无向图最小割

王文涛

绍兴市第一中学

- 首先来看这道题（相信不少同学已经做过）：

- 首先来看这道题（相信不少同学已经做过）：

[ZJOI2011]最小割

给定一张边权非负的无向图。有q次询问，每次给出一个数x，求有多少点对满足两点之间的最小割权值不超过x。
数据组数$T \leq 10$，点数$n \leq 150$, 边数$m \leq 3000$, 询问数$q \leq 30$

- 粗一看，这不SB题吗？直接枚举两个点跑最大流就好了呀。

- 粗一看，这不SB题吗？直接枚举两个点跑最大流就好了呀。
- 然而……一看数据范围，复杂度无法承受。

- 粗一看，这不SB题吗？直接枚举两个点跑最大流就好了呀。
- 然而……一看数据范围，复杂度无法承受。
- 怎么办？

- 粗一看，这不SB题吗？直接枚举两个点跑最大流就好了呀。
- 然而……一看数据范围，复杂度无法承受。
- 怎么办？
- 搜题解呗……

- 粗一看，这不SB题吗？直接枚举两个点跑最大流就好了呀。
- 然而……一看数据范围，复杂度无法承受。
- 怎么办？
- 搜题解呗……
- 发现了一个叫最小割树的东西。

- 粗一看，这不SB题吗？直接枚举两个点跑最大流就好了呀。
- 然而……一看数据范围，复杂度无法承受。
- 怎么办？
- 搜题解呗……
- 发现了一个叫最小割树的东西。
- 任意两点间的最小割的权值就是这两点在最小割树上路径边权的最小值。

- 粗一看，这不SB题吗？直接枚举两个点跑最大流就好了呀。
- 然而……一看数据范围，复杂度无法承受。
- 怎么办？
- 搜题解呗……
- 发现了一个叫最小割树的东西。
- 任意两点间的最小割的权值就是这两点在最小割树上路径边权的最小值。
- 有了这个东西，这题似乎就可以随便做了。

- 粗一看，这不SB题吗？直接枚举两个点跑最大流就好了呀。
- 然而……一看数据范围，复杂度无法承受。
- 怎么办？
- 搜题解呗……
- 发现了一个叫最小割树的东西。
- 任意两点间的最小割的权值就是这两点在最小割树上路径边权的最小值。
- 有了这个东西，这题似乎就可以随便做了。
- 所以怎么构这个树呢？

- 粗一看，这不SB题吗？直接枚举两个点跑最大流就好了呀。
- 然而……一看数据范围，复杂度无法承受。
- 怎么办？
- 搜题解呗……
- 发现了一个叫最小割树的东西。
- 任意两点间的最小割的权值就是这两点在最小割树上路径边权的最小值。
- 有了这个东西，这题似乎就可以随便做了。
- 所以怎么构这个树呢？
- Gusfield算法！

- 在讲这个算法之前，我们先定义一些记号。

- 在讲这个算法之前，我们先定义一些记号。
- 令整张图的点集为V，边集为E。

- 在讲这个算法之前，我们先定义一些记号。
- 令整张图的点集为V，边集为E。
- 对于某条边$e \in E$，$c(e)$为e的边权。$c(e) \geq 0$。

- 在讲这个算法之前，我们先定义一些记号。
- 令整张图的点集为V，边集为E。
- 对于某条边$e \in E$，$c(e)$为e的边权。$c(e) \geq 0$。
- 定义一个划分为二元组$(U, V-U)$ $(U \subset V,\ U \neq \emptyset)$。

- 在讲这个算法之前，我们先定义一些记号。
- 令整张图的点集为V，边集为E。
- 对于某条边$e \in E$，$c(e)$为e的边权。$c(e) \geq 0$。
- 定义一个划分为二元组$(U, V-U)$ $(U \subset V,\ U \neq \emptyset)$。
- 对于两个不相交的点集(A, B)，定义$c(A, B)$为所有一端在A、另一端在B的边的边权之和。一个划分的权值就是$c(U, V-U)$。

- 在讲这个算法之前，我们先定义一些记号。
- 令整张图的点集为V，边集为E。
- 对于某条边$e \in E$，$c(e)$为e的边权。$c(e) \geq 0$。
- 定义一个划分为二元组$(U, V-U)$ $(U \subset V,\ U \neq \emptyset)$。
- 对于两个不相交的点集(A, B)，定义$c(A, B)$为所有一端在A、另一端在B的边的边权之和。一个划分的权值就是$c(U, V-U)$。
- 对于某两点$u, v \in V,\ u \neq v$，定义划分$(U, V-U)$为u, v的一个割如果$u \in U,\ v \in V-U$。我们称U为这个割的u侧，V为这个割的v侧。

- 在讲这个算法之前，我们先定义一些记号。
- 令整张图的点集为V，边集为E。
- 对于某条边$e \in E$，$c(e)$为e的边权。$c(e) \geq 0$。
- 定义一个划分为二元组$(U, V-U)$ $(U \subset V,\ U \neq \emptyset)$。
- 对于两个不相交的点集(A, B)，定义$c(A, B)$为所有一端在A、另一端在B的边的边权之和。一个划分的权值就是$c(U, V-U)$。
- 对于某两点$u, v \in V,\ u \neq v$，定义划分$(U, V-U)$为u, v的一个割如果$u \in U,\ v \in V-U$。我们称U为这个割的u侧，V为这个割的v侧。
- 定义$\alpha(u, v)$为u, v的最小割，即权值最小的u, v 割。

- 在讲这个算法之前，我们先定义一些记号。
- 令整张图的点集为V，边集为E。
- 对于某条边$e \in E$，$c(e)$为e的边权。$c(e) \geq 0$。
- 定义一个划分为二元组$(U, V-U)$ $(U \subset V,\ U \neq \emptyset)$。
- 对于两个不相交的点集(A, B)，定义$c(A, B)$为所有一端在A、另一端在B的边的边权之和。一个划分的权值就是$c(U, V-U)$。
- 对于某两点$u, v \in V,\ u \neq v$，定义划分$(U, V-U)$为u, v的一个割如果$u \in U,\ v \in V-U$。我们称U为这个割的u侧，V为这个割的v侧。
- 定义$\alpha(u, v)$为u, v的最小割，即权值最小的u, v 割。
- 定义$\lambda(u, v)$为u, v最小割的权值，即$c(\alpha(u, v))$。

- 首先我们来证明这样一个性质：

- 首先我们来证明这样一个性质：
- 对于任意不同的三点a, b, c，$\lambda(a, b) \geq \min(\lambda(a, c), \lambda(c, b))$

- 首先我们来证明这样一个性质：
- 对于任意不同的三点a, b, c，$\lambda(a, b) \geq \min(\lambda(a, c), \lambda(c, b))$
- 证明：

- 首先我们来证明这样一个性质：
- 对于任意不同的三点a, b, c，$\lambda(a, b) \geq \min(\lambda(a, c), \lambda(c, b))$
- 证明：
- 考虑$\lambda(a, b), \lambda(b, c), \lambda(c, a)$中最小值，不妨假设是$\lambda(a, b)$。

- 首先我们来证明这样一个性质：
- 对于任意不同的三点a, b, c，$\lambda(a, b) \geq \min(\lambda(a, c), \lambda(c, b))$
- 证明：
- 考虑$\lambda(a, b), \lambda(b, c), \lambda(c, a)$中最小值，不妨假设是$\lambda(a, b)$。
- 讨论c在$\alpha(a, b)$的哪一侧，不妨假设是b侧，另一侧同理。

- 首先我们来证明这样一个性质：
- 对于任意不同的三点a, b, c，$\lambda(a, b) \geq \min(\lambda(a, c), \lambda(c, b))$
- 证明：
- 考虑$\lambda(a, b), \lambda(b, c), \lambda(c, a)$中最小值，不妨假设是$\lambda(a, b)$。
- 讨论c在$\alpha(a, b)$的哪一侧，不妨假设是b侧，另一侧同理。
- 那么显然有$\lambda(a, c) \leq \lambda(a, b)$。

- 首先我们来证明这样一个性质：
- 对于任意不同的三点a, b, c，$\lambda(a, b) \geq \min(\lambda(a, c), \lambda(c, b))$
- 证明：
- 考虑$\lambda(a, b), \lambda(b, c), \lambda(c, a)$中最小值，不妨假设是$\lambda(a, b)$。
- 讨论c在$\alpha(a, b)$的哪一侧，不妨假设是b侧，另一侧同理。
- 那么显然有$\lambda(a, c) \leq \lambda(a, b)$。
- 又由于$\lambda(a, b)$是最小值，所以$\lambda(a, c) \geq \lambda(a, b)$。

- 首先我们来证明这样一个性质：
- 对于任意不同的三点a, b, c，$\lambda(a, b) \geq \min(\lambda(a, c), \lambda(c, b))$
- 证明：
- 考虑$\lambda(a, b), \lambda(b, c), \lambda(c, a)$中最小值，不妨假设是$\lambda(a, b)$。
- 讨论c在$\alpha(a, b)$的哪一侧，不妨假设是b侧，另一侧同理。
- 那么显然有$\lambda(a, c) \leq \lambda(a, b)$。
- 又由于$\lambda(a, b)$是最小值，所以$\lambda(a, c) \geq \lambda(a, b)$。
- 得到$\lambda(a, c) = \lambda(a, b)$。

- 首先我们来证明这样一个性质：
- 对于任意不同的三点a, b, c，$\lambda(a, b) \geq \min(\lambda(a, c), \lambda(c, b))$
- 证明：
- 考虑$\lambda(a, b), \lambda(b, c), \lambda(c, a)$中最小值，不妨假设是$\lambda(a, b)$。
- 讨论c在$\alpha(a, b)$的哪一侧，不妨假设是b侧，另一侧同理。
- 那么显然有$\lambda(a, c) \leq \lambda(a, b)$。
- 又由于$\lambda(a, b)$是最小值，所以$\lambda(a, c) \geq \lambda(a, b)$。
- 得到$\lambda(a, c) = \lambda(a, b)$。
- 由此可见对于任意不同的三点a, b, c，$\lambda(a, b), \lambda(b, c), \lambda(c, a)$必然是两个相同的较小值和一个较大值。命题得证。

- 首先我们来证明这样一个性质：
- 对于任意不同的三点a, b, c，$\lambda(a, b) \geq \min(\lambda(a, c), \lambda(c, b))$
- 证明：
- 考虑$\lambda(a, b), \lambda(b, c), \lambda(c, a)$中最小值，不妨假设是$\lambda(a, b)$。
- 讨论c在$\alpha(a, b)$的哪一侧，不妨假设是b侧，另一侧同理。
- 那么显然有$\lambda(a, c) \leq \lambda(a, b)$。
- 又由于$\lambda(a, b)$是最小值，所以$\lambda(a, c) \geq \lambda(a, b)$。
- 得到$\lambda(a, c) = \lambda(a, b)$。
- 由此可见对于任意不同的三点a, b, c，$\lambda(a, b), \lambda(b, c), \lambda(c, a)$必然是两个相同的较小值和一个较大值。命题得证。
- 实际上，我们还可以扩展这个性质，得到：
- 对于任意不同的两点u, v，有$\lambda(u, v) \geq \min(\lambda(u, w_1), \lambda(w_1, w_2), \ldots, \lambda(w_k, v))$。

- 接下来我们再来证明这样一个事情：

- 接下来我们再来证明这样一个事情：
- 对于点x, y, z, w，令$\alpha(x, y) = (X, Y)$，令$\alpha(z, w) = (Z, W)$。

- 接下来我们再来证明这样一个事情：
- 对于点x, y, z, w，令$\alpha(x, y) = (X, Y)$，
  令$\alpha(z, w) = (Z, W)$。
- 那么(X, Y)和(Z, W)可以不相交。即可以做
  到$X \cap Z$, $X \cap W$, $Y \cap Z$, $Y \cap W$至少有一个为空。

- 接下来我们再来证明这样一个事情：
- 对于点x, y, z, w，令$\alpha(x, y) = (X, Y)$，
  令$\alpha(z, w) = (Z, W)$。
- 那么(X, Y)和(Z, W)可以不相交。即可以做
  到$X \cap Z$, $X \cap W$, $Y \cap Z$, $Y \cap W$至少有一个为空。
- 我们分三种情况讨论，其余情况都可以通过调整转化为这些
  情况之一。

- 接下来我们再来证明这样一个事情：
- 对于点x, y, z, w，令$\alpha(x, y) = (X, Y)$，
  令$\alpha(z, w) = (Z, W)$。
- 那么(X, Y)和(Z, W)可以不相交。即可以做
  到$X \cap Z$, $X \cap W$, $Y \cap Z$, $Y \cap W$至少有一个为空。
- 我们分三种情况讨论，其余情况都可以通过调整转化为这些情况之一。
- 情况1：$z \in X$, $w \in Y$, $x \in Z$, $y \in W$

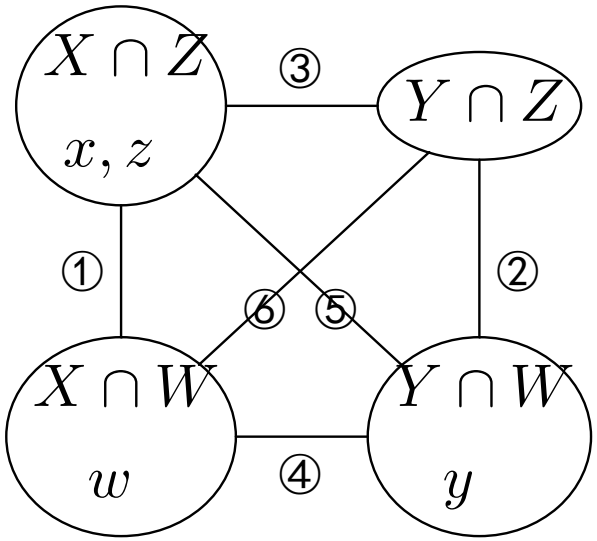
- 接下来我们再来证明这样一个事情：
- 对于点x, y, z, w，令$\alpha(x, y) = (X, Y)$，
  令$\alpha(z, w) = (Z, W)$。
- 那么(X, Y)和(Z, W)可以不相交。即可以做
  到$X \cap Z,\ X \cap W,\ Y \cap Z,\ Y \cap W$至少有一个为空。
- 我们分三种情况讨论，其余情况都可以通过调整转化为这些情况之一。
- 情况1：$z \in X,\ w \in Y,\ x \in Z,\ y \in W$
- 那么(Z, W)也是x, y的一个割，(X, Y)也是z, w的一个割。

- 接下来我们再来证明这样一个事情：
- 对于点x, y, z, w，令$\alpha(x, y) = (X, Y)$，
  令$\alpha(z, w) = (Z, W)$。
- 那么(X, Y)和(Z, W)可以不相交。即可以做
  到$X \cap Z,\ X \cap W,\ Y \cap Z,\ Y \cap W$至少有一个为空。
- 我们分三种情况讨论，其余情况都可以通过调整转化为这些情况之一。
- 情况1：$z \in X,\ w \in Y,\ x \in Z,\ y \in W$
- 那么(Z, W)也是x, y的一个割，(X, Y)也是z, w的一个割。
- 由于都是最小割，那么这两个割权值相同，都是另一对点的最小割。

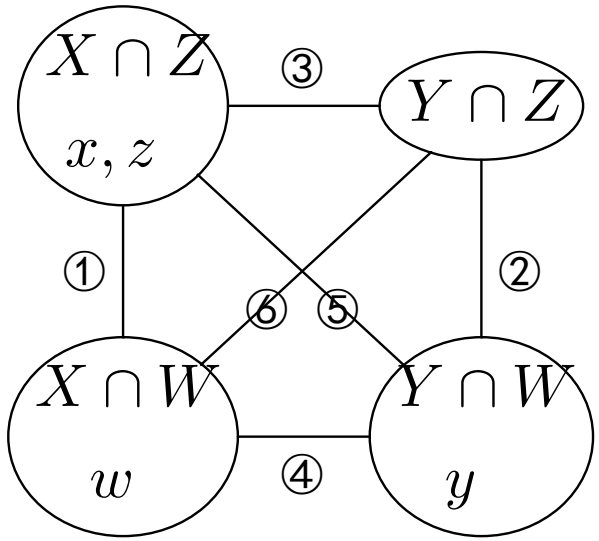
- 接下来我们再来证明这样一个事情：
- 对于点x, y, z, w，令$\alpha(x, y) = (X, Y)$，
  令$\alpha(z, w) = (Z, W)$。
- 那么(X, Y)和(Z, W)可以不相交。即可以做
  到$X \cap Z,\ X \cap W,\ Y \cap Z,\ Y \cap W$至少有一个为空。
- 我们分三种情况讨论，其余情况都可以通过调整转化为这些情况之一。
- 情况1：$z \in X,\ w \in Y,\ x \in Z,\ y \in W$
- 那么(Z, W)也是x, y的一个割，(X, Y)也是z, w的一个割。
- 由于都是最小割，那么这两个割权值相同，都是另一对点的最小割。
- 所以可以把$\alpha(z, w)$调整为(X, Y)，这样就不相交了。

- 情况2：$z \in X,\ w \in X,\ x \in Z,\ y \in W$

- 情况2：$z \in X,\ w \in X,\ x \in Z,\ y \in W$



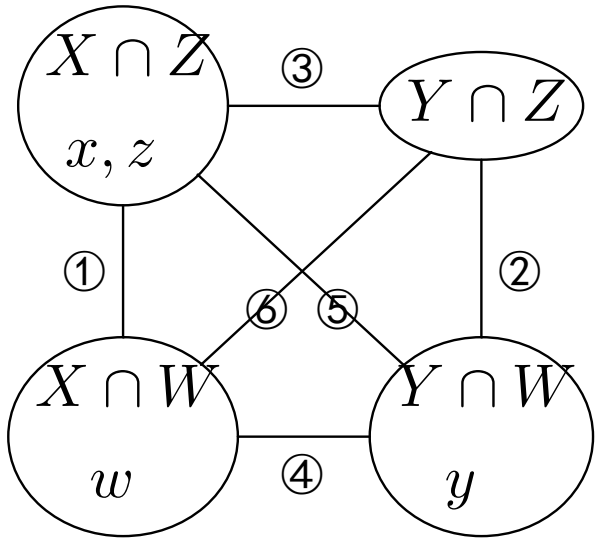
- 情况2：$z \in X,\ w \in X,\ x \in Z,\ y \in W$



- 可以发现

$$c(Z,W) + C(X,Y) \geq c(X \cap Z, Y \cup W) + c(X \cup Z, Y \cap W)$$

- 情况2：$z \in X,\ w \in X,\ x \in Z,\ y \in W$



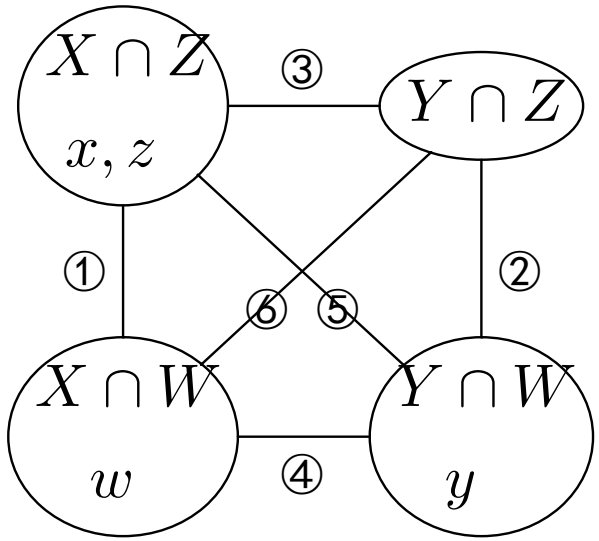
- 可以发现

$$c(Z,W) + C(X,Y) \geq c(X \cap Z, Y \cup W) + c(X \cup Z, Y \cap W)$$

- 因为

$$(①+②+⑤+⑥)+(③+④+⑤+⑥) \geq (①+③+⑤)+(②+④+⑤)$$

- 情况2：$z \in X,\ w \in X,\ x \in Z,\ y \in W$

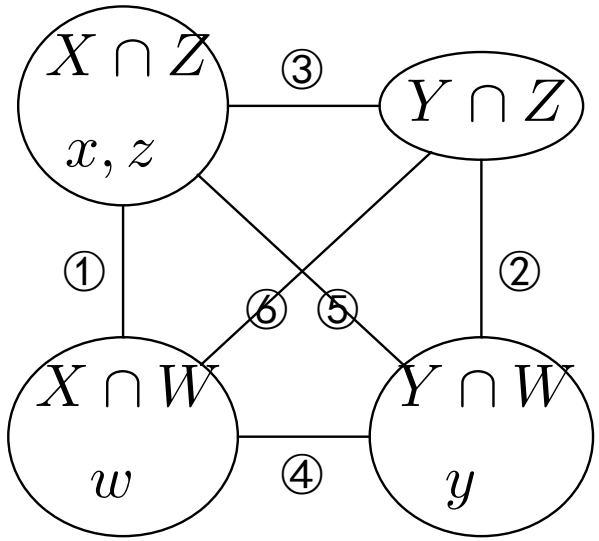


$$c(Z,W) + C(X,Y) \geq c(X \cap Z, Y \cup W) + c(X \cup Z, Y \cap W)$$

- 观察可知$(X \cap Z, Y \cup W)$也是z, w的一个割，同时由于(Z,W)是z,w的最小割，所以

$$c(X \cap Z, Y \cup W) \geq c(Z,W)$$

- 情况2：$z \in X,\ w \in X,\ x \in Z,\ y \in W$

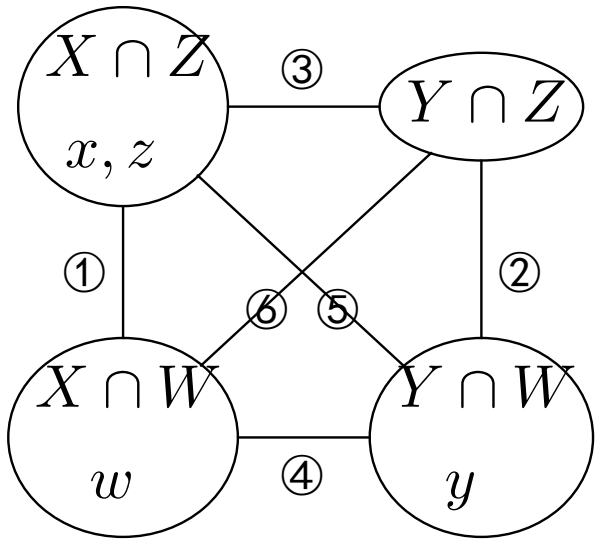


$$c(Z,W) + C(X,Y) \geq c(X \cap Z, Y \cup W) + c(X \cup Z, Y \cap W)$$

- 同理，观察可知$(X \cup Z, Y \cap W)$也是x, y的一个割，同时由于(X,Y)是z, w的最小割，所以

$$c(X \cup Z, Y \cap W) \geq c(X,Y)$$

- 情况2：$z \in X,\ w \in X,\ x \in Z,\ y \in W$



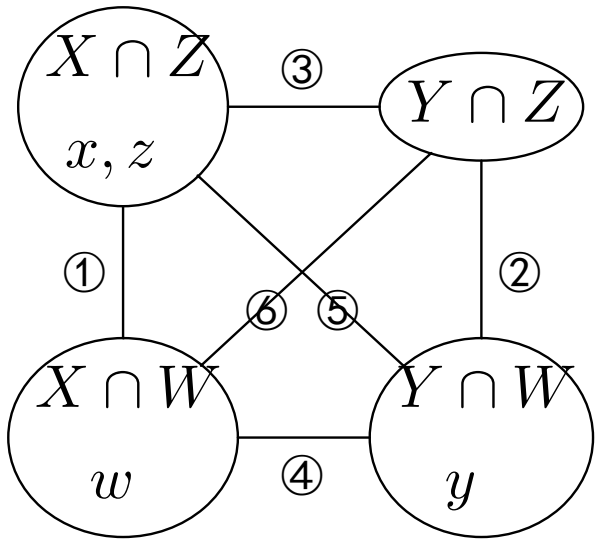
- 因此

$$c(Z,W) + C(X,Y) = c(X \cap Z, Y \cup W) + c(X \cup Z, Y \cap W)$$

并且

$$c(Z,W) = c(X \cap Z, Y \cup W),\ C(X,Y) = c(X \cup Z, Y \cap W)$$

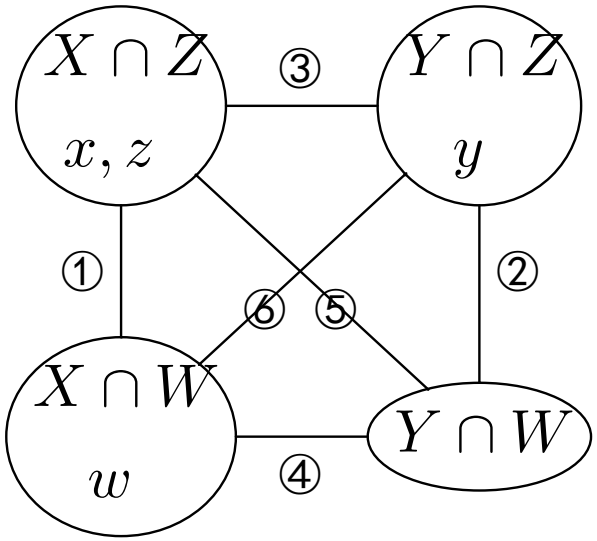
- 情况2：$z \in X,\ w \in X,\ x \in Z,\ y \in W$



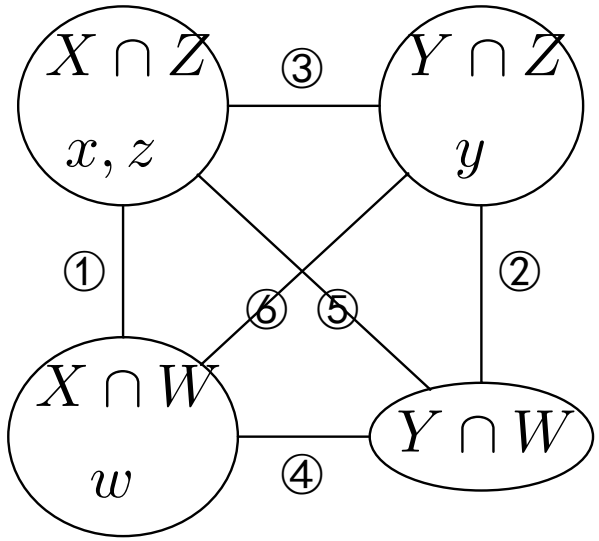
- 于是，可以把$\alpha(Z, W)$调整为$(X \cap Z, Y \cup W)$，这样就不相交了。

- 情况3：$z \in X,\ w \in X,\ x \in Z,\ y \in Z$

- 情况3：$z \in X,\ w \in X,\ x \in Z,\ y \in Z$



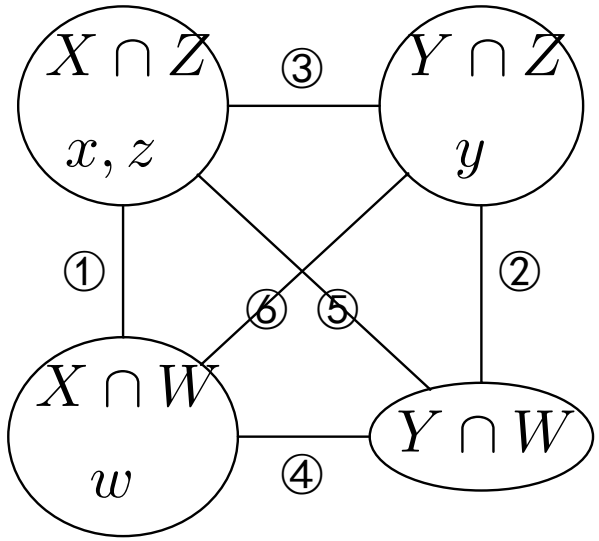
- 情况3：$z \in X,\ w \in X,\ x \in Z,\ y \in Z$



- 可以发现

$$c(Z, W) + C(X, Y) \geq c(X \cap W, Y \cup Z) + c(X \cup W, Y \cap Z)$$

- 情况3：$z \in X,\ w \in X,\ x \in Z,\ y \in Z$



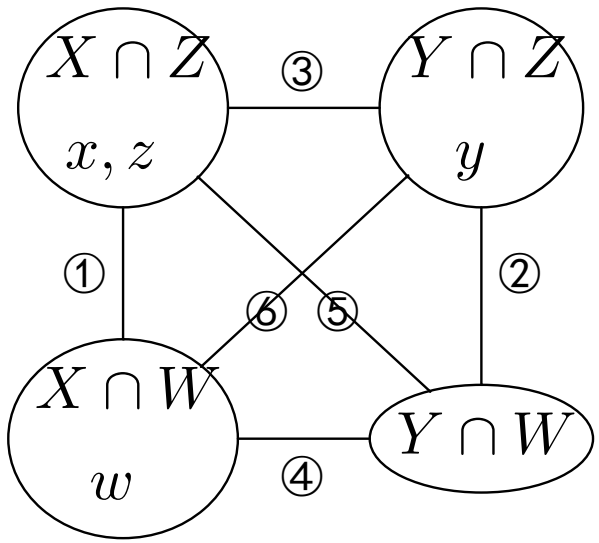
- 可以发现

$$c(Z, W) + C(X, Y) \geq c(X \cap W, Y \cup Z) + c(X \cup W, Y \cap Z)$$

- 因为

$$(①+②+⑤+⑥)+(③+④+⑤+⑥) \geq (①+④+⑥)+(②+③+⑥)$$

- 情况3：$z \in X,\ w \in X,\ x \in Z,\ y \in Z$

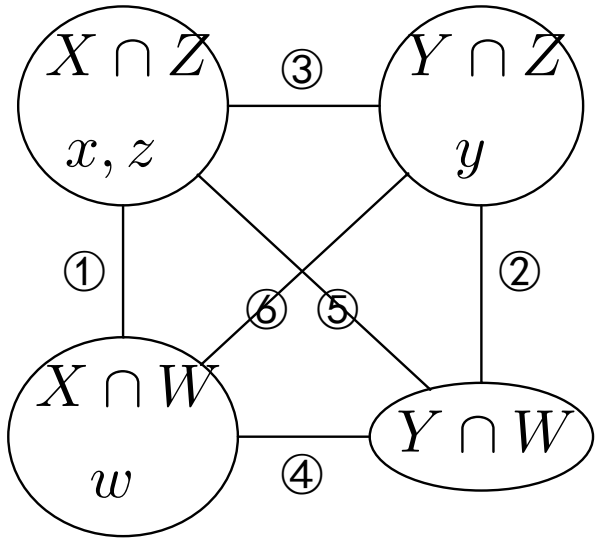


$$c(Z,W) + C(X,Y) \geq c(X \cap W, Y \cup Z) + c(X \cup W, Y \cap Z)$$

- 观察可知$(X \cap W, Y \cup Z)$也是z, w的一个割，同时由于(Z,W)是z,w的最小割，所以

$$c(X \cap W, Y \cup Z) \geq c(Z,W)$$

- 情况3：$z \in X,\ w \in X,\ x \in Z,\ y \in Z$

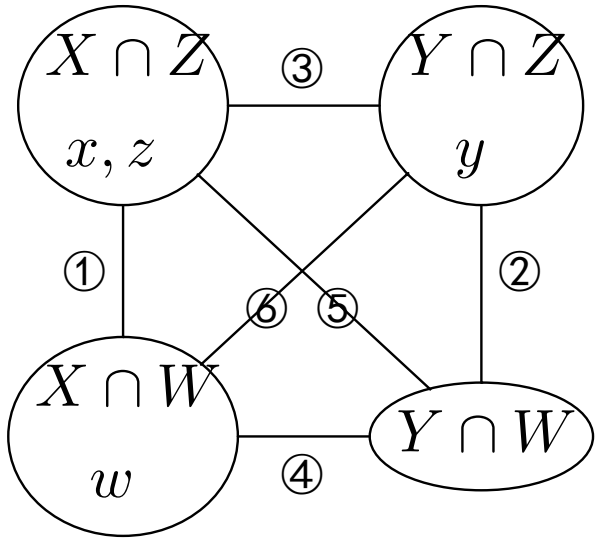


$$c(Z,W) + C(X,Y) \geq c(X \cap W, Y \cup Z) + c(X \cup W, Y \cap Z)$$

- 同理，观察可知$(X \cup W, Y \cap Z)$也是x, y的一个割，同时由于(X,Y)是x, y的最小割，所以

$$c(X \cup W, Y \cap Z) \geq c(X,Y)$$

- 情况3：$z \in X,\ w \in X,\ x \in Z,\ y \in Z$



$$c(Z,W) + C(X,Y) \geq c(X \cap W, Y \cup Z) + c(X \cup W, Y \cap Z)$$

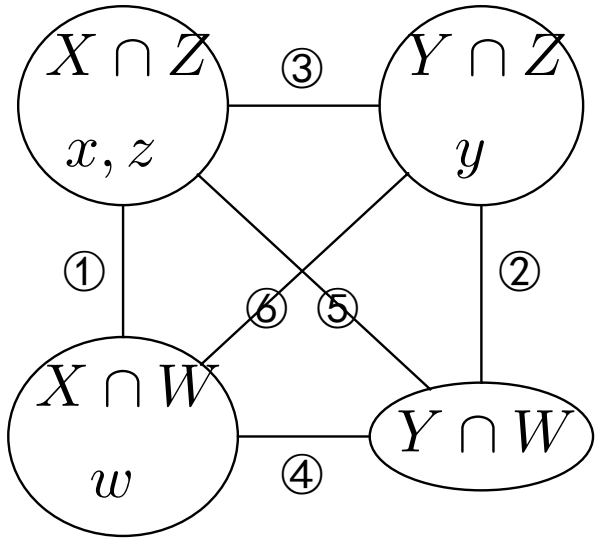
- 因此

$$c(Z,W) + C(X,Y) = c(X \cap W, Y \cup Z) + c(X \cup W, Y \cap Z)$$

并且

$$c(Z,W) = c(X \cap W, Y \cup Z),\ C(X,Y) = c(X \cup W, Y \cap Z)$$

- 情况3：$z \in X,\ w \in X,\ x \in Z,\ y \in Z$



$$c(Z,W) + C(X,Y) \geq c(X \cap W, Y \cup Z) + c(X \cup W, Y \cap Z)$$

- 于是，可以把$\alpha(Z,W)$调整为$(X \cap W, Y \cup Z)$，这样就不相交了。

- 有了这些性质，我们就可以定义一种树叫等价流树。

- 有了这些性质，我们就可以定义一种树叫等价流树。
- 等价流树就是这样一棵树，使得图中两个点的最小割的权就是树上对应链上的最小边权。

- 有了这些性质，我们就可以定义一种树叫等价流树。
- 等价流树就是这样一棵树，使得图中两个点的最小割的权就是树上对应链上的最小边权。
- 如何构造等价流树？

- 有了这些性质，我们就可以定义一种树叫等价流树。
- 等价流树就是这样一棵树，使得图中两个点的最小割的权就是树上对应链上的最小边权。
- 如何构造等价流树？
- Gusfield算法！

- 有了这些性质，我们就可以定义一种树叫等价流树。
- 等价流树就是这样一棵树，使得图中两个点的最小割的权就是树上对应链上的最小边权。
- 如何构造等价流树？
- Gusfield算法！
- 对点集分治：令当前处理的点集为W。

- 有了这些性质，我们就可以定义一种树叫等价流树。
- 等价流树就是这样一棵树，使得图中两个点的最小割的权就是树上对应链上的最小边权。
- 如何构造等价流树？
- Gusfield算法！
- 对点集分治：令当前处理的点集为W。
- 随便选W中不同的两点s, t。

- 有了这些性质，我们就可以定义一种树叫等价流树。
- 等价流树就是这样一棵树，使得图中两个点的最小割的权就是树上对应链上的最小边权。
- 如何构造等价流树？
- Gusfield算法！
- 对点集分治：令当前处理的点集为W。
- 随便选W中不同的两点s, t。
- 在整张图上求出任意一个$\alpha(s, t)$。

- 有了这些性质，我们就可以定义一种树叫等价流树。
- 等价流树就是这样一棵树，使得图中两个点的最小割的权就是树上对应链上的最小边权。
- 如何构造等价流树？
- Gusfield算法！
- 对点集分治：令当前处理的点集为W。
- 随便选W中不同的两点s,t。
- 在整张图上求出任意一个$\alpha(s,t)$。
- 在s,t之间直接连边，边权为$\lambda(s,t)$。

- 有了这些性质，我们就可以定义一种树叫等价流树。
- 等价流树就是这样一棵树，使得图中两个点的最小割的权就是树上对应链上的最小边权。
- 如何构造等价流树？
- Gusfield算法！
- 对点集分治：令当前处理的点集为W。
- 随便选W中不同的两点s,t。
- 在整张图上求出任意一个$\alpha(s,t)$。
- 在s,t之间直接连边，边权为$\lambda(s,t)$。
- 设$\alpha(s,t)$把W分成两部分(S,T)。

- 有了这些性质，我们就可以定义一种树叫等价流树。
- 等价流树就是这样一棵树，使得图中两个点的最小割的权就是树上对应链上的最小边权。
- 如何构造等价流树？
- Gusfield算法！
- 对点集分治：令当前处理的点集为W。
- 随便选W中不同的两点s,t。
- 在整张图上求出任意一个$\alpha(s,t)$。
- 在s,t之间直接连边，边权为$\lambda(s,t)$。
- 设$\alpha(s,t)$把W分成两部分(S,T)。
- S中的两点之间的最小割可以调整为不与$\alpha(s,t)$相交，T也是如此。

- 有了这些性质，我们就可以定义一种树叫等价流树。
- 等价流树就是这样一棵树，使得图中两个点的最小割的权就是树上对应链上的最小边权。
- 如何构造等价流树？
- Gusfield算法！
- 对点集分治：令当前处理的点集为W。
- 随便选W中不同的两点s,t。
- 在整张图上求出任意一个$\alpha(s,t)$。
- 在s,t之间直接连边，边权为$\lambda(s,t)$。
- 设$\alpha(s,t)$把W分成两部分(S,T)。
- S中的两点之间的最小割可以调整为不与$\alpha(s,t)$相交，T也是如此。
- (S,T)两部分互不影响。

- 有了这些性质，我们就可以定义一种树叫等价流树。
- 等价流树就是这样一棵树，使得图中两个点的最小割的权就是树上对应链上的最小边权。
- 如何构造等价流树？
- Gusfield算法！
- 对点集分治：令当前处理的点集为W。
- 随便选W中不同的两点s,t。
- 在整张图上求出任意一个$\alpha(s,t)$。
- 在s,t之间直接连边，边权为$\lambda(s,t)$。
- 设$\alpha(s,t)$把W分成两部分(S,T)。
- S中的两点之间的最小割可以调整为不与$\alpha(s,t)$相交，T也是如此。
- (S,T)两部分互不影响。
- 递归处理两部分，直到点集中只剩一个点为止。

- 我们可以来证明一下这样做的正确性。

- 我们可以来证明一下这样做的正确性。
- 首先，递归的两部分构出来的都是合法的等价流树。

- 我们可以来证明一下这样做的正确性。
- 首先，递归的两部分构出来的都是合法的等价流树。
- 那么只要证中间连了一条s,t的边后整棵树是等价流树。

- 我们可以来证明一下这样做的正确性。
- 首先，递归的两部分构出来的都是合法的等价流树。
- 那么只要证中间连了一条s,t的边后整棵树是等价流树。
- 也就是要证明所有从S中某个点u到T中某个点v的路径都满足等价流树的性质，即

$$\lambda(u,v) = \min\left(\lambda(u,s), \lambda(s,t), \lambda(t,v)\right)$$

- 我们可以来证明一下这样做的正确性。
- 首先，递归的两部分构出来的都是合法的等价流树。
- 那么只要证中间连了一条s,t的边后整棵树是等价流树。
- 也就是要证明所有从S中某个点u到T中某个点v的路径都满足等价流树的性质，即

$$\lambda(u,v) = \min\left(\lambda(u,s), \lambda(s,t), \lambda(t,v)\right)$$

- 由我们开头证明的那个性质可知，上式左边部分大于等于右边部分，只要证左边部分小于等于右边部分即可。

- $$\lambda(u,v) = \min\left(\lambda(u,s), \lambda(s,t), \lambda(t,v)\right)$$

- $$\lambda(u,v) = \min\left(\lambda(u,s), \lambda(s,t), \lambda(t,v)\right)$$

- 显然，s,t的最小割也是u,v的割，因此$\lambda(u,v) \leq \lambda(s,t)$。

- $$\lambda(u,v) = \min\left(\lambda(u,s), \lambda(s,t), \lambda(t,v)\right)$$

- 显然，s,t的最小割也是u,v的割，因此$\lambda(u,v) \leq \lambda(s,t)$。
- 然后考虑$\alpha(u,s)$。

- $$\lambda(u,v) = \min\left(\lambda(u,s), \lambda(s,t), \lambda(t,v)\right)$$

- 显然，s,t的最小割也是u,v的割，因此$\lambda(u,v) \leq \lambda(s,t)$。
- 然后考虑$\alpha(u,s)$。
- 由于两个最小割可以不相交，$\alpha(u,s)$可以不割开T，设这样的(U,S')。

- $$\lambda(u,v) = \min\left(\lambda(u,s), \lambda(s,t), \lambda(t,v)\right)$$

- 显然，s,t的最小割也是u,v的割，因此$\lambda(u,v) \leq \lambda(s,t)$。
- 然后考虑$\alpha(u,s)$。
- 由于两个最小割可以不相交，$\alpha(u,s)$可以不割开T，设这样的(U,S')。
- 因此$T \subseteq S'$或U。

- $$\lambda(u,v) = \min\left(\lambda(u,s), \lambda(s,t), \lambda(t,v)\right)$$

- 显然，s,t的最小割也是u,v的割，因此$\lambda(u,v) \leq \lambda(s,t)$。
- 然后考虑$\alpha(u,s)$。
- 由于两个最小割可以不相交，$\alpha(u,s)$可以不割开T，设这样的(U,S')。
- 因此$T \subseteq S'$或U。
- 如果$T \subseteq S'$，那么(U,S')也是(u,v)的割，满足上式。

- $$\lambda(u,v) = \min\left(\lambda(u,s), \lambda(s,t), \lambda(t,v)\right)$$

- 显然，s,t的最小割也是u,v的割，因此$\lambda(u,v) \leq \lambda(s,t)$。
- 然后考虑$\alpha(u,s)$。
- 由于两个最小割可以不相交，$\alpha(u,s)$可以不割开T，设这样的(U,S')。
- 因此$T \subseteq S'$或U。
- 如果$T \subseteq S'$，那么(U,S')也是(u,v)的割，满足上式。
- 如果$T \subseteq U$，那么(U,S')也是(s,t)的割，$c(U,S') \geq \lambda(s,t)$，同样满足上式。

- $$\lambda(u,v) = \min\left(\lambda(u,s), \lambda(s,t), \lambda(t,v)\right)$$

- 显然，s,t的最小割也是u,v的割，因此$\lambda(u,v) \leq \lambda(s,t)$。
- 然后考虑$\alpha(u,s)$。
- 由于两个最小割可以不相交，$\alpha(u,s)$可以不割开T，设这样的(U,S')。
- 因此$T \subseteq S'$或U。
- 如果$T \subseteq S'$，那么(U,S')也是(u,v)的割，满足上式。
- 如果$T \subseteq U$，那么(U,S')也是(s,t)的割，$c(U,S') \geq \lambda(s,t)$，同样满足上式。
- 同理，对于$\alpha(t,v)$也满足$\alpha(u,s)$的性质。

- $$\lambda(u,v) = \min\left(\lambda(u,s), \lambda(s,t), \lambda(t,v)\right)$$

- 显然，s,t的最小割也是u,v的割，因此$\lambda(u,v) \leq \lambda(s,t)$。
- 然后考虑$\alpha(u,s)$。
- 由于两个最小割可以不相交，$\alpha(u,s)$可以不割开T，设这样的(U,S')。
- 因此$T \subseteq S'$或U。
- 如果$T \subseteq S'$，那么(U,S')也是(u,v)的割，满足上式。
- 如果$T \subseteq U$，那么(U,S')也是(s,t)的割，$c(U,S') \geq \lambda(s,t)$，同样满足上式。
- 同理，对于$\alpha(t,v)$也满足$\alpha(u,s)$的性质。
- 这样就证明了上式成立。

- $$\lambda(u,v) = \min\left(\lambda(u,s), \lambda(s,t), \lambda(t,v)\right)$$

- 显然，s,t的最小割也是u,v的割，因此$\lambda(u,v) \leq \lambda(s,t)$。
- 然后考虑$\alpha(u,s)$。
- 由于两个最小割可以不相交，$\alpha(u,s)$可以不割开T，设这样的(U,S')。
- 因此$T \subseteq S'$或U。
- 如果$T \subseteq S'$，那么(U,S')也是(u,v)的割，满足上式。
- 如果$T \subseteq U$，那么(U,S')也是(s,t)的割，$c(U,S') \geq \lambda(s,t)$，同样满足上式。
- 同理，对于$\alpha(t,v)$也满足$\alpha(u,s)$的性质。
- 这样就证明了上式成立。
- 对于$u = s$或$v = t$的情况，同样也是这样证明。

- $$\lambda(u,v) = \min\left(\lambda(u,s), \lambda(s,t), \lambda(t,v)\right)$$

- 显然，s,t的最小割也是u,v的割，因此$\lambda(u,v) \leq \lambda(s,t)$。
- 然后考虑$\alpha(u,s)$。
- 由于两个最小割可以不相交，$\alpha(u,s)$可以不割开T，设这样的(U,S')。
- 因此$T \subseteq S'$或U。
- 如果$T \subseteq S'$，那么(U,S')也是(u,v)的割，满足上式。
- 如果$T \subseteq U$，那么(U,S')也是(s,t)的割，$c(U,S') \geq \lambda(s,t)$，同样满足上式。
- 同理，对于$\alpha(t,v)$也满足$\alpha(u,s)$的性质。
- 这样就证明了上式成立。
- 对于$u = s$或$v = t$的情况，同样也是这样证明。
- 所以这样构出的等价流树是正确的。

- Gusfield算法不仅可以递归实现，也有非常简单的非递归实现。

- Gusfield算法不仅可以递归实现，也有非常简单的非递归实现。
- 伪代码：

$fa_1 = NULL$
$fa_2, fa_3, \ldots, fa_{|V|} = 1$
for $u = 2$ **to** $|V|$ **do**
  $v = fa_u$
  求出$\alpha(v,u), \lambda(v,u)$
  $w_u = \lambda(v,u)$
  for $x = u+1$ **to** $|V|$ **do**
    if $fa_x = v$ **and** x被划到v侧 **then**
      $fa_x = u$
    end if
  end for
end for

- 相当于有很多点集，每次从中选一个点集拆分。

- 相当于有很多点集，每次从中选一个点集拆分。
- fa_u在u被枚举到之前表示u所属点集的代表点，枚举到之后表示u在等价流树上的父亲。

- 相当于有很多点集，每次从中选一个点集拆分。
- fa_u在u被枚举到之前表示u所属点集的代表点，枚举到之后表示u在等价流树上的父亲。
- 初始时所有点都属于以1为代表点的点集。

- 相当于有很多点集，每次从中选一个点集拆分。
- fa_u在u被枚举到之前表示u所属点集的代表点，枚举到之后表示u在等价流树上的父亲。
- 初始时所有点都属于以1为代表点的点集。
- 从2到$|V|$依次枚举u，把u所在点集以u和fa_u为源和汇拆分，所有被分到u侧的点x把fa_x修改为u。

- 等等，这个等价流树不就是最小割树吗？

- 等等，这个等价流树不就是最小割树吗？
- 其实最小割树(Gomory-Hu树)是在等价流树的基础上再加一个性质：

- 等等，这个等价流树不就是最小割树吗？
- 其实最小割树(Gomory-Hu树)是在等价流树的基础上再加一个性质：
- 对于树上每条边(s,t)，删去这条边后树的两部分就是一个s,t的最小割。

- 等等，这个等价流树不就是最小割树吗？
- 其实最小割树(Gomory-Hu树)是在等价流树的基础上再加一个性质：
- 对于树上每条边(s,t)，删去这条边后树的两部分就是一个s,t的最小割。
- 由于最小割树构造及证明较复杂，而且等价流树在OI中已经够用，这里就不讲最小割树了。

- 接下来我们再来看一个问题。

- 接下来我们再来看一个问题。
- 给定一张边权非负的无向图G。定义全局最小割为权值最小的划分。求全局最小割。

- 接下来我们再来看一个问题。
- 给定一张边权非负的无向图G。定义全局最小割为权值最小的划分。求全局最小割。
- 咦，这不是直接构出等价流树然后找个最小的就好了吗？

- 接下来我们再来看一个问题。
- 给定一张边权非负的无向图G。定义全局最小割为权值最小的划分。求全局最小割。
- 咦，这不是直接构出等价流树然后找个最小的就好了吗？
- 然而有复杂度更好的、不用网络流的做法。

- 接下来我们再来看一个问题。
- 给定一张边权非负的无向图G。定义全局最小割为权值最小的划分。求全局最小割。
- 咦，这不是直接构出等价流树然后找个最小的就好了吗？
- 然而有复杂度更好的、不用网络流的做法。
- 这就是Stoer-Wagner算法。

- 对于图中任意两点，它们可能在全局最小割的同侧，也可能在异侧。

- 对于图中任意两点，它们可能在全局最小割的同侧，也可能在异侧。
- 如果在同侧，显然可以把这两个点缩成一个点，不会影响答案；

- 对于图中任意两点，它们可能在全局最小割的同侧，也可能在异侧。
- 如果在同侧，显然可以把这两个点缩成一个点，不会影响答案；
- 如果在异侧，这两个点的最小割就是全局最小割。

- 对于图中任意两点，它们可能在全局最小割的同侧，也可能在异侧。
- 如果在同侧，显然可以把这两个点缩成一个点，不会影响答案；
- 如果在异侧，这两个点的最小割就是全局最小割。
- 所以我们就有了这样的想法：

- 对于图中任意两点，它们可能在全局最小割的同侧，也可能在异侧。
- 如果在同侧，显然可以把这两个点缩成一个点，不会影响答案；
- 如果在异侧，这两个点的最小割就是全局最小割。
- 所以我们就有了这样的想法：
- 不停地进行操作直到只剩一个点为止：

- 对于图中任意两点，它们可能在全局最小割的同侧，也可能在异侧。
- 如果在同侧，显然可以把这两个点缩成一个点，不会影响答案；
- 如果在异侧，这两个点的最小割就是全局最小割。
- 所以我们就有了这样的想法：
- 不停地进行操作直到只剩一个点为止：
- 任选两个点求最小割，更新答案；

- 对于图中任意两点，它们可能在全局最小割的同侧，也可能在异侧。
- 如果在同侧，显然可以把这两个点缩成一个点，不会影响答案；
- 如果在异侧，这两个点的最小割就是全局最小割。
- 所以我们就有了这样的想法：
- 不停地进行操作直到只剩一个点为止：
- 任选两个点求最小割，更新答案；
- 把这两个点缩起来。

- 对于图中任意两点，它们可能在全局最小割的同侧，也可能在异侧。
- 如果在同侧，显然可以把这两个点缩成一个点，不会影响答案；
- 如果在异侧，这两个点的最小割就是全局最小割。
- 所以我们就有了这样的想法：
- 不停地进行操作直到只剩一个点为止：
- 任选两个点求最小割，更新答案；
- 把这两个点缩起来。
- 如何找图中任意两个点的最小割呢？

- 有个方法叫最大邻接搜索。

- 有个方法叫最大邻接搜索。
- 有一个点集A。初始时A中包含某个初始的点。

- 有个方法叫最大邻接搜索。
- 有一个点集A。初始时A中包含某个初始的点。
- 每次，选择一个不在A中的点v使得$c(A, \{v\})$最大，将v加入到A 中。

- 有个方法叫最大邻接搜索。
- 有一个点集A。初始时A中包含某个初始的点。
- 每次，选择一个不在A中的点v使得$c(A, \{v\})$最大，将v加入到A 中。
- 直到所有点都加入到A中。

- 有个方法叫最大邻接搜索。
- 有一个点集A。初始时A中包含某个初始的点。
- 每次，选择一个不在A中的点v使得$c(A, \{v\})$最大，将v加入到A 中。
- 直到所有点都加入到A中。
- 设第i个加入A的点为v_i（初始的点是v_1）。

- 有个方法叫最大邻接搜索。
- 有一个点集A。初始时A中包含某个初始的点。
- 每次，选择一个不在A中的点v使得$c(A, \{v\})$最大，将v加入到A 中。
- 直到所有点都加入到A中。
- 设第i个加入A的点为v_i（初始的点是v_1）。
- 然后有性质：

- 有个方法叫最大邻接搜索。
- 有一个点集A。初始时A中包含某个初始的点。
- 每次，选择一个不在A中的点v使得$c(A, \{v\})$最大，将v加入到A 中。
- 直到所有点都加入到A中。
- 设第i个加入A的点为v_i（初始的点是v_1）。
- 然后有性质：
- $(\{v_1, v_2, \ldots, v_{n-1}\}, \{v_n\})$是最大邻接搜索中最后加入的两个点$v_{n-1}, v_n$的最小割。

- 有个方法叫最大邻接搜索。
- 有一个点集A。初始时A中包含某个初始的点。
- 每次，选择一个不在A中的点v使得$c(A, \{v\})$最大，将v加入到A 中。
- 直到所有点都加入到A中。
- 设第i个加入A的点为v_i（初始的点是v_1）。
- 然后有性质：
- $(\{v_1, v_2, \ldots, v_{n-1}\}, \{v_n\})$是最大邻接搜索中最后加入的两个点$v_{n-1}, v_n$的最小割。
- 我们用归纳法来证明一下：

- 有个方法叫最大邻接搜索。
- 有一个点集A。初始时A中包含某个初始的点。
- 每次，选择一个不在A中的点v使得$c(A, \{v\})$最大，将v加入到A 中。
- 直到所有点都加入到A中。
- 设第i个加入A的点为v_i（初始的点是v_1）。
- 然后有性质：
- $(\{v_1, v_2, \ldots, v_{n-1}\}, \{v_n\})$是最大邻接搜索中最后加入的两个点$v_{n-1}, v_n$的最小割。
- 我们用归纳法来证明一下：
- 如果$n = 2$，显然是对的。因此假定$n > 2$。

- 有个方法叫最大邻接搜索。
- 有一个点集A。初始时A中包含某个初始的点。
- 每次，选择一个不在A中的点v使得$c(A, \{v\})$最大，将v加入到A 中。
- 直到所有点都加入到A中。
- 设第i个加入A的点为v_i（初始的点是v_1）。
- 然后有性质：
- $(\{v_1, v_2, \ldots, v_{n-1}\}, \{v_n\})$是最大邻接搜索中最后加入的两个点$v_{n-1}, v_n$的最小割。
- 我们用归纳法来证明一下：
- 如果$n = 2$，显然是对的。因此假定$n > 2$。
- 考虑最后加入的三个点v_{n-2}, v_{n-1}, v_n。

- 有个方法叫最大邻接搜索。
- 有一个点集A。初始时A中包含某个初始的点。
- 每次，选择一个不在A中的点v使得$c(A, \{v\})$最大，将v加入到A 中。
- 直到所有点都加入到A中。
- 设第i个加入A的点为v_i（初始的点是v_1）。
- 然后有性质：
- $(\{v_1, v_2, \ldots, v_{n-1}\}, \{v_n\})$是最大邻接搜索中最后加入的两个点$v_{n-1}, v_n$的最小割。
- 我们用归纳法来证明一下：
- 如果$n = 2$，显然是对的。因此假定$n > 2$。
- 考虑最后加入的三个点v_{n-2}, v_{n-1}, v_n。
- 首先我们可以删去边(v_{n-1}, v_n)，因为任何v_{n-1}, v_n的割都要割这条边，删去了对这些割的相对大小没有影响。

- 现在我们来证明$c(\{v_1,\ldots,v_{n-1}\},\{v_n\}) \leq \lambda(v_{n-2},v_{n-1})$。

- 现在我们来证明$c(\{v_1,\ldots,v_{n-1}\},\{v_n\}) \leq \lambda(v_{n-2},v_{n-1})$。
- 考虑从图中移除点v_n（及其相连的边）。按原顺序重新进行最大邻接搜索，那么v_{n-2},v_{n-1}成为最后两点。

- 现在我们来证明$c(\{v_1,\ldots,v_{n-1}\},\{v_n\}) \leq \lambda(v_{n-2},v_{n-1})$。
- 考虑从图中移除点v_n（及其相连的边）。按原顺序重新进行最大邻接搜索，那么v_{n-2},v_{n-1}成为最后两点。
- 由数学归纳可知，$(\{v_1,\ldots,v_{n-2}\},\{v_{n-1}\})$ 是v_{n-2},v_{n-1}的最小割。

- 现在我们来证明$c(\{v_1,\ldots,v_{n-1}\},\{v_n\}) \leq \lambda(v_{n-2},v_{n-1})$。
- 考虑从图中移除点v_n（及其相连的边）。按原顺序重新进行最大邻接搜索，那么v_{n-2},v_{n-1}成为最后两点。
- 由数学归纳可知，$(\{v_1,\ldots,v_{n-2}\},\{v_{n-1}\})$ 是v_{n-2},v_{n-1}的最小割。
- 现在把v_n（及其相连的边）加回图中。

- 现在我们来证明$c(\{v_1,\ldots,v_{n-1}\},\{v_n\}) \leq \lambda(v_{n-2},v_{n-1})$。
- 考虑从图中移除点v_n（及其相连的边）。按原顺序重新进行最大邻接搜索，那么v_{n-2},v_{n-1}成为最后两点。
- 由数学归纳可知，$(\{v_1,\ldots,v_{n-2}\},\{v_{n-1}\})$ 是v_{n-2},v_{n-1}的最小割。
- 现在把v_n（及其相连的边）加回图中。
- 那么显然v_{n-2},v_{n-1}的最小割不可能变得更优。

- 现在我们来证明$c(\{v_1,\ldots,v_{n-1}\},\{v_n\}) \leq \lambda(v_{n-2},v_{n-1})$。
- 考虑从图中移除点v_n（及其相连的边）。按原顺序重新进行最大邻接搜索，那么v_{n-2},v_{n-1}成为最后两点。
- 由数学归纳可知，$(\{v_1,\ldots,v_{n-2}\},\{v_{n-1}\})$ 是v_{n-2},v_{n-1}的最小割。
- 现在把v_n（及其相连的边）加回图中。
- 那么显然v_{n-2},v_{n-1}的最小割不可能变得更优。
- 而由于v_n与v_{n-1}间没有边，所以我们把v_n加到原最小割的$\{v_1,\ldots,v_{n-2}\}$侧，最小割权不变。

- 现在我们来证明$c(\{v_1,\ldots,v_{n-1}\},\{v_n\}) \leq \lambda(v_{n-2},v_{n-1})$。
- 考虑从图中移除点v_n（及其相连的边）。按原顺序重新进行最大邻接搜索，那么v_{n-2},v_{n-1}成为最后两点。
- 由数学归纳可知，$(\{v_1,\ldots,v_{n-2}\},\{v_{n-1}\})$ 是v_{n-2},v_{n-1}的最小割。
- 现在把v_n（及其相连的边）加回图中。
- 那么显然v_{n-2},v_{n-1}的最小割不可能变得更优。
- 而由于v_n与v_{n-1}间没有边，所以我们把v_n加到原最小割的$\{v_1,\ldots,v_{n-2}\}$侧，最小割权不变。
- 因此$(\{v_1,\ldots,v_{n-2},v_n\},\{v_{n-1}\})$是$v_{n-2},v_{n-1}$的最小割。

- 又由最大邻接搜索倒数第二步的选择可知

$$c(\{v_1, \ldots, v_{n-2}\}, \{v_{n-1}\}) \geq c(\{v_1, \ldots, v_{n-2}\}, \{v_n\})$$

- 又由最大邻接搜索倒数第二步的选择可知

$$c(\{v_1, \ldots, v_{n-2}\}, \{v_{n-1}\}) \geq c(\{v_1, \ldots, v_{n-2}\}, \{v_n\})$$

- 由于v_{n-1}与v_n间没有边，

$$c(\{v_1, \ldots, v_{n-2}, v_n\}, \{v_{n-1}\}) \geq c(\{v_1, \ldots, v_{n-2}, v_{n-1}\}, \{v_n\})$$

- 又由最大邻接搜索倒数第二步的选择可知

$$c(\{v_1, \ldots, v_{n-2}\}, \{v_{n-1}\}) \geq c(\{v_1, \ldots, v_{n-2}\}, \{v_n\})$$

- 由于v_{n-1}与v_n间没有边，

$$c(\{v_1, \ldots, v_{n-2}, v_n\}, \{v_{n-1}\}) \geq c(\{v_1, \ldots, v_{n-2}, v_{n-1}\}, \{v_n\})$$

- 即

$$\lambda(v_{n-2}, v_{n-1}) \geq c(\{v_1, \ldots, v_{n-1}\}, \{v_n\})$$

- 又由最大邻接搜索倒数第二步的选择可知

$$c(\{v_1, \ldots, v_{n-2}\}, \{v_{n-1}\}) \geq c(\{v_1, \ldots, v_{n-2}\}, \{v_n\})$$

- 由于v_{n-1}与v_n间没有边，

$$c(\{v_1, \ldots, v_{n-2}, v_n\}, \{v_{n-1}\}) \geq c(\{v_1, \ldots, v_{n-2}, v_{n-1}\}, \{v_n\})$$

- 即

$$\lambda(v_{n-2}, v_{n-1}) \geq c(\{v_1, \ldots, v_{n-1}\}, \{v_n\})$$

- 接下来我们再用相似的方法来证明

$$c(\{v_1, \ldots, v_{n-1}\}, \{v_n\}) \leq \lambda(v_{n-2}, v_n)$$

- 考虑从图中移除点v_{n-1}（及其相连的边）。

- 考虑从图中移除点v_{n-1}（及其相连的边）。
- 按原顺序重新进行最大邻接搜索，那么v_{n-2}, v_n成为最后两点。

- 考虑从图中移除点v_{n-1}（及其相连的边）。
- 按原顺序重新进行最大邻接搜索，那么v_{n-2}, v_n成为最后两点。
- 由数学归纳可知，$(\{v_1, \ldots, v_{n-2}\}, \{v_n\})$是$v_{n-2}, v_n$的最小割。

- 考虑从图中移除点v_{n-1}（及其相连的边）。
- 按原顺序重新进行最大邻接搜索，那么v_{n-2}, v_n成为最后两点。
- 由数学归纳可知，$(\{v_1, \ldots, v_{n-2}\}, \{v_n\})$是$v_{n-2}, v_n$的最小割。
- 现在把v_{n-1}（及其相连的边）加回图中。

- 考虑从图中移除点v_{n-1}（及其相连的边）。
- 按原顺序重新进行最大邻接搜索，那么v_{n-2}, v_n成为最后两点。
- 由数学归纳可知，$(\{v_1, \ldots, v_{n-2}\}, \{v_n\})$是$v_{n-2}, v_n$的最小割。
- 现在把v_{n-1}（及其相连的边）加回图中。
- 那么显然v_{n-2}, v_n的最小割不可能变得更优。

- 考虑从图中移除点v_{n-1}（及其相连的边）。
- 按原顺序重新进行最大邻接搜索，那么v_{n-2}, v_n成为最后两点。
- 由数学归纳可知，$(\{v_1, \ldots, v_{n-2}\}, \{v_n\})$是$v_{n-2}, v_n$的最小割。
- 现在把v_{n-1}（及其相连的边）加回图中。
- 那么显然v_{n-2}, v_n的最小割不可能变得更优。
- 而由于v_{n-1}与v_n间没有边，所以我们把v_{n-1}加到原最小割的$\{v_1, \ldots, v_{n-2}\}$侧，最小割权不变。

- 考虑从图中移除点v_{n-1}（及其相连的边）。
- 按原顺序重新进行最大邻接搜索，那么v_{n-2}, v_n成为最后两点。
- 由数学归纳可知，$(\{v_1, \ldots, v_{n-2}\}, \{v_n\})$是$v_{n-2}, v_n$的最小割。
- 现在把v_{n-1}（及其相连的边）加回图中。
- 那么显然v_{n-2}, v_n的最小割不可能变得更优。
- 而由于v_{n-1}与v_n间没有边，所以我们把v_{n-1}加到原最小割的$\{v_1, \ldots, v_{n-2}\}$侧，最小割权不变。
- 因此$(\{v_1, \ldots, v_{n-2}, v_{n-1}\}, \{v_n\})$是$v_{n-2}, v_n$的最小割。即

$$c(\{v_1, \ldots, v_{n-1}\}, \{v_n\}) = \lambda(v_{n-2}, v_n)$$

- 考虑从图中移除点v_{n-1}（及其相连的边）。
- 按原顺序重新进行最大邻接搜索，那么v_{n-2}, v_n成为最后两点。
- 由数学归纳可知，$(\{v_1, \ldots, v_{n-2}\}, \{v_n\})$是$v_{n-2}, v_n$的最小割。
- 现在把v_{n-1}（及其相连的边）加回图中。
- 那么显然v_{n-2}, v_n的最小割不可能变得更优。
- 而由于v_{n-1}与v_n间没有边，所以我们把v_{n-1}加到原最小割的$\{v_1, \ldots, v_{n-2}\}$侧，最小割权不变。
- 因此$(\{v_1, \ldots, v_{n-2}, v_{n-1}\}, \{v_n\})$是$v_{n-2}, v_n$的最小割。即

$$c(\{v_1, \ldots, v_{n-1}\}, \{v_n\}) = \lambda(v_{n-2}, v_n)$$

- 也就证明了$c(\{v_1, \ldots, v_{n-1}\}, \{v_n\}) \leq \lambda(v_{n-2}, v_n)$。

- 综合之前所说的两部分，可得

$$\lambda(v_{n-1}, v_n) \leq c(\{v_1, \ldots, v_{n-1}\}, \{v_n\}) \leq \min\left(\lambda(v_{n-1}, v_n), \lambda(v_{n-2}, v\right.$$

- 综合之前所说的两部分，可得

$$\lambda(v_{n-1}, v_n) \leq c(\{v_1, \ldots, v_{n-1}\}, \{v_n\}) \leq \min\left(\lambda(v_{n-1}, v_n), \lambda(v_{n-2}, v\right.$$

- 又根据开头证明的性质，

$$\lambda(v_{n-1}, v_n) \geq \min\left(\lambda(v_{n-1}, v_n), \lambda(v_{n-2}, v_n)\right)$$

- 综合之前所说的两部分，可得

$$\lambda(v_{n-1}, v_n) \leq c(\{v_1, \ldots, v_{n-1}\}, \{v_n\}) \leq \min\left(\lambda(v_{n-1}, v_n), \lambda(v_{n-2}, v\right.$$

- 又根据开头证明的性质，

$$\lambda(v_{n-1}, v_n) \geq \min\left(\lambda(v_{n-1}, v_n), \lambda(v_{n-2}, v_n)\right)$$

- 因此，上述不等式中所有不等号均取等号。也就是说，

$$\lambda(v_{n-1}, v_n) = c(\{v_1, \ldots, v_{n-1}\}, \{v_n\})$$

- 我们来看看怎么实现。

- 我们来看看怎么实现。
- 对于缩点，我们可以用链表记边，缩两个点时把边表接起来就好了。

- 我们来看看怎么实现。
- 对于缩点，我们可以用链表记边，缩两个点时把边表接起来就好了。
- 枚举边的时候只要用并查集询问一下就好了。

- 我们来看看怎么实现。
- 对于缩点，我们可以用链表记边，缩两个点时把边表接起来就好了。
- 枚举边的时候只要用并查集询问一下就好了。
- 这些都不是复杂度瓶颈。

- 我们来看看怎么实现。
- 对于缩点，我们可以用链表记边，缩两个点时把边表接起来就好了。
- 枚举边的时候只要用并查集询问一下就好了。
- 这些都不是复杂度瓶颈。
- 对于更新一个值和查找最大值的操作，可以用堆来维护。

- 我们来看看怎么实现。
- 对于缩点，我们可以用链表记边，缩两个点时把边表接起来就好了。
- 枚举边的时候只要用并查集询问一下就好了。
- 这些都不是复杂度瓶颈。
- 对于更新一个值和查找最大值的操作，可以用堆来维护。
- 如果用二叉堆，一遍最大邻接搜索的复杂度为$O((|V|+|E|)\log|V|)$。

- 我们来看看怎么实现。
- 对于缩点，我们可以用链表记边，缩两个点时把边表接起来就好了。
- 枚举边的时候只要用并查集询问一下就好了。
- 这些都不是复杂度瓶颈。
- 对于更新一个值和查找最大值的操作，可以用堆来维护。
- 如果用二叉堆，一遍最大邻接搜索的复杂度为$O((|V|+|E|)\log|V|)$。
- 如果用斐波那契堆，一遍最大邻接搜索的复杂度为$O(|E|+|V|\log|V|)$。

- 我们来看看怎么实现。
- 对于缩点，我们可以用链表记边，缩两个点时把边表接起来就好了。
- 枚举边的时候只要用并查集询问一下就好了。
- 这些都不是复杂度瓶颈。
- 对于更新一个值和查找最大值的操作，可以用堆来维护。
- 如果用二叉堆，一遍最大邻接搜索的复杂度为$O((|V|+|E|)\log|V|)$。
- 如果用斐波那契堆，一遍最大邻接搜索的复杂度为$O(|E|+|V|\log|V|)$。
- 一共要做$|V|$遍，所以总复杂度为$O(|V||E|+|V|^2\log|V|)$。

- 我们来看看怎么实现。
- 对于缩点，我们可以用链表记边，缩两个点时把边表接起来就好了。
- 枚举边的时候只要用并查集询问一下就好了。
- 这些都不是复杂度瓶颈。
- 对于更新一个值和查找最大值的操作，可以用堆来维护。
- 如果用二叉堆，一遍最大邻接搜索的复杂度为$O((|V|+|E|)\log|V|)$。
- 如果用斐波那契堆，一遍最大邻接搜索的复杂度为$O(|E|+|V|\log|V|)$。
- 一共要做$|V|$遍，所以总复杂度为$O(|V||E|+|V|^2\log|V|)$。
- 当然如果$|E|=O(|V|^2)$的话，直接全部暴力就好了，总复杂度就是$O(|V|^3)$。

- 可是讲了这么多，这些算法有什么用呢？

- 可是讲了这么多，这些算法有什么用呢？
- 我们来看几道题。

- 可是讲了这么多，这些算法有什么用呢？
- 我们来看几道题。

Codeforces Round #200 Div.1 E. Pumping Stations

给定一张边权非负的无向图。求一个所有点的排列，使得排列中相邻两个点之间的最大流之和最大。

点数$n \leq 200$，边数$m \leq 1000$。

- 可是讲了这么多，这些算法有什么用呢？
- 我们来看几道题。

Codeforces Round #200 Div.1 E. Pumping Stations

给定一张边权非负的无向图。求一个所有点的排列，使得排列中相邻两个点之间的最大流之和最大。
点数$n \leq 200$，边数$m \leq 1000$。

- 首先我们可以构出等价流树。

- 可是讲了这么多，这些算法有什么用呢？
- 我们来看几道题。

Codeforces Round #200 Div.1 E. Pumping Stations

给定一张边权非负的无向图。求一个所有点的排列，使得排列中相邻两个点之间的最大流之和最大。
点数$n \leq 200$，边数$m \leq 1000$。

- 首先我们可以构出等价流树。
- 然后就有神奇的结论：最优排列的答案等于等价流树上所有边权之和。

- 那这个结论是怎么来的呢？如何构造一个排列呢？

- 那这个结论是怎么来的呢？如何构造一个排列呢？
- 首先，一个排列在树上对应一条路径。

- 那这个结论是怎么来的呢？如何构造一个排列呢？
- 首先，一个排列在树上对应一条路径。
- 考虑树上边权最小的那条边（如果有多条选任意一条）。

- 那这个结论是怎么来的呢？如何构造一个排列呢？
- 首先，一个排列在树上对应一条路径。
- 考虑树上边权最小的那条边（如果有多条选任意一条）。
- 那么从这条边一侧的某个点走到另一侧的某个点的权值必然就是这条边的权值。

- 那这个结论是怎么来的呢？如何构造一个排列呢？
- 首先，一个排列在树上对应一条路径。
- 考虑树上边权最小的那条边（如果有多条选任意一条）。
- 那么从这条边一侧的某个点走到另一侧的某个点的权值必然就是这条边的权值。
- 如果一条路径不止一次跨过这条边，那么必然可以通过调整使得答案不变劣：

- 那这个结论是怎么来的呢？如何构造一个排列呢？
- 首先，一个排列在树上对应一条路径。
- 考虑树上边权最小的那条边（如果有多条选任意一条）。
- 那么从这条边一侧的某个点走到另一侧的某个点的权值必然就是这条边的权值。
- 如果一条路径不止一次跨过这条边，那么必然可以通过调整使得答案不变劣：
- 在整条路径上跨过这条边的地方断开，我们会得到若干段不跨过的路径。

- 那这个结论是怎么来的呢？如何构造一个排列呢？
- 首先，一个排列在树上对应一条路径。
- 考虑树上边权最小的那条边（如果有多条选任意一条）。
- 那么从这条边一侧的某个点走到另一侧的某个点的权值必然就是这条边的权值。
- 如果一条路径不止一次跨过这条边，那么必然可以通过调整使得答案不变劣：
- 在整条路径上跨过这条边的地方断开，我们会得到若干段不跨过的路径。
- 不妨设跨过了k次，那么就会得到$k+1$段路径。

- 我们可以把同一侧的路径都首尾相接拼起来，这样就在两侧各得到了一条不跨过中间这条边的路径。

- 我们可以把同一侧的路径都首尾相接拼起来，这样就在两侧各得到了一条不跨过中间这条边的路径。
- 然后再把两侧的路径首尾相接，这就必然会跨过中间的边。

- 我们可以把同一侧的路径都首尾相接拼起来，这样就在两侧各得到了一条不跨过中间这条边的路径。
- 然后再把两侧的路径首尾相接，这就必然会跨过中间的边。
- 经过调整后，我们就只跨过中间边一次了。

- 我们可以把同一侧的路径都首尾相接拼起来，这样就在两侧各得到了一条不跨过中间这条边的路径。
- 然后再把两侧的路径首尾相接，这就必然会跨过中间的边。
- 经过调整后，我们就只跨过中间边一次了。
- 而由于中间这条边是最小的，不跨过这条边一定不比跨过劣。

- 我们可以把同一侧的路径都首尾相接拼起来，这样就在两侧各得到了一条不跨过中间这条边的路径。
- 然后再把两侧的路径首尾相接，这就必然会跨过中间的边。
- 经过调整后，我们就只跨过中间边一次了。
- 而由于中间这条边是最小的，不跨过这条边一定不比跨过劣。
- 因此，这样调整过后必然不会变劣。

- 我们可以把同一侧的路径都首尾相接拼起来，这样就在两侧各得到了一条不跨过中间这条边的路径。
- 然后再把两侧的路径首尾相接，这就必然会跨过中间的边。
- 经过调整后，我们就只跨过中间边一次了。
- 而由于中间这条边是最小的，不跨过这条边一定不比跨过劣。
- 因此，这样调整过后必然不会变劣。
- 然后，我们对这条边的左右两边递归进行这个推导。

- 我们可以把同一侧的路径都首尾相接拼起来，这样就在两侧各得到了一条不跨过中间这条边的路径。
- 然后再把两侧的路径首尾相接，这就必然会跨过中间的边。
- 经过调整后，我们就只跨过中间边一次了。
- 而由于中间这条边是最小的，不跨过这条边一定不比跨过劣。
- 因此，这样调整过后必然不会变劣。
- 然后，我们对这条边的左右两边递归进行这个推导。
- 这样就证明了结论，递归的同时也构造出了一个最优排列。

UVALive 5099 Nubulsa Expo

给定一张无向图，对于某个确定的源，求出与所有可能的汇的最大流的最小值。

点数$n \leq 300$，边数$m \leq 50000$。

UVALive 5099 Nubulsa Expo

给定一张无向图，对于某个确定的源，求出与所有可能的汇的最大流的最小值。
点数$n \leq 300$，边数$m \leq 50000$。

- 显然，答案就是全局最小割。

UVALive 5099 Nubulsa Expo

给定一张无向图，对于某个确定的源，求出与所有可能的汇的最大流的最小值。
点数$n \leq 300$，边数$m \leq 50000$。

- 显然，答案就是全局最小割。
- 因为源必然在全局最小割的某侧。

UVALive 5099 Nubulsa Expo

给定一张无向图，对于某个确定的源，求出与所有可能的汇的最大流的最小值。
点数$n \leq 300$，边数$m \leq 50000$。

- 显然，答案就是全局最小割。
- 因为源必然在全局最小割的某侧。
- 只要选另一侧中某个点作为汇，就可以把全局最小割作为源汇的最小割了。

非负边权平面图最小环

给定一张平面图，平面图中每条边都有一个非负的权。求权最小的环。

非负边权平面图最小环

给定一张平面图，平面图中每条边都有一个非负的权。求权最小的环。

- 构出平面图的对偶图。

非负边权平面图最小环

给定一张平面图，平面图中每条边都有一个非负的权。求权最小的环。

- 构出平面图的对偶图。
- 那么平面图的一个环就对应对偶图的一个割，环的权就是割的权。

非负边权平面图最小环

给定一张平面图，平面图中每条边都有一个非负的权。求权最小的环。

- 构出平面图的对偶图。
- 那么平面图的一个环就对应对偶图的一个割，环的权就是割的权。
- 因此，只要求出对偶图的全局最小割即可。

非负边权平面图最小环

给定一张平面图，平面图中每条边都有一个非负的权。求权最小的环。

- 构出平面图的对偶图。
- 那么平面图的一个环就对应对偶图的一个割，环的权就是割的权。
- 因此，只要求出对偶图的全局最小割即可。
- 可能存在复杂度更优的利用平面图性质的算法。

51Nod 算法马拉松12 有趣的游戏 by SkyDec

51Nod 算法马拉松12 有趣的游戏 by SkyDec

有n个人在玩游戏，游戏的规则是这样的：

51Nod 算法马拉松12 有趣的游戏 by SkyDec

有n个人在玩游戏，游戏的规则是这样的：

(1) 随机选择一个还未出局的人x；

51Nod 算法马拉松12 有趣的游戏 by SkyDec

有n个人在玩游戏，游戏的规则是这样的：

(1) 随机选择一个还未出局的人x；

(2) x对剩下的未出局的人都做一次攻击，一个人受到攻击后，有$(1-p)$的概率受伤且出局；

51Nod 算法马拉松12 有趣的游戏 by SkyDec

有n个人在玩游戏，游戏的规则是这样的：

(1) 随机选择一个还未出局的人x；

(2) x对剩下的未出局的人都做一次攻击，一个人受到攻击后，有$(1-p)$的概率受伤且出局；

(3) x出局。

51Nod 算法马拉松12 有趣的游戏 by SkyDec

有n个人在玩游戏，游戏的规则是这样的：

(1) 随机选择一个还未出局的人x；

(2) x对剩下的未出局的人都做一次攻击，一个人受到攻击后，有$(1-p)$的概率受伤且出局；

(3) x出局。

重复以上步骤直到所有人出局。

51Nod 算法马拉松12 有趣的游戏 by SkyDec

有n个人在玩游戏，游戏的规则是这样的：

(1) 随机选择一个还未出局的人x；

(2) x对剩下的未出局的人都做一次攻击，一个人受到攻击后，有$(1-p)$的概率受伤且出局；

(3) x出局。

重复以上步骤直到所有人出局。

现在ZYB想知道，一个人不受伤出局，且一共受到k次攻击的概率是多少。

51Nod 算法马拉松12 有趣的游戏 by SkyDec

有n个人在玩游戏，游戏的规则是这样的：

(1) 随机选择一个还未出局的人x；

(2) x对剩下的未出局的人都做一次攻击，一个人受到攻击后，有$(1-p)$的概率受伤且出局；

(3) x出局。

重复以上步骤直到所有人出局。

现在ZYB想知道，一个人不受伤出局，且一共受到k次攻击的概率是多少。

为了避免精度误差，你需要输出答案在模10^9+7域下的值（分数用逆元表示）。

51Nod 算法马拉松12 有趣的游戏 by SkyDec

有n个人在玩游戏，游戏的规则是这样的：

(1) 随机选择一个还未出局的人x；

(2) x对剩下的未出局的人都做一次攻击，一个人受到攻击后，有$(1-p)$的概率受伤且出局；

(3) x出局。

重复以上步骤直到所有人出局。

现在ZYB想知道，一个人不受伤出局，且一共受到k次攻击的概率是多少。

为了避免精度误差，你需要输出答案在模10^9+7域下的值（分数用逆元表示）。

$p \notin \{0,1\}$，保证$p^0, p^1, \ldots, p^n$在模10^9+7域下互不相同。

51Nod 算法马拉松12 有趣的游戏 by SkyDec

有n个人在玩游戏，游戏的规则是这样的：

(1) 随机选择一个还未出局的人x；

(2) x对剩下的未出局的人都做一次攻击，一个人受到攻击后，有$(1-p)$的概率受伤且出局；

(3) x出局。

重复以上步骤直到所有人出局。

现在ZYB想知道，一个人不受伤出局，且一共受到k次攻击的概率是多少。

为了避免精度误差，你需要输出答案在模10^9+7域下的值（分数用逆元表示）。

$p \notin \{0,1\}$，保证$p^0, p^1, \ldots, p^n$在模10^9+7域下互不相同。

先给定n, p，然后再给出Q次询问，每次询问给出一个整数k。

51Nod 算法马拉松12 有趣的游戏 by SkyDec

有n个人在玩游戏，游戏的规则是这样的：

(1) 随机选择一个还未出局的人x；

(2) x对剩下的未出局的人都做一次攻击，一个人受到攻击后，有$(1-p)$的概率受伤且出局；

(3) x出局。

重复以上步骤直到所有人出局。

现在ZYB想知道，一个人不受伤出局，且一共受到k次攻击的概率是多少。

为了避免精度误差，你需要输出答案在模10^9+7域下的值（分数用逆元表示）。

$p \notin \{0,1\}$，保证$p^0, p^1, \ldots, p^n$在模10^9+7域下互不相同。

先给定n, p，然后再给出Q次询问，每次询问给出一个整数k。

$1 \le n, Q \le 10^6,\ 0 \le k < n,\ \sum k \le 10^6$

“保证$p^0, p^1, \ldots, p^n$在模$10^9 + 7$域下互不相同”这个条件有什么用呢？

“保证$p^0, p^1, \ldots, p^n$在模$10^9 + 7$域下互不相同”这个条件有什么用呢？别急，等下就知道了。

“保证$p^0, p^1, \ldots, p^n$在模$10^9 + 7$域下互不相同”这个条件有什么用呢？别急，等下就知道了。

首先，我们可以把游戏转化为：

“保证$p^0, p^1, \ldots, p^n$在模$10^9 + 7$域下互不相同”这个条件有什么用呢？别急，等下就知道了。
首先，我们可以把游戏转化为：

- 一个人受伤出局则标记为受伤，不让他出局；

“保证$p^0, p^1, \ldots, p^n$在模$10^9 + 7$域下互不相同”这个条件有什么用呢？别急，等下就知道了。
首先，我们可以把游戏转化为：

- 一个人受伤出局则标记为受伤，不让他出局；
- 每次从所有未出局的人中随机选择，若受伤则无视他并让他出局，否则执行第(2)步和第(3)步；

“保证$p^0, p^1, \ldots, p^n$在模$10^9 + 7$域下互不相同”这个条件有什么用呢？别急，等下就知道了。
首先，我们可以把游戏转化为：

- 一个人受伤出局则标记为受伤，不让他出局；
- 每次从所有未出局的人中随机选择，若受伤则无视他并让他出局，否则执行第(2)步和第(3)步；

这样做和原来的游戏是等价的。

“保证$p^0, p^1, \ldots, p^n$在模$10^9 + 7$域下互不相同”这个条件有什么用呢？别急，等下就知道了。
首先，我们可以把游戏转化为：

- 一个人受伤出局则标记为受伤，不让他出局；
- 每次从所有未出局的人中随机选择，若受伤则无视他并让他出局，否则执行第(2)步和第(3)步；

这样做和原来的游戏是等价的。和原题目相比，只是选的过程中夹杂了一些受伤的人。剩下的未受伤的人被选为下一个x的概率还是相等的。

“保证$p^0, p^1, \ldots, p^n$在模$10^9 + 7$域下互不相同”这个条件有什么用呢？别急，等下就知道了。
首先，我们可以把游戏转化为：

- 一个人受伤出局则标记为受伤，不让他出局；
- 每次从所有未出局的人中随机选择，若受伤则无视他并让他出局，否则执行第(2)步和第(3)步；

这样做和原来的游戏是等价的。和原题目相比，只是选的过程中夹杂了一些受伤的人。剩下的未受伤的人被选为下一个x的概率还是相等的。
这种选法会等概率地选出一个所有人的排列。

“保证$p^0, p^1, \ldots, p^n$在模10^9+7域下互不相同”这个条件有什么用呢？别急，等下就知道了。
首先，我们可以把游戏转化为：

- 一个人受伤出局则标记为受伤，不让他出局；
- 每次从所有未出局的人中随机选择，若受伤则无视他并让他出局，否则执行第(2)步和第(3)步；

这样做和原来的游戏是等价的。和原题目相比，只是选的过程中夹杂了一些受伤的人。剩下的未受伤的人被选为下一个x的概率还是相等的。
这种选法会等概率地选出一个所有人的排列。因而一个人在排列的每个位置的概率是相等的，即$\frac{1}{n}$。

于是我们有了一个DP：

于是我们有了一个DP：令$f[i][j]$表示选了i轮、i轮中有j个被选的人未受伤的概率。

于是我们有了一个DP：令$f[i][j]$表示选了i轮、i轮中有j个被选的人未受伤的概率。则有

$$f[i][j] = f[i-1][j-1] \times p^{j-1} + f[i-1][j] \times (1-p^j)$$

于是我们有了一个DP：令$f[i][j]$表示选了i轮、i轮中有j个被选的人未受伤的概率。则有

$$f[i][j] = f[i-1][j-1] \times p^{j-1} + f[i-1][j] \times (1-p^j)$$

这个DP方程的含义是，考虑第i次选的人是否在之前受伤，根据相应的概率转移。

于是我们有了一个DP：令$f[i][j]$表示选了i轮、i轮中有j个被选的人未受伤的概率。则有

$$f[i][j] = f[i-1][j-1] \times p^{j-1} + f[i-1][j] \times (1-p^j)$$

这个DP方程的含义是，考虑第i次选的人是否在之前受伤，根据相应的概率转移。
初值为$f[0][0] = 1$。

于是我们有了一个DP：令$f[i][j]$表示选了i轮、i轮中有j个被选的人未受伤的概率。则有

$$f[i][j] = f[i-1][j-1] \times p^{j-1} + f[i-1][j] \times (1-p^j)$$

这个DP方程的含义是，考虑第i次选的人是否在之前受伤，根据相应的概率转移。

初值为$f[0][0] = 1$。

答案就是

$$\frac{1}{n} \sum_{i=0}^{n-1} f[i][k] \times p^k$$

于是我们有了一个DP：令$f[i][j]$表示选了i轮、i轮中有j个被选的人未受伤的概率。则有

$$f[i][j] = f[i-1][j-1] \times p^{j-1} + f[i-1][j] \times (1-p^j)$$

这个DP方程的含义是，考虑第i次选的人是否在之前受伤，根据相应的概率转移。

初值为$f[0][0] = 1$。

答案就是

$$\frac{1}{n} \sum_{i=0}^{n-1} f[i][k] \times p^k$$

意思就是枚举这个人是第几个被选的，乘上他不受伤的概率。

于是我们有了一个DP：令$f[i][j]$表示选了i轮、i轮中有j个被选的人未受伤的概率。则有

$$f[i][j] = f[i-1][j-1] \times p^{j-1} + f[i-1][j] \times (1-p^j)$$

这个DP方程的含义是，考虑第i次选的人是否在之前受伤，根据相应的概率转移。
初值为$f[0][0] = 1$。
答案就是

$$\frac{1}{n} \sum_{i=0}^{n-1} f[i][k] \times p^k$$

意思就是枚举这个人是第几个被选的，乘上他不受伤的概率。
这个做法时间复杂度是$O(n^2)$。看起来已经相当厉害。

考虑优化这个DP。

考虑优化这个DP。

令$g[i][j] = f[i][j] \times p^j$。

考虑优化这个DP。

令$g[i][j] = f[i][j] \times p^j$。

则答案是

$$\frac{1}{n}\sum_{i=0}^{n-1} g[i][k]$$

考虑优化这个DP。
令$g[i][j] = f[i][j] \times p^j$。
则答案是

$$\frac{1}{n}\sum_{i=0}^{n-1} g[i][k]$$

DP方程就变成

$$g[i][j] = g[i-1][j-1] \times p^j + g[i-1][j] \times (1-p^j)$$

考虑优化这个DP。
令$g[i][j] = f[i][j] \times p^j$。
则答案是

$$\frac{1}{n}\sum_{i=0}^{n-1} g[i][k]$$

DP方程就变成

$$g[i][j] = g[i-1][j-1] \times p^j + g[i-1][j] \times (1-p^j)$$

注意到这些DP都与n无关，可以无限做下去。

考虑优化这个DP。
令$g[i][j] = f[i][j] \times p^j$。
则答案是

$$\frac{1}{n}\sum_{i=0}^{n-1} g[i][k]$$

DP方程就变成

$$g[i][j] = g[i-1][j-1] \times p^j + g[i-1][j] \times (1-p^j)$$

注意到这些DP都与n无关，可以无限做下去。
我们把$g[][k]$表示成幂级数。令

$$h_k(x) = \sum g[i][k]x^i$$

则由DP方程有

$$h_k(x) = p^k x \cdot h_{k-1}(x) + (1 - p^k)x \cdot h_k(x)$$

则由DP方程有

$$h_k(x) = p^k x \cdot h_{k-1}(x) + (1 - p^k)x \cdot h_k(x)$$

移项得

$$h_k(x) = \frac{p^k x}{1 - (1 - p^k)x} \cdot h_{k-1}(x)$$

则由DP方程有

$$h_k(x) = p^k x \cdot h_{k-1}(x) + (1 - p^k)x \cdot h_k(x)$$

移项得

$$h_k(x) = \frac{p^k x}{1 - (1 - p^k)x} \cdot h_{k-1}(x)$$

由DP初值可知$h_0(x) = 1$。

则由DP方程有

$$h_k(x) = p^k x \cdot h_{k-1}(x) + (1 - p^k)x \cdot h_k(x)$$

移项得

$$h_k(x) = \frac{p^k x}{1 - (1 - p^k)x} \cdot h_{k-1}(x)$$

由DP初值可知$h_0(x) = 1$。
所以

$$h_k(x) = \prod_{i=1}^{k} \frac{p^i x}{1 - (1 - p^i)x}$$

则由DP方程有

$$h_k(x) = p^k x \cdot h_{k-1}(x) + (1 - p^k)x \cdot h_k(x)$$

移项得

$$h_k(x) = \frac{p^k x}{1 - (1 - p^k)x} \cdot h_{k-1}(x)$$

由DP初值可知$h_0(x) = 1$。
所以

$$\begin{aligned} h_k(x) &= \prod_{i=1}^{k} \frac{p^i x}{1 - (1 - p^i)x} \\ &= p^{\frac{k(k+1)}{2}} x^k \cdot \prod_{i=1}^{k} \frac{1}{1 - (1 - p^i)x} \end{aligned}$$

接下来就是神来一笔了。

接下来就是神来一笔了。

令

$$\prod_{i=1}^{k} \frac{1}{1-(1-p^i)x} = \sum_{i=1}^{k} \frac{A_{k,i}}{1-(1-p^i)x}$$

接下来就是神来一笔了。
令

$$\prod_{i=1}^{k} \frac{1}{1-(1-p^i)x} = \sum_{i=1}^{k} \frac{A_{k,i}}{1-(1-p^i)x}$$

如何证明必然存在这样的A_k？又如何构造？

接下来就是神来一笔了。
令

$$\prod_{i=1}^{k} \frac{1}{1-(1-p^i)x} = \sum_{i=1}^{k} \frac{A_{k,i}}{1-(1-p^i)x}$$

如何证明必然存在这样的A_k？又如何构造？
如果我们构造出了这样一组A_k，也就证明了这样的方案存在。

接下来就是神来一笔了。
令

$$\prod_{i=1}^{k} \frac{1}{1-(1-p^i)x} = \sum_{i=1}^{k} \frac{A_{k,i}}{1-(1-p^i)x}$$

如何证明必然存在这样的A_k？又如何构造？
如果我们构造出了这样一组A_k，也就证明了这样的方案存在。
有一个非常厉害的构造：

接下来就是神来一笔了。
令

$$\prod_{i=1}^{k} \frac{1}{1-(1-p^i)x} = \sum_{i=1}^{k} \frac{A_{k,i}}{1-(1-p^i)x}$$

如何证明必然存在这样的A_k？又如何构造？
如果我们构造出了这样一组A_k，也就证明了这样的方案存在。
有一个非常厉害的构造：

$$A_{k,i} = \prod_{\substack{1 \le j \le k \\ j \ne i}} \frac{1}{1-\frac{1-p^j}{1-p^i}}$$

接下来就是神来一笔了。
令

$$\prod_{i=1}^{k} \frac{1}{1-(1-p^i)x} = \sum_{i=1}^{k} \frac{A_{k,i}}{1-(1-p^i)x}$$

如何证明必然存在这样的A_k？又如何构造？
如果我们构造出了这样一组A_k，也就证明了这样的方案存在。
有一个非常厉害的构造：

$$A_{k,i} = \prod_{\substack{1 \le j \le k \\ j \ne i}} \frac{1}{1-\frac{1-p^j}{1-p^i}}$$

我们来证明这个构造是正确的。

接下来就是神来一笔了。
令

$$\prod_{i=1}^{k} \frac{1}{1-(1-p^i)x} = \sum_{i=1}^{k} \frac{A_{k,i}}{1-(1-p^i)x}$$

如何证明必然存在这样的A_k？又如何构造？
如果我们构造出了这样一组A_k，也就证明了这样的方案存在。
有一个非常厉害的构造：

$$A_{k,i} = \prod_{\substack{1 \le j \le k \\ j \ne i}} \frac{1}{1-\frac{1-p^j}{1-p^i}}$$

我们来证明这个构造是正确的。
假设我们要证明对于任何$x \notin \{\frac{1}{1-p^i} \mid 1 \le i \le n\}$都正确。

对k进行数学归纳。

对k进行数学归纳。

$k = 1$时，$A_{k,1} = 1$，显然正确。

对k进行数学归纳。

$k = 1$时，$A_{k,1} = 1$，显然正确。

$k > 1$时，假设已经证明了$k-1$正确。只要证

$$\left(\sum_{i=1}^{k-1} \frac{A_{k-1,i}}{1-(1-p^i)x}\right) \cdot \frac{1}{1-(1-p^k)x} = \sum_{i=1}^{k} \frac{A_{k,i}}{1-(1-p^i)x}$$

对k进行数学归纳。

$k=1$时，$A_{k,1}=1$，显然正确。

$k>1$时，假设已经证明了$k-1$正确。只要证

$$\left(\sum_{i=1}^{k-1}\frac{A_{k-1,i}}{1-(1-p^i)x}\right)\cdot\frac{1}{1-(1-p^k)x}=\sum_{i=1}^{k}\frac{A_{k,i}}{1-(1-p^i)x}$$

考虑对于每个i $(1\le i\le k-1)$，只要构造两个数$B_{k,i},C_{k,i}$，使得

$$\frac{A_{k-1,i}}{1-(1-p^i)x}\cdot\frac{1}{1-(1-p^k)x}=\frac{B_{k,i}}{1-(1-p^i)x}+\frac{C_{k,i}}{1-(1-p^k)x}$$

对k进行数学归纳。
$k=1$时，$A_{k,1}=1$，显然正确。
$k>1$时，假设已经证明了$k-1$正确。只要证

$$\left(\sum_{i=1}^{k-1}\frac{A_{k-1,i}}{1-(1-p^i)x}\right)\cdot\frac{1}{1-(1-p^k)x}=\sum_{i=1}^{k}\frac{A_{k,i}}{1-(1-p^i)x}$$

考虑对于每个i $(1\le i\le k-1)$，只要构造两个数$B_{k,i},C_{k,i}$，使得

$$\frac{A_{k-1,i}}{1-(1-p^i)x}\cdot\frac{1}{1-(1-p^k)x}=\frac{B_{k,i}}{1-(1-p^i)x}+\frac{C_{k,i}}{1-(1-p^k)x}$$

并且

$$A_{k,i}=B_{k,i}\ (1\le i\le k-1),\ A_{k,k}=\sum_{i=1}^{k-1}C_{k,i}$$

$$\frac{B_{k,i}}{1-(1-p^i)x}+\frac{C_{k,i}}{1-(1-p^k)x}$$

$$
\begin{aligned}
& \frac{B_{k,i}}{1-(1-p^i)x} + \frac{C_{k,i}}{1-(1-p^k)x} \\
= \quad & \frac{B_{k,i}(1-(1-p^k)x) + C_{k,i}(1-(1-p^i)x)}{(1-(1-p^i)x)(1-(1-p^k)x)}
\end{aligned}
$$

$$
\begin{aligned}
& \frac{B_{k,i}}{1-(1-p^i)x} + \frac{C_{k,i}}{1-(1-p^k)x} \\
= \quad & \frac{B_{k,i}(1-(1-p^k)x) + C_{k,i}(1-(1-p^i)x)}{(1-(1-p^i)x)(1-(1-p^k)x)} \\
= \quad & \frac{B_{k,i} + C_{k,i} - ((1-p^k)B_{k,i} + (1-p^i)C_{k,i})x}{(1-(1-p^i)x)(1-(1-p^k)x)}
\end{aligned}
$$

$$
\begin{aligned}
& \frac{B_{k,i}}{1-(1-p^i)x} + \frac{C_{k,i}}{1-(1-p^k)x} \\
= \quad & \frac{B_{k,i}(1-(1-p^k)x) + C_{k,i}(1-(1-p^i)x)}{(1-(1-p^i)x)(1-(1-p^k)x)} \\
= \quad & \frac{B_{k,i} + C_{k,i} - ((1-p^k)B_{k,i} + (1-p^i)C_{k,i})x}{(1-(1-p^i)x)(1-(1-p^k)x)} \\
= \quad & \frac{A_{k-1,i}}{(1-(1-p^i)x)(1-(1-p^k)x)}
\end{aligned}
$$

$$
\begin{aligned}
& \frac{B_{k,i}}{1-(1-p^i)x} + \frac{C_{k,i}}{1-(1-p^k)x} \\
= & \frac{B_{k,i}(1-(1-p^k)x) + C_{k,i}(1-(1-p^i)x)}{(1-(1-p^i)x)(1-(1-p^k)x)} \\
= & \frac{B_{k,i} + C_{k,i} - ((1-p^k)B_{k,i} + (1-p^i)C_{k,i})x}{(1-(1-p^i)x)(1-(1-p^k)x)} \\
= & \frac{A_{k-1,i}}{(1-(1-p^i)x)(1-(1-p^k)x)}
\end{aligned}
$$

得到

$$
\begin{cases}
B_{k,i} + C_{k,i} = A_{k-1,i} \\
(1-p^k)B_{k,i} + (1-p^i)C_{k,i} = 0
\end{cases}
$$

由于p^k互不相同，$1-p^k \neq 1-p^i$，方程组有唯一解。

解得

$$\begin{cases} B_{k,i} = \frac{(1-p^i)A_{k-1,i}}{(1-p^i)-(1-p^k)} \\ C_{k,i} = \frac{(1-p^k)A_{k-1,i}}{(1-p^k)-(1-p^i)} \end{cases}$$

解得

$$\begin{cases} B_{k,i} = \frac{(1-p^i)A_{k-1,i}}{(1-p^i)-(1-p^k)} \\ C_{k,i} = \frac{(1-p^k)A_{k-1,i}}{(1-p^k)-(1-p^i)} \end{cases}$$

观察$B_{k,i}$，

解得

$$\begin{cases} B_{k,i} = \frac{(1-p^i)A_{k-1,i}}{(1-p^i)-(1-p^k)} \\ C_{k,i} = \frac{(1-p^k)A_{k-1,i}}{(1-p^k)-(1-p^i)} \end{cases}$$

观察$B_{k,i}$，

$$B_{k,i} = \frac{1-p^i}{(1-p^i)-(1-p^k)} \cdot A_{k-1,i}$$

解得

$$\begin{cases} B_{k,i} = \frac{(1-p^i)A_{k-1,i}}{(1-p^i)-(1-p^k)} \\ C_{k,i} = \frac{(1-p^k)A_{k-1,i}}{(1-p^k)-(1-p^i)} \end{cases}$$

观察$B_{k,i}$，

$$\begin{aligned} B_{k,i} &= \frac{1-p^i}{(1-p^i)-(1-p^k)} \cdot A_{k-1,i} \\ &= \frac{1}{1-\frac{1-p^k}{1-p^i}} \cdot A_{k-1,i} \end{aligned}$$

解得

$$\begin{cases} B_{k,i} = \frac{(1-p^i)A_{k-1,i}}{(1-p^i)-(1-p^k)} \\ C_{k,i} = \frac{(1-p^k)A_{k-1,i}}{(1-p^k)-(1-p^i)} \end{cases}$$

观察$B_{k,i}$，

$$\begin{aligned} B_{k,i} &= \frac{1-p^i}{(1-p^i)-(1-p^k)} \cdot A_{k-1,i} \\ &= \frac{1}{1-\frac{1-p^k}{1-p^i}} \cdot A_{k-1,i} \\ &= \frac{1}{1-\frac{1-p^k}{1-p^i}} \prod_{\substack{1 \le j \le k-1 \\ j \neq i}} \frac{1}{1-\frac{1-p^j}{1-p^i}} \end{aligned}$$

解得

$$\begin{cases} B_{k,i} = \frac{(1-p^i)A_{k-1,i}}{(1-p^i)-(1-p^k)} \\ C_{k,i} = \frac{(1-p^k)A_{k-1,i}}{(1-p^k)-(1-p^i)} \end{cases}$$

观察$B_{k,i}$，

$$\begin{aligned} B_{k,i} &= \frac{1-p^i}{(1-p^i)-(1-p^k)} \cdot A_{k-1,i} \\ &= \frac{1}{1-\frac{1-p^k}{1-p^i}} \cdot A_{k-1,i} \\ &= \frac{1}{1-\frac{1-p^k}{1-p^i}} \prod_{\substack{1\le j\le k-1 \\ j\neq i}} \frac{1}{1-\frac{1-p^j}{1-p^i}} \\ &= \prod_{\substack{1\le j\le k \\ j\neq i}} \frac{1}{1-\frac{1-p^j}{1-p^i}} = A_{k,i} \end{aligned}$$

再来看$C_{k,i}$，

再来看$C_{k,i}$，

$$\sum_{i=1}^{k-1} C_{k,i} = \sum_{i=1}^{k-1} \frac{(1-p^k)A_{k-1,i}}{(1-p^k)-(1-p^i)}$$

再来看$C_{k,i}$，

$$\begin{aligned}\sum_{i=1}^{k-1} C_{k,i} &= \sum_{i=1}^{k-1} \frac{(1-p^k)A_{k-1,i}}{(1-p^k)-(1-p^i)} \\ &= \sum_{i=1}^{k-1} \frac{A_{k-1,i}}{1-\frac{1-p^i}{1-p^k}}\end{aligned}$$

再来看$C_{k,i}$，

$$\begin{aligned}\sum_{i=1}^{k-1} C_{k,i} &= \sum_{i=1}^{k-1} \frac{(1-p^k)A_{k-1,i}}{(1-p^k)-(1-p^i)} \\ &= \sum_{i=1}^{k-1} \frac{A_{k-1,i}}{1-\frac{1-p^i}{1-p^k}}\end{aligned}$$

注意到这就是$k-1$的情况中取$x=\frac{1}{1-p^k}$时的式子。

再来看$C_{k,i}$，

$$\begin{aligned}\sum_{i=1}^{k-1} C_{k,i} &= \sum_{i=1}^{k-1} \frac{(1-p^k)A_{k-1,i}}{(1-p^k)-(1-p^i)} \\ &= \sum_{i=1}^{k-1} \frac{A_{k-1,i}}{1-\frac{1-p^i}{1-p^k}}\end{aligned}$$

注意到这就是$k-1$的情况中取$x=\frac{1}{1-p^k}$时的式子。（由于p^k互不相同，$x \notin \{\frac{1}{1-p^i} \mid 1 \le i \le k-1\}$。）

再来看$C_{k,i}$，

$$\begin{aligned}\sum_{i=1}^{k-1} C_{k,i} &= \sum_{i=1}^{k-1} \frac{(1-p^k)A_{k-1,i}}{(1-p^k)-(1-p^i)} \\ &= \sum_{i=1}^{k-1} \frac{A_{k-1,i}}{1-\frac{1-p^i}{1-p^k}}\end{aligned}$$

注意到这就是$k-1$的情况中取$x=\frac{1}{1-p^k}$时的式子。（由于p^k互不相同，$x \notin \{\frac{1}{1-p^i} \mid 1 \le i \le k-1\}$。）由归纳可知，这个式子等于

$$\prod_{i=1}^{k-1} \frac{1}{1-(1-p^i)x} = \prod_{\substack{1 \le i \le k \\ i \ne k}} \frac{1}{1-\frac{1-p^i}{1-p^k}} = A_{k,k}$$

至此，我们已经证好了A_k这样的构造是正确的。

至此，我们已经证好了A_k这样的构造是正确的。
考虑如何快速计算A_k。

至此，我们已经证好了A_k这样的构造是正确的。
考虑如何快速计算A_k。

$$A_{k,i} = \prod_{\substack{1\le j\le k\\ j\neq i}} \frac{1}{1-\frac{1-p^j}{1-p^i}} = \prod_{\substack{1\le j\le k\\ j\neq i}} \frac{1-p^i}{p^j-p^i} = \left(\frac{1-p^i}{p^i}\right)^{k-1} \cdot \prod_{\substack{1\le j\le k\\ j\neq i}} \frac{1}{p^{j-i}-1}$$

至此，我们已经证好了A_k这样的构造是正确的。
考虑如何快速计算A_k。

$$A_{k,i} = \prod_{\substack{1\le j\le k\\ j\neq i}} \frac{1}{1-\frac{1-p^j}{1-p^i}} = \prod_{\substack{1\le j\le k\\ j\neq i}} \frac{1-p^i}{p^j-p^i} = \left(\frac{1-p^i}{p^i}\right)^{k-1} \cdot \prod_{\substack{1\le j\le k\\ j\neq i}} \frac{1}{p^{j-i}-1}$$

$O(k)$预处理$\prod_{i=1}^{j} \frac{1}{p^i-1}$（包括$j<0$），就可以在$O(k\log k)$时间内求出所有$A_i$了。

至此，我们已经证好了A_k这样的构造是正确的。
考虑如何快速计算A_k。

$$A_{k,i} = \prod_{\substack{1\le j\le k\\ j\neq i}} \frac{1}{1-\frac{1-p^j}{1-p^i}} = \prod_{\substack{1\le j\le k\\ j\neq i}} \frac{1-p^i}{p^j-p^i} = \left(\frac{1-p^i}{p^i}\right)^{k-1} \cdot \prod_{\substack{1\le j\le k\\ j\neq i}} \frac{1}{p^{j-i}-1}$$

$O(k)$预处理$\prod_{i=1}^{j} \frac{1}{p^i-1}$（包括$j<0$），就可以在$O(k\log k)$时间内求出所有$A_i$了。
你问我怎么想到这样构造的？

至此，我们已经证好了A_k这样的构造是正确的。
考虑如何快速计算A_k。

$$A_{k,i} = \prod_{\substack{1\le j\le k\\ j\neq i}} \frac{1}{1-\frac{1-p^j}{1-p^i}} = \prod_{\substack{1\le j\le k\\ j\neq i}} \frac{1-p^i}{p^j-p^i} = \left(\frac{1-p^i}{p^i}\right)^{k-1} \cdot \prod_{\substack{1\le j\le k\\ j\neq i}} \frac{1}{p^{j-i}-1}$$

$O(k)$预处理$\prod_{i=1}^{j} \frac{1}{p^i-1}$（包括$j<0$），就可以在$O(k\log k)$时间内求出所有$A_i$了。
你问我怎么想到这样构造的？我只能说，无可奉告。

至此，我们已经证好了A_k这样的构造是正确的。
考虑如何快速计算A_k。

$$A_{k,i} = \prod_{\substack{1\le j\le k\\ j\neq i}} \frac{1}{1-\frac{1-p^j}{1-p^i}} = \prod_{\substack{1\le j\le k\\ j\neq i}} \frac{1-p^i}{p^j-p^i} = \left(\frac{1-p^i}{p^i}\right)^{k-1} \cdot \prod_{\substack{1\le j\le k\\ j\neq i}} \frac{1}{p^{j-i}-1}$$

$O(k)$预处理$\prod_{i=1}^{j} \frac{1}{p^i-1}$（包括$j<0$），就可以在$O(k\log k)$时间内求出所有$A_i$了。
你问我怎么想到这样构造的？我只能说，无可奉告。
这个做法是我校一个叫SYC1999的同学受到CodeChef CHANGE一题的启发想到的。

至此，我们已经证好了A_k这样的构造是正确的。
考虑如何快速计算A_k。

$$A_{k,i} = \prod_{\substack{1\le j\le k\\ j\neq i}} \frac{1}{1-\frac{1-p^j}{1-p^i}} = \prod_{\substack{1\le j\le k\\ j\neq i}} \frac{1-p^i}{p^j-p^i} = \left(\frac{1-p^i}{p^i}\right)^{k-1} \cdot \prod_{\substack{1\le j\le k\\ j\neq i}} \frac{1}{p^{j-i}-1}$$

$O(k)$预处理$\prod_{i=1}^{j} \frac{1}{p^i-1}$（包括$j<0$），就可以在$O(k\log k)$时间内求出所有$A_i$了。
你问我怎么想到这样构造的？我只能说，无可奉告。
这个做法是我校一个叫SYC1999的同学受到CodeChef CHANGE一题的启发想到的。不过他通过直接令$x=\frac{1}{1-p^i}$代入得到。

至此，我们已经证好了A_k这样的构造是正确的。
考虑如何快速计算A_k。

$$A_{k,i} = \prod_{\substack{1\le j\le k\\ j\neq i}} \frac{1}{1-\frac{1-p^j}{1-p^i}} = \prod_{\substack{1\le j\le k\\ j\neq i}} \frac{1-p^i}{p^j-p^i} = \left(\frac{1-p^i}{p^i}\right)^{k-1} \cdot \prod_{\substack{1\le j\le k\\ j\neq i}} \frac{1}{p^{j-i}-1}$$

$O(k)$预处理$\prod_{i=1}^{j} \frac{1}{p^i-1}$（包括$j<0$），就可以在$O(k\log k)$时间内求出所有$A_i$了。
你问我怎么想到这样构造的？我只能说，无可奉告。
这个做法是我校一个叫SYC1999的同学受到CodeChef CHANGE一题的启发想到的。不过他通过直接令$x=\frac{1}{1-p^i}$代入得到。然而由于之后的推导中$x=\frac{1}{1-p^i}$可能导致某个分母为0而出错。

至此，我们已经证好了A_k这样的构造是正确的。
考虑如何快速计算A_k。

$$A_{k,i} = \prod_{\substack{1\le j\le k\\ j\neq i}} \frac{1}{1-\frac{1-p^j}{1-p^i}} = \prod_{\substack{1\le j\le k\\ j\neq i}} \frac{1-p^i}{p^j-p^i} = \left(\frac{1-p^i}{p^i}\right)^{k-1} \cdot \prod_{\substack{1\le j\le k\\ j\neq i}} \frac{1}{p^{j-i}-1}$$

$O(k)$预处理$\prod_{i=1}^{j} \frac{1}{p^i-1}$（包括$j<0$），就可以在$O(k\log k)$时间内求出所有$A_i$了。
你问我怎么想到这样构造的？我只能说，无可奉告。
这个做法是我校一个叫SYC1999的同学受到CodeChef CHANGE一题的启发想到的。不过他通过直接令$x=\frac{1}{1-p^i}$代入得到。然而由于之后的推导中$x=\frac{1}{1-p^i}$可能导致某个分母为0而出错。而且还可能会导致一个形如$\frac{1}{1-x}$的无穷级数展开不收敛。

至此，我们已经证好了A_k这样的构造是正确的。
考虑如何快速计算A_k。

$$A_{k,i}=\prod_{\substack{1\le j\le k\\ j\neq i}}\frac{1}{1-\frac{1-p^j}{1-p^i}}=\prod_{\substack{1\le j\le k\\ j\neq i}}\frac{1-p^i}{p^j-p^i}=\left(\frac{1-p^i}{p^i}\right)^{k-1}\cdot\prod_{\substack{1\le j\le k\\ j\neq i}}\frac{1}{p^{j-i}-1}$$

$O(k)$预处理$\prod_{i=1}^{j}\frac{1}{p^i-1}$（包括$j<0$），就可以在$O(k\log k)$时间内求出所有$A_i$了。
你问我怎么想到这样构造的？我只能说，无可奉告。
这个做法是我校一个叫SYC1999的同学受到CodeChef CHANGE一题的启发想到的。不过他通过直接令$x=\frac{1}{1-p^i}$代入得到。然而由于之后的推导中$x=\frac{1}{1-p^i}$可能导致某个分母为0而出错。而且还可能会导致一个形如$\frac{1}{1-x}$的无穷级数展开不收敛。具体做法详见51Nod上的题解。

求出了A_k之后，我们再回到关于幂级数$h_k(x)$的推导。

求出了A_k之后，我们再回到关于幂级数$h_k(x)$的推导。

$$h_k(x) = p^{\frac{k(k+1)}{2}} x^k \cdot \prod_{i=1}^{k} \frac{1}{1-(1-p^i)x} = p^{\frac{k(k+1)}{2}} x^k \cdot \sum_{i=1}^{k} \frac{A_{k,i}}{1-(1-p^i)x}$$

求出了A_k之后，我们再回到关于幂级数$h_k(x)$的推导。

$$h_k(x) = p^{\frac{k(k+1)}{2}} x^k \cdot \prod_{i=1}^{k} \frac{1}{1-(1-p^i)x} = p^{\frac{k(k+1)}{2}} x^k \cdot \sum_{i=1}^{k} \frac{A_{k,i}}{1-(1-p^i)x}$$

我们要的答案$\times n$就是$h_k(x)$前n项系数之和，即

$$\sum_{j=0}^{n-1} [x^j] h_k(x)$$

求出了A_k之后，我们再回到关于幂级数$h_k(x)$的推导。

$$h_k(x) = p^{\frac{k(k+1)}{2}} x^k \cdot \prod_{i=1}^{k} \frac{1}{1-(1-p^i)x} = p^{\frac{k(k+1)}{2}} x^k \cdot \sum_{i=1}^{k} \frac{A_{k,i}}{1-(1-p^i)x}$$

我们要的答案$\times n$就是$h_k(x)$前n项系数之和，即

$$\begin{aligned} & \sum_{j=0}^{n-1} [x^j] h_k(x) \\ = & \sum_{j=0}^{n-1} [x^j] \left(p^{\frac{k(k+1)}{2}} x^k \cdot \sum_{i=1}^{k} \frac{A_{k,i}}{1-(1-p^i)x} \right) \end{aligned}$$

$$
\begin{aligned}
& \sum_{j=0}^{n-1} [x^j] h_k(x) \\
= & \sum_{j=0}^{n-1} [x^j] \left(p^{\frac{k(k+1)}{2}} x^k \cdot \sum_{i=1}^{k} \frac{A_{k,i}}{1-(1-p^i)x} \right)
\end{aligned}
$$

$$
\begin{aligned}
& \sum_{j=0}^{n-1} [x^j] h_k(x) \\
= & \sum_{j=0}^{n-1} [x^j] \left(p^{\frac{k(k+1)}{2}} x^k \cdot \sum_{i=1}^{k} \frac{A_{k,i}}{1-(1-p^i)x} \right) \\
= & \sum_{j=0}^{n-1} [x^j] \left(p^{\frac{k(k+1)}{2}} x^k \cdot \sum_{i=1}^{k} \left(A_{k,i} \cdot \sum_{v=0}^{\infty} (1-p^i)^v x^v \right) \right)
\end{aligned}
$$

$$
\begin{aligned}
& \sum_{j=0}^{n-1} [x^j] h_k(x) \\
= & \sum_{j=0}^{n-1} [x^j] \left(p^{\frac{k(k+1)}{2}} x^k \cdot \sum_{i=1}^{k} \frac{A_{k,i}}{1-(1-p^i)x} \right) \\
= & \sum_{j=0}^{n-1} [x^j] \left(p^{\frac{k(k+1)}{2}} x^k \cdot \sum_{i=1}^{k} \left(A_{k,i} \cdot \sum_{v=0}^{\infty} (1-p^i)^v x^v \right) \right) \\
= & \; p^{\frac{k(k+1)}{2}} \sum_{j=0}^{n-1-k} \sum_{i=1}^{k} A_{k,i} \cdot (1-p^i)^j
\end{aligned}
$$

$$
\begin{aligned}
& \sum_{j=0}^{n-1} [x^j] h_k(x) \\
= & \sum_{j=0}^{n-1} [x^j] \left(p^{\frac{k(k+1)}{2}} x^k \cdot \sum_{i=1}^{k} \frac{A_{k,i}}{1-(1-p^i)x} \right) \\
= & \sum_{j=0}^{n-1} [x^j] \left(p^{\frac{k(k+1)}{2}} x^k \cdot \sum_{i=1}^{k} \left(A_{k,i} \cdot \sum_{v=0}^{\infty} (1-p^i)^v x^v \right) \right) \\
= & \; p^{\frac{k(k+1)}{2}} \sum_{j=0}^{n-1-k} \sum_{i=1}^{k} A_{k,i} \cdot (1-p^i)^j \\
= & \; p^{\frac{k(k+1)}{2}} \sum_{i=1}^{k} A_{k,i} \cdot \frac{(1-p^i)^{n-k}-1}{(1-p^i)-1}
\end{aligned}
$$

$$
\begin{aligned}
& \sum_{j=0}^{n-1} [x^j] h_k(x) \\
= & \sum_{j=0}^{n-1} [x^j] \left(p^{\frac{k(k+1)}{2}} x^k \cdot \sum_{i=1}^{k} \frac{A_{k,i}}{1-(1-p^i)x} \right) \\
= & \sum_{j=0}^{n-1} [x^j] \left(p^{\frac{k(k+1)}{2}} x^k \cdot \sum_{i=1}^{k} \left(A_{k,i} \cdot \sum_{v=0}^{\infty} (1-p^i)^v x^v \right) \right) \\
= & \; p^{\frac{k(k+1)}{2}} \sum_{j=0}^{n-1-k} \sum_{i=1}^{k} A_{k,i} \cdot (1-p^i)^j \\
= & \; p^{\frac{k(k+1)}{2}} \sum_{i=1}^{k} A_{k,i} \cdot \frac{(1-p^i)^{n-k}-1}{(1-p^i)-1}
\end{aligned}
$$

这样就可以在$O(k \log n)$时间内求出答案了。问题解决。

集训队互测2016 赚奖金

集训队互测2016 赚奖金

给定一个长度为n的字符串s，你需要对于每个i $(0 \leq i < n)$确定一个t_i $(0 \leq t_i \leq n - i)$。

集训队互测2016 赚奖金

给定一个长度为n的字符串s，你需要对于每个i $(0 \leq i < n)$确定一个t_i $(0 \leq t_i \leq n - i)$。确定t_i的同时你可以获得w_{i,t_i}的奖金，其中对于某个i，w_i以一个分成k段的分段函数的形式给出，每一段的数都相同。

集训队互测2016 赚奖金

给定一个长度为n的字符串s，你需要对于每个i $(0 \leq i < n)$确定一个t_i $(0 \leq t_i \leq n - i)$。确定t_i的同时你可以获得w_{i,t_i}的奖金，其中对于某个i，w_i以一个分成k段的分段函数的形式给出，每一段的数都相同。接下来给出m“条件”。

集训队互测2016 赚奖金

给定一个长度为n的字符串s，你需要对于每个i $(0 \leq i < n)$确定一个t_i $(0 \leq t_i \leq n - i)$。确定t_i的同时你可以获得w_{i,t_i}的奖金，其中对于某个i，w_i以一个分成k段的分段函数的形式给出，每一段的数都相同。接下来给出m“条件”。一个“条件”以这样的形式给出：

集训队互测2016 赚奖金

给定一个长度为n的字符串s，你需要对于每个i $(0 \leq i < n)$确定一个t_i $(0 \leq t_i \leq n - i)$。确定t_i的同时你可以获得w_{i,t_i}的奖金，其中对于某个i，w_i以一个分成k段的分段函数的形式给出，每一段的数都相同。接下来给出m“条件”。一个“条件”以这样的形式给出：给定l, r $(0 \leq l < r \leq n)$和一个整数p，如果对于任意i $(0 \leq i < n)$都有$LCP(s[i, n), s[l, r)) \leq t_i$(其中$s[i, j)$表示字符串$s$下标从$i$ 到$j - 1$的字符构成的子串，$LCP(a, b)$表示字符串a, b的最长公共前缀长度)，就说这个“条件”被满足了，可以获得p元奖金，否则就不能获得奖金。

集训队互测2016 赚奖金

给定一个长度为n的字符串s，你需要对于每个i $(0 \leq i < n)$确定一个t_i $(0 \leq t_i \leq n - i)$。确定t_i的同时你可以获得w_{i,t_i}的奖金，其中对于某个i，w_i以一个分成k段的分段函数的形式给出，每一段的数都相同。接下来给出m“条件”。一个“条件”以这样的形式给出：给定l, r $(0 \leq l < r \leq n)$和一个整数p，如果对于任意i $(0 \leq i < n)$都有$LCP(s[i, n), s[l, r)) \leq t_i$(其中$s[i, j)$表示字符串$s$下标从$i$ 到$j - 1$的字符构成的子串，$LCP(a, b)$表示字符串a, b的最长公共前缀长度)，就说这个“条件”被满足了，可以获得p元奖金，否则就不能获得奖金。求最多可以获得多少奖金。

集训队互测2016 赚奖金

给定一个长度为n的字符串s，你需要对于每个i $(0 \leq i < n)$确定一个t_i $(0 \leq t_i \leq n-i)$。确定t_i的同时你可以获得w_{i,t_i}的奖金，其中对于某个i，w_i以一个分成k段的分段函数的形式给出，每一段的数都相同。接下来给出m“条件”。一个“条件”以这样的形式给出：给定l, r $(0 \leq l < r \leq n)$和一个整数p，如果对于任意i $(0 \leq i < n)$都有$LCP(s[i,n), s[l,r)) \leq t_i$(其中$s[i,j)$表示字符串$s$下标从$i$ 到$j-1$的字符构成的子串，$LCP(a,b)$表示字符串a,b的最长公共前缀长度)，就说这个“条件”被满足了，可以获得p元奖金，否则就不能获得奖金。求最多可以获得多少奖金。

数据范围：

集训队互测2016 赚奖金

给定一个长度为n的字符串s，你需要对于每个i $(0 \le i < n)$确定一个t_i $(0 \le t_i \le n - i)$。确定t_i的同时你可以获得w_{i,t_i}的奖金，其中对于某个i，w_i以一个分成k段的分段函数的形式给出，每一段的数都相同。接下来给出m“条件”。一个“条件”以这样的形式给出：给定l, r $(0 \le l < r \le n)$和一个整数p，如果对于任意i $(0 \le i < n)$都有$LCP(s[i, n), s[l, r)) \le t_i$(其中$s[i, j)$表示字符串$s$下标从$i$ 到$j - 1$的字符构成的子串，$LCP(a, b)$表示字符串a, b的最长公共前缀长度)，就说这个“条件”被满足了，可以获得p元奖金，否则就不能获得奖金。求最多可以获得多少奖金。

数据范围：

- 20%: $m \le 17$
- 40%: $n, m \le 200$
- 70%: $n \le 200,\ m \le 20100$
- 100%: $n, m \le 200000,\ \sum k \le 400000$

我们发现，t_i越小，w_{i,t_i}越大；t_i越大，满足的条件越多。所以我们就是要找到两者的“平衡”。这让人联想到最小割建模。

我们发现，t_i越小，w_{i,t_i}越大；t_i越大，满足的条件越多。所以我们就是要找到两者的“平衡”。这让人联想到最小割建模。

由于最小割的目标是最小化割边的流量之和，而原问题的目标是最大化奖金，所以要设法把最大化问题转化为最小化问题。

我们发现，t_i越小，w_{i,t_i}越大；t_i越大，满足的条件越多。所以我们就是要找到两者的“平衡”。这让人联想到最小割建模。
由于最小割的目标是最小化割边的流量之和，而原问题的目标是最大化奖金，所以要设法把最大化问题转化为最小化问题。
令网络流图中源点为S，汇点为T。

我们发现，t_i越小，w_{i,t_i}越大；t_i越大，满足的条件越多。所以我们就是要找到两者的“平衡”。这让人联想到最小割建模。

由于最小割的目标是最小化割边的流量之和，而原问题的目标是最大化奖金，所以要设法把最大化问题转化为最小化问题。

令网络流图中源点为S，汇点为T。我们对所有i, j $(0 \leq i < n,\ 0 \leq j \leq n - i + 1)$建点$A_{i,j}$。

我们发现，t_i越小，w_{i,t_i}越大；t_i越大，满足的条件越多。所以我们就是要找到两者的“平衡”。这让人联想到最小割建模。

由于最小割的目标是最小化割边的流量之和，而原问题的目标是最大化奖金，所以要设法把最大化问题转化为最小化问题。

令网络流图中源点为S，汇点为T。我们对所有i, j $(0 \leq i < n,\ 0 \leq j \leq n - i + 1)$建点$A_{i,j}$。然后，我们从$A_{i,j}$到$A_{i,j+1}$连流量为$10^9 - w_{i,j}$的边，从$S$ 到$A_{i,0}$连流量为∞的边，从$A_{i,n-i+1}$ 到T连流量为∞的边。

我们发现，t_i越小，w_{i,t_i}越大；t_i越大，满足的条件越多。所以我们就是要找到两者的“平衡”。这让人联想到最小割建模。

由于最小割的目标是最小化割边的流量之和，而原问题的目标是最大化奖金，所以要设法把最大化问题转化为最小化问题。

令网络流图中源点为S，汇点为T。我们对所有i, j $(0 \le i < n,\ 0 \le j \le n - i + 1)$建点$A_{i,j}$。然后，我们从$A_{i,j}$到$A_{i,j+1}$连流量为$10^9 - w_{i,j}$的边，从$S$ 到$A_{i,0}$连流量为∞的边，从$A_{i,n-i+1}$ 到T连流量为∞的边。这样i对应的一条链上必然会割流量最小的一条边，设这条边是从$A_{i,j}$到$A_{i,j+1}$的，那么我们就令$t_i = j$。

我们发现，t_i越小，w_{i,t_i}越大；t_i越大，满足的条件越多。所以我们就是要找到两者的“平衡”。这让人联想到最小割建模。

由于最小割的目标是最小化割边的流量之和，而原问题的目标是最大化奖金，所以要设法把最大化问题转化为最小化问题。

令网络流图中源点为S，汇点为T。我们对所有i, j $(0 \le i < n,\ 0 \le j \le n - i + 1)$建点$A_{i,j}$。然后，我们从$A_{i,j}$到$A_{i,j+1}$连流量为$10^9 - w_{i,j}$的边，从$S$ 到$A_{i,0}$连流量为∞的边，从$A_{i,n-i+1}$ 到T连流量为∞的边。这样i对应的一条链上必然会割流量最小的一条边，设这条边是从$A_{i,j}$到$A_{i,j+1}$的，那么我们就令$t_i = j$。这样，割尽可能小的边就对应了权尽可能大的边。

对于第k个条件，我们建点C_k，从S到C_k连流量为p的边。

对于第k个条件，我们建点C_k，从S到C_k连流量为p的边。然后对于每个i，从C_k到$A_{i,LCP(s[l,r),s[i,n))}$连流量为∞的边。

对于第k个条件，我们建点C_k，从S到C_k连流量为p的边。然后对于每个i，从C_k到$A_{i,LCP(s[l,r),s[i,n))}$连流量为∞的边。这样，如果不割S到C_k的边，所有i对应的链上割的边必然在C_k连出去的边之后，也就会满足条件；如果割了S到C_k的边，也就失去了这个条件的奖金。

对于第k个条件，我们建点C_k，从S到C_k连流量为p的边。然后对于每个i，从C_k到$A_{i,LCP(s[l,r),s[i,n))}$连流量为∞的边。这样，如果不割S到C_k的边，所有i对应的链上割的边必然在C_k连出去的边之后，也就会满足条件；如果割了S到C_k的边，也就失去了这个条件的奖金。

因此，我们只要用$10^9 \times n + \sum p$减去整张图的最小割就可以得到答案了。

对于第k个条件，我们建点C_k，从S到C_k连流量为p的边。然后对于每个i，从C_k到$A_{i,LCP(s[l,r),s[i,n))}$连流量为∞的边。这样，如果不割S到C_k的边，所有i对应的链上割的边必然在C_k连出去的边之后，也就会满足条件；如果割了S到C_k的边，也就失去了这个条件的奖金。

因此，我们只要用$10^9 \times n + \sum p$减去整张图的最小割就可以得到答案了。

时间复杂度：$O(maxflow(n^2+m, n^2+mn))$，期望得分40分。

考虑优化算法3的连边。

考虑优化算法3的连边。
我们要想办法利用LCP的性质。

考虑优化算法3的连边。
我们要想办法利用LCP的性质。想一想我们可以用什么来快速地求这类LCP?

考虑优化算法3的连边。
我们要想办法利用LCP的性质。想一想我们可以用什么来快速地求这类LCP？没错！一种方法是用后缀数组。

考虑优化算法3的连边。
我们要想办法利用LCP的性质。想一想我们可以用什么来快速地求这类LCP？没错！一种方法是用后缀数组。对于某个条件，我们只要找到后缀$s[l,n)$在后缀数组中的那一行（设为第$rank_l$ 行），那么从第$rank_l$行往上走，LCP将不断地与$height$取min，也就是LCP会呈现阶梯状单调不增的样子。往下走也是一样。

于是，我们可以用连边来模拟这个过程。

于是，我们可以用连边来模拟这个过程。我们求出后缀数组。

于是，我们可以用连边来模拟这个过程。我们求出后缀数组。然后对于所有i, j $(0 \leq i < n,\ 0 \leq j \leq n - i + 1)$建两个点$A_{i,j}, B_{i,j}$，并从$B_{i,j}$ 向$A_{i,j}$ 连流量为∞的边。

于是，我们可以用连边来模拟这个过程。我们求出后缀数组。然后对于所有i, j $(0 \leq i < n,\ 0 \leq j \leq n - i + 1)$建两个点$A_{i,j}, B_{i,j}$，并从$B_{i,j}$ 向$A_{i,j}$ 连流量为∞的边。对于所有$A_{i,j}$与S, T这些点之间的连边仍旧按照算法3中的方式。

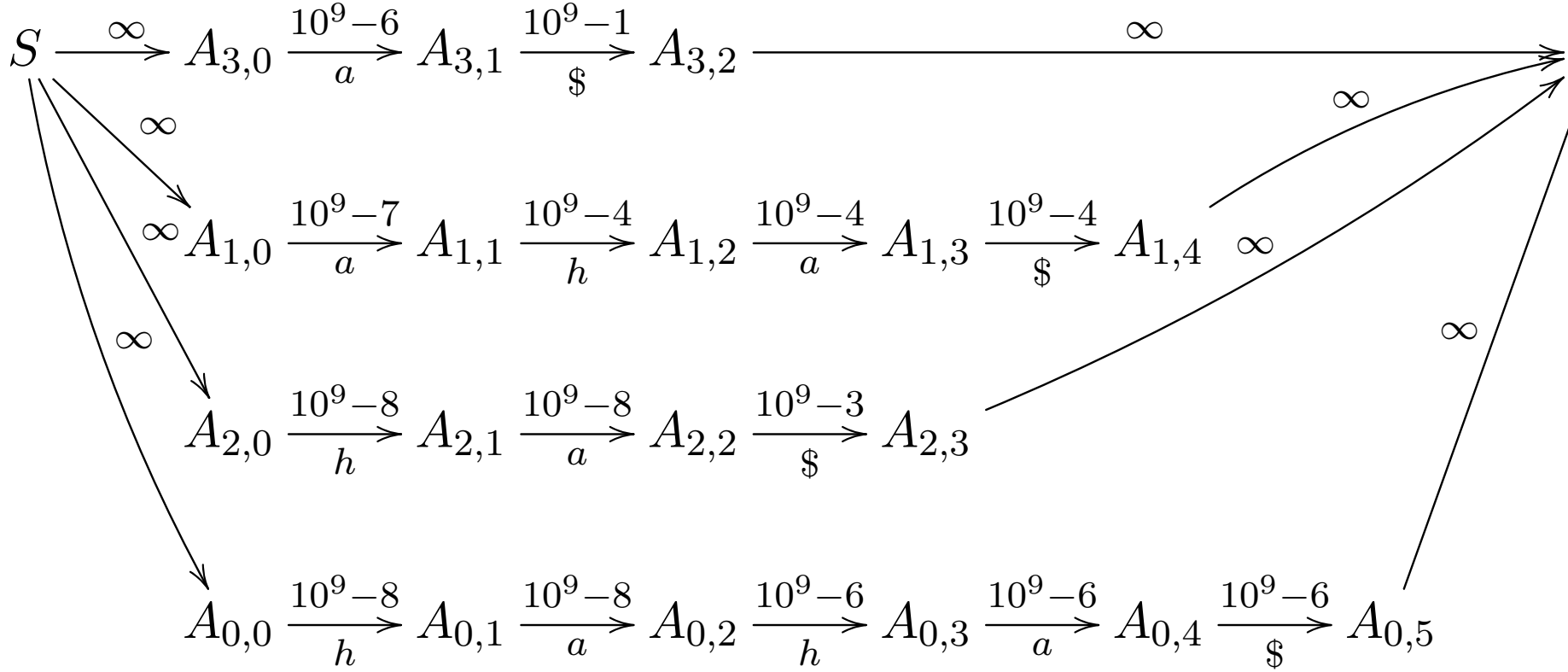
于是，我们可以用连边来模拟这个过程。我们求出后缀数组。然后对于所有i, j $(0 \leq i < n,\ 0 \leq j \leq n - i + 1)$建两个点$A_{i,j}, B_{i,j}$，并从$B_{i,j}$ 向$A_{i,j}$ 连流量为∞的边。对于所有$A_{i,j}$与S, T这些点之间的连边仍旧按照算法3中的方式。然后，我们从$B_{i,j}$到$B_{i,j-1}$连流量为∞的边。

于是，我们可以用连边来模拟这个过程。我们求出后缀数组。然后对于所有i, j $(0 \leq i < n,\ 0 \leq j \leq n - i + 1)$建两个点$A_{i,j}, B_{i,j}$，并从$B_{i,j}$ 向$A_{i,j}$ 连流量为∞的边。对于所有$A_{i,j}$与S, T这些点之间的连边仍旧按照算法3中的方式。然后，我们从$B_{i,j}$到$B_{i,j-1}$连流量为∞的边。如果对于后缀数组中某两个相邻后缀（设为sa_{i-1}和sa_i）的LCP长度（设为$height_i$），我们对所有j $(0 \leq j \leq height_i)$，在$B_{sa_{i-1},j}$ 与$B_{sa_i,j}$之间连双向的流量为∞的边。

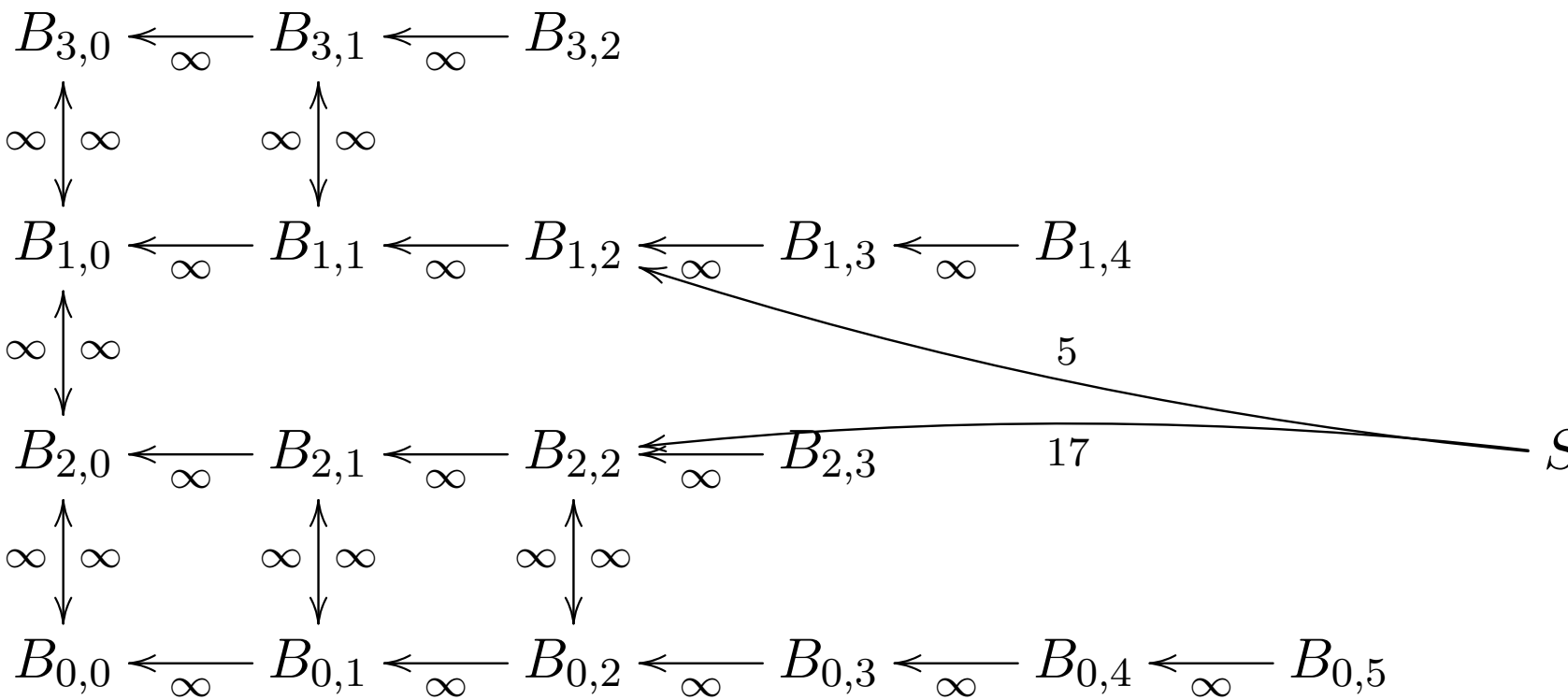
于是，我们可以用连边来模拟这个过程。我们求出后缀数组。然后对于所有i, j $(0 \leq i < n,\ 0 \leq j \leq n - i + 1)$建两个点$A_{i,j}, B_{i,j}$，并从$B_{i,j}$ 向$A_{i,j}$ 连流量为∞的边。对于所有$A_{i,j}$与S, T这些点之间的连边仍旧按照算法3中的方式。然后，我们从$B_{i,j}$到$B_{i,j-1}$连流量为∞的边。如果对于后缀数组中某两个相邻后缀（设为sa_{i-1}和sa_i）的LCP长度（设为$height_i$），我们对所有j $(0 \leq j \leq height_i)$，在$B_{sa_{i-1},j}$ 与$B_{sa_i,j}$之间连双向的流量为∞的边。最后，对于每个条件，我们从S到$B_{l,r-l}$连流量为p的边。整张图就等价于算法3中的图。

以下是样例构出来的图。第一张图是A点的连边，第二张图是B点的连边（没有画出$A_{i,j} \to B_{i,j}$的边）。

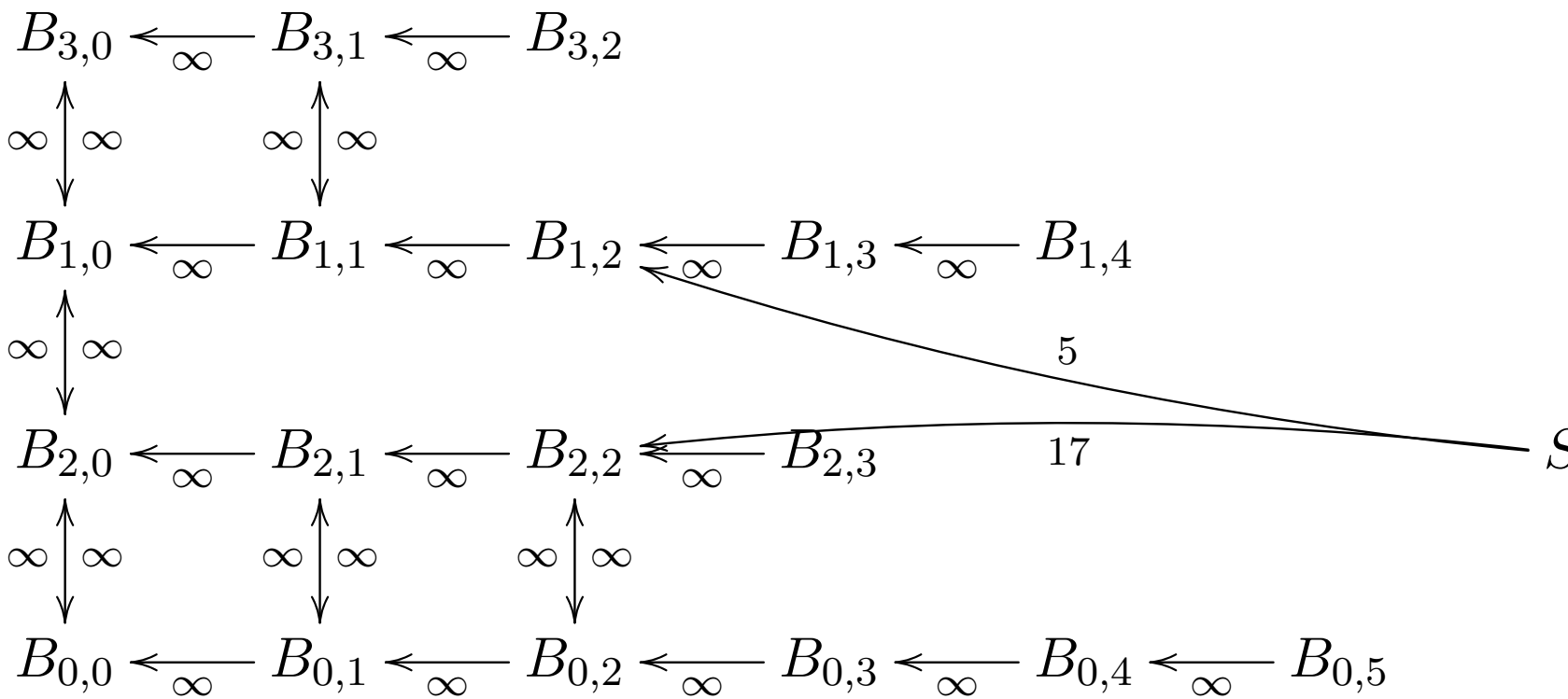
以下是样例构出来的图。第一张图是A点的连边，第二张图是B点的连边（没有画出$A_{i,j} \to B_{i,j}$的边）。



以下是样例构出来的图。第一张图是A点的连边，第二张图是B点的连边（没有画出$A_{i,j} \to B_{i,j}$的边）。



以下是样例构出来的图。第一张图是A点的连边，第二张图是B点的连边（没有画出$A_{i,j} \to B_{i,j}$的边）。



时间复杂度：$O(maxflow(n^2, n^2+m))$，期望得分70分。

我们重新回过头去想如何利用LCP的性质来优化连边。

我们重新回过头去想如何利用LCP的性质来优化连边。可以发现后缀树也可以用来求LCP。

我们重新回过头去想如何利用LCP的性质来优化连边。可以发现后缀树也可以用来求LCP。只是后缀树会把相同的前缀并成一条链，这样看上去不同的后缀割的位置会互相影响，似乎不能像后缀数组那样建图。

我们重新回过头去想如何利用LCP的性质来优化连边。可以发现后缀树也可以用来求LCP。只是后缀树会把相同的前缀并成一条链，这样看上去不同的后缀割的位置会互相影响，似乎不能像后缀数组那样建图。但实际上是可以的。

我们重新回过头去想如何利用LCP的性质来优化连边。可以发现后缀树也可以用来求LCP。只是后缀树会把相同的前缀并成一条链，这样看上去不同的后缀割的位置会互相影响，似乎不能像后缀数组那样建图。但实际上是可以的。也就是说，如果我们把后缀数组中关于$height$的连边两端的点都并在一起，整张图仍然是正确的。

我们重新回过头去想如何利用LCP的性质来优化连边。可以发现后缀树也可以用来求LCP。只是后缀树会把相同的前缀并成一条链，这样看上去不同的后缀割的位置会互相影响，似乎不能像后缀数组那样建图。但实际上是可以的。也就是说，如果我们把后缀数组中关于$height$的连边两端的点都并在一起，整张图仍然是正确的。

我们来证明这个性质。

我们重新回过头去想如何利用LCP的性质来优化连边。可以发现后缀树也可以用来求LCP。只是后缀树会把相同的前缀并成一条链，这样看上去不同的后缀割的位置会互相影响，似乎不能像后缀数组那样建图。但实际上是可以的。也就是说，如果我们把后缀数组中关于$height$的连边两端的点都并在一起，整张图仍然是正确的。

我们来证明这个性质。这个性质相当于是说，对于任意两个后缀（设为$s[i,n)$和$s[j,n)$），如果$t_i < LCP(s[i,n), s[j,n))$，那么必有$t_j = t_i$。

我们重新回过头去想如何利用LCP的性质来优化连边。可以发现后缀树也可以用来求LCP。只是后缀树会把相同的前缀并成一条链，这样看上去不同的后缀割的位置会互相影响，似乎不能像后缀数组那样建图。但实际上是可以的。也就是说，如果我们把后缀数组中关于$height$的连边两端的点都并在一起，整张图仍然是正确的。

我们来证明这个性质。这个性质相当于是说，对于任意两个后缀（设为$s[i,n)$和$s[j,n)$），如果$t_i < LCP(s[i,n), s[j,n))$，那么必有$t_j = t_i$。假设$t_j > t_i$，由于$t_j$肯定是越小越好，所以$t_j$不能变成更小的$t_i$一定是为了满足某个$t_i < r - l \leq t_j$的条件。

我们重新回过头去想如何利用LCP的性质来优化连边。可以发现后缀树也可以用来求LCP。只是后缀树会把相同的前缀并成一条链，这样看上去不同的后缀割的位置会互相影响，似乎不能像后缀数组那样建图。但实际上是可以的。也就是说，如果我们把后缀数组中关于$height$的连边两端的点都并在一起，整张图仍然是正确的。

我们来证明这个性质。这个性质相当于是说，对于任意两个后缀（设为$s[i,n)$和$s[j,n)$），如果$t_i < LCP(s[i,n), s[j,n))$，那么必有$t_j = t_i$。假设$t_j > t_i$，由于$t_j$肯定是越小越好，所以$t_j$不能变成更小的$t_i$一定是为了满足某个$t_i < r - l \leq t_j$的条件。可是$t_i$显然不满足这个条件。

我们重新回过头去想如何利用LCP的性质来优化连边。可以发现后缀树也可以用来求LCP。只是后缀树会把相同的前缀并成一条链，这样看上去不同的后缀割的位置会互相影响，似乎不能像后缀数组那样建图。但实际上是可以的。也就是说，如果我们把后缀数组中关于$height$的连边两端的点都并在一起，整张图仍然是正确的。

我们来证明这个性质。这个性质相当于是说，对于任意两个后缀（设为$s[i,n)$和$s[j,n)$），如果$t_i < LCP(s[i,n), s[j,n))$，那么必有$t_j = t_i$。假设$t_j > t_i$，由于$t_j$肯定是越小越好，所以$t_j$不能变成更小的$t_i$一定是为了满足某个$t_i < r - l \leq t_j$的条件。可是$t_i$显然不满足这个条件。因此这样的情况是不存在的。

我们重新回过头去想如何利用LCP的性质来优化连边。可以发现后缀树也可以用来求LCP。只是后缀树会把相同的前缀并成一条链，这样看上去不同的后缀割的位置会互相影响，似乎不能像后缀数组那样建图。但实际上是可以的。也就是说，如果我们把后缀数组中关于$height$的连边两端的点都并在一起，整张图仍然是正确的。

我们来证明这个性质。这个性质相当于是说，对于任意两个后缀（设为$s[i,n)$和$s[j,n)$），如果$t_i < LCP(s[i,n), s[j,n))$，那么必有$t_j = t_i$。假设$t_j > t_i$，由于$t_j$肯定是越小越好，所以$t_j$不能变成更小的$t_i$一定是为了满足某个$t_i < r - l \leq t_j$的条件。可是$t_i$显然不满足这个条件。因此这样的情况是不存在的。$t_j < t_i$的情况只要把两个后缀交换一下进行上面的推导就好了。

我们重新回过头去想如何利用LCP的性质来优化连边。可以发现后缀树也可以用来求LCP。只是后缀树会把相同的前缀并成一条链，这样看上去不同的后缀割的位置会互相影响，似乎不能像后缀数组那样建图。但实际上是可以的。也就是说，如果我们把后缀数组中关于$height$的连边两端的点都并在一起，整张图仍然是正确的。

我们来证明这个性质。这个性质相当于是说，对于任意两个后缀（设为$s[i,n)$和$s[j,n)$），如果$t_i < LCP(s[i,n), s[j,n))$，那么必有$t_j = t_i$。假设$t_j > t_i$，由于$t_j$肯定是越小越好，所以$t_j$不能变成更小的$t_i$一定是为了满足某个$t_i < r - l \leq t_j$的条件。可是$t_i$显然不满足这个条件。因此这样的情况是不存在的。$t_j < t_i$的情况只要把两个后缀交换一下进行上面的推导就好了。这样就证明了这个性质。

我们重新回过头去想如何利用LCP的性质来优化连边。可以发现后缀树也可以用来求LCP。只是后缀树会把相同的前缀并成一条链，这样看上去不同的后缀割的位置会互相影响，似乎不能像后缀数组那样建图。但实际上是可以的。也就是说，如果我们把后缀数组中关于$height$的连边两端的点都并在一起，整张图仍然是正确的。

我们来证明这个性质。这个性质相当于是说，对于任意两个后缀（设为$s[i,n)$和$s[j,n)$），如果$t_i < LCP(s[i,n), s[j,n))$，那么必有$t_j = t_i$。假设$t_j > t_i$，由于$t_j$肯定是越小越好，所以$t_j$不能变成更小的$t_i$一定是为了满足某个$t_i < r - l \leq t_j$的条件。可是$t_i$显然不满足这个条件。因此这样的情况是不存在的。$t_j < t_i$的情况只要把两个后缀交换一下进行上面的推导就好了。这样就证明了这个性质。

时间复杂度：$O(maxflow(n^2, n^2 + m))$，期望得分70分。

仔细一看可以发现，这个后缀树上的最小割问题可以直接用树形DP来解决。

仔细一看可以发现，这个后缀树上的最小割问题可以直接用树形DP来解决。因为对于某个条件，对应在后缀树上的那个B点会往它所有祖先一路连边，根据A的连边方式可知，该点到根的路径上的所有A与A之间的边都不能割。

仔细一看可以发现，这个后缀树上的最小割问题可以直接用树形DP来解决。因为对于某个条件，对应在后缀树上的那个B点会往它所有祖先一路连边，根据A的连边方式可知，该点到根的路径上的所有A与A之间的边都不能割。也就是说，如果该点对应的A点与根连通，就可以获得该条件的奖金p，否则不能获得。

仔细一看可以发现，这个后缀树上的最小割问题可以直接用树形DP来解决。因为对于某个条件，对应在后缀树上的那个B点会往它所有祖先一路连边，根据A的连边方式可知，该点到根的路径上的所有A与A之间的边都不能割。也就是说，如果该点对应的A点与根连通，就可以获得该条件的奖金p，否则不能获得。因此，DP时可以用f_u表示如果点u的父亲与根连通，u的子树里最多可以获得多少奖金。

仔细一看可以发现，这个后缀树上的最小割问题可以直接用树形DP来解决。因为对于某个条件，对应在后缀树上的那个B点会往它所有祖先一路连边，根据A的连边方式可知，该点到根的路径上的所有A与A之间的边都不能割。也就是说，如果该点对应的A点与根连通，就可以获得该条件的奖金p，否则不能获得。因此，DP时可以用f_u表示如果点u的父亲与根连通，u的子树里最多可以获得多少奖金。如果割u到父亲的边，就可以获得相应w的奖金，否则就获得u上的所有奖金p以及所有儿子的f值之和。

仔细一看可以发现，这个后缀树上的最小割问题可以直接用树形DP来解决。因为对于某个条件，对应在后缀树上的那个B点会往它所有祖先一路连边，根据A的连边方式可知，该点到根的路径上的所有A与A之间的边都不能割。也就是说，如果该点对应的A点与根连通，就可以获得该条件的奖金p，否则不能获得。因此，DP时可以用f_u表示如果点u的父亲与根连通，u的子树里最多可以获得多少奖金。如果割u到父亲的边，就可以获得相应w的奖金，否则就获得u上的所有奖金p以及所有儿子的f值之和。

于是，我们可以直接建出$O(N^2)$个点的后缀树，做一遍DP就好了。

仔细一看可以发现，这个后缀树上的最小割问题可以直接用树形DP来解决。因为对于某个条件，对应在后缀树上的那个B点会往它所有祖先一路连边，根据A的连边方式可知，该点到根的路径上的所有A与A之间的边都不能割。也就是说，如果该点对应的A点与根连通，就可以获得该条件的奖金p，否则不能获得。因此，DP时可以用f_u表示如果点u的父亲与根连通，u的子树里最多可以获得多少奖金。如果割u到父亲的边，就可以获得相应w的奖金，否则就获得u上的所有奖金p以及所有儿子的f值之和。

于是，我们可以直接建出$O(N^2)$个点的后缀树，做一遍DP就好了。

时间复杂度：$O(n^2+m)$

考虑进一步优化。

考虑进一步优化。由于分段函数每一段相同的值如果不会影响其他东西的话割哪一条边都是一样的，所以不需要建出每个点来。

考虑进一步优化。由于分段函数每一段相同的值如果不会影响其他东西的话割哪一条边都是一样的，所以不需要建出每个点来。我们用后缀自动机或Ukkonen算法或后缀仙人掌建出缩边后的后缀树。

考虑进一步优化。由于分段函数每一段相同的值如果不会影响其他东西的话割哪一条边都是一样的，所以不需要建出每个点来。我们用后缀自动机或Ukkonen算法或后缀仙人掌建出缩边后的后缀树。对于后缀树中的每个点记录奖金之和p以及它到父亲的边的边权w。

考虑进一步优化。由于分段函数每一段相同的值如果不会影响其他东西的话割哪一条边都是一样的，所以不需要建出每个点来。我们用后缀自动机或Ukkonen算法或后缀仙人掌建出缩边后的后缀树。对于后缀树中的每个点记录奖金之和p以及它到父亲的边的边权w。然后对于每个分段函数，找到分段点在后缀树上的位置插入一个点。

考虑进一步优化。由于分段函数每一段相同的值如果不会影响其他东西的话割哪一条边都是一样的，所以不需要建出每个点来。我们用后缀自动机或Ukkonen算法或后缀仙人掌建出缩边后的后缀树。对于后缀树中的每个点记录奖金之和p以及它到父亲的边的边权w。然后对于每个分段函数，找到分段点在后缀树上的位置插入一个点。对于每个条件，也找到对应的位置插入一个点，把奖金p累加到这个点上。

考虑进一步优化。由于分段函数每一段相同的值如果不会影响其他东西的话割哪一条边都是一样的，所以不需要建出每个点来。我们用后缀自动机或Ukkonen算法或后缀仙人掌建出缩边后的后缀树。对于后缀树中的每个点记录奖金之和p以及它到父亲的边的边权w。然后对于每个分段函数，找到分段点在后缀树上的位置插入一个点。对于每个条件，也找到对应的位置插入一个点，把奖金p累加到这个点上。然后对于每个分段函数，把后缀树上相应的链上所有边都加上相应的权值。

考虑进一步优化。由于分段函数每一段相同的值如果不会影响其他东西的话割哪一条边都是一样的，所以不需要建出每个点来。我们用后缀自动机或Ukkonen算法或后缀仙人掌建出缩边后的后缀树。对于后缀树中的每个点记录奖金之和p以及它到父亲的边的边权w。然后对于每个分段函数，找到分段点在后缀树上的位置插入一个点。对于每个条件，也找到对应的位置插入一个点，把奖金p累加到这个点上。然后对于每个分段函数，把后缀树上相应的链上所有边都加上相应的权值。这个可以作差然后线性地求一遍子树和。这样就可以在后缀树上DP了。

考虑进一步优化。由于分段函数每一段相同的值如果不会影响其他东西的话割哪一条边都是一样的，所以不需要建出每个点来。我们用后缀自动机或Ukkonen算法或后缀仙人掌建出缩边后的后缀树。对于后缀树中的每个点记录奖金之和p以及它到父亲的边的边权w。然后对于每个分段函数，找到分段点在后缀树上的位置插入一个点。对于每个条件，也找到对应的位置插入一个点，把奖金p累加到这个点上。然后对于每个分段函数，把后缀树上相应的链上所有边都加上相应的权值。这个可以作差然后线性地求一遍子树和。这样就可以在后缀树上DP了。
虽然DP是线性的，但是找位置需要在树上倍增、二分，所以还是有一个log。

考虑进一步优化。由于分段函数每一段相同的值如果不会影响其他东西的话割哪一条边都是一样的，所以不需要建出每个点来。我们用后缀自动机或Ukkonen算法或后缀仙人掌建出缩边后的后缀树。对于后缀树中的每个点记录奖金之和p以及它到父亲的边的边权w。然后对于每个分段函数，找到分段点在后缀树上的位置插入一个点。对于每个条件，也找到对应的位置插入一个点，把奖金p累加到这个点上。然后对于每个分段函数，把后缀树上相应的链上所有边都加上相应的权值。这个可以作差然后线性地求一遍子树和。这样就可以在后缀树上DP了。

虽然DP是线性的，但是找位置需要在树上倍增、二分，所以还是有一个log。

时间复杂度：$O(n + m \log n + \sum k(\log n + \log m))$，期望得分100分。

祝大家省选顺利！