T191 - Distributing Cards

Author : Alex Poon
Data : Jeremy Chow
4-5-2019

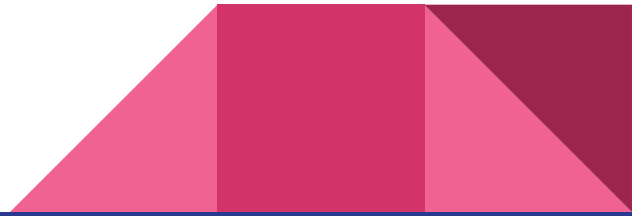
Problem

Given N cards, consist of 'F' or ‘S’ card

Each time you pick a ‘F’ card, some cards on the right of it will be deleted

Find the number of ways to pick all ‘F’ cards

➔ No ‘F’ card should be deleted



Subtask 1

$1 \le N \le 10$

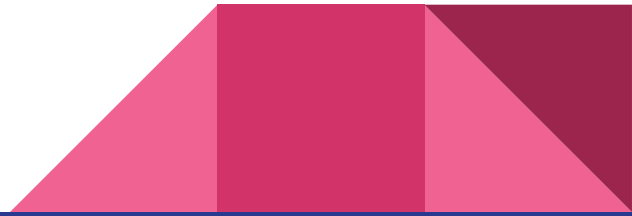
➔ Try all pick sequence (permutation)

➔ Determine if a given pick sequence is valid or not

- Naive delete = $O(N^2)$
- Linked list = $O(N)$

➔ Time complexity = $O(N! * N)$ or $O(N! * N^2)$

➔ C++ std::next_permutation



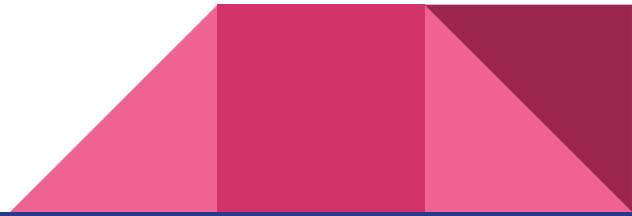
Subtask 2

of 'F' cards ≤ 10

➔ Similar to subtask 1

➔ Try all ‘F’ cards’ pick sequence (permutation)

➔ Determine if a given pick sequence is valid or not



Subtask 2

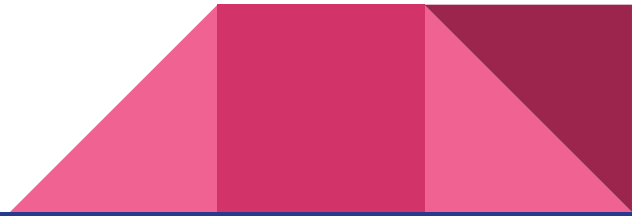
Determine if a given pick sequence is valid or not

- ➔ For each ‘F’ card, find the closest ‘F’ card which is not deleted
- ➔ Calculate the number of card between them which is not deleted
  - ◆ Minus the effect brought by previous deleted ‘F’ card
- ➔ Compare it with c_i

Subtask 2

M = # of ‘F’ cards

Time complexity = $O(M! * M^2)$

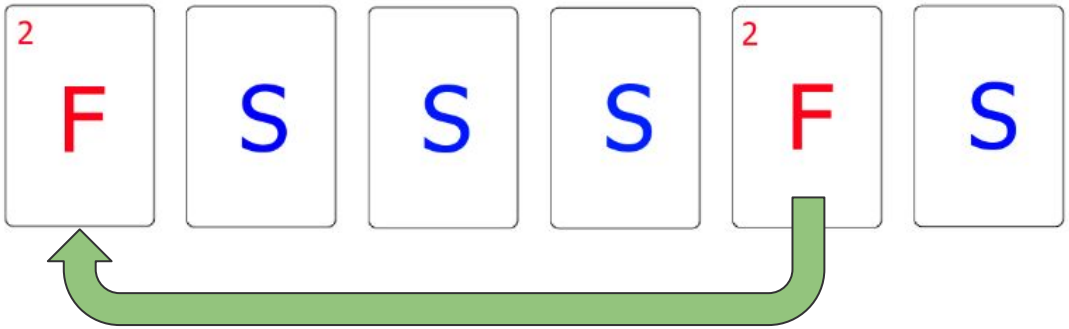


Subtask 3

Observation : Some ‘F’ cards depends on other ‘F’ cards

➔ You can only pick it after picking some other ‘F’ cards

Sample 1



Subtask 3

$0 \le c_i \le 1$

$c_i = 0$ if the card in its left is a ‘F’ card

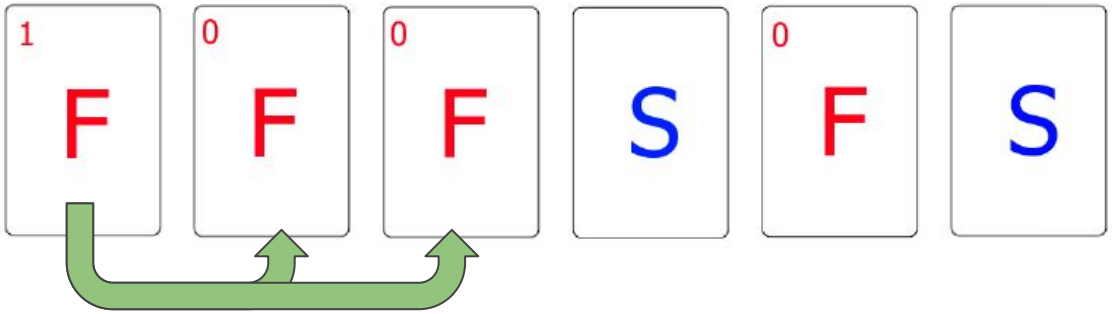
➔ Two patterns for ‘F’ cards

- cost = 1 followed by zero or more cost = 0
- cost = 0

Subtask 3

Two patterns for ‘F’ cards

➔ Two type of dependence relationship

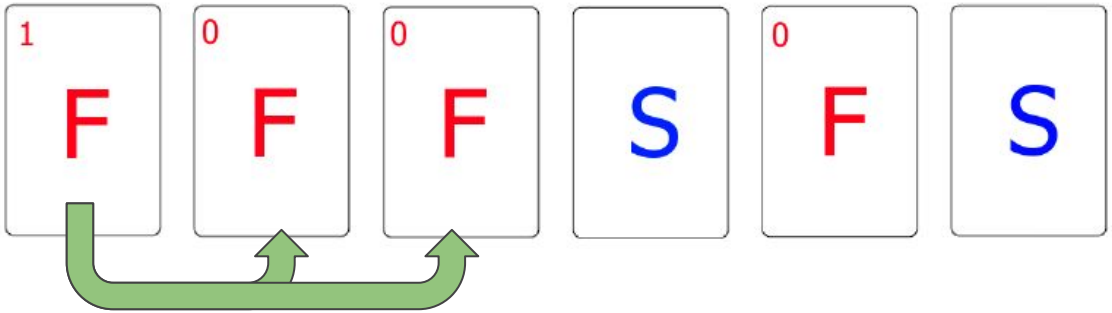


Subtask 3

Arrangement within group // = (size - 1)!

➔ First type : (# of cost = 0 ‘F’ card)!

➔ Second type : 1

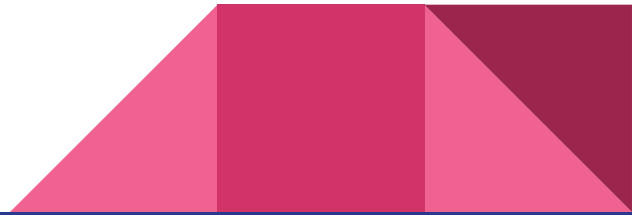


Subtask 3

Arrangement between groups

Assume there are groups of size s_1, s_2, ..., s_m

Ways = $ncr(s_1, s_1) * ncr(s_1 + s_2, s_2) * \ldots * ncr(s_1 + s_2 + \ldots + s_m, s_m)$

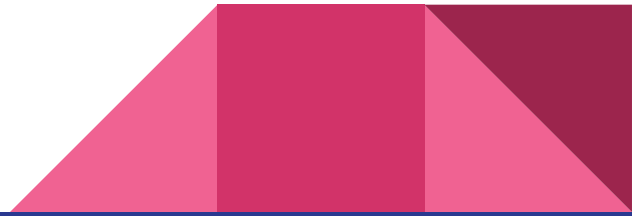


Subtask 3

Ans = arrangement between groups * arrangement within group

Guarantee ways > 0 (No impossible case)

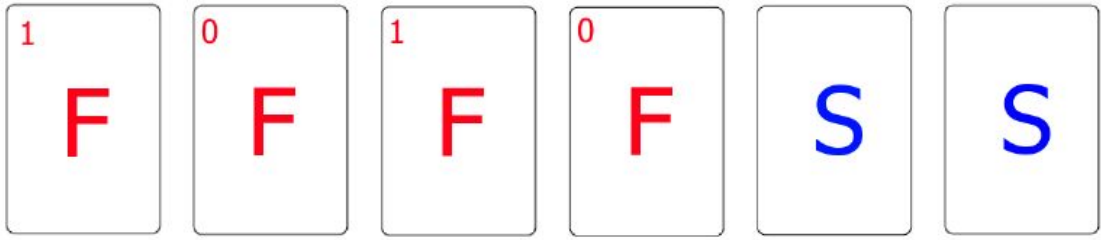
Time complexity = O(N)



Subtask 4

$1 \le N \le 1000$

Complex dependence relationship might appear

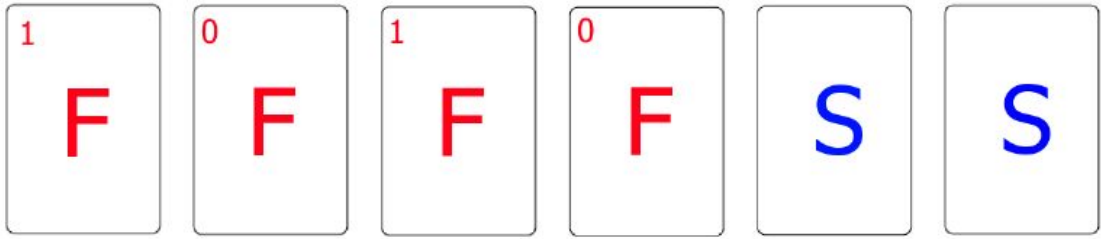


Subtask 4

How to find the dependence relationship?

len[i] = length of range s.t. you can pick i^{th} card

if all ‘F’ cards in [(i + 1) mod n..(i + len[i] - 1) mod n] are picked



Subtask 4

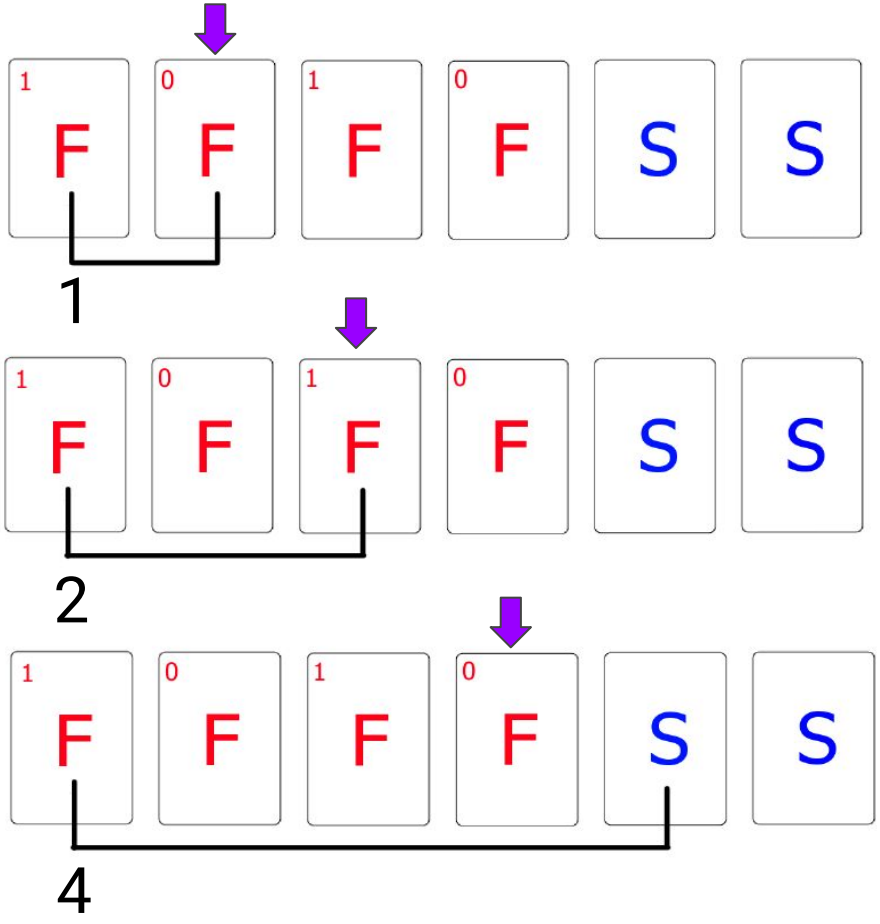
len[i] = c_i, lb = (i + 1) mod n

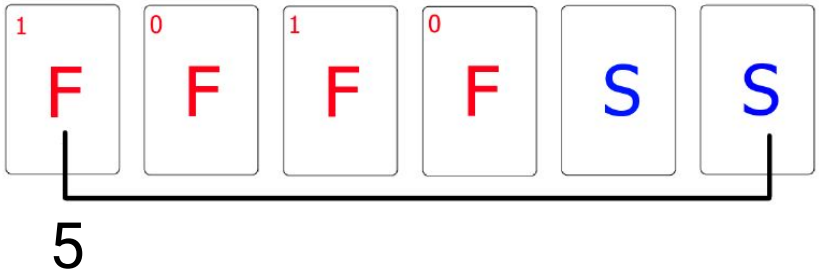
while there is a ‘F’ card ($card_j$ = ‘F’) in the range [lb..(i + len[i] - 1) mod n]

    len[i] += cost[j] + 1

    lb = (j + 1) % n

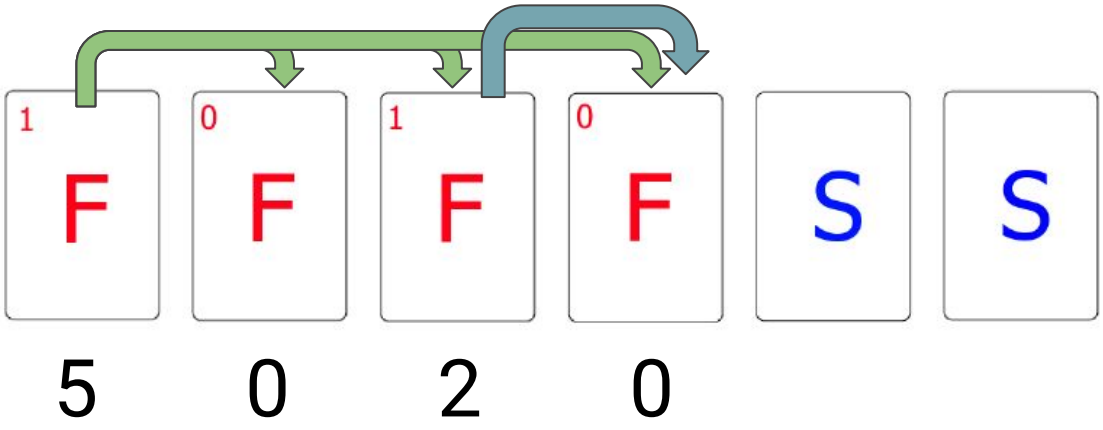
➔ pick a card will remove (cost of that card) + 1 cards


1 F
0 F
1 F
0 F
S
S
1
1 F
0 F
1 F
0 F
S
S
2
1 F
0 F
1 F
0 F
S
S
4


1 F
0 F
1 F
0 F
S
S
5

Subtask 4

i^{th} card depends on j^{th} card if j is in [(i + 1) mod n..(i + len[i] - 1) mod n]



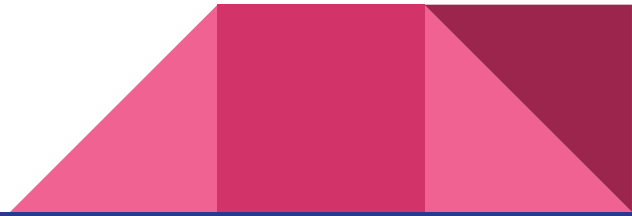
Subtask 4

len[i] ≥ n - 1 ⇒ ways = 0 (impossible)

Find all the dependence relationship

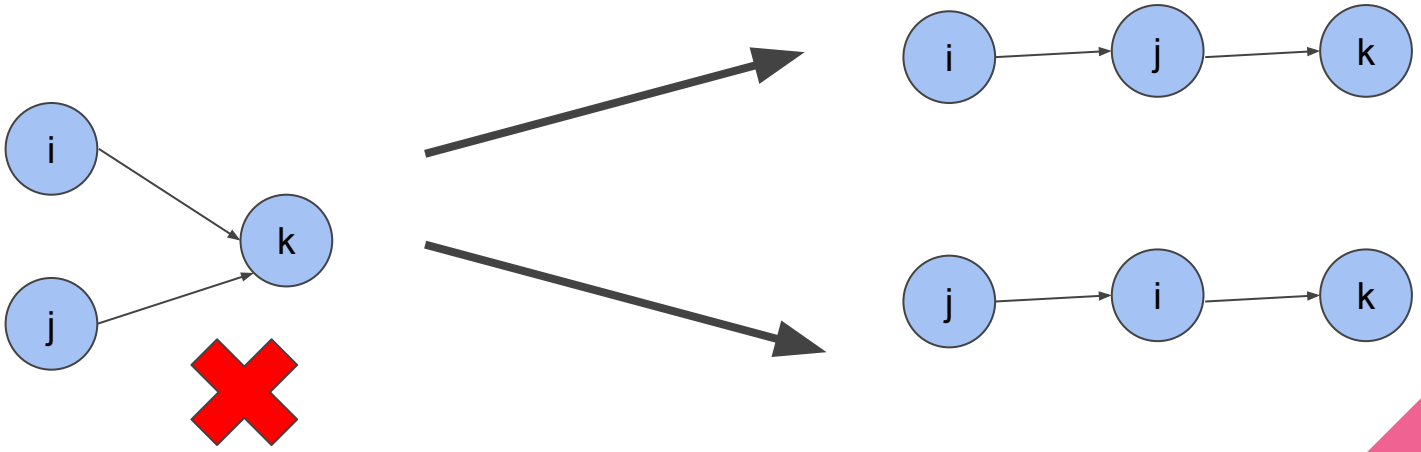
Ans = # of possible topological sort in DAG

#P-complete problem

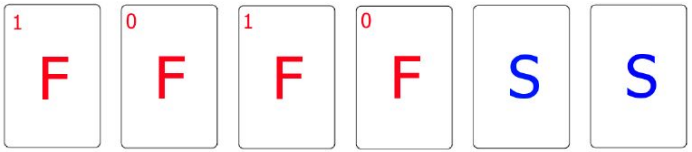


Subtask 4

Observation : The dependence relationship can always simplify to a tree



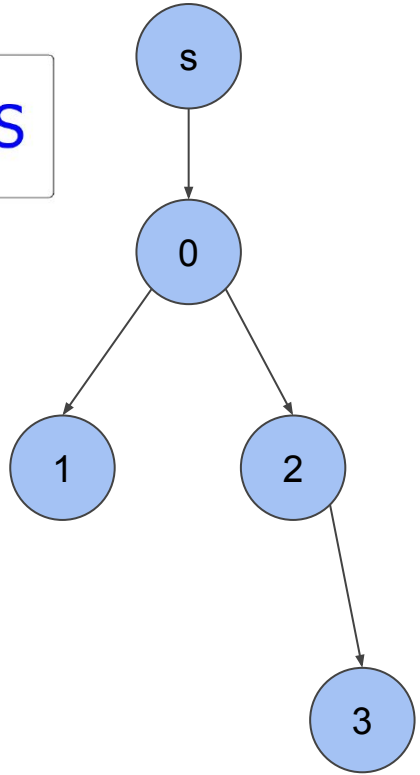
Subtask 4



For each i^{th} card (‘F’ card)

Find the rightmost ‘F’ card on its left that “cover” i

Build an edge from it to i



➔ Can be a forest

➔ Add an extra node to connect all the root

Subtask 4

How to calculate # of topological sort in a tree?

Dp[i] = # of topoplogical sort in the subtree of i^{th} node

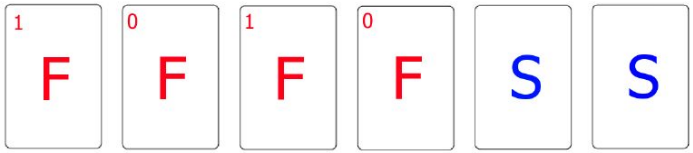
Let $a_{i,1}$, $a_{i,2}$ ...$a_{i,m}$ be the children of i^{th} node

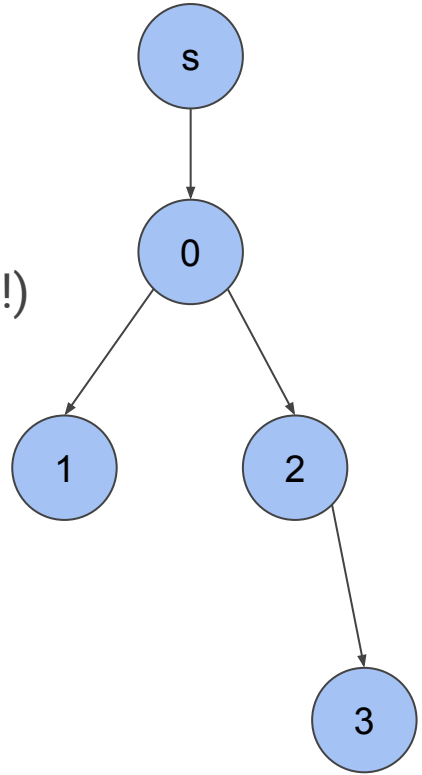
Dp[i] = arrangment within group * arrangement between groups

= (siz[i] - 1)! * dp[$a_{i,1}$] * … * dp[$a_{i,m}$] / (siz[$a_{i,1}$]! * … * siz[$a_{i,m}$]!)

Subtask 4

$dp[i] = (siz[i] - 1)! * dp[a_{i,1}] * \dots * dp[a_{i,m}] / (siz[a_{i,1}]! * \dots * siz[a_{i,m}]!)$





$dp[0] = (siz[0] - 1)! * dp[1] * dp[2] / (siz[1]! * siz[2]!)$

$= 3! * 1 * 1 / (1! * 2!) = 3$

Ans = dp[s]

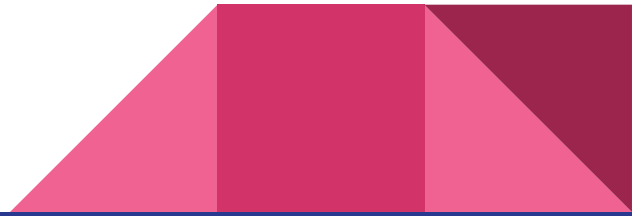
Subtask 4

Remeber to module the answer by $10^9 + 7$

- ➔ $O(N^2)$ to calculate pascal triangle for NCR
- ➔ O(N / NlogN) to calculate modular inverse
  - Math in OI (I)

Time complexity = $O(N^2)$

- ➔ You need $O(N^2)$ to build the tree



Full solution

Optimize the process of building tree to O(N)

while there is a ‘F’ card ($card_j$ = ‘F’) in the range [lb..(i + len[i] - 1) mod n]

len[i] += len[j] + 1 ~~cost[j] + 1~~

lb = (j + 1) % n

➔ Calculate len[] recursively! (or with stack)
➔ Like memorization in dp

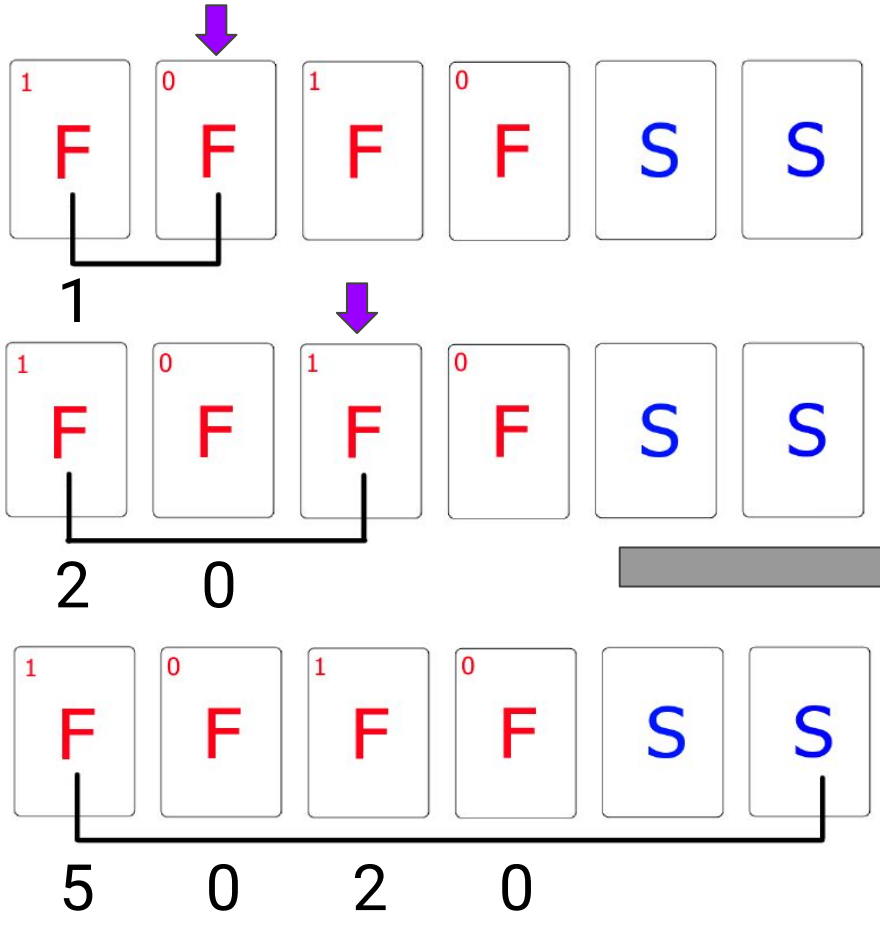
Full solution

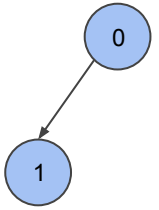
while there is a ‘F’ card ($card_j$ = ‘F’) in the range [lb..(i + len[i] - 1) mod n]

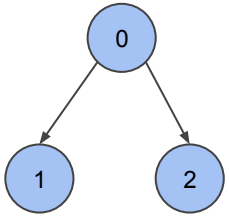
len[i] += len[j] + 1 ~~cost[j] + 1~~

add edge from i to j

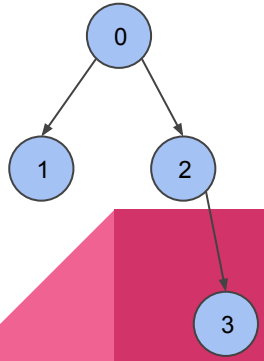
lb = (j + 1) % n







SOLVE(2)



Full solution

Build tree = O(N)

Calculate dp = O(N)

Calculate modular inverse = O(N) or O(NlogN)

Overall time complexity = O(N) or O(NlogN)