NTUEE 電資工程入門設計與實作

第四週： Introduction to Search Algorithm

蘇柏青

March 15, 2021

Table of contents

問題

- 要問哪些問題來整理回顧影片的東西。
- Question 1: 我們將一個迷宮的地圖用 Graph (V, E) 來描述後，怎麼樣將 (V, E) 儲存下來你認為是對實現 BFS 演算法最好的。
  1. 直接儲存 (V, E) where $E = \{(u, v) \mid u, v \in V\}$
  2. 儲存 Adjacency list
  3. 儲存 Adjacency matrix
  4. 其他

今日工作與目標

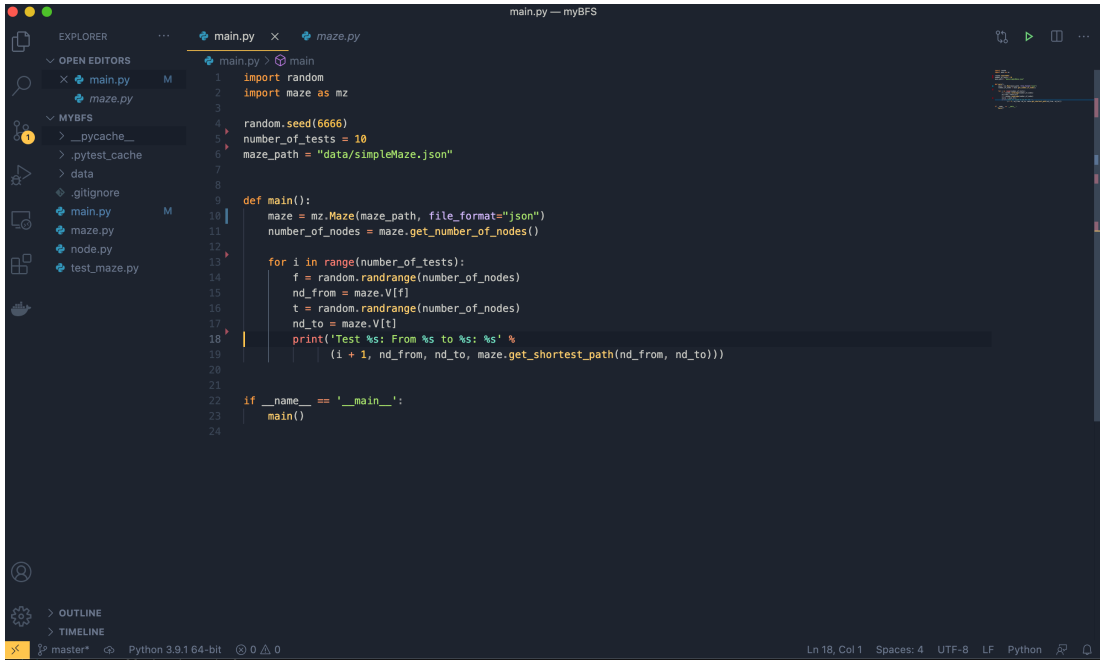
- 分組: 一共分八組，每組 3 人。有些組可能到四人。
- 將影片中的 BFS algorithm 實作出來。
  - 將使用 python 來實作。
  - 我們將提供一個簡單的程式，內含測試資料檔案以及解讀為 Adjacency list 的程式碼，但 get_shortest_path() 函式留空，待各組組員合作完成。(每組完成一份)
  - 各組應設定執行 python 的基本環境。
- (Optional) 進一步探討較 BFS 演算法更進階的問題，例如:
  - 每兩個 Nodes 之間的 edge 可能不等長時的應對方式。
  - 如果車車開進每一個 Node 都還要有一個停留時間的應對方式。
  - (Optional) 將上述問題的想法實現於現有的程式碼基礎之上。

Installing Python and the Corresponding IDE

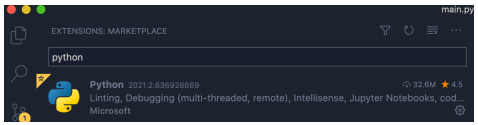
- Please install any version of Python 3. Make sure it can be executed from a terminal using a command line request.
- Specifically, type `python --version` at your terminal.
- Setup any IDE which you feel comfortable working with. Popular choices include PyCharm, Spyder, IDLE, Visual Studio Code, SublimeText, Atom, Jupyter, etc.
- The following introduction will be based on Visual Studio Code.

Visual Studio Code

- Advantages: Lightweight, tons of extensions to choose from, useful for many other languages too.

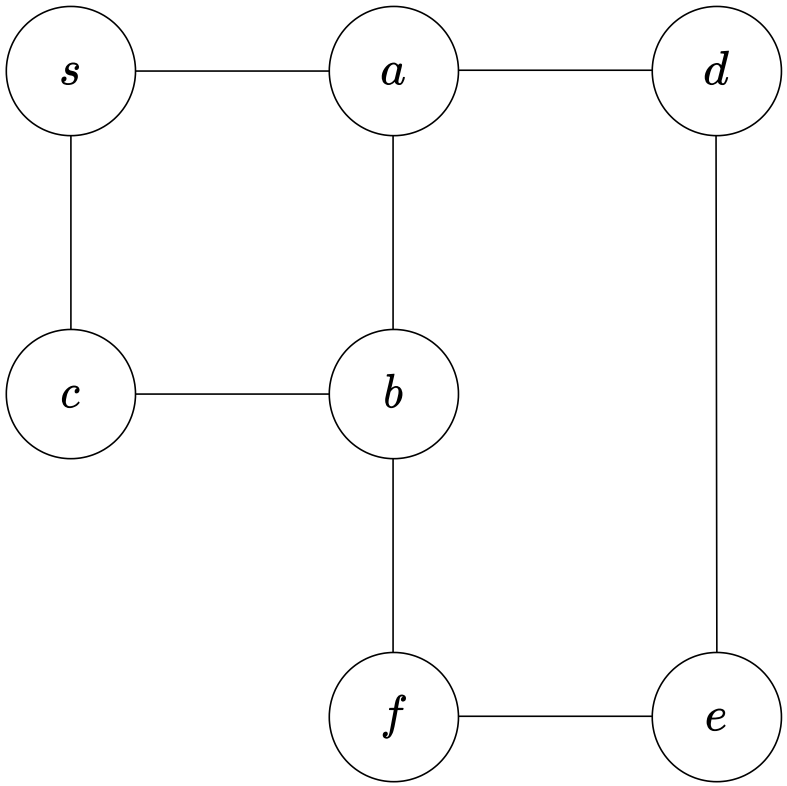


- It is recommended to install the Python extension

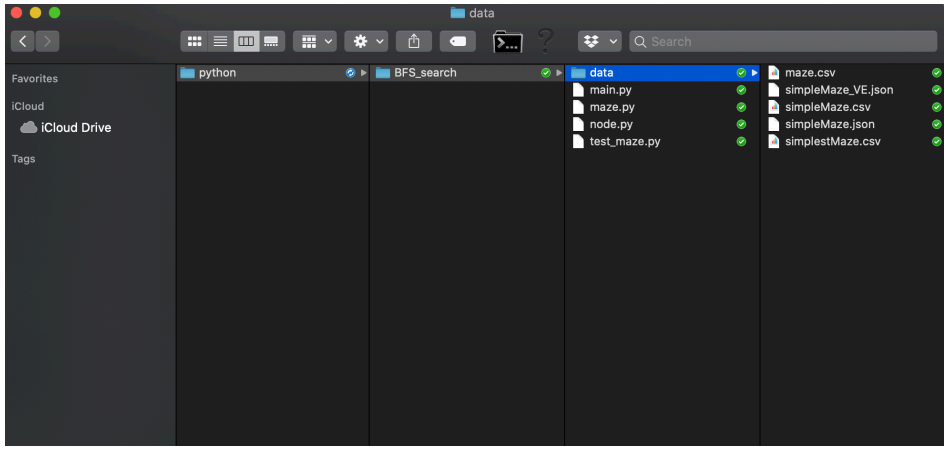


Pseudo Code for the BFS Algorithm

- Initialization
  - Q contains only the node s.
  - Only s is marked
  - $d[s] = 0$
  - $\pi[s] = \text{Null}$
- While Q is not empty
  - Let u be first node of Q
  - Remove u from Q
  - For all v in $Adj[u]$:
    - Mark v
    - Add v to Q
    - $d[v] = d[u] + 1$
    - $\pi[v] = u$

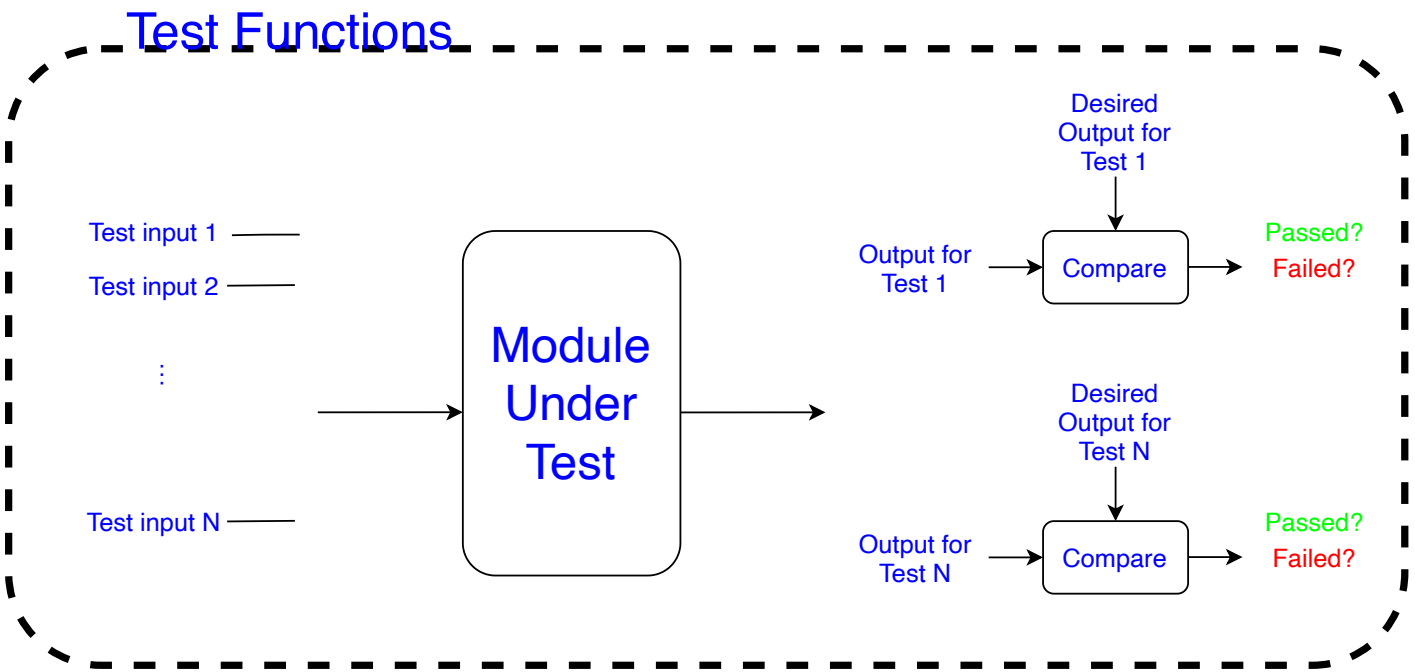


Download The Example Codes for You to Start From



- The code base contains four python files (including `main.py`, `maze.py`, `node.py`) and a number of data files (in `csv` and `JSON` formats).

The Classes of Maze and Node



Data Structure In Python and Data Stroage Formats

- Some basic python data stuctures may be useful:
  - dict(): key-value pairs
  - list(): zero-based indexed array
  - set(): unordered
- We use the following formats to storage the topologies of our mazes.
  - csv: can be edited by Excel or other tools.
  - JSON: ab open standard file and data interchange format, human-readable texts.

Basic Workflow to Load Maze Maps for Further Processing

- File Storage –> Load File and Parsing –> Python Data Structures

Our Map in the Language of Graph Theory

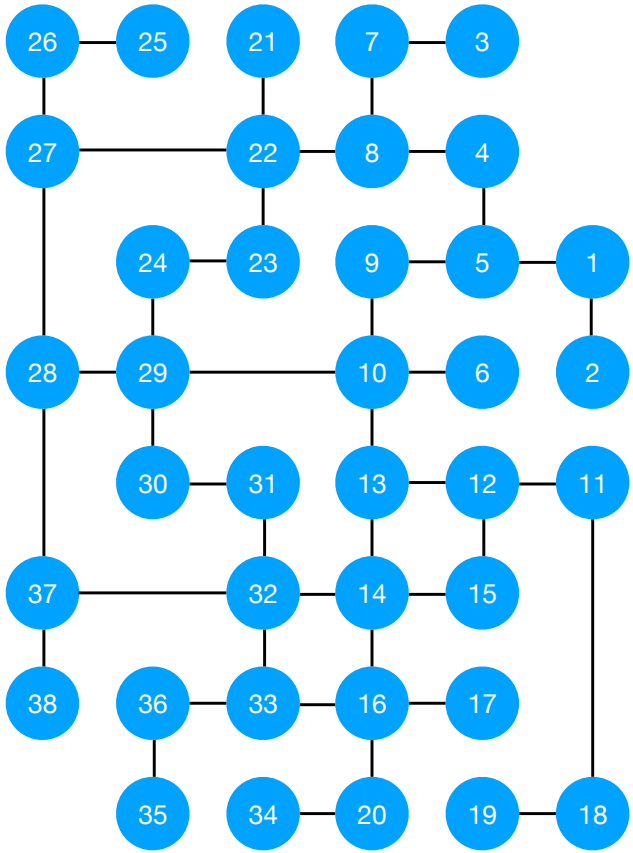
- Our map has the form like the righthand side.
- It can be modeled as a graph

$$G = (V, E)$$

or the corresponding adjacency list:

$$\text{Adj} : V \to 2^V$$

defined so that
$u \in \text{Adj}[v], \forall u, v \in V$ if and only if $(u, v) \in E$.



CSV File

The csv data file has the following format

```
index,North,South,East,West
1,,5,2,
2,1,,,
3,,7,,
4,,8,5,
5,4,9,,1
6,,10,,
7,,,8,3
8,7,22,,4
9,,,10,5
10,9,29,13,6
11,,12,18,
...
```

	A	B	C	D	E
1	index	North	South	East	West
2	1		2		5
3	2	1			
4	3				7
5	4		5		8
6	5	4		1	9
7	6				10
8	7		8	3	
9	8	7		4	22
10	9		10	5	
11	10	9	13	6	29
12	11		18		12
13	12		15	11	13
14	13	10	14	12	

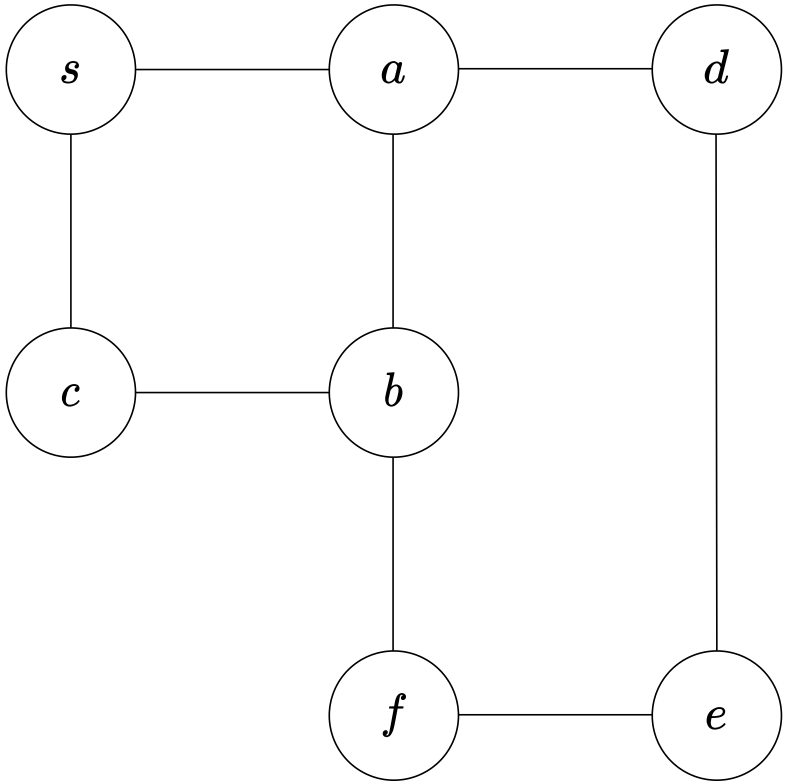
JSON Format – For the Adjacent List Format

simpleMaze.json — BFS_search

{} simpleMaze.json ×

data > {} simpleMaze.json > ...

```
{
    "maze_format": "adjacency list",
    "maze_data":
    {
        "s": ["a", "c"],
        "a": ["s", "b", "d"],
        "b": ["a", "c", "f"],
        "c": ["s", "b"],
        "d": ["a", "e"],
        "e": ["d", "f"],
        "f": ["b", "e"]
    }
}
```



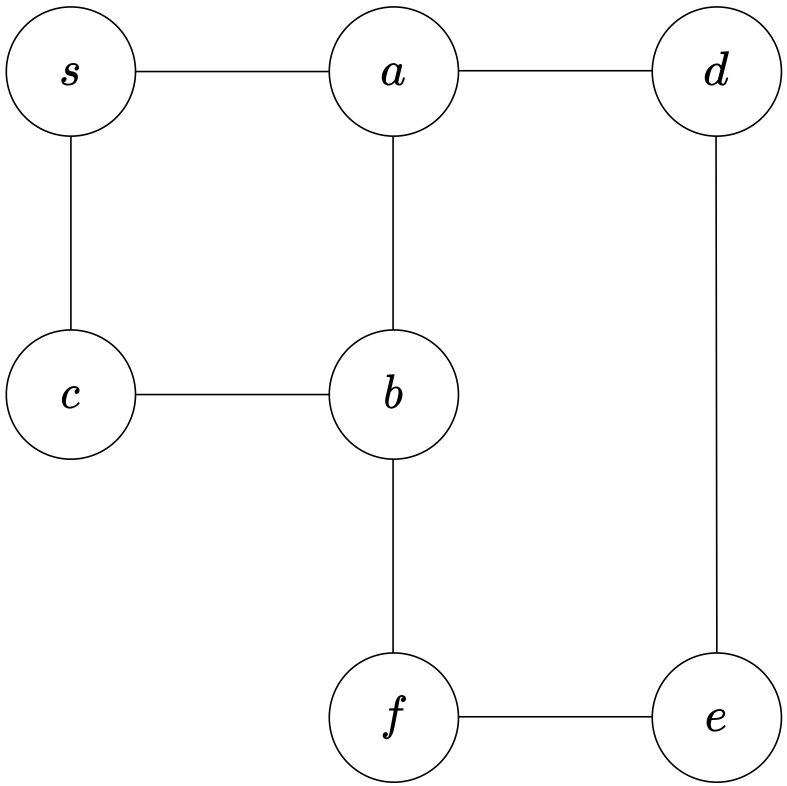
JSON Format – For the Vertex-Edge Format

simpleMaze_VE.json — BFS_search

{} simpleMaze_VE.json ×

data > {} simpleMaze_VE.json > ...

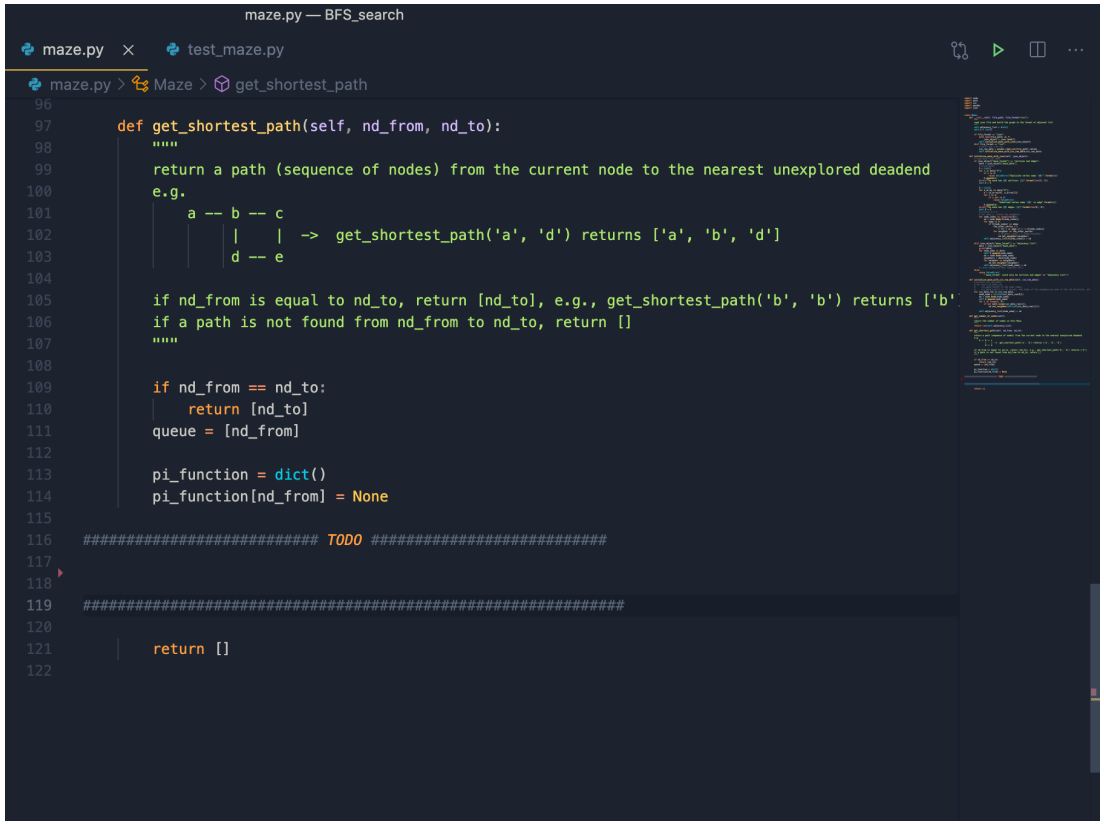
```
{
    "maze_format": "vertices and edges",
    "maze_data":
    {
        "V": ["s", "a", "b", "c", "d", "e", "f"],
        "E": [
            ["s", "a"],
            ["s", "c"],
            ["a", "b"],
            ["b", "c"],
            ["a", "d"],
            ["d", "e"],
            ["b", "f"],
            ["e", "f"]
        ]
    }
}
```



Main.py

- Random starting and exiting nodes are generated to test the `get_shortest_path()` function of `Maze` class.

Write Your Own Codes for the BFS Algorithm



Ideal Results

- Current Result

```
BorchingdeMacBook-Pro:BFS_search borching$ git checkout student_version
warning: refname 'student_version' is ambiguous.
Switched to branch 'student_version'
BorchingdeMacBook-Pro:BFS_search borching$ python3 ./main.py
{'s': ['a', 'c'], 'a': ['s', 'b', 'd'], 'b': ['a', 'c', 'f'], 'c': ['s', 'b'], 'd': ['a', 'e'], 'e': ['d', 'f'], 'f': ['b', 'e']}
Test 1: From a to a: ['a']
Test 2: From a to c: []
Test 3: From c to c: ['c']
Test 4: From s to a: []
Test 5: From e to f: []
Test 6: From a to a: ['a']
Test 7: From d to f: []
Test 8: From s to s: ['s']
Test 9: From c to a: []
Test 10: From f to b: []
BorchingdeMacBook-Pro:BFS_search borching$
```

- Ideal Result

```
BorchingdeMacBook-Pro:BFS_search borching$ python3 ./main.py
{'s': ['a', 'c'], 'a': ['s', 'b', 'd'], 'b': ['a', 'c', 'f'], 'c': ['s', 'b'], 'd': ['a', 'e'], 'e': ['d', 'f'], 'f': ['b', 'e']}
Test 1: From a to a: ['a']
Test 2: From a to c: ['a', 's', 'c']
Test 3: From c to c: ['c']
Test 4: From s to a: ['s', 'a']
Test 5: From e to f: ['e', 'f']
Test 6: From a to a: ['a']
Test 7: From d to f: ['d', 'e', 'f']
Test 8: From s to s: ['s']
Test 9: From c to a: ['c', 's', 'a']
Test 10: From f to b: ['f', 'b']
BorchingdeMacBook-Pro:BFS_search borching$
```

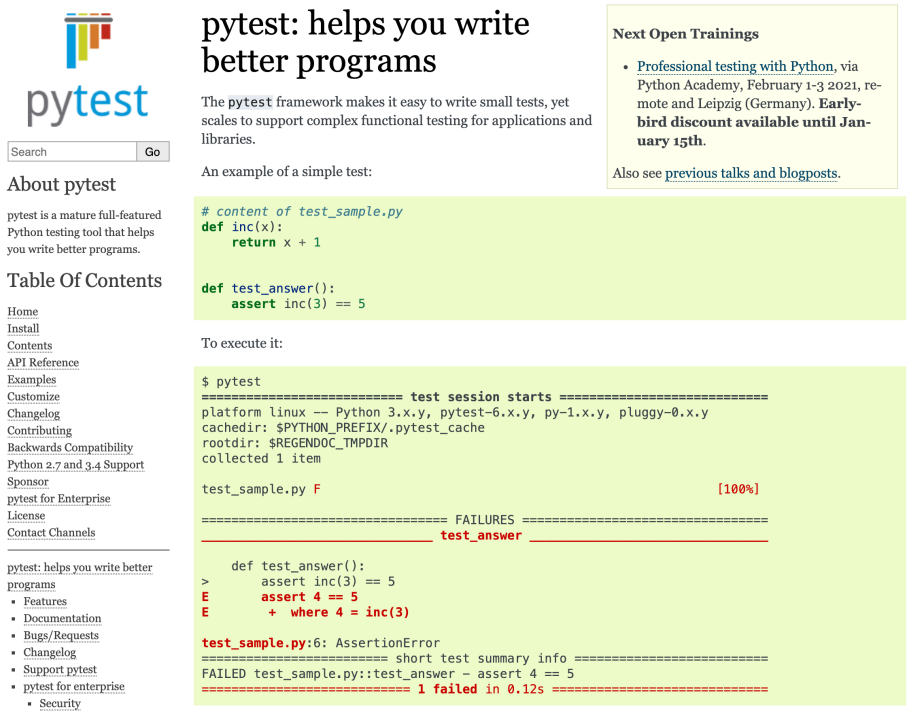
Change the Input Data To Try The Code You Modified

- Try to modfiy: number_of_tests, maze_path (and file_format)

- We have 5 sets of data to try. Is there a better and more efficient way to avoid the hassle?

Testing the Module Using PyTest

- We can use PyTest to test our codes to ensure the code integrity.



pytest

Search Go

About pytest

pytest is a mature full-featured Python testing tool that helps you write better programs.

Table Of Contents

Home
Install
Contents
API Reference
Examples
Customize
Changelog
Contributing
Backwards Compatibility
Python 2.7 and 3.4 Support
Sponsor
pytest for Enterprise
License
Contact Channels

pytest: helps you write better programs
- Features
- Documentation
- Bugs/Requests
- Changelog
- Support pytest
- pytest for enterprise
  - Security

pytest: helps you write better programs

The `pytest` framework makes it easy to write small tests, yet scales to support complex functional testing for applications and libraries.

An example of a simple test:

Next Open Trainings

- Professional testing with Python, via Python Academy, February 1-3 2021, remote and Leipzig (Germany). **Early-bird discount available until January 15th**.

Also see previous talks and blogposts.

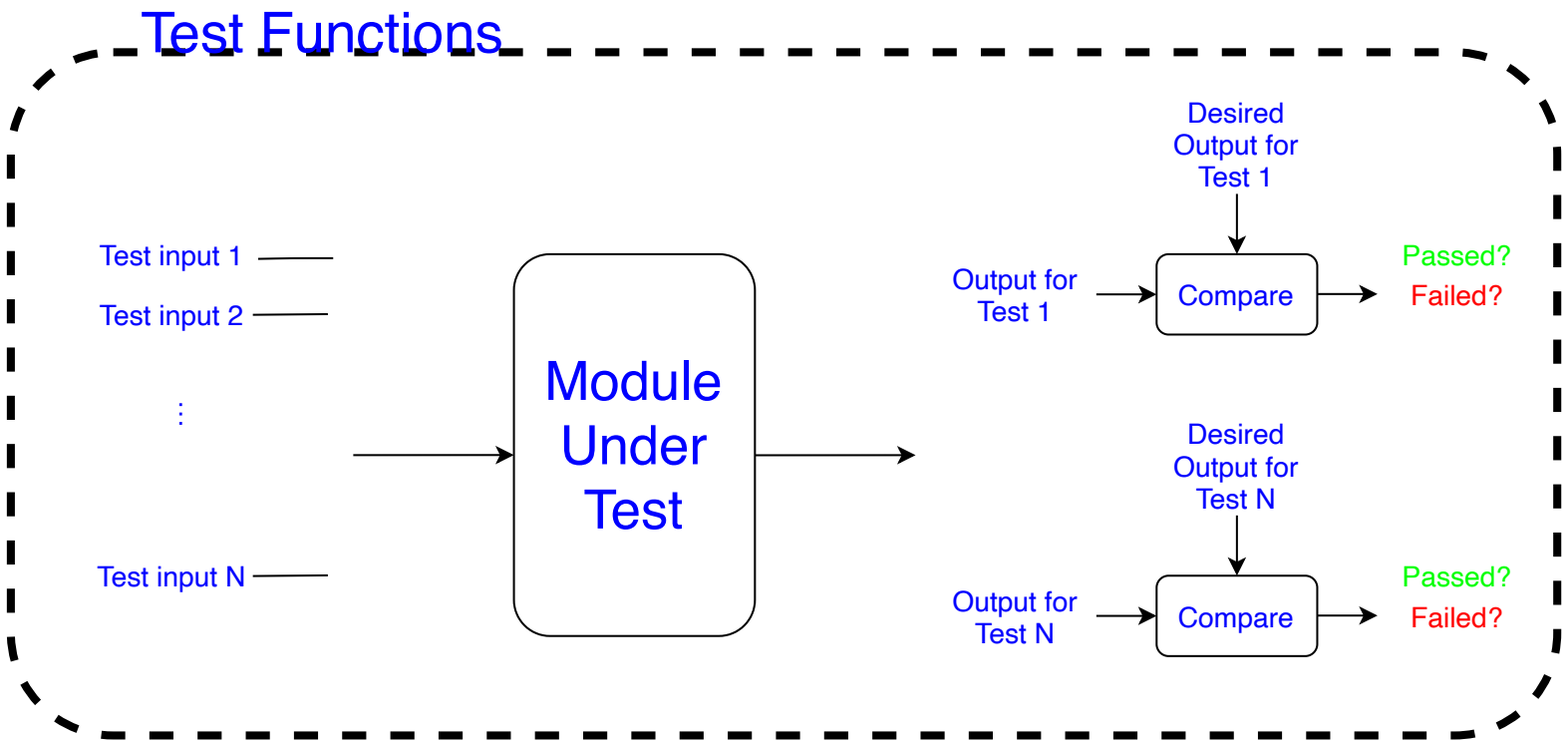
```
# content of test_sample.py
def inc(x):
    return x + 1


def test_answer():
    assert inc(3) == 5
```

To execute it:

```
$ pytest
========================== test session starts ==========================
platform linux -- Python 3.x.y, pytest-6.x.y, py-1.x.y, pluggy-0.x.y
cachedir: $PYTHON_PREFIX/.pytest_cache
rootdir: $REGENDOC_TMPDIR
collected 1 item

test_sample.py F                                                  [100%]

================================ FAILURES ================================
_____________________________ test_answer _____________________________

    def test_answer():
>       assert inc(3) == 5
E       assert 4 == 5
E        +  where 4 = inc(3)

test_sample.py:6: AssertionError
====================== short test summary info ======================
FAILED test_sample.py::test_answer - assert 4 == 5
========================== 1 failed in 0.12s ==========================
```

- Please install pytest following the instructions from the official documentations.

Testing the Modules – Illustration



PyTest Execution Example

BFS_search — -bash — 162×56

```
E         Use -v to get the full diff

test_maze.py:43: AssertionError
_____________________________________________________ test_simple_maze_vertices_and_edges _____________________________________________________

    def test_simple_maze_vertices_and_edges():
        maze_path = "data/simpleMaze_VE.json"
        maze = mz.Maze(maze_path, "json")

        assert maze.get_number_of_nodes() == 7

        shortest_path = maze.get_shortest_path('a', 'a')
        assert shortest_path == ['a']

        shortest_path = maze.get_shortest_path('a', 'c')
>       assert shortest_path == ['a', 's', 'c'] or shortest_path == ['a', 'b', 'c']
E       AssertionError: assert ([] == ['a', 's', 'c']
E         Right contains 3 more items, first extra item: 'a'
E         Use -v to get the full diff or [] == ['a', 'b', 'c']
E         Right contains 3 more items, first extra item: 'a'
E         Use -v to get the full diff)

test_maze.py:65: AssertionError
------------------------------------------------------------ Captured stdout call ------------------------------------------------------------
The maze has 7 vertices: ['s', 'a', 'b', 'c', 'd', 'e', 'f']
The maze has 8 edges: [{'s', 'a'}, {'s', 'c'}, {'b', 'a'}, {'c', 'b'}, {'d', 'a'}, {'e', 'd'}, {'f', 'b'}, {'f', 'e'}]
_____________________________________________________ test_simple_maze_json_adjacency_list _____________________________________________________

    def test_simple_maze_json_adjacency_list():
        maze_path = "data/simpleMaze.json"
        maze = mz.Maze(maze_path, "json")

        assert maze.get_number_of_nodes() == 7

        shortest_path = maze.get_shortest_path('a', 'a')
        assert shortest_path == ['a']

        shortest_path = maze.get_shortest_path('a', 'c')
>       assert shortest_path == ['a', 's', 'c'] or shortest_path == ['a', 'b', 'c']
E       AssertionError: assert ([] == ['a', 's', 'c']
E         Right contains 3 more items, first extra item: 'a'
E         Use -v to get the full diff or [] == ['a', 'b', 'c']
E         Right contains 3 more items, first extra item: 'a'
E         Use -v to get the full diff)

test_maze.py:84: AssertionError
------------------------------------------------------------ Captured stdout call ------------------------------------------------------------
{'s': ['a', 'c'], 'a': ['s', 'b', 'd'], 'b': ['a', 'c', 'f'], 'c': ['s', 'b'], 'd': ['a', 'e'], 'e': ['d', 'f'], 'f': ['b', 'e']}
============================================================ short test summary info =============================================================
FAILED test_maze.py::test_simplest_maze - AssertionError: assert [] == ['2', '4']
FAILED test_maze.py::test_simple_maze - AssertionError: assert [] == ['2', '3', '4']
FAILED test_maze.py::test_maze - AssertionError: assert [] == ['34', '20', ...3', '12', ...]
FAILED test_maze.py::test_simple_maze_vertices_and_edges - AssertionError: assert ([] == ['a', 's', 'c']
FAILED test_maze.py::test_simple_maze_json_adjacency_list - AssertionError: assert ([] == ['a', 's', 'c']
============================================================ 5 failed in 0.46s ============================================================
BorchingdeMacBook-Pro:BFS_search borching$ pytest
```

PyTest Execution Example – Ideal Case

```
---------------------------------------------------------------------- Captured stdout call ----------------------------------------------------------------------
{'s': ['a', 'c'], 'a': ['s', 'b', 'd'], 'b': ['a', 'c', 'f'], 'c': ['s', 'b'], 'd': ['a', 'e'], 'e': ['d', 'f'], 'f': ['b', 'e']}
=========================================================================== short test summary info ============================================================================
FAILED test_maze.py::test_simplest_maze - AssertionError: assert [] == ['2', '4']
FAILED test_maze.py::test_simple_maze - AssertionError: assert [] == ['2', '3', '4']
FAILED test_maze.py::test_maze - AssertionError: assert [] == ['34', '20', ...3', '12', ...]
FAILED test_maze.py::test_simple_maze_vertices_and_edges - AssertionError: assert ([] == ['a', 's', 'c']
FAILED test_maze.py::test_simple_maze_json_adjacency_list - AssertionError: assert ([] == ['a', 's', 'c']
============================================================================ 5 failed in 0.59s =============================================================================
BorchingdeMacBook-Pro:BFS_search borching$ git checkout TA_version
Switched to branch 'TA_version'
BorchingdeMacBook-Pro:BFS_search borching$ pytest
============================================================================= test session starts ==============================================================================
platform darwin -- Python 3.9.1, pytest-6.2.2, py-1.10.0, pluggy-0.13.1
rootdir: /Users/borching/Dropbox/NTU1/2_Teaching/109_2_CornerStone/python/python/BFS_search
collected 5 items

test_maze.py .....                                                                                                                                          [100%]

============================================================================== 5 passed in 0.34s ===============================================================================
BorchingdeMacBook-Pro:BFS_search borching$
```

More About PyTest

- PyTest automatically recognizes file names and functions prefixed with `'test_'` or suffixed with `'_test'`.
- As the project moving on, new test cases can be added. And any hidden errors can be spotted much earlier and catch your attention to fix them before they cause big troubles.

Question 1

What if the distance between each pair of nodes is not a constant?

Question 2

What if there is an additional nonzero delay time when transiting in a particular node?

Summary

- We learned to implementation BFS algorithm in python.
- Various tools and techniques on file format, data structures, code testing are surveyed and introduced.
- Extended questions beyond the BFS algorithm are discussed.