Dynamic Programming
Hidden Parameters and on Trees

CSE 417 22AU
Lecture 15

Plan For This Week

Today

I'll walk through 2 more examples with you, and maybe a little history.

Wednesday

Practice Day; it's hard to learn just by watching, we'll have you work through a problem or two in full detail.

Friday (and a bit next Monday)

Two "advanced" DP applications; see really clever ideas (that we wouldn't expect you to find on your own, but are good to see).

Longest Increasing Subsequence

0	1	2	3	4	5	6	7
5	-6	3	6	-5	2	8	10

Longest set of (not necessarily consecutive) elements that are increasing

5 is optimal for the array above

(indices $1,2,3,6,7$; elements $-6,3,6,8,10$)

For simplicity – assume all array elements are distinct.

Longest Increasing Subsequence

What do we need to know to decide on element i?

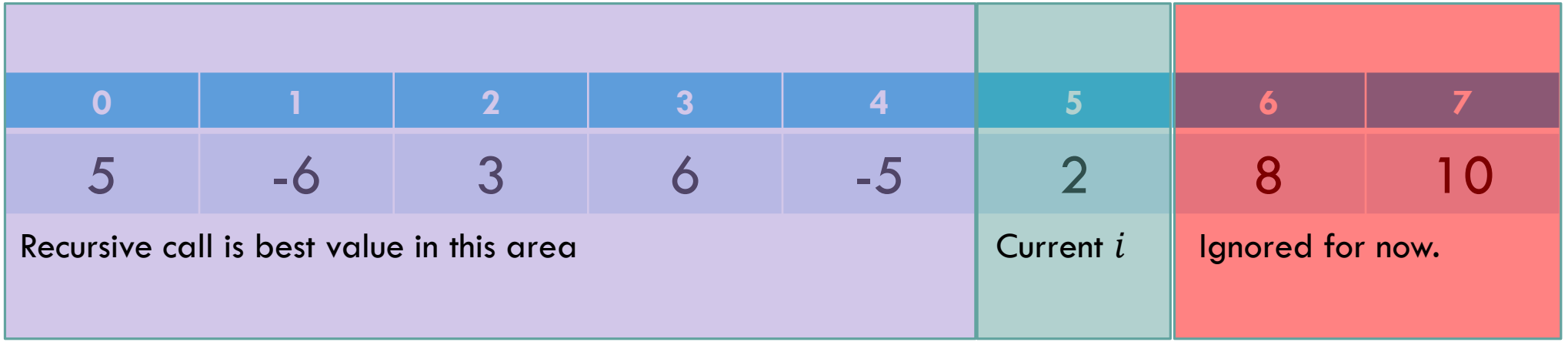
Is it allowed?

Will the sequence still be increasing if it's included?

Still thinking right to left --

Two indices: index we're looking at, and index of upper bound on elements (i.e. the value we need to decide if we're still increasing).

Recurrence



Need recursive answer to the left

Currently processing i

Recursive calls to the left are needed to know optimum from $1 \ldots i$

Will move i to the right in our iterative algorithm

Longest Increasing Subsequence

$LIS(i, j)$ is "Number of elements of the maximum increasing subsequence from $0, \ldots, i$ where every element of the sequence is at most $A[j]$"

Need a recurrence

$$LIS(i,j) = \begin{cases} ? & \text{if } i < 0 \\ ? & \text{if } i = 0 \\ ? & \text{if } A[i] > A[j] \\ ? & \text{otherwise} \end{cases}$$

Longest Increasing Subsequence

$LIS(i, j)$ is "Number of elements of the maximum increasing subsequence from $0, \ldots, i$ where every element of the sequence is at most $A[j]$"

Need a recurrence

$$LIS(i,j) = \begin{cases} ? & \text{if } i < 0 \\ ? & \text{if } i = 0 \\ ? & \text{if } A[i] > A[j] \\ ? & \text{otherwise} \end{cases}$$

If $A[i] > A[j]$ element i cannot be included in an increasing subsequence where every element is at most $A[j]$. So taking the largest among the first $i - 1$ suffices.

If $A[i] \leq A[j]$, then if we include i, we may include elements to the left only if they are less than $A[i]$ (since $A[i]$ will now be the last, and therefore largest, of elements $1 \ldots i$. If we don't include i we want the maximum increasing subsequence among $1 \ldots i - 1$.

Longest Increasing Subsequence

$LIS(i,j)$ is "Number of elements of the maximum increasing subsequence from $0, \ldots, i$ where every element of the sequence is at most $A[j]$"

Need a recurrence

$$LIS(i,j) = \begin{cases} 0 & \text{if } i < 0 \\ \mathbb{I}[A[i] \leq A[j]] & \text{if } i = 0 \\ LIS(i-1,j) & \text{if } A[i] > A[j] \\ \max\{1 + LIS(i-1,i), LIS(i-1,j)\} & \text{otherwise} \end{cases}$$

If $A[i] > A[j]$ element i cannot be included in an increasing subsequence where every element is at most $A[j]$. So taking the largest among the first $i-1$ suffices.

If $A[i] \leq A[j]$, then if we include i, we may include elements to the left only if they are less than $A[i]$ (since $A[i]$ will now be the last, and therefore largest, of elements $0 \ldots i$. If we don't include i we want the maximum increasing subsequence among $0 \ldots i-1$.

Longest Increasing Subsequence

$$LIS(i,j) = \begin{cases} 0 & \text{if } i < 0 \\ \mathbb{I}[A[i] \leq A[j]] & \text{if } i = 0 \\ LIS(i-1,j) & \text{if } A[i] > A[j] \\ \max\{1 + LIS(i-1,i), LIS(i-1,j)\} & \text{otherwise} \end{cases}$$

Memoization structure? $n \times n$ array.

Filling order?

LIS

$$LIS(i,j) = \begin{cases} 0 & \text{if } i < 0 \\ \mathbb{I}[A[i] \le A[j]] & \text{if } i = 0 \\ LIS(i-1,j) & \text{if } A[i] > A[j] \\ \max\{1 + LIS(i-1,i), LIS(i-1,j)\} & \text{otherwise} \end{cases}$$

j →

	0, 5	1, −6	2, 3	3, 6	4, −5	5, 2	6, 8	7, 10
0, 5								
1, −6								
2, 3								
3, 6								
4, −5								
5, 2								
6, 8								
7, 10								

i ↓

LIS

$LIS(1,0)$ $A[1] < A[0]$ not allowed:
Take $LIS(0,0)$

$$LIS(i,j) = \begin{cases} 0 & \text{if } i < 0 \\ \mathbb{I}[A[i] \le A[j]] & \text{if } i = 0 \\ LIS(i-1,j) & \text{if } A[i] > A[j] \\ \max\{1 + LIS(i-1,i), LIS(i-1,j)\} & \text{otherwise} \end{cases}$$

j →

i ↓	0, 5	1, −6	2, 3	3, 6	4, −5	5, 2	6, 8	7, 10
0, 5	1	0	0	1	0	0	1	1
1, −6	1							
2, 3								
3, 6								
4, −5								
5, 2								
6, 8								
7, 10								

LIS

$LIS(1,1)$ $A[1] \leq A[1]$ can add, $1 + LIS(0,1)$ or $LIS(0,1)$

$$LIS(i,j) = \begin{cases} 0 & \text{if } i < 0 \\ \mathbb{I}[A[i] \leq A[j]] & \text{if } i = 0 \\ LIS(i-1,j) & \text{if } A[i] > A[j] \\ \max\{1 + LIS(i-1,i), LIS(i-1,j)\} & \text{otherwise} \end{cases}$$

j →

	0, 5	1, −6	2, 3	3, 6	4, −5	5, 2	6, 8	7, 10
0, 5	1	0	0	1	0	0	1	1
1, −6	1	1						
2, 3								
3, 6								
4, −5								
5, 2								
6, 8								
7, 10								

i ↓

LIS

$LIS(1,2)$ $A[1] \leq A[2]$ allowed to add:
$1 + LIS(0,1)$ or $LIS(0,2)$

$$LIS(i,j) = \begin{cases} 0 & \text{if } i < 0 \\ \mathbb{I}[A[i] \leq A[j]] & \text{if } i = 0 \\ LIS(i-1,j) & \text{if } A[i] > A[j] \\ \max\{1 + LIS(i-1,i), LIS(i-1,j)\} & \text{otherwise} \end{cases}$$

j →

i ↓	0, 5	1, −6	2, 3	3, 6	4, −5	5, 2	6, 8	7, 10
0, 5	1	0	0	1	0	0	1	1
1, −6	1	1	1					
2, 3								
3, 6								
4, −5								
5, 2								
6, 8								
7, 10								

LIS

$LIS(2,0)$ $A[2] \leq A[0]$ allowed to add
$1 + LIS(1,2)$ or $LIS(1,0)$

$$LIS(i,j) = \begin{cases} 0 & \text{if } i < 0 \\ \mathbb{I}[A[i] \leq A[j]] & \text{if } i = 0 \\ LIS(i-1,j) & \text{if } A[i] > A[j] \\ \max\{1 + LIS(i-1,i), LIS(i-1,j)\} & \text{otherwise} \end{cases}$$

j →

i ↓	0, 5	1, −6	2, 3	3, 6	4, −5	5, 2	6, 8	7, 10
0, 5	1	0	0	1	0	0	1	1
1, −6	1	1	1	1	1	1	1	1
2, 3	2							
3, 6								
4, −5								
5, 2								
6, 8								
7, 10								

LIS

$$LIS(i,j) = \begin{cases} 0 & \text{if } i < 0 \\ \mathbb{I}[A[i] \le A[j]] & \text{if } i = 0 \\ LIS(i-1,j) & \text{if } A[i] > A[j] \\ \max\{1 + LIS(i-1,i), LIS(i-1,j)\} & \text{otherwise} \end{cases}$$

j →

	0, 5	1, −6	2, 3	3, 6	4, −5	5, 2	6, 8	7, 10
0, 5	1	0	0	1	0	0	1	1
1, −6	1	1	1	1	1	1	1	1
2, 3	2	1	2	2	1	1	2	2
3, 6	2	1	2	3	1	1	3	3
4, −5	2	1	2	3	2	2	3	3
5, 2	3	1	3	3	2	3	3	3
6, 8	3	1	3	3	2	3	4	4
7, 10	3	1	3	3	2	3	4	5

i ↓

pseudocode

```
//real code snippet that actually generated the table on the last slide
for(int j=0; j < n; j++){
     vals[0][j] = (A[0] <= A[j]) ? 1 : 0;
}
for(int i = 1; i < 8; i++){
     for(int j = 0; j < n; j++){
          if(A[i] > A[j])
               vals[i][j] = vals[i-1][j];
          else{
               vals[i][j] = Math.max(1+vals[i-1][i], vals[i-1][j]);
          }
     }
}
```

Longest Increasing Subsequence

$$LIS(i,j) = \begin{cases} 0 & \text{if } i < 0 \\ \mathbb{I}[A[i] \leq A[j]] & \text{if } i = 0 \\ LIS(i-1,j) & \text{if } A[i] > A[j] \\ \max\{1 + LIS(i-1,i), LIS(i-1,j)\} & \text{otherwise} \end{cases}$$

Memoization structure? $n \times n$ array.

Filling order?

Outer loop: increasing i

Inner loop: increasing j

LIS

One more thing….what's the final answer?

We want the longest increasing sequence in the whole array.

$LIS(i, j)$ is "Number of elements of the maximum increasing subsequence from $0, \ldots, i$ where every element of the sequence is at most $A[j]$"

What do we want?

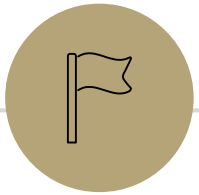
LIS

One more thing….what's the final answer?

We want the longest increasing sequence in the whole array.

$LIS(i, j)$ is "Number of elements of the maximum increasing subsequence from $0, \ldots, i$ where every element of the sequence is at most $A[j]$"

$\max_j LIS(n, j)$. Intuitively, j represents "the last element" in the array. Anything could be the last one! Take the maximum.



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.

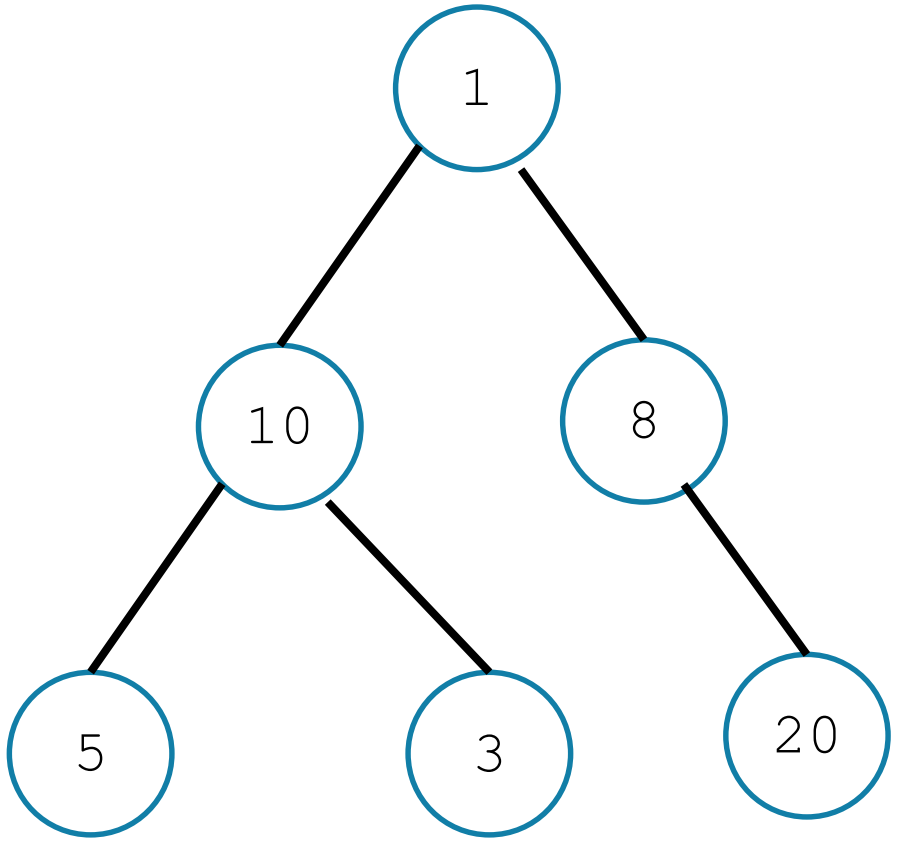
Vertex Cover

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

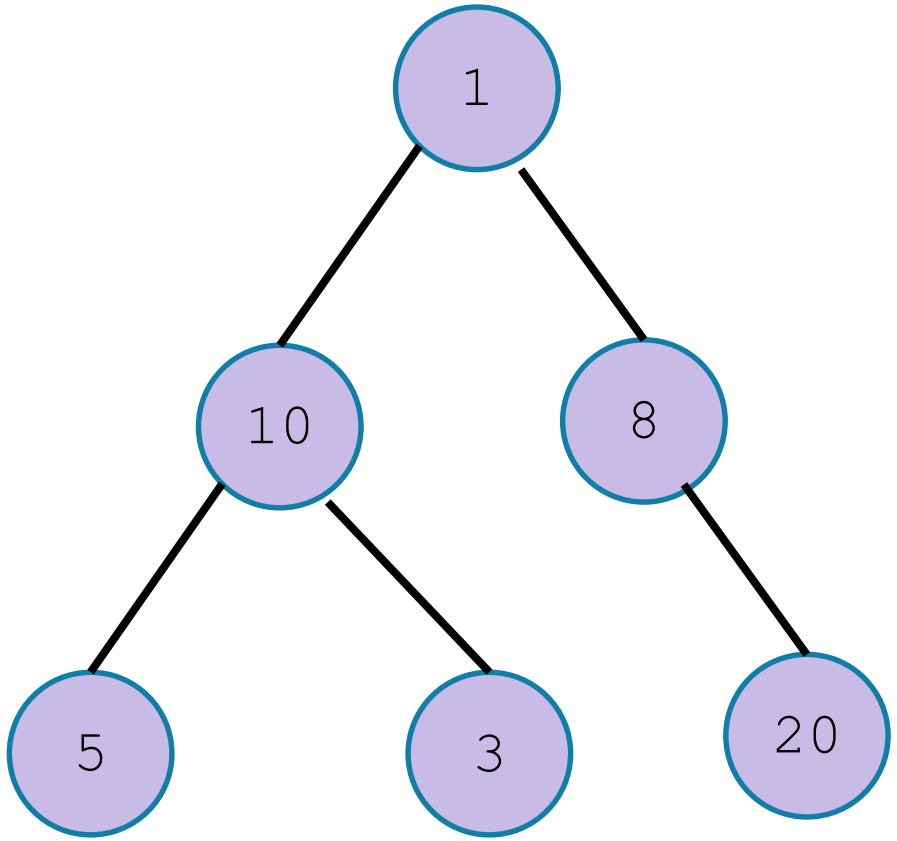
Vertex Cover

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

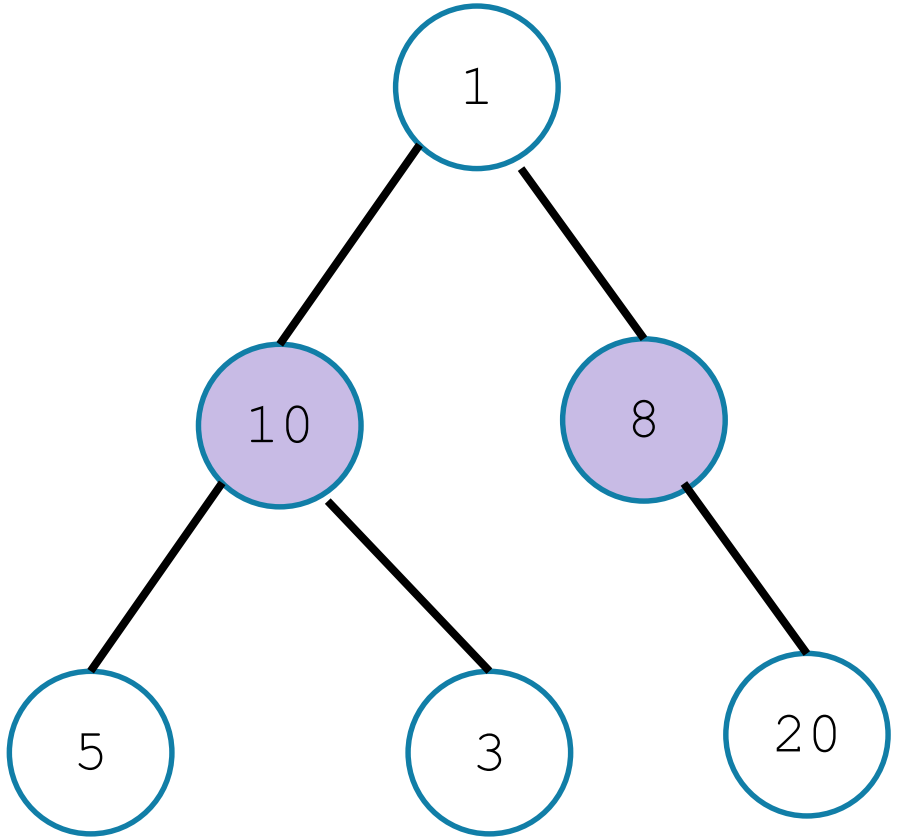
Vertex Cover

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



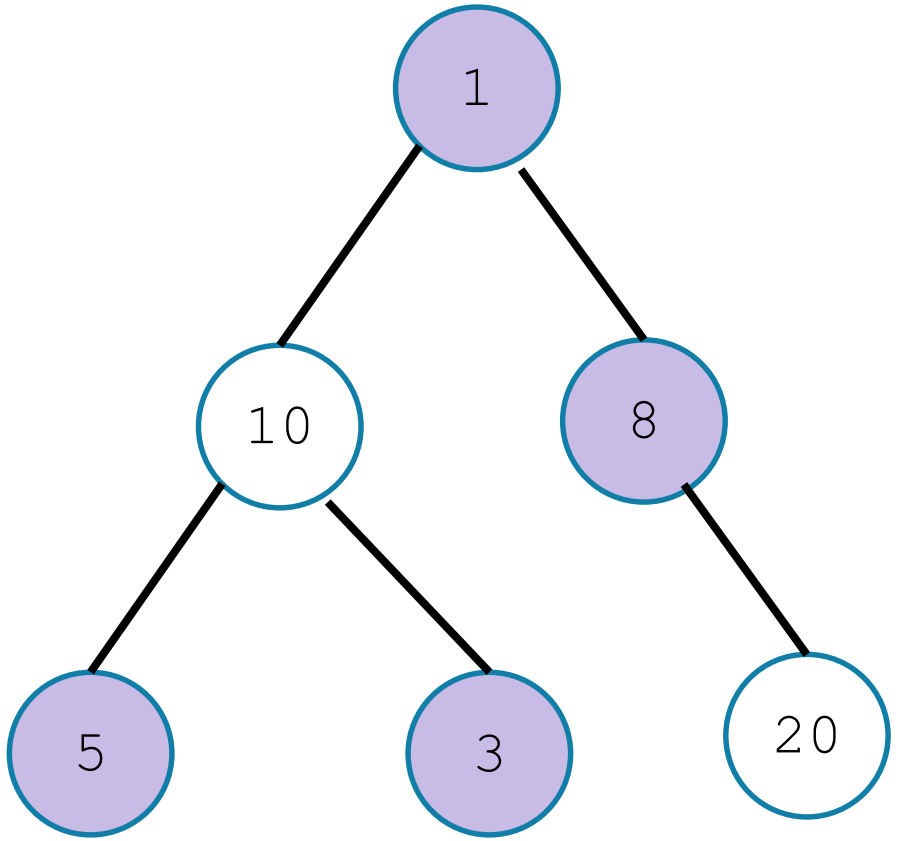
Vertex Cover

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

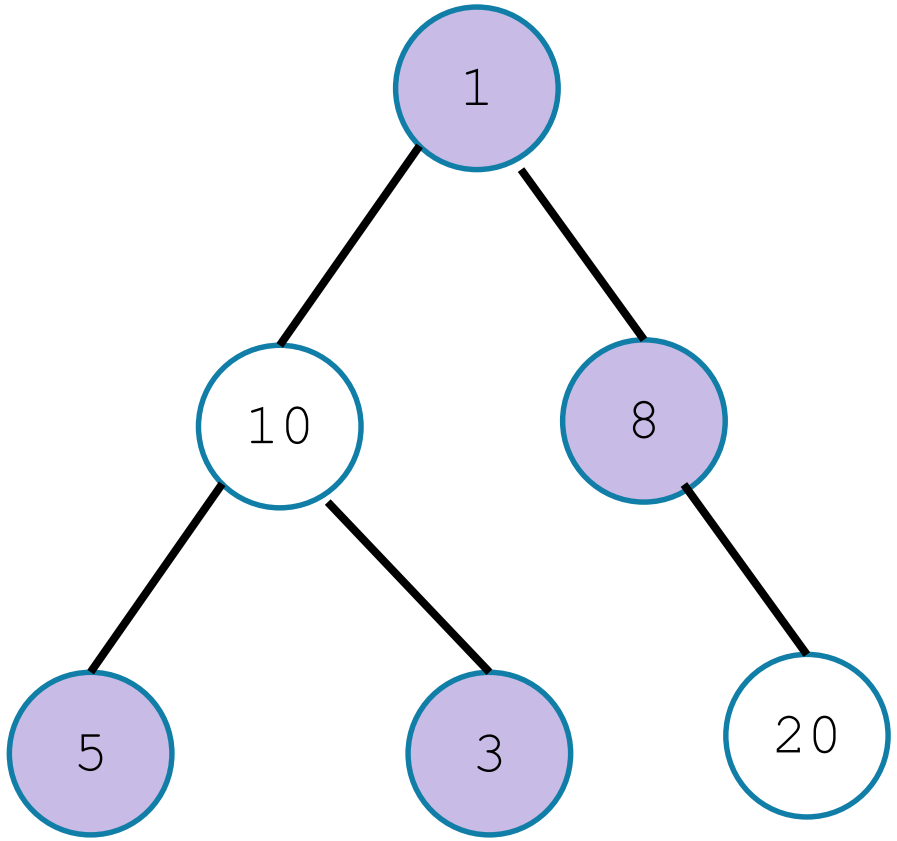
Vertex Cover

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.

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

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

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?

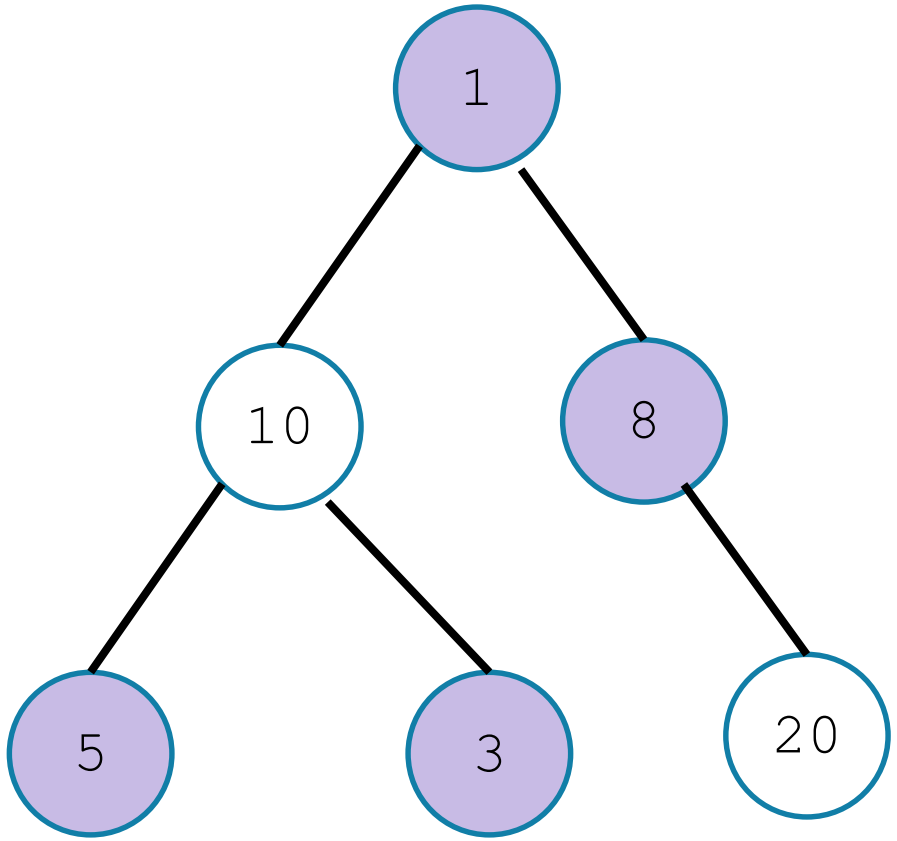
the tree itself!

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?

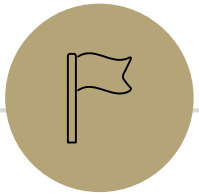
the tree itself!

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?

$\Theta(n)$



DP Context

DP Design Notes

We haven't done a single proof for DP…

We won't ask you to do one.

DP proofs are almost always just "the code does the recurrence"

But that just moves the correctness question – why is the recurrence correct?

And the proof of the recurrence being correct is almost always "I included all the cases"

I'd rather you focus on checking it than trying to explain it.

DP Design Notes

When in doubt, ask yourself "what are the possibilities for element i?"

Include or exclude in structure (subarray/subset/solution) we're building

It will be substituted, deleted, deferred to later to allow an insertion, or matched

From here, does Baby Yoda go left or down

Ask "if my recursive call gives me the *best* option for the subproblem, do I really get the best thing overall?"

Usually you'll say something like "well the only possibilities are going left or down, so if we have the best of the two, then yes!"

Make sure you're building a valid solution

Subarray is genuinely contiguous, subsequence is actually increasing.

Try on an example!!!

DP history

So…why is it called “dynamic programming?”

“programming” is an old-timey meaning of the word.

It means “scheduling”

Like a conference has a “program” of who speaks where when.
Or a television executive decides on the nightly programming (what show airs when).

DP history

So…*dynamic*?

The phrase “dynamic programming” was popularized by Richard Bellman (we’ll see one of his algorithms on Wednesday)

He was a researcher, funded by the U.S. military….

But the Secretary of Defense [as Bellman tells it] hated research. And hated math even more.

So Bellman needed a description of his research that everyone would approve of.

DP history

Dynamic

Is actually an accurate adjective – what we think is the best option (include/exclude) can change over time.

Even better

“It’s impossible to use the word ‘dynamic’ in a pejorative sense”

“It was something not even a Congressman could object to.”