基礎演算法 2

baluteshih

2019 年 10 月 2 日

1 分治法 (Divide and Conquer)

「由大化小，分而治之。」分治法就是將原問題分成若干個規模較小的子問題 (「分」)，再將子問題的答案統整起來解決原問題 (「治」)。

1.1 分治法的思路

大致上可以分為以下幾項：

1. 直觀的把規模最小的問題 (基底) 做掉。
2. 將問題拆成多個小問題，遞迴下去。(分)
3. 相信遞迴完的結果是對的，並執行合併。(治)

分治法給了許多問題更好的解決方法，也有許多演算法是基於分治的底子出身的。由於他的概念有點模糊，所以接下來將會以例題出發。

1.2 合併排序法 (Merge Sort)

這是一個利用分治法來排序的做法，應用也非常廣，推薦各位一定要把這個學起來。不如我們就拿前面提到的思路來現學現賣好了：

1. 直觀的把規模最小的問題 (基底) 做掉。
   當數字只有一個的時候，就是排序好的序列 (注意其實兩個數字的問題規模並不是最小)。
2. 將問題拆成多個小問題，遞迴下去。(分)
   不妨把序列拆成兩半，「假設」兩半都已經排序好了 (請不要理會遞迴後發生了什麼事情)。

3. 相信遞迴完的結果是對的，並執行合併。(治)
   使用或自己實作 std::merge！時間是線性的。

如果不是很懂可以去查查維基百科的 GIF 動圖，配合著看非常好懂。
這樣我們就能保證分治法成功了。
那如何分析時間複雜度呢？我們可以這樣寫出複雜度的遞迴式。
假設時間複雜度為 $T(n)$，則：

$$T(n) = 2T(\frac{n}{2}) + O(n)$$

這種東西看起來有點麻煩，其實實際上我們也能用另一種想法去證明 merge sort 的複雜度。
但是大部分的分治法都會出現這個問題，也不是每次都能用別的想法去計算。
於是有了一個好用的工具，我們稱之為主定理。

1.3 主定理 (Master Theorem)

整個主定理的敘述其實有些複雜，但畢竟我們是競賽程式，還不需要知道太多。
只要知道以下就好了：
如果 a 是正整數，$b > 1, k \geq 0$，而且

$$T(n) = aT(\frac{n}{b}) + O(n^\alpha \text{log}^k n)$$

那麼有：

$$T(n) = \begin{cases} O(n^{\log_b a}) & \text{log}_b a > \alpha \\ O(n^\alpha \text{log}^k n) & \text{log}_b a < \alpha \\ O(n^\alpha \text{log}^{k+1} n) & \text{log}_b a = \alpha \end{cases}$$

拿 Merge sort 當例子，$a = 2, b = 2, k = 0, \alpha = 1$，$\text{log}_b a = 1 = \alpha$，所以 $T(n) = O(n \log n)$。
主定理的證明大概在大學的時候會被提及，各位有興趣也可以自己先研究看看。

主定理不一定是萬能的，在有些長相奇怪的分治法也是會失靈，但這種情況不多，各位還是可以多多利用他。

1.4 分治法的常數優化

由於遞迴本身的常數就很大，有時候在 n 不大的時候採用暴力法還會比較快。
如果很閒或需要的話，可以嘗試在子問題規模小的時候直接採用暴力法來解決問題 (也就是把最小規模的問題放大)，有可能可以達到較好的速度。
適當的選取規模暴力也是一項小問題，可以試著多試幾次大小 submit 看看之後評估。
更有趣的是，因為規模的選取通常會是 U 型函數，所以你可以手動利用 Judge 來三分搜。

1.5 分治法的經典用途

1. 快速排序法。
2. 逆序數對。(經典問題，搭配 Merge Sort)
3. Median of medians。(一個線性找中位數的算法，主定理在他身上會失效)
4. Karatsuba algorithm。(一個 $O(n^{\log_2 3})$ 的多項式乘法演算法)
5. 複雜度更好的矩陣乘法。($O(n^{\log_2 7})$)
6. 最近點對。(經典問題，搭配 Merge Sort)

1.6 習題

1. (TIOJ 1080) 對一個數列 S 來說，若 S 的第 i 項 s_i 與第 j 項 s_j 符合 $s_i > s_j$，並且 $i < j$ 的話，那麼我們說 (i, j) 是一個逆序數對。請問給定 S，總共有多少個逆序數對呢？(經典問題)
2. (TIOJ 1500) 最近點對問題。
3. (TIOJ 1208) 給你一個長度為 $N \leq 20000$ 的陣列。請找出 $\frac{n(n+1)}{2}$ 個區間中，總和第 k 大的區間，其區間和是多少？

2 動態規劃 (Dynamic Programming)

動態規劃（簡稱 DP）簡單來說就是以空間複雜度為代價，換取時間複雜度的優化。
方法就是將之後可能會用到的計算結果存起來，以後就不用再重新算一遍了。

2.1 DP 的理念及基本概念

DP 的想法在於，透過把問題分解成相似的子問題時，從中發現某些子問題會一直被重複計算，進而存起來省下不必要的時間。
其特性如下：

1. 重複子問題：相同的一個子問題，需要多次查詢。
2. 最佳子結構：當問題被拆成若干個規模較小的問題時，可以透過這些子問題得到這個問題的最佳解。
3. 無後效性：子問題不會互相呼叫，一定存在一個能完整求值的計算順序 (到了圖論我們可以講得更清楚)。

DP 可以說是基於分治法出現的進化概念，但其理念有時候不一定要從分治法切入，但我們可以用類似的歸納出 DP 主要會使用到的步驟：

1. 設置狀態。
2. 導出轉移。
3. 打好基底。

DP 通常會被定義成一個算出答案的函數，並不僅限於一對一的函數，DP 的函數多變，可以容納多種變數，其長相都是由自己決定的。
當然，在遇到題目時想到如何定義函數型態是很重要的事情，而這就是所謂的「設置狀態」。
在狀態定義出來之後，試著去思考如何利用遞迴子問題對應到的函數值組合出答案，就是所謂的「導出轉移」。
最後，和分治法一樣的，也必須確認好規模最小的子問題的答案是正確的，便是「打好基底」。
最後時間複雜度的計算，通常就會是「使用到的狀態個數」乘上「轉移時間」了。

2.2 線性遞迴

當一個函數 $f(n)$ 可以用一或多個 $f(i)$ 的常數倍轉移得到答案時，$f(n)$ 就稱為線性遞迴函數。
最經典的例子就是所謂的費氏數列，我們就直接套用前面提到的步驟吧。

1. 設置狀態。
   $dp[i]$ = 費氏數列的第 i 項。
2. 導出轉移。
   $dp[i] = dp[i-1] + dp[i-2]$
3. 打好基底。
   $dp[0] = 1, dp[1] = 1$

如此一來我們就完成了 dp 的前置作業，這時衝動的話可能就會寫出如以下的 code：

Algorithm 1: Fibonacci without DP

```
int f(int n){
    if(n==0||n==1) return 1;
    return f(n-1)+f(n-2);
}
```

這樣執行完剛開始可能還正常，但 n 才沒多大就會出事了 QQ。
還記得我們說過，DP 的想法在於把重複子問題的結果存起來省時間嗎？
沒錯，於是就有了以下的 code 出現了：

Algorithm 2: Fibonacci with DP, topdown version

```
int f(int n){
    if(n==0||n==1) return 1;
    if(dp[n]!=0) return dp[n];
    return dp[n]=f(n-1)+f(n-2);
}
```

可以發現，dp 陣列一開始預設成 0 的話，相當於每個函數值只要被計算一次，就不需要重新計算，每次呼叫都是 $O(1)$ 回傳了。
這樣子的時間複雜度就是好好的 $O(n)$，其便是 DP 的精髓所在。

2.2.1 button up

事實上撰寫 DP 時，我們可以不用撰寫遞迴函式，利用好的順序寫迴圈就能得到答案了。

Algorithm 3: Fibonacci with DP, button up version

```
dp[0]=dp[1]=1;
for(int i=2;i<=n;++i)
    dp[i]=dp[i-1]+dp[i-2];
```

只要確定好 DP 值被計算出來的順序是對的，就可以省略使用函數的操作。
其優點在於，由於一般遞迴的時間常數都不小，使用迴圈撰寫通常能省下不少時間。
但缺點也在於，定出一個好的順序有時候不太容易，其實一般用遞迴去思考可能會好寫許多。

2.2.2 習題

1. (ZJ d212) 費氏數列。
2. (ZJ d054) 給你一個 $2 \times n$ 的表格，你可以用 1×1 或 L 型三方塊來填滿它，試問有幾種方法數。
3. (TIOJ 1291) n 個相同的箱子要放入 m 個不同的球，問有幾種放法。
4. (2009 TOI 初選 pA/ZJ b229) 考慮在 $X-Y$ 平面上的整數格子點上建構長度為 N 的路徑。其中在格子點 (x,y) 時，路徑可以往右走到 $(x+1,y)$；或往左走到 $(x-1,y)$；或往上走到 $(x,y+1)$。長度為 N 的路徑必須經過 N 個相異的邊。試問由原點 $(0,0)$ 出發並按照上述規則所形成長度為 N 的路徑有幾條?
5. (ZJ d105) n 個同學圍成一圈，每次傳球，手上有球的人可以選擇要往左還往右傳，問 1 號同學傳了 m 次球後傳回自己手上的方法數。
6. (ZJ a697) 有 n 種花，標號 1 到 n。為了在門口展出更多種花，規定第 i 種花不能超過 a_i 盆，擺花時同一種花放在一起，且不同種類的花需按標號的從小到大的順序依次擺列，問擺了 m 盆花的方法數。
7. (105 全國賽 pB/TIOJ 1939) 題敘略。

2.3 前綴 DP

老實說筆者覺得這不太像 DP，不過大家都說這也是一種 DP 我只好放了 (X)。
理念在於 DP 的轉移若出現 min 或 max(DP 式的轉移方式是多元的)，若每次取極值的範圍不是一樣就是原本的範圍再更大，就可以省下重跑一次迴圈的動作，直接把值存起來轉移了。
常見在 $dp[i]$ 表示「以第 i 個位置為結尾的最佳答案」的狀態中。

2.3.1 習題

1. (ZJ d784) 已知一 n 個元素的整數數列，找出該數列連續元素的和的最大值。
2. (TIOJ 1277) 最大子矩陣和。
3. (105 全國賽 pA/TIOJ 1938) 題敘略。

2.4 區間 DP

常見的通常是，把狀態設成 $dp[l][r]$ 之後，用短的 DP 值來轉移出現在的答案。
這裡附上某題 ZJ 的題目思維供各位參考：ZJ d652

1. 設置狀態。
   $dp[l][r]=$ 區間 $[l,r]$ 吸收完後的最小汙染值 (會剩下最後側邊兩隻)。
2. 導出轉移。
   枚舉最後被 $a[l],a[r]$ 吸收掉的那隻糊。
   $dp[l][r]=min\{dp[l][k]+dp[k][r]+a[l]*a[k]*a[r]\}$
3. 打好基底。
   $dp[l][l]=0,dp[l][l+1]=0$

這題的時間複雜度約為 $O(n^3)$，同樣的思維也可以推廣到高維的 DP，所以區間 DP 並不是只侷限於序列而已。

2.4.1 習題

1. (ZJ d652) 例題 (其實這題也被稱為「最小矩陣鏈乘積」，是一題經典題，存在著 $O(n\log n)$ 的做法，但由於過度噁心，所以不提。)
2. (ZJ d686) 你有一些切割木棍的斷點，切割一個木棍的權重是木棍長度，請你決定好一個切割順序使得權重和最小。
3. (TIOJ 1388) 當你把兩坨史萊姆混合之後，將會產生一隻全新的史萊姆，體積則是原先兩隻的總合，而攻擊力也將重新計算。如果兩隻的體積合為奇數，新的攻擊力則為原本兩隻的乘積；若非則為兩隻的總合。現在這些史萊姆在你面前排成一排，你不能隨意更動牠們的順序，但是可以利用機器讓他們混合。那這些史萊姆可以產生的史萊姆攻擊力最強為多少呢？
4. (ZJ c263) 給你一個 $n\times m$ 的表格，每次可以選取一個格子，並以其為中心覆蓋一個十字，得到跟十字相關的權重，並將表格分為四個小表格後持續操作，試問全部格子都被覆蓋到後的最小權重和。
5. (104 全國賽 pD/TIOJ 1914) 給你一個空白紙條，每次可以對一個區間上色，問最少要上色幾次才能把紙條塗成想要的樣子。

2.5 某些離散的 DP 與經典概念

這類題目很雜，但也說不太上他們是怎樣的 DP，也沒有一個統一的方向。
以下會舉幾個經典的題目，並在其中導入一些概念。

2.5.1 0/1 背包問題

你有一個包包能裝 C 公斤的東西。有 N 樣物品，已知所有物品分別的重量 w 和價值 v。請問這個包包最多可以裝價值多高的物品？

這是一題再也經典不過的 DP 題，事實上他的理念對於初學者來說卻有些複雜，但我們一樣採取同樣的策略分析看看。

1. 設置狀態。
   $dp[n][i] =$ 使用 $1 \sim n$ 個物品湊出重量 i 時，所可得到的最大價值。
2. 導出轉移。
   $dp[n][i] = max(dp[n-1][i-w[n]] + v[n], dp[n-1][i])$
3. 打好基底。
   $dp[0][0] = 0, dp[0][i] = -INF \quad when \quad i > 0$

湊不出來的重量就要預設為負無限大，不過實作時通常只要做一些小判斷就不需要預設了。
這種怪異的狀態也是 DP 的常態之一，如果你夠熟 DP，各種怪異的狀態被定義出來通常都不是問題。
但這時我們就會發現背包問題的空間複雜度會是 $O(NC)$，似乎有辦法優化？

2.5.2 滾動 DP

當發現狀態轉移的某個維度只會用到固定相對的 DP 狀態時，就可以捨棄更前面用不到的 DP 狀態了。
對於背包問題我們便能發現，$dp[n][i]$ 只會用到 $dp[n-1][x]$，所以更前面的狀態後面也用不到，那乾脆就不需要了。
於是就會出現以下的 code：

Algorithm 4: 0/1 Knapsack problem with rolling DP

```
fill(dp[0],dp[0]+C,-INF),fill(dp[1],dp[1]+C,-INF);
dp[0][0]=0;
for(int i=1;i<=N;++i)
    for(int j=w[i];j<=C;++j)
        dp[i&1][j]=max(dp[i&1^1][j-w[i]]+v[i],dp[i&1^1][j]);
```

充其量就只是把原本的轉移狀態做點小改寫而已，其理念可以被廣泛運用。
不過像背包問題這種有點性質的題目，其實有更好的滾動 DP 寫法可以直接讓第一個維度消失，供讀者自行研究。

Algorithm 5: 0/1 Knapsack problem with rolling DP, version 2

```
fill(dp,dp+C,-INF);
dp[0]=0;
for(int i=1;i<=N;++i)
    for(int j=C;j>=w[i];--j)//這個反過來跑的動作很重要
        dp[j]=max(dp[j-w[i]]+v[i],dp[j]);
```

2.5.3 無限背包問題

你有一個包包能裝 C 公斤的東西。有 N 樣物品可以無限拿，已知所有物品分別的重量 w 和價值 v。請問這個包包最多可以裝價值多高的物品？
這種變形的背包問題有個很漂亮的性質，如果我們一步一步分析的話。

1. 設置狀態。
   $dp[n][i] =$ 使用 $1 \sim n$ 個物品湊出重量 i 時，所可得到的最大價值。
2. 導出轉移。
   $dp[n][i] = max(dp[n-1][i-w[n]] + v[n], dp[n][i-w[n]] + v[n], dp[n-1][i])$
3. 打好基底。
   $dp[0][0] = 0, dp[0][i] = -INF \quad when \quad i > 0$

再嘗試滾動的話就可以寫出下列和 0/1 背包只有微小差異的程式碼，讀者可以嘗試理解兩者的差別：

Algorithm 6: Unbounded Knapsack problem with rolling DP

```
fill(dp,dp+C,-INF);
dp[0]=0;
for(int i=1;i<=N;++i)
    for(int j=w[i];j<=C;++j)//這裡不需要反過來跑
        dp[j]=max(dp[j-w[i]]+v[i],dp[j]);
```

2.5.4 有限背包問題

你有一個包包能裝 C 公斤的東西。有 N 樣物品，第 i 樣有 a_i 個，已知所有物品分別的重量 w 和價值 v。請問這個包包最多可以裝價值多高的物品？
這種問題喪失了無限背包的漂亮性質，所以看似沒辦法有更好的做法。
能最直觀想到的方法就是把 a_i 個物品都拆開成一個一個獨立的物品，轉換成 0/1 背包之後硬寫，這樣如果個數最大是 A 的話，複雜度就會是慘慘的 $O(NCA)$。
但稍微轉換一下就會發現，如果把 a_i 個物品都拆開成 1 個、2 個、4 個、...、2^x、$a_i - 2^x$ 個的話，不也能利用這些物品湊出各種個數的物品嗎？於是我們拆出來的物品只會有 $\log a_i$ 個，大大減少了 0/1 背包的物品個數，複雜度可降至 $O(NC \log A)$。
其實有限背包問題存在著 $O(NC)$ 的做法，不過會牽扯到所謂的「單調隊列優化」，屆時講到更深入的 DP 時會再跟讀者介紹。

2.5.5 不同狀態的 0/1 背包問題

其實 0/1 背包問題可以用「價值」當狀態，定義大約是 $dp[n][i] =$ 使用 $1 \sim n$ 個物品湊出價值 i 時，所可得到的最小重量，最後找最大的價值滿足重量 $\leq C$ 就好了。複雜度會是 $O(NV)$，其中 V 是最大價值和。

2.5.6 最長遞增子序列 (Longest Increasing Subsequence)

給你長度為 N 的一個正整數序列，請你求出最長嚴格遞增子序列的長度。
所謂嚴格遞增子序列，是指去掉序列中的某些數字之後，剩下的子序列是嚴格遞增的。
這題的狀態設置感覺很麻煩，但我們可以捨棄的平常 DP 得到的東西就是答案的想法來定義狀態。

1. 設置狀態。
   $dp[n][i] =$ 使用 $1 \sim n$ 個數字湊出長度 i 的 LIS，末端數字最小為何。

2. 導出轉移。
   $$dp[n][i] = \begin{cases} min(dp[n-1][i], a[n]) & a[n] > dp[n-1][i-1] \\ dp[n-1][i] & else \end{cases}$$

3. 打好基底。
   $dp[0][0] = -INF, dp[0][i] = INF \quad when \quad i > 0$

如此只要檢查最大的 i 滿足 $dp[N][i]$ 不是 INF 就是答案了。
但在這題上，有個非常迥異的性質可以供我們優化。

2.5.7 二分搜優化

我們令 $g[i] = min(dp[x][i]) \mid 1 \le x \le k$，其 k 是現在跑到第幾個元素，並隨時更新 $g[i]$ 就會發現，這個序列恰好是嚴格遞增，且每次更新只會有一個數字被改變，且被改變的數字恰好會改成 $a[i]$！
更好的是，被改成 $a[i]$ 的數字恰好就是 lower_bound $a[i]$，於是我們就可以利用二分搜來得到一個 $O(n \log n)$ 的演算法了。

其實這就相當於是在把滾動 DP 壓平，和背包問題的一維滾動版本的想法很類似。這也告訴了我們 DP 的各種複雜度都不一定會完全跟著狀態和轉移走。

2.5.8 最長共同子序列 (Longest Common Subsequence)

給你兩個序列，問你最長共同子序列的長度。
所謂共同子序列，是指去掉兩個序列中的某些數字之後，剩下的兩個序列是完全相同的。
這題的理念就稍微簡單一些，不過其存的值跟序列本身的值並沒有太大關係，有關係的是轉移。

1. 設置狀態。
   $dp[n][m] =$ 使用 $a[1] \sim a[n]$ 和 $b[1] \sim b[m]$ 得到的 LCS 長度。

2. 導出轉移。
   $$dp[n][m] = \begin{cases} dp[n-1][m-1] + 1 & a[n] == b[m] \\ max(dp[n-1][m], dp[n][m-1]) & else \end{cases}$$

3. 打好基底。
   $dp[0][i] = 0, dp[i][0] = 0 \quad when \quad i \geq 0$

概念並不難懂，這裡就不多做說明。

2.5.9 習題

除了三題經典題外，這裡還會附上許多 DP 題目供大家思考。
可以發現 DP 的題目特別多，因為他是一個重要的概念，甚至可以獨立出來當作一個領域，所以熟悉 DP 會是非常吃香的一件事情。

1. (ZJ b184) 背包問題。
2. (TIOJ 1175) LIS。
3. (TIOJ 1387) 有限背包問題。
4. (ZJ c499) LCS。
5. (ZJ c264) 變形的背包問題。
6. (105 全國賽 pD/TIOJ 1941) 變形的 LIS，請用類似的方法觀察，並想出要用什麼資料結構或演算法優化。
7. (ZJ a252) 變形的 LCS。
8. (ZJ b855) 給你一堆數字，你必須決定分別要往上或往右這些數字步，問最後和原點的最小距離為何。
9. (ZJ b589) 每個路徑都有一些分數，如果在一條路徑上按正常速度來跑，就只能拿到原始分數，如果加速跑，就能拿到兩倍分數，不過需要在加速跑完後的下一條路徑上休息而速度變慢得到 0 分，試求最大得分。
10. (2016 NPSC/TIOJ 2020) 問你 $N \times M$ 的表格可以切割出的最小正方形數量 (這題當年一堆人以為是 greedy 一堆人 WA XD)。
11. (TIOJ 2061) 求一個字串的最長回文子序列。
12. (2017 TOI 二模 pC/TIOJ 1973) 題敘略。

2.6 DP 回溯

大家寫的 DP 題常常都只會出現「求極值」之類的問題，但在現實上真的只需要「求極值」嗎？
像是背包問題，儘管求出了最大價值，可是真正應該拿些什麼東西來達到最大價值才應該是真正的關鍵，總不可能叫人胡亂湊物品湊到最大值才收手吧？
於是被稱為「回溯」的動作便相對重要，大概有以下兩種方式：

1. 記錄好每個狀態的轉移來源，然後從最終狀態逆著走回去把答案推出來，需要多一倍的空間和時間。
2. 從最終狀態逆著找合理的轉移來源走回去，省下空間，但實作通常困難 (甚至會破壞掉原本優化好的時間，例如 $O(n \log n)$ 的 LIS 便不適用。)
3. 在每個狀態多存一次當前答案，連著答案一起轉移，優點是很好想，完全不用怕出錯；缺點是通常時間和空間都會需要多乘上 $O($ 答案長度 $)$。
4. 做一個反著跑的 DP，從開頭 greedy 取數字，只要取了這個數字會導致答案變差，就不能用這個數字。複雜度通常與選擇答案的種類數有關。

根據不同的題目，DP 回溯的實作方法也會有許多不同，在此無法舉出什麼典型的例子，建議讀者還是打開習題，慢慢體會 DP 回溯的感覺。

2.6.1 非特殊 Judge

讀者可能會發現一個問題，有時候回溯出來的答案可能不止一種，這使出題者需要多寫一個評測程式來做驗證，儘管驗證的方法有可能比較簡單，但比起純測資比對來說還是稍顯麻煩。
這時候題目常會有各式各樣的限制來把答案限制在唯一，只要在程式裡稍微動個手腳，就能使出題變得輕鬆許多。
舉個最常見的例子：最小字典序！
這種題目常常很適合使用第四種方法，在 greedy 的時候 greedy 取能造成最小字典序那條路就可以了，不過可惜的是有些題目無法反著 DP，此時很有可能就得退一步採取第三種做法，並在每個狀態裡存下「該狀態下，最小字典序的答案」。

題外話，有些題目可能會要求你輸出第 k 大字典序，但問題是答案排列組合可能超過 long long，這時候有個小技巧就是，可以把擁有的組合數超過題目給定 k 的上限以上的狀態直接砍成上限，回溯的時候就不用去碰那些狀態了 (當然是要在能計算組合數的 DP 題目裡實作)。

2.6.2 習題

1. (ZJ d242) 找出最大字典序的 LIS 序列。
2. (TIOJ 1997) 把一個序列切成 K 份並使「偶數份中所有數字總和」減掉「奇數份中所有數字總和」最大，並輸出任意一種切 $K-1$ 刀方法每刀的位置。
3. (ZJ b397) 輸出所有的 LCS。
4. (Codeforces 56 D) 給你兩個字串，你現在可以對第一個字串插入、刪除、修改任何字元，請花最少步的操作將第一個字串修成第二個字串，並輸出修改過程。

2.7 狀態壓縮

又稱「位元 DP」。
有時候當狀態很複雜，需要用到多個變數的時候，如果這時多個變數可能的值只是 0/1，或只是一些很少的數字可能性，就可以把狀態壓縮成一個變數，利用其位元或模數來表示狀態，節省 coding 複雜度和實作方法的思考。
讓我們看看一道經典的例題 (相關名詞在圖論會講得詳細一些)：
Description

給予一張完全圖，請輸出在每個點都恰遍歷過一遍並回到起點的情況下，路徑的最小權重和。

Input

首行輸入為一個正整數 V，代表點有 V 個。接下來 $V-1$ 行，第 $i+1$ 行會有 $V-i$ 個數字，第 $i+1$ 行的第 j 個數字代表點 i 到點 j 的路徑權重。

這道題目被稱為「旅行推銷員（Travelling salesman problem）」問題。很明顯有個暴力 $V!$ 的作法在試著騙大家去實作，但這題的 V 是可以到 20 的！聽起來頗複雜，不如我們先考慮 $V=5$ 的情況：

$$dp[i][a_1][a_2][a_3][a_4] = \text{走到點 } i\text{，}a_x = 1 \text{ 表示 } x \text{ 已走過時的最小權重和，且 0 是起點。}$$

稍微寫出其中一種狀態的狀態轉移：

$$dp[2][1][1][0][1] = min\,(dp[i][1][0][0][1] + dis[i][2]), i = 1, 4\text{。}$$

由於數學式的寫法有些複雜，這裡就讓讀者自行理解看看以上轉移式。而答案想必就是取 $dp[i][1][1][1][1] + dis[i][0]$ 的最小值了。
那該不會在 V=20 的時候要開到 20 維陣列吧……？
當然不是。現在讓我們試著利用狀態壓縮的精神來完成題目。

$$dp[i][a] = \text{走到點 } i\text{，}(a\&(1 << x)) = 1 \text{ 表示 } x \text{ 已走過時的最小權重和。}$$

因為撰寫上的方便，起點可以自行決定後修改沒關係 (畢竟是繞一圈，起點是誰都沒差)。
觀察到除了第一維以外，所有的維度都只會出現 0 和 1，所以我們根本就不需要開到 20 維，只要用位元處理，便能輕鬆到達到要求！dp 的狀態數有 $V \times 2^V$ 種，轉移 $O(V)$，複雜度 $O(V^2 2^V)$

這類題目的實作複雜度即使優化過了還是會有點高，大概要多練習才會熟悉。

2.7.1 習題

1. (ZJ c239) 旅行推銷員問題。

2. (TIOJ 1014) 基座上每隔一公尺就會有一個地鼠洞，由左至右編號為 $1, 2, \square, n$。玩家站在這個基地的最左邊，與第一個地鼠洞相距 1 公尺，準備開始這個遊戲。編號為 i 的地鼠洞每 Ti 秒地鼠會出現一次。被打的地鼠不再出現，只要將所有地鼠打完，就結束遊戲。已知玩家以每秒 1 公尺的速度移動，問結束遊戲所需的最少秒數。
3. (TIOJ 2070) 給你由小寫英文字母組成的字串 X，請判斷是不是每一個前 K 個英文字母的子集的排列都是 X 的子序列。
4. (ZJ a128) 你有一個 $x \times y$ 的方格表。給你 $n \leq 15$ 個數，請問你是否能沿著某條橫線或直線切割方格表 $n-1$ 次，使得切出來的 n 塊面積跟那 n 個數一樣。
5. (TIOJ 1452) 給你一塊 $n \times m$ 的房間地板，請找出用相同的 2×1 大小的巧拚鋪滿整塊房間地板的方法數。
6. (2015 TOI 一模/TIOJ 1908) 你有一個 $n \times n$ 的方格表 $n \leq 22$。請選出任意多的數字，使得所有數字的兩兩皆不被對方的周圍 8 格蓋到，並使總和最大。

3 隨機演算法 (random algorithm)

計算機率後適當地捨棄一些尊嚴 (?)，進而透過隨機函數來得到最佳解。
至於什麼是計算機率呢？如果你對機率有一定的自信的話，可以嘗試計算 random 一次戳中答案的機率，然後重複執行好幾次來降低失敗的機率，如果失敗的機率夠低的話，就可以嘗試丟丟看。
這類題目不太好解釋，就請各位跟著習題多加學習了。
這邊只是想告訴各位，random 在演算法競賽是允許的一件事情，但也不可過度利用，以免造成 Judge 損害。

3.0.1 習題

1. (104 全國賽 pF/TIOJ 1916) 題敘略。
2. (TIOJ 1093) 最小覆蓋圓問題。
3. (Codeforces 995C) 題敘略。
4. (2017 TOI 三模 pD/TIOJ 1978) 最大團問題。(此題正解並不是 random，但是 random 有很優良的唬爛方法。)