Exercise 8 – Shortest Path, Strings & Lindenmayer

Informatik I für Mathematiker und Physiker (HS 2015)

Yeara Kozlov

Slides courtesy of Kaan Yücer & Endri Dibra

ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

inf | Informatik Computer Science

Agenda

- HW #6 Feedback
- Shortest path
- Reading sequences of unknown lengths
- Strings
- Lindenmayer Systems
- Pointers on arrays
- HW #8 Pre discussion

HW #6 Feedback

Agenda

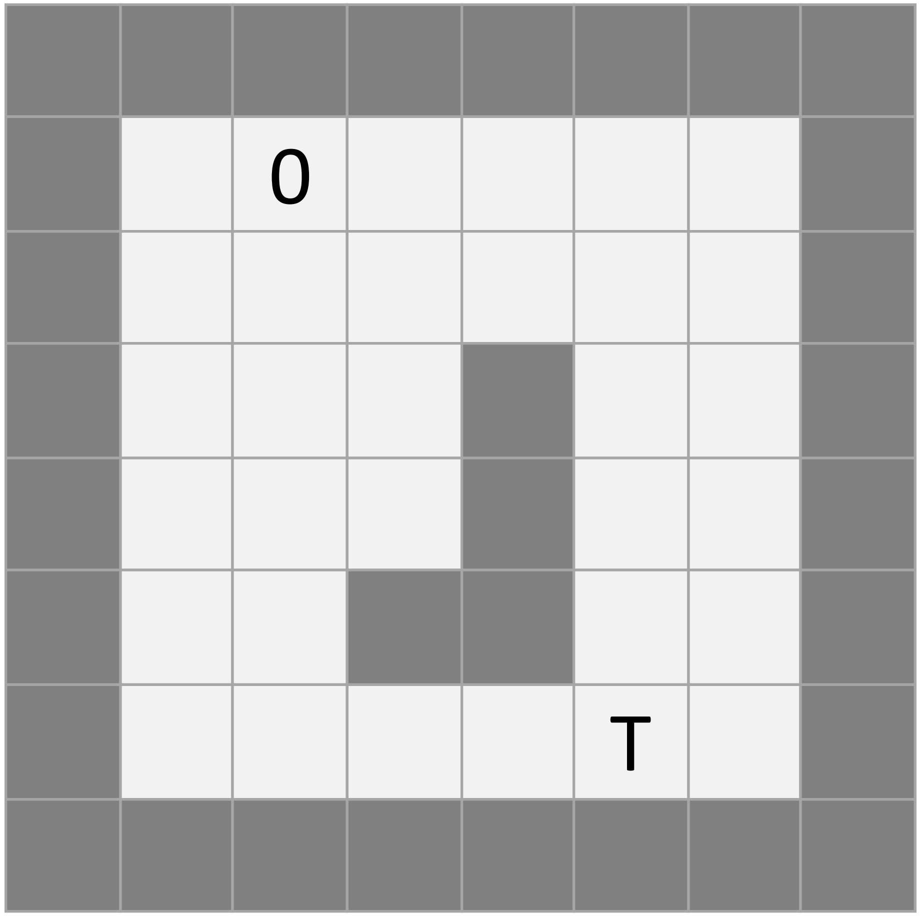
- HW #6 Feedback
- **Shortest path**
- Reading sequences of unknown lengths
- Strings
- Lindenmayer Systems
- Pointers on arrays
- HW #8 Pre discussion

Shortest Path Problem

Slow Version

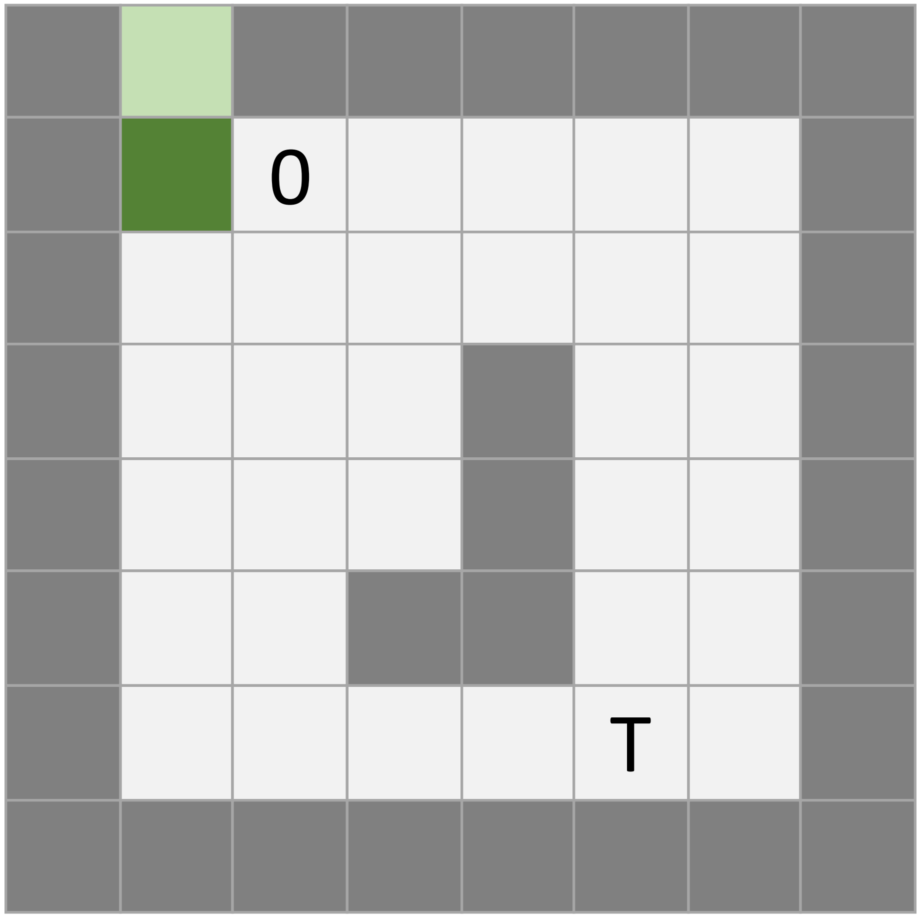
Slow Version

Path Length
`i = 1`



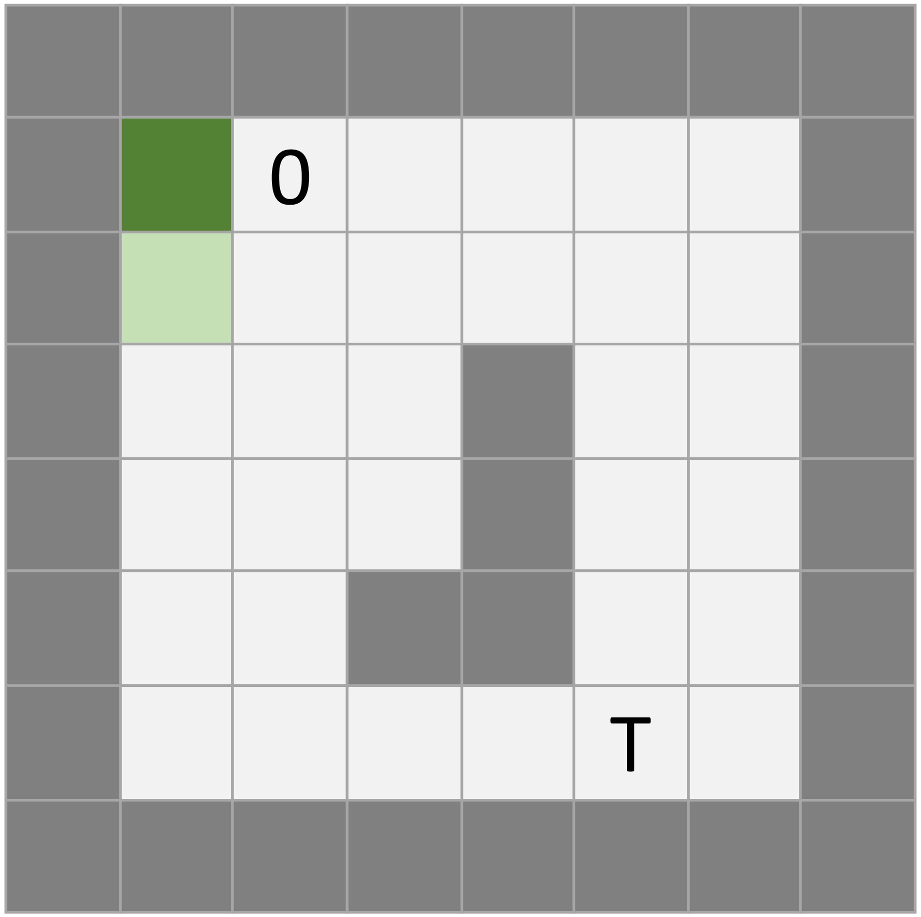
Slow Version

Path Length
`i = 1`



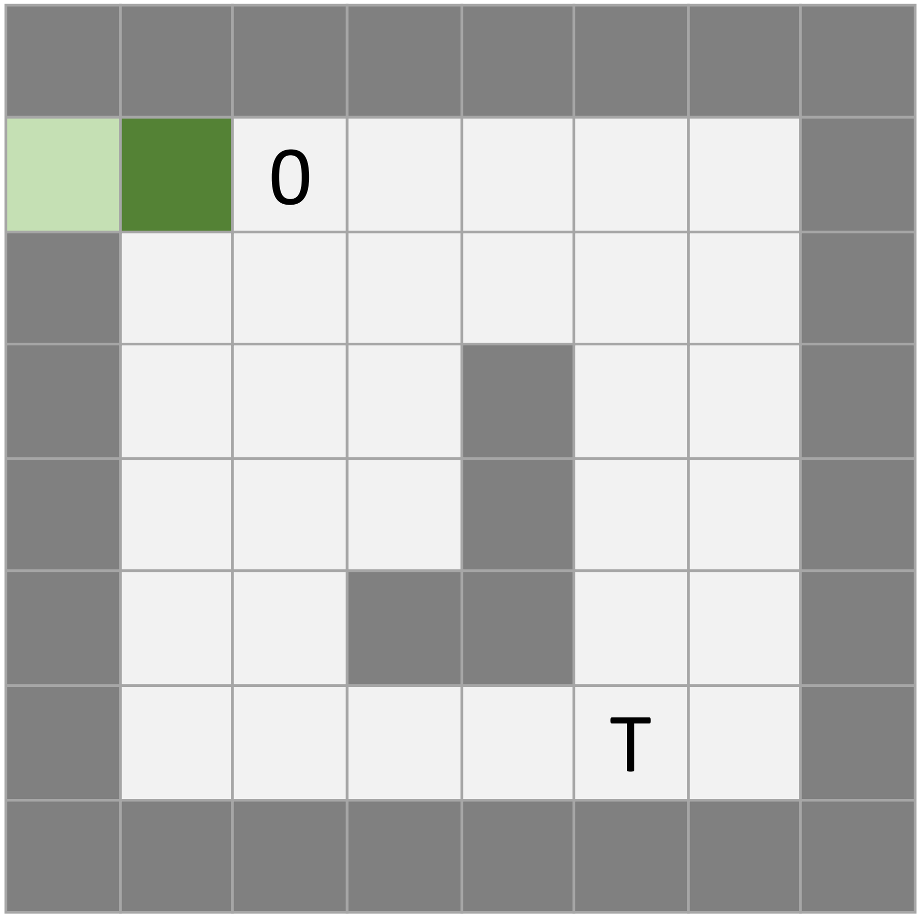
Slow Version

Path Length
`i = 1`



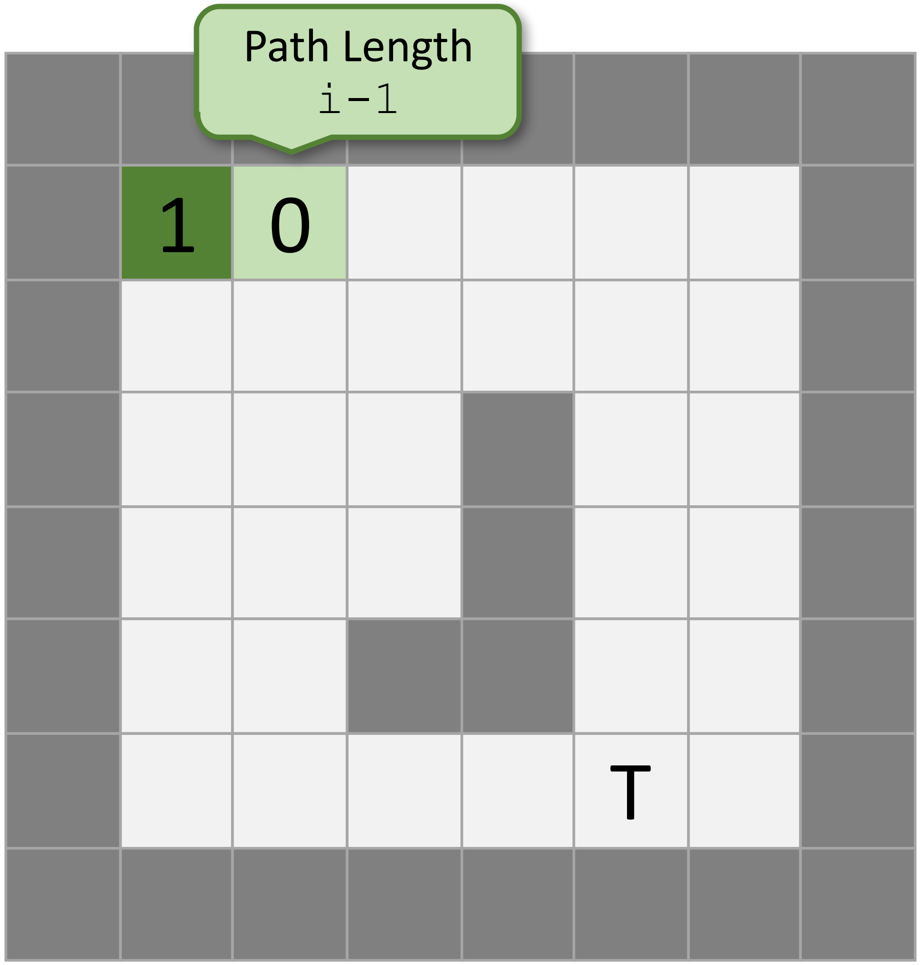
Slow Version

Path Length
`i = 1`



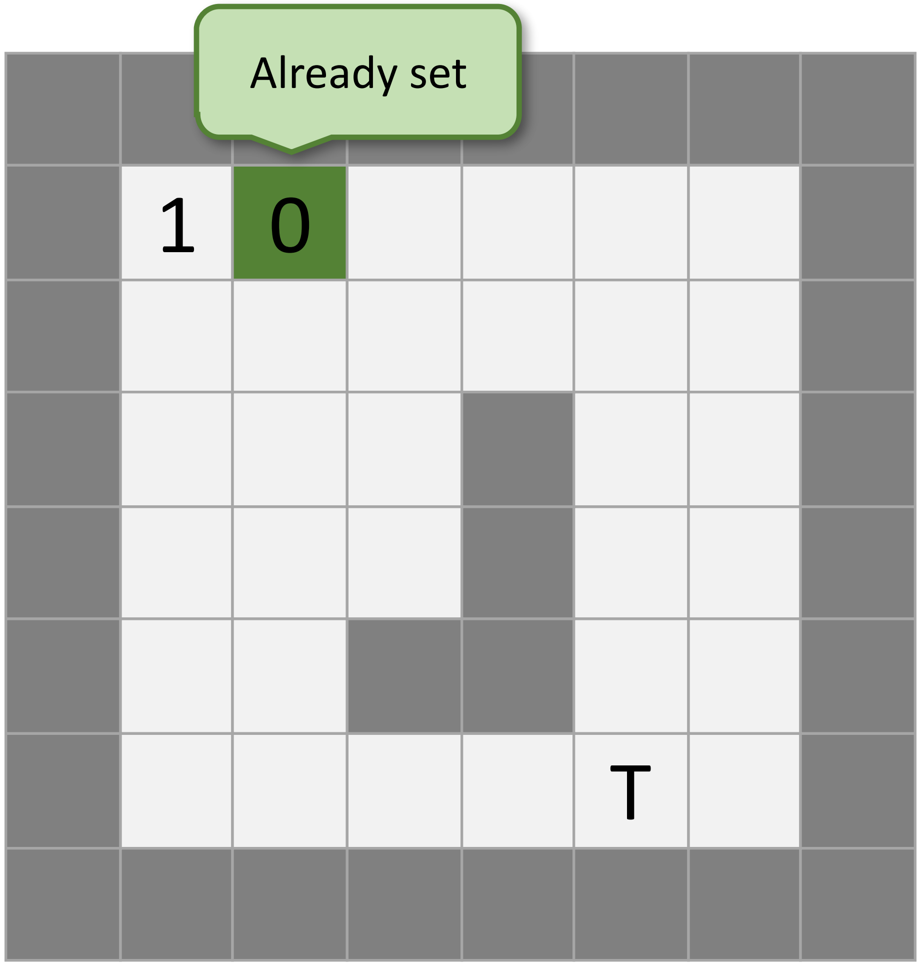
Slow Version

Path Length
`i = 1`



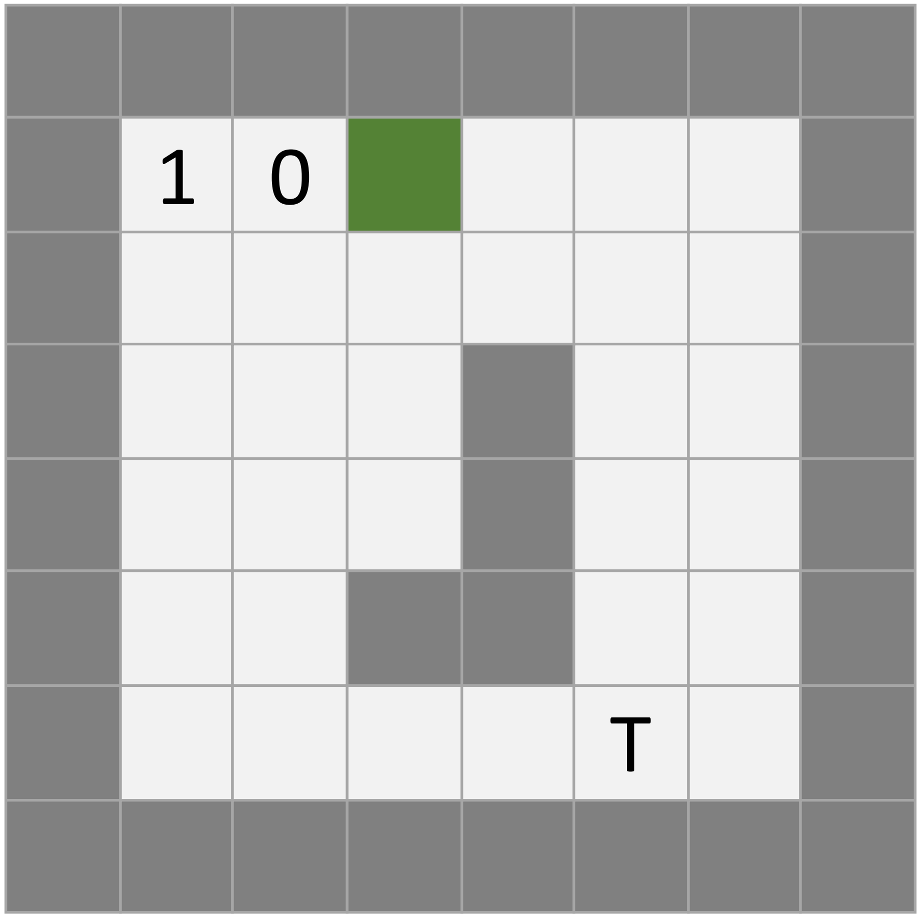
Slow Version

Path Length
`i = 1`



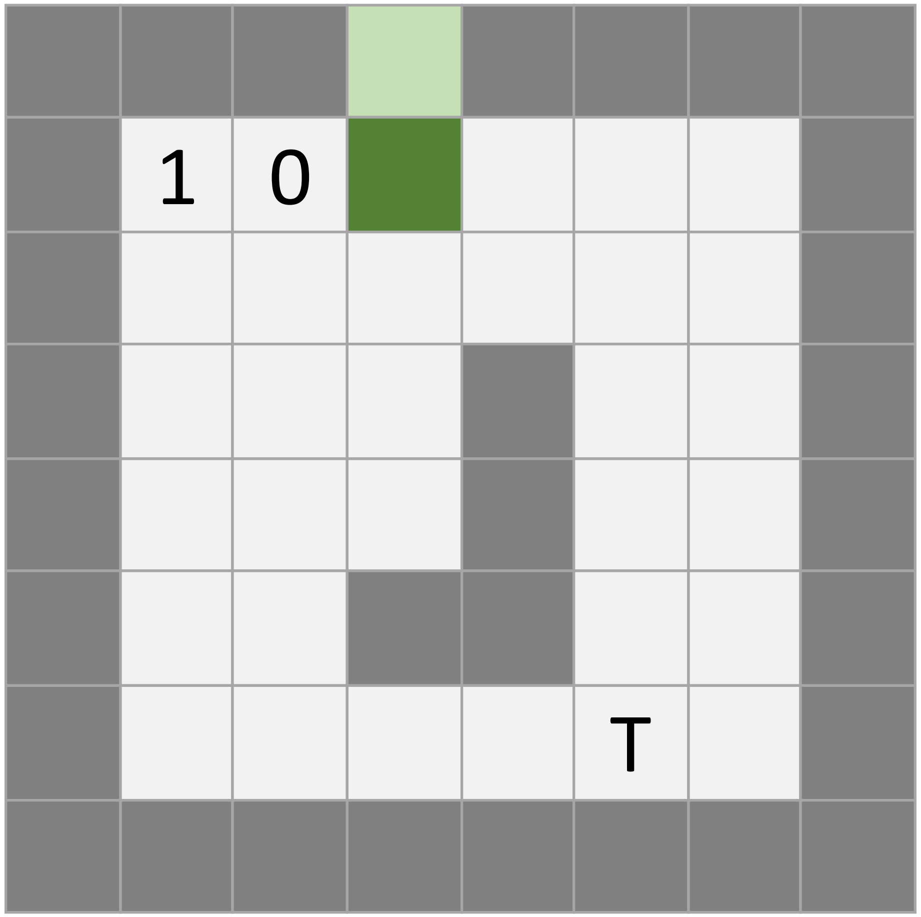
Slow Version

Path Length
`i = 1`



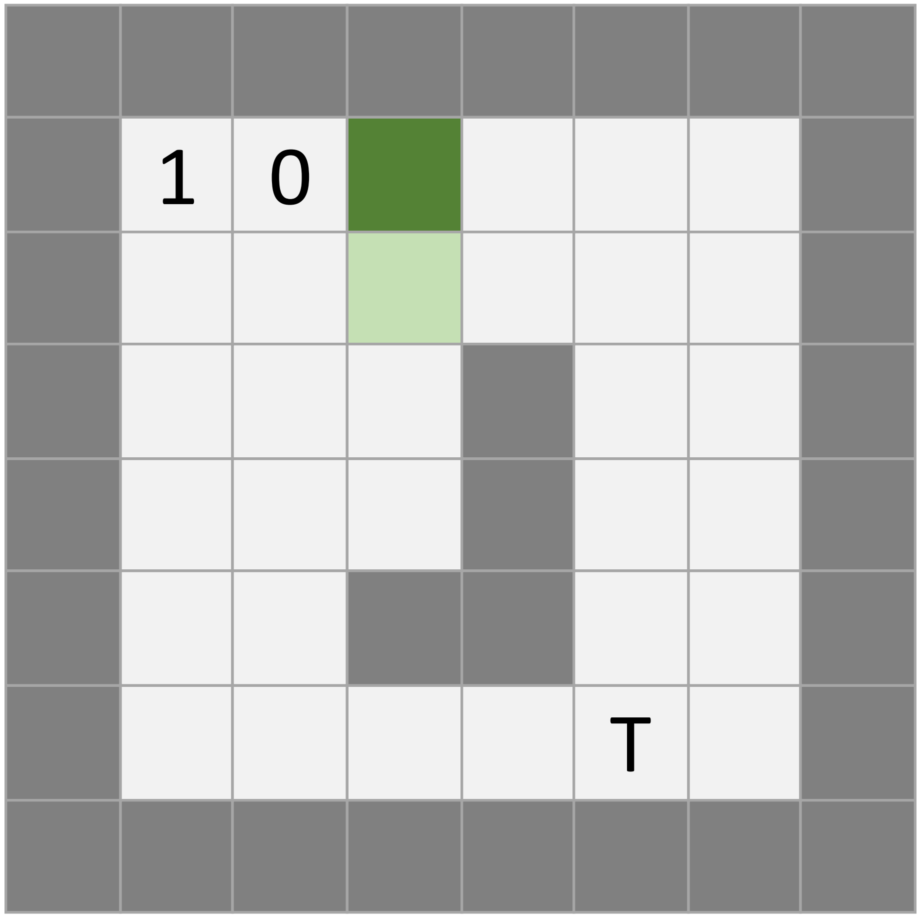
Slow Version

Path Length
`i = 1`



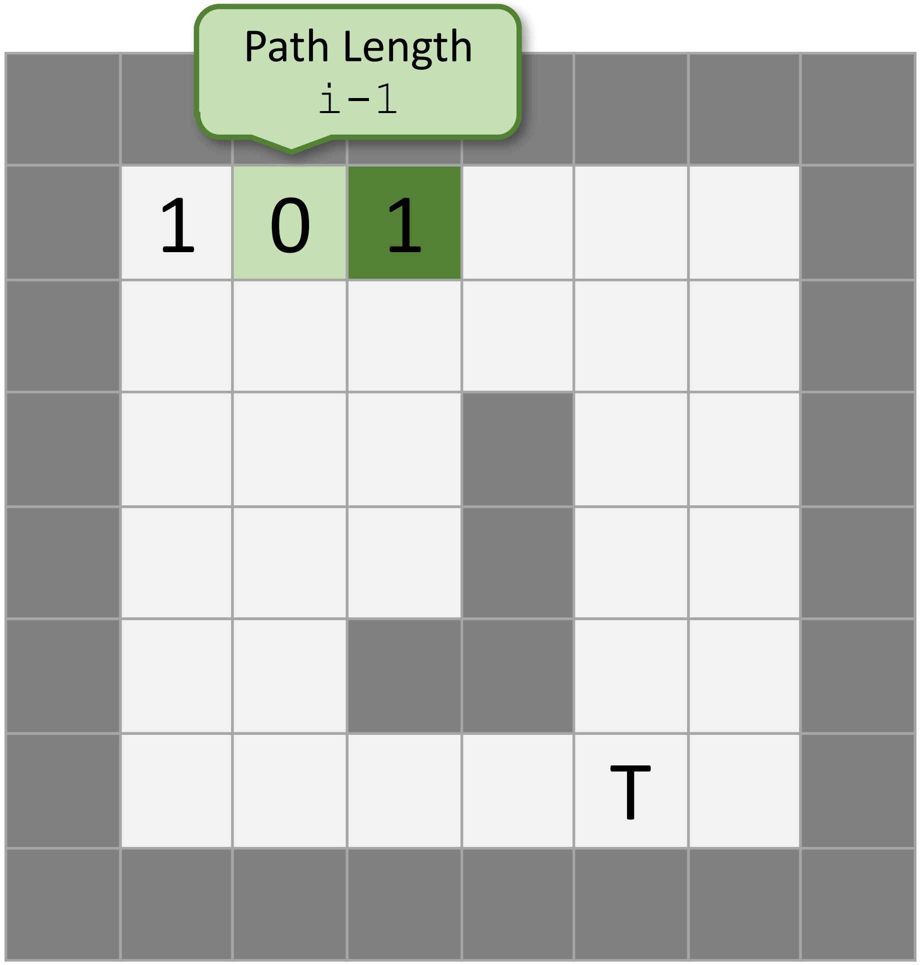
Slow Version

Path Length
`i = 1`



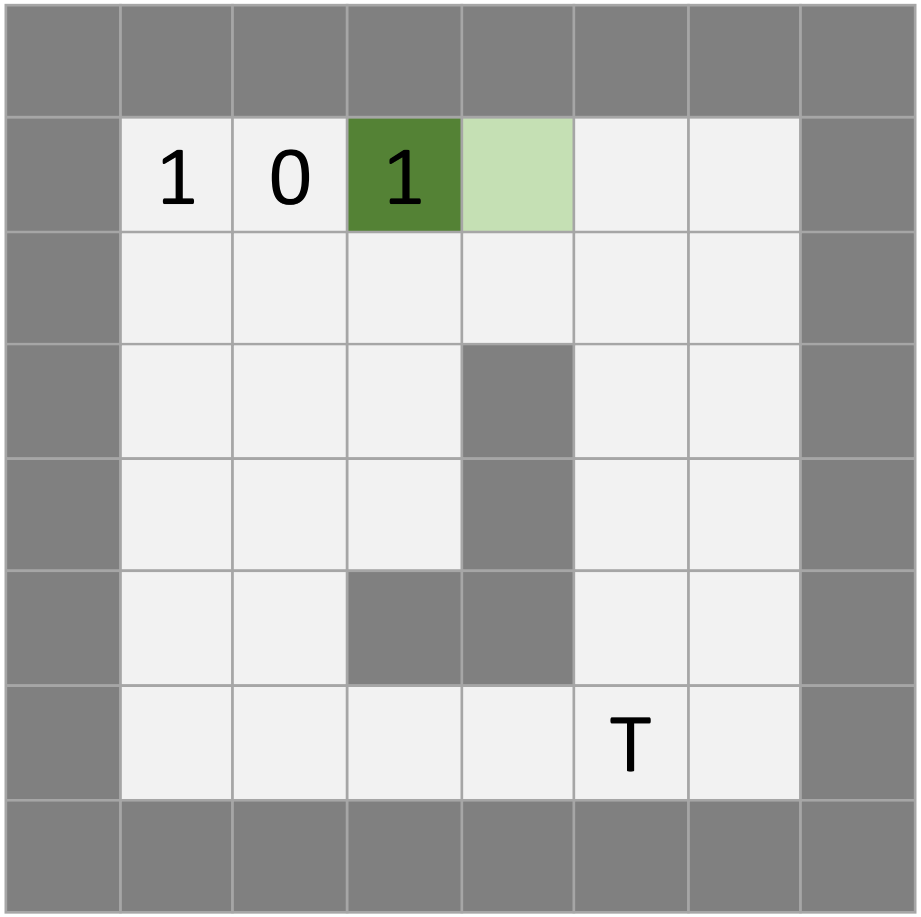
Slow Version

Path Length
`i = 1`



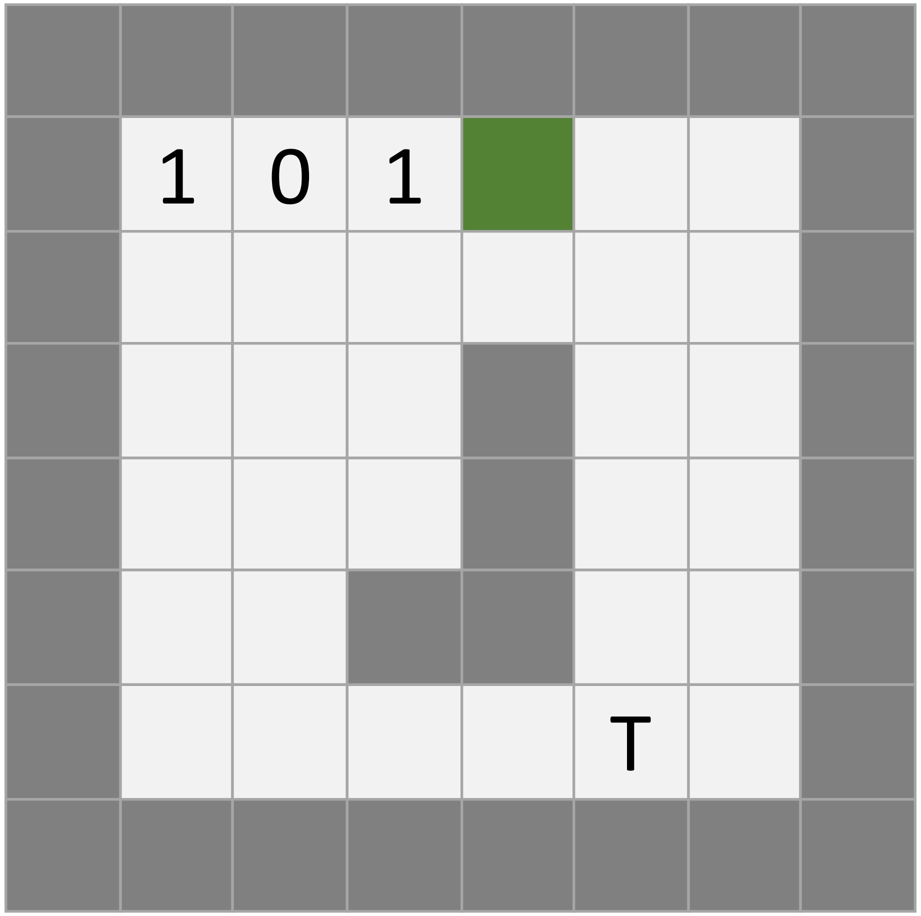
Slow Version

Path Length
`i = 1`



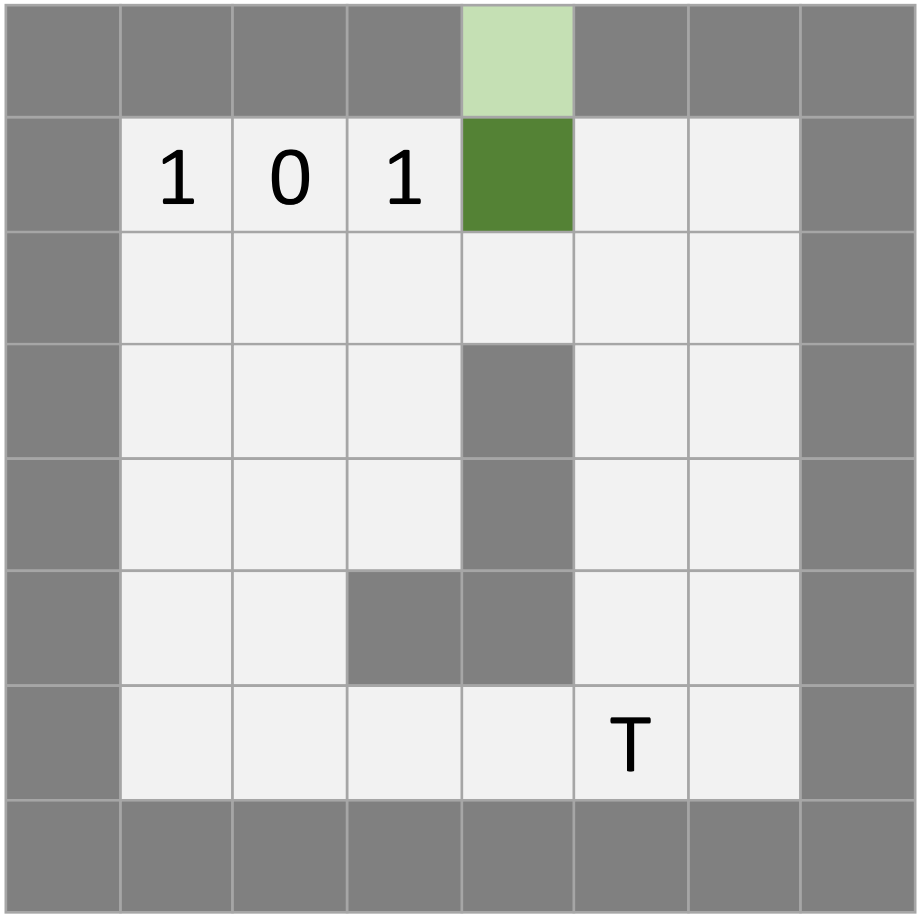
Slow Version

Path Length
`i = 1`



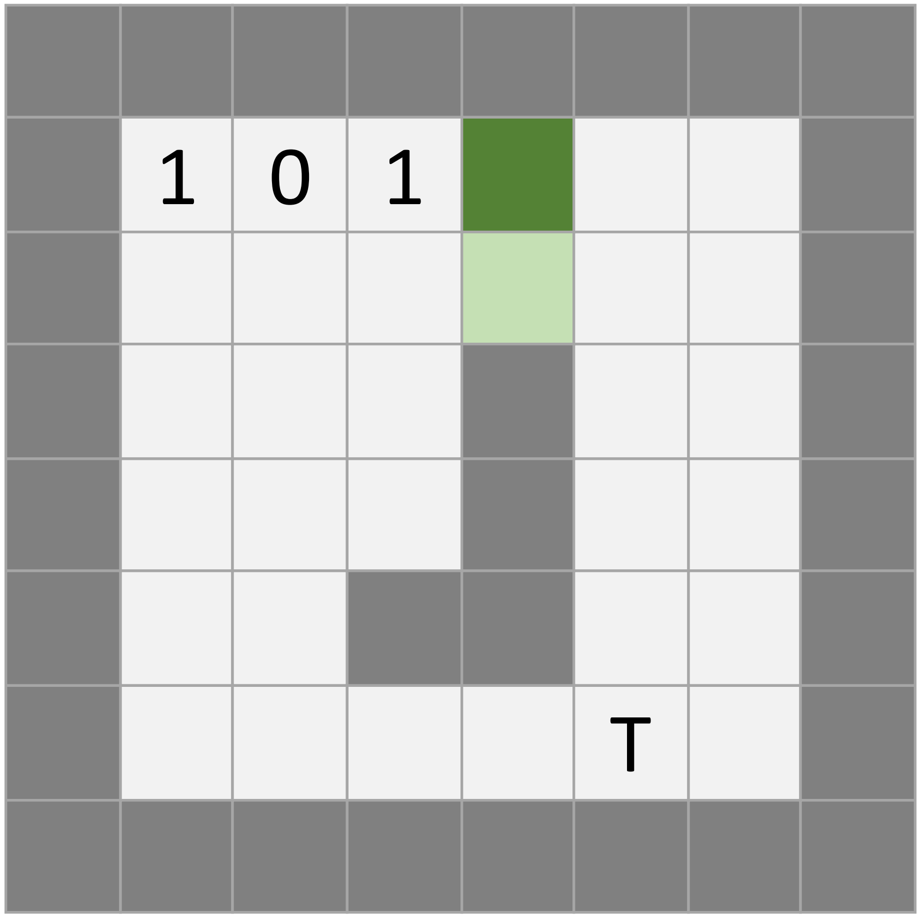
Slow Version

Path Length
`i = 1`



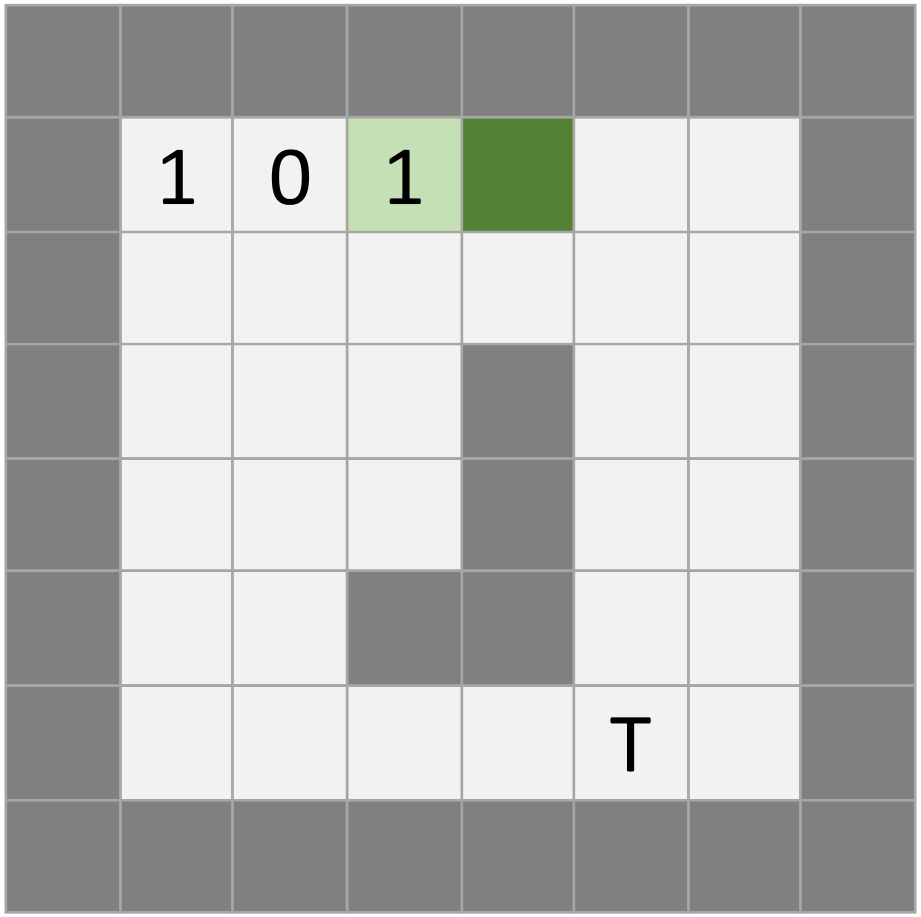
Slow Version

Path Length
`i = 1`



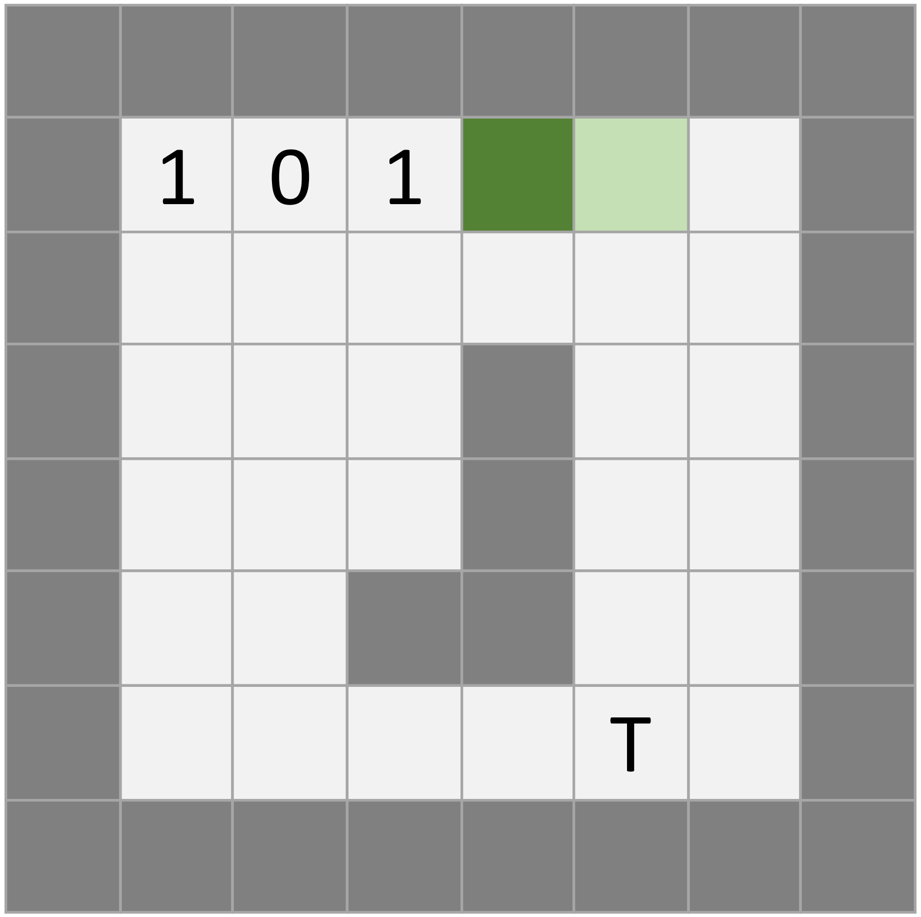
Slow Version

Path Length
`i = 1`



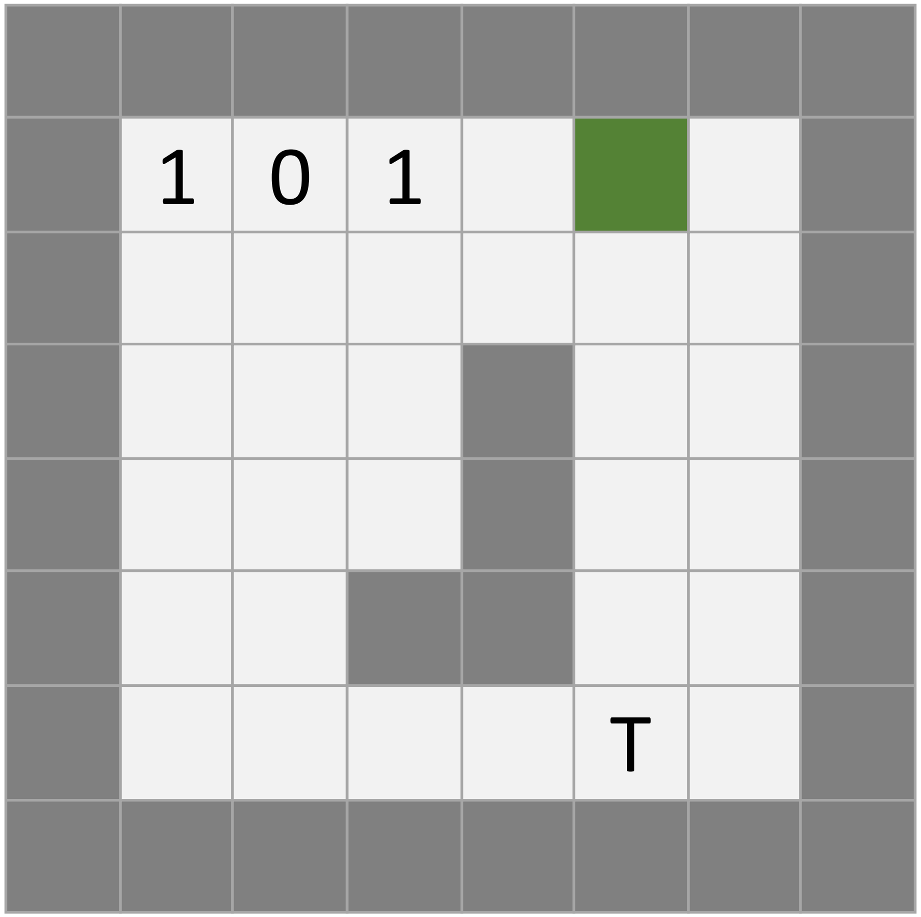
Slow Version

Path Length
`i = 1`



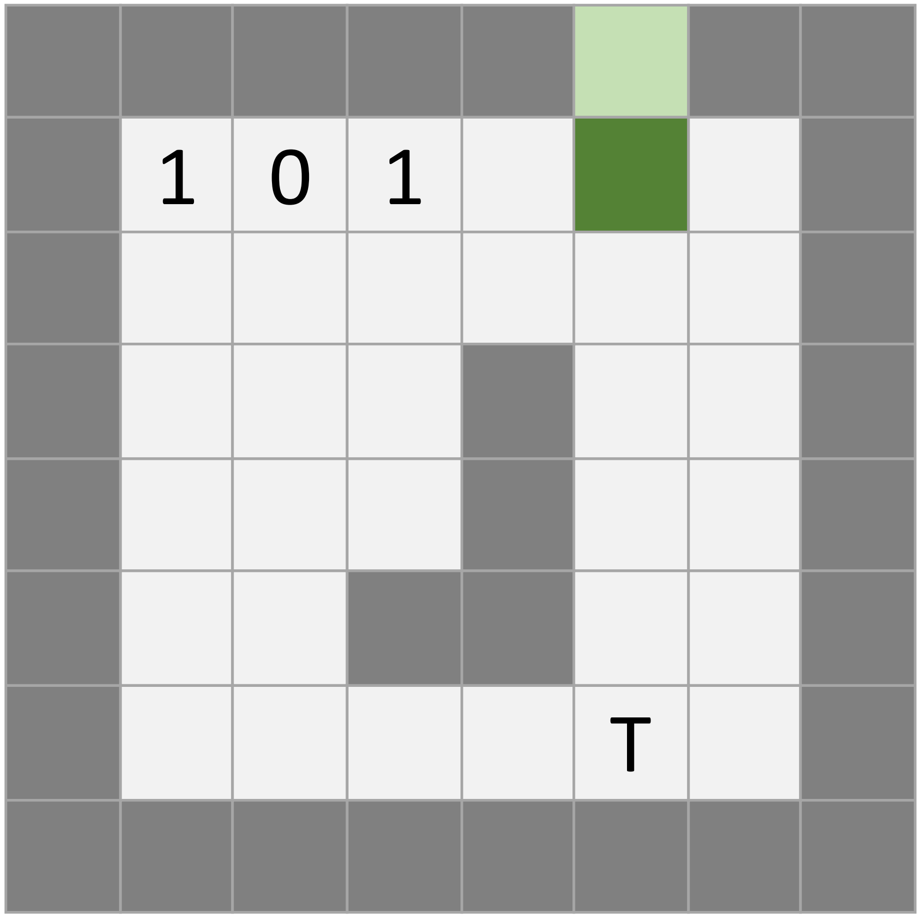
Slow Version

Path Length
`i = 1`



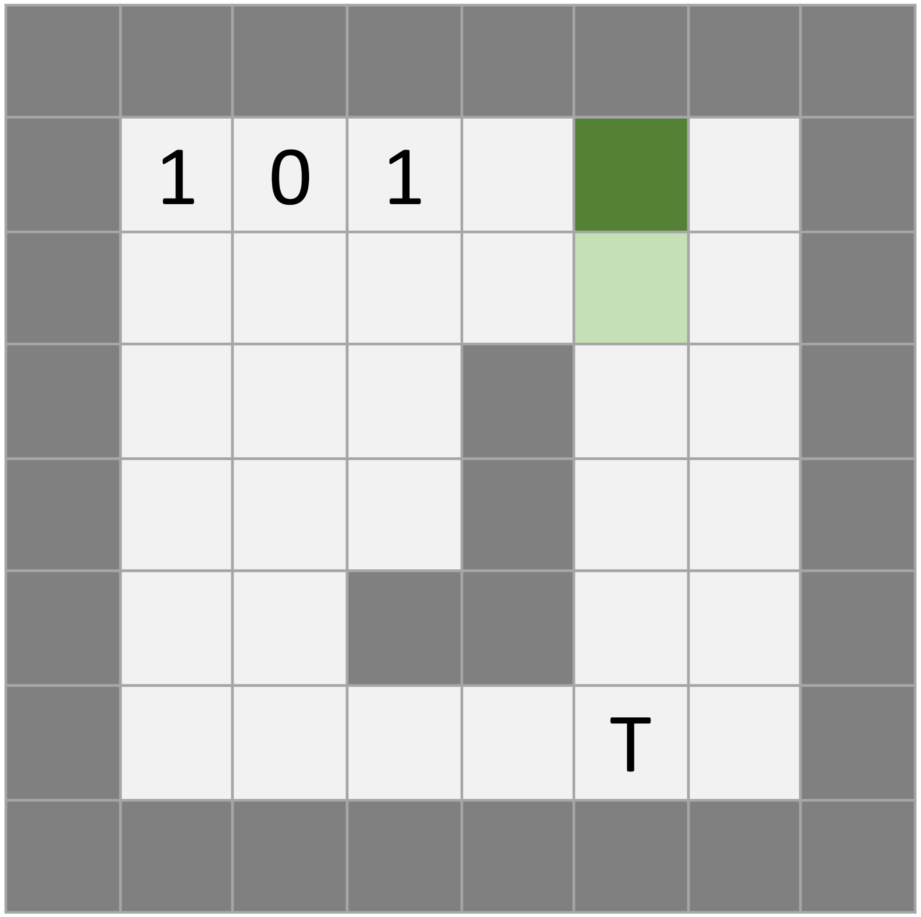
Slow Version

Path Length
`i = 1`



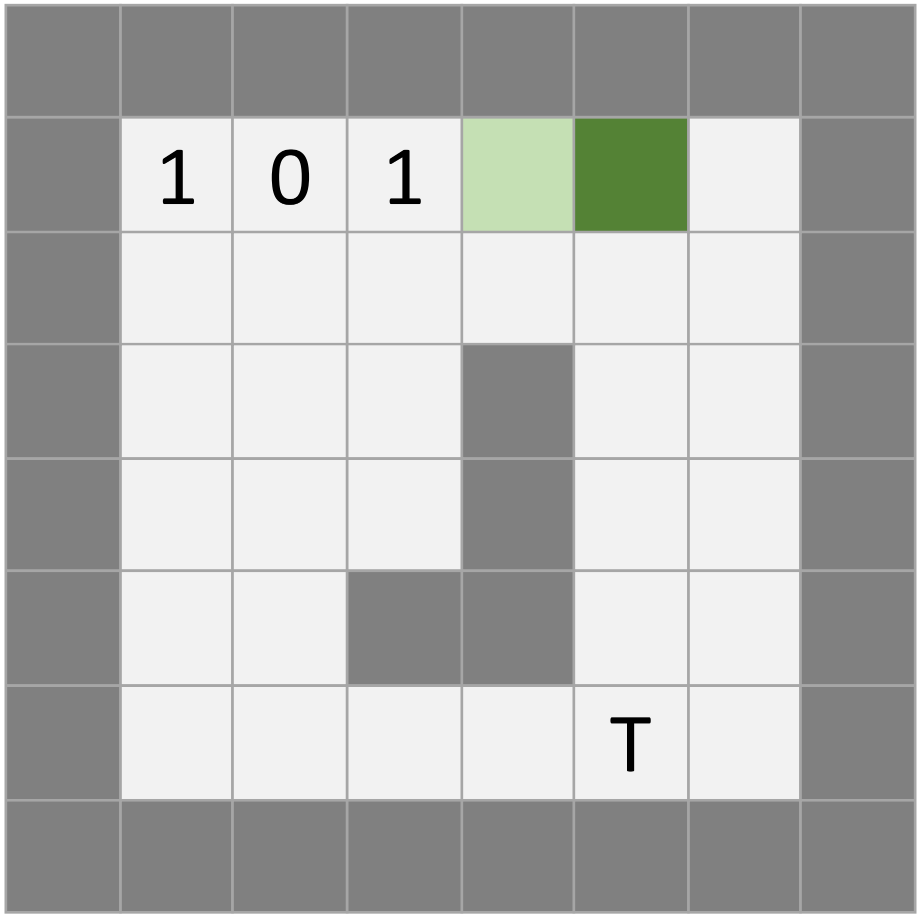
Slow Version

Path Length
`i = 1`



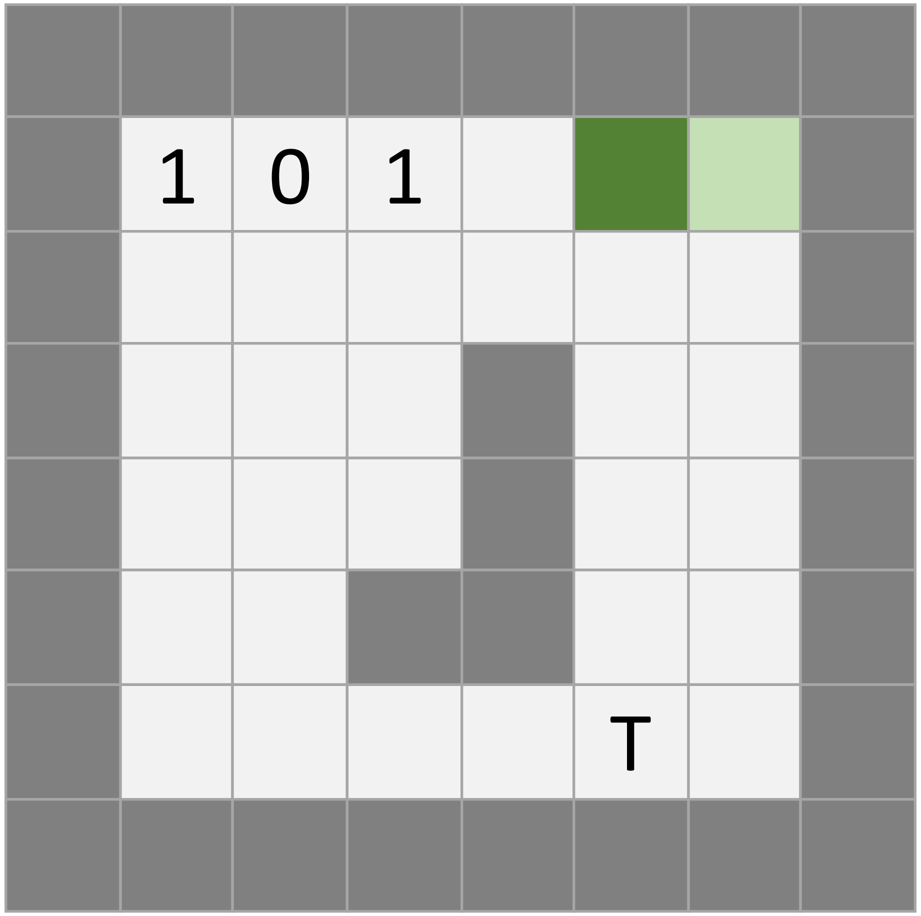
Slow Version

Path Length
`i = 1`



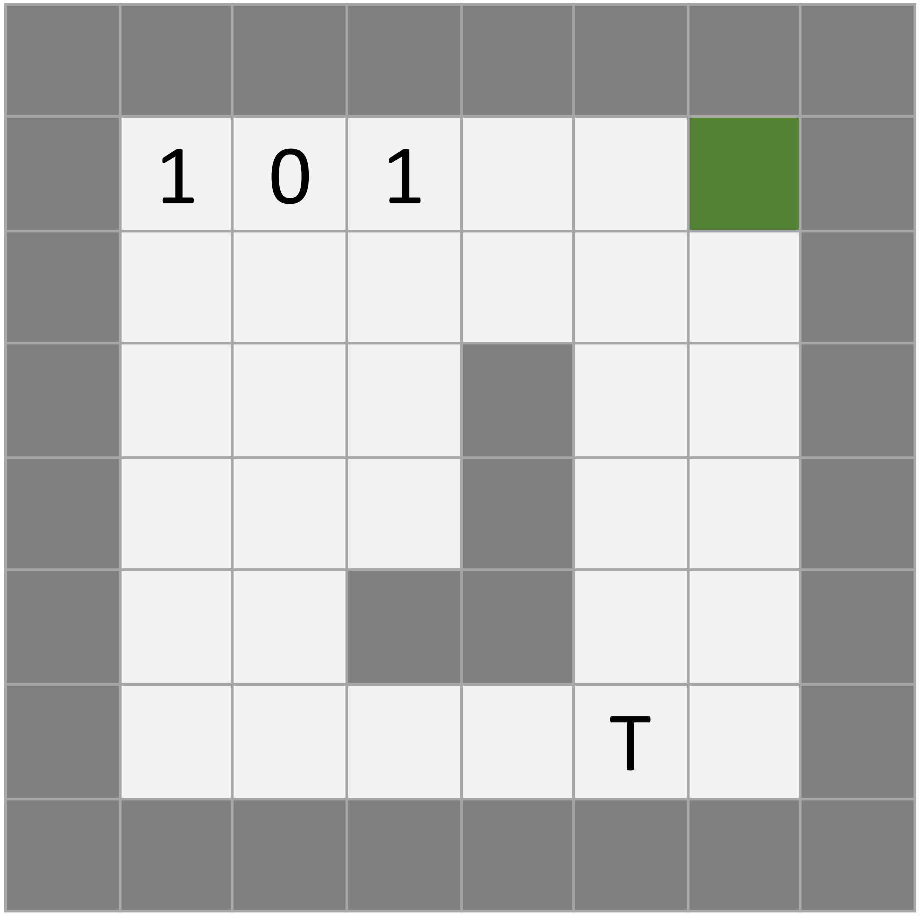
Slow Version

Path Length
`i = 1`



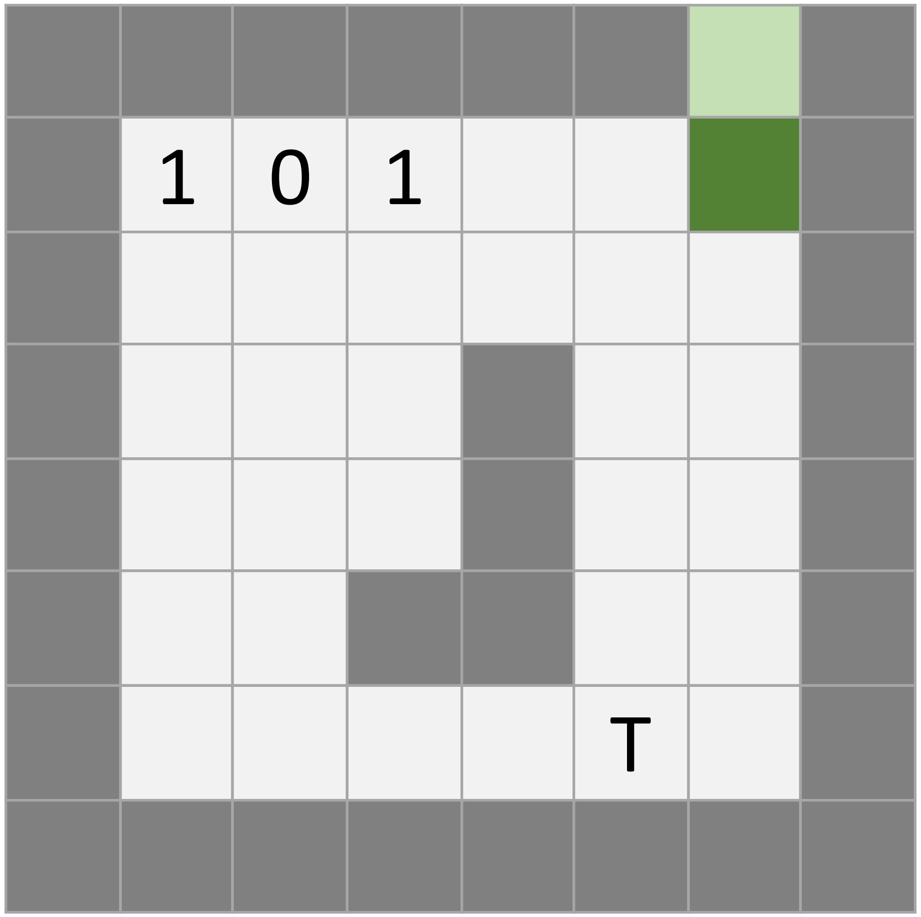
Slow Version

Path Length
`i = 1`



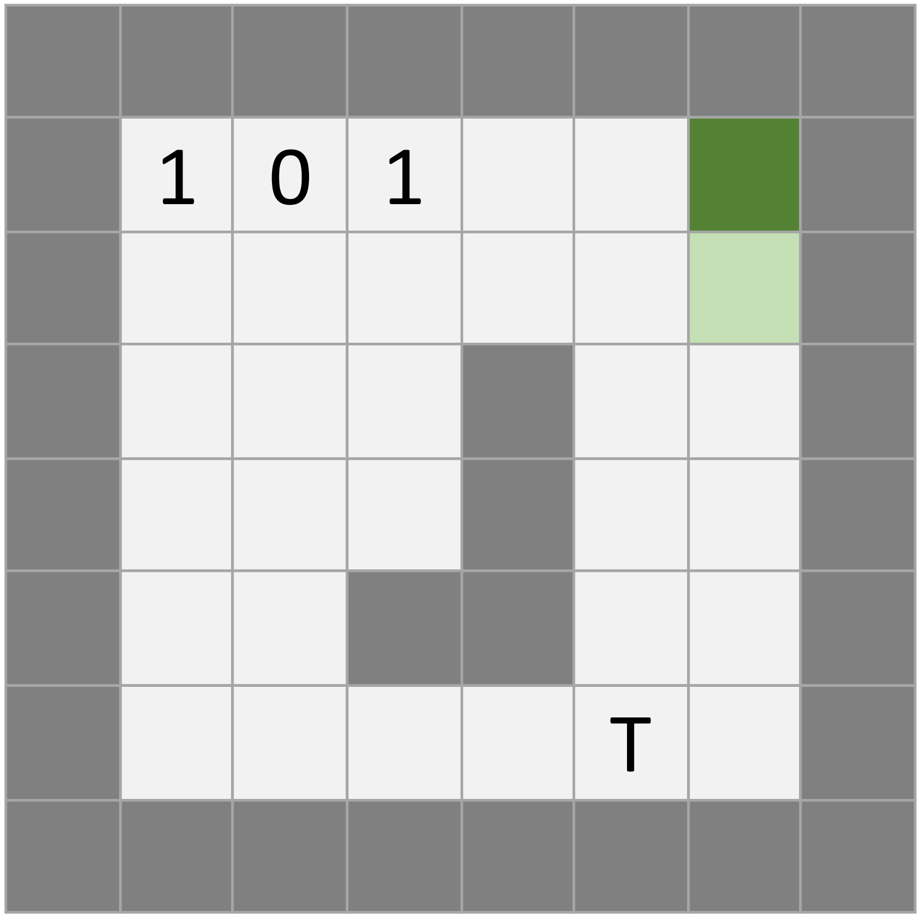
Slow Version

Path Length
`i = 1`



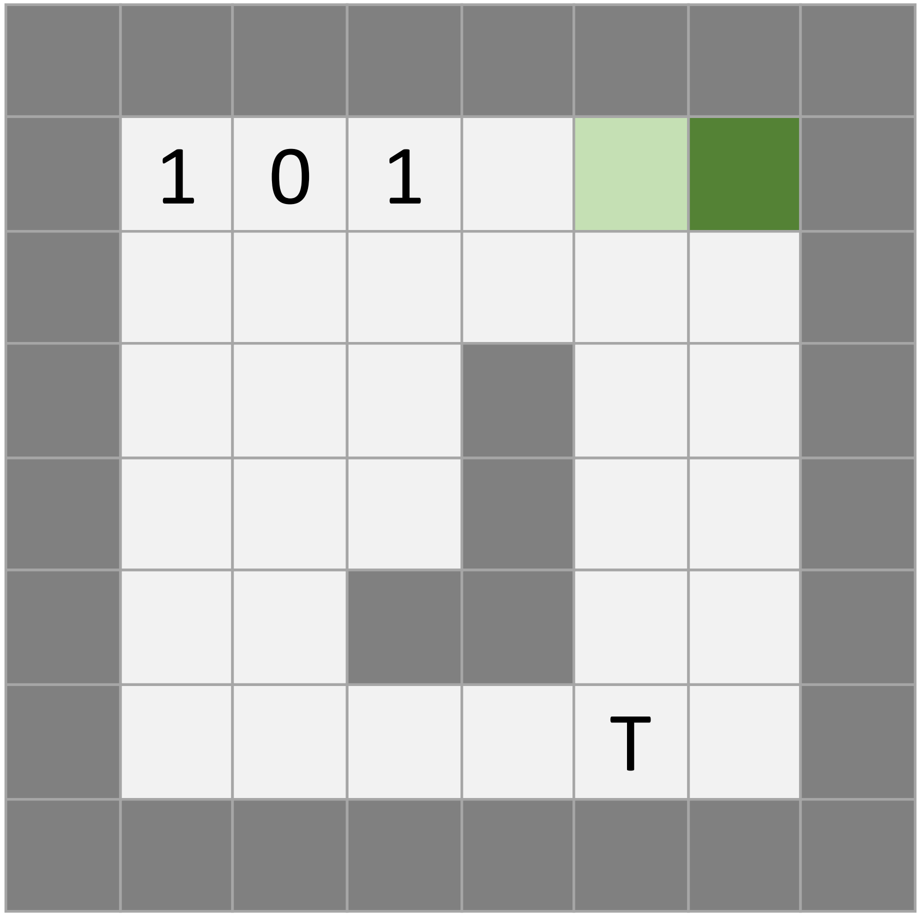
Slow Version

Path Length
`i = 1`



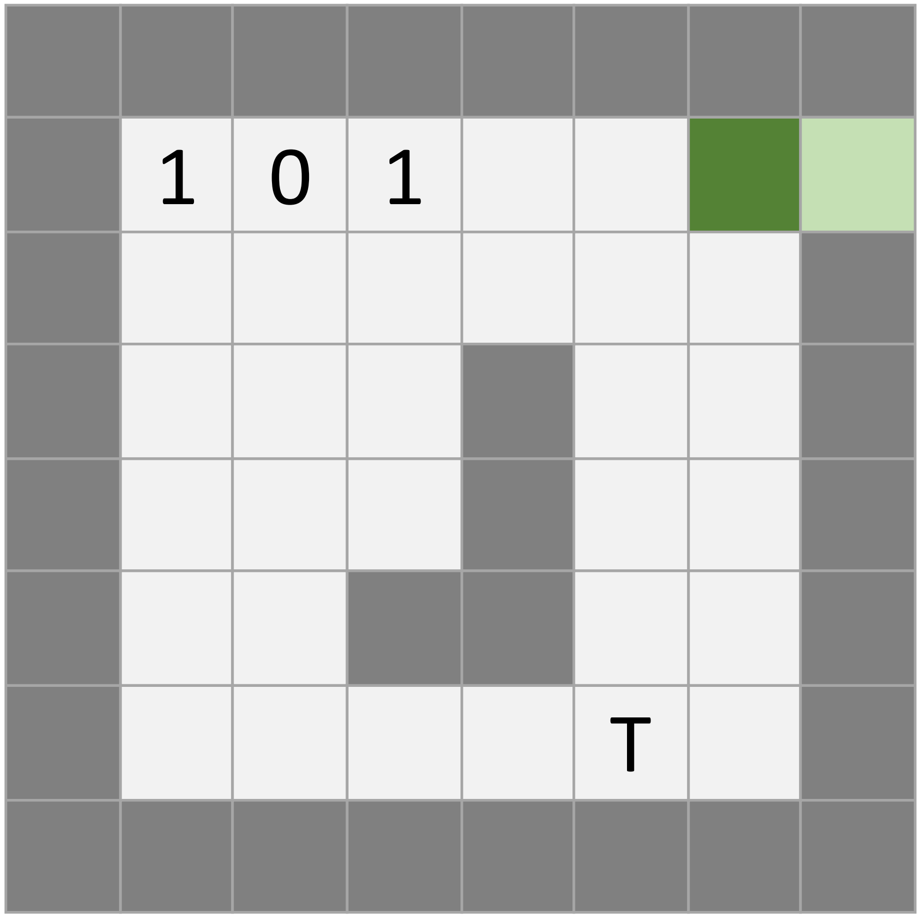
Slow Version

Path Length
`i = 1`



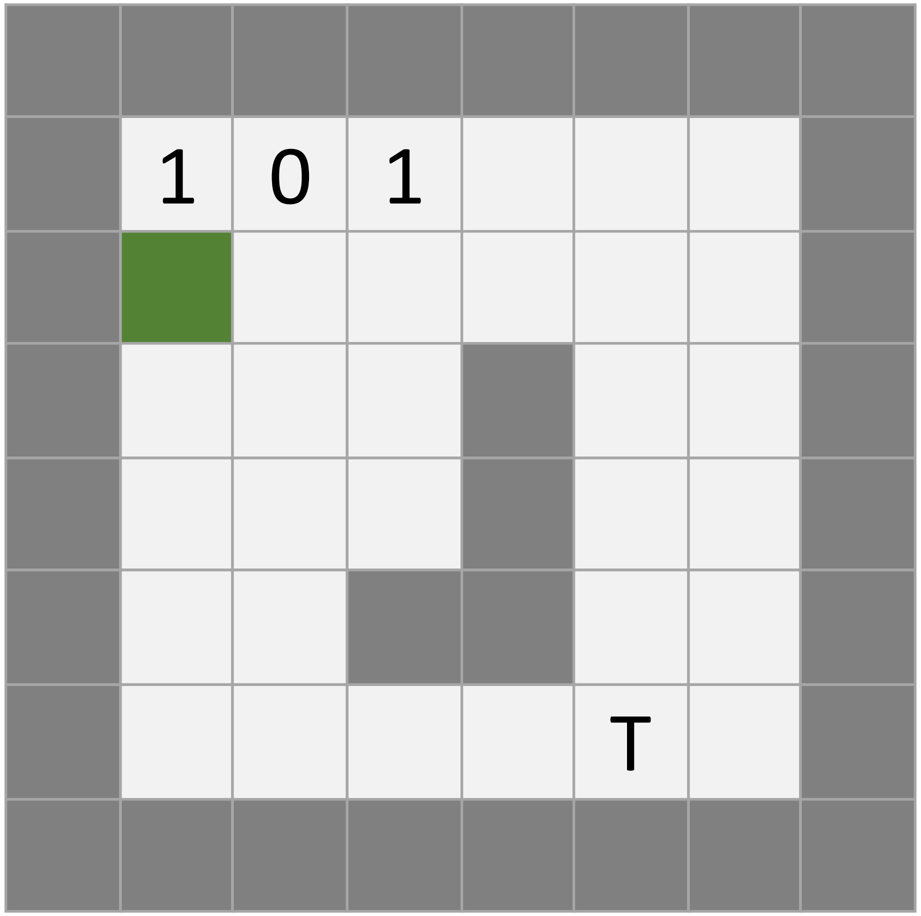
Slow Version

Path Length
`i = 1`



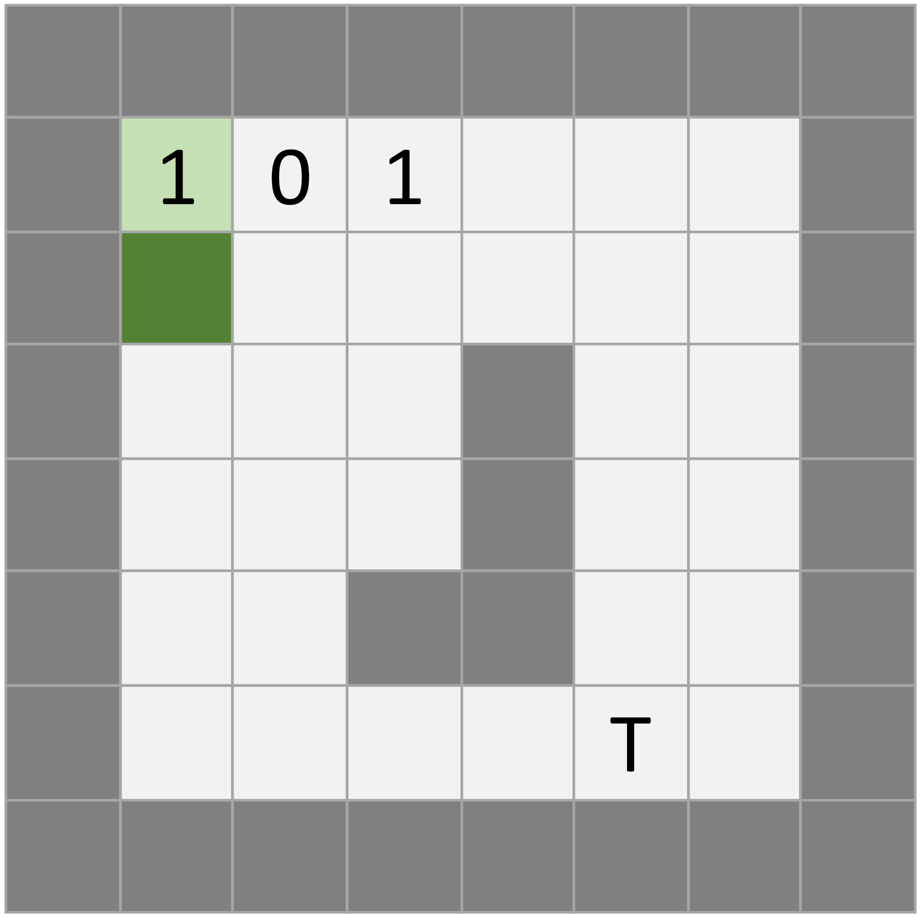
Slow Version

Path Length
`i = 1`



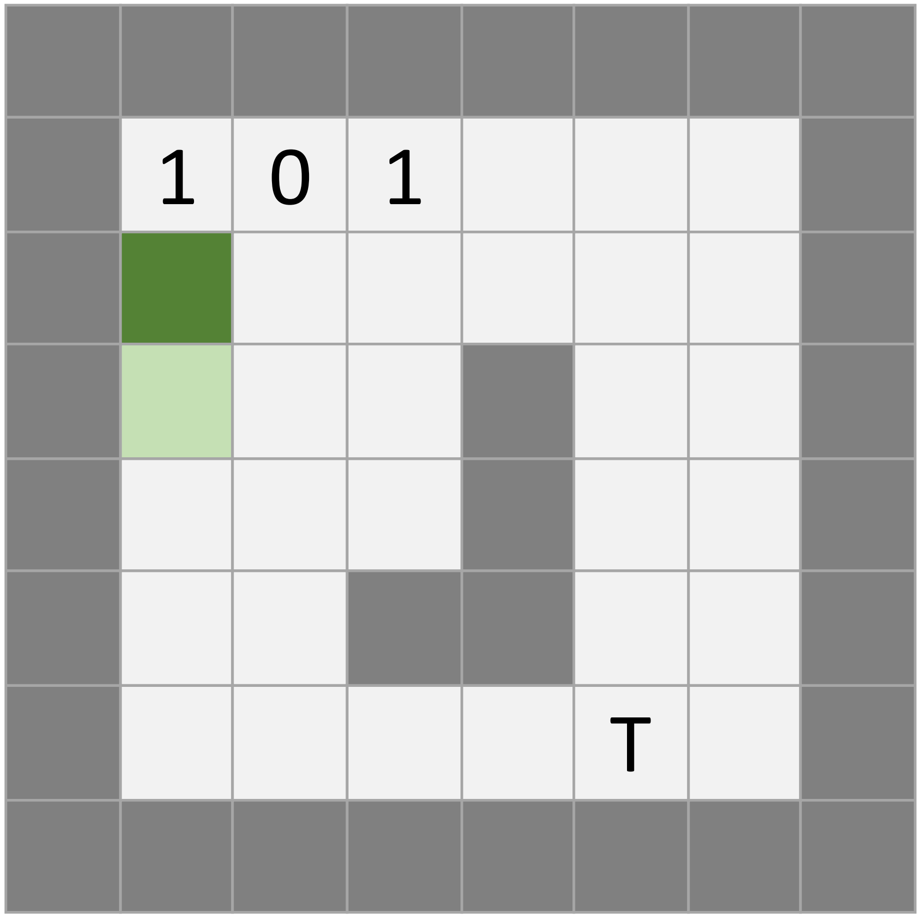
Slow Version

Path Length
`i = 1`



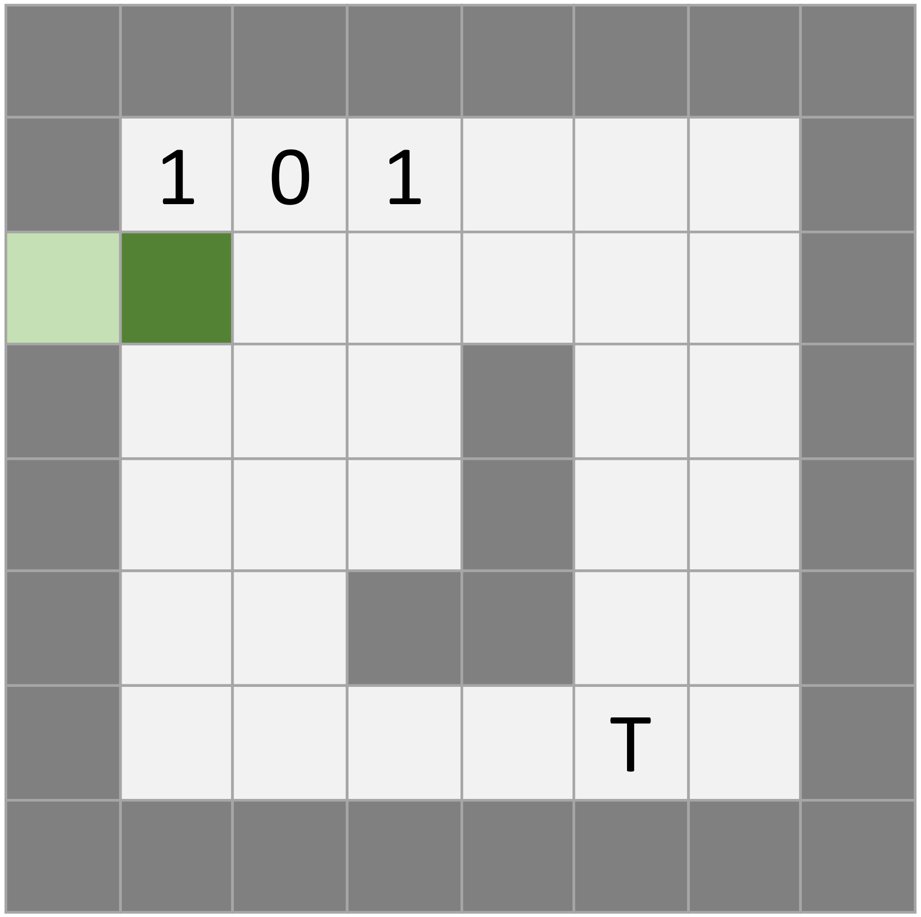
Slow Version

Path Length
`i = 1`



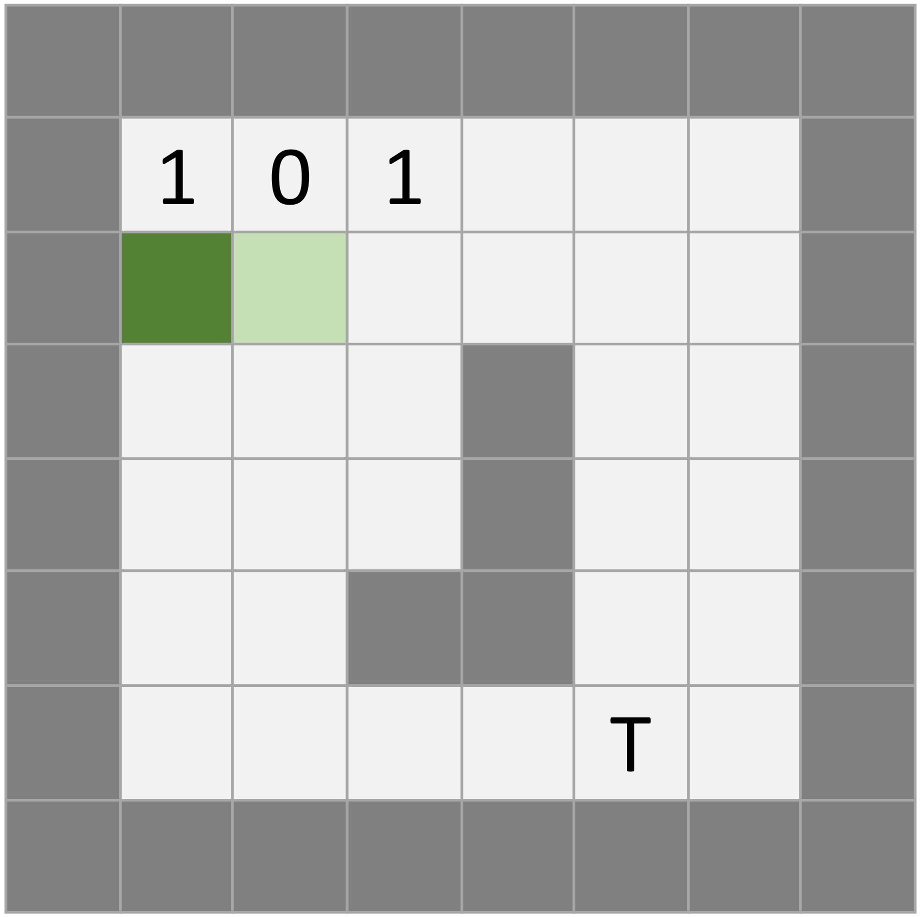
Slow Version

Path Length
`i = 1`



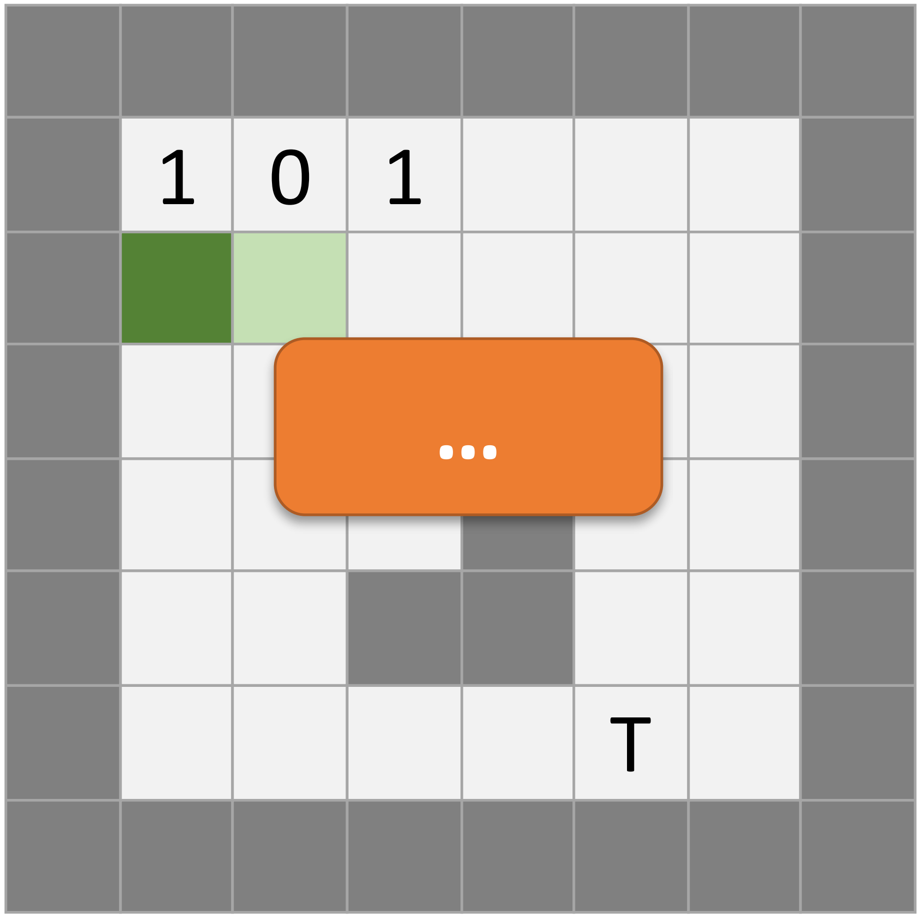
Slow Version

Path Length
`i = 1`



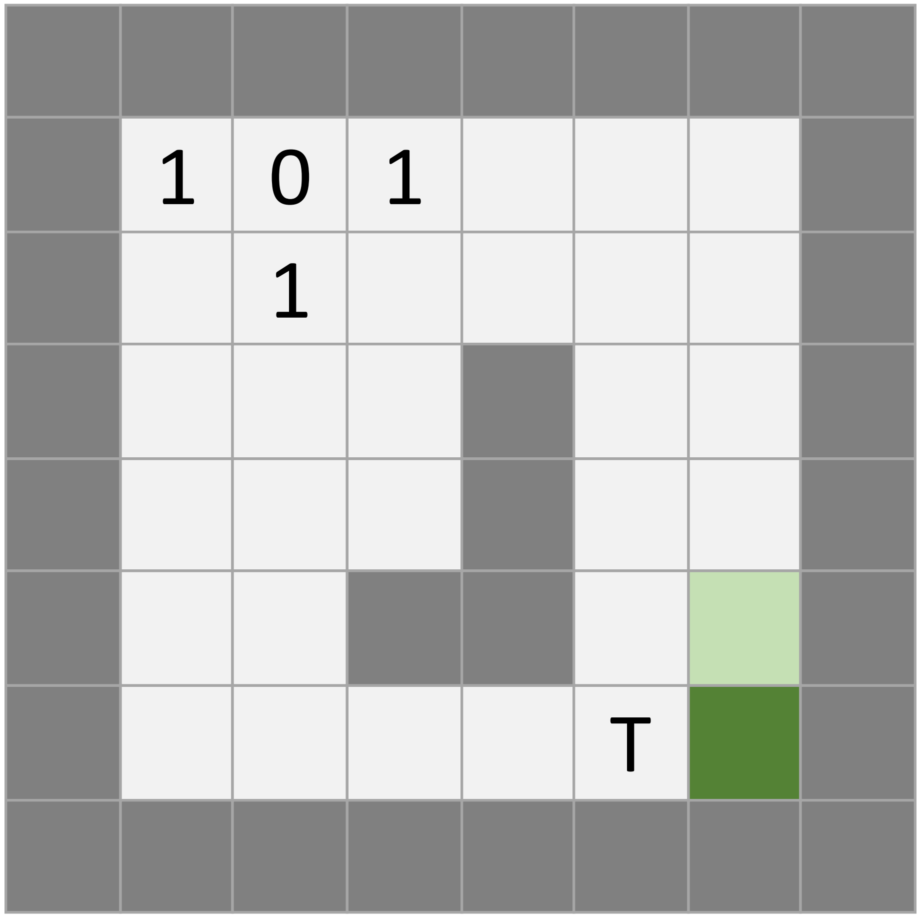
Slow Version

Path Length
`i = 1`



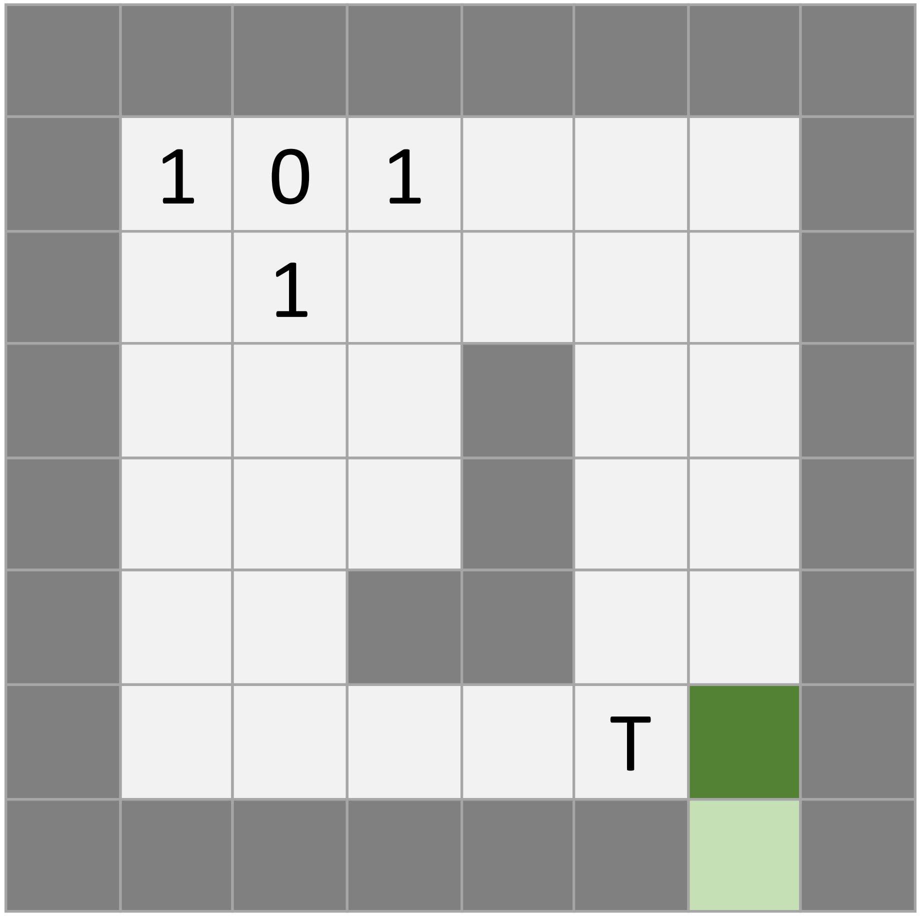
Slow Version

Path Length
`i = 1`



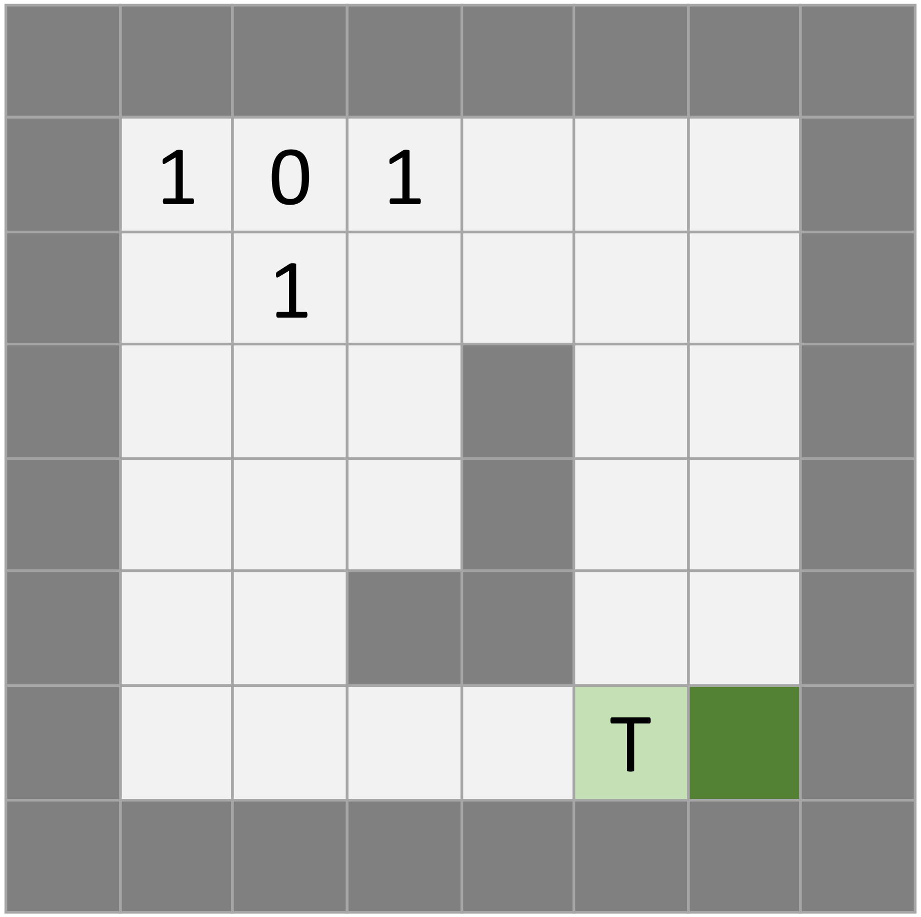
Slow Version

Path Length
`i = 1`



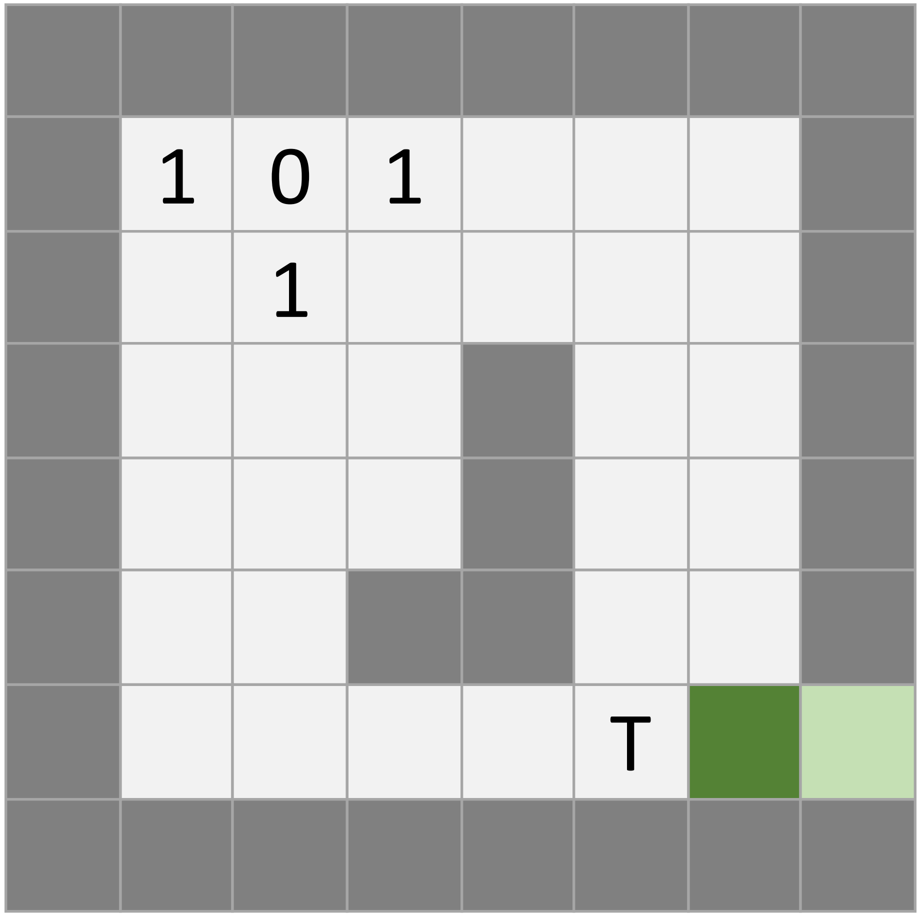
Slow Version

Path Length
`i = 1`



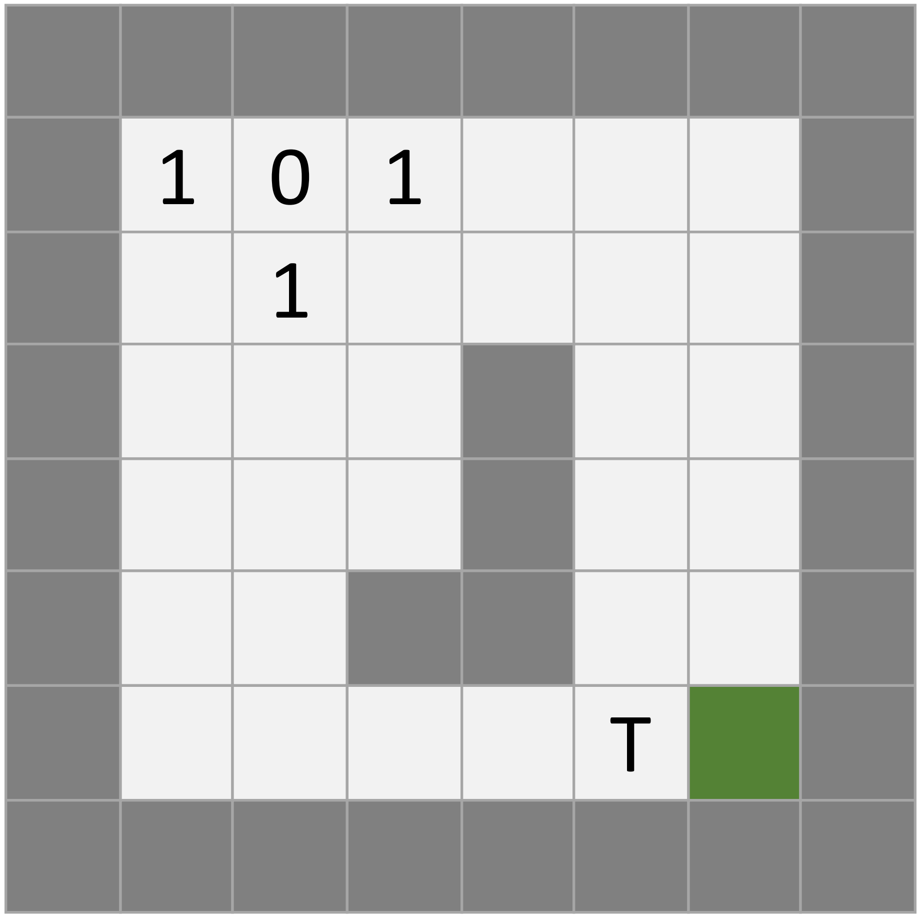
Slow Version

Path Length
`i = 1`

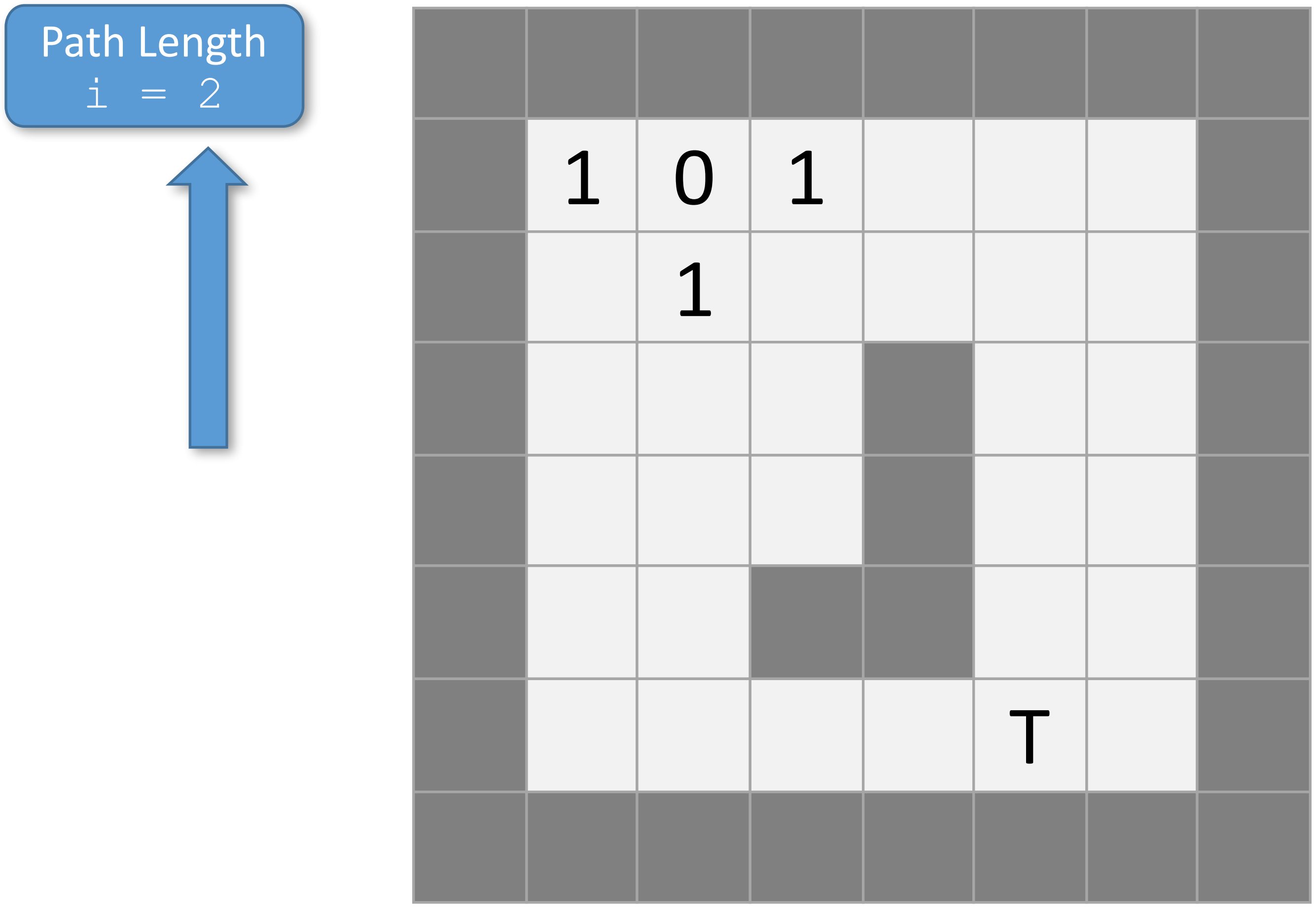


Slow Version

Path Length
`i = 1`

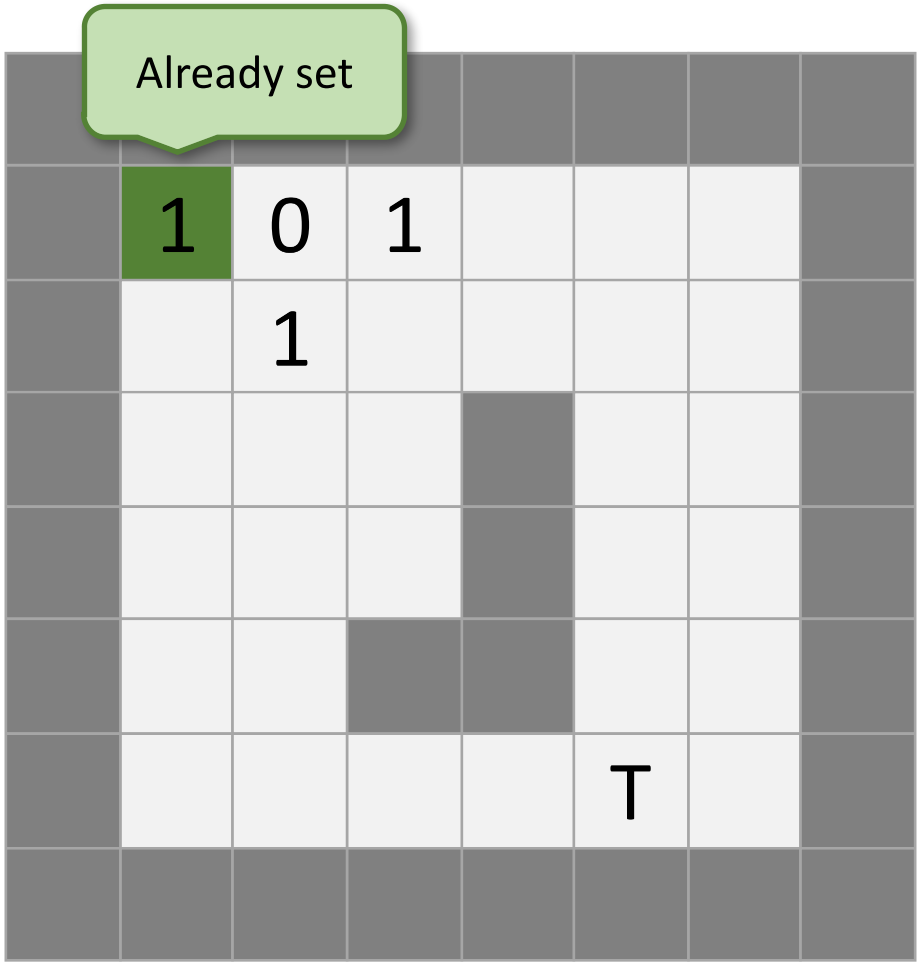


Slow Version



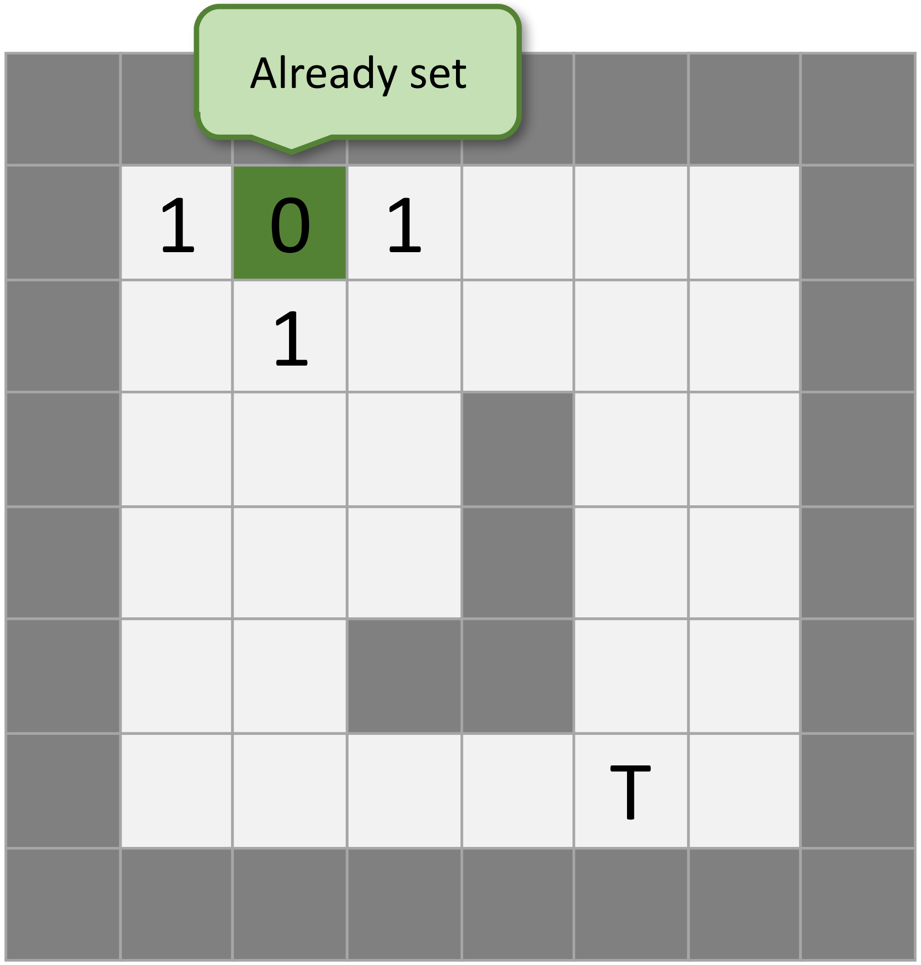
Slow Version

Path Length
`i = 2`



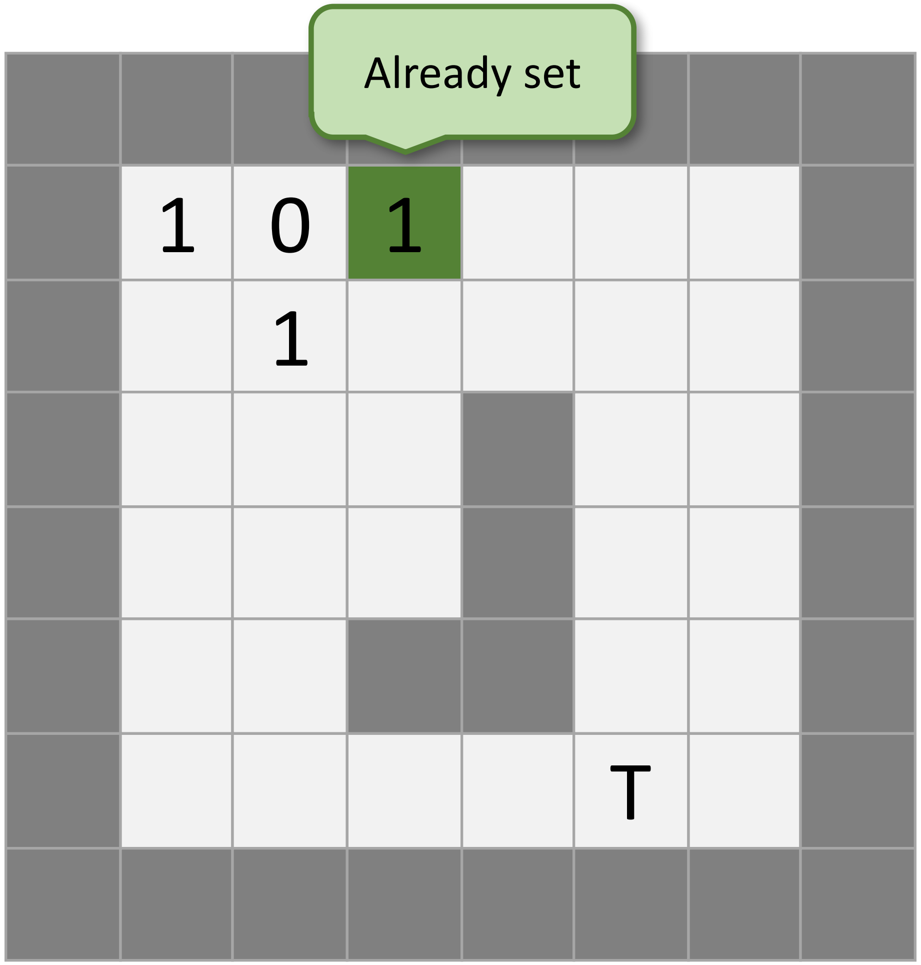
Slow Version

Path Length
`i = 2`



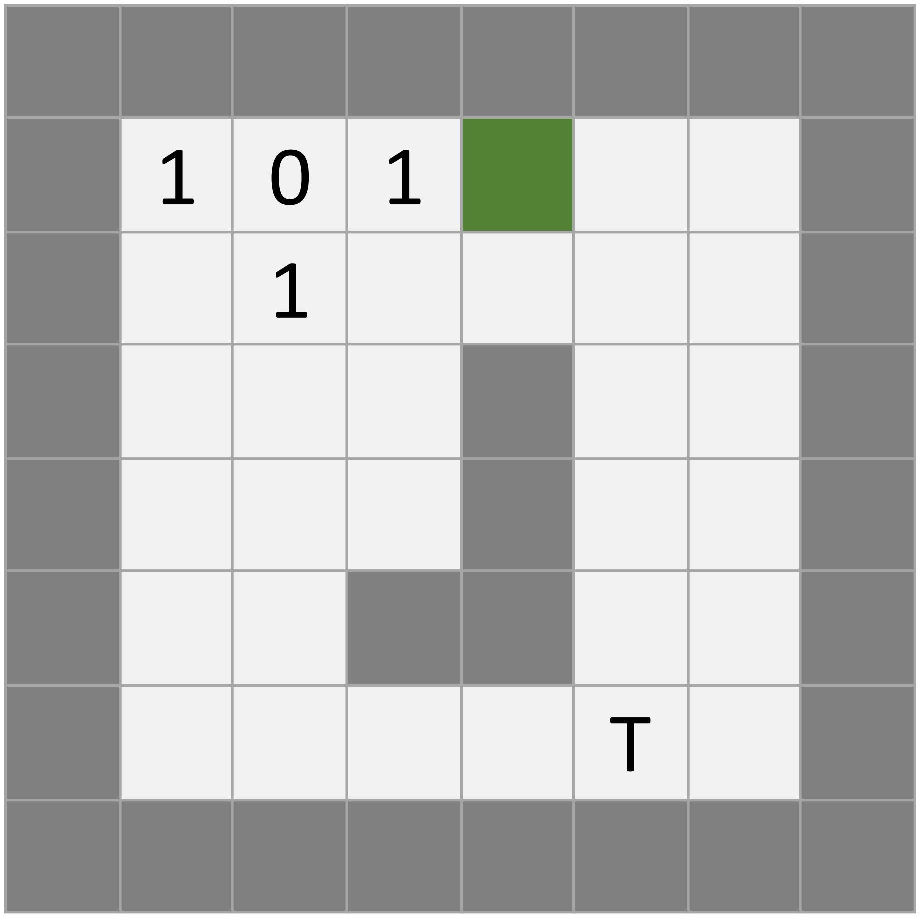
Slow Version

Path Length
`i = 2`



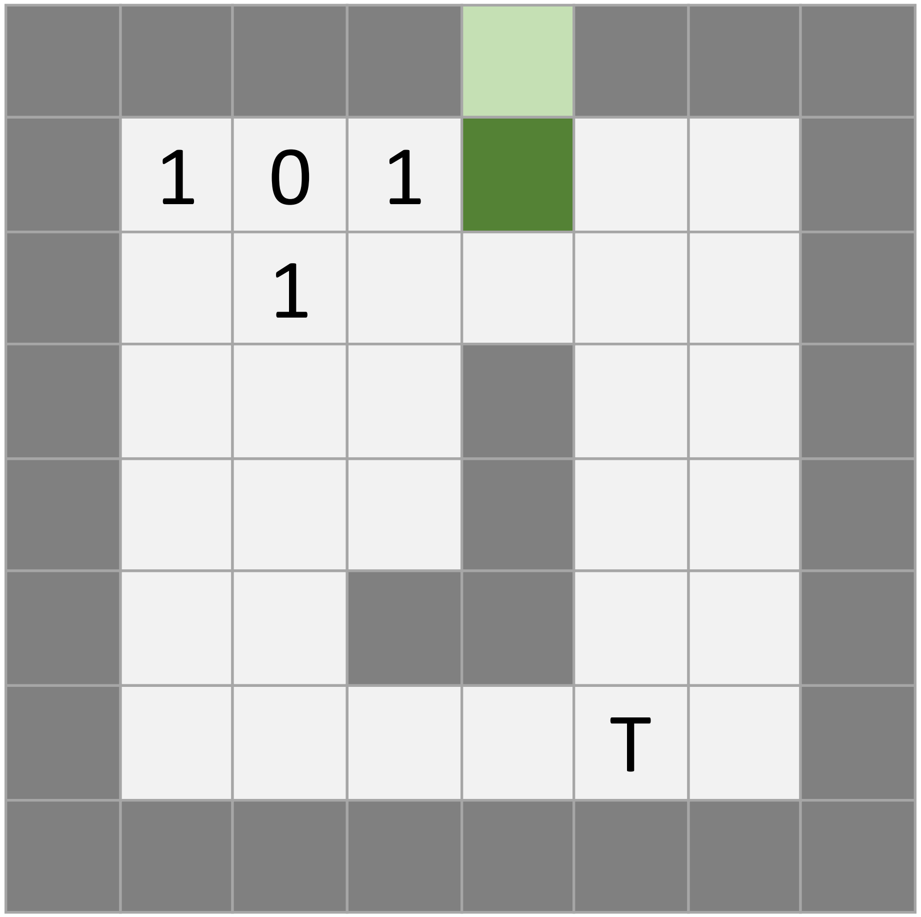
Slow Version

Path Length
`i = 2`



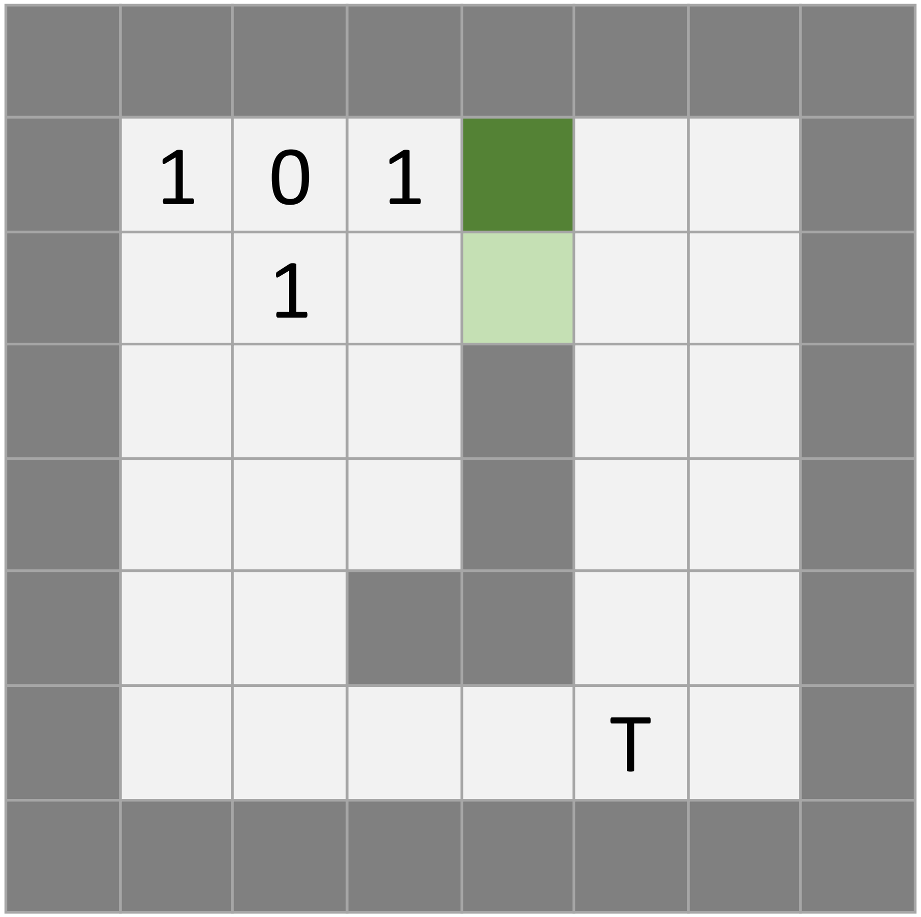
Slow Version

Path Length
`i = 2`

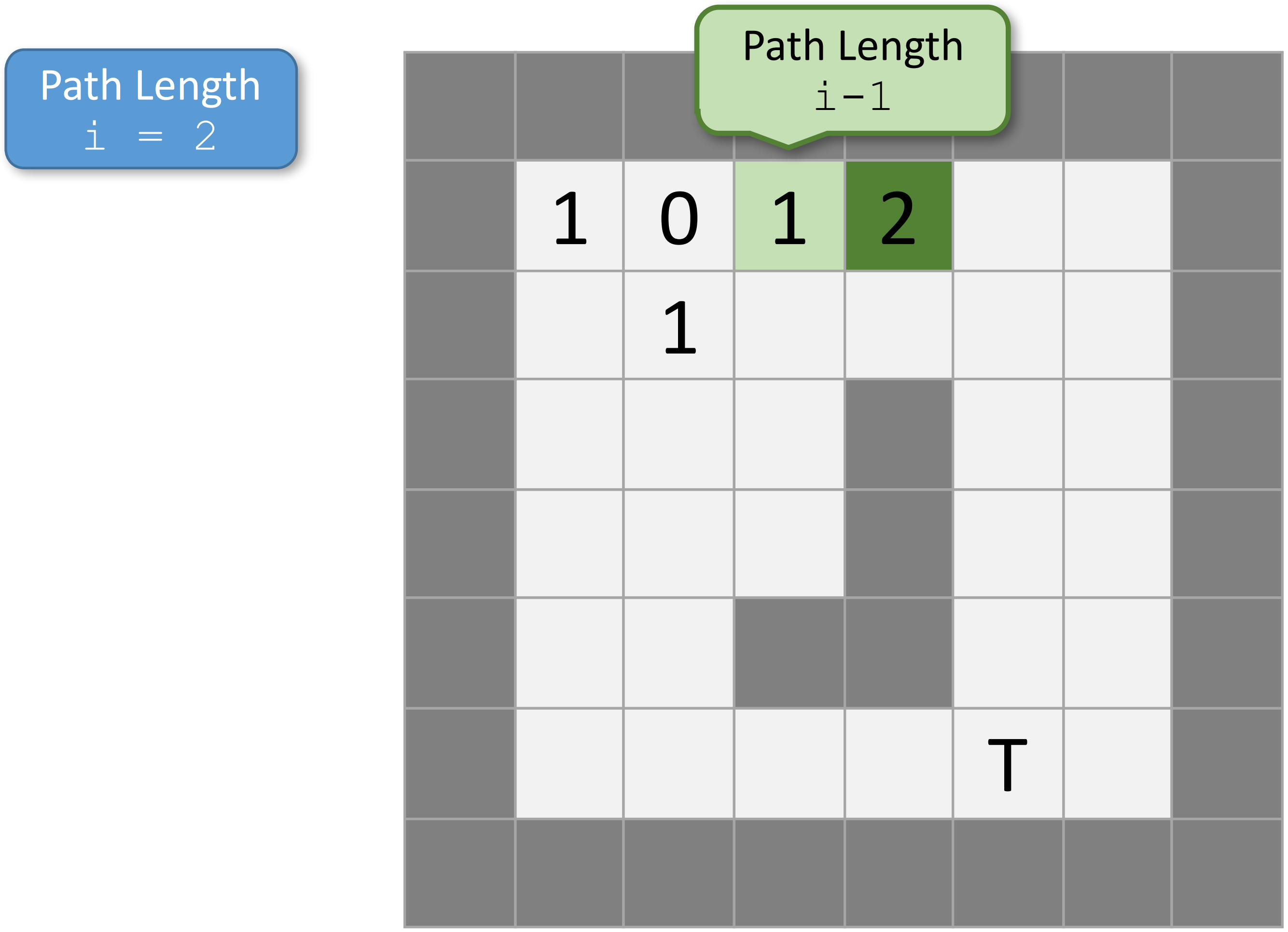


Slow Version

Path Length
`i = 2`

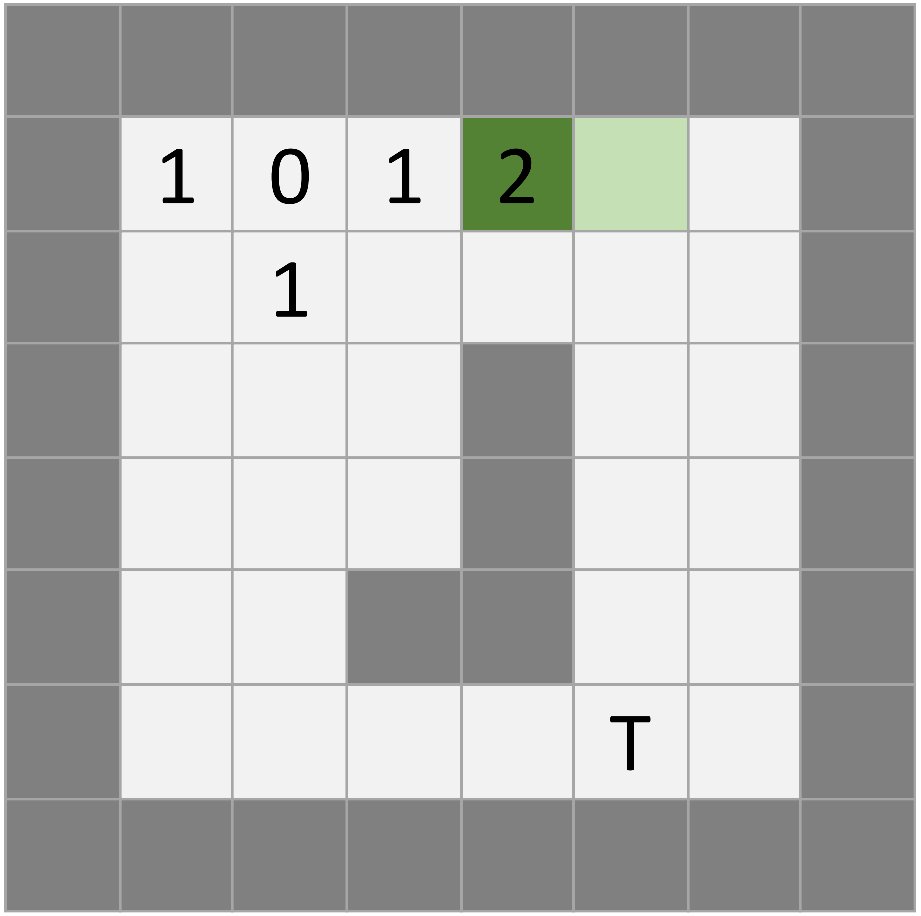


Slow Version



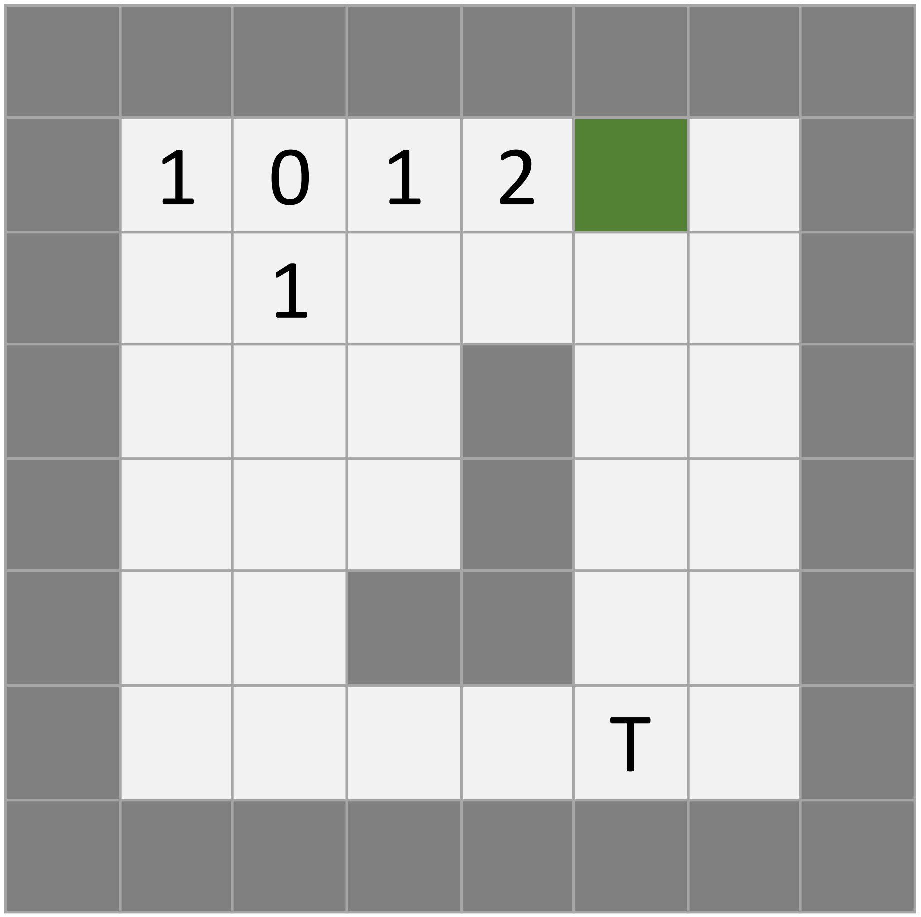
Slow Version

Path Length
`i = 2`



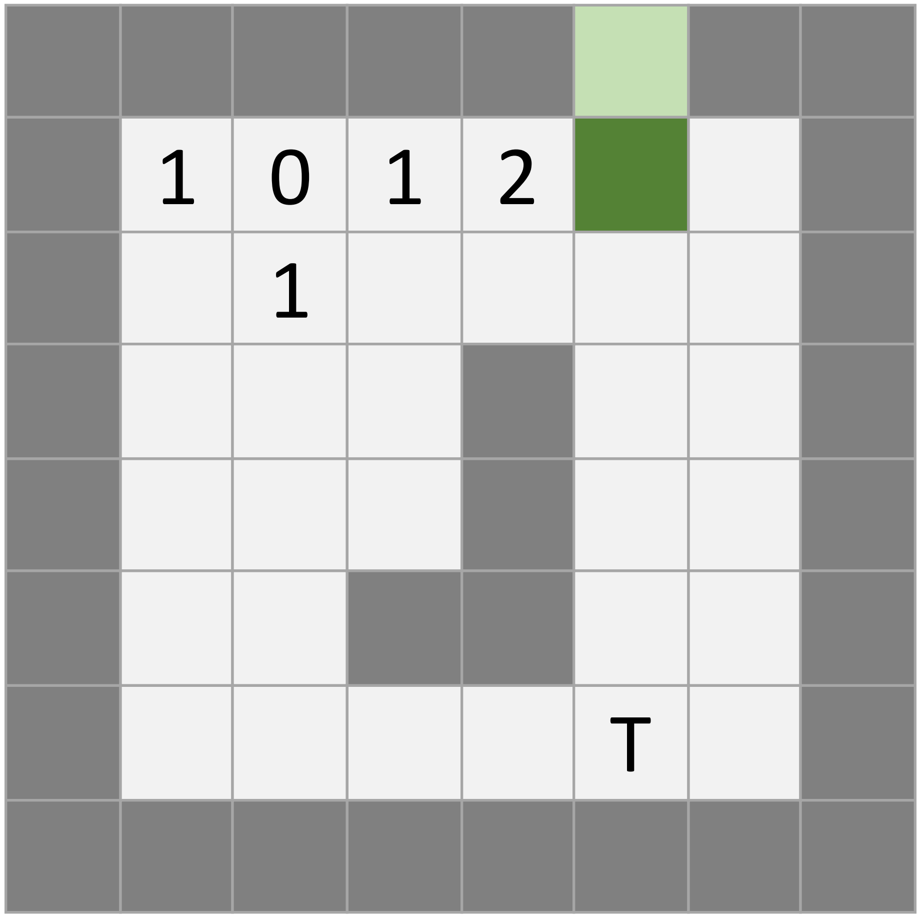
Slow Version

Path Length
`i = 2`



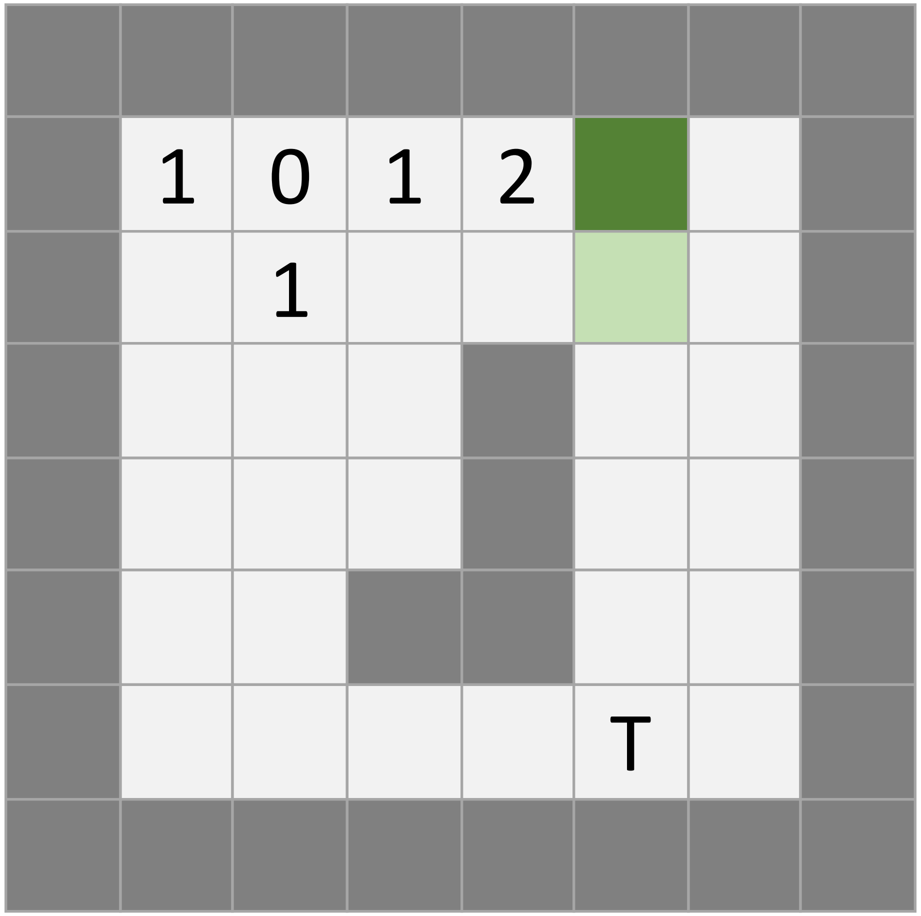
Slow Version

Path Length
`i = 2`



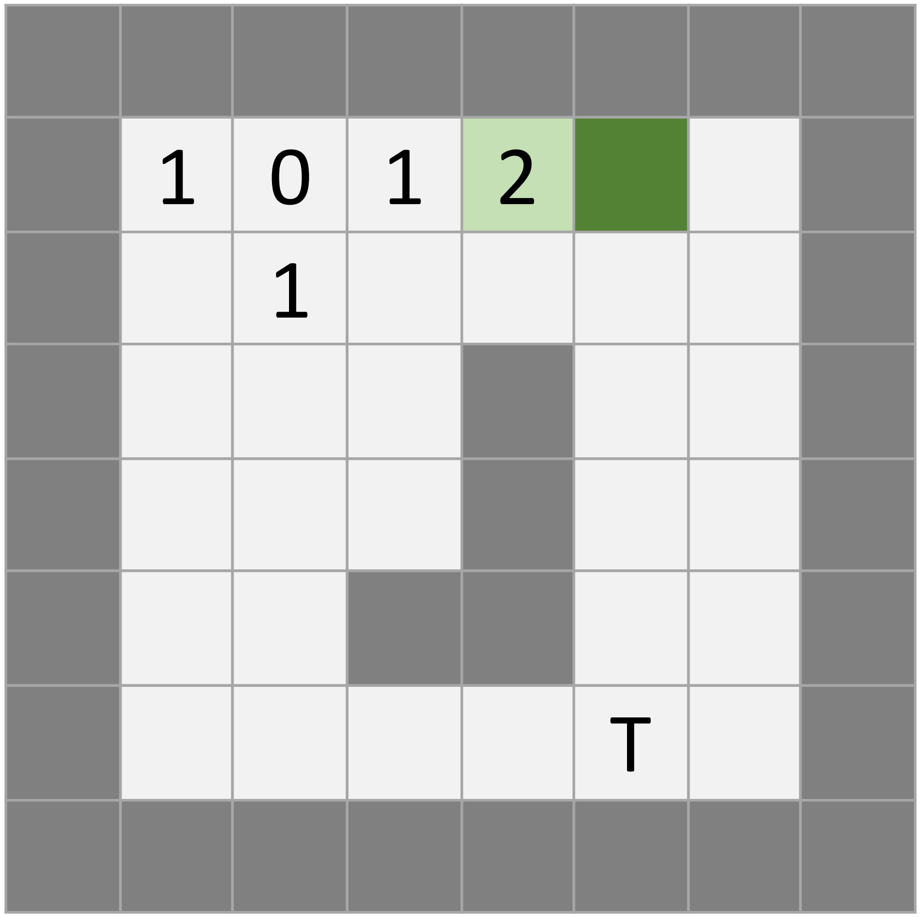
Slow Version

Path Length
`i = 2`



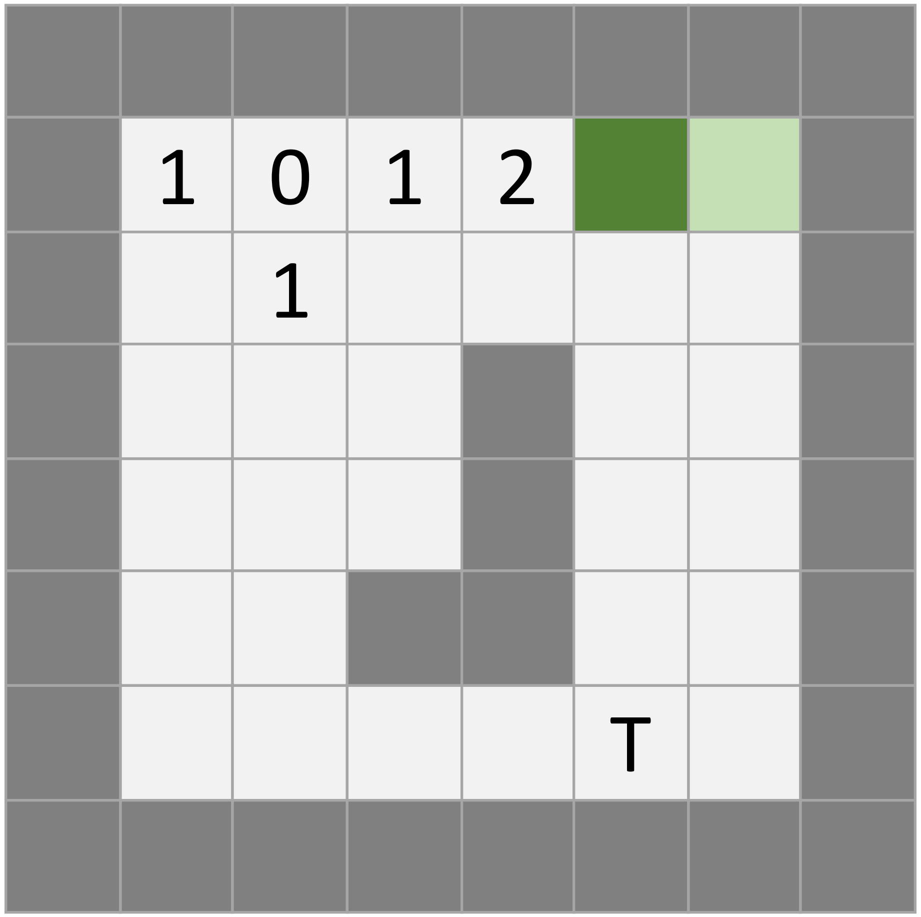
Slow Version

Path Length
i = 2



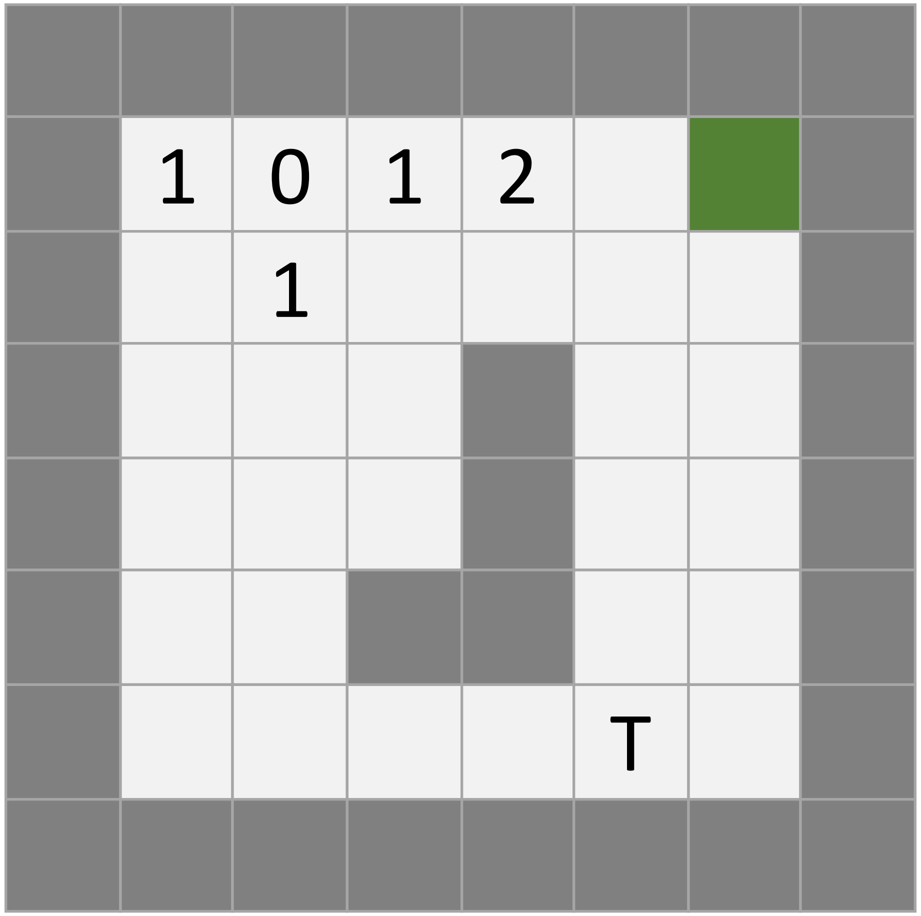
Slow Version

Path Length
`i = 2`



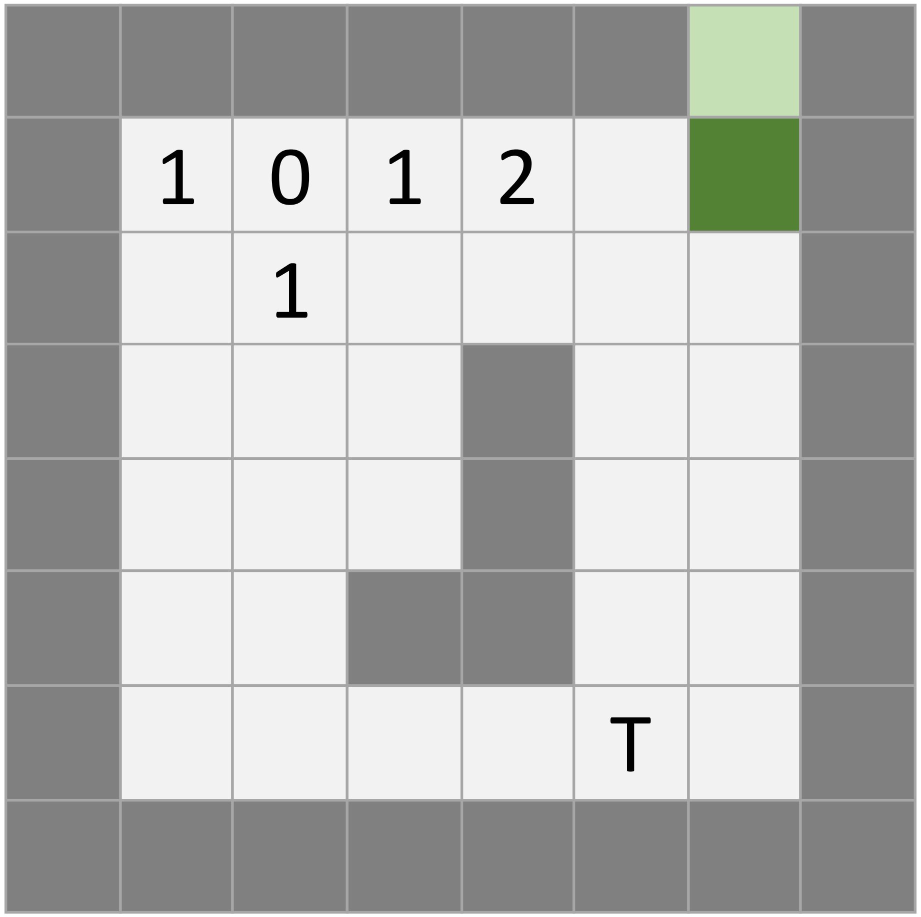
Slow Version

Path Length
`i = 2`



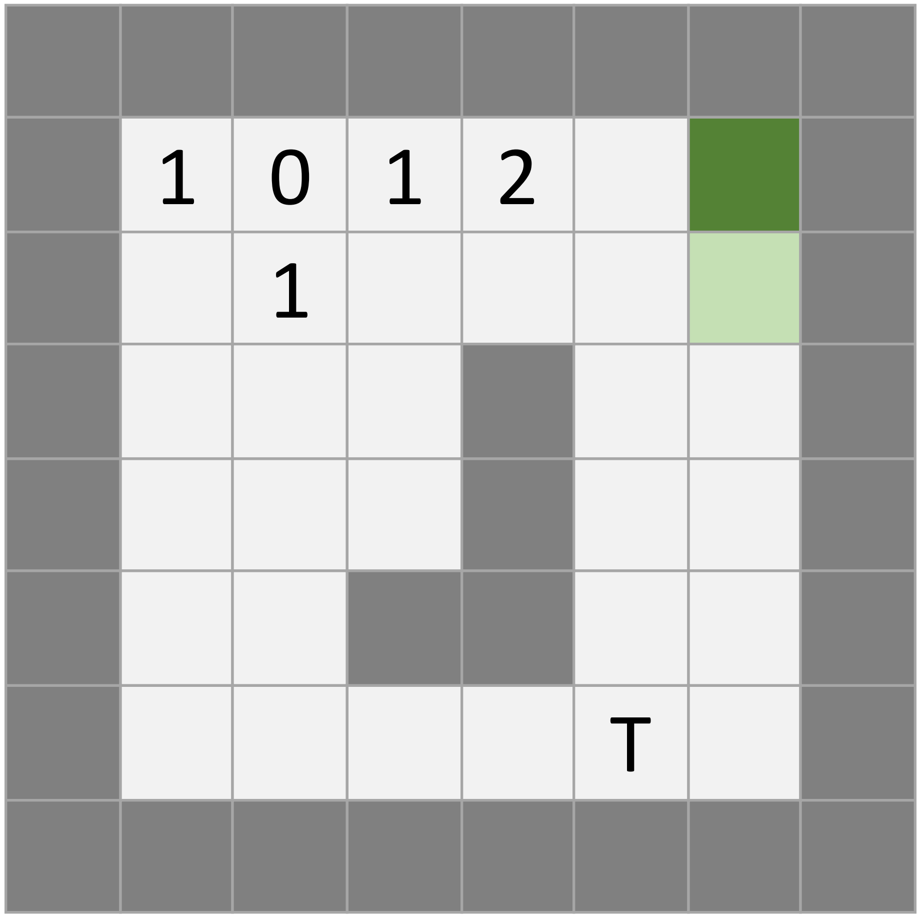
Slow Version

Path Length
`i = 2`



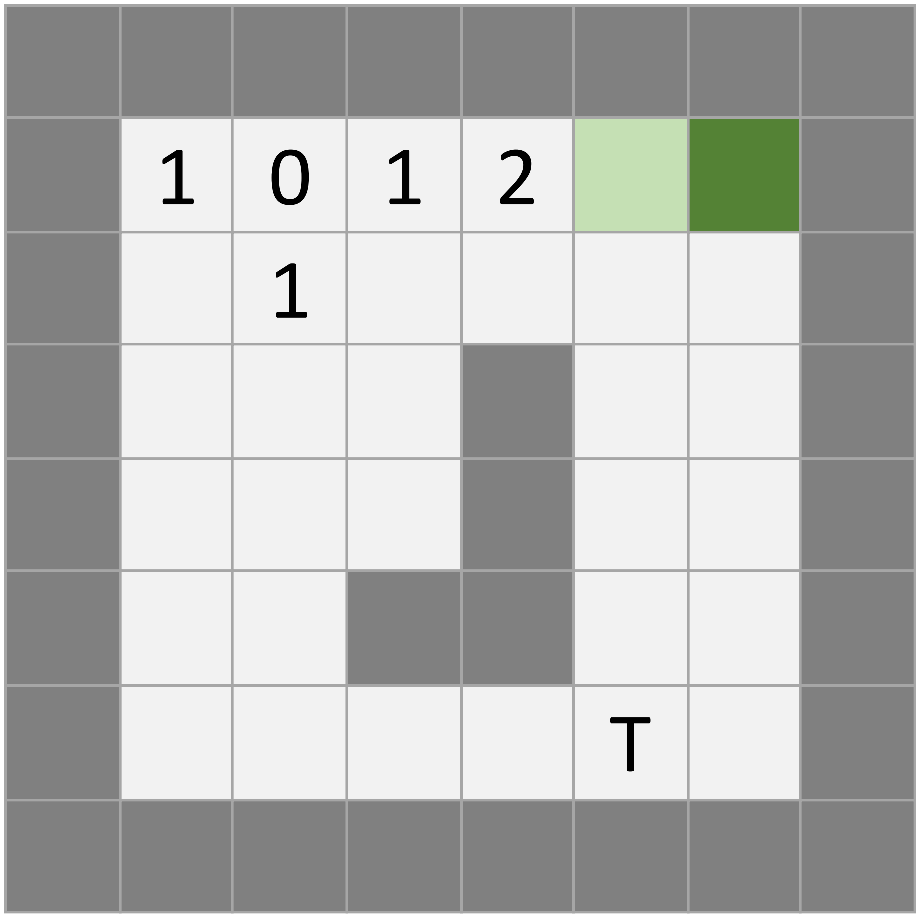
Slow Version

Path Length
`i = 2`



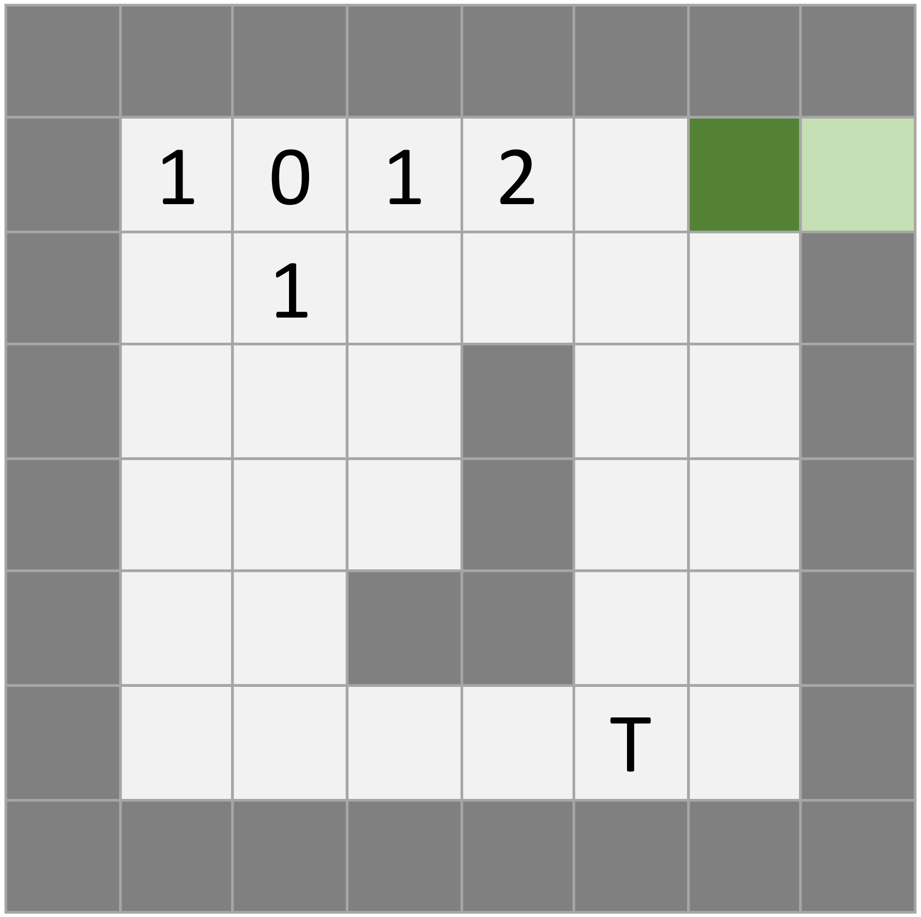
Slow Version

Path Length
`i = 2`



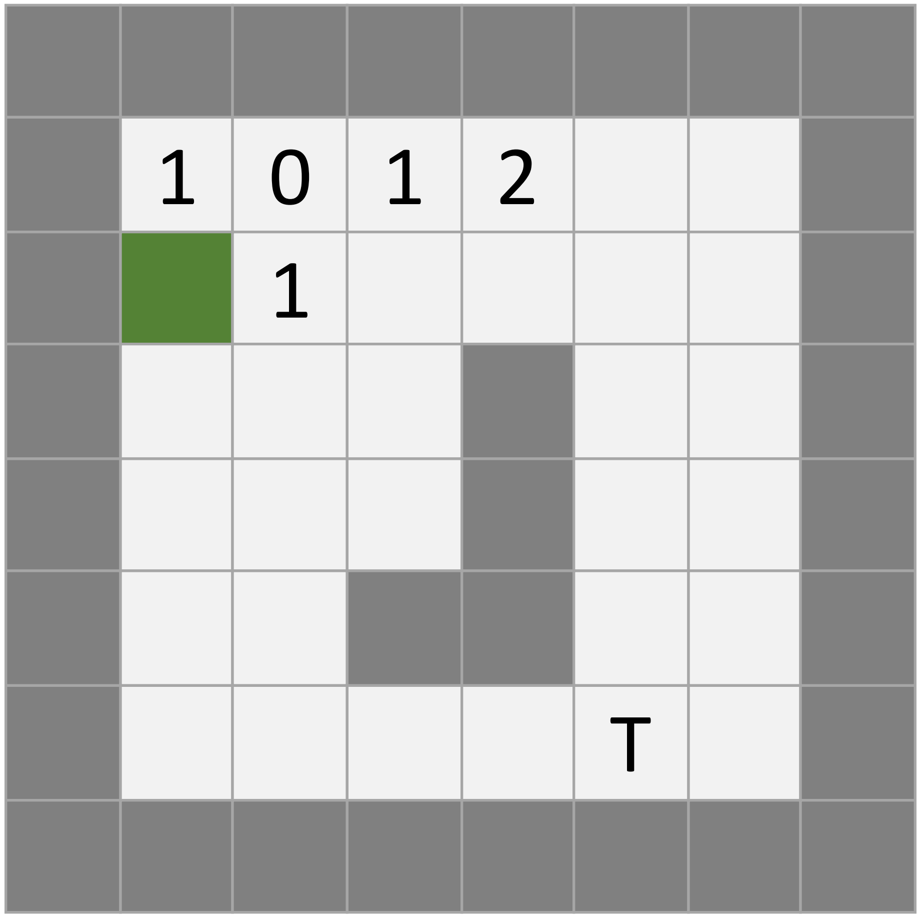
Slow Version

Path Length
`i = 2`

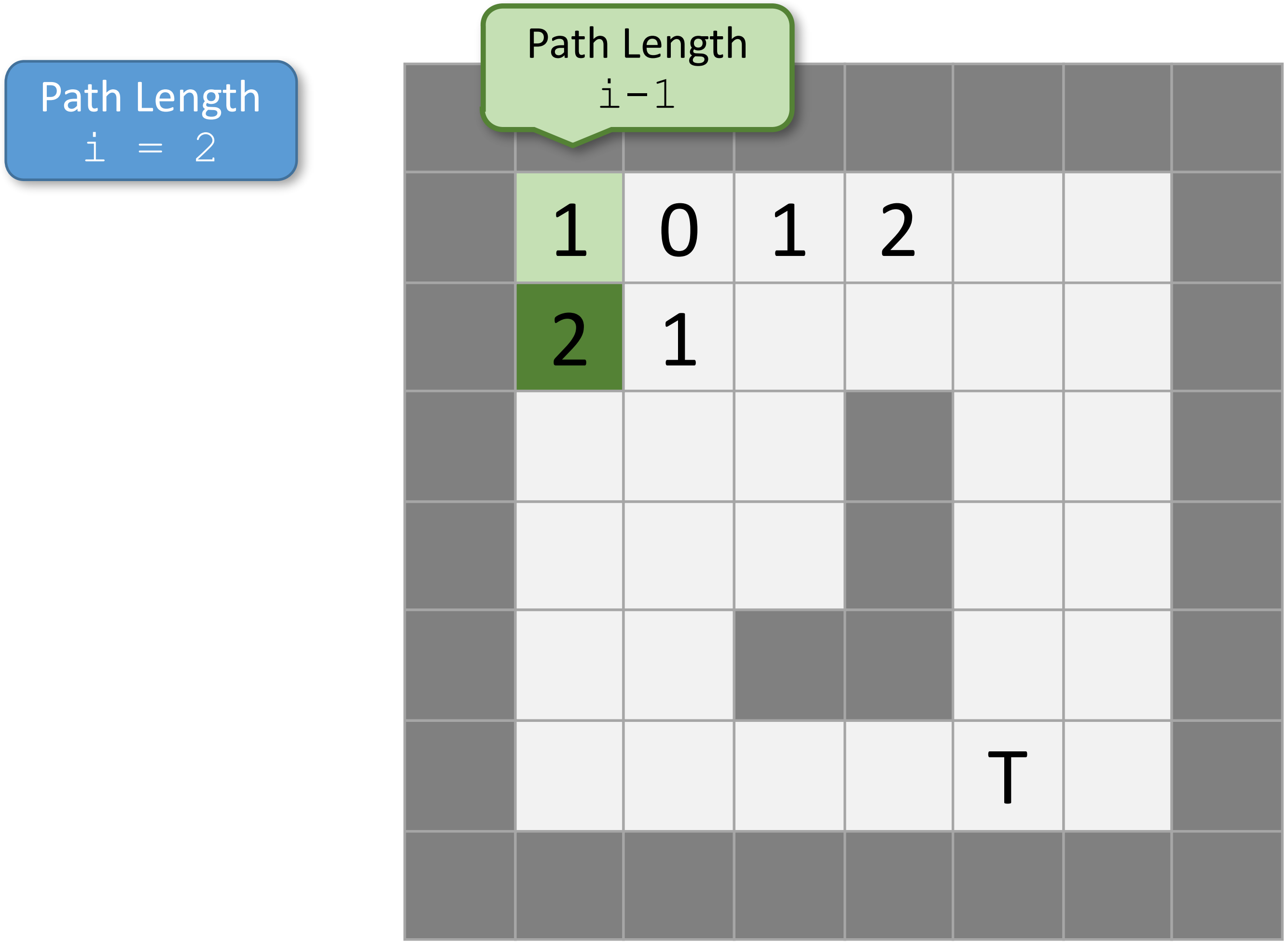


Slow Version

Path Length
`i = 2`

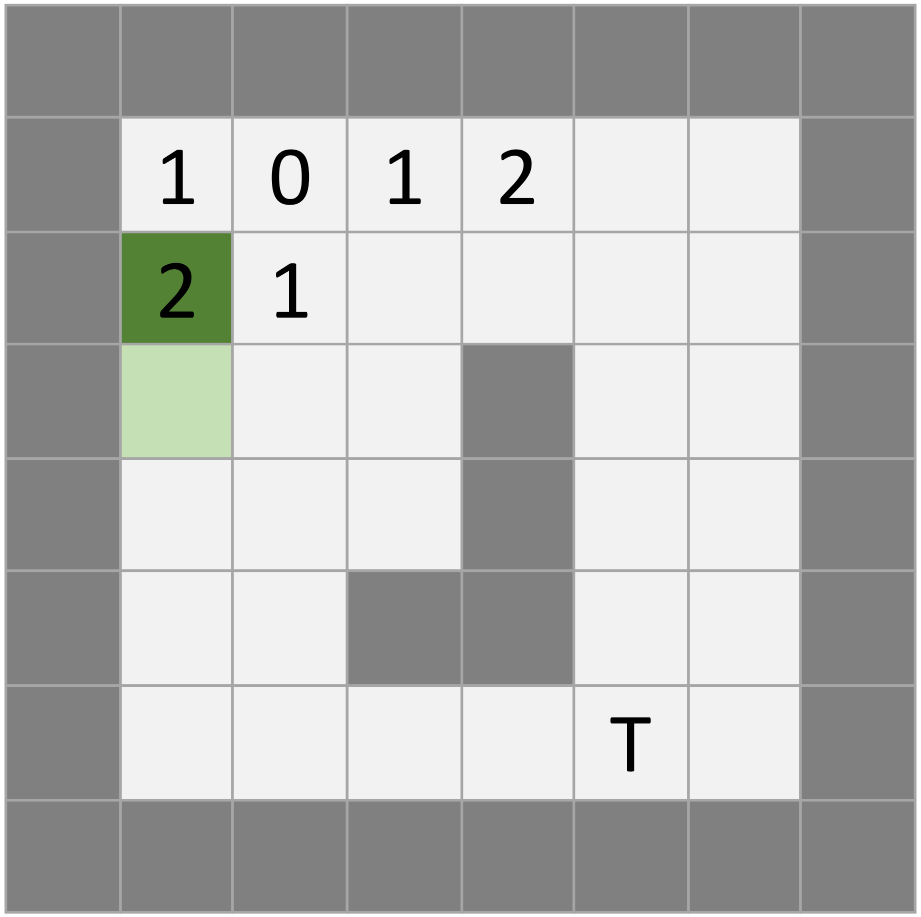


Slow Version



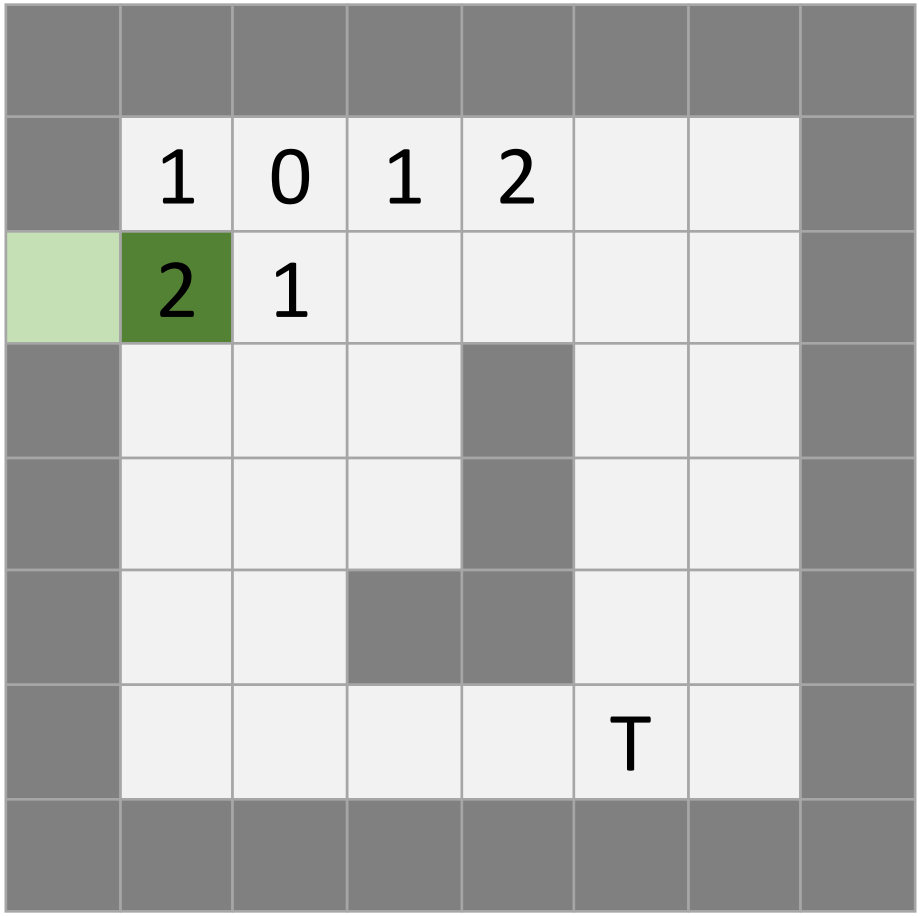
Slow Version

Path Length
i = 2



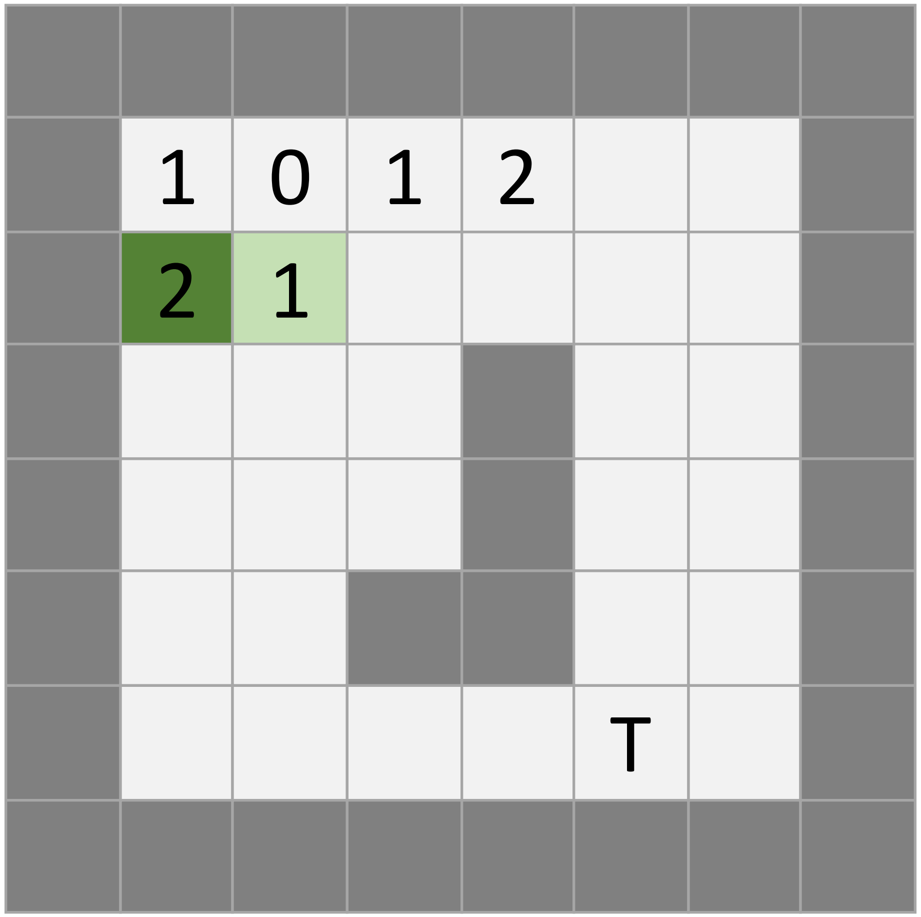
Slow Version

Path Length
`i = 2`



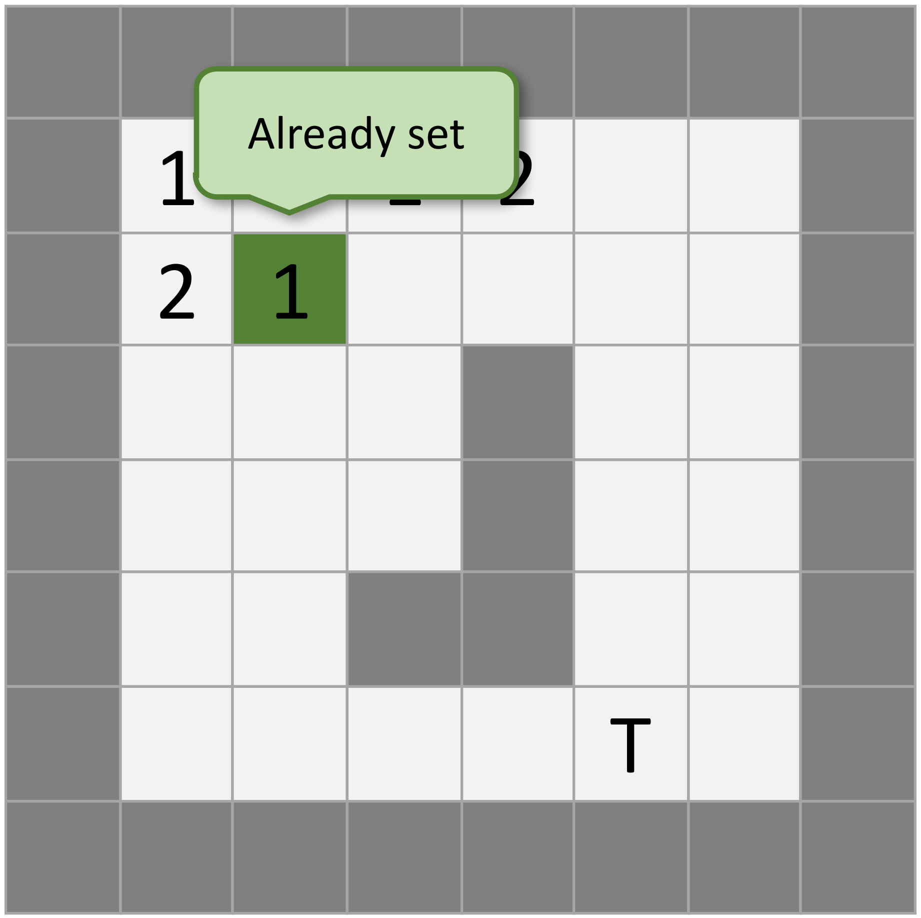
Slow Version

Path Length
`i = 2`



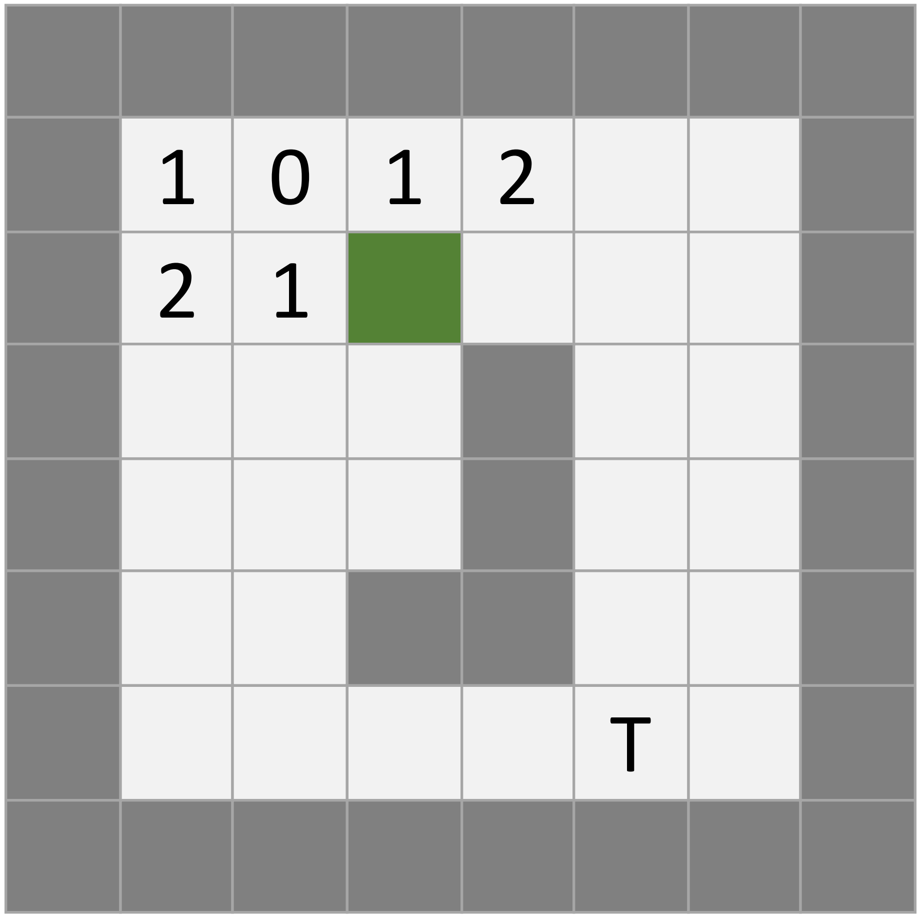
Slow Version

Path Length
`i = 2`



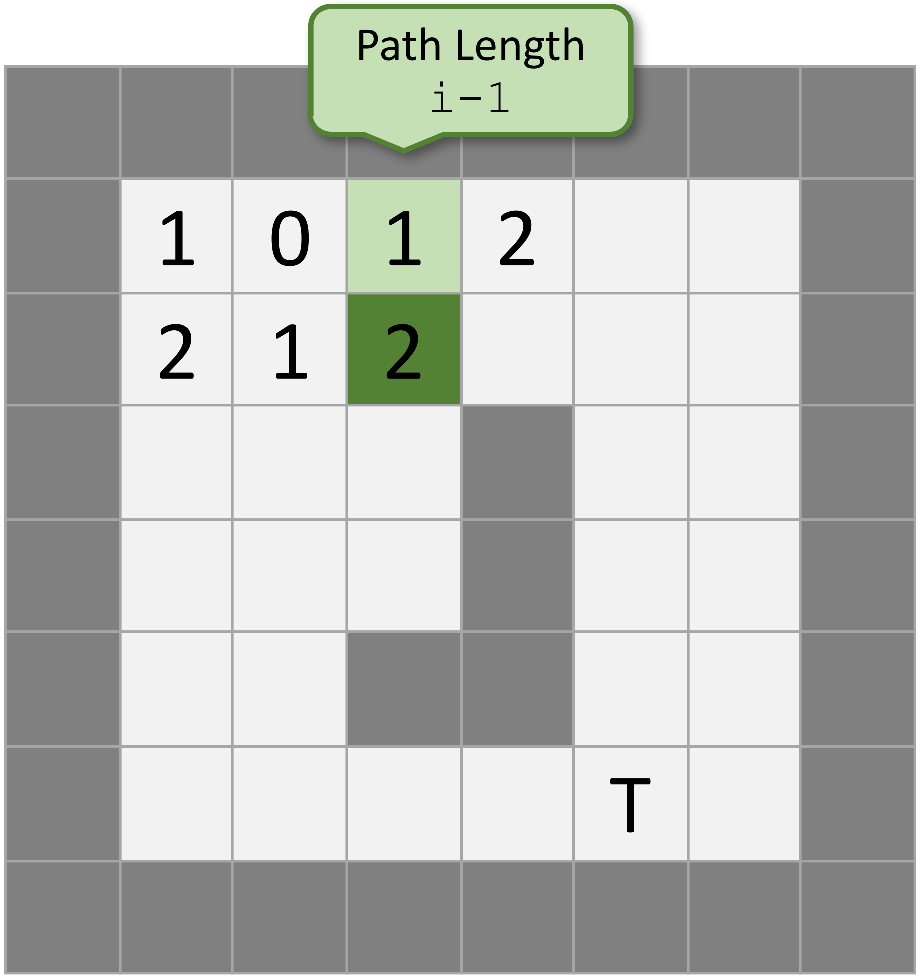
Slow Version

Path Length
`i = 2`



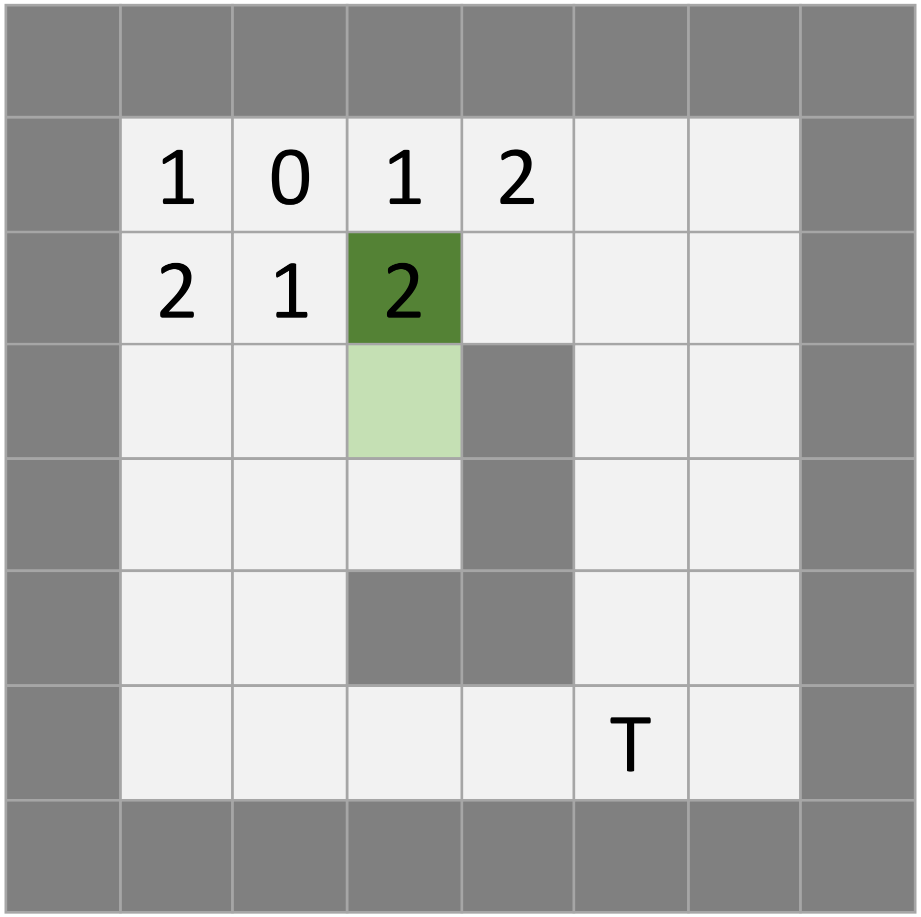
Slow Version

Path Length
`i = 2`



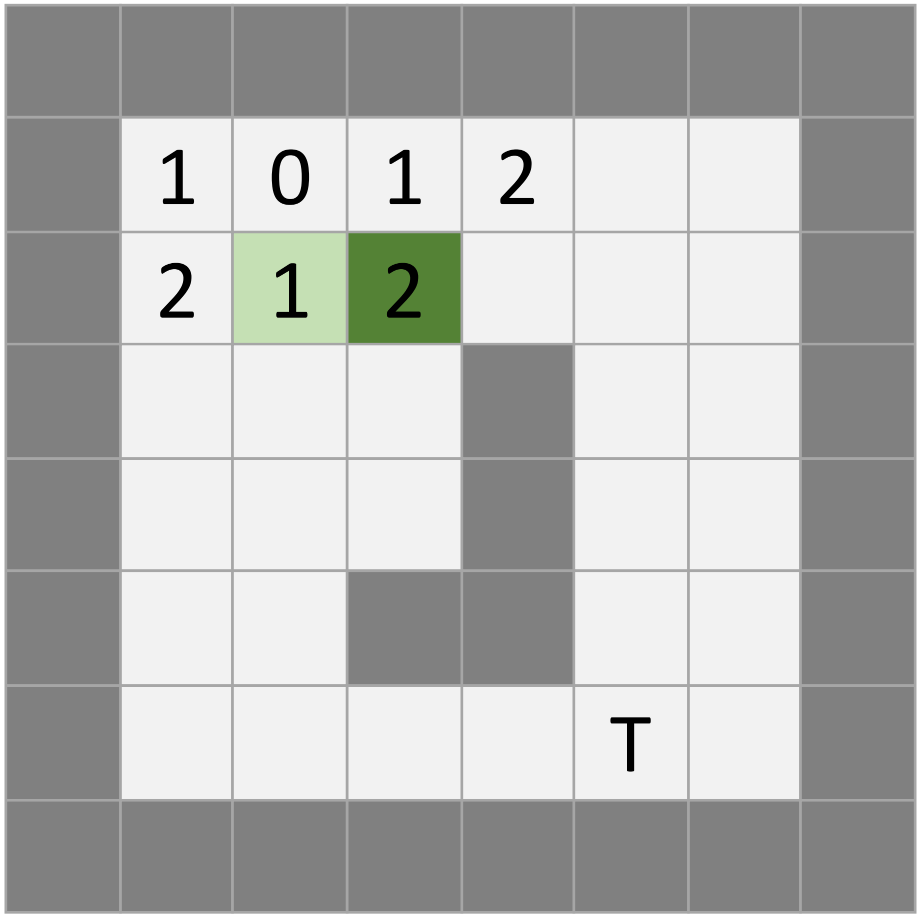
Slow Version

Path Length
`i = 2`



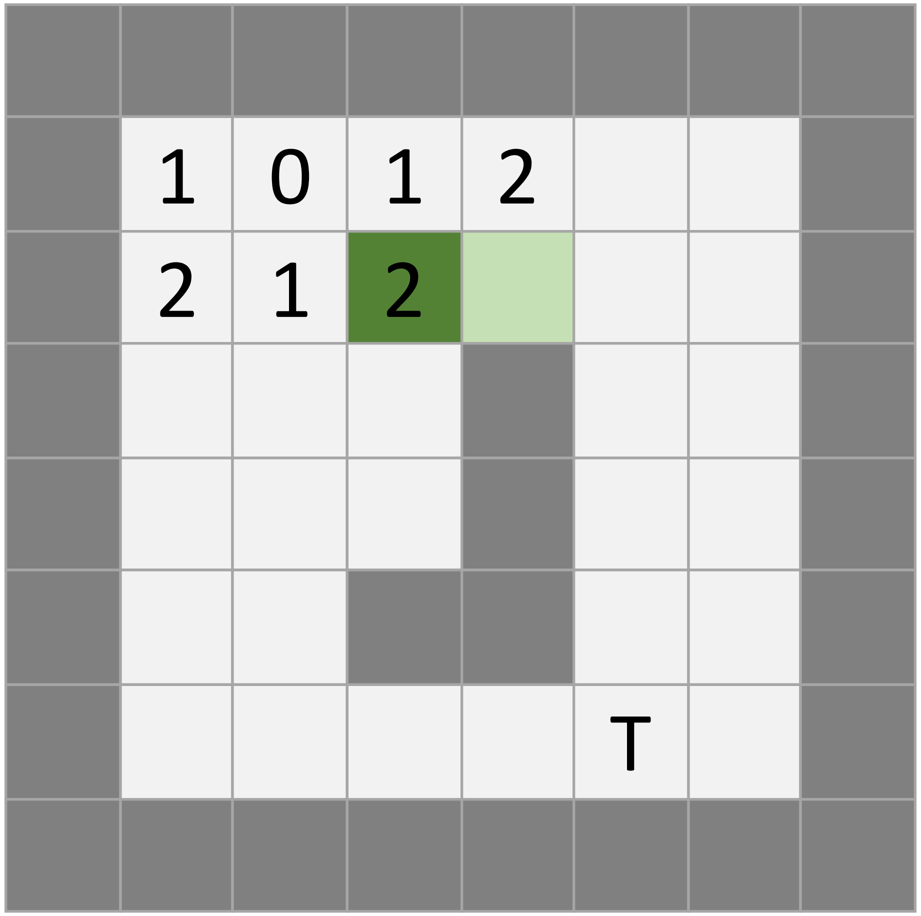
Slow Version

Path Length
`i = 2`



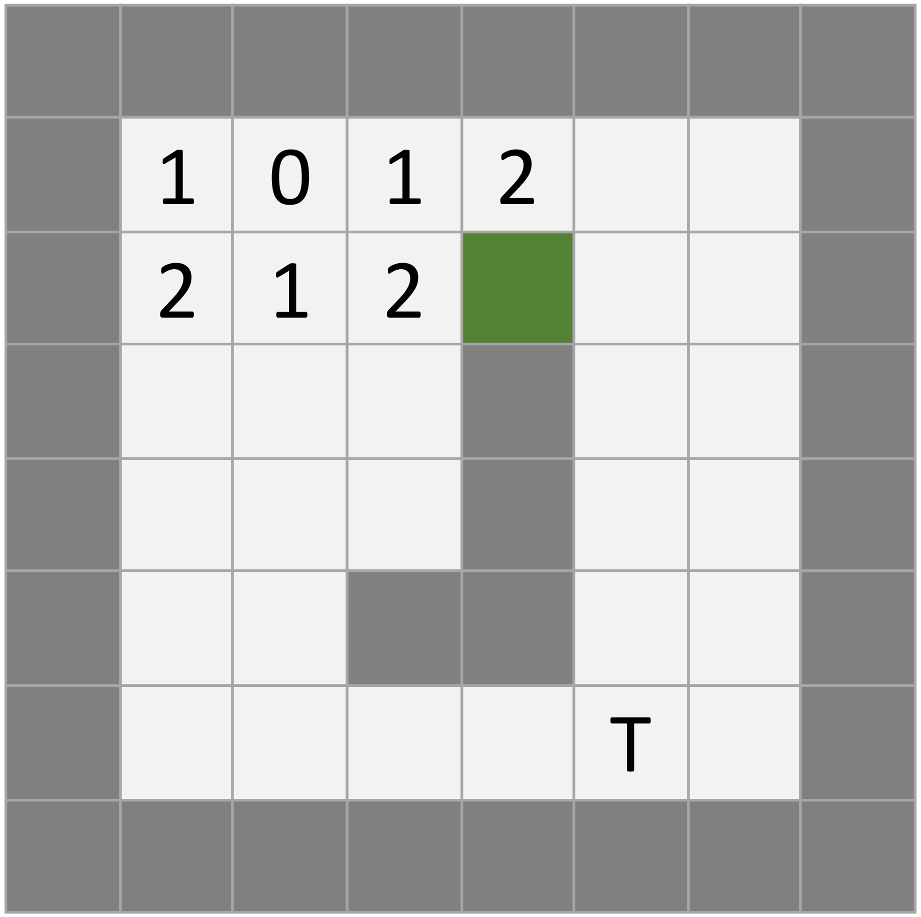
Slow Version

Path Length
`i = 2`



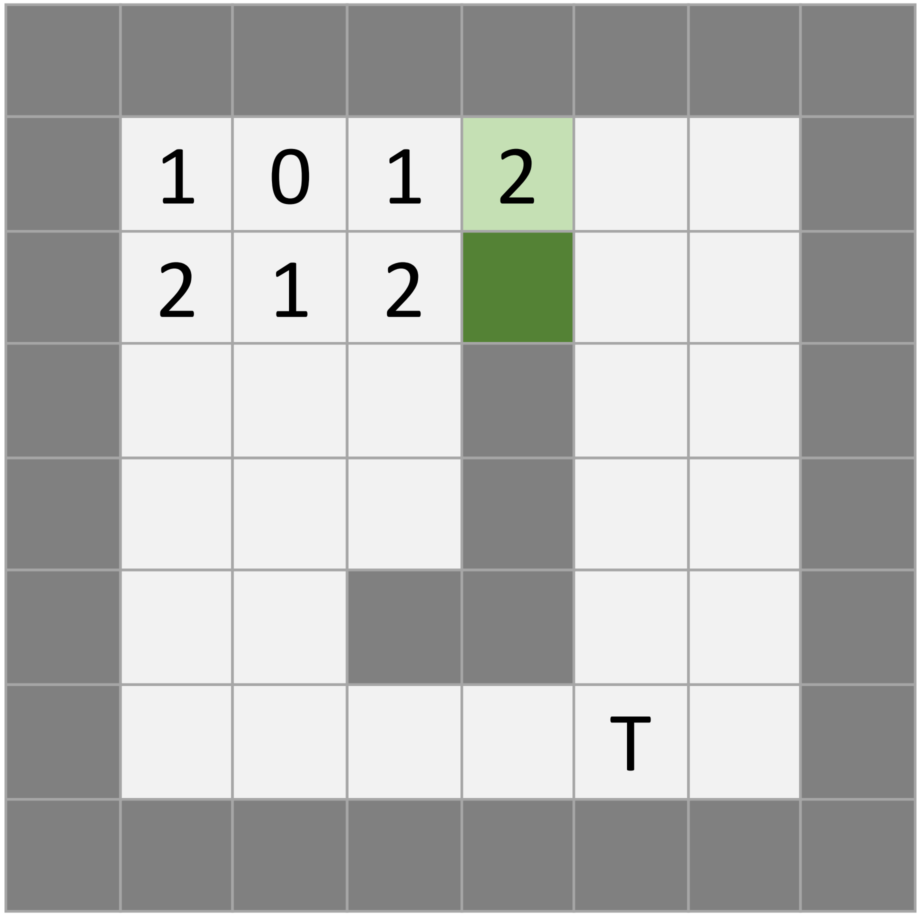
Slow Version

Path Length
`i = 2`



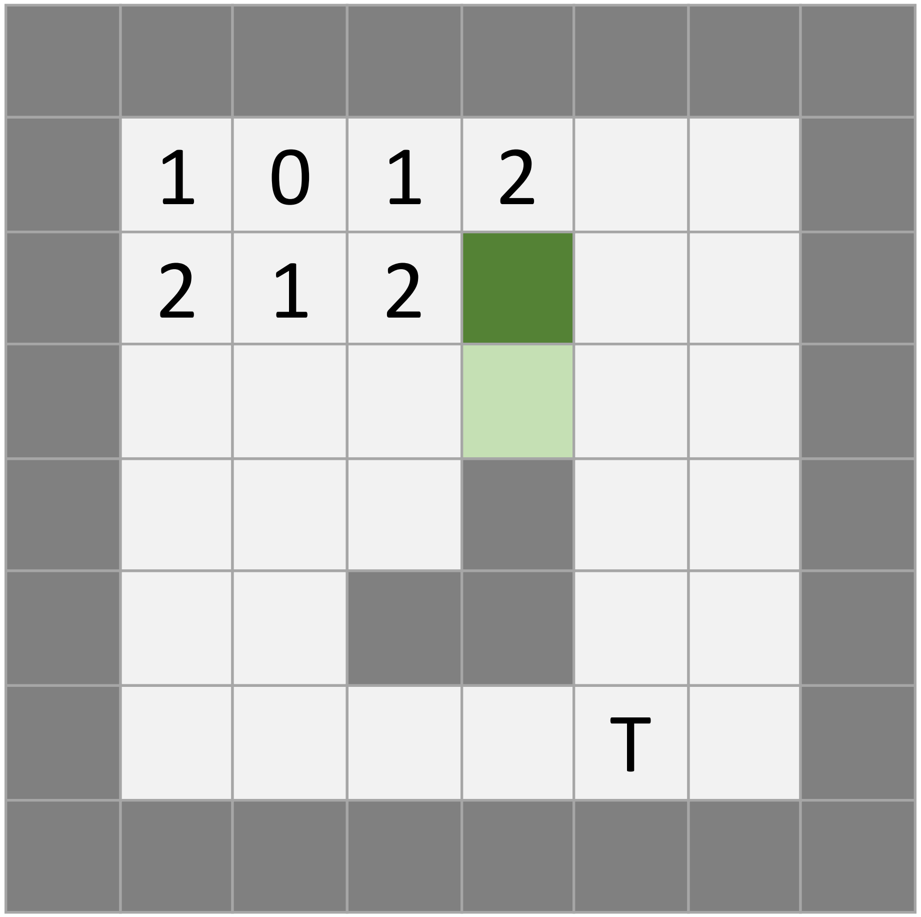
Slow Version

Path Length
`i = 2`



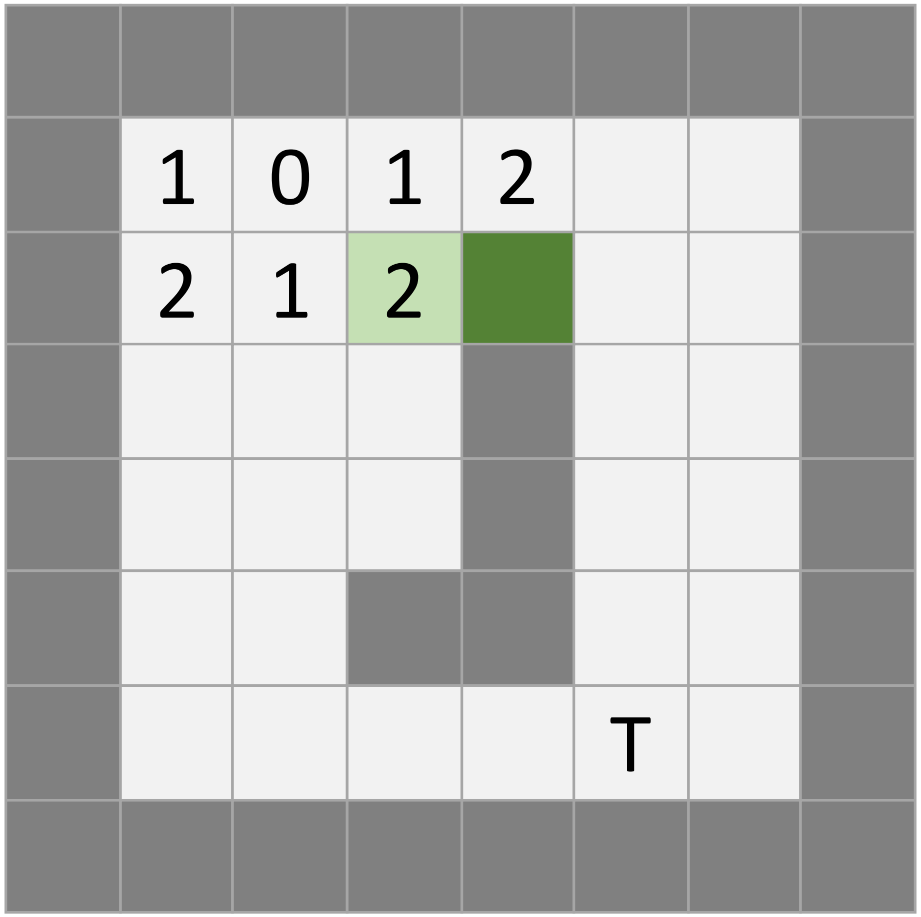
Slow Version

Path Length
`i = 2`



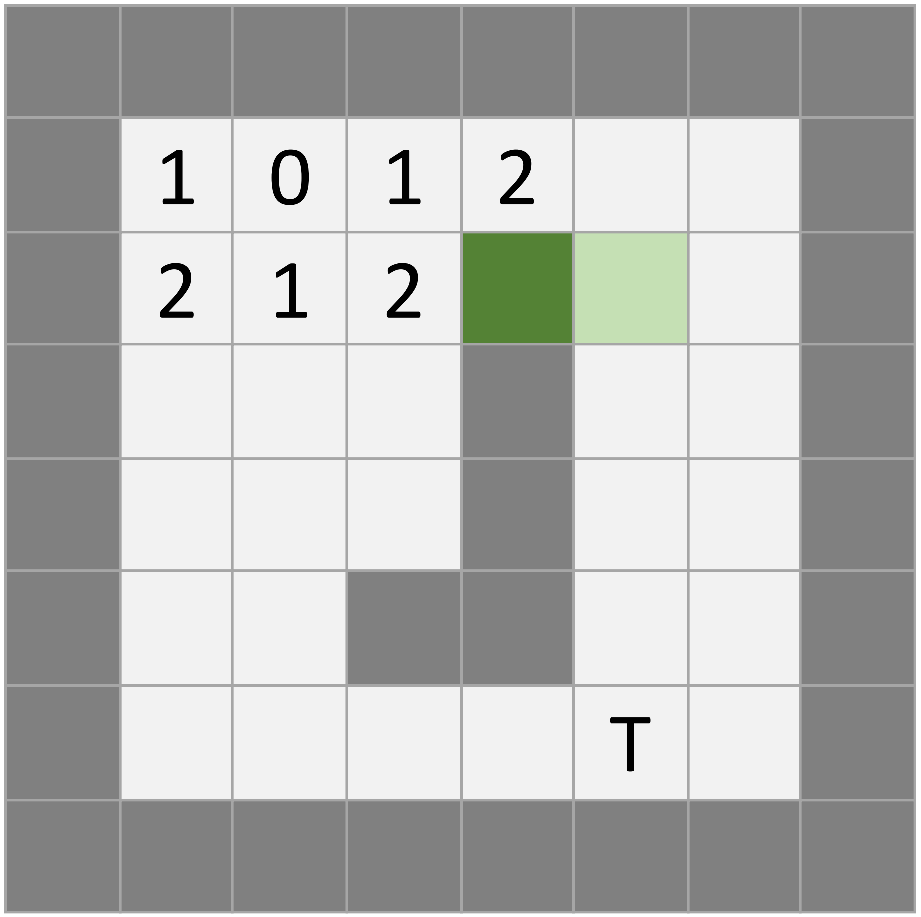
Slow Version

Path Length
`i = 2`



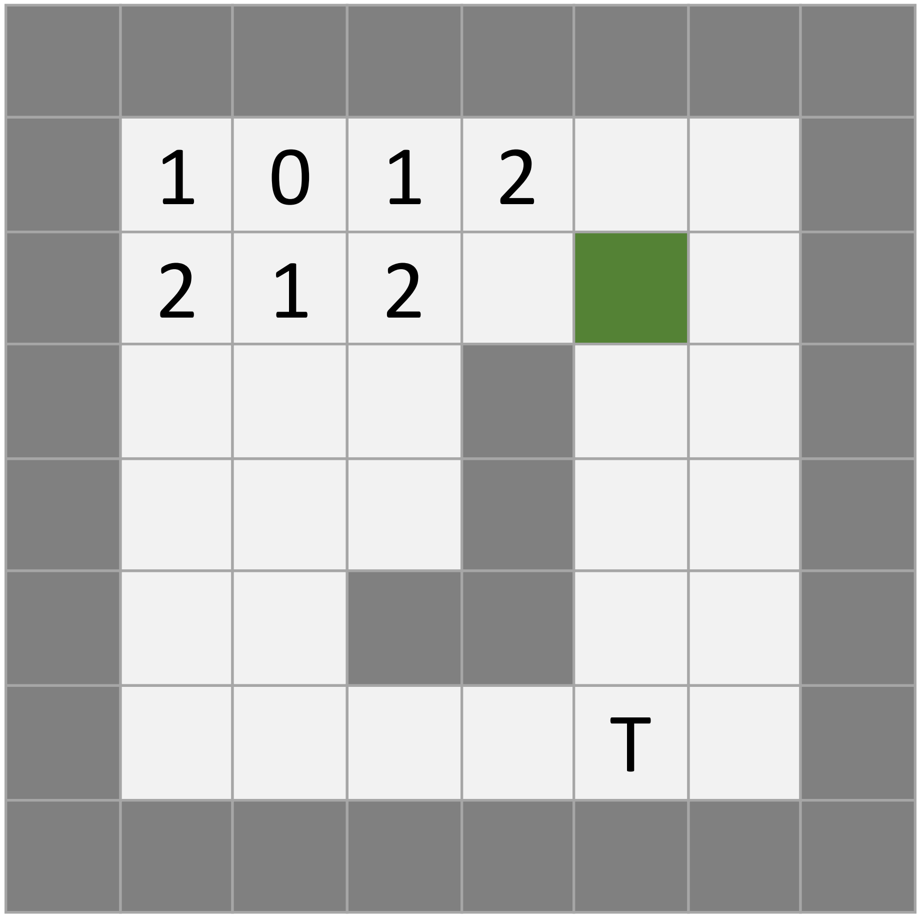
Slow Version

Path Length
`i = 2`



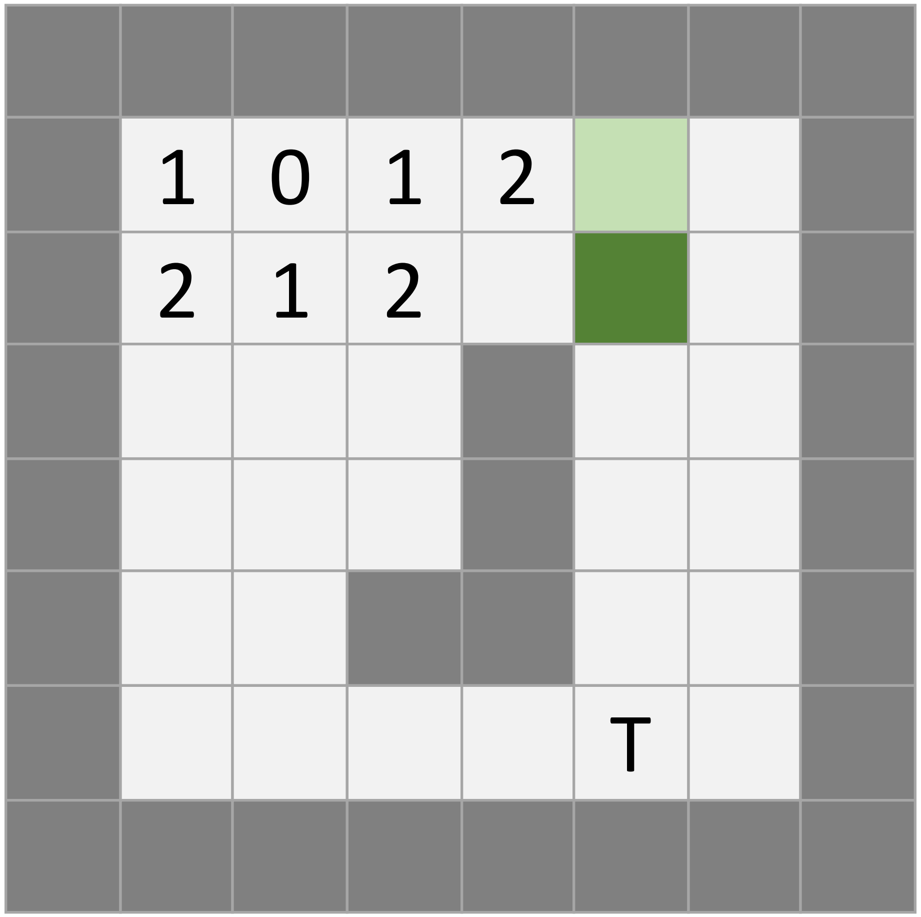
Slow Version

Path Length
`i = 2`



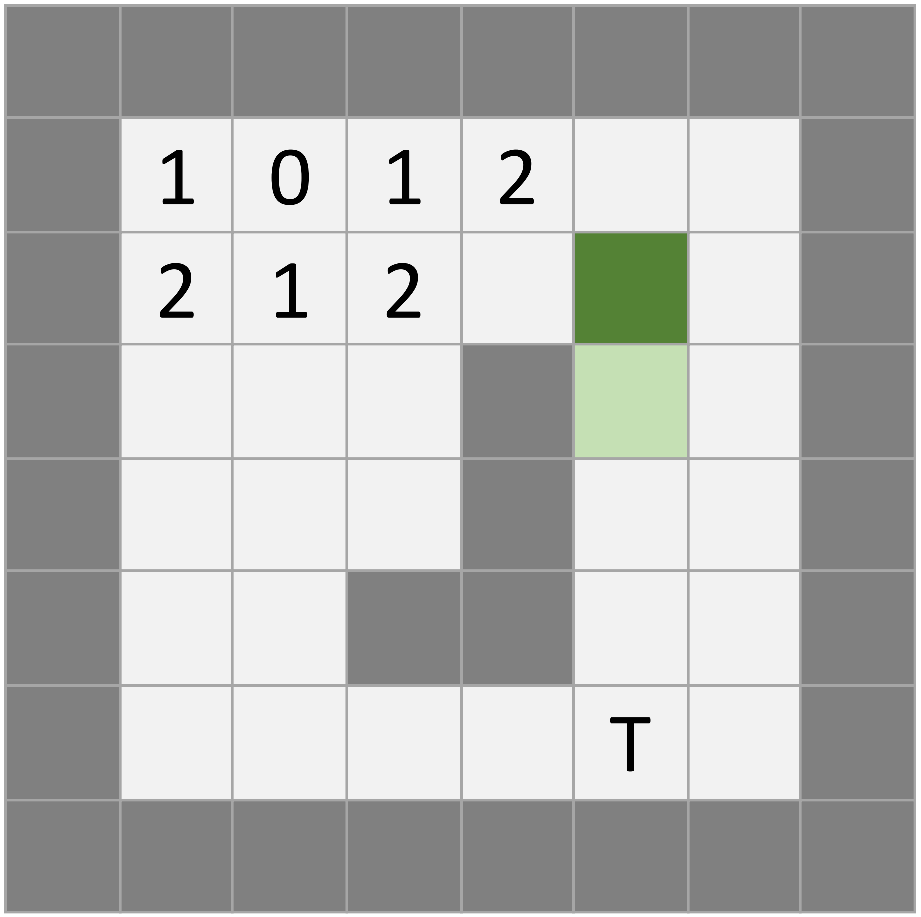
Slow Version

Path Length
`i = 2`



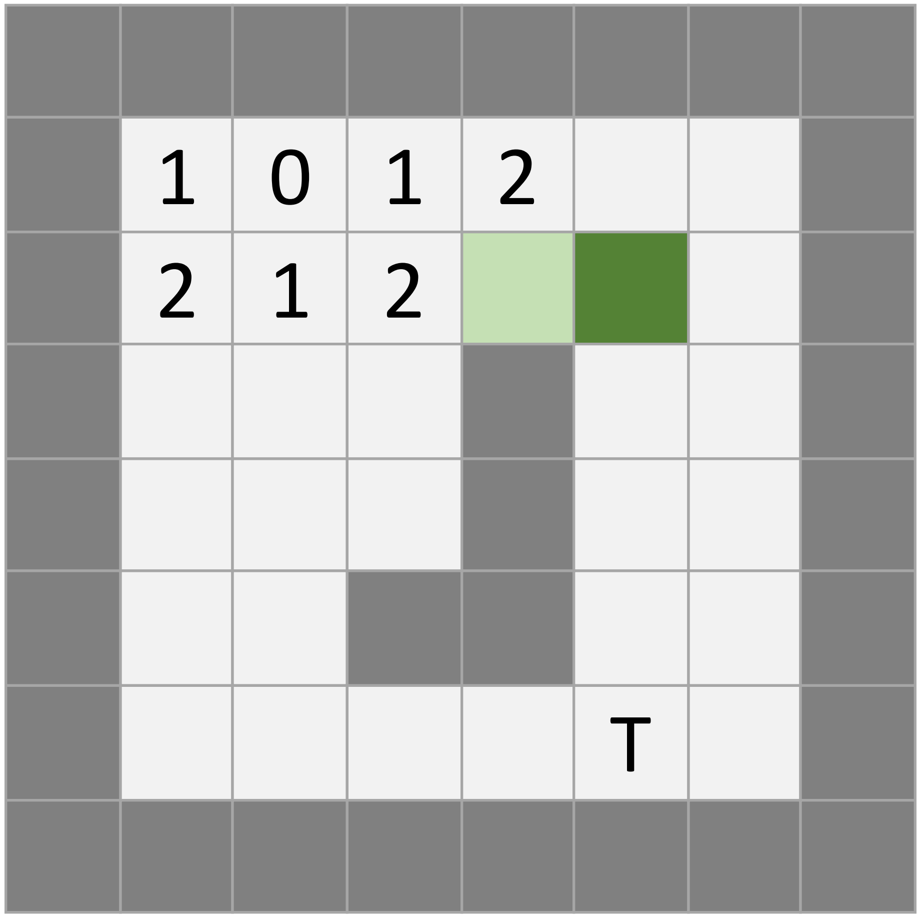
Slow Version

Path Length
`i = 2`



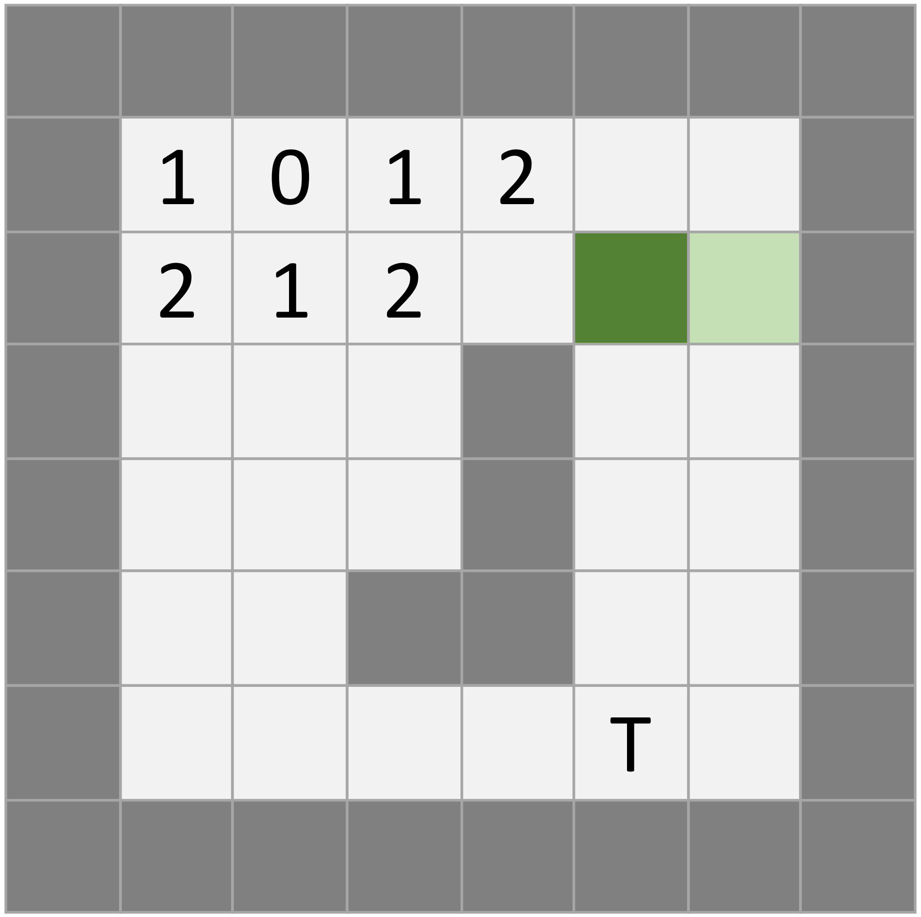
Slow Version

Path Length
`i = 2`



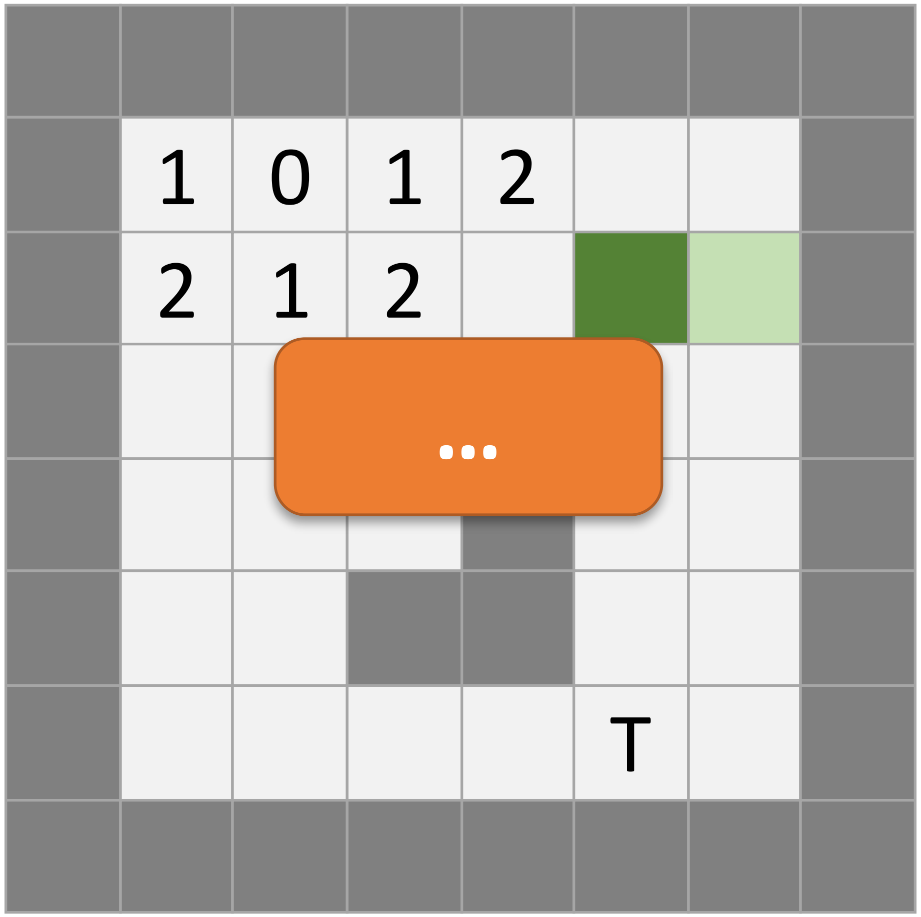
Slow Version

Path Length
`i = 2`



Slow Version

Path Length
`i = 2`



Slow Version

Path Length
`i = 9`

■	■	■	■	■	■	■	■
■	1	0	1	2	3	4	■
■	2	1	2	3	4	5	■
■	3	2	3	■	5	6	■
■	4	3	4	■	6	7	■
■	5	4	■	■	7	8	■
■	6	5	6	7	8	9	■
■	■	■	■	■	■	■	■

Improvement

- Problem:

In each step the whole floor is examined.

Improvement

- Problem:

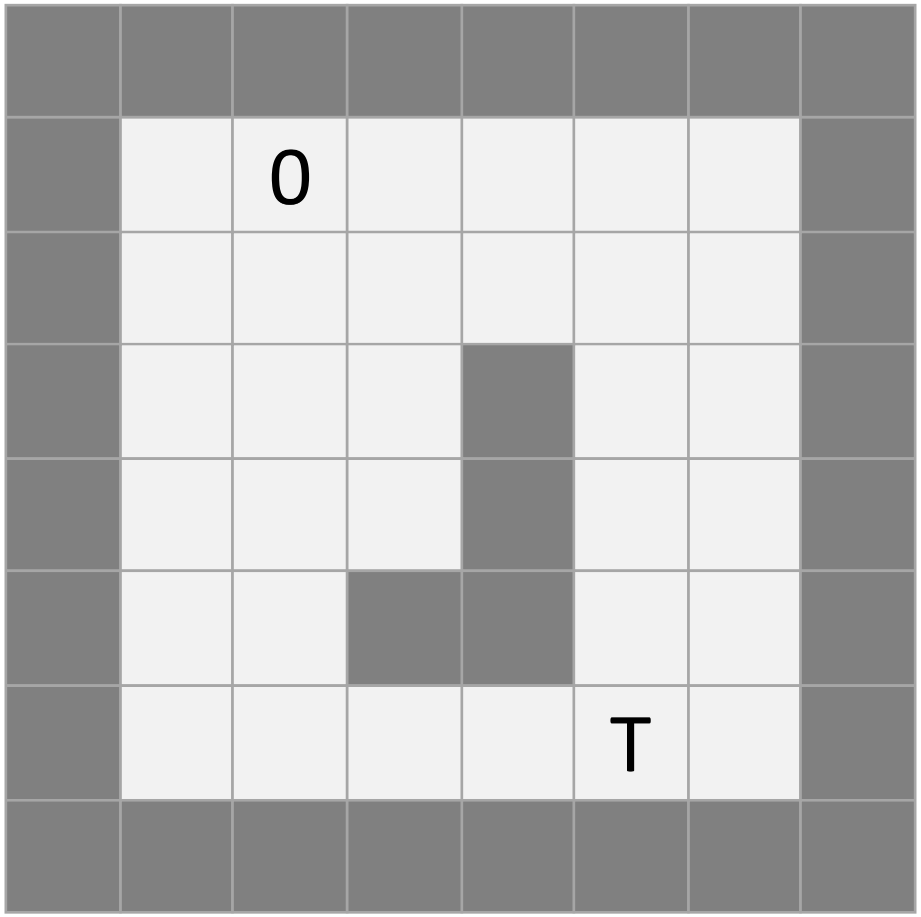
In each step the whole floor is examined.

- Idea:

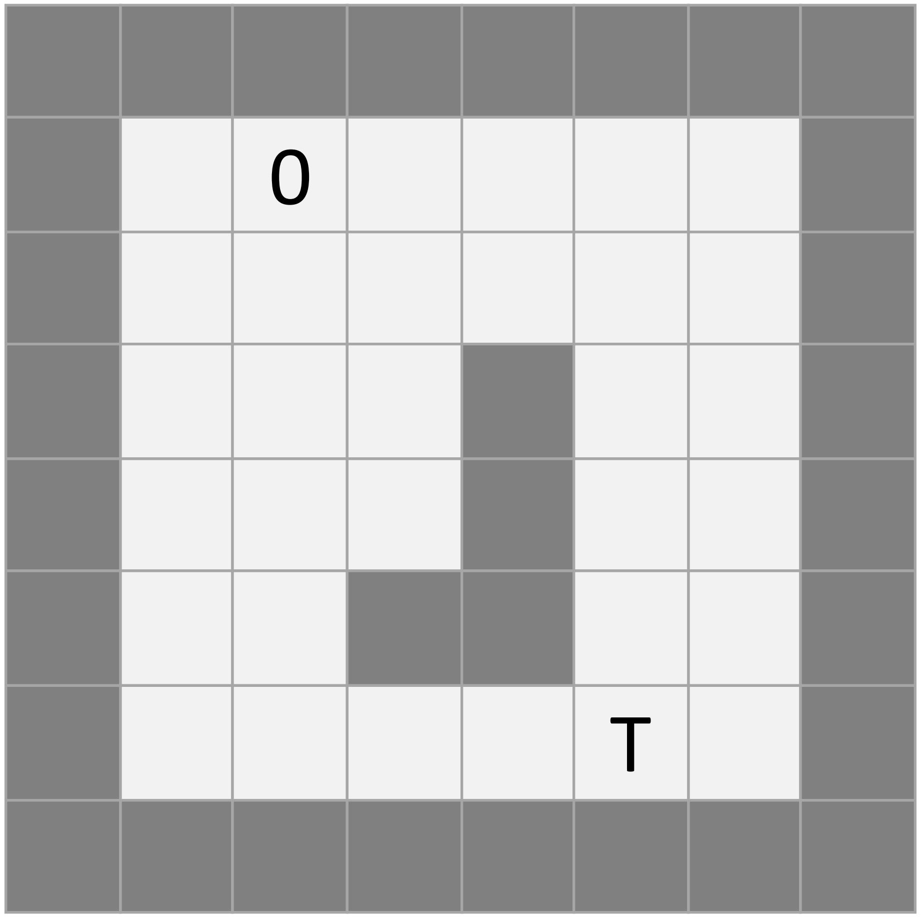
Just examine neighbours.

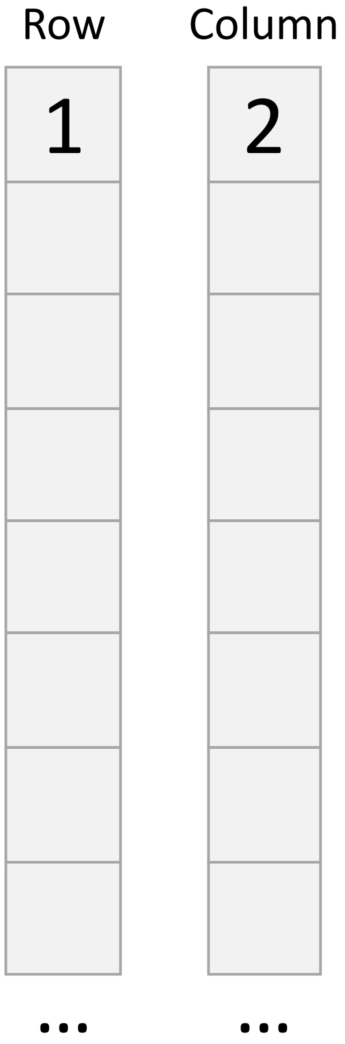
Faster Version

Faster Version

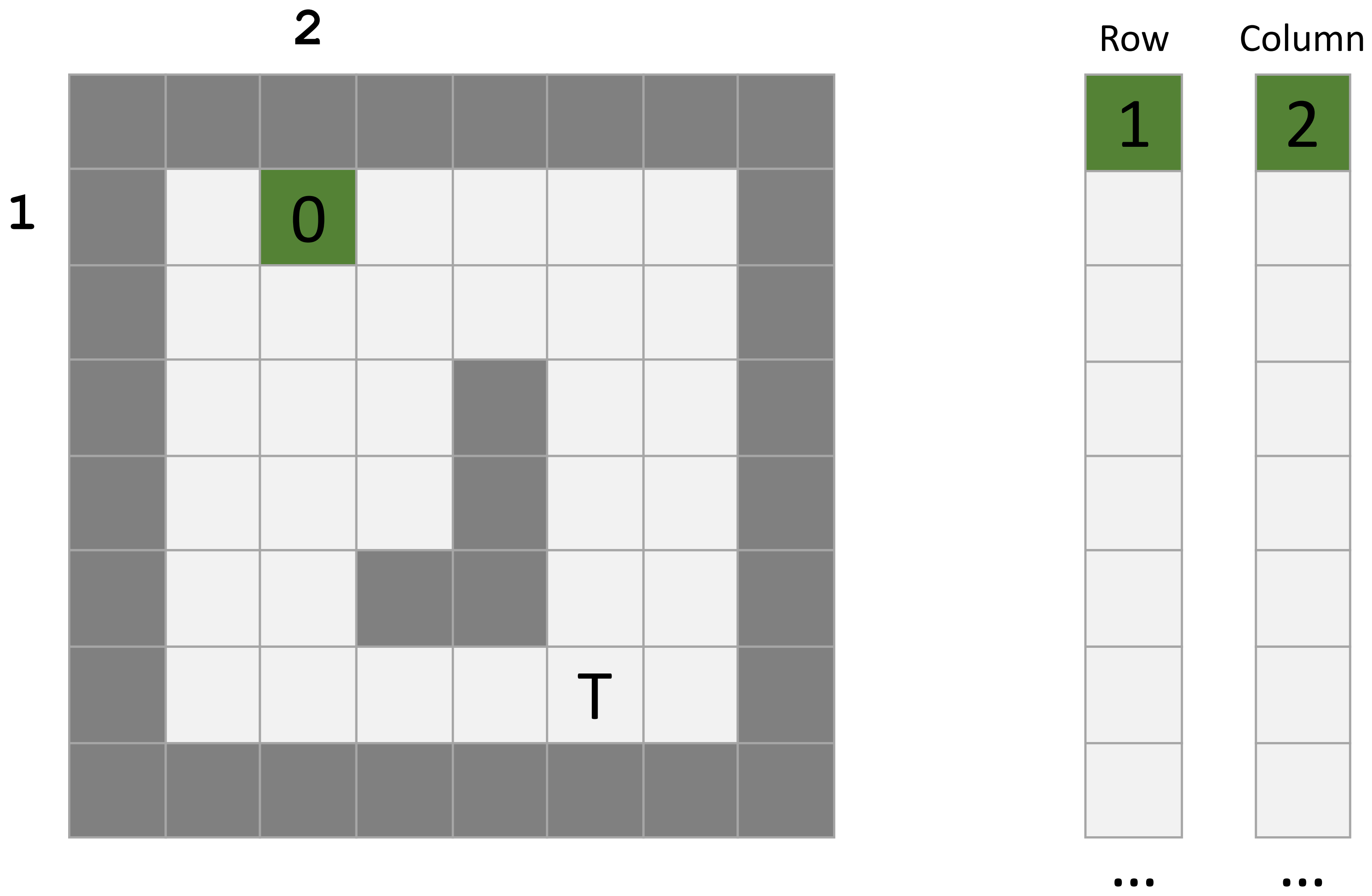


Faster Version

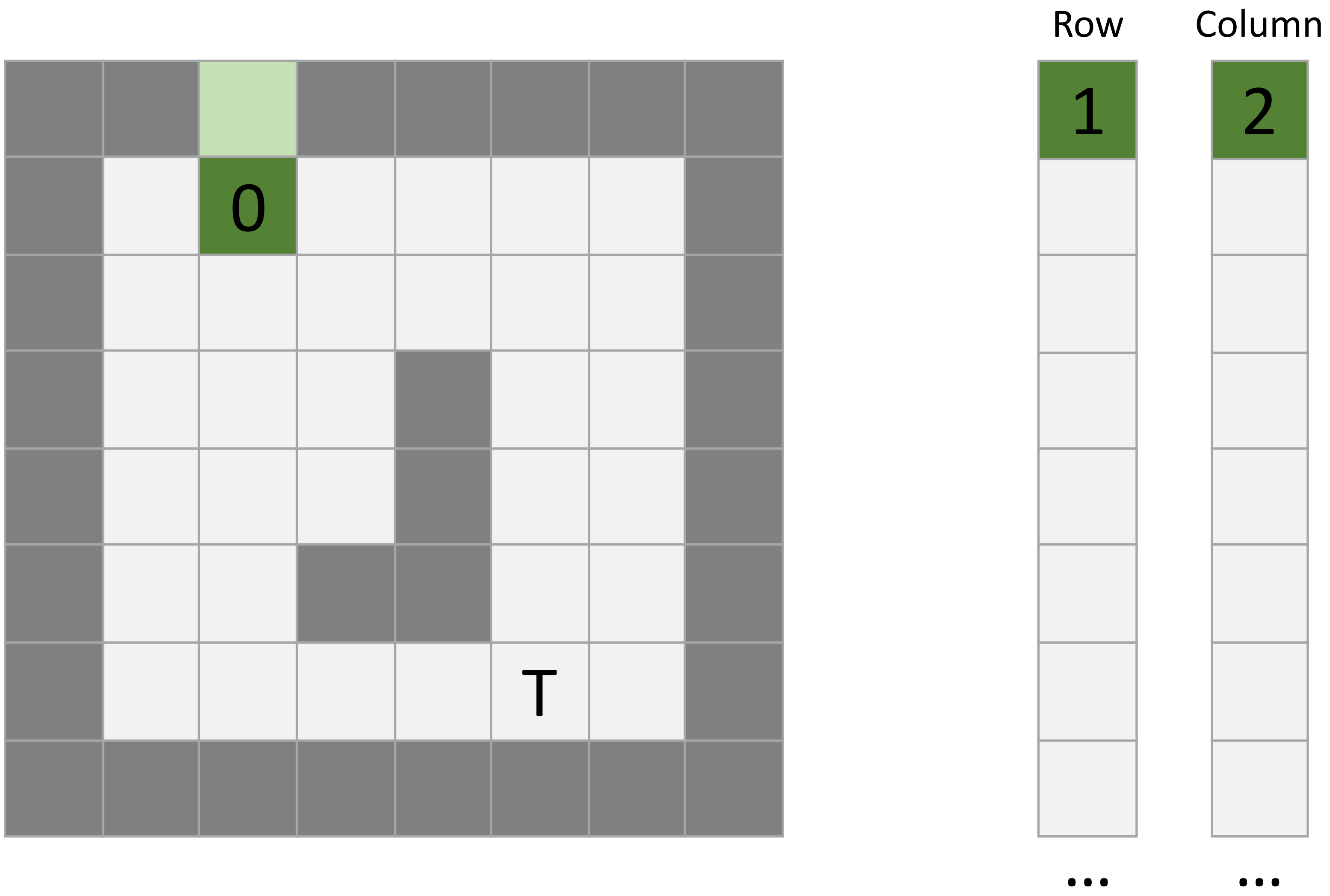




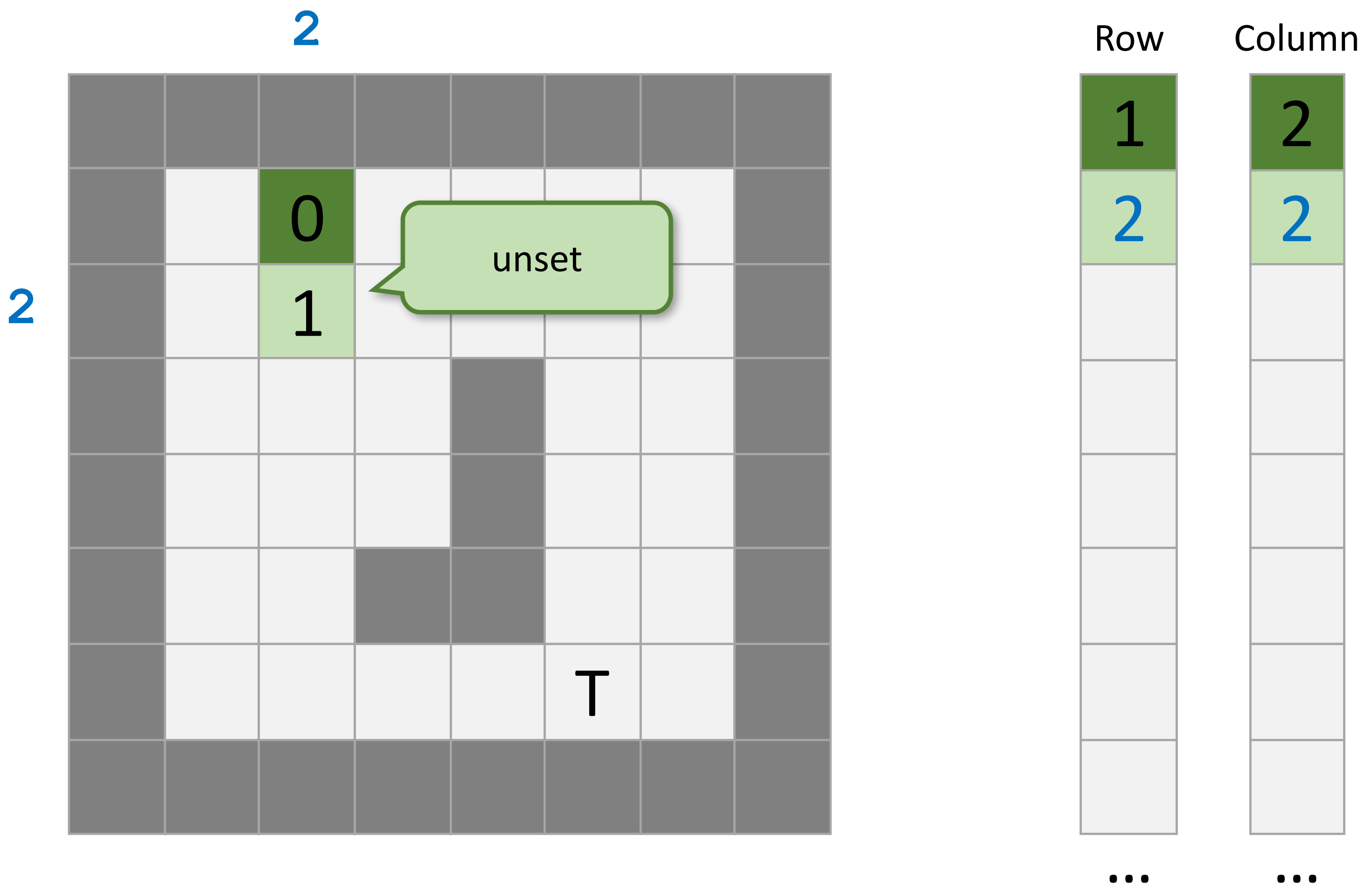
Faster Version



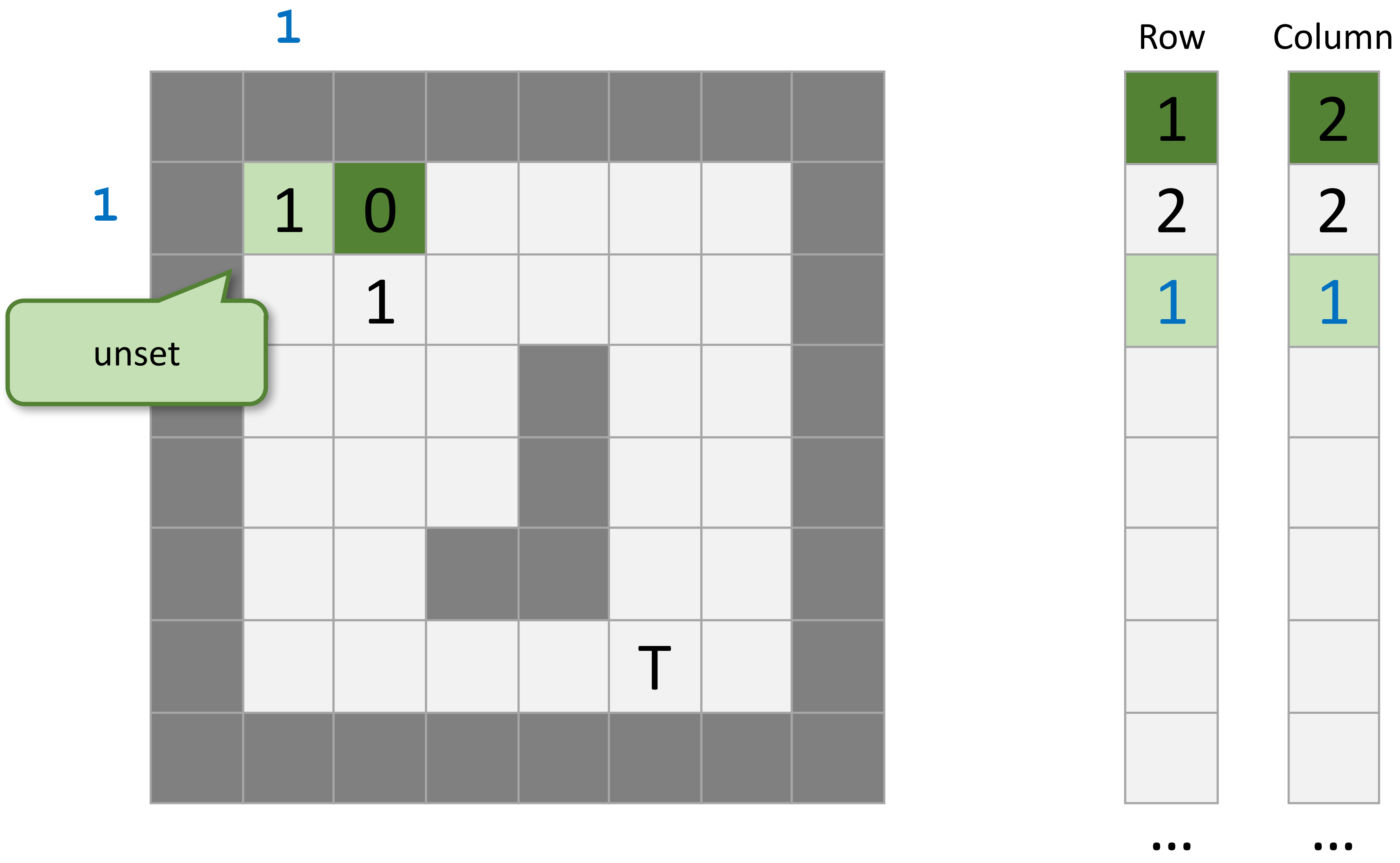
Faster Version



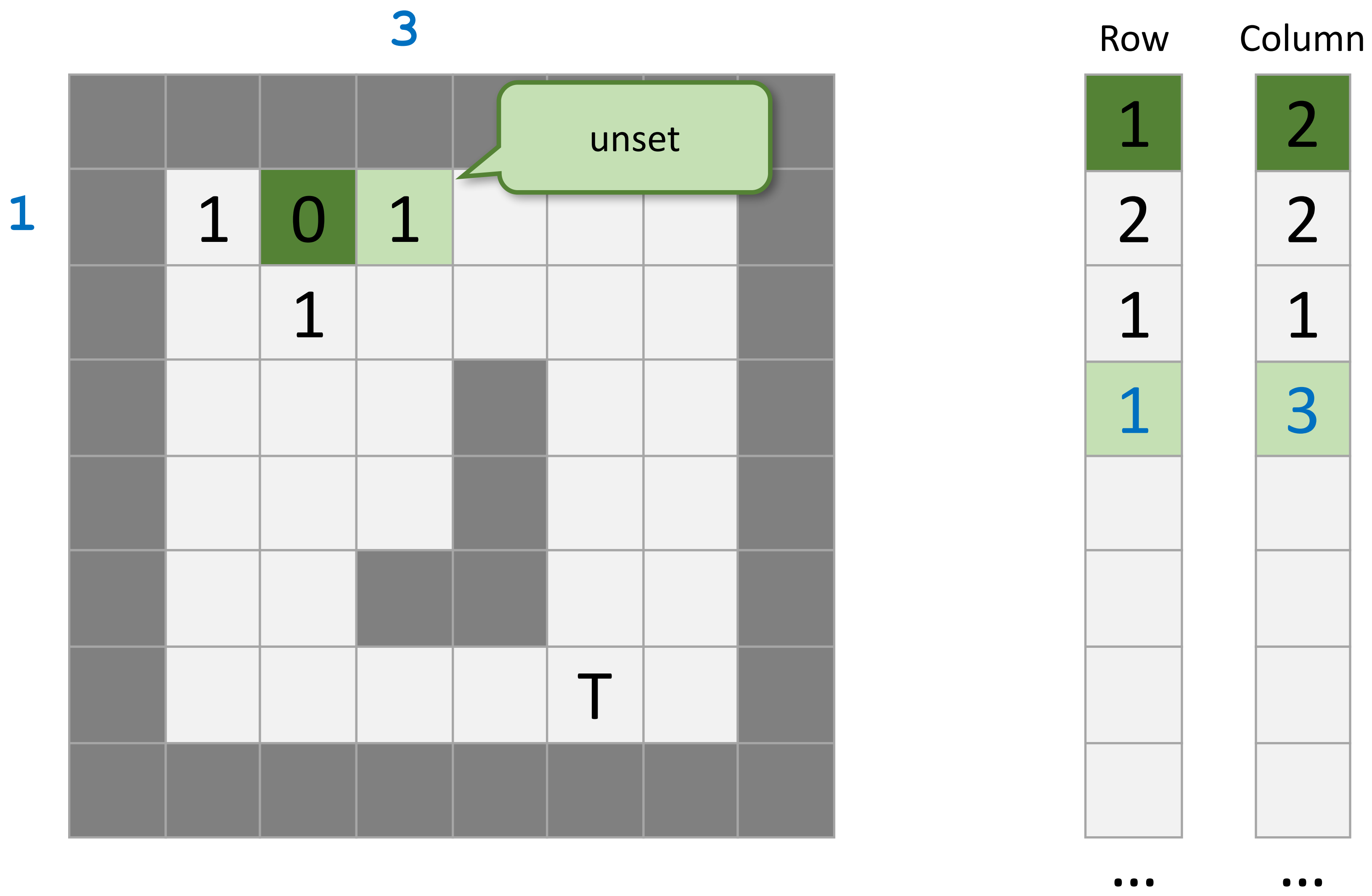
Faster Version



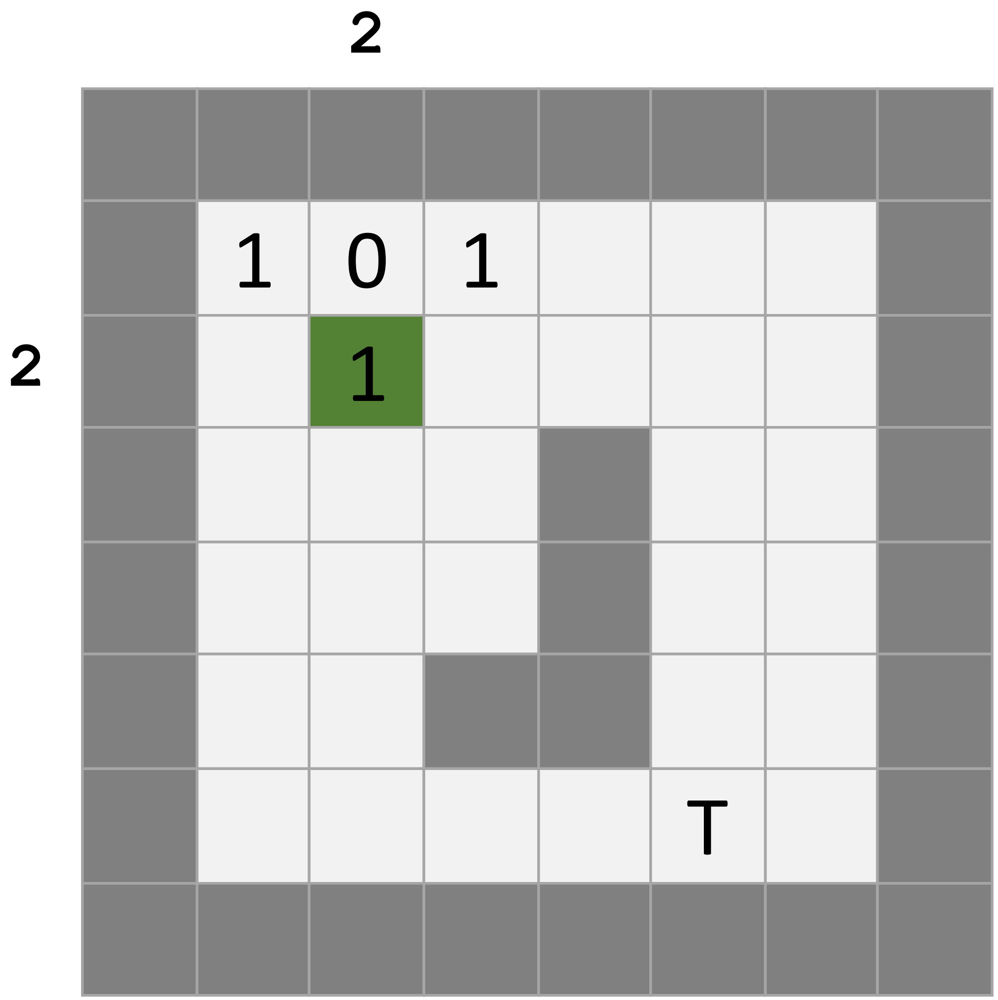
Faster Version



Faster Version

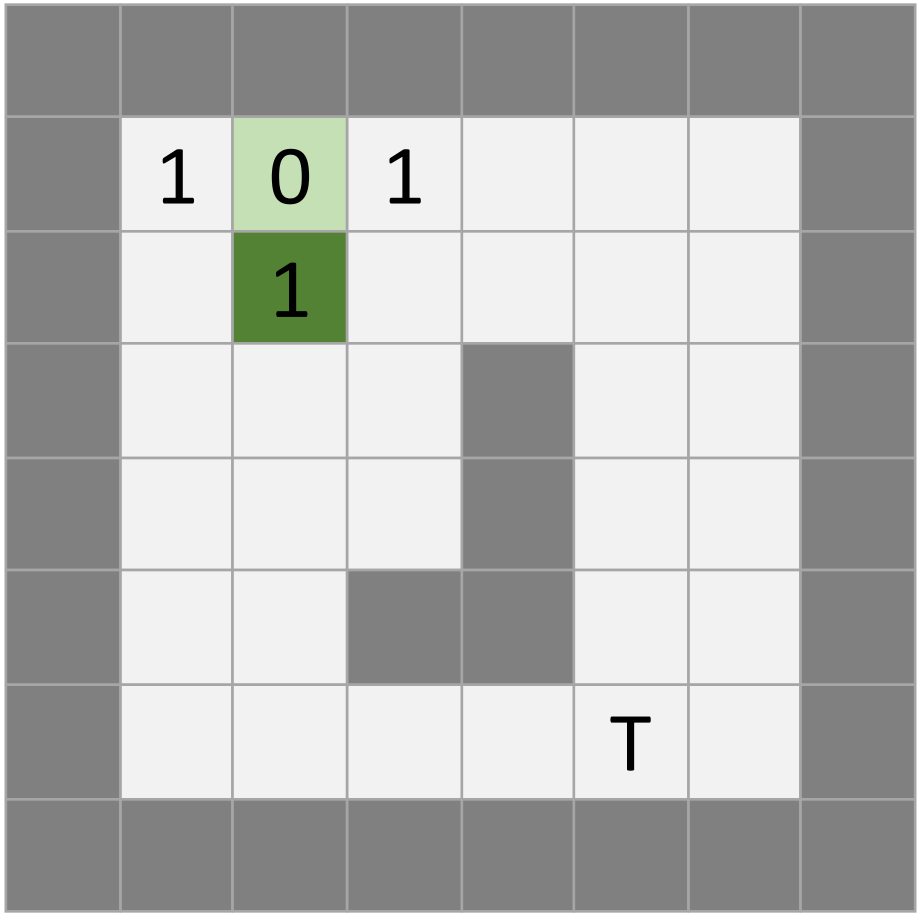


Faster Version



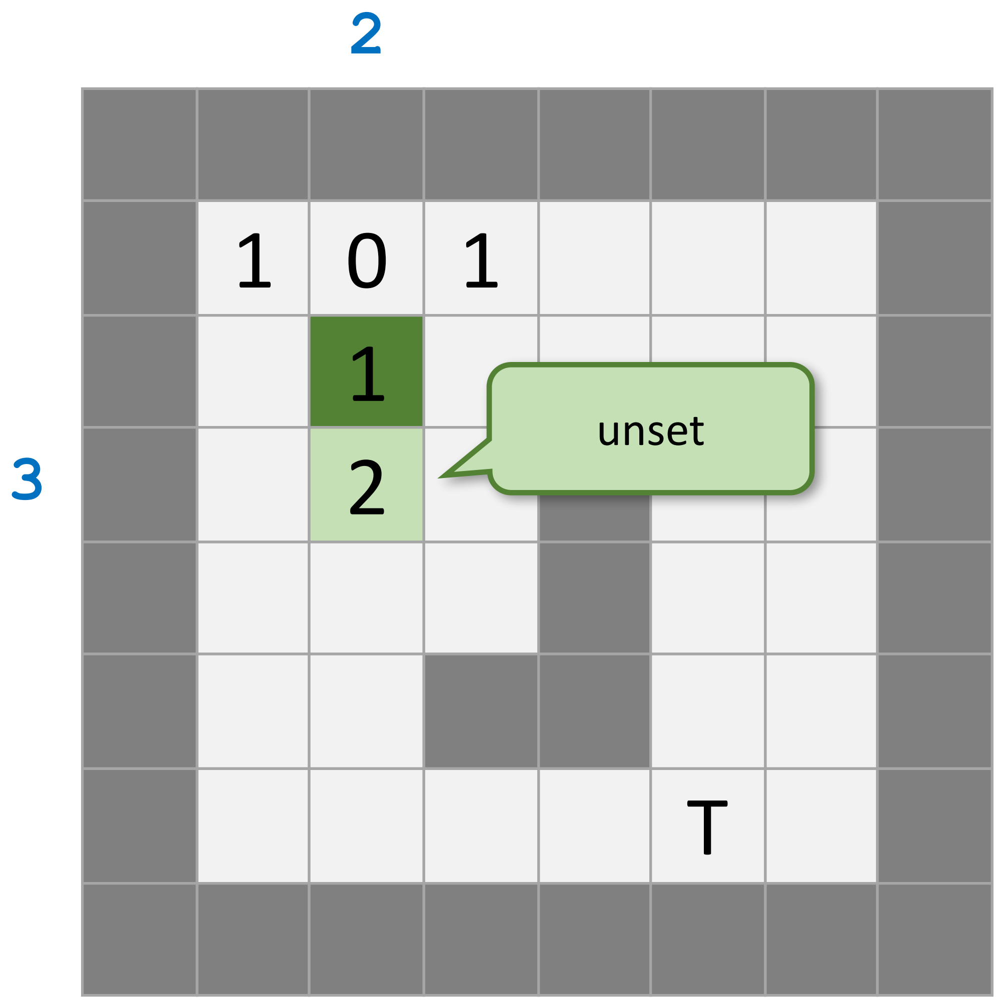
Row	Column
1	2
2	2
1	1
1	3
...	...

Faster Version



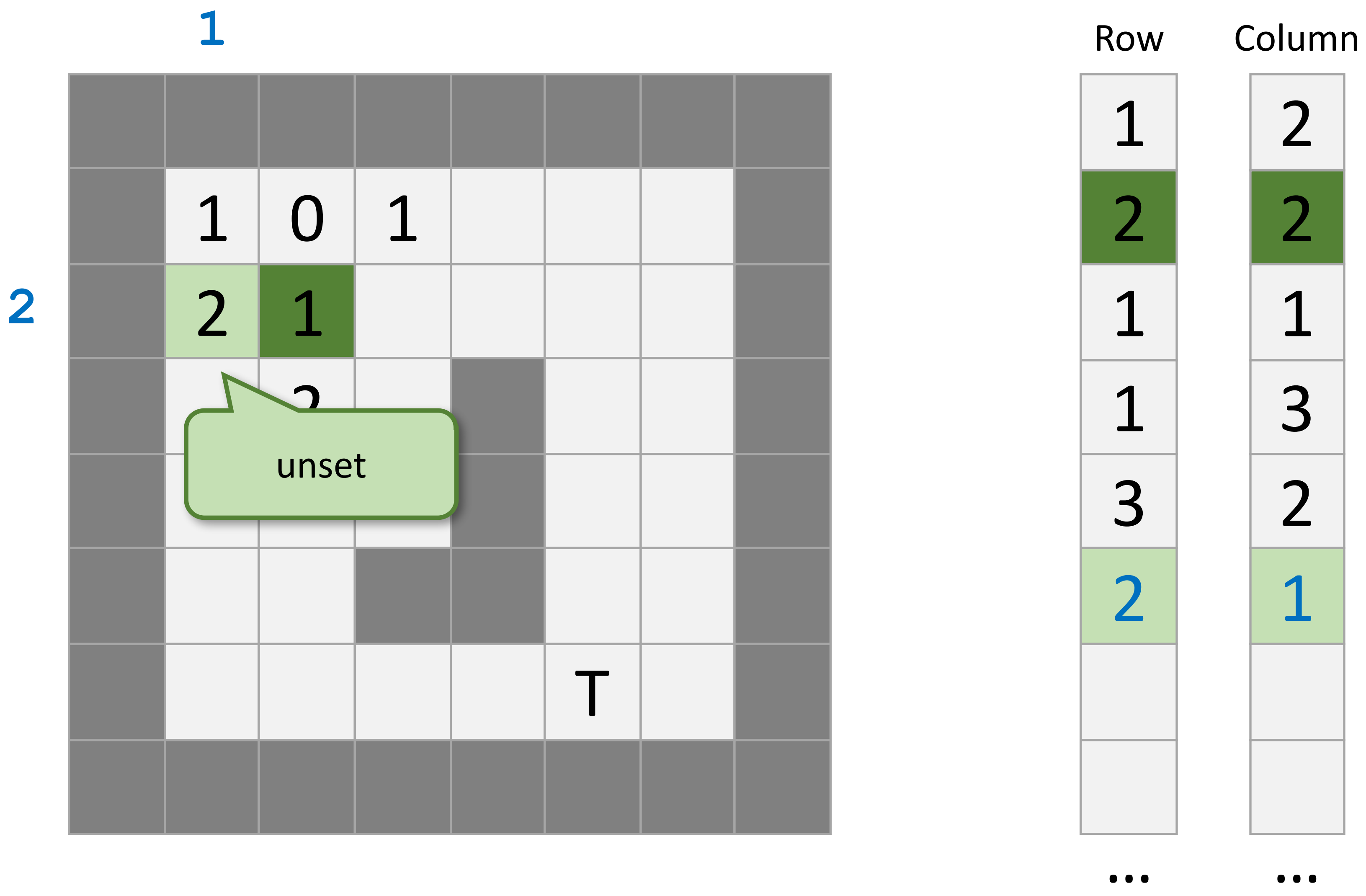
Row	Column
1	2
2	2
1	1
1	3
…	…

Faster Version

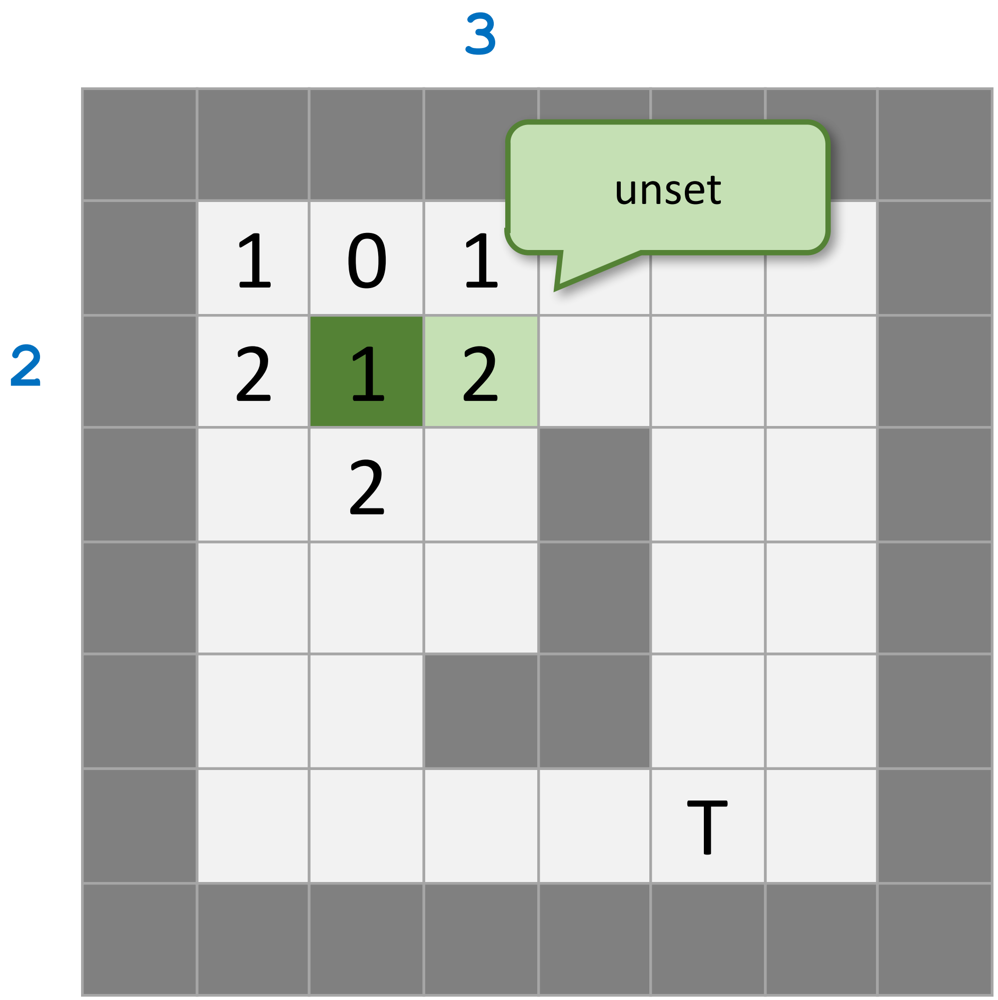


Row	Column
1	2
2	2
1	1
1	3
3	2
…	…

Faster Version

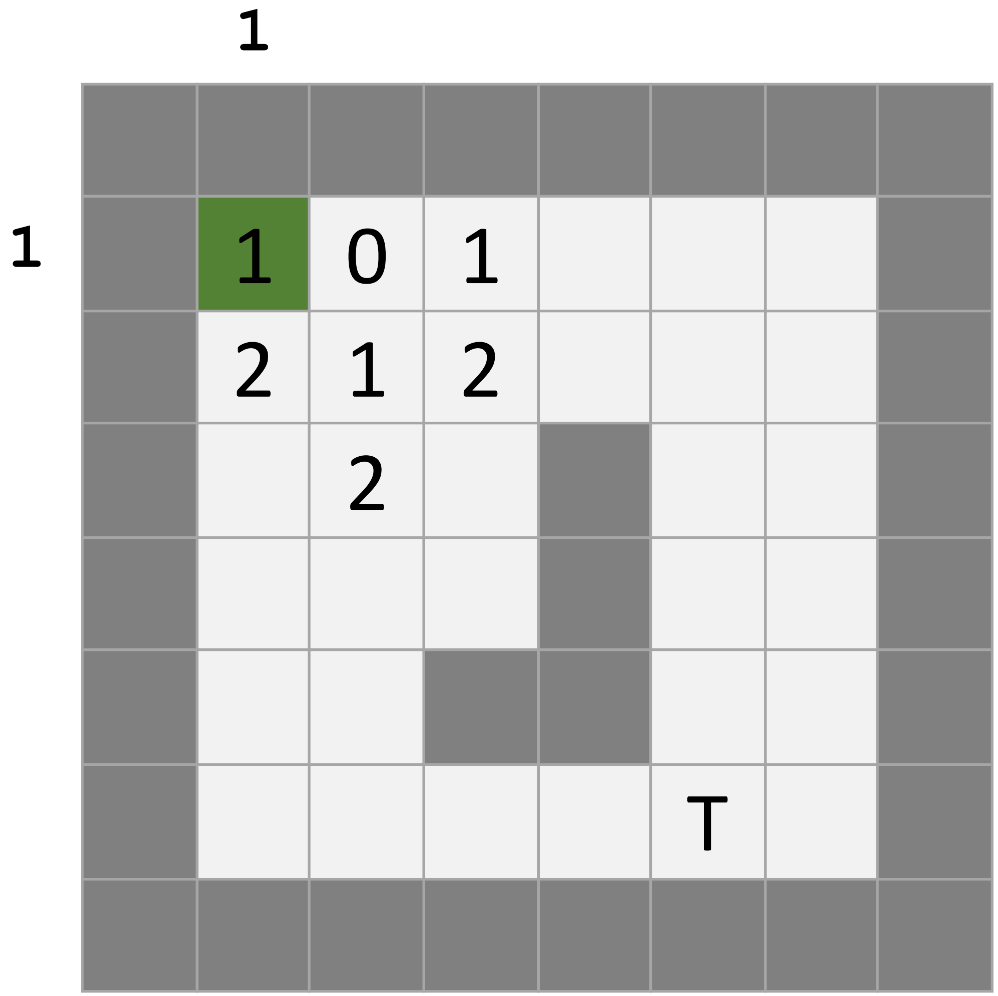


Faster Version



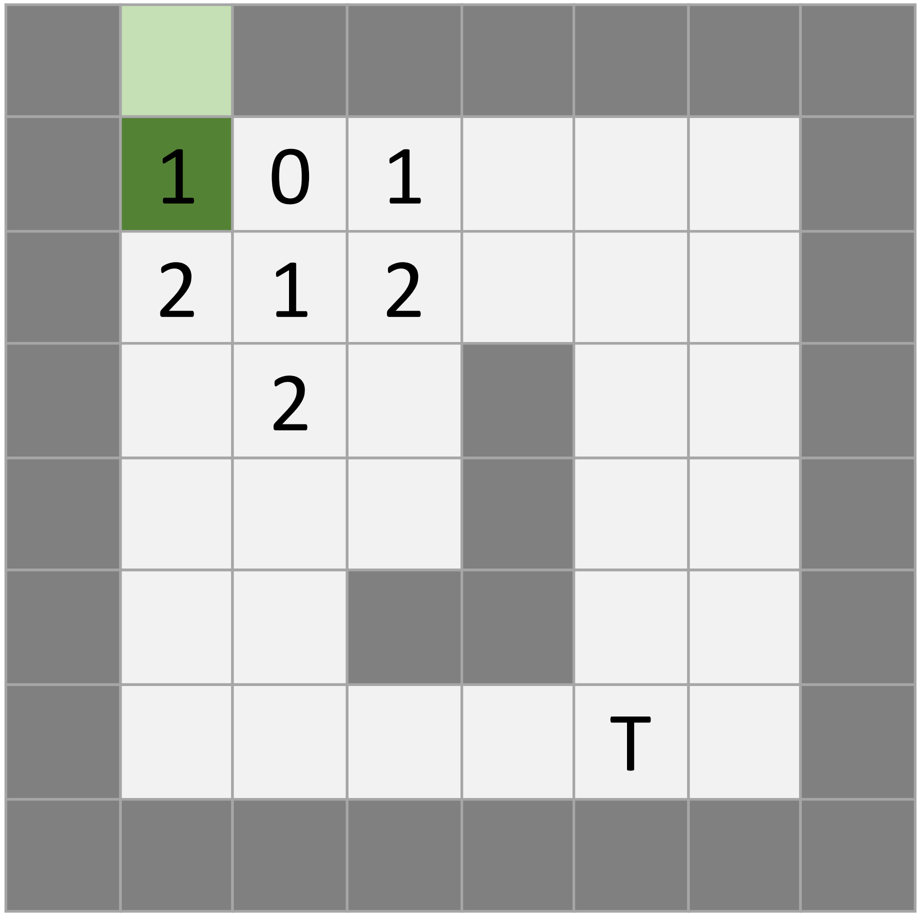
Row	Column
1	2
2	2
1	1
1	3
3	2
2	1
2	3
…	…

Faster Version



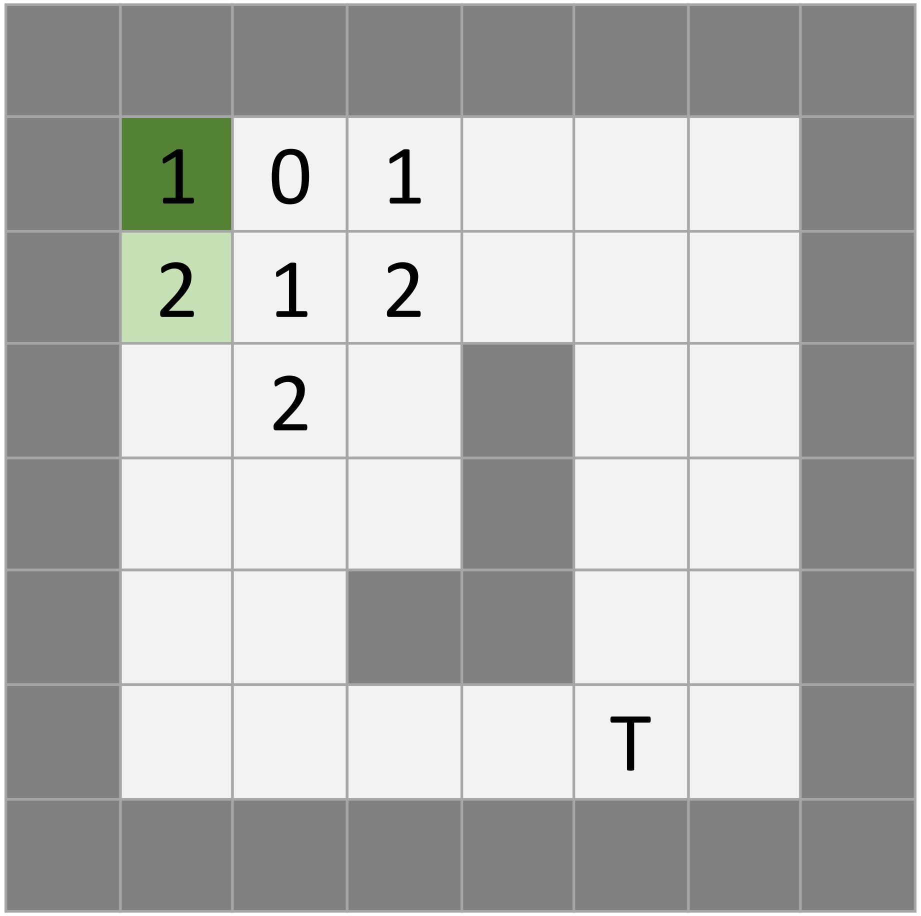
Row	Column
1	2
2	2
1	1
1	3
3	2
2	1
2	3
...	...

Faster Version



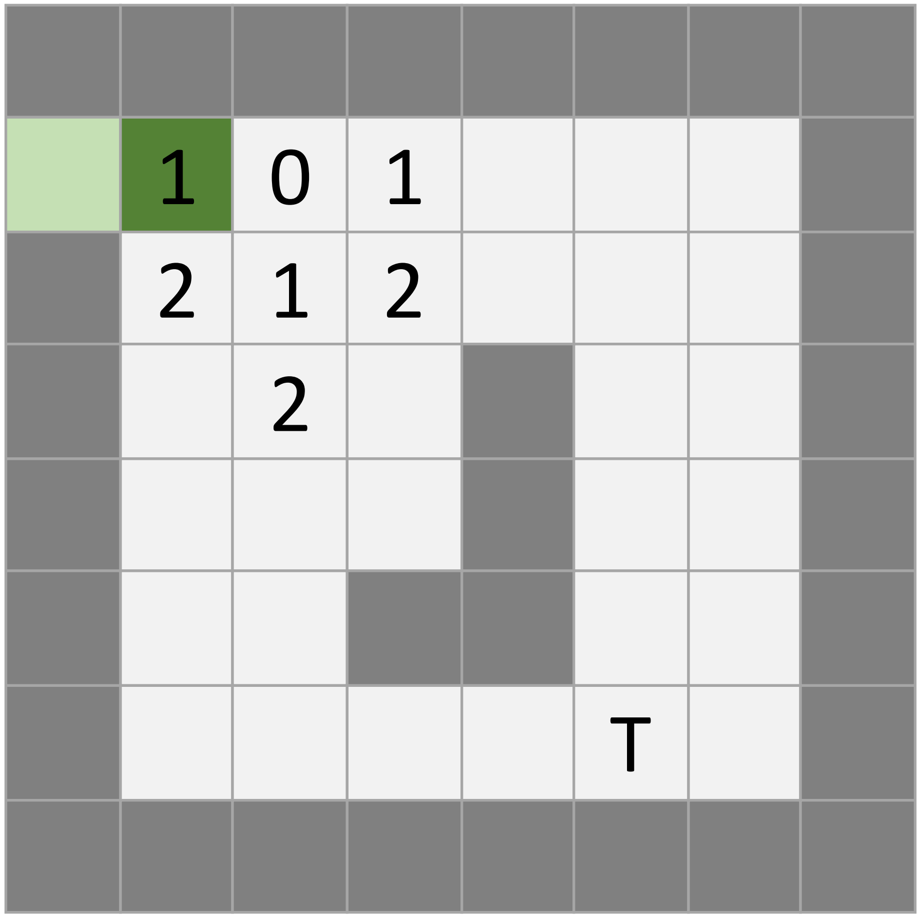
Row	Column
1	2
2	2
1	1
1	3
3	2
2	1
2	3
...	...

Faster Version



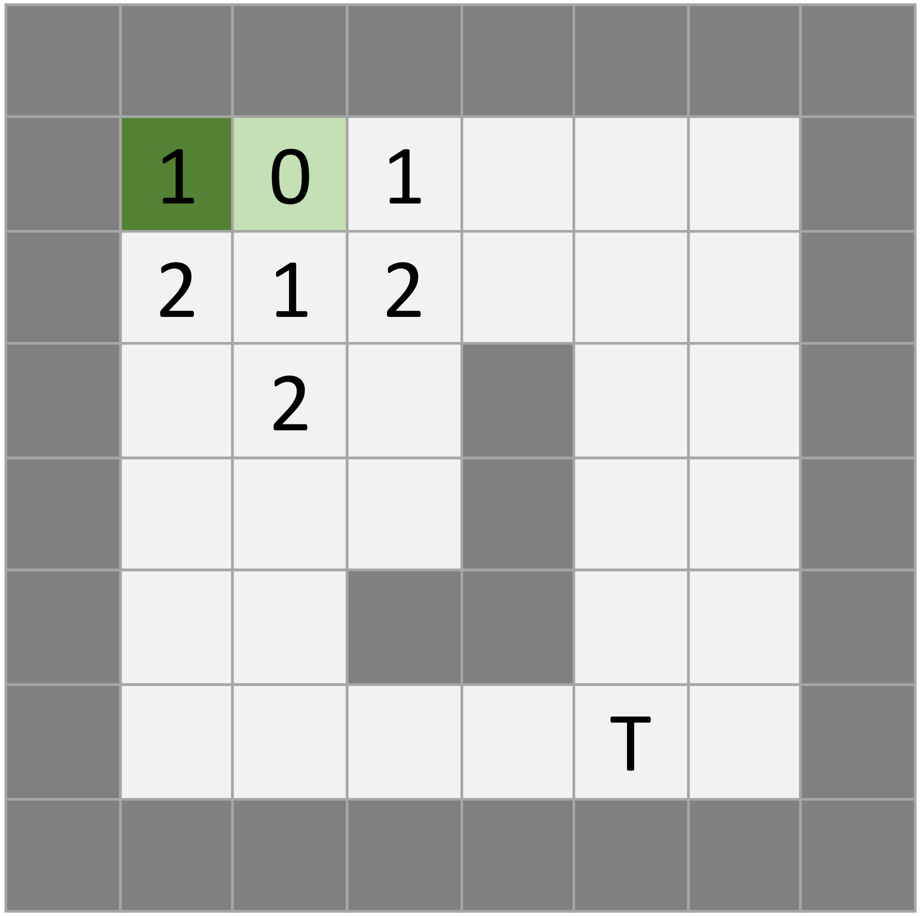
Row	Column
1	2
2	2
1	1
1	3
3	2
2	1
2	3
…	…

Faster Version



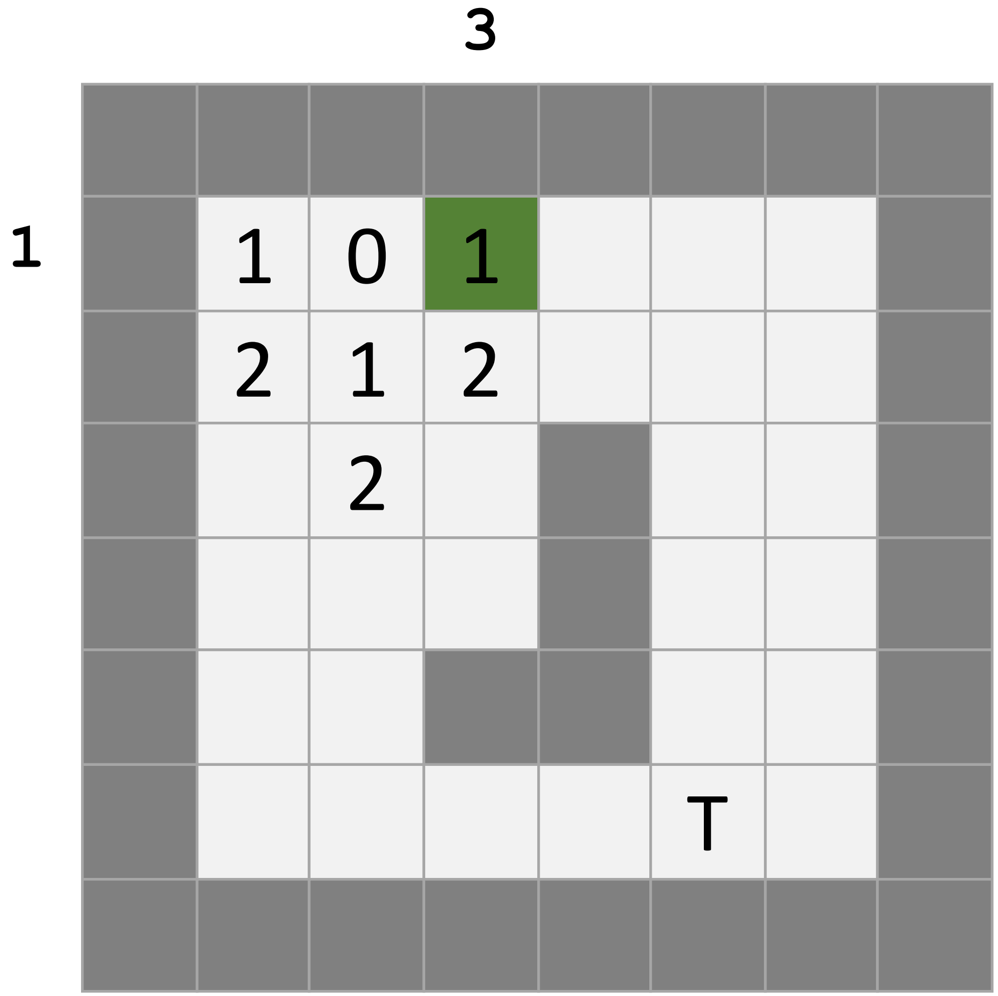
Row	Column
1	2
2	2
1	1
1	3
3	2
2	1
2	3
…	…

Faster Version



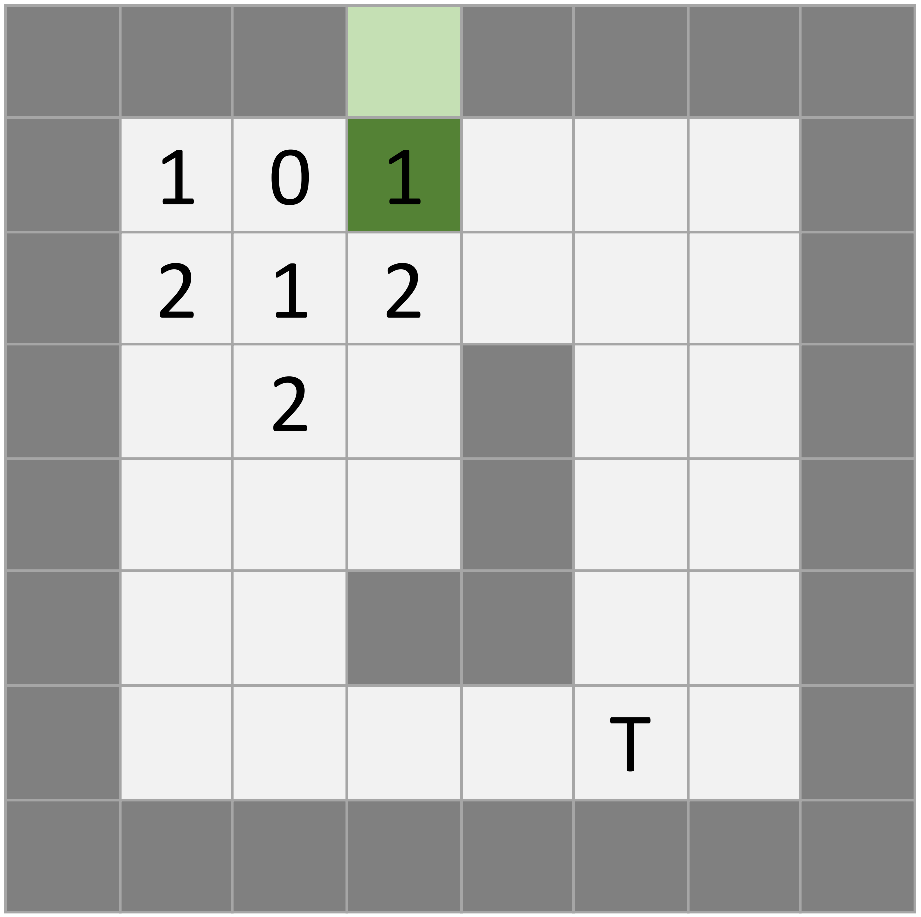
Row	Column
1	2
2	2
1	1
1	3
3	2
2	1
2	3
...	...

Faster Version



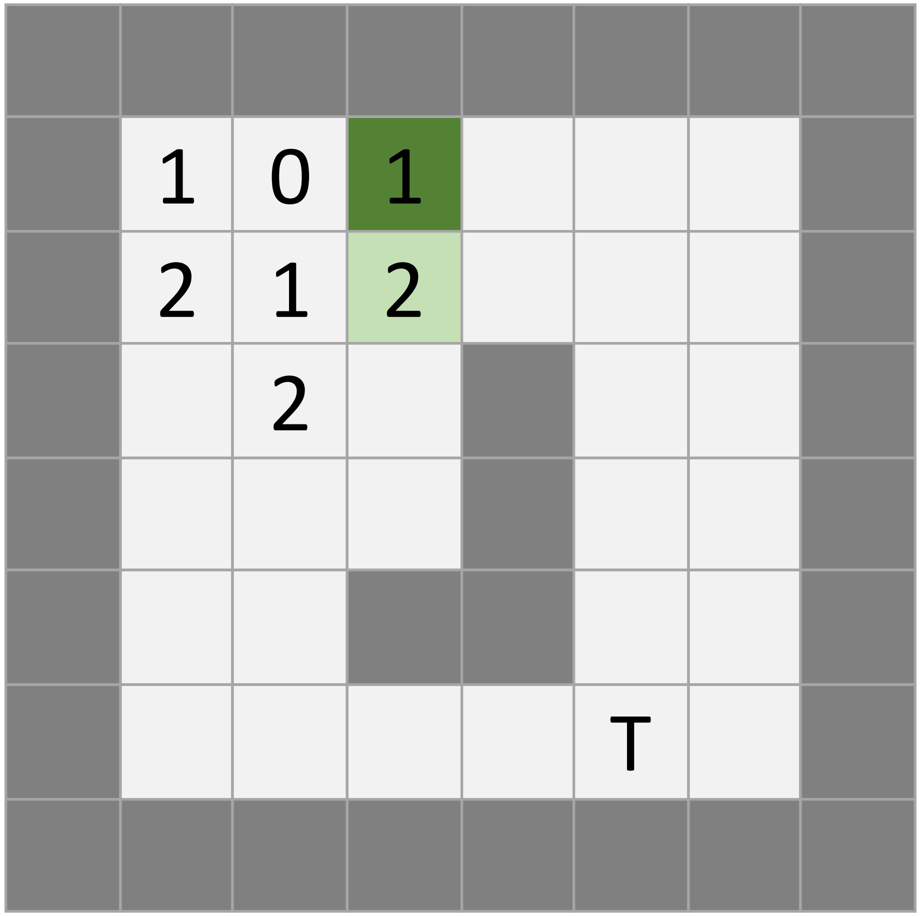
Row	Column
1	2
2	2
1	1
1	3
3	2
2	1
2	3
…	…

Faster Version



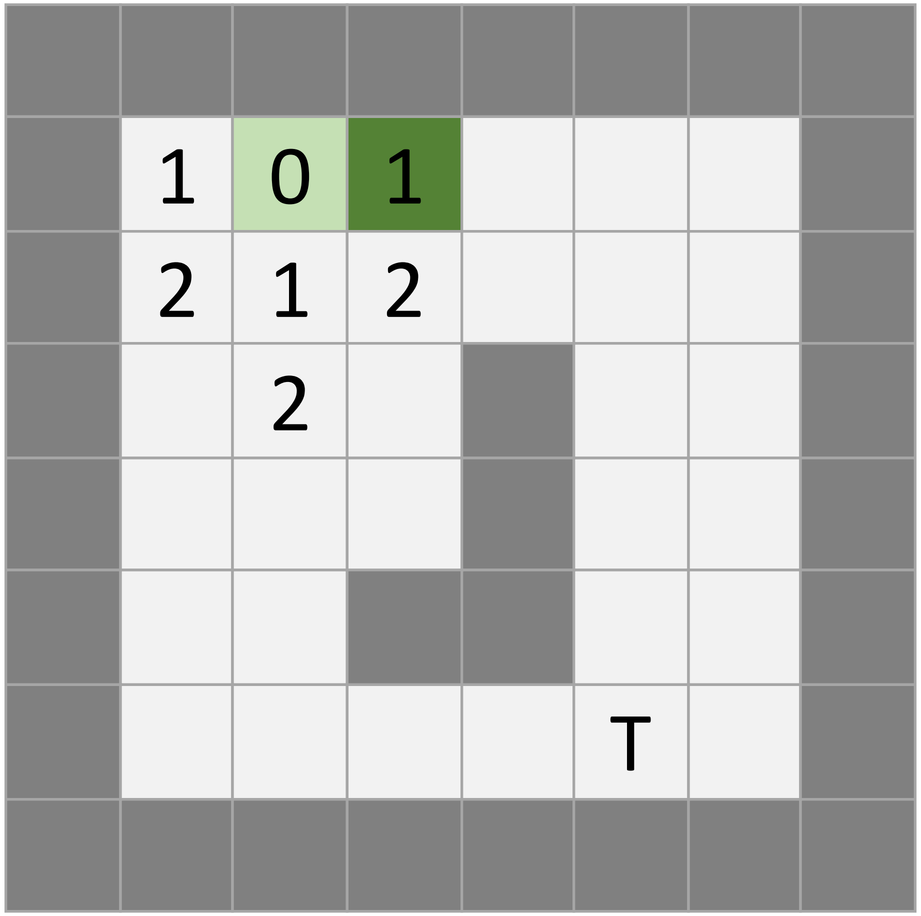
Row	Column
1	2
2	2
1	1
1	3
3	2
2	1
2	3
...	...

Faster Version



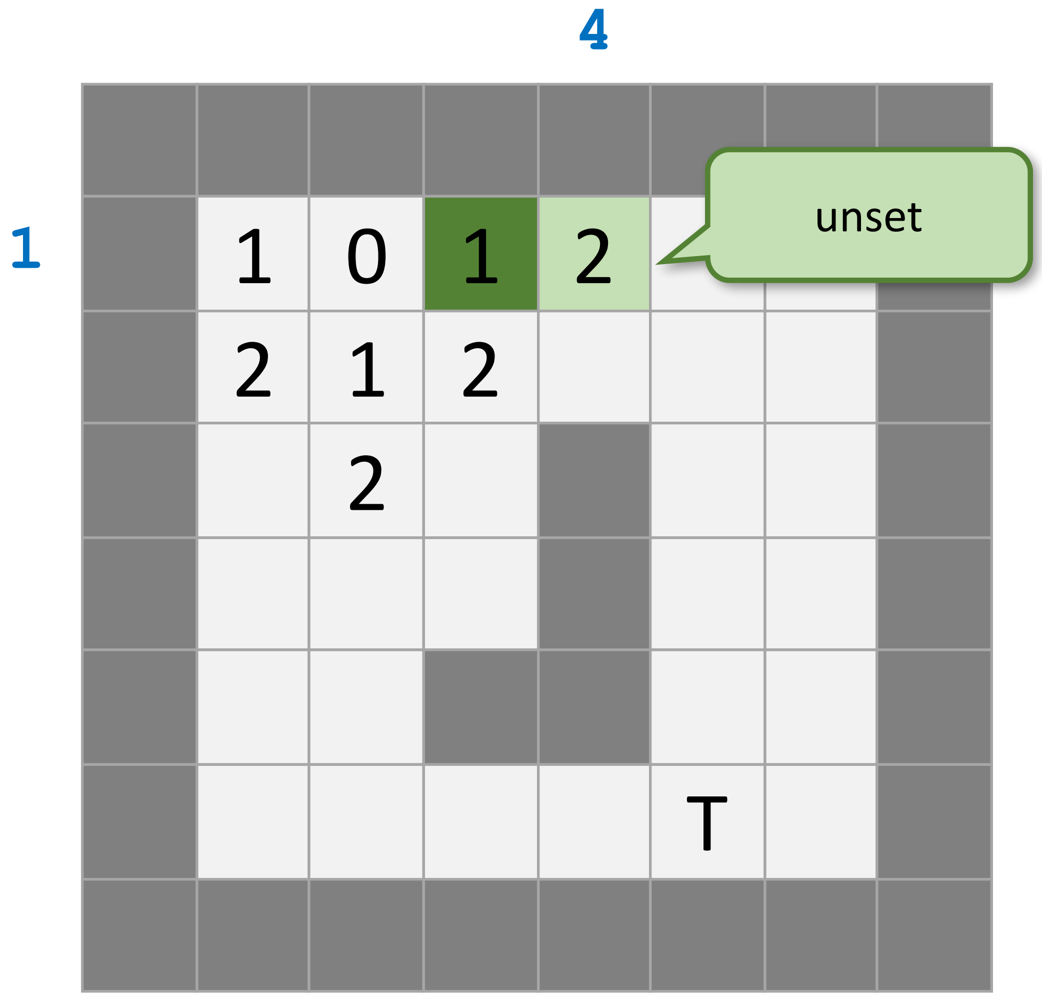
Row	Column
1	2
2	2
1	1
1	3
3	2
2	1
2	3
…	…

Faster Version



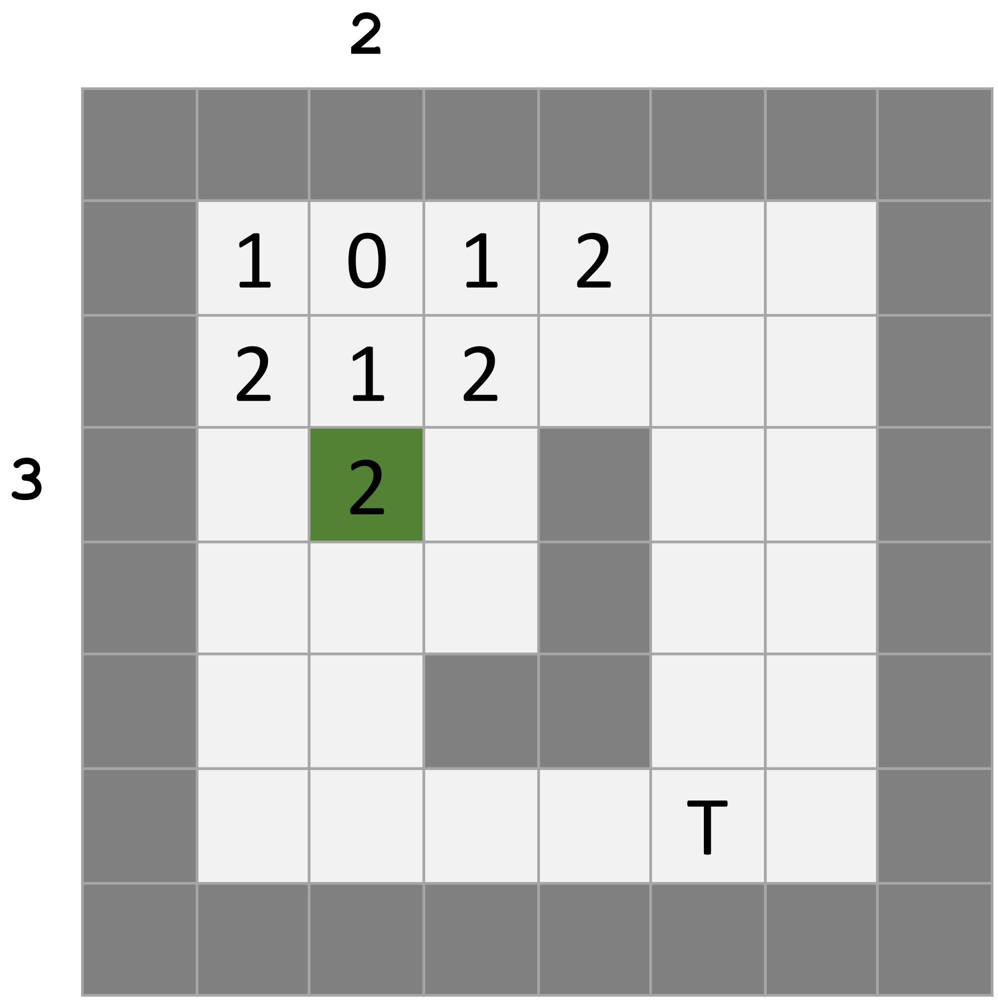
Row	Column
1	2
2	2
1	1
1	3
3	2
2	1
2	3
...	...

Faster Version



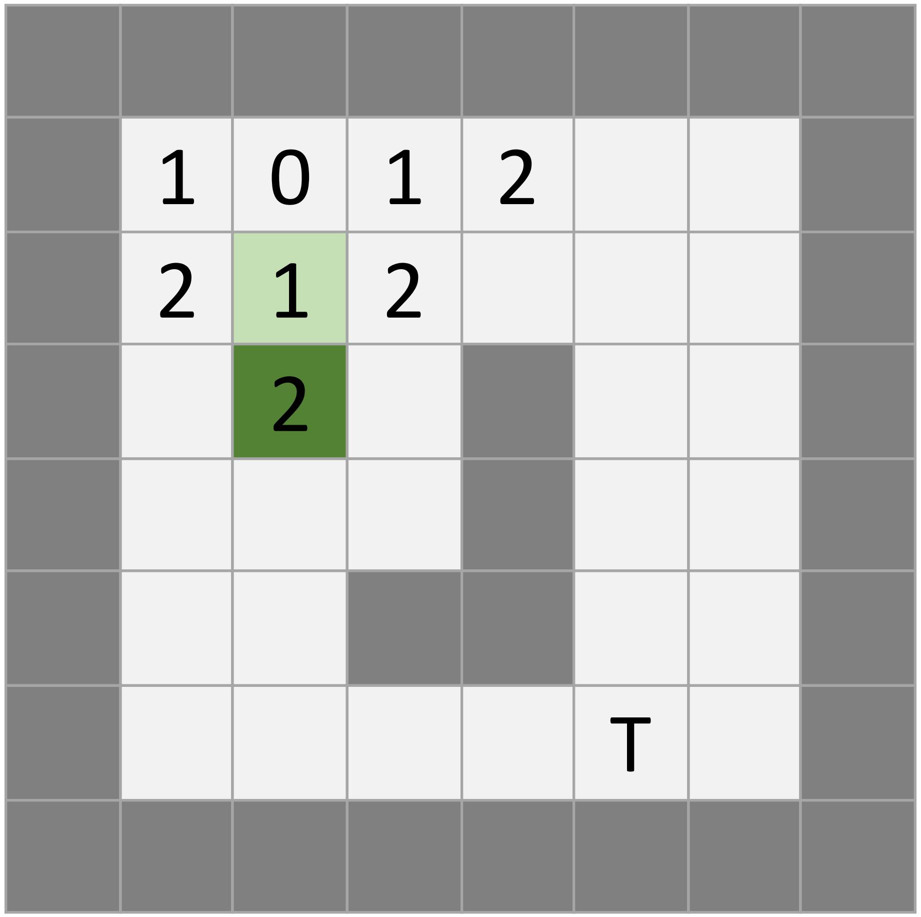
Row	Column
1	2
2	2
1	1
1	3
3	2
2	1
2	3
1	4
…	…

Faster Version



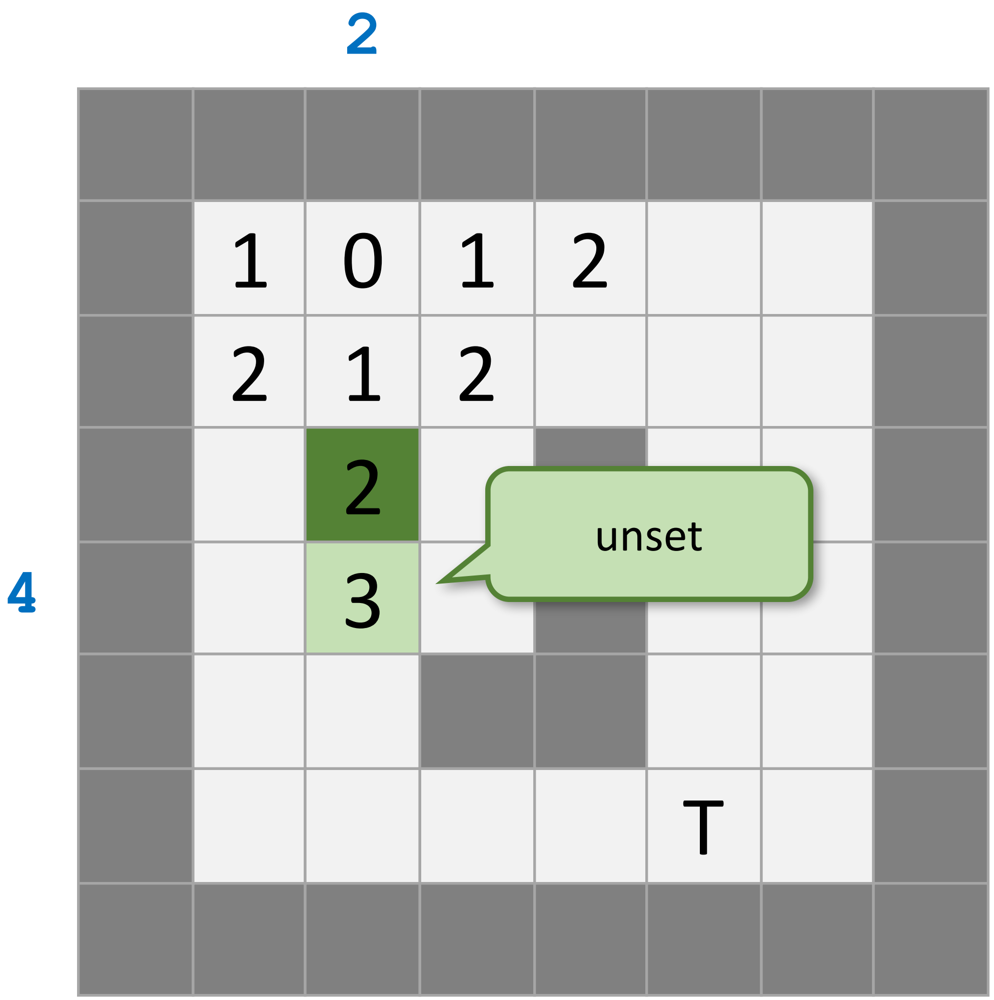
Row	Column
1	2
2	2
1	1
1	3
3	2
2	1
2	3
1	4
...	...

Faster Version



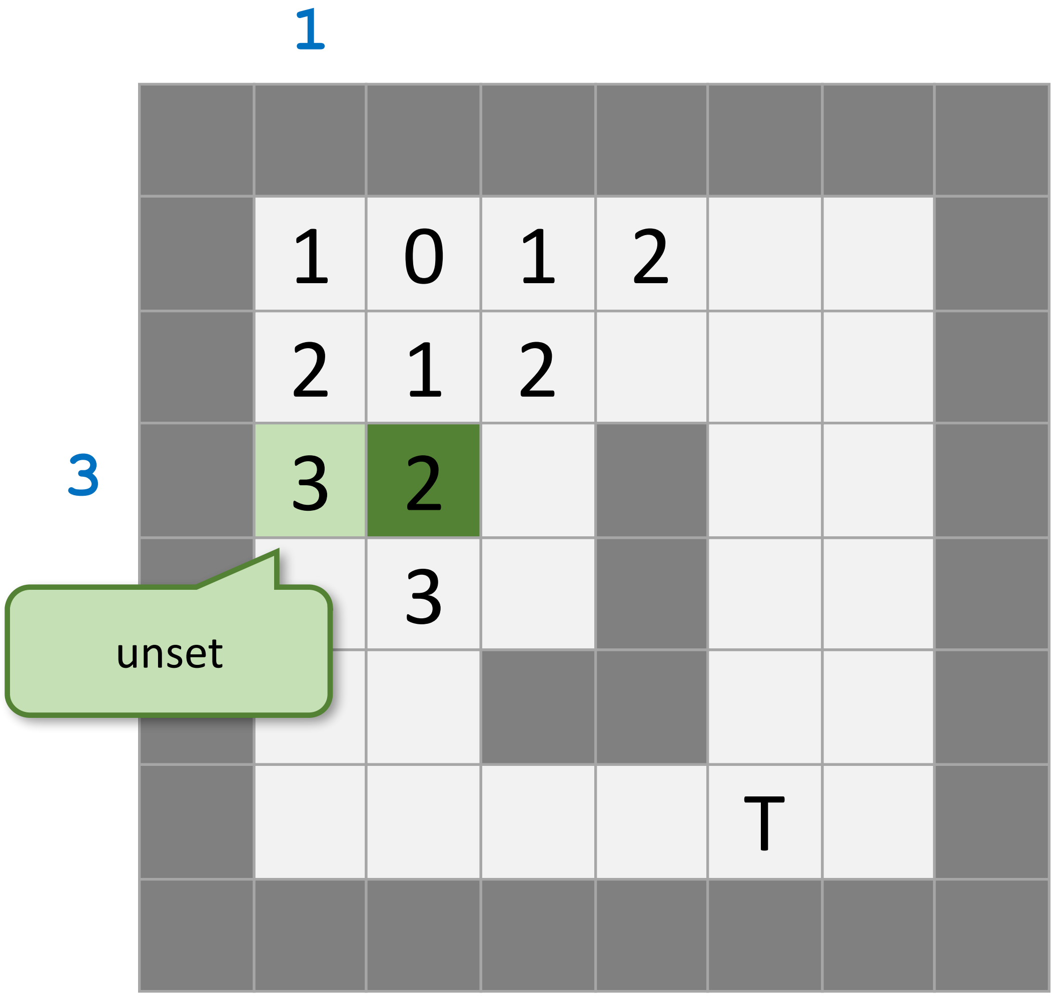
Row	Column
1	2
2	2
1	1
1	3
3	2
2	1
2	3
1	4
…	…

Faster Version



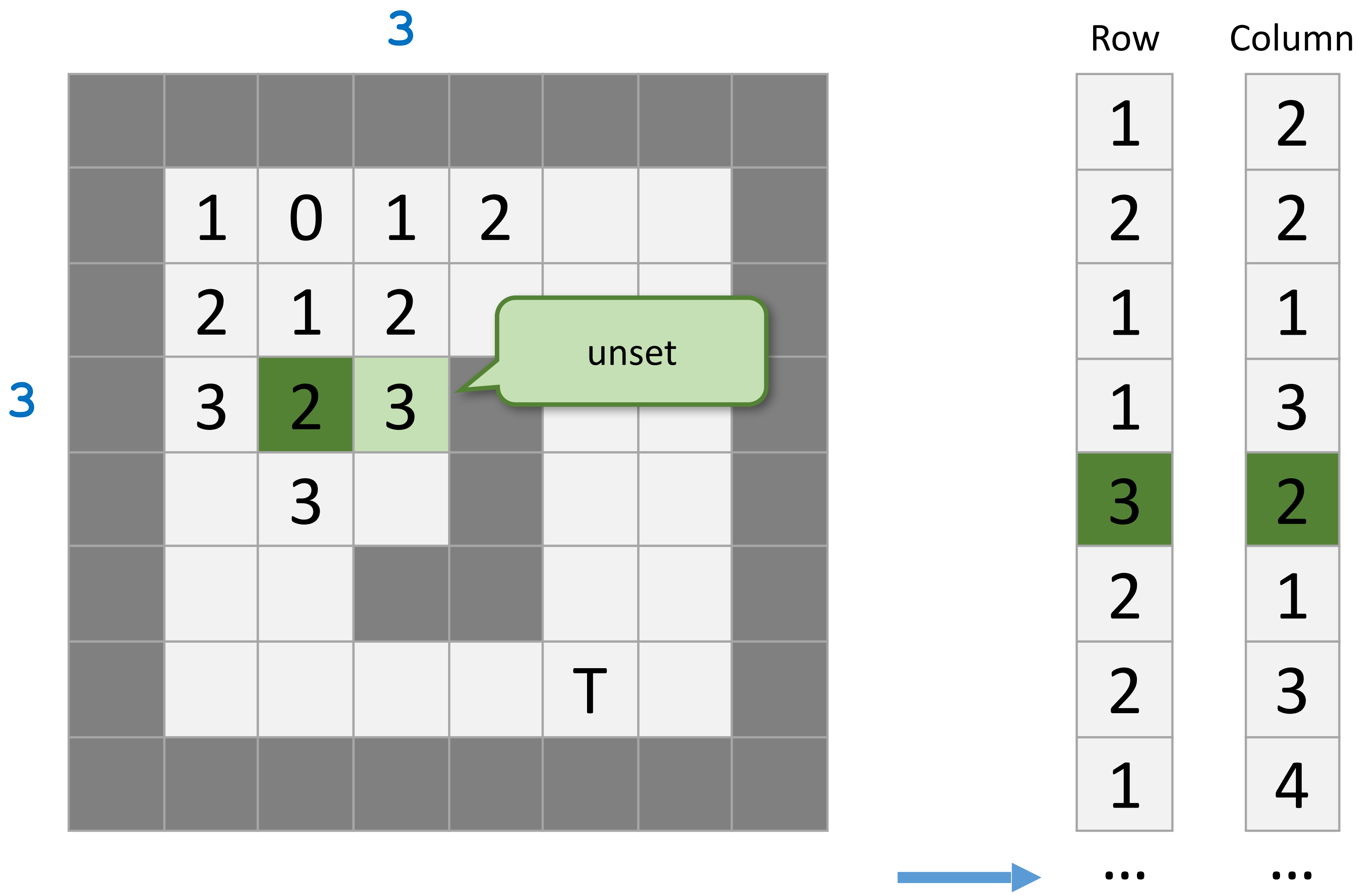
Row	Column
1	2
2	2
1	1
1	3
3	2
2	1
2	3
1	4
...	...

Faster Version

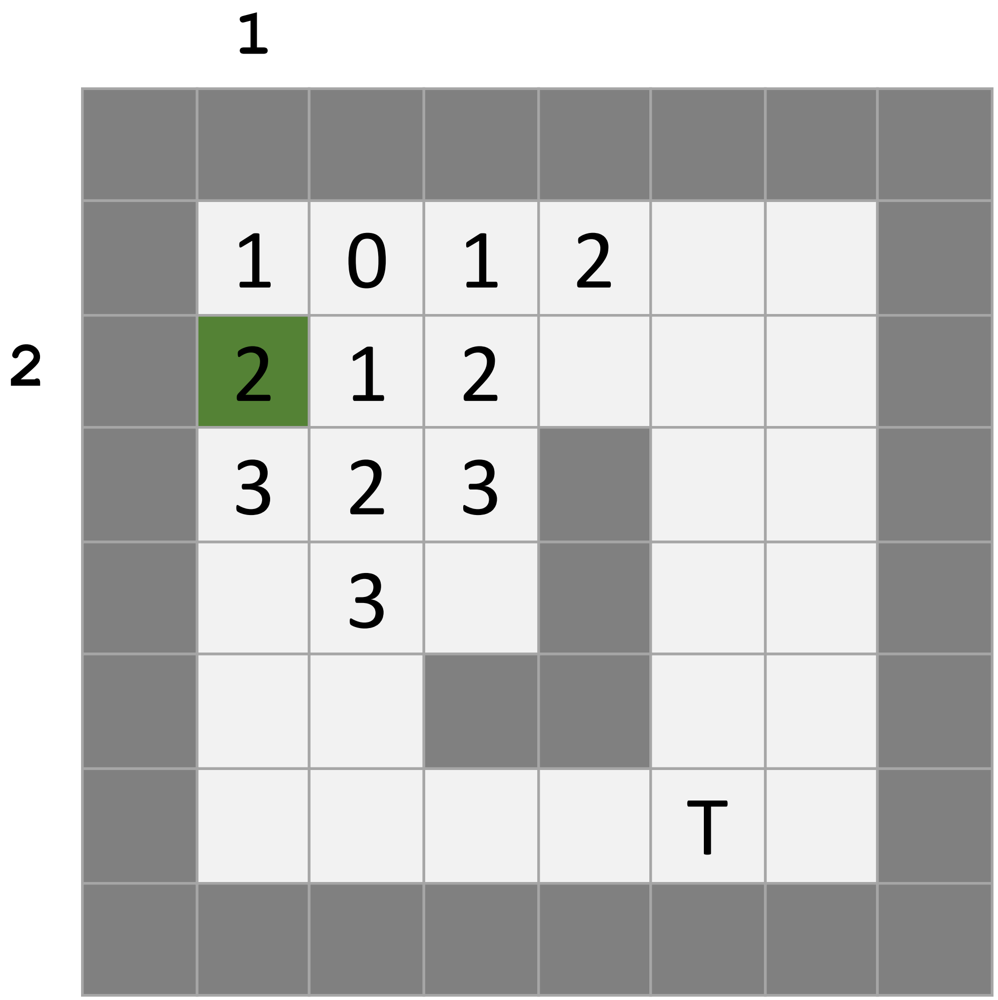


Row	Column
1	2
2	2
1	1
1	3
3	2
2	1
2	3
1	4
…	…

Faster Version

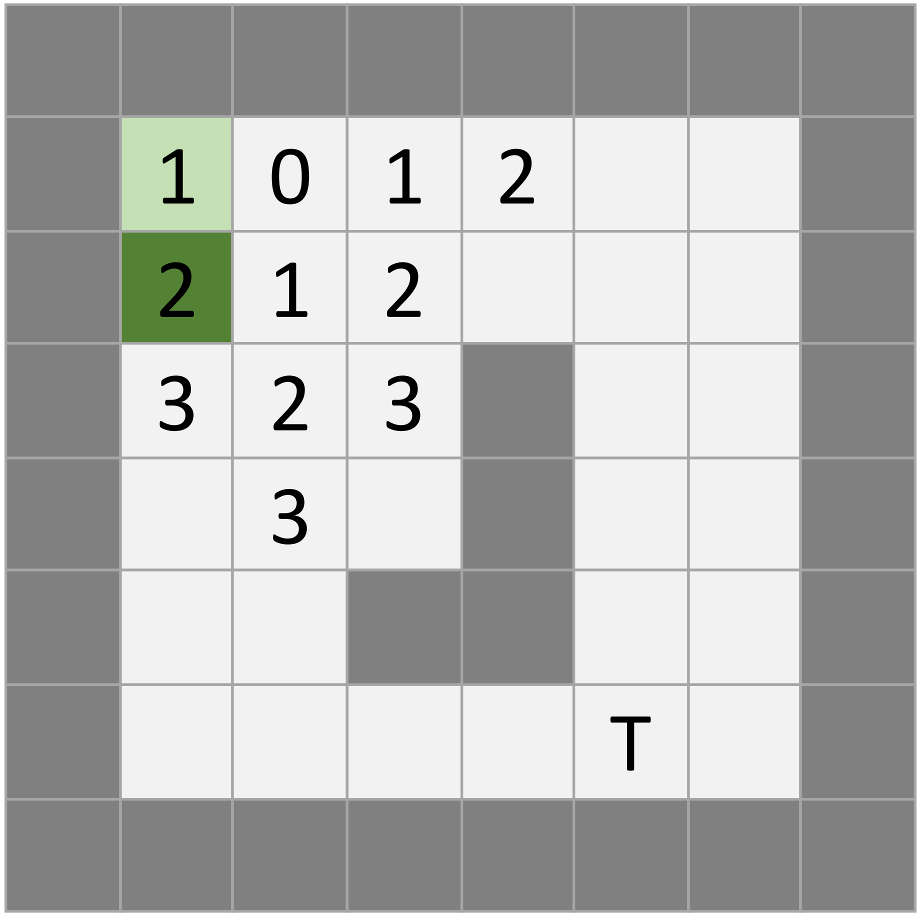


Faster Version



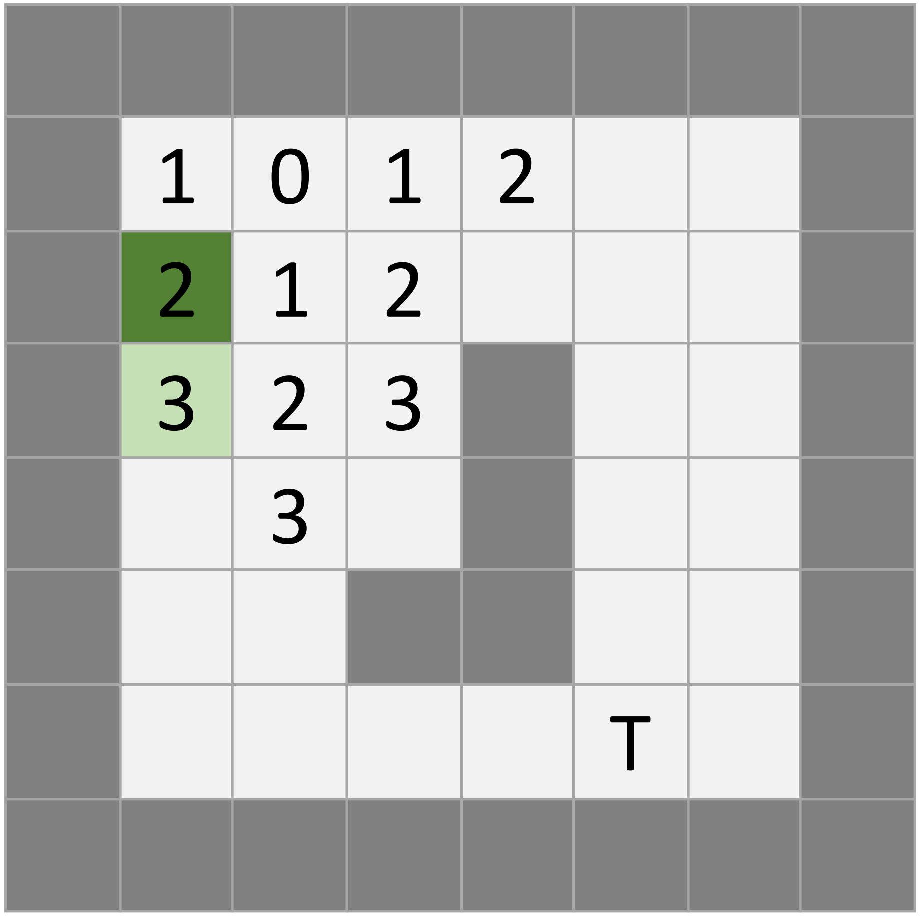
Row	Column
1	2
2	2
1	1
1	3
3	2
2	1
2	3
1	4
...	...

Faster Version



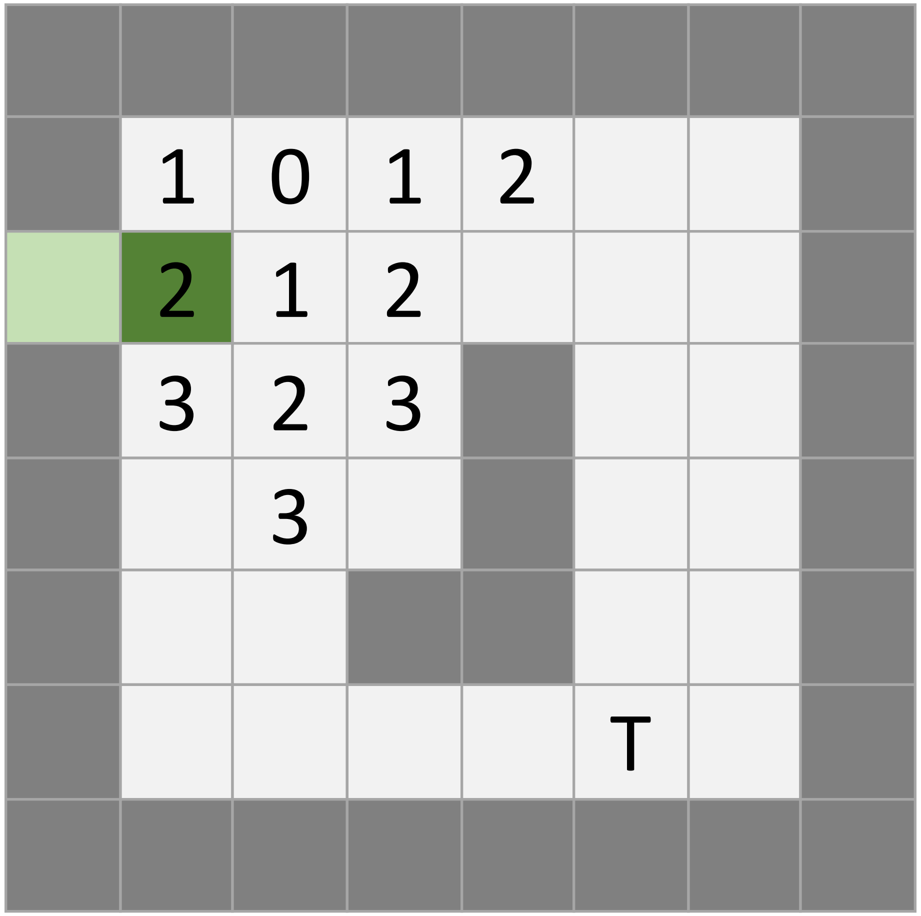
Row	Column
1	2
2	2
1	1
1	3
3	2
2	1
2	3
1	4
...	...

Faster Version



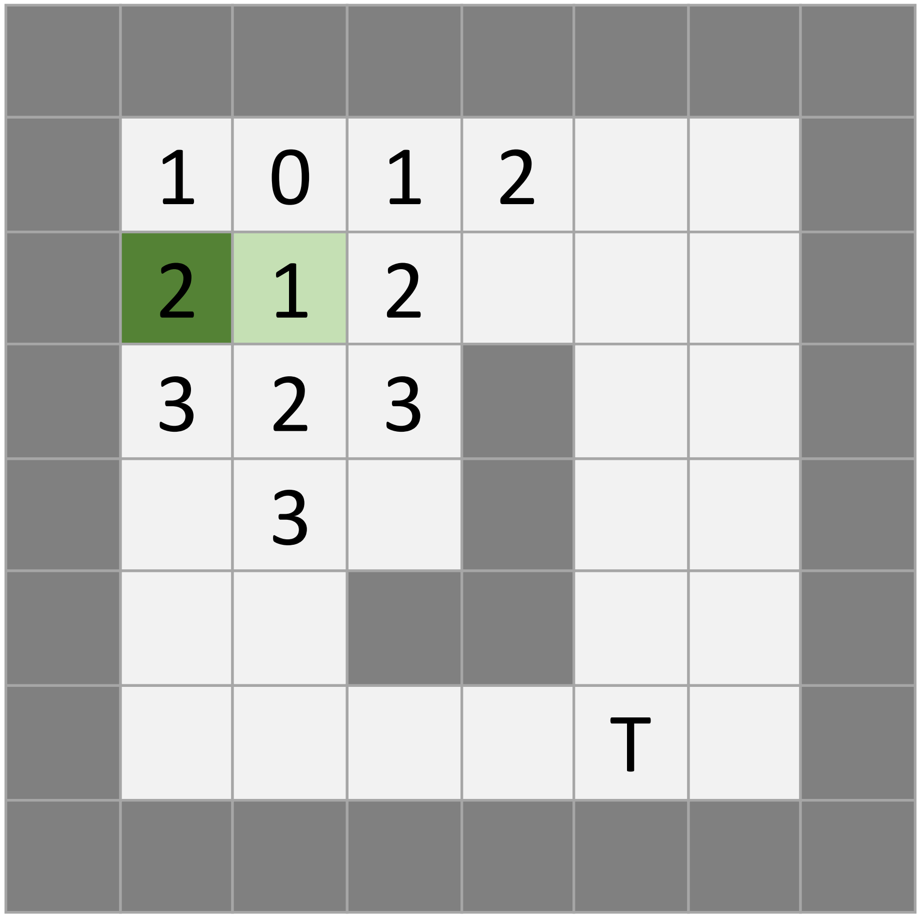
Row	Column
1	2
2	2
1	1
1	3
3	2
2	1
2	3
1	4
…	…

Faster Version



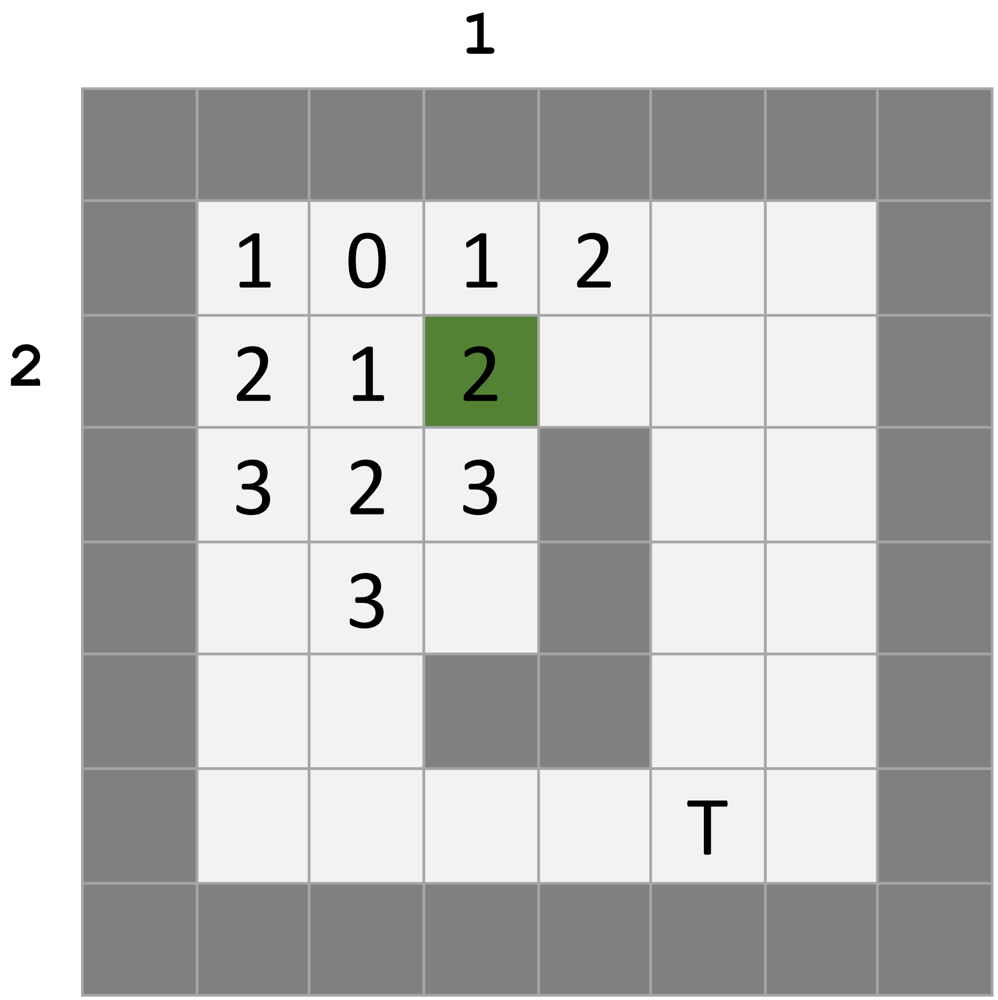
Row	Column
1	2
2	2
1	1
1	3
3	2
2	1
2	3
1	4
…	…

Faster Version



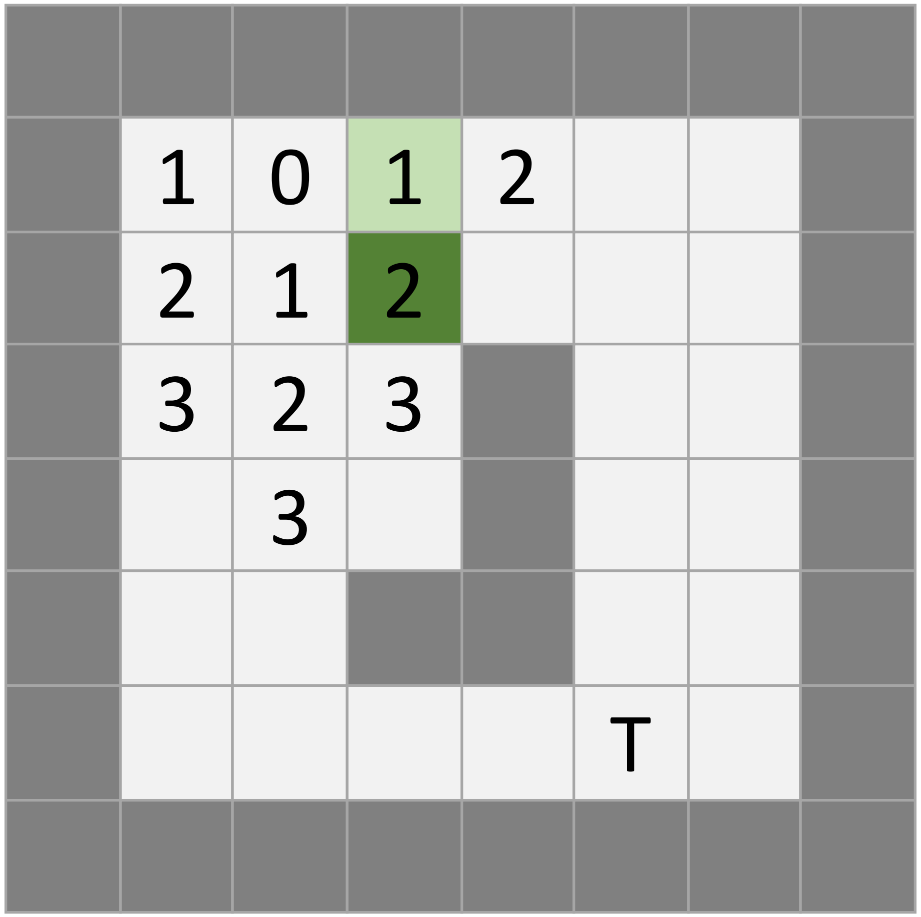
Row	Column
1	2
2	2
1	1
1	3
3	2
2	1
2	3
1	4
…	…

Faster Version



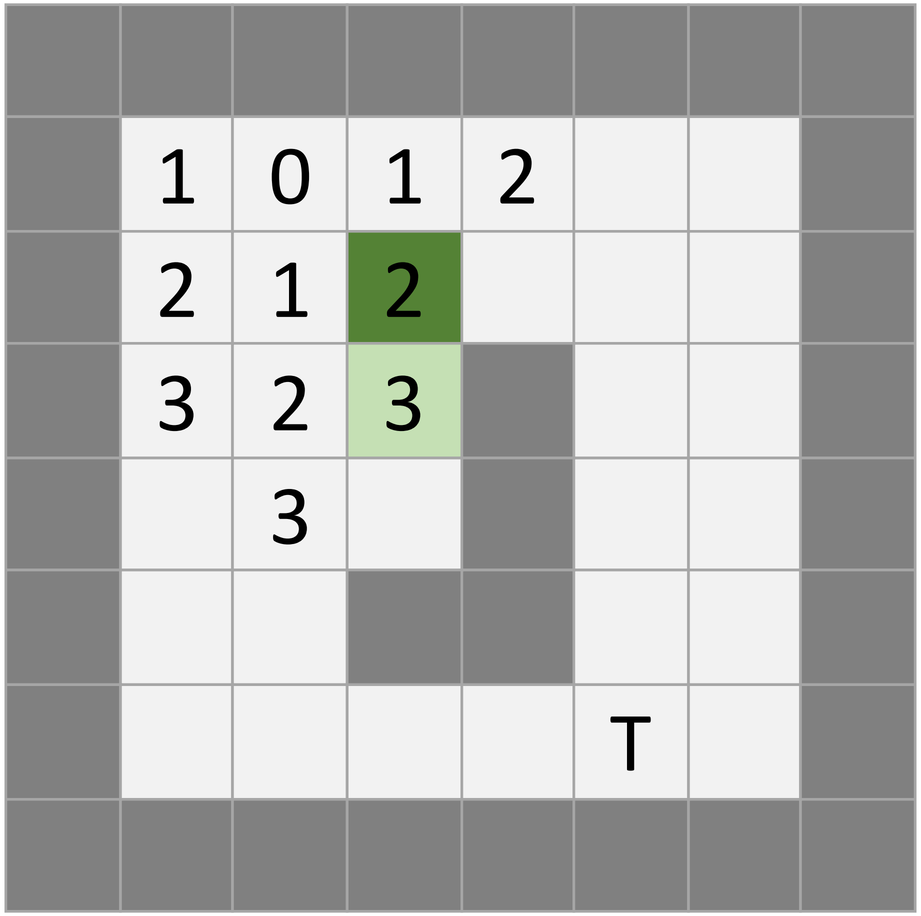
Row	Column
1	2
2	2
1	1
1	3
3	2
2	1
2	3
1	4
…	…

Faster Version



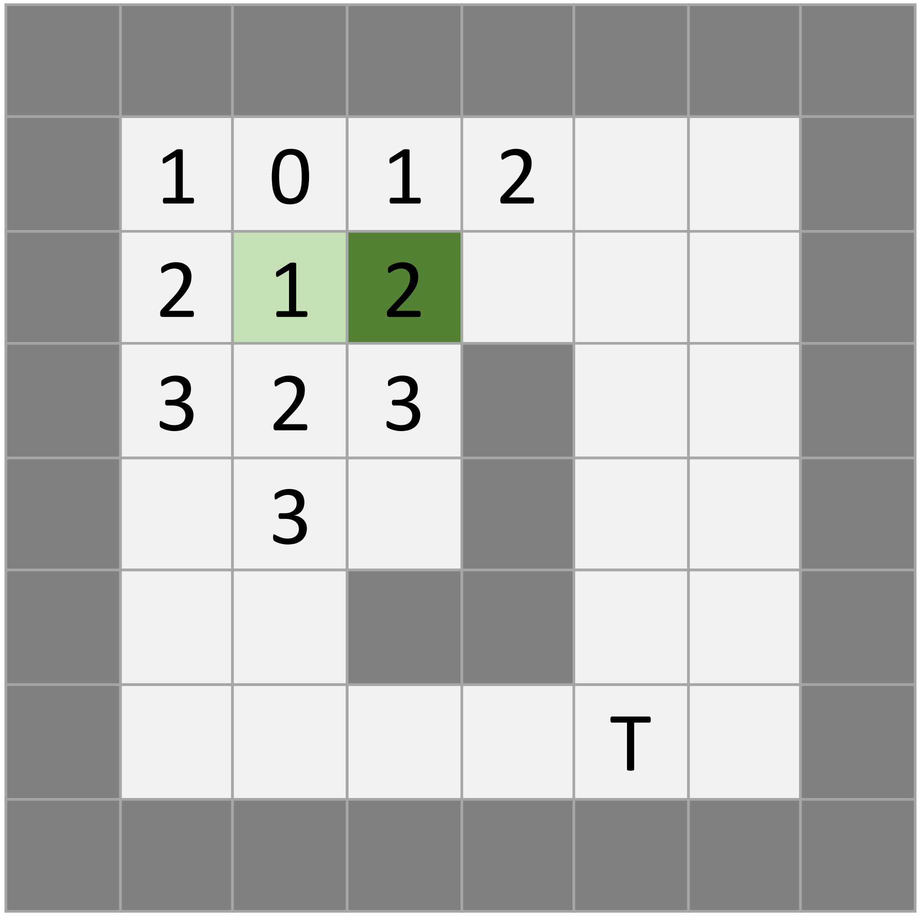
Row	Column
1	2
2	2
1	1
1	3
3	2
2	1
2	3
1	4
…	…

Faster Version



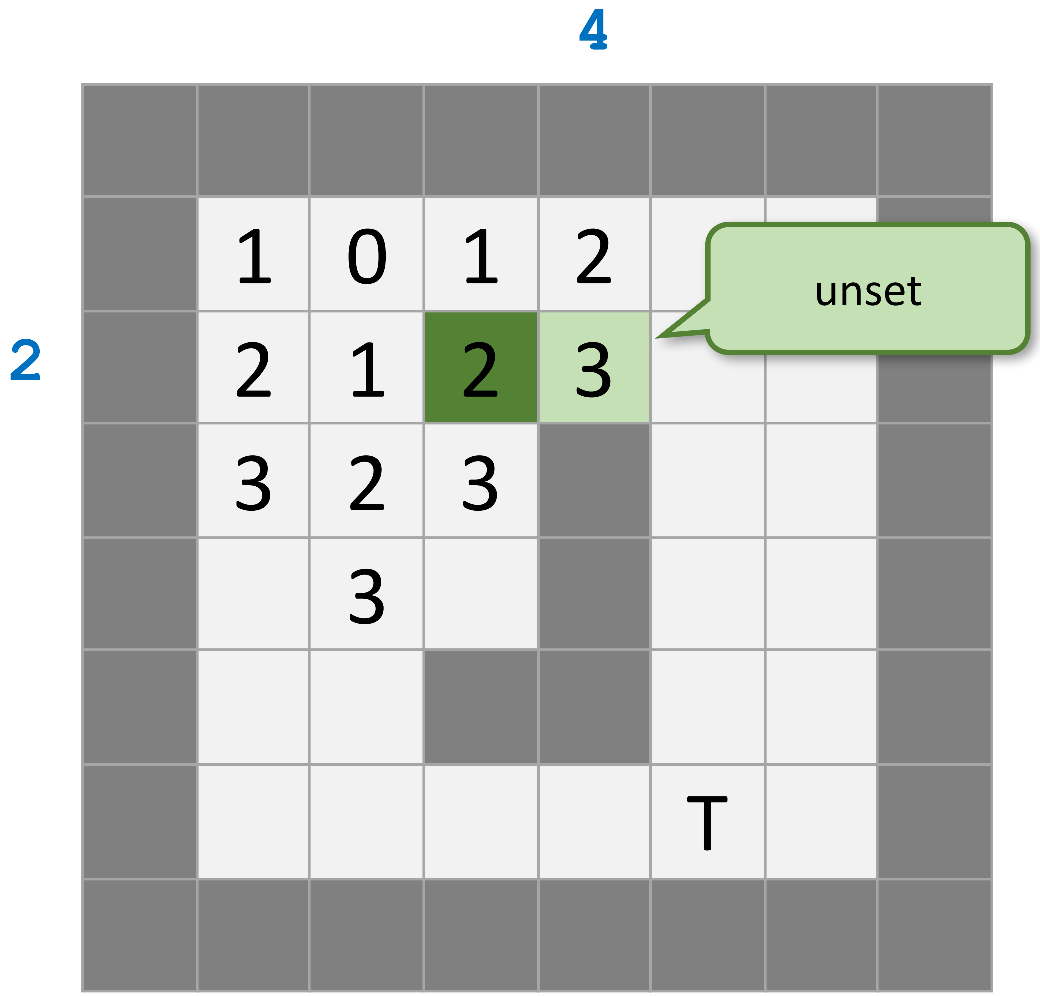
Row	Column
1	2
2	2
1	1
1	3
3	2
2	1
2	3
1	4
…	…

Faster Version



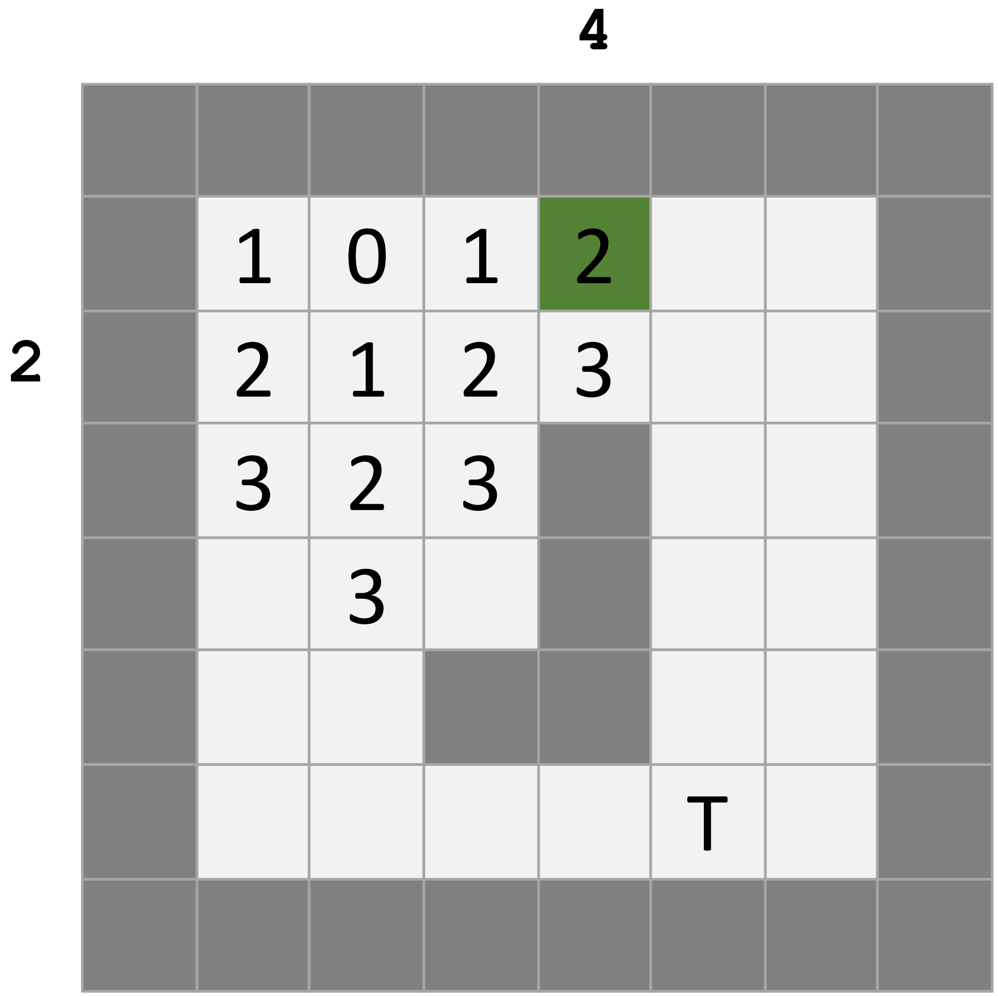
Row	Column
1	2
2	2
1	1
1	3
3	2
2	1
2	3
1	4
…	…

Faster Version



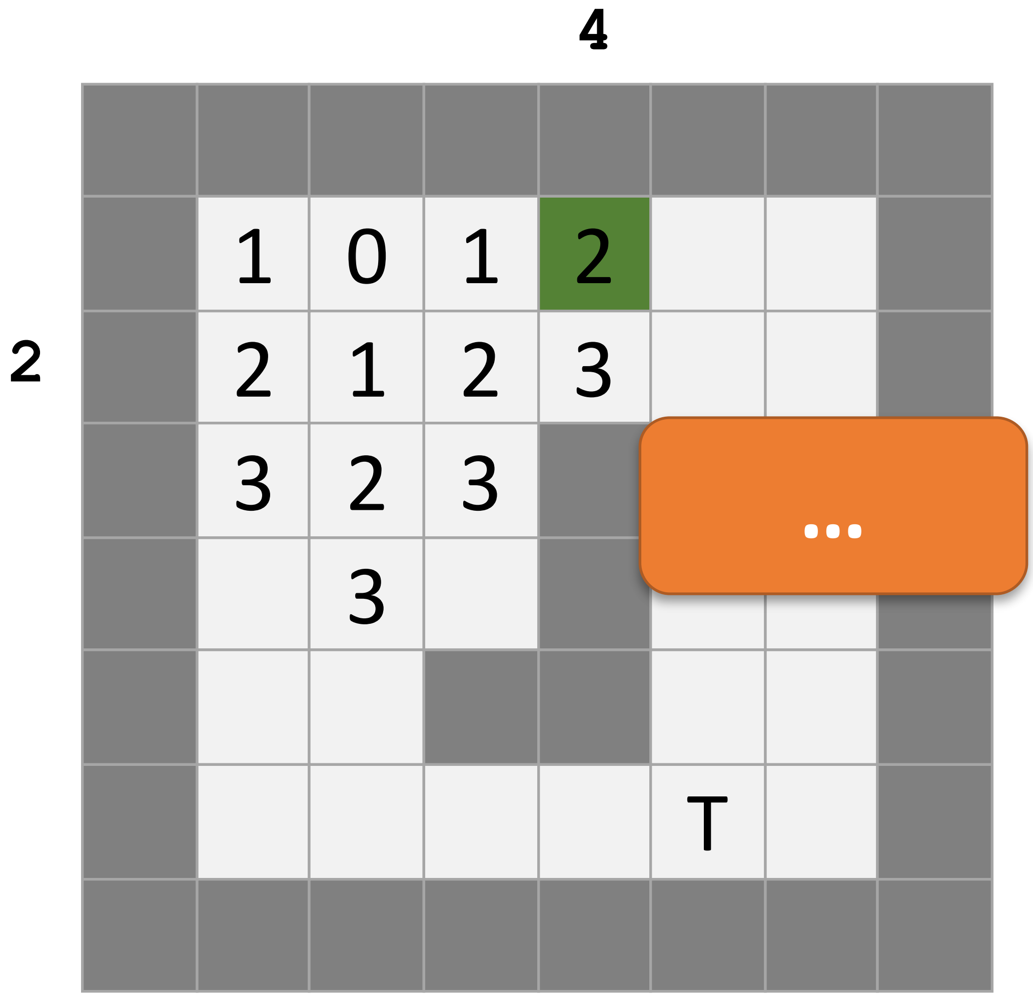
Row	Column
1	2
2	2
1	1
1	3
3	2
2	1
🟩 2	🟩 3
1	4
…	…

Faster Version



Row	Column
1	2
2	2
1	1
1	3
3	2
2	1
2	3
1	4
…	…

Faster Version



Row	Column
1	2
2	2
1	1
1	3
3	2
2	1
2	3
1	4
...	...

Faster Version

▨	▨	▨	▨	▨	▨	▨	▨
▨	1	0	1	2	3	4	▨
▨	2	1	2	3	4	5	▨
▨	3	2	3	▨	5	6	▨
▨	4	3	4	▨	6	7	▨
▨	5	4	▨	▨	7	8	▨
▨	6	5	6	7	8	9	▨
▨	▨	▨	▨	▨	▨	▨	▨

Row	Column
1	2
2	2
1	1
1	3
3	2
2	1
2	3
1	4
...	...

Agenda

- HW #6 Feedback
- Shortest path
- **Reading sequences of unknown lengths**
- Strings
- Lindenmayer Systems
- Pointers on arrays
- HW #8 Pre discussion

Reading input of unknown length

- Length of input is unknown
  - Example: a file with an unknown amount of integers.
- Read data from std::cin
  - std::cin >> x;
- The expression can be converted to a bool value: true if input is read, false for failed reading state.

Reading input of unknown length

```
#include <iostream>

int main () {

  int x;

  while(std::cin >> x){
    if (x % 2 == 0)
      std::cout << "The number " << x << " is even \n";
    else
      std::cout << "The number " << x << " is odd \n";
  }

  return 0;
}
```

Reading input of unknown length

- How does one read data from a file?
  - Input redirection
- Redirect input stream to a file when calling the program

```
./my_program < file.txt
```

- std::cin will now read from the file “file.txt”
- To stop: [Ctrl] + [D] (linux)

Agenda

- HW #6 Feedback
- Shortest path
- Reading sequences of unknown lengths
- **Strings**
- Lindenmayer Systems
- Pointers on arrays
- HW #8 Pre discussion

Characters - recap

`char` is a primitive C type which stores a single character.

```
char ch = 'z'; // !single! quotes

// read 5 chars from user and count number of 'n's

unsigned int counter = 0;
for (int i=0; i<5; ++i)
{
    std::cin >> ch;
    if (ch == 'n')
        ++counter;
}
std::cout << counter << "\n";
```

Characters

- Special characters
  - newline: `'\n'`
  - tab: `'\t'`
  - backslash: `'\\'`

- every character is actually represented by a number
  - 65 = 1000001 = `'A'` 97 = 1100001 = `'a'`
  - 66 = 1000010 = `'B'` 98 = 1100010 = `'b'`

Characters

```
char letter = 'a';
int number = letter;    // implicit type conversion: number = 97
```

```
int number = 66;
char letter = number;   // implicit type conversion: letter = 'B'
```

```
// convert from uppercase to lowercase
char a; std::cin >> a;


if ('A' <= a && a <= 'Z') {
   a = a + ('a'- 'A');      // assume fix difference
}
```

Strings

- Special vectors for storing text as a sequence of characters.

  ```
  ~~std::vector<char> my_char_vector;~~
  std::string my_string;
  ```

- Part of the standard library
- Provide special operators and functions: +=, .length(), ==,
- Usage:
  - std::string
  - #include <string>
- Initialization:

  ```
  std::string text;
  ```

Strings - Example

```
std::string text;
std::cin >> text;                           // reads in a text of arbitrary
                                            // length, for example "Hello"
text += " world!";                          // text now: "Hello world!"

std::string text2 = text;                   // initialization also works

if (text == text2)                          // comparisons
  std::cout << text2 << "\n";               // outputs:  Hello world!

std::cout << "Length: " << text.length() << "\n"; // outputs 12
```

Agenda

- HW #6 Feedback
- Shortest path
- Reading sequences of unknown lengths
- Strings
- **Lindenmayer Systems**
- Pointers on arrays
- HW #8 Pre discussion

Turtle Plots

Moving the Turtle

- Idea: trace walk-path

Moving the Turtle

- Idea: trace walk-path

C++ Easy Commands

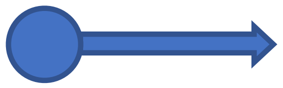
• Step (drawn):	`turtle::forward();`
• Rotation left:	`turtle::left(my_angle);`
• Rotation right:	`turtle::right(my_angle);`

Requires:
a) `#include "turtle.cpp"`
b) `turtle.cpp` and `bitmap.cpp` have to be in the same folder as your program.

turtle.cpp and bitmap.cpp can be downloaded from the lecture website. Furthermore, there are additional turtle commands available, see Turtle_Extended.pdf

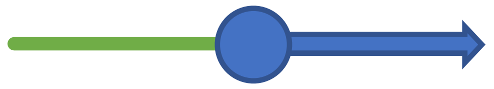
Moving the Turtle

```
turtle::forward();
turtle::left(45);
turtle::forward();
turtle::right(90);
turtle::forward();
```



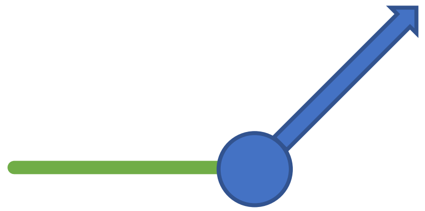
Moving the Turtle

```
turtle::forward();
turtle::left(45);
turtle::forward();
turtle::right(90);
turtle::forward();
```



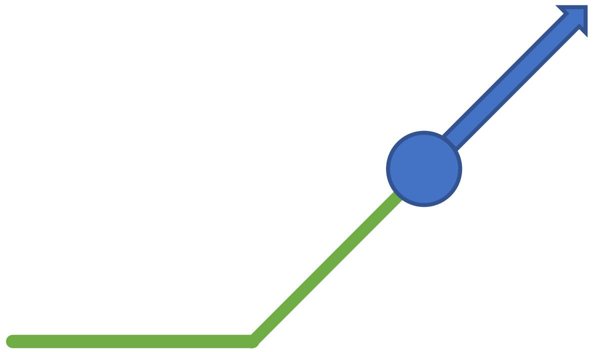
Moving the Turtle

```
turtle::forward();
turtle::left(45);
turtle::forward();
turtle::right(90);
turtle::forward();
```



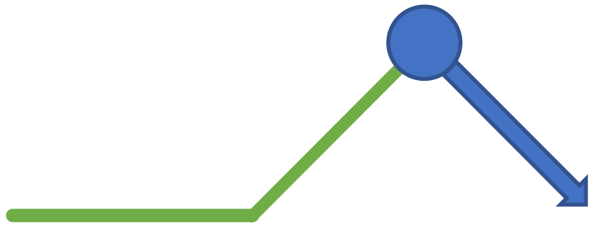
Moving the Turtle

```
turtle::forward();
turtle::left(45);
turtle::forward();
turtle::right(90);
turtle::forward();
```



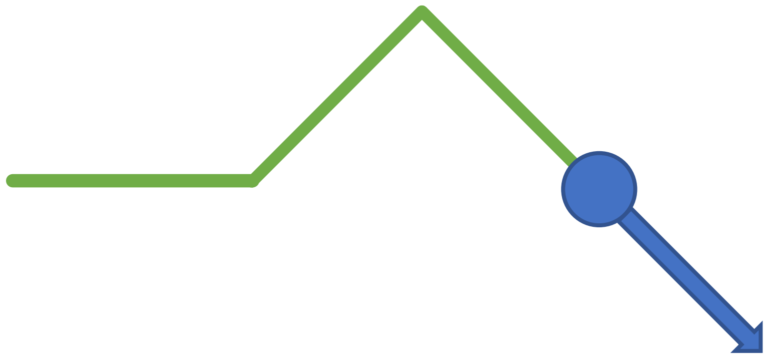
Moving the Turtle

```
turtle::forward();
turtle::left(45);
turtle::forward();
turtle::right(90);
turtle::forward();
```



Moving the Turtle

```
turtle::forward();
turtle::left(45);
turtle::forward();
turtle::right(90);
turtle::forward();
```



Lindenmayer Systems

Lindenmayer Systems

- Characterized by three things:

1.	Alphabet Σ	-	the allowed symbols
2.	Production P	-	how to replace each symbol
3.	Initial word s	-	the word to start with

- Example:

1. $\Sigma := \{F, +, -\}$
2. $P := \begin{cases} F \mapsto F+F+ \\ + \mapsto + \\ - \mapsto - \end{cases}$
3. $s := F$

Lindenmayer Systems

- Characterized by three things:

1.	Alphabet Σ	-	the allowed symbols
2.	Production P	-	how to replace each symbol
3.	Initial word s	-	the word to start with

- Example:

1. $\Sigma := \{F, +, -\}$
2. $P := \begin{cases} F & \mapsto F + F + \\ + & \mapsto + \\ - & \mapsto - \end{cases}$
3. $s := F$

Lindenmayer Systems

- How does it look after 3 rounds?

1. $\Sigma := \{F, +, -\}$
2. $P := \begin{cases} F \mapsto F+F+ \\ + \mapsto + \\ - \mapsto - \end{cases}$
3. $s := F$

s: F

w_1:

w_2:

w_3:

Lindenmayer Systems

- How does it look after 3 rounds?

1. $\Sigma := \{F, +, -\}$

2. $P := \begin{cases} F \mapsto F+F+ \\ + \mapsto + \\ - \mapsto - \end{cases}$

3. $s := F$

s: F

w_1: F+F+

w_2:

w_3:

Lindenmayer Systems

- How does it look after 3 rounds?

1. $\Sigma := \{F, +, -\}$
2. $P := \begin{cases} F \mapsto F+F+ \\ + \mapsto + \\ - \mapsto - \end{cases}$
3. $s := F$

s: F

w_1: F+F+

w_2:

w_3:

Lindenmayer Systems

- How does it look after 3 rounds?

1. $\Sigma := \{F, +, -\}$
2. $P := \begin{cases} F \mapsto F + F + \\ + \mapsto + \\ - \mapsto - \end{cases}$
3. $s := F$

s:	F
w_1:	F+F+
w_2:	F+F+
w_3:	

Lindenmayer Systems

- How does it look after 3 rounds?

1. $\Sigma := \{F, +, -\}$

2. $P := \begin{cases} F \mapsto F + F + \\ + \mapsto + \\ - \mapsto - \end{cases}$

3. $s := F$

s: F

w_1: F+F+

w_2: F+F++

w_3:

Lindenmayer Systems

- How does it look after 3 rounds?

1. $\Sigma := \{F, +, -\}$
2. $P := \begin{cases} F \mapsto F+F+ \\ + \mapsto + \\ - \mapsto - \end{cases}$
3. $s := F$

s: F

w_1: F+F+

w_2: F+F++F+F+

w_3:

Lindenmayer Systems

- How does it look after 3 rounds?

1. $\Sigma := \{F, +, -\}$
2. $P := \begin{cases} F \mapsto F + F + \\ + \mapsto + \\ - \mapsto - \end{cases}$
3. $s := F$

s: F

w_1: F+F+

w_2: F+F++F+F++

w_3:

Lindenmayer Systems

- How does it look after 3 rounds?

1. $\Sigma := \{F, +, -\}$
2. $P := \begin{cases} F \mapsto F + F + \\ + \mapsto + \\ - \mapsto - \end{cases}$
3. $s := F$

s:	F
w_1:	F+F+
w_2:	F+F++F+F++
w_3:	

Lindenmayer Systems

- How does it look after 3 rounds?

1. $\Sigma := \{F, +, -\}$
2. $P := \begin{cases} F \mapsto F+F+ \\ + \mapsto + \\ - \mapsto - \end{cases}$
3. $s := F$

s:	F
w_1:	F+F+
w_2:	F+F++F+F++
w_3:	F+F+

Lindenmayer Systems

- How does it look after 3 rounds?

1. $\Sigma := \{F, +, -\}$
2. $P := \begin{cases} F \mapsto F+F+ \\ + \mapsto + \\ - \mapsto - \end{cases}$
3. $s := F$

s: `F`

w_1: `F+F+`

w_2: `F+F++F+F++`

w_3: `F+F++`

Lindenmayer Systems

- How does it look after 3 rounds?

1. $\Sigma := \{F, +, -\}$
2. $P := \begin{cases} F \mapsto F+F+ \\ + \mapsto + \\ - \mapsto - \end{cases}$
3. $s := F$

s:	F
w_1:	F+F+
w_2:	F+F++F+F++
w_3:	F+F++F+F+

Lindenmayer Systems

- How does it look after 3 rounds?

1. $\Sigma := \{F, +, -\}$
2. $P := \begin{cases} F \mapsto F+F+ \\ + \mapsto + \\ - \mapsto - \end{cases}$
3. $s := F$

s: F

w_1: F+F+

w_2: F+F++F+F++

w_3: F+F++F+F++

Lindenmayer Systems

- How does it look after 3 rounds?

1. $\Sigma := \{F, +, -\}$
2. $P := \begin{cases} F \mapsto F+F+ \\ + \mapsto + \\ - \mapsto - \end{cases}$
3. $s := F$

s:	F
w_1:	F+F+
w_2:	F+F+++F+F++
w_3:	F+F+++F+F++++

Lindenmayer Systems

- How does it look after 3 rounds?

1. $\Sigma := \{F, +, -\}$
2. $P := \begin{cases} F \mapsto F + F + \\ + \mapsto + \\ - \mapsto - \end{cases}$
3. $s := F$

s:	F
w_1:	F+F+
w_2:	F+F++F+F++
w_3:	F+F++F+F+++F+F+

Lindenmayer Systems

- How does it look after 3 rounds?

1. $\Sigma \coloneqq \{F, +, -\}$
2. $P \coloneqq \begin{cases} F \mapsto F+F+ \\ + \mapsto + \\ - \mapsto - \end{cases}$
3. $s \coloneqq F$

s:	F
w_1:	F+F+
w_2:	F+F++F+F++
w_3:	F+F++F+F+++F+F++

Lindenmayer Systems

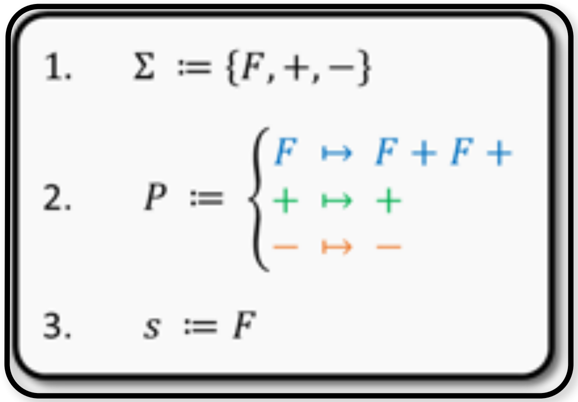
- How does it look after 3 rounds?

1. $\Sigma := \{F, +, -\}$
2. $P := \begin{cases} F \mapsto F+F+ \\ + \mapsto + \\ - \mapsto - \end{cases}$
3. $s := F$

s:	F
w_1:	F+F+
w_2:	F+F++F+F++
w_3:	F+F++F+F+++F+F++F+F+

Lindenmayer Systems

- How does it look after 3 rounds?



1. $\Sigma := \{F, +, -\}$
2. $P := \begin{cases} F \mapsto F + F + \\ + \mapsto + \\ - \mapsto - \end{cases}$
3. $s := F$

s:	F
w_1:	F+F+
w_2:	F+F++F+F++
w_3:	F+F++F+F+++F+F++F+F++

Lindenmayer Systems

- How does it look after 3 rounds?

1. $\Sigma := \{F, +, -\}$
2. $P := \begin{cases} F \mapsto F+F+ \\ + \mapsto + \\ - \mapsto - \end{cases}$
3. $s := F$

s:	F
w_1:	F+F+
w_2:	F+F++F+F++
w_3:	F+F++F+F+++F+F++F+F+++

Lindenmayer Systems

- How does it look after 3 rounds?

1. $\Sigma := \{F, +, -\}$
2. $P := \begin{cases} F \mapsto F+F+ \\ + \mapsto + \\ - \mapsto - \end{cases}$
3. $s := F$

s: F

w_1: F+F+

w_2: F+F++F+F++

w_3: F+F++F+F+++F+F++F+F+++

Draw Lindenmayer Systems

Two Step Procedure

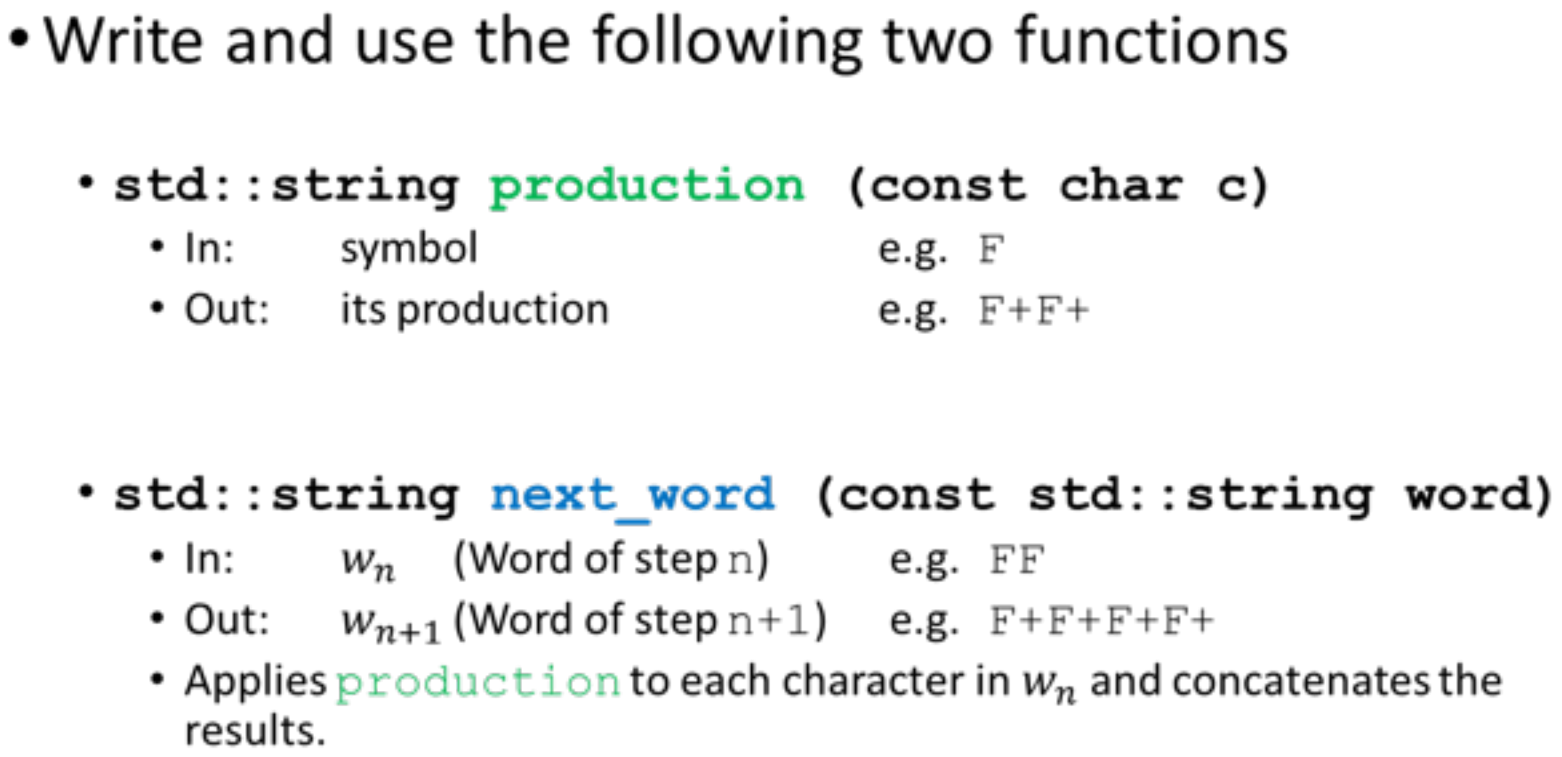
- Goal: Draw n-th step of Lindenmayer system

- Done in 2 steps
  1. Obtain n-th step
  2. Draw it

Step 1 – Obtain n-th Word

- Write and use the following two functions
  - `std::string production (const char c)`
    - In: symbol e.g. `F`
    - Out: its production e.g. `F+F+`

Step 1 – Obtain n-th Word



- Write and use the following two functions
  - **`std::string production (const char c)`**
    - In: symbol e.g. `F`
    - Out: its production e.g. `F+F+`
  - **`std::string next_word (const std::string word)`**
    - In: w_n (Word of step `n`) e.g. `FF`
    - Out: w_{n+1} (Word of step `n+1`) e.g. `F+F+F+F+`
    - Applies `production` to each character in w_n and concatenates the results.

Step 2 – Draw It

- Idea: view alphabet as turtle commands

- Example:

Alphabet: $\Sigma := \{F,\ +\ ,\ -\}$

F	`turtle::forward()`
$+$	`turtle::left(90)`
$-$	`turtle::right(90)`

Agenda

- HW #6 Feedback
- Shortest path
- Reading sequences of unknown lengths
- Strings
- Lindenmayer Systems
- **Pointers on arrays**
- HW #8 Pre discussion

Array to pointer conversion

- The address of the first element in the array can be implicitly converted to a pointer:

```
int arr[] = {7,1,0,2,5};

int* point = arr;    // arr gets converted to the address of the first array element a[0]

std::cout << *point << "\n";           // outputs 7

std::cout << *(point + 3) << "\n";     // outputs 2
```

- Can also use: `&arr[0]`

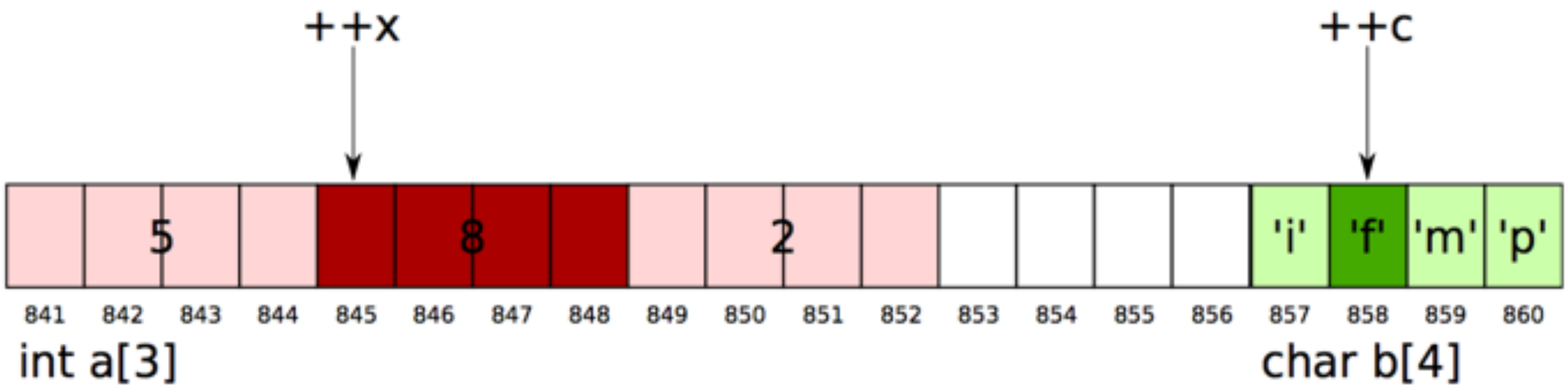
Pointer arithmetics

- Advancing a pointer:

  int * ptr = arr;

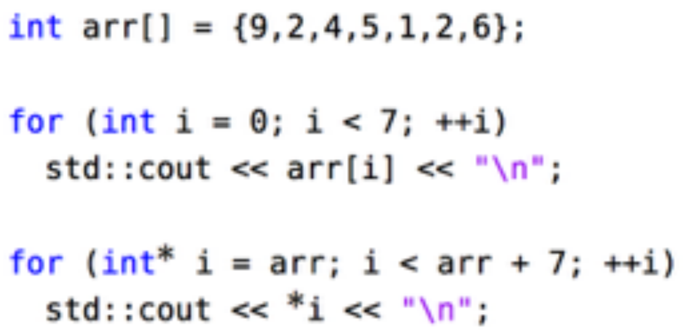
  ++ptr;

- Pointers point to a certain type - the type dedicates the amount of memory storage for each variable / array



Array to pointer conversion

- Pointers point to a certain type - the type dedicates the amount of memory storage for each variable / array element



```
int arr[] = {9,2,4,5,1,2,6};

for (int i = 0; i < 7; ++i)
  std::cout << arr[i] << "\n";

for (int* i = arr; i < arr + 7; ++i)
  std::cout << *i << "\n";
```

- Caution: arr + 7 points to the first element after the array, but it is never accessed.

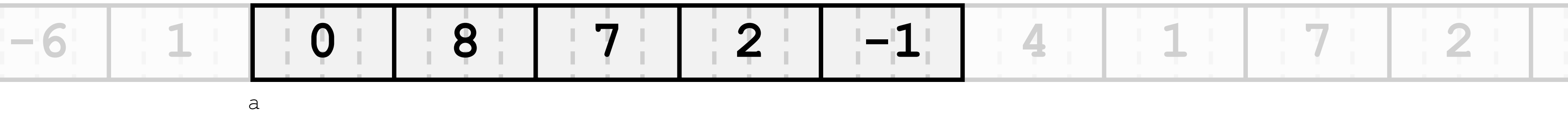
Pointers on Arrays

Pointer Program

```
int a[5] = {0, 8, 7, 2, -1};
int* ptr = a;                      // array-to-pointer conv
++ptr;                             // shift to the right
int my_int = *ptr;                 // read target
ptr += 2;                          // shift by 2 elements
*ptr = 18;                         // overwrite target
int* past = a+5;
std::cout << (ptr < past) << "\n"; // compare pointers
```

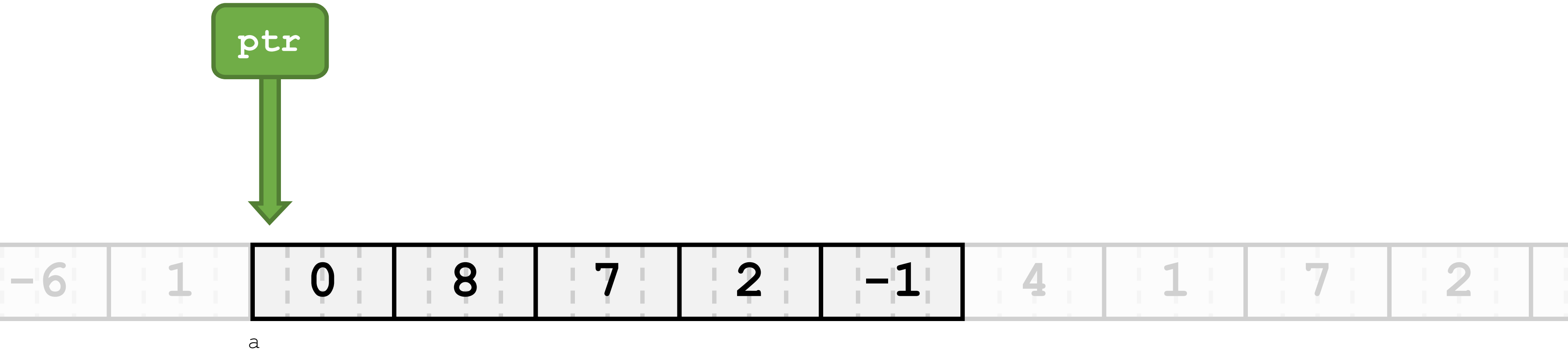
Pointer Program

```
int a[5] = {0, 8, 7, 2, -1};
int* ptr = a;                        // array-to-pointer conv
++ptr;                               // shift to the right
int my_int = *ptr;                   // read target
ptr += 2;                            // shift by 2 elements
*ptr = 18;                           // overwrite target
int* past = a+5;
std::cout << (ptr < past) << "\n"; // compare pointers
```



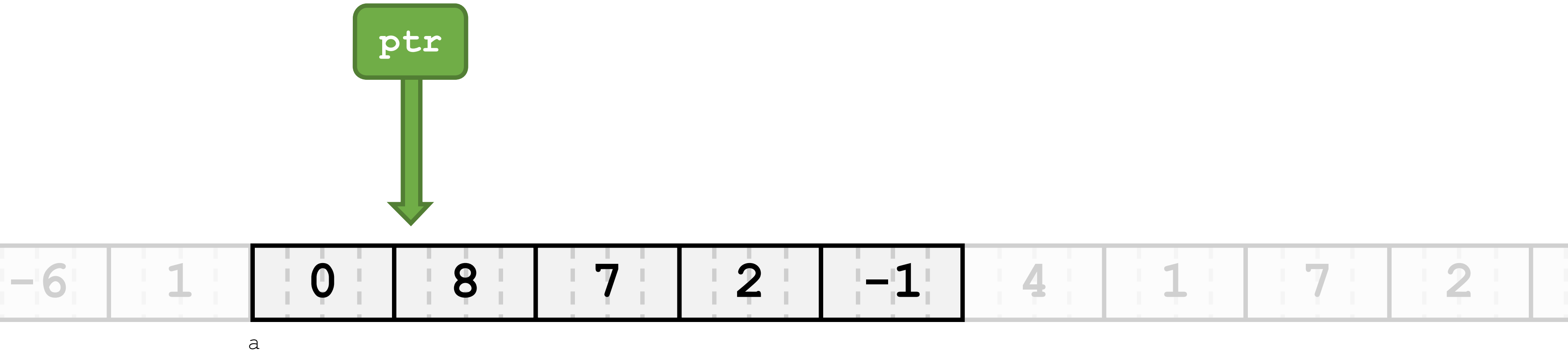
Pointer Program

```
int a[5] = {0, 8, 7, 2, -1};
int* ptr = a;                      // array-to-pointer conv
++ptr;                             // shift to the right
int my_int = *ptr;                 // read target
ptr += 2;                          // shift by 2 elements
*ptr = 18;                         // overwrite target
int* past = a+5;
std::cout << (ptr < past) << "\n"; // compare pointers
```



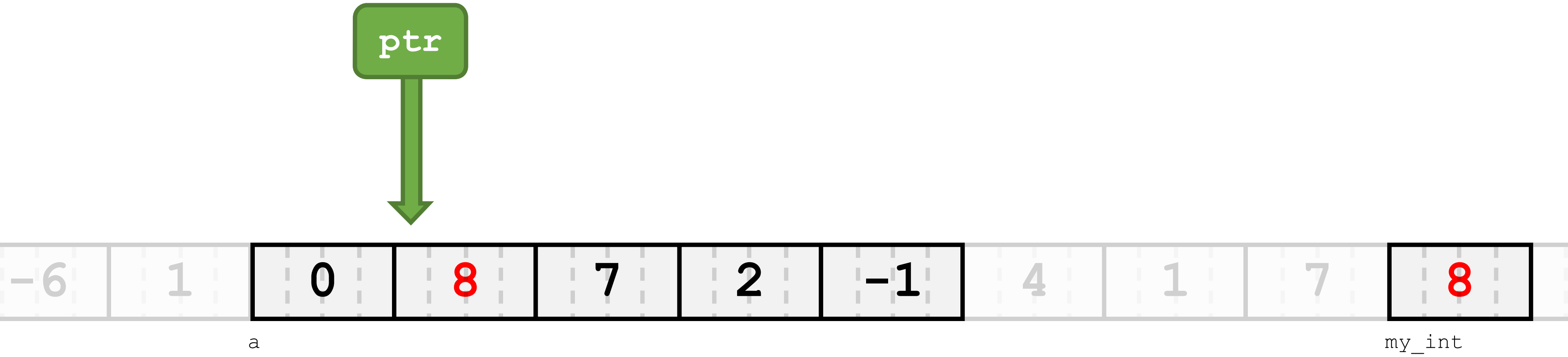
Pointer Program

```
int a[5] = {0, 8, 7, 2, -1};
int* ptr = a;                         // array-to-pointer conv
++ptr;                                // shift to the right
int my_int = *ptr;                    // read target
ptr += 2;                             // shift by 2 elements
*ptr = 18;                            // overwrite target
int* past = a+5;
std::cout << (ptr < past) << "\n"; // compare pointers
```



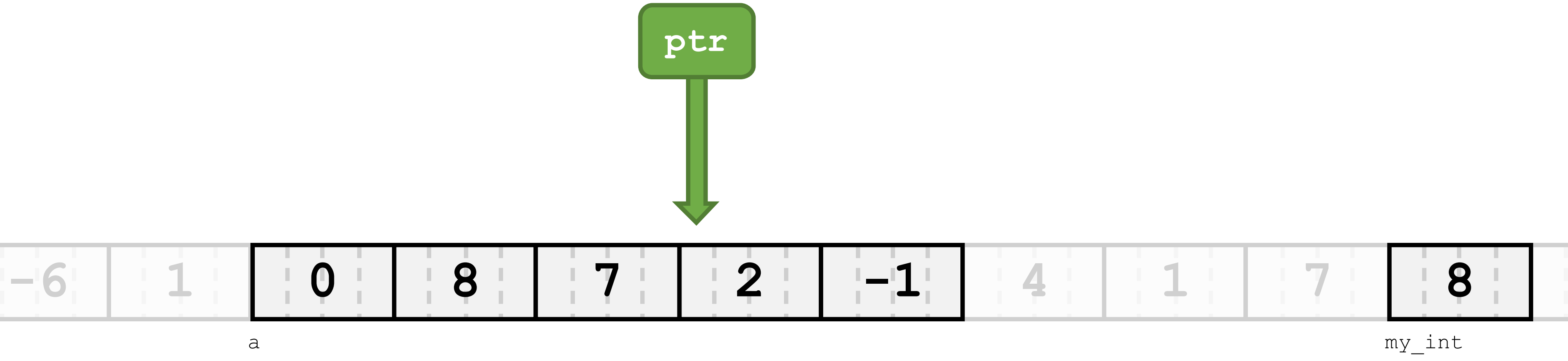
Pointer Program

```
int a[5] = {0, 8, 7, 2, -1};
int* ptr = a;                          // array-to-pointer conv
++ptr;                                 // shift to the right
int my_int = *ptr;                     // read target
ptr += 2;                              // shift by 2 elements
*ptr = 18;                             // overwrite target
int* past = a+5;
std::cout << (ptr < past) << "\n"; // compare pointers
```



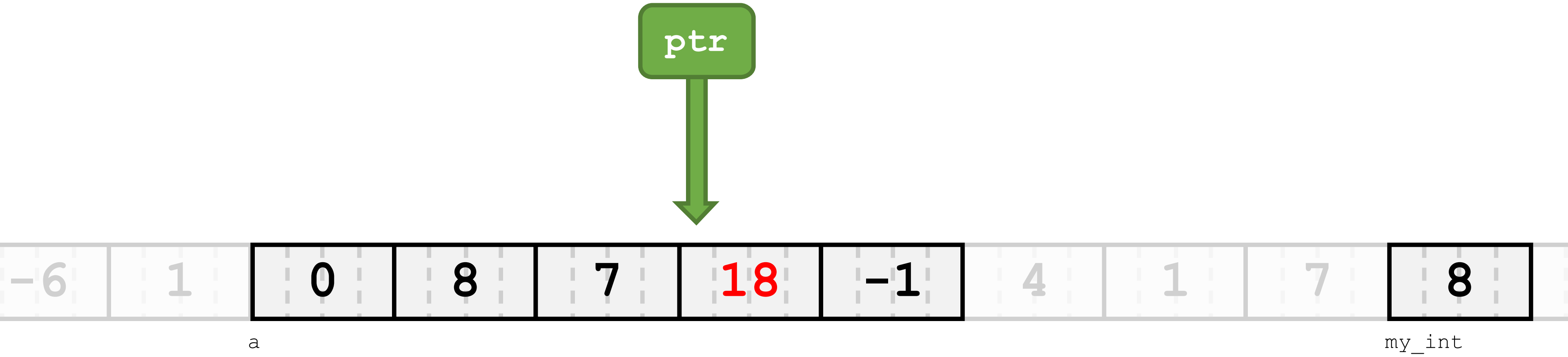
Pointer Program

```
int a[5] = {0, 8, 7, 2, -1};
int* ptr = a;                       // array-to-pointer conv
++ptr;                              // shift to the right
int my_int = *ptr;                  // read target
ptr += 2;                           // shift by 2 elements
*ptr = 18;                          // overwrite target
int* past = a+5;
std::cout << (ptr < past) << "\n"; // compare pointers
```



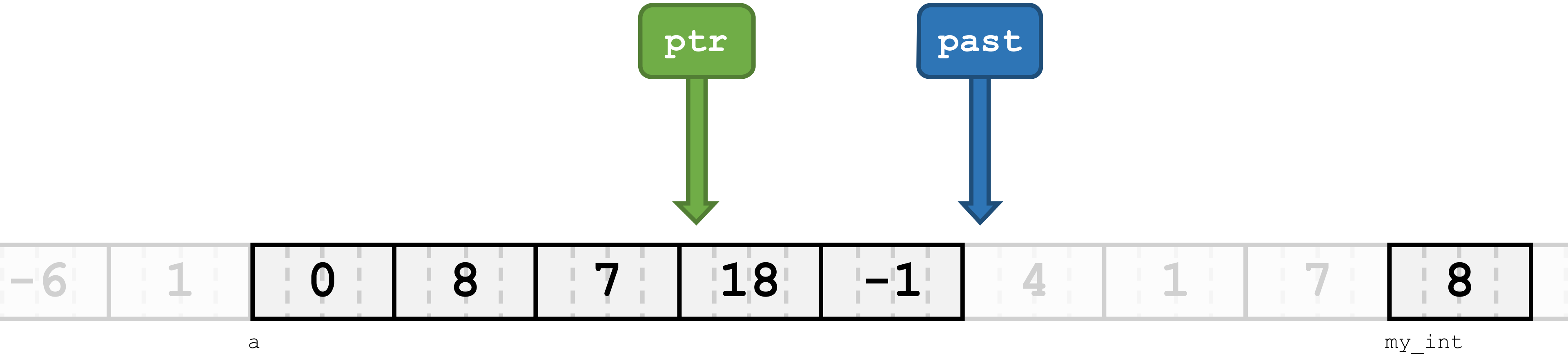
Pointer Program

```
int a[5] = {0, 8, 7, 2, -1};
int* ptr = a;                        // array-to-pointer conv
++ptr;                               // shift to the right
int my_int = *ptr;                   // read target
ptr += 2;                            // shift by 2 elements
*ptr = 18;                           // overwrite target
int* past = a+5;
std::cout << (ptr < past) << "\n"; // compare pointers
```



Pointer Program

```
int a[5] = {0, 8, 7, 2, -1};
int* ptr = a;                        // array-to-pointer conv
++ptr;                               // shift to the right
int my_int = *ptr;                   // read target
ptr += 2;                            // shift by 2 elements
*ptr = 18;                           // overwrite target
int* past = a+5;
std::cout << (ptr < past) << "\n"; // compare pointers
```

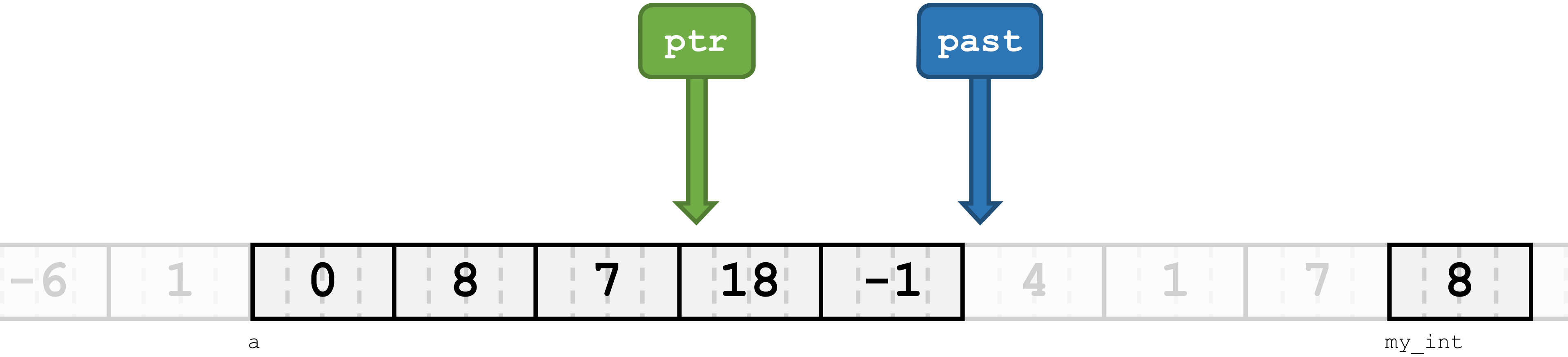


Pointer Program

```
int a[5] = {0, 8, 7, 2, -1};
int* ptr = a;                       // arr
++ptr;                              // shi
int my_int = *ptr;                  // read
ptr += 2;                           // shift b
*ptr = 18;                          // overwr
int* past = a+5;
std::cout << (ptr < past) << "\n"; // compare pointers
```

Output: `true`

Because `ptr` is "to the left" of `past`.



Program

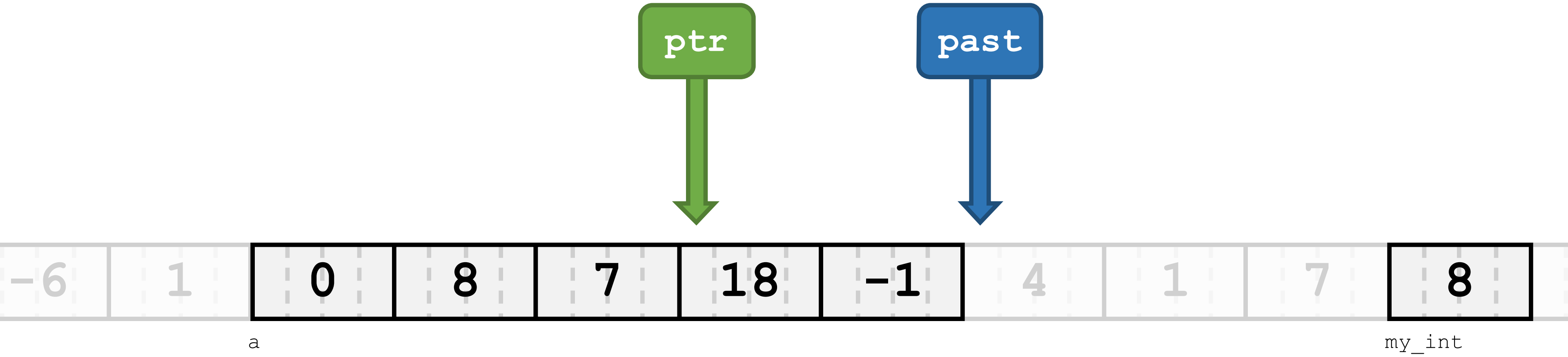
```
int a[5] = {
int* ptr = a;                        // arra
++ptr;                               // shi
int my_int = *pt                     // read
ptr += 2;                            // shift b
*ptr = 18;                           // overwri
int* past = a+5;
std::cout << (ptr < past) << "\n"; // compare pointers
```

"To the left" means:

smaller index
of element
pointed to
in array

Output: `true`

Because `ptr` is
"to the left" of `past`.



Program

"To the left" means:

smaller index
of element
pointed to
in array

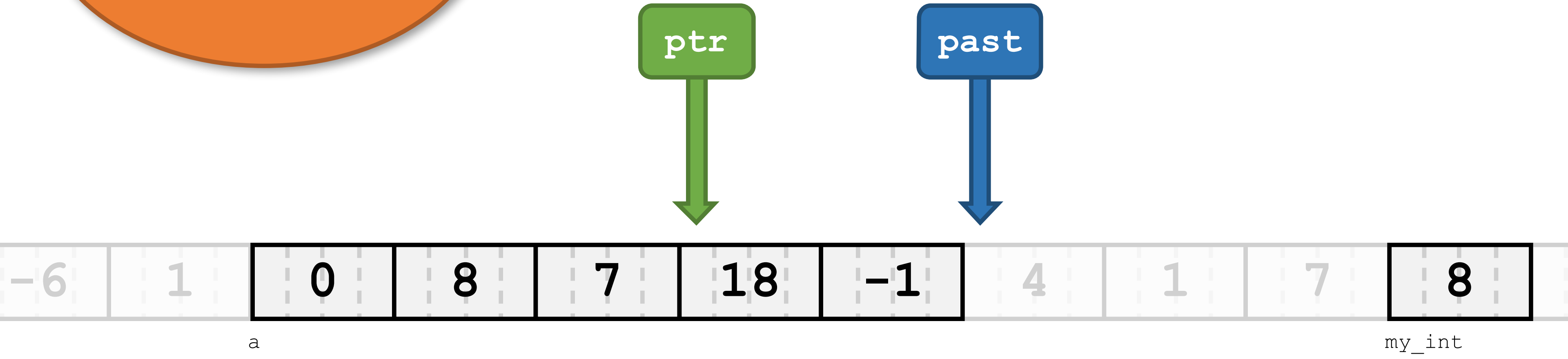
Output: `true`

Because `ptr` is
"to the left" of `past`.

```
int a[5] = {
int* ptr = a                              // arra
++ptr;                                    // shi
int my_int = *pt                          // read
ptr += 2;                                 // shift b
*ptr                                      // overwr
                  ;
              < past) << "\n"; // compare pointers
```

Here:

Index 3 < Index 5



Pointer Program

Pointer Program

```
#include <iostream>
int main () {
    int a[7] = {0, 6, 5, 3, 2, 4, 1}; // static array
    int b[7];
    int* c = b;

    // copy a into b using pointers
    for (int* p = a; p <= a+7; ++p)
        *c++ = *p;

    // cross-check with random access
    for (int i = 0; i <= 7; ++i)
        if (a[i] != c[i])
            std::cout << "Oops, copy error...\n";

    return 0;
}
```

Find and fix at least 3 problems in the following program.

Pointer Program

```
#include <iostream>
int main () {
    int a[7] = {0, 6, 5, 3, 2, 4, 1}; // static array
    int b[7];
    int* c = b;

    // copy a into b using pointers
    for (int* p = a; p <= a+7; ++p)
        *c++ = *p;

    // cross-check with random access
    for (int i = 0; i <= 7; ++i)
        if (a[i] != c[i])
            std::cout << "Oops, copy error...\n";

    return 0;
}
```

`p = a+7` is dereferenced

Solution:
Use `<` instead of `<=`

(From: Script Exercise 117)

Pointer Program

```
#include <iostream>
int main () {
    int a[7] = {0, 6, 5, 3, 2, 4, 1}; // static array
    int b[7];
    int* c = b;

    // copy a into b using pointers
    for (int* p = a; p <= a+7; ++p)
        *c++ = *p;

    // cross-check with random access
    for (int i = 0; i <= 7; ++i)
        if (a[i] != c[i])
            std::cout << "Oops, copy er

    return 0;
}
```

`p = a+7` is dereferenced

Solution:
Use `<` instead of `<=`

Same problem as above

(From: Script Exercise 117)

Pointer Program

```
#include <iostream>
int main () {
    int a[7] = {0, 6, 5, 3, 2, 4, 1}; // static array
    int b[7];
    int* c = b;

    // copy a into b using pointers
    for (int* p = a; p <= a+7; ++p)
        *c++ = *p;

    // cross-check with random access
    for (int i = 0; i <= 7; ++i)
        if (a[i] != c[i])
            std::cout << "Oops, copy er

    return 0;
}
```

`p = a+7` is dereferenced

Solution:
Use `<` instead of `<=`

`c` doesn't point to `a[0]` anymore.

Solution:
Use `b` instead of `c`

Same problem as above

Exercise

Write a program to_center.cpp which outputs the array

```
int a[] = {1, 2, 3, 4, 5, 6, 7};
```

from both ends towards the centre.

The desired output:

```
1 7 2 6 3 5 4
```

You are not allowed to use the subscript operator [ ]

Agenda

- HW #6 Feedback
- Shortest path
- Reading sequences of unknown lengths
- Strings
- Lindenmayer Systems
- Pointers on arrays
- **HW #8 Pre discussion**