

**HỘI THẢO KHOA HỌC
CÁC TRƯỜNG THPT CHUYÊN
KHU VỰC DUYÊN HẢI VÀ ĐỒNG BẮC BỘ
NĂM 2020**

Môn: Tin học

Chủ đề: QUY HOẠCH ĐỘNG (DYNAMIC PROGRAMMING)

Tháng 9/2020

MỤC LỤC

0. Mở đầu.....	4
1. Một số khái niệm, kiến thức cơ bản.....	5
1.1. Bài toán tối ưu.....	5
1.2. Phương pháp chia để trị.....	5
1.3. Phương pháp Quy hoạch động.....	7
2. Một số bài tập áp dụng.....	8
2.1. Bài tập 1. Bài toán cái túi - version I (Mức độ cơ bản).....	8
2.1.1. Đề bài.....	8
2.1.2. Phân tích bài toán.....	9
2.1.3. Chương trình minh họa.....	11
2.1.4. Test.....	13
2.2. Bài tập 2. QBSEQ - Dãy con dài nhất có tổng chia hết cho K (Mức độ cơ bản)	13
2.2.1. Đề bài.....	13
2.2.2. Phân tích bài toán.....	13
2.2.3. Chương trình minh họa.....	16
2.2.4. Test.....	17
2.3. Bài tập 3. SEQDIV (Mức độ khá).....	17
2.3.1. Đề bài.....	17
2.3.2. Phân tích bài toán.....	18
2.3.3. Chương trình minh họa.....	18
2.3.4. Test.....	19
2.4. Bài tập 4. Dãy nhị phân (Mức độ khá).....	19
2.4.1. Đề bài.....	19
2.4.2. Phân tích bài toán	20
2.4.3. Chương trình minh họa.....	21
2.4.4. Test.....	22
2.5. Bài tập 5. Chọn đồ chơi (Mức độ khá).....	22
2.5.1. Đề bài.....	22
2.5.2. Phân tích bài toán.....	24
2.5.3. Chương trình minh họa.....	24
2.5.4. Test.....	25

2.6. Bài tập 6. Bầu cử (Mức độ khó)	26
2.6.1. Đề bài.....	26
2.6.2. Phân tích bài toán	27
2.6.3. Chương trình minh họa.....	27
2.6.4. Test.....	28
2.7. Bài tập 7. Đếm mảng (Mức độ khó).....	28
2.7.1. Đề bài.....	28
2.7.2. Phân tích bài toán	29
2.7.3. Chương trình minh họa.....	30
2.7.4. Test.....	31
2.8. Bài tập 8. Giá sách [BOOKSHELF.*] (Mức độ khó).....	31
2.8.1. Đề bài.....	31
2.8.2. Phân tích bài toán	32
2.8.3. Chương trình minh họa.....	34
2.8.4. Test.....	36
3. Cải tiến bài toán quy hoạch động bằng kỹ thuật chia để trị	36
3.1. Bài tập 9. Famous Pagoda (F - ACM ICPC Vietnam Regional 2017) (Mức độ khó)	37
3.1.1. Đề bài.....	37
3.1.2. Phân tích bài toán	38
3.1.3. Chương trình minh họa.....	40
3.1.4. Test.....	42
3.2. Bài tập 10. SEQPART (Mức độ khó).....	42
3.2.1. Đề bài.....	42
3.2.2. Phân tích bài toán	43
3.2.3. Chương trình minh họa.....	44
3.2.4. Test.....	46
4. Một số bài khác	46
5. Kết luận	46
6. Nguồn tài liệu tham khảo	46

QUY HOẠCH ĐỘNG (DYNAMIC PROGRAMMING)

0. Mở đầu

Trong các kỳ thi học sinh giỏi tin học như: Học sinh giỏi khu vực duyên hải; Học sinh giỏi quốc gia; Olympic tin học quốc tế...Thì các lớp bài toán về tối ưu hóa luôn được ưu tiên lựa chọn trong các đề thi, vì tính ứng dụng vào thực tiễn cao.

Quy hoạch động (Dynamic Programming) là một phương pháp rất hiệu quả để giải nhiều bài toán tin học, đặc biệt là những bài toán tối ưu, có một số bài toán sử dụng phương pháp quy hoạch động lại cho hiệu quả cao hơn hẳn so với nhiều phương pháp khác. Số lượng bài toán tin học được giải bằng phương pháp quy hoạch động cũng rất lớn. Số lượng các bài thi có thể áp dụng phương pháp quy hoạch động để giải trong đề thi học sinh giỏi môn Tin học thường có điểm rất cao. Vậy “*Có phải tất cả các bài toán tối ưu đều có thể áp dụng phương pháp quy hoạch động để giải?*; “*Làm thế nào để nhận dạng được bài toán đó có thể áp dụng phương pháp quy hoạch động để giải?*; “*Làm thế nào có thể giải một bài toán bằng phương pháp quy hoạch động?*”;

Có nhiều chiến lược để thiết kế giải thuật: Chia để trị (divide and conquer); Đệ quy quay lui (backtracking); Quy hoạch động (dynamic programming); Tham lam (greedy strategy), ... Mỗi chiến lược phù hợp cho một số dạng bài toán, mỗi chiến lược có ưu và nhược điểm riêng; Nhiệm vụ: cần quyết định chọn chiến lược phù hợp để giải quyết bài toán đó một cách hợp lí.

Chuyên đề này, tôi xin phép chia sẻ một số bài tập có thể áp dụng phương pháp quy hoạch động trong quá trình dạy đội tuyển HSG môn Tin học lớp 10 trong hè chuẩn bị lên lớp 11.

Nhân đây, tôi cũng xin gửi lời cảm ơn tới các thầy cô, các bạn đồng nghiệp đã giúp đỡ, cung cấp tài liệu và góp ý để tôi bổ sung, hoàn thiện chuyên đề. Hi vọng rằng, chuyên đề sẽ cung cấp cho các bạn đồng nghiệp và các em học sinh một phần kiến thức bổ ích.

1. Một số khái niệm, kiến thức cơ bản

1.1. Bài toán tối ưu

* Khái niệm

Bài toán tối ưu gồm có 1 hàm f gọi là hàm mục tiêu hay hàm đánh giá; các hàm g_1, g_2, \dots, g_n cho giá trị logic gọi là hàm ràng buộc. Yêu cầu của bài toán là tìm một cấu hình x thoả mãn tất cả các ràng buộc g_1, g_2, \dots, g_n : $g_i(x) = \text{TRUE}$ ($\forall i : 1 \leq i \leq n$) và x là tốt nhất, theo nghĩa không tồn tại một cấu hình y nào khác thoả mãn các hàm ràng buộc mà $f(y)$ tốt hơn $f(x)$.

Ví dụ 1: Trong mặt phẳng tọa độ Oxy tìm tọa độ (x,y) để tổng $x + y$ đạt giá trị lớn nhất mà $x^2 + y^2 \leq 1$.

Ở bài toán trên ta thấy:

Hàm mục tiêu: $x + y \rightarrow \max$

Hàm ràng buộc: $x^2 + y^2 \leq 1$.

Ví dụ 2: Bài toán xếp Ba lô Có một ba lô có thể chứa tối đa trọng lượng M và có n đồ vật ($n \leq 100$), mỗi đồ vật có trọng lượng w_i và giá trị b_i ; M, w_i, b_i là các số nguyên. Hãy chọn và xếp các đồ vật vào ba lô để tổng giá trị của ba lô là lớn nhất.

Với bài toán trên ta thấy:

Hàm mục tiêu: $\sum b_i \rightarrow \max, i=1,2, \dots, n$

Hàm ràng buộc: $\sum w_i \leq M, i=1,2, \dots, n$

Tóm lại, bài toán tối ưu rất phong phú, đa dạng, được ứng dụng nhiều trong thực tế nhưng chúng ta cũng cần biết rằng đa số các bài toán tối ưu là không giải được hoặc chưa giải được.

1.2. Phương pháp chia để trị

Ý tưởng :

+ Chia vấn đề cần giải thành một số vấn đề con cùng dạng với vấn đề đã cho, chỉ khác là cỡ của chúng nhỏ hơn.

+ Mỗi vấn đề con được giải quyết độc lập, nếu vấn đề con chưa tìm được nghiệm ngay thì lại tiếp tục chia nhỏ theo tư tưởng như trên.

+ Sau đó, ta kết hợp nghiệm của các vấn đề con để nhận được nghiệm của vấn đề đã cho.

Chia để trị là một trong những phương pháp hiệu để thiết kế thuật toán. Nguyên lý thực hiện của thuật toán chia để trị thực hiện qua hai bước sau:

- Bước 1 (chia): Chia bài toán lớn thành các bài toán con nhỏ hơn
- Bước 2 (trị): Gọi đệ quy giải các bài toán con, sau đó gộp lời giải của bài toán con thành lời giải bài toán lớn.

Đối với nhiều thuật toán đệ quy chúng ta đã tìm hiểu, nguyên lý chia để trị (devide and conquer) thường đóng vai trò chủ đạo trong việc thiết kế thuật toán.

Ví dụ: Bài toán Tính dãy Fibonacci

Ta có định nghĩa như sau: $F_n = \begin{cases} 1 & (n = 0 \text{ or } n = 1) \\ F(n-1) + F(n-2) & (n \geq 2) \end{cases}$

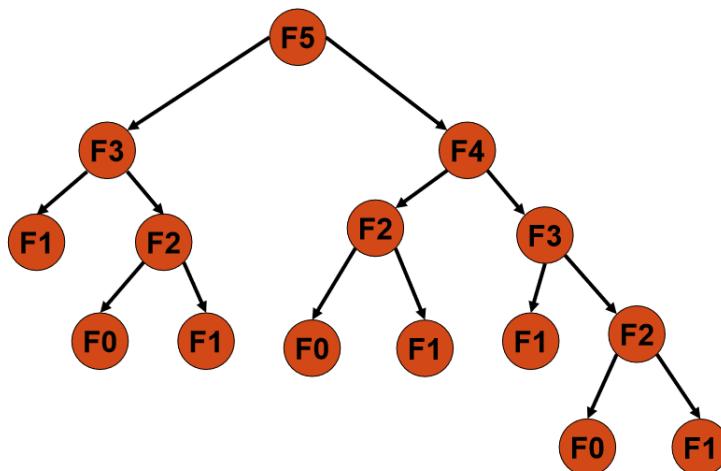
Một số phần tử đầu tiên của dãy Fibonacci: 0, 1, 1, 2, 3, 5, 8, ... Tìm số Fibonacci thứ n.

Dữ liệu vào: Một số nguyên dương duy nhất n ($n \leq 1000000$).

Kết quả ra: Một số nguyên duy nhất là số Fibonacci thứ n (kết quả lấy phần dư cho 1000000007).

Ví dụ:

FIBO.INP	FIBO.OUT
5	8
20	6765



1.3. Phương pháp Quy hoạch động

1.3.1. Bài toán quy hoạch động

Như chúng ta đã biết, một bài toán có thể giải được bằng phương pháp quy hoạch động cần đảm bảo 2 đặc điểm nổi bật sau:

- Có tính chất của các bài toán con gối nhau (overlapping subproblem): Có thể chia nhỏ một bài toán lớn thành các bài toán con.
- Có cấu trúc con tối ưu (optimal substructure): Kết hợp lời giải của các bài toán con ta được lời giải của bài toán lớn hơn.

1.3.2. Các khái niệm

- Bài toán giải theo phương pháp quy hoạch động gọi là bài toán quy hoạch động.
- Công thức phối hợp nghiệm của các bài toán con để có nghiệm của bài toán lớn gọi là công thức truy hồi của quy hoạch động.
- Tập các bài toán nhỏ nhất có ngay lời giải để từ đó giải quyết các bài toán lớn hơn gọi là cơ sở quy hoạch động.
- Không gian lưu trữ lời giải các bài toán con để tìm cách phối hợp chúng gọi là bảng phương án của quy hoạch động.

Phương pháp quy hoạch động (Dynamic Programming) là một kỹ thuật nhằm đơn giản hóa việc tính toán các công thức truy hồi bằng cách lưu toàn bộ hay một phần kết quả tính toán tại mỗi bước trước đó với mục đích sử dụng lại.

Như vậy, Quy hoạch động = Chia để trị + Mảng (lưu lại kết quả).

Phương pháp quy hoạch động do nhà toán học người Mỹ Richard Bellman (1920-1984) phát minh năm 1953. Phương pháp này dùng để giải các bài toán tối ưu có bản chất đệ qui, tức là tìm phương án tối ưu cho bài toán đó có thể đưa về tìm phương án tối ưu của một số hữu hạn các bài toán con.

Điểm khác nhau cơ bản giữa quy hoạch động và phương pháp đệ quy là :

+ Phương pháp đệ quy giải quyết bài toán theo hướng topdown, nghĩa là để giải bài toán ban đầu, ta phải đi giải tất cả các bài toán con của nó. Đây là một phương pháp hay, tuy nhiên phương pháp này sẽ gặp hạn chế về mặt thời gian, tốc độ do phải tính đi tính lại nhiều lần một số bài toán con giống nhau nào đó.

+ Phương pháp quy hoạch động sử dụng nguyên lý bottom-up, nghĩa là "đi từ dưới lên". Đầu tiên, ta sẽ phải giải các bài toán con đơn giản nhất, có thể tìm ngay ra nghiệm. Sau đó kết hợp các bài toán con này lại để tìm lời giải cho bài toán lớn hơn và cứ như thế cho đến khi giải được bài toán yêu cầu. Với phương pháp này, mỗi bài toán con sau khi giải xong đều được lưu trữ lại và đem ra sử dụng nếu cần. Do đó tiết kiệm bộ nhớ và cải thiện được tốc độ.

1.3.3. Qui trình giải toán bằng Quy hoạch động

Phương pháp này được thực hiện dựa trên những thao tác cơ bản sau:

Bước 1: Tìm nghiệm của các bài toán con nhỏ nhất

Bước 2: Tìm ra công thức (hoặc quy tắc) xây dựng nghiệm của bài toán con thông qua nghiệm của các bài toán con cỡ nhỏ hơn

Bước 3: Tạo ra một bảng lưu giữ các nghiệm của các bài toán con. Sau đó tính nghiệm của các bài toán con theo công thức đã tìm ra và lưu vào bảng

Bước 4: Từ các bài toán con đã giải để tìm ra nghiệm của bài toán.

2. Một số bài tập áp dụng

2.1. Bài tập 1. Bài toán cái túi - version I (Mức độ cơ bản)

2.1.1. Đề bài

Trong siêu thị có n đồ vật ($N \leq 1000$), đồ vật thứ i có trọng lượng là $W[i] \leq 1000$ và giá trị $V[i] \leq 1000$. Một tên trộm đột nhập vào siêu thị, tên trộm mang theo một cái túi có thể mang được tối đa trọng lượng M ($M \leq 1000$). Hỏi tên trộm sẽ lấy đi những đồ vật nào để được tổng giá trị lớn nhất.

Giải quyết bài toán trong trường hợp mỗi vật chỉ được chọn một lần.

Dữ liệu vào: File văn bản **Bag.inp**

- Dòng 1: N, M cách nhau ít nhất một dấu cách
- N dòng tiếp theo: Mỗi dòng gồm 2 số W_i, V_i , là chi phí và giá trị đồ vật thứ i.

Kết quả ra: File văn bản **Bag.out**: Ghi giá trị lớn nhất tên trộm có thể lấy.

Ví dụ:

Bag.inp	Bag.out
5 13	16 4
3 4	1
4 5	2
5 6	3
2 3	5
1 1	

2.1.2. Phân tích bài toán

➤ Tham số thể hiện kích thước bài toán

- Kết quả bài toán là tổng giá trị lớn nhất của các món hàng được chọn trong n món sao cho tổng khối lượng không lớn hơn M cho trước, ký hiệu là $F(n)$
- Tham số thể hiện kích thước bài toán là số món hàng n ;
- Giá trị của $F(n)$ có thể được tính từ giá trị của $F(n-1)$ cộng thêm hoặc không cộng thêm giá trị của món hàng thứ n nhưng tổng khối lượng không lớn hơn M ;
- Nếu chọn thêm món hàng thứ n thì tổng khối lượng được chọn trong $(n-1)$ món hàng không lớn hơn $(M - A[n])$;
- Suy ra bài toán có 2 tham số: số món hàng và khối lượng giới hạn.

➤ Lập công thức đệ qui:

- Gọi $F(i, v)$ là tổng giá trị lớn nhất của các gói hàng được chọn trong i gói hàng sao cho tổng khối lượng không lớn hơn v .
- Trường hợp $A[i] > v$: $F(i, v) = F(i-1, v)$;
- Trường hợp $A[i] \leq v$:
 - ✓ Nếu gói hàng thứ i không được chọn thì: $F(i, v) = F(i-1, v)$
 - ✓ Nếu gói hàng thứ i được chọn thì: $F(i, v) = F(i-1, v - A[i]) + C[i]$
 $\rightarrow F(i, v) = \text{Max}\{ F(i-1, v); F(i-1, v - A[i]) + C[i] \}$;
- Bài toán nhỏ nhất ứng với $i = 0$ ta có: $F(0, v) = 0$

➤ Xây dựng bảng phương án:

Cấu trúc bảng phương án: Dùng mảng $F[0..n][0..M]$ chứa giá trị của các $F(i, v)$;

Cách tính giá trị trên bảng phương án:

- Điền số 0 cho các ô trên dòng 0
- Sử dụng công thức đệ qui và giá trị trên dòng $(i-1)$ để tính dòng i :
 - ✓ Trường hợp $A[i] > v$: $F(i, v) = F(i-1, v)$

✓ Trường hợp $A[i] \leq v$: $F(i, v) = \text{Max}\{ F(i-1, v); F(i-1, v - A[i]) + C[i] \}$

Ví dụ:

C	A	v	0	1	2	3	4	5	6	7	8	9	10	11	12	13
		i	0	0	0	0	0	0	0	0	0	0	0	0	0	0
4	3	1	0	0	0	4	4	4	4	4	4	4	4	4	4	4
5	4	2	0	0	0	4	5	5	5	9	9	9	9	9	9	9
6	5	3	0	0	0	4	5	6	6	9	10	11	11	11	15	15
3	2	4	0	0	3	4	5	7	8	9	10	12	13	14	15	15
1	1	5	0	1	3	4	5	7	8	9	10	12	13	14	15	16

➤ Thuật toán tạo bảng phương án:

```
void taobang()
{
    for(int j=0;j<=m;j++) f[0][j]=0;
    for(int i=1;i<=n;i++)
        for(int j=0;j<=m;j++)
    {
        f[i][j]=f[i-1][j];
        if((v[i]<=j)&&(f[i][j]<f[i-1][j-v[i]]+w[i]))
            f[i][j]=f[i-1][j-v[i]]+w[i];
        kq=max(kq,f[i][j]);
    }
}
```

➤ Truy vết tìm lại các gói hàng đã chọn:

Bắt đầu từ ô $F[n, M]$ trên dòng n ta dò ngược về dòng 0 theo nguyên tắc:

- Nếu $F[i, v] > F[i-1, v]$ thì gói thứ i được chọn, ta truy tiếp ô $F[i-1, v-A[i]]$.
- Nếu $F[i, v] = F[i-1, v]$ thì gói thứ i không được chọn, ta truy tiếp ô $F[i-1, v]$.

Ví dụ:

C	A	v	0	1	2	3	4	5	6	7	8	9	10	11	12	13
		i	0	0	0	0	0	0	0	0	0	0	0	0	0	0
4	3	1	0	0	0	4	4	4	4	4	4	4	4	4	4	4
5	4	2	0	0	0	4	5	5	5	9	9	9	9	9	9	9
6	5	3	0	0	0	4	5	6	6	9	10	11	11	11	15	15
3	2	4	0	0	3	4	5	7	8	9	10	12	13	14	15	15

1	1	5	0	1	3	4	5	7	8	9	10	12	13	14	15	16
---	---	---	---	---	---	---	---	---	---	---	----	----	----	----	----	----

➤ Thuật toán truy vết tìm lại các gói hàng đã chọn:

```

void TruyVet(F[0..n][0..M])
{
    Bắt đầu từ ô F[n, M] trên dòng n: i = n; v = M;
    for (; i > 0; i --)
        if (F[i, v] != F[i-1, v])
    {
        <Món hàng thứ i được chọn>;
        v = v - A[i];
    }
}

```

2.1.3. Chương trình minh họa

```

#include <bits/stdc++.h>
using namespace std;
int n,m,v[10000],w[100000],f[1000][1000],d,kq;
bool kt[1000];
void nhap()
{
    cin>>n>>m;
    for(int i=1;i<=n;i++)
        cin>>v[i]>>w[i];
    d=kq=0;
}
void taobang()
{
    for(int j=0;j<=m;j++) f[0][j]=0;
    for(int i=1;i<=n;i++)
        for(int j=0;j<=m;j++)
    {

```

```

f[i][j]=f[i-1][j];
if((v[i]<=j)&&(f[i][j]<f[i-1][j-v[i]]+w[i]))
    f[i][j]=f[i-1][j-v[i]]+w[i];
    kq=max(kq,f[i][j]);
}
}

void truyvet()
{
    int a=m;
    for(int i=n;i>=1;i--)
        if(f[i][a]!=f[i-1][a])
    {
        d++;
        a=a-v[i];
        kt[i]=1;
    }
}
int main()
{
    freopen("bag.inp","r",stdin);
    freopen("bag.out","w",stdout);
    nhap();
    taobang();
    truyvet();
    cout<<kq<<" "<<d<<endl;
    for(int i=1;i<=n;i++)
        if(kt[i]==1) cout<<i<<endl;
    return 0;
}

```

2.1.4. Test

<https://drive.google.com/file/d/1PFLKadBMVP2q539P5pURGEkMAMBXoouG/view?usp=sharing>

2.2. Bài tập 2. QBSEQ - Dãy con dài nhất có tổng chia hết cho K (Mức độ cơ bản)

2.2.1. Đề bài

Nguồn bài: <https://codeforces.com/group/FLVn1Sc504/contest/274711/problem/L>

Cho một dãy gồm n ($n \leq 1000$) số nguyên dương A_1, A_2, \dots, A_n và số nguyên dương k ($k \leq 50$). Hãy tìm dãy con gồm nhiều phần tử nhất của dãy đã cho sao cho tổng các phần tử của dãy con này chia hết cho k.

Dữ liệu vào: Dòng đầu tiên chứa hai số n, k ghi cách nhau bởi ít nhất 1 dấu trống.

Các dòng tiếp theo chứa các số A_1, A_2, \dots, A_n được ghi theo đúng thứ tự cách nhau ít nhất một dấu trống hoặc xuống dòng

Kết quả ra: Gồm 1 dòng duy nhất ghi số lượng phần tử của dãy con dài nhất thỏa mãn.

Ví dụ:

Input	Output
6 5 11 6 7 12 20 8	4
10 3 2 3 5 7 9 6 12 7 11 15	9

2.2.2. Phân tích bài toán

Nhắc lại phép toán mod:

Giả sử: $r = a \bmod k$ và $z = b \bmod k$

Ta có:

1. $(a + b) \bmod k = (r + z) \bmod k$
2. $(-r) \bmod k = (-r + k) \bmod k$
3. $(r + z) \bmod k = v \rightarrow z = (v - r) \bmod k = (v - r + k) \bmod k$

i	1	2	3	4	5	6
A[i]	11	6	7	12	20	8
A[i] % 5	1	1	2	2	0	3
	0	0	0	3	4	4

Xác định tham số:

Gọi $F(i)$ = chiều dài dãy con dài nhất trong miền $[1..i]$ có tổng chia hết cho k .

1. Nếu dãy con dài nhất không có $A[i]$ thì $F(i) = F(i-1)$ Với $F(i-1)$ = chiều dài dãy con dài nhất trong miền $[1..i-1]$ có tổng chia hết cho k ;

2. Nếu dãy con dài nhất có chứa $A[i]$: thì $F(i) = F(i-1) + 1$;

Gọi $r = A[i] \bmod k$

- Nếu $r = 0$: thì $F(i-1)$ = chiều dài dãy con dài nhất trong miền $[1..i-1]$ có tổng chia hết cho k ;

- Nếu $r > 0$: do $(r + k - r) \bmod k = 0$ nên $F(i-1)$ bằng chiều dài dãy con dài nhất trong miền $[1..(i-1)]$ có tổng chia với k dư $(k-r)$;

Tham số thể hiện kích thước của bài toán: kích thước miền và số dư của tổng chia với k .

Lập công thức đệ quy:

Gọi $F(i, v)$ = chiều dài dãy con dài nhất trong miền $[1..i]$ có tổng chia với k dư là v .

Nếu dãy con dài nhất không có $A[i]$ thì $F(i, v) = F(i-1, v)$

Nếu dãy con dài nhất có chứa $A[i]$:

Gọi $r = A[i] \bmod k$ ta có: $F(i, v) = F(i-1, v - r) + 1$

- Nếu $v - r = 0$: $F(i, v) = F(i-1, 0) + 1$

- Nếu $v - r > 0$ và $F(i-1, v - r) > 0$: $F(i, v) = F(i-1, v - r) + 1$

- Nếu $v - r < 0$ và $F(i-1, v - r + k) > 0$: $F(i, v) = F(i-1, v - r + k) + 1$ thay $(v - r) \bmod k = (v - r + k) \bmod k$

• Bài toán nhỏ nhất ứng với $i = 1$:

$F(1, v) = 0$ nếu $r <> v$

$F(1, v) = 1$ nếu $r = v$

Công thức đệ quy:

Gọi $r = A[i] \bmod k$

• Với $i = 1$:

- $F(1, v) = 0$ nếu $r <> v$

- $F(1, v) = 1$ nếu $r = v$

• Với $i > 1$:

- Nếu $v = r$ thì : $F(i, v) = \text{Max}\{F(i-1, v), F(i-1, 0) + 1\}$

- Nếu $v > r$ và $F(i-1, v-r) > 0$ thì: $F(i, v) = \max\{F(i-1, v), F(i-1, v-r) + 1\}$
- Nếu $v < r$ và $F(i-1, v-r+k) > 0$ thì: $F(i, v) = \max\{F(i-1, v), F(i-1, v-r+k) + 1\}$

Tinh chế công thức đệ quy:

Gọi $r = A[i] \bmod k$

- Với $i = 1$:

- $F(1, v) = 0$ nếu $r <> v$
- $F(1, v) = 1$ nếu $r = v$

- Với $i > 1$:

- Nếu $v = r$ thì: $F(i, v) = \max\{F(i-1, v), F(i-1, 0) + 1\}$
- Nếu $v <> r$ và $F(i-1, (v-r+k) \bmod k) > 0$ thì $F(i, v) = \max\{F(i-1, v), F(i-1, (v-r+k) \bmod k) + 1\}$

Xây dựng bảng phương án:

Cấu trúc bảng phương án: Dùng mảng $F[1..n][0..K-1]$ chứa giá trị của các $F(i, v)$

Cách tính giá trị trên bảng phương án:

- Điền giá trị dòng 1: Nếu $A[1] \bmod k = v$ thì $F[1, v] = 1$ ngược lại $F[1, v] = 0$
- Sử dụng công thức đệ quy và giá trị trên dòng $i - 1$ để tính dòng i

Ví dụ:

Với $i > 1$:

- Nếu $v = r$ thì: $F(i, v) = \max\{F(i-1, v), F(i-1, 0) + 1\}$
- Nếu $v <> r$ và $F(i-1, (v-r+k) \bmod k) > 0$ thì $F(i, v) = \max\{F(i-1, v), F(i-1, (v-r+k) \bmod k) + 1\}$

Thuật toán tạo bảng phương án:

```

void TaoBangPhuongAn (F[1..n][0..k-1])
{
    for (v=0; v <= k-1; v++)
        F[1, v] = (A[i] % k == v) ? 1 : 0;
    for (i = 2; i <= n; i++)
        for (v=0; v <= k-1; v++)
            {
                r = A[i] % k;
                F[i, v] = F[i-1, v];
                if (v == r && F[i, v] <= F[i-1, 0])
                    F[i, v] = F[i-1, 0] + 1;
                else if (F[i-1, (v - r + k) % k] > 0 && F[i, v] <= F[i-1, (v - r + k) % k])

```

```

F[i, v] = F[ i -1, (v - r + k)%k] + 1;
}
}

```

Truy vết:

Bắt đầu từ ô $F[n, 0]$ trên dòng n ta dò ngược về dòng 0 theo nguyên tắc:

- Nếu $F[i-1, (v-r+k)\%k] > 0$ và $F[i, v] > F[i-1, (v-r+k)\%k]$:
 - $A[i]$ được chọn
 - Truy tiếp ô $F[i-1, (v-r+k)\%k]$.
- Ngược lại thì $A[i]$ không được chọn, ta truy tiếp ô $F[i-1, v]$.

2.2.3. Chương trình minh họa

```

#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
typedef pair<int,int> ii;
typedef unsigned long long ull;
#define X first
#define Y second
#define pb push_back
#define mp make_pair
#define ep emplace_back
#define EL printf("\n")
#define sz(A) (int) A.size()
#define FOR(i,l,r) for (int i=l;i<=r;i++)
#define FOD(i,r,l) for (int i=r;i>=l;i--)
#define fillchar(a,x) memset(a, x, sizeof (a))
#define faster ios_base::sync_with_stdio(false); cin.tie(NULL); cout.tie(NULL);
const int N = 1500;
const int inf = 1e9+7;
int n, k, a[N], F[N][55];
int main() {
    scanf("%d%d", &n,&k);
    FOR(i,1,n) scanf("%d", &a[i]), a[i] %= k;
    FOR(i,1,k-1) F[1][i] = -inf;
    F[1][a[1]] = 1;
}

```

```

FOR(i,2,n)
FOR(j,0,k-1)
F[i][j] = max(F[i-1][j], F[i-1][(j-a[i]+k)%k] + 1);
printf("%d\n", F[n][0]);
return 0;
}

```

2.2.4. Test

Test trực tiếp trên link <https://vn.spoj.com/problems/QBSEQ/>

Hoặc: <https://codeforces.com/group/FLVn1Sc504/contest/274711/problem/L>

2.3. Bài tập 3. SEQDIV (Mức độ khá)

2.3.1. Đề bài

Cho n và k. Hãy đếm số lượng dãy a_1, a_2, \dots, a_n sao cho:

$$\begin{cases} a_i \in [1; k] & i = 1..n \\ \sum a_i \leq k \\ \sum a_{i+1} \leq k \end{cases}$$

Dữ liệu vào: Vào từ file văn bản **SEQDIV.INP** một dòng duy nhất chứa hai số nguyên dương n và k.

Kết quả ra: Ghi ra file văn bản **SEQDIV.OUT** một số nguyên duy nhất là số lượng dãy thỏa mãn theo modulo 10^9+7 .

Ví dụ:

SEQDIV.INP	SEQDIV.OUT	Giải thích
4 3	8	(1;2;1;2), (1;2;1;3), (1;3;1;2), (1;3;1;3) (2;1;2;1), (2;1;3;1), (3;1;2;1), (3;1;3;1)
8 1	0	

Ràng buộc:

- 20% số test có $n, k \leq 8$
- 30% số test khác có $n, k \leq 100$
- 50% số test còn lại có $n \leq 100, k \leq 20000$

2.3.2. Phân tích bài toán

Ta sử dụng phương pháp quy hoạch động: $F[i][j]$ là số dãy thỏa mãn có độ dài i , và phần tử thứ i có giá trị là j , công thức :

- $F[i][j] = \sum F[i][j'], \forall j' : j' \text{ hoặc } j' < j$
- Độ phức tạp : $O(n.k.k)$

Cải tiến : Ta tạo 1 vector 2 chiều, với mỗi vector $A[i]$, ta lưu giá trị các bội và ước của số có giá trị i , để tạo $A[i]$ ta sử dụng cách sau :

```
for (int i = 1; i <= k; i++)
    for (int j = i + i; j <= k; j += i)
    {
        A[i].push_back(j);
        A[j].push_back(i);
    }
```

Với mỗi i , thì số lượng bội và ước của nó là rất ít nên độ phức tạp chỉ khoảng $1e6$.

2.3.3. Chương trình minh họa

```
#include <bits/stdc++.h>
#define maxn 102
#define maxk 20009
using namespace std;
const long long MOD = 1000000007;
long long n, k, kq, F[maxn][maxk];
vector<vector<int>> A(maxk);
int main()
{
    ios_base::sync_with_stdio(0); cin.tie(0); cout.tie(0);
    freopen("SEQDIV.inp", "r", stdin);
    freopen("SEQDIV.out", "w", stdout);

    cin >> n >> k;
    for (int i = 1; i <= k; i++)
        for (int j = i + i; j <= k; j += i)
        {
            A[i].push_back(j);
            A[j].push_back(i);
        }
}
```

```

        A[j].push_back(i);
    }
    for (int i = 1; i <= k; i++) F[1][i] = 1;
    for (int i = 2; i <= n; i++)
        for (int j = 1; j <= k; j++)
            for (int v = 0; v < A[j].size(); v++)
            {
                int cmp = A[j][v];
                F[i][j] = (F[i - 1][cmp] + F[i][j]) % MOD;
            }
        for (int i = 1; i <= k; i++) kq = (kq + F[n][i]) % MOD;
        cout<<kq;
    }
}

```

2.3.4. Test

https://drive.google.com/file/d/1MpDMopHWQnCS5qWXoVIIcgV94aE_2FUX/view?usp=sharing

2.4. Bài tập 4. Dãy nhị phân (Mức độ khá)

2.4.1. Đề bài

Nam là một học sinh thông minh, cần cù chịu khó. Vì muốn các bạn trong cùng đội tuyển cùng nhớ lại kiến thức về dãy nhị phân, Nam đã đưa ra một đề bài như sau: *Cho trước số nguyên dương n ($n \leq 50$). Hãy đếm số lượng xâu nhị phân có độ dài n mà trong xâu đó không có 2 ký tự '1' nào đứng cạnh nhau.* Các em hãy cùng giải đề mà Nam đưa ra nhé!

Dữ liệu vào: File DNP.INP chứa một số nguyên duy nhất.

Kết quả ra: File DNP.OUT chứa số nguyên duy nhất là số lượng xâu nhị phân có độ dài n mà trong xâu đó không có 2 ký tự '1' nào đứng cạnh nhau.

Ví dụ:

DNP.INP	DNP.OUT
5	13

Giải thích: 13 xâu nhị phân độ dài 5 thỏa mãn điều kiện đề bài là:

00000 00001 00010 00100 00101 01000 01001
01010 10000 10001 10010 10100 10101

Giới hạn:

Subtask 1 (60%): $n \leq 20$

Subtask 2 (40%): $20 < n \leq 50$

2.4.2. Phân tích bài toán

Giải thuật 1:

1.1. Độ phức tạp cấp số nhân.

Tạo ra tất cả các chuỗi nhị phân có độ dài n , chỉ in những chuỗi thỏa mãn điều kiện đã cho.

1.2: Độ phức tạp hàm mũ. Ý tưởng là sử dụng đệ quy.

Ta chỉ tạo ra các chuỗi N chữ số thỏa mãn các ràng buộc đã cho. Tại mỗi điểm trong đệ quy, nối 0 và 1 vào số được hình thành một phần và lặp lại với một chữ số ít hơn. Mẹo ở đây là ta nối thêm 1 và chỉ lặp lại nếu chữ số cuối của số được tạo thành một phần là 0. Bằng cách đó, chúng ta sẽ không bao giờ có bất kỳ hai số 1 liên tiếp nào trong chuỗi đầu ra.

Giải thuật 2: Độ phức tạp là $O(n)$

Nếu vẽ cây đệ quy của Sub 2, chúng ta có thể thấy các bài toán con tương tự đang được tính toán lặp đi lặp lại. Các bài toán con có cấu trúc tối ưu, chồng chéo nhau nên có thể được giải quyết bằng cách sử dụng quy hoạch động.

```
T[0][0] = 0, T[0][1] = 0;      // Không có kí tự nào
T[1][0] = 2, T[1][1] = 1;      // Nếu chỉ có duy nhất một kí tự
for (int i = 2; i <= n; i++)
{
    T[i][0] = T[i-1][0] + T[i-1][1];
    T[i][1] = T[i-1][0];
}
```

2.4.3. Chương trình minh họa

```
#include <bits/stdc++.h>
#include <string>
using namespace std;
int n;
/*SUB1
int count1(int n, int last)
{
    // trường hợp n = 0
    if (n == 0)
        return 0;
    // nếu xâu chỉ gồm 1 kí tự
    if (n == 1)
    {
        if (last)      // nếu kí tự cuối là 1
            return 1;
        else          // nếu kí tự cuối là 0
            return 2;
    }
    // nếu kí tự cuối là 0 thì ta có thể nối 0 vào các xâu có độ dài n-1 trước đó
    // kết thúc bởi 0 hoặc 1
    if (last == 0)
        return count1(n - 1, 0) + count1(n - 1, 1);
    // nếu kí tự cuối là 1 thì ta chỉ có thể nối 1 vào các xâu có độ dài n-1
    // trước đó kết thúc bởi 0
    else
        return count1(n - 1, 0);
}
*/
long long count2(int n)
{
    long long T[n+1][2]; // T[i][j] là số lượng xâu nhị phân độ dài i thỏa
    mãn, kết thúc bởi kí tự j
    T[0][0] = 0, T[0][1] = 0; // Không có kí tự nào
    T[1][0] = 2, T[1][1] = 1; // Nếu chỉ có duy nhất một kí tự
```

```

for (int i = 2; i <= n; i++)
{
    // nếu kí tự cuối là 0 thì ta có thể nối 0 vào các xâu có độ dài n-1 trước
    // đó kết thúc bởi 0 hoặc 1
    T[i][0] = T[i-1][0] + T[i-1][1];
    // nếu kí tự cuối là 1 thì ta chỉ có thể nối 1 vào các xâu có độ dài n-1
    // trước đó kết thúc bởi 0
    T[i][1] = T[i-1][0];
}
return T[n][0];
}

int main()
{
    freopen("DNP.INP","r",stdin);
    freopen("DNP.OUT","w",stdout);
    cin >> n;
    // cout << count1(n,0) << endl;
    cout << count2(n) << endl;
    return 0;
}

```

2.4.4. Test

<https://drive.google.com/file/d/1-H89godFO5L1bsRuWXj28DuY8KNyx0sz/view?usp=sharing>

2.5. Bài tập 5. Chọn đồ chơi (Mức độ khá)

2.5.1. Đề bài

Bạn Bi rất yêu thích việc mua sắm những đồ chơi khoa học viễn tưởng hiếm và đắt. Cậu ấy giữ chúng theo một dãy theo thứ tự ngày mua và để trong một cái tủ. Vì vậy, bạn Bo sẽ không bao giờ lấy được đồ chơi của cậu ấy. Nhưng vì một lần không may mắn, Bi đã thua Bo trong một vụ cá cược. Và Bo đã yêu cầu Bi chia sẻ đồ chơi. Bởi vì Bi không muốn mất nhiều tiền nên cậu ấy đã quyết định dựa trên một chiến lược để giảm thiểu sự mất mát xuống thấp nhất.

Bi bắt đầu chọn từ đồ chơi đầu tiên ở trong tủ, sẽ lấy một số đồ chơi, gọi là X đồ chơi. Bo sau đó sẽ chọn X đồ chơi (chú ý là Bo sẽ chọn số đồ chơi bằng với Bi, trừ

khi số đồ chơi còn lại nhỏ hơn X). Việc này sẽ tiếp tục cho đến khi không còn lại đồ chơi nào nữa.

Bạn được đưa cho một dãy của đồ chơi với giá của chúng. Hãy đưa ra số tiền lớn nhất Bi có thể giữ lại ứng với số đồ chơi mà Bi chọn. Bi chỉ có thể chọn 1, 2 hay 3 đồ chơi (X có giá trị 1, 2 hay 3)

Dữ liệu vào: Từ tệp TOYS.INP có cấu trúc như sau:

- + Dòng đầu tiên ghi số nguyên dương N là số đồ chơi.
- + N dòng tiếp theo ghi N số nguyên dương là giá tiền của các đồ chơi.

Các số trên một dòng ghi cách nhau bởi một dấu cách.

Kết quả ra: Ghi ra tệp TOYS.OUT một số nguyên duy nhất là tổng số tiền lớn nhất của các đồ chơi mà bạn Bi chọn

Ràng buộc:

$$1 \leq T \leq 10$$

$$1 \leq N \leq 100000$$

$$1 \leq \text{Giá của đồ chơi} \leq 1000000$$

Ví dụ:

TOYS.INP	TOYS.OUT
4 5 4 3 2	12
6 10 8 7 11 15 20	53

Giải thích:

- Ở test case 1: Bi chọn 3 đồ chơi trong lần đầu tiên của anh ta là: 5, 4, 3. Do đó, Bo không còn lựa chọn nào khác nên phải chọn 2. Khi đó, Bi thu được tổng số tiền lớn nhất là $5 + 4 + 3 = 12$

- Ở test case 2: Bi chọn 10 và 8. Sau đó, Bo chọn 7 và 11. Cuối cùng Bi chọn 15 và 20. Do đó, số tiền lớn nhất mà Bi thu được là: $10 + 8 + 15 + 20 = 53$

Subtask 1 (30%) : $1 \leq N \leq 10^2$, $1 \leq \text{giá tiền các đồ chơi} \leq 10^3$

Subtask 2 (30%) : $10^2 < N \leq 10^3$, $10^3 < \text{giá tiền các đồ chơi} \leq 10^5$

Subtask 3 (40%) : $10^3 < N \leq 10^5$, $10^5 < \text{giá tiền các đồ chơi} \leq 10^7$

2.5.2. Phân tích bài toán

Sử dụng quy hoạch động.

Lưu ý:

- Bi sẽ có 3 cách lựa chọn là: chọn 1, 2 hoặc 3 vật.
- Kết quả có thể rất lớn nên phải sử dụng kiểu **long long** (64 bit) để lưu kết quả

// Nếu chọn 1 đồ chơi

MaxMoney[i] = (long long)Price[i] + MaxMoney[i + 2];

// Nếu chọn 2 đồ chơi

long long t = (long long)(Price[i] + Price[i+1] + MaxMoney[i+4]

if (t > MaxMoney[i]) MaxMoney[i] = t;

// Nếu chọn 3 đồ chơi

long long t = (long long)(Price[i] + Price[i+1] + Price[i+2] + MaxMoney[i+6]

if (t > MaxMoney[i]) MaxMoney[i] = t;

2.5.3. Chương trình minh họa

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

const int MAX = 1000010;

int N; // Số đồ chơi
int Price[MAX]; // Giá của các đồ chơi
long long MaxMoney[MAX]; // MaxMoney[i] là số tiền lớn nhất
// khi Leonard chọn từ đồ chơi thứ i.
int main(int argc, char** argv)
{
    freopen("TOYS.INP", "r", stdin);
    freopen("TOYS.OUT", "w", stdout);
    int T, test_case;
    ios::sync_with_stdio(false);
```

```

//freopen("input.txt", "r", stdin);
// Nhập đầu vào
cin >> N;
for(int i = 1; i <= N; i++)
{
    cin >> Price[i];
    MaxMoney[i] = 0;
}
// Trường hợp cơ sở
MaxMoney[N] = Price[N];
for(int i = N + 1; i <= N + 5; i++)
    Price[i] = MaxMoney[i] = 0;

for(int i = N-1; i >= 1; i--)
{
    // Nếu chọn 1 đồ chơi
    MaxMoney[i] = (long long)Price[i] + MaxMoney[i + 2];

    // Nếu chọn 2 đồ chơi
    long long t = (long long)(Price[i] + Price[i + 1]) +
    MaxMoney[i + 4];
    if(t > MaxMoney[i]) MaxMoney[i] = t;

    // Nếu chọn 3 đồ chơi
    t = (long long)(Price[i] + Price[i + 1] + Price[i + 2]) +
    MaxMoney[i + 6];
    if(t > MaxMoney[i]) MaxMoney[i] = t;
}
cout << MaxMoney[1] << endl;

return 0;
}

```

2.5.4. Test

https://drive.google.com/file/d/1nK7Vk1mp75dkAO_xyvXNukjCg4o-AK8R/view?usp=sharing

2.6. Bài tập 6. Bầu cử (Mức độ khó)

2.6.1. Đề bài

Người dân Byteland đã đi bầu cử quốc hội. Khi kết quả được công bố, các đảng phái lên kế hoạch liên minh để thành lập chính phủ.

Mỗi chính đảng có một số ghế nhất định trong quốc hội. Một liên minh bao gồm một số chính đảng được quyền thành lập chính phủ nếu có tổng số ghế của các đảng trong liên minh lớn hơn nữa số ghế quốc hội.

Một liên minh được gọi là dư nếu có thể loại bỏ một đảng nào đó ra khỏi liên minh mà số ghế còn lại vẫn quá bán. Việc loại bỏ như vậy cho phép các thành viên còn lại thông qua luật không phụ thuộc vào ý kiến của các đảng ngoài liên minh.

Các đảng được đánh số từ 1 đến n ($1 \leq n \leq 1000$). Đảng thứ i giành được a_i ghế. Tổng số lượng ghế trong quốc hội không quá 100 000.

Yêu cầu: Cho biết n, a_1, a_2, \dots, a_n . Hãy xác định một liên minh quá bán không dư có tổng số ghế lớn nhất, chỉ ra số lượng đảng tham gia liên minh và số thứ tự của các đảng thuộc liên minh.

Dữ liệu: Vào từ file văn bản ELECTION.INP:

- Dòng đầu tiên chứa số nguyên n ,
- Dòng thứ 2 chứa n số nguyên a_1, a_2, \dots, a_n .

Kết quả: Đưa ra file văn bản ELECTION.OUT:

- Dòng đầu tiên chứa số nguyên p – số lượng đảng tham gia liên minh,
- Dòng thứ 2 chứa p số nguyên xác định các đảng thuộc liên minh.

Ví dụ:

ELECTION.INP	ELECTION.OUT
4	2
1 3 2 4	2 4

Ràng buộc:

- 40% test $n \leq 20$
- 30% test $20 < n \leq 100$
- 30% test $100 < n \leq 1000$

2.6.2. Phân tích bài toán

- Nhận thấy ta cần tìm một tập B là tập con của $A = \{A_1, \dots, A_n\}$ sao cho tổng S của các phần tử thuộc B thỏa mãn:

$$\begin{aligned} - S &> \frac{\text{sum}}{2} \\ - S - \min B_i &\leq \frac{\text{sum}}{2} \end{aligned}$$

- Từ đó ta có ý tưởng sau:

- Sắp xếp các phần tử giảm dần.
 - Ta thực hiện thuật toán quy hoạch động sinh tổng với các phần tử lần lượt được xét theo thứ tự giảm dần. Khi đó, số đang được xét (A_i) là số nhỏ nhất. Ta có tổng S được xét nếu ta tạo ra tổng S từ tổng $S - A_i \leq \frac{\text{sum}}{2}$. Tối ưu, lựa chọn giá trị S lớn nhất.

2.6.3. Chương trình minh họa

```
#include <bits/stdc++.h>
#define sz(x) int(x.size())
#define PB push_back
#define mp make_pair
#define F first
#define S second
#define Task "election"
#define maxn 100005
#define MOD 1000000007
#define remain(x) if (x > MOD) x -= MOD
#define pii pair<int, int>
using namespace std;
int n, s = 0, tr[maxn], res;
pii a[1001];
int main()
{
    ios_base::sync_with_stdio(0);
    freopen(Task".inp", "r", stdin);
    freopen(Task".out", "w", stdout);
    cin >> n;
    for (int i = 1; i <= n; i++) cin >> a[i].F, a[i].S = i, s+= a[i].F;
```

```

sort(a+1, a+n+1, greater<pii>());
tr[0] = -1;
int ns = s/2;
for (int i = 1; i <= n; i++)
{
    for (int j = s; j >= a[i].F; j--)
        if (tr[j] == 0 && tr[j-a[i].F] != 0)
    {
        tr[j] = i;
        if (j > ns && j-a[i].F <= ns) res = max(res, j);
    }
}
vector <int> ans;
while (res)
{
    ans.PB(a[tr[res]].S);
    res -= a[tr[res]].F;
}
sort(ans.begin(), ans.end());
cout << ans.size() << endl;
for (int i = 0; i < ans.size(); i++) cout << ans[i] << " ";
return 0;
}

```

2.6.4. Test

<https://drive.google.com/file/d/1KjZ2YYxUq40xrAveNDe4Qs2usYAV33EA/view?usp=sharing>

2.7. Bài tập 7. Đếm mảng (Mức độ khó)

2.7.1. Đề bài

Bài toán đơn giản: Hãy đếm xem có bao nhiêu mảng khác nhau a_1, a_2, \dots, a_n trong đó a_i nhận các giá trị nguyên dương trong đoạn $[1, M]$ sao cho tồn tại ít nhất một đoạn K giá trị liên tiếp giống nhau?

Ở đây hai mảng được gọi là khác nhau nếu như tồn tại ít nhất một vị trí mà giá trị phần tử hai mảng ở vị trí này là khác nhau.

Dữ liệu vào: Vào từ file văn bản CARRAYS.INP một dòng duy nhất chứa ba số nguyên dương n, M, K ($1 \leq n, M, K \leq 10^6$)

Kết quả ra: Ghi ra file văn bản CARRAYS.OUT một số nguyên duy nhất là số lượng mảng khác nhau tìm được. Con số này có thể rất lớn nên bạn chỉ cần lấy phần dư của nó khi chia cho $10^9 + 7$

Ví dụ:

CARRARS.INP	CARRAYS.OUT
3 2 2	6

Giải thích:

Các mảng tìm được là $(1, 1, 1), (1, 1, 2), (1, 2, 2), (2, 1, 1), (2, 2, 1), (2, 2, 2)$

Subtasks:

- Subtask 1: $1 \leq n, M, K \leq 10^3$ [50%]
- Subtask 2: $1 \leq n, M, K \leq 10^6$ [50%]

2.7.2. Phân tích bài toán

Thay vì đếm số lượng dãy có ít nhất một đoạn K giá trị liên tiếp giống nhau ta đếm số lượng dãy mà trong đó không có K giá trị liên tiếp nào giống nhau.

Đặt $dp[n]$ là số lượng dãy không có K giá trị liên tiếp giống nhau ta có công thức qui hoạch động:

- +) $dp[n] = M^n$ nếu như $n < K$
- +) $dp[n] = (M - 1) \cdot \sum_{c=1}^{K-1} dp[n - c]$ nếu $n \geq K$

Độ phức tạp thuật toán là $O(NK)$. Đáp số của bài toán là:

$$M^n - dp[n]$$

Để có thuật toán tốt hơn ta đặt $s[n] = dp[1] + dp[2] + \dots + dp[n]$ Khi đó

$$\begin{aligned} s[n] - s[n - 1] &= dp[n] \\ &= (M - 1) \cdot \sum_{c=1}^{K-1} dp[n - c] = (M - 1) \cdot (s[n - 1] - s[n - K]) \end{aligned}$$

Vậy nên:

$$s[n] = M \cdot s[n-1] - (M-1) \cdot s[n-K]$$

Do vậy có thể tính mảng $s[...]$ trong thời gian $O(n)$ và $dp[n]=s[n]-s[n-1]$

2.7.3. Chương trình minh họa

```
#include <bits/stdc++.h>
#define maxn 1000001
#define mod 1000000007
using namespace std;
int n, M, K;
int dp[maxn], s[maxn];
int LuyThua(int a,int n) {
    if (n==0) return 1;
    int T=LuyThua(a,n/2);
    T = (int64_t(T)*T) % mod;
    if (n%2) T=(int64_t(T)*a) % mod;
    return T;
}
int main() {
    freopen("carrays.inp","r",stdin);
    freopen("carrays.out","w",stdout);
    scanf("%d %d %d", &n, &M, &K);
    s[0]=0; dp[0]=0;
    for(int i=1;i<K;++i) {
        dp[i]=LuyThua(M,i);
        s[i]=(s[i-1]+dp[i]) % mod;
    }
    for(int i=K;i<=n;++i) {
        s[i]=(int64_t(M)*s[i-1]-int64_t(M-1)*s[i-K]+int64_t(mod)*mod) % mod;
        dp[i]=((s[i]-s[i-1]) % mod + mod) % mod;
    }
    int ans=((LuyThua(M,n)-dp[n]) % mod + mod) % mod;
    printf("%d", ans);
}
```

2.7.4. Test

<https://drive.google.com/file/d/1Q7jrI5Pn76194Iw3Rp5dg2mDf2mzfLOt/view?usp=sharing>

2.8. Bài tập 8. Giá sách [BOOKSHELF.*] (Mức độ khó)

2.8.1. Đề bài

An là một học sinh giỏi tin của trường và cậu rất yêu sách. Sau mỗi cuộc thi Hùng Vương, Duyên Hải bộ sưu tập sách của cậu lại dày thêm, An đã thu thập được N ($1 \leq N \leq 10^5$) quyển sách và An quyết định đóng một cái giá sách để đựng chúng. Quyển sách thứ i có chiều cao là H_i và chiều dày là W_i . Các quyển sách sẽ lần lượt thêm vào các tầng của giá sách theo đúng thứ tự đọc.

Ví dụ: tầng 1 chứa các quyển sách theo thứ tự $1..k$ thì tầng 2 bắt đầu chứa các quyển sách từ số hiệu $k+1$... Tất cả tầng của của giá sách có chiều rộng không quá L ($1 \leq L \leq 10^9$) tức là chỉ xếp được số sách có tổng chiều dày nhỏ hơn hoặc bằng L . Chiều cao của một tầng được đóng là chiều cao của quyển sách cao nhất trong tầng đó. Và chiều cao của giá sách là tổng chiều cao của tất cả các tầng.

Yêu cầu: Hãy giúp An xác định xem chiều cao nhỏ nhất của giá sách là bao nhiêu?

Input: Vào từ file văn bản **BOOKSHEFT.INP** gồm:

- Dòng đầu tiên chứa hai số nguyên N, L
- N dòng tiếp theo, dòng thứ i chứa 2 số nguyên H_i, W_i ($1 \leq H_i \leq 10^6$, $1 \leq W_i \leq L$)

Output: Ghi ra file văn bản **BOOKSHEFT.OUT** một số nguyên duy nhất là kết quả tìm được.

Ví dụ:

BOOKSHEFT.INP	BOOKSHEFT.OUT
5 10	21
5 7	
9 2	
8 5	
13 2	
3 8	

2.8.2. Phân tích bài toán

Ta sử dụng phương pháp quy hoạch động: Đặt $dp[k]$ là tổng chiều cao nhỏ nhất của giá sách khi xếp k quyển sách. Ta có công thức quy hoạch động:

$$dp[k] = \min dp[l] + \max(H[l+1], \dots, H[k])$$

Ở đây $l < k$ thỏa mãn $W[l+1] + \dots + W[k] = sW[k] - sW[l-1] \leq L$

Ở đây $sW[k] = W[1] + \dots + W[k]$

Code bình thường ta có độ phức tạp $O(n^2)$. Ta cần một giải pháp tốt hơn. Để có điều này ta có hai nhận xét quan trọng:

+) $dp[k]$ là hàm không giảm theo k

+) Với k cố định thì $a[l] = \max(H[l+1], \dots, H[k])$ không tăng theo l ;

Nếu $H[i] = \max(H[l+1], \dots, H[k])$ thì ta có một cặp (l, i) với giá trị $dp[l] + H[i]$. Ta duy trì một hàng đợi các phần tử dạng (l, i) trong danh sách công thức quy hoạch động. Nhận xét rằng khi xuất hiện thêm phần tử thứ k :

- Tất cả các cặp (l, i) với $H[i] \leq H[k]$ bị bỏ đi, thêm vào cặp $(Prev[k-1], H[k])$ trong đó $Prev[k]$ là chỉ số u nhỏ nhất thỏa mãn $H[u] \leq H[k]$, $sW[k] - sW[u-1] \leq L$ (có thể lập trước bằng cách sử dụng stack)
- Xét cặp đầu tiên (l, i) có $H[i] > l$. Nếu $sW[k] - sW[l] > L$ thì bỏ cặp (l, i) khỏi hàng đợi. Nếu $l < k-1$ thì thêm cặp $(l+1, i)$ vào hàng đợi.

Trong quá trình bỏ đi và thêm vào ta duy trì một **multiset** để quản lý $dp[l] + H[i]$.

Đoạn code cơ bản có thể viết:

```
deque<II> que; // Hàng đợi các cặp (l,i)
multiset<int64_t> bst; // Đa tập quản l. các giá trị
```

....

```

// Tinh mang qui hoach dong
dp[0]=0;
bst.clear();
que.clear();
for(int i=1;i<=n;++i)
{
    // Bỏ đi các phần tử cuối hàng đợi có chiều cao ≤ H[i]
    // chú . khi bỏ đi một phần tử ta cũng loại giá trị dp[..]+H[..]
    // của nó ra khỏi multiset
    while (!que.empty() && H[que.back().first]≤H[i])
    {
        II t=que.back();
        // Chú . cấu trúc dưới là xóa 1 phần tử trong multiset
        bst.erase(bst.find(dp[t.second]+H[t.first]));
        que.pop_back();
    }
    // Thêm phần tử (Prev[i]-1,i) vào cuối hàng đợi
    // đồng thời thêm dp[...]+ H[...] vào multiset
    que.push_back({i,Prev[i]-1});
    bst.insert(dp[Prev[i]-1]+H[i]);
    // Kiểm tra phần tử đầu hàng đợi, nếu khoảng cách >L th. bỏ nó đi
    // sau đó nếu l<i-1 th. thêm (l+1,i) vào đầu hàng đợi, đồng thời
    // cũng thêm giá trị dp[...]+H[...] của nó vào multiset
    while (sW[i]-sW[que.front().second]>L)
    {
        II t=que.front();
        bst.erase(bst.find(dp[t.second]+H[t.first]));
        que.pop_front();
        if (t.second<t.first-1)
        {
            que.push_front({t.first,t.second+1});
        }
    }
}

```

```

        bst.insert(dp[t.second+1]+H[t.first]);
    }
}

dp[i]=*bst.begin();
}

```

2.8.3. Chương trình minh họa

```

#include <bits/stdc++.h>
#define maxn 1000000
using namespace std;
multiset<long long> S;
deque<pair<long long, long long>> Q;
stack<long long> SS;
long long n, dp[maxn], L, W[maxn], H[maxn], Prev[maxn], Sw[maxn];
void prepare()
{
    for (int i = 1; i <= n; i++) Sw[i] = Sw[i - 1] + W[i];
    for (int i = 1; i <= n; i++)
    {
        while (!SS.empty() && H[SS.top()] <= H[i]) SS.pop();
        if (!SS.empty()) Prev[i] = SS.top();
        else Prev[i] = 0;
        SS.push(i);
    }
    long long j = 1;
    for (int i = 1; i <= n; i++)
    {
        while (Sw[i] - Sw[j - 1] > L) j++;
        Prev[i] = max(Prev[i], j - 1);
    }
}
int main()
{
    ios_base::sync_with_stdio(0); cin.tie(0); cout.tie(0);
    freopen("BOOKSHELF.inp", "r", stdin);
}

```

```

freopen("BOOKSHELF.out", "w", stdout);
scanf("%lld%lld", &n, &L);
for (int i = 1; i <= n; i++) scanf("%lld%lld", &H[i], &W[i]);
dp[0] = 0;
Q.clear();
S.clear();
prepare();
for (int i = 1; i <= n; i++)
{
    while (!Q.empty() && H[i] >= H[Q.back().first])
    {
        pair<long long, long long> cmp = Q.back();
        S.erase(S.find(dp[cmp.second] + H[cmp.first]));
        Q.pop_back();
    }
    Q.push_back({i, Prev[i]});
    S.insert(dp[Prev[i]] + H[i]);
    while (!Q.empty() && Sw[i] - Sw[Q.front().second] > L)
    {
        pair<long long, long long> cmp = Q.front();
        S.erase(S.find(dp[cmp.second] + H[cmp.first]));
        Q.pop_front();
        if (cmp.second < cmp.first - 1)
        {
            Q.push_front({cmp.first, cmp.second + 1});
            S.insert(dp[cmp.second + 1] + H[cmp.first]);
        }
    }
    dp[i] = *S.begin();
}
printf("%lld", dp[n]);
}

```

2.8.4. Test

<https://drive.google.com/file/d/12i7nqhEDiSFsssvX2ZingXtKyDglCuCG/view?usp=sharing>

3. Cải tiến bài toán quy hoạch động bằng kỹ thuật chia để trị

Tiếp cận bài toán quy hoạch động mở rộng kỹ thuật chia để trị là cách tiếp cận với hai kỹ thuật chính là dùng đệ quy có nhớ kết hợp lập bảng quy hoạch động dựa trên công thức truy hồi. Khi đó chúng ta có thể vừa lưu trữ và thực hiện các giải pháp của các bài toán giúp nâng cao hiệu suất thực hiện bài toán (xử lý tối ưu).

Ví dụ xét bài toán Fibonaci:

- Sử dụng đệ quy với độ phức tạp $O(2^n)$ bằng cách tiếp cận top - down

```
Fib(n)
{
    if (n < 2) result = n
    else result = Fib(n-2) + Fib(n-1)
    F[n] = result
    return F[n]
}
```

- Tiếp cận bottom up bằng phương pháp lập bảng quy hoạch động để giảm thời gian thực hiện thuật toán với độ phức tạp $O(n)$ như sau:

```
tabFib(n)
{
    F[0] = 0
    F[1] = 1
    for i = 2...n
        F[i] = F[i-2] + F[i-1]
    return F[n]
}
```

Ý tưởng chính của bài toán quy hoạch động được mở rộng kỹ thuật chia để trị là chúng ta phải xác định được bài toán lập bảng phương án quy hoạch động dựa trên công thức truy hồi nào và chia để trị ở đâu khi đó việc lưu trữ giá trị và

sử dụng chúng để tính cho các thời điểm của chương trình khi thực hiện nó được tối ưu hơn.

3.1. Bài tập 9. Famous Pagoda (F - ACM ICPC Vietnam Regional 2017) (Mức độ khó)

3.1.1. Đề bài

Khi xây dựng cầu thang đến các ngôi chùa nổi tiếng ở trên đỉnh núi, Chính quyền địa phương đã xác định N vị trí dọc theo sườn núi với các độ cao a_1, a_2, \dots, a_n . Trong đó $a_i < a_{i+1}$ và $0 < i < N$

Giá để xây dựng cầu thang từ vị trí đến vị trí j là:

$$\min_{v \in Z} \sum_{s=i}^j |a_s - v|^k$$

Để đẩy nhanh quá trình xây dựng cầu thang từ vị trí 1 đến vị trí N, chính quyền địa phương đã quyết định giao việc cho G nhà thầu xây dựng để xây dựng cầu thang song song nhau. Với N vị trí sẽ được chia thành G đoạn khác nhau và mỗi đoạn sẽ được phụ trách bởi một nhà thầu xây dựng khác nhau.

Với G nhà thầu xây dựng ($0 < G \leq N$) bạn hãy phân chia để G nhà thầu xây dựng cây cầu với tổng chi phí bé nhất.

Dữ liệu vào : PAGODA.INP có cấu trúc sau:

- Dòng 1 ghi ba số nguyên N, G, K ($N \leq 2000, 1 \leq G \leq N, 1 \leq k \leq 2$)
- Dòng 2 ghi dãy số nguyên a_1, a_2, \dots, a_n ($a_i \leq 10^6, a_i \leq a_{i+1}, 0 < i < N$) các vị trí cần xây dựng.

Kết quả ra: ghi vào file PAGODA.OUT giá trị xây dựng bé nhất.

Ví dụ

PAGODA.INP	PAGODA.OUT
5 1 2 1 2 3 4 5	10
5 1 1 1 2 3 4 5	6
5 2 2 1 2 3 4 5	3

3.1.2. Phân tích bài toán

Đặt $\text{cost}(i, j)$ là chi phí để xây cầu thang từ điểm i đến j .

Đặt $f(k, i) = \text{chi phí nhỏ nhất để k nhóm thợ xây tất cả cầu thang từ 1 đến } i$. Ta có công thức QHĐ:

- $f(k, i) = \min(f(k-1, j) + \text{cost}(j+1, i))$, với $j = 1..i-1$.

Kết quả của bài toán chính là $f(G, N)$.

Có 2 điểm mấu chốt của bài này:

- Tính $\text{cost}(i, j)$
- Tính nhanh bảng $f(k, i)$. Nếu tính trực tiếp theo công thức trên, ta mất độ phức tạp $O(G*N^2)$, không đủ để AC bài này.

1. Tính $\text{cost}(i, j)$

Với $k = 1$:

- Khi $k = 1$, điểm v chính là median của dãy $A(i)..A(j)$.
- Do dãy A đã được sắp xếp tăng dần, $v = A[i + (j - i + 1) / 2]$
- Với mỗi (i, j) , ta có thể tính nhanh $\text{cost}(i, j)$ trong $O(1)$ bằng [mảng cộng dồn](#).

Với $k = 2$:

- Đặt $L = \text{số phần tử của đoạn } A(i)..A(j) = j - i + 1$
- $\text{cost}(i, j) = \sum((a(s) - v)^2)$
 - $= \sum(a(s)^2) + L*v^2 + 2*\sum(a(s))*v$
- Đặt $B = 2*\sum(a(s))$, $C = \sum(a(s)^2)$. Ta tính được B và C trong $O(1)$ bằng [mảng cộng dồn](#).
- $\text{cost}(i, j) = L*v^2 + B*v + C$, là một hàm bậc 2 của v . Do đó điểm v để làm $\text{cost}(i, j)$ nhỏ nhất chính là $v = -B / L$. Tuy nhiên $(-B / L)$ có thể là số thực, còn trong bài toán này v phải là số nguyên, nên ta cần xét 2 số nguyên v gần nhất với $(-B / L)$ bởi vì đồ thị hàm số lồi

2. Tính $f(k, i)$

Để tính $f(k, i)$, ta cần dùng kỹ thuật [QHĐ chia để trị](#).

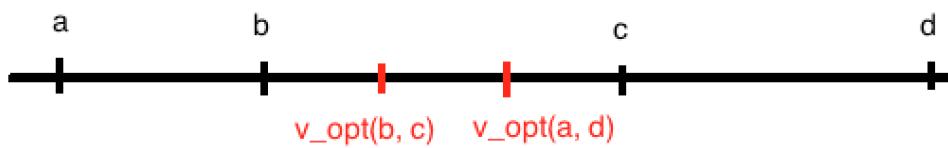
Ta có thể sử dụng được QHĐ chia để trị nếu:

- $\text{cost}(a, d) + \text{cost}(b, c) \geq \text{cost}(a, c) + \text{cost}(b, d)$ với mọi $a < b < c < d$

Đặt:

- $f(i, j, v) = (\sum (|a(s) - v|^k) \text{ với } s = i..j)$
- $v_{\text{opt}}(a, b) = v$ để $f(a, b, v)$ đạt giá trị nhỏ nhất
- $\text{cost}(a, b) = f(a, b, v_{\text{opt}}(a, b)) \leq f(a, b, v)$ với mọi v .

Không làm mất tính tổng quát, giả sử $v_{\text{opt}}(b, c) \leq v_{\text{opt}}(a, d)$. Ngoài ra ta cũng biết rằng $v_{\text{opt}}(b, c)$ nằm trong khoảng $[b, c]$.



Ta có:

$$\begin{aligned}
 & \text{cost}(a, d) \\
 &= f(a, d, v_{\text{opt}}(a, d)) \\
 &= f(a, b-1, v_{\text{opt}}(a, d)) + f(b, d, v_{\text{opt}}(a, d)) \\
 &\geq f(a, b-1, v_{\text{opt}}(a, d)) + \text{cost}(b, d)
 \end{aligned}$$

Mặt khác, do tất cả đoạn $[a, b-1]$ ở bên trái $v_{\text{opt}}(b, c)$ và $v_{\text{opt}}(b, c) < v_{\text{opt}}(a, d)$ nên:

$$\begin{aligned}
 & f(a, b-1, v_{\text{opt}}(a, d)) + \text{cost}(b, c) \\
 &\geq f(a, b-1, v_{\text{opt}}(b, c)) + \text{cost}(b, c) \\
 &= f(a, b-1, v_{\text{opt}}(b, c)) + f(b, c, v_{\text{opt}}(b, c)) \\
 &= f(a, c, v_{\text{opt}}(b, c)) \\
 &\geq \text{cost}(a, c)
 \end{aligned}$$

Kết hợp 2 bất đẳng thức trên:

$$\begin{aligned}
 & \text{cost}(a, d) + \text{cost}(b, c) \\
 &\geq f(a, b-1, v_{\text{opt}}(a, d)) + \text{cost}(b, d) + \text{cost}(b, c) \\
 &= f(a, b-1, v_{\text{opt}}(a, d)) + \text{cost}(b, c) + \text{cost}(b, d) \\
 &\geq \text{cost}(a, c) + \text{cost}(b, d)
 \end{aligned}$$

Chứng minh hoàn tất. Kết luận ta có thể dùng QHĐ chia để trị để giải bài toán với độ phức tạp $O(G^*N^*\log(G))$

3.1.3. Chương trình minh họa

```
#include <bits/stdc++.h>
using namespace std;
const int maxn = 2002;
int n, g, k;
int a[maxn];
long long cost[maxn][maxn];
inline long long calc(long long a, long long b, long long c) {
    if (b % (2 * a) == 0) {
        long long x = b / (2 * a);
        return a * x * x - b * x + c;
    } else {
        long long x = b / (2 * a);
        long long y = x + 1;
        return min(a * x * x - b * x + c, a * y * y - b * y + c);
    }
}
void prepare() {
    if (k == 1) {
        for (int i = 1; i <= n; ++i) {
            long long cur_cost = -a[i];
            for (int j = i; j <= n; ++j) {
                cur_cost += a[j];
                cost[i][j] = cur_cost;
                cur_cost -= a[j + i + 1 >> 1];
            }
        }
    }
}
```

```

} else {
    for (int i = 1; i <= n; ++i) {
        long long sa = 0, ssa = 0, s = 0;
        for (int j = i; j <= n; ++j) {
            sa += a[j];
            ssa += (long long)a[j] * a[j];
            s++;
            cost[i][j] = calc(s, 2 * sa, ssa);
        }
    }
}
}

long long f[maxn][maxn];
void divide(long long f[], long long g[], int l, int r, int pl, int pr) {
    if (l > r) {
        return;
    }
    int m = (l + r) / 2;
    int ptr;
    for (int i = pl; i < min(m, pr + 1); ++i) {
        long long val = f[i] + cost[i + 1][m];
        if (val <= g[m]) {
            g[m] = val;
            ptr = i;
        }
    }
    divide(f, g, l, m - 1, pl, ptr);
    divide(f, g, m + 1, r, ptr, pr);
}

```

```

int main() {
    #ifdef LOCAL
        freopen("input.txt", "r", stdin);
        freopen("output.txt", "w", stdout);
    #endif // LOCAL
    scanf("%d%d%d", &n, &g, &k);
    for (int i = 1; i <= n; ++i) {
        scanf("%d", a + i);
    }
    prepare();
    memset(f, 0x3f, sizeof(f));
    f[0][0] = 0;
    for (int i = 1; i <= g; ++i) {
        divide(f[i - 1], f[i], i, n, i - 1, n - 1);
    }
    printf("%lld\n", f[g][n]);
    return 0;
}

```

3.1.4. Test

https://drive.google.com/file/d/1tF_pUJsHc8H5K5v5qx2fHrwsIXzYv7RJ/view?usp=sharing

3.2. Bài tập 10. SEQPART (Mức độ khó)

3.2.1. Đề bài

Cho dãy L gồm các số $C[1..L]$, cần chia dãy này thành G đoạn liên tiếp. Với phần tử thứ i , ta định nghĩa chi phí của nó là tích của C_i và số lượng các số nằm cùng đoạn liên tiếp với C_i . Chi phí của dãy số ứng với một cách phân hoạch là tổng các chi phí của các phần tử của G đoạn đã chia.

Yêu cầu: Hãy xác định cách phân hoạch dãy số để chi phí là nhỏ nhất.

Dữ liệu vào : tệp SEQPART.INP có cấu trúc sau

- Dòng đầu tiên chứa 2 số L và G.
- L dòng tiếp theo, chứa giá trị của dãy C_1, C_2, \dots, C_n .

Kết quả ra: ghi vào tệp SEQPART.OUT một dòng duy nhất là chi phí nhỏ nhất.

Ràng buộc:

- $1 \leq L \leq 8000$.
- $1 \leq G \leq 800$.
- $1 \leq C_i \leq 10^9$.

Ví dụ:

SEQPART.INP	SEQPART.OUT
6 3	299
11	
11	
11	
24	
26	
100	

Giải thích:

- Cách tối ưu là $C[]=(11,11,11), (24,26), (100)$.
- Chi phí: $11*3 + 11*3 + 11*3 + 24*2 + 26*2 + 100*1 = 299$.

3.2.2. Phân tích bài toán

Đây là dạng bài toán phân hoạch dãy số có thể dễ dàng giải bài QHĐ. Gọi $F(g,i)$ là chi phí nhỏ nhất nếu ta phân hoạch i phần tử đầu tiên thành g nhóm, khi đó kết quả bài toán sẽ là $F(G,L)$.

Để tìm công thức truy hồi cho hàm $F(g,i)$, ta sẽ quan tâm đến nhóm cuối cùng. Coi phần tử 0 là phần tử cầm canh ở trước phần tử thứ nhất, thì người cuối cùng không thuộc nhóm cuối có chỉ số trong đoạn $[0,i][0,i]$. Giả sử đó là người với chỉ số k, thì chi phí của cách phân hoạch sẽ là $F(g-1,k)+Cost(k+1,i)$ với $Cost(i,j)$ là chi phí nếu phân $j-i+1$ người có chỉ số $[i,j]$ vào một nhóm. Như vậy:

$$F(g,i)=\min(F(g-1,k)+Cost(k+1,i)) \text{ với } 0 \leq k \leq i.$$

Chú ý là công thức này chỉ được áp dụng với $g > 1$, nếu $g = 1, F(1,i) = \text{Cost}(1,i)$ đây là trường hợp cơ sở.

Chú ý là ta sử dụng mảng sum[] tiền xử lí $O(L)$ để có thể truy vấn tổng một đoạn (dùng ở hàm cost()) trong $O(1)$. Như vậy độ phức tạp của thuật toán này là $O(G * L * L)$.

➤ Thuật toán tối ưu hơn

Gọi $P(g,i)$ là k nhỏ nhất để cực tiểu hóa $F(g,i)$, nói cách khác $P(g,i)$ là k nhỏ nhất mà $F(g,i) = F(g-1,k) + \text{Cost}(k+1,i)$.

Tính chất quan trọng để có thể tối ưu thuật toán trên là dựa vào tính đơn điệu của $P(g,i)$, cụ thể: $P(g,0) \leq P(g,1) \leq P(g,2) \leq \dots \leq P(g,L-1) \leq P(g,L)$.

✓ Chia để trị

Để ý rằng để tính $F(g,i)$, ta chỉ cần quan tâm tới hàng trước $F(g-1)$ của ma trận: $F(g-1,0), F(g-1,1), \dots, F(g-1,L)$.

Như vậy, ta có thể tính hàng $F(g)$ theo thứ tự bất kỳ.

Ý tưởng là với hàng g , trước hết ta tính $F(g,mid)F(g,mid)$ và $P(g,mid)$ với $mid = L/2$, sau đó sử dụng tính chất nêu trên $P(g,i) \leq P(g,mid)$ với $i < mid$ và $P(g,i) \geq P(g,mid)$ với $i > mid$ để gọi đệ quy đi tính hai nửa còn lại.

3.2.3. Chương trình minh họa

```
#include <bits/stdc++.h>
const int MAXL = 8008;
const int MAXG = 808;
const long long INF = (long long)1e18;
using namespace std;
long long F[MAXG][MAXL], sum[MAXL], C[MAXL];
int P[MAXG][MAXL];

long long cost(int i, int j) {
    if (i > j) return 0;
    return (sum[j] - sum[i - 1]) * (j - i + 1);
```

```

}

void divide(int g, int L, int R, int optL, int optR) {
    if (L > R) return;
    int mid = (L + R) / 2;
    F[g][mid] = INF;
    for (int i = optL; i <= optR; ++i) {
        long long new_cost = F[g - 1][i] + cost(i + 1, mid);
        if (F[g][mid] > new_cost) {
            F[g][mid] = new_cost;
            P[g][mid] = i;
        }
    }
    divide(g, L, mid - 1, optL, P[g][mid]);
    divide(g, mid + 1, R, P[g][mid], optR);
}
int main() {
    int G, L;
    cin >> L >> G;
    for (int i = 1; i <= L; ++i) {
        cin >> C[i];
        sum[i] = sum[i - 1] + C[i];
    }
    for (int i = 1; i <= L; ++i) F[1][i] = cost(1, i);
    for (int g = 2; g <= G; ++g) divide(g, 1, L, 1, L);
    cout << F[G][L] << endl;
    return 0;
}

```

3.2.4. Test

https://drive.google.com/file/d/1fShvZRU_NyEuK2PG_8oQXiPTtQXlSjE/view?usp=sharing

4. Một số bài khác

1. <https://codeforces.com/group/FLVn1Sc504/contest/272391/problem/D> VOI 2013 - Trộn xâu

2. <https://vn.spoj.com/problems/LSFIGHT/> LSFIGHT - Đấu trường VM08

3. <https://codeforces.com/group/FLVn1Sc504/contest/274825/problem/D>

Đường đi trên lưới

4. <https://codeforces.com/group/FLVn1Sc504/contest/274804/problem/E> VOI 2011 Phản thường

5. <https://codeforces.com/group/FLVn1Sc504/contest/274821/problem/K>

VOI09 Trò chơi với bảng số

6. <https://codeforces.com/group/FLVn1Sc504/contest/274804/problem/S> Hàng động

7. <https://vn.spoj.com/problems/VSTEPS/> Bậc thang

8. <https://codeforces.com/group/FLVn1Sc504/contest/274829/problem/I>

PVOI14_4 - Chữ M

5. Kết luận

Bài toán tối ưu là một trong những lớp bài toán có nhiều ứng dụng trong nghiên cứu và thực tế. Kỹ thuật quy hoạch động là dạng hay gặp trong các bài toán tối ưu. Chính vì vậy việc rèn tư duy cho học sinh khối chuyên có thể áp dụng được trong khi lập trình cho các bài toán đặc trưng là rất quan trọng và cần thiết. Chuyên đề trên là một số vấn đề cần dạy các tư duy và tiếp cận cho học sinh chuyên khi bắt đầu học về quy hoạch động. Mở rộng bài toán quy hoạch động với kỹ thuật chia để trị là cách để giảm độ phức tạp khi lập trình với các bài toán quy hoạch động.

Do kinh nghiệm chưa nhiều, nên chắc chắn có thiếu sót. Tôi rất mong đồng nghiệp và bạn bè chia sẻ, góp ý thêm cho tôi. Tôi xin chân thành cảm ơn!

6. Nguồn tài liệu tham khảo

<http://vnoin.info>

<https://vn.spoj.com>

<https://www.geeksforgeeks.org>

<https://codeforces.com>

<https://www.codechef.com>