网络流算法和建模

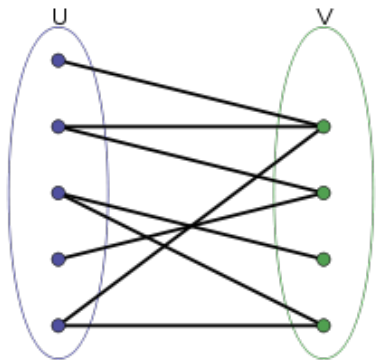
hzwer,miskcoo

北京大学, 清华大学

2016 年 12 月 23 日

二分图

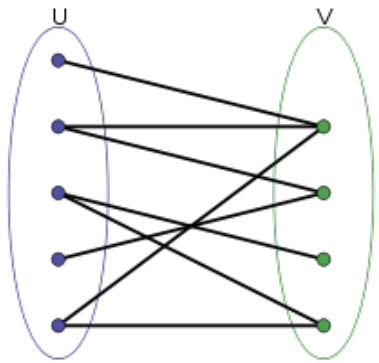
在图论中，**二分图**是由两个节点集组成的，这两个点集不相交，我们可以称为左部和右部。点集内部节点没有互相连边，边只在点集之间相连。



二分图

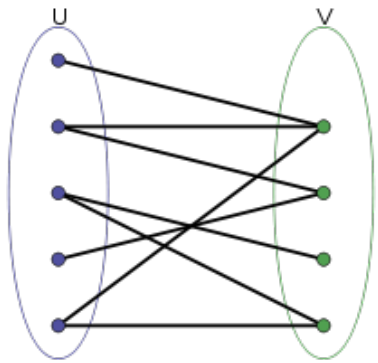
在图论中，**二分图**是由两个节点集组成的，这两个点集不相交，我们可以称为左部和右部。点集内部节点没有互相连边，边只在点集之间相连。

怎样的图才是二分图呢？



二分图

在图论中，**二分图**是由两个节点集组成的，这两个点集不相交，我们可以称为左部和右部。点集内部节点没有互相连边，边只在点集之间相连。



怎样的图才是二分图呢？

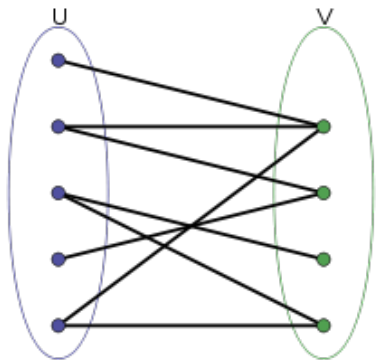
从任意一个点开始进行搜索，第一个节点标 1

与 1 相连的节点标 0，与 0 相连的节点标 1

如果不产生矛盾，那么将所有标号为 1 的节点放左部，标号为 0 的放右部。

二分图

在图论中，**二分图**是由两个节点集组成的，这两个点集不相交，我们可以称为左部和右部。点集内部节点没有互相连边，边只在点集之间相连。



怎样的图才是二分图呢？

从任意一个点开始进行搜索，第一个节点标 1

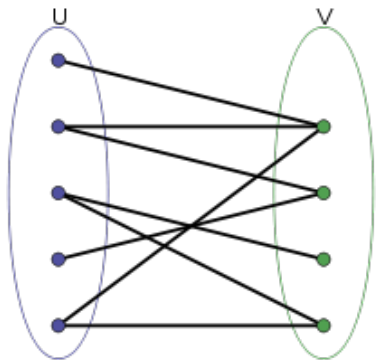
与 1 相连的节点标 0，与 0 相连的节点标 1

如果不产生矛盾，那么将所有标号为 1 的节点放左部，标号为 0 的放右部。

什么时候会有矛盾？

二分图

在图论中，**二分图**是由两个节点集组成的，这两个点集不相交，我们可以称为左部和右部。点集内部节点没有互相连边，边只在点集之间相连。



怎样的图才是二分图呢？

从任意一个点开始进行搜索，第一个节点标 1

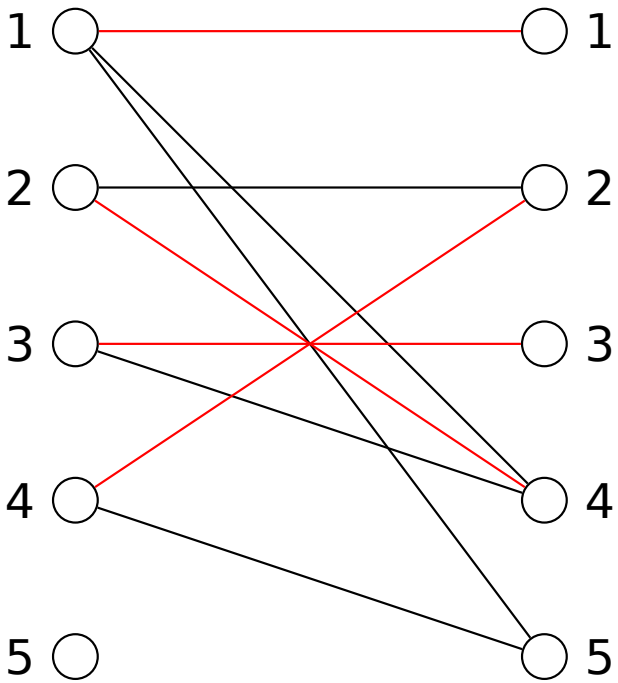
与 1 相连的节点标 0，与 0 相连的节点标 1

如果不产生矛盾，那么将所有标号为 1 的节点放左部，标号为 0 的放右部。

什么时候会有矛盾？

存在节点个数为奇数的环。

二分图匹配



二分图的**匹配**是一些边，要求满足每个节点最多只被这些边里面的一条边覆盖。

所有匹配中，边数最多的匹配被称为**二分图的最大匹配**。

例如，左图中就是一个最大匹配。

匈牙利算法

匈牙利算法是用来计算二分图最大匹配的一种算法。

匈牙利算法

匈牙利算法是用来计算二分图最大匹配的一种算法。

先来介绍一个**交错路**的概念。

匈牙利算法

匈牙利算法是用来计算二分图最大匹配的一种算法。

先来介绍一个**交错路**的概念。

如果从一个还没有被边覆盖的节点出发，不断经过未被匹配的边，匹配边，未被匹配的边……直到最后经过一条未被匹配的边到达一个节点。

这样走过的路径就被称为交错路。如果最后一个结点是未被匹配的节点，那么这条路径叫做**增广路**。

匈牙利算法

匈牙利算法是用来计算二分图最大匹配的一种算法。

先来介绍一个**交错路**的概念。

如果从一个还没有被边覆盖的节点出发，不断经过未被匹配的边，匹配边，未被匹配的边……直到最后经过一条未被匹配的边到达一个节点。

这样走过的路径就被称为交错路。如果最后一个结点是未被匹配的节点，那么这条路径叫做**增广路**。

如果我们将增广路上的匹配边和未匹配边交换一下，这样匹配边的数目就会增加 1 条。

匈牙利算法就是不断寻找增广路来增广的。

匈牙利算法

我们可以直接从左部按顺序对每个结点寻找一次增广路。

miskcoo@bw:~

```
30
31      bool augment(int u)
32      {
33          for(int k = head[u]; k; k = next[k])
34          {
35              int v = point[k];
36              if(mark[v] == mark_count) continue;
37              mark[v] = mark_count;
38              if(mat[v] == 0 || augment(mat[v]))
39              {
40                  mat[v] = u;
41                  return true;
42              }
43          }
44          return false;
45      }
46
```

匈牙利算法

miskcoo@bw:~

```
46
47      int match()
48      {
49          mark_count = 0;
50          int count = 0;
51          std::memset(mat, 0, sizeof(mat));
52          std::memset(mark, 0, sizeof(mark));
53          for(int u = 1; u <= vertex_size; ++u)
54          {
55              ++mark_count;
56              if(augment(u))
57                  ++count;
58          }
59          return count;
60      }
61
```

点独立集和点覆盖集

点覆盖集是无向图的一个点集，使得该图中的所有边至少有一个端点在该集合内。点数最少的点覆盖集被称为**最小点覆盖集**。

点独立集是无向图的一个点集，使得该集合中任意两个点之间不连通。点数最多的点被称为**最大点独立集**。

点独立集和点覆盖集

点覆盖集是无向图的一个点集，使得该图中的所有边至少有一个端点在该集合内。点数最少的点覆盖集被称为**最小点覆盖集**。

点独立集是无向图的一个点集，使得该集合中任意两个点之间不连通。点数最多的点被称为**最大点独立集**。

Theorem

一个二分图的最小的覆盖集大小等于最大点独立集大小。

例题. 分食物

有 n 对情侣坐成一个圈，有两种食物，要给每个人分其中一种，要求每对情侣的食物不同，任意连续的三个人必须要有两人食物不同。

例题. 分食物

有 n 对情侣坐成一个圈，有两种食物，要给每个人分其中一种，要求每对情侣的食物不同，任意连续的三个人必须要有两人食物不同。

求分配方案，无解输出-1

例题. 分食物

有 n 对情侣坐成一个圈，有两种食物，要给每个人分其中一种，要求每对情侣的食物不同，任意连续的三个人必须要有两人食物不同。

求分配方案，无解输出-1

$1 \leq n \leq 100000$

例题. 分食物

改变限制，要求 $2i$ 和 $2i-1$ 食物类型不同

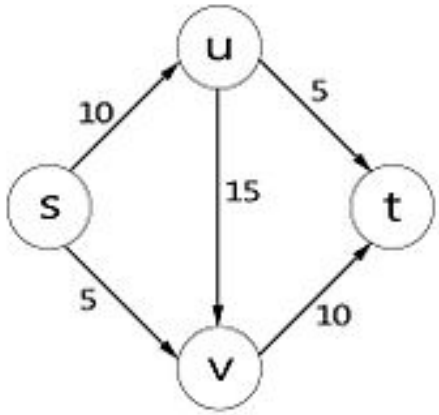
例题. 分食物

改变限制，要求 $2i$ 和 $2i-1$ 食物类型不同

二分图染色

运输网络

先来看一个运输网络的例子。



假设上图是有一个运输货物的网络，节点表示中转站，边表示运输路线。每条路线都有一定的最大运输能力。

现在你需要将货物从 S 运输到 T，并且货物不能在中途堆积。

运输网络

我们现在来看看一个合理的运输方案应该满足什么？

运输网络

我们现在来看看一个合理的运输方案应该满足什么？

- 每条路线上的运输量不能是负的。

运输网络

我们现在来看看一个合理的运输方案应该满足什么？

- 每条路线上的运输量不能是负的。
- 每条路线上的运输量不能超过它的最大运输能力。

运输网络

我们现在来看看一个合理的运输方案应该满足什么？

- 每条路线上的运输量不能是负的。
- 每条路线上的运输量不能超过它的最大运输能力。
- 每个中转站中运入的货物量应该等于运出的货物辆。

运输网络

我们现在来看看一个合理的运输方案应该满足什么？

- 每条路线上的运输量不能是负的。
- 每条路线上的运输量不能超过它的最大运输能力。
- 每个中转站中运入的货物量应该等于运出的货物辆。

现在来考虑一个问题：从 S 到 T 的最大运输量是多少？

网络流模型

事实上，这就是一个经典的网络流模型。现在先来介绍一下网络流的概念。

网络流模型

事实上，这就是一个经典的网络流模型。现在先来介绍一下网络流的概念。

容量网络是一个有向图，图的边 (u, v) 有非负的权 $c(u, v)$，被称为**容量**。图中有一个被称为**源**的节点和一个被称为**汇**的节点。

实际通过每条边的流量记为 $f(u, v)$。

残量网络是一个结构和容量网络相同的有向图，只不过边的权值为 $c(u, v) - f(u, v)$。

网络流模型

事实上，这就是一个经典的网络流模型。现在先来介绍一下网络流的概念。

容量网络是一个有向图，图的边 (u, v) 有非负的权 $c(u, v)$，被称为**容量**。图中有一个被称为**源**的节点和一个被称为**汇**的节点。

实际通过每条边的流量记为 $f(u, v)$。

残量网络是一个结构和容量网络相同的有向图，只不过边的权值为 $c(u, v) - f(u, v)$。

所有边上的流量集合被称为**网络流**。

像刚刚的运输网络就是一个容量网络。

网络流模型

在容量网络中，满足以下条件的网络流被称为**可行流**。

网络流模型

在容量网络中，满足以下条件的网络流被称为**可行流**。

- **容量限制**：

$$0 \leq f(u, v) \leq c(u, v) \tag{1}$$

网络流模型

在容量网络中，满足以下条件的网络流被称为**可行流**。

- **容量限制**：

$$0 \leq f(u,v) \leq c(u,v) \tag{1}$$

- **流量守恒**：对于非源汇节点 u，满足

$$\sum_{(u,v)\in E} f(u,v) = \sum_{(v,u)\in E} f(v,u) \tag{2}$$

网络流模型

在容量网络中，满足以下条件的网络流被称为**可行流**。

- **容量限制**：

$$0 \leq f(u,v) \leq c(u,v) \tag{1}$$

- **流量守恒**：对于非源汇节点 u，满足

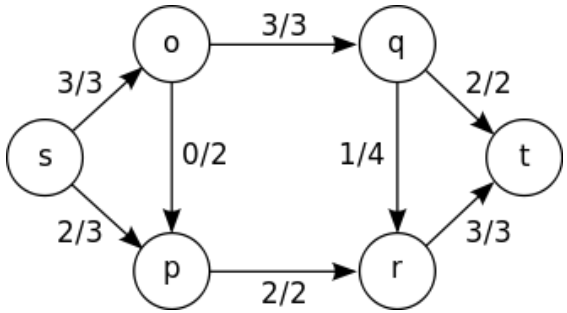
$$\sum_{(u,v)\in E} f(u,v) = \sum_{(v,u)\in E} f(v,u) \tag{2}$$

为了方便，我们还定义一个性质

- **反对称性**：

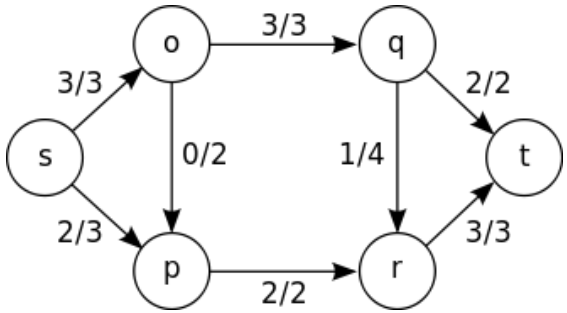
$$f(u,v) = -f(v,u) \tag{3}$$

网络流模型



像上面这个网络就是一个可行流，边上的标号表示流量和容量。

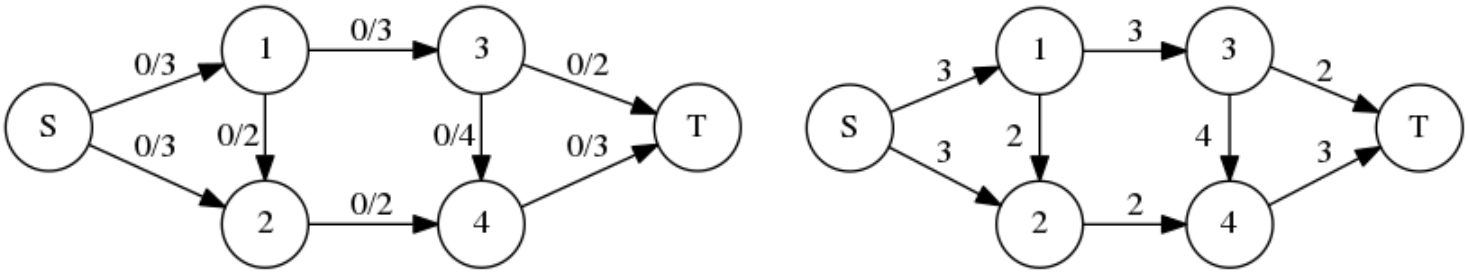
网络流模型



像上面这个网络就是一个可行流，边上的标号表示流量和容量。

我们称有最大流量的网络流为**最大流**。上面这个网络同时也是一个最大流。

增广路



如果我们在容量网络（左图）网络的残量网络（右图）上找出一条从 S 到 T 的路径，并且路径上边权最小的那条边权值为正，那么我们就说找到了一条**增广路**。

Ford–Fulkerson 算法

最大流的 Ford–Fulkerson 算法基本思想就是**增广**。

每次增广先在残量网络上找到一条增广路，然后将这条路上每条边的边权减去增广路上边权最小边的边权。再在该边的反向边上加上这个权值。

Ford–Fulkerson 算法

最大流的 Ford–Fulkerson 算法基本思想就是**增广**。

每次增广先在残量网络上找到一条增广路，然后将这条路上每条边的边权减去增广路上边权最小边的边权。再在该边的反向边上加上这个权值。

为什么要在反向边加上权值呢？

Ford–Fulkerson 算法

最大流的 Ford–Fulkerson 算法基本思想就是**增广**。

每次增广先在残量网络上找到一条增广路，然后将这条路上每条边的边权减去增广路上边权最小边的边权。再在该边的反向边上加上这个权值。

为什么要在反向边加上权值呢？

为了可以撤销这一次的增广操作，因为网络流并不能够用过贪心解决，这次的增广不一定是全局最优的。

Ford–Fulkerson 算法

最简单的网络流算法就是不断增广，直到没有增广路为止。

可以证明，在没有增广路的时候就找到了最大流。

Ford–Fulkerson 算法

最简单的网络流算法就是不断增广，直到没有增广路为止。

可以证明，在没有增广路的时候就找到了最大流。

这样的算法**最坏**是 $\mathcal{O}(|E| \max f)$ 的，f 是最大流大小。

Ford–Fulkerson 算法

最简单的网络流算法就是不断增广，直到没有增广路为止。

可以证明，在没有增广路的时候就找到了最大流。

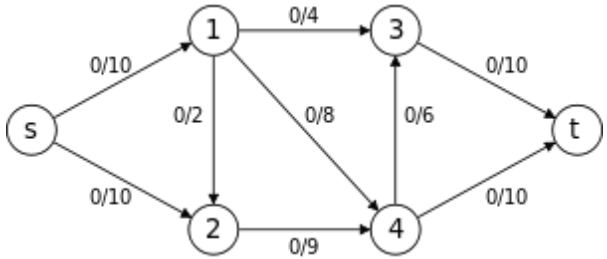
这样的算法**最坏**是 $\mathcal{O}(|E| \max f)$ 的，f 是最大流大小。

和 Bellman-Ford 一样，通常没有人会写这个算法。

Dinic 算法

Ford-Fulkerson 算法每次都寻找一条增广路。那么我们可以不可以一次寻找多个增广路呢？

Dinic 算法就是每次都走最短的增广路，并且每次增广允许多条增广路一起增广。



层次网络

为了避免走太长的路径浪费时间，Dinic 算法每次增广前都对残量网络进行了一个标号，形成一个**层次网络**。

层次网络

为了避免走太长的路径浪费时间，Dinic 算法每次增广前都对残量网络进行了一个标号，形成一个**层次网络**。

它先将边全部反向（实际实现中不必进行这个操作，稍后介绍），然后从 T 开始进行 BFS，距离 T 距离为 k 的边标号为 k。

层次网络

为了避免走太长的路径浪费时间，Dinic 算法每次增广前都对残量网络进行了一个标号，形成一个**层次网络**。

它先将边全部反向（实际实现中不必进行这个操作，稍后介绍），然后从 T 开始进行 BFS，距离 T 距离为 k 的边标号为 k。

这样标号后就形成了一个层次网络。你也可以从 S 开始进行 BFS 标号。

层次网络

miskcoo@bw:~/Code/oicode

```
44  bool label_vertex()
45  {
46      std::memset(label, 0, sizeof(label));
47      int qhead = 0, qtail = 0;
48      label[t] = 1;
49      que[qtail++] = t;
50      while(qhead != qtail)
51      {
52          int u = que[qhead++];
53          for(int k = head[u]; k; k = next[k])
54          {
55              int v = point[k];
56              if(cap[k ^ 1] && !label[v])
57              {
58                  label[v] = label[u] + 1;
59                  que[qtail++] = v;
60                  if(v == s) return true;
61              }
62          }
63      }
64
65      return label[s] != 0;
66  }
```

多路增广

在增广的时候，我们只走满足 $\text{level}[u] = \text{level}[v] + 1$ 的边，这里 level 是前面的 BFS 时的标号。我们称满足上面等式的边为**允许弧**。

多路增广

在增广的时候，我们只走满足 $\text{level}[u] = \text{level}[v] + 1$ 的边，这里 level 是前面的 BFS 时的标号。我们称满足上面等式的边为**允许弧**。

增广的操作实际上是 DFS，从 S 开始，只走允许弧。

多路增广

在增广的时候，我们只走满足 $\text{level}[u] = \text{level}[v] + 1$ 的边，这里 level 是前面的 BFS 时的标号。我们称满足上面等式的边为**允许弧**。

增广的操作实际上是 DFS，从 S 开始，只走允许弧。

在 DFS 到一个节点 u 的时候额外再传入一个参数 rest，表示从 S 到这里最多能流入的流量。DFS 返回的值表示实际使用的流量。

多路增广

在增广的时候，我们只走满足 $\text{level}[u] = \text{level}[v] + 1$ 的边，这里 level 是前面的 BFS 时的标号。我们称满足上面等式的边为**允许弧**。

增广的操作实际上是 DFS，从 S 开始，只走允许弧。

在 DFS 到一个节点 u 的时候额外再传入一个参数 rest，表示从 S 到这里最多能流入的流量。DFS 返回的值表示实际使用的流量。

然后就从这个节点 u 开始，通过允许弧 (u, v) 往下，传给下一个节点的 rest 就是当前 rest 和 $c(u, v) - f(u, v)$ 中较小的那一个。然后当前 rest 减去 DFS 的返回值再找下一条允许弧继续 DFS，直到 rest 为 0 或允许弧都走完的时候就返回。

多路增广

在增广的时候，我们只走满足 $\text{level}[u] = \text{level}[v] + 1$ 的边，这里 level 是前面的 BFS 时的标号。我们称满足上面等式的边为**允许弧**。

增广的操作实际上是 DFS，从 S 开始，只走允许弧。

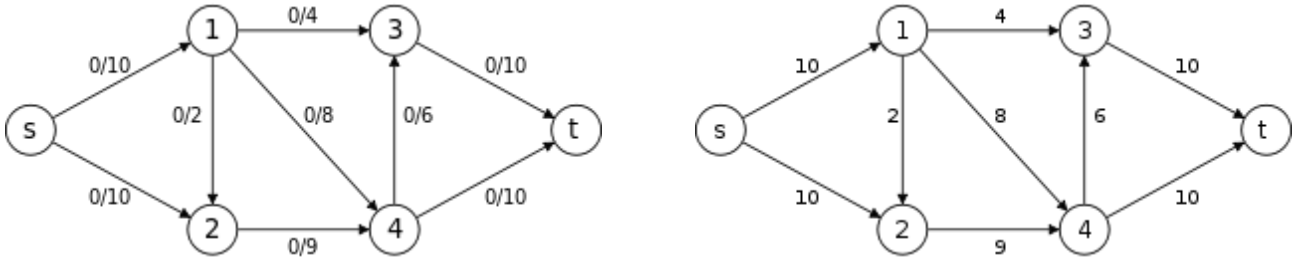
在 DFS 到一个节点 u 的时候额外再传入一个参数 rest，表示从 S 到这里最多能流入的流量。DFS 返回的值表示实际使用的流量。

然后就从这个节点 u 开始，通过允许弧 (u, v) 往下，传给下一个节点的 rest 就是当前 rest 和 $c(u, v) - f(u, v)$ 中较小的那一个。然后当前 rest 减去 DFS 的返回值再找下一条允许弧继续 DFS，直到 rest 为 0 或允许弧都走完的时候就返回。

DFS 的终点就是遇到 T 的时候，这时候直接返回 rest 就可以了。

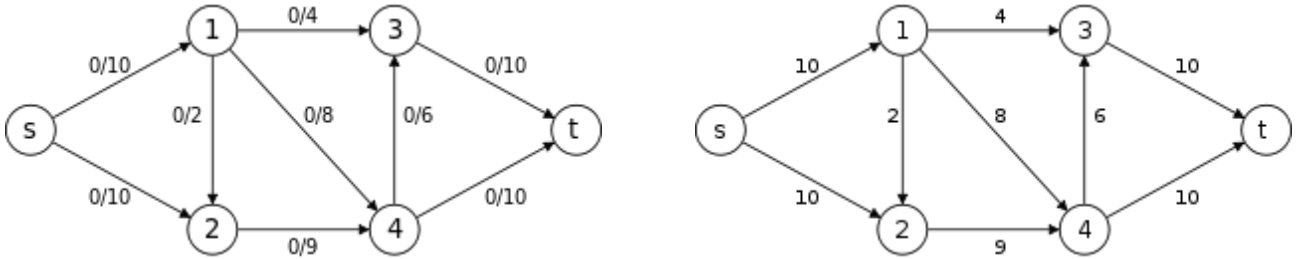
来个模拟

这是前面那个网络，以及它的残量网络

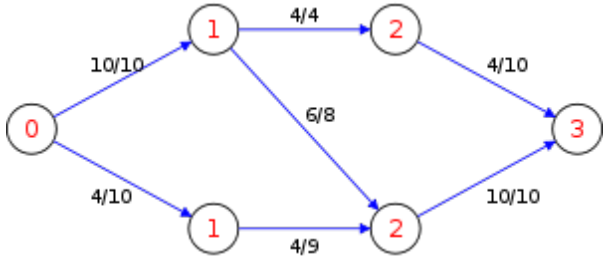


来个模拟

这是前面那个网络，以及它的残量网络

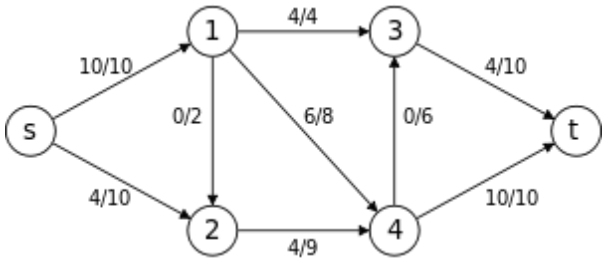


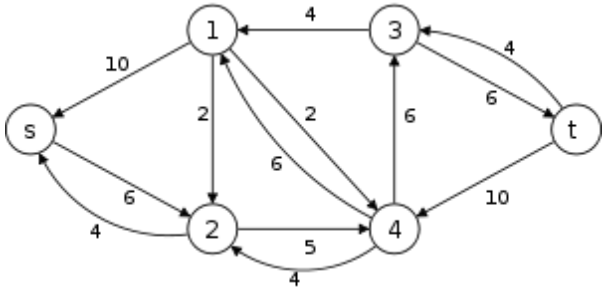
增广路是这样的



来个模拟

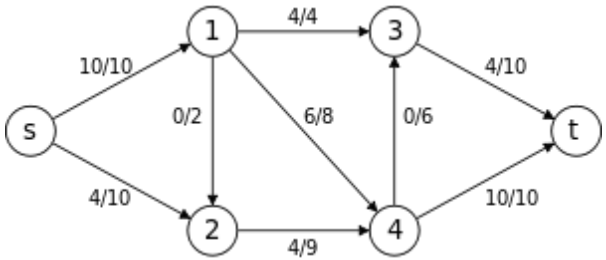
第一次增广后的容量网络和残量网络（加上了反向边）

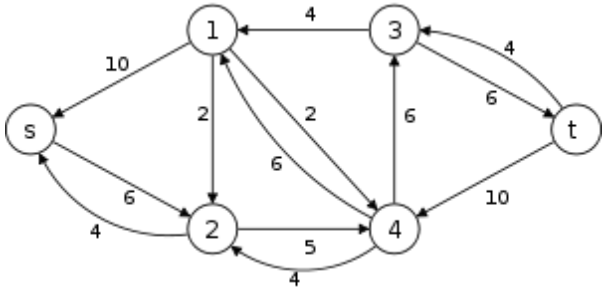




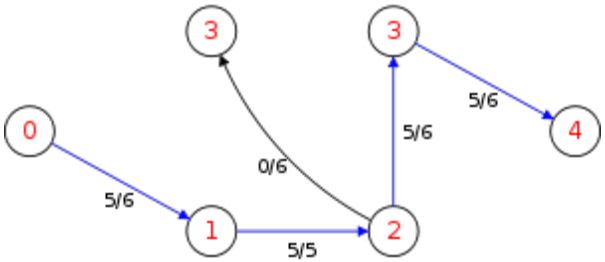
来个模拟

第一次增广后的容量网络和残量网络（加上了反向边）



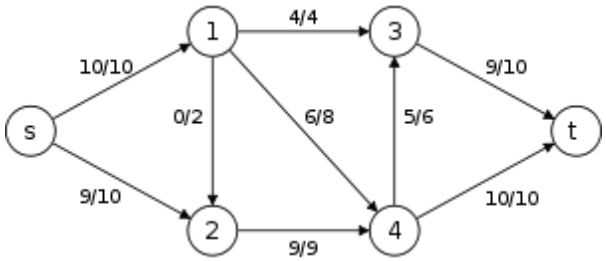


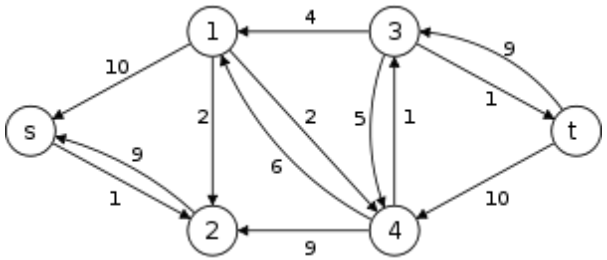
增广路是这样的



来个模拟

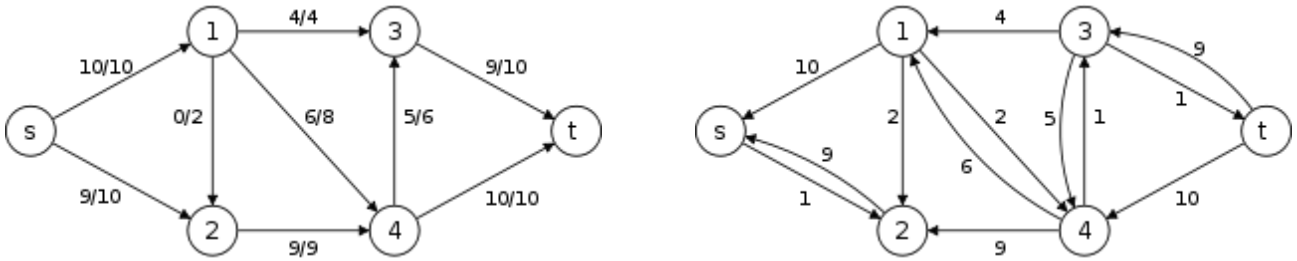
第二次增广后的容量网络和残量网络（加上了反向边）



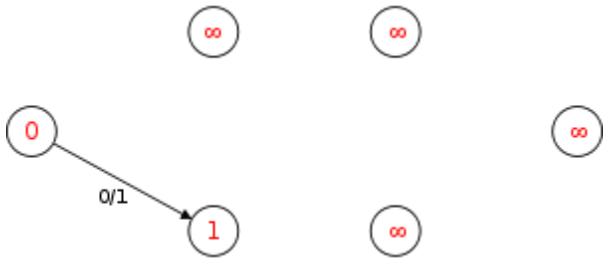


来个模拟

第二次增广后的容量网络和残量网络（加上了反向边）



这时候已经没有增广路了，找到了最大流



多路增广

miskcoo@bw:~/Code/oicode

```
68  int multi_augment(int u, int cap_limit)
69  {
70      if(u == t) return cap_limit;
71
72      int cap_used = 0;
73      for(int& k = cur[u]; k; k = next[k])
74      {
75          int v = point[k];
76          if(!cap[k] || label[v] + 1 != label[u])
77              continue;
78          int ts = std::min(cap_limit - cap_used, cap[k]);
79          int ret = multi_augment(v, ts);
80          cap_used += ret;
81          cap[k] -= ret;
82          cap[k ^ 1] += ret;
83          if(cap_used == cap_limit)
84              return cap_used;
85      }
86
87      return cap_used;
88  }
```

当前弧优化

Dinic 在增广的时候还有一个强有力的优化—当前弧优化。

当前弧优化

Dinic 在增广的时候还有一个强有力的优化—当前弧优化。

想一想，如果一条弧在 DFS 后流量达到了容量限制，那么这次增广就不可能再通过这条弧了，我们之后的 DFS 过程中如果还有遇到这个节点在查找允许弧的时候就不用再查找这条弧了。

当前弧优化

Dinic 在增广的时候还有一个强有力的优化—当前弧优化。

想一想，如果一条弧在 DFS 后流量达到了容量限制，那么这次增广就不可能再通过这条弧了，我们之后的 DFS 过程中如果还有遇到这个节点在查找允许弧的时候就不用再查找这条弧了。

由于我们都是按照边表的顺序存储边的。因此，可以存储下以 u 开头的节点的当前弧 cur[u]，之后遇到 u 直接从 cur[u] 这条弧之后开始寻找，而不必再从 head[u] 开始。

Dinic 算法

Dinic 算法最坏的复杂度是 $O(|V|^2|E|)$ 的。

Dinic 算法

Dinic 算法最坏的复杂度是 $O(|V|^2|E|)$ 的。

虽然如此，实际上复杂度**远远小于**这个值，在 OI 中可以放心使用。

Dinic 算法

Dinic 算法最坏的复杂度是 $O(|V|^2|E|)$ 的。

虽然如此，实际上复杂度**远远小于**这个值，在 OI 中可以放心使用。

有一点就是，由于需要反向边，边的这两条边可以相邻存储，一个存在 2k 这个位置，另一条存在 2k+1 这个位置，在访问的时候会比较方便。

最小割

现在来介绍一个叫做割的概念。

一个网络的**割**是这样一个边集，如果把这个集合的边删去，这个网络就不再连通。

这个边集中边的容量和被称为割的**容量**。

容量最小的割被称为**最小割**。

最大流最小割定理

在网络流中很重要的一个定理是**最大流最小割定理**。

最大流最小割定理

在网络流中很重要的一个定理是**最大流最小割定理**。

Theorem

最大流最小割定理

对于一个容量网络，其最大流等于最小割的容量。

例题. 二分图匹配

给出一个二分图，求其最大匹配。

例题. 二分图匹配

网络流模型可以用来解决二分图匹配问题。

例题. 二分图匹配

网络流模型可以用来解决二分图匹配问题。

将原二分图的边的容量设为 1，S 向左部的节点连一条容量为 1 的边，右部的节点像 T 连一条容量为 1 的边。

例题. 二分图匹配

网络流模型可以用来解决二分图匹配问题。

将原二分图的边的容量设为 1，S 向左部的节点连一条容量为 1 的边，右部的节点像 T 连一条容量为 1 的边。

如果二分图的一条边在匹配中，那么两个点对应的 S 和 T 的边一定满流，不可能再被匹配。

例题. 电力网络

一个电力网络由节点（电站、消费者或中转站）和输电线组成。

一个电站可以生产一些电，一个消费者可以消耗一些电，中转站用来传递电力。

每条输电线单位时间能够传输的电力是有限的。

你需要计算单位时间内这个电力网络最多能够支撑多少电力消耗。

Source：POJ1459. Power Network

例题. 电力网络

我们将电站看成源，消费者看成汇。这样直接连边可以得到一个网络。

例题. 电力网络

我们将电站看成源，消费者看成汇。这样直接连边可以得到一个网络。

但是这个网络可能有多个源多个汇

例题. 电力网络

我们将电站看成源，消费者看成汇。这样直接连边可以得到一个网络。

但是这个网络可能有多个源多个汇

建立一个超级源，一个超级汇

例题. 电力网络

我们将电站看成源，消费者看成汇。这样直接连边可以得到一个网络。

但是这个网络可能有多个源多个汇

建立一个超级源，一个超级汇

从超级源向所有电站连接一条容量为无穷的边，将消费者向超级汇连接一条容量为无穷的边。

例题. 运输网络

给定运输网络，每个中转站和道路都有最大运输量的限制，求最大运输量。

例题. 运输网络

对于边来说没什么问题，点怎么办？

例题. 运输网络

对于边来说没什么问题，点怎么办？

拆点！

例题. 运输网络

对于边来说没什么问题，点怎么办？

拆点！

将一个点 u 拆成入点 u_{in} 和出点 u_{out} 两个，将所有指向 u 的边连接到 u_{in}，容量不变，从 u 连出的边改为从 u_{out} 连出，容量不变。

例题. 运输网络

对于边来说没什么问题，点怎么办？

拆点！

将一个点 u 拆成入点 u_{in} 和出点 u_{out} 两个，将所有指向 u 的边连接到 u_{in}，容量不变，从 u 连出的边改为从 u_{out} 连出，容量不变。

最后再从 u_{in} 向 u_{out} 连接一条容量为 u 容量限制的边。

例题. Sightseeing tour

给定一个 n 个节点，m 条边的混合图，有一些边是有向边，有一些边是无向边。

请给无向边定向，使得最后的有向图存在经过所有边仅一次的回路。并给出方案。

其中 $1 \leq \mathrm{n} \leq 200, 1 \leq \mathrm{m} \leq 2000$。

Source：POJ1637. Sightseeing tour

例题. Sightseeing tour

欧拉回路

例题. Sightseeing tour

欧拉回路

要求每个结点入度等于出度，并且每个结点度数都是偶数

例题. Sightseeing tour

欧拉回路

要求每个结点入度等于出度，并且每个结点度数都是偶数

先计算一个节点的总的度数 d，除以 2 后就是要求的出度和入度

例题. Sightseeing tour

欧拉回路

要求每个结点入度等于出度，并且每个结点度数都是偶数

先计算一个节点的总的度数 d，除以 2 后就是要求的出度和入度

将无向边随意定向，计算入度和出度

例题. Sightseeing tour

欧拉回路

要求每个结点入度等于出度，并且每个结点度数都是偶数

先计算一个节点的总的度数 d，除以 2 后就是要求的出度和入度

将无向边随意定向，计算入度和出度

需要改变 (入度-出度)/2 条边

例题. Sightseeing tour

删去原先就是有向边的边，按照定向后的网络连边，边权为 1

例题. Sightseeing tour

删去原先就是有向边的边，按照定向后的网络连边，边权为 1

节点 u 入度大于出度，连接边 (s, u) 边权为需要改变的边数

例题. Sightseeing tour

删去原先就是有向边的边，按照定向后的网络连边，边权为 1

节点 u 入度大于出度，连接边 (s, u) 边权为需要改变的边数

节点 u 入度小于出度，连接边 (u, t) 边权为需要改变的边数

例题.cf498C.Array and Operations

给定一个长为 n 的数组，以及 m 对下标 (a, b) 且满足 $a + b$ 为奇数，每次操作可以将同一组的两个数同时除以一个公约数

问最多能进行多少次操作

$1 \leq n, m \leq 100, 1 \leq a_i \leq 10^9$

题目显然给出的是二分图

题目显然给出的是二分图

每个质因数分开考虑

题目显然给出的是二分图

每个质因数分开考虑

将有关的点连 inf 的边，源向奇点连容量为因子数量的边，偶点向汇连

题目显然给出的是二分图

每个质因数分开考虑

将有关的点连 inf 的边，源向奇点连容量为因子数量的边，偶点向汇连

最大流

题目显然给出的是二分图

每个质因数分开考虑

将有关的点连 inf 的边，源向奇点连容量为因子数量的边，偶点向汇连

最大流

例题. 路障

给定一张 n 个点 m 条边无向图，要求选择最少的边，使得 1 到 n 的最短路一定经过其中的边

例题. 路障

给定一张 n 个点 m 条边无向图，要求选择最少的边，使得 1 到 n 的最短路一定经过其中的边

$1 \leq n \leq 1000, 1 \leq m \leq 10000$

例题. 路障

求最短路图上的最小割

例题. Matrix

有一个 n $\times$ m 的网格，每个格子有价值 c_{ij} 的宝石。

问最多能够取价值多少的宝石，使得任意两块宝石不相邻。

其中 $1 \leq n, m \leq 50, 0 \leq c_{ij} \leq 40000$。

Source：HOJ 2713. Matrix1

例题. Matrix

先将网格黑白染色

例题. Matrix

先将网格黑白染色

源向黑色点连接容量为价值的边，白色点向汇连接容量为价值的边。

例题. Matrix

先将网格黑白染色

源向黑色点连接容量为价值的边，白色点向汇连接容量为价值的边。

每个点向周围四个点连接容量 ∞ 的边。

例题. Matrix

先将网格黑白染色

源向黑色点连接容量为价值的边，白色点向汇连接容量为价值的边。

每个点向周围四个点连接容量 ∞ 的边。

答案是价值和减去最小割（最大流）。

例题. 移动干草

一个牧场由 r * c 个格子组成，牧场内有 n 条干草运输通道，每条连接两个水平或垂直相邻的方格，最大运输量为 L_i。

$(1,1)$ 有很多干草，求其向 (r,c) 能运输的最大运输量。

例题. 移动干草

一个牧场由 r * c 个格子组成，牧场内有 n 条干草运输通道，每条连接两个水平或垂直相邻的方格，最大运输量为 L_i。

$(1,1)$ 有很多干草，求其向 (r,c) 能运输的最大运输量。

$1 \leq r, c \leq 200$

例题. 移动干草

网络流？

例题. 移动干草

网络流？

点数 40000，边数 80000 TLE...

例题. 移动干草

网络流？

点数 40000，边数 80000 TLE...

题目给出的是平面图

平面图是可以画在平面上并且使得不同的边可以互不交叠的图。

例题. 移动干草

网络流？

点数 40000，边数 80000 TLE...

题目给出的是平面图

平面图是可以画在平面上并且使得不同的边可以互不交叠的图。

利用最短路求最小割！

图

给一张联通无向图，定义 Dist[i][j] 表示点 i 到点 j 之间的最短路

当前已经有了若干条的边，再给定 N 个 A[i]

要求添加若干条无向边，使得 $\sum$(a[i]–Dist[1][i])^2 最小

输出最小的答案

其中 $1 \leq n \leq 40, 1 \leq a_i \leq 40$

最短路满足 $\left|\text{Dist}[i]–\text{Dist}[j]\right| <= 1$，0 号节点的 Dist 一定是 0，其他的一定 >=1

因为可以任意添加边，所以任意一组满足上面的条件的 dist 都是合法的

我们要给每一个节点指定一个 dist，且满足对于原图的任意一条边，$\left|\text{Dist}[i] - \text{Dist}[j]\right| <= 1$

要求最小化 $\sum(\text{Dist}[i] - \text{A}[i])^2$

拆开绝对值，就是 $\text{Dist}[i] - \text{Dist}[j] <= 1$，并且 $\text{Dist}[j] - \text{Dist}[i] <= 1$

每一个限制都可以理解为: 如果 i 取了 Dist[i] 的话，j 至少得取 Dist[i]-1 以上。

我们用网络流模型来刻画这个东西

因为 dist 的取值只有 N 种，我们对于每个点，拆成 N 个点

$(i,j)->(i,j+1)$ 容量为 $(A[i]-j)^2$

对于原图的一条边 (u,v)

对于所有的 j

连接 $(u,j)->(v,j-1),(v,j)->(u,j-1)$，容量均为正无穷

考虑这个图的割，很显然它只可能割在第一种的边上

连接 $(u,j)->(v,j-1)$，显然如果 u 这个点割在了 $(u,j)->(u,j+1)$ 的话，v 至少得割一条 $>=j-1$ 的边才能满足没有增广路

构图成立，直接跑最小割即可

又见运输网络

假设还是开头的运输网络。这次事情发生了一些改变，每条运输路线运输一单位的货物需要缴纳一些费用。

你想要知道在运输量最大的前提下，最小的费用是多少。

又见运输网络

假设还是开头的运输网络。这次事情发生了一些改变，每条运输路线运输一单位的货物需要缴纳一些费用。

你想要知道在运输量最大的前提下，最小的费用是多少。

这样的模型被称为**最小费用最大流**。当然，如果你想要求的是最大费用，这就被称为**最大费用最大流**。

最小费用最大流

由于前提是最大流，回忆 Ford-Fulkerson 算法，我们每次在找增广路的时候都找费用最小的增广路就可以了。

这可以直接利用 SPFA 算法来寻找，边权就是费用。

例题. 带权二分图匹配

给出一个二分图，点带有权，求其最大匹配，并且满足点权和最小。

例题. 带权二分图匹配

从 S 到左部节点及从右部节点到 T 连容量为点权的边。

原二分图的边保留，容量 ∞。

例题. k 取方格数

有一个 n × n 的方格图，方格中有正整数。从左上角开始走到右下角，只能向右或者向下走，走的过程中取出方格的数（取出后方格数字变为 0）。

一共可以走 k 次，你需要求出最多能够得到多少的数。

例题. k 取方格数

源向左上角节点，右下角节点向汇分别连接一条容量为 k，费用为 0 的边

例题. k 取方格数

源向左上角节点，右下角节点向汇分别连接一条容量为 k，费用为 0 的边

将每个方格拆成两个点，入点和出点

例题. k 取方格数

源向左上角节点，右下角节点向汇分别连接一条容量为 k，费用为 0 的边

将每个方格拆成两个点，入点和出点

入点和出点连接一条容量为 1，费用为方格数的边，再连接一条容量无穷费用为 0 的边

每个方格出点向右侧和下方方格入点连接流量无穷费用 0 的边

例题. k 取方格数

源向左上角节点，右下角节点向汇分别连接一条容量为 k，费用为 0 的边

将每个方格拆成两个点，入点和出点

入点和出点连接一条容量为 1，费用为方格数的边，再连接一条容量无穷费用为 0 的边

每个方格出点向右侧和下方方格入点连接流量无穷费用 0 的边

最大费用最大流