HƯỚNG DẪN GIẢI BÀI TẬP PHẦN QUY HOẠCH ĐỘNG BAO LỒI

BÀI 1. PHÂN NHÓM

Ví dụ:

Input	Output
4 100 1 15 15 20 5 1 100	500

Giải thích: cách phân nhóm thích hợp là (1), (2,3) và (4)

Thuật toán

Nhận xét 1

Nếu tồn tại hai cặp số (x_1,y_1) và (x_2,y_2) mà $x_1>x_2$ và $y_1>y_2$ thì ta nói cặp số $(x_2\ y_2)$ là không tiềm năng và có thể loại bỏ mà không cần quan tâm đến nó. Bởi vì ta có thể cho nó vào cùng nhóm với cặp đầu tiên mà không làm tồi đi kết quả.

Như vậy ta có thể sắp xếp lại các cặp số tăng dần theo x, sau đó sử dụng stack để loại bỏ đi những cặp số không tiềm năng, cuối cùng còn lại một dãy các cặp với x tăng dần và y giảm dần. Từ đây ta chỉ cần giải bài toán cho dãy các cặp số này.

Nhận xét 2

Nếu ta có hai cặp số (x_1,y_1) và (x_2,y_2) thuộc cùng một nhóm, thì ta có thể thêm vào nhóm đấy các cặp số (x,y) mà $x_1\le x\le x_2$ và $y_1\ge y\ge y_2$ mà không làm tồi đi

kết quả. Như vậy có nhận xét: các nhóm được phân hoạch gồm các phần tử liên tiếp nhau.

Quy hoạch động

Với hai nhận xét trên, ta đã có thể có được thuật toán QHĐ đơn giản với độ phức tạp đa thức. Gọi F(i) là chi phí nhỏ nhất để phân nhóm các phần tử có chỉ số không quá ii. Công thức truy hồi:

$F(i)=\min[F(j)+x_i*y_j]$ với $0\leq j<i0\leq j<i$

Có thể cài đặt thuật toán trên với độ phức tạp $O(N^2)$ tuy nhiên như vậy vẫn chưa đủ tốt với giới hạn của đề bài.

Áp dụng bao lồi

Đặt $y_j=a$, $x_i=x$, và $F(j)=b$. Rõ ràng ta cần cực tiểu hóa một hàm bậc nhất $y=a*x+b$ bằng việc chọn j hợp lí. Đồng thời trong bài toán này thì hệ số góc a của các đường thẳng là giảm dần, như vậy có thể áp dụng trực tiếp convex hull trick. Để ý một tí là các truy vấn (các giá trị x) là tăng dần, nên ta không cần phải tìm kiếm nhị phân mà có thể tịnh tiến để tìm kết quả. Độ phức tạp cho phần QHĐ này là O(N).

Code tham khảo:

```
#include <stdio.h>

#include <vector>

#include <iostream>

#include <algorithm>


using namespace std;
```

```
#define long long long
#define f1(i,n) for (int i=1;i<=n;i++)
#define f0(i,n) for (int i=0;i<n;i++)
typedef pair<int, int> ii;
typedef pair<long, long> ll;
#define X first
#define Y second
#define  N 1000006
int n;
ii a[N];
bool useless(ll a, ll b, ll c)
{
    return (b.Y-a.Y)*(a.X-c.X)>=(c.Y-a.Y)*(a.X-b.X);
}
long h(long x, ll d)
{
    return x*d.X+d.Y;
}
int main()
{
    freopen("bai2.inp","r",stdin);
    freopen("bai2.out","w",stdout);
```

```
    scanf("%d",&n);
    f1(i,n) scanf("%d%d",&a[i].X,&a[i].Y);
    sort (a+1,a+n+1);
    vector <ii> b;
    f1(i,n)
    {
        while (b.size()&& b.back().Y<=a[i].Y)
b.pop_back();
        b.push_back(a[i]);
    }
    vector <ll> d;
    long Lastf=0;
    int Best=0;
    f0(i,b.size())
    {
        d.push_back(ll(b[i].Y,Lastf));
        #define sz d.size()
        while (sz>=3 && useless(d[sz-3],d[sz-2],d[sz-1]))
        {
            if (Best>=sz-2) Best--;
            d.erase(d.end()-2);
        }
```

```
        while (Best+1<d.size() &&

h(b[i].X,d[Best+1])<=h(b[i].X,d[Best]))Best++;
        Lastf=h(b[i].X,d[Best]);
    }
    cout << Lastf;
    return 0;
}
```

Như vậy với một bài toán chỉ cần ta tìm ra công thức QHĐ mà có thể biểu diễn dưới dạng nhị thức bậc nhất, đồng thời hệ số góc có thể tăng/giảm dần, thì có thể áp dụng kỹ thuật trên để tối ưu.

BÀI 2. HARBINGERS

Ví dụ

Input	Output
5 1 2 20 2 3 12 2 4 1 4 5 3 26 9 1 10 500 2 2 30	206 321 542 328

Giới hạn

- $3 \le N \le 100000$
- $0 \le Si, Vi \le 10^9$
- Độ dài mỗi con đường không vượt quá 1000010000

Thuật toán QHĐ

Gọi F(i) là thời gian ít nhất để truyền tin từ thành phố thứ ii đến thủ đô, ta có công thức truy hồi:

$$F(i)=\min[F(j)+dist(j,i)*Vi+Si]$$

với j là một nút trên đường từ thành phố i đến thành phố 1. Trong đó dist(j,i) là khoảng cách giữa 2 thành phố i và j, có thể tính trong O(1) sử dụng mảng cộng dồn D[] với D[i] là khoảng cách từ thành phố i tới thủ đô. Thuật toán này có thể dễ dàng cài đặt với độ phức tạp là $O(N^2)$

Áp dụng bao lồi

Công thức truy hồi có thể viết lại thành

$$F(i)=\min[F(j)-Dj*Vi+Di*Vi+Si]$$

Khi ta tính F(i)F(i), thì giá trị $Di*Vi+Si$ là hằng số với mọi j, vì vậy

$$F(i)=\min[F(j)-Dj*Vi]+Di*Vi+Si$$

Có thể thấy rằng ta cần tìm giá trị nhỏ nhất của hàm bậc nhất $y=-Dj*x+F(j)$

Trong trường hợp tổng quát, ta cần một cấu trúc dữ liệu cho phép xử lí hai thao tác:

- Khi DFS xuống một nút con, ta cần thêm một đường thẳng.
- Khi quá trình DFS tính F[]F[] cho gốc cây con đã hoàn tất, ta cần xóa một đường thẳng, trả cấu trúc dữ liệu về trạng thái ban đầu.

Các thao tác này có thể được thực hiện hiệu quả trong O(logN). Cụ thể ta sẽ biểu diễn stack bằng một mảng cũng một biến size (kích thước stack). Khi thêm một đường thẳng vào, ta sẽ tìm kiếm nhị phân vị trí mới của nó, rồi chỉnh sửa

biến size cho phù hợp, chú ý là sẽ có tối đa một đường thẳng bị ghi đè, nên ta chỉ cần lưu lại nó. Khi cần trả về trạng thái ban đầu, ta chỉ cần chỉnh sửa lại biến size đồng thời ghi lại đường thẳng đã bị ghi đè trước đó. Để quản lí lịch sử các thao tác ta sử dụng một vector lưu lại chúng. Độ phức tạp cho toàn bộ thuật toán là O(NlogN)

Code tham khảo:

```
#include <bits/stdc++.h>
#define X first
#define Y second
const int N = 100005;
const long long INF = (long long)1e18;
using namespace std;
typedef pair<int, int> Line;
struct operation {
    int pos, top;
    Line overwrite;
    operation(int _p, int _t, Line _o) {
        pos = _p; top = _t; overwrite = _o;
    }
};
vector<operation> undoLst;
```

```
Line lines[N];

int n, top;



long long eval(Line line, long long x) {return line.X *
x + line.Y;}

bool bad(Line a, Line b, Line c)

    {return (double)(b.Y - a.Y) / (a.X - b.X) >=
(double)(c.Y - a.Y) / (a.X - c.X);}



long long getMin(long long coord) {

    int l = 0, r = top - 1; long long ans =
eval(lines[l], coord);

    while (l < r) {

        int mid = l + r >> 1;

        long long x = eval(lines[mid], coord);

        long long y = eval(lines[mid + 1], coord);

        if (x > y) l = mid + 1; else r = mid;

        ans = min(ans, min(x, y));

    }

    return ans;

}



bool insertLine(Line newLine) {
```

```
    int l = 1, r = top - 1, k = top;
    while (l <= r) {
        int mid = l + r >> 1;
        if (bad(lines[mid - 1], lines[mid], newLine)) {
            k = mid; r = mid - 1;
        }
        else l = mid + 1;
    }
    undoLst.push_back(operation(k, top, lines[k]));
    top = k + 1;
    lines[k] = newLine;
    return 1;
}

void undo() {
    operation ope = undoLst.back(); undoLst.pop_back();
    top = ope.top; lines[ope.pos] = ope.overwrite;
}

long long f[N], S[N], V[N], d[N];
vector<Line> a[N];
```

```
void dfs(int u, int par) {
    if (u > 1)
        f[u] = getMin(V[u]) + S[u] + V[u] * d[u];
    insertLine(make_pair(-d[u], f[u]));
    for (vector<Line>::iterator it = a[u].begin(); it
!= a[u].end(); ++it) {
        int v = it->X;
        int uv = it->Y;
        if (v == par) continue;
        d[v] = d[u] + uv;
        dfs(v, u);
    }
    undo();
}


int main() {
    ios::sync_with_stdio(0); cin.tie(0);
    cin >> n;
    int u, v, c;
    for (int i = 1; i < n; ++i) {
        cin >> u >> v >> c;
```

```
        a[u].push_back(make_pair(v, c));
        a[v].push_back(make_pair(u, c));
    }
    for (int i = 2; i <= n; ++i) cin >> S[i] >> V[i];
    dfs(1, 0);
    for (int i = 2; i <= n; ++i) cout << f[i] << ' ';
    return 0;
}
```

BÀI 3. ACQUIRE

Solution:

*Ta có các nhận xét sau:

- Tồn tại các hình chữ nhật không quan trọng: Giả sử tồn tại hai hình chữ nhật A và B mà mà cả chiều dài và chiều rộng của hình B đều bé hơn hình A thì ta có thể nói hình B là không quan trọng vì ta có thể để hình B chung với hình A từ đó chi phí của hình B không còn quan trọng. Sau khi đã loại hết tất cả hình không quan trọng đi và sắp xếp lại các hình theo chiều dài giảm dần thì chiều rộng các hình đã sắp xếp sẽ theo chiều tăng

- Đoạn liên tiếp:Sau khi sắp xếp, ta có thể hình dung được rằng nếu chúng ta chọn hai hình chữ nhật ở vị trí i và ở vị trí j thì ta có thể chọn tất cả hình chữ nhật từ $i+1$đến $j-1$ mà không tốn chi phí nào cả. Vậy ta có thể thấy rằng cách phân hoạch tối ưu là một cách phân dãy thành các đoạn liên tiếp và chi phí của một đoạn là bằng tích của chiều dài của hình chữ nhật đầu tiên và chiều rộng của hình chữ nhật cuối cùng.

* Giải bài toán theo phương pháp quy hoạch động bình thường: bài toán trở về bài toán phân dãy sao cho tổng chi phí của các dãy là tối ưu. Đây là một dạng bài quy hoạch động hay gặp và chúng ta có thể dễ dàng nghĩ ra thuật toán $O(N^2)$ như mã giả phía dưới. (Giả sử các hình đã được sắp xếp và bỏ đi những hình chữ nhật không quan trọng)

```
input N
for i ∈ [1..N]
     input rect[i].h
     input rect[i].w
let cost[0] = 0
for i ∈ [1..N]
     let cost[i] = ∞
     for j ∈ [0..i-1]
         cost[i] = min(cost[i],cost[j]+rect[i].h*rect[j+1].w)
print cost[N]
```

Ở trên cost[k] lưu lại chi phí cực tiểu để lấy được k hình chữ nhật đầu tiên. Hiển nhiên, cost[0]=0. Để tính toán được cost[i] với i khác 0, ta có tính tổng chi phí để lấy được các tập trước và cộng nó với chi phí của tập cuối cùng(có chứa i). Chi phí của một tập có thể dễ dàng tính bằng cách lấy tích của chiều dài hình chữ nhật đầu tiên và chiều rộng của hình chữ nhật cuối cùng.

Vậy ta có min(cost[i],cost[j]+rect[i].h*rect[j+1].w) với j là hình chữ nhật đầu tiên của tập cuối cùng. Với N=50000 thì thuật toán $O(N^2)$ này là quá chậm.

*Để cải tiến thuật toán trên, ta sử dụng kỹ thuật bao lồi như sau:

Với m_j=rect[j+1].w,b_j=cost[j],x=rect[i].h, với rect[x].h là chiều rộng của hình chữ nhật x và rect[x].w là chiều dài của hình chữ nhật x. Vậy thì bài toán trở về tìm hàm cực tiểu của $y=m_jx+b_j$ bằng cách tìm j tối ưu. Nó giống hoàn toàn bài toán chúng ta đã đề cập ở trên. Giả sử ta đã hoàn thành việc cài đặt cấu trúc đã đề cập ở trên chúng ta có thể có mã giả ở dưới đây:

```
input N
for i ∈ [1..N]
    input rect[i].h
    input rect[i].w
let E = empty lower envelope structure
let cost[0] = 0
add the line y=mx+b to E, where m=rect[1].w and b=cost[0] //b is zero
for i ∈ [1..N]
     cost[i] = E.query(rect[i].h)
    if i<N
        E.add(m=rect[i+1].w,b=cost[i])
print cost[N]
```

Rõ ràng các đường thẳng đã được sắp xếp giảm dần về độ lớn của hệ số góc do chúng ta đã sắp xếp các chiều dài giảm dần. Do mỗi truy vấn có thể thực hiện trong thời gian O(logN), ta có thể dễ dàng thấy thời gian thực hiện của cả bài toán là O(NlogN). Do các truy vấn của chúng ta cũng tăng dần (do chiều rộng đã được sắp xếp tăng tần) ta có thể thay thế việc chặt nhị phân bằng một con trỏ chạy song song với việc quy hoạch động đưa bước quy hoạch động còn O(N) nhưng tổng độ phức tạp vẫn là O(NlogN) do chi phí sắp xếp.

Code tham khảo

```
#include<bits/stdc++.h>
#include <iostream>
#include <vector>
#include <algorithm>
using namespace std;
int pointer; //Keeps track of the best line from previous query
vector<long long> M; //Holds the slopes of the lines in the envelope
vector<long long> B; //Holds the y-intercepts of the lines in the envelope
//Returns true if either line l1 or line l3 is always better than line l2
bool bad(int l1,int l2,int l3)
{
```

```
	/*
	intersection(l1,l2) has x-coordinate (b1-b2)/(m2-m1)
	intersection(l1,l3) has x-coordinate (b1-b3)/(m3-m1)
	set the former greater than the latter, and cross-multiply to
	eliminate division
	*/
	return (B[l3]-B[l1])*(M[l1]-M[l2])<(B[l2]-B[l1])*(M[l1]-M[l3]);
}
//Adds a new line (with lowest slope) to the structure
void add(long long m,long long b)
{
	//First, let's add it to the end
	M.push_back(m);
	B.push_back(b);
	//If the penultimate is now made irrelevant between the antepenultimate
	//and the ultimate, remove it. Repeat as many times as necessary
	while (M.size()>=3&&bad(M.size()-3,M.size()-2,M.size()-1))
	{
		M.erase(M.end()-2);
		B.erase(B.end()-2);
	}
}
//Returns the minimum y-coordinate of any intersection between a given vertical
//line and the lower envelope
long long query(long long x)
{
	//If we removed what was the best line for the previous query, then the
	//newly inserted line is now the best for that query
	if (pointer>=M.size())
		pointer=M.size()-1;
	//Any better line must be to the right, since query values are
	//non-decreasing
	while (pointer<M.size()-1&&
	  M[pointer+1]*x+B[pointer+1]<M[pointer]*x+B[pointer])
		pointer++;
	return M[pointer]*x+B[pointer];
}
```

```
int main()
{
	int M,N,i;
	pair<int,int> a[50000];
	pair<int,int> rect[50000];
	freopen("acquire.inp","r",stdin);
	freopen("acquire.out","w",stdout);
	scanf("%d",&M);
	for (i=0; i<M; i++)
		scanf("%d %d",&a[i].first,&a[i].second);
	//Sort first by height and then by width (arbitrary labels)
	sort(a,a+M);
	for (i=0,N=0; i<M; i++)
	{
		/*
		When we add a higher rectangle, any rectangles that are also
		equally thin or thinner become irrelevant, as they are
		completely contained within the higher one; remove as many
		as necessary
		*/
		while (N>0&&rect[N-1].second<=a[i].second)
			N--;
		rect[N++]=a[i]; //add the new rectangle
	}
	long long cost;
	add(rect[0].second,0);
	//initially, the best line could be any of the lines in the envelope,
	//that is, any line with index 0 or greater, so set pointer=0
	pointer=0;
	for (i=0; i<N; i++) //discussed in article
	{
		cost=query(rect[i].first);
		if (i < N-1)
			add(rect[i+1].second,cost);
	}
	printf("%lld\n",cost);
	return 0;
```

```
}
```

BÀI 4. COMMANDO

Ví dụ

Commando.inp	Commando.out
4 -1 10 -20 2 2 3 4	9

Ràng buộc

- 20% số test, $n \leq 1000$;
- 50% số test, $n \leq 10{,}000$;
- 100% số test, $n \leq 1000000$; $-5 \leq a \leq -1$, $|b| \leq 10{,}000{,}000$, $|c| \leq 10{,}000{,}000$, $1 \leq x_i \leq 100$

Phân tích

Subtask 1

O (n^3): Sử dụng Quy hoạch động, Gọi F(n) là trận chiến tối đa hiệu quả sau khi điều chỉnh. Chúng ta có công thức

$$f(n) = \max_{0 \leq i \leq n} \left\{ f(i) + g\left(\sum_{j=i+1}^{n} X_j\right) \right\}, g(x) = Ax^2 + Bx + c$$

Subtask 2

Định nghĩa rằng:

```
sum(i,j) = x[i] + x[i+1] + ... + x[j]
```

```
adjust(i,j) = a*sum(i,j)^2 + b*sum(i,j) + c
```

Ta có: $dp(n)=\max[dp(k)+adjust(k+1,n)]$ $0 \leq k<n$

Độ phức tạp: $O(n^2)$

Subtask 3

Biến đổi hàm "adjust" . Định nghĩa sum(1,x) là $\delta(x)$. Vậy với một số k bất kì ta có thể viết là:

- $dp(n)=dp(k)+a(\delta(n)-\delta(k))^2+b(\delta(n)-\delta(k))+c$
- $dp(n)=dp(k)+a(\delta(n)^2+\delta(k)^2-2\delta(n)\delta(k))+b(\delta(n)-\delta(k))+c$

- $dp(n)=(a\delta(n)^2+b\delta(n)+c)+dp(k)-2a\delta(n)\delta(k)+a\delta(k)^2-b\delta(k)$

Nếu:

- $z=\delta(n)$
- $m=-2a\delta(k)$
- $p=dp(k)+a\delta(k)^2-b\delta(k)$

Ta có thể thấy $mz+p$ là đại lượng mà chúng ta muốn tối ưu hóa bằng cách chọn k. $dp(n)$ sẽ bằng đại lượng đó cộng thêm với $a\delta(n)+b\delta(n)+c$ (độc lập so với k). Trong đó z cũng độc lập với k, m và p phụ thuộc vào k.

Ngược với bài "acquire" khi chúng ta phải tối thiểu hóa hàm quy hoạch động thì bài này chúng ta phải cực đại hóa nó. Chúng ta phải xây dựng một hình bao trên với các đường thẳng tăng dần về hệ số góc. Do đề bài đã cho $a<0$ hệ số góc của chúng ta tăng dần và luôn dương thỏa với điều kiện của cấu trúc.

Do dễ thấy $\delta(n)>\delta(n-1)$, giống như bài "acquire" các truy vấn chúng ta cũng tăng dần theo thứ tự do vậy chúng ta có thể khởi tạo một biến chạy để chạy song song khi làm quy hoạch động (bỏ được phần chặt nhị phân).

Chương trình tham khảo

```
#include <cstdio>
#include <algorithm>
#include <vector>
using namespace std;

const int MAXN = 1000001;
int N , a , b , c;
int x[MAXN];
long long sum[MAXN];
long long dp[MAXN];
// The convex hull trick code below was derived from acquire.cpp
vector <long long> M;
vector <long long> B;
bool bad(int l1,int l2,int l3)
{
	return (B[l3]-B[l1])*(M[l1]-M[l2])<(B[l2]-B[l1])*(M[l1]-M[l3]);
}
void add(long long m,long long b)
{
	M.push_back(m);
	B.push_back(b);
```

```
        while (M.size()>=3&&bad(M.size()-3,M.size()-2,M.size()-1))
        {
                M.erase(M.end()-2);
                B.erase(B.end()-2);
        }
}
int pointer;
long long query(long long x)
{
        if (pointer >=M.size())
                pointer=M.size()-1;
        while (pointer<M.size()-1&&
          M[pointer+1]*x+B[pointer+1]>M[pointer]*x+B[pointer])
                pointer++;
        return M[pointer]*x+B[pointer];
}

int main(){
        scanf("%d" , &N);
        scanf("%d %d %d" , &a , &b , &c);
        for(int n = 1 ; n <= N ; n++){
                scanf("%d" , &x[n]);
                sum[n] = sum[n - 1] + x[n];
        }
        dp[1] = a * x[1] * x[1] + b * x[1] + c;
        add(-2 * a * x[1] , dp[1] + a * x[1] * x[1] - b * x[1]);

        for(int n = 2 ; n <= N ; n++){
                dp[n] = a * sum[n] * sum[n] + b * sum[n] + c;
                dp[n] = max(dp[n] , b * sum[n] + a * sum[n] * sum[n] + c + query(sum[n]));
                add(-2 * a * sum[n] , dp[n] + a * sum[n] * sum[n] - b * sum[n]);
        }
        printf("%lld\n" , dp[N]);
        return 0;
}
```

Phân nhóm	20	
2		Harbingers
3		acquire
4		Commando
5		Cưa máy
6		Building
7		Rào Vườn
8		Mưa thiên thạch
9		Câu chuyện người lính
10		Beginner Free Contest 28 - NEAREST
11		Bộ ba điểm thẳng hàng
12		Mảnh đất tổ tiên
13		RIVER
14		Bắn máy bay
15		COCI 2016/2017 - Contest 6 - Gauss

Cải tiến quy hoạch động bằng bao lồi

CẢI TIẾN QUY HOẠCH ĐỘNG BẰNG BAO LỒI

MỤC LỤC

Giới thiệu

Mục đích của chuyên đề nhằm cải tiến thuật toán quy hoạch động với thời gian $O(n^2)$ xuống còn $O(n \log n)$ hoặc tốt hơn là xuống còn $O(n)$ cho một lớp các bài toán sử dụng kĩ thuật bao lồi - convex hull trick.

Bài toán 1.

Cho một bài toán với n phần tử khác nhau $1, 2, \dots, n$. Phần tử thứ i có hai tính chất $A[i], B[i]$. Hàm mục tiêu của bài toán này có thể được biểu diễn thông qua bảng quy hoạch động một chiều $C[1,2, \dots, n]$ thoả mãn hệ thức sau:

$$C[i] = \begin{cases} B[i] \ nếu \ i = 1 \\ \min\limits_{j<i}\{C[j] + A[i]B[j]\}, trong \ trường \ hợp \ khác \end{cases}$$

Tìm giá trị $C[n]$

Từ công thức quy hoạch động trên, ta có thể thiết kế giải thuật $O(n^2)$ để giải bài toán như sau:

```
function(A[1, 2, ..., n], B[1, 2, ..., n]);
C[1] ← B[1]
 for i ← 2 to n
tmp ← +∞
for j ← 1 to i − 1
tmp ← min(tmp, C[j] + A[i]B[j]
C[j] ← tmp

return C[n]
```

Mục tiêu tiếp theo: Cải tiến từ $O(n^2)$ xuống còn $O(n \ logn)$

Kĩ thuật bao lồi

Trước hết, xem xét bài toán hình học sau:

Bài toán 2.

Cho một tập n đường thẳng $D = \{d_1, d_2, \dots, d_n\}$ trong đó $(d_i)\ y_i = a_i x + b_i (a_i \geq 0)$ và k điểm trên trục Ox: $p_1, p_2, \dots, p_k$. Tìm $q_1, q_2, \dots, q_k$ trong đó mỗi giá trị q_ℓ được định nghĩa như sau: $q_\ell = \min_{1 \leq i \leq n} a_i \cdot p_\ell + b_i$

Với mỗi điểm p_ℓ, bằng cách duyệt qua tất cả các đường thẳng trong D, ta có thể tính được q_ℓ trong thời gian $O(n)$. Do đó, tổng thời gian để tìm tất cả các điểm $q_1, a_2, \dots, q_k$ là O_{kn}. Tuy nhiên ta cũng có thể tìm các điểm này nhanh hơn dựa theo bổ đề sau:

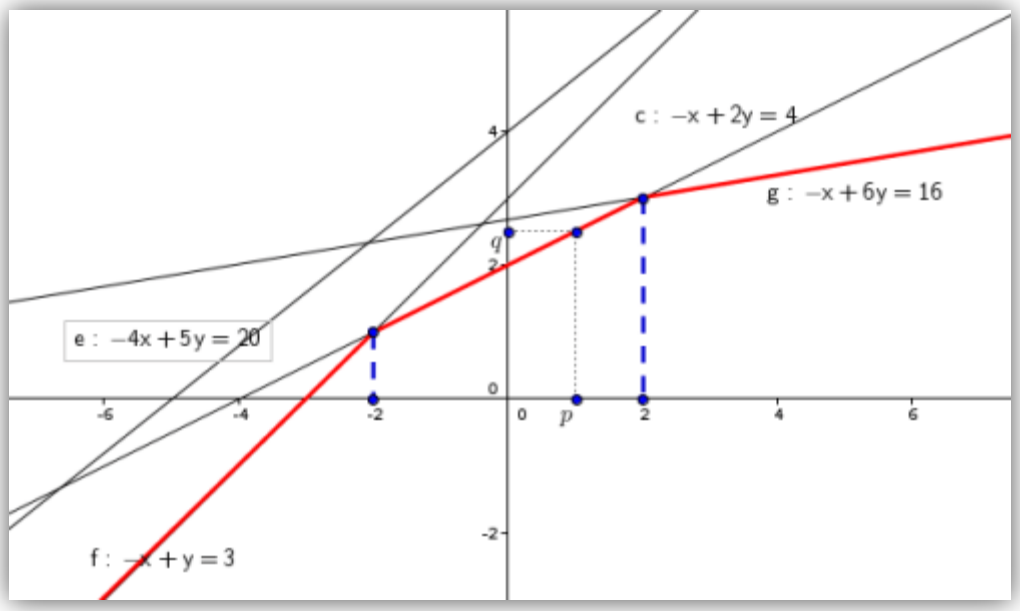
Bổ đề 1.

Tồn tại một thuật toán tìm tất cả các điểm $q_1, q_2, \dots, q_k$ trong thời gian $O((k+n)\log n)$.

Bổ đề 1 đúng cả với trường hợp tập các đường thẳng D có đường thẳng với $a_i < 0$. Tuy nhiên, trong các ứng dụng với bài toán quy hoạch động, các a_i luôn không âm. Ngoài ra, điều kiện $a_i \geq 0$ cũng làm cho việc phân tích thuật toán trở nên đơn giản hơn.

Trước hết chúng ta hãy xem ví dụ với $D = \{-x + y = 3, -x + 2y = 4, -x + 6y = 16, -4x + 5y = 20\}$.

Có thể thấy, với mỗi điểm $(p, 0) \in Ox$ giá trị nhỏ nhất tương ứng $(0, q)$ thoả mãn (p, q) thuộc phần gạch đỏ (hình trang 5). Phần gạch đỏ đó gọi là bao lồi (do đó kĩ thuật này còn gọi là kĩ thuật bao lồi). Như vậy, dựa vào hình vẽ trên, đường thẳng $-4x + 5y = 20$ trong D là dư thừa (tung độ của điểm có hoành độ p trên đường thẳng này luôn lớn hơn tung độ của điểm có cùng hoành độ trên một đường thẳng khác của D).



Dựa vào ví dụ trên, ta thấy nếu chúng ta có một cách biểu diễn bao lồi phù hợp, chúng ta có thể nhanh chóng xác định được tung độ q nhỏ nhất ứng với hoành độ p.

Ta có thể biểu diễn bao lồi bằng tập các interval và một đường thẳng tương ứng mỗi interval. Với ví dụ trên, tập các interval là $I_1 = [-\infty, -2], I_2 = [-2, 2], I_3 = [2, +\infty]$ và các đường thẳng tương ứng lần lượt là $-x + y = 3, -x + 2y = 4, -x + 6y = 16$. Bổ đề 2 sau đây cho ta biết số interval trong biểu diễn bao lồi luôn nhỏ hơn hoặc bằng số đường thẳng.

Bổ đề 2

Bao lồi của tập các đường thẳng $D = \{d_1, d_2, \dots, d_n\}$ cho trước có thể biểu diễn bằng m interval $I_1, I_2, \dots, I_m$ và m đường thẳng tương ứng với mỗi đoạn $d_1, d_2, \dots, d_m$ với $m \leq n$. Hơn nữa, ta có thể tìm được biểu diễn đó trong thời gian $O(nlogn)$.

Để đơn giản, ta giả sử trong D không có hai đường thẳng nào song song. Nếu có hai đường thẳng song song, ta chỉ cần thay đổi một chút trong giả mã dưới

đây mà không thay đổi thời gian tính của thuật toán. Thuật toán tìm bao lồi của D như sau :

$\underline{FindHull(d_1, d_2, \dots, d_n)}$:
  $sort\ \{d_1, d_2, \dots, d_n\}\ by\ decreasing\ order\ of\ slope$ $Stack\ S \leftarrow \emptyset$
    $S.push(d_1)$
    $I_1 \leftarrow [-\infty, +\infty]$
    $m \leftarrow 1$
    for $i \leftarrow 2\ to\ n$
      $d \leftarrow S.peek()$
      $x_p \leftarrow FindIntersectionX(d, d_i)$
      while $x_p \leq left(I_m)$
        $S.pop()$
        $m \leftarrow m - 1$
        $d \leftarrow S.peek()$
        $x_p \leftarrow FindIntersectionX(d_i, d)$
      $S.push(d_i)$
      $I_m \leftarrow [left(I_m), x_p]$
      $I_{m+1} \leftarrow [x_p, +\infty]$
      $associate\ I_{m+1}\ with\ d_i$
    $m \leftarrow m + 1$

Full Code

```
typedef struct{
    double a;
    double b;
  } double_pair;

 struct Stack_node {
    int lineId;
    struct Stack_node *next;
  };
typedef struct Stack_node stack_node;

stack_node *S;                // the stack
double_pair D[MAX_SIZE];      // the set of lines
double_pair I[MAX_SIZE];      // the set of intervals
double      Q[MAX_SIZE];      // the set of query points
int         ALines[MAX_SIZE];// the set of lines associated with
intervals

int find_hull(int n){
```

```
    qsort(D, n, sizeof(*D), slope_compare);
    S = malloc( sizeof(stack_node));
    S->lineId = -1; // head of the stack
    push(0);        // push d_1 to the stack;
    I[0].a = -(double)INFTY;
    I[0].b = (double)INFTY;
    int m = 0;
    ALines[m] = 0;
    int i = 0;
    stack_node *d;
    double x = 0.0;
    for(i = 1; i <n ;i++){
        d = peek();
        x = find_intersection_x(d->lineId, i);
        while(x <= I[m].a){  // found a redundant line
            pop();       // remove the redundant line
            m--;
            d = peek();
            x = find_intersection_x(d->lineId, i);
        }
        push(i);             // push d_i to the stack
        I[m].b = x;
        I[m+1].a = x;
        I[m+1].b = (double)INFTY;
        ALines[m+1] = i;
        m++;
    }

    return m+1;        // the number of intervals
}

double find_intersection_x(int x, int y){
    double_pair d_a = D[x];
    double_pair d_b = D[y];
    return (d_b.b - d_a.b)/(d_a.a - d_b.a);
}
```

Thủ tục $FindIntersectionX(d_1, d_2)$ tìm hoành độ giao điểm của hai đường thẳng d_1, d_2 (chi tiết coi như bài tập cho bạn đọc).

Ta có thể thấy vòng for của thuật toán tìm bao lồi có thời gian chạy $O(n)$ vì mỗi đường thẳng sẽ được xem xét tối đa hai lần: khi đưa vào $stack$ và khi lấy ra khỏi $stack$.

Nếu một đường thẳng bị lấy ra khỏi $stack$, nó sẽ không bao giờ được kiểm tra lại nữa. Do đó, thời gian của thuật toán tìm bao lồi chủ yếu dành để sắp xếp các đường thẳng theo slope và mất $O(nlogn)$.

Chứng minh Bổ đề 1

Giả sử chúng ta đã tìm được bao lồi của D, với mỗi điểm p_ℓ trên trục hoành, sử dụng tìm kiếm nhị phân để tìm interval $I \in I_1, I_2, \dots, I_m$ sao cho $p_\ell \in I$.

Giả sử $(d)y = ax + b$ là đường thẳng tương ứng với i, ta sẽ có $q_\ell = a \cdot p_\ell + b$.

Do đó, ta có thể tính q_ℓ trong thời gian $O(logn)$. Nếu ta có k điểm $p_1, p_2, \dots, p_k$, ta có thể tìm $q_1, q_2, \dots, q_k$ trong thời gian $O(klogn)$.

Chi tiết thuật toán như sau:

```
Query({d_1, d_2, ..., d_n}, {p_1, p_2, ..., p_k}):
  FindHull(d_1, d_2, ..., d_n)
  for i ← 1 to k
    I ← IntervalBinSearch({I_1, I_2, ..., I_m}, p_i)
    d ← the line associated with I
    output q_i = a · p_i + b          [[assuming (d)y = ax + b]]
```

```
IntervalBinSearch({I1, I2, ..., Im}, p):
  if m = 1
  return Im
  else
    ℓ = ⌊m/2⌋
    if p ∈ Iℓ
      return Iℓ
    else if p > right(Iℓ)
      return IntervalBinSearch({Iℓ + 1, ..., Im}, p)
    else
      return IntervalBinSearch({I1, ..., Iℓ − 1}, p)
```

CODE

```
void query(double points[], int k, int n){
    int m = find_hull(n);
    int i = 0, q = 0;
    for(i = 0; i < k; i++){
        q = interval_search(0, m-1, points[i]);
        Q[i] = D[ALines[q]].a*points[i] + D[ALines[q]].b;
    }
}


int interval_search(int x, int y, double p){
    if(y <= x){
        return x;
    } else {
        int mid = (x+y)/2;
        if((I[mid].a <= p) && (p <= I[mid].b)){
            return mid;
        } else if (p > I[mid].b){
            return interval_search(mid+1, y, p);
        } else {
            return interval_search(x, mid-1, p);
        }
    }
}
```

Quay trở lại bài toán quy hoạch động chúng ta đang xét. Nếu ta đặt $(d_j)y_j = B[j]x + C[j], 1 \leq j \leq n$ thì $C[i]$ chính là tung độ q nhỏ nhất thuộc một trong các đường thẳng $d_1, d_2, \ldots, d_{i-1}$ tương ứng với hoành độ $p = A[i]$.

Do đó, chúng ta có thể áp dụng ý tưởng của kĩ thuật bao lồi vào bài toán quy hoạch động này.

Trường hợp đặc biệt: Thuật toán $O(nlogn)$

Trước tiên chúng ta xét trường hợp mảng $B[1,2,\dots,n]$ thỏa mãn tính chất sau:

$$B[1] \geq B[2] \geq \cdots \geq B[n]$$

Giả sử $B[1] \geq B[2] \geq \cdots \geq B[n]$ tuy khá mạnh nhưng rất nhiêu bài toán chúng ta gặp sẽ thảo mãn điều kiện này. Với giả sử này, slope cuả các đường thẳng $d_1, d_2, \dots, d_n$ đã được sắp xếp theo thứ tự giảm dần. Do đó ta tiết kiệm được bước sắp xếp trong thuật tóan $FindHull(d1, d2, \dots, dn)$.

Để giả mã đơn giản, ta giả sử trong $B[i] \neq B[i+1]$ với mọi i. Nếu tồn tại i sao cho $B[i] = B[i+1]$, ta chỉ cần thay đổi một chút trong giả mã dưới đây mà không thay đổi thời gian tính của thuật toán. Thuật toán chi tiết như sau :

```
FastDynamic(A[1,2,...,n], B[1,2,...,n]):
  d1 ← B[1]x + C[1]
  Stack S ← ∅
  S.push(d1)
  I_1 ← [−∞, +∞]
  m ← 1
  for i ← 2 to n
    I ← IntervalBinSearch({I_1, I_2, ..., I_m}, A[i])
    d ← the line associated with I
    C[i] ← a · A[i] + b          [[assuming (d)y = ax + b]]
    di ← B[i]x + C[i]
    d ← S.peek()          [[examine the top element]]
    xp ← FindIntersectionX(d, d_i)
    while xp ≤ left(I_m)          [[found a redundant line]]
      S.pop()
      m ← m − 1
      d ← S.peek()
      xp ← FindIntersectionX(d_i, d)
    S.push(d_i)
    I_m ← [left(I_m), x_p]
    I_m + 1 ← [x_p, +∞]
    associate I_{m+1} with d_i
    m ← m + 1
  return C[n]
```

Trường hợp tổng quát

Trong trường hợp tổng quát, chúng ta có thể áp dụng thủ tục $IntervalBinSearch(\{I_1, I_2, \dots, I_m\}, A[i])$ để tìm kiếm interval chứa $A[i]$.
Do đó chúng ta mất $O(logn)$ cho việc tìm kiếm interval cho mỗi vòng lặp của thuật toán.
Vấn đề còn lại chỉ là làm sao để cập nhật bao lồi trong mỗi bước khi ta thêm đường thẳng $(d)B[i]x + C[i]$.
Ở đây chỉ đưa ra ý tưởng để thực hiện cập nhật bao lồi trong thời gian trung bình $O(logn)$ mỗi bước. Do đó, tổng thời gian của thuật toán là $O(nlogn)$.

Ý tưởng của thuật toán dựa trên thuật toán Sweep Line. Ta sẽ luôn luôn lưu các đường thẳng trong bao lồi theo thứ tự tăng dần của slope sau mỗi bước. Gọi $d_1, d_2, \dots, d_m$ là các đường thẳng trong bao lồi sau bước $i - 1$.

Để đơn giản ta giả sử không có hai đường thẳng nào song song. Trong trường hợp có hai đường song song, ta chỉ cần thay đổi thuật toán một chút.

Bây giờ ta có thêm đường thẳng $(di)y = a_i x + b_i$, để cập nhật bao lồi trước hết ta thực hiện tìm kiếm nhị phân để tìm ra đường thẳng $(dj)y = a_j y + b_j$ sao cho $a_j > a_i > a_j + 1$.

Trường hợp $j = m$, ta cập nhật như trong trường hợp đặc biệt. Do đó, ta có thể giả sử $j < m$.

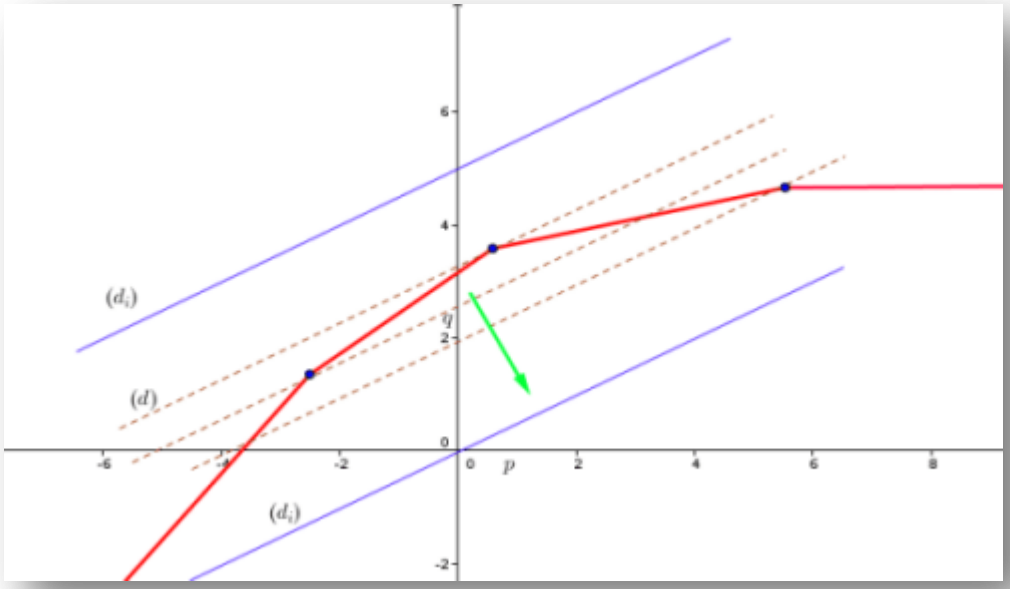
Gọi (x_p, y_p) là giao điểm của d_j và d_{j+1}.

Xét đường thẳng $(d)y = a_i x + b_i'$ có cùng slope a_i với d_i và đi qua (x_p, y_p). Nếu $b_i' \leq b_i$ (đường thẳng d_i có màu xanh trong hình tại trang 12), đường thẳng d_i là dư thừa và do đó, bao lồi không thay đổi khi ta thêm đường thẳng d_i.

Nếu $b_i' > b_i$, ta lần lượt "dịch" đường thẳng d về phía d_i cho đến khi d trùng với d_i.

Nếu có sự "thay đổi" của bao lồi (các đường thẳng màu nâu), ta cập nhật lại bao lồi.

Ta có thể nhận thấy trong trường hợp tồi nhất, ta phải dịch đường thẳng d $O(m)$ lần. Tuy nhiên, nếu ta dịch K lần, thì mỗi lần dịch ta sẽ "loại bớt" được K đường thẳng dư thừa và các đường thẳng này sẽ không được xét đến trong các vòng lặp tiếp theo. Do đó, thời gian trung bình mỗi bước lặp vẫn là $O(logn)$



Ví dụ áp dụng

Bài 1. Kalila and Dimna in the Logging Industry

(http://codeforces.com/contest/319/problem/C)

Tóm tắt đề bài: Có n cái cây $1,2,\dots,n$ trong đó cây thứ i có chiều dài $A[i] \in N$. Bạn phải cắt tất cả các cây thành các đoạn có chiều dài 1.

Bạn có một cái cưa máy (lởm), mỗi lần chỉ chặt được một đơn vị chiều dài, và mỗi lần sử dụng thì lại phải sạc pin. Chi phí để sạc pin phụ thuộc vào chi phí của cây đã bị cắt hoàn toàn (một cây bị cắt hoàn toàn có chiều dài 0).

Nếu chỉ số lớn nhất của cây bị cắt hoàn toàn là i thì chi phí sạc là $B[i]$, và khi bạn đã chọn một cây để cắt, bạn phải cắt nó hoàn toàn.

Ban đầu cái cưa máy được sạc đầy pin.

Giả sử $a_1 = 1 \le a_2 \le \cdots \le a_n$ và $b_1 \ge b_2 \ge \cdots \ge b_n = 0$.

Tìm một cách cưa cây với chi phí nhỏ nhất.

Hướng dẫn giải thuật

Ví dụ: $n = 6, A[1,2,\dots,6] = \{1,2,3,10,20,30\}, B[1,2,\dots,6] = \{6,5,4,3,2,0\}$

Nếu bạn chặt cây theo thứ tự $1 \to 2 \to 6 \to 3 \to 4 \to 5$ chi phí bạn phải trả là $2 \times 6 + 30 \times 5 + 3 \times 0 + 10 \times 0 + 20 \times 0 = 162$.

Nếu bạn chặt theo thứ tự: $1 \to 3 \to 6 \to 2 \to 4 \to 5$ chí phí bạn phải trả là $3 \times 6 + 30 \times 4 + 4 \times 0 + 10 \times 0 + 20 \times 0 = 138$

Từ ví dụ trên, ta có nhận xét sau: Chi phí để chặt các cây sau khi đã chặt cây thứ n là 0.

Do đó, bài toán trên có thể được quy về bài toán: tìm chi phí nhỏ nhất để chặt cây thứ n.

Gọi $OPT = \{1 = i_1, i_2, \dots, i_k\}$ là một thứ tự chặt cây tối ưu với $i_k = n$. Ta có:

$$1 = i_1 \le i_2 \le \cdots \le i_k = n$$

Phát triển quy hoạch động như sau:

Gọi $C[i]$ là chi phí nhỏ nhất để chặt cây thứ i "sử dụng" các cây $1, 2, \dots, i-1$. Ta có công thức sau:

$$C[i] = \begin{cases} B[i] \text{ nếu } i = 1 \\ \min\limits_{j<i}\{C[j] + A[i]B[j] + B[j]\}, \textit{trong trường hợp khác} \end{cases}$$

Dễ thấy, chi phí để chặt cây thứ n là $C[n]$.

Để ý kĩ sẽ thấy công thức quy hoạch động để tính $C[n]$ với hai mảng A, B là các mảng đã sắp xếp. Do đó, có thể giải bài toán này trong thời gian $O(n)$.

Code xem trang 14

```
#include<stdio.h>
#include<iostream>
```

```
#include<stdlib.h>
#include<math.h>
#include<stack>

#define min(x, y) (((x) < (y)) ? (x) : (y))
#define max(x, y) (((x) < (y)) ? (y) : (x))
#define is_equal(x,y) (((x) - (y) == 0) ? 1 : 0)
#define MAX_SIZE 100005
#define INFTY  1e15 + 7
#define NULL 0

using namespace std;

typedef struct{
    double a;
    double b;
  } double_pair;


stack<int> S;
double_pair I[MAX_SIZE];
long long           A[MAX_SIZE];
long long      B[MAX_SIZE];
long long C[MAX_SIZE];
int            ALines[MAX_SIZE];// the set of lines associated
with I

double find_intersection_x(int x, int y);
void readinput();
void fast_fats_dynamic(long long A[], long long B[],int n);
int interval_search(int x, int y, double p);
int n;



void readinput(){
     cin >> n;
     for(int i = 0 ; i < n; i++) cin>>A[i];
     for(int i = 0 ; i < n; i++) cin>>B[i];
}

void fast_fast_dynamic(long long A[], long long B[], int n){
     C[0] = B[0]; // d_1 = B[0]x + C[0]
     S.push(0); // push d_1 to the top of the stack;
     I[0].a = -(double)INFTY;
     I[0].b = (double)INFTY;
     int m = 0;
```

```
     ALines[m] = 0;
     int i = 0, q = 0, l =0;
     int d;
     double x = 0.0;
     int prev_interval = 0;
     for(i = 1; i <n ;i++){
        l = prev_interval-1; // the interval associated with
A[i-1]
        while(1){
          l++;
          if((I[l].a   <=   (double)A[i])   &&   ((double)A[i]<=
I[l].b)){
                q = l;
                break;
          }
        }

          C[i] = B[ALines[q]]*(A[i]-1) + C[ALines[q]] + B[i];
          d = S.top();    // examine the top element of the
stack
          x = find_intersection_x(d, i);
          while(x <= I[m].a){  // found a redundant line
               S.pop();           // remove the redundant line
               m--;
               d = S.top();
               x = find_intersection_x(d, i);
          }
          prev_interval = l;
        if(x < (double) (INFTY-1)){
            S.push(i);
            I[m].b = x;
            I[m+1].a = x;
            I[m+1].b = INFTY;
            if( l >= m){
                 if    ((I[m].a    <=    ((double)A[i]))    &&
(((double)A[i]) <= I[m].b)){
                     prev_interval = m;
                }else {
                     prev_interval = m+1;
                }
            }
            ALines[m+1] = i;
            m++;
        }
     }
cout<<C[n-1]<<endl;
}
```

```
int interval_search(int x, int y, double p){
     if(y <= x){
          return x;
     } else {
          int mid = (x+y)/2;
          if((I[mid].a <= p) && (p <= I[mid].b)){
               return mid;
          } else if (p > I[mid].b){
               return interval_search(mid+1, y, p);
          } else {
               return interval_search(x, mid-1, p);
          }
     }
}
double find_intersection_x(int x, int y){
     long long a1 = B[x], b1 = C[x];
    long long a2 = B[y], b2 = C[y];
    return ((double)(b2-b1))/((double)(a1 - a2));
}


int main(int argc, const char * argv[]){
     readinput();
     fast_fast_dynamic(A,B,n);
     return 0;
}
```

Bài 2. USACO Tháng 3 năm 2008 "Acquire"

Cho $N(N \leq 50000)$ hình chữ nhật khác nhau về hình dạng, mục tiêu của bài toán là phải lấy được tất cả hình chữ nhật. Một tập hình chữ nhật có thể thu được với chi phí bằng tích của chiều dài dài nhất và chiều rộng dài nhất. Chúng ta cần phân hoạch tập các hình chữ nhật này một cách khôn khéo sao cho tổng chi phí có thể được tối thiểu hóa và tính chi phí này. Hình chữ nhật không thể được xoay (đổi chiều dài và chiều rộng).

Hướng dẫn giải thuật bằng bao lồi

Với $m_j = rect[j+1].w, b_j = cost[j], x = rect[i].h$ với $rect[x].h$ là chiều rộng của hình chữ nhật x và $rect[x].w$ là chiều dài của hình chữ nhật x.

Vậy bài toán trở về tìm hàm cực tiểu của $y = m_j x + b_j$ bằng cách tìm j tối ưu.

```
input N
for i ∈ [1..N]
     input rect[i].h
     input rect[i].w
let E = empty lower envelope structure
let cost[0] = 0
add the line y=mx+b to E, where m=rect[1].w and b=cost[0] //b
is zero
for i ∈ [1..N]
     cost[i] = E.query(rect[i].h)
     if i<N
          E.add(m=rect[i+1].w,b=cost[i])
print cost[N]
```

Rõ ràng các đường thẳng đã được sắp xếp giảm dần về độ lớn của hệ số góc do chúng ta đã sắp xếp các chiều dài giảm dần. Do mỗi truy vấn có thể thực hiện trong thời gian $O(logN)$, ta có thể dễ dàng thấy thời gian thực hiện của cả bài toán là $O(NlogN)$. Do các truy vấn của chúng ta cũng tăng dần (do chiều rộng đã được sắp xếp tăng tần) ta có thể thay thế việc chặt nhị phân bằng một con trỏ chạy song song với việc quy hoạch động đưa bước quy hoạch động còn $O(N)$ nhưng tổng độ phức tạp vẫn là $O(NlogN)$ do chi phí sắp xếp.

Bài 3. Phân nhóm

Cho $n \leq 300000$ cặp số (x, y) $(1 \leq x, y \leq 1000000)$. Ta có thể nhóm một vài cặp số lại thành một nhóm.
Giả sử một nhóm gồm các cặp số thứ $a_1, a_2, \dots, a_m$ thì chi phí cho nhóm này sẽ là $max(x_{a_1}, x_{a_2}, \dots, x_{a_m}) * max(y_{a_1}, y_{a_2}, \dots, y_{a_m}$.
Yêu cầu: tìm cách phân nhóm có tổng chi phí bé nhất.
Input

- Dòng đầu tiên là số nguyên dương N.
- N dòng tiếp theo dòng thứ i gồm hai số xi và yi.

Output

- Gồm 1 số duy nhất là kết quả tìm được.

Ví dụ
Input
4
100 1
15 15
20 5
1 100

Output
500
Giải thích: cách phân nhóm thích hợp là (1), (2,3) và (4).

Hướng dẫn giải thuật.

Nhận xét 1

Nếu tồn tại hai cặp số $(x1, y1)$ và $(x2, y2)$ mà $x1 > x2$ và $y1 > y2$ thì ta nói cặp số $(x2, y2)$ là không tiềm năng, và có thể loại bỏ không cần quan tâm tới nó. Bởi vì ta có thể cho nó vào cùng nhóm với cặp đầu tiên mà không làm tồi đi kết quả.

Như vậy ta có thể sắp xếp lại các cặp số tăng dần theo x . Sau đó sử dụng stack để loại bỏ đi những cặp số không tiềm năng, cuối cùng còn lại một dãy các cặp với x tăng dần và y giảm dần. Từ đây ta chỉ cần giải bài toán cho dãy các cặp số này.

Nhận xét 2

Nếu ta có hai cặp số $(x1, y1)$ và $(x2, y2)$ thuộc cùng một nhóm, thì ta có thể thêm vào nhóm đấy các cặp số (x, y) mà $x1 \leq x \leq x2$ và $y1 \geq y \geq y2$ mà không làm tồi đi kết quả. Như vậy có nhận xét: các nhóm được phân hoạch gồm các phần tử liên tiếp nhau.

Quy hoạch động

Với hai nhận xét trên, ta đã có thể có được thuật toán quy hoạch động đơn giản với độ phức tạp đa thức. Gọi $F(i)$ là chi phí nhỏ nhất để phân nhóm các phần tử có chỉ số không quá I .

Công thức truy hồi: $F(i) = min[F(j) + xi * yj]$ $với$ $0 \leq j < i$

Có thể cài đặt thuật toán trên với độ phức tạp $O(N^2)$, tuy nhiên như vậy vẫn chưa đủ tốt với giới hạn của đề bài.

Áp dụng bao lồi

Đặt $yj = a$, $xi = x$, và $F(j) = b$. Rõ ràng ta cần cực tiểu hóa một hàm bậc nhất $y = a * x + b$ bằng việc chọn j hợp lí. Đồng thời trong bài toán này thì hệ số góc a của các đường thẳng là giảm dần, như vậy có thể áp dụng trực tiếp bao lồi.

Để ý là các truy vấn (các giá trị x) là tăng dần, nên ta không cần phải tìm kiếm nhị phân mà có thể tịnh tiến để tìm kết quả. Độ phức tạp cho phần quy hoạch động này là $O(N)$.

```
#include <bits/stdc++.h>
#define X first
#define Y second

const int N = 300005;

using namespace std;
typedef pair<long long, long long> Line;
```

```
int n;
pair<int, int> a[N];

long long eval(long long x, Line line) {
    return x * line.X + line.Y;
}

bool bad(Line d1, Line d2, Line d3) {
    return (d2.Y - d1.Y) * (d1.X - d3.X) >= (d3.Y - d1.Y) *
(d1.X - d2.X);
}

int main() {
    scanf("%d", &n);
    int i;
    for (i = 1; i <= n; i++) scanf("%d %d", &a[i].X, &a[i].Y);
    sort(a + 1, a + 1 + n);
    vector<pair<int, int> > b;
    for (i = 1; i <= n; i++) {
        while (b.size() && b.back().Y < a[i].Y) b.pop_back();
        b.push_back(a[i]);
    }
    vector<Line> d; long long last = 0; Line new_line; int best
= 0;
    for (i = 0; i < b.size(); i++) {
        new_line = Line(b[i].Y, last);
        while (d.size() >= 2 && bad(d[d.size() - 2], d[d.size()
- 1], new_line)) {
            if (best >= d.size() - 1) best--; d.pop_back();
        }
        d.push_back(new_line);
        while (best + 1 < d.size() &&
          eval(b[i].X, d[best]) >= eval(b[i].X, d[best + 1]))
best++;
        last = intersectX(b[i].X, d[best]);
    }
    cout << last << endl;
    return 0;
}
```

Một số bài tập tự luyện

1. http://lequydon.ntucoder.net/Problem/Details/4612
2. https://open.kattis.com/problems/joiningnetwork
3. https://vn.spoj.com/problems/RAOVUON/

Tài liệu tham khảo

1. http://codeforces.com/blog/entry/8219
2. http://www.giaithuatlaptrinh.com/?p=176
3. http://wcipeg.com/wiki/Convex_hull_trick
4. https://vnoi.info/wiki/translate/wcipeg/Convex-Hull-Trick
5. http://vnoi.info/wiki/algo/dp/Mot-so-ky-thuat-toi-uu-hoa-thuat-toan-Quy-Hoach-Dong

Một số bài tập tự luyện thêm QHĐ bao lồi trên Codeforces

http://codeforces.com/contest/311/problem/B

http://codeforces.com/contest/660/problem/F

http://lequydon.ntucoder.net/Problem/Details/4612

http://codeforces.com/contest/319/problem/C

QUY HOẠCH ĐỘNG VỊ TRÍ CẤU HÌNH

I. MỘT SỐ BÀI TOÁN ĐIỂN HÌNH:

1. Nhiphan1

Cho tập A gồm tất cả xâu nhị phân có độ dài N được sắp xếp theo thứ tự từ điển. Ví dụ với N=3 ta có tập A như sau:

Vị trí	Dãy nhị phân	Vị trí	Dãy nhị phân	Vị trí	Dãy nhị phân	Vị trí	Dãy nhị phân
1	000	9	1000	17	11000	25	11000
2	001	10	1001	18	11001	26	11001
3	010	11	1010	19	11010	27	11010
4	011	12	1011	20	11011	28	11011
5	100	13	1100	21	11100	29	11100
6	101	14	1101	22	11101	30	11101
7	110	15	1110	23	11110	31	11110
8	111	16	1111	24	11111	32	11111

Trong tập trên ta thấy

- Dãy nhị phân 110: ở vị trí số 7
- vị trí thứ 4 trong tập là dãy nhị phân: 011

Dữ liệu vào: Cho số N bất kỳ (n<=100) là độ dài dãy nhị phân sẽ xét sau:

- Cho dãy nhị phân S có N chữ số, hỏi S nằm ở vị trí thứ mấy?
- Cho ví trí thứ K ($k<10^{18}$), hỏi dãy nhị phân là bao nhiêu?

Dữ liệu vào ra:

Nhiphan1.inp	**Nhiphan1.out**
4 1101 8	14 0111

Với bài toán này ta có cách đơn giản: Vị trí của dãy nhị phân chính là giá trị thập phân tương ứng. Nhưng để làm quen với Quy hoạch động vị trí cấu hình ta sẽ làm theo cách sau:

Ta xây dựng mảng một chiều F như sau: F[i] là số lượng dãy nhị phân có độ dài i bít. Khi đó ta có mảng F sẽ được tính bằng quy hoạch động:

- F[0] = 1
- F[i] = 2* F[i-1] Với i=1..n

Hỏi dãy nhị phân S nằm vị trí nào?

Ta xem dãy S như sau: $S = S_1S_2S_3 \dots S_{length(s)}$

1. Ta nhận thấy nếu S[i] = 1 thì khi đó ta thấy có S[i-1] dãy nhị phân có vị trí bé hơn vị trí của S. *Ví dụ: S = '0100' ta có S_2 = 1 vậy thì ta có S[2] vị trí bé hơn, Tính A[2] = 4 dãy nhị phân có vị trí bé hơn S. đó là 000, 001, 010, 011. Mảng A như sau:*

0	1	2	3	4
1	2	4	8	16

Kết quả vị trí của xâu S trong tập đã xếp theo từ điển chính là vt+1=A[2]+1.

Hỏi dãy nhị phân nằm ở vị trí K?

Thực hiện ngược lại.

2. Nhiphan2

Một tập S gồm các dãy nhị phân N bit 0 và 1 trong đó không có hai bit 1 nào liền nhau. Ví dụ với N=5 thì S gồm các dãy 00000, 00001, 00101, … Tập S được sắp xếp theo chiều tăng dần của số nguyên tương ứng mà dãy bit biểu diễn.

Cho một số nguyên N (N<100) là độ dài số nhị phân.

- Cho biết số M, Hỏi dãy bít thứ M trong tập S có thứ tự là bao nhiêu?
- Ngược lại, Cho dãy bit R, Hỏi R có số thứ tự là bao nhiêu trong tập S.

Nhiphan2.inp	Nhiphan2.out
5 3 00001	000101 2

* **Phần chung:**

Tương tự xây dựng mảng A với ý nghĩa;

A[i] là số lượng dãy nhị phân có độ dài i bít (không có 2 bít 1 liền nhau)

Ta có:

A[0] = 1

A[1] = 2 Gồm: 0, 1

A[2] = 3 Gồm: 00, 01, 10

A[3] = 5 Gồm: 000, 001, 010, 100, 101

A[4] = 8 Gồm: 0000, 0001, 0010, 0100, 0101, 1000, 1001, 1010

…

Vậy A[i] = A[i-1] + A[i-2] Với i=2..N

3. Số hiệu hoán vị: SHHV

Xét tập A gồm tất cả các hoán vị của dãy số tự nhiên (1,2,..,n) (n<=20), các hoán vị được sắp xếp theo thứ thự từ điển.

Yêu cầu:

- Cho trước 1 hoán vị: Tìm số hiệu của hoán vị đó trong tập A
- Cho trước 1 số hiệu: Tìm hoán vị tương ứng.

Dữ liệu vào:

- Dòng 1: Dãy hoán vị S
- Dòng 2: Chứa số hiệu K

Kết quả: Lưu vào SHHV.out

- Dòng 1: q (là số hiệu của hoán vị S)
- Dòng 2: Dãy hoán vị tương ứng với số hiệu K

SHHV.inp	SHHV.out
213 4	2 231

*** Phần yêu cầu chung:**

Ta xây dựng mảng A với ý nghĩa:

A[i] là số lượng hoán vị có i chữ số.

Ta có:

A[1] = 1 Gồm: 1

A[2] = 2 Gồm: 12, 21

A[3] = 6 Gồm: 123, 132, 213, 231, 312, 321

A[4] = 24 Gồm: 1234, 1243, 1324, 1342, 1423, 1432… 4321

…

A[i] = i! hay F[i-1]*i

Vậy vt := vt + 2 * A[n – 1] = 12.

II. BÀI TẬP TỰ LUYỆN

2.1 Số hiệu tổ hợp:

Cho tập hợp A gồm N phần tử. Mỗi tập con gồm K (1 <= K <= N) phần tử của A được gọi là một tổ hợp chập K của N phần tử đã cho

Bài toán đặt ra là:

- Cho số hiệu của một tổ hợp chập K của N số nguyên dương đầu tiên, hãy tìm tổ hợp chập đó.

- Cho tổ hợp chập K của N số nguyên dương đầu tiên, hãy tính số hiệu của tổ hợp chập đó.

Input

Gồm 3 dòng có dạng như sau:

- Dòng 1: Ghi 2 số nguyên N, K (3 <= N <= 300)
- Dòng 2: Ghi số nguyên S
- Dòng 3: Gồm K số nguyên $B_1, B_2, ... B_K$ ($B_1 < B_2 < ... < B_K$)

Output

- Dòng 1: Ghi ra dãy số $A_1, A_2, ... A_K$ là tổ hợp chập K của N số nguyên dương đầu tiên có số hiệu S. Các số viết theo thứ tự tăng dần.
- Dòng 2: Ghi số hiệu của tổ hợp chập K: $B_1, B_2, ... B_K$.

Ví dụ:

SHTH.inp	**SHTH.out**
3 2 2 2 3	1 3 3

2.2 Liên lạc vũ trụ

Để liên lạc với tàu thăm dò tự động người ta chuẩn bị danh sách các thông báo, đánh số từ 1 trở đi và cài vào bộ nhớ trong của máy tính trên trạm thăm dò. Số lượng thông báo là 10^5. Trạm điều khiển mặt đất hoặc tàu thăm dò chỉ cần phát đi số thứ tự thay vì cho chuyển bằng hệ thống phát xung laser định hướng. Nhưng việc phát xung cần phải chuyển các số thứ tự này thành dãy nhị phân trước khi phát đi.

Em hãy lập trình cài vào máy phát việc chuyển đổi từ giá trị số thứ tự sang xâu nhị bit tương ứng cần phát đi.

Dữ liệu vào: impulse.inp

Dòng 1: Số lượng thông báo cần phát đi R (10^5)

Các dòng sau: Chứa các số nguyên dương (Số thứ tự thông báo), các số cách nhau ít nhất một dấu cách.

Kết quả:

Mỗi dòng là một xâu bit cần phát. Bỏ qua các số 0 trước 1 đầu tiên trong xâu, trừ trường hợp số cần phát là 1 thì kết quả được ghi là 0.

Ví dụ:

impulse.inp	impulse.out
6 1 2 3 4 5 100	0 1 10 11 100 100000110

2.3 Đánh số tập con:

Giả sử A là tập N số nguyên dương đầu tiên (N<=30). Với mỗi tập con B của A ta luôn viết các phần tử của B theo thứ tự tăng dần.

Ta xếp thứ tự các tập con này của A theo nguyên tắc: giả sử B và C là hai tập con, nếu có i sao cho $B_i <= C_i$. Trên cơ sở các tập con này ta đánh số các tập con từ 1 đến 2N, tập rỗng được đánh số 1.

Ví dụ: Với n = 3 ta có

Thứ tự	Tập con
1	
2	1
3	1 2
4	1 2 3
5	1 3
6	2
7	2 3
8	3

Yêu cầu:

- Cho một số hiệu S>1, tìm tập B có số hiệu này?
- Cho tập C không rỗng, tìm số hiệu của tập C

Dữ liệu vào: Subset.inp

- Dòng 1: Ghi N
- Dòng 2: S
- Dòng 3: Tập C

Kết quả: Subset.out

- Dòng 1: Tập B ứng với số hiệu S
- Dòng 2: Dãy số C

Subset.inp	Subset.out
3 8 2 3	3 7

2.4 Chuỗi hạt Nlace

Khi tiến hành khai quật khảo cổ một vương quốc nọ, các nhà khoa học khai quật được rất nhiều chuỗi hạt lạ. Sau khi quan sát các nhà khoa học thấy rằng các chuỗi hạt này có một số đặc điểm chung.

Mỗi chuỗi hạt là một sợi dây được đính các hạt ngọc làm bằng chất liệu cổ xưa. Các chuỗi hạt đều có số lượng hạt ngọc bằng nhau. Hơn nữa mỗi hạt ngọc là một hình cầu có đường kính là một số nguyên dương. Nếu lần từ trái sang phải ta thấy đường kính các hạt ngọc tăng dần. Nếu đánh số vị trí các hạt ngọc bắt đầu từ 1, theo thứ tự từ trái sang phải người ta nhận thấy rằng hạt ngọc thứ i có đường kính không vượt quá 2i. Các nhà khoa học cho rằng dân tộc cổ xưa này hẳn đã làm ra tất cả các chuỗi hạt có cùng những đặc điểm này.

Sau đó không lâu, các nhà khoa học tìm ra một mảnh da trên đó có ghi một con số kỳ lạ theo chữ số cổ xưa. Họ cho rằng mảnh da này có liên quan đến các chuỗi hạt. Sau nhiều cố gắng các nhà khoa học đã đưa được con số trên mảnh da về hệ thập phân và ký hiệu X.

Các nhà khoa học nhận định hãy xác định chuỗi hạt có thứ tự từ điển là X, biết đâu đây sẽ là manh mối về mối liên hệ với các chuỗi hạt.

Bạn hãy lập trình giúp các nhà khoa học tìm chuỗi hạt thứ X

Dữ liệu vào: Nlace.inp

- Dòng 1: N là số ngọc trong mỗi chuỗi hạt
- Dòng 2: X

Kết quả: Nlace.out

- Là tập đường kính của các hạt trong chuỗi hạt thứ X.

Ví dụ:

Nlace.inp	Nlace.out	Giải thích
2 4	2 3	Các chuỗi hạt là: 1 2, 1 3, 1 4, 2 3, 2 4.

2.5. Thứ tự xâu (STRING)

Các chữ cái trong bộ chữ cái tiếng Anh a..z được đánh số từ 1 đến 26.

Một xâu W= c_1 ... c_{i-1} c_i chỉ gồm các chữ cái thường tiếng Anh khác nhau được gọi là một từ đúng nếu trong xâu đó, ký tự c_k đứng sau ký tự c_{k-1} trong bộ chữ cái với mọi 1<k<i. , , ,

Với thứ tự này ta có thể đánh số các từ theo nguyên tắc sau:

- Xem mỗi chữ cái là một từ độ dài 1 với số hiệu như trên
- Chia mọi từ thành các nhóm W(1), W(2), ., W (K) trong đó W (i) là nhóm gồm mọi từ có độ dài i
- Trong mỗi nhóm, ta xếp các từ theo thứ tự từ điển
- Lần lượt đánh số liên tiếp từ 1, nhóm có độ dài bé đánh số trước, trong mỗi nhóm, thứ tự đánh số là thứ tự từ điển

Ví dụ về số hiệu một số từ: ac - 28; bc - 52; vwxyz - 83681

Cho một xâu S chỉ gồm các chữ cái tiếng Anh thường. Nói chung S không là từ nhưng ta có thể cắt S một cách duy nhất thành một dãy từ sao cho không thể ghép hai từ kề nhau thành một từ. Khi đó ta có thể mã hóa S thành một dãy số nguyên dương C(S) mà các số hạng của dãy này tương ứng là các số hiệu của các từ của dãy từ nhận được. Ví dụ: nếu S = abazvwxyz thì C(S) = 275183681

Yêu cầu:

- Cho một xâu S độ dài không quá 1000, tìm dãy C(S).
- Cho một dãy C(S') là mã hóa của một xâu S' thỏa mãn điều kiện như yêu cầu 1, hãy khôi phục lại từ S'.

Dữ liệu: STRING.INP

- Dòng thứ nhất ghi từ đầu dòng xâu S.
- Dòng thứ hai ghi dãy C(S'), hai số liên tiếp cách nhau ít nhất một dấu trống.

Kết quả: STRING.OUT

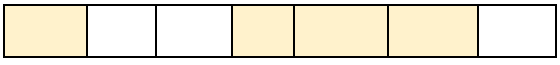
- Dòng thứ nhất ghi dãy C(S).
- Dòng thứ hai ghi xâu S'.

Ví dụ:

STRING.INP	STRING.OUT
abazvwxyz 27 51 83681	27 51 83681 abazvwxyz

2.6. Mã vạch (BARCODE)

Một mã vạch (thường dùng để ghi giá trị hàng hoá) là một dãy các vạch đen trắng xen kẽ nhau bắt đầu bằng vạch đen bên trái , mỗi vạch có một độ rộng tính bằng số nguyên dương . Hình sau cho ví dụ về một mã 4 vạch trải trên 1+2+3+1 = 7 đơn vị rộng:



Tổng quát , một tập mã vạch Mv(n,k,m) là tập mọi mã vạch gồm k vạch trải trên

đúng n đơn vị rộng và mỗi vạch không rộng quá m đơn vị .

Mã vạch trong ví dụ trên thuộc tập Mv(7,4,3) nhưng không thuộc tập Mv(7,5,2) . ta có thể biểu diễn mỗi mã vạch bởi một dãy nhị phân bằng cách dùng số 1 biểu thị một đơn vị rộng màu đen và số 0 biểu thị một đơn vị rộng mầu trắng . Ví dụ mã vạch trong hình trên sẽ được biểu thị bởi dãy 1001110

Với biểu diễn nhị phân như vậy mỗi tập Mv(n,k,m) với các giá trị cụ thể của m , k , n là một tập các xâu độ dài n chỉ gồm các ký tự 0 hay 1 . Ta có thể liệt kê tập đó theo thứ tự từ điển với quy ớc ký tự 0 trước ký tự l và theo thứ tự liệt kê đó ta cho mỗi mã vạch một số hiệu .

Ví dụ tập Mv(7,4,3) gồm ló mã vạch sẽ được biểu diễn bởi 16 xâu nhị phân độ dài 7 và theo cách sắp xếp từ điển của các biểu diễn nhị phân ta có các mã vạch với các số hiệu từ 1 đến 16 là :

01	1000100	09	1100100
02	1000110	10	1100110
03	1001000	11	1101000
04	1001100	12	1101100
05	1001110	13	1101110
Oó	1011000	14	1110010
07	1011100	15	1110100
0S	1100010	ló	1110110

Dữ liệu: BARCODE.INP

- Dòng thứ nhất ghi 3 số n , k , m (l<n,k,m<33) là ba tham số của tập mã vạch
- Dòng thứ hai ghi số s (s<50)
- Trong s dòng tiếp theo ghi mỗi dòng một xâu gồm n ký tự 0 hay 1 là biểu nhị phân của một mã vạch thuộc tập Mv(n,k,m) .

Kết quả: BARCODE.OUT

- S dòng , mỗi dòng ghi số hiệu của mã vạch đã cho

Ví dụ :

BARCODE.INP	BARCODE.OUT
7 4 3 5 5 1001110 1110110 1001100 1001110 1000100	5 16 4 5 l

Gợi ý: Xem mỗi vạch như một số nguyên dương có giá trị bằng độ rộng của

vạch đó. Gọi F[n,k,m] là số dãy mà:.

- Mỗi dãy có k số
- Tổng các số trong mỗi dãy là n
- Mỗi số trong dãy có giá trị không lớn hơn m

Chuyển một mã vạch nhị phân sang dãy số tương ứng. Khi đó số ở vị trí lẻ tương ứng với vạch 1 và số ở vị trí chẵn tương ứng với vạch 0. Dựa vào dãy số và mảng F để xuất ra số hiệu của mã vạch.

2.7. Dãy số Catalan (CATALAN)

Cho số nguyên dương N, dãy Catalan cấp n là dãy C(1), C(2) ... C(2n+1) gồm các số nguyên không âm thoả mãn : C(1) = C(2n+1) = 0 với i bất kì $1 \leq i \leq 2n$ thì C(i), C(i+1) hơn kém nhau 1 đơn vị.

Với mỗi n ta sắp xếp các dãy Catalan theo thứ tự từ điển, đánh số từ 1 trở đi .

Yêu cầu :

- Cho một dãy Catalan, hãy tìm thứ tự của dãy.
- Cho số nguyên dương k hãy tìm dãy có thứ tự k.

Dữ liệu: CATALAN.INP

- Dòng đầu ghi n. ($n \leq 15$)
- Dòng hai ghi một dãy Catalan cấp n
- Dòng ba ghi một số nguyên dương k (k có thể rất lớn nhưng đảm bảo luôn có nghiệm)

Kết quả: CATALAN.OUT

^ Dòng 1 ghi số thứ tự dãy ở dòng hai của file dữ liệu

Dòng 2 ghi dãy ứng với số thứ tự k

Ví dụ:

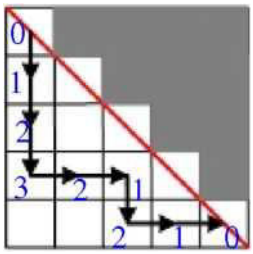
CATALAN.INP	CATALAN.OUT
4 0 1 2 3 2 1 2 1 0 12	12 0 1 2 3 2 1 2 1 0

Gợi ý: Xét lưới vuông (n+1)x(n+1). Một quy tắc đi thỏa mãn:

- Xuất phát từ ô (1,1) đến ô (n+1,n+1)
- Mỗi bước đi chỉ được di chuyển đến ô kề cạnh bên phải hoặc bên dưới.
- Không được di chuyển qua ranh giới là đường chéo nối đỉnh trái trên và phải dưới của lưới.

Nếu quy định ô (1,1) tương ứng với số 0 và

- mỗi bước di chuyển sang phải, ta tạo số mới bằng số liền trước trừ đi 1.
- mỗi bước di chuyển xuống dưới, ta tạo số mới bằng số liền trước cộng thêm 1. Dễ thấy mỗi cách đi từ ô(1,1) đến (n+1, n+1) tương đương với một dãy catalan tương ứng.



Cách đi tương ứng với dãy 0 1 2 3 2 1 2 1 0 (n = 4)

Gọi F[i,j] là số đường đi từ ô (i,j) đến ô (n+l,n+l) - F[i,j] được tính bằng công thức Quy hoạch động.

Giả sử tại ô (u,v) ta di chuyển xuống ô phía dưới thì cách đi này sẽ có số thứ tự lớn hơn cách đi từ ô (u,v) di chuyển sang ô bên phải (với u > v). Do đó ta chỉ quan tâm đến những ô (u,v) mà tại đó thực hiện di chuyển xuống ô phía dưới, ta cộng vào s thêm F[u,v+l]. Kết quả số thứ tự của dãy catalan tương ứng với cách đi là s+1

CHUYÊN ĐỀ
QUY HOẠCH ĐỘNG BAO LỒI

A. MỞ ĐẦU

Như chúng ta đã biết Quy hoạch động là một lớp thuật toán rất quan trọng và có nhiều ứng dụng trong ngành khoa học máy tính. Trong các kỳ thi quy hoạch động luôn là một trong những chủ đề chính. Tuy nhiên để tối ưu hoá các bài toán quy hoạch động là một vấn đề rất khó khăn. Có một số kỹ thuật tối ưu hoá độ phức tạp của một số thuật toán quy hoạch động trong đó có kỹ thuật bao lồi.

Trong chuyên đề này tôi sẽ trình bày những kiến thức và các bài tập đã sưu tầm trong quá trình giảng dạy về kỹ thuật bao lồi (convex hull trick) để tối ưu hoá bài toán quy hoạch động.

Việc hệ thống hóa bài tập kinh điển về chủ đề này đã giúp tôi giảng dạy hiệu quả hơn, học sinh code được các thuật toán để giải quyết các bài toán này. Trên cơ sở đó hình thành cho học sinh kiến thức nền cần thiết từ đó có thể vận dụng vào các bài toán mới.

B. NỘI DUNG

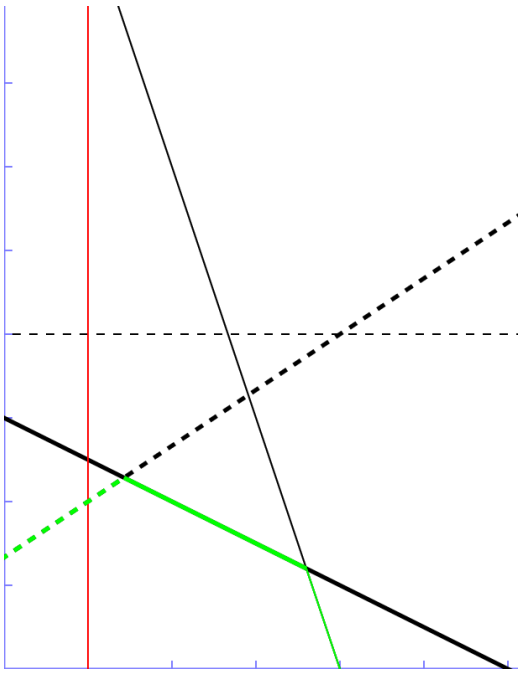
Bài toán 1:

Đề bài: Cho một tập n hàm bậc nhất $y = a_i x + b_i$ và Q truy vấn, mỗi truy vấn là một số thực x, cần trả về số thực y là giá trị nhỏ nhất của các hàm số với biến số là x.

Thuật toán:

Thuật toán đơn giản: Với mỗi truy vấn, ta duyệt qua tất cả các hàm số để tìm giá trị nhỏ nhất. Thuật toán này có độ phức tạp là O(N*Q)

Kỹ thuật bao lồi: Mỗi hàm số bậc nhất có thể biểu diễn bởi một đường thẳng trong hệ toạ độ Oxy như hình:



Hình 1. Đồ thị

Xét đồ thị ở trên:

Ta thấy đường thẳng y = 4 không bao giờ là kết quả của truy vấn với mọi x. Với 3 đường thẳng còn lại, mỗi đường là "thấp nhất" trên một khoảng liên tục (có thể tiến tới vô cùng). Ngoài ra, với x tăng dần, thì hệ số góc của đường thẳng "thấp nhất" tương ứng là giảm dần, điều này luôn luôn đúng.

Như vậy, nếu ta bỏ đi đường thẳng không tiềm năng y = 4, đồng thời sắp xếp các đường thẳng tiềm năng còn lại theo hệ số góc, với mỗi truy vấn, ta tìm kiếm nhị phân là có thể trả về kết quả. Ở đây từ "bao lồi" thực ra chưa đúng lắm, trong trường hợp này (ví dụ như hình vẽ) thì có thể tạm coi là "một nửa trên của bao lồi" (phần tô màu xanh).

Thêm một đường thẳng

Như đã nói ở trên, nếu các đường thẳng tiềm năng được lọc ra và sắp xếp theo hệ số góc, ta có thể dễ dàng trả lời truy vấn trong O(logN) sử dụng tìm kiếm nhị phân. Dưới đây sẽ trình bày một thuật toán, trong đó ta lần lượt thêm các đường thẳng vào một cấu trúc dữ liệu rỗng, tính lại các đường thẳng tiềm năng, cuối cùng khi tất cả các đường thẳng đã được thêm vào thì cấu trúc dữ liệu sẽ hoàn chỉnh.

Giả sử ta có thể sắp xếp tất cả các đường thẳng giảm dần theo hệ số góc, thì việc còn lại chỉ là lọc ra những đường thẳng tiềm năng. Khi thêm một đường thẳng mới, một

số đường thẳng có thể được bỏ đi khi mà chúng không còn tiềm năng nữa. Ta sẽ sử dụng một stack để chứa những đường thẳng tiềm năng, trong đó những đường thẳng vừa mới được thêm vào sẽ nằm ở đỉnh của stack.

Khi thêm một đường thẳng mới, ta sẽ kiểm tra xem đường thẳng nằm ở đỉnh stack có còn tiềm năng nữa hay không. Nếu nó vẫn còn tiềm năng, ta chỉ việc đẩy thêm đường thẳng mới vào và tiếp tục. Nếu không, ta bỏ nó đi và lại tiếp tục lại quá trình cho đến khi đường thẳng ở đỉnh stack vẫn tiềm năng hoặc stack chỉ còn một đường thẳng.

Vấn đề còn lại chỉ là làm sao để kiểm tra xem một đường thẳng có thể bỏ đi hay không. Gọi l_1, l_2, l_3 lần lượt là đường thẳng thứ hai, thứ nhất (ở đỉnh stack), và đường thẳng mới để thêm vào. Thì l_2 trở nên không tiềm năng nếu và chỉ nếu điểm cắt của l_1 và l_3 nằm ở bên trái (có hoành độ nhỏ hơn) điểm cắt của l_1 và l_2. Điều này là xảy ra khi l_3 có thể phủ hết khoảng mà trước đó l_2 là thấp nhất.

Thuật toán ở trên đúng nếu tập đường thẳng đôi một không song song, nếu không ta chỉ việc sửa đổi thuật toán sắp xếp để những đường thẳng tiềm năng hơn (có hằng số nhỏ hơn) ở sau những đường thẳng song song với nó.

Đánh giá độ phức tạp

Về mặt bộ nhớ, độ phức tạp thuật toán là O(N), khi mà ta chỉ cần lưu trữ danh sách các đường thẳng đã được sắp xếp. Bước sắp xếp ban đầu có thể làm trong O(NlogN). Khi duyệt qua các đường thẳng, mỗi một trong chúng được đẩy vào stack đúng một lần, và có thể bị pop ra không quá một lần. Như vậy bước dựng bao lồi này chỉ mất thời gian O(N).

Tóm lại toàn bộ thuật toán bao gồm cả phần sắp xếp mất thời gian O(NlogN), nếu các đường thẳng đã được sắp xếp sẵn, thì thuật toán chạy trong thời gian tuyến tính.

<u>Bài toán 2:</u>

Đề bài:

Cho $n \leq 300000$ cặp số (x,y) ($1 \leq x, y \leq 1000000$). Ta có thể nhóm một vài cặp số lại thành một nhóm. Giả sử một nhóm gồm các cặp số thứ $a_1,a_2,\ldots,a_m$ thì chi phí cho nhóm này sẽ là $\max(x_{a1}, x_{a2},\ldots,x_{am}) * \max(y_{a1},y_{a2},\ldots,y_{am})$

Yêu cầu: Tìm cách phân nhóm có tổng chi phí bé nhất.

Input

- Dòng đầu tiên là số nguyên dương N.
- N dòng tiếp theo dòng thứ i gồm hai số xi và yi.

Output

- Gồm 1 số duy nhất là kết quả tìm được.

Ví dụ:

Input	Output
4 100 1 15 15 20 5 1 100	500

Giải thích: cách phân nhóm thích hợp là (1), (2,3) và (4)

Thuật toán

Nhận xét 1

Nếu tồn tại hai cặp số (x_1,y_1) và (x_2,y_2) mà $x_1>x_2$ và $y_1>y_2$ thì ta nói cặp số $(x_2\ y_2)$ là không tiềm năng và có thể loại bỏ mà không cần quan tâm đến nó. Bởi vì ta có thể cho nó vào cùng nhóm với cặp đầu tiên mà không làm tồi đi kết quả.

Như vậy ta có thể sắp xếp lại các cặp số tăng dần theo x, sau đó sử dụng stack để loại bỏ đi những cặp số không tiềm năng, cuối cùng còn lại một dãy các cặp với x tăng dần và y giảm dần. Từ đây ta chỉ cần giải bài toán cho dãy các cặp số này.

Nhận xét 2

Nếu ta có hai cặp số (x_1,y_1) và (x_2,y_2) thuộc cùng một nhóm, thì ta có thể thêm vào nhóm đấy các cặp số (x,y) mà $x_1 \le x \le x_2$ và $y_1 \ge y \ge y_2$ mà không làm tồi đi kết quả. Như vậy có nhận xét: các nhóm được phân hoạch gồm các phần tử liên tiếp nhau.

Quy hoạch động

Với hai nhận xét trên, ta đã có thể có được thuật toán QHĐ đơn giản với độ phức tạp đa thức. Gọi F(i) là chi phí nhỏ nhất để phân nhóm các phần tử có chỉ số không quá ii. Công thức truy hồi:

$F(i)=\min[F(j)+x_i*y_j]$ với $0 \le j < i 0 \le j < i$

Có thể cài đặt thuật toán trên với độ phức tạp $O(N^2)$ tuy nhiên như vậy vẫn chưa đủ tốt với giới hạn của đề bài.

Áp dụng bao lồi

Đặt $y_j=a$, $x_i=x$, và $F(j)=b$. Rõ ràng ta cần cực tiểu hóa một hàm bậc nhất $y=a*x+b$ bằng việc chọn j hợp lí. Đồng thời trong bài toán này thì hệ số góc a của các đường thẳng là giảm dần, như vậy có thể áp dụng trực tiếp convex hull trick. Để ý một tí là các truy vấn (các giá trị x) là tăng dần, nên ta không cần phải tìm kiếm nhị phân mà có thể tịnh tiến để tìm kết quả. Độ phức tạp cho phần QHĐ này là O(N).

Code tham khảo:

```
#include <stdio.h>
#include <vector>
#include <iostream>
#include <algorithm>

using namespace std;
#define long long long
#define f1(i,n) for (int i=1;i<=n;i++)
```

```
#define f0(i,n) for (int i=0;i<n;i++)
typedef pair<int, int> ii;
typedef pair<long, long> ll;
#define X first
#define Y second
#define  N 1000006
int n;
ii a[N];
bool useless(ll a, ll b, ll c)
{
    return (b.Y-a.Y)*(a.X-c.X)>=(c.Y-a.Y)*(a.X-b.X);
}
long h(long x, ll d)
{
    return x*d.X+d.Y;
}
int main()
{
    freopen("bai2.inp","r",stdin);
    freopen("bai2.out","w",stdout);
    scanf("%d",&n);
    f1(i,n) scanf("%d%d",&a[i].X,&a[i].Y);
    sort (a+1,a+n+1);
    vector <ii> b;
```

```
    f1(i,n)
    {
        while (b.size()&& b.back().Y<=a[i].Y) b.pop_back();
        b.push_back(a[i]);
    }
    vector <ll> d;
    long Lastf=0;
    int Best=0;
    f0(i,b.size())
    {
        d.push_back(ll(b[i].Y,Lastf));
        #define sz d.size()
        while (sz>=3 && useless(d[sz-3],d[sz-2],d[sz-1]))
        {
            if (Best>=sz-2) Best--;
            d.erase(d.end()-2);
        }
        while (Best+1<d.size() &&
             h(b[i].X,d[Best+1])<=h(b[i].X,d[Best]))Best++;
        Lastf=h(b[i].X,d[Best]);
    }
    cout << Lastf;
    return 0;
}
```

Như vậy với một bài toán chỉ cần ta tìm ra công thức QHĐ mà có thể biểu diễn dưới dạng nhị thức bậc nhất, đồng thời hệ số góc có thể tăng/giảm dần, thì có thể áp dụng kỹ thuật trên để tối ưu.

Bài toán 3:

Đề bài: HARBINGERS (CEOI 2009)

Ngày xửa ngày xưa, có N thị trấn kiểu trung cỏ trong khu tự trị Moldavian. Các trhị trấn này được đánh số từ 1 đến N. Thị trấn 1 là thủ đô. Các thị trấn được nối với nhau bằng N−1 con đường hai chiều, mỗi con đường có độ dài được đo bằng km. Có duy nhất một tuyến đường nối giữa hai điểm bất kỳ (đồ thị các con đường là hình cây). Mỗi thị trấn không phải trung tâm có một người truyền tin.

Khi một thị trấn bị tấn công, tình hình chiến sự cần được bảo về thủ đô càng sớm càng tốt. Mội thông điệp được truyền bằng các người truyền tin. Mỗi người truyền tin được đặc trưng bởi lượng thời gian khởi động và vận tốc không đổi sau khi xuất phát. Thông điệp luôn được truyền trên con đường ngắn nhất đến trung tâm. Ban đầu, thông tin chiến sự được đưa cho người truyền tin tại thị trấn bị tấn công. Từ thị trấn đó người truyền tin sẽ đi theo con đường về gần thủ đô hơn. Khi đến một thị trấn mới vừa đến. Lưu ý rằng khi chuyển sang người truyền tin mới thì người này cần một lượng thời gian để khởi động rồi mới đi chuyển tin. Như vậy, thông điệp sẽ được chuyển bằng một số người truyền tin trước khi đến thử đô.

Hãy xác định thời gian ít nhất cần chuyển tin từ các thị trấn về thủ đô.

Input

- Dòng đầu ghi số N.
- N−1 dòng tiếp theo, mỗi dòng ghi ba số u, v, và d thể hiện một con đường nối từ u đến v với độ dài bằng d.
- N−1 dòng tiếp theo, dòng thứ i gồm hai số S_i và V_i thể hiện thời gian cần để khởi động và số lượng phút để đi được 1km của người truyền tin ở thị trấn i+1.

Output

- Ghi $N-1$ số trên một dòng. Số thứ i thể hiện thời gian ít nhất cần truyền tin từ thành phố i+1 về thủ đô.

Ví dụ

Input	Output
5 1 2 20 2 3 12 2 4 1 4 5 3 26 9 1 10 500 2 2 30	206 321 542 328

Giới hạn

- $3 \leq N \leq 100000$
- $0 \leq Si, Vi \leq 10^9$
- Độ dài mỗi con đường không vượt quá 1000010000

Thuật toán QHĐ

Gọi F(i) là thời gian ít nhất để truyền tin từ thành phố thứ ii đến thủ đô, ta có công thức truy hồi:

$$F(i)=\min[F(j)+dist(j,i)*Vi+Si]$$

với j là một nút trên đường từ thành phố i đến thành phố 1. Trong đó dist(j,i) là khoảng cách giữa 2 thành phố i và j, có thể tính trong $O(1)$ sử dụng mảng cộng dồn D[] với D[i] là khoảng cách từ thành phố i tới thủ đô. Thuật toán này có thể dễ dàng cài đặt với độ phức tạp là $O(N^2)$

Áp dụng bao lồi

Công thức truy hồi có thể viết lại thành

$$F(i)=\min[F(j)-Dj*Vi+Di*Vi+Si]$$

Khi ta tính F(i)F(i), thì giá trị $Di*Vi+Si$ là hằng số với mọi j, vì vậy

$$F(i)=\min[F(j)-Dj*Vi]+Di*Vi+Si$$

Có thể thấy rằng ta cần tìm giá trị nhỏ nhất của hàm bậc nhất $y=-Dj*x+F(j)$

Trong trường hợp tổng quát, ta cần một cấu trúc dữ liệu cho phép xử lí hai thao tác:

- Khi DFS xuống một nút con, ta cần thêm một đường thẳng.
- Khi quá trình DFS tính F[]F[] cho gốc cây con đã hoàn tất, ta cần xóa một đường thẳng, trả cấu trúc dữ liệu về trạng thái ban đầu.

Các thao tác này có thể được thực hiện hiệu quả trong O(logN). Cụ thể ta sẽ biểu diễn stack bằng một mảng cũng một biến size (kích thước stack). Khi thêm một đường thẳng vào, ta sẽ tìm kiếm nhị phân vị trí mới của nó, rồi chỉnh sửa biến size cho phù hợp, chú ý là sẽ có tối đa một đường thẳng bị ghi đè, nên ta chỉ cần lưu lại nó. Khi cần trả về trạng thái ban đầu, ta chỉ cần chỉnh sửa lại biến size đồng thời ghi lại đường thẳng đã bị ghi đè trước đó. Để quản lí lịch sử các thao tác ta sử dụng một vector lưu lại chúng. Độ phức tạp cho toàn bộ thuật toán là O(NlogN)

Code tham khảo:

```
#include <bits/stdc++.h>
#define X first
#define Y second
const int N = 100005;
const long long INF = (long long)1e18;
using namespace std;
typedef pair<int, int> Line;
struct operation {
```

```
    int pos, top;

    Line overwrite;

    operation(int _p, int _t, Line _o) {

        pos = _p; top = _t; overwrite = _o;

    }

};

vector<operation> undoLst;

Line lines[N];

int n, top;



long long eval(Line line, long long x) {return line.X * x
+ line.Y;}

bool bad(Line a, Line b, Line c)

    {return (double)(b.Y - a.Y) / (a.X - b.X) >=
(double)(c.Y - a.Y) / (a.X - c.X);}



long long getMin(long long coord) {

    int l = 0, r = top - 1; long long ans =
eval(lines[l], coord);

    while (l < r) {

        int mid = l + r >> 1;

        long long x = eval(lines[mid], coord);

        long long y = eval(lines[mid + 1], coord);

        if (x > y) l = mid + 1; else r = mid;

        ans = min(ans, min(x, y));
```

```
    }
    return ans;
}

bool insertLine(Line newLine) {
    int l = 1, r = top - 1, k = top;
    while (l <= r) {
        int mid = l + r >> 1;
        if (bad(lines[mid - 1], lines[mid], newLine)) {
            k = mid; r = mid - 1;
        }
        else l = mid + 1;
    }
    undoLst.push_back(operation(k, top, lines[k]));
    top = k + 1;
    lines[k] = newLine;
    return 1;
}

void undo() {
    operation ope = undoLst.back(); undoLst.pop_back();
    top = ope.top; lines[ope.pos] = ope.overwrite;
}
```

```
long long f[N], S[N], V[N], d[N];
vector<Line> a[N];

void dfs(int u, int par) {
    if (u > 1)
        f[u] = getMin(V[u]) + S[u] + V[u] * d[u];
    insertLine(make_pair(-d[u], f[u]));
    for (vector<Line>::iterator it = a[u].begin(); it != a[u].end(); ++it) {
        int v = it->X;
        int uv = it->Y;
        if (v == par) continue;
        d[v] = d[u] + uv;
        dfs(v, u);
    }
    undo();
}

int main() {
    ios::sync_with_stdio(0); cin.tie(0);
    cin >> n;
    int u, v, c;
    for (int i = 1; i < n; ++i) {
```

```
        cin >> u >> v >> c;
        a[u].push_back(make_pair(v, c));
        a[v].push_back(make_pair(u, c));
    }
    for (int i = 2; i <= n; ++i) cin >> S[i] >> V[i];
    dfs(1, 0);
    for (int i = 2; i <= n; ++i) cout << f[i] << ' ';
    return 0;
}
```

Bài toán 4:

Đề bài:

Có n cái cây 1, 2, …, n trong đó cây thứ i có chiều dài A[i]∈N. Bạn phải cắt tất cả các cây thành các đoạn có chiều dài 1. Bạn có một cái cưa máy, mỗi lần chỉ chặt được một đơn vị chiều dài, và mỗi lần sử dụng thì lại phải sạc pin. Chi phí để sạc pin phụ thuộc vào chi phí của cây đã bị cắt hoàn toàn (một cây bị cắt hoàn toàn có chiều dài 0). Nếu chỉ số lớn nhất của cây bị cắt hoàn toàn là ii thì chi phí sạc là B[i], và khi bạn đã chọn một cây để cắt, bạn phải cắt nó hoàn toàn. Ban đầu cái cưa máy được sạc đầy pin. Giả sử $a_1=1\leq a_2\leq \ldots \leq a_n$ và $b_1\geq b_2 \ldots \geq b_n=0$.

Tìm một cách cưa cây với chi phí nhỏ nhất.

Ví dụ: n = 6, A[1,2,…,6] = {1, 2, 3, 10, 20, 30} B[1, 2, …, 6] = {6, 5, 4, 3, 2, 0}.

Nếu bạn chặt cây theo thứ tự 1→2→6→3→4→5 thì chi phí bạn phải trả là 2×6+30×5+3×0+10×0+20×0=162.

Nếu bạn chặt theo thứ tự: 1→3→6→2→4→5 thì chí phí bạn phải trả là 3×6+30×4+4×0+10×0+20×0=138.

Thuật toán:

Từ ví dụ trên, ta có nhận xét sau: Chi phí để chặt các cây sau khi đã chặt cây thứ n là 0.

Do đó, bài toán trên có thể được quy về bài toán: tìm chi phí nhỏ nhất để chặt cây thứ n. Gọi OPT={$1=i_1,i_2,\ldots,i_k$} là một thứ tự chặt cây tối ưu với $i_k=n$. Ta có bổ đề sau: $1=i_1\leq i_2\leq\ldots\leq i_k=n$

Chứng minh: Giả sử bổ đề là sai, suy ra tồn tại aa nhỏ nhất sao cho sao cho $1=i_1\leq i_2\leq i_a\geq i_{a+1}$. Xét dãy OPT\{$i_{a+1}$}={$1=i_1,i_2,\ldots,i_a,i_{a+2}\ldots,i_k=n$}. Chi phí chặt cây theo thứ tự của dãy này nhỏ hơn OPT, do đó, trái với giả thiết dãy OPT là thứ tự chặt cây tối ưu.

Dựa vào bổ đề 1, ta có thể phát triển thuật toán quy hoạch động như sau: Gọi C[i] là chi phí nhỏ nhất để chặt cây thứ i"sử dụng" các cây 1,2,…,i−1. Ta có công thức sau:

C[i] = {B[i], if i=1 min j<i{C[j]+A[i]B[j]}+B[j]

Dễ thấy, chi phí để chặt cây thứ n là C[n]. Công thức quy hoạch động tính C[n] với mảng A, B là các mảng đã săp xếp tương ứng với trường hợp đặc biệt 2.

Do đó, bạn có thể giải quyết bài toán này trong thời gian O(n).

Code tham khảo:

```
#include <iostream>
#include <fstream>
#include <cmath>
#include <algorithm>
#include <vector>
#include <set>
#include <map>
```

```
#include <stack>
#include <queue>
#include <cstdlib>
#include <cstdio>
#include <string>
#include <cstring>
#include <cassert>
#include <utility>
#include <iomanip>

using namespace std;

const int MAXN = 105000;

int n;
long long height[MAXN], tax[MAXN];
long long dp[MAXN];
vector <long long> mvals, bvals;
int cur = 0;



bool bad(long long m1, long long b1, long long m2, long
long b2, long long m3, long long b3) {

    return 1.0 * (b1 - b3) * (m2 - m1) < 1.0 * (b1 - b2)
* (m3 - m1);
}
```

```
void add(long long m, long long b) {
    while ( (int) mvals.size() >= 2 &&
bad(mvals[mvals.size() - 2], bvals[bvals.size() - 2],
mvals[mvals.size() - 1], bvals[bvals.size() - 1], m, b))
{
        mvals.pop_back(); bvals.pop_back();
    }
    mvals.push_back(m); bvals.push_back(b);
}

void setCur(long long x) {
    if (cur > (int) mvals.size() - 1)
        cur = (int) mvals.size() - 1;

    while (cur < (int) mvals.size() - 1 && 1.0 *
mvals[cur + 1] * x + bvals[cur + 1] <= 1.0 * mvals[cur] *
x + bvals[cur])
        cur++;
}

int main() {
    freopen("bai4.inp", "r", stdin);
    freopen("bai4.out", "w", stdout);

    scanf("%d", &n);
    for (int i = 1; i <= n; i++)
        scanf("%I64d", &height[i]);
```

```
    for (int i = 1; i <= n; i++)
        scanf("%I64d", &tax[i]);



    for (int i = 1; i <= n; i++) {
        if (i == 1) {
            dp[i] = 0;
        }
        else {
            setCur(height[i]);
            dp[i] = mvals[cur] * height[i] + bvals[cur];
        }
        add(tax[i], dp[i]);
    }
    cout << dp[n];
    return 0;
}
```

Bài luyện tập

Lời dẫn: Sau khi dạy xong các bài trên, hãy cùng học sinh làm việc với các bài tập mới sau đây để hiểu rõ hơn về chuyên đề này, đồng thời nâng cao kĩ năng lập trình cho học sinh. Phải đảm bảo rằng học sinh tự code được các bài toán kinh điển trên.

Các bài tập đề xuất sau đây:

http://vnoi.info/problems/show/VMPIZZA/

http://codeforces.com/contest/319/problem/C

https://www.spoj.com/problems/ACQUIRE/

https://www.spoj.com/problems/APIO10A/

https://vn.spoj.com/problems/NKLEAVES/

C. KẾT LUẬN

Quy hoạch động bao lồi là một nội dung khó và học sinh thường ít áp dụng được vào bài tập thực tế do không code thành công được các bài toán kinh điển.

Khi dạy chuyên đề này, ngoài việc đưa ra thuật toán tôi thường mô phỏng code chuẩn cho học sinh để học sinh giải quyết được bài toán thực tế. Thiết nghĩ đây cũng là cách làm khi dạy các chuyên đề khó để học sinh nắm chắc kiến thức hơn.

Do năng lực còn hạn chế nên trong chuyên đề tôi chỉ trình bày một số bài toán áp dụng bao lồi trong quá trình tối ưu hoá thuật toán quy hoạch động đã sưu tầm và biên soạn lại. Rất mong nhận được sự góp ý, đánh giá của các đồng nghiệp.

HƯỚNG DẪN GIẢI THUẬT

Bài 1:

Sub 1: Duyệt $O(n^2)$

Sub 2: Mỗi lần ta lưu chênh lệch số lượng 2 bên. $O(n)$

Bài 2 (7 điểm).

Ta chia bài toán thành 2 bước

- Bước 1 : tạo 1 mảng f[i][j] mang ý nghĩa là xét từ vị trí thứ i của xâu X và vị trí thứ j của xâu Y thì độ dài đoạn tương đồng bậc k dài nhất là bao nhiêu. Dễ thấy f[i][j] = f[i+1][j+1] + 1 nếu X[i] và Y[j] tương đồng bậc k, f[i][j] =0 trong trường hợp còn lại.
- Bước 2: ta xét các vị trí cắt 1 và kiểm tra xem có tạo được xâu như vậy thỏa mãn không. Rõ ràng, với 2 vị trí i,j ta có 3 xâu (1,i), (i+1,j), (j+1,n) ta sẽ thử duyệt giai thừa các vị trí của 3 xâu này, cách xếp hiện tại sẽ thỏa mãn nếu ta đang bắt đầu ở vị trí thứ x của xâu X và có độ dài y, f[x][z] >= y, với z là tổng độ dài các xâu đã xét trước đó + 1.

Bài 3 (7 điểm).

- Subtask 1: Với n=1, dãy số này chỉ có một số thì chỉ có một cặp số (1,1), ta chỉ cần kiểm tra xem số này có phải là số fibonacci hay không, nếu có in ra 1, ngược lại in ra 0.
- Subtask 2 và Subtask 3: Gọi mảng prefix là mảng cộng dồn từ 1 → n

Với mỗi i duyệt từ Q → N, vì độ dài đoạn nằm trong đoạn P →Q nên với mỗi i ta xét các prefix trong đoạn i – Q +1 → i– P +1, duyệt j từ i – Q +1 → i– P +1 nếu prefix[i]- prefix[j] là số fibonacci thì kết quả tăng 1.

Độ phức tạp là $O(n^2)$

- Subtask 4: Nhận xét dãy có 10^5 phần tử với giá trị lớn nhất của các phần tử là 10^6 thì tổng dãy lớn nhất là 10^{11}, mà số fibonacci thứ 60 là 956722026041, lớn hơn 10^{11}. Do đó ta có thể lật ngược bài toán.

Xét với mỗi số fibonacci thì ta đếm có bao nhiêu đoạn có tổng chính bằng số fibonacci này có độ dài ≥ P và ≤ Q.

Độ phức tạp: O(60*n)

---------- Hết ----------

CÂU 1

Ta nhận thấy với mỗi giá trị ai thêm vào ra chỉ cần quan tâm là ai - 1 với ai +1 có tồn tại hay không,vì khi đó nó mới ảnh hưởng tới kết quả của bài toán.Ta sẽ sử dụng cây IT để lưu với mỗi node của cây sẽ lưu [l..r] bằng 1 mảng dp 2 chiều với 4 trạng thái là có sử dụng thằng l hay không hoặc có sử dụng thằng r hay không.

CÂU 2

Ta sẽ tính tổng tiền tố lớn nhất tại các vị trí cho các mảng wa, wb, wc (gọi là maA, maB, maC). Ta sẽ duyệt số lượng phần tử c được chọn sau đó dùng two pointer để tìm số lượng phần tử a nhiều nhất đc chọn nếu như chọn i phần tử của c. Ta sẽ cần tính thêm các mảng sau để phục vụ cho quá trình kiểm tra điều kiện đề bài :

- f[i] là vị trí nhỏ nhất sao cho khi chọn i phần tử dãy a thì các số trong dãy f[i] phần tử đầu của dãy b và i phần tử của dãy a khác nhau
- u[i] là vị trí nhỏ nhất sao cho khi chọn i phần tử dãy c thì các số trong dãy u[i] phần tử đầu của dãy a và i phần tử của dãy c khác nhau
- v[i] là vị trí nhỏ nhất sao cho khi chọn i phần tử dãy c thì các số trong dãy u[i] phần tử đầu của dãy b và i phần tử của dãy c khác nhau

Khi chọn i phần tử của dãy c thì vị trí xa nhất mà ta có thể chọn trên dãy a là u[i] và vị trí xa nhất mà ta có thể chọn trên dãy b là f[u[i]]

- Nếu như v[i] <= f[u[i]] nghĩa là v[i] là vị trí xa nhất mà vẫn thỏa mãn điều kiện dãy b và c khác nhau mà vẫn <= vị trí xa nhất mà thỏa mãn c khác a và a khác b thì ta có thể chọn được v[i] phần tử vào dãy b và u[i] phần tử vào dãy a.
- Ngược lại thì nếu như ta chọn v[i] phần tử vào dãy b thì sẽ ta sẽ cần tìm ra vị trí xa nhất mà vẫn thỏa mãn điều kiện đề bài. Ta hoàn toàn có thể tìm ra vị trí này dựa vào mảng f[i]. Gọi rev[i] là vị trí xa nhất trên dãy a mà vẫn còn thỏa mãn đk nếu ta chọn >= i phần tử trên dãy b. Tính rev[i] như sau : for i từ 1 -> n rev[f[i]] = i và for i từ n -> 1 rev[i] = max(rev[i], rev[i + 1])

Ta sẽ duyệt ngược số phần tử được chọn trên dãy c và dùng 2 con trỏ l r trên dãy a mang ý nghĩa là chọn đc l số trên dãy a và r số trên dãy a, ta

sẽ xét 2 trường hợp l r riêng biệt. Đặt l = vị trí xa nhất trên dãy a sao cho từ 1 -> l trên dãy a không có số nào xuất hiện quá 2 lần, r =0. Nhận thấy rằng khi duyệt i giảm thì l sẽ giảm và r thì tăng dần nên ta có thể làm như sau. Chừng nào mà lấy i số trên c và l số trên a mà thỏa mãn f[l] <= v[i] có nghĩa chọn l số trên a khác vs b và chọn i số trên c khác vs b thì giảm l, chừng nào mà lấy i số trên c và r số trên a mà thỏa mãn r <= u[i] tức r vẫn nằm trong khoảng trên a thỏa mãn a và c đang khác nhau thì tăng r. Và trong mỗi vòng while nếu như l < r thì ta có thể cập nhật đc đáp án vì khi đó tất cả các l >= r đều thỏa mãn điều kiện là f[l] <= v[i] và r <= u[i] nên chọn r số trên a thỏa mãn điều kiện và khi xét trong vòng while của l thì r > l và r thỏa mãn <= u[i] nên việc chọn l số trên a sẽ thỏa mãn đề bài và ta có thể cập nhật đáp án.

Code mẫu : [Q1FdZC - Online C++0x Compiler & Debugging Tool - Ideone.com]

CÂU 3

Triple :

Sub1 :

- For từng cặp đỉnh trước để tính dmin của từng điểm. Sau đó For i j k để chọn ra 3 điểm phân biệt và kiểm tra dmin[i] = dmin[j] = dmin[k] = khoảng cách của mỗi cặp đỉnh với nhau. Đpt(n ^ 3)

Sub2 :

-Ta cũng For từng cặp đỉnh giống như sub 1. Nhưng thay vì for i j k tất cả cặp đỉnh thì ta chỉ For những đỉnh j có dmin[i] = dmin[j] = Khoảng cách của i và j .

-Ta cũng chỉ For những k có dmin[k] = dmin[j] = khoang cach j và k.

-Sau đó ta chỉ cần xét dmin[i] = dmin[k] = khoảng cách i và k.

-Để tính được những đỉnh có dmin bằng nhau và bằng khoảng cách của chúng thì ta sẽ For với mỗi cặp điểm sẽ tính được dmin và For lại 1 lần để tính được những điểm thỏa mãn và lưu danh sách kề của điểm có chỉ số nhỏ hơn lưu những điểm có chỉ số lớn hơn thỏa mãn.

Sub4 :

-Với ý tưởng từ subtask 2. Ta sẽ tối ưu thời gian tính được dmin của điểm i bằng segment tree.
-Ta sẽ sort lại các điểm theo x tăng dần. Với mỗi điểm i ta sẽ xét những điểm i có chỉ số nhỏ hơn ta sẽ có 2 trường hợp Yj > Yi và Yj < Yi. Ta sẽ lưu giá trị trên 2 cây low và high. Khi có giá trị lớn hơn hay không rồi ta sẽ dễ dàng bỏ đi trị tuyệt đối của khoảng cách manhattan.

-Sau đó phải For vòng ngược lại để tính được trường hợp mà x giảm dần.

-Với mỗi dmin thì những điểm có dmin như vậy ta sẽ dùng xoay bảng để tính những điểm có khoảng cách bằng dmin mình đang xét trong bảng cho trước.

-Những điểm thỏa mãn sẽ lưu lại trong danh sách kề của điểm i và để tính kết quả ta sẽ cố định điểm i và for 2 vòng để chọn ra 2 điểm thỏa mãn điều kiện với điểm i và kiểm tra 2 điểm đấy có khoảng cách bằng dmin hay không và cộng vào đáp án.
-Nếu tính như vậy mỗi cặp 3 điểm sẽ được tính lại 3 lần nên khi in ra kết quả phải chia cho 3
code : https://ideone.com/wrzKva

CÂU 4 {
COMCAT:
-sub1: nhận xét thấy giá trị 0 được đi qua càng nhiều lần thì tổng MEX càng lớn, do vậy ta sẽ xây dựng trọng số cạnh từ giữa ra, vd: 3-1-0-2-4.
-sub2: sinh hoán vị
-sub3: theo nhận xét từ sub1 ta sẽ định nghĩa f[u][v] là tổng mex của cây khi điền đầy đủ các số theo thứ tự trên lên đường đi từ u -> v. khi điền xong sẽ duyệt qua tất cả các cặp đỉnh để tính tổng Mex. dpt O(n^4).
-sub4: dựa vào sub3 ta sẽ cải tiến hàm f[u][v] bằng quy hoạch động.
gọi dp[u][v] là tổng mex của cây khi điền đầy đủ các số theo thứ tự trên lên đường đi từ u -> v. với những đỉnh ở 2 phía u, v mà không nằm trên đường đi từ u -> v thì Mex giữa chúng đều tăng lên 1 do đó ta có công thức sau:
dp[u][v] = max(dp[u][par(v)] + size(u) * size(v), dp[par(u)][v] + size(u) * size(v));
DPT O(N^2).
code: https://ideone.com/LlD1nU
}

CÂU 5 {

COLDWAR

sub1 : (N <= 200 & M <= 500)

+ xét từng cạnh lần lượt, với mỗi cạnh ta xóa cạnh đó khỏi đồ thị rồi tính lại cây khung nhỏ nhất bằng kruskal.

sub2 : (M <= N + 100)

ta tính trước cây khung nhỏ nhất sau đó xét từng cạnh như sub1.

+ TH1 :Với những cạnh ko thuộc cây khung ban đầu thì đáp án chính là trọng số của MST ban dầu
+ TH2 :Với những cạnh thuộc MST ban đầu, ta xóa cạnh đó đi, khi đó cây khung sẽ chia ra thành 2 cây nhỏ, sau đó xét những cạnh ko thuộc MST xem có cạnh nào nối đc 2 cây con và lấy min của các cạnh đó
+ ĐPT : M log M + (M - N + 1) * (N - 1)

subfull:

+ Ta làm y như sub2 nhưng cải tiến xét TH2, khi đó ta sử dụng đến HLD kết hợp IT để giải quyết bài toán.
+ Với những cạnh ko thuộc MST, ta cập nhật min cho đường đi từ u -> v với trọng số w trên MST ban đầu. Khi đó khi truy vấn chỉ cần lấy giá trị của cạnh đó là xong

ĐPT : O(M log M + (M - N + 1) * log(N))

Code mẫu : PhcKhnhTapVietSolution

}

CÂU 6

- Bài toán truy vấn trên đoạn nhưng mọi thao tác cập nhật lại tác động lên cả mảng nên ta không cần dựng segment tree mà ta có thể sử dụng trie tương tự như tổng tiền tố.
- Ta có T_i là trie gồm i số đầu tiên của dãy, với mỗi truy vấn hỏi, ta chỉ cần lấy $T_H - T_{L-1}$ thì sẽ được trie gồm các số trong đoạn [L, H].
- Với truy vấn 1 x, rõ ràng T_{n+1} sẽ được xây từ T_n kết hợp với số x, ta xây dựng tương tự persistent segment tree, khi thêm phần tử x vào, ta chỉ thêm tối đa $\log_2(5e5)$ bit, nên thao tác tạo trie mới chỉ mất ĐPT $O(\log_2(5e5))$.
- Với truy vấn 3 x, ta lưu một tổng sum_xor của mọi số, khi có một truy vấn 3 x, ta chỉ cần lấy sum_xor = sum_xor ^ x, khi đó giá trị thực của phần tử thứ i là a_i ^ sum_xor, nếu có truy vấn 1 x thì thay vì thêm số x vào trie, ta phải thêm số x ^ sum_xor.

- Các truy vấn 4, 5, 6 ta dễ dàng thực hiện như trên trie bình thường.
- Code mẫu: https://ideone.com/aGJLny
-

SOLUTION 02/10

Bài 1:

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

using namespace std;

long long sumn2(long long v) {
  return v * (v + 1) * (2 * v + 1) / 6;
}

int main() {
  int N; cin >> N;
  vector<long long> A(N);

  int c = 0;
  for (int i = 0; i < N; i++) {
    cin >> A[i];
    c = max(0ll, c + A[i] - 1);
  }

  for (int i = 0; ; i++) {
    if (c == 0) {
      rotate(A.begin(), A.begin() + i, A.begin() + N);
      break;
    }
    c = max(0ll, c + A[i] - 1);
  }


  long long result = 0;
  for (int i = 0; i < N; i++) {
    result += sumn2(A[i] + c - 1) - sumn2(c - 1);
    c = max(0ll, c + A[i] - 1);
  }
  cout << result << endl;
}
```

Bài 2

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

using namespace std;

#define MAXN 1010
#define MAXK 10
#define INF 0x3FFFFFFFFFFFFFFFLL

int N, K;
long long A[MAXN];
long long DP[MAXK][MAXN];

int main() {
  cin >> N >> K;
  for (int i = 0; i < N; i++) {
    cin >> A[i];
  }

  long long result = INF;
  for (int i = 0; i < N; i++) {
    memset(DP, 0x3F, sizeof(DP));
    DP[0][N] = 0;

    for (int k = 1; k <= K; k++) {
      for (int j = 0; j < N; j++) {
        long long val = 0;
        for (int a = j + 1; a <= N; a++) {
          val += A[a - 1] * (a - j - 1);
          DP[k][j] = min(DP[k][j], val + DP[k - 1][a]);
        }
      }
    }
    result = min(result, DP[K][0]);
    rotate(A, A + 1, A + N);
  }
  cout << result << endl;
}
```

Bài 3

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

using namespace std;

int main() {
  int A, B, N, M;
  cin >> A >> B >> N >> M;

  vector<int> VA(N + 1), HA(M + 1);
  for (int i = 0; i < N; i++) {
    cin >> VA[i];
  }
  for (int j = 0; j < M; j++) {
    cin >> HA[j];
  }

  sort(VA.begin(), VA.end());
  for (int i = 0; i < N; i++) {
    VA[i] = VA[i + 1] - VA[i];
  }
  VA[N] = A - VA[N];

  sort(HA.begin(), HA.end());
  for (int i = 0; i < M; i++) {
    HA[i] = HA[i + 1] - HA[i];
  }
  HA[M] = B - HA[M];

  sort(VA.begin(), VA.end());
  sort(HA.begin(), HA.end());
  N++; M++;

  long long result = 1ll * VA[0] * (M - 1) + 1ll * HA[0] * (N - 1);
  for (int vi = 1, hi = 1; vi < N && hi < M; ) {
    if (VA[vi] < HA[hi]) {
      result += VA[vi++] * (M - hi);
    } else {
      result += HA[hi++] * (N - vi);
    }
  }
  cout << result << endl;

  return 0;
}
```

SOLUTION 03/10

Bài 1

Dễ

Bài 2

Sắp xếp topo, cần lưu ý các con bò không phục thuộc có thể thực hiện cùng lúc.

- Dùng DS liên kết A[i] sẽ lưu tất cả các con bò mà i phụ thuộc.
- Như vậy để tính time hoàn thành con bò i: Time[i] = max{time[j]} + T[i], với j là những con bò mà i phụ thuộc.
- Kết quả cuối cùng là thời gian của con bò có time lớn nhất.

For i:=1 to n do res = max(res, get(i))
Hàm get(i):
If time[i] = -1 then //con bò i chưa tính
{
star = 0
While A[i].next <> nil do
{j := A[i].next;
Star := max(star, get(j)) ;
}
time[i] = star + T[i];
}
Exit(time[i]);

Bài 3

This problem is solved by dynamic programming. Let good[n][B_2][B_3] be "true" if one can reach a state using only bales 1..n in which B_2 and B_3 denote the total amount of hay in barns 2 and 3 (just as in the problem statement). Note that we don't need to include B_1 in our state space, since B_1 is determined by knowing n, B_2, and B_3: B_1 is the total size of bales 1..n minus B_2 minus B_3. We now fill in the table of all good[][][] values, and the solution is obtained by looking at good[N][][] for the "true" entry with the smallest value of max(B_1,B_2,B_3). To make the algorithm as memory-efficient as possible, we note that we only need to store two "levels" of the table (for n and n-1) at a time. Furthermore, we know that the optimal solution is at most the sum of all bale sizes divided by 3, so we can upper bound the optimal solution by at most 700, limiting the maximum necessary size of dimensions 2 and 3 in our state space to 700 and speeding up the running time considerably.

```
#include <iostream>
#include <fstream>

using namespace std;
const int MAXS = 700;
```

```
ifstream fin ("baleshare.in");
ofstream fout ("baleshare.out");

int n, bale, tsum;
bool good[2][MAXS+100][MAXS+100];

int main()
{
    for (int i = 0; i < 2; i++)
        for (int j = 0; j < MAXS; j++)
            for (int k = 0; k < MAXS; k++)
                good[i][j][k] = false;
    good[0][0][0] = true;
    tsum = 0;

    fin >> n;
    for (int i = 0; i < n; i++)
    {
        fin >> bale;
        tsum += bale;
        for (int j = 0; j < MAXS; j++)
            for (int k = 0; k < MAXS; k++)
            {
                if (good[i%2][j][k])
                {
                    good[(i+1)%2][j][k] = true;
                    good[(i+1)%2][j+bale][k] = true;
                    good[(i+1)%2][j][k+bale] = true;
                }
            }
    }

    int ans = MAXS;
    for (int i = 0; i < MAXS; i++)
        for (int j = 0; j < MAXS; j++)
            if (good[n%2][i][j])
                ans = min (ans, max (i, max (j, tsum - (i + j))));
    fout << ans << "\n";
    return 0;
}
```

SOLUTION 04/10

Bài 1. Sort, duyệt

Xét phạm vi của bức ảnh hiện tại là (x,y). Nếu có một cặp (a,b) (giả sử a < b) không thích nhau trong vùng này, thì ảnh sẽ thu hẹp phạm vi thành (x,b-1). Tương tự như vậy ta có thể lặp lại các bức ảnh hiện tại thông qua các cặp không thích nhau và thu hẹp vị trí kết thúc.

Bài 2. Sort, duyệt

- Đầu tiên đếm có bao nhiêu ID khác nhau (S).
- Sắp xếp theo vị trí đại biểu.
- Duyệt từ đại biểu đầu tiên, nếu đại biểu tiếp theo có ID giống thì bỏ qua cho đến khi gặp ID khác... chọn vị trí đầu tiên của bức ảnh. Để chọn vị trí cuối của bức ảnh, ta cứ việc bổ sung cho đến khi đủ S ID là được.
- Lưu ý: dùng mảng đánh dấu để biết ID của đại biểu nào đã chọn rồi.

Độ phức tạp O(NlogN) do chí phí sắp xếp.

Bài 3. Quy hoạch động

Nhận xét: khi ta đặt một trạm wifi, nó sẽ phủ sóng trên một đoạn các con bò, và những con bò còn lại được chia thành hai bài toán con độc lập (một đoạn bên trái và một đoạn bên phải của trạm đã lắp). Vì vậy ta có thể áp dụng giải thuật quy hoạch động.

Vì mỗi đoạn bao phủ nên bắt đầu và kết thúc tại một vị trí có con bò đang đứng, nên nhiều nhất là có n^2 vị trí có thể đặt các trạm wifi. Vì mỗi bài toán con có thể bắt đầu và kết thúc tại một con bò, nên có n^2 bài toán có thể xem xét. Vì vậy dẫn tới độ phức tạp O(n^4), quá lớn.

Một nhận xét tiếp theo là: một vài trạm sẽ phải phủ lên con bò ở biên trái (tận cùng bên trái), vì vậy ta có thể đặt ở nơi tốt nhất với con bò kế tiếp. Vì vậy chỉ có tối đa n vị trí đặt trạm (mỗi trạm bắt đầu và kết thúc trên 1 con bò). Khi đặt được 1 trạm, như vậy chỉ còn lại đoạn sau chưa phủ sóng, vì vậy chỉ có n bài toán con vần xét. (nói cách khác).

Gọi vị trí các con bò là x1 < x2 < … < xn. Gọi D[i] là chi phí để phủ sóng lên các con bò từ thứ 1 đến i. Ta có D[0] = 0, D[i] = min{D[j] + chi phí từ con bò j đến i, j = 1..i-1}. với chi phí từ con bò i đến j là: A+B*(x_j – x_i)/2.

DAY 05/10

GỢI Ý VỀ THUẬT TOÁN

Câu 1: *(6,0 điểm)* **Tam giác.**

Ba đoạn thẳng a, b, c là ba cạnh của tam giác khi: $a + b > c, a + c > b, b + c > a$.

Subtask 1: $1 \leq N \leq 100$.

Duyệt mọi bộ ba chỉ số (i, j, k) và kiểm tra các đoạn có chỉ số i, j, k là ba cạnh của tam giác và có các màu khác nhau. Độ phức tạp: $O(N^3)$.

Subtask 2: $1 \leq N \leq 1500$.

Lưu độ dài của các đoạn có màu giống nhau vào một mảng $B[], G[], R[]$, khi đó độ dài trung bình của mỗi mảng là $M = N/3$. Bài toán đưa về việc chỉ kiểm tra điều kiện ba độ dài tạo nên tam giác $(B[i], G[j], R[k])$. Độ phức tạp: $O(M^3)$.

Subtask 3: $1 \leq N \leq 7500$.

Từ điều kiện: $a + b > c, a + c > b, b + c > a$ ta có $a + b > c, -c < a - b < c$ hay: $x = a + b > c$ và $y = |a - b| < c$. Khi đó nếu có $B[i], G[j]$ ta sẽ tìm tất cả các $R[k]$ thỏa mãn $y < R[k] < x$ trên mảng $R[]$ được sắp xếp bằng phương pháp tìm kiếm nhị phân. Độ phức tạp: $O(M^2 log M)$.

Subtask 4: $1 \leq N \leq 10000$.

Chọn mảng có nhiều phần tử nhất trong ba mảng $B[], G[], R[]$ để tìm nhị phân. Độ phức tạp như *subtask* 3 tuy nhiên sẽ giảm lượng lớn trong M^2.

Câu 2*: (7,0 điểm)* **Mật mã**

Subtask 1:

Với $S = a_1 \times a_2 \times ... \times a_n$ *và* $S \leq 10^{12}$ tính các ước của S với độ phức tạp $O(\sqrt{S})$.

Subtask 2:

Các giá trị a_i là các số nguyên tố, đáp án bài toán 2^n % $(10^9 + 7)$ với độ phức tạp $O(n)$.

Subtask 3:

Với $n \leq 10^6$ *v*à $a_i \leq 10^6$, số nguyên dương S được phân tích $S = a^p * b^q * c^t * ...$ khi đó ta có số ước của S chính là $(p + 1) * (q + 1) * (t + 1) * (...)$

Mỗi giá trị a_i phân tích thành thừa số nguyên tố như S ở trên, dữ liệu cho như đầu vào:

- **Cách 1:** Sàng nguyên tố sau đó phân tích thừa số nguyên tố để đếm phân phối các thừa số nguyên tố của a_i với độ phức tạp $O(\max(nlogn, a_i loga_i)$.
- **Cách 2:** Phân tích thừa số nguyên tố kết hợp đếm phân phối với độ phức tạp $O(\max(nloga_i)$.

Câu 3: *(7,0 điểm)* **Kế hoạch**.

Subtask 1: $N \leq 10, M \leq 6$.

Duyệt với mỗi học sinh ta gán một địa điểm có xe để đi bộ đến và sau đó sẽ đi xe về trường. Độ phức tạp: $O(M^N)$.

Subtask 2: $N \leq 14, M \leq 10^2$

Quy hoạch động. Gọi $f(mask, i)$ là chi phí nhỏ nhất phải trả khi đưa i học sinh đầu tiên về trường, thuê xe j nếu bit thứ $j-1$ trong $mask$ là 1. Từ các thông tin được lưu trong $mask$ ta có:

$f(mask, i) = \infty$ nếu với mỗi xe j được chọn ($bit\ j$ bằng 1) trong $mask$ nhưng $x_i < y_j$, ngược lại

$f(mask, i) = min(f(mask, i-1) + (x_i - y_j) \times v_i$, với mỗi xe j được chọn trong $mask$)

Khởi tạo $f(mask, 0) = c_{j_1} + c_{j_2} + \cdots + c_{j_p}$ ($j_1, j_2, \ldots, j_p$ là các bit bằng 1 trong $mask$) là chi phí thuê xe cho tất cả các xe được chọn thuê trong $mask$.

Nhận xét: ta thấy trong việc phân bổ tối ưu M học sinh vào các xe, nếu học sinh i đến xe j thì học sinh $i+1$ sẽ không đến xe có số hiệu bé hơn j (xa hơn - không tối ưu). Bởi vậy học sinh $i+1$ đến xe nào chỉ phụ thuộc vào xe mà học sinh i tới, không phụ thuộc vào các học sinh trước đó.

Subtask 3: $N \leq 10^3, M \leq 10^2$

Từ nhận xét trên (bài toán/trạng thái i chỉ phụ thuộc bài toán/trạng thái $i-1$) nên ta dùng quy hoạch động như sau: gọi $dp(i,j)$ là chi phí nhỏ nhất để đưa i học sinh đầu tiên về trường và học sinh i sẽ đi xe j. Từ nhận xét trên:

$dp(i,j) = \infty$ nếu $x_i < y_j$, ngược lại

$dp(i,j) = (x_i - y_j) \times v_i + min(dp(i-1,j), dp(i-1,j-1) + c_j, dp(i-1,j-2) + c_j, \ldots, dp(i-1,1) + c_j)$

Cần $O(N)$ để chuyển trạng thái, độ phức tạp chung sẽ là: $O(MN^2)$.

Subtask 4: $N \leq 2 \times 10^4, M \leq 10^3$

Tối ưu việc tính $dp(i,j)$. Công thức trên tương ứng với hai trường hợp:

- học sinh i đến cùng một xe với học sinh $i-1$, khi đó i sẽ không cần trả tiền xe
- học sinh i đến xe khác với xe học sinh $i-1$, khi đó i phải trả tiền xe

Công thức trên có thể viết lại:

$dp(i,j) = \infty$ nếu $x_i < y_j$, ngược lại

$dp(i,j) = (x_i - y_j) \times v_i + min(dp(i-1,j), \min(dp(i-1,j-1), dp(i-1,j-2), \ldots, dp(i-1,1)) + c_j)$

Trường hợp đầu không liên quan đến việc tính, để thực hiện trường hợp thứ hai cho mỗi trạng thái ta cần tính trước với $O(N)$. Gọi $l(i,j) = min(dp(i,1), dp(i,2), \ldots, dp(i,j))$, ta có:

$dp(i,j) = \infty$ nếu $x_i < y_j$, ngược lại:

$dp(i,j) = (x_i - y_j) \times v_i + min(dp(i-1,j), l(i-1,j-1) + c_j)$ và $l(i,j) = min(l(i,j-1), dp(i,j))$.

Khi đó chỉ cần $O(1)$ cho mỗi lần chuyển trạng thái, độ phức tạp chung sẽ là $O(NM)$. Kết quả của bài toán sẽ là: $l(1,N), l(2,N), \ldots, l(M,N)$. Khi lập trình sẽ phải cần bộ nhớ lớn, do

trạng thái hiện tại chỉ phụ thuộc vào trạng thái kế trước đó nên dùng ta sẽ dùng hai mảng đảo nhau.

----------HẾT----------

DAY 06/10

GỢI Ý VỀ THUẬT TOÁN

Câu 1: Đường truyền.

Subtask 1:

Để có thể tìm được chi phí nhỏ nhất kết nối mạng máy tính thì ta sử dụng thuật toán Kruskal tìm cây khung nhỏ nhất. Vì vậy nên ta cần duy trì 1 mảng các cạnh sắp xếp chi phí tăng dần. Khi thêm 1 cạnh vào trong tập cạnh, ta chỉ cần tìm vị trí đúng của nó ở trong mảng rồi thêm vào chứ không cần mất công sắp xếp lại cả mảng.

Độ phức tạp: $\boldsymbol{O(m * (n + m))}$.

Subtask 2:

Cũng sử dụng cây khung nhỏ nhất nhưng ta nhận thấy rằng mảng tập cạnh của chúng ta không cần phải lưu hết tất cả các cạnh mà chỉ cần lưu những cạnh của cây khung nhỏ nhất trước đó. Bởi vì nếu như 1 cạnh không được chọn lúc trước thì lúc sau cũng chắc chắn không được chọn. Vậy nên mỗi truy vấn ta chỉ cần xét tập gồm n cạnh.

Độ phức tạp: $\boldsymbol{O(m * n)}$.

Câu 2*: DU LỊCH*

Subtask 1:

Sử dụng quy hoạch động bằng Dijkstra với $dp[i][j][k][p]$ là số lần đổ xăng ít nhất để có thể đi đến thị trấn i, trong bình vẫn còn j lít xăng, còn k vé đổ xăng và p cho biết rằng đã từng nhận khuyến mãi hay chưa.

Mỗi khi đến 1 thị trấn ta có thể đi tiếp hoặc là đổ xăng rồi mới đi. Nếu như $p = false$ nghĩa là chưa từng nhận khuyến mại thì ta có thể nhận khuyến mại ở đây rồi mới đi. Nếu $p = true$ nghĩa là đã nhận khuyến mại thì ta có thể đổ xăng bất cứ lúc nào.

Độ phức tạp: $\boldsymbol{O(n * x * a * log)}$.

Subtask 2:

Vì tất cả các $a[i]$ đều bằng nhau nên ta sẽ nhận khuyến mại tại thị trấn 1 luôn. Sau đó tìm đường đi ngắn nhất từ thị trấn 1 đến n.

Độ phức tạp: $\boldsymbol{O(n * log)}$.

Subtask 3:

Ta thấy rằng hành trình có thể được chia làm 2 đoạn:

- Đầu tiên là bắt đầu ở thị trấn 1 rồi ta đi đến thị trấn u nào đó để nhận khuyến mại tại đó.
- Sau đó tìm đường đi ngắn nhất để đi từ thị trấn u đến n.

Ban đầu ta chuẩn bị $dp[i]$ là cách đi đến thị trấn i sao cho đổ xăng ít nhất và trong bình còn nhiều xăng nhất có thể. Mảng $dijk[i]$ cho biết đường đi ngắn nhất từ thị trấn n tới thị trấn i (bằng với đường đi từ i tới n). Tiếp đó ta chỉ cần chọn được thị trấn u làm trung gian sao cho tổng của 2 đoạn đường này tối ưu nhất.

Độ phức tạp: $\boldsymbol{O(n * log)}$.

Câu 3: Điều kiện lỏng.

Ta sẽ đếm số cách chọn các bit riêng biệt xong nhân kết quả lại với nhau.

Nhận xét, nếu một vị trí trong dãy nằm được phủ bởi một đoạn l_i, r_i có giá trị 0 thì vị trí đó chắc chắn phải mang giá trị 0. Với các đoạn có giá trị 1 thì trong đoạn đó phải tồn tại ít nhất một vị trí mang giá trị 1.

SUBTASK 1: Các đoạn điều kiện tách biệt nhau -> Nếu một vị trí không nằm trong đoạn nào thì sẽ có 2 cách chọn, nếu nằm trong đoạn mang giá trị 0 thì chỉ có 1 cách chọn duy nhất, với đoạn mang giá trị 1 độ dài x, số cách chọn cho cả đoạn đó là 2^x -1.

Độ phức tạp: O(N)

SUBTASK 2: Quy hoạch động: dp_i là số dãy khác nhau mà vị trí 1 cuối cùng là i và tất cả các đoạn mang giá trị 1 ở đằng trước đều có ít nhất 1 số 1 trong đó.

- dp[0] = 1
- Nếu i nàm trong một đoạn mang giá trị 0 -> $dp_i = 0$
- dp[i] = $\sum_{j=p}^{i-1} dp[j]$ trong đó p là số nguyên không âm, nhỏ nhất thỏa mãn không tồn tại chỉ số h sao cho $p < l_h \le r_h < i$ và đoạn h mang giá trị 1, bởi vì khi đó đoạn h sẽ không có số 1 nào. Để tính số p ta có thể thêm dần các đoạn mang giá trị 1 và có cận phải r < I và lấy max(I) + 1.

Kết quả số cách chọn mỗi bit sẽ là $\sum_{j=p}^{n} dp[j]$

Độ phức tạp: $O(N^2 + M)$

SUBTASK 3: Tối ưu hàm dp trong SUBTASK 2 bằng tổng tiền tố và tính trước giá trị p là số nguyên không âm, nhỏ nhất thỏa mãn không tồn tại chỉ số h sao cho $p < l_h \le r_h < i$ và đoạn h mang giá trị 1 với mỗi vị trí i.

Độ phức tạp: O(N + M)

----------HẾT----------

SOLUTION 12/10

Bài 1. Đường đi dài (7 điểm)

Phân bổ điểm:

- Subtask 1 (30%): $1 \leq k \leq 6$;
- Subtask 3 (30%): $1 \leq k \leq 100$;
- Subtask 4 (40%): Không có thêm ràng buộc nào.

Thuật toán:

Đầu tiên ta có nhận xét: người ở tọa độ lớn hơn sẽ nhận chiếc bánh có tọa độ lớn hơn chiếc bánh của người ở tọa độ nhỏ hơn

Gọi f[i][j] là thời gian tối thiểu của bài toán khi xét i người đầu tiên và j chiếc bánh đầu tiên, ta có công thức truy hồi:

- F[i][j] = min(f[i][j], f[i][j - 1]);
- F[i][j] = min(f[i][j], max(f[i - 1][j – 1], cost(i, j));

Ở đây hàm cost(i, j) là chi phí của người thứ i di chuyển tới tọa độ của chiếc bánh thứ j sau đó di chuyển tới địa điểm p.

Độ phức tạp của thuật toán: O(n*k).

Bài 2. Đường đi trên cây (7 điểm)

Phân bổ điểm:

- Subtask 1 (20%): Giá trị của tất cả các đỉnh bằng 1.
- Subtask 2 (40%): $1 \leq n \leq 10^3$;
- Subtask 2 (40%): Không có thêm ràng buộc nào.

Thuật toán:

Xử dụng thuật toán đổi gốc ở trên cây.

Giả sử ta có gốc ban đầu là đỉnh 1. Ta sẽ duyệt lần đầu tiên để tính chi phí của tất cả đường đi bắt đầu từ 1. Công thức như sau:

```
void ffin(int u, int fu) {
  s[u] = 1;
  for (int v : adj[u]) {
    if (v == fu) continue;

    ffin(v, u);
    dp[u] = (dp[u] - dp[v] + mod) % mod;
    s[u] += s[v];
  }
  dp[u] = (dp[u] + w[u] * s[u] % mod) % mod;
}
```

Thực hiện duyệt lần thứ 2. Mỗi lần duyệt tới 1 đỉnh ta coi đỉnh đó làm gốc và tính lại giá trị của tất cả đường đi bắt đầu của đỉnh đó, có thể thực hiện bằng công thức đơn giản như sau:

```
    ll dpu = dp[u], dpv = dp[v], su = s[u], sv = s[v];
    s[u] = n - sv;
    s[v] = n;
    dp[u] = (dp[u] + dp[v] - w[u]*sv%mod + mod) % mod;
    dp[v] = (dp[v] - dp[u] + w[v]*s[u]%mod + mod) % mod;
```

Toàn bộ chương trình dfs lần 2 như sau:

```
void ffout(int u, int fu) {
  an = (an + dp[u] + mod) % mod;
  for (int v : adj[u]) {
    if (v == fu) continue;

    ll dpu = dp[u], dpv = dp[v], su = s[u], sv = s[v];
    s[u] = n - sv;
    s[v] = n;
    dp[u] = (dp[u] + dp[v] - w[u]*sv%mod + mod) % mod;
    dp[v] = (dp[v] - dp[u] + w[v]*s[u]%mod + mod) % mod;

    ffout(v, u);

    dp[u] = dpu;
    dp[v] = dpv;
    s[u] = su;
    s[v] = sv;
  }
}
```

Độ phúc tạp của thuật toán O (n).

Bài 3. Màu sắc trên cây (6 điểm)

Phân bổ điểm:

- Subtask 1 (20%): $1 \leq n \leq 100, 1 \leq l,\ r \leq 10^5$.
- Subtask 2 (23%): $1 \leq n \leq 1000, 1 \leq l,\ r \leq 10^5$.
- Subtask 3 (20%): $1 \leq l,\ r \leq 10^5$.
- Subtask 4 (20%): Không có truy vấn loại 3.
- Subtask 5 (17%): Không có thêm ràng buộc nào.

Thuật toán:

Vì những truy vấn đều là offline nên ta có thể rời rạc hóa L và R trong mỗi truy vấn. Khi đó ta có 2×10^5 số khác nhau và có thể sử dụng segment tree để có thể tìm kết quả.

Với truy vấn loại 1, 2 tương tự như "gán giá trị 1 hoặc 0 cho đoạn liên tiếp". Với truy vấn loại 3 ta sẽ đảo một đoạn liên tiếp.

Để tìm được kết quả ta tìm phần tử trái nhất mang giá trị 0: Khi đứng ở nút v, ta kiểm tra xem nếu nút con bên trái đã đủ giá trị 1 (đoạn quản lí của nút con không mang giá trị 0) ta sẽ đi xuống nút con bên phải. Nếu không thì đi xuống nút con bên trái.

Sử dụng lazy propagation để thực hiện $\log n$ với mỗi truy vấn.

Độ phức tạp của thuật toán O (n log n).

Code tham khảo

Bài 1:

Bài 2:

Bài 3:

SOLUTION 20/9

Bài 1. Robot (7 điểm)

+NX1: Nếu không đi chéo thì số bước là $n + m$: đi lên n bước và sang phải m bước

+NX2: Mỗi lần đi chéo thì số lần đi lên giảm 1, số lần sang phải giảm 1. Tổng số bước giảm 1.

Như vậy nếu có k lần đi chéo, thì tổng số bước là $n + m - k$.

Số cách chọn k bước chéo trong $n + m - k$ bước bằng C_{n+m-k}^{k}

Còn lại $n + m - 2k$. Số bước đi lên là $n - k$. Số cách chọn $n - k$ bước lên trong số $n + m - 2k$ bước bằng C_{n+m-2k}^{n-k}

Công thức:

$$\boldsymbol{DS} = \sum_{k=0}^{n}(C_{n+m-k}^{k} \times C_{n+m-2k}^{n-k})$$

Bài 2. Trò chơi (7 điểm)

Qui ước $s_i = 0$ nếu hạt màu đỏ và $s_i = 1$ nếu hạt màu xanh

Bằng kỹ thuật "nhân đôi mảng" Bài toán qui về cho đoạn xâu $s_u s_{u+1} \dots s_{u+n-1}$ hỏi rằng người đi trước thắng hay thua?

Bài toán được giải bằng phương pháp qui hoạch động:

Đặt $dp[i, j, b]$ = 1 nếu như trạng thái chơi là đoạn $s_i \dots s_j$ và người đi đầu tiên hiện có b hạt màu xanh. Khi đó:

+TH1: Nếu $b \geq k$ thì $dp[i, j, b] = 0$ (người đi đầu chắc chắn thua)

+TH2: Trường hợp $b < k$

- TH2a: Người đi đầu chọn bên trái. Khi đó người này đã có $b + s_i$ hạt màu xanh. Do vậy người kia có $b' = t_n - (b + s_i) - (t_j - t_i)$ hạt màu xanh. Do đó khả năng thắng của người kia là $dp[i + 1, j, b']$. Vậy khả năng thăng của người thứ nhất là $T1 = 1 - dp[i + 1, j, b']$
- TH2b: Người đi đầu chọn bên phải: Lập luận tương tự ta có $T2 = 1 - dp[i, j - 1, b']$ với $b' = t_n - (b + s_j) - (t_{j-1} - t_{i-1})$

Chú ý $t_i = s_1 + \cdots s_i$

Do đó:

$$dp[i, j, b] = \max(T1, T2)$$

Bài 3. Du lịch (6 điểm)

Sử dụng thuật toán Dijkstra, trong đó mảng khoảng cách sẽ thêm thông tin theo dõi vé bay khuyến mãi đã được sử dụng hay chưa.

- kc[i][false] sẽ biểu thị khoảng cách ngắn nhất từ nút bắt đầu đến nút i mà chưa sử dụng vé bay khuyến mãi.

- kc[i][true] sẽ biểu thị khoảng cách ngắn nhất sau khi sử dụng vé bay khuyến mãi.

219

Câu 1 *(7 điểm).* **Đàn kiến**

Bài này chỉ cần chú ý tới nhận xét: Khi hai con kiến gặp nhau, cả hai cùng quay đầu thì chỉ có sự khác biệt ở bản thân hai con kiến đó còn về tổng thể cả đàn không có sự thay đổi (giống như khi hai con kiến gặp nhau chúng cứ đi thẳng).

Do đó: Tổng quãng đường di chuyển của cả đàn kiến bằng tổng quãng đường mỗi con đi thẳng từ vị trí nó đứng đến biên (tọa độ 0 nếu nó quay về bên trái và tọa độ L nếu nó quay về bên phải).

ĐPT: $O(n)$

Code tham khảo (C++)

```
#include <bits/stdc++.h>
using namespace std;
#define task "ANTS"
typedef long long ll;
int main() {
    freopen(task".INP", "r", stdin);
    freopen(task".OUT", "w", stdout);
    ios::sync_with_stdio(0);
    cin.tie(0); cout.tie(0);

    int x, L, n;
    cin >> n >> L;
    ll res = 0;
    while (n--) {
        cin >> x;
        if (x < 0)
            res -= x;
        else
            res += L - x;
    }
    cout << res;
}
```

Câu 2 *(7 điểm).* **Lễ hội bơi chải**

Có thể phát biểu lại bài toán: Cho dãy số $a_1, a_2, \dots, a_n$ và số nguyên K ($K = 2k + 2, 2 \le K \le n$). Trong mỗi đoạn con gồm K phần tử liên tiếp chọn hai phần tử có tổng lớn nhất. Trong tất cả các đoạn con như vậy, tìm đoạn có tổng hai phần tử lớn nhất là lớn nhất.

Subtask #1: Có 60% số điểm của bài có $2 \le 2 \times k + 2 \le n \le 1000$;

Đối với subtask này có thể giải quyết như sau:

+ Xét các số $1 \le d < K$:

+ Với mỗi số d như trên, xét các cặp hai phần tử a_i và a_{i+d}, tìm cặp có tổng lớn nhất.

ĐPT: $O(n \times K)$

Code tham khảo (C++):

```
#include <bits/stdc++.h>
using namespace std;
#define task "SWIMMING"
#define nmax 100001
int a[nmax], n, k;
```

```
int main() {
    freopen(task".INP", "r", stdin);
    freopen(task".OUT", "w", stdout);
    ios::sync_with_stdio(0);
    cin.tie(0); cout.tie(0);

    cin >> n >> k;
    for (int i = 1; i <= n; i++)
        cin >> a[i];
    int K = 2 * k + 2;
    int res = -1e9;
    for (int d = 1; d < K; d++)
        for (int i = 1; i + d <= n; i++)
            res = max(res, a[i] + a[i + d]);
    cout << res;
}
```

Subtask #2: Có 40% số điểm còn lại không có ràng buộc bổ sung.\
Để giải quyết Subtask này ta có thể dùng cây IT để truy vấn cặp có tổng lớn nhất trong mỗi đoạn con.

ĐPT: $O(nlogn)$

Code tham khảo (C++)

```
#include <bits/stdc++.h>
using namespace std;
#define task "SWIMMING"
vector<int> a;
int n, k;
struct IT {
    struct node {
        int m, s;
        node(): m(-1e9), s(-1e9){};
        node(int x): m(x), s(-1e9) {};
        node(node l, node r) {
            m = max(l.m, r.m);
            s = max(max(l.s, r.s), l.m + r.m);
        }
    };
    vector<node> it;
    IT(vector<int>& a) {
        int n = a.size();
        it.resize(4 * n);
        buildIT(1, 0, n - 1, a);
    }
    void buildIT(int i, int l, int r, vector<int>& a) {
        if (l == r)
            it[i] = node(a[l]);
        else {
            buildIT(i * 2, l, (l + r)/2, a);
            buildIT(i * 2 + 1, (l + r)/2 + 1, r, a);
            it[i] = node(it[i * 2], it[i * 2 + 1]);
        }
    }
    node query(int i, int l, int r, int u, int v) {
        if (r < u || v < l) return node();
        if (u <= l && r <= v) return it[i];
        int il = i << 1, m = (l + r) >> 1;
        return node(query(il, l, m, u, v), query(il + 1, m + 1, r, u, v));
    }
};
```

```
int main() {
    freopen(task".INP", "r", stdin);
    freopen(task".OUT", "w", stdout);
    ios::sync_with_stdio(0);
    cin.tie(0); cout.tie(0);

    cin >> n >> k;
    a.resize(n);
    for (int& x : a)
        cin >> x;
    IT it(a);
    int res = -1e9;
    k = 2 * k + 2;
    for (int i = 0; i + k - 1 < n; i++)
        res = max(res, it.query(1, 0, n - 1, i, i + k - 1).s);
    cout << res;
}
```

Câu 3 *(6 điểm).* **Tam sao thất bản**

Gọi n là độ dài xâu s.

Nhận xét: Xét một cách copy xâu s thành xâu s' bằng cách đảo một đoạn con $s[i..j]$ của xâu s. Nếu $s[i] = s[j]$ thì xâu s' hoàn toàn có thể có được bằng cách đảo đoạn $s[i+1..j-1]$ của sâu s, do đó các copy này không làm tăng thêm số xâu copy được.

Từ đó có thể thấy được kết quả của bài toán là: $1 +$ số cặp (i,j) mà $1 \le i < j \le n$ và $s[i] \ne s[j]$

Subtask #1: Có 60% số điểm của bài có độ dài xâu s không quá 1000;

Subtask này có thể giải quyết bằng hai vòng lặp với độ phức tạp $O(n^2)$.

Code tham khảo (C++)

```
#include <bits/stdc++.h>
using namespace std;
#define task "COPY"
typedef long long ll;
int main() {
    freopen(task".INP", "r", stdin);
    freopen(task".OUT", "w", stdout);
    string s;
    cin >> s;
    int n = s.size();
    ll res = 1;
    for (int i = 0; i < n - 1; i++)
        for (int j = i + 1; j < n; j++)
            res += s[i] != s[j];
    cout << res;
}
```

Subtask #2: Có 40% số điểm còn lại có độ dài xâu s không quá 10^5.

Để giải quyết subtask này ta cần giảm độ phức tạp xuống. Với việc sử dụng một mảng đếm phân phối các ký tự của xâu s, với mỗi vị trí i (khi xét từ trái sang phải) ta dễ dàng đếm được có bao nhiêu vị trí j trước i mà có $s[i] == s[j]$. Từ đó ta giảm độ phức tạp xuống còn $O(n)$.

Code tham khảo (C++)

```
#include <bits/stdc++.h>
using namespace std;
#define task "COPY"
```

```cpp
typedef long long ll;
int main() {
    freopen(task".INP", "r", stdin);
    freopen(task".OUT", "w", stdout);
    ll c[255];
    string s;
    cin >> s;
    memset(c, 0, sizeof c);
    ll res = 1;
    for (int i = 0; i < s.length(); i++) {
        res += (i - c[s[i]]);
        c[s[i]]++;
    }
    cout << res;
}
```

----------HẾT----------

SOL 22/9

Bài 1. DÃY SỐ (7 điểm)

Phân bổ điểm:

- Subtask 1 (60%): Duyệt bằng các vòng lặp – Độ phức tạp là $O(n^2.k)$
- Subtask 2 (40%):
- Ta xây dựng mảng c: c[i]=max(a[i]+a[i+1], a[i]+a[i+2])
- Với mỗi truy vấn [l,r] ta tìm max của mảng c trên đoạn [l, r-1] so với $a_{l-1}+ a_l$.
- Dùng cấu trúc cây để tìm max trên mảng c.

Bài 2. Mua nhà (7 điểm)

Phân bổ điểm:

- Subtask 1, 2 (30% + 30%): Dùng đệ qui tạo dãy nhị phân.
- Subtask 3 (20%): Dùng qui hoạch động dãy con có tổng bằng S.
- Subtask 4 (20%):
  - Duyệt loại bỏ từng phần tử rồi áp dụng subtask 3.

Bài 3. Màu sắc trên cây (6 điểm)

Phân bổ điểm:

- Subtask 1 (20%): $1 \leq n \leq 100, 1 \leq l,\ r \leq 10^5$.
- Subtask 2 (23%): $1 \leq n \leq 1000, 1 \leq l,\ r \leq 10^5$.
- Subtask 3 (20%): $1 \leq l,\ r \leq 10^5$.
- Subtask 4 (20%): Không có truy vấn loại 3.
- Subtask 5 (17%): Không có thêm ràng buộc nào.

Thuật toán:

Vì những truy vấn đều là offline nên ta có thể rời rạc hóa L và R trong mỗi truy vấn. Khi đó ta có 2×10^5 số khác nhau và có thể sử dụng segment tree để có thể tìm kết quả.

Với truy vấn loại 1, 2 tương tự như "gán giá trị 1 hoặc 0 cho đoạn liên tiếp". Với truy vấn loại 3 ta sẽ đảo một đoạn liên tiếp.

Để tìm được kết quả ta tìm phần tử trái nhất mang giá trị 0: Khi đứng ở nút v, ta kiểm tra xem nếu nút con bên trái đã đủ giá trị 1 (đoạn quản lí của nút con không mang giá trị 0) ta sẽ đi xuống nút con bên phải. Nếu không thì đi xuống nút con bên trái.

Sử dụng lazy propagation để thực hiện $\log n$ với mỗi truy vấn.

Độ phức tạp của thuật toán O (n log n).

---------- Hết ----------

SOLUTION DAY 25/9

Bài 1: Tìm kiếm văn bản.

Thuật toán 1. Tính trực tiếp, sử dụng các hàm và thủ tục chuẩn (copy(), pos()) Độ phức tạp $O(N^3)$, 30% số điểm.

Thuật toán 2. Chỉ tính với xâu độ dài $N/2$ và tìm bằng nhị phân, độ phức tạp $O(N^2 \log N)$, hoặc tính và cập nhật dần các đoạn có độ dài cần tìm, độ phức tạp $O(N^2)$ 30% số điểm

Thuật toán 3. Dùng kỹ thuật hai con trỏ (trái, phải) để bỏ vòng lặp trong cho thuật toán 2: ($O(N^2)$), độ phức tạp chỉ còn $O(N)$. 100% số điểm.

Bài 2. Hương chè

Thuật toán: Tìm kiếm nhị phân, chặt nhị phân kết quả, duyệt

Bài toán yêu cầu tìm số K (số chè lấy hương hằng ngày) ít nhất, sao cho An có thể thực hiện lấy hương được ít nhất X% tổng số chè ban đầu.

Lưu ý: trong bài này các phép tính với % là các phép tính trên số thực, chỉ làm tròn xuống số nguyên trong thao tác cuối cùng.

N × X% là một số thực, ví dụ: N=43, X = 60 ➔ Lượng chè mục tiêu cần lấy hương tối thiểu là 43 * 60% = 25.8 kg, chứ không phải 25 kg

Sub 1: Với P = 0 thì kết quả bài toán luôn là 1

Sub2: K tối thiểu bằng 1, K tối đa chính là N (thực hiện lấy hương chè 1 ngày duy nhất là xong)

Duyệt tìm K từ 1 đến N, thử tìm giá trị K đầu tiên thoả đề bài.

ĐPT: O(N)

Sub3: Dùng tìm kiếm nhị phân để tìm K trong đoạn từ 1 đến N

ĐPT: O(logN)

Bài 3. PAINT

Sử dụng hàng đợi ưu tiên và giải thuật tham lam. ☺

CODE THAM KHẢO

Bài 1.

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

using namespace std;

const int MAXN = 10000001;

string s;
int bl[MAXN], n;

int cmp(int p1, int p2) {
   int j, t;
   if (p1>=n || p2>=n) return 0;
   if (p1>p2) t = n-p1-1;
   else t=n-p2-1;
   j=0;
   while(j<=t && s[p1+j]==s[p2+j]) j++;
   return j;
}

int main() {

   ios::sync_with_stdio(false); cin.tie(NULL);
   freopen("findnext.inp", "r", stdin);
   freopen("findnext.out", "w", stdout);

   cin >> s;
   n = s.size();
   int ans=0, pos=0, l=0, r=0;
   bl[0]=0;
   for(int i=1; i<n; i++) {
      bl[i]=0;
      if(i>r) {
         bl[i]=cmp(0,i);
         if (bl[i]>0) {
                  l=i;
                  r = i+bl[i]-1;
         }
      }
      else {
         int k=i-l;
         if (bl[k]<r-i+1) bl[i]=bl[k];
         else {
            bl[i]=r-i+1;
            l=i;
```

```
            int q=cmp(r-i+1,r+1);
            if (q>0) {bl[i]+=q; r=i+bl[i]-1;}
          }
        }

                int z = min(bl[i],i);
        if(z>ans) {ans=z; pos=i;}
      }

          cout << ans << " " << pos+1 << "\n";

          return 0;
    }
```

Bài 2.

```
    #include <bits/stdc++.h>
    #define nmax uint64_t(1e18)
    #define ll long long
    using namespace std;
    ll n,x,p, res = nmax;
    long double m;
    bool check(ll k)
    {
      ll s = 0, nn = n;
      while (nn > 0)
      {
        if (nn <= k)
        {
          s += nn;
          nn = 0;
        }
        else{
          nn -= k; s+=k;
          if (nn >= p)
            nn = nn - (ll)(p/100.0 * nn);
          if (nn >= 2*p)
            nn = nn - (ll)(p/200.0 * nn);
        }
      }
      return (s >= m);
    }
    int main()
    {
      freopen("HuongChe.inp","r",stdin);
      freopen("HuongChe.out","w",stdout);
      cin>>n>>x>>p; m = x/100.0 * n;
      ll L = 1, R = n;
      while (L <= R){
```

```
        ll mid = (L+R)/2;
        if (check(mid) == true)
        {
            R = mid -1;
            res = min(res,mid);
        }
        else
            L = mid +1;
    }
    cout<<res;
    return 0;
}
```

Bài 3.

```
#include <bits/stdc++.h>
#define ll long long
using namespace std;
const int N=1000005;
ll n,x, res,kor,last;
ll qpos,qkr,minqpos,minqkr,maxqpos,maxqkr;
ll vis[N], h[N],q[N],minq[N],maxq[N];

void readinp(){
    cin>>n>>x;
    for(int i=1;i<=n;i++) cin>>vis[i];
}
void Push(ll a){
    q[qkr]=a;
    qkr++;
    while(minqkr>minqpos && minq[minqkr-1]>a) minqkr--;
    while(maxqkr>maxqpos && maxq[maxqkr-1]<a) maxqkr--;
    minq[minqkr]=a; minqkr++;
    maxq[maxqkr]=a; maxqkr++;
}
void Pop(){
    if(minq[minqpos]==q[qpos]) minqpos++;
    if(maxq[maxqpos]==q[qpos]) maxqpos++;
    qpos++;
}
ll getmax(){
    return(maxq[maxqpos]);
}
ll getmin(){
    return(minq[minqpos]);
}
void init(){
    qpos=1; qkr=1;
    minqpos=1; minqkr=1;
```

```
    maxqpos=1; maxqkr=1;
}
int main()
{
    freopen("PAINT.INP","r",stdin);
    freopen("PAINT.OUT","w",stdout);
    res=0; kor=1;last=1;
    init();
    readinp();
    for(int i=n;i>=n-x+2;i--){
        Push(vis[i]);
        h[i]=0;
    }
    for(int i=n-x+1;i>=1;i--){
        Push(vis[i]);
        h[i]=getmin();
        Pop();
    }
    vis[n+1]=0;
    h[n+1]=0;
    for(int i=1;i<=n-x+1;i++){
        if(last+x <= i || h[i]>h[last] || vis[i+x]<h[i]){
            last=i;
            kor++;
        }
    }
    init();
    for(int i=1;i<=n;i++){
        if(i<=n-x+1) Push(h[i]);
        if(i>x) Pop();
        res = res + vis[i] - getmax();
    }
    cout<<res<<'\n';
    cout<<kor;
    return 0;
}
```

Bài 1: (7 điểm)

Subtask 1: $n \leq 8$: Duyệt n^6

Subtask 3: $n \leq 5\,000$ và đáp án bài toán tìm được chỉ bằng cách sử dụng phép biến đổi trên một phần tử của dãy số:
Với sub này chỉ cần xét 2 phần từ liên tiếp $a[i]$ và $a[i+1]$

Subtask 2: $n \leq 1000$

- Gọi $F[i]\,[j]$ giá trị tối ưu xét i phần tử đầu tiên của dãy và sự thay đổi giá trị sao cho phần tử thứ i bằng j.
- Dễ dàng nhận thấy rằng giải pháp tối ưu là các giá trị thay đổi của các phần tử của chuỗi đã cho nằm trong khoảng từ $a1 = min\, a\,[i]$ đến $a2 = max\, a\,[i]$. Do đó F[0] [j] = $abs\ (a\ [0]\ - j)$.
- Khi tính được $F[0]\,[j], F[1]\,[j], \dots, F[i\ -\ 1]\,[j]\ \ \forall j\ =\ a1, \dots, a2$, chúng ta có thể tính $F[i]\,[\,j]$ cho mỗi $j\ =\ \mathrm{a1}, \dots, \mathrm{a2}$

$$F[i][\,j] = \min\{F[i\ -\ 1]\,[jj]\ +\ abs\ (a\,[i\,]\ -j)\,\}\ \ (j \geq jj)$$

- Đáp án: giá trị nhỏ nhất $F[n\ -\ 1][j],\ j\ =\ \mathrm{a1}, \dots, \mathrm{a2}$
- Độ phức tạp $O(n^2)$

Subtask 2: $n \leq 100\,000$

- Sử dụng hàng đợi ưu tiên $priority\_queue$.
- Chúng ta $push$ tuần tự các phần tử của dãy vào trong $priority\_queue$ và kiểm tra xem phần tử lớn nhất của $priority\_queue$ có lớn hơn phần tử của dãy mà chúng sẽ $push$.
- Nếu $(priority\_queue.top\ ()\ >\ a\ [i])\ res+=\ (priority\_queue.top\ ()\ -\ a\ [i])$,
- Độ phức tạp $o(nlogn)$

Bài 2: (7 điểm)

- Xóa tất cả các cầu khỏi đồ thị.
- Đồ thị mới tạo thành các thành phần liên thông. Dễ thấy nếu một cặp đỉnh thỏa mãn đề bài, thì chúng sẽ nằm trong một thành phần liên thông và ngược lại.
- Nếu một thành phần liên thông có k đỉnh thì kết quả bài toán được cộng thêm

$$res+= \frac{k(k-1)}{2}$$

- Độ phức tạp $O(N\ +\ M)$.

Bài 3: (6 điểm)

Lưu ý: *Trong trường hợp thí sinh chỉ in kết quả -1 thì không được điểm còn trường hợp subtask 2: xâu nhập vào chỉ có 1 kí tự* $'0'$ *nếu học sinh phát hiện được kết quả -1 thì vẫn được điểm.*

Subtask 1: $n \leq 8$

- ✓ Nhận xét nếu có đáp án, thì luôn tồn tại dãy thỏa mãn mà không cần dùng quá n giá trị.
- ✓ Duyệt trâu độ phức tạp n^{n+1}.

Subtask 2: xâu nhập vào chỉ có 1 kí tự $'0'$:

- ✓ Nhận xét nếu vị trí i là 0 thì phải tồn tại ít nhất một vị trí j khác cũng là 0 nên để có đáp án thì phải tồn tại ít nhất 2 vị trí 0.
- ✓ Suy ra subtask này chỉ cần in -1.

Subtask 3,4,5:

- ✓ 3 Subtask này dùng cùng một ý tưởng quy hoạch động, nhưng vì những giới hạn đặc biệt nên subtask 3,4 không cần tối ưu tốt như subtask 5.
- ✓ Mấu chốt của bài này là nhận ra rằng dấu của một vị trí phụ thuộc vào số lượng số nhỏ hơn nó, nếu có chẵn số nhỏ hơn thì là $+$, còn có lẻ số nhỏ hơn thì là $-$.
- ✓ Quy ước các vị trí có giá trị bằng nhau ta gọi là một "$nhóm$".
- ✓ Các phần tử trong $nhóm$ thứ i sẽ mang giá trị i.
- ✓ Nhận xét là một nhóm chỉ có thể nằm trong 3 trường hợp:
  1. Chỉ gồm 1 kí tự $+$.
  2. Chỉ gồm 1 kí tự $-$.
  3. Gồm ít nhất 2 kí tự 0.
- ✓ Gọi A, B, C lần lượt là tổng số lượng các kí tự $+,0,-$.
- ✓ Quy hoạch động $f[i][j][t][k]$ là dãy thõa mản có thứ tự từ điển nhỏ nhất mà đang xét đến nhóm thứ i, đã dùng j kí tự $+$, t kí tự 0 và k kí tự $-$.
  1. Ban đầu, $f[0][0][0][0]$ là một dãy gồm n giá trị $n+1$.
  2. Từ $f[i][j][t][k]$, ta sẽ xét các phần tử của nhóm giá trị thứ $i+1$.
  3. Nếu ta quyết định đặt cho nhóm $i+1$ dấu $+$ hoặc $-$ thì phải xét số lượng giá trị nhỏ hơn nó, chính là $j+t+k$. Nếu $j+t+k$ chẵn thì ta mới được chọn dấu $+$ và lẻ thì ta mới được chọn dấu $-$. Hai cách này lần lượt cập nhật cho $f[i+1][j+1][t][k]$ và $f[i+1][j][t][k+1]$.
  4. Nếu ta quyết định đặt cho nhóm $i+1$ kí tự 0 thì $f[i][j][t][k]$ sẽ cập nhật cho $f[i+1][j][p][k]$ với $t+1 < p \leq B$ (vì phải nhóm gồm toàn 0 phải có ít nhất 2 phần tử).
  5. Đáp án của bài toán chính là $\min(f[i][A][B][C])$ với $1 \leq i \leq n$.
  6. Cách trên dùng n^5 bộ nhớ với độ phức tạp là n^6.
- ✓ Nghe độ phức tạp có vẻ sẽ không chạy qua nhưng đây lại là thuật chuẩn của bài toán này.
  1. Số trạng thái (n^4): Vì trong mọi trạng thái của quy hoạch động, $i+j+t \leq n$ nên số trạng thái không bao giờ đạt đến n^4, trong bài này thì sẽ rơi vào khoảng $\frac{n^3}{10}$, giảm độ phức tạp xuống n^5 với hằng số nhỏ, thừa qua subtask cuối.
  2. Bộ nhớ (n^5): n^5 thì sẽ không thể qua được giới hạn bộ nhớ. Nhận thấy $f[i+1]$ chỉ được cập nhật bởi $f[i]$ nên có thể bỏ bớt một chiều giá trị, giảm bộ nhớ xuống n^4.
- ✓ Sau khi tối ưu xong thì thuật toán trên sử dụng n^4 bộ nhớ và độ phức tạp n^5.

SOLUTION DAY 27/9

Sol TOMMY

Sub 1: O(m.n.y) với y lá số lượng số nguyên tố trong đoạn [l,r]

B1) Đọc file và xác định giá trị c[i] là số lần xuất hiện của giá trị i trong dãy số.

B2) Dùng sàng Eratosthenes để xác định các số nguyên tố trong đoạn $[1..10^7]$

B3) Với mỗi truy vấn trong m truy vấn, ta lần lượt xét từng số nguyên tố i trong đoạn [li,ri].

- Với mỗi số nguyên tố i, ta duyệt lại mảng x và đếm số lượng bội của i là f(i);
- Tổng các f(i) chính là kết quả cần tìm.

Sub 2: O(max(m,n).y)

B1) Tương tự cách 1

B2) Dùng sàng số nguyên tố, kết hợp tính các giá trị f(i) (với i là số nguyên tố trong đoạn $[1, 10^7)$

- Tính f(2)? f(2) = c[2]+ c[4] + c[6] + c[8],...
- Tính f(5)? f(5) = c[5] + c[10]+ c[15] + c[20],...
- Tính f(n)? f(n) = $c[n] + c[2 \cdot n] + c[3 \cdot n] + c[4 \cdot n]$,...

Ta thấy tư tưởng này rất giống với thuật toán sàng Eratosthenes. Vì vậy ta có thể dùng thuật toán sàng và chỉnh sửa lại. Kết quả tính sẽ được lưu trữ vào f[n]

B3) Với mỗi truy vấn trong m truy vấn, ta lần lượt xét từng số nguyên tố i trong đoạn [li,ri]. Ta tính tổng các f(i) chính là kết quả cần tìm.

Sub 3: O(max(m,n))

Để giải bài toán này ta cần giải quyết một số vấn đề sau:

B1, 2) Tương tự cách 2

B3) Bây giờ ta cần tính tổng tiền tố S của mảng num. Với S[i] = f[1] + f[2] + …+f[i]

B4) Sau khi tính tổng tiền tố xong, ta có thể tính toán tổng số lượng phần tử giữa l và r trong thời gian O(1), nghĩa là ta tính s[r] – s[l-1]. Bây giờ ta có thể đọc các truy vấn và trả lời chúng dễ dàng.

Cần lưu ý là cận phải r có thể lớn hơn 10^7, vì vậy ta có thể giải r xuống chỉ còn 10^7 thôi và tất cả các số được cho đều bé hơn hoặc bằng 10^7.

Sol CROAD

- Xác định vector nghiệm: (x_1, x_2) trong đó $x_1 \in$ [1, N-1], $x_2 \in$ [x1+1, N] là hai chỉ số của dãy số A.

Sub 1: O(N^3)

- Viết hàm check(int x1, int x2): Có chức năng kiểm tra giá trị trung bình đoạn [x1,x2] có bằng k không. Nếu bằng k và thỏa mãn là đoạn dài nhất, ta cập nhật kết quả u, v. Ta tốn O(n)

Thuật toán:

- Sử dụng 2 vòng lặp để duyệt qua tất cả các cặp (x1,x2). Với mỗi cặp giá trị ta cập nhật kết quả bằng cách gọi hàm check(x1,x2)
- Độ phức tạp: O(n^3)

Sub 2: O(N^2)

- Giảm độ phức tạp của hàm check(x1,x2) ở trên xuống còn O(1) sử dụng mảng tính tổng tích lũy đã học. Ta có: sum[i] = a[1]+a[2]+…+a[i]

➔Sử dụng 1 vòng lặp để tính mảng sum[]

- Thuật toán vẫn tương tự ở cách 1.
- Độ phức tạp: O(n^3)

Sub 3: O(N)

Nhận xét:

- Ta tiến hành trừ giá trị mỗi phần tử a[i] đi k đơn vị. Nghĩa là a[i] = a[i] – k.
- Khi đó, ta có nếu đoạn [x1, x2] có giá trị trung bình bằng k, thì sum[x2] – s[x1-1] = 0.

Thật vậy:

- Giả sử đoạn [3, 6] có giá trị trung bình là k: $\frac{a[3]+a[4]+a[5]+a[6]}{4} = k \leftrightarrow a[3] + a[4] + a[5] + a[6] = 4k$
- Khi giảm mỗi phần tử đi k đơn vị:

$$a[3] - k + a[4] - k + a[5] - k + a[6] - k = a[3] + a[4] + a[5] + a[6] - 4k = 4k - 4k = 0$$

Thuật toán:

1/ Ta thực hiện trừ giá trị mỗi phần tử a[i] đi k đơn vị.
2/ Tính mảng tổng tích lũy sum[] trên dãy a mới.
3/ Tìm đoạn [x1,x2] có tổng = 0 dựa trên mảng sum đã tính:
a. Ta sắp xếp giá trị mảng sum tăng dần theo giá trị. Nếu trùng giá trị ta sắp tăng dần theo chỉ số i tương ứng ➔ Phải dùng 1 cấu trúc để lưu trữ cặp giá trị <sum, i>. Khi hoán đổi thì hoán đổi cả 2 (có thể dùng pair<long long,int> hoặc struct. Tuy nhiên dùng struck phải viết một hàm so sánh cmp(struct a, struct b) theo tiêu chí đã nêu để sử dụng khi dùng hàm sort)

- Giả sử vector cấu trúc: vector<pair<long long, int> > f; //lưu ý phải dùng long long lưu sum nhé!

➔ Để tạo mảng f, trong lúc tính mảng sum[i] và thực hiện f.push_back({sum[i],i});
lưu ý: các em nên push thêm phần tử thứ 0 và n+1 để làm rào ở đầu, cuối dãy f. Nghĩa là

1. f.push_back({0,0});
2. Xây dựng mảng f từ mảng sum[i]: f.push_back({sum[i],i});
3. f.push_back({INT_MAX, n+1});

- Thực hiện sắp xếp mảng f với hàm sort: sort(f.begin(), f.end());

b. Duyệt lại dãy f[] đã sắp ở bước a. Sử dụng biến d để kiểm soát việc tìm 2 cận x1, x2.

- Nếu d=1, ta đang tìm x1
- Nếu d=0, ta đang tìm x2
- Gọi ix, lx: là chỉ số và chiều dài của dãy con lớn nhất thỏa yêu cầu đề bài.

Với i: 0 → n

nếu d=1 và f[i].first = f[i+1].first thì ta **gán x1 = i;** d = 0;// tìm x2

nếu d=0 và f[i].first != f[i+1].first thì ta **cập nhật ix và lx** và gán d=1; //tìm x1

Kết quả là cặp giá trị: ix+1 và lx

SOL_CHILLIS

Subtask 1: Với mỗi lựa chọn cắt cành, ta dùng dfs để tính số lượng phần tử trong mỗi thành phần liên thông.

Độ phức tạp: O(n^3).

Subtask 2: Ta lựa hai cành để cắt có thể được xem là lựa hai trái ớt mà ta sẽ cắt cành nối giữa nó và cha của nó.

Khi đã cắt được trái ớt x, coi độ lớn(số phần tử) cây con của x là S(x). Ta có 2 trường hợp cắt trái ớt y:

1. Trái ớt y là tổ tiên của trái ớt x, tức là trái ớt y nằm trên đường đi từ x đến gốc. Độ lớn của các thành phần sẽ là S(x), S(y) – S(x) và n – S(y).
2. Trái ớt y không là tổ tiên của x và x cũng không là tổ tiên của y. Độ lớn của các thành phần sẽ là S(x), S(y) và n – S(x) – S(y).

Độ phức tạp: O(n^2).

Subtask 3: Để giải subtask 3, ta tiến hành cải tiến subtask 2. Trong quá trình dfs, ta có 2 set giữ nhiệm vụ: lưu những trái ớt là tổ tiên của trái ớt hiện tại (set A) và những trái ớt không phải (set B). Khi đến thăm một trái ớt x nào đó thêm S(x) vào set A. Khi ra khỏi, ta xóa S(x) ra khỏi set A và thêm nó vào set B.

Xét trường hợp đầu tiên với trái ớt hiện tại là x. Ta sẽ tìm y trong set A sao cho S(x), $S(y) - S(x)$ và $n - S(y)$ gần nhau nhất có thể. Cho nên ta phải tìm y mà giá trị $|2*S(y) - (n+ S(x))|$ là tối thiểu. Vậy phải tìm S(y) gần $\frac{n+S(x)}{2}$ nhất. Ta có thể dùng set::lower_bound() để tìm giá trị gần $\frac{n+S(x)}{2}$ nhất. Trường hợp còn lại giống như trên nhưng dùng set B.

Độ phức tạp: O(nlogn).

CODE THAM KHẢO

Bài 1.

```
#include <bits/stdc++.h>
using namespace std;
#define MAX 10000001
int prime[MAX];
long c[MAX], f[MAX];
long long sum[MAX];
long t,n,m;
//FILE * fi = freopen("tommy.inp","r",stdin);
//FILE * fo = freopen("tommy.out", "w", stdout);
ifstream fi ("tommy.inp");
ofstream fo ("tommy.out");

//--------------------------
int eratosthene(){
	for(long long i=2;i*i<=MAX; ++i){
		if (!prime[i]){
			for(long long j=i;j<=MAX;j+=i){
				f[i] += c[j];
				prime[j]=true;
			}
		}
	}
	return 0;
}
//------------------------------
void process(){


	fi>>n;
	for(long i = 1; i<= n; ++i){

		fi>>t;
		++c[t];
	}

	eratosthene();

	for(long i = 2; i<= MAX ; ++i)
		sum[i] = sum[i-1] + f[i];

	fi>>m;
	long li,ri;
	for (long i = 1; i<= m; ++i){
```

```
            fi>>li>>ri;
            if (ri>1E7) ri = 1E7;
            fo<<sum[ri] - sum[li-1]<<'\n';
        }

}
//--------------------------------
int main(){
        ios_base::sync_with_stdio(false);
        process();
        return 0;
}
```

Bài 2.

```
#include <bits/stdc++.h>
using namespace std;
#define nmax 100001
int n,k,a[nmax],sum[nmax],lx,ix,ib,ie;
long long sum1;
vector<pair<long long, int> > f;
//-----------------------
void nhap()
{
   ios_base::sync_with_stdio(false);
   cin>>n>>k;
   for(int i=1;i<=n; i++)
   {
      cin>>a[i];
      sum[i] = sum[i-1]+a[i];
   }

}
//-------------------------
void sub1()
{
   for(int i=1;i<= n-1; ++i)
      for(int j=i+1;j<= n; ++j)
      {
         int s=0;
         for(int t=i; t<=j; ++t)
            s+=a[t];
         if(s == k*(j-i+1))
            if(j-i+1 > lx)
            {
               lx= j-i+1;
```

```
                ix = i;
            }
        }
    if(lx>0) cout<<ix<<" "<<lx;
    else cout<<0;
}
//----------------------------
void sub2()
{
    for(int i=1;i<= n-1; ++i)
        for(int j=i+1;j<= n; ++j)
        {
            int s = sum[j] - sum[i-1];
            if(s == k*(j-i+1))
                if(j-i+1 > lx)
                {
                    lx= j-i+1;
                    ix = i;
                }
        }
    if(lx>0) cout<<ix<<" "<<lx;
    else cout<<0;
}
//------------------------------
void init()//Xây dựng dãy f
{
    for(int i=1; i<=n ; i++)
        a[i] -= k;
    f.push_back({0,0});
    for(int i=1; i<=n; i++)
    {
        sum1+= a[i];
        f.push_back({sum1,i});
    }
    f.push_back({INT_MAX, n+1});
    sort(f.begin(),f.end());
}
//-----------------------------------
void sub3()
{
    init();
    int d = 1;
    for(int i=0; i<=n; i++)
    {
        if(d && f[i].first == f[i+1].first)
        {
```

```
            ib = f[i].second;
            d=0;
        }
        if(!d && f[i].first != f[i+1].first)
        {
            ie = f[i].second;
            int l = ie - ib;
            if(l>lx)
            {
                lx=l;
                ix = ib;
            }
            d=1;
        }
    }
    if(lx>0) cout<<ix+1<<" "<<lx;
    else cout<<0;
}
int main()
{
    freopen("CROAD.INP","r",stdin);
    freopen("CROAD.OUT","w",stdout);
    nhap();
    if(n<=500) sub1();
    else if(n<=5000) sub2();
         else sub3();
    return 0;
}
```

Bài 3.

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

using namespace std;

FILE *fi=freopen("CHILLIS.inp","r",stdin);
FILE *fo=freopen("CHILLIS.out","w",stdout);

const int N=200001;

int n,d[N],kq=INT_MAX;
vector<int> a[N];
set<int> q[2];

void nhap()
{
```

```
    cin>>n;
    for (int i=1;i<n;++i)
    {
        int u,v;
        cin>>u>>v;
        a[u].push_back(v);
        a[v].push_back(u);
    }
}

void dfs1(int u,int pa)
{
    d[u]=1;
    for (auto v:a[u])
    {
        if (v==pa)
            continue;
        dfs1(v,u);
        d[u]+=d[v];
    }
}

void dfs2(int u,int pa)
{
    q[0].insert(d[u]);

    int k=n,x=0;
    auto j=q[0].lower_bound((n+d[u])/2);
    if (j!=q[0].end())
    {
        k=min(k,abs((*j)-(n+d[u])/2));
        x=(*j);
    }
    if (j!=q[0].begin())
    {
        j--;
        if (k>abs((*j)-(n+d[u])/2))
            x=(*j);
    }
    kq=min(kq,max({d[u],x-d[u],n-x})-min({d[u],x-d[u],n-x}));

    k=n;x=0;
    j=q[1].lower_bound((n-d[u])/2);
    if (j!=q[1].end())
    {
        k=min(k,abs((*j)-(n-d[u])/2));
```

```
        x=(*j);
    }
    if (j!=q[1].begin())
    {
        j--;
        if (k>abs((*j)-(n-d[u])/2))
            x=(*j);
    }
    kq=min(kq,max({d[u],x,n-x-d[u]})-min({d[u],x,n-x-d[u]}));

    for (auto v:a[u])
    {
        if (v==pa)
            continue;
        dfs2(v,u);
    }

    q[0].erase(q[0].lower_bound(d[u]));
    q[1].insert(d[u]);
}

void process()
{
    dfs1(1,0);
    dfs2(1,0);
    cout<<kq;
}

int main()
{
    ios_base::sync_with_stdio(0);
    cin.tie(0);
    nhap();
    process();
    return 0;
}
```