1. Maze router (迷宫布线器)

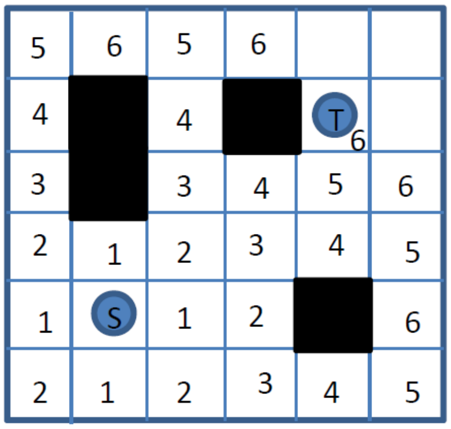
Nowadays, a multi-core CPU (多核 CPU) in our computers can have more than 1 billion transistors (晶体管). These transistors are integrated (集成) and interconnected (互连) in the so-called integrated circuits (集成电路) for the correct functionality. How are these transistors interconnected? There is a research area called EDA (Electronic Design Automation, 电子设计自动化) or VLSI CAD (Very Large Scale Integration Computer-Aided Design, 集成电路辅助设计), where there are many interesting and/or boring (也许有趣/也许无聊，在己不在人) optimization problems and algorithms. For the above-mentioned interconnection problem, many routing algorithms (布线算法) have been presented. To get a little bit flavor of how the routing is done, we will try the basic routing algorithm called Lee's algorithm (李氏算法) or maze (maze-running) algorithm (迷宫算法).

Maze algorithm is a classic algorithm for finding a shortest path between two terminals (or points) on a grid with obstacles (障碍). This algorithm typically consists of two stages: searching stage (搜索阶段) and backtrace stage (回溯阶段). Mark one terminal as the source terminal and the other one as target terminal. During the searching stage, we perform a breadth-first search (BFS)[1] (宽度优先搜索) from the source terminal to all the adjacent grid points and label all visited grid points until reaching target terminal. The label of a point indicates its Manhattan distance[2] (曼哈顿距离) from the source. Please note that routing paths (布线路径) generally run in rectilinear

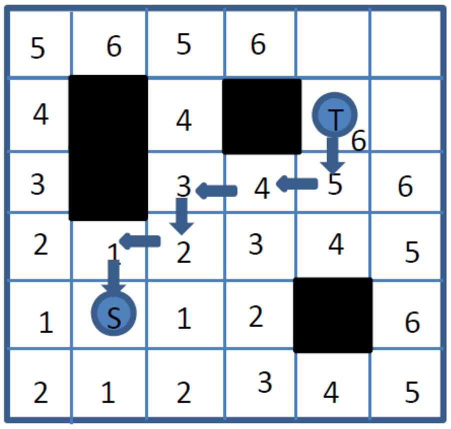
[1] In graph theory, breadth-first search is a graph search algorithm that begins at the root node and explores all the neighboring nodes. Then for each of those nearest nodes, it explores their unexplored neighbor nodes, and so on, until it finds the goal.

[2] Manhattan distance gives a metric in which the distance between two points is the sum of the absolute differences of their coordinates. For example, given two points p1 (x1,y1) and p2 (x2,y2), their Manhattan distance is d = |x1-x2|+|y1-y2|.

directions in VLSI designs (集成电路设计). When the target terminal is reached, we start the backtrace stage, which finds one path from target to source according to the labels.



(a) Searching process.



(b) Backtrace process.

Figure 1. Maze algorithm.

Figure 1 gives an example of maze algorithm including the searching and backtrace processes. During the searching process, grids are labeled with increasing values on the wavefront (波前) of the flood-fill (洪水填充) at source terminal S until reaching target terminal T. Then during backtrace process we can find a path back from the target terminal to the source terminal.

QUESTION:

(1) In Figure 1(b), we can see that there are many paths of Manhattan distance 6, but a path with 4 turns[3] is computed. Can we find a path of distance 6 (i.e., shortest distance) and with the smallest number of turns? You may enumerate all the paths of distance 6 and find one with smallest

[3] When there is a change in direction of the path, we say there is a turn. In Figure 1(b), the 4 turns are at labels 1, 2, 3, and 5 respectively.

number of turns. You may also try some optimization strategies for speedup if you think the enumeration method (枚举法) runs too slow, e.g., backtrack (回溯) or branch and bound (分支定界) strategies.

(2) Please provide two versions of the source code: (a) provide all the source files along with the makefile, and (b) assume the BaseRouter was implemented by others, provide a library file (maze.a) and the header file (BaseRouter.h), the source files of OptRouter and makefile.

TIPS:

(1) The source codes are provided where the BaseRouter class implements the basic maze searching ("bool mazeSearch();") and backtrace ("void mazeBacktrace();") stages. You should not modify BaseRouter (BaseRouter.h and BaseRouter.cxx).

(2) Actually, the source file of BaseRouter can be provided in library format (Please check the attached makefile to see how to create and link a static library "maze.a", and memorize it) such that you can never see its implementation[4]. You need to implement a derived class OptRouter from base class BaseRouter. Then reuse function mazeSearch() of BaseRouter for the searching process. And write a new function mazeBacktrace() to override the one in base class. The new function mazeBacktrace() should be able to find a shortest path with minimum number of turns.

(3) Please use our official CodeBlocks compiler to generate the library so that others will be able to compile your provided library for homework review.

[4] This is what companies often do for copyright protection. You can use their libraries, but you can never see their implementation details. Please note that even under such circumstances, you can still reuse the codes by inheritance.

2. **Nothing is impossible in the programming world!** The following piece of magic code may crash and/or have different outputs on different systems (e.g., Mac OS, Linux, Windows, etc.). Actually, it is not magic, but a BUG! Catch the bugs and fix them.

```
#include <iostream> // cout, endl
#include <cstring> // strlen , strcpy
using namespace std;
class bug {
    char* data_;
public:
    bug(const char *str) {
        data_ = new char[strlen(str)+1];
        strcpy(data_, str);
    }
    ~bug() {
        delete data_;
    }
    void show() {
        cout << data_ << endl;
    }
};

void test(bug str1) {
    str1.show();
    bug str2("tsinghua");
    str2.show();
    str2 = str1;
    str2.show();
}

int main () {
    bug str1 ("2011");
    str1.show();
    test(str1);
    str1.show();
    return 0;
}
```