搜索与图论-DFS

一、AcWing 842. 排列数字

【题目描述】

给定一个整数n，将数字$1 \sim n$排成一排，将会有很多种排列方法。

现在，请你按照字典序将所有的排列方法输出。

【输入格式】

共一行，包含一个整数n。

【输出格式】

按字典序输出所有排列方案，每个方案占一行。

【数据范围】

$1 \leq n \leq 7$

【输入样例】

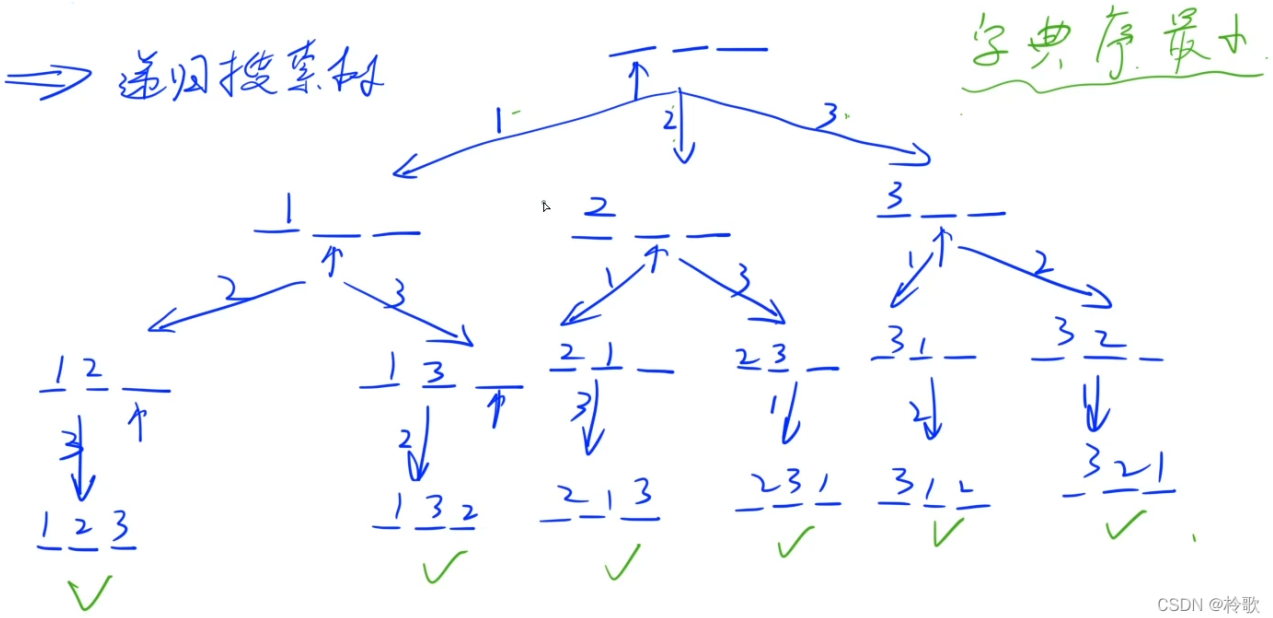
```
3
```

【输出样例】

```
1 2 3
1 3 2
2 1 3
2 3 1
3 1 2
3 2 1
```

【分析】

递归搜索树如下图所示：



【代码】

```cpp
#include <iostream>
using namespace std;

const int N = 10;
int res[N];//res保存排列结果
int n;

void dfs(int u, int state)//state的第i位如果为1表示i已被使用,为0表示未被使用
{
    if (u == n)
    {
        for (int i = 0; i < n; i++) cout << res[i] << ' ';
        cout << endl;
        return;
    }
    for (int i = 1; i <= n; i++)
        if (!(state >> i & 1)) res[u] = i, dfs(u + 1, state | 1 << i);
}

int main()
{
    cin >> n;
    dfs(0, 0);
    return 0;
}
```

二、AcWing 843. n-皇后问题

【题目描述】

N皇后问题是指将n个皇后放在$n \times n$的国际象棋棋盘上，使得皇后不能相互攻击到，即任意两个皇后都**不能处于同一行、同一列或同一斜线上**。

现在给定整数n，请你输出所有的满足条件的棋子摆法。

【输入格式】

共一行，包含整数n。

【输出格式】

每个解决方案占n行，每行输出一个长度为n的字符串，用来表示完整的棋盘状态。

其中`.`表示某一个位置的方格状态为空，`Q`表示某一个位置的方格上摆着皇后。

每个方案输出完成后，输出一个空行。

注意：行末不能有多余空格。

输出方案的顺序任意，只要不重复且没有遗漏即可。

【数据范围】

$1 \leq n \leq 9$

【输入样例】

```
4
```

【输出样例】

```
.Q..
...Q
Q...
..Q.

..Q.
Q...
...Q
.Q..

```

【代码】

```cpp
#include <iostream>
using namespace std;

const int N = 30;
int n;
char g[N][N];
bool col[N], dg[N], udg[N];

//逐行搜索
void dfs(int x)
{
    if (x == n)
    {
        for (int i = 0; i < n; i++) puts(g[i]);
        puts("");
        return;
    }
    //枚举每一列
    for (int y = 0; y < n; y++)
        //col标记该列是否危险，dg和udg标记正反对角线是否危险
        //由y=x+b和y=-x+b知可用b唯一表示对角线，防止y-x为负加上偏移量n即可
        if (!col[y] && !dg[y - x + n] && !udg[y + x])
        {
            g[x][y] = 'Q';
            col[y] = dg[y - x + n] = udg[y + x] = true;
            dfs(x + 1);
            col[y] = dg[y - x + n] = udg[y + x] = false;
            g[x][y] = '.';
        }
}

int main()
{
    cin >> n;
    for (int i = 0; i < n; i++)
        for (int j = 0; j < n; j++)
            g[i][j] = '.';
    dfs(0);//从第0行开始搜索
    return 0;
}
```

搜索与图论-BFS

一、AcWing 844. 走迷宫

【题目描述】

给定一个$n \times m$的二维整数数组，用来表示一个迷宫，数组中只包含0或1，其中0表示可以走的路，1表示不可通过的墙壁。

最初，有一个人位于左上角$(1,1)$处，已知该人每次可以向上、下、左、右任意一个方向移动一个位置。

请问，该人从左上角移动至右下角(n,m)处，至少需要移动多少次。

数据保证$(1,1)$处和(n,m)处的数字为0，且一定至少存在一条通路。

【输入格式】

第一行包含两个整数n和m。

接下来n行，每行包含m个整数(0或1)，表示完整的二维数组迷宫。

【输出格式】

输出一个整数，表示从左上角移动至右下角的最少移动次数。

【数据范围】

$1 \leq n,m \leq 100$

【输入样例】

```
5 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】

```
8
```

【代码】

```
#include <iostream>
#include <cstring>
#include <algorithm>
#include <queue>
```

```cpp
using namespace std;

typedef pair<int, int> PII;
const int N = 110;
int g[N][N], dis[N][N];
int dx[4] = { 0, 1, 0, -1 }, dy[4] = { 1, 0, -1, 0 };
int n, m;

int bfs()
{
    memset(dis, -1, sizeof dis);
    dis[1][1] = 0;
    queue<PII> Q;
    Q.push({ 1, 1 });
    while (Q.size())
    {
        auto t = Q.front();
        Q.pop();
        for (int i = 0; i < 4; i++)
        {
            int x = t.first + dx[i], y = t.second + dy[i];
            if (x >= 1 && x <= n && y >= 1 && y <= m && g[x][y] == 0 &&
!~dis[x][y])
                dis[x][y] = dis[t.first][t.second] + 1, Q.push({ x, y });
        }
    }
    return dis[n][m];
}

int main()
{
    cin >> n >> m;
    for (int i = 1; i <= n; i++)
        for (int j = 1; j <= m; j++)
            cin >> g[i][j];
    cout << bfs() << 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
```

【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;
}
```

搜索与图论-树与图的深度优先遍历

一、AcWing 846. 树的重心

【题目描述】

给定一颗树，树中包含n个结点（编号$1 \sim n$）和$n-1$条无向边。

请你找到树的重心，并输出将重心删除后，剩余各个连通块中点数的最大值。

重心定义：重心是指树中的一个结点，如果将这个点删除后，剩余各个连通块中点数的最大值最小，那么这个节点被称为树的重心。

【输入格式】

第一行包含整数n，表示树的结点数。

接下来$n-1$行，每行包含两个整数a和b，表示点a和点b之间存在一条边。

【输出格式】

输出一个整数m，表示将重心删除后，剩余各个连通块中点数的最大值。

【数据范围】

$1 \leq n \leq 10^5$

【输入样例】

```
9
1 2
1 7
1 4
2 8
2 5
4 3
3 9
4 6
```

【输出样例】

```
4
```

【代码】

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

const int N = 100010, M = N * 2;
const int INF = 0x3f3f3f3f;
bool vis[N];
int e[M], ne[M], h[N], idx;
int n, ans = INF;

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

//返回以u为根结点的子树中点的数量
int dfs(int u)
{
    vis[u] = true;
    //sum表示子树结点数量，res表示去掉u后各连通块点数的最大值
    int sum = 1, res = 0;
    for (int i = h[u]; ~i; i = ne[i])
    {
        int j = e[i];
        if (!vis[j])
        {
```

```
            int s = dfs(j);
            res = max(res, s);
            sum += s;
        }
    }
    //与u的父节点所在的连通块点数取max
    res = max(res, n - sum);
    ans = min(ans, res);
    return sum;
}

int main()
{
    cin >> n;
    memset(h, -1, sizeof h);
    for (int i = 1; i < n; i++)
    {
        int a, b;
        cin >> a >> b;
        add(a, b), add(b, a);
    }
    dfs(1);
    cout << ans << endl;
    return 0;
}
```

搜索与图论-树与图的广度优先遍历

一、AcWing 847. 图中点的层次

【题目描述】

给定一个n个点m条边的有向图，图中可能存在重边和自环。

所有边的长度都是1，点的编号为$1 \sim n$。

请你求出1号点到n号点的最短距离，如果从1号点无法走到n号点，输出-1。

【输入格式】

第一行包含两个整数n和m。

接下来m行，每行包含两个整数a和b，表示存在一条从a走到b的长度为1的边。

【输出格式】

输出一个整数，表示1号点到n号点的最短距离。

【数据范围】

$1 \leq n, m \leq 10^5$

【输入样例】

```
4 5
1 2
2 3
3 4
1 3
1 4
```

【输出样例】

```
1
```

【代码】

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

const int N = 100010, M = 100010;
int e[M], ne[M], h[N], idx;
int n, m, dis[N];

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

int bfs()
{
    memset(dis, -1, sizeof dis);
    queue<int> Q;
    dis[1] = 0;

    Q.push(1);
```

```
    while (Q.size())
    {
        int t = Q.front();
        Q.pop();
        for (int i = h[t]; ~i; i = ne[i])
            if (!~dis[e[i]]) Q.push(e[i]), dis[e[i]] = dis[t] + 1;
    }
    return dis[n];
}

int main()
{
    cin >> n >> m;
    memset(h, -1, sizeof h);
    for (int i = 0; i < m; i++)
    {
        int a, b;
        cin >> a >> b;
        add(a, b);
    }
    cout << bfs() << endl;
    return 0;
}
```

搜索与图论-拓扑排序

一、AcWing 848. 有向图的拓扑序列

【题目描述】

给定一个n个点m条边的有向图，点的编号是$1 \sim n$，图中可能存在重边和自环。

请输出任意一个该有向图的拓扑序列，如果拓扑序列不存在，则输出-1。

若一个由图中所有点构成的序列A满足：对于图中的每条边(x, y)，x在A中都出现在y之前，则称A是该图的一个拓扑序列。

【输入格式】

第一行包含两个整数n和m。

接下来m行，每行包含两个整数x和y，表示存在一条从点x到点y的有向边(x, y)。

【输出格式】

共一行，如果存在拓扑序列，则输出任意一个合法的拓扑序列即可。

否则输出-1。

【数据范围】

$1 \leq n, m \leq 10^5$

【输入样例】

```
3 3
1 2
2 3
1 3
```

【输出样例】

```
1 2 3
```

【代码】

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

const int N = 100010, M = 100010;
int e[M], ne[M], h[N], idx;
int in[N], res[N], cnt;
int n, m;

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

void topSort()
{
    queue<int> Q;
    for (int i = 1; i <= n; i++)
        if (!in[i]) Q.push(i);
    while (Q.size())
    {

        int t = Q.front();
```

```
        Q.pop();
        res[++cnt] = t;
        for (int i = h[t]; ~i; i = ne[i])
            if (!--in[e[i]]) Q.push(e[i]);
    }
}

int main()
{
    cin >> n >> m;
    memset(h, -1, sizeof h);
    while (m--)
    {
        int a, b;
        cin >> a >> b;
        add(a, b);
        in[b]++;//b的入度++
    }
    topSort();
    if (cnt == n)
        for (int i = 1; i <= n; i++) cout << res[i] << ' ';
    else cout << -1 << endl;
    return 0;
}
```

搜索与图论-Dijkstra

一、AcWing 849/850. Dijkstra求最短路 I/II

【题目描述】

给定一个n个点m条边的有向图，图中可能存在重边和自环，所有边权均为正值。

请你求出1号点到n号点的最短距离，如果无法从1号点走到n号点，则输出-1。

【输入格式】

第一行包含整数n和m。

接下来m行每行包含三个整数x, y, z，表示存在一条从点x到点y的有向边，边长为z。

【输出格式】

输出一个整数，表示1号点到n号点的最短距离。

如果路径不存在，则输出-1。

【朴素版数据范围】

$1 \leq n \leq 500$

$1 \leq m \leq 10^5$

图中涉及边长均不超过10000

【堆优化版数据范围】

$1 \leq n, m \leq 1.5 \times 10^5$

图中涉及边长均不小于0，且不超过10000。

数据保证：如果最短路存在，则最短路的长度不超过10^9。

【输入样例】

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

【输出样例】

```
3
```

【朴素版Dijkstra代码】

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

const int N = 510;
int n, m;
int g[N][N], dis[N], vis[N];

int dijkstra()
{
    memset(dis, 0x3f, sizeof dis);
    dis[1] = 0;
    for (int i = 0; i < n; i++)

    {
```

```
        int t = -1;
        for (int j = 1; j <= n; j++)
            if (!vis[j] && (t == -1 || dis[j] < dis[t]))
                t = j;
        vis[t] = 1;
        for (int j = 1; j <= n; j++)
            dis[j] = min(dis[j], dis[t] + g[t][j]);
    }
    return dis[n] == 0x3f3f3f3f ? -1 : dis[n];
}

int main()
{
    cin >> n >> m;
    memset(g, 0x3f, sizeof g);
    while (m--)
    {
        int u, v, w;
        cin >> u >> v >> w;
        g[u][v] = min(g[u][v], w);//可能会有重边
    }
    cout << dijkstra() << endl;
    return 0;
}
```

【堆优化版Dijkstra代码】

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

typedef pair<int, int> PII;
const int N = 150010, M = 150010;
int e[M], ne[M], d[M], h[N], idx;
int n, m, dis[N], vis[N];

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


```

```cpp
int dijkstra()
{
    memset(dis, 0x3f, sizeof dis);
    dis[1] = 0;
    priority_queue<PII, vector<PII>, greater<PII> > Q;
    Q.push({ 0, 1 });
    while (Q.size())
    {
        int t = Q.top().second;
        Q.pop();
        if (vis[t]) continue;
        vis[t] = 1;
        for (int i = h[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] });
    }
    return dis[n] == 0x3f3f3f3f ? -1 : dis[n];
}

int main()
{
    cin >> n >> m;
    memset(h, -1, sizeof h);
    while (m--)
    {
        int a, b, c;
        cin >> a >> b >> c;
        add(a, b, c);
    }
    cout << dijkstra() << endl;
    return 0;
}
```

搜索与图论-Bellman-Ford

一、AcWing 853. 有边数限制的最短路

【题目描述】

给定一个n个点m条边的有向图，图中可能存在重边和自环，**边权可能为负数**。

请你求出从1号点到n号点的最多经过k条边的最短距离，如果无法从1号点走到n号点，输出`impossible`。

注意：图中可能存在负权回路。

【输入格式】

第一行包含三个整数n, m, k。

接下来m行，每行包含三个整数x, y, z，表示存在一条从点x到点y的有向边，边长为z。

【输出格式】

输出一个整数，表示从1号点到n号点的最多经过k条边的最短距离。

如果不存在满足条件的路径，则输出`impossible`。

【数据范围】

$1 \leq n, k \leq 500$

$1 \leq m \leq 10000$

任意边长的绝对值不超过10000。

【输入样例】

```
3 3 1
1 2 1
2 3 1
1 3 3
```

【输出样例】

```
3
```

【代码】

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

const int N = 510, M = 10010;
int n, m, k;
int dis[N], backup[N];//backup用来备份dis数组防止出现串联问题

struct Edge
{
    int x, y, w;
```

```
}e[M];

void bellman_ford()
{
    memset(dis, 0x3f, sizeof dis);
    dis[1] = 0;
    //最多经过k条边的最短路
    for (int i = 0; i < k; i++)
    {
        memcpy(backup, dis, sizeof dis);//备份上一次的最短距离
        for (int j = 0; j < m; j++)//遍历每条边，更新最短距离
            dis[e[j].y] = min(dis[e[j].y], backup[e[j].x] + e[j].w);
    }
}

int main()
{
    cin >> n >> m >> k;
    for (int i = 0; i < m; i++)
        cin >> e[i].x >> e[i].y >> e[i].w;
    bellman_ford();
    dis[n] > 0x3f3f3f3f / 2 ? cout << "impossible\n" : cout << dis[n] <<
endl;
    return 0;
}
```

搜索与图论-SPFA

一、AcWing 851. spfa求最短路

【题目描述】

给定一个n个点m条边的有向图，图中可能存在重边和自环，**边权可能为负数**。

请你求出1号点到n号点的最短距离，如果无法从1号点走到n号点，则输出`impossible`。

数据保证**不存在负权回路**。

【输入格式】

第一行包含整数n和m。

接下来m行每行包含三个整数x, y, z，表示存在一条从点x到点y的有向边，边长为z。

【输出格式】

输出一个整数，表示1号点到n号点的最短距离。

如果路径不存在，则输出`impossible`。

【数据范围】

$1 \leq n, m \leq 10^5$

图中涉及边长绝对值均不超过10000。

【输入样例】

```
3 3
1 2 5
2 3 -3
1 3 4
```

【输出样例】

```
2
```

【代码】

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

const int N = 100010, M = 100010;
int e[M], ne[M], d[M], h[N], idx;
int dis[N], vis[N];//vis记录某个点是否在队列中
int n, m;

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

//spfa算法思想是只有当某个点的前驱结点最短距离更新了该点才可能被更新
void spfa()
{

    memset(dis, 0x3f, sizeof dis);
```

```
    dis[1] = 0;
    queue<int> Q;
    Q.push(1);
    vis[1] = 1;//标记已入队
    while (Q.size())
    {
        int t = Q.front();
        Q.pop();
        vis[t] = 0;//标记已出队
        for (int i = h[t]; ~i; i = ne[i])
            if (dis[t] + d[i] < dis[e[i]])
            {
                dis[e[i]] = dis[t] + d[i];
                if (!vis[e[i]])//如果该点不在队列中则将其入队
                    Q.push(e[i]), vis[e[i]] = 1;
            }
    }
}

int main()
{
    cin >> n >> m;
    memset(h, -1, sizeof h);
    while (m--)
    {
        int a, b, c;
        cin >> a >> b >> c;
        add(a, b, c);
    }
    spfa();
    dis[n] == 0x3f3f3f3f ? cout << "impossible\n" : cout << dis[n] << endl;
    return 0;
}
```

二、AcWing 852. spfa判断负环

【题目描述】

给定一个n个点m条边的有向图，图中可能存在重边和自环，**边权可能为负数**。

请你判断图中是否存在负权回路。

【输入格式】

第一行包含整数n和m。

接下来m行每行包含三个整数x, y, z，表示存在一条从点x到点y的有向边，边长为z。

【输出格式】

如果图中存在负权回路，则输出`Yes`，否则输出`No`。

【数据范围】

$1 \leq n \leq 2000$

$1 \leq m \leq 10000$

图中涉及边长绝对值均不超过10000。

【输入样例】

```
3 3
1 2 -1
2 3 4
3 1 -4
```

【输出样例】

```
Yes
```

【代码】

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

const int N = 100010, M = 100010;
int e[M], ne[M], d[M], h[N], idx;
int dis[N], vis[N], cnt[N];//cnt记录到某个点经过了多少条边
int n, m;

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

//spfa求负权环的思想是当到某个点所经过的边数>=n时则存在负权环
```

```cpp
bool spfa()
{
    //因为不确定从哪个点能走到负环因此初始化将所有点入队
    queue<int> Q;
    for (int i = 1; i <= n; i++) Q.push(i), vis[i] = 1;
    while (Q.size())
    {
        int t = Q.front();
        Q.pop();
        vis[t] = 0;//标记已出队
        for (int i = h[t]; ~i; i = ne[i])
            if (dis[t] + d[i] < dis[e[i]])
            {
                dis[e[i]] = dis[t] + d[i];
                cnt[e[i]] = cnt[t] + 1;
                if (cnt[e[i]] >= n) return true;//存在负环
                if (!vis[e[i]])//如果该点不在队列中则将其入队
                    Q.push(e[i]), vis[e[i]] = 1;
            }
    }
    return false;
}

int main()
{
    cin >> n >> m;
    memset(h, -1, sizeof h);
    while (m--)
    {
        int a, b, c;
        cin >> a >> b >> c;
        add(a, b, c);
    }
    spfa() ? puts("Yes") : puts("No");
    return 0;
}
```

搜索与图论-Floyd

一、AcWing 854. Floyd求最短路

【题目描述】

给定一个n个点m条边的有向图，图中可能存在重边和自环，**边权可能为负数**。

再给定k个询问，每个询问包含两个整数x和y，表示查询从点x到点y的最短距离，如果路径不存在，则输出`impossible`。

数据保证图中**不存在负权回路**。

【输入格式】

第一行包含三个整数n, m, k。

接下来m行，每行包含三个整数x, y, z，表示存在一条从点x到点y的有向边，边长为z。

接下来k行，每行包含两个整数x, y，表示询问点x到点y的最短距离。

【输出格式】

共k行，每行输出一个整数，表示询问的结果，若询问两点间不存在路径，则输出`impossible`。

【数据范围】

$1 \leq n \leq 200$

$1 \leq k \leq n^2$

$1 \leq m \leq 20000$

图中涉及边长绝对值均不超过10000。

【输入样例】

```
3 3 2
1 2 1
2 3 2
1 3 1
2 1
1 3
```

【输出样例】

```
impossible
1
```

【代码】

```
#include <iostream>
#include <algorithm>
```

```cpp
#include <cstring>
using namespace std;

const int N = 210;
const int INF = 0x3f3f3f3f;
int n, m, q;
int g[N][N];

void floyd()
{
    for (int k = 1; k <= n; k++)
        for (int i = 1; i <= n; i++)
            for (int j = 1; j <= n; j++)
                g[i][j] = min(g[i][j], g[i][k] + g[k][j]);
}

int main()
{
    cin >> n >> m >> q;
    memset(g, 0x3f, sizeof g);
    for (int i = 1; i <= n; i++) g[i][i] = 0;
    while (m--)
    {
        int a, b, c;
        cin >> a >> b >> c;
        g[a][b] = min(g[a][b], c);
    }
    floyd();
    while (q--)
    {
        int a, b;
        cin >> a >> b;
        if (g[a][b] > INF / 2) puts("impossible");
        else cout << g[a][b] << endl;
    }
    return 0;
}
```

搜索与图论-Prim

一、AcWing 858. Prim算法求最小生成树

【题目描述】

给定一个n个点m条边的无向图，图中可能存在重边和自环，**边权可能为负数**。

求最小生成树的树边权重之和，如果最小生成树不存在则输出`impossible`。

给定一张边带权的无向图$G=(V,E)$，其中V表示图中点的集合，E表示图中边的集合，$n=|V|,m=|E|$。

由V中的全部n个顶点和E中$n-1$条边构成的无向连通子图被称为G的一棵生成树，其中边的权值之和最小的生成树被称为无向图G的最小生成树。

【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u,v,w，表示点u和点v之间存在一条权值为w的边。

【输出格式】

共一行，若存在最小生成树，则输出一个整数，表示最小生成树的树边权重之和，如果最小生成树不存在则输出`impossible`。

【数据范围】

$1\leq n\leq 500$

$1\leq m\leq 10^5$

图中涉及边的边权的绝对值均不超过10000。

【输入样例】

```
4 5
1 2 1
1 3 2
1 4 3
2 3 2
3 4 4
```

【输出样例】

```
6
```

【代码】

```
#include <iostream>
#include <cstring>
```

```cpp
#include <algorithm>
using namespace std;

const int N = 510, INF = 0x3f3f3f3f;
int g[N][N];
int vis[N], dis[N];//dis表示每个点到集合的最小值
int n, m;

int prim()
{
    memset(dis, 0x3f, sizeof dis);
    int res = 0;
    for (int i = 0; i < n; i++)
    {
        int t = -1;
        for (int j = 1; j <= n; j++)
            if (!vis[j] && (t == -1 || dis[j] < dis[t]))
                t = j;
        vis[t] = 1;
        if (i && dis[t] == INF) return INF;
        if (i) res += dis[t];//可能存在负的自环更新dis[t]，因此需在更新前加入res
        for (int j = 1; j <= n; j++)
            dis[j] = min(dis[j], g[t][j]);
    }
    return res;
}

int main()
{
    cin >> n >> m;
    memset(g, 0x3f, sizeof g);
    while (m--)
    {
        int a, b, c;
        cin >> a >> b >> c;
        g[a][b] = g[b][a] = min(g[a][b], c);
    }
    int ans = prim();
    ans == INF ? cout << "impossible\n" : cout << ans << endl;
    return 0;
}
```

搜索与图论-Kruskal

一、AcWing 859. Kruskal算法求最小生成树

【题目描述】

给定一个n个点m条边的无向图，图中可能存在重边和自环，**边权可能为负数**。

求最小生成树的树边权重之和，如果最小生成树不存在则输出`impossible`。

给定一张边带权的无向图$G=(V,E)$，其中V表示图中点的集合，E表示图中边的集合，$n=|V|,m=|E|$。

由V中的全部n个顶点和E中$n-1$条边构成的无向连通子图被称为G的一棵生成树，其中边的权值之和最小的生成树被称为无向图G的最小生成树。

【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u,v,w，表示点u和点v之间存在一条权值为w的边。

【输出格式】

共一行，若存在最小生成树，则输出一个整数，表示最小生成树的树边权重之和，如果最小生成树不存在则输出`impossible`。

【数据范围】

$1\leq n\leq 10^5$

$1\leq m\leq 2\times 10^5$

图中涉及边的边权的绝对值均不超过1000。

【输入样例】

```
4 5
1 2 1
1 3 2
1 4 3
2 3 2
3 4 4
```

【输出样例】

```
6
```

【代码】

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

const int INF = 0x3f3f3f3f;
const int N = 100010, M = 200010;
int pre[N];
int n, m;

struct Edge
{
    int x, y, w;
    bool operator< (const Edge &t) const
    {
        return w < t.w;
    }
}e[M];

int find(int k)
{
    if (pre[k] == k) return k;
    return pre[k] = find(pre[k]);
}

//kruskal算法思想是每次选择权值最小的边，若该边所连的两个顶点不在一个连通块内则选取该边
int kruskal()
{
    sort(e, e + m);
    int res = 0, cnt = 0;//cnt记录已选择边的数量
    for (int i = 0; i < m; i++)
    {
        int px = find(e[i].x), py = find(e[i].y);
        if (px != py) pre[px] = py, res += e[i].w, cnt++;
        //最小生成树的边数为顶点数-1，已生成最小生成树
        if (cnt == n - 1) break;
    }

    if (cnt != n - 1) return INF;//不存在最小生成树
```

```
    return res;
}

int main()
{
    cin >> n >> m;
    //初始化并查集
    for (int i = 1; i <= n; i++) pre[i] = i;
    for (int i = 0; i < m; i++)
        cin >> e[i].x >> e[i].y >> e[i].w;
    int ans = kruskal();
    if (ans == INF) puts("impossible");
    else cout << ans << endl;
    return 0;
}
```

搜索与图论-染色法判定二分图

一、AcWing 860. 染色法判定二分图

【题目描述】

给定一个n个点m条边的无向图，图中可能存在重边和自环。

请你判断这个图是否是二分图。

【输入格式】

第一行包含两个整数n和m。

接下来m行，每行包含两个整数u和v，表示点u和点v之间存在一条边。

【输出格式】

如果给定图是二分图，则输出`Yes`，否则输出`No`。

【数据范围】

$1 \leq n, m \leq 10^5$

【输入样例】

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

【输出样例】

```
Yes
```

【代码】

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

const int N = 100010, M = 200010;
int e[M], ne[M], h[N], idx;
int color[N];//color为0表示未染色，1和2表示两种不同的颜色
int n, m;

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

bool dfs(int u, int c)
{
    color[u] = c;//将当前点染色
    for (int i = h[u]; ~i; i = ne[i])//遍历每条边
        if (color[e[i]] == c) return false;//若下一个点颜色和当前点一样则冲突
        else if (!color[e[i]])//如果下一个点未被染色
            if (!dfs(e[i], 3 - c)) return false;//3 - c可实现1和2的互换
    return true;
}

int main()
{
    cin >> n >> m;
    memset(h, -1, sizeof h);
    while (m--)
```

```
    {
        int a, b;
        cin >> a >> b;
        add(a, b), add(b, a);
    }
    bool flag = true;//标记是否为二分图
    //遍历每个点，检查是否染色，未染色则深搜进行染色
    for (int i = 1; i <= n; i++)
        if (!color[i])
            if (!dfs(i, 1))
            {
                flag = false;
                break;
            }
    if (flag) puts("Yes");
    else puts("No");
    return 0;
}
```

搜索与图论-匈牙利算法

一、AcWing 861. 二分图的最大匹配

【题目描述】

给定一个二分图，其中左半部包含n_1个点（编号$1 \sim n_1$），右半部包含n_2个点（编号$1 \sim n_2$），二分图共包含m条边。

数据保证任意一条边的两个端点都不可能在同一部分中。

请你求出二分图的最大匹配数。

二分图的匹配：给定一个二分图G，在G的一个子图M中，M的边集$\{E\}$中的任意两条边都不依附于同一个顶点，则称M是一个匹配。

二分图的最大匹配：所有匹配中包含边数最多的一组匹配被称为二分图的最大匹配，其边数即为最大匹配数。

【输入格式】

第一行包含三个整数n_1, n_2和m。

接下来m行，每行包含两个整数u和v，表示左半部点集中的点u和右半部点集中的点v之间存在一条边。

【输出格式】

输出一个整数，表示二分图的最大匹配数。

【数据范围】

$1 \leq n_1, n_2 \leq 500$

$1 \leq u \leq n_1$

$1 \leq v \leq n_2$

$1 \leq m \leq 10^5$

【输入样例】

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

【输出样例】

```
2
```

【代码】

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

const int N = 510, M = 100010;
int e[M], ne[M], h[N], idx;
int n1, n2, m;
int vis[N], match[N];//match表示女生所匹配的男生

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

bool find(int u)
{
    for (int i = h[u]; ~i; i = ne[i])
```

```
    {
        int j = e[i];
        if (!vis[j])//每个女生只考虑一次
        {
            vis[j] = 1;
            //如果该女生还没有对象或者可以为其对象找到下家
            if (!match[j] || find(match[j]))
            {
                match[j] = u;//匹配当前的男生
                return true;
            }
        }
    }
    return false;
}

int main()
{
    cin >> n1 >> n2 >> m;
    memset(h, -1, sizeof h);
    while (m--)
    {
        int a, b;
        cin >> a >> b;
        add(a, b);
    }
    int res = 0;
    for (int i = 1; i <= n1; i++)
    {
        memset(vis, 0, sizeof vis);
        if (find(i)) res++;
    }
    cout << res << endl;
    return 0;
}
```