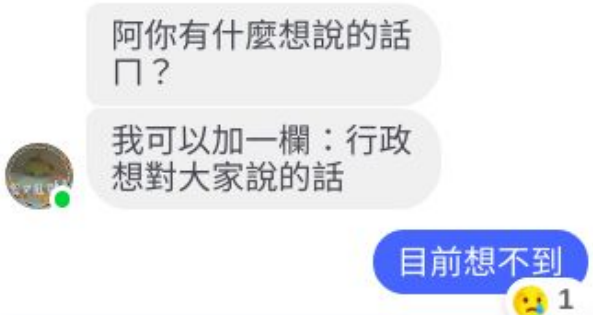
分治 Divide and Conquer

Lecture & modified by yp155136
Credit by TreapKing

一些公告

- 這周的作業已經放上去了
  - 上機 + 手寫
- 4/25(六)認證考
- 上機作業 & bonus 題 deadline:
  4/27(一) 12:00 pm
- 行政想對大家說的話:目前想不到



Q & A

- 影片有什麼問題嗎？～～
- 有問題歡迎 ~~舉手~~ 在 slido 上面發問喔

目錄

- 先列一下影片中例題的重要程度

- 放 L 型格子 ★★★
- 構造等差數列 ★★★
- 快速冪(算 a^n % M)★★★★★
- merge sort ★★★★★
- 快速排序法 ★★★★★
- 逆序數對 ★★★★★

目錄

何時使用分治

- 經驗！
  - 刷題！

何時使用分治

- 經驗！
  - 刷題！
- 通常是你們要你算一個東西，O(N^2) 很簡單
- 但是如果從中間切一半，會找到很棒的 conquer 性質
- 於是就可以用分治來優化XD
- 「先 divide 出兩個小部份，之後在想辦法 conquer 起來」，就是分治的精隨～

分治小技巧

- 養成良好習慣
  - 左閉右閉(我個人常用的方式)
  - 左閉右開
  - ......(?)
- 當 n 很小的時候，可以暴力做，可能會減少遞迴的 cost
  - EX: merge sort 到 n 很小時，就隨便用一個 O(n^2) 的 sort
- 多寫題目(?) XD

分析遞迴的工具

- 為什麼要學這個(?)
- 可以分析你寫的程式的時間複雜度
- 可以耍帥XD

- Credit to Algorithm Design and Analysis, 2019 Fall

先來看看符號的定義

- big-O 定義

$$\mathrm{f(n)} \in O(g(n)) \Rightarrow \exists c, n_0$$
$$\text{s.t.} \quad 0 \leq f(n) \leq cg(n) \forall n \geq n_0$$

- 存在一個常數 c, n_0, 使得當 n 比 n_0 大時, f(n) 都會比 c * g(n) 小
- 例如, 當 f(n) = x^2 + 10x , g(n) = x^3 時, 我們可以選擇 c = 0.5, n_0 = 10

Substitution Method

- 取代法
- 三個大步驟: Guess 、 Verify 、 Solve

- Guess:
  - 猜 f(n) 是屬於哪個複雜度
- Verify:
  - 驗證猜想是否正確
  - 常用 **數學歸納法** 驗證
- Solve :
  - 解出正確的常數
  - 證明 big-O <3

Substitution Method

- 範例:

$$T(n) = \begin{cases} O(1) & \text{if } n = 1 \\ 2T(\frac{n}{2}) + O(n) & \text{if } n \geq 2 \end{cases}$$

- 三個大步驟: Guess 、 Verify 、 Solve

Substitution Method

- 範例:

$$T(n) = \begin{cases} O(1) & \text{if } n = 1 \\ 2T(\frac{n}{2}) + O(n) & \text{if } n \geq 2 \end{cases}$$

- 先來做個簡單的化簡，把 big-O 給去掉

$$T(n) = \begin{cases} a & \text{if } n = 1 \\ 2T(\frac{n}{2}) + bn & \text{if } n \geq 2 \end{cases}$$

其中 a, b > 0

Substitution Method

- 範例:

$$T(n) = \begin{cases} a & \text{if } n = 1 \\ 2T(\frac{n}{2}) + bn & \text{if } n \geq 2 \end{cases}$$

- 猜測 $\mathrm{T(n)} \leq b \times n \log n + an$

Substitution Method

$$\mathrm{T(n)} \leq b \times n \log n + an$$

- 猜測
- 使用數學歸納法
  - n = 1: trivial

Substitution Method

- 猜測 $\mathrm{T(n)} \leq b \times n \log n + an$
- 使用數學歸納法
  - n = 1: trivial
  - n > 1: $T(n) \leq 2T(\frac{n}{2}) + bn$

Substitution Method

- 猜測 $\mathrm{T(n)} \leq b \times n \log n + an$
- 使用數學歸納法
  - n = 1: trivial
  - n > 1: $T(n) \leq 2T(\frac{n}{2}) + bn$

$$\leq 2(b \times \frac{n}{2}\log\frac{n}{2} + a\frac{n}{2}) + bn$$

Substitution Method

- 猜測 $\mathrm{T(n)} \leq b \times n \log n + an$
- 使用數學歸納法
  - n = 1: trivial
  - n > 1: $$\begin{aligned} T(n) &\leq 2T(\tfrac{n}{2}) + bn \\ &\leq 2(b \times \tfrac{n}{2}\log\tfrac{n}{2} + a\tfrac{n}{2}) + bn \\ &= bn\log n - bn\log 2 + an + bn \\ &= bn\log n + an \end{aligned}$$
- 因此，由數學歸納法，可以得到 T(n) 是 O(N log N) 的

Substitution Method

- 剛剛的例子似乎沒有用到「解出正確常數」的樣子
- 那是因為我們運氣特別好 XD
- 讓我們來看更多例子吧～～

Substitution Method

- 範例:

$$T(n) = \begin{cases} O(1) & \text{if } n = 1 \\ 4T(\frac{n}{2}) + O(n) & \text{if } n \geq 2 \end{cases}$$

- 試證明: T(n) 是 O(n^3)
- Hint:

$$T(n) \in O(n^3) \Rightarrow \exists c, n_0, \text{s.t.} \forall n \geq n_0, T(n) \leq cn^3$$

Substitution Method

- 來用用數學歸納法吧
- n = 1:trivial
- n > 1: $T(n) \leq 4T(\frac{n}{2}) + bn$

Substitution Method

- 來用用數學歸納法吧
- n = 1:trivial
- n > 1: $T(n) \leq 4T(\frac{n}{2}) + bn$

$$\leq 4(c(\frac{n}{2})^3) + bn = \frac{cn^3}{2} + bn$$

$$= cn^3 - (\frac{cn^3}{2} - bn)$$

Substitution Method

- 來用用數學歸納法吧
- n = 1:trivial
- n > 1: $T(n) \leq 4T(\frac{n}{2}) + bn$

$$\leq 4(c(\frac{n}{2})^3) + bn = \frac{cn^3}{2} + bn$$

$$= cn^3 - (\frac{cn^3}{2} - bn)$$

$$\leq cn^3$$

為了讓不等式成立，
也就是說，cn^3 > 2bn

Substitution Method

- 來用用數學歸納法吧
- n = 1:trivial
- n > 1: $T(n) \leq 4T(\frac{n}{2}) + bn$

$$\leq 4(c(\frac{n}{2})^3) + bn = \frac{cn^3}{2} + bn$$

$$= cn^3 - (\frac{cn^3}{2} - bn)$$

$$\leq cn^3$$

為了讓不等式成立，
也就是說，cn^3 > 2bn

我們可以設c >= 2b，n >= 1

- 所以，當 $c = 2b, n_0 = 1$時，
  T(n) 就是 O(n^3)

Substitution Method

- 範例:

$$T(n) = \begin{cases} O(1) & \text{if } n = 1 \\ 4T(\frac{n}{2}) + O(n) & \text{if } n \geq 2 \end{cases}$$

- 試證明: T(n) 是 O(n^2)

Substitution Method

- $\exists c, n_0, \text{s.t.} T(n) \leq cn^2 \forall n \geq n_0$
- n = 1 時, trivial
- n > 1 時, $T(n) \leq 4(c(\frac{n}{2})^2) + bn$

Substitution Method

- $\exists c, n_0, \text{s.t.} T(n) \leq cn^2 \forall n \geq n_0$
- n = 1 時, trivial
- n > 1 時, $T(n) \leq 4(c(\frac{n}{2})^2) + bn$

$$= cn^2 + bn$$

- 哇, 好像沒辦法繼續推了
- 我們真的猜錯了嗎？

Substitution Method

- $\exists c, n_0, \text{s.t.} T(n) \leq cn^2 \forall n \geq n_0$
- n = 1 時, trivial
- n > 1 時,

$$T(n) \leq 4(c(\frac{n}{2})^2) + bn$$

$$= cn^2 + bn$$

- 哇, 好像沒辦法繼續推了
- 我們真的猜錯了嗎？

- 並沒有XD ，這也是取代法的一個盲點:有的時候猜錯就證不出來了

Substitution Method

- $\exists c_1, c_2, n_0, \text{s.t.} T(n) \leq c_1 n^2 - c_2 n \forall n \geq n_0$
- n = 1 時, $T(1) \leq c_1 - c_2 \Rightarrow c_1 \geq c_2 + 1$

Substitution Method

- $\exists c_1, c_2, n_0, \text{s.t.} T(n) \leq c_1 n^2 - c_2 n \forall n \geq n_0$
- n = 1 時, $T(1) \leq c_1 - c_2 \Rightarrow c_1 \geq c_2 + 1$
- n > 1 時,

$$
\begin{aligned}
T(n) &\leq 4T(\tfrac{n}{2}) + bn \\
&\leq 4(c_1(\tfrac{n}{2})^2 - c_2(\tfrac{n}{2})) + bn
\end{aligned}
$$

Substitution Method

- $\exists c_1, c_2, n_0, \text{s.t.} T(n) \leq c_1 n^2 - c_2 n \forall n \geq n_0$
- n = 1 時, $T(1) \leq c_1 - c_2 \Rightarrow c_1 \geq c_2 + 1$
- n > 1 時,

$$
\begin{aligned}
T(n) &\leq 4T(\tfrac{n}{2}) + bn \\
&\leq 4(c_1(\tfrac{n}{2})^2 - c_2(\tfrac{n}{2})) + bn \\
&= c_1 n^2 - 2c_2 n + bn \\
&= c_1 n^2 - c_2 n - (c_2 n - bn) \\
&\leq c_1 n^2 - c_2 n
\end{aligned}
$$

為了讓不等式成立,
也就是說, c_2n > bn

Substitution Method

- $\exists c_1, c_2, n_0, \text{s.t.} T(n) \leq c_1 n^2 - c_2 n \forall n \geq n_0$
- n = 1 時, $T(1) \leq c_1 - c_2 \Rightarrow c_1 \geq c_2 + 1$
- n > 1 時,

$$\begin{aligned} T(n) &\leq 4T(\tfrac{n}{2}) + bn \\ &\leq 4(c_1(\tfrac{n}{2})^2 - c_2(\tfrac{n}{2})) + bn \\ &= c_1 n^2 - 2c_2 n + bn \\ &= c_1 n^2 - c_2 n - (c_2 n - bn) \\ &\leq c_1 n^2 - c_2 n \end{aligned}$$

- 所以, 當 $n_0 = 1, c_1 = b + 2, c_2 = b$ 時, T(n) 就是 O(n^2) !

為了讓不等式成立,
也就是說, c_2n > bn

我們可以設c_2 >= b, n >= 1

Substitution Method

- 除了上面那個技巧之外，Substitution 還有一個經典用法：變數變換
- 例：$T(n) = 2T(\sqrt{n}) + \log n$

Substitution Method

- 例: $T(n) = 2T(\sqrt{n}) + \log n$
- 變數變換1:

$$k = \log n, n = 2^k \Rightarrow T(2^k) = 2T(2^{\frac{k}{2}}) + k$$

Substitution Method

- 例: $T(n) = 2T(\sqrt{n}) + \log n$
- 變數變換1:

$$k = \log n, n = 2^k \Rightarrow T(2^k) = 2T(2^{\frac{k}{2}}) + k$$

- 變數變換2:

$$S(k) = T(2^k) \Rightarrow S(k) = 2S(\frac{k}{2}) + k$$

Substitution Method

- 例: $T(n) = 2T(\sqrt{n}) + \log n$
- 變數變換1:

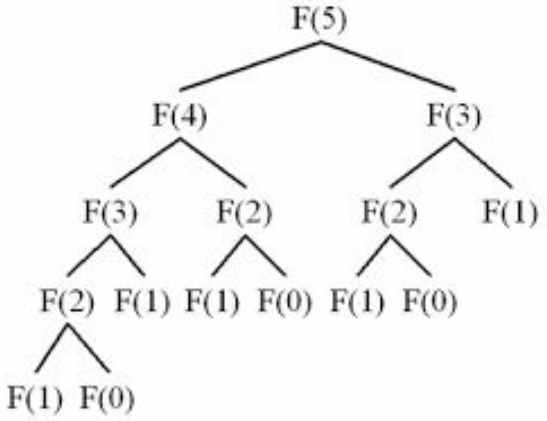
  $k = \log n, n = 2^k \Rightarrow T(2^k) = 2T(2^{\frac{k}{2}}) + k$
- 變數變換2:

  $S(k) = T(2^k) \Rightarrow S(k) = 2S(\frac{k}{2}) + k$
- 所以:

$S(k) = T(2^k) = O(k \log k) \Rightarrow T(n) = O(\log n \log \log n)$

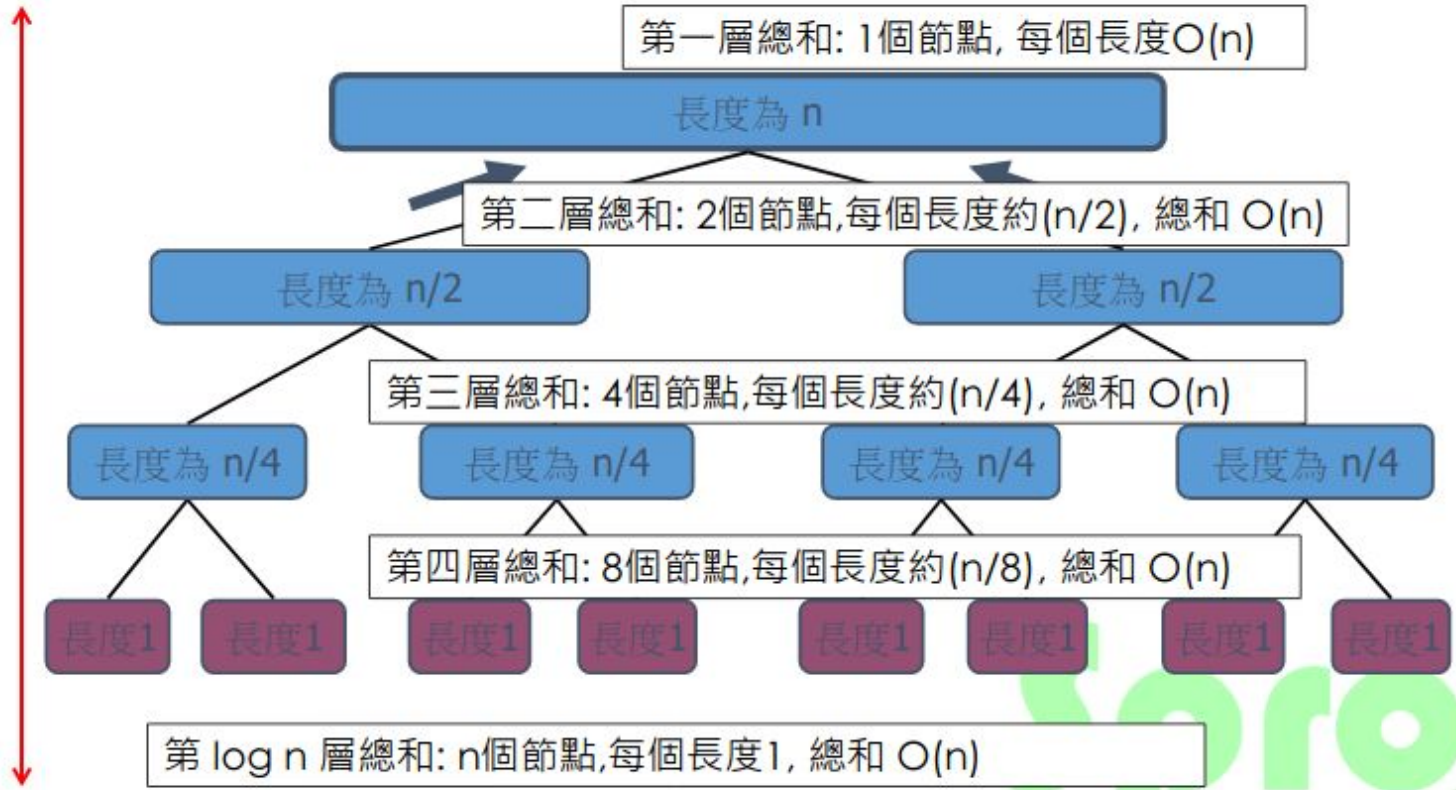
Recursion-Tree Method

- 遞迴樹法
- 顧名思義，就是把**遞迴樹**畫出來之後，去把花費的**時間總和**算出來
- 算出來之後，在用前面介紹的 substitution method 檢查
- 遞迴樹：遞迴的過程



reference：http://kkc47.blogspot.com/2014/08/learning-dynamic-programming-day-1.html

分析一下執行時間


第一層總和: 1個節點, 每個長度O(n)
長度為 n
第二層總和: 2個節點,每個長度約(n/2), 總和 O(n)
長度為 n/2
長度為 n/2
第三層總和: 4個節點,每個長度約(n/4), 總和 O(n)
長度為 n/4
長度為 n/4
長度為 n/4
長度為 n/4
第四層總和: 8個節點,每個長度約(n/8), 總和 O(n)
長度1
長度1
長度1
長度1
長度1
長度1
長度1
長度1
第 log n 層總和: n個節點,每個長度1, 總和 O(n)

Recursion-Tree Method

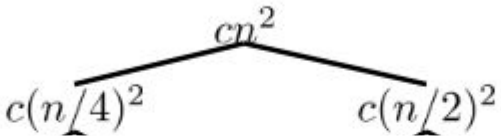
- 來看個例子吧～
- 例:

$$T(n) = T(\frac{n}{2}) + T(\frac{n}{4}) + cn^2$$

- 試試看把遞迴樹畫出來吧～～

Recursion-Tree Method

$$T(n) = T(n/4) + T(n/2) + cn^2$$

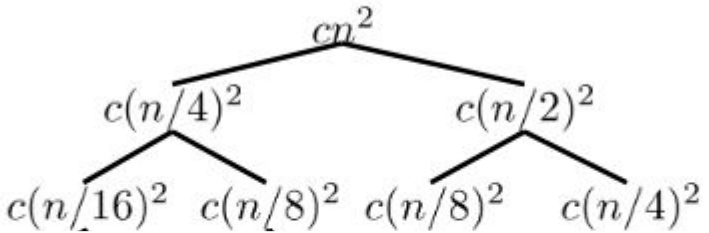


cn^2

$\frac{5}{16}cn^2$

Recursion-Tree Method

$$T(n) = T(n/4) + T(n/2) + cn^2$$



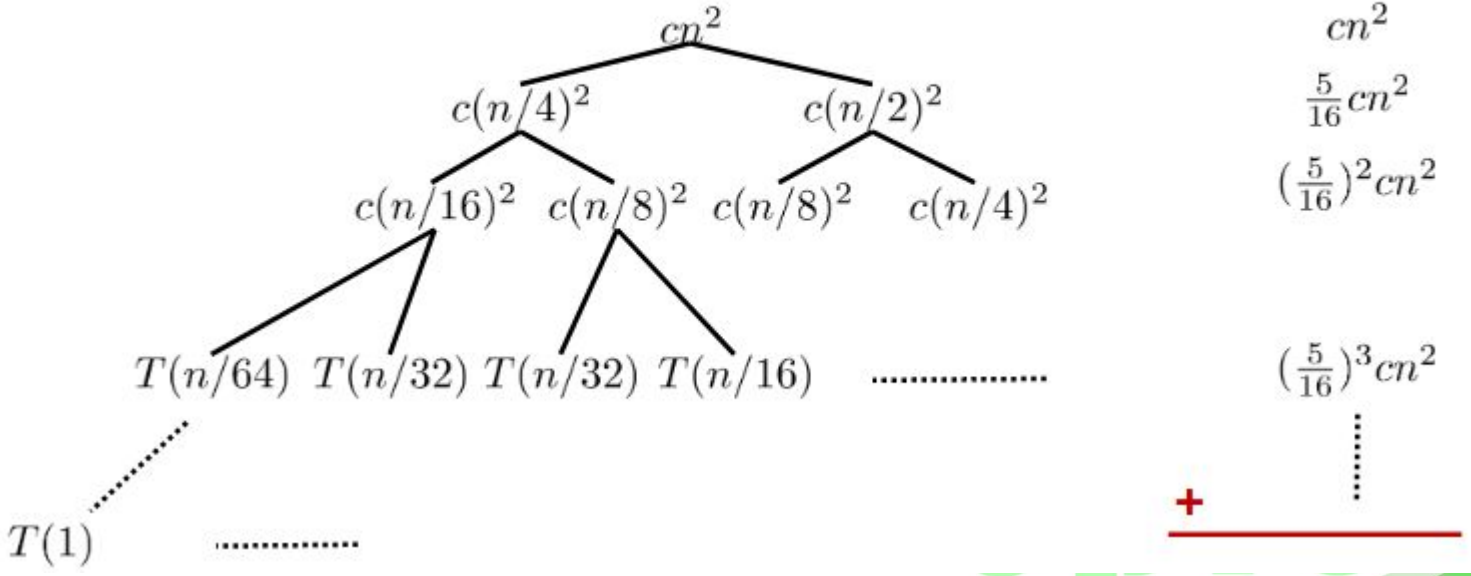
cn^2

$\frac{5}{16}cn^2$

$(\frac{5}{16})^2cn^2$

Recursion-Tree Method

$$T(n) = T(n/4) + T(n/2) + cn^2$$



Recursion-Tree Method

- 例: $T(n) = T(\frac{n}{2}) + T(\frac{n}{4}) + cn^2$
- 可以得到

$$T(n) \leq (1 + \frac{5}{16} + (\frac{5}{16})^2 + \ldots)cn^2$$

Recursion-Tree Method

- 例: $T(n) = T(\frac{n}{2}) + T(\frac{n}{4}) + cn^2$
- 可以得到

$$T(n) \leq (1 + \frac{5}{16} + (\frac{5}{16})^2 + \ldots)cn^2$$

- 根據等比級數，可得

$$T(n) \leq \frac{1}{1-\frac{5}{16}}cn^2 = \frac{16}{11}cn^2$$

- 因此，T(n) 是 O(n^2)

Master Theorem

- 主定理
- 很好用的工具！
- ~~會這個就不用會前面兩個方法子~~

Master Theorem

Let $T(n)$ be a positive function satisfying the following recurrence relation

$$T(n) = \begin{cases} O(1) & \text{if } n \leq 1 \\ a \cdot T(\frac{n}{b}) + f(n) & \text{if } n > 1, \end{cases}$$

where $a \geq 1$ and $b > 1$ are constants.

- Case 1: If $f(n) = O(n^{\log_b a - \epsilon})$ for some constant $\epsilon > 0$, then $T(n) = \Theta(n^{\log_b a})$.
- Case 2: If $f(n) = \Theta(n^{\log_b a})$, then $T(n) = \Theta(n^{\log_b a} \cdot \log n)$.
- Case 3: If
  - $f(n) = \Omega(n^{\log_b a + \epsilon})$ for some constant $\epsilon > 0$, and
  - $a \cdot f(\frac{n}{b}) \leq c \cdot f(n)$ for some constant $c < 1$ and all sufficiently large n,

  then $T(n) = \Theta(f(n))$.

reference: 隨機客的演算法投影片

Master Theorem

- 簡單來說，就是比較 f(n) 跟 $n^{\log_b a}$ 的關係
- 如果一樣的話，就加個 log ，否則就是比較大的那個
- 以下會分三種情況說明，順便會舉點例子
  - 小於
  - 等於
  - 大於

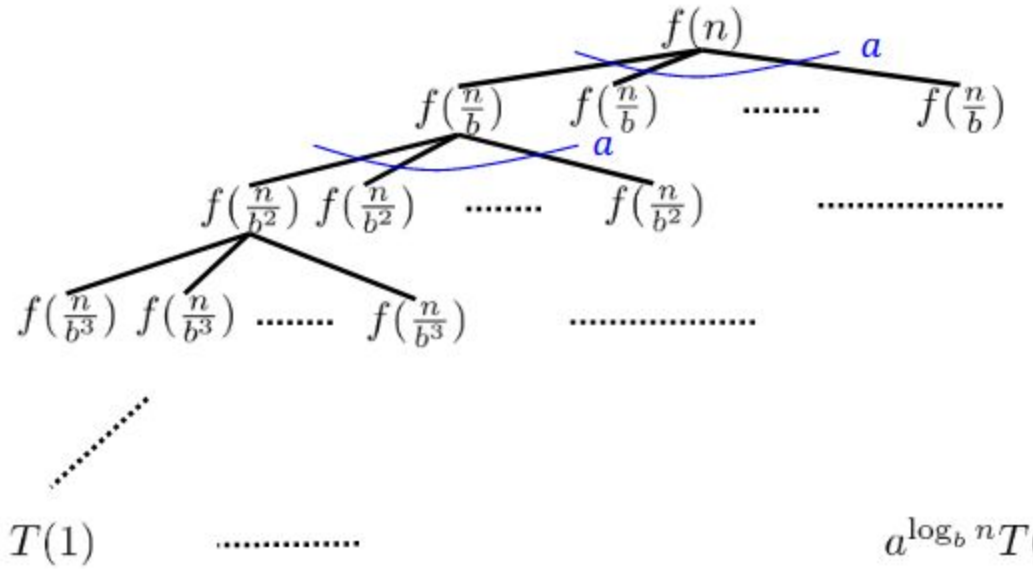
Master Theorem

- 比較 f(n) 跟 $n^{\log_b a}$ 的關係
- 如果 $f(n) < n^{\log_b a}$，那麼 $T(n) = O(n^{\log_b a})$
- 例如，$T(n) = 9T(\frac{n}{3}) + n$
- 根據主定理，$n < n^{\log_3 9}$

  所以 T(n) 是 O(n^2)
- 可以試著用 substitution method 寫寫看 (?)
- 可以試著畫畫看 recursion tree (?)

Master Theorem

$$T(n) = 9T(\tfrac{n}{3}) + n, T(1) = 1$$



$$f(n) = n$$

$$af(\tfrac{n}{b}) = \tfrac{9}{3}n$$

$$a^2 f(\tfrac{n}{b^2}) = (\tfrac{9}{3})^2 n$$

$$a^3 f(\tfrac{n}{b^3}) = (\tfrac{9}{3})^3 n$$

$$a^{\log_b n} T(1) = 9^{\log_3 n} = (\tfrac{9}{3})^{\log_3 n} n$$

$f(n)$ grows polynomially slower than $n^{\log_b a}$

Master Theorem

- 比較 f(n) 跟 $n^{\log_b a}$ 的關係
- 如果 $f(n) = n^{\log_b a}$，那麼

$$T(n) = O(f(n)\log n) = O(n^{\log_b a}\log n)$$

- 例如，$T(n) = 2T(\frac{n}{2}) + O(n)$
- T(n) = O(n log n)，前面也有用 substitution 方法證明過XD
- 可以想像成原本的複雜度多一個 log (?

Master Theorem

- 比較 f(n) 跟 $n^{\log_b a}$ 的關係
- 如果 $f(n) > n^{log_b a}$，那麼

$$T(n) = O(f(n))$$

- 例如，$T(n) = T(\frac{n}{2}) + O(n^2)$
- T(n) = O(n^2)，前面有用 recursion-tree 證過了 XD
- 注意到這個 case 有個額外的條件，但是不常發生(?)

分析遞迴的工具

- 數學理論的部份就講到這邊啦
- 什麼東西都記不起來，記得 Master Therorem 就好 XD
- 接下來要講實際上會遇到的題目囉，~~前面睡著的可以起床ㄌXD~~

最大連續和

- 給你一個長度為 N 的序列 a_1, a_2, ..., a_N，請你找到 (L, R)，滿足 a_L + ... + a_R 最大
- N <= 10^5
- 先來想想看要怎麼做吧～
- 練習題 :https://zerojudge.tw/ShowProblem?problemid=d784

最大連續和

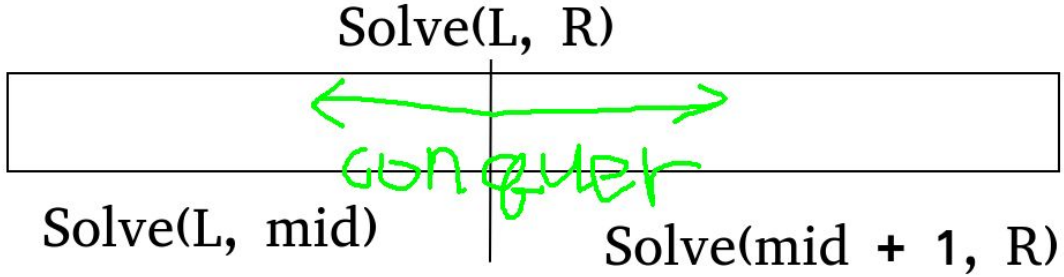
- O(N^3)
- O(N^2)
- O(N log N)
- O(N) (?)
- 那換一個限制，只能用「分治法」來做
  - 因為我們現在在教分治嘛XD

最大連續和

- 答案有 O(N^2) 種
- 我們有沒有辦法好好的 divide 成一半，然後想盡辦法 conquer 起來呢？

最大連續和

- 是可以的！
- 我們先把序列切成兩半，左右遞迴求出各自半部的最佳解



  - Solve(L, R) 會回傳 a_L, ..., a_R 的最佳解

最大連續和 --- conquer

- 因為一定會拿 a[mid] 跟 a[mid + 1]
- 所以我們要求的東西 a[L] + ... + a[R]，可以換簡成 (a[L] + ... + a[mid]) + (a[mid + 1] + ... + a[R])
- 有沒有發現，上面的式子就是從中間點開始，往左 & 往右的走一段路後，得到的總和
- 於是乎，分別取左、右兩邊能有到的的最大總和就可以了～

最大連續和

```
int solve(int L, int R) {
    if (L == R) {
        return a[L];
    }
    int mid = (L + R) >> 1;
    int ans = max(solve(L, mid), solve(mid + 1, R));
    int lmax = a[mid], lpre = a[mid];
    for (int i = mid - 1; i >= L; --i) {
        lpre += a[i];
        lmax = max(lmax, lpre);
    }
    int rmax = a[mid + 1], rpre = a[mid + 1];
    for (int i = mid + 2; i <= R; ++i) {
        rpre += a[i];
        rmax = max(rmax, rpre);
    }
    return max(ans, lmax + rmax);
}
```

最大連續和

- 複雜度分析

最大連續和

- 複雜度分析
- 假設 T(n) 是 solve() 的長度為 n 的複雜度
- 那麼, T(n) = 2T(n / 2) + O(n)

最大連續和

- 複雜度分析
- 假設 T(n) 是 solve() 的長度為 n 的複雜度
- 那麼, T(n) = 2T(n / 2) + O(n)
- 所以, 最後的複雜度是 T(n) = O(n log n)

多項式乘法

- 給你兩個n次多項式，請你求出兩個多項式的乘積。
- Ex:

$$f(x) = 2x^2 + 3x - 1, g(x) = x + 4$$

$$f(x) \times g(x) = 2x^3 + 11x^2 + 11x - 4$$

- 練習題：https://tioj.ck.tp.edu.tw/problems/1064
  - 拿部份分數就好了(?)
  - 把 x 當成 10 這樣

多項式乘法

- 給你兩個n次多項式，請你求出兩個多項式的乘積。
- Ex：

$$f(x) = 2x^2 + 3x - 1, g(x) = x + 4$$

$$f(x) \times g(x) = 2x^3 + 11x^2 + 11x - 4$$

- 來想想看要怎麼做吧～ XD

多項式乘法

- Naïve作法
- 用個雙重迴圈跑一跑乘一乘加一加
- 時間複雜度是O(n^2)

多項式乘法

- Naïve作法
- 用個雙重迴圈跑一跑乘一乘加一加
- 時間複雜度是O(n^2)
- 好慢(?)
- 有沒有更快的方式(?)

多項式乘法

- Naïve作法
- 用個雙重迴圈跑一跑乘一乘加一加
- 時間複雜度是O(n^2)
- 據說在古早的時代，從來沒有人質疑過乘法可以做得更快
- 可能會有人想到FFT(Fast Fourier Transform，快速傅立葉轉換)，但我現在沒有要講這個XD
  - 有興趣的同學可以查查相關資料呦

多項式乘法

- 把多項式表達為

$$f(x) = a_0 + a_1 x + a_2 x^2 + \ldots + a_n x^n = \sum_{i=0}^{n} a_i x^i$$

- 同理： $g(x) = \sum_{i=0}^{n} b_i x^i$
- 順便不失一般性的假設n+1是2的次方(如果不是的話，就加入一些係數為0的高次項)
  - 為了方便之後的講解 :)

多項式乘法

- 既然是在教分治，那就先把多項式切兩半看看
- 令 $A = a_0 + a_1 x + \ldots + a_{[n/2]} x^{[n/2]}$

  $B = a_{[n/2]+1} + a_{[n/2]+2} x + \ldots + a_n x^{[n/2]}$
- 同理 $C = b_0 + b_1 x + \ldots + b_{[n/2]} x^{[n/2]}$

  $D = b_{[n/2]+1} + b_{[n/2]+2} x + \ldots + b_n x^{[n/2]}$

多項式乘法

- 既然是在教分治，那就先把多項式切兩半看看
- 令 $A = a_0 + a_1 x + \ldots + a_{[n/2]} x^{[n/2]}$

  $B = a_{[n/2]+1} + a_{[n/2]+2} x + \ldots + a_n x^{[n/2]}$

  $C = b_0 + b_1 x + \ldots + b_{[n/2]} x^{[n/2]}$

  $D = b_{[n/2]+1} + b_{[n/2]+2} x + \ldots + b_n x^{[n/2]}$
- 所以， $f(x) \times g(x) = (A + x^{\frac{n+1}{2}} B) \times (C + x^{\frac{n+1}{2}} D)$

  $= AC + x^{\frac{n+1}{2}} (AD + BC) + x^{n+1} BD$

多項式乘法

$$f(x) \times g(x) = (A + x^{\frac{n+1}{2}} B) \times (C + x^{\frac{n+1}{2}} D)$$
$$= AC + x^{\frac{n+1}{2}} (AD + BC) + x^{n+1} BD$$

- A，B，C，D都是(n/2)次的多項式
- 所以做四次比較小的乘法 & 一點加法就可以解決原問題了！

多項式乘法

$$f(x) \times g(x) = (A + x^{\frac{n+1}{2}} B) \times (C + x^{\frac{n+1}{2}} D)$$
$$= AC + x^{\frac{n+1}{2}} (AD + BC) + x^{n+1} BD$$

- A, B, C, D都是(n/2)次的多項式
- 所以做四次比較小的乘法 & 一點加法就可以解決原問題了！
- 成功切成子問題 => Divide & Conquer!
- 可是，這樣真的有比較快嗎？

多項式乘法

$$f(x) \times g(x) = (A + x^{\frac{n+1}{2}} B) \times (C + x^{\frac{n+1}{2}} D)$$
$$= AC + x^{\frac{n+1}{2}} (AD + BC) + x^{n+1} BD$$

- A, B, C, D都是(n/2)次的多項式
- 所以做四次比較小的乘法 & 一點加法就可以解決原問題了！
- 成功切成子問題 => Divide & Conquer!
- 可是，這樣真的有比較快嗎？

- 來分析一下複雜度，T(n) = 4T(n / 2) + O(n)
  - O(n) 是加法的部份

多項式乘法

$$f(x) \times g(x) = (A + x^{\frac{n+1}{2}} B) \times (C + x^{\frac{n+1}{2}} D)$$
$$= AC + x^{\frac{n+1}{2}} (AD + BC) + x^{n+1} BD$$

- A, B, C, D都是(n/2)次的多項式
- 所以做四次比較小的乘法 & 一點加法就可以解決原問題了！
- 成功切成子問題 => Divide & Conquer!
- 可是，這樣真的有比較快嗎？

- 來分析一下複雜度，T(n) = 4T(n / 2) + O(n)
- 用前面的教的遞迴分析，或者是 Master Theorem，可以得到 T(n) = O(n^2)

- 沒有變快 QQQ

Karatsuba

$$f(x) \times g(x) = (A + x^{\frac{n+1}{2}} B) \times (C + x^{\frac{n+1}{2}} D)$$
$$= AC + x^{\frac{n+1}{2}} (AD + BC) + x^{n+1} BD$$

- 我們要算 AC, AD + BC, BD

Karatsuba

$$f(x) \times g(x) = (A + x^{\frac{n+1}{2}} B) \times (C + x^{\frac{n+1}{2}} D)$$
$$= AC + x^{\frac{n+1}{2}} (AD + BC) + x^{n+1} BD$$

- 我們要算 AC，AD + BC，BD
- 靈光一閃，如果我們計算
  $E = (A + B) \times (C + D) = AC + AD + BC + BD$

Karatsuba

$$f(x) \times g(x) = (A + x^{\frac{n+1}{2}} B) \times (C + x^{\frac{n+1}{2}} D)$$
$$= AC + x^{\frac{n+1}{2}} (AD + BC) + x^{n+1} BD$$

- 我們要算 AC, AD + BC, BD
- 靈光一閃, 如果我們計算
  $E = (A + B) \times (C + D) = AC + AD + BC + BD$
- 就可以把原本的式子化為
  $f(x) \times g(x) = AC + x^{\frac{n+1}{2}} (E - AC - BD) + x^{n+1} BD$
- 乘法只剩下三次了！！

Karatsuba -- 複雜度

- 剛剛上面那個演算法，名子就叫做 Karatsuba
  - ~~不是 konosuba 喔~~
- 我們來分析一下時間複雜度吧～

Karatsuba -- 複雜度

- 剛剛上面那個演算法，名子就叫做 Karatsuba
  - ~~不是 konosuba 喔~~
- 我們來分析一下時間複雜度吧～

$$T(n) = 3T(\frac{n}{2}) + O(n)$$

Karatsuba -- 複雜度

- 剛剛上面那個演算法，名子就叫做 Karatsuba
  - ~~不是 konosuba 喔~~
- 我們來分析一下時間複雜度吧～

$$T(n) = 3T(\frac{n}{2}) + O(n)$$

- 根據前面教的 Master Theorem，可以得到 T(n) 就是

$$O(n^{log_2 3}) \approx O(n^{1.58})$$

矩陣乘法

- 給你兩個 n * n 的矩陣 A, B , 請你計算 C = A * B
- 應該不用舉例吧 OAO

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 0 & 3 \end{bmatrix}$$

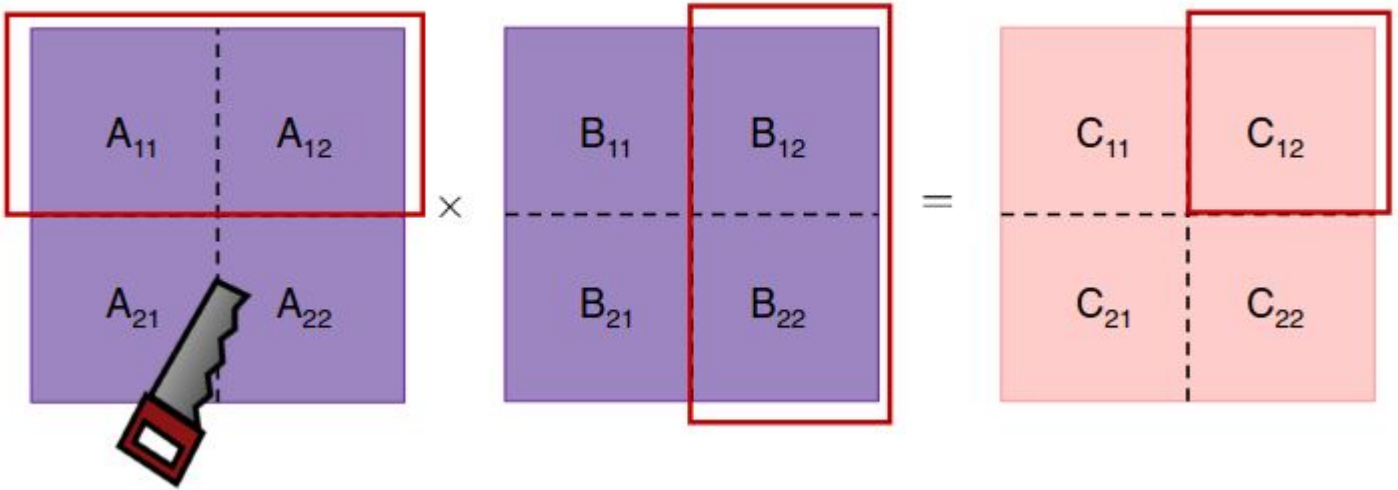
- 先假設 n 是 2 的冪次(存在正整數 k , 使得 n = 2^k)

矩陣乘法

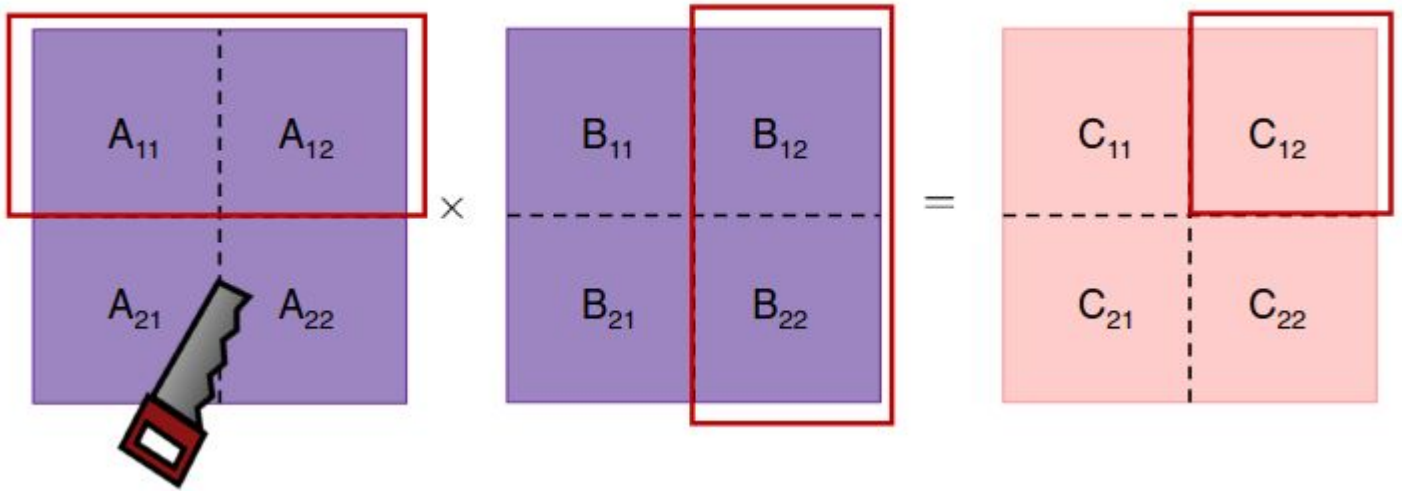
- Naive 作法:O(N^3)
- Too slow(?)

矩陣乘法

- 既然現在在教分治，那我們也來把矩陣 divide 一下吧XD



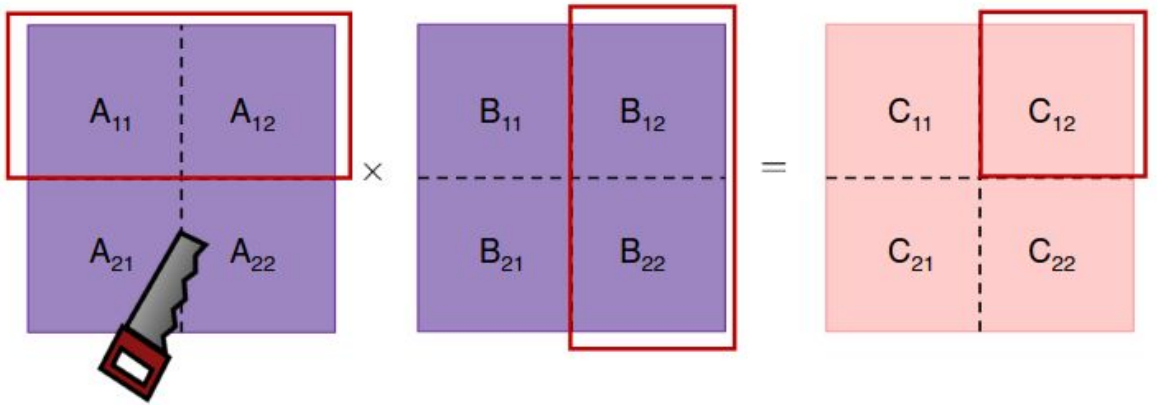
矩陣乘法



- 可以得到:

$$
\begin{aligned}
C_{11} &= A_{11}B_{11} + A_{12}B_{21} \\
C_{12} &= A_{11}B_{12} + A_{12}B_{22} \\
C_{21} &= A_{21}B_{11} + A_{22}B_{21} \\
C_{22} &= A_{21}B_{12} + A_{22}B_{22}
\end{aligned}
$$

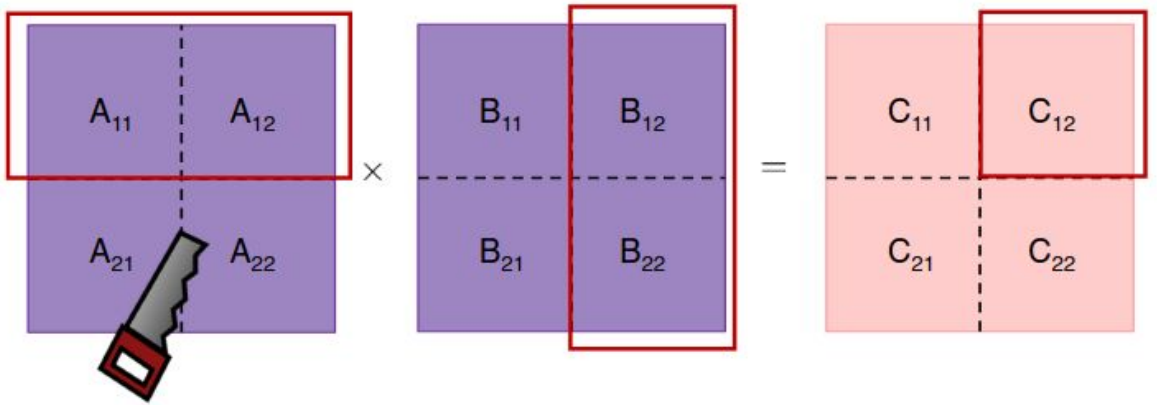
矩陣乘法



$$
\begin{aligned}
C_{11} &= A_{11}B_{11} + A_{12}B_{21} \\
C_{12} &= A_{11}B_{12} + A_{12}B_{22} \\
C_{21} &= A_{21}B_{11} + A_{22}B_{21} \\
C_{22} &= A_{21}B_{12} + A_{22}B_{22}
\end{aligned}
$$

- 來分析一下複雜度吧～

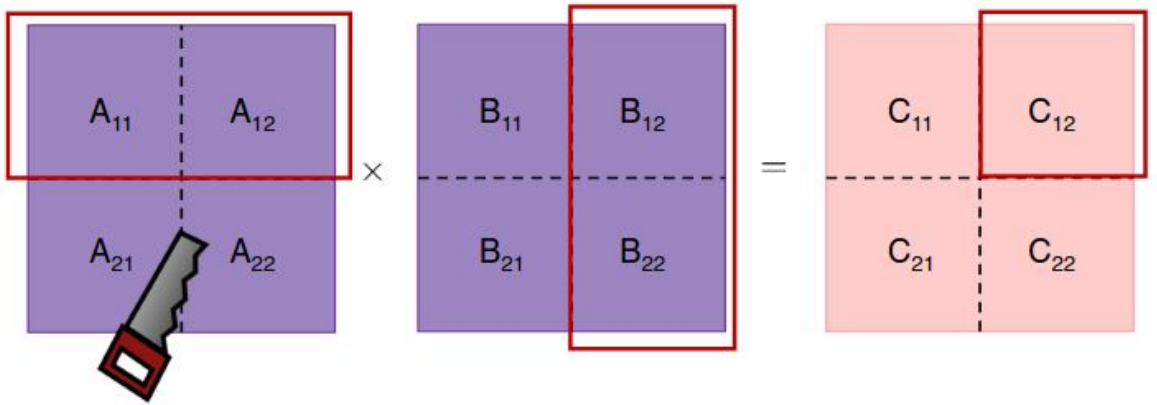
矩陣乘法



$$
\begin{aligned}
C_{11} &= A_{11}B_{11} + A_{12}B_{21} \\
C_{12} &= A_{11}B_{12} + A_{12}B_{22} \\
C_{21} &= A_{21}B_{11} + A_{22}B_{21} \\
C_{22} &= A_{21}B_{12} + A_{22}B_{22}
\end{aligned}
$$

- 來分析一下複雜度吧～
- T(n) = 8T(n / 2) + O(n^2)
  - 要做八次乘法，以及常數次的加法

矩陣乘法



$$
\begin{aligned}
C_{11} &= A_{11}B_{11} + A_{12}B_{21} \\
C_{12} &= A_{11}B_{12} + A_{12}B_{22} \\
C_{21} &= A_{21}B_{11} + A_{22}B_{21} \\
C_{22} &= A_{21}B_{12} + A_{22}B_{22}
\end{aligned}
$$

- 來分析一下複雜度吧～
- T(n) = 8T(n / 2) + O(n^2)
  - 要做八次乘法，以及常數次的加法
- 根據主定理，T(n) = O(n^3)，QQ

Strassen algorithm

- 於是乎, Strassen 來了神來一筆
- 原本要算的東西如下:

$$
\begin{aligned}
C_{11} &= A_{11}B_{11} + A_{12}B_{21} \\
C_{12} &= A_{11}B_{12} + A_{12}B_{22} \\
C_{21} &= A_{21}B_{11} + A_{22}B_{21} \\
C_{22} &= A_{21}B_{12} + A_{22}B_{22}
\end{aligned}
$$

- Strassen 想到的化簡方式如右:
- 乘法從八次變成七次了！！

$$
\begin{aligned}
C_{11} &= M_1 + M_4 - M_5 + M_7 \\
C_{12} &= M_3 + M_5 \\
C_{21} &= M_2 + M_4 \\
C_{22} &= M_1 - M_2 + M_3 + M_6 \\
M_1 &= (A_{11} + A_{22})(B_{11} + B_{22}) \\
M_2 &= (A_{21} + A_{22})B_{11} \\
M_3 &= A_{11}(B_{12} - B_{22}) \\
M_4 &= A_{22}(B_{21} - B_{11}) \\
M_5 &= (A_{11} + A_{12})B_{22} \\
M_6 &= (A_{21} - A_{11})(B_{11} + B_{12}) \\
M_7 &= (A_{12} - A_{22})(B_{21} + B_{22})
\end{aligned}
$$

Strassen algorithm

- 把做乘法的次數降低，就是 Strassen algorithm 的精華
- 來分析一下複雜度：T(n) = 7T(n / 2) + O(n^2)
- 根據 Master Theorem

$$T(n) = O(n^{\log_2 7}) \approx O(n^{2.807})$$

- 這是最快的方式嗎（？

矩陣乘法

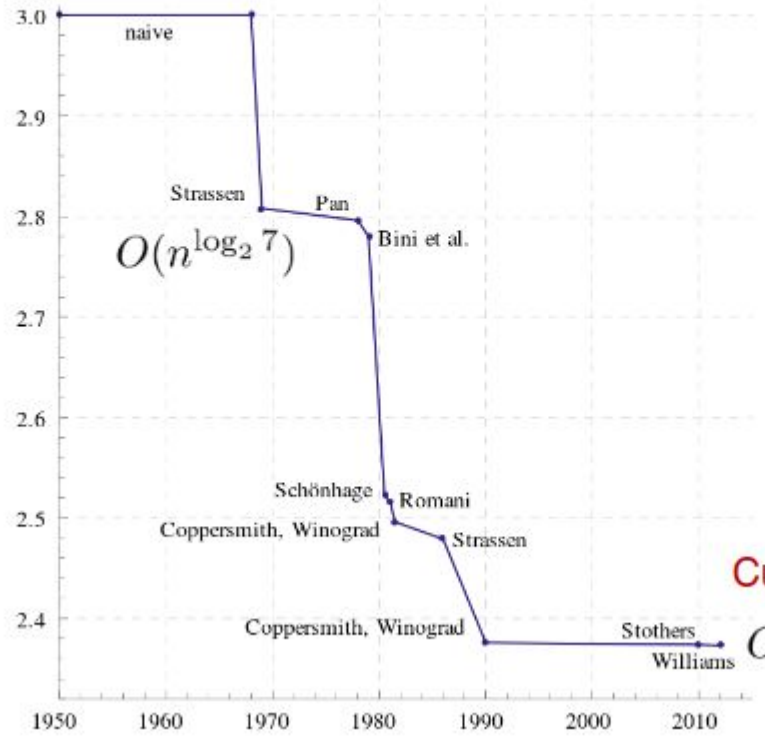
- 如果把矩陣切成 3 * 3，會不會有更好的效果(?)
- 如果把矩陣切成 100 * 100 呢？？？？
- 如果加上一堆數學性質呢？？？？？？？
- 目前已知最好的複雜度是 O(n^2.372)
- 但是這不是最好的複雜度！

矩陣乘法

- 我們還沒有辦法證明矩陣乘法複雜度的下界
- 也就是說，我們還沒有辦法證明矩陣乘法一定會多慢
- 目前在論文中，矩陣乘法的複雜度都被表達成 $O(n^\omega)$
  目前已知 $2 \leq \omega < 2.373$ ，但是還無法證明 omega 比 2 大
- 不過雖然複雜度越來越好，但是背後的常數會越來越大！
- 小心服用(?) XD

矩陣乘法

- Each algorithm gives an upper bound



Current lowest upper bound

平面最近點對

- 在平面上給你 N 個點，要你找出歐式距離最短的兩個點。
- N <= 100,000

$$dis(p_1, p_2) = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$$

- 練習題：https://tioj.ck.tp.edu.tw/problems/1500 、 https://codeforces.com/contest/429/problem/D

平面最近點對

- 開心C(N, 2) = O(N^2)
- 顯然會壞掉QQ
- 如果在平面上想要做divide and conquer, 該怎麼分割問題？

- 備註:這題還有掃描線O(N log N), 隨機期望 O(N) 的作法, 有興趣可以私底下問講師 or 自行上網搜尋喔～～～

平面最近點對

- 我們可以先把所有輸入的點照 x 座標排序後，在中間畫一條分隔線。

平面最近點對

- 我們可以先把所有輸入的點照 x 座標排序後，在中間畫一條分隔線。
- 這樣可能的答案就分成(兩個點都在左邊)、(兩個點都在右邊)、(橫跨分隔線) 三種情況
- 一些平面上的D&C題都會做類似的事。
- 顯然只有點對「橫跨分隔線」的情況需要討論，如果這個情況可以解決的話，其他兩個情況只要遞迴下去解就好了。

平面最近點對

- 如果只是要算分隔線兩邊的最近點對，有什麼好方法嗎？

平面最近點對

- 如果只是要算分隔線兩邊的最近點對，有什麼好方法嗎？
- 因為 x 座標的大小關係已經確立了，所以可以把兩邊直接照 y 座標排序
- 然後就發現還是好困難喔.___.

平面最近點對

- 定神一想，會發現如果遞迴下去後找到的最近點對的距離是 d，那麼我們根本就不需要考慮那些距離超過 d 的點對

平面最近點對

- 定神一想，會發現如果遞迴下去後找到的最近點對的距離是 d，那麼我們根本就不需要考慮那些距離超過 d 的點對
- 所以離分隔線超過 d 的點都不需要去考慮。

平面最近點對

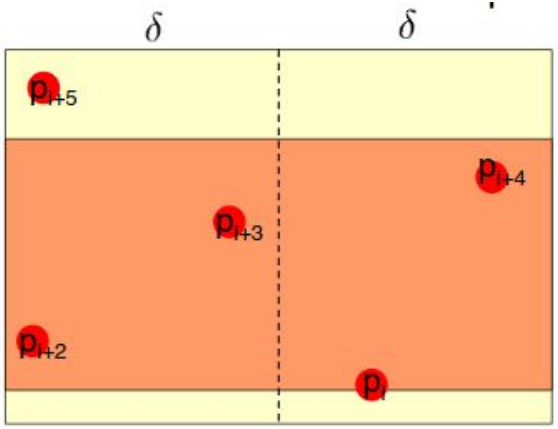
- 定神一想，會發現如果遞迴下去後找到的最近點對的距離是 d，那麼我們根本就不需要考慮那些距離超過 d 的點對
- 所以離分隔線超過 d 的點都不需要去考慮。
- 對於每個點，也只有 y 座標差距不超過 d 的點可能可以讓你找到更近的點對
- 這樣子複雜度會是好的嗎？ OAO

平面最近點對

- 定神一想，會發現如果遞迴下去後找到的最近點對的距離是 d，那麼我們根本就不需要考慮那些距離超過 d 的點對
- 所以離分隔線超過 d 的點都不需要去考慮。
- 對於每個點，也只有 y 座標差距不超過 d 的點可能可以讓你找到更近的點對
- 聽起來只是個壓常數的剪枝，但在什麼情況下，這個做法一樣會退化成 O(N^2) 呢？

平面最近點對

- 因為已經知道各自的最近點對的距離是 d ，所以每個點附近的點不會太多!



- 附近的點只會有常數個，所以直接跑下去複雜度就是好(?)的！

平面最近點對

- 來分析一下複雜度吧～
- $T(n) = 2T(n / 2) + O(n \log n)$
  - 要把點按照 y 座標排序

平面最近點對

- 來分析一下複雜度吧～
- T(n) = 2T(n / 2) + O(n log n)
  - 要把點按照 y 座標排序
- 根據 recursion-tree，這看起來是 O(n (log n)^2)

平面最近點對

- 來分析一下複雜度吧～
- T(n) = 2T(n / 2) + O(n log n)
  - 要把點按照 y 座標排序
- 根據 recursion-tree，這看起來是 O(n (log n)^2)
- 根據 master theorem，這看起來是 O(n log n) !?!
- 到底哪個才是對的 OAO

再看一次主定理

Let $T(n)$ be a positive function satisfying the following recurrence relation

$$T(n) = \begin{cases} O(1) & \text{if } n \leq 1 \\ a \cdot T(\frac{n}{b}) + f(n) & \text{if } n > 1, \end{cases}$$

where $a \geq 1$ and $b > 1$ are constants.

- Case 1: If $f(n) = O(n^{\log_b a - \epsilon})$ for some constant $\epsilon > 0$, then $T(n) = \Theta(n^{\log_b a})$.
- Case 2: If $f(n) = \Theta(n^{\log_b a})$, then $T(n) = \Theta(n^{\log_b a} \cdot \log n)$.
- Case 3: If
  - $f(n) = \Omega(n^{\log_b a + \epsilon})$ for some constant $\epsilon > 0$, and
  - $a \cdot f(\frac{n}{b}) \leq c \cdot f(n)$ for some constant $c < 1$ and all sufficiently large n,

  then $T(n) = \Theta(f(n))$.

平面最近點對

- Master theorem case 3 的一個額外條件：

  $af\left(\frac{n}{b}\right) \leq cf(n)$，對於一個 c < 1，以及足夠大的 n

- 把 T(n) = 2T(n / 2) + O(n log n) 帶進去看看

平面最近點對

- Master theorem case 3 的一個額外條件：

  $af(\frac{n}{b}) \leq cf(n)$，對於一個 c < 1，以及足夠大的 n

- 把 T(n) = 2T(n / 2) + O(n log n) 帶進去看看

- $2(\frac{n}{2}\log\frac{n}{2}) \leq cn\log n$

平面最近點對

- Master theorem case 3 的一個額外條件：

  $af(\frac{n}{b}) \leq cf(n)$，對於一個 c < 1，以及足夠大的 n

- 把 T(n) = 2T(n / 2) + O(n log n) 帶進去看看

- $2(\frac{n}{2}\log\frac{n}{2}) \leq cn\log n$

- $n\log n - n \leq cn\log n$

平面最近點對

- Master theorem case 3 的一個額外條件：

  $af(\frac{n}{b}) \leq cf(n)$，對於一個 c < 1，以及足夠大的 n

- 把 T(n) = 2T(n / 2) + O(n log n) 帶進去看看

- $2(\frac{n}{2}\log\frac{n}{2}) \leq cn\log n$

- $n\log n - n \leq cn\log n$

- 好像找不到 c QQQ

平面最近點對

- 來分析一下複雜度吧～
- T(n) = 2T(n / 2) + O(n log n)
  - 要把點按照 y 座標排序
- 根據 recursion-tree，這看起來是 O(n (log n)^2)
- 根據 master theorem，這看起來是 O(n log n) !?!
- 沒有辦法根據 master thoerem，上面那個遞迴的複雜度是 O(n (log n)^2) 沒錯

平面最近點對

- 不能作到更好嗎？！？

平面最近點對

- 不能作到更好嗎？！？
- 靈光一閃，想起 merge sort

平面最近點對

- 不能作到更好嗎？！？
- 靈光一閃，想起 merge sort
- 後面那個 O(n log n) ，可以用 merge sort 壓到 O(n)
- T(n) = 2T(n / 2) + O(n)
- T(n) = O(n log n) ！！

高維度(?)最近點對

- 剛剛考慮的點對，是在二維平面上的
- 那現在，如果點是三維空間，或者是更高維度的話
- ~~還能做嗎？~~

高維度(?)最近點對

- 剛剛考慮的點對，是在二維平面上的
- 那現在，如果點是三維空間，或者是更高維度的話
- ~~還能做嗎？~~還能用 Divide and Conquer 嗎？

- 練習題：https://tioj.ck.tp.edu.tw/problems/1918
  題解
  ：https://tioj.ck.tp.edu.tw/uploads/attachment/6/34/4_prog_ans.pdf

高維度(?)最近點對

- 想想看，在二維的時候，我們是怎麼 conquer 的？

高維度(?)最近點對

- 想想看，在二維的時候，我們是怎麼 conquer 的？
- 我們只考慮 y 座標相差 <= d 的點，根據稀疏性，這些點對的量集是 O(n) 的

高維度(?)最近點對

- 想想看，在二維的時候，我們是怎麼 conquer 的？
- 我們只考慮 y 座標相差 <= d 的點，根據稀疏性，這些點對的量集是 O(n) 的
- 從原本的二維問題，轉換成一維問題
- **關鍵:降低維度**

高維度(?)最近點對

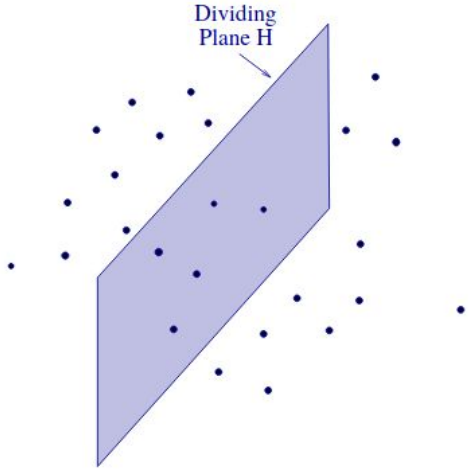
- 想想看，在二維的時候，我們是怎麼 conquer 的？
- 我們只考慮 y 座標相差 <= d 的點，根據稀疏性，這些點對的量集是 O(n) 的
- 我們不管 x 座標的影響，只考慮 y 座標
- 因為如果有更進的點對(距離 <= d)，只考慮 y 座標的話，他們的 y 座標差也一定會 <= d
- **關鍵：投影！**

三維最近點對

- 讓我們先來考慮三維最近點對

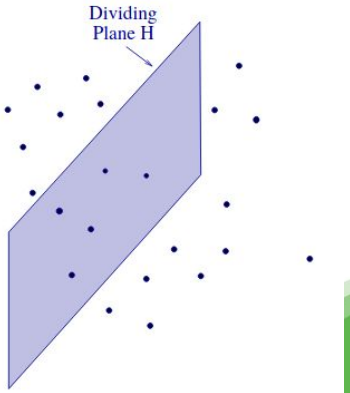
三維最近點對

- 讓我們先來考慮三維最近點對
- 根據上面的想法，我們可以先把點按照 x 座標切兩半，各自遞迴求出答案



三維最近點對

- 讓我們先來考慮三維最近點對
- 根據上面的想法，我們可以先把點按照 x 座標切兩半，各自遞迴求出答案
- 接下來考慮 conquer ，把這些點「投影」到你切的平面上（也就是說，只考慮 y，z 座標）



三維最近點對

- 讓我們先來考慮三維最近點對
- 根據上面的想法，我們可以先把點按照 x 座標切兩半，各自遞迴求出答案
- 接下來考慮 conquer ，把這些點「投影」到你切的平面上（也就是說，只考慮 y，z 座標）
- 遞迴解出這個「二維最近點對」的問題。找到一組點對的時候，再套回去三維，檢查距離是不是最小的

三維最近點對

- 來算一下複雜度吧～～
- $T(n) = 2T(n / 2) + O(n \log n)$

三維最近點對

- 來算一下複雜度吧～～
- $T(n) = 2T(n / 2) + O(n \log n)$
- $T(n) = O(n (\log n)^2)$

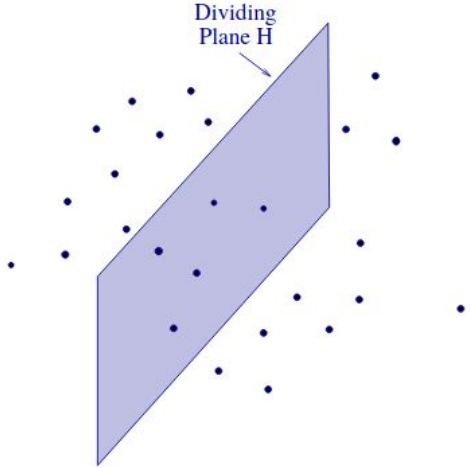
d維最近點對

- 根據前面三維最近點對的作法，我們來試著拓展成 d 維最近點對！

d維最近點對

- 根據前面三維最近點對的作法，我們來試著拓展成 d 維最近點對的作法！

- 在 d 維時，把空間用一個平面切開(可以想成把一個座標切成一半)，左右遞迴求出答案



d維最近點對

- 在 d 維時，把空間用一個平面切開(可以想成把一個座標切成一半)，左右遞迴求出答案
- 在 conquer 的時候，把所有的點投影到這個平面上(可以想成不考慮某一個座標)，變成 d - 1 維的最近點對問題
- 解決這個 d - 1 維的最近點對問題。找到一個合法點對時，記得用最一開始的維度更新答案！
- Reference:https://sites.cs.ucsb.edu/~suri/cs235/ClosestPair.pdf

d維最近點對

- 來算算複雜度吧～
- 假設 T(d, n) 是解決 n 個點中, d 維最近點對的複雜度
- 那麼, T(d, n) = 2T(d, n / 2) + T(d - 1, n) + O(n)

d維最近點對

- 來算算複雜度吧～
- 假設 T(d, n) 是解決 n 個點中, d 維最近點對的複雜度
- 那麼, T(d, n) = 2T(d, n / 2) + T(d - 1, n) + O(n)
- 體感一下, 可以發現 T(n) = O(n (log n)^(d - 1)) !
- 帶入二維、三維, 都符合條件～
- 更詳細的證明, 可以參考 https://sites.cs.ucsb.edu/~suri/cs235/ClosestPair.pdf

樹分治

- 在這之前，我們最常在序列上 divide and conquer
- 我們還有在矩陣上 divide and conquer
- 上一題中，我們在幾何平面上進行 divide and conquer
- 現在，我們要在樹上面進行 divide and conquer ！
- 先來看看一個例題吧～

樹分治

- 給你一棵樹，樹有邊權
- 請問有多少點對的距離 <= k
- a 跟 b 在樹上的距離就是 a 走到 b 的邊權和
- N <= 10^5

- 練習題：poj.org/problem?id=1741

樹分治

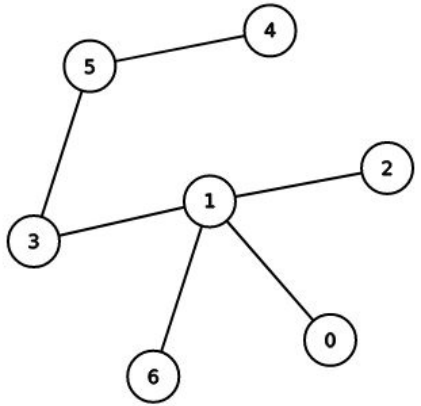
- 要怎麼做ㄋ？？？
- 枚舉 N^2 種路徑，O(N)算一下他們的長度，總複雜度O(N^3)

樹分治

- 要怎麼做ㄋ？？？
- 枚舉 N^2 種路徑，O(N)算一下他們的長度，總複雜度O(N^3)
- 枚舉起點，DFS or BFS遍歷整棵樹數有幾條路徑長<=k，O(N^2)
- 不過我們顯然要找到一個不用枚舉路徑的方法
- 這週的主題是分治，所以一定是要找一個分治的解法！

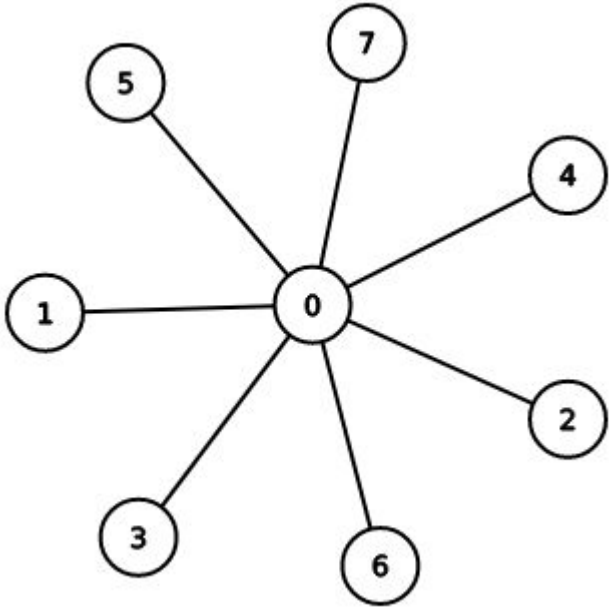
樹分治 -- 如何砍樹

- 我們先試著用「邊」來砍樹
- 根據前面的經驗，我們要試著把樹盡量分的平均(或者是說，盡量砍成一半)
- 有辦法嗎？？？



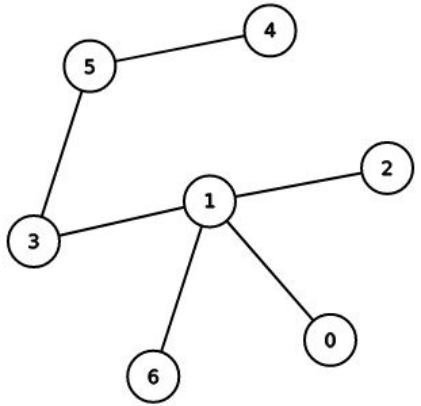
樹分治 -- 如何砍樹

- 星狀圖就慘掉了 >////<



樹分治 -- 如何砍樹

- 那那，我們只好用點來砍樹了 (?)
- 根據前面的經驗，我們要試著把樹盡量分的平均(或者是說，盡量砍成一半)
- 有辦法嗎？？？



樹分治 -- 如何砍樹

- 可以用點砍樹！
- 對於任何一棵樹，保證存在一個點，使得把這個點當成根的話，所有的子樹大小都會比 n / 2 小！
  - 可以自己想想簡單的證明喔～～
- 這個點又稱為樹的**重心**！
  - NEOJ 293 XD
- 所以，我們已經知道怎麼砍樹了！

樹分治

- 接下來，我們來試著套用前面分治的架構：
- 1．找出樹重心 g ，有多個的話就隨便挑一個

樹分治

- 接下來，我們來試著套用前面分治的架構：

- 1. 找出樹重心 g，有多個的話就隨便挑一個
- 2. Divide：把樹重心的所有兒子子樹們遞迴下去求解

樹分治

- 接下來，我們來試著套用前面分治的架構：

- 1. 找出樹重心 g ，有多個的話就隨便挑一個
- 2. Divide：把樹重心的所有兒子子樹們遞迴下去求解
- 3. Conquer：算出有通過樹重心，且長度 <= k 的路徑個數
  - 3.1：樹重心是其中一個端點
  - 3.2：樹重心不是端點

- 想想看我們還不會什麼？

樹分治

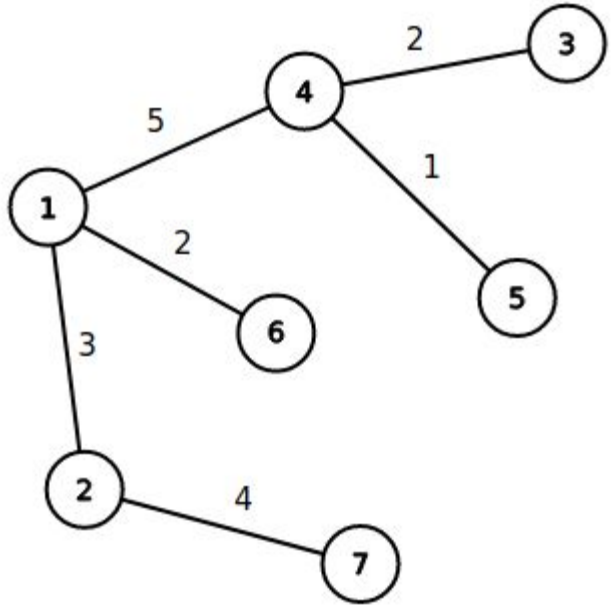
- 接下來，我們來試著套用前面分治的架構：

- 1. 找出樹重心 g，有多個的話就隨便挑一個
- 2. Divide：把樹重心的所有兒子子樹們遞迴下去求解
- 3. Conquer：算出有通過樹重心，且長度 <= k 的路徑個數
  - 3.1：樹重心是其中一個端點
  - 3.2：樹重心不是端點

- 1 你們應該會了(?)，2 也應該不難
- 3.1：從樹重心 dfs 這棵樹，看看距離有沒 <= k
- 好像只有 3.2 比較難！！

樹分治 -- 3.2

- 從 3.1 開始延伸，儲存樹重心到每個點的距離，並且紀錄他是屬於哪個兒子子樹的
- 例如，如果把 1 當樹重心的話，(距離，子樹) 就是

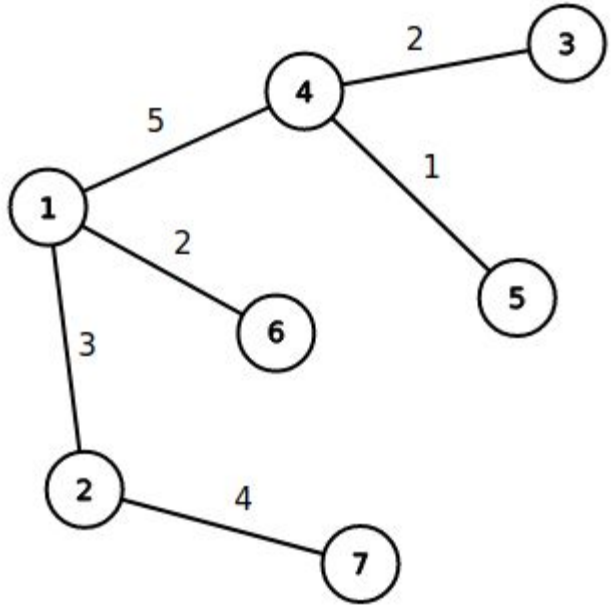


樹分治 -- 3.2

- 從 3.1 開始延伸，儲存樹重心到每個點的距離，並且紀錄他是屬於哪個兒子子樹的

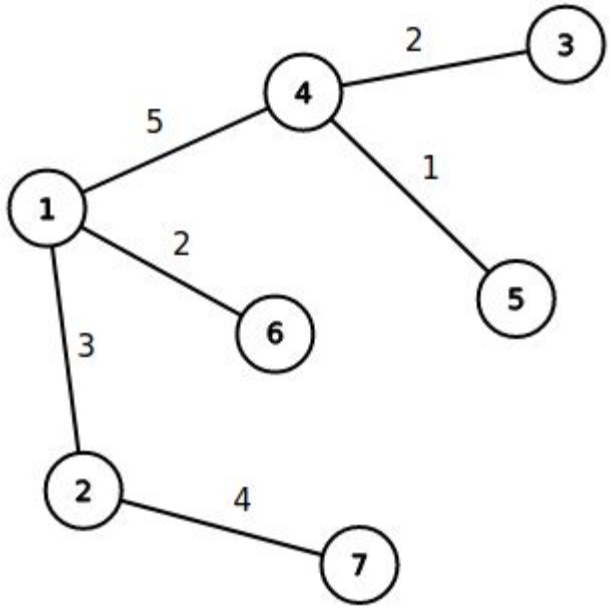
- 例如，如果把 1 當樹重心的話，(距離，子樹) 就是
  (3, 2), (7, 2),
  (2, 6),
  (5, 4), (6, 4), (7, 4)

- 把這些 pair 們排序好
  - 先比較第一個元素，一樣的話再比較第二個元素



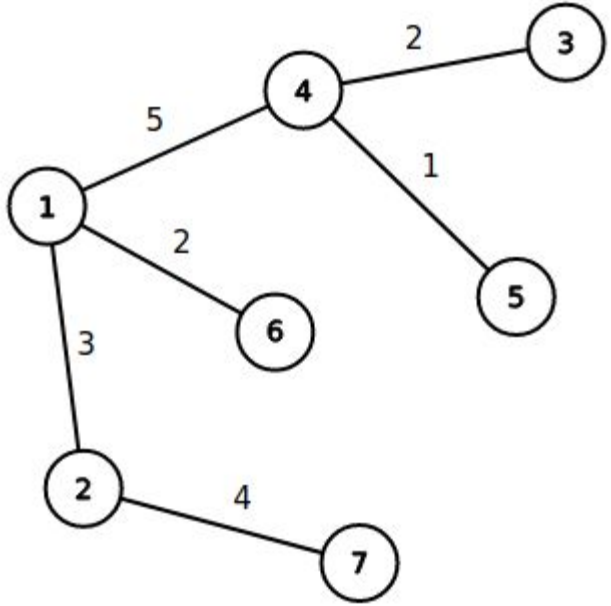
樹分治 -- 3.2

- (2, 6), (3, 2), (5, 4), (6, 4), (7, 2), (7, 4)
- 如果把任意兩個屬於不同子樹的 pair 放在一起看，就可以視為 **一條路徑**！
- 例如，把 (7, 2) 跟 (5, 4) 放在一起看的話，就是 點 7 到點 4 這條長度為12的路徑



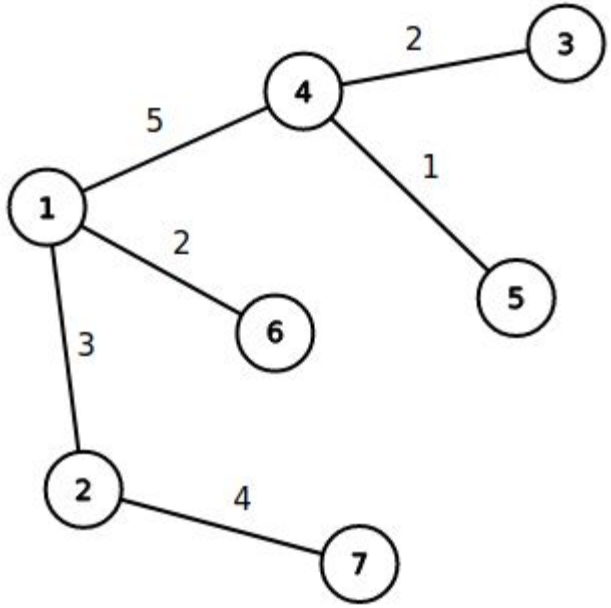
樹分治 -- 3.2

- (2, 6), (3, 2), (5, 4), (6, 4), (7, 2), (7, 4)
- 接著, 對於每個pair p, 可以找出某個右界, 使得 p 到這個右界之前的 pair , 距離總和都 <= k !
- 例如, 假設 k = 9 的話,
  (2, 6) 的右界就是 (7, 4)
  (3, 2) 的右界就是 (6, 4)
- 假設現在看到的pair p, 稱為左界



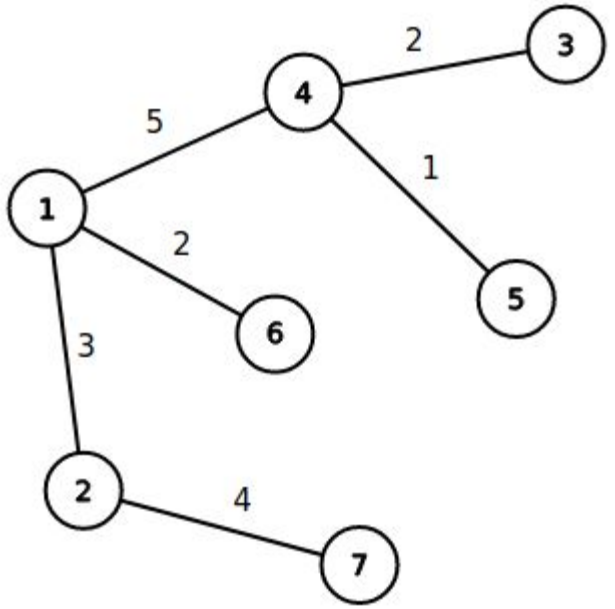
樹分治 -- 3.2

- (2, 6), (3, 2), (5, 4), (6, 4), (7, 2), (7, 4)
- 可以發現，當
  左界遞增時，右界會遞減
- 而這個右界，某種意義上就是答案要加上的數字(先不考慮來自相同子樹的問題)
- 因此，我們可以使用
  雙指標(two pointer)來解決



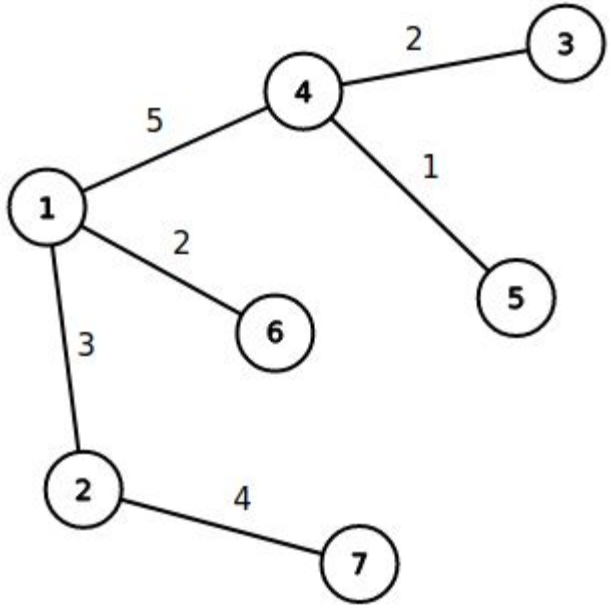
樹分治 -- 3.2

- (2, 6), (3, 2), (5, 4), (6, 4), (7, 2), (7, 4)
- 一開始把左界設成第一個元素，右界設成最後一個元素
- 每當看到一個左界是，把右界調整到正確的位置
- 例如，如果上面 k = 9 ，右界的移動，會按照 粉紅、紫、藍、棕 這個順序



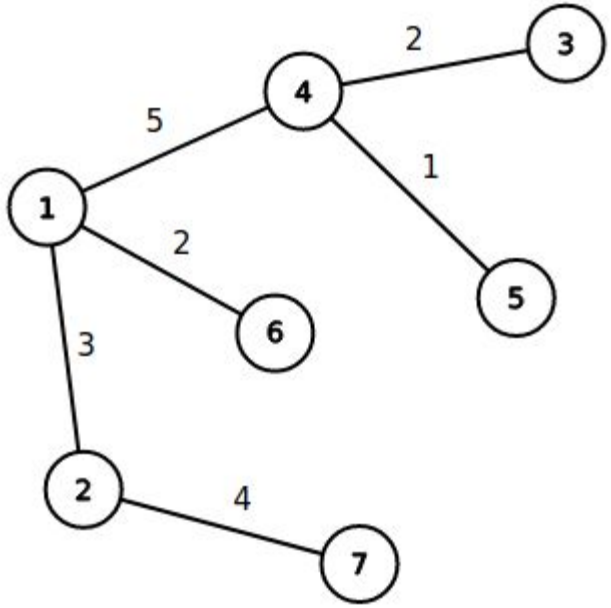
樹分治 -- 3.2

- (2, 6), (3, 2), (5, 4), (6, 4), (7, 2), (7, 4)
- 如果上面 k = 9 , 右界的移動,
  會按照 粉紅、紫、藍、棕
  這個順序
- 因此, 我們幾乎要算完 3.2 了,
  剩下的問題是要扣掉兩個 pair
  都是從同一個子樹來的 case



樹分治 -- 3.2

- (2, 6), (3, 2), (5, 4), (6, 4), (7, 2), (7, 4)
- 因此，我們幾乎要算完 3.2 了，
  剩下的問題是要扣掉兩個 pair
  都是從同一個子樹來的 case
- 開個 cnt 陣列，紀錄從每個子樹
  來的pair的數量，在算答案的時候，
  扣掉 cnt[左界子樹]
  - 因此，右界在移動時，要更新 cnt！
- 於是乎，我們就解出這題了～～～



樹分治

- 於是，經歷千辛萬苦，我們終於解決這題了～
- 來分析複雜度吧～

樹分治

- 於是，經歷千辛萬苦，我們終於解決這題了～
- 來分析複雜度吧～
- T(n) = ?????? + O(N log N)
- 不會分析了，OAO (?)

樹分治

- 於是，經歷千辛萬苦，我們終於解決這題了～
- 來分析複雜度吧～
- T(n) = ?????? + O(N log N)
- 解決方法 1 :用 recursion tree
- 可以想想，O(N log N) 至多只會出現幾層？

樹分治

- 於是，經歷千辛萬苦，我們終於解決這題了～
- 來分析複雜度吧～
- T(n) = ?????? + O(N log N)
- 解決方法 1 :用 recursion tree
- O(N log N) 至多只會出現 log N 次(因為每次都會分至少一半)，因次 T(n) = O(n (log n)^2)

樹分治

- 於是，經歷千辛萬苦，我們終於解決這題了～
- 來分析複雜度吧～
- T(n) = ?????? + O(N log N)
- 解決方法 2 ：直接帶入 Worst Case

樹分治

- 於是，經歷千辛萬苦，我們終於解決這題了～
- 來分析複雜度吧～
- T(n) = ?????? + O(n log n)
- 解決方法 2 ：直接帶入 Worst Case
- T(n) = 2T(n / 2) + O(n log n)
- T(n) = O(n (log n)^2)

樹分治 -- 實做

- 有很多細節需要小心
- 算子樹的時候，可以把拔掉的點設成不能走，各種dfs的時候不要走到被拔掉的點上面
- 注意一下遞迴的時候做事情的順序，有些全域陣列可能遞迴下去之後會被改掉
- 寫過就知道了XD

樹分治的應用

- 基本上，如果需要在樹上分治的話
- 題目不會這麼簡單XD
- 通常還會需要搭配「重心剖分」這個技巧
- 有興趣可以自行上網搜尋這個技巧XD
  - 或者是私底下詢問講師ww

總結分治

- 已經看過很多在序列、平面、樹上分治的題目了
- 但是其實分治還能在奇怪(?) 的地方分治
- 往往分治還需要搭配很多噁心的資料結構、演算法
- 分治往往是一場比賽中偏難的題目
- 大家加油～～