搜索-最小步数模型

一、AcWing 1107. 魔板

【题目描述】

Rubik先生在发明了风靡全球的魔方之后，又发明了它的二维版本—魔板。

这是一张有8个大小相同的格子的魔板：

```
1 2 3 4
8 7 6 5
```

我们知道魔板的每一个方格都有一种颜色。

这8种颜色用前8个正整数来表示。

可以用颜色的序列来表示一种魔板状态，规定从魔板的左上角开始，沿顺时针方向依次取出整数，构成一个颜色序列。

对于上图的魔板状态，我们用序列$(1,2,3,4,5,6,7,8)$来表示，这是基本状态。

这里提供三种基本操作，分别用大写字母`A`，`B`，`C`来表示（可以通过这些操作改变魔板的状态）：

- `A`：交换上下两行；
- `B`：将最右边的一列插入到最左边；
- `C`：魔板中央对的4个数作顺时针旋转。

下面是对基本状态进行操作的示范：

`A`：

```
8 7 6 5
1 2 3 4
```

`B`：

```
4 1 2 3
5 8 7 6
```

`C`：

```
1 7 2 4
8 6 3 5
```

对于每种可能的状态，这三种基本操作都可以使用。

你要编程计算用最少的基本操作完成基本状态到特殊状态的转换，输出基本操作序列。

注意：数据保证一定有解。

【输入格式】

输入仅一行，包括8个整数，用空格分开，表示目标状态。

【输出格式】

输出文件的第一行包括一个整数，表示最短操作序列的长度。

如果操作序列的长度大于0，则在第二行输出字典序最小的操作序列。

【数据范围】

输入数据中的所有数字均为1到8之间的整数。

【输入样例】

```
2 6 8 4 5 7 3 1
```

【输出样例】

```
7
BCABCCB
```

【分析】

我们需要求出从初始状态`st = "12345678"`到某个终止状态`ed`所需的最少变换次数以及变换的步骤，通过观察规律可以找到`A、B、C`三种操作产生的变换后的字符串。

在BFS搜索的时候，我们按`A、B、C`的顺序进行状态的变换，那么一定能保证最后得到的操作序列字典序最小。

在记录不同状态的最短步数时可以用`unordered_map<string, int>`建立一个映射dis，$dis[s]$表示起始状态st变换到状态s所需的最少步数；记录每种状态的前驱状态时可以用`unordered_map<string, pair<char, string>>`建立一个映射pre，$pre[s].first$表示状态s的前驱状态变换过来所用的操作，$pre[s].second$表示状态s的前驱状态。

【代码】

```cpp
#include <iostream>
#include <cstring>
#include <algorithm>
#include <string>
#include <queue>
#include <unordered_map>
using namespace std;

string st = "12345678", ed;
unordered_map<string, int> dis;
unordered_map<string, pair<char, string> > pre;

string changeA(string s)
{
    reverse(s.begin(), s.end());//操作A就是翻转字符串
    return s;
}

string changeB(string s)
{
    string res;
    int op[8] = { 3, 0, 1, 2, 5, 6, 7, 4 };//操作B产生的字符串分别由原字符串的这些位置的字符构成
    for (int i = 0; i < 8; i++) res += s[op[i]];
    return res;
}

string changeC(string s)
{
    string res;
    int op[8] = { 0, 6, 1, 3, 4, 2, 5, 7 };//操作C产生的字符串分别由原字符串的这些位置的字符构成
    for (int i = 0; i < 8; i++) res += s[op[i]];
    return res;
}

void bfs()
{
    dis[st] = 0;
    queue<string> Q;

    Q.push(st);
```

```
    while (Q.size())
    {
        auto t = Q.front();
        Q.pop();
        if (t == ed) break;
        string sA = changeA(t), sB = changeB(t), sC = changeC(t);
        if (!dis[sA]) Q.push(sA), dis[sA] = dis[t] + 1, pre[sA].first = 'A', pre[sA].second = t;
        if (!dis[sB]) Q.push(sB), dis[sB] = dis[t] + 1, pre[sB].first = 'B', pre[sB].second = t;
        if (!dis[sC]) Q.push(sC), dis[sC] = dis[t] + 1, pre[sC].first = 'C', pre[sC].second = t;
    }
}

int main()
{
    for (int i = 0; i < 8; i++) { char c; cin >> c; ed += c; }
    bfs();
    cout << dis[ed] << endl;
    if (dis[ed] > 0)
    {
        string res;
        while (ed != st)
        {
            res += pre[ed].first;
            ed = pre[ed].second;
        }
        reverse(res.begin(), res.end());
        cout << res << endl;
    }
    return 0;
}
```

二、AcWing 845. 八数码

【题目描述】

在一个3×3的网格中，$1 \sim 8$这8个数字和一个`x`恰好不重不漏地分布在这3×3的网格中。

例如：

```
1 2 3
x 4 6
7 5 8
```

在游戏过程中，可以把`x`与其上、下、左、右四个方向之一的数字交换（如果存在）。

我们的目的是通过交换，使得网格变为如下排列（称为正确排列）：

```
1 2 3
4 5 6
7 8 x
```

例如，示例中图形就可以通过让`x`先后与右、下、右三个方向的数字交换成功得到正确排列。

交换过程如下：

```
1 2 3   1 2 3   1 2 3   1 2 3
x 4 6   4 x 6   4 5 6   4 5 6
7 5 8   7 5 8   7 x 8   7 8 x
```

现在，给你一个初始网格，请你求出得到正确排列至少需要进行多少次交换。

【输入格式】

输入占一行，将3×3的初始网格描绘出来。

例如，如果初始网格如下所示：

```
1 2 3
x 4 6
7 5 8
```

则输入为：`1 2 3 x 4 6 7 5 8`

【输出格式】

输出占一行，包含一个整数，表示最少交换次数。

如果不存在解决方案，则输出-1。

【输入样例】

```
2  3  4  1  5  x  7  6  8
```

【输出样例】

```
19
```

【分析】

本题同上题的分析一样，模拟出每一种状态的变换方式，搜索所有能够变换到的状态即可。

【代码】

```cpp
#include <iostream>
#include <cstring>
#include <algorithm>
#include <string>
#include <queue>
#include <unordered_map>
using namespace std;

string st, ed = "12345678x";
unordered_map<string, int> dis;
int dx[4] = { 0, 1, 0, -1 }, dy[4] = { 1, 0, -1, 0 };

int bfs()
{
    dis[st] = 0;
    queue<string> Q;
    Q.push(st);
    while (Q.size())
    {
        auto t = Q.front();
        Q.pop();
        if (t == ed) return dis[t];
        int idx = t.find('x');//找到字符'x'所对应的下标
        int x = idx / 3, y = idx % 3;//一维长度映射到二维坐标
        for (int i = 0; i < 4; i++)
        {
            int nx = x + dx[i], ny = y + dy[i];
            if (nx >= 0 && nx < 3 && ny >= 0 && ny < 3)
            {
                //交换一维字符串中的对应字符
                int d = dis[t];

                swap(t[nx * 3 + ny], t[idx]);
```

```
                if (!dis.count(t)) Q.push(t), dis[t] = d + 1;
                swap(t[nx * 3 + ny], t[idx]);//注意将t还原
            }
        }
    }
    return -1;
}

int main()
{
    for (int i = 0; i < 9; i++) { char c; cin >> c; st += c; }
    cout << bfs() << endl;
    return 0;
}
```