Lab11 solution

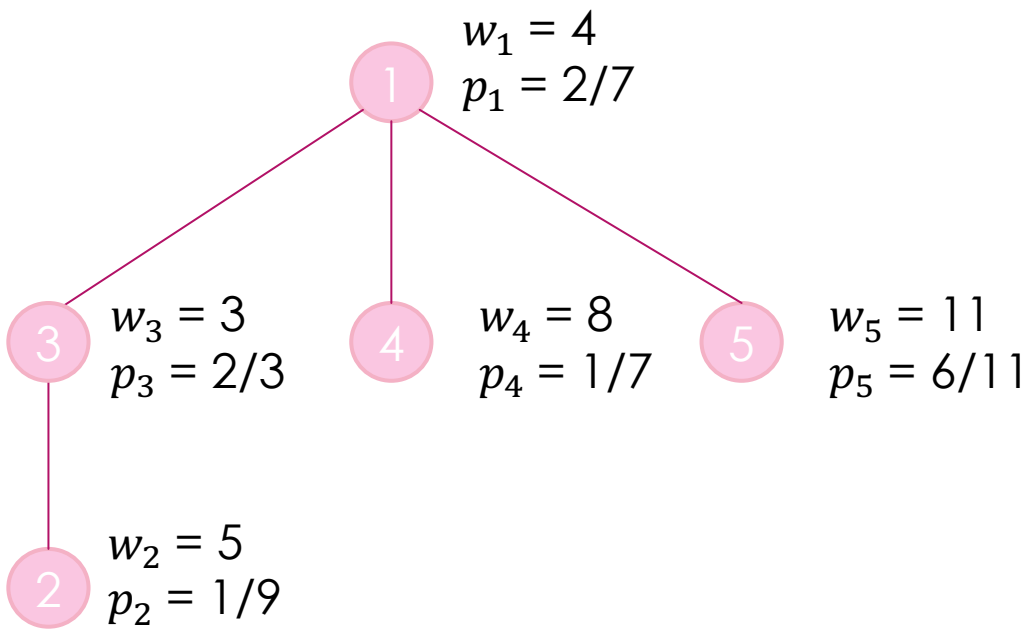
YAO ZHAO

Lab11.A Federated Learning

- Federated Learning(FL) is a branch of machine learning which is developed for better data and privacy security. In FL, let's consider a very simple and interesting case.
- Assume we build a FL system in a tree structure, there are n clients in this FL system. Each client has its local model w_i. Now the trained global model will be sent to this FL system from the center server. Each node has a probability of $\frac{w_i}{\sum_{j=1}^{n} w_j}$ to receive the global model.
- Starting from the client which has received the global model, it will update itself to the global model and send the global model to its adjacent client to update. Each not updated client has a probability of p_i to be updated if **at least** one of its adjacent clients has updated to the global model. Also, the updated client can continue to send the global model to its adjacent client.
- Now the problem is that, if exactly one client will get the global model initially, can you calculate the probability that exactly $\boldsymbol{k}$ clients are eventually updated?

Sample input:

```
5  3
3  1
3  2
1  5
1  4
4  2  7
5  1  9
3  2  3
8  1  7
11  6  11
```



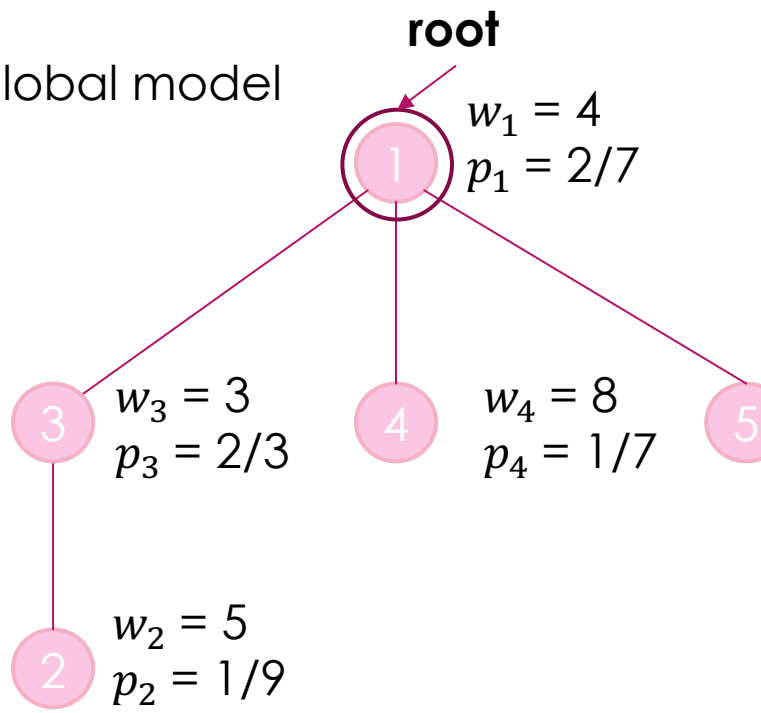
assume client 1 first gets the global model

$\sum_{j=1}^{n} w_j$=4+5+3+8+11= 31

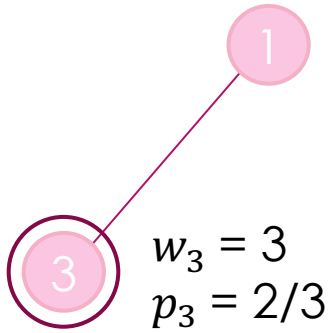
Probability: FP[1][1] = 4/31

client 1

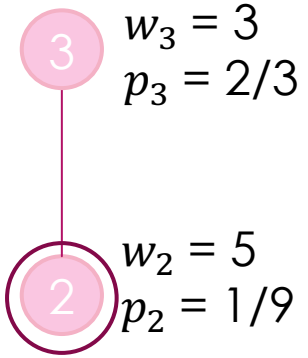
How many nodes are updated to the global model in the subtree of client 1.



Try to do DFS from client 1 to calculate the probability:



For the subtree client 3, which is a child of client 1. Initial the client3:
The probability of updating to the global model for client 3: P[3][1] = 2/3
The probability of not updating to the global model for client 3 : P[3][0] = 1/3



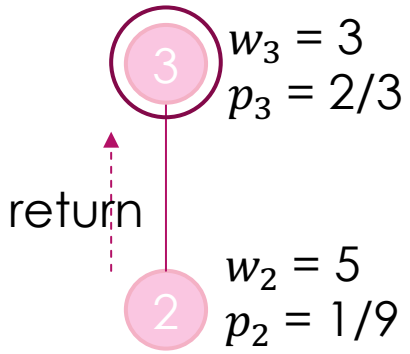
client 3 has a child Client 2, go to Client 2, Initial the client2:
The probability of updating to the global model for client 2: P[2][1] = 1/9
The probability of not updating to the global model for client 2 : P[2][0] = 8/9

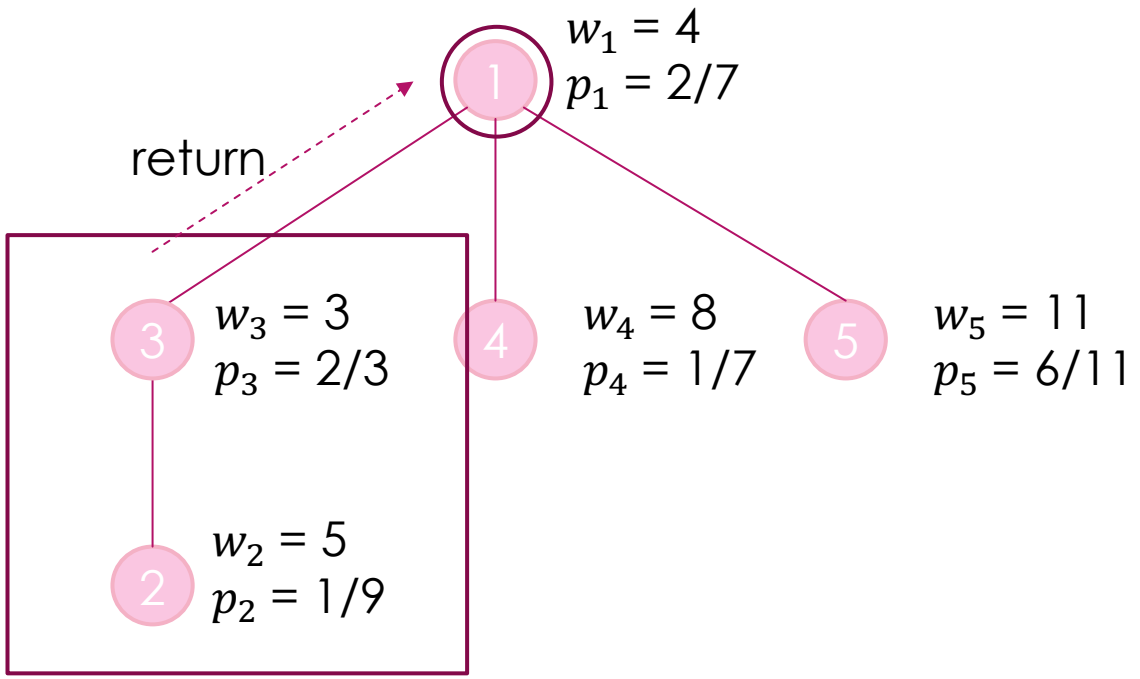
Go on DFS, Client 2 has no child, return to Client 3.
Then Client 3 began to merge with Client 2:
If Client 3 updates to the global model, client 2 maybe update too.
If Client 2 is updated successfully, then the node number of updated to the global model in the subtree of Client 3 becomes 2.

P[3][2] = P[3][1]*P[2][1]

If Client 2 failed to update, the updated node number in the subtree of Client 3 is still 1:

P[3][1] = P[3][1]*P[2][0]



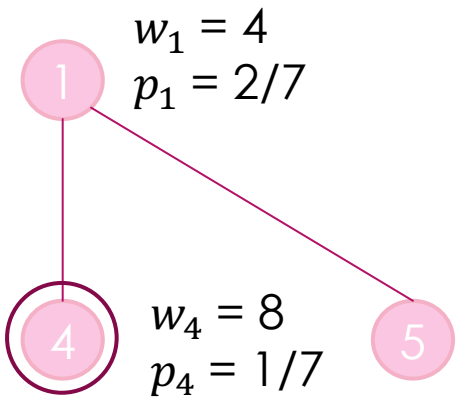


Go on DFS, Client 3 has no other child, return to Client 1.

Client 1 merge with subtree client 3:
FP[1][3] = FP[1][1]*P[3][2]
FP[1][2] = FP[1][1]*P[3][1]
FP[1][1] = FP[1][1]*P[3][0]

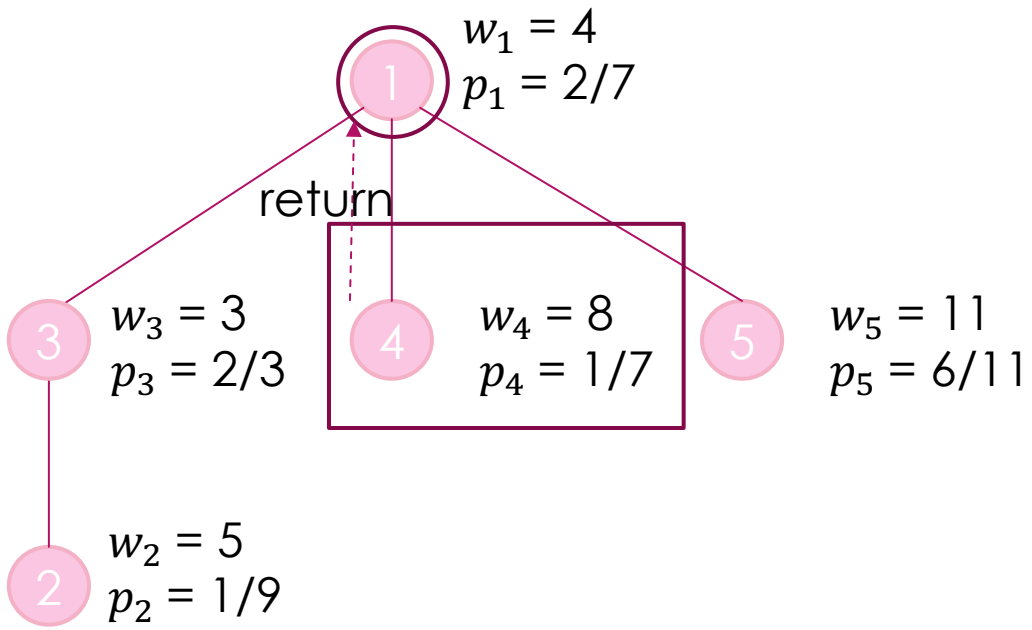
Go on DFS, Client 1 has another child Client 4.

Initial the client4:
The probability of updating to the global model for client 4: P[4][1] = 1/7
The probability of not updating to the global model for client 4 : P[4][0] = 6/7



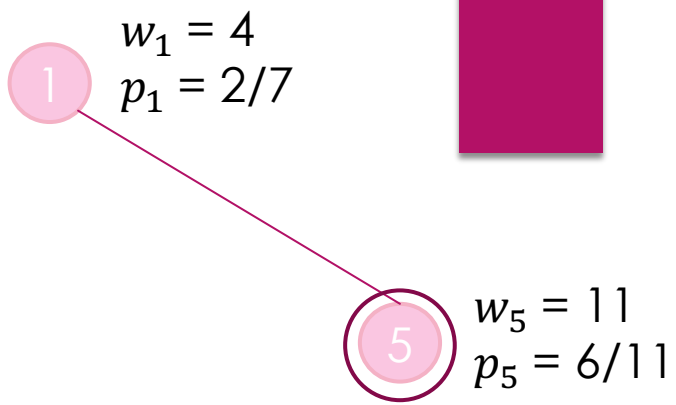
Go on DFS, Client 4 has no child, return to Client 1.

Client 1 merge with subtree client 4:
FP[1][4] = FP[1][3]*P[4][1]
FP[1][3] = FP[1][3]*P[4][0]+FP[1][2]*P[4][1]
FP[1][2] = FP[1][2]*P[4][0]+FP[1][1]*P[4][1]
FP[1][1] = FP[1][1]* P[4][0]



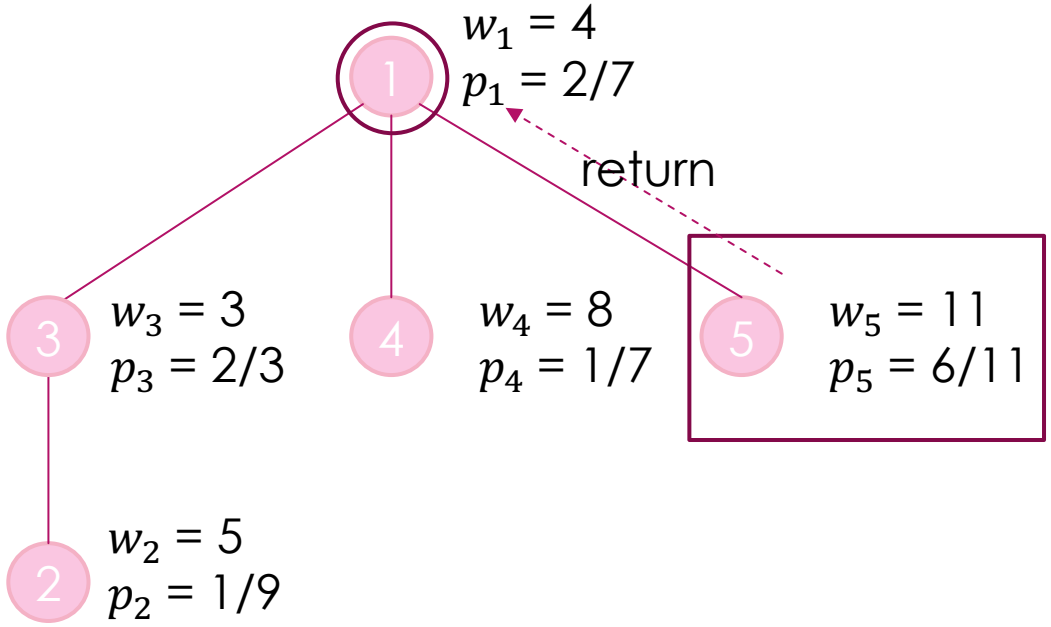
Go on DFS, Client 1 has another child Client 5, return to Client 1.

Initial the client5:
The probability of updating to the global model for client 5: P[5][1] = 6/11
The probability of not updating to the global model for client 5 : P[5][0] = 5/11



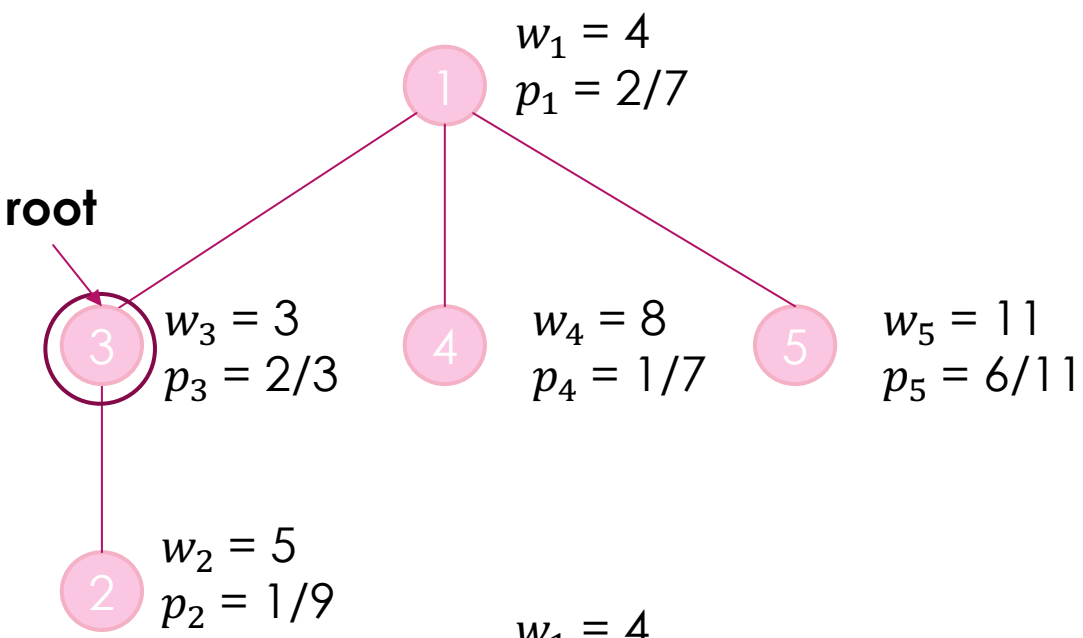
Go on DFS, Client 5 has no child, return to Client 1.

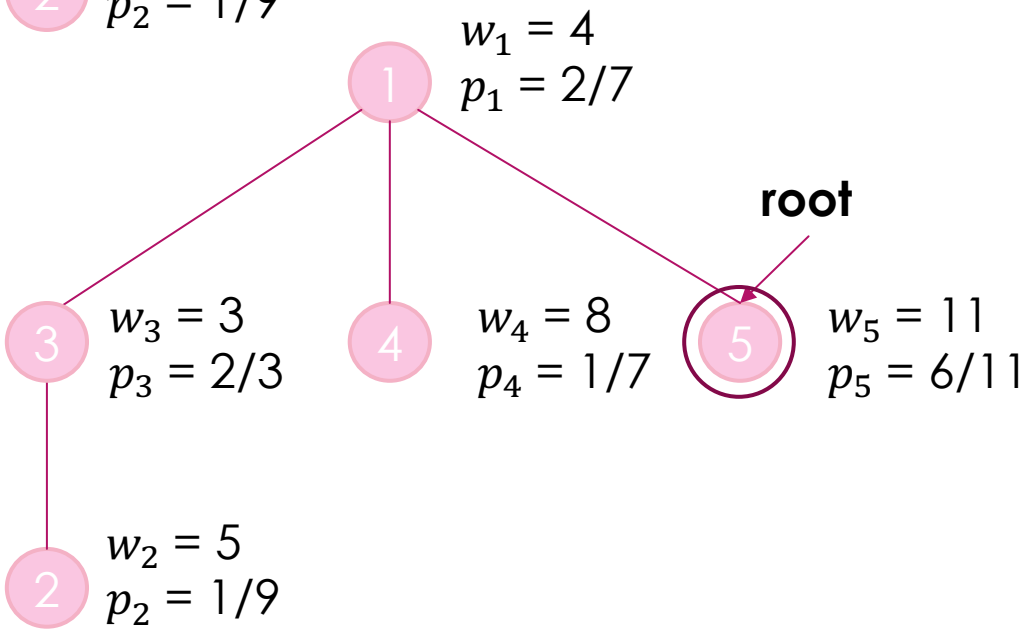
Client 1 merge with subtree client 5:
FP[1][5] = FP[1][4]*P[5][1]
FP[1][4] = FP[1][4]*P[5][0]+FP[1][3]*P[5][1]
FP[1][3] = FP[1][3]*P[5][0]+FP[1][2]*P[5][1]
FP[1][2] = FP[1][2]*P[5][0]+FP[1][1]*P[5][1]
FP[1][1] = FP[1][1]*P[5][0]

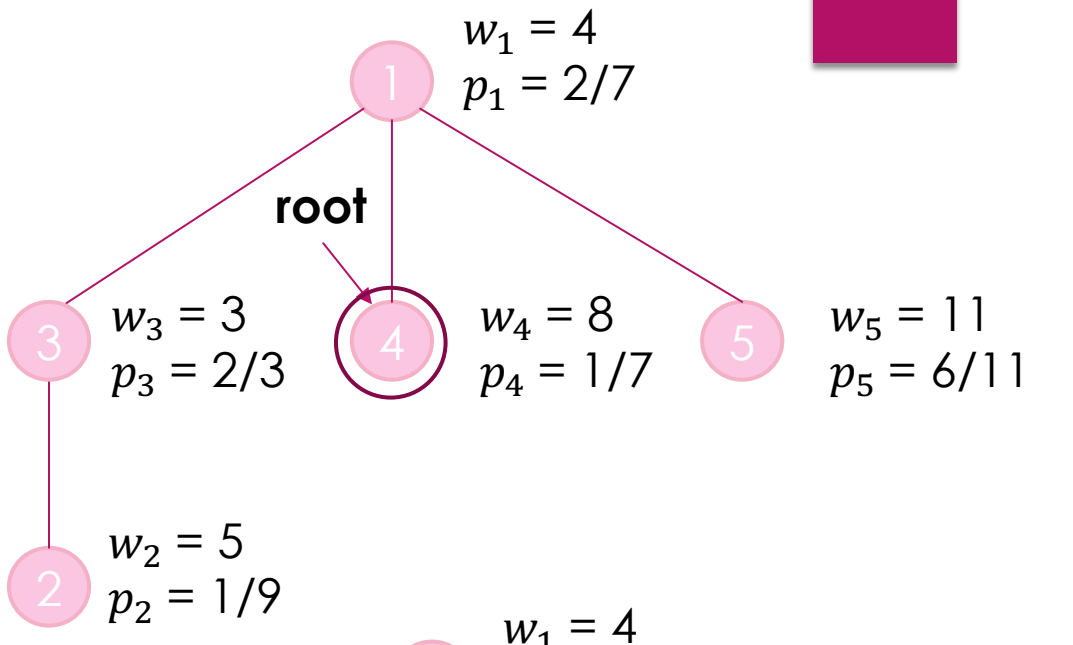


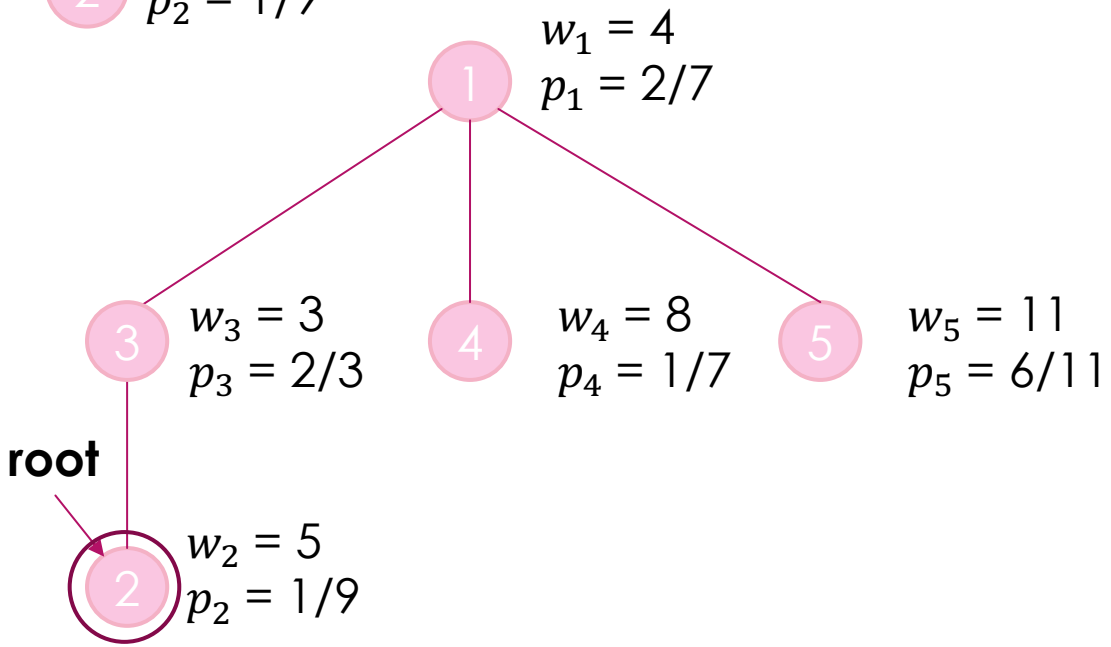
Client 1 has no other children, now FP[1][3] is the probability that exactly **3** clients are eventually updated if Client1 gets the global model initially.

Continue to do the same thing 4 times, we can get the final answer.









Modulo operator

- In algorithmic problems, we sometimes encounter data that goes beyond long(C++: long long). In this case, the problem often asks us to output the answer modulo some prime number. This is where modulo operations come in.
  - +: (a + b) % p = (a % p + b % p) % p
  - -: (a - b) % p = (a % p - b % p) % p
  - *: (a * b) % p = (a % p * b % p) % p
  - ^: a ^ b % p = ((a % p)^b) % p
- **/ will talk later**

Error 1:

```
#include <iostream>
using namespace std;

using ll = long long;
const ll mod = 998244353;

int main() {

    ll a = 1e10, b = 1e10, c;
    c = a * b; //Wrong: maybe overflow
    c %= mod;
    // Right: c = (a % mod) * (b%mod) % mod;

    return 0;
}
```

Error 2:

```
#include <iostream>
using namespace std;

using ll = long long;
const ll mod = 998244353;

int main() {

    ll a = 2, b = 9, c;
    c = (a - b) % mod;//Wrong: The result may be a negative number
    //Right: c = (a – b + mod) % mod;

    return 0;
}
```

Modulo operator :/

(a / b) %p = (a * inv(b)) % p

Here inv[b] is modular multiplicative inverse of b.

In mathematics, particularly in the area of arithmetic, a modular multiplicative inverse of an integer b is an integer such that the product of b*inv(b)is congruent to 1 with respect to the modulus m. In the standard notation of modular arithmetic this congruence is written as:

b*inv(b) (mod p) = 1

How to compute Modular multiplicative inverse:

https://oi-wiki.org/math/number-theory/inverse/

Error 3:

```
#include <iostream>
using namespace std;

using ll = long long;
const ll mod = 998244353;

int inv[20];

int main() {

    ll a = 10, b = 5, c;
    c = (a / b) % mod //Wrong
    // Right: c = a * inv[b] % mod;

    return 0;
}
```

https://oi-wiki.org/math/number-theory/inverse/

Lab11.B

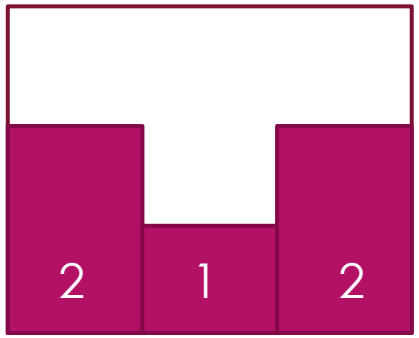
- You and Justin are playing a game on a $n \times n$ chessboard. Justin first cuts out some squares in the chessboard, for the $i-th$ column from left to right, he only leaves the bottommost h_i squares.
- Then you can place any number of rooks on the chessboard. A rook can move horizontally or vertically through any number of squares, but cannot move through squares that have been cut out. A rook can attack a square if and only if it can move to that square. After you place these rooks, Justin is supposed to find a square that is not occupied by rooks and cannot be attacked by any rook.
- You want to win this game, that is to say, Justin cannot find any square satisfying these requirements. Now you need to calculate the number of ways to place rooks to ensure you can win. Since the answer may be large, output it modulo **998244353**.

Sample input:

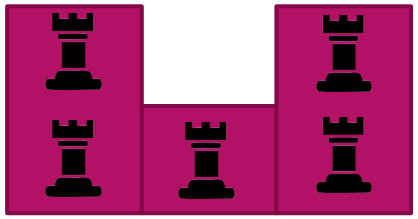
3 → n,n x n chessboard

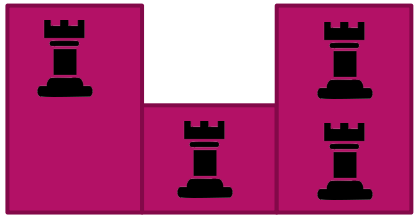
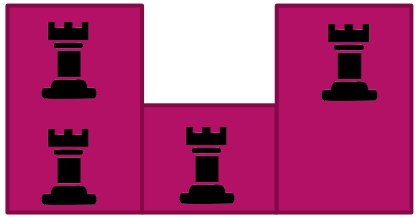
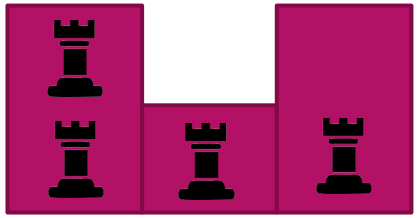
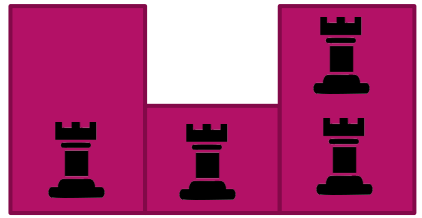
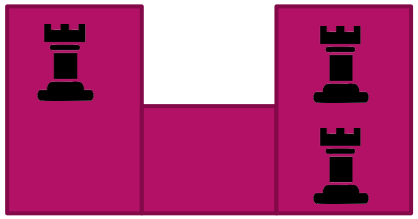
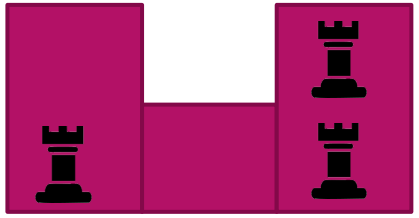
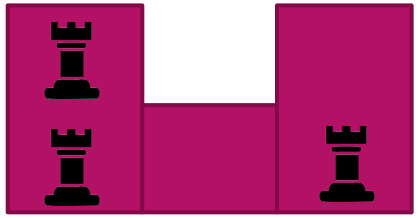
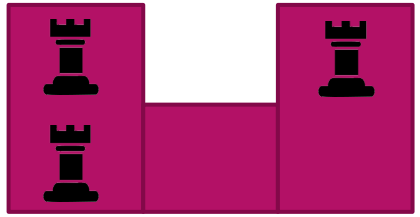
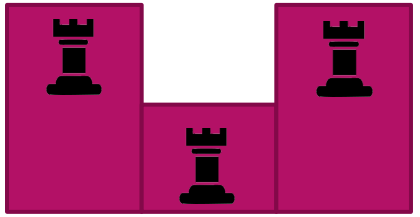
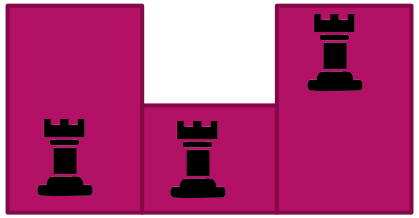
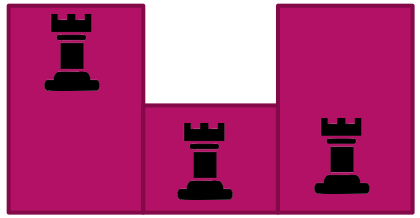
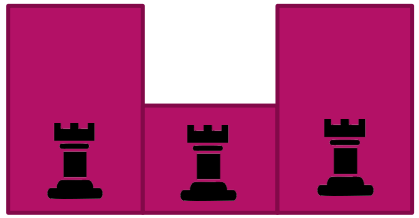
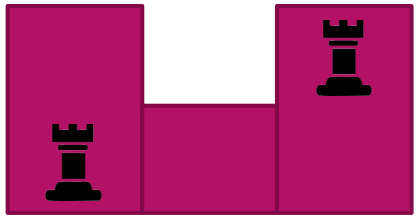
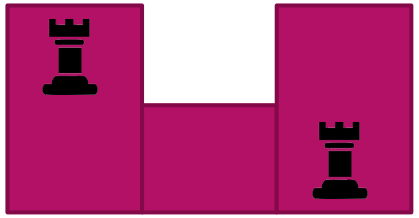
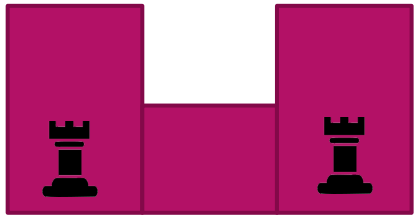
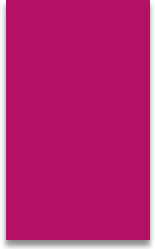
2 1 2

$h_1, h_2, \ldots, h_n$



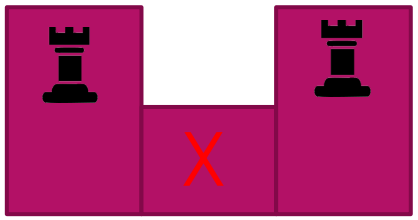
3X3

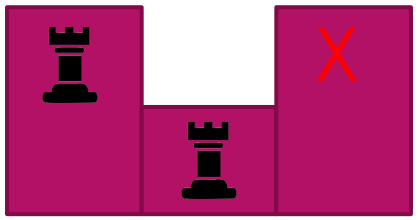


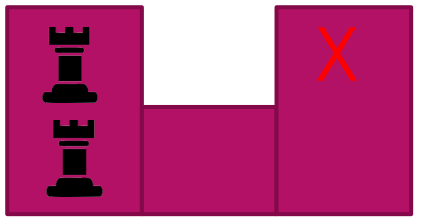

















Sample output:

17

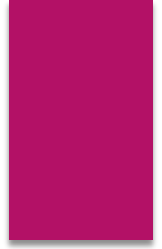

X


X

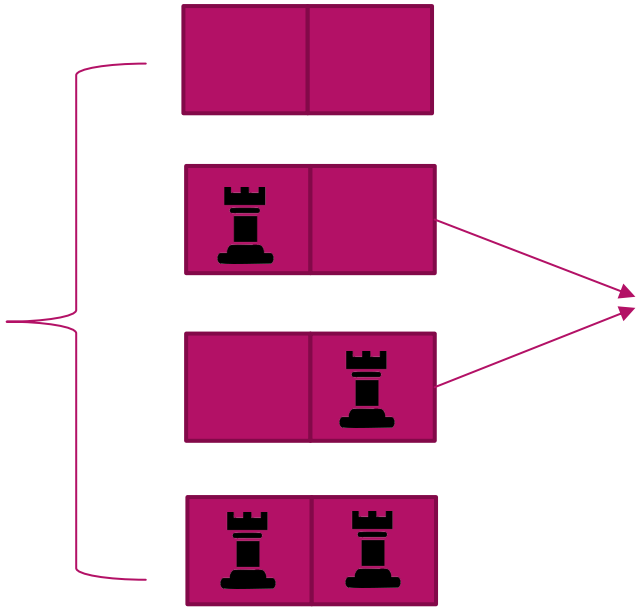

X

the number of ways = 2

the number of ways = 0



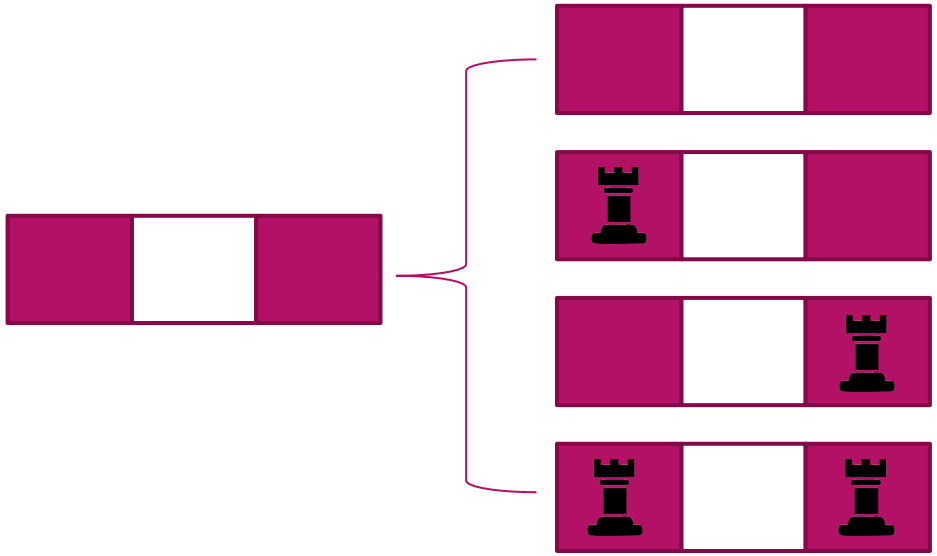
increase size horizontally



no column containing at least 1 rook

1 column containing at least 1 rook

2 column containing at least 1 rook



no column containing at least 1 rook

1 column containing at least 1 rook

2 column containing at least 1 rook

Difference between

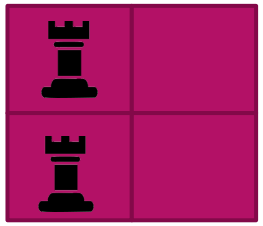
can cover here

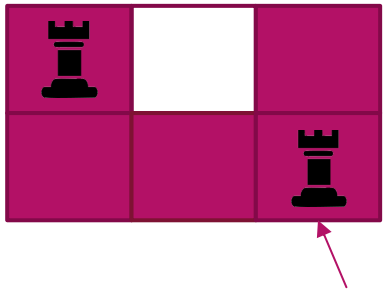
cannot cover here

one subproblem

the combination of 2 subproblem

How to effect the following rows:



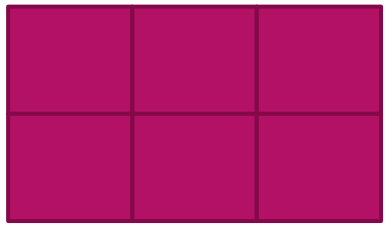


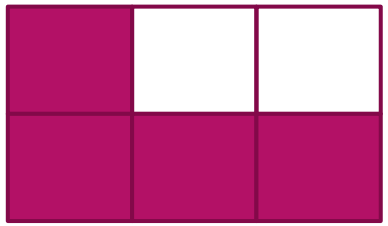
should cover this column

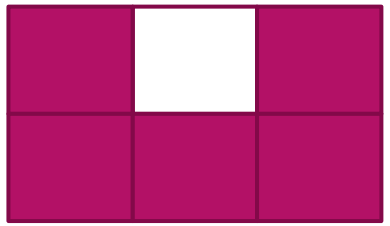
should store the number of columns containing at least 1 rook

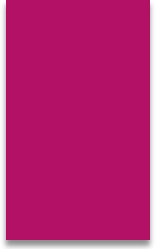
should store the number of columns containing no rook but can be covered by other rooks

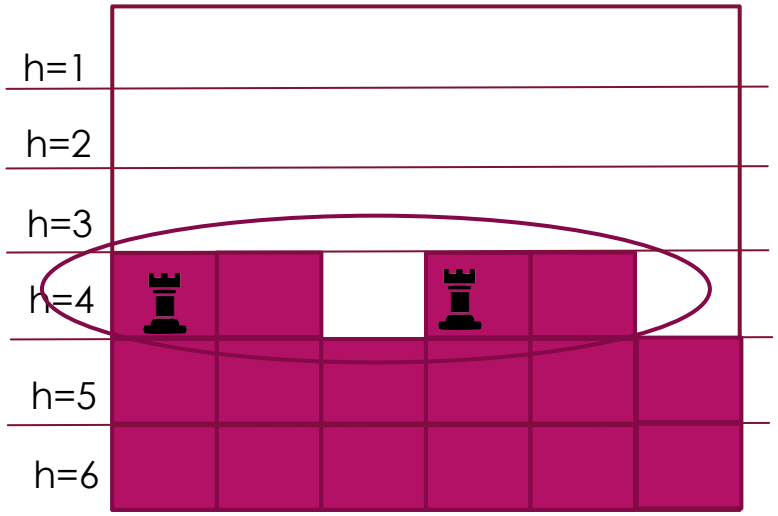
increase size vertically

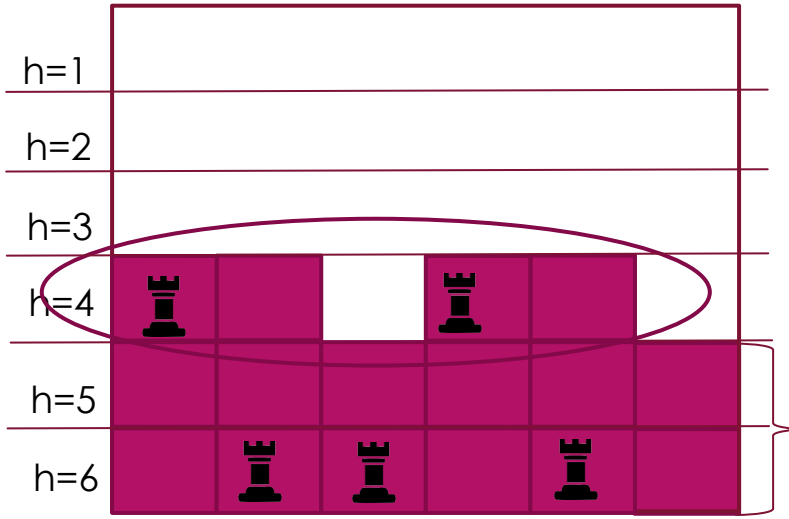












In this case, the rooks on row 6 should be put on Column 2,3,5, which not containing any rooks.

row index

i is the number of columns containing at least 1 rook

dp[l][h][i][0]

prepare for the following rows not always have at least 1 rook

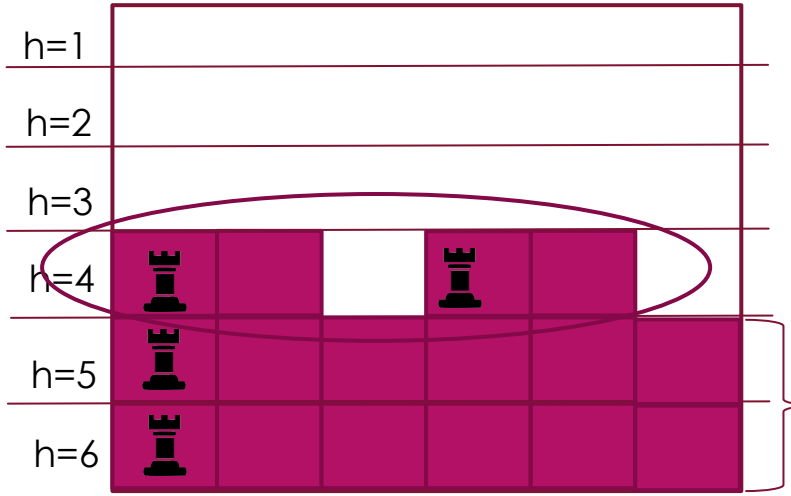
l is the left point of a continuous interval in the h row

dp[l][h][i+k][1]

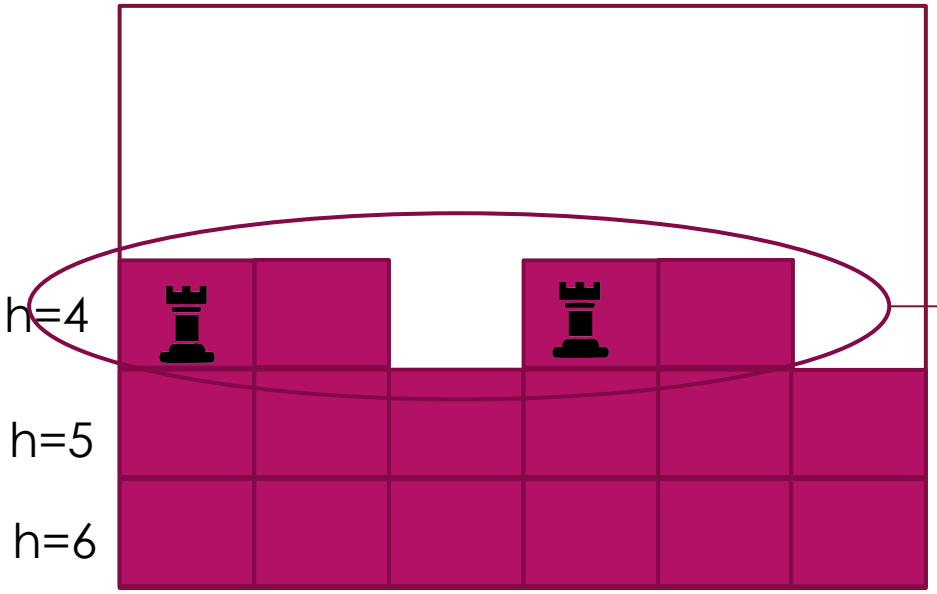
prepare for the following rows all have at least 1 rook

i is the number of columns containing at least 1 rook

k is the number of columns containing no rook but can covered by other rooks

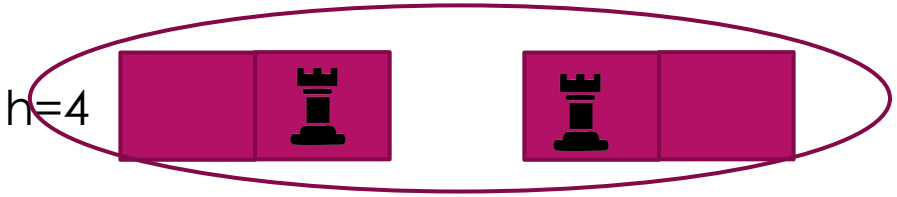


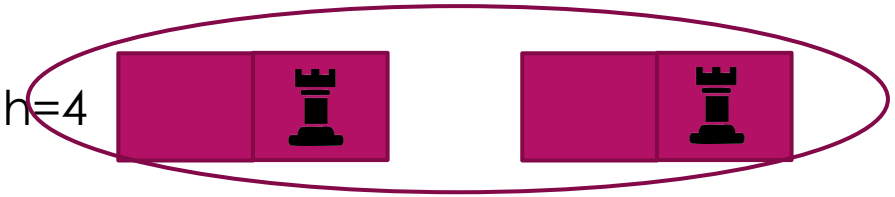
In this case, the rooks on row 6 should be put on any Column not containing rooks or not be covered by other rooks

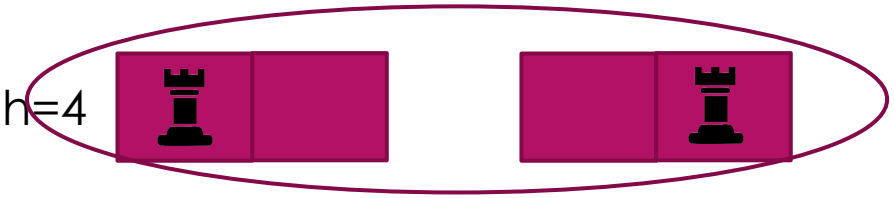


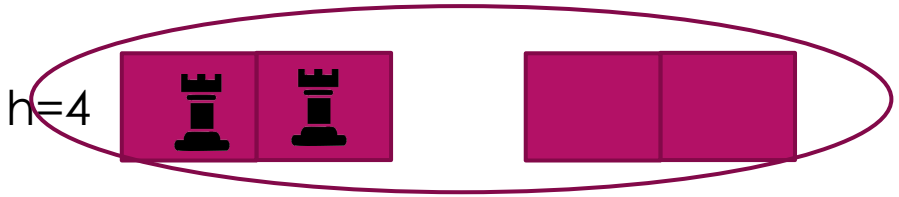
i = 2

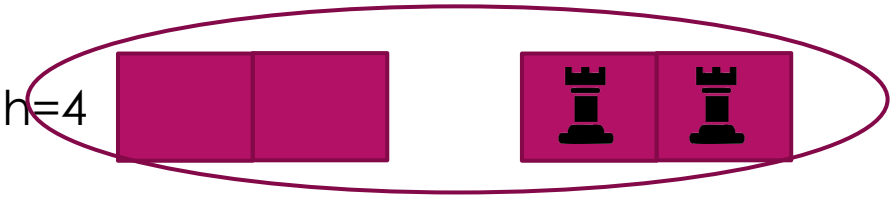
dp[1][4][2][0]=6

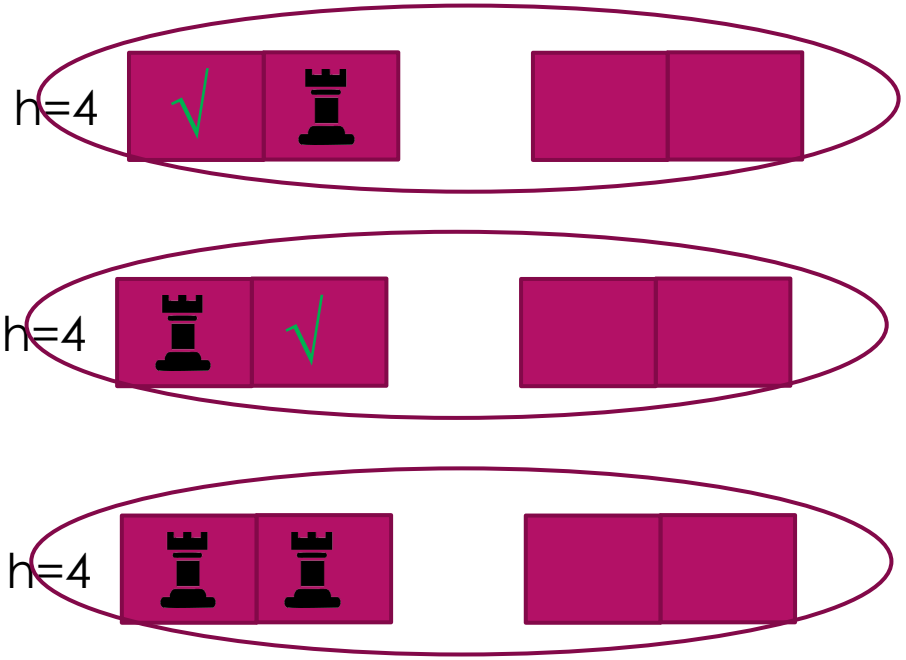






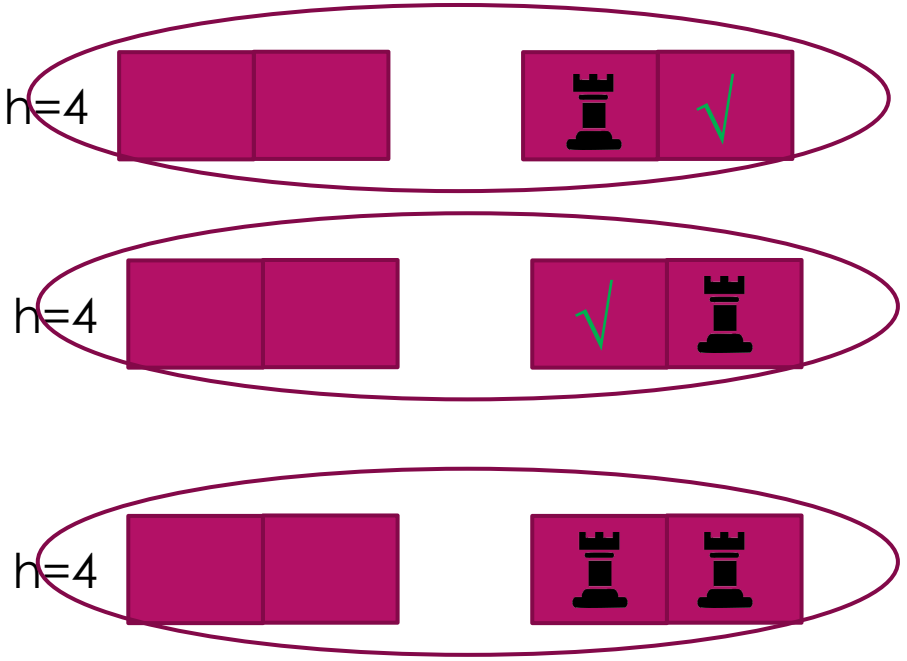


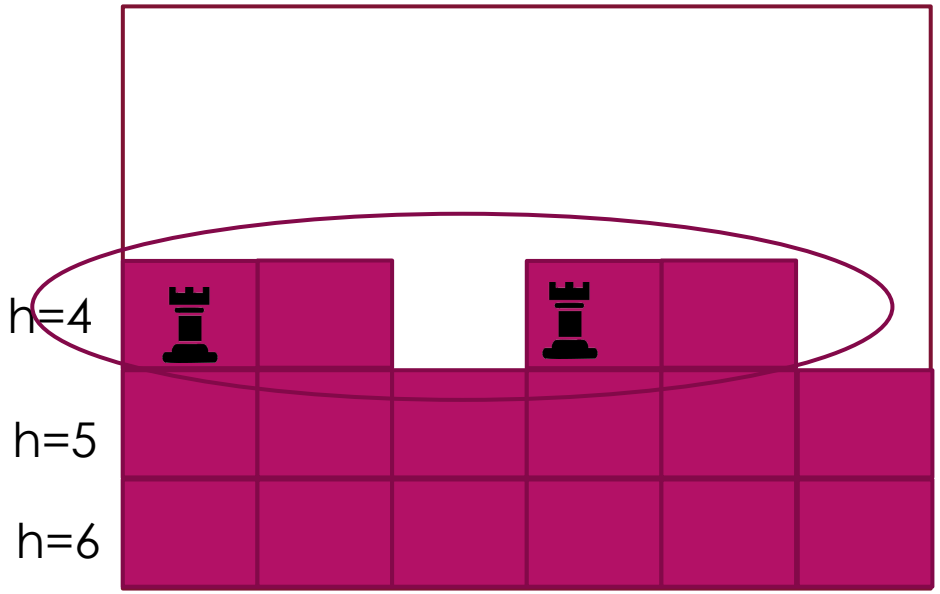




dp[1][4][2][1]=6

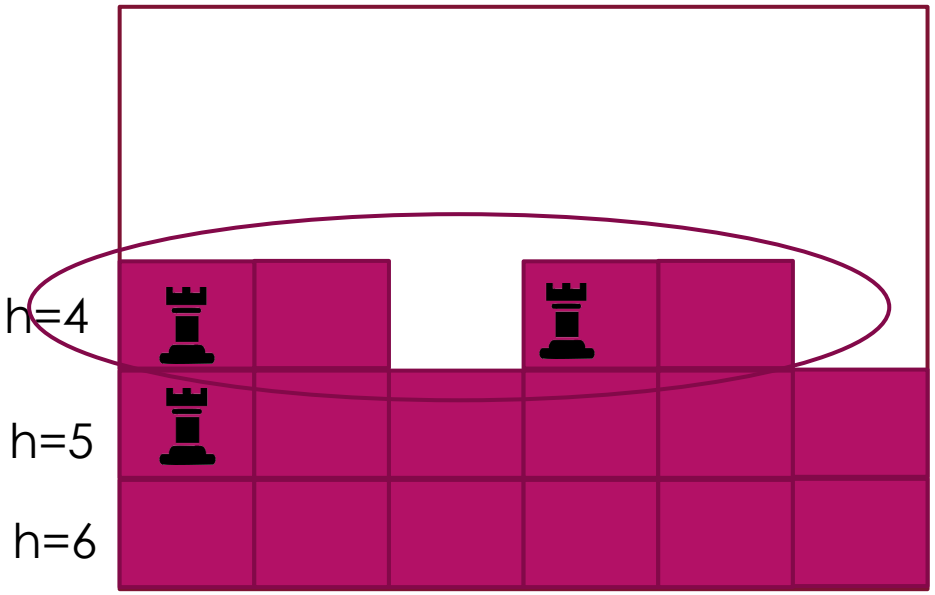
dp[4][4][2][1]=3





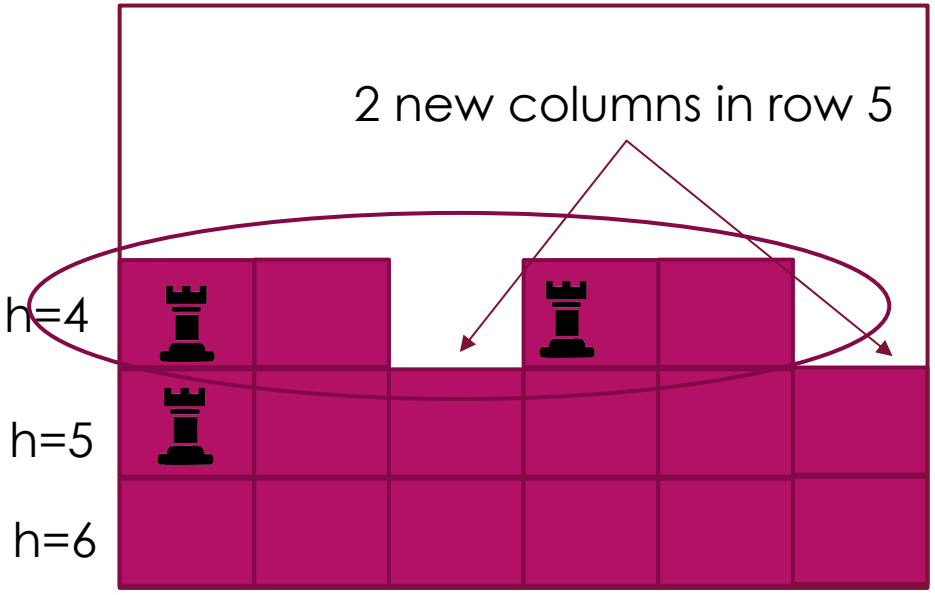
if don't put any rooks on Row 5
dp[1][5][2][0] += dp[1][4][2][0]
dp[1][5][2][1] += dp[1][4][2][0]

Think about why?



if only put rooks on the columns containing rooks before:

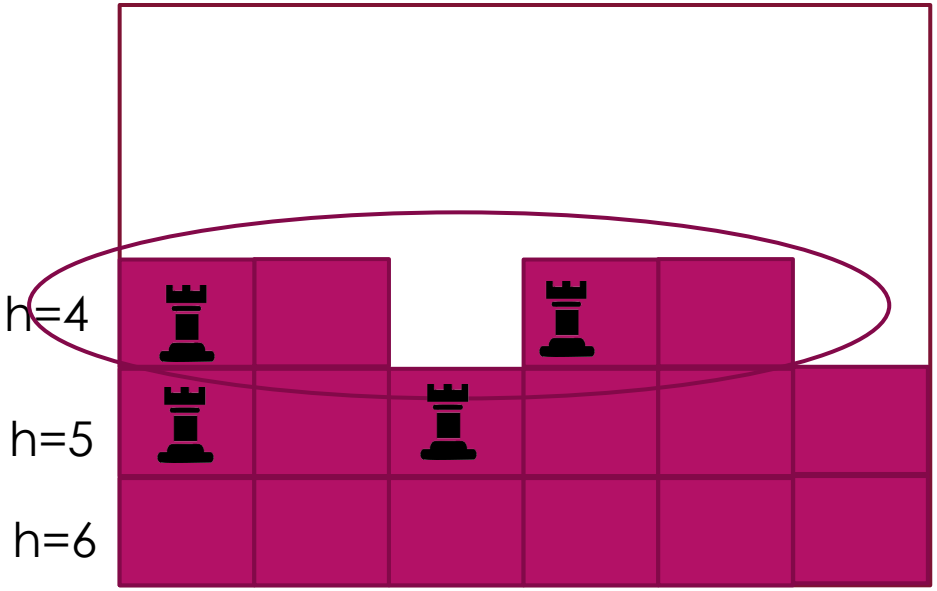
$dp[1][5][2][0] \mathrel{+}= dp[1][4][2][0]*(2^{2}-1)$



if only put rooks on the columns containing rooks or can covered by ~~other rooks~~ before:

$dp[1][5][4+2][1] \mathrel{+}= dp[1][4][4][1]*(2^{4+2}-1)$

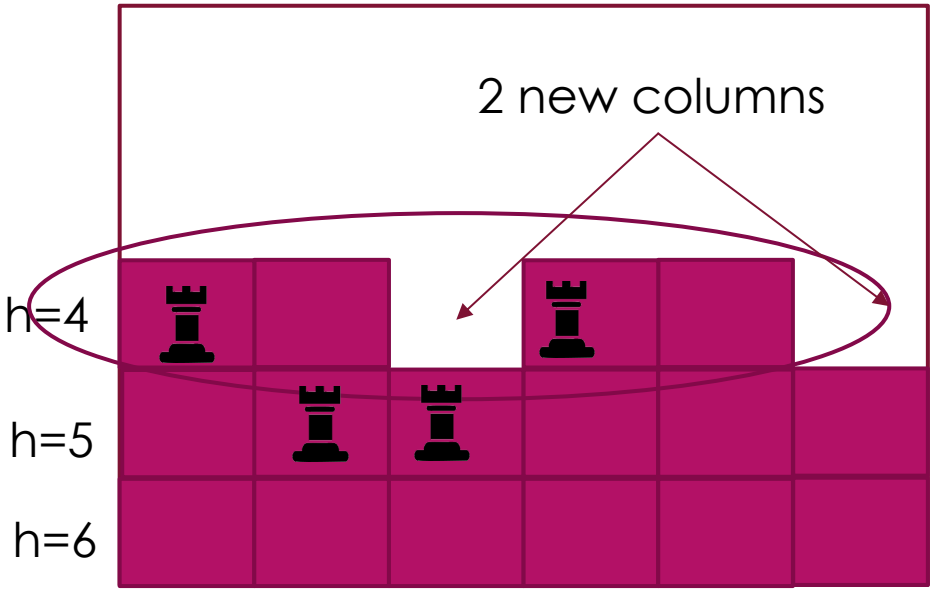
here should +2 since this row has 2 new columns



If put j rooks on the columns not containing rooks before and combine the ways of put rooks on the columns with rooks:

dp[1][5][2+j][0] += dp[1][4][2][0]*C_p^j* 2^2

(assume p is the total number of columns not containing rooks before, here p is 4)



If put k rooks on the columns not containing rooks or can covered by other rooks before and combine the ways of put rooks on the columns with rooks:

dp[1][5][4+2+k][1] += dp[1][4][4][1]* C_q^k * (2^{4+2})

here should +2 since this row has 2 new columns

here should +j

(assume q is the total number of columns not containing rooks before, here q is 0)