2019-2020 学年第一学期算法设计与分析

上机实验报告

实验名称：跳马问题

学 号	秦敏浩	姓 名	17122490	评 分	
专 业	计算机科学与技术	实验类型	综合	任课教师	岳晓冬
完成日期	2019.10.17	实验学时	2		

一、实验问题描述

1、实验内容

跳马问题：给定 8*8 方格棋盘，求棋盘上一只马从一个位置到达另一位置的最短路径长。

2、实验目的

- 分析问题，说明原理和方法
- 正确实现跳马问题
- 探讨使用 DFS 与 BFS 求解此题的效率和可能的问题

3、实验环境

操作系统：Linux/Windows

编译环境：GNU G++14

二、算法设计与分析

1、算法基本思想

考虑到每一个位置最多有 8 个可能的转移。从起点出发，遍历每一种转移，若出现已经被走过的情况，则不进行尝试；否则，继续尝试下一步，直至走到目的地为止。

该题本质是一个 64 节点的无向图求任一两点之间的最短路问题，可以使用弗洛伊德算法一次性解决，或使用 Dijkstra 算法。同理，也可以使用深度优先搜索或广度优先搜索的方法解决。无论表现形式如何，它们的本质都是在寻找图上最短路。

2、算法设计与求解步骤

vis：当前该节点被走到过的最短时间，若未访问为-1

d：常量，记录马可以走的八个转移方向

que：目前仍然可以更新的节点组成的队列

- 将起点放入队列，标记起点 vis 为 0
- 每次取出队列中第一个元素，如果是目标点，输出 vis，结束
- 遍历该节点的八个转移，如有合法未访问的转移，将其加入队列
- 重复上述过程直至队列为空或走到目标点为止

3、算法分析

考虑到每一个节点最多会进入队列一次，最多被其他节点访问八次，对于一个 n*n 的棋盘，采用 BFS 算法求解时间复杂度为每次询问 $O(n^2)$ 。由于采用 DFS 算法可能会导致一个节点被多次访问，只能在当前访问时间大于 vis 时进行剪枝，代码效率更低，时间复杂度可能是指数级的。

4、算法程序实现

篇幅原因，见附录或附件。

三、调试与运行

输入：

```
a1 a2
a1 a3
a1 h8
g2 b8
```

输出：

```
a1 a2
a0==>a1: 3 moves
a1 a3
a0==>a2: 2 moves
a1 h8
a0==>h7: 6 moves
g2 b8
g1==>b7: 5 moves
```

四、结果分析

本题做法多样，再此分析各算法的优点和缺点。

弗洛伊德：复杂度$O\left(n^6+q\right)$，可以直接求出任意两点之间需要最少移动的步数，和询问次数是并行关系。

Dijkstra：复杂度$O\left(q\cdot n^2\log n\right)$，考虑到**原图稀疏**，边数约为 4 倍点数，故 Dijkstra 是一种可以考虑的解法。常数较大。

BFS：复杂度$O\left(q\cdot n^2\right)$，证明如上。

DFS：复杂度为指数级，原因如上。

五、本次实验的收获、心得体会

本次实验是一道我也不知道为什么会出现在这里的题目。解法的思路十分多样，对于输入规模的数据大小，无论采用何种算法应该都是可以容忍的（DFS 除外）。

由于接触 BFS 算法较早已经习以为常，加之时间紧张，没有给出有关 BFS 算法合理性的证明。BFS 算法的本质是需要保证队列里的 vis 值是单调不减的，不然就会产生答案错误的问题。当然，这也是为什么 Dijkstra 不能跑负环的根本原因。

此外，此题可能还有复杂度更加优秀的做法。例如当棋盘规模达到 10000*10000 时，即使使用 BFS 可能也不可容忍。可以采用一定的贪心地策略去保证答案正确的情况下将问题划归到一个很小的规模（如目的地周围 80*80 的大小）去求解，可以得到更好的结果。

附录：代码实现（也可见附件）

```cpp
#include <cstring>
#include <cstdio>
#include <queue>
using namespace std;

int vis[10][10]; // minimal time to get there (-1: unvisited)
int d[8][2]={{2,1},{2,-1},{-2,1},{-2,-1},{1,2},{1,-2},{-1,2},{-1,-2}}; // 8 directions

inline bool can_vis(int x,int y){ // check if the chess is out of the board
    if (x<0||x>=8) return false;
    if (y<0||y>=8) return false;
    return true;
```

```
}

int main(){
    int x1,x2,y1,y2,curx,cury,nxtx,nxty;
    char in[5];
    while (fgets(in,10,stdin)){ // scan until it reaches '\n'
        x1=in[0]-'a'; // transfer to number
        x2=in[1]-'1';
        y1=in[3]-'a';
        y2=in[4]-'1';
        memset(vis,-1,sizeof(vis)); // set all the places unvisited
        queue<pair<int,int>> que; // a queue for BFS
        que.emplace(x1,x2);
        vis[x1][x2]=0;
        while (!que.empty()){
            curx=que.front().first; // current position x
            cury=que.front().second; // current position y
            que.pop();
            if (curx==y1&&cury==y2){ // already at the destination
                printf("%c%d==>%c%d: %d moves\n",'a'+x1,x2,'a'+y1,y2,vis[curx][cury]);
                break;
            }
            for (int i=0;i<8;++i){ // try to get a next step
                nxtx=curx+d[i][0]; // next position x
                nxty=cury+d[i][1]; //next position y
                if (!can_vis(nxtx,nxty)) continue; // out of the board
                if (vis[nxtx][nxty]!=-1) continue; // already visited
                vis[nxtx][nxty]=vis[curx][cury]+1;
                que.emplace(nxtx,nxty);
            }
        }
    }
    return 0;
}
```