Report

CS214-Algorithm and Complexity, Spring 2018

Name: 杨培灏　Student ID: 516021910233

问题概述: 判断连连看游戏中两个给定位置是否能够合法相连。
输入: 读取 in.dat 文件，文件格式为

$$\begin{bmatrix} x & y \end{bmatrix}$$
$$\begin{bmatrix} a_{11} & \cdots & a_{1n} \\ \vdots & \ddots & \vdots \\ a_{n1} & \cdots & a_{nn} \end{bmatrix}$$
$$\begin{bmatrix} p & q \\ m & n \end{bmatrix}$$

首行为输入规模，表示 x 行 y 列棋盘。之后 x 行 y 列矩阵为棋盘上各个位置的情况，用非负数表示棋盘各个位置的情况，0 表示没有图案。最后两行 m, n 和 p, q 为给定点坐标。默认可以超出地图。

1. **算法**

   在忽略相关规则的简单情况下，即只需要考虑是否存在连通的路径：

   (a) 直接 DFS 遍历图寻找两点间是否存在路径

   (b) 直接 BFS 遍历图寻找两点间是否存在路径

   但是连连看有特殊的规则，两点连接时所经过的路径（连接路径）不能超过两个拐点。加上这个限制条件后，改进的方法为：

   (a) 分类搜索。实现的思路是将问题分解，运用归纳法的思想，如果已经找到了起点 A 到中转点 B 的 $(n-1)$ 个拐点的路径，那么只需要找到 B 到终点 C 的直线路径，把他们接在一起就得到了从 A 到 C 的 n 个拐点的路径。又由于这里拐点数不能超过两个，所以这里的两点间连通实际上只有三种情况：

      i. 直线连接。此时 A、B 两点的横坐标 x 或纵坐标 y 是相等的。

      ii. 一个拐点。此时 A、B 两点的横坐标 x 和纵坐标 y 是不相等的。此时 A、B 分别为一个矩形的对角顶点，而矩形的两侧边缘如果存在一边是畅通的，就可以完成一个拐点的连通。

      iii. 两个拐点。根据之前的分析，两个拐点的情况可以由两个基本的情况组成，即直线连接，一个拐点连接组成。

   (b) 修改的广度优先搜索。算法的思想是如果能将所有与起点 A 点经过不多于两个拐点的路径相连的位置全部找出来，加入一个集合 S 中。由于拐点数不能超过 2，所以可以重复两步。那么判断终点 B 能否与 A 相连消除，只需要判断 B 是否属于 S 即可。

算法 1: enhencedBFS()

Input: 起点 A，终点 B，地图

initialize set $S, T = \emptyset$;
S.push(A);
for $i = 0 \to 2$ **do**
  for $v \in S$ **do**
    for $u \in v'sneighbors$ **do**
      T.push(u);
  for $u \in T$ **do**
    S.push(u);

2. 代码几个通用的功能函数

```
void showDFSPath(pair<int, int> now)
{ //path中保留的是后继
  while (now != endPoint)
  {
    cout << now.first << "," << now.second << "->";
    now = path[now.first][now.second];
  }
}
void showBFSPath(pair<int, int> now)
{ //path中保留的是前继
  if (now != pair<int, int>(0, 0))
  {
    showBFSPath(path[now.first][now.second]);
  }
  else return;
  cout << now.first << "," << now.second << "->";
}

void showPath(pathType type)
{
  if (type == BFSpath || type == enhencedBFSpath)
  {
    showBFSPath(path[endPoint.first][endPoint.second]);
    cout << endPoint.first << "," << endPoint.second << endl;
  }
  else if (type == DFSpath || classificationPath)
  {
    showDFSPath(startPoint);
    cout << endPoint.first << "," << endPoint.second << endl;
  }
}

void clear()
{
  testTimes = 0;
  for (int i = 0; i < row; ++i)
  {
    for (int j = 0; j <= col + 1; ++j)
    {
      path[i][j] = pair<int, int>(0, 0);
    }
  }
  for (int i = 0; i <= row + 1; ++i)
  {
```

```
    for (int j = 0; j <= col + 1; ++j)
    {
      visited[i][j] = false;
    }
  }
}
```

接下来是不考虑规则简化问题吼，前两个基础算法的实现：

(a)

```
bool DFS(int x, int y)
{
  testTimes++;
  visited[x][y] = true;
  if (x == endPoint.first && y == endPoint.second)
  {
    return true;
  }
  else
  {
    if (matrix[x][y] == 0)
    {
      if (x < row + 1 && !visited[x + 1][y])
      {
        if (DFS(x + 1, y))
        {
          path[x][y] = pair<int, int>(x + 1, y);
          return true;
        }
      }
      if (y < col + 1 && !visited[x][y + 1])
      {
        if (DFS(x, y + 1))
        {
          path[x][y] = pair<int, int>(x, y + 1);
          return true;
        }
      }
      if (x > 0 && !visited[x - 1][y])
      {
        if (DFS(x - 1, y))
        {
          path[x][y] = pair<int, int>(x - 1, y);
          return true;
        }
      }
      if (y > 0 && !visited[x][y - 1])
      {
        if (DFS(x, y - 1))
        {
          path[x][y] = pair<int, int>(x, y - 1);
          return true;
        }
      }
    }
    return false;
  }
}

```

```
50 bool DFS()
51 {
52   clear();
53   visited[startPoint.first][startPoint.second] = true;
54   if (startPoint.first < row + 1)
55   {
56     if (DFS(startPoint.first + 1, startPoint.second))
57     {
58     path[startPoint.first][startPoint.second] = pair<int, int>(startPoint.
       first + 1, startPoint.second);
59     return true;
60     }
61   }
62   if (startPoint.second < col + 1)
63   {
64     if (DFS(startPoint.first, startPoint.second + 1))
65     {
66       path[startPoint.first][startPoint.second] = pair<int, int>(startPoint
       .first, startPoint.second + 1);
67       return true;
68     }
69   }
70   if (startPoint.first > 0)
71   {
72     if (DFS(startPoint.first - 1, startPoint.second))
73     {
74       path[startPoint.first][startPoint.second] = pair<int, int>(startPoint
       .first - 1, startPoint.second);
75       return true;
76     }
77   }
78   if (startPoint.second > 0)
79   {
80     if (DFS(startPoint.first, startPoint.second - 1))
81     {
82       path[startPoint.first][startPoint.second] = pair<int, int>(startPoint
       .first, startPoint.second - 1);
83       return true;
84     }
85   }
86   return false;
87 }
88
```

(b)

```
1 bool BFS(int x, int y)
2 {
3   pair<int, int> child;
4   BFSqueue.push(pair<int, int>(x, y));
5   while (!BFSqueue.empty())
6   {
7     child = BFSqueue.front();
8     BFSqueue.pop();
9     visited[child.first][child.second] = true;
10    testTimes++;
11    if (child.first == endPoint.first && child.second == endPoint.second)
12    {
13      return true;
14    }
```

```
15      if (matrix[child.first][child.second] == 0)
16      {
17        if (child.first < row + 1 && !visited[child.first + 1][child.second])
18        {
19          BFSqueue.push(pair<int, int>(child.first + 1, child.second));
20          path[child.first + 1][child.second] = child;
21        }
22        if (child.second < col + 1 && !visited[child.first][child.second +
      1])
23        {
24          BFSqueue.push(pair<int, int>(child.first, child.second + 1));
25          path[child.first][child.second + 1] = child;
26        }
27        if (child.first > 0 && !visited[child.first - 1][child.second])
28        {
29          BFSqueue.push(pair<int, int>(child.first - 1, child.second));
30          path[child.first - 1][child.second] = child;
31        }
32        if (child.second > 0 && !visited[child.first][child.second - 1])
33        {
34          BFSqueue.push(pair<int, int>(child.first, child.second - 1));
35          path[child.first][child.second - 1] = child;
36        }
37      }
38    }
39    return false;
40  }
41
42  bool BFS()
43  {
44    clear();
45    visited[startPoint.first][startPoint.second] = true;
46    if (startPoint.first < row + 1)
47    {
48      if (BFS(startPoint.first + 1, startPoint.second))
49      {
50        path[startPoint.first + 1][startPoint.second] = pair<int, int>(
      startPoint.first, startPoint.second);
51        return true;
52      }
53    }
54    if (startPoint.second < col + 1)
55    {
56      if (BFS(startPoint.first, startPoint.second + 1))
57      {
58        path[startPoint.first][startPoint.second + 1] = pair<int, int>(
      startPoint.first, startPoint.second);
59        return true;
60      }
61    }
62    if (startPoint.first > 0)
63    {
64      if (BFS(startPoint.first - 1, startPoint.second))
65      {
66        path[startPoint.first - 1][startPoint.second] = pair<int, int>(
      startPoint.first, startPoint.second);
67        return true;
68      }
69    }
70    if (startPoint.second > 0)
```

```
71    {
72      if (BFS(startPoint.first, startPoint.second - 1))
73      {
74        path[startPoint.first][startPoint.second - 1] = pair<int, int>(
      startPoint.first, startPoint.second);
75        return true;
76      }
77    }
78    return false;
79  }
```

在了解规则后，完整的代码如下：

(a)

```
1  bool enhencedBFS()
2  { //使用两个临时的集合进行遍历，确定可以到达后存储进稳定的集合
3    clear();
4    visited[startPoint.first][startPoint.second] = true;
5    templinkedPoints.insert(pair<int, int>(startPoint.first, startPoint.
       second));
6    for (int loopTimes = 0; loopTimes < 3; ++loopTimes)
7    {
8      for (set<pair<int, int>>::iterator it = templinkedPoints.begin(); it !=
       templinkedPoints.end(); ++it)
9      {
10       int i = 1;
11       while ((*it).first + i < row + 1 && !visited[(*it).first + i][(*it).
       second] && matrix[(*it).first + i][(*it).second] == 0)
12       {
13         tempStorePoints.insert(pair<int, int>((*it).first + i, (*it).second
       ));
14         path[(*it).first + i][(*it).second] = *it;
15         visited[(*it).first + i][(*it).second] = true;
16         ++i;
17         ++testTimes;
18       }
19       //上方while遍历某个方向上的通路，停止后遇到的第一个阻塞无法加入临时集
       合，应该加入稳定集合
20       if ((*it).first + i < row + 1 && matrix[(*it).first + i][(*it).second
       ] != 0)
21       {
22         linkedPoints.insert(pair<int, int>((*it).first + i, (*it).second));
23         path[(*it).first + i][(*it).second] = *it;
24         visited[(*it).first + i][(*it).second] = true;
25       }
26
27       i = 1;
28       while ((*it).second + i < col + 1 && !visited[(*it).first][(*it).
       second + i] && matrix[(*it).first][(*it).second + i] == 0)
29       {
30         tempStorePoints.insert(pair<int, int>((*it).first, (*it).second + i
       ));
31         path[(*it).first][(*it).second + i] = *it;
32         visited[(*it).first][(*it).second + i] = true;
33         ++i;
34         ++testTimes;
35       }
36       if ((*it).second + i < col + 1 && matrix[(*it).first][(*it).second +
       i] != 0)
```

```
37        {
38          linkedPoints.insert(pair<int, int>((*it).first, (*it).second + i));
39          path[(*it).first][(*it).second + i] = *it;
40          visited[(*it).first][(*it).second + i] = true;
41        }
42
43        i = 1;
44        while ((*it).first - i > 0 && !visited[(*it).first - i][(*it).second]
      && matrix[(*it).first - i][(*it).second] == 0)
45        {
46          tempStorePoints.insert(pair<int, int>((*it).first - i, (*it).second
      ));
47          path[(*it).first - i][(*it).second] = *it;
48          visited[(*it).first - i][(*it).second] = true;
49          ++i;
50          ++testTimes;
51        }
52        if ((*it).first - i > 0 && matrix[(*it).first - i][(*it).second] !=
      0)
53        {
54          linkedPoints.insert(pair<int, int>((*it).first - i, (*it).second));
55          path[(*it).first - i][(*it).second] = *it;
56          visited[(*it).first - i][(*it).second] = true;
57        }
58
59        i = 1;
60        while ((*it).second - i > 0 && !visited[(*it).first][(*it).second - i
      ] && matrix[(*it).first][(*it).second - i] == 0)
61        {
62          tempStorePoints.insert(pair<int, int>((*it).first, (*it).second - i
      ));
63          path[(*it).first][(*it).second - i] = *it;
64          visited[(*it).first][(*it).second - i] = true;
65          ++i;
66          ++testTimes;
67        }
68        if ((*it).second - i > 0 && matrix[(*it).first][(*it).second - i] !=
      0)
69        {
70          linkedPoints.insert(pair<int, int>((*it).first, (*it).second - i));
71          path[(*it).first][(*it).second - i] = *it;
72          visited[(*it).first][(*it).second - i] = true;
73        }
74      }
75      templinkedPoints.clear();
76      for (set<pair<int, int>>::iterator it = tempStorePoints.begin(); it !=
      tempStorePoints.end(); ++it)
77      {
78        templinkedPoints.insert(*it);
79      }
80      tempStorePoints.clear();
81      for (set<pair<int, int>>::iterator it = templinkedPoints.begin(); it !=
       templinkedPoints.end(); ++it)
82      {
83        linkedPoints.insert(*it);
84      }
85      if (linkedPoints.find(endPoint) != linkedPoints.end())
86      {
87        path[startPoint.first][startPoint.second] = pair<int, int>(0, 0);
88        testTimes = linkedPoints.size();
```

```
89          return true;
90        }
91      }
92      testTimes = linkedPoints.size();
93      path[startPoint.first][startPoint.second] = pair<int, int>(0, 0);
94      return linkedPoints.find(endPoint) != linkedPoints.end();
95  }
96
```

(b)

```
1   bool straightLinked(pair<int, int> startNow, pair<int, int> endNow)
2   { //直接相连
3     testTimes++;
4     if (startNow.first == endNow.first)
5     {
6       for (int i = min(startNow.second, endNow.second) + 1; i <= max(startNow
        .second, endNow.second) - 1; ++i)
7       {
8         if (matrix[startNow.first][i] != 0) return false;
9       }
10      path[startNow.first][startNow.second] = endNow;
11      return true;
12    }
13    else if (startNow.second == endNow.second)
14    {
15      for (int i = min(startNow.first, endNow.first) + 1; i <= max(startNow.
        first, endNow.first) - 1; ++i)
16      {
17        if (matrix[i][startNow.second] != 0) return false;
18      }
19      path[startNow.first][startNow.second] = endNow;
20      return true;
21    }
22    else return false;
23  }
24
25  bool oneTurnLinked(pair<int, int> startNow, pair<int, int> endNow)
26  { //一拐点相连
27    testTimes++;
28    if (startNow.first != endNow.first && startNow.second != endNow.second)
29    {
30      if (matrix[startNow.first][endNow.second] == 0)
31      {
32        if (straightLinked(startNow, pair<int, int>(startNow.first, endNow.
          second)) && straightLinked(pair<int, int>(startNow.first, endNow.second
          ), endNow))
33        {
34          return true;
35        }
36      }
37      if (matrix[endNow.first][startNow.second] == 0)
38      {
39        if (straightLinked(startNow, pair<int, int>(endNow.first, startNow.
          second)) && straightLinked(pair<int, int>(endNow.first, startNow.second
          ), endNow))
40        {
41          return true;
42        }
43      }
```

```
    }
    return false;
}
bool twoTurnLinked(pair<int, int> startNow, pair<int, int> endNow)
{ //两拐点相连
    testTimes++;
    if (startNow.first == endNow.first)
    {
        for (int i = 0; i <= row + 1; ++i)
        {
            if (i == startNow.first) continue;
            if (straightLinked(startNow, pair<int, int>(i, startNow.second)) &&
     oneTurnLinked(pair<int, int>(i, startNow.second), endNow))
            {
                return true;
            }
        }
    }
    else if (startNow.second == endNow.second)
    {
        for (int i = 0; i <= col + 1; ++i)
        {
            if (i == startNow.second)continue;
            if (straightLinked(startNow, pair<int, int>(startNow.first, i)) &&
     oneTurnLinked(pair<int, int>(startNow.first, i), endNow))
            {
                return true;
            }
        }
    }
    return false;
}

bool classification()
{
    clear();
    visited[startPoint.first][startPoint.second] = true;
    return straightLinked(startPoint, endPoint) || oneTurnLinked(startPoint,
     endPoint) || twoTurnLinked(startPoint, endPoint);
}
```

3. 测试

测试文件如下

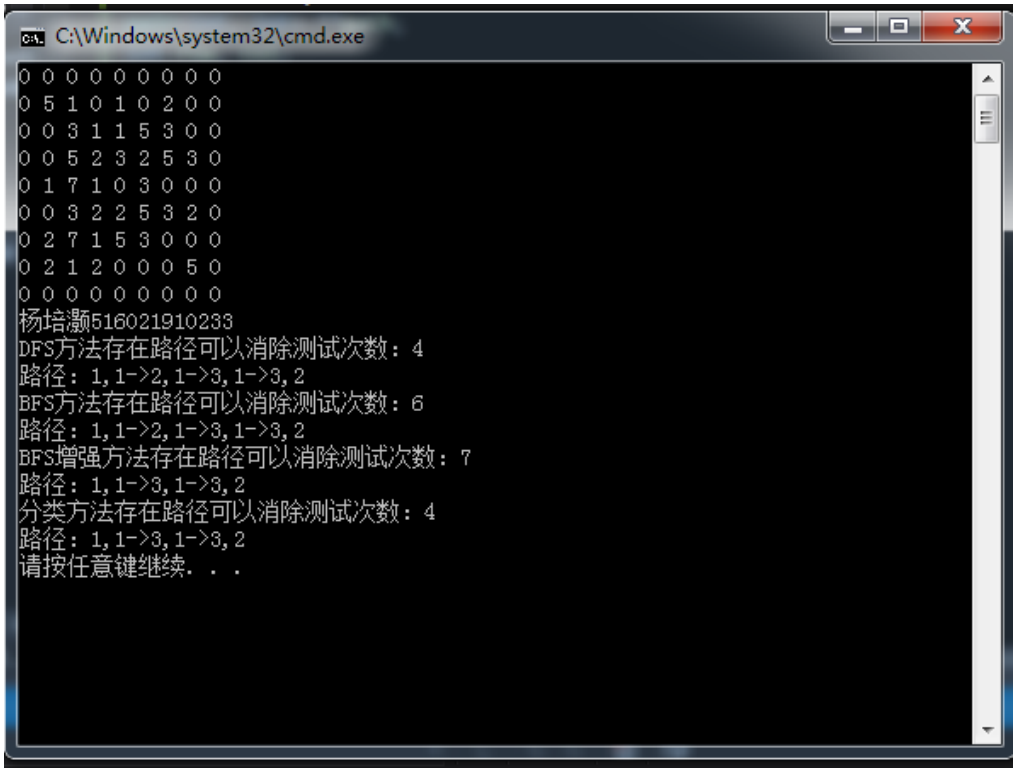
```
#include <iostream>
#include <string>
#include <ctime>
#include <queue>
#include <set>
#include <utility>
#include <algorithm>
#pragma warning(disable : 4996)
using namespace std;

enum pathType
{
    DFSpath,
    BFSpath,
```

```
15    enhencedBFSpath,
16    classificationPath
17  };
18
19  const string answer[2] = { "不存在路径，无法消除", "存在路径可以消除" };
20  pair<int, int> startPoint, endPoint;
21  int **matrix;
22  bool **visited;
23  pair<int, int> **path;
24  int row, col;
25  int testTimes;
26  queue<pair<int, int>> BFSqueue;
27  set<pair<int, int>> linkedPoints, tempStorePoints, templinkedPoints;
28  int main()
29  {
30    bool answerIndex = false;
31    freopen("test.txt", "r", stdin);
32    cin >> row >> col;
33    matrix = new int *[row + 2];
34    for (int i = 0; i < row + 2; ++i)
35    {
36      matrix[i] = new int[col + 2];
37    }
38    visited = new bool *[row + 2];
39    for (int i = 0; i < row + 2; ++i)
40    {
41      visited[i] = new bool[col + 2];
42    }
43    path = new pair<int, int> *[row + 2];
44    for (int i = 0; i < row + 2; ++i)
45    {
46      path[i] = new pair<int, int>[col + 2];
47    }
48    for (int i = 0; i < row + 2; ++i)
49    {
50      matrix[i][0] = 0;
51      matrix[i][col + 1] = 0;
52    }
53    for (int i = 0; i < col + 2; ++i)
54    {
55      matrix[0][i] = 0;
56      matrix[row + 1][i] = 0;
57    }
58    for (int i = 1; i <= row; ++i)
59    {
60      for (int j = 1; j <= col; ++j)
61      {
62        cin >> matrix[i][j];
63      }
64    }
65    for (int i = 0; i <= row + 1; ++i)
66    {
67      for (int j = 0; j <= col + 1; ++j)
68      {
69        cout << matrix[i][j] << " ";
70      }
71      cout << endl;
72    }
73    cin >> startPoint.first >> startPoint.second >> endPoint.first >> endPoint.
       second;
```

```
74    cout << "杨培灏516021910233" << endl;
75    if (matrix[startPoint.first][startPoint.second] != matrix[endPoint.first][
        endPoint.second])
76    {
77      cout << "图形不同不可能消除" << endl;
78    }
79    else
80    {
81      answerIndex = DFS();
82      cout << "DFS方法" << answer[answerIndex];
83      if (answerIndex)
84      {
85        cout << "测试次数: " << testTimes << endl << "路径: ";
86        showPath(DFSpath);
87      }
88      answerIndex = BFS();
89      cout << "BFS方法" << answer[answerIndex];
90      if (answerIndex)
91      {
92        cout << "测试次数: " << testTimes << endl << "路径: ";
93        showPath(BFSpath);
94      }
95      answerIndex = enhencedBFS();
96      cout << "BFS增强方法" << answer[answerIndex];
97      if (answerIndex)
98      {
99        cout << "测试次数: " << testTimes << endl << "路径: ";
100       showPath(enhencedBFSpath);
101     }
102     answerIndex = classification();
103     cout << "分类方法" << answer[answerIndex];
104     if (answerIndex)
105     {
106       cout << "测试次数: " << testTimes << endl << "路径: ";
107       showPath(classificationPath);
108     }
109   }
110   return 0;
111 }
112
```

地图输入见下图，为 7×7 的地图，处理时补充成了 9×9 的地图。测试点为 $(1,1)$ 和 $(3,1)$。测试结果输出如下：



4. **讨论**

   在不考虑规则的简单情况下，直接遍历寻找连通路径即可。最坏情况遍历整个地图矩阵，时间复杂度为 $\mathcal{O}(|V|^2)$。之后的加入游戏规则的算法，最坏情况也会遍历所有的点，时间复杂度和解决简单之前问题相同，为 $\mathcal{O}(|V|^2)$。