Tutorial 9

Swinburne University of Technology

Software Testing and Reliability (SWE30009)

Semester 2, 2023

Lecturer: Prof T. Y. Chen

Tutor: Dr Hung Q Luu

Task 3

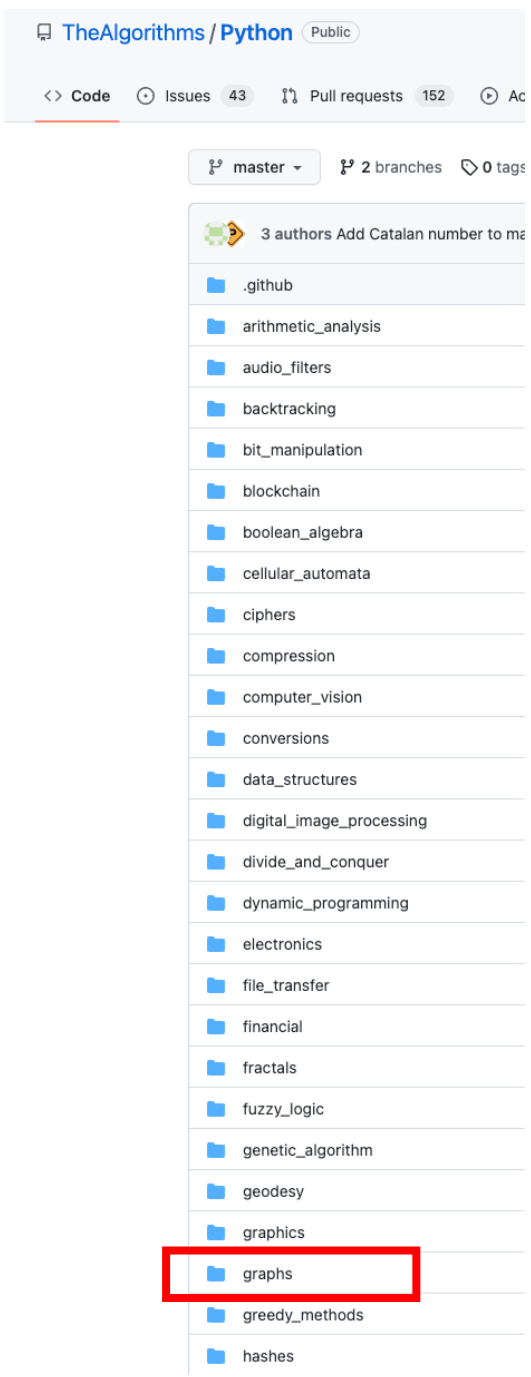
Final Assignment

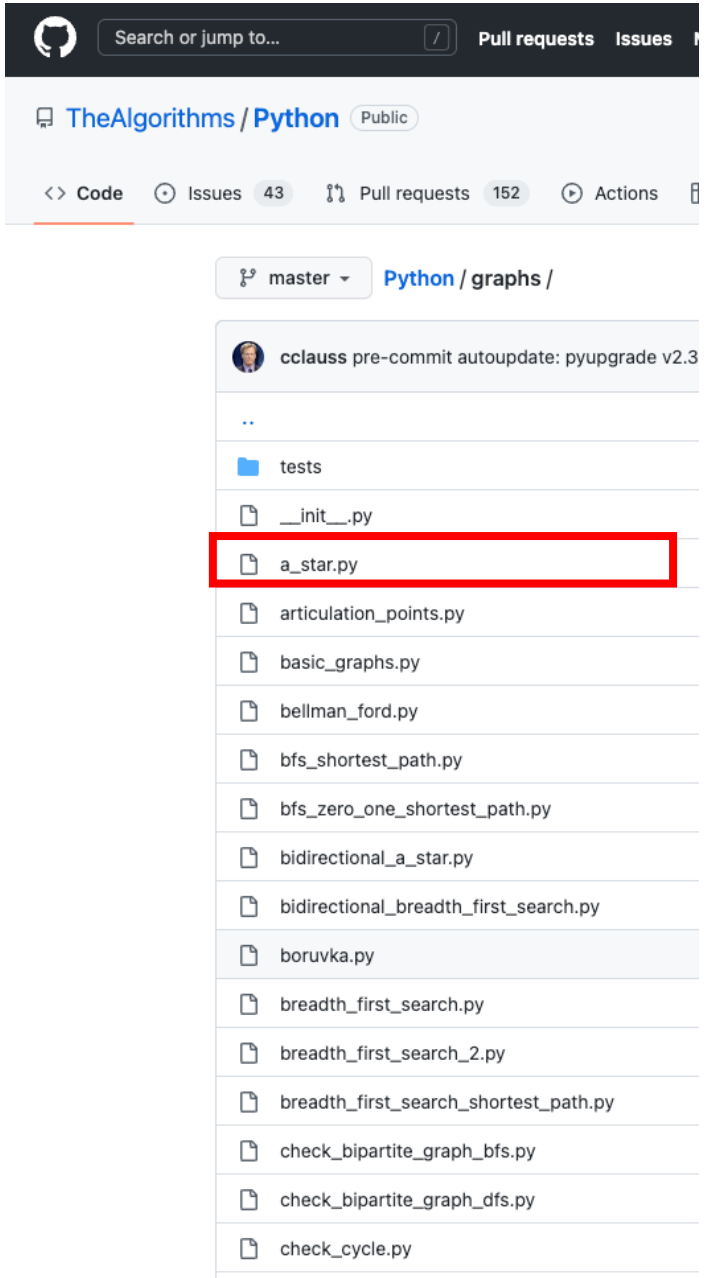
Project Report - Task 3

1) Select program P for testing
2) Define Metamorphic Relations (MRs)
3) Generate & evaluate mutants
4) Prepare source test cases & execute mutants using them
5) Generate follow-up test cases (with MRs) & execute mutants using them
6) Verify MRs against relevant Metamorphic Groups (MGs) & their outputs
7) Report results

Step 1: Select program P

- https://github.com/TheAlgorithms
  - Python/graphs
    - a_star.py



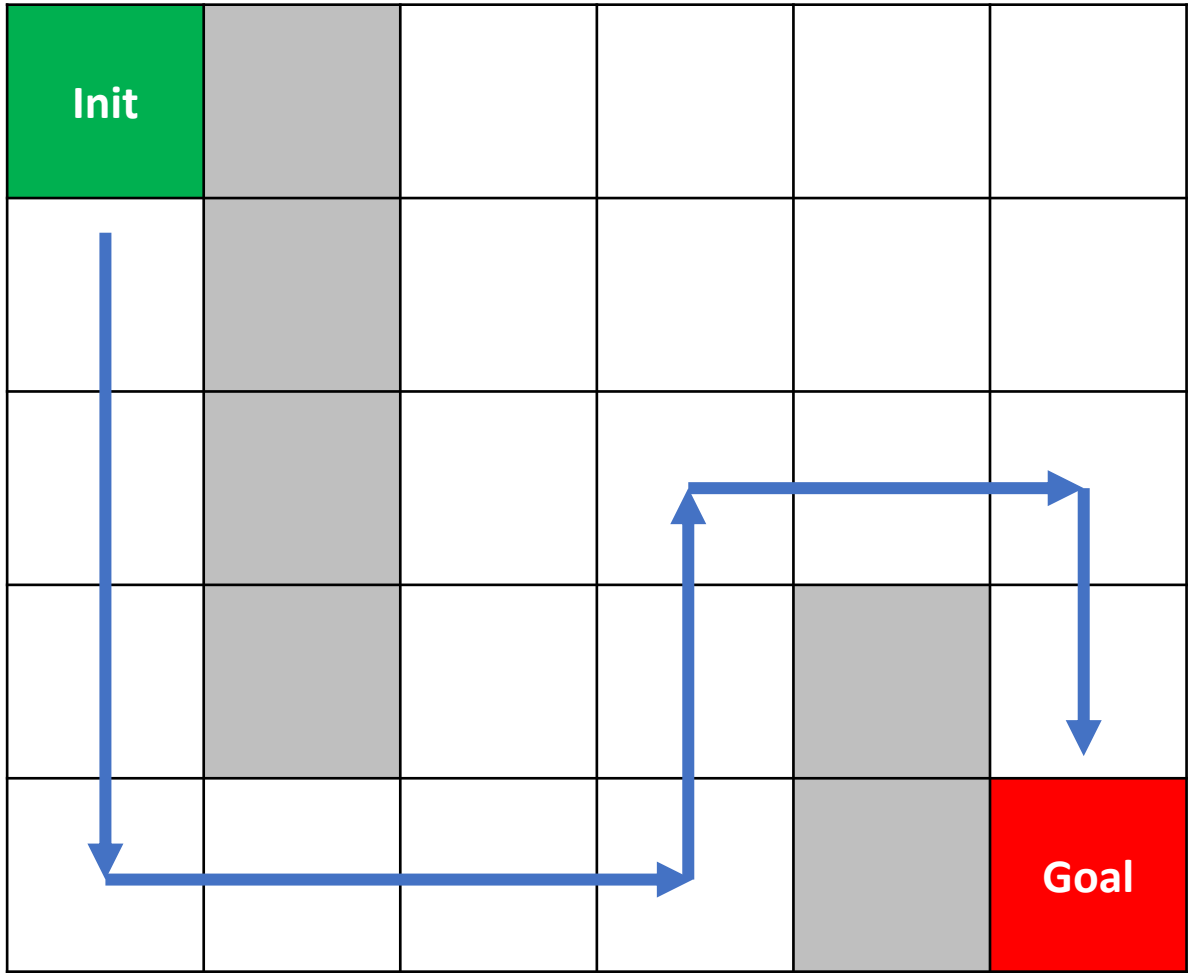


Path-finding algorithm

- Input
  - Graph
  - Init
  - Goal
- Output
  - Paths

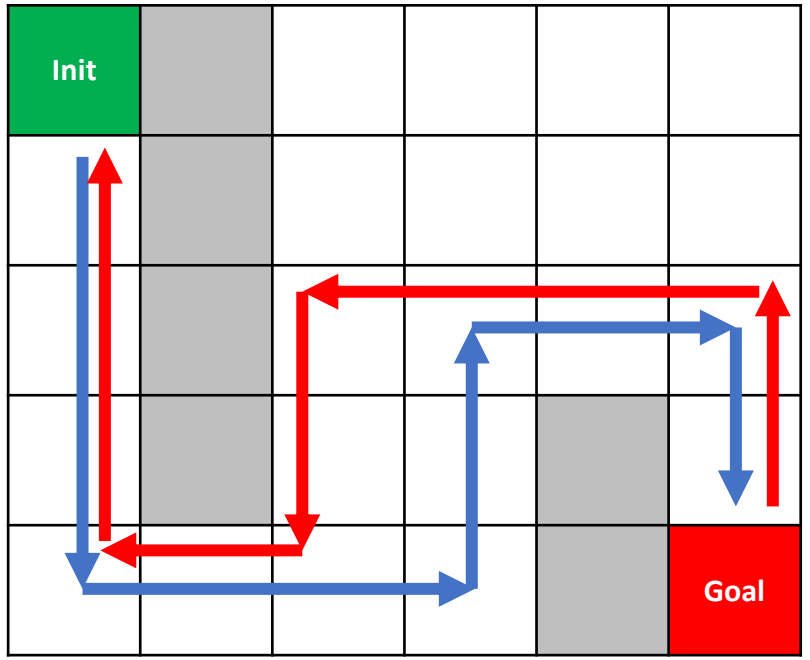
```
INIT: [0, 0]
GOAL: [4, 5]
FILENAME:
[0, 0, 0, 0, 0, 0]
[2, 0, 0, 0, 0, 0]
[2, 0, 0, 0, 3, 3]
[2, 0, 0, 0, 0, 2]
[2, 3, 3, 3, 0, 2]
[0, 0]
[1, 0]
[2, 0]
[3, 0]
[4, 0]
[4, 1]
[4, 2]
[4, 3]
[3, 3]
[2, 3]
[2, 4]
[2, 5]
[3, 5]
[4, 5]
```

GraphA: matrix 6 x 5 cells (with “obstacles”)



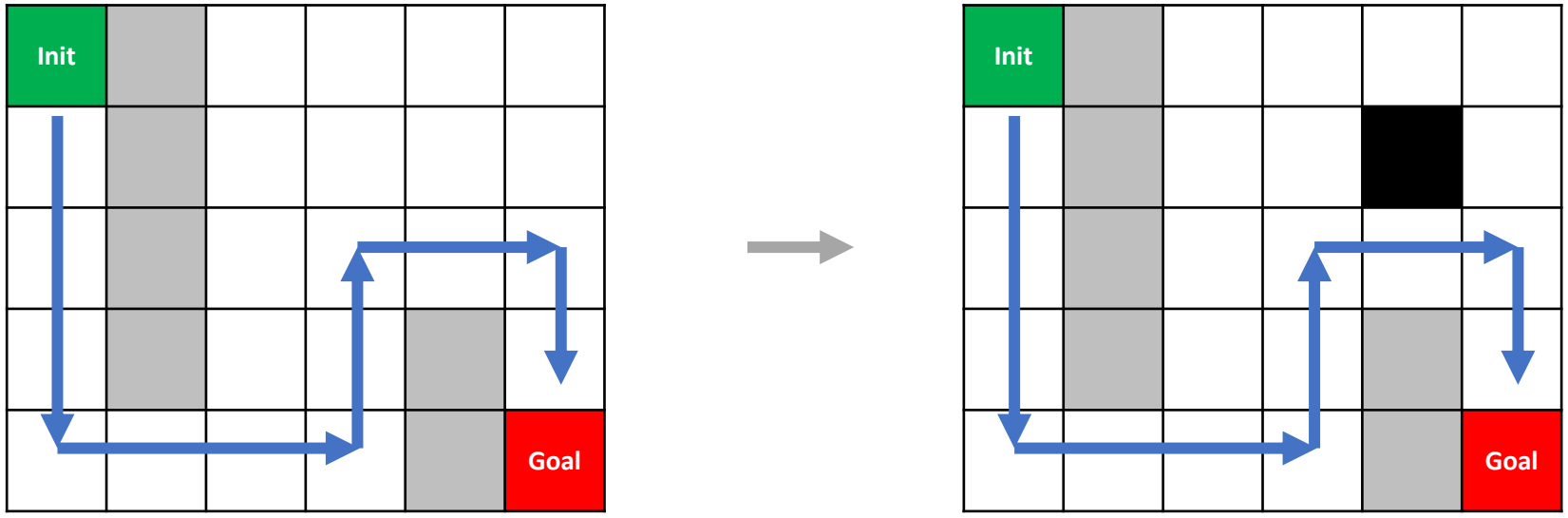
Step 2: Define Metamorphic Relations

- MR1: Reversing goal and init nodes
  - Description: since the graph is bi-directional, reversing the init and the goal will give you the same number of nodes.
  - SI = [GraphA, init: [0,0], goal: [4,5]]
  - FI = [GraphA, init: [4,5], goal: [0,0]]
  - SO = Paths { [0,0]->[1,0] -> … -> [3,5] -> [4,5] }
  - FO = Paths { [4,5]->[3,5] -> … [0,0] }
  - MR satisfaction condition: n(FO) == n(SO) with n is number of nodes (or "cells")



Step 2: Define Metamorphic Relations

- MR2: Adding obstacle(s)
  - Description: Adding random obstacle(s) may make the paths longer or unchanged…
  - MR satisfaction condition: n(FO) >= n(SO)

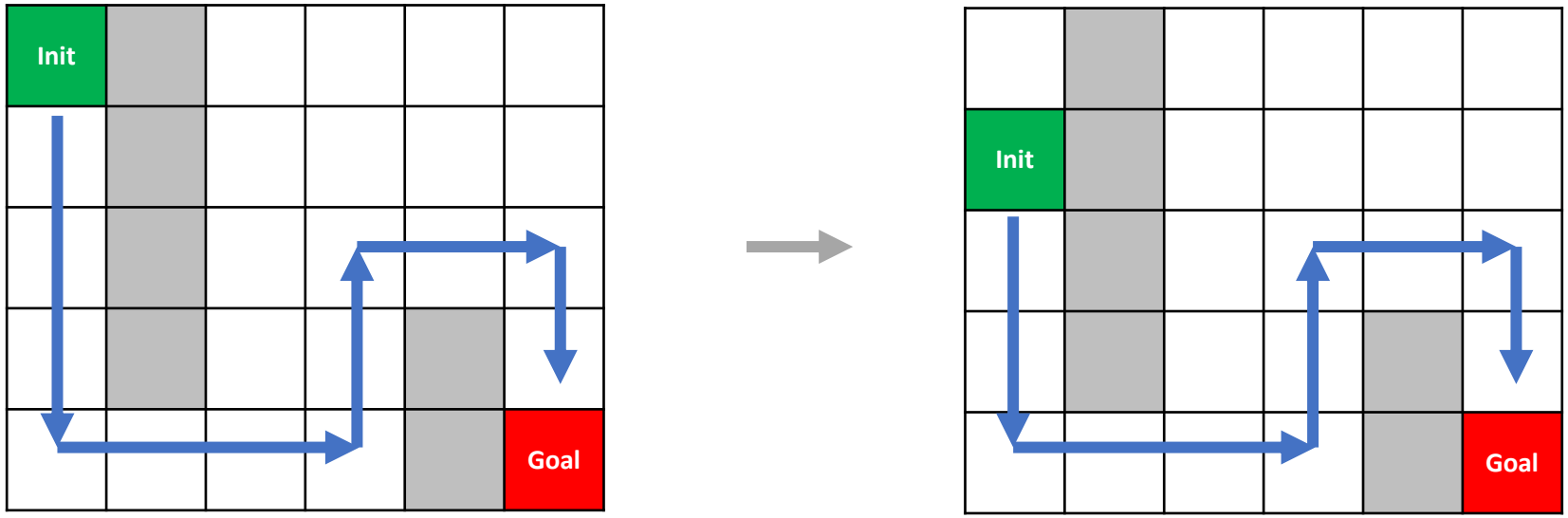


Source test case

Follow-up test case

Step 2: Define Metamorphic Relations

- MR3: Moving closer
  - Description: Moving the init or goal nodes closer may have a shorter path…
  - MR satisfaction condition: n(FO) <= n(SO)

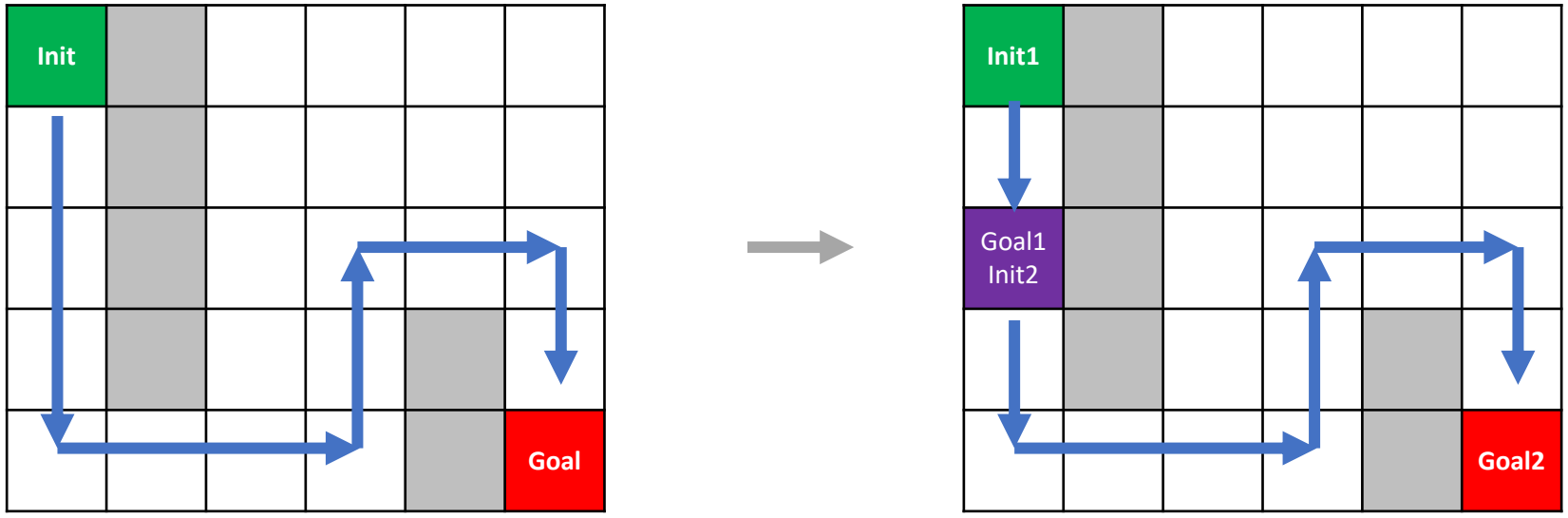


Source test case

Follow-up test case

Step 2: Define Metamorphic Relations

- MR4: Dividing paths
  - Description: Using a point from a path as new init & goal doesn't change the path
  - MR satisfaction condition: n(FO1) + n(FO2) == n(SO)

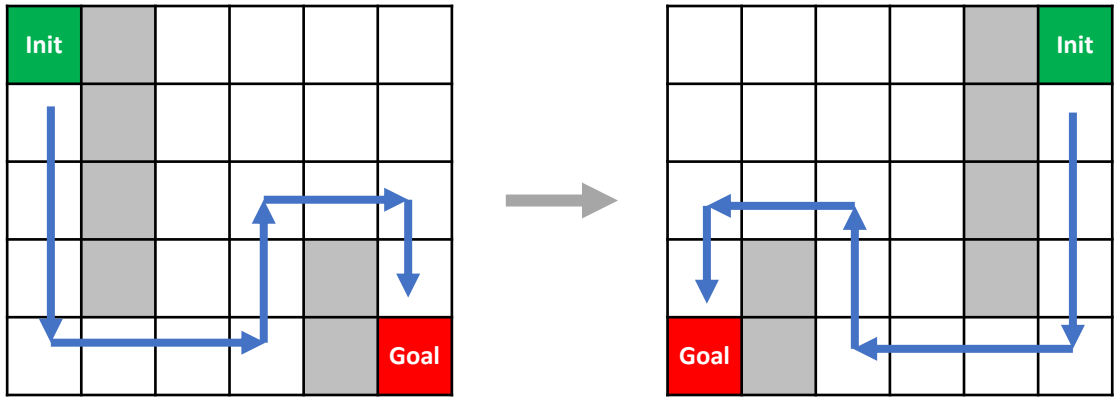


Source test case

Follow-up test cases

Step 2: Define Metamorphic Relations

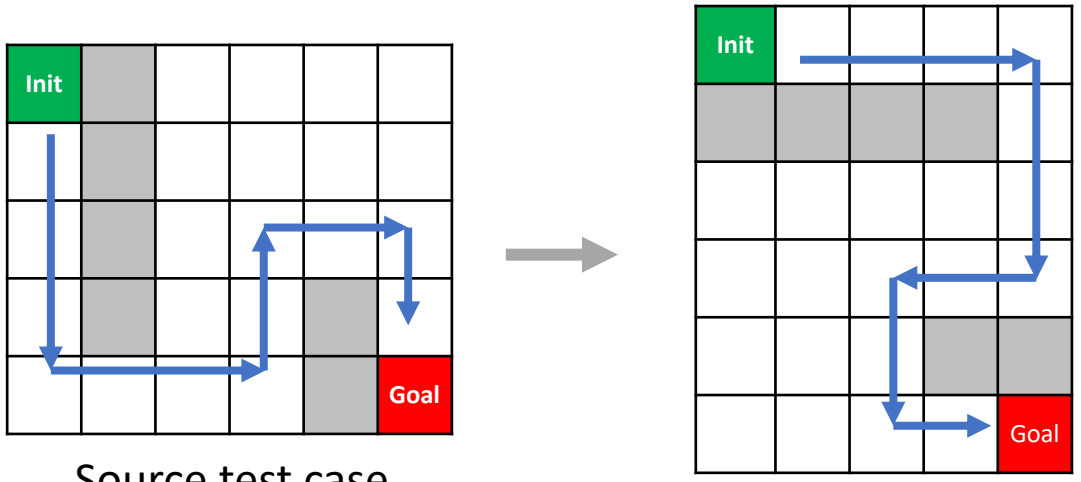
- MR5: Flipping graph



Source test case

Follow-up test case

- MR6: Rotating graph



Source test case

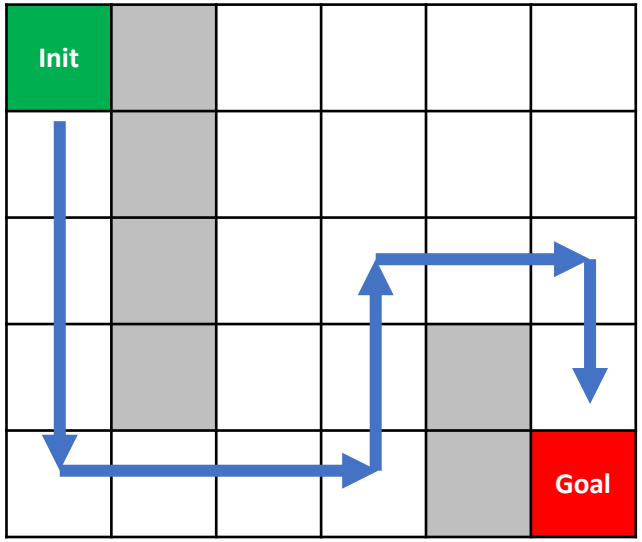
Follow-up test case

Step 3: Generate and evaluate mutants

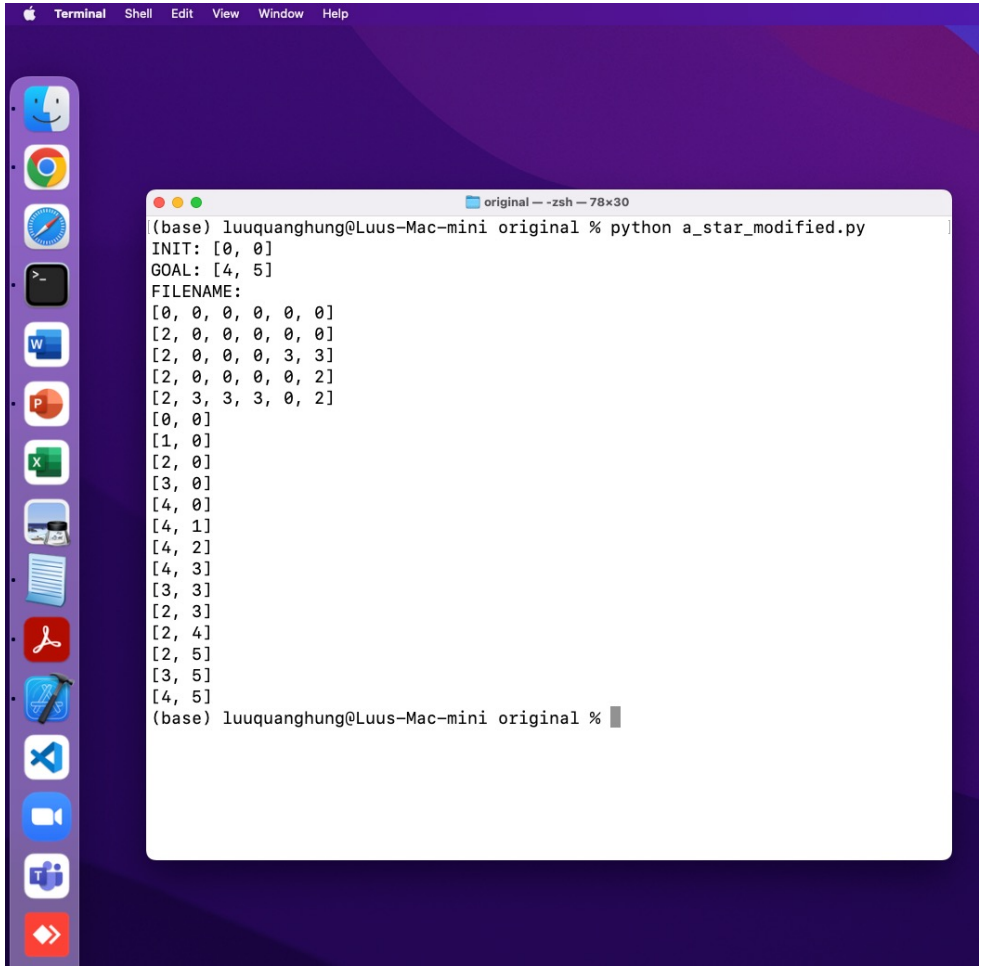
- Generate mutants
  - Do it manually
  - Do it automatically using mutant generation tool
- Evaluate mutants
  - Execute mutants (with sample test case) and obtain non-trivial mutants
    - If the are trivial (crashed, error or do not return output), then ignore them
    - If they are non-trivial (compilable/intepretale and return output), then use them
  - Evaluate mutants and obtain non-equivalent mutants
    - If collection of mutants is small (<30) then you can manually get non-equivalent mutants
    - If collection of mutants is large (>100) then you can test each of them with 10 test cases, and use the mutants that have at least one output differs the output of original program P

Step 4: Prepare and execute source test cases

- MR1 with a mutant
  - Source test case(s): SI
    - Graph: A
    - Init: [0,0]
    - Goal: [4,5]
  - Source output(s): SO
    - Paths:



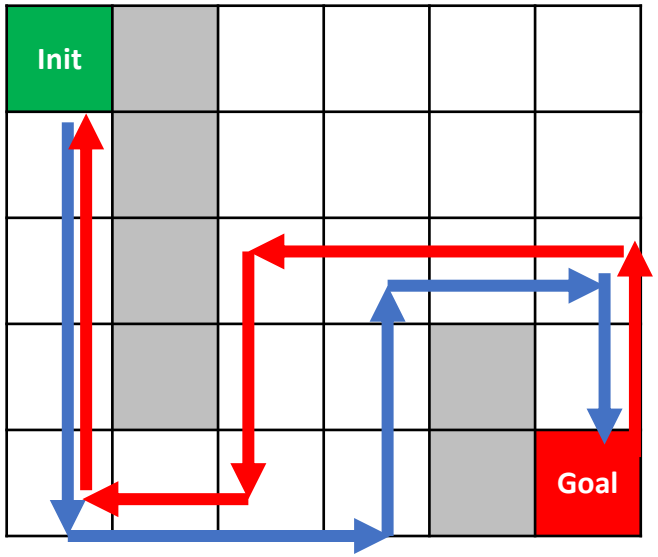
Follow-up test case (SI)



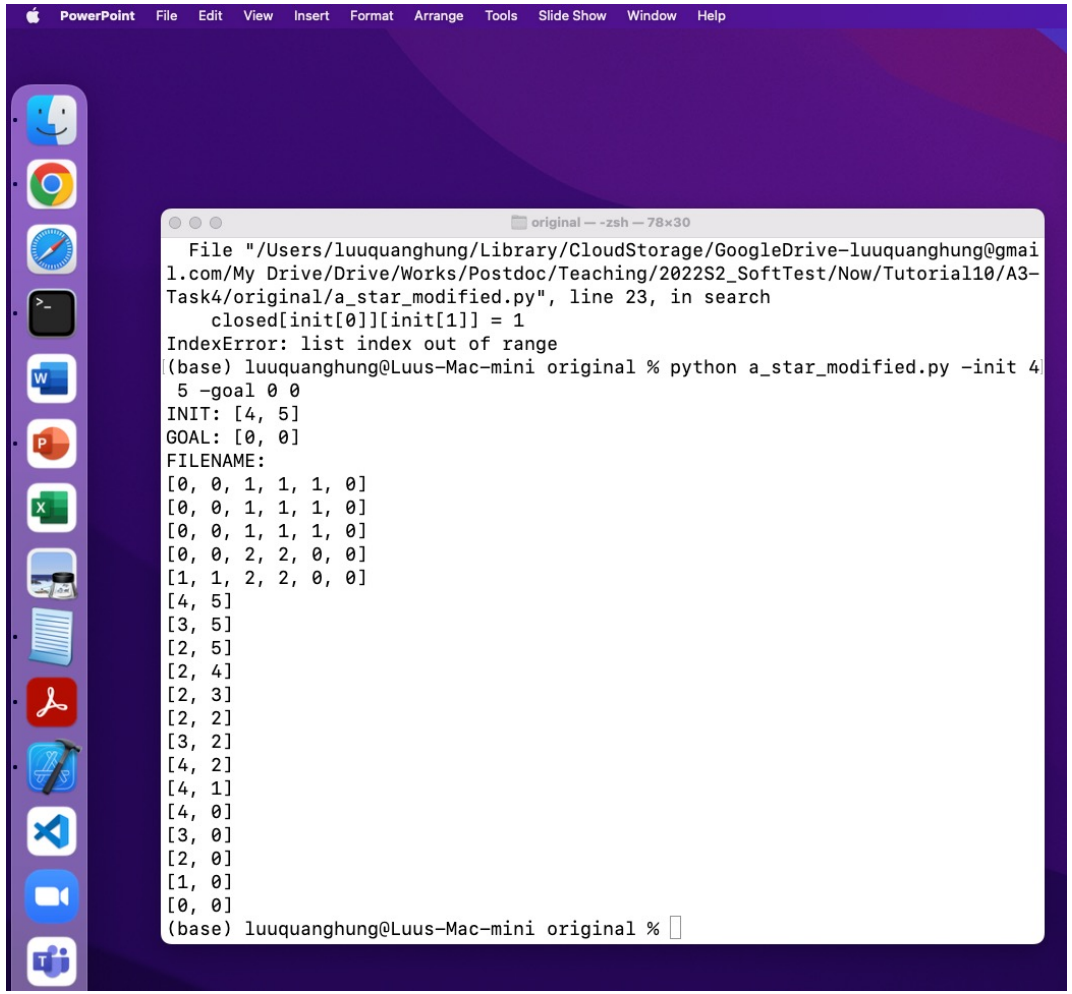
Mutant execution

Step 5: Generate and execute follow-up test cases

- MR1 with a mutant
  - Follow-up test case(s): FI
    - Graph: A
    - Init: **[4,5]**
    - Goal: **[0,0]**
  - Follow-up output(s): FO
    - Paths:



Follow-up test case (FI)



Mutant execution

Step 6: Verify MRs

- MR satisfaction condition:
  - n(FO) == n(SO)
- Checking result:
  - 13 == 13
- Conclusion for testing this mutant with this test group using this MR:
  - No violation
- Post-processing:
  - Add the result to mutation score

```
INIT: [0, 0]
GOAL: [4, 5]
FILENAME:
[0, 0, 0, 0, 0, 0]
[2, 0, 0, 0, 0, 0]
[2, 0, 0, 0, 3, 3]
[2, 0, 0, 0, 0, 2]
[2, 3, 3, 3, 0, 2]
[0, 0]
[1, 0]
[2, 0]
[3, 0]
[4, 0]
[4, 1]
[4, 2]
[4, 3]
[3, 3]
[2, 3]
[2, 4]
[2, 5]
[3, 5]
[4, 5]
```

Source output (SO)

```
INIT: [4, 5]
GOAL: [0, 0]
FILENAME:
[0, 0, 1, 1, 1, 0]
[0, 0, 1, 1, 1, 0]
[0, 0, 1, 1, 1, 0]
[0, 0, 2, 2, 0, 0]
[1, 1, 2, 2, 0, 0]
[4, 5]
[3, 5]
[2, 5]
[2, 4]
[2, 3]
[2, 2]
[3, 2]
[4, 2]
[4, 1]
[4, 0]
[3, 0]
[2, 0]
[1, 0]
[0, 0]
```

Follow-up output (FO)

Step 7: Report results (suggestion)

- Table 1: List of Metamorphic Relations (MRs)
- Table 2: List of source test cases
  - Test case ID, test case values
- Table 3: List of non-equivalent and non-trivial mutants
  - Mutant ID, line with original code, line with changed code, mutation operator
- Table 4: Effectiveness of MRs
  - Mutation scores for various MRs, e.g., MR1, MR2, etc.