Render

ข้อนี้มีภารกิจ $N \leq 20$ ภารกิจ โดยต้องทำทุกภารกิจเพียงแต่ต้องเลือกลำดับที่จะทำ

หากทำภารกิจที่ j เป็นลำดับที่ i จะมีโอกาสสำเร็จ $a_{(j,i)}$ โจทย์ถามว่าผลคูณความน่าจะเป็นเหล่านี้ที่เป็นไปได้มากสุดคือเท่าไหร่

แนวคิด

ข้อนี้เป็นโจทย์ Bitmask Dynamic Programming นั่นคือเป็นโจทย์ Dynamic Programming ที่เก็บ State เป็น Bitmask

สังเกตว่าเราสามารถเก็บคำตอบของแต่ละ State เป็น $DP[S]$ ซึ่งแทนผลคูณความน่าจะเป็นที่จะสำเร็จโดยที่ภารกิจที่สำเร็จแล้วคือ S เมื่อ State $S=(b_{N}b_{N-1}\dots b_0)_2$ เป็นเลขฐานสองโดยที่ $b_j=1$ ถ้าเราทำภารกิจที่ j แล้ว คำตอบจะเป็น $DP[2^N -1]$ เพราะ $2^N-1 = (11\dots1)_2$ (มี 1 N ตัว)

เช่นถ้า $S=1010_2$ แสดงว่าทำภารกิจที่ 2 กับ 4 แล้ว

สังเกตว่าสำหรับ State S จำนวนภารกิจที่ทำไปแล้วจะเท่ากับจำนวน bit ที่เป็น 1 ให้จำนวนนี้เป็น i_{S}

สำหรับ $DP[0]$ สามารถตั้งเป็น 100 แทนโอกาส 100%

ในการคำนวณ $DP[S]$ สังเกตว่าจะต้องมีงานอันภารกิจ j ที่ $b_j=1$ ใน S ดังนั้นสามารถพิจารณาทีละงาน j ดังกล่าวว่าผลคูณที่ดีที่สุดที่เป็นไปได้คือเท่าไหร่ ซึ่งจะได้ว่าเป็น $a_{(j, i_{S})} DP[S - (1<<j)]$ นั่นคือผลคูณของความน่าจะเป็นเมื่อทำงาน j เป็นลำดับที่ i กับผลคูณความน่าจะเป็นที่มากสุดที่เป็นไปได้สำหรับ $S - (1<<j)$ (ซึ่งเป็น State S ก่อนทำภารกิจที่ j)

ดังนั้นสำหรับแต่ละ S หากมีค่า $DP[0], \dots, DP[S-1]$ แล้วจะใช้เวลาคำนวณเพียง $\mathcal{O}(N)$ เมื่อพิจารณาทีละภารกิจ

ดังนั้นเมื่อต้องพิจารณา 2^N State เวลาทั้งหมดที่ใช้คือ $\mathcal{O}(N2^N)$

ตัวอย่างโค้ด

```cpp
dp[0] = 100.0;
for (int s = 1; s <= ((1 << n) - 1); s++) {
  int i = 0;
  for (int j = 0; j < n; j++)
    i += (((1 << j) & s) != 0);

  dp[s] = 0;
  for (int j = 0; j < n; j++)
    if (((1 << j) & s) != 0)
      dp[s] = max(dp[s], dp[s ^ (1 << j)] * a[i - 1][j] / 100.0);
}
```

ตามคำอธิบายสำหรับแต่ละ S จะนับจำนวนภารกิจที่สำเร็จแล้วใน State S จากนั้นจะไล่ภารกิจที่สำเร็จใน S ว่าควรทำอันไหนเป็นลำดับที่ i

ข้อนี้มีภารกิจ $N \leq 20$ ภารกิจ โดยต้องทำทุกภารกิจเพียงแต่ต้องเลือกลำดับที่จะทำ

หากทำภารกิจที่ j เป็นลำดับที่ i จะมีโอกาสสำเร็จ $a_{(j,i)}$ โจทย์ถามว่าผลคูณความน่าจะเป็นเหล่านี้ที่เป็นไปได้มากสุดคือเท่าไหร่

แนวคิด

ข้อนี้เป็นโจทย์ Bitmask Dynamic Programming นั่นคือเป็นโจทย์ Dynamic Programming ที่เก็บ State เป็น Bitmask

สังเกตว่าเราสามารถเก็บคำตอบของแต่ละ State เป็น $DP[S]$ ซึ่งแทนผลคูณความน่าจะเป็นที่จะสำเร็จโดยที่ภารกิจที่สำเร็จแล้วคือ S เมื่อ State $S = (b_N b_{N-1} \dots b_0)_2$ เป็นเลขฐานสองโดยที่ $b_j = 1$ ถ้าเราทำภารกิจที่ j แล้ว คำตอบจะเป็น $DP[2^N - 1]$ เพราะ $2^N - 1 = (11 \dots 1)_2$ (มี 1 N ตัว)

เช่นถ้า $S = 1010_2$ แสดงว่าทำภารกิจที่ 2 กับ 4 แล้ว

สังเกตว่าสำหรับ State S จำนวนภารกิจที่ทำไปแล้วจะเท่ากับจำนวน bit ที่เป็น 1 ให้จำนวนนี้เป็น i_S

สำหรับ $DP[0]$ สามารถตั้งเป็น 100 แทนโอกาส 100%

ในการคำนวณ $DP[S]$ สังเกตว่าจะต้องมีงานอันภารกิจ j ที่ $b_j = 1$ ใน S ดังนั้นสามารถพิจารณาทีละงาน j ดังกล่าวว่าผลคูณที่ดีที่สุดที่เป็นไปได้คือเท่าไหร่ ซึ่งจะได้ว่าเป็น $a_{(j,i_S)} DP[S - (1 << j)]$ นั่นคือผลคูณของความน่าจะเป็นเมื่อทำงาน j เป็นลำดับที่ i กับผลคูณความน่าจะเป็นที่มากสุดที่เป็นไปได้สำหรับ $S - (1 << j)$ (ซึ่งเป็น State S ก่อนทำภารกิจที่ j)

ดังนั้นสำหรับแต่ละ S หากมีค่า $DP[0], \dots, DP[S-1]$ แล้วจะใช้เวลาคำนวณเพียง $\mathcal{O}(N)$ เมื่อพิจารณาทีละภารกิจ

ดังนั้นเมื่อต้องพิจารณา 2^N State เวลาทั้งหมดที่ใช้คือ $\mathcal{O}(N2^N)$

ตัวอย่างโค้ด

```cpp
dp[0] = 100.0;
for (int s = 1; s <= ((1 << n) - 1); s++) {
  int i = 0;
  for (int j = 0; j < n; j++)
    i += (((1 << j) & s) != 0);

  dp[s] = 0;
  for (int j = 0; j < n; j++)
    if (((1 << j) & s) != 0)
      dp[s] = max(dp[s], dp[s ^ (1 << j)] * a[i - 1][
}
```

ตามคำอธิบายสำหรับแต่ละ S จะนับจำนวนภารกิจที่สำเร็จแล้วใน State S จากนั้นจะไล่ภารกิจที่สำเร็จใน S ว่าควรทำอันไหนเป็นลำดับที่ i

PROGRAMMING.IN.TH

โปรแกรมมิ่งอินทีเอช ศูนย์รวมของโจทย์และเนื้อหาสำหรับ การเขียน
โปรแกรมเพื่อการแข่งขัน และวิทยาการคอมพิวเตอร์

ค้นหาโจทย์