搜索-A*

一、AcWing 179. 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
```

把x与上下左右方向数字交换的行动记录为`u、d、l、r`。

现在，给你一个初始网格，请你通过最少的移动次数，得到正确排列。

【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，包含一个字符串，表示得到正确排列的完整行动记录。

如果答案不唯一，输出任意一种合法方案即可。

如果不存在解决方案，则输出`unsolvable`。

【输入样例】

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

【输出样例】

```
ullddrurdllurdruldr
```

【分析】

什么是A*算法？

1. 做法：

   引入一个估值函数，用来估计某个点到达终点的距离。

   记f是估值函数，g是真实值，那么$f(state) \leq g(state)$，越接近越好（当估值是0时，类似于$Dijkstra$算法）；

   记$dis[state]$是从起点到$state$状态的步数；

   利用的是优先队列，排序依据是$dis[state] + f(state)$

2. 证明：

   假设终点第一次出堆时不是最小值，那么意味着$dis[end] > dis_{最优}$

   那么说明堆中存在一个最优路径中的某个点（起码起点在路径上），记该点为u

   $dis_{最优} = dis[u] + g(u) \geq dis[u] + f(u)$，即$dis[end] > dis_{最优} \geq dis[u] + f(u)$，说明优先队列中存在一个比出堆元素更小的值，这就矛盾了。

   所以终点第一次出堆时就是最优的。

3. 应用的环境：

（1）有解（无解时，仍然会把所有空间搜索，会比一般的BFS慢，因为优先队列的操作是$O(logn)$的）；

（2）边权非负，如果是负数，那么终点的估值有可能是负无穷，终点可能会直接出堆。

4. 性质：

除了终点以外的其他点无法在出堆或者入堆的时候确定距离，只能保证终点出堆时是最优的可以。

【代码】

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

typedef pair<int, string> PIS;
string seq, st, ed = "12345678x";
unordered_map<string, int> dis;
unordered_map<string, pair<char, string> > pre;
int dx[4] = { -1, 0, 1, 0 }, dy[4] = { 0, 1, 0, -1 };
char op[4] = { 'u', 'r', 'd', 'l' };

//启发函数，返回值为当前状态中每个数字所在的位置与最终状态中每个数字所在的位置的曼哈顿距离
int f(string str)
{
    int res = 0;
    for (int i = 0; i < str.size(); i++)
        if (str[i] != 'x')
        {
            int t = str[i] - '1';//该数字在ed中的位置
            res += abs(i / 3 - t / 3) + abs(i % 3 - t % 3);//res加上该数字当前位置与最终位置的曼哈顿距离
        }
    return res;
}


```

```cpp
string a_star()
{
    priority_queue<PIS, vector<PIS>, greater<PIS> > Q;
    dis[st] = 0;
    Q.push({ dis[st] + f(st), st });
    while (Q.size())
    {
        auto t = Q.top().second;
        Q.pop();
        if (t == ed) break;
        int idx = t.find('x');
        int x = idx / 3, y = idx % 3;
        string source = t;
        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)
            {
                swap(t[nx * 3 + ny], t[idx]);
                if (!dis.count(t) || dis[source] + 1 < dis[t])
                {
                    dis[t] = dis[source] + 1;
                    Q.push({ dis[t] + f(t), t });
                    pre[t] = { op[i], source };
                }
                t = source;//复原
            }
        }
    }
    string res;
    while (ed != st) res += pre[ed].first, ed = pre[ed].second;
    reverse(res.begin(), res.end());
    return res;
}

int main()
{
    for (int i = 0; i < 9; i++) { char c; cin >> c; st += c; if (c != 'x') seq += c; }
    int cnt = 0;//逆序对数量，如果为奇数那么八数码问题无解
    for (int i = 0; i < seq.size() - 1; i++)
        for (int j = i + 1; j < seq.size(); j++)

            if (seq[j] < seq[i]) cnt++;
```

```
72      if (cnt & 1) puts("unsolvable");
73      else cout << a_star() << endl;
74      return 0;
75  }
```

二、AcWing 178. 第K短路

【题目描述】

给定一张N个点（编号$1,2...N$），M条边的有向图，求从起点S到终点T的第K短路的长度，路径允许重复经过点或边。

注意：每条最短路中至少要包含一条边。

【输入格式】

第一行包含两个整数N和M。

接下来M行，每行包含三个整数A,B和L，表示点A与点B之间存在有向边，且边长为L。

最后一行包含三个整数S,T和K，分别表示起点S，终点T和第K短路。

【输出格式】

输出占一行，包含一个整数，表示第K短路的长度，如果第K短路不存在，则输出-1。

【数据范围】

$1 \leq S,T \leq N \leq 1000$

$0 \leq M \leq 10^5$

$1 \leq K \leq 1000$

$1 \leq L \leq 100$

【输入样例】

```
2 2
1 2 5
2 1 4
1 2 2
```

【输出样例】

```
14
```

【分析】

第几次出队就是第几短，于是终点出了k次就是第k短路了。

按照$Dijkstra$的思想，我们每次取出$distance + dis[x]$最小的，其中，$distance$表示当前走过的真实距离，dis表示某个点到终点的最短距离，这样尽量大且一定比现在的解小。

然后更新所有能到达的点。

于是先从终点倒着跑一遍$Dijkstra$，求出dis，然后A*，直到终点第k次出队列此时的$distance$即为第k短路的长度。

【代码】

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

typedef pair<int, int> PII;
typedef pair<int, PII> PIII;
const int N = 1010, M = 200010;
int e[M], ne[M], d[M], h[N], rh[N], idx;
int dis[N];
bool st[N];
int n, m, S, T, K;

void add(int h[], int u, int v, int w)
{
    e[idx] = v, d[idx] = w, ne[idx] = h[u], h[u] = idx++;
}

//本题建反向边跑dijkstra求各点到终点的最短距离作为估计值，一定小于等于真实路线走的距离
void dijkstra()
{
    memset(dis, 0x3f, sizeof dis);
    dis[T] = 0;
    priority_queue<PII, vector<PII>, greater<PII> > Q;
    Q.push({ 0, T });
    while (Q.size())
    {
        auto t = Q.top().second;
        Q.pop();
```

```
        if (st[t]) continue;
        st[t] = true;
        for (int i = rh[t]; ~i; i = ne[i])
            if (dis[t] + d[i] < dis[e[i]])
            {
                dis[e[i]] = dis[t] + d[i];
                Q.push({ dis[e[i]], e[i] });
            }
    }
}

int a_star()
{
    if (dis[S] == 0x3f3f3f3f) return -1;//如果不连通则无解
    priority_queue<PIII, vector<PIII>, greater<PIII> > Q;
    Q.push({ dis[S], { 0, S } });//分别存储估计距离、真实距离、顶点号
    int cnt = 0;//记录终点是第几次出队，第K次出队即第K小
    while (Q.size())
    {
        auto t = Q.top().second.second;
        auto distance = Q.top().second.first;
        Q.pop();
        if (t == T) if (++cnt == K) return distance;
        for (int i = h[t]; ~i; i = ne[i])//把所有可以扩展到的点都加进来
            Q.push({ distance + d[i] + dis[e[i]], { distance + d[i], e[i]
} });
    }
    return -1;
}

int main()
{
    ios::sync_with_stdio(false);
    cin >> n >> m;
    memset(h, -1, sizeof h);
    memset(rh, -1, sizeof rh);
    for (int i = 0; i < m; i++)
    {
        int a, b, c;
        cin >> a >> b >> c;
        add(h, a, b, c);
        add(rh, b, a, c);

    }
```

```
    cin >> S >> T >> K;
    if (S == T) K++;//由于每条最短路中至少要包含一条边，因此当起点终点重合时要多走一遍
    dijkstra();
    cout << a_star() << endl;
    return 0;
}
```