5131017321 ณัฐรดา ลิ้มศุภวานิช
5131026021 นายปัญจมพงศ์ เสริมสวัสดิ์ศรี

Generating all possible answer

Combination

แสดงผลที่เป็นไปได้ทั้งหมดในการเลือกของ k ชิ้นจาก n ชิ้น

Combination with replacement : การเลือกของ k ชิ้นจาก n ชิ้นโดยเลือกซ้ำได้

ตัวอย่างปัญหา

- **Breaking the padlock :** ถอดรหัสแม่กุญแจ

ถ้าแม่กุญแจมีตัวเลขที่เป็นไปได้ 4 ตัวเลข และความยาวรหัสเท่ากับ 4

การแก้โจทย์ คือสร้างเลขทั้งหมดที่เป็นไปได้ เช่น มีตัวเลข ABCD สามารถแยกได้เป็น

AAAA

AAAB

…

AAAD

AABA<<เกิดจากการ backtracking : การที่เปลี่ยนหลักหลังจนครบแล้ว

…. จึงย้อนกลับไปเปลี่ยนค่าของหลักก่อนหน้า

AADD

…

ABAA

…

DDDD

โดย Framework ทั่วไปคือการกำหนด Initial Step แล้วใส่เข้าไปใน Data Structure จากนั้นวนลูปนำของออกมาจาก Data Structure ถ้าตรวจสอบแล้วว่าขั้นนั้นไม่ใช่คำตอบ แล้วใส่ขั้นต่อไปทั้งหมดที่เป็นไปได้เข้าไปใน Data Structure เป็นกระบวนการสร้าง state ทีละขั้นๆ ไปเรื่อยๆ โดย Data Structure ที่ใช้นั้นปกติจะมี Stack กับ Queue ซึ่งการใช้ Stack มักจะเขียนแทนด้วย Recursive

ในตัวอย่างเรื่อง Padlock นั้นใช้การ recursive แก้ปัญหาดังนี้

```
void backtracking(int step,int *sol){
  If(step<num_step){
        for(int i=0;i<num_symbol;i++){
```

```
                    sol[step]=i;
                    backtracking(step+1,sol);
        }else{
                check(sol);
        }
    }
```

Search Tree

จากตัวอย่าง Pad Lock

0

00

000

0000 <-- Check

000

0001 <-- Check

000

0002 <-- Check

000

0003 <-- Check

000

00

001

0010 <-- Check

▶ 8-Queen Problem

กำหนดตารางหมากรุกขนาด 8x8 มี queen 8 ตัว

ให้หาวิธีวาง Queen ทั้งหมดที่วาง Queen แล้วไม่ให้กินกันเองได้

(กินได้ 8 ทิศ จนสุดกระดาน)

การแก้ปัญหา : นิยาม search space กำหนดขอบเขตและขนาด เลือกทางเหมาะสมที่ดีที่สุด

1st Attempt

ตารางหมากรุกมี 64 ช่อง Queen 8 ตัว ขนาด 64^8

ตัวแปรที่ใช้เก็บคำตอบเป็นอาเรย์ความยาว 8 สองอัน เก็บ แกน x แกน y

ปัญหา : มีตำแหน่งของ queen ที่ซ้อนทับกันอยู่ด้วย

```
void e_queen(int step,int *mark_on_board)
{
  if (step < 64) {
    mark_on_board[step] = 0;
    e_queen(step + 1,mark_on_board);
    mark_on_board[step] = 1;
    e_queen(step + 1,mark_on_board);
  } else {
     check(mark_on_board);
  }
}
```

<u>2nd Attempt</u>

ลองเอาครั้งที่ซ้ำกันออก โดยใช้การเลือกช่องว่างว่าจะให้ Queen วางหรือไม่วาง จำนวนแบบทั้งหมดจึงเป็น 2^{64}

ปัญหาคือมีการวาง Queen เกินหรือขาดจาก 8 ตัวได้ แก้ได้โดย เพิ่มตัวแปรตอนเรียก Function ที่คอยตรวจสอบว่ามีการเลือก Queen ไปแล้วกี่ช่อง หรือสะดวกกว่านั้น เพิ่มตัวแปร 2 ตัวชื่อ one กับ zero ที่ระบุว่าจะต้องเลือก 1 กับ 0 อีกกี่ตัว วิธีนี้มีจำนวนแบบทั้งหมด 64C8

```
void e_queen(int step,int *mark_on_board)
{
  if (step < 64) {
    mark_on_board[step] = 0;
    e_queen(step + 1,mark_on_board);
    mark_on_board[step] = 1;
    e_queen(step + 1,mark_on_board);
  } else {
     check(mark_on_board);
  }
}
```

3rd Attempt

การวาง Queen ในแต่ละแถวจะต้องมี Queen 1 ตัวเท่านั้น วิธีนี้จะทำให้เหลือเพียง 8^8 วิธี

```
void e_queen(int step,int *queen_pos)
{
   if (step < 8) {
      for (int i =  0; i < 8; i++) {
      queen_pos[step] = i;
      e_queen(step + 1, queen_pos);
   }
   } else {
      check(queen_pos);
   }
}
```

4th Attempt

Queen ต้องไม่วางใน Column เดียวกันด้วย วิธีนี้ลงจำนวนวิธีทั้งหมดเหลือเพียง 8!

Permutation

การเรียงสับเปลี่ยน โดยไม่มีการเรียงซ้ำๆกัน

Permutation by Backtracking

ต้องมี Special Condition คือการจำว่าเคยใช้อะไรไปบ้างแล้ว โดยการตรวจสอบจากอาร์เรย์ sol ว่าใช้อะไรไปบ้างแล้ว วิธีที่ดีกว่าคือมีอาร์เรย์ของ boolean ว่าสัญลักษณ์ตัวนั้นถูกใช้ไปแล้วหรือยัง

```
void backtracking(int step,int *sol,bool *used) {
   if (step < num_step)
      for (int i =  0; i < num_symbol; i++)
         if (!used[i]) {
                  used[i] = true;
                  sol[step] = i;
                  backtracking(step,sol,used);
                  used[i] = false;
         } else check(sol); }
```