隨機算法

Slido: 4945896

課程會在 14:05 開始～

by boook

Hash Credit by qazwsxedcrfvtg14

大綱

- When
  - 為甚麼、何時要用隨機
- How
  - 如何產生隨機的數字
  - Hash
  - 隨機選擇
- Algorithm
  - 最近點對
  -

為甚麼、何時用隨機

- 模板很難寫
- ~~大家都喜歡唬爛~~
- AC 的效率 vs 出題者的良心

- 在題目限制很寬的時候
- 當出題者的題目出壞掉的時候

Sprout

用隨機的後果

- 犧牲正確率
  - 可能程式不總是正確
    - -> 讓程式錯誤的機率特別低
-
- 犧牲執行時間
  - 執行時間不固定
    - -> 讓程式期望執行的時間不長

如何產生隨機的數字

- 大家都會??
- `rand()`
- `srand(time(0))`
- 這樣足夠安全嗎？
  - 哪裡不安全

Sprout


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

int n = 1000000, arr[1000000 + 10];
int main() {
    double sum = 0, dis = 0;
    for (int i = 0; i < n; ++ i)
        arr[i] = i;

    random_shuffle(arr, arr + n);
    for (int i = 0; i < n; ++ i) {
        sum += rand() % n;
        dis += abs(arr[i] - i);
    }
    cout << "Average value: " << sum / n << '\n';
    cout << "Average distance: " << dis / n << '\n';
    return 0;
}
```

執行結果

```
Average value: 16402.3
Average distance: 62323.3
```

預期結果

```
Average value: 500000
Average distance: 333333
```

不要直接使用 rand()

- rand() 在 windows 上預設的回傳範圍為 [0, 65535]

- 解決方法:
  - rand() 很多次後將結果拼接成比較大的數字
  - 用其他亂數產生器, 如 mt19937
  - 追求更快效率 -> 自己寫亂數

Sprout

使用其他寫好的函式庫

- Mt19937

```
int n = 1000000, arr[1000000];
void Use_mt19937() {
    mt19937 rng(chrono::steady_clock::now().time_since_epoch().count());
    shuffle(arr, arr + n, rng);
}
```

自己實作

```c
static inline uint64_t rotl(const uint64_t x, int k) {
    return (x << k) | (x >> (64 - k));
}

static uint64_t s[2] = {1234, 4321}; // init seed

uint64_t RandomBigInt() {
    uint64_t s0 = s[0], s1 = s[1], result = s0 + s1;
    s1 ^= s0;
    s[0] = rotl(s0, 24) ^ s1 ^ (s1 << 16);
    s[1] = rotl(s1, 37);
    return result;
}
```

How to shuffle

```
void shuffle() {
    for (int i = 1; i < n; ++ i)
        swap(arr[i], arr[next() % i]);
}
```

暖身 - 計算隨機正確率

- 題目一共 T 筆測資，每筆測資有 Q 筆詢問，
- 每筆詢問，你跑了 k 次演算法取最好值
- 每次演算法有 p 個機率會回答正確的答案
  - 每次演算法錯誤的機率是 $1-p$
  - 每筆詢問錯誤的機率是 $(1-p)^k$
  - 每筆測資正確的機率是 $(1-(1-p)^k)^Q$
  - 上傳後 AC 的機率是 $(1-(1-p)^k)^{QT}$

Hash

Sprout

Hash

- 目的:
  - 將一個物品的很多資訊壓縮起來，當要判斷兩個物品是否相同時就判斷兩個物品分別壓縮後的結果是否相同。
- 壓縮過程中會失去一些資訊，因此可能會有兩個不同的物品被壓縮後產生一樣的結果。

- 如果有兩個物品 A，B ，以及壓縮函數 f
  - 錯誤的機率:
    - f(A) != f(B)，但 A = B。 => 不可能
    - f(A) = f(B)，但 A != B。 => 錯誤的機率

Hash

- 譬如:
  - `f(x) = x % 2`
- 所以我們會認為:
  - `1 != 2, 因為 1 = f(1) != f(2) = 0`
  - `2 = 2, 因為 0 = f(2) = f(2) = 0`
- 但是我們會認為:
  - `4 = 2, 因為 0 = f(4) = f(2) = 0`

f(A) = f(B), 但 A != B

- 好的 hash function 要滿足「碰撞」的機率足夠小, 也就是 f(A) = f(B), 但 A != B 的機率。
- P(f(A) = f(B), 但 A != B) < 1e-9
  - 當錯誤的機率「足夠小」, 那麼就可以「假裝」hash function 不會碰撞
  - 有時候 1e-9 仍然不夠, 需視情況調整
  - 生活上的例子 (checksum)

Sprout

- 可以構造讓 hash table 變得非常慢，但在隨機的使用下，insert、delete 都是 O(1)。

```cpp
#include <unordered_set>
#include <unordered_map>
using namespace std;

unordered_set<int> u2;
unordered_map<int, int> u1;

#include <ext/pb_ds/assoc_container.hpp>
using namespace __gnu_pbds;
cc_hash_table<int, int> cc;
gp_hash_table<int, int> gp;
```

Hash 安全性

- 對於一個 hash function x -> f(x) 有三種等級的安全性:
  - Preimage attack:
    - 知道 f(x) 還原出 x
  - Second preimage resistance:
    - 在知道 f(x) 的情況下, 找到另一組 y -> f(y) = f(x)
  - collision resistance:
    - 找到一對 a, b 使得 f(a) = f(b)

Hash 的安全性 - 生日悖論

- 一個房間要多少人，則兩個人的生日相同的機率大於 50%?
  - 23人
- 有 n 個人，每人都隨機地從 N 個特定的數中選擇出來一個數。
  - p(n) = 有兩個人選擇了同樣的數字。
- 這個機率有多大?

$$p(n) \sim 1 - 1/\exp(n^2/(2N))$$

Rolling hash

- 又稱 RK 算法(Robin-Karp Algorithm)
- 核心

$$H(s) = \sum_{i=1}^{n} s_i \times C^{n-i} \mod M$$

- 也可以用來當作亂數產生器

Rolling hash - 如果 C = 10

- S = 21334531
- M = 1000000000000000000000000000000000000000

- H(1) = 2
- H(2) = 21
- H(3) = 213
- H(4) = 2133
- H(5) = 21334
- H(6) = 213345
- H(7) = 2133453
- H(8) = 21334531

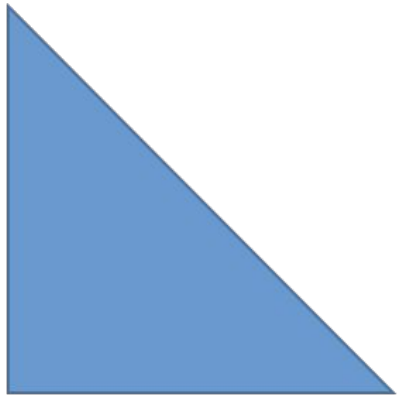
Rolling hash

$$H(s) = \sum_{i=1}^{n} s_i \times C^{n-i} \mod M$$

- 假設你有一個字串，且你已經知道其hash值
- 下列哪些操作可以 O(1) 執行?
- 加一個字元在前面
- 加一個字元在後面
- 從前面刪除一個字元
- 從後面刪除一個字元

Rolling hash

- 可以把 Rolling hash 想像成一個前高後低三角形

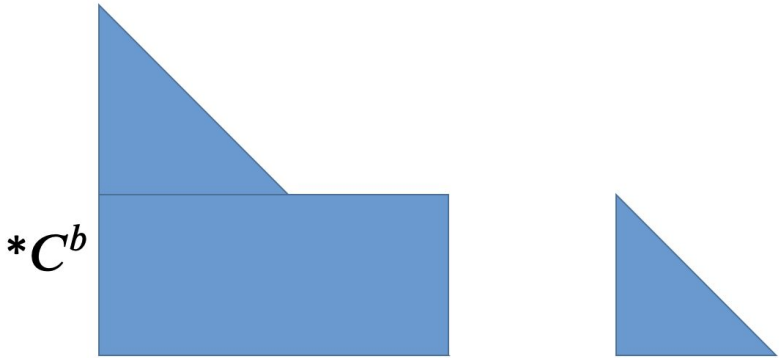


```
H(1) = 2
H(2) = 21
H(3) = 213
H(4) = 2133
H(5) = 21334
```

Rolling hash

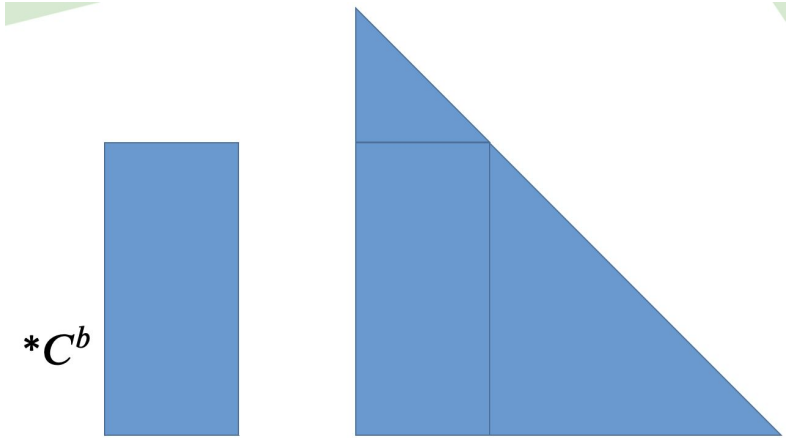
- 兩個 hash 接起來
- 一個字元也可以算是一個hash

- S = 213
- P = 123
-
- H(S) = 213
- H(P) = 123
- H(S+P) = 213 * 1000 + 123



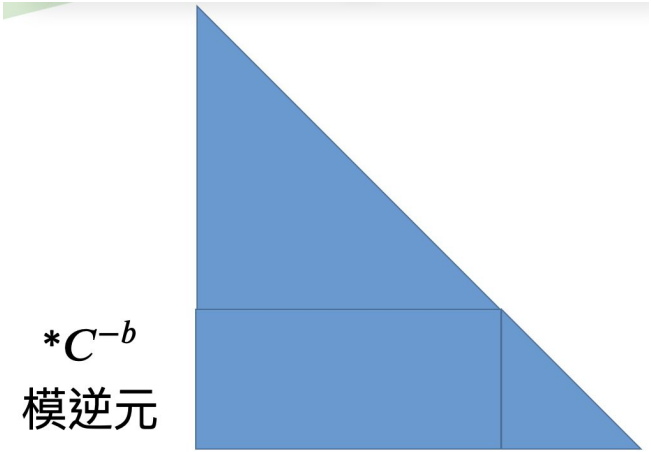
Rolling hash

- 刪除前面的字元
- S = 12345
-
- H(S) = 12345
- 12345 - 12000
- = 345



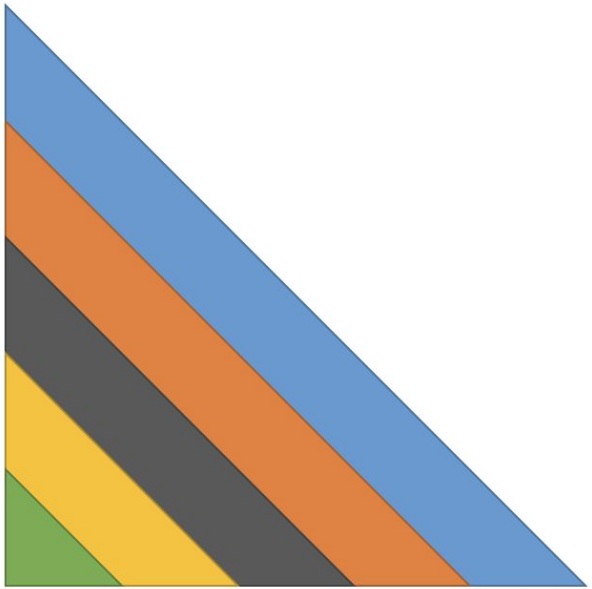
Rolling hash

- 刪除後面的字元
  - S = 12345
  - H(S) = 12345
  - (12345 - 45) / 100 = 123
- Why 模逆元？
  - 8 / 2 != 2 / 2 (mod 6)



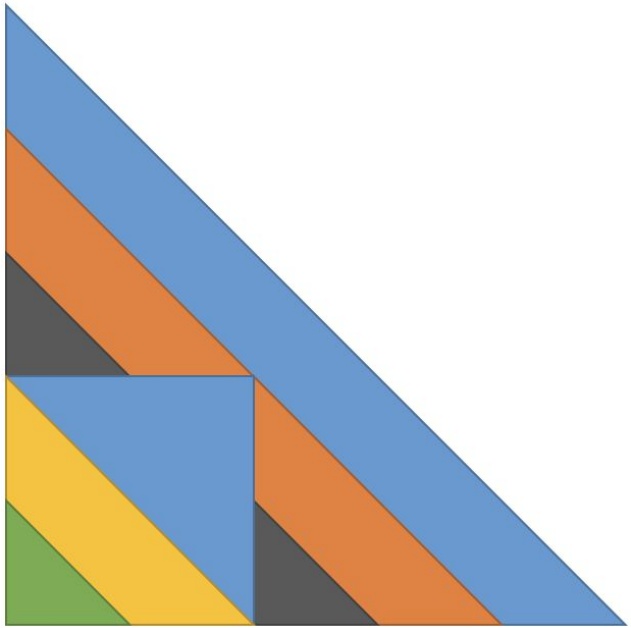
Rolling hash

- 因為這些性質
- 我們經常會預處理好字串每個前綴的Rolling hash



Rolling hash

- 這樣之後就能直接算出任意一段的hash值



S = 21334
H(1) = 2
H(2) = 21
H(3) = 213
H(4) = 2133
H(5) = 21334
H(3~4)
= H(4) – H(2) * 100 = 33

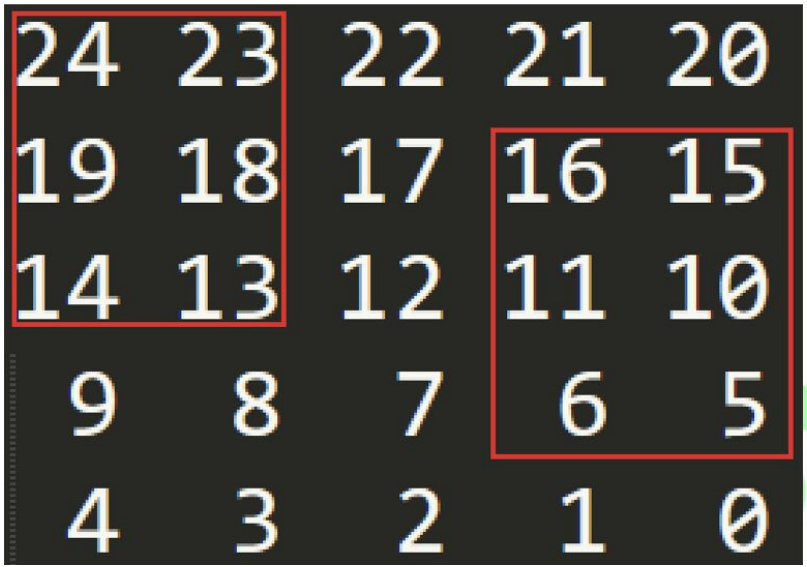
例題 - 怎麼訂 hash 值

- 給一個矩陣大小為 N * N
- Q 次詢問兩個大小相同的子矩陣是否相同。
  - N <= 5000
  - Q <= 500000

- hash 可不可以推廣到二維？

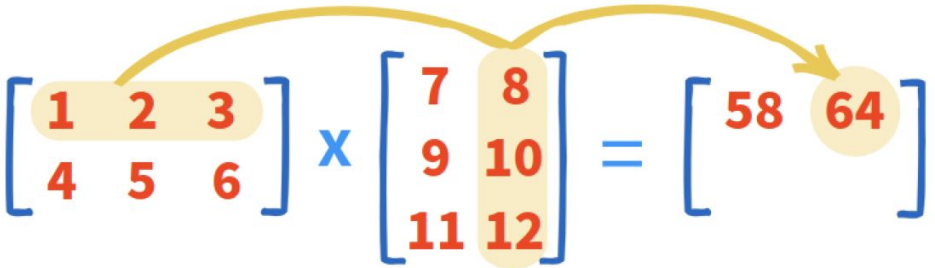
二維 hash

$$H(s) = \sum_{i=1}^{n} s_i \times C^{n-i} \mod M$$



例題 – 矩陣相成的結果

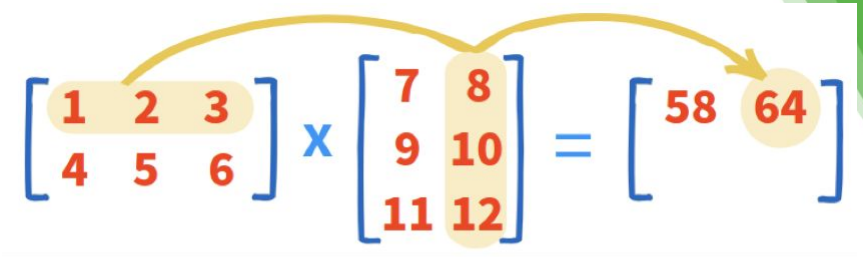
- 給定三個大小為 N 的 N × N 矩陣 A、B、C, 請問 A × B = C 是否成立？
  - N <= 1000



- 直接做是 O(N^3)
- 能不能更快一點?

例題 – 矩陣相成的結果

- 讓 R 是一個 N × 1 的矩陣，那麼只需要檢查
- ((A×B))=(C)



- ((A×B)×R)=(C×R)=C′
- (A×(B×R))=(C×R)=C′

$$H(s) = \sum_{i=1}^{n} s_i \times C^{n-i} \mod M$$

- 乘法過程中與字串 hash 做的事情相同

例題 – 成雙成對

- 給出一個 N 個數字的序列 a_1, a_2, ..., a_n。現在有 Q 個詢問，每次詢問會給出 l, r, 請問如果考慮 a_l, ..., a_r 是否每種數字都出現偶數次？
  - N <= 1,000,000
  - Q <= 1,000,000

例題 – 成雙成對

- XOR 好性質:
  - X ⊕ Y ⊕ Z ⊕ Y ⊕ X ⊕ Z = 0
- XOR 不夠好性質:
  - 1 ⊕ 2 ⊕ 3 = 0
- One time pad:一次性密碼本
- N 個 ∈ [0, 2^k) 的隨機數字 XOR 起來的結果是隨機的
- 隨機將 a_i 映射成一個隨機的大數字
- 每次錯誤率 1 / (2^k), 讓 k = 60

例題 – 成堆成堆

- 給出一個 N 個數字的序列 a_1, a_2, ..., a_n。現在有 Q 個詢問，每次詢問會給出 l, r，請問如果考慮 a_l, ..., a_r 是否每種數字都出現 K 的倍數次？
  - N <= 1,000,000
  - Q <= 1,000,000

例題 – 成堆成堆

- **設計一個** hash function
  - XOR **可以 加法也可以**

- **只要滿足**:
  - 答案是 YES 的時候一定判斷得出來
  - 答案是 NO 的時候有很低的機率會錯

例題 – 成堆成堆

- **設計一個 hash function:加法**

1	1	1	1	1	1	1
x	x	-2x	x	x	-2x	x

1	1	2	1	3	1	2	2	2	3
x	x	y	-2x	z	x	y	-2y	y	z

- **改成在模 p 底下做事情避免溢位**
- **單筆詢問答對機率: 1 / p**

例題 – 長度為 8 的簡單路

- 給 N 點 M 邊的有向圖，並給出兩點 S 和 T
- 請找一條 S 到 T 且含頭尾恰經過 8 個點的簡單路徑
- 簡單路徑：路徑中不存在兩個相同的點
-
- 1 <= N <= 100, 1 <= M <= 2000

例題 – 長度為 8 的簡單路

- 也就是說不含頭尾一共 6 個點 ~~=> O(n^6)~~
-
- 問題在簡單路徑上，不然就可以？
  - 為什麼不是簡單路徑就可以做

例題 – 長度為 8 的簡單路

- 也就是說不含頭尾一共 6 個點 ~~=> O(n^6)~~
-
- 問題在簡單路徑上，不然就可以？
  - 為什麼不是簡單路徑就可以做
-
- Dp[point][step] = True or false

例題 – 長度為 8 的簡單路

- 那如果現在點上面有顏色呢
-
- 給 N 點 M 邊的有向圖，並給出兩點 S 和 T。
  圖上除了 S 和 T 之外每個點 i 都被塗上了顏色 C_i。
  請找一條 S 到 T 且含頭尾恰經過 8 個點的簡單路徑，
  並在路上蒐集滿 6 種不同的顏色
-
- 1 <= N <= 100, 1 <= M <= 2000, 0 <= C_i <= 5

例題 – 長度為 8 的簡單路

- 給 N 點 M 邊的有向圖，並給出兩點 S 和 T。
  圖上除了 S 和 T 之外每個點 i 都被塗上了顏色 C_i。
  請找一條 S 到 T 且含頭尾恰經過 8 個點的簡單路徑，
  並在路上蒐集滿 6 種不同的顏色
-
- 1 <= N <= 100, 1 <= M <= 2000, 0 <= C_i <= 5
-
- Dp[point][bit_mask] = True or False
  - O(point bit_mask) O(轉移) = O(2^6 M)

例題 – 長度為 8 的簡單路

- 如果點有顏色多好 => 自己上色
- 隨機把每個點塗上 C_i = $x \in [0, 5]$
- 至少多少機率會答對:
  - 如果只有一條符合條件的路徑
    - 中間六個點的顏色都要不一樣: 6!
    - 總方法數量: 6^6
- 隨機一次會答錯的機率: $1 - \frac{6!}{6^6}$

例題 – 長度為 8 的簡單路

不會對到不會錯

- 隨機 1 次的錯誤率：$(1-\frac{6!}{6^6})^1 \approx 0.9845$
- 隨機 10 次的錯誤率：$(1-\frac{6!}{6^6})^{10} \approx 0.8559$
- 隨機 100 次的錯誤率：$(1-\frac{6!}{6^6})^{100} \approx 0.2111$
- 隨機 300 次的錯誤率：$(1-\frac{6!}{6^6})^{300} \approx 0.0094$
- 隨機 1000 次的錯誤率：$(1-\frac{6!}{6^6})^{1000} \approx 1.7606 \times 10^{-7}$

隨機選擇

- How:
  - 字面上的意思
  - 隨機選擇一條路 => 估計答對的機率

隨機選擇 - 例題

- 給定 N 個二為平面上面的點，請問有沒有一條通過至少 k % 的直線？
  - 2 <= N <= 2 * 10^6
  - k >= 20

直觀的想法

- 直觀的想法
- 1．隨機兩個點
- 2．檢查有多少個點在兩點構成的直線上面
- 3．做很多次

- 重複隨機選擇　兩點 50 次，錯誤率

$$(1 - 20\% \times 20\%)^{50} \approx 12.98\%$$

隨機少一點「維度」

- 1. 隨機一個點　a
- 2. 以 a 點作為圓心的極角進行排序
- 3. 做很多次
-
- 重複隨機選擇 a 點 40 次，錯誤率

$$(1 - 20\%)^{40} \approx 0.013\%$$

Sprout

錯排數 (Derangement)

錯排數 D_N ：滿足對於所有的$1 \leq i \leq N$ ，$A_i \neq i$ 這樣的 permutation

$D_N / N! \approx 1/e \approx 0.367879$

也就是說有大概 1/3 的機率隨機一個 permutation 可以隨機到一個 Derangement。

隨機選擇 CF330E

- 給出一張 N 個點 M 條邊的無向圖，並且圖上保證每個點最多只接了兩條邊，請構造出一張新圖滿足以下條件，若無解則輸出 -1 ：
  - 新圖上與原圖有相同數量的點和相同數量的邊。
  - 圖上保證每個點最多只接了兩條邊。
  - 對於圖上兩個點，若在原圖上這兩點之間存在一條邊，則在新圖上這兩點之間必定沒有邊。

$$1 \leq M \leq N \leq 10^5$$

隨便做?

- 把圖的編號隨機打亂　　=> 做完了！
-
- 估計錯誤率 => 想想看什麼樣的測資容易錯
  - 題意裡面的圖為很多環和很多鍊
    - 加一條邊 = 加上一個限制 (很多環)
  - 很多連通塊會導致條件被獨立分開
    - 圖是一個很大的環

- 拓展錯排數：滿足對任意的相鄰的兩數字差不是 1 或 N – 1 條件的排列 p_1, p_2, ..., p_n

性質

- 滿足條件的排列 / N! = e^-2

$$(1-e^{-2})^{90} \approx 2.07\times10^{-6}$$

- 可以寫個簡單的程式估計一下:滿足條件的排列 / N!

最近點對

最近點對

- 給定 N 個二維平面上面的點，請找出距離最近的兩個點。

- 1 <= N <= 10^5

- ~~傳說做法？旋轉排序~~

最近點對

- 1. 把 N 個點平移到第一象限
- 2. 把 N 個點的順序隨機打亂, d = dis(a1, a2)
- 3. 以 r = d / 2 的大小將二維平面切成網格狀, 並將這些網格也以座標表示:點（x, y）會落入座標為（x/r,y/r）
- 不會有兩個點在同一個格子內

最近點對

- 4. 一直把點加進網格中
  - 若產生新的最近點對一定會出現在以某個點為中心的 5 * 5 宮格內

d/2~d	d/2~d	d/2~d	d/2~d	d/2~d
d/2~d	0~d/2	0~d/2	0~d/2	d/2~d
d/2~d	0~d/2	**某個點**	0~d/2	d/2~d
d/2~d	0~d/2	0~d/2	0~d/2	d/2~d
d/2~d	d/2~d	d/2~d	d/2~d	d/2~d

最近點對

- 5.找到更近的最近點對:
  - 更新 r = 新的最近點對 / 2
  - 回到步驟三重來 O(i+1)

最近點對

- 期望複雜度
  - 令 d_i 為考慮 $a_1, a_2, \ldots, a_i$ 這些點的最近點對距離。
  - 當加入點 a_{i+1} 時，出現一個 a_j 使得 (a_j, a_{i+1}) 是新的最近點對的機率為：
  - $$P\left(dis\left(a_j, a_{i+1}\right) < d_i\right) = \frac{i}{C_2^{i+1}} = \frac{i \times 2}{i \times (i+1)} = \frac{2}{i+1}$$
  - 出現一個 a_j 要付出的時間代價： $O(i+1)$
  - 每加入一個點考慮期望複雜度為

    $$\text{機率} \times O(\text{代價}) = \frac{2}{i+1} \times (i+1) = 2 = O(1)$$
  - 總共要加 N 個點 => O(N)

- End