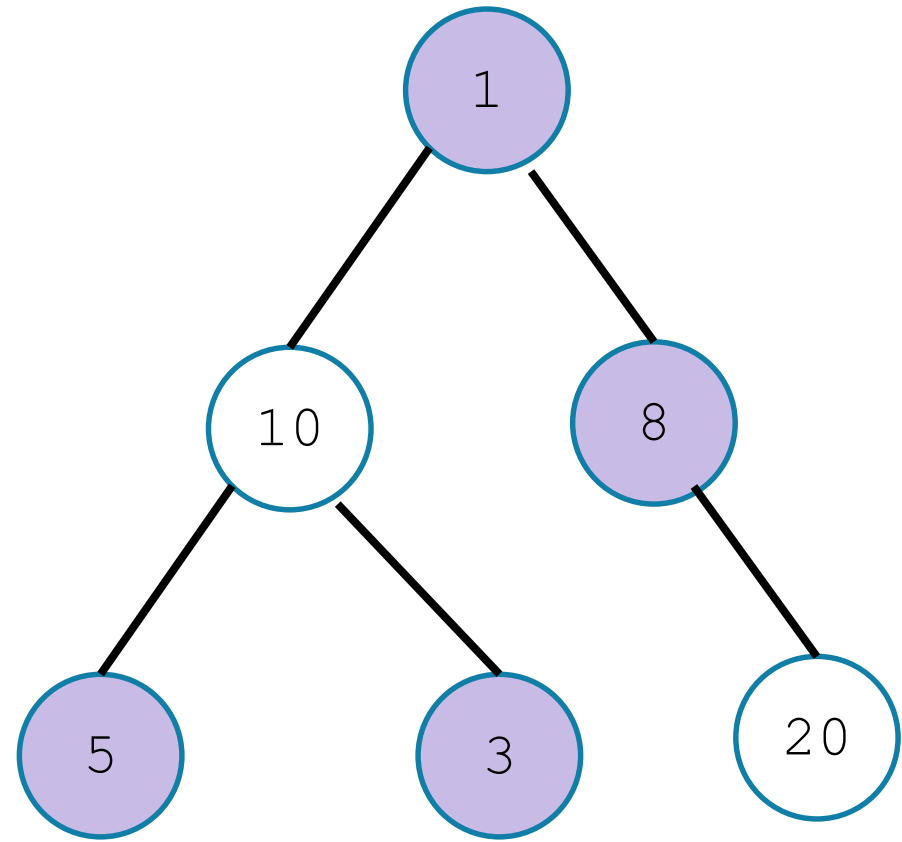
the tree itself!

What code should we write?

What's the running time?

Vertex Cover

What order do we do the calculation?



Vertex Cover Dynamic Program

What memoization structure should we use?

the tree itself!

What code should we write?

A post-order traversal (make recursive calls, then look up values in children to do calculations)

What's the running time?

$\Theta(n)$