Sira Songpolrojjanakul

Basic Dynamic Programming

จำนวนฟีโบนัชชี (Fibonacci number)

- คือ จำนวนที่เกิดจากสองตัวก่อนหน้าบวกกัน
- เขียนเป็นความสัมพันธ์เวียนเกิดได้ว่า
- $$f(n) = \begin{cases} f(n-1) + f(n-2) & n \geq 2 \\ n & 0 \leq n < 2 \end{cases}$$
- เราจะหา f(n) อย่างไร

- เขียนเป็นฟังก์ชันเรียกตัวเองตรงๆ เลย

$$f(n) = \begin{cases} f(n-1) + f(n-2) & n \geq 2 \\ n & 0 \leq n < 2 \end{cases}$$

```
int f(int n)
{
    if(n<2)     return n;
    return f(n-1)+f(n-2);
}
```

ฟังก์ชันเรียกตัวเอง (recursive function)

- ประกอบด้วย 2 ส่วนหลักๆ คือ
- Base case คือ ขั้นที่เมื่อมาถึงแล้วจะไม่ทำการเรียกตัวเองต่อ
- Recursive case คือ ขั้นที่มาถึงแล้วจะมีการเรียกตัวเองต่อไป

Base case สำคัญมาก
ถ้าไม่มีมันก็จะไม่รู้ว่าหยุดที่ไหน

```
int f(int n)
{
    if(n<2) return n;          //base
    return  f(n-1)+f(n-2); //recursive
}
```

การใช้ฟังก์ชันเรียกตัวเอง

- ใช้แทน `for loop`
- ใช้ `search` หาทุกกรณี (ก็เหมือนอันบนนั่นแหละ)
- `! search` ทุกกรณี โดยใช้ `recursive` นี้มีชื่ออย่างทางการว่า `dfs (depth first search)` ซึ่งจะว่ากันทีหลัง
- ต่อไปจะเป็นตัวอย่างโจทย์

Skgrader: subset

- ระบุวิธีสร้าง `subset`ทั้งหมดของเซตที่มีขนาด `n (1<=n<=15)` ตอบเรียงลำดับตามตัวอักษรด้วย
- ตัวอย่าง `n=3`

```
Output
000
001
010
011
100
101
110
111
```

Analysis: subset

- หากเรารู้จำนวนตัวแน่ๆ เช่น n=3 ทำไง
- เราก็แค่ลูป 3 ชั้น แบบนี้

```
for(int i=0;i<=1;i++)
  for(int j=0;j<=1;j++)
    for(int k=0;k<=1;k++)
      printf("%d%d%d",i,j,k);
```

- แต่ข้อนี้ n มีค่าได้ตั้งแต่ 1 ถึง 15 นั่นคือถ้าจะใช้วิธีนี้ในข้อนี้ต้องแยกกรณีออกเป็น 15 แบบ และเขียนลูปตั้งแต่ 1-15 ชั้น **ซึ่งลำบาก**
- เลือกใช้ฟังก์ชันเรียกตัวเองดีกว่า

Analysis: subset (cont.)

- หัวฟังก์ชัน เป็นตัวนับว่าถึงลูปชั้นไหนแล้ว

```
void gen(int lv)
```

- แต่ละชั้นก็ต้องมีลูปวิ่งระหว่าง `0` กับ `1`

```
for(int i=0;i<=1;i++)
```

- เมื่อเข้าในลูป ก็ต้องไปเรียกชั้นต่อไป

```
for(int i=0;i<=1;i++)
   gen(lv+1);
```

Analysis: subset (cont.)

- สังเกตว่าเมื่อขึ้นลูปต่อไป มันก็จะมาทำฟังก์ชันใหม่ ก็คือไม่มีค่าอะไรในชั้นที่แล้วมาด้วยเลย นั่นคือควรจะมีตัวแปร `global` เก็บค่าเพื่อให้คนอื่นมาเข้าถึงได้

```
for(int i=0;i<=1;i++)
  data[lv]=i; // อยู่ที่ชั้น lv เลยเก็บที่ช่องนี้
  gen(lv+1);
```

Analysis: subset (cont.)

- แล้วทำลูปไปถึงไหน?
- ก็ทำถึง `n`
- สมมติเราเรียก `gen(0)` เป็นตัวแรก นั่นคือเราวิ่งไปถึง `n-1` ก็จะทำครบ `n` รอบ นั่นคือในรอบที่ `lv==n` เราจะเข้าสู่ลูปในรอบ `n+1`

```
if(lv==n)  // n is global
  printf data
  return
```

Sol: subset

- เขียนส่วนฟังก์ชันนี้ได้ว่า

```
int n,data[20];
void gen(int lv) {
  if(lv==n) {
    for(int i=0;i<n;i++)
      printf("%d",data[i]);
    printf("\n");
    return;
  }
  for(int i=0;i<=1;i++) {
    data[lv]=i;
    gen(lv+1);
  }
}
```

Simulation: subset

- ตัวอย่างการทำงานของโค้ดเมื่อ `n=3`
- เริ่มต้นเราจะเรียก `gen(0);` ใน `main`

```
void gen(int lv) {
  if(lv==n) {
    PRINT
    return;
  }
  for(int i=0;i<=1;i++) {
    data[lv]=i;
    gen(lv+1);
  }
}
```

Simulation: subset

- lv=0
- data: 0 0 0 0

```
void gen(int lv) {
  if(lv==n) {
    PRINT
    return;
  }
  for(int i=0;i<=1;i++) {
    data[lv]=i;
    gen(lv+1);
  }
}
```

Simulation: subset

- lv!=n
- data: 0 0 0 0

```
void gen(int lv) {
  if(lv==n) {
    PRINT
    return;
  }
  for(int i=0;i<=1;i++) {
    data[lv]=i;
    gen(lv+1);
  }
}
```

Simulation: subset

- lv=0 i=0
- data: 0 0 0 0

```
void gen(int lv) {
  if(lv==n) {
    PRINT
    return;
  }
  for(int i=0;i<=1;i++) {
    data[lv]=i;
    gen(lv+1);
  }
}
```

Simulation: subset

- lv=0 i=0
- data: 0 0 0 0

```
void gen(int lv) {
  if(lv==n) {
    PRINT
    return;
  }
  for(int i=0;i<=1;i++) {
    data[lv]=i;
    gen(lv+1);
  }
}
```

Simulation: subset

- ไปทำ `gen(lv+1)` ก็คือ `gen(1)`
- `data: 0 0 0 0`

```
void gen(int lv) {
  if(lv==n) {
    PRINT
    return;
  }
  for(int i=0;i<=1;i++) {
    data[lv]=i;
    gen(lv+1);
  }
}
```

Simulation: subset

- lv=1
- data: 0 0 0 0

```
void gen(int lv) {
  if(lv==n) {
    PRINT
    return;
  }
  for(int i=0;i<=1;i++) {
    data[lv]=i;
    gen(lv+1);
  }
}
```

lv=0 / i=0 / line 8

Simulation: subset

- lv!=n
- data: 0 0 0 0

```
void gen(int lv) {
  if(lv==n) {
    PRINT
    return;
  }
  for(int i=0;i<=1;i++) {
    data[lv]=i;
    gen(lv+1);
  }
}
```

lv=0 / i=0 / line 8

Simulation: subset

- lv=1 i=0
- data: 0 0 0 0

```
void gen(int lv) {
  if(lv==n) {
    PRINT
    return;
  }
  for(int i=0;i<=1;i++) {
    data[lv]=i;
    gen(lv+1);
  }
}
```

lv=0 / i=0 / line 8

Simulation: subset

- `lv=1  i=0`
- `data: 0 0 0 0`

```
void gen(int lv) {
  if(lv==n) {
    PRINT
    return;
  }
  for(int i=0;i<=1;i++) {
    data[lv]=i;
    gen(lv+1);
  }
}
```

lv=o / i=o / line 8

Simulation: subset

- ไปทำ `gen(lv+1)` ก็คือ `gen(2)`
- `data: 0 0 0 0`

```
void gen(int lv) {
  if(lv==n) {
    PRINT
    return;
  }
  for(int i=0;i<=1;i++) {
    data[lv]=i;
    gen(lv+1);
  }
}
```

lv=0 / i=0 / line 8

Simulation: subset

- `lv=2`
- `data: 0 0 0 0`

```
void gen(int lv) {
  if(lv==n) {
    PRINT
    return;
  }
  for(int i=0;i<=1;i++) {
    data[lv]=i;
    gen(lv+1);
  }
}
```

lv=1 / i=0 / line 8

lv=0 / i=0 / line 8

Simulation: subset

- `lv!=n`
- `data: 0 0 0 0`

```
void gen(int lv) {
  if(lv==n) {
    PRINT
    return;
  }
  for(int i=0;i<=1;i++) {
    data[lv]=i;
    gen(lv+1);
  }
}
```

lv=1 / i=0 / line 8

lv=0 / i=0 / line 8

Simulation: subset

- `lv=2  i=0`
- `data: 0 0 0 0`

```
void gen(int lv) {
  if(lv==n) {
    PRINT
    return;
  }
  for(int i=0;i<=1;i++) {
    data[lv]=i;
    gen(lv+1);
  }
}
```

lv=1 / i=0 / line 8

lv=0 / i=0 / line 8

Simulation: subset

- `lv=2  i=0`
- `data: 0 0 0 0`

```
void gen(int lv) {
  if(lv==n) {
    PRINT
    return;
  }
  for(int i=0;i<=1;i++) {
    data[lv]=i;
    gen(lv+1);
  }
}
```

lv=1 / i=0 / line 8

lv=0 / i=0 / line 8

Simulation: subset

- ไปทำ `gen(lv+1)` ก็คือ `gen(3)`
- data: 0 0 0 0

```
void gen(int lv) {
  if(lv==n) {
    PRINT
    return;
  }
  for(int i=0;i<=1;i++) {
    data[lv]=i;
    gen(lv+1);
  }
}
```

lv=1 / i=0 / line 8

lv=0 / i=0 / line 8

Simulation: subset

- `lv=3`
- `data: 0 0 0 0`

```
void gen(int lv) {
  if(lv==n) {
    PRINT
    return;
  }
  for(int i=0;i<=1;i++) {
    data[lv]=i;
    gen(lv+1);
  }
}
```

lv=2 / i=0 / line 8

lv=1 / i=0 / line 8

lv=0 / i=0 / line 8

Simulation: subset

- lv==n
- data: 0 0 0 0

```
void gen(int lv) {
  if(lv==n) {
    PRINT
    return;
  }
  for(int i=0;i<=1;i++) {
    data[lv]=i;
    gen(lv+1);
  }
}
```

lv=2 / i=0 / line 8

lv=1 / i=0 / line 8

lv=0 / i=0 / line 8

Simulation: subset

- **PRINT 000**
- data: 0 0 0 0

```
void gen(int lv) {
  if(lv==n) {
    PRINT
    return;
  }
  for(int i=0;i<=1;i++) {
    data[lv]=i;
    gen(lv+1);
  }
}
```

lv=2 / i=0 / line 8

lv=1 / i=0 / line 8

lv=0 / i=0 / line 8

Simulation: subset

- return ไปยังชั้นที่แล้ว
- data: 0 0 0 0

```
void gen(int lv) {
  if(lv==n) {
    PRINT
    return;
  }
  for(int i=0;i<=1;i++) {
    data[lv]=i;
    gen(lv+1);
  }
}
```

lv=2 / i=0 / line 8

lv=1 / i=0 / line 8

lv=0 / i=0 / line 8

Simulation: subset

- กลับมายังจุดเดิมที่บันทึกไว้ก่อนที่กระโดดไปทำชั้นถัดไป แล้วทำต่อจากจุดนี้
- data: 0 0 0 0

```
void gen(int lv) {
  if(lv==n) {
    PRINT
    return;
  }
  for(int i=0;i<=1;i++) {
    data[lv]=i;
    gen(lv+1);
  }
}
```

lv=2 / i=0 / line 8

lv=1 / i=0 / line 8

lv=0 / i=0 / line 8

Simulation: subset

- ทำบรรทัดต่อจากบรรทัดนี้ (lv=2,i=0)
- data: 0 0 0 0

```
void gen(int lv) {
  if(lv==n) {
    PRINT
    return;
  }
  for(int i=0;i<=1;i++) {
    data[lv]=i;
    gen(lv+1);
  }
}
```

lv=1 / i=0 / line 8

lv=0 / i=0 / line 8

Simulation: subset

- lv=2 i=1
- data: 0 0 0 0

```
void gen(int lv) {
  if(lv==n) {
    PRINT
    return;
  }
  for(int i=0;i<=1;i++) {
    data[lv]=i;
    gen(lv+1);
  }
}
```

lv=1 / i=0 / line 8

lv=0 / i=0 / line 8

Simulation: subset

- `lv=2  i=1`
- `data: 0 0 1 0`

```
void gen(int lv) {
  if(lv==n) {
    PRINT
    return;
  }
  for(int i=0;i<=1;i++) {
    data[lv]=i;
    gen(lv+1);
  }
}
```

lv=1 / i=0 / line 8

lv=0 / i=0 / line 8

Simulation: subset

- ไปทำ `gen(lv+1)` ก็คือ `gen(3)`
- `data: 0 0 1 0`

```
void gen(int lv) {
  if(lv==n) {
    PRINT
    return;
  }
  for(int i=0;i<=1;i++) {
    data[lv]=i;
    gen(lv+1);
  }
}
```

lv=1 / i=0 / line 8

lv=0 / i=0 / line 8

Simulation: subset

- lv=3
- data: 0 0 1 0

```
void gen(int lv) {
  if(lv==n) {
    PRINT
    return;
  }
  for(int i=0;i<=1;i++) {
    data[lv]=i;
    gen(lv+1);
  }
}
```

lv=2 / i=1 / line 8

lv=1 / i=0 / line 8

lv=0 / i=0 / line 8

Simulation: subset

- `lv==n`
- `data: 0 0 1 0`

```
void gen(int lv) {
  if(lv==n) {
    PRINT
    return;
  }
  for(int i=0;i<=1;i++) {
    data[lv]=i;
    gen(lv+1);
  }
}
```

lv=2 / i=1 / line 8

lv=1 / i=0 / line 8

lv=0 / i=0 / line 8

Simulation: subset

- **PRINT** 001
- data: 0 0 1 0

```
void gen(int lv) {
  if(lv==n) {
    PRINT
    return;
  }
  for(int i=0;i<=1;i++) {
    data[lv]=i;
    gen(lv+1);
  }
}
```

lv=2 / i=1 / line 8

lv=1 / i=0 / line 8

lv=0 / i=0 / line 8

Simulation: subset

- return ไปยังชั้นที่แล้ว
- data: 0 0 1 0

```
void gen(int lv) {
  if(lv==n) {
    PRINT
    return;
  }
  for(int i=0;i<=1;i++) {
    data[lv]=i;
    gen(lv+1);
  }
}
```

lv=2 / i=1 / line 8

lv=1 / i=0 / line 8

lv=0 / i=0 / line 8

Simulation: subset

- กลับมายังจุดเดิมที่บันทึกไว้ก่อนที่กระโดดไปทำชั้นถัดไป แล้วทำต่อจากจุดนี้
- data: 0 0 1 0

```
void gen(int lv) {
  if(lv==n) {
    PRINT
    return;
  }
  for(int i=0;i<=1;i++) {
    data[lv]=i;
    gen(lv+1);
  }
}
```

$\boxed{\text{lv=2 / i=1 / line 8}}$

lv=1 / i=0 / line 8

lv=0 / i=0 / line 8

Simulation: subset

- ทำบรรทัดต่อจากบรรทัดนี้ (lv=2,i=1)
- data: 0 0 1 0

```
void gen(int lv) {
  if(lv==n) {
    PRINT
    return;
  }
  for(int i=0;i<=1;i++) {
    data[lv]=i;
    gen(lv+1);
  }
}
```

lv=1 / i=0 / line 8

lv=0 / i=0 / line 8

Simulation: subset

- lv=2 i=2 หลุดลูป
- data: 0 0 1 0

```
void gen(int lv) {
  if(lv==n) {
    PRINT
    return;
  }
  for(int i=0;i<=1;i++) {
    data[lv]=i;
    gen(lv+1);
  }
}
```

lv=1 / i=0 / line 8

lv=0 / i=0 / line 8

Simulation: subset

- จบฟังก์ชัน กลับไปทำต่อที่ชั้นก่อนหน้า
- data: 0 0 1 0

```
void gen(int lv) {
  if(lv==n) {
    PRINT
    return;
  }
  for(int i=0;i<=1;i++) {
    data[lv]=i;
    gen(lv+1);
  }
}
```

lv=1 / i=0 / line 8

lv=0 / i=0 / line 8

Simulation: subset

- ทำบรรทัดต่อจากบรรทัดนี้ (lv=1,i=0)
- data: 0 0 1 0

```
void gen(int lv) {
  if(lv==n) {
    PRINT
    return;
  }
  for(int i=0;i<=1;i++) {
    data[lv]=i;
    gen(lv+1);
  }
}
```

lv=0 / i=0 / line 8

Simulation: subset

- `lv=1  i=1`
- `data: 0 0 1 0`

```
void gen(int lv) {
  if(lv==n) {
    PRINT
    return;
  }
  for(int i=0;i<=1;i++) {
    data[lv]=i;
    gen(lv+1);
  }
}
```

lv=o / i=o / line 8

Simulation: subset

- lv=1 i=1
- data: 0 1 1 0

```
void gen(int lv) {
  if(lv==n) {
    PRINT
    return;
  }
  for(int i=0;i<=1;i++) {
    data[lv]=i;
    gen(lv+1);
  }
}
```

lv=0 / i=0 / line 8

Simulation: subset

- ไปทำ gen(lv+1) ก็คือ gen(2)
- data: 0 1 1 0

```
void gen(int lv) {
  if(lv==n) {
    PRINT
    return;
  }
  for(int i=0;i<=1;i++) {
    data[lv]=i;
    gen(lv+1);
  }
}
```

lv=0 / i=0 / line 8

Simulation: subset

- lv=2
- data: 0 1 1 0

```
void gen(int lv) {
  if(lv==n) {
    PRINT
    return;
  }
  for(int i=0;i<=1;i++) {
    data[lv]=i;
    gen(lv+1);
  }
}
```

lv=1 / i=1 / line 8

lv=0 / i=0 / line 8

Simulation: subset

- lv!=n
- data: 0 1 1 0

```
void gen(int lv) {
  if(lv==n) {
    PRINT
    return;
  }
  for(int i=0;i<=1;i++) {
    data[lv]=i;
    gen(lv+1);
  }
}
```

lv=1 / i=1 / line 8

lv=0 / i=0 / line 8

Simulation: subset

- lv=2 i=0
- data: 0 1 1 0

```
void gen(int lv) {
  if(lv==n) {
    PRINT
    return;
  }
  for(int i=0;i<=1;i++) {
    data[lv]=i;
    gen(lv+1);
  }
}
```

lv=1 / i=1 / line 8

lv=0 / i=0 / line 8

Simulation: subset

- lv=2 i=0
- data: 0 1 0 0

```
void gen(int lv) {
  if(lv==n) {
    PRINT
    return;
  }
  for(int i=0;i<=1;i++) {
    data[lv]=i;
    gen(lv+1);
  }
}
```

lv=1 / i=1 / line 8

lv=0 / i=0 / line 8

Simulation: subset

- ไปทำ gen(lv+1) ก็คือ gen(3)
- data: 0 1 0 0

```
void gen(int lv) {
  if(lv==n) {
    PRINT
    return;
  }
  for(int i=0;i<=1;i++) {
    data[lv]=i;
    gen(lv+1);
  }
}
```

lv=1 / i=1 / line 8

lv=0 / i=0 / line 8

Simulation: subset

- lv=3
- data: 0 1 0 0

```
void gen(int lv) {
  if(lv==n) {
    PRINT
    return;
  }
  for(int i=0;i<=1;i++) {
    data[lv]=i;
    gen(lv+1);
  }
}
```

lv=2 / i=0 / line 8

lv=1 / i=1 / line 8

lv=0 / i=0 / line 8

Simulation: subset

- `lv==n`
- `data: 0 1 0 0`

```
void gen(int lv) {
  if(lv==n) {
    PRINT
    return;
  }
  for(int i=0;i<=1;i++) {
    data[lv]=i;
    gen(lv+1);
  }
}
```

lv=2 / i=0 / line 8

lv=1 / i=1 / line 8

lv=0 / i=0 / line 8

Simulation: subset

- **PRINT** 010
- data: 0 1 0 0

```
void gen(int lv) {
  if(lv==n) {
    PRINT
    return;
  }
  for(int i=0;i<=1;i++) {
    data[lv]=i;
    gen(lv+1);
  }
}
```

lv=2 / i=0 / line 8

lv=1 / i=1 / line 8

lv=0 / i=0 / line 8

Simulation: subset

- `return` ไปยังชั้นที่แล้ว
- `data: 0 1 0 0`

```
void gen(int lv) {
  if(lv==n) {
    PRINT
    return;
  }
  for(int i=0;i<=1;i++) {
    data[lv]=i;
    gen(lv+1);
  }
}
```

lv=2 / i=0 / line 8

lv=1 / i=1 / line 8

lv=0 / i=0 / line 8

Simulation: subset

- กลับมายังจุดเดิมที่บันทึกไว้ก่อนที่กระโดดไปทำชั้นถัดไป แล้วทำต่อจากจุดนี้
- data: 0 1 0 0

```
void gen(int lv) {
  if(lv==n) {
    PRINT
    return;
  }
  for(int i=0;i<=1;i++) {
    data[lv]=i;
    gen(lv+1);
  }
}
```

ทำเช่นนี้ไปเรื่อยๆ จนจบลูปทั้งหมด

$\boxed{\text{lv=2 / i=0 / line 8}}$

lv=1 / i=1 / line 8

lv=0 / i=0 / line 8

Need to know

- ถ้าเราไม่อยากใช้ตัวแปร `global` เราสามารถส่งตัวแปรไปในฟังก์ชันด้วยก็ได้ ซึ่งมีสองลักษณะ
- `Pass by value` คือ ส่งค่าเข้าไปอย่างเดียว ถ้าแก้ค่านี้ในฟังก์ชันอื่น ถ้าจะไม่เปลี่ยนแปลง

เช่น `gen(int n)` ไปทำอะไร `n` ในนี้ก็ไม่ส่งผลต่อค่า `n` เมื่อตัวอื่นมาเข้าถึง

- `Pass by reference` คือ ส่งที่อยู่ไป ถ้าแก้ค่าในฟังก์ชัน ค่านี้ก็จะเปลี่ยนไปด้วย

เช่น `gen(int &a)` คือ ส่งตัวแปรเดียวแบบ `ref`
`gen(int a[]) / gen(int *a)` คือ ส่งอาเรย์เข้ามาในฟังก์ชัน **[array pass by value ไม่ได้]**
`* Function` ไม่มี `parameter` ก็ได้

Need to know

- Return type ที่ระบุที่หัวฟังก์ชัน มีได้หลายลักษณะ ดังนี้
- ไม่คืนค่า → void
- พื้นฐาน → int/char/double/bool อื่นๆ
- โครงสร้างข้อมูล ให้ระบุชื่อนั้นเลย เช่น node funt()
- อาเรย์ ให้ใส่ * หลังจากtype เช่น int* funt()
- ข้อมูลอื่นๆ ของ C++ ให้ระบุชื่อนั้นเลย เช่น string f()/ vector<int> f()

Did you know

- รู้หรือไม่ Parameter กับ Argument ต่างกันอย่างไร
- Parameter คือ สิ่งที่ต้องใส่ในฟังก์ชัน
- Argument คือ ค่าที่ใส่ในฟังก์ชัน
- เช่น
- fibo(n) // n is parameter
- fibo(5) // 5 is argument

Skgrader: light_puzzle

- มีไฟขนาด 4x4 ระบุสถานะเริ่มต้นให้
- เมื่อกดสวิตช์ไฟดวง(i,j) สวิคช์ไฟดวง (i+1,j),(i-1,j),(i,j-1),(i,j+1) จะถูกกดด้วย
- การกดสวิตช์ไฟดวงที่เปิดอยู่ จะเปลี่ยนเป็นปิด /กดสวิตช์ไฟดวงที่ปิดอยู่ จะเปลี่ยนเป็นเปิด
- ต้องการให้ไฟทุกดวงเปิด ต้องกดอย่างไร

Analysis: light_puzzle

- พิจารณาสวิตช์ไฟ ที่ X พบว่า กดครั้งแรกสถานะจะเปลี่ยนไป
- กด 2 ครั้งสถานะก็จะเป็นเหมือนเดิม เหมือนว่ายังไม่ได้กด
- กด 3 ครั้ง ก็ไม่ต่างจากกดครั้งเดียว
- นั่นคือ เราเลือกแค่ว่าจะกด หรือไม่กดไฟดวงนั้นก็เพียงพอแล้ว
- ฉะนั้น เราก็ลองกดมันทุกแบบ หาดูว่าแบบไหนใช้ได้
- ซึ่งเราต้องทำทั้งสิ้น 2^16 = 65536 ครั้ง [OK]

มาดู fibo กันต่อ

- มาหา fibo ตัวที่ n กัน
- ต่อไปนี้จะจำลองการหา fibo ตัวที่ 5 ด้วยโค้ดที่แสดงไว้เมื่อตอนแรก
- เรียก f(5) เพื่อหา fibo ตัวที่ 5

```
int f(int n)
{
    if(n<2)   return n;
    return    f(n-1)+f(n-2);
}
```

Example: Fibo 5

```
f(5)
```

Example: Fibo 5

f(5)

f(4)

Example: Fibo 5

f(5)

f(4)

f(3)

Example: Fibo 5

```
                  f(5)
              f(4)
          f(3)
      f(2)
```

Example: Fibo 5

```
                      f(5)
                 f(4)
            f(3)
       f(2)
  f(1)
```

Example: Fibo 5

```
                    f(5)
                f(4)
            f(3)
        f(2)
    1
```

Example: Fibo 5

```
                    f(5)
                f(4)
            f(3)
        f(2)
1       f(0)
```

Example: Fibo 5

```
                    f(5)
                f(4)
            f(3)
        f(2)
1       0
```

Example: Fibo 5

```
                    f(5)
                f(4)
            f(3)
       1
   1   0
```

Example: Fibo 5

```
                f(5)
            f(4)
        f(3)
    1   f(1)
1   0
```

Example: Fibo 5

```
                    f(5)
                f(4)
            f(3)
       1    1
   1   0
```

Example: Fibo 5

```
                    f(5)
                f(4)
            2
        1   1
    1   0
```

Example: Fibo 5

```
                    f(5)
                f(4)
        2       f(2)
    1   1
1   0
```

Example: Fibo 5

```
                  f(5)
              f(4)
          2   f(2)
      1   1   f(1)
  1   0
```

Example: Fibo 5

```
                f(5)
           f(4)
      2    f(2)
  1   1    1
1 0
```

Example: Fibo 5

```
                f(5)
            f(4)
        2   f(2)
    1   1   1   f(0)
1   0
```

Example: Fibo 5

```
                f(5)
            f(4)
        2   f(2)
    1   1   1   0
1   0
```

Example: Fibo 5

```
                    f(5)
               f(4)
          2    1
     1    1    1    0
1    0
```

Example: Fibo 5

```
                f(5)
            3
        2   1
    1   1   1   0
1   0
```

Example: Fibo 5

```
                f(5)
            3          f(3)
        2   1
    1   1   1   0
1   0
```

Example: Fibo 5

```
                  f(5)
              3       f(3)
          2   1       f(2)
      1   1   1   0
  1   0
```

Example: Fibo 5

```
                  f(5)
              3       f(3)
          2   1       f(2)
      1   1   1   0   f(1)
  1   0
```

Example: Fibo 5

```
                f(5)
            3       f(3)
        2   1       f(2)
    1   1   1   0   1
1   0
```

Example: Fibo 5

```
                    f(5)
          3              f(3)
     2    1              f(2)
1    1    1    0    1         f(0)
1    0
```

Example: Fibo 5

```
                    f(5)
               3         f(3)
          2    1         f(2)
     1    1    1    0    1    0
1    0
```

Example: Fibo 5

```
                 f(5)
            3          f(3)
        2   1          1
    1   1   1   0   1   0
1   0
```

Example: Fibo 5

```
               f(5)
          3         f(3)
     2    1         1         f(1)
1    1    1    0    1    0
1    0
```

Example: Fibo 5

```
                 f(5)
             3        f(3)
         2   1        1        1
     1   1   1   0    1   0
 1   0
```

Example: Fibo 5

```
                    f(5)
            3              2
        2   1              1        1
    1   1   1   0   1   0
1   0
```

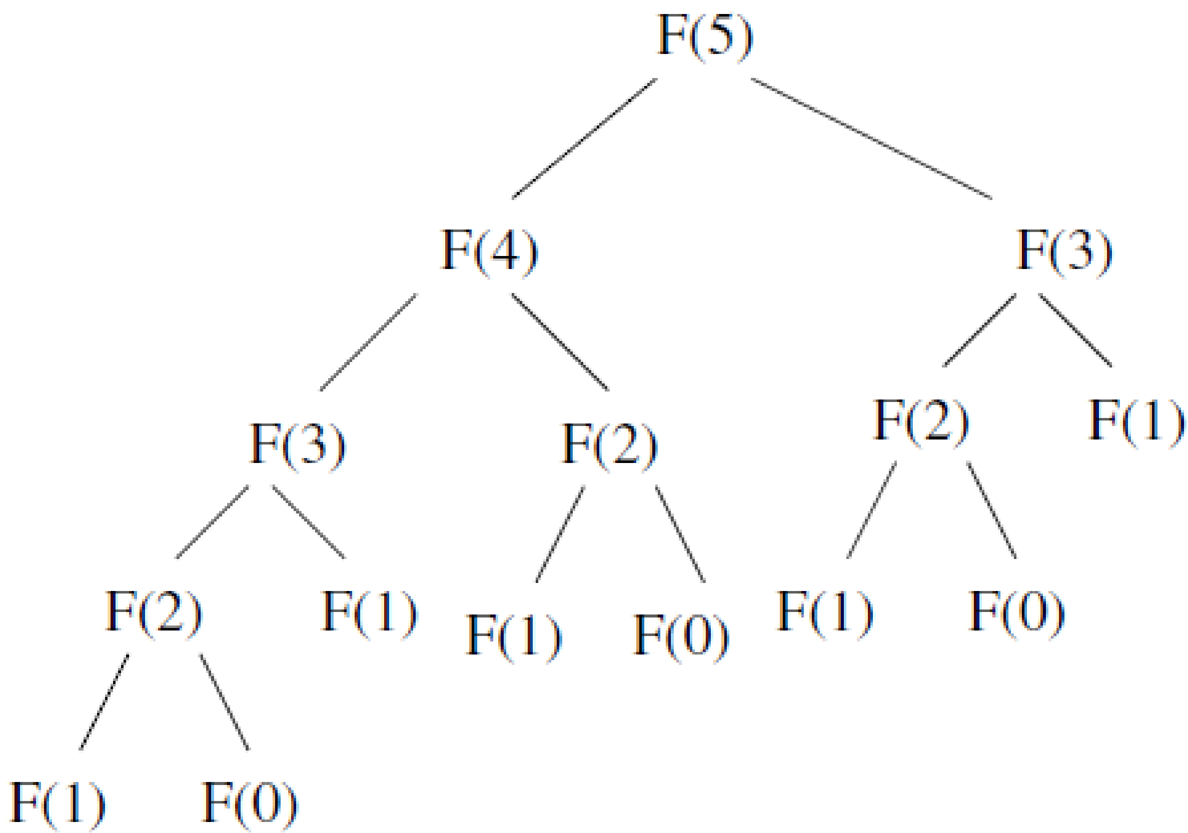
Example: Fibo 5

```
                  5
              3       2
          2   1       1       1
      1   1   1   0   1   0
  1   0
```

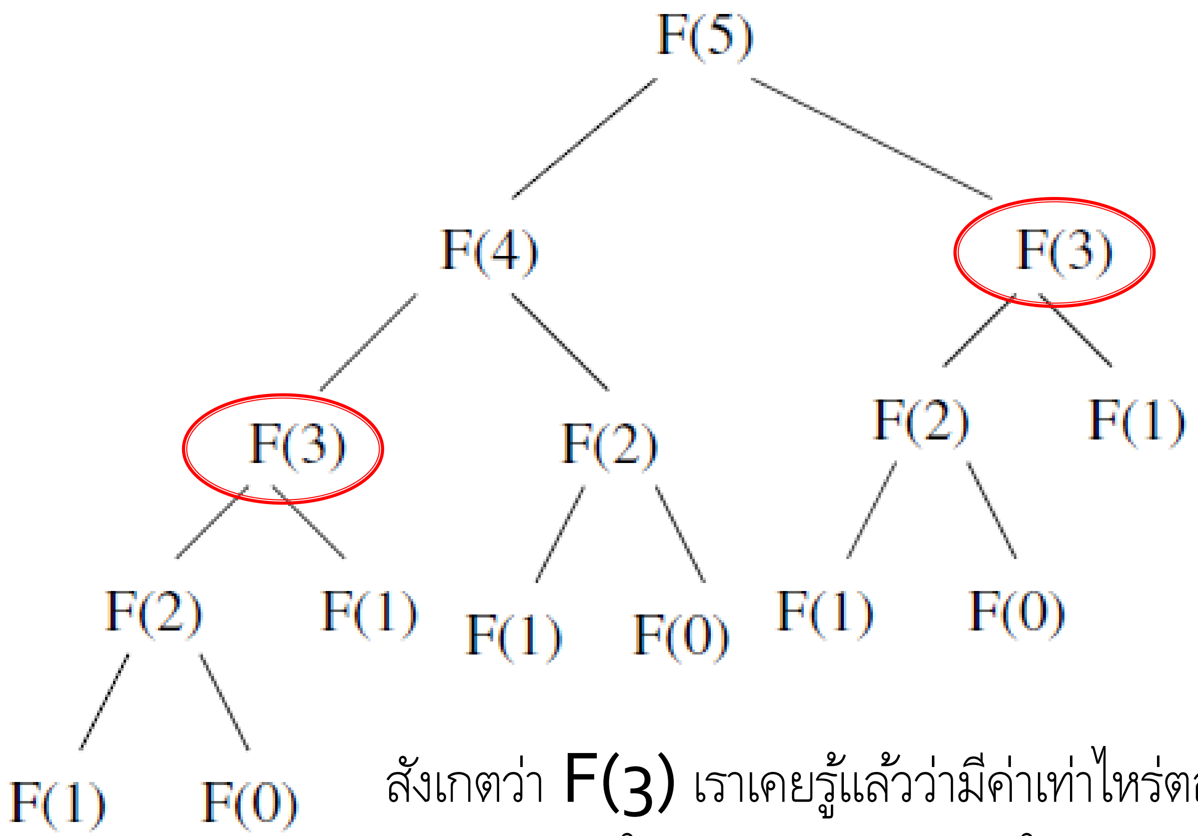
Fibo(5)=5

วิเคราะห์ประสิทธิภาพ fibo

- การหา `fibo` ด้วยวิธีดังกล่าว มีประสิทธิภาพเท่าใด
- จาก $\frac{F_{n+1}}{F_n} \approx \phi = \frac{1+\sqrt{5}}{2} \approx 1.61803$
- นั่นคือ $F_n > 1.6^n$
- แสดงว่า ต้องเรียกฟังก์ชันประมาณ 1.6^n ครั้ง
- ลองใส่ n=45 ดูว่ามันช้าแค่ไหนกว่าจะได้คำตอบ
- แล้วมันมีวิธีที่ดีกว่านี้มั้ย



พิจารณาลำดับการคิดเมื่อครู่นี้ แล้วเห็นอะไรบ้าง

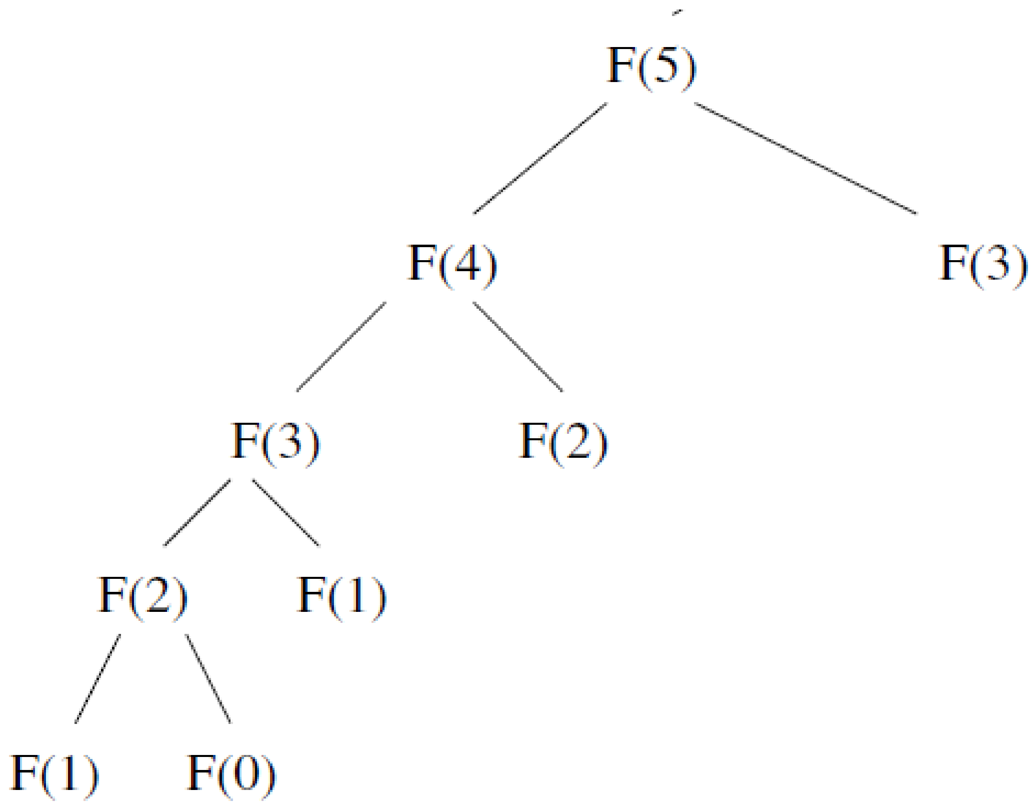


สังเกตว่า F(3) เราเคยรู้แล้วว่ามีค่าเท่าไหร่ตอนคิดทางซ้าย แต่พอมาถามใหม่ด้านขวา ก็ต้องคิดใหม่อีก

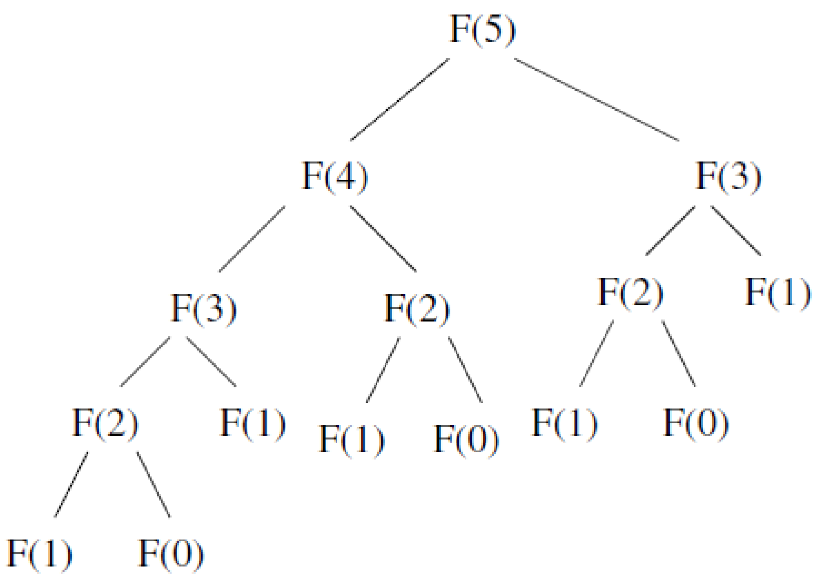
หยุดพินิจ

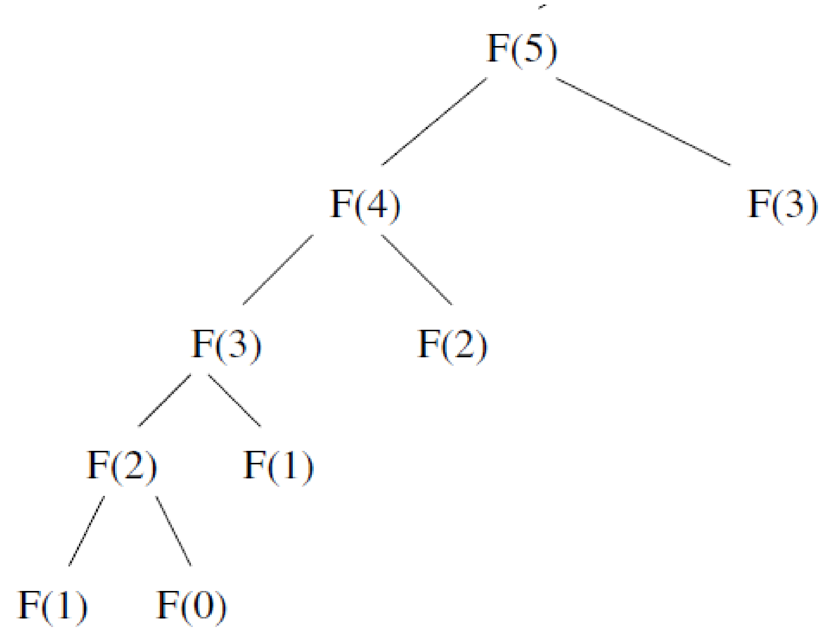
- แล้วทำไมมันต้องคิดใหม่ด้วยล่ะ ?
- ก็มันไม่มีอะไรเก็บค่าเดิมไว้ เลยต้องหาใหม่ตลอด
- ถ้าใส่ตัวจำเข้าไป จะเป็นอย่างไร

ลำดับการคิดเมื่อมีตัวจำ



เปรียบเทียบลำดับการคิด ไม่มีตัวจำ กับมีตัวจำ





จะเห็นว่าลดการคำนวณไปพอสมควรเลย

แล้วมันลดลงไปซักแค่ไหน

วิเคราะห์ประสิทธิภาพ fibo เมื่อมีตัวจำ

- การหา `fibo` แบบที่มีตัวจำนี้จะเห็นว่ามันจะดำลงไปทำเรื่อยๆ วิ่งจาก `n` สู่ `0`
- แต่มันจะทำเพียงครั้งเดียว ถ้ามีการเรียกใช้ค่า `f(x)`อีก จะไม่มีการคิดใหม่
- นั่นคือจะได้ว่า การหา `fibo` โดยเพิ่มตัวจำเข้าไป จะมีประสิทธิภาพเป็น `O(n)`
- ดีกว่าเดิมเยอะเลย แต่ว่าเขียนอย่างไร

Fibo แบบมีตัวจำ

โค้ดเดิม

```
int data[127];     // ใส่ตัวจำเข้าไป
int f(int n)
{
    if(n<2)   return n;
    data[n]=f(n-1)+f(n-2);
        // เก็บค่าลงในตาราง
    return    f(n-1)+f(n-2);
    return    data[n]; // ใช้อันนี้แทน
}
```

คิดว่าโอเค หรือยัง

Fibo แบบมีตัวจำ

```
int data[127];      // ใส่ตัวจำเข้าไป
int f(int n)
{

    if(n<2)   return n;
    data[n]=f(n-1)+f(n-2);
        // เก็บค่าลงในตาราง

    return   data[n]; // ใช้อันนี้แทน
}
```

ยัง หากลองไล่โค้ดดูจะพบว่า เราก็ยังต้องคิดใหม่ทุกรอบ
เพราะเราเก็บข้อมูลลงตารางจริงอยู่ แต่ไม่เคยเอามาใช้เลย

Fibo แบบมีตัวจำ

```
int data[127];      // ใส่ตัวจำเข้าไป
int f(int n)
{
    if(data[n]>0)  return data[n];
    if(n<2)   return n;
    data[n]=f(n-1)+f(n-2);
        // เก็บค่าลงในตาราง

    return    data[n]; // ใช้อันนี้แทน
}
```

สรุป (ในส่วนนี้)

- เอาโค้ด `fibo` ที่มีตัวจำไปรัน พบว่า `n` เป็นล้าน ยังรันได้ดี ในขณะที่แบบไม่มีตัวจำแค่ 45 ก็เริ่มแย่แล้ว
- การเขียนโปรแกรมในลักษณะนี้มีชื่อเรียกว่า กำหนดการเชิงพลวัต (dynamic programming) ซึ่งอธิบายลักษณะเฉพาะได้ดังนี้

ลักษณะปัญหาของกำหนดการเชิงพลวัต

1. มีโครงสร้างปัญหาย่อยเหมาะสุด (optimal sub-structure)
2. มีการซ้อนเหลื่อมกันของปัญหาย่อย(overlapping sub-problem)

โครงสร้างปัญหาย่อยเหมาะสุด

- ก็คือ คำตอบของปัญหาย่อยๆ เป็นส่วนหนึ่งของปัญหาที่ใหญ่กว่า
- ตัวอย่างเช่น `fibo(2)` เป็นค่าที่ดีที่สุดและเป็นที่สิ้นสุดแล้ว ถ้าค่าอื่นๆ มี `fibo(2)` เป็นส่วนประกอบด้วย ก็จะเอาค่า `fibo(2)`นี้ไปใช้เป็นส่วนประกอบไปเลย โดยไม่ต้องมาคิด `fibo(2)`อีก

การซ้อนเหลื่อมกันของปัญหาย่อย

- ก็คือ ในการคิดคำตอบออกมานั้น มีการเข้าไปทำปัญหาย่อยที่เคยทำไปแล้วอีก
- เราก็มีตัวจำมันซะ ก็ไม่ต้องไปทำใหม่ ทำให้เร็วขึ้น
- ที่มีตัวจำแล้วใช้ได้เพราะว่า ค่าที่จำนั้นมันสิ้นสุดแล้ว (มีสมบัติข้อ1)
- ถ้าไม่มีการซ้อนเหลื่อมกัน ก็ใช้วิธีนี้ไม่ได้ (ใช้แล้วไม่work) เพราะเราทำกำหนดการเชิงพลวัตเพราะเราจะตัดส่วนที่มันคิดซ้ำทิ้ง

จะทำกำหนดการเชิงพลวัตต้องหาอะไร

1. State นิยามสถานะแต่ละจุดว่าคืออะไร
2. Transition จากสถานะนี้ เปลี่ยนไปสถานะอื่นอย่างไร หรือกว่าจะมาที่สถานะนี้ได้ต้องผ่านอะไรมา

จะทำกำหนดการเชิงพลวัตต้องสนใจอะไร

1. Space คือ memory ที่ใช้จำคำตอบ
2. Time คือ เวลาที่ใช้กว่าจะได้คำตอบ

รูปแบบ state

- มีสามลักษณะ คือ

1. ณ

   คือ ตอนนี้เราอยู่ ณ ตำแหน่งไหน

2. ช่วง

   คือ ช่วงที่เรากำลังพิจารณา

3. เช็ค *

   คือ bitmask ที่เราให้จำว่า อันไหนบ้างที่เราทำไปแล้ว

รูปแบบ transition

- มีสองลักษณะ คือ

1. จุดนี้ มาจากไหน

   แบบนี้มันพบได้บ่อยใน dp เพราะมันเป็นลักษณะแบบความสัมพันธ์เวียนเกิดเลย แล้วเราก็แค่ใส่ตัวจำเข้าไป ซึ่งวิธีเขียนแบบนี้เรียกว่าการเขียนแบบ top down

2. จากจุดนี้ไปที่ไหนได้

   เราจะใส่ทุกอันที่เป็น base case ก่อน แล้วค่อยๆ เติมกรณีที่เติมได้ไปเรื่อยๆ อาจจะลำบากในการหาลำดับในการใส่ตาราง แต่วิธีนี้ก็ง่ายสำหรับคนที่ไม่ชอบ recursive ซึ่งวิธีเขียนแบบนี้เรียกว่าการเขียนแบบ bottom up

ตัวอย่าง fibo แบบ bottom up

```
F[0]=0;
F[1]=1;
for(int i=2;i<=n;i++)
    F[i]=F[i-1]+F[i-2];
```

Top down / Bottom up

- อธิบายง่ายๆ เปรียบเทียบกับการเรียน
- Top down คือ เราคิดว่าจะเป็นอะไร ก็ค่อยๆ ไล่ลงมาว่าเราต้องเรียนอะไรบ้าง
- Bottom up คือ เราเรียนไปก่อน เป็นอะไรค่อยว่ากัน แต่ก็จะเห็นว่าเราจะเป็นอะไรก็ได้

ข้อดี Top down

- ง่าย ไม่พลิกแพลงมาก กล่าวคือถ้าเรามี recurrence อยู่เราก็เขียนตามนั้นเลย แล้วเพิ่มตัวจำเข้าไป
- เข้าทำปัญหาย่อย เฉพาะที่เป็นประเด็น อะไรที่ไม่เกี่ยวข้องจะไม่เข้าไปทำเลย

ข้อดี Bottom up

- เร็วกว่า top down เพราะใช้ loop ตรงๆ ไม่ต้องไปทำ recursive
- สามารถใช้กลยุทธ์ประหยัด memory ได้
- ยกตัวอย่างเช่น หา fibo ตัวที่ 10M (mod 100M+1) โดยมี memory 512KB
- จะเห็นว่าเราไม่สามารถใช้ top down ได้เนื่องจากต้องประกาศอาเรย์ 10M ช่อง
- แต่ถ้าเราใช้ Bottom up เราสามารถ**ใช้ตัวแปร 3 ตัว**ในการหาคำตอบได้ [ลองคิดดู]

Top down / Bottom up

- ! เวลาทำ dynamic นั้นเลือกเขียนตัวไหนก็ได้ โดยแต่ละข้อมักเขียนได้ทั้งสองวิธี แต่ว่าความยาก - ง่ายในการเขียนอาจต่างกัน เลือกให้เหมาะสมกับการใช้
- บางทีก็ต้องเก็บวิธีการได้มาซึ่งคำตอบด้วย ซึ่งก็ต้องหาวิธีเก็บ

สรุป dynamic programming

- มักจะถามเกี่ยวกับ ค่าที่ดีที่สุด (มากสุด/น้อยสุด)/จำนวนวิธี/ค่าความคาดหวัง(expected value)
- ค่าที่เก็บในอาเรย์ มักเป็นสิ่งที่โจทย์ถาม
- มิติต่างๆ ของอาเรย์ที่เก็บข้อมูล มักขึ้นกับเงื่อนไขโจทย์
- Transition ของแต่ละสเตจก็ขึ้นกับเงื่อนไขโจทย์เช่นกัน

อยากโหด dynamic ทำอย่างไร

- ทำโจทย์
- ทำโจทย์
- ทำโจทย์
- ทำโจทย์
- ทำโจทย์
- ทำโจทย์
- ทำโจทย์

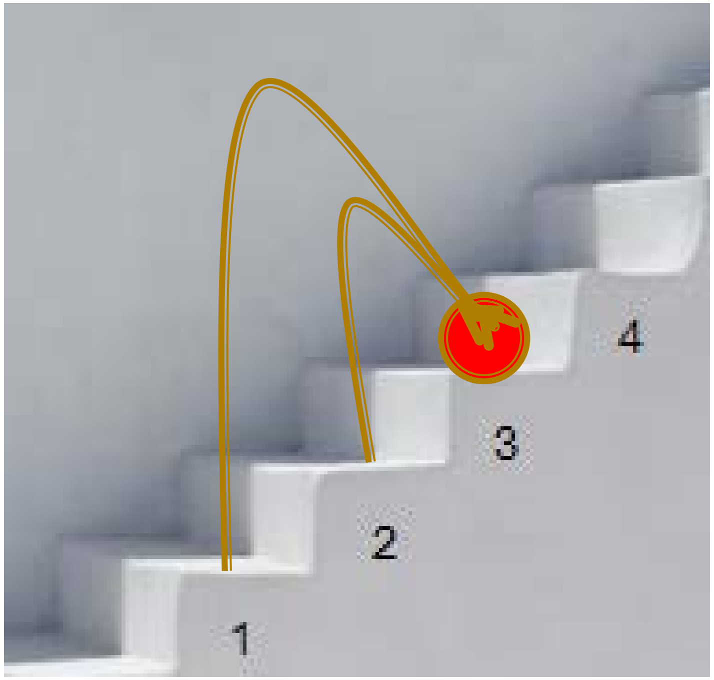
Skgrader: stair1

- เริ่มต้นที่พื้นดิน (บันไดขั้นที่ 0) จะก้าวไปบันไดขั้นที่ n ได้กี่วิธี หากหนึ่งก้าวเราก้าวขึ้นบันไดได้ 1 หรือ 2 ขั้น

Analysis: stair1

- State: จำนวนวิธีในการมาบันไดขั้นบันไดที่ x
- Transition:
- ถ้าจะมายันขั้นที่ x เรามาจากไหนได้บ้าง
- ก็ขั้น x-1 หรือ x-2
- นั่นคือจะได้ dp[x]=dp[x-1]+dp[x-2]
- Base case:
- dp[0]=1 //0->0 ก็มีหนึ่งวิธี คือไม่ขยับ
- dp[1]=1 //0->1 ได้แบบเดียว
- เขียนรวมได้ว่า $dp(x) = \begin{cases} dp(x-1)+dp(x-2) & x \geq 2 \\ 1 & x = 1 \; or \; 0 \\ 0 & otherwise \end{cases}$
- คำตอบคือ dp(n)

Analysis: stair1



Skgrader: stair3.5

- เริ่มต้นที่พื้นดิน (บันไดขั้นที่ 0) จะก้าวไปบันไดขั้นที่ n ได้กี่วิธี หากหนึ่งก้าวเราก้าวขึ้นบันไดได้ 1 หรือ 2 ขั้น
- และเรายังสามารถถอยลงมาหนึ่งขั้นได้ครั้งหนึ่งด้วย
- ! ถ้าถึงขั้นที่ n แล้ว แต่ยังไม่เคยถอย ก็ถอยมาได้
- ! อย่าลืมว่าบันไดมี n ขั้น

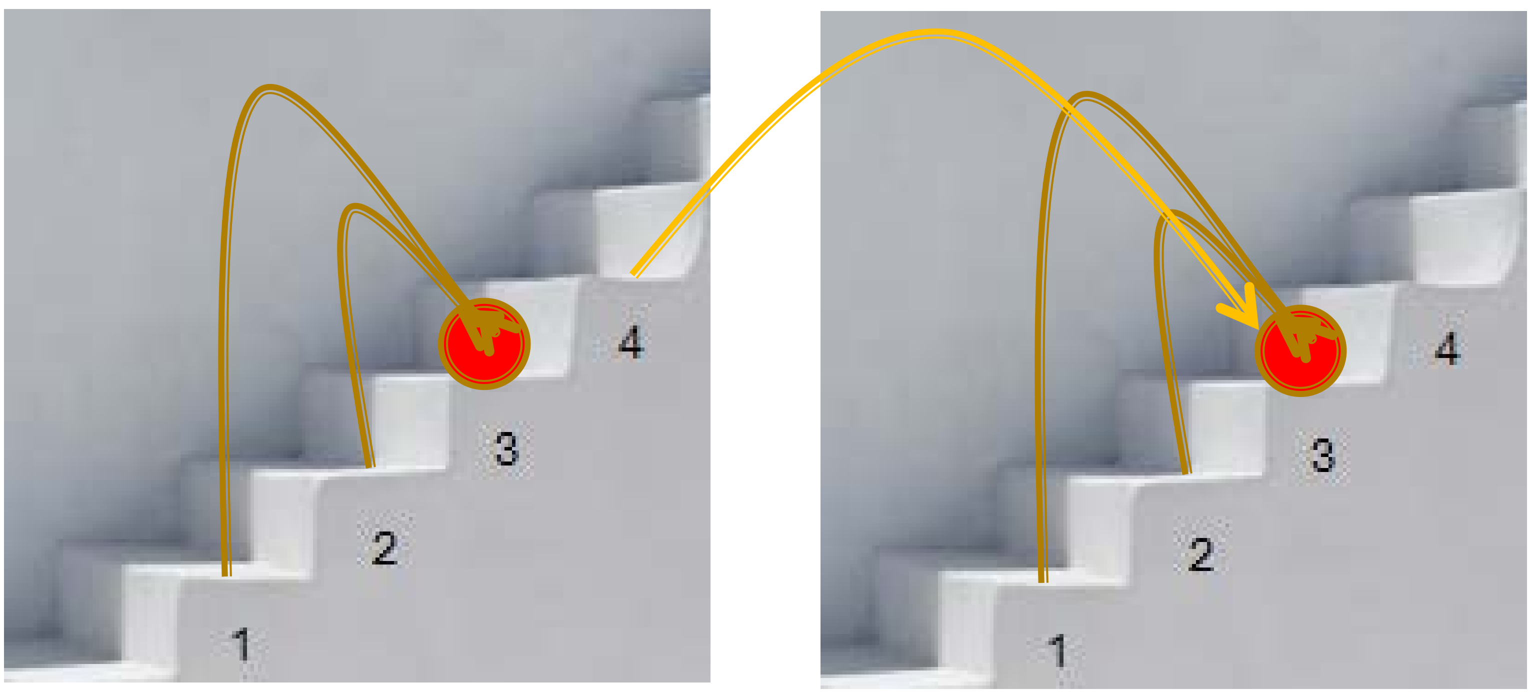
Analysis: stair3.5

- พิจารณาดูแล้วก็ไม่ได้ต่างจากข้อที่แล้วเท่าไหร่ แต่มันมีการถอยได้หนึ่งครั้ง
- State เราก็มีอีกมิติหนึ่ง เก็บว่าเคยถอยยัง
- เขียนได้ดังนี้
- ยังไม่ถอยอันนี้เหมือนเดิม $dp(x) = \begin{cases} dp(x-1) + dp(x-2) & x \geq 2 \\ 1 & x = 1 \\ 1 & x = 0 \end{cases}$
- ถอยแล้ว $dp2(x) = \begin{cases} dp2(x-1) + dp2(x-2) + dp(x+1) & x \geq 2 \\ dp2(x-1) + dp(2) & x = 1 \\ dp(1) & x = 0 \end{cases}$
- คำตอบคือ dp(n)+dp2(n)
- !Otherwise เป็น 0 [ไม่ได้เขียนไว้ในสมการ]

Analysis: stair3.5

ยังไม่เคยถอย

เคยถอยแล้ว



Skgrader: ant_walk

- เดินบนกริด จาก `(0,0)` ไป `(N,M)` ได้กี่วิธี
- เราเดินได้เฉพาะ ทางขวา กับขึ้นบน เท่านั้น

Analysis: ant_walk

- `State` ก็คือ เราอยู่ที่ไหน `(i,j)`
- `State` เก็บจำนวนวิธีที่มายัง `(i,j)`
- สังเกตว่าการมายัง `(i,j)` เรามาได้จาก 2 ทาง คือมาจากทางด้านล่าง กับทางด้านซ้าย (สังเกตว่ามองกลับกับโจทย์)
- เขียนสถานะได้ว่า
- $dp(i,j) =$

$$\begin{cases} dp(i-1,j) + dp(i,j-1) & i > 1 \; and \; j > 1 \\ 1 & i = 1 \; or \; j = 1 \\ 0 & otherwise \end{cases}$$

- คำตอบคือ `dp(N,M)`

Skgrader: ban_word

- คำในภาษาหนึ่ง มีแต่อักขระ a b และ c
- แต่คำในภาษานี้ จะไม่มี ab ปรากฏอยู่ในคำใดๆ เลย
- อยากทราบว่า คำในภาษานี้ ที่ยาว N(<=44)อักขระ มีกี่คำ

Analysis: ban_word

- โจทย์ถามว่า กี่คำ <-   ค่าที่เก็บในอาเรย์
- สังเกตว่าเราเกี่ยวพันกับ ความยาวของสตริง และมีเงื่อนไขอะไรบางอย่างบนตัวอักษร
- ต้องเก็บตัวท้ายด้วย เราจะได้รู้ว่าไปไหนได้
- `State` เราจึงมีความยาวของสตริงกับ ตัวอักษรท้ายเป็นอะไร
- นิยามละเอียด ก็คือ จำนวนรูปแบบสตริงยาว `n` ที่ลงท้ายด้วย `ch`
- ถ้าเราลงท้ายด้วย `b` ตัวก่อนหน้าเป็น `a` ไม่ได้ นอกนั้นก็ไม่มีปัญหาอะไร
- เขียนสถานะได้ว่า

$$dp(n, ch) = \begin{cases} dp(n-1,a) + dp(n-1,b) + dp(n-1,c) & ch = a \text{ } or \text{ } ch = c \\ dp(n-1,b) + dp(n-1,c) & ch = b \\ 1 & n = 0 \end{cases}$$

คำตอบคือ `dp(n,'a')+dp(n,'b')+dp(n,'c')`

เพราะความยาว `n` ลงท้ายด้วยอะไรก็ได้ เราต้องนับทุกกรณี

SPOJ: 12471 – DIE HARD

- เริ่มต้น อัศวินเรามีเลือด H เกราะ A (1<=H,A<=1000)
- ถ้าเราอยู่บนฟ้า 1 ติ๊ก เลือด +3 เกราะ +2
- ถ้าเราอยู่ในน้ำ 1 ติ๊ก เลือด -5 เกราะ -10
- ถ้าเราอยู่ในไฟ 1 ติ๊ก เลือด -20 เกราะ -5
- เราสามารถอยู่แดนใดๆ ได้คราวละ 1 ติ๊กเท่านั้น
- เริ่มต้นเราอยู่ที่ดินแดนไหนก็ได้ (เวลาที่ 0)
- ถ้า เลือด หรือ เกราะ มีค่า <=0 จะตาย
- อัศวินเราจะมีชีวิตอยู่ถึงติ๊กที่เท่าใด

Analysis: 12471 – DIE HARD

- ให้ state คือ (H,A)
- เห็นว่าเส้นทางการเดินไม่มีทางมาวนซ้ำใหม่ เพราะจำนวนที่ลดมันมากกว่าที่เพิ่ม นั่นคือสอดคล้องกับสบบัติข้อแรก
- แต่สเตจมีมิติแค่นี้ไม่พอ เพราะเราต้องไม่อยู่แดนซ้ำเดิมด้วย นั่นคือสเตจจะมี 3 มิติ คือ (H,A,area)
- นิยามสเตจ เวลาที่อยู่รอดนานสุด ที่มีเลือก H เกราะ A และอยู่แดน area
- อย่าลืม วิธีแบบ top down (เขียน recurrence) ต้องมองว่า ณ จุดนี้มาจากไหน

Analysis: 12471 – DIE HARD

- ดังนั้น เขียนสมการได้ดังนี้
- $dp(H, A, area)$

$$= \begin{cases} \max\big(dp(H+20, A+5, air), dp(H+20, A+5, water)\big)+1 & area = fire \\ \max\big(dp(H+5, A+10, air), dp(H+5, A+10, fire)\big)+1 & area = water \\ \max\big(dp(H-3, A-2, fire), dp(H-3, A-2, water)\big)+1 & area = air \\ 0 & H \le 0 \ or\ A \le 0 \end{cases}$$

คำตอบคือ max(`dp(H,A,fire), dp(H,A,water), dp(H,A,air)`)

UVa: 10910 – Marks Distribution

- มี N วิชา สอบได้คะแนนรวม T คะแนน และทุกวิชาได้อย่างน้อย P คะแนน มีกี่แบบ
- 1<=N,T,P<=70

Analysis:
10910 – Marks Distribution

- เห็นว่ามีจำนวนวิชา คะแนนรวม และคะแนนที่ได้ ณ วิชานั้นๆ
- `State -> (N,T,s)` นิยามคือ วิชาที่ N มีคะแนน s และมีคะแนนรวมทั้งหมดของวิชา 1..N คือ T
- เวลาไล่ลงไป ก็ให้ดูว่าคะแนนที่ได้ต้องมีอย่างน้อย P
- เขียนสถานะได้ว่า
- $dp(N,T,s) = \begin{cases} \sum_{i=P}^{T-s} dp(N-1, T-s, i) & N > 1 \\ 1 & N = 1 \end{cases}$
- คำตอบคือ $\sum_{i=p}^{T} dp(N,T,i)$

Analysis: 10910 – Marks Distribution (cont.)

- มองต่ออีกนิด จะเห็นว่า เราต้องลูปไล่คะแนนวิชาสุดท้าย
- ถ้าเราเปลี่ยนเป็นมี `N+1` วิชา และวิชาที่ `N+1` มีคะแนนเป็น `0` ก็น่าจะดี เพราะเราไม่ต้องไปลูปไล่เองแบบตะกี้
- คำตอบคือ $dp(N+1, T, 0)$

Analysis:
10910 – Marks Distribution (cont.)

- ยังมีวิธีอื่นอีกในการแก้ข้อนี้
- มองเป็นคณิตศาสตร์ เราจะได้ว่า ก็แจกคะแนนให้ทุกวิชาไปเท่ากับ P ก่อน
- ที่เหลือก็ใช้ stars and bars โดยจำนวนวิธีคือ $\binom{T-NP+N}{N-1}$
- ปัญหาจะกลายเป็นไปหาค่า nCr แทน
- สามารถหาได้โดยใช้ สามเหลี่ยมปาสคาล ซึ่งเขียนความสัมพันธ์ได้ว่า
- $C(n,r) =$

$$\begin{cases} C(n-1,r-1)+C(n-1,r) & n>1\ and\ r>1 \\ 1 & r=0 \\ 0 & otherwise \end{cases}$$

Analysis: 10910 – Marks Distribution (cont.)

- Complexity วิธีแรก คือ $O(NTP) = O(N^3)$
- แต่ Complexity วิธีหลัง คือ $O(N^2)$
- หลักการคิดที่ต่างกัน ก็ใช้เวลาต่างกัน
- รวมถึงตั้งสเตจต่างกัน ก็ใช้เวลาต่างกัน

UVa: 10036 – Divisibility

- มีเลข N ตัว (1<=N<=10000) เรียงจากซ้ายไปขวา
- ต้องการแทรกเครื่องหมาย +/- ลงไป**ระหว่างตัวเลข** (แทรกที่ละตัว) ให้ผลลัพธ์ที่ได้หารด้วย K ลงตัว (2<=K<=100)
- สามารถทำได้หรือไม่

Analysis: 10036 – Divisibility

- โจทย์ถามว่าได้หรือไม่ state เก็บแค่ yes/no
- ค่าที่เกี่ยวข้องคือ ตัวที่เท่าไหร่ กับผลลัพธ์ที่ mod แล้ว ได้เศษอะไร
- State: (N,M) คือ พิจารณาถึงตัวที่ N มีเศษเป็น M
- เขียนสถานะได้ว่า

$$dp(N,M) = \begin{cases} dp(N-1,(M+a_N)\%K) \text{ or } dp(N-1,(M-a_N)\%K) & N>1 \\ 1 & N=1 \text{ and } M=a_1\%K \\ 0 & \text{otherwise} \end{cases}$$

- คำตอบคือ dp(N,0)
- เวลาเอาไปเขียน C ระวังด้วยว่า –กับ+ mod กันได้ -

Analysis: 10036 – Divisibility (cont.)

- ตัวอย่างโค้ดส่วนที่คำนวณ

```
bool dp(int n,int m)
{
    if(tb[n][m]>-1) return tb[n][m];
    if(n==1)
    {
        if(((a[1]%K+K)%K)==m)   return 1;
        else    return 0;
    }
    tb[n][m]=
max(dp(n-1,((m+a[n])%K+K)%K),dp(n-1,((m-a[n])%K+K)%K));
    return tb[n][m];
}
```

Analysis: 10036 – Divisibility (cont.)

- มองแบบ `bottom up`
- ก็เริ่มต้นจาก `base case`
- นั่นคือ `dp[1][(a_1%K)]=1`
- แล้วไปดูที่ชั้น`n+1` โดยเราไล่ที่ชั้น `n` โดยหาว่าอันไหนเป็น `1` เราก็จะไปใส่ค่า `1` ที่ช่อง `dp[n+1][(x+a_(n+1))%K]` กับ `dp[n+1][(x-a_(n+1))%K]`
- ทำจนหมดทั้งตาราง แล้วดูว่าช่อง `dp[n][0]` เป็น `1` หรือไม่ ถ้าเป็นก็แสดงว่าหารได้ลงตัว

Analysis: 10036 – Divisibility (cont.)

- ทั้งสองแบบนี้ ก็ใช้เวลาเป็น `O(NK)` ทั้งคู่
- เพียงแต่ว่า `top down` จะดูเฉพาะที่เราอยากจะดู
- แต่ `bottom up` จะเสียเวลาเป็น `NxK` แน่ๆ เพราะมันไล่ทั้งกระดานจริงๆ เลย
- แต่ `bottom up` ก็ดูจะเข้าใจง่ายกว่า เพราะมันก็ค่อยๆ ต่อท้ายมาเลย

UVa: 11407 – Squares

- เขียน N(1<=N<=10000) ในรูปของผลรวมของกำลังสอง
- กล่าวคือ N=a1^2+a2^2+…+an^2 เมื่อ ai เป็นจำนวนเต็ม
- ให้หา n น้อยสุด ที่สามารถทำได้
- เช่น 4=1^2+1^2+1^2+1^2=2^2 ดังนั้น n น้อยสุดที่ทำได้ในกรณี N=4 คือ 1

Analysis: 11407 – Squares

- โจทย์ถามจำนวนตัวที่น้อยที่สุดที่ใช้
- สิ่งที่เกี่ยวข้องก็คือ ผลรวมของเลขที่เราเลือก (ต้องเป็นมิติหนึ่งแน่ๆ)
- แล้วเราจะเลือกเลขอย่างไรดี
- เราเลือกสะเปะสะปะ มันก็ไม่รู้ว่านิยามสเตจอย่างไร
- เลยคิดว่า เลือก 1 ให้พอใจก่อนแล้วก็ค่อยเลือก 2,3,4,…
- แล้วต้องเลือกถึงแค่ไหน
- เพราะว่ามิติหนึ่งเป็นผลรวม (N) ก็มีค่ามากสุดคือ 10000 ถ้าเราเลือกเลขถึง 10000 ตารางก็จะมี 10K x 10K = 100M ซึ่งประกาศไม่ได้
- ข่าวดีคือ เราเลือกถึง 100 ก็พอ เพราะว่า 100^2 = 10000 ถ้าเลือก 101 ก็เกิน 10000 แล้ว
- ดังนั้นตารางมี 10000x100=1M ช่อง พอไหว

Analysis: 11407 – Squares (cont.)

- นิยามสเตจ dp(N,M) คือ จำนวนตัวที่ต้องใช้ เมื่อผลรวมเป็น N หากพิจารณาเลข 1…M
- ในแต่ละสเตจนั้น เราจะเลือกว่า จะเอาค่า M^2 นี้ใส่เข้าไปหรือไม่
- เราจะเลือกว่า ระหว่างเอาตัวเก่ามาเลย กับ ใส่ตัวนี้เข้าไป แบบไหนดีกว่า
- อย่าลืมว่า เราสามารถใส่ตัวเดิมเข้าไปได้อีกด้วย
- เขียนสถานะได้ว่า

$$dp(N,M) = \begin{cases} \min\begin{cases} \min(dp(N-M^2, M-1), dp(N-M^2, M)) + 1) \\ dp(N, M-1) \end{cases} & M > 1 \\ 1 & N = 1\ and\ M = 1 \\ \infty & otherwise \end{cases}$$

คำตอบคือ $\min_{i=1 \to \sqrt{N}} dp(N,i)$

Analysis: 11407 – Squares (cont.)

- คิดต่ออีกหน่อย จะเห็นว่า $dp(N - M^2, M - 1)$ นั้นถูกรวมเข้ากับ $dp(N - M^2, M)$ แล้ว [มองออกมั้ย]
- เพราะ $dp(N - M^2, M) = \min(\Delta, dp(N - M^2, M - 1))$
- ถ้า $dp(N - M^2, M - 1)$ ดีกว่า ค่า $dp(N - M^2, M)$ ก็จะเก็บค่า $dp(N - M^2, M - 1)$ อยู่แล้ว ไม่ต้องดูอีก
- แต่ถ้าค่า Δ ดีกว่า ค่า $dp(N - M^2, M)$ ก็จะเก็บค่า Δ นั่นคือมันไม่สน $dp(N - M^2, M - 1)$
- สรุปแล้วก็ละ $dp(N - M^2, M - 1)$ ออกไปในเงื่อนไขได้เลย

Analysis: 11407 – Squares (cont.)

- จึงเขียนสถานะได้ว่า

$$dp(N,M) = \begin{cases} \min(dp(N-M^2,M)+1, dp(N,M-1)) & M>1 \\ 1 & N=1 \text{ } and \text{ } M=1 \\ \infty & otherwise \end{cases}$$

คำตอบคือ $\min_{i=1\rightarrow\sqrt{N}} dp(N,i)$

Analysis: 11407 – Squares (cont.)

- คิดต่ออีกนิด เราจะทำเป็นแบบ bottom up จะได้ว่า อาเรย์มีขนาด 10000 x 100
- แต่ถ้าดูดีๆ มันใช้อาเรย์ที่เล็กกว่านี้ได้ คือใช้แค่ 10000 x 2
- เนื่องจากความสัมพันธ์นั้นสนใจแค่แถวที่ติดกันเท่านั้น $dp(N - M^2, M) = \min(dp(N - M^2, M) + 1, dp(N, M - 1))$
- จึงสามารถสลับแถวกันใช้ได้

Analysis: 11407 – Squares (cont.)

- คิดต่ออีกซักเล็กน้อย เราลดเหลืออาเรย์ 10000 ช่องได้
- ทำไมถึงได้
- เพราะ ค่าที่มันเลือกมี สองอย่าง คือ (1) ค่าในแถวเดียวกันตัวที่ถอยไป M^2 หรือ (2) เอาช่องนี้แต่อยู่อีกแถว
- นั่นคือ ก็มีแถวเดียวก็เก็บค่า (2) ได้แล้ว
- แล้วที่ x ก็ไปดูช่อง $x - M^2$ ว่าดีกว่ามั้ย ดีกว่าก็ใส่แทนลงไป

Analysis: 11407 – Squares (cont.)

- ตัวอย่างการเขียน
- Set ค่าเริ่มต้นคือ dp[0]=0 ที่เหลือเป็น inf

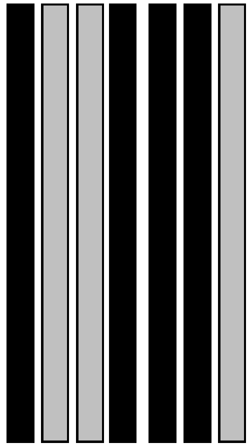
```
for(int j=1;j<=M;j++) // หาค่าที่ใส่
  for(int i=0;i<=N-j*j;i++) // ผลรวม
    dp[i+j*j]=min(dp[i+j*j],dp[i]+1);
```

Need to know

- การเขียนสเตจ ของข้อ UVa: 11407 – Squares ระหว่าง (ผลรวม,ตัว) กับ (ตัว ,ผลรวม) มีสมรรถนะ**ต่างกัน**
- ในทางทฤษฎี ไม่ว่าจะเขียนสเตจแบบไหน จะได้ว่ามันมีประสิทธิภาพ O(NK) เมื่อ N คือค่าที่เข้ามา และ K คือ จำนวนตัว
- แต่การวิ่งค่าในอาเรย์ของสองวิธีนี้ มีลักษณะต่างกัน
- (ผลรวม,ตัว) วิ่งอาเรย์ที่ตัวหน้า ส่วน (ตัว,ผลรวม) จะวิ่งอาเรย์ที่ตัวหลัง
- แบบแรกเรียกว่า column major คือวิ่งข้อมูลที่หลัก แบบหลังเรียก row major คือวิ่งข้อมูลที่แถว
- Memory มันเป็น แถบๆ เดียว การประกาศอาเรย์สองมิติ มันจะจองแถวหนึ่งต่อด้วยแถวสอง แถวสาม ไปเรื่อยๆ
- ดังนั้น การเข้าถึงข้อมูลแบบ row major จึงเร็วกว่า ซึ่งจะเห็นได้ชัดใน bottom up
- ในกรณีของ top down มันเข้าถึงข้อมูลไม่มีความต่อเนื่องอยู่แล้ว ดังนั้นจึงมีผลไม่มาก

UVa: 10721 – Bar Codes

- รหัสแท่ง (bar-code) ยาว N มี K แถบ แต่ละแถบยาวได้อย่างมาก M มีกี่แบบ (1<=N,M,K<=50) แถบซ้ายสุดเป็นสีดำเสมอ มีทั้งหมดกี่แบบ
- อธิบายอีกลักษณะหนึ่งคือ เลข 0,1 ยาว N ตัว กลุ่มเลขติดกันได้ยาวไม่เกิน M ตัว รวมแล้วมี K กลุ่ม หลักแรกสุดเป็น 1มีกี่แบบ
- เช่น N=7,K=4,M=3 ตอบ 16



Analysis: 10721 – Bar Codes (cont.)

- โจทย์ถามว่า กี่วิธี นั่นคือสเตจเก็บจำนวนวิธี
- เราสนใจที่ความยาวต้องได้ จำนวนแถบต้องได้ แล้วห้ามติดกันเกินเงื่อนไข ก็ให้เหล่านี้คือสเตจไป
- `State:(n,k,m)=`จำนวนรูปแบบของรหัสแท่งที่ยาว `n` มี `k` แถบ และความยาวแต่ละแถบเป็น `m`
- ทว่าถ้าเก็บเท่านี้ จะรู้ได้อย่างไรว่าถ้าเราเอาแถบดำ หรือแถบขาวมาต่อแล้วจะทำให้แถบที่ติดกันยาวขึ้น หรือเป็นการเปิดกลุ่มใหม่
- จึงเก็บอีกมิติคือ สีของแถบสุดท้ายด้วย `->(n,k,m,c)`

Analysis: 10721 – Bar Codes (cont.)

- การเปลี่ยนสเตจ ก็มีใส่แถบนี้เป็นแถบใหม่ กับต่อแถบเดิม
- ทำให้เขียนความสัมพันธ์ได้ว่า $dp(n,k,m,c) =$

$$\begin{cases} dp(n-1,k,m-1,c) & n>1\ and\ m>1 \\ \sum_{i=1}^{m} dp(n-i,k-1,i,c') & n>1\ and\ m=1 \\ 1 & n=1\ and\ k=1 \\ 0 & otherwise \end{cases}$$

- ถ้า `c=1` แล้ว `c'=0 /` ถ้า `c=0` แล้ว `c'=1`
- คำตอบคือ `sum(dp(N,K,i,c)) i` วิ่งตั้งแต่ `1` ถึง `M`
- มีประสิทธิภาพเป็นเท่าใด ?

Analysis: 10721 – Bar Codes (cont.)

- มีอยู่ NK(M-1) state ที่เข้าเงื่อนไขแรก
- มีอยู่ NK state ที่เข้าเงื่อนไขที่สอง
- เงื่อนไขแรก ทำ 1 คำสั่ง
- ในขณะที่ เงื่อนไขที่สองทำ M คำสั่ง
- ดังนั้นรวมแล้วก็เป็น O(NKM)

Analysis: 10721 – Bar Codes (cont.)

- คิดแบบ bottom up
- ใช้ state เหมือนเดิมเลย คือ (n,k,m,c)
- Set dp[1][1][1][1]=1
- แล้วก็วนลูปไป แบ่งเป็น 2 กรณี คือ เอา 1ต่อท้าย กับ เอา 0ต่อท้าย
- เขียนโค้ดส่วนวนลูปได้ว่า

Analysis: 10721 – Bar Codes (cont.)

```
for(i=1;i<=N;i++)
  for(j=1;j<=K;j++)
    for(k=1;k<=M;k++) {
      tb[i+1][j][k+1][0]+=tb[i][j][k][0];
      tb[i+1][j][k+1][1]+=tb[i][j][k][1];
      tb[i+1][j+1][1][0]+=tb[i][j][k][1];
      tb[i+1][j+1][1][1]+=tb[i][j][k][0];
    }
```

- คำตอบคือ `sum(dp(N,K,i,c))` `i` วิ่งตั้งแต่ 1 ถึง M
- มีประสิทธิภาพเป็น O(NKM)

Analysis: 10721 – Bar Codes (cont.)

- เรายังสามารถนิยามสเตจแบบอื่นๆ ได้อีก
- เช่น (n, k) นิยามคือ จำนวนรูปแบบซึ่งมีรหัสแท่งยาว n และมี k แถบ
- ในแต่ละสเตจ จะคิดตั้งแต่ 1 ถึง M เลย
- เขียนความสัมพันธ์ได้เป็น
- $$dp(n,k) = \begin{cases} \sum_{i=1}^{M} dp(n-i,k-1) & n > 1 \; and \; k \geq 1 \\ 1 & n = 1 \; and \; 1 \leq k \leq M \\ 0 & otherwise \end{cases}$$

คำถาม

Programming Practice

- UVa – 11703 sqrt log sin
- UVa – 10520 Determine it
- UVa – 10943 How do you add?
- UVa – 10446 The Marriage Interview :-)
- UVa – 11420 Chest of Drawers

จบเนื้อหา

สวัสดี