Artificial Intelligence HW2

By 400540213S 林亞辰

Description:

First of all, suppose there is a (H + 2) x (W + 2) maze with an outer wall, the grid content of * is the wall and cannot be walked into. The grid content 0 ~ 9 indicates the height of the grid, which can be walked into, but if you walk from a grid with height c to the next grid with height d, it costs the robot 10 + (c-d)^2 in power cost. The robot starts from the top left corner and can move to any of the four adjacent spaces. If it finally reaches the end point in the lower right corner, the task is completed.
Of course, it would be great to find the best solution that costs the least total power!

It will use the following three algorithms to perform the exercise:
1. Uniform-cost search (An algorithm likes Dijkstra's single-source-shortest-path algorithm)
2. Iterative Deeping Depth-First Search (IDS or IDDFS)
3. Iterative Deeping A* (IDA*)

Exercise Essentials
> How to display the output plate?
> How the route should be generated?
> How to implement Frontier?
> What informations to store at the node?
> How to distinguish repetition?
> Will there be infinite loops?
> Will the memory explode?
> Will the result be the best solution?
> Which heuristic result is better to use?
> How to estimate the Time complexity and Space complexity?

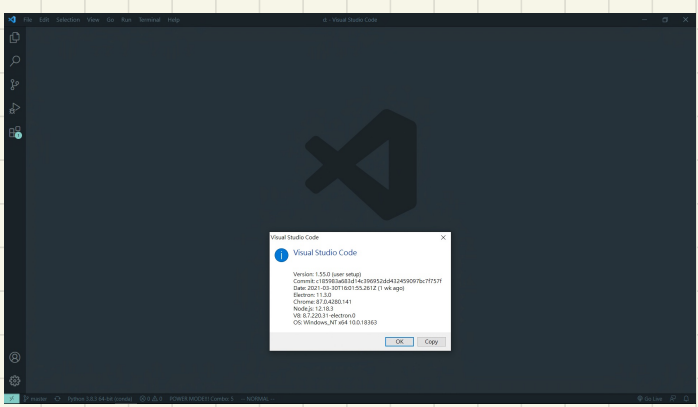
Written Report :

1. Detailed information about the hardware and software specifications of the machine you are using, the operating system, the development software version, and how to execute the program. In addition, please provide your contact phone number for your convenience.

(1) Hardware and software specifications of the machine :

```
命令提示字元
Microsoft Windows [版本 10.0.18363.1379]
(c) 2019 Microsoft Corporation. 著作權所有，並保留一切權利。

C:\Users\林亞辰>systeminfo

主機名稱:                  DESKTOP-70B4947
作業系統名稱:              Microsoft Windows 10 家用版
作業系統版本:              10.0.18363 N/A 組建 18363
作業系統製造商:            Microsoft Corporation
作業系統設定:              獨立工作站
作業系統組建類型:          Multiprocessor Free
註冊的擁有者:              N/A
註冊公司:                  N/A
產品識別碼:                00325-80000-00000-AAOEM
原始安裝日期:              2020/3/22, 11:24:20
系統開機時間:              2021/3/2, 14:40:09
系統製造商:                ASUSTeK COMPUTER INC.
系統型號:                  X510UNR
系統類型:                  x64-based PC
處理器:                    已安裝 1 處理器。
                           [01]: Intel64 Family 6 Model 142 Stepping 10 GenuineIntel ~1792 Mhz
BIOS 版本:                 American Megatrends Inc. X510UNR.309, 2019/5/14
Windows 目錄:              C:\WINDOWS
系統目錄:                  C:\WINDOWS\system32
開機裝置:                  \Device\HarddiskVolume3
系統地區設定:              zh-tw;中文 (台灣)
輸入法地區設定:            zh-tw;中文 (台灣)
時區:                      (UTC+08:00) 台北
實體記憶體總計:            16,271 MB
可用實體記憶體:            9,160 MB
虛擬記憶體: 大小上限:      18,703 MB
虛擬記憶體: 可用:          9,984 MB
虛擬記憶體: 使用中:        8,719 MB
分頁檔位置:                C:\pagefile.sys
網域:                      WORKGROUP
登入伺服器:                \\DESKTOP-70B4947
Hotfix:                    已安裝 18 Hotfix。
```

(2) Develop Software Version :



(3) How to execute the program :

可以用 make 指令來執行程式，亦可直接下達 "python filename" 來執行程式。(見下頁)

詳細資訊可參照 README.md 的內容：

README.md

If you have the `make` commend :

If your device supports the `make` command, this will be much easier (because my file name is very long...).
You can type `make` in Terminal to see the output of all Python files directly.
You can also type in :

```
>>>make
// Output all the execution results
```

```
>>>make p1
// Output the execution result of P1_UCS
```

```
>>>make p2
// Output the execution result of P2_IDS
```

```
>>>make p3
// Output the execution result of P3_IDSSTAR
```

If you do not have the `make` commend :

If your device **does not** supports the `make` command, there will be a little inconvenience (because my file name is very long...).
You can type in :

```
>>>python mazeMaker.py
// If you want to random the looks of maze in input.txt
```

```
>>>python P1_UCS.py
// Output the execution result of P1_UCS
```

```
>>>python P2_IDS.py
// Output the execution result of P2_IDS
```

```
>>>python P3_IDSSTAR.py
// Output the execution result of P3_IDSSTAR
```

(4) Contact phone number : 0972-065282

2. First you set up at least 5 input files for the test. The size of the disk should be large or small, and the difficulty of the solution should be different. Please explain how you create these input files for testing.

(1) 5x5 maze having some ways to the target

mazeMaker.py M input.txt M

1-4-4-5thGrade(II) > Artifical Intelligence > HW

```
1 *****
2 *580*
3 *311*
4 *244*
5 *****
6
```

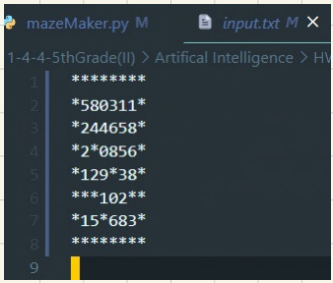
(2) 5x5 maze having no way to the target

mazeMaker.py M input.txt M

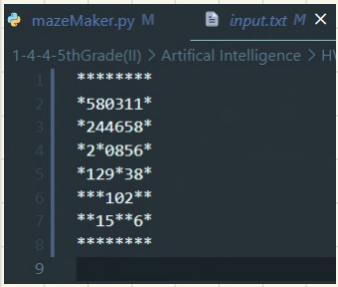
1-4-4-5thGrade(II) > Artifical Intelligence > HW

```
1 *****
2 *580*
3 *31**
4 *1*2*
5 *****
6
```

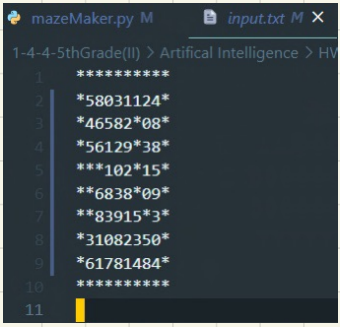
(3) 8x8 maze having some ways to the target



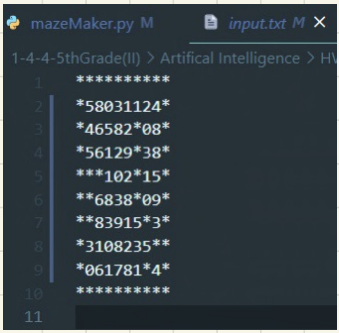
(4) 8x8 maze having no way to the target



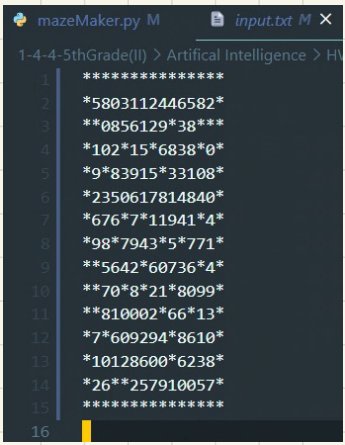
(5) 10x10 maze having some ways to the target



(6) 10x10 maze having no way to the target



(7) 15x15 maze having some ways to the target



* How to create these input files for testing:

我主要是寫另一支程式叫 mazeMaker.py 來製作測試用迷宮(詳細可見 code)，其中最核心的建構部分為 20~34 行的 def MazeMaker(maze, h, w) 這個 function:

mazeMaker.py M

1-4-4-5thGrade(II) > Artifical Intelligence > HW2 > Aritificial-Intelligence_Path-Finding-of-Maze > mazeMaker.p

```python
import numpy as np
import sys

# Global Variable
np.random.seed(1)
# needs to small than 1000 x 1000
hight = 15
width = 15
maze = [[0] * width for i in range(hight)]


def PrintMaze(m):…
# end func PeintMaze


def MazeMaker(maze, h, w):
    for i in range(h):
        for j in range(w):
            if i == 0 or j == 0 or i == h-1 or j == w-1:
                maze[i][j] = '*'
            # Set no solution statement
            # elif (i == h - 3 and j == w-2) or (i == h - 2 and j == w-3):
            #     maze[i][j] = '*'
            else:
                tmp = int(np.random.random()*12)
                if tmp >= 10:
                    maze[i][j] = '*'
                else:
                    maze[i][j] = str(tmp)
# end func MazeRandCreate


if __name__ == '__main__':…
```

23~24 行為設製迷宮牆的部分。
26~27 行為設製無解的迷宮(利用將 target 封住的方式)
29~33 行利用亂數設值，並保設計約 1/4 的機率生成"*"，
也就是牆的部分。

3. three programs (please name them P1_UCS, P2_IDS, P3_IDSSTAR) should be commented in the source code, please explain how to execute these three codes

詳見 README.md

README.md

If you have the `make` commend :

If your device supports the `make` command, this will be much easier (because my file name is very long...).
You can type `make` in Terminal to see the output of all Python files directly.
You can also type in :

```
>>>make
// Output all the execution results

>>>make p1
// Output the execution result of P1_UCS

>>>make p2
// Output the execution result of P2_IDS

>>>make p3
// Output the execution result of P3_IDSSTAR
```

If you do not have the `make` commend :

If your device **does not** supports the `make` command, there will be a little inconvenience (because my file name is very long...).
You can type in :

```
>>>python mazeMaker.py
// If you want to random the looks of maze in input.txt

>>>python P1_UCS.py
// Output the execution result of P1_UCS

>>>python P2_IDS.py
// Output the execution result of P2_IDS

>>>python P3_IDSSTAR.py
// Output the execution result of P3_IDSSTAR
```

4. Please explain what method, what data structure, and what technique you used to solve this problem with the program P1_UCS, and explain how you performed when you tested some plates, how much time and space you used horizontally (assuming plate (H+2)x(W+2)), and how large a plate your program can solve the problem? Please use some examples to help illustrate.

我主要參考講義上有關 Uniform-cost search algorithm 的敘述去查看要如何實作，理所當然用 Dijkstra's algorithm 來走路。一開始用陣列來實作，但發現存取資料相當不容易，因此改採用 class 的 data structure 來建構。不久再次遇到瓶頸，即 Graph 的部分稍嫌不足，因此參考網路上的 Graph 如何建立，經修改過後方完成 UCS。

路線的產生方式由 def ShortestPath(v, path) 此 function 來找查，其主要理念為：由 target 回推 .previous 的座標紀錄在 path 中；

```
def ShortestPath(v, path):
    # make shortest path from v.previous
    if v.previous:
        path.append(v.previous.id)
        ShortestPath(v.previous, path)
    return
# end func Shor
```

```
class Vertex:
    def __init__(self, node):
        self.id = node
        self.adjacent = {}
        # Set distance to infinity for all nodes
        self.distance = sys.maxsize
        # Mark all nodes unvisited
        self.visited = False
        # Predecessor
        self.previous = None

    def add_neighbor(self, neighbor, weight=0):
        self.adjacent[neighbor] = weight

    def get_weight(self, neighbor):
        return self.adjacent[neighbor]

    def set_distance(self, dist):
        self.distance = dist

    def set_visited(self):
        self.visited = True

    def set_previous(self, prev):
        self.previous = prev
# end class Vertex
```

class Vertex 中包含了節點所紀錄的資料；

利用 priority queue 的方式存取接下來要走的節點；

最後完成的程式不會有無窮迴圈，測試下來存放有記憶體不足的問題。

實測結果：

5x5 maze with path

```
python mazeMaker.py
The maze is :
* * * * *
* 5 8 0 *
* 3 1 1 *
* 2 4 4 *
* * * * *
python P1_UCS.py

============ P1_UCS ============

The shortest path in maze is :
* * * * *
* S 8 0 *
* v 1 1 *
* v > T *
* * * * *

The shortest path in a list is :
['1_1', '2_1', '3_1', '3_2', '3_3']
The cost is : 49

Execution time = 0.010924 sec
```

5x5 maze without path

```
python mazeMaker.py
The maze is :
* * * * *
* 5 8 0 *
* 3 1 * *
* 1 * 2 *
* * * * *
python P1_UCS.py

============ P1_UCS ============

No solution ! ( There is no route from the start-position to the
 end-position )
Execution time = 0.012919 sec
```

8x8 maze with path

```
python mazeMaker.py
The maze is :
* * * * * * * *
* 5 8 0 3 1 1 *
* 2 4 4 6 5 8 *
* 2 * 0 8 5 6 *
* 1 2 9 * 3 8 *
* * * 1 0 2 * *
* 1 5 * 6 8 3 *
* * * * * * * *
python P1_UCS.py

============ P1_UCS ============

The shortest path in maze is :
* * * * * * * *
* S 8 0 3 1 1 *
* v > > > > 8 *
* 2 * 0 8 v 6 *
* 1 2 9 * v 8 *
* * * 1 0 v * *
* 1 5 * 6 v T *
* * * * * * * *

The shortest path in a list is :
['1_1', '2_1', '2_2', '2_3', '2_4', '2_5', '3_5', '4_5', '5_5',
'6_5', '6_6']
The cost is : 184

Execution time = 0.011013 sec
```

8x8 maze without path

```
python mazeMaker.py
The maze is :
* * * * * * * *
* 5 8 0 3 1 1 *
* 2 4 4 6 5 8 *
* 2 * 0 8 5 6 *
* 1 2 9 * 3 8 *
* * * 1 0 2 * *
* * 1 5 * * 6 *
* * * * * * * *
python P1_UCS.py

============ P1_UCS ============

No solution ! ( There is no route from the start-position to the
 end-position )
Execution time = 0.009937 sec
```

10x10 maze with path

```
python mazeMaker.py
The maze is :
* * * * * * * * * *
* 5 8 0 3 1 1 2 4 *
* 4 6 5 8 2 * 0 8 *
* 5 6 1 2 9 * 3 8 *
* * * 1 0 2 * 1 5 *
* * 6 8 3 8 * 0 9 *
* * 8 3 9 1 5 * 3 *
* 3 1 0 8 2 3 5 0 *
* 6 1 7 8 1 4 8 4 *
* * * * * * * * * *
python P1_UCS.py

============ P1_UCS ============

The shortest path in maze is :
* * * * * * * * * *
* 5 8 0 3 1 1 2 4 *
* v > > 8 2 * 0 8 *
* 5 6 v 2 9 * 3 8 *
* * * v > 2 * 1 5 *
* * 6 8 v 8 * 0 9 *
* * 8 3 v 1 5 * 3 *
* 3 1 0 v > > > 0 *
* 6 1 7 8 1 4 v T *
* * * * * * * * * *

The shortest path in a list is :
['1_1', '2_1', '2_2', '2_3', '3_3', '4_3', '4_4', '5_4', '6_4',
'7_4', '7_5', '7_6', '7_7', '8_7', '8_8']
The cost is : 275

Execution time = 0.010972 sec
```

10x10 maze without path

```
python mazeMaker.py
The maze is :
* * * * * * * * * *
* 5 8 0 3 1 1 2 4 *
* 4 6 5 8 2 * 0 8 *
* 5 6 1 2 9 * 3 8 *
* * * 1 0 2 * 1 5 *
* * 6 8 3 8 * 0 9 *
* * 8 3 9 1 5 * 3 *
* 3 1 0 8 2 3 5 0 *
* 6 1 7 8 1 4 8 4 *
* * * * * * * * * *
python P1_UCS.py

============ P1_UCS ============

No solution ! ( There is no route from the start-position to the
 end-position )
Execution time = 0.009974 sec
```

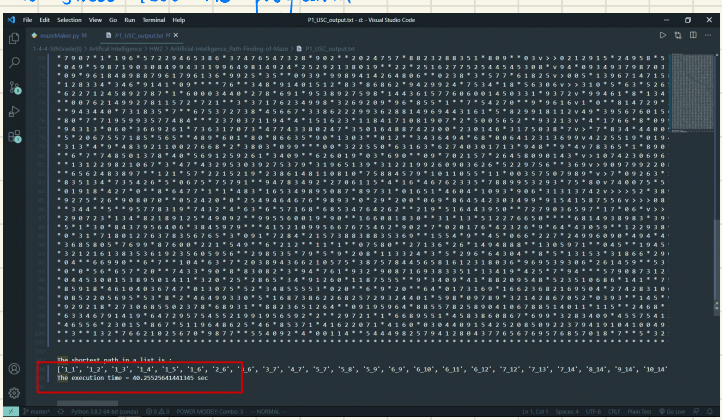
15x15 maze with path

```
python mazeMaker.py
The maze is :
* * * * * * * * * * * * * * *
* 5 8 0 3 1 1 2 4 4 6 5 8 2 *
* * 0 8 5 6 1 2 9 * 3 8 * * *
* 1 0 2 * 1 5 * 6 8 3 8 * 0 *
* 9 * 8 3 9 1 5 * 3 3 1 0 8 *
* 2 3 5 0 6 1 7 8 1 4 8 4 0 *
* 6 7 6 * 7 * 1 1 9 4 1 * 4 *
* 9 8 * 7 9 4 3 * 5 * 7 7 1 *
* * * 5 6 4 2 * 6 0 7 < v * 4 *
* * 7 0 * 8 * 2 1 * 8 0 9 9 *
* * 8 1 0 0 0 2 * 6 6 * 1 3 *
* 7 * 6 0 9 2 9 4 * 8 6 1 0 *
* 1 0 1 2 8 0 0 * 6 2 3 8 *
* 2 6 * * 2 5 7 9 1 0 0 5 7 *
* * * * * * * * * * * * * * *

python P1_UCS.py

============ P1_UCS ============

The shortest path in maze is :
* * * * * * * * * * * * * * *
* 5 > > > > > > > > > 5 8 2 *
* * 0 8 5 6 1 2 9 * v 8 * * *
* 1 0 2 * 1 5 * 6 8 * 8 * 0 *
* 9 * 8 3 9 1 5 * 3 v 1 0 8 *
* 2 3 5 0 6 1 7 8 1 v 8 4 0 *
* 6 7 6 * 7 * 1 1 9 v > * 4 *
* 9 8 * 7 9 4 3 * 5 * v 7 1 *
* * 5 6 4 2 * 6 0 7 < v * 4 *
* * 7 0 * 8 * 2 1 * v 0 9 9 *
* * 8 1 0 0 0 2 * 6 v * 1 3 *
* 7 * 6 0 9 2 9 4 * v > 1 0 *
* 1 0 1 2 8 0 0 * 6 v > 8 *
* 2 6 * * 2 5 7 9 1 0 0 v T *
* * * * * * * * * * * * * * *

The shortest path in a list is :
['1_1', '1_2', '1_3', '1_4', '1_5', '1_6', '1_7', '1_8', '1_9',
'1_10', '2_10', '3_10', '4_10', '5_10', '6_10', '6_11', '7_11',
'8_11', '8_10', '9_10', '10_10', '11_10', '11_11', '12_11', '12_
12', '13_12', '13_13']
The cost is : 482

Execution time = 0.017941 sec
```

When I tried to stress test the program...

100x100 maze

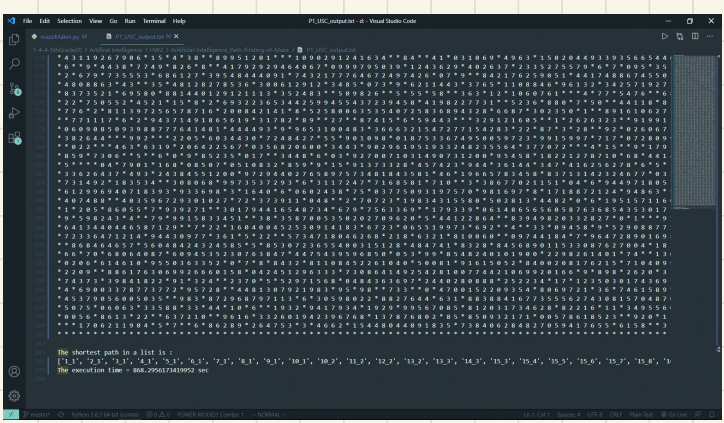
only 40 sec...



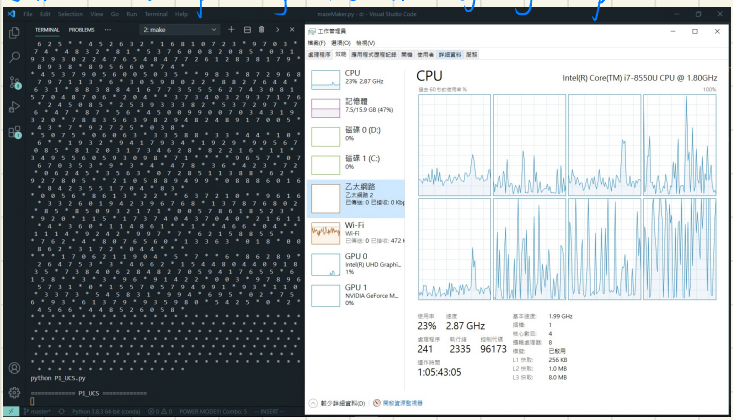
More 9

150x150 maze

about 14.5 min



But there is still room to spare of the CPU of my computer when running...



5. Please explain what method, what data structure, and what technique you used to solve this problem with the program P2_IDS, and explain how you performed when you tested some plates, how much time and space you used horizontally (assuming plate (H+2)x(W+2)), and how large a plate your program can solve the problem? Please use some examples to help illustrate.

原本是用 recursion 的方式实作，但實在是太慢了，因此嘗試用迴圈的方式实作，分成 def LDS 与 def IDDFS 兩部分。

```python
def DLS(current, target, limit, cost, path):
    current.reset()
    while(1):
        if current.id == '1_1' and len(current.adjacent) == len(current.visited_adjacent):
            current.reset()
            return False
        if current.id == target.id:
            cur_cost = current.distance
            if cur_cost < cost:
                cost = current.distance
                ResetPath(target, path)
                ShortestPath(target, path)
            prev = current.previous
            current.reset()
            current = prev
            continue
        if current.level == limit:
            prev = current.previous
            current.reset()
            current = prev
            continue
        if CheckAllVisited(current):
            prev = current.previous
            current.reset()
            current = prev
            continue

        for next in current.adjacent:
            if len(next.visited_adjacent):
                current.visited_adjacent.append(next.id)
                continue

            if next.id not in current.visited_adjacent and next.previous == None:
                current.visited_adjacent.append(next.id)
                new_dist = current.distance + current.get_weight(next)
                next.set_distance(new_dist)
                next.set_previous(current)
                # print('%s to %s' % (current.id, next))
                next.visited_adjacent.append(current.id)
                next.level = current.level + 1
                current = next
                break
    return False
# end func DLS
```

```python
def IDDFS(start, target, limit, path, aGraph):
    num = 0
    for i in range(1, limit):
        # Set the diatance for the start node to zero
        start.set_distance(0)
        start.set_level(0)
        # print("limit = %d" % (i))
        DLS(start, target, i, sys.maxsize, path)
        if len(path) > 1:
            num = num + 1
        # for v in aGraph:
        #     v.reset()
        if num >= 5:
            break
        if len(path) == 1 and i > (limit/2 + limit**0.5):
            break
# ebd func IDDFS
```

其中 IDDFS 主要用来控制 limit 数值并設立初值的部分，比較特別的是223~224与225~226的部分，223~224用来控制找解的長度，當找到第一組解則再跑5輪便回传，因為大多数 maze 是不会有繞路較有力的事件，若有也就差一點點，因此僅多 run 5 輪提前結束以減到時間上的忽益（尤其愈

後面執行愈久)。

255~256 的部分是來判斷兩種情況用的，因為考慮 maze 又有一條最蜿蜒的路

eg. 最多為 W×H+H 步(經換算迭代後可得 $\frac{limit}{2}+\sqrt{limit}$)，因此設此限制以縮短執行的時間(亦可直接放入 for 更省時，但考慮此寫法更為清晰因保留。

DLS 的部分：

① 此用來判斷是否尋回根點，即表示此路線無解。

② 判斷是否已達到終點，若是則檢查是否比之前紀錄的路徑消耗的能量更少，是則更新 cost 值與 path，接著退前一向繼續找。

③ 判斷是否到 limit 值，是則回推

④ 若 current 的點的鄰居皆造訪完了則，返回前一點繼續搜尋。

注意：此 ①,②,③,④ 的判斷皆有 current.reset() 指令目的為讓下一條路徑再次到此節點時為初始狀態可繼續向下搜尋。

⑤ for 迴圈尋找 current 的臨點，並記錄下 cost 值。

⑥ len(next.visited_queue) 若不為 0 則表示此點為自己的祖先，所以將其放入 current.visited_queue 以防再次走到形成迴路。

⑦ 若 next 未被造訪過，則更新 next 的 distance (也就是 cost) 值，並將 next 設為新的基準點(current = next)

實測結果：

5x5 maze with path

```
python mazeMaker.py
The maze is :
* * * * *
* 5 8 0 *
* 3 1 1 *
* 2 4 4 *
* * * * *

python P2_IDS.py

============ P2_IDS ============

The shortest path in maze is :
* * * * *
* S 8 0 *
* v 1 1 *
* v > T *
* * * * *

The shortest path in a list is :
['1_1', '2_1', '3_1', '3_2', '3_3']
The cost is : 49

Execution time = 0.010930 sec
```

5x5 maze without path

```
python mazeMaker.py
The maze is :
* * * * *
* 5 8 0 *
* 3 1 * *
* 1 * 2 *
* * * * *

python P2_IDS.py

============ P2_IDS ============

No solution ! ( There is no route from the start-position to the end-position )
Execution time = 0.007940 sec
```

8x8 maze with path

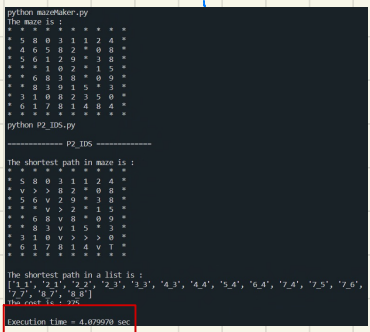
```
python mazeMaker.py
The maze is :
* * * * * * * *
* 5 8 0 3 1 1 *
* 2 4 4 6 5 8 *
* 2 * 0 8 5 6 *
* 1 2 9 * 3 8 *
* * * 1 0 2 * *
* 1 5 * 6 8 3 *
* * * * * * * *

python P2_IDS.py

============ P2_IDS ============

The shortest path in maze is :
* * * * * * * *
* S 8 0 3 1 1 *
* v > > > > 8 *
* 2 * 0 8 v 6 *
* 1 2 9 * v 8 *
* * * 1 0 v * *
* 1 5 * 6 v T *
* * * * * * * *

The shortest path in a list is :
['1_1', '2_1', '2_2', '2_3', '2_4', '2_5', '3_5', '4_5', '5_5', '6_5', '6_6']
The cost is : 184

Execution time = 0.068840 sec
```

8x8 maze without path

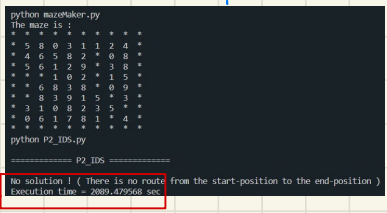
```
python mazeMaker.py
The maze is :
* * * * * * * *
* 5 8 0 3 1 1 *
* 2 4 4 6 5 8 *
* 2 * 0 8 5 6 *
* 1 2 9 * 3 8 *
* * * 1 0 2 * *
* * 1 5 * * 6 *
* * * * * * * *

python P2_IDS.py

============ P2_IDS ============

No solution ! ( There is no route from the start-position to the end-position )
Execution time = 0.554768 sec
```

時間有稍微長一點點

10×10 maze with path



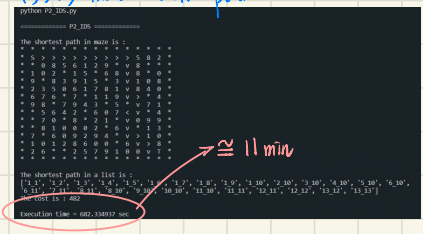
10×10 maze without path



從 8×8 → 10×10，計算的時間天差地遠，此次花了將近 35 min 😱

已經很明顯有停頓的感覺了

15×15 maze with path



≅ 11 min

⇒ 有鑑於 10×10 without path 已經跑了 34 分鐘了，因此 15×15 without path 就不做測試了。(肯定很恐怖)

另一方面，我想 11 min 已經滿久的了，而且之後的時間花費會指數飆升，因此 IDDFS 的極限可能差不多就在這裡了 (更何況是 Recursion method)

補充：

前面忘記提即此 program 的 class Vertex 是有稍作更動的：

```python
class Vertex:
    def __init__(self, node):
        self.id = node
        self.adjacent = {}
        self.visited_adjacent = []
        # Set distance to infinity for all nodes
        self.distance = sys.maxsize
        # Mark all nodes unvisited
        self.visited = False
        # Predecessor
        self.previous = None
        # Level
        self.level = int(0)

    def add_neighbor(self, neighbor, weight=0):
        self.adjacent[neighbor] = weight

    def get_weight(self, neighbor):
        return self.adjacent[neighbor]

    def set_distance(self, dist):
        self.distance = dist
```

增加限制紀錄已造訪點

```python
    def set_visited(self):
        self.visited = True

    def set_unvisited(self):
        self.visited = False

    def set_previous(self, prev):
        self.previous = prev

    def set_level(self, level):
        self.level = level

    def reset(self):
        for i in range(len(self.visited_adjacent)):
            self.visited_adjacent.pop()
        self.adjacent_num = 0
        self.visited = False
        self.level = 0
        if self.id != '1_1':
            self.previous = None
# end class Vertex
```

重設初值用

6. Please explain what method, what data structure, and what technique you used to solve this problem with the program P3_IDSSTAR, and explain how you performed when you tested some plates, how much time and space you used horizontally (assuming plate (H+2)x(W+2)), and how large a plate your program can solve the problem? Please use some examples to help illustrate.

至於 IDA* 的運作方式，類似於前兩种 Algorithm 相加，因此寫起來較為順利。
IDA* 的實作由於一方面要計算"最短距離 + weight"，另一方面也要紀錄走際的耗能，(若不另外增加一個儲存空間，它可能会忽略統返路反而耗能低的可能性，实則時即有出現此現象)，因此 class Vertex 的部分再加入变数 cost 与 distance 作區別。

```python
class Vertex:
    def __init__(self, node):
        self.id = node
        self.adjacent = {}
        self.visited_adjacent = []
        # Set distance to infinity for all nodes
        self.distance = sys.maxsize
        # Set cost to infinity for all nodes
        self.cost = sys.maxsize
        # Mark all nodes unvisited
        self.visited = False
        # Predecessor
        self.previous = None
        # Level
        self.level = int(0)

    def add_neighbor(self, neighbor, weight=0):
        self.adjacent[neighbor] = weight

    def get_weight(self, neighbor):
        return self.adjacent[neighbor]

    def set_distance(self, dist):
        self.distance = dist
```

```python
    def set_cost(self, costs):
        self.cost = costs

    def set_visited(self):
        self.visited = True

    def set_unvisited(self):
        self.visited = False

    def set_previous(self, prev):
        self.previous = prev

    def set_level(self, level):
        self.level = level

    def reset(self):
        for i in range(len(self.visited_adjacent)):
            self.visited_adjacent.pop()
        self.distance = sys.maxsize
        self.cost = sys.maxsize
        self.adjacent_num = 0
        self.visited = False
        self.level = 0
        if self.id != '1_1':
            self.previous = None
# end class Vertex
```

Algorithm 實作的 function 則分 def DLSSTAR 与 def IDDFSSTAR：

mazeMaker.py M　P3_IDSSTAR.py M
1-4-4-5thGrade(II) > Artifical Intelligence > HW2 > Aritificial-Intelligence_Path-Finding-of-Maze

```python
def DLSSTAR(start, target, limit, cost, path):
    ① unvisited_queue = [(start.distance, start.id, start)]

    while len(unvisited_queue):
    ② heapq.heapify(unvisited_queue)
        uv = heapq.heappop(unvisited_queue)
        current = uv[2]
        current.set_visited()

    ③ if current.id == target.id:
            cur_cost = current.cost
            if cur_cost < cost:
                cost = current.cost
                ResetPath(target, path)
                ShortestPath(target, path)
            continue
    ④ if current.level == limit:
            continue

    ⑤ for next in current.adjacent:
        ⑥ if next.visited:
                continue

        ⑦ next_id = next.id.split('_')
            id_y = int(next_id[0])
            id_x = int(next_id[1])
            new_dist = current.distance + current.get_weight(
                next) + ((width-2) - id_x) + ((hight-2) - id_y)
            new_cost = current.cost + current.get_weight(next)
        ⑧ if new_cost < next.cost:
                next.set_distance(new_dist)
                next.set_cost(new_cost)
                next.set_previous(current)
                next.level = current.level + 1
                heapq.heappush(unvisited_queue,
                               (next.distance, next.id, next))
# end func DLSSTAR
```

```python
def IDDFSSTAR(aGraph, start, target, limit, path):
    cost = sys.maxsize
    for i in range(1, limit):
        # Set the distance for the start node to zero
        start.set_distance(0)
        # Set the cost for the start node to zero.
        # Because we use ".distance" to filter the future path to take,
        # so the cost must be calculated separately, in case the long
        # way around but lower energy consumption is ignored.
        start.set_cost(0)
        start.set_level(0)
        DLSSTAR(start, target, i, cost, path)
        for v in aGraph:
            v.reset()
# end fund IDDFSSTAR
```

DLSSTAR 的部分：

① 結合 Dijkstra 的概念，先將起点放進 unvisited_queue 中，並且优先權順序為 distance 低（近 target）者

② 設點好後選出 distance 最低者（heapq.heappop），並將其設為已走訪

③ 若已找到 target，檢查"cost"（非 distance）是否較小，是則更新 shortest path

④ 若達極限步数則回頭

注意：有別於 IDS 的是，以上這些操作皆不用 reset，拜訪過則紀錄並不会再走訪（和 Dijkstra 用）。

⑤ for 迴圈尋找臨点。

⑥ 若已走訪則跳過。

⑦ 計算"next"前点的 distance（new_dist）和 cost（new_cost）

⑧ 若新路徑的 cost 比原本紀錄的 cost 還要低，則更新 next 点的值，包含 distance 的部分，並放入 unvisited_queue 中，加入可能走訪的 priority queue.

IDDFSSTAR 的部分則較單純，將起点設初值，便丟給 DLSSTAR 執行，回來後重設所有 vertex 的值，再給 limit，再檢測。

实測結果：

5x5 maze with path

```
python mazeMaker.py
The maze is :
*  *  *  *  *
*  5  8  0  *
*  3  1  1  *
*  2  4  4  *
*  *  *  *  *
python P3_IDSSTAR.py

============ P3_IDSSTAR ============

The shortest path in maze is :
*  *  *  *  *
*  S  8  0  *
*  v  1  1  *
*  v  >  T  *
*  *  *  *  *

The shortest path in a list is :
['1_1', '2_1', '3_1', '3_2', '3_3']
The cost is : 49

Execution time = 0.010021 sec
```

5x5 maze without path

```
python mazeMaker.py
The maze is :
*  *  *  *  *
*  5  8  0  *
*  3  1  *  *
*  1  *  2  *
*  *  *  *  *
python P3_IDSSTAR.py

============ P3_IDSSTAR ============

No solution ! ( There is no route from the start-position to the end-position )
Execution time = 0.008976 sec
```

8x8 maze with path

```
python mazeMaker.py
The maze is :
*  *  *  *  *  *  *  *
*  5  8  0  3  1  1  *
*  2  4  4  6  5  8  *
*  2  *  0  8  5  6  *
*  1  2  9  *  3  8  *
*  *  *  1  0  2  *  *
*  1  5  *  6  8  3  *
*  *  *  *  *  *  *  *
python P3_IDSSTAR.py

============ P3_IDSSTAR ============

The shortest path in maze is :
*  *  *  *  *  *  *  *
*  5  8  0  3  1  1  *
*  v  >  >  >  >  8  *
*  2  *  0  8  v  6  *
*  1  2  9  *  v  8  *
*  *  *  1  0  v  *  *
*  1  5  *  6  v  T  *
*  *  *  *  *  *  *  *

The shortest path in a list is :
['1_1', '2_1', '2_2', '2_3', '2_4', '2_5', '3_5', '4_5', '5_5', '6_5', '6_6']
The cost is : 184

Execution time = 0.018950 sec
```

8x8 maze without path

```
python mazeMaker.py
The maze is :
*  *  *  *  *  *  *  *
*  5  8  0  3  1  1  *
*  2  4  4  6  5  8  *
*  2  *  0  8  5  6  *
*  1  2  9  *  3  8  *
*  *  *  1  0  2  *  *
*  *  1  5  *  *  6  *
*  *  *  *  *  *  *  *
python P3_IDSSTAR.py

============ P3_IDSSTAR ============

No solution ! ( There is no route from the start-position to the end-position )
Execution time = 0.012965 sec
```

10×10 maze with path

```
python mazeMaker.py
The maze is :
* * * * * * * * * *
* 5 8 0 3 1 1 2 4 *
* 4 6 5 8 2 * 0 8 *
* 5 6 1 2 9 * 3 8 *
* * * 1 0 2 * 1 5 *
* * 6 8 3 8 * 0 9 *
* * 8 3 9 1 5 * 3 *
* 3 1 0 8 2 3 5 0 *
* 6 1 7 8 1 4 8 4 *
* * * * * * * * * *
python P3_IDSSTAR.py

============ P3_IDSSTAR ============

The shortest path in maze is :
* * * * * * * * * *
* S 8 0 3 1 1 2 4 *
* v > > 8 2 * 0 8 *
* 5 6 v 2 9 * 3 8 *
* * * v > 2 * 1 5 *
* * 6 8 v 8 * 0 9 *
* * 8 3 v 1 5 * 3 *
* 3 1 0 v > > > 0 *
* 6 1 7 8 1 4 v T *
* * * * * * * * * *

The shortest path in a list is :
['1_1', '2_1', '2_2', '2_3', '3_3', '4_3', '4_4', '5_4', '6_4', '7_4', '7_5', '7_6',
'7_7', '8_7', '8_8']
The cost is : 275

Execution time = 0.024934 sec
```

10×10 maze without path

```
python mazeMaker.py
The maze is :
* * * * * * * * * *
* 5 8 0 3 1 1 2 4 *
* 4 6 5 8 2 * 0 8 *
* 5 6 1 2 9 * 3 8 *
* * * 1 0 2 * 1 5 *
* * 6 8 3 8 * 0 9 *
* * 8 3 9 1 5 * 3 *
* 3 1 0 8 2 3 5 * *
* 0 6 1 7 8 1 * 4 *
* * * * * * * * * *
python P3_IDSSTAR.py

============ P3_IDSSTAR ============

No solution ! ( There is no route from the start-position to the end-position )
Execution time = 0.022918 sec
```

15×15 maze with path

```
python mazeMaker.py
The maze is :
* * * * * * * * * * * * * * *
* 5 8 0 3 1 1 2 4 4 6 5 8 2 *
* * 0 8 5 6 1 2 9 * 3 8 * * *
* 1 0 2 * 1 5 * 6 8 3 8 * 0 *
* 9 * 8 3 9 1 5 * 3 3 1 0 8 *
* 2 3 5 0 6 1 7 8 1 4 8 4 0 *
* 6 7 6 * 7 * 1 1 9 4 1 * 4 *
* 9 8 * 7 9 4 3 * 5 * 7 7 1 *
* * 5 6 4 2 * 6 0 7 3 6 * 4 *
* * 7 0 * 8 * 2 1 * 8 0 9 9 *
* * 8 1 0 0 0 2 * 6 6 * 1 3 *
* 7 * 6 0 9 2 9 4 * 8 6 1 0 *
* 1 0 1 2 8 6 0 0 * 6 2 3 8 *
* 2 6 * * 2 5 7 9 1 0 0 5 7 *
* * * * * * * * * * * * * * *
python P3_IDSSTAR.py

============ P3_IDSSTAR ============

The shortest path in maze is :
* * * * * * * * * * * * * * *
* S > > > > > > > > > 5 8 2 *
* * 0 8 5 6 1 2 9 * v 8 * * *
* 1 0 2 * 1 5 * 6 8 v 8 * 0 *
* 9 * 8 3 9 1 5 * 3 v 1 0 8 *
* 2 3 5 0 6 1 7 8 1 v 8 4 0 *
* 6 7 6 * 7 * 1 1 9 v > * 4 *
* 9 8 * 7 9 4 3 * 5 * v 7 1 *
* * 5 6 4 2 * 6 0 7 < v * 4 *
* * 7 0 * 8 * 2 1 * v 0 9 9 *
* * 8 1 0 0 0 2 * 6 v * 1 3 *
* 7 * 6 0 9 2 9 4 * v > 1 0 *
* 1 0 1 2 8 6 0 0 * 6 v > 8 *
* 2 6 * * 2 5 7 9 1 0 0 v T *
* * * * * * * * * * * * * * *

The shortest path in a list is :
['1_1', '1_2', '1_3', '1_4', '1_5', '1_6', '1_7', '1_8', '1_9', '1_10', '2_10', '3_10
', '4_10', '5_10', '6_10', '6_11', '7_11', '8_11', '8_10', '9_10', '10_10', '11_10',
'11_11', '12_11', '12_12', '13_12', '13_13']
The cost is : 482

Execution time = 0.101768 sec
```

vs.

```
python mazeMaker.py
The maze is :
* * * * * * * * * * * * * * *
* 5 8 0 3 1 1 2 4 4 6 5 8 2 *
* * 0 8 5 6 1 2 9 * 3 8 * * *
* 1 0 2 * 1 5 * 6 8 3 8 * 0 *
* 9 * 8 3 9 1 5 * 3 3 1 0 8 *
* 2 3 5 0 6 1 7 8 1 4 8 4 0 *
* 6 7 6 * 7 * 1 1 9 4 1 * 4 *
* 9 8 * 7 9 4 3 * 5 * 7 7 1 *
* * 5 6 4 2 * 6 0 7 3 6 * 4 *
* * 7 0 * 8 * 2 1 * 8 0 9 9 *
* * 8 1 0 0 0 2 * 6 6 * 1 3 *
* 7 * 6 0 9 2 9 4 * 8 6 1 0 *
* 1 0 1 2 8 6 0 0 * 6 2 3 8 *
* 2 6 * * 2 5 7 9 1 0 0 5 7 *
* * * * * * * * * * * * * * *
python P1_UCS.py

============ P1_UCS ============

The shortest path in maze is :
* * * * * * * * * * * * * * *
* S > > > > > > > > > 5 8 2 *
* * 0 8 5 6 1 2 9 * v 8 * * *
* 1 0 2 * 1 5 * 6 v 8 * 0 *
* 9 * 8 3 9 1 5 * 3 v 1 0 8 *
* 2 3 5 0 6 1 7 8 1 v 8 4 0 *
* 6 7 6 * 7 * 1 1 9 v > * 4 *
* 9 8 * 7 9 4 3 * 5 * v 7 1 *
* * 5 6 4 2 * 6 0 7 < v * 4 *
* * 7 0 * 8 * 2 1 * v 0 9 9 *
* * 8 1 0 0 0 2 * 6 v * 1 3 *
* 7 * 6 0 9 2 9 4 * v > 1 0 *
* 1 0 1 2 8 6 0 0 * 6 v > 8 *
* 2 6 * * 2 5 7 9 1 0 0 v T *
* * * * * * * * * * * * * * *

The shortest path in a list is :
['1_1', '1_2', '1_3', '1_4', '1_5', '1_6', '1_7', '1_8', '1_9',
'1_10', '2_10', '3_10', '4_10', '5_10', '6_10', '6_11', '7_11',
'8_11', '8_10', '9_10', '10_10', '11_10', '11_11', '12_11', '12_
12', '13_12', '13_13']
The cost is : 482

Execution time = 0.017941 sec
```

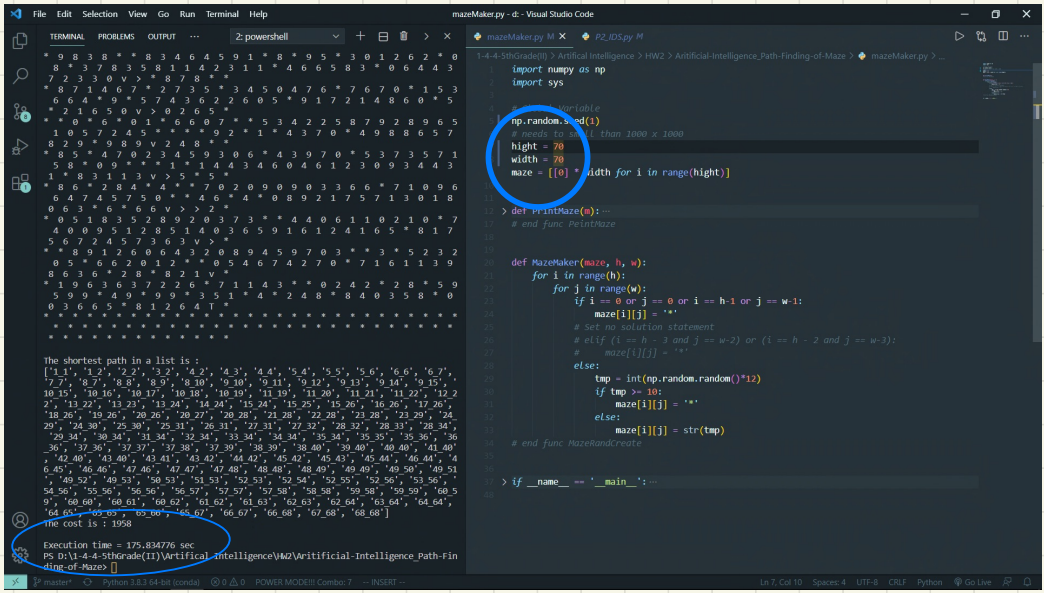
IDA* UCS

雖然皆在1秒內，但差距有5倍以上的時間。

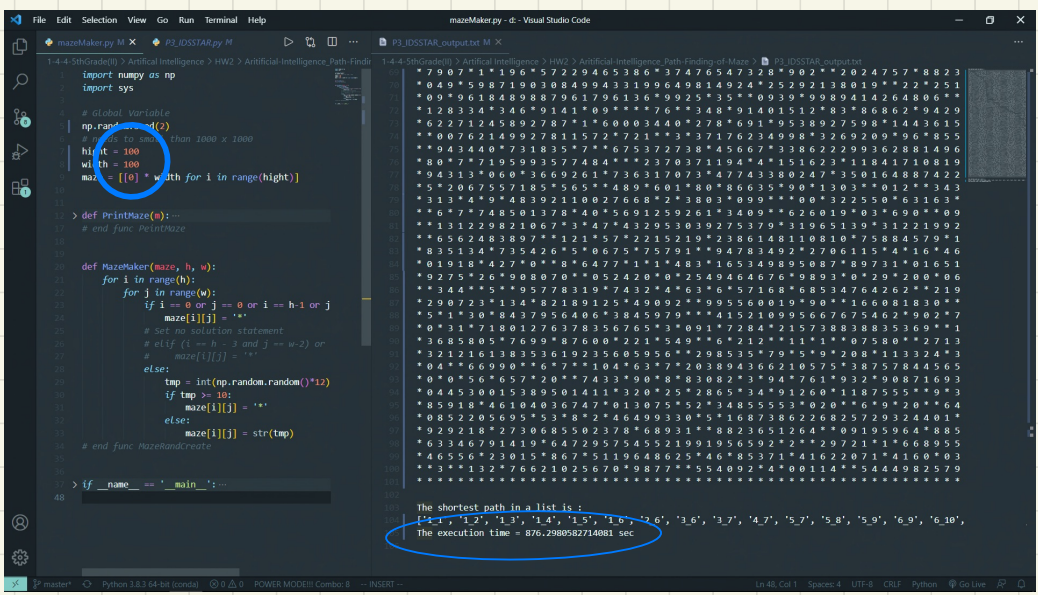
```
========== P2_IDS ==========

The shortest path in maze is :
* * * * * * * * * * * * * * *
* S > > > > > > > > > 5 8 2 *
* * 0 8 5 6 1 2 9 * v 8 * * *
* 1 0 2 * 1 5 * 6 8 v 8 * 0 *
* 9 * 8 3 9 1 5 * 3 v 1 0 8 *
* 2 3 5 0 6 1 7 8 1 v 8 4 0 *
* 6 7 6 * 7 * 1 1 9 v > * 4 *
* 9 8 * 7 9 4 3 * 5 * v 7 1 *
* * 5 6 4 2 * 6 0 7 < v * 4 *
* * 7 0 * 8 * 2 1 * v 0 9 9 *
* * 8 1 0 0 0 2 * 6 v * 1 3 *
* 7 * 6 0 9 2 9 4 * v > 1 0 *
* 1 0 1 2 8 6 0 0 * 6 v > 8 *
* 2 6 * * 2 5 7 9 1 0 0 v T *
* * * * * * * * * * * * * * *

The shortest path in a list is :
['1_1', '1_2', '1_3', '1_4', '1_5', '1_6', '1_7', '1_8', '1_9', '1_10', '2_10', '3_10', '4_10', '5_10', '6_10',
'6_11', '7_11', '8_11', '8_10', '9_10', '10_10', '11_10', '11_11', '12_11', '12_12', '13_12', '13_13']
The cost is : 482

Execution time = 682.334937 sec
```

相較於 IDS 來說則有顯著的時間壓縮

不免俗的，我們還是要來压力測試一下性能：



⇒ 175.8 sec 感覺尚有發展空間 ㋡



⇒ 876.3 sec ≅ 15 min @.@

相較於當初 P1_UCS 的 150×150 maze 花 "868 sec ≅ 14.5 min" 是差不多的結果，但格数卻差了 2.25 倍啊，可見在我的 program 中就屬 UCS 最快。

7. Please describe some of the situations and difficulties you have encountered in doing this work

(1) 其實一開始自己在嘗試寫第一個 program 時方向就已經有點偏了，我寫成 DFS 的程式碼，而且是單純用陣列的方式紀錄資料，這導致當訊息愈多時，就不得不改變儲存方式了。

(2) 最大的瓶頸在於不知道要如何建立 Graph，包含兩 node 間的 weight 要如何存取等等。因此，上網找資料後才較有頭緒，開始著手題目所要求的項目。

(3) P1_USC 基本上就是 Dijkstra's algorithm the shortest path 的應用，因此很快就完成了，難在 P2_IDS 的實作。
IDS 的簡單明瞭版本即依靠 Recursion 的方式計算，但我印象中其速度真的慢上許多，而且當伝遞的參數愈多，再加上內部判斷式拖慢速度，僅僅是 8x8 的迷宮就跑上許久。因此毅然決然將其改為迴圈形式。

(4) IDS 迴圈化花了我非常多時間思考與修正，因為在每個 vertex 中必須要存取資料的同時亦要 reset 走到 limit 的 vertex，讓下次走到此 vertex 時可以以初始狀態繼續探索（達到探索完後此點仍可再次探索 i.e 允許繞遠路）。

(5) IDA* 的部分延用 UCS 與 IDS 的概念，實作起來基本上就能應付題目，但測試的時候就會發現不一定會是最佳解，原因在於若依照原本的只紀錄 distance 來當作 priority queue 的資料，那麼若繞遠路的 cost 較小將可能被忽略，因此才要另設 "cost" 變數去計算並紀錄當前的耗量，比較是否比以往的要小，從而更新資料。

8. Please list the sources of your references (including websites) and indicate which parts were used in the assignment.

1. DIJKSTRA'S SHORTEST PATH ALGORITHM :
https://www.bogotobogo.com/python/python_Dijkstras_Shortest_Path_Algorithm.php

⇒主要參考他 Graph 與 Vertex 的 class 架構（但後來發現我的 Dijkstra 與他的幾乎一模一樣，而他的是有註解且命名更清晰，所以也跟著變更了）

2. Iterative deepening depth-first search
https://en.wikipedia.org/wiki/Iterative_deepening_depth-first_search

⇒主要參考其 IDS 的 sudo code 概念去實作 IDS。

~ END ~