講義 09 - DP2 , 合併演算法

VI

2012/12/17

1 DP 優化

在前面的章節我們提到了用動態規劃 (Dynamic Programming) 來解決各式各樣的問題，而在這裡我們要討論一些神奇的優化方式。

1.1 單調隊列優化

先回憶一下雙向佇列 (Deque) 這個資料結構，他可以支援兩種操作：

- 從兩端 push 新的元素進去。
- 從兩端 pop 元素出去。

可以知道 Deque 完全涵蓋了 Stack 和 Queue 的操作。
而 Deque 通常用來優化 DP 中取極值的動作，尤其是要查詢多個範圍的極值，且這些範圍滿足一定的單調性，我們將在下面一一討論。

1.1.1 $Ans_i = \min_{L_i \le j \le R_i} v_j \quad ; \quad L_i \le L_{i+1}, R_i \le R_{i+1}$

簡單來說我們要依序查詢序列的某些區間的最大值，而且這些序列的區間左界和右界都是遞增的。假設序列的長度為 N, 如果我們對於每個區間都直接掃過一遍，時間複雜度會到 $O(N^2)$。
但仔細觀察可以發現，如果存在兩個元素 v_k, v_h 使得 $v_k \ge v_h$ 且 $k < h \le R_i$，那麼對於第 i 筆之後的詢問，v_k 絕不可能是區間的最小值。因此我們可以維護 Deque 從前面到後面是遞增的，即加入元素時保持單調性而從後面 pop 出元素，而當要查詢的時候先從前端 pop 出過期的元素後，極值即為 Deque 中最前端的元素。

1.1.2 $Ans_i = \min_{L_i \le j \le i} v_j + (d_i - d_j) \quad ; \quad L_i \le L_{i+1}$

對於這個情況，如果我們令 $a_i = v_i - d_i$，原本的式子就可以改寫成

$$Ans_i = \min_{L_i \le j \le i} a_j + d_i$$

而因為對於同一個 i 來說 d_i 是固定的，這樣就轉化成第一種情況了。這個式子可以讓我們加速背包問題的 DP，回憶一下背包問題的 DP 式，假設一個物品的重量為 w，價值為 v 且有 c 個：

$$D(n,m) = \max_{0\le k\le c; 0\le m-wk} D(n-1, m-wk) + vk$$

這個式子可以改寫為

$$D(n, wi+r) = \max_{0\le j\le i} D(n-1, wj+r) + v(i-j)$$

可以看出 $D(n-1, wj+r)$ 即為 v_j，v_i 即為 d_i。這可以讓我們在 $O(WN)$ 的時間解決背包問題，其中 W 是背包的耐重，N 為物品的數量，是一個與同一個物品的數量 c 無關的算法。

1.1.3 $Ans_i = \max_{L_i\le j\le i} d_j t_i + v_j \quad ; \quad L_i \le L_{i+1}, d_i \le d_{i+1}, t_i \le t_{i+1}$

這個情況即是所謂的斜率優化，對於一個 j，我們可以把每一個 $d_j t_i + v_j$ 看做是的一條直線 $y = d_j x + v_j$，這些線的斜率都是遞增的，而在我們加入一條條的直線，維持凸包的單調性，也就是說我們在加入新的一條線 l 時，假設 Deque 中最後兩條線為 l_1, l_2，如果 l_1 超過 l_2 的 x 座標比 l 超過 l_2 的 x 座標大，那我們直接將 l_1 pop 出，而每次查詢從前面 pop 出過期或是已經被後面的線超過的元素。

1.2 四邊形優化

四邊形優化是一個非常恐怖的東西，當你對 DP 的熱愛程度爆表了可以考慮研究一下。首先我們先定義兩種 DP 問題。

1.2.1 1D/1D DP

這種式子的標準式

$$D_i = \min_{i\le k<i} D_j + w(j,i)$$

簡單來說就是 D_i 會它前面得出的答案 D_j 再加上一個轉移的函數 $w(i,j)$ 中的最小值。

1.2.2 1D/2D DP

標準式

$$D_{i,j} = \min_{0\le k<i} D_{i,k} + D_{k+1,j} + w(i,j) \quad ; \quad D_{i,i} = 0$$

這種 DP 式的意義大概是一個區間 $[i,j]$ 的答案會是將這個區間分成兩塊 $[i,k], [k+1,j]$ 的分法中的最小值，再加上一個與 k 無關的權。

1.2.3 四邊形單調性

我們將單調性分成兩種。

- 凹四邊形單調性
  如果對於任何 $a < b, c < d$ 且 $F(a,c) \leq F(b,c)$，就有 $F(a,d) \leq F(b,d)$。
- 凸四邊形單調性
  如果對於任何 $a < b, c < d$ 且 $F(a,c) \geq F(b,c)$，就有 $F(a,d) \geq F(b,d)$。

但直接驗證 F 滿足這個性質往往是不容易的，因此我們常用以下不等式檢驗。

- 凹四邊形不等式
  如果對於任何 $a < b, c < d$ 都有 $F(a,c) + F(b,d) \geq F(a,d) + F(b,c)$，則 F 滿足凹四邊形單調性。
- 凸四邊形不等式
  如果對於任何 $a < b, c < d$ 都有 $F(a,c) + F(b,d) \leq F(a,d) + F(b,c)$，則 F 滿足凸四邊形單調性。

更進一步的結果，我們只需要檢查 $F(i,j) + F(i+1,j+1)$ 和 $F(i+1,j) + F(i,j+1)$ 的大小關係即可。

1.2.4 1D/1D 凹性優化

回憶一下 1D/1D 的 DP 式：

$$D_i = \min_{i \leq k < i} D_j + w(i,j)$$

如果我們令 $F(j,i) = D_j + w(j,i)$，代表用 j 轉移 i 的花費，而 D_i 就相當於 $F(i,k)$ 中的最小值。

此時可以知道如果 $w(j,i)$ 滿足凹四邊形單調性，$F(j,i)$ 也會滿足凹四邊形單調性。代表對於 $i, i+1, j, j+1$ 來說，一旦 $F(j,i) \leq F(j+1,i)$，也就是說如果用 j 來轉移 i 較 $j+1$ 來轉移 i 好的話，那麼必有 $F(j,i+1) \leq F(j+1,i+1)$，即接下來用 j 轉移 $i+1$ 也會較用 $j+1$ 來的好！換句話說如果我們用 k_i 代表用哪一個 j 值來轉移 i 會得到最小的值，必有 $k_i \geq k_{i+1}$。

利用這個單調性，我們便可以用特殊的方法加速。我們使用 Stack 維護當前最佳解，對於 stack 裡的元素 s 我們紀錄 (L,R,p)，代表對於所有 $L \leq i \leq R$，我們用 $j = p$ 來轉移到 i 最佳。因此當我們要求 D_i 時只需先從 Stack pop 出過期的元素 (即 $R < i$)，之後把 Stack 的 top 元素計算 $F(p,i)$ 即可。而當我們要將 $j = i$ 加入 Stack 時，假設 Stack 的 top 元素為 s，有幾種情況：

1. $F(i, s.L) \geq F(s.p, s.L)$
   這代表用 p 轉移完勝 i，因此我們保留 s 在 Stack 中。
2. $F(i, s.R) \leq F(s.p, s.R)$
   與上面相反，這代表用 i 轉移完勝 p，因此我們直接將 s 給 pop 出。
3. $F(i, s.L) \leq F(s.p, s.L), F(i, s.R) \geq F(s.p, s.R)$
   這時候代表存在一個界線，使得在界線前 i 較 p 好，而在後面 p 較 i 好。由單調性我們可以使用二分搜找出這個界線 L'，並將 s 更新為 (L', R, p)。

最後我們再將 i 加入 Stack 中，即加入 $(i+1, s.L-1, i)$，便完成更新了。(當然有些時候我們根本無需加入，如 $s.L-1 < i+1$ 時。複雜度為 $O(N \lg N)$。

1.2.5 1D/1D 凸性優化

與凹性相反，我們用 k_i 代表用哪一個 j 值來轉移 i 會得到最小的值，此時則為 $k_i \le k_{i+1}$。這時候我們也用和剛才類似的方法優化，只不過我們將 Stack 改為 Deque，取值的時候先從前端 pop 出過期的元素，而加入新的值時，我們就從後邊刪除元素，最後一樣二分搜出確切位置。

1.2.6 2D/1D 凹性優化

2D/1D 凹性優化並沒有特別特殊的性質，我們可以考慮枚舉 i 後每次視為 1D/1D 的問題。

令 $F_i(j') = D_{i,i+j'}$，此時有

$$F_i(j') = \min_{0 \le k < j'} F_i(k) + u_i(k, j')$$

其中

$$u_i(k, j') = D_{i+k+1, j'-k-1} + w(i, i+j')$$

只要證明 $u_i(k, j')$ 有凹單調性，便可用前面所講的方法做到 $O(N^2 \lg N)$ 的複雜度。

1.2.7 2D/1D 凸性優化

相較於 2D/1D 凹性優化，2D/1D 凸性有更多的性質，因此有更好的優化方式。首先 2D/1D 凸性在驗證上也比較容易，我們有以下的引理：2D/1D 的 DP 式為

$$D_{i,j} = \min_{0 \le j < i} D_{i,k} + D_{k+1,j} + w(i,j) \quad ; \quad D_{i,i} = 0$$

如果 $w(i,j)$ 為凸單調性的且 $w(i,i+2) \ge \max(w(i,i+1), w(i+1,i+2))$，則 $D_{i,j}$ 也會符合凸單調性。

而且 2D/1D 凸性還有一個強大的性質：令 $K(i,j)$ 為使 $D_{i,j}$ 達到最小值的 k 值，即 k 為使 $D_{i,k} + D_{k+1,j} + w(i,j)$ 最小者，則有 $K(i,j-1) \le K(i,j) \le K(i+1,j)$。

也因為以上定理，我們 DP 時可以從 $j - i = 1, 2, \cdots$ 的順序開始 DP。當我們在求 $DP_{i,j}$ 時我們只需枚舉 $K(i,j-1)$ 到 $K(i+1,j)$ 即可，故對於所有 $j - i = c$ 的狀態，將他們都求出所需枚舉的狀態數只有：

$$\sum_{0 \le i < N-c} K(i+1,j) - K(i,j-1) = K(N-c, N) - K(0, c-1) + N - c = O(N)$$

故求出所有 DP 值只需要 $O(N^2)$。

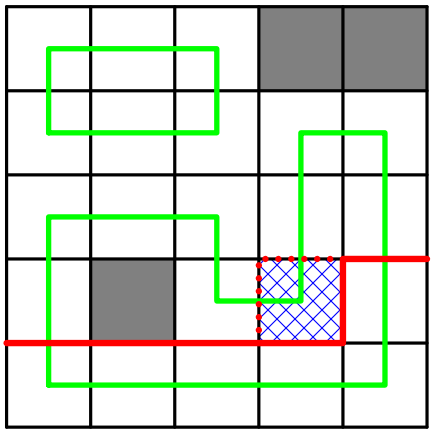
1.3 插頭 DP

根據某篇文章，他又稱作「基於連通性狀態壓縮的動態規劃」。

插頭 DP 通常用來處理 $m \times n$ 方格的一些組合計數，尤其是計算不同的某種特定路徑覆蓋的方法數。

就拿最基本的來舉例，假設我們有一個 $m \times n$ 的方格，有些方格可以通行有些不能，求用數個漢米頓圈覆蓋所有可以通行的方法樹有多少種？

而插頭 DP 的想法即是從右到左，從下到上依序討論每一個方格內路徑的樣式，即路徑要從方格的上下左右哪個相鄰方格進來/出去。雖然說我們要求漢米頓圈覆蓋覆蓋的方式，但我們可以想像成，我們先用路徑的方式討論，只要最後能把路徑「接起來」即可。



如圖所示，假設我們正在討論內部為交叉線條的那個方格，而我們 DP 的狀態便紀錄那條粗線的每個線段，是否有「插頭」，即是否有路徑通過那個位置，而當我們要遞推到下一格時，可以知道格子內路徑的樣式可以為那些只和格子右邊以及下邊的插頭有關，因此我們只需枚舉一些情況就可以了，而如果格子是不能通行的，那相當於右邊以及下邊都不能有插頭。

因此數個漢米頓圈覆蓋我們只需用 2 進位即可儲存狀態，但隨著題目越複雜，存的狀態可能也會越複雜，比如題目如果要求只用 1 個漢米頓圈覆蓋，那就需要用 3 進制儲存。詳細的話有興趣可以研究看看。

1.4 Exercise

1. 有 N 棟建築物，求出有多少個連續 M 棟建築的區間，使得區間內最高建築與最矮建築高度差不超過 k 公尺。(<TIOJ 1566> [簡單易懂的現代都市] , $N \leq 10^6$)
2. 求出一顆 N 個點的頂環樹上的最長路徑的長度。(<IOI 2008> [Island] , $N \leq 10^6$)
3. 給你一個序列 $x_1, x_2, \cdots, x_n$，你要將這個序列分成數塊，使得所有區塊的戰鬥力總和最大，假設一個區塊中所有數字的總合為 x，區塊的戰鬥力為 $Ax^2 + Bx + C$。(<APIO 2010> [Commando] , $N \leq 10^6; -5 \leq A \leq -1; |B|, |C| \leq 10^7; 1 \leq x_i \leq 100$)
4. 給你一個邊框，滿足 (1) 邊框可分為上邊框和下邊框。(2) 兩個邊框的起點為 $(0, 0)$，終點為 (x, y)。(3) 邊框是一條折線，即由 N 條水平線和鉛直線交替而成。(4) 上邊框永遠

在下邊框上面。請你找出這個邊框中長寬平行於 x, y 軸的最大矩形。(<TIOJ ???> [超大?? 設置] $N \leq 10^5$)

5. 給一條線上的 N 個點，請你選出其中的 k 個點，使得所有點到這 k 個中最短的距離的總和最小。(<TIOJ 1449> [郵局設置問題] $N, k \leq 1000$)

6. 現在有一堆烏龜堆疊成一堆，由下到上每隻烏龜都有為和度 x_i，現在你可以選一段連續的烏龜融合起來：

   (a) 最多只能把 R 隻烏龜融合成一段。

   (b) 把一堆烏龜模型融合起來，他們的違和度會相加。

   (c) k 個模型合出的模型會有強度 k^2 的違和光芒。

   (d) 一隻違和度 x 的烏龜放在第 m 層會有 $c * (m-1)$ 的違和度。

   (e) 烏龜塔的違和度是所有烏龜的違和度減掉所有違合光芒的強度。

   求出烏龜塔的違和度的最大值 (<TIOJ ???> [烏龜疊疊樂] $N, R \leq 5 \times 10^5$)

7. 有一個 $M \times N$ 的方格，有些格子可以通行有些則不能，求一筆畫不重複經過格子並通過所有可通行的方格，再回到起點的方法數有多少種。

8. 有一個 $M \times N$ 的方格，有些格子可以通行有些則不能，求一筆畫不重複經過格子並通過所有可通行的方格的方法數有多少種。

2 其他的主題

2.1 最近共同祖先 (LCA)

最近共同祖先 (LCA) 要求在一棵樹上的其中兩點，他們的一個共同祖先，使得這個祖先的深度最深，而詢問通常都是多筆的。假設樹上有 N 個點，詢問數有 Q 個，那如果最普通的做法會得到 $O(QN)$ 的時間複雜度，在 Q, N 都很大的時候我們顯然需要更好的方法。

2.1.1 Tarjan's Algorithm

在許多領域都有 Tarjan 的演算法，在這裡也不例外。

Tarjan's Algorithm 是一個離線的演算法，亦即必須先將所有詢問讀入才開始計算。其原理為：考慮一個節點 v，對於其兩個不同子節點 u, w，u 的某個子孫和 w 的某個子孫的 LCA 必為 v。因此我們可以對樹 DFS，當我們 DFS 完一個節點的某個子樹後，就把這些點和 v union 起來，這個動作讓我們想到可以使用並查集優化。詳細的演算法在下面，時間複雜度為 $O((Q+N)\alpha(N))$。

Algorithm 1 Tarjan's LCA

```
1:  function DFS(Current vertex : v)
2:      ancestor[find(v)] ← v
3:      for each child vertex u of v do
4:          DFS(u)
5:          Union(u,v)
6:          ancestor[find(v)] ← v
7:      end for
8:      visited[v] ← true
9:      for each query LCA(v, w) do
10:         if visited[w] = true then
11:             LCA(v, w) ← ancestor[find(w)]
12:         end if
13:     end for
14: end function
```

2.1.2 樹的壓平與 RMQ

這裡我們要介紹一個將樹壓平，變成一個序列的方法。在一開始序列為空，接著我們從樹根開始 DFS，每次進入一個節點時我們就將這個點加到序列的最後端，而每次出去一個節點的時候，我們把他的父節點加到序列的最後端。對於一棵有 N 個節點的樹我們便會得到一個長度為 $2N-1$ 的序列。而可以發現，對於任兩個節點 u,v，假設他在序列中出現的位置分別在 i,j(如果出現多次任取一個)，那麼 LCA(u,v) 便會是序列在範圍 $[i,j]$ 中深度最小的一個點。因此我們便將 LCA 轉化為 RMQ 問題了，用一般線段樹的做法可以得到一個 $O(N \lg N)$ 的作法，事實上因為序列中相鄰兩點的深度差最多是 1，因此這其實一個特殊的 $\pm$1RMQ 問題，可以用特殊的方法做到 $O(N)$ 的時間複雜度，如有興趣可以自行研究。

2.1.3 倍增法

倍增法式一個巧妙的方法，對於每一個點，記錄其第 $1, 2, \cdots, 2^k$ 祖先為哪個節點，這樣如果我們要求 LCA(u,v)，我們便可以先將 u,v 往上調整至同一深度，之後再用類似 2 分搜的方式求出他們最近的共同祖先。

2.2 比率二分搜

有時候我們要處理最優比率的問題，比如說要求 $\frac{f(v)}{g(v)}$ 的最大值，這時候因為有分數往往不好處理。因此通常我們會用二分搜來將題目改成判斷性問題，首先我們設定一個猜測值 t，如果有一組解不小於 t，那麼必有 $\frac{f(v)}{g(v)} \geq t$，這相當於存在一組解使得 $f(v) - tg(v) \geq 0$，注意到在每次二分蒐的過程中我們只需要判斷是否有解滿足條件即可，並不需要直接求出最佳解，而當猜測值等於極值的時候自然會給出一組最佳的答案。

2.3 疊代

與二分搜的想法不同，一步步的增加猜測值 g 來求得最佳解，這是因為有的時候我們可以直接用當前的最佳解 g 篩掉許多資料，檢查是否存在 g' 比 g 更好，如果行得通就可以壓掉一個 $\lg N$。

2.4 合併演算法

如果我們有兩個演算法 A, B，而他們演算法的複雜度取決於不同的條件，那往往可以將兩個演算法合併得到更佳的演算法。比如輸入為一組滿足 $\sum x_i = n$ 的序列，$\{x_1, x_2, \cdots, x_k\}$，如果 A 演算法的複雜度取決於數字的個數 k，B 演算法的複雜度取決於數字中的最大值 $\max x_i$，可以知道單獨兩個演算法的最差時間複雜度都是 $O(N)$，但這時候如果我們把不超過 q 的數字挑出來給演算法 B 處裡，A 處理剩下的數字，可以知道 A 處裡的數字不會超過 N/q 個，因此我們的時間複雜度為 $q + N/q$，取 $q = \sqrt{N}$ 即得到一個 $O(\sqrt{N})$ 的演算法。

2.5 Exercise

- 給你一個圖 $G = (V, E)$，每個邊 e 上有一個權值 $a(e)$，要維護兩種操作：
  - 求出一條從 u 到 v 的路徑，使得路徑上邊權最大的一條邊最小。
  - 將一條邊的權值改小。
- 給一個 $N \times M$ 矩陣，數字為 1 到 NM。求一個長寬為 H, W 的矩形，使得矩形內的中位數最小。(<IOI 2010> [Quality of Living] , $M, N \leq 3000; H \equiv W \equiv 1 \pmod 2$)
- 給你一個圖 $G = (V, E)$，每個邊 e 上有兩個權值 $a(e), b(e)$，求出一個生成樹 T 使得 $\frac{\sum_{e \in T} a(e)}{\sum_{e \in T} b(e)}$ 最小。([最優比率樹] , $O(|V|^2)$)
- 給你一個圖 $G = (V, E)$，每個邊 e 上有權值 $w(e)$。給一個點 v 和一個正整數 k，求出一個生成樹 T 使得 v 在生成樹上的度數恰為 k 且 T 的權重總和最小。
- 給你長度為 N 的 $0-1$ 序列，找一個長度不少於 K 的區間使得 1 占的比率最高。(<TIOJ 1670> [新聞採訪] , $N \leq 10^6$)
- 給你長度為 N 的序列 $a_1, a_2, \cdots, a_n$，對於 Q 筆詢問 (s, d, t) 求出 $a_s + a_{s+d} + a_{s+2d} + \cdots + a_{s+td}$。(<Codeforces 103D> [Time to Raid Cowavans] , $N, Q \leq 80000$)
- 給你一個整數 N，求出 $\sum_{1 \leq x \leq N} (N \bmod x)$。(<TIOJ 1674> [新專輯] , $N \leq 10^12$)
- 給你長度為 N 的序列，對於 Q 筆詢問求出範圍內逆序數隊的數量。(<TIOJ 1694> [你的重生之旅] , $N, Q \leq 80000$)
- 給你長度為 N 的序列，對於 Q 筆詢問 (a, b) 求出數列中數字 a 和數字 b 最近的一對的距離。(<TIOJ ???> [Dice War] , $N, Q \leq 80000$)
- 有 N 隻大象站在一條直線上，你有一個照相機可以將長度為 x 中的所有大象都照下來，對於 Q 筆尋問求出一動一隻大象後你需要照幾次才能將所有大象照下來。(<IOI 2010> [Elephant] , $N, Q \leq 10^5$)
- 一個公司中有 N 個人，他們之間的上司關係恰成為一棵樹，並且每一個人來自一個地區 v_i，對於 Q 筆詢問 (a, b)，求出有多少對人 (x, y) 使得 x 是 y 在樹上的某個祖先並且 x, y 分別來自 a, b。(<IOI 2009> [Regions] , $N, Q \leq 80000$)

3 Flow & Cut

在生活中，我們常常會碰到有關輸送的問題，比如說我們要從 A 輸送物資到 B，而兩地之間有許多個中繼站，中繼站之間可以輸送，但有流量的限制，要請問從 A 輸送物資到 B 最大的流量是多少以及如何輸送。這種問題同樣的會在通訊網路、世紀帝國、BC2..... 以及在資訊題目時出現，所以當然非會不可。

3.1 定義

首先我們先將題目建成一個圖論的模型。給一個網路流圖 $G = (V, E)$，可以是有向、無向甚至是混合圖 (但為了方便以下沒有特別說明我們都討論有向圖的情況)。圖中有兩個特別的點，源點 s 以及匯點 t，並且有個函數 $c : E \mapsto \mathbb{R}^+$，即每一條邊都有一個權重，通常我們將 c 看作邊的容量。接著我們定義兩種不同的問題。

3.1.1 Maximum Flow

求出一個函數 $f : E \mapsto \mathbb{R}$，我們將 f 看作是邊的實際流量，也就是你要分派每一條邊一個固定的流量，而且 f 要滿足：

1. 斜對稱性：如果 u 到 v 有一個大小為 x 的流量，那我們也可以看作 v 到 u 有一個大小為 $-x$ 的流量。

$$\forall (u, v) \in E, \quad f(u, v) = -f(v, u)$$

2. 容量限制：流量不能超過容量限制，並且如果 $(u, v) \notin E$，則定義 $c(u, v) = 0$。

$$\forall (u, v) \in E, \quad f(u, v) \leq c(u, v)$$

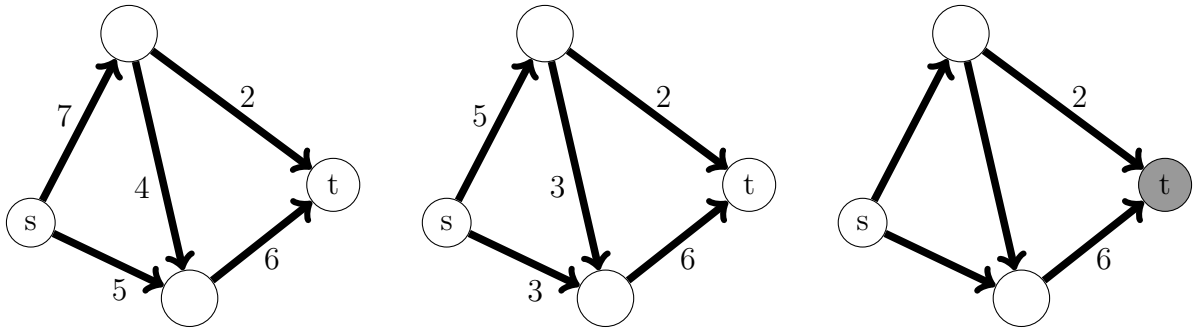
3. 流量守恆：除了源點只有流量出去，以及匯點只有流量進來以外，其他的點都要滿足進去的流量等於出去的流量。

$$\forall v \in E \setminus \{s, t\}, \quad \sum_{u \in V} f(v, u) = 0$$

且定義一個網路的流量 $|f| = \sum_{v \in V} f(v, t)$，也就是最後有多少流量流到匯點，Maximum Flow 即是要求一個 f 使得 $|f|$ 最大。

3.1.2 Minimum S-T Cut

接下來我們介紹另外一個完全不同的問題，S-T cut(S-T 割) 要將圖中的點分成兩個集合 S, T，使得 $S \cup T = V, S \cap T = \varnothing$，並且 $s \in S, t \in T$。接著定義 cut 的大小為 $\sum_{v \in S, u \in T} c(v, u)$，而最小割 (Minimum S-T cut) 即是要求 cut 大小的最小值。



3.1.3 Max-flow min-cut theorem

我們可以發現以上兩個問題有點像是兩個「相對」，但實質相同的問題，Max-flow 要求圖上的一個最大的流，而 min-cut 可以想成要求圖上的一個最小的瓶頸。事實上這兩個問題是線性規劃中的強對偶問題，Max-flow min-cut theorem 告訴我們 Max-flow 的大小恰等於 min-cut 的大小。

Question

- 給一個方法將無向圖和混和圖的流量網路轉化為有向圖流量網路處理。
- 請問一個 maximal flow 是否必為 maximum flow？
- 假設最大流的流量為 F，最小割的大小為 C。

  1. 證明 $F \leq C$。
  2. 證明 $C \leq F$。

3.2 Maximum flow Algorithm

解決最大流問題有許多有名的演算法，但基本上都是建立在剩餘流量網路上慢慢增加流量。

3.2.1 Residual network

首先我們定義一個網路流 $G = (V, E)$，他的剩餘流量網路 $G_f = (V, E_f)$，其中 $E_f = \{(u, v) : 0 < f(u, v) < c(u, v) \vee f(u, v) < 0\}$。
此外，定義邊 $(u, v) \in E_f$ 的剩餘容量 $c_f(u, v) = c(u, v) - f(u, v)$。
而如果當前的流量為 f，且 G_f 中存在一條從 s 到 t 的路徑，並且其中剩餘容量最小的一條邊為 $f' > 0$，那顯然我們可以將流量增加為 $f + f'$，我們把這條路徑稱作增廣路徑 (Augmenting path)。
Augmenting path theory 更告訴我們：一個網路流他達到最大流若且為若不存在一條增廣路徑。有了這個定理我們便可以求出最大流了。

3.2.2 Ford-Fulkerson's Algorithm

Ford-fulkerson's Algorithm 是 Augmenting path theory 立即的結果，即我們要求最大流，就從初始的剩餘網路開始不停的找任何一條增廣路徑增加流量，直到不存在增廣路徑為止。但這個演算法的時間複雜度最差為 $O(E|f|)$，即與最大流的值有關。一個最差的情況如下圖所示。也因此這個演算法一定要在容量限制皆為整數 (或是分數) 才可以運作，但事實上我們只要將每次的增廣路徑依照一定的方法尋找，就可以大大避免掉這些情況，因此有 Edmonds-Karp 演算法。

3.2.3 Edmonds-Karp's Algorithm

Edmonds-Karp's Algorithm 做的事情很簡單，只是將 Ford-fulkerson's Algorithm 中任意找一條增廣路徑改成每次都尋找最短的增廣路徑增廣，即可證明增廣次數不會超過 $O(VE)$，如果我們用 BFS 實作，則可以做到 $O(VE^2)$ 的時間複雜度。

3.2.4 Edmonds-Karp's Algorithm

Edmonds-Karp's Algorithm 做的事情很簡單，只是將 Ford-fulkerson's Algorithm 中任意找一條增廣路徑改成每次都尋找最短的增廣路徑增廣，即可證明增廣次數不會超過 $O(VE)$，如果我們用 BFS 實作，則可以做到 $O(VE^2)$ 的時間複雜度。

3.2.5 Improved Shortest Augmenting Path Algorithm

Edmonds-Karp's Algorithm 可以說是一個每次都找 Shortest Augmenting Path 來增廣的演算法，而 Ahuja 與 Orlin 更引入距離標號的概念，將之改進，複雜度為 $O(V^2E)$。

首先我們先定義距離標號，距離標號 d 給每一個點一個標號，代表這個點到匯點的距離下界，定義為 $d(t) = 0$ 且

$$d(u) \leq d(v) + 1, \quad \forall (u, v) \in E_f$$

顯然如果有一條從 s 到 t 的增廣路徑，那麼必有 $d(s) < |V|$，因此為了方便起見，我們定義那些在剩餘網路上無法連到匯點的點 v 的距離標號 $d(v) = |V|$。

接著我們定義可行弧，對於 $(u, v) \in E_f$，如果 $d(u) = d(v) + 1$，那我們就稱 (u, v) 為一條可行弧。而一條從 s 到 t、由可行弧組成的路徑必為一條最短增廣路徑，因此我們希望能不斷找由可行弧組成的增廣路徑來增廣，直到沒有增廣路徑為止。

但注意到 d 只是一個下界而已，甚至我們一開始只會將 d 初始化為 0 而已，因此我們要造出可行弧。定義兩種操作，Advance(v) 代表從 v 往下走一個可行弧，Relabel(v) 對 v 進行重新標號。

所以演算法要做的事情就很明確了，我們從源點 s 開始，假設現在到了點 v，先試著看能不能 Advance(v)，即往下走一個可行弧，如果不行我們就 Relabel(v)，而如果走到了 t，那就沿著走來的路徑增廣。一旦 $d(s) \geq |V|$ 那就確定我們已找到最大流了。

而這個演算法還有幾個重要的優化，如果在一次 Relabel 的時候我們發現距離標號 d 的值不連續，也就是說存在 $x_1 < x_2 < x_3$，使得有點的 $d(v) = x_1, d(u) = x_3$，但是沒有任何點的距

離標號為 x_2，這時候必定達到最大流了。

其次是 Relabel 的時候直接把 $d(v)$ 加一會比真的去取最小值還快一些。

Algorithm 2 Improved Shortest Augmenting Path Algorithm

```
1: function ADVANCE(v)
2:     if ∃u ⇒ (v, u) is an admissible arc then
3:         return u
4:     else
5:         return v
6:     end if
7: end function
8: function RELABEL(v)
9:     if ∃u ⇒ (v, u) ∈ E_f then
10:        d(v) ← min_{(v,u)∈E_f} d(u) + 1
11:    else
12:        d(v) ← |V|
13:    end if
14: end function
15: function *RELABEL(v)
16:    d(v) ← d(v) + 1
17:    if ∄u ⇒ d(u) = d(v) − 1 then
18:        return true
19:    end if
20:    return false
21: end function
22: function MAX_FLOW(Residual Flow network : G with source s and sink t)
23:    |f| ← 0
24:    d ← 0
25:    v ← s, π(s) ← s
26:    while d(s) < |V| do
27:        if ADVANCE(v) ≠ v then
28:            π(v) ← v
29:            v ← Advance(v)
30:            if v = t then
31:                |f| ← |f|+ AUGMENT(G,π)
32:            end if
33:        else
34:            if *RELABEL(v) = true then
35:                d(s) = |V|
36:            end if
37:        end if
38:    end while
39:    return |f|
40: end function
```

3.3 Minimum Cost Maximum Flow

假設我們現在流量網路的邊是有成本的，與流過邊的單位流量成正比。也就是說假設 e 的權重為 $c(e)$，如果有一條大小為 x 的流量流過，那我們便要負擔 $c(e)x$ 的成本，而一個流量網路的花費即是所有邊的花費總和。
最小花費最大流即是要求在流量網路上的一個最大流，且其花費最小。

3.3.1 Successive Shortest Path Algorithm

神奇的是，我們只要證明每次都拿剩餘網路中邊權總和最小的一條路徑增廣，那演算法結束後我們便會得到最小花費最大流。注意到這種題目通常會有負邊，所以必須用可以處理負權的演算法，如 SPFA。

但這樣還是沒辦法處理負環的情況，因此必須預處理，將圖中所有的負權環預先流滿，而在接下來便可以證明，如果一開始沒有負環，那在拿邊權總和最小的路徑增廣後也不會有負環。

3.4 最小割

如果要求一個最小割，我們只要先找出流量網路的一個最大流，最後在剩餘網路上從 s 開始可以走到的點的集合為 S，其他不可走到的點為 T，則 S, T 即為一個最小割。

3.5 建構模型

在上面我們會了解決最大流的演算法，現在我們要做的事就是把題目建模，成為網路流的問題。

3.5.1 基本流量建模

- 點有容量
  將點 v 拆成兩個點 v, v' 然後中間連一條限制容量的邊。
- 邊的容量沒有限制
  通常用很大的一個數字代替，只要保證操作不會溢位就可以。
- 要求整數解
  由 Augmenting path theory 可以知道只要邊是整數，那一定存在一組邊流量皆為整數的解，即使是最小花費最大流也一樣。
- 並不要求流量流滿，但少流一單位會有代價
  加一條成本為 c 且流量為無限大的邊。

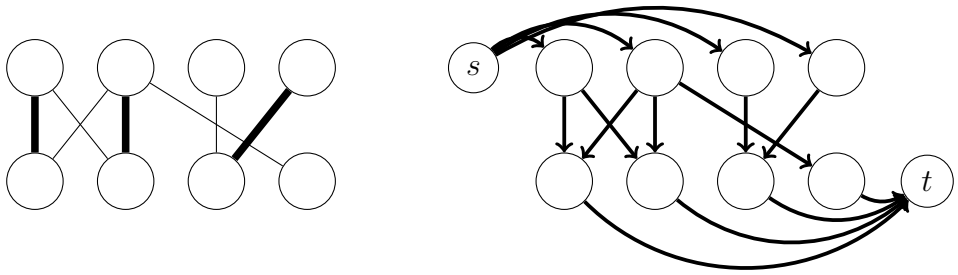
3.5.2 二分圖模型

回憶一下一個圖 $G = (V, E)$ 如果是二分圖，那就代表我們可以將點分成兩個集合 A, B 使得 $A \cap B = \varnothing$ 且任何邊 (u, v) 都滿足 $u \in A, v \in B$。

很多 NP 問題到了二分圖上都會有不錯的時間複雜度的解。

二分圖匹配是要求找出一個邊的子集 M，滿足其中的邊都不相鄰，並且我們希望邊的數量越多越好。可以想成我們用 M 中的邊將圖中的某些點兩兩匹配，並且一個點至多只能匹配另一個點。

而最大二分圖匹配可以很簡單的轉化為最大流模型，我們從源點把所有 $u \in A$ 都連一條容量為 1 的邊 (s, u)，把所有 $v \in B$ 都連一條容量為 1 的邊 (v, t) 到匯點，最後如果原圖有一條 $(u, v), u \in A, v \in B$ 的邊，那就在流量網路上加一條容量為 1 的邊 (u, v)，此時容易知道 s 到 t 的最大流即為二分圖的最大匹配數。



如果邊加上了花費 $-v(e)$，那我們可以想成選兩個點做配對會有價值 $v(u, v)$，要求價值最大的一組匹配，這可以轉化為最小花費最大流求解。而關於二分圖還有兩個重要的問題，最小點覆蓋還有最大獨立點集，這兩個問題恰好是相對的。

二分圖最小點覆蓋要求二分圖上的一個點的子集 C 使得所有的邊都與 C 中的某個點相鄰，而且希望 $|C|$ 越小越好，而最大獨立點集 I 則是希望能找到二分圖上的一個點的子集 I 使得 I 中任兩點都不相鄰。

一個巧妙的結果是，最小點覆蓋恰好會是最大獨立點集的補集，而且最小點覆蓋的大巧恰好會是最大匹配的大小。

而找一個最小點覆蓋的方法是這樣子的，先找出最大匹配後，如果沒有未匹配點，那隨便挑全部一側的點即會是一個點覆蓋，否則我們用以下方法將 V 分層。

- S_0 包含所有的未匹配點。(根據條件 $S_0 \neq \varnothing$)
- S_{2k-1} 包含所有與 S_{2k} 中其中一點相鄰，並且還不屬於前面任何一層的點。
- S_{2k} 包含所有與 S_{2k-1} 中其中一點以匹配邊相鄰，並且還不屬於前面任何一層的點。

- 如果 $S_{2k-1} = \varnothing$，任取一點丟進 S_{2k-1} 並繼續。

可以知道挑選出奇數層的點即可蓋住所有的邊，並且挑出的點不會超過最大匹配數。

Question

- 假設二分圖 $G = (V, E)$ 中 C 為任一個點覆蓋，證明 G/C 是一個獨立點集。
- 假設二分圖 $G = (V, E)$ 中 I 為任一個獨立點集，證明 G/I 是一個點覆蓋。
- 假設二分圖 $G = (V, E)$ 中 M 為一最大匹配，C 為一點覆蓋，證明 $|M| \leq |C|$。

3.5.3 下界流

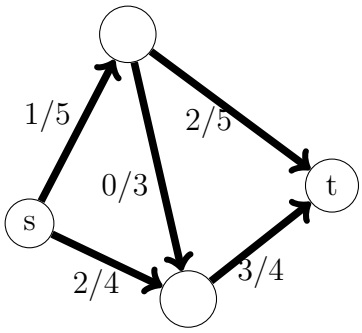
有時候我們會有流量下界的限制，即每條邊的流量除了容量為其上限以外，還不能低於一個值 $h(e)$。

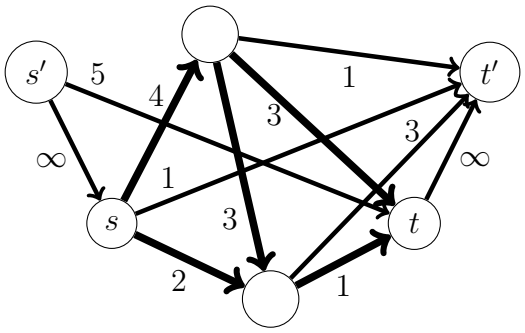
而要求一個滿足條件限制的下界流，想法是先把所有的邊都給一個等同於其下界的流量，這樣會使得一開始某些點進去和出來的流量不合，我們在用新增的源點和匯點來供給/接收多餘的流量。具體的寫法為：
新建一個流量網路 $G' = (V', E')$，其中 s', t' 為新的源點和匯點，並且 $V' = V \cup \{s', t'\}$。
令

$$f^+(v) = \sum_{(u,v)\in E} h(e), \quad f^-(v) = \sum_{(v,u)\in E} h(e)$$

- 對所有點 v 加一條容量為 $f^+(v)$ 的邊 (s', v)。
- 對所有點 v 加一條容量為 $f^-(v)$ 的邊 (v, t')。
- 對於原本存在的邊 e，將其容量減去 $h(e)$ 後加入 E'。
- 加一條容量為 ∞ 的邊 (s', s)。
- 加一條容量為 ∞ 的邊 (t, t')。

此時求 s' 到 t' 的最大流，如果 $|f| = \sum_{v\in V} f^+(v) = \sum_{v\in V} f^-(v)$ 那麼下界流存在，否則不存在。
而找到了一個下界流後，如果要找最大下界流，在已是合理的下界流 G' 的 $V' \setminus \{s', t'\}$ 中以 s, t 為原點匯點找最大流即可。





3.5.4 最小割模型

有時候會碰到的題目是，要把東西分成兩個集合，而屬於不同的兩個集合會有花費，或著是把某個集合選到其中一個集合會有花費等等，這時候可能可以化成最小割來解決，列出一些常見的建模方法。

- 一個點 v 被選到 S 會有花費 c
  連一條容量為 c 的邊 (v,t)，選到 T 的狀況同理。
- 如果某一個點 v 要在 T 中，那 u 也必須在 T 中
  連一條容量為 ∞ 的邊 (u,v)。

3.6 Exercise

- 有 N 個事件，每個事件會在 t_i 時間 (x_i, y_i) 位置發生。現在你要派出一些鎢絲處理這些事件，鎢絲一開始在原點，並且單位時間只能水平或垂直移動一單位，且移動速度是一格/單位時間。對每個事件都必須要在該事件發生前有至少一個鎢絲到達，問最少需要幾個鎢絲。(<NPSC 2005 初賽 pE> [魔法部的任務] , $N \leq 1000$)
- 給你一個混合圖 $G = (V, E)$，輸出任何一組歐拉迴路。(<Uva 10989> [Euler Circuit] , $|V| \leq 200$)
- 給你一個有向圖 $G = (V, E)$，求出最少要用多少有向路徑才能覆蓋所有的點。
- 現在你有數字 $1, 2, \cdots, N$，你希望把這些數字分成許多數列 $A_1, A_2, \cdots, A_k$，滿足對於每個數列 $A_i = \{a_1, a_2, \cdots, a_{m_i} =\}$ 來說都有 $a_i < a_{i+1}$ 且 $a_i + a_{i+1}$ 是質數，問你至少要分成多少數列。
- 在一個 $M \times N$ 的方格上，有些格子上有敵人。每次可以選擇炸一整行或一整列上的敵人，問至少要炸幾次。(<Uva 11419> [SAM I AM] , $M, N \leq 200$)
- 一間有 N 個員工的公司決定要裁員，每個人都有一個可正可負的貢獻值，可是也有許多影響關係：A 對 B 有影響的話，A 被裁掉 B 也會走人，問留下來的員工貢獻值和最大為多少。(<99 建中校內賽 p4> [房屋裁員] , $N \leq 500$)
- 在一張 $M \times N$ 的方格上，有些格子有障礙物。現在要用兩種蛇去把這張圖所有空格覆蓋恰好一次：1. 一個頭和尾接起來的蛇或是 2. 頭尾都貼在地圖邊界的蛇，問最少要幾種 2 型蛇才能達成目標。(<NPSC 2009 決賽 pF> [飛機上有蛇] , $N, M \leq 30$)
- 給一個序列 A 和 K，選出 K 條沒有重複數字的嚴格遞增子序列，求 K 條序列長度和最大值。(<NPSC 2009 決賽 pB> [多條嚴格遞增序列] , $|A|, |K| \leq 1000$)
- 數線上有 N 個點，現在要連其中 K 個點對，一個點最多連到一條線，求一個最短的連法。(<APIO 2007> [Backup] , $N \leq 10000$)
- 給你一個圖 $G = (V, E)$，求一個導出子圖 $G' = (V', E')$ 使其密度 $\frac{|E'|}{|V'|}$ 最大。

- 有 N 個儀器還有 M 種實驗，買每個儀器有花費 c_i，而做完每個實驗會有收入 b_i，但要做一個實驗就必須買特定的儀器。問你最多可以賺多少錢。($N, M \leq 1000$)
- 在一張 $M \times N$ 的方格上，有些格子有障礙物。現在 A,B 玩一個玩命遊戲，由 A 決定一個起點，之後由 B 先開始輪流開車，每次把車子開到上下左右相鄰的方格，並且走過的方格就會被放置一個地雷，誰被撞死或是炸死就輸了。問 A 選那些點可以獲勝。(<tioj ???> [棋盤策略遊戲] , $N, M \leq 300$)
- 現在有尺寸為 $[1, N]$ 的溜冰鞋各 k 雙，每個足尺寸為 r 的人可以穿尺寸界在 $[r, r + d]$ 的溜冰鞋，每個事件描述時間 i 有 r_i 個尺寸為 x_i 的人來或是離開，輸出每個事件後是否有方法分配溜冰鞋使得每個人都恰能分到一雙可穿的溜冰鞋。(<POI XVI> [Ice Skates] , $O(N \leq 10^5)$)