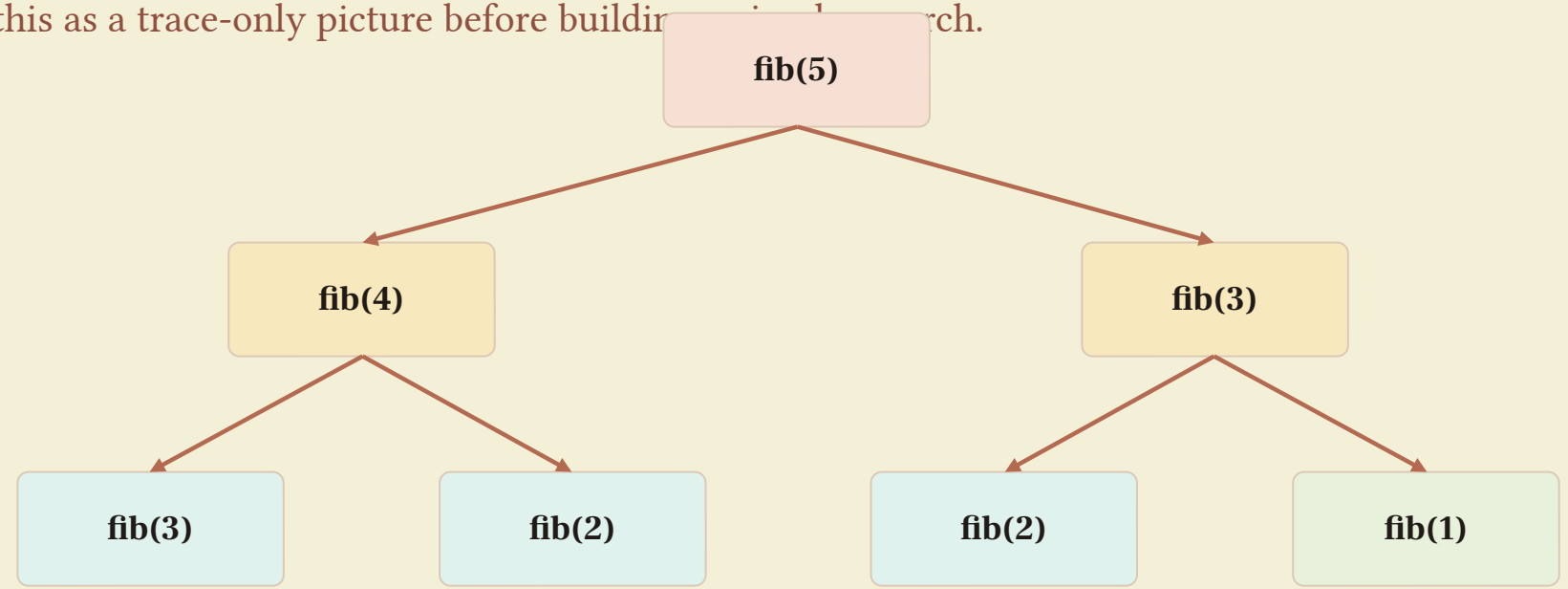
L13

AI Game Development & Computer Science Summer Program

Lecture 13: Recursion III: Branching Search

Branching recursion can make more than one recursive call

Today we use this as a trace-only picture before buildi rch.



Read the shape

One call can create two smaller calls.
Repeated smaller calls create a call tree.
Repeated inputs are a warning sign, not today's coding target.

Fibonacci is a recurrence before it is code

Read the stopping rules and combine rule first; code comes after the trace.

Case	Meaning	Return
n = 0	base case	0
n = 1	base case	1
n >= 2	ask two smaller questions	fib(n-1) + fib(n-2)

What the recurrence gives you

- Two stopping rules.
- Two smaller recursive questions.
- One combine step: add both answers.

Lab handoff

- Turn these rules into code.
- Test inputs 0 through 5.
- Trace repeated calls before optimizing.

Trace before you trust

A call tree tells you what work the program actually does.

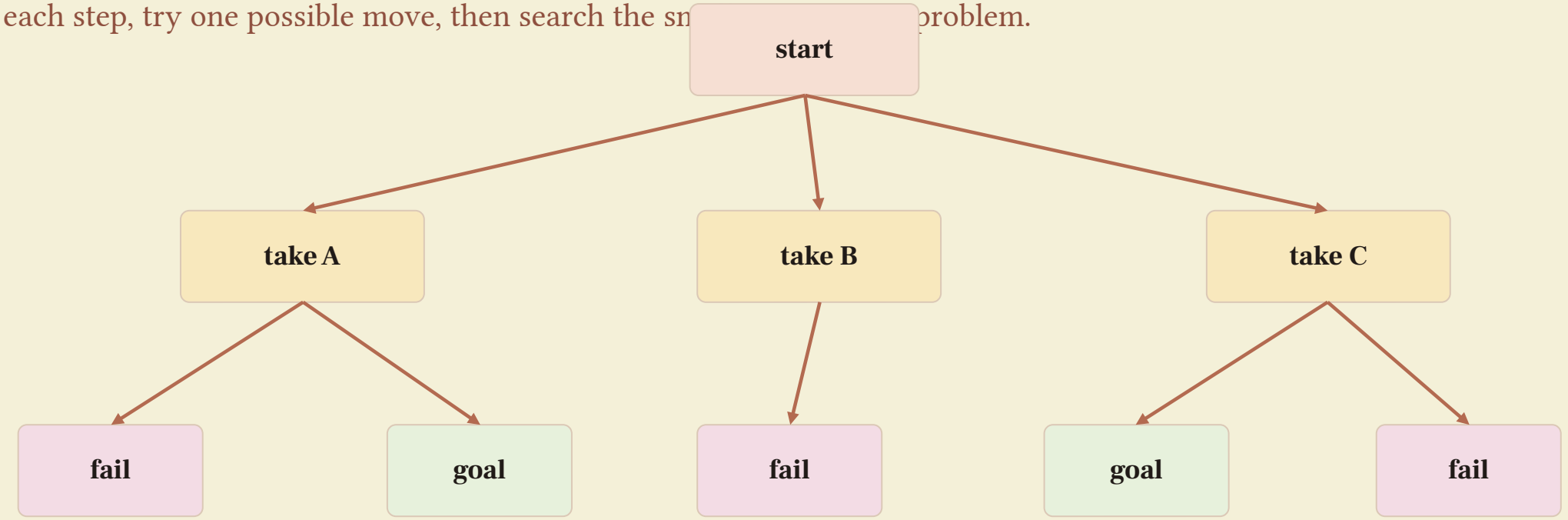
Call	Splits into	Returns
fib(4)	fib(3), fib(2)	3
fib(3)	fib(2), fib(1)	2
fib(2)	fib(1), fib(0)	1
fib(1)	base case	1
fib(0)	base case	0

Trace habit

- Write the base returns first.
- Move upward only after both child calls return.
- Look for repeated calls.

Recursive search explores a tree of choices

At each step, try one possible move, then search the sm problem.



Search rule

Stop when you find a goal.
Back up when a path fails.
Continue until every path is known.

A search function returns success or failure

```
const canReachGoal = (state: State | undefined): boolean => {
  if (!state) return false;
  if (isGoal(state)) return true;
  if (isDeadEnd(state)) return false;

  return canReachGoal(state.firstChoice)
    || canReachGoal(state.nextChoice);
};
```

Three cases

- The current state is already a goal.
- The current state cannot continue.
- A linked next choice can reach the goal.

The recursive call searches a smaller or later state, not a random new problem.

Counting stair paths has the same branching shape

Each move chooses a one-step path or a two-step path.

Question	Take 1 step	Take 2 steps
ways(4)	ways(3)	ways(2)
ways(3)	ways(2)	ways(1)
ways(2)	ways(1)	ways(0)
Base case	ways(0)	1 path
Base case	ways(<0)	0 paths

Why this is useful

- It feels concrete: every branch is a move.
- It uses the same combine step as Fibonacci.
- It gives small test cases you can check by hand.

Count change branches on a choice

```
const countChange = (amount: number, coins: readonly number[]): number => {
  if (amount === 0) return 1;       // one successful way
  if (amount < 0) return 0;         // overshot
  if (coins.length === 0) return 0; // no choices left

  const [coin, ...rest] = coins;
  return countChange(amount - coin, coins)
    + countChange(amount, rest);
};
```

Two recursive questions

- Use the current coin at least once.
- Skip the current coin and try smaller choice set.
- Add both counts because either branch can produce valid ways.

This is harder than Fibonacci because one branch changes the amount and the other changes the choice set.

Nested data recursion has two kinds of smaller work

```
type NestedNumber = number | NestedNumber[];

const flatten = (items: readonly NestedNumber[]): number[] => {
  if (items.length === 0) return [];

  const [first, ...rest] = items;
  const left = Array.isArray(first) ? flatten(first) : [first];
  return [...left, ...flatten(rest)];
};
```

Question	When true	Smaller work
empty list?	return []	none
first is list?	flatten first	sublist
first is number?	keep first	rest
always	combine	flatten(rest)

The return type drives the combine step: both branches must return number arrays that can be joined.

Branching recursion can grow fast

A clean idea may still repeat a lot of work.

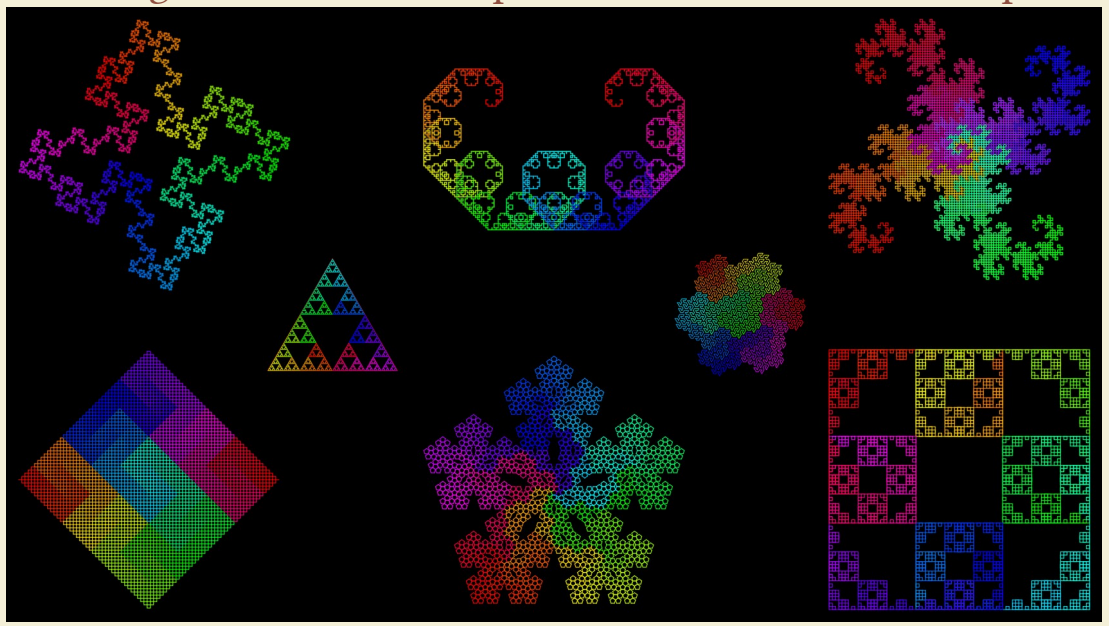
Input	Shape	What to watch
factorial	one branch	linear calls
simple tree traversal	one call per node	data size
plain fibonacci	two branches	repeated calls
choice search	many branches	possible explosion

Engineering habit

- First make the recursive meaning correct.
- Then measure or trace the amount of work.
- Only optimize after you know what repeats.

Fractals are recursive output, not decoration

A drawing rule becomes complex because the same rule repeats at smaller sizes.



Visual adapted for this course's recursion trace.

A fractal rule

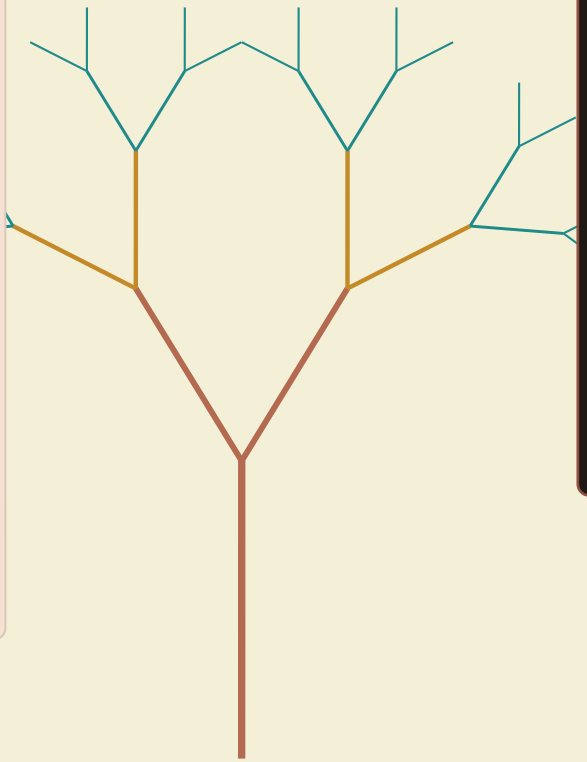
- Draw one shape.
- Move to a smaller position.
- Draw the same rule again.
- Stop after a fixed depth.

One repeated rule can draw a tree

The rule is simple; the repeated structure becomes rich.

Recursive drawing rule

- Draw a trunk.
- At the end, draw two smaller branches.
- Reduce the depth each time.
- Stop when depth reaches zero.



```
drawBranch(x, y, length, angle, depth)
  if depth is 0: stop
  draw one line
  draw left branch with depth - 1
  draw right branch with depth - 1
```

Lab block: trace one branch, then test the solver

Use examples first; use AI only after you have one manual trace.

Guided practice

- Trace ways(4) with one-step and two-step moves.
- Trace countChange(4, [1, 2]).
- Trace flatten([1, [2, 3], 4]).

Project practice

- Choose one project decision that branches.
- Draw two levels of choices.
- Ask AI for a trace table, then correct it by hand.

Lab build: count stair paths

Input	Expected paths	Reason
0	1	already at the top
1	1	one 1-step move
2	2	1+1 or 2
3	3	1+1+1, 1+2, 2+1
4	5	combine ways(3) and ways(2)

Build order

- Write the two base cases.
- Call the function on steps - 1 and steps - 2.
- Add both answers.
- Test inputs 0 through 4 before asking AI for variants.

Checkpoint

Move on when you can explain the branch shape and the stopping rules.

Exit ticket

- One branching recursive function.
- One drawn or written call tree.
- One base case for success.
- One base case for failure.
- One test created from an AI trace you checked.