Compilers Can Be Cool

What this presentation is

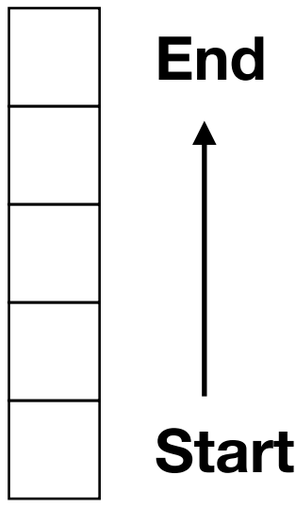
A demonstration of something I find valuable about rust
in the context of something I experienced IRL

What this presentation is <u>not</u>

- A rust tutorial
- An example of great code you should write
- A comparison of rust and ruby
  - jk, it's actually me on a soapbox saying rust is better in 100% of situations and it DOES NOT depend, Kevin
    - jkjk trollolol

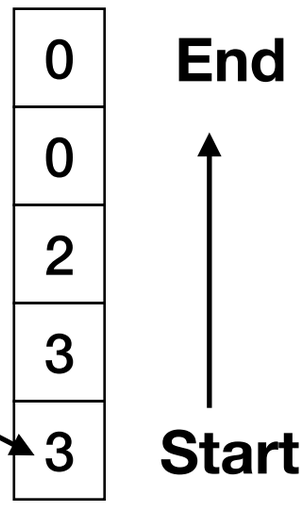
Problem

Given this walkway



And these arbitrary constraints:

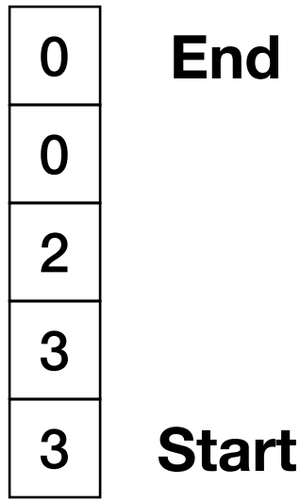
*each number indicates the biggest step you are allowed to take from that box



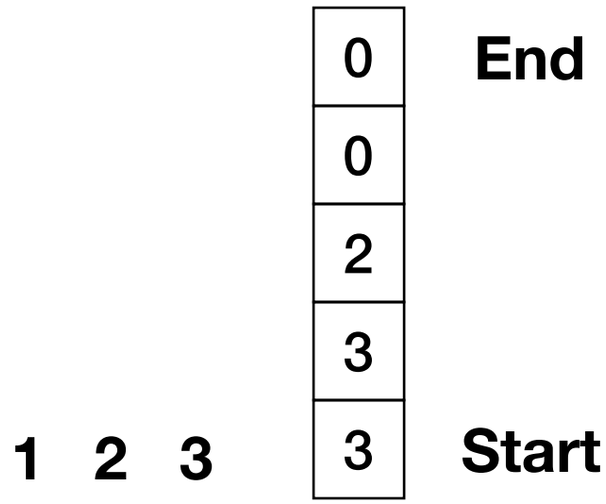
Calculate # of paths from start to end

How do we solve this?

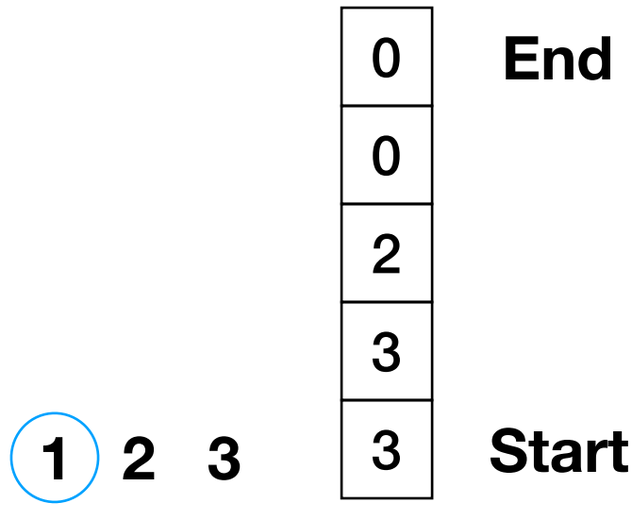
LET'S BRUTE FORCE IT, BABY!



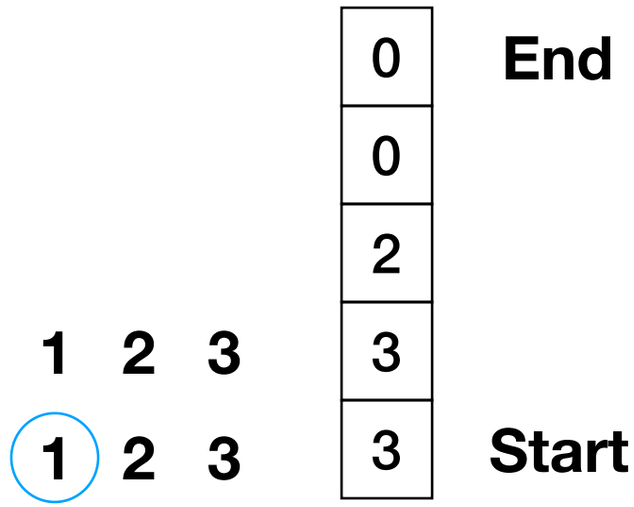
How do we solve this?



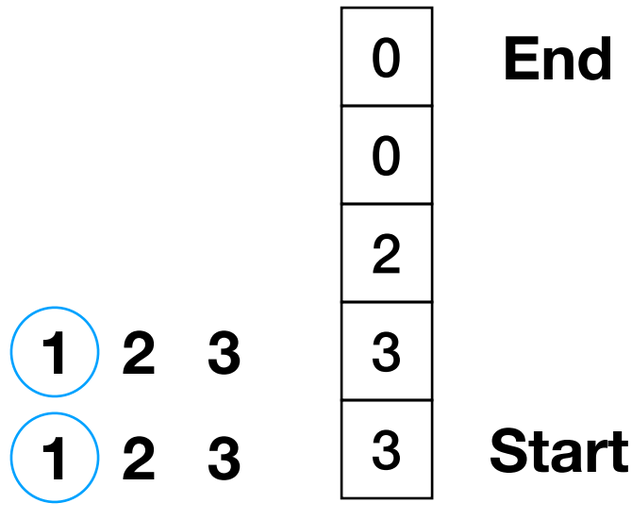
How do we solve this?



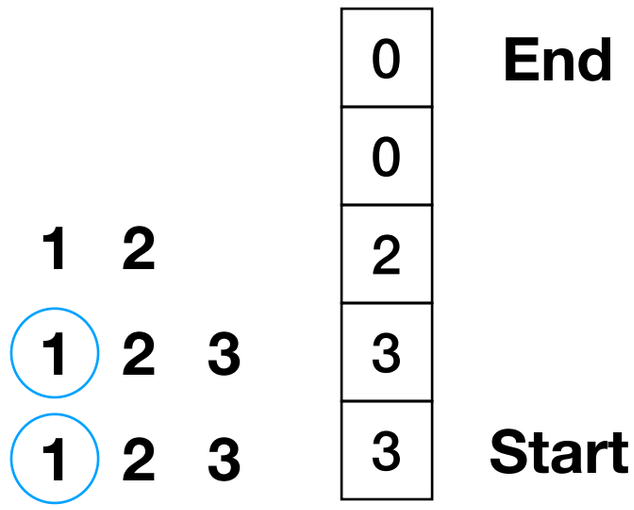
How do we solve this?



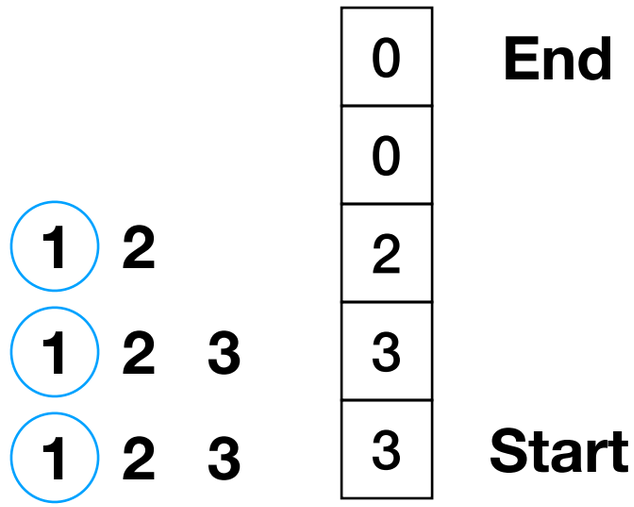
How do we solve this?



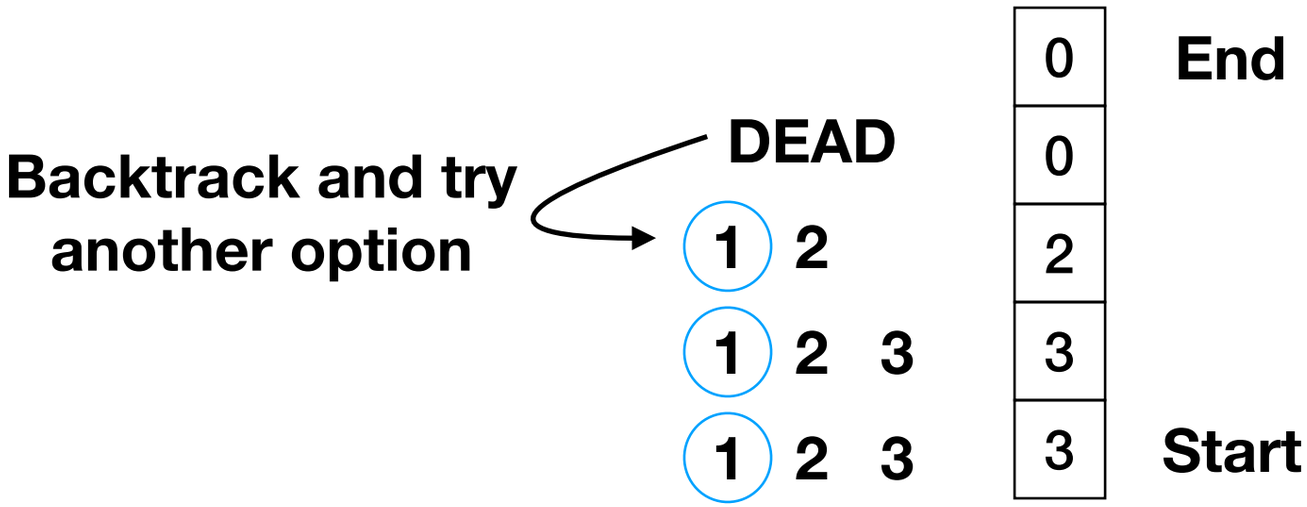
How do we solve this?



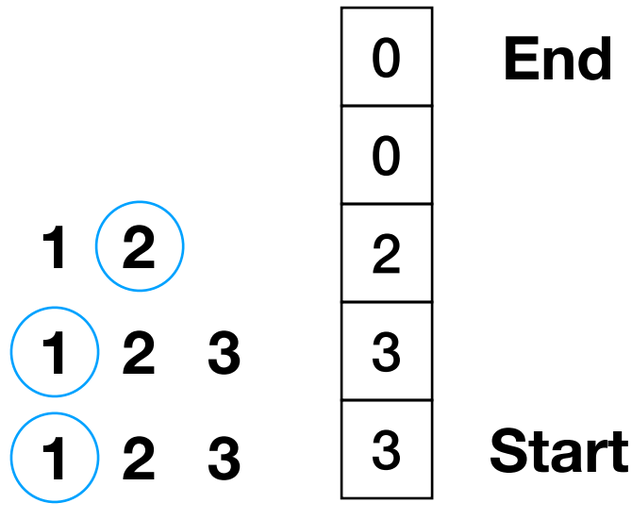
How do we solve this?



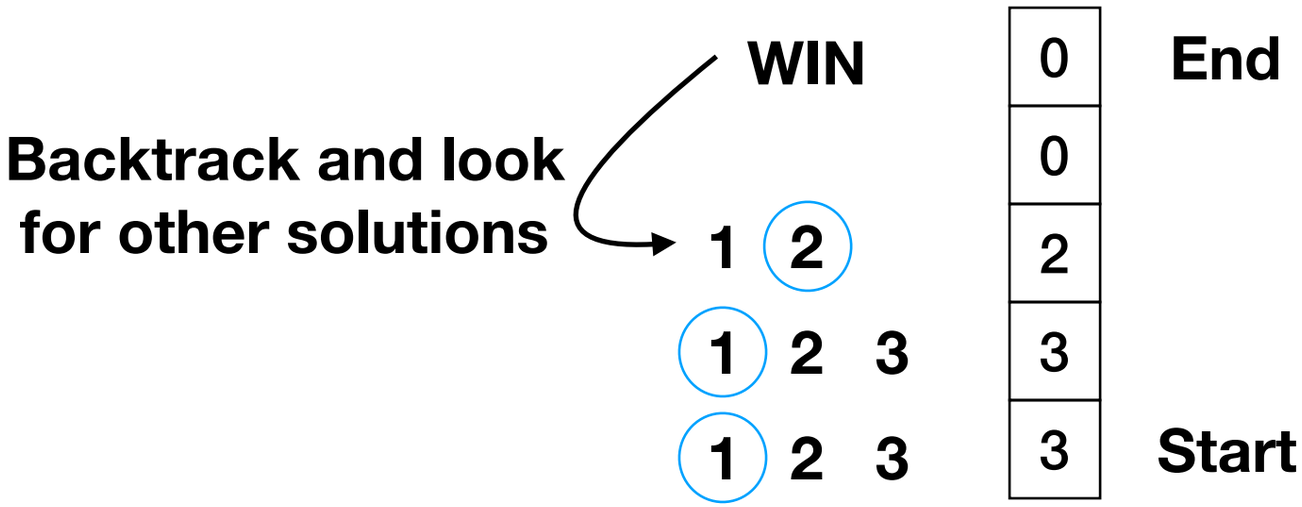
How do we solve this?



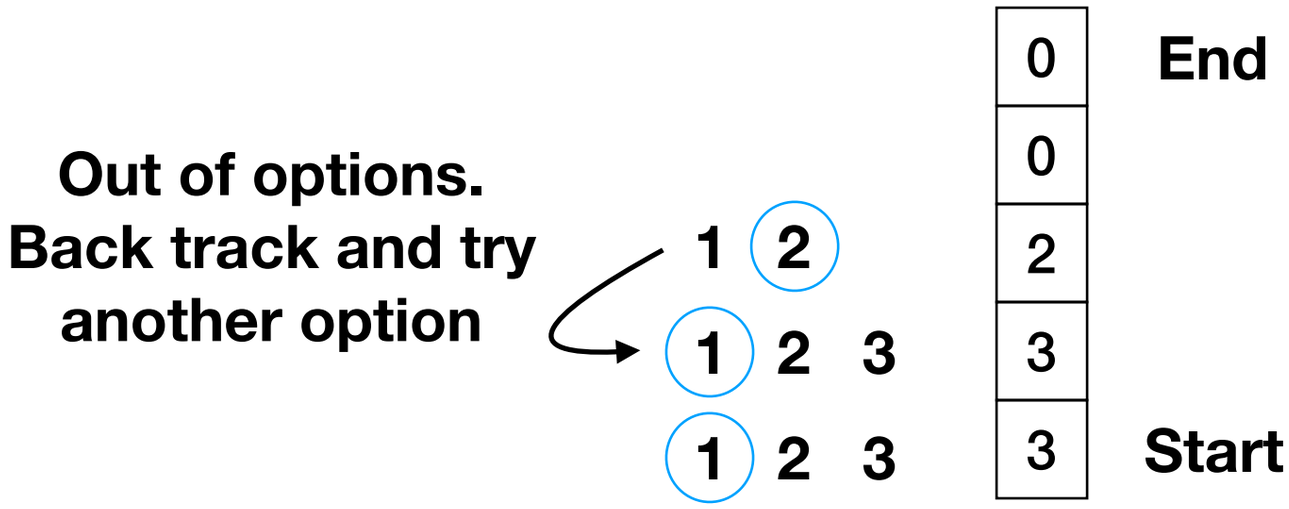
How do we solve this?



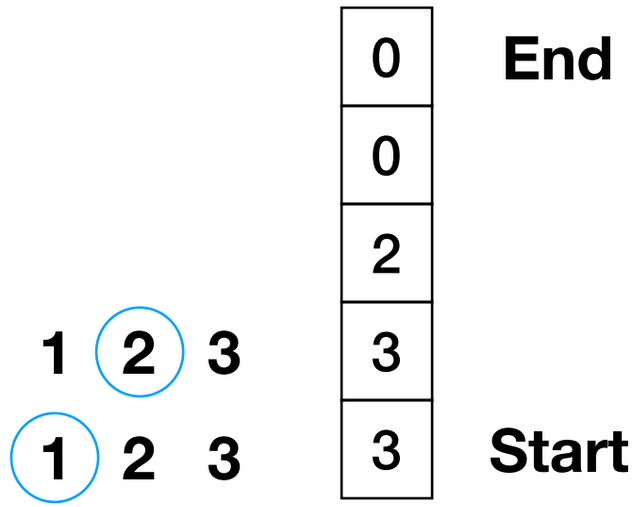
How do we solve this?



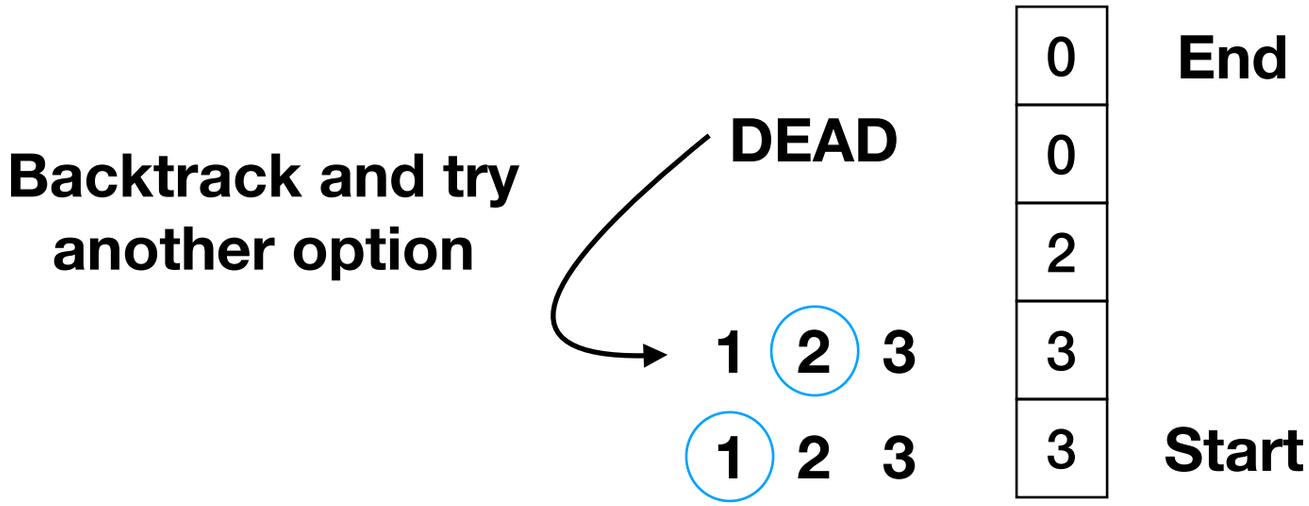
How do we solve this?



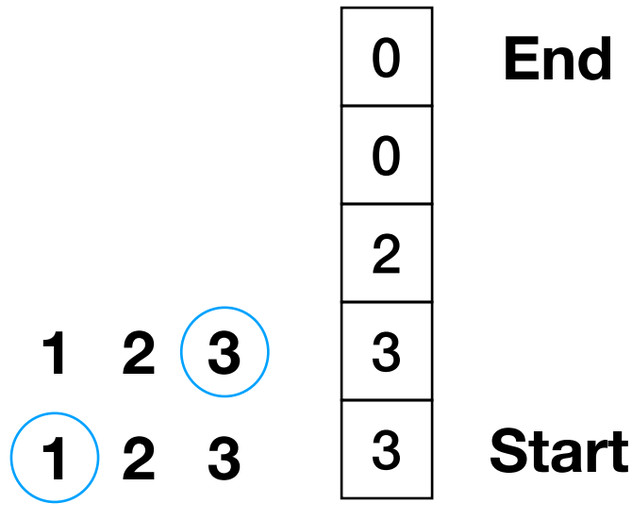
How do we solve this?



How do we solve this?

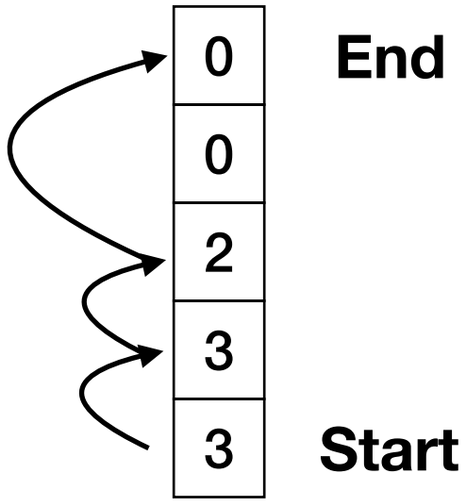


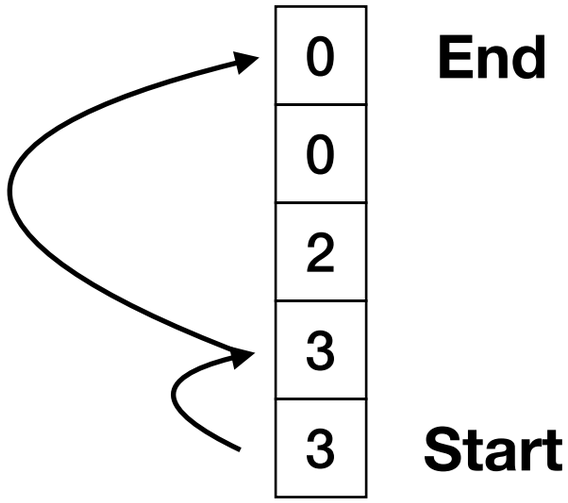
How do we solve this?

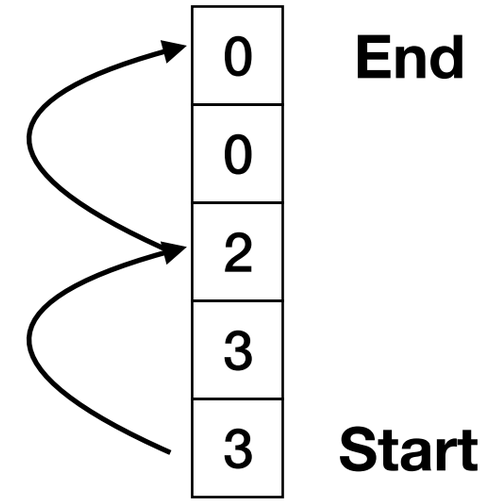


And so on until we've tried all options...

Answer = 3







How do we solve this with code?

Recursive Backtracking

Trying all options in a set and backtracking when options are exhausted or found invalid:

```
def problem_solver(x)
  if base_case?(x)
    base_case_value
  elsif invalid?(x)
    zero_value
  else
    options.each do |option|
      x.update_with!(option)
      problem_solver(x)
      x.backtrack_from!(option)
    end
  end
end
```

What's this look like?

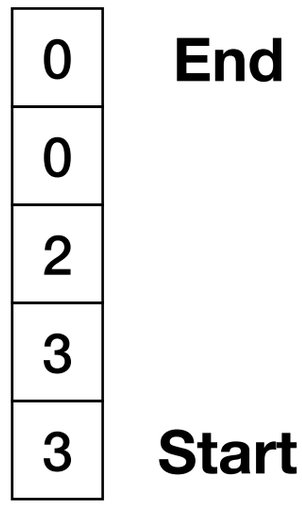
```
def problem_solver(x)
  if base_case?(x)
    base_case_value
  elsif invalid?(x)
    zero_value
  else
    options.each do |option|
      x.update_with!(option)
      problem_solver(x)
      x.backtrack_from!(option)
    end
  end
end
```

```ruby
STEPS =  [3, 3, 2, 0, 0].freeze

def num_valid_walks(position = 0)

  if position == STEPS.length - 1
    1
  elsif (
    position >= STEPS.length ||
    STEPS[position] == 0
  )
    0
  else
    count = 0

    max_step = STEPS[position]
    (1..max_step).each do |step|
      position += step
      count += num_valid_walks(position)
      position -= step
    end

    count
  end
end
```

```
~/g/s/t/rust-teaser $ ruby stc_count.rb
3
```

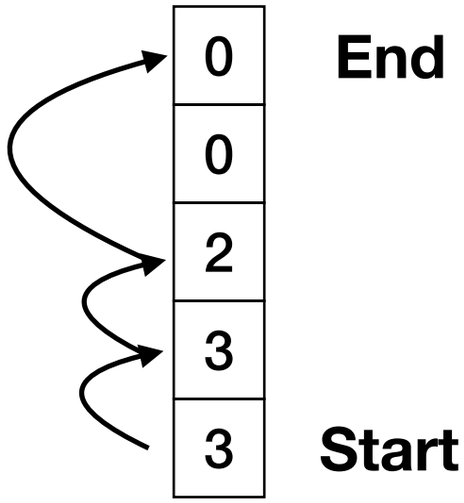
New Requirements!

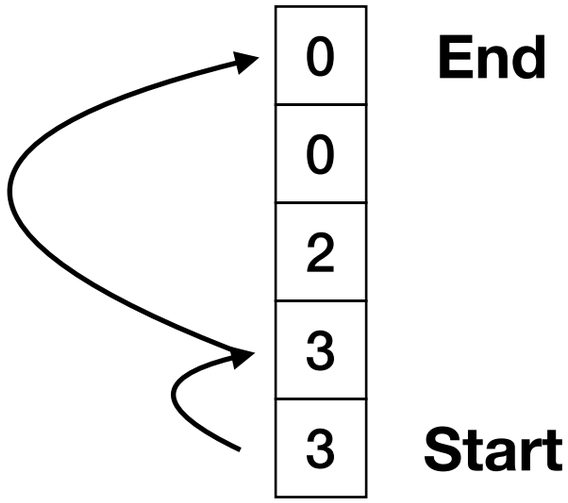
Given this walkway

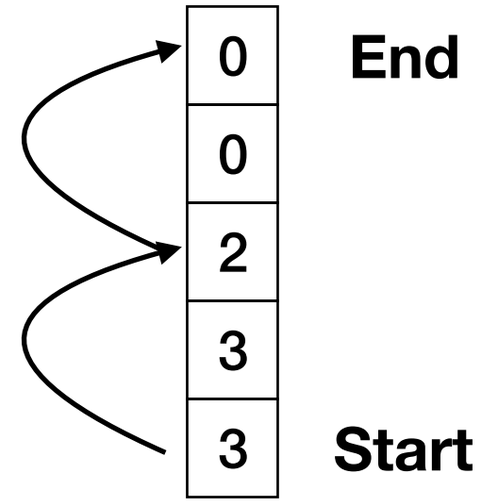


Calculate ~~# of~~ **all paths** from start to end

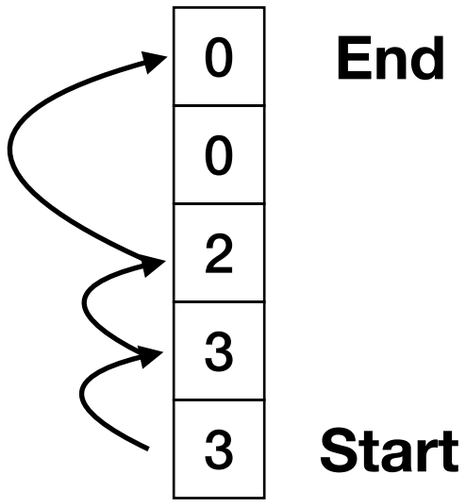
Answer

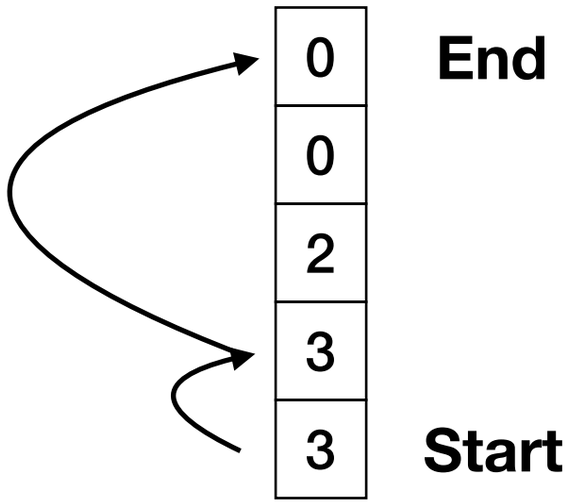


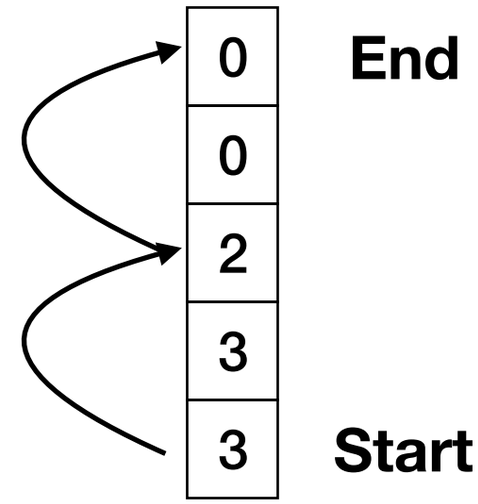




Answer







[[1, 1, 2] , [1, 3] , [2, 2]]

What's this look like?

```ruby
STEPS =  [3, 3, 2, 0, 0].freeze

def num_valid_walks(position = 0)

  if position == STEPS.length - 1
    1
  elsif (
    position >= STEPS.length ||
    STEPS[position] == 0
  )
    0
  else
    count = 0

    max_step = STEPS[position]
    (1..max_step).each do |step|
      position += step
      count += num_valid_walks(position)
      position -= step
    end

    count
  end
end
```

→

```ruby
STEPS =  [3, 3, 2, 0, 0].freeze

def valid_walks(walk = [])
  position = walk.sum
  if position == STEPS.length - 1
    [walk]
  elsif (
    position >= STEPS.length ||
    STEPS[position] == 0
  )
    []
  else
    walks = []

    max_step = STEPS[position]
    (1..max_step).each do |step|
      walk.push(step)
      walks += valid_walks(walk)
      walk.pop
    end

    walks
  end
end
```

```
~/g/s/t/rust-teaser $ ruby stc_walks.rb
[[], [], []]
```

What went wrong?

```
1 STEPS =  [3, 3, 2, 0, 0].freeze
2
3 def valid_walks(walk = [])
4   position = walk.sum
5   if position == STEPS.length - 1
6     [walk]
7   elsif (
8     position >= STEPS.length ||
9     STEPS[position] == 0
10  )
11    []
12  else
13    walks = []
14
15    max_step = STEPS[position]
16    (1..max_step).each do |step|
17      walk.push(step)
18      walks += valid_walks(walk)
19      walk.pop
20    end
21
22    walks
23  end
24 end
```

Let's try it in rust

```ruby
STEPS =  [3, 3, 2, 0, 0].freeze

def valid_walks(walk = [])
  position = walk.sum
  if position == STEPS.length - 1
    [walk]
  elsif (
    position >= STEPS.length ||
    STEPS[position] == 0
  )
    []
  else
    walks = []

    max_step = STEPS[position]
    (1..max_step).each do |step|
      walk.push(step)
      walks += valid_walks(walk)
      walk.pop
    end

    walks
  end
end
```

```rust
static STEPS: [usize; 5] = [3, 3, 2, 0, 0];
type Walk = Vec<usize>;

fn valid_walks(mut walk: Walk) -> Vec<Walk> {
    let position: usize = walk.iter().sum();
    if position == STEPS.len() - 1 {
        vec![walk]
    } else if
        position >= STEPS.len() ||
        STEPS[position] == 0
    {
        vec![]
    } else {
        let mut walks = Vec::new();

        let max_step = STEPS[position];
        for step in 1..=max_step {
            walk.push(step);
            walks.append(&mut valid_walks(walk));
            walk.pop();
        }

        walks
    }
}
```

A wild error appears!

```
~/g/s/t/r/stc_walks (master|…) $ cargo check
    Checking stc_walks v0.1.0 (/Users/marklodato/gnar/stc/todo/rust-teaser/stc_walks)
error[E0382]: borrow of moved value: `walk`
  --> src/main.rs:18:13
   |
4  | fn valid_walks(mut walk: Walk) -> Vec<Walk> {
   |                --------- move occurs because `walk` has type `std::vec::Vec<usize>`, which does not implement the `Copy` trait
...
18 |             walk.push(step);
   |             ^^^^ value borrowed here after move
19 |             walks.append(&mut valid_walks(walk));
   |                                           ---- value moved here, in previous iteration of loop

error: aborting due to previous error

For more information about this error, try `rustc --explain E0382`.
error: Could not compile `stc_walks`.
```

What can we do?

- Copy `walk` and move `walk_copy` into `valid_walks`
- Borrow references to `walk` instead of moving `walk` itself

Borrowing references

```rust
static STEPS: [usize; 5] = [3, 3, 2, 0, 0];
type Walk = Vec<usize>;

fn valid_walks(mut walk: Walk) -> Vec<Walk> {
    let position: usize = walk.iter().sum();
    if position == STEPS.len() - 1 {
        vec![walk]
    } else if
        position >= STEPS.len() ||
        STEPS[position] == 0
    {
        vec![]
    } else {
        let mut walks = Vec::new();

        let max_step = STEPS[position];
        for step in 1..=max_step {
            walk.push(step);
            walks.append(&mut valid_walks(walk));
            walk.pop();
        }

        walks
    }
}
```

```rust
static STEPS: [usize; 5] = [3, 3, 2, 0, 0];
type Walk = Vec<usize>;

fn valid_walks(walk: &mut Walk) -> Vec<&mut Walk> {
    let position: usize = walk.iter().sum();
    if position == STEPS.len() - 1 {
        vec![walk]
    } else if
        position >= STEPS.len() ||
        STEPS[position] == 0
    {
        vec![]
    } else {
        let mut walks = Vec::new();

        let max_step = STEPS[position];
        for step in 1..=max_step {
            walk.push(step);
            walks.append(&mut valid_walks(walk));
            walk.pop();
        }

        walks
    }
}
```

A(nother) wild error appears!

```
error[E0499]: cannot borrow `*walk` as mutable more than once at a time
  --> src/main.rs:20:13
   |
4  | fn valid_walks(walk: &mut Walk) -> Vec<&mut Walk> {
   |                      - let's call the lifetime of this reference `'1`
...
19 |             walks.append(&mut valid_walks(walk));
   |                                           ---- first mutable borrow occurs here
20 |             walk.pop();
   |             ^^^^ second mutable borrow occurs here
...
23 |         walks
   |         ----- returning this value requires that `*walk` is borrowed for `'1`

error: aborting due to 3 previous errors

For more information about this error, try `rustc --explain E0499`.
```

"Please note that in rust, you can either have many immutable references, or one mutable reference"

Final solution

```ruby
STEPS =  [3, 3, 2, 0, 0].freeze

def valid_walks(walk = [])
  position = walk.sum
  if position == STEPS.length - 1
    [walk.dup]
  elsif (
    position >= STEPS.length ||
    STEPS[position] == 0
  )
    []
  else
    walks = []

    max_step = STEPS[position]
    (1..max_step).each do |step|
      walk.push(step)
      walks += valid_walks(walk)
      walk.pop
    end

    walks
  end
end
```

```
~/g/s/t/rust-teaser $ ruby stc_walks.rb
[[1, 1, 2], [1, 3], [2, 2]]
```

```rust
static STEPS: [usize; 5] = [3, 3, 2, 0, 0];
type Walk = Vec<usize>;

fn valid_walks(walk: &mut Walk) -> Vec<Walk> {
    let position: usize = walk.iter().sum();
    if position == STEPS.len() - 1 {
        vec![walk.clone()]
    } else if
        position >= STEPS.len() ||
        STEPS[position] == 0
    {
        vec![]
    } else {
        let mut walks = Vec::new();

        let max_step = STEPS[position];
        for step in 1..=max_step {
            walk.push(step);
            walks.append(&mut valid_walks(walk));
            walk.pop();
        }

        walks
    }
}
```

```
~/g/s/t/r/stc_walks (master|…) $ cargo run
   Compiling stc_walks v0.1.0 (/Users/markl
    Finished dev [unoptimized + debuginfo]
     Running `target/debug/stc_walks`
[[1, 1, 2], [1, 3], [2, 2]]
```

Questions?