Lecture 1
流 | Flow

網路流與我們生活中的一些問題息息相關，比如我們要從一工作站 s 傳輸一個資料到工作站 t ，傳輸的過程有一些中繼站，並且某些中繼站間和某些中繼站到工作站間有傳輸率爲 c 的網路線相通，現在要如何找出一個最佳的傳輸方案，使得 s 到 t 的資料傳輸速率最快?

以上就是網路流問題的原型，但隨後人們驚訝的發現網路流還可以用以解決一些看似無關的問題! 如匹配問題、歐拉回路問題等等。網路流的問題千變萬化，常常令人異想不到，也因此常成爲程式競賽的難題，如果能精熟這個領域，便如有了一把解題的利刃!

1.1 前言

1.1.1 問題的定義

首先我們定義何謂網路流。

定義： 一個 **s-t 網路流**是一張圖 (有向或無向) $G = (V, E)$ ，每條邊上有一個非負的權重 $c(u, v) \geq 0$ 代表邊的**流量上限** (如果 $(u, v) \notin E$ 我們定義 $c(u, v) = 0$)。並且有二個特別的點，源點 s 與匯點 t 。

一個 **s-t 可行流** 是一個函數 $f : V \times V \mapsto \mathbb{R}$ 滿足以下兩個條件

- $f(u, v) \leq c(u, v),\ \forall (u, v) \in V \times V$ (流量限制) (1.1)
- $f(u, v) = -f(v, u),\ \forall (u, v) \in V \times V$ (流量對稱) (1.2)
- 對於所有 $v \in V$ ，有 $\sum_{u \in V} f(v, u) = 0$ (流量守衡) (1.3)

而我們定義這個 s-t 流的**流量**爲

$$|f| = \sum_{v \in V} f(s, v)$$

換句話說，我們必須給出每一條邊應往哪個方向流、流多少。當然，不是隨隨便便流都可以的，必須滿足這三個限制!

(1.1)：每條邊的流量都沒有超過他的限制。
(1.2)：如果 u 往 v 流 c 單位就相當於 v 往 u 流 $-c$ 單位。
(1.3)：流不能無中生有! 每一點除了 s, t 以外進去的流量要等於出去的流量，也就是淨流量必須是 0 。

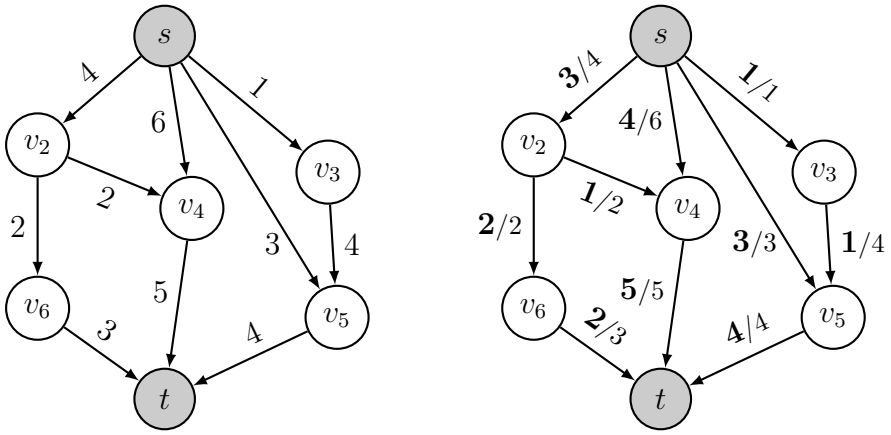


圖 1.1: 一個流量網路的例子

如圖 1.1 就是一個流量網路的例子，而右圖則是他的一組解。讀者可以驗證這個解的確滿足 s-t 可行流的三個限制。

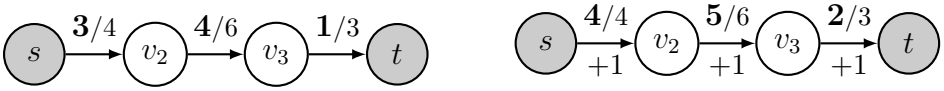
而通常我們會希望流量 $|f|$ 可以越大越好，$|f|$ 最大的 s-t 可行流就稱作**最大流**。

1.2 最大流演算法

1.2.1 剩餘網路

我們定義了最大流後，現在的問題是要如何找出一個最大流以及他的流量呢?

一個最基本的想法應該是這樣子：不斷的增加流量，直到流滿爲止。但如何「增加流量」，又何謂流滿呢? 觀察一下可以發現，如果我們可以找到一條從 s 到 t 的路徑，而且路徑上的邊都還有剩餘流量，那我們就可以延著這條路徑擴充容量!



因此我們就有以下的方法擴充容量：把圖上所有還沒流滿的邊都拿出來，如果這些邊構成了一個從 s 到 t 的路徑，那我們就延著這條邊增加流量。而直到沒有一條從 s 到 t 的路徑，我

們就說已經「流滿」了!

但這個演算法眞的是正確的嗎? 很不幸的，以下就是一個簡單的反例。

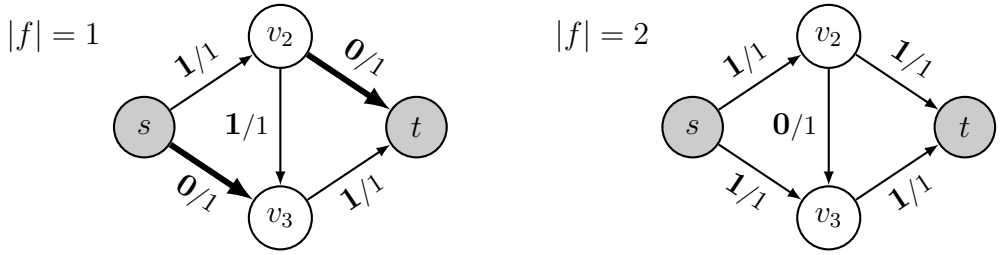


圖 1.2: 一個反例，粗邊表示還沒流滿的邊。
雖然沒有 s 到 t 的路徑，但右圖才是最佳解。

其實我們以經非常接近答案了，只是因爲我們少考慮了一種「可以流」的情況!

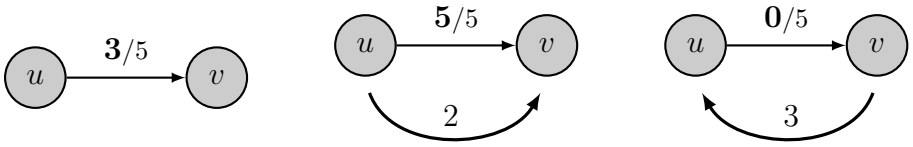


圖 1.3

如圖 1.3 所示，假設 (u, v) 的流量上限是 5 ，而已經流了 3 單位的流量。我們可以增大流量，相當於再流 x 單位過去，也可以減少流量，**其實相當於流 x 單位回來!**

因此我們定義剩餘流量如下:

定義: 一個邊的剩餘流量 $r(u, v)$ 定義爲

$$r(u, v) = c(u, v) - f(u, v)$$

如果 $r(u, v) > 0$ ，則可以沿著這些邊擴充流量，因此 $r(u, v) > 0$ 的所有邊構成的圖就稱作**剩餘網路** ，對於一個流 f 記作 G_f。

比如圖 1.3 中 $r(u, v) = c(u, v) - f(u, v) = 2$，至於 $r(v, u)$ ，因爲 (v, u) 沒有 (有向) 邊，所以可以看作 $c(v, u) = 0$ ，因此 $r(v, u) = c(v, u) - f(v, u) = 0 - (-f(u, v)) = 3$ 。

如果我們把反向的剩餘流量也考慮進去後，可以證明之前的演算法就是正確的了! 也就是我們只要不斷在剩餘網路中找 $s \rightsquigarrow t$ 的擴充路徑增加流量，直到沒有 $s \rightsquigarrow t$ 的路徑存在後，我們就找到最大流了。

1.2.2 對偶

讓我們先暫停一會兒，來看看什麼是對偶問題。

我們遇到的問題往往是一體兩面的，在求某個問題的最小值其實就是在求另一個問題的最小值。比如說「某 A 最多可以有幾個蘿莉在身旁還能保持理智」其實就是在問「要陷害某 A 去坐牢最少要準備多少蘿莉」。或著「給你 1000 元你最多可以活幾天」其實就跟「你要活一個月至少需要多少錢」是同一類問題。

那最大流的對偶問題是什麼呢？想像一下今天你家裝了光世代 1G ，回家開 speedtest.net 測速發現只有 10M ，爲什麼會這樣呢？原因就在光世代 1G 只是把你家出去的那條網線路換成 1G 流量而已，而網路連線要經過很多條線路，如果有一條線路只有 10M 的流量，那不管從你家出去的那條網路線有多快，瓶頸永遠卡在那條慢網路網。

這告訴了我們一件事情:

> 你的網速的最大值，就是途中經過的那些網路線中流速的最小值

對最大流問題來說，就是

> **一個網路的最大流，就是把 s 和 t 卡住的那些邊的流量總和的最小值。**

我們把這個敍述用正式一點的語言描述:

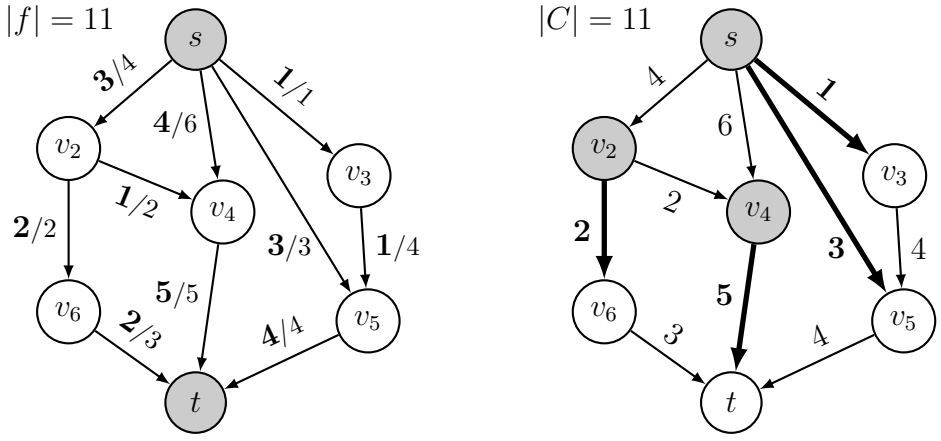
定義： 一個 **s-t 割** (cut) C 是一個圖的分割，將圖中的點分成兩個集合 $C = (S, T)$，滿足 $s \in S$, $t \in T$。我們定義這個割的大小爲

$$|C| = \sum_{u \in S} \sum_{v \in T} c(u, v).$$

也就是所有滿足 u 在 S，v 在 T 的邊 (u, v) 的流量上限合。
(並不計算 u 在 T ，v 在 S 的邊。)

一個**最小割**就是最小的一個割。

而一個圖的最大流就是最小割的大小!



我們用以下這個定理做總結。

Max-flow min-cut theorem： 對一個流量網路 G，以下三件事情是等價的。

1. f 是一個 s-t 最大流。
2. 關於 f 的剩餘網路 G_f 沒有從 s 到 t 的擴充路徑。
3. $|f| = |C|$，其中 C 是最小 s-t 割。

1.2.3 Max-flow min-cut theorem 的證明 *

Proof. (1) ⇒ (2)，由上面的討論可知如果 G_f 有一條 s 到 t 的擴充路徑，就可沿著這條路徑擴充增加流量，因此 G_f 沒有擴充路徑。

(2) ⇒ (3)，我們要先證明一個引理

引理 對於一個流量 f，定義一個 s-t cut (S, T) 的截流爲

$$|f(S,T)| = \sum_{u \in S, v \in T} f(u,v)$$

則 $|f(S,T)| = |f|$ 。

Proof. 注意到 $\sum_{u,v \in A} f(u,v) = 0$，因爲 $f(u,v) = -f(v,u)$，且由點的流量守恆，$\sum_{v \in V} f(u,v) = 0,\ \forall u$，因此

$$\begin{aligned} |f(S,T)| &= \sum_{u \in S, v \in T} f(u,v) = \sum_{u \in S, v \in T \cup S} f(u,v) \\ &= \sum_{u \in S \setminus \{s\}, v \in V} f(u,v) + \sum_{v \in V} f(s,v) \\ &= 0 + |f| = |f| \end{aligned}$$

□

現在我們回來證明原來的問題，我們**令 S 表示所有可以從 s 經由 G_f 上的邊**走到的點，$T = V \setminus S$，則 (S,T) 是一個 s-t cut，因爲 $s \in S$ 且由 G_f 沒有 s 到 t 的擴充路徑可以知道 $t \in T$。由引理我們可以知道 $|f(S,T)| = |f|$，而因爲剩餘網路沒有從 S 到 T 的邊了，可以知道原本圖上 S 到 T 的邊都以經流滿了，因此 $|f| = |f(S,T)| = |C|$。

而在由引理我們知道對於任何一個割 C，$C \geq |f|$，因此剛剛我們討論的 (S,T) 的確是最小割了，這也順便證明了 (3) ⇒ (1)。 □

注意到這個證明也順便給了找一個最小割的答案的方法，**把所有 s 在最大流 f 的剩餘網路 G_f 可以走到的點的集合當作 S，剩下的點當作 T，就是一個最小割。**

1.2.4 Ford–Fulkerson algorithm

之前以經說過，就算忘了 Max-flow min-cut theorem 再提醒了我們一次只要不斷在剩餘網路中找 $s \rightsquigarrow t$ 的擴充路徑增加流量，直到沒有 $s \rightsquigarrow t$ 的路徑存在後，就可以找到最大流了。因此一個最簡單的演算法如下。

Algorithm 1: Ford-Fulkerson algorithm

```
1 Algorithm FordFulkerson()
2   while We could find an augmenting P path from s to t do
3     Δf ← AugmentAlongPath(P)
4     f ← f + Δf
5   end
```

這邊有幾個問題要注意

1. 如何找一個擴充路徑?
   基本上可以直接遞迴下去做 (相當於最普通的 DFS) 即可。
2. 找到了擴充路徑 P 後，如何擴充?
   首先顯然增加的流量就是**路徑上剩餘流量的最小值**，$\Delta f = \min_{e \in P} r(e)$。之後我們便將路徑上的邊的流量都加 Δf，但別忘了由定義 $f(u,v) = -f(v,u)$，也就是我們必須把反向的邊的流量減掉! 因此我們總結 f, r 的變化，假設 $(u,v) \in P$:

$$f(u,v) \leftarrow f(u,v) + \Delta f, \quad r(u,v) \leftarrow r(u,v) - \Delta f$$
$$f(v,u) \leftarrow f(v,u) - \Delta f, \quad r(v,u) \leftarrow r(v,u) + \Delta f$$

3. 圖的儲存:
   由於 c, f, r 只要知其二便可知剩下的一個，因此只需要存任兩個就好! 不過在 2. 我們發現其實只要知道 r 我們就知道 Δf 了，因此**在大部分的情況下，只需要存 r 就足夠了**。而在一開始時，$f = 0$，因此 $r = c$。

但還有一個最重要的問題，要做幾次擴充路徑這個演算法才會結束呢? 很遺憾的是 Ford-Fulkerson algorithm 沒有辦法給出一個好的次數上限! 事實上有:

- 如果流量上限都是整數，那麼這個演算法最差要做 $\mathcal{O}(|f|)$ 次的擴充，其中 $|f|$ 是最大流的流量。總時間複雜度是 $\mathcal{O}(|E||f|)$。
- 如果流量上限有無理數，那麼這個演算法無法保證會結束!

但神奇的是，我們只需要改變一點就可以改進這個演算法!

1.2.5 Edmonds-Karp algorithm

Edmonds-Karp algorithm 是 Ford-Fulkerson algorithm 的一個小改進，他的演算法如下:

Algorithm 2: Ford-Fulkerson algorithm

```
1 Algorithm FordFulkerson()
2   while We could find an shortest augmenting P path from s to t do
3     Δf ← AugmentAlongPath(P)
4     f ← f + Δf
5   end
```

咦! 怎麼好像一點差別也沒有? 要仔細看! 原來之前 Ford-Fulkerson algorithm 是隨便找一條擴充路徑，現在 Edmonds-Karp 規定每次找的擴充路徑要是**最短的一條**。

令人意外的，加上了這個優化之後，可以證明最多只會找 $\mathcal{O}(VE)$ 次擴充路徑，不論流量是不是無理數! 有時候一個小小的優化也可以造成很大的改變。

Edmonds-Karp algorithm 的證明 *

定義 我們定義 $d(u)$ 表示一個點到 t 的最短距離。

定理 Edmonds-Karp 最多只會進行 $\mathcal{O}(VE)$ 次擴充。

Proof. 我們先證明一些引理:

引理 在 Edmonds-Karp 的演算法中，也就是如果每次都找最短路擴充，則 $d(u)$ 永遠不會減少。

Proof. 假設經過一次擴充，流量從 f 變成 f'，而某個 $d_f(u)$ 減少成了 $d_{f'}(u)$，找 $d_{f'}(u)$ 最少的那一個 u。且令 $G_{f'}$ 中到 u 的最短路是 $s \rightsquigarrow v \to u$，也就是說 $d_{f'}(v) = d_{f'}(u) - 1$，由 u 的挑選我們知道 $d_{f'}(v)$ 沒有減少，也就是 $d_{f'}(v) \geq d_f(v)$。我們宣稱 $(v,u) \notin E(G_f)$，否則

$$d_f(u) \leq d_f(v) + 1 \leq d_{f'}(v) + 1 = d_{f'}(u) < d_f(u), \quad \Rightarrow\Leftarrow$$

所以 $(v,u) \notin E(G_f)$，但 $(v,u) \in E(G_{f'})$，因此擴充的路徑一定有 (u,v) ，但

$$d_f(u) = d_f(v) - 1 \leq d_{f'}(v) - 1 = d_{f'}(u) - 2 < d_f(u) - 2, \quad \Rightarrow\Leftarrow$$

固 $d(u)$ 永遠不會減少。 □

定義 一個邊 (u, v) 是**重邊**表示他是在擴充路徑上剩餘流量最小的一條。

引理

1. 在 Edmonds-Karp 的演算法中，每次擴充至少會有一條重邊。
2. 每個邊最多成爲 $\mathcal{O}(V)$ 次重邊。

Proof. (1) 是顯然的。

假設在一次擴充，(u, v) 是重邊，表示在擴充前 $d_f(v) = d_f(u) + 1$，那經過這次擴充後，因爲擴充的量正好就是 (u, v) 的剩餘流量，所以 $(u, v) \notin G_{f'}$。而 (u, v) 要再一次成爲重邊前，一定要從 (v, u) 擴充過，否則 (u, v) 不會在剩餘網路裡。在 (v, u) 的那次擴充，有

$$d'(u) = d'(v) + 1 \geq d_f(v) + 1 = d_f(u) + 2$$

也就是在下一次 (u, v) 成爲重邊後，$d(u)$ 至少增加了 2 ，注意到 $d(u) > |V|$ 就表示 v 連不到 t 了，所以 (u, v) 至多成爲重邊 $|V|/2 = \mathcal{O}(V)$ 次。 □

由上一個引理知道一個邊最多成爲重邊 $\mathcal{O}(V)$ 次，而每次擴充都要有一個重邊，因此擴充的次數不會超過 $\mathcal{O}(VE)$ 次，整體的複雜度就是 $\mathcal{O}(VE^2)$。 □

1.2.6 Dinic's algorithm *

Dinic 是 Edmonds 的一個優化，時間複雜度是 $\mathcal{O}(V^2E)$，他的想法其實很簡單。

剛剛我們證明了 $d(u)$ 永遠不會減少，也就是最短路只會越來越長，因此我們甘脆一次把長度 1 的擴充路徑擴充完，再把長度 2 的擴充路徑全部擴充...一直到最後把長度 $|V| - 1$ 的路徑擴充完，我們就找到最大流了。

具體的做法是這樣的，我們先求出所有點與 s 的距離 $d(u)$ ，假設我們現在在進行長度爲 k 的擴充。對於每一個點 u ，我們定義 $N_u = \{v : d(v) = d(u) + 1\}$，也就是如果 (u, v) 在擴充路徑上，v 一定在這個集合內。

現在我們要找一條擴充路徑，我們就從 s 開始 DFS，假設我們 DFS 到了 u 這個點，令 $N_u = \{n_u(1), n_u(2), \cdots, \}$，我們就先試 $n_u(1)$，如果這個點可以走到 t，那我們就找到一條擴充路徑了，否則我們就在試 $n_u(2)$。**關鍵在於如果 $n_u(m)$ 試過了沒有成功，下次就不用在試了，從 $n_u(m+1)$ 開始試就可以了。**也就是我們把 $m(u)$ 記下來，代表下次我們 DFS 到這個點，下一個點從哪裡開始試就可以了。

這樣的複雜度是多少呢? 注意到每個點要試的下一個點不超過 $\deg(u)$，因此全部人要試的選擇不超過 $\mathcal{O}(E)$，令外每一次擴充一定會「毀掉」至少一條邊 (也就是擴充路徑上剩餘流量最少的那一條)，也就是一定有一個人的 $m(u)$ 會增加，所以至多擴充 $\mathcal{O}(E)$ 次。而每一次

擴充的路徑長是 $\mathcal{O}(k)$，因此擴充完一個長度的擴充路徑要 $\mathcal{O}(kE)$，總複雜度是

$$\sum_{k=1}^{|V|-1} \mathcal{O}(kE) = \mathcal{O}\left(V^2E\right).$$

以下是一個範例的程式碼。

Dinic 程式碼片段 1-1

```cpp
#include <bits/stdc++.h>
using namespace std;

template<typename T>
struct Dinic{
    static const int MXV = 10000;
    static const T   INF = 1e9;
    struct Edge{
        int v;
        T   f;
        int re;
    };
    int n, s, t, level[MXV];
    vector<Edge> E[MXV];
    int now[MXV];
    void init(int _n, int _s, int _t){
        n = _n; s = _s; t = _t;
        for (int i=0; i<n; i++) E[i].clear();
    }
#define SZ(x) ((int)(x).size())
    void add_edge(int u, int v, T f){
        E[u].PB({v, f, SZ(E[v])  });
        E[v].PB({u, f, SZ(E[u])-1});
    }
    bool BFS(){
        fill(level, level+n, -1);
        queue<int> que;
        que.push(s); level[s] = 0;
        while (que.size()) {
            int u = que.front(); que.pop();
            for (auto it : E[u]){
                if (it.f > 0 && level[it.v] == -1){
                    level[it.v] = level[u]+1;
                    que.push(it.v);
                }
            }
        }
        return level[t] != -1;
    }
```

```
40      T DFS(int u, T nf) {                                              ▲
41          if (u == t) return nf;
42          T res = 0;
43          while (now[u] < SZ(E[u])){
44              Edge &it = E[u][now[u]];
45              if (it.f > 0 && level[it.v] == level[u]+1){
46                  T tf = DFS(it.v, min(nf, it.f));
47                  res += tf; nf -= tf; it.f -= tf;
48                  E[it.v][it.re].f += tf;
49                  if (nf == 0) return res;
50              } else now[u] ++;
51          }
52          if (!res) level[u] = -1;
53          return res;
54      }
55      T flow(T res=0){
56          while ( BFS() ) {
57              T temp;
58              memset(now, 0, sizeof(now));
59              while ((temp = DFS(s, INF))) {
60                  res += temp;
61              }
62          }
63          return res;
64      }
65  };
```

1.3 最大流模型

知道了如何解最大流問題之後，我們就可以開始解一些題目了。

例題 1-1 Internet Bandwidth — UVa 820

現在有一些機器可以互相傳訊，但兩個機器 u, v 間會有一個傳訊的頻寬上限 $c(u, v)$，問你從機器 s 傳訊到 t 的最大頻寬是多少？

這題沒有什麼好說的，裸的最大流問題。

但最大流問題常常會出的讓你乍看之下不像最大流的題目，要經過一些轉換或是先得出一些結論後，才能用最大流問題來解！這也是爲什麼最大流問題往往是難題的原因!

比如以下這題

例題 1-2 不重複路徑數 經典問題

給你一張圖 G，請找出最多的 u 到 v 的路徑，使得在這些路徑上的點都不重複。

嗯？好像題目裡沒有一個「流」字啊？不過我們可以這樣想，路徑不能重複，就是每條邊最多只能被走過一次！因此把每個邊的流量設爲 1，求 u 到 v 的最大流就是答案!

這種把看似不相關的題目轉化成別的題目來求解就是建模。我們整理一下常見的建模手法。

1. 點有容量限制：我們最大流問題中的流量限制都是在邊上而已，無法限制點容量！不過我們可以把每個原來的每個點 v 都拆成 v^+, v^-，分別連接進來的流量和出去的流量，而 $c(v^+, v^-)$ 就設成點的流量上限即可!
2. 多個源/匯點：有時候源/匯點不只有一個，這是後可以多建一個超級源/匯點，並連一個無限大流量的邊到所有源/匯點。
3. 無限制流量邊：通常選一個很大的數，如 2^{30} 等等當做容量即可，只要保證流量永遠不可能達到這個數值，以及這個數值不會在計算過程中溢位即可。

我們再來看一些例題。

例題 1-3 二分圖最大匹配 經典問題

給你一個二分圖，求他的最大匹配。

所謂的匹配就是把盡量多的點對配在一起，兩個點可以被配對若且爲若他們有邊相連，而且一個點只能被配到一次。你可以想像點就代表一些男生女生，有邊相連表示彼此喜歡，你要當月下老人湊合最多對出來。

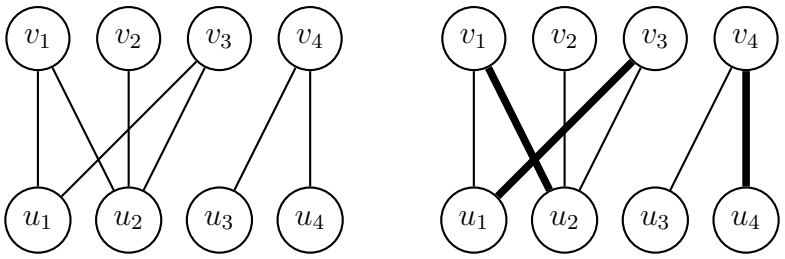


圖 1.4: 一個二分圖匹配

經過一翻思考我們可以發現「只能被配到一次」可以用流量 1 來限制，具體來說假設二分圖的兩個點集是 V_1, V_2，V_1 裡的所有的點就從 s 流一條流量爲 1 的邊，V_2 裡的點則流一條流量 1 的邊到 t。而本來的邊流量上限也全部設成 1，求 s 到 t 的最大流就是答案。

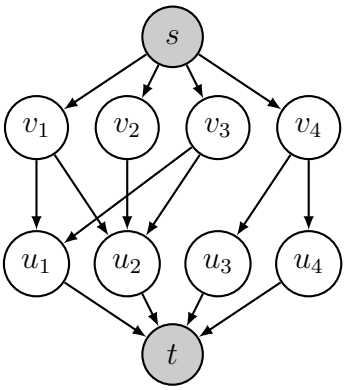


圖 1.5: 二分圖的建模

例題 1-4 混合圖的歐拉回路 經典問題

給你一個連通的混合圖，也就是同時存在有向邊和無向邊的圖，問你存不存在一條歐拉回路。

要解決這題必須先有一些知識！一個無向圖有沒有歐拉回路的條件大家都會，就是每個點的度數都要是偶數且連通。而有向圖的歐拉回路存在的條件也差不多:

- 每個點的入度要等於出度，也就是 $\deg^+(v) = \deg^-(v)$。
- 所有的點連通。

那有向圖呢？其實我們要做的事情就是把所有的無向圖定向，讓整張圖變成有向圖，並滿足以上兩點的限制。首先我們先定義 $\delta(v) = \deg^-(v) - \deg^+(v)$，也就是不計算無向邊的情況下，入邊跟出邊的差。我們的目標就是最後要讓 $\forall v, \delta(v) = 0$。可以這樣想，$\delta(v) > 0$ 的點表示入邊太多了，必須想辦法從無向邊「流出」$\delta(v)$ 的量，才可以變成 $\delta(v) = 0$，$\delta(v) < 0$ 亦同，表示要「流入」$|\delta(v)|$。有沒有發現這個看似與流量無關的題目瞬間與最大流有了聯繫! 我們只需要對於 $\delta(v) > 0$ 的點從 s 建一條流量爲 $\delta(v)$ 的邊 [1]，$\delta(v) < 0$ 的點則建一條到 t 流量爲 $-\delta(v)$ 的邊。而如果 (u, v) 本來有無向邊相連，就建一條 (u, v) 流量爲 1 的無向流量。最後看最大流是不是等於 $\sum_{\delta(v)>0} \delta(v)$，也就是是否所有的點都被補滿了即可。

1.4 最小割模型

大家應該還記得前面有提到最小割的對偶問題就是最大流，因此有些題目會以最小割的型式出現，這些題目也可以轉化成最大流來解。比如這個問題

[1]可能有人會認爲是要從 v 建到 t 才對，其實不然，因爲這些點必需要流出 $\delta(v)$，可以想成我們從 s 強制給了他 $\delta(v)$ 的流，這樣他就必定要流出 $\delta(v)$ 了。

例題 **1-5** Angry Programmer — UVa 11506

你被你老闆開除了，因此你決定在他打 LOL 的時候讓他斷線。從他的電腦 s 到 LOL 伺服器 t 有一些 Router，Router 間有一些網路線相連，剪掉 u 到 v 的網路線要花 $c(u,v)$，如果 s 到 t 沒辦法經過一些 Router 連到就會斷線，問你要達成計畫的最小花費。

這個問題其實就是裸的最小割問題，直接轉成最大流來解就可以了。不過當然，還是有一些經典的建模方式要了解!

1.4.1 「分兩類」問題

這個是最小割最常見的應用! 通常題目會要你把點分成兩堆，比如以下這題。

例題 **1-6** 生產產品問題 — 經典問題

你有 n 個產品可以生產，並且有 m 種不同的機器，生產第 i 個產品必須要有一些機器，但不同的產品如果用到相同的機器的話只需要一個機器即可，現在給你每個機器的價格，和生產每個產品的獲利，你要決定要生產哪些產品使你的獲利最高。

其實你要做的事情就是把產品和機器分成兩類，要的和不要的。可以想像每一個產品和機器都是一個點，現在要把他們分成 (S,T)，其中 S 是我們要的，T 是我們不要的。有沒有突然有一種 Cut 的感覺了? 我們只要讓他的割剛好等於題目的 cost 就可以了! 以下列出一些經典的建模方式。

1. 選 u 但不選 v 要花費 c: 由我們的定義 $u \in S, v \in T$，因此由割的定義我們知道只要建 $u \to v$，流量是 c 的邊即可。
2. 選 u 就一定要選擇 v: 其實這只是 1. 的一個特例而已，就是選 u 但不選 v 要花費 ∞，因此建 $u \to v$ 流量是 ∞ 的邊即可。
3. 選 u 要花費 c: 這表示 $u \in S$，因此建 $u \to t$ 流量是 c 的邊即可。
4. 選 u 會賺到 c: 這個比較麻煩，因爲沒有負花費，也就是負容量這種東西。不過我們可以換個角度想，變成「沒有選到 u 會失去賺 c 的機會」，因此我們先把所有選了會賺的點 u_i 都找出來，假設會賺 c_i，計算 $C = \sum c_i$，表示你全部有機會賺到的量，之後就建 $s \to u$，流量爲 c 的邊，表示你一但不選就要失去這個 c_i 了。最後求完最小割在把 C 加上去即可。

剛剛那題的解法就很清楚了，先計算所有產品的總獲益 C，對每個產品，依照 4.，建從 s 流量等於其獲利的邊。再依 3. 對每個機器建一條到 t 流量爲其花費的邊。最後有些產品需要某些機器，也就是「選了這個產品就一定要選一些機器」，由 2.，建一些無限大的邊即可。

1.5 最小花費最大流

我們將原本的最大流問題做一個推廣，現在每個邊除了有容量限制 c 以外，還有一個花費 $k(u,v)$ ，表示每單位流量流過要花 $k(u,v)$ 的價格。也就是說對於一個流 f，這個流的總花費爲

$$k(f) = \sum_{f(u,v)>0} k(u,v) \cdot f(u,v)$$

最小花費最大流就是在滿足一個流 f 是圖上的最大流之下，使 f 的花費越小越好。[2]

1.5.1 最小花費最大流演算法

我們先定義一個路徑的花費。

定義： 一個路徑 P 的花費 $k(P)$ 就是其所有邊的花費總合。

現在我們來思考最小花費最大流的剩餘網路應該如何定出，首先，因爲最小花費最大流也是一個最大流，所以一個邊是否在剩餘網路裡的條件應該不變，也就是

$$(u,v) \in E(G_f) \Leftrightarrow r(u,v) > 0.$$

那剩餘網路的花費呢? 如果 $(u,v) \in E(G)$，那當然在剩餘網路中 $k_{G_f}(u,v) = k_G(u,v)$。但 $(u,v) \notin E(G)$ 呢? 注意到在剩餘網路上流過 (u,v) x 單位就相當於在原圖上少流 (v,u) x 單位，也就是花費少了 $k(u,v) \cdot x$，因此應設 $k_{G_f}(u,v) = -k_G(v,u)$。

構造完剩餘網路後，我們應該與最大流類似，在上面擴充就可以得到答案了! 最小花費最大流有以下的性質:

定理： 一個最大流 f 是一個最小花費最大流若且唯若其剩餘網路上沒有**負環**，也就是花費爲負的環。

而且還有一個很重要的性質

定理： 如果在剩餘網路上沒有負環，現在我們找一個花費最小的路徑擴充，則擴充後剩餘網路上還是不會有負環。

[2]這裡爲了證明方便我們都假設 G 是有向圖，且 $(u,v) \in E(G) \Rightarrow (v,u) \notin E(G)$。注意到無向圖的最小花費最大流仍然是可以做的，且實作上差異不大，只是在書寫時證明會有點麻煩!

因此我們就立刻可以得出以下的一個演算法!

Algorithm 3: Successive shortest path algorithm

```
Algorithm SSP()
  while We could find an minimum cost augmenting P path from s to t do
    Δf ← AugmentAlongPath(P)
    f ← f + Δf
  end
```

至於「找一個花費最小的路徑」，可以用任何最短路演算法來求，但要注意圖上可能會有負權邊，因此必須要使用能處理負權邊的演算法 (如 SPFA 等等)。而這個演算法的整體複雜度，因爲花費最小的路徑顯然不一定是最短 [3] 的擴充路徑，所以跟 Ford-Fulkerson 一樣，擴充的次數沒有被保證! 在邊權都是整數的情況下複雜度爲 $\mathcal{O}(SP \cdot |f|)$ ，其中 $\mathcal{O}(SP)$ 是一次最短路演算法的複雜度。

以下是一個範例的程式碼:

Dinic 程式碼片段 1-2

```cpp
template<typename TF, typename TC>
struct CostFlow {
    typedef pair<TF, TC> pll;
    static const int MXN = 205;
    static const TC  INF = 1e16; //Assume TC = long long
    struct Edge {
        int v, r;
        TF f;
        TC c;
    };
    int n, s, t, prv[MXN], prvE[MXN], inq[MXN];
    TF fl;
    TC dis[MXN], cost;
    vector<Edge> E[MXN];
    void init(int _n, int _s, int _t) {
        n = _n; s = _s; t = _t;
        for (int i=0; i<n; i++) E[i].clear();
        fl = cost = 0;
    }
    void add_edge(int u, int v, TF f, TC c) {
        E[u].PB({v, SZ(E[v])  , f,  c});
        E[v].PB({u, SZ(E[u])-1, 0, -c});
    }

    pll flow() {
        while (true) {
            for (int i=0; i<n; i++) {
```

[3]這邊指邊的數量最少

```
28                  dis[i] = INF;                                          ▲
29                  inq[i] = 0;
30              }
31              dis[s] = 0;
32              queue<int> que;
33              que.push(s);
34              while (!que.empty()) {
35                  int u = que.front(); que.pop();
36                  inq[u] = 0;
37                  for (int i=0; i<SZ(E[u]); i++) {
38                      int v = E[u][i].v;
39                      TC  w = E[u][i].c;
40                      if (E[u][i].f > 0 && dis[v] > dis[u] + w
                            ) {
41                          prv[v] = u; prvE[v] = i;
42                          dis[v] = dis[u] + w;
43                          if (!inq[v]) {
44                              inq[v] = 1;
45                              que.push(v);
46                          }
47                      }
48                  }
49              }
50              if (dis[t] == INF) break;
51              TF tf = INF;
52              for (int v=t, u, l; v!=s; v=u) {
53                  u=prv[v]; l=prvE[v];
54                  tf = min(tf, E[u][l].f);
55              }
56              for (int v=t, u, l; v!=s; v=u) {
57                  u=prv[v]; l=prvE[v];
58                  E[u][l].f -= tf;
59                  E[v][E[u][l].r].f += tf;
60              }
61              cost += tf * dis[t];
62              fl += tf;
63          }
64          return {fl, cost};
65      }
66  };
```

最小花費最大流演算法的證明 *

引理 一個流量爲 0 的流一定可以分解成不超過 E 個**環流**。
一個**環流**是一個環，且上面的邊的流量都一樣。

Proof. 流量爲 0 表示所有的點，包括 s,t 的淨流量都等於 0。我們找一個 $f^-(u_1) > 0$，也

就是出流量不等於 0 的點開始，因爲 $f^-(u_1) > 0$，所以我們一定可以找到一個點 u_2 ，使得 $f(u_1, u_2) > 0$。這表示 $f^+(u_2) > 0$，但因淨流量爲 0，$f^+(u_2) = f^-(u_2)$，所以 $f^-(u_2) > 0$，於是我們又可以找到 u_3 使得 $f(u_2, u_3) > 0$，依此規則找出 $u_1, u_2, u_3, \cdots$，因點有限個，所以一定會有 i, j 滿足 $u_i = u_j$，那 $u_i, u_{i+1}, \cdots, u_j = u_i$ 就是一個環了，而我們可以讓這個環流的大小爲裡面流量最小的一個。將這個環流從原本的流扣除後，流量最小的那個邊就消失了。因此每次進行這個操作，就會有一條邊消失，最多 E 次就做完了，因此可以分解成不超過 E 個環流。 □

定理 一個最大流 f 是一個最小花費最大流若且唯若其剩餘網路上沒有**負環**，也就是花費爲負的環。

Proof. 如果 G_f 有負環，那加上這個負環流後總花費更小了，因此最小花費最大流的剩餘網路一定沒有負環，我們只需要證明如果 f 不是最小花費的最大流，那他一定有負環。

假設 f_1 是最小花費最大流，f_2 是一個花費較大的最大流。考慮 $f = f_1 - f_2$，注意到如果 $f(u, v) > 0 \Rightarrow (u, v) \in E(G_{f_2})$，因爲

$$r_{f_2}(u, v) = c(u, v) - f_2(u, v) = c(u, v) - f_1(u, v) + f(u, v) \geq 0 + f(u, v) > 0$$

再注意到有 $|f| = |f_1| - |f_2| = 0$ 和 $k(f) = k(f_1) - k(f_2) < 0$，由剛剛的引理可知，$f$ 可以拆成有限個的環，且這些環的花費和 $\sum k(P_i) = k(f) < 0$，因此必有一個環 P_m 滿足 $k(P_i) < 0$，而我們知道如果 $(u, v) \in P_m$ ，則 $(u, v) \in E(G_{f_2})$，因此 G_{f_2} 一定有負環。 □

定理 如果在剩餘網路上沒有負環，現在我們找一個花費最小的路徑擴充，則擴充後剩餘網路上還是不會有負環。

Proof. 現在假設我們找了一個花費最小的路徑擴充 P ，但擴充後剩餘網路上有負環 C 了。不過原來的剩餘網路上沒有負環，所以一定有某些 (u, v) 滿足 $(v, u) \in P$。令這些擴充後才出現在 C 上的邊的集合叫作 Q，則

$$k(P) + k(C) = \sum_{(u,v)\in P,\ (v,u)\notin Q} k(v, u) + \sum_{(u,v)\in C\ (u,v)\notin Q} k(v, u)$$

因爲在 Q 裡的邊，他的反邊滿足 $k(u, v) = -k(v, u)$，剛好消掉了。而 $\{(u, v) \in P,\ (v, u) \notin Q\} \cup (u, v) \in C\ (u, v) \notin Q\}$ 其實就是把 P, C 裡的邊都看作無向邊後，他們的對稱差 D。由引理可以知道 D 可以分解成一個 $s \rightsquigarrow t$ 的路徑 P' 和一些環 C'_i。注意到 C'_i 裡的邊都存在於原本的剩餘網路，因爲 $C'_i \cap Q = \varnothing$。所以他要嘛是本來 P 上的邊，要嘛是 C 裡面不是因擴充後才出現的邊。而原本的剩於網路沒有負環，所以 $k(C'_i) \geq 0$，有

$$k(P) > k(P) + k(C) = k(D) = k(P') + k(C'_i) \geq k(P') \ \Rightarrow\ k(P') < k(P) \quad \Rightarrow\Leftarrow$$

因此擴充之後仍不會有負環。 □

1.6 最小花費最大流的應用

有了求最小花費最大流後，不少原來的題目都可以做一個推擴。

例題 1-7 二分圖最小花費最大匹配 經典問題

給你一個邊有權值的二分圖，求他的最小花費最大匹配。一個匹配的花費就是所有邊的權值的和。

這個問題就只是把本來的二分圖最大匹配用最小花費最大流下去做就可以了。這邊就不在多述。

例題 1-8 Machine Programming Codeforces 164C

你有 n 個工作可以分配給 k 個機器人，每個工作只能給一個機器人做，而且第 i 個工作必須從 s_i 做到 $s_i + t_i - 1$ ，不能中斷，做完了這個工作可以得到 c_i 元。問你最多可以賺到多少錢?

如果我們把每一天看做是一個點，且每個機器人對應到一單位的流量，那答案就呼之欲出了。假設每一天對應到 $v_1, v_2, \ldots, v_m$ ，那對於工作 i ，我們就建一條 $v_{s_i} \to v_{s_i+t_i-1}$ 的邊，其流量爲 1 ，也就是只能一個人來做，並且他的花費是 $-c_i$，表示如果做了這件事情就會賺 c_i 元。之後每一天都建一個無限大流量，花費爲 0 的邊到下一天，表示機器人也可以什麼事都不做到下一天。之後建 $s \to v_1, v_m \to t$ 流量各爲 1 的邊，最後求最小花費最大流即可。

1.7 額外的問題

這邊我們在舉一些和最大流有關的一些有趣的問題。

1.7.1 最大密度子圖

例題 1-9 最大密度子圖 經典問題

給你一個圖 $G = (V, E)$ ，你要求一個 G 的導出子圖 $H = (V', E')$ ，使得他的密度 $\rho = |E'|/|V'|$ 最大。

通常對於這種要最大化一個分數的問題，第一個直覺就是二分搜答案。假設我們當前猜答案

爲 k，也就是我們變成在問，是否有一個導出子圖滿足

$$\frac{|E'|}{|V'|} \geq k \quad \Leftrightarrow \quad |E'| \geq k|V'| \quad \Leftrightarrow \quad |E'| - k|V'| \geq 0$$

這個等價於一個求最大值的問題

$$\max_{H=(V',E')} |E'| - k|V'| \geq 0 \tag{1.4}$$

注意到

$$|E'| = \frac{1}{2} \sum_{v \in V'} \deg_H(v)$$

而

$$\deg_H(v) = \deg_G(v) - \sum_{\substack{u \notin H \\ (v,u) \in E}} 1$$

代回 (1.4)

$$\max_{H=(V',E')} |E'| - k|V'| \geq 0 \Leftrightarrow \max_{H=(V',E')} \sum_{v \in V'} \frac{1}{2} \left(\deg_G(v) - \sum_{\substack{u \notin H \\ (v,u) \in E}} 1 \right) - k$$

$$\Leftrightarrow - \min_{H=(V',E')} \sum_{v \in V'} \left((2k - \deg_G(v)) + \sum_{\substack{u \notin H \\ (v,u) \in E}} 1 \right)$$

有沒有發現這個變成了「分兩類問題」了? 對於每一個點，選他的 cost 就是 $2k - \deg(v)$，而如果 $(v, u) \in E$，那麼選了 v 卻不選 v 要花額外 1 的 cost。轉化成最小割後用最大流解即可。

1.7.2 最少互斥路徑覆蓋

例題 1-10 Minimum disjoint path covering 經典問題

給你一個有向圖 $G = (V, E)$，你要用最少不相交的簡單路徑 P_i 把所有的點都蓋過。一個簡單路徑是一個點都沒有重複的路徑。兩個路徑不相交表示他們裡面沒有重複的點。

一開始想這個問題可能會想把每個路徑對應到一個流。可是我們要最小化路徑數量，與「最大」流相悖。因此我們換個想法，先想一下不相交的路徑是什麼意思，其實就是每個點都只能恰屬於一條路徑而已，而對一個路徑 $v_1, v_2, \cdots, v_n$ 可以發現，每一個元素都有一個後繼，如 $\texttt{next}(v_1) = v_2, \texttt{next}(v_2) = v_3, \ldots$，除了最後一個點以外。因此我們要做的事情其實

就是把所有的點「配到」他的後繼 $\texttt{next}(v)$，而路徑的數量就是沒有被配到的點數，所以要最少的路徑就是要匹配最多！現在做法就很明瞭了。對於每個點 v_i，構造 u_i, w_i，而如果 $(v_i, v_j) \in E$ ，就建造 $u_i \to v_j$ 的邊，最後就是要求二分圖的最大匹配!

1.7.3 下界流

例題 1-11 下界流 經典問題

現在網路流上的每個邊不只有流量上限 $c^+(u,v)$，還有流量下界 $c^-(u,v)$，求一個合法的流 (不需最大)。

加了下界限制後，好像本來的最大流問題都不能用了，好像無從著手。

不妨這樣想，先把每條邊的容量都補成其流量下界，那現在每個邊就只能在流 $c^+(u,v) - c^-(u,v)$ 了，相當於新的「邊容量限制」。但這樣一開始點的流量守恆就不一定會滿足了，有些點流了太多進去，要「流一點出來」。有些則是要「流一點進去」。不知道有沒有聯想到「混合圖的歐拉迴路」那一題了？其實原理一模一樣！對於每個點，計算

$$\epsilon(u) = \sum c^-(u,v)$$

如果 $\epsilon(u) > 0$，表示補完流量後流出去太多了，需要流一點進來，因此新增兩個點 s', t'，建 (u, t') 流量爲 $\epsilon(u)$ 的邊，強迫他要流一些進去 (然後這些流量就會流到 t。)。而如果 $\epsilon(u) < 0$ 就建 (s', u) 流量爲 $-\epsilon(u)$ 的邊，最後求 s'-t' 的最大流即可。

1.8 習題

1.8.1 String Matching

習題 1-1 ABA 經典問題

給定字串 S，求出最長的 A 使得 $S = ABA$。[$\mathcal{O}(|S|)$]

習題 1-2 Periods of Words POI XIII

給定字串 S，求出最長的 A 使得 A 是 S 的前綴且 S 是 AA 的前綴。[$|S| \leq 10^6$]

習題 1-3 Template POI XII

給定字串 S，求出最短的 A 使得 A 可以覆蓋 S 。比如說 "abaabaab" 可以用兩個 "abaab" ，分別開頭在 $0, 3$ 蓋住。[$|S| \leq 10^6$]

習題 1-4 近似匹配 經典問題

給定字串 A, B，以及一個整數 k，求出所有 B 在 A 中 k 幾乎匹配的位置。我們稱 $A[i, i+n-1], B[j, j+n-1]$ 是 k 幾乎匹配如果 $\{A[i+x] \neq B[j+x],\ 0 \leq x \leq n-1\}$ 的個數不超過 k 個。[$\mathcal{O}(k|A| + |B|)$]

習題 1-5 k-口吃子字串 TIOJ 1735 / Kelvin

定義一個 k-口吃字串爲某一個長度爲 k 的字串重複兩次的字串。如 "abcabc", "aaaaaa" 都是 3-口吃字串。給一個字串 S 和 k，請問有多少 (i, j) 滿足 $S[i, j]$ 是 k-口吃字串。[$|S| \leq 10^5$]

習題 1-6 KMP and Z value 經典問題

給你一個字串的 fail function $\mathcal{F}$ 和他的長度 n (也就是說你並不知道原本的字串)，請求出他的 Z value Z。反之亦然，給你一個字串的 Z value Z，請求出他的 fail function $\mathcal{F}$。[$\mathcal{O}(n)$]

習題 1-7 Massacre at Camp Happy TIOJ 1725

定義兩個字串 A, B 是 k-幾乎相同如果把 A 的前 k 個字元搬到最後面，那兩者恰相差一個字元，給你 A, B，求出所有的 k 使得他們是 k-幾乎相同。[$|A| = |B| \leq 10^6$]

習題 1-8 最長回文子字串 經典問題

給一個字串 A，求出他最長的一個回文子字串。[$\mathcal{O}(|A|)$]

這題可以用類似 Z-algorithm 的方法。我們先考慮如何求最長的**奇數**回文子字串。令 $\mathcal{Z}(i)$ 表示以 i 爲中心最長的回文字串爲 $A[i - \mathcal{Z}(i), i + \mathcal{Z}(i)]$。現在假設 $(L, R) = (i - \mathcal{Z}(m), i + \mathcal{Z}(m))$, 即 $A[L, R]$ 是一個以 $a[m]$ 爲中心的回文。如果 $j \in [m+1, R]$，考慮 $j' = 2m - j$，即 j 對 m 的反射點。現在與 Z-algorithm 類似，令 $L' = j - \mathcal{Z}(j')$，分別考慮 $L' > L, L' = L, L' < L$ 三種情況。

但此方法僅能求得奇數的回文長度。因此我們把原字串相鄰的兩個字元都插入一個不在字元集 Σ 中的字元，如 $A =$ "abbab" $\rightarrow A' =$ "abbab"，這樣所有回文的長度都變成奇數了。

習題 1-9 Anti-hash text Codeforces Zlobober's blog

在 Hashing 實作上，因爲 C++ 的模運算 % 是非常花時間的，因此有一種 Lazy hash 的方法是直接 $\bmod 2^{32}$ 或是 $\bmod 2^{64}$，這樣直接在 `int`, `long long` 下做運算就可以了。但這樣做有其風險在，以下說明其原因。

不妨假設此 Hash function 爲

$$f(A) = \sum_{i=0}^{n-1} a_i p^{n-1-i} \bmod q$$

其中 p 爲一奇數 (否則此 Hash function 無意義)，$q = 2^m$，並且 "a", "b" 轉換成數字後

分別爲 $\alpha, \alpha+1$ 。此時令

$$A_0 = \text{"a"}, \quad A_j = \widetilde{A}_{j-1} A_{j-1}$$

其中 $\widetilde{A}$ 表示把字串 A 中的 "a" 換成 "b"，"b" 換成 "a"。如:

$$A_1 = \text{"ba"}, \quad A_2 = \text{"abba"}, \quad A_3 = \text{"baababba"}, \quad \cdots$$

現在考慮 $A_k, \widetilde{A}_k$，注意這兩個字串恰好是 A_{k+1} 的前後半且 $A_k \neq \widetilde{A}_k$。請證明

$$\begin{aligned} f(\widetilde{A}_k) - f(A_k) &= p^0 - p^1 - p^2 + p^3 - p^4 + p^5 + \cdots + (-1)^{k-1} p^{2^k-1} \\ &= \sum_{i=0}^{2^k-1} b_i p^i \quad = S \end{aligned}$$

其中 $b_i \in \{1, -1\}$，且如果 i 的二進位表示有偶數個 1 ，則 $b_i = 1$，否則 $b_i = 0$。並且有

$$\sum_{i=0}^{2^k-1} b_i p^i = (p-1)(p^2-1)(p^4-1)\cdots(p^{2^{k-1}}-1)$$

注意到 $(p^{2^r}-1)$ 可以被 2^{r+1} 整除 (Why?) ，所以 S 可以被 $2^{1+2+\cdots+k} = 2^{k(k+1)/2}$ 整除。因此只要 $2^{k(k+1)/2}$ 被 2^m 整除，也就是 $\frac{k(k+1)}{2} > m$，A_k 和 $\widetilde{A}_k$ 的 Hash value 就會相同。因此可以用一個長度爲 $2^{\mathcal{O}(x)}$ 的字串構造出一個使 $q = 2^{\mathcal{O}(x^2)}$ 的 Hash function 失效的反例，並且與 p 的選擇無關。

習題 1-10 二維匹配 經典問題

給你一個 $R_A \times C_A$ 的二維字串 A ，問一個 $R_B \times C_B$ 的二維字串 B 有沒有出現在其中? $[\,\mathcal{O}(R_A C_A + R_B C_B)\,]$。

一個巧妙的做法可參考 Baker-Bird Algorithm，令外也有用 Hash 的方法，令 $c = C_B$, $a_i[j] = f(A[i][j, j+c-1])$，原題等價於問 $f(B[0]), f(B[1]), \cdots f(B[R_B-1])$ 是不是等於
$a_i[j], a_{i+1}[j], \cdots, a_{i+R_B-1}[j]$。這是一個一維的匹配問題。詳細可參考去年的講義。

習題 1-11 sed ACM ICPC 2013-2014 NEERC Northern Subregional

給你 A, B，求出字串 X, Y 使得將 A 中的 X 取代爲 Y 後會變成 B，並且 $|X| + |Y|$ 最短。取代的定義爲找 X 出現在 A 的第一個位置，假設 $A[i, j] = X$ ，立刻將 $A[i, j]$ 變成 Y ，並從 $j+1$ 繼續找下一個 X 直到沒有爲止。$[\,|A|, |B| \leq 5000\,]$。

1.8.2 Suffix structure

習題 1-12 最小的後綴 經典問題

給一個字串 A，求出他字典序最小的後綴。[$\mathcal{O}(|A|)$]

有不需要建出 Suffix Array 的方法。[4]

這有一些應用，如最小循環表示法。有時候我們會把一個狀態或物體用一個字串表示，並且可能旋轉需視爲相同的，也就是把字串前 k 個字元接到後面視爲相同。這時候如果要比較兩個字串，可以先求他們的最小循環表示在進行比較。

習題 1-13 重複子字串 經典問題

給一個字串 A，求出他最長的一個重複的子字串，重複的部分可以重疊。比如說 $A =$ "cababababc"，最長的重複子字串爲 "aba"。[$\mathcal{O}(\mathcal{SA})$] [5]
如果改成不能重複呢？[$\mathcal{O}(|A|\log|A|)$]

習題 1-14 連續重複的子字串 經典問題

給一個字串 A，求出他的連續重複次數最多的子字串。比如說 $A =$ "cababababc"，"ab" 在子字串 "abababab" 中連續重複了 4 次。[$\mathcal{O}(|A|\log|A|)$]

習題 1-15 Binary Suffix Array ASC 40

給定一個 n 個整數的序列 $a_1, a_2, \cdots, a_n$，找出一個 01 字串 A 使得 A 的後綴數組 $\mathcal{SA}$ 恰好是給定的序列，注意到有可能無解。[$|A| \leq 3 \cdot 10^5$]

習題 1-16 Lexicographical Substring Search SPOJ 7258

給一個字串 A，現在把 A 的所有子字串列出，將重複的刪去後排序。對於 Q 比詢問，輸出字典序第 k 小的子字串。[$|A| \leq 90000, Q \leq 500$]

習題 1-17 Beautiful Substring NTUOJ 2200

給一個字串 A，把他所有不同的子字串找出來，令這個集合爲 S，接著每次會給一個字串 B_i，假設他所有不同的子字串的集合爲 T_i，就令 $S \leftarrow S \setminus T_i$，並輸出此時的 $|S|$，也就是 S 裡還剩多少字串。[$|A| \leq 2 \times 10^5,\ \sum|B_i| \leq 10^6$]

習題 1-18 Incomparable Suffixes ASC 42

給一個字串 A，我們說兩個字串 s, t 是**可分辨的** 若且爲若存在一個字串 z 使得 sz, tz 中恰好只有一個是 A 的後綴，說兩個字串 s, t 是**無可比較的** 若且爲若對所有 s 的前綴 x，t 的前綴 y，x, y 都是可分辨的。現在請找出 $a_1, a_2, \cdots, a_n$ 滿足：

1. 所有 a_i 都是 A 的後綴。
2. n 越大越好。
3. 在 n 是所有可能的最大值下，$\sum|a_i|$ 越大越好。

[$|A| \leq 10^5$]

[4] Hint: 雙指針
[5] $\mathcal{O}(\mathcal{SA})$ 表示建出後綴數組所需的時間。

1.9 Special Thanks

此分講義部分參考自前年的講義，特別感謝前年的編輯者。