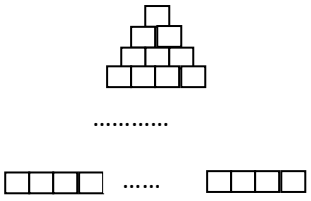
设计一个算法，生成如下的三角形的迷宫，并允许用户指定迷宫的入口和出口



【解】与程序设计题 1 类似，我们还是用并查集来实现。对所有的单元按层编号。最顶上的是 0 号单元。第二层第一个单元是 1 号单元，以此类推。一个 n 层的三角形迷宫一共有 n *（n + 1）/ 2 个单元。与程序设计题 1 的一个不同之处是迷宫的入口和出口可以由用户指定。用户可以指定第几行的第几个单元是入口或出口。但要注意，入口和出口一定要是边上的单元，不能是中间的单元。三角形迷宫的生成过程与矩形迷宫类似。不断地选择一堵尚未敲开的墙，如果这堵墙分割的两个单元是不连通的，则敲开这堵墙，使之连通。重复这个过程，直到入口和出口连通。

三角形迷宫的生成函数见代码清单 11-6。生成一个三角形迷宫需要给出迷宫的层数、入口位置和出口位置。参数 n 表示迷宫的层数，startRow 和 startCol 是迷宫的入口位置，endRow 和 endCol 是出口的位置。函数首先检查入口和出口的位置是否正确，这是通过调用 check 函数完成的。将迷宫的层数和入口或出口的位置传给 check 函数，该函数返回一个 bool 类型的值。如果位置是可行的，返回 true，否则返回 false。check 函数的定义见代码清单 11-7。如果入口和出口的位置是正确的，接着为生成迷宫做准备。先计算所需的并查集的规模以及入口和出口的编号，然后定义一个并查集。

接下去的处理过程和程序设计题 1 基本类似。不断地选择一堵合适的墙，敲墙，使单元连通，直到连通了入口和出口。敲墙时，首先随机选择一个单元，然后检查相邻的单元。如果这两个单元没有连通，则敲墙，否则检查其他相邻单元。但找相邻单元的工作比矩形的迷宫要复杂。在矩形迷宫中，相邻的单元就是上、下、左、右四个单元。但在三角形迷宫中，上面可能有两个单元也可能只有一个单元，除了最下面一层外下面都有两个单元。所以找相邻单元时必须检查 6 个单元。

代码清单 11-6 生成三角形迷宫的函数

```
1.  void createPuzzle(int n, int startRow, int startCol, int endRow, int endCol)
2.  {   if (!check(n,startRow, startCol) || !check(n, endRow, endCol)) {
3.          cout << "非法的入口或出口" << endl;
4.          return;
5.      }
6.
7.      // 计算并查集的规模以及起始和终止位置的编号
8.    int size = n * (n + 1)/2, row, col, pos1, pos2;   // size：通道数
9.      int start = startRow * (startRow - 1) / 2 + startCol - 1; // 入口编号
10.     int end = endRow * (endRow - 1) / 2 + endCol - 1;      // 出口编号
11.
```

```
12.    DisjointSet ds(size);            // 定义表示迷宫的并查集
13.      srand(time(NULL));
14.    while ( ds.Find(start) != ds.Find(end) ) {//当入口和出口不在一个等价类中
15.            while (true) {                    // 选择一堵没有敲过的墙
16.                  row = rand() % n;              // 随机选择一个单元
17.                  col = rand() % (row+1);
18.                  pos1 = row * (row + 1) / 2 + col;  // 计算该单元的编号
19.                  if (row != 0) {         // 不是第一行的单元，检查上面的相邻单元
20.                        if (col > 0) {     // 不是最左边的单元，检查左上角
21.                              pos2 = row * (row - 1) / 2 + col - 1;
22.                           if (ds.Find(pos1) != ds.Find(pos2))  break;
23.                        }
24.                        if (col < row) {    // 不是最右边的单元，检查右上角
25.                           pos2 = row * (row - 1) / 2 + col;
26.                           if (ds.Find(pos1) != ds.Find(pos2))  break;
27.                        }
28.                  }
29.                  if (col > 0) {         // 不是最左边的单元，检查左边的相邻单元
30.                        pos2 = pos1 - 1;
31.                     if (ds.Find(pos1) != ds.Find(pos2)) break;
32.                  }
33.                  if (col < row) {     // 不是最右边的单元，检查右边的相邻单元
34.                        pos2 = pos1 + 1;
35.                     if (ds.Find(pos1) != ds.Find(pos2)) break;
36.                  }
37.                  if (row < n - 1) {   // 不是最后一行，检查下面的相邻单元
38.                        pos2 = (row + 1) * (row + 2) / 2 + col;
39.                        if (ds.Find(pos1) != ds.Find(pos2)) break;
40.                        ++pos2;
41.                        if (ds.Find(pos1) != ds.Find(pos2)) break;
42.                  }
43.            }
44.            ds.Union(ds.Find(pos1), ds.Find(pos2));          //敲墙
45.            cout << "<" << pos1 << ", " << pos2 << ">";
46.       }
47.    cout << endl;
48.  }
```

代码清单 11-7 给出了 check 函数的实现。一个合法的入口和出口必须位于三角形的边缘。也就是说，第一行和最后一行的单元或者是中间那些行的第一个或最后一个单元。代码清单 11-7 的第 2 行检查了输入非法的情况，即用户指定的入口或出口超出了迷宫的范围，返回 false。第 3 行检查合法的情况，即入口或出口是否位于第一列、最后一列或者最后一行，返回 true。其他的都是非法情况，返回 false。

代码清单 11-7 check 函数的实现

```
49.  bool check(int n, int row, int col)
50.  {     if (row > n || col > n) return false;
51.        if (col == 1 || row == col || row == n) return true;
52.        return false;
53.  }
```