广度优先搜索与深度优先搜索

邓岩*

1604210502

2018 年 11 月 13 日

*计算机 11605

1 算法描述

1.1 广度优先搜索

广度优先搜索是一种盲目搜寻法，目的是系统地展开并检查图中的所有节点，以找寻结果。换句话说，它并不考虑结果的可能位置，彻底地搜索整张图，直到找到结果为止。

从算法的观点，所有因展开节点而得到的子节点都会被加进一个先进先出的队列中。一般的实现里，其邻居节点尚未被检验过的节点会被放置在一个被称为 open 的容器中（例如队列或是链表），而被检验过的节点则被放置在被称为 closed 的容器中。

1.2 深度优先搜索

深度优先搜索使用的策略就像是其名字就像其名字所隐含的: 只要可能，就在途中尽量深入。深度优先搜索总是对最近才发现的结点 v 的出发边进行探索，直到该节点的所有出发边都被发现为止。一旦节点 v 的所有出发边都被发现，搜索则“回溯”到 v 的前驱节点 (v 是经过该节点才被发现的)，来搜索该前驱节点的出发边。该过程一直持续到从源节点可以达到从源节点可以达到的所有节点都被发现为止。如果还存在尚未发现的节点，则深度优先搜索将从这些未被发现的结点中任选一个作为新的源节点，并重复同样的搜索过程。该算法重复整个过程，直到图中的所有节点都被发现为止。

2 程序

2.1 广度优先搜索

```
def BFS(maze):
    num = 2
    queue = [(0,0)]
    recall = maze.a.clone()
    recall[0, 0] = num
    while (len(queue)):
        cur = queue.pop()
        num = recall[cur[0], cur[1]] + 1
        for i, j in [(0, 1),(1, 0),(-1, 0),(0, -1)]:
            x = cur[0] + i
            y = cur[1] + j
            if (x >= 0 and x < maze.a.numRows() and y >= 0 and
                    y < maze.a.numCols() and recall[x, y] == 1):
                queue.insert(0, (x, y))
                recall[x, y] = num
    return recall
```

2.2 深度优先搜索

```
def DFS(maze):
    num = 2
    queue = [(0, 0)]
    recall = maze.a.clone()
    recall[0, 0] = num
    while (len(queue)):
        cur = queue.pop()
        num = recall[cur[0], cur[1]] + 1
        for i, j in [(0, 1), (1, 0), (-1, 0), (0, -1)]:
            x = cur[0] + i
            y = cur[1] + j
            if (x >= 0 and x < maze.a.numRows() and y >= 0 and
                    y < maze.a.numCols() and recall[x, y] == 1):
                queue.append((x, y))
                recall[x, y] = num
    return recall
```

2.3 主程序

```
#
# Created by dengyan on 2018/11/12.
#
from myarray2d import *
from graphics import *

class Maze:
    def __init__(self):
        self.a = Array2D(20, 20)
        for i in range(20):
            for j in range(20):
                self.a[i, j] = 1
            self.a[i, round(random.random() * 20) % 20] = 0
            self.a[i, round(random.random() * 20) % 20] = 0
            self.a[i, round(random.random() * 20) % 20] = 0
            self.a[i, round(random.random() * 20) % 20] = 0
            self.a[i, round(random.random() * 20) % 20] = 0
        self.a[0, 0] = 1
        self.a[19, 19] = 1

    def draw(self):
        self.win = GraphWin("maze", 400, 400)
        self.win.setCoords(0, 0, 400, 400)
        for i in range(20):
            for j in range(20):
                if (self.a[i, j] == 0):
                    p2 = Rectangle(Point(j * 20, i * 20),
                        Point(j * 20 + 20, i * 20 + 20))
                    p2.setFill("red")
                    p2.setOutline("black")
                    p2.draw(self.win)
                else:
                    p1 = Rectangle(Point(j * 20, i * 20),
                        Point(j * 20 + 20, i * 20 + 20))
                    p1.setFill("green")
                    p1.setOutline("black")
```

```
                    p1.draw(self.win)
        return self.win

    def close(self):
        self.win.close()

    def print(self):
        for i in range(self.a.numRows()):
            for j in range(self.a.numCols()):
                print(self.a[i, j],end='')
            print()

def path_maze(recall, start, end, win):
    if start == end:
        print(end)
    elif recall[end[0],end[1]] == 0:
        print("can't pass")
    else:
        for i, j in [(0, 1),(1, 0),(-1, 0),(0, -1)]:
            x = end[0] + i
            y = end[1] + j
            if (x >= 0 and x <= 19 and y >= 0 and y <= 19):
                if recall[x, y] == recall[end[0], end[1]] - 1:
                    line = Line(Point(y * 20 + 10, x * 20 + 10),
                        Point(end[1] * 20 + 10, end[0] * 20 + 10))
                    line.setArrow("last")
                    line.setOutline("white")
                    line.draw(win)
                    break
        path_maze(recall, start, (x, y), win)
        print(end)



def BFS(maze):
    num = 2
    queue = [(0,0)]
```

```
74      recall = maze.a.clone()
75      recall[0, 0] = num
76      while (len(queue)):
77          cur = queue.pop()
78          num = recall[cur[0], cur[1]] + 1
79          for i, j in [(0, 1),(1, 0),(-1, 0),(0, -1)]:
80              x = cur[0] + i
81              y = cur[1] + j
82              if (x >= 0 and x < maze.a.numRows() and y >= 0 and
83                      y < maze.a.numCols() and recall[x, y] == 1):
84                  queue.insert(0, (x, y))
85                  recall[x, y] = num
86      return recall
87
88  def DFS(maze):
89      num = 2
90      queue = [(0, 0)]
91      recall = maze.a.clone()
92      recall[0, 0] = num
93      while (len(queue)):
94          cur = queue.pop()
95          num = recall[cur[0], cur[1]] + 1
96          for i, j in [(0, 1), (1, 0), (-1, 0), (0, -1)]:
97              x = cur[0] + i
98              y = cur[1] + j
99              if (x >= 0 and x < maze.a.numRows() and y >= 0 and
100                     y < maze.a.numCols() and recall[x, y] == 1):
101                 queue.append((x, y))
102                 recall[x, y] = num
103     return recall
104
105 def main():
106     a = Maze()
107     a.draw()
108     c = input(" 广度优先请按 1, 否则深度优先:\n")
109     if (c == '1'):
110         recall = BFS(a)
```

```
111         else:
112             recall = DFS(a)
113         path_maze(recall, (0, 0), (19, 19), a.win)
114         input("Please enter any key to exit")
115         a.close()
116 
117 if __name__ == '__main__':
118         main()
```

3 应用与拓展

3.1 广度优先搜索

- 最短路径
- 最小生成树

3.2 深度优先搜索

- 拓扑排序
- 有向图分解为强连通分量

参考文献

[1] Thomas, H.Cormen, Charles, E.Leiserson, Ronald, L.Rivest, Clifford, Stein, 殷建平, and 徐云. *算法导论 (原书第 3 版)*. 机械工业出版社, third edition, 2013.