提高图图

CSP 图完图 NOIP，图完 NOIP 再图省选，接下来没比赛能图啦。

CaijiMK

2023 年 4 月 24 日

威尼斯第一中学

目录

前言

前言

- 本课件中出现的代码均**未**测试过，**不保证**没有细节问题。欢迎指出。

前言

- 本课件中出现的代码均**未**测试过，**不保证**没有细节问题。欢迎指出。
- 做过的同学请不要大声嘴巴。

前言

- 本课件中出现的代码均**未**测试过，**不保证**没有细节问题。欢迎指出。
- 做过的同学请不要大声嘴巴。
- 本节课内容较多，可能会讲得比较快，请各位同学集中注意力。

前言

- 本课件中出现的代码均**未**测试过，**不保证**没有细节问题。欢迎指出。
- 做过的同学请不要大声嘴巴。
- 本节课内容较多，可能会讲得比较快，请各位同学集中注意力。
- 也可能视情况 skip 一些内容。

前言

- 本课件中出现的代码均**未**测试过，**不保证**没有细节问题。欢迎指出。
- 做过的同学请不要大声嘴巴。
- 本节课内容较多，可能会讲得比较快，请各位同学集中注意力。
- 也可能视情况 skip 一些内容。
- 作业多少沾点个人恩怨，你忍一下。

前言

- 本课件中出现的代码均**未**测试过，**不保证**没有细节问题。欢迎指出。
- 做过的同学请不要大声嘴巴。
- 本节课内容较多，可能会讲得比较快，请各位同学集中注意力。
- 也可能视情况 skip 一些内容。
- 作业多少沾点个人恩怨，你忍一下。
- 讲课人或许缺乏同理心。

前言

- 本课件中出现的代码均**未**测试过，**不保证**没有细节问题。欢迎指出。
- 做过的同学请不要大声嘴巴。
- 本节课内容较多，可能会讲得比较快，请各位同学集中注意力。
- 也可能视情况 skip 一些内容。
- 作业多少沾点个人恩怨，你忍一下。
- 讲课人或许缺乏同理心。
- 验题人也不是什么好东西。

前言

- 本课件中出现的代码均**未**测试过，**不保证**没有细节问题。欢迎指出。
- 做过的同学请不要大声嘴巴。
- 本节课内容较多，可能会讲得比较快，请各位同学集中注意力。
- 也可能视情况 skip 一些内容。
- 作业多少沾点个人恩怨，你忍一下。
- 讲课人或许缺乏同理心。
- 验题人也不是什么好东西。
- 特邀嘉宾：djwj233 <– 什么成分就不用我多说了吧。

一些约定

- 无特殊说明时，V 表示图的点集，E 表示图的边集。

一些约定

- 无特殊说明时，V 表示图的点集，E 表示图的边集。
- 无特殊说明时，G 表示一张图，即 $G=(V,E)$。

一些约定

- 无特殊说明时，V 表示图的点集，E 表示图的边集。
- 无特殊说明时，G 表示一张图，即 $G=(V,E)$。
- 无特殊说明时，图无重边无自环。

图论建模

- 图论题的重要技巧是建模。

图论建模

- 图论题的重要技巧是建模。
- 图论建模即将题目中的问题转化为图论模型，然后用图论算法去解决。

图论建模

- 图论题的重要技巧是建模。
- 图论建模即将题目中的问题转化为图论模型，然后用图论算法去解决。
- 但是图论建模并没有什么特定的套路，所以学图论主要得多做题。

图论建模

- 图论题的重要技巧是建模。
- 图论建模即将题目中的问题转化为图论模型，然后用图论算法去解决。
- 但是图论建模并没有什么特定的套路，所以学图论主要得多做题。
- 下面的例题会有很多建模题。

图论建模

- 图论题的重要技巧是建模。
- 图论建模即将题目中的问题转化为图论模型，然后用图论算法去解决。
- 但是图论建模并没有什么特定的套路，所以学图论主要得多做题。
- 下面的例题会有很多建模题。
- **大胆猜想，小心求证。**

二分图

定义

- **二分图 (Bipartite graph)**，又称二部图，是图论中的一种特殊模型。

定义

- **二分图 (Bipartite graph)**，又称二部图，是图论中的一种特殊模型。
- 设 G 是一个无向图，如果 V 可以分割为两个互不相交的子集 (A, B) 且 $\forall (u, v) \in E$，u, v 分别属于两个不同的点集，则称图 G 为一个二分图。

性质

- 一些前置知识。
  - **黑白染色**：将图中的每一个点染成黑色或白色，使得对于图上的任意一条边，它连接的两个端点的颜色不同。
  - **奇环**：长度为奇数的环。

性质

性质 1

图是二分图 $\Leftrightarrow$ 图可以被黑白染色。

性质

性质 1

图是二分图 $\Leftrightarrow$ 图可以被黑白染色。

证明

构造证明充分性：将二分图的一个子集中的点染成黑色，另一个子集中的点染成白色即可。

构造证明必要性：黑白染色后将黑点放到同一个集合中，将白点放到另一个集合中，满足二分图的定义。

性质

性质 2

图是二分图 $\Leftrightarrow$ 图上无奇环。

性质

性质 2

图是二分图 $\Leftrightarrow$ 图上无奇环。

证明

证明充分性：因为每一条边都是从一个集合走到另一个集合，只有走偶数次才可能回到同一个集合。

证明必要性：考虑若图不是二分图，则无法黑白染色。则出现冲突的地方一定是一个奇环。

洛谷 P1330 封锁阳光大学

- 给你一张无向图，要求选中最少的点使得任意一条边都有且仅有一个端点被选中。
  若无法满足要求则输出 IMPOSSIBLE。
- $1 \leq |V| \leq 10^4, 1 \leq |E| \leq 10^5$

洛谷 P1330 封锁阳光大学 题解

- 一个显然的结论：每个连通块互不影响。

洛谷 P1330 封锁阳光大学 题解

- 一个显然的结论：每个连通块互不影响。
- 于是想到对于每个连通块分别求解。

洛谷 P1330 封锁阳光大学 题解

- 一个显然的结论：每个连通块互不影响。
- 于是想到对于每个连通块分别求解。
- 如果某个点被选中，则所有与它相连的点都不被选中。同理，如果某个点不被选中，则所有与它相连的点都被选中。

洛谷 P1330 封锁阳光大学 题解

- 一个显然的结论：每个连通块互不影响。
- 于是想到对于每个连通块分别求解。
- 如果某个点被选中，则所有与它相连的点都不被选中。同理，如果某个点不被选中，则所有与它相连的点都被选中。
- 将选中看作染黑色，不选中看作染白色，则当图为二分图时，答案为两个点集中较小的那个的大小。
  当图不是二分图时，无法黑白染色，故无解。

洛谷 P1330 封锁阳光大学 题解

- 一个显然的结论：每个连通块互不影响。
- 于是想到对于每个连通块分别求解。
- 如果某个点被选中，则所有与它相连的点都不被选中。同理，如果某个点不被选中，则所有与它相连的点都被选中。
- 将选中看作染黑色，不选中看作染白色，则当图为二分图时，答案为两个点集中较小的那个的大小。
  当图不是二分图时，无法黑白染色，故无解。
- 时间复杂度：$O(|V|+|E|)$。

NOIP2010 提高组 关押罪犯

- 给你一张带边权无向图，你要将它的点集划分成两个子集，使得两个子集的诱导子图的边权最大值最小。
- $|V| \leq 20000, |E| \leq 100000$

NOIP2010 提高组 关押罪犯 题解

- 发现答案具有单调性，于是考虑二分答案。

NOIP2010 提高组 关押罪犯 题解

- 发现答案具有单调性，于是考虑二分答案。
- 假设当前二分到的答案为 mid，则要使所有边权 $> mid$ 的边的两个端点在两个不同的点集中，发现当且仅当这些边组成的图是二分图时有合法的分配方案。

NOIP2010 提高组 关押罪犯 题解

- 发现答案具有单调性，于是考虑二分答案。
- 假设当前二分到的答案为 mid，则要使所有边权 $> mid$ 的边的两个端点在两个不同的点集中，发现当且仅当这些边组成的图是二分图时有合法的分配方案。
- 时间复杂度：$O((|V|+|E|)\log|E|)$。

二分图最大匹配

- **图匹配**：对于一张无向图 G，一组两两不共顶点的边的集合 $M \subseteq E$ 被称为这张图的匹配。

二分图最大匹配

- **图匹配**：对于一张无向图 G，一组两两不共顶点的边的集合 $M \subseteq E$ 被称为这张图的匹配。
- **最大匹配**：$|M|$ 最大的匹配被称为最大匹配。

二分图最大匹配

- **图匹配**：对于一张无向图 G，一组两两不共顶点的边的集合 $M \subseteq E$ 被称为这张图的匹配。
- **最大匹配**：$|M|$ 最大的匹配被称为最大匹配。
- 匹配中的边被称为**匹配边**，反之被称为**未匹配边**。一个点如果是匹配中的某条边的顶点，则被称为**匹配点**，否则被称为**未匹配点**。

二分图最大匹配

- **图匹配**：对于一张无向图 G，一组两两不共顶点的边的集合 $M \subseteq E$ 被称为这张图的匹配。
- **最大匹配**：$|M|$ 最大的匹配被称为最大匹配。
- 匹配中的边被称为**匹配边**，反之被称为**未匹配边**。一个点如果是匹配中的某条边的顶点，则被称为**匹配点**，否则被称为**未匹配点**。
- **完美匹配**：若图中的所有点都是匹配点，则称这个匹配为完美匹配。

二分图最大匹配

- **图匹配**：对于一张无向图 G，一组两两不共顶点的边的集合 $M \subseteq E$ 被称为这张图的匹配。
- **最大匹配**：$|M|$ 最大的匹配被称为最大匹配。
- 匹配中的边被称为**匹配边**，反之被称为**未匹配边**。一个点如果是匹配中的某条边的顶点，则被称为**匹配点**，否则被称为**未匹配点**。
- **完美匹配**：若图中的所有点都是匹配点，则称这个匹配为完美匹配。
- 二分图的最大匹配即为二分图最大匹配。~~(废话)~~

二分图最大匹配

- **图匹配**：对于一张无向图 G，一组两两不共顶点的边的集合 $M \subseteq E$ 被称为这张图的匹配。
- **最大匹配**：$|M|$ 最大的匹配被称为最大匹配。
- 匹配中的边被称为**匹配边**，反之被称为**未匹配边**。一个点如果是匹配中的某条边的顶点，则被称为**匹配点**，否则被称为**未匹配点**。
- **完美匹配**：若图中的所有点都是匹配点，则称这个匹配为完美匹配。
- 二分图的最大匹配即为二分图最大匹配。~~(废话)~~
- 不过值得一提的是，二分图完美匹配的定义为大小较小的那个点集中所有点都是匹配点的匹配。

增广路定理

- 定义

增广路定理

- 定义
  - **交错路（alternating path）**：始于非匹配点且非匹配边和匹配边交替出现的路径。

增广路定理

- 定义
  - **交错路（alternating path）**：始于非匹配点且非匹配边和匹配边交替出现的路径。
  - **增广路（augmenting path）**：是始于非匹配点且终于非匹配点的交错路。

增广路定理

- 定义
  - **交错路（alternating path）**：始于非匹配点且非匹配边和匹配边交替出现的路径。
  - **增广路（augmenting path）**：是始于非匹配点且终于非匹配点的交错路。
- 增广路上非匹配边比匹配边数量多一。如果将增广路上的非匹配边改为匹配边，匹配边改为非匹配边，则匹配大小会增加一，且该路径仍然是交错路。

增广路定理

- 定义
  - **交错路（alternating path）**：始于非匹配点且非匹配边和匹配边交替出现的路径。
  - **增广路（augmenting path）**：是始于非匹配点且终于非匹配点的交错路。
- 增广路上非匹配边比匹配边数量多一。如果将增广路上的非匹配边改为匹配边，匹配边改为非匹配边，则匹配大小会增加一，且该路径仍然是交错路。
- 这种使匹配数增加的过程称为**增广**。

增广路定理

增广路定理（Berge’s lemma）

当图上无增广路时，得到最大匹配。

增广路定理

证明

考虑证明其逆否命题：不是最大匹配时一定有增广路。

假设当前匹配为 M，有另一个比他更大的匹配 $M'(|M| < |M'|)$，令 $N = M \cup M' - M \cap M'$。

显然 N 中每个点的度数 ≤ 2。则 N 的每一个联通块都一定是如下情况之一：

- 一条链，其中的边必定是在 M 中和在 M' 中交错出现，且在 M 中的边数较少。
- 一条链，其中的边必定是在 M 中和在 M' 中交错出现，且在 M 中的边数较大或相同。
- 一个偶环。

显然第一种情况必然会出现，则 M 必然有增广路。

增广路算法

- 根据增广路定理，可以得到增广路算法的核心思想：**找到一个未匹配点，从该点开始寻找增广路进行增广，不断寻找增广路直到找不到为止。**

增广路算法

- 根据增广路定理，可以得到增广路算法的核心思想：**找到一个未匹配点，从该点开始寻找增广路进行增广，不断寻找增广路直到找不到为止。**

引理 1

每个未匹配点只需要枚举一次。

增广路算法

- 根据增广路定理，可以得到增广路算法的核心思想：**找到一个未匹配点，从该点开始寻找增广路进行增广，不断寻找增广路直到找不到为止。**

引理 1

每个未匹配点只需要枚举一次。

证明

考虑反证法，假设某一轮沿着增广路 $a-b$ 增广后新增了以未匹配点 x 为起点的增广路 P_x，则 P_x 与 $a-b$ 必然有公共边。
由于 $a-b$ 是交错路，原先 x 就能走到 $a-b$ 中的某个未匹配点，所以此前就已经存在从 x 出发的增广路。
故已枚举过的点不再可能成为增广路的起点。

增广路算法

- 由此我们可以得到增广路算法的具体实现：

增广路算法

- 由此我们可以得到增广路算法的具体实现：
  1. 枚举每个左部点作为增广路的起点。

增广路算法

- 由此我们可以得到增广路算法的具体实现：
  1. 枚举每个左部点作为增广路的起点。
  2. 寻找以该点为起点的增广路。

增广路算法

- 由此我们可以得到增广路算法的具体实现：
  1. 枚举每个左部点作为增广路的起点。
  2. 寻找以该点为起点的增广路。
  3. 如果找到增广路则进行增广。

增广路算法

- 由此我们可以得到增广路算法的具体实现：
  1. 枚举每个左部点作为增广路的起点。
  2. 寻找以该点为起点的增广路。
  3. 如果找到增广路则进行增广。
  4. 枚举下一个点。

增广路算法

- 增广路算法核心代码如下：

```
int dfs(int now) {
    for (int i = g.hd[now]; i; i = g.nxt[i]) {
        if (vis[g.to[i]]) continue;
        vis[g.to[i]] = true;
        if (!chos[g.to[i]] || dfs(chos[g.to[i]])) {
            chos[g.to[i]] = now;
            return 1;
        }
    }
    return 0;
}
for (int i = 1; i <= n; i++) {
    memset(vis, 0, sizeof(vis));
    ans += dfs(i);
}
```

增广路算法

- 增广路算法核心代码如下：

```
int dfs(int now) {
    for (int i = g.hd[now]; i; i = g.nxt[i]) {
        if (vis[g.to[i]]) continue;
        vis[g.to[i]] = true;
        if (!chos[g.to[i]] || dfs(chos[g.to[i]])) {
            chos[g.to[i]] = now;
            return 1;
        }
    }
    return 0;
}
for (int i = 1; i <= n; i++) {
    memset(vis, 0, sizeof(vis));
    ans += dfs(i);
}
```

- 由于要枚举每个点，每次枚举都要遍历一遍整张图，故时间复杂度为 $O(|V||E|)$。

HNOI2006 超级英雄

- 给你 n 个二元组，你要从每个二元组中取一个数，使得取出的数的集合的 $\operatorname{mex}-1$ 最大，求这个值。
- $1 \leq n \leq 10^3$

HNOI2006 超级英雄 题解

- 考虑将二元组当作右部点，二元组内的数当作左部点。

HNOI2006 超级英雄 题解

- 考虑将二元组当作右部点，二元组内的数当作左部点。
- 题目转化为求一个数 x 满足左部点编号为 $1 \sim x$ 的点都匹配。

HNOI2006 超级英雄 题解

- 考虑将二元组当作右部点，二元组内的数当作左部点。
- 题目转化为求一个数 x 满足左部点编号为 $1 \sim x$ 的点都匹配。
- 发现只要用增广路算法从前往后枚举直到遇到无法匹配的点即可。

HNOI2006 超级英雄 题解

- 考虑将二元组当作右部点，二元组内的数当作左部点。
- 题目转化为求一个数 x 满足左部点编号为 $1 \sim x$ 的点都匹配。
- 发现只要用增广路算法从前往后枚举直到遇到无法匹配的点即可。
- 时间复杂度： $O(n^2)$。

ZJOI2007 矩阵游戏

- T 组询问，对于每组询问：
- 给你一个 $n \times n$ 的 $0/1$ 矩阵，你可以执行以下操作任意次：
  - 交换任意两行。
  - 交换任意两列。
- 问是否存在一种方案使得最终矩阵的主对角线上全是 1。
- $1 \leq T \leq 20, 1 \leq n \leq 200$

ZJOI2007 矩阵游戏 题解

- 将每一行和每一列都抽象成点。

ZJOI2007 矩阵游戏 题解

- 将每一行和每一列都抽象成点。
- 若第 i 行第 j 列的格子为 1，则在第 i 行和第 j 列之间连一条边。

ZJOI2007 矩阵游戏 题解

- 将每一行和每一列都抽象成点。
- 若第 i 行第 j 列的格子为 1，则在第 i 行和第 j 列之间连一条边。
- 对得到的图跑最大匹配。

ZJOI2007 矩阵游戏 题解

- 将每一行和每一列都抽象成点。
- 若第 i 行第 j 列的格子为 1，则在第 i 行和第 j 列之间连一条边。
- 对得到的图跑最大匹配。
- 显然最大匹配上的边不会在同一列或同一行上。

ZJOI2007 矩阵游戏 题解

- 将每一行和每一列都抽象成点。
- 若第 i 行第 j 列的格子为 1，则在第 i 行和第 j 列之间连一条边。
- 对得到的图跑最大匹配。
- 显然最大匹配上的边不会在同一列或同一行上。
- 考虑操作，若交换第 i, j 行，则相当于交换 i 的匹配与 j 的匹配。

ZJOI2007 矩阵游戏 题解

- 将每一行和每一列都抽象成点。
- 若第 i 行第 j 列的格子为 1，则在第 i 行和第 j 列之间连一条边。
- 对得到的图跑最大匹配。
- 显然最大匹配上的边不会在同一列或同一行上。
- 考虑操作，若交换第 i, j 行，则相当于交换 i 的匹配与 j 的匹配。
- 主对角线全是 1 则表示第 1 行与第 1 列匹配，第 2 行与第 2 列匹配，$\ldots$，第 n 行与第 n 列匹配。

ZJOI2007 矩阵游戏 题解

- 将每一行和每一列都抽象成点。
- 若第 i 行第 j 列的格子为 1，则在第 i 行和第 j 列之间连一条边。
- 对得到的图跑最大匹配。
- 显然最大匹配上的边不会在同一列或同一行上。
- 考虑操作，若交换第 i, j 行，则相当于交换 i 的匹配与 j 的匹配。
- 主对角线全是 1 则表示第 1 行与第 1 列匹配，第 2 行与第 2 列匹配，$\ldots$，第 n 行与第 n 列匹配。
- 显然若最大匹配是完美匹配则一定有解，最大匹配不是完美匹配时一定无解。

ZJOI2007 矩阵游戏 题解

- 将每一行和每一列都抽象成点。
- 若第 i 行第 j 列的格子为 1，则在第 i 行和第 j 列之间连一条边。
- 对得到的图跑最大匹配。
- 显然最大匹配上的边不会在同一列或同一行上。
- 考虑操作，若交换第 i, j 行，则相当于交换 i 的匹配与 j 的匹配。
- 主对角线全是 1 则表示第 1 行与第 1 列匹配，第 2 行与第 2 列匹配，$\ldots$，第 n 行与第 n 列匹配。
- 显然若最大匹配是完美匹配则一定有解，最大匹配不是完美匹配时一定无解。
- 时间复杂度：$O(Tn^3)$。

CF901C Bipartite Segments

- 给你一个无向图，保证图上无**偶环**。q 次询问。
- 每次询问一个区间 L, R，求有多少个子区间 $l, r (L \leq l \leq r \leq R)$ 满足编号为 $l, l+1, l+2, \ldots, r$ 的点的诱导子图是二分图。
- $1 \leq |V|, |E|, q \leq 3 \times 10^5$

CF901C Bipartite Segments 题解

- 保证图上无偶环，要求图是二分图，相当于要求图上无环。

CF901C Bipartite Segments 题解

- 保证图上无偶环，要求图是二分图，相当于要求图上无环。
- 图上无偶环 $\Rightarrow$ 图是仙人掌。因为如果要出现环套环则必然会出现至少一个偶环。故可以考虑暴力找环。

CF901C Bipartite Segments 题解

- 保证图上无偶环，要求图是二分图，相当于要求图上无环。
- 图上无偶环 $\Rightarrow$ 图是仙人掌。因为如果要出现环套环则必然会出现至少一个偶环。故可以考虑暴力找环。
- 对于任意一个环，设环上点编号最小值为 mn，最大值为 mx。

CF901C Bipartite Segments 题解

- 保证图上无偶环，要求图是二分图，相当于要求图上无环。
- 图上无偶环 $\Rightarrow$ 图是仙人掌。因为如果要出现环套环则必然会出现至少一个偶环。故可以考虑暴力找环。
- 对于任意一个环，设环上点编号最小值为 mn，最大值为 mx。
- 则对于左端点在 $[1, mn]$ 上的区间，右端点必须 $\leq mx$。

CF901C Bipartite Segments 题解

- 保证图上无偶环，要求图是二分图，相当于要求图上无环。
- 图上无偶环 $\Rightarrow$ 图是仙人掌。因为如果要出现环套环则必然会出现至少一个偶环。故可以考虑暴力找环。
- 对于任意一个环，设环上点编号最小值为 mn，最大值为 mx。
- 则对于左端点在 $[1, mn]$ 上的区间，右端点必须 $\leq mx$。
- 考虑设 $right_i$ 表示以 i 为左端点的区间的右端点的最大值。发现可以 dfs 一遍在 $O(|V| + |E|)$ 时间内求出。

CF901C Bipartite Segments 题解

- 考虑询问，答案即为：

$$\sum_{i=l}^{r} \min(right_i, r) - i + 1$$

CF901C Bipartite Segments 题解

- 考虑询问，答案即为：

$$\sum_{i=l}^{r} \min(right_i, r) - i + 1$$

- 容易发现 $right_i$ 具有单调性，故可以二分出第一个 $\geq r$ 的 $right_i$，之前的点全部可以取到 $right_i$，之后的点只能取到 r。
  前缀和后即可 $O(\log |V|)$ 查询。

CF901C Bipartite Segments 题解

- 考虑询问，答案即为：

$$\sum_{i=l}^{r} \min(right_i, r) - i + 1$$

- 容易发现 $right_i$ 具有单调性，故可以二分出第一个 $\geq r$ 的 $right_i$，之前的点全部可以取到 $right_i$，之后的点只能取到 r。前缀和后即可 $O(\log |V|)$ 查询。
- 时间复杂度：$O(|V| + |E| + q \log |V|)$。

CF19E Fairy

- 给你一张无向图，要求删除一条边使得删边后图是二分图，求方案数。
- $1 \leq |V| \leq 10^4, 0 \leq |E| \leq 10^4$

CF19E Fairy 题解

- 首先如果原图是二分图则删除任意一条边都合法，方案数为 $|E|$。

CF19E Fairy 题解

- 首先如果原图是二分图则删除任意一条边都合法，方案数为 $|E|$。
- 否则删除的边必然在所有奇环的交集中。

CF19E Fairy 题解

- 首先如果原图是二分图则删除任意一条边都合法，方案数为 $|E|$。
- 否则删除的边必然在所有奇环的交集中。
- 显然只需要考虑简单环。

CF19E Fairy 题解

- 首先如果原图是二分图则删除任意一条边都合法，方案数为 $|E|$。
- 否则删除的边必然在所有奇环的交集中。
- 显然只需要考虑简单环。
- 考虑建出一棵生成树，然后枚举非树边。

CF19E Fairy 题解

- 首先如果原图是二分图则删除任意一条边都合法，方案数为 $|E|$。
- 否则删除的边必然在所有奇环的交集中。
- 显然只需要考虑简单环。
- 考虑建出一棵生成树，然后枚举非树边。
- 称一个由一条非树边和若干条树边组成的简单环为本原环。

CF19E Fairy 题解

- 首先如果原图是二分图则删除任意一条边都合法，方案数为 $|E|$。
- 否则删除的边必然在所有奇环的交集中。
- 显然只需要考虑简单环。
- 考虑建出一棵生成树，然后枚举非树边。
- 称一个由一条非树边和若干条树边组成的简单环为本原环。
- 若本原环为奇环，则称本原环上的边为坏边，否则称本原环上的边为好边。

CF19E Fairy 题解

- 首先如果原图是二分图则删除任意一条边都合法，方案数为 $|E|$。
- 否则删除的边必然在所有奇环的交集中。
- 显然只需要考虑简单环。
- 考虑建出一棵生成树，然后枚举非树边。
- 称一个由一条非树边和若干条树边组成的简单环为本原环。
- 若本原环为奇环，则称本原环上的边为坏边，否则称本原环上的边为好边。
- 分别考虑树边和非树边。

CF19E Fairy 题解

- 对于**树边**：

CF19E Fairy 题解

- 对于**树边**：
  - 答案边被所有本原环覆盖。

CF19E Fairy 题解

- 对于**树边**：
  - 答案边被所有本原环覆盖。
  - 答案边不被好边覆盖。
    证明：若一条边既是坏边又是好边，则其一定在一个偶环和一个奇环的交中。则这个奇环和这个偶环的并减去这个奇环和这个偶环的交一定也是个奇环，而这条边不在这个奇环中。

CF19E Fairy 题解

- 对于**树边**：
  - 答案边被所有本原环覆盖。
  - 答案边不被好边覆盖。
    证明：若一条边既是坏边又是好边，则其一定在一个偶环和一个奇环的交中。则这个奇环和这个偶环的并减去这个奇环和这个偶环的交一定也是个奇环，而这条边不在这个奇环中。
  - 满足上面两个条件的边一定是答案边。
    证明：考虑非本原环的奇环，其一定可以表示为若干个本原奇环（至少 1 个）和若干个本原偶环（可以是 0 个）的异或并（即并集减去出现偶数次的边）。显然其所有边要么是好边，要么是坏边，故不会产生贡献。

CF19E Fairy 题解

- 对于**树边**：
  - 答案边被所有本原环覆盖。
  - 答案边不被好边覆盖。
    证明：若一条边既是坏边又是好边，则其一定在一个偶环和一个奇环的交中。则这个奇环和这个偶环的并减去这个奇环和这个偶环的交一定也是个奇环，而这条边不在这个奇环中。
  - 满足上面两个条件的边一定是答案边。
    证明：考虑非本原环的奇环，其一定可以表示为若干个本原奇环（至少 1 个）和若干个本原偶环（可以是 0 个）的异或并（即并集减去出现偶数次的边）。显然其所有边要么是好边，要么是坏边，故不会产生贡献。

- 对于**非树边**：

CF19E Fairy 题解

- 对于**树边**：
  - 答案边被所有本原环覆盖。
  - 答案边不被好边覆盖。
    证明：若一条边既是坏边又是好边，则其一定在一个偶环和一个奇环的交中。则这个奇环和这个偶环的并减去这个奇环和这个偶环的交一定也是个奇环，而这条边不在这个奇环中。
  - 满足上面两个条件的边一定是答案边。
    证明：考虑非本原环的奇环，其一定可以表示为若干个本原奇环（至少 1 个）和若干个本原偶环（可以是 0 个）的异或并（即并集减去出现偶数次的边）。显然其所有边要么是好边，要么是坏边，故不会产生贡献。
- 对于**非树边**：
  - 若只有一个本原奇环，则这个环上的非树边可以作为答案，否则不能作为答案。

CF19E Fairy 题解

- 具体实现只要树上差分，对于奇环给路径上每条树边 $+1$，对于偶环给路径上每条非树边 -1，最后扫一遍统计一下即可。

CF19E Fairy 题解

- 具体实现只要树上差分，对于奇环给路径上每条树边 $+1$，对于偶环给路径上每条非树边 -1，最后扫一遍统计一下即可。
- 另外注意到 dfs 求生成树时每条非树边都是返祖边，故复杂度可以优化至 $O(|V|+|E|)$。

洋算法相关

Tarjan 其人

- Robert E. Tarjan(罗伯特·塔扬，1948～)，生于美国加州波莫纳，计算机科学家。

Tarjan 其人

- Robert E. Tarjan(罗伯特·塔扬，1948～)，生于美国加州波莫纳，计算机科学家。
- Tarjan 发明了很多算法和数据结构。不少他发明的算法都以他的名字命名，以至于有时会让人混淆几种不同的算法。比如求各种连通分量的 Tarjan 算法，求 LCA（Lowest Common Ancestor，最近公共祖先）的 Tarjan 算法。并查集、Splay、Toptree、预流推进也是 Tarjan 发明的。

研究论文 [编辑]

根据谷歌学术，他发表了 500 多篇研究论文，被引用超过 80,000 次。[21]

他的一些顶级论文包括：

- 1972 年：深度优先搜索和线性图算法，R Tarjan，SIAM 计算杂志1 (2)，146-160 [22]
- 1987 年：Fibonacci 堆及其在改进的网络优化算法中的应用，ML Fredman，RE Tarjan，ACM 杂志 (JACM) 34 (3)，596
- 1983 年：数据结构和网络算法，RE Tarjan，工业和应用数学学会[24]
- 1988 年：最大流量问题的新方法，V Goldberg, RE Tarjan, Journal of the ACM (JACM) 35 (4), 921-940 [25]

Tarjan 其人

- Robert E. Tarjan(罗伯特·塔扬，1948～)，生于美国加州波莫纳，计算机科学家。
- Tarjan 发明了很多算法和数据结构。不少他发明的算法都以他的名字命名，以至于有时会让人混淆几种不同的算法。比如求各种连通分量的 Tarjan 算法，求 LCA（Lowest Common Ancestor，最近公共祖先）的 Tarjan 算法。并查集、Splay、Toptree、预流推进也是 Tarjan 发明的。

研究论文 [编辑]

根据谷歌学术，他发表了 500 多篇研究论文，被引用超过 80,000 次。[21]

他的一些顶级论文包括：

- 1972 年：深度优先搜索和线性图算法，R Tarjan，SIAM 计算杂志1 (2)，146-160 [22]
- 1987 年：Fibonacci 堆及其在改进的网络优化算法中的应用，ML Fredman，RE Tarjan，ACM 杂志 (JACM) 34 (3)，596
- 1983 年：数据结构和网络算法，RE Tarjan，工业和应用数学学会[24]
- 1988 年：最大流量问题的新方法，V Goldberg, RE Tarjan, Journal of the ACM (JACM) 35 (4), 921-940 [25]

- 感兴趣的可以戳 Link（需科学上网）去详细了解一下这个~~老不死的~~。

定义

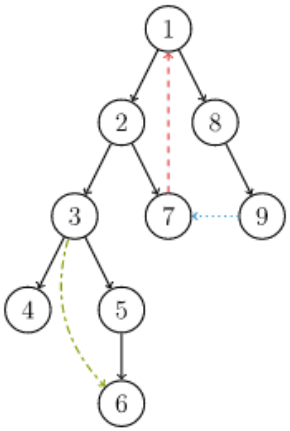
- **强连通**：若有向图中的两个结点 u, v 满足存在一条从 u 到 v 的路径且存在一条从 v 到 u 的路径，则称 u, v 强连通。

定义

- **强连通**：若有向图中的两个结点 u, v 满足存在一条从 u 到 v 的路径且存在一条从 v 到 u 的路径，则称 u, v 强连通。
- **强连通分量（Strongly Connected Components，SCC）**：若有向图中的一个**极大连通子图**满足其中任意两个结点 u, v 都是强连通，则称这个**极大连通子图**为强连通分量。
  显然强连通分量一定是一车环套环。

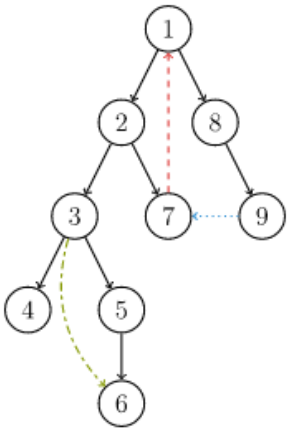
dfs 生成树

- 对于有向图来说，从某一点开始 dfs，每次遇到未访问过的点，就将其和对应的边加入生成树中，得到的外向树就是 dfs 生成树。



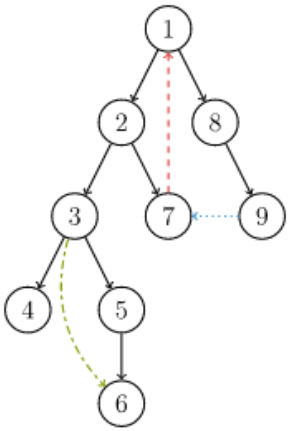
dfs 生成树

- 对于有向图来说，从某一点开始 dfs，每次遇到未访问过的点，就将其和对应的边加入生成树中，得到的外向树就是 dfs 生成树。
- 定义：
-



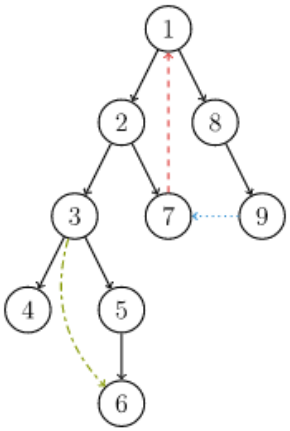
dfs 生成树

- 对于有向图来说，从某一点开始 dfs，每次遇到未访问过的点，就将其和对应的边加入生成树中，得到的外向树就是 dfs 生成树。
- 定义：
  - **树边（tree edge）**：dfs 生成树上的边（图中黑色实线）。
-

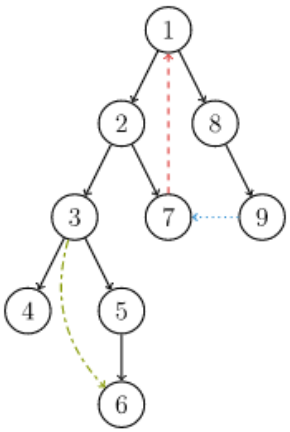


dfs 生成树

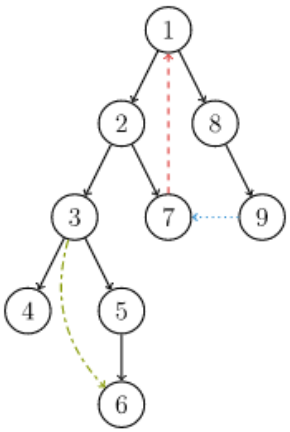
- 对于有向图来说，从某一点开始 dfs，每次遇到未访问过的点，就将其和对应的边加入生成树中，得到的外向树就是 dfs 生成树。
- 定义：
  - **树边（tree edge）**：dfs 生成树上的边（图中黑色实线）。
  - **反祖边（back edge）**：从某点连向它的祖先的边（图中红色虚线）。
-



dfs 生成树

- 对于有向图来说，从某一点开始 dfs，每次遇到未访问过的点，就将其和对应的边加入生成树中，得到的外向树就是 dfs 生成树。
- 定义：
  - **树边（tree edge）**：dfs 生成树上的边（图中黑色实线）。
  - **反祖边（back edge）**：从某点连向它的祖先的边（图中红色虚线）。
  - **横叉边（cross edge）**：从某点连向非祖先非子树的其他点的边（图中蓝色虚线）。
- 

dfs 生成树

- 对于有向图来说，从某一点开始 dfs，每次遇到未访问过的点，就将其和对应的边加入生成树中，得到的外向树就是 dfs 生成树。
- 定义：
  - **树边（tree edge）**：dfs 生成树上的边（图中黑色实线）。
  - **反祖边（back edge）**：从某点连向它的祖先的边（图中红色虚线）。
  - **横叉边（cross edge）**：从某点连向非祖先非子树的其他点的边（图中蓝色虚线）。
  - **前向边（forward edge）**：非树边且从某点连向它的子树中的点的边（图中绿色虚线）。
- 

Tarjan 求强连通分量

- 我们记录两个数组 dfn, low，其中 dfn_i 表示点 i 在 dfs 序中的位置，low_i 表示点 i 经过至多一条反向边能走到的点的 dfn 的最小值。

Tarjan 求强连通分量

- 我们记录两个数组 dfn, low，其中 dfn_i 表示点 i 在 dfs 序中的位置，low_i 表示点 i 经过至多一条反向边能走到的点的 dfn 的最小值。

引理

同一个强连通分量内有且仅有 dfn 最小的点满足 $dfn = low$。

Tarjan 求强连通分量

- 我们记录两个数组 dfn, low，其中 dfn_i 表示点 i 在 dfs 序中的位置，low_i 表示点 i 经过至多一条反向边能走到的点的 dfn 的最小值。

引理

同一个强连通分量内有且仅有 dfn 最小的点满足 $dfn = low$。

证明

首先 dfn 最小的点一定满足 $dfn = low$，因为若 $low < dfn$，则显然从当前点一直往下走，然后再走那条反祖边，再一直往下回到当前点也是一个环，则当前点不是 dfn 最小的点。

接下来考虑其他点，显然不可能无法它们通过反祖边走到上方。如果要通过多于 1 条反祖边走到祖先，则除了最后一条都走到子树内，那么显然可以只保留最后一条。

故结论正确。

Tarjan 求强连通分量

- 我们现在的问题就是要求 low 的值。

Tarjan 求强连通分量

- 我们现在的问题就是要求 low 的值。
- 我们考虑在求 dfs 生成树的过程中，将遇到的点放到一个栈中，并更新 dfn 和 low。

Tarjan 求强连通分量

- 我们现在的问题就是要求 low 的值。
- 我们考虑在求 dfs 生成树的过程中，将遇到的点放到一个栈中，并更新 dfn 和 low。
- 设当前点编号为 u，扫到的点编号为 v。

Tarjan 求强连通分量

- 我们现在的问题就是要求 low 的值。
- 我们考虑在求 dfs 生成树的过程中，将遇到的点放到一个栈中，并更新 dfn 和 low。
- 设当前点编号为 u，扫到的点编号为 v。
  - 若当前边是树边（$dfn_v = 0$），继续扫，并更新 $low_u \leftarrow \min(low_u, low_v)$。

Tarjan 求强连通分量

- 我们现在的问题就是要求 low 的值。
- 我们考虑在求 dfs 生成树的过程中，将遇到的点放到一个栈中，并更新 dfn 和 low。
- 设当前点编号为 u，扫到的点编号为 v。
  - 若当前边是树边（$dfn_v = 0$），继续扫，并更新 $low_u \leftarrow \min(low_u, low_v)$。
  - 若当前边是反祖边（v 在栈中），则 $low_u \leftarrow \min(low_u, dfn_v)$。

Tarjan 求强连通分量

- 我们现在的问题就是要求 low 的值。
- 我们考虑在求 dfs 生成树的过程中，将遇到的点放到一个栈中，并更新 dfn 和 low。
- 设当前点编号为 u，扫到的点编号为 v。
  - 若当前边是树边（$dfn_v = 0$），继续扫，并更新 $low_u \leftarrow \min(low_u, low_v)$。
  - 若当前边是反祖边（v 在栈中），则 $low_u \leftarrow \min(low_u, dfn_v)$。
  - 其他情况：不管。

Tarjan 求强连通分量

- 我们现在的问题就是要求 low 的值。
- 我们考虑在求 dfs 生成树的过程中，将遇到的点放到一个栈中，并更新 dfn 和 low。
- 设当前点编号为 u，扫到的点编号为 v。
  - 若当前边是树边（$dfn_v = 0$），继续扫，并更新 $low_u \leftarrow \min(low_u, low_v)$。
  - 若当前边是反祖边（v 在栈中），则 $low_u \leftarrow \min(low_u, dfn_v)$。
  - 其他情况：不管。
- 最后若 $dfn_u = low_u$，则栈中在点 u 上方的点和 u 在同一个强连通分量中。注意此时要将这些点全都弹出栈。

Tarjan 求强连通分量

- 核心代码如下：

```
1  void Tarjan(int now) {
2      dfn[now] = low[now] = ++Index;
3      s.push(now);
4      for (int i = g.hd[now]; i; i = g.nxt[i])
5          if (!dfn[g.to[i]]) {
6              Tarjan(g.to[i]);
7              low[now] = min(low[now], low[g.to[i]]);
8          } else if (!scc[g.to[i]]) low[now] = min(low[now], dfn[g.to[i]]);
9      if (low[now] == dfn[now]) {
10         scc_cnt++;
11         for (int x = 0; x != now; s.pop()) {
12             x = s.top();
13             scc[x] = scc_cnt;
14         }
15     }
16     return ;
17 }
18 for (int i = 1; i <= n; i++)
19     if (!dfn[i]) Tarjan(i);
```

Tarjan 求强连通分量

- 核心代码如下：

```
1  void Tarjan(int now) {
2      dfn[now] = low[now] = ++Index;
3      s.push(now);
4      for (int i = g.hd[now]; i; i = g.nxt[i])
5          if (!dfn[g.to[i]]) {
6              Tarjan(g.to[i]);
7              low[now] = min(low[now], low[g.to[i]]);
8          } else if (!scc[g.to[i]]) low[now] = min(low[now], dfn[g.to[i]]);
9      if (low[now] == dfn[now]) {
10         scc_cnt++;
11         for (int x = 0; x != now; s.pop()) {
12             x = s.top();
13             scc[x] = scc_cnt;
14         }
15     }
16     return ;
17 }
18 for (int i = 1; i <= n; i++)
19     if (!dfn[i]) Tarjan(i);
```

- 容易发现每条边和每个点都只会被访问 $O(1)$ 次，故时间复杂度 $O(|V|+|E|)$。

Kosaraju 算法

- 东洋算法

Kosaraju 算法

- 东洋算法
- 枚举每个未被访问的点，后序遍历。

Kosaraju 算法

- 东洋算法
- 枚举每个未被访问的点，后序遍历。
- 按后序遍历从大到小枚举，在反图上遍历所有能到达的未被访问的点，这些点构成一个强连通分量中。

Kosaraju 算法

- 感性理解正确性：

Kosaraju 算法

- 感性理解正确性：
- 首先一点重要结论：反图与原图有相同的强连通分量。

Kosaraju 算法

- 感性理解正确性：
- 首先一点重要结论：反图与原图有相同的强连通分量。
- 与 Tarjan 相反，Tarjan 是从叶子往上确定强连通分量，Kosaraju 是从根开始确定强连通分量。

Kosaraju 算法

- 感性理解正确性：
- 首先一点重要结论：反图与原图有相同的强连通分量。
- 与 Tarjan 相反，Tarjan 是从叶子往上确定强连通分量，Kosaraju 是从根开始确定强连通分量。
- 容易发现这个过程中每个点能到达的只可能是自己强连通分量中的点或者已经被确定的强连通分量，故算法正确。

Kosaraju 算法

- 感性理解正确性：
- 首先一点重要结论：反图与原图有相同的强连通分量。
- 与 Tarjan 相反，Tarjan 是从叶子往上确定强连通分量，Kosaraju 是从根开始确定强连通分量。
- 容易发现这个过程中每个点能到达的只可能是自己强连通分量中的点或者已经被确定的强连通分量，故算法正确。
- 时间复杂度：$O(|V|+|E|)$。

Kosaraju 算法

- 感性理解正确性：
- 首先一点重要结论：反图与原图有相同的强连通分量。
- 与 Tarjan 相反，Tarjan 是从叶子往上确定强连通分量，Kosaraju 是从根开始确定强连通分量。
- 容易发现这个过程中每个点能到达的只可能是自己强连通分量中的点或者已经被确定的强连通分量，故算法正确。
- 时间复杂度：$O(|V|+|E|)$。
- 由于 Kosaraju 被 Tarjan 吊打了，所以严谨证明和具体实现这里就不讲了，有兴趣的同学可以自行上网查阅。

缩点

- 将每个强连通分量缩成一个点后得到的新图是一张 DAG。因为如果不是 DAG 则存在至少一个环，这又是一个强连通分量。

USACO 不知道哪一场 Network of Schools

- 给你一张有向图，求：
  - 至少选择几个点，使得这些点能到达所有点。
  - 至少添加几条边，使得任意两点可以互相到达。
- $2 \leq$ 点数 ≤ 100

USACO 不知道哪一场 Network of Schools 题解

- 由于同一个强连通分量内的点可以互相到达，故先将原图缩点。

USACO 不知道哪一场 Network of Schools 题解

- 由于同一个强连通分量内的点可以互相到达，故先将原图缩点。
- 对于第一问，答案为缩点后 DAG 中入度为 0 的点。

USACO 不知道哪一场 Network of Schools 题解

- 由于同一个强连通分量内的点可以互相到达，故先将原图缩点。
- 对于第一问，答案为缩点后 DAG 中入度为 0 的点。
- 对于第二问，相当于要添加边使得原图是一个强连通图。

USACO 不知道哪一场 Network of Schools 题解

- 由于同一个强连通分量内的点可以互相到达，故先将原图缩点。
- 对于第一问，答案为缩点后 DAG 中入度为 0 的点。
- 对于第二问，相当于要添加边使得原图是一个强连通图。
- 答案为入度为 0 的点和出度为 0 的点的个数的较大值。

USACO 不知道哪一场 Network of Schools 题解

- 由于同一个强连通分量内的点可以互相到达，故先将原图缩点。
- 对于第一问，答案为缩点后 DAG 中入度为 0 的点。
- 对于第二问，相当于要添加边使得原图是一个强连通图。
- 答案为入度为 0 的点和出度为 0 的点的个数的较大值。
- 时间复杂度：$O(|V|+|E|)$。

CF1777E Edge Reverse

- 给你一张带边权的有向图，你要翻转一些边，使得至少有一个结点可以到达其他任意结点。
- 翻转的代价为所有翻转的边的边权的最大值。求最小代价。
- $2 \leq |V| \leq 2 \cdot 10^5, 1 \leq |E| \leq 2 \cdot 10^5$

CF1777E Edge Reverse 题解

- 最大值最小，直接二分。

CF1777E Edge Reverse 题解

- 最大值最小，直接二分。
- 对于可以翻转的边，容易证明可以将其视为无向边。

CF1777E Edge Reverse 题解

- 最大值最小，直接二分。
- 对于可以翻转的边，容易证明可以将其视为无向边。
- 然后缩点 check 即可。

CF1777E Edge Reverse 题解

- 最大值最小，直接二分。
- 对于可以翻转的边，容易证明可以将其视为无向边。
- 然后缩点 check 即可。
- 时间复杂度：$O((|V|+|E|)\log$ 值域$)$。

国家集训队 稳定婚姻

- 给你一张二分图，保证两部分都有 n 个点，并且告诉你其中一组完美匹配。
- 对于每一对匹配，你要判断删掉这条边后这张图是否依然存在完美匹配。
- $1 \leq \frac{|V|}{2} \leq 4000, \frac{|V|}{2} \leq |E| <= 20000$

国家集训队 稳定婚姻 解法一

- 匈牙利算法可以很方便地实现撤销匹配操作。只要把这个匹配删掉，然后从当前点开始重新跑增广路即可。

国家集训队 稳定婚姻 解法一

- 匈牙利算法可以很方便地实现撤销匹配操作。只要把这个匹配删掉，然后从当前点开始重新跑增广路即可。
- 时间复杂度：$O(|V||E|)$。

国家集训队 稳定婚姻 解法二

- 对于原本的匹配，从左部点向右部点连边，其他的边从右部点向左部点连边。

国家集训队 稳定婚姻 解法二

- 对于原本的匹配，从左部点向右部点连边，其他的边从右部点向左部点连边。
- 考虑一条合法的从当前点开始的增广路，把这条匹配边加进去后发现恰好成为一个环，并且容易证明这是充要条件。

国家集训队 稳定婚姻 解法二

- 对于原本的匹配，从左部点向右部点连边，其他的边从右部点向左部点连边。
- 考虑一条合法的从当前点开始的增广路，把这条匹配边加进去后发现恰好成为一个环，并且容易证明这是充要条件。
- 上面说过，强连通分量是一车环套环，即两个点在一个环里当且仅当他们在同一个强连通分量里。

国家集训队 稳定婚姻 解法二

- 对于原本的匹配，从左部点向右部点连边，其他的边从右部点向左部点连边。
- 考虑一条合法的从当前点开始的增广路，把这条匹配边加进去后发现恰好成为一个环，并且容易证明这是充要条件。
- 上面说过，强连通分量是一车环套环，即两个点在一个环里当且仅当他们在同一个强连通分量里。
- 于是直接 Tarjan 求强连通分量即可。

国家集训队 稳定婚姻 解法二

- 对于原本的匹配，从左部点向右部点连边，其他的边从右部点向左部点连边。
- 考虑一条合法的从当前点开始的增广路，把这条匹配边加进去后发现恰好成为一个环，并且容易证明这是充要条件。
- 上面说过，强连通分量是一车环套环，即两个点在一个环里当且仅当他们在同一个强连通分量里。
- 于是直接 Tarjan 求强连通分量即可。
- 时间复杂度：$O(|V|+|E|)$。

SAT 问题

- 有 n 个取值仅有 0 或 1 的变量 $a, b, c, \ldots$，每个变量有两种取值。另有若干条限制，每条限制形如 $a \vee b \vee c \vee \ldots = True$。求合法解。

SAT 问题

- 有 n 个取值仅有 0 或 1 的变量 $a, b, c, \ldots$，每个变量有两种取值。另有若干条限制，每条限制形如 $a \vee b \vee c \vee \ldots = True$。求合法解。
- 当每条限制所涉及的变量数均为 k 时，这个问题被称为 k-SAT 问题。

SAT 问题

- 有 n 个取值仅有 0 或 1 的变量 $a, b, c, \ldots$，每个变量有两种取值。另有若干条限制，每条限制形如 $a \vee b \vee c \vee \ldots = True$。求合法解。
- 当每条限制所涉及的变量数均为 k 时，这个问题被称为 k-SAT 问题。
- 已经证明当 $k > 2$ 时为 NPC。故这里仅讨论 2-SAT 问题。

SAT 问题

- 有 n 个取值仅有 0 或 1 的变量 $a, b, c, \ldots$，每个变量有两种取值。另有若干条限制，每条限制形如 $a \vee b \vee c \vee \ldots = True$。求合法解。
- 当每条限制所涉及的变量数均为 k 时，这个问题被称为 k-SAT 问题。
- 已经证明当 $k > 2$ 时为 NPC。故这里仅讨论 2-SAT 问题。
- **以下用 x 表示变量 $x = True$，$\neg x$ 表示 $x = False$。并用有序对 (a, b) 表示一条限制。**

2-SAT 问题的判定

- 使用种类并查集解决。

2-SAT 问题的判定

- 使用种类并查集解决。
- 对于任意限制 (a, b)，连接点 (i, j) 和 $(n + i, n + j)$。

2-SAT 问题的判定

- 使用种类并查集解决。
- 对于任意限制 (a, b)，连接点 (i, j) 和 $(n + i, n + j)$。
- 当且仅当 $\forall i \in [1, n]$，i 与 $n + i$ 不在同一集合时有合法解。

2-SAT 问题的判定

- 使用种类并查集解决。
- 对于任意限制 (a, b)，连接点 (i, j) 和 $(n + i, n + j)$。
- 当且仅当 $\forall i \in [1, n]$，i 与 $n + i$ 不在同一集合时有合法解。
- 正确性的话，相当于要选点，如果选了一个点则这个点所在连通块都要选。

2-SAT 问题求解

- 仍然考虑拆点。

2-SAT 问题求解

- 仍然考虑拆点。
- 对于限制条件 (a, b)，我们在图上连接边 $\neg a \to b$，$\neg b \to a$。

2-SAT 问题求解

- 仍然考虑拆点。
- 对于限制条件 (a, b)，我们在图上连接边 $\neg a \to b$，$\neg b \to a$。
- 对整张图缩点并求拓扑序。若 x 所在强连通分量拓扑序大于 $\neg x$ 所在强连通分量拓扑序，则变量 $x = True$。

2-SAT 问题求解

- 仍然考虑拆点。
- 对于限制条件 (a, b)，我们在图上连接边 $\neg a \to b$，$\neg b \to a$。
- 对整张图缩点并求拓扑序。若 x 所在强连通分量拓扑序大于 $\neg x$ 所在强连通分量拓扑序，则变量 $x = True$。
- 若 x 与 $\neg x$ 在同一强连通分量内则无解。

2-SAT 问题求解

- 感性理解其正确性。

2-SAT 问题求解

- 感性理解其正确性。
- 类似于并查集，显然对于任意限制，若不满足其中一个条件则必须满足另一个条件。

2-SAT 问题求解

- 感性理解其正确性。
- 类似于并查集，显然对于任意限制，若不满足其中一个条件则必须满足另一个条件。
- 发现选择一个点后必须选的点是这个点的传递闭包。故 x 与 $\neg x$ 只能选靠后的，否则也会矛盾。

2-SAT 问题求解

- 感性理解其正确性。
- 类似于并查集，显然对于任意限制，若不满足其中一个条件则必须满足另一个条件。
- 发现选择一个点后必须选的点是这个点的传递闭包。故 x 与 $\neg x$ 只能选靠后的，否则也会矛盾。
- 发现 Tarjan 求强连通分量时强连通分量的编号也满足我们的要求，故不需要拓扑排序。

2-SAT 问题求解

- 感性理解其正确性。
- 类似于并查集，显然对于任意限制，若不满足其中一个条件则必须满足另一个条件。
- 发现选择一个点后必须选的点是这个点的传递闭包。故 x 与 $\neg x$ 只能选靠后的，否则也会矛盾。
- 发现 Tarjan 求强连通分量时强连通分量的编号也满足我们的要求，故不需要拓扑排序。
- 码就不需要了吧。

2-SAT 问题求解

- 感性理解其正确性。
- 类似于并查集，显然对于任意限制，若不满足其中一个条件则必须满足另一个条件。
- 发现选择一个点后必须选的点是这个点的传递闭包。故 x 与 $\neg x$ 只能选靠后的，否则也会矛盾。
- 发现 Tarjan 求强连通分量时强连通分量的编号也满足我们的要求，故不需要拓扑排序。
- 码就不需要了吧。
- 时间复杂度 $O(n+$ 限制数量$)$。

HNOI2010 平面图判定

- 给你一张图，判断他是不是平面图。
- 平面图的定义为：可以将这张图放到一个平面中，使得任意两条边除了顶点外没有交点。
- 但是众所周知这个是 NPC 问题。所以保证图上有一条哈密顿回路，且顺序为 $1, 2, 3, \ldots, n$。
- $3 \leq |V| \leq 10000, |E| \leq 500000$

HNOI2010 平面图判定 题解

- 考虑先把哈密顿回路放好，此时图形成了一个圆。

HNOI2010 平面图判定 题解

- 考虑先把哈密顿回路放好，此时图形成了一个圆。
- 容易发现剩下的边只有两种情况：放圆内和放圆外。

HNOI2010 平面图判定 题解

- 考虑先把哈密顿回路放好，此时图形成了一个圆。
- 容易发现剩下的边只有两种情况：放圆内和放圆外。
- 如果两个区间相交则不能同时在圆内或同时在圆外。

HNOI2010 平面图判定 题解

- 考虑先把哈密顿回路放好，此时图形成了一个圆。
- 容易发现剩下的边只有两种情况：放圆内和放圆外。
- 如果两个区间相交则不能同时在圆内或同时在圆外。
- 这就是 2-SAT 问题，直接并查集判断就好了。

HNOI2010 平面图判定 题解

- 考虑先把哈密顿回路放好，此时图形成了一个圆。
- 容易发现剩下的边只有两种情况：放圆内和放圆外。
- 如果两个区间相交则不能同时在圆内或同时在圆外。
- 这就是 2-SAT 问题，直接并查集判断就好了。
- 可是连边是 $|E|^2$ 的啊？

HNOI2010 平面图判定 题解

- 考虑先把哈密顿回路放好，此时图形成了一个圆。
- 容易发现剩下的边只有两种情况：放圆内和放圆外。
- 如果两个区间相交则不能同时在圆内或同时在圆外。
- 这就是 2-SAT 问题，直接并查集判断就好了。
- 可是连边是 $|E|^2$ 的啊?
- 平面图判定定理：当且仅当 $|E| \leq 3|V| - 6$ 时图才有可能成为平面图。

HNOI2010 平面图判定 题解

- 考虑先把哈密顿回路放好，此时图形成了一个圆。
- 容易发现剩下的边只有两种情况：放圆内和放圆外。
- 如果两个区间相交则不能同时在圆内或同时在圆外。
- 这就是 2-SAT 问题，直接并查集判断就好了。
- 可是连边是 $|E|^2$ 的啊?
- 平面图判定定理：当且仅当 $|E| \leq 3|V| - 6$ 时图才有可能成为平面图。
- 时间复杂度：$O(|E|^2\alpha(|V|))$。

定义

- **边双连通**：对于无向图上两个点 u, v，若无论删去哪条边，u, v 都连通，则称 u 和 v 边双连通。

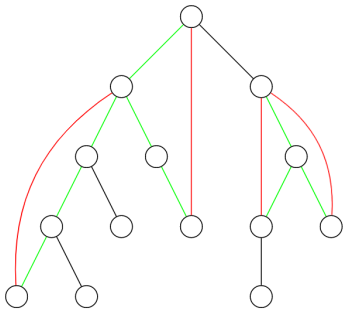
定义

- **边双连通**：对于无向图上两个点 u, v，若无论删去哪条边，u, v 都连通，则称 u 和 v 边双连通。
- **割边**：又称作桥。若删去无向图上一条边会使连通块个数增加，则称这条边为割边。

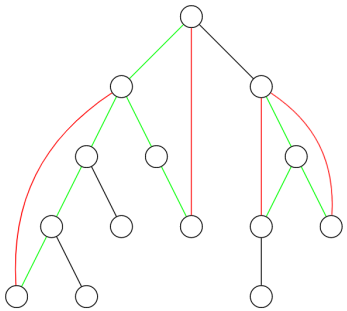
定义

- **边双连通**：对于无向图上两个点 u, v，若无论删去哪条边，u, v 都连通，则称 u 和 v 边双连通。
- **割边**：又称作桥。若删去无向图上一条边会使连通块个数增加，则称这条边为割边。
- **边双连通分量**：满足任意两点都边双连通的极大连通子图。容易发现一个子图任意两点边双连通 $\Leftrightarrow$ 子图中没有割边。

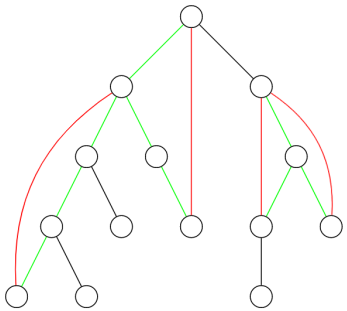
dfs 生成树

- 与有向图的 dfs 生成树类似，从某一点开始 dfs，每次遇到未访问过的点，就将其和对应的边加入生成树中，得到的树就是 dfs 生成树。
- 

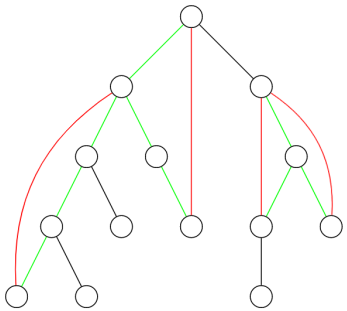
dfs 生成树

- 与有向图的 dfs 生成树类似，从某一点开始 dfs，每次遇到未访问过的点，就将其和对应的边加入生成树中，得到的树就是 dfs 生成树。
- 定义：
- 

dfs 生成树

- 与有向图的 dfs 生成树类似，从某一点开始 dfs，每次遇到未访问过的点，就将其和对应的边加入生成树中，得到的树就是 dfs 生成树。
- 定义：
  - **树边（tree edge）**：dfs 生成树上的边（图中黑色和绿色实线）。
- 

dfs 生成树

- 与有向图的 dfs 生成树类似，从某一点开始 dfs，每次遇到未访问过的点，就将其和对应的边加入生成树中，得到的树就是 dfs 生成树。
- 定义：
  - **树边（tree edge）**：dfs 生成树上的边（图中黑色和绿色实线）。
  - **反祖边（back edge）**：**非树边**且从某点连向它的祖先的边（图中红色实线）。
- 

Tarjan 求割边

- 与求强连通分量的 Tarjan 类似地定义 dfn 和 low。

Tarjan 求割边

- 与求强连通分量的 Tarjan 类似地定义 dfn 和 low。
- 若当前边连接 u, v 且 u 是 v 的祖先且 $low_v > dfn_u$ 则当前边是割边。

Tarjan 求割边

- 与求强连通分量的 Tarjan 类似地定义 dfn 和 low。
- 若当前边连接 u, v 且 u 是 v 的祖先且 $low_v > dfn_u$ 则当前边是割边。
- 非树边必然不是割边。证明略。

Tarjan 求割边

- 与求强连通分量的 Tarjan 类似地定义 dfn 和 low。
- 若当前边连接 u, v 且 u 是 v 的祖先且 $low_v > dfn_u$ 则当前边是割边。
- 非树边必然不是割边。证明略。
- 该算法正确当且仅当 $\forall (u, v) \in E, (u, v)$ 是割边 $\Leftrightarrow dfn_u < low_v$，其中 $dfn_u < dfn_v$。

Tarjan 求割边

- 与求强连通分量的 Tarjan 类似地定义 dfn 和 low。
- 若当前边连接 u, v 且 u 是 v 的祖先且 $low_v > dfn_u$ 则当前边是割边。
- 非树边必然不是割边。证明略。
- 该算法正确当且仅当 $\forall (u, v) \in E, (u, v)$ 是割边 $\Leftrightarrow dfn_u < low_v$，其中 $dfn_u < dfn_v$。
- **习题：证明上面这句话的正确性。**

Tarjan 求割边

习题答案

若 (u, v) 是割边，则删去 (u, v) 后 u, v 不连通，即 u, v 之间不存在一条不经过 (u, v) 的路径。反之则存在一条不经过 (u, v) 的路径。

只需要考虑 (u, v) 是树边的情况，故 u 是 v 的父亲。故存在一条不经过 (u, v) 的路径 $\Leftrightarrow$ 存在一条只经过至多一条反祖边的路径。证明略。

故存在一条不经过 (u, v) 的路径 $\Leftrightarrow dfn_u < low_v$。

Tarjan 求割边

- 核心代码如下：

```
1  void Tarjan(int now, int fa) {
2      dfn[now] = low[now] = ++Index;
3      for (int i = g.hd[now]; i; i = g.nxt[i])
4          if (!dfn[g.to[i]]) {
5              Tarjan(g.to[i], now);
6              low[now] = min(low[now], low[g.to[i]]);
7              if (low[g.to[i]] > dfn[now])
8                  printf("A Bridge of the Input Garph is (%d, %d)\n", now, g.to[i]);
9          } else if (g.to[i] != fa) low[now] = min(low[now], dfn[g.to[i]]);
10     return ;
11 }
12 for (int i = 1; i <= n; i++)
13     if (!dfn[i]) Tarjan(i, 0);
```

Tarjan 求割边

- 核心代码如下：

```
1  void Tarjan(int now, int fa) {
2      dfn[now] = low[now] = ++Index;
3      for (int i = g.hd[now]; i; i = g.nxt[i])
4          if (!dfn[g.to[i]]) {
5              Tarjan(g.to[i], now);
6              low[now] = min(low[now], low[g.to[i]]);
7              if (low[g.to[i]] > dfn[now])
8                  printf("A Bridge of the Input Garph is (%d, %d)\n", now, g.to[i]);
9          } else if (g.to[i] != fa) low[now] = min(low[now], dfn[g.to[i]]);
10     return ;
11 }
12 for (int i = 1; i <= n; i++)
13     if (!dfn[i]) Tarjan(i, 0);
```

- 时间复杂度 $O(|V| + |E|)$。

Tarjan 求边双连通分量

- 在求割边的基础上维护一个栈。

Tarjan 求边双连通分量

- 在求割边的基础上维护一个栈。
- 扫完一个点后，若 $dfn_u = low_u$，则栈中这个点及以上的点形成连通分量中没有割边。则这些点是一个边双连通分量。

Tarjan 求边双连通分量

- 在求割边的基础上维护一个栈。
- 扫完一个点后，若 $dfn_u = low_u$，则栈中这个点及以上的点形成连通分量中没有割边。则这些点是一个边双连通分量。
- 最后记得把这些点从栈中弹出。

Tarjan 求边双连通分量

- 核心代码如下：

```
1  void Tarjan(int now, int fa) {
2      dfn[now] = low[now] = ++Index;
3      s.push(now);
4      for (int i = g.hd[now]; i; i = g.nxt[i])
5          if (!dfn[g.to[i]]) {
6              Tarjan(g.to[i], now);
7              low[now] = min(low[now], low[g.to[i]]);
8          } else if (g.to[i] != fa) low[now] = min(low[now], dfn[g.to[i]]);
9      if (low[now] == dfn[now]) {
10         bcc_cnt++;
11         for (int x = 0; x != now; s.pop()) {
12             x = s.top();
13             bcc[x] = bcc_cnt;
14         }
15     }
16     return ;
17 }
18 for (int i = 1; i <= n; i++)
19     if (!dfn[i]) Tarjan(i, 0);
```

Tarjan 求边双连通分量

- 核心代码如下：

```
1  void Tarjan(int now, int fa) {
2      dfn[now] = low[now] = ++Index;
3      s.push(now);
4      for (int i = g.hd[now]; i; i = g.nxt[i])
5          if (!dfn[g.to[i]]) {
6              Tarjan(g.to[i], now);
7              low[now] = min(low[now], low[g.to[i]]);
8          } else if (g.to[i] != fa) low[now] = min(low[now], dfn[g.to[i]]);
9      if (low[now] == dfn[now]) {
10         bcc_cnt++;
11         for (int x = 0; x != now; s.pop()) {
12             x = s.top();
13             bcc[x] = bcc_cnt;
14         }
15     }
16     return ;
17 }
18 for (int i = 1; i <= n; i++)
19     if (!dfn[i]) Tarjan(i, 0);
```

- 时间复杂度 $O(|V| + |E|)$。

边双连通分量缩点

- 将无向图的边双连通分量缩点后所得的新图是一棵树。

NOIP2022 建造军营

- 给你一张无向连通图，你要标记至少一个点和若干条边，使得删掉任意一条未被标记的点之后标记的点仍然连通。
- 求方案数，对 10^9+7 取模。
- $1 \le |V| \le 5 \times 10^5, |V|-1 \le |E| \le 10^6$

NOIP2022 建造军营 题解

- 显然非割边标记与否不重要，所以先缩点。

NOIP2022 建造军营 题解

- 显然非割边标记与否不重要，所以先缩点。
- 剩下的某个雀批已经讲过了。

NOIP2022 建造军营 题解

- 显然非割边标记与否不重要，所以先缩点。
- 剩下的某个雀批已经讲过了。
- 什么，你没听懂？Click it.

定义

- **点双连通**：对于无向图上两个点 u, v，若无论删去哪个**除** u, v **外**的点，u, v 都连通，则称 u 和 v 点双连通。

定义

- **点双连通**：对于无向图上两个点 u, v，若无论删去哪个**除** u, v **外**的点，u, v 都连通，则称 u 和 v 点双连通。
- **割点**：若删去无向图上一个点会使连通块个数增加，则称这个点为割点。

定义

- **点双连通**：对于无向图上两个点 u, v，若无论删去哪个**除** u, v **外**的点，u, v 都连通，则称 u 和 v 点双连通。
- **割点**：若删去无向图上一个点会使连通块个数增加，则称这个点为割点。
- **点双连通分量**：满足任意两点都点双连通的极大连通子图。

Tarjan 求割点

- 对于某一点 u，若存在一个和他连接的点 v 满足 $dfn_u = low_v$，则 u 是割点。

Tarjan 求割点

- 对于某一点 u，若存在一个和他连接的点 v 满足 $dfn_u = low_v$，则 u 是割点。
- 然而这个做法是错误的，原因是对根是否是割点会判断错误，故需要特判。当根有多于一个子树时根是割点，否则不是。

Tarjan 求割点

- 核心代码如下：

```
1  void Tarjan(int now, int root) {
2      dfn[now] = low[now] = ++Index;
3      int sons=0, flag=0;
4      for (int i=g.hd[now]; i; i = g.nxt[i], sons++)
5          if (!dfn[g.to[i]]) {
6              Tarjan(g.to[i], now);
7              low[now] = min(low[now], low[g.to[i]]);
8              if (now!=root && low[g.to[i]] == dfn[now] && !flag)
9                  printf("A Cut Vertex of the Input Graph is %d.", now), flag=1;
10         } else low[now] = min(low[now], dfn[g.to[i]]);
11     if (now == root && sons >= 2)
12         printf("A Cut Vertex of the Input Graph is %d.", now);
13     return ;
14 }
15 for (int i = 1; i <= n; i++)
16     if (!dfn[i]) Tarjan(i, i);
```

Tarjan 求割点

- 核心代码如下：

```
1  void Tarjan(int now, int root) {
2      dfn[now] = low[now] = ++Index;
3      int sons=0, flag=0;
4      for (int i=g.hd[now]; i; i = g.nxt[i], sons++)
5          if (!dfn[g.to[i]]) {
6              Tarjan(g.to[i], now);
7              low[now] = min(low[now], low[g.to[i]]);
8              if (now!=root && low[g.to[i]] == dfn[now] && !flag)
9                  printf("A Cut Vertex of the Input Graph is %d.", now), flag=1;
10         } else low[now] = min(low[now], dfn[g.to[i]]);
11     if (now == root && sons >= 2)
12         printf("A Cut Vertex of the Input Graph is %d.", now);
13     return ;
14 }
15 for (int i = 1; i <= n; i++)
16     if (!dfn[i]) Tarjan(i, i);
```

- 时间复杂度：$O(|V|+|E|)$。

Tarjan 求点双连通分量

- 当出现 $dfn_u = low_v$ 时，栈中 v 及以上的点以及点 u 是一个点双连通分量。然后将 v 及以上的点从栈中弹出。

Tarjan 求点双连通分量

- 当出现 $dfn_u = low_v$ 时，栈中 v 及以上的点以及点 u 是一个点双连通分量。然后将 v 及以上的点从栈中弹出。
- 此时不需要特判根结点。(想一想，为什么?)

Tarjan 求点双连通分量

- 当出现 $dfn_u = low_v$ 时，栈中 v 及以上的点以及点 u 是一个点双连通分量。然后将 v 及以上的点从栈中弹出。
- 此时不需要特判根结点。(想一想，为什么?)
- 上述算法本质是先求出一个不包含割点的极大连通子图，并添加与之相连的割点。

Tarjan 求点双连通分量

- 当出现 $dfn_u = low_v$ 时，栈中 v 及以上的点以及点 u 是一个点双连通分量。然后将 v 及以上的点从栈中弹出。
- 此时不需要特判根结点。(想一想，为什么?)
- 上述算法本质是先求出一个不包含割点的极大连通子图，并添加与之相连的割点。
- **习题：简要证明求出的子图是点双连通分量。(Tips：归纳法)**

Tarjan 求点双连通分量

习题答案

对于一个不包含割点的极大连通子图，其中任意两点必然满足点双连通。考虑在满足上述条件的极大连通子图的基础上添加一个与之连通的割点，该割点与图中其他点也满足点双连通。对于一个任意两点点双连通的子图，添加一个与其中非割点的点连通的割点。

- 该点与子图中非割点的点点双连通。
- 该点与子图中割点点双连通。
  证明：若不满足点双连通，则这两点之间必然存在第三个割点和一条这三个点不相邻的路径，则路径中在第三个割点前后的任意两个非割点的点不点双连通，与条件不符。

重复上述操作直到无法添加割点，容易发现此时也无法添加非割点的点，故此时的子图为点双连通分量。

Tarjan 求点双连通分量

- 核心代码如下：

```
1  void Tarjan(int now) {
2      dfn[now] = low[now] = ++Index;
3      s.push(now);
4      for (int i = g.hd[now]; i; i = g.nxt[i], sons++)
5          if (!dfn[g.to[i]]) {
6              Tarjan(g.to[i]);
7              low[now] = min(low[now], low[g.to[i]]);
8              if (low[g.to[i]] == dfn[now]) {
9                  printf("BCC #%d:\n", ++bcc_cnt);
10                 for (int x = 0; x != g.to[i]; s.pop())
11                     printf("%d ", x = s.top());
12                 printf("%d\n", now);
13             }
14         } else low[now] = min(low[now], dfn[g.to[i]]);
15     return ;
16 }
17 for (int i = 1; i <= n; i++)
18     if (!dfn[i]) Tarjan(i, i);
```

Tarjan 求点双连通分量

- 核心代码如下：

```
1  void Tarjan(int now) {
2      dfn[now] = low[now] = ++Index;
3      s.push(now);
4      for (int i = g.hd[now]; i; i = g.nxt[i], sons++)
5          if (!dfn[g.to[i]]) {
6              Tarjan(g.to[i]);
7              low[now] = min(low[now], low[g.to[i]]);
8              if (low[g.to[i]] == dfn[now]) {
9                  printf("BCC #%d:\n", ++bcc_cnt);
10                 for (int x = 0; x != g.to[i]; s.pop())
11                     printf("%d ", x = s.top());
12                 printf("%d\n", now);
13             }
14         } else low[now] = min(low[now], dfn[g.to[i]]);
15     return ;
16 }
17 for (int i = 1; i <= n; i++)
18     if (!dfn[i]) Tarjan(i, i);
```

- 时间复杂度：$O(|V| + |E|)$。

圆方树

- 原无向图中每个点被称为『圆点』。

圆方树

- 原无向图中每个点被称为『圆点』。
- 对于无向图的每个点双连通分量，建立一个『方点』。

圆方树

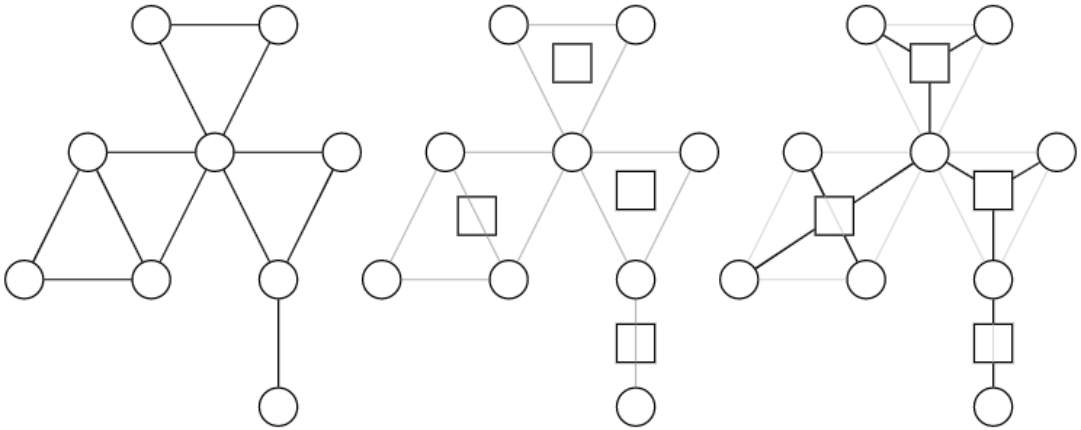
- 原无向图中每个点被称为『圆点』。
- 对于无向图的每个点双连通分量，建立一个『方点』。
- 删掉原图所有边。

圆方树

- 原无向图中每个点被称为『圆点』。
- 对于无向图的每个点双连通分量，建立一个『方点』。
- 删掉原图所有边。
- 每个『方点』向其对应的点双连通分量的每个『圆点』连边，得到的新图即为圆方树。

圆方树

- 原无向图中每个点被称为『圆点』。
- 对于无向图的每个点双连通分量，建立一个『方点』。
- 删掉原图所有边。
- 每个『方点』向其对应的点双连通分量的每个『圆点』连边，得到的新图即为圆方树。
-



圆方树

- 具体实现只要在点双的基础上添加连边操作即可。

圆方树

- 具体实现只要在点双的基础上添加连边操作即可。
- 每个图的点双连通分量个数至多为 n 个，故新图空间需开两倍。

圆方树

- 圆方树是干什么用的?

圆方树

- 圆方树是干什么用的?
- 看例题就知道了。

HNOI2012 矿场搭建

- 给你一张无向图，求至少标记几个点，使得删除任意点后其余点都能与至少一个被标记点连通。
- 输出最少的标记点数量和标记点数量最少的方案数。
- $1 \leq |V| \leq 1000, 1 \leq |E| \leq 500$

HNOI2012 矿场搭建 题解

- 对于每个点双分开考虑。

HNOI2012 矿场搭建 题解

- 对于每个点双分开考虑。
- 若该点双中没有割点，则至少要有 2 个被标记点，可以是任意点。

HNOI2012 矿场搭建 题解

- 对于每个点双分开考虑。
- 若该点双中没有割点，则至少要有 2 个被标记点，可以是任意点。
- 若该点双中有 1 个割点，则至少要有 1 个被标记点，可以是任意非割点的点。

HNOI2012 矿场搭建 题解

- 对于每个点双分开考虑。
- 若该点双中没有割点，则至少要有 2 个被标记点，可以是任意点。
- 若该点双中有 1 个割点，则至少要有 1 个被标记点，可以是任意非割点的点。
- 若该点双中有 ≥ 2 个割点，则无论删除哪个割点，其余点都能与至少一个只有一个割点的点双连通，故不需要被标记点。

HNOI2012 矿场搭建 题解

- 对于每个点双分开考虑。
- 若该点双中没有割点，则至少要有 2 个被标记点，可以是任意点。
- 若该点双中有 1 个割点，则至少要有 1 个被标记点，可以是任意非割点的点。
- 若该点双中有 ≥ 2 个割点，则无论删除哪个割点，其余点都能与至少一个只有一个割点的点双连通，故不需要被标记点。
- 时间复杂度：$O(|V|+|E|)$。

APIO2018 铁人两项

- 给你一张无向图，求满足条件的**有序**三元组 (u, v, w) 的数量。
- 条件为：$u \in V, v \in V, w \in V$，且存在一条以 u 为起点，以 w 为终点，经过点 v 的简单路径。
- $1 \leq |V| \leq 100000, 1 \leq |E| \leq 200000$

APIO2018 铁人两项 题解

- 首先建出圆方树。

APIO2018 铁人两项 题解

- 首先建出圆方树。
- 对于一条路径 (u, w)，发现 v 的取值为路径中所有方点对应的点双的点集的并。

APIO2018 铁人两项 题解

- 首先建出圆方树。
- 对于一条路径 (u, w)，发现 v 的取值为路径中所有方点对应的点双的点集的并。
- 于是考虑枚举 v 所在的方点，可以简单计算出 (u, w) 的数量，然后乘上该点双的大小即可。

APIO2018 铁人两项 题解

- 首先建出圆方树。
- 对于一条路径 (u, w)，发现 v 的取值为路径中所有方点对应的点双的点集的并。
- 于是考虑枚举 v 所在的方点，可以简单计算出 (u, w) 的数量，然后乘上该点双的大小即可。
- 但是这样会数重和数到不合法的方案，于是考虑去重。

APIO2018 铁人两项 题解

- 首先建出圆方树。
- 对于一条路径 (u, w)，发现 v 的取值为路径中所有方点对应的点双的点集的并。
- 于是考虑枚举 v 所在的方点，可以简单计算出 (u, w) 的数量，然后乘上该点双的大小即可。
- 但是这样会数重和数到不合法的方案，于是考虑去重。
- 对于数重的情况，每个方案的 v 会在他左右两个方点上被统计到，所以减去每个 v 的 (u, w) 的数量即可。

APIO2018 铁人两项 题解

- 首先建出圆方树。
- 对于一条路径 (u, w)，发现 v 的取值为路径中所有方点对应的点双的点集的并。
- 于是考虑枚举 v 所在的方点，可以简单计算出 (u, w) 的数量，然后乘上该点双的大小即可。
- 但是这样会数重和数到不合法的方案，于是考虑去重。
- 对于数重的情况，每个方案的 v 会在他左右两个方点上被统计到，所以减去每个 v 的 (u, w) 的数量即可。
- 对于不合法方案，对于每个 v 减去能到达的点的个数即可。

APIO2018 铁人两项 题解

- 首先建出圆方树。
- 对于一条路径 (u, w)，发现 v 的取值为路径中所有方点对应的点双的点集的并。
- 于是考虑枚举 v 所在的方点，可以简单计算出 (u, w) 的数量，然后乘上该点双的大小即可。
- 但是这样会数重和数到不合法的方案，于是考虑去重。
- 对于数重的情况，每个方案的 v 会在他左右两个方点上被统计到，所以减去每个 v 的 (u, w) 的数量即可。
- 对于不合法方案，对于每个 v 减去能到达的点的个数即可。
- 时间复杂度：$O(|V| + |E|)$。

省选联考 2023 城市建造

- 给你一张无向连通图，你要选出一个导出子图，使得：
- 在原图中删去该子图中的边后的连通块数等于该导出子图的点集大小。
- 任意两个连通块的点集大小之差不超过 k。
- 要求该子图大小至少为 2。
- 求方案数。
- $3 \leq |V| \leq 10^5, |V| - 1 \leq |E| \leq 2 \times 10^5, 0 \leq k \leq 1$

省选联考 2023 城市建造 题解

- 我不会。

省选联考 2023 城市建造 题解

- 我不会。
- 这个时候就需要特邀嘉宾了。

徐睿

嘉潘镂玉成皇家中学教育集团后置嘴巴大学附属阿鲁巴职业技术学院东威斯第一浦东校区

NOI2023 浙江省队队爷

感谢 **徐睿** 先生

对本次讲课作出的贡献。

差分约束系统

定义

- 像下面这种形式的不等式组被称为差分约束系统。

$$\begin{cases} x_{a_1} - x_{b_1} \leq c_1 \\ x_{a_2} - x_{b_2} \leq c_2 \\ x_{a_3} - x_{b_3} \leq c_3 \\ \vdots \\ x_{a_m} - x_{b_m} \leq c_m \end{cases}$$

求解

- 考虑将每个不等式转化成 $x_i - c_k \leq x_j$ 的形式。

求解

- 考虑将每个不等式转化成 $x_i - c_k \leq x_j$ 的形式。
- 对于每条限制从 x_i 到 x_j 连一条边权为 $-c_k$ 的边。

求解

- 考虑将每个不等式转化成 $x_i - c_k \leq x_j$ 的形式。
- 对于每条限制从 x_i 到 x_j 连一条边权为 $-c_k$ 的边。
- 对原图跑最长路，即可得到一组合法解。

求解

- 考虑将每个不等式转化成 $x_i - c_k \leq x_j$ 的形式。
- 对于每条限制从 x_i 到 x_j 连一条边权为 $-c_k$ 的边。
- 对原图跑最长路，即可得到一组合法解。
- 正确性显然：对于一条边 (u, v, w)，有 $dis_u + w \leq dis_v$。

求解

- 考虑将每个不等式转化成 $x_i - c_k \leq x_j$ 的形式。
- 对于每条限制从 x_i 到 x_j 连一条边权为 $-c_k$ 的边。
- 对原图跑最长路，即可得到一组合法解。
- 正确性显然：对于一条边 (u, v, w)，有 $dis_u + w \leq dis_v$。
- 把每条边取相反数，然后跑最短路即可。由于有负权边，所以需要用 SPFA。

求解

- 考虑将每个不等式转化成 $x_i - c_k \leq x_j$ 的形式。
- 对于每条限制从 x_i 到 x_j 连一条边权为 $-c_k$ 的边。
- 对原图跑最长路，即可得到一组合法解。
- 正确性显然：对于一条边 (u, v, w)，有 $dis_u + w \leq dis_v$。
- 把每条边取相反数，然后跑最短路即可。由于有负权边，所以需要用 SPFA。
- 若图中存在负环则无解。

求解

- 考虑将每个不等式转化成 $x_i - c_k \leq x_j$ 的形式。
- 对于每条限制从 x_i 到 x_j 连一条边权为 $-c_k$ 的边。
- 对原图跑最长路，即可得到一组合法解。
- 正确性显然：对于一条边 (u, v, w)，有 $dis_u + w \leq dis_v$。
- 把每条边取相反数，然后跑最短路即可。由于有负权边，所以需要用 SPFA。
- 若图中存在负环则无解。
- 注意判负环不要老老实实到松弛 n 遍才判，而是选择一个不会 TLE 的尽量大的上界。

洛谷 P4926 [1007] 倍杀测量者

- 给出一系列不等式，每条形如 $x_{a_i} \geq (k_i - t) \cdot x_{b_i}$ 或 $(k_i + t) \cdot x_{a_i} > x_{b_i}$。
- 已知若干个未知数的值，求最大的 t 使得不等式组无解。
- 设 n 为变量个数，m 为不等式个数，则 $1 \leq n, m \leq 1000$

洛谷 P4926 [1007] 倍杀测量者 题解

- 首先二分 t。

洛谷 P4926 [1007] 倍杀测量者 题解

- 首先二分 t。
- 对不等式两边求 $\log$，然后就变成了一般的差分约束形式。

洛谷 P4926 [1007] 倍杀测量者 题解

- 首先二分 t。
- 对不等式两边求 $\log$，然后就变成了一般的差分约束形式。
- 直接来即可。

欧拉路

定义

- **欧拉回路**：通过图中每条边恰好一次的回路。

定义

- **欧拉回路**：通过图中每条边恰好一次的回路。
- **欧拉通路**：通过图中每条边恰好一次的通路。

定义

- **欧拉回路**：通过图中每条边恰好一次的回路。
- **欧拉通路**：通过图中每条边恰好一次的通路。
- **欧拉图**：具有欧拉回路的图。

定义

- **欧拉回路**：通过图中每条边恰好一次的回路。
- **欧拉通路**：通过图中每条边恰好一次的通路。
- **欧拉图**：具有欧拉回路的图。
- **半欧拉图**：具有欧拉通路但不具有欧拉回路的图。

性质

- 无向图是欧拉图当且仅当非零度点连通且每个点度数均为偶数。

性质

- 无向图是欧拉图当且仅当非零度点连通且每个点度数均为偶数。
- 无向图是半欧拉图当且仅当非零度点连通且有且仅有 2 个点度数为奇数。

性质

- 无向图是欧拉图当且仅当非零度点连通且每个点度数均为偶数。
- 无向图是半欧拉图当且仅当非零度点连通且有且仅有 2 个点度数为奇数。
- 有向图是欧拉图当且仅当非零度点强连通且每个点入度等于出度。

性质

- 无向图是欧拉图当且仅当非零度点连通且每个点度数均为偶数。
- 无向图是半欧拉图当且仅当非零度点连通且有且仅有 2 个点度数为奇数。
- 有向图是欧拉图当且仅当非零度点强连通且每个点入度等于出度。
- 有向图是半欧拉图当且仅当非零度点弱连通且恰好一个点出度比入度多 1 且恰好一个点入度比出度多 1 且其余点入度等于出度。

性质

- 无向图是欧拉图当且仅当非零度点连通且每个点度数均为偶数。
- 无向图是半欧拉图当且仅当非零度点连通且有且仅有 2 个点度数为奇数。
- 有向图是欧拉图当且仅当非零度点强连通且每个点入度等于出度。
- 有向图是半欧拉图当且仅当非零度点弱连通且恰好一个点出度比入度多 1 且恰好一个点入度比出度多 1 且其余点入度等于出度。
- G 为欧拉图 $\Leftrightarrow$ G 可以被拆成若干个简单环。

Hierholzer 算法求欧拉路

- 首先找到任意一条路径。

Hierholzer 算法求欧拉路

- 首先找到任意一条路径。
- 每次将任意一个路径中的点替换成一个包含他的环。

Hierholzer 算法求欧拉路

- 首先找到任意一条路径。
- 每次将任意一个路径中的点替换成一个包含他的环。
- 但是如果直接写又难写复杂度又大，所以这里讲一种 dfs 的实现方法。

Hierholzer 算法求欧拉路

- 首先找到任意一条路径。
- 每次将任意一个路径中的点替换成一个包含他的环。
- 但是如果直接写又难写复杂度又大，所以这里讲一种 dfs 的实现方法。
- 我们直接 dfs 遍历整张图，然后在结束时将当前点加入路径中，将最终结果 reverse 后即可得到一条欧拉路。

Hierholzer 算法求欧拉路

- 核心代码如下：

```
1  void Hierholzer(int now) {
2      while (cur[now]) {
3          int to = g.to[cur[now]];
4          cur[now] = g.nxt[cur[now]];
5          Hierholzer(to);
6      }
7      ans.push_back(now);
8      return ;
9  }
10 for (int i = 1; i <= n; i++) cur[i] = g.hd[i];
11 Hierholzer(s);
12 reverse(ans.begin(), ans.end());
```

Hierholzer 算法求欧拉路

- 核心代码如下：

```
1  void Hierholzer(int now) {
2      while (cur[now]) {
3          int to = g.to[cur[now]];
4          cur[now] = g.nxt[cur[now]];
5          Hierholzer(to);
6      }
7      ans.push_back(now);
8      return ;
9  }
10 for (int i = 1; i <= n; i++) cur[i] = g.hd[i];
11 Hierholzer(s);
12 reverse(ans.begin(), ans.end());
```

- 时间复杂度 $O(|V| + |E|)$。

POI2011 SMI-Garbage

- 给你一张图，每条边有权值 $0/1$，并给出图的目标状态。
- 询问是否存在一种简单环覆盖的方法，使得对于目标状态和原状态权值相同的边被偶数个环覆盖，其他边被奇数个环覆盖。
- 输出方案。
- $1 \leq |V| \leq 100000, 1 \leq |E| \leq 1000000$

POI2011 SMI-Garbage 题解

- 发现对于一条被覆盖 2 次的边，若被两个环覆盖，则可以删掉这条边并合并两个环，否则可以将环拆成两个。

POI2011 SMI-Garbage 题解

- 发现对于一条被覆盖 2 次的边，若被两个环覆盖，则可以删掉这条边并合并两个环，否则可以将环拆成两个。
- 所以只需要覆盖权值不变的边。

POI2011 SMI-Garbage 题解

- 发现对于一条被覆盖 2 次的边，若被两个环覆盖，则可以删掉这条边并合并两个环，否则可以将环拆成两个。
- 所以只需要覆盖权值不变的边。
- 因为要被拆成若干个简单环，所以必须是欧拉图。

POI2011 SMI-Garbage 题解

- 发现对于一条被覆盖 2 次的边，若被两个环覆盖，则可以删掉这条边并合并两个环，否则可以将环拆成两个。
- 所以只需要覆盖权值不变的边。
- 因为要被拆成若干个简单环，所以必须是欧拉图。
- 跑 Hierholzer 算法即可。

AGC018F Two Trees

- 给你两棵大小为 n 的有根树，你要给每个编号一个权值，使得两棵树上任意子树和均为 1 或 -1。
- 输出方案。
- $1 \leq n \leq 100000$

AGC018F Two Trees 题解

- 可以根据儿子个数奇偶性确定一个点点权的奇偶性。

AGC018F Two Trees 题解

- 可以根据儿子个数奇偶性确定一个点点权的奇偶性。
- 如果一个编号在两棵树中儿子个数奇偶性则一定无解。

AGC018F Two Trees 题解

- 可以根据儿子个数奇偶性确定一个点点权的奇偶性。
- 如果一个编号在两棵树中儿子个数奇偶性则一定无解。
- 大胆猜想点权一定为 $-1/0/1$。

AGC018F Two Trees 题解

- 可以根据儿子个数奇偶性确定一个点点权的奇偶性。
- 如果一个编号在两棵树中儿子个数奇偶性则一定无解。
- 大胆猜想点权一定为 $-1/0/1$。
- 证明就是构造。

AGC018F Two Trees 题解

- 可以根据儿子个数奇偶性确定一个点点权的奇偶性。
- 如果一个编号在两棵树中儿子个数奇偶性则一定无解。
- 大胆猜想点权一定为 $-1/0/1$。
- 证明就是构造。
- 对于儿子个数是奇数的节点，其权值一定为 0。

AGC018F Two Trees 题解

- 可以根据儿子个数奇偶性确定一个点点权的奇偶性。
- 如果一个编号在两棵树中儿子个数奇偶性则一定无解。
- 大胆猜想点权一定为 $-1/0/1$。
- 证明就是构造。
- 对于儿子个数是奇数的节点，其权值一定为 0。
- 对于其他点，在两棵树相同编号的点之间连一条边。

AGC018F Two Trees 题解

- 可以根据儿子个数奇偶性确定一个点点权的奇偶性。
- 如果一个编号在两棵树中儿子个数奇偶性则一定无解。
- 大胆猜想点权一定为 $-1/0/1$。
- 证明就是构造。
- 对于儿子个数是奇数的节点，其权值一定为 0。
- 对于其他点，在两棵树相同编号的点之间连一条边。
- 另外建一个虚点连接两个根。

AGC018F Two Trees 题解

- 可以根据儿子个数奇偶性确定一个点点权的奇偶性。
- 如果一个编号在两棵树中儿子个数奇偶性则一定无解。
- 大胆猜想点权一定为 $-1/0/1$。
- 证明就是构造。
- 对于儿子个数是奇数的节点，其权值一定为 0。
- 对于其他点，在两棵树相同编号的点之间连一条边。
- 另外建一个虚点连接两个根。
- 跑一遍欧拉回路，根据新增的边确定其权值。

AGC018F Two Trees 题解

- 可以根据儿子个数奇偶性确定一个点点权的奇偶性。
- 如果一个编号在两棵树中儿子个数奇偶性则一定无解。
- 大胆猜想点权一定为 $-1/0/1$。
- 证明就是构造。
- 对于儿子个数是奇数的节点，其权值一定为 0。
- 对于其他点，在两棵树相同编号的点之间连一条边。
- 另外建一个虚点连接两个根。
- 跑一遍欧拉回路，根据新增的边确定其权值。
- 为什么这样是对的?

AGC018F Two Trees 题解

- 可以根据儿子个数奇偶性确定一个点点权的奇偶性。
- 如果一个编号在两棵树中儿子个数奇偶性则一定无解。
- 大胆猜想点权一定为 $-1/0/1$。
- 证明就是构造。
- 对于儿子个数是奇数的节点，其权值一定为 0。
- 对于其他点，在两棵树相同编号的点之间连一条边。
- 另外建一个虚点连接两个根。
- 跑一遍欧拉回路，根据新增的边确定其权值。
- 为什么这样是对的?
- 考虑到欧拉图任意一个子图和任意一条欧拉回路，回路进入子图的次数等于离开子图的次数。

AGC018F Two Trees 题解

- 可以根据儿子个数奇偶性确定一个点点权的奇偶性。
- 如果一个编号在两棵树中儿子个数奇偶性则一定无解。
- 大胆猜想点权一定为 $-1/0/1$。
- 证明就是构造。
- 对于儿子个数是奇数的节点，其权值一定为 0。
- 对于其他点，在两棵树相同编号的点之间连一条边。
- 另外建一个虚点连接两个根。
- 跑一遍欧拉回路，根据新增的边确定其权值。
- 为什么这样是对的?
- 考虑到欧拉图任意一个子图和任意一条欧拉回路，回路进入子图的次数等于离开子图的次数。
- 而对于每棵子树，只有一次进出子图不经过新添的边，即连接子树根和其父亲的边。

AGC018F Two Trees 题解

- 可以根据儿子个数奇偶性确定一个点点权的奇偶性。
- 如果一个编号在两棵树中儿子个数奇偶性则一定无解。
- 大胆猜想点权一定为 $-1/0/1$。
- 证明就是构造。
- 对于儿子个数是奇数的节点，其权值一定为 0。
- 对于其他点，在两棵树相同编号的点之间连一条边。
- 另外建一个虚点连接两个根。
- 跑一遍欧拉回路，根据新增的边确定其权值。
- 为什么这样是对的?
- 考虑到欧拉图任意一个子图和任意一条欧拉回路，回路进入子图的次数等于离开子图的次数。
- 而对于每棵子树，只有一次进出子图不经过新添的边，即连接子树根和其父亲的边。
- 故这样构造得到的答案一定合法。

优化建图

什么是优化建图

- 在做某些图论题时如果直接来会变得很繁琐或者复杂度很高，这时候就需要优化建图来解决问题。

什么是优化建图

- 在做某些图论题时如果直接来会变得很繁琐或者复杂度很高，这时候就需要优化建图来解决问题。
- 优化建图的目的一般是简化处理过程或降低复杂度。

什么是优化建图

- 在做某些图论题时如果直接来会变得很繁琐或者复杂度很高，这时候就需要优化建图来解决问题。
- 优化建图的目的一般是简化处理过程或降低复杂度。
- 接下来会结合一些具体的题目。

超级源点

- 在有些题目中我们建出图之后可能会有多个源点，此时可以建立一个超级源点。

超级源点

- 在有些题目中我们建出图之后可能会有多个源点，此时可以建立一个超级源点。
- 我们考虑差分约束的过程，发现每个点的初值不是很好处理。

超级源点

- 在有些题目中我们建出图之后可能会有多个源点，此时可以建立一个超级源点。
- 我们考虑差分约束的过程，发现每个点的初值不是很好处理。
- 一种处理方法是建立一个超级源点 x_0 并钦定他的值是 0，并且增加一些不影响结果的限制，如：每个点的值都要是非负数。

超级源点

- 在有些题目中我们建出图之后可能会有多个源点，此时可以建立一个超级源点。
- 我们考虑差分约束的过程，发现每个点的初值不是很好处理。
- 一种处理方法是建立一个超级源点 x_0 并钦定他的值是 0，并且增加一些不影响结果的限制，如：每个点的值都要是非负数。
- 那么这些限制相当于 $x_i - x_0 \geq 0$，所以只需要从 x_0 向其他点连值为 0 的边即可。

超级源点

- 在有些题目中我们建出图之后可能会有多个源点，此时可以建立一个超级源点。
- 我们考虑差分约束的过程，发现每个点的初值不是很好处理。
- 一种处理方法是建立一个超级源点 x_0 并钦定他的值是 0，并且增加一些不影响结果的限制，如：每个点的值都要是非负数。
- 那么这些限制相当于 $x_i - x_0 \geq 0$，所以只需要从 x_0 向其他点连值为 0 的边即可。
- 又比如，当我们要处理有根树森林的问题时，也可以建立一个超级根节点，并令所有树的根是超级根节点的儿子，这样就可以把森林转化成树。

拆点

- 如果图上点的入度会影响出度，且影响具有一定的规律，可以考虑将点拆成若干个点。

拆点

- 如果图上点的入度会影响出度，且影响具有一定的规律，可以考虑将点拆成若干个点。
- 先看一道题。

拆点

- 给你一张图，每条边有两个权值 (a, b)。
- 一条边的花费为：
  - 若这条边为第一条边，则为 a。
  - 否则，若上一条边的 a 小于这条边，则为 $a - b$，否则为 a。
- $|V| <= 100000, |E| <= 200000$

拆点

- 考虑每条边花费可以 $-b$ 的情况。

拆点

- 考虑每条边花费可以 $-b$ 的情况。
- 发现把这个点的入度按 a 排序后，这条边的贡献能 $-b$ 的上一条边的可能是一个前缀。

拆点

- 考虑每条边花费可以 $-b$ 的情况。
- 发现把这个点的入度按 a 排序后，这条边的贡献能 $-b$ 的上一条边的可能是一个前缀。
- 但是直接暴力连边是不行的，于是我们考虑拆点。

拆点

- 考虑每条边花费可以 $-b$ 的情况。
- 发现把这个点的入度按 a 排序后，这条边的贡献能 $-b$ 的上一条边的可能是一个前缀。
- 但是直接暴力连边是不行的，于是我们考虑拆点。
- 我们对于每个点拆成出边 $+1$ 个点。

拆点

- 考虑每条边花费可以 $-b$ 的情况。
- 发现把这个点的入度按 a 排序后，这条边的贡献能 $-b$ 的上一条边的可能是一个前缀。
- 但是直接暴力连边是不行的，于是我们考虑拆点。
- 我们对于每个点拆成出边 $+1$ 个点。
- 其中出边个点表示每条出边，其只连出一条边，边权为 $a-b$。

拆点

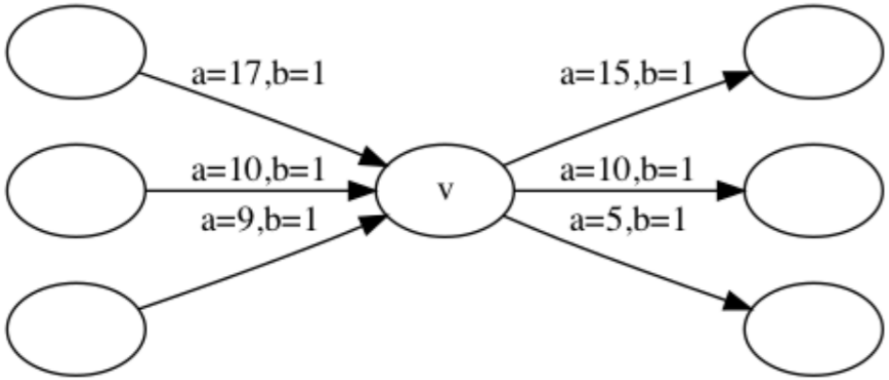
- 考虑每条边花费可以 $-b$ 的情况。
- 发现把这个点的入度按 a 排序后，这条边的贡献能 $-b$ 的上一条边的可能是一个前缀。
- 但是直接暴力连边是不行的，于是我们考虑拆点。
- 我们对于每个点拆成出边 $+1$ 个点。
- 其中出边个点表示每条出边，其只连出一条边，边权为 $a-b$。
- 并且对于每个出边，按从小到大的顺序用 0 边连起来，并在最后接一个点，连出每条边原边权的边。

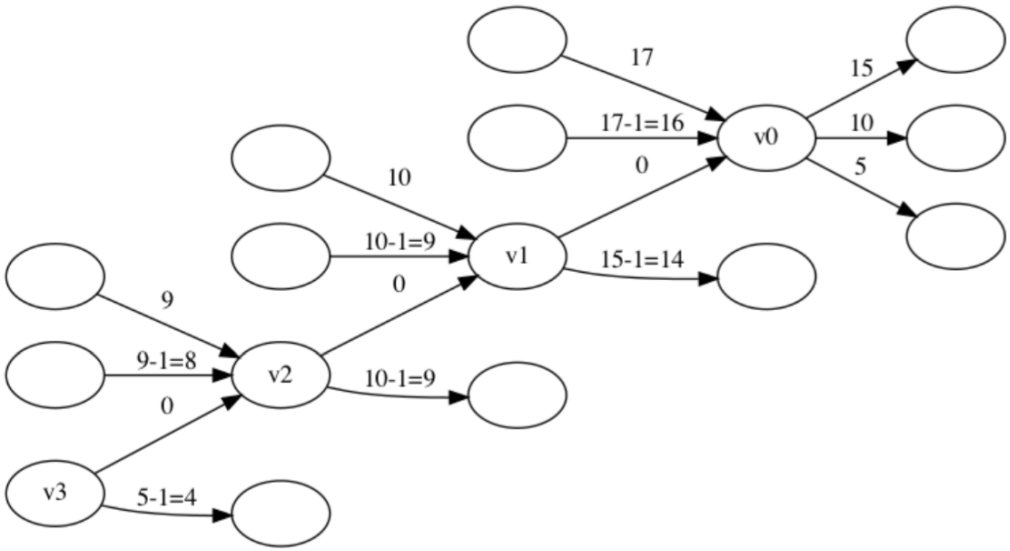
拆点

- 考虑每条边花费可以 $-b$ 的情况。
- 发现把这个点的入度按 a 排序后，这条边的贡献能 $-b$ 的上一条边的可能是一个前缀。
- 但是直接暴力连边是不行的，于是我们考虑拆点。
- 我们对于每个点拆成出边 $+1$ 个点。
- 其中出边个点表示每条出边，其只连出一条边，边权为 $a-b$。
- 并且对于每个出边，按从小到大的顺序用 0 边连起来，并在最后接一个点，连出每条边原边权的边。
- 这样对于每条入边，其对应的可能的出边就变成了一个后缀，所以只要二分出最前面的点即可。

拆点

- 官方题解图示：


a=17,b=1
a=10,b=1
a=9,b=1
v
a=15,b=1
a=10,b=1
a=5,b=1


17
17-1=16
0
v0
15
10
5
10
10-1=9
0
v1
15-1=14
9
9-1=8
0
v2
10-1=9
v3
5-1=4

虚点

- 考虑这样一个问题：你需要实现点集 U 中的点向点集 V 中的点两两连边，$\forall u \in U, v \in V$，边权为 $f(u)+g(v)$。

虚点

- 考虑这样一个问题：你需要实现点集 U 中的点向点集 V 中的点两两连边，$\forall u \in U, v \in V$，边权为 $f(u)+g(v)$。
- 直接连是肯定炸了的，于是我们考虑建立一个中转点。

虚点

- 考虑这样一个问题：你需要实现点集 U 中的点向点集 V 中的点两两连边，$\forall u \in U, v \in V$，边权为 $f(u)+g(v)$。
- 直接连是肯定炸了的，于是我们考虑建立一个中转点。
- 具体地，我们令 U 点向中转点连边权为 $f(u)$ 的边，再从中转点向 V 中的点连边权为 $f(v)$ 的点即可。

前缀和优化建图

- 如果我不保证 $|U|+|V|$ 的大小，但是保证他们都是一段前缀，且边权与连接的点无关怎么办？

前缀和优化建图

- 如果我不保证 $|U|+|V|$ 的大小，但是保证他们都是一段前缀，且边权与连接的点无关怎么办？
- 考虑使用拆点。

前缀和优化建图

- 如果我不保证 $|U|+|V|$ 的大小，但是保证他们都是一段前缀，且边权与连接的点无关怎么办？
- 考虑使用拆点。
- 我们将点从前往后连成一条链，这样从 i 连出去的边可以被 $1\sim i$ 的所有点使用。

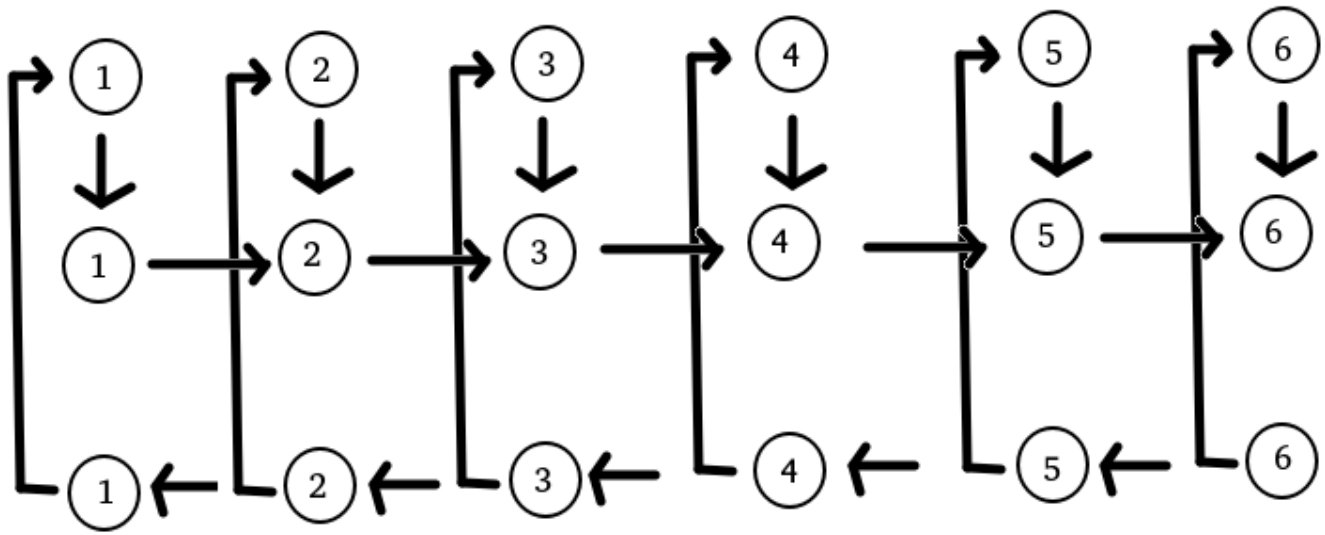
前缀和优化建图

- 如果我不保证 $|U|+|V|$ 的大小，但是保证他们都是一段前缀，且边权与连接的点无关怎么办？
- 考虑使用拆点。
- 我们将点从前往后连成一条链，这样从 i 连出去的边可以被 $1\sim i$ 的所有点使用。
- 同理，将点从后往前连成一条链后，指向 i 的边可以被用来更新 $1\sim i$ 的所有点。

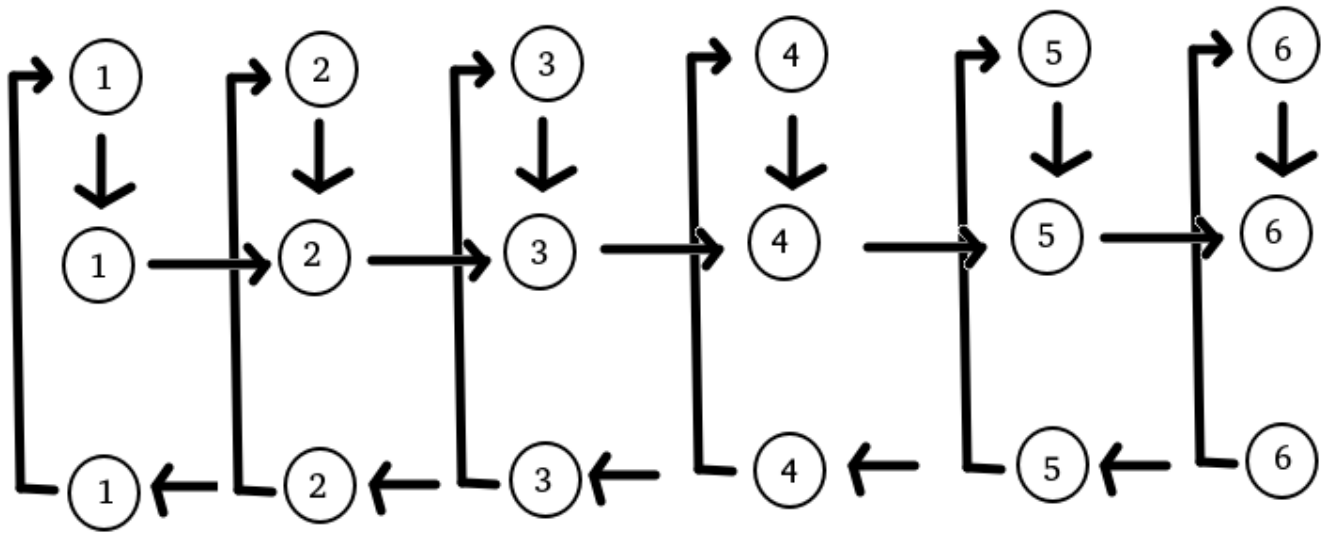
前缀和优化建图

- 如果我不保证 $|U|+|V|$ 的大小，但是保证他们都是一段前缀，且边权与连接的点无关怎么办？
- 考虑使用拆点。
- 我们将点从前往后连成一条链，这样从 i 连出去的边可以被 $1\sim i$ 的所有点使用。
- 同理，将点从后往前连成一条链后，指向 i 的边可以被用来更新 $1\sim i$ 的所有点。
- 但是如果只是将点拆成两个的话手玩一下会发现这样是有错误的，所以考虑再拆出来一个存储答案。

前缀和优化建图

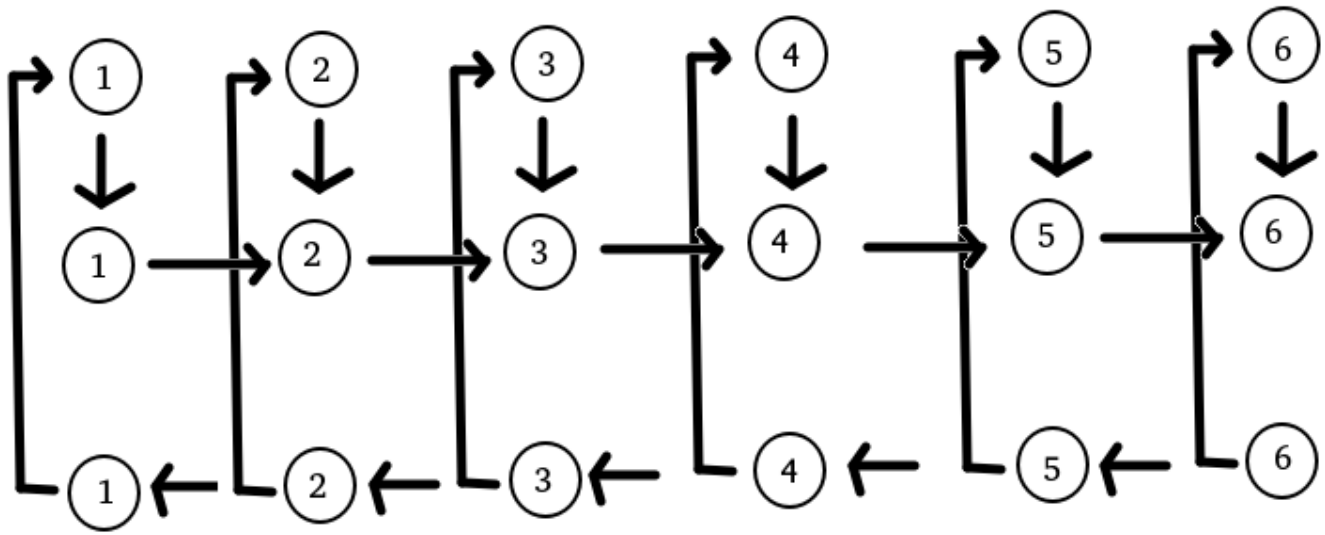


前缀和优化建图



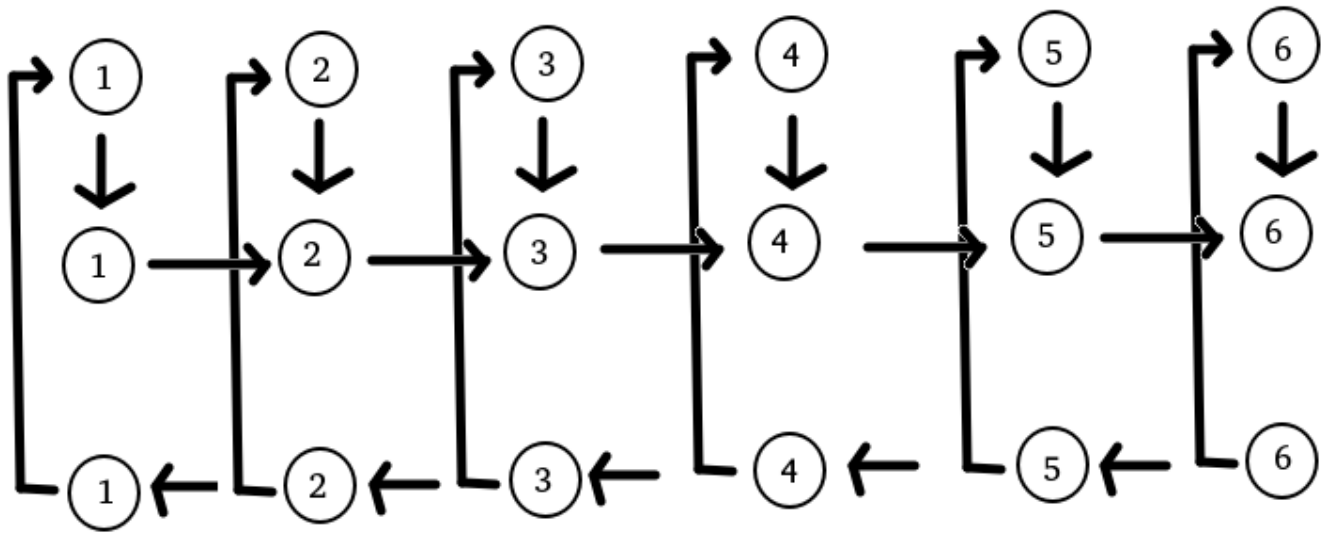
-
- 对于点到点的边：第一行的点之间直接连。

前缀和优化建图



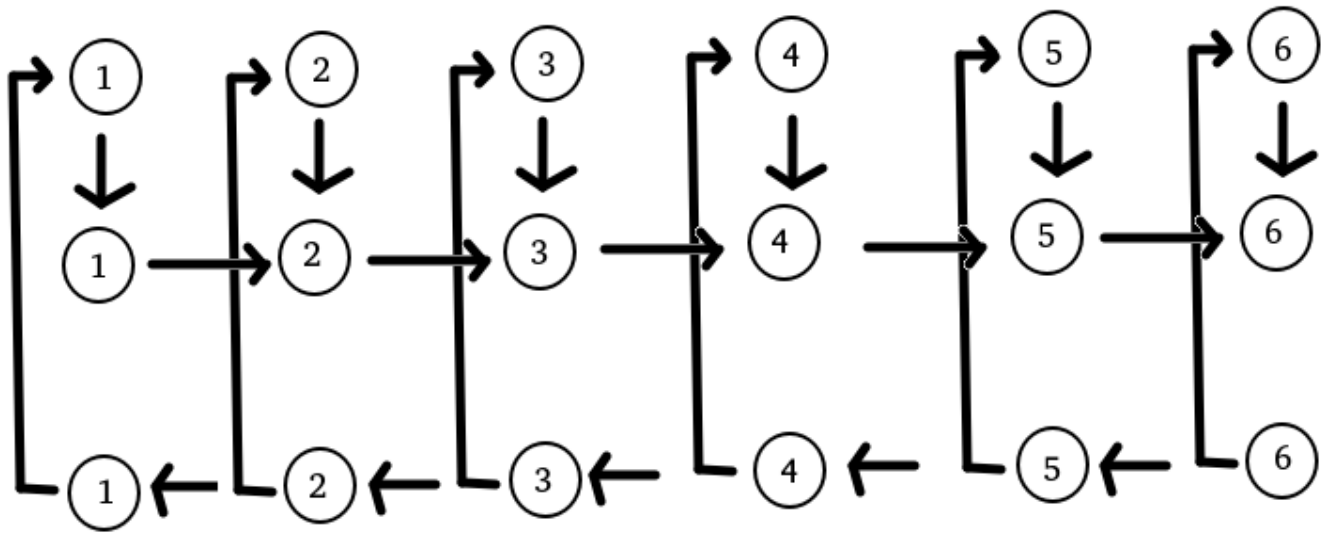
-
- 对于点到点的边：第一行的点之间直接连。
- 对于点到前缀连边：第一行的点连向第三行。

前缀和优化建图



-
- 对于点到点的边：第一行的点之间直接连。
- 对于点到前缀连边：第一行的点连向第三行。
- 对于前缀到点连边：第二行直接连向第一行。

前缀和优化建图



-
- 对于点到点的边：第一行的点之间直接连。
- 对于点到前缀连边：第一行的点连向第三行。
- 对于前缀到点连边：第二行直接连向第一行。
- 对于前缀到前缀连边：第二行连向第三行。

线段树优化建图

- 可是我现在想要实现任意区间到任意区间连边怎么办啊？

线段树优化建图

- 可是我现在想要实现任意区间到任意区间连边怎么办啊？
- 考虑用更强的数据结构维护。

线段树优化建图

- 可是我现在想要实现任意区间到任意区间连边怎么办啊？
- 考虑用更强的数据结构维护。
- 由于连边是不可减的，于是考虑通过合并维护信息的数据结构——线段树。

线段树优化建图

- 可是我现在想要实现任意区间到任意区间连边怎么办啊？
- 考虑用更强的数据结构维护。
- 由于连边是不可减的，于是考虑通过合并维护信息的数据结构——线段树。
- 我们按照线段树的样子建立虚点表示区间，如果是外向树则指向区间 $[l, r]$ 的边可以用来更新 $[l, r]$ 中的所有节点。

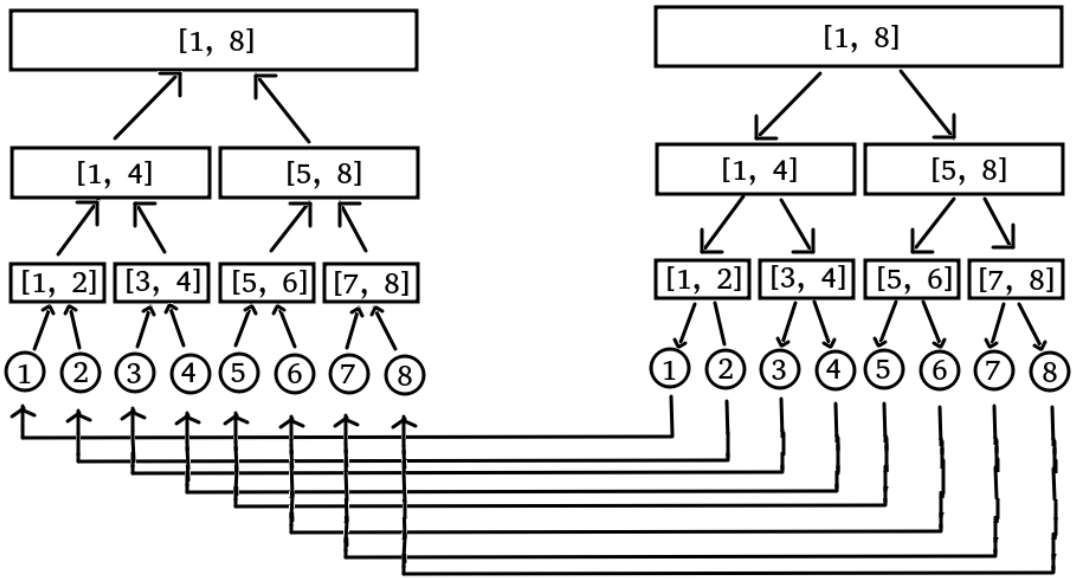
线段树优化建图

- 可是我现在想要实现任意区间到任意区间连边怎么办啊？
- 考虑用更强的数据结构维护。
- 由于连边是不可减的，于是考虑通过合并维护信息的数据结构——线段树。
- 我们按照线段树的样子建立虚点表示区间，如果是外向树则指向区间 $[l, r]$ 的边可以用来更新 $[l, r]$ 中的所有节点。
- 从 $[l, r]$ 指出去的边也可以被 $[l, r]$ 中所有节点使用。

线段树优化建图

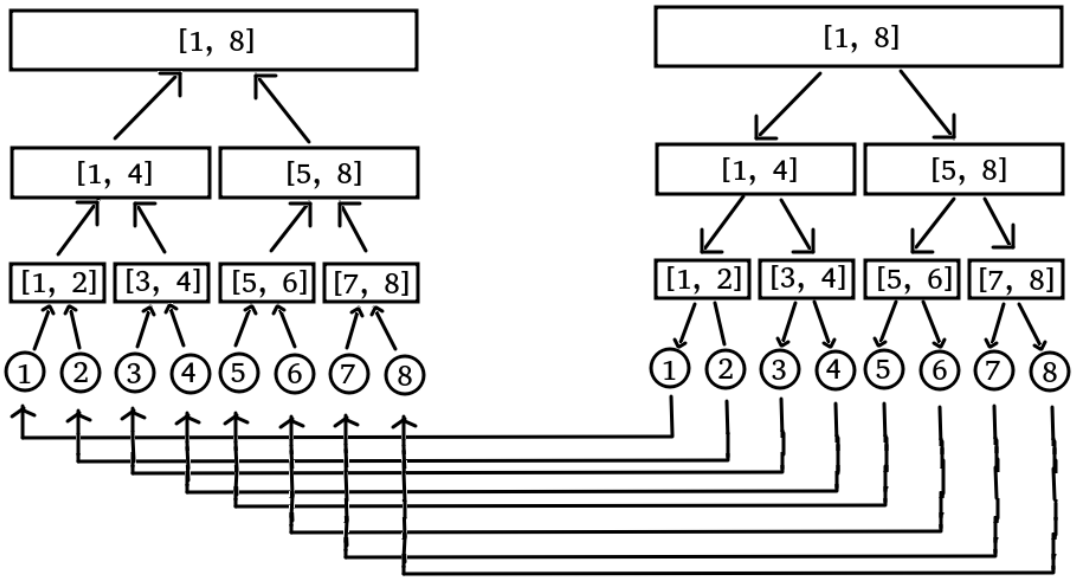
- 可是我现在想要实现任意区间到任意区间连边怎么办啊？
- 考虑用更强的数据结构维护。
- 由于连边是不可减的，于是考虑通过合并维护信息的数据结构——线段树。
- 我们按照线段树的样子建立虚点表示区间，如果是外向树则指向区间 $[l, r]$ 的边可以用来更新 $[l, r]$ 中的所有节点。
- 从 $[l, r]$ 指出去的边也可以被 $[l, r]$ 中所有节点使用。
- 这两个当然不能建在统一棵树上，所以还是把点拆成两份，一个维护出边，一个维护入边。

线段树优化建图



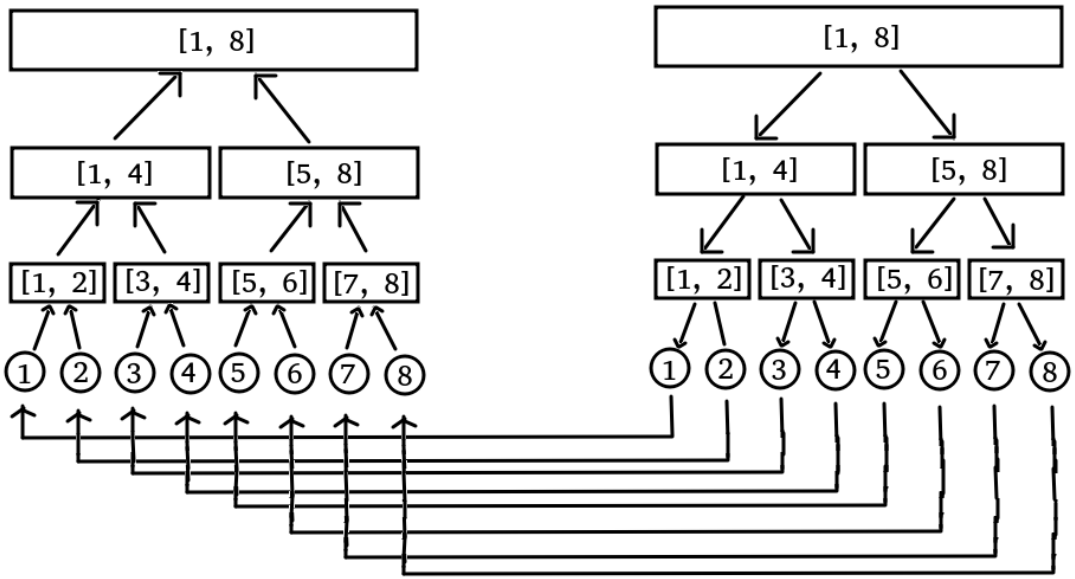
-

线段树优化建图



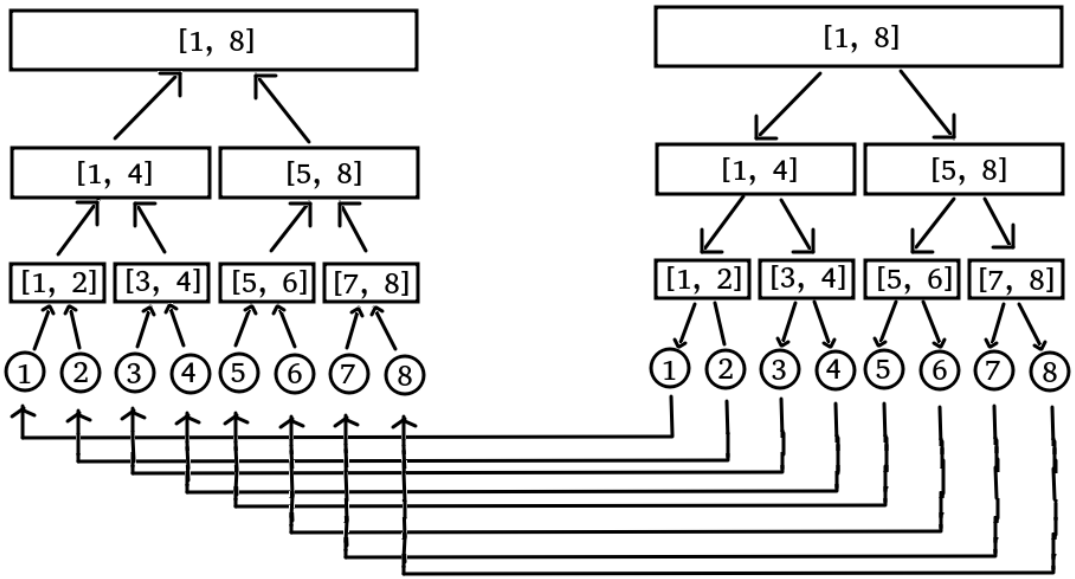
- 对于从区间 $[l_1, r_1]$ 到 $[l_2, r_2]$ 的连边，先将两个区间都在线段树上拆成 $\log$ 个区间，然后两两连边。

线段树优化建图



- 对于从区间 $[l_1, r_1]$ 到 $[l_2, r_2]$ 的连边，先将两个区间都在线段树上拆成 $\log$ 个区间，然后两两连边。
- 但是这样每次边数是 $\log^2$ 的，考虑优化。

线段树优化建图



- 对于从区间 $[l_1, r_1]$ 到 $[l_2, r_2]$ 的连边，先将两个区间都在线段树上拆成 $\log$ 个区间，然后两两连边。
- 但是这样每次边数是 $\log^2$ 的，考虑优化。
- 还是建虚点，对于每次连边新建一个虚点连接所有点即可。

GXOI/GZOI2019 旅行者

- 给你一张图和 k 个关键点，求 k 个关键点两两最短路的最小值。
- $2 \leq k \leq |V| \leq 100000, 1 \leq |E| \leq 500000$

GXOI/GZOI2019 旅行者 题解

- 考虑跑最短路时以一个点集 S 为起点。

GXOI/GZOI2019 旅行者 题解

- 考虑跑最短路时以一个点集 S 为起点。
- 得到的结果是所有点到 S 中点的最短路的最小值。

GXOI/GZOI2019 旅行者 题解

- 考虑跑最短路时以一个点集 S 为起点。
- 得到的结果是所有点到 S 中点的最短路的最小值。
- 所以如果我们建立超级源汇点 s, t，并从 s 向某个点集 S 连 0 边，从某个点集 T 向 t 连 0 边，则 s 到 t 的最短路即为 S 与 T 两两最短路的最小值。

GXOI/GZOI2019 旅行者 题解

- 考虑跑最短路时以一个点集 S 为起点。
- 得到的结果是所有点到 S 中点的最短路的最小值。
- 所以如果我们建立超级源汇点 s, t，并从 s 向某个点集 S 连 0 边，从某个点集 T 向 t 连 0 边，则 s 到 t 的最短路即为 S 与 T 两两最短路的最小值。
- 我们枚举二进制的每一位，将所有 k 个关键点按其编号二进制下这一位的值分组，则最短路最小值的两个点一定会在某一次被分到不同的组中，故可以求出正确答案。

CF1775D Friendly Spiders

- 有 n 个点，每个点有权值 a_i，u, v 之间有边当且仅当 $\gcd(a_u, a_v) \neq 1$。
- 求第 s 个点到第 t 个点的最短路。
- $2 \leq n \leq 300000, 1 \leq a_i \leq 300000$

CF1775D Friendly Spiders 题解

- 两个点 u, v 之间有边当且仅当存在一个质数 p 满足 $p \mid a_u$ 且 $p \mid a_v$。

CF1775D Friendly Spiders 题解

- 两个点 u, v 之间有边当且仅当存在一个质数 p 满足 $p \mid a_u$ 且 $p \mid a_v$。
- 于是枚举每个质数，将所有他的倍数的点之间都两两连边。

CF1775D Friendly Spiders 题解

- 两个点 u, v 之间有边当且仅当存在一个质数 p 满足 $p \mid a_u$ 且 $p \mid a_v$。
- 于是枚举每个质数，将所有他的倍数的点之间都两两连边。
- 但是这样边数不对，所以考虑对每个质数建一个虚点。

CF1775D Friendly Spiders 题解

- 两个点 u, v 之间有边当且仅当存在一个质数 p 满足 $p \mid a_u$ 且 $p \mid a_v$。
- 于是枚举每个质数，将所有他的倍数的点之间都两两连边。
- 但是这样边数不对，所以考虑对每个质数建一个虚点。
- 但是时间复杂度还是不对。

CF1775D Friendly Spiders 题解

- 两个点 u, v 之间有边当且仅当存在一个质数 p 满足 $p \mid a_u$ 且 $p \mid a_v$。
- 于是枚举每个质数，将所有他的倍数的点之间都两两连边。
- 但是这样边数不对，所以考虑对每个质数建一个虚点。
- 但是时间复杂度还是不对。
- 考虑线性筛求出每个数最小的质因数，这样就可以在 $\log$ 时间内求出每个数的所有质因数了。

CF723E One-Way Reform

- 给你一张无向图，要给每条边定向，使得尽量多的点入度等于出度。
- 输出方案。
- $1 \leq |V| \leq 200$

CF723E One-Way Reform 题解

- 答案就是度数为偶数的点的数量。

CF723E One-Way Reform 题解

- 答案就是度数为偶数的点的数量。
- 构造证明，我们建立一个虚点，与所有度数为奇数的点连边。

CF723E One-Way Reform 题解

- 答案就是度数为偶数的点的数量。
- 构造证明，我们建立一个虚点，与所有度数为奇数的点连边。
- 此时图是欧拉图，跑出一条欧拉回路，满足所有偶数度的点入度等于出度。

JOISC2020 美味しい美味しいハンバーグ

- 给你 n 个矩形，你要选择 k 个点，使得任意矩形内部至少有 1 个点。
- $1 \leq n \leq 2 \times 10^5, 1 \leq k \leq 4$

JOISC2020 美味しい美味しいハンバーグ 题解

- 结论：选择的点一定在最靠右的左边界、最靠左的右边界、最靠下的上边界和最靠上的下边界围成的矩形的每条边上各一个。

JOISC2020 美味しい美味しいハンバーグ 题解

- 结论：选择的点一定在最靠右的左边界、最靠左的右边界、最靠下的上边界和最靠上的下边界围成的矩形的每条边上各一个。
- 使用调整法即可证明。

JOISC2020 美味しい美味しいハンバーグ 题解

- 结论：选择的点一定在最靠右的左边界、最靠左的右边界、最靠下的上边界和最靠上的下边界围成的矩形的每条边上各一个。
- 使用调整法即可证明。
- 对于 $k = 1$ 的情况，直接求矩形交。

JOISC2020 美味しい美味しいハンバーグ 题解

- 结论：选择的点一定在最靠右的左边界、最靠左的右边界、最靠下的上边界和最靠上的下边界围成的矩形的每条边上各一个。
- 使用调整法即可证明。
- 对于 $k = 1$ 的情况，直接求矩形交。
- 对于 $k = 2$ 的情况，一定两个点在矩形的某条对角线的两个顶点上，枚举即可。

JOISC2020 美味しい美味しいハンバーグ 题解

- 结论：选择的点一定在最靠右的左边界、最靠左的右边界、最靠下的上边界和最靠上的下边界围成的矩形的每条边上各一个。
- 使用调整法即可证明。
- 对于 $k = 1$ 的情况，直接求矩形交。
- 对于 $k = 2$ 的情况，一定两个点在矩形的某条对角线的两个顶点上，枚举即可。
- 对于 $k = 3$ 的情况，一定有一个点在矩形的顶点上。

JOISC2020 美味しい美味しいハンバーグ 题解

- 结论：选择的点一定在最靠右的左边界、最靠左的右边界、最靠下的上边界和最靠上的下边界围成的矩形的每条边上各一个。
- 使用调整法即可证明。
- 对于 $k = 1$ 的情况，直接求矩形交。
- 对于 $k = 2$ 的情况，一定两个点在矩形的某条对角线的两个顶点上，枚举即可。
- 对于 $k = 3$ 的情况，一定有一个点在矩形的顶点上。
- 剩下两个点放在剩下两条边上，于是问题转化为：有若干个区间，要将区间分成两部分，使得任意一部分的区间的交不为空。

JOISC2020 美味しい美味しいハンバーグ 题解

- 结论：选择的点一定在最靠右的左边界、最靠左的右边界、最靠下的上边界和最靠上的下边界围成的矩形的每条边上各一个。
- 使用调整法即可证明。
- 对于 $k = 1$ 的情况，直接求矩形交。
- 对于 $k = 2$ 的情况，一定两个点在矩形的某条对角线的两个顶点上，枚举即可。
- 对于 $k = 3$ 的情况，一定有一个点在矩形的顶点上。
- 剩下两个点放在剩下两条边上，于是问题转化为：有若干个区间，要将区间分成两部分，使得任意一部分的区间的交不为空。
- 这个是简单贪心，直接按右端点排序即可。

JOISC2020 美味しい美味しいハンバーグ 题解

- 对于 $k = 4$ 的情况，先看看 $k = 3$ 能不能做。

JOISC2020 美味しい美味しいハンバーグ 题解

- 对于 $k = 4$ 的情况，先看看 $k = 3$ 能不能做。
- 否则考虑每个矩形与四条直线的关系。

JOISC2020 美味しい美味しいハンバーグ 题解

- 对于 $k=4$ 的情况，先看看 $k=3$ 能不能做。
- 否则考虑每个矩形与四条直线的关系。
- 如果其与至少 3 条直线有交，则其一定会被覆盖，故不需要考虑。

JOISC2020 美味しい美味しいハンバーグ 题解

- 对于 $k = 4$ 的情况，先看看 $k = 3$ 能不能做。
- 否则考虑每个矩形与四条直线的关系。
- 如果其与至少 3 条直线有交，则其一定会被覆盖，故不需要考虑。
- 如果其与 1 条直线有交，该直线上的点必须在这个范围内。

JOISC2020 美味しい美味しいハンバーグ 题解

- 对于 $k=4$ 的情况，先看看 $k=3$ 能不能做。
- 否则考虑每个矩形与四条直线的关系。
- 如果其与至少 3 条直线有交，则其一定会被覆盖，故不需要考虑。
- 如果其与 1 条直线有交，该直线上的点必须在这个范围内。
- 如果其与 2 条直线有交，则其要么被其中一条直线覆盖，要么被另一条覆盖。

JOISC2020 美味しい美味しいハンバーグ 题解

- 对于 $k=4$ 的情况，先看看 $k=3$ 能不能做。
- 否则考虑每个矩形与四条直线的关系。
- 如果其与至少 3 条直线有交，则其一定会被覆盖，故不需要考虑。
- 如果其与 1 条直线有交，该直线上的点必须在这个范围内。
- 如果其与 2 条直线有交，则其要么被其中一条直线覆盖，要么被另一条覆盖。
- 于是考虑 2-SAT，如果两个矩形有交的两条直线相同（即同为上下或同为左右），则如果这两个矩形没有交，那一定不能选同一条直线。

JOISC2020 美味しい美味しいハンバーグ 题解

- 对于 $k=4$ 的情况，先看看 $k=3$ 能不能做。
- 否则考虑每个矩形与四条直线的关系。
- 如果其与至少 3 条直线有交，则其一定会被覆盖，故不需要考虑。
- 如果其与 1 条直线有交，该直线上的点必须在这个范围内。
- 如果其与 2 条直线有交，则其要么被其中一条直线覆盖，要么被另一条覆盖。
- 于是考虑 2-SAT，如果两个矩形有交的两条直线相同（即同为上下或同为左右），则如果这两个矩形没有交，那一定不能选同一条直线。
- 但是直接来边数是 n^2 的。于是考虑优化建图。

JOISC2020 美味しい美味しいハンバーグ 题解

- 对于 $k=4$ 的情况，先看看 $k=3$ 能不能做。
- 否则考虑每个矩形与四条直线的关系。
- 如果其与至少 3 条直线有交，则其一定会被覆盖，故不需要考虑。
- 如果其与 1 条直线有交，该直线上的点必须在这个范围内。
- 如果其与 2 条直线有交，则其要么被其中一条直线覆盖，要么被另一条覆盖。
- 于是考虑 2-SAT，如果两个矩形有交的两条直线相同（即同为上下或同为左右），则如果这两个矩形没有交，那一定不能选同一条直线。
- 但是直接来边数是 n^2 的。于是考虑优化建图。
- 将区间按右端点排序，在该区间左边无交的区间是一段前缀；将区间按左端点排序，该区间右边无交的是一段后缀。

JOISC2020 美味しい美味しいハンバーグ 题解

- 对于 $k=4$ 的情况，先看看 $k=3$ 能不能做。
- 否则考虑每个矩形与四条直线的关系。
- 如果其与至少 3 条直线有交，则其一定会被覆盖，故不需要考虑。
- 如果其与 1 条直线有交，该直线上的点必须在这个范围内。
- 如果其与 2 条直线有交，则其要么被其中一条直线覆盖，要么被另一条覆盖。
- 于是考虑 2-SAT，如果两个矩形有交的两条直线相同（即同为上下或同为左右），则如果这两个矩形没有交，那一定不能选同一条直线。
- 但是直接来边数是 n^2 的。于是考虑优化建图。
- 将区间按右端点排序，在该区间左边无交的区间是一段前缀；将区间按左端点排序，该区间右边无交的是一段后缀。
- 于是前缀和优化建图即可。

Good Luck&Have Fun