Mục lục

1 Bài A - Fibonacci

1.1 Nhận xét

Đây là bài toán cơ bản trong Quy hoạch động để tính số Fibonacci thứ n chia lấy dư cho $10^9 + 7$.

1.2 Định nghĩa DP

Gọi f[i] là giá trị của số thứ fibonacci thứ i chia lấy dư cho $10^9 + 7$.

1.3 Suy ra công thức DP

Dựa vào định nghĩa của đề bài ta có: $f[i] = f[i-1] + f[i-2]$.
Tuy nhiên ta phải thêm phép mod vào công thức để tính kết quả nên công thức DP của ta là:

$$f[i] = (f[i-1] + f[i-2]) \ mod \ (10^9 + 7)$$

1.4 Khởi gán

$f[1] = 1$
$f[2] = 1$

1.5 Đáp án

Đáp án của ta sẽ là $f[n]$

1.6 Code mẫu

```
#include<bits/stdc++.h>
using namespace std;
const int N = 1e6 + 5;
int n , mod = 1e9 + 7;
int f[N];

int main()
{
    cin >> n;
    f[1] = 1;
    f[2] = 1;
    for(int i = 3 ; i <= n ; i++)
        f[i] = (f[i - 1] + f[i - 2]) % mod;
    cout << f[n];
}
```

Đọc code đầy đủ hơn ở *đây*

2 Bài B - Frog 1

2.1 Nhận xét

Đây là 1 bài toán cơ bản trong QHĐ.

2.2 Định nghĩa DP

Gọi DP[i] là chi phí bé nhất để tới được hòn đá thứ i.

2.3 Suy ra công thức DP

Như đề bài cho, con ếch có thể nhảy tới hòn đá $i+1$ và $i+2$ với chi phí lần lượt là $|h_{i+1}-h_i|$ và $|h_{i+2}-h_i|$.
Từ dữ liệu trên, ta đảo ngược lại góc nhìn và thấy, hòn đá thứ i có thể nhảy đến được từ hòn đá thứ $i-1$ và $i-2$ với chi phí lần lượt là $|h_i-h_{i-1}|$ và $|h_i-h_{i-2}|$.
Vậy ta sẽ có **công thức quy hoạch động** như sau:

$$DP[i] = min(DP[i-1]+abs(h_{i-1}-h[i])\ ,\ DP[i-2]+abs(h_{i-2}-h[i]))$$

2.4 Khởi gán

$DP[1] = 0$
$DP[2] = |h_2-h_1|$

2.5 Đáp án

Kết quả của ta sẽ là $DP[n]$ với ý nghĩa là chi phí nhỏ nhất để đến hòn đá thứ n.

2.6 Code mẫu

```
#include<bits/stdc++.h>
using namespace std;

const int N = 1e5 + 5;
int n;
int dp[N] , h[N];

int main()
{
    ios::sync_with_stdio(0);
    cin.tie(0);
```

```
    cin >> n;
    for(int i = 1 ; i <= n ; i++)
        cin >> h[i];
    dp[1] = 0;
    dp[2] = abs(h[2] - h[1]);
    for(int i = 3 ; i <= n ; i++)
        dp[i] = min(dp[i - 1] + abs(h[i] - h[i - 1]),
                    dp[i - 2] + abs(h[i] - h[i - 2]));
    cout << dp[n];
}
```

Đọc code đầy đủ hơn ở *đây*

3 Bài C- Mua vé

3.1 Nhận xét

Đây cũng là 1 bài toán QHĐ cơ bản.

3.2 Định nghĩa DP

Gọi $DP[i]$ là thời gian phục vụ ít nhất cho tới khi người thứ i rời khỏi hàng.

3.3 Suy ra công thức DP

Như đề bài cho, người thứ i có thể mua vé và người thứ $i+1$ có thể rời khỏi hàng và nhờ người thứ i mua.
Với cách nhìn ngược lại, ta có các cách chuyển trạng thái như sau:

1. Người thứ i mua vé
   $\Rightarrow DP[i] = DP[i-1] + t[i]$
2. Người thứ i rời khỏi hàng và nhờ người thứ $i-1$ mua vé cho cả 2
   $\Rightarrow DP[i] = DP[i-2] + r[i-1]$

Từ đó ta suy ra **công thức QHĐ:**

$$DP[i] = min(DP[i-1] + t[i], DP[i-2] + r[i-1])$$

3.4 Khởi gán

$DP[1] = t[1]$
$DP[2] = min(t[1] + t[2], r[1])$

3.5 Đáp án

Đáp án của ta là $DP[n]$ với định nghĩa là thời gian tối thiểu khi người thứ n rời khỏi hàng.

3.6 Code mẫu

```
#include<bits/stdc++.h>
using namespace std;

const int N = 1e5 + 5;
int n;
int t[N] , r[N] , dp[N];

int main()
{
    ios::sync_with_stdio(0);
    cin.tie(0);
    cin >> n;
    for(int i = 1 ; i <= n ; i++)
        cin >> t[i];
    for(int i = 1 ; i <= n - 1 ; i++)
        cin >> r[i];
    dp[1] = t[1];
    dp[2] = min(t[1] + t[2] , r[1]);
    for(int i = 3 ; i <= n ; i++)
    {
        dp[i] = min(r[i - 1] + dp[i - 2],
                    t[i] + dp[i - 1]);
    }
    cout << dp[n];
}
```

Đọc code đầy đủ hơn ở *đây*

4 Bài D - Bậc Thang

4.1 Nhận xét

Bài toán này yêu cầu tìm số cách để đi hết bậc thang sau khi chia lấy phần dư cho 14062008.

4.2 Định nghĩa DP

Gọi DP[i] là số cách để đi đến bậc thang thứ i.

4.3 Suy ra công thức DP

Như đề bài cho, trong 1 lần nhảy ta có thể nhảy lên bậc thứ $i+1$ hoặc bậc thứ $i+2$. Nhìn ngược lại, bậc thang thứ i có thể nhảy lên được từ bậc thứ $i-1$ và $i-2$.

Từ đó ta suy ra **công thức QHĐ** như sau:

$$DP[i] = (DP[i-1] + DP[i-2]) \ mod \ 14062008$$

.

Lưu ý: Vì ta có những bậc thang bị hỏng nên với x là 1 bậc thang bị hỏng, ta luôn có $DP[x] = 0$.

4.4 Khởi gán

$DP[0] = 0$
$DP[1] = 1$

4.5 Đáp án

Đáp án của ta là $DP[n]$ với định nghĩa là số cách đi tới bậc thang thứ n.

4.6 Code mẫu

```
#include<bits/stdc++.h>
using namespace std;
const int MAXN = 1e5 + 5;
const int MOD = 14062008;
int n , k;
bool biHong[MAXN];
int dp[MAXN];

int main(){
ios_base::sync_with_stdio(0);cin.tie(0);cout.tie(0);
    cin >> n >> k;
    for(int i = 1 ; i <= k ; i++){
        int x;
        cin >> x;
        biHong[x] = true;
    }
    dp[0] = 0;
    dp[1] = 1;
    for(int i = 2 ; i <= n ; i++){
        if(biHong[i])
            continue;
        dp[i] = (dp[i - 1] + dp[i - 2]) % MOD;
    }
    cout << dp[n] << endl;
    return 0;
}
```

Đọc code đầy đủ hơn ở *đây*

5 Bài E - Frog 2

5.1 Nhận xét

Bài này có cách suy luận y hệt với bài B - Frog 1.

5.2 Định nghĩa DP

Gọi $DP[i]$ là chi phí nhỏ nhất để nhảy tới hòn đá thứ i.

5.3 Suy ra công thức DP

Tương tự như bài Frog 1, ta đảo ngược góc nhìn và thấy từ hòn đá $i-1, i-2, ..., i-k$ đều có thể nhảy lên hòn đá thứ i.
Từ đó ta suy ra **công thức QHĐ** như sau:

$$DP[i] = min(DP[j] + |h[i] - h[j]|), 1 \leq i-k \leq j < i \leq n$$

5.4 Khởi gán

$DP[1] = 0$

5.5 Đáp án

Đáp án của ta là $DP[n]$ với định nghĩa là chi phí nhỏ nhất để tới hòn đá thứ n.

5.6 Code mẫu

```
#include<bits/stdc++.h>
using namespace std;

const int MAXN = 1e5 + 5;
int n , k;
int dp[MAXN] , h[MAXN];

int main(){
    ios_base::sync_with_stdio(0);cin.tie(0);cout.tie(0);
    cin >> n >> k;
    for(int i = 1 ; i <= n ; i++)
        cin >> h[i];

    dp[1] = 0;
```

```
    for(int i = 2 ; i <= n ; i++){
        dp[i] = 1e9;
        for(int j = i - 1 ; j >= max(1 , i - k) ; j--)
            dp[i] = min(dp[i] , dp[j] + abs(h[i] - h[j]));
    }
    cout << dp[n] << endl;
    return 0;
}
```

Đọc code đầy đủ hơn ở *đây*

6 Bài F - Lát gạch 1

6.1 Nhận xét

Đây là bài toán cơ bản giúp hỗ trợ tư duy trong QHĐ.

6.2 Định nghĩa DP

Gọi $DP[i]$ là số cách lát được gạch trong hình chữ nhật với kích thước là $2xi$ chia dư cho 10^9+7.

6.3 Suy ra công thức DP

Từ việc có 2 loại gạch là $1x2$ và $2x1$, ta có thể suy ra có thể thêm 1 lần các viên gạch nhỏ nhất vào vị trí i sao cho không có lỗ hổng như sau:

+) 1 viên $2x1$

+) 2 viên $1x2$ tạo thành 1 khối $2x2$

Bằng cách viên gạch trên, ta có cách chuyển trạng thái như sau:

+) 1 viên $2x1 \Rightarrow DP[i] += DP[i-1]$

+) 2 viên $1x2$ tạo thành 1 khối $2x2 \Rightarrow DP[i] += DP[i-2]$

Từ việc thêm cách viên gạch như trên ta có thể suy ra **công thức QHĐ:**

$$DP[i] = (DP[i-1] + DP[i-2]) \ mod \ (10^9+7)$$

6.4 Khởi gán

$DP[1] = 1$

$DP[2] = 2$

6.5 Đáp án

với mỗi số N đọc từ input, in ra $DP[N]$ với định nghĩa là số cách để lát hình chữ nhật có kích thước $2xN$ chia dư cho 10^9+7

6.6 Code mẫu

```
#include<bits/stdc++.h>
using namespace std;

const int N = 1e6 + 5;
int test, mod = 1e9 + 7;
int dp[N];

int main()
{
    ios::sync_with_stdio(0);
    cin.tie(0);
    dp[1] = 1;
    dp[2] = 2;
    for(int i = 3 ; i <= 1e6 ; i++)
        dp[i] = (dp[i - 1] + dp[i - 2]) % mod;
    cin >> test;
    for(int i = 1 ; i <= test ; i++)
    {
        int n;
        cin >> n;
        cout << dp[n] << "\n";
    }
}
```

Đọc code đầy đủ hơn ở *đây*

7 Bài G - LIQ

7.1 Nhận xét

Đây là 1 trong những bài toán kinh điển của phần quy hoạch động.

7.2 Định nghĩa DP

Gọi $DP[i]$ là dãy con tăng dài nhất xét tới vị trí i và chắc chắc có lấy i.

7.3 Suy ra công thức DP

Xét 1 dãy con tăng bất kì, nếu muốn thêm phần tử i đang xét vào dãy con tăng đấy thì phần tử i thêm vào sẽ phải lớn hơn phần tử cuối cùng của chính dãy đang xét. Nên ta sẽ thử xét mọi dãy con tăng kết thúc tại $j(j < i)$ thỏa tính chất trên và tìm dãy có độ dài lớn nhất để ghép i vào.

Khi đấy ta có được công thức QHĐ như sau:

$$DP[i] = max(DP[j]) + 1 \quad \forall\ j < i,\ \ a[j] < a[i]$$

Lưu ý: Với công thức này ta chỉ có thể giải bài toán dãy con tăng dài nhất với **N <=** 10^4.

7.4 Khởi gán

$DP[i] = 1\ \forall\ i \leq n$

7.5 Đáp án

Đáp án của ta sẽ là phần tử $DP[i]$ lớn nhất trong mảng DP, với định nghĩa là dãy con tăng dài nhất kết thúc tại vị trí bất kì.

7.6 Code mẫu

```
#include<bits/stdc++.h>
using namespace std;

const int MAXN = 1e3 + 5;
int n;
int dp[MAXN] , a[MAXN];

int main(){
    ios_base::sync_with_stdio(0);cin.tie(0);cout.tie(0);
    cin >> n;
    for(int i = 1 ; i <= n ; i++)
        cin >> a[i];
    int res = 0;
    for(int i = 1 ; i <= n ; i++){
        dp[i] = 1;
        for(int j = 1 ; j < i ; j++)
            if(a[i] > a[j]) dp[i] = max(dp[i] , dp[j] + 1);
        res = max(res , dp[i]);
    }
    cout << res << endl;
    return 0;
}
```

Đọc code đầy đủ hơn ở *đây*

8 Bài H - Dãy con chính phương

8.1 Nhận xét

Đây là 1 bài toán giống với bài dãy con tăng dài nhất, nhưng nếu nhìn sơ qua và không đọc kĩ điều kiện thì có thể nghĩ rằng:
"Độ phức tạp của bài là O(N^2) sao?"
Nhưng không, quan sát kĩ điều kiện trong đề bài thì ta thấy điều kiện $|i-j| \leq 10$ nên độ phức tạp tổng quát của ta là O($10 * N$)

8.2 Định nghĩa DP

Gọi $DP[i]$ là dãy con thỏa mãn dài nhất kết với i vị trí đầu và chắc chắn có lấy vị trí i.

8.3 Suy ra công thức DP

Xét 1 dãy con thỏa mãn, để thêm phần tử i đang xét ta chỉ cần xét tính thỏa phần tử i và phần tử cuối cùng của dãy. Nên ta sẽ thử xét tối đa 10 dãy con $j(j < i, i-j \leq 10)$ thỏa tính chất trên và tìm dãy có độ dài lớn nhất để ghép i vào.
Dưa vào tính chất trên, ta có **công thức QHĐ** như sau:

$DP[i] = DP[j] + 1 \ \ \forall \ \ j < i, \ \ i - j \leq 10, \ \ |a[j] - a[i]| > 0, \ \ |a[j] - a[i]|$ là số chính phương.

8.4 Khởi gán

$DP[i] = 1 \ \forall \ i \leq n$

8.5 Đáp án

Đáp án của ta sẽ là phần tử DP[i] lớn nhất trong mảng DP, với định nghĩa là dãy con thỏa mãn dài nhất kết thúc tại vị trí bất kì.

8.6 Code mẫu

```
#include<bits/stdc++.h>
using namespace std;

const int MAXN = 1e5 + 5;
int n;
int dp[MAXN] , a[MAXN];

bool soChinhPhuong(int x){
    if(x == 0) return false;
    int y = sqrt(x);
    return y * y == x;
```

```
}

int main(){
    ios_base::sync_with_stdio(0);cin.tie(0);cout.tie(0);
    cin >> n;
    for(int i = 1 ; i <= n ; i++)
        cin >> a[i];
    int res = 0;
    for(int i = 1 ; i <= n ; i++){
        dp[i] = 1;
        for(int j = max(1 , i - 10) ; j < i ; j++)
            if( soChinhPhuong(abs(a[j] - a[i])) )dp[i] = max(dp[i] , dp[j] + 1);
        res = max(res , dp[i]);
    }
    cout << res << endl;
    return 0;
}
```

Đọc code đầy đủ hơn ở *đây*

9 Bài I - Lát gạch 2

9.1 Nhận xét

Đây là 1 bài toán DP yêu cầu người làm phải suy nghĩ về cách chuyển trạng thái.

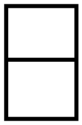
9.2 Định nghĩa DP

Gọi $DP[i]$ là số cách để lát hết một hình chữ nhật $2xi$ chia lấy dư cho 10^9+7.

9.3 Suy ra công thức DP

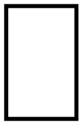
Đầu tiên, ta xét đến các cách để lát thêm vào duy nhất 1 cột ở cuối:

+) 2 khối $1x1$ ghép thành 1 khối $2x1$. Ảnh minh họa:



$$\Rightarrow DP[i] += DP[i-1]$$

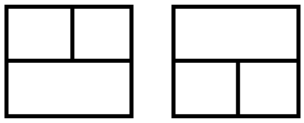
+) 1 khối $2x1$.Ảnh minh họa:



$$\Rightarrow DP[i] += DP[i-1]$$

Tiếp theo, ta xét đến các cách lát thêm vào cột ở cuối sao cho không bị trùng trường hợp với các cách lát thêm vào 1 cột:

+) 2 khối $1x1$ và 1 khối $1x2$ ghép thành 1 khối $2x2$. (ở đây có 2 cách).Ảnh minh họa:



$\Rightarrow DP[i] += 2 * DP[i-2]$

+) 2 khối $1x2$ ghép thành 1 khối $2x2$.Ảnh minh họa:

$\Rightarrow DP[i] += DP[i-2]$

Vậy cuối cùng, **công thức QHĐ** của ta là:

$$DP[i] = (2 * DP[i-1] + 3 * DP[i-2]) \mod (10^9 + 7)$$

9.4 Khởi gán

$DP[1] = 2$
$DP[2] = 7$

9.5 Đáp án

Đáp án của ta sẽ là $DP[n]$ với định nghĩa là số cách lát gạch hết hình chữ nhật có kích thước $2xn$.

9.6 Code mẫu

```
#include<bits/stdc++.h>
using namespace std;

const int MOD = 1e9 + 7;
const int MAXN = 1e6 + 5;
int n;
int dp[MAXN];

int main(){
    ios_base::sync_with_stdio(0);cin.tie(0);cout.tie(0);
    cin >> n;
    dp[1] = 2;
    dp[2] = 7;
    for(int i = 3 ; i <= n ; i++)
```

```
        dp[i] = (2LL * dp[i - 1] + 3LL * dp[i - 2]) % MOD;
    cout << dp[n] << endl;
    return 0;
}
```

Đọc code đầy đủ hơn ở *đây*

10 Bài J - Lát gạch 3

10.1 Nhận xét

Bài lát gạch 3 về mặt tư tưởng cũng sẽ giống như bài lát gạch 2 là suy ra công thức QHĐ từ các cách lát thêm.

10.2 Định nghĩa DP

Gọi $DP[i]$ là số cách lát hình chữ nhật có kích thước $2xi$.

10.3 Suy ra công thức DP

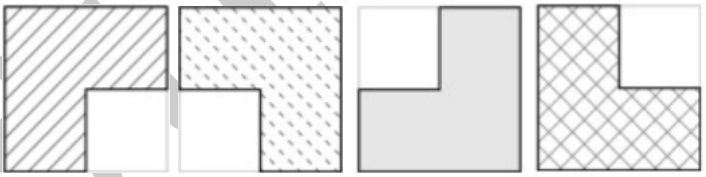
Cho rằng trạng thái i là số cột hiện tại bằng i, ta liệt kê ra các cách chuyển trạng thái:

+) Không thêm gạch vào cột thứ i, chiếm 1 cột

$\Rightarrow$ chuyển sang trạng thái $i+1$,

$\Rightarrow DP[i] += DP[i-1]$.

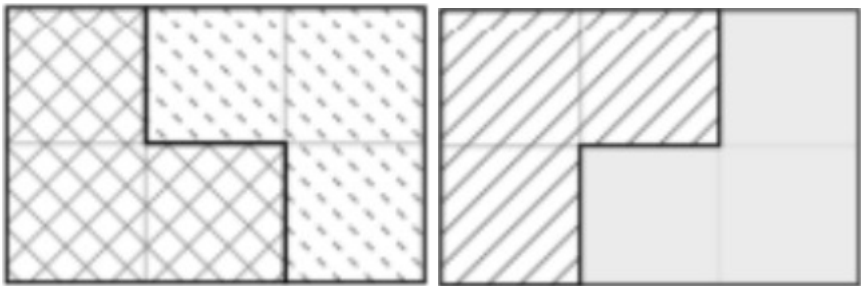
+) Thêm vào 1 hình chữ L, chiếm 2 cột, có 4 cách như sau:



$\Rightarrow$ chuyển sang trạng thái $i+2$.

$\Rightarrow DP[i] += 4 * DP[i-2]$.

+) Thêm vào 2 hình chữ L kết hợp lại thành 1 khối $2x3$, chiếm 3 cột, có 2 cách như sau:



$\Rightarrow$ chuyển sang trạng thái $i+3$.

$\Rightarrow DP[i] += 4 * DP[i-2]$.

Vậy **công thức QHĐ** của ta là:

$$DP[i] = (DP[i-1] + 4 * DP[i-2] + 2 * Dp[i-3]) \quad mod \ (10^9 + 7)$$

10.4 Khởi gán

$DP[1] = 1$
$DP[2] = 5$
$DP[3] = 11$

10.5 Đáp án

Đáp án của ta sẽ là $DP[n]$ với định nghĩa là số cách lát hình chữ nhật với kích thước $2xn$.

10.6 Code mẫu

```
#include<bits/stdc++.h>
using namespace std;

const int MOD = 998244353;
const int MAXN = 1e6 + 5;
int n;
int dp[MAXN];

int main(){
    ios_base::sync_with_stdio(0);cin.tie(0);cout.tie(0);
    cin >> n;
    dp[0] = 1;
    dp[1] = 1;
    dp[2] = 5;
    for(int i = 3 ; i <= n ; i++)
        dp[i] = (dp[i - 1] + dp[i - 2] * 4LL + dp[i - 3] * 2LL) % MOD;

    cout << dp[n] << endl;
    return 0;
}
```

Đọc code đầy đủ hơn ở *đây*

11 Bài K - Đoạn con liên tiếp có tổng lớn nhất

11.1 Nhận xét

Đây là bài toán kinh điển và có thể sử dụng QHĐ để giải. Ngoài ra còn một số cách giải khác như sử dụng thuật toán Kadane, Prefix Sum,....

11.2 Định nghĩa DP

Gọi $DP[i]$ là đoạn con có tổng lớn nhất kết thúc tại i.

11.3 Suy ra công thức DP

Xét về mặt bản chất, vì là đoạn con kết thúc tại vị trí i nên $DP[i]$ chỉ có 2 trường hợp:

+) $DP[i] = a[i]$ - nghĩa là chỉ lấy duy nhất phần tử $a[i]$.

+) $DP[i] = DP[i-1] + a[i]$ - thêm $a[i]$ vào cuối đoạn con có tổng lớn nhất kết thúc tại $i-1$.

Vậy công thức QHĐ của ta là: $DP[i] = max(a[i], DP[i-1] + a[i])$

11.4 Khởi gán

$DP[0] = 0$

11.5 Đáp án

Đáp án của ta là $max(DP[i]) \forall \ \ i \leq n$ với ý nghĩa là đoạn con có tổng lớn nhất kết thúc tại bất kì vị trí nào.

11.6 Code mẫu

```
#include<bits/stdc++.h>
using namespace std;

const int MAXN = 1e6 + 5;
int n;
long long dp[MAXN] , a[MAXN];

int main(){
    ios_base::sync_with_stdio(0);cin.tie(0);cout.tie(0);
    cin >> n;
    for(int i = 1 ; i <= n ; i++)
        cin >> a[i];

    long long res = 0;
    dp[0] = 0;
    for(int i = 1 ; i <= n ; i++){
        dp[i] = max(a[i] , dp[i - 1] + a[i]);
        res = max(res , dp[i]);
    }
    cout << res << endl;
    return 0;
}
```

Đọc code đầy đủ hơn ở *đây*

12 Bài L - Dãy con liên tiếp có tổng lớn nhất 2

12.1 Nhận xét

Bài toán nào tương tự với bài toán dãy con liên tiếp có tổng lớn nhất nhưng đoạn con phải có độ dài $\geq k$.

Với bài toán này ta sẽ sử dụng thêm 1 mảng $b[i]$ mang ý nghĩa là tổng các số trong đoạn $[i; i+k-1]$.

12.2 Định nghĩa DP

Gọi $DP[i]$ là tổng của đoạn con có tổng lớn nhất có phần tử cuối cùng là i.

12.3 Suy ra công thức DP

Gọi $DP[i]$ là đoạn con có tổng lớn nhất kết thúc tại i.

12.4 Suy ra công thức DP

Xét về mặt bản chất, vì là đoạn con kết thúc tại vị trí i nên $DP[i]$ chỉ có 2 trường hợp:

+) $DP[i] = a[i]$ - nghĩa là chỉ lấy duy nhất phần tử $a[i]$.

+) $DP[i] = DP[i-1] + a[i]$ - thêm $a[i]$ vào cuối đoạn con có tổng lớn nhất kết thúc tại $i-1$.

Vậy công thức QHĐ của ta là: $DP[i] = max(a[i], DP[i-1] + a[i])$

12.5 Khởi gán

$DP[0] = 0$

$b[i] = a[i] + a[i+1] + ..., a[i+k-1] \ \forall \ \ i \leq n-k+1$

12.6 Đáp án

Phần DP trên khá tương đồng với bài toán tìm đoạn con có tổng lớn nhất. Phần thú vị nhất của bài toán bắt đầu từ đây.

Ta xét với mỗi $b[i]$, sau đấy mình cộng thêm đoạn con có tổng lớn nhất kết thúc tại $i-1$. Thì ta sẽ có được đoạn con có độ dài $\geq k$ và có tổng lớn nhất kết thúc tại $i+k-1$.

Giải thích cho phần trên, việc ta lấy $b[i]$ nghĩa là đang cố định lấy toàn bộ đoạn con $[i; i+k-1]$ và sau đấy lấy thêm DP[i - 1] nghĩa là lấy thêm đoạn con có tổng lớn nhất kết thúc tại $i-1$.

Tóm lại, đáp án của ta sẽ là:

$$max(\ max(b[i]+DP[i-1],\ b[i])\ )\ \ \forall\ i \le n-k+1$$

12.7 Code mẫu

```
#include<bits/stdc++.h>
using namespace std;

const int MAXN = 1e6 + 5;
int n , k;
long long dp[MAXN] , b[MAXN] , a[MAXN];

int main(){
    ios_base::sync_with_stdio(0);cin.tie(0);cout.tie(0);
    cin >> n >> k;
    for(int i = 1 ; i <= n ; i++)
        cin >> a[i];

    long long cur = 0;
    for(int i = 1 ; i <= n ; i++){
        cur += a[i];
        if(i >= k){
            b[i - k + 1] = cur;
            cur -= a[i - k + 1];
        }
    }
    for(int i = 1 ; i <= n ; i++)
        dp[i] = max(a[i] , dp[i - 1] + a[i]);

    long long res = LLONG_MIN;
    for(int i = 1 ; i <= n - k + 1 ; i++)
        res = max(res , max(b[i] , b[i] + dp[i - 1]));
    cout << res << endl;

    return 0;
}
```

Đọc code đầy đủ hơn ở *đây*

13 Bài M - Hội trường 1

13.1 Nhận xét

Yêu cầu của bài toán là chọn các các yêu cầu sao cho thời gian không giao nhau và thời gian được sử dụng là nhiều nhất.

13.2 Định nghĩa DP

Gọi $DP[i]$ là tổng thời gian chọn nhiều nhất của các yêu cầu sao cho thời gian kết thúc của tất cả các yêu cầu được chọn đều $\leq i$

13.3 Suy ra công thức DP

Xét 1 yêu cầu có dạng $[L;R]$, giả sử yêu cầu này có thời gian kết thúc lớn nhất so với các yêu cầu đã chọn thì những yêu cầu trước buộc phải có thời gian kết thúc $\leq L$.

Từ nhận xét trên, ta thấy với 1 yêu cầu $[L;R]$, $DP[R]$ sẽ được cập nhật là $DP[R] = max(DP[L] + (R - L))$.

Vậy ta có **công thức QHĐ** như sau:

$$DP[R] = max(DP[L] + (R - L)) \quad \forall\ R \leq 10^5 \quad \& \quad \forall\ L \text{ tồn tại yêu cầu [L ; R]}$$

13.4 Khởi gán

DP[0] = 0;

13.5 Đáp án

Đáp án của ta là $DP[10^5]$ với ý nghĩa là tổng thời gian nhiều nhất có thể với các yêu cầu được chọn có thời gian kết thúc $\leq 10^5$.

13.6 Code mẫu

```
#include<bits/stdc++.h>
using namespace std;

const int MAXN = 1e6 + 5;
int n;
long long DP[MAXN];
vector<int> request[MAXN];

int main(){
    ios_base::sync_with_stdio(0);cin.tie(0);cout.tie(0);
```

```
    cin >> n;
    for(int i = 1 ; i <= n ; i++){
        int u , v;
        cin >> u >> v;
        request[v].push_back(u);
    }

    DP[0] = 0;
    for(int R = 1 ; R <= 100000 ; R++){
        DP[R] = DP[R - 1];
        for(int j = 0 ; j < request[R].size() ; j++){
            int L = request[R][j];
            DP[R] = max(DP[R] , DP[L] + (R - L));
        }
    }

    cout << DP[100000] << endl;
    return 0;
}
```

Đọc code đầy đủ hơn ở *đây*

14 Bài N - Hội trường 2

14.1 Nhận xét

Yêu cầu của bài toán là chọn các các yêu cầu sao cho thời gian không giao nhau và số yêu cầu được chọn là nhiều nhất.

14.2 Định nghĩa DP

Gọi $DP[i]$ là số yêu cầu nhiều nhất chọn được sao cho thời gian kết thúc của tất cả các yêu cầu được chọn đều $\leq i$

14.3 Suy ra công thức DP

Xét 1 yêu cầu có dạng $[L;R]$, giả sử yêu cầu này có thời gian kết thúc lớn nhất so với các yêu cầu đã chọn thì những yêu cầu trước buộc phải có thời gian kết thúc $\leq L$.

Từ nhận xét trên, ta thấy với 1 yêu cầu $[L;R]$, $DP[R]$ sẽ được cập nhật là $DP[R] = max(DP[L]+1)$.

Vậy ta có **công thức QHĐ** như sau:

$$DP[R] = max(DP[L]+1) \quad \forall\, R \leq 10^5 \quad \& \quad \forall\, L \text{ tồn tại yêu cầu [L ; R]}$$

14.4 Khởi gán

DP[0] = 0;

14.5 Đáp án

Đáp án của ta là $DP[10^5]$ với ý nghĩa là số yêu cầu được chọn nhiều nhất có thể với các yêu cầu được chọn có thời gian kết thúc $\leq 10^5$.

14.6 Code mẫu

```
#include<bits/stdc++.h>
using namespace std;

const int MAXN = 1e6 + 5;
int n;
long long DP[MAXN];
vector<int> request[MAXN];

int main(){
    ios_base::sync_with_stdio(0);cin.tie(0);cout.tie(0);
    cin >> n;
    for(int i = 1 ; i <= n ; i++){
        int u , v;
        cin >> u >> v;
        request[v].push_back(u);
    }

    DP[0] = 0;
    for(int R = 1 ; R <= 100000 ; R++){
        DP[R] = DP[R - 1];
        for(int j = 0 ; j < request[R].size() ; j++){
            int L = request[R][j];
            DP[R] = max(DP[R] , DP[L] + (R - L));
        }
    }

    cout << DP[100000] << endl;
    return 0;
}
```

Đọc code đầy đủ hơn ở *đây*

15 Bài O - Hội trường 3

15.1 Nhận xét

Yêu cầu của bài toán là chọn các các đơn đặt hàng sao cho thời gian không giao nhau và tiền thuê nhận được là nhiều nhất.

15.2 Định nghĩa DP

Gọi $DP[i]$ là số tiền thuê nhiều nhất lấy được sao cho thời gian kết thúc của tất cả các đơn đặt hàng được chọn đều $\leq i$

15.3 Suy ra công thức DP

Xét 1 yêu cầu có dạng $[L;R]$, giả sử đơn đặt hàng này có thời gian kết thúc lớn nhất so với các yêu cầu đã chọn thì những yêu cầu trước buộc phải có thời gian kết thúc $\leq L$.

Từ nhận xét trên, ta thấy với 1 đơn đặt hàng (L,R,C), $DP[R]$ sẽ được cập nhật là $DP[R] = max(DP[L]+C)$.

Vậy ta có **công thức QHĐ** như sau:

$$DP[R] = max(DP[L]+C) \ \ \forall\ R \leq 10^5 \ \ \& \ \ \forall\ L \text{ tồn tại đơn đặt hàng } (L,R,C)$$

15.4 Khởi gán

DP[0] = 0;

15.5 Đáp án

Đáp án của ta là $DP[10^5]$ với ý nghĩa là số tiền thuê kiếm được là nhiều nhất có thể với các yêu cầu được chọn có thời gian kết thúc $\leq 10^5$.

15.6 Code mẫu

```
#include<bits/stdc++.h>
using namespace std;

const int MAXN = 1e6 + 5;
int n;
long long DP[MAXN];
vector<int> request[MAXN];

int main(){
    ios_base::sync_with_stdio(0);cin.tie(0);cout.tie(0);
```

```
    cin >> n;
    for(int i = 1 ; i <= n ; i++){
        int u , v;
        cin >> u >> v;
        request[v].push_back(u);
    }

    DP[0] = 0;
    for(int R = 1 ; R <= 100000 ; R++){
        DP[R] = DP[R - 1];
        for(int j = 0 ; j < request[R].size() ; j++){
            int L = request[R][j];
            DP[R] = max(DP[R] , DP[L] + (R - L));
        }
    }

    cout << DP[100000] << endl;
    return 0;
}
```

Đọc code đầy đủ hơn ở *đây*

16 Bài P - Nối mạng

16.1 Nhận xét

Yêu cầu của bài toán là tìm tổng độ dài dây cáp mạng ít nhất để tất cả các máy đều được nối dây mạng với ít nhất 1 máy khác.

16.2 Định nghĩa DP

Gọi DP[i] là chi phí ít nhất để nối tới máy thứ i sao cho các máy từ $1 \rightarrow i$ đều được nối với ít nhất 1 máy khác.

16.3 Suy ra công thức DP

Với bài toán này mình có 2 trường hợp chuyển trạng thái như sau:

+) nối tiếp máy thứ i vào máy thứ $i-1$ cùng các máy ở trước.

$\Rightarrow DP[i] = DP[i-1] + a[i-1]$

+) nối máy thứ i với máy thứ $i-1$ riêng biệt với các máy ở trước.

$\Rightarrow DP[i] = DP[i-2] + a[i-1]$

Vậy **công thức QHĐ** của ta là:

$$DP[i] = min(DP[i-1] + a[i-1], DP[i-2] + a[i-1])$$

16.4 Khởi gán

$DP[1] = \infty$
$DP[2] = a[1]$

16.5 Đáp án

Đáp án của ta là $DP[n]$ với ý nghĩa là để nối hết n máy sao cho thỏa mãn điều kiện.

16.6 Code mẫu

```
#include<bits/stdc++.h>
using namespace std;

const int MOD = 1e9 + 7;
const int MAXN = 1e5 + 5;
int n;
long long dp[MAXN] , a[MAXN];

int main(){
    ios_base::sync_with_stdio(0);cin.tie(0);cout.tie(0);
    cin >> n;
    for(int i = 1 ; i <= n ; i++)
        cin >> a[i];
    dp[1] = 1e9;
    dp[2] = a[1];
    for(int i = 3 ; i <= n ; i++)
        dp[i] = min(dp[i - 1] + a[i - 1] , dp[i - 2] + a[i - 1]);
    cout << dp[n] << endl;
    return 0;
}
```

Đọc code đầy đủ hơn ở *đây*

17 Bài Q - Dãy WAVIO

17.1 Nhận xét

Trong bài toán này, ta có nhận xét. Với 1 dãy WAVIO độ dài $2 * N + 1$, ta chia dãy ra làm 2 phần:

+) Phần đầu là dãy con tăng độ dài N + 1 kết thúc tại vị trí i.

+) Phần sau là dãy con giảm độ dài N + 1 bắt đầu tại vị trí i.

17.2 Định nghĩa DP

Gọi:

+) DP1[i] là dãy con tăng dài nhất kết thúc tại i.

+) DP2[i] là dãy con giảm dài nhất bắt đầu tại i.

17.3 Suy ra công thức DP

Với $DP1[i]$, ta có thể tính được bằng công thức :

$$DP1[i] = max(DP1[j]+1) \ \ \forall \ j < i, \ a[j] < a[i]$$

Với $DP2[i]$, ta có nhận xét như sau, đảo ngược mảng lại và ta có dãy con tăng kết thúc tại i ở mảng đảo ngược sẽ là dãy con giảm bắt đầu tại $n-i+1$ ở mảng ban đầu.
Vậy $DP2[i]$ sẽ có công thức tương tự với $DP1[i]$ nhưng là với mảng sau khi đảo ngược

17.4 Khởi gán

$DP[i] = 1 \ \ \forall \ i \leq n$
$DP2[i] = 1 \ \ \forall \ i \leq n$

17.5 Đáp án

Đáp án của ta sẽ là:

$$max(\ min(DP1[i],\ DP2[n-i+1]) * 2 - 1\ ) \forall \ \ i \leq n$$

Với ý nghĩa là, $min(DP1[i],\ DP2[n-i+1])$ là độ dài của 2 phần đầu và cuối, nhân đôi và trừ đi phần tử thứ i bị lặp lại 2 lần sẽ có được độ dài dãy WAVIO dài nhất có vị trí ở giữa là i.

17.6 Code mẫu

```
#include <bits/stdc++.h>
using namespace std;

const int MAXN = 1005;

int n;
int a[MAXN];
int DP1[MAXN] , DP2[MAXN];

int main() {
        ios::sync_with_stdio(0);cin.tie(0);cout.tie(0);
        cin >> n;
        for(int i = 1 ; i <= n ; i++)
```

```
            cin >> a[i];

        for(int i = 1 ; i <= n ; i++){
            DP1[i] = 1;
            for(int j = 1 ; j < i ; j++)
                if(a[j] < a[i])
                    DP1[i] = max(DP1[i] , DP1[j] + 1);
        }

        reverse(a + 1 , a + 1 + n);

        for(int i = 1 ; i <= n ; i++){
            DP2[i] = 1;
            for(int j = 1 ; j < i ; j++)
                if(a[j] < a[i])
                    DP2[i] = max(DP2[i] , DP2[j] + 1);
        }

        int res = 0;
        for(int i = 1 ; i <= n ; i++)
            res = max(res , min(DP1[i] , DP2[n - i + 1]) * 2 - 1);
    cout << res << endl;
    return 0;
}
```

Đọc code đầy đủ hơn ở *đây*

18 Bài R - Bitcoin

18.1 Nhận xét

Với bài toán này, vì biết trước các giá tiền nên mỗi lần mua, ta sẽ mua hết và mỗi lần bán, ta sẽ bán hết. Và ta chỉ mua và bán khi biết nó sinh lời.

18.2 Định nghĩa DP

Gọi $DP[i]$ là số tiền lớn nhất có được cho tới hết ngày thứ i.

18.3 Suy ra công thức DP

Đầu tiền, ta viết 1 hàm $f(tien, j, i)$ với ý nghĩa là tính số tiền có được sau khi mua số Bitcoin có giá trị $tien$ tại ngày thứ j và bán đi hết vào ngày thú i. Hàm được tính như sau:

$$f(tien, j, i) = tien * \frac{a[i]}{a[j]} - tien * 2\%$$

Từ định nghĩa $DP[i]$ trên, ta có các trường hợp chuyển trạng thái sau:

+) Không giao dịch ở ngày thứ i, vì không có giao dịch nên số tiền kiếm được tại ngày thứ i sẽ bằng số tiền tại ngày thứ $i-1$.

$$\Rightarrow DP[i] = DP[i-1]$$

+) Thực hiện giao dịch bán ở ngày thứ i và mua ở ngày thứ j ($j < i$) với số tiền có được từ ngày $j-1$, vì ta sẽ mua hết tiền vào Bitcoin nên các ngày $[j+1; i-1]$ sẽ không có giao dịch nào và số tiền có được tại ngày thứ i là từ việc bán Bitcoin.

$$\Rightarrow DP[i] = f(DP[j-1],\ j,\ i)$$

Vậy cuối cùng, ta có **công thức QHĐ** như sau:

$$DP[i] = max(DP[i-1],\ max(f(DP[j-1],\ j,\ i))\ )\ \ \forall j < i$$

18.4 Khởi gán

$DP[1] = x$

18.5 Đáp án

Đáp án của ta là $DP[n]$ với ý nghĩa là số tiền nhiều nhất kiếm được cho tới hết ngày thứ i.

18.6 Code mẫu

```
#include <bits/stdc++.h>
using namespace std;

const int MAXN = 1005;

int n , x;
int a[MAXN];
long double DP[MAXN];

long double f(long double tien , int j , int i){
    return tien * ((long double)a[i] / a[j]) - (long double)tien * 2 / 100;
}

int main() {
        ios::sync_with_stdio(0);cin.tie(0);cout.tie(0);
        int testCase;
        cin >> testCase;
        while(testCase--){
        cin >> n >> x;
        for(int i = 1 ; i <= n ; i++)
            cin >> a[i];
```

```
        DP[0] = x;
        for(int i = 1 ; i <= n ; i++){
            DP[i] = DP[i - 1];
            for(int j = 1 ; j < i ; j++)
                DP[i] = max(DP[i] , f(DP[j - 1] , j , i));
        }
        cout << fixed << setprecision(5) << DP[n] << endl;
        }
    return 0;
}
```

Đọc code đầy đủ hơn ở *đây*

19 Bài S - Giao Lưu

19.1 Nhận xét

Đây là 1 bài toán DP cơ bản và ta có thể liên tưởng tới bài toán xếp gạch khi suy nghĩ công thức cho bài toán này.

19.2 Định nghĩa DP

Gọi $DP[i][0 \xrightarrow{2}]$ là số cách xếp hàng có i bạn sao cho có đúng $0 \xrightarrow{2}$ bạn nam ở cuối.

19.3 Suy ra công thức DP

Từ định nghĩa trên, ta suy ra cách chuyển trạng thái từ 2 việc:

+) thêm 1 bạn nữ vào cuối hàng, cuối hàng sẽ không còn bạn nam nào.

$\Rightarrow DP[i][0] += DP[i-1][0] + DP[i-1][1] + DP[i-1][2]$

+) thêm 1 bạn nam vào cuối hàng, thì cuối hàng sẽ có thêm 1 bạn nam.

$\Rightarrow DP[i][1] += DP[i-1][0]$

$\Rightarrow DP[i][2] += DP[i-1][1]$

Vậy cuối cùng, **công thức QHĐ** của chúng ta là:

$$DP[i][1] = DP[i-1][0] + DP[i-1][1] + DP[i-1][2] \quad \forall\ i \le n$$

$$DP[i][1] += DP[i-1][0] \quad \forall\ i \le n$$

$$DP[i][2] += DP[i-1][1] \quad \forall\ i \le n$$

19.4 Khởi gán

$DP[0][0] = 1$

19.5 Đáp án

Đáp án của ta là $DP[n][0]+DP[n][1]+DP[n][2]$ mang ý nghĩa là tổng số cách xếp n bạn vào hàng cho cho không có quá 2 bạn nam xếp cạnh nhau.

19.6 Code mẫu

```
#include <bits/stdc++.h>
using namespace std;

const int MAXN = 65;

int n;
long long dp[MAXN][3];

int main() {
        ios::sync_with_stdio(0);cin.tie(0);cout.tie(0);
        cin >> n;
    dp[0][0] = 1;
    for(int i = 1 ; i <= n ; i++){
        dp[i][0] = dp[i - 1][1] + dp[i - 1][2] + dp[i - 1][0];
        dp[i][1] = dp[i - 1][0];
        dp[i][2] = dp[i - 1][1];
    }

    cout << dp[n][0] + dp[n][1] + dp[n][2] << endl;
    return 0;
}
```

Đọc code đầy đủ hơn ở *đây*

20 Bài T - Giao lưu 2

20.1 Nhận xét

Bài toán này là nâng cấp của bài S, vì thuật toán của bài S là $O(N*K)$ với N là số bạn trên hàng và K là số lượng bạn nam không được đứng cùng nhau.

Vì vậy với bài T, ta có cải tiến như sau.

20.2 Định nghĩa DP

Gọi $DP[i]$ là số cách xếp hàng có i bạn sao cho không có K bạn nam nào đứng cạnh nhau.

20.3 Suy ra công thức DP

Từ định nghĩa trên, ta suy ra các cách chuyển trạng thái:

+) Thêm 1 bạn nữ vào hàng, vì thêm 1 bạn nữ nên sẽ không ảnh hưởng tới tính thỏa mã của bài toán.

$\Rightarrow DP[i] += DP[i-1]$

+) Thêm 1 bạn nam vào hàng, vì không được có K bạn nam đứng cùng nhau nên ta phải trừ đi các trường hợp có K bạn nam ở cuối, trừ đi $DP[i-K-1]$ mang ý nghĩa để lại K vị trí trống cuối cùng là K bạn nam.

$\Rightarrow DP[i] += DP[i-1] - DP[i-K-1]$

Vậy công thức QHĐ của ta là:

$$DP[i] = 2 * DP[i-1] - DP[i-K-1]$$

Lưu ý: Chia lấy dư cho 10^9+7

20.4 Khởi gán

$DP[0] = 1$

20.5 Đáp án

Đáp án của ta là $DP[n]$ với ý nghĩa là số cách xếp n bạn sao cho không có K bạn nam nào đứng cùng nhau.

20.6 Code mẫu

```
#include <bits/stdc++.h>
using namespace std;

const int MAXN = 1e6 + 5;
const int MOD = 1e9 + 7;

int n , k;
int dp[MAXN];

int main() {
        ios::sync_with_stdio(0);cin.tie(0);cout.tie(0);
        cin >> n >> k;

    dp[0] = 1;

    for(int i = 1 ; i <= n ; i++){
        dp[i] = (2 * dp[i - 1]) % MOD;
        if(i - k - 1 >= 0)
            dp[i] = ((long long)dp[i] - dp[i - k - 1] + (long long)MOD * MOD) % MOD;
```

```
        else if(i - k == 0)
            dp[i] = ((long long)dp[i] - 1 + (long long)MOD * MOD) % MOD;
    }

    cout << dp[n] << endl;
    return 0;
}
```

Đọc code đầy đủ hơn ở *đây*

21 Bài U - Giao lưu 3

21.1 Nhận xét

Đây là bài toán DP cơ bản.

21.2 Định nghĩa DP

Gọi $DP[i]$ là số cách xếp i bạn vào hàng sao cho mỗi bạn nam cách nhau ít nhất 1 bạn nữ.

21.3 Suy ra công thức DP

Với định nghĩa trên, ta có cách chuyển trạng thái sau:

+) Thêm 1 bạn nữ vào hàng.

$\Rightarrow DP[i] += DP[i-1]$

+) Thêm 1 bạn nam vào hàng, vì giữa mỗi bạn nam phải có ít nhất K bạn nữ ở giữa nên ta sẽ loại trượng hợp không thỏa mãn bằng cách cho 1 lần K bạn vào hàng bao gồm K bạn nữ và 1 bạn nam ở cuối cùng.

$\Rightarrow DP[i] += DP[i-k-1]$

21.4 Khởi gán

$DP[i] = i+1 \ \ \forall i \leq K$

Vì với 1 dãy có độ dài $i(i \leq K)$, thì ta có 1 cách là dãy toàn bộ là nữ hoặc là có i cách đặt sao cho 1 bạn nam được đặt tại các vị trí i.

21.5 Đáp án

Đáp án của ta là $DP[n]$ với định nghĩa là số cách xếp vào hàng n bạn sao cho không có 2 bạn nam nào cách nhau ít hơn K bạn nữ.

21.6 Code mẫu

```
#include<bits/stdc++.h>

using namespace std;

const int MAXN = 1e6 + 5;
const int MOD = 1e9 + 7;
int dp[MAXN], n, k;

int main() {
    ios::sync_with_stdio(0);cin.tie(0);cout.tie(0);
    cin >> n >> k;
        for (int i = 0; i <= k; i++) dp[i] = i + 1;
    for (int i = k + 1; i <= n; i++)
        dp[i] = ((long long)dp[i - 1] + dp[i - k - 1]) % MOD;
        cout << dp[n];
    return 0;
}
```

Đọc code đầy đủ hơn ở *đây*

22 Bài V - Lát gạch 4

22.1 Nhận xét

Bài toán này là tiếp nối của bài Lát gạch 3.

22.2 Định nghĩa DP

Gọi $DP[i]$ là tổng số bao phủ của tất cả các cách lát hình chữ nhật có diện tích $2xi$.

Gọi $w[i]$ là số cách lát hình chữ nhất có diện tích $2xi$ (Tương tự như bài Lát gạch 3).

22.3 Suy ra công thức DP

Thứ nhất, tương tự với bài Lát gạch 3, công thức QHĐ của $w[i]$ là:

$$w[i] = w[i-1] + 4 * w[i-2] + 2 * w[i-3]$$

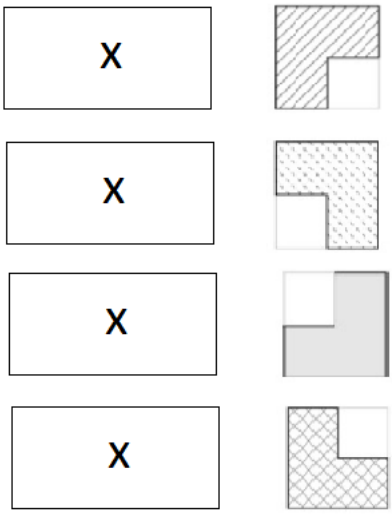
Tiếp theo, với $DP[i]$ ta có các cách chuyển trạng thái như sau:

+) không đặt gạch vào cột thứ i nên tổng số bao phủ sẽ là của tất cả các $2x(i-1)$ viên gạch trước

$$\Rightarrow DP[i] += DP[i-1]$$

+) Thêm vào 1 hình chữ L và có 4 cách, việc ta thêm 1 hình chữ L sẽ khiến tổng bao phủ của phần đã lát bị lặp lại và với 4 cách, tổng phần bao phủ phía trước sẽ được lặp lại 4 lần và đều là tổng độ bao phủ của tất cả các $2x(i-2)$ viên gạch trước.

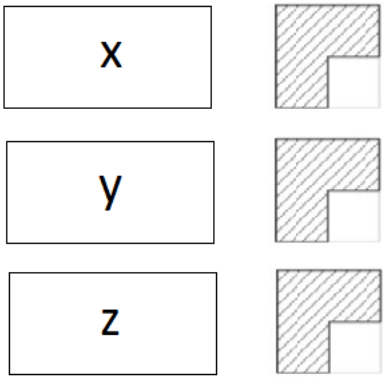
$$\Rightarrow DP[i] += 4 * DP[i-2]$$



- Giải thích cách nói trên, nếu hình chữ nhật có chữ x là trạng thái hiện tại, thì thêm vào 4 hình chữ L tượng trung cho 4 cách thì ta thấy hình chữ nhật của trạng thái hiện tại bị lặp 4 lần.

+) Cùng với việc thêm 1 hình chữ L và có 4 cách, ta cũng phải cộng thêm phần bao phủ của cả hình chứ L thêm vào. Đối với 1 cách, tổng bao phủ được thêm vào sẽ bằng với số cách mà hình chữ nhật $2xi$ có thể được lấp. Vậy với cả 4 cách, mỗi cách có thêm độ bao phủ là 3, có:

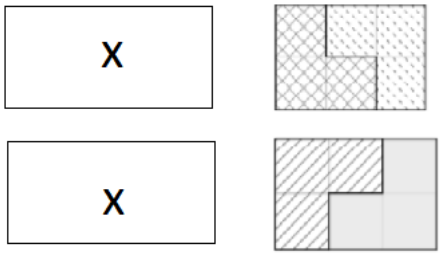
$$\Rightarrow DP[i] += w[i-2] * 4 * 3$$



- Giải thích cách nói trên, giả sử trạng thái hiện tại có 3 cách lát, tương ứng với 3 hình chữ nhật x , y , z thì 1 cách lát hình chữ L của ta sẽ được tính 3 lần.

+) Thêm vào 1 hình chữ nhật kích thước $2x3$ được tạo nên từ 2 hình chữ L và có 2 cách, việc ta thêm 1 hình chữ nhật sẽ lặp lại phần diện tích bao phủ phía trước. Vậy tổng phần bao phủ phía trước sẽ được lặp lại 2 lần và đều là tổng độ bao phủ của tất cả các $2x(i-3)$ viên gạch trước

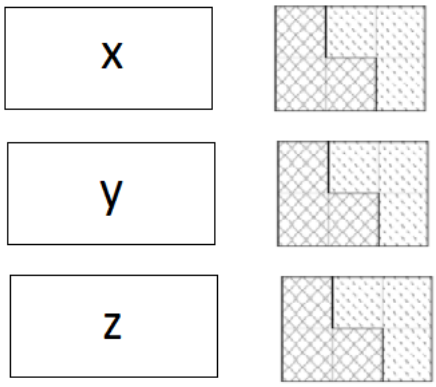
$$\Rightarrow DP[i] += 2 * DP[i-3]$$



- Giải thích cách nói trên, nếu hình chữ nhật có chữ x là trạng thái hiện tại, thì thêm vào 2 hình chữ nhật $2x3$ tượng trung cho 2 cách thì ta thấy hình chữ nhật của trạng thái hiện tại bị lặp 2 lần.

+) Cùng với việc thêm 1 hình chữ nhật $2x3$ và có 2 cách, ta phải cộng thêm tổng bao phủ được tạo ra từ các hình chữ nhật được thêm vào. Với 2 cách, mỗi cách thêm diện tích bao phủ là 6, có:

$$\Rightarrow DP[i] += w[i-3] * 2 * 6$$



- Giải thích cách nói trên, giả sử trạng thái hiện tại có 3 cách lát, tương ứng với 3 hình chữ nhật x , y , z thì 1 cách lát hình chứ nhật $2x3$ của ta sẽ được tính 3 lần.

Vậy cuối cùng, **công thức QHĐ cho** $DP[i]$ là:

$$DP[i] = DP[i-1] + 4 * DP[i-2] + 12 * w[i-2] + 2 * DP[i-3] + 12 * w[i-3]$$

22.4 Khởi gán

$w[0] = 1$
$w[1] = 1$
$w[2] = 5$
$DP[0] = 0$
$DP[1] = 0$

$DP[2] = 12$

22.5 Đáp án

Đáp án của ta sẽ là $DP[n]$ với ý nghĩa là tổng độ bao phủ của tất cả các cách lát gạch cho hình chữ nhất $2xn$.

22.6 Code mẫu

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

const int MAXN = 1e6 + 6;
const int MOD = 998244353;

int n;
long long w[MAXN] , dp[MAXN];

int main(){
    ios::sync_with_stdio(0);cin.tie(0);cout.tie(0);
    cin >> n;
    w[0] = 1;
    w[1] = 1;
    w[2] = 5;
    for(int i = 3 ; i <= n ; i++)
        w[i] = (w[i - 1] + w[i - 2] * 4 + w[i - 3] * 2) % MOD;

    dp[0] = 0;
    dp[1] = 0;
    dp[2] = 12;
    for(int i = 3 ; i <= n ; i++)
        dp[i] = (dp[i - 1] + 4*dp[i - 2] + 12*w[i - 2] + 2*dp[i - 3] + 12*w[i - 3]) % MOD;

    cout << dp[n] << endl;
    return 0;
}
```

Đọc code đầy đủ hơn ở *đây*