



State Elimination & Cyclic DP States

- Gaurish Baliga

State elimination

↳ identify the states

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

$$dp(i, j, k) = \dots$$

i + j = k

memoise??

$$i + j = k$$

~~dp(i, j)~~ \leftarrow $O(n^3)$

$$\rightarrow k = i + j :$$

$O(n^2)$

?
?

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

State Elimination



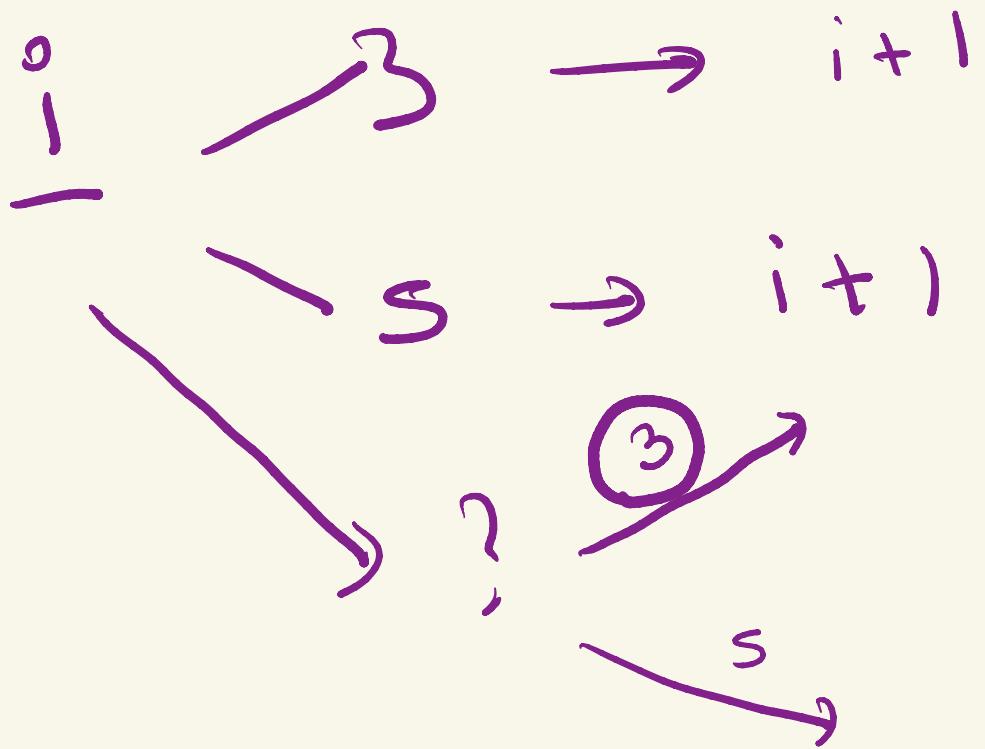
- Ask yourself do you need all the parameters in the dp state?
- If you have $\text{dp}[a][b][c]$, and $a + b = c$, do you need to store c as a parameter or can you just compute it on spot?
- If you can compute a parameter in dp state from other parameters, no need to store it.

What matter are independent states

Problem



Given an array of N (input) 3s, 5s and ?s. Every 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 k_1 (given in input) 3s and k_2 (given in input) 5s in the complete array.



i, j, k
↑ ↑
curr index no. of no. of fires
 three we have placed yet
 we have placed yet
 jet

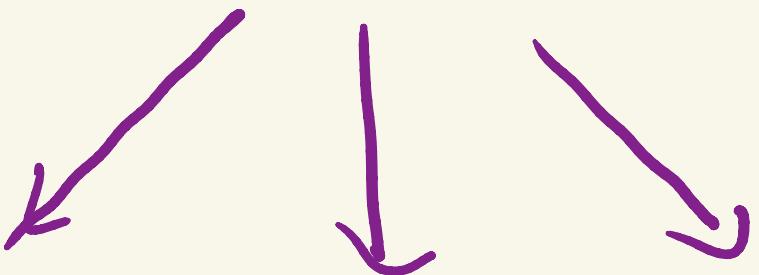
$dp(i, j, j, k) \rightarrow$

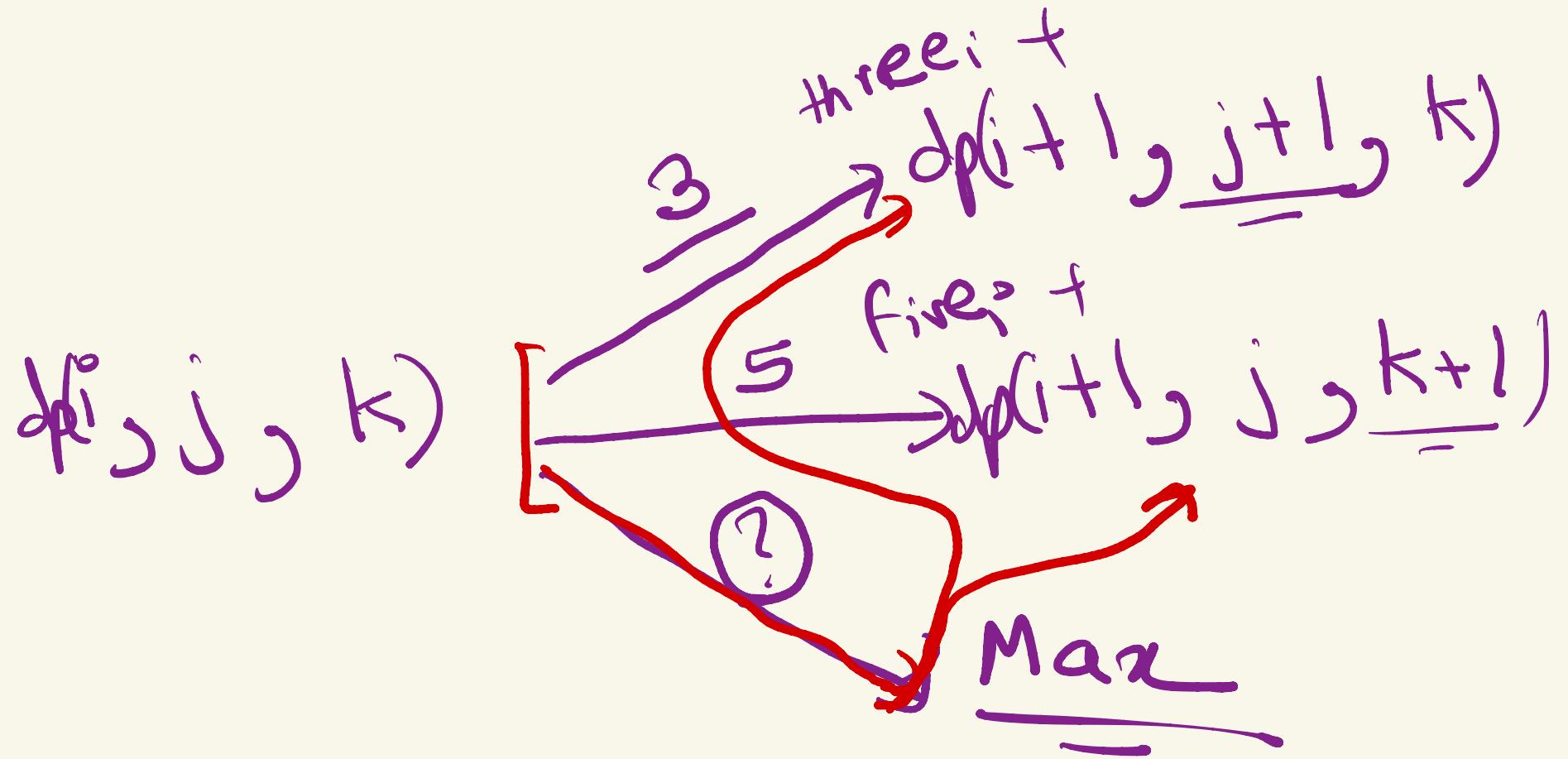
max score we
can get from

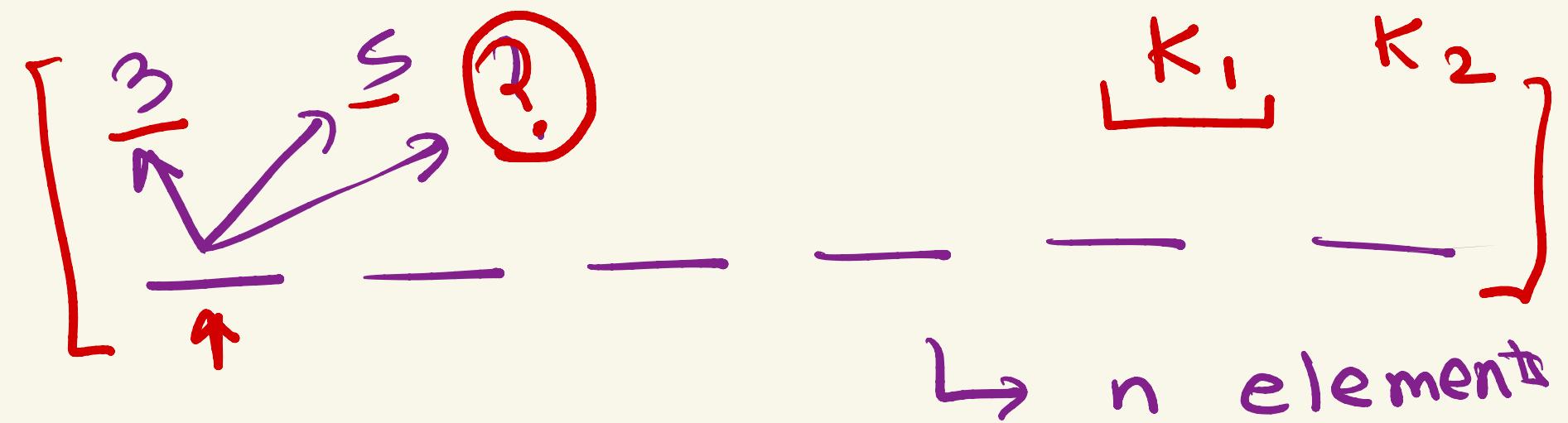
index $(i \dots n-1)$

j is placed

k is placed







Three

$$\frac{+}{\text{—}}$$



Five

$$\frac{+}{\text{—}}$$



Handwritten diagram illustrating the bonding between two nitrogen atoms (N₂). The diagram shows atomic orbitals (AOs) from each nitrogen atom combining to form molecular orbitals (MOs).

The AOs are represented by red numbers: 3, 5, ?, ., 3, ?, 5, 5.

The MOs are represented by purple numbers: 6, 3, 5, 0, 3, 5.

Curly arrows show the movement of electrons from the AOs to the MOs. One curly arrow starts from the first AO (3) and points to the first MO (6). Another curly arrow starts from the second AO (5) and points to the second MO (3). A third curly arrow starts from the fourth AO (3) and points to the fourth MO (0).

Below the diagram, the following labels are present:

- $\rightarrow K_1$ 3s
- $\rightarrow K_2$ 5s
- $\rightarrow a$



Approach 1: Naive Dynamic Programming

MLE
TLE

```
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"
```

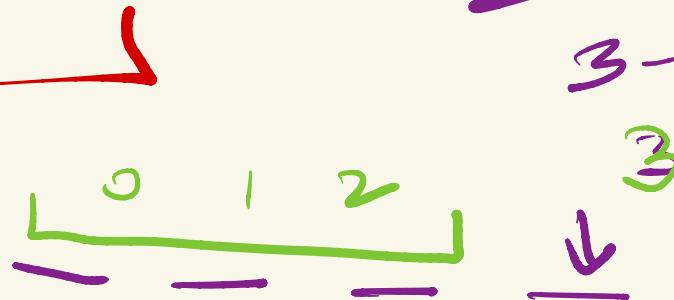
i, j threes

not memoise ??

$$j + k = i$$

$$i - j = k$$

$$3 \quad 3s \\ 3 - 2 = 1$$





Approach 2: Optimised Dynamic Programming

2D

3

```
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?

$$\begin{aligned} a^* b &= c \\ a^b &= c \end{aligned}$$



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

If $(1 \leq a \leq 1000)$ $(1 \leq b \leq 100)$ $(1 \leq c \leq 10000)$, which parameter is best to remove?

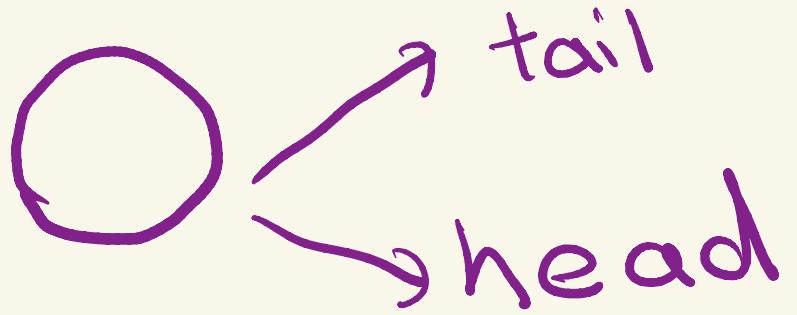
* Always remove the largest state

Cycling DP states

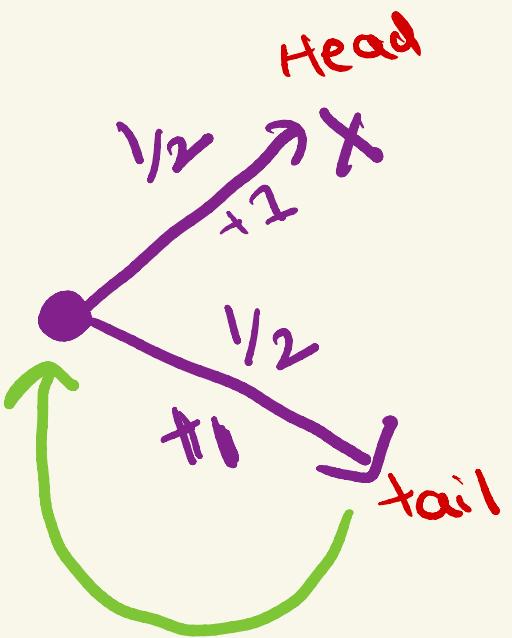
Mathematics



- 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

$$\frac{E(x)}{2} = \text{Head} - \frac{1}{2}(1+0) + \underline{\underline{1 + E(x)}}$$

stop

$$E(a) = \frac{1}{2} (1 + 0) + \frac{1}{2} (1 + \bar{e}(a))$$

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

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

$$E(a) = 2$$

* Problem

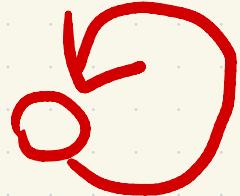


- Given a positive integer $N \leq 1e6$, at every step the following 3 things can happen to N with equal probability.
 - $N = N / 2$ $n = \lfloor n/2 \rfloor$
 - $N = N - 1$
 - N remains unchanged
- Find expected number of steps it will take to convert for N to become 0

$dp(n) \rightarrow$ exp. moves
required to convert
 $n \rightarrow 0$

$$dp[0] = 0$$

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



$dp(n)$

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

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

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

Solution



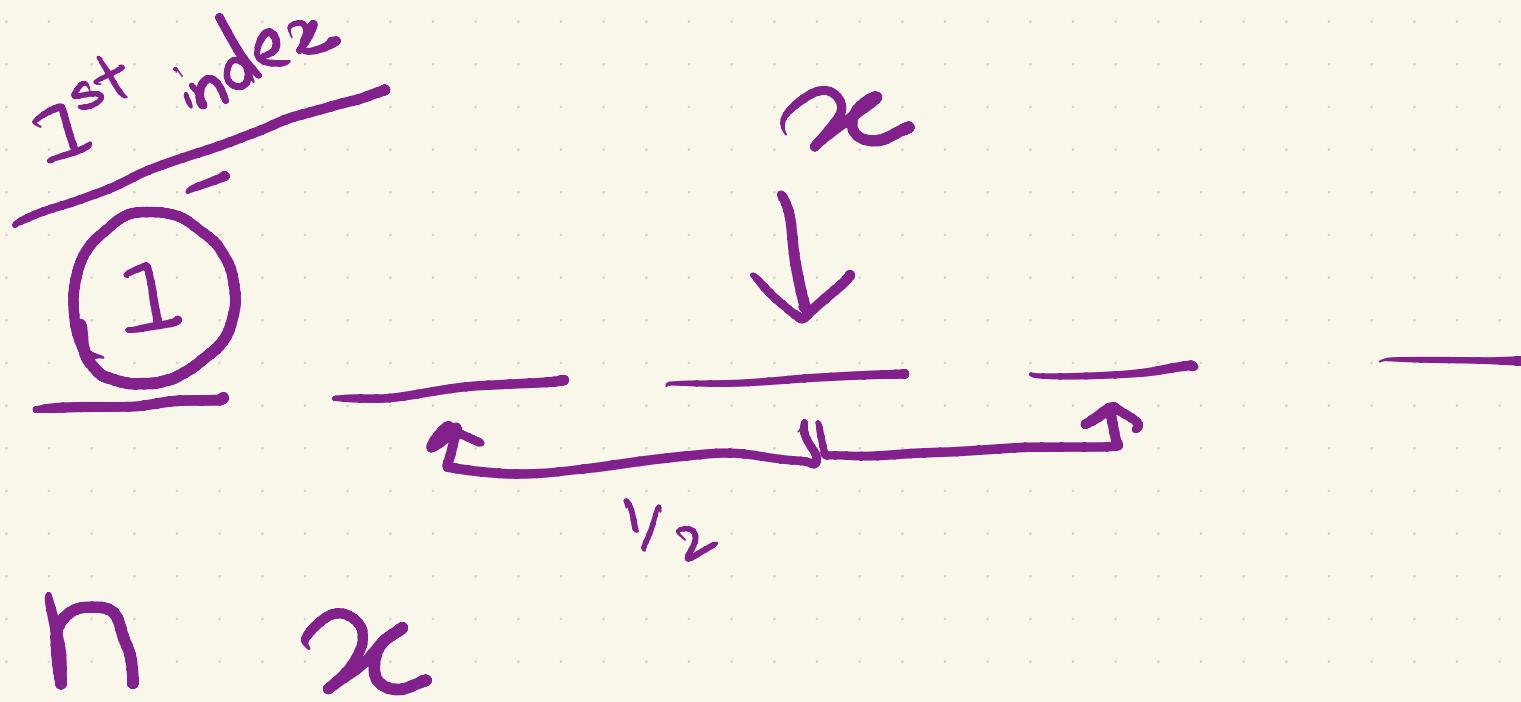
```
vector<double>dp(n, -1);
auto f = [&](int n, auto &&F) -> int {
    if(n == 0) return 0; ←
    → if(dp[n] != -1) return dp[n];
    // dp[n] = 3/2 + 1/2 * (dp[n-1] + dp[n/2]) .
    return dp[n] = 1.5 + 0.5 * (F(n - 1, F) + F(n / 2, F));
};

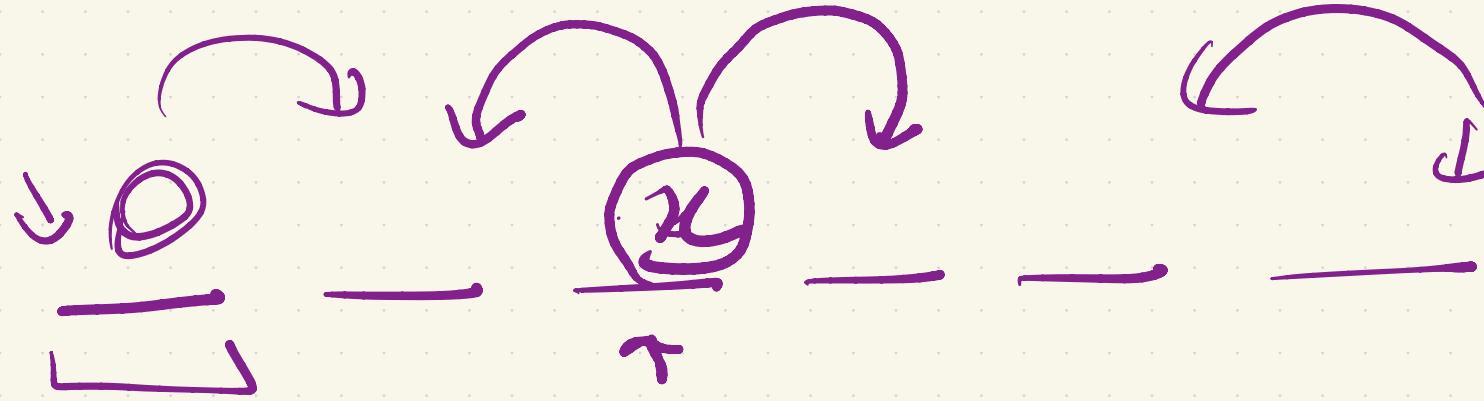
cout << f(5, f) << "\n";
```

The code implements a dynamic programming solution for a problem. It uses a vector `dp` of size `n` initialized to -1. The function `f` is a lambda that takes an integer `n` and a reference to another function `F`. If `n` is 0, it returns 0. Otherwise, it checks if the result for `n` is already computed and stored in `dp[n]`. If not, it calculates it using the formula $dp[n] = 1.5 + 0.5 * (F(n - 1, F) + F(n / 2, F))$, where `F` is the reference to the current function `f`. Finally, it prints the result of `f(5, f)`.

Atcoder DP contest

Sushi \leftarrow cyclic DP
states





Exp number of moves to
reach index 1

$E(i) \rightarrow$ exp moves to reach
1 from i

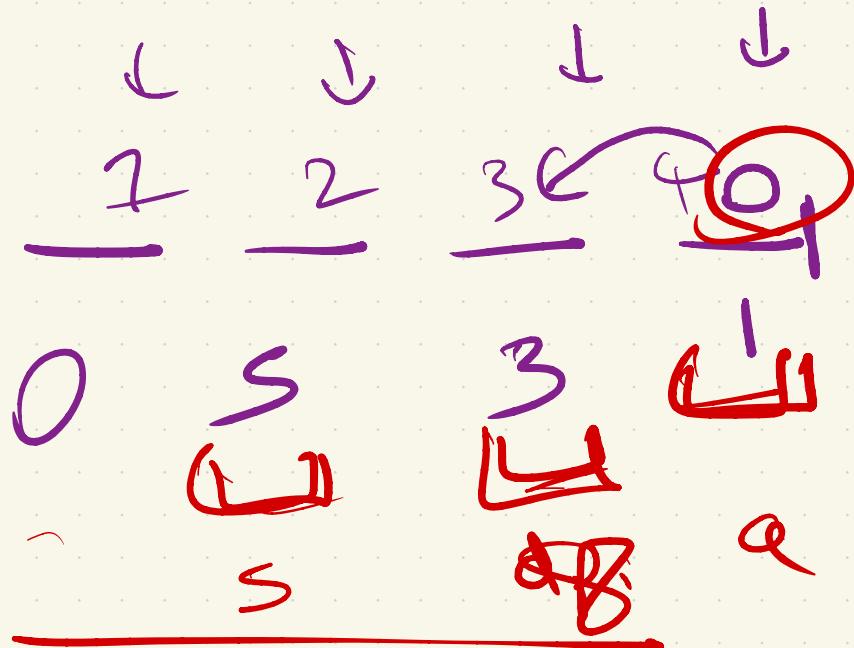
$$E(1) = 0$$

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

The diagram shows a horizontal line with tick marks labeled $i-1$, i , $i+1$, and $i+2$. A red bracket below the line spans from $i-1$ to $i+1$. A red curved arrow points from this bracket to the term $(1 + E(i+1))$. Another red curved arrow points from the same bracket to the term $(1 + E(i-1))$. A red L-shaped bracket on the far left encloses the tick marks $i-1$ and i .

→ Direction is only 1

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



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

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

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

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

$$\left[\begin{array}{l} E(4) = \underline{1} + E(3) \\ E(3) = \underline{3} + E(2) \\ E(2) = \underline{5} + E(1) \\ \hline 0 \end{array} \right.$$

Good
Trick

$E(2) = 5$
$E(3) = 8$
$E(4) = 9$

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

$$= 2$$