Mục lục

1 Bài A - Fibonacci **13**
1.1 Nhận xét . . . 13
1.2 Định nghĩa DP . . . 13
1.3 Suy ra công thức DP . . . 13
1.4 Khởi gán . . . 13
1.5 Đáp án . . . 13
1.6 Code mẫu . . . 13

2 Bài B - Frog 1 **14**
2.1 Nhận xét . . . 14
2.2 Định nghĩa DP . . . 14
2.3 Suy ra công thức DP . . . 14
2.4 Khởi gán . . . 14
2.5 Đáp án . . . 14
2.6 Code mẫu . . . 14

3 Bài C- Mua vé **15**
3.1 Nhận xét . . . 15
3.2 Định nghĩa DP . . . 15
3.3 Suy ra công thức DP . . . 15
3.4 Khởi gán . . . 15
3.5 Đáp án . . . 15
3.6 Code mẫu . . . 16

4 Bài D - Bậc Thang **16**
4.1 Nhận xét . . . 16
4.2 Định nghĩa DP . . . 16
4.3 Suy ra công thức DP . . . 16
4.4 Khởi gán . . . 17
4.5 Đáp án . . . 17
4.6 Code mẫu . . . 17

5 Bài E - Frog 2 **18**
5.1 Nhận xét . . . 18
5.2 Định nghĩa DP . . . 18
5.3 Suy ra công thức DP . . . 18
5.4 Khởi gán . . . 18
5.5 Đáp án . . . 18
5.6 Code mẫu . . . 18

6 Bài F - Lát gạch 1 **19**

6.1 Nhận xét . . . 19
6.2 Định nghĩa DP . . . 19
6.3 Suy ra công thức DP . . . 19
6.4 Khởi gán . . . 19
6.5 Đáp án . . . 20
6.6 Code mẫu . . . 20

7 Bài G - LIQ **20**

7.1 Nhận xét . . . 20
7.2 Định nghĩa DP . . . 20
7.3 Suy ra công thức DP . . . 21
7.4 Khởi gán . . . 21
7.5 Đáp án . . . 21
7.6 Code mẫu . . . 21

8 Bài H - Dãy con chính phương **22**

8.1 Nhận xét . . . 22
8.2 Định nghĩa DP . . . 22
8.3 Suy ra công thức DP . . . 22
8.4 Khởi gán . . . 22
8.5 Đáp án . . . 22
8.6 Code mẫu . . . 22

9 Bài I - Lát gạch 2 **23**

9.1 Nhận xét . . . 23
9.2 Định nghĩa DP . . . 23
9.3 Suy ra công thức DP . . . 23
9.4 Khởi gán . . . 25
9.5 Đáp án . . . 25
9.6 Code mẫu . . . 25

10 Bài J - Lát gạch 3 **26**

10.1 Nhận xét . . . 26
10.2 Định nghĩa DP . . . 26
10.3 Suy ra công thức DP . . . 26
10.4 Khởi gán . . . 27
10.5 Đáp án . . . 27
10.6 Code mẫu . . . 27

11 Bài K - Đoạn con liên tiếp có tổng lớn nhất **27**
11.1 Nhận xét . . . 27
11.2 Định nghĩa DP . . . 27
11.3 Suy ra công thức DP . . . 28
11.4 Khởi gán . . . 28
11.5 Đáp án . . . 28
11.6 Code mẫu . . . 28

12 Bài L - Dãy con liên tiếp có tổng lớn nhất 2 **29**
12.1 Nhận xét . . . 29
12.2 Định nghĩa DP . . . 29
12.3 Suy ra công thức DP . . . 29
12.4 Suy ra công thức DP . . . 29
12.5 Khởi gán . . . 29
12.6 Đáp án . . . 29
12.7 Code mẫu . . . 30

13 Bài M - Hội trường 1 **31**
13.1 Nhận xét . . . 31
13.2 Định nghĩa DP . . . 31
13.3 Suy ra công thức DP . . . 31
13.4 Khởi gán . . . 31
13.5 Đáp án . . . 31
13.6 Code mẫu . . . 31

14 Bài N - Hội trường 2 **32**
14.1 Nhận xét . . . 32
14.2 Định nghĩa DP . . . 32
14.3 Suy ra công thức DP . . . 32
14.4 Khởi gán . . . 33
14.5 Đáp án . . . 33
14.6 Code mẫu . . . 33

15 Bài O - Hội trường 3 **34**
15.1 Nhận xét . . . 34
15.2 Định nghĩa DP . . . 34
15.3 Suy ra công thức DP . . . 34
15.4 Khởi gán . . . 34
15.5 Đáp án . . . 34
15.6 Code mẫu . . . 34

16 Bài P - Nối mạng **35**
16.1 Nhận xét 35
16.2 Định nghĩa DP 35
16.3 Suy ra công thức DP 35
16.4 Khởi gán 36
16.5 Đáp án 36
16.6 Code mẫu 36

17 Bài Q - Dãy WAVIO **36**
17.1 Nhận xét 36
17.2 Định nghĩa DP 37
17.3 Suy ra công thức DP 37
17.4 Khởi gán 37
17.5 Đáp án 37
17.6 Code mẫu 37

18 Bài R - Bitcoin **38**
18.1 Nhận xét 38
18.2 Định nghĩa DP 38
18.3 Suy ra công thức DP 38
18.4 Khởi gán 39
18.5 Đáp án 39
18.6 Code mẫu 39

19 Bài S - Giao Lưu **40**
19.1 Nhận xét 40
19.2 Định nghĩa DP 40
19.3 Suy ra công thức DP 40
19.4 Khởi gán 40
19.5 Đáp án 41
19.6 Code mẫu 41

20 Bài T - Giao lưu 2 **41**
20.1 Nhận xét 41
20.2 Định nghĩa DP 41
20.3 Suy ra công thức DP 42
20.4 Khởi gán 42
20.5 Đáp án 42
20.6 Code mẫu 42

21 Bài U - Giao lưu 3 **43**
21.1 Nhận xét 43
21.2 Định nghĩa DP 43
21.3 Suy ra công thức DP 43
21.4 Khởi gán 43
21.5 Đáp án 43
21.6 Code mẫu 44

22 Bài V - Lát gạch 4 **44**
22.1 Nhận xét 44
22.2 Định nghĩa DP 44
22.3 Suy ra công thức DP 44
22.4 Khởi gán 46
22.5 Đáp án 47
22.6 Code mẫu 47

23 Bài W - Số cách đi trên ma trận **47**
23.1 Nhận xét 47
23.2 Định nghĩa DP 48
23.3 Suy ra công thức DP 48
23.4 Khởi gán 48
23.5 Đáp án 48
23.6 Code mẫu 48

24 Bài X - Đường đi lớn nhất **49**
24.1 Nhận xét 49
24.2 Định nghĩa DP 49
24.3 Suy ra công thức DP 49
24.4 Khởi gán 50
24.5 Đáp án 50
24.6 Code mẫu 50

25 Bài Y - Thanh và kỳ nghỉ **51**
25.1 Nhận xét 51
25.2 Định nghĩa DP 51
25.3 Suy ra công thức DP 51
25.4 Khởi gán 52
25.5 Đáp án 52
25.6 Code mẫu 52

26 Bài Z - Dãy con có tổng bằng S **53**
26.1 Nhận xét 53
26.2 Định nghĩa DP 53
26.3 Suy ra công thức DP 53
26.4 Khởi gán 53
26.5 Đáp án 53
26.6 Code mẫu 53

27 Bài ZA - Chia kẹo **54**
27.1 Nhận xét 54
27.2 Định nghĩa DP 54
27.3 Suy ra công thức DP 54
27.4 Khởi gán 55
27.5 Đáp án 55
27.6 Code mẫu 55

28 Bài ZB - Tích lớn nhất **56**
28.1 Nhận xét 56
28.2 Định nghĩa DP 56
28.3 Suy ra công thức DP 56
28.4 Khởi gán 56
28.5 Đáp án 57
28.6 Code mẫu 57

29 Bài ZC - Xúc xắc **58**
29.1 Nhận xét 58
29.2 Định nghĩa DP 58
29.3 Suy ra công thức DP 58
29.4 Khởi gán 58
29.5 Đáp án 58
29.6 Code mẫu 58

30 Bài ZD - Cái túi **59**
30.1 Nhận xét 59
30.2 Định nghĩa DP 59
30.3 Suy ra công thức DP 59
30.4 Khởi gán 60
30.5 Đáp án 60
30.6 Code mẫu 60

31 Bài ZE- Sự yêu đời 61
31.1 Nhận xét 61
31.2 Định nghĩa DP 61
31.3 Suy ra công thức DP 61
31.4 Khởi gán 61
31.5 Đáp án 61
31.6 Code mẫu 62

32 Bài ZF - cnum 62
32.1 Nhận xét 62
32.2 Định nghĩa DP 63
32.3 Suy ra công thức DP 63
32.4 Khởi gán 63
32.5 Đáp án 63
32.6 Code mẫu 63

33 Bài ZG - cscong 64
33.1 Nhận xét 64
33.2 Định nghĩa DP 64
33.3 Suy ra công thức DP 64
33.4 Khởi gán 65
33.5 Đáp án 65
33.6 Code mẫu 65

34 Bài ZH - Nhật Khôi và những đóa hoa 66
34.1 Nhận xét 66
34.2 Định nghĩa DP 66
34.3 Suy ra công thức DP 66
34.4 Khởi gán 66
34.5 Đáp án 66
34.6 Code mẫu 66

35 Bài ZI - MODULE 67
35.1 Nhận xét 67
35.2 Định nghĩa DP 68
35.3 Suy ra công thức DP 68
35.4 Khởi gán 68
35.5 Đáp án 68
35.6 Code mẫu 68

36 Bài ZJ - TIENPHAT **69**
36.1 Nhận xét 69
36.2 Định nghĩa DP 70
36.3 Suy ra công thức DP 70
36.4 Khởi gán 71
36.5 Đáp án 71
36.6 Code mẫu 71

37 Bài ZK - Quá kinh điển **72**
37.1 Nhận xét 72
37.2 Định nghĩa DP 72
37.3 Suy ra công thức DP 73
37.4 Khởi gán 73
37.5 Đáp án 73
37.6 Code mẫu 73

38 Bài ZL - Wrong Answer on test 2 **74**
38.1 Nhận xét 74
38.2 Định nghĩa DP 74
38.3 Suy ra công thức DP 74
38.4 Khởi gán 75
38.5 Đáp án 75
38.6 Code mẫu 75

39 Bài ZM - Xâu con chung dài nhất **76**
39.1 Nhận xét 76
39.2 Định nghĩa DP 76
39.3 Suy ra công thức DP 76
39.4 Khởi gán 76
39.5 Đáp án 77
39.6 Code mẫu 77

40 Bài ZO - Con đường hoa **77**
40.1 Nhận xét 77
40.2 Định nghĩa DP 78
40.3 Suy ra công thức DP 78
40.4 Khởi gán 78
40.5 Đáp án 78
40.6 Code mẫu 78

41 Bài ZP - dzero **79**
41.1 Nhận xét 79
41.2 Định nghĩa DP 79
41.3 Suy ra công thức DP 79
41.4 Khởi gán 79
41.5 Đáp án 80
41.6 Code mẫu 80

42 Bài ZQ - Xóa số **80**
42.1 Nhận xét 80
42.2 Định nghĩa DP 80
42.3 Suy ra công thức DP 81
42.4 Khởi gán 81
42.5 Đáp án 81
42.6 Code mẫu 81

43 Bài ZR - Perfect balance as all things should be **82**
43.1 Nhận xét 82
43.2 Định nghĩa DP 82
43.3 Suy ra công thức DP 82
43.4 Khởi gán 83
43.5 Đáp án 83
43.6 Code mẫu 83

44 Bài ZS - Lại là mua quà **84**
44.1 Nhận xét 84
44.2 Định nghĩa DP 84
44.3 Suy ra công thức DP 85
44.4 Khởi gán 85
44.5 Đáp án 85
44.6 Code mẫu 85

45 Bài ZT - Bốc quà **86**
45.1 Nhận xét 86
45.2 Định nghĩa DP 86
45.3 Suy ra công thức DP 86
45.4 Khởi gán 87
45.5 Đáp án 87
45.6 Code mẫu 87

46 Bài ZU - DGIFT **88**
46.1 Nhận xét 88
46.2 Định nghĩa DP 88
46.3 Suy ra công thức DP 88
46.4 Khởi gán 89
46.5 Đáp án 89
46.6 Code mẫu 89

47 Bài ZV - Công viên 1 **90**
47.1 Nhận xét 90
47.2 Định nghĩa DP 90
47.3 Suy ra công thức DP 90
47.4 Khởi gán 91
47.5 Đáp án 91
47.6 Code mẫu 91

48 Bài ZX - Chuỗi đối xứng **92**
48.1 Nhận xét 92
48.2 Định nghĩa DP 92
48.3 Suy ra công thức DP 92
48.4 Khởi gán 93
48.5 Đáp án 93
48.6 Code mẫu 93

49 Bài ZY - Chất nhờn **94**
49.1 Nhận xét 94
49.2 Định nghĩa DP 94
49.3 Suy ra công thức DP 94
49.4 Khởi gán 94
49.5 Đáp án 94
49.6 Code mẫu 94

50 Bài ZZ - NEGIKO **95**
50.1 Nhận xét 95
50.2 Định nghĩa DP 96
50.3 Suy ra công thức DP 96
50.4 Khởi gán 96
50.5 Đáp án 96
50.6 Code mẫu 96

51 Bài ZZA - Chip Move **97**
51.1 Nhận xét 97
51.2 Định nghĩa DP 98
51.3 Suy ra công thức DP 98
51.4 Khởi gán 98
51.5 Đáp án 99
51.6 Code mẫu 99

52 Bài ZZB - Trung bình **99**
52.1 Nhận xét 99
52.2 Định nghĩa DP 99
52.3 Suy ra công thức DP 100
52.4 Khởi gán 100
52.5 Đáp án 100
52.6 Code mẫu 100

53 Bài ZZC - Đếm dãy con **101**
53.1 Nhận xét 101
53.2 Định nghĩa DP 101
53.3 Suy ra công thức DP 101
53.4 Khởi gán 101
53.5 Đáp án 102
53.6 Code mẫu 102

54 Bài ZZD. Tổng tuyệt đối lớn nhất **103**
54.1 Nhận xét 103
54.2 Định nghĩa DP 103
54.3 Suy ra công thức DP 103
54.4 Khởi gán 103
54.5 Đáp án 104
54.6 Code mẫu 104

55 Bài ZZF - Lưu niệm **104**
55.1 Nhận xét 104
55.2 Định nghĩa DP 104
55.3 Suy ra công thức DP 105
55.4 Khởi gán 105
55.5 Đáp án 105
55.6 Code mẫu 105

56 Bài ZZG - Phần thưởng **106**
56.1 Nhận xét . . . 106
56.2 Định nghĩa DP . . . 106
56.3 Suy ra công thức DP . . . 106
56.4 Khởi gán . . . 107
56.5 Đáp án . . . 107
56.6 Code mẫu . . . 107

57 Bài ZZH. Dãy con chung không liền kề dài nhất **108**
57.1 Nhận xét . . . 108
57.2 Định nghĩa DP . . . 108
57.3 Suy ra công thức DP . . . 108
57.4 Khởi gán . . . 108
57.5 Đáp án . . . 108
57.6 Code mẫu . . . 109

58 Bài ZZI - Cột điện **109**
58.1 Nhận xét . . . 109
58.2 Định nghĩa DP . . . 109
58.3 Suy ra công thức DP . . . 110
58.4 Khởi gán . . . 111
58.5 Đáp án . . . 111
58.6 Code mẫu . . . 111

59 Bài ZZZ. Trồng cây **112**
59.1 Nhận xét . . . 112
59.2 Định nghĩa DP . . . 112
59.3 Suy ra công thức DP . . . 112
59.4 Khởi gán . . . 113
59.5 Đáp án . . . 113
59.6 Code mẫu . . . 113

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

```cpp
#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:

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

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 chắn 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:

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

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:

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

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.

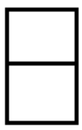
9.2 Định nghĩa DP

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

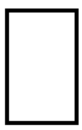
Đầu tiên, ta xét đến các cách để lát thêm vào duy nhất 1 cột ở cuối:

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



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

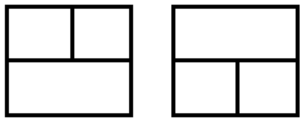
+) 1 khối $2x1$.Ảnh minh họa:



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

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:



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

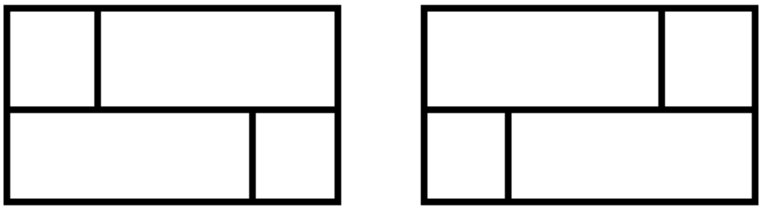
+) 2 khối $1x2$ ghép thành 1 khối $2x2$.Ảnh minh họa:

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

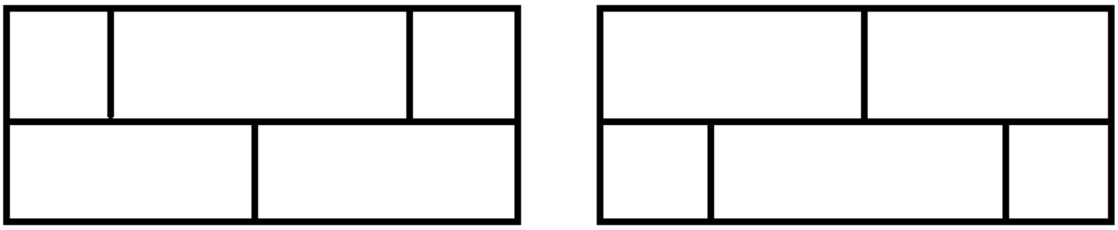
Và 1 trường hợp đặc biệt nữa là thêm vào cuối:

+) 1 khối $2xk$ có hình dạng là các khối $1x2$ giao nhau và 2 khối $1x1$ như sau:

+) là khối có $k = 3$:



+) là khối có $k = 4$:



$\Rightarrow DP[i] += 2 * DP[i-k] \ \ \forall\ k \geq 3.$

Vì nếu ta sử dụng for và lặp theo k sẽ bị TLE do có đpt là $O(n^2)$, ta sử dụng **PrefixSum**. Gọi $g[i]$ là tổng các $DP[1] \rightarrow DP[i]$.

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

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

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

9.4 Khởi gán

$dp[0] = 1$
$dp[1] = 2$
$dp[2] = 7$
$g[0] = 1$
$g[1] = 3$
$g[2] = 10$

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

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

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

int main(){
    ios_base::sync_with_stdio(0);cin.tie(0);cout.tie(0);
    cin >> n;
    dp[0] = 1;
    dp[1] = 2;
    dp[2] = 7;
    g[0] = 1;
    g[1] = 3;
    g[2] = 10;
    for(int i = 3 ; i <= n ; i++){
        dp[i] = (2LL * dp[i - 1] + 3LL * dp[i - 2] + 2LL * g[i - 3]) % MOD;
        g[i] = (g[i - 1] + dp[i]) % 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

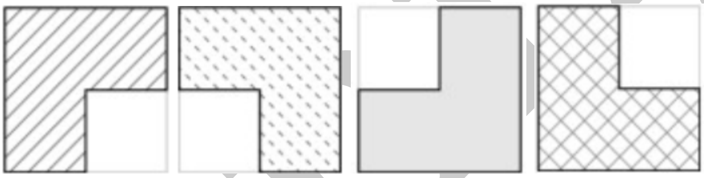
Cho rằng trạng thái i là số cột hiện tại bằng i, ta liệt kê ra các cách chuyển trạng thái:

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

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

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

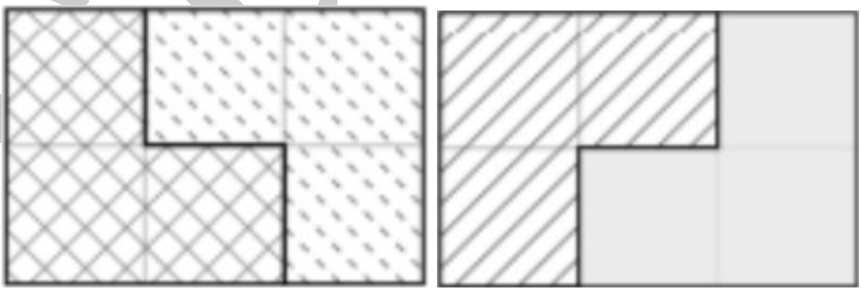
+) Thêm vào 1 hình chữ L, chiếm 2 cột, có 4 cách như sau:



$\Rightarrow$ 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:

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

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 $max(DP[i]) \forall \quad i \le n$ 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:

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

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 $\le i$

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 $\le L$.

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:

$DP[R] = max(DP[L]+(R-L)) \;\; \forall R \le 10^5 \;\; \& \;\; \forall L$ 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 $\le 10^5$.

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:

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

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 $\le i$

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 $\le L$.

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:

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

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 $\le 10^5$.

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 $1 \to i$ 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

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

$DP2[i] = 1 \ \ \forall \ i \le n$

17.5 Đáp án

Đáp án của ta sẽ là:

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

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 tiên, 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 thứ 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 $DP[i][0 \to 2]$ là số cách xếp hàng có i bạn sao cho có đúng $0 \to 2$ 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:

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

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

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

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

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

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 nhật 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:

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

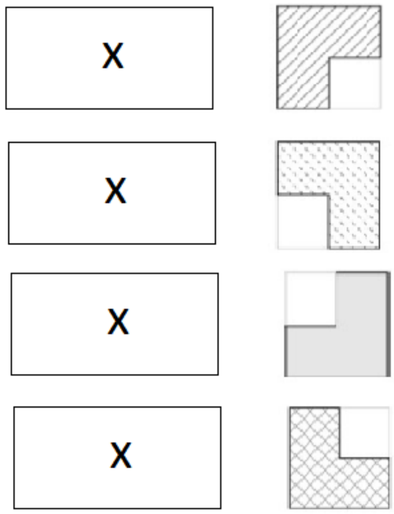
Tiếp theo, với $DP[i]$ ta có các cách chuyển trạng thái như sau:

+) 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.

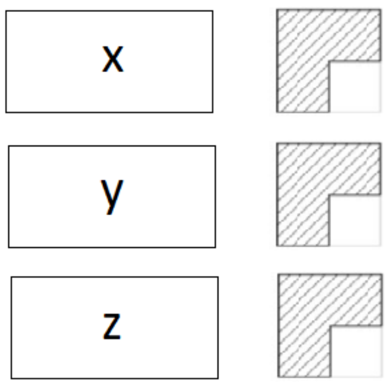
$\Rightarrow DP[i] += 4 * DP[i-2]$



- 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:

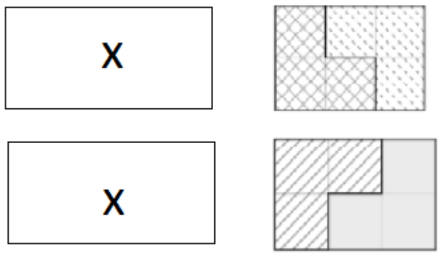
$\Rightarrow DP[i] += w[i-2] * 4 * 3$



- 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

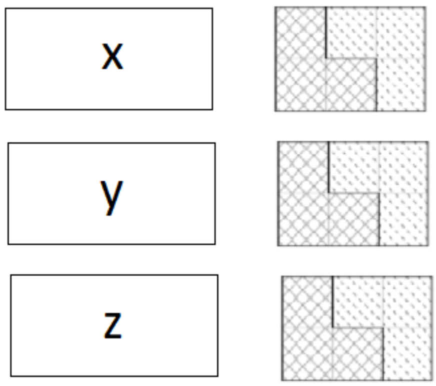
$\Rightarrow DP[i] += 2 * DP[i-3]$



- 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*

23 Bài W - Số cách đi trên ma trận

23.1 Nhận xét

Đây là bài toán DP cơ bản.

23.2 Định nghĩa DP

Gọi $DP[i][j]$ là số các để đến ô ở hàng i và cột j.

23.3 Suy ra công thức DP

Dựa vào cách đi được đề bài cho, ta có các cách chuyển trạng thái như sau:

+) Đi từ ô $(i-1, j) \to$ ô (i, j), số cách đi tới ô (i, j) được cộng thêm bằng số cách đi tới ô $(i-1, j)$

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

+) Đi từ ô $(i, j-1) \to$ ô (i, j) , số cách đi tới ô (i, j) được cộng thêm bằng số cách đi tới ô $(i, j-1)$

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

Vậy ta có **công thức QHĐ** như sau:

$$DP[i][j] = DP[i-1][j] + DP[i][j-1]$$

23.4 Khởi gán

$DP[1][1] = 1$

23.5 Đáp án

Đáp án của ta sẽ là $DP[H][W]$ với ý nghĩa là tổng số cách đi được tới ô (H, W).

Tuy nhiên, có một số ô bị chặn, giả sử ô (x, y) là ô bị chặn, ta có:

$$DP[x][y] = 0$$

23.6 Code mẫu

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

const int MAXW = 1e3 , MAXH = 1e3;
const int MOD = 1e9 + 7;

char a[MAXH + 1][MAXW + 1];
int DP[MAXH + 1][MAXW + 1];

int main()
{
    ios_base::sync_with_stdio(0);cin.tie(0);
    int H , W;
    cin >> H >> W;
```

```
    for(int i = 1 ; i <= H ; i++)
        for(int j = 1 ; j <= W ; j++) cin >> a[i][j];

    DP[1][1] = 1;
    for(int i = 1 ; i <= H ; i++){
        for(int j = 1 ; j <= W ; j++){
            if(i == 1 && j == 1) continue;
            if(a[i][j] == '#') continue;
            DP[i][j] = (DP[i - 1][j] + DP[i][j - 1]) % MOD;
        }
    }

    cout << DP[H][W] << endl;
    return 0;
}
```

Đọc code đầy đủ hơn *ở đây*

24 Bài X - Đường đi lớn nhất

24.1 Nhận xét

Đây là 1 bài toán DP cơ bản.

24.2 Định nghĩa DP

Gọi $DP[i][j]$ là tổng lớn nhất để tới được ô (i,j) trên bảng bằng cách đi trong đề.

24.3 Suy ra công thức DP

Với định nghĩ DP trên, ta có các cách chuyển trạng thái như sau:

+) Di chuyển tới ô (i,j) từ ô $(i-1,j-1)$

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

+) Di chuyển tới ô (i,j) từ ô (i,j)

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

+) Di chuyển tới ô (i,j) từ ô $(i+1,j-1)$

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

Vậy ta có **công thức QHĐ**:

$$DP[i][j] = max(DP[i-1][j-1], DP[i][j-1], DP[i+1][j-1]) + a[i][j]$$

Đến đây, ta cần lưu ý một điểm, **thứ tự 2 vòng for** của ta khi tính DP. Thường tình ta có:

```
for(int i = 1 ; i <= m ; i++)
    for(int j = 1 ; j <= n ; j++)
        ....
```

Nếu để ý, ta thấy $DP[i][j]$ sẽ được tính trước khi $DP[i+1][j-1]$ được tính.

⇒ Xét thiếu trường hợp đi từ ô $(i+1, j-1)$ Vậy để sửa lại, ta thay đổi vòng for thành

```
for(int j = 1 ; j <= n ; j++)
    for(int i = 1 ; i <= m ; i++)
        ....
```

Để tính **theo thứ tự các cột** thay vì theo **thứ tự các hàng**, ta ưu tiên tính theo j rồi tới i.

24.4 Khởi gán

$DP[i][1] = a[i][1] \ \ \forall\ i \leq m$

24.5 Đáp án

Đáp án của ta là $max(DP[n][i]) \ \ \forall\ i \leq m$ với ý nghĩa là đường đi có tổng lớn nhất để tới 1 ô bất kì trong cột n.

24.6 Code mẫu

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

const int MAXW = 1e3 , MAXH = 1e3;
const int MOD = 1e9 + 7;

int m , n;
int a[MAXH + 1][MAXW + 1];
int DP[MAXH + 1][MAXW + 1];

int main()
{
    ios_base::sync_with_stdio(0);cin.tie(0);
    cin >> m >> n;

    for(int i = 1 ; i <= m ; i++){
        for(int j = 1 ; j <= n ; j++)
            cin >> a[i][j];
    }

    for(int i = 1 ; i <= m ; i++) DP[i][1] = a[i][1];
```

```
    for(int j = 2 ; j <= n ; j++){
        for(int i = 1 ; i <= m ; i++){
            DP[i][j] = -1e9;
            if(i > 1)  DP[i][j] = max(DP[i][j] , DP[i - 1][j - 1] + a[i][j]);
            if(i < m)  DP[i][j] = max(DP[i][j] , DP[i + 1][j - 1] + a[i][j]);
            DP[i][j] = max(DP[i][j] , DP[i][j - 1] + a[i][j]);
        }
    }

    int res = -1e9;
    for(int i = 1 ; i <= m ; i++)
        res = max(res , DP[i][n]);
    cout << res << endl;
    return 0;
}
```

Đọc code đầy đủ hơn *ở đây*

25 Bài Y - Thanh và kỳ nghỉ

25.1 Nhận xét

Đây là 1 bài DP cơ bản.

25.2 Định nghĩa DP

Gọi $DP[i][1 \to 3]$ là độ vui lớn nhất xét tới ngày thứ i và có hoạt động cuối cùng tương ứng với hoạt động $1 \to 3$.

25.3 Suy ra công thức DP

Dựa vào định nghĩa trên và việc ta không được chọn 2 hoạt động liên tục ở ngày thứ i và ngày thứ $i-1$, ta có các cách chuyển trạng thái như sau:

+) Chọn ngày thứ i có hoạt động 1 và ngày thứ $i-1$ không được chọn hoạt động 1.

$\Rightarrow DP[i][1] = max(DP[i-1][2], DP[i-1][3]) + a[i]$

+) Chọn ngày thứ i có hoạt động 2 và ngày thứ $i-1$ không được chọn hoạt động 2.

$\Rightarrow DP[i][2] = max(DP[i-1][1], DP[i-1][3]) + b[i]$

+) Chọn ngày thứ i có hoạt động 3 và ngày thứ $i-1$ không được chọn hoạt động 3.

$\Rightarrow DP[i][3] = max(DP[i-1][1], DP[i-1][2]) + c[i]$

25.4 Khởi gán

$DP[1][1] = a[i]$

$DP[1][2] = b[i]$

$DP[1][3] = c[i]$

25.5 Đáp án

Đáp án của ta sẽ là $max(DP[i][1], DP[i][2], DP[i][3])$ với ý nghĩa là độ vui lớn nhất đạt được cho tới ngày thứ i sao cho không có 2 ngày liên tục có hoạt động giống nhau.

25.6 Code mẫu

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

const int MAXN = 1e5 + 5;

int n;
int a[MAXN] , b[MAXN] , c[MAXN];
long long dp[MAXN][4];

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] >> b[i] >> c[i];

    dp[1][1] = a[1];
    dp[1][2] = b[1];
    dp[1][3] = c[1];

    for(int i = 1 ; i <= n ; i++){
        dp[i][1] = max(dp[i - 1][2] , dp[i - 1][3]) + a[i];
        dp[i][2] = max(dp[i - 1][1] , dp[i - 1][3]) + b[i];
        dp[i][3] = max(dp[i - 1][1] , dp[i - 1][2]) + c[i];
    }

    cout << max(dp[n][1] , max(dp[n][2] , dp[n][3])) << endl;
    return 0;
}
```

Đọc code đầy đủ hơn ở *đây*

26 Bài Z - Dãy con có tổng bằng S

26.1 Nhận xét

Đây là bài toán cơ bản dạng DP cái túi - Knapsack DP.

26.2 Định nghĩa DP

Gọi $DP[i][j]$ là xét tới vị trí i và tổng hiện tại là j thì có bao nhiêu cách chọn dãy con có tổng bằng j.

26.3 Suy ra công thức DP

Dựa vào định nghĩa trên, ta có các cách chuyển trạng thái như sau:

+) Không thêm phần tử thứ i vào các dãy con xét tới $i-1$. Vì không thay đổi nên sẽ thừa hưởng $DP[i-1][j]$.

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

+) Thêm phần thử thứ i vào các dãy con xét tới $i-1$, giả sử sau khi thêm $a[i]$ dãy các dãy con có tổng là j thì trước khi thêm $a[i]$, dãy sẽ có tổng là $j-a[i]$.

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

Vậy ta có **công thức QHĐ** như sau:

$$DP[i][j] = DP[i-1][j] + DP[i-1][j-1]$$

26.4 Khởi gán

$DP[0][0] = 1$ với ý nghĩa là có 1 dãy con rỗng nên có tổng bằng 0.

26.5 Đáp án

Đáp án của ta là $DP[n][S]$ với ý nghĩa là số lượng dãy con xét tới hết n phần tử có tổng là S.

26.6 Code mẫu

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

const int MAXN = 105;
const int MAXS = 1005;
const int MOD = 1e9 + 7;

int n , S;
```

```
int a[MAXN];
long long dp[MAXN][MAXS];

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

    dp[0][0] = 1;

    for(int i = 1 ; i <= n ; i++){
        for(int j = 0 ; j <= S ; j++){
            dp[i][j] = dp[i - 1][j];
            if(j - a[i] >= 0)
                dp[i][j] = (dp[i][j] + dp[i - 1][j - a[i]]) % MOD;
        }
    }

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

Đọc code đầy đủ hơn *ở đây*

27 Bài ZA - Chia kẹo

27.1 Nhận xét

Đây là 1 bài toán DP cơ bản áp dụng Knapsack DP.

27.2 Định nghĩa DP

Gọi $DP[i][j]$ là xét tới phần tử thứ i có dãy con có tổng bằng j hay không?

27.3 Suy ra công thức DP

Tương tự như bài toán Knapsack DP, ta có các cách chuyển trạng thái sau:

+) Không thêm phần tử thứ i vào các dãy con xét tới $i-1$. Vì không thay đổi nên sẽ thừa hưởng $DP[i-1][j]$.

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

+) Thêm phần thử thứ i vào các dãy con xét tới $i-1$, giả sử sau khi thêm $a[i]$ dãy các dãy con có tổng là j thì trước khi thêm $a[i]$, dãy sẽ có tổng là $j-a[i]$.

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

Vậy ta có **công thức QHĐ** như sau:

$$DP[i][j] = DP[i-1][j] \ | \ DP[i-1][j-a[i]]$$

27.4 Khởi gán

$DP[0][0] = true$

27.5 Đáp án

Gọi S là tổng của tất cả dãy, ta có đáp án của bài toán là:

$$min(abs(j-(S-j))) \ \forall \ DP[n][j] = true$$

Giải thích cho công thức trên, nếu ta có 1 dãy con có tổng là j, phần còn lại sẽ có tổng là $S-j$ nên chênh lệch của 2 phần sẽ là $abs(j-(S-j))$.

27.6 Code mẫu

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

const int MAXN = 105;
const int MAXS = 1005;
const int MOD = 1e9 + 7;

int n , S = 0;
int a[MAXN];
long long dp[MAXN][MAXS];

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];
        S += a[i];
    }
    dp[0][0] = 1;

    for(int i = 1 ; i <= n ; i++){
        for(int j = 0 ; j <= S ; j++){
            dp[i][j] = dp[i - 1][j];
            if(j - a[i] >= 0){
                if(dp[i - 1][j - a[i]] == true)
```

```
                    dp[i][j] = true;
                }
            }
        }

    int res = 1e9;
    for(int j = 0 ; j <= S ; j++)
        if(dp[n][j] == true)
            res = min(res , abs(j - (S - j)));
    cout << res << endl;
    return 0;
}
```

Đọc code đầy đủ hơn ở *đây*

28 Bài ZB - Tích lớn nhất

28.1 Nhận xét

Đây là 1 bài toán DP cơ bản áp dụng Knapsack DP.

28.2 Định nghĩa DP

Gọi $DP[i][j]$ là xét tới phần tử thứ i có dãy con có tổng bằng j hay không?

28.3 Suy ra công thức DP

Tương tự như bài toán Knapsack DP, ta có các cách chuyển trạng thái sau:

+) Không thêm phần tử thứ i vào các dãy con xét tới $i-1$. Vì không thay đổi nên sẽ thừa hưởng $DP[i-1][j]$.

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

+) Thêm phần thử thứ i vào các dãy con xét tới $i-1$, giả sử sau khi thêm $a[i]$ dãy các dãy con có tổng là j thì trước khi thêm $a[i]$, dãy sẽ có tổng là $j-a[i]$.

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

Vậy ta có **công thức QHĐ** như sau:

$$DP[i][j] = DP[i-1][j] \ | \ DP[i-1][j-a[i]]$$

28.4 Khởi gán

$DP[0][0] = true$

28.5 Đáp án

Gọi S là tổng của tất cả dãy, ta có đáp án của bài toán là:

$$max(j*(S-j)) \ \ \forall \ DP[n][j] = true$$

Giải thích cho công thức trên, nếu ta có 1 dãy con có tổng là j, phần còn lại sẽ có tổng là $S-j$ nên tích của 2 phần sẽ là $j*(S-j)$.

28.6 Code mẫu

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

const int MAXN = 105;
const int MAXS = 1005;
const int MOD = 1e9 + 7;

int n , S = 0;
int a[MAXN];
long long dp[MAXN][MAXS];

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];
        S += a[i];
    }
    dp[0][0] = 1;

    for(int i = 1 ; i <= n ; i++){
        for(int j = 0 ; j <= S ; j++){
            dp[i][j] = dp[i - 1][j];
            if(j - a[i] >= 0){
                if(dp[i - 1][j - a[i]] == true)
                    dp[i][j] = true;
            }
        }
    }

    int res = -1e9;
    for(int j = 0 ; j <= S ; j++)
        if(dp[n][j] == true)
            res = max(res , j * (S - j));
    cout << res << endl;
    return 0;
}
```

Đọc code đầy đủ hơn ở *đây*

29 Bài ZC - Xúc xắc

29.1 Nhận xét

Bài toán này dựa trên việc sử dụng Knapsack DP để đếm số cách.

29.2 Định nghĩa DP

Gọi $DP[i][j]$ là số cách thả i xúc xắc đầu sao cho có tổng là j.

29.3 Suy ra công thức DP

Từ định nghĩa trên, ta thấy việc chuyển trạng thái sẽ dựa trên giá trị của xúc xắc ($1 \to S_i$).

Gọi x là giá trị của mặt nhận được khi thả xúc xắc thứ i và j là tổng của i mặt xúc xắc đầu tiên thì tổng của $i-1$ xúc xắc đầu tiền sẽ là $j-x$. Vậy ta có cách chuyển trạng thái sau:

+) $DP[i][j] += DP[i-1][j-x]$

Vậy ta có **công thức QHĐ** sau:

$$DP[i][j] += DP[i-1][j-x] \quad \forall \ 1 \le x \le S_i$$

29.4 Khởi gán

$DP[0][0] = 1$

29.5 Đáp án

Đáp án của ta là j nhỏ nhất sao cho $DP[n][j]$ lớn nhất.

29.6 Code mẫu

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

using namespace std;

const int MAXN = 11  , MAXSUM = 1005;

int n;
long long S[MAXN]  , dp[MAXN][MAXSUM];

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

```
    cin >> n;
    int sum = 0;
    for(int i = 1 ; i <= n ; i++){
        cin >> S[i];
        sum += S[i];
    }

    dp[0][0] = 1;
    for(int i = 1 ; i <= n ; i++){
        for(int j = 0 ; j <= sum ; j++){
            for(int x = 1 ; x <= S[i] ; x++){
                if(j >= x)dp[i][j] += dp[i - 1][j - x];
            }
        }
    }

    long long res = 0 , value = -1;
    for(int i = 1 ; i <= sum ; i++){
        if(value < dp[n][i]){
            res = i;
            value = dp[n][i];
        }
    }

    cout << res << endl;
    return 0;
}
```

Đọc code đầy đủ hơn ở *đây*

30 Bài ZD - Cái túi

30.1 Nhận xét

Đây là bài toán Knapsack kinh điển.

30.2 Định nghĩa DP

Gọi $DP[i][j]$ là tổng lớn nhất để lấy các vật trong i vật đầu tiền và tổng trọng số là j.

30.3 Suy ra công thức DP

Dựa vào định nghĩa trên, ta có các cách chuyển trạng thái sau:

+) Bỏ qua vật thứ i và có j là tổng trọng số của tập hiện tại

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

+) Lấy vật thứ i và giả sử j là tổng trọng số của tập sau khi thêm vật thứ i thì tổng trọng số của tập trước khi thêm i là $j - w[i]$. Khi đấy, ta cộng thêm giá trị của vật thứ i là $v[i]$ vào tổng giá trị lớn nhất có được

$\Rightarrow DP[i][j] = DP[i-1][j-w[i]] + v[i]$

Vậy ta có **công thức QHĐ** sau:

$$DP[i][j] = max(DP[i-1][j], DP[i-1][j-w[i]] + v[i])$$

30.4 Khởi gán

$DP[0][0] = 0$

30.5 Đáp án

Đáp án của ta sẽ là: $max(DP[n][i])\ \ \forall\ i \le W$

30.6 Code mẫu

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

using namespace std;

const int MAXN = 105 , MAXW = 100005;

int n , S;
long long dp[MAXN][MAXW] , w[MAXN] , v[MAXN];

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

    for(int i = 1 ; i <= n ; i++){
        for(int j = 0 ; j <= S ; j++){
            dp[i][j] = dp[i - 1][j];
            if(j - w[i] >= 0)
                dp[i][j] = max(dp[i][j] , dp[i - 1][j - w[i]] + v[i]);
        }
    }

    long long res = 0;
    for(int i = 1 ; i <= S ; i++)
        res = max(res , dp[n][i]);
```

```
    cout << res << endl;
    return 0;
}
```

Đọc code đầy đủ hơn ở *đây*

31 Bài ZE- Sự yêu đời

31.1 Nhận xét

Đây là 1 bài toán DP cơ bản.

31.2 Định nghĩa DP

Gọi $DP[i][j]$ là dãy con có tổng lớn nhất xét tới vị trí i và đã lấy được j phần tử.

31.3 Suy ra công thức DP

Từ định nghĩa trên, ta có các cách chuyển trạng thái như sau:

+) Không thêm vào phần tử thứ i vào dãy.

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

+) Thêm phần tử thứ i vào dãy, gọi j là số lượng phần tử sau khi thêm phần tử thứ i vào, có $j-1$ là số lượng phần tử trước khi thêm phần tử thứ i, ở đây ta xét thêm trường hợp j chẵn và j lẻ

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

$\Rightarrow DP[i][j] = DP[i-1][j-1] - a[i]$ nếu j chẵn

Vậy ta có **công thức QHĐ** như sau:

$DP[i][j] = max(DP[i-1][j], DP[i-1][j-1] + a[i])$ nếu j lẻ

$DP[i][j] = max(DP[i-1][j], DP[i-1][j-1] + a[i])$ nếu j chẵn

31.4 Khởi gán

$DP[0][0] = 0$

31.5 Đáp án

Đáp án của ta là $max(DP[n][i])\ \ \forall\ i \le n$ với ý nghĩa là dãy con lớn nhất thỏa mãn điều kiện khi xét qua hết n phần tử.

31.6 Code mẫu

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

using namespace std;

const int MAXN = 1005;

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

int main() {
        ios_base::sync_with_stdio(false);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++){
        for(int j = 1 ; j <= n ; j++){
            if(j % 2 == 0)
                dp[i][j] = max(dp[i - 1][j] , dp[i - 1][j - 1] + a[i]);
            else dp[i][j] = max(dp[i - 1][j] , dp[i - 1][j - 1] - a[i]);
        }
    }

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

Đọc code đầy đủ hơn ở *đây*

32 Bài ZF - cnum

32.1 Nhận xét

Dựa vào đề bài, ta có nhận xét, dãy sau cùng sau biến đổi của ta có dạng:

$$1,1,1,...,1,2,2,2,...,2$$

Giả sử x là vị trí cuối cùng của số 1, ta có tổng số thao tác cần sử dụng bao gồm:

+) Số lượng số 2 đứng trước x

+) Số lượng số 1 đứng sau x

32.2 Định nghĩa DP

Dựa vào điều trên, ta dựng nên 2 mảng:

+) Mảng *PrefixSum* đếm số lượng số 2 trong khoảng $[1; i]$ với mỗi i.

+) Mảng *SuffixSum* đếm số lượng số 1 trong khoảng $[i; n]$ với mỗi i.

32.3 Suy ra công thức DP

Gọi $Pref[i]$ là mảng *PrefixSum* cần tính, ta có công thức:

$$Pref[i] = Pref[i-1] + 1 \text{ nếu } a[i] = 2$$
$$Pref[i] = Pref[i-1] \text{ nếu } a[i] = 2$$

Gọi $Suf[i]$ là mảng *SuffixSum* cần tính, ta có công thức:

$$Suf[i] = Suf[i+1] + 1 \text{ nếu } a[i] = 1$$
$$Suf[i] = Suf[i+1] \text{ nếu } a[i] = 2$$

32.4 Khởi gán

$Pref[0] = 0$

$Suf[n+1] = 0$

32.5 Đáp án

Như đã nói ở trên, nếu ta giả sử vị trí x là vị trí của số 1 cuối cùng thì số thao tác của ta là

+) Số lượng số 2 đứng trước $x \Rightarrow Pref[x]$

+) Số lượng số 1 đứng sau $x \Rightarrow Suf[x+1]$

Vậy đáp án của ta sẽ là:

$$min(Pref[x] + Suf[x+1]) \ \ \forall \ x \leq n$$

32.6 Code mẫu

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

const int MAXN = 1e6 + 5;
int n;
int a[MAXN];
int Pref[MAXN] , Suf[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];

    for(int i = 1 ; i <= n ; i++){
        Pref[i] = Pref[i - 1];
        if(a[i] == 2)Pref[i]++;
    }

    for(int i = n ; i > 0 ; i--){
        Suf[i] = Suf[i + 1];
        if(a[i] == 1) Suf[i]++;
    }

    int res = 1e9;
    for(int x = 0 ; x <= n ; x++)
        res = min(res , Pref[x] + Suf[x + 1]);
    cout << res << endl;

    return 0;
}
```

Đọc code đầy đủ hơn ở *đây*

33 Bài ZG - cscong

33.1 Nhận xét

Vời bài toán này, ta có nhận xét $a[i] \leq 10^3$ nến công sai d của ta không vượt quá 10^3.

33.2 Định nghĩa DP

Gọi $DP[i][j]$ là độ dài dãy con cấp số cộng dài nhất kết thúc tại vị trí i và có công sai là j.

33.3 Suy ra công thức DP

Dựa vào định nghĩa trên, ta có cách chuyển trạng thái như sau. Giả sử k là vị trí của số cuối cùng trong dãy đang xét, ta thấy mình chỉ có thể thêm $a[i]$ vào dãy cấp số cộng có công sai là $a[i]-a[k]$.

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

Vậy ta có **công thức QHĐ** như sau:

$$DP[i][a[i]-a[k]] = DP[k][a[i]-a[k]]+1 \quad \forall\ k < i \ \&\ a[k] < a[i]$$

33.4 Khởi gán

$DP[i][j] = 1 \;\; \forall \; i, j$

33.5 Đáp án

Đáp án của ta sẽ là $max(DP[i][j]) \;\; \forall \; i, j$ với ý nghĩa dãy con là cấp số cộng dài nhất kết thúc tại vị trí bất kỳ với công sai dương.

33.6 Code mẫu

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

const int MAXN = 3e3;
const int MAXD = 1e3;
int n;
int a[MAXN + 1];
int dp[MAXN + 5][MAXD + 5];

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];

    for(int i = 1 ; i <= n ; i++)
        for(int j = 1 ; j <= MAXD ; j++)
            dp[i][j] = 1;

    for(int i = 1 ; i <= n ; i++){
        for(int k = 1 ; k < i ; k++){
            if(a[i] > a[k])
                dp[i][a[i] - a[k]] = dp[k][a[i] - a[k]] + 1;
        }
    }

    int res = 0;
    for(int i = 1 ; i <= n ; i++)
        for(int j = 1 ; j <= MAXD ; j++)
            res = max(res , dp[i][j]);
    cout << res << endl;
    return 0;
}
```

Đọc code đầy đủ hơn ở *đây*

34 Bài ZH - Nhật Khôi và những đóa hoa

34.1 Nhận xét

Đay là 1 bài DP chọn dãy thông thường

34.2 Định nghĩa DP

Gọi $DP[i][j]$ là tổng độ cao nhất có được khi xét i bông hoa đầu tiên và đã lấy được j bông hoa.

34.3 Suy ra công thức DP

Giả sử ta có i là bông hoa đang xét, j là số bông hoa trong dãy sau khi thêm bông hoa thứ i và k là bông hoa cuối cùng trong dãy đang xét, ta có cách chuyển trạng thái sau:

+) Thêm bông hoa i vào dãy, với điều kiện màu của bông hoa k khác màu của bông hoa i khác nhau.

$\Rightarrow DP[i][j] = DP[x][j-1] + b[i]$

Vậy ta có **công thức QHĐ**:

$$DP[i][j] = DP[x][j-1] + b[i] \text{ nếu } a[i] \neq a[x]$$

34.4 Khởi gán

$DP[i][1] = b[i] \quad \forall\, i$

34.5 Đáp án

Đáp án của ta là $max(DP[i][k]) \quad \forall\, i$

34.6 Code mẫu

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

const int MAXN = 3e3;
const int MAXD = 1e3;
int n , k;
int a[MAXN + 1] , b[MAXN + 1];
int dp[MAXN + 5][MAXD + 5];

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

```
    int numTest;
    cin >> numTest;

    while(numTest--){
        cin >> n >> k;
        for(int i = 1 ; i <= n ; i++) cin >> a[i] >> b[i];

        for(int i = 0 ; i <= n ; i++)
            for(int j = 0 ; j <= k ; j++)
                dp[i][j] = -1e9;

        for(int i = 1 ; i <= n ; i++){
            dp[i][1] = b[i];
            for(int j = 2 ; j <= k ; j++){
                dp[i][j] = -1e9;
                for(int x = 1 ; x < i ; x++){
                    if(a[i] != a[x])
                        dp[i][j] = max(dp[i][j] , dp[x][j - 1] + b[i]);
                }
            }
        }

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

Đọc code đầy đủ hơn ở *đây*

35 Bài ZI - MODULE

35.1 Nhận xét

Với bài toán này, ta có nhận xét sau:

+) Với giới hạn $n \le 10^3, m \le 10^3$ ta hoàn toàn có thể làm bằng Knapsack DP.

+) Với giới hạn lớn hơn, ta phải có cách làm khác.

Dựa vào việc quan sát, ta có tính chất như sau:

- Với $n \ge m$, luôn luôn tồn tại dãy con chia hết cho m.

Chứng minh:

- Ta sẽ dựa vào bài toán tìm đoạn con chia hết cho m để chứng minh. Gọi $Pref[i]$ là tổng các phần tử từ $1 \rightarrow i$ chia dư cho m, ta nhận thấy nếu tồn tại $Pref[l] \% m = Pref[r] \% m$ thì $(l+1, r)$ là đoạn con có tổng chia hết cho m.

$\Rightarrow$ Tồn tại dãy con chia hết cho m nếu tồn tại $Pref[l] \% m = Pref[r] \% m$.

$\Rightarrow$ nếu không tồn tại đoạn $Pref[l] \% m = Pref[r] \% m$ suy ra tất cả $Pref[i]$ phải phân biệt $\forall i$.

- Nhưng với định lý Dirichlet, với $n >= m$, sẽ luôn tồn tại ít nhất 1 cặp $Pref[l] \% m = Pref[r] \% m$.

$\Rightarrow$ Luôn tồn tại dãy con có tổng chia hết cho m với $n \geq m$.

35.2 Định nghĩa DP

Gọi $DP[i][j]$ là xét tới phần tử thứ i có tồn tại dãy con có tổng là j chia dư cho m hay không?

35.3 Suy ra công thức DP

Từ định nghĩa trên, giả sử j là tổng tất cả các phần tử trong dãy con sau khi thêm $a[i]$, ta có các cách chuyển trạng thái sau:

+) Bỏ qua $a[i]$

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

+) Thêm $a[i]$ vào dãy

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

Vì ta phải áp dụng các phép tính đồng dư nên có công thức QHĐ như sau:

$$DP[i][j] = DP[i-1][j] \mid DP[i-1][(j-a[i]+m) \% m]$$

35.4 Khởi gán

$DP[0][0] = true$

35.5 Đáp án

Đáp án của chúng ta được chia **2 trường hợp** như sau:

+) Nếu $n < m$, ta dùng DP với kết quả là YES nếu $DP[n][0] = true$ và ngược lại là NO.

+) Nếu $n >= m$, kết quả luôn là YES.

35.6 Code mẫu

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

using namespace std;

const int MAXN = 1e6 + 5 , MAXM = 1e3 + 5;

int n , m;
int a[MAXN];
bool dp[MAXM][MAXM];

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

    if(n >= m){
        cout << "YES" << endl;
    }else{
        dp[1][a[1]] = true;
        for(int i = 2 ; i <= n ; i++){
            for(int j = 0 ; j < m ; j++)
                dp[i][j] = dp[i - 1][j] | dp[i - 1][(j - a[i] + m) % m];
            dp[i][a[i]] = true;
        }

        if(dp[n][0] == true)
            cout << "YES" << endl;
        else cout << "NO" << endl;
    }
    return 0;
}
```

Đọc code đầy đủ hơn ở *đây*

36 Bài ZJ - TIENPHAT

36.1 Nhận xét

Yêu cầu của bài toán này là, đặt những chốt chặn vào vị trí của các viên bi sao cho không có viên nào bị mất và tổng tiền phạt là ít nhất.

Để tính được tổng tiền phạt ít nhất, ta sort các viên bi theo tọa độ và chọn vị trí đặt chốt chặn.

Giả sử, các vị trí của chốt chặn là:

$$x_{k_1} \le x_{k_2} \le ... \le x_{k_m}$$

Ta gọi hàm $Sum(l,r)$ là hàm tính tổng các tọa độ của các viên bi từ viên bi thứ l cho tới viên bi thứ r. Ta có thể dùng *PrefixSum* để tính hàm này.

Ta gọi hàm $SumDist(l,r)$, có:

$$SumDist(l,r) = |a[l] - a[l]| + |a[l+1] - a[l]| + ... + |a[r] - a[l]|$$

Vì tọa độ đã được sort tăng dần nên ta phá dấu giá trị tuyệt đối và có phương trình cuối cùng là:

$$SumDist(l,r) = Sum(l,r) - (r - l + 1) * x[l]$$

Vậy khi đó tổng chi phí của ta là:

$$c[k_1] + SumDist(k_1, k_2 - 1) + c[k_2] + SumDist(k_2, k_3 - 1) + ... + c[k_m] + SumDist(k_m, n)$$

Từ việc có tổng chi phí trên, ta thấy bài toán hiện tại là chia n viên bi thành các nhóm sao cho chi phí trên là nhỏ nhất. Để giải quyết bài toán này, ta sử dụng DP.

36.2 Định nghĩa DP

Gọi $DP[i][0 \to 1]$ là tổng chi phí bé nhất đạt được xét tới viên bi thứ i và đặt chốt chặn tại vị trí của viên bi thứ i hay không?

36.3 Suy ra công thức DP

Từ định nghĩa trên, ta có cách các cách chuyển trạng thái như sau:

+) Đặt chốt chặn tại vị trí i, tương ứng với $DP[i][1]$. Vì đặt chốt chặn nên ta tách riêng ra nhóm mới.

$\Rightarrow DP[i][1] = min(DP[i-1][0], DP[i-1][1]) + c[i]$

+) Không đặt chốt chặn tại vị trí i, tương ứng với $DP[i][0]$. Vì không đặt chốt chặn nên ta phải tìm chốt chặn gần nhất để tạo thành 1 nhóm mới.

$\Rightarrow DP[i][0] = min(DP[j-1][1] + SumDist(j,i)) \ \ \forall \ j < i$

Vậy ta có **công thức QHĐ** như sau:

$$DP[i][1] = min(DP[i-1][0], DP[i-1][1]) + c[i]$$

$$DP[i][0] = min(DP[j][1] + SumDist(j,i)) \ \ \forall \ j < i$$

36.4 Khởi gán

$DP[1][0] = \infty$
$DP[1][1] = c[1]$

Vì điểm có tọa độ bé nhất luôn phải là chốt chặn để tránh việc có viên bi nào lăn ra ngoài.

36.5 Đáp án

Đáp án của ta là $min(DP[n][0], DP[n][1])$ với ý nghĩa là xét tới viên bi thứ n, chi phí nhỏ nhất để đặt các chốt chặn sao cho thỏa mãn.

36.6 Code mẫu

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

using namespace std;
#define int long long
const int MAXN = 3e3 + 5;
const int inf = 1e18;

int n;
int x[MAXN] , c[MAXN];
pair<int , int> a[MAXN];
int pref[MAXN];
int DP[MAXN][2];

int Sum(int l , int r){
    return pref[r] - pref[l - 1];
}

int SumDist(int l , int r){
    return Sum(l , r) - (r - l + 1) * x[l];
}

int32_t main(){
    cin >> n;
    for(int i = 1 ; i <= n ; i++)
        cin >> a[i].first >> a[i].second;
    sort(a + 1 , a + 1 + n);
    for(int i = 1 ; i <= n ; i++){
        x[i] = a[i].first;
        pref[i] = pref[i - 1] + x[i];
        c[i] = a[i].second;
    }

    DP[1][0] = inf;
```

```
    DP[1][1] = c[1];

    for(int i = 2 ; i <= n ; i++){
        DP[i][1] = min(DP[i - 1][0] , DP[i - 1][1]) + c[i];
        DP[i][0] = inf;
        for(int j = 1 ; j < i ; j++)
            DP[i][0] = min(DP[i][0] , DP[j][1] + SumDist(j , i));
    }

    cout << min(DP[n][0] , DP[n][1]) << endl;
    return 0;
}
```

Đọc code đầy đủ hơn ở *đây*

37 Bài ZK - Quá kinh điển

37.1 Nhận xét

Như đã thấy trong đề bài, đây là 1 bài Knapsack DP. Tuy nhiên, giới hạn W quá cao, $W \le 10^9$ và nếu sử dụng $DP[i][j]$ là tổng giá trị lớn nhất xét tới vị trí thứ i và đã lấy được tổng trọng số là j thì độ phức tạp bộ nhớ ta sử dụng là $O(n * W) = O(100 * 10^9) = 10^11$ dẫn tới việc bị quá giới hạn bộ nhớ.

Nhưng quan sát giới hạn 1 lần nữa, ta thấy $v_i \le 10^3$ và trường hợp tệ nhất tổng $v_i \le 10^5$ với $n = 100$ và $v_i = 10^3 \quad \forall\ i$.

Với nhận xét trên, ta sử dụng **kĩ thuật đảo nhãn** và thực hiện DP với bài toán này.

37.2 Định nghĩa DP

Gọi $DP[i][j]$ là tổng trọng số nhỏ nhất xét tới vị trí thứ i và đã lấy được tổng giá trị là j.
So sánh 1 chút với bài toán Knapsack nguyên bản, ta có:

+) Đối với bài toán nguyên bản, ta sử dụng $DP[i][j]$ là tổng giá trị lớn nhất xét tới vị trí thứ i và đã lấy được tổng trọng số là j.

+) Đối với bài toán này, ta sử dụng $DP[i][j]$ là tổng trọng số nhỏ nhất xét tới vị trí thứ i và đã lấy được tổng giá trị là j.

Việc ta đảo nhãn tức ta đang đảo vị trí 2 biến tổng giá trị và tổng trọng số sao cho ta có thể phát triển 1 cách DP thích hợp và giải được bài toán.

37.3 Suy ra công thức DP

Từ định nghĩa DP trên và giả sử ta đang xét tới vật thứ i, ta có các cách chuyển đổi trạng thái sau:

+) Thêm vật thứ i vào dãy hiện tại, giả sử j là tổng giá trị sau khi thêm vật thứ i suy ra $j - v[i]$ là tổng giá trị trước khi thêm vật thứ i.

$\Rightarrow DP[i][j] = DP[i-1][j - v[i]] + w[i]$

+) Bỏ qua vật thứ i.

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

Vậy, ta có **công thức QHĐ** như sau:

$$DP[i][j] = min(DP[i-1][j], DP[i-1][j - v[i]] + w[i])$$

37.4 Khởi gán

$DP[0][0] = 0$

37.5 Đáp án

Đáp án của ta là $max(j)\ \ \forall\ DP[n][j] \leq W$ với ý nghĩa tổng giá trị lớn nhất với tổng trọng số nhỏ nhất để đạt được giá trị đấy $\leq W$.

37.6 Code mẫu

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

using namespace std;
#define int long long
const int MAXN = 105 , MAXV = 1e5 + 5;
const int inf = 1e18;

int n , W;
int w[MAXN] , v[MAXN];
int dp[MAXN][MAXV];

int32_t main(){
    cin >> n >> W;
    int sumValue = 0;
    for(int i = 1 ; i <= n ; i++){
        cin >> w[i] >> v[i];
        sumValue += v[i];
    }
```

```
    for(int i = 0 ; i <= n ; i++)
        for(int j = 0 ; j <= sumValue ; j++)
            dp[i][j] = inf;

    dp[0][0] = 0;
    for(int i = 1 ; i <= n ; i++){
        for(int j = 0 ; j <= sumValue ; j++){
            dp[i][j] = dp[i - 1][j];
            if(j - v[i] >= 0)
                dp[i][j] = min(dp[i][j] , dp[i - 1][j - v[i]] + w[i]);
        }
    }

    int res = 0;
    for(int j = 0 ; j <= sumValue ; j++){
        if(dp[n][j] <= W)
            res = j;
    }
    cout << res << endl;
    return 0;
}
```

Đọc code đầy đủ hơn ở *đây*

38 Bài ZL - Wrong Answer on test 2

38.1 Nhận xét

Ta thấy, bài toán này là 1 bài DP cơ bản.

38.2 Định nghĩa DP

Gọi $DP[i][0 \rightarrow]$ là đoan con có tổng lớn nhất kết thúc tại i và đang có trạng thái là:

+) 0 - chưa bắt đầu chọn đoạn con để nhân lên x.

+) 1 - đang chọn đoạn con để nhân lên x.

+) 2 - đã chọn đoạn con để nhân lên x.

38.3 Suy ra công thức DP

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

+) Duy trì việc không chọn đoạn con.

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

+) Thêm phần tử thứ i vào đoạn con đang được chọn và tiếp tục chọn.

$\Rightarrow DP[i][1] = max(DP[i-1][1], DP[i-1][0]) + a[i] * x)$

+) Thêm phần tử thứ i vào đoạn con và kết thúc việc chọn.

$\Rightarrow DP[i][2] = max(DP[i-1][1], DP[i-1][0]) + a[i] * x)$

+) Duy trì trạng thái đã chọn xong đoạn con ở trước đấy.

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

Vậy ta có **công thức QHĐ** như sau:

$$DP[i][0] = max(0, DP[i-1][0] + a[i])$$

$$DP[i][1] = max(0, max(DP[i-1][1], DP[i-1][0]) + a[i] * x))$$

$$DP[i][2] = max(0, DP[i-1][2] + a[i]), max(DP[i-1][1], DP[i-1][0]) + a[i] * x))$$

38.4 Khởi gán

$DP[0][0] = 0$

38.5 Đáp án

Đáp án của ta là $max(DP[i][0], DP[i][1], DP[i][2])\ \forall\ i.$

38.6 Code mẫu

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

#define int long long
const int MAXN = 1e5 + 5;

int n , x;
int a[MAXN];
int dp[MAXN][3];

int32_t main()
{
    ios_base::sync_with_stdio(0);cin.tie(0);cout.tie(0);
    int numTest;
    cin >> numTest;
    while(numTest--){
        cin >> n >> x;
        for(int i = 1 ; i <= n ; i++)
            cin >> a[i];

        for(int i = 1 ; i <= n ; i++){
            dp[i][0] = max(0LL , dp[i - 1][0] + a[i]);
            dp[i][1] = max(0LL , max(dp[i - 1][1] , dp[i - 1][0]) + a[i] * x);
            dp[i][2] = max(0LL , dp[i - 1][2] + a[i]);
            dp[i][2] = max(dp[i][2] , max(dp[i - 1][1] , dp[i - 1][0]) + a[i] * x);
```

```
        }

        int res = 0;
        for(int i = 1 ; i <= n ; i++)
            res = max(res , max(dp[i][0] , max(dp[i][1] , dp[i][2])));
        cout << res << endl;
    }
    return 0;
}
```

Đọc code đầy đủ hơn ở *đây*

39 Bài ZM - Xâu con chung dài nhất

39.1 Nhận xét

Đây là 1 bài toán kinh điển.

39.2 Định nghĩa DP

Gọi $DP[i][j]$ là xâu con chung dài nhất lấy được xét tới phần tử thứ i của xâu s và phần tử thứ j của xâu t.

39.3 Suy ra công thức DP

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

+) Bỏ qua phần tử thứ i của s.

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

+) Bỏ qua phần tử thứ j của t.

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

+) Lấy phần tử thứ i của s và phần tử thứ j của t là 1 cặp chữ cái giống nhau và được thêm vào xâu con chung dài nhất.

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

Vậy ta có **công thức QHĐ** như sau:

$$DP[i][j] = max(DP[i-1][j], DP[i][j-1])$$

$$DP[i][j] = DP[i-1][j-1]+1 \quad \forall\ s[i] = t[j]$$

39.4 Khởi gán

$DP[0][0] = 0$

39.5 Đáp án

Đáp án của ta là $DP[|s|][|t|]$ ($|a|$ là kích thước của xâu a), với ý nghĩa là xâu con chung dài nhất khi xét qua hết tất cả các phần tử của xâu s và xâu t.

39.6 Code mẫu

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

#define int long long
const int MAXN = 3e3 + 5;

string s , t;
int dp[MAXN][MAXN];

int32_t main()
{
    cin >> s >> t;
    s = '%' + s;
    t = '$' + t;

    for(int i = 1 ; i < s.size() ; i++)
        for(int j = 1 ; j < t.size() ; j++){
            dp[i][j] = max(dp[i - 1][j] , dp[i][j - 1]);
            if(s[i] == t[j])
                dp[i][j] = dp[i - 1][j - 1] + 1;
        }

    cout << dp[s.size() - 1][t.size() - 1] << endl;
    return 0;
}
```

Đọc code đầy đủ hơn ở *đây*

40 Bài ZO - Con đường hoa

40.1 Nhận xét

Đây là 1 bài tập cơ bản ứng dụng của bài toán tìm xâu con chung lớn nhất. Xét về mặt cơ bản, bài toán xâu con chung là tìm các cặp chỉ số theo thứ tự sao cho số lượng cặp số là nhiều nhất. Thay thế vào bài toán hiện tại, ta cần tìm số cặp sao cho tổng của cặp số không âm.

40.2 Định nghĩa DP

Gọi $DP[i][j]$ là số cặp nhiều nhất chọn được xét tới phần tử thứ i của dãy A và phần tử thứ j của dãy B.

40.3 Suy ra công thức DP

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

+) Bỏ qua phần tử thứ i trong dãy A.

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

+) Bỏ qua phần tử thứ j trong dãy B.

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

+) Chọn thêm cặp (i, j) nếu $A[i] + b[j] \geq 0$.

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

Vậy ta **có công thức QHĐ** như sau:

$$DP[i][j] = max(DP[i-1][j], DP[i][j-1])$$

$$DP[i][j] = DP[i-1][j-1] + 1 \quad \forall\ A[i] + B[j] \geq 0$$

40.4 Khởi gán

$DP[0][0] = 0$

40.5 Đáp án

Đáp án của ta là $DP[n][m]$ với ý nghĩa là số cặp nhiều nhất chọn được khi xét hết n phần tử của dãy A và n phần tử của dãy B.

40.6 Code mẫu

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

#define int long long
const int MAXN = 3e3 + 5;

int n;
int a[MAXN] , b[MAXN];
int dp[MAXN][MAXN];

int32_t main()
{
    cin >> n;
```

```
    for(int i = 1 ; i <= n ; i++)
        cin >> a[i] >> b[i];

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

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

Đọc code đầy đủ hơn ở *đây*

41 Bài ZP - dzero

41.1 Nhận xét

Đây là 1 bài DP cơ bản.

41.2 Định nghĩa DP

Gọi $DP[i]$ là số thao tác tối thiểu để biến đổi từ i thành 0.

41.3 Suy ra công thức DP

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

+) Thực hiện thao tác 1, giảm giá trị của i đi 1:

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

+) Thực hiện thao tác 2, tách i thành số lớn hơn trong 1 cặp số có tích bằng i:

$\Rightarrow DP[i] = DP[max(a,b)]+1 \ \ \forall \ a * b = i$

Vậy ta có **công thức QHĐ** như sau:

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

$$DP[i] = min(DP[max(a,b)])+1 \ \ \forall \ a * b = i$$

Note: Việc tính $DP[i]$ dựa trên thao tác 2 có thể thực hiện bằng cách duyệt ước của i trong $O(\sqrt{i})$.

41.4 Khởi gán

$DP[0] = 0$

41.5 Đáp án

Đáp án của ta là $DP[x]$ với mỗi x từ input.

41.6 Code mẫu

```
#include<bits/stdc++.h>
using namespace std;
const int MAXN = 1e6 ;

int dp[MAXN + 1];

int main()
{
    ios_base::sync_with_stdio(0);cin.tie(0);cout.tie(0);
    for(int i = 1 ; i <= MAXN ; i++){
        dp[i] = dp[i - 1] + 1;
        for(int a = 2 ; a * a <= i ; a++){
            if(i % a != 0) continue;
            int b = i / a;
            dp[i] = min(dp[i] , dp[max(a , b)] + 1);
        }
    }

    int numTest;
    cin >> numTest;
    while(numTest--){
        int x;
        cin >> x;
        cout << dp[x] << "\n";
    }
}
```

Đọc code đầy đủ hơn ở *đây*

42 Bài ZQ - Xóa số

42.1 Nhận xét

Đây là 1 bài toán DP cơ bản. Tuy nhiên, để giải bài toán một cách thuận tiện, ta thay vì tìm số phần tử cần xóa ít nhất, ta tìm số phần tử lớn nhất có thể giữ lại.

42.2 Định nghĩa DP

Gọi $DP[i]$ là số lượng số nhiều nhất chọn được với số lớn nhất được chọn là i.
Gọi $cnt[i]$ là số lần xuất hiện của giá trị i.

42.3 Suy ra công thức DP

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

+) Thêm tất cả giá trị i và 1 tập có số lớn nhất là ước của i.

$\Rightarrow DP[i] = DP[x] + cnt[i] \;\; \forall\, x \vdots i$

Vậy **công thức QHĐ** của ta là:

$$DP[i] = max(DP[x] + cnt[i]) \;\; \forall\, x \vdots i$$

Tuy nhiên: Nếu ta tính toán độ phức tạp, trường hợp tệ nhất $O(10^6 * \sqrt{10^6})$, sẽ bị **TLE**. Nên thay vì duyệt ước của i, ta duyệt bội của số x. Sử dụng Dp Push hay còn gọi là DP tương lai.

42.4 Khởi gán

$DP[1] = cnt[1]$

42.5 Đáp án

Vì ta đảo ngược bài toán thành số lượng phần tử nhiều nhất có thể giữ lại. Ta có đáp án của ta là

$$n - max(DP[i]) \;\; \forall\, i$$

42.6 Code mẫu

```cpp
#include<bits/stdc++.h>
using namespace std;
const int MAXN = 1e6 ;

int n , cnt[MAXN + 1] , dp[MAXN + 1];

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

    dp[1] = cnt[1];
    for(int i = 1 ; i <= MAXN ; i++){
        for(int j = 2 * i ; j <= MAXN ; j += i)
            dp[j] = max(dp[j] , dp[i] + cnt[j]);
    }
```

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

Đọc code đầy đủ hơn ở *đây*

43 Bài ZR - Perfect balance as all things should be

43.1 Nhận xét

Đây là 1 bài toán DP cơ bản.

43.2 Định nghĩa DP

Gọi $DP[i][j]$ là số sinh viên tối đa có thể tham gia cuộc thi xét tới sinh viên thứ i và đã chia thành j nhóm sao cho thỏa mãn điều kiện.

Tuy nhiên, ta phải kiểm soát min và max. Khi đấy ta sort lại mảng, với 1 nhóm có các sinh viên trong khoảng $[L,R]$ trong mảng đã được sort thì sinh viên cao nhất là R và sinh viên thấp nhất là L. Từ đây, việc kiểm soát điều kiện của ta trở nên dễ dàng hơn

43.3 Suy ra công thức DP

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

+) Thêm 1 nhóm là các sinh viên trong đoạn $[i,j]$ và có t là số nhóm sau khi thêm nhóm $[i,j]$.

$\Rightarrow DP[i][t] = DP[j-1][t-1] + (i-j+1) \ \ \forall \ a[i]-a[j] \le x$

+) Bỏ qua sinh viên thứ i.

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

Vậy ta có **công thức QHĐ** như sau:

$$DP[i][t] = max(DP[i-1][t], DP[j-1][t-1] + (i-j+1)) \ \ \forall \ a[i]-a[j] \le x$$

Tuy nhiên, với công thức trên ta vẫn chưa thể **AC** được bài. Vì đọ phức tạp của công thức trên lên tới $O(n^2 * k) \Rightarrow$ TLE.

Vì vậy, ta phải tìm cách tối ưu công thức DP của ta. Theo đề bài, ta nhận xét rằng việc chia nhóm của ta với mỗi sinh viên độ quan trọng đều như nhau. Nên khi lấy 1 nhóm, ta cố gắng lấy càng nhiều càng tốt và giả sử ta lấy từ sinh viên i thì lấy tới sinh viên j có j bé nhất thỏa mãn $a[i]-a[j] >= x$. Để số sinh viên trong nhóm đó là nhiều nhất.

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

$$DP[i][t] = max(DP[i-1][t], DP[opt-1][t-1] + (i - opt + 1))$$

Với opt là vị trí nhỏ nhất có thỏa mãn $a[i] - a[opt] <= x$.

Việc tìm opt, có thể giải quyết bằng 2-Pointer hoặc tìm kiếm nhị phân với độ phức tạp tương ứng là $O(n * k)$ **với 2-Pointer** và $O(n * k * log(n))$ **với tìm kiếm nhị phân**

43.4 Khởi gán

$DP[0][0] = 0$

43.5 Đáp án

Đáp án của ta là $max(DP[n][i])\ \ \forall\ i \le k$.

43.6 Code mẫu

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

const int MAXN = 5e3 + 5;

int n , k , x;
int a[MAXN];
int dp[MAXN][MAXN];

int32_t main()
{
    ios_base::sync_with_stdio(false);
    cin.tie(0);
    cout.tie(0);

    cin >> n >> k >> x;
    for(int i = 1 ; i <= n ; i++) cin >> a[i];
    sort(a + 1 , a + 1 + n);

    dp[0][0] = 0;
    for(int t = 1 ; t <= k ; t++){
        int opt = 1;
        for(int i = 1 ; i <= n ; i++){
            while(a[i] - a[opt] > x)
                opt++;

            dp[i][t] = max(dp[i - 1][t] , dp[opt - 1][t - 1] + (i - opt + 1));
```

```
        }
    }

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

Đọc code đầy đủ hơn ở *đây*

44 Bài ZS - Lại là mua quà

44.1 Nhận xét

Quan sát bài toán, ta nhận thấy việc mua đồ với chi phí ít nhất tức là ta đang cố gắng đạt được tiền được miễn phí là nhiều nhất có thể.

Đối với bài toán này, ta có nhận xét như sau. Với 1 dãy bất kì, giữa việc chọn dãy gồm 20 phần tử và chọn 2 dãy 10 phần tử thì việc chọn 2 dãy có 10 phần tử sẽ luôn tốt hơn. Ví dụ với dãy:

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20

Nếu chọn 1 nhóm có 20 phần tử thì ta lấy như sau:

[1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20]

Thì ta được miễn phí 1 và 2.

Nếu chọn 2 nhóm có 10 phần tử thì ta lấy như sau:

[1 2 3 4 5 6 7 8 9 10] [11 12 13 14 15 16 17 18 19 20]

Thì ta được miễn phí 1 và 11 nghĩa là sẽ tốt hơn.

Giải thích: Vì ta thấy nếu chọn 20 phần tử, ta được miễn phí 2 số nhỏ nhất trong dãy. Còn chọn 2 dãy 10 phần tử, ta được miễn phí số nhỏ nhất trong 10 số đầu và số nhỏ nhất trong 10 số sau. Vì thế, tổng miễn phí của 20 phần tử luôn ≤ tổng miễn phí của 2 dãy 10 phần tử.

44.2 Định nghĩa DP

Gọi $DP[i]$ là số tiền miễn phí nhiều nhất xét tới món hàng thứ i.

44.3 Suy ra công thức DP

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

+) Bỏ qua phần tử thứ i.

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

+) Chọn thêm 1 dãy bao gồm 10 phần tử là các phần tử trong đoạn $[i-9;i]$.

$\Rightarrow DP[i] = DP[i-10] + min(a[j])\ \ \forall\ i-9 \le j \le i$

Vậy ta có **công thức QHĐ** như sau:

$$DP[i] = max(DP[i-1], DP[i-10] + min(a[j]))\ \ \forall\ i-9 \le j \le i$$

44.4 Khởi gán

$DP[0 \to 9] = 0$

44.5 Đáp án

Đáp án của ta là tổng n phần tử - $DP[n]$ với ý nghĩa là tổng số tiền phải trả trừ đi số tiền được miễn phí nhiều nhất $\Rightarrow$ số tiền phải trả ít nhất.

44.6 Code mẫu

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

const int MAXN = 1e5 + 5;

int n;
int a[MAXN] , dp[MAXN];

int32_t main()
{
    ios_base::sync_with_stdio(false);cin.tie(0);cout.tie(0);

    int total = 0;

    cin >> n;
    for(int i = 1 ; i <= n ; i++){
        cin >> a[i];
        total += a[i];
    }
    for(int i = 10 ; i <= n ; i++){
        int Min = 1e9 + 5;
        for(int j = i - 9 ; j <= i ; j++)
```

```
            Min = min(Min , a[j]);
        dp[i] = max(dp[i - 1] , dp[i - 10] + Min);
    }

    cout << total - dp[n] << endl;

    return 0;
}
```

Đọc code đầy đủ hơn ở *đây*

45 Bài ZT - Bốc quà

45.1 Nhận xét

Đối với bài này, ta có nhận xét, việc lấy 2 đoạn thẳng không giao nhau sẽ luôn là tốt nhất cho ta. Khi lấy 2 đoạn giao nhau, thay vì mất đi độ dài vào phần giao, ta có thể tối ưu phương án bằng cách di chuyển sao cho không giao nhau. Khi đấy phương án của ta sẽ tối ưu hơn.

Nhìn chung, bài toán của ta là bài toán chia nhóm sao cho có nhiều phần tử được chọn nhất. Ta sẽ sort lại các tọa độ tăng dần cho việc quản lý tọa độ được và đếm số quá nằm trong đoạn được thuận tiện hơn.

45.2 Định nghĩa DP

Gọi $DP[i][1 \to 2]$ là số lượng quà nhiều nhất lấy được xét tới i, có $1 \to 2$ đoạn thẳng đã được đặt.

45.3 Suy ra công thức DP

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

+) Đặt thêm 1 đoạn thẳng kết thúc tại $a[i]$ và có j là vị trí của tọa độ đầu tiên $\geq a[i]-k$.

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

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

+) Không đặt thêm đoạn thẳng nào tại $a[i]$.

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

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

Vậy ta có **công thức QHĐ** như sau:

$$DP[i][1] = max(DP[i-1][1], i - j + 1)$$

$$DP[i][2] = max(DP[i-1][2], DP[j-1] + (i - j + 1))$$

Với j là phần tử đầu tiên $\geq a[i]-k$. Việc tìm phần tử j với mỗi i có thể thực hiện bằng 2-Pointer hoặc tìm kiếm nhị phân.

45.4 Khởi gán

$DP[0] = 0$

45.5 Đáp án

Đáp án của ta là $DP[n][2]$ với ý nghĩa là số lượng phần tử lớn nhất lấy được với 2 đoạn đã được đặt và đã xét qua n phần tử.

45.6 Code mẫu

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

const int MAXN = 1e6 + 5;

int n , k;
int a[MAXN] , dp[MAXN][3];

int32_t main()
{
    ios_base::sync_with_stdio(false);cin.tie(0);cout.tie(0);


    cin >> n >> k;
    for(int i = 1 ; i <= n ; i++)
        cin >> a[i];

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

    int j = 1;
    for(int i = 1 ; i <= n ; i++){
        while(a[j] < a[i] - k) j++;
        dp[i][1] = max(dp[i - 1][1] , i - j + 1);
        dp[i][2] = max(dp[i - 1][2] , dp[j - 1][1] + (i - j + 1));
    }
    cout << dp[n][2] << endl;

    return 0;
}
```

Đọc code đầy đủ hơn ở *đây*

46 Bài ZU - DGIFT

46.1 Nhận xét

Quan sát đề bài, ta có nhận xét như sau. Việc chọn 1 nhóm có nhiều hơn k món quà sẽ không tối ưu vì những món quá thừa ra ấy có thể chuyển qua nhóm khác để được nhiều món quà hơn.

Vì vậy, bài toán của ta là bài toán chia thành các nhóm gồm K phần tử liên tiếp sao cho tổng giá trị được thưởng là nhiều nhất.

46.2 Định nghĩa DP

Gọi $DP[i]$ là giá trị phần thưởng lớn nhất xét tới phần thưởng thứ i.

46.3 Suy ra công thức DP

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

+) Chọn thêm 1 đoạn độ dài K và kết thúc ở i ($[i-K+1, I]$).

$\Rightarrow DP[i] = DP[i-K] + min(a[j]) \;\; \forall \; i-K+1 \le j \le i$

+) Bỏ qua món quà thứ i

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

Vậy ta có công thức QHĐ như sau:

$$DP[i] = max(DP[i-1], DP[i-k] + min(a[j])) \;\; \forall \; i-K+1 \le j \le i$$

Như ta thấy, việc tìm $Min(a[j])$ nếu chỉ sử dụng vòng for sẽ dẫn đến việc TLE vì độ phức tạp quá cao. Lúc này, ta sẽ nghĩ đến việc sử dụng các kĩ thuật xử lí bài toán RMQ như Sparse Table, Segment Tree,... Nhưng ta sẽ sử dụng **Sparse Table** vì nó dễ cài đặt và dễ hiểu.

Hướng dẫn sơ về Sparse Table:

- Sparse Table là 1 kĩ thuật từ quy hoặc động dùng để quản lý các đoạn theo độ dài là lũy thừa của 2

- Gọi $ST[i][j]$ là giá trị của phần tử nhỏ nhất xuất hiện trong đoạn $[i; i+2^j-1]$. Từ đó, ta có công thức xác định $ST[i][j]$ như sau:

$$ST[i][j] = min(ST[i][j-1] \; , \; ST[i+2^{j-1}][j-1])$$

- Vậy làm sao để dùng mảng ST để tìm min trong đoạn $[L;R]$?. Như ta biết, mọi số tự nhiên đều có thể được phân tích thành tổng các lũy thừa của 2. Áp dụng vào bài toán, nếu ta coi độ dài của đoạn $[L;R]$ là 1 số nhị phân thì ta có thể tìm min trong đoạn bằng cách lấy min của từng đoạn bé hơn nằm trong $[L;R]$.

- Lấy ví dụ ta có đoạn [2;6] với độ dài là 5 chuyển sang số nhị phân là 101. Thì ta lấy các đoạn như sau:

+) $[2;2+2^0-1]-[2;2]$ tương ứng với $ST[2][0]$

+) $[2+2^0;2+2^0+2^2-1]-[3;6]$ - tương ứng với $ST[3][2]$

- Tới đây, ta có thể rút ra được cách lấy các đoạn sao cho bao phủ hết tât cả phần tử trong đoạn $[L;R]$.

46.4 Khởi gán

$DP[0 \to K-1] = 0$

$ST[i][0] = a[i] \;\; \forall \; i$

46.5 Đáp án

Đáp án của ta là $DP[n]$ với ý nghĩa là tổng giá trị phần thưởng nhiều nhất lấy được khi xét qua hết n phần tử.

46.6 Code mẫu

```cpp
#include <bits/stdc++.h>
using namespace std;
#define int long long

const int inf = 1e18;
const int MAXN = 2e5 + 5;

int n , k;
int a[MAXN] , dp[MAXN] , ST[MAXN][20];

int Query(int L , int R){
    int cur = L , result = inf;
    int length = R - L + 1;
    for(int i = 0 ; i <= 20 ; i++){
        if(!((length >> i) & 1))
            continue;
        result = min(result , ST[cur][i]);
        cur = cur + (1 << i);
    }
    return result;
```

```
}

int32_t main()
{
    ios_base::sync_with_stdio(false);cin.tie(0);cout.tie(0);
    cin >> n >> k;
    for(int i = 1 ; i <= n ; i++){
        cin >> a[i];
        ST[i][0] = a[i];
    }

    for(int j = 1 ; j < 20 ; j++){
        for(int i = 1 ; i + (1 << (j - 1)) <= n ; i++)
            ST[i][j] = min(ST[i][j - 1] , ST[i + (1 << (j - 1))][j - 1]);
    }

    for(int i = k ; i <= n ; i++)
        dp[i] = max(dp[i - 1] , dp[i - k] + Query(i - k + 1 , i));
    cout << dp[n] << endl;
    return 0;
}
```

Đọc code đầy đủ hơn ở *đây*

47 Bài ZV - Công viên 1

47.1 Nhận xét

Đây là 1 bài toán DP khá cơ bản.

47.2 Định nghĩa DP

Gọi $DP[i][j]$ là thời gian nhỏ nhất để 2 bạn đi sao cho Bờm đang đứng ở điểm thứ i và Cuội đang đứng ở điểm thứ j và các địa điểm đã đi qua là từ $1 \to max(i, j)$.

47.3 Suy ra công thức DP

Lưu ý: Ta sẽ xài DP Push. Từ định nghĩa trên, ta có các cách chuyển trạng thái:

+) Cho Bờm di chuyển. Tới đây, ta lại có 2 trường hợp như sau:

- Vị trí i của Bờm lớn hơn vị trí j của Cuội và theo định nghĩa DP của ta, Bờm chỉ có thể di chuyển tới vị trí $i+1$.

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

- Vị trí i của Bờm bé hơn vị trí j của Cuội, vì thế nên Bờm có thể di chuyển trong khoảng $[i+1; j+1]$.

$$\Rightarrow DP[k][j] = DP[i][j] + a[i][k] \ \ \forall\ i+1 \le k \le j+1$$

+) Tương tự, cho Cuội di chuyển. Tới đây, ta lại có 2 trường hợp như sau:

- Vị trí j của Cuội lớn hơn vị trí i của Bờm.

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

- Vị trí j của Cuội bé hơn vị trí i của Bờm và vì thế nên Cuội có thể di chuyển trng khoảng $[j+1; i+1]$.

$\Rightarrow DP[i][k] = DP[i][j] + a[j][k] \;\; \forall \; j+1 \le k \le i+1$

Vậy ta sẽ sử dụng **DP Push** với các công thức chuyển trạng thái như sau:

- Nếu $i \ge j$
  - $DP[i+1][j] = max(DP[i+1][j],\; DP[i][j] + a[i][i+1])$
  - $DP[k][j] = max(DP[k][j],\; DP[i][j] + a[i][k]) \;\; \forall \; i+1 \le k \le j+1$
- Nếu $i \le j$
  - $DP[i][j+1] = max(DP[i][j+1],\; DP[i][j] + a[j][j+1])$
  - $DP[i][k] = max(DP[i][k],\; DP[i][j] + a[j][k]) \;\; \forall \; j+1 \le k \le i+1$

47.4 Khởi gán

$DP[1][1] = 0$

47.5 Đáp án

Đáp án của ta là:

$$min(min(DP[i][n], DP[n][i]) + a[i][1] + a[n][n]) \;\; \forall \; i \le n$$

47.6 Code mẫu

```
#include<bits/stdc++.h>
using namespace std;
#define int long long
const int MAXN = 205;
const int inf = 1e18;

int n;
int a[MAXN][MAXN];
int dp[MAXN][MAXN];

int32_t main(){
    cin >> n;
    for(int i = 1 ; i <= n ; i++)
        for(int j = 1 ; j <= n ; j++)
            cin >> a[i][j];
```

```
    for(int i = 0 ; i <= n + 1 ; i++)
        for(int j = 0 ; j <= n + 1 ; j++)
            dp[i][j] = inf;

    dp[1][1] = 0;
    for(int i = 1 ; i <= n ; i++){
        for(int j = 1 ; j <= n ; j++){
            if(i == min(i , j)){
                for(int k = i + 1 ; k <= j + 1 ; k++)
                    dp[k][j] = min(dp[k][j] , dp[i][j] + a[i][k]);
                dp[i][j + 1] = min(dp[i][j + 1] , dp[i][j] + a[j][j + 1]);
            }

            if(j == min(i , j)){
                for(int k = j + 1 ; k <= i + 1 ; k++)
                    dp[i][k] = min(dp[i][k] , dp[i][j] + a[j][k]);
                dp[i + 1][j] = min(dp[i + 1][j] , dp[i][j] + a[i][i + 1]);
            }
        }
    }

    int res = inf;
    for(int i = 1 ; i <= n ; i++)
        res = min(res , min(dp[i][n], dp[n][i]) + a[i][1] + a[n][1]);
    cout << res << endl;
    return 0;
}
```

Đọc code đầy đủ hơn ở *đây*

48 Bài ZX - Chuỗi đối xứng

48.1 Nhận xét

Đây là 1 bài DP cơ bản.

48.2 Định nghĩa DP

Gọi $DP[i][j]$ là độ dài của xâu con palindrome dài nhất thu được từ đoạn (i,j).

48.3 Suy ra công thức DP

Từ định nghĩa trên, ta có các chuyển trạng thái sau:

- Bỏ qua kí tự i.

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

- Bỏ qua kí tự j.
  $\Rightarrow DP[i][j] = DP[i][j-1]$
- Thêm kí tự i vào đầu xâu và kí tự j vào cuối xâu nếu giống nhau nhau
  $\Rightarrow DP[i][j] = DP[i+1][j-1]+2$

Vậy ra có công thức DP như sau:

$$DP[i][j] = max(DP[i+1][j], DP[i][j-1])$$

$$DP[i][j] = max(DP[i][j], DP[i+1][j-1]+2)\ \forall\ s[i] = s[j]$$

48.4 Khởi gán

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

48.5 Đáp án

Đáp án của ta là $DP[1][n]$ với định nghĩa là độ dài xâu con là palindrome dài nhất thu được trong đoạn $[1;n]$.

48.6 Code mẫu

```
#include<bits/stdc++.h>
using namespace std;
#define int long long
const int MAXN = 2005;
const int inf = 1e18;

string s;
int dp[MAXN][MAXN];

int32_t main(){
    cin >> s;
    s = '&' + s;
    int n = (int)s.size() - 1;

    for(int len = 1 ; len <= n ; len++){
        for(int i = 1 ; i + len - 1 <= n ; i++){
            int j = i + len - 1;

            if(i == j){
                dp[i][j] = 1;
            }else{
                dp[i][j] = max(dp[i + 1][j] , dp[i][j - 1]);
                if(s[i] == s[j])
                    dp[i][j] = max(dp[i][j] , dp[i + 1][j - 1] + 2);
```

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

Đọc code đầy đủ hơn ở *đây*

49 Bài ZY - Chất nhờn

49.1 Nhận xét

Đây là 1 bài toán DP cơ bản.

49.2 Định nghĩa DP

Gọi $DP[i][j]$ là chi phí nhỏ nhất để ghép các khối trong đoạn $[i;j]$ thành một khối.

49.3 Suy ra công thức DP

Từ định nghĩa trên và đề bài, ta có cách chuyển trạng thái như:

+) Ghép 2 khối được ghép từ đoạn $[i;k]$ và $[k+1;j]$ lại cùng nhau.

$\Rightarrow DP[i][j] = DP[i][k] + DP[k+1][j] + Sum(i,j)$

Với $Sum(i,j)$ là tổng các khối trong đoạn $[i;j]$ và ta có thể dùng **PrefixSum** để tính.

Vậy ta có công thức QHĐ như sau:

$$DP[i][j] = min(DP[i][j], DP[i][k] + DP[k+1][j] + Sum(i,j))$$

49.4 Khởi gán

$DP[i][i] = 0 \ \ \forall\ i \le n$

49.5 Đáp án

Đáp án của ta là $DP[1][n]$ với định nghĩa là chi phí nhỏ nhất để ghép hết các đoạn trong khoảng $[1;n]$ thành 1 khối.

49.6 Code mẫu

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

```
const int MAXN = 405;
const int inf = 1e18;

int n;
int dp[MAXN][MAXN];
int a[MAXN] , pref[MAXN];

int Sum(int l , int r){
    return pref[r] - pref[l - 1];
}

int32_t main(){
    cin >> n;
    for(int i = 1 ; i <= n ; i++){
        cin >> a[i];
        pref[i] = pref[i - 1] + a[i];
    }
    for(int len = 1 ; len <= n ; len++){
        for(int i = 1 ; i + len - 1 <= n ; i++){
            int j = i + len - 1;
            dp[i][j] = inf;
            if(len == 1){
                dp[i][j] = 0;
            }else{
                for(int k = i ; k < j ; k++)
                    dp[i][j] = min(dp[i][j] , dp[i][k] + dp[k + 1][j] + Sum(i , j));
            }
        }
    }

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

Đọc code đầy đủ hơn ở *đây*

50 Bài ZZ - NEGIKO

50.1 Nhận xét

Yêu cầu của bài toán đếm số cách chọn ra 2 dãy độ dài k từ mảng a và mảng b sao cho phần tử lớn thứ i được chọn từ dãy a lớn hơn phần tử lớn thứ i được chọn từ dãy b.

Không mất tính tổng quát, nếu ta sort lại dãy a và dãy b tăng dần, bài toán thu hẹp lại về đếm số cách chọn k cặp $a[i] > b[j]$.

50.2 Định nghĩa DP

Với nhận xét trên, ta gọi $DP[i][j][t]$ là số cách chọn được t cặp bài xét tới là bài thứ i của dãy a và là bài thứ j của dãy b.

50.3 Suy ra công thức DP

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

+) Bỏ qua phần tử thứ i của dãy a.

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

+) Bỏ qua phần tử thứ j của dãy b.

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

Lưu ý: Với 2 cách chuyển trạng thái trên, ta nhận thấy vấn đề như sau:

- $DP[i][j-1][t] = DP[i-1][j-1][t] + DP[i][j-2][t]$
- $DP[i-1][j][t] = DP[i-1][j-1][t] + DP[i-2][j][t]$.

Nếu $DP[i][j][t] = DP[i-1][j][t] + DP[i][j-1][t]$ thì phân tích ra sẽ có 2 lần $DP[i-1][j-1][t]$ bị lặp lại.

Phân tích: $DP[i][j] = 2 * DP[i-1][j-1][t] + DP[i-2][j][t] + DP[i][j-2][t]$.

Nên vì vậy, ta phải trừ đi 1 lần $DP[i-1][j-1][t]$.

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

+) Chọn thêm cặp $(a[i], b[j])$ nếu $a[i] > b[j]$.

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

Vậy ta có công thức QHĐ như sau:

$$DP[i][j][t] += DP[i-1][j][t] + DP[i][j-1][t] - DP[i-1][j-1][t]$$

$$DP[i][j][t] += DP[i-1][j-1][t-1] \ \ \forall\ a[i] > b[j]$$

50.4 Khởi gán

$DP[i][j][0] = 1 \ \ \forall\ i, j$

50.5 Đáp án

Đáp án của ta là $DP[n][m][k]$ với định nghĩa là số cách chọn dãy thỏa mãn với độ dài bằng k sau khi xét qua n phần tử của dãy a và m phần tử của dãy b.

50.6 Code mẫu

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

const int MAXN = 1e3 + 5;
const int inf = 1e18;
const int MOD = 1e9 + 9;

int n , m , k;
int a[MAXN] , b[MAXN];
int dp[MAXN][MAXN][15];


int32_t main(){
    ios_base::sync_with_stdio(0);cin.tie(0);cout.tie(0);

    cin >> n >> m >> k;
    for(int i = 1 ; i <= n ; i++) cin >> a[i];
    for(int i = 1 ; i <= m ; i++) cin >> b[i];

    sort(a + 1 , a + 1 + n);sort(b + 1 , b + 1 + m);

    for(int i = 0 ; i <= n ; i++)
        for(int j = 0 ; j <= m ; j++) dp[i][j][0] = 1;

    for(int t = 1 ; t <= k ; t++){
        for(int i = 1 ; i <= n ; i++){
            for(int j = 1 ; j <= m ; j++){
                dp[i][j][t] = (dp[i][j - 1][t] + dp[i - 1][j][t]) % MOD;
                dp[i][j][t] = (dp[i][j][t] - dp[i - 1][j - 1][t] + MOD * MOD) % MOD;
                if(a[i] > b[j])dp[i][j][t] = (dp[i][j][t] + dp[i - 1][j - 1][t - 1]) % MOD
            }
        }
    }

    cout << dp[n][m][k] << endl;

    return 0;
}
```

Đọc code đầy đủ hơn *ở đây*

51 Bài ZZA - Chip Move

51.1 Nhận xét

Đây là 1 bài toán DP khá khó và yêu có kĩ thuật tối ưu và chứng minh toán học.

51.2 Định nghĩa DP

- Ta sử dụng DP Push, gọi $DP[i][j]$ là số cách để đi tới điểm thứ i tại thời điểm j.

51.3 Suy ra công thức DP

- Từ định nghĩa trên, ta có công thức như sau:

$$DP[i][j] += DP[x][j-1] \quad \forall\ (i-x)\ \%(k+j-1)\ = 0$$

- Với x là điểm nhảy tới trong bước thứ $j+1$ và như yêu cầu đề bài, khoảng cách giữa x và i phải chia hết cho $(k+j-1)$. Độ phức tạp hiện tại của ta nếu áp dụng thô công thức vào là $O(n^2 * k)$. Nên ta áp dụng **PrefixSum**.

- Gọi $Pref[t]$ là tổng các $DP[x][j-1]$ xét tới vị trí i và bước j sao cho $x\ \%\ (k+j-1)\ = t$. Tới đây, ta đã cải tiến được công thức của ta như sau:

$$DP[i][j] = Pref[i\ \%\ (k+j-1)]$$

$$Pref[i\ \%\ (k+j-1)] += DP[i][j-1]$$

- Tới đây, đa phần mọi người sẽ nghĩ tới việc giải bài toán $O(n*k)$. Nhưng nếu nhìn kĩ hơn, ta có nhận xét cho bài này như sau. Ta nhận thấy sẽ không bao giờ phải nhảy quá $\sqrt{\frac{2n}{k}}$ bước nhưng với trường hợp tệ nhất là $n = 2*10^5,\ k = 1$ thì có thể nhảy nhiều nhất $\sqrt{\frac{2n}{k}} = \frac{\sqrt{2*2*10^5}}{1} = 672$ bước. Vì vậy độ phức tạp tổng quát của ta là $O(\sqrt{n}*n)$.

Ta chứng minh việc không phải nhảy quá 672 bước như sau. Giả sử m là số bước hiện tại của ta, có:

$$k\ +\ (k+1)\ +\ ..\ +\ (k+m-1) \le n$$
$$\Leftrightarrow k\ *\ (1\ +\ 2\ +\ ...\ +\ m)\ \le\ n$$
$$\Leftrightarrow k * \frac{m*(m+1)}{2} \le n$$
$$\Leftrightarrow m\ \le\ \sqrt{\frac{2n}{k}}$$

Vì vậy số bước của ta phải luôn bé hơn hoặc bằng 672.

Tiếp đến, ta phải xét tới độ phức tạp bộ nhớ của chúng ta. Cùng với độ phức tạp thời gian, bộ nhớ của ta chiếm tối đa $O(672*n) \sim 10^8$, nên sẽ bị quá bộ nhớ. Việc ta có thể làm là tối ưu không gian của mảng DP. Thay vì dùng 2 chiều, ta giảm xuống 1 chiều. Vậy thay vì $DP[i][j]$, ta còn lại $DP[i]$.

51.4 Khởi gán

$DP[0][0] = 1$

51.5 Đáp án

Đáp án của ta với mỗi vị tri i là tổng $DP[i][j] \quad \forall j$. Nhưng vì việc tối ưu bộ nhớ nên ta sẽ lưu lại bằng mảng $ans[i]$ với mọi j duyệt qua.

51.6 Code mẫu

```
#include<bits/stdc++.h>
using namespace std;
const int MAXN = 2e5 + 5;
const int MOD = 998244353;

int n , k;
int dp[MAXN] , ans[MAXN];

int32_t main(){
    ios_base::sync_with_stdio(0);cin.tie(0);cout.tie(0);
    cin >> n >> k;

    dp[0] = 1;
    for(int minDist = k , j = 1 ; minDist <= n ; minDist += k + j , j++){
        vector<int> pref(k + j + 1 , 0);
        for(int i = 0 ; i <= n ; i++){
            int tmp = dp[i];
            dp[i] = pref[i % (k + j - 1)];
            pref[i % (k + j - 1)] = (pref[i % (k + j - 1)] + tmp) % MOD;
            ans[i] = (ans[i] + dp[i]) % MOD;
        }
    }

    for(int i = 1 ; i <= n ; i++)
        cout << ans[i] << " ";
    return 0;
}
```

Đọc code đầy đủ hơn ở *đây*

52 Bài ZZB - Trung bình

52.1 Nhận xét

Đây là 1 bài toán DP cơ bản.

52.2 Định nghĩa DP

Gọi $DP[i][j]$ là độ dài lớn nhất của dãy con là cấp số cộng có công sai là j và có phần tử cuối cùng có giá trị là i.

52.3 Suy ra công thức DP

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

+) Thêm số thứ i và dãy có cấp số cộng là j nên phần tử cuối cùng phải là $i-j$.
$\Rightarrow DP[i][j] = DP[k][j] + 1 \ \ \forall \ a[k] = i - j, \ k < i.$

Với công thức như trên, ta có độ phức tạp là $O(n^2 * 100)$ để giải hết bài toán. Nhưng, với nhận xét ta chỉ cần lấy k lớn nhất bằng $i-j$ vì nếu so sánh $DP[k_1][j]$ và $DP[k_2][j]$ có $k_1 < k_2$ thì ta chứng minh được rằng $DP[k_1][j] \le DP[k_2][j]$. Tới đây ta chỉ cần lấy k lớn nhất thỏa mãn $a[k] = i - j$. Vậy công thức QHĐ của ta là:

$$DP[i][j] = DP[k][j] + 1 \text{ với } k \text{ lớn nhất thỏa } a[k] = i - j.$$

Tuy nhiên, ta thấy với việc $a[i] \le 10^9$ và $D \le 100$ thì việc khai báo mảng DP với bộ nhớ là $10^9 * 100$ là không cần thiết vì ta chỉ cần n giá trị tồn tại trong mảng a. Tới đây, ta sẽ sử dụng CTDL **map** để tìm giá trị $i - j$.

52.4 Khởi gán

$DP[i[j] = 1 \ \ \forall \ i, j$. Ta khởi gán với ý nghĩa $a[i]$ là phần tử đầu tiên với công sai là j. Vì là phần tử đầu tiên nên ta chưa biết công sai của dãy là thế nào nên ta khởi gán với tất cả mọi công sai ≤ 100.

52.5 Đáp án

Đáp án của ta là $max(DP[a[i]][j]) \ \ \forall i, j$

52.6 Code mẫu

```
#include<bits/stdc++.h>
using namespace std;
const int MAXN = 1e5 + 5;
const int MAXD = 105;

int n;
int a[MAXN] , dp[MAXN][105];
map<int , int> ind;

int32_t main(){
    ios_base::sync_with_stdio(0);cin.tie(0);cout.tie(0);
    int n;
    cin >> n;
    for(int i = 1 ; i <= n ; i++)
        cin >> a[i];
```

```
    int res = 1;

    for(int i = 1 ; i <= n ; i++){
        for(int j = 1 ; j <= 100 ; j++){
            dp[i][j] = 1;
            if(ind.find(a[i] - j) == ind.end())
                continue;
            dp[i][j] = dp[ind[a[i] - j]][j] + 1;
            res = max(res , dp[i][j]);
        }
        ind[a[i]] = i;
    }

    cout << res << endl;
}
```

Đọc code đầy đủ hơn ở *đây*

53 Bài ZZC - Đếm dãy con

53.1 Nhận xét

Đây là 1 bài toán DP.

53.2 Định nghĩa DP

Gọi $DP[i][j]$ là số cách chọn dãy con có tổng là j xét tới phần tử thứ i.

53.3 Suy ra công thức DP

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

- Thêm $a[i]$ vào dãy, gọi j là tổng sau khi thêm $a[i]$ thì ta có $j - a[i]$ là tổng sau khi thêm phần tử thứ i vào dãy.

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

- Bỏ qua $a[i]$.

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

Vậy ta có **công thức QHĐ** là:

$$DP[i][j] = DP[i-1][j] + DP[i-1][j-a[i]]$$

53.4 Khởi gán

$DP[i][a[i]] = 1 \ \forall \ i$

53.5 Đáp án

Đến đây, ta suy nghĩ:

"Làm sao để trả lời các truy vấn?".

Đầu tiên, ta có số lượng cách để chọn dãy con có tổng bằng x là $DP[n][x]$. Vậy để đếm số dãy con có tổng x ($L \le x \le R$), ta lấy tổng $DP[n][L]$ + $DP[n][L+1]$ + ... + $DP[n][R]$.
Tới đây, ta có thể sử dụng **PrefixSum** để tính $Pref[i]$ là tổng $DP[n][1]$ $+DP[n][2]$ + ... + $DP[n][i]$. Vậy với mỗi truy vấn dạng lấy số cách chọn dãy con có tổng trong đoạn $[L;R]$, ta trả lời:

$$Pref[R] - Pref[L-1]$$

53.6 Code mẫu

```
#include<bits/stdc++.h>
using namespace std;
#define int long long
const int MAXN = 105;
const int MOD = 1e9 + 7;

int n;
int pref[MAXN];
int a[MAXN];
int dp[MAXN][1005];

int32_t main(){
    ios_base::sync_with_stdio(0);cin.tie(0);cout.tie(0);
    int n;
    cin >> n;

    for(int i = 1 ; i <= n ; i++) cin >> a[i];

    dp[0][0] = 1;

    for(int i = 1 ; i <= n ; i++){
        for(int j = 1000 ; j >= 0 ; j--){
            dp[i][j] = dp[i - 1][j];
            if(j - a[i] >= 0)
                dp[i][j] = (dp[i][j] + dp[i - 1][j - a[i]]) % MOD;
        }
    }

    for(int i = 1 ; i <= 1000 ; i++)
        pref[i] = (pref[i - 1] + dp[n][i]) % MOD;

    int q;
    cin >> q;
```

```
    while(q--){
        int l , r;
        cin >> l >> r;
        cout << (pref[r] - pref[l - 1] + MOD * MOD) % MOD << endl;
    }
    return 0;
}
```

Đọc code đầy đủ hơn ở *đây*

54 Bài ZZD. Tổng tuyệt đối lớn nhất

54.1 Nhận xét

Đây là 1 bài toán DP cơ bản.
Ta có nhận xét như sau, vì việc phải xét $|a[i]-a[i-1]|$ nên việc chọn $a[i]$ và $a[i-1]$ là giá trị nhỏ nhất hoặc giá trị lớn nhất có thể thì sẽ tốt nhất.

54.2 Định nghĩa DP

Với nhận xét trên, ta gọi $DP[i][0 \to 1]$ là giá trị hoàn hảo cao nhất có được với i phần tử đầu và có phần tử thứ i là 1 hoặc là B_i - giá trị lớn nhất có thể của phần tử thứ i.

54.3 Suy ra công thức DP

Từ định nghĩa trên, ta có công thức QHĐ như sau:

+) Chọn $a[i] = b[i]$ - tương ứng với $DP[i][1]$. Ở đây ta lấy max từ cả 2 trạng thái ở trước.

$\Rightarrow DP[i][1] = max(DP[i-1][0] + |b[i]-1|,\ DP[i-1][1] + |b[i]-b[i-1]|)$

+) Chọn $a[i] = 1$ tương ứng với $DP[i][0]$. Tới đây, ta thừa hưởng từ 2 trạng thái ở trước.

$\Rightarrow DP[i][0] = max(DP[i-1][0] + |1-1|,\ DP[i-1][1] + |1-b[i-1]|)$

Vậy ta có **công thức QHĐ** như sau:

$$DP[i][1] = max(DP[i-1][0] + |b[i]-1|,\ DP[i-1][1] + |b[i]-b[i-1]|)$$

$$DP[i][0] = max(DP[i-1][0] + |1-1|,\ DP[i-1][1] + |1-b[i-1]|)$$

54.4 Khởi gán

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

54.5 Đáp án

Đáp án của ta là $max(DP[n][0], DP[n][1])$.

54.6 Code mẫu

```
#include<bits/stdc++.h>
using namespace std;
#define int long long
const int MAXN = 1e6 + 5;
int n;
int b[MAXN];
int dp[MAXN][2];

int32_t main(){
    ios_base::sync_with_stdio(0);cin.tie(0);cout.tie(0);
    cin >> n;
    for(int i = 1 ; i <= n ; i++)
        cin >> b[i];

    dp[1][0] = dp[1][1] = 0;
    for(int i = 2 ; i <= n ; i++){
        dp[i][1] = max(dp[i - 1][0] + abs(b[i] - 1) , dp[i - 1][1] + abs(b[i] - b[i - 1]))
        dp[i][0] = max(dp[i - 1][1] + b[i - 1] - 1 , dp[i - 1][0] + 1 - 1);
    }

    cout << max(dp[n][0] , dp[n][1]) << endl;
    return 0;
}
```

Đọc code đầy đủ hơn ở *đây*

55 Bài ZZF - Lưu niệm

55.1 Nhận xét

Đây là bài toán DP Knapsack cơ bản.

55.2 Định nghĩa DP

Gọi $DP[i][j]$ là tổng giá trị thẩm mỹ lớn nhất lấy được khi xét qua i loại ảnh với j MB dung lượng đã lấy.

55.3 Suy ra công thức DP

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

+) Thêm vào 1 bức ảnh loại i, giả sử j là dung lượng sau khi thêm nên ta có $j - a[i]$ là dung lượng trước khi thêm ảnh.

$\Rightarrow DP[i][j] = DP[i][j - a[i]] + b[i]$

+) Không thêm vào bết kì ảnh loại i nào.

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

Vậy ta có **công thức QHĐ** là:

$$DP[i][j] = max(DP[i-1][j], DP[i][j - a[i]] + b[i])$$

55.4 Khởi gán

$DP[0][0] = 0$

55.5 Đáp án

Đáp án của ta là $DP[n][K * 1024]$ với ý nghĩa là tổng độ thẩm mỹ lớn nhất lấy được sau khi xét qua n loại ảnh.

55.6 Code mẫu

```
#include<bits/stdc++.h>
using namespace std;
#define int long long
const int MAXN = 1e3 + 5;
const int MAXV = 4096 + 5;

int n , k;
int dp[MAXN][MAXV];
pair<int , int> a[MAXN];

int32_t main(){
    ios_base::sync_with_stdio(0);cin.tie(0);cout.tie(0);
    cin >> n >> k;
    k *= 1024;
    for(int i = 1 ; i <= n ; i++)
        cin >> a[i].first >> a[i].second;

    int res = 0;

    for(int i = 1 ; i <= n ; i++){
        for(int j = 0 ; j <= k ; j++)
            if(j - a[i].first >= 0){
```

```
                dp[i][j] = max(dp[i - 1][j] , dp[i][j - a[i].first] + a[i].second);
            }else dp[i][j] = dp[i - 1][j];
    }

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

Đọc code đầy đủ hơn ở *đây*

56 Bài ZZG - Phần thưởng

56.1 Nhận xét

Đây là 1 bài VOI.

56.2 Định nghĩa DP

Gọi $DP[i][j][t]$ là tổng tiền thưởng lớn nhất lấy được bằng các lấy các lá bài trong đoạn $[i;j]$ và đã lấy được t lá bài.

56.3 Suy ra công thức DP

Từ định nghĩa trên, ta có các cách chuyển trạng thái như sau:

+) Chọn 2 thẻ đầu.

$\Rightarrow DP[i][j][t] = DP[i+2][j][t-1] + abs(a[i] - a[i+1])$

+) Chọn 2 thẻ cuối hàng.

$\Rightarrow DP[i][j][t] = DP[i][j-2][t-1] + abs(a[t-1] - a[t])$

+) Chọn 1 thẻ cuối hàng và 1 thẻ đầu hàng.

$\Rightarrow DP[i][j][t] = DP[i+1][j-1][t-1] + abs(a[i] - a[j])$

+) Loại 1 thẻ đầu hàng ra khỏi hàng.

$\Rightarrow DP[i][j][t] = DP[i+1][j][t]$

+) Loại 1 thẻ cuối hàng ra khỏi hàng.

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

Như vậy ta có **công thức QHĐ** như sau:

$$DP[i][j][t] = max(DP[i][j][t], DP[i+2][j][t-1] + abs(a[i] - a[i+1]))$$

$$DP[i][j][t] = max(DP[i][j][t], DP[i][j-2][t-1] + abs(a[t-1] - a[t]))$$

$$DP[i][j][t] = max(DP[i][j][t], DP[i+1][j-1][t-1] + abs(a[i] - a[j]))$$

$$DP[i][j][t] = max(DP[i][j][t], DP[i+1][j][t], DP[i][j-1][t])$$

56.4 Khởi gán

$DP[i][i+1][1] = abs(a[i] - a[i+1]) \quad \forall i < n$

56.5 Đáp án

Đáp án của ta là $DP[1][n][k]$ với ý nghĩa là tổng tiền thưởng lớn nhất lấy được bằng các lấy các lá bài trong đoạn $[1;n]$ và lấy được k lá bài.

56.6 Code mẫu

```
#include<bits/stdc++.h>
using namespace std;
#define int long long
const int MAXN = 300 + 5;
int n , k;
int a[MAXN];
int dp[MAXN][MAXN][MAXN];

int32_t 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];

    for(int t = 1 ; t <= k ; t++){
        for(int len = 1 ; len <= n ; len++){
            for(int l = 1 ; l + len - 1 <= n ; l++){
                int r = l + len - 1;
                if(len < 2 * t) continue;

                if(len == 1){
                    dp[l][r][t] = 0;
                }else if(len == 2){
                    dp[l][r][t] = abs(a[l] - a[r]);
                }else{
                    dp[l][r][t] = max(dp[l][r][t] , dp[l + 2][r][t - 1] + abs(a[l] - a[l +
                    dp[l][r][t] = max(dp[l][r][t] , dp[l][r - 2][t - 1] + abs(a[r] - a[r -
                    dp[l][r][t] = max(dp[l][r][t] , dp[l + 1][r - 1][t - 1] + abs(a[l] - a
                    dp[l][r][t] = max(dp[l][r][t] , dp[l + 1][r][t]);
                    dp[l][r][t] = max(dp[l][r][t] , dp[l][r - 1][t]);
                }
            }
        }
    }

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

Đọc code đầy đủ hơn ở *đây*

57 Bài ZZH. Dãy con chung không liền kề dài nhất

57.1 Nhận xét

Đây là 1 bài DP cơ bản.

57.2 Định nghĩa DP

Gọi $DP[i][j]$ là độ dài của dãy con chung không liền kề dài nhất xét qua i phần tử đầu của dãy A và j phần tử đầu cả dãy B.

57.3 Suy ra công thức DP

Từ định nghĩa trên, ta có các cách chuyển trạng thái như sau:

+) Bỏ qua phần tử thứ i của dãy A.

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

+) Bỏ qua phần tử thứ j của dãy B.

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

+) Chọn cặp phần tử $a[i]$ và $b[j]$ nếu $a[i] = b[j]$. Vì là dãy con chung không liền kề nên để tránh việc chọn 2 phần tử kề nhau, ta cập nhật từ $DP[i-2][j-2]$. Mang ý nghĩa chỉ xét tới phần tử tới $i-2$ của A và $j-2$ của B nên sẽ không có trường hợp kề nhau với do i và $i-1$ hoặc j và $j-1$.

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

Vậy ta có **công thức QHĐ** như sau:

$$DP[i][j] = max(DP[i-1][j], DP[i][j-1])$$

$$DP[i][j] = max(DP[i][j], DP[i-2][j-2] + 1) \ \ \forall \ a[i] = b[j]$$

57.4 Khởi gán

$DP[0][0] = 0$

57.5 Đáp án

Đáp án của ta là $DP[m][n]$.

57.6 Code mẫu

Lưu ý: Để tránh các trường hợp $DP[i-2][j-2]$ ra ngoài vùng dữ liệu, ta điều chỉnh mảng DP bắt đầu từ ô $(2,2)$ thay vì ô $(1,1)$.

```
#include<bits/stdc++.h>
using namespace std;
#define int long long
const int MAXN = 1e6 + 5;
int n , m;
int a[MAXN] , b[MAXN];
int dp[5005][5005];

int32_t main(){
    ios_base::sync_with_stdio(0);cin.tie(0);cout.tie(0);
    cin >> n >> m;
    for(int i = 2 ; i <= n + 1 ; i++)
        cin >> a[i];
    for(int i = 2 ; i <= m + 1 ; i++)
        cin >> b[i];

    for(int i = 2 ; i <= n + 1 ; i++){
        for(int j = 2 ; j <= m + 1 ; j++){
            dp[i][j] = max(dp[i - 1][j] , dp[i][j - 1]);
            if(a[i] == b[j])
                dp[i][j] = max(dp[i - 2][j - 2] + 1 , dp[i][j]);
        }
    }

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

Đọc code đầy đủ hơn ở *đây*

58 Bài ZZI - Cột điện

58.1 Nhận xét

Bài toán này là 1 bài DP yêu cầu kĩ thuật tối ưu.

58.2 Định nghĩa DP

Gọi $DP[i][j]$ là chi phí thấp nhất để xếp i cột điền đầu với cột điện thứ i có độ cao là j.

58.3 Suy ra công thức DP

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

+) Đặt j là độ cao của cột thứ i và nối vào cột $i-1$ có độ cao là k.

$\Rightarrow DP[i][j] = min(DP[i-1][k] + c * |j-k| + (j-h[i])^2)\ \forall h[i] \le j \le 1000,\ k$

Với công thức như trên, dễ dàng ta thấy độ phức tạp của ta hiện tại là $O(n * 1000^2)$. Vì thế ta cần phải tối ưu.

Chúng ta có một tính chất của hàm giá trị tuyệt đối như sau:

$$|a-b| = max(a,b) - min(a,b)$$

Từ tính chất trên, ta biến đổi lại chi phí nối giữa 2 cột điện:

$$c * |h_i - h_{i-1}| \Leftrightarrow c * max(h_i, h_{i-1}) - c * min(h_i, h_{i-1})$$

Từ đây, ta thay đổi công thức QHĐ của ta thành:

+) Nếu $j \ge k$

$\Rightarrow DP[i][j] = min(DP[i][j], DP[i-1][k] - (c*k) + (c*j) + (j-h[i])^2)$

+) Nếu $j < k$

$\Rightarrow DP[i][j] = min(DP[i][j], DP[i-1][k] + (c*k) - (c*j) + (j-h[i])^2)$

Nếu ta để ý, mình vẫn có thể biến đổi công thức như sau:

+) Nếu $j \ge k$

$\Rightarrow DP[i][j] = min(DP[i][j],\ min(DP[i-1][k] - (c*k)) + (c*j) + (j-h[i])^2)$

+) Nếu $j < k$

$\Rightarrow DP[i][j] = min(DP[i][j],\ min(DP[i-1][k] + (c*k)) - (c*j) + (j-h[i])^2)$

Vì vậy, với mỗi j, ta có bài toán tìm:

+) $min(DP[i-1][1], DP[i-1][2], ..., DP[i-1][j])$ với trường hợp $j >= k$.

+) $min(DP[i-1][j+1], DP[i-1][j+2], ..., DP[i-1][1000])$ với trường hợp $j < k$.

Đến đây, ta sử dụng **PrefixMin** và **SuffixMin** để tối ưu như sau:

+) Với **PrefixMin**, ta gọi $Pref[i][j]$ là $min(DP[i][1], DP[i][2], ..., DP[i][j])$.

+) Với **SuffixMin**, ta gọi $Suffix[i][j]$ là $min(DP[i][j], DP[i][j+1], ..., DP[i][1000])$.

Như vậy, với việc dựng mảng PrefixMin và SuffixMin, ta tối ưu chương trình của ta từ $O(n * 1000^2) \rightarrow O(n * 1000)$, với công thức cuối cùng là:

$$DP[i][j] = min(Pref[i-1][j] + c*j, Suffix[i-1][j] - c*j) + (j-h[i])^2$$

58.4 Khởi gán

$DP[0][i] = 0 \ \forall h[1] \leq i \leq 1000$

$DP[i][j] = \infty \ \ \forall \ j < h[i]$

58.5 Đáp án

Đáp án của ta là $max(DP[n][i]) \ \forall \ i \leq 1000$.

58.6 Code mẫu

```
#include<bits/stdc++.h>
using namespace std;
#define int long long
const int MAXN = 1e4 + 5;
const int MAXK = 1e3 + 5;
const int inf = 1e18;
int n , c;
int h[MAXN];
int dp[MAXN][MAXK];
int pref[MAXN][MAXK] , suffix[MAXN][MAXK];

int32_t main(){
    ios_base::sync_with_stdio(0);cin.tie(0);cout.tie(0);
    cin >> n >> c;
    for(int i = 1; i <= n ; i++) cin >> h[i];

    for(int i = 1 ; i <= n ; i++){
        for(int j = 0 ; j <= 1001 ; j++)
            dp[i][j] = pref[i][j] = suffix[i][j] = inf;
    }

    for(int i = 1 ; i <= n ; i++){
        for(int j = h[i] ; j <= 1000 ; j++){
            int add = (j - h[i]) * (j - h[i]);
            if(i > 1)dp[i][j] = min(pref[i - 1][j] + c * j , suffix[i - 1][j] - c * j) + a
            else dp[i][j] = add;
        }
        for(int j = 1 ; j <= 1000 ; j++)
            pref[i][j] = min(pref[i][j - 1] , dp[i][j] - c * j);
        for(int j = 1000 ; j > 0 ; j--)
            suffix[i][j] = min(suffix[i][j + 1] , dp[i][j] + c * j);
    }
    int res = inf;
    for(int i = 1 ; i <= 1000 ; i++)
        res = min(res , dp[n][i]);
    cout << res << endl;
```

```
    return 0;
}
```

Đọc code đầy đủ hơn ở *đây*

59 Bài ZZZ. Trồng cây

59.1 Nhận xét

Đây là 1 bài DP Knapsack khá khó, yêu cầu tối ưu.

59.2 Định nghĩa DP

Gọi $DP[i][j]$ là tổng hiệu quả kinh tế lớn nhất có thể tạo ra xét tới loại cây thú i và tổng số đồng đã dùng hiện tại là j.

59.3 Suy ra công thức DP

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

+) Ta thêm vào k cây loại i và có j là số lượng đồng đã dùng sau khi thêm.

$$\Rightarrow DP[i][j] = max(DP[i][j], DP[i-1][j-k*c[i]] + v[i]*k - \frac{k*(k-1)}{2}*w[i].$$

Đến đây, độ phức tạp tối đa của ta là $O(n*max_c i*max_T) = 10^5*10^3*10^3 \Rightarrow$ TLE.
Vì vậy, ta có nhận xét như sau, với mỗi loại cây, thay vì duyệt chọn k cây mỗi lần, sinh ra m cây với m là số cây nhiều nhất đặt được cho tới khi giá trị kinh tế không dương. Với mỗi cây có giá trị như sau:

+) Cây thứ 1, có giá trị là $v[i]$ và giá tiền là $c[i]$.

+) Cây thứ 2, có giá trị là $v[i]-w[i]$ và giá tiền là $c[i]$

+) Cây thứ 3, có giá trị là $v[i]-2*w[i]$ và giá tiền là $c[i]$

$\vdots$

+) Cây thứ m, có giá trị là $v[i]-m*w[i]$ và giá tiền là $c[i]$

Tại sao ta lại sinh ra các cây như vậy?

Đơn giản, nếu nhìn nhận một cách tham lam, với cùng giá tiền nhưng giá trị kinh tế chênh lệch nhau, ta sẽ luôn ưu tiên lấy theo thứ tự từ 1. Vì vậy đến đây, bài toán của ta trở thành Knapsack kinh điển.

Tuy nhiên, nếu chỉ cài đặt không, ta vẫn TLE do số lượng vật có thể sinh ra là rất nhiều, với trường hợp tệ nhất là 10^8 vật riêng biệt.

Đến đây, ta lại có nhận xét như sau. Với mỗi vật có giá tiền c, ta chỉ cần lưu không quá $\frac{1000}{c}$ vật có giá trị lớn nhất với giá tiền c như vậy. Đơn giản vì nếu ta lưu trữ nhiều hơn, các vật có thứ tự theo tăng dần > $\frac{1000}{c}$ sẽ không bao giờ được mua và ta đã lấy những vật có giá trị lớn hơn. Vì tổng số tiền của $\frac{1000}{c}$ vật đầu tiên sẽ lớn hơn hoặc bằng 1000. $\frac{1000}{c} * c = 1000$.

59.4 Khởi gán

59.5 Đáp án

59.6 Code mẫu

```
#include<bits/stdc++.h>
using namespace std;
#define int long long
const int MAXN = 1e5 + 5;
const int MAXV = 1e3;
const int inf = 1e18;

int n , m , maxT = -inf;
int c[MAXN] , v[MAXN] , w[MAXN];
int dp[MAXV + 5];
priority_queue<int , vector<int> , greater<int>> item[MAXV + 5];
vector<pair<int , int>> q;
vector<int> T;

int32_t main(){
    ios_base::sync_with_stdio(0);cin.tie(0);cout.tie(0);
    cin >> n >> m;
    for(int i = 1 ; i <= n ; i++)
        cin >> c[i] >> v[i] >> w[i];
    for(int i = 1 ; i <= m ; i++){
        int x; cin >> x; T.push_back(x);
        maxT = max(maxT , x);
    }

    for(int i = 1 ; i <= n ; i++){
        int cur = v[i];
        while(cur > 0){
            item[c[i]].push(cur);
            cur -= w[i];
            if(item[c[i]].size() > maxT / c[i])
                item[c[i]].pop();
        }
    }

    q.push_back({0 , 0});
    for(int i = 1 ; i <= MAXV ; i++){
```

```
        while(!item[i].empty()){
            q.push_back({i , item[i].top()});
            item[i].pop();
        }
    }

    for(int i = 1 ; i < q.size() ; i++){
        for(int j = MAXV ; j >= q[i].first ; j--)
            dp[j] = max(dp[j] , dp[j - q[i].first] + q[i].second);
    }

    for(auto x : T)
        cout << dp[x] << " ";
    return 0;
}
```

Đọc code đầy đủ hơn ở *đây*