ankhangluonvuituoi@gmail.com | 0967 670 770 | https://github.com/GrootTheDeveloper

QUY HOẠCH ĐỘNG

Tài liệu ôn tập Competitive Programming

Đặng Phúc An Khang

Sinh viên ngành CNTT (AI & DS) — Trường Đại học Quản lý & Công nghệ TP.HCM (UMT)

Ngày 27 tháng 8 năm 2025

Mục lục

1 Giới thiệu **2**
- 1.1 Các nguồn tài nguyên . . . 2
- 1.2 Tài khoản trên các Online Judge . . . 2
- 1.3 Một vài lưu ý . . . 2

2 Một số bài toán Quy hoạch động cổ điển **3**
- 2.1 Giới thiệu . . . 3
- 2.2 Cơ sở lý thuyết . . . 3
- 2.3 Ví dụ minh hoạ . . . 4
  - 2.3.1 Fibonacci . . . 4
- 2.4 Bài tập . . . 4

3 Một số bài toán Quy hoạch động không cổ điển **11**

4 Quy hoạch động nâng cao **18**

5 Kỹ thuật đổi biến số trong Quy hoạch động **19**

6 Bitmask + Dynamic Programming **20**

7 SOS **21**

8 Digit Dynamic Programming **22**

9 Matrix Multiplication Dynamic Programming **23**

10 Optimize DP by Divide and Conquer **24**

11 Optimize DP by Knuth - Yao **25**

CHƯƠNG 1

GIỚI THIỆU

Contents

1.1 Các nguồn tài nguyên 2
1.2 Tài khoản trên các Online Judge 2
1.3 Một vài lưu ý 2

Bài viết này được biên soạn với mục tiêu giúp tác giả hệ thống hoá và vận dụng các kiến thức thuộc chuyên đề *Quy hoạch động (Dynamic Programming)*, từ đó áp dụng hiệu quả trong *Competitive Programming* (Lập trình thi đấu).

1.1 Các nguồn tài nguyên

- C/C++: `https://github.com/GrootTheDeveloper/OLP-ICPC/tree/master/2025/C%2B%2B`
- [Kho23]. *CP10. Competitive Programming* `https://drive.google.com/drive/folders/1MTEVHT-7nBnMJ7C9LgyAR_pEVSE3FlKz?fbclid=IwAR3TovIj2rKCRe1a4oZxW-LQCoEoVkipVAvCzwrr0nJ6GzcAd47P6LO1Rwc`
- [CP-]. *Algorithms for Competitive Programming* `https://cp-algorithms.com`
- [VNO]. *Thư viện VNOI* `https://wiki.vnoi.info`

1.2 Tài khoản trên các Online Judge

- Codeforces: `https://codeforces.com/profile/vuivethoima`
- VNOI: `oj.vnoi.info/user/Groot`
- IUHCoder: `oj.iuhcoder.com/user/ankhang2111`
- MarisaOJ: `https://marisaoj.com/user/grootsiuvip/submissions`
- CSES: `https://cses.fi/user/212174`
- UMTOJ: `sot.umtoj.edu.vn/user/grootsiuvip`
- SPOJ: `www.spoj.com/users/grootsiuvip/`
- POJ: `http://poj.org/userstatus?user_id=vuivethoima`
- AtCoder: `https://atcoder.jp/users/grootsiuvip`
- OnlineJudge.org: `vuivethoima`

1.3 Một vài lưu ý

Chuyên đề này được viết bởi hai "tác giả":

- **vuivethoima** – tác giả chính, chịu trách nhiệm biên soạn nội dung.
- **Groot** – một thằng chuyên chọc ngoáy, đặt những câu hỏi nghe thì rất ngu ngơ nhưng lại gợi mở những góc khuất của bài toán mà thường ít ai để ý (chắc vậy?).

Nói cho sang thì là "cộng tác", nhưng thực chất đây là quá trình DPAK tự viết, rồi tự hỏi, rồi tự tranh luận. Hai "nhân vật" trong đầu thay phiên nhau đóng vai *tác giả* và *độc giả khó tính*. Và thế là hình thành nên chuyên đề này.

CHƯƠNG 2

MỘT SỐ BÀI TOÁN QUY HOẠCH ĐỘNG CỔ ĐIỂN

Contents

2.1 Giới thiệu **3**
2.2 Cơ sở lý thuyết **3**
2.3 Ví dụ minh hoạ **4**
2.3.1 Fibonacci 4
2.4 Bài tập **4**

2.1 Giới thiệu

Quy hoạch động (Dynamic Programming, DP) là kỹ thuật thiết kế thuật toán nhằm giải các bài toán tối ưu hoặc đếm bằng cách:

- Chia bài toán thành các *bài toán con* có cấu trúc tương tự.
- Tận dụng tính *chồng lặp* của các bài toán con bằng cách *ghi nhớ* (memoization) hoặc *lập bảng* (tabulation).
- Dựa vào *tính tối ưu con* (optimal substructure) để xây dựng *công thức truy hồi* (recurrence).

Lợi ích chính: giảm độ phức tạp từ hàm mũ/đệ quy thuần xuống đa thức/ giả đa thức bằng cách tránh tính lặp.

2.2 Cơ sở lý thuyết

Khi nào dùng DP?

1. **Bài toán con chồng lặp:** nhiều lời gọi lặp lại cùng trạng thái.
2. **Tối ưu con:** nghiệm tối ưu toàn cục được ghép từ nghiệm tối ưu của các phần.

Hai hướng tiếp cận

Top-down (Memoization): Viết đệ quy theo công thức truy hồi, lưu kết quả trạng thái đã tính để dùng lại.
Bottom-up (Tabulation): Xác định *thứ tự* tính các trạng thái từ nhỏ đến lớn, điền vào bảng.

Quy trình thiết kế DP (tư duy theo bước)

Bước 1. Xác định trạng thái $dp[\cdot]$: mỗi trạng thái mã hoá "bài toán con" gì?

Bước 2. Công thức chuyển (recurrence): từ trạng thái nhỏ hơn suy ra trạng thái hiện tại.

Bước 3. Điều kiện biên (base cases): giá trị khởi đầu.

Bước 4. Thứ tự tính / hướng duyệt: để mọi phụ thuộc đã sẵn sàng khi cần.

Bước 5. Kết quả cần lấy ở đâu (ô nào trong bảng)?

Bước 6. Phân tích độ phức tạp thời gian/bộ nhớ; cân nhắc tối ưu hoá không gian nếu được.

2.3 Ví dụ minh hoạ

2.3.1 Fibonacci

Bài toán: Tính $F(n)$ với $F(0) = 0$, $F(1) = 1$, $F(n) = F(n-1) + F(n-2)$.
Thiết kế:

- Trạng thái: $dp[i] = F(i)$.
- Biên: $dp[0] = 0$, $dp[1] = 1$.
- Chuyển: $dp[i] = dp[i-1] + dp[i-2]$.
- Thứ tự: $i = 2 \to n$.

Pseudocode:

```
function Fibonacci(n):
    if n <= 1: return n
    a <- 0; b <- 1          # a = F(0), b = F(1)
    for i from 2 to n:
        c <- a + b          # c = F(i)
        a <- b
        b <- c
    return b
```

2.4 Bài tập

Bài tập 1. A - Frog 1 **link:** https://atcoder.jp/contests/dp/tasks/dp_a

Cho N tảng đá được đánh số từ 1 đến N, mỗi đá có độ cao h_i. Ếch ban đầu đứng ở đá số 1 và muốn đến đá số N. Từ đá i, ếch có thể nhảy đến đá $i+1$ hoặc đá $i+2$. Chi phí khi ếch nhảy từ đá i đến đá j là

$$|h_i - h_j|.$$

Hãy tính chi phí nhỏ nhất để ếch đi từ đá 1 đến đá N.
Giới hạn

- Tất cả số trong input đều là số nguyên.
- $2 \le N \le 10^5$
- $1 \le h_i \le 10^4$

Ví dụ

Sample Input	Sample Output
4 10 30 40 20	30

Phân tích bài toán
Gọi $f[i]$ là chi phí nhỏ nhất để ếch đi từ đá 1 đến đá i.
Trường hợp cơ sở: $f[1] = 0$ (chi phí để ếch đi từ đá 1 đến đá 1 là 0, vì nó đứng tại chỗ).
Khi ếch đứng tại đá 2, trước đó nó chỉ có 1 cách nhảy là từ đá 1 sang. Vậy: $f[2] = f[1] + |h[2] - h[1]|$
Khi ếch đứng tại đá 3, có 2 cách có thể nhảy trước đó:

- Nhảy từ đá 1 sang đá 3, hoặc
- Nhảy từ đá 2 sang đá 3.

Đương nhiên ta sẽ chọn cách tốn ít chi phí nhất. Vậy: $f[3] = \min\left(f[2] + |h[3] - h[2]|,\ f[1] + |h[3] - h[1]|\right)$
Tương tự, khi ếch đứng tại đá 4, có 2 cách nhảy có thể nhảy:

- Nhảy từ đá 2 sang đá 4, hoặc
- Nhảy từ đá 3 sang đá 4.

Vậy: $f[4] = \min\left(f[2] + |h[4] - h[2]|,\ f[3] + |h[4] - h[3]|\right)$
Từ những tính toán trên, ta rút ra được công thức truy hồi tổng quát:

$$\text{Với } i \ge 3: \quad f[i] = \min\left(f[i-2] + |h[i] - h[i-2]|,\ f[i-1] + |h[i] - h[i-1]|\right)$$

Kết quả bài toán: $f[n]$

[Groot]: Quá hay, người tạch giải 2 OLP'24 viết có khác. Mà đột nhiên tao nãy ra suy nghĩ là cái công thức này dựa trên việc ếch chỉ được nhảy 1 hoặc 2 bước. Nếu lỡ tao thay đổi luật cho nhảy từ i đến $i+3$ thì sao?

[vuivethoima]: Tập trung vào bài tập đi thằng chó. À thì nếu thay đổi luật thì tao chỉ cần mở rộng công thức thành:

$$f[i] = \min\Big(f[i-1] + |h[i] - h[i-1]|,\ f[i-2] + |h[i] - h[i-2]|,\ f[i-3] + |h[i] - h[i-3]|\Big).$$

Ý tưởng không đổi, chỉ khác ở tập trạng thái chuyển tiếp thôi.

[Groot]: À tao hiểu rồi, vậy là quan trọng không phải học vẹt cái công thức mà phải hiểu cách nó sinh ra, để khi thay đổi đề thì vẫn ứng biến được.

[vuivethoima]: Bingo!!! Cái đó mới là tinh thần "học DP": nắm nguyên lý chứ không chỉ chép công thức.

Cài đặt

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

signed main() {
    int n; cin >> n;
    vector<int> h(n + 1), f(n + 1);

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

    f[1] = 0;
    f[2] = abs(h[2] - h[1]);

    for (int i = 3; i <= n; i++) {
        f[i] = min(f[i - 1] + abs(h[i] - h[i - 1]),
                   f[i - 2] + abs(h[i] - h[i - 2]));
    }

    cout << f[n];
    return 0;
}
```

Bài tập 2. Xếp hàng mua vé **link:** https://oj.vnoi.info/problem/nktick

Có N người mua vé dự concert, đánh số từ 1 đến N theo thứ tự đứng trong hàng. Mỗi người cần mua một vé, song người bán vé được phép bán cho mỗi người tối đa 2 vé. Vì vậy, một số người có thể rời hàng và nhờ người đứng trước mình mua hộ vé. Biết t_i là thời gian cần thiết để người i mua xong vé cho mình. Nếu người $i+1$ rời khỏi hàng và nhờ người i mua hộ vé thì thời gian để người i mua vé cho cả hai là r_i.

Yêu cầu: Hãy xác định tổng thời gian phục vụ cho N người là thấp nhất.

Input

- Dòng đầu tiên chứa số N $(1 \le N \le 6 \cdot 10^4)$
- Dòng thứ hai ghi N số nguyên dương $t_1, t_2, \ldots, t_N (1 \le t_i \le 3 \cdot 10^4)$
- Dòng thứ ba ghi $N-1$ số nguyên dương $r_1, r_2, \ldots, r_{N-1} (1 \le r_i \le 3 \cdot 10^4)$

Output In ra tổng thời gian phục vụ nhỏ nhất.

Sample Input	Sample Output
5 2 5 7 8 4 4 9 10 10	18

Phân tích bài toán.

Gọi $f[i]$ là tổng thời gian phục vụ thấp nhất đến người thứ i.

Trường hợp cơ sở: $f[1] = t[1]$

Với $i = 2$: người thứ 2 có thể nhờ người thứ 1 mua vé, hoặc cả hai mua độc lập: $f[2] = \min(r[1],\ t[1] + t[2])$

Với $i = 3$: người thứ 3 có thể nhờ người 2 mua vé hộ, hoặc tự mua: $f[3] = \min(f[1] + r[2],\ f[2] + t[3])$

Với $i = 4$: tương tự $f[4] = \min(f[2] + r[3],\ f[3] + t[4])$

Từ những tính toán trên, ta rút ra được công thức truy hồi tổng quát:

$$\text{Với } i \ge 3: \quad f[i] = \min(f[i-2] + r[i-1],\ f[i-1] + t[i])$$

Kết quả bài toán: $f[N]$.

[Groot]: Ủa, bài Ếch thì tao còn hình dung được: nhảy 1 bước hoặc 2 bước, công thức thấy rõ ràng. Còn bài xếp hàng mua vé này sao mày lại nảy ra được cái công thức DP nghe hợp lý vậy? Tao thấy giống như mày bịa ra mà nó lại đúng ấy?

[vuivethoima]: Cái thằng ngoo này, phân tích quá rõ ràng ở trên mà còn không hiểu. Thôi để tao cho mày xem lại cái luồng suy nghĩ của tao. Tao nghĩ thử: để phục vụ tới người thứ i, thì có 2 tình huống thôi. Một là thằng i tự mua vé $\Rightarrow f[i] = f[i-1] + t[i]$. Hai là nó nhờ thằng $i-1$ mua hộ $\Rightarrow f[i] = f[i-2] + r[i-1]$. Hết phim, không có lựa chọn nào khác nữa.

[Groot]: À, tức là bản chất nó vẫn là "chia trường hợp" hả? Giống bài Ếch mà mỗi lần tới đá mới thì coi xem bước từ đâu tới.

[vuivethoima]: Đúng rồi, y chang. Chỉ khác là ở đây "bước nhảy" không phải 1 hay 2 đá, mà là "tự mua" hoặc "nhờ thằng trước mua hộ". Nhiều khi DP nó vậy đó, nhìn thì ảo ma vcl nhưng bản chất chỉ là liệt kê hết trường hợp hợp lệ rồi chọn cái tối ưu thôi.

Cài đặt

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

signed main() {
    int N; cin >> N;
    vector<int> t(N + 1), r(N);

    for (int i = 1; i <= N; i++) cin >> t[i];
    for (int i = 1; i < N; i++) cin >> r[i];

    vector<int> f(N + 1, oo);

    f[1] = t[1];
    if (N >= 2) f[2] = min(t[1] + t[2], r[1]);

    for (int i = 3; i <= N; i++) {
        f[i] = min(f[i - 1] + t[i],
                   f[i - 2] + r[i - 1]);
    }

    cout << f[N];
    return 0;
}
```

Bài tập 3. Dãy con tăng dài nhất (bản dễ) **link:** https://oj.vnoi.info/problem/liq

Cho một dãy số nguyên gồm N phần tử $A_1, A_2, ..., A_N$.
Biết rằng dãy con tăng đơn điệu là một dãy $A_{i_1}, ..., A_{i_k}$ thỏa mãn $i_1 < i_2 < ... < i_k$ và $A_{i_1} < A_{i_2} < ... < A_{i_k}$.
Hãy cho biết dãy con tăng đơn điệu dài nhất của dãy này có bao nhiêu phần tử. **Input**

- Dòng 1 gồm 1 số nguyên là số N $(1 \le N \le 1000)$.
- Dòng thứ 2 ghi N số nguyên $A_1, A_2, \ldots, A_N (1 \le A_i \le 1000)$.

Output Ghi ra độ dài của dãy con tăng đơn điệu dài nhất.

Sample Input	Sample Output
6 1 2 5 4 6 2	4

Phân tích bài toán

Gọi $f[i]$ là dãy con tăng đơn điệu dài nhất khi xét phần tử thứ i.
Ta thấy được ban đầu tất cả mọi dãy con đều có độ dài là 1 vì chỉ có chính nó, hay $\forall i, f[i] = 1$.
Khi dãy chỉ có 2 phần tử, độ dài tối đa là 2 nếu $a[2] > a[1]$, hay $f[2] = f[1] + 1$ nếu $a[2] > a[1]$. Ngược lại, nếu $a[2] \le a[1]$ thì $f[2]$ vẫn giữ nguyên độ dài là 1.
Khi dãy có 3 phần tử, độ dài tối đa là 3 nếu $a[3] > a[2] > a[1]$, hay $f[3] = f[2] + 1$. Hoặc độ dài tối đa là 2 trong trường hợp:

- $a[3] > a[2]$ và $a[3] \le a[1]$, hay $f[3] = f[2] + 1$ $(f[2] = 1)$
- $a[3] > a[1]$ và $a[3] \le a[2]$, hay $f[3] = f[1] + 1$ $f[1] = 1$

Nếu không có phần tử $a[j]$ nào thỏa mãn $a[i] > a[j]$ thì độ dài lớn nhất tại i sẽ là $f[i] = 1$.
Từ những tính toán trên, ta rút ra được công thức truy hồi tổng quát:

$$f[i] = \max\left(1, \max_{1 \le j < i, a[i] > a[j]} (f[j] + 1)\right)$$

Kết quả bài toán: $\max_{1 \le i \le N} f[i]$

[Groot]: Ủa tự nhiên ở đâu lòi ra cái $\max\limits_{j<i, a[i]>a[j]} (f[j]+1)$ vậy?

[vuivethoima]: Mày nghĩ coi, muốn có dãy tăng kết thúc ở $a[i]$ thì trước đó phải có thằng nào nhỏ hơn $a[i]$ chứ. Nối $a[i]$ vô sau nó thì độ dài tăng thêm 1, thành $f[j]+1$.

[Groot]: Ờ, nghe hợp lý... Nhưng mày nói rõ ràng hơn cái vụ max đi.

[vuivethoima]: Vì đâu chỉ có một thằng nối được. Có khi $a[i]$ lớn hơn nhiều thằng trước đó, mỗi thằng cho ra một độ dài khác nhau. Thì mình phải chọn cái dài nhất trong mớ đó chứ.

[Groot]: À à, hiểu rồi. Tức là "thử nối với tất cả đứa trước đó, lấy cái lời nhất". Nếu không nối được với ai thì tự đứng một mình, $f[i]=1$.

Cài đặt

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

signed main() {
    int N; cin >> N;
    vector<int> A(N + 1), f(N + 1, 1);

    for (int i = 1; i <= N; i++) cin >> A[i];

    int ans = 1;
    for (int i = 2; i <= N; i++) {
        for (int j = 1; j < i; j++) {
            if (A[i] > A[j]) {
                f[i] = max(f[i], f[j] + 1);
            }
        }
        ans = max(ans, f[i]);
    }

    cout << ans;
    return 0;
}
```

Bài tập 4. Longest Common Subsequence **link:** https://oj.vnoi.info/problem/atcoder_dp_f

Bạn được cho hai xâu s và t. Hãy tìm xâu con chung dài nhất của 2 xâu đó.

Lưu ý: Xâu con của xâu x là xâu được tạo bằng cách xóa 0 hoặc một số kí tự thuộc xâu x và nối các ký tự còn lại mà không thay đổi vị trí của chúng.

Input

- Dòng 1 gồm 1 số nguyên là số N $(1 \le N \le 1000)$.
- Dòng thứ 2 ghi N số nguyên $A_1, A_2, \ldots, A_N (1 \le A_i \le 1000)$.

Output Ghi ra xâu con chung dài nhất của 2 xâu s và t.

Sample Input	Sample Output
axyb abyxb	axb

Phân tích bài toán

Gọi $f[i][j]$ là độ dài xâu con chung dài nhất khi xét xâu $s[1..i]$ và xâu $t[1..j]$.

Trường hợp cơ sở: $f[0][j]=0$ và $f[i][0]=0$ vì xâu con chung dài nhất giữa xâu rỗng và một xâu bất kỳ đương nhiên là rỗng (độ dài = 0).

$$f[0][j]=0, \quad f[i][0]=0 \quad \forall i,j$$

Khi xét xâu $s[1..i]$ và $t[1..j]$, ta có 2 trường hợp xảy ra:

- Nếu kí tự cuối cùng trùng nhau ($s[i] == t[j]$), độ dài xâu con dài nhất lúc này sẽ là: $f[i][j] = f[i-1][j-1]+1$, tức là độ dài xâu con chung dài nhất hiện tại sẽ bằng độ xài xâu con liền trước và cộng thêm kí tự trùng nhau hiện tại.
- Ngược lại, nếu kí từ cuối cùng khác nhau $s[i] \ne t[j]$, ta có 2 cách là bỏ bớt 1 kí tự ở s hoặc t hiện tại để lấy độ dài xâu con lớn nhất giữa hai xâu con đó, hay $f[i][j] = \max(f[i-1][j], f[i][j-1])$.

Từ những tính toán trên, ta rút ra được công thức truy hồi tổng quát:

$$f[i][j] = \begin{cases} f[i-1][j-1] + 1 & \text{nếu } s[i] = t[j] \\ \max(f[i-1][j],\ f[i][j-1]) & \text{nếu } s[i] \neq t[j] \end{cases}$$

Kết quả bài toán: $f[|s|][|t|]$
Truy vết:
Sau khi tính xong bảng f, ta có $f[|s|][|t|]$ là độ dài xâu con chung dài nhất. Để dựng lại xâu này, ta bắt đầu từ ô $(|s|, |t|)$ và đi ngược lại:

- Nếu $s[i] = t[j]$: ký tự này thuộc LCS. Ta thêm $s[i]$ vào kết quả và chuyển về ô $(i-1, j-1)$.
- Nếu $s[i] \neq t[j]$:
  - Nếu $f[i][j] = f[i-1][j]$ thì đi lên ô $(i-1, j)$.
  - Ngược lại, đi sang trái ô $(i, j-1)$.

Tiếp tục như vậy cho đến khi $i = 0$ hoặc $j = 0$. Lúc này ta thu được xâu LCS theo thứ tự ngược, vì ta đi từ cuối về đầu. Đảo ngược lại sẽ ra xâu con chung dài nhất cần tìm.

[Groot]: Nếu có nhiều xâu con chung dài nhất thì sao? Ví dụ như cùng độ dài 4 mà ra tới 2–3 xâu khác nhau thì mày in cái nào?

[vuivethoima]: Hỏi đỡ ngu rồi đó, thì cái câu mày thắc mắc là chuyện hay gặp. Thuật toán truy vết bình thường chỉ lần theo một nhánh, nên kết quả sẽ là một xâu LCS bất kỳ. Nếu đề chỉ cần "một đáp án đúng" thì vậy là đủ.

[Groot]: Còn nếu đề bắt in ra hết mọi xâu LCS thì?

[vuivethoima]: Thì rắc rối hơn nhiều. Mày phải duyệt hết các nhánh có cùng độ dài, dùng backtracking hoặc DFS. Số lượng nghiệm có thể rất lớn nên thường đề bài không yêu cầu. Người ta chỉ cần một nghiệm đại diện thôi.

[Groot]: Vậy mày trình bày chi tiết cho tao phần liệt kê này đi.

[vuivethoima]: Muốn biết thì liên hệ **Bruce**.

Cài đặt

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

signed main() {
    string s, t;
    cin >> s >> t;

    s = "␣" + s;
    t = "␣" + t;

    int m = s.size() - 1;
    int n = t.size() - 1;
    vector<vector<int>> f(m + 1, vector<int>(n + 1, 0));

    for (int i = 1; i <= m; i++) {
        for (int j = 1; j <= n; j++) {
            if (s[i] == t[j]) {
                f[i][j] = f[i - 1][j - 1] + 1;
            } else {
                f[i][j] = max(f[i - 1][j], f[i][j - 1]);
            }
        }
    }

    int i = m, j = n;
    string ans;
    while (i > 0 && j > 0) {
        if (s[i] == t[j]) {
            ans.push_back(s[i]);
            --i; --j;
        } else if (f[i - 1][j] >= f[i][j - 1]) {
            --i;
        } else {
            --j;
        }
```

```
    }

    reverse(ans.begin(), ans.end());
    cout << ans << "\n";
    return 0;
}
```

Bài tập 5. Knapsack **link:** https://atcoder.jp/contests/dp/tasks/dp_d

Có N vật phẩm được đánh số $1, 2, ..., N$. Với mỗi i $(1 \leq i \leq N)$, vật phẩm i có khối lượng w_i và giá trị v_i. Taro có một cái túi, anh được chọn vài món trong N vật phẩm và mang về nhà. Sức chứa của cái túi là W, nghĩa là tổng khối lượng vật phẩm được chứa trong túi tối đa W. Hãy tìm tổng giá trị lớn nhất các vật phẩm mà Taro có thể đem về.

Input

- Dòng đầu tiên chứa hai số nguyên N, W $(1 \leq N \leq 10^2,\ 1 \leq W \leq 10^5)$.
- N dòng tiếp theo, mỗi dòng chứa hai số nguyên w_i, v_i $(1 \leq w_i \leq W,\ 1 \leq v_i \leq 10^9)$.

Output In ra tổng giá trị lớn nhất mà Taro có thể mang về.

Ví dụ

Sample Input	Sample Output
3 8 3 30 4 50 5 60	90

Phân tích bài toán

Gọi $f[i][j]$ là tổng giá trị lớn nhất khi xét vật phẩm thứ i và sức chứa hiện tại của túi là j.

Trường hợp cơ sở: $f[i][0] = 0$ và $f[0][j] = 0$, vì ta không thể mang theo được vật phẩm nào khi sức chứa của túi là 0, hoặc bất kể sức chứa của túi là bao nhiêu thì khi xét vật phẩm thứ 0, nó không tồn tại nên không thể chứa được.

Khi xét $f[i][j]$ ta có 2 trường hợp xảy ra:

- Nếu vật phẩm i hiện tại (w_i, v_i) chứa được trong túi có sức chứa j, hay $j - w_i \geq 0$, tổng giá trị lớn nhất hiện tại có 2 lựa chọn là không chọn vật phẩm đang xét, hoặc chọn vật phẩm đang xét và cộng giá trị vật phẩm. Hay:

$$f[i][j] = max(f[i-1][j], f[i-1][j-w_i] + v_i)$$

- Ngược lại nếu không chứa được, tổng giá trị lớn nhất hiện tại (xét vật phẩm thứ i) sẽ là tổng giá trị lớn nhất khi xét vật phẩm thứ $i-1$, với cùng sức chứa túi hiện tại. Hay:

$$f[i][j] = f[i-1][j]$$

Từ những tính toán trên, ta rút ra được công thức truy hồi tổng quát:

$$f[i][j] = \begin{cases} f[i-1][j] & \text{nếu } j < w_i \\ \max(f[i-1][j],\ f[i-1][j-w_i] + v_i) & \text{nếu } j \geq w_i \end{cases}$$

Kết quả bài toán: $f[n][W]$ - Sau khi xét toàn bộ vật phẩm với tất cả sức chứa mà túi có thể chứa được.

[Groot]: Ủa, công thức mày ghi là $f[i][j] = \max(f[i-1][j], f[i-1][j-w_i]+v_i)$. Tại sao không phải $f[i][j-w_i]$ mà là $f[i-1][j-w_i]$?

[vuivethoima]: À, cái này quan trọng lắm. Nếu mày dùng $f[i][j-w_i]$ thì hóa ra trong cùng một bước tính, mày cho phép chọn đi chọn lại vật phẩm i nhiều lần. Mà đề Knapsack 0/1 thì mỗi vật phẩm chỉ được lấy tối đa một lần thôi.

Cài đặt

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


signed main() {
    int N, W; cin >> N >> W;
    vector<int> w(N + 1), v(N + 1);

    for (int i = 1; i <= N; i++) {
        cin >> w[i] >> v[i];
    }

    vector<vector<int>> f(N + 1, vector<int>(W + 1, 0));
```

```
15
16      for (int i = 1; i <= N; i++) {
17          for (int j = 0; j <= W; j++) {
18              if (j < w[i]) {
19                  f[i][j] = f[i - 1][j];
20              } else {
21                  f[i][j] = max(f[i - 1][j],
22                                f[i - 1][j - w[i]] + v[i]);
23              }
24          }
25      }
26
27      cout << f[N][W];
28      return 0;
29  }
```

CHƯƠNG 3

MỘT SỐ BÀI TOÁN QUY HOẠCH ĐỘNG KHÔNG CỔ ĐIỂN

Bài tập 6. Vacation **link:** https://oj.vnoi.info/problem/atcoder_dp_c

Kỳ nghỉ hè của Taro bắt đầu vào ngày mai, nên anh ấy đã quyết định lên kế hoạch cho nó ngay bây giờ.
Kỳ nghỉ bao gồm N ngày. Vào ngày thứ i ($1 \leq i \leq N$), Taro sẽ chọn và tham gia một trong các hoạt động sau:

- A: Bơi ở biển – nhận được a_i điểm hạnh phúc.
- B: Bắt bọ trên núi – nhận được b_i điểm hạnh phúc.
- C: Làm bài tập ở nhà – nhận được c_i điểm hạnh phúc.

Vì Taro dễ cảm thấy buồn chán nên anh không thể tham gia cùng một hoạt động trong hai ngày liên tiếp.
Input

- Dòng đầu tiên chứa số nguyên N ($1 \leq N \leq 10^5$).
- N dòng tiếp theo, mỗi dòng gồm ba số nguyên a_i, b_i, c_i ($1 \leq a_i, b_i, c_i \leq 10^4$) – lần lượt là điểm hạnh phúc nếu Taro chọn hoạt động A, B, C trong ngày thứ i.

Output In ra tổng số điểm hạnh phúc tối đa mà Taro có thể đạt được.
Ví dụ

Sample Input	Sample Output
3 10 40 70 20 50 80 30 60 90	210

Phân tích bài toán
Gọi $f[i][j]$ là tổng số điểm hạnh phúc tối đa ngày i có thể đạt được khi tham gia hoạt động j ($0 \leq j \leq 2$) trong ngày đó (0 là hoạt động A, 1 là hoạt động B, 2 là hoạt động C).

Trường hợp cơ sở: $f[1][0] = a_1$, $f[1][1] = b_1$, $f[1][2] = c_1$

Vì không được tham gia cùng 1 hoạt động 2 ngày liên tiếp nhau, nên với ngày thứ i ($i \geq 2$) tổng điểm hạnh phúc có thể đạt được là tổng điểm lớn nhất ngày hôm trước (hoạt động khác hôm nay) và hôm nay. Tức:

$$f[i][j] = \max_{k \neq j} (f[i-1][k]) + \text{điểm hạnh phúc của hoạt động } j \text{ ở ngày } i$$

Hay:

$$f[i][0] = \max(f[i-1][1],\ f[i-1][2]) + a_i$$
$$f[i][1] = \max(f[i-1][0],\ f[i-1][2]) + b_i$$
$$f[i][2] = \max(f[i-1][0],\ f[i-1][1]) + c_i$$

Sau khi tính hết $f[N][0],\ f[N][1],\ f[N][2]$, ta được kết quả bài toán là:

$$\text{Đáp án} = \max(f[N][0],\ f[N][1],\ f[N][2])$$

Cài đặt

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

signed main() {
    int n; cin >> n;
    vector<int> a(n + 1), b(n + 1), c(n + 1);
    vector<vector<int>>f(n + 1, vector<int>(3, 0));
    for (int i = 1; i <= n; i++) {
        cin >> a[i] >> b[i] >> c[i];
    }
    f[1][0] = a[1];
    f[1][1] = b[1];
    f[1][2] = c[1];

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

    int ans = max({f[n][0], f[n][1], f[n][2]});
    cout << ans;
}
```

Bài tập 7. Little Shop of Flowers **link:** https://dmoj.ca/problem/ioi99p1

Bạn muốn sắp xếp cửa sổ trưng bày của cửa hàng hoa sao cho đẹp nhất có thể.
Có F bó hoa, mỗi bó hoa thuộc một loại khác nhau, và có ít nhất V bình hoa được đặt thành một hàng.
Các bình hoa được cố định trên giá và đánh số từ 1 đến V từ trái sang phải. Các bó hoa có thể di chuyển, được đánh số từ 1 đến F. Thứ tự số hiệu bó hoa có ý nghĩa: bạn phải đặt các bó hoa sao cho bó hoa số i nằm ở bên trái bó hoa số j nếu $i < j$.
Mỗi bình hoa có một đặc tính riêng – khi đặt một bó hoa vào một bình hoa thì sẽ có một điểm thẩm mỹ A_{ij} (có thể âm hoặc dương). Bình hoa để trống có điểm 0.
Bạn cần sắp xếp các bó hoa vào các bình hoa (theo đúng thứ tự yêu cầu), sao cho tổng điểm thẩm mỹ là lớn nhất có thể. Nếu có nhiều cách sắp xếp đạt cùng giá trị tối đa, bạn chỉ cần in ra một cách hợp lệ bất kỳ.

Input

- Dòng đầu tiên chứa hai số nguyên F, V ($1 \le F \le 100$, $F \le V \le 100$).
- F dòng tiếp theo, mỗi dòng chứa V số nguyên A_{ij} ($-50 \le A_{ij} \le 50$) – điểm thẩm mỹ khi đặt bó hoa thứ i vào bình hoa thứ j.

Output

- Dòng đầu tiên in tổng điểm thẩm mỹ tối đa.
- Dòng thứ hai in F số nguyên – thứ tự các bình hoa đã chọn cho từng bó hoa (theo thứ tự từ bó hoa 1 đến bó hoa F).

Ví dụ

Sample Input	Sample Output
3 5 7 23 -5 -24 16 5 21 -4 10 23 -21 5 -4 -20 20	53 2 4 5

Phân tích bài toán

Gọi $f[i][j]$ là tổng điểm tối đa khi xét bó hoa thứ i đặt vào bình hoa thứ j.

Với bó hoa thứ i, ta có thể đặt từ chậu j trong khoảng từ i đến $V - F + i$. Xét chậu j, ta có thể chọn bó hoa thứ i cùng với bó hoa thứ $i-1$ trong các chậu k trong khoảng từ $i-1$ đến $V - F + i - 1$.

Trường hợp cơ sở: Xét bó hoa đầu tiên, ta có thể chọn $V - F + 1$ bình hoa đầu tiên, tức là:

$$f[1][j] = A[1][j], \quad \forall j \in [1,\ V - F + 1]$$

Với $\forall i \ge 2$, ta có công thức truy hồi tổng quát:

$$f[i][j] = \max_{k<j} (f[i-1][k]) + A[i][j], \quad \forall j \in [i,\ V - F + i]$$

Kết quả bài toán:

$$\text{Kết quả} = \max\left(f[F][j]\right), \forall j \in [V, F]$$

Truy vết:
Sau khi có kết quả bài toán là tổng điểm tối đa, kí hiệu: answer.

Xét $i = F \to 1$, với i ta tìm $f[i][j] == \text{answer}, \forall j \in [i, V - F + i]$. Khi tìm được $f[i][j]$ thỏa mãn, ta đồng thời tìm được bó hoa thứ i được đặt ở chậu j. Sau đó lấy kết quả answer $-= f[i][j]$ để dịch về tổng điểm tối đa khi xét bó hoa thứ $i - 1$, đồng thời giảm i đi 1 đơn vị. Lặp lại đến khi $i < 1$, ta sẽ tìm được các chậu hoa đặt bó hoa tương ứng thỏa mãn yêu cầu bài toán.

[Groot]: Ê, tao thắc mắc chỗ này. Trong công thức mày chỉ cho j chạy từ i tới $V - F + i$. Ủa sao không để j từ 1 tới V cho tiện? Tự nhiên lại giới hạn nhìn hơi kì.

[vuivethoima]: À, cái này là do ràng buộc thứ tự. Bó hoa thứ i bắt buộc phải nằm bên phải bó hoa thứ $i - 1$, nên chắc chắn bình hoa nó chọn phải có chỉ số ít nhất là i. Còn bên phải thì cũng không thể chọn quá xa, vì phải chừa chỗ cho những bó hoa còn lại. Nên giới hạn trên là $V - F + i$.

[Groot]: À, tức là để đảm bảo cuối cùng F bó hoa đều đặt vừa trong V bình. Nếu cho j rộng ra thì sẽ bị tình huống "dồn không kịp" cho mấy bó sau.

[vuivethoima]: Chuẩn rồi. Giới hạn đoạn $[i, V - F + i]$ là để đảm bảo tính hợp lệ, chứ không thì DP có thể tính ra mấy phương án "không thể xếp" được.

Cài đặt

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

signed main() {
    int F, V; cin >> F >> V;
    vector<vector<int>> a(F + 1, vector<int>(V + 1)), f(F + 1, vector<int>(V + 1, -oo));

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

    for (int j = 1; j <= V - F + 1; j++) {
        f[1][j] = a[1][j];
    }

    for (int i = 2; i <= F; i++) {
        for(int j = i; j <= V - F + i; j++) {
            for (int k = i - 1; k < j; k++) {
                f[i][j] = max(f[i][j], f[i - 1][k]);
            }
            f[i][j] += a[i][j];
        }
    }

    int ans = INT_MIN;
    for (int j = 1; j <= V; j++) {
        ans = max(ans, f[F][j]);
    }
    cout << ans;

    int i = F;
    vector<int> res;

    while (i >= 1) {
        for (int j = i; j <= V - F + i; j++) {
            if (f[i][j] == ans) {
                res.push_back(j);
                ans -= a[i][j];
                i--;
                break;
            }
        }
    }
    cout << endl;
    reverse(res.begin(), res.end());
    for (auto p : res) {
```

```
        cout << p << "␣";
    }
}
```

Bài tập 8. Palindrome 2000 **link:** https://www.spoj.com/problems/IOIPALIN/
Ta được cho một chuỗi $S[1..N]$, cần biến chuỗi thành Palindrome (chuỗi đối xứng) bằng cách chèn thêm ký tự với số lần ít nhất.
Input

- Dòng đầu tiên chứa số nguyên N ($3 \leq N \leq 5000$) — độ dài chuỗi.
- Dòng tiếp theo chứa chuỗi S độ dài N.

Output In ra số lần chèn ít nhất để biến S thành palindrome.
Ví dụ

Sample Input	Sample Output
5 Ab3bd	2

Phân tích bài toán
Gọi $f[i][j]$ là số thao tác ít nhất để biến chuỗi $S[i..j]$ thành chuỗi đối xứng.

Trường hợp cơ sở:

- Với $i == j$, chuỗi có độ dài 1 luôn là chuỗi đối xứng, vậy $f[i][j] = 0$
- Với $i > j$, không có chuỗi nào có chỉ số $i > j$ được, tức là chuỗi rỗng. Vậy số thao tác $f[i][j] = 0$

Truy hồi:

- Nếu $s[i] == s[j]$, 2 đầu chuỗi đã đối xứng, không cần insert gì cả. Vậy chỉ cần xét chuỗi $s[i+1..j-1]$ để tính số insert tối thiểu để biến nó thành đối xứng. Tức là: $f[i][j] = f[i+1][j-1]$
- Nếu $s[i] \neq s[j]$, ta có 2 thao tác:
  - Insert $s[j]$ vào trước $s[i]$, chuỗi hiện tại sẽ là $s[j]s[i..j]s[j]$, sau đó xử lý đoạn $s[i..j]$ sau khi insert $\rightarrow s[i+1..j]$
  - Insert $s[i]$ vào sau $s[j]$, chuỗi hiện tại sẽ là $s[i..j]s[i]$, sau đó xử lý đoạn $s[i..j-1]$.
- Vậy số thao tác trong trường hợp này: $f[i][j] = \min(f[i+1][j], f[i][j-1]) + 1$

Tóm lại, công thức truy hồi tổng quát là:

$$f[i][j] = \begin{cases} 0 & \text{nếu } i \geq j \\ f[i+1][j-1] & \text{nếu } s[i] = s[j] \\ \min\left(f[i+1][j],\ f[i][j-1]\right) + 1 & \text{nếu } s[i] \neq s[j] \end{cases}$$

Kết quả bài toán: $f[1][n]$
Lưu ý với $f[i][j]$, để biết $f[i+1][j-1]$ là gì thì ta cần phải tính $f[i+1][j-1]$ trước, tương tự với $f[i+1][j]$ hay $f[i][j-1]$. Tức là ta cần phải tính lần lượt các chuỗi con có độ dài tăng dần lên, hay xét lần lượt các chuỗi có độ dài length từ 2 đến N và tính $f[i][j]$ với $j - i + 1 = \text{length}$

[Groot]: Ủa, mày nói phải tính $f[i][j]$ theo độ dài tăng dần. Sao vậy? Không lẽ không thể tính kiểu đệ quy từ trên xuống hả?

[vuivethoima]: Thật ra làm đệ quy top–down có memo vẫn được, nhưng dễ bị chậm vì số lời gọi hàm rất lớn. Làm bottom–up theo độ dài tăng dần thì mình chắc chắn khi tính $f[i][j]$ các trạng thái nhỏ hơn như $f[i+1][j-1], f[i+1][j], f[i][j-1]$ đã có sẵn, đỡ phải gọi đi gọi lại.

[Groot]: À ha. Rồi còn cái đoạn $s[i] \neq s[j]$ đó, mày bảo chỉ có 2 cách insert thôi. Biết đâu còn cách khác thì sao?

[vuivethoima]: Thực ra bản chất chỉ có 2. Hoặc tao làm cho $s[i]$ "có đôi" bằng cách chèn thêm một ký tự $s[i]$ bên phải, hoặc tao làm cho $s[j]$ "có đôi" bằng cách chèn thêm một ký tự $s[j]$ bên trái. Ngoài ra thì cũng quy về hai tình huống này thôi, chứ không còn nước đi nào khác.

[Groot]: Hmmm, tao vẫn hơi mù mờ. Tại sao insert $s[j]$ trước $s[i]$ thì lại biến bài toán thành $f[i+1][j]$?

[vuivethoima]: Nghĩ thế này nè: giả sử $s[i] \neq s[j]$, tao chèn thêm $s[j]$ vào trước $s[i]$. Lúc này $s[j]$ mới chèn khớp với $s[j]$ cũ ở cuối, coi như cặp này xong xuôi. Phần còn lại cần xử lý chỉ còn đoạn $s[i..j-1]$, mà bỏ ký tự đầu $s[i]$ đi là ra $s[i+1..j]$. Thế nên nó rút gọn thành $f[i+1][j]$.

Cài đặt

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

signed main() {
    int n; cin >> n;
    string s;
    cin >> s;
    s = "␣" + s + "␣";

    vector<vector<int>>f(n + 2, vector<int>(n + 2, oo));

    for (int i = 1; i <= n; i++) {
        for (int j = 1; j <= n; j++) {
            if (i >= j) {
                f[i][j] = 0;
            }
        }
    }

    for (int len = 2; len <= n; len++) {
        for (int i = 1; i <= n - len + 1; i++) {
            int j = i + len - 1;
            if (s[i] == s[j]) {
                f[i][j] = f[i + 1][j - 1];
            } else {
                f[i][j] = min(f[i + 1][j], f[i][j - 1]) + 1;
            }
        }
    }

    cout << f[1][n];
}
```

Bài tập 9. BLAST **link:** https://oj.vnoi.info/problem/mblast

Cho 2 chuỗi S_1 và S_2 lần lượt gồm n ký tự và m ký tự. Cho một số nguyên dương k, ta có thể mở rộng 2 chuỗi S_1 và S_2 bằng cách chèn một vài dấu "_" và sau khi chèn, độ dài 2 chuỗi phải bằng nhau.
Tìm tổng khoảng cách nhỏ nhất giữa 2 chuỗi, biết rằng:
- Nếu $S_1[i]$ và $S_2[j]$ có ít nhất 1 ký tự là "_" thì khoảng cách giữa chúng là k.
- Ngược lại, khoảng cách là $|S_1[i] - S_2[j]|$.

Input

- Dòng đầu tiên chứa chuỗi S_1 độ dài n ($n \leq 2000$).
- Dòng thứ hai chứa chuỗi S_2 độ dài m ($m \leq 2000$).
- Dòng thứ ba chứa số nguyên k ($1 \leq k \leq 100$).

Output In ra tổng khoảng cách nhỏ nhất giữa 2 chuỗi sau khi chèn.
Ví dụ

Sample Input	Sample Output
cmc snmn 2	10

Phân tích bài toán
Gọi $f[i][j]$ là khoảng cách nhỏ nhất khi xét $S_1[1..i]$ và $S_2[1..j]$.

Trường hợp cơ sở: Khi chuỗi S_1 rỗng, để 2 chuỗi bằng nhau, ta phải insert các ký tự "_" vào S_1 để độ dài 2 chuỗi bằng nhau, khoảng cách giữa nó và các xâu con trong S_2 lần lượt là $f[0][j] = j * k$. Tương tự với S_2 : $f[i][0] = i * k$

Với $S_1[1..i]$ và $S_2[1..j]$, ta có 3 lựa chọn:

- Tính khoảng cách giữa $|S_1[i] - S_2[j]|$
- Chèn "_" vào ngay vị trí i để ghép với $S_2[j]$, vậy tất nhiên lúc này ta chỉ cần tính tổng khoảng cách giữa $S_1[1..i-1]$ và $S_2[1..j]$ với k : $f[i][j] = f[i-1][j] + k$
- Tương tự, chèn "_" vào vị trí j để ghép với $S_1[i]$: $f[i][j] = f[i][j-1] + k$

Từ đó, ta rút ra được công thức truy hồi tổng quát:

$$f[i][j] = \min \begin{cases} f[i-1][j-1] + |S_1[i] - S_2[j]| \\ f[i-1][j] + k \\ f[i][j-1] + k \end{cases}$$

Kết quả bài toán: $f[m][n]$

[Groot]: Ủa, tao thấy trong công thức mày chỉ cho 3 lựa chọn thôi. Chẳng lẽ không còn cách ghép nào khác hả? Lỡ tao muốn vừa chèn ở S_1, vừa chèn ở S_2 cùng lúc thì sao?

[vuivethoima]: Nếu mày chèn cả hai bên cùng lúc thì thực chất mày chỉ đang tăng thêm độ dài cả hai chuỗi bằng dấu "_". Khoảng cách giữa "_" và "_" bằng 0, mà làm vậy thì vô nghĩa vì không thay đổi gì cả. Thế nên không cần xét thêm tình huống này.

Cài đặt

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

signed main() {
    string S1, S2;
    cin >> S1 >> S2;
    int m = S1.size();
    int n = S2.size();
    S1 = "␣" + S1;
    S2 = "␣" + S2;

    int k; cin >> k;

    vector<vector<int>> f(m + 1, vector<int>(n + 1, oo));
    for (int i = 0; i <= m; i++) {
        f[i][0] = i * k;
    }
    for (int j = 0; j <= n; j++) {
        f[0][j] = j * k;
    }

    for (int i = 1; i <= m; i++) {
        for (int j = 1; j <= n; j++) {
            f[i][j] = min(f[i][j], f[i - 1][j - 1] + abs(S1[i] - S2[j]));
            f[i][j] = min(f[i][j], f[i - 1][j] + k);
            f[i][j] = min(f[i][j], f[i][j - 1] + k);
        }
    }
    cout << f[m][n];
}
```

Bài tập 10. Rectangle Cutting **link:** https://cses.fi/problemset/task/1744

Cho hình chữ nhật kích thước $a \times b$, cần cắt nó thành các hình vuông. Ở mỗi lượt, ta có thể chọn một hình chữ nhật bất kỳ và cắt nó thành hai hình chữ nhật (các cạnh sau khi cắt đều là số nguyên dương).
Hãy tính số thao tác cắt tối thiểu để cắt hình chữ nhật $a \times b$ thành các hình vuông.

Input

- Dòng duy nhất chứa hai số nguyên a, b $(1 \leq a, b \leq 500)$.

Output In ra số thao tác cắt tối thiểu.

Ví dụ

Sample Input	Sample Output
3 5	3

Phân tích bài toán

Gọi $f[a][b]$ là số thao tác tối thiểu để cắt hình chữ nhật $a \times b$ thành các hình vuông.

Trường hợp cơ sở:

- Với các hình chữ nhật có cạnh $i = j$, nó đã là một hình vuông, không cần tốn thao tác nào cả, vậy:

$$f[i][i] = 0, \forall i \in [1..\min(a, b)]$$

- Với các hình chữ nhật có cạnh i x j $(i \neq j)$, ta có 2 cách cắt:
  - Cắt dọc: chọn $k \in [1..j-1]$, ta chia được hình chữ nhật thành:
    - Hình chữ nhật $i \times k$ và hình chữ nhật i x $(j-k)$
    - Tổng số bước sẽ là: $f[i][j] = min(f[i][j], f[i][k] + f[i][j-k] + 1)$
  - Cắt ngang: chọn $k \in [1..i-1]$, ta chia được hình chữ nhật thành:
    - Hình chữ nhật $k \times j$ và hình chữ nhật $(i-k) \times j$
    - Tổng số bước sẽ là: $f[i][j] = min(f[i][j], f[k][j] + f[i-k][j] + 1)$

Từ những tính toán trên, ta rút ra được công thức truy hồi tổng quát:

$$f[i][j] = \begin{cases} 0, & i == j \\ \min(\min_{k=1}^{j-1}(f[i][k] + f[i][j-k] + 1), \min_{k=1}^{i-1}(f[k][j] + f[i-k][j] + 1)), & \forall i \in [1..a], \forall j \in [1..b], i \neq j \end{cases}$$

Kết quả bài toán: $f[a][b]$

[Groot]: Ủa, sao công thức chỉ xét cắt ngang với cắt dọc thôi? Có khi nào cắt chéo ra hình vuông nhanh hơn không?

[vuivethoima]: Vãi cả l**, mày nghiêm túc à? Cắt chéo thì hai phần không còn là hình chữ nhật nữa?

[Groot]: Rồi đùa thôi đừng nóng, nhưng tao thấy trong công thức mày cộng thêm "+1". Tại sao vậy? Không phải mình đã tính $f[\cdot][\cdot]$ cho hai mảnh nhỏ rồi sao?

[vuivethoima]: "+1" là chi phí của lần cắt hiện tại. Mỗi lần chia ra hai mảnh thì dù mảnh nào cũng tính riêng, nhưng thao tác cắt vẫn phải đếm thêm một bước nữa.

Cài đặt

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

signed main() {
    int a, b; cin >> a >> b;
    vector<vector<int>> f(a + 1, vector<int>(b + 1, oo));
    for (int i = 1; i <= min(a,b); i++) {
        f[i][i] = 0;
    }

    for (int i = 1; i <= a; i++) {
        for (int j = 1; j <= b; j++) {
            if (i == j) continue;
            for (int k = 1; k <= i - 1; k++) {
                f[i][j] = min(f[i][j], f[k][j] + f[i - k][j] + 1);
            }
            for (int k = 1; k <= j - 1; k++) {
                f[i][j] = min(f[i][j], f[i][k] + f[i][j - k] + 1);
            }
        }
    }
    cout << f[a][b];
}
```

CHƯƠNG 4

QUY HOẠCH ĐỘNG NÂNG CAO

CHƯƠNG 5

KỸ THUẬT ĐỔI BIẾN SỐ TRONG QUY HOẠCH ĐỘNG

CHƯƠNG 6

BITMASK + DYNAMIC PROGRAMMING

CHƯƠNG 7

SOS

CHƯƠNG 8

DIGIT DYNAMIC PROGRAMMING

CHƯƠNG 9

MATRIX MULTIPLICATION DYNAMIC PROGRAMMING

CHƯƠNG 10

OPTIMIZE DP BY DIVIDE AND CONQUER

CHƯƠNG 11

OPTIMIZE DP BY KNUTH - YAO

BIBLIOGRAPHY

[CP-] CP-Algorithms. *CP-Algorithms*. URL: `https://cp-algorithms.com/` (visited on 08/26/2025).

[Kho23] Dinh Nguyen Khoi. *Competitive Programming 10*. drive, 2023.

[VNO] VNOI. *VNOI Wiki*. URL: `https://wiki.vnoi.info/` (visited on 08/26/2025).