网络流入门

yijan

2024.7

约定

- **网络是指一个特殊的有向图 $G = (V, E)$ 满足边有容量且有源点 s 和汇点 t 。记每条边 (u, v) 容量为 $c(u, v)$ ，可以认为 $c(u, v) = 0$ 时 u, v 无边。**

约定

- **网络是指一个特殊的有向图** $G=(V,E)$ **满足边有容量且有源点** s **和汇点** t **。记每条边** (u,v) **容量为** $c(u,v)$ **，可以认为** $c(u,v)=0$ **时** u,v **无边。**
- **对于网络** G **，流是一个从边集** E **到整数集（或实数集）的映射，满足：**
  - **对每条边有** $0\le f(u,v)\le c(u,v)$
  - **对** $u\notin\{s,t\}$ **有** $f(u)=\sum_{x\in V}f(u,x)-f(x,u)=0$ **，即每个点流量平衡。且流的流量定义为** $f(s)$ **。**

约定

- **网络是指一个特殊的有向图 $G=(V,E)$ 满足边有容量且有源点 s 和汇点 t 。记每条边 (u,v) 容量为 $c(u,v)$ ，可以认为 $c(u,v)=0$ 时 u,v 无边。**
- **对于网络 G ，流是一个从边集 E 到整数集（或实数集）的映射，满足：**
  - **对每条边有 $0\le f(u,v)\le c(u,v)$**
  - **对 $u\notin\{s,t\}$ 有 $f(u)=\sum_{x\in V}f(u,x)-f(x,u)=0$ ，即每个点流量平衡。且流的流量定义为 $f(s)$ 。**
- **对于 $G=(V,E)$ ，如果 $S\cup T=V,S\cap T=\emptyset,s\in S,t\in T$ ，则称 $\{S,T\}$ 为 G 的一个 $s-t$ 割，定义其容量为 $\sum_{u\in S}\sum_{v\in T}c(u,v)$ 。**
- **如果没有特殊说明，$n=|V|,m=|E|$ 。**

最大流问题

- 对于网络 $G=(V,E)$，给每条边指定流量得到流 f 使 f 的流量尽量大，此时我们称 f 是 G 的最大流。
- 求解最大流的经典算法有 EK 算法和 Dinic 算法。

增广

- 首先，对于一条边 (u, v) 和流 f，我们称其容量和流量的差为剩余容量 $c_f(u, v)$，将 G 中所有节点和剩余容量大于 0 的边构成的子图称为残量网络 G_f。

增广

- 首先，对于一条边 (u, v) 和流 f，我们称其容量和流量的差为剩余容量 $c_f(u, v)$，将 G 中所有节点和剩余容量大于 0 的边构成的子图称为残量网络 G_f。
- 对于 G_f 上的一条 s 到 t 的路径称为增广路，对于增广路的所有边增加相同流量的过程被称作增广。显然增广会使整个图的流量增加。

增广

- 首先，对于一条边 (u, v) 和流 f，我们称其容量和流量的差为剩余容量 $c_f(u, v)$，将 G 中所有节点和剩余容量大于 0 的边构成的子图称为残量网络 G_f。
- 对于 G_f 上的一条 s 到 t 的路径称为增广路，对于增广路的所有边增加相同流量的过程被称作增广。显然增广会使整个图的流量增加。
- 同时，在增广的过程中，我们对每条边 (u, v) 新建一条反向边 (v, u)，约定 $f(v, u) = -f(u, v)$，也就是说在增广的过程中会对反向边的流量减少相同的量，即退流。

Edmonds-Karp 算法

- 在一个网络上，显然可以不断进行增广，直到无法找到增广路为止，此时网络的流量不可能继续提高，就找到了最大流，这就是 Ford-Fulkerson 增广的过程。

Edmonds-Karp 算法

- 在一个网络上，显然可以不断进行增广，直到无法找到增广路为止，此时网络的流量不可能继续提高，就找到了最大流，这就是 Ford-Fulkerson 增广的过程。
- 在具体实现时，我们可以使用 BFS 找到增广路，然后更新 G_f，这就是 EK 算法的流程：
  - 如果在残量网络 G_f 上可以通过 BFS 找到 $s \to t$ 的路径，则我们发现了增广路，记录每个点 BFS 过程中的的父节点。
  - 对于找到的增广路 p，取其边的剩余容量的最小值 d，给每条边的流量都加上 d 并给反向边的流量都减去 d，得到 f'。
  - 在新的 $G_{f'}$ 上重复该过程，直到不存在 $s \to t$ 的路径。

Edmonds-Karp 算法

时间复杂度和细节

- 如果用邻接矩阵存图，那么反向边是很好找到的。否则，我们往往用前向星存图，并且初始的边的编号设置为 1 ，从而保证正边和反边恰好是奇偶性不同，也就是 $i \leftrightarrow i \oplus 1$ 。
- 显然，EK 算法在找一条增广路时复杂度是 $O(m)$ 的，找到最大流的时间取决于增广的次数。
- 可以证明，EK 算法找到最大流时增广的次数上界为 $O(nm)$ ，具体参考 OI-Wiki 。后文所有忽略的证明都可以参考这篇文章。

Edmonds-Karp 算法

时间复杂度和细节

- 如果用邻接矩阵存图，那么反向边是很好找到的。否则，我们往往用前向星存图，并且初始的边的编号设置为 1 ，从而保证正边和反边恰好是奇偶性不同，也就是 $i \leftrightarrow i \oplus 1$ 。
- 显然，EK 算法在找一条增广路时复杂度是 $O(m)$ 的，找到最大流的时间取决于增广的次数。
- 可以证明，EK 算法找到最大流时增广的次数上界为 $O(nm)$ ，具体参考 OI-Wiki 。后文所有忽略的证明都可以参考这篇文章。
- 如果你只想要找到一个流量为 w 的网络流，那么复杂度显然不高于 $O(wm)$ 。

Dinic 算法

- Dinic 算法是在 EK 算法的基础上，讲每次寻找一条增广路变成计算一个增广网络。

Dinic 算法

- Dinic 算法是在 EK 算法的基础上，讲每次寻找一条增广路变成计算一个增广网络。
- 如果残量网络中从 s 出发到一条边两端的两点距离恰好差 1，我们称其为**可行边**。所有可行边构成的图，也就是最短路图，成为可行网络。

Dinic 算法

- Dinic 算法是在 EK 算法的基础上，讲每次寻找一条增广路变成计算一个增广网络。
- 如果残量网络中从 s 出发到一条边两端的两点距离恰好差 1，我们称其为**可行边**。所有可行边构成的图，也就是最短路图，成为可行网络。
- 在可行网络上的无法再扩充流量的流称为**阻塞流**，阻塞流不一定是残量网络的最大流。

Dinic 算法

算法流程

- 在 G_f 上通过 BFS 得到可行网络，也就是求出源点到每个点的最短路，只考虑所有 $d_v = d_u + 1$ 的边。
- 在可行网络中 DFS 出阻塞流 f_b 。
- 将 f_b 加入到原来的流中，即 $f' = f + f_b$ 。
- 重复上面的过程直到不存在 $s \to t$ 的路径。

Dinic 算法

算法流程和优化

- BFS 的过程是非常简单的，重要的是如何求出阻塞流。

```
int dfs( int u , int lim ) {
    if( !lim || u == t ) return lim;
    int flow = 0;
    for( int& i = cur[u] ; ~i ; i = nex[i] ) {
        int v = to[i];
        if( dep[v] == dep[u] + 1 ) {
            int f = dfs( v , min( lim , wto[i] ) );
            lim -= f , flow += f , wv -= f , wto[i ^ 1] += f;
            if( !lim ) break;
        }
    }
    return flow;
}
```

- 这份代码加入了当前弧优化（用一个数组存储当前考虑到每个点的哪个出边了）。这份代码的优化均不能省略。

Dinic 算法

复杂度分析

- 我摆烂了，反正复杂度是 $O(n^2m)$ 的，有兴趣自己去读。

Dinic 算法

复杂度分析

- 我摆烂了，反正复杂度是 $O(n^2m)$ 的，有兴趣自己去读。
- 实际上多数时候 Dinic 的复杂度是能过，不用太纠结复杂度本身，因为建模得到的图往往是具有很强的性质的，不能卡的很满，Dinic 跑起来就很快。
- 相对来说更重要的是把问题转化为流的模型的能力。

Dinic 算法

特殊复杂度分析

- 在单位容量的图上，Dinic 的复杂度是 $O(\min(m^{\frac{2}{3}}, n^{\frac{1}{2}})m)$ 的。

Dinic 算法

特殊复杂度分析

- 在单位容量的图上，Dinic 的复杂度是 $O(\min(m^{\frac{2}{3}}, n^{\frac{1}{2}})m)$ 的。
- 在单位网络的图上，Dinic 的复杂度是 $O(n^{\frac{1}{2}}m)$ 的。这里单位网络图是指每个点要么只有一个入边，要么只有一个出边。例如二分图匹配问题。

最小割

- 最大流和最小割是对偶的问题。一个网络的 $s \to t$ 的最大流的大小就是这张图的 $s-t$ 最小割。
- 任意流的大小都小于等于割的大小：

$$
\begin{aligned}
|f| &= f(s) \\
&= \sum_{u \in S} f(u) \\
&= \sum_{u \in S} \left(\sum_{v \in T} f(u,v) - \sum_{v \in T} f(v,u) \right) \\
&\leq \sum_{u \in S} \sum_{v \in T} f(u,v) \leq ||S - T||
\end{aligned} \tag{1}
$$

最小割

- 等号在最大流和最小割成立。最大流执行结束后，由于没有增广路了，图显然按照 s 是否能走到分成 S, T 两份，自然就形成了一个割。在这个割上，对于 $u \in S, v \in T$ 有 $f(u, v) = c(u, v), f(v, u)$ 。

最小割

- 等号在最大流和最小割成立。最大流执行结束后，由于没有增广路了，图显然按照 s 是否能走到分成 S, T 两份，自然就形成了一个割。在这个割上，对于 $u \in S, v \in T$ 有 $f(u, v) = c(u, v), f(v, u)$ 。
- 因此我们就有了一个求一张图最小割的算法。首先通过 Dinic 求出最大流，然后通过搜索找到与 s 连通的点，从而构造出一张图的最小割。

二分图的最大流和最小割

- 最简单的最大流的应用就是二分图最大匹配问题。从源点向每个左部点连权为 1 的边，中间边权均为 $+\infty$ ，右部点向汇点连权为 1 的边求最大流即可。由前文所讲，复杂度为 $O(m\sqrt{n})$ 。

二分图的最大流和最小割

- 最简单的最大流的应用就是二分图最大匹配问题。从源点向每个左部点连权为 1 的边，中间边权均为 $+\infty$ ，右部点向汇点连权为 1 的边求最大流即可。由前文所讲，复杂度为 $O(m\sqrt{n})$ 。
- 二分图中，最小割就是最小点覆盖。和源或汇割开的点看作选择的点，那么每条边两边至少选一个点，因此是最小点覆盖。

二分图的最大流和最小割

- 最简单的最大流的应用就是二分图最大匹配问题。从源点向每个左部点连权为 1 的边，中间边权均为 $+\infty$ ，右部点向汇点连权为 1 的边求最大流即可。由前文所讲，复杂度为 $O(m\sqrt{n})$ 。
- 二分图中，最小割就是最小点覆盖。和源或汇割开的点看作选择的点，那么每条边两边至少选一个点，因此是最小点覆盖。
- 一般图中，我们知道独立集等于点覆盖取反。因此二分图中有最小点覆盖等于最大匹配等于 n 减去最大独立集。

费用流模型

- 如果对于网络 G，每个点除了有容量限制 c 还有单位容量的费用 w，且对于流 f，每条边 (u,v) 的费用为 $f(u,v)\times w(u,v)$，则该网络中总花费最小的最大流称作最小费用最大流（MCMF）。
- 注意，费用流是在最大化流量的前提下让费用最小。
- 一般来说我们认为费用满足斜对称性 $w(u,v)=-w(v,u)$。

SSP 算法

- SSP 算法是一个贪心算法，其思路是找到单位费用最小的增广路进行增广，直到图上不存在增广路为止。

SSP 算法

- SSP 算法是一个贪心算法，其思路是找到单位费用最小的增广路进行增广，直到图上不存在增广路为止。
- 如果图上存在负费用的圈，那么 SSP 算法无法正确求解 MCMF ，需要先通过一些操作消去负圈。

SSP 算法

- SSP 算法是一个贪心算法，其思路是找到单位费用最小的增广路进行增广，直到图上不存在增广路为止。
- 如果图上存在负费用的圈，那么 SSP 算法无法正确求解 MCMF ，需要先通过一些操作消去负圈。
- 具体流程和 dinic 几乎没有区别，唯一的区别就是将 BFS 分层的过程改为使用 SPFA 来求出整个图的最短路，并按照 $d_v = d_u + w(u, v)$ 来决定可行网络。

SSP 算法

- SSP 算法是一个贪心算法，其思路是找到单位费用最小的增广路进行增广，直到图上不存在增广路为止。
- 如果图上存在负费用的圈，那么 SSP 算法无法正确求解 MCMF ，需要先通过一些操作消去负圈。
- 具体流程和 dinic 几乎没有区别，唯一的区别就是将 BFS 分层的过程改为使用 SPFA 来求出整个图的最短路，并按照 $d_v = d_u + w(u, v)$ 来决定可行网络。
- 这个算法的最坏复杂度是 $O(nmf)$ 的，其中 f 是最大流的大小，最坏可以达到指数级别的复杂度。但是一般的费用流模型仍然图的性质非常特殊，因此复杂度也是“能过”。

Primal-Dual

- 众所周知，SPFA 的复杂度是 $O(nm)$ ，但是在有负权的图上无法直接应用 Dijkstra ，因此可以使用原始对偶算法来消除负边。

Primal-Dual

- 众所周知，SPFA 的复杂度是 $O(nm)$ ，但是在有负权的图上无法直接应用 Dijkstra ，因此可以使用原始对偶算法来消除负边。
- 在最初的图上，跑一次最短路算法。对于每个点，设置其势能 $h_i = d_i$ 。接下来对于每条边 (u, v) ，设置 $w'(u, v) = w(u, v) + h_u - h_v$ 。可以发现这样设置势能后，由最短路的性质每条边的权值均非负，且新图的最短路和原图最短路一一对应。

Primal-Dual

- 众所周知，SPFA 的复杂度是 $O(nm)$ ，但是在有负权的图上无法直接应用 Dijkstra ，因此可以使用原始对偶算法来消除负边。
- 在最初的图上，跑一次最短路算法。对于每个点，设置其势能 $h_i = d_i$ 。接下来对于每条边 (u, v) ，设置 $w'(u, v) = w(u, v) + h_u - h_v$ 。可以发现这样设置势能后，由最短路的性质每条边的权值均非负，且新图的最短路和原图最短路一一对应。
- 问题是每次增广后图的形态会有变化。事实上，每次增广后如果源点向 u 的最短距离为 d'_u ，只需要让 $h'_u = h_u + d'_u$ 即可。

Primal-Dual

- 下面说明增广结束时每个点的权值都非负。
- 对于一条增广路上的边 (u, v)，残量网络上会多处 (v, u) 边，且满足

$$d'_v + w(u, v) + h_u - h_v = d'_u$$

故有 $w(v, u) + (h_v - d'_v) - (h_u - d'_u) = 0$，因此新增边权值为 0 非负。

Primal-Dual

- 下面说明增广结束时每个点的权值都非负。
- 对于一条增广路上的边 (u, v)，残量网络上会多处 (v, u) 边，且满足

$$d'_v + w(u, v) + h_u - h_v = d'_u$$

故有 $w(v, u) + (h_v - d'_v) - (h_u - d'_u) = 0$，因此新增边权值为 0 非负。

- 对于原来就存在的边，在增广前做最短路时有

$$d'_u + (w(u, v) + h_u - h_v) - d'_v \geq 0$$

因此有 $w(u, v) + (d'_u + h_u) - (d'v + h_v) = 0$，即新增的边非负。

上下界网络流

- 上下界网络流指的是每条边有流量的上界和下界的网络流，也就是说一个可行的流必须满足 $b(u,v) \leq f(u,v) \leq c(u,v)$ ，同时除源汇之外流量平衡。
- 在上下界网络流中往往有可行流和最大流两个问题，解决方法基本基于合理构造一张图并将上下界流问题转化为一般问题。

上下界可行流

- 给定无源汇网络 G，需要判断是否存在每条边的流量使得整个图流量平衡。

上下界可行流

- 给定无源汇网络 G，需要判断是否存在每条边的流量使得整个图流量平衡。
- 我们可以假装每条边其实已经流了 b 的流量了，也就是说让每条 $u \to v$ 的边只有 $c - b$ 的容量。

上下界可行流

- 给定无源汇网络 G，需要判断是否存在每条边的流量使得整个图流量平衡。
- 我们可以假装每条边其实已经流了 b 的流量了，也就是说让每条 $u \to v$ 的边只有 $c - b$ 的容量。
- 此时问题是即使在最开始有些点可能流量就不平衡。我们设对一个点 u 它的最初流入流量减去流出流量为 M，那么：
  - $M > 0$ 时从虚点 S' 向这个点连容量为 M 的边。
  - $M < 0$ 时从这个点向虚点 T' 连容量为 $-M$ 的边。
  - $M = 0$ 时什么也不需要做。
- 计算 $S' \to T'$ 的最大流，可以流满则说明存在可行流，否则不存在。

上下界可行流

- 给定无源汇网络 G，需要判断是否存在每条边的流量使得整个图流量平衡。
- 我们可以假装每条边其实已经流了 b 的流量了，也就是说让每条 $u \to v$ 的边只有 $c - b$ 的容量。
- 此时问题是即使在最开始有些点可能流量就不平衡。我们设对一个点 u 它的最初流入流量减去流出流量为 M，那么：
  - $M > 0$ 时从虚点 S' 向这个点连容量为 M 的边。
  - $M < 0$ 时从这个点向虚点 T' 连容量为 $-M$ 的边。
  - $M = 0$ 时什么也不需要做。
- 计算 $S' \to T'$ 的最大流，可以流满则说明存在可行流，否则不存在。
- 如果原网络存在源和汇，则只需要从 T 向 S 连一条上界 ∞ 的边。

上下界最大流

- 首先找到一个可行流，也就是添加虚点和边并跑出一个可行流。
- 然后在可行流的基础上跑原来的 $S \to T$ 的最大流。这个时候是不可能在增广途中经过 S', T' 的，所以限制的下界总是被满足。给两部分加起来就行了。

上下界最大流

- 首先找到一个可行流，也就是添加虚点和边并跑出一个可行流。
- 然后在可行流的基础上跑原来的 $S \to T$ 的最大流。这个时候是不可能在增广途中经过 S', T' 的，所以限制的下界总是被满足。给两部分加起来就行了。
- 如果要求最小流，删除最后那条 $T \to S$ 的边再从 T 到 S 跑最大流，减去这个值即可。

网络流建模

- 网络流的难度往往体现在怎么把实际问题转化为网络流问题，然后用网络流算法解决。
- 网络流建模主要分为两个部分：最大流和最小割。彻底理解一些基础的模型也是很重要的。

例题 1

- 给出 DAG ， 选出最少的不交路径，覆盖所有点。
- $1 \leq n \leq 150, 1 \leq m \leq 6000$

例题 1

- 最初我们让每个点自己覆盖一个点，需要 n 个路径。
- 每次可以合并两个有边的路径让路径数减少正好 1 。

例题 1

- 最初我们让每个点自己覆盖一个点，需要 n 个路径。
- 每次可以合并两个有边的路径让路径数减少正好 1 。
- 对每个点 u 建 u_l, u_r ， $S \to u_l, u_r \to T$ 连 1 。
- 对原图的边 u, v ，建 $u_l \to v_r$ ，容量为 1 。
- 答案为 n 减最大流。

例题 1.1

- 给出 DAG ，选出最少的路径，允许相交，覆盖所有点。
- $1 \leq n \leq 150, 1 \leq m \leq 6000$

例题 1.1

- 先对图执行传递闭包，然后就是上一个问题。

例题 1.2

- 给出 DAG ，选出一些的路径，允许相交，覆盖所有点。
- 每个点作为起点有代价，每条边也有代价，路径的代价是起点代价和边的代价和，选出代价最小的选法。
- $1 \leq n \leq 150, 1 \leq m \leq 6000$

例题 1.2

- 类似地建图。可以发现 $u_l \to v_r$ 如果流了流量就说明这个点不是起点。因此可以连 $S \to u_r$ 的边表示 u 作为一个起点，最后最大流一定是 n 。
- 对于每条边都可以设置一个费用，计算最小费用最大流即可。

例题 1.3

- 给出 DAG ，选出一些的路径，允许相交，覆盖所有点。
- 每个点作为起点有代价，每条边也有代价，路径的代价是起点代价和边的代价和，选出代价最小的选法。
-
- $1 \leq n \leq 150, 1 \leq m \leq 6000$

例题 1.3

- 类似地建图。可以发现 $u_l \to v_r$ 如果流了流量就说明这个点不是起点。因此可以连 $S \to u_r$ 的边表示 u 作为一个起点，最后最大流一定是 n 。
- 对于每条边都可以设置一个费用，计算最小费用最大流即可。

例题 2

- 定义一个序列是好的当且仅当为任意相邻元素差为 1 或模 7 同余。
- 给出一个长为 n 的序列，求出这个序列选出 4 个互不相交的好的子序列的方案，使得它们的并尽量大，输出大小。
- $n \leq 3000, A_i \leq 10^5$ 。

例题 2

- 一种常见的建图技巧：对于“每个点只能选一次”的限制，建立入点和出点，从入点向出点连容量为 1 的边，费用为 0 。
- 对于 $i \to j$ 可达，连 $out_i \to in_j$ 的边，费用为 1 。
- 从 S 到入点连边，从出点到 T 连边，费用为 0 。
- 只需要求 $S \to T$ 的容量为 4 的费用流即可。

例题 2

- 一种常见的建图技巧：对于“每个点只能选一次”的限制，建立入点和出点，从入点向出点连容量为 1 的边，费用为 0 。
- 对于 $i \to j$ 可达，连 $out_i \to in_j$ 的边，费用为 1 。
- 从 S 到入点连边，从出点到 T 连边，费用为 0 。
- 只需要求 $S \to T$ 的容量为 4 的费用流即可。
- 边数好像是 $O(n^2)$ 的，复杂度 $O(n^3)$ 了。

例题 2

- 一种常见的建图技巧：对于“每个点只能选一次”的限制，建立入点和出点，从入点向出点连容量为 1 的边，费用为 0 。
- 对于 $i \to j$ 可达，连 $out_i \to in_j$ 的边，费用为 1 。
- 从 S 到入点连边，从出点到 T 连边，费用为 0 。
- 只需要求 $S \to T$ 的容量为 4 的费用流即可。
- 边数好像是 $O(n^2)$ 的，复杂度 $O(n^3)$ 了。
- 对每个点建两个虚点，一个表示从模 7 同余的点过来，另一个表示从值相同的点的过来。然后对每个点就只需要从出点向下一个模 7 为 0 的点和下一个值为 $v \pm 1$ 的对应虚点连边了。
- 点边数均为 $O(n)$ ，直接跑复杂度就是 $O(n^2)$ 。

例题 3

- 给定一张有向图，点有点权，在这张图上选出选出权值和最大的一个子图，满足每个点的后继都在子图中。
- $n \leq 150, m \leq 1500$ 。

例题 3

- 定义和 S 在一边的点表示不选，和 T 在一边的点表示选。
- 从 S 连向正权点，容量为权值。从负权点连向 T，容量为权值相反数。
- 答案是正权点权和减去最大流。
- 负权点被割表示它被选了，需要减去。正权点被割了表示它不选，也是从正权和中减去。

例题 4

- 给出一张 $n \times m$ 的网格图，每条边的两个方向人流量给出，每个格点都有一个海拔，一个人沿着某一方向通过一条边的代价是终点海拔和起点海拔的差的绝对值。左下角海拔为 0，右上角海拔为 1，求安排剩下点海拔最小化总代价。
- $n, m \leq 500$

例题 4

- 显然可以通过调整让一个最优解为所有点的海拔都是 0 或 1 。
- 很容易建立最小割模型，与 S 相连的点海拔为 0 ，与 T 相连的点海拔为 1 ，最小割就是最小的代价。
- 可以发现平面图的边数和点数都是 $O(nm)$ 级别的没办法通过。

例题 4

- 显然可以通过调整让一个最优解为所有点的海拔都是 0 或 1 。
- 很容易建立最小割模型，与 S 相连的点海拔为 0 ，与 T 相连的点海拔为 1 ，最小割就是最小的代价。
- 可以发现平面图的边数和点数都是 $O(nm)$ 级别的没办法通过。
- 网格图（平面图）的最小割可以通过对偶转成最短路，将格子看成点，边权为格子间的人流，就可以变成左上到右下的最短路。
- 直接使用 dijkstra 求最短路即可。

例题 5

- 给出一个长为 n 的未知数组 A，每个元素在 $[1,n]$，给出 q 个限制，每个形如：
  - $[l,r]$ 所有元素大于等于 v。
  - $[l,r]$ 所有元素小于等于 v。
- 设 $cnt(i)$ 是 i 在其中出现次数，求这个值的最小值：

$$\sum_{i=1}^{n} cnt(i)^2$$

- $1 \le n \le 50, 0 \le q \le 100$

例题 5

- 把区间对应到每个位置上，每个位置相当于有个可以放的值域。
- 首先对每个值一个点，然后从这个点向汇点连费用为 $1^2, 2^2-1^2, 3^2-2^2, \ldots$，容量均为 1 的边。
- 对于每个位置，假设能选的值在 $[L, R]$，$L, L+1, \ldots, R$ 的点，$i \to i+1$ 有容量为 1 费用为 0 的边，从 i 到值为 i 的点也有容量为 1 费用为 0 的边。
- 最后从 S 到 L 连一条边，求最小费用最大流即可。

例题 6

- 给出一张无向图，边权均为 1 。请你在图中添加若干条边，最小化：

$$\sum_{i=1}^{n}(A[i]-dis[i])^2$$

- 其中 $A[i]$ 给定，$dis[i]$ 是 1 到 i 的最短路。
- $T \leq 20, n \leq 40, m \leq 1600, A_i \leq 1000$

例题 6

- dis 数组可以得到的充要条件是；
  - 有且仅有 1 的 dis 是 0 。
  - 对于任意无向边 (u, v) 保证 $|dis[u] - dis[v]| \leq 1$ 。
  - 对于 $1 \sim \max dis$ 的任意整数都有点取到。
- 对每个点拉一条 $S \to 1 \to 2 \to 3 \to \cdots \to n-1 \to T$ 的链，用“割开其中一条边”表示让这个点的权为这个值，代价为 $(A_i - x)^2$ 。
- 对于 $|dis[u] - dis[v]| \leq 1$ 的限制，可以从 $(u, k) \to (v, k-1), (v, k) \to (u, k-1)$ 连一条正无穷边，这样就可以保证两个值的差不超过 1 。

例题 7

- 给出一个长为 n 的序列，一次操作可以在序列选出两个数，分数是前一个数减后一个数的差，然后删掉它们。
- 对于每个 k 求出执行 k 次操作的最大分数。
- $n \leq 5 \times 10^5$

例题 7

- 费用流模型非常简单，源点向每个点连费用为权值的边，容量为 1，每个点向汇点连费用为负权值的边，容量为 1，然后 $i \to i+1$ 连费用为 0，容量无限的边。

例题 7

- 费用流模型非常简单，源点向每个点连费用为权值的边，容量为 1，每个点向汇点连费用为负权值的边，容量为 1，然后 $i \to i+1$ 连费用为 0，容量无限的边。
- 直接跑费用流肯定是超时。但是可以发现每次操作都是选择 i, j 满足 $A_i - A_j$ 尽量大，并且 $i \to j$ 的流量都大于 1，然后给流量加一或者减一。

例题 7

- 费用流模型非常简单，源点向每个点连费用为权值的边，容量为 1，每个点向汇点连费用为负权值的边，容量为 1，然后 $i \to i+1$ 连费用为 0，容量无限的边。
- 直接跑费用流肯定是超时。但是可以发现每次操作都是选择 i, j 满足 $A_i - A_j$ 尽量大，并且 $i \to j$ 的流量都大于 1，然后给流量加一或者减一。
- 每个点只能被选择一次，所以可以把流量放在点上，用线段树维护区间内的点的最小流量以及如果把最小流量看成 0 时这个区间的答案、与区间左端点连通的部分的最大值最小值和与区间右端点连通的部分的最大值和最小值以及整个区间的最大值和最小值。
- 如果整体最小值大于 0，则可以把区间最大值到最小值的两个数的结果转移答案。这样维护标记是可以下放的。

例题 8

- 给出两个长为 n 的序列 A, B，在两个序列分别选择恰好 k 个下标，满足至少有 L 个下标在两个序列相同，并且这些位置的和最大。
- $n \leq 10^6$

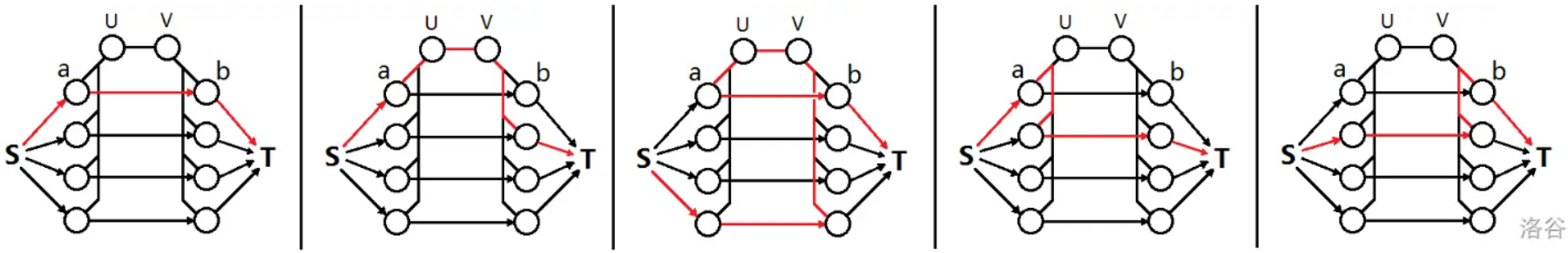
例题 8

- 建图就是 $S \to L_i$ 连 $(1, A_i)$ ，$L_i \to R_i$ 连 $(1, 0)$ ，$R_i \to T$ 连 $(1, B_i)$ ，走这条路就是选择两边相同的下标。
- 然后由于可以选 $k - L$ 对不同下标，对于左边所有点向 A_L 连 $(1, 0)$ ，从 A_L 向 A_R 连 $(k - L, 0)$ ，从 A_R 所有右部点连 $(1, 0)$ 。

例题 8

- 建图就是 $S \to L_i$ 连 $(1, A_i)$，$L_i \to R_i$ 连 $(1, 0)$，$R_i \to T$ 连 $(1, B_i)$，走这条路就是选择两边相同的下标。
- 然后由于可以选 $k - L$ 对不同下标，对于左边所有点向 A_L 连 $(1, 0)$，从 A_L 向 A_R 连 $(k - L, 0)$，从 A_R 所有右部点连 $(1, 0)$。
- 首先，即使出现退流也不可能经过 S, T 两次，否则说明之前走了非常浪费的流。
- 所以当一个点被走过一次后，它不会再被 S, T 直接经历了，因此增广路的形态只有五种。

例题 8



- 维护两边未选数的最大值、未选数的同标号和最大值、本侧未选另一个侧选了的最大值。
- 物种情况都可以拆成维护的东西的加和，取最大值更新即可。

例题 9

- 给出一个长为 n 的序列 A，对于每个 k，在序列中选出一个长为 k 的子序列，最大化 $\sum_i (-1)^i v_i$ 。
- $n \leq 5 \times 10^5$

例题 9

- 其实这题不是流。

例题 9

- 其实这题不是流。
- 如果 k 是偶数，就是一个选出 $\frac{k}{2}$ 个区间，满足 $A_l - A_r$ 和最大的问题，这个和前面的例题很像，可以建出费用流模型。而 k 是奇数也可以类似地建费用流模型，钦定流一下第一个点即可。

例题 9

- 其实这题不是流。
- 如果 k 是偶数，就是一个选出 $\frac{k}{2}$ 个区间，满足 $A_l - A_r$ 和最大的问题，这个和前面的例题很像，可以建出费用流模型。而 k 是奇数也可以类似地建费用流模型，钦定流一下第一个点即可。
- 因此 k 为偶数的情况和 k 为奇数的情况分别关于 k 是凸的。
- 暴力 DP 可以设 $f[i][j]$ 表示 i 个数选了 j 个数的最大贡献，直接转移。但是由于是凸的，可以把状态中钦定区间前面选的数的个数的奇偶性质以及区间内的个数的奇偶性，将左右两边通过 $\max, +$ 卷积合并起来即可。

例题 10

- 给定一个序列 A ，给出一些偏序关系 (u, v) 表示必须满足 $B_u \leq B_v$ ，求出一个 B 序列最小化：

$$\sum_{i=1}^{n} w_i (B_i - A_i)^2$$

- $n \leq 150$

例题 10

- 假设想要给所有数确定区间 $[l, r]$ 里的权值，那么我们给所有数确定权值为 $m, m+1$ 中的一个数并使得权值和最小，可以证明此时取 m 的数最终会取 $[l, m]$ ，此时取 $m+1$ 的数最终会取 $[m+1, r]$ 。
- 所以如果能确定每个点在 $[m, m+1]$ 里面的分类，就可以把两部分的点分别作为子问题继续做，也就是说得到了一个整体二分的做法。

例题 10

- 假设想要给所有数确定区间 $[l, r]$ 里的权值，那么我们给所有数确定权值为 $m, m+1$ 中的一个数并使得权值和最小，可以证明此时取 m 的数最终会取 $[l, m]$ ，此时取 $m+1$ 的数最终会取 $[m+1, r]$ 。
- 所以如果能确定每个点在 $[m, m+1]$ 里面的分类，就可以把两部分的点分别作为子问题继续做，也就是说得到了一个整体二分的做法。
- 确定 $[m, m+1]$ 是很简单的，可以发现其实就是 DAG 上的最大权闭合子图问题，直接建图最大流即可。
- 如果想要更深刻的理解可以参考 2018 国家集训队论文《浅谈保序回归问题》(高睿泉)。

例题 Ex

- 给出一个 n 个点 m 条边的有向图，边有边权，保证 1 能到所有点。
- 对于每个点 u ，求出 $1 \to u$ 的两条无公共边路径，满足边权和最小。
- $n \leq 10^5, m \leq 3 \times 10^5$

例题 Ex

- 其实这个题放在这里不太合适（?）但是我很喜欢这个题。
- $O(n^2)$ 的暴力是很简单的。选出两条边不相交路径可以直接看成一个网络流问题，也就是求出 1 到点 u 的大小为 2 的最小费用流，且每条边容量为 1，跑两次单路增广即可。

例题 Ex

- 其实这个题放在这里不太合适（?）但是我很喜欢这个题。
- $O(n^2)$ 的暴力是很简单的。选出两条边不相交路径可以直接看成一个网络流问题，也就是求出 1 到点 u 的大小为 2 的最小费用流，且每条边容量为 1 ，跑两次单路增广即可。
- 首先第一次流其实可以非常简单的得出，也就是当我们建出最短路树后，第一次流可以从 1 直接在最短路树上流到 u 。现在的问题是在流完一次之后路径上的边都是负权边。

例题 Ex

- 其实这个题放在这里不太合适（?）但是我很喜欢这个题。
- $O(n^2)$ 的暴力是很简单的。选出两条边不相交路径可以直接看成一个网络流问题，也就是求出 1 到点 u 的大小为 2 的最小费用流，且每条边容量为 1 ，跑两次单路增广即可。
- 首先第一次流其实可以非常简单的得出，也就是当我们建出最短路树后，第一次流可以从 1 直接在最短路树上流到 u 。现在的问题是在流完一次之后路径上的边都是负权边。
- 可以在一开始做一次 Primal Dual ，也就是让每个边 (u, v, w) 的权值变成 $w - d_v + d_u$ 。如此一来，最短路树上的边权值都变成了 0 ，同时第一条流增广结束后仅仅是将边反向，所有边长度仍然是大于等于 0 的。

例题 Ex

- 现在我们需要求出 d'_u 表示 $1 \to u$ 被反向时 $1 \to u$ 的最短路长度。
- 由于所有边的权值仍然是非负的，同样可以用 dijkstra 的方式算出每个点的 d'_u 。我们假设当前堆顶为 u ，考虑 d'_u 能更新哪些点的 d' 。

例题 Ex

- 现在我们需要求出 d'_u 表示 $1 \to u$ 被反向时 $1 \to u$ 的最短路长度。
- 由于所有边的权值仍然是非负的，同样可以用 dijkstra 的方式算出每个点的 d'_u 。我们假设当前堆顶为 u ，考虑 d'_u 能更新哪些点的 d' 。
- 对于一条边 (a, b, w) ，能用 $d'_u + w$ 松弛 b 当且仅当 $a = u$ 或 a, b 不在 u 为根的一个子树。

例题 Ex

- 现在我们需要求出 d'_u 表示 $1 \to u$ 被反向时 $1 \to u$ 的最短路长度。
- 由于所有边的权值仍然是非负的，同样可以用 dijkstra 的方式算出每个点的 d'_u 。我们假设当前堆顶为 u ，考虑 d'_u 能更新哪些点的 d' 。
- 对于一条边 (a, b, w) ，能用 $d'_u + w$ 松弛 b 当且仅当 $a = u$ 或 a, b 不在 u 为根的一个子树。
- 这是因为在原图上把 $1 \to b$ 反向等价于在当前这个图上把 $u \to b$ 反向。所以如果 a, v 不再同一个子树，那么存在 $u \to a$ 的权值为 0 的路径。
- 第一种边的松弛是非常简单的，直接做即可。问题是第二种边的数量可能非常多。

例题 Ex

- 可以发现如果 u 的一个子树内存在一个点 v 使得 $d_u < d_v$ ，那么 v 的子树就已经被更新过了，所以更新完一个点可以删除掉这个点并且把树分成几块，这是一个类似点分的过程。
- 但是这种启发式点分也很容易被卡到 $O(n^2)$ 。

例题 Ex

- 可以发现如果 u 的一个子树内存在一个点 v 使得 $d_u < d_v$ ，那么 v 的子树就已经被更新过了，所以更新完一个点可以删除掉这个点并且把树分成几块，这是一个类似点分的过程。
- 但是这种启发式点分也很容易被卡到 $O(n^2)$ 。
- 可以对每个点都保存入边和出边，然后每次只枚举非最大的子树并松弛入边和出边。很容易分析出每个点被枚举的次数是 $O(\log n)$ 级别的。

例题 Ex

- 可以发现如果 u 的一个子树内存在一个点 v 使得 $d_u < d_v$ ，那么 v 的子树就已经被更新过了，所以更新完一个点可以删除掉这个点并且把树分成几块，这是一个类似点分的过程。
- 但是这种启发式点分也很容易被卡到 $O(n^2)$ 。
- 可以对每个点都保存入边和出边，然后每次只枚举非最大的子树并松弛入边和出边。很容易分析出每个点被枚举的次数是 $O(\log n)$ 级别的。
- 最后一个问题是怎么判断哪个子树最大。考虑维护当前最大的子树 t ，可以通过倍增 DFS 次数上界的方式 $O(\min(siz_t, siz_v))$ 的时间比较两个子树的大小，这样就可以在轻子树大小和内的时间找出最大子树。
- 复杂度 $O(m \log n)$ 。