ชยุตม์ อังค์วัฒนะ 5130114321

ชลภัค ณรงคนานุกูล 5130117221

Algorithm Design Lecture note 2

Closest pair

กำหนดจุดในสองมิติ สิ่งที่เราต้องการรู้คือคู่ที่อยู่ใกล้กันมากที่สุด

วิธีแรก → ลองทุกคู่

มีทั้งหมด $n(n+1)/2$ คู่

คู่หนึ่งใช้ $\Theta(1)$

ดังนั้น ใช้เวลาทั้งหมด $\Theta(n^2)$

วิธี Divide& Conquer

หาคำตอบจากปัญหาที่เล็กกว่า → โดยการแบ่งครึ่ง

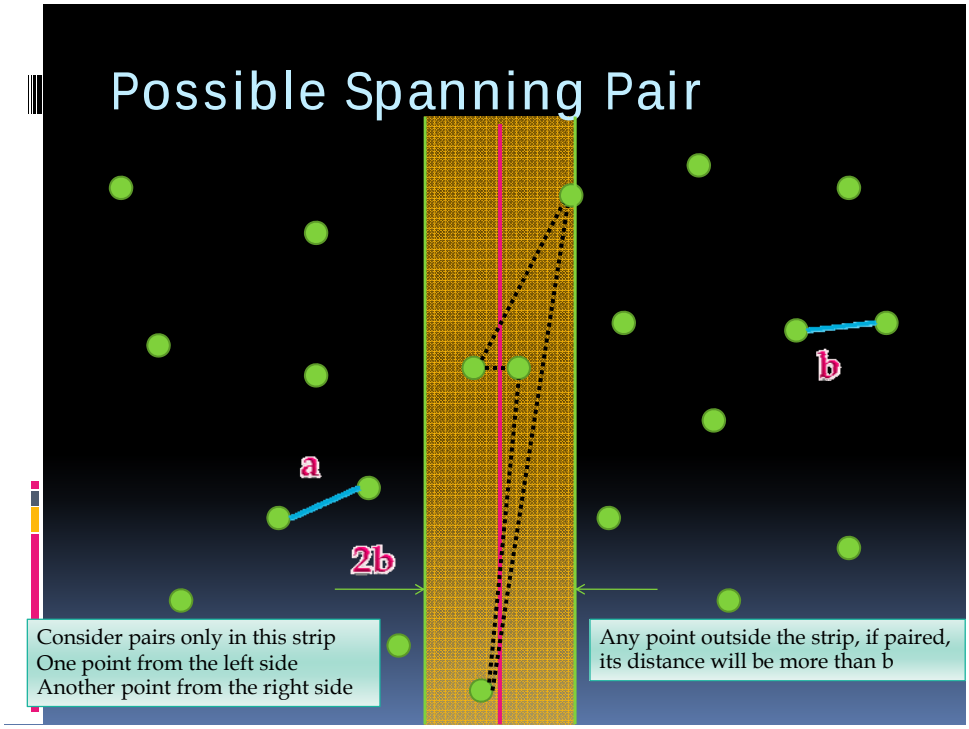
Divide → แบ่งตามแกน x

โดยหาคู่ใกล้ที่สุดทั้งฝั่งซ้ายและขวา

Conquer → เหมือนกับปัญหาเรื่อง MSS คือ คำตอบของปัญหาย่อยๆไม่ได้ครอบคลุมคู่ที่เป็นไปได้ทั้งหมด ต้องดูคู่ที่อยู่ระหว่างขอบเขตทั้งสองฝั่งด้วย

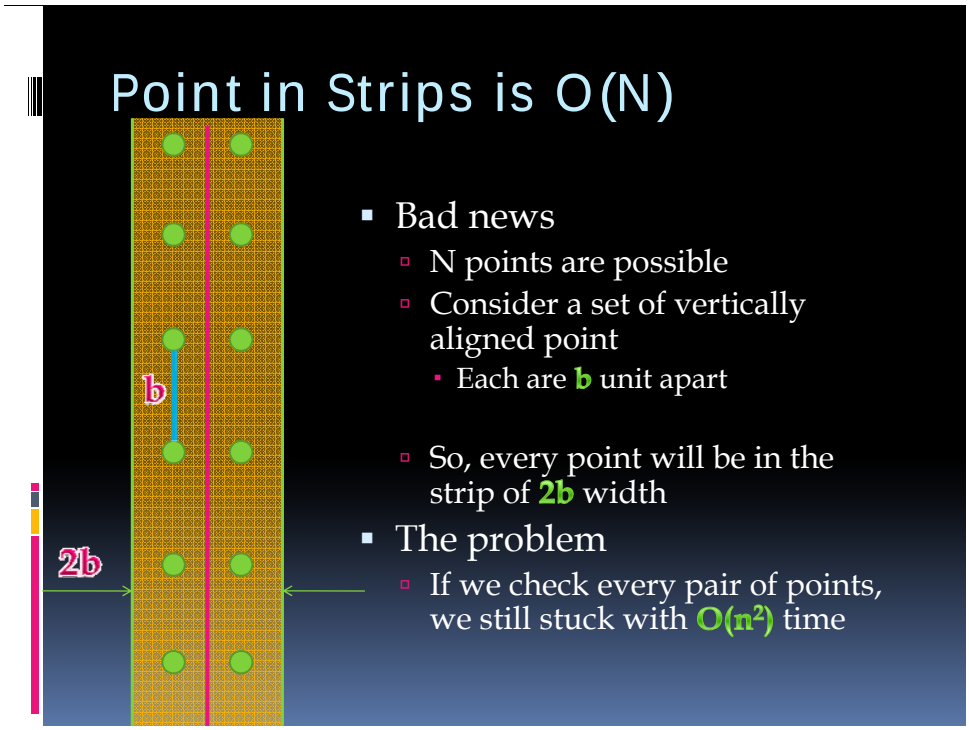
ดังนั้นมีคู่ที่ต้องเช็ค $(n/2)^2$ คู่ ก็ยัง $O(n^2)$ อยู่ดี มีวิธีที่ดีกว่านี้มั้ย?

คู่ที่อยู่ไกลกันมากๆ ไม่จำเป็นต้องเช็คก็ได้



หาระยะของคู่ที่ใกล้ที่สุดของแต่ละฝั่ง(b)แล้วกำหนดขอบเขตจากแกนแบ่งฝั่งละ b เป็น 2b เพื่อที่จะหาคู่ที่อยู่ใกล้กันมากกว่านั้น แบ่งแบบนี้คู่เช็คควรจะน้อยกว่า n มั้ย?

ยังมีกรณีแย่สุด



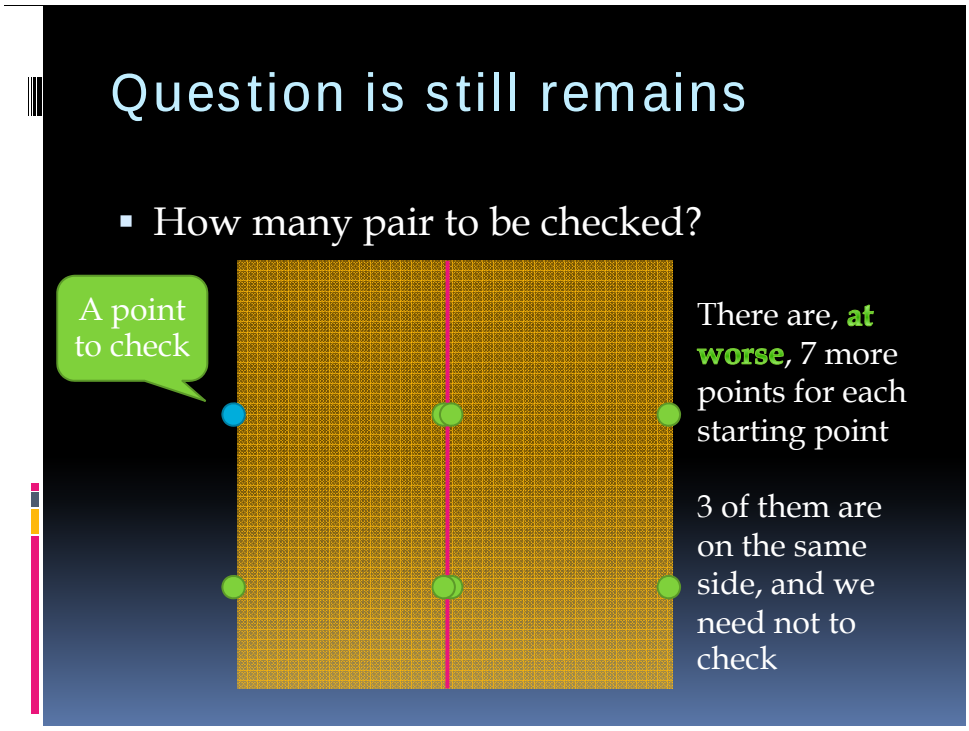
คือทุกตัวห่างกัน b และอยู่ในขอบเขตหมด ทำให้ต้องเช็คทุกตัวอยู่ดี สุดท้ายแล้วก็ $O(n^2)$

ลองใหม่ เราจำเป็นต้องเช็คทุกจุดในแถบขอบเขตมั้ย?

ไม่จำเป็น เราไม่จำเป็นที่จะต้องเช็คคู่จุดในแถบที่ห่างกันมากกว่า b !

พิจารณาระยะห่างทางแกน x เฉพาะที่อยู่ในแถบ และระยะห่างทางแกน y ที่ห่างกันไม่เกิน b

ยังมีกรณีที่แย่มากคือ



แย่สุดคือต้องเช็คถึง 7 จุด แต่3จุดในนั้นอยู่ฝั่งเดียวกัน เราไม่จำเป็นต้องเช็ค

ในทางปฏิบัติเราเช็คแค่ 4 จุดที่อยู่ฝั่งตรงกันข้ามกัน

ถ้าเราวนลูปเช็คระยะห่างทางแกน y ก็ยังตั้ง O(n)

เพราะฉะนั้นเราต้อง sort จุด !

วิธีแรกใช้ recursive ใช้ $O(n \lg^2 n)$

วิธีที่ดีกว่านั้น sort จุดตามแกนx โดยแยกทำได้ O(1) ดังนั้นจุดก้จะ sort ตามแกน y ด้วย เมื่อเช็คจุดในแถบ พอเจอค่าyที่ห่างกันเกินไปก็จะไม่พิจารณา

การ sort ทั้ง2 ครั้งใช้เวลา O(n lg n)

นำข้อมูลเข้าฟังก์ชั่นที่แบ่งเป็น 2 แบบคือแบบเรียงตามแกนxและแกนy โดยใช้เวลา O(n)

สรุป

T(n) = 2 T(n/2) (ฝั่งซ้ายขวาที่แบ่งตามแกนx)+ 4O(n)(จุดที่เช็คในแถบ) + O(n)(แบ่งลิสต์sort ตามแกน x,y)

$= \Theta(n \log n)$

โค้ดของอาจารย์

```
float void closet(Point *A, int n){

  Point *l,*r;

  // Sort A by X-Axis O(n log n)

  for(int i=0 ; i<n/2 ; i++) l[i] = A[i];

  for(int i=n/2 ; i<n ; i++) r[i] = A[i];



  float cl = closet(l,n/2);

  float cr = closet(r,n/2);

  float b = min(cl,cr);
```

```
  //Sort l,r by Y-Axis O(n log n)



  for(int i=0 ; i<n/2){

    int j=0;

    if(l[i] in orange-stripe){

      while(l[i].y - r[i].y <b){

          //check

          j++;

      }

    }

  }

}
```

การคูณเมทริกซ์

กำหนด A และ B เป็นเมทริกซ์ที่มีขนาด n x n

ให้ C = AB

โดยที่ $C_{i,j} = \sum(A_{i,k} * B_{k,j})$

Naïve Method

```
for (i = 1; i <= n;i++) {
  for (j = 1; i <= n;j++) {
    sum = 0;
    for (k = 1;k <= n;k++) {
      sum += a[i][k] * b[k][j];
    }
    c[i][j] = sum;
  }
}
```

ใช้เวลาการทำงานเป็น $O(n^3)$

วิธี Divide & Conquer

มองเมทริกซ์เป็น 4 ส่วน (ซ้ายบน, ซ้ายล่าง, ขวาบน, ขวาล่าง)

$$\mathbf{A} = \begin{bmatrix} \mathbf{A}_{1,1} & \mathbf{A}_{1,2} \\ \mathbf{A}_{2,1} & \mathbf{A}_{2,2} \end{bmatrix}, \mathbf{B} = \begin{bmatrix} \mathbf{B}_{1,1} & \mathbf{B}_{1,2} \\ \mathbf{B}_{2,1} & \mathbf{B}_{2,2} \end{bmatrix}, \mathbf{C} = \begin{bmatrix} \mathbf{C}_{1,1} & \mathbf{C}_{1,2} \\ \mathbf{C}_{2,1} & \mathbf{C}_{2,2} \end{bmatrix}$$

โดยที่

$C_{1,1} = A_{1,1} B_{1,1} + A_{1,2} B_{2,1}$

$C_{1,2} = A_{1,1} B_{1,2} + A_{1,2} B_{2,2}$

$C_{2,1} = A_{2,1} B_{1,1} + A_{2,2} B_{2,1}$

$C_{2,2} = A_{2,1} B_{1,2} + A_{2,2} B_{2,2}$

ได้ Recurrence

$T(n) = 8T(n/2) + O(n^2)$

Master Method

$C = \log_b a = \log_2 8 = 3$

$n^c = n^3$

$n^d = n^2$

$n^d < n^c \rightarrow O(n^3)$

จะเห็นได้ว่าเวลาการทำงานแบบวน for กับแบบ Divide & Conquer เท่ากัน คือ $O(n^3)$

Algorithm ของ Strassen

Define

$M_1 = (A_{1,1} + A_{2,2})(B_{1,1} + B_{2,2})$

$M_2 = (A_{2,1} + A_{2,2}) B_{1,1}$

$M_3 = A_{1,1} (B_{1,2} - B_{2,2})$

$M_4 = A_{2,2} (B_{2,1} - B_{1,1})$

$M_5 = (A_{1,1} + A_{1,2}) B_{2,2}$

$M_6 = (A_{2,1} - A_{1,1})(B_{1,1} + B_{1,2})$

$M_7 = (A_{1,2} - A_{2,2})(B_{2,1} + B_{2,2})$

M แต่ละตัวสามารถคำนวณได้โดย เมทริกซ์ n/2 * n/2

จำนวนการบวกเป็น $10(n/2)^2$

$C_{1,1} = M_1 + M_4 - M_5 + M_7$

$C_{1,2} = M_3 + M_5$

$C_{2,1} = M_2 + M_4$

$C_{2,2} = M_1 - M_2 + M_3 + M_6$

มีการบวกอีก 8 ครั้ง ของ เมทริกซ์ n/2 * n/2

ดังนั้น การบวกทั้งหมดเป็น $18(n/2)^2$

จะเห็นว่าวิธีนี้จะทำให้การคูณกันลดเหลือ 7 ครั้ง จากของเดิม 8 ครั้ง

ได้ Recurrence

$T(n) = 7T(n/2) + O(n^2)$

Master Method

$C = \log_b a = \log_2 7 = 2.807$

$n^c = n^{2.807}$

$n^d = n^2$

$n^d < n^c \rightarrow O(n^{2.807})$

จะเห็นวิธีนี้จะเร็วกว่าอีก 2 วิธีที่ยกตัวอย่างมาข้างต้น