最小割

概念

割

对于一个网络流图 $G=(V,E)$ 其割的定义为一种 **点的划分方式**：将所有的点划分为 S 和 $T=V-S$ 两个集合，其中源点 $s\in S$ 汇点 $t\in T$

割的容量

我们定义割 (S,T) 的容量 $c(S,T)$ 表示所有从 S 到 T 的边的容量之和，即 $c(S,T)=\sum_{u\in S,v\in T}c(u,v)$ 当然我们也可以用 $c(s,t)$ 表示 $c(S,T)$

最小割

最小割就是求得一个割 (S,T) 使得割的容量 $c(S,T)$ 最小。

证明

最大流最小割定理

定理 $f(s,t)_{max}=c(s,t)_{min}$

对于任意一个可行流 $f(s,t)$ 的割 (S,T) 我们可以得到：

$f(s,t)=S_{出边的总流量}-S_{入边的总流量}\le S_{出边的总流量}=c(s,t)$

如果我们求出了最大流 f 那么残余网络中一定不存在 s 到 t 的增广路径，也就是 S 的出边一定是满流 S 的入边一定是零流，于是有：

$f(s,t)=S_{出边的总流量}-S_{入边的总流量}=S_{出边的总流量}=c(s,t)$

结合前面的不等式，我们可以知道此时 f 已经达到最大。

代码

最小割

通过 **最大流最小割定理**，我们可以直接得到如下代码：

参考代码：

```
#include <algorithm>
#include <cstdio>
#include <cstring>
#include <queue>

const int N = 1e4 + 5, M = 2e5 + 5;
int n, m, s, t, tot = 1, lnk[N], ter[M], nxt[M], val[M], dep[N], cur[N];

void add(int u, int v, int w) {
  ter[++tot] = v, nxt[tot] = lnk[u], lnk[u] = tot, val[tot] = w;
}
void addedge(int u, int v, int w) { add(u, v, w), add(v, u, 0); }
int bfs(int s, int t) {
  memset(dep, 0, sizeof(dep));
  memcpy(cur, lnk, sizeof(lnk));
  std::queue<int> q;
  q.push(s), dep[s] = 1;
  while (!q.empty()) {
    int u = q.front();
    q.pop();
    for (int i = lnk[u]; i; i = nxt[i]) {
      int v = ter[i];
      if (val[i] && !dep[v]) q.push(v), dep[v] = dep[u] + 1;
    }
  }
  return dep[t];
}
int dfs(int u, int t, int flow) {
  if (u == t) return flow;
  int ans = 0;
  for (int &i = cur[u]; i && ans < flow; i = nxt[i]) {
    int v = ter[i];
    if (val[i] && dep[v] == dep[u] + 1) {
      int x = dfs(v, t, std::min(val[i], flow - ans));
      if (x) val[i] -= x, val[i ^ 1] += x, ans += x;
    }
  }
  if (ans < flow) dep[u] = -1;
  return ans;
}
int dinic(int s, int t) {
  int ans = 0;
  while (bfs(s, t)) {
    int x;
    while ((x = dfs(s, t, 1 << 30))) ans += x;
  }
  return ans;
}
int main() {
```

```
  scanf("%d%d%d%d", &n, &m, &s, &t);
  while (m--) {
    int u, v, w;
    scanf("%d%d%d", &u, &v, &w);
    addedge(u, v, w);
  }
  printf("%d\n", dinic(s, t));
  return 0;
}
```

方案

我们可以通过从源点 s 开始 DFS 每次走残量大于 0 的边，找到所有 S 点集内的点。

```
void dfs(int u) {
  vis[u] = 1;
  for (int i = lnk[u]; i; i = nxt[i]) {
    int v = ter[i];
    if (!vis[v] && val[i]) dfs(v);
  }
}
```

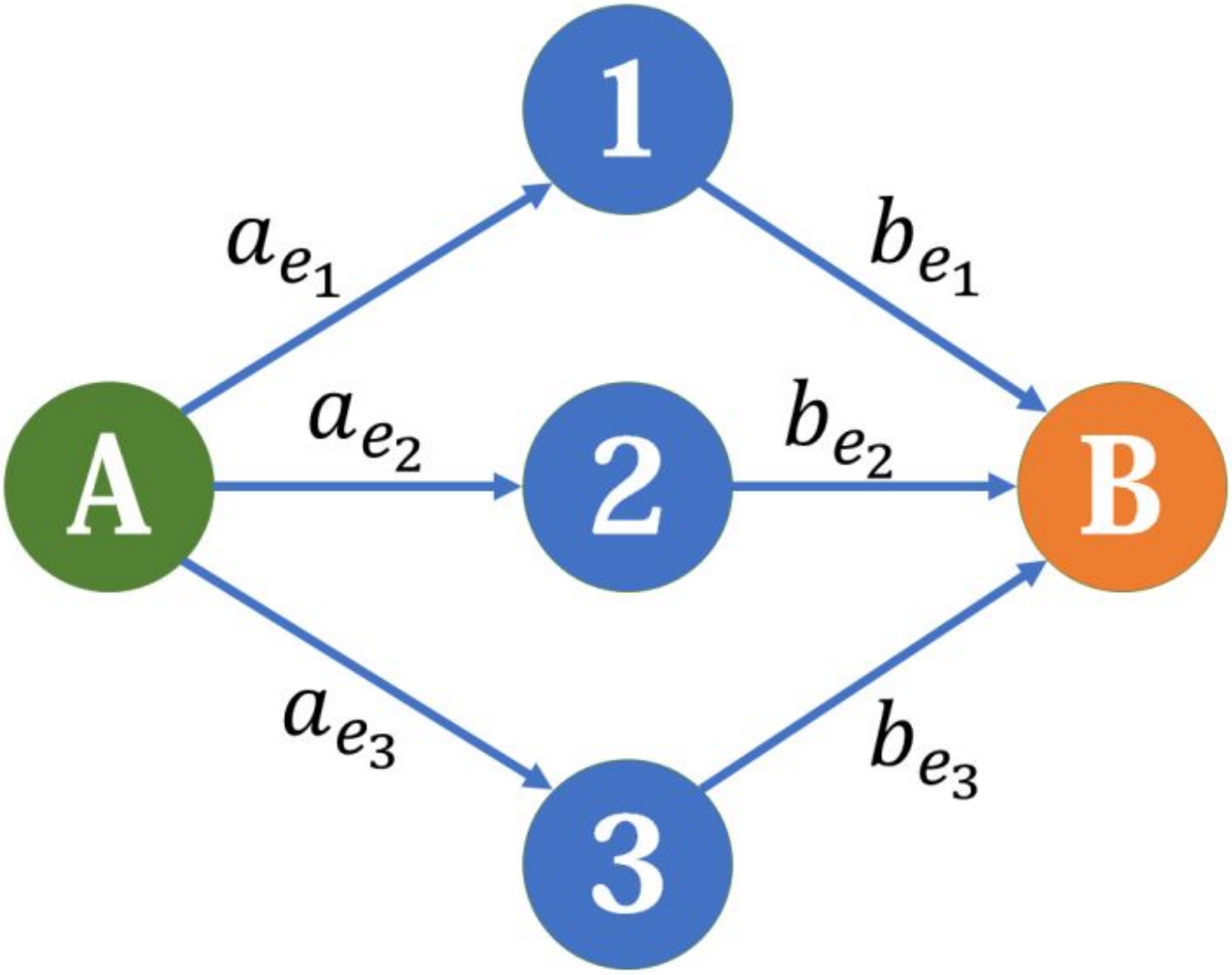
割边数量

只需要将每条边的容量变为 1，然后重新跑 Dinic 即可。

问题模型——二者取一式问题

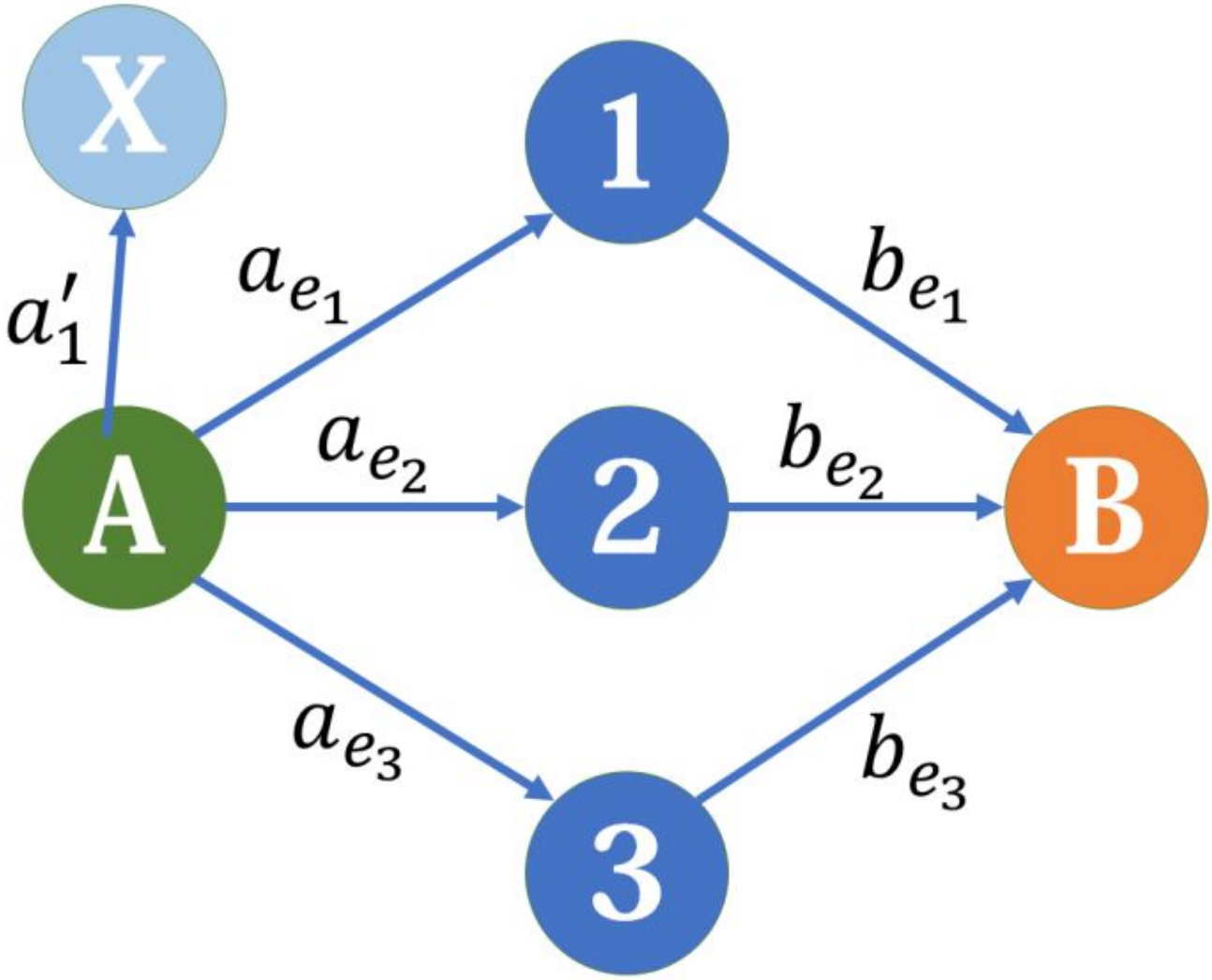
二者取一式问题可以这样描述：将若干元素 $e_1,e_2,e_3,\cdots,e_n$ 划分到两个集合 A,B 中。对于元素 e_i 它被划分到 A 或 B 中分别能获得一个 a_{e_i} 或 b_{e_i} 的分值。除此之外，还给出若干个组合 $C_i\subseteq A$ 当组合中的元素被同时划分到 A 或 B 时，可以获得额外的分值 a_i' 或 b_i' 求最大的分值。

这个问题可以被转化为网络流中的 **最小割**问题。如果我们把 A 作为源点 B 作为汇点，那么这个网络的一个 **割** 就是一种划分方法。如果没有组合的话，我们很容易就能建出这样的模型：



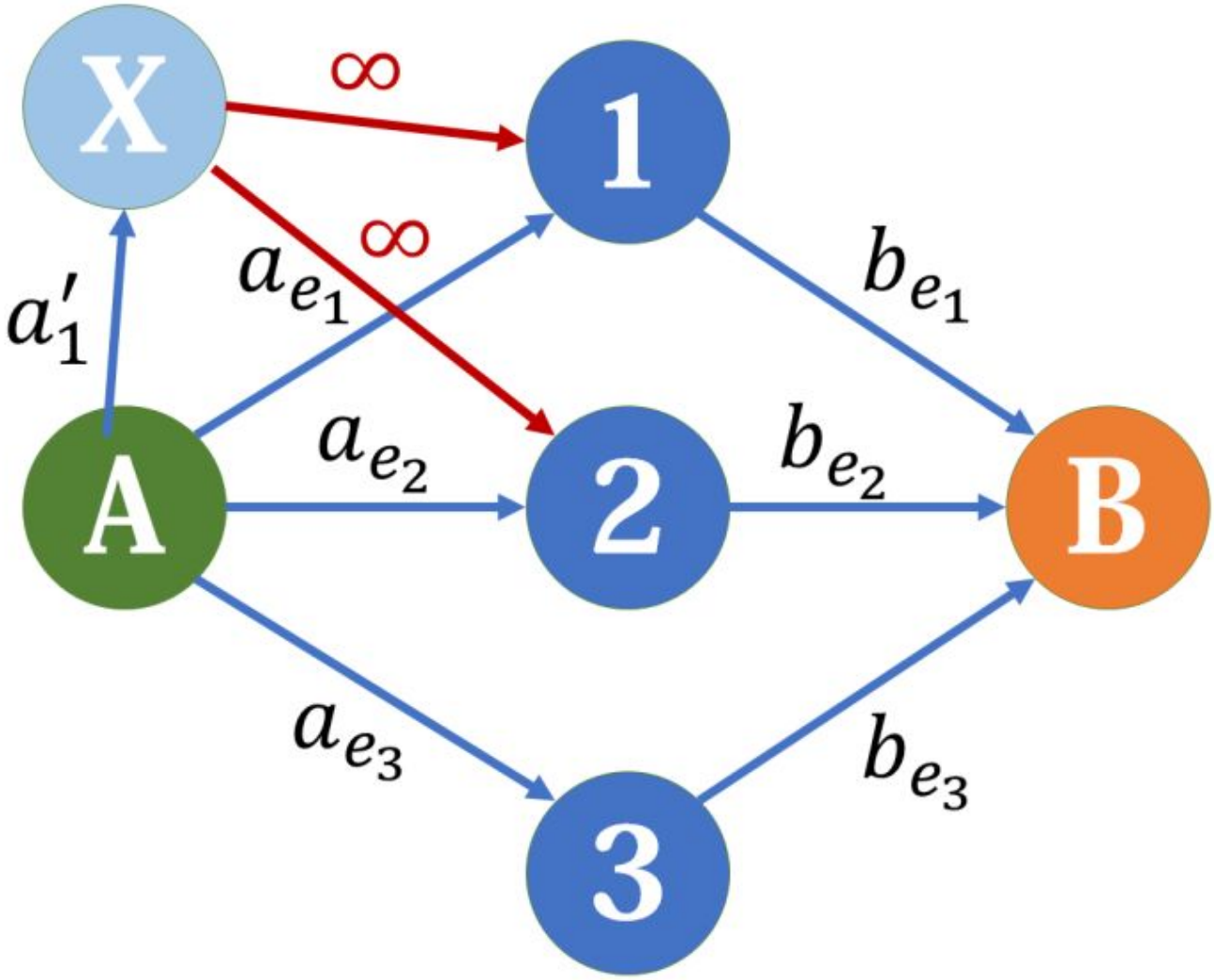
当我们去割它时，与 A 连通的点代表放到 A 集合中，与 B 连通的点代表放到 B 集合中。当这个割是最小割时，剩下的边的容量和是最大的，故设最小割为 cut，边权总和为 sum，则所求最大分值为 sum-cut。

现在我们考虑组合。假设 $C_1=\{e_1,e_2\}$ 且对应的额外分值为 a_1' 和 b_1' 我们从 A 点伸出一条容量为 a_1' 的边通向虚点 X。

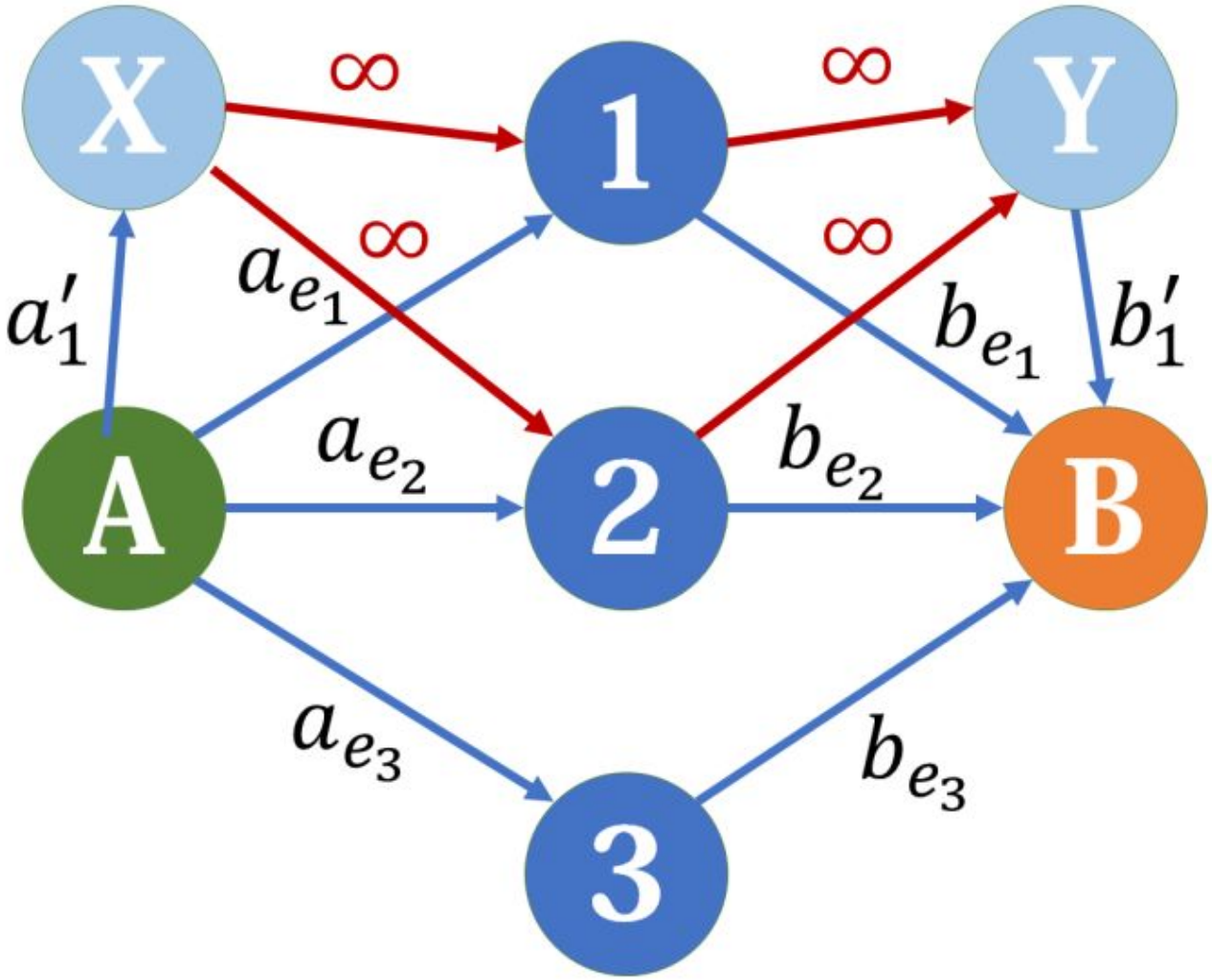


现在我们的需求是：只有当 1、2 点都被归入 A 所在点集时，X 才与 A 连通。

反过来想，当 1 被归入 B 所在点集时，要让 $A\rightarrow X$ 被割掉。很自然地想到，让 X 连向 1，这样当 1 被归入 B 所在点集时，$A\rightarrow X\rightarrow 1$ 必然会断，否则 A 就与 B 连通了。但如何确保割掉的是 $A\rightarrow X$ 而不是 $X\rightarrow 1$ 呢？只要令 $X\rightarrow 1$ 的容量为 INF 即可，无穷大边不会被割掉。2 号点同理。



对于 B 号点，道理是一样的：



好了，这就是我们需要的模型。这时我们求最小割 `cut`，并记非无穷边权和为 `sum`，那么跟刚刚一样，`sum-cut` 就是所求分数。

习题

[USACO 4.4]Pollutant Control

[USACO 5.4]Telecowmunication

[Luogu 1361]小 M 的作物

[SHOI 2007]善意的投票

参考链接

OI Wiki

算法学习笔记(29): 二者取一式问题