การค้นหาข้อมูล (ต่อ)

ประมุข ขันเงิน
pramook@gmail.com

แบ่งเลข

- ให้จำนวนเต็มมาทั้งหมด n ตัว
- ต้องการแบ่งเลขทั้ง n ตัวออกเป็นสองชุด
- โดยที่ให้ค่าสัมบูรณ์ของผลต่างระหว่างผลบวกของเลขทั้งสองชุดมีค่าน้อยที่สุด

ตัวอย่าง 1

- สำหรับเลข 1, 10, 5, 9, 3
- เราสามารถแบ่งเลขออกเป็นสองชุด 1, 10, 3 และ 5, 9
- แต่ละชุดมีผลบวกเท่ากับ 14
- ค่าสัมบูรณ์ของผลต่างของผลบวกคือ |14-14| = 0 ซึ่งมีค่าน้อยที่สุด

ตัวอย่าง 2

- สำหรับเลข 30, 2, 9, 11
- เราสามารถแบ่งเลขออกเป็นสองชุด 30 และ 2, 9, 11
- ค่าสัมบูรณ์ของผลต่างของผลบวกคือ |30 - (2 + 9 + 11)| = 8 ซึ่งน้อยที่สุดเท่าที่จะเป็นไปได้แล้ว

วัตถุที่ต้องการค้นหา

- วัตถุที่ต้องการค้นหา: วิธีการแบ่งเซตของเลขที่ได้เป็นสองแบบ
- "วิธีการแบ่ง" คืออะไร?
- คิดง่ายๆ คือการเอาป้ายไปแปะไว้ที่เลขแต่ละตัวว่ามันอยู่กลุ่มไหน

- สมมติว่าเลขที่ให้มาเก็บไว้ในอะเรย์ int *a ซึ่งมีขนาด n ช่อง
- เราสร้างอะเรย์ int *g ซึ่งมีขนาด n เช่นกัน โดยเราตั้งใจให้
  - g[i] = 1 ถ้าเลข a[i] อยู่ในกลุ่มที่ 1
  - g[i] = 2 ถ้าเลข a[i] อยู่ในกลุ่มที่ 2

วัตถุที่ต้องการค้นหา

- เรามีวิธีแบ่งทั้งหมดเท่าไหร่?
- g[i] มีค่าได้ 2 แบบ (1 หรือ 2)
- เรามี g[0], g[1], g[2], …, g[n-1]
- ดังนั้นมีวัตถุที่เราสนใจทั้งหมด 2^n

ค่าของวิธีแบ่ง

- เมื่อเราได้วัตถุมาแล้วเราจะหาค่าของมันอย่างไร?
- เขียนฟังก์ชัน value สำหรับหาค่าสัมบูรณ์ของผลต่างของผลบวกของแต่ละกลุ่ม

```
int value(int *a, int *g, int n)
{
  int sum_g1 = 0, sum_g2;
  int i;
  for(i=0;i<n;i++)
    if (g[i] == 1)
      sum_g1 += a[i];
    else
      sum_g2 += a[i];
  return abs(sum_g1 - sum_g2);
}
```

การสร้างวิธีแบ่ง

- ปัญหาการสร้างวิธีแบ่งในกรณีนี้ยากกว่าปัญหาการหาค่าของมันมาก
- เรามาดูตัวอย่างของวิธีแบ่งกันก่อน
- ถ้า n = 1 มีวิธีแบ่งอะไรบ้าง?
  - (1), (2)
- ถ้า n = 2 มีวิธีแบ่งอะไรบ้าง?
  - (1,1), (2,1), (1,2), (2,2)
- ถ้า n = 3 มีวิธีแบ่งอะไรบ้าง?
  - (1,1,1), (2,1,1), (1,2,1), (2,2,1), (1,1,2), (2,1,2), (1,2,2), (2,2,2)

การสร้างวัตถุที่ต้องการค้นหา

- ถ้า n = 4 มีวิธีแบ่งอะไรบ้าง?
  - (1,1,1,1), (2,1,1,1), (1,2,1,1), (2,2,1,1), (1,1,2,1), (2,1,2,1), (1,2,2,1), (2,2,2,1), (1,1,1,2), (2,1,1,2), (1,2,1,2), (2,2,1,2), (1,1,2,2), (2,1,2,2), (1,2,2,2), (2,2,2,2)

การสร้างวัตถุที่ต้องการค้นหา

- ถ้า n = 5 มีวิธีแบ่งอะไรบ้าง?
  - (1,1,1,1,1), (1,2,1,1,1), (1,1,2,1,1), (1,2,2,1,1), (1,1,1,2,1), (1,2,1,2,1), (1,1,2,2,1), (1,2,2,2,1), (1,1,1,1,2), (1,2,1,1,2), (1,1,2,1,2), (1,2,2,1,2), (1,1,1,2,2), (1,2,1,2,2), (1,1,2,2,2), (1,2,2,2,2), (2,1,1,1,1), (2,2,1,1,1), (2,1,2,1,1), (2,2,2,1,1), (2,1,1,2,1), (2,2,1,2,1), (2,1,2,2,1), (2,2,2,2,1), (2,1,1,1,2), (2,2,1,1,2), (2,1,2,1,2), (2,2,2,1,2), (2,1,1,2,2), (2,2,1,2,2), (2,1,2,2,2), (2,2,2,2,2)

ข้อสังเกต

- กรณี n = 1
  - (1), (2)
- กรณี n = 2
  - (1,1), (2,1), (1,2), (2,2)

ข้อสังเกต

- กรณี n = 2
  - (1,1), (2,1), (1,2), (2,2)
- กรณี n = 3
  - (1,1,1), (2,1,1), (1,2,1), (2,2,1), (1,1,2), (2,1,2), (1,2,2), (2,2,2)

ข้อสังเกต

- กรณี n = 3
  - (1,1,1), (2,1,1), (1,2,1), (2,2,1), (1,1,2), (2,1,2), (1,2,2), (2,2,2)
- กรณี n = 4
  - (1,1,1,1), (2,1,1,1), (1,2,1,1), (2,2,1,1), (1,1,2,1), (2,1,2,1), (1,2,2,1), (2,2,2,1), (1,1,1,2), (2,1,1,2), (1,2,1,2), (2,2,1,2), (1,1,2,2), (2,1,2,2), (1,2,2,2), (2,2,2,2)

ข้อสังเกต

- สรุปอะไรได้บ้าง?

ข้อสังเกต

- สรุปอะไรได้บ้าง?
- **วิธีแบ่งในกรณีที่ n = k+1 เกิดจากการนำเอาวิธีแบ่งในกรณีที่ n = k ทั้งหมดมาเพิ่ม 1 หรือ 2 ต่อท้าย**

โค้ดสำหรับสร้างวิธีแบ่ง

- การสร้างวิธีแบ่งสำหรับกรณี n = 1 ทำได้อย่างไร?
- ตัดสินใจเลือกค่า g[0]

```
for(g[0]=1;g[0]<=2;g[0]++)
  dosomething();
```

โค้ดสำหรับสร้างวิธีแบ่ง

- แล้วการสร้างวิธีแบ่งสำหรับกรณี n = 2 ล่ะ?
  - ตัดสินใจเลือกค่า g[1] ก่อน ให้มีค่าเป็น 1 หรือ 2 แล้วค่อยกำหนด g[0]

```
for(g[1]=1;g[1]<=2;g[1]++)
   for(g[0]=1;g[0]<=2;g[0]++)
     dosomething();
```

โค้ดสำหรับสร้างวิธีแบ่ง

- แล้วการสร้างวิธีแบ่งสำหรับกรณี n = 3 ล่ะ?
  - ตัดสินใจเลือกค่า g[2] ก่อน ให้มีค่าเป็น 1 หรือ 2 แล้วค่อยกำหนด g[1] และ g[0]

```
for(g[2]=1;g[2]<=2;g[2]++)
  for(g[1]=1;g[1]<=2;g[1]++)
    for(g[0]=1;g[0]<=2;g[0]++)
      dosomething();
```

โค้ดสำหรับสร้างวิธีแบ่ง

- แล้วการสร้างวิธีแบ่งสำหรับกรณี n = 4 ล่ะ?
  - ตัดสินใจเลือกค่า g[3] ก่อน ให้มีค่าเป็น 1 หรือ 2 แล้วค่อยกำหนด g[2], g[1] และ g[0]

```
for(g[3]=1;g[3]<=2;g[3]++)
  for(g[2]=1;g[2]<=2;g[2]++)
    for(g[1]=1;g[1]<=2;g[1]++)
      for(g[0]=1;g[0]<=2;g[0]++)
        dosomething();
```

โค้ดสำหรับสร้างวิธีแบ่ง

- แล้วการสร้างวิธีแบ่งสำหรับกรณี n = k ล่ะ?
  - ตัดสินใจเลือกค่า g[k] ก่อน ให้มีค่าเป็น 1 หรือ 2 แล้วค่อยกำหนด g[k-1], g[k-2], …, และ g[0]

```
for(g[k-1]=1;g[k-1]<=2;g[k-1]++)
   .
    .
     .
  for(g[2]=1;g[2]<=2;g[2]++)
    for(g[1]=1;g[1]<=2;g[1]++)
      for(g[0]=1;g[0]<=2;g[0]++)
        dosomething();
```

โค้ดสำหรับสร้างวิธีแบ่ง

- แต่ถ้าเราไม่รู้ค่า k ก่อนล่วงหน้า
- เช่นถ้าผู้ใช้เป็นคนกำหนดค่า k ให้
- แล้วจะทำอย่างไร?

Recursion

- เราจะเขียนฟังก์ชัน generate(int k) ให้ทำงานเหมือน

```
for(g[k-1]=1;g[k-1]<=2;g[k-1]++)
  for(g[k-2]=1;g[k-2]<=2;g[k-2]++)
     .
      .
       .
     for(g[1]=1;g[1]<=2;g[1]++)
       for(g[0]=1;g[0]<=2;g[0]++)
         dosomething();
```

Recursion

- ข้อสังเกต: จากเป้าหมายของเรา เราได้ว่า generate(k) ทำงานเหมือนกับ

```
for(g[k-1]=1;g[k-1]<=2;g[k-1]++)
  generate(k-1);
```

- ข้อยกเว้น: ถ้า k = 0 มันจะทำงานเหมือนกับ dosomething();

Recursion

```
void generate(int k)
{
  if (k == 0)
    dosomething();
  else
  {
    for(g[k-1]=1;g[k-1]<=2;g[k-1]++)
      generate(k-1);
  }
}
```

เอาส่วนประกอบต่างๆ มารวมกัน

- ในกรณีการแก้ปัญหาการแบ่งเลข dosomething(); ของเราคืออะไร?
  - เช่นเดียวกับปัญหาเก่า
  - generate ช่วยให้เราสามารถหยิบวิธีแบ่งมาดูทีละวิธี
  - หาค่าสัมบูรณ์ของผลต่างของผลบวกของวิธีแบ่งนั้น
  - เปรียบเทียบค่าที่ได้กับค่าที่น้อยที่สุดที่เคยเจอมา
  - เปลี่ยนค่าที่น้อยที่สุดและวิธีแบ่งที่ทำให้ได้ค่านั้นหากค่าใหม่ที่ได้น้อยกว่า

เอาส่วนประกอบต่างๆ มารวมกัน

```
int *a, *g, *min_g, n;
int min = 100000000;

void generate(int k)
{
  int v,i;
  if (k == 0) {
    v = value(a,g,n);
    if (v < min) {
      min = v;
      for(i=0;i<n;i++) min_g[i] = g[i];
    }
  } else {
    for(g[k-1]=1;g[k-1]<=2;g[k-1]++)
      generate(k-1);
  }
}
```

Permutation

- ในปัญหาการหาค่ามากสุด/น้อยสุด หลายๆ ปัญหา วัตถุที่เราสนใจอาจจะเป็น permutation
- Permutation คืออะไร?
  - กำหนดจำนวนเต็มบวก 1 ตัว คือ n
  - Permutation บน n คือการนำเอาเลข 1 ถึง n มาเรียงสับเปลี่ยนกันทุกแบบ

Permutation

- กรณี n = 1 มีเพียง 1 permutation
  - (1)
- กรณี n = 2 มี 2 permutation
  - (1,2), (2,1)
- กรณี n = 3 มี 3 permutation
  - (1,2,3), (1,3,2), (2,1,3), (2,3,1), (3,1,2), (3,2,1)

Permutation

- กรณี n = 4 มี 24 permutation
  - (1,2,3,4), (1,2,4,3), (1,3,2,4), (1,3,4,2), (1,4,2,3), (1,4,3,2), (2,1,3,4), (2,1,4,3), (2,3,1,4), (2,3,4,1), (2,4,1,3), (2,4,3,1), (3,1,2,4), (3,1,4,2), (3,2,1,4), (3,2,4,1), (3,4,1,2), (3,4,2,1), (4,1,2,3), (4,1,3,2), (4,2,1,3), (4,2,3,1), (4,3,1,2), (4,3,2,1)

โปรแกรมพิมพ์ Permutation

- ต้องการ: เขียนโปรแกรมให้ผู้ใช้ป้อนค่า n แล้วพิมพ์ permutation บน n ออกมาทั้งหมด
- ทำคล้ายๆ กับปัญหาที่แล้ว
- เราเก็บ permutation ไว้ในอะเรย์ int *p
- ในครั้งที่แล้ว g[i] แต่ละช่องเก็บค่า 1 หรือ 2
- ตอนนี้ p[i] เก็บค่า 1 ถึง n
- แต่มีข้อแม้คือ p[i] แต่ละช่องจะมีค่าซ้ำกันไม่ได้

โปรแกรมพิมพ์ Permutation

- เราจะเขียนฟังก์ชัน generate(int k) ที่ทำหน้าที่พิมพ์ permutation ของตัวเลขที่ยัง "ใช้ได้อยู่" ซึ่งมีจำนวน k ตัวมาทั้งหมด
- แล้วจะรู้ว่าตัวเลขไหนใช้ได้หรือใช้ไม่ได้?
- มีอะเรย์ int *used
  - used[i] = 1 ถ้า i เคยถูกใช้ไปแล้ว
  - used[i] = 0 ถ้า i ยังไม่เคยถูกใช้

โปรแกรมพิมพ์ Permutation

```
int *p, *used, int n;

void generate(int k)
{
  int i;
  if (k == 0) {
    print_permutation(p,n);
  } else {
    for(i=1;i<=n;i++) {
      if (!used[i]) {
        used[i] = 1;
        p[k-1] = i;

        generate(k-1);

        used[i] = 0;
      }
    }
  }
}
```

โปรแกรมพิมพ์ Permutation

```
int *p, *used, int n;

void generate(int k)
{
  int i;
  if (k == 0) {
    print_permutation(p,n);
  } else {
    for(i=1;i<=n;i++) {
      if (!used[i]) {
        used[i] = 1;
        p[k-1] = i;

        generate(k-1);

        used[i] = 0;
      }
    }
  }
}
```

Terminal Condition:
กรณีมีเลขเหลือ **0** ตัว
ให้พิมพ์ **permutation**
ออกมา

โปรแกรมพิมพ์ Permutation

```
int *p, *used, int n;

void generate(int k)
{
  int i;
  if (k == 0) {
    print_permutation(p,n);
  } else {
    for(i=1;i<=n;i++) {
      if (!used[i]) {
        used[i] = 1;
        p[k-1] = i;

        generate(k-1);

        used[i] = 0;
      }
    }
  }
}
```

กรณีที่มีเลขเหลืออย่างน้อย **1** ตัว
เราไล่หาเลขตั้งแต่ **1** ถึง **n**
เพื่อใช้เป็นค่า **p[k-1]**

โปรแกรมพิมพ์ Permutation

```
int *p, *used, int n;

void generate(int k)
{
  int i;
  if (k == 0) {
    print_permutation(p,n);
  } else {
    for(i=1;i<=n;i++) {
      if (!used[i]) {
        used[i] = 1;
        p[k-1] = i;

        generate(k-1);

        used[i] = 0;
      }
    }
  }
}
```

ตรวจสอบว่าเลขตัวที่กำลัง
พิจารณาเคยถูกใช้ไปหรือยัง

โปรแกรมพิมพ์ Permutation

```
int *p, *used, int n;

void generate(int k)
{
  int i;
  if (k == 0) {
    print_permutation(p,n);
  } else {
    for(i=1;i<=n;i++) {
      if (!used[i]) {
        used[i] = 1;
        p[k-1] = i;

        generate(k-1);

        used[i] = 0;
      }
    }
  }
}
```

ถ้ายังไม่ถูกใช้เราก็จะใช้มัน
โดยกำหนดให้ **p[k-1]** เท่ากับ **i**
และให้ **used[i] = 1** เพื่อแสดง
ว่าเราใช้มันแล้ว

โปรแกรมพิมพ์ Permutation

```
int *p, *used, int n;

void generate(int k)
{
  int i;
  if (k == 0) {
    print_permutation(p,n);
  } else {
    for(i=1;i<=n;i++) {
      if (!used[i]) {
        used[i] = 1;
        p[k-1] = i;

        generate(k-1);

        used[i] = 0;
      }
    }
  }
}
```

หลังจากนั้นจึงพิมพ์ **permutation** ที่เกิดจากการเรียงสับเปลี่ยนเลข **k-1** ตัวที่เหลือ

โปรแกรมพิมพ์ Permutation

```
int *p, *used, int n;

void generate(int k)
{
  int i;
  if (k == 0) {
    print_permutation(p,n);
  } else {
    for(i=1;i<=n;i++) {
      if (!used[i]) {
        used[i] = 1;
        p[k-1] = i;

        generate(k-1);

        used[i] = 0;
      }
    }
  }
}
```

เมื่อพิมพ์เสร็จแล้ว เราเลิกใช้ **i** แล้ว จึงให้ **used[i] = 0**

โปรแกรมพิมพ์ Permutation

```
void print_permutation(int *p, int n)
{
  int i;
  for(i=0;i<n;i++)
    printf("%d ", p[i]);
  printf("\n");
}
```

Combination

- ในบางทีวัตถุที่เราต้องการหาค่ามากสุด/น้อยสุดของมัน อาจเป็น combination
- Combination
  - กำหนดจำนวนเต็มบวกสองตัว n และ k โดย k <= n
  - Combination คือ subset ที่มีสมาชิก k ตัวของเซต {1,2,3,…,n}

Combination

- n = 4, k = 0
  {}
- n = 4, k = 1
  {1}, {2}, {3}, {4}
- n = 4, k = 2
  {1,2}, {1,3}, {1,4}, {2,3}, {2,4}, {3,4}
- n = 4, k =3
  {1,2,3}, {1,2,4}, {1,3,4}, {2,3,4}
- n = 4, k =4
  {1,2,3,4}

Combination

- n = 5, k = 0
  {}
- n = 5, k = 1
  {1}, {2}, {3}, {4}, {5}
- n = 5, k = 2
  {1,2}, {1,3}, {1,4}, {1,5}, {2,3},
  {2,4}, {2,5}, {3,4}, {3,5}, {4,5}
- n = 5, k =3
  {1,2,3}, {1,2,4}, {1,2,5}, {1,3,4}, {1,3,5},
  {1,4,5}, {2,3,4}, {2,3,5}, {2,4,5}, {3,4,5}
- n = 5, k =4
  {1,2,3,4}, {1,2,3,5}, {1,2,4,5}, {1,3,4,5}, {2,3,4,5}
- n = 5, k =5
  {1,2,3,4,5}

Combination

- เมื่อกำหนด n และ k ให้แล้ว เรามี combination รวมทั้งหมด

$$C(n,k) = \binom{n}{k} = \frac{n!}{k!(n-k)!}$$

การสร้าง combination

- เราเคยสร้าง subset มาแล้วหรือไม่?
- เคยสิ! ดูปัญหาการแบ่งเลขให้ค่าสัมบูรณ์ของผลต่างของผลบวกน้อยที่สุด
- เราแบ่งเลขออกเป็นสองกลุ่ม แต่ละกลุ่มก็คือ subset ของเลขทั้งหมด
- ถ้าลองแปลความหมายของอะเรย์ g ใหม่
  - g[i] = 1 หมายความว่าเรา "เลือก" i
  - g[i] = 2 หมายความว่าเรา "ไม่เลือก" i
- Combination คืออะเรย์ g ที่มีเลข 1 อยู่ k ตัว

การสร้าง combination

- แล้วจะสร้าง combination อย่างไร?
- ก็สร้าง subset ทั้งหมดขึ้นมา
- แล้วก็เลือกแค่ subset ที่มีสมาชิกทั้งหมด k ตัว

การสร้าง combination

```
int n, k, *g;

void generate(int i)
{
  if (k == 0) {
    if (has_k_elements(g, n, k)) {
      dosomething();
    }
  } else {
    for(g[k-1]=1;g[k-1]<=2;g[k-1]++)
      generate(k-1);
  }
}
```

การสร้าง combination

```
int has_k_elements(int *g, int n, int k)
{
  int count = 0, i;
  for(i=0;i<n;i++)
    if (g[i] == 1)
      count++;
  return (count == k);
}
```

ฟังก์ชันนี้มีไว้สำหรับเช็คว่าเซตที่แทนด้วยอะเรย์ g มีสมาชิก k ตัวหรือไม่

ประสิทธิภาพของการสร้าง combination

- การสร้าง combination แบบที่บอกเมื่อกี้ใช้เวลาเท่าไหร่?
- เท่ากับจำนวน subset ทั้งหมด = 2^n
- แต่ถ้า C(n,k) อาจมีค่าน้อยกว่า 2^n มากๆ
- โปรแกรมเราจึงช้ากว่าที่ควรจะเป็น
- เราสามารถทำได้ดีกว่านี้ได้ไหม?
- ได้สิ!

สร้าง combination ให้เร็วขึ้น

- เปลี่ยนวิธีการเก็บข้อมูลของ combination ใหม่
  - อะเรย์ int *c จำนวน k ช่อง
  - c[i] = เลขตัวที่ i ใน combination
  - เลขที่เก็บใน c จะเรียงจากน้อยไปหามาก
- เราจะเขียนฟังก์ชัน generate(int m, int l) เพื่อสร้าง combination ที่มีสมาชิก l ตัวของเซต {1,2,…,m}

สร้าง combination ให้เร็วขึ้น

- เราแบ่งการทำงานออกเป็น l ขั้นตอน
  - ขั้นแรก เราจะเลือกค่า c[l-1]
  - ขั้นที่สอง เราจะเลือกค่า c[l-2]
  - ขั้นที่สาม เราจะเลือกค่า c[l-3]
  - :
  - :
  - ขั้นที่ l เราจะเลือกค่า c[0]
- ข้อแม้ c[0] < c[1] < … < c[l-2] < c[l-1]

สร้าง combination ให้เร็วขึ้น

- พิจารณาค่า c[l-1] ก่อน
- มันมีค่ามากที่สุดเท่าไหร่ ค่าน้อยที่สุดเท่าไหร่?
- c[l-1] <= m เพราะ combination เป็น subset ของ {1,2,…,m}
- c[l-1] >= l เพราะใน combination ต้องมีเลข l ตัวและ c[l-1] เป็นเลขที่มากที่สุด

สร้าง combination ให้เร็วขึ้น

- เราจะเลือกค่า c[l-1] อย่างไร?

```
for(c[l-1]=l;c[l-1]<=m;c[l-1]++)
  . . .
```

สร้าง combination ให้เร็วขึ้น

- หลังจากเลือก c[l-1] เสร็จแล้วจะต้องทำอะไร?
  - เลือกค่า c[l-2], c[l-3], …, c[0]
  - แต่ละค่ามีค่าเท่าไหร่ได้บ้าง?: 1, 2, …, c[l-1]-1
  - เหมือนกับการเลือก subset ของ {1,2,…,c[l-1]-1} ซึ่งมีสมาชิก l-1 ตัว
- ดังนั้น:

```
for(c[l-1]=l;c[l-1]<=m;c[l-1]++)
  generate(c[l-1]-1,l-1);
```

สร้าง combination ให้เร็วขึ้น

- กรณีพิเศษ:
  - l = 0 แสดงว่าเราเลือก subset ครบแล้ว
  - ฉะนั้น dosomething();

สร้าง combination ให้เร็วขึ้น

```
void generate(int m, int l)
{
  if (l == 0)
    dosomething();
  else {
    for(c[l-1]=l;c[l-1]<=m;c[l-1]++)
      generate(c[l-1]-1, l-1);
  }
}
```

โปรแกรมพิมพ์ combination

```
int n, k, *c;

void generate(int m, int l)
{
  if (l == 0)
    print_combination(c,k);
  else {
    for(c[l-1]=l;c[l-1]<=m;c[l-1]++)
      generate(c[l-1]-1, l-1);
  }
}
```

โปรแกรมพิมพ์ combination

```
void print_combination(int *c, int k)
{
  int i;
  for(i=0;i<k;i++) {
    printf("%d ", c[i]);
  }
}
```

สรุป

- การแก้ปัญหาการหาค่าสูงสุด/ต่ำสุด
  - หาให้ได้ว่าวัตถุที่เราต้องการหาค่าคืออะไร
    - อาจเป็น combination
    - อาจเป็น permutation
    - อาจเป็นวิธีแบ่ง
    - ฯลฯ
  - หาวิธีการหาค่าของวัตถุนั้นคืออะไร
  - สร้างวัตถุที่เราต้องการหาทีละอัน
  - หาค่าของมันแล้วเอาไปเปรียบเทียบกับค่าที่มากสุด/น้อยสุดที่เคยหาได้
  - ถ้ามากกว่า/น้อยกว่า ก็เก็บวัตถุใหม่เอาไว้