立方数码说明文档

PB14011009 邹易澄

操作说明

本目录下有4个cpp文件、4个Windows系统的exe可执行文件、4个linux系统的可执行文件、本说明文档。

Windows系统：将input.txt和target.txt文件放在此目录下，双击exe可执行文件，会在本目录下生成结果文件。

Linux系统：将input.txt和target.txt文件放在此目录下，在终端下执行，会在本目录下生成结果文件。

程序说明

为了优化空间，我采用了一个长度为27的字符串来存储每一个状态。

f1启发函数是错位的棋子数，代码如下：

```
int f1 (string curr_status) {
    int count = 0;
    for (int i = 0; i < final_status.size(); ++i)
        //0的位置错位将不被计入耗散值
        if (curr_status[i] != char(50) && final_status[i] !=
        curr_status[i])
            ++count;
    return count;
}
```

f2启发函数是所有棋子到其目标位置的三个方向的曼哈顿距离和，代码如下：

```
int f2 (string curr_status) {
    int sum = 0;
    for (int i = 0; i < curr_status.size(); ++i) {
        // 0的位置将不被计入耗散值
        if (curr_status[i] == final_status[i] ||
            curr_status[i] == char(50))
            continue;
        int index = final_status.find(curr_status[i]);
        sum += abs(index % 3 - i % 3) + abs((index / 3) % 3
        - (i / 3) % 3) + abs((index / 9) % 3 - (i / 9) % 3);
    }
    return sum;
}
```

- A*搜索

A*搜索算法要维护两个集合，close集记录已经搜索过的结点，open集记录还未搜索过的结点。首先，将初始状态放入open集。之后，在open集里面，选择不在close集中且耗散值最小的状态s进行扩展，将扩展后的新状态放入open集中。然后，将s从open集里面删除，并加入close集中。并重复上述步骤，直到搜索到目标状态。

为了优化算法，open集采用了优先队列来得到每一次耗散值最小的状态。

A*搜索的时间复杂度为O(b^d)，其中b为平均每一个状态扩展后的状态数，d为搜索空间的最大深度。

A*搜索的空间复杂度较大。可以证明，除非启发函数与实际路径耗散的偏差的增长不超过实际路径耗散的对数，即| h(n) - h*(n) | <= O(log h*(n))，否则结点数量会呈现指数级的增长。

```
string AStar(int(*f)(string)) {

    //moving记录转移状态和动作序列
    vector<pair<string, string> > moving;
    //close集记录已经搜索过的结点
    set<string> close;
    //优先队列，即open集，记录还未搜索过的结点，
    //队列的第一个结点是耗散值最小的结点
    queueStruct best_status;
    //队列非空，则一直循环
    while (!statusAStar.empty()) {
        //将耗散值最小的状态出队，将其放入close集
        best_status = statusAStar.top();
        statusAStar.pop();
        close.insert(best_status.s);
        //调用nextMove获得新的扩展状态
        moving = nextMove(best_status);
        for (int i = 0; i < moving.size(); ++i) {
            //如果新的状态中存在目标状态，直接返回相应的动作序列
            if (moving[i].first == final_status) {
                return moving[i].second;
            }
            //如果新的状态在close集中存在，直接跳过
            if (close.find(moving[i].first) != close.end()) continue;
            //将新的状态放入open集中
            queueStruct next_status;
            next_status.s = moving[i].first;
            next_status.first = f(moving[i].first);
            next_status.second = best_status.second + 1;
            next_status.mov = moving[i].second;
            statusAStar.push(next_status);
        }
    }
    //找不到返回空
    return "";
}
```

- IDA*搜索

IDA*搜索与A*搜索不同，它是一个深度优先遍历的过程，且不保留已经搜索过的状态。算法维护一个栈，程序一开始将初始状态压入栈中。在一个循环中，若栈非空，弹出栈顶的状态，由这个状态扩展新的状态，并放入栈中。直到找到目标状态，或者耗散值超过截断值，则结束这一结点的扩展。之后，选择超过截断值最小的耗散值，作为新的截断值，进入下一轮循环。

与A*搜索相比，IDA*最大的优点就是空间复杂度很小，大约为O(d)，其中d为最大搜索深度。

IDA*的时间复杂度为O(b^d)，其中b为平均每一个状态扩展后的状态数，d为搜索空间的最大深度。

```
string IDAStar(int(*f)(string)) {
    //statusIDAStar是一个栈，用于DFS
    queueStruct init_status = statusIDAStar[0];
    int d_limit = init_status.first + init_status.second;
    //每次更新截断值之后，进行新一轮的循环
    while (d_limit < INF) {
        int next_d_limit = INF;
        //从根部开始遍历
        statusIDAStar.clear();
        statusIDAStar.push_back(init_status);
        //栈不为空，则继续DFS
        while (!statusIDAStar.empty()) {
            //从栈顶弹出一个状态
            queueStruct s = statusIDAStar.back();
            statusIDAStar.pop_back();
            int ds = s.first + s.second;
            //如果状态的耗散值超过截断值，不进行扩展，同时更新截断值
            if (ds > d_limit) {
                next_d_limit = ((next_d_limit<ds)?next_d_limit:ds);
            }
            else {
                //若搜索到目标状态，返回动作序列
                if (s.s == final_status) {
                    return s.mov;
                }
                //进行状态的扩展，并压入栈
                vector<pair<string, string> > moving = nextMove(s);
                for (int i = 0; i < moving.size(); ++i) {
                    queueStruct next_status;
                    next_status.s = moving[i].first;
                    next_status.first = f(moving[i].first);
                    next_status.second = s.second + 1;
                    next_status.mov = moving[i].second;
                    statusIDAStar.push_back(next_status);
                }
            }
        }
        //更新截断值
        d_limit = next_d_limit;
    }
```

```
        //找不到结果，返回空
        return "";
    }
```