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 29 tháng 8 năm 2025

Mục lục

CHƯƠNG 1

GIỚI THIỆU

Contents

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-LQCoEoVkipVAvCzwrrOnJ6GzcAd47P6LO1Rwc`
- [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

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.
- $2 \leq N \leq 10^5$
- $1 \leq h_i \leq 10^4$

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).

Kết quả bài toán: $f[n]$

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: $f[3] = \min(f[2] + |h[3] - h[2]|,\ f[1] + |h[3] - h[1]|)$
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: $f[4] = \min(f[2] + |h[4] - h[2]|,\ f[3] + |h[4] - h[3]|)$
Từ những tính toán trên, ta rút ra được công thức truy hồi tổng quát:

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

[Groot]: 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]: 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 $(1 \leq N \leq 6 \cdot 10^4)$
- Dòng thứ hai ghi N số nguyên dương $t_1, t_2, \ldots, t_N (1 \leq t_i \leq 3 \cdot 10^4)$
- Dòng thứ ba ghi $N-1$ số nguyên dương $r_1, r_2, \ldots, r_{N-1} (1 \leq r_i \leq 3 \cdot 10^4)$

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: $f[1] = t[1]$

Kết quả bài toán: $f[N]$.

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:

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

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

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 **Trường hợp cơ sở:** $\forall i, f[i] = 1$.

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

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 $a[2] \leq a[1]$ 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 $a[3] \leq a[1]$, hay $f[3] = f[2] + 1$ $(f[2] = 1)$

- $a[3] > a[1]$ và $a[3] \leq a[2]$, 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:

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

[Groot]: Ủa tự nhiên ở đâu lòi ra cái $\max_{j<i, a[i]>a[j]}(f[j]+1)$ 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 $(1 \leq N \leq 1000)$.
- Dòng thứ 2 ghi N số nguyên $A_1, A_2, \ldots, A_N (1 \leq A_i \leq 1000)$.

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 $= 0$).

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

Kết quả bài toán: $f[|s|][|t|]$

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 $s[i] \neq t[j]$, 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:

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

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.

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

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

[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;

```

```cpp

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));

    for (int i = 1; i <= N; i++) {
        for (int j = 0; j <= W; j++) {
            if (j < w[i]) {
                f[i][j] = f[i - 1][j];
            } else {
                f[i][j] = max(f[i - 1][j],
                              f[i - 1][j - w[i]] + v[i]);
            }
        }
    }

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

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 ($1 \le i \le N$), 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 ($1 \le N \le 10^5$).
- N dòng tiếp theo, mỗi dòng gồm ba số nguyên a_i, b_i, c_i ($1 \le a_i, b_i, c_i \le 10^4$) – 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 ($0 \le j \le 2$) trong ngày đó (0 là hoạt động A, 1 là hoạt động B, 2 là hoạt động C).

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

Kết quả bài toán: $\max\left(f[N][0],\ f[N][1],\ f[N][2]\right)$

Vì không được tham gia cùng 1 hoạt động 2 ngày liên tiếp nhau, nên với ngày thứ i ($i \ge 2$) 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:

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

Hay:

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

Cài đặt

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

```

```cpp
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:

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

Kết quả bài toán: $\max\left(f[F][j]\right), \forall j \in [V, F]$

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

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

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 $\text{answer} -= f[i][j]$ 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 $(3 \le N \le 5000)$ — 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 $f[i][j] = 0$

Kết quả bài toán: $f[1][n]$

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:

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

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

```
#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 $S_2 : f[i][0] = i * k$

Kết quả bài toán: $f[m][n]$

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:

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

[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, po));
    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

Bài tập 11. Trộn xâu **link:** https://oj.vnoi.info/problem/stmerge

Cho 2 chuỗi X gồm N ký tự và Y gồm M ký tự.
$X = X_1X_2 \dots X_N$
$Y = Y_1Y_2 \dots Y_M$

Hãy trộn 2 chuỗi X và Y này lại thành 1 chuỗi T gồm $N + M$ ký tự sao cho vẫn bảo toàn được thứ tự xuất hiện của các ký tự trong 2 chuỗi.
Ví dụ: $X = X_1X_2$, $Y = Y_1Y_2Y_3$
Một cách trộn hợp lệ: $T = X_1Y_1Y_2X_2Y_3$

Xét 2 ký tự kề nhau $T[i], T[i+1]$: - Nếu chúng cùng thuộc X hoặc cùng thuộc Y thì chi phí cộng thêm là 0. - Nếu một thuộc X, một thuộc Y thì chi phí cộng thêm là $cost[p][q]$ với p, q tương ứng vị trí trong X, Y.
Input

- Dòng đầu tiên chứa số Q – số bộ dữ liệu.
- Mỗi test:
  - Dòng đầu tiên chứa hai số N, M $(1 \le N, M \le 10^3)$.
  - Dòng tiếp theo chứa M số nguyên – $cost[1][j]$ với $j = 1..M$.
  - $N - 1$ dòng tiếp theo, mỗi dòng chứa M số nguyên – các giá trị $cost[i][j]$ $(1 \le i \le N, 1 \le j \le M)$.

Output Với mỗi test, in ra chi phí nhỏ nhất để trộn.
Ví dụ

Sample Input	Sample Output
1 2 3 3 2 30 15 5 4	6

Phân tích bài toán
Gọi $f[i][j][k]$ là tổng chi phí nhỏ nhất khi trộn $X[1..i]$ và $Y[1..j]$

- $k = 0$: ký tự cuối cùng thuộc chuỗi X
- $k = 1$: ký tự cuối cùng thuộc chuỗi Y

Trường hợp cơ sở:

- $f[1][0][0] = 0$
- $f[0][1][1] = 0$
- $f[i][0][0] = 0, \forall i \ge 2$
- $f[0][j][1] = 0, \forall j \ge 2$

Kết quả bài toán: $\min(f[M][N][0], f[M][N][1])$

Xét $X[1..i]$ và $Y[1..j]$, nếu ký tự cuối cùng thuộc X, ta có 2 trường hợp xảy ra:

- $f[i][j][0] = f[i-1][j][1] + cost[i][j]$: Chuyển từ Y sang X

- $f[i][j][0] = f[i-1][j][0] + 0$: Chuyển từ X sang X.

Ngược lại nếu ký tự cuối cùng thuộc Y, ta có 2 trường hợp xảy ra:

- $f[i][j][1] = f[i][j-1][0] + cost[i][j]$: Chuyển từ X sang Y
- $f[i][j][1] = f[i][j-1][1] + 0$: Chuyển từ Y sang Y

Từ những phân tích trên, ta rút ra được công thức truy hồi tổng quát:

$$f[i][j][k] = \min\left(\begin{cases} f[i-(k==0)][j-(k==1)][k], \\ f[i-(k==0)][j-(k==1)][1-k] + cost[i][j] \end{cases} \right)$$

[Groot]: Ủa, tao để ý nè. Mấy bài trước thì toàn $f[i]$ hay $f[i][j]$ thôi, tức là 1 chiều, 2 chiều. Mà qua bài này tự nhiên lại mọc thêm cái k, thành 3 chiều. Sao biết được bao nhiêu chiều mới đủ? Lỡ bài khác phải tới 4–5 chiều thì sao?

[vuivethoima]: Vậy nên tao mới đặt bài này trong chương "QHĐ Nâng cao". Nói chứ chiều của DP không phải "càng nhiều càng xịn", mà là đủ để mô tả trạng thái cần nhớ thôi. Ví dụ:

- Bài Ếch chỉ cần nhớ vị trí hiện tại $\Rightarrow$ 1 chiều.
- Bài LCS phải nhớ 2 con trỏ đang đứng ở đâu trong s và t $\Rightarrow$ 2 chiều.
- Bài này ngoài số ký tự đã dùng ở X và Y, còn phải nhớ thêm ký tự cuối cùng thuộc chuỗi nào để tính phí đổi chuỗi $\Rightarrow$ thêm 1 chiều nữa.

[Groot]: Vậy tức là số chiều DP = số "thông tin" mình cần nhớ lại để tính tiếp hả?

[vuivethoima]: Ừ thông minh hơn rồi đó. Mày cần nhớ cái gì thì định nghĩa thêm một chiều cho nó, chứ không phải thích nhét thêm bao nhiêu chiều thì nhét. Có mấy bài "hardcore" thì thật sự cần 4–5 chiều hay nhiều hơn thiệt, nhưng lúc code người ta hay tìm cách nén trạng thái hoặc rút gọn lại cho nhẹ, bấn quá mới phải đẻ ra nhiều chiều. Hoặc mày cảm thấy cái nhiều chiều đó ngầu thì cứ việc, rồi nộp bài ăn quả TLE xong tạch giải OLP thì đừng có khóc với tao.

[Groot]: À hiểu rồi. Không phải ngồi đoán mò số chiều cho vui mà phải xem bài toán bắt mình nhớ những thông tin nào để đi tiếp. Rồi từ đó mới quyết định cần mấy chiều.

Cài đặt

```cpp
#include <bits/stdc++.h>
#define int long long
const int oo = 1e18 + 7;
using namespace std;

signed main() {
    int q; cin >> q;
    while (q--) {
        int m, n; cin >> m >> n;
        vector<vector<int>> cost(m + 1, vector<int>(n + 1));
        for (int i = 1; i <= m; i++) {
            for (int j = 1; j <= n; j++) {
                cin >> cost[i][j];
            }
        }

        vector<vector<vector<int>>> f(m + 1, vector<vector<int>>(n + 1, vector<int>(2, oo)));

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

        for (int i = 1; i <= m; i++) {
            for (int j = 1; j <= n; j++) {
                f[i][j][0] = min(f[i - 1][j][0], f[i - 1][j][1] + cost[i][j]);

                f[i][j][1] = min(f[i][j - 1][1], f[i][j - 1][0] + cost[i][j]);
            }
        }
        cout << min(f[m][n][0], f[m][n][1]) << endl;
    }
}
```

Bài tập 12. Trò chơi với băng số **link:** https://oj.vnoi.info/problem/linegame

Có một băng số gồm n ô vuông, đánh số từ 1 đến n. Ô vuông thứ i ghi một số nguyên dương a_i.
Trong một lượt chơi, người tham gia chọn một dãy các ô $(i_1, i_2, \ldots, i_k)$ theo thứ tự từ trái sang phải. Điểm số thu được là:

$$a_{i_1} - a_{i_2} + a_{i_3} - a_{i_4} + \cdots + (-1)^{k-1} a_{i_k}$$

Hãy tính số điểm lớn nhất có thể đạt được từ một lượt chơi.

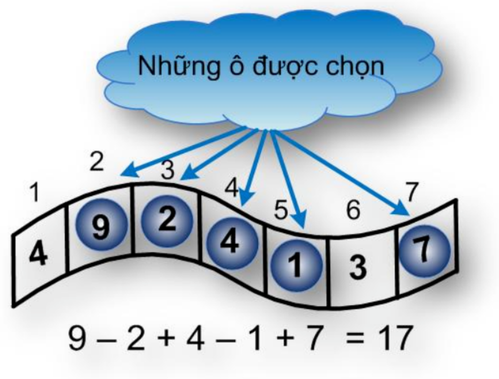
Input

- Dòng đầu tiên chứa số n $(1 \le n \le 10^6)$.
- Dòng thứ hai chứa n số nguyên a_i $(1 \le a_i \le 10^4)$.

Output In ra số điểm lớn nhất có thể đạt được.

Ví dụ

Sample Input	Sample Output
7 4 9 2 4 1 3 7	17



Hình 4.1: Hình minh họa bài toán.

Phân tích bài toán

Gọi $f[i][k]$ là số điểm lớn nhất có thể đạt được khi xét i ô đầu tiên:

- $k = 0$: ô i được chọn và đóng vai trò dấu $+$.
- $k = 1$: ô i được chọn và đóng vai trò dấu $-$.

Trường hợp cơ sở: $f[1][0] = a[1], \quad f[1][1] = -a[1], \quad f[0][0] = f[0][1] = 0.$

Kết quả bài toán: $\max(f[n][0],\ f[n][1])$

Xét ô i, ta có các lựa chọn như sau:

- Nếu $k = 0$ (ô i mang dấu $+$):
  - Nối tiếp từ trạng thái trước đó có dấu $-$: $f[i][0] = f[i-1][1] + a[i]$.
  - Bắt đầu một dãy mới tại ô i: $f[i][0] = a[i]$.
  - Bỏ qua ô i: $f[i][0] = f[i-1][0]$.

  Vậy: $f[i][0] = \max\left(f[i-1][1] + a[i],\ a[i],\ f[i-1][0]\right)$

- Nếu $k = 1$ (ô i mang dấu $-$):
  - Nối tiếp từ trạng thái trước đó có dấu $+$: $f[i][1] = f[i-1][0] - a[i]$.
  - Bắt đầu một dãy mới tại ô i: $f[i][1] = -a[i]$.
  - Bỏ qua ô i: $f[i][1] = f[i-1][1]$.

  Vậy: $f[i][1] = \max\left(f[i-1][0] - a[i],\ -a[i],\ f[i-1][1]\right)$

Cài đặt

```cpp
#include <bits/stdc++.h>
#define int long long
using namespace std;
const int oo = 1e18 + 7;

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

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

        f[i][1] = max(f[i][1], f[i - 1][1]);
        f[i][1] = max(f[i][1], f[i - 1][0] - a[i]);
    }

    int ans = -oo;

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

Bài tập 13. Two Paths **link:** https://lqdoj.edu.vn/problem/twopaths

Hai anh em An và Bình tham gia một trò chơi thám hiểm trên bảng số **xTremeMaze**.
Bảng có kích thước $N \times M$ (N dòng và M cột). Mỗi ô (i, j) chứa một số nguyên (có thể âm hoặc dương) biểu diễn điểm kinh nghiệm nhận được khi đi vào ô đó.
An và Bình bắt đầu tại ô $(1, 1)$ và kết thúc tại ô (N, M). Mỗi lượt, một người chỉ được đi xuống hoặc sang phải, và không được đi ra ngoài bảng. Điểm kinh nghiệm tại ô $(1, 1)$ và (N, M) luôn bằng 0.
Yêu cầu: tìm tổng điểm kinh nghiệm lớn nhất mà hai anh em đạt được, với điều kiện hai người không đi qua cùng một ô, ngoại trừ ô $(1, 1)$ và ô (N, M).

Input

- Dòng đầu tiên chứa hai số nguyên N, M ($2 \le N, M \le 200$).
- N dòng tiếp theo, mỗi dòng chứa M số nguyên có giá trị tuyệt đối không quá 100 — điểm kinh nghiệm của từng ô.
- Bảo đảm $A_{1,1} = A_{N,M} = 0$.

Output In ra tổng số điểm kinh nghiệm lớn nhất mà An và Bình đạt được.

Ví dụ

Sample Input	Sample Output
3 3 0 2 3 4 5 6 7 8 0	32

Phân tích bài toán

Gọi $f[\text{step}][x_1][x_2]$ là tổng số điểm kinh nghiệm lớn nhất khi An và Bình đã thực hiện $\text{step} = x + y - 2$ bước, trong đó:

- An đang ở vị trí (x_1, y_1) với $y_1 = \text{step} - x_1 + 2$,
- Bình đang ở vị trí (x_2, y_2) với $y_2 = \text{step} - x_2 + 2$.

Tổng số bước di chuyển là $n + m - 2$, do đó ta duyệt lần lượt các bước $\text{step} = 1 \to n + m - 2$.

Trường hợp cơ sở: $f[0][1][1] = 0$

Kết quả bài toán: $f[n + m - 2][n][n]$

Công thức truy hồi: Tại mỗi bước, giá trị $f[\text{step}][x_1][x_2]$ được cập nhật từ 4 trạng thái trước đó, ứng với việc An/Bình đi xuống hoặc sang phải:

$$\begin{aligned} f[\text{step}][x_1][x_2] = \max\Big\{ & f[\text{step}-1][x_1-1][x_2-1], \\ & f[\text{step}-1][x_1-1][x_2], \\ & f[\text{step}-1][x_1][x_2-1], \\ & f[\text{step}-1][x_1][x_2] \Big\} + \text{điểm} \end{aligned}$$

Trong đó phần *điểm* được tính như sau:

$$\text{nếu } (x_1, y_1) = (x_2, y_2) \Rightarrow \text{điểm} = a[x_1][y_1]$$

$$\text{ngược lại } \Rightarrow \text{điểm} = a[x_1][y_1] + a[x_2][y_2]$$

Ngoài ra, cần đảm bảo các điều kiện sau:

- (x_1, y_1) và (x_2, y_2) phải nằm trong bảng.
- Nếu $(x_1, y_1) = (x_2, y_2)$ thì đó chỉ có thể là ô đích (n, m).

[Groot]: Ê sao phức tạp vậy, tao vừa nghĩ ra cái này. Sao không chơi thẳng $f[x_1][y_1][x_2][y_2]$ luôn? Hai thằng đi đâu thì mình lưu hết bốn toạ độ, rõ ràng dễ hiểu, hạnh phúc, thành công, khai phóng vcđ.

[vuivethoima]: Ừ, ý tưởng đó là cách tự nhiên nhất, cũng đúng luôn. Nhưng nó hoang dã vl, mày thử coi: N, M tối đa 200. Nếu làm 4D thì số trạng thái là $200^4 \approx 1.6 \times 10^9$. Với mỗi trạng thái lại duyệt 4 cách đi tiếp, thì TLE chếc cụ ấy.

[Groot]: À, tức là dễ viết nhưng chậm. Vậy thì cái trick ở đây là gì?

[vuivethoima]: Trick là "đồng bộ bước đi". Mỗi lần cả An lẫn Bình đi thêm 1 bước, thì tổng chỉ số $x + y$ của họ đều tăng lên đúng 1. Nói cách khác, sau step bước thì $x_1 + y_1 = x_2 + y_2 = \text{step} + 2$. Vậy là từ 4 toạ độ (x_1, y_1, x_2, y_2) mình rút gọn còn 3 thông tin: (step, x_1, x_2). Khi đó y_1, y_2 tính lại được từ step.

[Groot]: À ha, nên mới có công thức $y_1 = \text{step} - x_1 + 2$, $y_2 = \text{step} - x_2 + 2$ đó hả. Nhờ vậy bộ nhớ giảm từ $O(N^2M^2)$ xuống $O((N + M)N^2)$, đủ chơi với cái giới hạn bài cho.

[Groot]: Vậy có khi nào gặp bài mà không nén được như vầy, bắt buộc phải chơi 4D không?

[vuivethoima]: Chắc có, nếu hai thực thể di chuyển độc lập mà không có ràng buộc "đồng bộ" giữa chúng thì đúng là phải 4D, tao nghĩ vậy?

Cài đặt

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


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

    vector<vector<vector<int>>> f(n + m, vector<vector<int>>(n + 1, vector<int>(n + 1, -oo)));

    f[0][1][1] = 0;

    for (int step = 1; step <= n + m - 2; step++) {
        for (int x1 = 1; x1 <= n; x1++) {
            int y1 = step - x1 + 2;
            if (y1 <= 0 || y1 > m) continue;
            for (int x2 = 1; x2 <= n; x2++) {
                int y2 = step - x2 + 2;
                if (y2 <= 0 || y2 > m) continue;

                int val = -oo;
                val = max(val, f[step - 1][x1 - 1][x2]);
                val = max(val, f[step - 1][x1][x2 - 1]);
```

```
                    val = max(val, f[step - 1][x1 - 1][x2 - 1]);
                    val = max(val, f[step - 1][x1][x2]);


                    if (x1 == x2 && y1 == y2 && (x1 != n || y1 != m)) continue;

                    if (x1 == x2 && y1 == y2)
                        val += a[x1][y1];
                    else
                        val += a[x1][y1] + a[x2][y2];
                    f[step][x1][x2] = val;
                }
            }
        }
    cout << f[n + m - 2][n][n];
}
```

Bài tập 14. IOI07 Miners **link:** https://oj.vnoi.info/problem/nkminers

Có hai mỏ than, mỗi mỏ có một nhóm thợ mỏ làm việc. Khai thác than là công việc vất vả, do đó các thợ mỏ cần thực phẩm để hoạt động. Mỗi khi một đợt vận chuyển thực phẩm đến mỏ, các thợ mỏ sẽ khai thác được một lượng than nào đó. Có 3 loại thực phẩm được vận chuyển: thịt (M), cá (F) và bánh mì (B).

Mỗi đợt vận chuyển thực phẩm được đưa đến một trong hai mỏ, và sản lượng than của đợt đó phụ thuộc vào số loại thực phẩm khác nhau trong hai đợt liên tiếp mà mỏ đó nhận được:

- Nếu các đợt vận chuyển cùng một loại thực phẩm $\Rightarrow$ 1 đơn vị than
- Nếu có 2 loại thực phẩm khác nhau $\Rightarrow$ 2 đơn vị than
- Nếu có 3 loại thực phẩm khác nhau $\Rightarrow$ 3 đơn vị than

Các đợt vận chuyển không thể chia nhỏ, và tất cả thực phẩm trong một đợt phải gửi đến một trong hai mỏ. Có thể gửi tất cả các đợt đến một mỏ.

Hãy tìm cách phân chia các đợt vận chuyển sao cho tổng lượng than khai thác được từ hai mỏ là lớn nhất.

Input

- Dòng đầu tiên chứa số nguyên N $(1 \leq N \leq 10^5)$ — số đợt vận chuyển.
- Dòng thứ hai chứa xâu S độ dài N, mỗi ký tự là một trong {M, F, B}.

Output In ra tổng lượng than lớn nhất có thể khai thác.

Ví dụ

Sample Input	Sample Output
6 MBMFFB	12
16 MMBMBBBBMMMMMBMB	29

Phân tích bài toán

Gọi $f[i][a_1][a_2][b_1][b_2]$ là tổng lượng than lớn nhất có thể sản xuất được sau i đợt vận chuyển, trong đó:

- a_1, a_2 là hai loại thực phẩm gần nhất mỏ 1 đã nhận (Hiện tại là a_1, ngày hôm trước là a_2).
- b_1, b_2 là hai loại thực phẩm gần nhất mỏ 2 đã nhận (Hiện tại là b_1, ngày hôm trước là b_2).
- Giá trị của mỗi loại: 0 (không có), 1 (M), 2 (F), 3 (B).

Trường hợp cơ sở:

Giả sử thực phẩm đầu tiên là loại $t = \texttt{code}(s[1])$, ta có:

$$f[1][t][0][0][0] = 1 \quad (\text{gửi đến mỏ 1})$$
$$f[1][0][0][t][0] = 1 \quad (\text{gửi đến mỏ 2})$$

Kết quả bài toán:

$$\max_{a,b,c,d \in \{0,1,2,3\}} f[n][a][b][c][d]$$

Với $i \geq 2$, giả sử loại thực phẩm hiện tại là $c = \texttt{code}(s[i])$, ta có hai lựa chọn:

- **Chuyển đến mỏ 1**:

$$f[i][c][a_1][b_1][b_2] = \max(f[i][c][a_1][b_1][b_2], f[i-1][a_1][a_2][b_1][b_2] + \texttt{energy}(c, a_1, a_2))$$

- **Chuyển đến mỏ 2**:

$$f[i][a_1][a_2][c][b_1] = \max(f[i][a_1][a_2][c][b_1], f[i-1][a_1][a_2][b_1][b_2] + \texttt{energy}(c, b_1, b_2))$$

Trong đó, hàm `energy`(a, b, c) là hàm đếm số loại thực phẩm khác nhau trong 3 ngày gần nhất:

$$\texttt{energy}(a, b, c) = |\{a, b, c\} \setminus \{0\}|$$

Cài đặt

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

int code(char c) {
    return (c == 'M' ? 1 : (c == 'F' ? 2 : (c == 'B' ? 3 : 0)));
}

int energy(int a, int b, int c) {
    return (a != 0) + (b != 0 && b != a) + (c != 0 && c != a && c != b);
}

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

    int f[n + 1][4][4][4][4];
    for (int i = 0; i <= n; i++)
        for (int a1 = 0; a1 < 4; a1++)
            for (int a2 = 0; a2 < 4; a2++)
                for (int b1 = 0; b1 < 4; b1++)
                    for (int b2 = 0; b2 < 4; b2++)
                        f[i][a1][a2][b1][b2] = -oo;

    int t = code(s[1]);
    f[1][t][0][0][0] = 1;
    f[1][0][0][t][0] = 1;

    for (int i = 2; i <= n; i++) {
        int cur = code(s[i]);
        for (int t1 = 0; t1 <= 3; t1++) {
            for (int t2 = 0; t2 <= 3; t2++) {
                for (int t3 = 0; t3 <= 3; t3++) {
                    for (int t4 = 0; t4 <= 3; t4++) {
                        f[i][cur][t1][t3][t4] = max(f[i][cur][t1][t3][t4],
                                                    f[i - 1][t1][t2][t3][t4] + energy(cur, t1,t2));

                        f[i][t1][t2][cur][t3] = max(f[i][t1][t2][cur][t3],
                                                    f[i - 1][t1][t2][t3][t4] + energy(cur,t3,t4));
                    }
                }
            }
        }
    }


    int ans = 0;
    for (int a1 = 0; a1 < 4; ++a1)
        for (int a2 = 0; a2 < 4; ++a2)
            for (int b1 = 0; b1 < 4; ++b1)
                for (int b2 = 0; b2 < 4; ++b2)
                    ans = max(ans, f[n][a1][a2][b1][b2]);
    cout << ans;
}
```

[Groot]: Ủa, nhưng tao coi lời giải trên mạng thấy có người làm 3 chiều, có người 4 chiều thôi à. Sao mày lại dựng hẳn tới 5 chiều? Có dư thừa gì không?

[vuivethoima]: Ờ thì cách tao viết là kiểu full state - giữ luôn 2 món gần nhất của mỗi mỏ. Đảm bảo tính đúng, nhưng hơi minh béo.

[Groot]: Vậy sao mày không làm tối ưu hơn, cách mày cài đặt tốn $4^4 \times 10^5$ ô nhớ lận. Viết kiểu đó làm cái gì cho tốn bộ nhớ?

[Groot]: Mày coi thử cách giảm chiều của tao bên dưới oke hông?

Phân tích bài toán (2):
Gọi $f[i][a][b][c]$ là tổng lượng than lớn nhất có thể sản xuất được sau i đợt vận chuyển, trong đó:

- Mỏ vừa nhận đợt i (gọi là mỏ *đang hoạt động*) có hai loại thực phẩm gần nhất là $(s[i], a)$, tức $s[i]$ là loại hiện tại, a là loại liền trước đó.
- Mỏ còn lại có hai loại thực phẩm gần nhất là (b, c).
- Giá trị của mỗi loại: 0 (không có), 1 (M), 2 (F), 3 (B).

Trường hợp cơ sở:
Với đợt vận chuyển đầu tiên có loại $t = \texttt{code}(s[1])$, ta có: $f[1][0][0][0] = 1$ vì $\texttt{energy}(t, 0, 0) = 1$.
Kết quả bài toán:

$$\max_{a,b,c\in\{0,1,2,3\}} f[n][a][b][c]$$

Công thức chuyển:
Giả sử ở đợt i hiện tại loại thực phẩm là $c = \texttt{code}(s[i])$, khi xét đến đợt tiếp theo $s[i+1]$ có loại $t = \texttt{code}(s[i+1])$, ta có hai lựa chọn:

- **Tiếp tục chuyển đến mỏ đang hoạt động**: Khi đó hai món gần nhất của mỏ này là $(s[i], a)$, nên điểm cộng thêm là $\texttt{energy}(t, s[i], a)$. Trạng thái mới ở bước $i + 1$ sẽ là:

$$f[i+1][s[i]][b][c] = \max\Big(f[i+1][s[i]][b][c],\ f[i][a][b][c] + \texttt{energy}(t, s[i], a)\Big)$$

- **Chuyển sang mỏ còn lại**: Mỏ còn lại hiện có hai món gần nhất là (b, c), nên điểm cộng thêm là $\texttt{energy}(t, b, c)$. Sau khi giao, mỏ này trở thành mỏ đang hoạt động với (t, b), còn mỏ cũ trở thành mỏ còn lại với $(s[i], a)$. Do đó:

$$f[i+1][b][s[i]][a] = \max\Big(f[i+1][b][s[i]][a],\ f[i][a][b][c] + \texttt{energy}(t, b, c)\Big)$$

Cài đặt

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

int code(char c) {
    return (c == 'M' ? 1 : (c == 'F' ? 2 : (c == 'B' ? 3 : 0)));
}

int energy(int a, int b, int c) {
    return (a != 0) + (b != 0 && b != a) + (c != 0 && c != a && c != b);
}

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

    int f[n + 1][4][4][4];
    for (int i = 0; i <= n; i++)
        for (int a = 0; a < 4; a++)
            for (int b = 0; b < 4; b++)
                for (int c = 0; c < 4; c++)
                    f[i][a][b][c] = -1;

    f[0][0][0][0] = 0;
    f[1][0][0][0] = 1;
    for (int i = 0; i < n; i++) {
        for (int a = 0; a < 4; a++) {
            for (int b = 0; b < 4; b++) {
                for (int c = 0; c < 4; c++) {
                    if (f[i][a][b][c] == -1) continue;

                    f[i + 1][code(s[i])][b][c] = max(f[i + 1][code(s[i])][b][c], f[i][a][b][c] + energy(code(
                        s[i]), code(s[i + 1]), a));

                    f[i + 1][b][code(s[i])][a] = max(f[i + 1][b][code(s[i])][a], f[i][a][b][c] + energy(code(
                        s[i+1]), b, c));
```

```
                }
            }
        }
    }

    int ans = 0;
    for (int a = 0; a < 4; a++) {
        for (int b = 0; b < 4; b++) {
            for (int c = 0; c < 4; c++) {
                ans = max(ans, f[n][a][b][c]);
            }
        }
    }
    cout << ans;
}
```

Bài tập 15. Exam Cheating **link:** https://codeforces.com/problemset/problem/796/E

Zane và người mà Zane thích vừa mới bắt đầu hẹn hò! Tuy nhiên, cô gái đang gặp khó khăn với kỳ thi cuối kỳ môn Vật lý, và cần bạn giúp đỡ.

Có n câu hỏi, được đánh số từ 1 đến n. Câu hỏi i sẽ đứng trước câu hỏi $i+1$ $(1 \leq i < n)$. Mỗi câu hỏi không thể chọn bừa, vì nếu trả lời sai sẽ bị phạt rất nặng. May mắn thay, cô gái đang ngồi giữa hai thiên tài, và cô ấy sẽ quay cóp bài họ.

Tuy nhiên, hai thiên tài cũng có giới hạn. Mỗi người có thể biết hoặc không biết câu trả lời của một số câu hỏi. Dù vậy, những câu trả lời trên bài của họ thì luôn chính xác tuyệt đối.

Để không bị giám thị phát hiện, cô gái chỉ được quay cóp tối đa p lần, mỗi lần nhìn được nhiều nhất k câu hỏi liên tiếp từ bài của một trong hai thiên tài. Khi cô ấy nhìn vào bài một thiên tài, nếu câu hỏi đó có trong bài của họ, cô ấy sẽ chép lại, nếu không thì bỏ qua.

Nhiệm vụ của bạn là giúp cô ấy trả lời đúng được nhiều nhất bao nhiêu câu hỏi.

Input

- Dòng đầu tiên: ba số nguyên n, p, k $(1 \leq n, p \leq 1000,\ 1 \leq k \leq \min(n, 50))$.
- Dòng thứ hai: số r — số câu hỏi thiên tài 1 biết, tiếp theo r số nguyên phân biệt $a_1, \ldots, a_r$ (tăng dần).
- Dòng thứ ba: số s — số câu hỏi thiên tài 2 biết, tiếp theo s số nguyên phân biệt $b_1, \ldots, b_s$ (tăng dần).

Output In ra số câu hỏi tối đa mà cô gái có thể trả lời đúng.

Ví dụ

Sample Input	Sample Output
6 2 3 3 1 3 6 4 1 2 5 6	4
8 3 3 4 1 3 5 6 5 2 4 6 7 8	7

Ghi chú

Ví dụ 1: Có thể chọn các hành động $(1, 1, 3)$ và $(2, 5, 6)$ để trả lời đúng 4 câu hỏi.
Ví dụ 2: Có thể chọn $(1, 3, 5)$, $(2, 2, 4)$ và $(2, 6, 8)$ để trả lời đúng 7 câu hỏi.

Phân tích bài toán

Gọi hai thiên tài lần lượt là A và B.

Gọi $f[i][j][k_A][k_B]$ là tổng số câu tối đa có thể chép được khi xét i câu trắc nghiệm đầu tiên, ta đã có j đợt chép bài. Và:

- Đối với người A: ta đã chép được k_A câu liên tục.
- Đối với người B: ta đã chép được k_B câu liên tục.

Trường hợp cơ sở: $f[0][0][0][0] = 0$
Nếu $k_A = 0$ và $k_B = 0$: hiện tại câu thứ i của bạn A (hoặc B) không được chép.
Kết quả bài toán:

$$\max_{j \leq p, 0 \leq kA, kB \leq k} f[n][j][k_A][k_B]$$

Cài đặt

Bài tập 16. SC6 **link:** Summer Contest 2022

Cho một dãy số nguyên dương a gồm N số. Bạn hãy chia dãy số trên thành k đoạn, mỗi đoạn gồm một dãy gồm các số ở vị trí liên tục sao cho với mỗi số thuộc đúng một đoạn. Gọi $s_1, s_2, \ldots, s_k$ lần lượt là tổng các số trên mỗi đoạn.
Hỏi bạn cần chia như thế nào để được tổng

$$|s_2 - s_1| + |s_3 - s_2| + \ldots + |s_k - s_{k-1}|$$

lớn nhất.

Dữ liệu đầu vào (SC6.INP):

- Dòng đầu tiên gồm một số nguyên dương T $(1 \leq T \leq 40)$, là số lượng bộ dữ liệu.
- T bộ dữ liệu tiếp theo, mỗi bộ dữ liệu có cấu trúc:
  - Dòng đầu tiên chứa 2 số nguyên dương N và k.
  - Dòng thứ hai chứa N số nguyên (có thể âm), số nguyên dương thứ i có giá trị là a_i.

Dữ liệu đầu ra (SC6.OUT):

- Kết quả cho T bộ dữ liệu, mỗi bộ dữ liệu in ra một số nguyên duy nhất là tổng $|s_2 - s_1| + |s_3 - s_2| + \ldots + |s_k - s_{k-1}|$ lớn nhất.

Ví dụ

Sample Input	**Sample Output**
3 6 4 5 2 1 5 4 6 5 2 1 1 3 6 2 6 3 5 2 4 4 1 3	18 11 16

Giải thích test ví dụ:

- Với test ví dụ 1: Chia dãy thành 4 đoạn: [5], [2], [1], [5, 4, 6] với tổng lần lượt $s_1 = 5, s_2 = 2, s_3 = 1, s_4 = 15$, tổng lớn nhất là $|2-5| + |1-2| + |15-1| = 18$.
- Với test ví dụ 2: Chia dãy như sau: [1], [1], [3, 6, 2] với $s_1 = 1, s_2 = 1, s_3 = 11$, tổng là $|12 - 1| = 11$.
- Với test ví dụ 3: Chia dãy như sau: [5, 2, 4, 4], [1], [3] với $s_1 = 15, s_2 = 1, s_3 = 3$, tổng lớn nhất là $|2-1|+|3-1|+|15-1| = 16$.

Bài toán sẽ được chia làm 2 Dataset:

- Small Dataset: $k = 2$.
- Large Dataset: $2 \leq k \leq \min(N, 200)$.
- Trong mọi dataset, 95% test có $2 \leq N \leq 200$, 5% còn lại có $2 \leq N \leq 10000$.
- Ngoài ra, $-10000 \leq a_i \leq 10000$.

Phân tích bài toán

Gọi $f[i][j]$[`prevState`][`curState`] là tổng lớn nhất có thể khi xét i phần tử đầu tiên của dãy a, hiện tại a_i đang nằm trong đoạn thứ j, prevState là trạng thái của trị tuyệt đối "trước đó" và curState là trạng thái của trị tuyệt đối hiện tại.

Trường hợp cơ sở:

Kết quả bài toán:

Cài đặt

CHƯƠNG 5

KỸ THUẬT ĐỔI BIẾN SỐ TRONG QUY HOẠCH ĐỘNG

Contents

5.1 Lý thuyết

Với kỹ thuật đổi biến số, chúng ta đừng nhìn nhận/gọi $f[\cdot]$ là đáp án của bài toán nữa, mà hãy xem $f[\cdot]$ như là một cái chiều của QHĐ.

5.2 Vấn đề

5.2.1 Bài toán Knapsack

Bài tập 17. Knapsack 2 **link:** https://atcoder.jp/contests/dp/tasks/dp_e
Có N đồ vật, được đánh số từ $1, 2, \ldots, N$. Với mỗi đồ vật i $(1 \leq i \leq N)$, đồ vật i có trọng lượng w_i và giá trị v_i.
Taro muốn chọn một số đồ vật trong N đồ vật này và mang về bằng một chiếc ba lô. Chiếc ba lô có sức chứa W, nghĩa là tổng trọng lượng các đồ vật được chọn không được vượt quá W.
Yêu cầu
Hãy tìm tổng giá trị lớn nhất của các đồ vật mà Taro có thể mang về.
Giới hạn

- $1 \leq N \leq 100$
- $1 \leq W \leq 10^9$
- $1 \leq w_i \leq W$
- $1 \leq v_i \leq 10^3$

Input

- Dòng đầu tiên chứa hai số nguyên N, W — số lượng đồ vật và sức chứa của ba lô.
- Trong N dòng tiếp theo, dòng thứ i chứa hai số w_i, v_i — trọng lượng và giá trị của đồ vật i.

Output In ra một số nguyên duy nhất — tổng giá trị lớn nhất có thể.
Ví dụ

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

Giải thích
Trong ví dụ, chọn đồ vật 1 và 3. Khi đó tổng trọng lượng $3 + 5 = 8$, và tổng giá trị $30 + 60 = 90$.

5.2.2 Nhận xét

Bài toán khác với Knapsack cổ điển ở chỗ giới hạn tải trọng rất lớn ($W \le 10^9$), nên không thể quy hoạch động theo *trọng lượng* như thường lệ (vì chiều theo W sẽ quá lớn).
Ta có thể tư duy rằng **đảo chiều** quy hoạch động theo *giá trị*: đặt

$$f[i][j] = \text{khối lượng nhỏ nhất có thể đạt được khi xét } i \text{ đồ vật đầu tiên và tổng giá trị đúng bằng } j,$$

và quy ước $f[i][j] = +\infty$ nếu không thể đạt tổng giá trị j.
Trường hợp cơ sở:

$$f[0][0] = 0, \qquad f[0][j] = +\infty,\ \forall j > 0.$$

Chuyển trạng thái: với đồ vật i có trọng lượng w_i và giá trị v_i,

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

Kết quả bài toán: $\max\{\, j \mid f[n][j] \le W \,\}$.

Vì j chỉ cần chạy đến $V_{\max} = \sum_{i=1}^{N} v_i \le 10^5$, độ phức tạp là $O(N\, V_{\max})$, phù hợp với giới hạn bài toán.

Cài đặt

```cpp
#include <bits/stdc++.h>
#define int long long
const int oo = 1e18 + 7;
using namespace std;

struct item {
    int w, v;
};

signed main() {
    int n, w; cin >> n >> w;
    vector<item> a(n + 1);
    for (int i = 1; i <= n; i++) {
        cin >> a[i].w >> a[i].v;
    }
    int MAXV = 1e5;
    vector<vector<int>> dp(n + 1, vector<int>(MAXV + 5, oo));

    dp[0][0] = 0;

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

    int ans = 0;
    for (int j = 0; j <= MAXV; j++) {
        if (dp[n][j] <= w)
            ans = j;
    }
    cout << ans;
}
```

5.2.3 Bài toán Dãy con tăng dài nhất

Bài tập 18. Dãy con tăng dài nhất **link:** https://oj.vnoi.info/problem/lis
Cho một dãy gồm N số nguyên ($1 \le N \le 30000$). Hãy tìm dãy con tăng dài nhất trong dãy đó. In ra số lượng phần tử của dãy con. Các số trong phạm vi longint.
Input

- Dòng đầu tiên chứa số nguyên N.
- Dòng thứ hai gồm N số mô tả dãy.

Output In ra một số nguyên duy nhất là đáp số bài toán.
Ví dụ

Sample Input	Sample Output
5 2 1 4 3 5	3

5.2.4 Nhận xét

Với bài toán LIS cơ bản, ta có thể đặt $f[i] =$ độ dài dãy con tăng dài nhất kết thúc tại vị trí i, và duyệt hai vòng `for` để tính toán. Tuy nhiên, với ràng buộc $N \leq 30000$,thì cách làm này có độ phức tạp $O(N^2) \approx 30000^2 = 9 \times 10^8$ phép toán, chắc chắn sẽ bị `TLE`.

Do đó, ta chuyển sang cách định nghĩa khác. Gọi $f[\texttt{len}]$ là **chỉ số** của phần tử kết thúc có giá trị nhỏ nhất trong tất cả những dãy con tăng có độ dài bằng `len` tính đến **thời điểm hiện tại**.

Xét `a[20] = {0, 9, 1, 5, 6, 3, 8, 8, 4, 3, 9, 10, 15, 2, 7, 1, 5, 6, 3, 8}` (a[0] được thêm vào chỉ để thuận tiện cho việc định nghĩa bài toán cơ sở sau này).
Giả sử ta xét $i = 7$:

- Với `len = 2`, ta có những dãy con tăng như sau: [1, 5], [1, 6], [1, 3], [1, 8], [5, 6], [5, 8], [6, 8].
- Vậy $f[2] = 5$ (dãy [a[2], a[5]])

Giả sử ta xét $i = 10$:

- f[3] = 8

Bài toán cơ sở: f[0] $= 0$, $f[i] = -1,\ 1 \leq i \leq n$
Kết quả bài toán: Ta tìm `len` lớn nhất sao cho $f[\texttt{len}] \neq -1$
Phân tích bài toán:
Vì mảng f là một dãy không giảm ($f_i \leq f_{i+1},\ \forall i$), để tìm độ dài dãy con tăng dài nhất kết thúc tại vị trí i, ta phải tìm j lớn nhất thỏa mãn: $a_{f_j} < a_i$. Như vậy, ta có thể tạo ra một dãy con tăng có độ dài $j + 1$.
Chứng minh f là dãy không giảm
Xét định nghĩa: $f[\texttt{len}]$ là chỉ số của phần tử có giá trị nhỏ nhất trong tất cả các dãy con tăng có độ dài đúng bằng `len`, tính đến thời điểm hiện tại.
Ta cần chứng minh:

$$f[1] \leq f[2] \leq \cdots \leq f[\texttt{len}_{\max}]$$

Lý do:

- Một dãy con tăng có độ dài $k + 1$ được tạo ra bằng cách nối thêm một phần tử vào đuôi của một dãy con tăng độ dài k.
- Do đó, nếu $f[k]$ là chỉ số phần tử kết thúc của dãy con tăng độ dài k, thì $f[k + 1]$ phải là chỉ số phần tử nằm *bên phải* $f[k]$ trong mảng gốc.
- Mặt khác, ta luôn chọn phần tử kết thúc có *giá trị nhỏ nhất* cho mỗi độ dài. Điều này đảm bảo rằng chỉ số kết thúc của dãy độ dài $k + 1$ không thể "quay ngược" về trước, tức là không thể nhỏ hơn $f[k]$.

Vậy ta có quan hệ:

$$f[k] \leq f[k + 1], \quad \forall k.$$

Hay nói cách khác, dãy các chỉ số f là **không giảm**.

Do mảng f là một dãy không giảm, ta dễ dàng tìm kiếm được vị trí j lớn nhất bằng `tìm kiếm nhị phân`.

Cài đặt

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

int main() {
    int n; cin >> n;
    vector<int> a(n + 1), f(n + 1, -1);
    for (int i = 1; i <= n; i++) {
        cin >> a[i];
    }
    f[0] = 0;
    a[0] = 0;
    int len = 0;
    for (int i = 1; i <= n; i++) {
```

```
        int l = 0, r = len, pos = 0;
        while (l <= r) {
            int mid = l + r >> 1;
            if (a[i] > a[f[mid]]) {
                pos = mid;
                l = mid + 1;
            }
            else {
                r = mid - 1;
            }
        }
        f[pos + 1] = i;
        if (pos + 1 > len) len++;
    }
    cout << len;
}
```

5.3 Bài tập

Bài tập 19. SC1 **link:** SUMMER CONTEST 2021

Kỳ nghỉ Tết cũng đã kết thúc, Quang đã kiếm chác được một số tiền khá lớn dựa vào tiền lì xì, tiền đánh bài, lô tô, tổ tôm, bầu cua, cá ngựa. Số tiền của Quang tích góp sau Tết sau khi thống kê lại, được a_1 tờ 1.000 đồng, a_2 tờ 2.000 đồng, a_3 tờ 5.000 đồng, a_4 tờ 10.000 đồng, a_5 tờ 20.000 đồng, a_6 tờ 50.000 đồng, a_7 tờ 100.000 đồng, a_8 tờ 200.000 đồng, a_9 tờ 500.000 đồng.

Quang quyết định bỏ tiền của mình vào ống heo để tiết kiệm. Tuy nhiên, Quang sẽ muốn ăn mừng 1 bữa thắng lớn. Thế là Quang chạy ra tiệm trà sữa Giun Giun, Quang gọi 1 ly trà sữa Thái trân châu đường đen full topping, ca cao, phô mai, kem cheese, bánh flan, bánh quế, 100% đường 20% đá size XXL. Quả nhiên là 1 ly trà sữa hảo hạng.

Số tiền để mua ly trà sữa này là C (đồng) và C chia hết cho 1000. Tiệm trà sữa Giun Giun hiện tại đang có b_1 tờ 1.000 đồng, b_2 tờ 2.000 đồng, b_3 tờ 5.000 đồng, b_4 tờ 10.000 đồng, b_5 tờ 20.000 đồng, b_6 tờ 50.000 đồng, b_7 tờ 100.000 đồng, b_8 tờ 200.000 đồng, b_9 tờ 500.000 đồng trong quầy tính tiền.

Vì chú heo đất của Quang không to lắm, nên Quang muốn tống khứ càng nhiều tờ tiền của mình càng tốt. Quang có thể trả tiền cho ly trà sữa với một số tiền là X $(X > C)$ và quán Giun Giun phải thối lại cho Quang $X - C$ đồng. Vì là 1 tiệm kinh doanh trà sữa, nên bà chủ Giun Giun muốn tống khứ càng ít tiền của quán càng tốt.

Hãy giúp Quang tính xem số tiền tối thiểu mà Quang sẽ có nếu thực hiện xong thương vụ mua trà sữa.

Input

- Dòng đầu tiên chứa số nguyên dương T $(1 \le T \le 40)$ là số lượng bộ dữ liệu.
- Mỗi bộ dữ liệu gồm:
  - Dòng đầu tiên chứa số nguyên dương C là giá tiền của ly trà sữa (C chia hết cho 1000).
  - 9 dòng tiếp theo, dòng thứ i có định dạng `c[i]: a[i] b[i]`, trong đó $c[i] \in \{1000, 2000, 5000, 10000, 20000, 50000, 100000, 200000, 500000\}$ là mệnh giá, $a[i]$ là số tờ tiền Quang có, $b[i]$ là số tờ tiền quán có.

Output

- Với mỗi bộ dữ liệu, in ra số tờ tiền mà Quang sẽ có sau khi mua trà sữa.
- Nếu không thể mua, in ra chuỗi `SORRY!!!`.

Ví dụ

Sample Input	Sample Output
Input:	**Output:**
5	
52000	7
1000: 0 1	SORRY!!!
2000: 0 1	10
5000: 5 0	7
10000: 2 0	6
20000: 1 5	
50000: 1 0	
100000: 5 12	
200000: 0 15	
500000: 2 20	
50000	
1000: 0 0	
2000: 3 0	
5000: 9 0	
10000: 0 0	
20000: 0 0	
50000: 0 0	
100000: 0 0	
200000: 0 0	
500000: 0 0	
200000	
1000: 0 0	
2000: 0 0	
5000: 0 0	
10000: 0 0	
20000: 0 0	
50000: 0 0	
100000: 2 0	
200000: 5 0	
500000: 5 0	
600000	
1000: 1 51	
2000: 1 24	
5000: 1 42	
10000: 1 52	
20000: 1 11	
50000: 1 4	
100000: 1 0	
200000: 1 5	
500000: 1 24	
58000	
1000: 3 2	
2000: 1 5	
5000: 0 5	
10000: 2 12	
20000: 5 12	
50000: 9 24	
100000: 1 15	
200000: 4 5	
500000: 9 12	

Giới hạn dữ liệu:

- Small Dataset: $0 \le a[i], b[i] \le 1$
- Large Dataset: $0 \le a[i], b[i] \le 70$
- $C \le 70{,}000{,}000$

Phân tích bài toán:
Bài toán cơ sở: Kết quả bài toán:

Cài đặt

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).