图论选讲

ylsoi

Yali

2019 年 2 月 14 日

Preface

Preface

由于本人水平有限，准备时间不长，讲的内容大家估计都会，放的水题欢迎大家上来秒。

一般的最短路

一般的最短路

在一般的比赛求最短路我们均采用堆优化的 Dijkstra 算法，因为它复杂度稳定，不会被出题人卡。

一般的最短路

在一般的比赛求最短路我们均采用堆优化的 Dijkstra 算法，因为它复杂度稳定，不会被出题人卡。

SPFA 一般情况下不用，但是有负权的话还是不得不用。

一般的最短路

在一般的比赛求最短路我们均采用堆优化的 Dijkstra 算法，因为它复杂度稳定，不会被出题人卡。

SPFA 一般情况下不用，但是有负权的话还是不得不用。

SPFA 的优化

- SLF(Small Lable First):

一般的最短路

在一般的比赛求最短路我们均采用堆优化的 Dijkstra 算法，因为它复杂度稳定，不会被出题人卡。

SPFA 一般情况下不用，但是有负权的话还是不得不用。

SPFA 的优化

- SLF(Small Lable First):
- 元素进队的时候判断和队首哪个状态更优，如果这个更优直接丢进队首，否则丢进队尾。

一般的最短路

在一般的比赛求最短路我们均采用堆优化的 Dijkstra 算法，因为它复杂度稳定，不会被出题人卡。

SPFA 一般情况下不用，但是有负权的话还是不得不用。

SPFA 的优化

- SLF(Small Lable First):
- 元素进队的时候判断和队首哪个状态更优，如果这个更优直接丢进队首，否则丢进队尾。
- LLL(Large Lable Last):

一般的最短路

在一般的比赛求最短路我们均采用堆优化的 Dijkstra 算法，因为它复杂度稳定，不会被出题人卡。

SPFA 一般情况下不用，但是有负权的话还是不得不用。

SPFA 的优化

- SLF(Small Lable First):
- 元素进队的时候判断和队首哪个状态更优，如果这个更优直接丢进队首，否则丢进队尾。
- LLL(Large Lable Last):
- 元素出队的时候判断和队列中元素平均值的大小关系，如果比平均值大，那么丢进队尾。

- 如果想要卡 SPFA 的话，只需要建网格图，行比列的个数多很多，同时行的边权尽可能大，列的边权尽可能小。

- 如果想要卡 SPFA 的话，只需要建网格图，行比列的个数多很多，同时行的边权尽可能大，列的边权尽可能小。
- 当数据范围较小或者需要求任意两点之间的最短路时一般用 Floyd。

- 如果想要卡 SPFA 的话，只需要建网格图，行比列的个数多很多，同时行的边权尽可能大，列的边权尽可能小。
- 当数据范围较小或者需要求任意两点之间的最短路时一般用 Floyd。
- 有的题目要求判负环也会用 Bellman-Ford。

最短路树

最短路树

定义

定义最短路树如下：从源点 1 经过边集 T 到任意一点 i 有且仅有一条路径，且这条路径是整个图 1 到 i 的最短路径，边集 T 构成最短路树。

最短路树

定义

定义最短路树如下：从源点 1 经过边集 T 到任意一点 i 有且仅有一条路径，且这条路径是整个图 1 到 i 的最短路径，边集 T 构成最短路树。

找最短路树的话直接在跑最短路的时候记前驱就好了。

最短路树

定义

定义最短路树如下：从源点 1 经过边集 T 到任意一点 i 有且仅有一条路径，且这条路径是整个图 1 到 i 的最短路径，边集 T 构成最短路树。

找最短路树的话直接在跑最短路的时候记前驱就好了。

网上题目也不是特别多，接下来讲 K 短路的时候会用到。

K 短路

K 短路

定义

给定 s,t，在 s 和 t 的所有路径中，长度为第 k 小的路径即为 s 到 t 的 k 短路。

K 短路

定义

给定 s,t，在 s 和 t 的所有路径中，长度为第 k 小的路径即为 s 到 t 的 k 短路。

A* 算法

K 短路

定义

给定 s,t，在 s 和 t 的所有路径中，长度为第 k 小的路径即为 s 到 t 的 k 短路。

A* 算法

- 一般数据不大的情况下采用 A* 算法，即启发式搜索。

K 短路

定义

给定 s,t，在 s 和 t 的所有路径中，长度为第 k 小的路径即为 s 到 t 的 k 短路。

A* 算法

- 一般数据不大的情况下采用 A* 算法，即启发式搜索。
- 考虑广搜的过程，如果我们将普通的队列换成当前距离的优先队列，那么第 i 个点第 k 次出队列时的距离一定是起点到第 i 个点的 k 短路（必定按照从大到小的顺序出列）。

K 短路

定义

给定 s,t，在 s 和 t 的所有路径中，长度为第 k 小的路径即为 s 到 t 的 k 短路。

A* 算法

- 一般数据不大的情况下采用 A* 算法，即启发式搜索。
- 考虑广搜的过程，如果我们将普通的队列换成当前距离的优先队列，那么第 i 个点第 k 次出队列时的距离一定是起点到第 i 个点的 k 短路（必定按照从大到小的顺序出列）。
- 如果直接暴力的广搜的话，不难发现堆中的状态数很多，因为有的点可能目前距离很近但是最终距离 n 号点距离很远。

A* 算法

- 于是考虑估价函数 $h(x) = f(x) + g(x)$，其中 $f(x)$ 为目前的距离，$g(x)$ 为估计的当前点距离 n 的距离，这里取 $g(x)$ 为从 x 到 n 的最短路。

A* 算法

- 于是考虑估价函数 $h(x) = f(x) + g(x)$，其中 $f(x)$ 为目前的距离，$g(x)$ 为估计的当前点距离 n 的距离，这里取 $g(x)$ 为从 x 到 n 的最短路。
- 不难发现如果我们将堆中的键值换成 $h(x)$ 之后，无用的状态将大大减少，效率提高了很多。

A* 算法

- 于是考虑估价函数 $h(x) = f(x) + g(x)$，其中 $f(x)$ 为目前的距离，$g(x)$ 为估计的当前点距离 n 的距离，这里取 $g(x)$ 为从 x 到 n 的最短路。
- 不难发现如果我们将堆中的键值换成 $h(x)$ 之后，无用的状态将大大减少，效率提高了很多。
- ~~然而还是过不了洛谷的模板~~

A* 算法的缺陷

- 考虑上面的做法哪里有缺陷（~~口胡一波~~）：

A* 算法

- 于是考虑估价函数 $h(x) = f(x) + g(x)$，其中 $f(x)$ 为目前的距离，$g(x)$ 为估计的当前点距离 n 的距离，这里取 $g(x)$ 为从 x 到 n 的最短路。
- 不难发现如果我们将堆中的键值换成 $h(x)$ 之后，无用的状态将大大减少，效率提高了很多。
- ~~然而还是过不了洛谷的模板~~

A* 算法的缺陷

- 考虑上面的做法哪里有缺陷（~~口胡一波~~）：
- 虽然每个点的估价函数比较优秀，同时我们也是尽可能地选则估价函数小的点走，但是从最初的状态转移到最后的状态中间有许多步数，又因为每次是拓展一个点的所有连边，一个估价函数小的点可能会拓展出大量的无用的状态，这些无用的状态就一直放在堆里面，当然过不了。

可并堆做法

- 于是有了一种每次拓展出常数个状态的方法，即用可持久化可并堆来优化。

可并堆做法

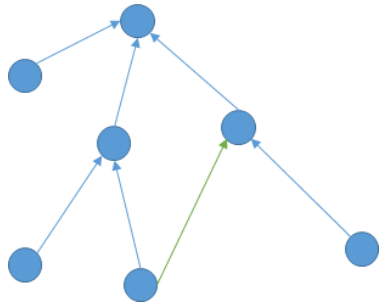
- 于是有了一种每次拓展出常数个状态的方法，即用可持久化可并堆来优化。
- 做法略有不同但是大致思想一样：

可并堆做法

- 于是有了一种每次拓展出常数个状态的方法，即用可持久化可并堆来优化。
- 做法略有不同但是大致思想一样：
- 考虑以 n 为起点的最短路树，一条路径当经过一条非树边 (u,v,w) 的时候，多余的代价为 $-dis_u + w + dis_v$，同时这个代价非负。

可并堆做法

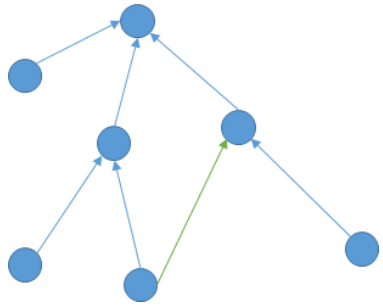
- 于是有了一种每次拓展出常数个状态的方法，即用可持久化可并堆来优化。
- 做法略有不同但是大致思想一样：
- 考虑以 n 为起点的最短路树，一条路径当经过一条非树边 (u,v,w) 的时候，多余的代价为 $-dis_u + w + dis_v$，同时这个代价非负。



(其中蓝色为树边，绿色为非树边)

可并堆做法

- 于是有了一种每次拓展出常数个状态的方法，即用可持久化可并堆来优化。
- 做法略有不同但是大致思想一样：
- 考虑以 n 为起点的最短路树，一条路径当经过一条非树边 (u,v,w) 的时候，多余的代价为 $-dis_u + w + dis_v$，同时这个代价非负。

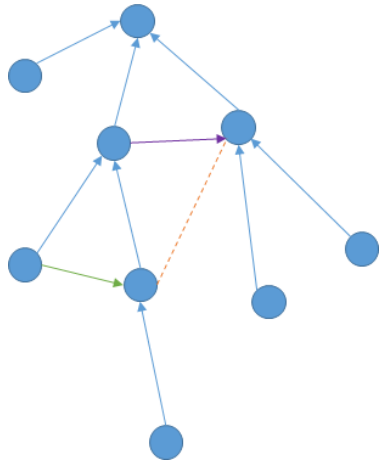


(其中蓝色为树边，绿色为非树边)

- 那么一条路径必定可以通过一个非树边的序列表示出来，那么我们可以枚举出所有的这样的序列同时计算前面的 k 小。

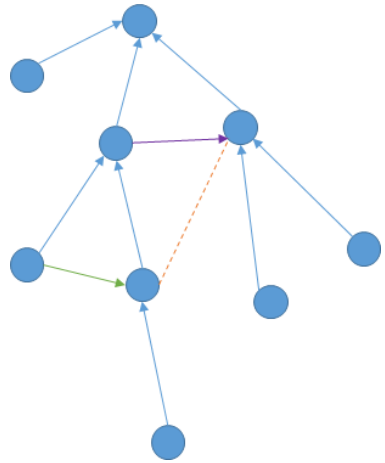
- 考虑如何枚举，依然用一个优先队列来储存目前拓展出来的状态，每次可以在目前的序列最后面添加一个最小的元素，或者替换最后一个元素为目前最后一个元素的后继，这个可以保证拓展出来的状态有限且优秀。

- 考虑如何枚举，依然用一个优先队列来储存目前拓展出来的状态，每次可以在目前的序列最后面添加一个最小的元素，或者替换最后一个元素为目前最后一个元素的后继，这个可以保证拓展出来的状态有限且优秀。



(其中蓝色为树边，黄色为被替换边，紫色绿色为目前的边集.)

- 考虑如何枚举，依然用一个优先队列来储存目前拓展出来的状态，每次可以在目前的序列最后面添加一个最小的元素，或者替换最后一个元素为目前最后一个元素的后继，这个可以保证拓展出来的状态有限且优秀。



(其中蓝色为树边，黄色为被替换边，紫色绿色为目前的边集.)

- 考虑如何实现上一过程，不难发现到了一个新点之后新的可以拓展的元素集合即从这个点到根的路径的所有点的出边，又由于每次需要找最小和后继，对用一个可持久化可并堆或者可持久化平衡树来继承它父亲的状态即可。

k 短路例题

- poj2449 Remmarguts' Date

k 短路例题

- poj2449 Remmarguts' Date
- 魔法猪学院

差分约束系统

差分约束系统

定义

差分约束系统就是给出一些形如 $x - y \leq a$ 不等式的约束，问你是否有满足问题的解，或者求最小，最大解。

差分约束系统

定义

差分约束系统就是给出一些形如 $x-y \leq a$ 不等式的约束，问你是否有满足问题的解，或者求最小，最大解。

转化为图论问题

一般将这个问题转化为图论上的最短路和最长路问题，举个例子，对于 $x-y \leq a$ 和 $y-z \leq b$，连边 (x,y,a),(y,z,b)，这样一条 x->y->z 的路径长度为 a+b 表示 $x-y+y-z \leq a+b$。

差分约束系统

定义

差分约束系统就是给出一些形如 $x-y\leq a$ 不等式的约束，问你是否有满足问题的解，或者求最小，最大解。

转化为图论问题

一般将这个问题转化为图论上的最短路和最长路问题，举个例子，对于 $x-y\leq a$ 和 $y-z\leq b$，连边 (x,y,a),(y,z,b)，这样一条 x->y->z 的路径长度为 a+b 表示 $x-y+y-z\leq a+b$。

于是要求变量 x 和 y 的关系只需要以 x 和 y 分别作为源点和汇点就好了。

欧拉回路

欧拉回路

定义

如果图 G 中的一个路径包括每个边恰好一次又回到起点，则该路径称为欧拉回路。

欧拉回路

定义

如果图 G 中的一个路径包括每个边恰好一次又回到起点，则该路径称为欧拉回路。

有向图和无向图的欧拉回路

- 一个无向图存在欧拉回路，当且仅当该图所有顶点度数都为偶数，且该图是连通图。

欧拉回路

定义

如果图 G 中的一个路径包括每个边恰好一次又回到起点，则该路径称为欧拉回路。

有向图和无向图的欧拉回路

- 一个无向图存在欧拉回路，当且仅当该图所有顶点度数都为偶数，且该图是连通图。
- 一个有向图存在欧拉回路，所有顶点的入度等于出度且该图是连通图。

欧拉回路

定义

如果图 G 中的一个路径包括每个边恰好一次又回到起点，则该路径称为欧拉回路。

有向图和无向图的欧拉回路

- 一个无向图存在欧拉回路，当且仅当该图所有顶点度数都为偶数，且该图是连通图。
- 一个有向图存在欧拉回路，所有顶点的入度等于出度且该图是连通图。
- 如果要输出欧拉回路的话直接 dfs 一遍再用栈来保存，在退出这个节点的时候将它压入栈中即可。

混合图的欧拉回路

混合图的欧拉回路

定义

混合图的欧拉回路是指图中既有无向边又有有向边，同时应当满足不管是有向边还是无向边都只可以走一次的欧拉回路。

混合图的欧拉回路

定义

混合图的欧拉回路是指图中既有无向边又有有向边，同时应当满足不管是有向边还是无向边都只可以走一次的欧拉回路。

如何求判定混合图是否有欧拉回路

这里采用对无向边随机定向之后再用网络最大流来调整的算法。

混合图的欧拉回路

定义

混合图的欧拉回路是指图中既有无向边又有有向边，同时应当满足不管是有向边还是无向边都只可以走一次的欧拉回路。

如何求判定混合图是否有欧拉回路

这里采用对无向边随机定向之后再用网络最大流来调整的算法。

- 首先对图中的无向边随机定向，统计每一个点的入度和出度。

混合图的欧拉回路

定义

混合图的欧拉回路是指图中既有无向边又有有向边，同时应当满足不管是有向边还是无向边都只可以走一次的欧拉回路。

如何求判定混合图是否有欧拉回路

这里采用对无向边随机定向之后再用网络最大流来调整的算法。

- 首先对图中的无向边随机定向，统计每一个点的入度和出度。
- 检查每一个点，如果入度和出度的奇偶性不一样，那么则无解（翻转一条边入度和出度都会变化 1）。

混合图的欧拉回路

定义

混合图的欧拉回路是指图中既有无向边又有有向边，同时应当满足不管是有向边还是无向边都只可以走一次的欧拉回路。

如何求判定混合图是否有欧拉回路

这里采用对无向边随机定向之后再用网络最大流来调整的算法。

- 首先对图中的无向边随机定向，统计每一个点的入度和出度。
- 检查每一个点，如果入度和出度的奇偶性不一样，那么则无解（翻转一条边入度和出度都会变化 1）。
- 对于出度 $>$ 入度的点，我们建源点 S 连接到这个点，容量为 (出度 - 入度)/2，入度 $>$ 出度的点连到汇点 T，容量为 (入度 - 出度)/2。

如何求判定混合图是否有欧拉回路

- 对于一条已经定向了的无向边 (u,v)，在网络图中连一条从 u 到 v 的流量为 1 的边。

如何求判定混合图是否有欧拉回路

- 对于一条已经定向了的无向边 (u,v)，在网络图中连一条从 u 到 v 的流量为 1 的边。
- 最后跑网络最大流，检查流量是否等于从 s 连向的所有边的流量和。

如何求判定混合图是否有欧拉回路

- 对于一条已经定向了的无向边 (u,v)，在网络图中连一条从 u 到 v 的流量为 1 的边。
- 最后跑网络最大流，检查流量是否等于从 s 连向的所有边的流量和。

正确性

考虑这样为什么是对的，将每个点的流量限制认为是每个点需要调整的边数，对于出度 $>$ 入度的点，我们要尽量减少它的出度，反之同理，但是当我们翻转一条边的时候，两个点的入度和出度的差都会变化 2，于是我们将有连边关系的两个点在网络流中也连上一条流量为 1 的边，一条中间的边存在流量表示这条边被翻转，同时两边的点的流量都会产生变化。如果每个点的限制都可以凑齐的话，用最大流一定可以得到结果。

如何求判定混合图是否有欧拉回路

- 对于一条已经定向了的无向边 (u,v)，在网络图中连一条从 u 到 v 的流量为 1 的边。
- 最后跑网络最大流，检查流量是否等于从 s 连向的所有边的流量和。

正确性

考虑这样为什么是对的，将每个点的流量限制认为是每个点需要调整的边数，对于出度 $>$ 入度的点，我们要尽量减少它的出度，反之同理，但是当我们翻转一条边的时候，两个点的入度和出度的差都会变化 2，于是我们将有连边关系的两个点在网络流中也连上一条流量为 1 的边，一条中间的边存在流量表示这条边被翻转，同时两边的点的流量都会产生变化。如果每个点的限制都可以凑齐的话，用最大流一定可以得到结果。

如果需要输出的话我们只需要将流量为 1 的边给翻转，再用有向图的方法就好了。

prufer 序

prufer 序

定义

- prufer 序是带标号无根树的一种表示方法。

prufer 序

定义

- prufer 序是带标号无根树的一种表示方法。
- 生成 prufer 序：一种生成 Prufer 序列的方法是迭代删点，直到原图仅剩两个点。对于一棵顶点已经经过编号的树 T，顶点的编号为 $\{1, 2, ..., n\}$，在第 i 步时，移去所有叶子节点（度为 1 的顶点）中标号最小的顶点和相连的边，并把与它相邻的点的编号加入 Prufer 序列中，重复以上步骤直到原图仅剩 2 个顶点。

prufer 序

定义

- prufer 序是带标号无根树的一种表示方法。
- 生成 prufer 序：一种生成 Prufer 序列的方法是迭代删点，直到原图仅剩两个点。对于一棵顶点已经经过编号的树 T，顶点的编号为 $\{1, 2, ..., n\}$，在第 i 步时，移去所有叶子节点（度为 1 的顶点）中标号最小的顶点和相连的边，并把与它相邻的点的编号加入 Prufer 序列中，重复以上步骤直到原图仅剩 2 个顶点。
- 从 prufer 序还原树：设 $\{a1, a2, ..an - 2\}$ 为一棵有 n 个节点的树的 Prufer 序列，另建一个集合 G 含有元素 1..n，找出集合中最小的未在 Prufer 序列中出现过的数，将该点与 Prufer 序列中首项连一条边，并将该点和 Prufer 序列首项删除，重复操作 n-2 次，将集合中剩余的两个点之间连边即可。

「THUPC2018」城市地铁规划 / City

给定 n 个点，要求你在里面连边形成一棵树，每个度数为 d_i 的点的价值是 v_{d_i}，最大化整棵树的价值和，同时需要输出方案。

「THUPC2018」城市地铁规划 / City

给定 n 个点，要求你在里面连边形成一棵树，每个度数为 d_i 的点的价值是 v_{d_i}，最大化整棵树的价值和，同时需要输出方案。$n \leq 5000$.

思路

思路

- 根据 prufer 序和树一一对应的性质，我们可以直接在最终的 prufer 序上 dp。

思路

- 根据 prufer 序和树一一对应的性质，我们可以直接在最终的 prufer 序上 dp。
- 考虑对于每一种点我们直接考虑它在 prufer 序中出现的次数，然后直接对于 n-2 种物品做完全背包。

思路

- 根据 prufer 序和树一一对应的性质，我们可以直接在最终的 prufer 序上 dp。
- 考虑对于每一种点我们直接考虑它在 prufer 序中出现的次数，然后直接对于 n-2 种物品做完全背包。
- 但是未在 prufer 序中出现的点它的权值我们也需要计算，同时应当满足计算的点的总数为 n 个。

思路

- 根据 prufer 序和树一一对应的性质，我们可以直接在最终的 prufer 序上 dp。
- 考虑对于每一种点我们直接考虑它在 prufer 序中出现的次数，然后直接对于 n-2 种物品做完全背包。
- 但是未在 prufer 序中出现的点它的权值我们也需要计算，同时应当满足计算的点的总数为 n 个。
- 考虑一个小技巧，即在一开始设状态的时候假设所有点的出现次数都是 0 次，然后每次添加一个物品替换里面一个 0 即可。

树

树

求 LCA

- 使用 tarjan 算法可以在线性时间内离线求 lca。

树

求 LCA

- 使用 tarjan 算法可以在线性时间内离线求 lca。
- 一般情况下使用倍增求 lca，时间复杂度 $n \log n$。

树

求 LCA

- 使用 tarjan 算法可以在线性时间内离线求 lca。
- 一般情况下使用倍增求 lca，时间复杂度 $n \log n$。
- 同样用欧拉序 +RMQ 可以在 $n \log n + m$ 的时间内在线求 lca。

树上差分

- 树上差分即通过点的标记，然后再通过对子树的求和来实现整条链或者是一条路径的标记。

CF19E

CF19E

给定一个无向图，求哪些边被删除之后图变成二分图（只可以删除一条边）。

$n \leq 1e6$

思路

思路

- 二分图中不存在奇环，而我们去掉一个奇环只能通过删除奇环上的边来做到，于是我们需要保证我们删除的边必须被所有奇环给包括。

思路

- 二分图中不存在奇环，而我们去掉一个奇环只能通过删除奇环上的边来做到，于是我们需要保证我们删除的边必须被所有奇环给包括。
- 考虑建立图的 dfs 树，如果返祖边上不存在奇环，那么整张图没有奇环，否则：

思路

- 二分图中不存在奇环，而我们去掉一个奇环只能通过删除奇环上的边来做到，于是我们需要保证我们删除的边必须被所有奇环给包括。
- 考虑建立图的 dfs 树，如果返祖边上不存在奇环，那么整张图没有奇环，否则：
  - 一条返祖边可以被删除当且仅当图中只有这一个返祖边存在奇环。

思路

- 二分图中不存在奇环，而我们去掉一个奇环只能通过删除奇环上的边来做到，于是我们需要保证我们删除的边必须被所有奇环给包括。
- 考虑建立图的 dfs 树，如果返祖边上不存在奇环，那么整张图没有奇环，否则：
  - 一条返祖边可以被删除当且仅当图中只有这一个返祖边存在奇环。
  - 一条树边可以被删除当且仅当这条树边被所有的奇环返祖边给包括，同时它不包括任何一个偶环返祖边。

思路

- 二分图中不存在奇环，而我们去掉一个奇环只能通过删除奇环上的边来做到，于是我们需要保证我们删除的边必须被所有奇环给包括。
- 考虑建立图的 dfs 树，如果返祖边上不存在奇环，那么整张图没有奇环，否则：
  - 一条返祖边可以被删除当且仅当图中只有这一个返祖边存在奇环。
  - 一条树边可以被删除当且仅当这条树边被所有的奇环返祖边给包括，同时它不包括任何一个偶环返祖边。
- 具体的实现用树上差分来标记树上路径即可。

树链剖分

树链剖分

定义

树链剖分是按照特定的方式将一棵树划分为若干条互不相交的链的方法。

树链剖分

定义

树链剖分是按照特定的方式将一棵树划分为若干条互不相交的链的方法。

种类

- 重链剖分：按照每个节点的子树大小划分轻重儿子，即目前这条链向下延伸至哪个儿子。

树链剖分

定义

树链剖分是按照特定的方式将一棵树划分为若干条互不相交的链的方法。

种类

- 重链剖分：按照每个节点的子树大小划分轻重儿子，即目前这条链向下延伸至哪个儿子。
- 一般用于处理树上的路径问题，对于树上任意一条路径，我们可以证明，它跨过的链至多为 $\log$ 条。

树链剖分

定义

树链剖分是按照特定的方式将一棵树划分为若干条互不相交的链的方法。

种类

- 重链剖分：按照每个节点的子树大小划分轻重儿子，即目前这条链向下延伸至哪个儿子。
- 一般用于处理树上的路径问题，对于树上任意一条路径，我们可以证明，它跨过的链至多为 $\log$ 条。
- 长链剖分：按照每个节点的子树最大深度划分轻重儿子。

树链剖分

定义

树链剖分是按照特定的方式将一棵树划分为若干条互不相交的链的方法。

种类

- 重链剖分：按照每个节点的子树大小划分轻重儿子，即目前这条链向下延伸至哪个儿子。
- 一般用于处理树上的路径问题，对于树上任意一条路径，我们可以证明，它跨过的链至多为 $\log$ 条。
- 长链剖分：按照每个节点的子树最大深度划分轻重儿子。
- 一般用于处理按照深度为状态的 DP 问题，同时对于树上任意一条路径，我们可以证明，它跨过的链至多为 $\sqrt{n}$ 条。

长链剖分

长链剖分

性质

- 所有的链长 $=n$

长链剖分

性质

- 所有的链长 $=n$
- 任意一个点的 k 次祖先所在的链链长 $\geq k$。

长链剖分

性质

- 所有的链长 $=n$
- 任意一个点的 k 次祖先所在的链链长 $\geq k$。

求一个点的 k 次祖先

- 我们在每条链的端点记录两个表，一个表示向下的点有哪些，一个表示向上的链长个点有哪些。

长链剖分

性质

- 所有的链长 $=$n
- 任意一个点的 k 次祖先所在的链链长 $\geq$k。

求一个点的 k 次祖先

- 我们在每条链的端点记录两个表，一个表示向下的点有哪些，一个表示向上的链长个点有哪些。
- 询问时我们可以将 k 折半，x 首先跳到 x 的 $\frac{k}{2}$ 次祖先 y，不难发现这时 y 所在的链长 $\geq \frac{k}{2}$，同时它距离 x 的 k 次祖先的距离也 $\leq \frac{k}{2}$，不难发现这个时候无论 y 和 x 的 k 次祖先在不在一条链上，都可以通过向上或者向下的数组跳到。

长链剖分

性质

- 所有的链长 $=n$
- 任意一个点的 k 次祖先所在的链链长 $\geq k$。

求一个点的 k 次祖先

- 我们在每条链的端点记录两个表，一个表示向下的点有哪些，一个表示向上的链长个点有哪些。
- 询问时我们可以将 k 折半，x 首先跳到 x 的 $\frac{k}{2}$ 次祖先 y，不难发现这时 y 所在的链长 $\geq \frac{k}{2}$，同时它距离 x 的 k 次祖先的距离也 $\leq \frac{k}{2}$，不难发现这个时候无论 y 和 x 的 k 次祖先在不在一条链上，都可以通过向上或者向下的数组跳到。
- 实际操作中预处理倍增数组来折半。

优化按照深度为状态的 DP

- 对于以深度的状态的 DP，每个点的状态可以直接从重儿子那里继承，对于状态的不同直接通过数组的位移或者是指针的变化来实现。

优化按照深度为状态的 DP

- 对于以深度的状态的 DP，每个点的状态可以直接从重儿子那里继承，对于状态的不同直接通过数组的位移或者是指针的变化来实现。
- 对于轻儿子，直接暴力转移，由于轻儿子必定是一条链的链顶，同时链的总长为 n，所以总的复杂度也是线性。

WC2010

给定一颗带权的树，求一条长路在 $[L, R]$ 的路径，权值的平均数最大。

思路

思路

- 先分数规划，二分答案，然后考虑怎么 check。

思路

- 先分数规划，二分答案，然后考虑怎么 check。
- 考虑一个简单的树型 DP，记 $f_{i,j}$ 为 i 子树内距离 i 为 j 的点中路径长度和最大是多少，然后一个点可以从它的儿子转移过来，在转移的时候每次记录前缀枚举新添加进来的子树深度计算一遍答案。

思路

- 先分数规划，二分答案，然后考虑怎么 check。
- 考虑一个简单的树型 DP，记 $f_{i,j}$ 为 i 子树内距离 i 为 j 的点中路径长度和最大是多少，然后一个点可以从它的儿子转移过来，在转移的时候每次记录前缀枚举新添加进来的子树深度计算一遍答案。
- 这样复杂度 $O(n^2)$，发现转移和子树的深度有关，然后第一颗转移的子树只是涉及到了下标的改变，于是考虑长链剖分优化复杂度，每一次直接先找到重儿子继承，然后再枚举每一条轻边，暴力转移每一条轻边上面的链。

思路

- 先分数规划，二分答案，然后考虑怎么 check。
- 考虑一个简单的树型 DP，记 $f_{i,j}$ 为 i 子树内距离 i 为 j 的点中路径长度和最大是多少，然后一个点可以从它的儿子转移过来，在转移的时候每次记录前缀枚举新添加进来的子树深度计算一遍答案。
- 这样复杂度 $O(n^2)$，发现转移和子树的深度有关，然后第一颗转移的子树只是涉及到了下标的改变，于是考虑长链剖分优化复杂度，每一次直接先找到重儿子继承，然后再枚举每一条轻边，暴力转移每一条轻边上面的链。
- 由于有长度限制，考虑将每一条长链放到线段树上维护，查询的时候直接在线段树上面查询区间最大值即可。

点分治

点分治

定义

- 点分治，是一种处理树上路径问题的工具。

点分治

定义

- 点分治，是一种处理树上路径问题的工具。
- 重心：一棵树最大的子树最小的那个节点叫做重心，重心可能会有 2 个。

点分治

定义

- 点分治，是一种处理树上路径问题的工具。
- 重心：一棵树最大的子树最小的那个节点叫做重心，重心可能会有 2 个。
- 它通过不断地将树划分为若干个互相独立的子树来处理，对于每一次划分选择一个重心同时处理所有经过重心的路径。

点分治

定义

- 点分治，是一种处理树上路径问题的工具。
- 重心：一棵树最大的子树最小的那个节点叫做重心，重心可能会有 2 个。
- 它通过不断地将树划分为若干个互相独立的子树来处理，对于每一次划分选择一个重心同时处理所有经过重心的路径。
- 由于每次选择重心进行分治，所以分治的层数至多为 $\log n$ 层。

点分治

定义

- 点分治，是一种处理树上路径问题的工具。
- 重心：一棵树最大的子树最小的那个节点叫做重心，重心可能会有 2 个。
- 它通过不断地将树划分为若干个互相独立的子树来处理，对于每一次划分选择一个重心同时处理所有经过重心的路径。
- 由于每次选择重心进行分治，所以分治的层数至多为 $\log n$ 层。
- 当我们在一颗子树里面选择了一个重心并且所有经过重心的路径时，往往需要对整颗子树以重心为根遍历，以获取路径上的信息，这样时间复杂度为 $n \log n$。

bzoj3451

给定一棵树，求随机点分治的期望复杂度，每次对一颗大小为 n 的子树需要 $O(n)$ 的复杂度。

$n \leq 1e5$。

思路

思路

- 考虑计算每个点期望下被算的次数，根据期望的线性性，最后将每个点的答案加起来就可以了。

思路

- 考虑计算每个点期望下被算的次数，根据期望的线性性，最后将每个点的答案加起来就可以了。
- 计算点 u 的计算次数可以考虑 v 对点 u 的贡献，即在 v 作为分治重心的时候 u 在 v 所在的子树里面。

思路

- 考虑计算每个点期望下被算的次数，根据期望的线性性，最后将每个点的答案加起来就可以了。
- 计算点 u 的计算次数可以考虑 v 对点 u 的贡献，即在 v 作为分治重心的时候 u 在 v 所在的子树里面。
- 不难发现如果 v 对 u 产生了贡献，那么从 u 到 v 的路径上，v 必定是第一个选的，路径外的点怎么选没有影响，于是期望贡献为 $\frac{1}{dis(u,v)+1}$。

思路

- 考虑计算每个点期望下被算的次数，根据期望的线性性，最后将每个点的答案加起来就可以了。
- 计算点 u 的计算次数可以考虑 v 对点 u 的贡献，即在 v 作为分治重心的时候 u 在 v 所在的子树里面。
- 不难发现如果 v 对 u 产生了贡献，那么从 u 到 v 的路径上，v 必定是第一个选的，路径外的点怎么选没有影响，于是期望贡献为 $\frac{1}{dis(u,v)+1}$。
- 答案即 $\sum_{i=1}^{n}\sum_{j=1}^{n}\frac{1}{dis(i,j)+1}$，又转化成了树上路径问题，考虑点分治，计算出每颗子树内的所有路径长度的出现次数，直接 FFT 优化即可。

uoj276

给定一颗带边权的树，求一条路径使得这条路径上的边权的平均值最接近一个给定的值。

uoj276

给定一颗带边权的树，求一条路径使得这条路径上的边权的平均值最接近一个给定的值。

$n \leq 5e4$

思路

思路

- 既然是求平均值，那么自然而然就想到了分数规划了，即最小化 $|\frac{\sum_{i=1}^{len} w_i}{len} - k|$。

思路

- 既然是求平均值，那么自然而然就想到了分数规划了，即最小化 $|\frac{\sum_{i=1}^{len} w_i}{len} - k|$。
- 然后二分答案 x，考虑是否存在比 x 更优的答案：$|\frac{\sum_{i=1}^{len} w_i}{len} - k| \leq x$，带有绝对值的好像不太好处理，于是将绝对值拆开：$-x \leq \frac{\sum_{i=1}^{len} w_i}{len} - k \leq x$，一般的分数规划都是求一个式子的最值，而这里不难发现需要有两个式子的值同时满足：

思路

- 既然是求平均值，那么自然而然就想到了分数规划了，即最小化 $|\frac{\sum_{i=1}^{len} w_i}{len} - k|$。
- 然后二分答案 x，考虑是否存在比 x 更优的答案：$|\frac{\sum_{i=1}^{len} w_i}{len} - k| \leq x$，带有绝对值的好像不太好处理，于是将绝对值拆开：$-x \leq \frac{\sum_{i=1}^{len} w_i}{len} - k \leq x$，一般的分数规划都是求一个式子的最值，而这里不难发现需要有两个式子的值同时满足：

$$\sum_{i=1}^{len} w_i - k + x \geq 0$$

思路

- 既然是求平均值，那么自然而然就想到了分数规划了，即最小化 $|\frac{\sum_{i=1}^{len} w_i}{len} - k|$。
- 然后二分答案 x，考虑是否存在比 x 更优的答案：$|\frac{\sum_{i=1}^{len} w_i}{len} - k| \leq x$，带有绝对值的好像不太好处理，于是将绝对值拆开：$-x \leq \frac{\sum_{i=1}^{len} w_i}{len} - k \leq x$，一般的分数规划都是求一个式子的最值，而这里不难发现需要有两个式子的值同时满足：

$$\sum_{i=1}^{len} w_i - k + x \geq 0$$

$$\sum_{i=1}^{len} w_i - k - x \leq 0$$

思路

思路

- 由于这里是树上的路径问题，考虑用点分治来解决。

思路

- 由于这里是树上的路径问题，考虑用点分治来解决。
- 考虑以某一个分治重心为根的子树内所有点，按照他们到根的边权和从小到大排序。

思路

- 由于这里是树上的路径问题，考虑用点分治来解决。
- 考虑以某一个分治重心为根的子树内所有点，按照他们到根的边权和从小到大排序。
- 我们考虑每一个点在满足第一维的情况下最小化第二维的值，于是只需要维护两个指针，一个指向目前处理的节点，一个指向满足第一维的节点的范围。

思路

- 由于这里是树上的路径问题，考虑用点分治来解决。
- 考虑以某一个分治重心为根的子树内所有点，按照他们到根的边权和从小到大排序。
- 我们考虑每一个点在满足第一维的情况下最小化第二维的值，于是只需要维护两个指针，一个指向目前处理的节点，一个指向满足第一维的节点的范围。
- 不难发现第一个指针单调移动的时候，第二个指针也单调移动，同时满足第一维的节点的集合不断大，对于这个集合，我们只需要记录两个属于不同子树的点的第二维的最小值即可。

支配树

支配树

如何感性理解支配树

The End

THANKS FOR LISTENING