搜索-最短路模型

一、AcWing 1076. 迷宫问题

【题目描述】

给定一个由$0,1$组成的$n \times n$的二维数组。

它表示一个迷宫，1表示墙壁，0表示可以走的路，只能横着走或竖着走，不能斜着走，要求编程序找出从左上角到右下角的最短路线。

数据保证至少存在一条从左上角走到右下角的路径。

【输入格式】

第一行包含整数n。

接下来n行，每行包含n个整数0或1，表示迷宫。

【输出格式】

输出从左上角到右下角的最短路线，如果答案不唯一，输出任意一条路径均可。

按顺序，每行输出一个路径中经过的单元格的坐标，左上角坐标为$(0,0)$，右下角坐标为$(n-1,n-1)$。

【数据范围】

$0 \leq n \leq 1000$

【输入样例】

```
5
0 1 0 0 0
0 1 0 1 0
0 0 0 0 0
0 1 1 1 0
0 0 0 1 0
```

【输出样例】

```
0 0
1 0
2 0
2 1
2 2
2 3
2 4
3 4
4 4
```

【分析】

本题需要记录最短路的路径信息，因此可以维护一个数组$pre[x][y]$表示在某条最短路径中，点(x,y)的前驱点坐标。然后我们从终点$(n-1,n-1)$开始搜索，搜到起点停止，最后从起点开始不断寻找前驱点直到找到终点为止，所输出的路径即为起点至终点的最短路径。

【代码】

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

typedef pair<int, int> PII;
const int N = 1010;
int g[N][N];
PII pre[N][N];//存储每个点的前驱结点
int n;
int dx[4] = { 0, 1, 0, -1 }, dy[4] = { 1, 0, -1, 0 };

void bfs(int x, int y)
{
    memset(pre, -1, sizeof pre);//当某个点的前驱结点为-1时说明还没被搜索到
    pre[x][y] = { x, y };//终点的pre就是自己
    queue<PII> Q;
    Q.push({ x, y });
    while (Q.size())
    {
        auto t = Q.front();

        Q.pop();
```

```
        for (int i = 0; i < 4; i++)
        {
            int nx = t.first + dx[i], ny = t.second + dy[i];
            if (nx >= 0 && nx < n && ny >= 0 && ny < n && !g[nx][ny] &&
!~pre[nx][ny].first)
                Q.push({ nx, ny }), pre[nx][ny] = { t.first, t.second };
        }
    }
}

int main()
{
    ios::sync_with_stdio(false);
    cin >> n;
    for (int i = 0; i < n; i++)
        for (int j = 0; j < n; j++)
            cin >> g[i][j];
    bfs(n - 1, n - 1);//从终点反向搜索，这样从起点根据pre找路径更方便
    int x = 0, y = 0;
    while (pre[x][y] != make_pair(x, y))//还没到终点就不断往前查找
    {
        cout << x << ' ' << y << endl;
        auto t = pre[x][y];
        x = t.first, y = t.second;
    }
    cout << x << ' ' << y << endl;//输出终点
    return 0;
}
```

二、AcWing 188. 武士风度的牛

【题目描述】

农民John有很多牛，他想交易其中一头被Don称为The Knight的牛。

这头牛有一个独一无二的超能力，在农场里像Knight一样地跳（就是我们熟悉的象棋中马的走法）。

虽然这头神奇的牛不能跳到树上和石头上，但是它可以在牧场上随意跳，我们把牧场用一个x,y的坐标图来表示。

这头神奇的牛像其它牛一样喜欢吃草，给你一张地图，上面标注了The Knight的开始位置，树、灌木、石头以及其它障碍的位置，除此之外还有一捆草。

现在你的任务是，确定The Knight要想吃到草，至少需要跳多少次。

The Knight的位置用`K`来标记，障碍的位置用`*`来标记，草的位置用`H`来标记。

这里有一个地图的例子：

```
11 | . . . . . . . . . .
10 | . . . . * . . . . .
 9 | . . . . . . . . . .
 8 | . . . * . * . . . .
 7 | . . . . . . . * . .
 6 | . . * . . * . . . H
 5 | * . . . . . . . . .
 4 | . . . * . . . * . .
 3 | . K . . . . . . . .
 2 | . . . * . . . . . *
 1 | . . * . . . . * . .
 0 ----------------------
   0 1 2 3 4 5 6 7 8 9 10
```

注意：数据保证一定有解。

【输入格式】

第1行：两个数，表示农场的列数C和行数R。

第$2\sim R+1$行：每行一个由C个字符组成的字符串，共同描绘出牧场地图。

【输出格式】

一个整数，表示跳跃的最小次数。

【数据范围】

$1\leq R,C\leq 150$

【输入样例】

```
10 11
..........
....*.....
..........
...*.*....
.......*..
..*..*...H
*.........
...*...*..
.K........
...*.....*
..*....*..
```

【输出样例】

```
5
```

【分析】

BFS直接爆搜即可，注意设置好八个方向的移动向量。

【代码】

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

typedef pair<int, int> PII;
const int N = 160;
char g[N][N];
int dis[N][N];
int n, m, sx, sy;
int dx[8] = { -2, -1, 1, 2, 2, 1, -1, -2 };
int dy[8] = { 1, 2, 2, 1, -1, -2, -2, -1 };

int bfs()
{
    memset(dis, -1, sizeof dis);

    dis[sx][sy] = 0;
```

```
    queue<PII> Q;
    Q.push({ sx, sy });
    while (Q.size())
    {
        auto t = Q.front();
        Q.pop();
        if (g[t.first][t.second] == 'H') return dis[t.first][t.second];
        for (int i = 0; i < 8; i++)
        {
            int nx = t.first + dx[i], ny = t.second + dy[i];
            if (nx >= 0 && nx < n && ny >= 0 && ny < m && g[nx][ny] != '*' && !~dis[nx][ny])
                Q.push({ nx, ny }), dis[nx][ny] = dis[t.first][t.second] + 1;
        }
    }
    return 20030925;
}

int main()
{
    cin >> m >> n;
    for (int i = 0; i < n; i++)
        for (int j = 0; j < m; j++)
            if (cin >> g[i][j], g[i][j] == 'K')
                sx = i, sy = j;
    cout << bfs() << endl;
    return 0;
}
```

三、AcWing 1100. 抓住那头牛

【题目描述】

农夫知道一头牛的位置，想要抓住它。

农夫和牛都位于数轴上，农夫起始位于点N，牛位于点K。

农夫有两种移动方式：

- 从X移动到$X-1$或$X+1$，每次移动花费一分钟
- 从X移动到$2*X$，每次移动花费一分钟

假设牛没有意识到农夫的行动，站在原地不动。

农夫最少要花多少时间才能抓住牛？

【输入格式】

共一行，包含两个整数N和K。

【输出格式】

输出一个整数，表示抓到牛所花费的最少时间。

【数据范围】

$0 \leq N, K \leq 10^5$

【输入样例】

```
5 17
```

【输出样例】

```
4
```

【分析】

- 对于$X-1$的移动，如果移动后的坐标≥ 0，那么将$X-1$的状态加入BFS队列；
- 对于$X+1$的移动，如果移动后的坐标$\leq K$，那么将$X+1$的状态加入BFS队列；
- 对于$X*2$的移动，如果移动后的坐标$< K+2$，那么将$X*2$的状态加入BFS队列，因为如果移动后的坐标$\geq K+2$，那么就应该先进行$X-1$再进行$X*2$，这样花费的时间一定更少。

【代码】

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

const int N = 100010;
int dis[N];
int n, k;

int bfs()
{
```

```
13      memset(dis, -1, sizeof dis);
14      dis[n] = 0;
15      queue<int> Q;
16      Q.push(n);
17      while (Q.size())
18      {
19          int t = Q.front();
20          Q.pop();
21          if (t == k) return dis[t];
22          if (t - 1 >= 0 && !~dis[t - 1]) Q.push(t - 1), dis[t - 1] =
    dis[t] + 1;
23          if (t + 1 <= k && !~dis[t + 1]) Q.push(t + 1), dis[t + 1] =
    dis[t] + 1;
24          if ((t << 1) - k < 2 && !~dis[t << 1]) Q.push(t << 1), dis[t <<
    1] = dis[t] + 1;
25      }
26      return 20030925;
27  }
28
29  int main()
30  {
31      cin >> n >> k;
32      cout << bfs() << endl;
33      return 0;
34  }
```