Array Advance Game

In this lesson, you will learn how to solve the problem of the Array Advance Game in Python.

WE'LL COVER THE FOLLOWING

In this lesson, we will be considering the “array advance game”. In this game, you are given an array of non-negative integers. For example:

```
[3,3,1,0,2,0,1]
```

Each number in the array represents the maximum you can advance in the array.

Problem

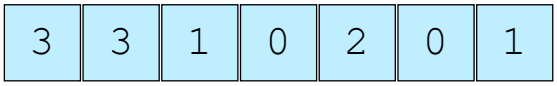
Now the problem is as follows:

Is it possible to advance from the start of the array to the last element given that the maximum you can advance from a position is based on the value of the array at the index you are currently present on?

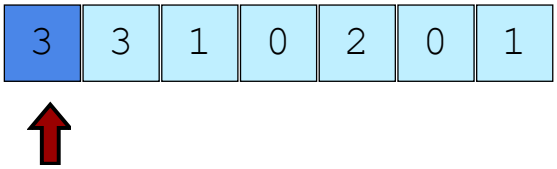
We will cover how to solve this problem algorithmically, and then code up a solution to this problem in Python.

Let’s start with a simple example. Refer to the slides below:

Array A

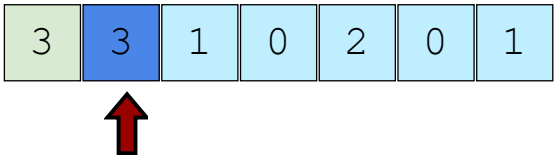


Array A



Current Position : index 0
Maximum that you can advance:
= A[0] = 3

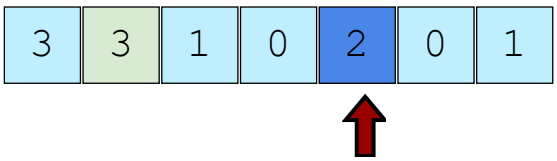
Array A



Moved 1 place forward

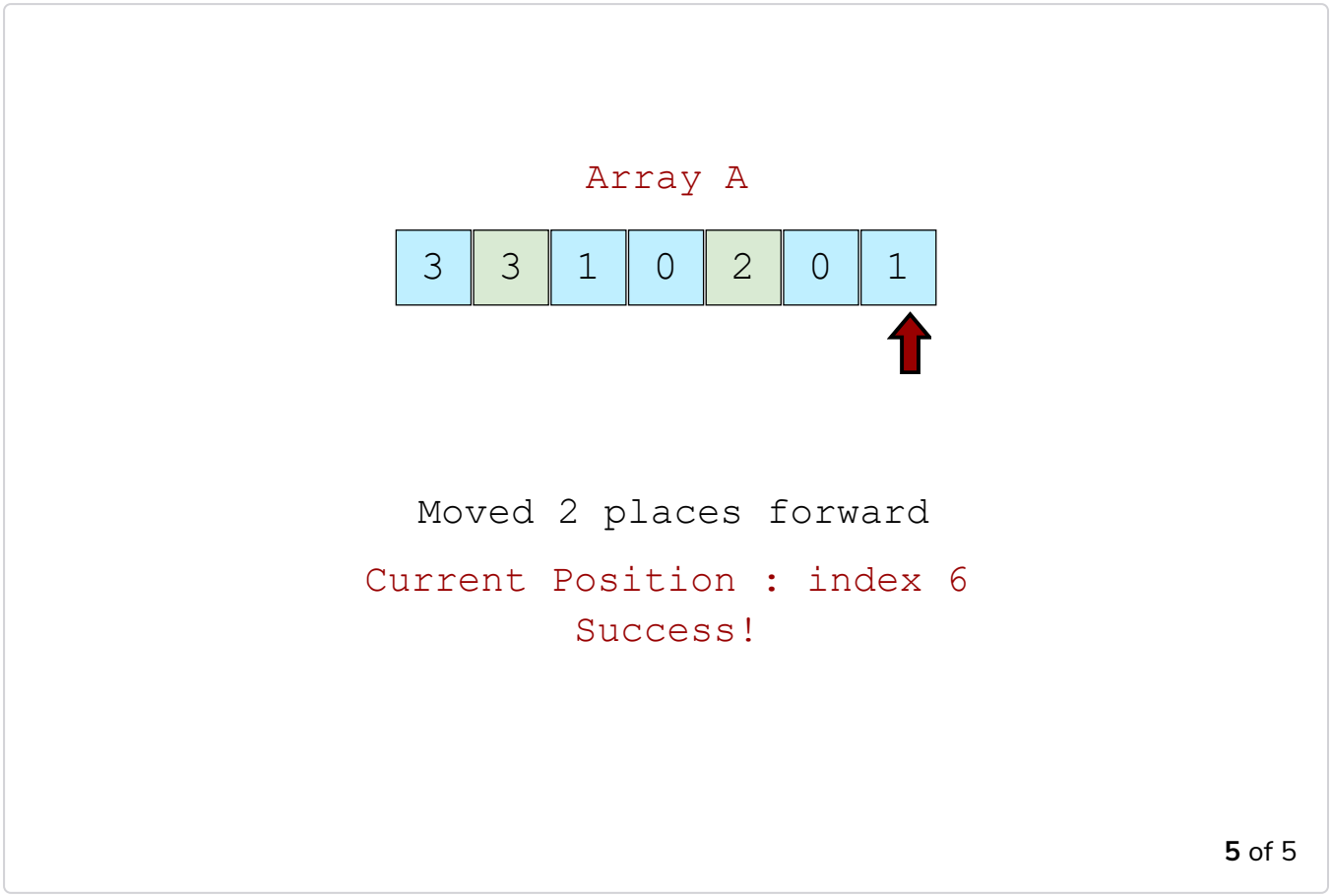
Current Position : index 1
Maximum that you can advance:
= A[1] = 3

Array A

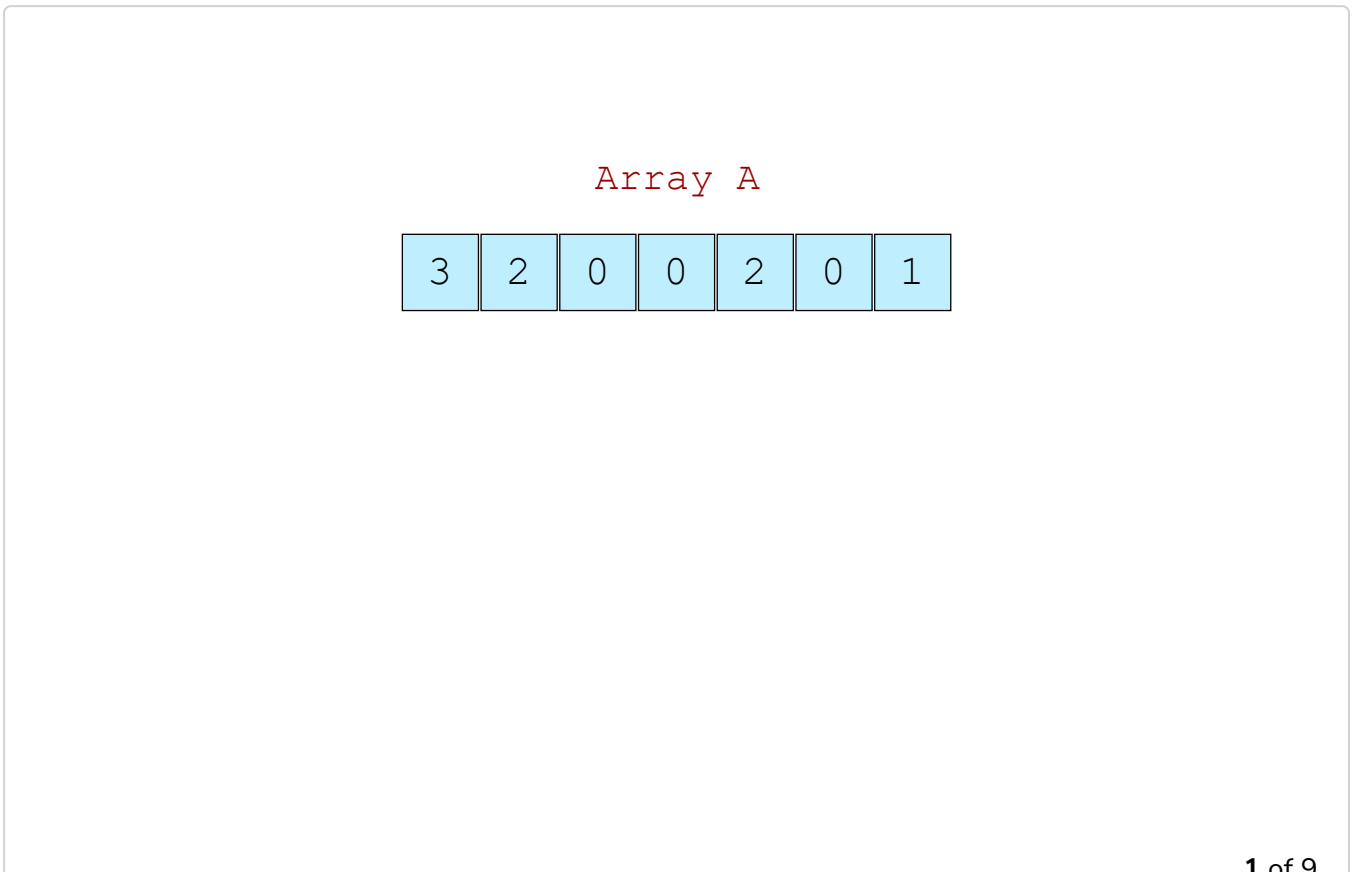


Moved 3 places forward

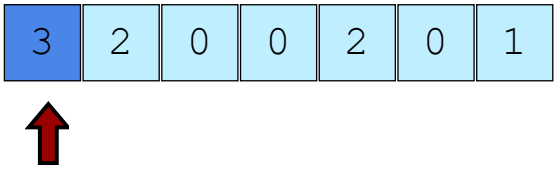
Current Position : index 4
Maximum that you can advance:
= A[4] = 2



In the example above, we have successfully reached the end of the array. Let's look at a case that is *unwinnable* where we can't reach the end of the array.

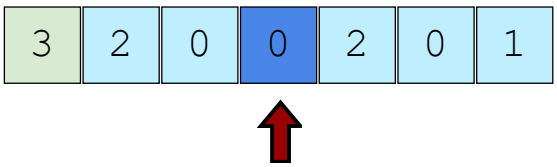


Array A



Current Position : index 0
Maximum that you can advance:
= A[0] = 3

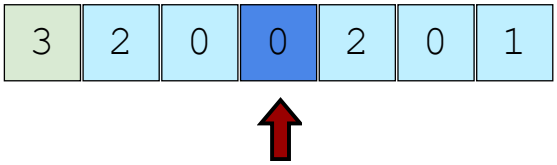
Array A



Moved 3 places

Current Position : index 3
Maximum that you can advance:
= A[3] = 0

Array A

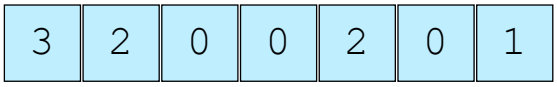


Unwinnable!

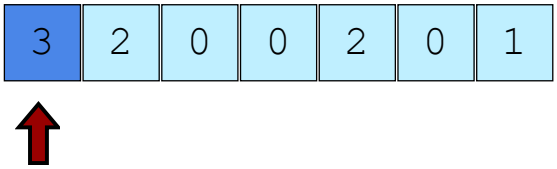
Current Position : index 3
Maximum that you can advance:
= A[3] = 0

Let's try another strategy!

Array A

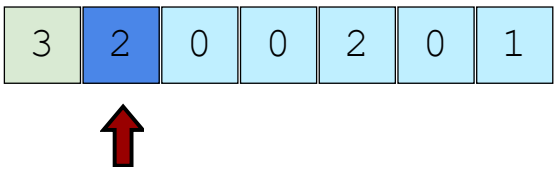


Array A



Current Position : index 0
Maximum that you can advance:
= A[0] = 3

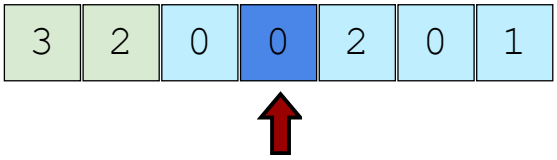
Array A



Move 1 place

Current Position : index 1
Maximum that you can advance:
= A[1] = 2

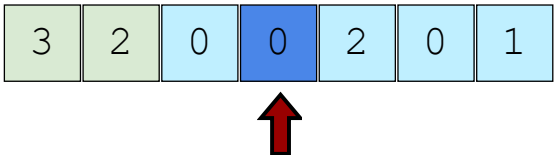
Array A



Move 2 places

Current Position : index 3
Maximum that you can advance:
= A[3] = 0

Array A



Unwinnable!

Current Position : index 3
Maximum that you can advance:
= A[3] = 0

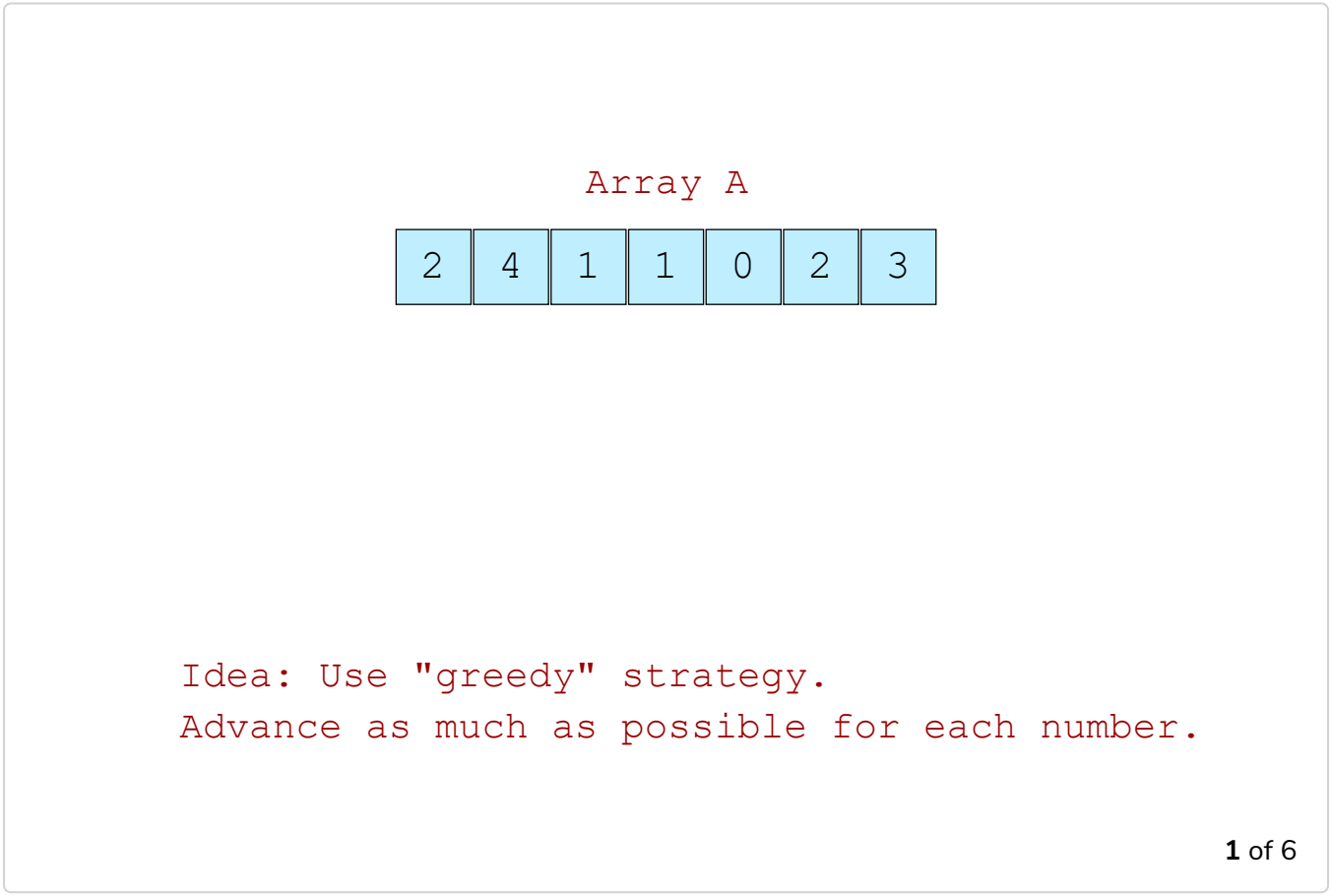
Algorithm

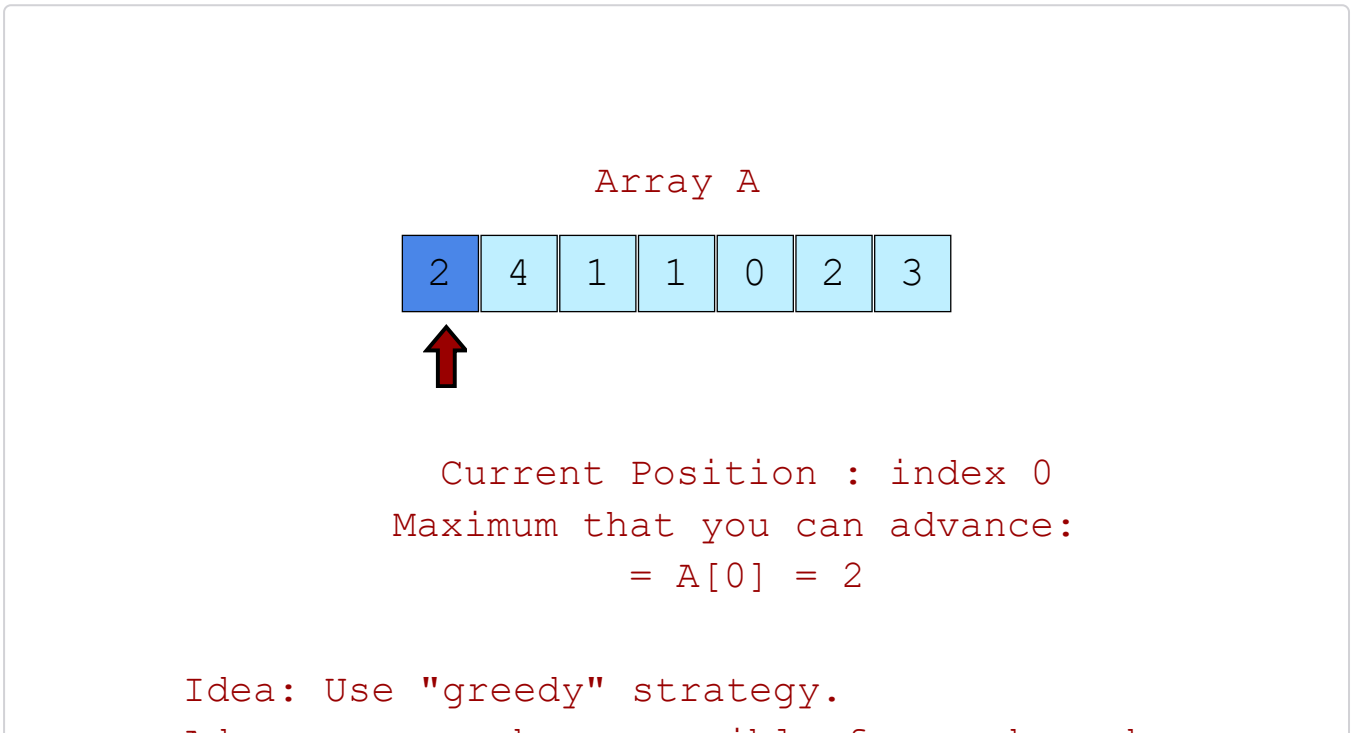
I hope you are clear about the game and the problem statement at this point. We have to figure out the best way to advance in the array so that we reach the last index.

Greedy Approach

Let's try the *greedy* approach in which we always advance the maximum we can at every index.

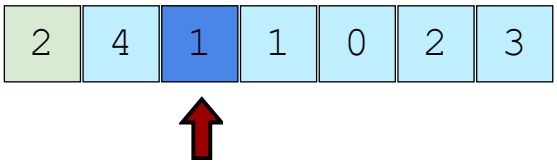
The example below illustrates the greedy approach.





Advance as much as possible for each number.

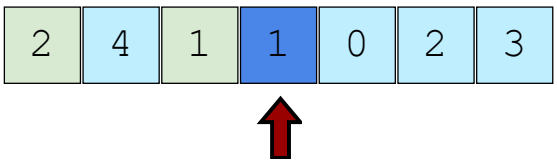
Array A



Current Position : index 2
Maximum that you can advance:
= A[2] = 1

Idea: Use "greedy" strategy.
Advance as much as possible for each number.

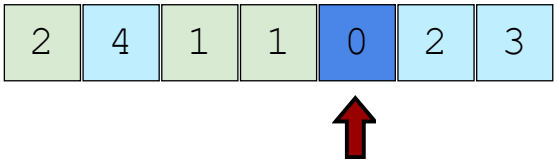
Array A



Current Position : index 3
Maximum that you can advance:
= A[3] = 1

Idea: Use "greedy" strategy.
Advance as much as possible for each number.

Array A

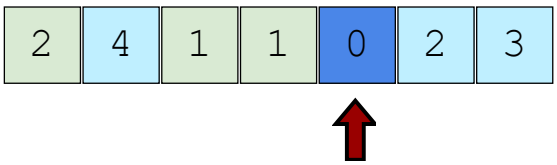


Current Position : index 4
Maximum that you can advance:
= A[4] = 0

Idea: Use "greedy" strategy.
Advance as much as possible for each number.

5 of 6

Array A



Unwinnable!
Current Position : index 4
Maximum that you can advance:
= A[4] = 0

Idea: Use "greedy" strategy.
Advance as much as possible for each number.

6 of 6

The example above perfectly illustrates that the *greedy* approach doesn't work. However, the array is not unwinnable. Have a look at the slides below:

Array A

2	4	1	1	0	2	3

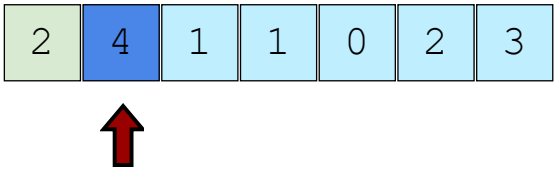
Idea: Use "non-greedy" strategy.

Array A

2	4	1	1	0	2	3

Current Position : index 0
Maximum that you can advance:
= A[0] = 2

Array A



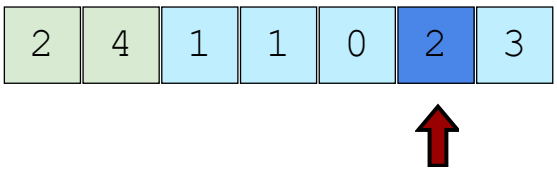
Moved 1 place

Current Position : index 1

Maximum that you can advance:

= A[1] = 4

Array A

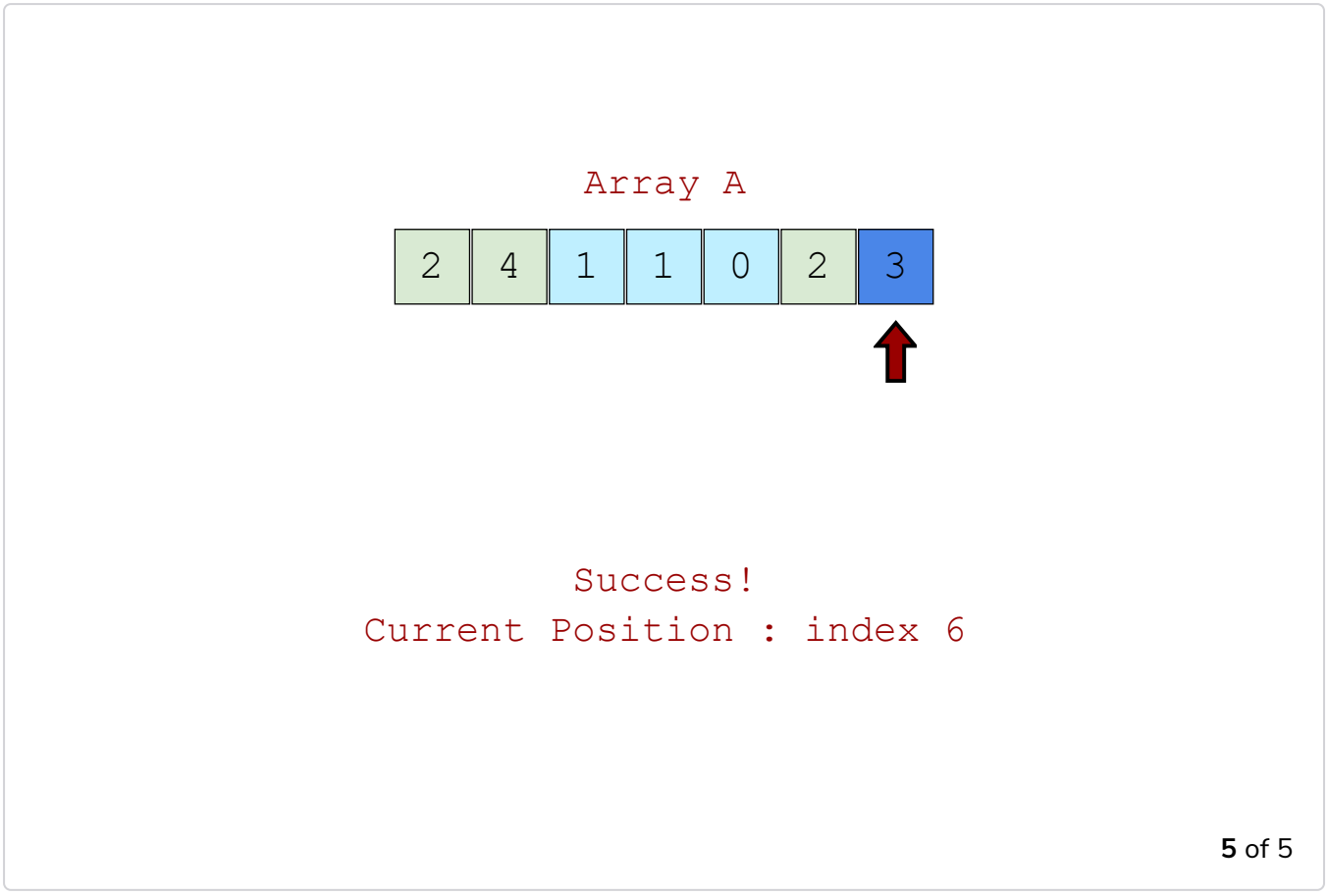


Moved 4 places forward

Current Position : index 5

Maximum that you can advance:

= A[5] = 2

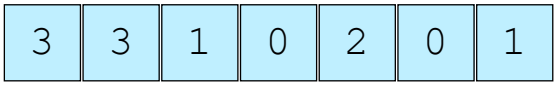


As you can see, the greedy algorithm doesn't work for this solution. Let's look at the algorithm which will solve the problem:

- Iterate through each entry in an array, `A`.
- Track the furthest we can reach from entry (`A[i]` + `i`)
- If for some `i`, we don't reach the end and that is the furthest we can reach, then we can't reach the last index. Otherwise, the end is reached.
- `i`: index processed. Furthest possible to advance from `i`: `A[i]` + `i`

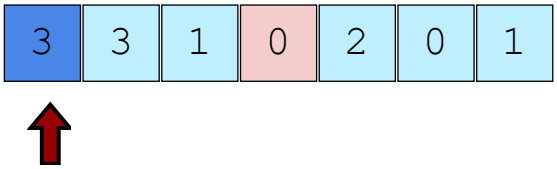
The example below will run through each step of the algorithm stated below:

Array A



furthest_reached = 0

Array A



i=0

furthest_reached = 0

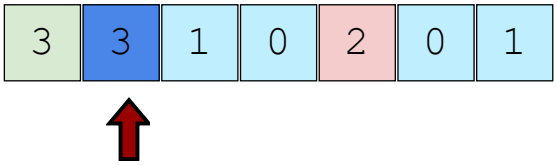
furthest_reached=max(furthest_reached, A[i] + i)

furthest_reached=max(furthest_reached, A[0] + 0)

= max(0,3+0)

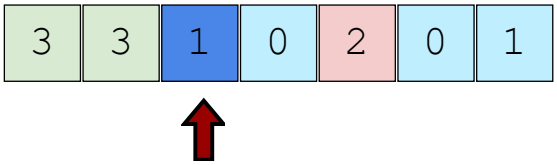
= 3

Array A



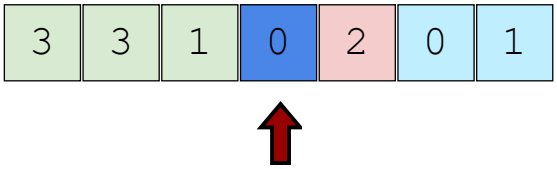

```
i=1
furthest_reached = 3
furthest_reached=max(furthest_reached, A[i] + i)
furthest_reached=max(furthest_reached, A[1] + 1)
= max(3,3+1)
= 4
```


Array A




```
i=2
furthest_reached = 4
furthest_reached=max(furthest_reached, A[i] + i)
furthest_reached=max(furthest_reached, A[2] + 2)
= max(4,1+2)
= 4
```

Array A



i=3

furthest_reached = 4

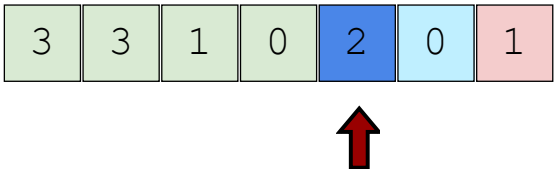
furthest_reached=max(furthest_reached, A[i] + i)

furthest_reached=max(furthest_reached, A[3] + 3)

= max(4,0+3)

= 4

Array A



i=4

furthest_reached = 4

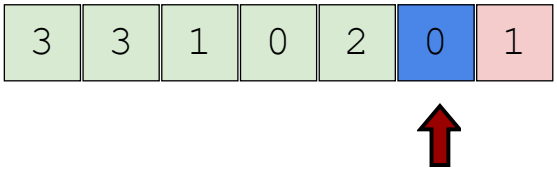
furthest_reached=max(furthest_reached, A[i] + i)

furthest_reached=max(furthest_reached, A[4] + 4)

= max(4,2+4)

= 6

Array A

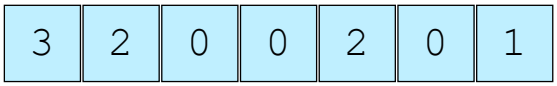



```
i=5
furthest_reached = 6
furthest_reached=max(furthest_reached, A[i] + i)
furthest_reached=max(furthest_reached, A[5] + 5)
= max(6,0+5)
= 6
```

7 of 7

The above example was of a win situation. Let's see how our algorithm works for arrays which are not winnable.

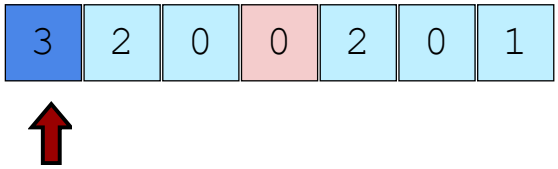
Array A




```
furthest_reached = 0
```

1 of 6

Array A



i=0

furthest_reached = 0

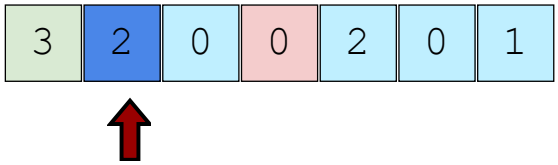
furthest_reached=max(furthest_reached, A[i] + i)

furthest_reached=max(furthest_reached, A[0] + 0)

= max(0,3+0)

= 3

Array A



i=1

furthest_reached = 3

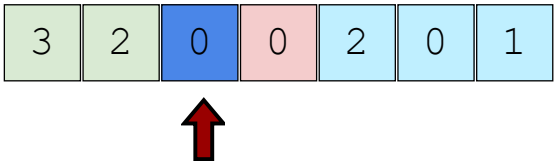
furthest_reached=max(furthest_reached, A[i] + i)

furthest_reached=max(furthest_reached, A[1] + 1)

= max(3,2+1)

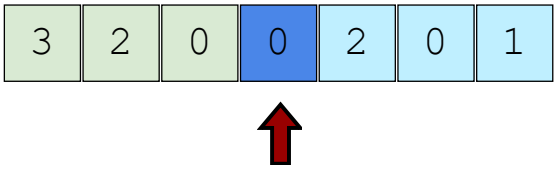
= 3

Array A

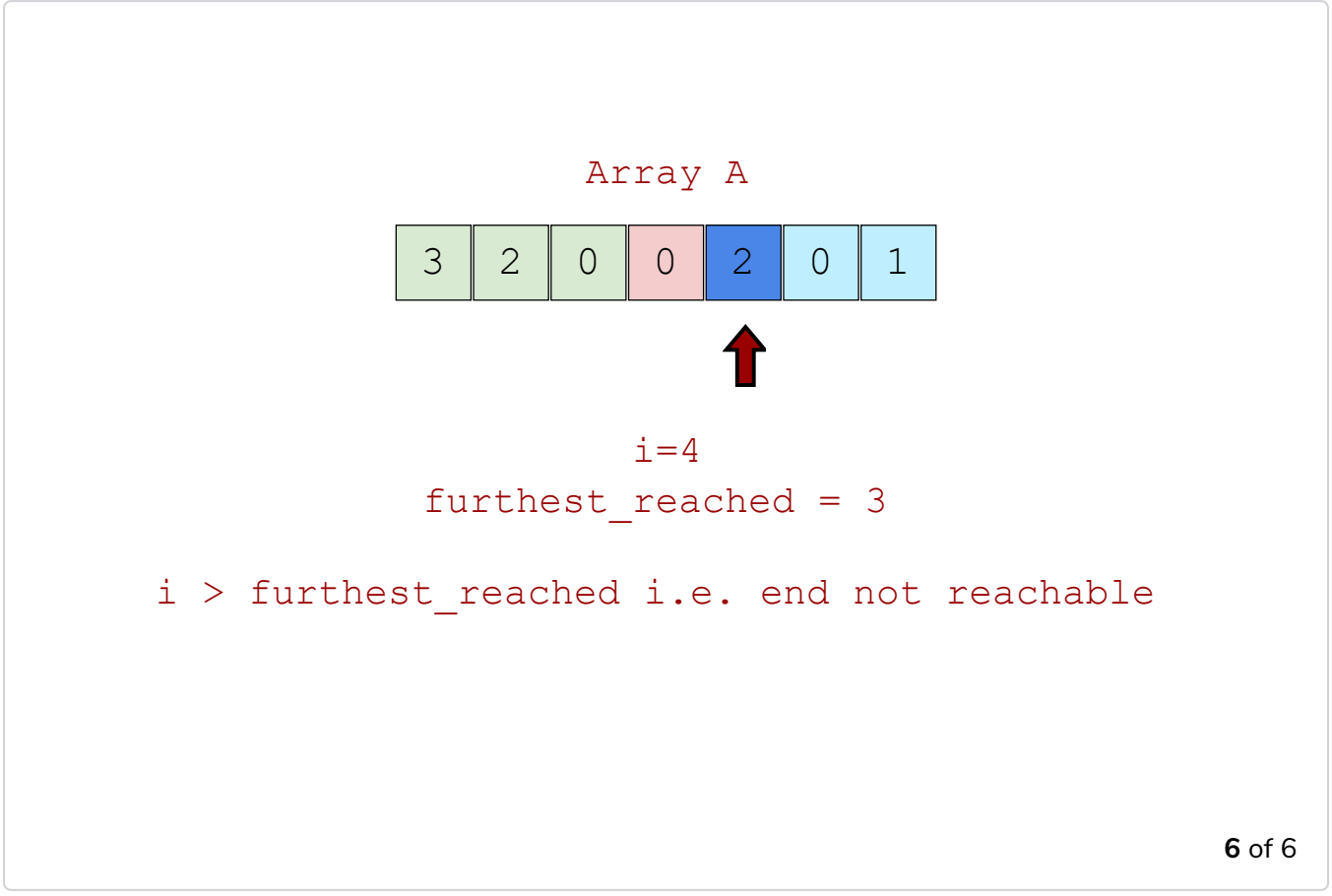



```
i=2
furthest_reached = 3
furthest_reached=max(furthest_reached, A[i] + i)
furthest_reached=max(furthest_reached, A[2] + 2)
= max(3,0+2)
= 3
```


Array A




```
i=3
furthest_reached = 3
furthest_reached=max(furthest_reached, A[i] + i)
furthest_reached=max(furthest_reached, A[3] + 3)
= max(3,0+3)
= 3
```



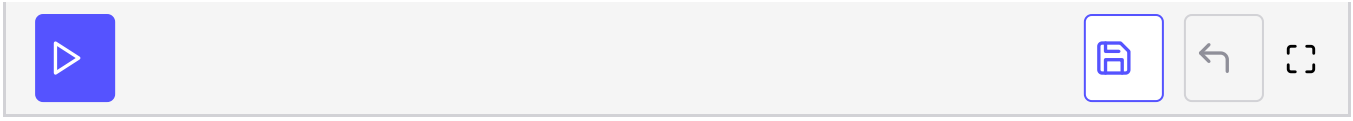
Implementation

Hopefully, by now, you will clearly understand the algorithm. Let's jump to the implementation in Python, which will be easier to understand once you get the algorithm:

```
def array_advance(A):
    furthest_reached = 0
    last_idx = len(A) - 1
    i = 0
    while i <= furthest_reached and furthest_reached < last_idx:
        furthest_reached = max(furthest_reached, A[i] + i)
        i += 1
    return furthest_reached >= last_idx


# True: Possible to navigate to last index in A:
# Moves: 1,3,2
A = [3, 3, 1, 0, 2, 0, 1]
print(array_advance(A))

# False: Not possible to navigate to last index in A:
A = [3, 2, 0, 0, 2, 0, 1]
print(array_advance(A))
```



array_advance(A)

Explanation

`furthest_reached` and `i` are initialized to `0` on **line 2** and **line 4** respectively. We calculate `last_idx` on **line 3** by subtracting 1 from the length of the array. Next, we proceed to a `while` loop which terminates on the following conditions:

- `i` > `furthest_reached`: This implies that the end is not reachable.
- `furthest_reached` >= `last_idx` : This implies that the end is reachable.

In each iteration of the `while` loop, we update `furthest_reached` to the maximum of `furthest_reached` and (`A[i]` + `i`) on **line 6**. `i` increments by `1` in the next line.

After the `while` loop terminates, we'll check for the condition which terminates the `while` loop. If `furthest_reached >= last_idx`, then `True` is returned from the function. Otherwise, `False` is returned.

In the next lesson, we'll study another interesting and challenging problem. Stay tuned!