EECS2040 Data Structure Hw #2 (Chapter 3 Stack/Queue)
due date 4/17/2022
by 108011234 陳昭維

Part 2 Coding (5% of final Grade)

You should submit:
(a) All your source codes (C++ file). Proper comments should be included in the code.
(b) Show the execution trace of your program.

1. (30%) Based on the circular queue and template queue ADT in **ADT 3.2** shown below (or pptx pp.82), write a C++ program to implement the queue ADT using dynamic (circular) array (10%)**.** Then add three more functions to
   (a) (5%) Return the size of a queue (int Size()).
   (b) (5%) Return the capacity of a queue (int Capacity()).
   (c) (10%) Merge two queues into a one by alternately taking elements from each queue. The relative order of queue elements is unchanged. What is the complexity of your function?

   You should **demonstrate all the functions** using at least one example.

```
template < class T >
class Queue
{
public:
    Queue (int queueCapacity = 0);
    bool IsEmpty( ) const;
    void Push(const T& item);       // add an item into the queue
    void Pop( );       // delete an item
    T& Front() const;       // return top element of stack
    T& Rear() const;       // return top element of stack
} ;
```

<Answer>

```
template<class T>
void Queue<T>::ShowProperty() {
    cout << "show property: " << endl;
    cout << "---------------------------------"<< endl;
    Show();
    cout << "queue size: " << Size() << endl;
    cout << "queue capacity: " << Capacity() << endl;
    if(size) {
        cout << "front element: " << Front() << endl;
        cout << "rear element: " << Rear() << endl;
    }

    cout << "---------------------------------"<< endl;
    cout << endl;
}
```

Demo 會多次利用到 Queue 的 class method, showProperty() 來展示 queue 的變化和內容。

其中也包含了 part(a) 的 int Size()
以及 part(b) 的 int Capacity()
以及我自己定義的 Show() function 來展示 queue 的元素，以及使用 Front()和 Rear 來看 queue 最前和最後的元素。

```
show property:
---------------------------------
queue:
  1  5  7
queue size: 3
queue capacity: 5
front element: 1
rear element: 7
---------------------------------
```

```
Queue<int> q1(5);
q1.Push(1);
q1.Push(5);
q1.Push(7);

q1.ShowProperty();
```

```
show property:
---------------------------------
queue:
  1  5  7  8  9
queue size: 5
queue capacity: 10
front element: 1
rear element: 9
---------------------------------
```

```
q1.Push(8);
q1.ShowProperty();


q1.Push(9);
q1.ShowProperty();
```

```
show property:
---------------------------------
queue:
  5  7  8  9
queue size: 4
queue capacity: 10
front element: 5
rear element: 9
---------------------------------
```

```
q1.Pop();
q1.ShowProperty();
```

```
show property:
---------------------------------
queue:
  5  7  8  9  19  26
queue size: 6
queue capacity: 10
front element: 5
rear element: 26
---------------------------------
```

```
q1.Push(19);
q1.Push(26);
q1.ShowProperty();
```

```
show property:
---------------------------------
queue:
  0  1  2  3  4  5  6
queue size: 7
queue capacity: 10
front element: 0
rear element: 6
---------------------------------
```

```
Queue<int> q2(5);
for(int k = 0; k < 7; k++) {
    q2.Push(k);
}
```

```
show property:
---------------------------------
queue:
  5  0  7  1  8  2  9  3  19  4  26  5  6
queue size: 13
queue capacity: 20
front element: 5
rear element: 6
---------------------------------

show property:
---------------------------------
queue:
empty queue
queue size: 0
queue capacity: 10
---------------------------------
```

```
q1.Merge(q2);
q1.ShowProperty();
q2.ShowProperty();
```

```
show property:
---------------------------------
queue:
  5  0  7  1  8  2  9  3  19  4  26  5  6  19  19
queue size: 15
queue capacity: 20
front element: 5
rear element: 19
---------------------------------
```

```
q1.Push(19);
q1.Push(19);
q1.ShowProperty();
```

```
show property:
---------------------------------
queue:
  7  1  8  2  9  3  19  4  26  5  6  19  19
queue size: 13
queue capacity: 20
front element: 7
rear element: 19
---------------------------------
```

```
q1.Pop();
q1.Pop();
q1.ShowProperty();
```

2. (20%) Design a C++ function template, reverseQueue, that takes as a parameter a queue object and uses a stack object to reverse the elements of the queue. The operations on queue and stack should strictly follow the ADT 3.2 Queue ADT and ADT 3.1 Stack ADT. You should **demonstrate the functions** using at least one example, e.g., queue1=(1,3,5,7), queue2=(2,4,6,8), mergedqueue=(1,2,3,4,5,6,7,8)

```
[Initializing the first queue, queue1(1, 3, 5, 7)]
[Initializing the second queue, queue2(2, 4, 6, 8)]
```

```cpp
template<class T>
void Queue<T>::ShowProperty() {
    cout << "[show property]: " << endl;
    cout << "----------------------------------"<< endl;
    Show();
    cout << "queue size: " << Size() << endl;
    cout << "queue capacity: " << Capacity() << endl;
    if(size) {
        cout << "front element: " << Front() << endl;
        cout << "rear element: " << Rear() << endl;
    }

    cout << "----------------------------------"<< endl;
    cout << endl;
}
```

Demo 會多次利用到 Queue 的 class method, showProperty() 來展示 queue 的變化和內容

```
[queue1]:                          [queue2]:
[show property]:                   [show property]:
----------------------------------  ----------------------------------
queue:                             queue:
  1  3  5  7                         2  4  6  8
queue size: 4                      queue size: 4
queue capacity: 8                  queue capacity: 8
front element: 1                   front element: 2
rear element: 7                    rear element: 8
----------------------------------  ----------------------------------
```

先用 showProperty() 展示 queue1 和 queue2 的內容

```
Reversing both queue1 and queue2

[The queue has been reversed]
(This message is from the reverseQueue function)

[The queue has been reversed]
(This message is from the reverseQueue function)
```

再利用 class method, reverseQueue() 來反轉整個 queue

```cpp
template <class T>
void Queue<T>::reverseQueue() {
        Stack<T> tempStack(size);
        while(!IsEmpty()) {
                tempStack.Push(Front());
                Pop();
        }

        while(!tempStack.IsEmpty()) {
                Push(tempStack.Top());
                tempStack.Pop();
        }
        cout << "\n[The queue has been reversed]\n(This message is from the reverseQueue function)" << endl;
}
```

```
[queue1]:
[show property]:
----------------------------------
queue:
  7  5  3  1
queue size: 4
queue capacity: 8
front element: 7
rear element: 1
----------------------------------
```

```
[queue2]:
[show property]:
----------------------------------
queue:
  8  6  4  2
queue size: 4
queue capacity: 8
front element: 8
rear element: 2
----------------------------------
```

再來我們使用第一題的 merge function 把 queue2 merge 到 queue1. 再來再使用 reverseQueue() 來反轉整個 merged 過的 queue1

Merge 過的應該會長(7,8,5,6,3,4,1,2)

Reverse 過會長(2,1,4,3,6,5,8,7)

```
Now we will merge queue2 to queue1 and then reverse queue1
[queue1]:
[show property]:
----------------------------------
queue:
  7  8  5  6  3  4  1  2
queue size: 8
queue capacity: 16
front element: 7
rear element: 2
----------------------------------
```

```
[The queue has been reversed]
(This message is from the reverseQueue function)

[queue1 after reversed queue]:
[show property]:
----------------------------------
queue:
  2  1  4  3  6  5  8  7
queue size: 8
queue capacity: 16
front element: 2
rear element: 7
----------------------------------
```

最終我們再來使用另一個 class method, CreateReverseQueue() 來把一個 queue object 便成一個 queue 的反轉版本

所以原先 empty queue3 會變成 queue1 的倒轉版本 (7,8,5,6,3,4,1,2)

```
Now we will create a empty queue: queue3

Then we will make queue3 the reversed version of queue1
by using CreateReverseQueue()
```

```
[queue3]:
[show property]:
----------------------------------
queue:
  7  8  5  6  3  4  1  2
queue size: 8
queue capacity: 12
front element: 7
rear element: 2
----------------------------------
```

```
template <class T>
void Queue<T>::CreateReverseQueue(Queue<T> &b){
        while(!b.IsEmpty()){
            b.Pop();
        }
        Stack<T> tempStack(size);

        for(int i = 1; i <= size; i++) {
                tempStack.Push(queue[(front+i) % Capacity()]);
        }

        while(!tempStack.IsEmpty()) {
                b.Push(tempStack.Top());
                tempStack.Pop();
        }

}
```

3. (25%) Referring to **Program 3.13** in textbook (pptx pp.98),
   (a) (5%) Implement Stack as a publicly derived class of Bag using template. **Demonstrate** your C++ code using at least two element types (e.g., int, float,…). **Show results** of a series of Pushes and Pops and Size functions.
   (b) (5%) Implement Queue as a publicly derived class of Bag using template. **Demonstrate** your C++ code using at least two element types (e.g., int, float,…). **Show results** of a series of Pushes and Pops and Size functions.
   (c) (15%) A template double-ended queue (deque) is a linear list in which additions and deletions may be made at either end. Implement the class Deque as a publicly derived templated class of Queue. The class Deque must have public functions (either via inheritance from Queue or by direct implementation in Deque) to add and delete elements from either end of the deque and also to return an element from either end. The complexity of each function (excluding array doubling) should be Θ(1).
   Demonstrate your C++ code using at least two element types (e.g., int, float,…). **Show results** of a series of two types of Pushes and Pops and Size functions to illustrate your code is working.

<answer>

Test for stack

```
[This is test bench for Stack]
--test pushing--

[stack1]:

Showing properties:
--------------------------------------
Size: 11
Capacity: 20
items:
[0][1][2][3][4][5][6][7][8][9][10]
--------------------------------------

[stack2]:

Showing properties:
--------------------------------------
Size: 11
Capacity: 20
items:
[0][0.00990099][0.019802][0.029703][0.039604][0.049505][0.0594059][0.0693069][0.0792079][0.0891089][0.0990099]
--------------------------------------
```

```
cout << "[This is test bench for Stack]" << endl;
cout << "--test pushing--" << endl;
for(int i = 0; i < 11; i++) {
    s1.Push(i);
    s2.Push((float)i/101);
}
```

```
--------------------------------------
--test popping--

[stack1]:

Showing properties:
--------------------------------------
Size: 4
Capacity: 20
items:
[0][1][2][3]
--------------------------------------

[stack2]:

Showing properties:
--------------------------------------
Size: 4
Capacity: 20
items:
[0][0.00990099][0.019802][0.029703]
--------------------------------------
--test pushing--

[stack1]:

Showing properties:
--------------------------------------
Size: 14
Capacity: 20
items:
[0][1][2][3][100][101][102][103][104][105][106][107][108][109]
--------------------------------------

[stack2]:

Showing properties:
--------------------------------------
Size: 14
Capacity: 20
items:
[0][0.00990099][0.019802][0.029703][0][0.0990099][0.19802][0.29703][0.39604][0.49505][0.594059][0.693069][0.792079][0.891089]
--------------------------------------
```

```
cout << "\n[stack1]:" << endl;
s1.ShowProperties();
cout << "\n[stack2]:" << endl;
s2.ShowProperties();

cout << "--test popping--" << endl;
for(int i = 0; i < 7; i++) {
    s1.Pop();
    s2.Pop();
}

cout << "\n[stack1]:" << endl;
s1.ShowProperties();
cout << "\n[stack2]:" << endl;
s2.ShowProperties();

cout << "--test pushing--" << endl;

for(int i = 0; i < 10; i++) {
    s1.Push(i+100);
    s2.Push((float)i/10.1);
}
```

Test for queue

```
*******************************
[This is testbench for queue]
*******************************
--test pushing--

[queue1]:
show property:
-----------------------------------
queue:
  0  1  2  3  4  5  6  7  8  9  10
queue size: 11
queue capacity: 20
Front element: 0
Rear element: 10
-----------------------------------


[queue2]:
show property:
-----------------------------------
queue:
  0  0.00990099  0.019802  0.029703  0.039604  0.049505  0.0594059  0.0693069  0.0792079  0.0891089  0.0990099
queue size: 11
queue capacity: 20
Front element: 0
Rear element: 0.0990099
-----------------------------------
```

```
cout << "--test pushing--" << endl;
for(int i = 0; i < 11; i++) {
    q1.Push(i);
    q2.Push((float)i/101);
}
cout << "\n[queue1]:" << endl;
q1.ShowProperties();
cout << "\n[queue2]:" << endl;
q2.ShowProperties();
```

```
--test popping--

[queue1]:
show property:
-----------------------------------
queue:
  7  8  9  10
queue size: 4
queue capacity: 20
Front element: 7
Rear element: 10
-----------------------------------


[queue2]:
show property:
-----------------------------------
queue:
  0.0693069  0.0792079  0.0891089  0.0990099
queue size: 4
queue capacity: 20
Front element: 0.0693069
Rear element: 0.0990099
-----------------------------------

--test pushing--

[queue1]:
show property:
-----------------------------------
queue:
  7  8  9  10  100  101  102  103  104  105  106  107  108  109
queue size: 14
queue capacity: 20
Front element: 7
Rear element: 109
-----------------------------------


[queue2]:
show property:
-----------------------------------
queue:
  0.0693069  0.0792079  0.0891089  0.0990099  0  0.0990099  0.19802  0.29703  0.39604  0.49505  0.594059  0.693069  0.792079  0.891089
queue size: 14
queue capacity: 20
Front element: 0.0693069
Rear element: 0.891089
-----------------------------------
```

```
cout << "--test popping--" << endl;
for(int i = 0; i < 7; i++) {
    q1.Pop();
    q2.Pop();
}

cout << "\n[queue1]:" << endl;
q1.ShowProperties();
cout << "\n[queue2]:" << endl;
q2.ShowProperties();       You, 3 minute
```

```
cout << "--test pushing--" << endl;

for(int i = 0; i < 10; i++) {
    q1.Push(i+100);
    q2.Push((float)i/10.1);
}

cout << "\n[queue1]:" << endl;
q1.ShowProperties();
cout << "\n[queue2]:" << endl;
q2.ShowProperties();
```

Test for deque

```
*******************************
[This is testbench for deque]
*******************************

[deque1]:
show property:
-----------------------------------
queue:
  9  8  7  6  5  4  3  2  1  0  100  101  102  103  104  105  106  107  108  109
queue size: 20
queue capacity: 40
Front element: 9
Rear element: 109
-----------------------------------


[deque2]:
show property:
-----------------------------------
queue:
  0.9  0.8  0.7  0.6  0.5  0.4  0.3  0.2  0.1  0  0  0.2  0.4  0.6  0.8  1  1.2  1.4  1.6  1.8
queue size: 20
queue capacity: 40
Front element: 0.9
Rear element: 1.8
-----------------------------------
```

```
for(int i = 0; i < 10; i++) {
    dq1.PushFromFront(i);
    dq2.PushFromFront((float)i/10);
    dq1.Push(i+100);
    dq2.Push((float) i/5);
}
```

```
[deque1]:
show property:
-----------------------------------
queue:
empty queue
queue size: 0
queue capacity: 40
-----------------------------------


[deque2]:
show property:
-----------------------------------
queue:
empty queue
queue size: 0
queue capacity: 40
-----------------------------------
```

```
for(int i = 0; i < 10; i++) {
    dq1.PopFromBack();
    dq1.Pop();
    dq2.PopFromBack();
    dq2.Pop();
}
```

```
[deque1]:
show property:
-----------------------------------
queue:
  11  10  9  8  7  6  5  4  3  2  50  51  52  53  54  55  56  57  58  59
queue size: 20
queue capacity: 40
Front element: 11
Rear element: 59
-----------------------------------


[deque2]:
show property:
-----------------------------------
queue:
  2.25  2  1.75  1.5  1.25  1  0.75  0.5  0.25  0  0.02  1.02  2.02  3.02  4.02  5.02  6.02  7.02  8.02  9.02
queue size: 20
queue capacity: 40
Front element: 2.25
Rear element: 9.02
-----------------------------------
```

```
for(int i = 0; i < 10; i++) {
    dq1.PushFromFront(i+2);
    dq1.Push(i+50);
    dq2.PushFromFront((float)i/4);
    dq2.Push(i+0.02);
}
```

(25%) Write a C++ program to implement the maze in textbook using the example codes of **Program 3.15** (pptx pp.106 Algorithm()) and **3.16 (pptx void Path(const int m, const int p)**. You should use a text editor to edit a file containing the maze matrix and then **read in the file to establish the maze matrix** in your program. The default entrance and exit are located in the upper left corner and lower right corner, respectively as shown in textbook.

(a) (10%) Demonstrate your maze program using the maze shown in **Figure 3.11**.

(b) (5%) Find a path manually through the maze shown in **Figure 3.11**.

(c) (10%) Trace out the action of function path (**Program 3.16**) on the maze shown in Figure 3.11. Compare this to your own attempt in (b).

<answer>

```
// map structure
You, yesterday | 1 author (You)
typedef struct {
    int dx, dy;
} Offsets;

You, yesterday | 1 author (You)
typedef struct {
    int x, y;
    int dir;
} Items;

enum directions {E, S, SE, NE, SW, W, N, NW};
```

```
// initialize variable
Offsets navigation[8];
stack<Items> route;
string mazeFileName = "maze.txt";
vector<vector<int>> maze;
vector<vector<int>> mark;
```

Initialize some of the basic variables for the map trace, Offsets struct for the navigation direction, we will create an Offsets array for different (y,x) tuple to move.

Then create Items structure for later stack objects that stores direction and current position.

And then initialize the maze file name as string.

Initialize a vector of vector<int> to store the map element. Also initialize a identical size map to mark the visited grid.

```
// function initialization
void initialize(int&, int&);
void initialize_mark(int, int);
void navigate();
void path(int, int);
void showMaze(int, int);
void ShowMazeAfterTraverse(int, int);
bool path_recursive(int, int, int&, int&);
```

Some function initializations

```
// operator overloading
ostream& operator<<(ostream&, Items&);
template<class T>
ostream& operator<<(ostream&, stack<T>&);
```

```
void initialize(int& w, int& l) {
    ifstream in_file(mazeFileName);

    if(!in_file.is_open())
        cerr << "Cannot open the given filename.";
    else {
        in_file >> w;
        in_file >> l;
```

Read file from given file name.

Output exceptions if file not found.

Read the width and length of the map in the first row of the text file

```
    maze.clear();
    vector<int> tempRow;
    char temp;

    for(int i = -1; i <=w; i++)
        tempRow.push_back(3);

    maze.push_back(tempRow);

    for(int y = 1; y <= l; y++) {
        tempRow.clear();
        tempRow.push_back(3);
        for(int x = 1; x <= w; x++) {
            in_file >> temp;
            if(x == w && y == l)
                tempRow.push_back(2);
            else
                tempRow.push_back(temp - '0');
        }
        tempRow.push_back(3);
        maze.push_back(tempRow);
    }

    tempRow.clear();
    for(int i = -1; i <=w; i++)
        tempRow.push_back(3);

    maze.push_back(tempRow);
}
```

Read file from given file name.

Output exceptions if file not found.

Read the width and length of the map in the first row of the text file

Then sequentially read every grid data and push it into a tempRow vector, and after one row, the program pushes the vector to the map (vector of vector).

We also pushes boundary grid which is consists of integer 3.

```
/* initialize_mark*/
/*************************************************************
 * @brief this initialize the identical map but with all entries*
 * of 0 for the use of marking visited grid of the maze        *
 *                                                             *
 * @param w takes the map width as variable                    *
 * @param l takes the map length as variable                   *
 *************************************************************/
void initialize_mark(int w, int l) {···
```

```
/* navigate*/
/*******************************************************
 * @brief this function sets the navigation directions*
 *                                                     *
 *******************************************************/
void navigate() {
   navigation[E].dx = 1;  navigation[E].dy = 0;
   navigation[SE].dx = 1; navigation[SE].dy = 1;
   navigation[S].dx = 0;  navigation[S].dy = 1;
   navigation[SW].dx = -1;navigation[SW].dy = 1;
   navigation[W].dx = -1; navigation[W].dy = 0;
   navigation[NW].dx = -1;navigation[NW].dy = -1;
   navigation[N].dx = 0;  navigation[N].dy = -1;
   navigation[NE].dx = 1; navigation[NE].dy = -1;
}
```

```
/*showMaze*/
/*************************************************************
 * @brief This shows the whole map with numerical representation *
 * where 0 stands for trailable path                           *
 *       1 stands for untrailable path                         *
 *       2 stands for the exit.                                *
 *                                                             *
 * @param w takes map width as variable                        *
 * @param l takes map length as variable                       *
 *************************************************************/
void showMaze(int w, int l) {
    for (int y = 0; y <= l + 1; y++) {
        for (int x = 0; x <= w + 1; x++) {
            cout << maze[y][x] << " ";
        }
        cout << endl;
    }
}
```

```
/*ShowMazeAfterTraverse*/
/*******************************************************
 * @brief This function has identical purpose as show maze*
 * but since we modified the map if there is way out so we*
 * make the map graphicalize by denoting                  *
 * 0 into ' '                                             *
 * 1 into '//'                                            *
 * 2 into '.  '                                           *
 * 3 into '##'                                            *
 * @param w takes map width as variable                    *
 * @param l takes map length as variable                   *
 *******************************************************/
void ShowMazeAfterTraverse(int w, int l) {···
```

```
/* path */
/*************************************************************
 * @brief This function uses iterative method to find whether   *
 * there is way out to the maze                                 *
 *                                                              *
 * @param w takes the map width as variable                     *
 * @param l takes the map length as variable                    *
 *************************************************************/
void path(int w, int l) {···
```

Driver code:

```
// main driver code
int main() {
    int width;
    int length;
    initialize(width, length);
    initialize_mark(width,length);
    navigate();
    showMaze(width, length);
    path(width, length);
    ShowMazeAfterTraverse(width, length);
    //cout << path_recursive(1, 1, width, length) << endl;
    cout << "end" << endl;
    return 0;
}
```

First, we initialize width and length variables, then we initialize the map and the mark vector of vectors.

Then we initialize the navigation direction.

And show the raw map before traversing.

Use the path function to find the path out and list out the path

Then show the map after traversing.

(a)

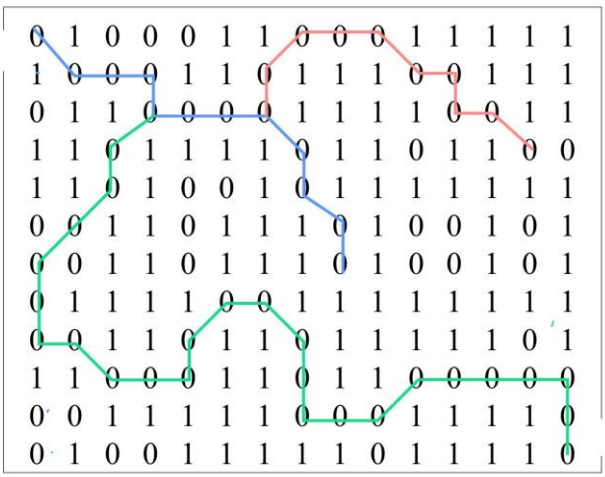
```
3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
3 0 1 0 0 0 1 1 0 0 0 1 1 1 1 1 1 3
3 1 0 0 0 1 1 0 1 1 1 0 0 1 1 1 3
3 0 1 1 0 0 0 0 1 1 1 1 0 0 1 1 3
3 1 1 0 1 1 1 1 1 0 1 1 0 1 1 0 0 3
3 1 1 0 1 0 0 1 0 1 1 1 1 1 1 1 3
3 0 0 1 1 0 1 1 1 0 1 0 0 1 0 1 3
3 0 0 1 1 0 1 1 1 0 1 0 0 1 0 1 3
3 0 1 1 1 1 0 0 1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

```
Success path:
  1,   1, SE
  2,   2, E
  2,   3, E
  2,   4, S
  3,   4, SW
  4,   3, S
  5,   3, SW
  6,   2, S
  7,   2, SW
  8,   1, S
  9,   1, E
  9,   2, SE
  10,   3, E
  9,   5, NE
  9,   5, NE
  8,   6, E
  8,   7, SE
  9,   8, S
  10,   8, S
  11,   8, E
  11,   9, E
  11,   10, NE
  10,   11, E
  10,   12, E
  10,   13, E
  10,   14, E
  10,   15, S
  11,   15, S
  12,   15
```

```
###################################
##. //      ////      //////////##
##//. . . ////  //////    //////##
##  ////.       ////////    ////##
##////. ////////  ////  ////    ##
##////. //    //  ////////////////##
##  . ////  //////  //    //  //##
##  . ////  //////  //    //  //##
##. ////////. . ////////////////##
##. . ////. ////. //////////  //##
##////. . . ////. ////. . . . . ##
##    //////////. . . ////////. ##
##  //    //////////  ////////. ##
###################################
end
```

(b)

入口



出口

My attempt has 2 failure before finding the path out.

(c)

```
pushed in:   1,   1, E
popped out:  1,   1, E
pushed in:   1,   1, SE
pushed in:   2,   2, E
pushed in:   2,   3, E
pushed in:   2,   4, S
pushed in:   3,   4, E
pushed in:   3,   5, E
pushed in:   3,   6, E
pushed in:   3,   7, SE
```

```
pushed in:   4,   8, S
pushed in:   5,   8, SE
pushed in:   6,   9, S
popped out:  6,   9, S
popped out:  5,   8, SE
popped out:  4,   8, S
popped out:  3,   7, SE
pushed in:   3,   7, N
pushed in:   2,   7, NE
```

```
pushed in:   1,   8, E
pushed in:   1,   9, E
pushed in:   1,  10, SE
pushed in:   2,  11, E
pushed in:   2,  12, S
pushed in:   3,  12, E
pushed in:   3,  13, SE
pushed in:   4,  14, E
popped out:  4,  14, E
```

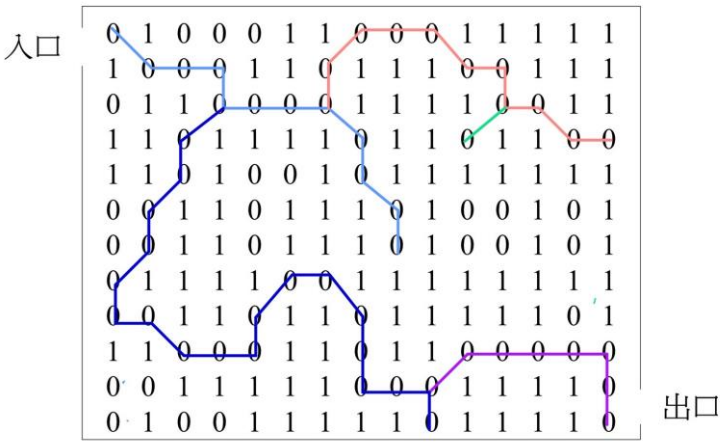
```
popped out:  3,  13, SE
popped out:  3,  12, E
pushed in:   3,  12, SW
popped out:  3,  12, SW
popped out:  2,  12, S
popped out:  2,  11, E
popped out:  1,  10, SE
popped out:  1,   9, E
popped out:  1,   8, E
```

```
popped out:  2,   7, NE
popped out:  3,   7, N
popped out:  3,   6, E
popped out:  3,   5, E
popped out:  3,   4, E
pushed in:   3,   4, SW
pushed in:   4,   3, S
pushed in:   5,   3, SW
pushed in:   6,   2, S
```

```
pushed in:   7,   2, SW
pushed in:   8,   1, S
pushed in:   9,   1, E
pushed in:   9,   2, SE
pushed in:  10,   3, E
pushed in:  10,   4, E
pushed in:  10,   5, N
pushed in:   9,   5, NE
pushed in:   8,   6, E
```

```
pushed in:   8,   7, SE
pushed in:   9,   8, S
pushed in:  10,   8, S
pushed in:  11,   8, E
pushed in:  11,   9, E
pushed in:  11,  10, S
popped out: 11,  10, S
pushed in:  11,  10, NE
pushed in:  10,  11, E
```

```
pushed in:  10,  12, E
pushed in:  10,  13, E
pushed in:  10,  14, E
pushed in:  10,  15, S
```



The attempt of the path function has 4 failure before finding way out. Human attempts has less failure because we can scrutinize the whole map and gaining insights of what to move for further step, but the Path() uses a greedy algorithm DFS to search the way out of the current state, thus it takes a lot more step.