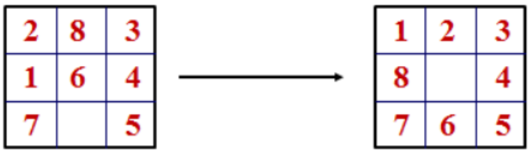
重排九宫问题

一、问题描述

在 3×3 的方格棋盘上放置分别标有数字 1，2，3，4，5，6，7，8 的 8 张牌，初始状态为 S_0，目标状态为 S_g，如下图所示。可使用的算符有空格左移、空格上移、空格右移和空格下移，即它们只允许把位于空格左、上、右、下边的牌移入空格。要求寻找从初始状态到目标状态的路径。



2	8	3
1	6	4
7		5

（初始状态）

1	2	3
8		4
7	6	5

（目标状态）

二、算法描述

1）判断结点是否重复

康托展开：

$$X = a[n] \cdot (n-1)! + a[n-1] \cdot (n-2)! + \cdots + a[i] \cdot (i-1)! + \cdots + a[1] \cdot 0!$$

其中，$a[i]$表示原数第 i 位在当前未出现元素中排在第几个，且$0 \le a[i] < i$，$1 \le i \le n$。

通过康托展开，可以将结点状态映射为 0~n!-1 间的整数，从而可判断结点的重复性。

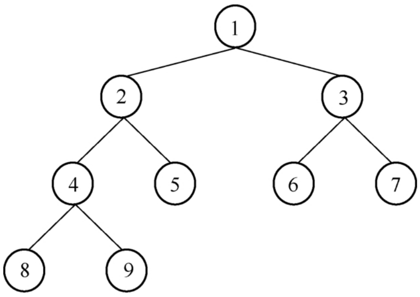
2）判断问题是否有解

一个状态表示成一维的形式，求出：除 0 之外所有数字的逆序数之和，也就是每个数字前面比它大的数字的个数的和，称为这个状态的逆序。若两个状态的逆序奇偶性相同，则可相互到达，否则不可相互到达。

3）广度优先搜索

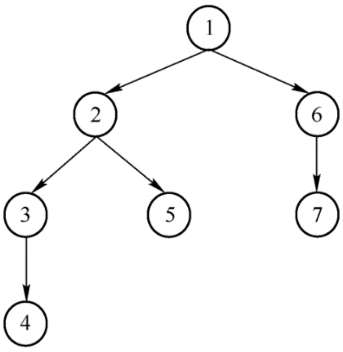
基本思想：从初始结点开始，逐层地对结点进行扩展并考察它是否为目标结点，在第 n

层的结点没有全部扩展并考察完之前，不对第 n+1 层的结点进行扩展。OPEN 表中的结点总是按进入的先后顺序排列，先进入的结点排在前面，后进入的结点排在后面。(示意图如下)



4）有界深度优先搜索

基本思想：从初始结点开始，在其子结点中选择一个结点进行考察，若不是目标结点，则再在该子结点中选择一个结点进行考察，一直如此向下搜索。当到达某个子结点，且该子结点既不是目标结点又不能继续扩展时，才选择其兄弟结点进行考察。但是该过程有可能陷入无穷分支的死循环而得不到解，因此，可以对深度优先搜索引入搜索深度的界限。当搜索深度达到了深度界限，且尚未出现目标结点时，就换一个分支进行搜索。(示意图如下)



5）A*算法

满足以下条件的搜索过程称为 A*算法：

1）把 OPEN 表中的结点按估价函数$f(x) = g(x) + h(x)$的值从小至大进行排序；

2）$g(x)$是对$g^*(x)$的估计，且$g(x) > 0$；

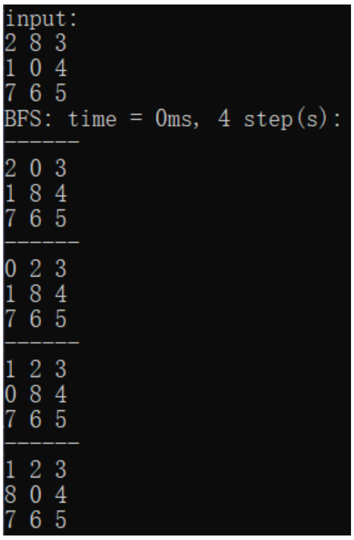
3）$h(x)$是$h^*(x)$的下界，即对所有的结点 x 均有$h(x) \leq h^*(x)$。

其中，$g^*(x)$是从初始结点到结点 x 的最小代价；$h^*(x)$是从结点 x 到目标结点的最小代价，若有多个目标结点，则为其中最小的一个。

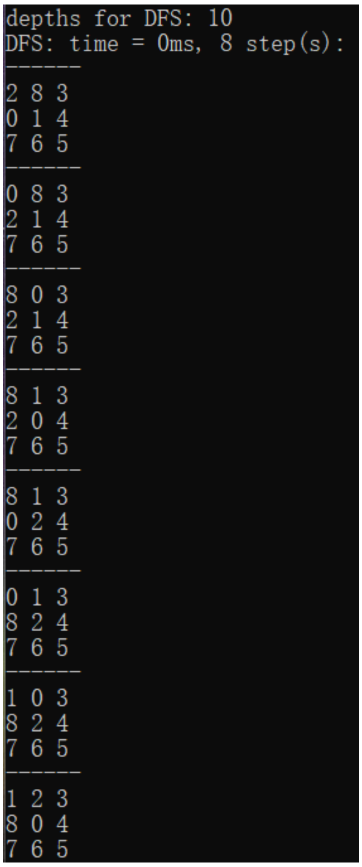
在重排九宫问题中，可选择将当前结点的深度作为$g(x)$，当前结点到目标结点的曼哈顿距离之和作为$h(x)$。

三、实验结果

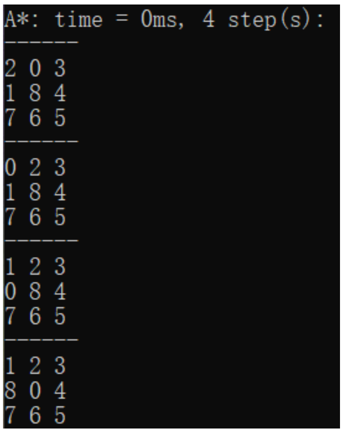
1）



```
input:
2 8 3
1 0 4
7 6 5
BFS: time = 0ms, 4 step(s):
------
2 0 3
1 8 4
7 6 5
------
0 2 3
1 8 4
7 6 5
------
1 2 3
0 8 4
7 6 5
------
1 2 3
8 0 4
7 6 5
```

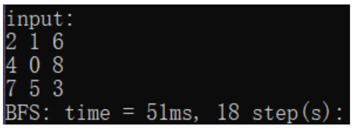


```
depths for DFS: 10
DFS: time = 0ms, 8 step(s):
------
2 8 3
0 1 4
7 6 5
------
0 8 3
2 1 4
7 6 5
------
8 0 3
2 1 4
7 6 5
------
8 1 3
2 0 4
7 6 5
------
8 1 3
0 2 4
7 6 5
------
0 1 3
8 2 4
7 6 5
------
1 0 3
8 2 4
7 6 5
------
1 2 3
8 0 4
7 6 5
```

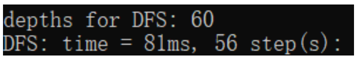


```
A*: time = 0ms, 4 step(s):
------
2 0 3
1 8 4
7 6 5
------
0 2 3
1 8 4
7 6 5
------
1 2 3
0 8 4
7 6 5
------
1 2 3
8 0 4
7 6 5
```

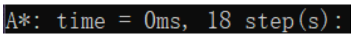
2）



```
input:
2 1 6
4 0 8
7 5 3
BFS: time = 51ms, 18 step(s):
```



```
depths for DFS: 60
DFS: time = 81ms, 56 step(s):
```



```
A*: time = 0ms, 18 step(s):
```

四、结果分析

1）测试样例：2，8，3，1，0，4，7，6，5

三种算法均给出了问题的解，且由于测试样例简单三者用时都极小，但有界深度优先算法的步骤最长；

2）测试样例：2，1，6，4，0，8，7，5，3

在该测试样例下，有界深度优先算法耗时最久，步骤最长；A*算法耗时最短，且步骤与

广度优先算法长度相等。

五、不同算法的对比

1）差异：广度优先搜索是将结点 n 的子结点放入到 OPEN 表的尾部（类似队列）；深度优先搜索是把结点 n 的子结点放入到 OPEN 表的首部（类似栈）；A*算法则是将结点 n 的子结点放入到 OPEN 表后，再将 OPEN 表中的结点按照估价函数进行排序（类似优先队列）。

2）性能：广度优先算法与 A*算法求解出的步骤长度都较短，且 A*算法耗时最短。而有界深度优先算法根据深度界限的不同，其求解出的步骤与耗时也不相同。

六、源代码

```
#include <iostream>
#include <algorithm>
#include <ctime>
#include <limits>
#include <vector>
#include <queue>
#include <stack>

//存储0!--(9-1)!
int fact[] = { 1, 1, 2, 6, 24, 120, 720, 5040, 40320 };

//康托展开
int cantor(int board[]) {
    int ans = 0;
    for (int i = 0; i < 9; ++i) {
        int a = 0;
        for (int j = i + 1; j < 9; ++j)
            if (board[i] > board[j]) ++a;
        ans += a * fact[8 - i];
    }
    return ans;
}

//逆康托展开
void rev_cantor(int num, int board[]) {
    std::vector<int> vec;
    for (int i = 0; i < 9; ++i) vec.push_back(i);
    for (int i = 0; i < 9; ++i) {
        int pos = num / fact[8 - i];
        board[i] = vec[pos];
        vec.erase(vec.begin() + pos);
        num %= fact[8 - i];
    }
}
```

```
//使用逆序数判断是否有解
bool access(int board1[], int board2[]) {
    int n1 = 0, n2 = 0;
    for (int i = 0; i < 9; ++i)
        for (int j = i + 1; j < 9; ++j)
            if (board1[i] != 0 && board1[j] != 0 && board1[i] > board1[j]) ++n1;
    for (int i = 0; i < 9; ++i)
        for (int j = i + 1; j < 9; ++j)
            if (board2[i] != 0 && board2[j] != 0 && board2[i] > board2[j]) ++n2;
    return n1 % 2 == n2 % 2;
}

//搜索树结点
struct Node {
    Node(int n = 0, Node* p1 = NULL, Node* p2 = NULL, int g = 0):
        num(n), self(p1), parent(p2), G(g), H(0) {
        if (g > 0) {
            //计算启发函数值
            int goal_pos[9][2] = { {1, 1}, {0, 0}, {0, 1}, {0, 2}, {1, 2}, {2, 2}, {2,
1}, {2, 0}, {1, 0} };
            int board[9];
            rev_cantor(num, board);
            for (int i = 0; i < 9; ++i) {
                int x = i / 3, y = i % 3;
                H += std::abs(x - goal_pos[board[i]][0]) + std::abs(y -
goal_pos[board[i]][1]);
            }
        }
    }
    bool operator<(const Node& node) const {
        return (this->G + this->H) > (node.G + node.H);
    }
    int num;          //康托展开值
    int G;            //初始结点到当前结点的代价（当前结点深度）
    int H;            //当前结点到目标结点的代价（曼哈顿距离）
    Node* self;
    Node* parent;
    static int goal[9];
};
int Node::goal[9] = { 1, 2, 3, 8, 0, 4, 7, 6, 5 };

//输入
void input(int board[]) {
    int flag[9];
begin:
    std::cout << "input:" << std::endl;
    for (int i = 0; i < 9; ++i) flag[i] = 0;
    for (int i = 0; i < 9; ++i) {
        std::cin >> board[i];
        if (board[i] >= 0 && board[i] < 9) {
            ++flag[board[i]];
            if (flag[board[i]] > 1) {
                std::cout << "Error input!" << std::endl;
                std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
                goto begin;
            }
        }
        else {
            std::cout << "Error input!" << std::endl;
            std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
            goto begin;
        }
    }
}

//输出
void output(Node node) {
    std::vector<Node> ans;
    while (node.parent) {
        ans.push_back(node);
        node = *(node.parent);
    }
    std::cout << ans.size() << " step(s):" << std::endl;
    for (int i = ans.size() - 1; i >= 0; --i) {
        std::cout << "------" << std::endl;
        int board[9];
        rev_cantor(ans[i].num, board);
        for (int j = 0; j < 3; ++j) {
            for (int k = 0; k < 3; ++k)
                std::cout << board[3 * j + k] << ' ';
            std::cout << std::endl;
        }
    }
}

bool visited[362880] = { false };

//广度优先搜索
void BFS(int Board[]) {
    int board[9];
    for (int i = 0; i < 9; ++i) board[i] = Board[i];

    clock_t start, end;
    start = clock();

    int ans = cantor(Node::goal);
    for (int i = 0; i < 362880; ++i) visited[i] = false;

    std::vector<Node> tree;
    std::queue<Node> Q;
    Node* node_ptr = new Node(cantor(board), NULL, NULL);
    node_ptr->self = node_ptr;
    Q.push(*node_ptr);
    visited[Q.front().num] = true;

    while (!Q.empty()) {
        Node node = Q.front();
        Q.pop();
        tree.push_back(node);

        if (node.num == ans) {
            end = clock();
            std::cout << "BFS: " << "time = " << end - start << "ms, ";
            output(node);
            while (!Q.empty()) {
                delete Q.front().self;
                Q.pop();
            }
            for (int i = tree.size() - 1; i >= 0; --i) delete tree[i].self;
            break;
        }

        rev_cantor(node.num, board);
        int pos = 0;
        for (/**/; board[pos] != 0; ++pos);
        int row = pos / 3;
        int col = pos % 3;

        //左移
        if (col > 0) {
            std::swap(board[pos], board[pos - 1]);
            int num = cantor(board);
            if (!visited[num]) {
                node_ptr = new Node(num, NULL, node.self);
                node_ptr->self = node_ptr;
                Q.push(*node_ptr);
                visited[num] = true;
            }
            std::swap(board[pos], board[pos - 1]);
        }
        //上移
        if (row > 0) {
            std::swap(board[pos], board[pos - 3]);
            int num = cantor(board);
            if (!visited[num]) {
                node_ptr = new Node(num, NULL, node.self);
                node_ptr->self = node_ptr;
                Q.push(*node_ptr);
                visited[num] = true;
            }
            std::swap(board[pos], board[pos - 3]);
        }
        //右移
        if (col < 2) {
            std::swap(board[pos], board[pos + 1]);
            int num = cantor(board);
            if (!visited[num]) {
                node_ptr = new Node(num, NULL, node.self);
                node_ptr->self = node_ptr;
                Q.push(*node_ptr);
                visited[num] = true;
            }
            std::swap(board[pos], board[pos + 1]);
        }
        //下移
        if (row < 2) {
            std::swap(board[pos], board[pos + 3]);
            int num = cantor(board);
```

```
            if (!visited[num]) {
                node_ptr = new Node(num, NULL, node.self);
                node_ptr->self = node_ptr;
                Q.push(*node_ptr);
                visited[num] = true;
            }
            std::swap(board[pos], board[pos + 3]);
        }
    }
}

//有界深度优先搜索
void DFS(int Board[], int d) {
    int board[9];
    for (int i = 0; i < 9; ++i) board[i] = Board[i];

    clock_t start, end;
    start = clock();

    int depth = d;
    int ans = cantor(Node::goal);
    for (int i = 0; i < 362880; ++i) visited[i] = false;

    std::vector<Node> tree;
    std::stack<Node> S;
    Node* node_ptr = new Node(cantor(board), NULL, NULL);
    node_ptr->self = node_ptr;
    S.push(*node_ptr);
    visited[S.top().num] = true;

    while (!S.empty()) {
        if (depth < 0) {
            ++depth;
            tree.push_back(S.top());
            S.pop();
        }

        Node node = S.top();

        if (node.num == ans) {
            end = clock();
            std::cout << "DFS: " << "time = " << end - start << "ms, ";
            output(node);
            while (!S.empty()) {
                delete S.top().self;
                S.pop();
            }
            for (int i = tree.size() - 1; i >= 0; --i) delete tree[i].self;
            break;
        }

        rev_cantor(node.num, board);
        int pos = 0;
        for (/**/; board[pos] != 0; ++pos);
        int row = pos / 3;
        int col = pos % 3;
```

```
//左移
if (col > 0) {
    std::swap(board[pos], board[pos - 1]);
    int num = cantor(board);
    if (!visited[num]) {
        node_ptr = new Node(num, NULL, node.self);
        node_ptr->self = node_ptr;
        S.push(*node_ptr);
        visited[num] = true;
        --depth;
        continue;
    }
    std::swap(board[pos], board[pos - 1]);
}
//上移
if (row > 0) {
    std::swap(board[pos], board[pos - 3]);
    int num = cantor(board);
    if (!visited[num]) {
        node_ptr = new Node(num, NULL, node.self);
        node_ptr->self = node_ptr;
        S.push(*node_ptr);
        visited[num] = true;
        --depth;
        continue;
    }
    std::swap(board[pos], board[pos - 3]);
}
//右移
if (col < 2) {
    std::swap(board[pos], board[pos + 1]);
    int num = cantor(board);
    if (!visited[num]) {
        node_ptr = new Node(num, NULL, node.self);
        node_ptr->self = node_ptr;
        S.push(*node_ptr);
        visited[num] = true;
        --depth;
        continue;
    }
    std::swap(board[pos], board[pos + 1]);
}
//下移
if (row < 2) {
    std::swap(board[pos], board[pos + 3]);
    int num = cantor(board);
    if (!visited[num]) {
        node_ptr = new Node(num, NULL, node.self);
        node_ptr->self = node_ptr;
        S.push(*node_ptr);
        visited[num] = true;
        --depth;
        continue;
    }
    std::swap(board[pos], board[pos + 3]);
```

```cpp
        }

        ++depth;
        tree.push_back(S.top());
        S.pop();
    }
    if (depth == d + 1) {
        for (int i = tree.size() - 1; i >= 0; --i) delete tree[i].self;
        std::cout << "No solution at current depth!" << std::endl;
    }
}

//A*算法
void Astar(int Board[]) {
    int board[9];
    for (int i = 0; i < 9; ++i) board[i] = Board[i];

    clock_t start, end;
    start = clock();

    int ans = cantor(Node::goal);
    for (int i = 0; i < 362880; ++i) visited[i] = false;

    std::vector<Node> tree;
    std::priority_queue<Node> Q;
    Node* node_ptr = new Node(cantor(board), NULL, NULL, 0);
    node_ptr->self = node_ptr;
    Q.push(*node_ptr);
    visited[Q.top().num] = true;

    while (!Q.empty()) {
        Node node = Q.top();
        Q.pop();
        tree.push_back(node);

        if (node.num == ans) {
            end = clock();
            std::cout << "A*: " << "time = " << end - start << "ms, ";
            output(node);
            while (!Q.empty()) {
                delete Q.top().self;
                Q.pop();
            }
            for (int i = tree.size() - 1; i >= 0; --i) delete tree[i].self;
            break;
        }

        rev_cantor(node.num, board);
        int pos = 0;
        for (/**/; board[pos] != 0; ++pos);
        int row = pos / 3;
        int col = pos % 3;

        //左移
        if (col > 0) {
            std::swap(board[pos], board[pos - 1]);
            int num = cantor(board);
            if (!visited[num]) {
                node_ptr = new Node(num, NULL, node.self, node.G + 1);
                node_ptr->self = node_ptr;
                Q.push(*node_ptr);
                visited[num] = true;
            }
            std::swap(board[pos], board[pos - 1]);
        }
        //上移
        if (row > 0) {
            std::swap(board[pos], board[pos - 3]);
            int num = cantor(board);
            if (!visited[num]) {
                node_ptr = new Node(num, NULL, node.self, node.G + 1);
                node_ptr->self = node_ptr;
                Q.push(*node_ptr);
                visited[num] = true;
            }
            std::swap(board[pos], board[pos - 3]);
        }
        //右移
        if (col < 2) {
            std::swap(board[pos], board[pos + 1]);
            int num = cantor(board);
            if (!visited[num]) {
                node_ptr = new Node(num, NULL, node.self, node.G + 1);
                node_ptr->self = node_ptr;
                Q.push(*node_ptr);
                visited[num] = true;
            }
            std::swap(board[pos], board[pos + 1]);
        }
        //下移
        if (row < 2) {
            std::swap(board[pos], board[pos + 3]);
            int num = cantor(board);
            if (!visited[num]) {
                node_ptr = new Node(num, NULL, node.self, node.G + 1);
                node_ptr->self = node_ptr;
                Q.push(*node_ptr);
                visited[num] = true;
            }
            std::swap(board[pos], board[pos + 3]);
        }
    }
}

int main(void) {
    //int board[] = { 2, 8, 3, 1, 0, 4, 7, 6, 5 };   //test
    //int board[] = { 2, 1, 6, 4, 0, 8, 7, 5, 3 };   //test
    int board[9];
    char ch;
    do {
        input(board);
        if (access(board, Node::goal)) {
```

```cpp
            BFS(board);
            std::cout << std::endl;
            int depth = 10;
            std::cout << "depths for DFS: ";
            std::cin >> depth;
            DFS(board, depth);
            std::cout << std::endl;
            Astar(board);
            std::cout << std::endl;
        }
        else
            std::cout << "No solution!" << std::endl;
        std::cout << "Press 'q' to exit:" << std::endl;
        std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
    } while (std::cin >> ch && ch != 'q');
}
```