DFS

1.数字全排列

题面

给定一个整数 n，将数字 1∼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

$1 \le n \le 7$

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
```

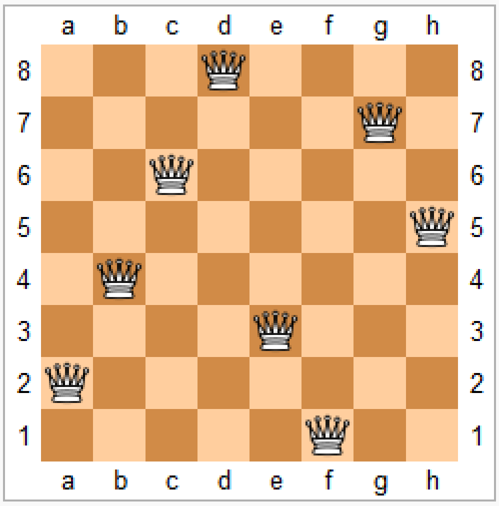
代码实现

```python
def perm(li):
    from itertools import permutations
    for i in permutations(li):
        print(*i)
perm([i for i in range(1,int(input())+1)])
```

2.n皇后问题

题面

n−皇后问题是指将 n 个皇后放在 $n×n$ 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。



One solution to the eight queens puzzle

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

$1≤n≤9$

输入样例

```
4
```

输出样例

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

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

代码实现

```python
def nqueue(n):
    li = [i for i in range(1, n + 1)]

    def check(i):
        for j in range(len(i) - 1):
            for k in range(1, len(i) - j):
                if abs(i[j + k] - i[j]) == k:
                    return False
        return True

    def fprint(i):
        for j in i:
            temp = ['.' for k in range(len(li))]
            temp[j - 1] = 'Q'
            print(''.join(temp))


    from itertools import permutations
    for i in permutations(li):
        if check(i):
            print(i)
            fprint(i)
            print()
nqueue(int(input()))
```

BFS

1.走迷宫

题面

给定一个 $n×m$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

$1 \le n, m \le 100$

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
```

代码实现

```python
def maze_solve(n, m, maze):
    q = [[0, 0, 0]]
    v = [[1 for i in range(m)] for j in range(n)]
    v[0][0] = 0
    while True:
        cur = q.pop(0)
        curx, cury, curd = cur[0], cur[1], cur[2]
        if [curx, cury] == [n - 1, m - 1]:
            print(curd)
            return

        for i, j in [[1, 0], [0, 1], [-1, 0], [0, -1]]:
            x, y, d = curx + i, cury + j, curd + 1
            if 0 <= x < n and 0 <= y < m and v[x][y] and maze[x][y] == 0:
                v[x][y] = 0
                q.append([x, y, d])


n, m = map(int, input().split())
maze = []
for i in range(n):
    maze.append(list(map(int, input().split())))
maze_solve(n, m, maze)
```

2.八数码

题面

在一个 3×3 的网格中，1~8 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
```

代码实现

```python
def eig_segments(li):
    from collections import deque
    d = set()
    q = deque([[li, 0]])
    d.add(''.join(li))
    while q:
        cur, step = q.popleft()
        if cur == ['1', '2', '3', '4', '5', '6', '7', '8', 'x']:
            print(step)
            return
        id = cur.index('x')
        x, y = id // 3, id % 3
        for i, j in [[-1, 0], [0, -1], [1, 0], [0, 1]]:
            nx, ny = x + i, y + j
            nid = nx * 3 + ny
            nstep = step + 1
            temp = cur[:]
            if 0 <= nx < 3 and 0 <= ny < 3:
                temp[id], temp[nid] = temp[nid], temp[id]
                string = ''.join(temp)
                if string not in d:
                    q.append([temp, nstep])
                    d.add(string)

    print(-1)
    return
```

树与图的深度优先遍历

1.树的重心

题面

给定一颗树，树中包含 n 个结点（编号 1~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

$1 \le n \le 10^5$

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
```

代码实现

```python
def tree_dfs(n, li):
    tree = [[] for i in range(n + 1)]
    for i, j in li:
        tree[i].append(j)
        tree[j].append(i)
    weight = [0 for i in range(n + 1)]  # 存储所有节点的权重
    v = [1 for i in range(n + 1)]
    child = [[] for i in range(n + 1)]  # 存储子节点

    def _dfs(root):  # root子树总共多少节点
        v[root] = 0
        if tree[root] == []:
            weight[root] = 1
            return 1
        ans = 1  # 本身
        for i in tree[root]:
            if v[i]:
                child[root].append(i)
                ans += _dfs(i)
        weight[root] = ans
        return ans

    _dfs(1)
    # print(weight, child)
    ans = 0xfffffff
    for i in range(1, n + 1):
        ans = min(ans, max(n - weight[i], max([weight[j] for j in child[i]] + [0])))
```

```python
    print(ans)
    return


n = int(input())
li = []
for i in range(n - 1):
    a, b = map(int, input().split())
    li.append([a, b])
tree_dfs(n, li)
```

树与图的的广度优先遍历

1.图中点的层次

题面

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~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 mm。

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

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

输入样例

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

输出样例

```
1
```

代码实现

```python
"""
dijkstra 的实现
"""
def graph(n, m, g):
    d = [0xfffffff for i in range(n + 1)]
    s = set([1])
    d[1] = 0
    from queue import PriorityQueue as pq
    q = pq()
    q.put([0, 1])
    while not q.empty():
        dis, point = q.get()
        for to, weight in g[point]:
            if d[to] > d[point] + weight and to not in s:
                q.put([d[point], to])
                d[to] = d[point] + weight
        s.add(to)
    print(d[n] if d[n] != 0xfffff else -1)


n, m = map(int, input().split())
g = [[] for i in range(n + 1)]
for i in range(m):
    a, b = map(int, input().split())
    g[a].append([b, 1])

graph(n, m, g)
```

```python
"""
BFS 的实现
"""
from collections import deque


def main():
    n, m = map(int, input().split())
    p = [set() for i in range(n + 1)]
    for i in range(m):
        a, b = map(int, input().split())
        p[a].add(b)


    q = deque([[1, 0]])
    vis = [True for i in range(n + 1)]
    while q:
        cur, step = q.popleft()
        if cur == n:
            print(step)
            return
        vis[cur] = False
        for i in p[cur]:
            if vis[i]:
```

```
                    q.append([i, step + 1])

    print(-1)
    return


if __name__ == '__main__':
    main()

```

$dijkstra = bfs + gready(+dp)$

拓扑排序

1.有向图的拓扑序列

题面

给定一个 n个点 m 条边的有向图，点的编号是 1到 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

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

输入样例

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

输出样例

```
1 2 3
```

代码实现

```
def topology(n, m, g):
    rudu = [0 for i in range(n + 1)]
    for i in range(1, n + 1):
        for j in g[i]:
            rudu[j] += 1
    q, topo_sequence = [], []
    for i in range(1, n + 1):
        if rudu[i] == 0:
            q.append(i)
    while q:
        cur = q.pop(0)
        topo_sequence.append(cur)
        for i in g[cur]:
            rudu[i] -= 1
            if rudu[i] == 0:
                q.append(i)
    if len(topo_sequence) < n:
        print(-1)
        return
    print(*topo_sequence)
    return


n, m = map(int, input().split())
g = [[] for i in range(n + 1)]
for i in range(m):
    a, b = map(int, input().split())
    g[a].append(b)
topology(n, m, g)
```

Dijkstra

1.Dijkstra求最短路 I

题面

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。

数据范围

1≤n≤500
1≤m≤10^5
图中涉及边长均不超过10000。

输入样例

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

输出样例

```
3
```

代码实现

```python
def dijkstra(n, m, g):
    inf = 0xffffffff
    d = [inf for i in range(n + 1)]
    start, end = 1, n
    s = set([start])
    d[start] = 0
    from queue import PriorityQueue
    q = PriorityQueue()
    q.put([0, start])

    while not q.empty():
        dis, cur = q.get()
        if cur in s:continue
        for to, weig in g[cur]:
            if d[to] > d[cur] + weig:
                d[to] = d[cur] + weig
                q.put([d[to], to])
        # 届时，cur的状态已经确定，将其标记在set里面
        s.add(cur)

    if d[end] != inf:
        print(d[end])
    else:
        print(-1)
    return


n, m = map(int, input().split())
g = [[] for i in range(n + 1)]
for i in range(m):
    a, b, c = map(int, input().split())
    g[a].append([b, c])
dijkstra(n, m, g)
```

2.Dijkstra求最短路II

题面

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，所有边权均为非负值。

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。

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
```

代码实现

```python
def dijkstraII(g, start, end, inf=0xfffffff):
    from queue import PriorityQueue
    d = [inf for i in range(n + 1)]
    d[start] = 0
    pq = PriorityQueue()
    s = set()  # point whose statues have been known
    pq.put([0, start])
    while not pq.empty():
        distance, point = pq.get()
        if point in s:
            continue
        for to, weight in g[point]:
            if d[to] > d[point] + weight:
```

```python
                    d[to] = d[point] + weight
                    pq.put([d[to], to])

            s.add(point)
    print(d[end] if d[end] != inf else -1)
    return


n, m = map(int, input().split())
g = [[] for i in range(n + 1)]
for i in range(m):
    a, b, c = map(int, input().split())
    g[a].append([b, c])
dijkstraII(g, 1, n)
```

> 注意: 不要用pq = PriorityQueue([0 ,start])，另外，尽可能使用empty()方法对PriorityQueue判空。

Bellman-Ford算法

1.有边数限制的最短路

题面

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。

点的编号为 $1 \sim n$。

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

$1 \le n,k \le 500$
$1 \le m \le 10000$
$1 \le x,y \le n$
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
```

代码实现

```python
def Bellman_Ford(n, m, k, edges):
    inf = 0x7fffffff
    d = [inf for i in range(n + 1)]
    d[1] = 0
    for i in range(k):
        temp = d[:]
        for FROM, TO, COST in edges:
            d[TO] = min(d[TO], temp[FROM] + COST)
    print(d[n] if d[n] < inf / 2 else 'impossible')
    return


n, m, k = map(int, input().split())
edges = []
for i in range(m):
    a, b, c = map(int, input().split())
    edges.append([a, b, c])
Bellman_Ford(n, m, k, edges)
```

SPFA算法

1.spfa求最短路

题面

> 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

$1 \le n, m \le 10^5$,
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
```

代码实现

```python
def SPFA_shortest(n, m, g):
    inf = 0x7fffffff
    d = [inf for i in range(n + 1)]
    from collections import deque
    start, end = 1, n
    q = deque([start])
    s = set([start])
    d[start] = 0
    while q:
        cur = q.popleft()
        s.remove(cur)
        for to, weight in g[cur]:
            if d[to] > d[cur] + weight:
                d[to] = d[cur] + weight
                if to not in s:
                    q.append(to)
                    s.add(to)
    print(d[end] if d[end] != inf else 'impossible')
    return


n, m = map(int, input().split())
g = [[] for i in range(n + 1)]
for i in range(m):
    a, b, c = map(int, input().split())
    g[a].append([b, c])

SPFA_shortest(n, m, g)
```

2.spfa判断负环

题面

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

$1 \le n \le 2000$
$1 \le m \le 10000$
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
```

代码实现

```python
from collections import deque

n, m = map(int, input().split())
p = [[] for i in range(n + 1)]

for i in range(m):
    a, b, c = map(int, input().split())
    p[a].append([b, c])


def spfa_negative_circle(p, start, en, inf=0x7fffffff):
    dis = [inf for i in range(len(p))]
    counter = [0 for i in range(len(p))]
    dis[start] = 0
    q = deque([i for i in range(1, n + 1)])
    s = set(q)
    while q:
        cur = q.popleft()
        s.remove(cur)
        for to, wei in p[cur]:
```

```python
                if dis[to] > dis[cur] + wei:
                    dis[to] = dis[cur] + wei
                    counter[to] = counter[cur] + 1
                    if counter[to] > n:
                        return 'Yes'
                    if to not in s:
                        q.append(to)
                        s.add(to)
    return "No"


print(spfa_negative_circle(p, 1, n))
```

> 注意: python里的float('inf')不可用于做具体数值的加减法，因为inf+2=inf-2，代数上是错误的。

Floyd算法

1.Floyd求最短路

题面

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

$1 \le n \le 200$
$1 \le k \le n^2$
$1 \le m \le 20000$
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
```

代码实现

```python
def Floyd(m, n, g):
    inf = 0x7fffffff
    d = [[inf for i in range(n + 1)] for i in range(n + 1)]
    for i in range(1, n + 1):
        d[i][i] = 0
        for j, v in g[i]:
            d[i][j] = min(d[i][j], v)

    for k in range(1, n + 1):
        for i in range(1, n + 1):
            for j in range(1, n + 1):
                d[i][j] = min(d[i][j], d[i][k] + d[k][j])
    return d


n, m, k = map(int, input().split())
g = [[] for i in range(n + 1)]
for i in range(m):
    a, b, c = map(int, input().split())
    g[a].append([b, c])
dis = Floyd(n, m, g)
for i in range(k):
    a, b = map(int, input().split())
    print(dis[a][b] if dis[a][b] != 0x7fffffff else 'impossible')
```

Prime最小生成树

1.Prim算法求最小生成树

题面

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

$1 \le n \le 500$,
$1 \le m \le 10^5$,
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
```

代码实现

```python
def Prime(n, m, g):
    from queue import PriorityQueue
    inf = 0xfffffff
    p = [[inf for i in range(n + 1)] for i in range(n + 1)]
    for i in range(1, n + 1):
        p[i][i] = 0
        for j, k in g[i]:
            p[i][j] = min(p[i][j], k)
            p[j][i] = min(p[j][i], k)

    dis = [inf for i in range(n + 1)]  # i到集合的最短距离
    s = set([])
    pq = PriorityQueue()
    ans = 0
    for i in range(1, n + 1):  # 总共要加入n个点
        # 找到马上要新加入几何的点，要求这个点不在集合中并且离集合最近
        closet_point = min(range(1, n + 1), key=lambda x: inf * inf * int(x in s) + dis[x])
        if i != 1:  # 排除第一次，第一个点离空集距离是正无穷的
```

```python
            if dis[closet_point] == inf:  # 如果最近的点没有通路，说明构造不出最小生成树
                print("impossible")
                return
            ans += dis[closet_point]
        # 用新加入的点，也就是closet_point更新其余的各个点到集合s的最小距离，此处宜思考
        for j in range(1, n + 1):
            dis[j] = min(dis[j], p[closet_point][j])
        s.add(closet_point)  # 把这个点加入到集合内部

    print(ans)
    return


n, m = map(int, input().split())
g = [[] for i in range(n + 1)]
for i in range(m):
    a, b, c = map(int, input().split())
    g[a].append([b, c])
Prime(n, m, g)
```

Kruskal算法

1.Kruskal 算法求最小生成树

题面

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 EE 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

$1 \le n \le 10^5$,
$1 \le m \le 2*10^5$
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
```

代码实现

```python
def Kruskal(n, m, edges):
    s = [i for i in range(n + 1)]

    def _fa(a):
        if s[a] == a:
            return a
        s[a] = _fa(s[a])
        return s[a]

    def _h(a):
        return 0 if s[a] == a else _h(s[a]) + 1

    def _union(a, b):
        fa = _fa(a)
        fb = _fa(b)
        if _h(a) < _h(b):
            s[fa] = fb
        else:
            s[fb] = fa

    def _common(a, b):
        return _fa(a) == _fa(b)

    def _cluster():
        for i in range(1, n + 1):
            s[i] = _fa(i)
        return len(set(s)) - 1


    edges.sort(key=lambda x: x[2])
    a, b, ans = edges.pop(0)
    _union(a, b)
    for i, j, k in edges:
        if not _common(i, j):
            ans += k
            _union(i, j)

    if _cluster() == 1:
        print(ans)
```

```python
    else:
        print("impossible")
    return


n, m = map(int, input().split())
edges = []
for i in range(m):
    a, b, c = map(int, input().split())
    edges.append([a, b, c])

Kruskal(n, m, edges)
```

匈牙利算法

1.二分图的最大匹配

题面

给定一个二分图，其中左半部包含 n_1个点（编号 1~n_1），右半部包含n_2个点（编号 1~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*1n1、 *n*2n2 和 *m*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

$1 \le n_1, n_2 \le 500$
$1 \le u \le n_1$
$1 \le v \le n_2$
$1 \le m \le 10^5$

输入样例

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

输出样例:

```
2
```

代码实现

```python
def Hungary(n, m, li):
    boy = [0 for i in range(n + 1)]
    girl = [0 for i in range(m + 1)]

    def _match(n):
        for i in li[n]:
            if girl[i] == 0:
                girl[i] = n
                return 1
            else:
                j = girl[i]
                k = 0
                while k < len(li[j]) and girl[li[j][k]]:
                    k += 1
                if k < len(li[j]):
                    girl[li[j][k]] = j
                    girl[i] = n
                    return 1
        return 0

    ans = 0
    for i in range(1, n + 1):
        ans += _match(i)
    print(ans)
    return ans


n, m, k = map(int, input().split())
edges = [[] for i in range(n + 1)]
for i in range(k):
    a, b = map(int, input().split())
    edges[a].append(b)

Hungary(n, m, edges)
```