图论

ExfJoe

清华大学

February 9, 2018

Part I

基础知识

基础定义

基础定义

- 图：一个有序二元组 (V, E)，其中 V 是顶点集，E 为边集，$E \subseteq V \times V$

基础定义

- 图：一个有序二元组 (V, E)，其中 V 是顶点集，E 为边集，$E \subseteq V \times V$
- 若 $G = (V, E)$ 中对任意 $(u, v) \in E$ 有 $(v, u) \in E$，则称 G 为无向图，否则为有向图，下面以无向图为例介绍

基础定义

- 图：一个有序二元组 (V, E)，其中 V 是顶点集，E 为边集，$E \subseteq V \times V$
- 若 $G = (V, E)$ 中对任意 $(u, v) \in E$ 有 $(v, u) \in E$，则称 G 为无向图，否则为有向图，下面以无向图为例介绍
- 子图：(V', E') 是 (V, E) 的子图当且仅当 $V' \subseteq V, E' \subseteq E$

基础定义

- 图：一个有序二元组 (V, E)，其中 V 是顶点集，E 为边集，$E \subseteq V \times V$
- 若 $G = (V, E)$ 中对任意 $(u, v) \in E$ 有 $(v, u) \in E$，则称 G 为无向图，否则为有向图，下面以无向图为例介绍
- 子图：(V', E') 是 (V, E) 的子图当且仅当 $V' \subseteq V, E' \subseteq E$
- 生成子图：(V, E') 是 (V, E) 的生成子图当且仅当 $E' \subseteq E$

基础定义

- 图：一个有序二元组 (V, E)，其中 V 是顶点集，E 为边集，$E \subseteq V \times V$
- 若 $G = (V, E)$ 中对任意 $(u, v) \in E$ 有 $(v, u) \in E$，则称 G 为无向图，否则为有向图，下面以无向图为例介绍
- 子图：(V', E') 是 (V, E) 的子图当且仅当 $V' \subseteq V, E' \subseteq E$
- 生成子图：(V, E') 是 (V, E) 的生成子图当且仅当 $E' \subseteq E$
- 导出子图：$G' = (V', E')$ 是 $G = (V, E)$ 的导出子图当且仅当 G' 是 G 的子图，且对于任意 $u \in V', v \in V', (u, v) \in E$ 有 $(u, v) \in E'$

基础定义

- 图：一个有序二元组 (V, E)，其中 V 是顶点集，E 为边集，$E \subseteq V \times V$
- 若 $G = (V, E)$ 中对任意 $(u, v) \in E$ 有 $(v, u) \in E$，则称 G 为无向图，否则为有向图，下面以无向图为例介绍
- 子图：(V', E') 是 (V, E) 的子图当且仅当 $V' \subseteq V, E' \subseteq E$
- 生成子图：(V, E') 是 (V, E) 的生成子图当且仅当 $E' \subseteq E$
- 导出子图：$G' = (V', E')$ 是 $G = (V, E)$ 的导出子图当且仅当 G' 是 G 的子图，且对于任意 $u \in V', v \in V', (u, v) \in E$ 有 $(u, v) \in E'$
- 度：函数 $d : V \to \mathbb{Z}$，$d(x) = |E \cap (\{x\} \times V)|$

基础定义

- 图：一个有序二元组 (V, E)，其中 V 是顶点集，E 为边集，$E \subseteq V \times V$
- 若 $G = (V, E)$ 中对任意 $(u, v) \in E$ 有 $(v, u) \in E$，则称 G 为无向图，否则为有向图，下面以无向图为例介绍
- 子图：(V', E') 是 (V, E) 的子图当且仅当 $V' \subseteq V, E' \subseteq E$
- 生成子图：(V, E') 是 (V, E) 的生成子图当且仅当 $E' \subseteq E$
- 导出子图：$G' = (V', E')$ 是 $G = (V, E)$ 的导出子图当且仅当 G' 是 G 的子图，且对于任意 $u \in V', v \in V', (u, v) \in E$ 有 $(u, v) \in E'$
- 度：函数 $d : V \to \mathbb{Z}$，$d(x) = |E \cap (\{x\} \times V)|$
- 路径：设 $G = (V, E), n \in \mathbb{Z}$，一组有序对 $A \in V^n$ 被称为一条 G 的长度为 $n-1$ 的路径当且仅当 $\forall 1 \leq i < n, (A_i, A_{i+1}) \in E$。特别地，若 $n \geq 2$ 且 $A_1 = A_n$ 则称 A 是 G 的长度为 $n-1$ 的一个环 (圈、回路)

基础定义

- 图：一个有序二元组 (V, E)，其中 V 是顶点集，E 为边集，$E \subseteq V \times V$
- 若 $G = (V, E)$ 中对任意 $(u, v) \in E$ 有 $(v, u) \in E$，则称 G 为无向图，否则为有向图，下面以无向图为例介绍
- 子图：(V', E') 是 (V, E) 的子图当且仅当 $V' \subseteq V, E' \subseteq E$
- 生成子图：(V, E') 是 (V, E) 的生成子图当且仅当 $E' \subseteq E$
- 导出子图：$G' = (V', E')$ 是 $G = (V, E)$ 的导出子图当且仅当 G' 是 G 的子图，且对于任意 $u \in V', v \in V', (u, v) \in E$ 有 $(u, v) \in E'$
- 度：函数 $d : V \to \mathbb{Z}$，$d(x) = |E \cap (\{x\} \times V)|$
- 路径：设 $G = (V, E), n \in \mathbb{Z}$，一组有序对 $A \in V^n$ 被称为一条 G 的长度为 $n-1$ 的路径当且仅当 $\forall 1 \le i < n, (A_i, A_{i+1}) \in E$。特别地，若 $n \ge 2$ 且 $A_1 = A_n$ 则称 A 是 G 的长度为 $n-1$ 的一个环 (圈、回路)
- 很多其他术语仍然可以用集合论的方法阐述，为了方便下面开始说人话

基础定义

基础定义

- 连通性：设 $G=(V,E), u,v \in V$，若 G 中存在一条长度为 n 的路径 A 使得 $A_1 = u, A_n = v$ 则称 u,v 连通

基础定义

- 连通性：设 $G = (V, E), u, v \in V$，若 G 中存在一条长度为 n 的路径 A 使得 $A_1 = u, A_n = v$ 则称 u, v 连通
- 独立集：图中两两之间无边的点集

基础定义

- 连通性：设 $G = (V, E), u, v \in V$，若 G 中存在一条长度为 n 的路径 A 使得 $A_1 = u, A_n = v$ 则称 u, v 连通
- 独立集：图中两两之间无边的点集
- 最大独立集：含点数最多的独立集，极大独立集：不存在点数更多的独立集是自身超集的独立集。这两种集合都可能有多个

基础定义

- 连通性：设 $G = (V, E), u, v \in V$，若 G 中存在一条长度为 n 的路径 A 使得 $A_1 = u, A_n = v$ 则称 u, v 连通
- 独立集：图中两两之间无边的点集
- 最大独立集：含点数最多的独立集，极大独立集：不存在点数更多的独立集是自身超集的独立集。这两种集合都可能有多个
- 团：图中两两之间都有边的点集，类似的有最大团、极大团。显然一个图的独立集对应其补图的团

基础定义

- 连通性：设 $G = (V, E), u, v \in V$，若 G 中存在一条长度为 n 的路径 A 使得 $A_1 = u, A_n = v$ 则称 u, v 连通
- 独立集：图中两两之间无边的点集
- 最大独立集：含点数最多的独立集，极大独立集：不存在点数更多的独立集是自身超集的独立集。这两种集合都可能有多个
- 团：图中两两之间都有边的点集，类似的有最大团、极大团。显然一个图的独立集对应其补图的团
- 点覆盖：若图中所有边都至少有一个端点在点集 P 中，则称 P 是一个点覆盖，最小点覆盖：点数最少的覆盖集

基础定义

- 连通性：设 $G=(V,E), u,v \in V$，若 G 中存在一条长度为 n 的路径 A 使得 $A_1=u, A_n=v$ 则称 u,v 连通
- 独立集：图中两两之间无边的点集
- 最大独立集：含点数最多的独立集，极大独立集：不存在点数更多的独立集是自身超集的独立集。这两种集合都可能有多个
- 团：图中两两之间都有边的点集，类似的有最大团、极大团。显然一个图的独立集对应其补图的团
- 点覆盖：若图中所有边都至少有一个端点在点集 P 中，则称 P 是一个点覆盖，最小点覆盖：点数最少的覆盖集
- 定理：$|\text{最小点覆盖}|+|\text{最大独立集}|=\text{图中点数}$

基础定义

- 连通性：设 $G=(V,E), u,v \in V$，若 G 中存在一条长度为 n 的路径 A 使得 $A_1 = u, A_n = v$ 则称 u,v 连通
- 独立集：图中两两之间无边的点集
- 最大独立集：含点数最多的独立集，极大独立集：不存在点数更多的独立集是自身超集的独立集。这两种集合都可能有多个
- 团：图中两两之间都有边的点集，类似的有最大团、极大团。显然一个图的独立集对应其补图的团
- 点覆盖：若图中所有边都至少有一个端点在点集 P 中，则称 P 是一个点覆盖，最小点覆盖：点数最少的覆盖集
- 定理：|最小点覆盖| + |最大独立集| = 图中点数
- 证明：考虑将最小点覆盖删去，得到一个独立集；将最大独立集删掉，得到一个点覆盖

基础定义

- 连通性：设 $G=(V,E), u, v \in V$，若 G 中存在一条长度为 n 的路径 A 使得 $A_1 = u, A_n = v$ 则称 u, v 连通
- 独立集：图中两两之间无边的点集
- 最大独立集：含点数最多的独立集，极大独立集：不存在点数更多的独立集是自身超集的独立集。这两种集合都可能有多个
- 团：图中两两之间都有边的点集，类似的有最大团、极大团。显然一个图的独立集对应其补图的团
- 点覆盖：若图中所有边都至少有一个端点在点集 P 中，则称 P 是一个点覆盖，最小点覆盖：点数最少的覆盖集
- 定理：|最小点覆盖| + |最大独立集| = 图中点数
- 证明：考虑将最小点覆盖删去，得到一个独立集；将最大独立集删掉，得到一个点覆盖
- 支配集：对于点集 P，若 $V-P$ 中的任意点都有边与 P 中的点相连则称 P 是一个支配集

基础定义

基础定义

- 拓扑图 (有向无环图，DAG)：不存在任何环的连通 (有向) 图

基础定义

- 拓扑图 (有向无环图，DAG)：不存在任何环的连通 (有向) 图
- 最短路：若边带边权，令路径长度为路径中经过的边权之和，两点之间长度最短的路径称为最短路

基础定义

- 拓扑图 (有向无环图，DAG)：不存在任何环的连通 (有向) 图
- 最短路：若边带边权，令路径长度为路径中经过的边权之和，两点之间长度最短的路径称为最短路
- 最短路径图：固定起点 s，求出 s 到其他所有点的所有最短路径，将不在最短路径中出现的边删掉，得到的图就是以 s 为起点的最短路径图，若边权均为正，则它还是一个 DAG

基础定义

- 拓扑图 (有向无环图，DAG)：不存在任何环的连通 (有向) 图
- 最短路：若边带边权，令路径长度为路径中经过的边权之和，两点之间长度最短的路径称为最短路
- 最短路径图：固定起点 s，求出 s 到其他所有点的所有最短路径，将不在最短路径中出现的边删掉，得到的图就是以 s 为起点的最短路径图，若边权均为正，则它还是一个 DAG
- 树：若 $G = (V, E)$ 中任意两点间都连通，且 $|E|$ 最小，则 G 称为树，显然 $|E| = |V| - 1$

基础定义

- 拓扑图 (有向无环图，DAG)：不存在任何环的连通 (有向) 图
- 最短路：若边带边权，令路径长度为路径中经过的边权之和，两点之间长度最短的路径称为最短路
- 最短路径图：固定起点 s，求出 s 到其他所有点的所有最短路径，将不在最短路径中出现的边删掉，得到的图就是以 s 为起点的最短路径图，若边权均为正，则它还是一个 DAG
- 树：若 $G = (V, E)$ 中任意两点间都连通，且 $|E|$ 最小，则 G 称为树，显然 $|E| = |V| - 1$
- 生成树：若 $G = (V, E)$ 的生成子图 $G' = (V, E')$ 是一棵树，则称 G' 是 G 的一棵生成树

基础定义

- 拓扑图 (有向无环图，DAG)：不存在任何环的连通 (有向) 图
- 最短路：若边带边权，令路径长度为路径中经过的边权之和，两点之间长度最短的路径称为最短路
- 最短路径图：固定起点 s，求出 s 到其他所有点的所有最短路径，将不在最短路径中出现的边删掉，得到的图就是以 s 为起点的最短路径图，若边权均为正，则它还是一个 DAG
- 树：若 $G = (V, E)$ 中任意两点间都连通，且 $|E|$ 最小，则 G 称为树，显然 $|E| = |V| - 1$
- 生成树：若 $G = (V, E)$ 的生成子图 $G' = (V, E')$ 是一棵树，则称 G' 是 G 的一棵生成树
- 竞赛图：任意两点之间都有且仅有一条边相连的有向图

基础定义

- 拓扑图 (有向无环图，DAG)：不存在任何环的连通 (有向) 图
- 最短路：若边带边权，令路径长度为路径中经过的边权之和，两点之间长度最短的路径称为最短路
- 最短路径图：固定起点 s，求出 s 到其他所有点的所有最短路径，将不在最短路径中出现的边删掉，得到的图就是以 s 为起点的最短路径图，若边权均为正，则它还是一个 DAG
- 树：若 $G = (V, E)$ 中任意两点间都连通，且 $|E|$ 最小，则 G 称为树，显然 $|E| = |V| - 1$
- 生成树：若 $G = (V, E)$ 的生成子图 $G' = (V, E')$ 是一棵树，则称 G' 是 G 的一棵生成树
- 竞赛图：任意两点之间都有且仅有一条边相连的有向图
- 欧拉路径 (回路)：经过所有边恰好一次的路径 (回路)

基础定义

- 拓扑图 (有向无环图，DAG)：不存在任何环的连通 (有向) 图
- 最短路：若边带边权，令路径长度为路径中经过的边权之和，两点之间长度最短的路径称为最短路
- 最短路径图：固定起点 s，求出 s 到其他所有点的所有最短路径，将不在最短路径中出现的边删掉，得到的图就是以 s 为起点的最短路径图，若边权均为正，则它还是一个 DAG
- 树：若 $G = (V, E)$ 中任意两点间都连通，且 $|E|$ 最小，则 G 称为树，显然 $|E| = |V| - 1$
- 生成树：若 $G = (V, E)$ 的生成子图 $G' = (V, E')$ 是一棵树，则称 G' 是 G 的一棵生成树
- 竞赛图：任意两点之间都有且仅有一条边相连的有向图
- 欧拉路径 (回路)：经过所有边恰好一次的路径 (回路)
- 哈密顿路径 (回路)：经过每个点恰好一次的路径 (回路)

基础定义

基础定义

- 二分图：若对于图 $G = (V, E)$ 存在一个 V 的划分 (A, B) 使得任意一条边的两个端点不属于同一个集合，则 G 是一个二分图

基础定义

- 二分图：若对于图 $G = (V, E)$ 存在一个 V 的划分 (A, B) 使得任意一条边的两个端点不属于同一个集合，则 G 是一个二分图
- 匹配：在图 $G = (V, E)$ 中，边集 $E' \subseteq E$ 被称为 G 的一个匹配当且仅当对于 V 中的每个点，E' 中与其关联的边不超过一条

基础定义

- 二分图：若对于图 $G = (V, E)$ 存在一个 V 的划分 (A, B) 使得任意一条边的两个端点不属于同一个集合，则 G 是一个二分图
- 匹配：在图 $G = (V, E)$ 中，边集 $E' \subseteq E$ 被称为 G 的一个匹配当且仅当对于 V 中的每个点，E' 中与其关联的边不超过一条
- 最大匹配：边数最多的匹配

基础定义

- 二分图：若对于图 $G = (V, E)$ 存在一个 V 的划分 (A, B) 使得任意一条边的两个端点不属于同一个集合，则 G 是一个二分图
- 匹配：在图 $G = (V, E)$ 中，边集 $E' \subseteq E$ 被称为 G 的一个匹配当且仅当对于 V 中的每个点，E' 中与其关联的边不超过一条
- 最大匹配：边数最多的匹配
- 完备匹配：使图中每个顶点都和匹配中某条边相关联的匹配

基础定义

- 二分图：若对于图 $G = (V, E)$ 存在一个 V 的划分 (A, B) 使得任意一条边的两个端点不属于同一个集合，则 G 是一个二分图
- 匹配：在图 $G = (V, E)$ 中，边集 $E' \subseteq E$ 被称为 G 的一个匹配当且仅当对于 V 中的每个点，E' 中与其关联的边不超过一条
- 最大匹配：边数最多的匹配
- 完备匹配：使图中每个顶点都和匹配中某条边相关联的匹配
- 定理：任意一个二分图的最小点覆盖等于其最大匹配数

基础定义

- 二分图：若对于图 $G = (V, E)$ 存在一个 V 的划分 (A, B) 使得任意一条边的两个端点不属于同一个集合，则 G 是一个二分图
- 匹配：在图 $G = (V, E)$ 中，边集 $E' \subseteq E$ 被称为 G 的一个匹配当且仅当对于 V 中的每个点，E' 中与其关联的边不超过一条
- 最大匹配：边数最多的匹配
- 完备匹配：使图中每个顶点都和匹配中某条边相关联的匹配
- 定理：任意一个二分图的最小点覆盖等于其最大匹配数
- Hall 定理：一个二分图 $G = (A \cup B, E)$ 具有完备匹配当且仅当：对于任意 $X \subseteq A, |X| = k$，至少存在 k 个 B 中的点与 X 中的点相连

基础定义

- 二分图：若对于图 $G=(V,E)$ 存在一个 V 的划分 (A,B) 使得任意一条边的两个端点不属于同一个集合，则 G 是一个二分图
- 匹配：在图 $G=(V,E)$ 中，边集 $E'\subseteq E$ 被称为 G 的一个匹配当且仅当对于 V 中的每个点，E' 中与其关联的边不超过一条
- 最大匹配：边数最多的匹配
- 完备匹配：使图中每个顶点都和匹配中某条边相关联的匹配
- 定理：任意一个二分图的最小点覆盖等于其最大匹配数
- Hall 定理：一个二分图 $G=(A\cup B,E)$ 具有完备匹配当且仅当：对于任意 $X\subseteq A,|X|=k$，至少存在 k 个 B 中的点与 X 中的点相连
- 邻接矩阵：存图方式，用二维数组记录所有点对间的边

基础定义

- 二分图：若对于图 $G=(V,E)$ 存在一个 V 的划分 (A,B) 使得任意一条边的两个端点不属于同一个集合，则 G 是一个二分图
- 匹配：在图 $G=(V,E)$ 中，边集 $E'\subseteq E$ 被称为 G 的一个匹配当且仅当对于 V 中的每个点，E' 中与其关联的边不超过一条
- 最大匹配：边数最多的匹配
- 完备匹配：使图中每个顶点都和匹配中某条边相关联的匹配
- 定理：任意一个二分图的最小点覆盖等于其最大匹配数
- Hall 定理：一个二分图 $G=(A\cup B,E)$ 具有完备匹配当且仅当：对于任意 $X\subseteq A,|X|=k$，至少存在 k 个 B 中的点与 X 中的点相连
- 邻接矩阵：存图方式，用二维数组记录所有点对间的边
- 邻接表：存图方式，将所有边按照起点分类，每一类边都利用链表存储，遍历时访问当前所在点的边链表

基础定义

基础定义

- 强连通：若对于图 G 中任意两点 u, v 都存在 $u \to v$ 和 $v \to u$ 的路径，则称 G 是一个强连通图

基础定义

- 强连通：若对于图 G 中任意两点 u, v 都存在 $u \to v$ 和 $v \to u$ 的路径，则称 G 是一个强连通图
- 强连通分量：有向图的极大强连通子图

基础定义

- 强连通：若对于图 G 中任意两点 u, v 都存在 $u \to v$ 和 $v \to u$ 的路径，则称 G 是一个强连通图
- 强连通分量：有向图的极大强连通子图
- 割点：无向连通图 $G = (V, E)$ 中，对于点 $x \in V$ 若删去 x 后 G 不连通，称 x 是 G 的割点

基础定义

- 强连通：若对于图 G 中任意两点 u, v 都存在 $u \to v$ 和 $v \to u$ 的路径，则称 G 是一个强连通图
- 强连通分量：有向图的极大强连通子图
- 割点：无向连通图 $G = (V, E)$ 中，对于点 $x \in V$ 若删去 x 后 G 不连通，称 x 是 G 的割点
- 桥：无向连通图 $G = (V, E)$ 中，对于边 $e \in E$ 若删去 e 后 G 不连通，称 e 是 G 的桥边

基础定义

- 强连通：若对于图 G 中任意两点 u, v 都存在 $u \to v$ 和 $v \to u$ 的路径，则称 G 是一个强连通图
- 强连通分量：有向图的极大强连通子图
- 割点：无向连通图 $G = (V, E)$ 中，对于点 $x \in V$ 若删去 x 后 G 不连通，称 x 是 G 的割点
- 桥：无向连通图 $G = (V, E)$ 中，对于边 $e \in E$ 若删去 e 后 G 不连通，称 e 是 G 的桥边
- 点双连通图：任意两点间都存在两条除起始点外互不相交的路径的图

基础定义

- 强连通：若对于图 G 中任意两点 u, v 都存在 $u \to v$ 和 $v \to u$ 的路径，则称 G 是一个强连通图
- 强连通分量：有向图的极大强连通子图
- 割点：无向连通图 $G = (V, E)$ 中，对于点 $x \in V$ 若删去 x 后 G 不连通，称 x 是 G 的割点
- 桥：无向连通图 $G = (V, E)$ 中，对于边 $e \in E$ 若删去 e 后 G 不连通，称 e 是 G 的桥边
- 点双连通图：任意两点间都存在两条除起始点外互不相交的路径的图
- 类似的有边双连通图、点双连通分量、边双连通分量

基础定义

- 强连通：若对于图 G 中任意两点 u, v 都存在 $u \to v$ 和 $v \to u$ 的路径，则称 G 是一个强连通图
- 强连通分量：有向图的极大强连通子图
- 割点：无向连通图 $G = (V, E)$ 中，对于点 $x \in V$ 若删去 x 后 G 不连通，称 x 是 G 的割点
- 桥：无向连通图 $G = (V, E)$ 中，对于边 $e \in E$ 若删去 e 后 G 不连通，称 e 是 G 的桥边
- 点双连通图：任意两点间都存在两条除起始点外互不相交的路径的图
- 类似的有边双连通图、点双连通分量、边双连通分量
- 强连通分量之间交集一定为空，点双连通分量间可能有交，交点是原图中的割点

最短路

最短路

- 常用有三种方法：

最短路

- 常用有三种方法：
- Floyd 算法，时间复杂度 $O(n^3)$，图中不能有负环，能求出任意两点间最短路

最短路

- 常用有三种方法：
- Floyd 算法，时间复杂度 $O(n^3)$，图中不能有负环，能求出任意两点间最短路
- Dijkstra 算法：时间复杂度 $O(n^2)$，可利用数据结构优化至 $O(n \log n)$，图中不能有负权边，适用于求单源最短路

最短路

- 常用有三种方法：
- Floyd 算法，时间复杂度 $O(n^3)$，图中不能有负环，能求出任意两点间最短路
- Dijkstra 算法：时间复杂度 $O(n^2)$，可利用数据结构优化至 $O(n \log n)$，图中不能有负权边，适用于求单源最短路
- Bellman-Ford 算法：$O(nm)$，适用于求单源最短路，能用来判负环

Floyd

Floyd

- Floyd 算法的思想是动态规划，$dp[k][i][j]$ 表示 i 到 j 且中间结点标号不超过 k 的最短路径，$dp[k][i][j] = \min(dp[k-1][i][j], dp[k-1][i][k] + dp[k-1][k][j])$

Floyd

- Floyd 算法的思想是动态规划，$dp[k][i][j]$ 表示 i 到 j 且中间结点标号不超过 k 的最短路径，$dp[k][i][j] = \min(dp[k-1][i][j], dp[k-1][i][k] + dp[k-1][k][j])$
- 第一维可以滚动数组优化掉，空间 $O(n^2)$

Floyd

- Floyd 算法的思想是动态规划，$dp[k][i][j]$ 表示 i 到 j 且中间结点标号不超过 k 的最短路径，$dp[k][i][j] = \min(dp[k-1][i][j], dp[k-1][i][k] + dp[k-1][k][j])$
- 第一维可以滚动数组优化掉，空间 $O(n^2)$
- 枚举 k，枚举 i，枚举 j，$dis[i][j] = \min(dis[i][j], dis[i][k] + dis[k][j])$

Floyd

- Floyd 算法的思想是动态规划，$dp[k][i][j]$ 表示 i 到 j 且中间结点标号不超过 k 的最短路径，$dp[k][i][j] = \min(dp[k-1][i][j], dp[k-1][i][k] + dp[k-1][k][j])$
- 第一维可以滚动数组优化掉，空间 $O(n^2)$
- 枚举 k，枚举 i，枚举 j，$dis[i][j] = \min(dis[i][j], dis[i][k] + dis[k][j])$
- 算法正确性可以这样理解：首先最短路径中任意一段连续子路径仍然是最短路径

Floyd

- Floyd 算法的思想是动态规划，$dp[k][i][j]$ 表示 i 到 j 且中间结点标号不超过 k 的最短路径，$dp[k][i][j] = \min(dp[k-1][i][j], dp[k-1][i][k] + dp[k-1][k][j])$
- 第一维可以滚动数组优化掉，空间 $O(n^2)$
- 枚举 k，枚举 i，枚举 j，$dis[i][j] = \min(dis[i][j], dis[i][k] + dis[k][j])$
- 算法正确性可以这样理解：首先最短路径中任意一段连续子路径仍然是最短路径
- 考虑 s 到 t 的最短路径，路上经过的顶点序列是 P_i，考虑 P_i 中标号最小的点 P_x，则中间点枚举到 P_x 时，点对 $P_{x-1} \to P_{x+1}$ 的最短路将会被更新，随后将 P_x 从 P 中删去

Floyd

- Floyd 算法的思想是动态规划，$dp[k][i][j]$ 表示 i 到 j 且中间结点标号不超过 k 的最短路径，$dp[k][i][j] = \min(dp[k-1][i][j], dp[k-1][i][k] + dp[k-1][k][j])$
- 第一维可以滚动数组优化掉，空间 $O(n^2)$
- 枚举 k，枚举 i，枚举 j，$dis[i][j] = \min(dis[i][j], dis[i][k] + dis[k][j])$
- 算法正确性可以这样理解：首先最短路径中任意一段连续子路径仍然是最短路径
- 考虑 s 到 t 的最短路径，路上经过的顶点序列是 P_i，考虑 P_i 中标号最小的点 P_x，则中间点枚举到 P_x 时，点对 $P_{x-1} \to P_{x+1}$ 的最短路将会被更新，随后将 P_x 从 P 中删去
- 反复进行上述操作，则任意一条最短路径一定能被正确更新出来

Floyd

- Floyd 算法的思想是动态规划，$dp[k][i][j]$ 表示 i 到 j 且中间结点标号不超过 k 的最短路径，$dp[k][i][j] = \min(dp[k-1][i][j], dp[k-1][i][k] + dp[k-1][k][j])$
- 第一维可以滚动数组优化掉，空间 $O(n^2)$
- 枚举 k，枚举 i，枚举 j，$dis[i][j] = \min(dis[i][j], dis[i][k] + dis[k][j])$
- 算法正确性可以这样理解：首先最短路径中任意一段连续子路径仍然是最短路径
- 考虑 s 到 t 的最短路径，路上经过的顶点序列是 P_i，考虑 P_i 中标号最小的点 P_x，则中间点枚举到 P_x 时，点对 $P_{x-1} \to P_{x+1}$ 的最短路将会被更新，随后将 P_x 从 P 中删去
- 反复进行上述操作，则任意一条最短路径一定能被正确更新出来
- 上述这个过程也说明中间点 k 枚举的顺序并不重要

Dijkstra

Dijkstra

算法

1. 起始点为 s，初始时已选结点集合 $V = \varnothing$

Dijkstra

算法

1. 起始点为 s，初始时已选结点集合 $V = \varnothing$
2. 初始化距离标号，$d_s = 0$，其他点 $d_i = \infty$

Dijkstra

算法

1. 起始点为 s，初始时已选结点集合 $V = \varnothing$
2. 初始化距离标号，$d_s = 0$，其他点 $d_i = \infty$
3. 找到不在 V 中且 d 值最小的结点 u,$V = V + \{u\}$

Dijkstra

算法

1. 起始点为 s，初始时已选结点集合 $V = \varnothing$
2. 初始化距离标号，$d_s = 0$，其他点 $d_i = \infty$
3. 找到不在 V 中且 d 值最小的结点 u,$V = V + \{u\}$
4. 对于还不在 V 中的点 v，更新 $d_v = \min(d_v, d_u + w(u, v))$

Dijkstra

算法

1. 起始点为 s，初始时已选结点集合 $V = \varnothing$
2. 初始化距离标号，$d_s = 0$，其他点 $d_i = \infty$
3. 找到不在 V 中且 d 值最小的结点 $u, V = V + \{u\}$
4. 对于还不在 V 中的点 v，更新 $d_v = \min(d_v, d_u + w(u, v))$
5. 若 V 包含所有点则算法结束，d_u 即为 s 到 u 的最短距离。否则返回第 3 步

Dijkstra

算法

1. 起始点为 s，初始时已选结点集合 $V = \varnothing$
2. 初始化距离标号，$d_s = 0$，其他点 $d_i = \infty$
3. 找到不在 V 中且 d 值最小的结点 $u, V = V + \{u\}$
4. 对于还不在 V 中的点 v，更新 $d_v = \min(d_v, d_u + w(u, v))$
5. 若 V 包含所有点则算法结束，d_u 即为 s 到 u 的最短距离。否则返回第 3 步

- 边权 w 应非负

Dijkstra

算法

1. 起始点为 s，初始时已选结点集合 $V = \varnothing$
2. 初始化距离标号，$d_s = 0$，其他点 $d_i = \infty$
3. 找到不在 V 中且 d 值最小的结点 $u, V = V + \{u\}$
4. 对于还不在 V 中的点 v，更新 $d_v = \min(d_v, d_u + w(u, v))$
5. 若 V 包含所有点则算法结束，d_u 即为 s 到 u 的最短距离。否则返回第 3 步

- 边权 w 应非负
- 利用优先队列加速

Dijkstra

算法

1. 起始点为 s，初始时已选结点集合 $V = \varnothing$
2. 初始化距离标号，$d_s = 0$，其他点 $d_i = \infty$
3. 找到不在 V 中且 d 值最小的结点 $u, V = V + \{u\}$
4. 对于还不在 V 中的点 v，更新 $d_v = \min(d_v, d_u + w(u, v))$
5. 若 V 包含所有点则算法结束，d_u 即为 s 到 u 的最短距离。否则返回第 3 步

- 边权 w 应非负
- 利用优先队列加速
- $O(n \log m)$

Dijkstra

- 考虑证明：

Dijkstra

- 考虑证明：
- 考虑归纳证明结点 u 进入 V 时，d_u 就是到 u 的最短路

Dijkstra

- 考虑证明：
- 考虑归纳证明结点 u 进入 V 时，d_u 就是到 u 的最短路
- 初始 $d_s = 0$ 满足条件

Dijkstra

- 考虑证明：
- 考虑归纳证明结点 u 进入 V 时，d_u 就是到 u 的最短路
- 初始 $d_s = 0$ 满足条件
- 假设之前集合 V 中的点全部满足，现在加入结点 u，考虑任意一条从 s 到 u 的路径 $P = (s \to \cdots \to x \to y \to \cdots \to u)$，其中 $s \to x$ 均在 V 中，$y \to u$ 均不在 V 中

Dijkstra

- 考虑证明:
- 考虑归纳证明结点 u 进入 V 时，d_u 就是到 u 的最短路
- 初始 $d_s = 0$ 满足条件
- 假设之前集合 V 中的点全部满足，现在加入结点 u，考虑任意一条从 s 到 u 的路径 $P = (s \to \cdots \to x \to y \to \cdots \to u)$, 其中 $s \to x$ 均在 V 中，$y \to u$ 均不在 V 中
- 根据归纳，d_x 为到 x 的最短路

Dijkstra

- 考虑证明:
- 考虑归纳证明结点 u 进入 V 时，d_u 就是到 u 的最短路
- 初始 $d_s = 0$ 满足条件
- 假设之前集合 V 中的点全部满足，现在加入结点 u，考虑任意一条从 s 到 u 的路径 $P = (s \to \cdots \to x \to y \to \cdots \to u)$, 其中 $s \to x$ 均在 V 中，$y \to u$ 均不在 V 中
- 根据归纳，d_x 为到 x 的最短路
- $d_x + w(x, y) \geq d_y$

Dijkstra

- 考虑证明:
- 考虑归纳证明结点 u 进入 V 时，d_u 就是到 u 的最短路
- 初始 $d_s = 0$ 满足条件
- 假设之前集合 V 中的点全部满足，现在加入结点 u，考虑任意一条从 s 到 u 的路径 $P = (s \to \cdots \to x \to y \to \cdots \to u)$, 其中 $s \to x$ 均在 V 中，$y \to u$ 均不在 V 中
- 根据归纳，d_x 为到 x 的最短路
- $d_x + w(x, y) \geq d_y$
- 由于算法选择最小的 d 加入 V，因此 $d_y \geq d_u$

Dijkstra

- 考虑证明：
- 考虑归纳证明结点 u 进入 V 时，d_u 就是到 u 的最短路
- 初始 $d_s = 0$ 满足条件
- 假设之前集合 V 中的点全部满足，现在加入结点 u，考虑任意一条从 s 到 u 的路径 $P = (s \to \cdots \to x \to y \to \cdots \to u)$，其中 $s \to x$ 均在 V 中，$y \to u$ 均不在 V 中
- 根据归纳，d_x 为到 x 的最短路
- $d_x + w(x, y) \geq d_y$
- 由于算法选择最小的 d 加入 V，因此 $d_y \geq d_u$
- $length(P) = length(s \to x) + w(x, y) + length(y \to u) \geq d_x + w(x, y) + length(y \to u) \geq d_y + length(y \to u) \geq d_y \geq d_u$

Dijkstra

- 考虑证明：
- 考虑归纳证明结点 u 进入 V 时，d_u 就是到 u 的最短路
- 初始 $d_s = 0$ 满足条件
- 假设之前集合 V 中的点全部满足，现在加入结点 u，考虑任意一条从 s 到 u 的路径 $P = (s \to \cdots \to x \to y \to \cdots \to u)$，其中 $s \to x$ 均在 V 中，$y \to u$ 均不在 V 中
- 根据归纳，d_x 为到 x 的最短路
- $d_x + w(x, y) \geq d_y$
- 由于算法选择最小的 d 加入 V，因此 $d_y \geq d_u$
- $length(P) = length(s \to x) + w(x, y) + length(y \to u) \geq d_x + w(x, y) + length(y \to u) \geq d_y + length(y \to u) \geq d_y \geq d_u$
- 因此 d_u 也是最短路

Bellman-Ford

Bellman-Ford

- 它的思想也是动态规划，$dp[j][i]$ 表示从 S 到 i 经过不超过 j 条边的最短路径，第一维滚动掉空间是 $O(n)$

Bellman-Ford

- 它的思想也是动态规划，$dp[j][i]$ 表示从 S 到 i 经过不超过 j 条边的最短路径，第一维滚动掉空间是 $O(n)$
- Floyd 是利用点进行松弛，而 Bellman-Ford 是利用边进行松弛

Bellman-Ford

- 它的思想也是动态规划，$dp[j][i]$ 表示从 S 到 i 经过不超过 j 条边的最短路径，第一维滚动掉空间是 $O(n)$
- Floyd 是利用点进行松弛，而 Bellman-Ford 是利用边进行松弛
- 枚举图中每条边，对于边 $e = (u, v)$，进行更新：
  $dis[v] = \min(dis[v], dis[u] + len[e])$

Bellman-Ford

- 它的思想也是动态规划，$dp[j][i]$ 表示从 S 到 i 经过不超过 j 条边的最短路径，第一维滚动掉空间是 $O(n)$
- Floyd 是利用点进行松弛，而 Bellman-Ford 是利用边进行松弛
- 枚举图中每条边，对于边 $e = (u, v)$，进行更新：
  $dis[v] = \min(dis[v], dis[u] + len[e])$
- 上述枚举进行 $n - 1$ 次即可，按顺序考虑最短路上经过的边

Bellman-Ford

- 它的思想也是动态规划，$dp[j][i]$ 表示从 S 到 i 经过不超过 j 条边的最短路径，第一维滚动掉空间是 $O(n)$
- Floyd 是利用点进行松弛，而 Bellman-Ford 是利用边进行松弛
- 枚举图中每条边，对于边 $e = (u, v)$，进行更新：
  $dis[v] = \min(dis[v], dis[u] + len[e])$
- 上述枚举进行 $n - 1$ 次即可，按顺序考虑最短路上经过的边
- 任意一条最短路包含的边数不超过 $n - 1$，所以进行 $n - 1$ 次枚举一定能把最短路从起点更新到终点

Bellman-Ford

- 它的思想也是动态规划，$dp[j][i]$ 表示从 S 到 i 经过不超过 j 条边的最短路径，第一维滚动掉空间是 $O(n)$
- Floyd 是利用点进行松弛，而 Bellman-Ford 是利用边进行松弛
- 枚举图中每条边，对于边 $e = (u, v)$，进行更新：
  $dis[v] = \min(dis[v], dis[u] + len[e])$
- 上述枚举进行 $n - 1$ 次即可，按顺序考虑最短路上经过的边
- 任意一条最短路包含的边数不超过 $n - 1$，所以进行 $n - 1$ 次枚举一定能把最短路从起点更新到终点
- 考虑它的优化：1. 若上一轮松弛中 dis 改变，才枚举以它为起点的边进行新一轮松弛；2. 将 dis 改变的点放入一个队列中，依次取出并枚举出边进行松弛操作

Bellman-Ford

- 它的思想也是动态规划，$dp[j][i]$ 表示从 S 到 i 经过不超过 j 条边的最短路径，第一维滚动掉空间是 $O(n)$
- Floyd 是利用点进行松弛，而 Bellman-Ford 是利用边进行松弛
- 枚举图中每条边，对于边 $e = (u, v)$，进行更新：
  $dis[v] = \min(dis[v], dis[u] + len[e])$
- 上述枚举进行 $n - 1$ 次即可，按顺序考虑最短路上经过的边
- 任意一条最短路包含的边数不超过 $n - 1$，所以进行 $n - 1$ 次枚举一定能把最短路从起点更新到终点
- 考虑它的优化：1. 若上一轮松弛中 dis 改变，才枚举以它为起点的边进行新一轮松弛；2. 将 dis 改变的点放入一个队列中，依次取出并枚举出边进行松弛操作
- 优化后的算法就是常说的 SPFA 算法，它的理论复杂度仍然是 $O(nm)$，但在随机数据下往往能跑出 $O(n + m)$ 的效果

Bellman-Ford

- 它的思想也是动态规划，$dp[j][i]$ 表示从 S 到 i 经过不超过 j 条边的最短路径，第一维滚动掉空间是 $O(n)$
- Floyd 是利用点进行松弛，而 Bellman-Ford 是利用边进行松弛
- 枚举图中每条边，对于边 $e = (u, v)$，进行更新：
  $dis[v] = \min(dis[v], dis[u] + len[e])$
- 上述枚举进行 $n - 1$ 次即可，按顺序考虑最短路上经过的边
- 任意一条最短路包含的边数不超过 $n - 1$，所以进行 $n - 1$ 次枚举一定能把最短路从起点更新到终点
- 考虑它的优化：1. 若上一轮松弛中 dis 改变，才枚举以它为起点的边进行新一轮松弛；2. 将 dis 改变的点放入一个队列中，依次取出并枚举出边进行松弛操作
- 优化后的算法就是常说的 SPFA 算法，它的理论复杂度仍然是 $O(nm)$，但在随机数据下往往能跑出 $O(n + m)$ 的效果
- 判断负环：第 n 次循环时 Bellman-Ford 算法仍能成功松弛；SPFA 中某个点入队次数超过 n 次

差分约束

差分约束

- 差分约束是一类特殊的线性规划问题，它只有一种约束条件：$p_x \leq p_y + c$，要求最大化 p_n

差分约束

- 差分约束是一类特殊的线性规划问题，它只有一种约束条件：$p_x \leq p_y + c$，要求最大化 p_n
- 在最短路中有一个和约束式子非常相似的性质：求得某源点的单源最短路后，对于任意一条边 $e = (u, v)$，一定有 $dis_v \leq dis_u + len_e$

差分约束

- 差分约束是一类特殊的线性规划问题，它只有一种约束条件：$p_x \leq p_y + c$，要求最大化 p_n
- 在最短路中有一个和约束式子非常相似的性质：求得某源点的单源最短路后，对于任意一条边 $e = (u, v)$，一定有 $dis_v \leq dis_u + len_e$
- 由于这个性质差分约束可以转化成最短路问题

差分约束

- 差分约束是一类特殊的线性规划问题，它只有一种约束条件：$p_x \leq p_y + c$，要求最大化 p_n
- 在最短路中有一个和约束式子非常相似的性质：求得某源点的单源最短路后，对于任意一条边 $e = (u, v)$，一定有 $dis_v \leq dis_u + len_e$
- 由于这个性质差分约束可以转化成最短路问题
- 假设差分约束中有 n 个变量，将每个变量看做点，对于每个限制连一条 $y \to x$ 的长为 c 的有向边

差分约束

- 差分约束是一类特殊的线性规划问题，它只有一种约束条件：$p_x \leq p_y + c$，要求最大化 p_n
- 在最短路中有一个和约束式子非常相似的性质：求得某源点的单源最短路后，对于任意一条边 $e = (u, v)$，一定有 $dis_v \leq dis_u + len_e$
- 由于这个性质差分约束可以转化成最短路问题
- 假设差分约束中有 n 个变量，将每个变量看做点，对于每个限制连一条 $y \to x$ 的长为 c 的有向边
- 建立一个超级源 S 从它向图中每个点连一条权为 0 的有向边

差分约束

- 差分约束是一类特殊的线性规划问题，它只有一种约束条件：$p_x \leq p_y + c$，要求最大化 p_n
- 在最短路中有一个和约束式子非常相似的性质：求得某源点的单源最短路后，对于任意一条边 $e = (u, v)$，一定有 $dis_v \leq dis_u + len_e$
- 由于这个性质差分约束可以转化成最短路问题
- 假设差分约束中有 n 个变量，将每个变量看做点，对于每个限制连一条 $y \to x$ 的长为 c 的有向边
- 建立一个超级源 S 从它向图中每个点连一条权为 0 的有向边
- 以 S 为源求图的最短路，若图中有负环则该问题无解；否则 dis_i 就是一组符合条件的 p_i

差分约束

差分约束

- 由于可能有负环，所以要用 Bellman-Ford 求解

差分约束

- 由于可能有负环，所以要用 Bellman-Ford 求解
- 添加超级源相当于规定所有变量值不超过 d_S 的初始值

差分约束

- 由于可能有负环，所以要用 Bellman-Ford 求解
- 添加超级源相当于规定所有变量值不超过 d_S 的初始值
- 若题目中已经规定了起点，比如最大化的是 $p_n - p_1$，那么 1 就是起点不需要添加额外的 S

差分约束

- 由于可能有负环，所以要用 Bellman-Ford 求解
- 添加超级源相当于规定所有变量值不超过 d_S 的初始值
- 若题目中已经规定了起点，比如最大化的是 $p_n - p_1$，那么 1 就是起点不需要添加额外的 S
- 实际上若起点是 s，则求出的解满足 $dis_i - dis_s$ 最大

最小生成树

最小生成树

- 求解 MST 常用的有两种算法:

最小生成树

- 求解 MST 常用的有两种算法：
- Prim 算法，时间复杂度 $O(n^2)$

最小生成树

- 求解 MST 常用的有两种算法：
- Prim 算法，时间复杂度 $O(n^2)$
- Kruskal 算法：时间复杂度 $O(m \log m)$

最小生成树

- 求解 MST 常用的有两种算法：
- Prim 算法，时间复杂度 $O(n^2)$
- Kruskal 算法：时间复杂度 $O(m \log m)$
- 适用情况由时间复杂度可得

Prim

Prim

算法

1. 初始选择边集 $E = \varnothing$，点集 $V = \{$任意一个结点$\}$

Prim

算法

1. 初始选择边集 $E = \varnothing$，点集 $V = \{$任意一个结点$\}$
2. 选择一条权值 w 最小的边 $e = (u, v)$，满足 $u \in V, v \notin V$

Prim

算法

1. 初始选择边集 $E = \varnothing$，点集 $V = \{任意一个结点\}$
2. 选择一条权值 w 最小的边 $e = (u, v)$，满足 $u \in V, v \notin V$
3. $E = E + \{e\}$, $V = V + \{v\}$

Prim

算法

1. 初始选择边集 $E = \varnothing$，点集 $V = \{\text{任意一个结点}\}$
2. 选择一条权值 w 最小的边 $e = (u, v)$，满足 $u \in V, v \notin V$
3. $E = E + \{e\}$, $V = V + \{v\}$
4. 点集 V 内若包含所有结点则算法结束，否则返回第二步

Prim

算法

1. 初始选择边集 $E = \varnothing$，点集 $V = \{$任意一个结点$\}$
2. 选择一条权值 w 最小的边 $e = (u, v)$，满足 $u \in V, v \notin V$
3. $E = E + \{e\}$, $V = V + \{v\}$
4. 点集 V 内若包含所有结点则算法结束，否则返回第二步

- 使用优先队列加速过程 2

Prim

算法

1. 初始选择边集 $E = \varnothing$，点集 $V = \{$任意一个结点$\}$
2. 选择一条权值 w 最小的边 $e = (u, v)$，满足 $u \in V, v \notin V$
3. $E = E + \{e\}$, $V = V + \{v\}$
4. 点集 V 内若包含所有结点则算法结束，否则返回第二步

- 使用优先队列加速过程 2
- $O(n \log m)$

Prim

- 令 Prim 算法得到的树为 P，有一棵最小生成树 T，假设他们不同

Prim

- 令 Prim 算法得到的树为 P，有一棵最小生成树 T，假设他们不同
- 假设前 $k-1$ 步 P 选择的边都在 T 中，令此时的树为 P'

Prim

- 令 Prim 算法得到的树为 P，有一棵最小生成树 T，假设他们不同
- 假设前 $k-1$ 步 P 选择的边都在 T 中，令此时的树为 P'
- 第 k 步选择的 $e=(u,v)$ 不在 T 中，假设 u 在 P' 中，而 v 不在

Prim

- 令 Prim 算法得到的树为 P，有一棵最小生成树 T，假设他们不同
- 假设前 $k-1$ 步 P 选择的边都在 T 中，令此时的树为 P'
- 第 k 步选择的 $e=(u,v)$ 不在 T 中，假设 u 在 P' 中，而 v 不在
- T 中必有一条 $u \to v$ 的路径，路径上必有一条边 $e'=(x,y)$ 满足此时 x 在 P' 中而 y 不在

Prim

- 令 Prim 算法得到的树为 P，有一棵最小生成树 T，假设他们不同
- 假设前 $k-1$ 步 P 选择的边都在 T 中，令此时的树为 P'
- 第 k 步选择的 $e=(u,v)$ 不在 T 中，假设 u 在 P' 中，而 v 不在
- T 中必有一条 $u \to v$ 的路径，路径上必有一条边 $e'=(x,y)$ 满足此时 x 在 P' 中而 y 不在
- 若 $w(e') > w(e)$ 则在 T 中用 e 换掉 e' 可得到一个更小的生成树，矛盾

Prim

- 令 Prim 算法得到的树为 P，有一棵最小生成树 T，假设他们不同
- 假设前 $k-1$ 步 P 选择的边都在 T 中，令此时的树为 P'
- 第 k 步选择的 $e=(u,v)$ 不在 T 中，假设 u 在 P' 中，而 v 不在
- T 中必有一条 $u \to v$ 的路径，路径上必有一条边 $e'=(x,y)$ 满足此时 x 在 P' 中而 y 不在
- 若 $w(e') > w(e)$ 则在 T 中用 e 换掉 e' 可得到一个更小的生成树，矛盾
- 若 $w(e') < w(e)$ 则第 k 步时选的是 e' 而不是 e，矛盾

Prim

- 令 Prim 算法得到的树为 P，有一棵最小生成树 T，假设他们不同
- 假设前 $k-1$ 步 P 选择的边都在 T 中，令此时的树为 P'
- 第 k 步选择的 $e=(u,v)$ 不在 T 中，假设 u 在 P' 中，而 v 不在
- T 中必有一条 $u \to v$ 的路径，路径上必有一条边 $e'=(x,y)$ 满足此时 x 在 P' 中而 y 不在
- 若 $w(e') > w(e)$ 则在 T 中用 e 换掉 e' 可得到一个更小的生成树，矛盾
- 若 $w(e') < w(e)$ 则第 k 步时选的是 e' 而不是 e，矛盾
- 若 $w(e')=w(e)$，在 T 中用 e 换掉 e'，则 P 前 k 步中选择边都在 T 中

Prim

- 令 Prim 算法得到的树为 P，有一棵最小生成树 T，假设他们不同
- 假设前 $k-1$ 步 P 选择的边都在 T 中，令此时的树为 P'
- 第 k 步选择的 $e=(u,v)$ 不在 T 中，假设 u 在 P' 中，而 v 不在
- T 中必有一条 $u \to v$ 的路径，路径上必有一条边 $e'=(x,y)$ 满足此时 x 在 P' 中而 y 不在
- 若 $w(e') > w(e)$ 则在 T 中用 e 换掉 e' 可得到一个更小的生成树，矛盾
- 若 $w(e') < w(e)$ 则第 k 步时选的是 e' 而不是 e，矛盾
- 若 $w(e') = w(e)$，在 T 中用 e 换掉 e'，则 P 前 k 步中选择边都在 T 中
- 有限步后可把 T 变为 P 且权值不变，因此 P 就是最小生成树

Kruskal

Kruskal

算法

1. 将所有边按权值 $w(e)$ 的大小排序

Kruskal

算法

1. 将所有边按权值 $w(e)$ 的大小排序
2. 初始选择边集 $E = \varnothing$

Kruskal

算法

1. 将所有边按权值 $w(e)$ 的大小排序
2. 初始选择边集 $E = \varnothing$
3. 按顺序考虑每条边 e，e 与已在 E 中的边不构成环则可选择。$E = E + \{e\}$. 若构成环则放弃 e

Kruskal

算法

1. 将所有边按权值 $w(e)$ 的大小排序
2. 初始选择边集 $E = \varnothing$
3. 按顺序考虑每条边 e，e 与已在 E 中的边不构成环则可选择。$E = E + \{e\}$. 若构成环则放弃 e
4. 选出 $n-1$ 条边后 E 即为一棵最小生成树，否则原图不连通

Kruskal

算法

1. 将所有边按权值 $w(e)$ 的大小排序
2. 初始选择边集 $E = \varnothing$
3. 按顺序考虑每条边 e，e 与已在 E 中的边不构成环则可选择。$E = E + \{e\}$. 若构成环则放弃 e
4. 选出 $n-1$ 条边后 E 即为一棵最小生成树，否则原图不连通

- 使用并查集维护过程 3 中的选择

Kruskal

算法

1. 将所有边按权值 $w(e)$ 的大小排序
2. 初始选择边集 $E = \varnothing$
3. 按顺序考虑每条边 e，e 与已在 E 中的边不构成环则可选择。$E = E + \{e\}$. 若构成环则放弃 e
4. 选出 $n-1$ 条边后 E 即为一棵最小生成树，否则原图不连通

- 使用并查集维护过程 3 中的选择
- $O(m \log m + m \log n)$

Kruskal

- 令 Kruskal 算法得到的树为 K，有一棵最小生成树 T，假设他们不同

Kruskal

- 令 Kruskal 算法得到的树为 K，有一棵最小生成树 T，假设他们不同
- 找到边权最小的在 K 但不在 T 中的边 e

Kruskal

- 令 Kruskal 算法得到的树为 K，有一棵最小生成树 T，假设他们不同
- 找到边权最小的在 K 但不在 T 中的边 e
- 把 e 加入 T 中，形成一个环，删掉这个环中一条不在 K 中的边 e'，得到新生成树 T'

Kruskal

- 令 Kruskal 算法得到的树为 K，有一棵最小生成树 T，假设他们不同
- 找到边权最小的在 K 但不在 T 中的边 e
- 把 e 加入 T 中，形成一个环，删掉这个环中一条不在 K 中的边 e'，得到新生成树 T'
- 若不存在 e' 则 K 存在环，矛盾

Kruskal

- 令 Kruskal 算法得到的树为 K，有一棵最小生成树 T，假设他们不同
- 找到边权最小的在 K 但不在 T 中的边 e
- 把 e 加入 T 中，形成一个环，删掉这个环中一条不在 K 中的边 e'，得到新生成树 T'
- 若不存在 e' 则 K 存在环，矛盾
- 若 $w(e') > w(e)$，则 T' 权值和小于 T，矛盾

Kruskal

- 令 Kruskal 算法得到的树为 K，有一棵最小生成树 T，假设他们不同
- 找到边权最小的在 K 但不在 T 中的边 e
- 把 e 加入 T 中，形成一个环，删掉这个环中一条不在 K 中的边 e'，得到新生成树 T'
- 若不存在 e' 则 K 存在环，矛盾
- 若 $w(e') > w(e)$，则 T' 权值和小于 T，矛盾
- 若 $w(e') < w(e)$，则 Kruskal 执行时先考虑了 e'，由于成环没加入 e'，因此在 e' 之前加入的边权值均 $\leq w(e') < w(e)$. 由 e 的定义，K 中边权小于 $w(e)$ 的边均在 T 中，说明 T 中边与 e' 会成环，矛盾

Kruskal

- 令 Kruskal 算法得到的树为 K，有一棵最小生成树 T，假设他们不同
- 找到边权最小的在 K 但不在 T 中的边 e
- 把 e 加入 T 中，形成一个环，删掉这个环中一条不在 K 中的边 e'，得到新生成树 T'
- 若不存在 e' 则 K 存在环，矛盾
- 若 $w(e') > w(e)$，则 T' 权值和小于 T，矛盾
- 若 $w(e') < w(e)$，则 Kruskal 执行时先考虑了 e'，由于成环没加入 e'，因此在 e' 之前加入的边权值均 $\leq w(e') < w(e)$. 由 e 的定义，K 中边权小于 $w(e)$ 的边均在 T 中，说明 T 中边与 e' 会成环，矛盾
- $w(e') = w(e)$，在 T 中用 e 换掉 e'

Kruskal

- 令 Kruskal 算法得到的树为 K，有一棵最小生成树 T，假设他们不同
- 找到边权最小的在 K 但不在 T 中的边 e
- 把 e 加入 T 中，形成一个环，删掉这个环中一条不在 K 中的边 e'，得到新生成树 T'
- 若不存在 e' 则 K 存在环，矛盾
- 若 $w(e') > w(e)$，则 T' 权值和小于 T，矛盾
- 若 $w(e') < w(e)$，则 Kruskal 执行时先考虑了 e'，由于成环没加入 e'，因此在 e' 之前加入的边权值均 $\leq w(e') < w(e)$. 由 e 的定义，K 中边权小于 $w(e)$ 的边均在 T 中，说明 T 中边与 e' 会成环，矛盾
- $w(e') = w(e)$，在 T 中用 e 换掉 e'
- 有限步后可把 T 变为 K 且权值不变，因此 K 就是最小生成树

斯坦纳树

斯坦纳树

- 斯坦纳树是一种最小生成树问题的变形，即给定一个点集，求连通这个点集的最小子图 (显然它也是棵树)

斯坦纳树

- 斯坦纳树是一种最小生成树问题的变形，即给定一个点集，求连通这个点集的最小子图 (显然它也是棵树)
- 该问题是一个 NP 问题，但在点集大小不大时有一个状态压缩 DP 的方法求解

斯坦纳树

- 斯坦纳树是一种最小生成树问题的变形，即给定一个点集，求连通这个点集的最小子图 (显然它也是棵树)
- 该问题是一个 NP 问题，但在点集大小不大时有一个状态压缩 DP 的方法求解
- 设 $dp[x][S]$ 表示包含 S 中所有点 (S 是给定点集的一个子集) 以及点 x(x 可以不在给定的点集中) 的最小生成树

斯坦纳树

- 斯坦纳树是一种最小生成树问题的变形，即给定一个点集，求连通这个点集的最小子图 (显然它也是棵树)
- 该问题是一个 NP 问题，但在点集大小不大时有一个状态压缩 DP 的方法求解
- 设 $dp[x][S]$ 表示包含 S 中所有点 (S 是给定点集的一个子集) 以及点 x(x 可以不在给定的点集中) 的最小生成树
- $dp[x][S] = \min\{dp[y][S'] + w(x, y)\}$，其中 S' 加入点 x 后变为 S

斯坦纳树

- 斯坦纳树是一种最小生成树问题的变形，即给定一个点集，求连通这个点集的最小子图 (显然它也是棵树)
- 该问题是一个 NP 问题，但在点集大小不大时有一个状态压缩 DP 的方法求解
- 设 $dp[x][S]$ 表示包含 S 中所有点 (S 是给定点集的一个子集) 以及点 x(x 可以不在给定的点集中) 的最小生成树
- $dp[x][S] = \min\{dp[y][S'] + w(x, y)\}$，其中 S' 加入点 x 后变为 S
- $dp[x][S] = \min(dp[x][S], dp[x][T] + dp[x][S - T])$

斯坦纳树

- 斯坦纳树是一种最小生成树问题的变形，即给定一个点集，求连通这个点集的最小子图 (显然它也是棵树)
- 该问题是一个 NP 问题，但在点集大小不大时有一个状态压缩 DP 的方法求解
- 设 $dp[x][S]$ 表示包含 S 中所有点 (S 是给定点集的一个子集) 以及点 x(x 可以不在给定的点集中) 的最小生成树
- $dp[x][S] = \min\{dp[y][S'] + w(x, y)\}$，其中 S' 加入点 x 后变为 S
- $dp[x][S] = \min(dp[x][S], dp[x][T] + dp[x][S - T])$
- 正确性可以这么理解：最优解是一棵树，将树拆分为若干条链，式子 1 可以求出这一条条链，式子 2 则将这些链粘连成一棵树

斯坦纳树

- 斯坦纳树是一种最小生成树问题的变形，即给定一个点集，求连通这个点集的最小子图 (显然它也是棵树)
- 该问题是一个 NP 问题，但在点集大小不大时有一个状态压缩 DP 的方法求解
- 设 $dp[x][S]$ 表示包含 S 中所有点 (S 是给定点集的一个子集) 以及点 x(x 可以不在给定的点集中) 的最小生成树
- $dp[x][S] = \min\{dp[y][S'] + w(x, y)\}$，其中 S' 加入点 x 后变为 S
- $dp[x][S] = \min(dp[x][S], dp[x][T] + dp[x][S - T])$
- 正确性可以这么理解：最优解是一棵树，将树拆分为若干条链，式子 1 可以求出这一条条链，式子 2 则将这些链粘连成一棵树
- 对于重复的取点不需要担心，因为最优解中一定不会包含这种情况

斯坦纳树

斯坦纳树

- 按照点集大小从小到大枚举 S，然后先将第二种转移做完，再利用最短路转移第一种式子 (此时转移过程中可以不改变 S，只改变第一维，可以看做第二维的改变会在将来的第二种转移中处理)

斯坦纳树

- 按照点集大小从小到大枚举 S，然后先将第二种转移做完，再利用最短路转移第一种式子 (此时转移过程中可以不改变 S，只改变第一维，可以看做第二维的改变会在将来的第二种转移中处理)
- DP 的状态数是 $O(2^k n)$ 的，转移时需要枚举子集，所以算法复杂度是 $O(3^k n + 2^k n \log n)$，k 是给定特殊点集的大小

Tarjan 算法

Tarjan 算法

- 求解强连通分量、割点、桥都需要用到 Tarjan 算法

Tarjan 算法

- 求解强连通分量、割点、桥都需要用到 Tarjan 算法
- Tarjan 算法基于 DFS，令 dfn_i 表示点 i 在 DFS 中搜索到的次序，low_i 表示从 i 点出发只经过树边 (向子孙方向) 以及至多一条非树边，所能到达的点的 dfn 最小值

Tarjan 算法

- 求解强连通分量、割点、桥都需要用到 Tarjan 算法
- Tarjan 算法基于 DFS，令 dfn_i 表示点 i 在 DFS 中搜索到的次序，low_i 表示从 i 点出发只经过树边 (向子孙方向) 以及至多一条非树边，所能到达的点的 dfn 最小值
- 强连通分量 (SCC) 的性质：1. 从任意一点出发都能遍历 SCC；2. 将 SCC 缩成一个点，则图变为 DAG

Tarjan 算法

- 求解强连通分量、割点、桥都需要用到 Tarjan 算法
- Tarjan 算法基于 DFS，令 dfn_i 表示点 i 在 DFS 中搜索到的次序，low_i 表示从 i 点出发只经过树边 (向子孙方向) 以及至多一条非树边，所能到达的点的 dfn 最小值
- 强连通分量 (SCC) 的性质：1. 从任意一点出发都能遍历 SCC；2. 将 SCC 缩成一个点，则图变为 DAG
- 由性质 1，SCC 在搜索树中一定是连续的一块点，所以搜索树最底层的 SCC 一定是一个子树，重复地将它们找出后删除便能求出所有 SCC

Tarjan 算法

- 求解强连通分量、割点、桥都需要用到 Tarjan 算法
- Tarjan 算法基于 DFS，令 dfn_i 表示点 i 在 DFS 中搜索到的次序，low_i 表示从 i 点出发只经过树边 (向子孙方向) 以及至多一条非树边，所能到达的点的 dfn 最小值
- 强连通分量 (SCC) 的性质：1. 从任意一点出发都能遍历 SCC；2. 将 SCC 缩成一个点，则图变为 DAG
- 由性质 1，SCC 在搜索树中一定是连续的一块点，所以搜索树最底层的 SCC 一定是一个子树，重复地将它们找出后删除便能求出所有 SCC
- 一个以 i 为根的子树是 SCC 当且仅当 $dfn_i = low_i$ 且子树中不存在满足此性质的点

Tarjan 算法

- 求解强连通分量、割点、桥都需要用到 Tarjan 算法
- Tarjan 算法基于 DFS，令 dfn_i 表示点 i 在 DFS 中搜索到的次序，low_i 表示从 i 点出发只经过树边 (向子孙方向) 以及至多一条非树边，所能到达的点的 dfn 最小值
- 强连通分量 (SCC) 的性质：1. 从任意一点出发都能遍历 SCC；2. 将 SCC 缩成一个点，则图变为 DAG
- 由性质 1，SCC 在搜索树中一定是连续的一块点，所以搜索树最底层的 SCC 一定是一个子树，重复地将它们找出后删除便能求出所有 SCC
- 一个以 i 为根的子树是 SCC 当且仅当 $dfn_i = low_i$ 且子树中不存在满足此性质的点
- 利用栈存储子树中未被删除的点，则能求出 SCC 具体的点集

Tarjan 算法

Tarjan 算法

- 若 u 的儿子 v 满足 $low_v \geq dfn_u$ 则 u 是割点

Tarjan 算法

- 若 u 的儿子 v 满足 $low_v \geq dfn_u$ 则 u 是割点
- 若 u 的儿子 v 满足 $low_v > dfn_u$ 则 (u, v) 是桥 (要求树边对应的那条反向边不能用于更新 low)

Tarjan 算法

- 若 u 的儿子 v 满足 $low_v \geq dfn_u$ 则 u 是割点
- 若 u 的儿子 v 满足 $low_v > dfn_u$ 则 (u, v) 是桥 (要求树边对应的那条反向边不能用于更新 low)
- 求三种量时 Tarjan 算法的细节有差异，比如求割点要特判根，求桥要忽略树边的反向边，求 SCC 要注意已经求得 SCC 的点不能用于更新 low

Tarjan 算法

- 若 u 的儿子 v 满足 $low_v \geq dfn_u$ 则 u 是割点
- 若 u 的儿子 v 满足 $low_v > dfn_u$ 则 (u, v) 是桥 (要求树边对应的那条反向边不能用于更新 low)
- 求三种量时 Tarjan 算法的细节有差异，比如求割点要特判根，求桥要忽略树边的反向边，求 SCC 要注意已经求得 SCC 的点不能用于更新 low
- 边双连通分量之间交集为空，故将它们缩成一个点后，与所有桥边一起组成一棵树，所以求边双常常删掉桥边后利用并查集求解

Tarjan 算法

- 若 u 的儿子 v 满足 $low_v \geq dfn_u$ 则 u 是割点
- 若 u 的儿子 v 满足 $low_v > dfn_u$ 则 (u, v) 是桥 (要求树边对应的那条反向边不能用于更新 low)
- 求三种量时 Tarjan 算法的细节有差异，比如求割点要特判根，求桥要忽略树边的反向边，求 SCC 要注意已经求得 SCC 的点不能用于更新 low
- 边双连通分量之间交集为空，故将它们缩成一个点后，与所有桥边一起组成一棵树，所以求边双常常删掉桥边后利用并查集求解
- 点双连通分量之间会有交集，尽管它不能像 SCC 和边双一样缩点，但是若将割点看成边，则点双之间也仍然还有类似的树形结构

2SAT 问题

2SAT 问题

- 2SAT 问题：有 n 个待赋值的布尔变量 b_i 以及 m 个约束条件，每个约束条件形如 b_i 取 x 时 b_j 必定取 y，问是否有一个赋值方法满足条件 (实际上该条件即为一个合取范式)

2SAT 问题

- 2SAT 问题：有 n 个待赋值的布尔变量 b_i 以及 m 个约束条件，每个约束条件形如 b_i 取 x 时 b_j 必定取 y，问是否有一个赋值方法满足条件 (实际上该条件即为一个合取范式)
- 构造一个 $2n$ 个点的有向图，每个布尔变量 b_i 有两个点 t_i, f_i 分别代表取真或假，一个点被选表示使用该赋值方法

2SAT 问题

- 2SAT 问题：有 n 个待赋值的布尔变量 b_i 以及 m 个约束条件，每个约束条件形如 b_i 取 x 时 b_j 必定取 y，问是否有一个赋值方法满足条件 (实际上该条件即为一个合取范式)
- 构造一个 $2n$ 个点的有向图，每个布尔变量 b_i 有两个点 t_i, f_i 分别代表取真或假，一个点被选表示使用该赋值方法
- 将约束条件建成边：例如 b_i 取 0 时 b_j 一定取 1，则建边：(f_i, t_j)，(f_j, t_i)，即图中的边 (a, b) 表示选 a 必定选 b，且满足逆否命题成立 (即直观上看边是对称的)

2SAT 问题

- 2SAT 问题：有 n 个待赋值的布尔变量 b_i 以及 m 个约束条件，每个约束条件形如 b_i 取 x 时 b_j 必定取 y，问是否有一个赋值方法满足条件 (实际上该条件即为一个合取范式)
- 构造一个 $2n$ 个点的有向图，每个布尔变量 b_i 有两个点 t_i, f_i 分别代表取真或假，一个点被选表示使用该赋值方法
- 将约束条件建成边：例如 b_i 取 0 时 b_j 一定取 1，则建边：(f_i, t_j)，(f_j, t_i)，即图中的边 (a, b) 表示选 a 必定选 b，且满足逆否命题成立 (即直观上看边是对称的)
- 可以发现该图中，任何一个 SCC 内的点都必须全选或全不选，因此先求出该图的 SCC

2SAT 问题

2SAT 问题

- 若 f_i 与 t_i 同在一个 SCC 中则问题无解；否则一定有解，考虑如何求方案

2SAT 问题

- 若 f_i 与 t_i 同在一个 SCC 中则问题无解；否则一定有解，考虑如何求方案
- 首先将 SCC 缩成点，考虑缩点后的 DAG

2SAT 问题

- 若 f_i 与 t_i 同在一个 SCC 中则问题无解；否则一定有解，考虑如何求方案
- 首先将 SCC 缩成点，考虑缩点后的 DAG
- 若按 DAG 的拓扑序考虑每个 SCC 是否取，则假定某个 SCC 被取，那么 SCC 内每个变量对应取值为反的那个变量所在的 SCC 需被设定为不可取

2SAT 问题

- 若 f_i 与 t_i 同在一个 SCC 中则问题无解；否则一定有解，考虑如何求方案
- 首先将 SCC 缩成点，考虑缩点后的 DAG
- 若按 DAG 的拓扑序考虑每个 SCC 是否取，则假定某个 SCC 被取，那么 SCC 内每个变量对应取值为反的那个变量所在的 SCC 需被设定为不可取
- 容易发现这种方式会引起矛盾，所以应该按照 DAG 拓扑序的逆序进行选择，此时能取就取即可满足条件

2SAT 问题

- 若 f_i 与 t_i 同在一个 SCC 中则问题无解；否则一定有解，考虑如何求方案
- 首先将 SCC 缩成点，考虑缩点后的 DAG
- 若按 DAG 的拓扑序考虑每个 SCC 是否取，则假定某个 SCC 被取，那么 SCC 内每个变量对应取值为反的那个变量所在的 SCC 需被设定为不可取
- 容易发现这种方式会引起矛盾，所以应该按照 DAG 拓扑序的逆序进行选择，此时能取就取即可满足条件
- 由于边的对称性，所以若 f_i 所在 SCC 的拓扑序在 t_i 所在 SCC 之后则 b_i 取 0 这种方式也可得出满足条件的方案

2SAT 问题

- 若 f_i 与 t_i 同在一个 SCC 中则问题无解；否则一定有解，考虑如何求方案
- 首先将 SCC 缩成点，考虑缩点后的 DAG
- 若按 DAG 的拓扑序考虑每个 SCC 是否取，则假定某个 SCC 被取，那么 SCC 内每个变量对应取值为反的那个变量所在的 SCC 需被设定为不可取
- 容易发现这种方式会引起矛盾，所以应该按照 DAG 拓扑序的逆序进行选择，此时能取就取即可满足条件
- 由于边的对称性，所以若 f_i 所在 SCC 的拓扑序在 t_i 所在 SCC 之后则 b_i 取 0 这种方式也可得出满足条件的方案
- 注意到 Tarjan 算法所求的强连通分量就是按拓扑序的逆序得出的，因此不需要真的缩点建新图求拓扑序，直接利用强连通分量的编号来当做顺序即可

欧拉回路

欧拉回路

- 首先考虑无向连通图

欧拉回路

- 首先考虑无向连通图
- 无向图中存在欧拉回路当且仅当该图不存在度数为奇数的点

欧拉回路

- 首先考虑无向连通图
- 无向图中存在欧拉回路当且仅当该图不存在度数为奇数的点
- 必要性：欧拉回路中每个点度数都是偶数

欧拉回路

- 首先考虑无向连通图
- 无向图中存在欧拉回路当且仅当该图不存在度数为奇数的点
- 必要性：欧拉回路中每个点度数都是偶数
- 充分性：考虑归纳，在没有奇点的连通图中一定能找到一个环，将这个环删去后会得到多个联通块，由于删去的是环所以联通块中也不存在奇点

欧拉回路

- 首先考虑无向连通图
- 无向图中存在欧拉回路当且仅当该图不存在度数为奇数的点
- 必要性：欧拉回路中每个点度数都是偶数
- 充分性：考虑归纳，在没有奇点的连通图中一定能找到一个环，将这个环删去后会得到多个联通块，由于删去的是环所以联通块中也不存在奇点
- 根据归纳每个连通块间都存在一个欧拉回路，那么将这些回路套在原图找到的环上，则得到了原图的一条欧拉回路

欧拉回路

- 首先考虑无向连通图
- 无向图中存在欧拉回路当且仅当该图不存在度数为奇数的点
- 必要性：欧拉回路中每个点度数都是偶数
- 充分性：考虑归纳，在没有奇点的连通图中一定能找到一个环，将这个环删去后会得到多个联通块，由于删去的是环所以联通块中也不存在奇点
- 根据归纳每个连通块间都存在一个欧拉回路，那么将这些回路套在原图找到的环上，则得到了原图的一条欧拉回路
- 利用这个过程我们也得到了求解欧拉回路的算法：从任意一点出发开始 DFS，经过一条边就将其删去，点访问顺序的逆序就是一条欧拉回路

欧拉回路

欧拉回路

- 无向图中存在欧拉路径当且仅当该图包含不超过两个奇点 (实际上，奇点个数一定是偶数)

欧拉回路

- 无向图中存在欧拉路径当且仅当该图包含不超过两个奇点 (实际上，奇点个数一定是偶数)
- 求欧拉路径可以在两个奇点间连一条边然后求出欧拉回路，再在回路中删去这条边并进行适当修改

欧拉回路

- 无向图中存在欧拉路径当且仅当该图包含不超过两个奇点 (实际上，奇点个数一定是偶数)
- 求欧拉路径可以在两个奇点间连一条边然后求出欧拉回路，再在回路中删去这条边并进行适当修改
- 有向图的欧拉回路有一个类似结论：有向连通图存在欧拉回路当且仅当所有点的入度等于出度，证明与求法与无向图一致

Part II

网络流

引入

引入

- 如同我们可以把实际的道路地图抽象成一个有向图来计算两点之间最短路，我们也可以将一个有向图看作一个网络流图来解决另一类问题

引入

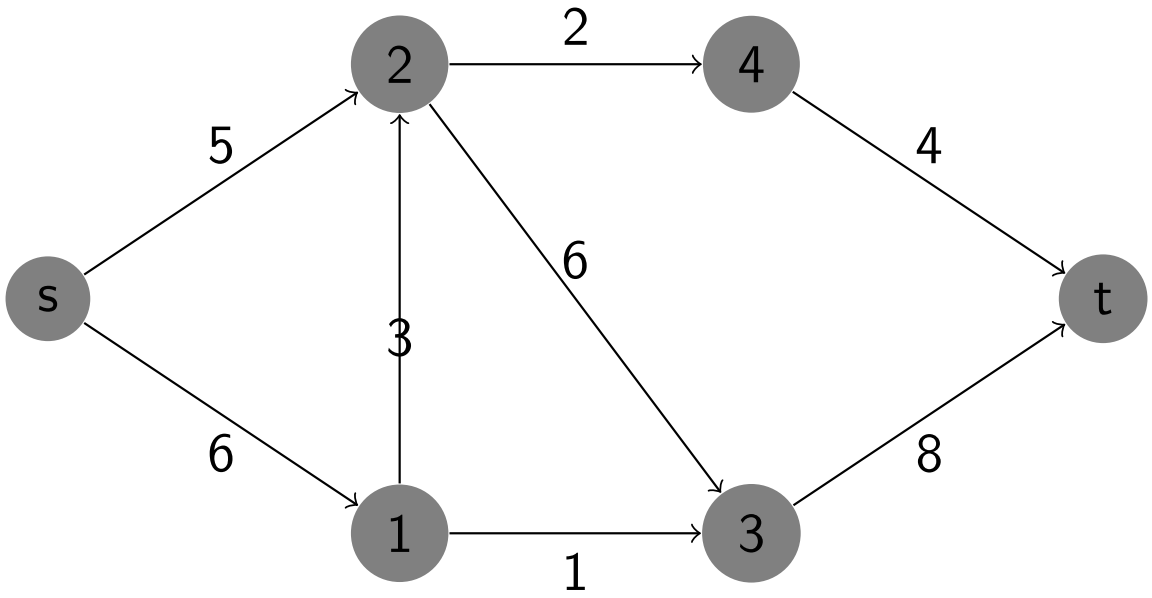
- 如同我们可以把实际的道路地图抽象成一个有向图来计算两点之间最短路，我们也可以将一个有向图看作一个网络流图来解决另一类问题
- 网络流可以用来模拟水流经管道、电流在电路网络中的运动、信息网络中信息的传递等等过程

引入

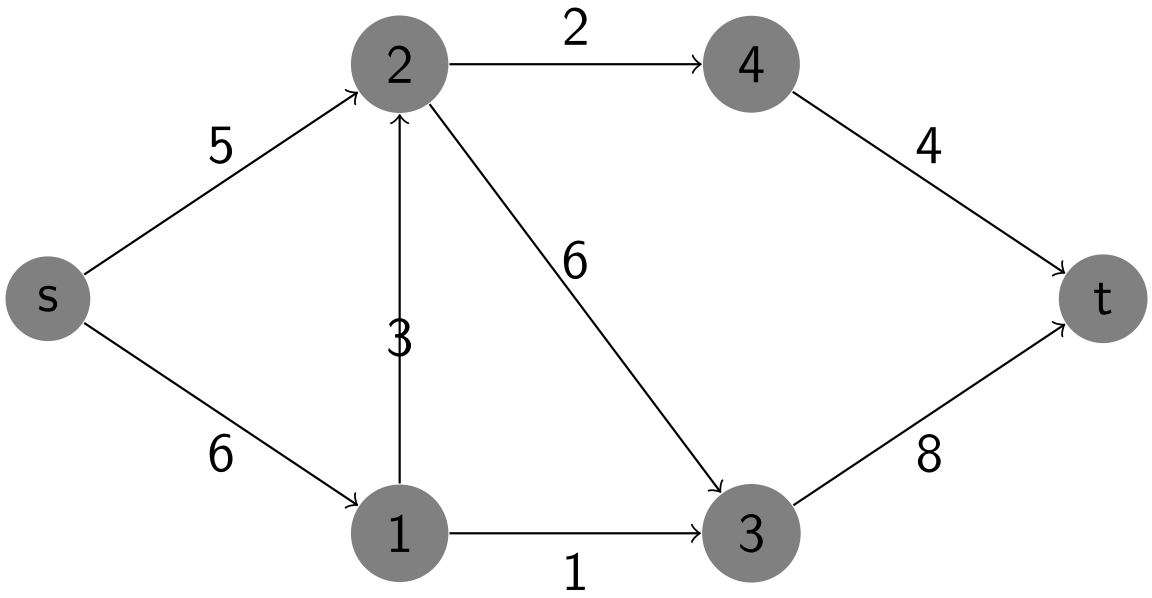
- 如同我们可以把实际的道路地图抽象成一个有向图来计算两点之间最短路，我们也可以将一个有向图看作一个网络流图来解决另一类问题
- 网络流可以用来模拟水流经管道、电流在电路网络中的运动、信息网络中信息的传递等等过程
- 例如我们将网络流类比成水流，图中的边可以想象成管道，顶点则是管道的连接点

引入

引入

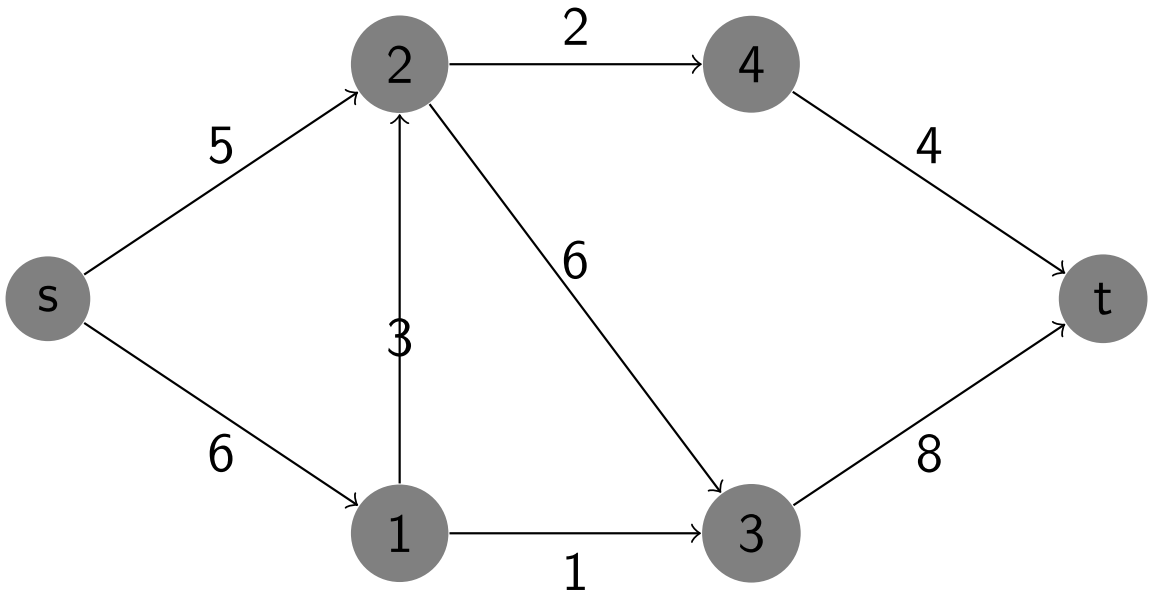


引入



- 将上图看成一个水流图，s 是水源能流出无穷多水，t 是一个蓄水池能容纳无穷多水，每条边是一个管道，边上的权值代表着单位时间内通过的水流量的上限

引入



- 将上图看成一个水流图，s 是水源能流出无穷多水，t 是一个蓄水池能容纳无穷多水，每条边是一个管道，边上的权值代表着单位时间内通过的水流量的上限
- 流动的水不能凭空出现也不能凭空消失，所以除源汇外流入一个结点的水量等于从它流出的水量

定义

定义

- 给定有向图 $G = (V, E)$，其中 V 是点集，E 是边集

定义

- 给定有向图 $G = (V, E)$，其中 V 是点集，E 是边集
- 每条边 $e \in E$ 有一个非负权值 $c(e)$ 称为容量

定义

- 给定有向图 $G = (V, E)$，其中 V 是点集，E 是边集
- 每条边 $e \in E$ 有一个非负权值 $c(e)$ 称为容量
- 图中有两个特殊点：源点 s（可提供无穷多流量），以及汇点 t（可接纳无穷多流量）

定义

- 给定有向图 $G = (V, E)$，其中 V 是点集，E 是边集
- 每条边 $e \in E$ 有一个非负权值 $c(e)$ 称为容量
- 图中有两个特殊点：源点 s（可提供无穷多流量），以及汇点 t（可接纳无穷多流量）
- 有源汇网络的可行流是一个实值函数 $f(e)$：

定义

- 给定有向图 $G = (V, E)$，其中 V 是点集，E 是边集
- 每条边 $e \in E$ 有一个非负权值 $c(e)$ 称为容量
- 图中有两个特殊点：源点 s（可提供无穷多流量），以及汇点 t（可接纳无穷多流量）
- 有源汇网络的可行流是一个实值函数 $f(e)$：
  - 容量限制：$0 \leq f(e) \leq c(e), e \in E$

定义

- 给定有向图 $G = (V, E)$，其中 V 是点集，E 是边集
- 每条边 $e \in E$ 有一个非负权值 $c(e)$ 称为容量
- 图中有两个特殊点：源点 s（可提供无穷多流量），以及汇点 t（可接纳无穷多流量）
- 有源汇网络的可行流是一个实值函数 $f(e)$：
  - 容量限制：$0 \leq f(e) \leq c(e), e \in E$
  - 流量守恒：

$$\sum_{e=(u,v)\in E} f(e) = \sum_{e=(v,w)\in E} f(e), v \in V \setminus \{s, t\}$$

定义

- 给定有向图 $G = (V, E)$，其中 V 是点集，E 是边集
- 每条边 $e \in E$ 有一个非负权值 $c(e)$ 称为容量
- 图中有两个特殊点：源点 s（可提供无穷多流量），以及汇点 t（可接纳无穷多流量）
- 有源汇网络的可行流是一个实值函数 $f(e)$：
  - 容量限制：$0 \le f(e) \le c(e), e \in E$
  - 流量守恒：

$$\sum_{e=(u,v)\in E} f(e) = \sum_{e=(v,w)\in E} f(e), v \in V \setminus \{s, t\}$$

  - 总流量：$F = \sum_{e=(s,u)\in E} f(e)$，即源点流出的总流量，它也等于 $\sum_{e=(u,t)\in E} f(e)$，即流入汇点的总流量

定义

- 给定有向图 $G = (V, E)$，其中 V 是点集，E 是边集
- 每条边 $e \in E$ 有一个非负权值 $c(e)$ 称为容量
- 图中有两个特殊点：源点 s（可提供无穷多流量），以及汇点 t（可接纳无穷多流量）
- 有源汇网络的可行流是一个实值函数 $f(e)$：
  - 容量限制：$0 \le f(e) \le c(e), e \in E$
  - 流量守恒：
$$\sum_{e=(u,v)\in E} f(e) = \sum_{e=(v,w)\in E} f(e), v \in V \setminus \{s, t\}$$
  - 总流量：$F = \sum_{e=(s,u)\in E} f(e)$，即源点流出的总流量，它也等于 $\sum_{e=(u,t)\in E} f(e)$，即流入汇点的总流量
- 使 F 最大化的可行流方案称为最大流 (不一定唯一)

求解最大流

求解最大流

- 考虑如下的贪心算法：

求解最大流

- 考虑如下的贪心算法：
  - 找一条 s 到 t 的只经过 $f(e) < c(e)$ 的边的路径

求解最大流

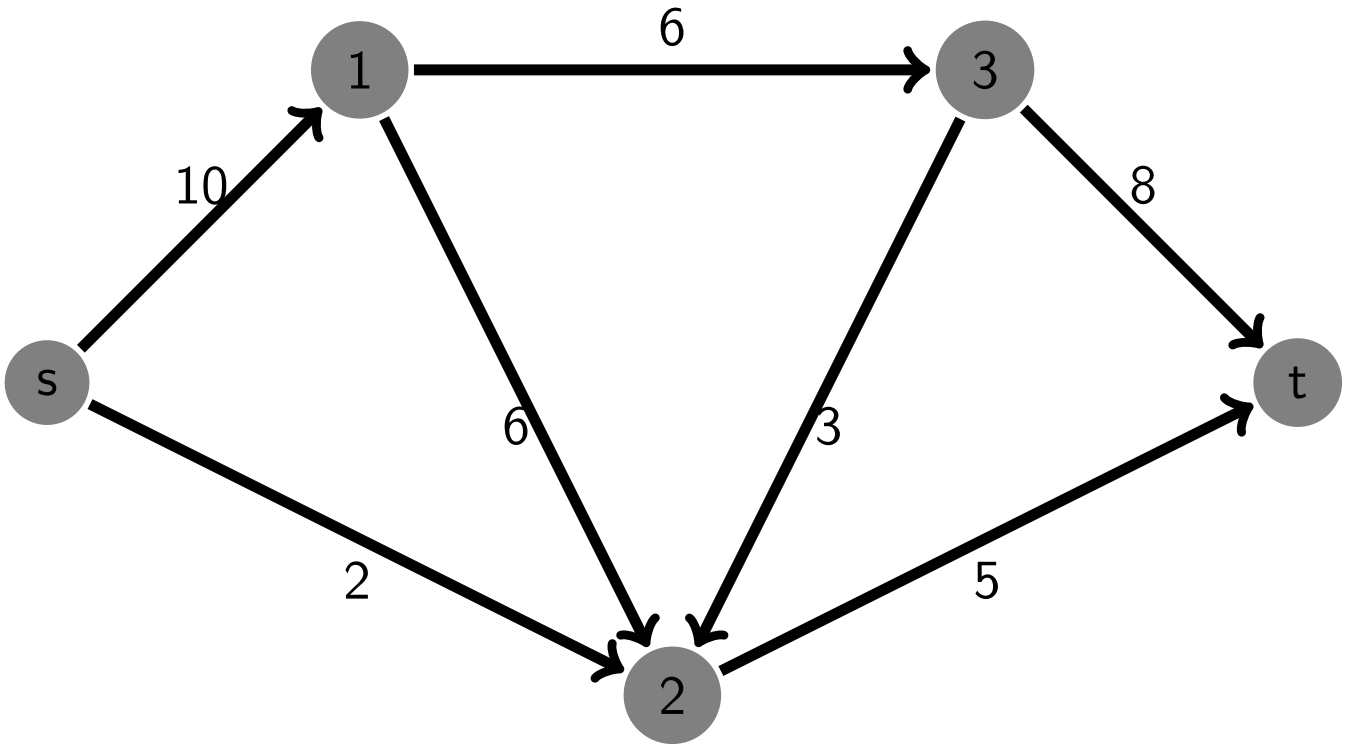
- 考虑如下的贪心算法：
  - 找一条 s 到 t 的只经过 $f(e) < c(e)$ 的边的路径
  - 若不存在满足条件的路径则算法结束，否则，沿着该路径尽可能地增加 $f(e)$，然后返回上一步。我们可以将这步称为增广

求解最大流

- 考虑如下的贪心算法：
  - 找一条 s 到 t 的只经过 $f(e) < c(e)$ 的边的路径
  - 若不存在满足条件的路径则算法结束，否则，沿着该路径尽可能地增加 $f(e)$，然后返回上一步。我们可以将这步称为增广
- 我们考虑对下图使用该算法：

求解最大流

- 考虑如下的贪心算法：
  - 找一条 s 到 t 的只经过 $f(e) < c(e)$ 的边的路径
  - 若不存在满足条件的路径则算法结束，否则，沿着该路径尽可能地增加 $f(e)$，然后返回上一步。我们可以将这步称为增广
- 我们考虑对下图使用该算法：



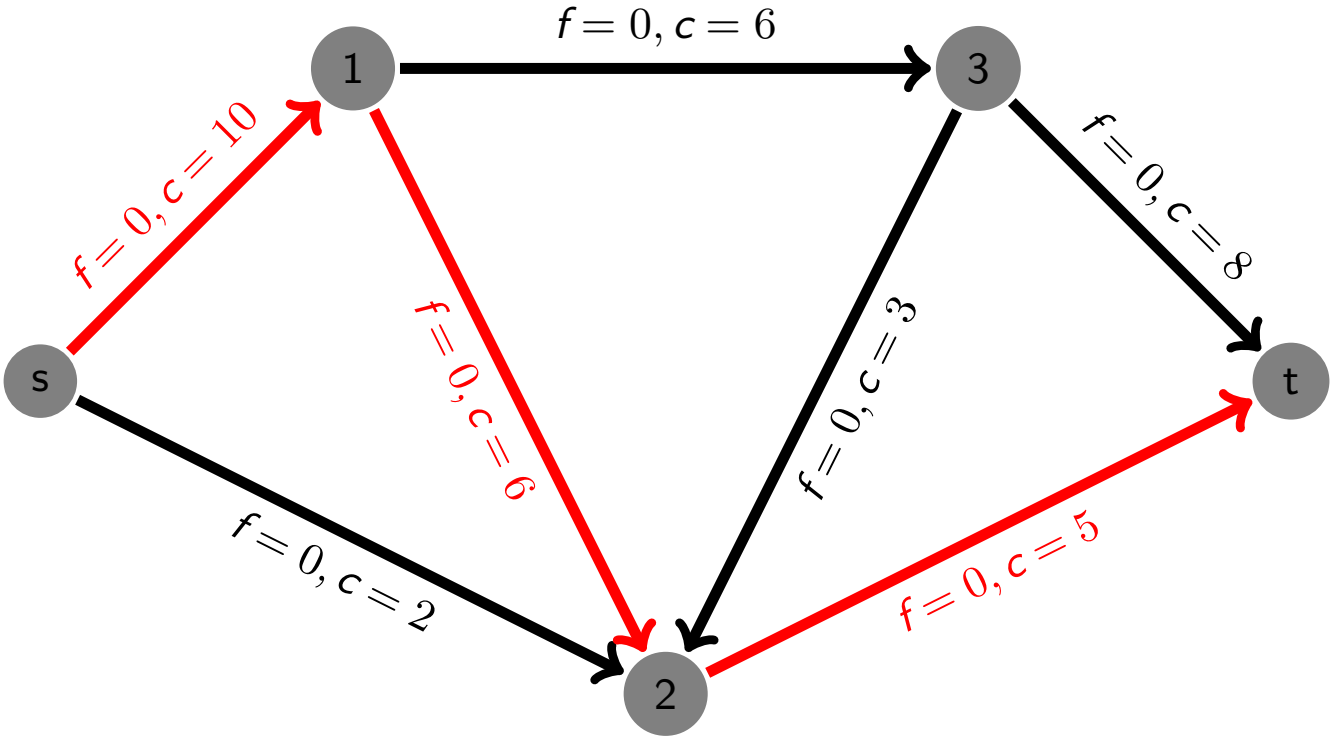
算法流程

算法流程

沿着 $s \to 1 \to 2 \to t$ 流 5

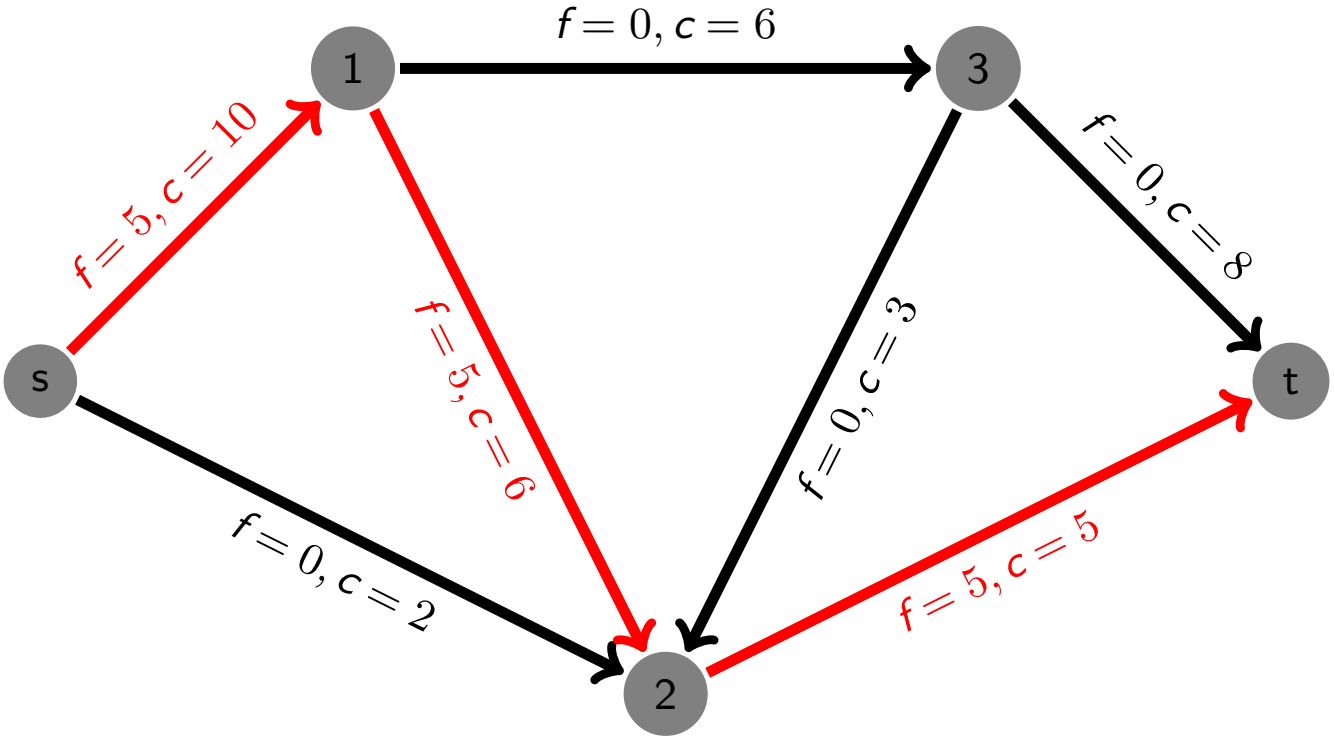
算法流程

沿着 $s \to 1 \to 2 \to t$ 流 5



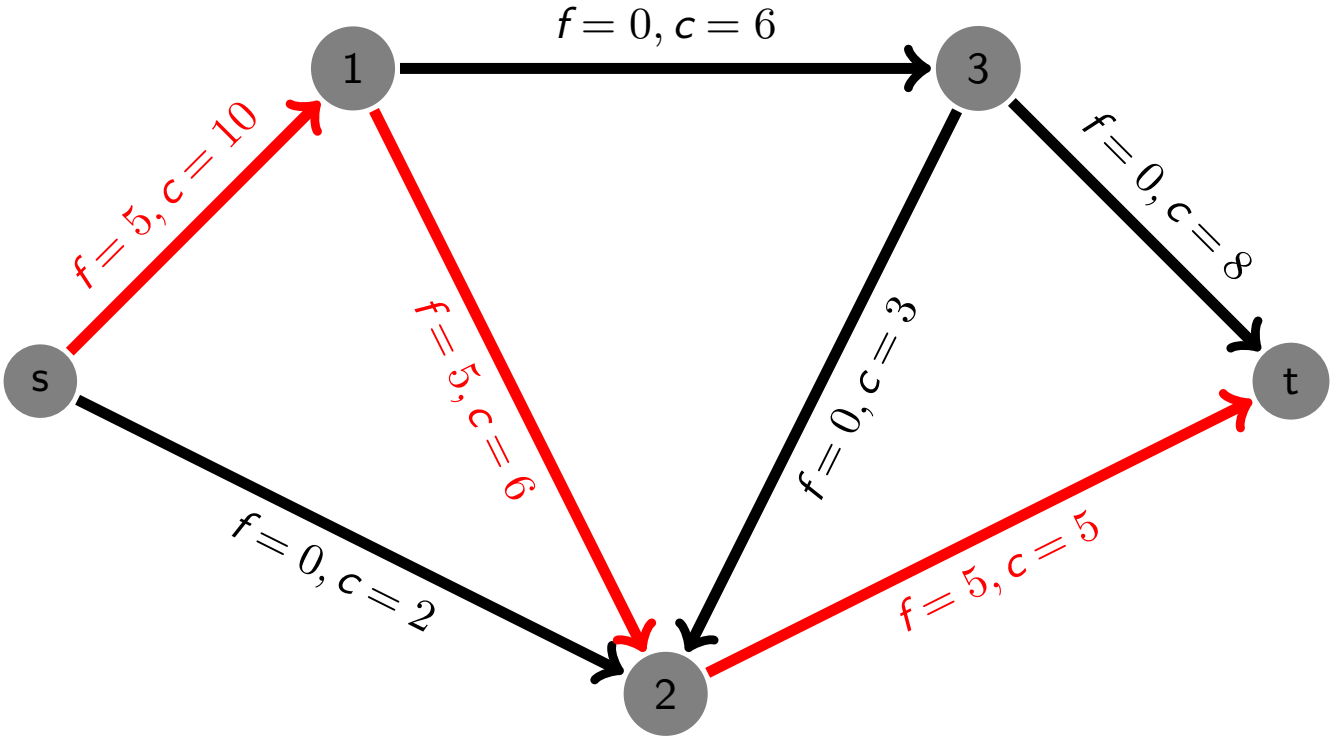
算法流程

沿着 $s \to 1 \to 2 \to t$ 流 5



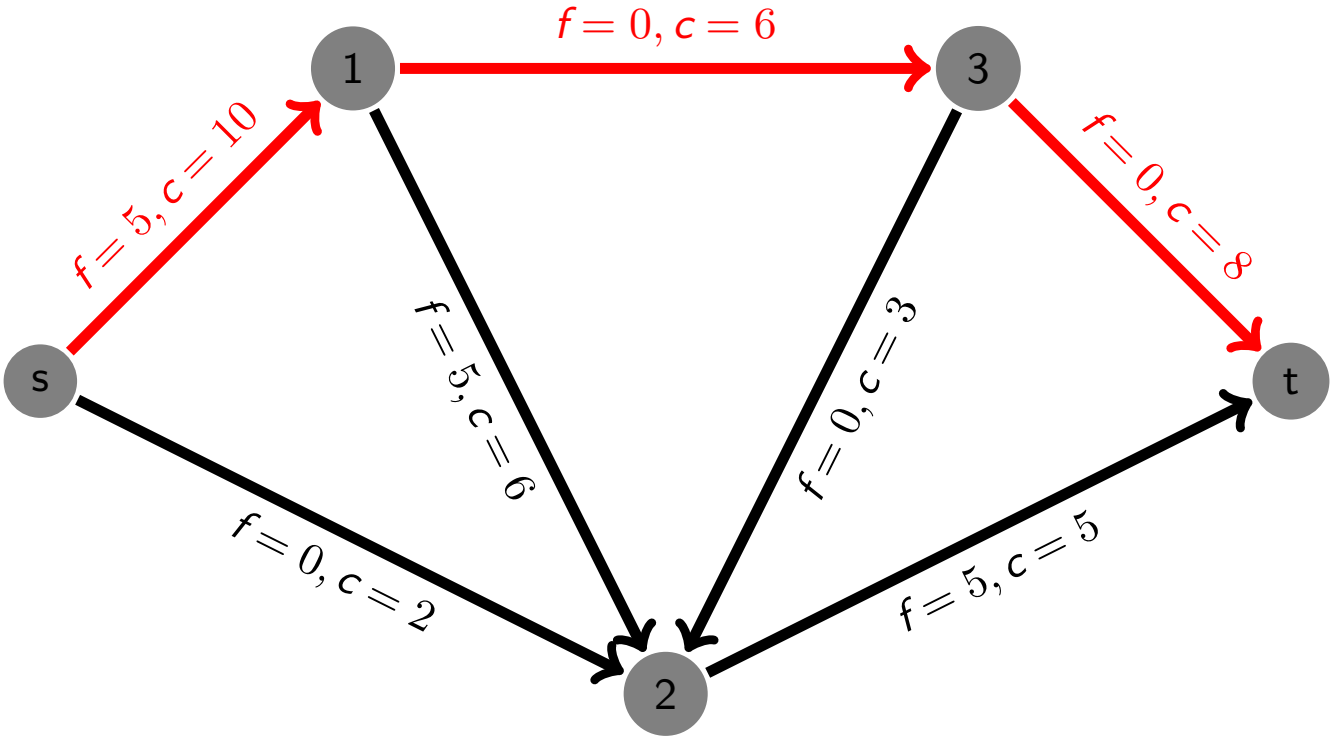
算法流程

再沿着 $s \to 1 \to 3 \to t$ 流 5



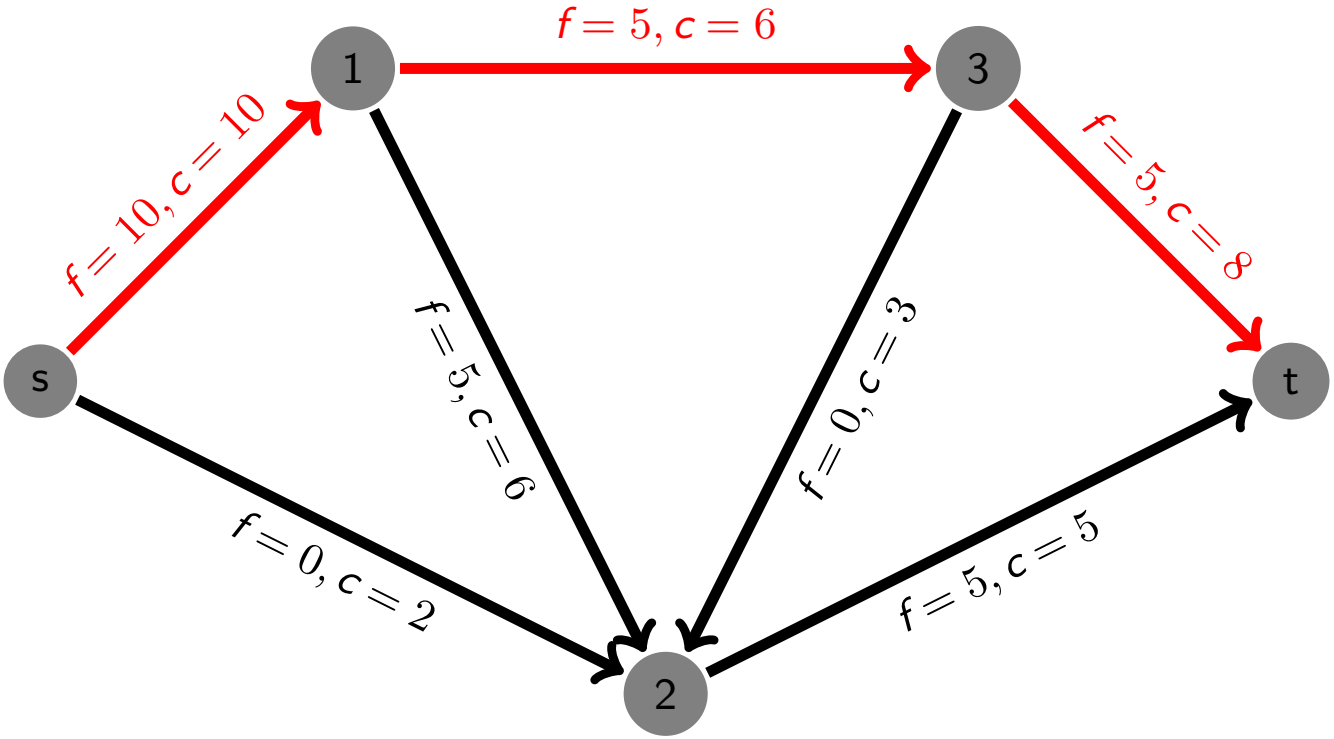
算法流程

再沿着 $s \to 1 \to 3 \to t$ 流 5



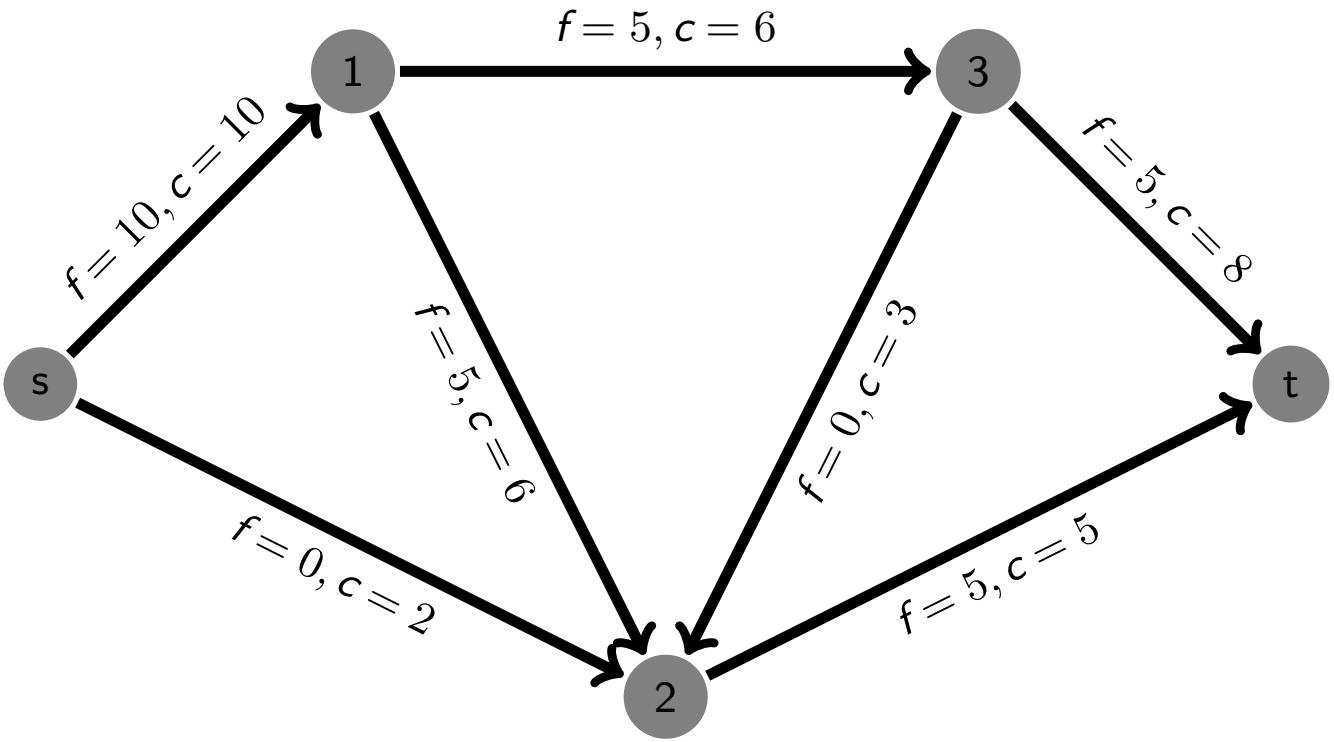
算法流程

再沿着 $s \to 1 \to 3 \to t$ 流 5



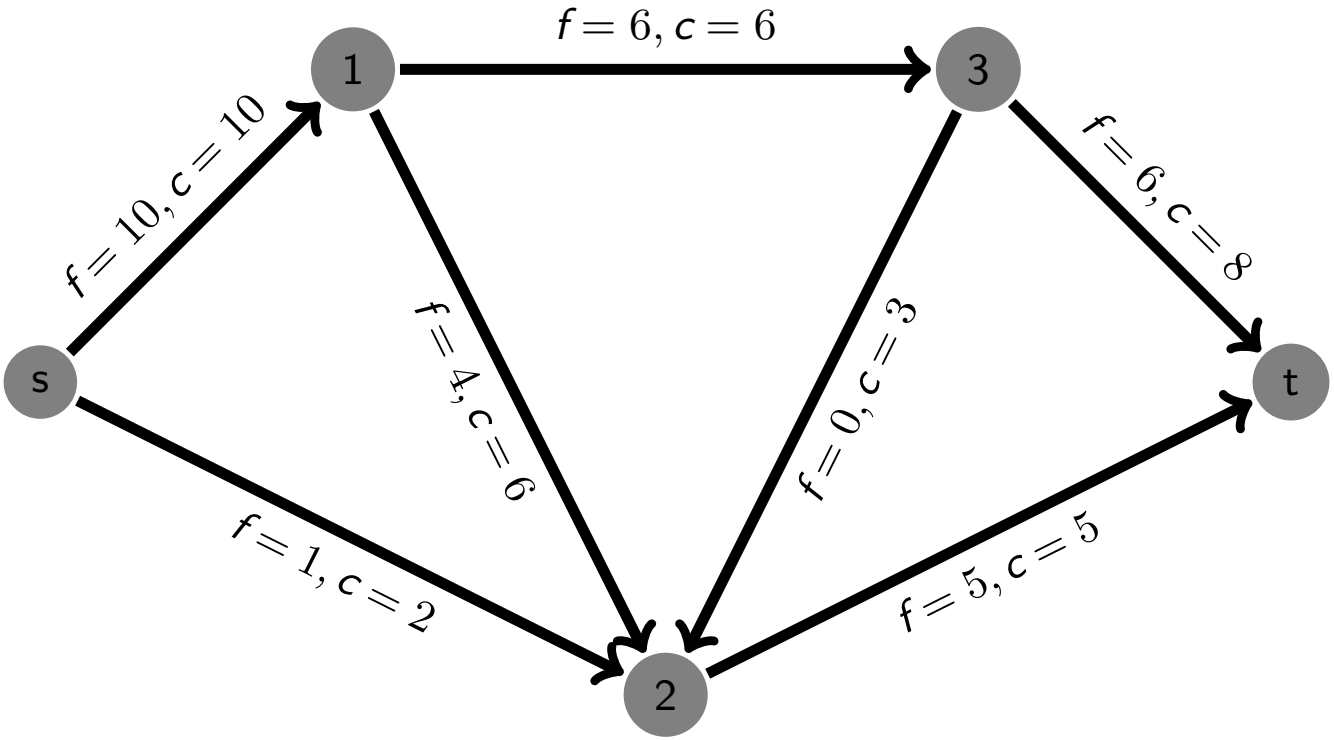
算法流程

找不到新的路径可增广，算法结束，求得答案为 10.



算法流程

然而答案并不正确，下图是一个流为 11 的方案：



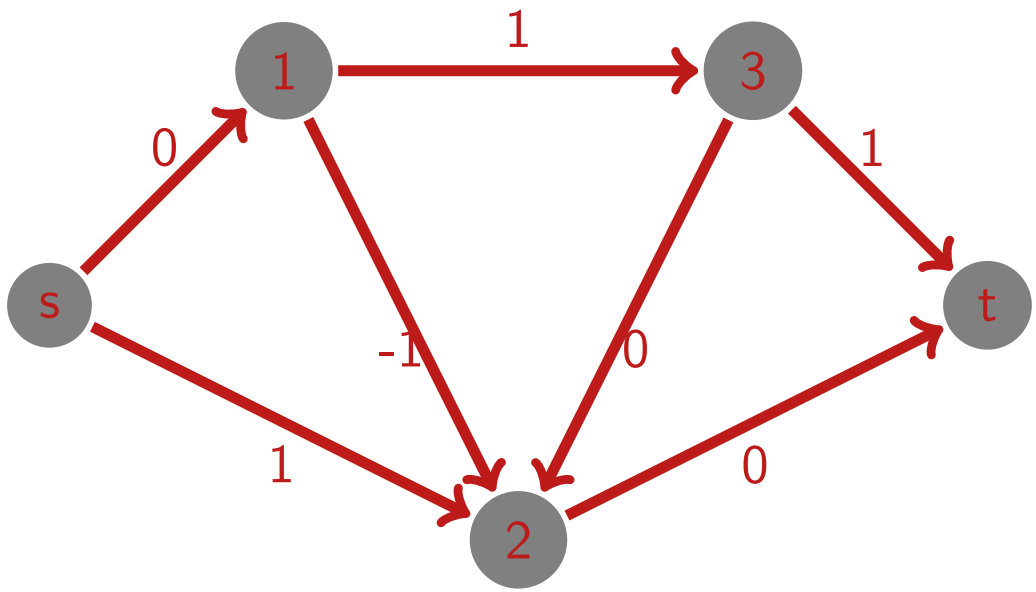
修正算法

修正算法

- 为了找出二者区别，我们不妨来看看它们的流量差 (正确答案-算法得到的结果):

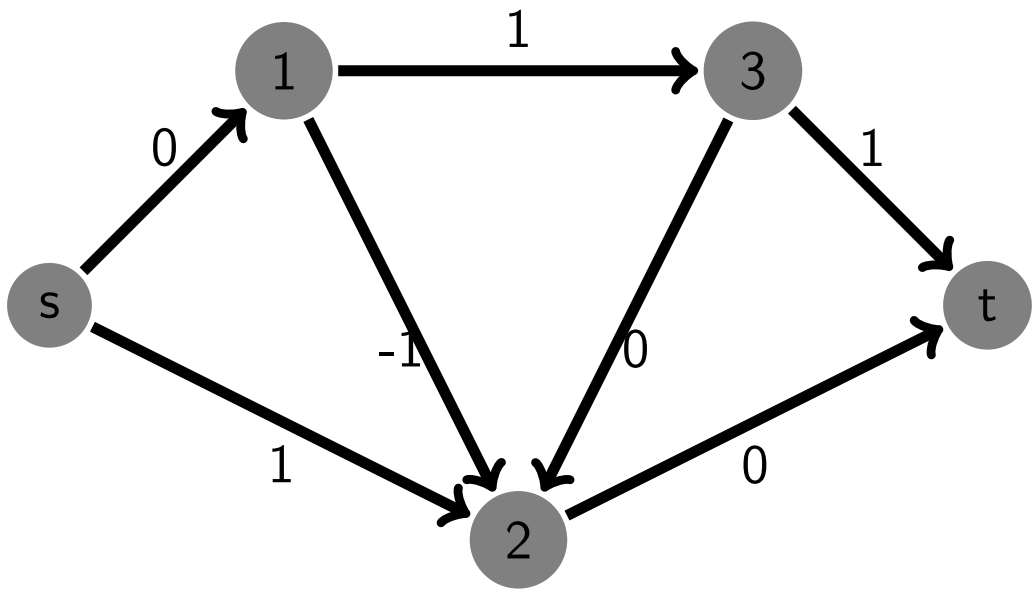
修正算法

- 为了找出二者区别，我们不妨来看看它们的流量差 (正确答案-算法得到的结果)：



修正算法

- 为了找出二者区别，我们不妨来看看它们的流量差 (正确答案-算法得到的结果)：



- 可以发现，我们若让之前 $s \to 1 \to 2 \to t$ 这条路径中的 1 点流量改走 $s \to 1 \to 3 \to t$，然后再沿着 $s \to 2 \to t$ 增广 1 点流量，就可使答案增大

修正算法

修正算法

- 即我们可以将原先得到的流退回去并改道，得到新的流

修正算法

- 即我们可以将原先得到的流退回去并改道，得到新的流
- 此时流量不变，在新流基础上继续增广就能找到更优的答案

修正算法

- 即我们可以将原先得到的流退回去并改道，得到新的流
- 此时流量不变，在新流基础上继续增广就能找到更优的答案
- 现在算法改进如下：

修正算法

- 即我们可以将原先得到的流退回去并改道，得到新的流
- 此时流量不变，在新流基础上继续增广就能找到更优的答案
- 现在算法改进如下：
  - 只利用 $f(e) < c(e)$ 的 e 或 $f(e) > 0$ 的 e 的反向边 $rev(e)$，寻找一条 s 到 t 的路径

修正算法

- 即我们可以将原先得到的流退回去并改道，得到新的流
- 此时流量不变，在新流基础上继续增广就能找到更优的答案
- 现在算法改进如下：
  - 只利用 $f(e) < c(e)$ 的 e 或 $f(e) > 0$ 的 e 的反向边 $rev(e)$，寻找一条 s 到 t 的路径
  - 若不存在满足条件的路径则算法结束；否则沿着该路径尽可能增加流，并返回上一步

修正算法

- 即我们可以将原先得到的流退回去并改道，得到新的流
- 此时流量不变，在新流基础上继续增广就能找到更优的答案
- 现在算法改进如下：
  - 只利用 $f(e) < c(e)$ 的 e 或 $f(e) > 0$ 的 e 的反向边 $rev(e)$，寻找一条 s 到 t 的路径
  - 若不存在满足条件的路径则算法结束；否则沿着该路径尽可能增加流，并返回上一步
- 若沿着 (u, v) 的反向边增广流，则相当于将原来的某条流退回去，并将那条流的 $s \to u \to v$ 改成当前流方案 $s \to v$

修正算法

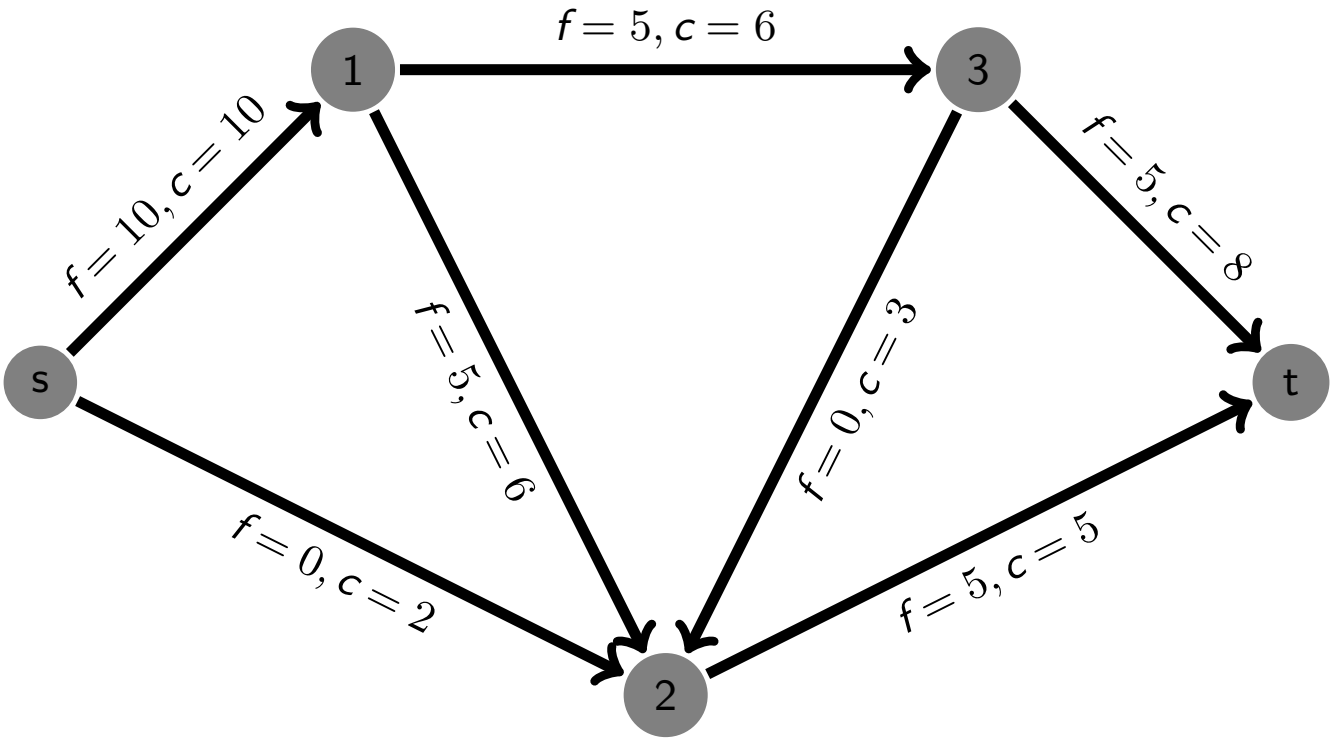
- 即我们可以将原先得到的流退回去并改道，得到新的流
- 此时流量不变，在新流基础上继续增广就能找到更优的答案
- 现在算法改进如下：
  - 只利用 $f(e) < c(e)$ 的 e 或 $f(e) > 0$ 的 e 的反向边 $rev(e)$，寻找一条 s 到 t 的路径
  - 若不存在满足条件的路径则算法结束；否则沿着该路径尽可能增加流，并返回上一步
- 若沿着 (u, v) 的反向边增广流，则相当于将原来的某条流退回去，并将那条流的 $s \to u \to v$ 改成当前流方案 $s \to v$
- 由于最后一定会再走到 t，所以这相当于将那条流改道后，利用那条流的 $s \to u$ 部分再找到了一条 $u \to t$ 的路径

修正算法

- 即我们可以将原先得到的流退回去并改道，得到新的流
- 此时流量不变，在新流基础上继续增广就能找到更优的答案
- 现在算法改进如下：
  - 只利用 $f(e) < c(e)$ 的 e 或 $f(e) > 0$ 的 e 的反向边 $rev(e)$，寻找一条 s 到 t 的路径
  - 若不存在满足条件的路径则算法结束；否则沿着该路径尽可能增加流，并返回上一步
- 若沿着 (u, v) 的反向边增广流，则相当于将原来的某条流退回去，并将那条流的 $s \to u \to v$ 改成当前流方案 $s \to v$
- 由于最后一定会再走到 t，所以这相当于将那条流改道后，利用那条流的 $s \to u$ 部分再找到了一条 $u \to t$ 的路径
- 现在我们在原来基础上进行改进后的贪心算法

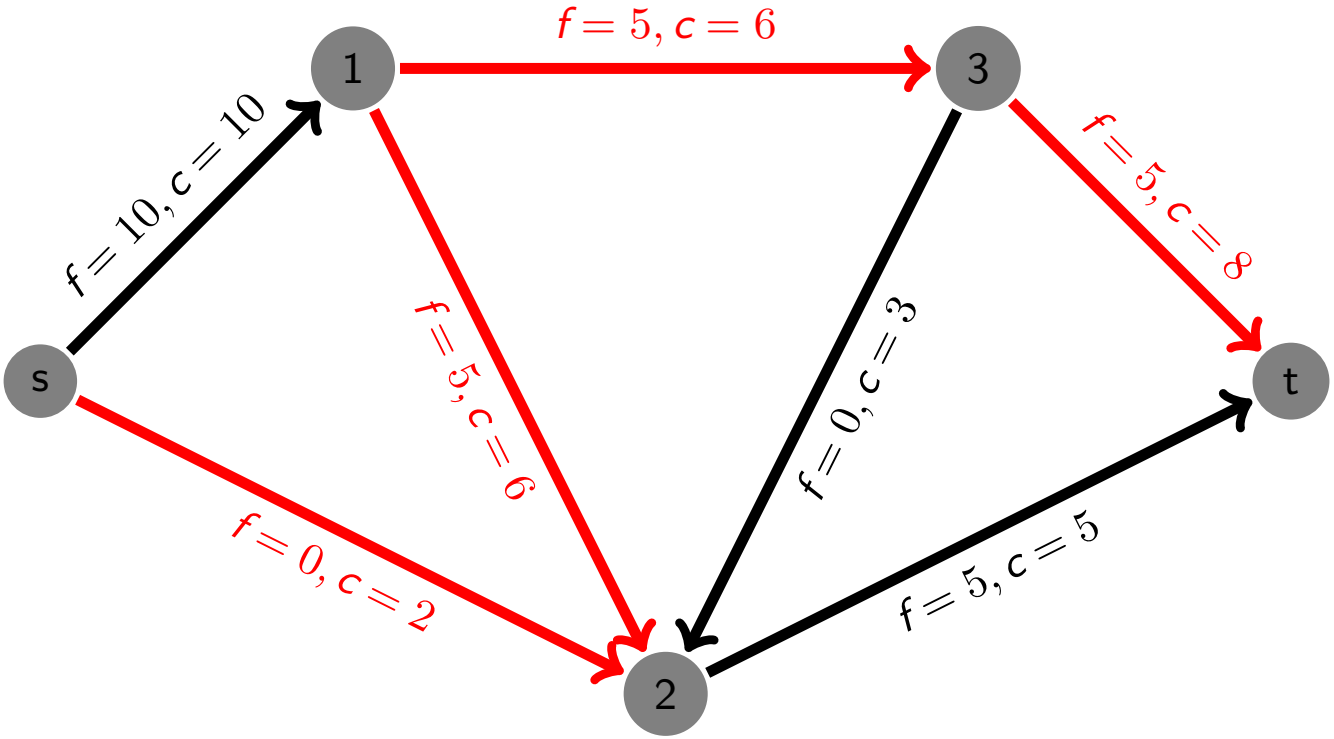
修正算法

修正算法



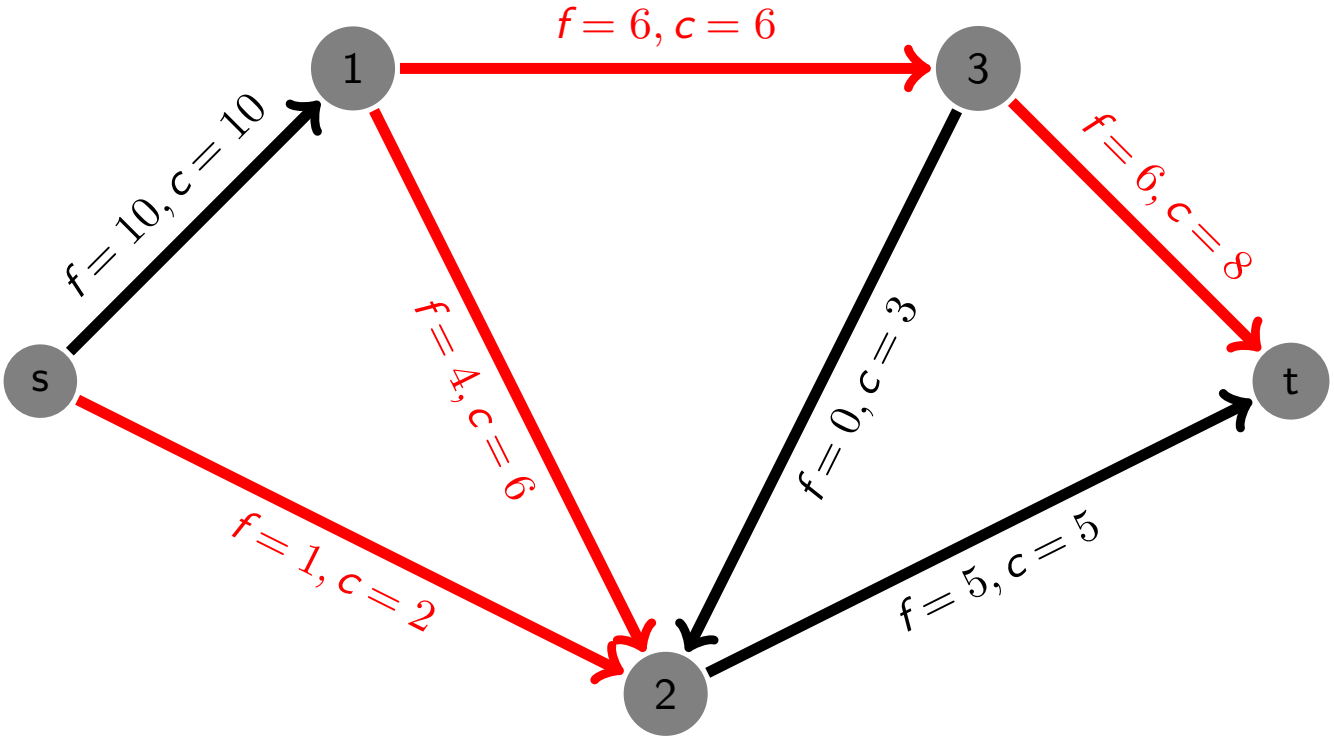
修正算法

沿着 $s \to 2 \to 1 \to 3 \to t$ 流 1



修正算法

沿着 $s \to 2 \to 1 \to 3 \to t$ 流 1



修正算法

修正算法

- 我们称在第一步中所考虑的 $f(e) < c(e)$ 的 e 和满足 $f(e) > 0$ 的 e 的反向边 $rev(e)$ 所组成的图为残余 (残量网络)，并称残余网络上的 $s - t$ 路径为增广路

修正算法

- 我们称在第一步中所考虑的 $f(e) < c(e)$ 的 e 和满足 $f(e) > 0$ 的 e 的反向边 $rev(e)$ 所组成的图为残余 (残量网络)，并称残余网络上的 $s - t$ 路径为增广路
- 不断在残余网络上找到增广路并进行增广，直至不存在增广路，每次增广所产生的新流量之和即为最大流

修正算法

- 我们称在第一步中所考虑的 $f(e) < c(e)$ 的 e 和满足 $f(e) > 0$ 的 e 的反向边 $rev(e)$ 所组成的图为残余 (残量网络)，并称残余网络上的 $s - t$ 路径为增广路
- 不断在残余网络上找到增广路并进行增广，直至不存在增广路，每次增广所产生的新流量之和即为最大流
- 下面我们来证明这个算法的正确性

最小割

最小割

- 为了证明上述算法正确性，我们需要先介绍割这一概念

最小割

- 为了证明上述算法正确性，我们需要先介绍割这一概念
- 图的割指的是对于某个 G 的顶点集合 $P \subset V$，从 P 出发指向 P 外部的那些 G 中的边的集合，我们可以把它记为割 $(P, V \setminus P)$

最小割

- 为了证明上述算法正确性，我们需要先介绍割这一概念
- 图的割指的是对于某个 G 的顶点集合 $P \subset V$，从 P 出发指向 P 外部的那些 G 中的边的集合，我们可以把它记为割 $(P, V \setminus P)$
- 割中的边的容量之和被称为割的容量，若有 $s \in P$ 且 $t \in V \setminus P$，那么此时的割又称为 $s - t$ 割

最小割

- 为了证明上述算法正确性，我们需要先介绍割这一概念
- 图的割指的是对于某个 G 的顶点集合 $P \subset V$，从 P 出发指向 P 外部的那些 G 中的边的集合，我们可以把它记为割 $(P, V \setminus P)$
- 割中的边的容量之和被称为割的容量，若有 $s \in P$ 且 $t \in V \setminus P$，那么此时的割又称为 $s-t$ 割
- 将图中 $s-t$ 割所包含的边都删去，就不再有 s 到 t 的路径了

最小割

- 为了证明上述算法正确性，我们需要先介绍割这一概念
- 图的割指的是对于某个 G 的顶点集合 $P \subset V$，从 P 出发指向 P 外部的那些 G 中的边的集合，我们可以把它记为割 $(P, V \setminus P)$
- 割中的边的容量之和被称为割的容量，若有 $s \in P$ 且 $t \in V \setminus P$，那么此时的割又称为 $s-t$ 割
- 将图中 $s-t$ 割所包含的边都删去，就不再有 s 到 t 的路径了
- 考虑如下问题：对于给定的网络流图，为了保证没有从 s 到 t 的路径，需要删去的边的总容量的最小值是多少。该问题即最小割问题

最小割

最小割

- 首先对于任意的 $s-t$ 流 F 和任意的 $s-t$ 割 $(P, V \setminus P)$，有

$$F\text{的流量} = s\text{出边的总流量}$$

最小割

- 首先对于任意的 $s-t$ 流 F 和任意的 $s-t$ 割 $(P, V \setminus P)$，有

$$F\text{的流量} = s\text{出边的总流量}$$

- 而对 $v \in P \setminus \{s\}$，又有

$$v\text{出边的总流量} = v\text{入边的总流量}$$

最小割

- 首先对于任意的 $s-t$ 流 F 和任意的 $s-t$ 割 $(P, V \setminus P)$，有

$$F\text{的流量} = s\text{出边的总流量}$$

- 而对 $v \in P \setminus \{s\}$，又有

$$v\text{出边的总流量} = v\text{入边的总流量}$$

- 所以有

$$\begin{aligned} F\text{的流量} &= (s+v)\text{出边总流量} - (s+v)\text{入边总流量} \\ &= P\text{出边总流量} - P\text{入边总流量} \end{aligned}$$

最小割

最小割

- 即

$$F\text{的流量} \leq \text{割的容量} = P\text{出边总流量}$$

最小割

- 即

$$F\text{的流量} \leq \text{割的容量} = P\text{出边总流量}$$

- 接下来我们考虑通过上述贪心算法求得的流 F

最小割

- 即

$$F\text{的流量} \leq \text{割的容量} = P\text{出边总流量}$$

- 接下来我们考虑通过上述贪心算法求得的流 F
- 记 F 对应的残余网络中从 s 可达的顶点 v 组成集合 S

最小割

- 即

$$F\text{的流量} \leq \text{割的容量} = P\text{出边总流量}$$

- 接下来我们考虑通过上述贪心算法求得的流 F
- 记 F 对应的残余网络中从 s 可达的顶点 v 组成集合 S
- 因为 F 对应的残余网络中不存在 $s-t$ 路径，所以 $(S, V \setminus S)$ 就是一个 $s-t$ 割

最小割

- 即

$$F\text{的流量} \leq \text{割的容量} = P\text{出边总流量}$$

- 接下来我们考虑通过上述贪心算法求得的流 F
- 记 F 对应的残余网络中从 s 可达的顶点 v 组成集合 S
- 因为 F 对应的残余网络中不存在 $s-t$ 路径，所以 $(S, V \setminus S)$ 就是一个 $s-t$ 割
- 根据 S 的定义，对包含在割中的边 e 应该有 $F(e) = c(e)$，而对 $V \setminus S$ 到 S 的边应该有 $F(e) = 0$，因此：

最小割

- 即

$$F\text{的流量} \leq \text{割的容量} = P\text{出边总流量}$$

- 接下来我们考虑通过上述贪心算法求得的流 F
- 记 F 对应的残余网络中从 s 可达的顶点 v 组成集合 S
- 因为 F 对应的残余网络中不存在 $s-t$ 路径，所以 $(S, V \setminus S)$ 就是一个 $s-t$ 割
- 根据 S 的定义，对包含在割中的边 e 应该有 $F(e) = c(e)$，而对 $V \setminus S$ 到 S 的边应该有 $F(e) = 0$，因此：
-

$$F\text{的流量} = S\text{出边总流量} - S\text{入边总流量} = \text{割的容量}$$

最小割

最小割

- 再有之前的不等式可知，F 就是最大流。同时 $(S, V \setminus S)$ 是一个最小割

最小割

- 再有之前的不等式可知，F 就是最大流。同时 $(S, V \setminus S)$ 是一个最小割
- 考虑反证，若它不是最小割，则 F 流量等于当前割并大于最小割，与上面不等式矛盾

最小割

- 再有之前的不等式可知，F 就是最大流。同时 $(S, V \setminus S)$ 是一个最小割
- 考虑反证，若它不是最小割，则 F 流量等于当前割并大于最小割，与上面不等式矛盾
- 同时这也是一个重要的性质：最大流等于最小割

最小割

- 再有之前的不等式可知，F 就是最大流。同时 $(S, V \setminus S)$ 是一个最小割
- 考虑反证，若它不是最小割，则 F 流量等于当前割并大于最小割，与上面不等式矛盾
- 同时这也是一个重要的性质：最大流等于最小割
- 从上面的算法也可以看出，若边的容量都是整数，则最大流与最小割也会是整数

算法实现

算法实现

- 朴素的实现上面所述的求解最大流的算法叫做 Ford-Fulkerson 算法

算法实现

- 朴素的实现上面所述的求解最大流的算法叫做 Ford-Fulkerson 算法
- 设求解的最大流的流量为 F，则它最多会进行 F 次增广，其复杂度为 $O(F|E|)$

算法实现

- 朴素的实现上面所述的求解最大流的算法叫做 Ford-Fulkerson 算法
- 设求解的最大流的流量为 F，则它最多会进行 F 次增广，其复杂度为 $O(F|E|)$
- 不过这个上界很松，所以实际效率还是比较快的

算法实现

- 朴素的实现上面所述的求解最大流的算法叫做 Ford-Fulkerson 算法
- 设求解的最大流的流量为 F，则它最多会进行 F 次增广，其复杂度为 $O(F|E|)$
- 不过这个上界很松，所以实际效率还是比较快的
- 事实上，还有许许多多不同的求解最大流问题的算法，它们主要有两类：增广路算法和预流推进算法

算法实现

- 朴素的实现上面所述的求解最大流的算法叫做 Ford-Fulkerson 算法
- 设求解的最大流的流量为 F，则它最多会进行 F 次增广，其复杂度为 $O(F|E|)$
- 不过这个上界很松，所以实际效率还是比较快的
- 事实上，还有许许多多不同的求解最大流问题的算法，它们主要有两类：增广路算法和预流推进算法
- 它们有不同的复杂度，不同的优缺点和对不同的图不同的实际运行效率。

算法实现

- 朴素的实现上面所述的求解最大流的算法叫做 Ford-Fulkerson 算法
- 设求解的最大流的流量为 F，则它最多会进行 F 次增广，其复杂度为 $O(F|E|)$
- 不过这个上界很松，所以实际效率还是比较快的
- 事实上，还有许许多多不同的求解最大流问题的算法，它们主要有两类：增广路算法和预流推进算法
- 它们有不同的复杂度，不同的优缺点和对不同的图不同的实际运行效率。
- 竞赛中常用 SAP 与 Dinic 两种最大流算法。下面来介绍一下 Dinic 算法

Dinic 算法

Dinic 算法

- Dinic 算法的主要思想就是每次寻找最短的增广路，并沿着它增广

Dinic 算法

- Dinic 算法的主要思想就是每次寻找最短的增广路，并沿着它增广
- 因为最短增广路的长度在增广过程中始终不会变短，所以无需每次都通过广搜来寻找最短增广路

Dinic 算法

- Dinic 算法的主要思想就是每次寻找最短的增广路，并沿着它增广
- 因为最短增广路的长度在增广过程中始终不会变短，所以无需每次都通过广搜来寻找最短增广路
- 可以先进行一次广搜，然后考虑由近距离顶点指向远距离顶点的边所组成的分层图，在上面进行深搜寻找最短增广路 (这里一次深搜就可以完成多次增广的工作，也称多路增广)

Dinic 算法

- Dinic 算法的主要思想就是每次寻找最短的增广路，并沿着它增广
- 因为最短增广路的长度在增广过程中始终不会变短，所以无需每次都通过广搜来寻找最短增广路
- 可以先进行一次广搜，然后考虑由近距离顶点指向远距离顶点的边所组成的分层图，在上面进行深搜寻找最短增广路 (这里一次深搜就可以完成多次增广的工作，也称多路增广)
- 如果在分层图上找不到新的增广路了 (此时我们得到了分层图所对应的阻塞流)，则说明最短增广路的长度变长了或不存在增广路了，于是重新构造新的分层图

Dinic 算法

Dinic 算法

- 设网络流图中顶点个数为 n 边数为 m，每一步构造分层图的复杂度为 $O(n+m)$

Dinic 算法

- 设网络流图中顶点个数为 n 边数为 m，每一步构造分层图的复杂度为 $O(n+m)$
- 每一步后最短路长度至少增加 1，由于增广路长度不超过 $n-1$，因此最多构造 $O(n)$ 次

Dinic 算法

- 设网络流图中顶点个数为 n 边数为 m，每一步构造分层图的复杂度为 $O(n+m)$
- 每一步后最短路长度至少增加 1，由于增广路长度不超过 $n-1$，因此最多构造 $O(n)$ 次
- 每次对分层图进行增广时，若避免对无用边进行多次检查 (即当前弧优化)，则可保证复杂度为 $O(nm)$，总时间复杂度为 $O(n^2m)$

Dinic 算法

- 设网络流图中顶点个数为 n 边数为 m，每一步构造分层图的复杂度为 $O(n+m)$
- 每一步后最短路长度至少增加 1，由于增广路长度不超过 $n-1$，因此最多构造 $O(n)$ 次
- 每次对分层图进行增广时，若避免对无用边进行多次检查 (即当前弧优化)，则可保证复杂度为 $O(nm)$，总时间复杂度为 $O(n^2m)$
- 实际应用中算法复杂度远达不到这个上界，即使图的规模较大也没问题 (常常能用来跑 10^4 甚至 10^5 级别的图)

Dinic 算法

- 设网络流图中顶点个数为 n 边数为 m，每一步构造分层图的复杂度为 $O(n+m)$
- 每一步后最短路长度至少增加 1，由于增广路长度不超过 $n-1$，因此最多构造 $O(n)$ 次
- 每次对分层图进行增广时，若避免对无用边进行多次检查 (即当前弧优化)，则可保证复杂度为 $O(nm)$，总时间复杂度为 $O(n^2m)$
- 实际应用中算法复杂度远达不到这个上界，即使图的规模较大也没问题 (常常能用来跑 10^4 甚至 10^5 级别的图)
- 当图是二分图时，即 $s \to X, X \to Y, Y \to t$，Dinic 算法效率为 $O(\sqrt{n}m)$

伪代码

伪代码

Algorithm 2 利用 BFS 将图分层

```
1: procedure BFS
2:     dis_{1...n} ← −1
3:     dis_s ← 0
4:     将 s 加入队尾
5:     while 队列非空 do
6:         u ← 队头元素
7:         删除队头元素
8:         for all u 的出边 e do
9:             v ← end_e
10:            if c(e) > 0 and dis_v = −1 then
11:                dis_v ← dis_u + 1
12:                将 v 加入队尾
13:            end if
14:        end for
15:    end while
16: end procedure
```

伪代码

伪代码

Algorithm 4 Dinic 多路增广

```
1: function DINIC(u, flow)
2:     if u = t then
3:         return flow
4:     end if
5:     res ← 0
6:     for all u 的出边 e do
7:         v ← end_e
8:         if c(e) > 0 and dis_u < dis_v then
9:             d ← Dinic(v, min(flow − res, c(e)))
10:            c(e) ← c(e) − d
11:            c(rev(e)) ← c(rev(e)) + d
12:            res ← res + d
13:        end if
14:    end for
15:    return res
16: end function
```

伪代码

伪代码

Algorithm 6 求解最大流

```
1: function MAX_FLOW(s, t)
2:     ans ← 0
3:     while true do
4:         Bfs()
5:         if dis_t = −1 then
6:             return ans
7:         end if
8:         ans ← ans + Dinic(s, ∞)
9:     end while
10:     return ans
11: end function
```

费用流

费用流

- 在网络流的基础上，每条边多了一个权值，单位流量费用 $d(e)$

费用流

- 在网络流的基础上，每条边多了一个权值，单位流量费用 $d(e)$
- 求从 s 到 t 流量最大的情况下，使得费用 $\sum f(e) \times d(e)$ 最小的问题称为最小费用最大流问题

费用流

- 在网络流的基础上，每条边多了一个权值，单位流量费用 $d(e)$
- 求从 s 到 t 流量最大的情况下，使得费用 $\sum f(e) \times d(e)$ 最小的问题称为最小费用最大流问题
- 算法流程与 Dinic 基本一致，只需要在增广时贪心地选取当前费用和最小的路径

费用流

- 在网络流的基础上，每条边多了一个权值，单位流量费用 $d(e)$
- 求从 s 到 t 流量最大的情况下，使得费用 $\sum f(e) \times d(e)$ 最小的问题称为最小费用最大流问题
- 算法流程与 Dinic 基本一致，只需要在增广时贪心地选取当前费用和最小的路径
- 需要注意反向边要将费用取反，因此做最短路时只能选用 SPFA

费用流

- 在网络流的基础上，每条边多了一个权值，单位流量费用 $d(e)$
- 求从 s 到 t 流量最大的情况下，使得费用 $\sum f(e) \times d(e)$ 最小的问题称为最小费用最大流问题
- 算法流程与 Dinic 基本一致，只需要在增广时贪心地选取当前费用和最小的路径
- 需要注意反向边要将费用取反，因此做最短路时只能选用 SPFA
- 可以用归纳法证明贪心的正确性

费用流

- 在网络流的基础上，每条边多了一个权值，单位流量费用 $d(e)$
- 求从 s 到 t 流量最大的情况下，使得费用 $\sum f(e) \times d(e)$ 最小的问题称为最小费用最大流问题
- 算法流程与 Dinic 基本一致，只需要在增广时贪心地选取当前费用和最小的路径
- 需要注意反向边要将费用取反，因此做最短路时只能选用 SPFA
- 可以用归纳法证明贪心的正确性
- 有时问题只需要求解最小费用 (流量不一定要满)，这只需要在增广过程中判断增广费用是否小于 0 即可

流表示方案

流表示方案

- 最大流最常用的构图方式

流表示方案

- 最大流最常用的构图方式
- 每一条 $s-t$ 流都对应着原题中的一个方案

流表示方案

- 最大流最常用的构图方式
- 每一条 $s-t$ 流都对应着原题中的一个方案
- 直观并且便于理解

二分图匹配

二分图匹配

- 给定一个二分图，请你选出一些边，使得选出的任意两条不同的边均没有公共端点，求最多能选出多少条边 (这些边就叫做匹配)，$n, m \leq 10^5$

二分图匹配

- 给定一个二分图，请你选出一些边，使得选出的任意两条不同的边均没有公共端点，求最多能选出多少条边 (这些边就叫做匹配)，$n, m \leq 10^5$
- 设图的两个集合为 (X, Y)

二分图匹配

- 给定一个二分图，请你选出一些边，使得选出的任意两条不同的边均没有公共端点，求最多能选出多少条边 (这些边就叫做匹配)，$n, m \leq 10^5$
- 设图的两个集合为 (X, Y)
- s 向 X 中的每个点 x_i 连容量为 1 的边

二分图匹配

- 给定一个二分图，请你选出一些边，使得选出的任意两条不同的边均没有公共端点，求最多能选出多少条边 (这些边就叫做匹配)，$n, m \leq 10^5$
- 设图的两个集合为 (X, Y)
- s 向 X 中的每个点 x_i 连容量为 1 的边
- Y 中的每个点 y_j 向 t 连容量为 1 的边

二分图匹配

- 给定一个二分图，请你选出一些边，使得选出的任意两条不同的边均没有公共端点，求最多能选出多少条边 (这些边就叫做匹配)，$n, m \leq 10^5$
- 设图的两个集合为 (X, Y)
- s 向 X 中的每个点 x_i 连容量为 1 的边
- Y 中的每个点 y_j 向 t 连容量为 1 的边
- 对于原图中的边 (x_i, y_j)，在网络流图中从 x_i 向 y_j 连容量为 1 的边

二分图匹配

- 给定一个二分图，请你选出一些边，使得选出的任意两条不同的边均没有公共端点，求最多能选出多少条边 (这些边就叫做匹配)，$n, m \leq 10^5$
- 设图的两个集合为 (X, Y)
- s 向 X 中的每个点 x_i 连容量为 1 的边
- Y 中的每个点 y_j 向 t 连容量为 1 的边
- 对于原图中的边 (x_i, y_j)，在网络流图中从 x_i 向 y_j 连容量为 1 的边
- 对建出的网络流图跑最大流即可

二分图匹配

- 给定一个二分图，请你选出一些边，使得选出的任意两条不同的边均没有公共端点，求最多能选出多少条边 (这些边就叫做匹配)，$n, m \leq 10^5$
- 设图的两个集合为 (X, Y)
- s 向 X 中的每个点 x_i 连容量为 1 的边
- Y 中的每个点 y_j 向 t 连容量为 1 的边
- 对于原图中的边 (x_i, y_j)，在网络流图中从 x_i 向 y_j 连容量为 1 的边
- 对建出的网络流图跑最大流即可
- 一条 $s \to x_i \to y_j \to t$ 的流代表着选 (x_i, y_j) 的边，并且由连边方式可知每个点最多会被选 1 次，符合题目条件

Dining

Dining

- n 个人到餐厅吃饭，共有 F 种食物和 D 种饮料，每个人各自有喜欢的食物和饮料，每种食物和饮料都只有一份，问最多能有几个人同时吃到喜欢的食物和饮料，$n, F, D \leq 100$

Dining

- n 个人到餐厅吃饭，共有 F 种食物和 D 种饮料，每个人各自有喜欢的食物和饮料，每种食物和饮料都只有一份，问最多能有几个人同时吃到喜欢的食物和饮料，$n, F, D \leq 100$
- 如果只有食物，那么这是个二分图最大匹配问题

Dining

- n 个人到餐厅吃饭，共有 F 种食物和 D 种饮料，每个人各自有喜欢的食物和饮料，每种食物和饮料都只有一份，问最多能有几个人同时吃到喜欢的食物和饮料，$n, F, D \leq 100$
- 如果只有食物，那么这是个二分图最大匹配问题
- 考虑将食物和饮料两种匹配结合起来

Dining

- n 个人到餐厅吃饭，共有 F 种食物和 D 种饮料，每个人各自有喜欢的食物和饮料，每种食物和饮料都只有一份，问最多能有几个人同时吃到喜欢的食物和饮料，$n, F, D \leq 100$
- 如果只有食物，那么这是个二分图最大匹配问题
- 考虑将食物和饮料两种匹配结合起来
- 源点 s 向每个食物连容量 1 的边

Dining

- n 个人到餐厅吃饭，共有 F 种食物和 D 种饮料，每个人各自有喜欢的食物和饮料，每种食物和饮料都只有一份，问最多能有几个人同时吃到喜欢的食物和饮料，$n, F, D \leq 100$
- 如果只有食物，那么这是个二分图最大匹配问题
- 考虑将食物和饮料两种匹配结合起来
- 源点 s 向每个食物连容量 1 的边
- 每个食物向喜欢他的人连容量 1 的边

Dining

- n 个人到餐厅吃饭，共有 F 种食物和 D 种饮料，每个人各自有喜欢的食物和饮料，每种食物和饮料都只有一份，问最多能有几个人同时吃到喜欢的食物和饮料，$n, F, D \leq 100$
- 如果只有食物，那么这是个二分图最大匹配问题
- 考虑将食物和饮料两种匹配结合起来
- 源点 s 向每个食物连容量 1 的边
- 每个食物向喜欢他的人连容量 1 的边
- 每个人向他喜欢的饮料连容量 1 的边

Dining

- n 个人到餐厅吃饭，共有 F 种食物和 D 种饮料，每个人各自有喜欢的食物和饮料，每种食物和饮料都只有一份，问最多能有几个人同时吃到喜欢的食物和饮料，$n, F, D \leq 100$
- 如果只有食物，那么这是个二分图最大匹配问题
- 考虑将食物和饮料两种匹配结合起来
- 源点 s 向每个食物连容量 1 的边
- 每个食物向喜欢他的人连容量 1 的边
- 每个人向他喜欢的饮料连容量 1 的边
- 每个饮料向汇点 t 连容量 1 的边

Dining

Dining

- 看起来这个图的最大流就是答案，因为每个流 $s \to F_i \to P_j \to D_k \to t$ 对应着人 P_j 吃到了喜欢的食物 F_i 与喜欢的饮料 D_k

Dining

- 看起来这个图的最大流就是答案，因为每个流 $s \to F_i \to P_j \to D_k \to t$ 对应着人 P_j 吃到了喜欢的食物 F_i 与喜欢的饮料 D_k
- 但是这个图的最大流的问题是，它可能会让一个人被多次计入答案

Dining

- 看起来这个图的最大流就是答案，因为每个流 $s \to F_i \to P_j \to D_k \to t$ 对应着人 P_j 吃到了喜欢的食物 F_i 与喜欢的饮料 D_k
- 但是这个图的最大流的问题是，它可能会让一个人被多次计入答案
- 解决方法是每个人拆成两个点 P_i, Q_i，食物向 P_i 连边，Q_i 向饮料连边

Dining

- 看起来这个图的最大流就是答案，因为每个流 $s \to F_i \to P_j \to D_k \to t$ 对应着人 P_j 吃到了喜欢的食物 F_i 与喜欢的饮料 D_k
- 但是这个图的最大流的问题是，它可能会让一个人被多次计入答案
- 解决方法是每个人拆成两个点 P_i, Q_i，食物向 P_i 连边，Q_i 向饮料连边
- P_i 向 Q_i 连容量为 1 的边，这样保证每个人只会被计入 1 次

Dining

- 看起来这个图的最大流就是答案，因为每个流 $s \to F_i \to P_j \to D_k \to t$ 对应着人 P_j 吃到了喜欢的食物 F_i 与喜欢的饮料 D_k
- 但是这个图的最大流的问题是，它可能会让一个人被多次计入答案
- 解决方法是每个人拆成两个点 P_i, Q_i，食物向 P_i 连边，Q_i 向饮料连边
- P_i 向 Q_i 连容量为 1 的边，这样保证每个人只会被计入 1 次
- 这也是在网络流问题中常用的拆点技巧，因为此时不仅边上有限制，点上也有限制 (每个点只能被选 1 次)

Dining

- 看起来这个图的最大流就是答案，因为每个流 $s \to F_i \to P_j \to D_k \to t$ 对应着人 P_j 吃到了喜欢的食物 F_i 与喜欢的饮料 D_k
- 但是这个图的最大流的问题是，它可能会让一个人被多次计入答案
- 解决方法是每个人拆成两个点 P_i, Q_i，食物向 P_i 连边，Q_i 向饮料连边
- P_i 向 Q_i 连容量为 1 的边，这样保证每个人只会被计入 1 次
- 这也是在网络流问题中常用的拆点技巧，因为此时不仅边上有限制，点上也有限制 (每个点只能被选 1 次)
- 拆点以后，将点的限制表示在拆出的点之间的边上，从而满足问题条件

Pig

Pig

- 有 m 个猪圈，每个猪圈里初始时有若干头猪。一开始所有猪圈都是关闭的。依次来了 n 个顾客，每个顾客分别会打开指定的几个猪圈，从中买若干头猪。每个顾客分别都有他能够买的数量的上限。每个顾客走后，他打开的那些猪圈中的猪，都可以被任意地调换到其它开着的猪圈里，然后所有猪圈重新关上。问总共最多能卖出多少头猪。$n \leq 100, m \leq 1000$

Pig

- 有 m 个猪圈，每个猪圈里初始时有若干头猪。一开始所有猪圈都是关闭的。依次来了 n 个顾客，每个顾客分别会打开指定的几个猪圈，从中买若干头猪。每个顾客分别都有他能够买的数量的上限。每个顾客走后，他打开的那些猪圈中的猪，都可以被任意地调换到其它开着的猪圈里，然后所有猪圈重新关上。问总共最多能卖出多少头猪。$n \leq 100, m \leq 1000$
- 按照一条 $s-t$ 表示卖出的一头猪的想法，不难构出下图：

Pig

- 有 m 个猪圈，每个猪圈里初始时有若干头猪。一开始所有猪圈都是关闭的。依次来了 n 个顾客，每个顾客分别会打开指定的几个猪圈，从中买若干头猪。每个顾客分别都有他能够买的数量的上限。每个顾客走后，他打开的那些猪圈中的猪，都可以被任意地调换到其它开着的猪圈里，然后所有猪圈重新关上。问总共最多能卖出多少头猪。$n \leq 100, m \leq 1000$
- 按照一条 $s-t$ 表示卖出的一头猪的想法，不难构出下图：
- 一个顾客就是一轮购买，每轮建 $n+1$ 个点，分别表示这轮的 n 个猪圈和顾客

Pig

- 有 m 个猪圈，每个猪圈里初始时有若干头猪。一开始所有猪圈都是关闭的。依次来了 n 个顾客，每个顾客分别会打开指定的几个猪圈，从中买若干头猪。每个顾客分别都有他能够买的数量的上限。每个顾客走后，他打开的那些猪圈中的猪，都可以被任意地调换到其它开着的猪圈里，然后所有猪圈重新关上。问总共最多能卖出多少头猪。$n \leq 100, m \leq 1000$
- 按照一条 $s-t$ 表示卖出的一头猪的想法，不难构出下图：
- 一个顾客就是一轮购买，每轮建 $n+1$ 个点，分别表示这轮的 n 个猪圈和顾客
- 源点 s 向第一轮的 n 个猪圈各连一条边，容量为初始数量

Pig

- 有 m 个猪圈，每个猪圈里初始时有若干头猪。一开始所有猪圈都是关闭的。依次来了 n 个顾客，每个顾客分别会打开指定的几个猪圈，从中买若干头猪。每个顾客分别都有他能够买的数量的上限。每个顾客走后，他打开的那些猪圈中的猪，都可以被任意地调换到其它开着的猪圈里，然后所有猪圈重新关上。问总共最多能卖出多少头猪。$n \leq 100, m \leq 1000$
- 按照一条 $s-t$ 表示卖出的一头猪的想法，不难构出下图：
- 一个顾客就是一轮购买，每轮建 $n+1$ 个点，分别表示这轮的 n 个猪圈和顾客
- 源点 s 向第一轮的 n 个猪圈各连一条边，容量为初始数量
- 每轮的顾客向汇点 t 连一条边，容量为顾客能买的数量上限

Pig

- 有 m 个猪圈，每个猪圈里初始时有若干头猪。一开始所有猪圈都是关闭的。依次来了 n 个顾客，每个顾客分别会打开指定的几个猪圈，从中买若干头猪。每个顾客分别都有他能够买的数量的上限。每个顾客走后，他打开的那些猪圈中的猪，都可以被任意地调换到其它开着的猪圈里，然后所有猪圈重新关上。问总共最多能卖出多少头猪。$n \leq 100, m \leq 1000$
- 按照一条 $s-t$ 表示卖出的一头猪的想法，不难构出下图：
- 一个顾客就是一轮购买，每轮建 $n+1$ 个点，分别表示这轮的 n 个猪圈和顾客
- 源点 s 向第一轮的 n 个猪圈各连一条边，容量为初始数量
- 每轮的顾客向汇点 t 连一条边，容量为顾客能买的数量上限
- 每轮中能被打开的猪圈都向这轮的顾客连容量 ∞ 的边

Pig

Pig

- 除最后一轮外，每一轮的 i 号猪圈向下一轮的 i 号猪圈连容量 ∞ 的边，表示剩下的猪留到下一轮

Pig

- 除最后一轮外，每一轮的 i 号猪圈向下一轮的 i 号猪圈连容量 ∞ 的边，表示剩下的猪留到下一轮
- 除最后一轮外，每轮能被打开的所有猪圈，到下一轮这些猪圈两两之间都要连容量 ∞，表示任意流通

Pig

- 除最后一轮外，每一轮的 i 号猪圈向下一轮的 i 号猪圈连容量 ∞ 的边，表示剩下的猪留到下一轮
- 除最后一轮外，每轮能被打开的所有猪圈，到下一轮这些猪圈两两之间都要连容量 ∞，表示任意流通
- 这张图的最大流就是答案，但点数有 10^5 个，边数更加多

Pig

- 除最后一轮外，每一轮的 i 号猪圈向下一轮的 i 号猪圈连容量 ∞ 的边，表示剩下的猪留到下一轮
- 除最后一轮外，每轮能被打开的所有猪圈，到下一轮这些猪圈两两之间都要连容量 ∞，表示任意流通
- 这张图的最大流就是答案，但点数有 10^5 个，边数更加多
- 尽管模型很直观，但是结点数与边数都太大，此时我们考虑用合并的方法优化 (简化) 这个网络流图

Pig

- 除最后一轮外，每一轮的 i 号猪圈向下一轮的 i 号猪圈连容量 ∞ 的边，表示剩下的猪留到下一轮
- 除最后一轮外，每轮能被打开的所有猪圈，到下一轮这些猪圈两两之间都要连容量 ∞，表示任意流通
- 这张图的最大流就是答案，但点数有 10^5 个，边数更加多
- 尽管模型很直观，但是结点数与边数都太大，此时我们考虑用合并的方法优化 (简化) 这个网络流图
- 可以发现对于一个猪圈，如果它连续几轮没被打开，那么它在每轮中的点的出边入边都是一样的，可以合并起来

Pig

- 除最后一轮外，每一轮的 i 号猪圈向下一轮的 i 号猪圈连容量 ∞ 的边，表示剩下的猪留到下一轮
- 除最后一轮外，每轮能被打开的所有猪圈，到下一轮这些猪圈两两之间都要连容量 ∞，表示任意流通
- 这张图的最大流就是答案，但点数有 10^5 个，边数更加多
- 尽管模型很直观，但是结点数与边数都太大，此时我们考虑用合并的方法优化 (简化) 这个网络流图
- 可以发现对于一个猪圈，如果它连续几轮没被打开，那么它在每轮中的点的出边入边都是一样的，可以合并起来
- 对于某两个顾客 i, j，如果它们能打开同一个猪圈，那么实际上 i 剩下没买的猪都可以给 j 买

Pig

- 除最后一轮外，每一轮的 i 号猪圈向下一轮的 i 号猪圈连容量 ∞ 的边，表示剩下的猪留到下一轮
- 除最后一轮外，每轮能被打开的所有猪圈，到下一轮这些猪圈两两之间都要连容量 ∞，表示任意流通
- 这张图的最大流就是答案，但点数有 10^5 个，边数更加多
- 尽管模型很直观，但是结点数与边数都太大，此时我们考虑用合并的方法优化 (简化) 这个网络流图
- 可以发现对于一个猪圈，如果它连续几轮没被打开，那么它在每轮中的点的出边入边都是一样的，可以合并起来
- 对于某两个顾客 i, j，如果它们能打开同一个猪圈，那么实际上 i 剩下没买的猪都可以给 j 买
- 有了以上两点我们发现所有猪圈的点都是无用的，因为猪圈中猪的连通可以被顾客之间的边所表示

Pig

Pig

- 考虑合并之后的新图，首先每个顾客建一个点来表示

Pig

- 考虑合并之后的新图，首先每个顾客建一个点来表示
- 对于每个猪圈的第一个顾客，源点 s 向他连一条容量为该猪圈中猪的初始数量的边 (若有多条这样的边，容量可以相加，合并成一条边)

Pig

- 考虑合并之后的新图，首先每个顾客建一个点来表示
- 对于每个猪圈的第一个顾客，源点 s 向他连一条容量为该猪圈中猪的初始数量的边 (若有多条这样的边，容量可以相加，合并成一条边)
- 对于每个猪圈，假设有 k 个顾客 p_i 能打开它，那么对于第 $i(i < k)$ 个顾客，p_i 向 p_{i+1} 连一条容量 ∞ 的边

Pig

- 考虑合并之后的新图，首先每个顾客建一个点来表示
- 对于每个猪圈的第一个顾客，源点 s 向他连一条容量为该猪圈中猪的初始数量的边 (若有多条这样的边，容量可以相加，合并成一条边)
- 对于每个猪圈，假设有 k 个顾客 p_i 能打开它，那么对于第 $i(i < k)$ 个顾客，p_i 向 p_{i+1} 连一条容量 ∞ 的边
- 每个顾客向汇点连一条容量为购买数量上限的边

Pig

- 考虑合并之后的新图，首先每个顾客建一个点来表示
- 对于每个猪圈的第一个顾客，源点 s 向他连一条容量为该猪圈中猪的初始数量的边 (若有多条这样的边，容量可以相加，合并成一条边)
- 对于每个猪圈，假设有 k 个顾客 p_i 能打开它，那么对于第 $i(i < k)$ 个顾客，p_i 向 p_{i+1} 连一条容量 ∞ 的边
- 每个顾客向汇点连一条容量为购买数量上限的边
- 新图点数最多只有 102 个，边数最多 10^4 条

Pig

- 考虑合并之后的新图，首先每个顾客建一个点来表示
- 对于每个猪圈的第一个顾客，源点 s 向他连一条容量为该猪圈中猪的初始数量的边 (若有多条这样的边，容量可以相加，合并成一条边)
- 对于每个猪圈，假设有 k 个顾客 p_i 能打开它，那么对于第 $i(i < k)$ 个顾客，p_i 向 p_{i+1} 连一条容量 ∞ 的边
- 每个顾客向汇点连一条容量为购买数量上限的边
- 新图点数最多只有 102 个，边数最多 10^4 条
- 每个 $s-t$ 流仍然是一个猪的卖出方案，但是省去了具体的流通过程，即我们知道它一定能被这个人买到，所以中间的过程我们并不关心

Pig

- 考虑合并之后的新图，首先每个顾客建一个点来表示
- 对于每个猪圈的第一个顾客，源点 s 向他连一条容量为该猪圈中猪的初始数量的边 (若有多条这样的边，容量可以相加，合并成一条边)
- 对于每个猪圈，假设有 k 个顾客 p_i 能打开它，那么对于第 $i(i < k)$ 个顾客，p_i 向 p_{i+1} 连一条容量 ∞ 的边
- 每个顾客向汇点连一条容量为购买数量上限的边
- 新图点数最多只有 102 个，边数最多 10^4 条
- 每个 $s-t$ 流仍然是一个猪的卖出方案，但是省去了具体的流通过程，即我们知道它一定能被这个人买到，所以中间的过程我们并不关心
- 构造最直观的模型，并通过题目条件慢慢简化模型，这也是一个常用的网络流解决办法

最小割模型

最小割模型

- 之前证明过的重要结论：最大流等于最小割

最小割模型

- 之前证明过的重要结论：最大流等于最小割
- 有的问题从最大流角度不好考虑，但从最小割的角度考虑更加直观，而求最小割便转为求解最大流，因此这也是一个常见的模型

最小割模型

- 之前证明过的重要结论：最大流等于最小割
- 有的问题从最大流角度不好考虑，但从最小割的角度考虑更加直观，而求最小割便转为求解最大流，因此这也是一个常见的模型
- 最小割有好几种经典模型，需要选手理解清楚

二分图最小顶点覆盖

二分图最小顶点覆盖

- 给定一个二分图，现在你要选出一些点，使得每条边至少有一个端点被选出，求最小要选多少个点 (这个点集被称为覆盖集)，$n, m \leq 10^5$

二分图最小顶点覆盖

- 给定一个二分图，现在你要选出一些点，使得每条边至少有一个端点被选出，求最小要选多少个点 (这个点集被称为覆盖集)，$n, m \leq 10^5$
- s 向 X_i 连容量 1 的边，Y_j 向 t 连容量 1 的边

二分图最小顶点覆盖

- 给定一个二分图，现在你要选出一些点，使得每条边至少有一个端点被选出，求最小要选多少个点 (这个点集被称为覆盖集)，$n, m \leq 10^5$
- s 向 X_i 连容量 1 的边，Y_j 向 t 连容量 1 的边
- X_i 向 Y_j 连容量 ∞ 的边

二分图最小顶点覆盖

- 给定一个二分图，现在你要选出一些点，使得每条边至少有一个端点被选出，求最小要选多少个点 (这个点集被称为覆盖集)，$n, m \leq 10^5$
- s 向 X_i 连容量 1 的边，Y_j 向 t 连容量 1 的边
- X_i 向 Y_j 连容量 ∞ 的边
- 考虑该图一个容量小于 ∞ 的割

二分图最小顶点覆盖

- 给定一个二分图，现在你要选出一些点，使得每条边至少有一个端点被选出，求最小要选多少个点 (这个点集被称为覆盖集)，$n, m \leq 10^5$
- s 向 X_i 连容量 1 的边，Y_j 向 t 连容量 1 的边
- X_i 向 Y_j 连容量 ∞ 的边
- 考虑该图一个容量小于 ∞ 的割
- 每条割边都是 (s, X_i) 或是 (Y_j, t)，取这些 X_i, Y_j 作为覆盖集

二分图最小顶点覆盖

- 给定一个二分图，现在你要选出一些点，使得每条边至少有一个端点被选出，求最小要选多少个点 (这个点集被称为覆盖集)，$n, m \leq 10^5$
- s 向 X_i 连容量 1 的边，Y_j 向 t 连容量 1 的边
- X_i 向 Y_j 连容量 ∞ 的边
- 考虑该图一个容量小于 ∞ 的割
- 每条割边都是 (s, X_i) 或是 (Y_j, t)，取这些 X_i, Y_j 作为覆盖集
- 边 (X_i, Y_j) 未被覆盖，这意味着 (s, X_i) 与 (Y_j, t) 都不是割边，即存在 $s - t$ 路径，与假设矛盾

二分图最小顶点覆盖

二分图最小顶点覆盖

- 因此这张图的一个 $s-t$ 割对应着一个顶点覆盖，割容量等于点集大小

二分图最小顶点覆盖

- 因此这张图的一个 $s-t$ 割对应着一个顶点覆盖，割容量等于点集大小
- 容易证明一个顶点覆盖也能对应着一个 $s-t$ 割

二分图最小顶点覆盖

- 因此这张图的一个 $s-t$ 割对应着一个顶点覆盖，割容量等于点集大小
- 容易证明一个顶点覆盖也能对应着一个 $s-t$ 割
- 这张图的最小割，也就是最大流即是答案

二分图最小顶点覆盖

- 因此这张图的一个 $s-t$ 割对应着一个顶点覆盖，割容量等于点集大小
- 容易证明一个顶点覆盖也能对应着一个 $s-t$ 割
- 这张图的最小割，也就是最大流即是答案
- 容易发现这个构图与二分图最大匹配一致

二分图最小顶点覆盖

- 因此这张图的一个 $s-t$ 割对应着一个顶点覆盖，割容量等于点集大小
- 容易证明一个顶点覆盖也能对应着一个 $s-t$ 割
- 这张图的最小割，也就是最大流即是答案
- 容易发现这个构图与二分图最大匹配一致
- 结论：二分图中，最小顶点覆盖等于最大匹配

二分图最大独立集

二分图最大独立集

- 给定一个二分图，现在你要选出一些点，使得这些点两两之间没有边相连，求最多能选出多少个点 (这个点集被称为独立集)，$n, m \leq 10^5$

二分图最大独立集

- 给定一个二分图，现在你要选出一些点，使得这些点两两之间没有边相连，求最多能选出多少个点 (这个点集被称为独立集)，$n, m \leq 10^5$
- 结论：最小顶点覆盖 + 最大独立集 = 总点数

二分图最大独立集

- 给定一个二分图，现在你要选出一些点，使得这些点两两之间没有边相连，求最多能选出多少个点 (这个点集被称为独立集)，$n, m \leq 10^5$
- 结论：最小顶点覆盖 + 最大独立集 = 总点数
- 将独立集从点集中删去，剩下的点集一定是个覆盖集

二分图最大独立集

- 给定一个二分图，现在你要选出一些点，使得这些点两两之间没有边相连，求最多能选出多少个点 (这个点集被称为独立集)，$n, m \leq 10^5$
- 结论：最小顶点覆盖 + 最大独立集 = 总点数
- 将独立集从点集中删去，剩下的点集一定是个覆盖集
- 否则有一条边的两个端点都被删除，而这两个端点不可能有边 (独立集)，即出现矛盾

常见结论

常见结论

- 对于图 $G = (V, E)$

常见结论

- 对于图 $G = (V, E)$
- 匹配：G 中两两没有公共端点的边集合 $M \subseteq E$

常见结论

- 对于图 $G = (V, E)$
- 匹配：G 中两两没有公共端点的边集合 $M \subseteq E$
- 边覆盖：G 中任意顶点都至少是 F 中某条边的端点的边集合 $F \subseteq E$

常见结论

- 对于图 $G = (V, E)$
- 匹配：G 中两两没有公共端点的边集合 $M \subseteq E$
- 边覆盖：G 中任意顶点都至少是 F 中某条边的端点的边集合 $F \subseteq E$
- 独立集：在 G 中两两互不相连的顶点集合 $S \subseteq V$

常见结论

- 对于图 $G = (V, E)$
- 匹配：G 中两两没有公共端点的边集合 $M \subseteq E$
- 边覆盖：G 中任意顶点都至少是 F 中某条边的端点的边集合 $F \subseteq E$
- 独立集：在 G 中两两互不相连的顶点集合 $S \subseteq V$
- 顶点覆盖：G 中的任意边都至少有一个端点属于 S 的顶点集合 $S \subseteq V$

常见结论

- 对于图 $G = (V, E)$
- 匹配：G 中两两没有公共端点的边集合 $M \subseteq E$
- 边覆盖：G 中任意顶点都至少是 F 中某条边的端点的边集合 $F \subseteq E$
- 独立集：在 G 中两两互不相连的顶点集合 $S \subseteq V$
- 顶点覆盖：G 中的任意边都至少有一个端点属于 S 的顶点集合 $S \subseteq V$
- 对于不存在孤立点的图：最大匹配 + 最小边覆盖 = 总点数

常见结论

- 对于图 $G = (V, E)$
- 匹配：G 中两两没有公共端点的边集合 $M \subseteq E$
- 边覆盖：G 中任意顶点都至少是 F 中某条边的端点的边集合 $F \subseteq E$
- 独立集：在 G 中两两互不相连的顶点集合 $S \subseteq V$
- 顶点覆盖：G 中的任意边都至少有一个端点属于 S 的顶点集合 $S \subseteq V$
- 对于不存在孤立点的图：最大匹配 + 最小边覆盖 = 总点数
- 最大独立集 + 最小顶点覆盖 = 总点数

常见结论

- 对于图 $G = (V, E)$
- 匹配：G 中两两没有公共端点的边集合 $M \subseteq E$
- 边覆盖：G 中任意顶点都至少是 F 中某条边的端点的边集合 $F \subseteq E$
- 独立集：在 G 中两两互不相连的顶点集合 $S \subseteq V$
- 顶点覆盖：G 中的任意边都至少有一个端点属于 S 的顶点集合 $S \subseteq V$
- 对于不存在孤立点的图：最大匹配 + 最小边覆盖 = 总点数
- 最大独立集 + 最小顶点覆盖 = 总点数
- 二分图中，最小顶点覆盖等于最大匹配

二分图最大点权独立集

二分图最大点权独立集

- 给定一个二分图，每个点有点权，现在你要选出一些点，使得这些点两两之间没有边相连，求选出的点的权值之和最大能是多少，$n, m \leq 10^5$

二分图最大点权独立集

- 给定一个二分图，每个点有点权，现在你要选出一些点，使得这些点两两之间没有边相连，求选出的点的权值之和最大能是多少，$n, m \leq 10^5$
- 结论：二分图中，最大点权独立集 + 最小点权覆盖集 = 总权值

二分图最大点权独立集

- 给定一个二分图，每个点有点权，现在你要选出一些点，使得这些点两两之间没有边相连，求选出的点的权值之和最大能是多少，$n, m \leq 10^5$
- 结论：二分图中，最大点权独立集 + 最小点权覆盖集 = 总权值
- 最小点权覆盖集建图方式与最小点覆盖一致，只需要把容量将 1 变成点权即可

二分图最大点权独立集

- 给定一个二分图，每个点有点权，现在你要选出一些点，使得这些点两两之间没有边相连，求选出的点的权值之和最大能是多少，$n, m \leq 10^5$
- 结论：二分图中，最大点权独立集 + 最小点权覆盖集 = 总权值
- 最小点权覆盖集建图方式与最小点覆盖一致，只需要把容量将 1 变成点权即可
- 因此答案为总权值减去最小割，即总权值减去最大流

How to earn more

How to earn more

- 有 m 个项目和 n 个员工。做项目 i 可以获得 A_i 元，但是必须雇用若干个指定的员工。雇用员工 j 需要花费 B_j 元，且一旦雇用，员工 j 可以参加多个项目的开发。问经过合理的项目取舍，最多能挣多少钱。$n, m \leq 100$

How to earn more

- 有 m 个项目和 n 个员工。做项目 i 可以获得 A_i 元，但是必须雇用若干个指定的员工。雇用员工 j 需要花费 B_j 元，且一旦雇用，员工 j 可以参加多个项目的开发。问经过合理的项目取舍，最多能挣多少钱。$n, m \leq 100$
- 将员工和项目都看做点，项目向其需要雇佣的员工连边

How to earn more

- 有 m 个项目和 n 个员工。做项目 i 可以获得 A_i 元，但是必须雇用若干个指定的员工。雇用员工 j 需要花费 B_j 元，且一旦雇用，员工 j 可以参加多个项目的开发。问经过合理的项目取舍，最多能挣多少钱。$n, m \leq 100$
- 将员工和项目都看做点，项目向其需要雇佣的员工连边
- 题目变为给定一张有向图 $G = (V, E)$，每个点有点权，且不存在负权点连向正权点的边。现在要你选择一个点集 V'，满足
  $\forall e = (x, y) \in E, x \in V' \rightarrow y \in V'$

How to earn more

- 有 m 个项目和 n 个员工。做项目 i 可以获得 A_i 元，但是必须雇用若干个指定的员工。雇用员工 j 需要花费 B_j 元，且一旦雇用，员工 j 可以参加多个项目的开发。问经过合理的项目取舍，最多能挣多少钱。$n, m \leq 100$
- 将员工和项目都看做点，项目向其需要雇佣的员工连边
- 题目变为给定一张有向图 $G = (V, E)$，每个点有点权，且不存在负权点连向正权点的边。现在要你选择一个点集 V'，满足
  $\forall e = (x, y) \in E, x \in V' \rightarrow y \in V'$
- 这个问题即为最大权闭合子图问题

How to earn more

How to earn more

- 考虑如下的网络流建图模型：

How to earn more

- 考虑如下的网络流建图模型：
- 源点向所有正权点 P_i 连容量为权值的边

How to earn more

- 考虑如下的网络流建图模型：
- 源点向所有正权点 P_i 连容量为权值的边
- 所有负权点 Q_j 向汇点连容量为权值相反数的边

How to earn more

- 考虑如下的网络流建图模型：
- 源点向所有正权点 P_i 连容量为权值的边
- 所有负权点 Q_j 向汇点连容量为权值相反数的边
- 对于原图中的边 $e = (x, y)$，从 x 到 y 连一条容量为 ∞ 的边

How to earn more

- 考虑如下的网络流建图模型：
- 源点向所有正权点 P_i 连容量为权值的边
- 所有负权点 Q_j 向汇点连容量为权值相反数的边
- 对于原图中的边 $e = (x, y)$，从 x 到 y 连一条容量为 ∞ 的边
- 答案为正权总权值减去最小割

How to earn more

How to earn more

- 从割的意义来理解，得到最大的收益即舍弃最小的收益 (割)

How to earn more

- 从割的意义来理解，得到最大的收益即舍弃最小的收益 (割)
- 对于一个项目与一个员工，在网络流图中存在一条 $s \to P_i \to Q_j \to t$ 的路径

How to earn more

- 从割的意义来理解，得到最大的收益即舍弃最小的收益 (割)
- 对于一个项目与一个员工，在网络流图中存在一条 $s \to P_i \to Q_j \to t$ 的路径
- 由于 $P_i \to Q_j$ 容量为 ∞，所以它不可能被割

How to earn more

- 从割的意义来理解，得到最大的收益即舍弃最小的收益 (割)
- 对于一个项目与一个员工，在网络流图中存在一条 $s \to P_i \to Q_j \to t$ 的路径
- 由于 $P_i \to Q_j$ 容量为 ∞，所以它不可能被割
- 要么 $s \to P_i$ 被割，表示舍弃这个项目；要么 $Q_j \to t$ 被割，表示花钱雇佣它

How to earn more

- 从割的意义来理解，得到最大的收益即舍弃最小的收益 (割)
- 对于一个项目与一个员工，在网络流图中存在一条 $s \to P_i \to Q_j \to t$ 的路径
- 由于 $P_i \to Q_j$ 容量为 ∞，所以它不可能被割
- 要么 $s \to P_i$ 被割，表示舍弃这个项目；要么 $Q_j \to t$ 被割，表示花钱雇佣它
- 因此答案就是正权总权值减去最小的损失 (即最小割)

How to earn more

- 从割的意义来理解，得到最大的收益即舍弃最小的收益 (割)
- 对于一个项目与一个员工，在网络流图中存在一条 $s \to P_i \to Q_j \to t$ 的路径
- 由于 $P_i \to Q_j$ 容量为 ∞，所以它不可能被割
- 要么 $s \to P_i$ 被割，表示舍弃这个项目；要么 $Q_j \to t$ 被割，表示花钱雇佣它
- 因此答案就是正权总权值减去最小的损失 (即最小割)
- 需要注意的是若图中有负权点连向正权点的情况，则此建图模型不适用

Dual Core CPU

Dual Core CPU

- 有 n 个任务，两台机器。任务 i 在第一台机器上执行的花费为 A_i，在第二台机器上执行的花费为 B_i。有 m 个关系 (a_k, b_k)，如果 a_k, b_k) 两个任务在不同机器上执行会产生额外的代价 w_k。问，如何分配这 n 个任务使得总花费最少。$n \leq 2 \times 10^4, m \leq 2 \times 10^5$

Dual Core CPU

- 有 n 个任务，两台机器。任务 i 在第一台机器上执行的花费为 A_i，在第二台机器上执行的花费为 B_i。有 m 个关系 (a_k, b_k)，如果 a_k, b_k) 两个任务在不同机器上执行会产生额外的代价 w_k。问，如何分配这 n 个任务使得总花费最少。$n \leq 2 \times 10^4, m \leq 2 \times 10^5$
- 用最小的代价将对象分为两个集合，这类问题常常可以转化成最小割模型后解决，此题就是个经典例子

Dual Core CPU

- 有 n 个任务，两台机器。任务 i 在第一台机器上执行的花费为 A_i，在第二台机器上执行的花费为 B_i。有 m 个关系 (a_k, b_k)，如果 a_k, b_k) 两个任务在不同机器上执行会产生额外的代价 w_k。问，如何分配这 n 个任务使得总花费最少。$n \leq 2 \times 10^4, m \leq 2 \times 10^5$
- 用最小的代价将对象分为两个集合，这类问题常常可以转化成最小割模型后解决，此题就是个经典例子
- 源点 s 向每个任务连容量为 A_i 的边

Dual Core CPU

- 有 n 个任务，两台机器。任务 i 在第一台机器上执行的花费为 A_i，在第二台机器上执行的花费为 B_i。有 m 个关系 (a_k, b_k)，如果 a_k, b_k) 两个任务在不同机器上执行会产生额外的代价 w_k。问，如何分配这 n 个任务使得总花费最少。$n \leq 2 \times 10^4, m \leq 2 \times 10^5$
- 用最小的代价将对象分为两个集合，这类问题常常可以转化成最小割模型后解决，此题就是个经典例子
- 源点 s 向每个任务连容量为 A_i 的边
- 每个任务向汇点 t 连容量为 B_i 的边

Dual Core CPU

- 有 n 个任务，两台机器。任务 i 在第一台机器上执行的花费为 A_i，在第二台机器上执行的花费为 B_i。有 m 个关系 (a_k, b_k)，如果 a_k, b_k) 两个任务在不同机器上执行会产生额外的代价 w_k。问，如何分配这 n 个任务使得总花费最少。$n \leq 2 \times 10^4, m \leq 2 \times 10^5$
- 用最小的代价将对象分为两个集合，这类问题常常可以转化成最小割模型后解决，此题就是个经典例子
- 源点 s 向每个任务连容量为 A_i 的边
- 每个任务向汇点 t 连容量为 B_i 的边
- 每对关系 a_k 和 b_k 之间互连一条容量为 w_k 的边

Dual Core CPU

- 有 n 个任务，两台机器。任务 i 在第一台机器上执行的花费为 A_i，在第二台机器上执行的花费为 B_i。有 m 个关系 (a_k, b_k)，如果 a_k, b_k) 两个任务在不同机器上执行会产生额外的代价 w_k。问，如何分配这 n 个任务使得总花费最少。$n \leq 2 \times 10^4, m \leq 2 \times 10^5$
- 用最小的代价将对象分为两个集合，这类问题常常可以转化成最小割模型后解决，此题就是个经典例子
- 源点 s 向每个任务连容量为 A_i 的边
- 每个任务向汇点 t 连容量为 B_i 的边
- 每对关系 a_k 和 b_k 之间互连一条容量为 w_k 的边
- $\sum A_i + \sum B_i$ 减去最小割即是答案

Dual Core CPU

Dual Core CPU

- 考虑割的情况

Dual Core CPU

- 考虑割的情况
- 如果割的是 A_i，这表示这个任务放在第二台机器上执行

Dual Core CPU

- 考虑割的情况
- 如果割的是 A_i，这表示这个任务放在第二台机器上执行
- 如果割的是 B_i，这表示这个任务放在第一台机器上执行

Dual Core CPU

- 考虑割的情况
- 如果割的是 A_i，这表示这个任务放在第二台机器上执行
- 如果割的是 B_i，这表示这个任务放在第一台机器上执行
- 如果割的是 w_k，这表示 a_k 与 b_k 在不同机器上执行，这对关系需要花费额外的代价

Dual Core CPU

- 考虑割的情况
- 如果割的是 A_i，这表示这个任务放在第二台机器上执行
- 如果割的是 B_i，这表示这个任务放在第一台机器上执行
- 如果割的是 w_k，这表示 a_k 与 b_k 在不同机器上执行，这对关系需要花费额外的代价
- 这类问题只需要把代价都在容量上体现出来，并用割的方式理解即可

文理分科

文理分科

- 一个 $n \times m$ 的矩阵的同学要分文理科，你现在要将满意值最大化。对于第 i 行第 j 列的同学，若他选择文科，则有 $A_{i,j}$ 的满意值，若他选择理科则有 $B_{i,j}$ 的满意值；若他与相邻的同学都选了文科则有 $C_{i,j}$ 的满意值，若他与相邻的同学都选择了理科则有 $D_{i,j}$ 的满意值。$n, m \leq 100$

文理分科

- 一个 $n \times m$ 的矩阵的同学要分文理科，你现在要将满意值最大化。对于第 i 行第 j 列的同学，若他选择文科，则有 $A_{i,j}$ 的满意值，若他选择理科则有 $B_{i,j}$ 的满意值；若他与相邻的同学都选了文科则有 $C_{i,j}$ 的满意值，若他与相邻的同学都选择了理科则有 $D_{i,j}$ 的满意值。$n, m \leq 100$
- 首先将矩阵进行黑白染色，这样一个点与周围相邻的四个点分处两个集合

文理分科

- 一个 $n \times m$ 的矩阵的同学要分文理科，你现在要将满意值最大化。对于第 i 行第 j 列的同学，若他选择文科，则有 $A_{i,j}$ 的满意值，若他选择理科则有 $B_{i,j}$ 的满意值；若他与相邻的同学都选了文科则有 $C_{i,j}$ 的满意值，若他与相邻的同学都选择了理科则有 $D_{i,j}$ 的满意值。$n, m \leq 100$
- 首先将矩阵进行黑白染色，这样一个点与周围相邻的四个点分处两个集合
- 一个点的收益与周围四个点有关，但直接通过网络流无法进行判断

文理分科

- 一个 $n \times m$ 的矩阵的同学要分文理科，你现在要将满意值最大化。对于第 i 行第 j 列的同学，若他选择文科，则有 $A_{i,j}$ 的满意值，若他选择理科则有 $B_{i,j}$ 的满意值；若他与相邻的同学都选了文科则有 $C_{i,j}$ 的满意值，若他与相邻的同学都选择了理科则有 $D_{i,j}$ 的满意值。$n, m \leq 100$
- 首先将矩阵进行黑白染色，这样一个点与周围相邻的四个点分处两个集合
- 一个点的收益与周围四个点有关，但直接通过网络流无法进行判断
- 考虑新建点，表示周围的点是否都选了文/理科

文理分科

- 一个 $n \times m$ 的矩阵的同学要分文理科，你现在要将满意值最大化。对于第 i 行第 j 列的同学，若他选择文科，则有 $A_{i,j}$ 的满意值，若他选择理科则有 $B_{i,j}$ 的满意值；若他与相邻的同学都选了文科则有 $C_{i,j}$ 的满意值，若他与相邻的同学都选择了理科则有 $D_{i,j}$ 的满意值。$n, m \leq 100$
- 首先将矩阵进行黑白染色，这样一个点与周围相邻的四个点分处两个集合
- 一个点的收益与周围四个点有关，但直接通过网络流无法进行判断
- 考虑新建点，表示周围的点是否都选了文/理科
- 新点向原来矩阵中的点连容量 ∞ 的边，表示要么舍弃这个额外收益，要么他们全选文/理科

文理分科

- 一个 $n \times m$ 的矩阵的同学要分文理科，你现在要将满意值最大化。对于第 i 行第 j 列的同学，若他选择文科，则有 $A_{i,j}$ 的满意值，若他选择理科则有 $B_{i,j}$ 的满意值；若他与相邻的同学都选了文科则有 $C_{i,j}$ 的满意值，若他与相邻的同学都选择了理科则有 $D_{i,j}$ 的满意值。$n, m \leq 100$
- 首先将矩阵进行黑白染色，这样一个点与周围相邻的四个点分处两个集合
- 一个点的收益与周围四个点有关，但直接通过网络流无法进行判断
- 考虑新建点，表示周围的点是否都选了文/理科
- 新点向原来矩阵中的点连容量 ∞ 的边，表示要么舍弃这个额外收益，要么他们全选文/理科
- 其余连边与上一题类似

离散变量模型

离散变量模型

- 若干个变量需要决定取值，每个变量有若干个不同的取值 $x = \{0, 1, 2, \cdots, m\}$，并且两个变量之间会互相影响取值

离散变量模型

- 若干个变量需要决定取值，每个变量有若干个不同的取值 $x = \{0, 1, 2, \cdots, m\}$，并且两个变量之间会互相影响取值
- 不同的取值贡献也不一样，一般贡献会和相邻两个取值有关

离散变量模型

- 若干个变量需要决定取值，每个变量有若干个不同的取值 $x = \{0, 1, 2, \cdots, m\}$，并且两个变量之间会互相影响取值
- 不同的取值贡献也不一样，一般贡献会和相邻两个取值有关
- 建图方式一般为，将一个点 x 拆成 $m + 1$ 个点，并连成从 s 到 t 的一条链

离散变量模型

- 若干个变量需要决定取值，每个变量有若干个不同的取值 $x = \{0, 1, 2, \cdots, m\}$，并且两个变量之间会互相影响取值
- 不同的取值贡献也不一样，一般贡献会和相邻两个取值有关
- 建图方式一般为，将一个点 x 拆成 $m + 1$ 个点，并连成从 s 到 t 的一条链
- 跑最小割，割在哪条边就是选取了哪个取值，变量之间的影响用两条链之间的边表示

RIN

RIN

- 有 m 门课可以在 n 个学期内学习，第 i 门课在第 j 个学期学习的收益是 $X_{i,j}$，一个学期可以学多门课，有的课之间有依赖关系，即必须先学 a 再学 b，求最大收益。$n, m \leq 100$

RIN

- 有 m 门课可以在 n 个学期内学习，第 i 门课在第 j 个学期学习的收益是 $X_{i,j}$，一个学期可以学多门课，有的课之间有依赖关系，即必须先学 a 再学 b，求最大收益。$n, m \leq 100$
- 每门课的学习时间是离散变量，套用上面的模型

RIN

- 有 m 门课可以在 n 个学期内学习，第 i 门课在第 j 个学期学习的收益是 $X_{i,j}$，一个学期可以学多门课，有的课之间有依赖关系，即必须先学 a 再学 b，求最大收益。$n, m \leq 100$
- 每门课的学习时间是离散变量，套用上面的模型
- 将 $X_{i,j}$ 建成一条链，割在 $X_{i,j-1} \to X_{i,j}$ 表示这门课在第 j 个学期学

RIN

- 有 m 门课可以在 n 个学期内学习，第 i 门课在第 j 个学期学习的收益是 $X_{i,j}$，一个学期可以学多门课，有的课之间有依赖关系，即必须先学 a 再学 b，求最大收益。$n, m \leq 100$
- 每门课的学习时间是离散变量，套用上面的模型
- 将 $X_{i,j}$ 建成一条链，割在 $X_{i,j-1} \to X_{i,j}$ 表示这门课在第 j 个学期学
- 求 $X_{i,j}$ 的最大值，然后减去最小割，即可得到最大收益

RIN

- 有 m 门课可以在 n 个学期内学习，第 i 门课在第 j 个学期学习的收益是 $X_{i,j}$，一个学期可以学多门课，有的课之间有依赖关系，即必须先学 a 再学 b，求最大收益。$n, m \leq 100$
- 每门课的学习时间是离散变量，套用上面的模型
- 将 $X_{i,j}$ 建成一条链，割在 $X_{i,j-1} \to X_{i,j}$ 表示这门课在第 j 个学期学
- 求 $X_{i,j}$ 的最大值，然后减去最小割，即可得到最大收益
- 对于限制先学 a 再学 b，对于所有 j 连边 $X_{a,j} \to X_{b,j+1}$，容量为无穷大

RIN

- 有 m 门课可以在 n 个学期内学习，第 i 门课在第 j 个学期学习的收益是 $X_{i,j}$，一个学期可以学多门课，有的课之间有依赖关系，即必须先学 a 再学 b，求最大收益。$n, m \leq 100$
- 每门课的学习时间是离散变量，套用上面的模型
- 将 $X_{i,j}$ 建成一条链，割在 $X_{i,j-1} \to X_{i,j}$ 表示这门课在第 j 个学期学
- 求 $X_{i,j}$ 的最大值，然后减去最小割，即可得到最大收益
- 对于限制先学 a 再学 b，对于所有 j 连边 $X_{a,j} \to X_{b,j+1}$，容量为无穷大
- 这条边永远不会被割，所以它能用来限制方案，即如果 a 比 b 后学，则一定存在一条 $s \to X_{a,i} \to X_{b,j} \to t$ 的路径，因此割的方案还会被调整直至合法

费用流

费用流

- 与网络流模型一样，费用流最简单的题型就是一条流代表一个方案

费用流

- 与网络流模型一样，费用流最简单的题型就是一条流代表一个方案
- 最小费用流还有一个常用的性质是，每次增广产生的费用函数是不降的

Stone

Stone

- 在无向图 G 中，要从源点 s 购买一些石头并运到汇点 t。每块石头单价是 P 元。每条边 i 有一个初始容量 C_i，当容量超过 C_i 时，每增加一单位容量要额外花费 E_i 元。现在你手头只有 C 元，问最多能买多少块石头并成功运到目的地。$n \leq 1000$，$m, P, C_i, E_i \leq 10^4$，$C \leq 10^8$

Stone

- 在无向图 G 中，要从源点 s 购买一些石头并运到汇点 t。每块石头单价是 P 元。每条边 i 有一个初始容量 C_i，当容量超过 C_i 时，每增加一单位容量要额外花费 E_i 元。现在你手头只有 C 元，问最多能买多少块石头并成功运到目的地。$n \leq 1000$，$m, P, C_i, E_i \leq 10^4$，$C \leq 10^8$
- 容易发现费用随流的增加呈分段线性状态

Stone

- 在无向图 G 中，要从源点 s 购买一些石头并运到汇点 t。每块石头单价是 P 元。每条边 i 有一个初始容量 C_i，当容量超过 C_i 时，每增加一单位容量要额外花费 E_i 元。现在你手头只有 C 元，问最多能买多少块石头并成功运到目的地。$n \leq 1000$，$m, P, C_i, E_i \leq 10^4$，$C \leq 10^8$
- 容易发现费用随流的增加呈分段线性状态
- 加边转化为费用流：将原图中每条边 (u, v) 拆成两条：$(u, v, C_i, 0)$ 与 (u, v, ∞, E_i)

Stone

- 在无向图 G 中，要从源点 s 购买一些石头并运到汇点 t。每块石头单价是 P 元。每条边 i 有一个初始容量 C_i，当容量超过 C_i 时，每增加一单位容量要额外花费 E_i 元。现在你手头只有 C 元，问最多能买多少块石头并成功运到目的地。$n \leq 1000$，$m, P, C_i, E_i \leq 10^4$，$C \leq 10^8$
- 容易发现费用随流的增加呈分段线性状态
- 加边转化为费用流：将原图中每条边 (u, v) 拆成两条：$(u, v, C_i, 0)$ 与 (u, v, ∞, E_i)
- 每次找最小费用路并增广，直到钱不够为止

星际竞速

星际竞速

- n 个星球用 m 条带权 (P) 边连接，每条边只能从编号小的到编号大的，现在要求经过所有点恰好 1 次，你有瞬移能力，从任意点瞬移到 i 号点代价为 A_i，求最小代价。$n \leq 800, m \leq 15000$

星际竞速

- n 个星球用 m 条带权 (P) 边连接，每条边只能从编号小的到编号大的，现在要求经过所有点恰好 1 次，你有瞬移能力，从任意点瞬移到 i 号点代价为 A_i，求最小代价。$n \leq 800, m \leq 15000$
- 用流量保证每个点只访问一次 (拆点)，用费用保证代价最小

星际竞速

- n 个星球用 m 条带权 (P) 边连接，每条边只能从编号小的到编号大的，现在要求经过所有点恰好 1 次，你有瞬移能力，从任意点瞬移到 i 号点代价为 A_i，求最小代价。$n \leq 800, m \leq 15000$
- 用流量保证每个点只访问一次 (拆点)，用费用保证代价最小
- 将每次瞬移后的行动看做一条路径，考虑到每条路径除第一个点外都有一个前驱，利用这个条件来建图

星际竞速

- n 个星球用 m 条带权 (P) 边连接，每条边只能从编号小的到编号大的，现在要求经过所有点恰好 1 次，你有瞬移能力，从任意点瞬移到 i 号点代价为 A_i，求最小代价。$n \leq 800, m \leq 15000$
- 用流量保证每个点只访问一次 (拆点)，用费用保证代价最小
- 将每次瞬移后的行动看做一条路径，考虑到每条路径除第一个点外都有一个前驱，利用这个条件来建图
- 源点 s 向所有 u 连边，容量为 1 费用为 0

星际竞速

- n 个星球用 m 条带权 (P) 边连接，每条边只能从编号小的到编号大的，现在要求经过所有点恰好 1 次，你有瞬移能力，从任意点瞬移到 i 号点代价为 A_i，求最小代价。$n \leq 800, m \leq 15000$
- 用流量保证每个点只访问一次 (拆点)，用费用保证代价最小
- 将每次瞬移后的行动看做一条路径，考虑到每条路径除第一个点外都有一个前驱，利用这个条件来建图
- 源点 s 向所有 u 连边，容量为 1 费用为 0
- 所有 u' 向汇点 t 连边，容量为 1 费用为 0

星际竞速

- n 个星球用 m 条带权 (P) 边连接，每条边只能从编号小的到编号大的，现在要求经过所有点恰好 1 次，你有瞬移能力，从任意点瞬移到 i 号点代价为 A_i，求最小代价。$n \leq 800, m \leq 15000$
- 用流量保证每个点只访问一次 (拆点)，用费用保证代价最小
- 将每次瞬移后的行动看做一条路径，考虑到每条路径除第一个点外都有一个前驱，利用这个条件来建图
- 源点 s 向所有 u 连边，容量为 1 费用为 0
- 所有 u' 向汇点 t 连边，容量为 1 费用为 0
- 若原图中 u, v 有边，那么连 $(u, v', 1, P_{u,v})$

星际竞速

- n 个星球用 m 条带权 (P) 边连接，每条边只能从编号小的到编号大的，现在要求经过所有点恰好 1 次，你有瞬移能力，从任意点瞬移到 i 号点代价为 A_i，求最小代价。$n \leq 800, m \leq 15000$
- 用流量保证每个点只访问一次 (拆点)，用费用保证代价最小
- 将每次瞬移后的行动看做一条路径，考虑到每条路径除第一个点外都有一个前驱，利用这个条件来建图
- 源点 s 向所有 u 连边，容量为 1 费用为 0
- 所有 u' 向汇点 t 连边，容量为 1 费用为 0
- 若原图中 u, v 有边，那么连 $(u, v', 1, P_{u,v})$
- 源点 s 向所有 v' 连边，费用为瞬移到他的代价

Part III

题目选讲

Greg and graph

Greg and graph

Description

给定一个 n 个点 m 条边的无向图，再给定一个 $1 \sim n$ 的排列 P，现在要按排列中的次序一个个删掉图中的点，并求出每次删除后图中所有点之间最短路的和，不连通最短路算作 0

Constraints

$n \leq 500$

Source

Codeforces 295B

solution

solution

- 删点不利于统计最短路，倒过来思考，从空图中不断加点

solution

- 删点不利于统计最短路，倒过来思考，从空图中不断加点
- 加点之后可以类似 floyd 进行处理，为了方便我们假设加点顺序是 $1 \sim n$

solution

- 删点不利于统计最短路，倒过来思考，从空图中不断加点
- 加点之后可以类似 floyd 进行处理，为了方便我们假设加点顺序是 $1 \sim n$
- 首先计算 $i+1$ 到 $1 \sim i$ 的最短路：枚举 x, y，
  $dis[i+1][y] = \min(dis[i+1][y], dis[i+1][x] + dis[x][y])$

solution

- 删点不利于统计最短路，倒过来思考，从空图中不断加点
- 加点之后可以类似 floyd 进行处理，为了方便我们假设加点顺序是 $1 \sim n$
- 首先计算 $i+1$ 到 $1 \sim i$ 的最短路：枚举 x, y，
  $dis[i+1][y] = \min(dis[i+1][y], dis[i+1][x] + dis[x][y])$
- 再利用 $i+1$ 更新其他点对：枚举 x, y，
  $dis[x][y] = \min(dis[x][y], dis[x][i+1] + dis[i+1][y])$

Steam Rollar

Steam Rollar

Description

给定一个 $n \times m$ 的网格图，图中每条边都有长度，求从 $(1,1)$ 到 (n,m) 的最短时间。正常情况下速度为 1，但是在出发后的第一条边、到达前的最后一条边以及转弯前后的边速度都会减半。

Constraints

$n, m \leq 100$

Source

Worldfinal2008 K

solution

solution

- 如果没有速度减半的限制，那么简单的最短路即可

solution

- 如果没有速度减半的限制，那么简单的最短路即可
- 加上速度减半条件后，原来的 $dp[i][j]$ 无法满足后效性，所以在状态中多加一些信息

solution

- 如果没有速度减半的限制，那么简单的最短路即可
- 加上速度减半条件后，原来的 $dp[i][j]$ 无法满足后效性，所以在状态中多加一些信息
- $dis[i][j][dir][bo]$ 表示走到 (i,j) 点，上一步方向是 dir，上一条边是否已经速度减半情况下的最短用时

solution

- 如果没有速度减半的限制，那么简单的最短路即可
- 加上速度减半条件后，原来的 $dp[i][j]$ 无法满足后效性，所以在状态中多加一些信息
- $dis[i][j][dir][bo]$ 表示走到 (i,j) 点，上一步方向是 dir，上一条边是否已经速度减半情况下的最短用时
- 每一个状态当做点，转移当做边跑最短路

Meeting Her

Meeting Her

Description

给定一个 n 个点 m 条边的有向图，现在要从 a 点去到 b 点，一共有 k 条公交路线，分别从 s_i 开往 t_i. 每个时刻都会有一辆公交从 s_i 出发并随机选择一条到 t_i 的最短路开过。若某个时刻有公交经过你当前所在节点则你可以上车，并在之后任意点下车。请问最坏情况下你最少要上几辆车。

Constraints

$n, k \leq 100$, $m \leq 10000$

Source

Codeforces 238E

solution

solution

- 首先最坏的情况的意思是，除非公交必须经过你当前所在节点，否则你一定无法在此点上这辆公交

solution

- 首先最坏的情况的意思是，除非公交必须经过你当前所在节点，否则你一定无法在此点上这辆公交
- 因此所在节点是某条公交的必经点时才能坐上

solution

- 首先最坏的情况的意思是，除非公交必须经过你当前所在节点，否则你一定无法在此点上这辆公交
- 因此所在节点是某条公交的必经点时才能坐上
- 必经点对应着以 s_i 为源的最短路径图中的割点

solution

- 首先最坏的情况的意思是，除非公交必须经过你当前所在节点，否则你一定无法在此点上这辆公交
- 因此所在节点是某条公交的必经点时才能坐上
- 必经点对应着以 s_i 为源的最短路径图中的割点
- 令 dp_i 表示到点 i 至少要坐几辆公交，再利用最短路求解 (相当于重新建图)

solution

- 首先最坏的情况的意思是，除非公交必须经过你当前所在节点，否则你一定无法在此点上这辆公交
- 因此所在节点是某条公交的必经点时才能坐上
- 必经点对应着以 s_i 为源的最短路径图中的割点
- 令 dp_i 表示到点 i 至少要坐几辆公交，再利用最短路求解 (相当于重新建图)
- 时间复杂度 $O(nmk)$

Party

Party

Description

给定一个 $3n$ 个点的图，图中存在一个点数不小于 $2n$ 的团，请你求出一个点数为 n 的团

Constraints

$n \leq 1000$

Source

POI

solution

solution

- 考虑补图，在补图中找一个 n 个点的独立集

solution

- 考虑补图，在补图中找一个 n 个点的独立集
- 补图中存在一个 $2n$ 个点的独立集，将这个独立集记为 A，其他点的集合记为 B

solution

- 考虑补图，在补图中找一个 n 个点的独立集
- 补图中存在一个 $2n$ 个点的独立集，将这个独立集记为 A，其他点的集合记为 B
- 考虑不断删点，直到剩下的点都属于 A，且要保证数量足够

solution

- 考虑补图，在补图中找一个 n 个点的独立集
- 补图中存在一个 $2n$ 个点的独立集，将这个独立集记为 A，其他点的集合记为 B
- 考虑不断删点，直到剩下的点都属于 A，且要保证数量足够
- 显然补图中的边必然包含至少 1 个 B 点，于是不断找到补图中的一条边并将这条边对应的两个端点都删去

solution

- 考虑补图，在补图中找一个 n 个点的独立集
- 补图中存在一个 $2n$ 个点的独立集，将这个独立集记为 A，其他点的集合记为 B
- 考虑不断删点，直到剩下的点都属于 A，且要保证数量足够
- 显然补图中的边必然包含至少 1 个 B 点，于是不断找到补图中的一条边并将这条边对应的两个端点都删去
- 最多删 n 条边就能把 B 点全部删完，且此时 A 点至少还剩 n 个，剩余 A 点就是一个符合条件的答案

Infiltration

Infiltration

Description

给一个点数为 n 的竞赛图，求图的最小支配集。(集合内的点的出边覆盖了所有集合外的点)

Constraints

$n \leq 75$

Source

Worldfinal 2012E

solution

solution

- 最小支配集是 NP 问题，所以做法需要从特殊条件入手

solution

- 最小支配集是 NP 问题，所以做法需要从特殊条件入手
- 注意到将竞赛图中的某个结点删去，则图仍是竞赛图

solution

- 最小支配集是 NP 问题，所以做法需要从特殊条件入手
- 注意到将竞赛图中的某个结点删去，则图仍是竞赛图
- 并且竞赛图中所有点出度的最大值一定不小于 $\frac{n-1}{2}$

solution

- 最小支配集是 NP 问题，所以做法需要从特殊条件入手
- 注意到将竞赛图中的某个结点删去，则图仍是竞赛图
- 并且竞赛图中所有点出度的最大值一定不小于 $\frac{n-1}{2}$
- 所以将竞赛图中出度最大的点找出，将它以及它连向的点都删去，原图变为一个只有一半大小的竞赛图

solution

- 最小支配集是 NP 问题，所以做法需要从特殊条件入手
- 注意到将竞赛图中的某个结点删去，则图仍是竞赛图
- 并且竞赛图中所有点出度的最大值一定不小于 $\frac{n-1}{2}$
- 所以将竞赛图中出度最大的点找出，将它以及它连向的点都删去，原图变为一个只有一半大小的竞赛图
- 因此易知竞赛图中最小支配集大小是 $O(\log n)$ 级别的

solution

- 最小支配集是 NP 问题，所以做法需要从特殊条件入手
- 注意到将竞赛图中的某个结点删去，则图仍是竞赛图
- 并且竞赛图中所有点出度的最大值一定不小于 $\frac{n-1}{2}$
- 所以将竞赛图中出度最大的点找出，将它以及它连向的点都删去，原图变为一个只有一半大小的竞赛图
- 因此易知竞赛图中最小支配集大小是 $O(\log n)$ 级别的
- 本题答案至多为 6，爆搜剪剪枝即可

Inverse the Problem

Inverse the Problem

Description

有一个 n 个点的树，其中每条边有一个正整数边权，你并不知道这棵树的样子，只知道每两点间最短路长度，但是这些最短路长度可能是错误的，请你判断是否存在一棵树满足条件

Constraints

$n \leq 2000$

Source

Codeforces 472D

solution

solution

- 非常经典的 MST 题目

solution

- 非常经典的 MST 题目
- 把最短路当做边，求出该图的最小生成树，再到这棵树上判断即可

solution

- 非常经典的 MST 题目
- 把最短路当做边，求出该图的最小生成树，再到这棵树上判断即可
- 注意到边权都是正数，所以原树中任意两点它们之间若无边直接相连，那么它们在树中路径长度一定比路径中每条边都长，做 Kruskal 时不可能被加入

solution

- 非常经典的 MST 题目
- 把最短路当做边，求出该图的最小生成树，再到这棵树上判断即可
- 注意到边权都是正数，所以原树中任意两点它们之间若无边直接相连，那么它们在树中路径长度一定比路径中每条边都长，做 Kruskal 时不可能被加入
- 换句话说，求出的 MST 只可能是原树中的边

Kuglarz

Kuglarz

Description

魔术师的桌子上有 n 个杯子排成一行，其中某些杯子底下藏有一个小球，花费 $C_{i,j}$ 你可以知道 $i \sim j$ 底下藏有球的总数的奇偶性。求至少花费多少元才能保证猜出哪些杯子底下藏着球

Constraints

$n \leq 2000$, $1 \leq C_{i,j} \leq 10^9$

Source

PA2014

solution

solution

- 考虑 sum_i 表示 $1 \sim i$ 中球的数量

solution

- 考虑 sum_i 表示 $1 \sim i$ 中球的数量
- 知道了所有 sum_i 的奇偶性就能得出球的情况

solution

- 考虑 sum_i 表示 $1 \sim i$ 中球的数量
- 知道了所有 sum_i 的奇偶性就能得出球的情况
- 若已知 sum_{i-1} 的奇偶性，用 $C_{i,j}$ 可以得出 sum_j 的奇偶性

solution

- 考虑 sum_i 表示 $1 \sim i$ 中球的数量
- 知道了所有 sum_i 的奇偶性就能得出球的情况
- 若已知 sum_{i-1} 的奇偶性，用 $C_{i,j}$ 可以得出 sum_j 的奇偶性
- 将所有的 sum_i 当做点，将 $C_{i,j}$ 当做边，现在已知 sum_0，对这个图做 MST 将所有点连起来就相当于知道了所有 sum_i

solution

- 考虑 sum_i 表示 $1 \sim i$ 中球的数量
- 知道了所有 sum_i 的奇偶性就能得出球的情况
- 若已知 sum_{i-1} 的奇偶性，用 $C_{i,j}$ 可以得出 sum_j 的奇偶性
- 将所有的 sum_i 当做点，将 $C_{i,j}$ 当做边，现在已知 sum_0，对这个图做 MST 将所有点连起来就相当于知道了所有 sum_i
- 用 Prim 求解，$O(n^2)$

ZigZag MST

ZigZag MST

Description

对一张 n 个点的空图 G 做 Q 次加边操作，每次给定 A_i, B_i, C_i，然后按照以下规则按顺序连 10^{233} 条边：$(A_i, B_i, C_i), (B_i, A_i + 1, C_i + 1), (A_i + 1, B_i + 1, C_i + 2)$，以此类推，点的编号均为 $\mathrm{mod}\ n$ 意义下的，求图 G 的最小生成树。

Constraints

$n, Q \leq 2 \times 10^5$, $1 \leq C \leq 10^9$

Source

Atcoder Code Festival 2016 Final Problem G

solution

solution

- 考虑画个图，然后按照 Kruskal 的过程转化一下边

solution

- 考虑画个图，然后按照 Kruskal 的过程转化一下边
- $(A, A+i)$ 连一条 $(C+1+2i)$ 的边

solution

- 考虑画个图，然后按照 Kruskal 的过程转化一下边
- $(A, A+i)$ 连一条 $(C+1+2i)$ 的边
- $(B, B+i)$ 连一条 $(C+2+2i)$ 的边

solution

- 考虑画个图，然后按照 Kruskal 的过程转化一下边
- $(A, A+i)$ 连一条 $(C+1+2i)$ 的边
- $(B, B+i)$ 连一条 $(C+2+2i)$ 的边
- (A, B) 连一条 C 的边

solution

- 考虑画个图，然后按照 Kruskal 的过程转化一下边
- $(A, A+i)$ 连一条 $(C+1+2i)$ 的边
- $(B, B+i)$ 连一条 $(C+2+2i)$ 的边
- (A, B) 连一条 C 的边
- 前两种连了整张图一圈，考虑以第一种边为例

solution

- 考虑画个图，然后按照 Kruskal 的过程转化一下边
- $(A, A+i)$ 连一条 $(C+1+2i)$ 的边
- $(B, B+i)$ 连一条 $(C+2+2i)$ 的边
- (A, B) 连一条 C 的边
- 前两种连了整张图一圈，考虑以第一种边为例
- d_i 表示 $(i, i+1)$ 间的最小权值

solution

- 考虑画个图，然后按照 Kruskal 的过程转化一下边
- $(A, A+i)$ 连一条 $(C+1+2i)$ 的边
- $(B, B+i)$ 连一条 $(C+2+2i)$ 的边
- (A, B) 连一条 C 的边
- 前两种连了整张图一圈，考虑以第一种边为例
- d_i 表示 $(i, i+1)$ 间的最小权值
- $d_i = \min(d_i, C)$，随后循环一遍 $d_i = \min(d_i, d_{i-1} + 2)$

solution

- 考虑画个图，然后按照 Kruskal 的过程转化一下边
- $(A, A+i)$ 连一条 $(C+1+2i)$ 的边
- $(B, B+i)$ 连一条 $(C+2+2i)$ 的边
- (A, B) 连一条 C 的边
- 前两种连了整张图一圈，考虑以第一种边为例
- d_i 表示 $(i, i+1)$ 间的最小权值
- $d_i = \min(d_i, C)$，随后循环一遍 $d_i = \min(d_i, d_{i-1} + 2)$
- 总边数 $n+Q$，Kruskal 即可

Fairy

Fairy

Description

给一个点数为 n 边数为 m 的图，请你判断图中哪些边删掉后这个图是个二分图

Constraints

$n, m \leq 10^4$

Source

Codeforces 19E

solution

solution

- 若原图是二分图，则所有边都满足条件

solution

- 若原图是二分图，则所有边都满足条件
- 否则搜索树中有至少一条违规边，先考虑哪些非树边可删

solution

- 若原图是二分图，则所有边都满足条件
- 否则搜索树中有至少一条违规边，先考虑哪些非树边可删
- 若有多条违规边，则所有非树边都不可删；若只有一条则删掉它即可

solution

- 若原图是二分图，则所有边都满足条件
- 否则搜索树中有至少一条违规边，先考虑哪些非树边可删
- 若有多条违规边，则所有非树边都不可删；若只有一条则删掉它即可
- 再考虑树边，对任意一条违规边，它的两个端点在树中的路径中必有一条边需被删，因此将所有违规边的路径求交集

solution

- 若原图是二分图，则所有边都满足条件
- 否则搜索树中有至少一条违规边，先考虑哪些非树边可删
- 若有多条违规边，则所有非树边都不可删；若只有一条则删掉它即可
- 再考虑树边，对任意一条违规边，它的两个端点在树中的路径中必有一条边需被删，因此将所有违规边的路径求交集
- 若删除边同属于一个奇环和偶环，那么删除后还存在奇环，所以删除边不能属于偶环

solution

- 若原图是二分图，则所有边都满足条件
- 否则搜索树中有至少一条违规边，先考虑哪些非树边可删
- 若有多条违规边，则所有非树边都不可删；若只有一条则删掉它即可
- 再考虑树边，对任意一条违规边，它的两个端点在树中的路径中必有一条边需被删，因此将所有违规边的路径求交集
- 若删除边同属于一个奇环和偶环，那么删除后还存在奇环，所以删除边不能属于偶环
- 综上，奇环交集与偶环并集的差集就是可删除的树边集合，利用差分方法可求解

Mining Your Own Business

Mining Your Own Business

Description

有一个矿井，包含 n 个工作点，工作点之间由 m 条双向通道连接起来。现在这个矿井要考虑在一些工作点修井口，以便紧急情况下逃生。负责人希望在某个工作点塌陷而不能通行之后，其他所有工作点的人都能够到达某个井口。请你求出最少要建几个井口，以及有多少种最优方案。

Constraints

$n, m \leq 10^5$

Source

ACM WorldFinal 2011H

solution

solution

- 首先我们将图中的点双连通分量求出来，显然在割点建井口不优

solution

- 首先我们将图中的点双连通分量求出来，显然在割点建井口不优
- 若这个点双中恰好只有一个割点，那么这个点双必须选一个非割点作为井口，否则割点塌陷就不满足条件

solution

- 首先我们将图中的点双连通分量求出来，显然在割点建井口不优
- 若这个点双中恰好只有一个割点，那么这个点双必须选一个非割点作为井口，否则割点塌陷就不满足条件
- 若这个点双有多个割点，则由于将点双看成点后，图将会成为类似树的结构，因此无论哪个割点塌陷，通过剩下的割点一定能找到一个井口，所以该点双不需要建井口

solution

- 首先我们将图中的点双连通分量求出来，显然在割点建井口不优
- 若这个点双中恰好只有一个割点，那么这个点双必须选一个非割点作为井口，否则割点塌陷就不满足条件
- 若这个点双有多个割点，则由于将点双看成点后，图将会成为类似树的结构，因此无论哪个割点塌陷，通过剩下的割点一定能找到一个井口，所以该点双不需要建井口
- 任意两个点双的非割点不会相交，方案数可以使用乘法原理计算

solution

- 首先我们将图中的点双连通分量求出来，显然在割点建井口不优
- 若这个点双中恰好只有一个割点，那么这个点双必须选一个非割点作为井口，否则割点塌陷就不满足条件
- 若这个点双有多个割点，则由于将点双看成点后，图将会成为类似树的结构，因此无论哪个割点塌陷，通过剩下的割点一定能找到一个井口，所以该点双不需要建井口
- 任意两个点双的非割点不会相交，方案数可以使用乘法原理计算
- 特殊情况：整张图是个点双，那么需要任选两个点作为井口

The Ministers' Major Mess

The Ministers' Major Mess

Description

在一个国度中，有 n 位大臣正在商讨本国的 m 个议案是否要被通过。每个大臣有最多 4 个意见，每个意见都是诸如 x 号议案必须通过 (或放弃) 的形式。问有没有一种议案的通过方式，使得每位大臣都至少有超过一半的意见得到满足。

Constraints

$n \leq 100$, $m \leq 500$

Source

ACM WorldFinal 2009H

solution

solution

- 每位大臣最多只有一个意见不能被满足

solution

- 每位大臣最多只有一个意见不能被满足
- 将每个议案是否通过看成一个布尔变量，对于意见超过两个的大臣，将"它的某个意见不能被满足"对应的赋值方法两两之间建立不共存约束条件

solution

- 每位大臣最多只有一个意见不能被满足
- 将每个议案是否通过看成一个布尔变量，对于意见超过两个的大臣，将"它的某个意见不能被满足"对应的赋值方法两两之间建立不共存约束条件
- 转化成为了 2SAT 问题，求是否有解即可

solution

- 每位大臣最多只有一个意见不能被满足
- 将每个议案是否通过看成一个布尔变量，对于意见超过两个的大臣，将"它的某个意见不能被满足"对应的赋值方法两两之间建立不共存约束条件
- 转化成为了 2SAT 问题，求是否有解即可
- 小 Tips：使用 Tarjan 求强连通分量解 2SAT 时，并不需要将图反向求解方案，直接使用所在连通分量的编号作为选择顺序即可

Riddle

Riddle

Description

给定一张 n 个点 m 条边的图 G，n 个点被分成了 K 个集合，每个点恰好属于一个集合。现在要在每个集合中选恰好一个点，使得对于每条边，两个端点中至少有一个被选择。

Constraints

$n, m, k \leq 10^6$

Source

PA2010

solution

solution

- 对于一条边 (u, v)，以及同一个集合中的两个点 (u, v)，都存在二元关系

solution

- 对于一条边 (u, v)，以及同一个集合中的两个点 (u, v)，都存在二元关系
- 2SAT 模型，然而直接做的话关系数有 $O(n^2)$ 个

solution

- 对于一条边 (u, v)，以及同一个集合中的两个点 (u, v)，都存在二元关系
- 2SAT 模型，然而直接做的话关系数有 $O(n^2)$ 个
- 考虑优化集合中的二元关系

solution

- 对于一条边 (u, v)，以及同一个集合中的两个点 (u, v)，都存在二元关系
- 2SAT 模型，然而直接做的话关系数有 $O(n^2)$ 个
- 考虑优化集合中的二元关系
- 对一个 x 个点的集合新建 $2x$ 个点，分别表示前 i 以及后 i 个点中是否有点被选

solution

- 对于一条边 (u, v)，以及同一个集合中的两个点 (u, v)，都存在二元关系
- 2SAT 模型，然而直接做的话关系数有 $O(n^2)$ 个
- 考虑优化集合中的二元关系
- 对一个 x 个点的集合新建 $2x$ 个点，分别表示前 i 以及后 i 个点中是否有点被选
- 边数降为 $O(n + m)$，解 2SAT 即可

solution

- 对于一条边 (u, v)，以及同一个集合中的两个点 (u, v)，都存在二元关系
- 2SAT 模型，然而直接做的话关系数有 $O(n^2)$ 个
- 考虑优化集合中的二元关系
- 对一个 x 个点的集合新建 $2x$ 个点，分别表示前 i 以及后 i 个点中是否有点被选
- 边数降为 $O(n + m)$，解 2SAT 即可
- 小 Tips：使用 Tarjan 求强连通分量解 2SAT 时，并不需要将图反向求解方案，直接使用所在连通分量的编号作为选择顺序即可

TheTilesDivOne

TheTilesDivOne

Description

给定一个 $n \times m$ 的网格，对这个网格黑白染色，左上角为黑色。现在要用一些大小为 3 的 L 型图形覆盖这个网格，要求不能重复覆盖同一个格子，不能覆盖到障碍，L 型可以进行旋转，但转角处格子必须为黑色，求最多能覆盖多少个 L 型图形

Constraints

$1 \leq n, m \leq 47$

Source

Topcoder SRM 575 Div1 Hard

solution

solution

- 直接黑白染色，黑格连出 2 的容量，这样的图的流不能保证图形为 L

solution

- 直接黑白染色，黑格连出 2 的容量，这样的图的流不能保证图形为 L
- 我们发现 L 型不论怎么旋转，两个白格必定不在同一行

solution

- 直接黑白染色，黑格连出 2 的容量，这样的图的流不能保证图形为 L
- 我们发现 L 型不论怎么旋转，两个白格必定不在同一行
- 考虑三染色，原黑格染 0，奇数行原白格染 1，偶数行原白格染 2

solution

- 直接黑白染色，黑格连出 2 的容量，这样的图的流不能保证图形为 L
- 我们发现 L 型不论怎么旋转，两个白格必定不在同一行
- 考虑三染色，原黑格染 0，奇数行原白格染 1，偶数行原白格染 2
- S 向颜色 1 连边，颜色 1 向颜色 0 连边，颜色 0 向颜色 2 连边，这样流出来的肯定是个 L

solution

- 直接黑白染色，黑格连出 2 的容量，这样的图的流不能保证图形为 L
- 我们发现 L 型不论怎么旋转，两个白格必定不在同一行
- 考虑三染色，原黑格染 0，奇数行原白格染 1，偶数行原白格染 2
- S 向颜色 1 连边，颜色 1 向颜色 0 连边，颜色 0 向颜色 2 连边，这样流出来的肯定是个 L
- 为了保证每个格子只用 1 次，拆点限制流量即可

CurvyonRails

CurvyonRails

Description

给定一个 $N \times M$ 的网格，每个格子是城镇或者荒地。现在要在城镇间建设铁路，每个城镇需要建设恰好一条铁路，它连接着一个格子相邻四个方向上的某两条边，且这条铁路需要与其他城镇的铁路相接。荒地上不能建设铁路。(注意所有铁路不一定要全部连通，它们可以是几个不相交的环)。有些城镇是特殊点，现在要求一个铁路建造方案，使得满足铁路建设条件的情况下，特殊点上铁路是" 直" 的数量最少。

Constraints

$1 \leq N, M \leq 25$

Source

Topcoder SRM 570 Div1 Hard

solution

solution

- 若没有特殊点，那么黑白染色后，源向黑点连容量 2 的边，白点向汇连容量 2 的边，相邻点连容量为 1 的边，跑最大流即可

solution

- 若没有特殊点，那么黑白染色后，源向黑点连容量 2 的边，白点向汇连容量 2 的边，相邻点连容量为 1 的边，跑最大流即可
- 现在需要路径不为直，那么怎么来区别直路径和弯曲路径？

solution

- 若没有特殊点，那么黑白染色后，源向黑点连容量 2 的边，白点向汇连容量 2 的边，相邻点连容量为 1 的边，跑最大流即可
- 现在需要路径不为直，那么怎么来区别直路径和弯曲路径？
- 我们考虑拆点，将点拆为水平方向与垂直方向两个点。相邻格子间对应方向的点连双向边

solution

- 若没有特殊点，那么黑白染色后，源向黑点连容量 2 的边，白点向汇连容量 2 的边，相邻点连容量为 1 的边，跑最大流即可
- 现在需要路径不为直，那么怎么来区别直路径和弯曲路径？
- 我们考虑拆点，将点拆为水平方向与垂直方向两个点。相邻格子间对应方向的点连双向边
- 现在我们能保证这些点都为"L" 型，但是这张图不一定能保证有解，我们需要添加直边

solution

- 若没有特殊点，那么黑白染色后，源向黑点连容量 2 的边，白点向汇连容量 2 的边，相邻点连容量为 1 的边，跑最大流即可
- 现在需要路径不为直，那么怎么来区别直路径和弯曲路径？
- 我们考虑拆点，将点拆为水平方向与垂直方向两个点。相邻格子间对应方向的点连双向边
- 现在我们能保证这些点都为"L" 型，但是这张图不一定能保证有解，我们需要添加直边
- 对于每个格子拆出的两个点，我们在他们中间连一条双向边，若是特殊点则费用为 1，否则为 0，对这张图跑最小费用最大流

Divide

Divide

Description

给定一张 n 个点 m 条边的无向连通图，初始时 1 号点属于集合 A，n 号点属于集合 B. 现在要将其他点划分进两个集合，并使得评分最高，评分方式如下:

1. 对于点 i，划给 A 集合得 VA_i 分，划给 B 集合得 VB_i 分
2. 对于一条边 i，若它连接两个 A 集合点，则得 EA_i 分，若它连接两个 B 集合点，则得 EB_i 分，否则将扣除 EC_i 分

Constraints

$n, Q \leq 2 \times 10^5$, $1 \leq C \leq 10^9$

Source

BZOJ 3511

solution

solution

- 经典的二者取其一式的最小割模型题目。假设 S 代表 A 集合，也就是与 S 连通的是 A 集合点

solution

- 经典的二者取其一式的最小割模型题目。假设 S 代表 A 集合，也就是与 S 连通的是 A 集合点
- 对于每个点 i, $(S \to i)$ 新建容量为 VA_i 的边，$(i \to T)$ 新建容量为 VB_i 的边

solution

- 经典的二者取其一式的最小割模型题目。假设 S 代表 A 集合，也就是与 S 连通的是 A 集合点
- 对于每个点 i，$(S \to i)$ 新建容量为 VA_i 的边，$(i \to T)$ 新建容量为 VB_i 的边
- 对于每条边 e，假设它连接的是 i, j 两个点

solution

- 经典的二者取其一式的最小割模型题目。假设 S 代表 A 集合，也就是与 S 连通的是 A 集合点
- 对于每个点 i，$(S \to i)$ 新建容量为 VA_i 的边，$(i \to T)$ 新建容量为 VB_i 的边
- 对于每条边 e，假设它连接的是 i, j 两个点
- $(S \to i)$ 与 $(S \to j)$ 新建容量为 $\frac{EA_i}{2}$ 的边，$(i \to T)$ 与 $(j \to T)$ 新建容量为 $\frac{EB_i}{2}$ 的边

solution

- 经典的二者取其一式的最小割模型题目。假设 S 代表 A 集合，也就是与 S 连通的是 A 集合点
- 对于每个点 i，$(S \to i)$ 新建容量为 VA_i 的边，$(i \to T)$ 新建容量为 VB_i 的边
- 对于每条边 e，假设它连接的是 i, j 两个点
- $(S \to i)$ 与 $(S \to j)$ 新建容量为 $\frac{EA_i}{2}$ 的边，$(i \to T)$ 与 $(j \to T)$ 新建容量为 $\frac{EB_i}{2}$ 的边
- (i, j) 之间新建容量为 $\frac{EA_i}{2} + \frac{EB_i}{2} + EC_i$ 的双向边

solution

- 经典的二者取其一式的最小割模型题目。假设 S 代表 A 集合，也就是与 S 连通的是 A 集合点
- 对于每个点 i，$(S \to i)$ 新建容量为 VA_i 的边，$(i \to T)$ 新建容量为 VB_i 的边
- 对于每条边 e，假设它连接的是 i, j 两个点
- $(S \to i)$ 与 $(S \to j)$ 新建容量为 $\frac{EA_i}{2}$ 的边，$(i \to T)$ 与 $(j \to T)$ 新建容量为 $\frac{EB_i}{2}$ 的边
- (i, j) 之间新建容量为 $\frac{EA_i}{2} + \frac{EB_i}{2} + EC_i$ 的双向边
- $\sum(VA_i + VB_i + EA_i + EB_i) - Mincut$ 即是答案

Juice Junctions

Juice Junctions

Description

给定一张 n 个点 m 条边的无向图，每个点度数最多为 3，求任意两点间最大流。

Constraints

$n \leq 3000$

Source

cerc2015

solution

solution

- 由于度数的限制，答案至多为 3

solution

- 由于度数的限制，答案至多为 3
- 从最小割的角度考虑

solution

- 由于度数的限制，答案至多为 3
- 从最小割的角度考虑
- 两点不连通，答案为 0

solution

- 由于度数的限制，答案至多为 3
- 从最小割的角度考虑
- 两点不连通，答案为 0
- 两点不在一个边双中，答案为 1

solution

- 由于度数的限制，答案至多为 3
- 从最小割的角度考虑
- 两点不连通，答案为 0
- 两点不在一个边双中，答案为 1
- 两点在同一边双中，答案可能为 2 或 3

solution

- 由于度数的限制，答案至多为 3
- 从最小割的角度考虑
- 两点不连通，答案为 0
- 两点不在一个边双中，答案为 1
- 两点在同一边双中，答案可能为 2 或 3
- 若两点间最小割为 3，那么删去图中任意一条边，这两点仍在同一边双中

solution

solution

- 考虑枚举删去图中每一条边，每个点得到删去它后的边双编号

solution

- 考虑枚举删去图中每一条边，每个点得到删去它后的边双编号
- 每个点有 m 个值，判断两点答案是否为 3，只要判断这 m 个值是否对应相等，Hash 或者 Trie 均可

solution

- 考虑枚举删去图中每一条边，每个点得到删去它后的边双编号
- 每个点有 m 个值，判断两点答案是否为 3，只要判断这 m 个值是否对应相等，Hash 或者 Trie 均可
- 时间复杂度 $O(nm)$

solution

- 考虑枚举删去图中每一条边，每个点得到删去它后的边双编号
- 每个点有 m 个值，判断两点答案是否为 3，只要判断这 m 个值是否对应相等，Hash 或者 Trie 均可
- 时间复杂度 $O(nm)$
- 当然这也是一道最小割树模板题，并且它能做没有度数限制的情况

Rabbit Lunch

Rabbit Lunch

Description

有 n 种萝卜，每种萝卜 a_i 个，还有 m 种猕猴桃，每种猕猴桃有 b_i 个。现在有一群兔子要吃东西，每只兔子必须要吃一个萝卜和一个猕猴桃。每只兔子吃的萝卜和猕猴桃不能完全一样，问最多能满足多少只兔子。

Constraints

$n, m \leq 2 \times 10^6$

Source

Opentrain Makoto Soejima Contest 3 Problem A

solution

solution

- 考虑网络流建图：左边 n 个点，源往第 i 个点连流量 a_i 的边，右边 m 个点，第 j 个点往汇连流量为 b_j 的边，中间是个流量全为 1 的二分图，最大流就是答案

solution

- 考虑网络流建图：左边 n 个点，源往第 i 个点连流量 a_i 的边，右边 m 个点，第 j 个点往汇连流量为 b_j 的边，中间是个流量全为 1 的二分图，最大流就是答案
- 最大流等于最小割，考虑最小割的情况

solution

- 考虑网络流建图：左边 n 个点，源往第 i 个点连流量 a_i 的边，右边 m 个点，第 j 个点往汇连流量为 b_j 的边，中间是个流量全为 1 的二分图，最大流就是答案
- 最大流等于最小割，考虑最小割的情况
- 假设左边割了 x 个 a_i 右边割了 y 个 b_i，那么现在的割为

$$cut(x, y) = \sum_{k=1}^{x} a_{i_k} + \sum_{l=1}^{y} b_{j_l} + (n - x) \cdot (m - y)$$

solution

- 考虑网络流建图：左边 n 个点，源往第 i 个点连流量 a_i 的边，右边 m 个点，第 j 个点往汇连流量为 b_j 的边，中间是个流量全为 1 的二分图，最大流就是答案
- 最大流等于最小割，考虑最小割的情况
- 假设左边割了 x 个 a_i 右边割了 y 个 b_i，那么现在的割为

$$cut(x, y) = \sum_{k=1}^{x} a_{i_k} + \sum_{l=1}^{y} b_{j_l} + (n - x) \cdot (m - y)$$

- 为了让割最小，显然割最小的那些 a_i, b_j

solution

solution

- 将 a, b 排序，割就是 x, y 的函数，目标是求上式的最小值

solution

- 将 a, b 排序，割就是 x, y 的函数，目标是求上式的最小值
- 如果固定 x，$cut(x, y+1) > cut(x, y)$ 等价于 $b_{y+1} > n - x$，所以第一个满足这个条件的 y 就是最小取值点

solution

- 将 a, b 排序，割就是 x, y 的函数，目标是求上式的最小值
- 如果固定 x，$cut(x, y+1) > cut(x, y)$ 等价于 $b_{y+1} > n - x$，所以第一个满足这个条件的 y 就是最小取值点
- 从小到大枚举 x，则 $b_{y+1} > n - x$ 的右边严格下降，y 的最优取值点非增，利用双指针即可 $O(n)$ 求解

kro

kro

Description

给定一个 n 个点 m 条边的连通无向图 G。其中 m 是偶数，每条边长度均为 1。假设图中有 k 个奇点 (度数为奇数的点，显然 k 是偶数)，你需要把这 k 个奇点配成 $k/2$ 个点对。对于每个点对 (u, v) 你需要用一条长为偶数的路径连接它们。路径允许经过重复的点但不能经过重复的边，$k/2$ 条路径之间不能经过重复的边。

Constraints

$n, m \leq 2 \times 10^5$

Source

PA2014 Final

solution

solution

- 手画一些小数据，可以猜想一定存在某种方案能使得每条边都被经过恰好一次，容易联想到构造欧拉回路

solution

- 手画一些小数据，可以猜想一定存在某种方案能使得每条边都被经过恰好一次，容易联想到构造欧拉回路
- 新加一个点 X，令它与所有奇点连边，在新图中得出欧拉回路 C

solution

- 手画一些小数据，可以猜想一定存在某种方案能使得每条边都被经过恰好一次，容易联想到构造欧拉回路
- 新加一个点 X，令它与所有奇点连边，在新图中得出欧拉回路 C
- 将 C 根据奇点的位置断开，就可以得到 $k/2$ 条路径，但这样不能保证路径长度均为偶数

solution

- 手画一些小数据，可以猜想一定存在某种方案能使得每条边都被经过恰好一次，容易联想到构造欧拉回路
- 新加一个点 X，令它与所有奇点连边，在新图中得出欧拉回路 C
- 将 C 根据奇点的位置断开，就可以得到 $k/2$ 条路径，但这样不能保证路径长度均为偶数
- 考虑用类似奇偶染色的方法修正上述算法

solution

- 手画一些小数据，可以猜想一定存在某种方案能使得每条边都被经过恰好一次，容易联想到构造欧拉回路
- 新加一个点 X，令它与所有奇点连边，在新图中得出欧拉回路 C
- 将 C 根据奇点的位置断开，就可以得到 $k/2$ 条路径，但这样不能保证路径长度均为偶数
- 考虑用类似奇偶染色的方法修正上述算法
- 建一张新图 G'。将 G 中每个点 u 拆为 u_A, u_B，将 G 中的边 (u, v)，在 G' 连成 (u_A, v_B) 或 (u_B, v_A)(下面再讨论具体连哪种)

solution

- 手画一些小数据，可以猜想一定存在某种方案能使得每条边都被经过恰好一次，容易联想到构造欧拉回路
- 新加一个点 X，令它与所有奇点连边，在新图中得出欧拉回路 C
- 将 C 根据奇点的位置断开，就可以得到 $k/2$ 条路径，但这样不能保证路径长度均为偶数
- 考虑用类似奇偶染色的方法修正上述算法
- 建一张新图 G'。将 G 中每个点 u 拆为 u_A, u_B，将 G 中的边 (u, v)，在 G' 连成 (u_A, v_B) 或 (u_B, v_A)(下面再讨论具体连哪种)
- 仍然新建点 X，对于 G 中的奇点 u，连 (X, u_A)

solution

- 手画一些小数据，可以猜想一定存在某种方案能使得每条边都被经过恰好一次，容易联想到构造欧拉回路
- 新加一个点 X，令它与所有奇点连边，在新图中得出欧拉回路 C
- 将 C 根据奇点的位置断开，就可以得到 $k/2$ 条路径，但这样不能保证路径长度均为偶数
- 考虑用类似奇偶染色的方法修正上述算法
- 建一张新图 G'。将 G 中每个点 u 拆为 u_A, u_B，将 G 中的边 (u, v)，在 G' 连成 (u_A, v_B) 或 (u_B, v_A)(下面再讨论具体连哪种)
- 仍然新建点 X，对于 G 中的奇点 u，连 (X, u_A)
- 若 G' 存在欧拉回路，则现在按上面的方法断开，得到的路径长度一定是偶数，因为我们保证路径上点是 ABAB 交替出现，奇数长度的路径以 B 结尾无法回到 X

solution

solution

- 现在只需要对每条边决定连哪种边

solution

- 现在只需要对每条边决定连哪种边
- 由于 G' 需要存在欧拉回路，所以所有点度数均要为偶数

solution

- 现在只需要对每条边决定连哪种边
- 由于 G' 需要存在欧拉回路，所以所有点度数均要为偶数
- 而对于奇点 u，已经连有 (X, u_A)，所以 u_A 还需要奇数条边，而其他点都还需要偶数条边

solution

- 现在只需要对每条边决定连哪种边
- 由于 G' 需要存在欧拉回路，所以所有点度数均要为偶数
- 而对于奇点 u，已经连有 (X, u_A)，所以 u_A 还需要奇数条边，而其他点都还需要偶数条边
- 考虑随意取 G 的一棵生成树 T，对于非树边随意定方式

solution

- 现在只需要对每条边决定连哪种边
- 由于 G' 需要存在欧拉回路，所以所有点度数均要为偶数
- 而对于奇点 u，已经连有 (X, u_A)，所以 u_A 还需要奇数条边，而其他点都还需要偶数条边
- 考虑随意取 G 的一棵生成树 T，对于非树边随意定方式
- 将 T 看做有根树，并从下往上递推，根据非树边的情况来决定每条树边如何连，并使得对应的点的度数符合条件

solution

- 现在只需要对每条边决定连哪种边
- 由于 G' 需要存在欧拉回路，所以所有点度数均要为偶数
- 而对于奇点 u，已经连有 (X, u_A)，所以 u_A 还需要奇数条边，而其他点都还需要偶数条边
- 考虑随意取 G 的一棵生成树 T，对于非树边随意定方式
- 将 T 看做有根树，并从下往上递推，根据非树边的情况来决定每条树边如何连，并使得对应的点的度数符合条件
- 由于 m 是偶数，容易证明这样构造一定不会矛盾

solution

- 现在只需要对每条边决定连哪种边
- 由于 G' 需要存在欧拉回路，所以所有点度数均要为偶数
- 而对于奇点 u，已经连有 (X, u_A)，所以 u_A 还需要奇数条边，而其他点都还需要偶数条边
- 考虑随意取 G 的一棵生成树 T，对于非树边随意定方式
- 将 T 看做有根树，并从下往上递推，根据非树边的情况来决定每条树边如何连，并使得对应的点的度数符合条件
- 由于 m 是偶数，容易证明这样构造一定不会矛盾
- 时间复杂度 $O(n+m)$

tax

tax

Description

给定一个 n 个点 m 条边的无向图，经过一个点的代价是进入和离开这个点的两条边的边权较大值。求点 1 到点 n 的最小代价。

Constraints

$n, m \leq 2 \times 10^5$

Source

PA2012

solution

solution

- 可以将边看成点，有公共端点的新点之间连边，答案即为新图最短路，但这样建图边数太大

solution

- 可以将边看成点，有公共端点的新点之间连边，答案即为新图最短路，但这样建图边数太大
- 假设经过的边权依次为 $a_1, a_2, \cdots, a_k$

solution

- 可以将边看成点，有公共端点的新点之间连边，答案即为新图最短路，但这样建图边数太大
- 假设经过的边权依次为 $a_1, a_2, \cdots, a_k$
- 首先我们先令代价为进入该点的边权，即 $\sum a_i$

solution

- 可以将边看成点，有公共端点的新点之间连边，答案即为新图最短路，但这样建图边数太大
- 假设经过的边权依次为 $a_1, a_2, \cdots, a_k$
- 首先我们先令代价为进入该点的边权，即 $\sum a_i$
- 现在的代价即为相邻两个数较大值

solution

- 可以将边看成点，有公共端点的新点之间连边，答案即为新图最短路，但这样建图边数太大
- 假设经过的边权依次为 $a_1, a_2, \cdots, a_k$
- 首先我们先令代价为进入该点的边权，即 $\sum a_i$
- 现在的代价即为相邻两个数较大值
- 若 $a_i < a_{i+1}$ 则答案要加上 $a_{i+1} - a_i$，否则不变

solution

- 可以将边看成点，有公共端点的新点之间连边，答案即为新图最短路，但这样建图边数太大
- 假设经过的边权依次为 $a_1, a_2, \cdots, a_k$
- 首先我们先令代价为进入该点的边权，即 $\sum a_i$
- 现在的代价即为相邻两个数较大值
- 若 $a_i < a_{i+1}$ 则答案要加上 $a_{i+1} - a_i$，否则不变
- 考虑将每条边拆成两个新点，它们之间边权为原边权，并且两个点分属于原边的两端点

solution

solution

- 对于同一个点的新点，将它们按原边权排序，边权小的新点向大的新点连长为原边权之差的边，大的点向小的点连长为 0 的边

solution

- 对于同一个点的新点，将它们按原边权排序，边权小的新点向大的新点连长为原边权之差的边，大的点向小的点连长为 0 的边
- 这样如果走的下一条边比之前大，那么会补上与差值相同的代价，而下一条边比之前小，则答案不变

solution

- 对于同一个点的新点，将它们按原边权排序，边权小的新点向大的新点连长为原边权之差的边，大的点向小的点连长为 0 的边
- 这样如果走的下一条边比之前大，那么会补上与差值相同的代价，而下一条边比之前小，则答案不变
- 最后新建源点汇点，并分别与点 1、点 n 连边

solution

- 对于同一个点的新点，将它们按原边权排序，边权小的新点向大的新点连长为原边权之差的边，大的点向小的点连长为 0 的边
- 这样如果走的下一条边比之前大，那么会补上与差值相同的代价，而下一条边比之前小，则答案不变
- 最后新建源点汇点，并分别与点 1、点 n 连边
- 新图中点数为 $2m+2$，边数为 $6m-2n+2$，Dijkstra 求最短路即可，$O(m \log m)$

Rains Over Atlantis

Rains Over Atlantis

Description

给定一个 $n \times m$ 的网格地图，每个格子有海拔高度 a_i。现在下大雨，一个格子若四周都比它高就会积水，直到可以流出去。地图外是大海，海拔为 0 并能接纳无限多的水。令海拔 a_i 加上水深称为这个格子的水平面 h_i。格子里的水会流向四周水平面最低的格子。若某天格子 S 的水流向了相邻的 T，那么 S 的土地会被侵蚀，海拔降低 $\min(h_S - h_T, M)$，M 为定值。所有格子的侵蚀在一天的同一时刻发生，侵蚀后多余的水会流走，求多少天后所有格子海拔全为 0。

Constraints

$n, m \leq 20$, $a_i, M \leq 10^{15}$

Source

Google Code Jam World Final 2011 B

solution

solution

- 考虑暴力模拟，一个格子的水平面为从海面出发到其路径的最高点的最小值，Dijkstra 即可

solution

- 考虑暴力模拟，一个格子的水平面为从海面出发到其路径的最高点的最小值，Dijkstra 即可
- 海拔高度很大而 M 很小时模拟次数过大无法承受

solution

- 考虑暴力模拟，一个格子的水平面为从海面出发到其路径的最高点的最小值，Dijkstra 即可
- 海拔高度很大而 M 很小时模拟次数过大无法承受
- 容易发现，当所有未被淹没格子都以 M 被侵蚀，且未有新格子露出水面时，侵蚀状态不会改变，可以连续计算多天

solution

- 考虑暴力模拟，一个格子的水平面为从海面出发到其路径的最高点的最小值，Dijkstra 即可
- 海拔高度很大而 M 很小时模拟次数过大无法承受
- 容易发现，当所有未被淹没格子都以 M 被侵蚀，且未有新格子露出水面时，侵蚀状态不会改变，可以连续计算多天
- 为了方便，我们假定海的海拔无穷小，格子的海拔可以为负

solution

- 考虑暴力模拟，一个格子的水平面为从海面出发到其路径的最高点的最小值，Dijkstra 即可
- 海拔高度很大而 M 很小时模拟次数过大无法承受
- 容易发现，当所有未被淹没格子都以 M 被侵蚀，且未有新格子露出水面时，侵蚀状态不会改变，可以连续计算多天
- 为了方便，我们假定海的海拔无穷小，格子的海拔可以为负
- 计算格子的水平面，并判断侵蚀状态是否满足上述条件，若不满足只计算一天，否则可根据被淹没格子的海拔最大值得出连续计算天数

solution

- 考虑暴力模拟，一个格子的水平面为从海面出发到其路径的最高点的最小值，Dijkstra 即可
- 海拔高度很大而 M 很小时模拟次数过大无法承受
- 容易发现，当所有未被淹没格子都以 M 被侵蚀，且未有新格子露出水面时，侵蚀状态不会改变，可以连续计算多天
- 为了方便，我们假定海的海拔无穷小，格子的海拔可以为负
- 计算格子的水平面，并判断侵蚀状态是否满足上述条件，若不满足只计算一天，否则可根据被淹没格子的海拔最大值得出连续计算天数
- 格子露出水面后不会再被淹没，因此连续计算的次数不超过 nm 次

solution

solution

- 若某天没有新格子露出水面，且水面上格子并不都以 M 被侵蚀，则腐蚀后一定会出现新格子以 M 被腐蚀

solution

- 若某天没有新格子露出水面，且水面上格子并不都以 M 被侵蚀，则腐蚀后一定会出现新格子以 M 被腐蚀
- 考虑某个不以 M 被侵蚀的格子 S，假设它流向 T 且 T 以及 T 流向大海的路径均以 M 被侵蚀

solution

- 若某天没有新格子露出水面，且水面上格子并不都以 M 被侵蚀，则腐蚀后一定会出现新格子以 M 被腐蚀
- 考虑某个不以 M 被侵蚀的格子 S，假设它流向 T 且 T 以及 T 流向大海的路径均以 M 被侵蚀
- 由于之前假设海拔可以为负，因此一定能找到这个 S

solution

- 若某天没有新格子露出水面，且水面上格子并不都以 M 被侵蚀，则腐蚀后一定会出现新格子以 M 被腐蚀
- 考虑某个不以 M 被侵蚀的格子 S，假设它流向 T 且 T 以及 T 流向大海的路径均以 M 被侵蚀
- 由于之前假设海拔可以为负，因此一定能找到这个 S
- S 侵蚀过后与 T 原来高度一致，而 T 减少了 M，因此 S 也变为以 M 被侵蚀

solution

- 若某天没有新格子露出水面，且水面上格子并不都以 M 被侵蚀，则腐蚀后一定会出现新格子以 M 被腐蚀
- 考虑某个不以 M 被侵蚀的格子 S，假设它流向 T 且 T 以及 T 流向大海的路径均以 M 被侵蚀
- 由于之前假设海拔可以为负，因此一定能找到这个 S
- S 侵蚀过后与 T 原来高度一致，而 T 减少了 M，因此 S 也变为以 M 被侵蚀
- 单独计算不超过 nm 天后，可以进行一次连续计算

solution

- 若某天没有新格子露出水面，且水面上格子并不都以 M 被侵蚀，则腐蚀后一定会出现新格子以 M 被腐蚀
- 考虑某个不以 M 被侵蚀的格子 S，假设它流向 T 且 T 以及 T 流向大海的路径均以 M 被侵蚀
- 由于之前假设海拔可以为负，因此一定能找到这个 S
- S 侵蚀过后与 T 原来高度一致，而 T 减少了 M，因此 S 也变为以 M 被侵蚀
- 单独计算不超过 nm 天后，可以进行一次连续计算
- 总共 $(nm)^2$ 次单独计算以及 nm 次连续计算

solution

- 若某天没有新格子露出水面，且水面上格子并不都以 M 被侵蚀，则腐蚀后一定会出现新格子以 M 被腐蚀
- 考虑某个不以 M 被侵蚀的格子 S，假设它流向 T 且 T 以及 T 流向大海的路径均以 M 被侵蚀
- 由于之前假设海拔可以为负，因此一定能找到这个 S
- S 侵蚀过后与 T 原来高度一致，而 T 减少了 M，因此 S 也变为以 M 被侵蚀
- 单独计算不超过 nm 天后，可以进行一次连续计算
- 总共 $(nm)^2$ 次单独计算以及 nm 次连续计算
- 总时间复杂度 $O((nm)^3 \log(nm))$

reo

reo

Description

给定 m 对限制条件，每个条件形如：1. 点 a 为点 b 的祖先；2. 点 a 不为点 b 的祖先。现在请你构造出一棵 n 个点的符合所有限制条件的树，或是输出无解。

Constraints

$n \leq 1000$, $m \leq 10000$

Source

PA2016 Round2

solution

solution

- 考虑每次挑选出一个" 根"，并更新所有结点当前合法的父亲

solution

- 考虑每次挑选出一个" 根"，并更新所有结点当前合法的父亲
- 初始时，若一个点不是 1 中的 b 以及 2 中的 a，那么它就能当根

solution

- 考虑每次挑选出一个" 根"，并更新所有结点当前合法的父亲
- 初始时，若一个点不是 1 中的 b 以及 2 中的 a，那么它就能当根
- 先考虑关系 1，若两个点此时均未成为根，则将它们相连，并且 b 不能成为此轮的根

solution

- 考虑每次挑选出一个" 根"，并更新所有结点当前合法的父亲
- 初始时，若一个点不是 1 中的 b 以及 2 中的 a，那么它就能当根
- 先考虑关系 1，若两个点此时均未成为根，则将它们相连，并且 b 不能成为此轮的根
- 接下来考虑关系 2，若两个点此时在同一个连通块中，那么 a 不能成为此轮的根

solution

- 考虑每次挑选出一个" 根"，并更新所有结点当前合法的父亲
- 初始时，若一个点不是 1 中的 b 以及 2 中的 a，那么它就能当根
- 先考虑关系 1，若两个点此时均未成为根，则将它们相连，并且 b 不能成为此轮的根
- 接下来考虑关系 2，若两个点此时在同一个连通块中，那么 a 不能成为此轮的根
- 在还未成为过根并且此轮中可以当根的点中任意选取一点 u，而它所在的连通块中其他点当前将能以 u 作为父亲

solution

- 考虑每次挑选出一个" 根"，并更新所有结点当前合法的父亲
- 初始时，若一个点不是 1 中的 b 以及 2 中的 a，那么它就能当根
- 先考虑关系 1，若两个点此时均未成为根，则将它们相连，并且 b 不能成为此轮的根
- 接下来考虑关系 2，若两个点此时在同一个连通块中，那么 a 不能成为此轮的根
- 在还未成为过根并且此轮中可以当根的点中任意选取一点 u，而它所在的连通块中其他点当前将能以 u 作为父亲
- 并查集维护连通块，时间复杂度 $O(nm)$