109 學年度第二學期資訊期中專題

3D Maze Puzzle

作者：二年良班 15 號 柯尹萱

指導老師：陳怡芬老師

一、簡介

在學習 BFS 或 DFS 時，一定會遇到迷宮問題。而通常我們從解題網站上看到的都是二維的題目，也就是只有 x 軸和 y 軸。這次的專題的目的是要解決三維的迷宮問題，除了 x 軸和 y 軸以外，再加上 z 軸。

這次專題我是從 UVA Online Judge Q532：Dungeon Master 的程式碼稍作修改而完成的。Q532 的題目大致上是要求要我們從一個 3D 的迷宮裡找出一條路，且這條路必須耗時最短，換句話說，也就是最短路徑的概念。在破解迷宮這邊，我這次使用的是 BFS。我的專題是在這支程式碼上稍作修改，除了原題目要求的最短路徑，額外加上了走過的路的標示以及座標的輸出。

二、問題定義：迷宮地圖設計

由於我是從 Q532 的程式碼去做修改，因此我的迷宮設計和題目一樣，是一層一層手動輸入，並且在輸入迷宮前要先輸入 3D 迷宮的長、寬、高。範例如下（Q532 的範例輸入）：

```
3 4 5
S....
.###.
.##..
###.#

#####
#####
##.##
##...

#####
#####
#.###
####E
```

由上至下分別為第一層、第二層、第三層，紅色標示的部分是最短路徑走過的地方。一旦改變了最初輸入的長、寬、高就可以改變輸入地圖的大小。在我的程式碼中，長、寬、高的最大值都是 30。

三、資料儲存：迷宮數值表示法

在 Q532 裡，迷宮的開頭標示為「S」，終點則標示為「E」。除了起點與終點外，其他座標皆為「#」或「.」,「#」代表的是牆壁，也就是不能走的座標，「.」代表的則是可以走的部分。而最後要求的輸出是走出迷宮的最短時間，同時也是最短路徑。我的程式碼在最後輸出走過的路時，會將走過的「.」改成「*」再印出來，以方便觀察。

四、解迷宮之演算法設計與資料結構

在這支程式碼中，我開了一個 struct，目的是為了存被放到 queue 裡面的點的座標。以下是我次所使用到的全域變數。

```
constexpr int DIMENSION_SIZE = 35 ;
constexpr int QUEUE_MAX_SIZE = DIMENSION_SIZE*DIMENSION_SIZE*DIMENSION_SIZE+5 ;
struct Point{
  int l, r, c ;
};

int L, R, C, steps[DIMENSION_SIZE][DIMENSION_SIZE][DIMENSION_SIZE] ;
char maze[DIMENSION_SIZE][DIMENSION_SIZE][DIMENSION_SIZE] ;
bool visited[DIMENSION_SIZE][DIMENSION_SIZE][DIMENSION_SIZE] ;
Point queue_node[QUEUE_MAX_SIZE] ;
Point start_p, end_p ;
int q_len, q_current ;
// 0:up(l+1), 1:down(l-1), 2:front(r+1), 3:back(r-1), 4:right(c+1), 5:left(c-1),
-1:none
int from[DIMENSION_SIZE][DIMENSION_SIZE][DIMENSION_SIZE] ;
```

Point 是一個 struct，目的是為了存被放入 queue_node 裡的點的座標。L、R、C 分別為迷宮的高、長、寬，steps 則是儲存到某個座標所需的步數。Maze 存的是最初輸入的迷宮地圖，最後在輸出時會對走過的路進行修改。visited 的用意是要記錄已經走過的路，避免同一個點被走了太多次。queue_node 紀錄的是最終的答案，將最後會經過的點座標儲存起來。queue_node 有點像是一個 queue 型態的變數，在這邊是以自定義變數陣列的方式去做 queue 做的事。start_p、end_p 代表的是終點和起始點，麼做的原因是為了最後的輸出方便。q_len、q_current 則分別代表 queue_node 的長度及 queue_node 現在存到哪一格，最後，from 所記錄的是現在所在位置剛剛是從哪一個方位來的，從註解中可以看到六個方位都各自對應到一個號碼。

```
while(cin>>L>>R>>C){
    if(L==0 && R==0 && C==0) break ;
    for(int l=0 ; l<L ; l++){
      for(int r=0 ; r<R ; r++){
        for(int c=0 ; c<C ; c++){
          cin>>maze[l][r][c] ;
          if(maze[l][r][c]=='E') start_p={l, r, c} ;
          else if(maze[l][r][c]=='S') end_p={l, r, c} ;
          visited[l][r][c] = false ;
```

```
          steps[l][r][c] = -1 ;
          from[l][r][c] = -1 ;
        }
      }
    }
q_len = 0 ;
    tryPushQueue(start_p.l, start_p.r, start_p.c, 0, -1) ;
    q_current = 0 ;
    while(q_current<q_len){
      const int l=queue_node[q_current].l ;
      const int r=queue_node[q_current].r ;
      const int c=queue_node[q_current].c ;
      const int s=steps[l][r][c] ;
      tryPushQueue(l+1, r, c, s+1, 1) ;
      tryPushQueue(l-1, r, c, s+1, 0) ;
      tryPushQueue(l, r+1, c, s+1, 3) ;
      tryPushQueue(l, r-1, c, s+1, 2) ;
      tryPushQueue(l, r, c+1, s+1, 5) ;
      tryPushQueue(l, r, c-1, s+1, 4) ;
      q_current++ ;
    }
```

在主函式裡，由於 Q532 的題目要求在 L、R、C 皆等於 0 時要直接結束程式，因此在輸入完 L、R、C 後，就先判斷三者是否皆為 0，若是，則直接 break，離開 while 迴圈。若否，則以三層 for 迴圈輸入 maze，並且判斷輸入進來的是否為 S 或 E。在前一段已經提到為了輸出方便，start_p 紀錄的是 E 點座標，而 end_p 紀錄的是起點座標。q_len 紀錄的是 queue_node 的長度，因此在開始將座標放進 queue_node 前，要先將 q_len 設為 0，接著呼叫 tryPushQueue 這個函式。

我在這次磚體裡使用的是 BFS，也就是說，當我走到一個點以後，我會先對周圍所有能走的點進行判斷。在往前、後、左、右、上、下判斷前，先開 4 個變數將座標及步數存起來。呼叫這個函式的時候，必須要將點座標、步數以及從哪裡來的方位傳給它，而步數以 s+1 表示是因為到新的點步數本來就要+1，因此傳進去的時候就先加了。

```
void tryPushQueue(int l, int r, int c, int s, int f){
  if(l>=L || r>=R || c>=C || l<0 || r<0 || c<0 ||
     visited[l][r][c] || maze[l][r][c]=='#') return ;
  visited[l][r][c] = true ;
  steps[l][r][c] = s ;
  from[l][r][c] = f ;
  queue_node[q_len] = {l, r, c} ;
```

```
  q_len++ ;
}
```

tryPushQueue 是一個用來是為了要嘗試一個點可不可以被放到 queue_node 裡面的函式。在嘗試之前，得先判斷這個點是否合格。如果點座標的三個數值有任何一個大於或等於最大值（因為是從 0 開始存，所以是大於或等於）、小於 0，或是這個店已經被備走過或這個點是#的話，就直接 return。如果這個點是合格的話，就把它的 visited 改成 true，steps 改成 s，from 改成 f，並把這個點放入 queue_node 裡面，再讓 q_len+1。

```
if(!visited[end_p.l][end_p.r][end_p.c]) cout << "Trapped!" << endl ;
    else{
      int l=end_p.l, r=end_p.r, c=end_p.c ;
      cout << "Escaped in " << steps[l][r][c] << " minute(s)." << endl ;
      while(true){
        cout << "(" << c << ", " << r << ", " << l << ")" << endl ;
        if(from[l][r][c]==0) l++ ;
        else if(from[l][r][c]==1) l-- ;
        else if(from[l][r][c]==2) r++ ;
        else if(from[l][r][c]==3) r-- ;
        else if(from[l][r][c]==4) c++ ;
        else if(from[l][r][c]==5) c-- ;
        else break ;
        if(maze[l][r][c]!='E') maze[l][r][c] = '*' ;
      }
```

嘗試完以後，就先判斷終點有沒有被走過，如果沒有，就直接輸出 Trapped。有的話就先輸出最短時間，注意最短時間不包含起點，並把終點座標存起來。接著從中點藉由 from 開始往前推回起點，並輸出座標。一半來說這邊輸出的座標影該是從終點往回輸出，但因為我一開始把 start_p 和 end_p 紀錄的東西交換過來了，所以這邊的輸出會是從起點到終點的座標。再回推的過程中，由於等一下要輸出路線，因此在回推的同時，順便將走過的點從「.」改為「*」。

```
for(int l=0 ; l<L ; l++){
       for(int r=0 ; r<R ; r++){
         for(int c=0 ; c<C ; c++) cout << maze[l][r][c] ;
         cout << endl ;
       }
       cout << endl ;
     }
```

在將座標印出來後，路徑也改好了。以一個三層的 for 迴圈將改完的地圖印出來，即可看到走過的路徑分別是那些點。

五、程式實作

完整程式碼網址：https://replit.com/@d10831415/3D-Maze#main.cpp

完整程式碼：

```
#include<iostream>
using namespace std ;

constexpr int DIMENSION_SIZE = 35 ;
constexpr int QUEUE_MAX_SIZE = DIMENSION_SIZE*DIMENSION_SIZE*DIMENSION_SIZE+5 ;
struct Point{
  int l, r, c ;
};

int L, R, C, steps[DIMENSION_SIZE][DIMENSION_SIZE][DIMENSION_SIZE] ;
char maze[DIMENSION_SIZE][DIMENSION_SIZE][DIMENSION_SIZE] ;
bool visited[DIMENSION_SIZE][DIMENSION_SIZE][DIMENSION_SIZE] ;
Point queue_node[QUEUE_MAX_SIZE] ;
Point start_p, end_p ;
int q_len, q_current ;
// 0:up(l+1), 1:down(l-1), 2:front(r+1), 3:back(r-1), 4:right(c+1), 5:left(c-1),
-1:none
int from[DIMENSION_SIZE][DIMENSION_SIZE][DIMENSION_SIZE] ;

void tryPushQueue(int l, int r, int c, int s, int f){
  if(l>=L || r>=R || c>=C || l<0 || r<0 || c<0 ||
     visited[l][r][c] || maze[l][r][c]=='#') return ;
  visited[l][r][c] = true ;
  steps[l][r][c] = s ;
  from[l][r][c] = f ;
  queue_node[q_len] = {l, r, c} ;
  q_len++ ;
}

int main(){

  while(cin>>L>>R>>C){
    if(L==0 && R==0 && C==0) break ;
    for(int l=0 ; l<L ; l++){
      for(int r=0 ; r<R ; r++){
```

```
      for(int c=0 ; c<C ; c++){
        cin>>maze[l][r][c] ;
        if(maze[l][r][c]=='E') start_p={l, r, c} ;
        else if(maze[l][r][c]=='S') end_p={l, r, c} ;
        visited[l][r][c] = false ;
        steps[l][r][c] = -1 ;
        from[l][r][c] = -1 ;
      }
    }
  }
  q_len = 0 ;
  tryPushQueue(start_p.l, start_p.r, start_p.c, 0, -1) ;
  q_current = 0 ;
  while(q_current<q_len){
    const int l=queue_node[q_current].l ;
    const int r=queue_node[q_current].r ;
    const int c=queue_node[q_current].c ;
    const int s=steps[l][r][c] ;
    tryPushQueue(l+1, r, c, s+1, 1) ;
    tryPushQueue(l-1, r, c, s+1, 0) ;
    tryPushQueue(l, r+1, c, s+1, 3) ;
    tryPushQueue(l, r-1, c, s+1, 2) ;
    tryPushQueue(l, r, c+1, s+1, 5) ;
    tryPushQueue(l, r, c-1, s+1, 4) ;
    q_current++ ;
  }

  if(!visited[end_p.l][end_p.r][end_p.c]) cout << "Trapped!" << endl ;
  else{
    int l=end_p.l, r=end_p.r, c=end_p.c ;
    cout << "Escaped in " << steps[l][r][c] << " minute(s)." << endl ;
    while(true){
      cout << "(" << l << ", " << r << ", " << c << ")" << endl ;
      if(from[l][r][c]==0) l++ ;
      else if(from[l][r][c]==1) l-- ;
      else if(from[l][r][c]==2) r++ ;
      else if(from[l][r][c]==3) r-- ;
      else if(from[l][r][c]==4) c++ ;
      else if(from[l][r][c]==5) c-- ;
```

```
        else break ;
        if(maze[l][r][c]!='E') maze[l][r][c] = '*' ;
      }
      for(int l=0 ; l<L ; l++){
        for(int r=0 ; r<R ; r++){
          for(int c=0 ; c<C ; c++) cout << maze[l][r][c] ;
          cout << endl ;
        }
        cout << endl ;
      }
    }
  }
  return 0 ;
}
```

六、成果展現

1. Q532 第一筆範例測資：層數為 3、長度為 4、寬度為 5 的迷宮

```
3 4 5
S....
.###.
.##..
###.#

#####
#####
##.##
##...

#####
#####
#.###
####E
Escaped in 11 minute(s).
(0, 0, 0)
(0, 0, 1)
(0, 0, 2)
(0, 0, 3)
```

```
(0, 0, 4)
(0, 1, 4)
(0, 2, 4)
(0, 2, 3)
(0, 3, 3)
(1, 3, 3)
(1, 3, 4)
(2, 3, 4)
S****
.###*
.##**
###*#

#####
#####
##.##
##.**

#####
#####
#.###
####E
```

2. Q532 第二筆範例測資：層數為 1、寬度長度皆為 3

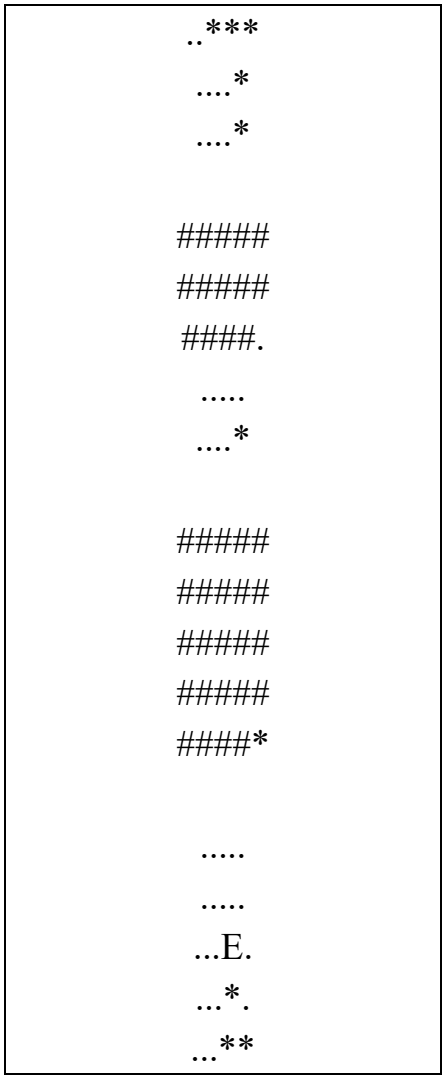
```
1 3 3
S##
#E#
###
Trapped!
```

3. 自訂測資：層數、寬度、長度皆為 5

```
5 5 5
#####
#####
##S##
#####
#####
.....
.....
```

```
.....
.....
.....
#####
#####
####.
.....
.....
#####
#####
#####
#####
####.
.....
.....
...E..
.....
.....
Escaped in 11 minute(s).
(0, 2, 2)
(1, 2, 2)
(1, 2, 3)
(1, 2, 4)
(1, 3, 4)
(1, 4, 4)
(2, 4, 4)
(3, 4, 4)
(4, 4, 4)
(4, 4, 3)
(4, 3, 3)
(4, 2, 3)
#####
#####
##S##
#####
#####

.....
.....
```



七、心得與討論

以前在解迷宮問題的時候，遇到的都是二維迷宮，而且通常都只需要輸出最短路徑的長度。這次找到 UVA Online Judge Q532 這題，讓我覺得很新奇，居然還有三維迷宮的問題。一開始以為三維迷宮會很難寫，因為我不知道要怎麼把立體圖形輸入到電腦，看到題目的範例輸入後才知道原來它是要一層一層的輸入，既然這樣就簡單多了。如果是一層一層慢慢輸入的話，基本上作的內容和二維的不會差太多，只要陣列多開一格就好，這樣差不多就已經完成大半了。

輸出的時候遇到一點點小困難，因為一開始輸出座標是從終點反過來輸出的，本來以為要重開一個 Point 存座標再跑 for 迴圈輸出，後來發現只要把 start_p 和 end_p 交換過來就可以解決問題了。

這次的專題我覺得算是偏比較簡單的，但是在座的過程中還是很有趣。因為迷宮問題有很多種解法，但我以前都是解二維的迷宮，這是第一次碰到三維的，覺得很特別。對於三維迷宮的印象一直是立方體的樣子，原本很擔心想說要如何把立方體輸入到電腦中，幸好後來問了老師才知道只要一層一層輸入就好，一層一層輸入瞬間把整個專題變得很簡單，也解決了我原來有的很多擔憂。

多虧這次專題，我才有機會回頭重新省視自己以前寫過解迷宮的程式碼，也才有機會發

現很多可以改進或優化的地方，讓這次三維迷宮專題的程式碼，能夠更加完美。

八、參考資料

1. Q532 Dungeon Master：
http://www3.tcgs.tc.edu.tw/~sagit/luckycat/q532.htm