COMP9021 Principles of Programming
Term 2, 2024

Assignment 2

Worth 13marks and due Week 11 Monday @ 10am

1. General Matters

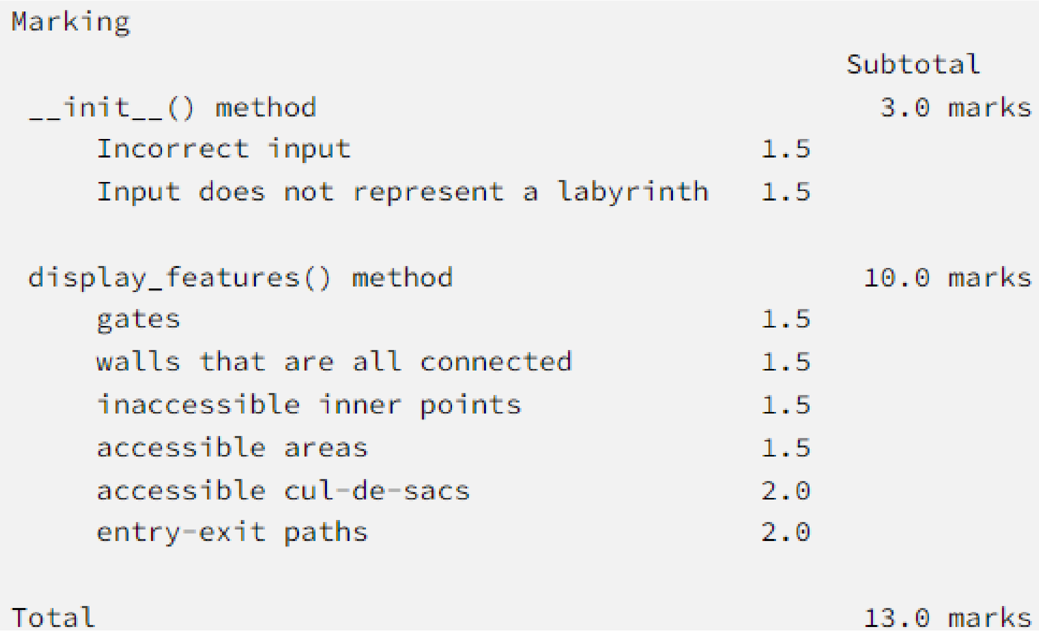
1.1 Aim

The purpose of this assignment is to:

- Develop your **problem-solving** skills.
- Design and implement the solution to a problem in the form of a **medium** sized Python program.
- Analyse the various **features** of a **labyrinth**, represented by a particular coding of its basic elements into numbers, from the set **{0, 1, 2, 3}** only, stored in a **text file**.
- Check that the input text file is **correct** and represents a **labyrinth**.
- Output the labyrinth features **nicely**.
- Use **object-oriented** programming.

1.2 Marking

This assignment is worth **13 marks** distributed as follows:



```
Marking
                                                 Subtotal
 __init__() method                                 3.0 marks
     Incorrect input                        1.5
     Input does not represent a labyrinth   1.5

 display_features() method                        10.0 marks
     gates                                  1.5
     walls that are all connected           1.5
     inaccessible inner points              1.5
     accessible areas                       1.5
     accessible cul-de-sacs                 2.0
     entry-exit paths                       2.0

Total                                             13.0 marks
```

Your program will be tested against several inputs. For each test, the auto-marking script will let your program run for **30 seconds**. The outputs of your program should be **exactly** as indicated.

1.3 Due Date and Submission

Your programs will be stored in a file named **labyrinth.py**. The assignment can be submitted more than once. The last version just before the due date and time will be marked (unless you submit late in which case the last late version will be marked).

Assignment 2 is due **Week 11 Monday 5 August 2024 @ 10:00am** (Sydney time).

Please note that **late** submission with **5% penalty per day** is allowed **up to 5 days** from the due date, that is, any late submission after **Week 11 Saturday 10 August 2024 @ 10:00am** will be discarded.

Please make sure not to change the filename **labyrinth.py** while submitting by clicking on **[Mark]** button in **Ed**. It is your responsibility to check that your submission did go through properly using **Submissions** link in Ed otherwise your mark will be **zero** for **Assignment 2**.

1.4 Reminder on Plagiarism Policy

You are permitted, indeed encouraged, to discuss ways to solve the assignment with other people. Such discussions must be in terms of **algorithms**, **not code**. But you **must implement the solution on your own**. Submissions are **scanned for similarities** that occur when students copy and modify other people's work or work very closely together on a single implementation. Severe penalties apply.

2. Description

The representation of the **labyrinth** is based on a **coding** with only the four digits **0**, **1**, **2** and **3** such that:

- **0** codes points that are **connected** to **neither** their **right** nor their **below** neighbours: □
- **1** codes points that are **connected** to their **right** neighbours but **not** to their **below** ones: □
- **2** codes points that are **connected** to their **below** neighbours but **not** to their **right** ones: □
- **3** codes points that are **connected** to both their **right** and **below** neighbours: □

A **point** that is connected to **none** of their **left**, **right**, **above**, and **below** neighbours represents a **pillar**: ●

Analysing the labyrinth will also allow to represent:

- **cul-de-sac:** ✗
- **entry-exit path:** ▬ ▬ ▬

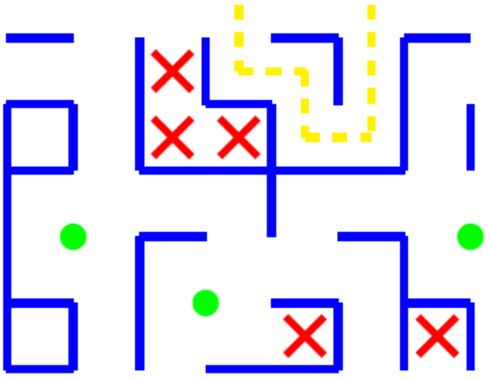
3. Examples

3.1 First Example

The file named **labyrinth_1.txt** contains the following:

```
1 0 2 2 1 2 3 0
3 2 2 1 2 0 2 2
3 0 1 1 3 1 0 0
2 0 3 0 0 1 2 0
3 2 2 0 1 2 3 2
1 0 0 1 1 0 0 0
```

As per the coding above, **labyrinth_1.txt** will look like the following:



Here is a possible interaction:

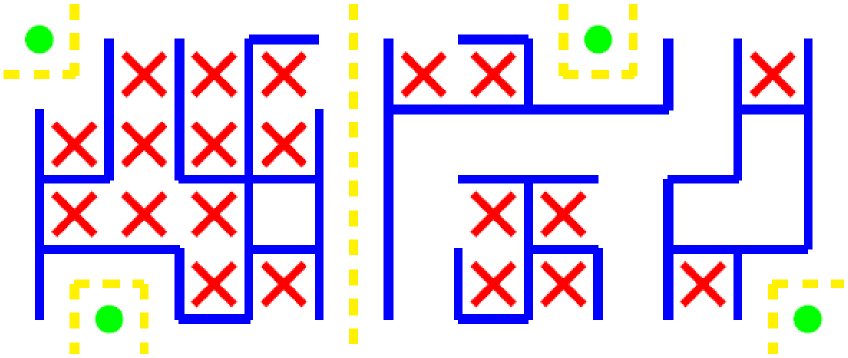
```
$ python3
...
>>> from labyrinth import *
>>> lab = Labyrinth('labyrinth_1.txt')
>>> lab.display_features()
The labyrinth has 12 gates.
The labyrinth has 8 sets of walls that are all connected.
The labyrinth has 2 inaccessible inner points.
The labyrinth has 4 accessible areas.
The labyrinth has 3 sets of accessible cul-de-sacs that are all connected.
The labyrinth has a unique entry-exit path with no intersection not to cul-de-sacs.
>>>
```

3.2 Second Example

The file named **labyrinth_2.txt** contains the following:

```
022302120222
222223111032
301322130302
312322232330
001000100000
```

As per the coding above, **labyrinth_2.txt** will look like the following:



Here is a possible interaction:

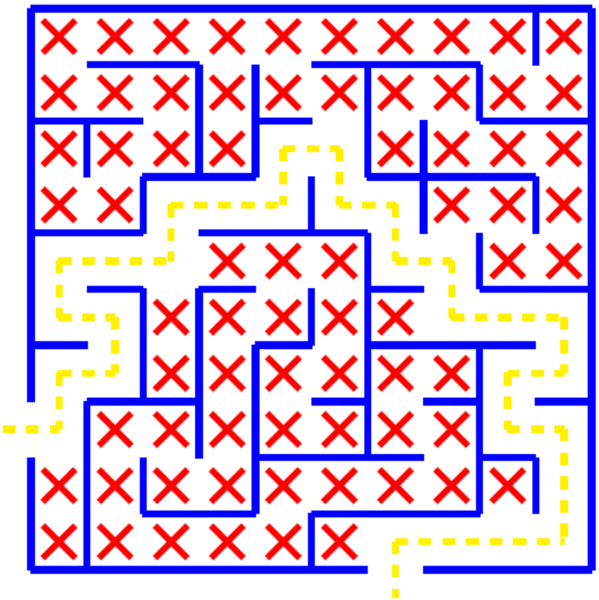
```
$ python3
...
>>> from labyrinth import *
>>> lab = Labyrinth('labyrinth_2.txt')
>>> lab.display_features()
The labyrinth has 20 gates.
The labyrinth has 4 sets of walls that are all connected.
The labyrinth has 4 inaccessible inner points.
The labyrinth has 13 accessible areas.
The labyrinth has 11 sets of accessible cul-de-sacs that are all connected.
The labyrinth has 5 entry-exit paths with no intersections not to cul-de-sacs.
>>>
```

3.3 Third Example

The file named **labyrinth_3.txt** contains the following:

```
31111111132
21122131202
33023022112
20310213122
31011120202
21230230112
30223031302
03122121212
22203110322
22110311002
11111101110
```

As per the coding above, **labyrinth_3.txt** will look like the following:



Here is a possible interaction:

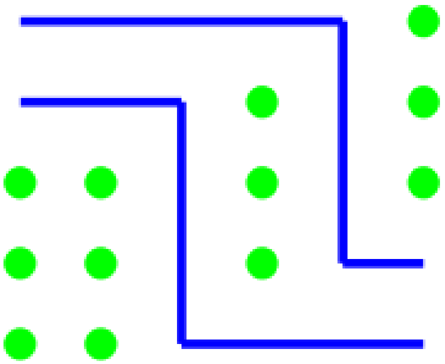
```
$ python3
...
>>> from labyrinth import *
>>> lab = Labyrinth('labyrinth_3.txt')
>>> lab.display_features()
The labyrinth has 2 gates.
The labyrinth has 2 sets of walls that are all connected.
The labyrinth has no inaccessible inner point.
The labyrinth has a unique accessible area.
The labyrinth has 8 sets of accessible cul-de-sacs that are all connected.
The labyrinth has a unique entry-exit path with no intersection not to cul-de-sacs.
>>>
```

3.4 Fourth Example

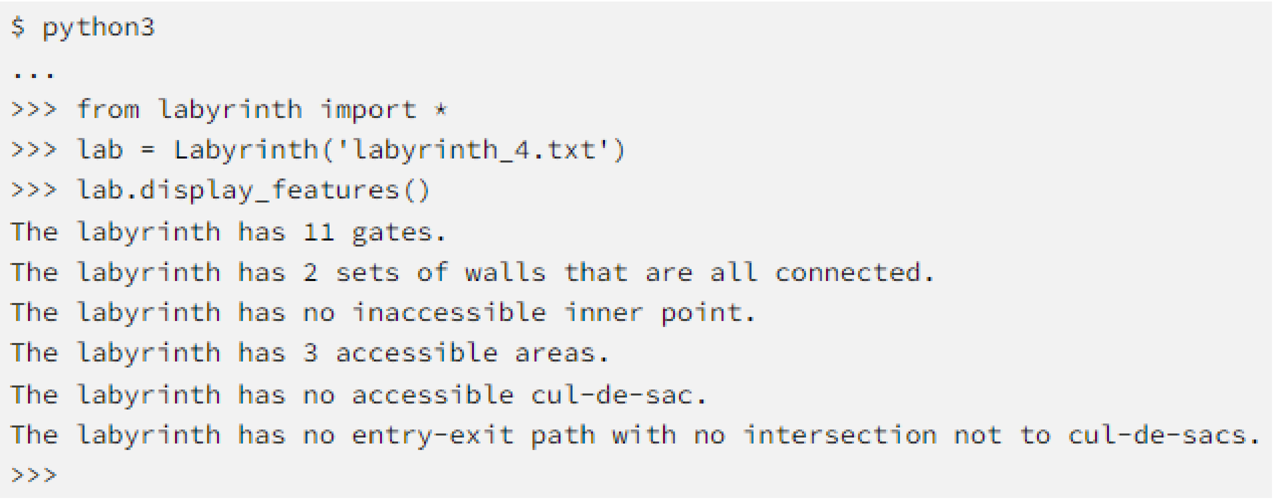
The file named **labyrinth_4.txt** contains the following:

```
111120
112020
002020
002010
001110
```

As per the coding above, **labyrinth_4.txt** will look like the following:



Here is a possible interaction:



```
$ python3
...
>>> from labyrinth import *
>>> lab = Labyrinth('labyrinth_4.txt')
>>> lab.display_features()
The labyrinth has 11 gates.
The labyrinth has 2 sets of walls that are all connected.
The labyrinth has no inaccessible inner point.
The labyrinth has 3 accessible areas.
The labyrinth has no accessible cul-de-sac.
The labyrinth has no entry-exit path with no intersection not to cul-de-sacs.
>>>
```

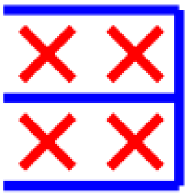
3.5 Fifth Example

The file named **labyrinth_5.txt** contains the following:

```
1  1  2
  1  12

 1 1       0
```

As per the coding above, **labyrinth_5.txt** will look like the following:



Here is a possible interaction:

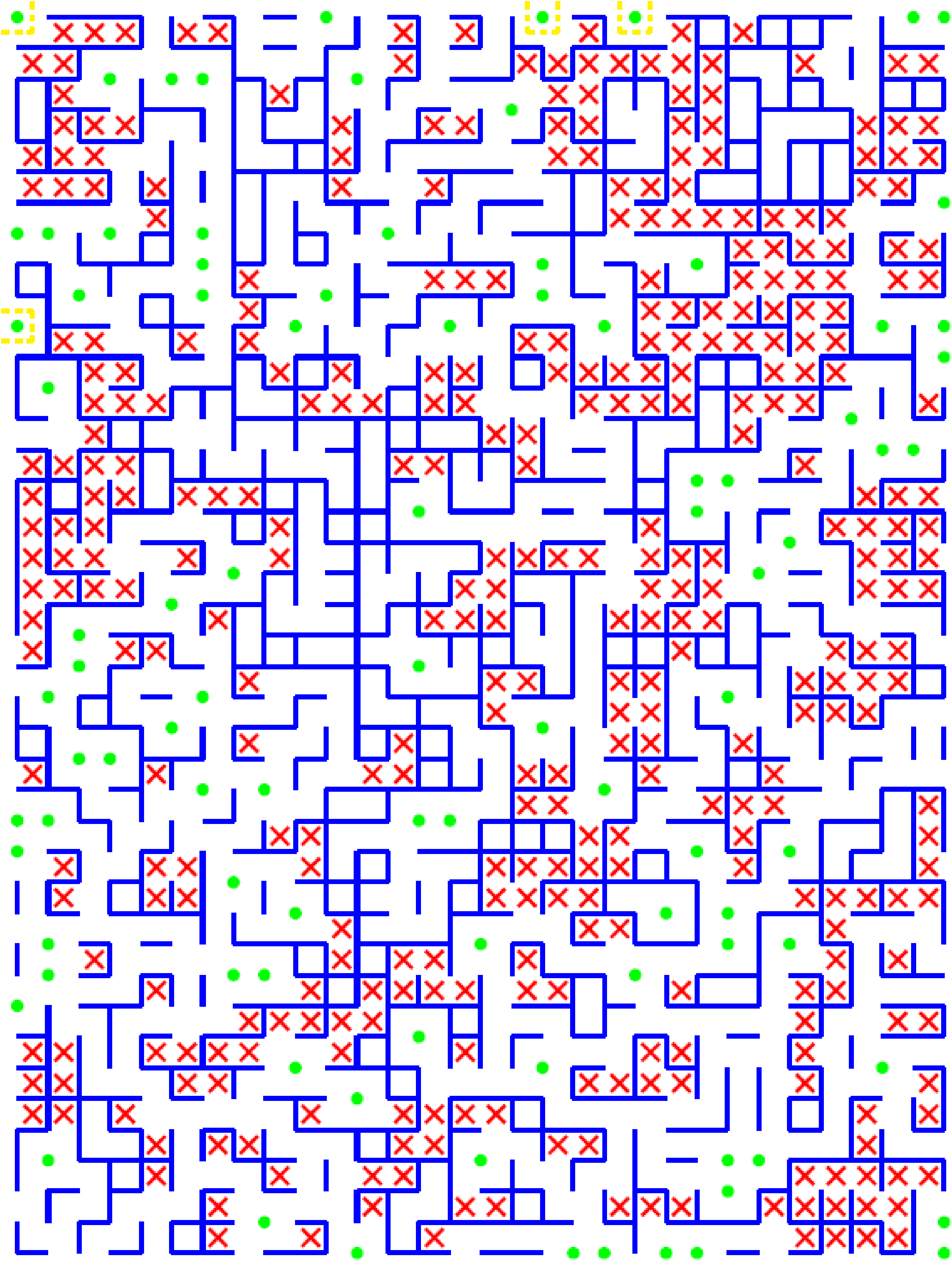
```
$ python3
...
>>> from labyrinth import *
>>> lab = Labyrinth('labyrinth_5.txt')
>>> lab.display_features()
The labyrinth has 2 gates.
The labyrinth has walls that are all connected.
The labyrinth has no inaccessible inner point.
The labyrinth has 2 accessible areas.
The labyrinth has 2 sets of accessible cul-de-sacs that are all connected.
The labyrinth has no entry-exit path with no intersection not to cul-de-sacs.
>>>
```

3.6 Sixth Example

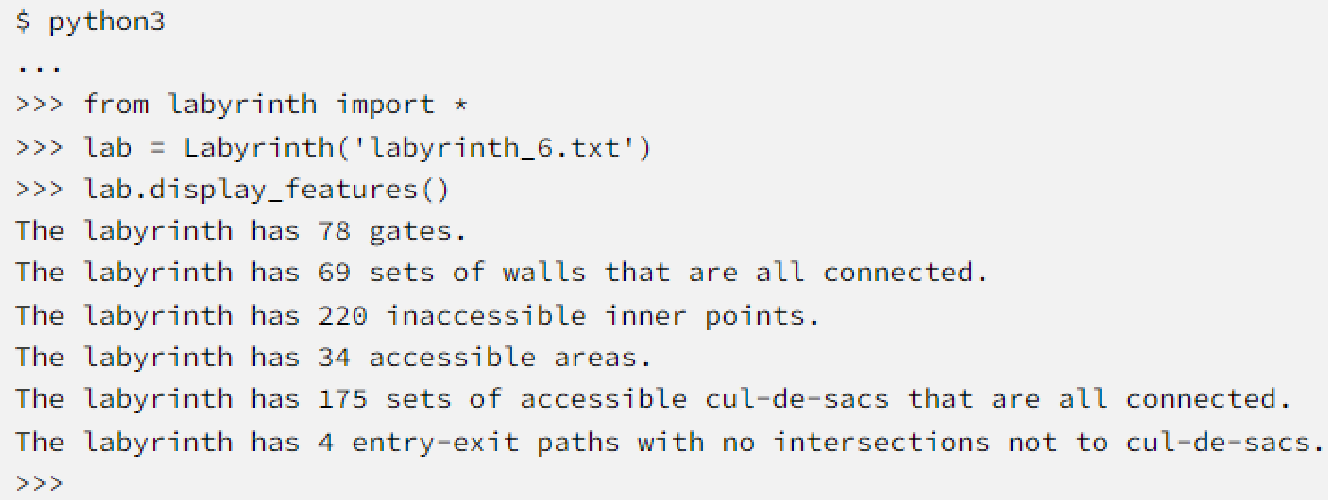
The file named **labyrinth_6.txt** contains the following:

```
0111221210021212201201323330200
1131011210301210323130331302310
3300200323203011010332232320332
2330312230220302032203223112310
1211020213223110101302032332212
1112222331300131013312312223130
1110120222130303102210113130100
0020320223200020113011131202302
3213100301021111202122030110112
1200320130030313001031302312110
0310130300203000312021011212020
3131210323320222322212332031120
2021033213003230103112331130222
1013300313133312220131222100010
1231322020123133221023110302002
3322231133223020311032001102310
2323101332332011010132023031132
2200112102133112231003320012120
2130210032120230313010200102112
2311003320123302322223032120112
0201320233130133202333323012120
1003010311132003122332112221132
2032100103021312110202200332310
3211202310233321202022122021022
1200330103010322222131033003000
1121200200313110310030131100332
0012021023310003332301123031222
0120302110232113312232012023020
2303222021330232031311210131312
0101112200130031003120203121102
2012100113233120321011002012220
0010322001332202013202312122110
0213000131303132112301110130112
1222303101032032301032210302110
1220131300113200003021222120012
1333010013001231321110022323022
3021223221133030131221100102210
2013320130211320203021000311312
0303003010033021320232203222020
2030233001202311110121110133220
1001011011001101100000010100100
```

As per the coding above, **labyrinth_6.txt** will look like the following:



Here is a possible interaction:



```
$ python3
...
>>> from labyrinth import *
>>> lab = Labyrinth('labyrinth_6.txt')
>>> lab.display_features()
The labyrinth has 78 gates.
The labyrinth has 69 sets of walls that are all connected.
The labyrinth has 220 inaccessible inner points.
The labyrinth has 34 accessible areas.
The labyrinth has 175 sets of accessible cul-de-sacs that are all connected.
The labyrinth has 4 entry-exit paths with no intersections not to cul-de-sacs.
>>>
```

4. Detailed Description

4.1 Input

The **input** is expected to consist of **y_{dim} lines** of **x_{dim} members** of **{0, 1, 2, 3}**, where **x_{dim}** and **y_{dim}** are **at least equal to 2** and **at most equal to 31 and 41**, respectively, with possibly lines consisting of spaces only that will be ignored and with possibly spaces anywhere on the lines with digits.

The **x_{th}** digit **n** of the **y_{th}** line, with **$0 \leq x < x_{dim}$** and **$0 \leq y < y_{dim}$**, is to be:

- associated with a point situated **x * 0.5 cm** to the right and **y * 0.5 cm** below an origin,
- connected to the point **0.5 cm** to its **right** if **n = 1** or **n = 3**, and
- connected to the point **0.5 cm below** itself if **n = 2** or **n = 3**.

The **last digit on every line** with digits (that is, not on blank lines) **cannot be equal to 1 or 3**, and the **digits on the last line** with digits **cannot be equal to 2 or 3**, which ensures that the input encodes a **labyrinth**, that is, a grid of width **$(x_{dim} - 1)$ * 0.5 cm** and of height **$(y_{dim} - 1)$ * 0.5 cm** (hence of maximum width **15 cm** and of maximum height **20 cm**), with possibly gaps on the sides and inside.

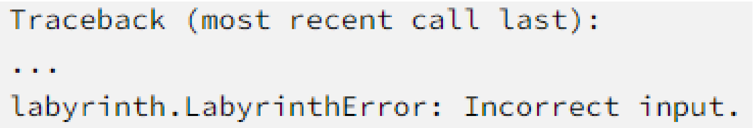
A point not connected to any of its neighbours is thought of as a **pillar** and a point connected to at least one of its neighbours is thought of as **part of a wall**.

We talk about **inner point** to refer to a point that lies **(x + 0.5) * 0.5 cm** to the **right** of and **(y + 0.5) * 0.5 cm below** the origin with **$0 \leq x < x_{dim} - 1$** and **$0 \leq y < y_{dim} - 1$**.

Practically, the **input** will be stored in a **text file** as shown in the six examples above. The program will exit immediately if the input is not as expected.

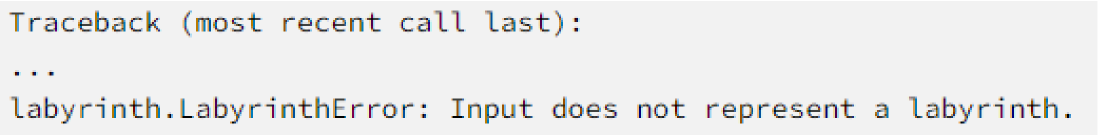
4.2 Output

Consider executing from the Python prompt the statement **from labyrinth import *** followed by the statement **lab = Labyrinth(filename)**. In case **filename** does not exist in the working directory, then Python will raise a **FileNotFoundError** exception, that **does not need to be caught**. Assume that **filename** does exist (in the working directory). If the input is **incorrect** in that it does not contain only digits in **{0, 1, 2, 3} besides spaces**, or in that it contains either **too few** or **too many nonblank lines**, or in that **some nonblank lines** contain **too many** or **too few digits**, or in that **two of its nonblank lines** do **not contain the same number of digits**, then the effect of executing **lab = Labyrinth(filename)** should be to generate a **LabyrinthError exception** that reads:



```
Traceback (most recent call last):
...
labyrinth.LabyrinthError: Incorrect input.
```

If the previous conditions hold but the further conditions spelled out above for the input to qualify as a labyrinth (that is, the condition on the **last digit on every line** with digits and the condition on the **digits on the last line**) do not hold, then the effect of executing **lab = Labyrinth(filename)** should be to generate a **LabyrinthError exception** that reads:



```
Traceback (most recent call last):
...
labyrinth.LabyrinthError: Input does not represent a labyrinth.
```

If the input is correct and represents a **labyrinth**, then **lab = Labyrinth(filename)** followed by **lab.display_features()** should have the effect of outputting the following:

1. the number of **gates**, that is, the number of **consecutive points** on one of the **four sides** of the labyrinth that are **not connected**,
2. the number of **sets of connected walls**,
3. the number of **inner points that cannot be accessed from any gate**, starting from the point in the middle of a gate and going from inner points to neighbouring inner points,
4. the number of **maximal areas that can be accessed from at least one gate** (the number of accessible inner points is at most equal to the number of gates),
5. the number of **accessible cul-de-sacs** (a cul-de-sac is a maximal set **S** of connected inner points that can all be accessed from the same gate **g** and such that for all points **p** in **S**, if **p** has been accessed from **g** for the first time, then either **p** is in a **dead end** or moving on **without ever getting back** leads into a **dead end**), and
6. the number of **entry-exit paths** with no intersections not to cul-de-sacs consisting of a maximal set **S** of connected inner points that go from a gate to another (necessarily different) gate and such that for all points **p** in **S**, there is only one way to move on from **p** without getting back and without entering a **cul-de-sac**, in other words, the resulting path is **choice-free**, that is, such that leaving the path, at any inner point where that is possible, immediately leads into a **cul-de-sac**.

The above should be displayed **exactly** as described below.

A **first line** that reads one of:

```
The labyrinth has no gate.
The labyrinth has a single gate.
The labyrinth has N gates.
```

with **N** an appropriate integer **at least equal to 2**.

A **second line** that reads one of:

```
The labyrinth has no wall.
The labyrinth has walls that are all connected.
The labyrinth has N sets of walls that are all connected.
```

with **N** an appropriate integer **at least equal to 2**.

A **third line** that reads one of:

```
The labyrinth has no inaccessible inner point.
The labyrinth has a unique inaccessible inner point.
The labyrinth has N inaccessible inner points.
```

with **N** an appropriate integer **at least equal to 2**.

A **fourth line** that reads one of:

```
The labyrinth has no accessible area.
The labyrinth has a unique accessible area.
The labyrinth has N accessible areas.
```

with **N** an appropriate integer **at least equal to 2**.

A **fifth line** that reads one of:

```
The labyrinth has no accessible cul-de-sac.
The labyrinth has accessible cul-de-sacs that are all connected.
The labyrinth has N sets of accessible cul-de-sacs that are all connected.
```

with **N** an appropriate integer **at least equal to 2**.

A **sixth line** that reads one of:

```
The labyrinth has no entry-exit path with no intersection not to cul-de-sacs.
The labyrinth has a unique entry-exit path with no intersection not to cul-de-sacs.
The labyrinth has N entry-exit paths with no intersections not to cul-de-sacs.
```

with **N** an appropriate integer **at least equal to 2**.