More Dynamic Programming

CSE 421 Fall 22
Lecture 12

Announcements

The midterm exam is two weeks from today.

Remember it's **in the evening**. 90 minutes, 6-7:30 PM.

We'll be in GWN 301.

- 210 seats (so about 1.5x the number of students)
- Long Tables! (not folding-

Form to let us know if you can't make the main exam

- Request a conflict (work/family responsibilities or something else immovable)
- Have a COVID-related reason to not be in a big lecture hall? We'll see what we can do.

Announcements

We'll have a reference sheet for you for the midterm (including, e.g., a list of some algorithms from 332 that you've been able to reference)

You'll also be able to bring your own 8.5x11 inch piece of paper with handwritten notes.

Practice materials, topics list, contents of the provided reference, etc. all coming next week.

Baby Yoda Searching

Baby Yoda has to get from the upper-right corner to the lower left. His cradle will only let him go left and down.

He can't get past the rocks (too high) – he has to go around them (but still only going left and down)

As usual…he's hungry.

He wants to eat as many frog eggs as possible on the way.

A Recursive Function

```
FindOPT(int i,int j, bool[][] rocks, bool[][] eggs)
    if(i<0 || j < 0) return -∞
    if(rocks[i][j]) return −∞
    if(i==0 && j==0) return eggs[0][0]
    int left = FindOPT(i-1,j,rocks,eggs)
    int down = FindOPT(i,j-1,rocks,eggs)
    return Max(left,down) + eggs[i][j]
```

Recurrence Form

$$OPT(i,j) = \begin{cases} -\infty & \text{if } rocks(i,j) \text{ is true} \\ -\infty & \text{if } i < 0 \text{ or } j < 0 \\ eggs(0,0) & \text{if } i = 0 \text{ and } j = 0 \\ \max\{OPT(i-1,j), OPT(i,j-1)\} + eggs(i,j) & \text{otherwise} \end{cases}$$

Recurrences can also be used for outputs of a recursive function (not just their running times!)

This definition is a little more compact than code.

And you could write a recursive function for a recurrence like this.

Speedup

How do we go faster? Don't recalculate! **memoize**

Once you know $OPT(i, j)$ put it in an array `OPT[i][j]`

Have some initial value (null?) to mark as uninitialized

If initialized, return that.

Otherwise do the algorithm from the last slide.

How fast? Now $\Theta(rc)$.

Why? There are that many spots, each is calculated at most once and looked up at most twice.

Baby Yoda Searching

(c-1, r-1)

r-1	1	1	1	2	3	4	4	4
	1	1	1	2	3	4	4	4
	0	0	1	2	3	3	3	3
	0	$-\infty$	$-\infty$	$-\infty$	3	3	3	3
	0	1	$-\infty$	2	2	2	2	2
0	0	1	2	2	2	2	2	2

Y-coordinate

X-coordinate

0 c-1

Where's the final answer?

In the top right. Where Baby Yoda starts.

Going Bottom-up

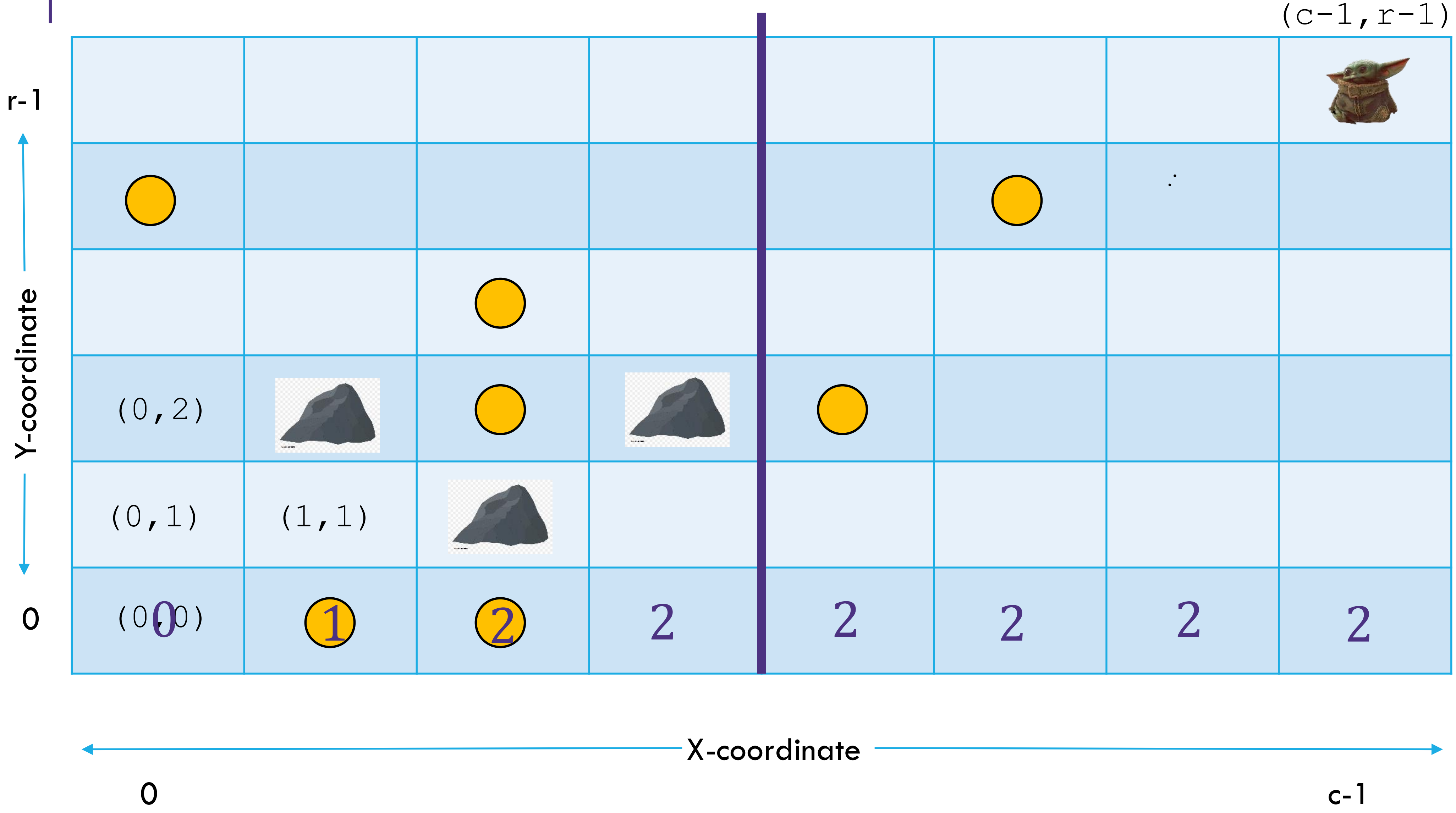
So how does that recursion work?

What's the first entry of the table that we fill?

`OPT[0][0]`

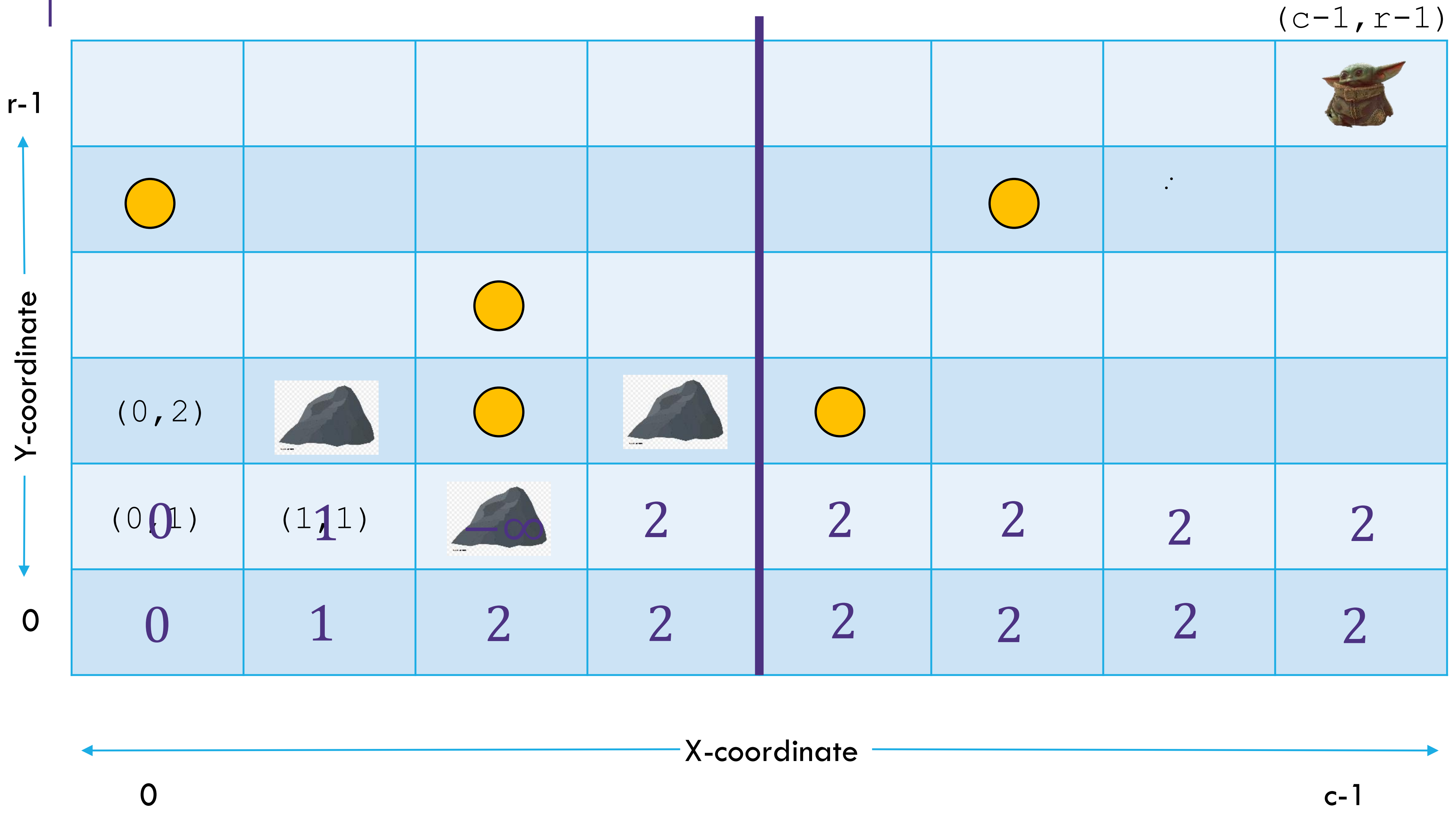
Why not just start filling in there?

Baby Yoda Searching



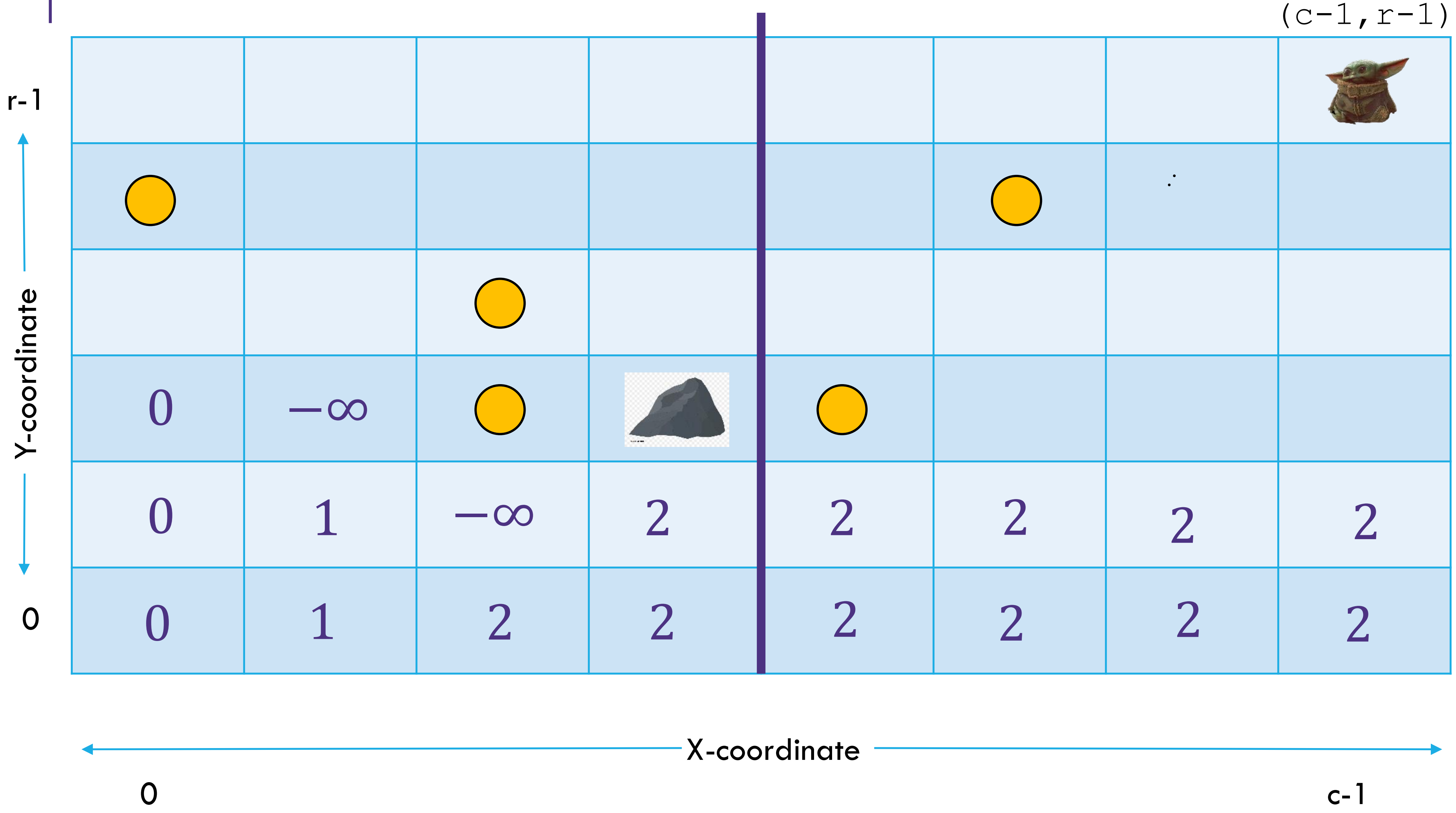
What else can we fill in?

Baby Yoda Searching



What else can we fill in?

Baby Yoda Searching



What else can we fill in?

Baby Yoda Searching

(c-1, r-1)

Y-coordinate: r-1 (top) to 0 (bottom)

1	1	1	2	3	4	4	4
1	1	1	2	3	4	4	4
0	0	1	2	3	3	3	3
0	$-\infty$	$-\infty$	$-\infty$	3	3	3	3
0	1	$-\infty$	2	2	2	2	2
0	1	2	2	2	2	2	2

X-coordinate: 0 (left) to c-1 (right)

Where's the final answer?

In the top right. Where Baby Yoda starts.

What order?

Fill in a row at a time (left to right)

Going up to the next row once a level is done.

In actual code, probably easier to handle edges first

Avoid the index-out-of-bound exceptions.

Pseudocode

```
int eggsSoFar=0;
Boolean rocksInWay=false
for(int x=0; x<c; x++)
   if(rocks[x][0]) rocksInWay = true
   eggsSoFar+=eggs[x][0]
   OPT[x][0]= rocksInWay ? −∞ : eggsSoFar

eggsSoFar=0
rocksInWay=false
for(int y=0; y<r; y++)
   eggsSoFar+=eggs[0][y]
   OPT[0][y]= rocksInWay ? −∞ : eggsSoFar

for(int y=0;y<r;y++)
  for(int x=0;x<c;x++)
    if(rocks[x][y])
        OPT[x][y]=-∞
    else
        OPT[x][y]=max(OPT[x-1][y], OPT[x][y-1])+eggs[x][y]
```

Updating the Problem

A new twist on the problem.

Baby Yoda can use the force to knock over rocks.

But he can only do it once (he tires out)

How do you decide which rocks to knock over?

Could run the algorithm once for every set of rocks knocked over.

k rocks -- $\Theta(krc)$. Can we do better?

Updating the Problem

$OPT(i, j, f)$ is the maximum amount of eggs Baby Yoda can collect on a legal path from (i, j) to $(0,0)$ using the force f times to knock over rocks.

For simplicity, assume there are no rocks at the starting location (r-1,c-1)

Here was the old rule without the force – how do we update?

$$OPT(i,j) = \begin{cases} -\infty & \text{if } rocks(i,j) \text{ is true} \\ -\infty & \text{if } i < 0 \text{ or } j < 0 \\ eggs(0,0) & \text{if } i = 0 \text{ and } j = 0 \\ \max\{OPT(i-1,j), OPT(i,j-1)\} + eggs(i,j) & \text{otherwise} \end{cases}$$

Updating the Problem

$OPT(i,j,f)$ is the maximum amount of eggs Baby Yoda can collect on a legal path from (i,j) to $(0,0)$ using the force f times to knock over rocks.

For simplicity, assume there are no rocks at the starting location (r-1,c-1)

Here was the old rule without the force – how do we update?

$$OPT(i,j,f) = \begin{cases} -\infty & \text{if } i < 0 \text{ or } j < 0 \text{ or } f < 0 \\ eggs(0,0) & \text{if } i = 0 \text{ and } j = 0 \text{ and } f \geq 0 \\ \max\{OPT(i-1,j,f - rocks(i-1,j)), OPT(i,j-1,f - rocks(i,j-1))\} + eggs(i,j) & \text{otherwise} \end{cases}$$

Casting Boolean as an integer
(subtract 1 if you would need to
knock over rocks)

Updating the Problem

$OPT(i,j,f)$ is the maximum amount of eggs Baby Yoda can collect on a legal path from (i,j) to $(0,0)$ using the force f times to knock over rocks.

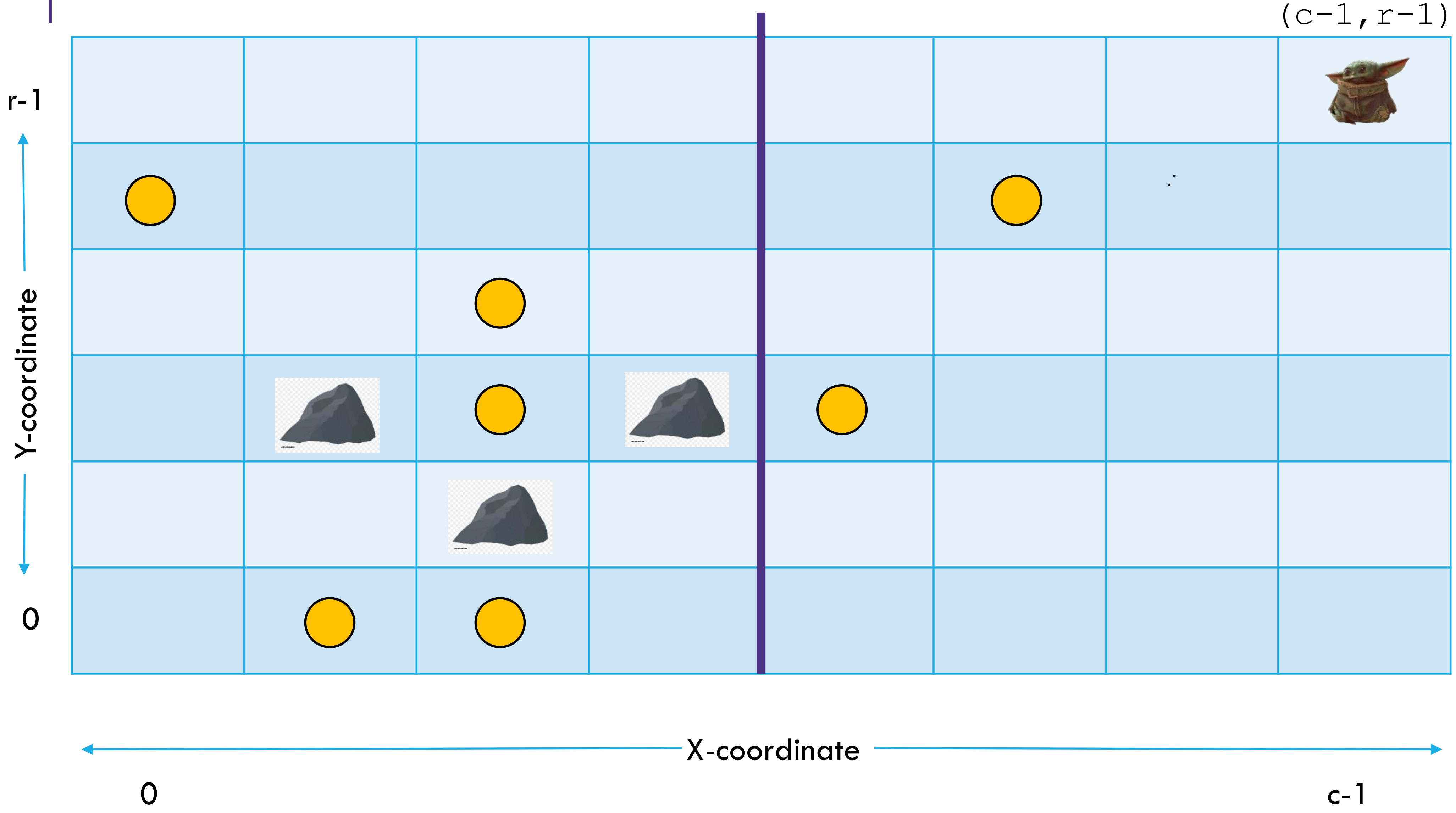
For simplicity, assume there are no rocks at the starting location (r-1,c-1)

Here was the old rule without the force – how do we update?

$$OPT(i,j,f) = \begin{cases} -\infty & \text{if } i < 0 \text{ or } j < 0 \text{ or } f < 0 \\ eggs(0,0) & \text{if } i = 0 \text{ and } j = 0 \text{ and } f \geq 0 \\ \max\{OPT(i-1,j,f-rocks(i-1,j)), OPT(i,j-1,f-rocks(i,j-1))\} + eggs(i,j) & \text{otherwise} \end{cases}$$

`rocks(i,j)` doesn't guarantee $-\infty$ anymore. Only if you were out of force uses before trying to jump onto that location.

Baby Yoda Searching



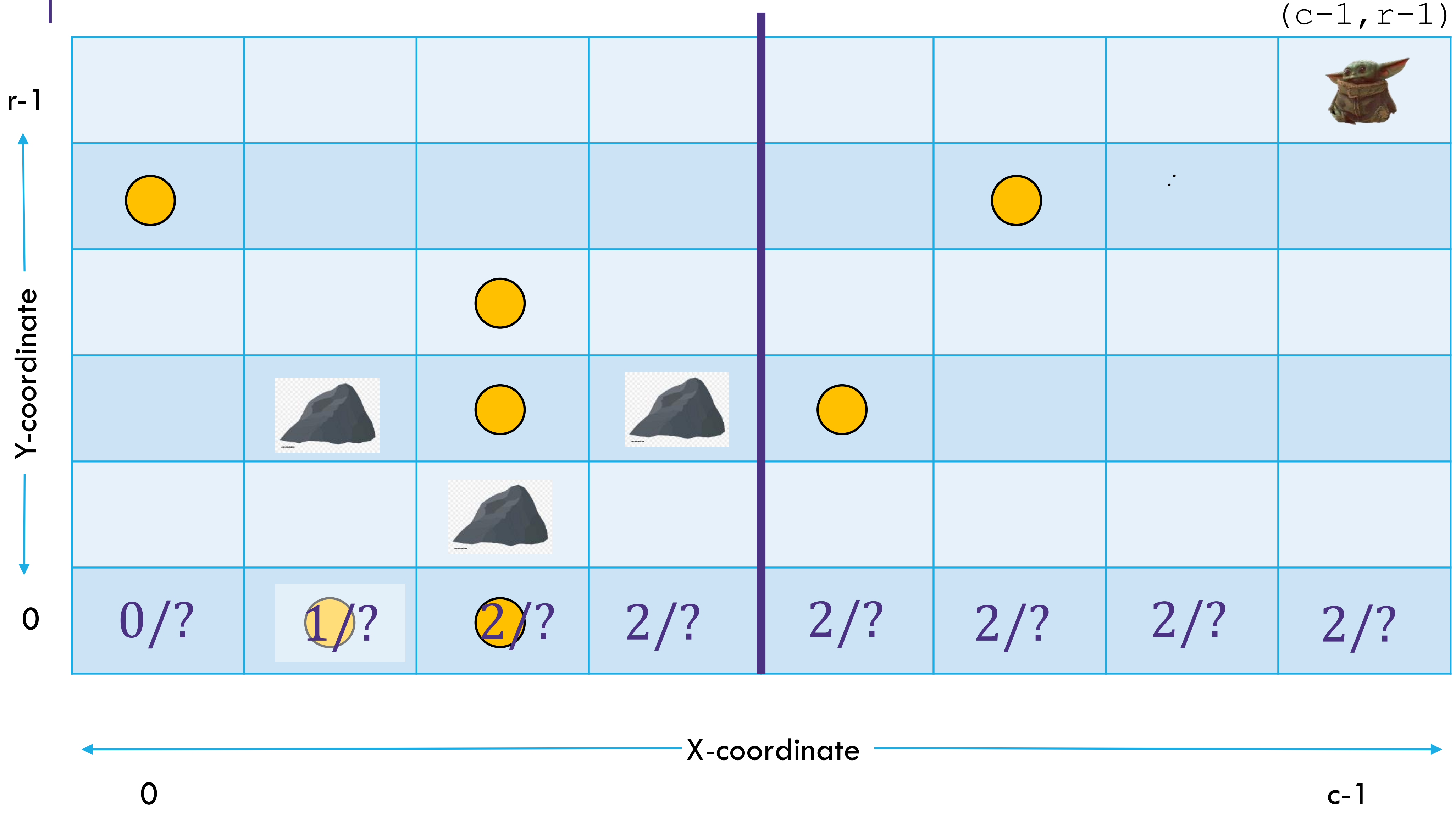
What can we fill in?

a/b

a is for (x,y,0)

b is for (x,y,1)

Baby Yoda Searching



What can we fill in?

a/b

a is for (x,y,0)

b is for (x,y,1)

Baby Yoda Searching

(c-1, r-1)

r-1	$1/?$	$1/?$	$1/?$	$1/?$	$3/?$	$4/?$	$4/?$	$4/?$
	$1/?$	$1/?$	$1/?$	$1/?$	$3/?$	$4/?$	$4/?$	$4/?$
	$0/?$	$0/?$	$1/?$	$1/?$	$3/?$	$3/?$	$3/?$	$3/?$
	$0/?$	$1/?$	$-\infty/?$	$2/?$	$3/?$	$3/?$	$3/?$	$3/?$
	$0/?$	$1/?$	$2/?$	$2/?$	$2/?$	$2/?$	$2/?$	$2/?$
0	$0/?$	$1/?$	$2/?$	$2/?$	$2/?$	$2/?$	$2/?$	$2/?$

Y-coordinate

X-coordinate

0 c-1

What can we fill in? Everything with $f = 0$ in the same order as before.

Entries are slightly different – we're handling rocks differently.

Baby Yoda Searching

(c-1, r-1)

Y-coordinate								
r-1	1/?	1/?	1/?	1/?	3/?	4/?	4/?	4/?
	1/?	1/?	1/?	1/?	3/?	4/?	4/?	4/?
	0/?	0/?	1/?	1/?	3/?	3/?	3/?	3/?
	0/?	1/?	$-\infty$/?	2/?	3/?	3/?	3/?	3/?
	0/?	1/?	2/?	2/?	2/?	2/?	2/?	2/?
0	0/?	1/?	2/?	2/?	2/?	2/?	2/?	2/?

X-coordinate

0 … c-1

What can we fill in?

Again from left to right, bottom to top, now filling in

Baby Yoda Searching

(c-1, r-1)

r-1	1/1	1/1	1/4	1/4	3/4	4/5	4/5	4/5
	1/1	1/1	1/4	1/4	3/4	4/5	4/5	4/5
	0/0	0/0	1/4	1/4	3/4	3/4	3/4	3/4
	0/0	1/1	$-\infty$/3	2/3	3/3	3/3	3/3	3/3
	0/0	1/1	2/2	2/2	2/2	2/2	2/2	2/2
0	0/0	1/1	2/2	2/2	2/2	2/2	2/2	2/2

Y-coordinate

X-coordinate

0 c-1

What can we fill in?
Again from left to right, bottom to top, now filling in

Dynamic Programming Process

1. Define the object you’re looking for

2. Write a recurrence to say how to find it

3. Design a memoization structure

4. Write an iterative algorithm

Bells,Whistles, and optimiziation

Baby Yoda Searching

(c-1, r-1)

Y-coordinate								
r-1	1/1	1/1	1/4	1/4	3/4	4/5	4/5	4/5
	1/1	1/1	1/4	1/4	3/4	4/5	4/5	4/5
	0/0	0/0	1/4	1/4	3/4	3/4	3/4	3/4
	0/0	1/1	$-\infty/3$	2/3	3/3	3/3	3/3	3/3
	0/0	1/1	2/2	2/2	2/2	2/2	2/2	2/2
0	0/0	1/1	2/2	2/2	2/2	2/2	2/2	2/2

X-coordinate

0 c-1

So should Baby Yoda go left or down?

Which Way to Go

When you're taking the `max` in the recursive case, you can also record which option gave you the max.

That's the way to go.

We'll ask you to do that once…but for the most part we'll just have you find the number.

Optimizing

Do we need all that memory?

Let’s go back to the simple version (no using the Force)

Recurrence Form

$$OPT(i,j) = \begin{cases} -\infty & \text{if } rocks(i,j) \text{ is true} \\ -\infty & \text{if } i < 0 \text{ or } j < 0 \\ eggs(0,0) & \text{if } i = 0 \text{ and } j = 0 \\ \max\{OPT(i-1,j), OPT(i,j-1)\} + eggs(i,j) & \text{otherwise} \end{cases}$$

What values do we need to keep around?

Baby Yoda Searching

(c-1, r-1)

r-1	1	1	1	2	3	4	4	4
	1	1	1	2	3	4	4	4
	0	0	1	2	3	**3**	3	3
	0	$-\infty$	$-\infty$	$-\infty$	3	3	3	3
	0	1	$-\infty$	2	2	2	2	2
0	0	1	2	2	2	2	2	2

Y-coordinate

X-coordinate

0 c-1

Need one spot left and one down.

Keep one full row, and a partially full row around. $\Theta(c)$ memory.

More Practical Problems

Edit Distance

Given two strings x and y, we'd like to tell how close they are.

Applications?

Spelling suggestions

DNA comparison

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 edit distance of the strings $x_1 x_2 \cdots x_i$ and $y_1 y_2 \cdots y_j$.
(we're indexing strings from 1, it'll make things a little prettier).

The recurrence

Poll at Pollev.com/robbie

"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.

Write a recurrence.

What do we need to keep track of? Where we are in each string!

Match right to left – be sure to keep track of characters remaining in each string!

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

"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.

Delete | Insert | Sub and matching

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

"Indicator" – math for "cast bool to int"

Dynamic Programming Process

1. Define the object you’re looking for

   Minimum Edit Distance between x and y

2. Write a recurrence to say how to find it

   ✅

3. Design a memoization structure

4. Write an iterative algorithm

Memoization

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

2D array n by m

`OPT[i][j]` is $OPT(i,j)$

Edit Distance

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

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									

Edit Distance

B	A	B	Y
T	A	S	

Delete: get rid of Y (cost 1)
BAB
TAS

$\mathrm{OPT}(i,j)$	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									

Edit Distance

B	A	B	Y
T	A	S	

Insert: insert S (cost 1)
BABY
TA

$\mathrm{OPT}(i,j)$	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									

Edit Distance

B	A	B	Y
T	A	S	

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

$\mathrm{OPT}(i, j)$	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									

Edit Distance

B	A	B	Y
T	A	S	

Gold entry will be min of:

$1 + \text{delete}$

$1 + \text{insert}$

$1 + \text{sub}$

$\text{OPT}(i,j)$	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									

Edit Distance

$1 + 2$
$1 + 3$
$1 + 2$
Min: 3

Gold entry will be min of:
$1 +$ delete
$1 +$ insert
$1 +$ sub

$OPT(i,j)$	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					
S 6										
O 7										
D 8										
A 9										

Edit Distance

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

$\mathrm{OPT}(i,j)$	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					
S 6										
O 7										
D 8										
A 9										

Edit Distance

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

$\text{OPT}(i,j)$	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

Can recover the choices, just like with Baby Yoda.

$\mathrm{OPT}(i,j)$	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

When there are ties, there may be more than one set of choices.

$\mathrm{OPT}(i,j)$	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

Dynamic Programming Process

1. Define the object you're looking for

   Minimum Edit Distance between x and y

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 rows (starting from 1)
   Inner loop: increasing column (starting from 1)