Objective: Determine exercise or not to maximize the total performance

- Brute force: $\Omega\left(2^N\right)$. There are two options for each day: exercise or not.
- The problem is linearly ordered $\rightarrow$ dynamic programming?

dynamic programming to solve the problem:

define $cw[i]$: performance if exercise continuously for i day
(cw for continuous work)
define $r[i]$: maximum performance if the i th day rest
define wn : maximum performance if the Nth day work.

Base case:

$$cw[1] = 0 + A - 1^2 \cdot B = A - B$$

$$r[1] = 0 - R[1] = -R[1]$$

(Assume index starts from 1)

Transition function:

$$cw[i] = cw[i-1]A - i^2 \cdot B$$

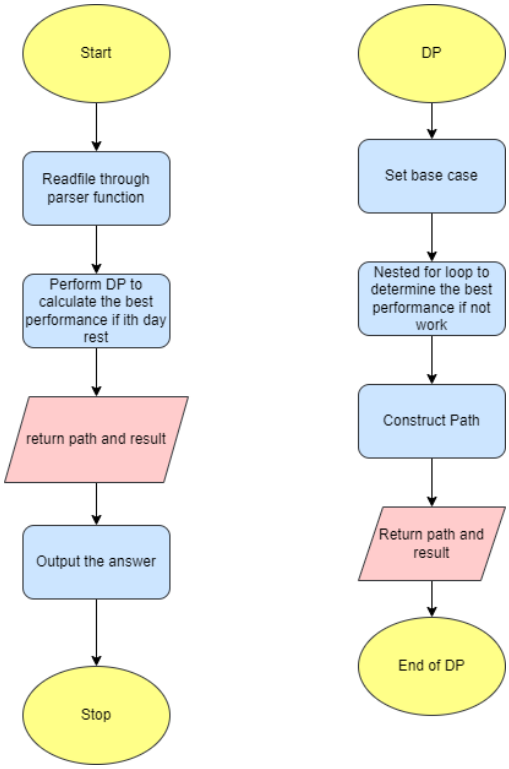
$$r[i] = \max_{j<i}(r[i-j-1] + cw[j])$$

$$wn = \max_{1 \le i \le N}(r[N-i] + cw[i])$$

$$result = \max(r[N], wn)$$

Early return if work continuously can't improve the performance.

Flow chart:



Time complexity analysis:

Read file: $\theta(N)$

Bottom-up dynamic programming for memorization: $\theta(N^2)$

Backtracking the solution path: $\theta(N)$

Output answer: $\theta(N)$

Therefore, the overall time complexity is $\theta(N^2)$