实验舱
青少年编程
走近科学 走进名校

提高算法班

并查集、最小生成树

Mas

并查集

并查集 (Disjoint – Set) 是一种用于管理元素所属集合的数据结构

实现为一个森林,其中每棵树表示一个集合,树中的节点表示对应集合中的元素

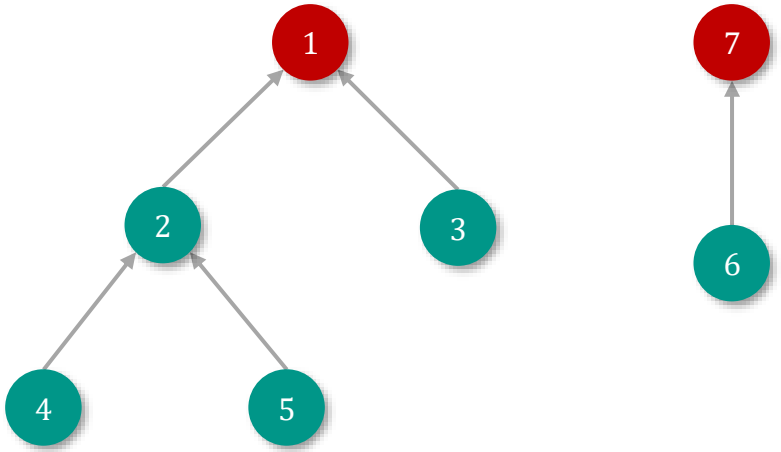
并查集支持两种操作

合并：把两个不相交的集合合并为一个集合

查询：查询两个元素是否在同一个集合中

并查集的重要思想在于 **用集合中的一个元素代表集合**

并查集可以用于判断两个元素是否存在直接或间接的联系、连通块计数等问题



并查集-路径压缩

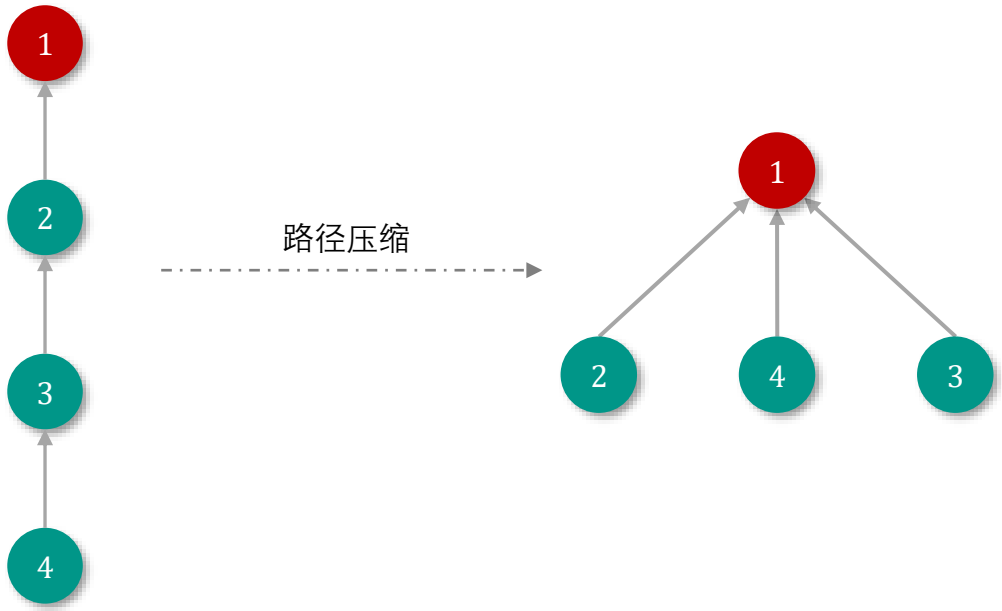
随着合并的元素越来越多

集合会形成一条长链,每次查询层数会越来越多

对于每个元素仅关心其集合代表的编号,并不关心它的上层节点编号

在查询的过程中, 将沿途的每个节点的父节点都设为根节点

下一次再查询时效率就会大大提高



并查集-启发式合并

当需要将两个集合并时,无论将哪一个集合连接到另一个集合的下面,都能得到正确的结果

但不同的连接方法存在时间复杂度的差异

若将一棵点数与深度都较小的树合并到一棵更大的树下

显然比另一种方案更优

鉴于点数与深度这两个特征都很容易维护,常常从中择一作为估价函数

而无论选择哪一个,时间复杂度都为 $O\left(m\,\alpha(m,n)\right)$

在 **Tarjan** 的论文中,证明了不使用启发式合并、只使用路径压缩的 **最坏** 时间复杂度是 $O\left(m\log n\right)$

在 **姚期智** 的论文中,证明了不使用启发式合并、只使用路径压缩 , 在 **平均** 情况下时间复杂度依然是 $O\left(m\,\alpha(m,n)\right)$

带权并查集

普通的并查集仅记录集合代表的编号

带权并查集不仅记录集合的关系

同时还记录着集合内元素的关系或者说是集合内元素连接线的权值

普通并查集本质是不带权图,带权并查集是带权图

每个节点都记录的是与根节点之间的权值

在 find 的路径压缩过程中权值也应该做相应的更新

在两个并查集做合并时,权值也要做相应的更新

#357、还是并查集

题目描述

有编号为 $1 \sim n$ 的 n 个方块正放在地上,每个构成一个立方柱

给出 P 个指令

指令有两种:

- 移动(`M`) : 将包含 X 的立方柱移动到包含 Y 的立方柱上
- 统计(`C`) : 统计含 X 的立方柱中,在 X 下方的方块数目

写个程序帮 Mas 完成游戏

输入格式

第 1 行输入 P

接下来 P 行每行输入一条指令,形式为 `M X Y` 或者 `C X`

输入保证不会有将立方柱放在自己头上的指令

输出格式

输出共 P 行,对于每个统计指令,输出其结果

输入样例

```
6
M 1 6
C 1
M 2 4
M 2 6
C 3
C 4
```

输出样例

```
1
0
2
```

数据规模

对于全部的数据 $1 \leq P \leq 10^5, 1 \leq n \leq 30000$

#357、还是并查集

令 cnt_i 表示 i 所在集合连通块大小, dis_i 表示 i 上方的木块数量

并查集维护木块信息

查询集合代表的编号时, 先递归更新 父节点 dis

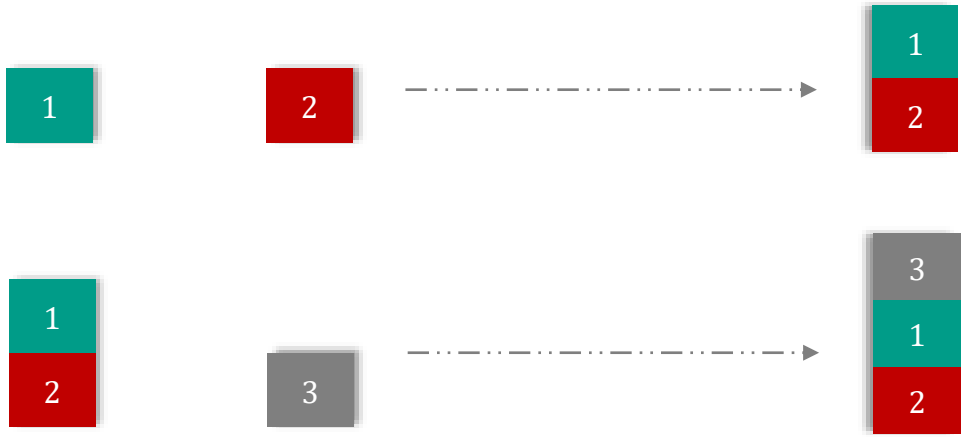
当父节点 dis 被更新后,再累加父节点 dis , 最后做路径压缩

在将 b 合并到 a 上时

先将 dis_{f_b} += cnt_{f_a} 再将 cnt_{f_a} += cnt_{f_b}

最后再修改 b 所在集合代表

对于 a 下方木块数量仅需要输出 $\text{cnt}_{f_a} - \text{dis}_a - 1$



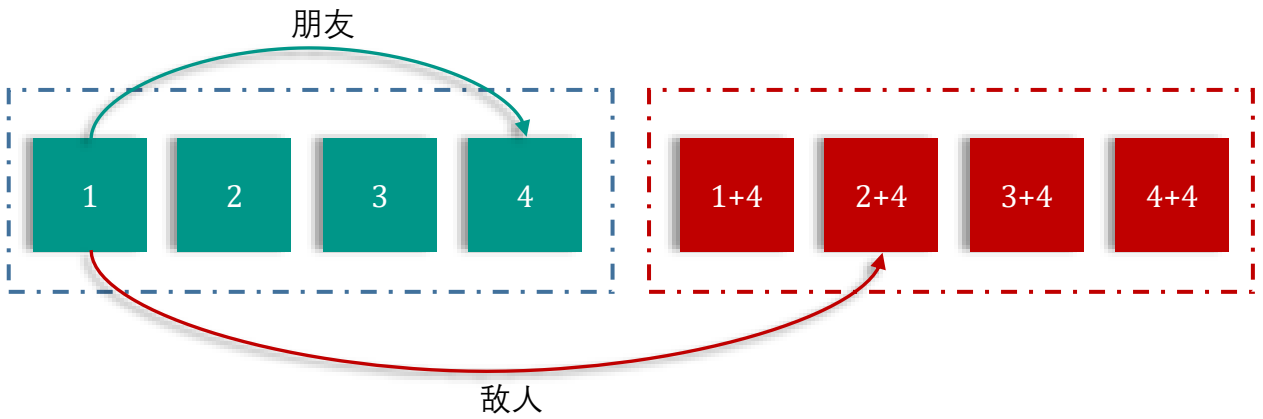

```
int find(int x)
{
  if (f[x] == x)
    return x;
  int fa = find(f[x]); //递归更新父
  dis[x] += dis[f[x]]; //累加距离
  return f[x] = fa;
}
void merge(int a, int b)
{
  int fa = find(a), fb = find(b);
  if (fa == fb)
    return;
  dis[fb] += cnt[fa];
  cnt[fa] += cnt[fb];
  f[fb] = fa;
}
```

种类并查集

普通的并查集维护的关系是： 朋友的朋友是朋友,即如果 $A \sim B$ 是一对朋友, $B \sim C$ 是一对朋友, 那么 $A \sim C$ 是一对朋友

但如果我们需要维护这样一个关系 “朋友的朋友是朋友 , 朋友的敌人是敌人,敌人的敌人是朋友” , 普通的并查集就无能为力了

引入种类并查集



准备两倍空间 , $1 \sim n$ 范围内的点与 $n+1 \sim 2n$ 存在映射关系, $x \rightarrow x+n$,其中 $x + n$ 为 敌对关系的对应点

如果 $1,2$ 是 朋友关系 , 将 1 与 2 之间连边

如果 $1,3$ 是 敌人关系 , 将 1 与 $3+4$ 之间连边 和 3 与 $1+4$ 连边

#377、关押罪犯

题目描述

S 城现有两座监狱,一共关押着 N 名罪犯,编号分别为 $1 \sim N$.

他们之间的关系自然也极不和谐,很多罪犯之间甚至积怨已久,如果客观条件具备则随时可能爆发冲突

我们用“怨气值”(一个正整数值)来表示某两名罪犯之间的仇恨程度,怨气值越大,则这两名罪犯之间的积怨越多

如果两名怨气值为 c 的罪犯被关押在同一监狱,他们俩之间会发生摩擦,并造成影响力为 c 的冲突事件

每年年末,警察局会将本年内监狱中的所有冲突事件按影响力从大到小排成一个列表,然后上报到 S 城 Z 市长那里

公务繁忙的 Z 市长只会去看列表中的第一个事件的影响力,如果影响很坏,他就会考虑撤换警察局长

在详细考察了 N 名罪犯间的矛盾关系后,警察局长觉得压力巨大

他准备将罪犯们在两座监狱内重新分配,以求产生的冲突事件影响力都较小,从而保住自己的乌纱帽

假设只要处于同一监狱内的某两个罪犯间有仇恨,那么他们一定会在每年的某个时候发生摩擦

那么,应如何分配罪犯,才能使 Z 市长看到的那个冲突事件的影响力最小?这个最小值是多少?

数据范围

对于 30% 的数据有 $N \leq 15$

对于 70% 的数据有 $N \leq 2000, M \leq 50000$

对于 100% 的数据有 $N \leq 20000, M \leq 100000$

输入格式

每行中两个数之间用一个空格隔开。第一行为两个正整数 N, M ,分别表示罪犯的数目以及存在仇恨的罪犯对数

接下来的 M 行每行为三个正整数 a_j, b_j, c_j ,表示 a_j 号和 b_j 号罪犯之间存在仇恨,其怨气值为 c_j

数据保证 $1 < a_j \leq b_j \leq N, 0 < c_j \leq 10^9$,且每对罪犯组合只出现一次

输出格式

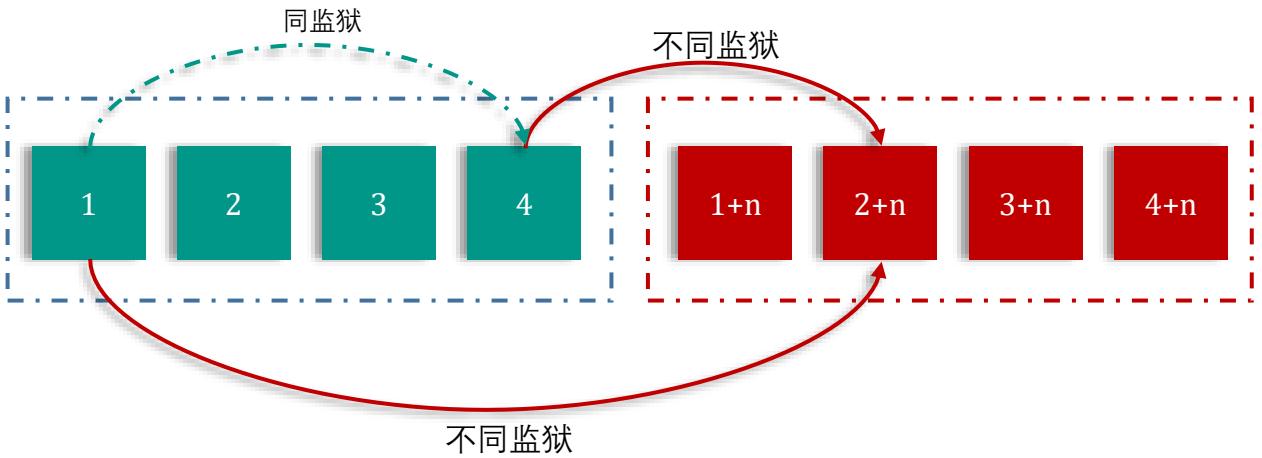
共 1 行,为 Z 市长看到的那个冲突事件的影响力

如果本年内监狱中未发生任何冲突事件,请输出 `0`

#377、关押罪犯

思路 1

尽可能将怨气值大的分配到两个不同的监狱,若不能分配时为答案



当 u_1 与 v 不同监狱, u_2 与 v 不同监狱, 那么 u_1 与 u_2 在同一监狱

按边权降序排序 , 从前往后遍历所有边：若 u, v 需要分配到不同的监狱， $u, v+n$ 以及 $v, u+n$连边

若发现 u, v 已在同一个监狱,说明此时冲突,答案即为该怨气值

#377、关押罪犯

思路 2

令 g_u 表示目前 u 和它所在的并查集的根 f_u 的关系

- 0 表示必须在同监狱
- 1 表示必须在不同监狱

每次把 f_u 合并至 f_v 时可通过异或操作维护出 g_{f_u} 的值，路径压缩时需同步更新 g

思路 3

题目希望一条边的两个端点不在同一集合,若全部边都能处于两个集合，该图为二分图 答案为0

为了让答案最小，考虑删去怨气值不超过 x 的所有边

删后若能否构成二分图，答案为 x

二分答案枚举怨气值 x 即可

不需要重新建图，二分图判定时走超过 x 的边即可

#1255、食物链

题目描述

动物王国中有三类动物 A,B,C ,这三类动物的食物链构成了有趣的环形

A 吃 B , B 吃 C , C 吃 A .

现有 N 个动物,以 $1\sim N$ 编号

每个动物都是 A,B,C 中的一种,但是我们并不知道它到底是哪一种

有人用两种说法对这 N 个动物所构成的食物链关系进行描述:

- 第一种说法是 `1 X Y` ,表示 X 和 Y 是同类
- 第二种说法是 `2 X Y` ,表示 X 吃 Y

此人对 N 个动物,用上述两种说法,一句接一句地说出 K 句话,这 K 句话有的是真的,有的是假的

当一句话满足下列三条之一时,这句话就是假话,否则就是真话

- 当前的话与前面的某些真的话冲突,就是假话
- 当前的话中 X 或 Y 比 N 大,就是假话
- 当前的话表示 X 吃 X ,就是假话

你的任务是根据给定的 N 和 K 句话,输出假话的总数

输入格式

第一行是两个整数 N 和 K ,以一个空格分隔

以下 K 行每行是三个正整数 D,X,Y ,两数之间用一个空格隔开,其中 D 表示说法的种类

- 若 $D=1$,则表示 X 和 Y 是同类
- 若 $D=2$,则表示 X 吃 Y

输出格式

只有一个整数,表示假话的数目

数据规模

对于全部的数据 $1\leq N\leq 50000,0\leq K\leq 100000$

#1255、食物链

思路 1

将各个动物 x 拆成三个节点：

- 同类域 x_{self}
- 捕食域 x_{eat}
- 天敌域 x_{enemy}

若 x 和 y 是同类说明

- x 的同类与 y 一样
- x 捕食的物种 与 y 捕食的物种一样
- x 的天敌 与 y 的天敌一样

需要合并 x_{self} 与 y_{self}、 x_{eat} 与 y_{eat}、 x_{enemy} 与 y_{enemy}

#1255、食物链

若 x 捕食 y 说明

- x 捕食的物种与 y 是同类
- x 同类都是 y 的天敌
- 由于仅存在长度为 3 的环 那么 y 捕食的物种都是 x 的天敌

需合并 x_{eat} 与 y_{self} 、 x_{self} 与 y_{enemy} 、 x_{enemy} 与 y_{eat}

令

- 节点 $1 \sim n$ 表示 同类域
- 节点 $n+1 \sim 2n$ 表示 捕食域
- 节点 $2n+1 \sim 3n$ 表示 天敌域

对于每句话处理之前需要检查这句话的真假

```
if (op == 1)
{
  if (find(x) == find(y + n) || find(x + n) == find(y))
    ans++;
  else
    merge(x, y), merge(x + n, y + n), merge(x + 2 * n, y + 2 * n);
}
else
{
  if (find(x) == find(y) || find(x) == find(y + n))
    ans++;
  else
    merge(x + n, y), merge(x, y + 2 * n), merge(x + 2 * n, y + n);
}
```

#1255、食物链

若 x 和 y 是同类，下列情况说明该句话为假

- x_{self} 与 y_{eat} 在同一集合
- x_{eat} 与 y_{self} 在同一集合

若 x 和 y 是天敌，下列情况说明该句话为假

- x_{self} 与 y_{self} 在同一集合
- x_{self} 与 y_{eat} 在同一集合

思路 2

也可用带权并查集做，权值拓展成 0，1，2

令 g 维护点 x 和 f_x 之间的距离差值

与 #357 类似 将 g 正确维护，判断类别时对 3 取模即可

Minimum Spanning Tree

子图 (Subgraph)

对于图 $G=(V,E)$，若存在 $H=(V',E')$ 满足 $V' \subseteq V$ 且 $E' \subseteq E$，则称 H 是 的 G **子图**，记作 $H \subseteq G$

生成子图/支撑子图 (Spanning Subgraph)

若 $H \subseteq G$ 满足 $V' = V$，则称 H为 G 的 **生成子图/支撑子图**

生成树 (Spanning Tree)

一个连通无向图的生成子图，同时要求是树，即在图的边集中选择 $n-1$ 条,将所有顶点连通

最小生成树 (Minimum Spanning Tree, MST)

无向连通图的 **最小生成树** 为边权和最小的生成树

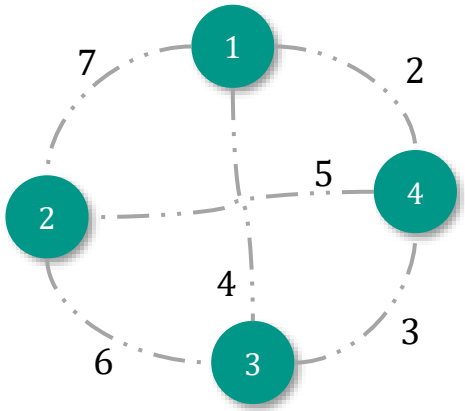
只有连通图才有生成树，对于非连通图只存在生成森林

Kruskal算法

从最小边权的边开始,按边权从小到大依次加入

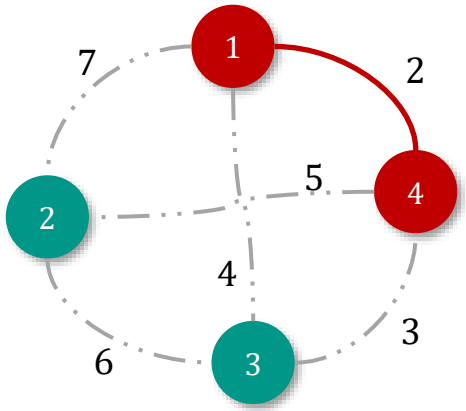
若某次加边产生了环，忽略这条边

直到加入了 $n-1$ 条边,即形成了一棵树，若不考虑连通性的维护时间复杂度 $O(m\log m)$



S = { { 1 } , { 2 } , { 3 } , { 4 } }

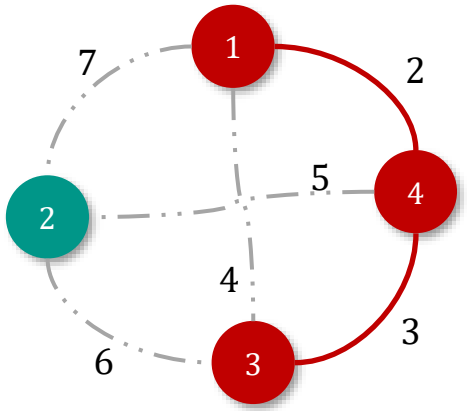
V = {}



选取未选最小权值边 <1 , 4>

S = { { 1 , 4 } , { 2 } , { 3 } }

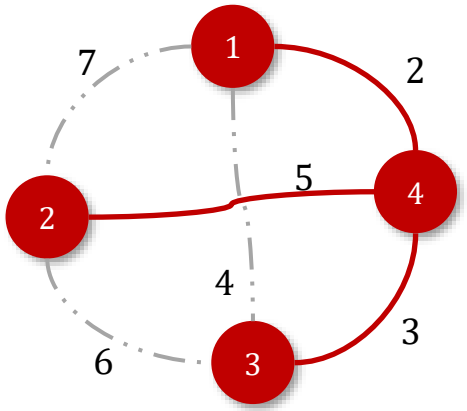
V = { <1 , 4> }



选取未选最小权值边 <4 , 3>

S = { { 1 , 4 , 3 } , { 2 } }

V = { <1 , 4> ,<4 , 3>}



选取未选最小权值边 <1 , 3>

发现在同一连通块内

再次选取 <2,4>

S = { { 1 , 4 , 3 , 2 } }

V = { <1 , 4> ,<4 , 3> , <2 , 4>}

Kruskal

命题

对于图 $G=(V,E)$ 当 $|V|\in\mathbb{N}^+$ 时 Kruskal 算法能找到一颗最小生成树

证明

当 $|V|=2$ 时，Kruskal 算法将找到最小边，此时命题显然成立

设 $|V|=n$ 时命题成立，考虑 $|V|=n+1$ 时

对于 G 中边权最小边 $e=(u,v)$，将 u,v 视为一个点，此时得到 n 个点的新图 G'

根据假设 G' 可由 Kruskal 算法得出最小生成树 T'，即 $T=T'\cup\{e\}$，T 为 G 的最小生成树

若不然，存在 包含 e 的最小生成树 T^*（若 $e\notin T^*$ 在 T^* 中加入 e 再去掉回路中任意边所得生成树更小）

即

$$W(T^*\setminus\{e\})=W(T^*)-W(e)<W(T\setminus\{e\})=W(T')$$

与假设矛盾

命题得证

#931、最小生成树kruskal

题目描述

给定结点数为 n ,边数为 m 的带权无向连通图 G ,所有结点编号为 $1 \sim n$

求 G 的最小生成树的边权和

输入格式

第一行两个正整数 n, m

之后的 m 行,每行三个正整数 u_i, v_i, w_i ,描述一条连接结点 u_i 和 v_i ,边权为 w_i 的边

输出格式

一个整数表示 G 的最小生成树的边权和

数据范围与提示

对于全部的数据 $1 \leq n \leq 2 \times 10^5$, $0 \leq m \leq 5 \times 10^5$

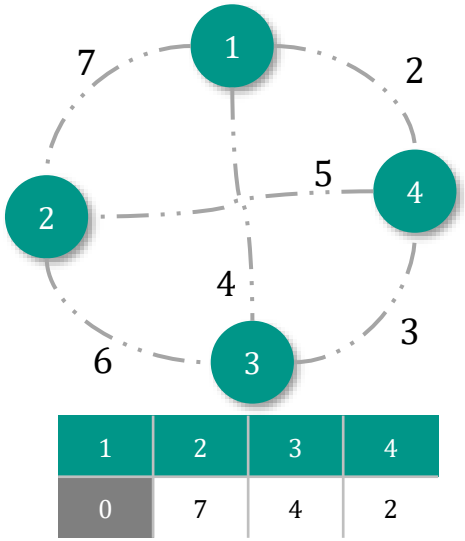
prim

从任意一个结点开始,将结点分成两类：已加入的,未加入的

每次从未加入的结点中,找一个与已加入的结点之间边权最小值最小的结点

然后将这个结点加入,并连上那条边权最小的边，重复 $n-1$ 次,时间复杂度 $O(n^2)$

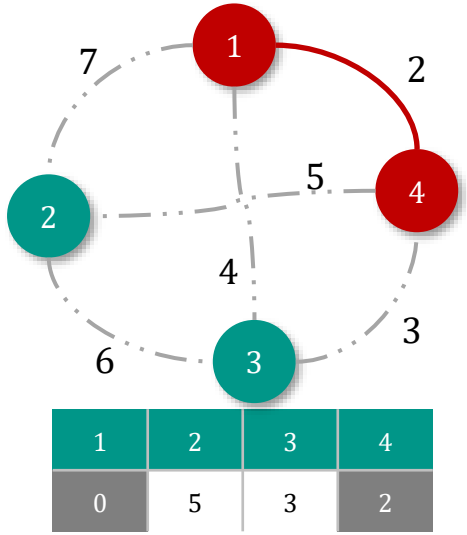
堆维护每个未加入的节点到已加入点的距离，每次取出最小值拓展距离，时间复杂度 $O(m\log n)$



1	2	3	4
0	7	4	2

初始时选择1号点

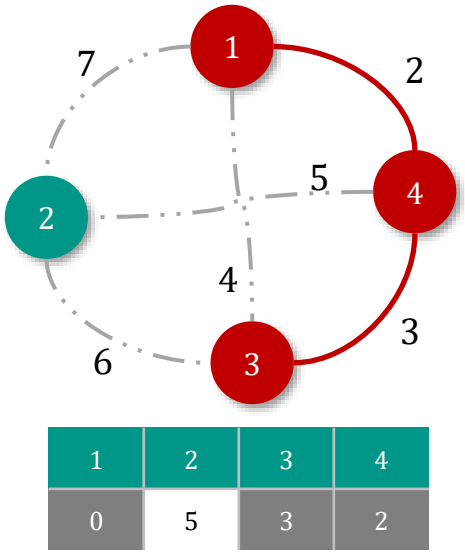
更新能到达的点距离



1	2	3	4
0	5	3	2

选择4号点

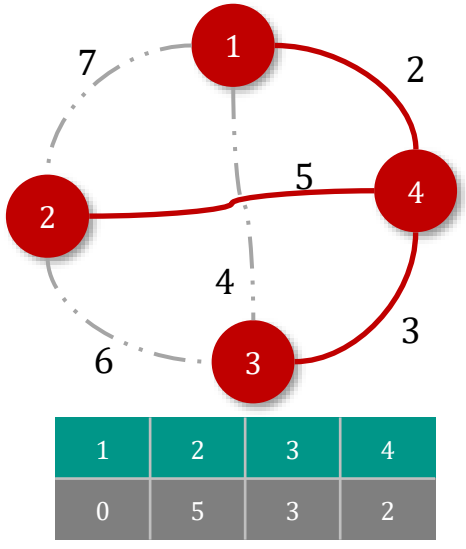
更新能到达的点距离



1	2	3	4
0	5	3	2

选择3号点

更新能到达的点距离



1	2	3	4
0	5	3	2

选择2号点

结束

prim算法

命题

对于图 $G=(V,E)$ 当 $k<|V|$ 时存在最小生成树包含 prim 算法前 k 步选择的边

证明

当 $k=1$ 时，prim 算法将找到与 1 有关权值最小边 $e=(1,u)$

若任意最小生成树都有 $e \notin T$

可将 e 加入 T（此时产生回路）替换与 1 关联的另一条边得到 T'

显然 T' 也为生成树，且有

$$W(T') \leq W(T)$$

即 $k=1$ 时命题成立

假设进行了 $k=n$ 步骤，选出的边集为 $e_1,e_2,\cdots,e_k$ 被被最小生成树 T 包含

记 $e_1,e_2,\cdots,e_k$ 中的节点构成点集 S

prim算法

考虑 $k = n+1$ 时，记此时选出边为 e_{k+1}

若 $e_{k+1} \in T$ 可说明 $k+1$ 步正确，若 $e_{k+1} \notin T$

将 e_{k+1} 加入 T 此时形成回路

此时 T 中必然存在连接 S 与 $V \setminus S$ 间的另一条边 e

令

$$T^* = T \setminus \{e\} \cup \{e_{k+1}\}$$

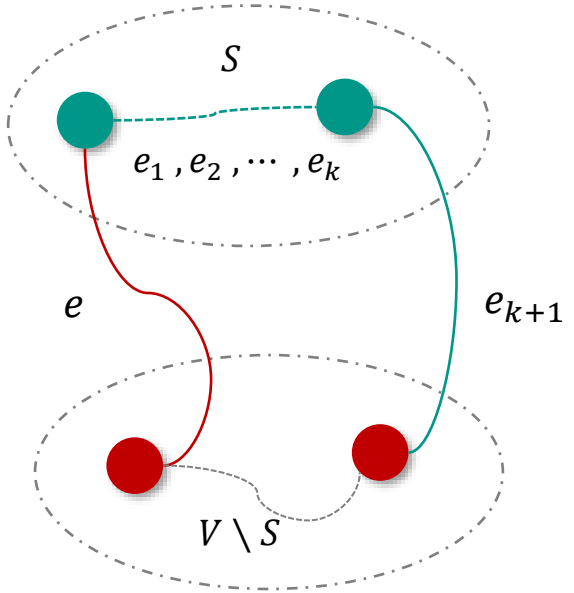
此时 T^* 也为一颗生成树包含 $e_1, e_2, \cdots, e_k$

又由于 e_{k+1} 为连接 S 与 $V \setminus S$ 的边权最小边，所以

$$W(T^*) \leq W(T)$$

即 $e_{k+1} \in T$

命题得证



#108、最小生成树prim

题面描述

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`

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 $u \sim v$ 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

对于全部的数据 $1 \leq n \leq 500, 1 \leq m \leq 10^5, 1 \leq w \leq 10000$

#280、唯一最小生成树

题目描述

给你一个无向连通图,判断其最小生成树是否唯一

输入格式

第一行包含两个整数 n 和 m ,分别表示节点和边的数目

接下来 m 行,每一行三个正整数 x_i, y_i, v_i ,表示边 (x_i, y_i) 的值为 v_i

对于任何两个节点,至多有一条边连接

输出格式

若改图有唯一最小生成树,则输出最小生成树的值

若不是,则输出 `Not Unique!`

数据规模

对于全部的数据 $1 \leq n \leq 100, 1 \leq v_i \leq 100000$

样例输入

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

样例输出

```
Not Unique!
```

#280、唯一最小生成树

思路 1

考虑最小生成树的唯一性

若一条边不在 MST 的边集中，其可被替换成权值相同、且在 MST 的另一条边

那么该 MST 是不唯一的

考虑 Kruskal 算法

统计当前权值的边可放几条，实际放了几条

若两值不一，说明这几条边与之前的边产生环，环中至少有两条当前权值的边

即最小生成树不唯一

可双指针优化

时间复杂度 $O(m \log m)$

#280、唯一最小生成树

思路 2

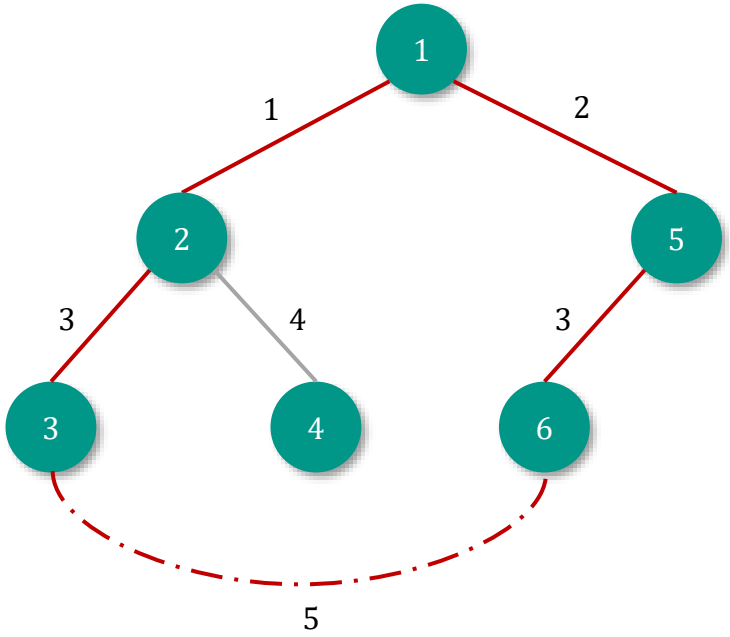
非严格次小生成树

在无向图中边权和最小的满足边权和 **大于等于** 最小生成树边权和的生成树

求出非严格次小生成树权值和 若与 MST 权值和相等说明不唯一

在最小生成树中一条非树边 (未选取的边) 若被加入必然成环

且该边不小于环上最大边权 , 若替换环上最大边,会使得权值和变大



求出无向图 MST 记为 T, 权值和记为 W

遍历每条未选中的边 $e = (u, v, w)$

找到 T 中 u 到 v 路径上边权最大的一条边 $e' = (s, t, w')$

在 T 中以 e' 替换 e,可得一棵权值和为 $M' = M - e + e'$ 的生成树

对所有替换得到的答案 M' 取最小值即可

考虑树上倍增 $u \to$ LCA , $v \to$ LCA 的路径上最大值

在求 LCA 过程中维护即可

#941、最小瓶颈路 加强版

题目描述

给定一个 n 个点 m 条边的无向连通图,编号为 1 到 n

> *没有自环,可能有重边,每一条边有一个正权值 w*

给出 q 个询问,每次给出两个不同的点 u 和 v ,求一条从 u 到 v 的路径上边权的最大值最小是多少

输出格式

输出共一行一个整数,表示所有询问的答案之和模 1000000007 的值

由于本题数据规模较大,直接输入输出会占用比计算多数倍的时间,因此对询问的输入输出进行了压缩

输出压缩方法是：输出所有询问的答案之和模 1000000007 的值

数据范围与提示

对于所有数据, $n \leq 70000, m \leq 100000, q \leq 10^7$

输入格式

输入第一行两个整数 n, m

接下来 m 行,每行三个整数 $a_i, b_i, w_i (a_i \neq b_i)$,表示一条边 (a_i, b_i) ,边权为 w_i

接下来一行一个整数 q ,表示询问数量

接下来一行四个整数 A, B, C, P ,表示询问的生成方式

由于本题数据规模较大,直接输入输出会占用比计算多数倍的时间,因此对询问的输入输出进行了压缩

输入压缩方法是：读入 4 个整数 A, B, C, P ,每次询问调用以下函数生成 u 和 v ：

```
int A,B,C,P;
inline int rnd(){return A=(A*B+C)%P;}
```

每次询问时的调用方法为：

```
u=rnd()%n+1,v=rnd()%n+1;
```

若 u 和 v 相等则答案为0

数据保证 $0 \leq A < P, 0 \leq C < P, P(B+1) < 2^{31} - 1$

#941、最小瓶颈路 加强版

最小瓶颈路

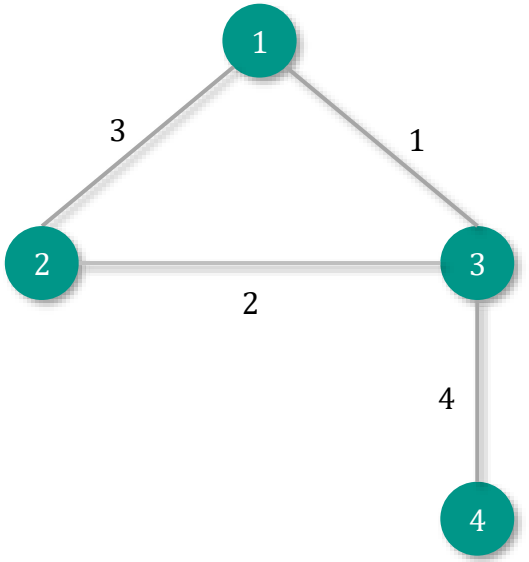
无向图 G 中 u 到 v 的最小瓶颈路是这样的一类简单路径

满足这条路径上的最大的边权在所有 u 到 v 的简单路径中是最小的

但并非所有最小瓶颈路都存在一棵最小生成树满足其为树上 u 到 v 的简单路径

如 1 到 4 的最小瓶颈路有$1 \to 2 \to 3 \to 4$ 和 $1 \to 3 \to 4$ 两条

但 $1 \to 2$ 并不会出现在 MST 上



询问任意两点 最小瓶颈路上的最大边权

考虑求出 MST , 问题转化为 树上简单路径 上的最大边权

可 树链剖分 $O(\log^2 n)$ 、树上倍增 $O(\log n)$ 求解

Kruskal 重构树

在 Kruskal 算法过程中会从小到大加入若干条边

维护 n 个集合，初始时各集合恰有 1 个节点,各点点权为 0

按照 Kruskal 算法流程 每一次加边会合并两个集合

可新建一个点，点权为加入边的边权

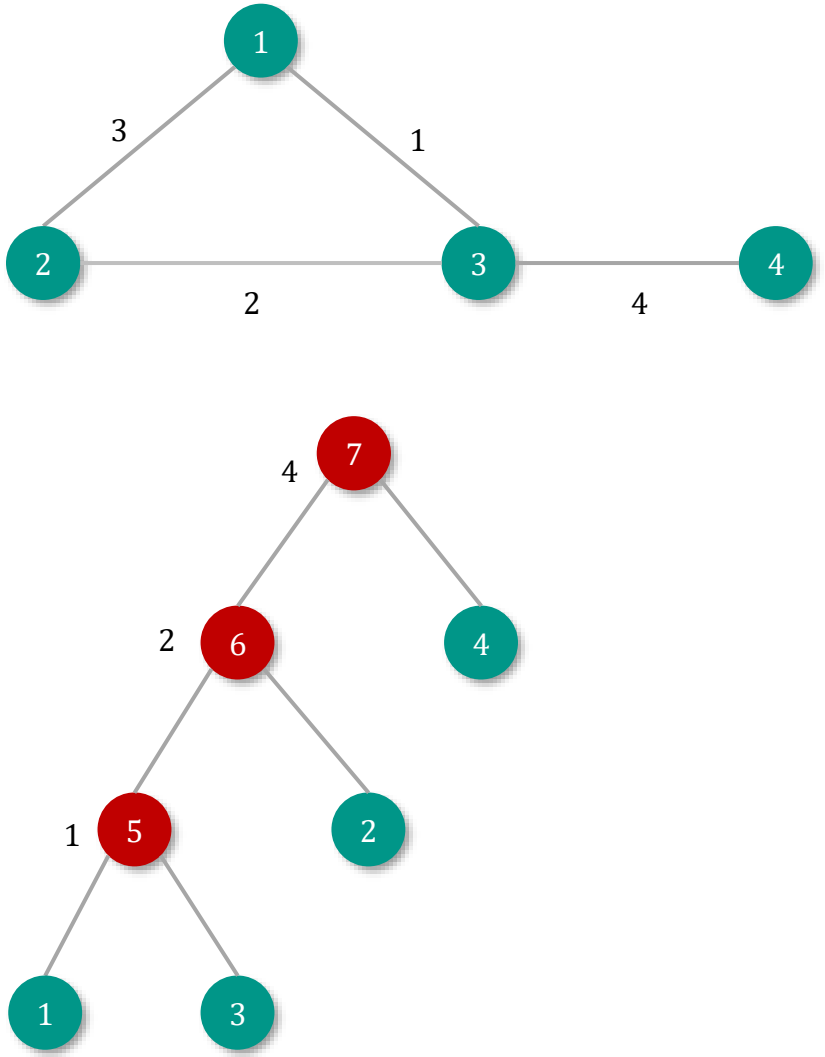
同时将两个集合的根节点分别设为新建点的左儿子和右儿子

再将两个集合和新建点合并成一个集合，将新建点设为根

进行 $n-1$ 轮之后得到了一棵恰有 n 个叶子的二叉树

各非叶子节点恰好有两个儿子

该树称为 Kruskal 重构树



Kruskal 重构树

原图中两个点之间的所有简单路径上最大边权的最小值

= 最小生成树上两个点之间的简单路径上的最大值

= Kruskal 重构树上两点之间的 LCA 的点权值

u 的简单路径上最大边权的最小值 $\leq$ val 的所有点 v 均在 Kruskal 重构树上的某一棵子树内

且恰好为该子树的所有叶子节点

在 Kruskal 重构树上找到 u 到根的路径上权值 $\leq$ val 的最浅节点

显然这就是所有满足条件的节点所在的子树的根节点

若求原图中两个点之间的所有简单路径上最小边权的最大值

执行 Kruskal 过程中按边权大到小的顺序加边即可

谢谢观看