图论 II

FOI 2022 算法夏令营基础班 Day 5

Jul 20, 2022

Contents

章节目录

回路

简单来说就是环。

回路

简单来说就是环。

初级回路

没有经过重复的**点**的回路称为**初级回路**。

回路

简单来说就是环。

初级回路

没有经过重复的**点**的回路称为**初级回路**。

简单回路

没有经过重复的**边**的回路称为**简单回路**。

回路

简单来说就是环。

初级回路

没有经过重复的**点**的回路称为**初级回路**。

简单回路

没有经过重复的**边**的回路称为**简单回路**。

初级回路一定是简单回路。

回路

简单来说就是环。

初级回路

没有经过重复的**点**的回路称为**初级回路**。

简单回路

没有经过重复的**边**的回路称为**简单回路**。

初级回路一定是简单回路。

欧拉回路

一个经过图上所有边的简单回路称为**欧拉回路**。

欧拉回路

关于欧拉回路的结论

一张连通无向图存在欧拉回路等价于所有点的度均为偶数。

欧拉回路

关于欧拉回路的结论

一张连通无向图存在欧拉回路等价于所有点的度均为偶数。
一张连通有向图存在欧拉回路等价于所有点的入度等于出度。

欧拉回路

关于欧拉回路的结论

一张连通无向图存在欧拉回路等价于所有点的度均为偶数。
一张连通有向图存在欧拉回路等价于所有点的入度等于出度。

欧拉路径

一个经过图上所有边的简单路径称为**欧拉路径**。

欧拉回路

关于欧拉回路的结论

一张连通无向图存在欧拉回路等价于所有点的度均为偶数。
一张连通有向图存在欧拉回路等价于所有点的入度等于出度。

欧拉路径

一个经过图上所有边的简单路径称为**欧拉路径**。

欧拉路径的存在性判定

1. 欧拉回路一定是欧拉路径，所以存在欧拉回路就一定存在欧拉路径。

欧拉回路

关于欧拉回路的结论

一张连通无向图存在欧拉回路等价于所有点的度均为偶数。
一张连通有向图存在欧拉回路等价于所有点的入度等于出度。

欧拉路径

一个经过图上所有边的简单路径称为**欧拉路径**。

欧拉路径的存在性判定

1. 欧拉回路一定是欧拉路径，所以存在欧拉回路就一定存在欧拉路径。
2. 如果不存在欧拉回路而存在欧拉路径，那么将欧拉路径的起点和终点连起来就可以得到欧拉回路。

欧拉回路

关于欧拉回路的结论

一张连通无向图存在欧拉回路等价于所有点的度均为偶数。
一张连通有向图存在欧拉回路等价于所有点的入度等于出度。

欧拉路径

一个经过图上所有边的简单路径称为**欧拉路径**。

欧拉路径的存在性判定

1. 欧拉回路一定是欧拉路径，所以存在欧拉回路就一定存在欧拉路径。
2. 如果不存在欧拉回路而存在欧拉路径，那么将欧拉路径的起点和终点连起来就可以得到欧拉回路。进一步得出若无向图存在欧拉路径那么只有可能有 0 个或 2 个点的度数为奇数的结论。

输出有向图的一条欧拉回路

算法流程

1. 由于在环路中每个点都是等价的，所以可以从任意一个点出发。

输出有向图的一条欧拉回路

算法流程

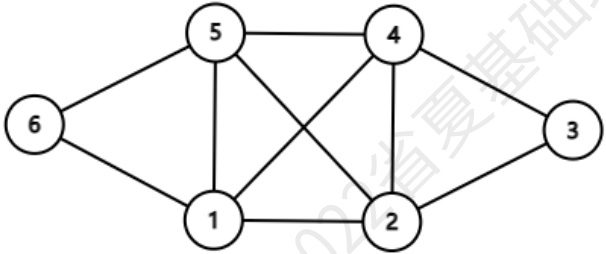
1. 由于在环路中每个点都是等价的，所以可以从任意一个点出发。
2. 到达某一个点后，我们枚举这个点的所有出边（记这条边的终点为 v）。若此时 v 没有还未走过的出边，则此时走这条边会提前形成回路，于是应该将这条边放到之后走；若 v 仍有未走过的出边，那么由于交换走边的顺序不会导致情况由有解变为无解，所以可以选择任意一条没有走过的边出发往下走。

输出有向图的一条欧拉回路

算法流程

1. 由于在环路中每个点都是等价的，所以可以从任意一个点出发。
2. 到达某一个点后，我们枚举这个点的所有出边（记这条边的终点为 v）。若此时 v 没有还未走过的出边，则此时走这条边会提前形成回路，于是应该将这条边放到之后走；若 v 仍有未走过的出边，那么由于交换走边的顺序不会导致情况由有解变为无解，所以可以选择任意一条没有走过的边出发往下走。
3. 不断重复以上过程，如果最后所有的边都能被走一遍，说明存在欧拉回路，否则不存在欧拉回路。

算法演示



参考代码

```
void dfs(int x) {
    for (int i=last[x];i;i=last[x]) {//和边链表的遍历略有不同
//由于其它次到达这个点时又会遍历一些边，所以直接用last更新
        if (!vis[i]) {//找一个没走过的边
            vis[i]=1;//标记为走过
            last[i]=e[i].next;//更新last，避免重复枚举
            dfs(e[i].to);//走下一个点
            ans[++cnt]=e[i].id;//记录答案
        }
    }  return;//全部边都走过之后会一路return回去
}
```

小测验 1：「CEOI1995」John's trip

求字典序最小的欧拉回路。

小测验 1：「CEOI1995」John's trip

求字典序最小的欧拉回路。
先将边按边权（字典序大小）排序，然后每到一个点就贪心地走字典序最小的边。

小测验 2:「USACO 2005 Jan. (Silver)」Watchcow

要求输出一张无向图的欧拉回路且每条边要正反各走一遍。

小测验 2：「USACO 2005 Jan. (Silver)」Watchcow

要求输出一张无向图的欧拉回路且每条边要正反各走一遍。
将每条无向边拆成两条有向边然后按有向图处理。
拓展：（UOJ 117 · 改）如果改成每条无向边只能经过一次呢？

小测验 2：「USACO 2005 Jan. (Silver)」Watchcow

要求输出一张无向图的欧拉回路且每条边要正反各走一遍。
将每条无向边拆成两条有向边然后按有向图处理。
拓展：（UOJ 117 · 改）如果改成每条无向边只能经过一次呢？
还是先拆边，不过在打 vis 的标记的时候要把反向边的 vis 也打上标记。

「CEOI1999」Play on Words

给出 n 个单词，两个单词能够相连当且仅当某一个单词的最后一个字母与另一个单词的第一个字母相同。试判断这些单词能否连接成一条链或者一个环（所有单词都要用且只能恰好用一次）。

数据范围：$1 \leq n \leq 10^5$。

「CEOI1999」Play on Words

给出 n 个单词，两个单词能够相连当且仅当某一个单词的最后一个字母与另一个单词的第一个字母相同。试判断这些单词能否连接成一条链或者一个环（所有单词都要用且只能恰好用一次）。

数据范围：$1 \leq n \leq 10^5$。

抓住关键词：连成一条链或一个环、所有的单词用一次，可以基本确定是欧拉路径/回路相关的问题。

「CEOI1999」Play on Words

给出 n 个单词，两个单词能够相连当且仅当某一个单词的最后一个字母与另一个单词的第一个字母相同。试判断这些单词能否连接成一条链或者一个环（所有单词都要用且只能恰好用一次）。

数据范围：$1 \leq n \leq 10^5$。

抓住关键词：连成一条链或一个环、所有的单词用一次，可以基本确定是欧拉路径/回路相关的问题。

可以将每个首位字母（单词的连接处）看成一个点，每个单词看成一条边（将当前连接成的字符串的末尾从一个字母变成另一个字母）。具体操作上，对 $a \sim z$ 的每个字母建一个点，若某单词从以字母 a 开头、字母 b 结尾，那么就从 a 连一条有向边到 b，最后判断是否存在欧拉路径即可。

章节目录

拓扑排序

拓扑排序是一个与现实生活紧密相连的算法。在做许多事情之前，我们需要先完成一些其它的事情才能保证整个过程的顺利进行。

拓扑排序

拓扑排序是一个与现实生活紧密相连的算法。在做许多事情之前，我们需要先完成一些其它的事情才能保证整个过程的顺利进行。

例如，在学习“差分约束系统”之前，要先学会用“SPFA 算法判断负环”，而这又需要先学习“最短路”的相关算法……

拓扑排序

拓扑排序是一个与现实生活紧密相连的算法。在做许多事情之前，我们需要先完成一些其它的事情才能保证整个过程的顺利进行。

例如，在学习“差分约束系统”之前，要先学会用“SPFA 算法判断负环”，而这又需要先学习“最短路”的相关算法……

在需要拓扑排序的事情较多、关系较为复杂时，我们很难根据直觉判断出做事情的正确顺序，而拓扑排序可以为我们提供一个合理的方案。

模板题

给定一个整数 n，要求输出一个 $1 \sim n$ 的排列，使其满足 m 个形如 x 在 y 之前出现的限制。复杂度要求 $\mathbb{O}(m)$。

模板题

给定一个整数 n，要求输出一个 $1 \sim n$ 的排列，使其满足 m 个形如 x 在 y 之前出现的限制。复杂度要求 $\mathbb{O}(m)$。

首先，这题可以转化为一个图论的模型：有 n 个点和 m 条有向边，只有当某一个点的满足所有指向它的点都被选取了之后这个点才能被选取，求一个满足条件的选点方案。

模板题

给定一个整数 n，要求输出一个 $1 \sim n$ 的排列，使其满足 m 个形如 x 在 y 之前出现的限制。复杂度要求 $\mathbb{O}(m)$。

首先，这题可以转化为一个图论的模型：有 n 个点和 m 条有向边，只有当某一个点的满足所有指向它的点都被选取了之后这个点才能被选取，求一个满足条件的选点方案。

算法流程

1. 维护一个队列，表示当前可以选的点。

模板题

给定一个整数 n，要求输出一个 $1 \sim n$ 的排列，使其满足 m 个形如 x 在 y 之前出现的限制。复杂度要求 $\mathbb{O}(m)$。

首先，这题可以转化为一个图论的模型：有 n 个点和 m 条有向边，只有当某一个点的满足所有指向它的点都被选取了之后这个点才能被选取，求一个满足条件的选点方案。

算法流程

1. 维护一个队列，表示当前可以选的点。
2. 从队首取出一个点（表示选取了这个点），打上标记。

模板题

给定一个整数 n，要求输出一个 $1 \sim n$ 的排列，使其满足 m 个形如 x 在 y 之前出现的限制。复杂度要求 $\mathbb{O}(m)$。

首先，这题可以转化为一个图论的模型：有 n 个点和 m 条有向边，只有当某一个点的满足所有指向它的点都被选取了之后这个点才能被选取，求一个满足条件的选点方案。

算法流程

1. 维护一个队列，表示当前可以选的点。
2. 从队首取出一个点（表示选取了这个点），打上标记。
3. 遍历所有的点，如果某个点满足条件，那么将它加入队列。

模板题

给定一个整数 n，要求输出一个 $1 \sim n$ 的排列，使其满足 m 个形如 x 在 y 之前出现的限制。复杂度要求 $\mathbb{O}(m)$。

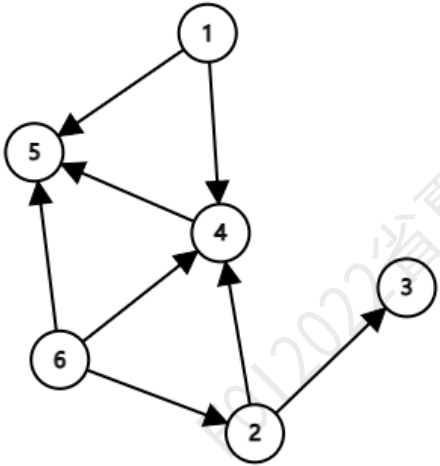
首先，这题可以转化为一个图论的模型：有 n 个点和 m 条有向边，只有当某一个点的满足所有指向它的点都被选取了之后这个点才能被选取，求一个满足条件的选点方案。

算法流程

1. 维护一个队列，表示当前可以选的点。
2. 从队首取出一个点（表示选取了这个点），打上标记。
3. 遍历所有的点，如果某个点满足条件，那么将它加入队列。

时间复杂度 $\mathbb{O}(n^2)$。

算法演示



模板题

给定一个整数 n，要求输出一个 $1 \sim n$ 的排列，使其满足 m 个形如 x 在 y 之前出现的限制。复杂度要求 $\mathbb{O}(m)$。

模板题

给定一个整数 n，要求输出一个 $1 \sim n$ 的排列，使其满足 m 个形如 x 在 y 之前出现的限制。复杂度要求 $\mathbb{O}(m)$。

算法优化

根据刚才对算法的模拟，可以发现每次有可能入队的只有当前选取的点所指向的点，于是只需要每次选出一个点后遍历它的所有出边并将可以选的点入队即可。

模板题

给定一个整数 n，要求输出一个 $1 \sim n$ 的排列，使其满足 m 个形如 x 在 y 之前出现的限制。复杂度要求 $\mathbb{O}(m)$。

算法优化

根据刚才对算法的模拟，可以发现每次有可能入队的只有当前选取的点所指向的点，于是只需要每次选出一个点后遍历它的所有出边并将可以选的点入队即可。
由于每条边只会被遍历到一次，可以得出时间复杂度为 $\mathbb{O}(m)$。

模板题

给定一个整数 n，要求输出一个 $1 \sim n$ 的排列，使其满足 m 个形如 x 在 y 之前出现的限制。复杂度要求 $\mathbb{O}(m)$。

算法优化

根据刚才对算法的模拟，可以发现每次有可能入队的只有当前选取的点所指向的点，于是只需要每次选出一个点后遍历它的所有出边并将可以选的点入队即可。
由于每条边只会被遍历到一次，可以得出时间复杂度为 $\mathbb{O}(m)$。

小测验 3

1. 如何输出一张 DAG 的拓扑序?

模板题

给定一个整数 n，要求输出一个 $1 \sim n$ 的排列，使其满足 m 个形如 x 在 y 之前出现的限制。复杂度要求 $\mathbb{O}(m)$。

算法优化

根据刚才对算法的模拟，可以发现每次有可能入队的只有当前选取的点所指向的点，于是只需要每次选出一个点后遍历它的所有出边并将可以选的点入队即可。
由于每条边只会被遍历到一次，可以得出时间复杂度为 $\mathbb{O}(m)$。

小测验 3

1. 如何输出一张 DAG 的拓扑序？每选出一个点就记录下来。

模板题

给定一个整数 n，要求输出一个 $1 \sim n$ 的排列，使其满足 m 个形如 x 在 y 之前出现的限制。复杂度要求 $\mathbb{O}(m)$。

算法优化

根据刚才对算法的模拟，可以发现每次有可能入队的只有当前选取的点所指向的点，于是只需要每次选出一个点后遍历它的所有出边并将可以选的点入队即可。
由于每条边只会被遍历到一次，可以得出时间复杂度为 $\mathbb{O}(m)$。

小测验 3

1. 如何输出一张 DAG 的拓扑序？每选出一个点就记录下来。
2. 如何判断是否无解？

模板题

给定一个整数 n，要求输出一个 $1 \sim n$ 的排列，使其满足 m 个形如 x 在 y 之前出现的限制。复杂度要求 $\mathbb{O}(m)$。

算法优化

根据刚才对算法的模拟，可以发现每次有可能入队的只有当前选取的点所指向的点，于是只需要每次选出一个点后遍历它的所有出边并将可以选的点入队即可。
由于每条边只会被遍历到一次，可以得出时间复杂度为 $\mathbb{O}(m)$。

小测验 3

1. 如何输出一张 DAG 的拓扑序？每选出一个点就记录下来。
2. 如何判断是否无解？当算法求出的拓扑序的长度不是 n 的时候就表示无解。

参考代码

```
queue<int> q;//维护一个队列
while (!q.empty()) q.pop();//清空队列
for (int i=1;i<=n;++i)
    if (!indeg[i]) q.push(i);//将所有没有入度的点入队
while (!q.empty()) {
    int cur=q.front();q.pop();//找一个点进行更新
    topo[++pnt]=cur;//记入拓扑序
    for (int i=last[cur];i;i=e[i].next) {
        int ot=e[i].to;--indeg[ot];//更新周围的点
        if (!indeg[ot]) q.push(ot);//将没有入度的点入队
    }
}
```

「NOIP2013 普及组」车站分级

一条铁路线上有 n 个站点，每个站点有一个等级。有 m 条线路，每条线路停靠其中的若干个站点，且满足如果某个线路停靠了站点 x，那么线路上任意一个级别不低于 x 的站点也一定要停靠。现已知整个运行图，求车站最少要分成几个等级。

数据范围：$1 \leq n, m \leq 1\,000$。

「NOIP2013 普及组」车站分级

一条铁路线上有 n 个站点，每个站点有一个等级。有 m 条线路，每条线路停靠其中的若干个站点，且满足如果某个线路停靠了站点 x，那么线路上任意一个级别不低于 x 的站点也一定要停靠。现已知整个运行图，求车站最少要分成几个等级。

数据范围：$1 \leq n, m \leq 1\,000$。

对于每一条线，我们可以得出的信息是停靠的车站比不停的车站等级更高，于是可以根据这个关系建立一张拓扑图。问题转化为给定一张拓扑图，要求算出这个拓扑图最少要分成几层，使所有的边都从前面的某一层指向后面的另一层。

「NOIP2013 普及组」车站分级

一条铁路线上有 n 个站点，每个站点有一个等级。有 m 条线路，每条线路停靠其中的若干个站点，且满足如果某个线路停靠了站点 x，那么线路上任意一个级别不低于 x 的站点也一定要停靠。现已知整个运行图，求车站最少要分成几个等级。

数据范围：$1 \leq n, m \leq 1\,000$。

对于每一条线，我们可以得出的信息是停靠的车站比不停的车站等级更高，于是可以根据这个关系建立一张拓扑图。问题转化为给定一张拓扑图，要求算出这个拓扑图最少要分成几层，使所有的边都从前面的某一层指向后面的另一层。

根据前面的经验，这种和原算法很像但是需要增加一个功能的情况常常可以在原来的算法上修修改改。

「NOIP2013 普及组」车站分级

一条铁路线上有 n 个站点，每个站点有一个等级。有 m 条线路，每条线路停靠其中的若干个站点，且满足如果某个线路停靠了站点 x，那么线路上任意一个级别不低于 x 的站点也一定要停靠。现已知整个运行图，求车站最少要分成几个等级。

数据范围：$1 \leq n, m \leq 1\,000$。

对于每一条线，我们可以得出的信息是停靠的车站比不停的车站等级更高，于是可以根据这个关系建立一张拓扑图。问题转化为给定一张拓扑图，要求算出这个拓扑图最少要分成几层，使所有的边都从前面的某一层指向后面的另一层。

根据前面的经验，这种和原算法很像但是需要增加一个功能的情况常常可以在原来的算法上修修改改。

具体来说，这次我们将算法改为一层一层取点，等整层的点都取完了之后再遍历所有相关的点，选出可以在下一层被选的点，不断重复这个过程直到所有的点都被选取即可得出最少分的层数。

章节目录

最小生成树

生成树

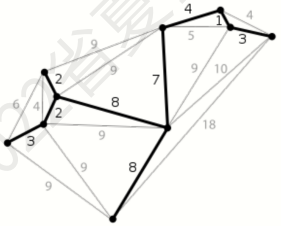
对于一张图 G，如果它的子图 G' 包括了 G 中的所有点且是一棵树，那么称 G' 为 G 的**生成树**。

最小生成树

生成树

对于一张图 G，如果它的子图 G' 包括了 G 中的所有点且是一棵树，那么称 G' 为 G 的**生成树**。

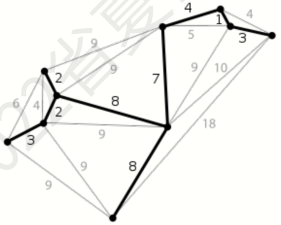
说人话版本：用原图中的所有点和部分边构成一棵树，那么这棵树是原图的生成树。



最小生成树

生成树

对于一张图 G，如果它的子图 G' 包括了 G 中的所有点且是一棵树，那么称 G' 为 G 的**生成树**。
说人话版本：用原图中的所有点和部分边构成一棵树，那么这棵树是原图的生成树。



在一张图的所有生成树中，边权之和最小的称为**最小生成树**。

Kruskal 算法

Kruskal 算法是一种基于贪心的最小生成树求解算法，可以在 $\mathbb{O}(m \log m)$ 的时间内完成求解。

算法流程

1. 将边按照边权从小到大排序，并建立一个只包含原图中所有的点的图 T。

Kruskal 算法

Kruskal 算法是一种基于贪心的最小生成树求解算法，可以在 $\mathbb{O}(m \log m)$ 的时间内完成求解。

算法流程

1. 将边按照边权从小到大排序，并建立一个只包含原图中所有的点的图 T。
2. 选出一条没有被选过的边权最小的边。

Kruskal 算法

Kruskal 算法是一种基于贪心的最小生成树求解算法，可以在 $\mathbb{O}(m \log m)$ 的时间内完成求解。

算法流程

1. 将边按照边权从小到大排序，并建立一个只包含原图中所有的点的图 T。
2. 选出一条没有被选过的边权最小的边。
3. 如果这条边两个顶点在 T 中所在的连通块不相同，那么将它加入图 T。

Kruskal 算法

Kruskal 算法是一种基于贪心的最小生成树求解算法，可以在 $\mathbb{O}(m \log m)$ 的时间内完成求解。

算法流程

1. 将边按照边权从小到大排序，并建立一个只包含原图中所有的点的图 T。
2. 选出一条没有被选过的边权最小的边。
3. 如果这条边两个顶点在 T 中所在的连通块不相同，那么将它加入图 T。
4. 不断重复步骤 2 和 3 直到 T 连通为止。

Kruskal 算法

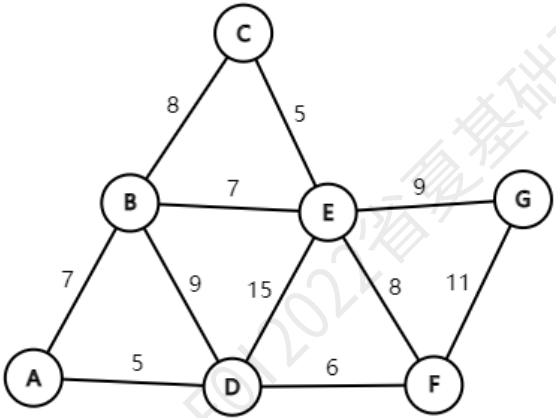
Kruskal 算法是一种基于贪心的最小生成树求解算法，可以在 $\mathbb{O}(m \log m)$ 的时间内完成求解。

算法流程

1. 将边按照边权从小到大排序，并建立一个只包含原图中所有的点的图 T。
2. 选出一条没有被选过的边权最小的边。
3. 如果这条边两个顶点在 T 中所在的连通块不相同，那么将它加入图 T。
4. 不断重复步骤 2 和 3 直到 T 连通为止。

在实际操作中，由于只需要维护 T 的连通性，可以不需要真正建出图 T，而只需要用并查集判断连通性即可。

Kruskal 算法模拟



Prim 算法

算法流程

1. 先建立一个只有一个结点的树，这个结点可以是原图中任意的一个结点。

Prim 算法

算法流程

1. 先建立一个只有一个结点的树，这个结点可以是原图中任意的一个结点。
2. 使用一条边扩展这个树，要求这条边恰有一个顶点在树中，并且要求这条边的权值在所有符合条件的边中最小。

Prim 算法

算法流程

1. 先建立一个只有一个结点的树，这个结点可以是原图中任意的一个结点。
2. 使用一条边扩展这个树，要求这条边恰有一个顶点在树中，并且要求这条边的权值在所有符合条件的边中最小。
3. 重复步骤 2 直到所有顶点都在树中。

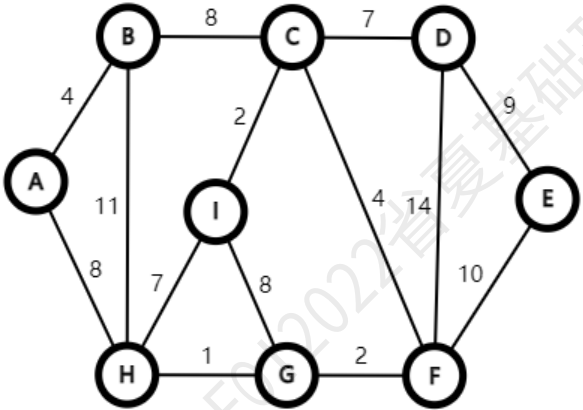
Prim 算法

算法流程

1. 先建立一个只有一个结点的树，这个结点可以是原图中任意的一个结点。
2. 使用一条边扩展这个树，要求这条边恰有一个顶点在树中，并且要求这条边的权值在所有符合条件的边中最小。
3. 重复步骤 2 直到所有顶点都在树中。

在实际操作中，可以维护每个结点到已经选中的顶点中权值最小的边的权值（记为 m_i）。每次选点时就通过堆选出还没有选过的点中 m_i 最小的点，将这个点和对应的边加入生成树中并更新与这个点周围的点的 m_i。时间复杂度 $\mathbb{O}(m \log n)$。

Prim 算法模拟



Boruvka 算法

Boruvka 算法可以看成 Kruskal 算法的多线程版本——每次不是考虑加入一条边而是同时考虑多条边的加入。

Boruvka 算法

Boruvka 算法可以看成 Kruskal 算法的多线程版本——每次不是考虑加入一条边而是同时考虑多条边的加入。

算法流程

1. 对所有的点维护一个并查集，初始时所有的点都是孤立的。

Boruvka 算法

Boruvka 算法可以看成 Kruskal 算法的多线程版本——每次不是考虑加入一条边而是同时考虑多条边的加入。

算法流程

1. 对所有的点维护一个并查集，初始时所有的点都是孤立的。
2. 对于每一个连通块（并查集）i，维护一个数组 $near[i]$ 表示恰有一个端点在这个连通块中的边中边权最小的边。

Boruvka 算法

Boruvka 算法可以看成 Kruskal 算法的多线程版本——每次不是考虑加入一条边而是同时考虑多条边的加入。

算法流程

1. 对所有的点维护一个并查集，初始时所有的点都是孤立的。
2. 对于每一个连通块（并查集）i，维护一个数组 $near[i]$ 表示恰有一个端点在这个连通块中的边中边权最小的边。
3. 每个连通块 i 通过 $near[i]$ 和别的连通块连边（根据贪心，由于每个连通块迟早要和别人连通，所以选取一条符合条件且最短的边一定是最优的）。

Boruvka 算法

Boruvka 算法可以看成 Kruskal 算法的多线程版本——每次不是考虑加入一条边而是同时考虑多条边的加入。

算法流程

1. 对所有的点维护一个并查集，初始时所有的点都是孤立的。
2. 对于每一个连通块（并查集）i，维护一个数组 $near[i]$ 表示恰有一个端点在这个连通块中的边中边权最小的边。
3. 每个连通块 i 通过 $near[i]$ 和别的连通块连边（根据贪心，由于每个连通块迟早要和别人连通，所以选取一条符合条件且最短的边一定是最优的）。
4. 不断重复步骤 2 和 3 直到所有点连通。

Boruvka 算法

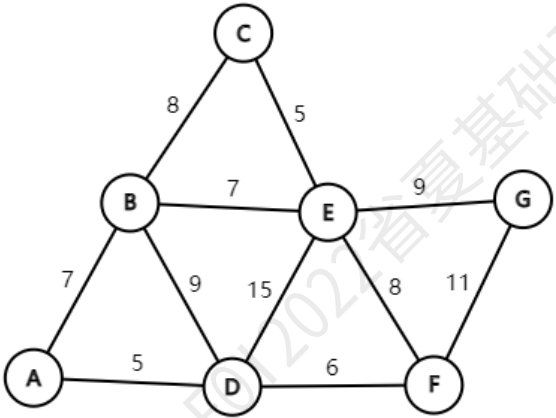
Boruvka 算法可以看成 Kruskal 算法的多线程版本——每次不是考虑加入一条边而是同时考虑多条边的加入。

算法流程

1. 对所有的点维护一个并查集，初始时所有的点都是孤立的。
2. 对于每一个连通块（并查集）i，维护一个数组 $near[i]$ 表示恰有一个端点在这个连通块中的边中边权最小的边。
3. 每个连通块 i 通过 $near[i]$ 和别的连通块连边（根据贪心，由于每个连通块迟早要和别人连通，所以选取一条符合条件且最短的边一定是最优的）。
4. 不断重复步骤 2 和 3 直到所有点连通。

Boruvka 算法的时间复杂度为 $\mathbb{O}(m \log n)$，和 Prim 一样可以接受 m 较大的情况；另外 Boruvka 算法和 Kruskal 算法一样不使用复杂的数据结构，可以为之后嵌套其它的数据结构留下空间。

Boruvka 算法模拟



「经典题」最小瓶颈路

给出一张 n 个点 m 条边的无向图，边有权值。记一条路径的权值为路径上各边边权的最大值。求从 s 到 t 的所有路径中权值最小的路径的权值。

数据范围：$1 \le n \le 10^4, 1 \le m \le 2 \times 10^4$。

「经典题」最小瓶颈路

给出一张 n 个点 m 条边的无向图，边有权值。记一条路径的权值为路径上各边边权的最大值。求从 s 到 t 的所有路径中权值最小的路径的权值。

数据范围：$1 \le n \le 10^4, 1 \le m \le 2 \times 10^4$。

涉及最大值的问题一般有两种情况：

「经典题」最小瓶颈路

给出一张 n 个点 m 条边的无向图，边有权值。记一条路径的权值为路径上各边边权的最大值。求从 s 到 t 的所有路径中权值最小的路径的权值。

数据范围：$1 \leq n \leq 10^4, 1 \leq m \leq 2 \times 10^4$。

涉及最大值的问题一般有两种情况：

1. 求一堆数里面的最大值：此时可以利用各种各样的数据结构来维护和求解。

「经典题」最小瓶颈路

给出一张 n 个点 m 条边的无向图，边有权值。记一条路径的权值为路径上各边边权的最大值。求从 s 到 t 的所有路径中权值最小的路径的权值。

数据范围：$1 \leq n \leq 10^4, 1 \leq m \leq 2 \times 10^4$。

涉及最大值的问题一般有两种情况：

1. 求一堆数里面的最大值：此时可以利用各种各样的数据结构来维护和求解。

2. 在计算贡献（如这题的权值）时式子中包含了对于最大值的计算：此时如果使用数据结构来维护则一般会很复杂，所以更为常用的方式是分别计算每个数作为最大值的情况对答案的贡献。当然，同样重要的条件是在要求某个数作为最大值的情况下，式子的其它部分（如这题的边权之和）是相对易于计算的。

「经典题」最小瓶颈路

给出一张 n 个点 m 条边的无向图，边有权值。记一条路径的权值为路径上各边边权的最大值。求从 s 到 t 的所有路径中权值最小的路径的权值。

数据范围：$1 \le n \le 10^4, 1 \le m \le 2 \times 10^4$。

涉及最大值的问题一般有两种情况：

1. 求一堆数里面的最大值：此时可以利用各种各样的数据结构来维护和求解。

2. 在计算贡献（如这题的权值）时式子中包含了对于最大值的计算：此时如果使用数据结构来维护则一般会很复杂，所以更为常用的方式是分别计算每个数作为最大值的情况对答案的贡献。当然，同样重要的条件是在要求某个数作为最大值的情况下，式子的其它部分（如这题的边权之和）是相对易于计算的。

回到这题，考虑如何判断用边权最小的若干条边让 s 与 t 连通。

「经典题」最小瓶颈路

给出一张 n 个点 m 条边的无向图，边有权值。记一条路径的权值为路径上各边边权的最大值。求从 s 到 t 的所有路径中权值最小的路径的权值。

数据范围：$1 \le n \le 10^4, 1 \le m \le 2 \times 10^4$。

涉及最大值的问题一般有两种情况：

1. 求一堆数里面的最大值：此时可以利用各种各样的数据结构来维护和求解。

2. 在计算贡献（如这题的权值）时式子中包含了对于最大值的计算：此时如果使用数据结构来维护则一般会很复杂，所以更为常用的方式是分别计算每个数作为最大值的情况对答案的贡献。当然，同样重要的条件是在要求某个数作为最大值的情况下，式子的其它部分（如这题的边权之和）是相对易于计算的。

回到这题，考虑如何判断用边权最小的若干条边让 s 与 t 连通。

我们可以将所有的边排序，然后用类似 Kruskal 算法的方法将边依次加入，当 s 与 t 连通时，最后一条加入的边就是所求的边。

章节目录

最近公共祖先（LCA）

祖先

一个节点到根节点上的各点称为这个点的**祖先**。

最近公共祖先（LCA）

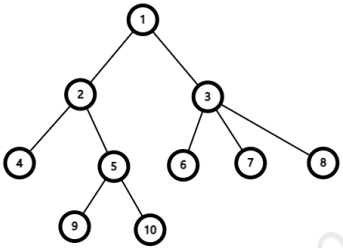
祖先

一个节点到根节点上的各点称为这个点的**祖先**。

最近公共祖先（LCA）

对于一棵树上的两个点 u, v，满足点 d 是这两个点共同的祖先，那么 d 就是 u 和 v 的公共祖先。在 u 和 v 所有的祖先中离 u 和 v 最近的一个就是 u 和 v 的**最近公共祖先**，记为 $LCA(u, v)$。

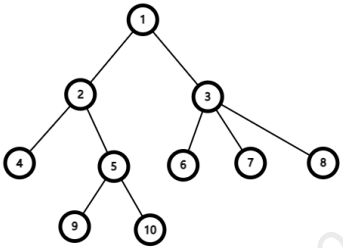
朴素的暴力求解方法



算法流程

1. 若两个节点深度不同，则先让深度较大的节点向上跳，直至两个点的深度相同。

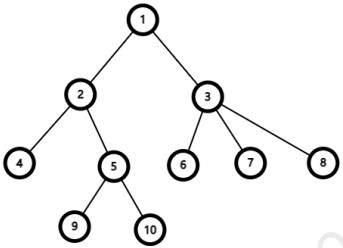
朴素的暴力求解方法



算法流程

1. 若两个节点深度不同，则先让深度较大的节点向上跳，直至两个点的深度相同。
2. 两个节点每次一起向上各走一步，直到两个点相遇，此时它们所在的位置就是 LCA。

朴素的暴力求解方法



算法流程

1. 若两个节点深度不同，则先让深度较大的节点向上跳，直至两个点的深度相同。
2. 两个节点每次一起向上各走一步，直到两个点相遇，此时它们所在的位置就是 LCA。

这个暴力算法效率不高的原因是两步操作都要靠一层一层向上走来解决，也就意味着在最坏情况下时间复杂度为 $\mathbb{O}(n)$。那么优化算法的方向就是让每个点每次能跳很多层。

用倍增求解

通过上面的分析，我们发现想要提高算法的效率，就要更加方便地知道某个点的若干级祖先是谁。但如果将每个点的所有祖先都记录下来，则又会因为预处理而浪费过多时间。通过类比二分的思想，我们可以得出下面这个算法。

算法流程

1. 记 $anc[i][j]$ 表示点 i 的 2^j 级祖先，则 $anc[i][0] = fa[i], anc[i][j] = anc[anc[i][j-1]][j-1]$。

用倍增求解

通过上面的分析，我们发现想要提高算法的效率，就要更加方便地知道某个点的若干级祖先是谁。但如果将每个点的所有祖先都记录下来，则又会因为预处理而浪费过多时间。通过类比二分的思想，我们可以得出下面这个算法。

算法流程

1. 记 $anc[i][j]$ 表示点 i 的 2^j 级祖先，则 $anc[i][0] = fa[i], anc[i][j] = anc[anc[i][j-1]][j-1]$。
2. 对于原来的第一步（调整至同一深度），可以将两个点的深度差（记为 k）进行二进制分解（即拆成若干个互不相同的 2 的次幂相加的形式），如果分解后存在 2^x 的项，那么就向上跳 2^x 层。在这一步之后，两个点就在同一高度上了（当然如果这两个点已经重合了那么就直接结束）。时间复杂度 $\mathbb{O}(\log k)$。

用倍增求解

算法流程

3. 让 u 和 v 一起往上跳。如果说两个点在跳之后没有碰到，那么这次的跳跃一定是正确的，于是可以执行。但是如果经过这一次的跳跃，两个点到达了同一个点，那么我们只能说这个点是公共祖先，但不一定是最近公共祖先，所以这样的跳跃暂不执行。我们根据这个方法，从大到小枚举 2 的次幂，如果 u 和 v 跳跃后碰到了一起，那么就将跳的层数除以 2，直到两个点跳完后不会碰到一起为止，然后将这两个点一起跳到新的位置，继续枚举 2 的次幂。注意，这一次的枚举不用从头进行，因为这次的跳跃距离不可能超过上一次的跳跃距离（否则在更早的时候就会跳了）。此时，我们发现 u 和 v 都停在最近公共祖先的子节点上，那么最近公共祖先就是 $anc[u][0]$。总时间复杂度 $\mathbb{O}(\log n)$。

总的时间复杂度为 $\mathbb{O}(\log n)$。

参考代码

```
int lca(int u,int v) {
    if (dep[u]<dep[v]) swap(u,v);
    int k=dep[u]-dep[v];
    for (int i=0;i<=20;++i)
        if (k&(1<<i)) u=anc[u][i];
    if (u==v) reutrn u;
    for (int i=20;i>=0;--i)
        if (anc[u][i]!=anc[v][i]) {
            u=anc[u][i];
            v=anc[v][i];
        }
    return anc[u][0];
}
```

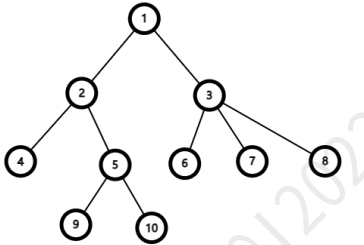
欧拉序 II

除了之前讲过的那种欧拉序外，还有一种序列也称作欧拉序——在 DFS 的过程中，**每经过**一个点就将这个点记录下来（注意区分：DFS 序是第一次经过的时候才记录），得到的序列也称作**欧拉序**。

欧拉序 II

除了之前讲过的那种欧拉序外，还有一种序列也称作欧拉序——在 DFS 的过程中，**每经过**一个点就将这个点记录下来（注意区分：DFS 序是第一次经过的时候才记录），得到的序列也称作**欧拉序**。

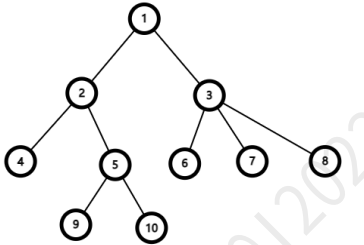
小测验 4：写出下面这棵树的欧拉序。



欧拉序 II

除了之前讲过的那种欧拉序外，还有一种序列也称作欧拉序——在 DFS 的过程中，**每经过**一个点就将这个点记录下来（注意区分：DFS 序是第一次经过的时候才记录），得到的序列也称作**欧拉序**。

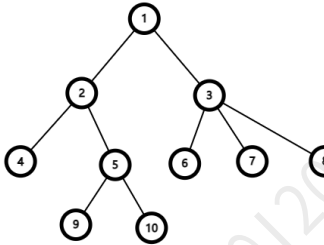
小测验 4：写出下面这棵树的欧拉序。



欧拉序 II

除了之前讲过的那种欧拉序外，还有一种序列也称作欧拉序——在 DFS 的过程中，**每经过**一个点就将这个点记录下来（注意区分：DFS 序是第一次经过的时候才记录），得到的序列也称作**欧拉序**。

小测验 4：写出下面这棵树的欧拉序。



欧拉序为：$1, 2, 4, 2, 5, 9, 5, 10, 5, 2, 1, 3, 6, 3, 7, 3, 8, 3, 1$。

欧拉序 II

欧拉序的性质

由于欧拉序是由 DFS 形成的，这些性质都可以借助 DFS 的过程来理解。

欧拉序 II

欧拉序的性质

由于欧拉序是由 DFS 形成的，这些性质都可以借助 DFS 的过程来理解。

1. 以某个点 u 为根的子树中的点一定出现在欧拉序中的连续一段，且以 u 第一次出现的位置开始到 u 最后一次出现的位置结束。

欧拉序 II

欧拉序的性质

由于欧拉序是由 DFS 形成的，这些性质都可以借助 DFS 的过程来理解。

1. 以某个点 u 为根的子树中的点一定出现在欧拉序中的连续一段，且以 u 第一次出现的位置开始到 u 最后一次出现的位置结束。
2. 每个节点的出现次数均为子节点个数加 1（第一次到达这个点的时候记一次，从每个子树回溯回来又各记一次）。进一步累加可知一棵 n 各点的树的欧拉序长度为 $2n-1$（所以空间也要开两倍）。

欧拉序 II

欧拉序的性质

由于欧拉序是由 DFS 形成的，这些性质都可以借助 DFS 的过程来理解。

1. 以某个点 u 为根的子树中的点一定出现在欧拉序中的连续一段，且以 u 第一次出现的位置开始到 u 最后一次出现的位置结束。
2. 每个节点的出现次数均为子节点个数加 1（第一次到达这个点的时候记一次，从每个子树回溯回来又各记一次）。进一步累加可知一棵 n 各点的树的欧拉序长度为 $2n-1$（所以空间也要开两倍）。
3. 欧拉序中点 u 与点 v 之间深度最小的点就是 $LCA(u,v)$。

复习 · RMQ

有一个数列 a_n 和 q 次询问，每次询问一个区间的最小值。
数据范围：$n \leq 10^5, q \leq 10^7$。

复习 · RMQ

有一个数列 a_n 和 q 次询问，每次询问一个区间的最小值。

数据范围：$n \le 10^5, q \le 10^7$。

令 $f[i][j]$ 表示区间 $[i, i+2^j-1]$ 的最小值，那么有 $f[i][j] = \min\{f[i][j-1], f[i+2^{j-1}][j-1]\}$，即区间的最小值为区间前一半的最小值和区间后一半的最小值中的较小值。假定询问的区间是 $[l, r]$，并且 $k = \lfloor \log(r-l+1) \rfloor$。那么区间的最小值就是 $\min\{f[l][k], f[r-2^k+1][k]\}$（虽然中间重复了一段，但是不会对答案造成影响）。

时间复杂度 $\mathbb{O}(n \log n + q)$（RMQ 之所以特殊是因为它是在类似问题上为数不多可以做到 $\mathbb{O}(1)$ 查询的算法）。

用欧拉序求解

根据性质 3，我们可以用数据结构维护欧拉序，此时也可以将单次询问的时间复杂度做到 $\mathbb{O}(\log n)$。不过当询问次数继续增加，连 $\mathbb{O}(\log n)$ 的时间复杂度都无法接受时，就可以考虑用 RMQ 来求解了。

用欧拉序求解

根据性质 3，我们可以用数据结构维护欧拉序，此时也可以将单次询问的时间复杂度做到 $\mathbb{O}(\log n)$。不过当询问次数继续增加，连 $\mathbb{O}(\log n)$ 的时间复杂度都无法接受时，就可以考虑用 RMQ 来求解了。

定义 $f[i][j]$ 表示欧拉序在区间 $[i, i+2^j-1]$ 的点中深度最小的点的编号。那么状态转移方程就是 $f[i][j] = cmp(f[i][j-1], f[i+2^j][j-1])$，其中 cmp 为一个比较二者深度并返回较浅的点的编号的函数。若点 u **最后一次**出现的位置为 l，点 v 第一次**第一次**出现的位置为 r，并且 $k = \lfloor \log(r-l+1) \rfloor$。那么 u 和 v 的最近公共祖先就是 $cmp(f[l][k], f[r-2^k+1][k])$。

用欧拉序求解

根据性质 3，我们可以用数据结构维护欧拉序，此时也可以将单次询问的时间复杂度做到 $\mathbb{O}(\log n)$。不过当询问次数继续增加，连 $\mathbb{O}(\log n)$ 的时间复杂度都无法接受时，就可以考虑用 RMQ 来求解了。

定义 $f[i][j]$ 表示欧拉序在区间 $[i, i+2^j-1]$ 的点中深度最小的点的编号。那么状态转移方程就是 $f[i][j] = cmp(f[i][j-1], f[i+2^j][j-1])$，其中 cmp 为一个比较二者深度并返回较浅的点的编号的函数。若点 u **最后一次**出现的位置为 l，点 v 第一次**第一次**出现的位置为 r，并且 $k = \lfloor \log(r-l+1) \rfloor$。那么 u 和 v 的最近公共祖先就是 $cmp(f[l][k], f[r-2^k+1][k])$。

预处理的时间复杂度为 $\mathbb{O}(n \log n)$，单次询问的时间复杂度为 $\mathbb{O}(1)$。

用 Tarjan 求解

对所有的点维护一个并查集，然后对整棵树做正常的 DFS，每当我们结束对一个点的搜索（即最后离开这个点）时，我们就检查一下是否有关于这个点和另外一个已经完成搜索（之后不再经过）的点 k 的询问，如果有，这个询问的答案就是 k 在并查集中的根。最后将当前点所在的并查集和父节点所在的并查集合并。

用 Tarjan 求解

对所有的点维护一个并查集，然后对整棵树做正常的 DFS，每当我们结束对一个点的搜索（即最后离开这个点）时，我们就检查一下是否有关于这个点和另外一个已经完成搜索（之后不再经过）的点 k 的询问，如果有，这个询问的答案就是 k 在并查集中的根。最后将当前点所在的并查集和父节点所在的并查集合并。

此处应有演示动画 ～

用 Tarjan 求解

对所有的点维护一个并查集，然后对整棵树做正常的 DFS，每当我们结束对一个点的搜索（即最后离开这个点）时，我们就检查一下是否有关于这个点和另外一个已经完成搜索（之后不再经过）的点 k 的询问，如果有，这个询问的答案就是 k 在并查集中的根。最后将当前点所在的并查集和父节点所在的并查集合并。

此处应有演示动画 ~

正确性证明：在进行搜索的过程中，从 u 到 v 的一条路径为 $u \to LCA(u,v) \to \cdots \to LCA(u,v) \to v$，即 u 在向上合并的时候会与 $LCA(u,v)$ 并入同一个并查集，但是由于 $LCA(u,v)$ 还没有完成搜索，所以 u 和 $fa_{LCA(u,v)}$ 不在一个并查集中，所以 $LCA(u,v)$ 就是 u 所在并查集的根。

用 Tarjan 求解

对所有的点维护一个并查集，然后对整棵树做正常的 DFS，每当我们结束对一个点的搜索（即最后离开这个点）时，我们就检查一下是否有关于这个点和另外一个已经完成搜索（之后不再经过）的点 k 的询问，如果有，这个询问的答案就是 k 在并查集中的根。最后将当前点所在的并查集和父节点所在的并查集合并。

此处应有演示动画 $\sim$

正确性证明：在进行搜索的过程中，从 u 到 v 的一条路径为 $u \to LCA(u,v) \to \cdots \to LCA(u,v) \to v$，即 u 在向上合并的时候会与 $LCA(u,v)$ 并入同一个并查集，但是由于 $LCA(u,v)$ 还没有完成搜索，所以 u 和 $fa_{LCA(u,v)}$ 不在一个并查集中，所以 $LCA(u,v)$ 就是 u 所在并查集的根。

注意：用 Tarjan 求解 LCA 是一个离线算法，即要在一开始就知道所有的询问。

「ZOJ 3195」Design the city

给出一棵大小为 n 的树（边有边权且均为正）和 m 个询问，每次询问会给出 3 个数 (a, b, c)，要求选出一些边使得 (a, b, c) 连通，求选出的边的边权之和最小值。

数据范围：$1 \leq n \leq 5 \times 10^4, 1 \leq m \leq 7 \times 10^4$。

「ZOJ 3195」Design the city

给出一棵大小为 n 的树（边有边权且均为正）和 m 个询问，每次询问会给出 3 个数 (a, b, c)，要求选出一些边使得 (a, b, c) 连通，求选出的边的边权之和最小值。

数据范围：$1 \le n \le 5 \times 10^4, 1 \le m \le 7 \times 10^4$。

记 $dis[i]$ 表示根节点到点 i 的距离（边权和），则 $dis(u, v) = dis[u] + dis[v] - 2dis[LCA(u, v)]$。

「ZOJ 3195」Design the city

给出一棵大小为 n 的树（边有边权且均为正）和 m 个询问，每次询问会给出 3 个数 (a, b, c)，要求选出一些边使得 (a, b, c) 连通，求选出的边的边权之和最小值。

数据范围：$1 \le n \le 5 \times 10^4, 1 \le m \le 7 \times 10^4$。

记 $dis[i]$ 表示根节点到点 i 的距离（边权和），则 $dis(u, v) = dis[u] + dis[v] - 2dis[LCA(u, v)]$。

通过画图不难发现答案为 $\frac{1}{2}[dis(a, b) + dis(b, c) + dis(c, a)]$。

「经典题」非严格次小生成树

给出一张 n 个点 m 条边图，求这张图中除了某一棵最小生成树以外的最小生成树。
数据范围：$1 \leq n \leq 10^5, 1 \leq m \leq 3 \times 10^5$。

「经典题」非严格次小生成树

给出一张 n 个点 m 条边图，求这张图中除了某一棵最小生成树以外的最小生成树。

数据范围：$1 \le n \le 10^5, 1 \le m \le 3\times10^5$。

（调整的思想）虽然次小生成树不是最小的，但与最小生成树的选边情况比较相似，即二者会有很多共用的边。进一步可以发现，最小生成树和次小生成树只会有 1 条边不一样。于是可以先对整张图跑一遍最小生成树，然后再从外面选一条边来替换原来生成树中的边。

「经典题」非严格次小生成树

给出一张 n 个点 m 条边图，求这张图中除了某一棵最小生成树以外的最小生成树。

数据范围：$1 \le n \le 10^5, 1 \le m \le 3\times10^5$。

（调整的思想）虽然次小生成树不是最小的，但与最小生成树的选边情况比较相似，即二者会有很多共用的边。进一步可以发现，最小生成树和次小生成树只会有 1 条边不一样。于是可以先对整张图跑一遍最小生成树，然后再从外面选一条边来替换原来生成树中的边。

具体来说，我们可以枚举没有在最小生成树上的每一条边，如果直接将这条边加在树上，那么必然会形成一个环，如果此时将环上的某一条边删去则又可以得到一棵生成树。为了保证最小，只需要将环上除了新加的边以外的边中最大的删掉即可。

「经典题」非严格次小生成树

给出一张 n 个点 m 条边图，求这张图中除了某一棵最小生成树以外的最小生成树。

数据范围：$1 \le n \le 10^5, 1 \le m \le 3\times10^5$。

（调整的思想）虽然次小生成树不是最小的，但与最小生成树的选边情况比较相似，即二者会有很多共用的边。进一步可以发现，最小生成树和次小生成树只会有 1 条边不一样。于是可以先对整张图跑一遍最小生成树，然后再从外面选一条边来替换原来生成树中的边。

具体来说，我们可以枚举没有在最小生成树上的每一条边，如果直接将这条边加在树上，那么必然会形成一个环，如果此时将环上的某一条边删去则又可以得到一棵生成树。为了保证最小，只需要将环上除了新加的边以外的边中最大的删掉即可。

于是问题转化为求某一条路径上的最长边。考虑在倍增的同时记 $maxx[i][j]$ 表示从 i 开始向上的 2^j 条边中最长边的长度。之后求解时只需要在用倍增算 LCA 的同时顺便更新最大值即可。

「经典题」非严格次小生成树

给出一张 n 个点 m 条边图，求这张图中除了某一棵最小生成树以外的最小生成树。

数据范围：$1 \le n \le 10^5, 1 \le m \le 3\times10^5$。

（调整的思想）虽然次小生成树不是最小的，但与最小生成树的选边情况比较相似，即二者会有很多共用的边。进一步可以发现，最小生成树和次小生成树只会有 1 条边不一样。于是可以先对整张图跑一遍最小生成树，然后再从外面选一条边来替换原来生成树中的边。

具体来说，我们可以枚举没有在最小生成树上的每一条边，如果直接将这条边加在树上，那么必然会形成一个环，如果此时将环上的某一条边删去则又可以得到一棵生成树。为了保证最小，只需要将环上除了新加的边以外的边中最大的删掉即可。

于是问题转化为求某一条路径上的最长边。考虑在倍增的同时记 $maxx[i][j]$ 表示从 i 开始向上的 2^j 条边中最长边的长度。之后求解时只需要在用倍增算 LCA 的同时顺便更新最大值即可。

如果要求严格次小（边权和不能与原来一样）生成树呢?

「经典题」非严格次小生成树

给出一张 n 个点 m 条边图，求这张图中除了某一棵最小生成树以外的最小生成树。

数据范围：$1 \le n \le 10^5, 1 \le m \le 3\times10^5$。

（调整的思想）虽然次小生成树不是最小的，但与最小生成树的选边情况比较相似，即二者会有很多共用的边。进一步可以发现，最小生成树和次小生成树只会有 1 条边不一样。于是可以先对整张图跑一遍最小生成树，然后再从外面选一条边来替换原来生成树中的边。

具体来说，我们可以枚举没有在最小生成树上的每一条边，如果直接将这条边加在树上，那么必然会形成一个环，如果此时将环上的某一条边删去则又可以得到一棵生成树。为了保证最小，只需要将环上除了新加的边以外的边中最大的删掉即可。

于是问题转化为求某一条路径上的最长边。考虑在倍增的同时记 $maxx[i][j]$ 表示从 i 开始向上的 2^j 条边中最长边的长度。之后求解时只需要在用倍增算 LCA 的同时顺便更新最大值即可。

如果要求严格次小（边权和不能与原来一样）生成树呢？同时再记录一条严格次长的边。

路径相交情况的判定：「洛谷 3398」仓鼠找 sugar

给出一棵 n 个点的树，有 q 组询问，每次询问给出两条简单路径，判断它们是否存在交点。

数据范围：$1 \leq n, q \leq 10^5$。

路径相交情况的判定：「洛谷 3398」仓鼠找 sugar

给出一棵 n 个点的树，有 q 组询问，每次询问给出两条简单路径，判断它们是否存在交点。

数据范围：$1 \leq n, q \leq 10^5$。

树上两条路径相交情况的判定

树上的两条路径有交等价于存在一条路径的两端点的最近公共祖先在另外一条路径上。

路径相交情况的判定：「洛谷 3398」仓鼠找 sugar

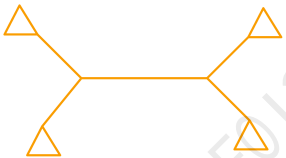
给出一棵 n 个点的树，有 q 组询问，每次询问给出两条简单路径，判断它们是否存在交点。

数据范围：$1 \le n, q \le 10^5$。

树上两条路径相交情况的判定

树上的两条路径有交等价于存在一条路径的两端点的最近公共祖先在另外一条路径上。

证明：



树上路径求交：「CF-GYM101908L」Subway Lines

给出一棵 n 个点的树，有 q 组询问，每次询问两条路径的交的长度。
数据范围：$1 \leq n, q \leq 10^5$。

树上路径求交：「CF-GYM101908L」Subway Lines

给出一棵 n 个点的树，有 q 组询问，每次询问两条路径的交的长度。
数据范围：$1 \le n, q \le 10^5$。

树上路径求交

令 $a = LCA(u_1, u_2), b = LCA(u_1, v_2), c = LCA(v_1, u_2), d = LCA(v_1, v_2)$，那么路径 $(u_1, v_1), (u_2, v_2)$ 的交就是 a, b, c, d 四个点中深度最大的两个点所构成的路径。

章节目录

复习 · 差分

基本操作

对于一个数列（也可以看成一维数组）a_n，定义它的差分数列 $b_n = a_n - a_{n-1}$。

复习 · 差分

基本操作

对于一个数列（也可以看成一维数组）a_n，定义它的差分数列 $b_n = a_n - a_{n-1}$。

使用情况

1. 题目中相邻数值的差值的重要性大于数值本身的重要性。

复习 · 差分

基本操作

对于一个数列（也可以看成一维数组）a_n，定义它的差分数列 $b_n = a_n - a_{n-1}$。

使用情况

1. 题目中相邻数值的差值的重要性大于数值本身的重要性。
2. 涉及大量的区间修改，且可以用差分降低时间复杂度（如区间加某个数）。

复习 · 差分

基本操作

对于一个数列（也可以看成一维数组）a_n，定义它的差分数列 $b_n = a_n - a_{n-1}$。

使用情况

1. 题目中相邻数值的差值的重要性大于数值本身的重要性。
2. 涉及大量的区间修改，且可以用差分降低时间复杂度（如区间加某个数）。
3. 若使用前缀和进行查询，则要求查询次数少或查询复杂度低（如可以使用数据结构完成）。

复习 · 差分

基本操作

对于一个数列（也可以看成一维数组）a_n，定义它的差分数列 $b_n = a_n - a_{n-1}$。

使用情况

1. 题目中相邻数值的差值的重要性大于数值本身的重要性。
2. 涉及大量的区间修改，且可以用差分降低时间复杂度（如区间加某个数）。
3. 若使用前缀和进行查询，则要求查询次数少或查询复杂度低（如可以使用数据结构完成）。
4. 不需要使用前缀和查询。

……

「JOI 2017 Final」焚风现象

给定一个数列 a_n，令

$$f(x)=\begin{cases}-xS, x>0\\-xT, x<0\end{cases} \tag{1}$$

并给出 Q 次操作，每次操作形如 l, r, d，要求将 $a_l \sim a_r$ 的每个数都加上 d（正负性未知），每次操作后输出 $\sum_{i=2}^{n} f(a_i - a_{i-1})$ 的值。

数据范围：$n, Q \le 10^6$。

「JOI 2017 Final」焚风现象

给定一个数列 a_n，令

$$f(x)=\begin{cases}-xS, x>0\\-xT, x<0\end{cases} \tag{1}$$

并给出 Q 次操作，每次操作形如 l,r,d，要求将 $a_l\sim a_r$ 的每个数都加上 d（正负性未知），每次操作后输出 $\sum_{i=2}^{n}f(a_i-a_{i-1})$ 的值。

数据范围：$n,Q\le 10^6$。

注意到这题只关心 a_i-a_{i-1} 的值而与 a_i 的值无关，于是可以考虑差分。

「JOI 2017 Final」焚风现象

给定一个数列 a_n，令

$$f(x)=\begin{cases}-xS, x>0\\-xT, x<0\end{cases} \tag{1}$$

并给出 Q 次操作，每次操作形如 l,r,d，要求将 $a_l\sim a_r$ 的每个数都加上 d（正负性未知），每次操作后输出 $\sum_{i=2}^{n}f(a_i-a_{i-1})$ 的值。

数据范围：$n,Q\le 10^6$。

注意到这题只关心 a_i-a_{i-1} 的值而与 a_i 的值无关，于是可以考虑差分。

在每次修改过程中，只有 $a_l-a_{l-1},a_{l+1}-a_l,a_r-a_{r-1},a_{r+1}-a_r$ 这四个值会受到影响，于是直接对这四个位置的值进行修改并更新答案即可。时间复杂度 $\mathbb{O}(n+Q)$。

「JOI 2021 Final」有趣的家庭菜园 4

给定一个数列 a_n，每次操作你可以选择一个区间，使其中的每个数加 1，求最少的操作次数使得存在一个整数 $m \in [1, n]$ 满足 $a_1 < a_2 < \cdots < a_{m-1} < a_m$ 且 $a_m > a_{m+1} > \cdots > a_{n-1} > a_n$。

数据范围：$2 \le n \le 2 \times 10^5$。

「JOI 2021 Final」有趣的家庭菜园 4

给定一个数列 a_n，每次操作你可以选择一个区间，使其中的每个数加 1，求最少的操作次数使得存在一个整数 $m \in [1, n]$ 满足 $a_1 < a_2 < \cdots < a_{m-1} < a_m$ 且 $a_m > a_{m+1} > \cdots > a_{n-1} > a_n$。

数据范围：$2 \le n \le 2 \times 10^5$。

这题中递增与递减的性质与 a_i 的具体数值无关，而由 $a_i - a_{i-1}$ 的正负性决定，于是考虑使用差分处理。

「JOI 2021 Final」有趣的家庭菜园 4

给定一个数列 a_n，每次操作你可以选择一个区间，使其中的每个数加 1，求最少的操作次数使得存在一个整数 $m \in [1, n]$ 满足 $a_1 < a_2 < \cdots < a_{m-1} < a_m$ 且 $a_m > a_{m+1} > \cdots > a_{n-1} > a_n$。

数据范围：$2 \le n \le 2 \times 10^5$。

这题中递增与递减的性质与 a_i 的具体数值无关，而由 $a_i - a_{i-1}$ 的正负性决定，于是考虑使用差分处理。

在差分之后，每一次对一个区间 $[l, r]$ 加 1 等价于对差分数组 b_n 做 $b_l+ = 1, b_{r+1}- = 1$ 的操作（从区间操作到单点操作的简化也证明了使用差分是正确的）。

「JOI 2021 Final」有趣的家庭菜园 4

给定一个数列 a_n，每次操作你可以选择一个区间，使其中的每个数加 1，求最少的操作次数使得存在一个整数 $m \in [1,n]$ 满足 $a_1 < a_2 < \cdots < a_{m-1} < a_m$ 且 $a_m > a_{m+1} > \cdots > a_{n-1} > a_n$。

数据范围：$2 \le n \le 2 \times 10^5$。

这题中递增与递减的性质与 a_i 的具体数值无关，而由 $a_i - a_{i-1}$ 的正负性决定，于是考虑使用差分处理。

在差分之后，每一次对一个区间 $[l,r]$ 加 1 等价于对差分数组 b_n 做 $b_l+=1, b_{r+1}-=1$ 的操作（从区间操作到单点操作的简化也证明了使用差分是正确的）。

于是可以枚举 m，题目变成求最少多少次操作后会出现 $b_2 \sim b_m > 0, b_{m+1} \sim b_n < 0$ 的情况，答案即为 $\max\{\sum_{i=2}^{m}(|b_i|+1)\times[b_i \le 0], \sum_{i=m+1}^{n}(|b_i|+1)\times[b_i \ge 0]\}$，其中 $[e]$ 当 e 为真时取 1，否则取 0。

树上差分：「JLOI 2014」松鼠的新家

给定一棵 n 个点的树和一个 n 的排列 a_n，要求走一个 $a_1 \to a_2 \to \cdots \to a_n$ 的路径，且任意的 $a_i \to a_{i+1}$ 的路径为简单路径。求每个点会被经过多少次。

数据范围：$2 \le n \le 3 \times 10^5$。

树上差分：「JLOI 2014」松鼠的新家

给定一棵 n 个点的树和一个 n 的排列 a_n，要求走一个 $a_1 \to a_2 \to \cdots \to a_n$ 的路径，且任意的 $a_i \to a_{i+1}$ 的路径为简单路径。求每个点会被经过多少次。

数据范围：$2 \le n \le 3 \times 10^5$。

如果这题改成在一个序列上操作，则可以用差分解决。于是我们考虑将差分的相关操作扩展到树上。

树上差分：「JLOI 2014」松鼠的新家

给定一棵 n 个点的树和一个 n 的排列 a_n，要求走一个 $a_1 \to a_2 \to \cdots \to a_n$ 的路径，且任意的 $a_i \to a_{i+1}$ 的路径为简单路径。求每个点会被经过多少次。

数据范围：$2 \le n \le 3 \times 10^5$。

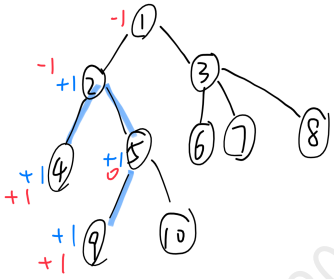
如果这题改成在一个序列上操作，则可以用差分解决。于是我们考虑将差分的相关操作扩展到树上。

树上差分

对于树上每个点，设它的权值为 a_i，子节点的集合为 S，那么差分结果 $b_i = a_i - \left(\sum_{j \in S} a_j\right)$。

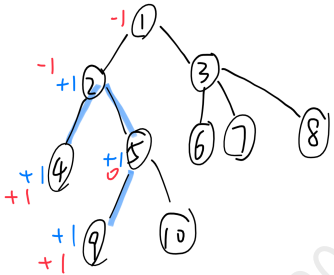
树上差分

下图中蓝色的部分为 a_i 的变化情况，红色的部分为 b_i 的变化情况。



树上差分

下图中蓝色的部分为 a_i 的变化情况，红色的部分为 b_i 的变化情况。



可以总结得出：如果在一条路径的每个点上加一个数 δ，那么在差分数组上产生的效果为：两端点的 b_i 各加 δ，LCA 及其父节点的 b_i 减去 δ。

「USACO 2015 Dec.」Max Flow

给定一棵 n 个点的树和 m 条个操作，每个操作形如在某一条路径的所有边上 $+1$，求操作之后每条边的权值。

数据范围：$2 \leq n \leq 5 \times 10^4, 1 \leq m \leq 10^5$。

「USACO 2015 Dec.」Max Flow

给定一棵 n 个点的树和 m 条个操作，每个操作形如在某一条路径的所有边上 $+1$，求操作之后每条边的权值。

数据范围：$2 \leq n \leq 5 \times 10^4, 1 \leq m \leq 10^5$。

这题只需要将每条边的权值记在它下方的点上即可转化为上一题的形式，不过要注意此时 LCA 的原始值不会被修改。

完结撒花