Graph Theory

StarRoute

2023 年 1 月 31 日

Will liuhengxi achieve LGM in grade 9?

By dXqwq, history, 10 days ago,

> As far as I know, he is in grade 9 now, which is one year younger than orzdevinwang.
>
> He is gaining rating in recent contests and previous LGMs in grade 9 are djq-cpp and orzdevinwang(comment below if you know more).
>
> Meanwhile I'm stuck between 2600~2700, hope I will become stronger next year o(>_<)o

+49 dXqwq 10 days ago 5

Comments (5)

Write comment?

N_z__ — 10 days ago, # Edit | +34

As a classmate of liuhengxi,I think he will.

→ Reply

moonrise__ — 10 days ago, # ^ | +68

As a classmate of liuhengxi, I think he wants a girlfriend.

→ Reply

User_Carrot — 10 days ago, # ^ | +20

As a classmate and also a fan of liuhengxi, I think he will. And he wants a girlfriend btw.

→ Reply

树的直径

1. 前言
2. 树上问题
   - 树的直径
   - LCA
   - 树链剖分
   - DSU on tree
   - 虚树
   - 例题
3. 抽象的东西
4. 图上问题

求法: 首先从任意节点 y 开始进行第一次 DFS，到达距离其最远的节点，记为 z，然后再从 z 开始做第二次 DFS，到达距离 z 最远的节点，记为 z'，则 $\delta(z, z')$ 即为树的直径。

求法: 首先从任意节点 y 开始进行第一次 DFS，到达距离其最远的节点，记为 z ，然后再从 z 开始做第二次 DFS，到达距离 z 最远的节点，记为 z' ，则 $\delta(z, z')$ 即为树的直径。

定理: 在一棵树上，从任意节点 y 开始进行一次 DFS，到达的距离其最远的节点 z 必为直径的一端。

求法：首先从任意节点 y 开始进行第一次 DFS，到达距离其最远的节点，记为 z，然后再从 z 开始做第二次 DFS，到达距离 z 最远的节点，记为 z'，则 $\delta(z, z')$ 即为树的直径。

定理：在一棵树上，从任意节点 y 开始进行一次 DFS，到达的距离其最远的节点 z 必为直径的一端。

证明：使用反证法。记出发节点为 y。设真实的直径是 $\delta(s, t)$，而从 y 进行的第一次 DFS 到达的距离其最远的节点 z 不为 t 或 s。共分三种情况：

1. 若 y 在 $\delta(s,t)$ 上：有 $\delta(y,z) > \delta(y,t) \Longrightarrow \delta(x,z) > \delta(x,t) \Longrightarrow \delta(s,z) > \delta(s,t)$ ，与 $\delta(s,t)$ 为树上任意两节点之间最长的简单路径矛盾。

1. 若 y 在 $\delta(s,t)$ 上：有 $\delta(y,z) > \delta(y,t) \implies \delta(x,z) > \delta(x,t) \implies \delta(s,z) > \delta(s,t)$，与 $\delta(s,t)$ 为树上任意两节点之间最长的简单路径矛盾。

2. 若 y 不在 $\delta(s,t)$ 上，且 $\delta(y,z)$ 与 $\delta(s,t)$ 存在重合路径：有 $\delta(y,z) > \delta(y,t) \implies \delta(x,z) > \delta(x,t) \implies \delta(s,z) > \delta(s,t)$，与 $\delta(s,t)$ 为树上任意两节点之间最长的简单路径矛盾。

1. 若 y 在 $\delta(s,t)$ 上：有 $\delta(y,z) > \delta(y,t) \Longrightarrow \delta(x,z) > \delta(x,t) \Longrightarrow \delta(s,z) > \delta(s,t)$，与 $\delta(s,t)$ 为树上任意两节点之间最长的简单路径矛盾。

2. 若 y 不在 $\delta(s,t)$ 上，且 $\delta(y,z)$ 与 $\delta(s,t)$ 存在重合路径：有 $\delta(y,z) > \delta(y,t) \Longrightarrow \delta(x,z) > \delta(x,t) \Longrightarrow \delta(s,z) > \delta(s,t)$，与 $\delta(s,t)$ 为树上任意两节点之间最长的简单路径矛盾。

3. 若 y 不在 $\delta(s,t)$ 上，且 $\delta(y,z)$ 与 $\delta(s,t)$ 不存在重合路径：有 $\delta(y,z) > \delta(y,t) \Longrightarrow \delta(x',z) > \delta(x',t) \Longrightarrow \delta(x,z) > \delta(x,t) \Longrightarrow \delta(s,z) > \delta(s,t)$，与 $\delta(s,t)$ 为树上任意两节点之间最长的简单路径矛盾。

1. 若 y 在 $\delta(s,t)$ 上：有 $\delta(y,z) > \delta(y,t) \implies \delta(x,z) > \delta(x,t) \implies \delta(s,z) > \delta(s,t)$，与 $\delta(s,t)$ 为树上任意两节点之间最长的简单路径矛盾。

2. 若 y 不在 $\delta(s,t)$ 上，且 $\delta(y,z)$ 与 $\delta(s,t)$ 存在重合路径：有 $\delta(y,z) > \delta(y,t) \implies \delta(x,z) > \delta(x,t) \implies \delta(s,z) > \delta(s,t)$，与 $\delta(s,t)$ 为树上任意两节点之间最长的简单路径矛盾。

3. 若 y 不在 $\delta(s,t)$ 上，且 $\delta(y,z)$ 与 $\delta(s,t)$ 不存在重合路径：有 $\delta(y,z) > \delta(y,t) \implies \delta(x',z) > \delta(x',t) \implies \delta(x,z) > \delta(x,t) \implies \delta(s,z) > \delta(s,t)$，与 $\delta(s,t)$ 为树上任意两节点之间最长的简单路径矛盾。

综上，三种情况下假设均会产生矛盾，故原定理得证。

上述求法及证明均不能在图中产生负权边，若存在，求法和证明均不适用。

上述求法及证明均不能在图中产生负权边，若存在，求法和证明均不适用。

此外，树的直径有个性质：若树上所有边边权均为正，则树的所有直径中点重合。这一性质也可以通过反证法证明。

上述求法及证明均不能在图中产生负权边，若存在，求法和证明均不适用。

此外，树的直径有个性质：若树上所有边边权均为正，则树的所有直径中点重合。这一性质也可以通过反证法证明。

前面讲数据结构时有个推论，设两不交点集的并的最远点对 (u, v) ，则 u, v 必在两个点集分别的最远点对 $(u1, v1)$ ，$(u2, v2)$ 四个点中。

求法有很多，这里主要讲四种较为常用的：

求法有很多，这里主要讲四种较为常用的：

分别是倍增法，树剖法，dfn 序法和欧拉序法。

倍增法

先让深度大的点倍增跳，直到与小的相等，然后一起倍增跳，直到同一个点。

时间复杂度: 预处理 $O(n \log n)$ ，单次询问 $O(\log n)$ 。

常数大，空间大，但是好理解。

倍增法

先让深度大的点倍增跳，直到与小的相等，然后一起倍增跳，直到同一个点。

时间复杂度：预处理 $O(n\log n)$ ，单次询问 $O(\log n)$ 。

常数大，空间大，但是好理解。

树剖法

若 u, v 在同一条重链上，返回深度小的点。

若 u, v 不在同一条重链上，则把更深的那条重链跳过去，可以发现 LCA 不变。

由于重链数是 $O(\log n)$ 级的，故此算法时间复杂度：预处理 $O(n\log n)$ ，单次询问 $O(\log n)$ ，但是较倍增法常数小。

dfn 序法

不妨设 $dfn_u < dfn_v$ ，分情况考虑:

若节点 u 在节点 v 的子树内，则 LCA 是节点 u 。

若不是，则发现 LCA 的 dfn 序不在 $[dfn_u, dfn_v]$ 内，但是我们发现在 $[dfn_u, dfn_v]$ 内深度最小的点的父亲恰好是 LCA。这可以使用 ST 表维护。

时间复杂度: 预处理 $O(n \log n)$ ，单次询问 $O(1)$ 。是一种非常优秀的求法。

dfn 序法

不妨设 $dfn_u < dfn_v$ ，分情况考虑：

若节点 u 在节点 v 的子树内，则 LCA 是节点 u 。

若不是，则发现 LCA 的 dfn 序不在 $[dfn_u, dfn_v]$ 内，但是我们发现在 $[dfn_u, dfn_v]$ 内深度最小的点的父亲恰好是 LCA。这可以使用 ST 表维护。

时间复杂度：预处理 $O(n \log n)$ ，单次询问 $O(1)$ 。是一种非常优秀的求法。

欧拉序法

和 dfn 序法本质类似，且 dfn 序法不仅预处理的时间常数小，空间常数也小，而且还更好写，也不需要担心忘记开两倍空间，可以说从各个方面吊打欧拉序。故在此不再赘述。

1 前言

2 树上问题
- 树的直径
- LCA
- 树链剖分
- DSU on tree
- 虚树
- 例题

3 抽象的东西

4 图上问题

重链剖分

给出一些定义：

- 定义 **重子节点**表示其子节点中子树最大的子结点。如果有多个子树最大的子结点，取其一。如果没有子节点，就无重子节点。

重链剖分

给出一些定义：

- 定义 **重子节点**表示其子节点中子树最大的子结点。如果有多个子树最大的子结点，取其一。如果没有子节点，就无重子节点。
- 定义 **轻子节点**表示剩余的所有子结点。

重链剖分

给出一些定义：

- 定义 **重子节点**表示其子节点中子树最大的子结点。如果有多个子树最大的子结点，取其一。如果没有子节点，就无重子节点。
- 定义 **轻子节点**表示剩余的所有子结点。
- 从这个结点到重子节点的边为 **重边**。

重链剖分

给出一些定义：

- 定义 **重子节点**表示其子节点中子树最大的子结点。如果有多个子树最大的子结点，取其一。如果没有子节点，就无重子节点。
- 定义 **轻子节点**表示剩余的所有子结点。
- 从这个结点到重子节点的边为 **重边**。
- 到其他轻子节点的边为 **轻边**。

重链剖分

给出一些定义:

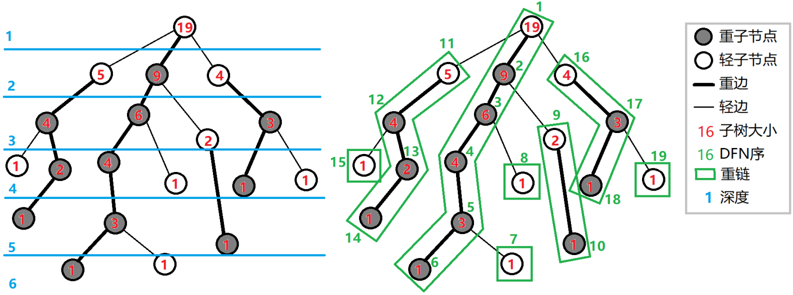
- 定义 **重子节点**表示其子节点中子树最大的子结点。如果有多个子树最大的子结点，取其一。如果没有子节点，就无重子节点。
- 定义 **轻子节点**表示剩余的所有子结点。
- 从这个结点到重子节点的边为 **重边**。
- 到其他轻子节点的边为 **轻边**。
- 若干条首尾衔接的重边构成 **重链**。

重链剖分

给出一些定义：

- 定义 **重子节点**表示其子节点中子树最大的子结点。如果有多个子树最大的子结点，取其一。如果没有子节点，就无重子节点。
- 定义 **轻子节点**表示剩余的所有子结点。
- 从这个结点到重子节点的边为 **重边**。
- 到其他轻子节点的边为 **轻边**。
- 若干条首尾衔接的重边构成 **重链**。
- 把落单的结点也当作重链，那么整棵树就被剖分成若干条重链。

如图：



重链剖分的性质如下：

- 每个节点属于且仅属于一条重链。重链开头的结点不一定是重子节点。

重链剖分的性质如下：

- 每个节点属于且仅属于一条重链。重链开头的结点不一定是重子节点。
- 所有的重链将整棵树完全剖分。重链内的 dfn 序是连续的。一颗子树内的 dfn 序是连续的。

重链剖分的性质如下：

- 每个节点属于且仅属于一条重链。重链开头的结点不一定是重子节点。
- 所有的重链将整棵树完全剖分。重链内的 dfn 序是连续的。一颗子树内的 dfn 序是连续的。
- 可以发现，当我们向下经过一条 **轻边**时，所在子树的大小至少会除以二。

重链剖分的性质如下：

- 每个节点属于且仅属于一条重链。重链开头的结点不一定是重子节点。
- 所有的重链将整棵树完全剖分。重链内的 dfn 序是连续的。一颗子树内的 dfn 序是连续的。
- 可以发现，当我们向下经过一条 **轻边**时，所在子树的大小至少会除以二。

因此，对于树上的任意一条路径，把它拆分成从 LCA 分别向两边往下走，分别最多走 $(\log n)$ 次，所以，树上的每条路径都可以被拆分成不超过 $O(\log n)$ 条重链。

链上维护
可以使用线段树、树状数组维护。
每次选择深度较大的链往上跳，直到两点在同一条链上。

链上维护

可以使用线段树、树状数组维护。
每次选择深度较大的链往上跳，直到两点在同一条链上。

子树内维护

子树中的结点的 DFS 序是连续的。
每一个结点记录所在子树连续区间末端的结点。
这样就把子树信息转化为连续的一段区间信息。

长链剖分

长链剖分本质上就是另外一种链剖分方式。

长链剖分

长链剖分本质上就是另外一种链剖分方式。

一般讲的树剖都指重链剖分，它可以用于维护树上路径的信息。

长链剖分

长链剖分本质上就是另外一种链剖分方式。

一般讲的树剖都指重链剖分，它可以用于维护树上路径的信息。

而长链剖分则是用于维护有关深度的信息。

长链剖分

长链剖分本质上就是另外一种链剖分方式。

一般讲的树剖都指重链剖分，它可以用于维护树上路径的信息。

而长链剖分则是用于维护有关深度的信息。

长链剖分的剖分方法与轻重链剖分极其相似。

长链剖分

长链剖分本质上就是另外一种链剖分方式。

一般讲的树剖都指重链剖分，它可以用于维护树上路径的信息。

而长链剖分则是用于维护有关深度的信息。

长链剖分的剖分方法与轻重链剖分极其相似。

只需要把以子树大小判断重儿子改成以节点深度判断即可。

长链剖分的性质如下：

- 长链剖分后，所有节点都仅属于一条链。

长链剖分的性质如下：

- 长链剖分后，所有节点都仅属于一条链。
- 任意节点 u 的第 k 级祖先 v 所在链的长度一定大于 k 。

长链剖分的性质如下：

- 长链剖分后，所有节点都仅属于一条链。
- 任意节点 u 的第 k 级祖先 v 所在链的长度一定大于 k 。
- 任意节点到达根节点经过的长链数是 $O(\sqrt{n})$ 级的。

长链剖分的性质如下：

- 长链剖分后，所有节点都仅属于一条链。
- 任意节点 u 的第 k 级祖先 v 所在链的长度一定大于 k 。
- 任意节点到达根节点经过的长链数是 $O(\sqrt{n})$ 级的。

故长链剖分多用于优化 DP。

引入

给出一棵 n 个节点以 1 为根的树，节点 u 的颜色为 c_u ，现在对于每个结点 u 询问 u 子树里一共出现了多少种不同的颜色。

引入

给出一棵 n 个节点以 1 为根的树，节点 u 的颜色为 c_u ，现在对于每个结点 u 询问 u 子树里一共出现了多少种不同的颜色。

$n \le 2 \times 10^5$。

求法

可以先预处理出每个节点子树的大小和它的重儿子，重儿子同树剖一样。

求法

可以先预处理出每个节点子树的大小和它的重儿子，重儿子同树剖一样。

用 $cnt[i]$ 表示颜色 i 的出现次数，$ans[u]$ 表示结点 u 的答案。

求法

可以先预处理出每个节点子树的大小和它的重儿子，重儿子同树剖一样。

用 $cnt[i]$ 表示颜色 i 的出现次数，$ans[u]$ 表示结点 u 的答案。

遍历一个节点 u ，步骤如下：

求法

可以先预处理出每个节点子树的大小和它的重儿子，重儿子同树剖一样。

用 $cnt[i]$ 表示颜色 i 的出现次数，$ans[u]$ 表示结点 u 的答案。

遍历一个节点 u ，步骤如下：

1. 先遍历 u 的轻儿子，并计算答案，但不保留遍历后它对 cnt 数组的影响；

求法

可以先预处理出每个节点子树的大小和它的重儿子，重儿子同树剖一样。

用 $cnt[i]$ 表示颜色 i 的出现次数，$ans[u]$ 表示结点 u 的答案。

遍历一个节点 u ，步骤如下：

1. 先遍历 u 的轻儿子，并计算答案，但不保留遍历后它对 cnt 数组的影响；
2. 遍历它的重儿子，保留它对 cnt 数组的影响；

求法

可以先预处理出每个节点子树的大小和它的重儿子，重儿子同树剖一样。

用 $cnt[i]$ 表示颜色 i 的出现次数，$ans[u]$ 表示结点 u 的答案。

遍历一个节点 u ，步骤如下：

1. 先遍历 u 的轻儿子，并计算答案，但不保留遍历后它对 cnt 数组的影响；
2. 遍历它的重儿子，保留它对 cnt 数组的影响；
3. 再次遍历 u 的轻儿子的子树结点，加入这些结点的贡献，以得到 u 的答案。

求法

可以先预处理出每个节点子树的大小和它的重儿子，重儿子同树剖一样。

用 $cnt[i]$ 表示颜色 i 的出现次数，$ans[u]$ 表示结点 u 的答案。

遍历一个节点 u ，步骤如下：

1. 先遍历 u 的轻儿子，并计算答案，但不保留遍历后它对 cnt 数组的影响；

2. 遍历它的重儿子，保留它对 cnt 数组的影响；

3. 再次遍历 u 的轻儿子的子树结点，加入这些结点的贡献，以得到 u 的答案。

通过执行这个过程，我们获得了这个节点所有子树的答案。

求法

可以先预处理出每个节点子树的大小和它的重儿子，重儿子同树剖一样。

用 $cnt[i]$ 表示颜色 i 的出现次数，$ans[u]$ 表示结点 u 的答案。

遍历一个节点 u，步骤如下：

1. 先遍历 u 的轻儿子，并计算答案，但不保留遍历后它对 cnt 数组的影响；
2. 遍历它的重儿子，保留它对 cnt 数组的影响；
3. 再次遍历 u 的轻儿子的子树结点，加入这些结点的贡献，以得到 u 的答案。

通过执行这个过程，我们获得了这个节点所有子树的答案。

注：可以水一些树套树的部分分，偏序树上莫队的 $O(n\sqrt{m})$ ！

1 前言

2 树上问题
- 树的直径
- LCA
- 树链剖分
- DSU on tree
- 虚树
- 例题

3 抽象的东西

4 图上问题

虚树

该树形结构的目的是浓缩信息，把一整颗大树浓缩成一颗小树。

该树形结构的目的是浓缩信息，把一整颗大树浓缩成一颗小树。

称某次询问中被选中的点为关键点。由于很多题目询问个数与 n 同阶，所以要让算法复杂度与关键点相关。

该树形结构的目的是浓缩信息，把一整颗大树浓缩成一颗小树。

称某次询问中被选中的点为关键点。由于很多题目询问个数与 n 同阶，所以要让算法复杂度与关键点相关。

显然只保存关键点信息不够，所以任意两个关键点的 LCA 也是需要保存的。

该树形结构的目的是浓缩信息，把一整颗大树浓缩成一颗小树。

称某次询问中被选中的点为关键点。由于很多题目询问个数与 n 同阶，所以要让算法复杂度与关键点相关。

显然只保存关键点信息不够，所以任意两个关键点的 LCA 也是需要保存的。

以下给出构造过程：

将关键点按 dfn 序排序。

将关键点按 dfn 序排序。

在关键点序列上，枚举相邻的两个数，两两求得 LCA 并且加入序列 A 中。

将关键点按 dfn 序排序。

在关键点序列上，枚举相邻的两个数，两两求得 LCA 并且加入序列 A 中。

把序列 A 按照 dfn 序从小到大排序并且去重。

将关键点按 dfn 序排序。

在关键点序列上，枚举相邻的两个数，两两求得 LCA 并且加入序列 A 中。

把序列 A 按照 dfn 序从小到大排序并且去重。

最后，在序列 A 上，枚举相邻的两个数 x, y ，求得它们的 LCA 并且连接 lca, y ，虚树就构造完成了。

将关键点按 dfn 序排序。

在关键点序列上，枚举相邻的两个数，两两求得 LCA 并且加入序列 A 中。

把序列 A 按照 dfn 序从小到大排序并且去重。

最后，在序列 A 上，枚举相邻的两个数 x, y ，求得它们的 LCA 并且连接 lca, y ，虚树就构造完成了。

复杂度 $O(m\log n)$ 。其中 m 为关键点数，n 为总点数。可以通过大部分题目。

将关键点按 dfn 序排序。

在关键点序列上，枚举相邻的两个数，两两求得 LCA 并且加入序列 A 中。

把序列 A 按照 dfn 序从小到大排序并且去重。

最后，在序列 A 上，枚举相邻的两个数 x, y ，求得它们的 LCA 并且连接 lca, y ，虚树就构造完成了。

复杂度 $O(m \log n)$ 。其中 m 为关键点数，n 为总点数。可以通过大部分题目。

还有一种方法是利用单调栈，在此不再赘述。

NOIP 2013 货车运输

Problem

A 国有 n 座城市，城市之间有 m 条双向道路，每一条道路对车辆限重。现在有 q 辆货车在运输货物，求每辆车在不超过车辆限重的情况下，最多能运多重的货物。

NOIP 2013 货车运输

Problem

A 国有 n 座城市，城市之间有 m 条双向道路, 每一条道路对车辆限重。现在有 q 辆货车在运输货物，求每辆车在不超过车辆限重的情况下，最多能运多重的货物。

可能存在重边，图不一定联通。

$(1 \le n \le 2 \times 10^5, 1 \le m \le 5 \times 10^5, 1 \le q \le 2 \times 10^5)$

NOIP 2013 货车运输

Solution

很简单啊。

NOIP 2013 货车运输

Solution

很简单啊。

对每个联通块，求出最大生成树。然后对于每个询问，倍增求 LCA 的时候顺便维护路径上边权的 $\min$ 即可。

NOIP 2013 货车运输

Solution

很简单啊。

对每个联通块，求出最大生成树。然后对于每个询问，倍增求 LCA 的时候顺便维护路径上边权的 $\min$ 即可。

时间复杂度 $O(n\log n+q\log n)$ 。

NOIP 2013 货车运输

Solution

很简单啊。

对每个联通块，求出最大生成树。然后对于每个询问，倍增求 LCA 的时候顺便维护路径上边权的 $\min$ 即可。

时间复杂度 $O(n \log n + q \log n)$ 。

原题 数据范围 $O(nq)$ 可过。

ZJOI2008 树的统计

Problem

对于一棵 n 个节点的树，执行操作如下：

ZJOI2008 树的统计

Problem

对于一棵 n 个节点的树，执行操作如下：

- 单点修改权值。
- 查询链上 $\max$ 。
- 查询链上和。

ZJOI2008 树的统计

Problem

对于一棵 n 个节点的树，执行操作如下：

- 单点修改权值。
- 查询链上 $\max$ 。
- 查询链上和。

保证 $1 \leq n \leq 30000$ $0 \leq q \leq 200000$ 。

ZJOI2008 树的统计

Solution

主要是查询比较困难。

ZJOI2008 树的统计

Solution

主要是查询比较困难。
首先将两个节点提到同一高度，然后将两个节点一起向上跳。

ZJOI2008 树的统计

Solution

主要是查询比较困难。

首先将两个节点提到同一高度，然后将两个节点一起向上跳。

在向上跳的过程中，如果当前节点在重链上，向上跳到重链顶端，如果当前节点不在重链上，向上跳一个节点，沿途更新/查询区间信息。

ZJOI2008 树的统计

Solution

主要是查询比较困难。

首先将两个节点提到同一高度，然后将两个节点一起向上跳。

在向上跳的过程中，如果当前节点在重链上，向上跳到重链顶端，如果当前节点不在重链上，向上跳一个节点，沿途更新/查询区间信息。

对于每个询问，最多经过 $O(\log n)$ 条重链，每条重链上线段树的复杂度为 $O(\log n)$ ，因此总时间复杂度为 $O(n\log n + q\log^2 n)$ 。

Codeforces 1009F

Problem

给定一棵以 1 为根，n 个节点的树。设 $d(u,x)$ 为 u 子树中到 u 距离为 x 的节点数。

Codeforces 1009F

Problem

给定一棵以 1 为根，n 个节点的树。设 $d(u, x)$ 为 u 子树中到 u 距离为 x 的节点数。

对于每个点，求一个最小的 k，使得 $d(u, k)$ 最大。

Codeforces 1009F

Problem

给定一棵以 1 为根，n 个节点的树。设 $d(u,x)$ 为 u 子树中到 u 距离为 x 的节点数。

对于每个点，求一个最小的 k，使得 $d(u,k)$ 最大。

$(1 \leq n \leq 1 \times 10^6)$。

Codeforces 1009F

Solution

设 $dp_{i,j}$ 表示在子树 i 内，和 i 距离为 j 的点数。

Codeforces 1009F

Solution

设 $dp_{i,j}$ 表示在子树 i 内，和 i 距离为 j 的点数。

考虑每次转移直接继承长儿子的 DP 数组和答案，并且考虑在此基础上进行更新。

Codeforces 1009F

Solution

设 $dp_{i,j}$ 表示在子树 i 内，和 i 距离为 j 的点数。

考虑每次转移直接继承长儿子的 DP 数组和答案，并且考虑在此基础上进行更新。

首先要将长儿子的 DP 数组前面插入一个元素 1，这代表着当前节点。

Codeforces 1009F

Solution

设 $dp_{i,j}$ 表示在子树 i 内，和 i 距离为 j 的点数。

考虑每次转移直接继承长儿子的 DP 数组和答案，并且考虑在此基础上进行更新。

首先要将长儿子的 DP 数组前面插入一个元素 1 ，这代表着当前节点。

然后将所有短儿子的 DP 数组暴力和当前节点的 DP 数组合并。

Codeforces 1009F

Solution

设 $dp_{i,j}$ 表示在子树 i 内，和 i 距离为 j 的点数。

考虑每次转移直接继承长儿子的 DP 数组和答案，并且考虑在此基础上进行更新。

首先要将长儿子的 DP 数组前面插入一个元素 1 ，这代表着当前节点。

然后将所有短儿子的 DP 数组暴力和当前节点的 DP 数组合并。

注意到因为短儿子的 DP 数组长度为短儿子所在长链长度，而所有长链长度和为 n 。

Codeforces 1009F

Solution

设 $dp_{i,j}$ 表示在子树 i 内，和 i 距离为 j 的点数。

考虑每次转移直接继承长儿子的 DP 数组和答案，并且考虑在此基础上进行更新。

首先要将长儿子的 DP 数组前面插入一个元素 1，这代表着当前节点。

然后将所有短儿子的 DP 数组暴力和当前节点的 DP 数组合并。

注意到因为短儿子的 DP 数组长度为短儿子所在长链长度，而所有长链长度和为 n。

时间复杂度为 $O(n)$。

Codeforces 600E

Problem

树的节点有颜色，一种颜色占领了一个子树，当且仅当没有其他颜色在这个子树中出现得比它多。

Codeforces 600E

Problem

树的节点有颜色，一种颜色占领了一个子树，当且仅当没有其他颜色在这个子树中出现得比它多。

求占领每个子树的所有颜色编号的和。

Codeforces 600E

Solution

对于一个点，首先遍历计算他所有轻儿子的答案，算完即清除。

Codeforces 600E

Solution

对于一个点，首先遍历计算他所有轻儿子的答案，算完即清除。

接下来计算重儿子的答案，并保留重儿子中的所有点的贡献，再暴力加入其他轻儿子的贡献。

Codeforces 600E

Solution

对于一个点，首先遍历计算他所有轻儿子的答案，算完即清除。

接下来计算重儿子的答案，并保留重儿子中的所有点的贡献，再暴力加入其他轻儿子的贡献。

此过程中只要保留子树内出现最多的颜色，回溯时记录即可。

SDOI2011 消耗战

Problem

一棵 n 个点的带边权的树，有 m 个询问，每次询问给出 k 个关键点，问切断若干边，使得 k 个关键点都不和根节点 1 联通的最小边权和。

SDOI2011 消耗战

Solution

考虑朴素 DP：

SDOI2011 消耗战

Solution

考虑朴素 DP：
若 v 不是关键点：$dp[u] = dp[u] + \min\{dp[v], w(u, v)\}$ 。

SDOI2011 消耗战

Solution

考虑朴素 DP：
若 v 不是关键点：$dp[u] = dp[u] + \min\{dp[v], w(u, v)\}$ 。
若 v 是关键点：$dp[u] = dp[u] + w(u, v)$ 。

SDOI2011 消耗战

Solution

考虑朴素 DP：

若 v 不是关键点： $dp[u]=dp[u]+\min\{dp[v],w(u,v)\}$ 。

若 v 是关键点： $dp[u]=dp[u]+w(u,v)$ 。

然后很显然可以建出虚树，在虚树上跑 DP ，复杂度为 $O(\sum_{i=1}^{m}k_i\log n)$ 。

HEOI2014 大工程

Problem

给定一棵 n 个点、边长为 1 的树，有 q 个询问，每次询问选中 k 个点，求：

HEOI2014 大工程

Problem

给定一棵 n 个点、边长为 1 的树，有 q 个询问，每次询问选中 k 个点，求：

- 两两树上路径的和。

HEOI2014 大工程

Problem

给定一棵 n 个点、边长为 1 的树，有 q 个询问，每次询问选中 k 个点，求：

- 两两树上路径的和。
- 两两树上路径的最小值。

HEOI2014 大工程

Problem

给定一棵 n 个点、边长为 1 的树，有 q 个询问，每次询问选中 k 个点，求：

- 两两树上路径的和。
- 两两树上路径的最小值。
- 两两树上路径的最大值。

HEOI2014 大工程

Solution

先建出虚树，然后对于三个问题分别考虑：

HEOI2014 大工程

Solution

先建出虚树，然后对于三个问题分别考虑:
求和:

HEOI2014 大工程

Solution

先建出虚树，然后对于三个问题分别考虑：
求和：
设 $sz[u]$ 为 u 子树内关键点个数。

HEOI2014 大工程

Solution

先建出虚树，然后对于三个问题分别考虑：
求和：
设 $sz[u]$ 为 u 子树内关键点个数。
计算一下贡献同时维护 $sz[u]$ 即可。

HEOI2014 大工程

Solution

先建出虚树，然后对于三个问题分别考虑：
求和：
设 $sz[u]$ 为 u 子树内关键点个数。
计算一下贡献同时维护 $sz[u]$ 即可。
max/min：

HEOI2014 大工程

Solution

先建出虚树，然后对于三个问题分别考虑：

求和：

设 $sz[u]$ 为 u 子树内关键点个数。

计算一下贡献同时维护 $sz[u]$ 即可。

max/min：

$\max$ ：子树中深度最大的关键点的深度 $+$ 子树中深度次大的关键点的深度。

HEOI2014 大工程

Solution

先建出虚树，然后对于三个问题分别考虑：

求和：

设 $sz[u]$ 为 u 子树内关键点个数。

计算一下贡献同时维护 $sz[u]$ 即可。

max/min：

$\max$ ：子树中深度最大的关键点的深度 $+$ 子树中深度次大的关键点的深度。

$\min$ ：子树中深度最小的关键点的深度 $+$ 子树中深度次小的关键点的深度。

HNOI2014 世界树

Problem

一棵树上 n 个点, 其中指定了 m 个关键点, 询问每个关键点所" 统治" 的结点，一个点仅会被离其最近的关键点所" 统治"(若距离相等, 则被标号最小的关键点" 统治")。

HNOI2014 世界树

Problem

一棵树上 n 个点，其中指定了 m 个关键点，询问每个关键点所" 统治" 的结点，一个点仅会被离其最近的关键点所" 统治"(若距离相等，则被标号最小的关键点" 统治")。

$1 \leq N \leq 3 \times 10^5, 1 \leq q \leq 3 \times 10^5, 1 \leq \sum_{i=1}^{q} m_i \leq 3 \times 10^5$ 。

HNOI2014 世界树

Solution

设 c_i 为节点 i 被 c_i 统治，d_i 记录 i 到 c_i 的距离。

HNOI2014 世界树

Solution

设 c_i 为节点 i 被 c_i 统治，d_i 记录 i 到 c_i 的距离。

朴素想法是对于一个节点 i，第一次先找其子树内离它最近的关键点，第二次向父节点方向更新答案。

HNOI2014 世界树

Solution

设 c_i 为节点 i 被 c_i 统治，d_i 记录 i 到 c_i 的距离。

朴素想法是对于一个节点 i，第一次先找其子树内离它最近的关键点，第二次向父节点方向更新答案。

发现这东西很虚树。

HNOI2014 世界树

Solution

设 c_i 为节点 i 被 c_i 统治，d_i 记录 i 到 c_i 的距离。

朴素想法是对于一个节点 i ，第一次先找其子树内离它最近的关键点，第二次向父节点方向更新答案。

发现这东西很虚树。

建出虚树，发现虚树上的一条边在原树种对应一条链 (包括链上的子树)，这条链上的点上一定是上半部分的最近距离在上面那个点，下半部分的最近距离在下面那个点。

HNOI2014 世界树

Solution

设 c_i 为节点 i 被 c_i 统治，d_i 记录 i 到 c_i 的距离。

朴素想法是对于一个节点 i ，第一次先找其子树内离它最近的关键点，第二次向父节点方向更新答案。

发现这东西很虚树。

建出虚树，发现虚树上的一条边在原树种对应一条链 (包括链上的子树)，这条链上的点上一定是上半部分的最近距离在上面那个点，下半部分的最近距离在下面那个点。

所以使用倍增法找出上下关键点统治的分界点即可。时间复杂度 $O(m\log n)$ 。


【管理员】NZ
有无老哥有会议号
会议详情
宁波大学-胡老师预定的会议
13:00
15:15
2023年01月17日
(GMT+08:00)
2023年01月17日
会议号 590 502 028
发起人 宁波大学-胡老师
会议密码 713461
文档
添加
进入会议
【管理员】looking_for_girlfriend
我要进去找妹子

求法

Kruskal 重构树是在 Kruskal 算法加边时，新建一个点，将这个点的点权设为新加的边的权值，同时将两个集合的根节点分别设为新建的点的左儿子和右儿子。

求法

Kruskal 重构树是在 Kruskal 算法加边时，新建一个点，将这个点的点权设为新加的边的权值，同时将两个集合的根节点分别设为新建的点的左儿子和右儿子。

然后，将两个集合和新建点合并成一个集合，将新建点设为根。

性质

Kruskal 重构树性质如下：

性质

Kruskal 重构树性质如下：

- 该树满足二叉堆的性质，节点个数为 $2n-1$ ，深度最大为 n 。

性质

Kruskal 重构树性质如下：

- 该树满足二叉堆的性质，节点个数为 $2n-1$ ，深度最大为 n 。
- 代表原树中的点的节点全是叶子节点，其余节点都代表了连接其二子节点的边的权值。

性质

Kruskal 重构树性质如下：

- 该树满足二叉堆的性质，节点个数为 $2n-1$ ，深度最大为 n 。
- 代表原树中的点的节点全是叶子节点，其余节点都代表了连接其二子节点的边的权值。
- 原图中两点间的所有简单路径上最大边权的最小值 (或最小边权的最大值) 等于 Kruskal 重构树上两点之间的 LCA 的权值。

NOIP 2013 货车运输

NOIP 2013 货车运输

前面的这道题可以转化为求 x 号城市到 y 号城市所有简单路径上小边权的最大值。

NOIP 2013 货车运输

前面的这道题可以转化为求 x 号城市到 y 号城市所有简单路径上小边权的最大值。

所以也可以用 Kruskal 重构树解决。

定义

给出 n 个集合，每个集合有两个元素，已知若干个 $< a, b >$ ，表示 a 与 b 矛盾（其中 a 与 b 属于不同的集合）。然后从每个集合选择一个元素，判断能否一共选 n 个两两不矛盾的元素。

定义

给出 n 个集合，每个集合有两个元素，已知若干个 $<a,b>$ ，表示 a 与 b 矛盾（其中 a 与 b 属于不同的集合)。然后从每个集合选择一个元素，判断能否一共选 n 个两两不矛盾的元素。

求法

假设有 $a1, a2$ 和 $b1, b2$ 两对，已知 $a1$ 和 $b2$ 间有矛盾，两者中必须选一个，所以建两条有向边 $(a1, b1)$ 和 $(b2, a2)$ 表示选了 $a1$ 则必须选 $b1$ ，选了 $b2$ 则必须选 $a2$ 才能够自洽。

定义

给出 n 个集合，每个集合有两个元素，已知若干个 $<a, b>$ ，表示 a 与 b 矛盾（其中 a 与 b 属于不同的集合)。然后从每个集合选择一个元素，判断能否一共选 n 个两两不矛盾的元素。

求法

假设有 $a1, a2$ 和 $b1, b2$ 两对，已知 $a1$ 和 $b2$ 间有矛盾，两者中必须选一个，所以建两条有向边 $(a1, b1)$ 和 $(b2, a2)$ 表示选了 $a1$ 则必须选 $b1$ ，选了 $b2$ 则必须选 $a2$ 才能够自洽。

然后跑一遍 Tarjan 判断是否有一个集合中的两个元素在同一个 SCC 中，若有则无合法方案，否则构造方案，只需要把几个不矛盾的 SCC 拼起来就好了。

JSOI2010 满汉全席

Problem

满汉全席大赛，每位参赛的选手可以得到 n 种材料，共有 m 位评审员分别把关。

JSOI2010 满汉全席

Problem

满汉全席大赛，每位参赛的选手可以得到 n 种材料，共有 m 位评审员分别把关。

每一位评审员对于满汉全席有各自独特的见解，只要参赛者能在这两种材料的做法中，其中一个符合评审的喜好即可通过该评审的审查。

JSOI2010 满汉全席

Problem

满汉全席大赛，每位参赛的选手可以得到 n 种材料，共有 m 位评审员分别把关。

每一位评审员对于满汉全席有各自独特的见解，只要参赛者能在这两种材料的做法中，其中一个符合评审的喜好即可通过该评审的审查。

判断是否存在一种做菜方法通过所有评审员的审查。

JSOI2010 满汉全席

Problem

满汉全席大赛，每位参赛的选手可以得到 n 种材料，共有 m 位评审员分别把关。

每一位评审员对于满汉全席有各自独特的见解，只要参赛者能在这两种材料的做法中，其中一个符合评审的喜好即可通过该评审的审查。

判断是否存在一种做菜方法通过所有评审员的审查。

$(1 \leq n \leq 100\ ,\ 1 \leq m \leq 1000)$

JSOI2010 满汉全席

Solution

每个评委的限制条件都可以看成或。

JSOI2010 满汉全席

Solution

每个评委的限制条件都可以看成或。
对于材料 i ，记节点 i 表示满式做法，节点 $i+n$ 表示汉式做法。

JSOI2010 满汉全席

Solution

每个评委的限制条件都可以看成或。
对于材料 i，记节点 i 表示满式做法，节点 $i+n$ 表示汉式做法。
所以对每个限制条件，分情况建边：

JSOI2010 满汉全席

Solution

每个评委的限制条件都可以看成或。
对于材料 i ，记节点 i 表示满式做法，节点 $i+n$ 表示汉式做法。
所以对每个限制条件，分情况建边：

- h_i, h_j ： 连 $(i, j+n)$ 和 $(j, i+n)$ 。

JSOI2010 满汉全席

Solution

每个评委的限制条件都可以看成或。
对于材料 i ，记节点 i 表示满式做法，节点 $i+n$ 表示汉式做法。
所以对每个限制条件，分情况建边：

- h_i, h_j ： 连 $(i, j+n)$ 和 $(j, i+n)$ 。
- h_i, m_j ： 连 (i, j) 和 $(j+n, i+n)$ 。

JSOI2010 满汉全席

Solution

每个评委的限制条件都可以看成或。
对于材料 i ，记节点 i 表示满式做法，节点 $i+n$ 表示汉式做法。
所以对每个限制条件，分情况建边：

- h_i, h_j ： 连 $(i, j+n)$ 和 $(j, i+n)$ 。
- h_i, m_j ： 连 (i, j) 和 $(j+n, i+n)$ 。
- m_i, h_j ： 连 $(i+n, j+n)$ 和 (j, i) 。

JSOI2010 满汉全席

Solution

每个评委的限制条件都可以看成或。
对于材料 i，记节点 i 表示满式做法，节点 $i+n$ 表示汉式做法。
所以对每个限制条件，分情况建边：

- h_i, h_j：连 $(i, j+n)$ 和 $(j, i+n)$。
- h_i, m_j：连 (i, j) 和 $(j+n, i+n)$。
- m_i, h_j：连 $(i+n, j+n)$ 和 (j, i)。
- m_i, m_j：连 $(i+n, j)$ 和 $(j+n, i)$。

JSOI2010 满汉全席

Solution

每个评委的限制条件都可以看成或。
对于材料 i，记节点 i 表示满式做法，节点 $i+n$ 表示汉式做法。
所以对每个限制条件，分情况建边：

- h_i, h_j：连 $(i, j+n)$ 和 $(j, i+n)$。
- h_i, m_j：连 (i, j) 和 $(j+n, i+n)$。
- m_i, h_j：连 $(i+n, j+n)$ 和 (j, i)。
- m_i, m_j：连 $(i+n, j)$ 和 $(j+n, i)$。

然后跑 Tarjan 判断 i 和 $i+n$ 是否在同一个 SCC 中即可。

1 前言

2 树上问题

3 抽象的东西

4 图上问题
- Kruskal 重构树
- 2-SAT
- 二分图
- 网络流
- 例题

5 Thanks

定义

二分图是指在一张图上存在一种划分，使得该图被划分成二集合 A, B ，不存在边的两端点同属于一个集合。

定义

二分图是指在一张图上存在一种划分，使得该图被划分成二集合 A, B ，不存在边的两端点同属于一个集合。
有几个显然的性质：

定义

二分图是指在一张图上存在一种划分，使得该图被划分成二集合 A, B ，不存在边的两端点同属于一个集合。
有几个显然的性质：
1. 对图可以进行黑白染色。

定义

二分图是指在一张图上存在一种划分，使得该图被划分成二集合 A, B ，不存在边的两端点同属于一个集合。

有几个显然的性质：

1. 对图可以进行黑白染色。
2. 图中不存在奇环。

定义

二分图是指在一张图上存在一种划分，使得该图被划分成二集合 A, B，不存在边的两端点同属于一个集合。
有几个显然的性质：
1. 对图可以进行黑白染色。
2. 图中不存在奇环。

判定方法：直接对图遍历，发现奇环就不是二分图，反之成立。

二分图最大匹配

问题：给定一个二分图 G ，要求选出一些边，使得这些边没有公共点，使选出的边数最大。

二分图最大匹配

问题：给定一个二分图 G ，要求选出一些边，使得这些边没有公共点，使选出的边数最大。
求法：不妨从左边的未匹配点找增广路。

二分图最大匹配

问题：给定一个二分图 G，要求选出一些边，使得这些边没有公共点，使选出的边数最大。

求法：不妨从左边的未匹配点找增广路。

注意到增广路上的第奇数条边都是非匹配边，第偶数条边都是匹配边，于是左到右都是非匹配边，右到左都是匹配边。

二分图最大匹配

问题：给定一个二分图 G ，要求选出一些边，使得这些边没有公共点，使选出的边数最大。

求法：不妨从左边的未匹配点找增广路。

注意到增广路上的第奇数条边都是非匹配边，第偶数条边都是匹配边，于是左到右都是非匹配边，右到左都是匹配边。

给二分图定向，故此问题等价给定起始点 s 能否走到终点 t ，只要从起始点开始 DFS 遍历，直到找到某个未匹配点。

二分图最大匹配

问题：给定一个二分图 G ，要求选出一些边，使得这些边没有公共点，使选出的边数最大。

求法：不妨从左边的未匹配点找增广路。

注意到增广路上的第奇数条边都是非匹配边，第偶数条边都是匹配边，于是左到右都是非匹配边，右到左都是匹配边。

给二分图定向，故此问题等价给定起始点 s 能否走到终点 t ，只要从起始点开始 DFS 遍历，直到找到某个未匹配点。

以上成为匈牙利算法。但是此方法复杂度为 $O(nm)$ ，不够优秀，后面讲网络流时会补充。

König 定理

最小点覆盖：选最少的点，满足每条边至少有一个端点被选。

König 定理

最小点覆盖：选最少的点，满足每条边至少有一个端点被选。
二分图中，最小点覆盖 = 最大匹配。

König 定理

最小点覆盖：选最少的点，满足每条边至少有一个端点被选。
二分图中，最小点覆盖 $=$ 最大匹配。
证明:

König 定理

最小点覆盖：选最少的点，满足每条边至少有一个端点被选。
二分图中，最小点覆盖 $=$ 最大匹配。
证明：
将二分图点集分成左右两个集合，使得所有边的两个端点都不在一个集合。

König 定理

最小点覆盖：选最少的点，满足每条边至少有一个端点被选。

二分图中，最小点覆盖 = 最大匹配。

证明：

将二分图点集分成左右两个集合，使得所有边的两个端点都不在一个集合。

考虑如下构造：从左侧未匹配的节点出发，按照匈牙利算法中增广路的方式走，即先走一条未匹配边，再走一条匹配边。由于已经求出了最大匹配，所以这样的增广路一定以匹配边结束。在所有经过这样“增广路”的节点上打标记。则最后构造的集合是：所有左侧未打标记的节点和所有右侧打了标记的节点。

König 定理

首先，易证这个集合的大小等于最大匹配。打了标记的节点一定都是匹配边上的点，一条匹配的边两侧一定都有标记（在增广路上）或都没有标记，所以两个节点中必然有一个被选中。

König 定理

首先，易证这个集合的大小等于最大匹配。打了标记的节点一定都是匹配边上的点，一条匹配的边两侧一定都有标记（在增广路上）或都没有标记，所以两个节点中必然有一个被选中。

其次，这个集合是一个点覆盖。一条匹配边一定有一个点被选中，而一条未匹配的边一定是增广路的一部分，而右侧端点也一定被选中。

König 定理

首先，易证这个集合的大小等于最大匹配。打了标记的节点一定都是匹配边上的点，一条匹配的边两侧一定都有标记（在增广路上）或都没有标记，所以两个节点中必然有一个被选中。

其次，这个集合是一个点覆盖。一条匹配边一定有一个点被选中，而一条未匹配的边一定是增广路的一部分，而右侧端点也一定被选中。

同时，不存在更小的点覆盖。为了覆盖最大匹配的所有边，至少要有最大匹配边数的点数。

König 定理

首先，易证这个集合的大小等于最大匹配。打了标记的节点一定都是匹配边上的点，一条匹配的边两侧一定都有标记（在增广路上）或都没有标记，所以两个节点中必然有一个被选中。

其次，这个集合是一个点覆盖。一条匹配边一定有一个点被选中，而一条未匹配的边一定是增广路的一部分，而右侧端点也一定被选中。

同时，不存在更小的点覆盖。为了覆盖最大匹配的所有边，至少要有最大匹配边数的点数。

补充：最大独立集：选最多的点，满足两两之间没有边相连。最大独立集 $= n-$ 最小点覆盖。

二分图最大权匹配

定义：二分图的最大权匹配是指二分图中边权和最大的匹配。

二分图最大权匹配

定义：二分图的最大权匹配是指二分图中边权和最大的匹配。
求法见后。

最大流

一些定义：

最大流

一些定义：

1. 残量网络：一条边的剩余容量 $c_f(u, v)$ ，表示的是这条边的容量与流量之差，即 $c_f(u, v) = c(u, v) - f(u, v)$ 。

最大流

一些定义：

1. 残量网络：一条边的剩余容量 $c_f(u,v)$ ，表示的是这条边的容量与流量之差，即 $c_f(u,v)=c(u,v)-f(u,v)$ 。

残量网络 G_f 是网络 G 中所有结点和剩余容量大于 0 的边构成的子图，即 $G_f=(V_f=V,E_f=\{(u,v)\in E,c_f(u,v)>0\})$ 。

最大流

一些定义：

1. 残量网络：一条边的剩余容量 $c_f(u,v)$ ，表示的是这条边的容量与流量之差，即 $c_f(u,v) = c(u,v) - f(u,v)$ 。

残量网络 G_f 是网络 G 中所有结点和剩余容量大于 0 的边构成的子图，即 $G_f = (V_f = V, E_f = \{(u,v) \in E, c_f(u,v) > 0\})$ 。

注：残量网络中包括了那些还剩了流量空间的边构成的图，也包括反向边。

最大流

一些定义：

1. 残量网络：一条边的剩余容量 $c_f(u,v)$ ，表示的是这条边的容量与流量之差，即 $c_f(u,v)=c(u,v)-f(u,v)$ 。

残量网络 G_f 是网络 G 中所有结点和剩余容量大于 0 的边构成的子图，即 $G_f=(V_f=V,E_f=\{(u,v)\in E,c_f(u,v)>0\})$ 。

注：残量网络中包括了那些还剩了流量空间的边构成的图，也包括反向边。

2. 增广路：在残量网络 G_f 中，一条从源点到汇点的路径被称为增广路。

EK

从源点一直 BFS 走来走去，碰到汇点就停，然后增广。

EK

从源点一直 BFS 走来走去，碰到汇点就停，然后增广。

增广的时候要注意建造反向边，原因是这条路不一定是最优的，这样子程序可以进行反悔。

EK

从源点一直 BFS 走来走去，碰到汇点就停，然后增广。

增广的时候要注意建造反向边，原因是这条路不一定是最优的，这样子程序可以进行反悔。

EK 算法的复杂度为 $O(nm^2)$ 。不够优秀。

Dinic

过程如下：每次增广前，我们先用 BFS 来分层。

Dinic

过程如下：每次增广前，我们先用 BFS 来分层。
设源点的层数为 0 ，那么一个点的层数是它离源点的最近距离。

Dinic

过程如下：每次增广前，我们先用 BFS 来分层。
设源点的层数为 0 ，那么一个点的层数是它离源点的最近距离。
通过分层，如果汇点层数不存在，即可停止增广，且确保我们找到的增广路是最短的。(证明自行查阅)

Dinic

过程如下：每次增广前，我们先用 BFS 来分层。

设源点的层数为 0 ，那么一个点的层数是它离源点的最近距离。

通过分层，如果汇点层数不存在，即可停止增广，且确保我们找到的增广路是最短的。(证明自行查阅)

Dinic 有两个优化：多路优化和当前弧优化。

Dinic

多路增广：每次找到一条增广路的时候，可以利用残余部分流量，再找出一条增广路。这样就可以在一次 DFS 中找出多条增广路，大大提高了算法的效率。

Dinic

多路增广：每次找到一条增广路的时候，可以利用残余部分流量，再找出一条增广路。这样就可以在一次 DFS 中找出多条增广路，大大提高了算法的效率。

当前弧优化：如果一条边已经被增广过，那么它就没有可能被增广第二次。那么，我们下一次进行增广的时候，就可以不必再走那些已经被增广过的边。

Dinic

多路增广：每次找到一条增广路的时候，可以利用残余部分流量，再找出一条增广路。这样就可以在一次 DFS 中找出多条增广路，大大提高了算法的效率。

当前弧优化：如果一条边已经被增广过，那么它就没有可能被增广第二次。那么，我们下一次进行增广的时候，就可以不必再走那些已经被增广过的边。

优化后的时间复杂度为 $O(n^2m)$ 。稀疏图与 EK 相当，但在稠密图有很大优势。

其余网络流做法

还有很多高级做法，例如：MPM，ISAP，HLPP。

其余网络流做法

还有很多高级做法，例如：MPM，ISAP，HLPP。
这些做法比较复杂，可以课后自行查阅和理解。

最小割

定义：

最小割

定义:

割: 对于一个图 $G=(V,E)$ ，称一种点的划分方式为割，当且仅当将所有的点划分为 S 和 $T=V-S$ 两个集合，其中源点 $s \in S$ ，汇点 $t \in T$ 。

最小割

定义：

割：对于一个图 $G=(V,E)$，称一种点的划分方式为割，当且仅当将所有的点划分为 S 和 $T=V-S$ 两个集合，其中源点 $s\in S$，汇点 $t\in T$。

割的容量：定义割 (S,T) 的容量 $c(S,T)$ 表示所有原图中从 S 到 T 的边的容量之和。

最小割

定义：

割：对于一个图 $G=(V,E)$，称一种点的划分方式为割，当且仅当将所有的点划分为 S 和 $T=V-S$ 两个集合，其中源点 $s\in S$，汇点 $t\in T$。

割的容量：定义割 (S,T) 的容量 $c(S,T)$ 表示所有原图中从 S 到 T 的边的容量之和。

最小割：即图 G 中容量最小的割。

最大流最小割定理

定理： $f(s,t)_{\max} = c(s,t)_{\min}$

最大流最小割定理

定理：$f(s,t)_{\max} = c(s,t)_{\min}$
对于任意一个可行流 $f(s,t)$ 的割 (S,T) ，我们可以得到：

最大流最小割定理

定理：$f(s,t)_{\max} = c(s,t)_{\min}$
对于任意一个可行流 $f(s,t)$ 的割 (S,T)，我们可以得到：

$$f(s,t) = S_{\text{出边的总流量}} - S_{\text{入边的总流量}} \le S_{\text{出边的总流量}} = c(s,t)$$

最大流最小割定理

定理：$f(s,t)_{\max} = c(s,t)_{\min}$
对于任意一个可行流 $f(s,t)$ 的割 (S,T)，我们可以得到：

$$f(s,t) = S_{\text{出边的总流量}} - S_{\text{入边的总流量}} \le S_{\text{出边的总流量}} = c(s,t)$$

如果求出了最大流，那么残量网络中一定不存在 s 到 t 的增广路经，也就是 S 的出边一定是满流，S 的入边一定是零流，于是有：

最大流最小割定理

定理：$f(s,t)_{\max} = c(s,t)_{\min}$
对于任意一个可行流 $f(s,t)$ 的割 (S,T)，我们可以得到：

$$f(s,t) = S_{\text{出边的总流量}} - S_{\text{入边的总流量}} \le S_{\text{出边的总流量}} = c(s,t)$$

如果求出了最大流，那么残量网络中一定不存在 s 到 t 的增广路经，也就是 S 的出边一定是满流，S 的入边一定是零流，于是有：

$$f(s,t) = S_{\text{出边的总流量}} - S_{\text{入边的总流量}} = S_{\text{出边的总流量}} = c(s,t)$$

最大流最小割定理

定理： $f(s,t)_{\max} = c(s,t)_{\min}$
对于任意一个可行流 $f(s,t)$ 的割 (S,T) ，我们可以得到：

$$f(s,t) = S_{\text{出边的总流量}} - S_{\text{入边的总流量}} \le S_{\text{出边的总流量}} = c(s,t)$$

如果求出了最大流，那么残量网络中一定不存在 s 到 t 的增广路经，也就是 S 的出边一定是满流， S 的入边一定是零流，于是有：

$$f(s,t) = S_{\text{出边的总流量}} - S_{\text{入边的总流量}} = S_{\text{出边的总流量}} = c(s,t)$$

结合前面的不等式，我们可以知道此时 f 已经达到最大。

费用流

定义：给定一个网络 $G=(V,E)$，每条边有容量限制 $c(u,v)$ 和一个单位流量的费用 $w(u,v)$。

费用流

定义：给定一个网络 $G=(V,E)$，每条边有容量限制 $c(u,v)$ 和一个单位流量的费用 $w(u,v)$。

则该网络中总花费最小的最大流称为 **最小费用最大流**，即在最大化 $\sum_{(s,v)\in E} f(s,v)$ 的前提下最小化 $\sum_{(u,v)\in E} f(u,v)\times w(u,v)$。

费用流

定义：给定一个网络 $G=(V,E)$ ，每条边有容量限制 $c(u,v)$ 和一个单位流量的费用 $w(u,v)$ 。

则该网络中总花费最小的最大流称为 **最小费用最大流**，即在最大化 $\sum_{(s,v)\in E} f(s,v)$ 的前提下最小化 $\sum_{(u,v)\in E} f(u,v)\times w(u,v)$ 。

求法：每次寻找单位费用最小的增广路进行增广，直到图上不存在增广路为止。故只需将 EK 算法或 Dinic 算法中找增广路的过程，替换为用最短路算法寻找单位费用最小的增广路即可。

故二分图最大权匹配可以建立一个源点和汇点。

故二分图最大权匹配可以建立一个源点和汇点。

将从源点向图的前一半连边，流量为 1 ，费用为 0 ，将图的另一半向汇点连边，边的信息与前者相同。

故二分图最大权匹配可以建立一个源点和汇点。

将从源点向图的前一半连边，流量为 1 ，费用为 0 ，将图的另一半向汇点连边，边的信息与前者相同。

对原图中的边流量为 1 ，费用为边权。跑个 MCMF 即可得到答案。

1 前言

2 树上问题

3 抽象的东西

4 图上问题
- Kruskal 重构树
- 2-SAT
- 二分图
- 网络流
- 例题

5 Thanks

Codeforces 1706E

Problem

给出 n 个点，m 条边的不带权连通无向图，q 次询问至少要加完编号前多少的边，才能使得 $[l, r]$ 中的所有点两两连通。t 组数据。

Codeforces 1706E

Problem

给出 n 个点，m 条边的不带权连通无向图，q 次询问至少要加完编号前多少的边，才能使得 $[l,r]$ 中的所有点两两连通。t 组数据。

$(2 \le \sum n \le 10^5$，$1 \le \sum m, \sum q \le 2 \times 10^5)$

Codeforces 1706E

Solution

将一条边的边权定义为该边的编号。

Codeforces 1706E

Solution

将一条边的边权定义为该边的编号。
要使 $[l, r]$ 联通，等价于使 $\forall i \in [l, r-1], [i, i+1]$ 联通。

Codeforces 1706E

Solution

将一条边的边权定义为该边的编号。
要使 $[l, r]$ 联通，等价于使 $\forall i \in [l, r-1], [i, i+1]$ 联通。
求出每个小区间的答案，随便整个 DS 维护就行了。

POJ3683 牧师忙碌日

Problem

n 对夫妇结婚，每次婚礼持续 d 时间，从 s 时间到 t 时间之间举行，只能选择从 s 到 $s+d$ 或 $t-d$ 到 t 时间两个时间段举行。现在有一个神父，问他有没有可能参加所有夫妇的婚礼，要求时间段完整且任意两对夫妇婚礼时间不重叠，若可以，则输出一个可行的方案。

POJ3683 牧师忙碌日

Problem

n 对夫妇结婚，每次婚礼持续 d 时间，从 s 时间到 t 时间之间举行，只能选择从 s 到 $s+d$ 或 $t-d$ 到 t 时间两个时间段举行。现在有一个神父，问他有没有可能参加所有夫妇的婚礼，要求时间段完整且任意两对夫妇婚礼时间不重叠，若可以，则输出一个可行的方案。

$(1 \leq N \leq 1 \times 10^3)$。

POJ3683 牧师忙碌日

Solution

将每个时间段看为一条线段，那么也就是有 $n \times 2$ 条线段，其中第 i 与 $i+n$ 只能选择一条，求是否能给出 n 条线段，使得这 n 条线段之间不存在交点。

POJ3683 牧师忙碌日

Solution

将每个时间段看为一条线段，那么也就是有 $n \times 2$ 条线段，其中第 i 与 $i+n$ 只能选择一条，求是否能给出 n 条线段，使得这 n 条线段之间不存在交点。

若 s_i $s_i + d_i$ 和 s_j $s_j + d_j$ 时间有冲突，选 s_i 的话，则必须选 t_j ，即 $s_i \to t_j$ ，同时逆否命题 $\neg t_j \to \neg s_i$ 也成立。

POJ3683 牧师忙碌日

Solution

将每个时间段看为一条线段，那么也就是有 $n \times 2$ 条线段，其中第 i 与 $i+n$ 只能选择一条，求是否能给出 n 条线段，使得这 n 条线段之间不存在交点。

若 s_i s_i+d_i 和 s_j s_j+d_j 时间有冲突，选 s_i 的话，则必须选 t_j ，即 $s_i \rightarrow t_j$ ，同时逆否命题 $\neg t_j \rightarrow \neg s_i$ 也成立。

若 s_i s_i+d_i 和 t_j-d_j t_j 有冲突的话，选 s_i 的话，则必须选 s_j ，即 $s_i \rightarrow s_j$ ，同时逆否命题 $\neg s_j \rightarrow \neg s_i$ 也成立。

POJ3683 牧师忙碌日

Solution

将每个时间段看为一条线段，那么也就是有 $n \times 2$ 条线段，其中第 i 与 $i+n$ 只能选择一条，求是否能给出 n 条线段，使得这 n 条线段之间不存在交点。

若 s_i s_i+d_i 和 s_j s_j+d_j 时间有冲突，选 s_i 的话，则必须选 t_j，即 $s_i \to t_j$，同时逆否命题 $\neg t_j \to \neg s_i$ 也成立。

若 s_i s_i+d_i 和 t_j-d_j t_j 有冲突的话，选 s_i 的话，则必须选 s_j，即 $s_i \to s_j$，同时逆否命题 $\neg s_j \to \neg s_i$ 也成立。

建图后跑 2-SAT 即可。

JSOI2009 游戏

Problem

给定一个 $N \times M$ 迷宫，有些不可走的点。

JSOI2009 游戏

Problem

给定一个 $N \times M$ 迷宫，有些不可走的点。

有一个棋子，玩家 A 可以决定他放哪，之后从玩家 B 开始两玩家轮流移动。

JSOI2009 游戏

Problem

给定一个 $N \times M$ 迷宫，有些不可走的点。

有一个棋子，玩家 A 可以决定他放哪，之后从玩家 B 开始两玩家轮流移动。

在一次游戏中，同一个格子不能进入两次，也不能进入不可走的点。

JSOI2009 游戏

Problem

给定一个 $N \times M$ 迷宫，有些不可走的点。
有一个棋子，玩家 A 可以决定他放哪，之后从玩家 B 开始两玩家轮流移动。
在一次游戏中，同一个格子不能进入两次，也不能进入不可走的点。
问玩家 A 能不能赢，能赢时需要输出全部方案。

JSOI2009 游戏

Solution

将迷宫可走的四相邻点连成一个图。

JSOI2009 游戏

Solution

将迷宫可走的四相邻点连成一个图。

发现这是一个二分图。对此二分图进行判断，是否是一个完美匹配的二分图。

JSOI2009 游戏

Solution

将迷宫可走的四相邻点连成一个图。

发现这是一个二分图。对此二分图进行判断，是否是一个完美匹配的二分图。

如果是则先手必寄，因为后手只需要走先手走的对应匹配边的另一端点即可。

JSOI2009 游戏

Solution

将迷宫可走的四相邻点连成一个图。

发现这是一个二分图。对此二分图进行判断，是否是一个完美匹配的二分图。

如果是则先手必寄，因为后手只需要走先手走的对应匹配边的另一端点即可。

如果不是，则需要找到所有最大匹配都非必须的点。发现找一个非必须点开始跑，若还能跑回这一边，则跑到的那个点也是非必须点。

JSOI2009 游戏

Solution

将迷宫可走的四相邻点连成一个图。

发现这是一个二分图。对此二分图进行判断，是否是一个完美匹配的二分图。

如果是则先手必寄，因为后手只需要走先手走的对应匹配边的另一端点即可。

如果不是，则需要找到所有最大匹配都非必须的点。发现找一个非必须点开始跑，若还能跑回这一边，则跑到的那个点也是非必须点。

故先跑一下最大匹配，标记这些点，未标记点已经是非必须的了。从未标记点跑 DFS，若最后点在同一边，则那个点也不是非必须的点。

JSOI2009 游戏

Solution

将迷宫可走的四相邻点连成一个图。

发现这是一个二分图。对此二分图进行判断，是否是一个完美匹配的二分图。

如果是则先手必寄，因为后手只需要走先手走的对应匹配边的另一端点即可。

如果不是，则需要找到所有最大匹配都非必须的点。发现找一个非必须点开始跑，若还能跑回这一边，则跑到的那个点也是非必须点。

故先跑一下最大匹配，标记这些点，未标记点已经是非必须的了。从未标记点跑 DFS，若最后点在同一边，则那个点也不是非必须的点。

所以如此构造方案即可。

最长不下降子序列问题

Problem

给定正整数序列 $x_1 \dots, x_n$。

最长不下降子序列问题

Problem

给定正整数序列 $x_1 \ldots, x_n$。

- 计算其最长不下降子序列的长度 s。

最长不下降子序列问题

Problem

给定正整数序列 $x_1 \ldots, x_n$。

- 计算其最长不下降子序列的长度 s。
- 如果每个元素只允许使用一次，计算从给定的序列中最多可取出多少个长度为 s 的不下降子序列。

最长不下降子序列问题

Problem

给定正整数序列 $x_1 \ldots, x_n$。

- 计算其最长不下降子序列的长度 s。
- 如果每个元素只允许使用一次，计算从给定的序列中最多可取出多少个长度为 s 的不下降子序列。
- 如果允许在取出的序列中多次使用 x_1 和 x_n（其他元素仍然只允许使用一次），则从给定序列中最多可取出多少个**不同的**长度为 s 的不下降子序列。

最长不下降子序列问题

Problem

给定正整数序列 $x_1 \ldots, x_n$。

- 计算其最长不下降子序列的长度 s。
- 如果每个元素只允许使用一次，计算从给定的序列中最多可取出多少个长度为 s 的不下降子序列。
- 如果允许在取出的序列中多次使用 x_1 和 x_n (其他元素仍然只允许使用一次)，则从给定序列中最多可取出多少个**不同的**长度为 s 的不下降子序列。

$(1 \leq n \leq 500)$。

最长不下降子序列问题

Solution

第一问是 dp 基础题，记录 $dp[i]$ ，将答案记为 len 。

最长不下降子序列问题

Solution

第一问是 dp 基础题，记录 $dp[i]$ ，将答案记为 len 。

考虑后两问，发现问题 3 本质上是问题 2 加 4 条容量为 $\inf$ 的边，$(S,1),(1,1+n),(n,T),(n+n,n)$ 。

最长不下降子序列问题

Solution

第一问是 dp 基础题，记录 $dp[i]$ ，将答案记为 len 。

考虑后两问，发现问题 3 本质上是问题 2 加 4 条容量为 $\inf$ 的边，$(S,1),(1,1+n),(n,T),(n+n,n)$ 。

若 dp 数组中第 i 位值为 1 ，则表示这是一个可能的 LIS 起点，连边 (S,i) ，容量为 1。

最长不下降子序列问题

Solution

第一问是 dp 基础题，记录 $dp[i]$ ，将答案记为 len 。

考虑后两问，发现问题 3 本质上是问题 2 加 4 条容量为 $\inf$ 的边，$(S,1),(1,1+n),(n,T),(n+n,n)$ 。

若 dp 数组中第 i 位值为 1 ，则表示这是一个可能的 LIS 起点，连边 (S,i) ，容量为 1。

若 dp 数组中第 i 位值为 len ，则表示这是一个 LIS 的终点，连边 (i,T) ，容量为 1。

最长不下降子序列问题

Solution

第一问是 dp 基础题，记录 $dp[i]$ ，将答案记为 len 。

考虑后两问，发现问题 3 本质上是问题 2 加 4 条容量为 $\inf$ 的边，$(S,1),(1,1+n),(n,T),(n+n,n)$ 。

若 dp 数组中第 i 位值为 1 ，则表示这是一个可能的 LIS 起点，连边 (S,i) ，容量为 1。

若 dp 数组中第 i 位值为 len ，则表示这是一个 LIS 的终点，连边 (i,T) ，容量为 1。

所有点 i 向 $i+n$ 连一条 1 的边，然后跑网络流即可。

Thanks

Liuhengxi is looking for girlfriend!