网络流

ExfJoe

福建省长乐第一中学

July 25, 2017

开始之前

开始之前

- 上课时间有限，并且许多同学是初学 OI，之前没有接触过太多的算法，所以这节课主要目的是让大部分同学能够听懂并实现网络流算法，做出例题，并理解一些简单的模型

开始之前

- 上课时间有限，并且许多同学是初学 OI，之前没有接触过太多的算法，所以这节课主要目的是让大部分同学能够听懂并实现网络流算法，做出例题，并理解一些简单的模型
- 对于水平较高的同学以及之前学习过网络流的同学来说这节课比较无聊，若不想听可以闷声做事，但不要影响其他同学听课

开始之前

- 上课时间有限，并且许多同学是初学 OI，之前没有接触过太多的算法，所以这节课主要目的是让大部分同学能够听懂并实现网络流算法，做出例题，并理解一些简单的模型
- 对于水平较高的同学以及之前学习过网络流的同学来说这节课比较无聊，若不想听可以闷声做事，但不要影响其他同学听课
- 讲课过程中任何有疑问的地方都可以积极大胆的提问，我会尽力帮助大家解释清楚

Part I

基础知识

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

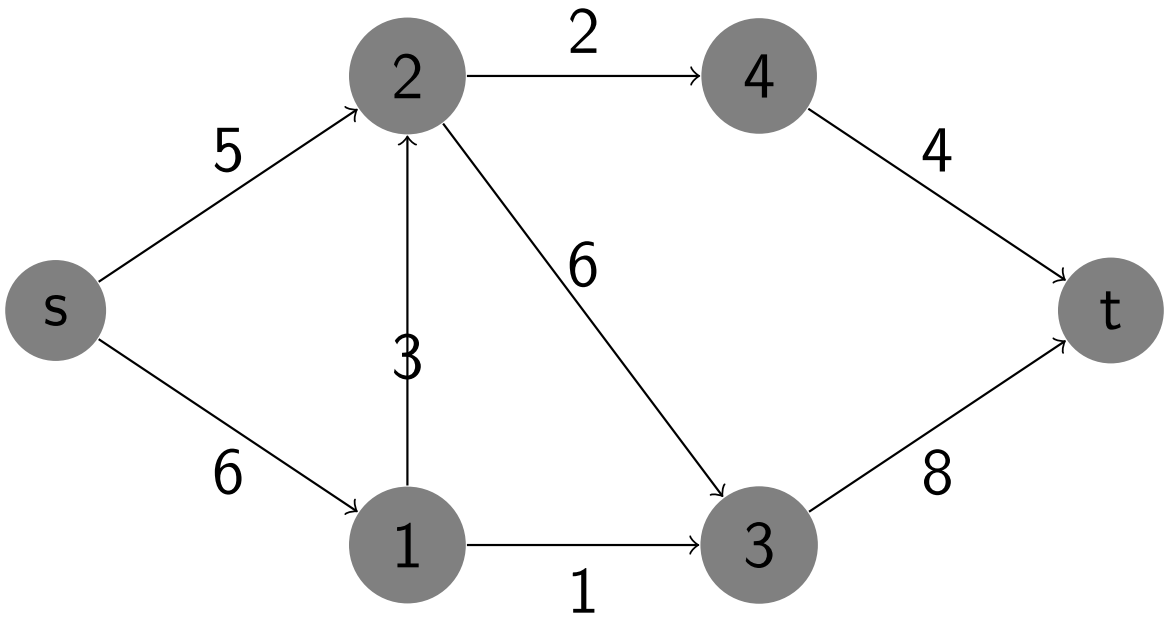
- 如同我们可以把实际的道路地图抽象成一个有向图来计算两点之间最短路，我们也可以将一个有向图看作一个网络流图来解决另一类问题
- 网络流可以用来模拟水流经管道、电流在电路网络中的运动、信息网络中信息的传递等等过程

引入

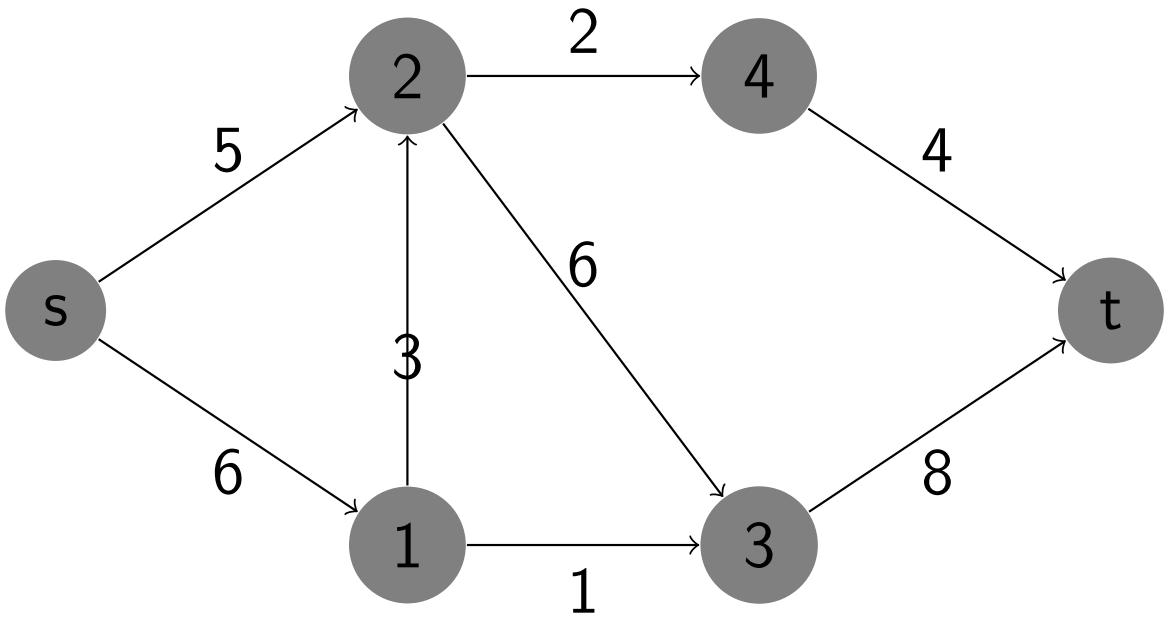
- 如同我们可以把实际的道路地图抽象成一个有向图来计算两点之间最短路，我们也可以将一个有向图看作一个网络流图来解决另一类问题
- 网络流可以用来模拟水流经管道、电流在电路网络中的运动、信息网络中信息的传递等等过程
- 例如我们将网络流类比成水流，图中的边可以想象成管道，顶点则是管道的连接点

引入

引入

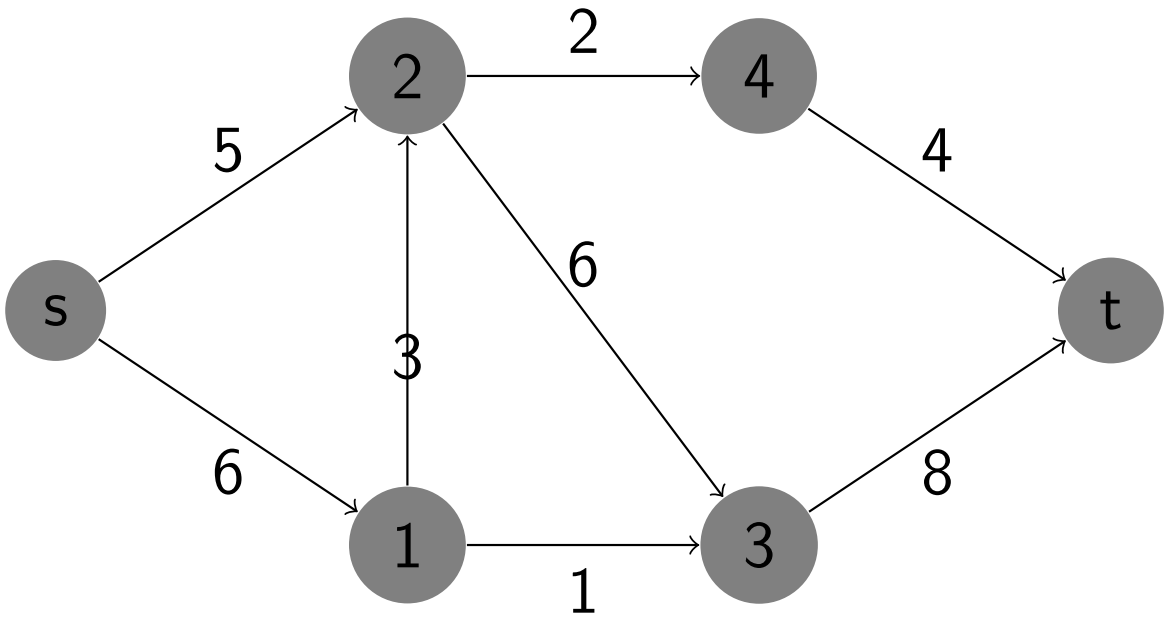


引入



- 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：$0\le f(e)\le c(e), e\in E$
  - 流量守恒：

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

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

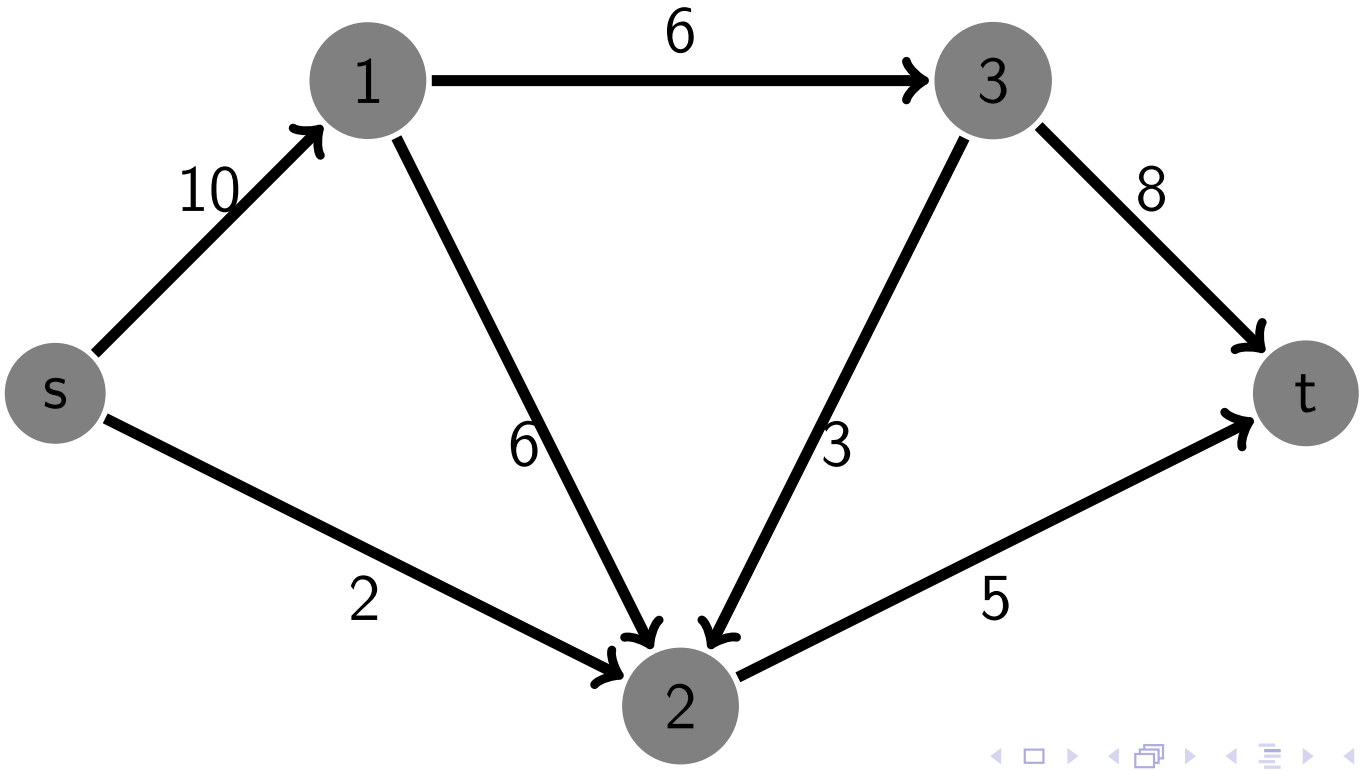
- 考虑如下的贪心算法：
  - 找一条 s 到 t 的只经过 $f(e) < c(e)$ 的边的路径
  - 若不存在满足条件的路径则算法结束，否则，沿着该路径尽可能地增加 $f(e)$，然后返回上一步。我们可以将这步称为增广

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：



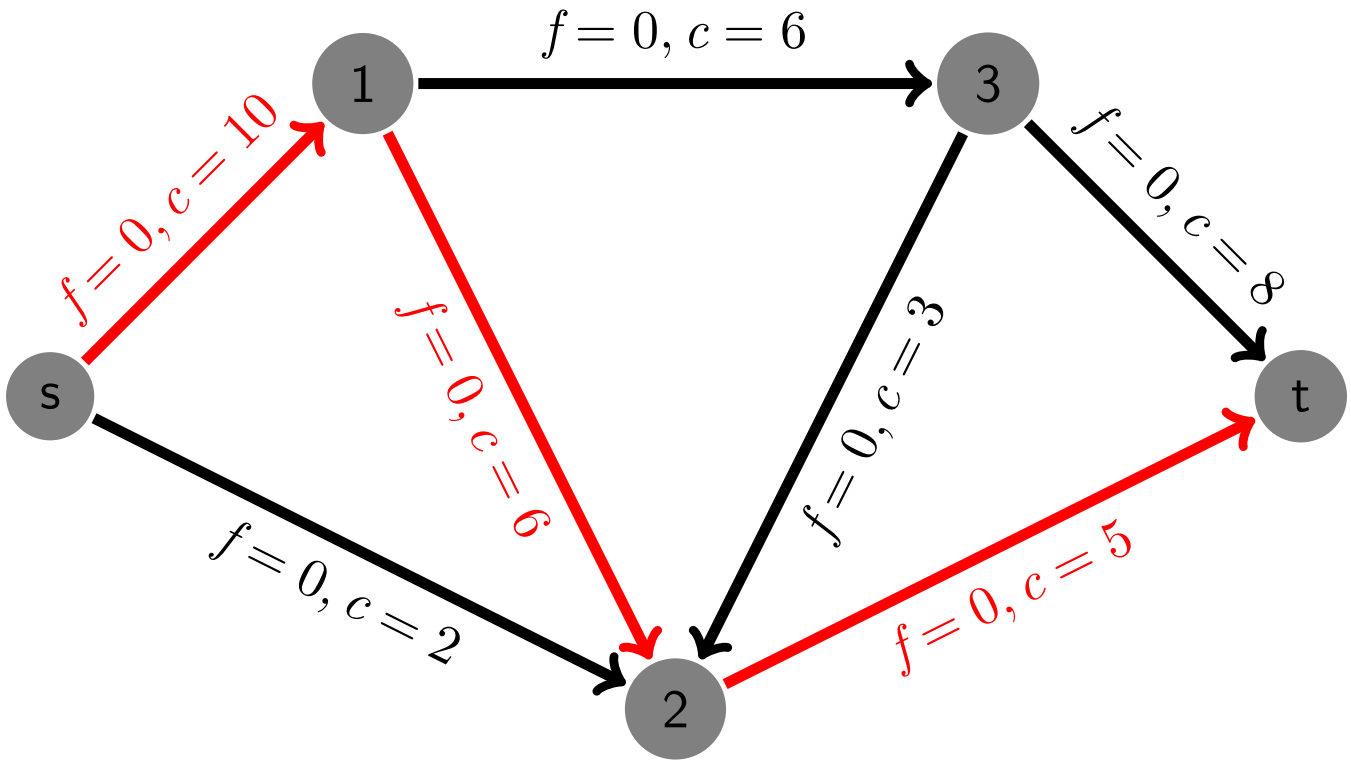
算法流程

算法流程

沿着 $s \to 1 \to 2 \to t$ 流 5

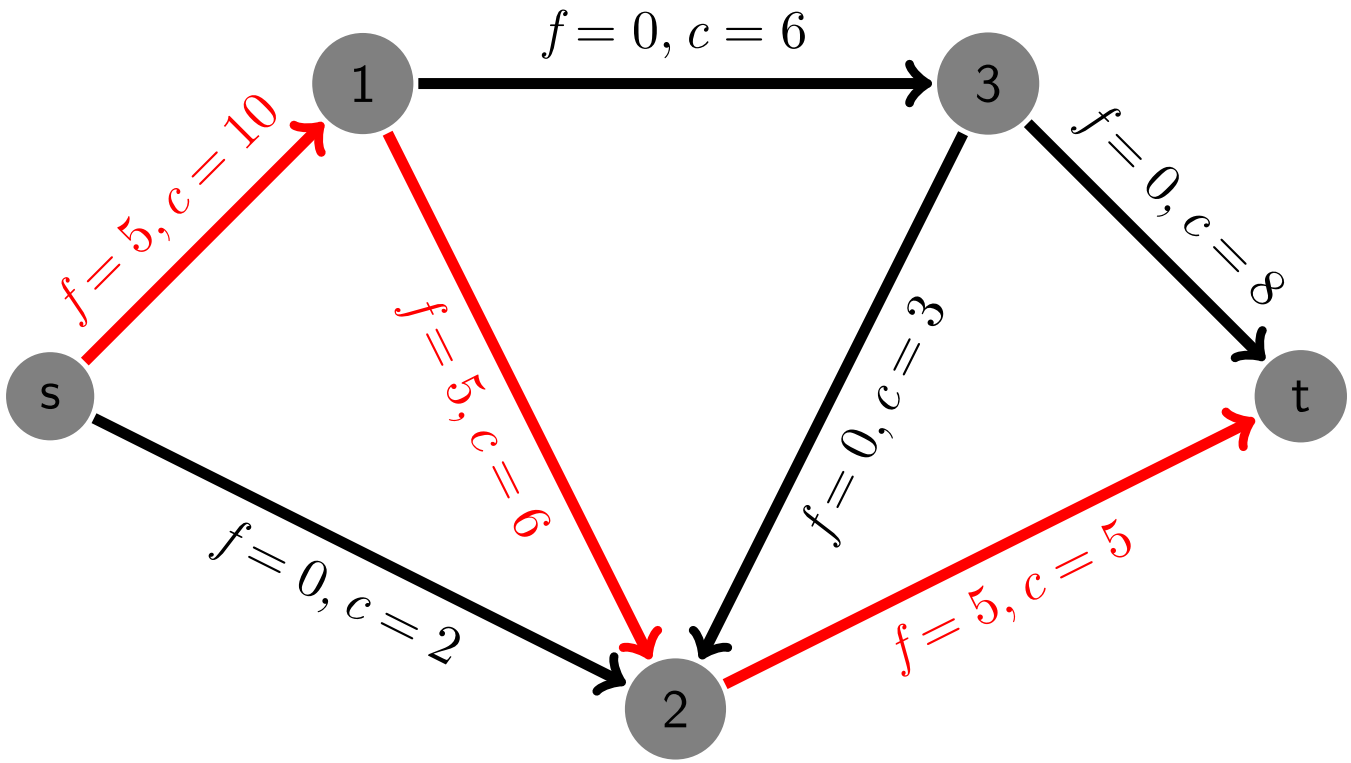
算法流程

沿着 $s \to 1 \to 2 \to t$ 流 5



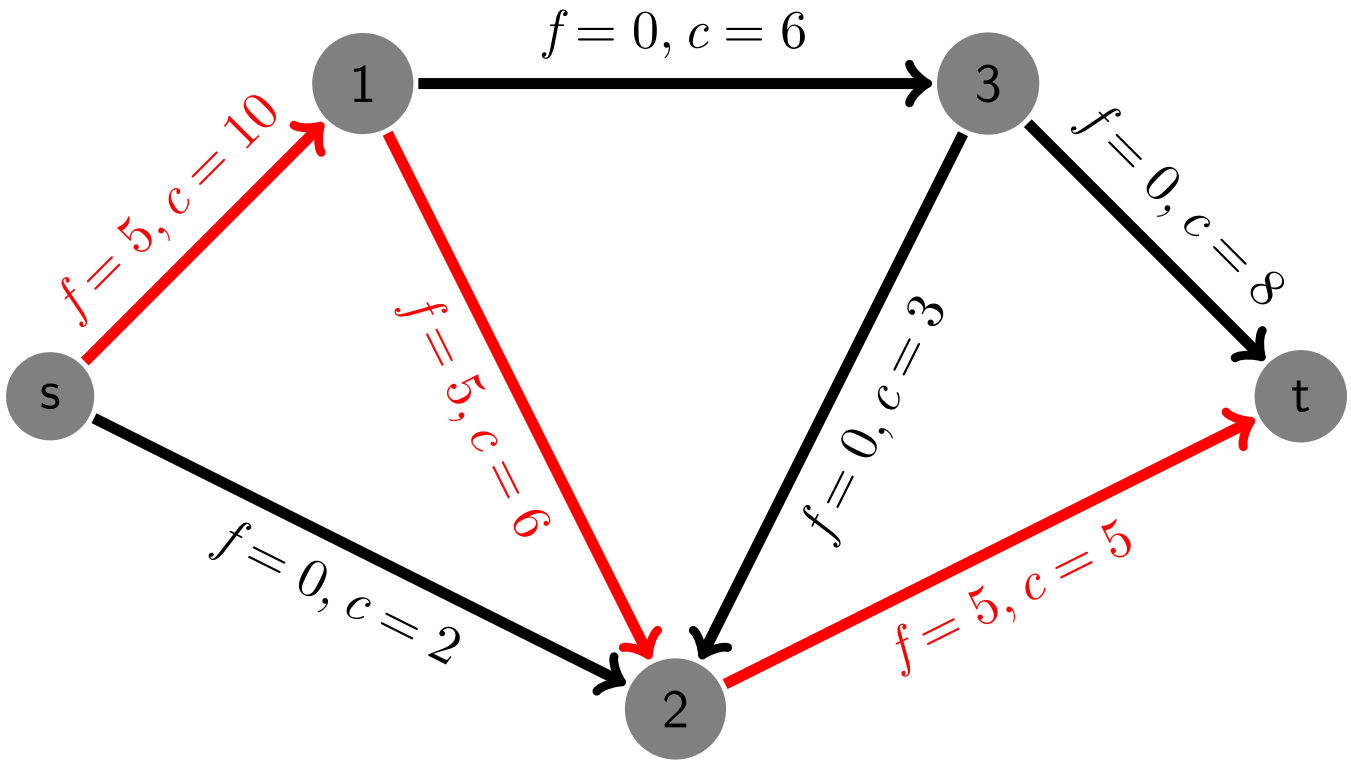
算法流程

沿着 $s \to 1 \to 2 \to t$ 流 5



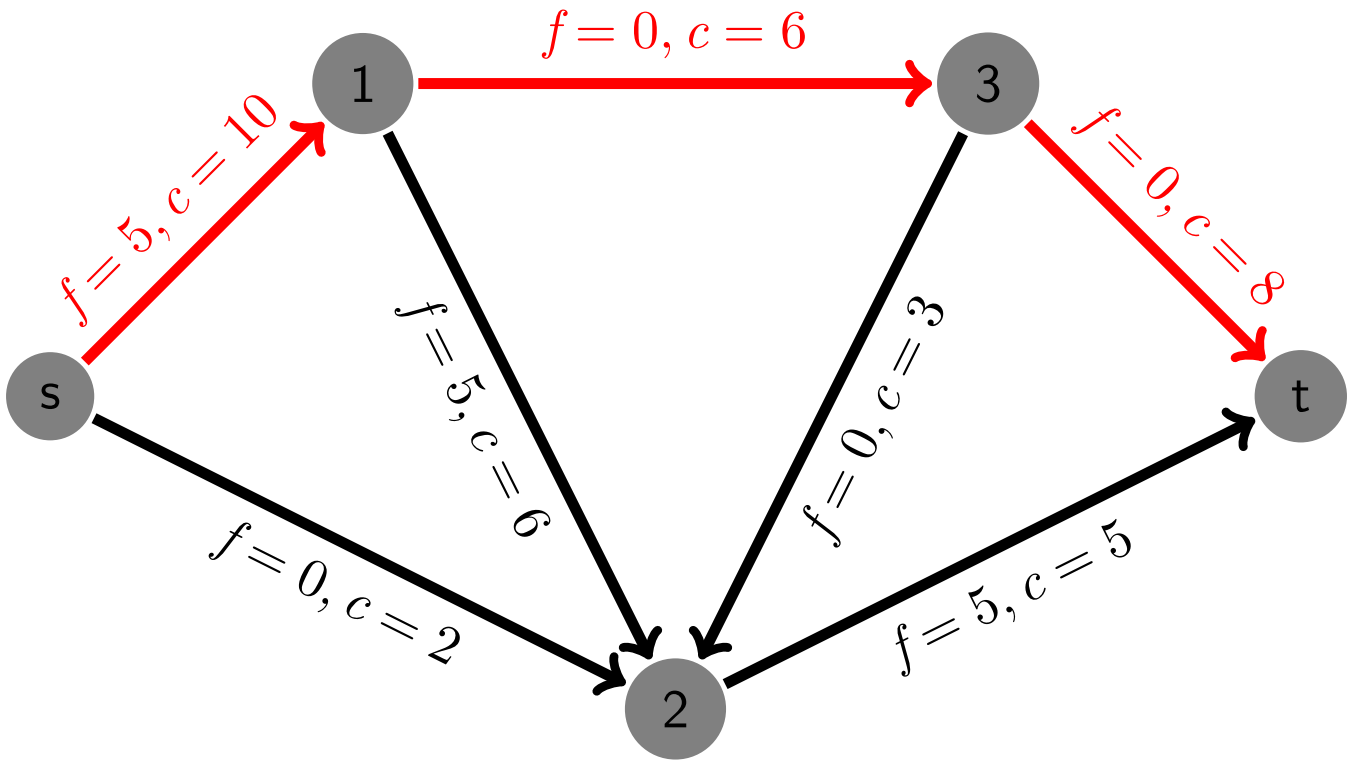
算法流程

再沿着 $s \to 1 \to 3 \to t$ 流 5



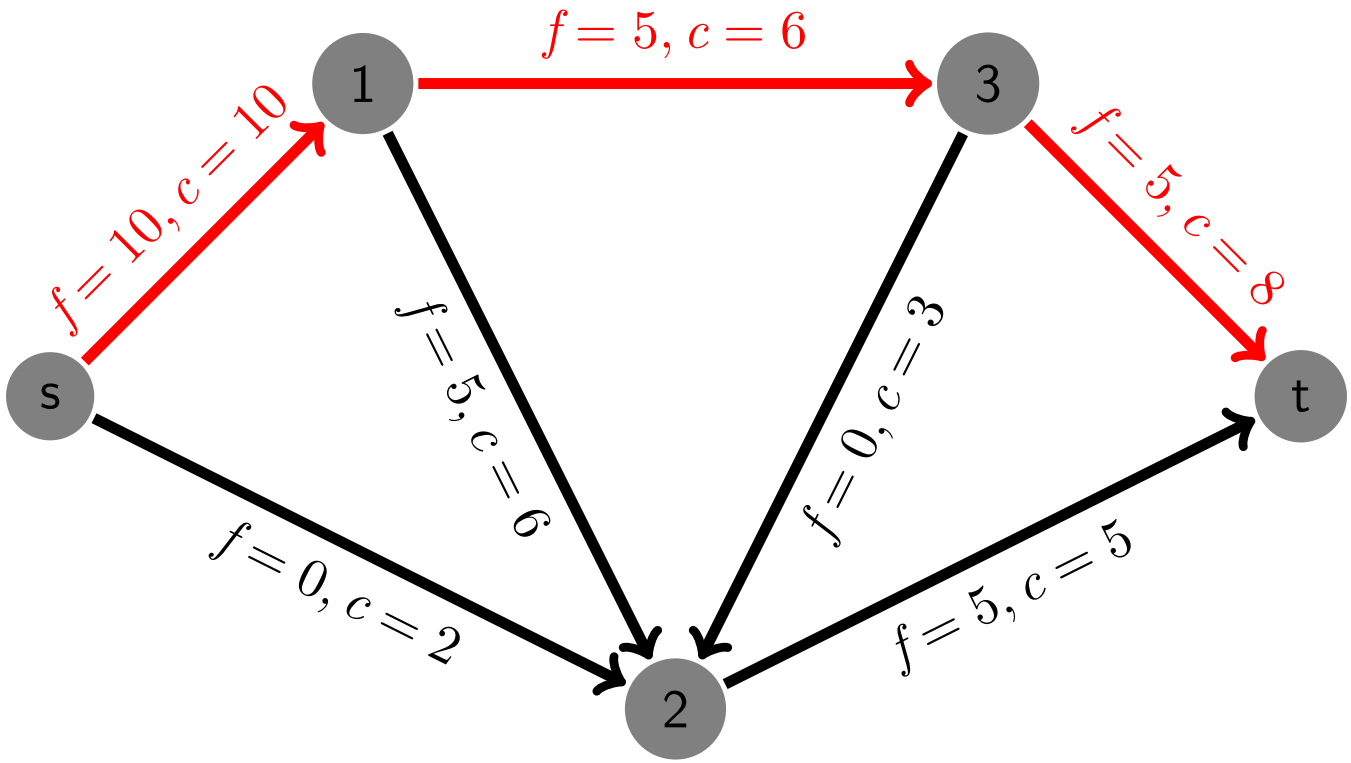
算法流程

再沿着 $s \to 1 \to 3 \to t$ 流 5



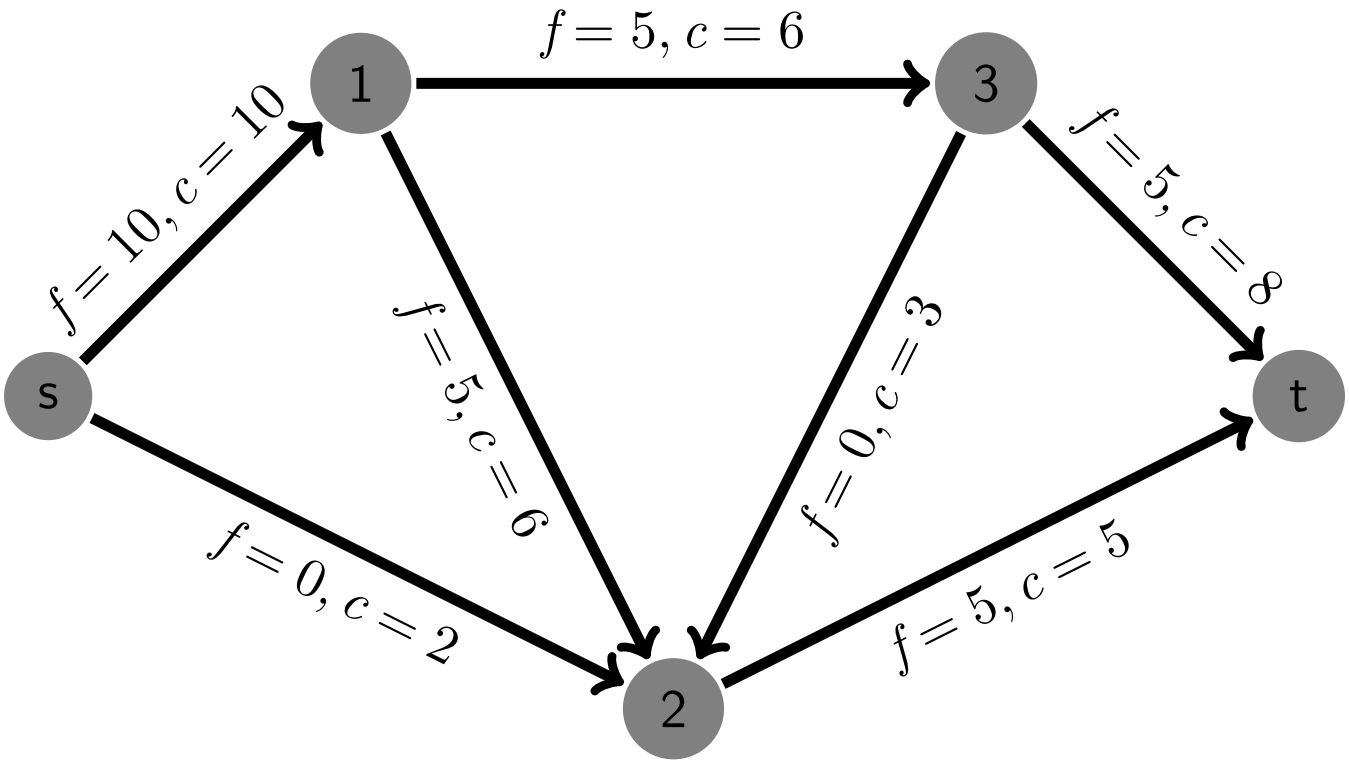
算法流程

再沿着 $s \to 1 \to 3 \to t$ 流 5



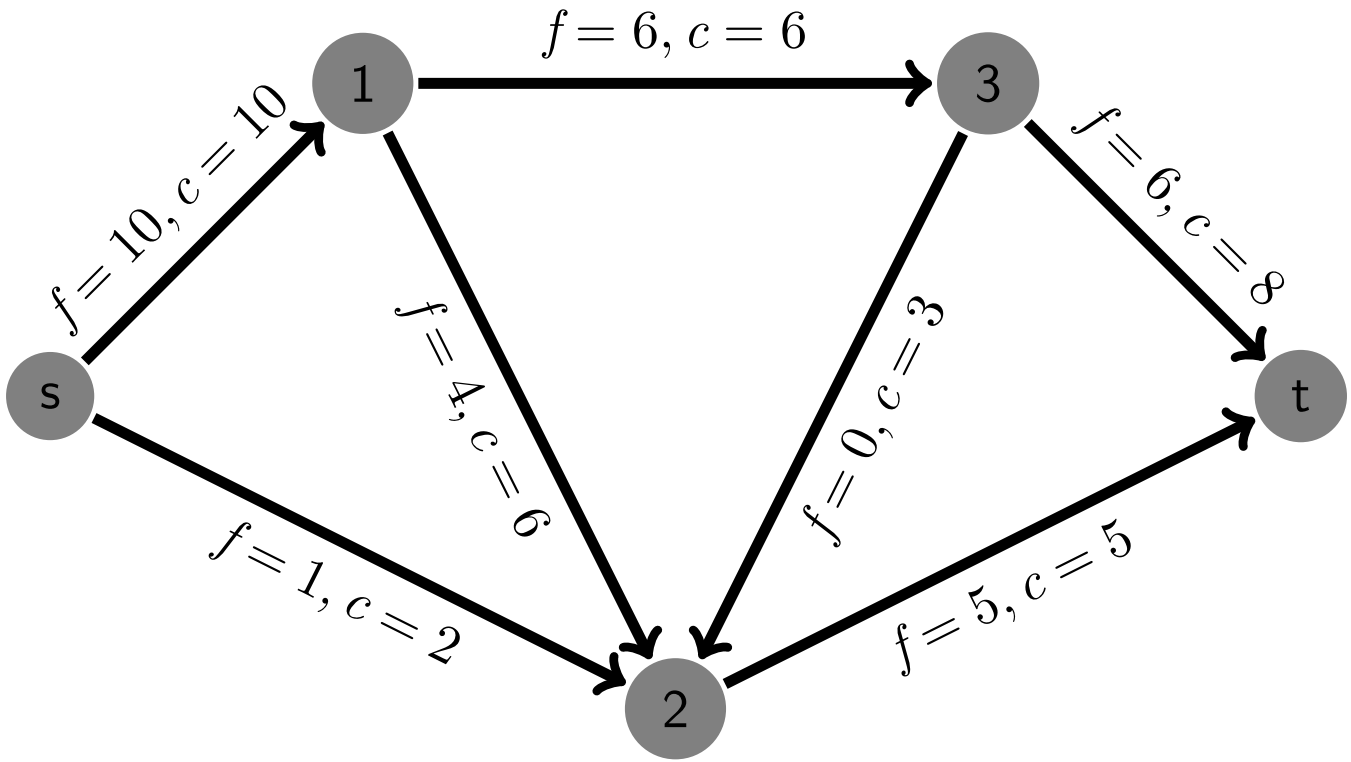
算法流程

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：



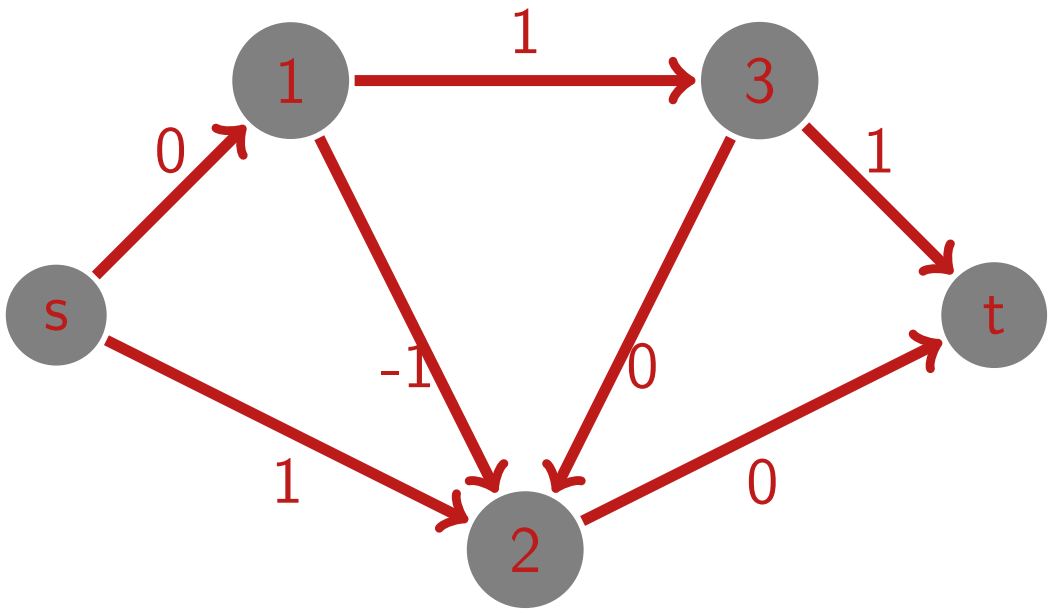
修正算法

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)：

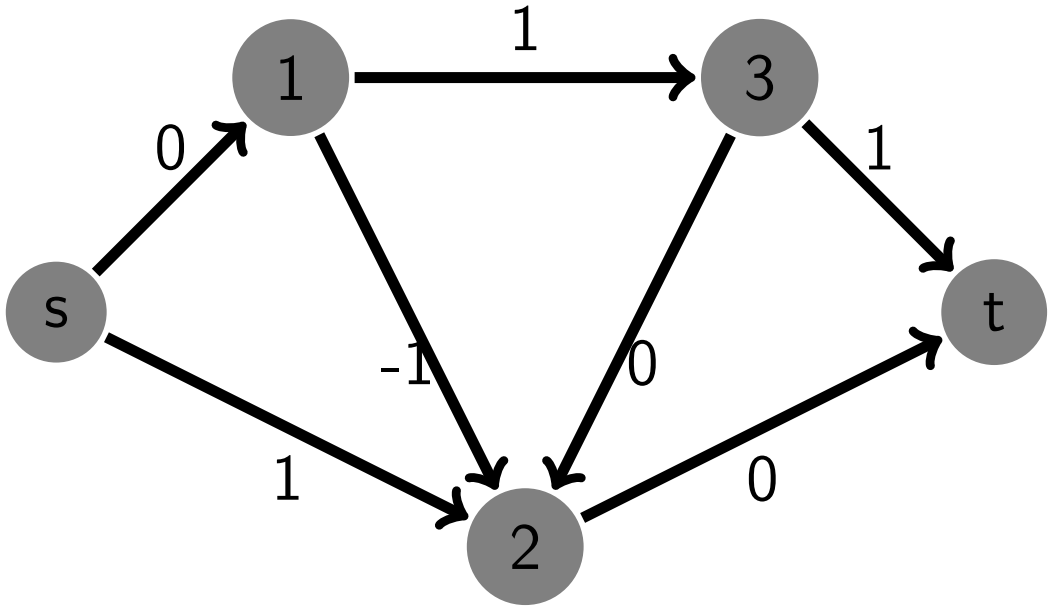
修正算法

- 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

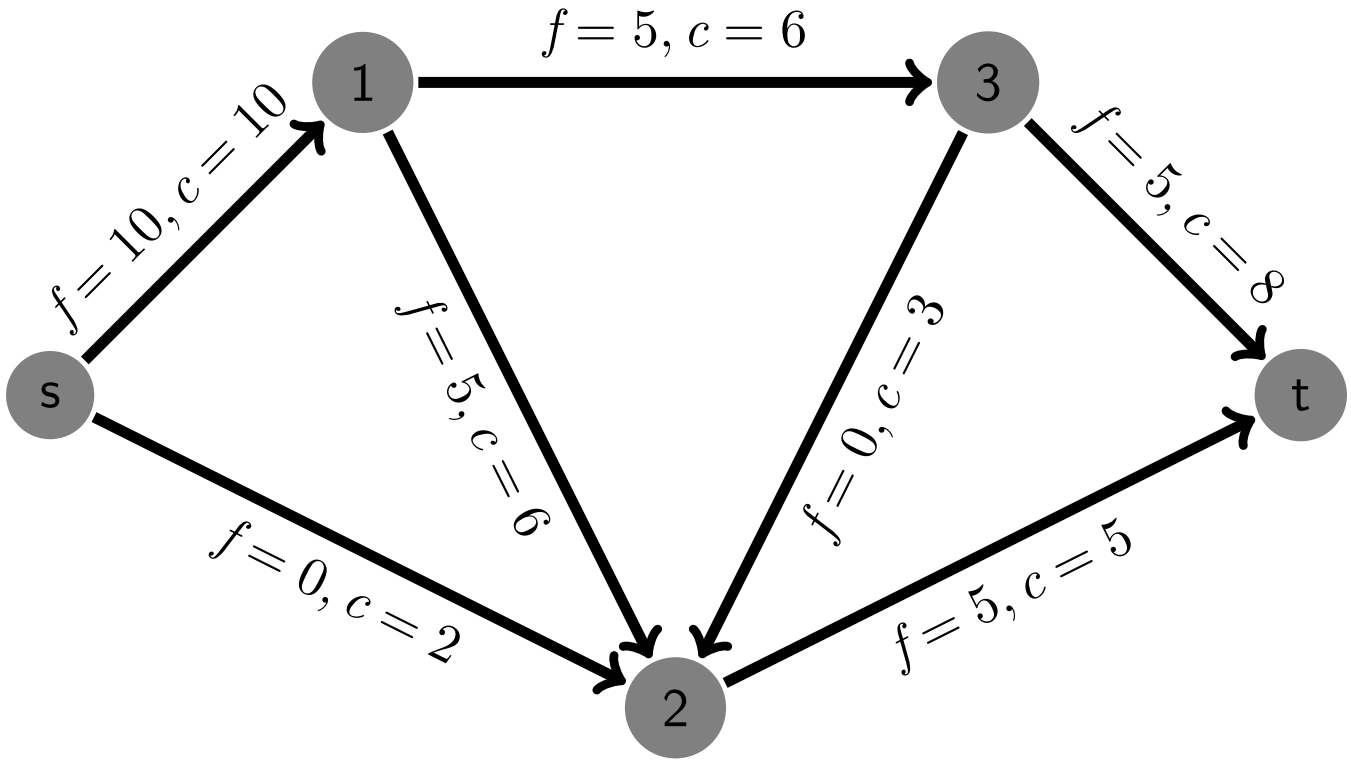
- 即我们可以将原先得到的流退回去并改道，得到新的流
- 此时流量不变，在新流基础上继续增广就能找到更优的答案
- 现在算法改进如下：
  - 只利用 $f(e) < c(e)$ 的 e 或 $f(e) > 0$ 的 e 的反向边 $rev(e)$，寻找一条 s 到 t 的路径
  - 若不存在满足条件的路径则算法结束；否则沿着该路径尽可能增加流，并返回上一步
- 若沿着 (u, v) 的反向边增广流，则相当于将原来的某条流退回去，并将那条流的 $s \to u \to v$ 改成当前流方案 $s \to v$
- 由于最后一定会再走到 t，所以这相当于将那条流改道后，利用那条流的 $s \to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

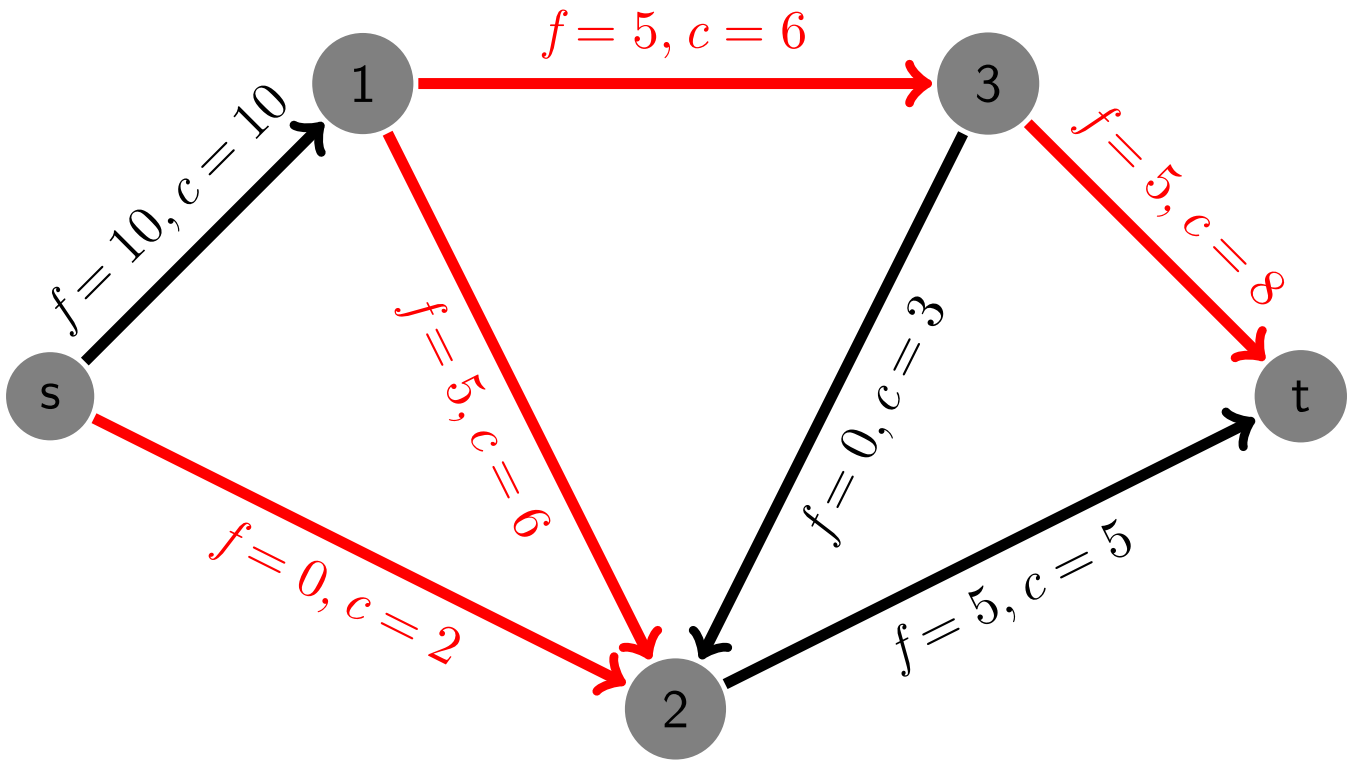
修正算法

修正算法



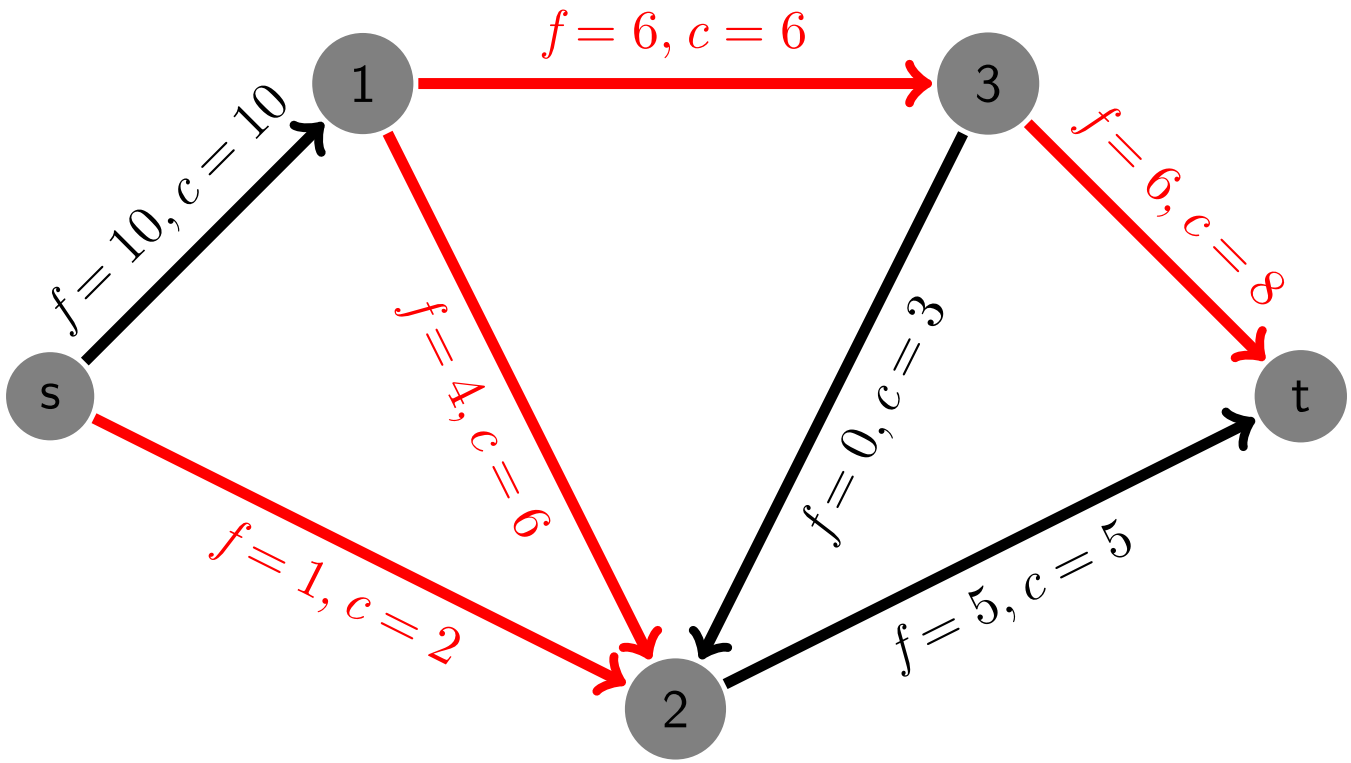
修正算法

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

Part II

常见模型

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