

MỤC LỤC

MỤC LỤC 1

1. Mở đầu	2
2. Cải tiến bài toán quy hoạch động mở rộng với kỹ thuật chia để trị	2
Xét ví dụ sau: bài toán <i>dãy số</i>	3
Hướng dẫn thuật toán:.....	3
Code mẫu:.....	4
Link Test và code mẫu:.....	5
3. Bài tập ứng dụng	5
Bài 1: Famous Pagoda (F - ACM ICPC Vietnam Regional 2017) Mức độ khó.....	7
Hướng dẫn thuật toán:.....	8
Code mẫu:.....	9
Link Test và code mẫu:.....	11
Bài 2: SEQPART – Mức độ khó	5
Hướng dẫn thuật toán:.....	6
Code mẫu:.....	6
Link Test và code mẫu:.....	7
Bài 3. Bài toán Dãy Fibonacci – FIBSEQ. Mức độ khó	11
Hướng dẫn thuật toán:.....	12
Code mẫu:.....	13
Link Test và code mẫu:.....	15
Bài 4: Mining – mức độ khó	15
Hướng dẫn thuật toán:.....	15
Code mẫu:.....	16
Link Test và code mẫu:.....	18
Bài 5: F. Yet Another Minimization Problem – Mức độ khó.....	18
Hướng dẫn thuật toán:.....	18
Code mẫu:.....	19
Link Test và code mẫu:.....	20
4. Bài tập tham khảo	20
5. Kết luận	20
6. Tài liệu tham khảo	20

CẢI TIẾN BÀI TOÁN QUY HOẠCH ĐỘNG BẰNG KỸ THUẬT CHIA ĐỂ TRỊ

1. Mở đầu

Trong những năm gần đây nội dung thi học sinh giỏi Tin học cấp Quốc gia nội dung và kiến thức ngày càng được nâng cao đòi hỏi học sinh phải có sự tư duy và sáng tạo trong lập trình. Bài toán tổng hợp từ nhiều kiến thức khác nhau. Trong mỗi bài toán đòi hỏi về tư duy thuật toán tốt và kỹ năng lập trình cao mới đạt điểm tối đa của bài. Kỹ thuật quy hoạch động là một trong những kỹ thuật thường hay gặp trong các kì thi học sinh giỏi Tin học Quốc gia và Quốc tế.

Bài toán giải bằng phương pháp quy hoạch động thường là bài toán tối ưu. **Để cải tiến thuật toán quy hoạch động thường có các kỹ thuật như chia để trị, sử dụng bao lồi đường thẳng Convex hull trick, kỹ thuật đổi biến số hoặc kết hợp các cấu trúc dữ liệu nâng cao như stack, set, multiset, deque..** để tăng tốc bài toán tối ưu. Để giảm độ phức tạp thuật toán từ $O(n^3)$ về $O(n^2 \log n)$, hay từ $O(n^2)$ về $O(n \log n)$ hoặc là từ $O(n)$ về $O(\log n)$ tùy thuộc vào độ lớn dữ liệu từng bài toán và kỹ thuật lập trình kết hợp công thức quy hoạch động để giải quyết bài toán tối ưu một cách hiệu nhất.

Chia để trị là một trong những phương pháp hiệu quả nhất 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.

Trong chuyên đề này tôi giới thiệu kỹ thuật cải tiến và tăng tốc bài toán quy hoạch động bằng phương pháp chia để trị nhằm giảm độ phức tạp thuật toán và nâng cao hiệu quả khi lập trình.

2. Cải tiến bài toán quy hoạch động mở rộng với 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** (memorization) 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:

```
Fib(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.

Xét ví dụ sau: bài toán dãy số.

Trong tiết học về dãy số tại trường, thầy giáo của Tý cho cả lớp chơi một trò chơi như sau: Cho một dãy số A bao gồm N số nguyên, yêu cầu hãy chia dãy số trên thành hai phần liên tiếp sao cho tổng các số ở phần bên trái bằng tổng các số ở phần bên phải. Với mỗi bước như vậy bạn được 1 điểm còn nếu không thể chia được thì trò chơi sẽ kết thúc. Sau khi chia thành công bạn sẽ được chọn dãy số bên trái hoặc bên phải để tiếp tục cuộc chơi với các bước như trên cho đến khi trò chơi kết thúc.

Là một học sinh giỏi trong lớp, Tý muốn đạt được số điểm cao nhất có thể. Bạn hãy tính xem số điểm lớn nhất mà Tý có thể đạt được là bao nhiêu?

Dữ liệu vào từ tệp văn bản SEQ.INP:

- ✓ Dòng đầu tiên ghi một số nguyên T ($1 \leq T \leq 10$) là số lượng bộ dữ liệu. Mỗi bộ liệu bao gồm hai dòng:
- ✓ Dòng đầu tiên ghi một số nguyên N là số lượng phần tử của dãy A .
- ✓ Dòng thứ hai gồm N phần tử của dãy A được ghi cách nhau bởi dấu cách ($0 \leq a_i \leq 10^9$).

Kết quả ghi ra tệp văn bản SEQ.OUT: Với mỗi bộ dữ liệu in ra một số nguyên trên một dòng là kết quả của bộ dữ liệu đó.

Ví dụ:

SEQ. INP	SEQ. OUT
3	0
3	2
3 3 3	3
4	
2 2 2 2	
7	
4 1 0 1 1 0 1	

Giới hạn:

- ✓ 30% số bộ dữ liệu có $N \leq 200$.
- ✓ 60% số bộ dữ liệu có $N \leq 2000$.
- ✓ 100% số bộ dữ liệu có $N \leq 20000$.

Hướng dẫn thuật toán:

Đây là dạng bài toán quy hoạch động cơ bản kết hợp với kỹ thuật chia để trị để có thể giải quyết tối ưu bài toán một cách dễ dàng:

- Đầu tiên xử lý dữ liệu vào bằng kỹ thuật tổng cộng dồn $O(n)$.
- Để phân chia dãy a_1, a_2, \dots, a_n thành 2 đoạn có tổng bằng nhau ta sử dụng thuật toán chặt nhị phân để tìm vị trí **low** vị trí đó chia dãy số thành hai dãy có tổng của mỗi dãy bằng nhau.
- Khi đó đáp án của bài toán là $ans = 1 + \max(tinh(1, low), tinh(low+1, n))$
- Kết hợp giữa kỹ thuật quy hoạch động và chia để trị như sau:
 - + Đầu tiên ta tìm vị trí mà tổng của hai dãy bằng nhau bằng kỹ thuật chặt nhị phân
 $long long half = (s[r] - s[l]) / 2;$
 $int low = l, high = r;$
 $while (low + 1 < high) {$
 $int mid = (low + high) / 2;$

```

if (s[mid] - s[l] <= half) {
    low = mid;
} else {
    high = mid;
}

```

Sau đó dựa vào công thức tính ở trên ta kết hợp với kĩ thuật chia đôi trị để xây dựng hàm Calc(int l, int r) để tính số điểm lớn nhất mà Tý có thể đạt được.

```

int calc(int l, int r) {
    if (r - l <= 1) {
        return 0;
    }
    if ((s[r] - s[l]) % 2) {
        return 0;
    }
    if (s[r] - s[l] == 0) {
        return r - l - 1;
    }
    long long half = (s[r] - s[l]) / 2;
    int low = l, high = r;
    while (low + 1 < high) {
        int mid = (low + high) / 2;
        if (s[mid] - s[l] <= half) {
            low = mid;
        } else {
            high = mid;
        }
    }
    return s[low] - s[l] == half ? max(calc(l, low), calc(low, r)) + 1 : 0;
}

```

Độ phức tạp của bài toán này là $O(T * N \log N)$.

Code mẫu:

```

#include <bits/stdc++.h>

using namespace std;

const int maxn = (int)2e4 + 5;

int t;
int n;
long long s[maxn];

int calc(int l, int r) {
    if (r - l <= 1) {
        return 0;
    }
    if ((s[r] - s[l]) % 2) {
        return 0;
    }
}

```

```

if (s[r] - s[l] == 0) {
    return r - l - 1;
}
long long half = (s[r] - s[l]) / 2;
int low = l, high = r;
while (low + 1 < high) {
    int mid = (low + high) / 2;
    if (s[mid] - s[l] <= half) {
        low = mid;
    } else {
        high = mid;
    }
}
return s[low] - s[l] == half ? max(calc(l, low), calc(low, r)) + 1 : 0;
}

void solve() {
    scanf("%d", &n);
    for (int i = 1; i <= n; ++i) {
        int a;
        scanf("%d", &a);
        s[i] = s[i - 1] + a;
    }
    printf("%d\n", calc(0, n));
}

int main() {
    freopen("seq.inp", "r", stdin);
    freopen("seq.out", "w", stdout);
    scanf("%d", &t);
    while (t--) {
        solve();
    }
    return 0;
}

```

Link Test và code mẫu:

https://drive.google.com/drive/folders/1Nn3rQORBbU_lws4x4f0HBpb8pV821i9k?usp=sharing

3. Bài tập ứng dụng

Bài 1: SEQPART – Mức độ khá

Link bài: [https://www.hackerrank.com/contests/ioi-2014-practice-contest-2 /challenges/guardians-lunatics-ioi14](https://www.hackerrank.com/contests/ioi-2014-practice-contest-2/challenges/guardians-lunatics-ioi14)

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 11 11 11 24 26 100	299

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

Hướng dẫn thuật 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 vào tối đa 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 $[0,i]$. 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 $g \geq 1$, nếu $g=1$, $F(1,i)=Cost(1,i)$ 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 $F(g,i) = F(g-1,k) + Cost(k+1,i)$.

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 $F(g,mid)$ 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 đi gọi đệ quy đi tính hai nửa $F[g]$ còn lại.

Độ phức tạp: $O(G \cdot L \cdot \log L)$.

Code mẫu:

```
#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;
}

```

Link Test và code mẫu:

<https://drive.google.com/drive/folders/15b9i12gwI7tATNGeVyDHih8aauu1KjT3?usp=sharing>

Bài 2: Famous Pagoda (F - ACM ICPC Vietnam Regional 2017) Mức độ khá

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 i 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 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 ($0 < G$) bạn hãy phân chia để G nhà thầu xây dựng cây cầu với tổng chi phí bé nhất. Nếu số nhà thầu lớn hơn hoặc bằng số vị trí xây dựng thì chi phí xây dựng bằng 0. Do nhà thầu muốn quảng bá hình ảnh của mình nên xây dựng miễn phí cho nhà chùa!

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

- Dòng 1 ghi ba số nguyên N, G, K ($1 \leq N \leq 2000, 1 \leq G \leq N, 1 \leq k \leq 2$).
- Dòng 2 ghi dãy số nguyên a_1, a_2, \dots, a_n ($1 \leq a_i \leq 10^6, a_i \leq a_{i+1} \forall 0 < i < N$) 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 1 1 2 3 4 5	6
5 1 2 1 2 3 4 5	10

Hướng dẫn thuật toán:

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

Đặt $f(k, i) = \text{chi phí nhỏ nhất để } k \text{ nhóm thợ xây tất cả cầu thang từ } 1 \text{ đến } i$. 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 $O(G*N^2)$ với dữ liệu như đề bài thì thuật toán chưa tối ưu.

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 $L = \text{số phần tử của đoạn } A(i)..A(j) = j - i + 1$
- ✓ $\text{cost}(i, j) = \text{sum}((a(s) - v)^2)$
 - $= \text{sum}(a(s)^2) - 2 * \text{sum}(a(s)) * v + L * v^2$
- ✓ Đặt $B = 2 * \text{sum}(a(s))$, $C = \text{sum}(a(s)^2)$. 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 kĩ thuật QHĐ chia để trị.

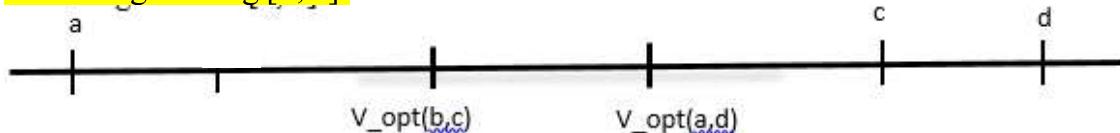
Ta có thể sử dụng được QHĐ chia để trị nếu:

- $\text{cost}(a, d) + \text{cost}(b, c) \geq \text{cost}(a, c) + \text{cost}(b, d)$ với mọi $a < b < c < d$ (Bất đẳng thức tú giác)

Đặt:

- ✓ $f(i, j, v) = (\sum |a(s) - v|^k)$ 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:

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

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

Code mẫu:

```
#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) {
                cost[i][j] = cur_cost;
            }
            cur_cost += a[i];
        }
    }
}
```

```

        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] + cos0t[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() {
    freopen("PAGODA.inp", "r", stdin);
    freopen("PAGODA.out", "w", stdout);
    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;
}

```

Link Test và code mẫu:

https://drive.google.com/drive/folders/1vh2FgsyUzEaHj_s2SxfvDxngC_3ap-1v?usp=sharing

Bài 3. Bài toán Dãy Fibonacci – FIBSEQ. Mức độ khá

(*Nguồn: Đề thi học sinh giỏi quốc gia năm 2017*)

Năm 1202, Leonardo Fibonacci, nhà toán học người Ý, tình cờ phát hiện ra tỉ lệ bằng 0.618 được tiệm cận bằng thương của hai số liên tiếp trong một loại dãy số vô hạn được một số nhà toán học ÁN ĐỘ xét đến từ năm 1150. Sau đó dãy số này được đặt tên là dãy số Fibonacci $\{F_i : i=1, 2, \dots\}$, trong đó $F_1=F_2=1$ và mỗi số tiếp theo trong dãy được tính bằng tổng của hai số ngay trước nó. Đây là 10 số đầu tiên của dãy số Fibonacci: 1, 1, 2, 3, 5, 8, 13, 21, 34, 55. Người ta đã khám phá ra mối liên hệ chặt chẽ của số Fibonacci và tỉ lệ vàng với sự phát triển trong tự nhiên (cánh hoa, cành cây, vân gỗ), trong vũ trụ (hình xoáy tròn ốc dải ngân hà, khoảng cách giữa các hành tinh), hay sự cân đối của cơ thể con người. Đặc biệt số Fibonacci được ứng dụng mạnh mẽ trong kiến trúc (Kim tự tháp Ai Cập, tháp Eiffel), trong mỹ thuật (các bức tranh của Leonardo da Vinci), trong âm nhạc (các bản giao hưởng của Mozart) và trong nhiều lĩnh vực khoa học kỹ thuật.

Trong toán học, dãy Fibonacci là một đối tượng tổ hợp quan trọng có nhiều tính chất đẹp. có nhiều phương pháp hiệu quả liệt kê và tính các số Fibonacci như phương pháp lặp hay phương pháp nhân ma trận.

Sau khi được học về dãy số Fibonacci, Sơn rất muốn phát hiện thêm những tính chất của dãy số này. Vì thế Sơn đặt ra bài toán sau đây: Hỏi rằng có thể tìm được một tập con các số trong n số Fibonacci liên tiếp bắt đầu từ số thứ i, sao cho tổng của chúng chia hết cho một số nguyên dương k ($k \leq n$) cho trước hay không? Nhắc lại, một tập con q số của một dãy n số là một cách chọn ra q số bất kỳ trong n số của dãy đó, mỗi số được chọn không quá một lần.

Yêu cầu: Hãy giúp Sơn giải quyết bài toán đặt ra.

Dữ liệu: Vào từ file văn bản FIBSEQ.INP bao gồm:

- Dòng thứ nhất ghi số nguyên dương T ($T \leq 10$) là số lượng bộ dữ liệu.
- Mỗi dòng trong T dòng tiếp theo chứa ba số nguyên dương n, i và k là thông tin của một bộ dữ liệu.

Các số trên cùng dòng được ghi cách nhau bởi dấu cách.

Kết quả: Ghi ra file văn bản FIBSEQ.OUT bao gồm T dòng tương ứng với kết quả của T bộ dữ liệu đầu vào, mỗi dòng có cấu trúc như sau: Đầu tiên ghi số nguyên q là số lượng các số trong tập con tìm được, tiếp đến ghi q số nguyên là các số thứ tự trong dãy Fibonacci của q số trong tập con tìm được. Nếu không tìm được tập con thỏa mãn điều kiện đặt ra thì ghi ra một số 0.

Nếu có nhiều cách chọn thì chỉ cần đưa ra một cách chọn bất kỳ.

Ràng buộc:

- Có 20% số lượng test thỏa mãn điều kiện: $n \leq 20, i \leq 10^6$
- Có 20% số lượng test thỏa mãn điều kiện: $n \leq 10^3, i \leq 10^6$

- Có 20% số lượng test thỏa mãn điều kiện: $n \leq 10^6, i \leq 10^6$
- Có 10% số lượng test thỏa mãn điều kiện: $n \leq 20, i \leq 10^{15}$
- Có 10% số lượng test thỏa mãn điều kiện: $n \leq 10^3, i \leq 10^{15}$
- Có 20% số lượng test còn lại thỏa mãn điều kiện: $n \leq 10^6, i \leq 10^{15}$

Ví dụ:

FIBSEQ.INP	FIBSEQ.OUT
1 10 3 9	2 5 7

Giải thích: Trong ví dụ trên một tập con thỏa mãn điều kiện đặt ra là tập gồm 2 số $F_5=5, F_7=13$ với tổng bằng 18.

Hướng dẫn thuật toán:

Một kĩ thuật của thuật toán sử dụng chia để trị thường dùng đó là kĩ thuật nhân ma trận.

Ta cần phải tính n số Fibonacci bắt đầu từ số thứ i (viết tắt là F_i). Để thuận tiện khi lập trình, ta sẽ lưu n số Fibonacci, bắt đầu từ số Fibonacci thứ i vào mảng **val**:

$\text{val}[1]=F_i, \text{val}[2]=F_{i+1}, \dots, \text{val}[n]=F_{i+n-1}$. Chú ý các số Fibonacci đã được mod cho k .

+ Với $i \leq 10^6$, ta chỉ cần thực hiện vòng lặp để tính các số Fibonacci chia lấy dư cho k .

+ Với $i \leq 10^{15}$, phải sử dụng kĩ thuật nhân ma trận để tính các số Fibonacci chia lấy dư cho k .

- Sub1: $n \leq 20, i \leq 10^6$

Dùng vòng lặp để tính các số Fibonacci chia lấy dư cho k : $O(i)$

Duyệt các dãy con liên tiếp các số Fibonacci tính từ số thứ i , với mỗi dãy con đó tính tổng các số trong dãy, nếu gặp được dãy có tổng chia hết cho k thì dừng. Dùng 3 vòng lặp lồng nhau: 2 biến i, j ($i, j: 1 \rightarrow n$) để duyệt qua vị trí đầu và cuối của dãy con; biến z trong vòng lặp thứ 3 để tính tổng các số Fibo từ vị trí thứ i đến vị trí thứ j : $O(n^3)$

Độ phức tạp: $O(T^*(i+n^3))$ với T là số lượng test.

- Sub2: $n \leq 10^3, i \leq 10^6$

Dùng vòng lặp để tính các số Fibonacci chia lấy dư cho k : $O(i)$

Dùng mảng S tính tổng dồn của các số Fibonacci. Dùng 2 vòng lặp lồng nhau: 2 biến i, j ($i, j: 1 \rightarrow n$) để duyệt qua vị trí đầu và cuối của dãy con; tính tổng các số trong mỗi dãy con dựa vào mảng tính tổng dồn, sau đó kiểm tra điều kiện. Nếu gặp một dãy con nào đó thỏa mãn thì dừng vòng lặp ngay.

Độ phức tạp: $O(T^*(i+n^2))$ với T là số lượng test.

- Sub3: $n \leq 10^6, i \leq 10^6$

Dùng vòng lặp để tính các số Fibonacci chia lấy dư cho k : $O(i)$

Dùng mảng S tính tổng dồn của các số Fibonacci. Nhưng ở đây ta phải sử dụng một thuật toán tối ưu hơn thuật toán ở sub2. Đó là dựa vào *định lí Dirichle*.

Mảng S sẽ có $n + 1$ phần tử từ 0 đến n , mà có dữ kiện $k \leq n$ nên theo nguyên lí Dirichle thì trong mảng S chắc chắn có 2 số S_p, S_q có cùng số dư cho k (hay tổng $S_{p+1} + \dots + S_q$ chia hết cho k). Vậy các phân tử ở vị trí từ $p+1$ tới q trong mảng val sẽ chia hết cho k . Nên sẽ đưa ra được vị trí của các phân tử đó trong dãy Fibonacci ban đầu.

Sử dụng kĩ thuật đánh dấu các giá trị tổng dồn đã có vào mảng index, khi tạo ra một giá trị tổng dồn mới, kiểm tra trong mảng đánh dấu đã có chưa, nếu có rồi thì in ra kết quả và kết thúc.

```

    memset(index, -1, sizeof(index)); //mảng đánh dấu
    for (int i=0; i<n+1; i=i+1){
        //mảng tổng dồn là mảng sum
        //nếu chưa có giá trị sum[i] thì đánh dấu chỉ số, ngược lại in ra kết //quả rồi kết thúc
        if (index[sum[i]] < 0) index[sum[i]] = i;
    }

```

```

else { int t = index[sum[i]];
    //in ra kết quả
    cout << i - t;
    FOR(j=t+1;j<=i; j++) cout << " " << s + j - 1;
    cout << endl;
    return; } }

```

Độ phức tạp: $O(T^*(i+n))$ với T là số lượng test.

- **Sub4: $n \leq 20$, $i \leq 10^{15}$:** Với subtask này ta áp dụng chia để trị thông qua kĩ thuật nhân ma trận.

Làm tương tự sub1 nhưng phải sử dụng kĩ thuật nhân ma trận để tính các số Fibonacci chia lấy dư cho k.

Độ phức tạp: $O(T^*(\log(i)+n^3))$ với T là số lượng test.

- **Sub5: $n \leq 10^3$, $i \leq 10^{15}$**

Làm tương tự sub2 nhưng phải sử dụng kĩ thuật nhân ma trận để tính các số Fibonacci chia lấy dư cho k.

Độ phức tạp: $O(T^*(\log(i)+n^2))$ với T là số lượng test.

- **Sub6: $n \leq 10^6$, $i \leq 10^{15}$**

Làm tương tự sub3 nhưng phải sử dụng kĩ thuật nhân ma trận để tính các số Fibonacci chia lấy dư cho k.

Độ phức tạp: $O(T^*(\log(i)+n))$ với T là số lượng test.

Code mẫu:

```

#include <bits/stdc++.h>

using namespace std;
#define N 1000001
int n;
long long ii,MOD;
long long aa[N],fr[N];
typedef long long ll;
#define matrix vector<vector<ll>>

matrix a;// = {{0,1},{1,1}};
matrix r;// = {{0,0},{0,0}};
matrix rr;// = {{0,0},{0,0}};

matrix operator* (const matrix &aa, const matrix &bb ) {
    matrix c = rr;
    for (int i = 0; i<2; i++) {
        for (int j = 0; j<2; j++) {
            for (int k = 0; k<2; k++) {
                c[i][j] = (c[i][j] + (aa[i][k] * bb[k][j]) % MOD) % MOD;
            }
        }
    }
    return c;
}
matrix power(long long n) {
    if (n == 1) return a;
    matrix tmp = power(n / 2);
    tmp = tmp * tmp;
}

```

```

if (n % 2 == 1) tmp = tmp * a;
return tmp;
}
vector<ll> cc;
void solve() {

for (int i = 0; i<2; i++) {
    a.push_back(cc);
    r.push_back(cc);
    rr.push_back(cc);
    for(int j = 0; j<2; j++) {
        a[i].push_back(1);
        r[i].push_back(0);
        rr[i].push_back(0);
    }
}
a[0][0] = 0;
cin>>n>>ii>>MOD;
r = power(ii);
aa[0] = r[1][0];
aa[1] = r[1][1];
for (int i = 2; i<n; i++) aa[i] = (aa[i-2] + aa[i-1]) % MOD;
memset(fr, -1, sizeof fr);
long long sum = 0;
fr[0] = 0;
for (int i = 0; i<n; i++) {
    sum = (sum + aa[i])%MOD;
    if (fr[sum] > -1) {
        cout<<i - fr[sum] + 1<<' ';
        for(int j = fr[sum]; j<=i; j++) cout<<j + ii<<' ';
        cout<<endl;
        return;
    }
    fr[sum] = i + 1;
}

int main() {
//freopen("FIBSEQ.inp","r",stdin);
//freopen("FIBSEQ.out","w",stdout);
ios::sync_with_stdio(false);
cin.tie(0);cout.tie(0);
int test;
cin>>test;
while (test--) {
    solve();
}
}

```

Link Test và code mẫu:

<https://drive.google.com/drive/folders/1TB7Yqw49NV-9TCbeVzUGlfBgltZcg0I-?usp=sharing>

Bài 4: Mining – mức độ khó

<https://www.hackerrank.com/contests/world-codesprint-5/challenges/mining>

Ở đất nước Yên Bình có N mỏ vàng dọc theo bờ của dòng sông và mỏ vàng thứ i có thể khai thác được W_i tấn vàng. Để phục vụ khai thác vàng, người ta cần hợp nhất và phân phối lại các mỏ vàng sao cho có đúng K đồng vàng để di chuyển vàng khai thác, ở các mỏ vàng có thể di chuyển được bằng xe tải theo quy tắc sau:

- ✓ Có thể di chuyển vàng hai mỏ vàng i, j bất kì cho nhau với điều kiện ($0 < i < j \leq N$).
- ✓ Khi di chuyển số vàng từ vị trí i sang vị trí j hoặc là di chuyển hết số vàng qua j hoặc là không di chuyển vàng tại i .
- ✓ Để di chuyển W tấn vàng từ vị trí x_i đến vị trí x_j thì phải mất chi phí $|x_i - x_j| * W$.

Yêu cầu: cho N, K và số lượng vàng được sản xuất tại mỗi mỏ. Tính chi phí bé nhất để di chuyển số vàng.

Dữ liệu vào: Mining.inp có cấu trúc sau:

- Dòng 1 ghi hai số N, K .
- N dòng tiếp theo mỗi dòng ghi hai giá trị x_i và w_i tương ứng với vị trí và số lượng vàng khai thác được tại mỏ thứ i .

Kết quả ra: ghi vào file Mining.out giá trị bé nhất để hợp nhất số vàng khai thác được di chuyển về K đồng vàng.

Ví dụ:

Mining.inp	Mining.out	Giải thích
3 1 20 1 30 1 40 1	20	Di chuyển vàng về đồng 2
6 2 10 15 12 17 16 18 18 13 30 10 32 1	182	Chuyển vàng ở vị trí 1,2,3,4 về đồng 2; vị trí 5,6 về đồng 5.

Ràng buộc: $1 \leq K \leq N \leq 5000, 0 \leq X_i, W_i \leq 10^6$.

Hướng dẫn thuật toán:

Ta dễ dàng tính được bằng công thức quy hoạch động sau:

$$F[i, j] = \min(F[k, j-1] + G[k+1, i]) \text{ với } k \leq i.$$

Trong đó:

$F[i, j]$ là chi phí bé nhất khi lấy vàng ở i mỏ vàng đã khai thác được khi chuyển thành j đồng để xe đến lấy.

$G(i, j)$ chi phí khi chuyển các mỏ vàng $i + 1, i + 2, \dots, j$ về một vị trí thuộc $[i..j]$.

Khi đó độ phức tạp của bài toán $O(k \cdot n^3)$ thì với ràng buộc dữ liệu như trên thì bài toán bị timeout.

Để giải bài toán này ta kết hợp kĩ thuật chia để trị như sau:

Bước 1: ta tính chi phí nhỏ nhất khi di chuyển vàng khai thác được từ mỏ thứ i đến mỏ thứ j về một vị trí nào trong đoạn $[i..j]$.

```

void prepare() {
    for (int i = 1; i <= n; ++i) {
        long long hsum = 0, sum = 0;
        long long cur_cost = (long long)x[i - 1] * w[i - 1];
        for (int j = i, k = i - 1; j <= n; ++j) {
            sum += w[j];
            cur_cost += (long long)x[j] * w[j];
            while (hsum * 2 < sum) {
                cur_cost -= (long long)x[k] * w[k];
                hsum += w[++k];
                cur_cost -= (long long)x[k] * w[k];
            }
            long long lef = hsum - w[k];
            long long rig = sum - hsum;
            cost[i][j] = cur_cost + (lef - rig) * x[k];
        }
    }
}

```

Bước 2: sử dụng kỹ thuật chia để trị để tính chi phí bé nhất khi chuyển vàng từ n mỏ vàng thành k đống vàng: (Có thể chứng minh tương tự bài SEQPART)

```

for (int i = 1; i <= k; ++i) {
    divide(f[i - 1], f[i], i, n, i - 1, n - 1);
} với độ phức tạp là O(k.logn).

```

Thủ tục chia để trị được mô tả như sau:

```

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);
}

```

Khi đó chi phí tối thiểu là $F[k][n]$. Độ phức tạp của bài toán là $O((k*n)*logn)$.

Code mẫu:

```

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

```

```

const int maxn = 5005;
int n, k;
int w[maxn], x[maxn];

long long cost[maxn][maxn];

void prepare() {
    for (int i = 1; i <= n; ++i) {
        long long hsum = 0, sum = 0;
        long long cur_cost = (long long)x[i - 1] * w[i - 1];
        for (int j = i, k = i - 1; j <= n; ++j) {
            sum += w[j];
            cur_cost += (long long)x[j] * w[j];
            while (hsum * 2 < sum) {
                cur_cost -= (long long)x[k] * w[k];
                hsum += w[++k];
                cur_cost -= (long long)x[k] * w[k];
            }
            long long lef = hsum - w[k];
            long long rig = sum - hsum;
            cost[i][j] = cur_cost + (lef - rig) * x[k];
        }
    }
}

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", &n, &k);
for (int i = 1; i <= n; ++i) {
    scanf("%d%d", x + i, w + i);
}
prepare();
memset(f, 0x3f, sizeof(f));
f[0][0] = 0;
for (int i = 1; i <= k; ++i) {
    divide(f[i - 1], f[i], i, n, i - 1, n - 1);
}
printf("%lld\n", f[k][n]);
return 0;
}

```

Link Test và code mẫu:

<https://drive.google.com/drive/folders/1Oxz-clwdSSnVAYgzAphoWH2w1c0shf26?usp=sharing>

Bài 5: F. Yet Another Minimization Problem – Mức độ khó

Link: <https://codeforces.com/problemset/problem/868/F>

Cho dãy số nguyên A_1, A_2, \dots, A_n . Giá trị của một đoạn trên dãy số chính là số cặp trên đoạn đó có giá trị bằng nhau.

Yêu cầu:: Chia dãy số A_1, A_2, \dots, A_n thành K đoạn không giao nhau sao cho tổng giá trị của K đoạn đạt giá trị bé nhất. mỗi phần tử trong dãy chỉ nằm duy nhất một đoạn con nào đó trong K đoạn.

Dữ liệu vào:: tệp văn bản YAMP.inp có cấu trúc sau:

- Dòng 1 ghi hai số nguyên dương N, K ($2 \leq N \leq 10^5$, $2 \leq K \leq \min(K, 20)$).
- Dòng tiếp theo ghi dãy A_1, A_2, \dots, A_n trong đó ($1 \leq A_i \leq N$).

Kết quả:: ghi vào tệp văn bản YAMP một giá trị duy nhất là tổng giá trị K đoạn con theo yêu cầu đề bài.

Ví dụ:

YAMP.INP	YAMP.OUT
7 3 1 1 3 3 3 2 1	1
10 2 1 2 1 2 1 2 1 2	8

Hướng dẫn thuật toán:

Bài toán có thể dễ dàng xử lý bằng phương pháp quy hoạch với $O(KN^2)$ như sau:

Gọi $F[i,j]$ là giá trị nhỏ nhất của j phần tử và i đoạn ta có

$F[i,j] = \min_{j' < j} F[i-1,j'] + \text{cost}(j', j)$. với mỗi $j' < j$ ta tìm vị trí $j' < j$ sao cho $F[i-1,j'] + \text{cost}(j', j)$ đạt giá trị bé nhất. trong đó $F[i-1,j']$ là tổng giá trị bé nhất của $i-1$ đoạn với j' phần tử. và $\text{cost}(j', j)$ giá trị đoạn còn lại. Để tối ưu hóa bài toán ta sử dụng kỹ thuật chia để trị:

Giả sử có đoạn $[l, r]$ chúng ta biết rằng $P(j) \in [L, R]$ với mỗi $j \in [l, r]$ ta tìm trung điểm m của đoạn $[l, r]$ và tìm $P(m) \in [L, R]$. khi đó ta đẽ quy hai khoảng còn lại $[l, m-1]$ xác định $P(j) \in [L, M]$ và $[m+1, r]$ xác định $P(j) \in [M, R]$. Khi đó đẽ xác định giá trị của một đoạn mất $O(N \log N)$. Có thể dựa vào cách chứng minh bài **Famous Pagoda** để chứng minh tính đúng của thuật toán này.

Thuật toán trình bày như sau:

void solve(int l, int r, int L, int R, int k) {

```

if(l>r) return;
int mid=(l+r)>>1,p=0;
for(int i=min(R,mid-1);i>=L;i--) {
    calc(i+1,mid);
    if(f[i][k-1]+ans<f[mid][k]) f[mid][k]=f[i][k-1]+ans,p=i;
}
solve(l,mid-1,L,p,k);solve(mid+1,r,p,R,k);
}

```

để tính Calc(i+1, mid) ta thực hiện :

```

void calc(int L,int R) {
    while(r<R) ans+=cnt[a[++r]]++;
    while(l>L) ans+=cnt[a[--l]]++;
    while(r>R) ans---cnt[a[r--]];
    while(l<L) ans---cnt[a[l++]];
}

```

Bài toán chia dãy thành k đoạn không giao nhau nên có chi phí thực hiện là O(KNlogN)

Code mẫu:

```

#include<bits/stdc++.h>
using namespace std;
const int N=1e5+10;
typedef long long ll;
int n,k,a[N],cnt[N],l=1,r=0;
ll ans=0,f[N][30];
void calc(int L,int R) {
    while(r<R) ans+=cnt[a[++r]]++;
    while(l>L) ans+=cnt[a[--l]]++;
    while(r>R) ans---cnt[a[r--]];
    while(l<L) ans---cnt[a[l++]];
}
void solve(int l,int r,int L,int R,int k) {
    if(l>r) return;
    int mid=(l+r)>>1,p=0;
    for(int i=min(R,mid-1);i>=L;i--) {
        calc(i+1,mid);
        if(f[i][k-1]+ans<f[mid][k]) f[mid][k]=f[i][k-1]+ans,p=i;
    }
    solve(l,mid-1,L,p,k);solve(mid+1,r,p,R,k);
}
int main() {
    cin>>n>>k;
    memset(f,0x3f,sizeof(f));
    for(int i=1;i<=n;i++) scanf("%d",a+i),ans+=cnt[a[i]]++,f[i][1]=ans;
    memset(cnt,0,sizeof(cnt));ans=0;
    for(int i=2;i<=k;i++) solve(1,n,1,n,i);
    printf("%lld\n",f[n][k]);
    return 0;
}

```

Link Test và code mẫu:

<https://drive.google.com/drive/folders/1fyCf5zFMtA2f12eeLJVaRTdr0WRYozLd?usp=sharing>

4. Bài tập tham khảo

- <https://vn.spoj.com/problems/VRATF/>
- <https://vn.spoj.com/problems/MYSTERY/>
- <https://www.hackerrank.com/challenges/turn-off-the-lights/>
- <https://codeforces.com/problemset/problem/1175/G>
- <https://codeforces.com/problemset/problem/1156/D>
- <https://codeforces.com/problemset/problem/1155/D>

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. 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. Trong chuyên đề này tôi trình bày ý tưởng của bài toán tối ưu sử dụng hai kỹ thuật lập trình thường dùng là kỹ thuật quy hoạch động kết hợp với chia để trị để giải quyết các bài toán tối ưu. Mặc dù số lượng bài tập chưa phong phú đa dạng nhưng đây là những dạng thường gặp trong các kì thi học sinh cấp quốc gia và kì thi lập trình ACM ICPC (FAMOUS PAGOGA, FIBOSEQ). Hi vọng trong thời gian tới sẽ có những bài tập nhiều hơn nữa về các bài toán tối ưu.

6. Tài liệu tham khảo

- Sách giáo khoa chuyên tin tập 1,2,3 – nhà xuất bản giáo dục năm 2009.
- Giải thuật và lập trình của TS. Lê Minh Hoàng ĐHSP Hà Nội.
- <http://vnoi.info/wiki/Home>
- <https://vn.spoj.com/>
- <https://www.hackerrank.com/>
- <https://codeforces.com/>
- <https://www.geeksforgeeks.org/>