Chuyên đề

QUY HOẠCH ĐỘNG LỒI

PHẦN MỞ ĐẦU

Quy hoạch động (QHĐ) là một lớp thuật toán rất quan trọng và có nhiều ứng dụng trong ngành khoa học máy tính. Trong các cuộc thi Olympic tin học hiện đại, QHĐ luôn là một trong những chủ đề chính. Việc áp dụng quy hoạch động vào giải các bài tập tin học là không hề đơn giản đối với học sinh. Nhất là hiện nay hầu hết các bài tập về quy hoạch động đều được nâng thêm một tầm cao mới. Điều này được thể hiện rõ trong đề thi VOI cũng như IOI. Tuy vậy, tài liệu nâng cao về QHĐ bằng tiếng Việt hiện còn rất khan hiếm, dẫn đến học sinhViệt Nam bị hạn chế khả năng tiếp cận với những kỹ thuật hiện đại. Trong một vài kỹ thuật để tối ưu hóa độ phức tạp của thuật toán QHĐ. Kỹ thuật bao lồi (convex hull trick) dùng để tối ưu hóa thuật toán quy hoạch động với thời gian $O(n^2)$ xuống còn O(nlogn), thậm chí với một số trường hợp xuống còn O(n), cho một lớp các bài toán.

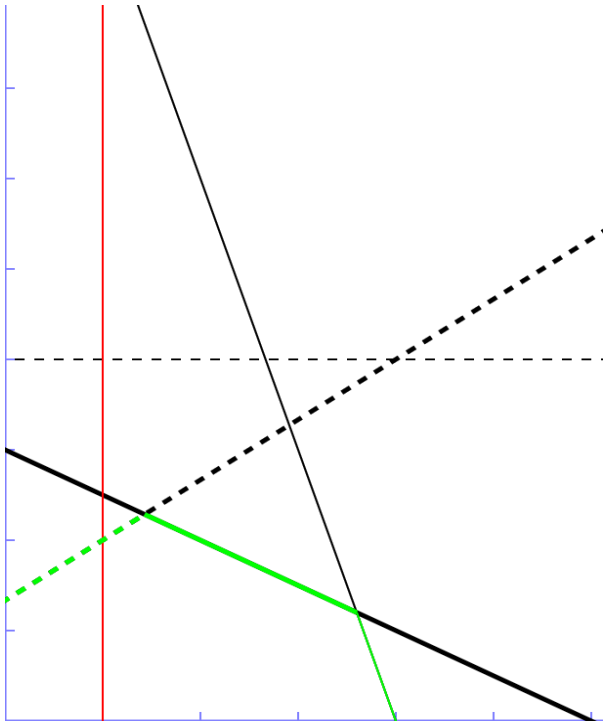
Code và Test của các bài tập trong chuyên đề Thầy cô có thể tham khảo theo đường dẫn sau: **https://app.box.com/s/n2m81i1l7ic9pcfggh26onokp5hqqiiv**

PHẦN NỘI DUNG CHUYÊN ĐỀ

I. Lý thuyết

Quy hoạch động bao lồi là một lớp của thuật toán quy hoạch động. Vấn đề bao gồm duy trì, tức là theo dõi bao lồi đối với dữ liệu đầu vào thay đổi động, tức là khi các yếu tố dữ liệu đầu vào có thể được chèn, xóa hoặc sửa đổi. Thuật toán này chỉ có thể áp dụng khi các điều kiện nhất định được đáp ứng. **Kĩ thuật bao lồi** là kĩ thuật (hoặc là cấu trúc dữ liệu) dùng để xác định hiệu quả, có tiền xử lý, cực trị của một tập các hàm tuyến tính tại một giá trị của biến độc lập. Mặc dù tên gọi giống nhưng kĩ thuật này lại khá khác biệt so với thuật toán bao lồi của hình học tính toán.

Bài toán mở đầu: Cho một tập N hàm bậc nhất $y=ai*x+bi$ 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. Ví dụ, cho các phương trình $y=4$, $y=4/3+2/3x$, $y=12-3x$ và $y=3-1/2$ x và truy vấn $x=1$. Chúng ta phải tìm phương trình mà trả về giá trị y cực tiểu với x = 1 (trong trường hợp này là phương trình $y=4/3+2/3x$ và giá trị cực tiểu đó là 2). Sau khi ta vẽ các đường thẳng lên hệ trục tọa độ, dễ thấy rằng: chúng ta muốn xác định, tại x=1(đường màu đỏ) đường nào có tọa độ y nhỏ nhất. Ở trong trường hợp này là đường nét đứt đậm $y=4/3+2/3x$.



Thuật toán đơn giản

Với mỗi truy vấn, ta duyệt qua tất cả các hàm số đê tìm giá trị nhỏ nhất. Nếu có M đường thẳng và Q truy vấn, độ phức tạp của thuật toán sẽ O(MQ). Kĩ thuật bao lồi sẽ giúp giảm độ phức tạp xuống còn O((Q+M)logM, một độ phức tạp hiệu quá hơn nhiều

Convex hull trick

Mỗi hàm số bậc nhất có thể biểu diễn bởi một đường thẳng trên hệ tọa độ Oxy

Xét hình vẽ ở trên. Đường thẳng $y=4$sẽ không bao giờ là giá trị nhỏ nhất với tất cả giá trị của x. Mỗi đường trong mỗi đường thẳng còn lại sẽ lại trả lại giá trị cực tiểu trong một và chỉ một đoạn liên tiếp (có thể có một biên là $+\infty$ hoặc $-\infty$). Đường chấm đậm sẽ cho giá trị cực tiểu với tất cả giá trị x nằm bên trái giao điểm của nó với đường đen đậm. Đường đen đậm sẽ cho giá trị cực tiểu với tất cả giá trị giữa giao điểm của nó với đường nhạt và đường chấm đậm. Và đường nhạt sẽ nhận cực tiểu cho tất cả giá trị x bên phải giao điểm với đường

đậm. Một nhận xét nữa là với giá trị của x càng tăng thì hệ số góc của các hàm số sẽ giảm, 2/3,−1/2,−3.

Điều này giúp chúng ta hiểu phần nào thuật toán:

- Bỏ đi các đường thẳng không quan trọng như y=4 trong ví dụ (những đường thẳng mà không nhận giá trị cực tiểu trong bất kì đoạn nào)
- Sắp xếp các đoạn thẳng còn lại theo hệ số góc và được một tập N đoạn thẳng (N là số đường thẳng còn lại)
- Nếu chúng ta xác định được điểm cuối của mỗi đoạn thì ta dùng thuật toán tìm kiếm nhị phân để có thể tìm kiếm đáp án cho từng truy vấn.

Ý nghĩa của tên

Cụm từ *bao lồi* được sử dụng để chỉ *hình bao trên/dưới* (upper / lower envelope). Trong ví dụ, nếu chúng ta coi mỗi phần đoạn thẳng tối ưu của đường thẳng (bỏ qua đường y=4), chúng ta sẽ thấy những đoạn đó tạo thành một *hình bao dưới* (lower envelope), một tập các đoạn thẳng chứa tất cả điểm cực tiểu cho mọi giá trị của x hình bao trên được tô bằng màu xanh trong hình. Cái tên *kĩ thuật bao lồi* xuất phát từ việc đường bao trên tạo thành một đường lồi, từ đó thành bao lồi của một tập điểm.

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 ta xét 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 l1, l2, l3 lần lượt là đường thẳng thứ hai, thứ nhất (ở đỉnh stack),và đường thẳng mới để thêm vào. Thì l2 trở nên không tiềm năng nếu và chỉ nếu điểm cắt của l1 và l3 nằm ở bên trái (có hoành độ nhỏ hơn) điểm cắt của l1 và l2. Điều này là xảy ra khi l3 có thể phủ hết khoảng mà trước đó l2là 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(M), 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(MlogM). 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(M). Tóm lại toàn bộ thuật toán bao gồm cả phần sắp xếp mất thời gian O(MlogM), nếu các đường thẳng đã được sắp xếp sẵn, thì thuật toán chạy trong thời gian tuyến tính.

II. BÀI TẬP ÁP DỤNG

Bài 1.HARBINGERS (CEOI 2009)

Ngày xửa ngày xưa, có N thị trấn kiểu trung cổ trong khu tự trị Moldavian. Các thị trấn này được đánh số từ 1 đến N. Thị trấn 1là 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 thị trấn 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 phải được báo cáo càng sớm càng tốt cho thủ đô. Mọi 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 thủ đô. Ban đầu, thông tin chiến sự được đưa cho người truyền tin tại thị trấn bị tấn công. Trong mỗi thị trấn mà anh ta đi qua, một người truyền tin có hai lựa chọn: hoặc đi đến thị trấn tiếp theo về phía thủ đô, hoặc để lại tin nhắn đến người truyền tin từ thị trấn này. Người truyền tin mới lại áp dụng tương tự cách trên. Nói chung một thông điệp có thể được thực hiện bởi một số người truyền tin trước khi đến thủ đô. Nhiệm vụ của bạn là đối với mỗi thị trấn tìm thời gian tối thiểu để gửi tin nhắn từ thị trấn đó đến thủ đô

Input: trong file harbingers.inp

- Dòng đầu ghi số N.
- N−1dòng 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 Si và Vi mô tả đặc điểm của người truyền tin trong thị trấn thứ i+1 Si thể hiện thời gian cần để khởi động và Vi là số lượng phút để đi được 1km của người truyền tin ở thị trấn i+1.

harbingers.inp	harbingers.out
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

Output: trong file harbingers.out

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

Ví dụ

Giới hạn

- $3 \le N \le 100\,000$
- $0 \le Si \le 10^9$
- $1 \le Vi \le 10^9$
- Độ dài mỗi con đường không vượt quá 10 000

Ràng buộc

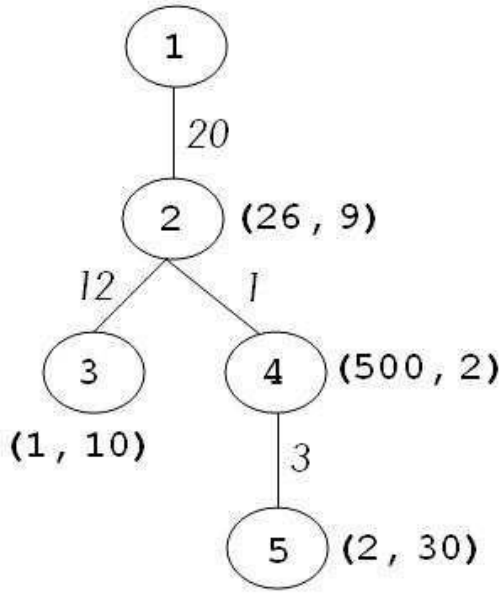
- 20% số test, $N \le 2\,500$
- 50% số test, mỗi thị trấn sẽ có nhiều nhất 2 con đường liền kề

Giải thích

Các con đường và độ dài của chúng được hiển thị trong hình bên trái. Thời gian khởi động và tốc độ của những người đưa tin được viết giữa dấu ngoặc.

Thời gian tối thiểu để gửi tin nhắn từ thị trấn 5 đến thủ đô là đạt được như sau. Người báo hiệu từ thị trấn 5 nhận thông điệp

và rời khỏi thị trấn sau 2 phút. Anh đi bộ 4 km trong 120 vài phút trước khi đến thị trấn 2. Anh ta để lại tin nhắn từ thị trấn đó. Tin nhắn thứ hai cần 26 phút để bắt đầu cuộc hành trình và đi bộ 180 phút đến thủ đô. Tổng thời gian là: 2 + 120 + 26 + 180 = 328.



Phân tích

Thuật toán QHĐ

Subtask1 . Cách tiếp cận đầu tiên là sử dụng lập trình động. Nếu chúng tôi coi opti là thời gian tối thiểu cần thiết để gửi tin nhắn từ thị trấn i đến thủ đô, thì chúng tôi có công thức đệ quy sau:

$Opt_i = \min (opt_j + dist_{j->i} \cdot V_i + S_i)$ (1)

Trong đó j là một nút trên đường dẫn từ thị trấn i đến thị trấn 1. Mối quan hệ này có được bằng cách xem xét mọi người đưa tin j có thể nhận được thông báo từ người đưa tin i. $dist_{j->i}$ biểu thị khoảng cách giữa các nút j và i. Khoảng cách này có thể được tính trong thời gian không đổi nếu ban đầu chúng ta tính một vectơ D, trong đó Di là khoảng cách từ thị trấn i đến thủ đô. Việc thực hiện trực tiếp (1) mất thời gian $O(N^2)$.

Subtask 2 Công thức (1) có thể được viết lại thành:

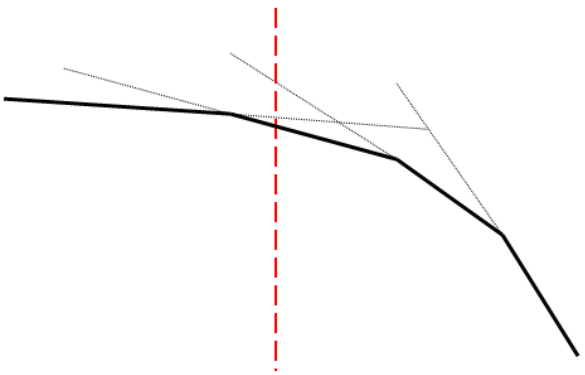
$Opt_i = \min (opt_j - D_j \bullet V_i + D_i \bullet V_i + S_i)$ (2)

Khi tính toán giá trị cho opt_i, $D_i \bullet V_i + S_i$ là hằng số cho tất cả j, vì vậy:

$Opt_i = \min (opt_j - D_j \bullet V_i) + D_i \bullet V_i + S_i$ (3)

Chúng ta cần tìm tối thiểu cho biểu thức $opt_j - D_j \bullet V_i$. Điều này có một giải thích hình học hữu ích: đối với mỗi nút i, chúng ta coi như một đường thẳng trong mặt phẳng được cho bởi phương trình $Y = opt_i - D_i \bullet X$

Tìm giá trị tối thiểu bây giờ tương đương với: tìm đường thấp nhất trong mặt phẳng cắt nhau bởi đường $X = V_i$. Để giải quyết truy vấn này một cách hiệu quả, chúng ta duy trì đường bao thấp hơn của các đường thẳng. Bởi vì độ dài đường là dương, chúng ta có $D_j < D_i$ cho mọi i và j, trong đó j là tổ tiên của i. Điều này có nghĩa là hệ số góc của các đường trong đường bao đang giảm dần. Vì vậy, chúng ta có thể lưu trữ các đường thẳng trong một ngăn xếp.



Ở mỗi bước i, chúng ta phải:

- Truy vấn đường thẳng nào trong đường bao tạo ra giá trị tối thiểu với $X = V_i$
- Chèn $Y = opt_i - D_i \bullet X$ vào đường bao (có thể xóa đường khác).

Bước đầu tiên có thể được giải quyết một cách hiệu quả bằng cách sử dụng tìm kiếm nhị phân, Bước thứ hai có thể được giải trong O (Δ), nếu Δ đường thẳng bị xóa khỏi đường bao. Chúng ta chỉ cần lấy ra đỉnh của ngăn xếp miễn là đường thẳng vừa được chèn hoàn toàn làm che đoạn ở đỉnh ngăn xếp. Vì mỗi đường thẳng trong ngăn xếp được lấy ra khỏi ngăn xếp nhiều nhất một lần, nên độ phức tạp chung của bước này là O (N). Do đó, toàn bộ thuật toán chạy trong thời gian O (N lg N).

Full test. chúng ta thực hiện Quy hoạch động với tìm kiếm theo chiều sâu. Để duy trì đường bao thấp hơn trong DFS, chúng ta phải hỗ trợ hai hoạt động cấu trúc dữ liệu:

1. Chèn một đường thẳng hiệu quả (khi chuyển xuống nút con).
2. Xóa một đường thẳng và khôi phục đường bao về trạng thái trước đó (khi chuyển lên nút cha).

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 2"Acquire" ***Nguồn: usaco 2008 march***

Nông dân John đang xem xét mua thêm đất cho trang trại và trước mắt anh có N (1 <= N <= 50.000) lô hình chữ nhật, mỗi ô với kích thước nguyên (1 <= width_i <= 1.000.000; 1 <= length_i <= 1.000.000).

Anh ấy có thể mua bất kỳ số lô đất nào với giá bằng tích của chiều dài dài nhất và chiều rộng dài nhất. Tất nhiên,thửa đất không thể được xoay (đổi chiều dài và chiều rộng)., tức là, nếu Nông dân John mua một mảnh đất 3x5 và một mảnh 5x3 trong một nhóm, anh ta sẽ trả 5x5 = 25.

John muốn phát triển trang trại của mình càng nhiều càng tốt và mong muốn mua tất cả Lô đất. Anh chợt nhận ra rằng anh ta có thể mua đất theo nhóm liên tiếp, khéo léo giảm thiểu tổng chi phí bằng cách nhóm các lô khác nhau có lợi thế giá trị chiều rộng hoặc chiều dài.

Cho số lượng lô để bán và kích thước của mỗi lô, xác định số tiền tối thiểu mà Nông dân John có thể mua tất cả

Đầu vào: **trong file Acquire.inp**

* Dòng 1: Một số nguyên duy nhất N
* Dòng 2..N + 1: Dòng i + 1 mô tả lô i với hai số nguyên: width_i và length_i

Đầu ra: **trong file Acquire.out**

Số tiền tối thiểu cần thiết để mua tất cả các lô.

Ví dụ

Acquire.inp	**Acquire.out**
4 100 1 15 15 20 5 1 100	500

Giải thích:

Nhóm đầu tiên chứa một lô 100x1 và có giá 100. Nhóm tiếp theo chứa một lô 1x100 và có giá 100. Nhóm cuối cùng chứa cả 20x5 lô và lô 15x15 và chi phí 300. Tổng chi phí là 500, đó là tối thiểu.

Phân tích

Nhận xét 1: 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.

Nhận xét 2: 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.

Lời giải **Quy Hoạch Động**

Vậy bài toán trờ 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 $O(N^2)$ (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.

Nhận xét 3: **Sử dụng bao lồi**

Với mj=rect[j+1].w,bj=cost[j],x=rect[i].h với rect[x].hlà chiều rộng của hình chữ nhật x và rect[x].w là chiều dài của hình chữ nhật x. Vậy thì bài toán trờ về tìm hàm cực tiểu của y=mjx+bj bằng cách tìm j tối ưu. 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. Vậy là ta đã giải quyết thành công bài toán

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 3 Commando Nguồn: APIO 2010

Bạn là chỉ huy của một đội quân gồm n binh sĩ, được đánh số từ 1 đến n. Đối với trận chiến phía trước, bạn có kế hoạch chia n người lính này thành các đơn vị đặc công. Để thúc đẩy sự đoàn kết và tăng cường tinh thần, mỗi đơn vị sẽ bao gồm một chuỗi các binh sĩ liên tục (i, i + 1, .., i + k). Mỗi người lính i có một đánh giá hiệu quả chiến đấu x_i. Ban đầu, trận chiến hiệu quả x của một đơn vị đặc công (i, i + 1, .., i + k) đã được tính bằng cách cộng hiệu quả chiến đấu cá nhân của các chiến sĩ trong đơn vị. Tức là $x = x_i + x_{i+1} + \cdots + x_{i+k}$.

Tuy nhiên, nhiều năm chiến thắng vẻ vang đã khiến bạn kết luận rằng hiệu quả chiến đấu của một đơn vị nên được điều chỉnh như sau: điều chỉnh hiệu quả x' được tính bằng cách sử dụng phương trình $x' = ax^2 + bx + c$, trong đó a, b, c là các hệ số đã biết ($a < 0$), x là hiệu quả ban đầu của đơn vị.

Nhiệm vụ của chỉ huy là chia lính thành các đơn vị đặc công để tối đa hóa tổng hiệu quả điều chỉnh của tất cả các đơn vị.

Chẳng hạn, giả sử bạn có 4 lính, x1 = 2, x2 = 2, x3 = 3, x4 = 4.

Hơn nữa, hệ số cho phương trình để điều chỉnh hiệu quả chiến đấu của một đơn vị là a = -1, b = 10, c = -20. Trong trường hợp này, giải pháp tốt nhất là chia binh lính thành ba đơn vị đặc công: Đơn vị đầu tiên chứa binh lính 1 và 2, đơn vị thứ hai chứa lính 3 và đơn vị thứ ba chứa lính 4. Hiệu quả chiến đấu của ba đơn vị lần lượt là 4, 3, 4 và hiệu quả điều chỉnh lần lượt là 4, 1, 4. Tổng hiệu quả điều chỉnh cho nhóm này là 9 và có thể kiểm tra rằng không có giải pháp nào tốt hơn.

Đầu vào: trong file Commando.inp

- Dòng đầu tiên chứa một số nguyên dương n, tổng số binh sĩ.
- Dòng thứ hai chứa 3 số nguyên a, b, và c, các hệ số cho phương trình để điều chỉnh hiệu quả chiến đấu của một đơn vị đặc công.
- Dòng cuối cùng chứa n số nguyên x1, x2 ,. . . , xn, tách bằng khoảng trắng, thể hiện hiệu quả chiến đấu của binh lính 1, 2,. . . , n, tương ứng.

Đầu ra: trong file Commando.out

- Một dòng có một số nguyên cho thấy hiệu quả được điều chỉnh tối đa có thể đạt được.

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

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

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≤ k<n

Độ phức tạp: $O(n^2)$

Subtask 3

Biến đổi hàm "adjust" . Định nghĩa sum(1,x) là δ(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;
}
```

Bài 4 Building Nguồn: CEOI 2017

Giới hạn thời gian: 3 giây Giới hạn bộ nhớ: 128 MB

Một dòng sông rộng có n trụ cột có thể có độ cao khác nhau nổi bật trên mặt nước. Chúng được sắp xếp theo một đường thẳng từ bờ này sang bờ khác. Chúng ta muốn xây dựng một cây cầu sử dụng các trụ làm hỗ trợ. Để đạt được điều này, chúng ta sẽ chọn một tập hợp con của các trụ cột và kết nối đỉnh của chúng thành các phần của cây cầu. Tập hợp con phải bao gồm trụ đầu tiên và trụ cột cuối cùng.

Chi phí xây dựng một phần cầu nối giữa các cột i và j là $(h_i - h_j)^2$ như chúng ta muốn để tránh các phần không bằng phẳng, trong đó h_i là chiều cao của cột i. Ngoài ra, chúng tôi cũng sẽ phải loại bỏ tất cả các trụ không phải là một phần của cây cầu, vì chúng cản trở giao thông đường sông. Chi phí loại bỏ trụ thứ i bằng w_i. Chi phí này thậm chí có thể âm, một số bên quan tâm sẵn sàng trả tiền cho bạn để thoát khỏi một số trụ cột nhất định. Tất cả các độ cao h_i và chi phí w_i là số nguyên.

Chi phí tối thiểu có thể để xây dựng cây cầu kết nối trụ đầu tiên và trụ cột cuối cùng là gì?

Đầu vào: trong file Building.inp

- Dòng đầu tiên chứa số trụ, n.
- Dòng thứ hai chứa chiều cao trụ cột h_i theo thứ tự, cách nhau bởi một khoảng trắng.
- Dòng thứ ba chứa w_i theo cùng thứ tự, chi phí tháo dỡ trụ cột.

Đầu ra: trong file Building.out

Đầu ra chi phí tối thiểu để xây dựng cây cầu. Lưu ý rằng nó có thể âm.

Giới hạn

- $2 \le n \le 10^5$
- $0 \le h_i \le 10^6$
- $0 \le |w_i| \le 10^6$

Ràng buộc

- Subtask 1: 30% điểm $n \le 1000$
- Subtask 2: 60% điểm giải pháp tối ưu bao gồm tối đa 2 trụ cột bổ sung (ngoài trụ đầu tiên và cuối cùng) $|w_i| \le 20$
- Subtask 3: 100% điểm không có ràng buộc bổ sung

Ví dụ

Building.inp	**Building.out**
6 3 8 7 1 6 6 0 -1 9 1 2 0	17

Phân tích

Subtask 1

Trước tiên, hãy để thay đổi một chút vấn đề để chỉ tập trung vào các trụ cột hỗ trợ cây cầu chứ không phải những cái khác cần được bỏ Chi phí tổng thể bao gồm chi phí do sự khác biệt trong chiều cao cộng với chi phí phá hủy các trụ cột không sử dụng. Cái sau bằng tổng của trụ bỏ đi chi phí trừ đi chi phí của những trụ giữ lại.

Quy hoạc động. Hãy xem xét một bài toán con về việc xây dựng những cây cầu sao cho cái cuối cùng kết thúc trên trụ i. Đặt f(i) đại diện cho chi phí tối thiểu để giải quyết bài toán con này. Cây cầu cuối cùng trong giải pháp sẽ trải dài từ một số trụ j <i đến i, dẫn đến công thức:

f (1) = $-w_1$

f (i) = min $(f(j)+(h_j - h_i)^2 - w_i)$ với j<i

Độ phức tạp: $O(n^2)$ chạy được

Subtasks 2. Chúng ta có thể làm tốt hơn nếu chúng ta biết rằng giải pháp tối ưu chỉ bao gồm bổ sung 2 trụ cột ngoài cái đầu tiên và cái cuối cùng? Nếu chỉ có một cột bổ sung, chúng ta có thể thử tất cả.

Đối với trường hợp có hai trụ cột bổ sung, chúng ta sẽ thử tất cả các cột thứ hai i và với mỗi cột chọn một trụ tối ưu đầu tiên j.

Đóng góp của trụ cột đầu tiên cho tổng chi phí là $(h_1 - h_j)^2 + (h_j - h_i)^2 - w_j$. Hãy bỏ qua w_j bây giờ (giả sử $w_j = 0$). Theo trực giác, chúng ta muốn chọn h_j gần với mức trung bình của h_1 và h_i. Do đó, chúng ta quan tâm đến chỉ có hai trụ cột - lớn nhất trong số những cột nhỏ hơn hoặc bằng $(h_1 + h_i) / 2$ và nhỏ nhất trong số những cái lớn hơn Khi chúng ta thử các cột i khác nhau từ 1 đến n, chúng ta có thể duy trì cấu trúc cây của các cột j <i đặt theo chiều cao của chúng. Điều này cho phép chúng ta tìm hai trụ trong O(log n).

Tất cả những gì còn lại là để xử lý w_j. Phạm vi giá trị giới hạn của w_j là chấp nhận được trong trường hợp này. Chúng ta có thể duy trì cấu trúc cây riêng biệt cho mọi giá trị của w_j và tìm kiếm trụ cột tối ưu j cho mọi giá trị có thể của w_j. Độ phức tạp thời gian là O(nalog n); a = max$|w_i|$.

Full test.

Viết lại định nghĩa đệ quy

- f(i) = min((A(j).hi + B(j) + C(i)) với j<i
- A(j) = $-2h_j$ B(j) = hj^2 +f(j) C(i) = hi^2 $-w_i$

Lưu ý rằng C là một số hạng không đổi chỉ phụ thuộc vào i và giống nhau bất kể sự lựa chọn của j. Chúng ta có thể xem xét mọi cặp A(j) và Bj) là hệ số góc của một đường thẳng. Vấn đề tìm j tốt nhất làm giảm việc tìm đường thấp nhất tại x = h_i. Để làm điều này trong O(log n), chúng ta sẽ duy trì một đường bao thấp hơn của một tập hợp các đường thẳng. Cấu trúc dữ liệu phải là động - nó nên hỗ trợ chèn một đường thẳng mới trong O (log n). Lưu ý rằng các đường thẳng bao gồm đường bao thấp hơn được sắp xếp theo hệ số góc của chúng. Vì vậy, chúng ta có thể duy trì một cấu trúc cây của các đoạn đường thẳng. Nó cũng cho phép chúng ta nhanh chóng tìm ra đường thẳng tối ưu với một x nhất định cũng như chèn một đường thẳng mới. Chúng ta có thể dùng SET trong C ++ C. Tối ưu hóa này

thường được gọi convex hull trick liên quan đến lập trình động và giải quyết vấn đề trong thời gian O (n logn).

Code tham khảo

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <assert.h>
#include <iostream>
#include <sstream>
#include <vector>
#include <string>
#include <math.h>
#include <queue>
#include <list>
#include <algorithm>
#include <map>
#include <set>
#include <stack>
#include <ctime>
#include <iterator>
using namespace std;
#define ALL(c) (c).begin(),(c).end()
#define IN(x,c) (find(c.begin(),c.end(),x) != (c).end())
#define REP(i,n) for (int i=0;i<(int)(n);i++)
#define FOR(i,a,b) for (int i=(a);i<=(b);i++)
#define INIT(a,v) memset(a,v,sizeof(a))
#define SORT_UNIQUE(c) (sort(c.begin(),c.end()),
c.resize(distance(c.begin(),unique(c.begin(),c.end()))))
template<class A, class B> A cvt(B x) { stringstream ss; ss<<x; A y; ss>>y; return y; }
typedef pair<int,int> PII;
typedef long long int64;
#define N 100000
int n;
int64 h[N],w[N];
int64 sqr(int64 x) { return x*x; }
struct line {
        char type;
        double x;
        int64 k, n;
```

```
};

bool operator<(line l1, line l2) {
	if (l1.type+l2.type>0) return l1.x<l2.x;
	else return l1.k>l2.k;
}
set<line> env;
typedef set<line>::iterator sit;
bool hasPrev(sit it) { return it!=env.begin(); }
bool hasNext(sit it) { return it!=env.end() && next(it)!=env.end(); }
double intersect(sit it1, sit it2) {
	return (double)(it1->n-it2->n)/(it2->k-it1->k);
}

void calcX(sit it) {
	if (hasPrev(it)) {
		line l = *it;
		l.x = intersect(prev(it), it);
		env.insert(env.erase(it), l);
	}
}

bool irrelevant(sit it) {
	if (hasNext(it) && next(it)->n <= it->n) return true; // x=0 cutoff
	return hasPrev(it) && hasNext(it) && intersect(prev(it),next(it)) <=
intersect(prev(it),it);
}

void add(int64 k, int64 a) {
	sit it;
	// handle collinear line
	it=env.lower_bound({0,0,k,a});
	if (it!=env.end() && it->k==k) {
		if (it->n <= a) return;
		else env.erase(it);
	}
	// erase irrelevant lines
	it=env.insert({0,0,k,a}).first;
	if (irrelevant(it)) { env.erase(it); return; }
```

```
        while (hasPrev(it) && irrelevant(prev(it))) env.erase(prev(it));
        while (hasNext(it) && irrelevant(next(it))) env.erase(next(it));
        // recalc left intersection points
        if (hasNext(it)) calcX(next(it));
        calcX(it);
}

int64 query(int64 x) {
        auto it = env.upper_bound((line){1,(double)x,0,0});
        it--;
        return it->n+x*it->k;
}

int64 g[N];

int64 solve() {
        int64 a=0;
        REP (i,n) a+=w[i];
        g[0]=-w[0];
        FOR (i,1,n-1) {
                add(-2*h[i-1],g[i-1]+sqr(h[i-1]));
                int64 opt=query(h[i]);
                g[i]=sqr(h[i])-w[i]+opt;
        }
        return a+g[n-1];
}

int main() {
        cin >> n;
        REP (i,n) cin >> h[i];
        REP (i,n) cin >> w[i];
        cout << solve() << endl;
        return 0;
}
```

Bài 5 GAUSS Nguồn COCI 2017

Giáo viên đưa cho Carl một loạt các số nguyên dương F(1), F(2), …, F(K). Chúng ta coi F(t) = 0 với t> K. Cô giáo đưa thêm cho Carl một bộ số may mắn và giá của mỗi con số may mắn. Nếu X là số may mắn thì C(X) biểu thị giá của nó.

Ban đầu, có một số nguyên dương A được viết trên bảng. Trong mỗi lần di chuyển, Carl phải thực hiện một những điều sau đây:

- Nếu số N hiện được viết trên bảng, thì Carl có thể viết một trong các ước của nó là M nhỏ hơn N, thay vì N. Nếu anh ta viết số M, giá của di chuyển là F(d (N / M)), trong đó d (N /M) là số ước của số nguyên dương N / M (không bao gồm N / M).
- Nếu N là số may mắn, Carl có thể để số đó trên bảng và giá của di chuyển là C(N).
- Carl phải thực hiện chính xác L di chuyển, và sau khi anh ta đã thực hiện tất cả các động tác của mình, số B phải được viết trên bảng. Gọi G(A, B, L) là mức giá tối thiểu mà Carl có thể đạt được. Nếu không thể thực hiện L di chuyển như vậy thì G(A, B, L) = -1.

Giáo viên đã đưa ra Q truy vấn cho Carl. Trong mỗi truy vấn, Carl nhận được số A và B và phải tính giá trị $G(A, B, L_1) + G(A, B, L_2) + \ldots + G(A, B, L_M)$, trong đó các số $L_1, \ldots, L_M$ là giống nhau cho tất cả các truy vấn.

ĐẦU VÀO: trong file GAUSS.INP

- Dòng đầu tiên chứa số nguyên dương K ($1 \le K \le 10\,000$).
- Dòng thứ hai chứa K số nguyên dương F(1), F(2), …, F(K) nhỏ hơn hoặc bằng 1000.
- Dòng sau chứa số nguyên dương M ($1 \le M \le 1\,000$).
- Dòng sau chứa M số nguyên dương $L_1, \ldots, L_M$ nhỏ hơn hoặc bằng 10 000.
- Dòng sau chứa số nguyên dương T, tổng số số may mắn ($1 \le T \le 50$).
- Mỗi dòng trong T dòng sau đây chứa các số X và C(X) biểu thị rằng X là một số may mắn và C(X) là giá của số ấy ($1 \le X \le 1\,000\,000$, $1 \le C(X) \le 1\,000$).
- Mỗi con số may mắn xuất hiện nhiều nhất một lần.
- Dòng sau chứa số nguyên dương Q ($1 \le Q \le 50\,000$).
- Mỗi dòng trong Q dòng sau đây chứa 2 số nguyên dương A và B ($1 \le A, B \le 1\,000\,000$).

ĐẦU RA: trong file GAUSS.OUT

- Bạn phải xuất Q dòng. Dòng thứ i chứa câu trả lời cho truy vấn thứ i được xác định trong đầu vào

VÍ DỤ

GAUSS.INP	GAUSS.OUT
4 1 1 1 1 2 1 2 2 2 5 4 10 1 4 2	7

Giải thích ví dụ

$L_1 = 1$, vì vậy Carl có thể thực hiện chính xác một lần di chuyển - thay thế số 4 bằng số 2, do đó G(4, 2, 1) = F(d (2)) = 1.

$L_2 = 2$ nên Carl có hai lựa chọn:

● *Anh ta có thể thay thế số 4 bằng số 2 và sau đó để lại số 2 (vì đó là một số may mắn), vì vậy anh ta trả giá F(d (4/2)) + C(2) = 1 + 5 = 6*

● *Anh ta có thể để lại số 4 trong lần di chuyển đầu*

tiên và thay thế số đó trong lần di chuyển thứ hai bằng số 2, vì vậy giá là C(4) + F(d (4/2)) = 10 + 1 = 11

Tùy chọn đầu tiên có chi phí ít hơn, vì vậy G(4, 2, 2) = 6.

Câu trả lời cho truy vấn là G(4.2,1) + G(4.2,2) = 7.

Phân tích

Gọi số ban đầu là A, số cuối cùng là B và C là số chúng ta để lại trên bảng.

Để ý rằng số lượng hoạt động giữa các số A và C và giữa C và B nhiều nhất là 20, vì trong mỗi thời điểm, số mới nhỏ hơn hoặc bằng một nửa số cũ. Không khó để nhận thấy rằng chi phí của con đường ngắn nhất từ A đến C bằng với chi phí của con đường ngắn nhất từ A / C đến 1 (tương tự cho C và C /B).

Dựa vào đặc thù của công thức tính các chi phí, chúng ta thấy rằng chi phí ngắn nhất đường đi đến 1 là bằng nhau, ví dụ, số 12 và 18. Chính xác hơn, gọi e_1, e_2,.., e_k là số mũ của các số nguyên tố trong phân tích của một số thành số nguyên tố. Chi phí của con đường ngắn nhất từ số đó đến 1 bằng với tất cả các số có cùng số mũ { e_1, e_2,.., e_k }.

Chúng ta có thể tạo ra tất cả các số như vậy và thấy rằng có ít hơn 250 số . Chúng ta gán cho mỗi số n số nhỏ nhất mà nó dùng chung bộ số mũ và biểu thị nó với k(n).

Vậy:

- Con đường ngắn nhất từ A đến B là đi qua C. Chi phí từ A đến C bằng với chi phí từ A / C đến 1, và bằng với giá của k(A / C) đến 1. Tương tự cho C đến B và k(B / C).
- Gọi b[x] [y] [C] [L] là con đường ngắn nhất từ A đến B sao cho k(A / C) = x và k(C / B) = y và tổng chiều dài của các đường từ A đến C và C đến B bằng L
- Mảng này có thể được tính đủ nhanh trong chương trình tối ưu hóa - chúng ta sẽ chỉ tính toán cho các cặp x và y sao cho x * y nhiều nhất là 1 000 000.

Làm thế nào để chúng ta trả lời các truy vấn khi chúng ta có mảng này?

- Xét tất cả các C có thể và xem công thức cho tổng số lần di chuyển L vàgọi L’ là số lượng di chuyển khi chúng ta để lại C trên bảng: Tổng chi phí là b[x] [y] [C] [L – L’]] + L’ * cost [C].
- Bây giờ chúng ta thấy rằng, đối với một L cố định, chúng ta cần tìm một cặp (C, L’) để giảm thiểu giá trị của biểu thức trên, đây thực sự là một vấn đề về kĩ thuật bao lồi và với mỗi L tìm giao điểm của đường thẳng đứng x = L với đường bao
- Chúng ta cũng lưu ý rằng tất cả các đường thẳng với một C cố định có cùng hệ số góc, vì vậy nó đủ để tính toán giao điểm lớn hơn của các đường thẳng của nó với trục y và chỉ thêm đường đó vào bao lồi.
- Các truy vấn trong đó L nhỏ hơn 20 cần được xử lý riêng bằng cách sử dụng quy hoạch động.

Chương trình minh họa

```
#include <cstdio>
#include <cstring>
#include <cassert>
```

```
#include <iostream>
#include <algorithm>
#include <vector>
#include <set>
#define num first
#define val second
#define FOR(i, a, b) for (int i = (a); i < (b); ++i)
#define REP(i, n) FOR (i, 0, n)
#define _ << " _ " <<
#define TRACE(x) cerr << #x << " = " << x << endl
#define debug(...) fprintf(stderr, __VA_ARGS__)
//#define debug
//#define TRACE(x)
using namespace std;
using llint = long long;
using vec = vector<int>;
using pii = pair<int, int>;
const int MAX = 1000000;
const int MAXK = 10010;
const int MAXC = 305;
const int MAXD = 6500;
const int MAXM = 55;
const int MAXLEN = 23;
const int INF = 1e9;
struct edge {
  int div, v, w;
};
struct line {
  llint k, l;
};
vector<pii> happy;
vec primes, s;
int dist[MAXC][MAXLEN], divs[MAXC], f[MAXK], K, hap[MAX + 10], price[MAX +
10];
vector<edge> edg[MAXC];
inline int get(int i) { return (i > K) ? 0 : f[i]; }
inline void add_edge(int a, int div, int b, int w) {
  edg[a].push_back({div, b, w});
  if (b == 0 && a != 0)
```

```
    dist[a][1] = min(dist[a][1], w);
}
inline int index(int x) {
  return (int)(lower_bound(s.begin(), s.end(), x) - s.begin());
}
int compress(int n) {
  vec e;
  for (int p : primes) {
    if (p * p > n) break;
    if (n % p > 0) continue;
    int ec = 0;
    while (n % p == 0) {
      n /= p;
      ++ec;
    }
    e.push_back(ec);
  }
  if (n > 1)
    e.push_back(1);

  sort(e.begin(), e.end());
  reverse(e.begin(), e.end());

  int ret = 1;
  for (int i = 0; i < (int)e.size(); ++i)
    for (int j = 0; j < e[i]; ++j)
      ret *= primes[i];
  return ret;
}

int divisors(int n) {
  int ret = 0;
  for (int m = 1; m * m <= n; ++m)
    if (n % m == 0) {
      ++ret;
      if (m * m != n)
          ++ret;
    }
  return ret;
```

```
}

bool isprime(int p) {
  for (int q = 2; q * q <= p; ++q)
    if (p % q == 0)
      return false;
  return true;
}
void gen(llint m, int i, int prv) {
  s.push_back((int)m);
  for (int x = 1; x <= prv; ++x) {
    m *= primes[i];
    if (m > MAX) break;
    gen(m, i + 1, x);
  }
}
//llint best[MAXC][MAXC][MAXM][MAXM];
llint bl[MAXD][MAXM][MAXLEN];
llint bl2[MAXD][MAXM][MAXLEN];
int idx[MAXC][MAXC];
vector<int> lens;

int sz;
line st[MAXM * MAXLEN];

double getx(line l1, line l2) {
  return (l2.l - l1.l) / (double)(l1.k - l2.k);
}
void insert(line l) {
  while (sz >= 1 && st[sz - 1].k == l.k) {
    if (st[sz - 1].l > l.l)
      --sz;
    else
      return;
  }

  while (sz >= 2 && getx(st[sz - 1], l) < getx(st[sz - 1], st[sz - 2]))
    --sz;
  st[sz++] = l;
```

```
}

void preprocess_fast() {
  int tot = 0;
  REP(x, (int)s.size())
    REP(y, (int)s.size()) {
      if ((llint)s[x] * s[y] > MAX) continue;
      idx[x][y] = tot++;
  }
  REP(x, (int)s.size()) {
    REP(y, (int)s.size()) {
      if ((llint)s[x] * s[y] > MAX) continue;
      for (int i = 0; i < (int)happy.size(); ++i) {
            REP(len, MAXLEN) bl[idx[x][y]][i][len] = INF;
            REP(len1, MAXLEN) {
              REP(len2, MAXLEN - len1) {
                if (dist[x][len1] != INF && dist[y][len2] != INF) {
                    llint tmp = 0;
                    tmp += dist[x][len1];
                    tmp += dist[y][len2];
                    tmp += (llint)(-len1 - len2) * happy[i].val;
                    bl[idx[x][y]][i][len1 + len2] =
                      min(bl[idx[x][y]][i][len1 + len2], tmp);
                }
              }
              bl2[idx[x][y]][i][0] = bl[idx[x][y]][i][0];
              FOR(len, 1, MAXLEN)
                bl2[idx[x][y]][i][len] =
                min(bl2[idx[x][y]][i][len - 1], bl[idx[x][y]][i][len]);
            }
      }
    }
  }
}

llint solve_fast2(int a, int b) {
  llint sol = 0;
  sz = 0;
  int ptr = 0;
```

```
    for (int j = 0; j < (int)happy.size(); ++j) {
    int c = happy[j].num;
    if (a % c != 0 || c % b != 0) continue;
    int x = index(compress(a / c));
    int y = index(compress(c / b));

    insert({happy[j].val, bl2[idx[x][y]][j][MAXLEN - 1]});
  }

  for (int i = 0; i < (int)lens.size(); ++i) {
    llint ret = INF;
    if (lens[i] < MAXLEN) {
      ret = dist[index(compress(a / b))][lens[i]];
      for (int j = 0; j < (int)happy.size(); ++j) {
          int c = happy[j].num;
          if (a % c != 0 || c % b != 0) continue;
          int x = index(compress(a / c));
          int y = index(compress(c / b));
          ret = min(ret, bl2[idx[x][y]][j][lens[i]] + (llint)lens[i] * happy[j].val);
      }
    } else {
      for (; ptr + 1 < sz && getx(st[ptr], st[ptr + 1]) < lens[i]; ++ptr);
      if (sz > 0)
          ret = min(ret, st[ptr].k * (llint)lens[i] + st[ptr].l);
    }
    if (ret == INF) ret = -1;
    sol += (a % b != 0) ? -1 : ret;
  }
  return sol;
}

bool cmp2(pii a, pii b) { return a.second > b.second; }

void init() {
  int M, L;
  scanf("%d",&K);
  FOR(i, 1, K + 1) scanf("%d",&f[i]);
  scanf("%d",&L);
  lens.resize(L);
```

```
  REP(i, L) scanf("%d",&lens[i]);
  sort(lens.begin(), lens.end());

  scanf("%d",&M);
  REP(i, M) {
    int x, cost;
    scanf("%d %d",&x,&cost);
    happy.push_back({x, cost});
    hap[x] = 1;
    price[x] = cost;
  }
  sort(happy.begin(), happy.end(), cmp2);
}

int main(void) {
  REP(a, MAXC) REP(len, MAXLEN) dist[a][len] = INF;
  dist[0][0] = 0;
  init();

  for (int p = 2; p * p <= MAX; ++p)
    if (isprime(p))
      primes.push_back(p);

  gen(1, 0, MAX);
  sort(s.begin(), s.end());

  REP(i, (int)s.size())
    divs[i] = divisors(s[i]);

  // TRACE((int)s.size());
  // TRACE((int)primes.size());

  for (int i = 0; i < (int)s.size(); ++i) {
    int a = s[i];
    for (int x = 1; x * x <= a; ++x)
      if (a % x == 0) {
          int j = index(compress(x));
          int k = index(compress(a / x));
          add_edge(i, x, j, get(divs[k]));
```

```
        if (x * x != a)
          add_edge(i, a / x, k, get(divs[j]));
      }
    }

    for (int len = 2; len < MAXLEN; ++len)
      for (int x = 0; x < MAXC; ++x)
        for (auto y : edg[x])
          if (y.v != x)
            dist[x][len] = min(dist[x][len], dist[y.v][len - 1] + y.w);

    preprocess_fast();

    int q, A, B;
    scanf("%d",&q);
    REP(it, q) {
      scanf("%d %d",&A,&B);
      printf("%lld\n",solve_fast2(A,B));
    }
    return 0;
}
```

III. Bài tập luyện tập

BÀI 1. GROUP (USACO MAR08 **)**

Đề bài

Cho n≤300000 cặp số (x,y) (1≤x,y≤1000000). Ta có thể nhóm một vài cặp số lại thành một nhóm. Giả sử một nhóm gồm các cặp số thứ a1,a2,…,am thì chi phí cho nhóm này sẽ là max(xa1,xa2,…,xam)∗max(ya1,ya2,…,yam).

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).

Phân tích

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 cho kết quả xấu đi.

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 $x1 \le x \le x2$ và $y1 \ge y \ge y2$ mà không làm kết quả xấu đi. 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 i. Công thức truy hồi:

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

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 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 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)

Chương trình tham khảo.

```
#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;
}
```

Bài 2 nguồn **codeforces**

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 máy cưa (rởm), 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 a1=1≤a2≤…≤an và b1≥b2≥…≥bn=0. Tìm một cách cưa cây với chi phí nhỏ nhất.

Vd 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 chi phí bạn phải trả là 2×6+30×5+3×0+10×0+20×0=1622×6+30×5+3×0+10×0+20×0=162. Nếu bạn chặt theo thứ tự: 1→3→6→2→4→51→3→6→2→4→5 chí phí bạn phải trả là 3×6+30×4+4×0+10×0+20×0=1383×6+30×4+4×0+10×0+20×0=138

Phân tích

Từ ví dụ trên, ta có nhận xét sau:

Observation 1: 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=i1,i2,…,ik} là một thứ tự chặt cây tối ưu với ik=n. Ta có bổ đề sau:

Lemma 1: 1=i1≤i2≤…≤ik=n

Dựa vào Lemma 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=1minj<i{C[j]+A[i]B[j]}+B[j] (1)

Dễ thấy, chi phí để chặt cây thứ n là C[n]. Để ý kĩ các bạn sẽ thấy công thức quy hoạch động để tính C[n] với hai mảng A,B là các mảng đã săp xếp tương ứng với trường hợp đặc biệt 2. Do đó, có thể giải bài toán này trong thời gian O(n).

Bài 3 Tham gia Mạng

Source: The 2017 ACM - ICPC Asia Ho Chi Minh City Regional Contest

Một mạng có kích thước N chứa N máy tính được kết nối bằng N-1 cáp, do đó có chính xác 1 đường dẫn giữa bất kỳ cặp máy tính nào. Chi phí truyền giữa 2 máy tính bằng bình phương số lượng cáp trên đường dẫn kết nối 2 máy tính. Chi phí truyền của một mạng bằng tổng chi phí truyền giữa tất cả các cặp máy tính không có thứ tự.

Cho mạng A với N máy tính và mạng B với M máy tính, quản trị viên muốn tạo mạng mới C, bằng cách thêm chính xác một cáp kết nối một máy tính trong A và một máy tính trong B. Nhiệm vụ của bạn là giảm thiểu chi phí truyền của mạng mới C.

Đầu vào

• Dòng đầu tiên chứa N nguyên - số lượng máy tính trong mạng A(1≤N≤50000).

• N – 1 dòng tiếp theo, mỗi dòng chứa hai số nguyên u và v, đại diện cho một cáp kết nối máy tính u và v trong mạng A (1≤u, v≤N).

• Dòng tiếp theo chứa số nguyên M - số lượng máy tính trong mạng B (1≤M≤50000).

• M – 1 dòng tiếp theo, mỗi dòng chứa hai số nguyên u và v, đại diện cho một cáp kết nối máy tính u và v trong mạng B (1≤u, v≤M). Nó được đảm bảo rằng mỗi mạng là một cây.

input	output
3 1 2 2 3 4 1 2 1 3 1 4	96

Đầu ra

Viết trong một dòng chi phí truyền tối thiểu của mạng kết quả C.

Ví dụ

Làm rõ ví dụ

Trong mẫu đầu tiên bên dưới, kết nối máy tính 2 của mạng A và máy tính 1 của mạng B sẽ giảm thiểu chi phí truyền của mạng.

BÀI 4 NKLEAVES – Leaves **Nguồn SPOJ**

Một ngày thu đẹp trời, Radu và Mars nhận ra rằng khu vườn của họ chứa đầy lá rụng. Họ quyết định gom lá thành đúng K đống lá.

Biết rằng khu vườn có dạng một đường thẳng. 2 người đã thiết lập một hệ tọa độ với gốc ở điểm đầu của khu vườn.

Có N chiếc lá nằm thẳng hàng với trọng lượng khác nhau, khoảng cách giữa 2 chiếc lá liên tiếp là 1. Nghĩa là, chiếc lá đầu tiên có tọa độ 1, chiếc lá thứ 2 có tọa độ 2,..., chiếc lá thứ N có tọa độ N. Ban đầu, 2 người đang đứng ở tọa độ N.

Radu và Mars thực hiện việc gom lá trong khi rời khỏi khu vườn, do đó những chiếc lá chỉ có thể di chuyển về bên trái. Chi phí di chuyển một chiếc lá bằng tích của trọng lượng chếc lá và khoảng cách di chuyển. Hiển nhiên, một trong K đống lá sẽ nằm ở tọa độ 1, tuy nhiên những đống còn lại có thể nằm ở bất kỳ vị trí nào.

Yêu cầu: tìm chi phí nhỏ nhất để gom N chiếc lá thành đúng K đống lá.

Dữ liệu

- Dòng đầu tiên chứa 2 số nguyên dương N và K, cách nhau bởi 1 khoảng trắng.
- Dòng thứ i trong số N dòng tiếp theo chứa 1 số nguyên dương cho biết trọng lượng của chiếc lá thứ i.

Kết qủa

In ra 1 số nguyên là chi phí nhỏ nhất để gom N chiếc lá lại thành đúng K đống lá.

Giới hạn

- $0 < N \leq 100000$
- $0 < K \leq 10, K < N$
- Trọng lượng của mỗi chiếc lá không vượt quá 1000.

Ví dụ

Input	**Output**
5 2 1 2 3 4 5	13

Giải thích: Cách tốt nhất là đặt 2 đống lá ở vị trí 1 và 4.

KẾT LUẬN

Quy hoạch động là một phương pháp rất hay và mạnh của tin học. Nhưng để giải được các bài toán bằng phương pháp quy hoạch động thật chẳng dễ dàng chút nào. Chủ yếu học sinh hiện nay sử dụng quy hoạch động theo kiểu làm từng bài cho nhớ mẫu và áp dụng vào những bài có dạng tương tự. Kỹ thuật bao lồi để dùng tối ưu hóa thuật toán quy hoạch động trên đây giúp chúng ta có sự ứng dụng linh hoạt trong từng bài toán. Khi giải một bài toán quy hoạch động phải đặt từng bài toán dưới nhiều góc nhìn khác nhau và đánh giá rồi lựa chọn các phương án.

Trong quá trình nghiên cứu chuyên đề đã tập trung làm rõ các vấn đề:

- Tìm hiểu và trình bày lý thuyết kĩ thuật bao lồi
- Trình bày một số bài toán sử dụng Quy hoạch động lồi
- Xây dựng chương trình cài đặt 5 bài toán điển hình

Mặc dù có nhiều thời gian tìm hiểu chuyên đề tuy nhiên do khả năng trình độ còn hạn chế việc thực hiện viết chuyên đề còn nhiều thiếu sót, rất mong được sự đóng góp của bạn bè và đồng nghiệp để chuyên đề được hoàn thiện hơn.

TÀI LIỆU THAM KHẢO

1) Kĩ thuật bao lồi (Convex Hull Trick) nguồn P3G, Người dịch**: Phan Minh Hoàng**

2) Một số kĩ thuật tối ưu hoá thuật toán Quy Hoạch Động, vnoi.info, Tác giả: **Lê Anh Đức**

3) https://www.giaithuatlaptrinh.com/?p=176

4) http://ceoi.inf.elte.hu/

5) http://apio-olympiad.org/