

Mục lục – Athena VIII

Tìm kiếm cầu trên đồ thi vô hướng	3
Giải thuật tìm cầu offline	3
Giải thuật tìm cầu online	4
Hệ thống các tập không giao nhau	10
Tổ chức dữ liệu	10
Giải pháp heuristic nén đường đi	11
Giải pháp heuristic hợp nhất theo bậc của cây	12
Hợp nhất các giải pháp heuristic – nén đường đi kết hợp với chọn bậc	15
Ứng dụng của hệ thống các tập không giao nhau	15
Quản lý thành phần liên thông của đồ thị	15
Tìm thành phần liên thông trên ảnh	16
Hỗ trợ lưu thông tin bổ sung với mỗi tập hợp	16
Nén thông tin rời rạc trên đoạn thẳng	16
Quản lý khoảng cách tới đỉnh đại diện	17
Giải thuật tìm min trên một đoạn (<i>Range Minimum Query – RMQ</i>)	20
Tìm cha chung nhỏ nhất trong cây (<i>Lowest Common Ancestor – LCA</i>)	20
Hợp nhất các cấu trúc dữ liệu khác nhau	23
Tìm cầu của đồ thị trong chế độ online với độ phức tạp $O(\alpha(n))$	24
Sơ lược về lịch sử DSU	25
Bài tập	26
VV07. BỘ TRÍ DỮ LIỆU Tên chương trình: DATA.CPP	26
VU35. SỐ NHI PHÂN MỚI Tên chương trình: NEWBIN.CPP	30
VU36. TÍN CHỈ Tên chương trình: CERTI.CPP	32
VU37. LỰA CHỌN Tên chương trình: CHOICE.CPP	36
VU38. TỔNG LỚN NHẤT Tên chương trình: MAXSUM.CPP	39
VU39. MẠNG LƯỚI GIAO THÔNG Tên chương trình: TRANSNET.CPP	42
VU40. MAI PHỤC Tên chương trình: AMBUSH.CPP	46
VU41. THANH GỖ Tên chương trình: TIMBER.CPP	57
VU42. MUỒNG HOÀNG YẾN Tên chương trình: CASSIA.CPP	59
VU43. ĐỘ TRẺ CỦA SÓ Tên chương trình: YOUTH.CPP	63
VU44. PHÂN SỐ Tên chương trình: FRACTIONS.CPP	65

VU45. TÔ MÀU	<i>Tên chương trình: TINGE.CPP</i>	68
VU46. TÁC ĐỘNG HỖ TRỢ	<i>Tên chương trình: SUPPORT.CPP</i>	71
VU47. VI TRÍ	<i>Tên chương trình: NUMPOS.CPP</i>	74
VU48. THAM QUAN	<i>Tên chương trình: TOUR.CPP</i>	77
VV01. CHUNG CỤ	<i>Tên chương trình: BUILDING.CPP</i>	81
VV02. MÁY TÍNH BẤM TAY	<i>Tên chương trình: CALC.CPP</i>	84
VV03. TRIỂN LÃM	<i>Tên chương trình: EXHIBITION.CPP</i>	87
VV04. KHU NGHỈ MÁT	<i>Tên chương trình: RESORT.CPP</i>	89
VV05. XÂU ĐỌC ĐÁO	<i>Tên chương trình: VARIETY.CPP</i>	92
VV06. ĐĨA PETRI	<i>Tên chương trình: PETRI.CPP</i>	97
VV08. CHUYỂN PHÁT HÀNG	<i>Tên chương trình: DELIVERY.CPP</i>	103
VV09. MÁY GIA TỐC HẠT	<i>Tên chương trình: COLLIDER.CPP</i>	105
VV10. TRƯỜNG NĂNG LƯỢNG	<i>Tên chương trình: POWER.CPP</i>	110
VV11. BỒI DƯỠNG NÂNG CAO	<i>Tên chương trình: TRAINING.CPP</i>	113
VV12. KEO EPOXY	<i>Tên chương trình: EPOXY.CPP</i>	125
VV13. ĐOÁN NHẬN NGHỀ NGHIỆP	<i>Tên chương trình: RECOGNIZE.CPP</i>	127
VV14. LẮP RÁP	<i>Tên chương trình: ASSEMBLE.CPP</i>	131
VV15. NGHIÊM TÚC	<i>Tên chương trình: SERIOUS.CPP</i>	136
VV16. PHAO NỘI	<i>Tên chương trình: FLOAT.CPP</i>	147

Tìm kiếm cầu trên đồ thị vô hướng

Xét đồ thị vô hướng G có n đỉnh và m cạnh. Giả thiết giữa 2 đỉnh của G có *không quá một cạnh* nối trực tiếp. Tập các đỉnh của G mà giữa 2 đỉnh bất kỳ của tập có đường đi tới nhau (trực tiếp hoặc qua các đỉnh khác) gọi là một thành phần liên thông.

Cầu của đồ thị G là cạnh mà nếu bỏ nó số thành phần của G sẽ tăng.

Có 2 loại giải thuật chính trong việc tìm cầu ứng với các trường hợp:

Trường hợp đồ thị đã cho trước và không thay đổi trong quá trình tìm kiếm: Giải thuật offline,

Đồ thị G thay đổi (thêm, bớt cạnh) trong quá trình tìm cầu: Giải thuật online.

Giải thuật tìm cầu offline

Đồ thị G đã cho không thay đổi trong quá trình giải bài toán. Việc tìm kiếm có thể thực hiện bằng giải thuật loang theo chiều sâu với *độ phức tạp $O(n+m)$* .

Gọi **root** là gốc của đồ thị. Xuất phát từ **root**, loang theo chiều sâu (*dfs*) ta tới đỉnh **v**. Nếu với **v** tồn tại cạnh (**v**, **to**) và từ đỉnh **to** hoặc từ các đỉnh con của **to** không có đường nối tới **v** hay tới đỉnh cha của **v** thì (**v**, **to**) là một cầu. Trong trường hợp ngược lại (**v**, **to**) không phải là cầu.

Phương pháp hiệu quả nhận biết điều kiện trên được thực hiện dựa trên thời điểm vào của mỗi đỉnh.

Gọi **tin[v]** – thời điểm vào đỉnh **v** khi duyệt theo chiều sâu.

Xây dựng mảng hỗ trợ **fup[v]**:

$$fup[v] = \min \begin{cases} tin[v], \\ tin[p] \text{ với mọi } p \text{ dẫn tới } v \text{ hoặc đỉnh cha của } v, \\ fup[to] \text{ với mọi } (v, to). \end{cases}$$

Ta thấy, từ **v** hoặc từ đỉnh con của **v** có đỉnh nối tới đỉnh cha của **v** khi và chỉ khi tồn tại đỉnh con **to** thỏa mãn điều kiện **fup[to] ≤ tin[v]**. Điều kiện bằng xảy ra khi tồn tại đỉnh con dẫn tới đúng **v**.

Ngoài ra, còn cần mảng **used[to]** đánh dấu đỉnh khi duyệt:

used[to] = false – đỉnh chưa được duyệt,

used[to] = true và **to ≠ parent** – điều kiện đi ngược lên trên,

to = parent – điều kiện cạnh dẫn ngược đã xét.

Giải thuật:

```
const int MAXN = ...;
vector<int> g[MAXN];
bool used[MAXN];
int timer, tin[MAXN], fup[MAXN];

void dfs (int v, int p = -1) {
    used[v] = true;
    tin[v] = fup[v] = timer++;
    for (size_t i=0; i<g[v].size(); ++i) {
        int to = g[v][i];
        if (to == p) continue;
        if (used[to])
            fup[v] = min (fup[v], tin[to]);
        else {
            dfs (to, v);
            fup[v] = min (fup[v], fup[to]);
            if (fup[to] > tin[v])
                IS_BRIDGE(v,to);
        }
    }
}

void find_bridges() {
    timer = 0;
    for (int i=0; i<n; ++i)
        used[i] = false;
    for (int i=0; i<n; ++i)
        if (!used[i])
            dfs (i);
}
```

Hàm xử lý câu tìm được

Chuẩn bị và kích hoạt tìm kiếm

Được phép trong C++

Giải thuật tìm cầu online

Giả thiết đồ thị G n đỉnh, trong đó giữa 2 đỉnh bất kỳ có không quá một đường nối trực tiếp. Xét trường hợp G không biết trước, từng cạnh một sẽ được *bổ sung dần* trong quá trình xử lý.

Ban đầu G không chứa cạnh nào. Có m truy vấn, mỗi truy vấn là yêu cầu xác định các cầu của đồ thị sau khi bổ sung thêm cạnh (a, b).

Giải thuật đặt nền tảng trên cấu trúc dữ liệu của hệ thống các tập không giao nhau và có độ phức tạp $O(n \log n + m)$.

Các cầu chia đỉnh của đồ thị thành các nhóm song liên thông. Nếu ánh xạ mỗi nhóm thành một đỉnh và chỉ giữ lại các cạnh là cầu thì ta có một rừng, tức là nhiều cây rời rạc.

Ban đầu, khi đồ thị còn rỗng (không có cạnh nào) ta có n nhóm đỉnh song liên thông 2 chiều và các nhóm này không giao nhau.

Có 3 tình huống có thể xảy ra khi bổ sung thêm một cạnh (**a**, **b**):

- ⊕ Cả 2 đỉnh **a** và **b** đều thuộc một nhóm song liên thông, như vậy cạnh này không thể là cầu, cấu trúc của rừng không thay đổi và có thể bỏ qua, không xử lý cạnh,
- ⊕ Các đỉnh **a** và **b** thuộc 2 nhóm liên thông khác nhau, (**a**, **b**) là một cầu nối 2 cây, 2 nhóm liên thông này sẽ được hợp nhất thành một và giữ nguyên các cầu đã có, số lượng cầu tăng thêm 1,
- ⊕ Các đỉnh **a** và **b** thuộc một nhóm liên thông nhưng thuộc 2 nhóm song liên thông khác nhau. Trong trường hợp này cạnh mới sẽ tạo với một số cầu hiện có thành chu trình, những cầu đó bây giờ không phải là cầu, các nhóm song liên thông tham gia vào chu trình hình thành được hợp nhất thành một nhóm song liên thông mới. Số lượng cầu của đồ thị sẽ giảm.

Toàn bộ giải thuật xoáy quanh việc nhận biết và xử lý một cách hiệu quả 3 tình huống trên.

Tổ chức dữ liệu lưu trữ rừng

Ta cần 2 hệ thống lưu trữ các tập không giao nhau, một để lưu trữ các *thành phần liên thông* và cấu trúc thứ 2 – lưu trữ các *thành phần song liên thông*.

Ngoài ra, còn cần thêm mảng **par[]** móc nối tới nút cha để lưu trữ cây trong các thành phần song liên thông.

Các phép xử lý cần hiện thực hóa

- ♣ Kiểm tra 2 đỉnh có thuộc một nhóm liên thông/song liên thông hay không. Việc kiểm tra được thực hiện theo sơ đồ xử lý chuẩn hệ thống các tập không giao nhau.
- ♣ Hợp nhất 2 cây theo cạnh (**a**, **b**). Các đỉnh **a** và **b** có thể không phải là gốc của cây nào trong 2 cây cần hợp nhất, vì vậy treo một cây vào cây khác. Ví dụ, có thể treo một cây vào đỉnh **a**, sau đó biến đỉnh **a** thành con của **b** và gắn vào đó cây thứ 2.

Như vậy quá trình xử lý đòi hỏi phải có giải thuật hiệu quả việc treo cây gốc **r** vào đỉnh **v**. Để làm được việc đó cần tìm đường đi từ **v** tới **r**, đảo móc nối **par[]** trên

đường đi, đồng thời cập nhật mốc nối tương ứng trong dữ liệu về hệ thống các tập không giao nhau tương ứng với nhóm liên thông đang xử lý.

Như vậy độ phức tạp của việc treo cây là $O(h)$, trong đó h là độ cao của cây. Về toàn cục, có thể đánh giá độ phức tạp là $O(\text{size})$, trong đó **size** – số đỉnh trong cây.

Để đảm bảo hiệu quả cao cần treo cây có ít đỉnh hơn vào cây còn lại.

Gọi **T(n)** là số phép tính cần thực hiện để nhận được cây n đỉnh từ các phép hợp nhất và treo cây, ta có:

$$T(n) = \max_{k=1 \div n-1} \{ T(k) + T(n-k) + O(n) \} = O(n \log n)$$

- Tìm chu trình hình thành bởi cạnh (**a**, **b**) được bổ sung. Trên thực tế, đó là việc tìm nút cha chung nhỏ nhất (**LCA**) của 2 đỉnh **a** và **b**. Do cuối cùng ta sẽ thay tất cả các đỉnh trong nhóm bằng một đỉnh đại diện nên có thể sử dụng giải thuật bất kỳ tìm **LCA** với độ phức tạp tỷ lệ với độ dài cần duyệt.

Vì trong tay ta chỉ có **par[]** – thông tin mốc nối tới nút cha, nên con đường duy nhất tìm **LCA** ở đây là đánh dấu các đỉnh cha từ **a** và từ **b** cho đến khi gặp đỉnh đã được đánh dấu, đó sẽ là **LCA** cần tìm. Xác định lại đường từ nút này tới **a** và tới **b** ta sẽ có chu trình cần tìm.

Rõ ràng giả thuật tìm kiếm trên có độ phức tạp bằng độ dài đường đi.

- Nén chu trình hình thành bởi sự xuất hiện của cạnh mới (**a**, **b**). Ta phải tạo nhóm song liên thông mới mà không làm thay đổi cấu trúc cây, không thay đổi các mốc nối **par[]** và bảo toàn tính đúng đắn của các tập không giao nhau.

Phương pháp đơn giản nhất là nén các đỉnh trong chu trình mới tìm được vào **LCA** của chúng. Thật vậy, đỉnh **LCA** là cao nhất trong số các đỉnh được nén, vì vậy **par[]** của nó sẽ vẫn giữ nguyên như cũ. Thông tin về các đỉnh còn lại cũng không cần cập nhật – chúng trở nên “trong suốt” trong các phép xử lý tiếp theo, còn trong hệ thống các tập không giao nhau các đỉnh này được trích dẫn tới đại diện là đỉnh **LCA**. Ta cũng không cần sắp xếp hay cập nhật trình tự đỉnh đại diện vì khi cần duyệt đã có **par[]** đảm bảo.

Độ phức tạp xử lý theo đúng sơ đồ lý thuyết về hệ thống các tập không giao nhau sẽ là $O(\log n)$. Tuy vậy, với đặc thù của bài toán đồ thị, ta có thể xử lý với **độ phức tạp $O(1)$** : đơn thuần là gán cho **par[LCA]** giá trị **par** mới!

Giải thuật:

Giải thuật nêu dưới đây không lưu trữ thông tin về các cầu mà chỉ lưu trữ số lượng cầu trong biến **bridges**. Nếu cần thông tin về chính các cầu có thể tổ chức một tập hợp **s** để lưu trữ.

Hàm **init()** có nhiệm vụ khởi tạo hai hệ thống lưu trữ các tập không giao nhau cũng như giá trị đầu của các **par[]**.

```
const int MAXN = ...;
int n, bridges, par[MAXN], bl[MAXN], comp[MAXN], size[MAXN];

void init() {
    for (int i=0; i<n; ++i) {
        bl[i] = comp[i] = i;
        size[i] = 1;
        par[i] = -1;
    }
    bridges = 0;
}

int get (int v) {
    if (v== -1) return -1;
    return bl[v]==v ? v : bl[v]=get(bl[v]);
}

int get_comp (int v) {
    v = get(v);
    return comp[v]==v ? v : comp[v]=get_comp(comp[v]);
}

void make_root (int v) {
    v = get(v);
    int root = v,
        child = -1;
    while (v != -1) {
        int p = get(par[v]);
        par[v] = child;
        comp[v] = root;
        child=v; v=p;
    }
    size[root] = size[child];
}

int cu, u[MAXN];

void merge_path (int a, int b) {
    ++cu;

    vector<int> va, vb;
```

```

int lca = -1;
for(;;) {
    if (a != -1) {
        a = get(a);
        va.pb (a);

        if (u[a] == cu) {
            lca = a;
            break;
        }
        u[a] = cu;

        a = par[a];
    }

    if (b != -1) {
        b = get(b);
        vb.pb (b);

        if (u[b] == cu) {
            lca = b;
            break;
        }
        u[b] = cu;

        b = par[b];
    }
}

for (size_t i=0; i<va.size(); ++i) {
    bl[va[i]] = lca;
    if (va[i] == lca) break;
    --bridges;
}
for (size_t i=0; i<vb.size(); ++i) {
    bl[vb[i]] = lca;
    if (vb[i] == lca) break;
    --bridges;
}
}

void add_edge (int a, int b) {
    a = get(a);    b = get(b);
    if (a == b) return;

    int ca = get_comp(a),
        cb = get_comp(b);
    if (ca != cb) {
        ++bridges;
        if (size[ca] > size[cb]) {
            swap (a, b);
            swap (ca, cb);

```

```

        }
        make_root (a) ;
        par[a] = comp[a] = b;
        size[cb] += size[a];
    }
else
    merge_path (a, b);
}
// Kết giải thuật

```

Chú thích:

Mảng **bl []** dùng để lưu trữ các tập không giao nhau áp dụng trên các nhóm đỉnh song liên thông,

Hàm **get (v)** trả về giá trị đỉnh đại diện cho nhóm đỉnh song liên thông, sau khi nén các đỉnh trong nhóm song liên thông trở nên “trong suốt”, mọi xử lý sẽ được thực hiện với đỉnh đại diện,

Mảng **comp []** – lưu các tập không giao nhau áp dụng trên các nhóm đỉnh liên thông,

Mảng **size []** – lưu kích thước tập xác định bởi comp,

Hàm **get_comp (v)** trả về đỉnh đại diện của nhóm liên thông (trên thực tế đó là gốc của cây),

Hàm **make_root (v)** có nhiệm vụ treo cây vào đỉnh **v**: di chuyển từ đỉnh **v** theo các nút cha về gốc, đảo mốc nối **par []** (cho chỉ xuống dưới, hướng về **v**), đồng thời cập nhật con trỏ **comp []** để chỉ về gốc mới, sau khi gắn cây vào gốc mới – cập nhật **size** của nhóm liên thông. Lưu ý là trong quá trình làm việc phải sử dụng hàm **get ()** để lấy thông tin về đỉnh đại diện nhóm liên thông vì các đỉnh trong nhóm về mặt lô gic – đã bị xóa.

Hàm **merge_path (a, b)** – tìm **LCA** của các đỉnh **a** và **b**. Việc dùng mảng đánh dấu sẽ tiết kiệm nhiều thời gian hơn dùng **set**. Các đường đã đi qua được lưu trữ trong **va** và **vb** để sau này duyệt lại, tìm tất cả các đỉnh thuộc chu trình. Tất cả các đỉnh này được nén lại, bằng cách nối chúng với **LCA**. Độ phức tạp của công đoạn này là O(logn). Số lượng các cạnh đã đi qua (số cầu cũ) được đếm đồng thời các xử lý khác và sau đó giảm giá trị số lượng cầu **bridges** một cách tương ứng.

Hàm xử lý truy vấn **add_edge (a, b)** xác định các thành phần liên thông chứa **a** và **b**. Nếu **a** và **b** thuộc các thành phần liên thông khác nhau thì treo cây nhỏ hơn vào gốc mới, sau đó hợp nhất kết quả với cây lớn hơn. Nếu **a** và **b** ở cùng một cây nhưng thuộc các nhóm song liên thông khác nhau thì gọi hàm **merge_path (a, b)** để tìm chu trình và sau đó – hợp nhất thành một nhóm song liên thông.

Hệ thống các tập không giao nhau

Hệ thống các tập không giao nhau (*disjoint-set-union – DSU*) là một cách tổ chức dữ liệu đơn giản nhưng rất hiệu quả để giải quyết nhiều loại bài toán khác nhau.

Ban đầu mỗi phần tử của dữ liệu thuộc một tập riêng. Các phép xử lý cơ sở trên cấu trúc là:

- ✚ **union_sets(x, y)** – Hợp nhất hai tập: tập chứa phần tử **x** với tập chứa phần tử **y**,
- ✚ **find_set(x)** – Xác định tập chứa phần tử **x**, chính xác hơn – trả về một phần tử (được gọi là *phần tử đại diện*) của tập,
- ✚ **make_set(x)** – Tạo tập riêng chứa phần tử **x** mới.

Nếu gọi hàm **find_set()** với hai phần tử khác nhau và nhận được cùng một kết quả thì có nghĩa là 2 phần tử đó thuộc cùng một tập hợp. Trong trường hợp kết quả khác nhau – hai phần tử thuộc 2 tập khác nhau.

Cấu trúc dữ liệu mô tả dưới đây cho phép thực hiện các phép xử lý trên với độ phức tạp xấp xỉ O(1).

Tổ chức dữ liệu

Các phần tử của mỗi tập hợp được lưu trữ dưới dạng cây và gốc của cây là phần tử đại diện cho tập.

Với tất cả các cây cần có *một mảng* parent lưu *mốc nối tới phần tử cha* trong cây. Với *phần tử đại diện*, mốc nối này chỉ tới chính nó.

Cách tổ chức giản đơn

Đầu tiên ta sẽ xét cách tổ chức tầm thường, tuy hoạt động không thật hiệu quả nhưng tạo tiền đề để cải tiến và nâng cấp.

Để tạo tập mới (phép **make_set(v)**) ta cho parent[v] chỉ tới chính nó, tạo cây mới chỉ chứa một phần tử.

Để hợp nhất 2 phần tử (phép **union_sets(a, b)**) ta tìm các đại diện của tập chứa a và tập chứa b. Nếu 2 đại diện này trùng nhau – không phải làm gì! Cả 2 phần tử đều thuộc một tập. Nếu 2 đại diện khác nhau: cho mốc nối cha của b chỉ tới a (hoặc ngược lại) và kết quả là 2 cây được hợp nhất thành một.

Việc tìm đại diện (phép **find_set(v)**) hoàn toàn đơn giản: duyệt cây từ đỉnh v cho đến khi gặp phần tử chỉ tới chính nó. Việc duyệt có thể thực hiện theo sơ đồ đệ quy.

```
void make_set (int v) {  
    parent[v] = v;
```

```

}

int find_set (int v) {
    if (v == parent[v])
        return v;
    return find_set (parent[v]);
}

void union_sets (int a, int b) {
    a = find_set (a);
    b = find_set (b);
    if (a != b)
        parent[b] = a;
}

```

Cách tổ chức trên đơn giản nhưng không hiệu quả. Sau nhiều lần hợp nhất cây có thể suy biến thành một chuỗi móc nối dài và độ phức tạp của phép **find_set()** sẽ tiến dần tới $O(n)$.

Có hai kỹ thuật tối ưu hóa cho phép tăng đáng kể hiệu quả tìm kiếm (ngay cả khi áp dụng một cách riêng rẽ các kỹ thuật này).

Giải pháp heuristic nén đường đi

Giải pháp này được áp dụng để nâng cao hiệu quả hoạt động của hàm tìm kiếm **find_set(v)**.

Khi gọi hàm này ta phải đi qua một con đường dài dẫn tới phần tử đại diện **p**. Có thể sê đơn giản hơn nếu **parent[]** của tất cả các phần tử tập hợp đều chỉ trực tiếp tới **p**? Như vậy vai trò của **parent[]** có thay đổi đôi chút. Mảng móc nối chỉ tới nút cha đã bị nén để chỉ xa hơn. Tuy vậy cũng dễ dàng thấy rằng không thể tổ chức để **parent[]** luôn chỉ tới phần tử đại diện. Khi đó ta sẽ gặp khó khăn với phép hợp nhất **union_sets()**: khi hợp nhất ta phải cập nhật móc nối tới đại diện ở $O(n)$ phần tử tập hợp. Như vậy việc nén chỉ nên thực hiện bộ phận.

Hàm **find_set()** sẽ có dạng mới như sau:

```

int find_set (int v) {
    if (v == parent[v])
        return v;
    parent[v] = find_set (parent[v]);
}

```

Sự cải tiến đơn giản này đã mang lại hiệu quả mà ta đang chờ đợi. Đầu tiên, bằng cách gọi đệ quy ta *tìm được phần tử đại diện*, sau đó quá trình làm rỗng stack sẽ *gán vị trí phần tử này cho các móc nối parent* ở các phần tử đã đi qua.

Giải thuật này có thể thực hiện theo sơ đồ lặp, nhưng khi đó ta phải *duyệt đường đi 2 lần*: một lần để tìm đại diện, lần thứ 2 – để cập nhật mốc nối, hiệu quả xử lý gần như không thay đổi.

Đánh giá hiệu quả của việc ứng dụng giải pháp heuristic nén đường đi

Ta sẽ chứng minh việc nén đường đi theo giải pháp heuristic mang lại hiệu quả trung bình $O(\log n)$ cho mỗi truy vấn.

Lưu ý rằng phép hợp nhất `union_sets()` đòi hỏi thực hiện `find_set()` hai lần và chi phí $O(1)$ để cập nhật mốc nối, vì vậy ta chỉ phải tập trung đánh giá độ phức tạp $O(m)$ của phép `find_set()`.

Gọi số đỉnh con của v (kể cả chính nó) là trọng số của v và ký hiệu là $w[v]$. Rõ ràng, trọng số các đỉnh chỉ có thể tăng trong quá trình xử lý truy vấn.

Gọi độ bao trùm của cạnh (a, b) là giá trị tuyệt đối hiệu các trọng số: $|w[a] - w[b]|$. Với cạnh (a, b) cố định độ bao trùm chỉ có thể tăng trong quá trình xử lý.

Các cạnh được phân loại thành các lớp. Một cạnh thuộc lớp k khi độ bao trùm của nó nằm trong khoảng $[2^k, 2^{k+1}-1]$. Như vậy lớp của cạnh là một số nằm trong phạm vi từ 0 đến $\lceil \log n \rceil$.

Chọn một đỉnh x cố định và xét sự thay đổi cạnh từ nó tới đỉnh cha. Ban đầu cạnh không tồn tại (chừng nào x còn là đỉnh đại diện). Sau đó x được nối tới một đỉnh nào đó (do kết quả hợp nhất tập) và rồi mốc nối lại thay đổi bởi hiệu ứng nén khi hàm `find_set()` được gọi.

Xét hoạt động của một lần gọi hàm `find_set()`: cây được duyệt theo một đường đi nào đó, mốc nối parent của các đỉnh trên đường đi bị thay bằng mốc nối chỉ tới đỉnh đại diện. Xet đường đi này và loại bỏ cạnh cuối cùng ở mỗi lớp (tức là không quá 1 cạnh ở mỗi lớp trong số các lớp từ 0 đến $\lceil \log n \rceil$). Như vậy có $O(\log n)$ cạnh bị loại từ mỗi truy vấn.

Xét các cạnh còn lại của đường đi này. Nếu một cạnh có lớp là k thì có nghĩa là trên đường đi còn có ít nhất một cạnh lớp k nữa (vì nếu nó là duy nhất ở lớp k thì đã bị xóa). Do đó, sau khi nén đường đi cạnh này được thay thế bởi cạnh có lớp tối thiểu là $k+1$. Trọng số của cạnh thay giảm nên với mỗi đỉnh được duyệt qua bởi `find_set()` cạnh từ nó tới đỉnh cha hoặc bị xóa hoặc chuyển sang lớp cao hơn.

Tổng hợp các đánh giá trên ta thấy độ phức tạp xử lý m truy vấn sẽ là $O((n+m)\log n)$. Khi $m \geq n$ độ phức tạp trung bình xử lý một truy vấn có bậc $O(\log n)$.

Giải pháp heuristic hợp nhất theo bậc của cây

Một giải pháp khác nâng cao hiệu quả giải thuật là cải tiến cách hợp nhất hai tập. Giải pháp này kết hợp với nén đường đi sẽ cho phép xử lý mỗi truy vấn với độ phức tạp gần như là một hằng.

Phép hợp nhất tập union_sets() được thực hiện theo 2 bước:

- ▣ Lựa chọn cây đưa đi kết nối dựa vào một trong 2 tiêu chí đánh giá bậc của cây:
 - ♠ Xác định bậc của cây theo số lượng đỉnh,
 - ♠ Xác định bậc của cây theo độ sâu, hay chính xác hơn – giới hạn trên của độ sâu,
- ▣ Cây có bậc thấp hơn sẽ được gắn và cây có bậc cao.

Giải thuật hợp nhất tập *dựa trên số đỉnh* của cây sẽ có dạng như sau:

```
void make_set (int v) {  
    parent[v] = v;  
    size[v] = 1;  
}  
  
void union_sets (int a, int b) {  
    a = find_set (a);  
    b = find_set (b);  
    if (a != b) {  
        if (size[a] < size[b])  
            swap (a, b);  
        parent[b] = a;  
        size[a] += size[b];  
    }  
}
```

Giải thuật hợp nhất tập *dựa theo độ sâu* của cây sẽ có dạng như sau:

```
void make_set (int v) {  
    parent[v] = v;  
    rank[v] = 0;  
}  
  
void union_sets (int a, int b) {  
    a = find_set (a);  
    b = find_set (b);  
    if (a != b) {  
        if (rank[a] < rank[b])  
            swap (a, b);  
        parent[b] = a;  
        if (rank[a] == rank[b])  
            ++rank[a];  
    }  
}
```

Cả hai phương án là tương đương trên quan điểm hiệu quả, vì vậy khi triển khai ứng dụng có thể lựa chọn phương pháp bất kỳ.

Đánh giá hiệu quả của giải pháp heuristic hợp nhất theo bậc

Ta sẽ chứng minh độ phức tạp xử lý hợp nhất tập với sự *lựa chọn heuristic theo bậc và không áp dụng kỹ thuật nén đường đi* là $O(\log n)$.

Không phụ thuộc vào cách lựa chọn kiểu xác định bậc ta sẽ chứng minh độ sâu của mỗi cây sẽ có bậc $O(\log n)$, điều này tự động dẫn đến độ phức tạp xử lý `find_set()` có bậc lô ga rit và kéo theo độ phức tạp xử lý `union_sets()` có sẽ có bậc lô ga rit.

Đầu tiên xét việc *đánh giá bậc theo độ sâu* của cây.

Bằng phương pháp quy nạp ta sẽ chứng minh, nếu bậc của cây bằng k thì cây phải có tối thiểu 2^k đỉnh:

- Với $k = 0$ – đó là điều hiển nhiên,
- Giả thiết điều cần chứng minh là đúng với $k-1$. Khi nén đường đi độ sâu của cây chỉ có thể giảm. Độ sâu của cây sẽ tăng từ $k-1$ lên k khi gắn vào nó cây bậc $k-1$. Khi gắn 2 cây độ sâu $k-1$ ta được cây độ sâu k và tổng số đỉnh, từ giả thiết quy nạp – sẽ không ít hơn 2^k , đó là điều phải chứng minh.

Xét việc *đánh giá bậc theo số đỉnh* của cây.

Nếu cây có số lượng đỉnh là \mathbf{k} thì độ sâu của nó không vượt quá $\lfloor \log k \rfloor$. Điều này có thể chứng minh bằng quy nạp:

- Với $\mathbf{k} = 1$ – kết luận trên là hiển nhiên,
- Xét việc hợp nhất 2 cây bậc \mathbf{k}_1 và \mathbf{k}_2 . Theo giả thiết quy nạp, độ sâu của cây tương ứng không vượt quá $\lfloor \log k_1 \rfloor$ và $\lfloor \log k_2 \rfloor$. Không mất tính chất tổng quát, có thể coi $\mathbf{k}_1 \geq \mathbf{k}_2$. Sau khi hợp nhất, độ sâu của cây có $\mathbf{k}_1 + \mathbf{k}_2$ đỉnh sẽ là

$$h = \max(\lfloor \log k_1 \rfloor, 1 + \lfloor \log k_2 \rfloor).$$

$$\begin{aligned} h &\stackrel{?}{\leq} \lfloor \log(k_1 + k_2) \rfloor, \\ 2^h &= \max(2^{\lfloor \log k_1 \rfloor}, 2^{\lfloor \log k_2 \rfloor}) \stackrel{?}{\leq} 2^{\lfloor \log(k_1 + k_2) \rfloor}, \end{aligned}$$

Vấn đề còn lại chỉ là phải chứng minh:

Nhưng đây là điều khá hiển nhiên bởi vì $\mathbf{k}_1 \leq \mathbf{k}_1 + \mathbf{k}_2$ và $2\mathbf{k}_2 \leq \mathbf{k}_1 + \mathbf{k}_2$.

Hợp nhất các giải pháp heuristic – nén đường đi kết hợp với chọn bậc

Việc kết hợp các giải pháp heuristic ở 2 hàm xử lý sẽ làm cho độ phức tạp xử lý mỗi truy vấn trở thành một hằng! Chứng minh điều này khá phức tạp và đã được nêu trong nhiều tài liệu khác nhau, ví dụ của tác giả Tarjan năm 1975, Kurt Mehlhorn và Peter Sanders năm 2008.

Người ta đã chứng minh được rằng, khi áp dụng kết hợp các giải pháp heuristic độ phức tạp xử lý mỗi truy vấn sẽ là $O(\alpha(n))$, trong đó $\alpha(n)$ là hàm nghịch đảo Akkerman có độ tăng rất chậm, đến mức trong phạm vi $n \leq 10^{600}$ $\alpha(n)$ có giá trị không quá 4.

Vì vậy có thể nói hệ thống các tập không giao nhau hoạt động với độ phức tạp gần như là một hằng.

Giải thuật sau hiện thực hóa việc kết hợp nén đường đi với xác định bậc theo độ sâu:

```
void make_set (int v) {
    parent[v] = v;
    rank[v] = 0;
}

int find_set (int v) {
    if (v == parent[v])
        return v;
    return parent[v] = find_set (parent[v]);
}

void union_sets (int a, int b) {
    a = find_set (a);
    b = find_set (b);
    if (a != b) {
        if (rank[a] < rank[b])
            swap (a, b);
        parent[b] = a;
        if (rank[a] == rank[b])
            ++rank[a];
    }
}
```

Ứng dụng của hệ thống các tập không giao nhau

Hệ thống các tập không giao nhau có thể áp dụng trực tiếp hoặc với một số cải biến nhỏ để giải một cách hiệu quả nhiều bài toán tin học khác nhau.

Quản lý thành phần liên thông của đồ thị

Đây là ứng dụng hiển nhiên của hệ thống các tập không giao nhau và là động lực đưa tới sự hình thành lý thuyết về hệ thống các tập không giao nhau.

Bài toán được phát biểu một cách hình thức như sau: Xuất phát từ đồ thị rỗng ban đầu người ta lần lượt bổ sung các đỉnh và cạnh vô hướng đan xen với truy vấn (**a**, **b**) – “các đỉnh **a** và **b** có cùng thuộc một nhóm liên thông hay không?”

Áp dụng các giải thuật đã nêu, ta có thời gian xử lý mỗi truy vấn gần như một hằng. Bài toán này cũng có thể giải quyết theo thuật toán Kruscal, nhưng ở đây ta có công cụ hiệu quả hơn nhiều, có độ phức tạp tỷ lệ tuyến tính với số truy vấn.

Đôi khi trong thực tế ta gặp bài toán nghịch đảo: Ban đầu có một đồ thị với số lượng đỉnh và cạnh nào đó. Cần xử lý các truy vấn xóa bớt cạnh.

Nếu bài toán cho ở chế độ offline, tức là ta biết được mọi truy vấn trước khi giải bài toán thì có thể đảo ngược quá trình xử lý: xuất phát từ đồ thị rỗng, duyệt từ truy vấn cuối cùng về đầu, thay việc xóa cạnh bằng bổ sung dần cạnh. Như vậy, có thể trực tiếp áp dụng giải thuật đã nêu, không cần phải bổ sung, sửa đổi.

Tìm thành phần liên thông trên ảnh

Bài toán: Xét ảnh hình chữ nhật hình thành từ $n \times m$ pixels. Ban đầu tất cả các pixels có màu trắng. Dần dần xuất hiện các pixels màu đen. Hãy xác định kích thước các miền liên thông màu trắng.

Để giải bài toán này ta duyệt tất cả các ô màu trắng, với mỗi ô – xét 4 ô kề cạnh, nếu là ô trắng thì gọi hàm **union_sets()** liên kết các ô này. Như vậy ta có DSU với $n \times m$ đỉnh. Số lượng cây trong DSU sẽ là kết quả cần tìm.

Bài toán này có thể giải bằng phương pháp loang theo chiều sâu hoặc chiều rộng, nhưng nếu ứng dụng DSU ta chỉ cần xử lý theo dòng hoặc cột với yêu cầu bộ nhớ chỉ là $O(\min(n,m))$.

Hỗ trợ lưu thông tin bổ sung với mỗi tập hợp

Các thông tin bổ sung có thể coi như *thuộc tính của đỉnh đại diện* và có thể lưu trữ độc lập với sơ đồ xử lý. Ví dụ, ta có thể lưu kích thước các miền liên thông.

Việc cập nhật thông tin bổ sung không làm thay đổi sơ đồ xử lý chung.

Như vậy cấu trúc dữ liệu này có tính mở và phục vụ được cho nhiều yêu cầu cụ thể khác nhau.

Nén thông tin rác trên đoạn thẳng

Nếu ta có một tập các phần tử và từ mỗi phần tử có một cạnh đi ra thì với DSU ta có thể nhanh chóng xác định được điểm cuối nếu bắt đầu chuyển từ điểm ban đầu cho trước.

Xét bài toán tô màu các đoạn con: Xét đoạn thẳng độ dài **L**. Ban đầu mọi đoạn con độ dài đơn vị kể từ đầu đoạn thẳng đều có màu số 0. Cho một số truy vấn dạng (**1**,

r, c) – tô đoạn thẳng [l, r] bằng màu c. Hãy xác định màu của mỗi đoạn đơn vị. Các truy vấn được cho trước, tức là bài toán thuộc loại offline.

Để giải bài toán ta cần xây dựng cấu trúc DSU ở mỗi đoạn đơn vị (gọi ngắn gọn là ô) lưu mốc nối tới đoạn đơn vị bên phải gần nhất chưa được tô. Ban đầu mốc nối chỉ tới chính nó. Sau khi tô màu lần đầu tiên ô trước đoạn được tô chỉ tới ô sau đoạn đã tô.

Để giải bài toán ta xét các truy vấn *ngược từ cuối về đầu*. Để thực hiện truy vấn cần tìm ô trái nhất chưa được tô trong đoạn thẳng và chuyển mốc nối của nó sang cho ô tiếp theo bên phải.

Như vậy ta có một DSU với giải pháp nén đường đi nhưng không phân cấp theo bậc (vì ta rất cần biết ô nào trở thành đại diện sau khi hợp nhất). Chi phí xử lý có bậc O(logn).

Các hàm xử lý:

```
void init() {
    for (int i=0; i<L; ++i)
        make_set (i);
}

void process_query (int l, int r, int c) {
    for (int v=l; ; ) {
        v = find_set (v);
        if (v >= r) break;
        answer[v] = c;
        parent[v] = v+1;
    }
}
```

Cũng có thể áp dụng kỹ thuật phân bậc ở đây: dùng mảng end[] lưu điểm cuối (điểm phải nhất) của mỗi tập. Khi đó việc hợp nhất hai tập có thể thực hiện theo bậc của tập thông qua việc cập nhật giới hạn phải mới. Độ phức tạp của giải thuật sẽ là O($\alpha(n)$).

Quản lý khoảng cách tới đỉnh đại diện

Trong một trường hợp ta cần phải tính khoảng cách theo cạnh từ một đỉnh tới gốc. Nếu không có nén đường đi thì mọi việc đều đơn giản: khoảng cách bằng số lần gọi đệ quy trong hàm **find_set()**.

Nhưng việc nén đã biến đường đi trên vài cạnh thành một cạnh. Như vậy ở mỗi đỉnh cần lưu thêm thông tin phụ: độ dài của cạnh này tính từ đỉnh đó tới đỉnh cha mà nó chỉ tới. Khi đó mảng **parent[]** lưu không phải một số nguyên mà là cặp giá trị: *đỉnh cha và khoảng cách tới đó*.

Chương trình sẽ có dạng:

```

void make_set (int v) {
    parent[v] = make_pair (v, 0);
    rank[v] = 0;
}

pair<int,int> find_set (int v) {
    if (v != parent[v].first) {
        int len = parent[v].second;
        parent[v] = find_set (parent[v].first);
        parent[v].second += len;
    }
    return parent[v];
}

void union_sets (int a, int b) {
    a = find_set (a).first;
    b = find_set (b).first;
    if (a != b) {
        if (rank[a] < rank[b])
            swap (a, b);
        parent[b] = make_pair (a, 1);
        if (rank[a] == rank[b])
            ++rank[a];
    }
}

```

Quản lý tính chẵn lẻ của độ dài đường đi và kiểm tra khả năng tồn tại đồ thị hai phía
 Nếu tính được độ dài đường đi thì tại sao lại phải xét riêng việc đánh tính chẵn – lẻ
 của độ dài?

Vấn đề ở chỗ là tính chẵn lẻ của độ dài đường đi là cần thiết để kiểm tra, sau khi bỏ
 sung cạnh thành phần liên thông có trở thành đồ thị hai phía hay không?

Để giải quyết vấn đề này ta tạo DSU quản lý các thành phần liên thông và lưu ở mỗi
 đỉnh tính chẵn lẻ của độ dài đường đi. Khi đó, mỗi lần bổ sung thêm cạnh ta dễ dàng
 biết được việc bổ sung này có phá vỡ tính hai phía của đồ thị hay không. Nếu các
 đỉnh của cạnh đề thuộc một thành phần liên thông và có độ chẵn lẻ của độ dài đường
 đi tới đỉnh giống nhau thì cạnh bổ sung sẽ tạo ra một chu trình độ dài lẻ và phá vỡ
 tính hai phía của đồ thị.

Điều rắc rối chủ yếu ở đây là ta phải cẩn thận lưu ý tính chẵn lẻ khi hợp nhất hai tập
 trong hàm **union_sets ()**.

Khi bổ sung cạnh (**a, b**) nối 2 thành phần liên thông thành một, ta phải gắn một cây
 vào cây khác sao cho **a** và **b** có độ chẵn lẻ khác nhau.

Ta sẽ dẫn xuất công thức tính độ chẵn lẻ cho một đại diện khi gắn nó tới đại diện của tập kia.

Ký hiệu x là độ chẵn lẻ của đường đi từ a tới đại diện của nó và y – độ chẵn lẻ của đường đi từ b tới đại diện của mình. Gọi t là độ chẵn lẻ cần tìm cho đại diện được gắn vào tập mới. Nếu a được gắn vào hợp như một cây con thì sau khi hợp nhất độ chẵn lẻ của b không thay đổi và vẫn bằng y , còn độ chẵn lẻ của a sẽ là $x \oplus t$, trong đó \oplus là phép tính **XOR**. Điều ta cần là độ chẵn lẻ ở a và ở b phải khác nhau, tức là $x \oplus t \oplus y = 1$, từ đây suy ra $t = x \oplus y \oplus 1$.

Như vậy, không quan trọng việc cây nào sẽ được mang đi gắn như một cây con, công thức trên phải được sử dụng để tính độ chẵn lẻ từ một đại diện tới đại diện kia.

Trong chương trình, cũng giống như ở mục trên, với mỗi đỉnh cần lưu cặp giá trị trong **parent** []. Ngoài ra, cần có mảng **bipartite** [] đánh dấu tập có phải là đồ thị hai phía hay không.

```
void make_set (int v) {
    parent[v] = make_pair (v, 0);
    rank[v] = 0;
    bipartite[v] = true;
}

pair<int,int> find_set (int v) {
    if (v != parent[v].first) {
        int parity = parent[v].second;
        parent[v] = find_set (parent[v].first);
        parent[v].second ^= parity;
    }
    return parent[v];
}

void add_edge (int a, int b) {
    pair<int,int> pa = find_set (a);
    a = pa.first;
    int x = pa.second;

    pair<int,int> pb = find_set (b);
    b = pb.first;
    int y = pb.second;

    if (a == b) {
        if (x == y)
            bipartite[a] = false;
    }
    else {
        if (rank[a] < rank[b])
```

```

        swap (a, b);
        parent[b] = make_pair (a, x ^ y ^ 1);
        bipartite[a] &= bipartite[b];
        if (rank[a] == rank[b])
            ++rank[a];
    }
}

bool is_bipartite (int v) {
    return bipartite[ find_set(v) .first ];
}

```

Giải thuật tìm min trên một đoạn (*Range Minimum Query – RMQ*)

Xét bài toán tìm RMQ với độ phức tạp $O(\alpha(n))$ cho bài toán truy vấn offline. Bài toán có thể phát biểu một cách hình thức như sau: Phải tổ chức dữ liệu xử lý hai loại truy vấn: **insert(i)**, $i = 1, 2, \dots, n$ – bổ sung một số vào dãy, mỗi số chỉ bổ sung một lần và **extract_min()** cạnh nó.

Giả thiết dãy các truy vấn đã biết trước, tức là ta có bài toán offline.

Đường lối giải bài toán là như sau. Thay vì lần lượt trả lời các truy vấn theo trình tự đã cho ta xét $i = 1, 2, \dots, n$ và xác định nó là câu trả lời cho truy vấn nào. Để làm được điều đó ta phải tìm truy vấn đầu tiên không trả lời được đúng sau phép $\text{insert}(i)$ của số đó. Dễ dàng thấy rằng đó chính là truy vấn có câu trả lời là i .

Như vậy ta có sơ đồ xử lý giống như ở bài toán tô màu đoạn thẳng đã xét ở trên.

Độ phức tạp trung bình cho việc xử lý một truy vấn là $O(\log n)$ nếu không áp dụng giải pháp heuristic theo bậc và với mỗi phần tử chỉ lưu mốc nối phải tới kết quả của truy vấn **extract_min()** và có sử dụng kỹ thuật nén đường đi đối với các mốc nối này khi hợp nhất.

Độ phức tạp xử lý sẽ là $O(\alpha(n))$ nếu áp dụng giải pháp heuristic theo bậc và với mỗi tập – lưu số của vị trí kết thúc của tập (ở phương án giải trước thông tin này nhận được một cách tự động vì mốc nối chỉ luôn luôn hướng sang phải, còn ở phương án giả này – phải lưu trữ một cách tường minh).

Tìm cha chung nhỏ nhất trong cây (*Lowest Common Ancestor – LCA*)

Cho cây **G** với n đỉnh và m truy vấn dạng (a_i, b_i) . Với mỗi truy vấn cần tìm cha chung nhỏ nhất của 2 đỉnh a_i và b_i , tức là tìm đỉnh c_i xa gốc nhất và là cha chung của các đỉnh a_i và b_i .

Bài toán được xét trong chế độ offline, tức là đã biết trước mọi truy vấn. Tarjan đã đề xuất giải thuật trả lời tất cả các truy vấn với chi phí thời gian bậc $O(n+m)$, tức là $O(1)$ với mỗi truy vấn.

Giải thuật Tarjan

Sơ đồ xử lý

Bản chất của giải thuật là tiến hành loang theo chiều sâu, bắt đầu từ gốc của cây và lần lượt tìm ra câu trả lời cho các truy vấn. Kết quả truy vấn (v, u) được xác định khi loang tới đỉnh u và đã thăm đỉnh v hoặc ngược lại.

Trong quá trình loang theo chiều sâu sẽ có lúc ta tới đỉnh v và đã thăm các đỉnh con của nó. Giả thiết tồn tại truy vấn (v, u), trong đó u đã được thăm.

Khi đó $\text{LCA}(v, u)$ sẽ là chính đỉnh v hoặc một trong số các đỉnh cha của v . Như vậy phải tìm đỉnh cha thấp nhất của v (kể cả chính nó) nhận u là đỉnh con.

Với một v cố định, theo dấu hiệu “đỉnh cha thấp nhất của v và cũng là cha của một đỉnh nào đó khác” các đỉnh của cây bị phân rã thành các lớp không giao nhau. Với mỗi đỉnh cha $p \neq v$ sẽ có một lớp chứa p và các cây con có đỉnh là con của p đồng thời nằm “bên trái” trên đường tới v (tức là đã được thăm trước khi tới v).

Các lớp này có thể được quản lý một cách hiệu quả bằng cấu trúc hệ thống tập không giao nhau, mỗi lớp tương ứng với một tập với p là đỉnh đại diện và được lưu trong mảng **ancestor** [] .

Giả thiết trong khi loang theo chiều sâu ta tới đỉnh v . Đặt v vào một tập không giao nhau riêng, **ancestor** [v] = v . Việc loang theo chiều sâu đòi hỏi duyệt tất cả các cạnh (v, to) đi ra từ v . Với mỗi đỉnh to trong số trên đầu tiên ta phải loang theo chiều sâu từ nó, sau đó bằng phép **union_sets()** bổ sung đỉnh này cùng tất cả các đỉnh của cây con vào lớp chứa đỉnh v và cập nhật lại giá trị **ancestor** [] của đại diện thành v (vì trong quá trình hợp nhất đại diện có thể bị thay đổi). Sau khi xử lý xong các cạnh ta duyệt mọi truy vấn dạng (v, u), nếu u đã được thăm trong quá trình loang theo chiều sâu thì ghi nhận kết quả truy vấn $\text{LCA}(v, u) = \text{ancestor}[\text{find_set}(u)]$.

Không khó để nhận thấy là với mỗi truy vấn sự kiện một đỉnh là đang xét và đỉnh kia đã được thăm chỉ xảy ra một lần.

Đánh giá độ phức tạp

Độ phức tạp của việc loang theo chiều sâu là $O(n)$,

Độ phức tạp của việc hợp nhất các tập với mọi lần hợp nhất trong khi duyệt là $O(n)$,

Với mỗi truy vấn: 2 lần kiểm tra điều kiện xử lý với chi phí $O(1)$.

Tổng hợp, ta có độ phức tạp của giải thuật là $O(n+m)$, với m đủ lớn (tức là $n = O(m)$) ta có chi phí xử lý mỗi truy vấn là $O(1)$.

Chương trình

```
const int MAXN = максимальное число вершин в графе;
vector<int> g[MAXN], q[MAXN]; // lưu cây và truy vấn
int dsu[MAXN], ancestor[MAXN];
bool u[MAXN];

int dsu_get (int v) {
    return v == dsu[v] ? v : dsu[v] = dsu_get (dsu[v]);
}

void dsu_unite (int a, int b, int new_ancestor) {
    a = dsu_get (a), b = dsu_get (b);
    if (rand() & 1) swap (a, b);
    dsu[a] = b, ancestor[b] = new_ancestor;
}

void dfs (int v) {
    dsu[v] = v, ancestor[v] = v;
    u[v] = true;
    for (size_t i=0; i<g[v].size(); ++i)
        if (!u[g[v][i]]) {
            dfs (g[v][i]);
            dsu_unite (v, g[v][i], v);
        }
    for (size_t i=0; i<q[v].size(); ++i)
        if (u[q[v][i]]) {
            printf ("%d %d -> %d\n", v+1, q[v][i]+1,
                    ancestor[dsu_get (q[v][i]) ]+1);
        }
}

int main() {
    ... nhập dữ liệu cây ...

    // đọc và xử lý lưu các truy vấn
    for (;;) {
        int a, b = ...; // thông tin về một truy vấn
        --a, --b;
        q[a].push_back (b);
        q[b].push_back (a);
    }

    // loang theo chiều sâu và xử lý truy vấn
    dfs (0);
}
```

```
}
```

Hợp nhất các cấu trúc dữ liệu khác nhau

Một trong các phương pháp lưu trữ DSU là dùng *danh sách tương minh* lưu trữ các phần tử của mỗi tập. Mỗi phần tử được lưu trữ cùng với mốc nối tới đại diện của tập chứa nó.

Thoạt nhìn, có vẻ cấu trúc dữ liệu này kém hiệu quả: khi hợp nhất hai tập ta phải bổ sung một danh sách vào danh sách khác và cập nhật lại mốc nối tới đại diện ở các phần tử của một danh sách.

Tuy nhiên nếu áp dụng giải pháp heuristic theo bậc ta luôn ghép tập nhỏ hơn vào tập lớn với chi phí thời gian tỷ lệ kích thước tập nhỏ, còn việc tìm phần tử đại diện – luôn luôn có chi phí $O(1)$.

Dánh giá độ phức tạp: Giả thiết cần thực hiện m truy vấn. Có định một phần tử \mathbf{x} và xét các tác động của **union_set()** lên nó. Khi lần đầu tiên tác động lên nó, \mathbf{x} sẽ rơi vào tập mới có ít nhất 2 phần tử, khi tác động lần thứ 2, tập mới chứa nó có ít nhất 4 phần tử, . . . Như vậy tối đa số lần tác động lên \mathbf{x} là $\lceil \log n \rceil$. Như vậy với toàn bộ tập, số lần tác động là $O(n \log n)$. Với mỗi truy vấn ta cần $O(1)$ để xử lý.

Tổng cộng, độ phức tạp của giải thuật là $O(m+n \log n)$.

Các hàm xử lý:

```
vector<int> lst[MAXN];
int parent[MAXN];

void make_set (int v) {
    lst[v] = vector<int> (1, v);
    parent[v] = v;
}

int find_set (int v) {
    return parent[v];
}

void union_sets (int a, int b) {
    a = find_set (a);
    b = find_set (b);
    if (a != b) {
        if (lst[a].size() < lst[b].size())
            swap (a, b);
        while (!lst[b].empty()) {
            int v = lst[b].back();
```

```

        lst[b].pop_back();
        parent[v] = a;
        lst[a].push_back(v);
    }
}
}

```

Tư tưởng kết nối với heuristic theo bậc có thể triển khai hiệu quả ở các bài toán khác.

Ví dụ, xét bài toán: cho cây, trên mỗi lá của cây có ghi một số (có thể giống nhau ở các lá khác nhau). Yêu cầu xác định với mỗi đỉnh của cây số lượng các số khác nhau trong cây con của đỉnh.

Theo tư tưởng đã xét, sơ đồ xử lý sẽ bao gồm các bước sau. Tiến hành loang theo chiều sâu trên cây và trả về con trỏ tới tập **set** – danh sách các số ở cây con. Để dẫn xuất kết quả ở mỗi đỉnh (không phải là lá) cần loang theo chiều sâu của các cây con thuộc đỉnh đang xét, hợp nhất tất cả các **set** nhận được. Kích thước của tập kết quả là nghiệm cần tìm.

Độ phức tạp của giải thuật là $O(n \log^2 n)$ vì việc bổ sung phần tử vào set có độ phức tạp $O(\log n)$.

Tìm cầu của đồ thị trong chế độ online với độ phức tạp $O(\alpha(n))$

Một trong những mặt mạnh của DSU là cho phép *lưu cấu trúc của cây* ở cả dưới dạng *nén* và *không nén*.

Dạng nén dùng để hợp nhất nhanh 2 cây và kiểm tra hai đỉnh có thuộc một cây hay không, dạng không nén – để tìm đường đi giữa hai đỉnh hoặc các phép xử lý khác khi duyệt cây.

Như vậy ngoài mảng **parent**[] phục vụ thông tin nén có thể còn cần thêm mảng **real_parent**[] lưu thông tin không nén. Sự tồn tại của mảng thứ 2 không ảnh hưởng đến độ phức tạp của giải thuật vì sự thay đổi trong đó chỉ xảy ra ở phép hợp nhất cây và cũng chỉ thay đổi ở một phần tử.

Nhiều bài toán thực tế đòi hỏi kết nối 2 cây bằng cạnh cho trước không nhất thiết xuất phát từ gốc. Như vậy buộc phải kết nối cây vào một trong 2 đỉnh đã chỉ định bằng cách biến gốc của cây mang đi kết nối thành đỉnh con của đỉnh kia trong cạnh chỉ định kết nối.

Có vẻ như phép kết nối phức tạp và rất tốn thời gian vì khi gắn một cây vào đỉnh v ta phải cập nhật tất cả các mốc nối trong **parent**[] và **real_parent**[] khi đi từ **v** tới gốc của cây.

Trên thực tế, tình hình không đến nỗi tồi tệ như vậy vì ta sẽ gắn cây có bậc nhỏ hơn vào cây lớn và độ phức tạp trung bình của việc kết nối là $O(\log n)$.

Sơ lược về lịch sử DSU

DSU ra đời cách đây khá lâu. Cấu trúc dữ liệu này được dùng để mô ta rừng cây và được Galler, Fisher mô tả năm 1964 trong bài báo "*An Improved Equivalence Algorithm*". Tuy vậy việc phân tích, đánh giá độ phức tạp được thực hiện muộn hơn rất nhiều.

Giải pháp heuristic nén đường đi và hợp nhất theo bậc được đề xuất bởi McIlroy cùng Morris và độc lập với họ – Tritter.

Trong một thời gian dài độ phức tạp của DSU được biết đến như $O(\log^* n)$ do Hopcroft và Ullman đưa ra năm 1973. Đó là một hàm tăng chậm, nhưng vẫn còn nhanh hơn rất nhiều so với hàm Akkerman $\alpha(n)$.

Đánh giá $O(\alpha(n))$ được Tarjan đưa ra năm 1975 trong công trình "*Efficiency of a Good But Not Linear Set Union Algorithm*". Muộn hơn, vào năm 1985 Tarjan cùng với Leeuwen đã công bố một số kết quả đánh giá độ phức tạp của các giải pháp heuristic áp dụng với DSU trong bài báo "*Worst-Case Analysis of Set Union Algorithms*".

Cuối cùng, vào năm 1989 trong công trình "*The cell probe complexity of dynamic data structures*" Fredman và Saks đã chứng minh được rằng giải thuật bất kỳ dựa cơ sở trên DSU có độ phức tạp trung bình không vượt quá $O(\alpha(n))$.

Tuy nhiên, cũng có một số tác giả như Zhang trong "*The Union-Find Problem Is Linear*", Wu, Otoo trong "*A Simpler Proof of the Average Case Complexity of Union-Find with Path Compression*" không đồng tình với đánh giá trên và cho rằng DSU cá áp dụng các giải pháp heuristic làm việc với độ phức tạp $O(1)$.

Bài tập

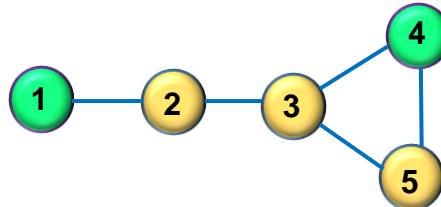
VV07. BỘ TRÍ DỮ LIỆU

Tên chương trình: DATA.CPP

Mạng lưới truyền thông của một hãng thông tin lớn có n servers, đánh số từ 1 đến n . Có m đường cáp nối trực tiếp 2 servers với nhau, đường cáp thứ i nối 2 servers a_i và b_i , $a_i \neq b_i$, $i = 1 \dots m$. Hệ thống đường cáp được bố trí đảm bảo từ một server có thể truyền thông tin tới server khác, trực tiếp hoặc qua các servers trung gian. Không có đường nối một server với chính nó.

Tập servers **A** được gọi là ổn định cao nếu trong trường hợp một đường cáp bị sự cố thì server x bất kỳ ngoài tập **A** vẫn có thể nhận thông tin từ một trong số các servers thuộc **A**.

Với mạng nối các servers như ở hình bên tập các servers (1, 2) là tập ổn định cao.



Các dữ liệu quan trọng được lưu lặp lại như nhau trong các servers thuộc tập **A**. Để giảm chi phí lưu và bảo trì người ta muốn có tập **A** càng nhỏ càng tốt. Ngoài ra, để đánh giá độ linh hoạt của toàn hệ thống người ta cũng muốn biết có tất cả bao nhiêu cách chọn tập **A** với kích thước tối thiểu. Hai cách chọn gọi là khác nhau nếu tồn tại ít nhất một server có trong các chọn thứ nhất và không có trong các chọn thứ 2.

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

- ✚ Dòng đầu tiên chứa 2 số nguyên n và m ($2 \leq n \leq 2 \times 10^5$, $1 \leq m \leq 2 \times 10^5$),
- ✚ Dòng thứ i trong m dòng sau chứa 2 số nguyên a_i và b_i ($1 \leq a_i, b_i \leq n$, $a_i \neq b_i$).

Kết quả: Đưa ra file văn bản DATA.OUT đưa ra trên một dòng 2 số nguyên – kích thước tối thiểu của tập A và số cách chọn theo mô đun $10^9 + 7$.

Ví dụ:

DATA.INP
5 5
1 2
2 3
3 4
3 5
4 5

DATA.OUT
2 3



VV07 Reg20170204 3

Giải thuật: Tìm cầu, chu trình và miền song liên thông của đồ thị.

Nhận xét:

Các servers nối với nhau tạo thành một đồ thị liên thông, trong đó đỉnh là các servers, Gọi B là tập các đỉnh tạo thành chu trình, nếu ta bỏ một cạnh nào đó trong chu trình thì giữa hai đỉnh bất kỳ thuộc B vẫn có đường đi, như vậy mọi đỉnh trong chu trình có kết nối nhau không phụ thuộc vào việc có đường truyền nào bị sự cố,

Giả thiết x và y là 2 đỉnh có đường nối trực tiếp, cạnh (x, y) được gọi là *cầu* của đồ thị nếu việc *loại bỏ cạnh này sẽ làm tăng số thành phần liên thông* của đồ thị,

Như vậy, sự tồn tại cầu sẽ tác động lên cách xây dựng tập A,

Tìm cầu là bài toán chuẩn trong lý thuyết đồ thị,

Ta có thể tìm tất cả các cạnh là cầu của đồ thị, loại bỏ chúng, đồ thị sẽ phân rã thành các miền liên thông riêng biệt,

Ở mỗi miền liên thông chọn một đỉnh bất kỳ làm đại diện và thay toàn bộ miền liên thông bằng đỉnh đại diện,

Khôi phục lại các cầu đã loại bỏ ta được một cây và các nút lá là những nút cần chọn để xây dựng A,

Như vậy số lượng nút lá là lực lượng của tập A và là một trong số các kết quả cần xác định,

Nếu nút lá là nút đại diện cho miền liên thông chứa p đỉnh, thì với nút lá đó ta có p các chọn đỉnh tham gia vào A,

Số cách xây dựng tập A sẽ là tích các khả năng chọn ở mỗi lá.

Tổ chức dữ liệu: sử dụng các mảng như trong sơ đồ lý thuyết chung xác định cầu của đồ thị.

Độ phức tạp của giải thuật: $O(n+m)$.

Chương trình:

```
#include <bits/stdc++.h>
#define NAME "data."
using namespace std;
typedef vector<vector<int>> vvi;
ifstream fi (NAME"inp");
ofstream fo (NAME"out");

const int64_t MOD = (int64_t)1e9 + 7;

void dfs(const vvi &graph,
         vector<bool> &used,
         set< pair<int, int>> &bridges,
         vector<int> &tin,
         vector<int> &fup,
         int &timer, int v, int p)
{
    used[v] = true;
    tin[v] = fup[v] = timer++;
    for (int u : graph[v])
    {
        if (u == p) continue;
        if (!used[u])
        {
            dfs(graph, used, bridges, tin, fup, timer, u, v);
            fup[v] = min(fup[v], fup[u]);
            if (fup[u] > tin[v]) bridges.insert({ min(v, u), max(v, u) });
        } else fup[v] = min(fup[v], tin[u]);
    }
}

void findComponent(const vvi &graph,
                   vector<bool> &used,
                   const set< pair<int, int>> &bridges,
                   vector<int> &getComponent,
                   int component, int &vs, int v)
{
    used[v] = true;
    getComponent[v] = component;
    vs++;
    for (int u : graph[v])
    {
        pair<int, int> p = { min(v, u), max(v, u) };
        if (bridges.find(p) != bridges.end()) continue;
        if (!used[u]) findComponent(graph, used, bridges, getComponent, component, vs, u);
    }
}

int main()
{
    int n, m;
    fi>>n>>m;
    vvi graph(n);
    for (int i = 0; i < m; i++)
    {
```

```

int v, u;
fi>>v>>u;
--v, --u;
graph[v].push_back(u);
graph[u].push_back(v);
}
set< pair<int, int>> bridges;
vector<bool> used(n, 0);
vector<int> tin(n, 0);
vector<int> fup(n, 0);
int timer = 0;
for (int i = 0; i < n; i++)
    if (!used[i]) dfs(graph, used, bridges, tin, fup, timer, i, -1);
fill(used.begin(), used.end(), false);
vector<int> componentSize(n, 0);
vector<int> getComponent(n, 0);
int component = 0;
for (int i = 0; i < n; i++)
{
    if (!used[i])
    {
        int vs = 0;
        findComponent(graph, used, bridges, getComponent, component, vs, i);
        componentSize[component++] = vs;
    }
}
vector<int> deg(component, 0);
for (const auto &bridge : bridges)
{
    deg[getComponent[bridge.first]]++;
    deg[getComponent[bridge.second]]++;
}
int ans = 0;
int cnt = 1;
for (int i = 0; i < component; i++)
    if (deg[i] <= 1)
    {
        ans++;
        cnt = (cnt * 1LL * componentSize[i]) % MOD;
    }
fo<<ans<< ' '<<cnt;
fo<<"\nTime: "<<clock() / (double)1000<<" sec";
return 0;
}

```



Cuộc thi “Tuổi trẻ và phát minh, sáng tạo” đã thu hút sự tham gia của đông đảo các bạn trẻ. Nhiều tư tưởng mới được đề xuất. Ban Giám khảo đặc biệt quan tâm tới đề xuất “Chế tạo máy tính với hệ cơ số 2 mới”.

Theo tư tưởng của tác giả, việc lưu trữ số nguyên dưới dạng nhị phân truyền thống đã làm lãng phí thông tin về độ dài số. Nếu đưa cả thông tin về độ dài số để biểu diễn thì có thể tiết kiệm được nhiều bộ nhớ hơn. Xét tất cả các xâu bít độ dài 1, 2, 3, ... và sắp xếp chúng theo độ dài, trong các xâu cùng độ dài – sắp xếp theo thứ tự từ điển. Xâu thứ i sẽ là dạng biểu diễn nhị phân mới của số i . Rõ ràng, xâu nhị phân này có độ dài ngắn hơn xâu bít hình thành từ dạng biểu diễn nhị phân truyền thống.

Cho số nguyên n . Hãy xác định dạng biểu diễn nhị phân mới của n .



0	0
1	1
2	00
3	01
4	10
5	11
6	000
...	...

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

- ✚ Dòng đầu tiên chứa một số nguyên t – số lượng tests ($1 \leq t \leq 1\,000$),
- ✚ Dòng thứ i trong n dòng sau chứa một số nguyên không âm không lớn hơn 10^{18} .

Kết quả: Đưa ra file văn bản NEWBIN.OUT các dạng biểu diễn mới nhận được, mỗi kết quả trên một dòng.

Ví dụ:

NEWBIN.INP
3
2
5
6

NEWBIN.OUT
00
11
000



Giải thuật: Xử lý bit.

Nhận xét:

Đánh số các nhóm theo độ dài của nhóm,

Với mỗi số nguyên không âm **n** cần xác định:

Độ dài xâu bít trong dạng biểu diễn mới,

Số thứ tự (tính từ 0) của **n** trong nhóm xâu bít có độ dài đã xác định.

Độ dài xâu bít của các nhóm là $2^1, 2^2, 2^3, \dots$

Điều kiện để **n** thuộc nhóm thứ **k** là

$$\sum_{i=1}^{k-1} 2^i < n \leq \sum_{i=1}^k 2^i$$

Số thứ tự của **n** trong nhóm sẽ là $n - \sum_{i=1}^{k-1} 2^i$.

Nguyên tắc tìm **k** và số thứ tự của **n** trong nhóm: tìm lùi, giảm dần giá trị của **n**, nếu **n** không thuộc nhóm **i** thì thay **n** bằng $n - 2^i$ và tăng **i**.

Độ phức tạp của giải thuật: O(log₂n) với mỗi số **n**.

Chương trình:

```
#include <bits/stdc++.h>
#define NAME "newbin."
using namespace std;
ifstream fi (NAME"inp");
ofstream fo (NAME"out");
const int64_t one=1;
int t,k,len;
int64_t n,m;

int main()
{
    fi>>t;
    for(int i=0;i<t;i++)
    {
        fi>>n; k=0;m=2;
        while(n>=m) ++k,n-=m,m<<=1;
        for(int j=k;j>=0;--j) fo<<( (n>>j)&1); fo<<'\
    }
    fo<<"\nTime: "<<clock() / (double)1000<<" sec";
}
```



Để sinh viên ra trường có kiến thức chuyên môn cũng như tay nghề phù hợp với khả năng của mình và yêu cầu của xã hội việc học theo tín chỉ được quy định như sau: có n chuyên ngành, mỗi ngành có một số lượng giáo trình chuyên đề khác nhau nhất định. Để tốt nghiệp mỗi sinh viên nhận được một tín chỉ (học và thi qua một giáo trình) ở mỗi ngành. Một cách nhận n tín chỉ để tốt nghiệp được gọi là một dạng kỹ sư. Hai dạng kỹ sư gọi là khác nhau nếu không trùng khớp tất cả các tín chỉ nhận được. Nhà trường muốn có số lượng dạng kỹ sư đào tạo không ít hơn k .

Hãy xác định tổng số lượng giáo trình khác nhau cần chuẩn bị để đáp ứng yêu cầu của nhà trường.

Dữ liệu: Vào từ file văn bản CERTI.INP gồm một dòng chứa 2 số nguyên n và k ($1 \leq n \leq 10^5$, $0 \leq k \leq 10^{18}$).

Kết quả: Đưa ra file văn bản CERTI.OUT một số nguyên – số giáo trình ít nhất cần chuẩn bị.

Ví dụ:

CERTI.INP
2 89

CERTI.OUT
19



VU36 Mos 20161211 3

Giải thuật: Xử lý số học.

Nhận xét:

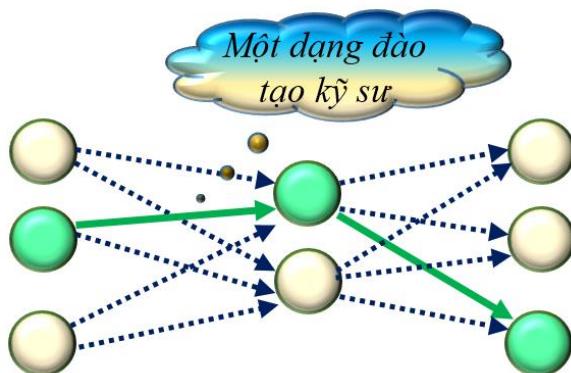
Gọi số lượng giáo trình ở chuyên ngành i là a_i , $i = 1 \dots n$,

Mỗi loại kỹ sư đào tạo tương ứng với một đường đi từ chuyên ngành 1 đến chuyên ngành n ,

Ta có tổng số lượng loại kỹ sư đào tạo là $a_1 \times a_2 \times \dots \times a_n$,

Ví dụ, với $n = 3$, $a_1 = 3$, $a_2 = 2$, $a_3 = 3$ ta có tổng số giáo trình là 8 và số lượng dạng kỹ sư đào tạo khác nhau là 18.

Như vậy ta có bài toán tối ưu: Xác định n số a_1, a_2, \dots, a_n có tổng nhỏ nhất và tích không nhỏ hơn k .



$$\begin{cases} \sum_{i=1}^n a_i \rightarrow \min \\ \prod_{i=1}^n a_i \geq k \\ a_i - \text{nguyên dương.} \end{cases}$$

Gọi $amin = \min\{a_1, a_2, \dots, a_n\}$, $amax = \max\{a_1, a_2, \dots, a_n\}$. Lời giải bài toán trên có tính chất $amax - amin \leq 1$.

Thật vậy, giả thiết $amax - amin > 1$. Loại bỏ một phần tử bằng $amax$ và một phần tử bằng $amin$, gọi P là tích các phần tử a_i còn lại, $P = 1$ nếu không còn phần tử nào ($n = 2$).

Trong dãy số a_1, a_2, \dots, a_n thay phần tử $amin$ bị loại bằng $amin+1$, thay $amax$ bị loại bằng $amax-1$ ta có dãy số mới với tổng không đổi.

Xét tích các số của dãy số mới:

$$\begin{aligned} \text{Product} &= P \times (amin+1) \times (amax-1) \\ &= P \times (amin \times amax + amax - amin - 1) \\ &= \underbrace{P \times amin \times amax}_{\text{Tích dãy số ban đầu}} + \underbrace{P \times (amax - amin - 1)}_{\text>Lớn hơn 0} \end{aligned}$$

Do đó phải có $amax - amin \leq 1$.

Dãy số cần tìm chỉ chứa 2 loại số: m ($0 < m \leq n$) số $ax = amax$ và $(n-m)$ số $bm = amin = amax-1$.

Xử lý:

Xử lý các trường hợp riêng:

- $k = 0$: $a_i = 0 \forall i$, kết quả $ans = 0$,
- $k = 1$: $a_i = 1 \forall i$, kết quả $ans = n$,
- $n = 1$: $ans = k$,

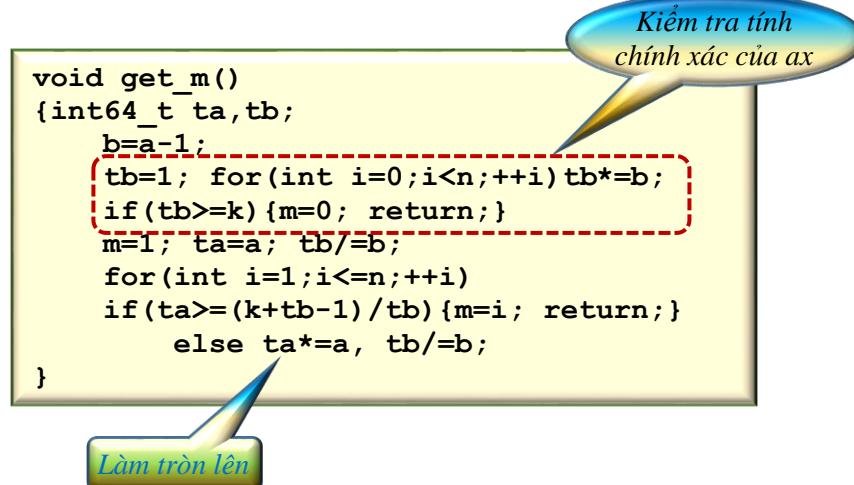
Xác định ax :

- Với $n \geq 60$: $ax = 2^{60} > 10^{18}$,
- Trường hợp $n < 60$: $ax^n \geq k \Rightarrow ax \geq 2^{n \times \log_2(k)}$, về phái bất đẳng thức là một số thực vì vậy cần làm tròn lên đến số nguyên gần nhất.

Xác định m :

Vì ax được xác định qua các đại lượng trung gian thực nên cần kiểm tra lại tính chính xác của ax , ví dụ biểu thức $2^{n \times \log_2(k)}$ có kết quả đúng là 5, nhưng kết quả tính toán có thể là 4.99999 hay 5.000...001 (sai số ở chữ số 19, 20 sau dấu chấm thập phân), ở trường hợp sau việc làm tròn lên sẽ cho kết quả 6,

Điều kiện $ax^m \times bm^{n-m} \geq k$ phải kiểm tra dưới dạng $axm \geq k/bm^{n-m}$ để chống khả năng tràn ô khi thực hiện phép nhân, kết quả về phái bất đẳng thức cần được làm tròn lên:



Độ phức tạp của giải thuật: $\approx O(1)$.

Chương trình:

```
#include <bits/stdc++.h>
#define NAME "certi."
using namespace std;
ifstream fi (NAME"inp");
ofstream fo (NAME"out");
int64_t n,k,m,ans,a,b;

void get_a()
{double x,y;
int64_t t;
if(n>=60) {a=2; return;}
x=exp2(log2((double)k)/n);
a=(int64_t)ceil(x);
return;
}

void get_m()
{int64_t ta,tb;
b=a-1;
tb=1; for(int i=0;i<n;++i) tb*=b;
if(tb>=k) {m=0; return;}
m=1; ta=a; tb/=b;
for(int i=1;i<=n;++i)
if(ta>=(k+tb-1)/tb) {m=i; return;}
else ta*=a, tb/=b;
}

int main()
{
    fi>>n>>k;
    if(k==0) {ans=0; fo<<ans; return 0;}
    if(k==1) {ans=n; fo<<ans; return 0;}
    if(n==1) {ans=k; fo<<ans; return 0;}
    get_a(); get_m();
    ans=a*m+(n-m)*b;
    fo<<ans;
    fo<<"\nTime: "<<clock()/(double)1000<<" sec";
}
```



Bến xe buýt nhanh BRT đến trờng nằm ở bên kia đường tàu hỏa, ngay cạnh rào chắn, còn ký túc xá của Steve ở bên này đường tàu và khoảng cách từ ký túc xá tới đường tàu là s .

Các phương tiện lưu thông trên tuyén cát ngang đường tàu phải tuân thủ các quy định an toàn sau:

- Trong phạm vi cách đường tàu không quá s tốc độ tối đa của mọi phương tiện không được vượt quá $v1$,
- Khi có tín hiệu đóng đường các phương tiện gia thông cách đường tàu không quá s đều phải dừng lại và chỉ được đi tiếp khi có tín hiệu mở đường.

Cụ thể hơn nữa, nếu tín hiệu đóng đường phát tại thời điểm 1 và tín hiệu mở đường – tại thời điểm x các xe vẫn được chạy cho đến hết thời điểm 1 và chỉ được chạy lại khi bắt đầu thời điểm $x+1$.

Ngay cạnh ký túc xá là bến đỗ xe buýt và điểm dừng tiếp theo cũng ở bên kia đường tàu, trùng với bến đỗ của xe BRT. Dĩ nhiên các lái xe luôn đi với tốc độ tối đa cho phép là $v1$. Thời gian để đạt tốc độ $v1$ khi đi tiếp sau mỗi lần dừng là không đáng kể.

Cạnh đường tàu có đường vượt ngầm dành cho người đi bộ. Thời gian để xuống và lên đường ngầm cũng không đáng kể. Như vậy việc chấn đường không ảnh hưởng tới người đi bộ.

Steve có thể đi bộ đến bến xe BRT với tốc độ $v2$.

Hàng ngày có n lần đường bị chấn, lần thứ i là từ l_i đến r_i , $i = 1 \dots n$. Steve muốn đến trờng nhanh nhất có thể và vì vậy cần tính toán lựa chọn đi xe buýt hay đi bộ. Nếu đã lên xe buýt thì không được xuống giữa đường, cũng như nếu đã đi bộ thì không thể lên xe buýt giữa đường. Luật an toàn giao thông không cho phép. Nếu thời gian đi bộ ít hơn thời gian dùng ô tô buýt Steve sẽ chọn phương án đi bộ, trong trường hợp ngược lại – đi xe buýt. Tại thời điểm 0 có ô tô buýt tại điểm đỗ.

Hãy xác định phương án Steve cần lựa chọn, đưa ra thông báo **WALK** (đi bộ) hoặc **BUS** (dùng xe buýt) và thời đi theo cách đã chọn.

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

- ⊕ Dòng đầu tiên chứa ba số nguyên s , $v1$ và $v2$ ($1 \leq s, v1, v2 \leq 10^9$),
- ⊕ Dòng thứ 2 chứa số nguyên n ($0 \leq n \leq 10^5$),
- ⊕ Dòng thứ i trong n dòng sau chứa 2 số nguyên l_i và r_i ($0 \leq l_i \leq r_i \leq 10^9$), các khoảng không giao nhau với độ dài khác 0, tuy nhiên có thể điểm cuối của một khoảng trùng với điểm đầu của đoạn khác.

Kết quả: Đưa ra file văn bản CHOICE.OUT thông báo lựa chọn trên dòng thứ nhất và một số nguyên – thời gian đi trên dòng thứ hai.

Ví dụ:

CHOICE.INP
20 5 1
3
0 10
12 14
40 100

CHOICE.OUT
BUS
16



VU37 Mos 20161211 4

Giải thuật: Xử lý sự kiện theo thời gian.

Nhận xét:

Với các bài toán xử lý thời gian điều quan trọng cần lưu ý là sự kiện xảy ra lúc nào: ở **đầu** một thời điểm hay **cuối thời điểm** được nêu,

Lưu ý: *Thời gian = Quãng đường / Tốc độ* và phải *làm tròn lên* nếu kết quả không nguyên,

Ở bài toán này thời gian đi bộ **tmb** không phụ thuộc vào lịch đóng đường,

Thời gian **tmb** ô tô buýt chạy: là thời gian đi hết quãng đường **s** như không bị tàu chặng cộng với tổng thời gian dừng chờ tàu,

Cần sắp xếp các khoảng thời gian đóng đường theo thời điểm đầu,

Để thuận tiện xử lý: cần bổ sung vào cuối dãy đã sắp xếp **phản tử hàng rào** với điểm đầu là vô cực.

Tổ chức dữ liệu:

- Mảng **vector<pii> sdl** – lưu các khoảng thời gian đóng bị đóng, mỗi phần tử của mảng là một cặp dữ liệu (**l, r**),
- Biến **tmb** – quản lý thời gian đi bộ,
- Biến **tmb** – quản lý thời điểm ô tô buýt tới đích,
- Biến **tr** – quản lý thời gian còn lại ô tô buýt cần chạy.

Xử lý:

Nhập **s**, **v1**, **v2** và xác định các tham số điều khiển thời gian,

Nhập thông tin về các khoảng thời gian đóng đường, sắp xếp và bổ sung phản tử hàng rào,

Cập nhật **tmb** cho đến khi **tr** bằng 0:

```

for(int i=0;i<=n;++i)
{
    a=sdl[i].first; b=sdl[i].second;
    if(tmb<=a)
    {
        r=a-tmb+1;
        if(r>=tr) {tmb+=tr;break;}
        else tr=r, tmb=b+1;
    } else tmb=b+1;
}

```

So sánh các thời gian và đưa ra kết quả tương ứng.

Độ phức tạp của giải thuật: $O(n \ln n)$.

Chương trình:

```
#include <bits/stdc++.h>
#define NAME "choice."
using namespace std;
ifstream fi (NAME"inp");
ofstream fo (NAME"out");
typedef pair<int,int> pii;
int n,s,v1,v2,a,b;
int64_t tmw,tmb=0,tr,r;
vector<pii> sdl;

int main()
{
    fi>>s>>v1>>v2>>n;
    tmw=(s+v2-1)/v2;
    tr=(s+v1-1)/v1;
    sdl.reserve(n+1);
    for(int i=0;i<n;++i)
    {
        fi>>a>>b; sdl[i]={a,b};
    }
    sdl[n]={sdl[n-1].first+tr+1,b};
    sort(sdl.begin(),sdl.end());
    for(int i=0;i<=n;++i)
    {
        a=sdl[i].first; b=sdl[i].second;
        if(tmb<=a)
        {
            r=a-tmb+1;
            if(r>=tr) {tmb+=tr; break;}
            else tr-=r, tmb=b+1;
        } else tmb=b+1;
    }
    if(tmw<tmb) fo<<"WALK\n" <<tmw; else fo<<"BUS\n" <<tmb;
    fo<<"\nTime: "<<clock() / (double)1000<<" sec";
}
```



VU38. TỔNG LỚN NHẤT

Tên chương trình: MAXSUM.CPP

Cho dãy số nguyên a_1, a_2, \dots, a_n . Trong trường hợp cần thiết được phép đổi chỗ 2 số a_i và a_j , $i \neq j$, $1 \leq i, j \leq n$.

Tổng s được xác định theo công thức $s = a_1 - a_2 + a_3 - a_4 + \dots + (-1)^{n-1} a_n$.

Hãy xác định tổng lớn nhất có thể đạt được.

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

- ✚ Dòng đầu tiên chứa một số nguyên n ($2 \leq n \leq 10^5$),
- ✚ Dòng thứ 2 chứa n số nguyên a_1, a_2, \dots, a_n ($1 \leq a_i \leq 1\,000$, $i = 1 \div n$).

Kết quả: Đưa ra file văn bản MAXSUM.OUT một số nguyên – tổng lớn nhất tìm được.

Ví dụ:

MAXSUM.INP
2
1 2

MAXSUM.OUT
1



VU38 Io20170211 A

Giải thuật: Tìm max – min.

Nhận xét:

Việc đổi chỗ 2 số ở cùng vị trí lẻ hay cùng vị trí chẵn không làm thay đổi kết quả,

Gọi s_0 là tổng các số ở vị trí chẵn (*vị trí được đánh số từ 0*), s_1 – tổng các số ở vị trí lẻ, ta có $s = s_0 - s_1$,

Nếu đổi chỗ a_i và a_j , trong đó i chẵn, j – lẻ, giá trị mới của s sẽ là

$$\begin{aligned}s &= s_0 - a_i + a_j - (s_1 - a_j + a_i) \\&= s_0 - s_1 + 2 \times (a_j - a_i)\end{aligned}$$

Giá trị s tăng khi $a_j > a_i$ và đạt cực đại khi a_j là cực đại trong số các phần tử ở vị trí lẻ và a_i – cực tiểu trong số các phần tử ở vị trí chẵn.

Xử lý:

Không cần lưu trữ các a_i ,

Khi nhập dữ liệu: đồng thời tính s_0 , s_1 , tìm giá trị min của các phần tử ở vị trí chẵn và max của các phần tử ở vị trí lẻ.

Độ phức tạp của giải thuật: $O(n)$.

Chương trình:

```
#include <bits/stdc++.h>
#define NAME "maxsum."
using namespace std;
ifstream fi (NAME"inp");
ofstream fo (NAME"out");
int n,s0=0,s1=0,x=10000,y=0,t,ans;
int main()
{
    fi>>n;
    for(int i=0;i<n;++i)
    {
        fi>>t;
        if(i&1)
        {
            s1+=t; if(t>y) y=t;
        }
        else
        {
            s0+=t; if(t<x) x=t;
        }
    }
    ans=s0-s1; if(x<y) ans+=2*(y-x);
    fo<<ans;
    fo<<"\nTime: "<<clock() / (double)1000<<" sec";
}
```



Có n khu công nghiệp được xây dựng. Để tạo thành một tổ hợp công nghiệp hoạt động có hiệu quả cần có đường nối trực tiếp các khu công nghiệp với nhau. Kinh phí còn lại chỉ cho phép xây dựng m đường, mỗi đường nối trực tiếp 2 khu công nghiệp và cho phép chuyển động hai chiều. Với phong cách quan liêu mệnh lệnh, một chỉ thị cung nhắc đã được đưa ra: “Phải xây dựng đúng m đường và không được xây dựng đường nối một khu công nghiệp với chính nó”!

Bộ phận Thiết kế - Quy hoạch phải lên phương án xây dựng đường. Nội dung chỉ thị không cấm xây dựng nhiều đường giữa một cặp 2 khu công nghiệp. Địa hình cụ thể của khu công nghiệp i cho thấy không thể xây dựng quá a_i đường từ nó.

Từ nhiệt huyết chuyên môn người ta cố gắng thiết kế sao cho tồn tại một nhóm nhân gồm nhiều các khu công nghiệp nhất được nối với nhau mà 2 khu công nghiệp bất kỳ trong nhân có đường nối trực tiếp.

Hãy xác định số lượng tối đa các khu công nghiệp tạo thành nhóm nhân. Nếu không thể thiết kế được mạng m đường thì đưa ra số -1.

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

- ✚ Dòng đầu tiên chứa một số nguyên n và m ($1 \leq n \leq 10^5$, $0 \leq m \leq 10^5$),
- ✚ Dòng thứ 2 chứa n số nguyên a_1, a_2, \dots, a_n ($0 \leq a_i \leq 10^5$, $i = 1 \div n$).

Kết quả: Đưa ra file văn bản TRANSNET.OUT một số nguyên – số -1 hoặc số lượng tối đa các khu công nghiệp trong nhóm nhân.

Ví dụ:

TRANSNET.INP
4 3
1 2 3 4

TRANSNET.OUT
3



Giải thuật: Số đường đi trên đồ thị đầy đủ.

Nhận xét:

Xét đồ thị bắt đầu từ các đỉnh (Khu công nghiệp) có khả năng xây dựng nhiều đường đi nhất,

Sắp xếp các đỉnh theo số khả năng giảm dần,

Gọi tổng số khả năng ở tất cả các đỉnh là s ,

Khi mở một đường thì sẽ có khả năng còn lại ở 2 đỉnh giảm 1, vì vậy số lượng đường tối đa không vượt quá $s/2$,

Nếu tồn tại phương án làm đường thì một trong số các phương án đó phải chứa đỉnh 0 với số đường tiềm năng a_0 , số đường sẽ không vượt quá tổng số tiềm năng các đỉnh còn lại,

Vì vậy, điều kiện để có thể xây dựng mạng lưới m đường là

$$\min\{s/2, s-a_0\} \geq m$$

Duyệt mọi trường hợp các khả năng tổ chức nhân với số lượng nút $j = 1, 2, \dots, n$,

Để có đồ thị đầy đủ (*giữa 2 nút bắt kỳ có đường nối*) với số lượng nút là j cần sử dụng j nút đầu tiên (từ 0 đến $j-1$) và cần có $rq=j \times (j-1)$ đường,

Nếu $rq > m$ hoặc từ nút $(j-1)$ không có đủ $(j-1)$ khả năng mở đường thì không thể có nhân chứa j nút.

Nếu có thể tổ chức đồ thị đầy đủ với j nút thì kiểm tra có còn khả năng mở thêm không ít hơn $m-rq$ đường hay không, nếu không thể thì dừng việc kiểm tra và đưa ra kết quả kích thước nhân đã nhận được trước đó.

Tổ chức dữ liệu: Chỉ cần mảng `int a[N]` chứa dữ liệu ban đầu.

Xử lý:

Các cách sắp xếp mảng theo thứ tự giảm dần:

- ▀ `sort(a, a+n, greater<int>());`
- ▀ `sort(a, a+n, [] (const int x, const int y) {return x>y;});`
- ▀ `sort(a, a+n); reverse(a, a+n);`

Điều kiện không thể làm đủ m đường:

```
if (m > 0 && n == 1) {fo<<"-1"; return 0;}  
if (min(s/2, s-a[0]) < m) {fo<<"-1"; return 0;}
```

Xác định kích thước nhẫn:

```
for (int j = 1; j <= n; ++j)
{
    req = j*(int64_t)(j-1)/2;
    if (req > m) break;
    if (a[j-1] < j-1) break;
    int mx = 0;
    for (int i=0;i<j;++i) mx=max(mx,a[i]-(j-1));
    for (int i=j; i<n;++i) mx = max(mx,a[i]);
    rest = s-req*2;
    mxcan = min(rest/2, rest-mx);
    if (req+mxcan >= m) ans = j;
}
```

Số đường trong nhẫn

Kiểm tra
điều kiện cần

Số khả năng còn lại

Độ phức tạp của giải thuật: $O(n \ln n)$.

Chương trình:

```
#include <bits/stdc++.h>
#define NAME "transnet."
using namespace std;
ifstream fi (NAME"inp");
ofstream fo (NAME"out");
const int N = 1e5 + 10;
int64_t s=0,req,rest,mxcan;
int n,m,a[N],ans=1;

int main()
{
    fi>>n>>m;
    for(int i=0;i<n;++i) {fi>>a[i]; s+=a[i];}
    if (m > 0 && n == 1) {fo<<"-1"; return 0;}
    sort(a, a + n);
    reverse(a, a + n);
    if (min(s/2, s-a[0]) < m) {fo<<"-1"; return 0;}
    for (int j = 1; j <= n; ++j)
    {
        req = j*(int64_t)(j-1)/2;
        if (req > m) break;
        if (a[j-1]< j-1) break;
        int mx = 0;
        for (int i = 0; i<j; ++i) mx = max(mx, a[i]-(j-1));
        for (int i=j; i<n; ++i) mx = max(mx, a[i]);
        rest = s-req*2;
        mxcan = min(rest/2, rest-mx);
        if (req+mxcan >= m) ans = j;
    }
    fo<<ans;
    fo<<"\nTime: "<<clock() / (double)1000<<" sec";
    return 0;
}
```



Địa bàn do một trinh sát phòng chống ma tuy phụ trách có n giao lộ và $n-1$ đường nối các giao lộ này. Giữa 2 giao lộ bất kỳ có một đường đi duy nhất (nối trực tiếp hoặc qua các giao lộ khác). Người trinh sát muốn có đầy đủ thông tin về các điểm tập kết hàng cũng như mạng lưới phân phối, tiêu thu ma túy. Để làm được việc đó anh cần theo dõi sát sao các hoạt động của tên trùm, bí mật quay mọi hoạt động của hắn.

Từ nguồn thông tin mật anh biết chiều tối nay tên trùm sẽ đi làm một vụ giao dịch lớn. Có m khả năng về đường đi của hắn: có thể hắn sẽ xuất phát từ giao lộ a_i và tới giao lộ b_i , $i = 1 \dots n$. Người trinh sát mật phục ở giao lộ x , đi ra như một người bình thường. Nếu từ x anh có thể đi qua hết các giao lộ trên đường từ a_i tới b_i và không có đoạn đường nào phải 2 lần thì có nghĩa là kiểm soát được lô trình (a_i, b_i).

Hãy xác định với việc lựa chọn x thích hợp anh có thể kiểm soát nhiều nhất bao nhiêu lô trình.

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

- ⊕ Dòng đầu tiên chứa một số nguyên n ($2 \leq n \leq 2 \times 10^5$),
- ⊕ Dòng thứ j trong $n-1$ dòng sau chứa 2 số nguyên x_j, y_j xác định 2 giao lộ có đường nối trực tiếp với nhau ($1 \leq x_j, y_j \leq n, x_j \neq y_j$),
- ⊕ Dòng tiếp theo chứa số nguyên m ($1 \leq m \leq 2 \times 10^5$),
- ⊕ Dòng thứ i trong m dòng sau chứa 2 số nguyên a_i, b_i xác định một lô trình có thể cần kiểm soát ($1 \leq a_i, b_i \leq n, a_i \neq b_i$).

Kết quả: Đưa ra file văn bản AMBUSH.OUT một số nguyên – số lô trình tối đa có thể kiểm soát.

Ví dụ:

AMBUSH.INP
7
1 2
2 3
3 4
3 5
4 6
4 7
3
1 4
2 5
6 7

AMBUSH.OUT
2

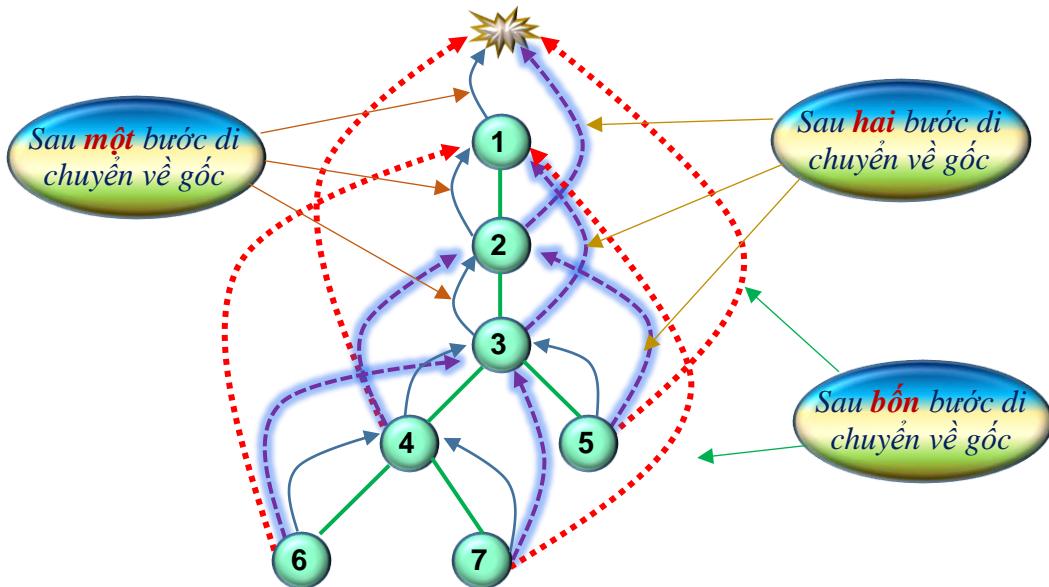


Giải thuật 1: Đồ thị cây, Quy hoạch động, Tìm kiếm nhị phân.

Nhận xét:

Chọn một nút làm gốc, nếu không có gì đặc biệt – là nút 1,

Ta phải xác định từ một nút **x** có thể tới bao nhiêu điểm đầu hoặc cuối của một số lô trình, trong đó đường đi tới không chứa một phần (hoặc toàn bộ) lô trình.



Độ dài các lô trình không biết trước, việc di chuyển lần lượt từ một nút sang nút tiếp theo đòi hỏi phải xử lý với độ phức tạp $O(n^2)$,

Để giảm thời gian duyệt ta dùng quy hoạch động lập bảng kết quả di chuyển sau 1 bước, sau 2 bước, sau 4 bước, . . . đối với mọi nút của đồ thị. Số lần xét sẽ không vượt quá 18 (vì $n \leq 2 \times 10^5$),

Đồng thời với việc tính bảng di chuyển ta có thể xác định độ sâu của mỗi nút,

Với mỗi lô trình cần xác định **đường đi từ gốc** tới các nút đầu/cuối có **chứa toàn bộ lô trình** hay không, nếu **chứa hết** thì **khả năng kiểm soát lô trình của điểm gốc tăng thêm 1**, nếu **không chứa hết** – có thể kiểm soát lô trình từ **điểm đầu** hoặc **điểm cuối**,

Để thuận tiện xử lý: chuẩn hóa dữ liệu, đảm bảo độ sâu của a_i lớn hơn hoặc bằng độ sâu của b_i ,

Khi đó nếu đường đi từ gốc tới a_i chứa b_i thì ta có trường hợp chứa toàn bộ lô trình, trong trường hợp ngược lại – không chứa hết!

Để kiểm tra khả năng đường đi từ gốc tới a_i chứa b_i hay không – dựa vào bảng quy hoạch động đã nêu, dùng phương pháp **Trượt xuống nhanh nhất** (một dạng của tìm kiếm nhị phân),

Vấn đề còn lại chỉ là tìm \max tổng khả năng kiểm soát ở gốc và ở các nhánh mà gốc không kiểm soát được.

Lưu ý:

- ⊕ Với mỗi lô trình, cần đánh dấu khả năng kiểm soát các nút đầu và cuối (a_i và b_i),
- ⊕ Nếu gốc có thể kiểm soát lô trình thì thay việc đánh dấu ở b_i bằng việc đánh dấu ở nút gốc,
- ⊕ Khi tổng hợp kết quả, lô trình được kiểm soát bởi nút gốc bị nhận dạng thêm một lần nữa ở nút ai, nhưng ta không thể xóa việc đánh dấu kiểm soát ở a_i (phục vụ cho việc thống kê ở các nút sau a_i), vì vậy ở một nút nào đó trên lô trình (khác a_i) cần giảm bớt số khả năng kiểm soát 1.

Tổ chức dữ liệu:

- ⊖ Mảng **vector <int> e[N]** – lưu thông tin về quan hệ kè của các nút,
- ⊖ Mảng **int dp[D][N]** – lưu kết quả chuyển động từ mỗi nút về gốc,
- ⊖ Mảng **int dep[N]** – lưu độ sâu của các nút,
- ⊖ Mảng **int add[N]** – lưu khả năng kiểm soát lô trình.

Xử lý:

Ghi nhận thông tin về đồ thị:

```
fi>>n;
for (int i = 0; i < n - 1; ++i)
{
    int x, y;
    fi>>x>>y;
    --x; --y;
    e[x].push_back(y);
    e[y].push_back(x);
}
```

Tính bảng di chuyển, tính độ sâu các đỉnh và rút gọn ghi nhận quan hệ kè:

- Dùng phương pháp loang theo chiều sâu,
- Đỉnh đồ thị được đánh số từ 0,
- Đỉnh ngoài đồ thị: đánh số -1,
- Xác định bảng $dp_{i,v}$ theo cột,
- Để đơn giản hóa việc lập trình: luôn luôn tính với $i = 0 \div 17$,
- Kết hợp xử lý đỉnh với việc loại bỏ thông tin lặp về cạnh.
- Xuất phát từ đỉnh 0, với kết quả tới -1 và độ sâu 0: $dfs(0, -1, 0)$;

```

void dfs(int v, int p, int cdep)
{
    dep[v] = cdep;
    dp[0][v] = p;
    for (int i = 1; i < D; ++i)
        dp[i][v]=(dp[i-1][v]==-1 ? -1:dp[i-1][dp[i-1][v]]);
    for (int i = 0; i < e[v].size(); ++i)
    {
        int nv = e[v][i];
        if (nv == p)
        {
            e[v][i] = e[v].back();
            e[v].pop_back();
            --i;
            continue;
        }
        dfs(nv, v, cdep + 1);
    }
}

```

Ghi nhận chiều sâu

C++ cho phép

Ghi nhận đỉnh đến

Xóa quan hệ lặp đã xử lý

Tính cột mới với đỉnh mới

Xử lý truy vấn:

- Chuẩn hóa dữ liệu: đưa về trường hợp điểm cuối lộ trình có độ sâu nhỏ hơn hoặc bằng độ sâu điểm đầu,
- Ghi nhận khả năng kiểm soát của điểm đầu,
- Ghi nhận khả năng kiểm soát của điểm cuối của lộ trình (**x, y**):

```

int x1 = goUp(x, dep[x]-dep[y]-1);
if (dp[0][x1] != y)
    add[y] += 1;
else
{
    add[0] += 1;
    add[x1] -= 1;
}

```

Tìm đỉnh trên lộ trình có cùng độ sâu với y

Điều kiện gốc không kiểm soát được lộ trình

Chống thông kê lặp

Để kiểm tra từ gốc có thẻ kiểm soát được lô trình (**x**, **y**) hay không cần tìm nút trên lô trình có độ sâu nhỏ hơn 1 so với **dep_y**, đây là nút tiềm năng đứng sau **y** trên đường về gốc. Nếu quả thật nó đứng sau **y** thì nút gốc kiểm soát được lô trình!

Tìm nút tiềm năng đứng sau **y**: **x1 = goUp(x, dep[x]-dep[y]-1);**

Việc tìm kiếm được thực hiện bằng phương pháp *Trượt xuống nhanh nhất*:

```
int goUp(int x, int p)
{
    for (int i = D - 1; i >= 0; --i)
        if (p >= (1 << i))
    {
        x = dp[i][x];
        p -= (1 << i);
    }
    return x;
}
```

Bước chuyển theo lũy thừa của 2, từ lớn đến bé

Ghi nhận kết quả: Cây *đã được rút gọn*, lược bỏ quan hệ lặp. Các lô trình mà nút gốc quả lý được đều đã ghi nhận ở khả năng quản lý ở gốc, vì vậy khi loang từ gốc ta có thể tích lũy tổng khả năng kiểm soát các lô trình không do gốc quản lý và từ đó – tìm *max*:

```
void dfs1(int v, int c)
{
    ans = max(ans, c += add[v]);
    for (int nv : e[v])
        dfs1(nv, c);
}
```

C++ 11

Độ phức tạp của giải thuật: O($n \ln n$).

Chương trình:

```
#include <bits/stdc++.h>
#define NAME "ambush."
using namespace std;
ifstream fi (NAME"inp");
ofstream fo (NAME"out");
const int D = 18;
const int N = 200179;

int n;
vector < int > e[N];
int dp[D][N];
int dep[N];
int add[N];

void dfs(int v, int p, int cdep)
{
    dep[v] = cdep;
    dp[0][v] = p;
    for (int i = 1; i < D; ++i)
        dp[i][v] = (dp[i-1][v] == -1 ? -1 : dp[i-1][dp[i-1][v]]);
    for (int i = 0; i < e[v].size(); ++i)
    {
        int nv = e[v][i];
        if (nv == p)
        {
            e[v][i] = e[v].back();
            e[v].pop_back();
            --i;
            continue;
        }
        dfs(nv, v, cdep + 1);
    }
}

int goUp(int x, int p)
{
    for (int i = D - 1; i >= 0; --i)
        if (p >= (1 << i))
    {
        x = dp[i][x];
        p -= (1 << i);
    }
    return x;
}

int ans = -1;

void dfs1(int v, int c)
{
    ans = max(ans, c += add[v]);
    for (int nv : e[v])
        dfs1(nv, c);
}
```

```

int main()
{
    fi>>n;
    for (int i = 0; i < n - 1; ++i)
    {
        int x, y;
        fi>>x>>y;
        --x;
        --y;
        e[x].push_back(y);
        e[y].push_back(x);
    }
    dfs(0, -1, 0);

    int m;
    fi>>m;
    for (int i = 0; i < m; ++i)
    {
        int x, y;
        fi>>x>>y;
        --x;
        --y;
        if (dep[x] < dep[y])
            swap(x, y);
        add[x] += 1;
        int x1 = goUp(x, dep[x] - dep[y] - 1);
        if (dp[0][x1] != y)
            add[y] += 1;
        else
        {
            add[0] += 1;
            add[x1] -= 1;
        }
    }

    dfs1(0, 0);
    fo << ans << "\n";
    fo<<"\nTime: "<<clock() / (double)1000<<" sec";
    return 0;
}

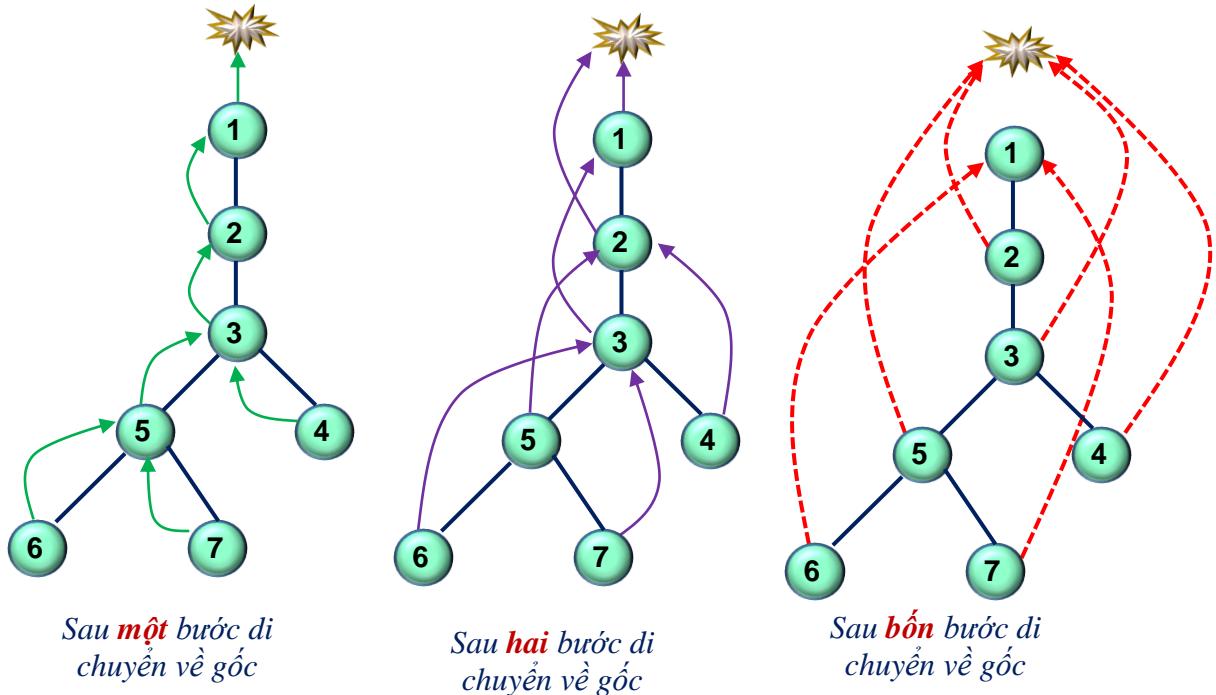
```



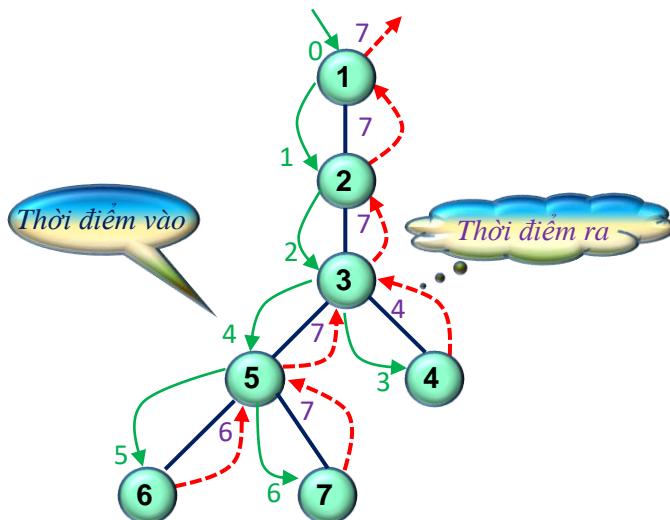
Giải thuật: Đồ thị cây, Quy hoạch động, Tìm kiếm nhị phân, Thời điểm vào/ra.

Nhận xét:

Xây dựng bảng kết quả chuyển về gốc sau một bước, sau 2 bước, sau 4 bước, . . .



Xác định thời điểm vào và ra ở mỗi nút,



Nếu thời điểm vào/ra của x và y thỏa mãn điều kiện:

$$\text{in}[x] \leq \text{in}[y] \leq \text{out}[y] \leq \text{out}[x]$$

hoặc $\text{in}[y] \leq \text{in}[x] \leq \text{out}[x] \leq \text{out}[y]$

thì điểm gốc có thể kiểm soát lộ trình từ x tới y,

Với lộ trình (x, y) cần đánh dấu kiểm soát ở đầu vào và xóa thông kê ở đầu ra.

Chương trình:

```
#include <bits/stdc++.h>
#define NAME "ambush."
using namespace std;
ifstream fi (NAME"inp");
ofstream fo (NAME"out");
typedef vector<int> vi;
const int N = 2e5 + 10;
const int LOG = 18;

vi v[N];
int up[N][LOG];
int n,m,in[N],out[N],T=0,ans=-1;

void wtr()
{
    for(int i=0;i<=10;++i)
    {for(int j=0;j<=n;++j) fo<<setw(3)<<up[i][j]; fo<<endl;}
    fo<<"... in/out:"<<endl;
    for(int i=0;i<n;++i) fo<<in[i]<<' '; fo<<endl;
    for(int i=0;i<n;++i) fo<<out[i]<<' '; fo<<endl;
}

void dfs(int x, int pr)
{
    up[x][0] = pr;
    for (int i = 1; i < LOG; ++i) up[x][i] = up[up[x][i - 1]][i - 1];
    in[x] = T++;

    for (int y : v[x])
    {
        if (y != pr) dfs(y, x);
    }

    out[x] = T;
}

int s[N];

void add(int l, int r, int dv)
{
    s[l] += dv;
    s[r] -= dv;
}

int main()
{
    fi>>n;
    for(int i=0;i<n-1;++i)
    {
        int x, y;
        fi>>x>>y;
        --x, --y;
        v[x].push_back(y);
    }
}
```

```

        v[y].push_back(x);
    }
dfs(0, 0);           wtr();

fi>>m;
for(int i=0; i<m; ++i)
{
    int x, y;
    fi>>x>>y;
    --x, --y;

    if (in[x] <= in[y] && out[y] <= out[x])
    {
        add(in[y], out[y], 1);
        for (int i = LOG - 1; i >= 0; --i)
        {
            int z = up[y][i];
            if (!(in[z] <= in[x] && out[x] <= out[z])) y = z;
        }
        add(0, n, 1);
        add(in[y], out[y], -1);
    }
    else if (in[y] <= in[x] && out[x] <= out[y])
    {
        add(in[x], out[x], 1);
        for (int i = LOG - 1; i >= 0; --i)
        {
            int z = up[x][i];
            if (!(in[z] <= in[y] && out[y] <= out[z])) x = z;
        }
        add(0, n, 1);
        add(in[x], out[x], -1);
    }
    else
    {
        add(in[x], out[x], 1);
        add(in[y], out[y], 1);
    }
}

int ss = 0;
for(int i = 0; i < n; ++i)
{
    ss += s[i];
    ans = max(ans, ss);
}
fo<<ans;

fo<<"\nTime: "<<clock() / (double) 1000<<" sec";
return 0;
}

```



Tranh thủ giờ các cháu tự chơi cô giáo ngồi cắt một thanh gỗ thành các đoạn con có độ dài khác nhau từng đôi một để chuẩn bị dạy các cháu về chủ đề phân biệt ngắn – dài. Các cháu dần dần bỏ chơi, xúm quanh xem cô làm và hỏi đủ mọi thứ trên đời. Đối với các cháu mẫu giáo thì quy tắc đối xử phải là *yêu thương, chân thành, công bằng* và *chuẩn xác*. Cô giáo kiên nhẫn trả lời mọi câu hỏi của các cháu. Cô đang cầm thanh gỗ độ dài 10.

Một cháu hỏi:

– Cô cắt nhiều nhất được thành mấy đoạn à? “Thành 4 đoạn” – cô trả lời.

Cả lớp nhao nhao:

– Thế nếu thanh gỗ dài bằng lớp học mình thì được nhiều nhất là mấy đoạn?

– Thế nếu nó dài bằng từ đây lên ông trăng?

– Bằng từ đây lên ông mặt trời?

.....

Cô toát cả mồ hôi hột, đành dùng kể hoãn binh: “Bằng từ đây lên ông mặt trời à? Dài quá. Để về nhà cô đếm đã, mai cô nói”.

May là trong số phụ huynh có người là lập trình viên. Khi nghe con kể chuyện trong lớp anh đã chuyển cho giáo một chương trình tính số lượng khúc gỗ cắt được từ thanh gỗ độ dài **1en**.

Hãy xác định kết quả của chương trình.

Dữ liệu: Vào từ file văn bản **TIMBER.INP** gồm một dòng chứa số nguyên **1en** ($1 \leq 1en \leq 10^{18}$).

Kết quả: Đưa ra file văn bản **TIMBER.OUT** một số nguyên – số lượng nhiều nhất các khúc gỗ nhận được sau khi cắt.

Ví dụ:

TIMBER.INP
10

TIMBER.OUT
4



Giải thuật: Cấp số cộng.

Nhận xét:

Để cắt được thành nhiều nhất các đoạn có độ dài khác nhau từng đôi một thì các đoạn đầu tiên phải có độ dài là 1, 2, 3, ...

Ta có thể nhận được **n** đoạn như vậy cho đến khi

$$\frac{n \times (n+1)}{2} \leq \text{len} \text{ và } d = \text{len} - \frac{n \times (n+1)}{2} \leq n$$

Thay đoạn cuối cùng bằng đoạn độ dài **n+d** ta được nhiều nhất số lượng các đoạn thỏa mãn điều kiện đã nêu.

Lưu ý: **len** có thể đạt tới 10^{18} , nên cần sử dụng dữ liệu kiểu **uint64_t**.

Độ phức tạp của giải thuật: O(1).

Chương trình:

```
#include "bits/stdc++.h"
#define NAME "timber."

using namespace std;
ifstream fi (NAME"inp");
ofstream fo (NAME"out");
uint64_t len,n;

int main()
{
    fi>>len;
    n=(-1+sqrt((uint64_t)1+8*len))/2;
    fo<<n;
}
```



VU42. MUỒNG HOÀNG YẾN

Tên chương trình: CASSIA.CPP

Trồng cây xanh dọc theo tuyến phố hoặc xa lộ không những mang lại bóng mát mà còn giúp giảm thiểu tiếng ồn, cải thiện chất lượng không khí và mang lại cảnh quan thân thiện. Các loại cây có thể trồng được chọn lựa rất kỹ và muồng hoàng yến là một trong số các cây thích hợp để trồng.

Xa lộ cao tốc Bắc – Nam có thể coi như một đường thẳng. Tại mỗi điểm có tọa độ nguyên có 1 cây được trồng. Trên toàn tuyến có tất cả n cây muồng hoàng yến, cây thứ i ở tọa độ a_i , $i = 1 \div n$.



Để tạo điểm nhấn du lịch người ta quyết định đánh cây, hoán đổi một số vị trí các cây để muồng hoàng yến tạo thành một dãy liên tục n cây.

Việc đánh cây để đổi chỗ là rất vất vả và tốn kém, vì vậy cần tìm phương án giảm thiểu số cặp cây cần đổi chỗ.

Hãy xác định số lượng ít nhất các cây cần đổi chỗ.

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

- + Dòng đầu tiên chứa một số nguyên n ($1 \leq n \leq 10^5$),
- + Dòng thứ 2 chứa n số nguyên a_1, a_2, \dots, a_n ($-10^9 \leq a_i \leq 10^9$, $i = 1 \div n$).

Kết quả: Đưa ra file văn bản CASSIA.OUT một số nguyên – số lượng ít nhất các cây cần đổi chỗ.

Ví dụ:

CASSIA.INP
5
3 1 -2 4 7

CASSIA.OUT
2



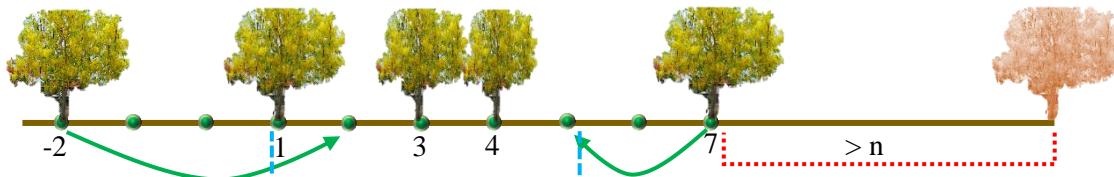
Giải thuật: Phương pháp 2 con trỏ.

Nhận xét:

Để giảm chi phí chuyển đổi ta cần tìm đoạn n điểm liên tiếp chứa nhiều muồng nhất, Cần sắp xếp các cây theo vị trí tăng dần,

Vị trí điểm đầu của đoạn không quan trọng, vì vậy có thể xuất phát từ vị trí của cây đang có,

Với điểm đầu của khoảng đang xét là cây thứ p tìm $q > p$ nhỏ nhất thỏa mãn điều kiện $a_q - a_p > n$,



Số lượng cây trong khoảng đang xét sẽ là $q-p$,

Xuất phát từ $p = 1, 2, \dots$ lần lượt xác định q , lưu ý là với mỗi p mới q sẽ *giữ nguyên hoặc tăng!*

Gọi **ans** là số lượng cây nhiều nhất trong các khoảng đã xét,

Quá trình tìm kiếm chấm dứt khi xét đến cây điểm đầu thứ n hoặc phần còn lại có ít hơn hoặc bằng **ans** cây,

Để tiện xử lý: thêm một *cây hàng rào* ở vị trí có khoảng cách tới cây cuối cùng lớn hơn n ,

Kết quả cần đưa ra: $n - ans$.

Tổ chức dữ liệu:

Mảng **int a[100002]** – lưu tọa độ các cây hiện có.

Lưu ý: Kỹ thuật 2 con trỏ luôn luôn có thể hiện thực hóa bằng hàng đợi 2 đầu **dequeue** hoặc các hàm phục vụ tìm kiếm nhị phân trong vector.

Độ phức tạp của giải thuật: $O(n \ln n)$ (do sử dụng sắp xếp).

Chương trình 1: Tô chúc 2 con trồ.

```
#include "bits/stdc++.h"
#define NAME "cassia."
using namespace std;
ifstream fi (NAME"inp");
ofstream fo (NAME"out");
int n,p=0,q=0,ans=0;
int a[100002];

int main()
{
    fi>>n;
    for(int i=0;i<n;++i) fi>>a[i];
    sort(a,a+n);
    a[n]=a[n-1]+n+1;
    for(int i=0; i<n && ans+i<n; ++i)
    {
        p=i;
        while(a[q]-a[p]<= n) ++q;
        if(ans<q-p) ans=q-p;
    }
    ans=n-ans;
    fo<<ans;
    fo<<"\nTime: "<<clock() / (double)1000<<" sec";
}
```

Chương trình 2: Dùng vector và các hàm tìm kiếm nhị phân.

```
#include <bits/stdc++.h>
#define NAME "cassia."
using namespace std;
ifstream fi (NAME"inp");
ofstream fo (NAME"out");

int main()
{
    int n;
    fi>>n;
    vector<int> cassia(n);
    for (int i = 0; i < n; i++)
        fi>>cassia[i];
    sort(cassia.begin(), cassia.end());
    int answer = INT_MAX;
    for (int i = 0; i < n; i++) {
        int l = cassia[i];
        int r = cassia[i] + n;
        int index = lower_bound(cassia.begin(), cassia.end(), r) -
cassia.begin();
        int cassiaCount = index - i;
        answer = min(answer, n - cassiaCount);
    }
    fo<<answer;
    fo<<"\nTime: "<<clock() / (double)1000<<" sec";
    return 0;
}
```

}



VU43. ĐỘ TRẺ CỦA SỐ

Tên chương trình: YOUTH.CPP

Chữ số 0 được phát minh muộn nhất trong số các chữ số. Giáo sư Braun trong công trình nghiên cứu về lịch sử số học đã dành hẳn một chương nói về số 0. Các chữ số 0 không có nghĩa (những chữ số 0 bên trái trước chữ số khác 0 đầu tiên) đương nhiên là không cần thiết và sẽ không được xét. Những chữ số cuối cùng của một số và là 0 cũng có thể tránh bằng cách thay cụm số 0 đó bằng từ chục, trăm, nghìn, . . . Nhưng những chữ số 0 ở giữa thì không thể bỏ qua hay tránh.

Giáo sư Braun đưa ra khái niệm “độ trẻ” của một số thập phân nguyên dương là số lượng chữ số 0 không đứng cuối trong số, tức là sau nó còn ít nhất một chữ số khác 0. Ví dụ, số 980090600 có độ trẻ là 3.

Cho số nguyên dương n . Hãy xác định độ trẻ của nó.

Dữ liệu: Vào từ file văn bản YOUTH.INP gồm một dòng chứa số nguyên n ($1 \leq n \leq 10^{18}$).

Kết quả: Đưa ra file văn bản YOUTH.OUT một số nguyên – độ trẻ của số đã cho.

Ví dụ:

YOUTH.INP	YOUTH.OUT
980090600	3



VU43 StP_Mui 20161212 A

Giải thuật: Xử lý xâu.

Nhận xét:

- ✚ Số cần được nhập và lưu trữ dưới dạng xâu,
- ✚ Tìm vị trí chữ số khác 0 bên phải nhất của xâu,
- ✚ Đếm các ký tự 0 từ đầu xâu đến vị trí đã xác định.

Độ phức tạp của giải thuật: O(1).

Chương trình:

```
#include "bits/stdc++.h"
#define NAME "youth."
using namespace std;
ifstream fi (NAME"inp");
ofstream fo (NAME"out");
int ln,k,ans=0;
string s;

int main()
{
    fi>>s; ln=s.size();
    k=ln-1;
    while(s[k]=='0') --k;
    for(int i=0;i<=k;++i) ans+=(s[i]=='0');

    fo<<ans;
    fo<<"\nTime: "<<clock() / (double)1000<<" sec";
}
```



VU44. PHÂN SỐ

Tên chương trình: FRACTIONS.CPP

Cho 2 số nguyên dương p và q ($q < p$). Hãy đưa ra tất cả các phân số tối giản f có mẫu số không vượt quá n cho trước và thỏa mãn điều kiện:

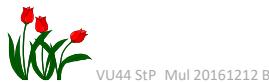
$$\frac{1}{p} < f < \frac{1}{q}$$

Dữ liệu: Vào từ file văn bản FRACTIONS.INP gồm một dòng chứa 3 số nguyên n , p và q ($1 \leq n \leq 100$, $1 \leq q < p \leq 100$).

Kết quả: Đưa ra file văn bản FRACTIONS.OUT các phân số tìm được theo thứ tự giá trị tăng dần, mỗi phân số trên một dòng và dưới dạng a/b .

Ví dụ:

FRACTIONS.INP	FRACTIONS.OUT
10 3 2	3/8 2/5 3/7 4/9



VU44 StP_Mul 20161212 B

Giải thuật: Duyệt vét cạn, Kỹ năng khai thác hệ thống lập trình C++.

Nhận xét:

Miền giá trị các phân số cần xét là không lớn: tất cả các phân số $\frac{x}{y}$ thỏa mãn các điều kiện:

- ♣ $x < y$, $1 < y \leq n$,
- ♣ x và y – nguyên tố cùng nhau,
- ♣ $\frac{1}{p} < \frac{x}{y} < \frac{1}{q}$

Vấn đề cần thực hiện là vét cạn các khả năng của x , y thỏa mãn điều kiện đầu, kiểm tra các điều kiện thứ 2 và thứ 3 và ghi nhận nếu thỏa mãn.

Tổ chức dữ liệu:

Mảng `vector<pii> a` – cặp giá trị, ghi nhận các phân số thỏa mãn điều kiện cần tìm.

Xử lý:

Kiểm tra 2 số x và y có nguyên tố cùng nhau hay không: ước số chung lớn nhất của x và y phải bằng 1, dùng hàm `__gcd(x,y)`,

Khi sắp xếp các phân số nhận được: cần xác định tiêu chuẩn sắp xếp:

```
[] (pii x,pii y) {return x.first*y.second<x.second*y.first; }
```

Độ phức tạp của giải thuật: $O(n^2)$.

Chương trình:

```
#include "bits/stdc++.h"
#define NAME "fractions."
using namespace std;
ifstream fi (NAME"inp");
ofstream fo (NAME"out");
typedef pair<int,int> pii;
int n,p,q;
vector<pii> a;

int main()
{
    fi>>n>>p>>q;
    for(int i=1;i<n;++i)
        for(int j=i+1;j<=n;++j)
            if(__gcd(i,j)==1 && j<p*i && i*q<j) a.push_back({i,j});

    sort(a.begin(),a.end(), [] (pii x,pii y)
        {return x.first*y.second<x.second*y.first;});
    for(auto &k:a) fo<<k.first<<'/'<<k.second<<'\n';
}

fo<<"\nTime: " <<clock() / (double)1000<<" sec";
```

C++11



VU45. TÔ MÀU

Tên chương trình: TINGE.CPP

Cuộc thi ROBOTCOM được tổ chức theo thể lệ mới, không có hoạt động đối kháng, nhưng các robot sau khi được khởi động phải hoạt động độc lập, không có sự điều khiển từ bên ngoài của các tác giả. Nhiệm vụ của robot là nhận dạng và tô màu lưới ô vuông kích thước $n \times n$ ô, mỗi ô được tô một trong số 26 màu tùy chọn, mỗi màu được ký hiệu bằng một chữ cái la tinh thường trong phạm vi từ ‘**a**’ đến ‘**z**’, các chữ cái khác nhau tương ứng với các màu khác nhau. Ban Giám khảo sẽ đánh giá về chất lượng tô, sự chuyển tiếp hài hòa các màu từ ô này sang ô khác kề cạnh, tốc độ tô, . . .

Steve cài đặt chương trình tô màu trong robot của mình như sau:

- Các ô trên hai đường chéo – tô màu **a**,
- Các ô còn lại: có màu phụ thuộc vào khoảng cách gần nhất tới các đường chéo, đầu tiên tô màu **b**, ô tiếp theo – màu **c**, . . . Tiếp sau màu **z** là màu **a**.

Cho số nguyên **n**. Hãy xác định màu ở các ô trong trường cần tô.

Dữ liệu: Vào từ file văn bản TINGE.INP gồm một dòng chứa số nguyên **n** ($1 \leq n \leq 100$).

Kết quả: Đưa ra file văn bản TINGE.OUT **n** dòng, mỗi dòng chữ xâu độ dài **n** xác định màu các ô trên lưới cần tô màu.

Ví dụ:

TINGE.INP	TINGE.OUT
5	abcba babab cbabc babab abcba



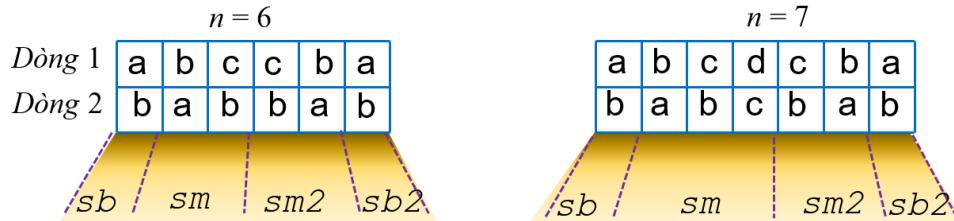
VU45 StP_Mul 20161212 C

Giải thuật: Xử lý xâu.

Nhận xét:

Chỉ cần xác định $(n+1)/2$ dòng đầu tiên, các dòng còn lại có thể xác định theo quy tắc đối xứng gương,

Với mỗi dòng: chỉ cần xác định giá trị ở nửa đầu, phần còn lại – theo quy tắc đối xứng,



Chia một dòng thành 4 phần: phần đầu **sb**, phần giữa **sm**, phần giữa thứ 2 **sm2** và phần cuối **sb2**,

sb của dòng thứ **i** với $1 \leq i \leq (n+1)/2$ có $i-1$ ký tự,

sm của dòng thứ **i** với $1 \leq i \leq (n+1)/2$ có $(n+1)/2 - i + 1$ ký tự,

sb2 là xâu đảo ngược của **sb**,

sm2 là xâu đảo ngược của **sm**, nếu **n** lẻ thì cần bót một ký tự đầu sau khi đảo ngược,

Dòng thứ **i** sẽ bằng dòng **n-i+1**, **i = 1, 2, ...**

Tổ chức dữ liệu:

Mảng **string a[51]** – lưu kết quả,

Các xâu **sb, sm, sm2, sb2**: có vai trò đã nêu ở trên.

Xử lý:

Khởi tạo các xâu:

```

fi>>n; p=(n+1)/2; flg=n&1;
sm="" ; c='a'; sb="" ; se="";
for(int i=0;i<p;++i)
{
    sm+=c; ++c; if(c>122)c=97;
}

```

Tạo các xâu còn lại: dùng hàm **reverse** để đảo ngược xâu.

Độ phức tạp của giải thuật: $O(n^2)$.

Chương trình:

```
#include "bits/stdc++.h"
#define NAME "tinge."
using namespace std;
ifstream fi (NAME"inp");
ofstream fo (NAME"out");
int n,k,p;
string a[101],sb,sb2,sm,sm2;
char c;
bool flg;

int main()
{
    fi>>n; p=(n+1)/2; flg=n&1;
    sm=""; c='a'; sb="";
    for(int i=0;i<p;++i)
    {
        sm+=c; ++c; if(c>122) c=97;
    }
    c='a'; k=p-1;
    for(int i=0;i<p;++i)
    {
        sm2=sm; reverse(sm2.begin(),sm2.end()); if(flg) sm2.erase(0,1);
        a[i]=sb+sm+sm2+sb2;
        ++c; if(c>122) c=97; sb=c+sb; sm.erase(k,1); --k;
        sb2=sb; reverse(sb2.begin(),sb2.end());
    }
    for(int i=0;i<p;++i) fo<<a[i]<<'\n';
    k=(flg)?p-1:p;
    for(int i=k-1;i>=0;--i) fo<<a[i]<<'\n';
    fo<<"\nTime: "<<clock()/(double)1000<<" sec";
}
```



VU46. TÁC ĐỘNG HỖ TRỢ

Tên chương trình: SUPPORT.CPP

Một số loại cây trồng xen nhau sẽ có tác dụng hỗ trợ và làm tăng năng suất chung. Ví dụ người ta thường trồng xen lạc với đỗ hay cà chua.

Có n loại cây có thể trồng xen nhau, cây thứ i sẽ tạo ra hiệu ứng hỗ trợ a_i , $i = 1 \div n$, trong đó các a_i khác nhau từng đôi một và nhận giá trị nguyên trong phạm vi từ 1 đến n . Khi trồng cạnh nhau thành một đường thẳng mỗi cây, trừ hai cây đầu và cuối, sẽ tác động lên hai cây bên cạnh tạo ra một hiệu ứng hỗ trợ chung là $a_1a_2 + a_2a_3 + \dots + a_{n-1}a_n$.

Thay đổi vị trí các cây trong hàng ta sẽ có những hiệu ứng hỗ trợ chung khác nhau.

Hãy xác định có bao nhiêu cách trồng để hiệu ứng hỗ trợ chung là bội của số nguyên k cho trước.

Hai cách trồng gọi là khác nhau nếu tồn tại ít nhất một vị trí i chừa cây khác nhau trong hai cách trồng.

Dữ liệu: Vào từ file văn bản SUPPORT.INP gồm một dòng chứa 2 số nguyên n và k ($1 \leq n \leq 10$, $2 \leq k \leq 1000$).

Kết quả: Đưa ra file văn bản SUPPORT.OUT một số nguyên – số cách trồng tìm được.

Ví dụ:

SUPPORT.INP	SUPPORT.OUT
3 2	2



Giải thuật: Vét cạn, Khai thác khả năng C++.

Nhận xét:

Số lượng các hoán vị cần xét không vượt quá $10! = 3\ 628\ 800$,

Nếu tìm cách khởi tạo nhanh được các hoán vị, ta có thể dùng phương pháp vét cạn: dẫn xuất hoán vị và tính trực tiếp hiệu ứng hỗ trợ chung,

Để khởi tạo nhanh hoán vị: cần chuyển từ một hoán vị sang hoán vị tiếp sau theo thứ tự từ điển,

Công cụ trong C++: hàm `next_permutation(a+1, a+n+1)` khởi tạo hoán vị tiếp theo theo thứ tự từ điển các phần tử a_1, a_2, \dots, a_n . Nếu không tồn tại hoán vị lớn hơn tiếp theo – trả về giá trị `false`. Độ phức tạp khởi tạo là $O(n)$.

Độ phức tạp của giải thuật: $O(n^2)$.

Chương trình:

```
#include "bits/stdc++.h"
#define NAME "support."
using namespace std;
ifstream fi (NAME"inp");
ofstream fo (NAME"out");
int n,k,p=1,ans=0;
int a[12];

int main()
{
    fi>>n>>k;
    for(int i=1;i<=n;++i) a[i]=i;
    do
    {
        int q=0;
        for(int i=1;i<n;++i) q+=a[i]*a[i+1];
        if(q%k==0) ++ans;
    }while(next_permutation(a+1,a+n+1));

    fo<<ans;
    fo<<"\nTime: "<<clock() / (double)1000<<" sec";
}
```



Cho lưới ô vuông r dòng và c cột. Dòng được đánh số từ trên xuống dưới bắt đầu từ 1, cột đánh số từ trái sang phải, bắt đầu từ 1. Các ô được đánh số theo đường chéo từ trên trái xuống dưới phải, bắt đầu bằng 1. Ví dụ, với $r = 3$, $c = 5$ ta có:

1	2	4	7	10
3	5	8	11	13
6	9	12	14	15

Cho q số. Với mỗi số đã cho hãy xác định dòng và cột của ô chứa số đó.

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

- ✚ Dòng đầu tiên chứa 3 số nguyên r , c và q ($1 \leq r, c \leq 10^9$, $1 \leq q \leq 100$),
- ✚ Dòng thứ 2 chứa q số nguyên dương, mỗi số không vượt quá $r \times c$.

Kết quả: Đưa ra file văn bản NUMPOS.OUT q dòng, mỗi dòng chứa 2 số nguyên xác định tọa độ dòng, cột của số đã cho.

Ví dụ:

NUMPOS.INP	NUMPOS.OUT
<pre>3 5 4 2 8 14 15</pre>	<pre>1 2 2 3 3 4 3 5</pre>



Giải thuật: Xử lý số học.

Nhận xét:

Đánh số các đường chéo từ 1 đến $r+c-1$,

Với mỗi số k cần xác định đường chéo d chứa nó và vị trí của k trên đường chéo,

Các đường chéo được chia thành 3 nhóm:

- ✚ Nhóm 1: độ dài đường chéo tăng dần và nhỏ hơn $mn = \min\{r, c\}$,
- ✚ Nhóm 2: các đường chéo độ dài bằng mn ,
- ✚ Nhóm 3: độ dài đường chéo giảm dần và nhỏ hơn $mn = \min\{r, c\}$,

Gọi p_1 – số lượng ô trên các đường chéo nhóm 1, p_2 – số lượng ô trên các đường chéo nhóm 2,

Khi $k \leq p_1$: giải phương trình bậc 2 để tìm d và từ đó – tìm vị trí trên đường chéo,

Nếu $p_1 < k \leq p_1 + p_2$ – dễ dàng tính trực tiếp các tham số cần tìm,

Trường hợp rơi vào nhóm 3: Giải quyết tương tự như trường hợp 1 với phần bù của k tới p_1 ,

Dẫn xuất nghiệm: phân biệt trường hợp $d \leq c$ và $d > c$.

Độ phức tạp của giải thuật: $O(q)$.

Chương trình:

```
#include "bits/stdc++.h"
#define NAME "numpos."
using namespace std;
ifstream fi (NAME"inp");
ofstream fo (NAME"out");
typedef pair<int,int> pii;
int64_t r,c,q,m,k,u,v,d,mn,mx,p1,p2,cr;

int64_t calc_d(int64_t x)
{int64_t y;
y=(-1+sqrt((int64_t)1+8*x))/2;
if(y*(y+1)/2<x)++y;
return y;
}
int main()
{
    fi>>r>>c>>q;
    mn=min(r,c); mx=max(r,c); cr=c+r;
    p1=mn*(mn-1)/2; p2=(mx-mn+1)*mn;
    for(int i=0;i<q;++i)
    {
        fi>>m; d=1;k=m;
        if(k<=p1)
        {
            d=calc_d(k);
            k-=d*(d-1)/2;
        }
        else { k-=p1;d=mn;
            if(k<=p2) {d=(k+d-1)/d+mn-1;k%=mn; if(k==0) k=mn; }
            else
            {
                k+=p1;m=c*r-k+1; d=calc_d(m); k=d-(m-d*(d-1)/2)+1;
                d=cr-d;
            }
        }
        if(d<=c) u=1,v=d; else u=d-c+1,v=c;
        u+=k-1; v-=k-1;
        fo<<u<<' ' <<v<<'\n';
    }
    fo<<"\nTime: "<<clock()/(double)1000<<" sec";
}
```



Các lâu đài cổ nổi tiếng bởi những lối đi bí mật nối giữa các căn phòng. Lâu đài do công ty du lịch quản lý có n phòng, 2 phòng bất kỳ của lâu đài được nối bằng một hành lang bí mật. Có hai nhóm phụ trách việc làm vệ sinh, lau chùi và khử nấm mốc ở các hành lang. Mỗi hành lang thuộc một trong hai nhóm trên. Một nhóm chuyên dùng tinh dầu sả để khử mốc, nhóm kia – dùng tinh dầu bạch đàn. Nhóm dùng tinh dầu sả phụ trách m hành lang, hành lang thứ i nối 2 phòng a_i và b_i , $i = 1 \dots m$, không có 2 hành lang nào trùng nhau.

Trên vé tham quan có chỉ ra 2 số nguyên x và y ($x \leq y$). Khách du lịch sẽ được dẫn đi theo một chu trình khép kín, bắt đầu từ một phòng trong đoạn $[x, y]$, đi theo hành lang tới các phòng khác có số cũng thuộc đoạn nói trên và cuối cùng – quay trở lại phòng xuất phát, mỗi phòng chỉ đi qua đúng một lần, trừ phòng xuất phát là 2 lần (khi khách trở về). Số lượng phòng ở mỗi tour không quá 100.

Kinh nghiệm cho thấy khi chuyển từ hành lang có mùi tinh dầu này sang hành lang có mùi tinh dầu khác, khách có thể bị dị ứng. Chính vì vậy khách phải được dẫn đi theo các hành lang cùng một loại mùi. Có q vé được bán ra. Với mỗi vé hãy chỉ ra số lượng phòng trên tour và các phòng theo trình tự đi. Nếu không có cách tổ chức tour thì đưa ra số -1. Trong trường hợp có nhiều cách tổ chức tour – đưa ra cách tùy chọn.

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

- + Dòng đầu tiên chứa 2 số nguyên n và m ($1 \leq n \leq 10^5$, $0 \leq m \leq \min\{10^5, n \times (n-1)/2\}$),
- + Dòng thứ i trong m dòng sau chứa 2 số nguyên a_i và b_i ($1 \leq a_i, b_i \leq n$, $a_i \neq b_i$),
- + Dòng $m+2$ chứa số nguyên q ($1 \leq q \leq 10^5$),
- + Mỗi dòng trong q dòng sau chứa 2 số nguyên x và y ($1 \leq x \leq y \leq n$).

Kết quả: Đưa ra file văn bản TOUR.OUT q dòng, mỗi dòng chứa thông tin về một tour, tương ứng với trình tự thông tin ở input.

Ví dụ:

TOUR.INP
5 5
1 2
2 3
3 4
4 5
5 1
2
1 5
1 4

TOUR.OUT
5 4 2 5 3 1
-1



Giải thuật: Loang trên đồ thị.

Nhận xét:

Đồ thị đã cho là đầy đủ: giữa 2 nút bất kỳ đều có đường đi,

Có 2 loại đường đi, đánh dấu là 0 hoặc 1,

Chỉ cần tìm chu trình chứa ít đỉnh nhất trong đoạn đã cho, vì vậy có thể dùng kỹ thuật loang tìm chu trình lần lượt với các loại đường đi,

Với việc cho đoạn $[x, y]$, dữ liệu đỉnh không có tính chất vòng tròn,

Số đỉnh trong chu trình không được vượt quá 100 và nằm trong đoạn đã cho, vì vậy không cần quan tâm đánh dấu các đường từ x tới y có $|x-y| > 7$, tức là chỉ trực tiếp kiểm soát các đoạn không quá 15 điểm liên tiếp (có hơn 100 đường đi để xét chu trình),

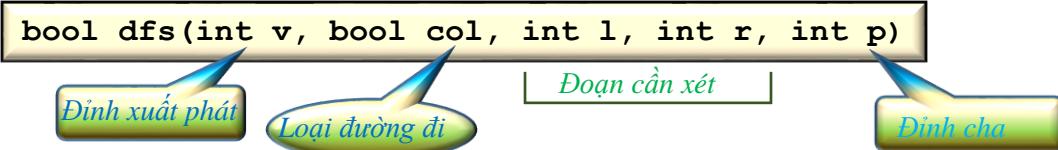
Dễ dàng chứng minh rằng với mỗi đoạn $[x, y]$ ta *không cần xét quá 5 điểm xuất phát kể từ x !*

Tổ chức dữ liệu:

- ▀ Mảng `bool e[N][K*2+1]` – đánh dấu hành lang mùi tinh dầu sả,
- ▀ Mảng `bool vis[N]` – đánh dấu các điểm đã duyệt,
- ▀ Mảng `vector < int > st` – lưu đường đi.

Xử lý:

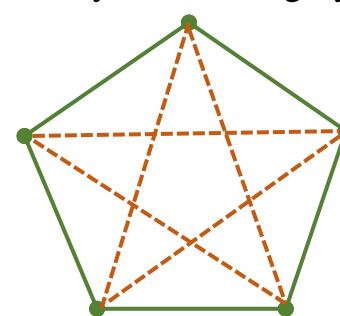
Dùng hàm đệ quy loang theo chiều sâu để tìm chu trình:



Chú ý:

- ♣ Xóa mảng đánh dấu điểm đã duyệt trước khi xử lý mỗi loại đường đi,
- ♣ Xóa mảng ghi kết quả trước mỗi lần tìm đường đi,
- ♣ Xóa phần tử cuối (bị lặp) trong mảng kết quả.

Độ phức tạp của giải thuật: $\approx O(q \lg n)$.



Chương trình:

```
#include <bits/stdc++.h>
#define NAME "tour."
using namespace std;
ifstream fi (NAME"inp");
ofstream fo (NAME"out");

const int N = 100002;
const int K = 7;
int n, m, q;
bool e[N][K*2+1];

bool vis[N];
vector < int > st;
bool dfs(int v, bool col, int l, int r, int p)
{
    st.push_back(v);
    vis[v] = true;
    for (int nv = l; nv <= r; ++nv)
    {
        if (nv == v || nv == p || e[v][nv] != col)
            continue;
        if (vis[nv])
        {
            st.push_back(nv);
            return true;
        } else if (dfs(nv, col, l, r, v))
            return true;
    }
    st.pop_back();
    return false;
}

int main()
{
    fi>>n>>m;
    for (int i = 0; i < m; ++i)
    {
        int x, y;
        fi>>x>>y;
        --x; --y;
        if (-K <= y-x && y-x <= K)
        {
            e[x][y]=1; e[y][x]=1;
        }
    }
    fi>>q;
    for (int i = 0; i < q; ++i)
    {
        int l, r;
        fi>>l>>r;
        --l; --r;
        r = min(r, l + 4);
        bool ok = false;
        for (int col = 0; col <= 1 && !ok; ++col)
        {
            if(ok)break;
```

```

memset(vis + l, 0, sizeof(bool)*(r-l+1));
for (int v = l; v <= r; ++v)
{
    if (vis[v]) continue;
    st.clear();
    if (dfs(v, col, l, r, -1))
    {
        st.pop_back();
        fo<<st.size()<<' ';
        for(int j:st) fo<<j+1<<' ';
        fo<<'\n';ok=true;break;
    }
}
if (!ok)
    fo << "-1\n";
}
fo<<"\nTime: "<<clock() / (double)1000<<" sec";
}

```



VV01. CHUNG CỤ

Tên chương trình: BUILDING.CPP

Nhà tập thể cho sinh viên đại học có n tầng, đánh số từ 1 đến n từ dưới lên trên và chia thành một số khu vực độc lập, mỗi khu có một lối vào ở tầng 1.

Ở mỗi tầng trong một khu có y phòng, trừ các tầng có số chia hết cho k – có x phòng. Các phòng được đánh số 1, 2, 3, … bắt đầu từ tầng 1 cho đến tầng n của khu thứ nhất, sau đó đánh số tiếp theo từ tầng 1 của khu thứ 2, … cứ như thế đến khu cuối cùng.

Ví dụ với $n = 7$, $k = 3$, với 3 lối vào và $x = 2$, $y = 3$, các phòng được đánh số như sau:

	Khu vực 1	Khu vực 2	Khu vực 3
Tầng 7	17, 18, 19	36, 37, 38	55, 56, 57
Tầng 6	15, 16	34, 35	53, 54
Tầng 5	12, 13, 14	31, 32, 33	50, 51, 52
Tầng 4	9, 10, 11	28, 29, 30	47, 48, 49
Tầng 3	7, 8	26, 27	45, 46
Tầng 2	4, 5, 6	23, 24, 25	42, 43, 44
Tầng 1	1, 2, 3	20, 21, 22	39, 40, 41

Cho q số phòng a_1, a_2, \dots, a_q . Với mỗi số phòng đã cho hãy xác định tầng của phòng đó.

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

- ✚ Dòng đầu tiên chứa 4 số nguyên n, k, x, y ($1 \leq n \leq 10^9, 1 \leq k \leq n, 1 \leq x, y \leq 10^9$),
- ✚ Dòng thứ 2 chứa số nguyên q ($1 \leq q \leq 1000$),
- ✚ Dòng thứ 3 chứa q số nguyên a_1, a_2, \dots, a_q ($1 \leq a_i \leq 10^{18}, i = 1 \div q$).

Kết quả: Đưa ra file văn bản BUILDING.OUT q số nguyên – các tầng tìm được, mỗi số trên một dòng.

Ví dụ:

BUILDING.INP	BUILDING.OUT
7 3 2 3 4 1 19 20 50	1 7 1 5



Giải thuật: Xử lý số học (Biến đổi địa chỉ).

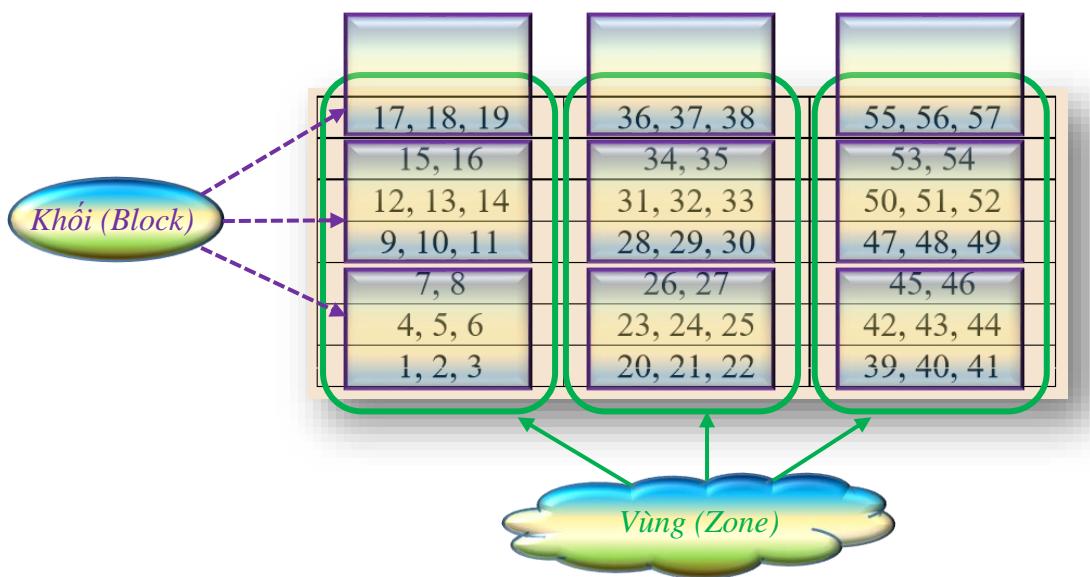
Nhận xét:

Mỗi cửa vào tạo thành một khu vực độc lập, gọi là **Vùng (Zone)**,

Mỗi vùng được chia thành các nhóm k tầng được gọi là **Khối (Block)**, khối trên cùng có thể không có đầy đủ **k** tầng,

Việc đánh số các phòng liên tiếp từ 1 trở đi đến hết được gọi là đánh số theo địa chỉ tuyệt đối (**Absolute Address**),

Phòng cũng có thể đánh dấu theo địa chỉ vật lý (**Physical Address**) bằng nhóm 3 số nguyên (**Zone**, **Block**, **Số thứ tự trong Block**),



Từ địa chỉ tuyệt đối đã cho ta có thể tính địa chỉ vật lý của phòng, từ đó dễ dàng xác định tầng của phòng đó.

Độ phức tạp của giải thuật: $O(q)$.

Chương trình:

```
#include <bits/stdc++.h>
#define NAME "building."
using namespace std;
ifstream fi (NAME"inp");
ofstream fo (NAME"out");
int64_t n,k,x,y,q,a,ans;
int64_t block,zone,t,z,b;

int main()
{
    fi>>n>>k>>x>>y;
    t=n/k;
    zone=(n-t)*y+t*x;
    block=(k-1)*y+x;
    fi>>q;
    for(int i=0;i<q;++i)
    {
        fi>>a;
        z=(a+zone-1)/zone; a==(z-1)*zone;
        b=(a+block-1)/block; a==(b-1)*block;
        if(a<=(k-1)*y) t=(a+y-1)/y; else t=k;
        ans=(b-1)*k+t;
        fo<<ans<<'\\n';
    }

    fo<<"\\nTime: "<<clock () / (double) 1000<<" sec";
}
```



VV02. MÁY TÍNH BẤM TAY

Tên chương trình: CALC.CPP

Máy tính bấm tay cũng được trang bị một số trò chơi giải trí. Một trong số các trò chơi có nội dung như sau. Trên màn hình hiển thị một số nguyên không âm. Người chơi có thể bấm 3 phím **A**, **B** và **C** để thay đổi giá trị của số trên màn hình.

Khi bấm phím **A** số trên màn hình sẽ bị chia đôi, phần lẻ sẽ bị bỏ nếu có. Ví dụ số trên màn hình là 80 thì kết quả sẽ là 40, nếu số là 89 thì kết quả là 44.

Khi bấm phím **B** số trên màn hình được cộng thêm 1 và sau đó chia đôi, bỏ phần lẻ (nếu có). Ví dụ với số 80, kết quả là 40, với số 89 – kết quả là 45.

Khi bấm phím **C**, nếu số trên màn hình là dương thì sẽ bị trừ 1 sau đó chia đôi, bỏ phần lẻ (nếu có). Nếu số trên màn hình là 0 thì kết quả không thay đổi. Ví dụ, với số 80, kết quả là 39, còn với số 89 kết quả là 44.

Khi kích hoạt trò chơi, trên màn hình hiển thị số nguyên dương **n** và 3 số nguyên **a**, **b**, **c**. Người chơi phải bấm tổng cộng là **a** lần phím **A**, **b** lần phím **B** và **c** lần phím **C**. Nhiệm vụ của người chơi là làm cho màn hình hiển thị số nhỏ nhất có thể có được từ số **n** ban đầu.

Hãy xác định số nhỏ nhất đó.

Dữ liệu: Vào từ file văn bản CALC.INP gồm một dòng chứa 4 số nguyên **n**, **a**, **b**, **c** ($1 \leq n \leq 10^{18}$, $0 \leq a, b, c \leq 60$).

Kết quả: Đưa ra file văn bản CALC.OUT một số nguyên – số nhỏ nhất có thể nhận được.

Ví dụ:

CALC.INP	CALC.OUT
72 2 1 1	4



VV02 Reg_Rus 20170204 B

Giải thuật: Quy hoạch động.

Nhận xét:

Mỗi lần bấm phím bất kỳ số trên màn hình sẽ giảm hoặc giữ nguyên (nếu bằng 0), Kết quả chỉ phụ thuộc vào số hiện tại trên màn hình và số lượng bấm còn lại của mỗi loại phím,

Như vậy bài toán có thể giải bằng phương pháp quy hoạch động,

Gọi $d_{i,j,k}$ – số nhỏ nhất có thể nhận được từ n bằng cách nhấn tổng cộng i lần phím **A**, j lần phím **B** và k lần phím **C**, ta có công thức truy hồi:

```
if (i) d[i][j][k] = min(d[i][j][k], d[i-1][j][k]/2);  
if (j) d[i][j][k] = min(d[i][j][k], (d[i][j-1][k]+1)/2);  
if (k) d[i][j][k] = min(d[i][j][k], (d[i][j][k-1]-1)/2);
```

Công thức truy hồi trên phải tính với mọi $i = 0 \div a$, $j = 0 \div b$, $k = 0 \div c$,

Chuẩn bị giá trị đầu: vì **a**, **b**, **c** đủ nhỏ nên có thể chuẩn bị $d_{i,j,k} = n$ với mọi i, j, k .

Kết quả: ở biến $d_{a,b,c}$.

Mảng giá trị **d** cần có kiểu **int64_t**.

Độ phức tạp của giải thuật: O(1).

Chương trình:

```
#include <bits/stdc++.h>
#define NAME "calc."
using namespace std;
ifstream fi (NAME"inp");
ofstream fo (NAME"out");
int a,b,c;
int64_t n,d[61][61][61];

int main()
{
    fi>>n>>a>>b>>c;

    for (int i = 0; i <= a; ++i)
        for (int j = 0; j <= b; ++j)
            for (int k = 0; k <= c; ++k) d[i][j][k]=n;

    for (int i = 0; i <= a; ++i)
        for (int j = 0; j <= b; ++j)
            for (int k = 0; k <= c; ++k)
            {
                if (i) d[i][j][k] = min(d[i][j][k], d[i - 1][j][k] / 2);
                if (j) d[i][j][k] = min(d[i][j][k], (d[i][j - 1][k]+1) / 2);
                if (k) d[i][j][k] = min(d[i][j][k], (d[i][j][k - 1]-1) / 2);
            }
    fo<<d[a][b][c];
}

fo<<"\nTime: "<<clock() / (double) 1000<<" sec";
}
```



VV03. TRIỂN LÃM

Tên chương trình: EXHIBITION.CPP

Thiết bị thông minh công nghệ cao thu hút sự chú ý của mọi tầng lớp xã hội. Những nhà phát minh sáng chế muốn thiết kế các loại thiết bị mới độc đáo, các nhà sản xuất muốn chọn các mẫu thiết kế ăn khách trên thị trường, người tiêu dùng muốn tìm những sản phẩm hợp với nhu cầu và khả năng của mình.

Khu công nghệ cao có n sản phẩm dự kiến có thể đưa ra giới thiệu ở Triển lãm Công nghệ thông minh. Nhưng ngay cả mặt trời cũng còn có vết đen, mỗi sản phẩm đều có mặt ưu điểm và mặt nhược điểm của mình. Ví dụ, có sản phẩm rất hoàn hảo về khía cạnh công nghệ, nhưng giá lại quá cao, có thiết bị cho phép quay phim, chụp ảnh cực tốt nhưng thời lượng pin lại hơi yếu, . . . Sản phẩm thứ i có chỉ số ưu việt là \mathbf{x}_i và nhược điểm là \mathbf{y}_i , $i = 1 \div n$. Để bộ sản phẩm mang ra triển lãm có tính đa dạng cao người ta quyết định chọn bộ sản phẩm sao cho với i và j được chọn cần có $\mathbf{x}_i - \mathbf{y}_j \neq \mathbf{x}_j - \mathbf{y}_i$.

Hãy xác định số lượng sản phẩm nhiều nhất được chọn để triển lãm.

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

- ✚ Dòng đầu tiên chứa một số nguyên n ($1 \leq n \leq 10^5$),
- ✚ Dòng thứ i trong n dòng sau chứa 2 số nguyên \mathbf{x}_i và \mathbf{y}_i ($1 \leq \mathbf{x}_i, \mathbf{y}_i \leq 10^9$).

Kết quả: Đưa ra file văn bản EXHIBITION.OUT một số nguyên – số lượng sản phẩm nhiều nhất được chọn.

Ví dụ:

EXHIBITION.INP
5
4 5
2 8
1 3
7 3
6 5

EXHIBITION.OUT
4



VV03_Io20170305_A

Giải thuật: Lọc dữ liệu.

Nhận xét:

Từ yêu cầu $\mathbf{x}_i - \mathbf{y}_j \neq \mathbf{x}_j - \mathbf{y}_i$ ta có $\mathbf{x}_i + \mathbf{y}_i \neq \mathbf{x}_j + \mathbf{y}_j$,

Như vậy chỉ cần lưu mảng tổng cặp giá trị $\mathbf{t}_i = \mathbf{x}_i + \mathbf{y}_i$, sắp xếp chúng và loại bỏ các giá trị lặp.

Việc loại bỏ giá trị lặp có thể thực hiện đơn giản hơn (trên quan điểm người lập trình) bằng cách sử dụng tập hợp,

Khi nạp vào tập hợp \mathbf{t}_i sẽ được ghi nhận nếu không trùng với giá trị nào đã có trong tập hợp,

Kết quả cần tìm sẽ là số lượng phần tử trong tập hợp.

Tổ chức dữ liệu: Tập hợp `set<int> s`.

Độ phức tạp của giải thuật: $O(n \ln n)$.

Chương trình:

```
#include <bits/stdc++.h>
#define NAME "exhibition."
using namespace std;
ifstream fi (NAME"inp");
ofstream fo (NAME"out");
int n,x,y,ans;
set<int> s;

int main()
{
    fi>>n;
    for(int i=0;i<n;++i)
    {
        fi>>x>>y; s.insert(x+y);
    }
    ans=s.size();
    fo<<ans;
    fo<<"\nTime: "<<clock() / (double)1000<<" sec";
}
```



VV04. KHU NGHỈ MÁT

Tên chương trình: RESORT.CPP

Khu nghỉ mát hoạt động liên tục n ngày, sau đó là một đợt nghỉ ngắn để duy tu, bảo dưỡng. Mỗi ngày làm việc ở khu nghỉ đều có một hoạt động độc đáo mang lại sự phấn khích lớn cho khách nghỉ. Chính vì vậy, đợt hoạt động mới chưa bắt đầu nhưng đã có q người đặt chỗ, người thứ j đặt chỗ từ ngày $1f$ đến hết ngày rh , $j = 1 \div q$.

Theo kế hoạch ban đầu, hoạt động tổ chức ngày thứ i sẽ mang lại hiệu quả hài lòng cho khách nghỉ là a_i . Với tình hình đặt chỗ cụ thể người ta quyết định thay đổi trình tự hoạt động để có tổng hiệu quả hài lòng hàng ngày của tất cả các người đặt chỗ là lớn nhất.

Hãy xác định tổng hiệu quả lớn nhất có thể đạt được.

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

- ✚ Dòng đầu tiên chứa 2 số nguyên n và q ($1 \leq n, q \leq 10^5$),
- ✚ Dòng thứ 2 chứa n số nguyên a_1, a_2, \dots, a_n ($1 \leq a_i \leq 10^8$, $i = 1 \div n$),
- ✚ Mỗi dòng trong q dòng tiếp theo chứa 2 số nguyên $1f$ và rh ($1 \leq 1f \leq rh \leq n$).

Kết quả: Đưa ra file văn bản RESORT.OUT một số nguyên – tổng hiệu quả lớn nhất có thể đạt được.

Ví dụ:

RESORT.INP
3 4
7 3 1
1 3
2 3
3 3
2 2

RESORT.OUT
31



VV04 Io20170305 B

Giải thuật: Tổng tiền tố.

Nhận xét:

Cần lập bảng thống kê mỗi ngày có bao nhiêu người tới nghỉ,
 Hoạt động mang lại hiệu quả lớn nhất sẽ tổ chức vào ngày đông khách nhất, hiệu
 quả hoạt động cần bố trí giảm dần theo số lượng khách,
 Vì chỉ cần tổng hiệu quả nên ta có thể sắp xếp 2 mảng (khách và hiệu quả) theo cùng
 một tiêu chí và tính tích vô hướng của 2 mảng.

Lưu ý: *Với các loại dữ liệu khác nhau hàm sort hoạt động theo những tiêu chuẩn
 ngầm định khác nhau!* Ví dụ với **vector**, nếu dữ liệu đã được sắp xếp theo một
 chiều nào đó thì hàm sort không thay đổi trình tự dữ liệu nếu không chỉ tường minh
 tiêu chuẩn sắp xếp.

Tổ chức dữ liệu:

- Mảng **int a[100001]** – lưu hiệu quả hoạt động,
- Mảng **int b[100003] = {0}** – lưu số lượng khách từng ngày.

Xử lý:

Đánh dấu ngày khách đến và ngày rời đi, từ đó tính tổng số lượng khách trong từng
 ngày theo quy tắc tích lũy tổng tiền tố:

Mảng b cần có kích thước lớn hơn n

```

for(int i=0;i<q;++i)
{
    fi>>lf>>rh; ++b[lf];--b[rh+1];
}
for(int i=1;i<=n;++i)b[i]+=b[i-1];

```

Độ phức tạp của giải thuật: $O(n \ln n)$.

Chương trình:

```
#include <bits/stdc++.h>
#define NAME "resort."
using namespace std;
ifstream fi (NAME"inp");
ofstream fo (NAME"out");
int n,q,lf,rh,t,b[100003]={0},a[100001];
int64_t ans=0;

int main()
{
    fi>>n>>q;
    a[0]=0;
    for(int i=1;i<=n;++i) fi>>a[i];
    for(int i=0;i<q;++i)
    {
        fi>>lf>>rh;++b[lf];--b[rh+1];
    }
    for(int i=1;i<=n;++i)b[i]+=b[i-1];

    sort(a,a+n+1);
    sort(b,b+n+1);
    for(int i=0;i<=n;++i)ans+=(int64_t)a[i]*b[i];
    fo<<ans;
    fo<<"\nTime: "<<clock() / (double)1000<<" sec";
}
```



Xét xâu s chỉ chứa các ký tự la tinh thường. Các 2 hàm xác định trên xâu này là G và F .

Xâu con của một xâu là xâu nhận được từ xâu ban đầu bằng cách xóa một số (có thể là 0) các ký tự đầu và cuối xâu. Tiền tố thứ i là xâu nhận được bằng cách chỉ xóa các ký tự cuối để nhận được xâu con độ dài i . Hậu tố thứ i là xâu nhận được bằng cách xóa đúng ($i-1$) ký tự đầu.

Với xâu s , G_i là độ dài hậu tố dài nhất của tiền tố thứ i , không trùng với tiền tố này và là tiền tố của xâu s . F_i là độ dài tiền tố dài nhất của hậu tố thứ i và là tiền tố của xâu s . Ngoài ra, có $G_1 = 0$ và $F_1 = 0$.

Mức độ độc đáo của xâu s được xác định như tổng các $G_i \times F_i$ với i chạy từ 1 đến độ dài của s .

Ví dụ, với $s = “aaaab”$ ta có $G = (0, 1, 2, 3)$, $F = (0, 3, 2, 1)$. Độ độc đáo của s sẽ là

$$1 \times 3 + 2 \times 2 + 3 \times 1 = 10$$

Cho k và m . Hãy xác định xâu có độ dài không quá m và có độ độc đáo bằng k . *Dữ liệu đảm bảo bài toán có lời giải.*

Dữ liệu: Vào từ file văn bản VARIETY.INP gồm một dòng chứa 2 số nguyên k và m ($1 \leq k \leq 10^{14}$, $1 \leq m \leq 10^5$).

Kết quả: Đưa ra file văn bản VARIETY.OUT một xâu tùy chọn tìm được.

Ví dụ:

VARIETY.INP	VARIETY.OUT
10 25	aaaab



Giải thuật: Tìm kiếm nhị phân.

Nhận xét:

Gọi $v(s)$ là hàm xác định độ độc đáo của s ,

Xét xâu s chỉ chứa một loại ký tự, ví dụ ‘**a**’,

Từ định nghĩa G_i và F_i dễ dàng thấy rằng trong số các xâu s cùng độ dài, hàm $v(s)$ sẽ *cho giá trị lớn nhất khi s chỉ chứa một loại ký tự*.

Ví dụ , với $s = \text{"aaaaa"}$ ta có:

i	<i>Prefix</i>	G_i	<i>Suffix</i>	F_i
1	a	0	aaaaa	0
2	aa	1	aaa	3
3	aaa	2	aa	2
4	aaaa	3	a	1

Gọi độ dài của s là ls ta có $G_i = i-1$, $F_i = ls-i+1$, $i = 2, 3, \dots, ls$.

Nếu $s = x + y$, trong đó:

- ✚ $x.size() \geq y.size()$,
- ✚ x và y chỉ chứa một loại ký tự (gọi là *ký tự chung*) trừ ký tự cuối cùng,
- ✚ Ký tự cuối cùng của x và y khác nhau và khác ký tự chung,

Xét $x = \text{"aaaab"}$, $y = \text{"aaac"}$ → $s = \text{"aaaabaaac"}$, ta có:

<i>Prefix</i>	G_i	<i>Suffix</i>	F_i
a	0	aaaabaaac	0
aa	1	aaabaaac	3
aaa	2	aabaaac	2
aaaa	3	abaaac	1
aaaab	0	baaac	0
aaaaba	1	aaac	3
aaaabaa	2	aac	2
aaaabaaa	3	ac	1
aaaabaaac	0	c	0

Bức tranh giá trị G_i và F_i được lặp như phần đầu của s và hàm $v(s)$ có tính chất *cộng tính*,

Vì bài toán được cho với *dữ liệu đảm bảo chắc chắn có nghiệm* nên ta chỉ cần tìm *xâu ngắn nhất* thỏa mãn, *bỏ qua dữ liệu m hạn chế độ dài!*

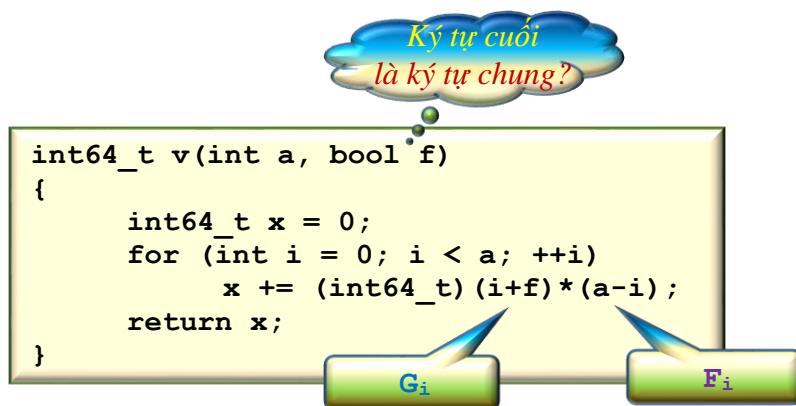
Xâu s cần tìm hoặc chỉ chứa một loại ký tự hoặc phải có dạng $s = z_1 + z_2 + \dots + z_k$, trong đó các xâu z_i có độ dài không tăng (khi i tăng) và có ký tự cuối khác nhau. Xâu cuối cùng có thể chỉ chứa các ký tự chung. Việc bổ sung thêm ký tự khác ký tự chung vào giữa xâu chỉ làm tăng độ dài của s và giảm giá trị $v(s)$.

Như vậy các xâu z_i có thể xác định theo phương pháp tìm kiếm nhị phân (vì độ dài không tăng).

Tổ chức dữ liệu: Dùng biến **char cc** quản lý ký tự cuối xâu cần tìm.

Xử lý:

Hàm **v** tính độ độc đáo của xâu theo độ dài và loại ký tự cuối:



Tìm kiếm nhị phân độ dài xâu chỉ chứa ký tự chung có độ độc đáo không vượt quá x :

```

int getCnt(int64_t x, bool f)
{
    int l = 1, r = 2000000;
    while (l + 1 < r)
    {
        int mid = (l + r) / 2;
        if (v(mid, f) <= x)
            l = mid;
        else
            r = mid;
    }
    return l;
}

```

Tìm kiếm và dẫn xuất xâu:

- ♣ Tìm độ dài xâu ngắn nhất chỉ chứa ký tự chung và có độ độc đáo $y \leq x$,
- ♣ Giảm độ độc đáo còn lại cần đảm bảo: $x -= y$;
- ♣ Đưa ra ký tự phân biệt cuối xâu và tìm tiếp nếu $x > 0$.

Lưu ý:

- Phải tính lại **y** trước khi cập nhật **x** theo độ dài tìm được,
- Các ký tự cuối cần khác ký tự cuối của xâu **z₁**,
- Bảng chữ cái đủ đảm bảo để các ký tự cuối là khác nhau.

```
char cc = 'b';
while (x != 0)
{
    int cnt = getCnt(x, cc != 'b');
    x -= v(cnt, cc != 'b');
    for (int i = 0; i < cnt; ++i) fo << 'a';
        if(x!=0) fo << cc;
        ++cc;
}
```

Độ phức tạp của giải thuật: O(lnk).

Chương trình 2:

```
#include <bits/stdc++.h>
#define NAME "variety."
using namespace std;
ifstream fi (NAME"inp");
ofstream fo (NAME"out");

int64_t v(int a, bool f)
{
    int64_t x = 0;
    for (int i = 0; i < a; ++i)
        x += (int64_t) (i+f)*(a-i);
    return x;
}

int getCnt(int64_t x, bool f)
{
    int l = 1, r = 2000000;
    while (l + 1 < r)
    {
        int mid = (l + r) / 2;
        if (v(mid, f) <= x)
            l = mid;
        else
            r = mid;
    }
    return l;
}

int main()
{
    int64_t x;
    fi >> x;
    char cc = 'b';
    while (x != 0)
    {
        int cnt = getCnt(x, cc != 'b');
        x -= v(cnt, cc != 'b');
        for (int i = 0; i < cnt; ++i) fo << 'a';
        if (x!=0) fo << cc;
        ++cc;
    }

    fo<<"\nTime: "<<clock() / (double)1000<<" sec";
}
```



Đĩa Petri là dụng cụ dùng để nuôi cấy vi khuẩn trong các thí nghiệm sinh học.

Giáo sư Braun tiến hành một thí nghiệm với n cá thể thuộc n loại vi khuẩn. Ông dùng 3 đĩa Petri trong quá trình thí nghiệm. Ban đầu, vi khuẩn thứ i được cấy ở đĩa Petri s_i , $i = 1 \div n$. Quá trình thí nghiệm bao gồm q bước, ở bước thứ j ông tách một vi khuẩn từ đĩa x_j và cấy sang đĩa y_j . Kết thúc thí nghiệm vi khuẩn thứ i nằm ở đĩa Petri t_i , $i = 1 \div n$.



Để chuẩn bị báo cáo khoa học, Giáo sư rà soát lại toàn bộ tài liệu về quá trình thí nghiệm và phát hiện ra người trợ lý của ông không ghi lại ở mỗi bước vi khuẩn nào đã được chuyển từ đĩa này sang đĩa khác. Sự tắc trách này làm dấy lên nghi ngờ liệu có thật cuối cùng vi khuẩn thứ i nằm ở đĩa Petri t_i ($i = 1 \div n$) hay không.

Hãy kiểm tra và đưa ra thông báo **YES** nếu kết quả cuối cùng về vị trí vi khuẩn là có thể và **NO** trong trường hợp ngược lại.

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

- + Dòng đầu tiên chứa 2 số nguyên n và q ($1 \leq n \leq 50, 0 \leq q \leq 100$),
- + Dòng thứ 2 chứa n số nguyên s_1, s_2, \dots, s_n ($1 \leq s_i \leq 3, i = 1 \div n$),
- + Dòng thứ 3 chứa n số nguyên t_1, t_2, \dots, t_n ($1 \leq t_i \leq 3, i = 1 \div n$),
- + Dòng thứ j trong q dòng tiếp theo chứa 2 số nguyên x_j và y_j ($1 \leq x_j, y_j \leq 3, x_j \neq y_j$).

Kết quả: Đưa ra file văn bản PETRI.OUT thông báo nhận được.

Ví dụ:

PETRI.INP
3 2
1 2 2
3 1 2
2 1
1 3

PETRI.OUT
YES



Giải thuật: Mô phỏng ô tô mát, Loang theo chiều rộng, Nén dữ liệu.

Nhận xét:

Từ vị trí ban đầu và kết thúc của mỗi vi khuẩn ta có thể xác định bảng kết quả cuối cùng chuyển vị trí của nó: **bảng trạng thái**,

Sau mỗi bước di chuyển đã cho một số vi khuẩn có thể xuất hiện ở những vị trí tiềm năn mới, tương ứng với các **trạng thái tiềm năng**, được ghi nhận bởi các bảng trạng thái tương ứng trong **tập trạng thái**,

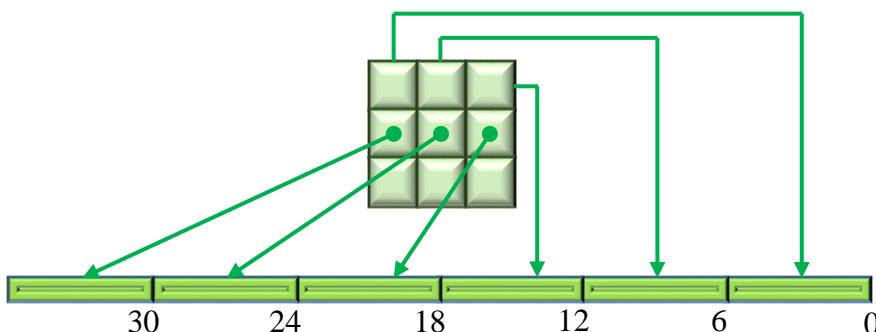
Từ một tập trạng thái, ta có thể xây dựng tập trạng thái mới theo phép chuyển đã cho,

Sau khi xây dựng tập trạng thái cho phép chuyển cuối cùng, chỉ việc kiểm tra trạng thái cuối đã cho có trong tập trạng thái hay không và đưa ra kết luận tương ứng,

Bảng trạng thái được tổ chức dưới dạng mảng **int a[3][3]**, trong đó $a_{i,j}$ – số lượng vi khuẩn nhận được trong đĩa i bởi phép chuyển vi khuẩn **giữa 2 đĩa i và j**,

Khi biết giá trị 2 dòng ta có thể xác định một cách đơn trị dòng còn lại, vì vậy có thể lấy 2 dòng đầu làm khóa phân biệt bảng trạng thái,

Giá trị n không quá 50, $a_{i,j} \leq n$, vì vậy chỉ cần dùng 6 bít để biểu diễn và lưu 6 biến ở 2 dòng đầu tiên dưới dạng số nguyên 36 bít.



Tổ chức dữ liệu:

- Các mảng **int w1[N], w2[N]** lưu dữ liệu về vi khuẩn,
- Mảng **int as[3]** lưu số lượng vi khuẩn trên mỗi đĩa petri,
- Các mảng **int a[3][3], b[3][3]** lưu trạng thái đĩa petri,
- Các tập **set < int64_t >d, nd** quản lý trạng thái phục vụ loang.

Xử lý:

Hàm tạo khóa:

Truyền tham số
theo địa chỉ

```
ll encode(int * a1, int * a2)
{
    return a1[0] | ((ll)a1[1] << 6) | ((ll)a1[2] << 12)
        | ((ll)a2[0] << 18) | ((ll)a2[1] << 24) | ((ll)a2[2] << 30);
}
```

Giải nén, xác định trạng thái:

Tham số ra phải truyền
theo địa chỉ

```
void decode(ll x, int * a1, int * a2, int * a3)
{
    a1[0] = x & 63;
    a1[1] = (x >> 6) & 63;
    a1[2] = (x >> 12) & 63;
    a2[0] = (x >> 18) & 63;
    a2[1] = (x >> 24) & 63;
    a2[2] = (x >> 30) & 63;
    a3[0] = as[0] - a1[0] - a2[0];
    a3[1] = as[1] - a1[1] - a2[1];
    a3[2] = as[2] - a1[2] - a2[2];
}
```

Ghi nhận trạng thái ban đầu:

Khởi tạo trạng
thái đầu

Khởi tạo tập
trạng thái

```
fi >> n >> q;
for (int i = 0; i < n; ++i)
{
    fi >> w1[i]; --w1[i];
}
for (int i = 0; i < n; ++i)
{
    fi >> w2[i]; --w2[i];
}
for (int i = 0; i < n; ++i)
{
    ++a[w1[i]][w2[i]];
    ++as[w2[i]];
}
set < ll > d;
d.insert(encoder(a[0], a[1]));
```

Tổ chức loang:

```
for (int i = 0; i < q; ++i)
{
    int x1, x2;
    fi >> x1 >> x2;
    --x1;
    --x2;
    set < ll > nd;
    for (ll pp : d)
    {
        int b[3][3];
        decode(pp, b[0], b[1], b[2]);
        for (int z = 0; z < 3; ++z)
            if (b[x1][z])
            {
                --b[x1][z]; ++b[x2][z];
                nd.insert(encode(b[0], b[1]));
                ++b[x1][z]; --b[x2][z];
            }
        d = nd;
    }
}
```

Trạng thái xuất phát

Trạng thái mới

Khôi phục trạng thái xuất phát

Đổi vai trò tập mới/cũ

Kiểm tra, ghi nhận kết quả:

Tạo khóa ứng với trạng thái kết thúc đã cho và kiểm tra sự có mặt của khóa đó trong tập các trạng thái cuối cùng có thể có.

```
int fa1[] = {as[0], 0, 0};
int fa2[] = {0, as[1], 0};
if (d.count(encode(fa1, fa2)))
    fo << "YES\n";
else
    fo << "NO\n";
```

Độ phức tạp của giải thuật: $\approx O(q^2)$.

Chương trình:

```
#include <bits/stdc++.h>
#define NAME "petri."
using namespace std;
typedef int64_t ll;
ifstream fi (NAME"inp");
ofstream fo (NAME"out");
const int N = 55;

int n, q;
int w1[N], w2[N];
int a[3][3], as[3];

ll encode(int * a1, int * a2)
{
    return a1[0] | ((ll)a1[1] << 6) | ((ll)a1[2] << 12)
        | ((ll)a2[0] << 18) | ((ll)a2[1] << 24) | ((ll)a2[2] << 30);
}

void decode(ll x, int * a1, int * a2, int * a3)
{
    a1[0] = x & 63;
    a1[1] = (x >> 6) & 63;
    a1[2] = (x >> 12) & 63;
    a2[0] = (x >> 18) & 63;
    a2[1] = (x >> 24) & 63;
    a2[2] = (x >> 30) & 63;
    a3[0] = as[0] - a1[0] - a2[0];
    a3[1] = as[1] - a1[1] - a2[1];
    a3[2] = as[2] - a1[2] - a2[2];
}

int main()
{
    fi >> n >> q;
    for (int i = 0; i < n; ++i)
    {
        fi >> w1[i];
        --w1[i];
    }
    for (int i = 0; i < n; ++i)
    {
        fi >> w2[i];
        --w2[i];
    }
    for (int i = 0; i < n; ++i)
    {
        ++a[w1[i]][w2[i]];
        ++as[w2[i]];
    }

    set<ll> d;
    d.insert(encode(a[0], a[1]));

    for (int i = 0; i < q; ++i)
    {
```

```

int x1, x2;
fi >> x1 >> x2;
--x1;
--x2;
set < ll > nd;
for (ll pp : d)
{
    int b[3][3];
    decode(pp, b[0], b[1], b[2]);
    for (int z = 0; z < 3; ++z)
        if (b[x1][z])
        {
            --b[x1][z]; ++b[x2][z];
            nd.insert(encode(b[0], b[1]));
            ++b[x1][z]; --b[x2][z];
        }
    d = nd;
}

int fa1[] = {as[0], 0, 0};
int fa2[] = {0, as[1], 0};
if (d.count(encode(fa1, fa2)))
    fo << "YES\n";
else
    fo << "NO\n";
fo<<"\nTime: "<<clock() / (double)1000<<" sec";
return 0;
}

```



VV08. CHUYỂN PHÁT HÀNG

Tên chương trình: DELIVERY.CPP

Trạm chuyển phát hàng ở bưu điện cơ sở nhận các gói hàng của người gửi, mỗi gói có trọng lượng nguyên bất kỳ trong phạm vi từ 1 kg và cho đến k kg.

Các gói hàng được bỏ vào thùng theo trình tự nhận được. Khi tổng trọng lượng hàng trong thùng bằng hoặc lớn hơn x kg, thùng sẽ được niêm phong và chuyển lên bưu điện trung tâm. Ở đó thùng hàng sẽ được xếp vào container đặc biệt. Khi tổng trọng lượng hàng trong container bằng hoặc vượt quá y kg, container sẽ được khóa lại và chuyển trung tâm phân loại để chuyển phát tới người nhận.

Tổng trọng lượng hàng trong các containers có thể khác nhau phụ thuộc vào trình tự và trọng lượng hàng tới gửi ở bưu điện cơ sở.

Hãy xác định trọng lượng nhỏ nhất có thể có của container.

Dữ liệu: Vào từ file văn bản DELIVERY.INP gồm 3 số nguyên k , x và y , mỗi số cho trên một dòng ($1 \leq k, x, y \leq 10^9$).

Kết quả: Đưa ra file văn bản DELIVERY.OUT một số nguyên – trọng lượng nhỏ nhất tìm được.

Ví dụ:

DELIVERY.INP
2
7
20

DELIVERY.OUT
21



VV08 Reg20170204 5

Giải thuật: Phân tích, đoán nhận giải thuật.

Nhận xét:

Thùng hàng từ cơ sở chuyển lên có thể có trọng lượng trong phạm vi từ x đến $x+k-1$ (ví dụ người ta có thể nhận được $x-1$ gói bưu phẩm mỗi gói 1 kg và sau đó – nhận được gói trọng lượng k kg), mọi giá trị trong phạm vi từ x đến $x+k-1$ đều có thể đạt được,

Bưu điện cơ sở có thể có *tối đa* $z = y/x$ lần chuyển hàng lên bưu điện trung tâm *trước khi* container được chuyển đi,

Container sau z lần chuyển hàng có trọng lượng tối thiểu là zx và tối đa là $zx(x+k-1)$, mọi giá trị giữa trọng lượng tối thiểu và tối đa đều có thể đạt được,

Nếu y nằm trong trong phạm vi giá trị nêu trên thì y là kết quả cần tìm, trong trường hợp ngược lại – cần thêm một kiện hàng nữa với trọng lượng nhỏ nhất có thể, tức là x . Trọng lượng tối thiểu của container sẽ là $zx(z+1)$.

Độ phức tạp của giải thuật: O(1).

Chương trình:

```
#include <bits/stdc++.h>
#define NAME "delivery."
using namespace std;
ifstream fi (NAME"inp");
ofstream fo (NAME"out");
int64_t k,x,y,ans;

int main()
{
    fi >> k >> x >> y;
    if (y % x == 0 || (y / x) * (k - 1) >= y % x) ans = y;
    else
        ans = x * (y / x + 1);
    fo << ans ;
    return 0;
}
```



VV09. MÁY GIA TỐC HẠT

Tên chương trình: COLLIDER.CPP

Để nghiên cứu về sự tương tác giữa vật chất và phản vật chất tại mỗi vị trí trong n vị trí ở đường ống của máy gia tốc người ta kích cho xuất hiện một electron hoặc positron (phản hạt electron): ở vị trí \mathbf{x}_i kích hoạt cho xuất hiện hạt e_i , $e_i = 1$ là electron, $e_i = -1$ là positron. Ông gia tốc có thể coi như một đường thẳng. Electron có điện tích dương và sau khi được kích hoạt sẽ chuyển động sang phải, tức là về phía tọa độ tăng dần. Positron có điện tích âm và chuyển động sang trái, tức là về phía tọa độ giảm dần. Các hạt chuyển động với tốc độ không đổi: 1 đơn vị độ dài trong một đơn vị thời gian. Khi hạt và phản hạt gặp nhau, một vụ nổ nhỏ xảy ra và hai hạt bị tiêu hủy gần như tức thời.

Các nhà khoa học muốn biết ở cuối mỗi thời điểm trong số các thời điểm t_1, t_2, \dots, t_m trong máy còn lại bao nhiêu hạt cơ bản. Nếu ở một thời điểm t_i hạt và phản hạt gặp nhau chúng sẽ bị tiêu diệt và cuối thời điểm t_i – không còn hai hạt đó.

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

- ✚ Dòng đầu tiên chứa một số nguyên n ($1 \leq n \leq 2 \times 10^5$),
- ✚ Dòng thứ i trong n dòng sau chứa 2 số nguyên \mathbf{x}_i và e_i ($-10^9 \leq \mathbf{x}_i \leq 10^9$, $\mathbf{x}_i < \mathbf{x}_j$ với $i < j$),
- ✚ Dòng tiếp theo chứa số nguyên m ($1 \leq m \leq 2 \times 10^5$),
- ✚ Dòng cuối cùng chứa m số nguyên t_1, t_2, \dots, t_m ($0 \leq t_j \leq 10^9$, $t_i < t_j$ với $i < j$).

Kết quả: Đưa ra file văn bản COLLIDER.OUT m số nguyên, xác định số lượng hạt còn lại cuối mỗi thời điểm đã cho, mỗi số trên một dòng.

Ví dụ:

COLLIDER.INP
4
-1 1
0 -1
1 1
5 -1
4
0 1 2 3

COLLIDER.OUT
4
2
0
0



Giải thuật: Kỹ thuật tổ chức dữ liệu, tổng tiền tố.

Nhận xét:

Nếu giữa electron (hạt) ở vị trí \mathbf{u} và positron (phản hạt) ở vị trí $\mathbf{v} > \mathbf{u}$ không có hạt nào khác thì cặp hạt này sẽ triệt tiêu lẫn nhau ở vị trí điểm giữa đoạn $[\mathbf{u}, \mathbf{v}]$,

Ta chỉ quan tâm tới các thời điểm nguyên, vì vậy chắc chắn ở cuối thời điểm $\mathbf{dt} = (\mathbf{v}-\mathbf{u}+1)/2$ chắc chắn cặp hạt này không tồn tại, tổng số lượng hạt giảm 2,

Tổ chức một stack \mathbf{s} lưu vị trí các electron chưa bị triệt tiêu với con trỏ \mathbf{p} chỉ tới đỉnh stack,

Khi gấp hạt với $\mathbf{e} = 1 \rightarrow$ nạp vị trí vào stack,

Khi gấp hạt với $\mathbf{e} = -1 \rightarrow$ ta có phản hạt, nếu stack rỗng phản hạt sẽ không bị triệt tiêu, trong trường hợp ngược lại – phản hạt này bị triệt tiêu bởi hạt có vị trí ghi nhận ở đầu stack, ghi nhận thời điểm triệt tiêu và cập nhật số lượng hạt bị triệt tiêu ở thời điểm đó,

Tạo mảng \mathbf{vdt} ghi nhận *theo thứ tự tăng dần các thời điểm xảy ra triệt tiêu hạt* và mảng \mathbf{fdt} ghi nhận *số hạt bị triệt tiêu* ở thời điểm tương ứng,

Từ \mathbf{fdt} xây dựng mảng tổng tiền tố tích lũy số hạt đã bị triệt tiêu,

Xác định thời điểm \mathbf{mxdt} sau đó không xảy ra việc triệt tiêu hạt và số lượng hạt các loại còn lại \mathbf{r} .

Với mỗi thời điểm \mathbf{ti} cần xét:

- ♣ Tìm \mathbf{kt} nhỏ nhất thỏa mãn điều kiện $\mathbf{ti} \leq \mathbf{vdt}_{\mathbf{kt}}$,
- ♣ Đưa ra số hạt còn lại là $\mathbf{n} - \mathbf{fdt}_{\mathbf{kt}}$,

Với $\mathbf{ti} < \mathbf{vdt}_0$ số hạt còn lại là \mathbf{n} ,

Với $\mathbf{ti} \geq \mathbf{mxdt}$ số hạt còn lại là \mathbf{r} .

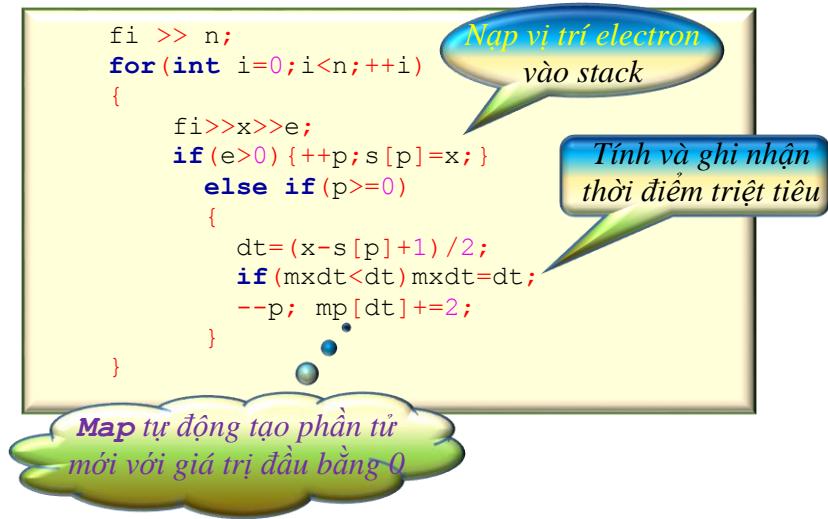
Lưu ý: \mathbf{kt} không giảm với \mathbf{ti} mới.

Tổ chức dữ liệu:

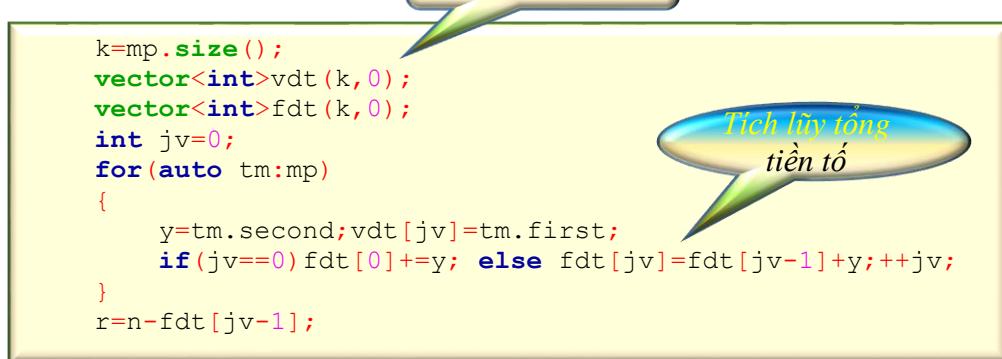
- ❑ Mảng `int s[200002]` đóng vai trò stack lưu các \mathbf{x}_i có $\mathbf{e}_i = 1$,
- ❑ Bản đồ `map<int, int> mp` tích lũy số lượng cặp hạt bị triệt tiêu ở mỗi thời điểm,
- ❑ Mảng `vector<int> vdt(k, 0)` lưu các thời điểm có triệt tiêu hạt theo trình tự thời gian tăng dần,
- ❑ Mảng `vector<int> fdt(k, 0)` lưu tổng tiền tố số lượng hạt bị triệt tiêu theo trình tự thời gian tăng dần.

Xử lý:

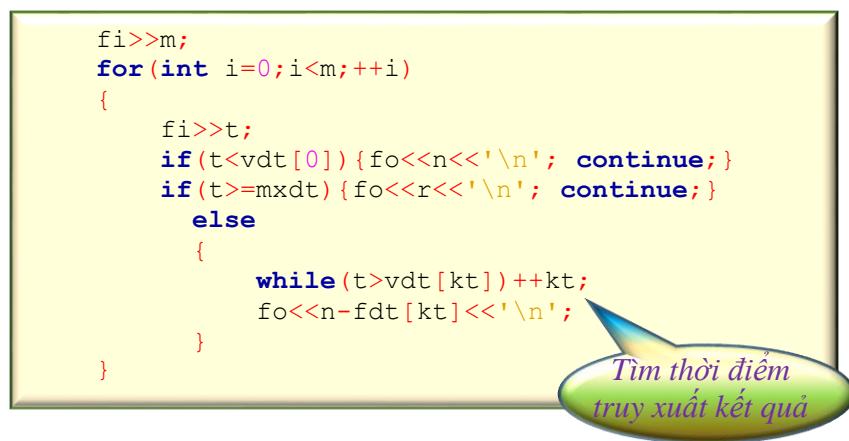
Ghi nhận dữ liệu về các hạt và kết quả tương tác giữa chúng:



Tính tổng tiền tố:



Xử lý truy vấn:



Độ phức tạp của giải thuật: O(nlogn).

Chương trình:

```
#include <bits/stdc++.h>
#define NAME "collider."
using namespace std;
ifstream fi (NAME"inp");
ofstream fo (NAME"out");
int n,m,k,kt=0,x,y,e,t;
int dt,p=-1,mxdt=-1,r;
int s[200002];
map<int,int>mp;

int main()
{
    fi >> n;
    for(int i=0;i<n;++i)
    {
        fi>>x>>e;
        if(e>0) {++p;s[p]=x;}
        else if(p>=0)
        {
            dt=(x-s[p]+1)/2;
            if(mxdt<dt) mxdt=dt;
            --p; mp[dt]+=2;
        }
    }
    k=mp.size();
    vector<int>vdt(k,0);
    vector<int>fdt(k,0);
    int jv=0;
    for(auto tm:mp)
    {
        y=tm.second; vdt[jv]=tm.first;
        if(jv==0) fdt[0]+=y; else fdt[jv]=fdt[jv-1]+y; ++jv;
    }
    r=n-fdt[jv-1];
    fi>>m;
    for(int i=0;i<m;++i)
    {
        fi>>t;
        if(t<vdt[0]) {fo<<n<<'\\n'; continue;}
        if(t>=mxdt) {fo<<r<<'\\n'; continue;}
        else
        {
            while(t>vdt[kt]) ++kt;
            fo<<n-fdt[kt]<<'\\n';
        }
    }
    fo <<"\\n*** Time: "<<clock() / (double)1000<<" sec" ;
    return 0;
}
```



VV10. TRƯỜNG NĂNG LƯỢNG

Tên chương trình: POWER.CPP

Để có thể canh tác trên sao Hỏa người ta cần che chắn bức xạ cực tím của mặt trời lên diện tích trồng trọt.

Các thiết bị tạo lá chắn được thử nghiệm trên cánh đồng hình chữ nhật kích thước $10^9 \times 10^9$. Máy tạo lá chắn được đặt ở điểm có tọa độ $(0, 0)$ và có khả năng tạo n dạng lá chắn khác nhau, mỗi lá chắn đều có hình chữ nhật cạnh song song với cạnh của cánh đồng, góc dưới trái của lá chắn ở điểm tọa độ $(0, 0)$, lá chắn thứ i có tọa độ đính trên phải là (x_i, y_i) , $i = 1 \div n$.

Do hạn chế về công suất, người ta chỉ có thể kích hoạt đồng thời k lá chắn.

Hãy xác định diện tích lớn nhất có thể được đồng thời bảo vệ bởi k lá chắn.

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

- ✚ Dòng đầu tiên chứa 2 số nguyên n và k ($1 \leq k \leq n \leq 2 \times 10^5$),
- ✚ Dòng thứ i trong n dòng sau chứa 2 số nguyên x_i và y_i ($1 \leq x_i, y_i \leq 10^9$).

Kết quả: Đưa ra file văn bản POWER.OUT một số nguyên – diện tích lớn nhất được đồng thời bảo vệ bởi k lá chắn.

Ví dụ:

POWER.INP	POWER.OUT
5 3	
3 5	
2 2	
2 5	
4 4	
5 3	9

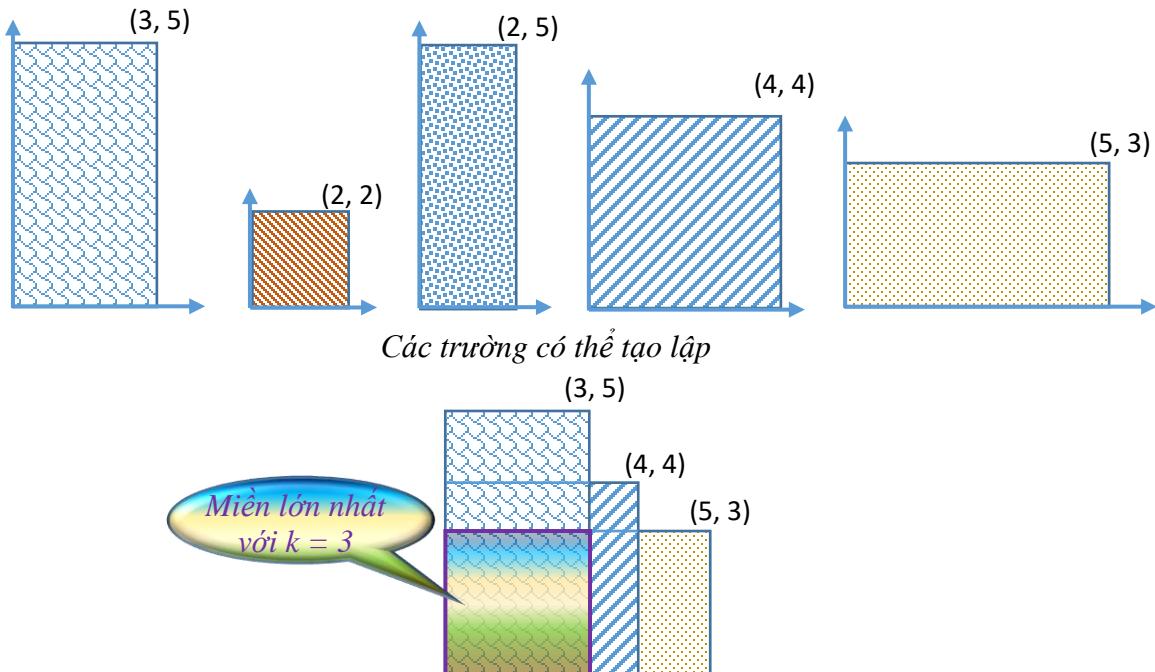


VV10 Reg20170204 7

Giải thuật: Nguyên lý cực trị.

Nhận xét:

Hình minh họa ví dụ đầu bài:



Sắp xếp các hình theo thứ tự giảm dần của \mathbf{x}_i , $i = 1 \div n$,

Chọn k hình đầu tiên (có các \mathbf{x}_i lớn nhất),

Nạp \mathbf{y}_i của các hình này vào hàng đợi ưu tiên theo *trình tự tăng dần* \mathbf{q} ,

Tính diện tích được bảo vệ bởi k trường: $\mathbf{best} = \mathbf{q}.\text{top}() * \mathbf{a}[k-1].\text{first}$,

Duyệt với mọi hình còn lại $i = k \div n - 1$:

- Nạp \mathbf{y}_i vào \mathbf{q} ,
- Loại bỏ phần tử đầu của \mathbf{q} ,
- Tính lại diện tích và cập nhật \mathbf{best} .

Tổ chức dữ liệu:

Mảng cấp dữ liệu lưu kích thước các lá chắn: `vector<pii> a(n)`,

Hàng đợi ưu tiên giảm dần lưu chiều cao y_i :

```
priority_queue<int, vector<int>, greater<int>> q
```

Độ phức tạp của giải thuật: $O(n \log n)$.

Chương trình:

```
#include <bits/stdc++.h>
#define NAME "power."
using namespace std;
typedef pair<int,int> pii;
ifstream fi(NAME"inp");
ofstream fo(NAME"out");

int main()
{
    int n, k;
    fi >> n >> k;
    vector<pii> a(n);
    for (int i = 0; i < n; i++)
        fi >> a[i].first >> a[i].second;
    sort(a.begin(), a.end(), greater<pii>());
    priority_queue<int, vector<int>, greater<int>> q;
    for (int i = 0; i < k; i++)
        q.push(a[i].second);
    int64_t best = q.top() * (int64_t)a[k - 1].first;
    for (int i = k; i < n; i++)
    {
        q.push(a[i].second);
        q.pop();
        best = max(best, a[i].first * (int64_t)q.top());
    }
    fo << best << "\n";
    fo <<"\nTime: "<<clock() / (double)1000<<" sec" ;
}
```

Một cách tổ chức dữ liệu khác: Sử dụng cấu trúc dữ liệu tuple (chuỗi) tổng quát hơn pair.

```
#include <bits/stdc++.h>
#define NAME "power."
using namespace std;
typedef pair<int,int> pii;
ifstream fi(NAME"inp");
ofstream fo(NAME"out");

int main()
{
    int n, k;
    fi >> n >> k;
    vector<tuple<int, int>> a(n);
    for (int i = 0; i < n; i++)
        fi >> get<0>(a[i]) >> get<1>(a[i]);
    sort(a.begin(), a.end(), greater<tuple<int, int>>());
    priority_queue<int, vector<int>, greater<int>> q;
    for (int i = 0; i < k; i++)
        q.push(get<1>(a[i]));
    int64_t best = q.top() * (int64_t) get<0>(a[k - 1]);
    for (int i = k; i < n; i++)
    {
        q.push(get<1>(a[i]));
        q.pop();
        best = max(best, get<0>(a[i]) * (int64_t) q.top());
    }
    fo << best << "\n";
    fo <<"\nTime: "<<clock() / (double)1000<<" sec" ;
}
```



VV11. BỘI DƯỠNG NÂNG CAO

Tên chương trình: TRAINING.CPP

Một công ty tổ chức quản lý nhân viên theo quan hệ phân cấp. Công ty có n người, đánh số từ 1 đến n . Giám đốc công ty được đánh số là 1 và không có thủ trưởng. Với những nhân viên còn lại, mỗi người có đúng một thủ trưởng trực tiếp và số thứ tự của thủ trưởng nhỏ hơn số thứ tự của nhân viên, tức là nếu nhân viên i có thủ trưởng trực tiếp là p_i , thì $p_i < i$.

Người x được gọi là nhân viên mức 1 của y nếu $p_x = y$. Với $k > 1$, người x được gọi là nhân viên mức k của y nếu p_x là nhân viên mức $k-1$ của y .

Có một khóa ngắn hạn nâng cao trình độ nghiệp vụ được tổ chức. Giám đốc quyết định sẽ cử những người có số thứ tự i trong phạm vi từ L đến R (tức là $L \leq i \leq R$) đi học. Nhiều người muốn nhân viên của mình được đi học nên kết quả là Giám đốc nhận được m đơn đề nghị, trong đó đơn thứ j của nhân viên u_j xin cử một nhân viên mức k_j của mình đi học.

Hãy xác định L và R sao cho số người được cử đi học là ít nhất nhưng mọi đơn đề nghị đều được đáp ứng.

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

- ⊕ Dòng đầu tiên chứa một số nguyên n ($2 \leq n \leq 2 \times 10^5$),
- ⊕ Dòng thứ 2 chứa $n-1$ số nguyên $p_2, p_3, p_4, \dots, p_n$ ($1 \leq p_i < i$, $i = 2 \div n$),
- ⊕ Dòng thứ 3 chứa số nguyên m ($1 \leq m \leq 2 \times 10^5$),
- ⊕ Dòng thứ j trong m dòng sau chứa 2 số nguyên u_j và k_j ($1 \leq u_j < n$, $1 \leq k_j < n$), dữ liệu đảm bảo tồn tại người ở mức đã nêu.

Kết quả: Đưa ra file văn bản TRAINING.OUT trên một dòng hai số nguyên L và R tìm được.

Ví dụ:

TRAINING.INP
7
1 1 2 2 3 3
3
1 1
3 1
1 2

TRAINING.OUT
3 6



VV11 Reg20170204 8

Giải thuật: Đồ thị cây, tổ chức dữ liệu.

Nhận xét:

Định hướng giải thuật:

- ☒ Với mỗi yêu cầu (\mathbf{p}_i , \mathbf{k}_i) xây dựng danh sách \mathbf{L}_i chứa các đỉnh là đỉnh con bậc \mathbf{k}_i của \mathbf{p}_i ,
- ☒ Với mỗi \mathbf{L}_i tìm một đại diện \mathbf{x}_i sao cho $\max\{\mathbf{x}_i\} - \min\{\mathbf{x}_i\}$ là nhỏ nhất.

Để giải quyết một cách hiệu quả các vấn đề trên ta cần:

- ✓ Tìm thời điểm vào (*timeIn*) và thời điểm ra (*timeOut*) của mỗi đỉnh,
- ✓ Dựa vào đó tạo các danh sách chứa đỉnh cùng độ sâu (*verticesByDepth*),
- ✓ Với mỗi i , \mathbf{L}_i (*levelSegments*) sẽ chứa thông tin về đoạn các đỉnh liên tục cùng độ sâu,
- ✓ Các \mathbf{L}_i hoặc không giao nhau hoặc chứa nhau,
- ✓ Nếu danh sách **A** chứa danh sách **B** thì ta có thể loại bỏ danh sách **A**, một đỉnh thuộc **B**, đại diện cho **B** thì đồng thời cũng đại diện cho **A**,
- ✓ Với tất cả các tập lồng nhau: chỉ giữ lại tập ở mức lồng sâu nhất, *số lượng tập cần xử lý có thể giảm* (khi có tập bị loại),
- ✓ Xác định \mathbf{mx}_j – đỉnh lớn nhất trong các \mathbf{L}_j ,
- ✓ Tính $\mathbf{mn}_j = \min\{\mathbf{mx}_j\}$,
- ✓ Rõ ràng, không thể chọn đỉnh \mathbf{x} lớn hơn \mathbf{mn}_j (sẽ thiếu đại diện), ngược lại khi $\mathbf{x} \leq \mathbf{mn}_j$ thì \mathbf{x} sẽ đại diện cho tất cả các \mathbf{L}_j ,
- ✓ Tương tự như vậy, tìm giới hạn trên của các đỉnh cần chọn và dẫn xuất kết quả.

Tổng số đỉnh trong các tập \mathbf{L}_j không quá n vì vậy độ phức tạp của giải thuật sẽ không lớn (sẽ đánh giá sau khi xét các công đoạn xử lý cụ thể).

Tổ chức dữ liệu:

Mảng `vector<int>` `tree[N]` – lưu cấu trúc cây ban đầu,

Mảng `vector<int>` `verticesByDepth[N]` – lưu độ sâu của các đỉnh,

Mảng `vector<segment>` `levelSegments[N]` – lưu danh khoảng đỉnh cùng độ sâu, trong đó segment – cặp dữ liệu kiểu `<int, int>`

Các mảng `int timeIn[N], timeOut[N], depth[N]` – lưu thời điểm vào, ra và độ sâu của mỗi nút.

Ngoài ra còn có một số cấu trúc dữ liệu hỗ trợ cục bộ hóa trong các hàm.

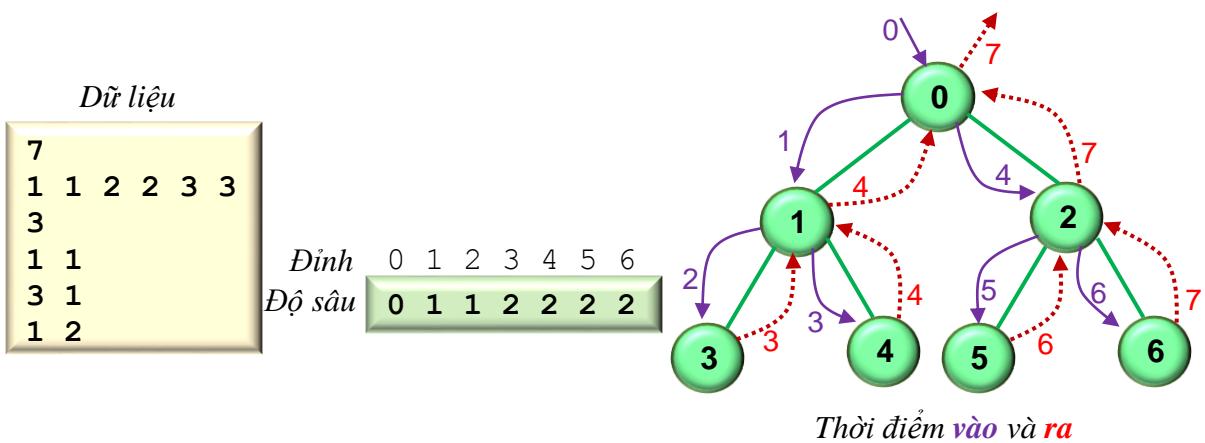
Xử lý:

Ghi nhận cấu trúc cây, tính thời điểm vào/ra và độ sâu của các đỉnh:

```
int dfsTime = 0;
int timeIn[N], timeOut[N], depth[N];

void dfs(int u, int d)
{
    timeIn[u] = dfsTime++;
    depth[u] = d;
    verticesByDepth[d].push_back(u);
    for (int t=0; t<tree[u].size(); t++) dfs(tree[u][t], d + 1);
    timeOut[u] = dfsTime;
}

void initializeTree()
{
    for (int i = 1; i < n; i++)
    {
        int parent;
        fi>>parent;
        tree[parent - 1].push_back(i);
    }
    dfs(0, 0);
}
```



Ghi nhận yêu cầu và xác định đoạn các đỉnh liên tục cùng độ sâu:

```
fi>>m;
for (int i = 0; i < m; i++)
{
    int u, k;
    fi>>u>>k;
    u--;
    levelSegments[depth[u]+k].push_back
        (findSegment(depth[u]+k, timeIn[u], timeOut[u]));
}
```

Hàm **levelSegments()** xác định điểm đầu (tính từ 0), điểm cuối của đoạn các điểm cùng độ sâu và là các đỉnh con độ sâu **k** của đỉnh **u**:

Việc xác định được thực hiện theo phương pháp tìm kiếm nhị phân, tìm **đỉnh lớn nhất** có **thời gian vào lớn hơn** thời gian vào của **u** và **đỉnh nhỏ nhất** có **thời gian ra nhỏ hơn** hoặc bằng thời gian ra của **u** trong số các đỉnh cùng độ sâu **k** tính từ **u**:

```

segment findSegment (int depth, int tIn, int tOut)
{
    int from, to;
    int left = -1, right = (int) verticesByDepth[depth].size();
    while (left < right - 1)
    {
        int mid = (left + right) / 2;
        if (timeIn[verticesByDepth[depth][mid]] >= tIn) right = mid;
        else left = mid;
    }
    from = right;

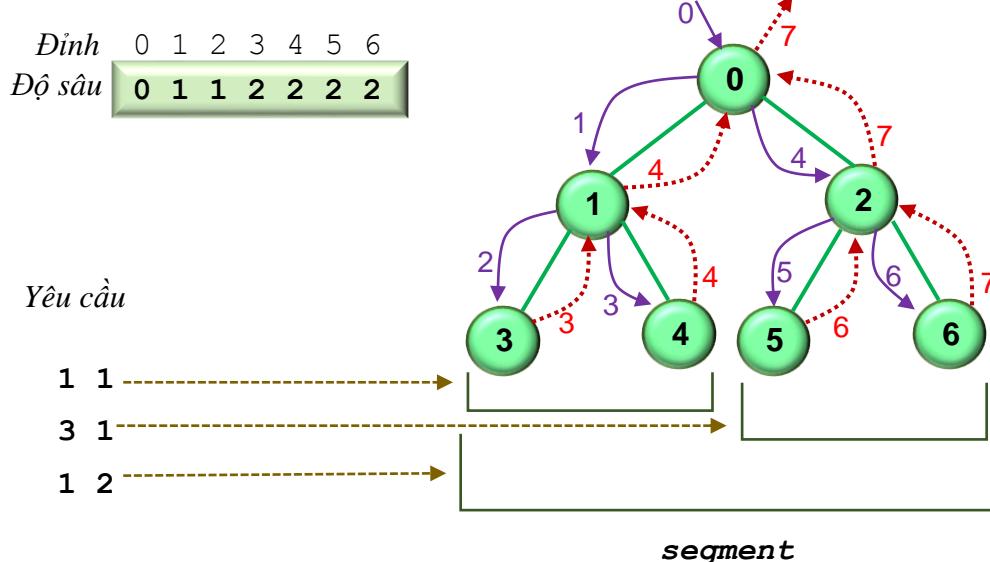
    left = -1; right = (int) verticesByDepth[depth].size();
    while (left < right - 1)
    {
        int mid = (left + right) / 2;
        if (timeOut[verticesByDepth[depth][mid]] <= tOut) left = mid;
        else right = mid;
    }
    to = left;

    return segment (from, to);
}

```

Tìm đầu danh sách

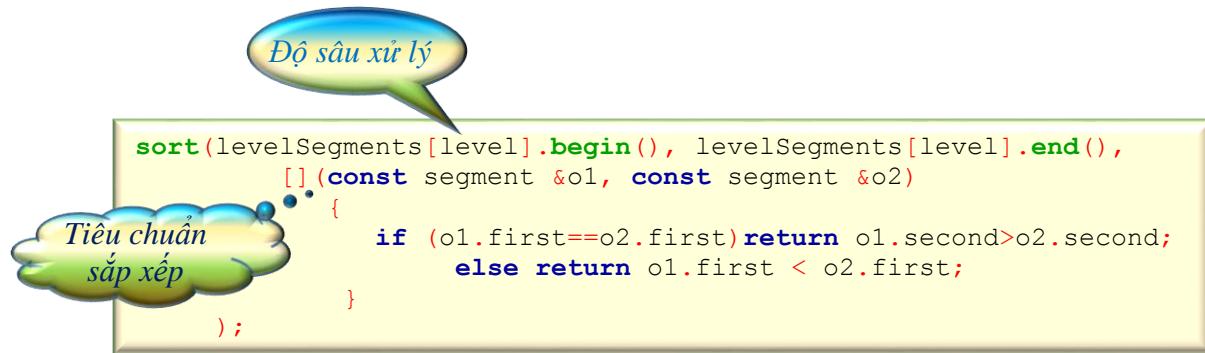
Tìm cuối danh sách



Loại bỏ các đoạn có chứa các đoạn khác trong các đoạn đã xác định:

Các đoạn cần xử lý hoặc rời nhau hoặc chứa trong nhau,

Với mỗi độ sâu, sắp xếp các đoạn theo thứ tự tăng dần của điểm đầu, nếu 2 đoạn chứa nhau thì đoạn lớn hơn đứng trước:



Hàm loại bỏ các đoạn bao đoạn khác:

Duyệt các độ sâu
có thể

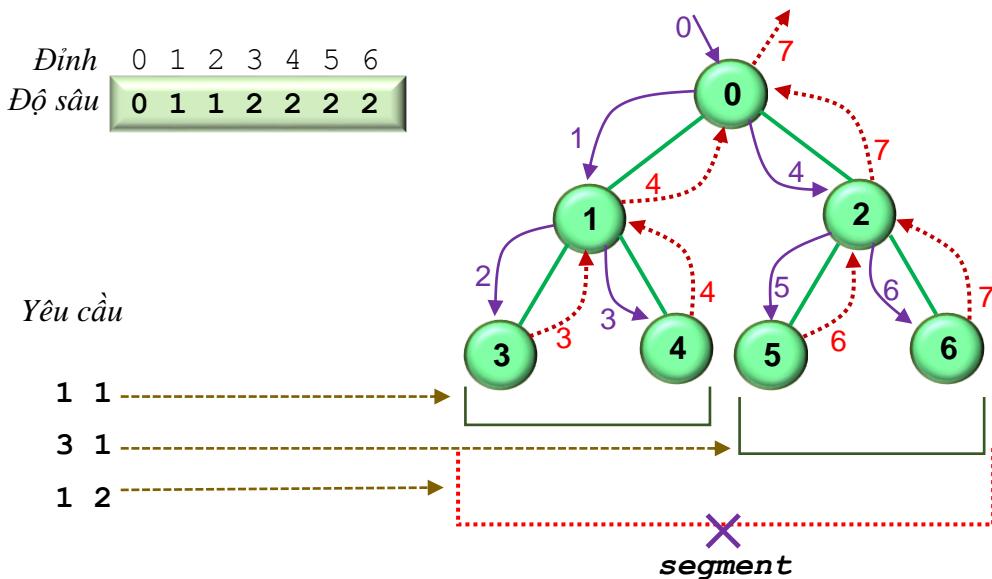
```

void removeIncluded()
{
    for (int level = 0; level < n; level++)
    {
        sort(levelSegments[level].begin(), levelSegments[level].end(),
              [](const segment &o1, const segment &o2)
        {
            if (o1.first==o2.first) return o1.second>o2.second;
            else return o1.first < o2.first;
        }
    );
    vector<segment> result;
    for (segment s : levelSegments[level])
    {
        while (result.size()>0 && result.back().second>=s.second)
            result.pop_back();
        result.push_back(s);
    }
    levelSegments[level] = result;
}
}

```

Điều kiện lồng

Kết quả loại bỏ các đoạn bao:



Xác định nghiệm:

Sử dụng **vector<int>** seg(n, -1) để đánh dấu các đoạn đã xác định, các nút trong một khoảng được đánh dấu bằng một số nguyên,

Gọi **segs** là tổng số lượng khoảng cần xét, dùng **vector<int>** segCount(segs) để ghi nhận những đoạn đã có đại diện (đã gấp khi duyệt),

Sử dụng kỹ thuật 2 con trỏ **left**, **right** để tìm đoạn ngắn nhất,
Biến **cursegs** ghi nhận số lượng đoạn khác nhau đã gặp,
Với **left** chạy từ 0 tới **n-1**, right tăng dần trong quá trình xử lý,
Với mỗi **left**, khi có **cursegs** bằng **segs** (có đủ đại diện), kiểm tra và cập nhật
kết quả,

Lưu ý:

- ♠ Trước khi chuyển sang điểm bắt đầu trái mới cần cập nhật kết quả thông kê hiện có do việc loại bỏ nút có số là **left**,
- ♠ Trong xử lý, các nút được đánh số từ 0, kết quả cần đưa ra: tương ứng với việc đánh số từ 1,
- ♠ Do không quay lui, nên độ phức tạp của quá trình duyệt là $O(n)$.

```

Dánh dấu phân  
biệt các đoạn

Diều kiện cập  
nhật kết quả

Xử lý đỉnh thuộc  
một đoạn

Dính thuộc  
đoạn mới

Xử lý trước khi  
sang left mới

Độ phức tạp của giải thuật: O(nlogn).

void findBestSegment ()
{
    vector<int> seg (n, -1);
    int segs = 0;
    for (int i = 0; i < n; i++)
        for (segment s : levelSegments[i])
        {
            for (int j = s.first; j <= s.second; j++)
                seg[verticesByDepth[i][j]] = segs;
            segs++;
        }

    vector<int> segCount(segs);
    int cursegs = 0;
    int rigth = 0;
    int bestL = 0, bestR = n;
    for (int left = 0; left < n; left++)
    {
        while (rigth < n && cursegs < segs)
        {
            if (seg[rigth] != -1)
            {
                if (segCount[seg[rigth]] == 0) cursegs++;
                segCount[seg[rigth]]++;
            }
            rigth++;
        }
        if (cursegs == segs)
            if (rigth-left < bestR-bestL) bestL=left, bestR=rigth;
        if (seg[left] != -1)
        {
            segCount[seg[left]]--;
            if (segCount[seg[left]] == 0) cursegs--;
        }
    }
    fo<<bestL + 1<< ' '<< bestR;
}

```

Chương trình:

```
#include <bits/stdc++.h>
#define NAME "training."

using namespace std;
ifstream fi (NAME"inp");
ofstream fo (NAME"out");
typedef pair<int, int> segment;

int const N = 200000;

int n,m;
vector<int> tree[N];
vector<int> verticesByDepth[N];
vector<segment> levelSegments[N];

int dfsTime = 0;
int timeIn[N], timeOut[N], depth[N];

void dfs(int u, int d)
{
    timeIn[u] = dfsTime++;
    depth[u] = d;
    verticesByDepth[d].push_back(u);
    for (int t = 0; t < (int) tree[u].size(); t++) dfs(tree[u][t], d + 1);
    timeOut[u] = dfsTime;
}

void initializeTree()
{
    for (int i = 1; i < n; i++)
    {
        int parent;
        fi>>parent;
        tree[parent - 1].push_back(i);
    }
    dfs(0, 0);
}

void removeIncluded()
{
    for (int level = 0; level < n; level++)
    {
        sort(levelSegments[level].begin(), levelSegments[level].end(),
              [] (const segment &o1, const segment &o2)
        {
            if (o1.first == o2.first) return o1.second > o2.second;
            else return o1.first < o2.first;
        });
        vector<segment> result;
        for (segment s : levelSegments[level])
        {
            while (result.size() > 0 && result.back().second >= s.second)
                result.pop_back();
            result.push_back(s);
        }
        levelSegments[level] = result;
    }
}
```

```

    }

}

segment findSegment(int depth, int tIn, int tOut)
{
    int from, to;
    int left = -1, right = (int) verticesByDepth[depth].size();
    while (left < right - 1)
    {
        int mid = (left + right) / 2;
        if (timeIn[verticesByDepth[depth][mid]] >= tIn) right = mid;
        else left = mid;
    }
    from = right;

    left = -1; right = (int) verticesByDepth[depth].size();
    while (left < right - 1)
    {
        int mid = (left + right) / 2;
        if (timeOut[verticesByDepth[depth][mid]] <= tOut) left = mid;
        else right = mid;
    }
    to = left;

    return segment(from, to);
}

void findBestSegment()
{
    vector<int> seg(n, -1);
    int segs = 0;
    for (int i = 0; i < n; i++)
        for (segment s : levelSegments[i])
    {
        for (int j = s.first; j <= s.second; j++)
            seg[verticesByDepth[i][j]] = segs;
        segs++;
    }

    vector<int> segCount(segs);
    int cursegs = 0;
    int rigth = 0;
    int bestL = 0, bestR = n;
    for (int left = 0; left < n; left++)
    {
        while (rigth < n && cursegs < segs)
        {
            if (seg[rigth] != -1)
            {
                if (segCount[seg[rigth]] == 0) cursegs++;
                segCount[seg[rigth]]++;
            }
            rigth++;
        }
        if (cursegs == segs)
            if (rigth - left < bestR - bestL) bestL = left, bestR = rigth;
        if (seg[left] != -1)
        {
            segCount[seg[left]]--;
        }
    }
}

```

```

        if (segCount[seg[left]] == 0) cursegs--;
    }
    fo<<bestL + 1<< ' '<< bestR;
}

int main()
{
    fi>>n;
    initializeTree();
    fi>>m;
    for (int i = 0; i < m; i++)
    {
        int u, k;
        fi>>u>>k;
        u--;
        levelSegments[depth[u]+k].push_back
            (findSegment(depth[u]+k, timeIn[u], timeOut[u]));
    }

    removeIncluded();
    findBestSegment();
    fo<<"\nTime: "<<clock() / (double)1000<<" sec";
    return 0;
}

```



Epoxy là một loại keo dán vạn năng, có thể gắn rất chắc nhiều loại vật liệu khác nhau. Keo gồm 2 thành phần **A** và **B** dưới dạng lỏng. Để dán người ta phải trộn đều 2 thành phần này với nhau với khối lượng mỗi thành phần là như nhau. Mỗi thành phần, nếu để riêng sẽ vẫn tồn tại dưới dạng lỏng. Nhưng nếu là hỗn hợp 2 thành phần thì sẽ đông cứng rất nhanh và chỉ trở thành chất dán rất chắc nếu khối lượng mỗi thành phần là như nhau. Vì vậy, để có keo dán người ta phải đổ hết vào bát tất cả chất lỏng thuộc cùng một thành phần, sau đó mới đổ chất lỏng thành phần thứ hai với khối lượng tương đương vào, trộn đều rất nhanh và mang đi dán.

Để dán lại một thiết bị thí nghiệm bị vỡ giáo sư Braun đặt mua gấp keo epoxy. Hàng được mang đến là n tuýp chất lỏng, ở ngoài ghi rõ khối lượng tính theo một mức chuẩn nào đó. Ví dụ, mức chuẩn là 10 ml, nếu tuýp chứa 12 ml thì trên nhãn ghi số 2, nếu tuýp chứa 6 ml thì trên nhãn ghi số -4. Do quá vội, người ta quên ghi ở mỗi tuýp loại chất lỏng mà nó chứa.

Khi gọi điện hỏi lại thì nhà cung cấp cho biết toàn bộ chất loại **B** được chứa trong một tuýp, các tuýp còn lại chứa chất loại **A**.

Khi xem lại các số ghi khối lượng, giáo sư Braun nhanh chóng nhận ra chỉ có đúng một tuýp chứa chất **B**. Ông đổ hết chất loại **A** ra bát, cuối cùng – đổ chất loại **B** và khuấy đều.

Hãy chỉ ra khối lượng các tuýp theo trình tự đổ ra bát. Nếu có nhiều trình tự thực hiện khác nhau – chỉ ra một cách tùy chọn.

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

- ✚ Dòng đầu tiên chứa một số nguyên n ($3 \leq n \leq 10^5$),
- ✚ Dòng thứ 2 chứa n số nguyên f_1, f_2, \dots, f_n ($-10^9 \leq f_i \leq 10^9$, $i = 1 \div n$).

Kết quả: Đưa ra file văn bản EPOXY.OUT n số nguyên xác định khối lượng các tuýp theo trình tự đổ ra bát ứng với cách đổ được chọn.

Ví dụ:

EPOXY.INP
3
2 5 3

EPOXY.OUT
2 3 5



VV12 Io20170319 A

Giải thuật: Số học đơn giản.

Nhận xét:

Theo điều kiện đã cho trong dãy tồn tại đúng một số có giá trị bằng tổng các số còn lại. Gọi số đó là **x**, ta có

$$2 \times \mathbf{x} = \sum_{i=1}^n a_i$$

Dễ dàng xác định được **x**, đưa ra các $\mathbf{a}_i \neq \mathbf{x}$ và cuối cùng – đưa ra **x**.

Độ phức tạp của giải thuật: O(n).

Chương trình:

```
#include <bits/stdc++.h>
#define NAME "epoxy."

using namespace std;
ifstream fi(NAME"inp");
ofstream fo(NAME"out");

int main()
{
    int n; int64_t total=0;
    fi >> n;
    vector<int64_t> f(n);
    for (int i = 0; i < n; i++)
        {fi >> f[i]; total+=f[i];}
    total>>=1;
    for(int i=0;i<n;++i) if(f[i]!=total) fo<<f[i]<<' ';
    fo<<total;
    fo <<"\nTime: "<<clock() / (double)1000<<" sec" ;
}
```



VV13. ĐOÁN NHẬN NGHỀ NGHIỆP

Tên chương trình: RECOGNIZE.CPP

Trong giờ Tâm lý học người ta chiếu một video clip để các học viên đoán nhận nghề nghiệp của các người trên một chuyến xe buýt. Xe có n hàng ghế đánh số từ 1 đến n , mỗi hàng có 2 ghế trái và phải (đánh số tương ứng 1 và 2). Có 5 nhóm nghề cần nhận dạng: **A** – những người làm toán, tin học, **B** – những người quản lý, lãnh đạo, **C** – những người làm nghề tự do, **D** – công nhân, viên chức, **E** – các nhà văn, nhà báo. Mỗi nhóm người có một cách chọn chỗ ngồi trên ô tô buýt theo thói quen nghề nghiệp của mình:

- ▲ Nhóm **A**: Thích gọn gàng, trật tự và luôn chọn ghế trống ở hàng có số thấp nhất và ngồi vào ghế trái nếu còn trống, trong trường hợp ngược lại – ngồi vào ghế phải,
- ▲ Nhóm **B**: theo thói quen hàng ngày, luôn chọn ghế trống ở hàng có số thấp nhất và ngồi vào ghế phải nếu còn trống, trong trường hợp ngược lại – ngồi vào ghế trái,
- ▲ Nhóm **C**: luôn chọn ghế trống ở hàng có số cao nhất và ngồi vào ghế trái nếu còn trống, trong trường hợp ngược lại – ngồi vào ghế phải,
- ▲ Nhóm **D**: luôn chọn ghế trống ở hàng có số cao nhất và ngồi vào ghế phải nếu còn trống, trong trường hợp ngược lại – ngồi vào ghế trái,
- ▲ Nhóm **E**: ngồi ở ghế trống bất kỳ tùy hứng.

Video quay ở bên hành, ban đầu mọi chỗ còn trống và có k khách (đánh số từ 1 đến k) lần lượt lên, người thứ i ngồi vào hàng \mathbf{x}_i , ghế \mathbf{y}_i , $i = 1 \div k$. Các chỗ ngồi là khác nhau.

Hãy xác định số người tối đa có thể ở mỗi nhóm và danh sách những người trong nhóm theo thứ tự tăng dần. Một người có thể được dự đoán ở nhiều nhóm. Có thể có nhóm không có ai.

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

- ✚ Dòng đầu tiên chứa 2 số nguyên n và k ($1 \leq n \leq 10^9$, $1 \leq k \leq \min\{2 \times 10^5, 2 \times n\}$),
- ✚ Dòng thứ i trong k dòng sau chứa 2 số nguyên \mathbf{x}_i và \mathbf{y}_i ($1 \leq \mathbf{x}_i \leq n$, $1 \leq \mathbf{y}_i \leq 2$).

Kết quả: Đưa ra file văn bản RECOGNIZE.OUT 5 dòng chứa thông tin về 5 nhóm, số đầu tiên của dòng là số lượng người trong nhóm, tiếp theo là danh sách những người trong nhóm theo thứ tự tăng dần.

Ví dụ:

RECOGNIZE.INP	RECOGNIZE.OUT
3 4	3 1 2 4
1 1	1 2
1 2	0
3 2	1 3
2 1	4 1 2 3 4



Giải thuật: Phương pháp hai con trỏ.

Nhận xét:

Tất cả mọi người vào đều có thể thuộc nhóm **E**, ta chỉ có thể lọc và đánh dấu các người từ nhóm **A** đến **D**,

Cần quản lý chỗ trống ở hàng đầu và hàng cuối,

Mỗi khi có người vào – kiểm tra ghế ngồi có thuộc hàng đầu hay cuối và ghi nhận vào tập tương ứng,

Khi hàng đầu đang quản lý hết chỗ trống: tìm hàng đầu mới có *số hàng lớn hơn*,

Khi hàng cuối đang quản lý hết chỗ trống: tìm hàng cuối mới có *số hàng nhỏ hơn*,

Đánh dấu chỗ đã có người ngồi bằng một cấu trúc dữ liệu cho phép dễ dàng xác định một chỗ đã có người ngồi hay chưa.

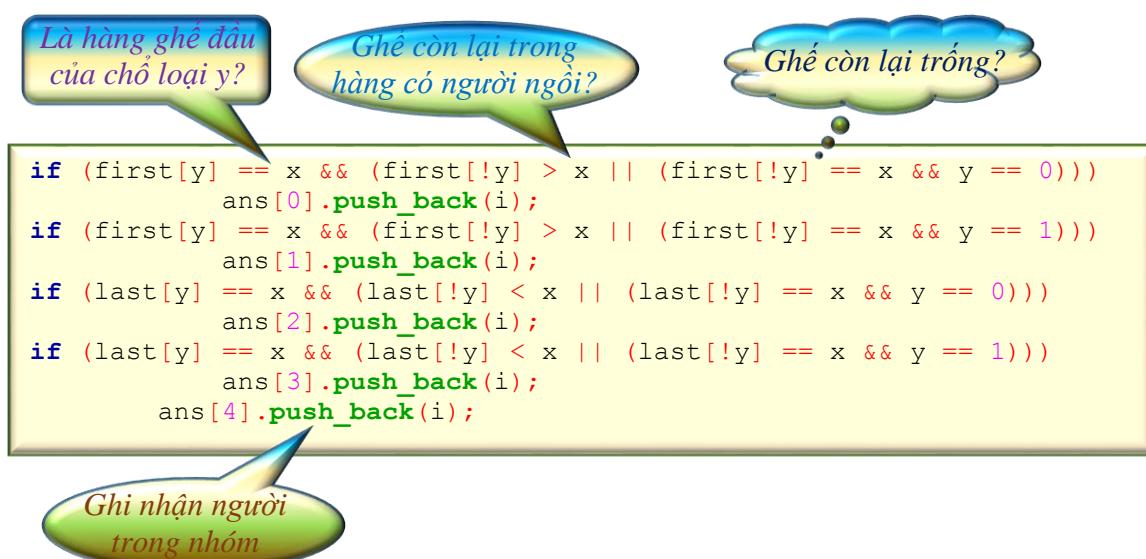
Tổ chức dữ liệu:

Để tiện xử lý dữ liệu được đánh số từ 0,

- Con trỏ quản lý hàng đầu `int first[2]: first[0]` xác định hàng nhỏ nhất có ghế 0 vẫn trống, `first[1]` xác định hàng nhỏ nhất có ghế 1 vẫn trống,
- Con trỏ quản lý hàng cuối `int last[2]: last[0]` xác định hàng lớn nhất có ghế 0 vẫn trống, `last[1]` xác định hàng lớn nhất có ghế 1 vẫn trống,
- Mảng tập `unordered_set<int> seat[2]` đánh dấu các chỗ đã có người ngồi,
- Mảng `vector<int> ans[5]` ghi nhận người thuộc các nhóm cần xác định.

Xử lý:

Nhận dạng nghề nghiệp:



Cập nhật trạng thái chỗ ngồi và con trỏ:

Lưu hàng của ghế mới được ngồi

Ghế y của hàng đầu đang xét không trống?

```
seat[y].insert(x);
while (seat[y].count(first[y]))  
    ++first[y];
while (seat[y].count(last[y]))  
    --last[y];
```

Độ phức tạp của giải thuật: $O(k \log k)$.

Lưu ý: Có thể dùng `set<int> seat[2]` thay cho `unordered_set<int> seat[2]`.

Chương trình:

```
#include <bits/stdc++.h>
#define NAME "recognize."
using namespace std;
ifstream fi (NAME"inp");
ofstream fo (NAME"out");
const int N = 200500;

int n, k;
unordered_set < int > seat[2];
int first[2]={0}, last[2];
vector < int > ans[5];

int main()
{
    fi>>n>>k;
    last[0] = last[1] = n - 1;

    for (int i = 0; i < k; ++i)
    {
        int x, y;
        fi>>x>>y;
        --x;
        --y;

        if (first[y] == x && (first[!y] > x || (first[!y] == x && y == 0)))
            ans[0].push_back(i);
        if (first[y] == x && (first[!y] > x || (first[!y] == x && y == 1)))
            ans[1].push_back(i);
        if (last[y] == x && (last[!y] < x || (last[!y] == x && y == 0)))
            ans[2].push_back(i);
        if (last[y] == x && (last[!y] < x || (last[!y] == x && y == 1)))
            ans[3].push_back(i);
        ans[4].push_back(i);

        seat[y].insert(x);
        while (seat[y].count(first[y]))
            ++first[y];
        while (seat[y].count(last[y]))
            --last[y];
    }

    for (int i = 0; i < 5; ++i)
    {
        fo<<ans[i].size();
        for (int x : ans[i])
            fo<<' '<<x+1;
        fo<<'\n';
    }
    fo<<"\nTime: "<<clock() / (double)1000<<" sec";
    return 0;
}
```



Các rô bốt hiện đại được lắp ráp theo các mô đun rời, nhờ đó, thay thế một mô đun bằng mô đun khác ta có thể dễ dàng thay đổi tính năng của rô bốt. Các mô đun phải đồng bộ theo tín hiệu điều khiển số và phải có khớp mốc nối cùng kiểu.

Một rô bốt đa năng được lắp ráp từ 3 mô đun **A**, **B** và **C**. Mỗi mô đun được đặc trưng bởi một số nguyên xác định kiểu loại, gọi là mã sản phẩm, chữ số đầu và chữ số cuối của mã xác định kiểu khớp mốc nối để lắp ráp. Mã sản phẩm không chứa các số 0 không có nghĩa.

Trong triển lãm giới thiệu sản phẩm có trưng bày **a** mô đun loại **A**, **b** mô đun loại **B** và **c** mô đun loại **C**, mỗi mô đun do một nhà máy sản xuất. Hàng ngày người ta lắp ráp mô đun thành rô bốt trình diễn theo các quy tắc sau:

- ❖ Chữ số đầu tiên (ở hàng đơn vị) của mã mô đun loại **A** phải giống cuối cùng (ở bậc cao nhất) của mã mô đun loại **B**,
- ❖ Chữ số đầu tiên (ở hàng đơn vị) của mã mô đun loại **B** phải giống chữ số cuối cùng (ở bậc cao nhất) của mã mô đun loại **C**,
- ❖ Rô bốt không chứa các mô đun có cùng mã sản phẩm.

Hai rô bốt gọi là khác nhau nếu các mô đun khác nhau ở mã sản phẩm hoặc cùng mã nhưng do các nhà máy khác nhau sản xuất.

Hãy xác định số rô bốt khác nhau có thể mang ra trình diễn.

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

- ✚ Dòng đầu tiên chứa 3 số nguyên **a**, **b**, **c** ($1 \leq a, b, c \leq 10^5$),
- ✚ Dòng thứ 2 chứa **a** số nguyên **x₁**, **x₂**, ..., **x_a** các mã của mô đun loại **A** ($0 \leq x_i \leq 10^9$, $i = 1 \div a$),
- ✚ Dòng thứ 3 chứa **b** số nguyên **y₁**, **y₂**, ..., **y_b** các mã của mô đun loại **B** ($0 \leq y_i \leq 10^9$, $i = 1 \div b$),
- ✚ Dòng thứ 4 chứa **c** số nguyên **z₁**, **z₂**, ..., **z_c** các mã của mô đun loại **C** ($0 \leq z_i \leq 10^9$, $i = 1 \div c$).

Kết quả: Đưa ra file văn bản ASSEMBLE.OUT một số nguyên – số rô bốt khác nhau có thể lắp ráp.

Ví dụ:

ASSEMBLE.INP
3 3 2
101 11 52
11 23 23
31 13

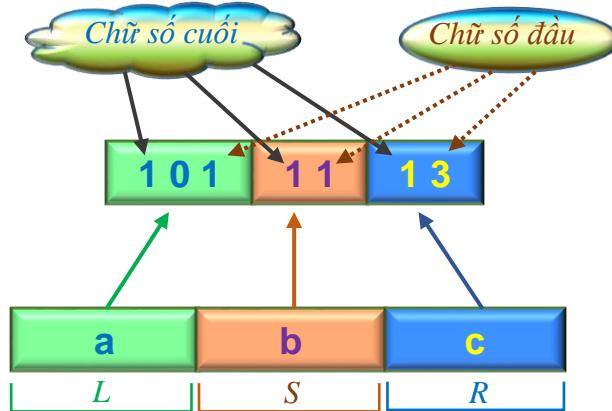
ASSEMBLE.OUT
3



Giải thuật: Kỹ thuật đếm cấu hình.

Nhận xét:

Với mỗi chữ số đầu i ($i = 0 \div 9$) và chữ số cuối j ($j = 1 \div 9$) ta có thể tính A_i – số lượng mã a có chữ số đầu là i , C_j – số lượng mã c có chữ số cuối là j , $B_{j,i}$ – số lượng mã b có chữ số đầu là i và chữ số cuối là j ,



Nếu bỏ qua ràng buộc khác mã thì số khả năng lắp ráp là

$$r = \sum_{i=0}^9 \sum_{j=0}^9 A_i \times B_{i,j} \times C_j \quad (*)$$

Loại bỏ các trường hợp trùng mã:

Nếu a và b trùng nhau thì ab tương đương với ba , điều này có nghĩa là mã a cần có chữ số đầu và chữ số cuối giống nhau,

Gọi L_t – số lượng mã a bằng t , S_t – số lượng mã b bằng t , R_t – số lượng mã c bằng t , C_{jt} – số lượng mã c có chữ số cuối của t ,

Với cả 3 mã a , b và c đều bằng t ta phải chỉnh lý lại r theo công thức:

$$r += -L_t \times S_t \times C_{jt} + 2 \times L_t \times S_t \times R_t$$

Trong công thức tính tổng (*) với mỗi t thuộc \mathbf{A} bị lặp lại ở \mathbf{C} cần chỉnh lý r theo công thức:

$$r -= -L_t \times B_{it, jt} \times R_t$$

Với mỗi t thuộc \mathbf{B} bị lặp lại ở \mathbf{C} cần chỉnh lý r theo công thức:

$$r -= -S_t \times A_t \times R_t$$

Lưu ý: giá trị r có thể rất lớn!

Tổ chức dữ liệu:

☞ Các mảng lưu số lượng mã bắt đầu và kết thúc với mọi khả năng có thể, tức là các giá trị **A_i**, **C_j**, **B_{j,i}** đã nói ở trên:

vector<int>A(10), C(10) và **vector<vector<int>>B(10, vector<int>(10))**

☞ Các tập **map<int, int> L, R, S** lưu tần số xuất hiện các mã.

Xử lý:

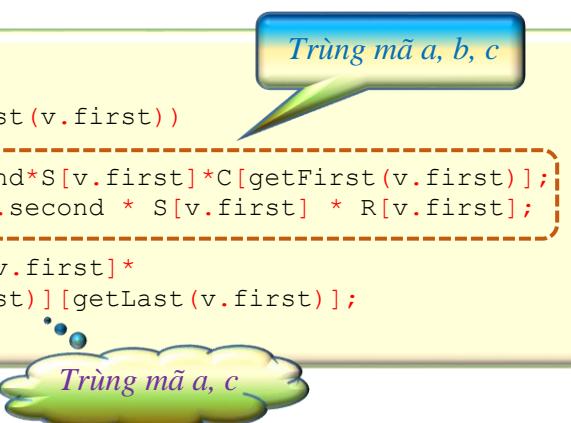
Sử dụng các hàm **int getFirst(int x)** và **int getLast(int x)** để xác định chữ số đầu/cuối của mã **x**,

Tích lũy tần số xuất hiện các mã thuộc mô đun loại **A** (và tương tự đối với mô đun loại **B, C**):

```
for(int i = 0; i < a; i++)
{
    int t;
    fi >> t;
    A[getFirst(t)]++;
    L[t]++;
}
```

Chỉnh lý kết quả khi có mã ở **A** trùng với mã ở **B** và có thể - với cả ở **C**:

```
for(auto v : L)
{
    if(getLast(v.first) == getFirst(v.first))
    {
        result -= (int64_t)v.second*S[v.first]*C[getFirst(v.first)];
        result += (int64_t) 2 * v.second * S[v.first] * R[v.first];
    }
    result -= (int64_t)v.second*R[v.first]*
              B[getFirst(v.first)][getLast(v.first)];
}
```



Trường hợp trùng mã ở A và C:

```
for(auto v : S)
    if(getLast(v.first) == getFirst(v.first))
        result -= (int64_t)v.second * A[getLast(v.first)]*R[v.first];
```

Chú thích: nếu **v** là một phần tử của **map<int, int> x**, thì **v.first** là khóa, **v.second** – giá trị của khóa.

Độ phức tạp của giải thuật: bậc O(nlogn).

Chương trình:

```
#include <bits/stdc++.h>
#define NAME "assemble."
using namespace std;
ifstream fi (NAME"inp");
ofstream fo (NAME"out");
const int N = 200500;

int getFirst(int x)
{
    return x % 10;
}

int getLast(int x)
{
    while(x >= 10) x /= 10;
    return x;
}

int main()
{
    int a, b, c;
    fi >> a >> b >> c;
    map<int, int> L, R, S;
    vector<vector<int> > B(10, vector<int>(10));
    vector<int> A(10), C(10);
    for(int i = 0; i < a; i++)
    {
        int t;
        fi >> t;
        A[getFirst(t)]++;
        L[t]++;
    }
    for(int i = 0; i < b; i++)
    {
        int t;
        fi >> t;
        B[getLast(t)][getFirst(t)]++;
        S[t]++;
    }
    for(int i = 0; i < c; i++)
    {
        int t;
        fi >> t;
        C[getLast(t)]++;
        R[t]++;
    }

    int64_t result = 0;
    for(int i = 0; i < 10; i++)
        for(int j = 0; j < 10; j++)
            result += (int64_t) A[i] * B[i][j] * C[j];

    for(auto v : L)
    {
        if(getLast(v.first) == getFirst(v.first))
        {
```

```

        result -= (int64_t) v.second * S[v.first] * C[getFirst(v.first)];
        result += (int64_t) 2 * v.second * S[v.first] * R[v.first];
    }
    result -=(int64_t)v.second*R[v.first]*
        B[getFirst(v.first)][getLast(v.first)];
}
for(auto v : S)
    if(getLast(v.first) == getFirst(v.first))
        result -= (int64_t) v.second * A[getLast(v.first)] * R[v.first];
fo << result;
fo<<"\nTime: "<<clock() / (double)1000<<" sec";
}

```



Hài hước là yếu tố khó cài đặt nhất trong trí tuệ nhân tạo (AI). Viện Nghiên cứu AI đã bước đầu thành công đưa yếu tố hài hước vào trí tuệ nhân tạo, tạo ra các rô bốt biết hoạt động cả trong chế độ vui đùa. Để kiểm tra tiến độ dự án Ban quản lý yêu cầu tổ chức một buổi trình diễn.

Người ta xếp các rô bốt đứng thành 2 hàng ngang quay mặt về nhau, mỗi hàng có **k** rô bốt. Một số rô bốt được kích hoạt chế độ vui đùa, số còn lại – ở chế độ nghiêm túc. Là sản phẩm công nghiệp, các rô bốt giống nhau như những giọt nước, nhưng các rô bốt qua sóng điện từ, biết được ai đang ở chế độ nào.

Một câu hỏi được đưa ra đồng thời cho mọi rô bốt. Câu hỏi có một trong hai dạng:

Dạng **A**: “*Có phải có đúng **x** rô bốt cạnh bạn là người nghiêm túc?*” hoặc “*Có phải có đúng **y** rô bốt cạnh bạn là người hài hước?*”

Dạng **B**: “*Có phải có đúng **x** rô bốt cạnh bạn là người nghiêm túc và có đúng **y** rô bốt cạnh bạn là người hài hước?*”

Với mỗi rô bốt, người đứng cạnh là các rô bốt bên phải, bên trái cùng hàng và rô bốt đối diện ở hàng kia, trừ các rô bốt đứng đầu và cuối hàng sẽ không có một người cùng hàng. Ở chế độ nghiêm túc rô bốt luôn trả lời đúng như được hỏi, ở chế độ hài hước – nói dối.

Những người trong Ban quản lý kinh ngạc khi thấy các rô bốt đồng thanh trả lời như trong quân đội “**YES, SIR!**” (“Đúng, thưa ngài!”) và để kiểm tra kết quả người ta muốn biết tối thiểu và tối đa trong các hàng tổng số có bao nhiêu rô bốt ở chế độ nghiêm túc và chúng được xếp như thế nào trong cả hai trường hợp.

Hãy đưa ra số lượng tối thiểu các rô bốt ở chế độ nghiêm túc, 2 xâu độ dài **k** chứa các ký tự từ tập {0, 1} xác định loại rô bốt trong hàng, 0 – chế độ hài hước, 1 – chế độ nghiêm túc và thông tin tương tự với số lượng tối đa.

Dữ liệu: Vào từ file văn bản SERIOUS.INP gồm một dòng chứa 3 số nguyên **k**, **x** và **y**, $1 \leq k \leq 10^5$, $-1 \leq x, y \leq 3$, giá trị -1 ký hiệu câu hỏi có dạng **A** và không hỏi về **x** nếu **x** = -1, không hỏi về **y** nếu **y** = -1.

Kết quả: Đưa ra file văn bản SERIOUS.OUT trên 6 dòng các thông tin cần xác định.

Ví dụ:

SERIOUS.INP	SERIOUS.OUT
10 0 3	5 0100010010 0001000100 8 0101010100 0010101010



Giải thuật: Đoán nhận giải thuật, xử lý chu trình, kỹ thuật lập trình 2 giai đoạn.

Nhận xét:

Về lý thuyết, bài toán này có thể giải bằng phương pháp quy hoạch động,

Ta phải xét 2 xâu chỉ chứa các ký tự trong tập {0, 1}, phạm vi ảnh hưởng của mỗi ký tự tới giá trị hàm mục tiêu chỉ là +1 hoặc -1 so với vị trí của nó, vì vậy các ký tự 0, 1 sẽ **xuất hiện theo chu kỳ** trong mỗi xâu,

Các giá trị **min**, **max** cần tìm cũng sẽ có mối phụ thuộc đơn giản vào **k**,

Do ký tự đầu và cuối xâu có ít hơn các ký tự khác một láng giềng nên trong một số trường hợp các phần đầu và cuối sẽ phụ thuộc vào tính chẵn lẻ của **k**,

Ta có thể xây dựng một chương trình đủ đơn giản (**chương trình khảo sát**), vét cạn mọi khả năng của xâu với **k** và **k+1** không lớn để xác định chu trình cho các loại câu hỏi có thể và công thức tính **min**, **max**,

Chương trình giải bài toán sẽ **dẫn xuất trực tiếp các xâu theo mẫu** đã xác định.

Chương trình khảo sát:

```
#include <bits/stdc++.h>
using namespace std;
ofstream fo("serious.txt");

int x, y, m, k;

bool ok(int who, int cnt0, int cnt1)
{
    bool res = true;
    if (y != -1)
        res &= (cnt0 == y);
    if (x != -1)
        res &= (cnt1 == x);
    if (who == 0)
        res = !res;
    return res;
}

void solve()
{
    fo<<"\n *** k,x,y: "<<k<<' '<<x<<' '<<y<<endl;
    int min_ans = 2 * k + 1, max_ans = -1;
    vector<vector<int>> a1, a2;
    for (int mask = 0; mask < (1 << (2 * k)); mask++)
    {
        vector<vector<int>> a(2, vector<int>(k));
        int cur = 0;
        for (int i = 0; i < 2 * k; i++)
        {
            int bit = (mask >> i) & 1;
            a[i / k][i % k] = bit;
            cur += bit;
        }
        bool flag = true;
        for (int i = 0; i < 2 && flag; i++)
        {
            int infront = (i + 1) % 2;
            for (int l = 0; l < k && flag; l++)
            {
                int cnt[2] = {0, 0};
                int left = l - 1;
                if (left >= 0)
                    cnt[a[i][left]]++;
                int right = l + 1;
                if (right < k)
                    cnt[a[i][right]]++;
                cnt[a[infront][l]]++;
                if (!ok(a[i][l], cnt[0], cnt[1]))
                    flag = false;
            }
        }
        if (flag)
        {
            if (cur < min_ans)
                min_ans = cur, a1 = a;
            if (cur > max_ans)
```

Hàm kiểm tra
điều kiện phù hợp

Hàm vết cạn

```

        max_ans = cur, a2 = a;
    }
}
fo << min_ans << endl;
for (int i = 0; i < 2; i++)
{
    for (int j = 0; j < k; j++)
        fo << a1[i][j];
    fo << endl;
}
fo << max_ans << endl;
for (int i = 0; i < 2; i++)
{
    for (int j = 0; j < k; j++)
        fo << a2[i][j];
    fo << endl;
}
}
Độ dài k khảo sát
int main()
{
    m=10;

    for(x=-1;x<4;++x)
        for(y=-1;y<4;++y)
    {
        if ((x== -1 && y == -1) || (x+y>3)) continue;
        k=m; solve();
        ++k; solve();
        fo<<" ..... \n";
    }

    fo<<"\n%%%%% Time: "<<clock() / (double)1000<<" sec";
    return 0;
}

```

Kết quả khảo sát: ở file **serios.txt**.

Chương trình giải cuối cùng:

```
#include <bits/stdc++.h>
#define NAME "serious."
using namespace std;
ifstream fi (NAME"inp");
ofstream fo (NAME"out");

void print0 (int k)
{
    fo << 0 << endl;
    for (int i = 0; i < 2; i++)
    {
        for (int j = 0; j < k; j++)
            fo << 0;
        fo << endl;
    }
}

void solve_x0 (int k)
{
    if (k == 1)
        fo << "1\n1\n0\n1\n1\n0\n1\n";
    else
    {
        fo << k / 2 + 1 << endl;
        int f = 0;
        if (k % 2 == 0)
            fo << 0, f = 1;
        for (int i = 0; i < k - f; i++)
            fo << (i % 4 == 0);
        fo << endl;
        if (k % 2 == 0)
            fo << 1;
        for (int i = 0; i < k - f; i++)
            fo << (i % 4 == 2);
        fo << endl;

        fo << k << endl;
        for (int i = 0; i < k; i++)
            fo << i % 2;
        fo << endl;
        for (int i = 0; i < k; i++)
            fo << (i + 1) % 2;
        fo << endl;
    }
}

void solve_x1 (int k)
{
    if (k == 1)
        fo << "0\n0\n0\n2\n1\n1\n";
    else
    {
        print0 (k);

        if (k % 2 == 0)
            print0 (k);
```

```

        else
        {
            fo << k + 1 << endl;
            for (int j = 0; j < 2; j++)
            {
                for (int i = 0; i < k; i++)
                    fo << (i + 1) % 2;
                fo << endl;
            }
        }
    }

void solve_x2 (int k)
{
    print0(k);
    if (k == 1)
    {
        print0(k);
        return;
    }
    int tmp = 0;
    while (4 * tmp + 2 <= k)
        tmp++;
    tmp = max(0, tmp - 1);
    fo << 4 * (tmp + 1) << endl;
    for (int j = 0; j < 2; j++)
    {
        for (int i = 0; i < 4 * tmp; i += 4)
        {
            fo << "1100";
        }
        fo << "11";
        for (int i = 0; i < k - 4 * tmp - 2; i++)
            fo << 0;
        fo << endl;
    }
}

void solve_x3 (int k)
{
    print0(k);
    print0(k);
}

void solve_y0 (int k)
{
    print0(k);
    fo << 2 * k << endl;
    for (int j = 0; j < 2; j++)
    {
        for (int i = 0; i < k; i++)
            fo << 1;
        fo << endl;
    }
}

void solve_y1 (int k)

```

```

{
    if (k == 1)
    {
        fo << "1\n1\n0\n1\n0\n1\n";
        return;
    }
    print0(k);
    if (k % 2 == 1)
    {
        fo << 2 * k - k / 2 - 1 << endl;
        for (int i = 0; i < k; i++)
            fo << (i % 4 != 0);
        fo << endl;
        for (int i = 0; i < k; i++)
            fo << (i % 4 != 2);
        fo << endl;
    }
    else
    {
        int f = 0;
        if (k % 4 == 2)
            f = 2;
        fo << k - f << endl;
        for (int i = 0; i < 2; i++)
        {
            for (int j = 0; j + 4 <= k; j += 4)
                fo << "1001";
            if (f)
                fo << "00";
            fo << endl;
        }
    }
}

void solve_y2(int k)
{
    for (int i = 0; i < 2; i++)
    {
        fo << k - k % 2 << endl;
        if (k % 2 == 0)
        {
            for (int j = 0; j < k; j += 2)
                fo << ((j % 4 == 0) ? "10" : "01");
            fo << endl;
            for (int j = 0; j < k; j += 2)
                fo << ((j % 4 == 0) ? "01" : "10");
            fo << endl;
        }
        else
        {
            for (int l = 0; l < 2; l++)
            {
                for (int j = 0; j < k; j++)
                    fo << j % 2;
                fo << endl;
            }
        }
    }
}

```

```

void solve_y3 (int k)
{
    if (k <= 2)
    {
        print0(k);
        print0(k);
        return;
    }
    vector<vector<int>> res(2, vector<int>(k, 0));
    int cnt = 0;
    for (int i = 1; i < k - 1; i++)
        res[0][i] = (i % 4 == 1), cnt += res[0][i];
    for (int i = 1; i < k - 1; i++)
        res[1][i] = (i % 4 == 3), cnt += res[1][i];
    if (res[0][k - 2] == 0 && res[1][k - 2] == 0)
    {
        if ((k - 2) % 4 == 0)
            res[0][k - 2] = 1;
        else
            res[1][k - 2] = 1;
        cnt++;
    }
    fo << cnt << endl;
    for (int j = 0; j < 2; j++)
    {
        for (int i = 0; i < k; i++)
            fo << res[j][i];
        fo << endl;
    }

    fo << k - 2 << endl;
    fo << 0;
    for (int i = 1; i < k - 1; i++)
        fo << i % 2;
    fo << 0 << endl;
    fo << 0;
    for (int i = 1; i < k - 1; i++)
        fo << (i + 1) % 2;
    fo << 0 << endl;
}

void solve_x0_y0 (int k)
{
    print0(k);
    print0(k);
}

void solve_x0_y1 (int k)
{
    if (k == 1)
    {
        fo << "1\n1\n0\n1\n0\n1\n";
        return;
    }
    print0(k);
    print0(k);
}

```

```

void solve_x0_y2(int k)
{
    if (k == 1)
    {
        print0(k);
        print0(k);
        return;
    }
    for (int i = 0; i < 2; i++)
    {
        fo << 2 << endl;
        fo << 1;
        for (int j = 1; j < k; j++)
            fo << 0;
        fo << endl;
        for (int j = 1; j < k; j++)
            fo << 0;
        fo << 1 << endl;
    }
}

void solve_x0_y3(int k)
{
    solve_y3(k);
}

void solve_x1_y0(int k)
{
    if (k == 1)
    {
        fo << "0\n0\n0\n2\n1\n1\n";
        return;
    }
    print0(k);
    print0(k);
}

void solve_x1_y1(int k)
{
    if (k <= 2)
    {
        print0(k);
        print0(k);
        return;
    }
    print0(k);
    fo << 4 << endl;
    fo << 1;
    for (int j = 1; j < k - 1; j++)
        fo << 0;
    fo << 1 << endl;
    fo << 1;
    for (int j = 1; j < k - 1; j++)
        fo << 0;
    fo << 1 << endl;
}

```

```

void solve_x1_y2 (int k)
{
    if (k == 1)
        fo << "0\n0\n0\n0\n0\n0\n";
    else
    {
        print0(k);

        if (k % 2 == 0)
            print0(k);
        else
        {
            fo << k - 1 << endl;
            for (int j = 0; j < 2; j++)
            {
                for (int i = 0; i < k; i++)
                    fo << i % 2;
                fo << endl;
            }
        }
    }
}

void solve_x2_y0 (int k)
{
    if (k == 2)
        fo << "0\n00\n00\n4\n11\n11\n";
    else
    {
        print0(k);
        print0(k);
    }
}

void solve_x2_y1 (int k)
{
    print0(k);
    if (k <= 3)
    {
        print0(k);
        return;
    }
    fo << k - k % 4 << endl;
    for (int j = 0; j < 2; j++)
    {
        int i;
        for (i = 0; i + 4 <= k; i += 4)
        {
            fo << "0110";
        }
        for (; i < k; i++)
            fo << 0;
        fo << endl;
    }
}

```

```

int main()
{
    int k, x, y;
    fi >> k >> x >> y;
    if (x == 0 && y == -1)
        solve_x0(k);
    else if (x == 1 && y == -1)
        solve_x1(k);
    else if (x == 2 && y == -1)
        solve_x2(k);
    else if (x == 3 && y == -1)
        solve_x3(k);
    else if (x == -1 && y == 0)
        solve_y0(k);
    else if (x == -1 && y == 1)
        solve_y1(k);
    else if (x == -1 && y == 2)
        solve_y2(k);
    else if (x == -1 && y == 3)
        solve_y3(k);
    else if (x == 0 && y == 0)
        solve_x0_y0(k);
    else if (x == 0 && y == 1)
        solve_x0_y1(k);
    else if (x == 0 && y == 2)
        solve_x0_y2(k);
    else if (x == 0 && y == 3)
        solve_x0_y3(k);
    else if (x == 1 && y == 0)
        solve_x1_y0(k);
    else if (x == 1 && y == 1)
        solve_x1_y1(k);
    else if (x == 1 && y == 2)
        solve_x1_y2(k);
    else if (x == 2 && y == 0)
        solve_x2_y0(k);
    else if (x == 2 && y == 1)
        solve_x2_y1(k);
    else
        print0(k), print0(k);
        fo<<"\nTime: "<<clock() / (double)1000<<" sec";
    return 0;
}

```



VV16. PHAO NỐI

Tên chương trình: FLOAT.CPP

Người ta cần làm gấp một phao nổi để trục vớt một thiết bị lặn bị chìm. Trong xưởng hiện đang có n tấm thép hình chữ nhật, tấm thứ i có kích thước $a_i \times b_i$, $i = 1 \dots n$. Phao có dạng hình hộp chữ nhật, lắp ghép từ 6 trong số các tấm thép hiện có. Các tấm thép có thể được xoay, nhưng không được cắt bớt hay gấp cong. Các đường hàn phải chạy theo mép của tấm thép để sau này khi tháo gỡ không làm hỏng các tấm thép.

Hãy xác định thể tích lớn nhất có thể có từ phao được làm. Nếu không có cách làm phao thì đưa ra số -1.

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

- ✚ Dòng đầu tiên chứa một số nguyên n ($6 \leq n \leq 2 \times 10^5$),
- ✚ Dòng thứ i trong n dòng sau chứa 2 số nguyên a_i và b_i ($1 \leq a_i, b_i \leq 10^6$).

Kết quả: Đưa ra file văn bản FLOAT.OUT một số nguyên – thể tích lớn nhất tìm được.

Ví dụ:

FLOAT.INP
6
3 6
6 9
9 3
6 3
3 9
9 6

FLOAT.OUT
162

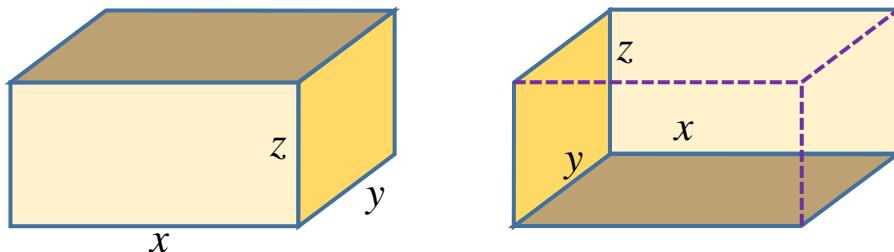


Giải thuật: Tổ chức dữ liệu.

Nhận xét:

Để xây dựng hình hộp chữ nhật kích thước $\mathbf{x} \times \mathbf{y} \times \mathbf{z}$ ta cần có 2 tấm kích thước $\mathbf{x} \times \mathbf{y}$, 2 tấm kích thước $\mathbf{y} \times \mathbf{z}$ và 2 tấm kích thước $\mathbf{x} \times \mathbf{z}$,

Chuẩn hóa dữ liệu để có $\mathbf{a}_i \leq \mathbf{b}_i$, $i = 1 \div n$,



Rút gọn tập dữ liệu xử lý:

- ❖ Loại bỏ các tấm thép không có tấm khác giống nó,
- ❖ Với mỗi tấm thép còn lại: lưu một nửa số lượng để xét khả năng từ đó có thể lắp được 3 vách vuông góc với nhau từng đôi một hay không,
- ❖ Để có thể làm điều đó, mỗi kích thước \mathbf{x} , \mathbf{y} , \mathbf{z} phải gấp 2 lần ở các tấm khác nhau, những tấm không thỏa mãn điều kiện này sẽ bị loại tiếp,

Lọc dữ liệu, ngoài việc *giảm kích thước xử lý* còn *đảm bảo tránh dùng lắp* một tấm thép ban đầu,

Với dữ liệu đã rút gọn, xét 3 khả năng:

- ❖ Cả 3 kích thước giống nhau,
- ❖ Có 2 kích thước giống nhau và khác kích thước thứ 3,
- ❖ Cả 3 kích thước khác nhau từng đôi một,

Hai trường hợp đầu dễ dàng xử lý bằng cách duyệt và kiểm tra trực tiếp,

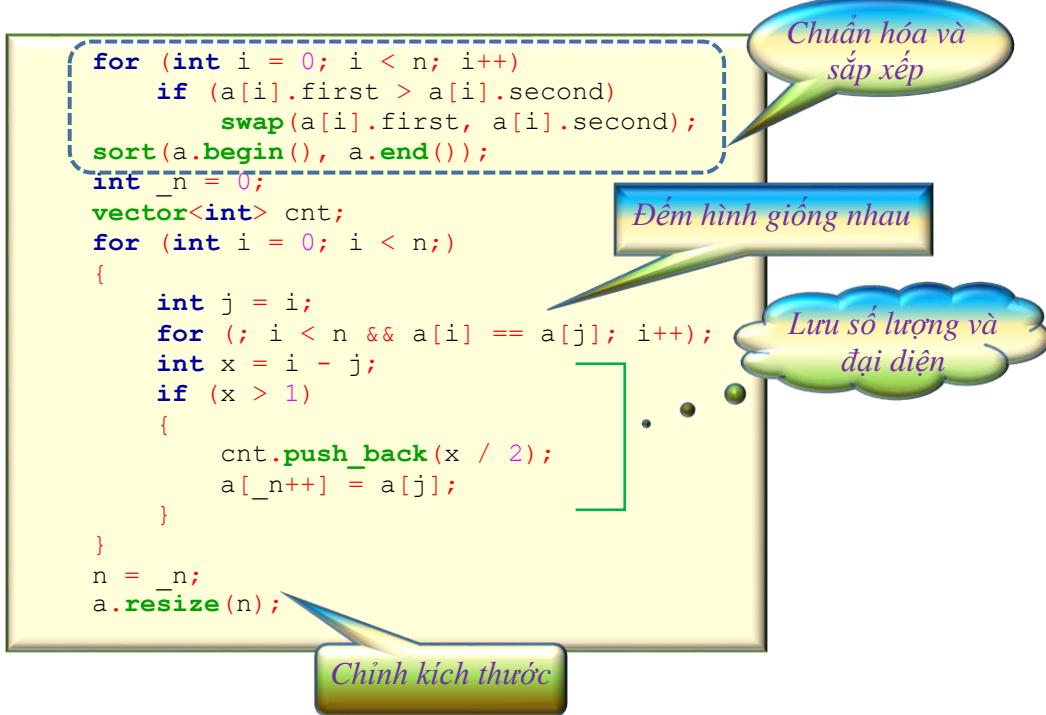
Trường hợp 3:

Xét đồ thị trong đó đỉnh là các số nguyên độ trong kích thước tấm thép, cạnh là cặp (a_i, b_i) ở tập dữ liệu đã được lọc,

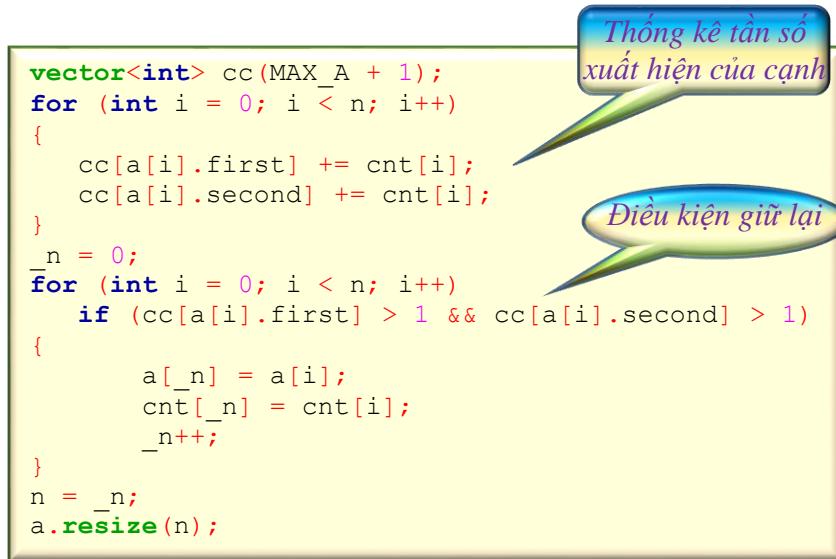
Xuất phát từ đỉnh có bậc thấp nhất tìm các chu trình độ dài 3 bằng cách duyệt các đỉnh kề với đỉnh đã chọn và có bậc cao thứ 2, tìm đỉnh thứ 3 từ tập đỉnh kề với đỉnh 2 và có bậc cao hơn, kiểm tra khả năng xuất hiện chu trình,

Tổ chức dữ liệu và Xử lý:

Chuẩn hóa dữ liệu và lọc các tám thép không có tám khác giống nó:



Loại bỏ các tám có kích thước không lặp:



Tạo mảng lưu các độ dài cạnh cần xử lý:

Mảng `vector<int>` vct lưu độ dài khác nhau của các cạnh cần xử lý,

Để sử dụng hàm `unique` lọc các dữ liệu giống nhau cần sắp xếp để các dữ liệu giống nhau đứng tiếp thành một nhóm.

```

vector<int> vct;
for (int i = 0; i < n; i++)
{
    vct.push_back(a[i].first);
    vct.push_back(a[i].second);
}
sort(vct.begin(), vct.end());
vct.resize(unique(vct.begin(), vct.end()) - vct.begin());
int m = vct.size();

```

Lọc dữ liệu và co vector

Xác định cạnh của đồ thị:

```

for (int i = 0; i < n; i++)
{
    a[i].first=lower_bound(vct.begin(),vct.end(),a[i].first)-vct.begin();
    a[i].second =lower_bound(vct.begin(),vct.end(),a[i].second)-vct.begin();
}

```

Xử lý trường hợp 3 hay 2 cạnh giống nhau:

```

ll ans = -1;
for (int i = 0; i < n; i++)
{
    if (a[i].first == a[i].second)
    {
        if (cnt[i] >= 3)
            ans=max(ans, 1LL*vct[a[i].first]*vct[a[i].first]*vct[a[i].first]);
        for (int j = i + 1; j < n && a[j].first == a[i].first; j++)
            if(cnt[j]>=2)
                ans= max(ans,1LL*vct[a[i].first]
                           *vct[a[j].first]* vct[a[j].second]);
    }
}

```

Quản lý cạnh theo giá trị tăng dần của đỉnh:

```

vector<pair<int, int> > edges;
for (int i = 0; i < n; i++) edges.push_back(a[i]);
sort(edges.begin(), edges.end());

```

Xét trường hợp 3 cạnh hình hộp khác nhau cùng đôi một:

Tạo ma trận kè của đỉnh và sắp xếp đỉnh theo bậc tăng dần, loại bỏ các cạnh móc nối một đỉnh tới chính nó:

Mảng `vector<vector<int> > y (m)` lưu danh sách đỉnh kè của cây,

Mảng `vector<int> p1 (m)` lưu đỉnh theo thứ tự tăng dần của bậc,

Mảng `vector<int> p2 (m)` lưu vị trí của đỉnh trong mảng xác định trình tự,

```

vector<vector<int>> y(m);
for (int i = 0; i < n; i++)
{
    if (a[i].first != a[i].second)
    {
        y[a[i].first].push_back(a[i].second);
        y[a[i].second].push_back(a[i].first);
    }
}
vector<int> p1(m), p2(m);
for (int i = 0; i < m; i++) p1[i] = i;
sort(p1.begin(), p1.end(),
    [&](int i, int j){return y[i].size() < y[j].size();});
for (int i = 0; i < m; i++) p2[p1[i]] = i;

```

Loại bỏ mốc nối
tới chính nó

Ghi nhận danh
sách đỉnh

Tạo danh sách đỉnh
theo bậc tăng dần

Bảng tra cù
bậc đỉnh

Tìm 3 đỉnh tạo thành chu trình xuất phát từ đỉnh có bậc thấp nhất:

```

for (int ii = 0; ii < m; ii++)
{
    int i = p1[ii];
    sort(y[i].begin(), y[i].end(),
        [&](int xx, int yy){return p2[xx]>p2[yy];});
    while (!y[i].empty() && p2[y[i].back()] < p2[i]) y[i].pop_back();
    for (int j = 0; j < y[i].size(); j++)
        for (int k = j + 1; k < y[i].size(); k++)
            if (binary_search(edges.begin(), edges.end(), pp(y[i][j], y[i][k])))
                ans = max(ans, 1LL * vct[i] * vct[y[i][j]] * vct[y[i][k]]);
}

```

Sắp xếp đỉnh theo
bậc tăng dần

Loại bỏ đỉnh có
bậc thấp hơn

Tìm đỉnh thứ 3

Chuẩn hóa cạnh

```
auto pp =[&] (int x, int y) {return make_pair(min(x, y), max(x, y));};
```

Độ phức tạp của giải thuật: O(nlogn).

Chương trình:

```
#include <bits/stdc++.h>
#define NAME "float."
using namespace std;
typedef long long ll;
ifstream fi (NAME"inp");
ofstream fo (NAME"out");
const int K = 1024;
const int MAX_A = (int)(1e6);

ll solve(vector<pair<int, int> > a)
{
    int n = a.size();
    for (int i = 0; i < n; i++)
        if (a[i].first > a[i].second)
            swap(a[i].first, a[i].second);
    sort(a.begin(), a.end());
    int _n = 0;
    vector<int> cnt;
    for (int i = 0; i < n; )
    {
        int j = i;
        for (; i < n && a[i] == a[j]; i++);
        int x = i - j;
        if (x > 1)
        {
            cnt.push_back(x / 2);
            a[_n++] = a[j];
        }
    }
    n = _n;
    a.resize(n);

    vector<int> cc(MAX_A + 1);
    for (int i = 0; i < n; i++)
    {
        cc[a[i].first] += cnt[i];
        cc[a[i].second] += cnt[i];
    }
    _n = 0;
    for (int i = 0; i < n; i++)
        if (cc[a[i].first] > 1 && cc[a[i].second] > 1)
        {
            a[_n] = a[i];
            cnt[_n] = cnt[i];
            _n++;
        }
    }

    n = _n;
    a.resize(n);

    vector<int> vct;
    for (int i = 0; i < n; i++)
    {
        vct.push_back(a[i].first);
        vct.push_back(a[i].second);
    }
}
```

```

sort(vct.begin(), vct.end());
vct.resize(unique(vct.begin(), vct.end()) - vct.begin()));
int m = vct.size();
for (int i = 0; i < n; i++)
{
    a[i].first=lower_bound(vct.begin(),vct.end(),a[i].first)-vct.begin();
    a[i].second =lower_bound(vct.begin(),vct.end(),a[i].second)-vct.begin();
}

ll ans = -1;
for (int i = 0; i < n; i++)
    if (a[i].first == a[i].second)
    {
        if (cnt[i] >= 3)
            ans=max(ans,1LL*vct[a[i].first]*vct[a[i].first]*vct[a[i].first]);
        for (int j = i + 1; j < n && a[j].first == a[i].first; j++)
            if(cnt[j]>=2) ans= max(ans,1LL*vct[a[i].first]
                                         *vct[a[j].first]* vct[a[j].second]);
    }

vector<int> p(n);
for (int i = 0; i < n; i++) p[i] = i;
sort(p.begin(), p.end(), [&](int i, int j)
     {return make_pair(a[i].second,-a[i].first) <
           make_pair(a[j].second,-a[j].first);});
for (int i = 0; i < n; i++) if(a[p[i]].first==a[p[i]].second)
    for (int j=i + 1; j < n && a[p[j]].second == a[p[i]].second;j++)
        if (cnt[p[j]] >= 2)
            ans = max(ans,1LL*vct[a[p[i]].first]*vct[a[p[j]].first]*
                      vct[a[p[j]]].second);

vector<pair<int, int> > edges;
for (int i = 0; i < n; i++) edges.push_back(a[i]);
sort(edges.begin(), edges.end());

auto pp = [&](int x, int y) {return make_pair(min(x, y), max(x, y));};
vector<vector<int> > y(m);
for (int i = 0; i < n; i++)
    if (a[i].first != a[i].second)
    {
        y[a[i].first].push_back(a[i].second);
        y[a[i].second].push_back(a[i].first);
    }
vector<int> p1(m), p2(m);
for (int i = 0; i < m; i++) p1[i] = i;
sort(p1.begin(), p1.end(),
      [&](int i, int j){return y[i].size() < y[j].size();});
for (int i = 0; i < m; i++) p2[p1[i]] = i;

for (int ii = 0; ii < m; ii++)
{
    int i = p1[ii];
    sort(y[i].begin(),y[i].end(),[&](int xx,int yy){return p2[xx]>p2[yy];});
    while (!y[i].empty() && p2[y[i].back()] < p2[i]) y[i].pop_back();
    for (int j = 0; j < y[i].size(); j++)
        for (int k = j + 1; k < y[i].size(); k++)
            if (binary_search(edges.begin(),edges.end(),pp(y[i][j],y[i][k])))
                ans = max(ans, 1LL * vct[i] * vct[y[i][j]] * vct[y[i][k]]);
}

```

```
    return ans;
}

int main()
{
    int n;
    fi>>n;
    vector<pair<int, int> > a(n);
    for (int i = 0; i < n; i++) fi>>a[i].first>>a[i].second;
    fo << solve(a) << endl;
    fo<<"\nTime: "<<clock() / (double)1000<<" sec";
    return 0;
}
```

