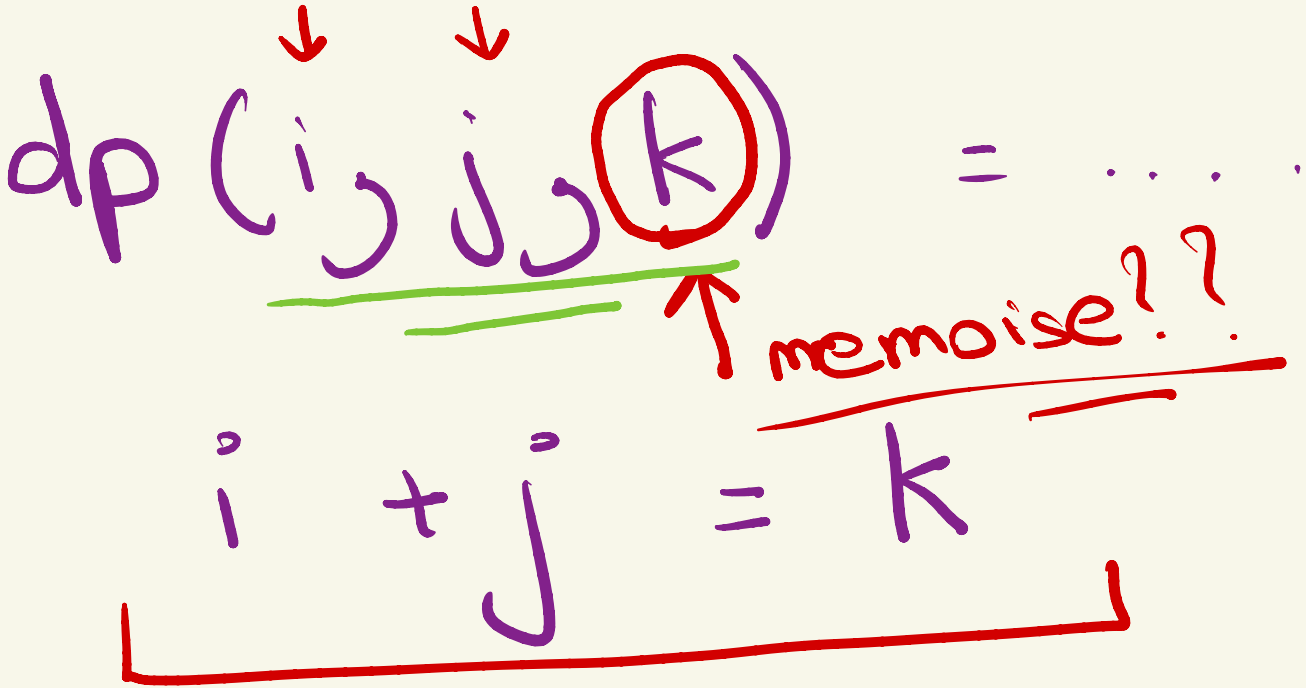
State Elimination & Cyclic DP States

- Gaurish Baliga

State elimination

↳ identify the states

$dp(i, j, t) =$ something



dp(i, j, k) ✗

→ k = i + j ;

O(n³)

↓

O(n²)

↓

dp[i][j] ~~[k]~~

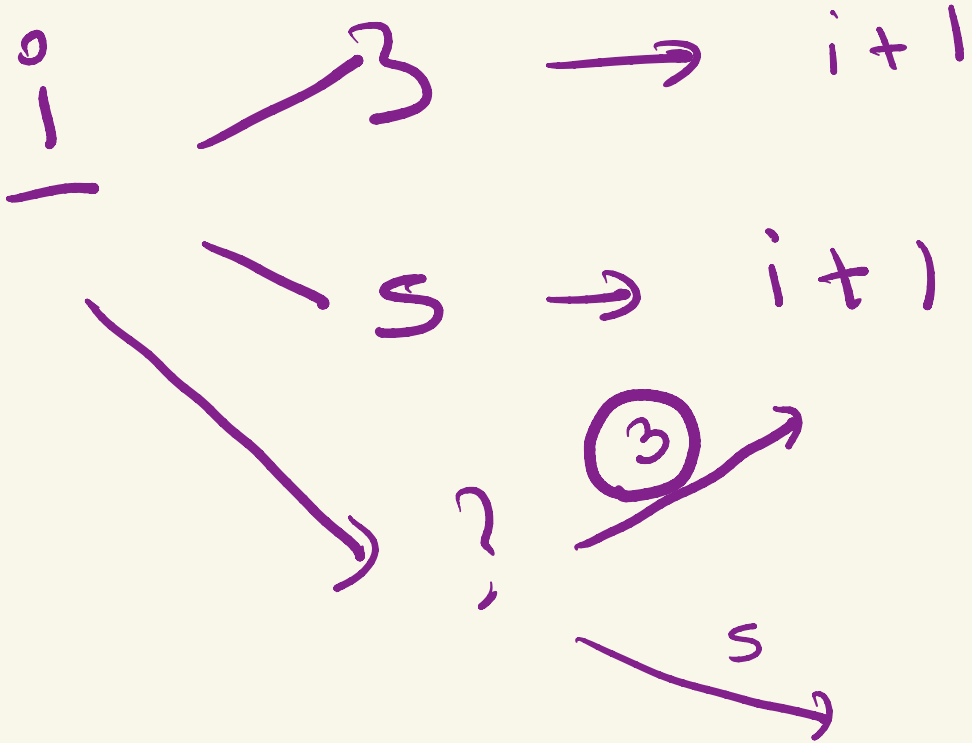
State Elimination

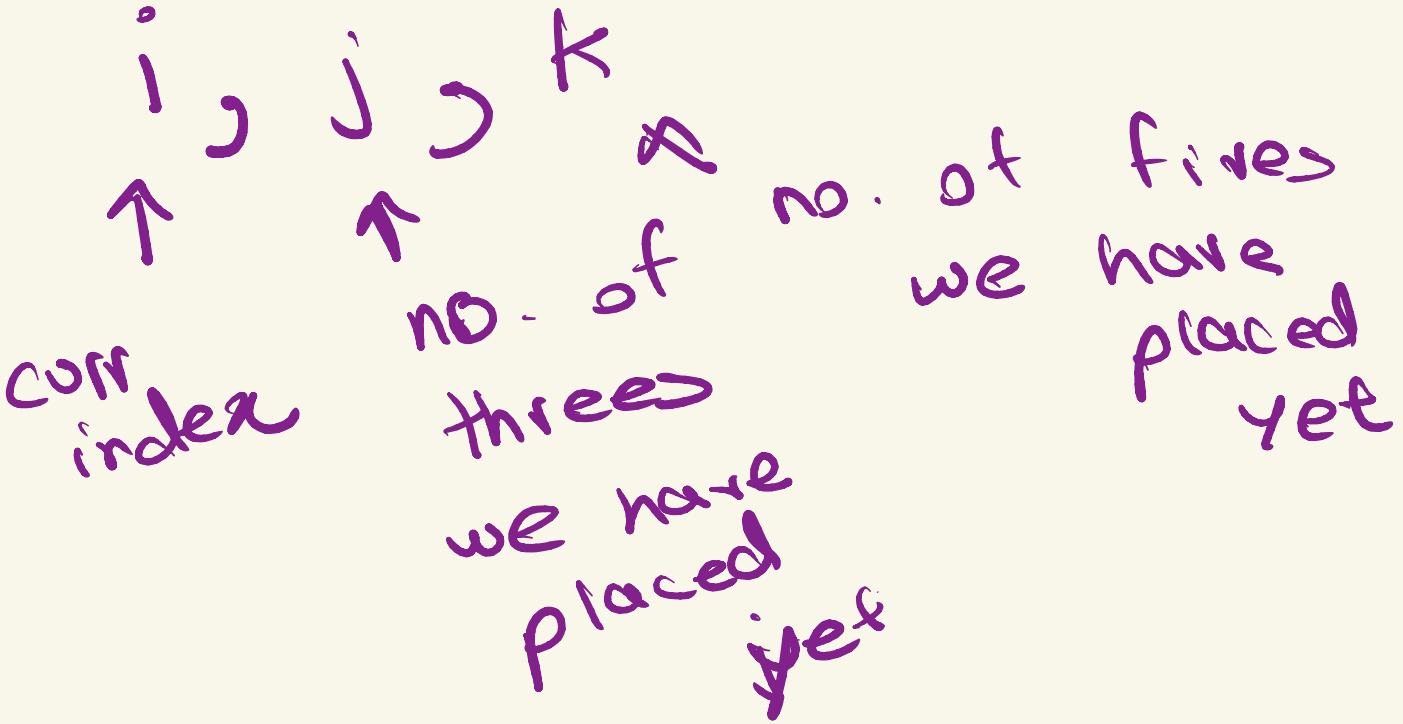
- Ask yourself do you need all the parameters in the dp state?
- If you have dp[a][b][c], and a + b = c, do you need to store c as a parameter or can you just compute it on spot?
- If you can compute a parameter in dp state from other parameters, no need to store it.

What matter are independent states

Problem

Given an array of **N** (input) 3s, 5s and ?s. Evey index with contains a 3 will have a score of X (given in input) and every index that contains a 5 will have a score of Y (given in input). Find the maximum score you can get by replacing each question mark with a 3 or a 5 such that you cannot use more than $\mathbf{k_1}$ (given in input) 3s and $\mathbf{k_2}$ (given in input) 5s in the complete array.

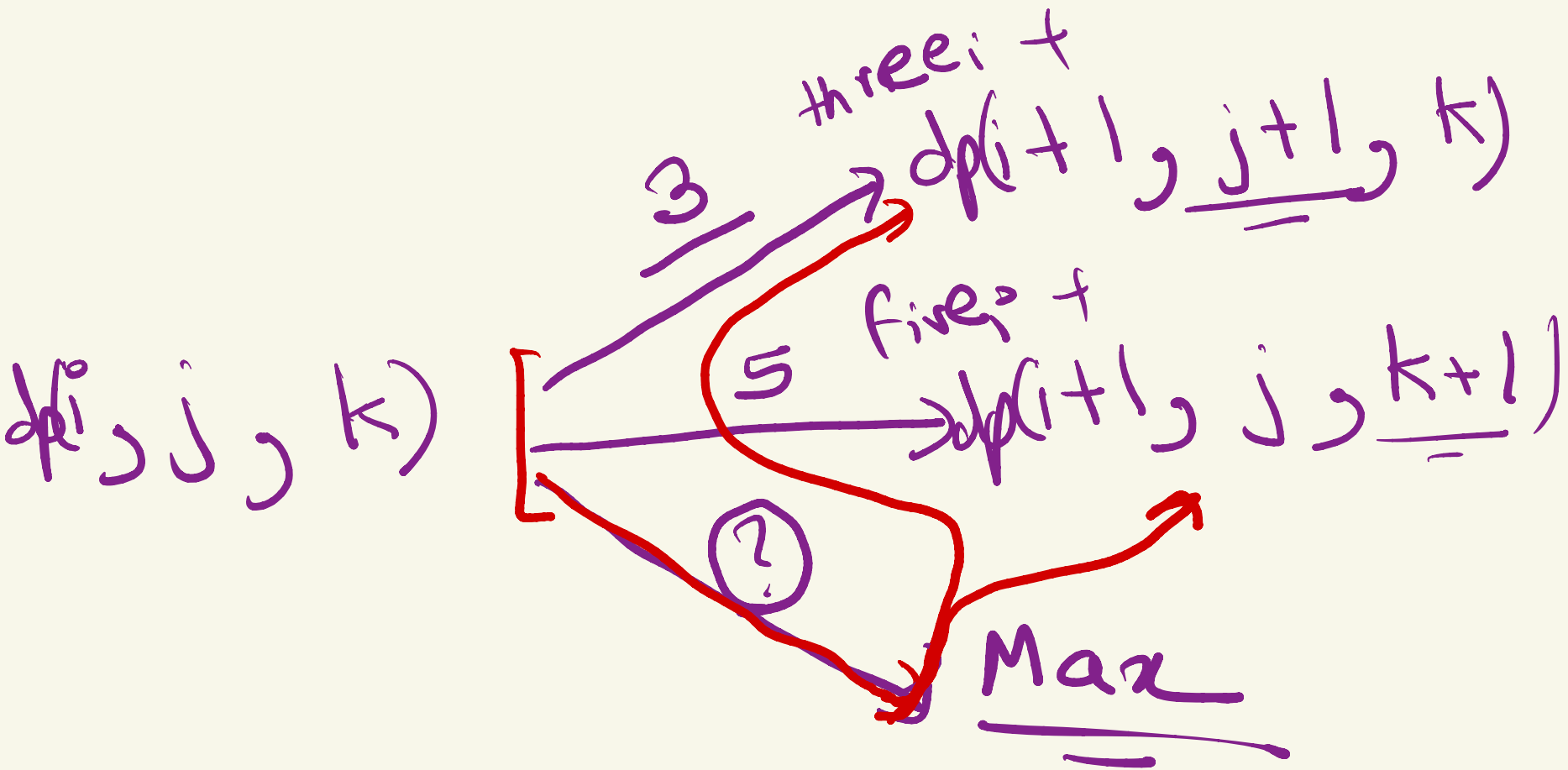


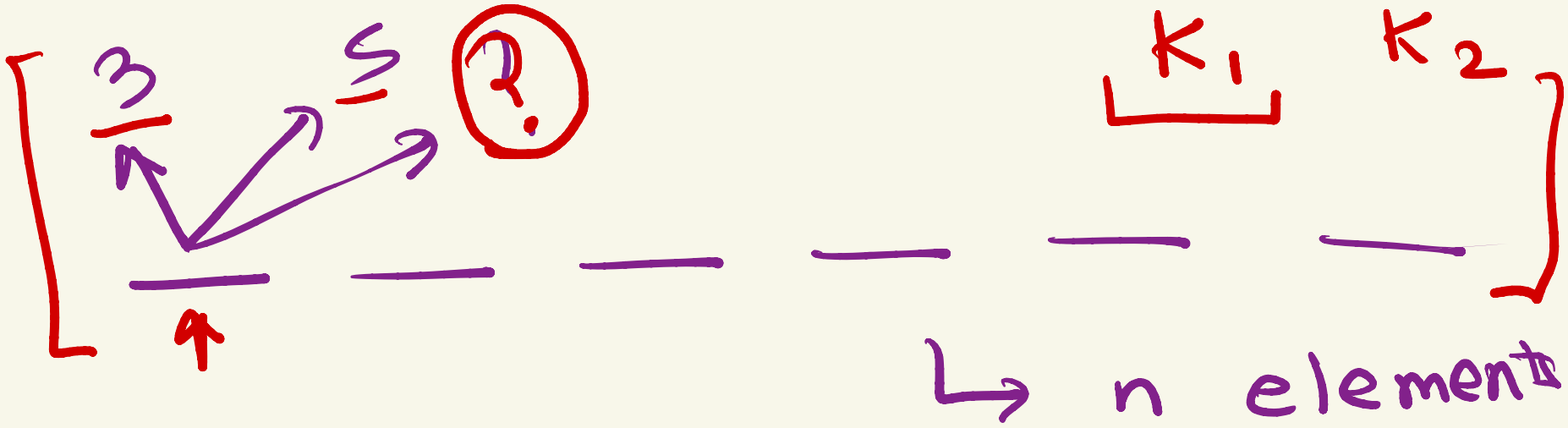


dp(i, j, k) → max score we can get from index (i ... n-1)

j 3s placed

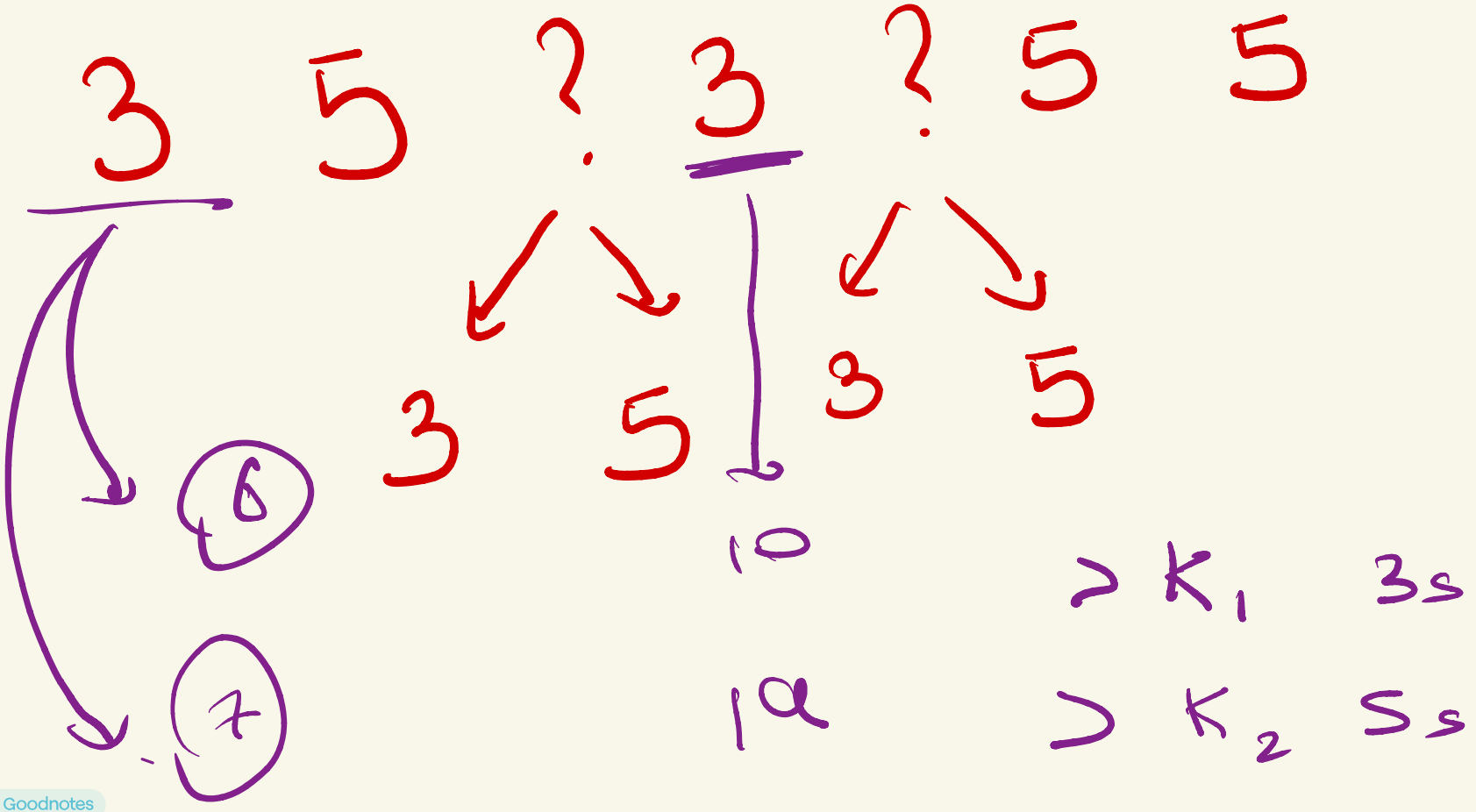
k 5s placed



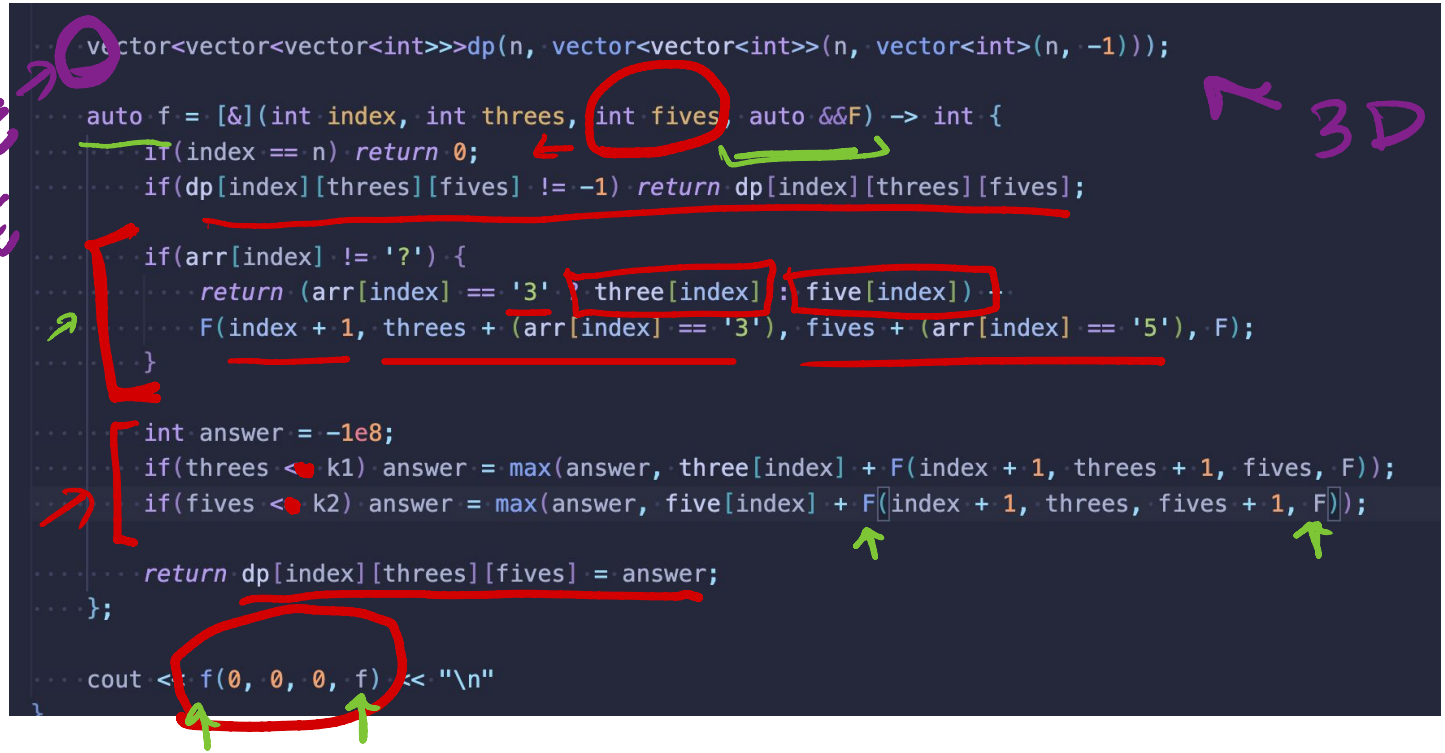


Three

five



Approach 1: Naive Dynamic Programming




```
vector<vector<vector<int>>>dp(n, vector<vector<int>>(n, vector<int>(n, -1)));

auto f = [&](int index, int threes, int fives, auto &&F) -> int {
    if(index == n) return 0;
    if(dp[index][threes][fives] != -1) return dp[index][threes][fives];

    if(arr[index] != '?') {
        return (arr[index] == '3' ? three[index] : five[index]) +
        F(index + 1, threes + (arr[index] == '3'), fives + (arr[index] == '5'), F);
    }

    int answer = -1e8;
    if(threes < k1) answer = max(answer, three[index] + F(index + 1, threes + 1, fives, F));
    if(fives < k2) answer = max(answer, five[index] + F(index + 1, threes, fives + 1, F));

    return dp[index][threes][fives] = answer;
};

cout << f(0, 0, 0, f) << "\n"
}
```

i , j threes

not memoise ??

j + k = i

i - j = k

3 3s

3 - 2 = 1

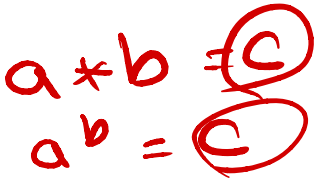
0 1 2

3

Approach 2: Optimised Dynamic Programming

```cpp
vector<vector<int>>dp(n, vector<int>(n, -1));

auto f = [&](int index, int threes, auto &&F) -> int {
    int fives = index - threes;
    if(index == n) return 0;
    if(dp[index][threes] != -1) return dp[index][threes];

    if(arr[index] != '?') {
        return (arr[index] == '3' ? three[index] : five[index]) +
        F(index + 1, threes + (arr[index] == '3'), F);
    }

    int answer = -1e8;
    if(threes < k1) answer = max(answer, three[index] + F(index + 1, threes + 1, F));
    if(fives < k2) answer = max(answer, five[index] + F(index + 1, threes, F));

    return dp[index][threes] = answer;
};

cout << f(0, 0, f) << "\n";
```

Tradeoff in State Elimination?



For a hypothetical DP state, dp[a][b][c] with $a*k_1 + b*k_2 = c*k_3$

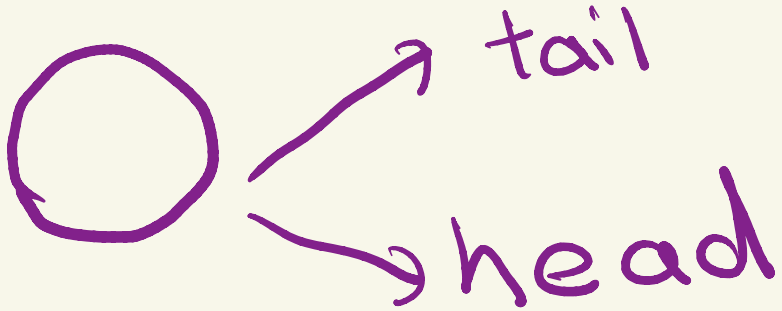
If (1 <= a <= 1000) (1 <= b <= 100) (1 <= c <= 10000), which parameter is best to remove?

* Always remove the largest state

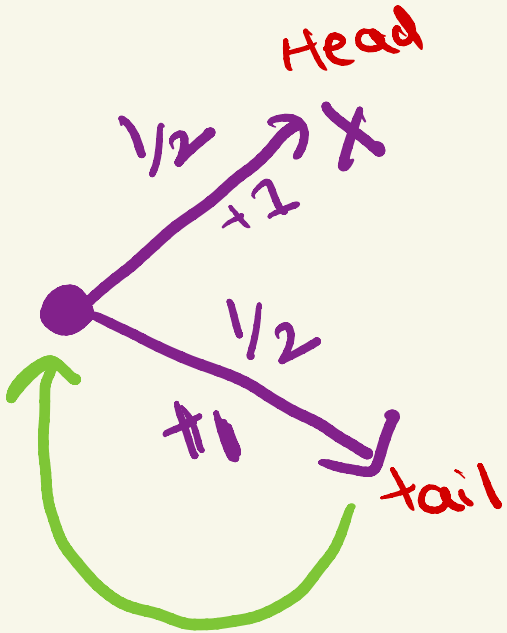
Cycling DP states

Mathematics

- What happens when your current state is dependent on itself?
  - dp[i] depending on dp[i] itself



Exp. number of coin tosses you need to get 1 head



Expectation

$$E(x) = \text{Head}$$

$$\frac{1}{2}\left(1 + \underset{\text{stop}}{0}\right) +$$

$$\frac{1}{2}\left(1 + E(x)\right)$$

$$E(x) = \frac{1}{2}(1+0) + \frac{1}{2}(1+E(x))$$

$$E(x) - \frac{1}{2}E(x) = \frac{1}{2} + \frac{1}{2} = 1$$

$$\frac{1}{2}E(x) = 1$$

$$E(x) = 2$$

Problem

- Given a positive integer N <= 1e6, at every step the following 3 things can happen to N with equal probability.
  - N = N //2 $n = \lfloor n/2 \rfloor$
  - N = N - 1
  - N remains unchanged
- Find expected number of steps it will take to convert for N to become 0

dp(n) → exp. moves required to convert n ⟶ 0

dp[0] = 0

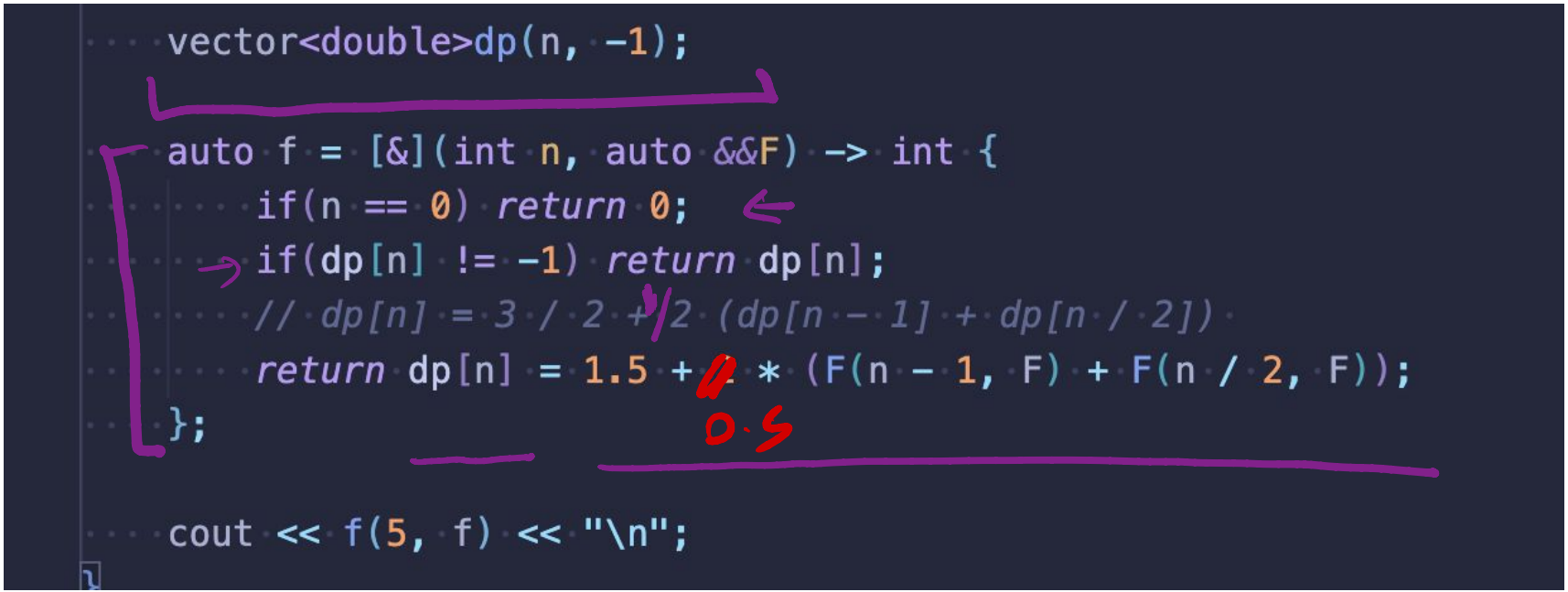
$$dp(n) = \frac{1}{3}(1 + dp(n-1)) +$$
$$\frac{1}{3}(1 + dp(n/2)) +$$
$$\frac{1}{3}(1 + dp(n))$$

$$dp(n) = 1 + \frac{1}{3}\left(dp(n-1) + dp(n-2) + dp(n)\right)$$

$$dp(n) - \frac{1}{3}dp(n) = 1 + \frac{7}{3}\left(dp_{n-1} + dp_{n/2}\right)$$

$$dp(n) = \frac{3}{2} + \frac{21}{2}\left(dp(n-1) + dp(n/2)\right)$$

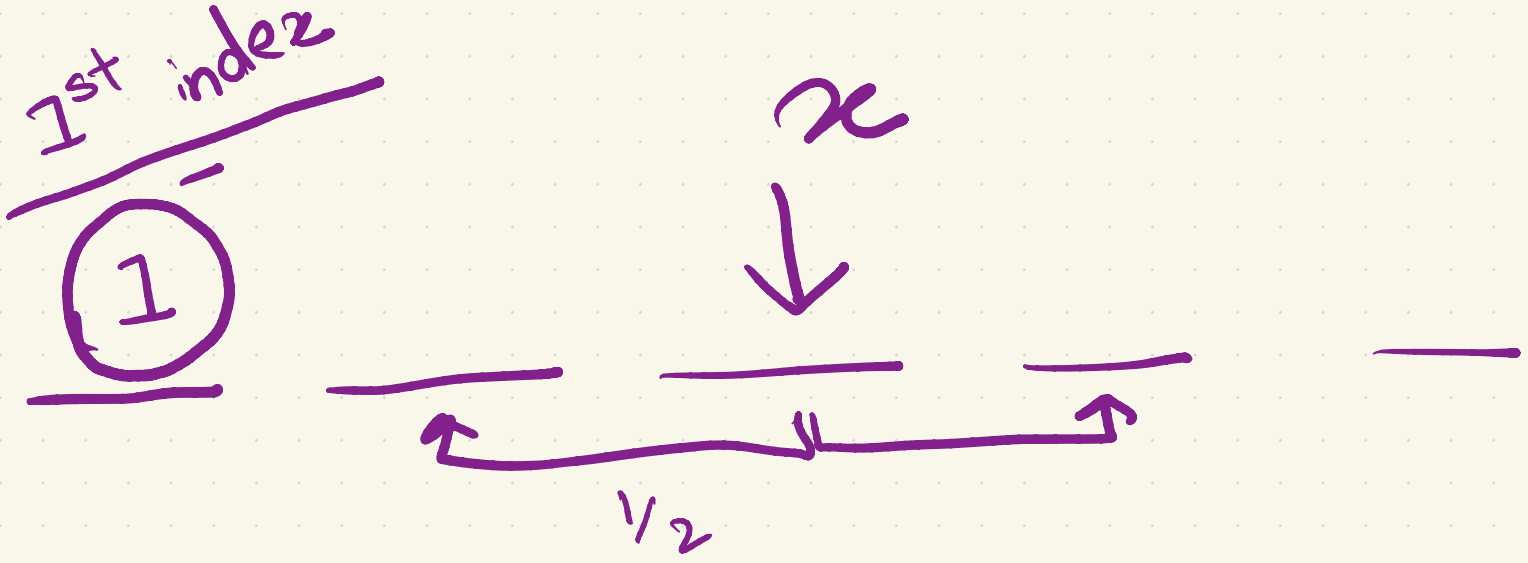
Solution



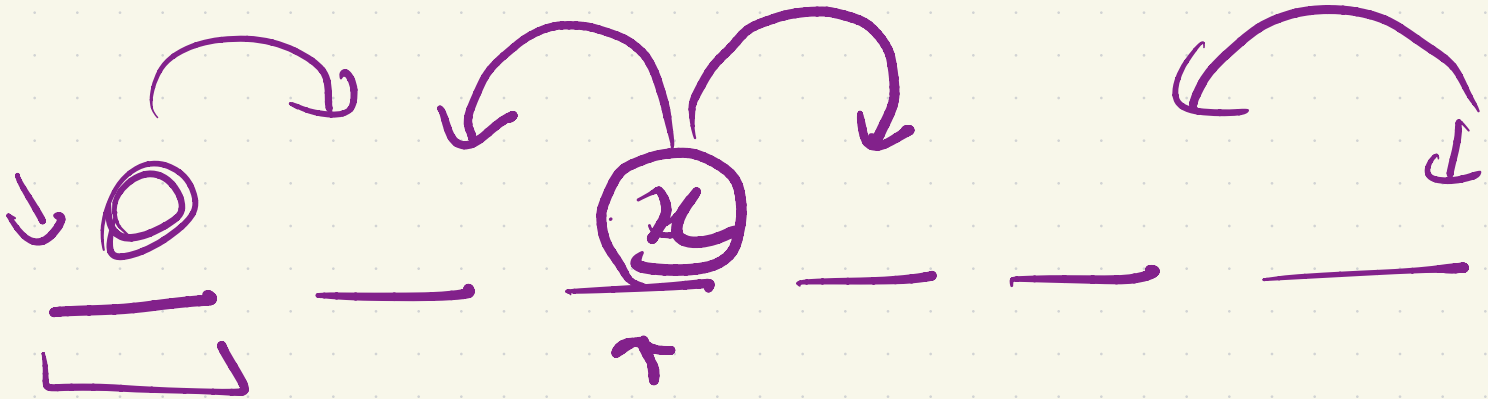
```
vector<double>dp(n, -1);

auto f = [&](int n, auto &&F) -> int {
    if(n == 0) return 0;
    if(dp[n] != -1) return dp[n];
    // dp[n] = 3 / 2 + 1/2 (dp[n - 1] + dp[n / 2])
    return dp[n] = 1.5 + 0.5 * (F(n - 1, F) + F(n / 2, F));
};

cout << f(5, f) << "\n";
}
```

Atcoder DP contest

Sushi ← cyclic DP states



n x

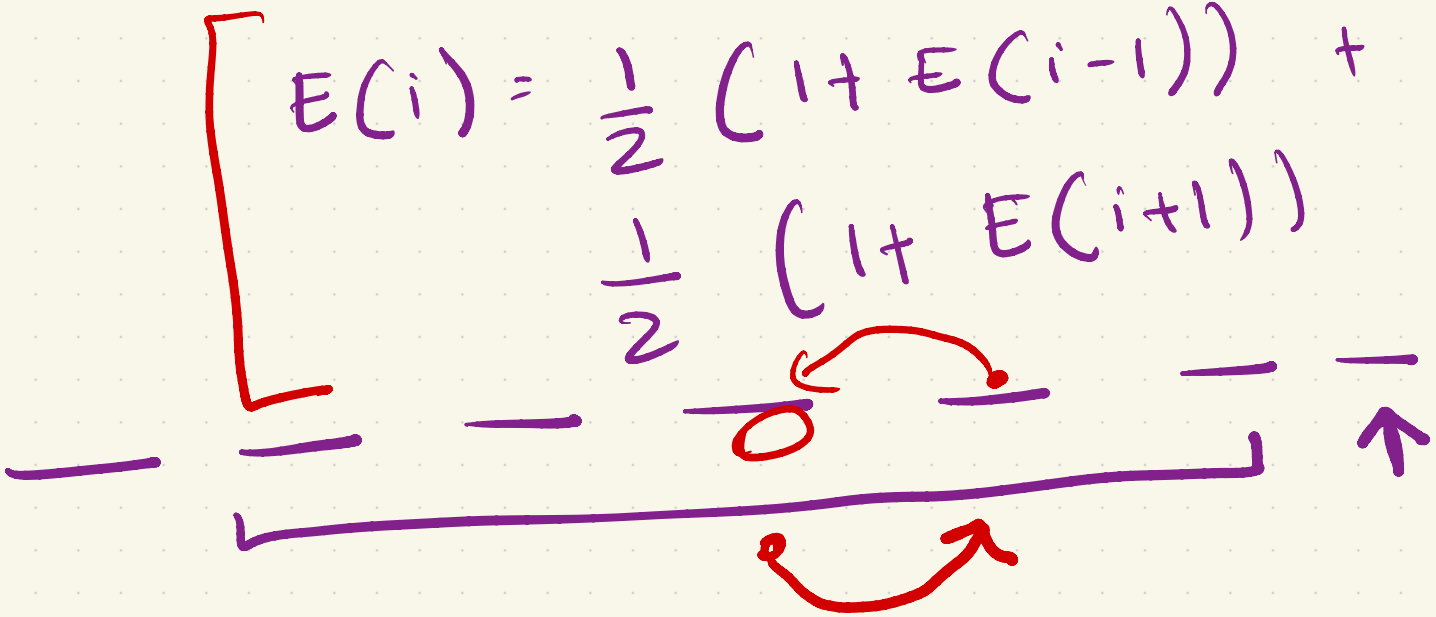


Exp number of moves to reach index I

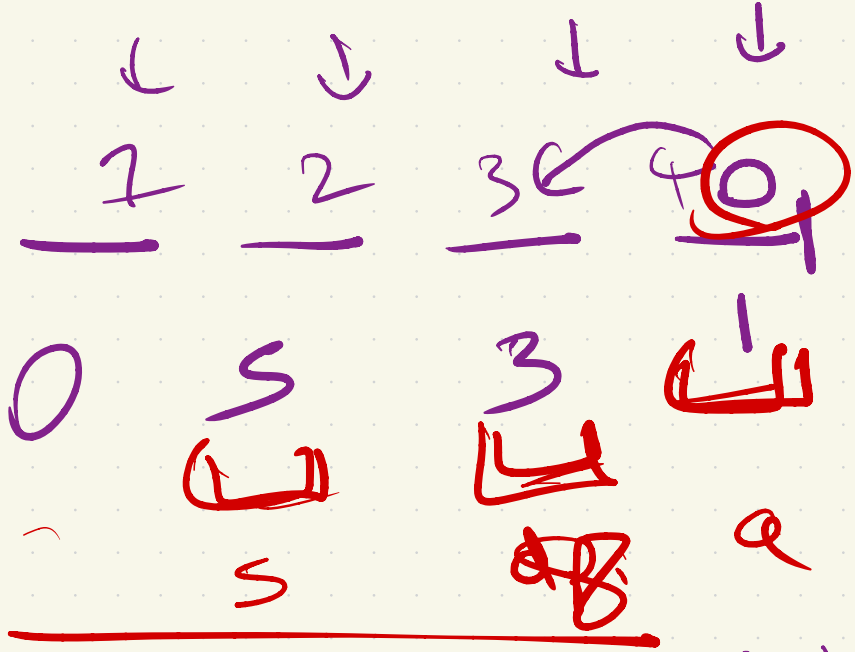
$E(i)$ → exp moves to reach 1 from i

$E(1) = 0$

$$E(i) = \frac{1}{2}(1 + E(i-1)) + \frac{1}{2}(1 + E(i+1))$$



→ Direction is only 1



$$* \quad E(4) = 1 + E(3)$$

$$E(3) = \frac{1}{2}(1 + E(2)) + \frac{1}{2}(1 + E(4))$$

$$= \frac{1}{2} + \frac{E(2)}{2} + 1 + \frac{E(3)}{2}$$

$$\Rightarrow \frac{E(3)}{2} = \frac{3}{2} + \frac{E(2)}{2}$$

$$E(4) = 1 + E(3)$$

$$E(3) = 3 + E(2)$$

$$E(2) = 5 + \underline{E(1)}$$

$$0$$

Good Trick

$$E(2) = 5$$

$$E(3) = 8$$

$$E(4) = 9$$

$$1 + \frac{1}{2} + \frac{1}{4} + \frac{1}{8} \cdots$$

$$= 2$$