การค้นหาข้อมูล

ประมุข ขันเงิน
pramook@gmail.com

ปัญหาการค้นหาข้อมูล

- ปัญหาการค้นหาข้อมูลเป็นปัญหาที่สำคัญ
  - เราใช้ Google กันทุกวัน
  - เราเปิดดิกชันนารีหรือพจนานุกรมบ่อยๆ
  - เราใช้สมุดหน้าเหลือง
  - Database
  - ฯลฯ
- แล้วมันคืออะไรกันแน่?

ปัญหาการค้นหาข้อมูล (ในทางคณิตศาสตร์)

- เรามีสิ่งของอยู่ n สิ่ง ได้แก่ $a_0, a_1, \ldots, a_{n-1}$
- เรามีประพจน์เปิด $P(x)$ เรียกง่ายๆ ว่า "เงื่อนไข"
- เราต้องการหา a_i หนึ่งตัวที่ทำให้ $P(a_i)$ เป็นจริง
- ในบางครั้งเราอาจต้องการของที่ทำให้ $P(x)$ เป็นจริงทั้งหมด หรือต้องการมากกว่าหนึ่งตัว

ตัวอย่างปัญหาการค้นหา

- การค้นหาในดิกชันนารี
  - วัตถุ = คู่ของ "คำ" กับ "ความหมาย"
  - เงื่อนไขมีได้หลายเงื่อนไข
  - ส่วนมากจะอยู่ในรูป "คำ = **xxx**" โดยที่ผู้ใช้เป็นคนกำหนด **xxx**

ตัวอย่างปัญหาการค้นหา

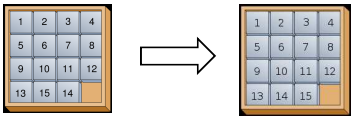
- การค้นหาในสมุดโทรศัพท์
  - วัตถุ: คู่ของ "ชื่อ" กับ "เบอร์โทรศัพท์"
  - เงื่อนไข: "ชื่อ = **xxx**"

ตัวอย่างปัญหาการค้นหา

- การค้นหาด้วย Google
  - วัตถุ: เว็บเพจ
  - เงื่อนไข: "มีคำว่า **xxx** และ **yyy** และ **zzz**"
  - เราต้องการเว็บเพจหลายๆ เวบเพจ แต่ผลลัพธ์ต้องเรียงตาม "คุณภาพ" ของเวบเพจเหล่านั้น

ตัวอย่างปัญหาการค้นหา

- 15 Puzzle



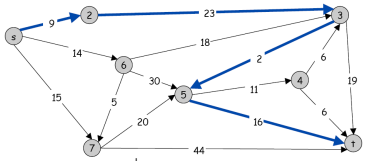
  - เลื่อนตัวเลขไปแทนที่ช่องว่าง ให้ท้ายสุดแล้วเลขเรียงกันเหมือนรูปทางขวา
  - ปัญหานี้ก็เป็นปัญหาการค้นหาเช่นกัน!

ตัวอย่างปัญหาการค้นหา

- 15 Puzzle
  - วัตถุ: วิธีการเล่นตัวเลขทั้งหมด
  - เงื่อนไข: "เมื่อทำตามวิธีการเสร็จแล้วได้เลขเรียงกัน และมีจำนวนการเลื่อนน้อยที่สุดเท่าที่จะเป็นไปได้"

ตัวอย่างปัญหาการค้นหา

- หาทางที่สั้นที่สุด



  - มี "เมือง" และ "เส้นทาง" เชื่อมระหว่างจุด
  - เส้นทางแต่ละเส้นมีระยะทาง
  - ต้องการเดินทางจากเมือง s ไปยังเมือง t
  - ต้องการเส้นทางที่สั้นที่สุดเท่าที่จะเป็นไปได้

ตัวอย่างปัญหาการค้นหา

- หาทางที่สั้นที่สุด
  - วัตถุ: เส้นทางจาก s ไปยัง t ทั้งหมด
  - เงื่อนไข: "เส้นทางมีระยะทางน้อยกว่าหรือเท่ากับเส้นทางอื่นๆ"

วิธีการแก้ปัญหาการค้นหาโดยทั่วไป

- ถึกมันเลย!
- เรียกวิธีนี้ว่า Brute-force algorithm
- ความต้องการสองอย่าง
  - วิธีการหยิบวัตถุขึ้นมาดูทีละชิ้น จนครบทุกวัตถุ
  - วิธีการตรวจสอบว่าวัตถุแต่ละชิ้นเป็นไปตามเงื่อนไขหรือไม่?
- ทำอย่างไร
  1. หยิบวัตถุชิ้นต่อไปขึ้นมา ถ้าวัตถุหมดก็เลิก
  2. เช็คว่าวัตถุนั้นเป็นของที่เราต้องการค้นหาหรือไม่
  3. ถ้าใช่ ก็คืนวัตถุนั้นไป
  4. ถ้าไม่ใช่ ก็กลับไปข้อ 1

Brute Force Algorithm

สำหรับวัตถุ a_i แต่ละชิ้น

{

ถ้า $P(a_i)$ แล้ว

คืนค่า a_i

}

หาจำนวนในอะเรย์

- มีอะเรย์ int *a ขนาด n ช่อง
- ให้จำนวนเต็ม k
- หาเลข i มาหนึ่งตัวที่ทำให้ a[i] = k

หาจำนวนในอะเรย์

- วัตถุ: เลข 0, 1, 2, 3, 4, 5, …, n-1
- ประพจน์เปิด: a[i] = k

- วิธีการหยิบวัตถุมาดูทีละตัว: for(i=0;i<n;i++)
- วิธีการเช็คว่าวัตถุเป็นไปตามเงื่อนไขหรือไม่: if (a[i] == k)

หาจำนวนในอะเรย์

```
int find(int *a, int n, int k)
{
  int i;
  for(i=0;i<n;i++)
    if (a[i] == k)
      return i;
  return -1;
}
```

หาจำนวนในอะเรย์

```
int find(int *a, int n, int k)
{
  int i;
  for(i=0;i<n;i++)
    if (a[i] == k)
      return i;
  return -1;
}
```

หยิบวัตถุมาดูทีละตัว

หาจำนวนในอะเรย์

```
int find(int *a, int n, int k)
{
  int i;
  for(i=0;i<n;i++)
    if (a[i] == k)
      return i;
  return -1;
}
```

เช็คว่าวัตถุตรงกับเงื่อนไขหรือไม่

หาคำในข้อความ

- โจทย์กำหนด
  - string คำสั้นๆ ความยาว m: char *pattern
  - string ข้อความยาวๆ ความยาว n: char *text
- ต้องการหาตำแหน่งแรกที่ pattern ปรากฏใน text
- ถ้า pattern ไม่ปรากฏใน text ตอบ -1
- ตัวอย่าง
  - pattern: `dolor`
  - text: `Lorem_ipsum_dolor_sit_amet`
  - คำตอบ: 12

หาคำในข้อความ

- ตัวอย่าง
  - pattern: `lux`
  - text: `dixitque_Deus_fiat_lux_et_facta_est_lux`
  - คำตอบ: 19
- ตัวอย่าง
  - pattern: `abcdef`
  - text: `quid_quid_latine_dictum_sit,_altum_videtur`
  - คำตอบ: -1

หาคำในข้อความ

- วัตถุ: เลข 0, 1, 2, 3, …, n-m
  - ทำไมไม่ถึง n?
- เงื่อนไข
  - เงื่อนไขเป็นประพจน์เปิด P(i) เมื่อ i เป็นจำนวนเต็ม
  - pattern ปรากฏเป็นครั้งแรกใน text ที่ตำแหน่ง i กล่าวคือ
  - pattern[0] = text[i]
  - pattern[1] = text[i+1]
  - pattern[2] = text[i+2]
  - :
  - :
  - pattern[m-1] = text[i+m-1]

หาคำในข้อความ

- ส่วนที่เขียนโปรแกรมยากคือส่วนในการเช็คว่า pattern เริ่มที่ตำแหน่ง i ของ text หรือไม่
- ฉะนั้นควรแยกออกมาเขียนเป็นฟังก์ชัน
- checkpattern (char *pattern, int m, char *text, int i)
  - คืน 1 ถ้า pattern เริ่มที่ตำแหน่ง i ของ text
  - คือ 0 ถ้าไม่เป็นเช่นนั้น

หาคำในข้อความ

```
int checkpattern(char *pattern, int m,
  char *text, int i)
{
  int j;
  for(j=0;j<m;j++)
    if (pattern[j] != text[j+i])
      return 0;
  return 1;
}
```

หาคำในข้อความ

- หลังจากนี้เราจึงนำ checkpattern มาใช้ในการเขียนฟังก์ชัน find(char *pattern, int m, char *text, int n)
  - ถ้า pattern ไม่ปรากฏใน text คืน -1
  - ถ้า pattern ปรากฏใน text เป็นที่แรก ณ ตำแหน่ง i ก็คืน i

หาคำในข้อความ

```
int find(char *pattern, int m,
  char *text, int n)
{
  int i;
  for(i=0;i<n-m+1;i++)
    if (checkpattern(pattern, m, text, i))
      return i;
  return -1;
}
```

หาคำในข้อความ

```
int find(char *pattern, int m,
  char *text, int n)
{
  int i;
  for(i=0;i<n-m+1;i++)
    if (checkpattern(pattern, m, text, i))
      return i;
  return -1;
}
```

หยิบวัตถุมาดูทีละตัว

หาคำในข้อความ

```
int find(char *pattern, int m,
  char *text, int n)
{
  int i;
  for(i=0;i<n-m+1;i++)
    if (checkpattern(pattern, m, text, i))
      return i;
  return -1;
}
```

เช็คว่าวัตถุตรงกับเงื่อนไขหรือไม่

การหาค่ามากสุด/น้อยสุด

- ปัญหานี้ต่างจากการค้นหาแบบที่เราเคยเจอมาแล้วเล็กน้อย
- เราไม่สามารถตรวจสอบว่าวัตถุที่เราหยิบมาดูตอนนี้มันมีค่าน้อยสุดหรือมากที่สุด โดยยังไม่ได้ดูค่าของวัตถุชิ้นอื่น
- ฉะนั้น brute-force algorithm แบบเดิมใช้ไม่ได้
- ต้องปรับปรุงเล็กน้อย

การหาค่ามากสุด/น้อยสุด

- สมมติว่าเราจะหาค่ามากที่สุด
- ให้จำค่ามากที่สุดที่เราเคยเจอไว้ พร้อมทั้งวัตถุนั้นด้วย
- ตอนแรกให้ค่าที่จำไว้นี้มีค่าน้อยมากๆ
- หลังจากนั้นหยิบวัตถุมาดูทีละชิ้น
- ถ้าชิ้นนี้มีค่ามากกว่า ก็เอาค่าของวัตถุนี้ไปแทนค่ามากที่สุดที่เคยเจอ และเก็บวัตถุนั้นไว้ด้วย
- เมื่อดูวัตถุหมดทุกชิ้นแล้วก็จะได้ค่ามากที่สุด

หาค่ามากที่สุดในอะเรย์

- เรามีอะเรย์ float *a ขนาด n ช่อง
- เราต้องการจำนวนเต็ม i ซึ่ง a[i] มีค่ามากที่สุด

- วัตถุ: เลข 0, 1, 2, 3, …, n-1

หาค่ามากที่สุดในอะเรย์

```
int findmax(float *a, int n)
{
  float max = -100000000;
  int max_i;
  int i;
  for(i=0;i<n;i++)
  {
    if (a[i] > max)
    {
      max = a[i];
      max_i = i;
    }
  }
  return max_i;
}
```

ค่ามากที่สุดที่เคยเจอ

หาค่ามากที่สุดในอะเรย์

```
int findmax(float *a, int n)
{
  float max = -100000000;
  int max_i;
  int i;
  for(i=0;i<n;i++)
  {
    if (a[i] > max)
    {
      max = a[i];
      max_i = i;
    }
  }
  return max_i;
}
```

วัตถุที่มีค่ามากที่สุดที่เคยเจอ

หาค่ามากที่สุดในอะเรย์

```
int findmax(float *a, int n)
{
  float max = -100000000;
  int max_i;
  int i;
  for(i=0;i<n;i++)
  {
    if (a[i] > max)
    {
      max = a[i];
      max_i = i;
    }
  }
  return max_i;
}
```

เอาวัตถุมาดูทีละอัน

หาค่ามากที่สุดในอะเรย์

```
int findmax(float *a, int n)
{
  float max = -100000000;
  int max_i;
  int i;
  for(i=0;i<n;i++)
  {
    if (a[i] > max)
    {
      max = a[i];
      max_i = i;
    }
  }
  return max_i;
}
```

เช็คว่าค่าของมันมากกว่าที่เคยเจอมาหรือไม่?

หาค่ามากที่สุดในอะเรย์

```
int findmax(float *a, int n)
{
  float max = -100000000;
  int max_i;
  int i;
  for(i=0;i<n;i++)
  {
    if (a[i] > max)
    {
      max = a[i];
      max_i = i;
    }
  }
  return max_i;
}
```

ถ้ามากกว่าก็เปลี่ยนค่ามากที่สุดและเก็บวัตถุไว้

หาค่ามากที่สุดในอะเรย์

```
int findmax(float *a, int n)
{
  float max = -100000000;
  int max_i;
  int i;
  for(i=0;i<n;i++)
  {
    if (a[i] > max)
    {
      max = a[i];
      max_i = i;
    }
  }
  return max_i;
}
```

คืนวัตถุที่มากที่สุดกลับไป

แบบฝึกหัด: หาค่าน้อยที่สุดในอะเรย์

- เขียน int findmin(float *a, int n)

หาช่วงที่ผลบวกมากที่สุด

- กำหนดลำดับของจำนวนเต็ม $a_0, a_1, \ldots, a_{n-1}$
- หา**ลำดับย่อย**ของลำดับที่ให้มาที่มีผลบวกมากที่สุด
- ลำดับย่อยคือลำดับ $a_i, a_{i+1}, \ldots, a_{j-1}, a_j$ เมื่อ $i \leq j$
- ผลบวกของลำดับย่อยคือ

$$a_i + a_{i+1} + a_{i+2} + \cdots + a_{j-1} + a_j$$

หาช่วงที่ผลบวกมากที่สุด

- ยกตัวอย่างเช่นมีลำดับ: -2, 11, -4, 13, -5, -2
- ลำดับย่อยที่มีผลบวกมากที่สุดคือ: -2, **11, -4, 13**, -5, -2
- ผลบวก = 20

หาช่วงที่ผลบวกมากที่สุด

- เราวิเคราะห์ปัญหานี้ด้วยวิธีการเดิม
- วัตถุคืออะไร?
  - "ช่วง"
  - จริงๆ แล้วช่วงก็คือเลขสองตัว i และ j เมื่อ $i \leq j$
- ค่าของวัตถุที่เราสนใจคืออะไร?

$$a_i + a_{i+1} + a_{i+2} + \cdots + a_{j-1} + a_j$$

หาช่วงที่ผลบวกมากที่สุด

- แก้ปัญหาแบบเดิม
  - ให้จำค่าผลบวกมากที่สุดที่เราเคยเจอไว้ พร้อมทั้งช่วงของผลบวกนั้นด้วย
  - ตอนแรกให้ค่าผลบวกที่มากที่สุดที่เคยเจอมีค่าน้อยมากๆ
  - หลังจากนั้นหยิบช่วงมาดูทีละช่วง
  - ถ้าช่วงนี้มีค่ามากกว่า ก็เอาผลบวกของมันไปแทนผลบวกที่มากที่สุดที่เคยเจอ และเก็บผลบวกนั้นไว้ด้วย
  - เมื่อดูช่วงหมดทุกช่วงแล้วก็จะได้ค่ามากที่สุด

หาช่วงที่ผลบวกมากที่สุด

- สิ่งที่ยังขาดหายไปมีสองอย่าง
  - วิธีการหยิบช่วงทั้งหมดมาดูทีละช่วง
  - วิธีการหาค่าของช่วง
- หยิบช่วงมาดูทีละช่วงทำอย่างไร: for-loop สองชั้น!

```
for(i=0;i<n;i++)
  for(j=i;j<n;j++)
    ...
```

หาช่วงที่ผลบวกมากที่สุด

- คำนวณค่าของช่วงทำอย่างไร
  - สมมติว่าข้อมูลจำนวนของเราเก็บอยู่นอะเรย์ int *a
  - เขียนฟังก์ชัน int value(int *a, int i, int j) เพื่อคำนวณค่าของช่วง (i, j)

```
int value(int *a, int i, int j)
{
  int k, result = 0;
  for(k=i; k<=j; k++)
    result += a[k];
  return result;
}
```

หาช่วงที่ผลบวกมากที่สุด

- เราต้องการเขียนฟังก์ชัน

  void maxinterval(int *a, int n, int *maxi, int *maxj)

  - a คือ อะเรย์ที่เก็บจำนวน
  - n คือ จำนวนสมาชิกของอะเรย์
  - ฟังก์ชันคืนคำตอบกลับไปผ่านตัวแปรที่ถูกชี้ด้วย maxi และ maxj
  - maxi ชี้ไปยังตัวแปรที่จะเก็บตำแหน่งเริ่มต้นของช่วงที่มีผลบวกมากสุด
  - maxj ชี้ไปยังตัวแปรที่จะเก็บตำแหน่งสิ้นสุดของช่วงที่มีผลบวกมากสุด

หาช่วงที่ผลบวกมากที่สุด

```
void maxinterval(int *a, int n,
                 int *maxi, int *maxj) {
  int max = -1000000;
  int i, j, v;
  for(i=0;i<n;i++)
    for(j=i;j<n;j++) {
      v = value(a, i, j);
      if (v > max) {
        max = v;
        *maxi = i; *maxj = j;
      }
    }
}
```

ค่ามากที่สุดที่เคยเจอ

หาช่วงที่ผลบวกมากที่สุด

```
void maxinterval(int *a, int n,
                 int *maxi, int *maxj) {
  int max = -1000000;
  int i, j, v;
  for(i=0;i<n;i++)
    for(j=i;j<n;j++) {
      v = value(a, i, j);
      if (v > max) {
        max = v;
        *maxi = i; *maxj = j;
      }
    }
}
```

วัตถุที่มีค่ามากที่สุดที่เคยเจอ

หาช่วงที่ผลบวกมากที่สุด

```
void maxinterval(int *a, int n,
                 int *maxi, int *maxj) {
  int max = -1000000;
  int i, j, v;
  for(i=0;i<n;i++)
    for(j=i;j<n;j++) {
      v = value(a, i, j);
      if (v > max) {
        max = v;
        *maxi = i; *maxj = j;
      }
    }
}
```

หยิบวัตถุมาดูทีละตัว

หาช่วงที่ผลบวกมากที่สุด

```
void maxinterval(int *a, int n,
                 int *maxi, int *maxj) {
  int max = -1000000;
  int i, j, v;
  for(i=0;i<n;i++)
    for(j=i;j<n;j++) {
      v = value(a, i, j);
      if (v > max) {
        max = v;
        *maxi = i; *maxj = j;
      }
    }
}
```

คำนวณค่าของวัตถุ

หาช่วงที่ผลบวกมากที่สุด

```
void maxinterval(int *a, int n,
                 int *maxi, int *maxj) {
  int max = -1000000;
  int i, j, v;
  for(i=0;i<n;i++)
    for(j=i;j<n;j++) {
      v = value(a, i, j);
      if (v > max) {
        max = v;
        *maxi = i; *maxj = j;
      }
    }
}
```

เช็คว่าค่าของมันมากกว่าที่เคยเจอมาหรือไม่?

หาช่วงที่ผลบวกมากที่สุด

```
void maxinterval(int *a, int n,
                 int *maxi, int *maxj) {
  int max = -1000000;
  int i, j, v;
  for(i=0;i<n;i++)
    for(j=i;j<n;j++) {
      v = value(a, i, j);
      if (v > max) {
        max = v;
        *maxi = i; *maxj = j;
      }
    }
}
```

ถ้ามากกว่าก็เปลี่ยนค่ามากที่สุดและเก็บวัตถุไว้

หาช่วงที่ผลบวกมากที่สุด

- สังเกตว่าเราแก้ปัญหาแบบเดียวกันกับปัญหาการหาค่ามากสุดในอะเรย์
  - เพียงแค่วัตถุเปลี่ยน
  - เพียงแค่วิธีการหยิบวัตถุมาดูเปลี่ยน
  - เพียงแค่วิธีการหาค่าของวัตถุเปลี่ยน
- **คุณต้องสร้างความสามารถในการสรุปรูปแบบการแก้ปัญหาต่างๆ ด้วยการสร้างประสบการณ์ เพื่อจะได้นำรูปแบบการแก้ปัญหาเหล่านี้มาประยุกต์ใช้ได้ในภายหลัง**

ประสิทธิภาพ

- เวลาในการทำงานของโปรแกรมขึ้นอยู่กับจำนวนคำสั่งที่มันทำ
- ส่วนมากเวลาเราประเมินประสิทธิภาพของโปรแกรม จะนับจำนวนครั้งของคำสั่งที่ถูกทำงานบ่อยที่สุด (หรือคำสั่งที่เราสนใจมากที่สุด)
- คำสั่งที่เราจะนับเรียกว่า main operation

ประสิทธิภาพของโปรแกรมแก้ปัญหาช่วงที่ผลบวกมากที่สุด

- สำหรับในกรณีโปรแกรมแก้ปัญหาช่วงที่มีผลบวกมากที่สุด อะไรคือ main operation?
- คำสั่งไหนถูกทำงานบ่อยที่สุด?

ประสิทธิภาพของโปรแกรมแก้ปัญหาช่วงที่ผลบวกมากที่สุด

```
int value(int *a, int i, int j)
{
  int k, result = 0;
  for(k=i; k<=j; k++)
    result += a[k];
  return result;
}
```

ประสิทธิภาพของโปรแกรมแก้ปัญหาช่วงที่ผลบวกมากที่สุด

- ในการเรียก value(a, i, j) คำสั่งนี้จะทำงาน j-i+1 ครั้ง
- เวลาฟังก์ชัน maxinterval ทำงาน มันจะวนรอบค่า i, j ทุกคู่ที่ i น้อยกว่าหรือเท่ากับ j และสำหรับทุกคู่จะมีการเรียก value(a,i,j)
- ดังนั้น ถ้าอะเรย์มีข้อมูล n ตัว คำสั่ง result += a[k] จึงถูกเรียกทำงานทั้งหมด

$$\sum_{i=0}^{n-1}\sum_{j=i}^{n-1}(j-i+1)$$

ประสิทธิภาพของโปรแกรมแก้ปัญหาช่วงที่ผลบวกมากที่สุด

$$\begin{aligned}\sum_{i=0}^{n-1}\sum_{j=i}^{n-1}(j-i+1) &= \sum_{i=1}^{n}\sum_{j=i}^{n}(j-i+1)\\ &= \sum_{i=1}^{n}\sum_{k=1}^{n-i+1} k\\ &= \sum_{i=1}^{n}\frac{(n-i+1)(n-i+2)}{2}\\ &= \sum_{i=1}^{n}\frac{i(i+1)}{2}\\ &= \frac{1}{2}\sum_{i=1}^{n}(i^2+i)\\ &= \frac{n(n+1)(n+2)}{3}\end{aligned}$$

ประสิทธิภาพของโปรแกรมแก้ปัญหาช่วงที่ผลบวกมากที่สุด

- n(n+1)(n+2)/3 มันมากขนาดไหน?

n	n(n+1)(n+2)/3
1	2
10	440
100	343400
1000	334334000
10000	3.33433E+11
100000	3.33343E+14
1000000	3.33334E+17

ประสิทธิภาพของโปรแกรมแก้ปัญหาช่วงที่ผลบวกมากที่สุด

- นี่หมายความว่าอย่างไร?
- เครื่องผมมีความเร็ว 2.6 GHz ตีซะว่าหนึ่งวินาทีมันทำงานได้ 2.6 พันล้านคำสั่ง

n	n(n+1)(n+2)/3	เวลาทำงาน (วินาที)	เวลาทำงาน
1	2	7.69231E-10	0.8 นาโนวินาที
10	440	1.69231E-07	170 นาโนวินาที
100	343400	0.000132077	0.1 มิลลิวินาที
1000	334334000	0.12859	0.1 วินาที
10000	3.33433E+11	128.2435923	2 นาทีกว่า
100000	3.33343E+14	128208.9744	36 ชั่วโมง
1000000	3.33334E+17	128205512.8	4 ปี

ประสิทธิภาพของโปรแกรมแก้ปัญหาช่วงที่ผลบวกมากที่สุด

- ใช้เวลา 4 ปีในการประมวลผลข้อมูลเพียง 4 MB!
- ถ้ามีตัวเลข 10 ล้านตัวล่ะ?
- จำนวนคำสั่งที่เราทำประมาณ n^3
- ดังนั้นจำนวนเพิ่มขึ้น 10 เท่า เวลาเพิ่มขึ้น 1,000 เท่า
- ดังนั้นน่าจะใช้เวลาประมาณ 4,000 ปี
- เครื่องคอมอาจจะพังก่อน หรือไม่มนุษย์อาจจะสูญพันธุ์ไปแล้วกว่าจะทำงานเสร็จ

ประสิทธิภาพของโปรแกรมแก้ปัญหาช่วงที่ผลบวกมากที่สุด

- บทเรียน: **ประสิทธิภาพของโปรแกรมคอมพิวเตอร์วัดกันที่ความสามารถในการรองรับการเพิ่มขนาดของข้อมูล**
- เราสามารถทำได้ดีกว่านี้หรือไม่?
- ได้สิ!

ประสิทธิภาพของโปรแกรมแก้ปัญหาช่วงที่ผลบวกมากที่สุด

- ความช้าของโปรแกรมของเราอยู่ที่ฟังก์ชัน value ซึ่งต้องทำ for loop เพื่อหาค่าผลบวกของช่วง

$$a_i + a_{i+1} + a_{i+2} + \cdots + a_{j-1} + a_j = \sum_{k=i}^{j} a_k$$

ประสิทธิภาพของโปรแกรมแก้ปัญหาช่วงที่ผลบวกมากที่สุด

- แต่

$$\sum_{k=i}^{j} a_k = \sum_{k=1}^{n} a_k - \sum_{k=1}^{i-1} a_k - \sum_{k=j+1}^{n} a_k$$

- $\sum_{k=1}^{n} a_k$ คือผลบวกทั้งหมดของค่าทั้งหมด
- $\sum_{k=1}^{i-1} a_k$ คือผลบวกของเลข i-1 ตัวแรก
- $\sum_{k=j+1}^{n} a_k$ คือผลบวกของเลข n-j ตัวสุดท้าย
- ค่าพวกนี้เราสามารถคำนวณเก็บไว้ก่อนได้ก่อนที่เราจะไปหาช่วงจริงๆ

ประสิทธิภาพของโปรแกรมแก้ปัญหาช่วงที่ผลบวกมากที่สุด

- สมมติว่าเราคำนวณ $\sum_{k=1}^{n} a_k$ เก็บไว้ในตัวแปร S
- สมมติว่าเราคำนวณ $\sum_{k=1}^{i-1} a_k$ เก็บไว้ในอะเรย์ float *b โดยที่ b[i] มีค่าเท่ากับ $\sum_{k=1}^{i-1} a_k$
- สมมติว่าเราคำนวณ $\sum_{k=j+1}^{n} a_k$ เก็บไว้ในอะเรย์ float *c โดยที่ c[j] มีค่าท่ากับ $\sum_{k=j+1}^{n} a_k$

ประสิทธิภาพของโปรแกรมแก้ปัญหาช่วงที่ผลบวกมากที่สุด

- เราเขียนฟังก์ชัน value ใหม่ได้ดังนี้

```
int value(int i, int j, float S,
          float *b, float *c)
{
  return S - b[i] - c[j];
}
```

ประสิทธิภาพของโปรแกรมแก้ปัญหาช่วงที่ผลบวกมากที่สุด

- แล้วเราจะคำนวณ S, b, และ c ได้อย่างไร?
- คำนวณ S

```
S = 0;
for(i=0;i<n;i++)
  S += a[i];
```

ประสิทธิภาพของโปรแกรมแก้ปัญหาช่วงที่ผลบวกมากที่สุด

- คำนวณ b

```
b[0] = 0;
for(i=1;i<n;i++)
  b[i] += b[i-1] + a[i-1];
```

- คำนวณ c

```
c[n-1] = 0;
for(i=n-2;i>=0;i--)
  c[i] += c[i+1] + a[i+1];
```

ประสิทธิภาพของโปรแกรมแก้ปัญหาช่วงที่ผลบวกมากที่สุด

```
void maxinterval(float *a, int n,
                 int *maxi, int
   *maxj)
{
  int i, j;
  float *b = (float *)malloc(
    n * sizeof(float));
  float *c = (float *)malloc(
    n * sizeof(float));
  float S = 0;
  int max = -1000000;
  int v;

  for(i=0;i<n;i++) S += a[i];

  b[0] = 0;
  for(i=0;i<n;i++)
    b[i] = b[i-1] + a[i-1];

  c[0] = 0;
  for(i=n-1;i>=0;i--)
    c[i] = c[i+1] + a[i+1];

  for(i=0;i<n;i++)
    for(j=i;j<n;j++) {
      v = value(i, j, S, b, c);
      if (v > max) {
        max = v;
        *maxi = i; *maxj = j;
      }
    }

  free(b);
  free(c);
}
```

ประสิทธิภาพของโปรแกรมแก้ปัญหาช่วงที่ผลบวกมากที่สุด

- ตอนนี้ value แทนที่จะทำงาน j-i+1 คำสั่ง มันทำงานแค่คำสั่งเดียว
- จำนวนคำสั่งที่ทำงานคือ

$$\sum_{i=0}^{n-1}\sum_{j=i}^{n-1} 1 = \frac{n(n+1)}{2}$$

ประสิทธิภาพของโปรแกรมแก้ปัญหาช่วงที่ผลบวกมากที่สุด

n	n(n+1)/2	เวลาทำงาน (วินาที)	เวลา
1	1	3.84615E-10	0.4 นาโนวินาที
10	55	2.11538E-08	20 นาโนวินาที
100	5050	1.94231E-06	2 ไมโครวินาที
1000	500500	0.0001925	0.2 มิลลิวินาที
10000	50005000	0.019232692	20 มิลลิวินาที
100000	5E+09	1.923096154	2 วินาที
1000000	5E+11	192.3078846	3 นาที
10000000	5E+13	19230.77115	5.3 ชั่วโมง

ประสิทธิภาพของโปรแกรมแก้ปัญหาช่วงที่ผลบวกมากที่สุด

- เราลดการทำงานจากประมาณ n^3 เป็นประมาณ n^2
- 4,000 ปี → 5 ชั่วโมง

- บทเรียน: **การคำนวณค่าเก็บไว้ก่อน (precomputation) สามารถทำให้โปรแกรมของคุณมีประสิทธิภาพดีขึ้นมาก**

ประสิทธิภาพของโปรแกรมแก้ปัญหาช่วงที่ผลบวกมากที่สุด

- เราสามารถทำได้ดีกว่านี้หรือไม่?
- ได้สิ! ความจริงมีวิธีการแก้ปัญหานี้ที่ทำงานได้ในเวลาประมาณ n ด้วย
- แต่คุณจะได้เรียนมันใน 418231: Algorithms and Data Structures