

Mục lục – Athena VII

TREAP 5

Các phép xử lý với Treap	6
Các phép xử lý cơ sở	6
<i>Khai báo cấu trúc phần tử của Treap</i>	6
<i>Merge (Hợp nhất Treaps)</i>	6
<i>split (Tách cây)</i>	7
Tìm kiếm trên Treap	8
<i>Bổ sung phần tử mới</i>	8
<i>Xóa phần tử của cây</i>	8
<i>Tìm phần tử</i>	9
Truy vấn theo nhóm phần tử	9
<i>Xử lý truy vấn trên toàn bộ cây</i>	10
<i>Xử lý truy vấn trên đoạn</i>	12
Cập nhật theo nhóm phần tử	12
<i>Cập nhật trên toàn cây</i>	12
<i>Cập nhật trên đoạn</i>	14
Phạm vi ứng dụng và hạn chế của cấu trúc Treap	14
<i>Ứng dụng</i>	14
<i>Hạn chế</i>	15
Treap với khóa ẩn	16
Ý tưởng chính	16
Khóa X	16
Đại lượng thay thế C	16
Các phép xử lý cấu trúc	17
Hàm split	17
Hàm merge	17
Đảm bảo tính nhất quán của c	18
Ứng dụng Treap với khóa ẩn	18
Tổ chức Treap với khóa ẩn trên C++	18
Bài tập ứng dụng	20
VU04. HAI BẢN ĐỒ <i>Tên chương trình: TWOMAPS.CPP</i>	20

Cáu trúc ROPE	30
Mô tả cấu trúc	30
Cộng xâu – merge	30
Xác định ký tự theo chỉ số - hàm get	31
Tách xâu – hàm split	31
Các phép xóa, bổ sung – delete và insert	32
Cân bằng cây	33
Các giải pháp nâng cao hiệu quả	33
BÀI TẬP	34
VT37. GIÁ ĐÈ DÀY <i>Tên chương trình: CUPBOARD.CPP</i>	34
VT38. ĐƯỜNG BẶC THANG <i>Tên chương trình: STAIRWAY.CPP</i>	37
VT39. IPHONE <i>Tên chương trình: IPHONE.CPP</i>	40
VT40. ĐIỂM THƯỞNG <i>Tên chương trình: BONUS.CPP</i>	42
VT41. NGÃ TƯ <i>Tên chương trình: CROSSY.CPP</i>	45
VT42. RUỘNG BẶC THANG <i>Tên chương trình: TERRACES.CPP</i>	50
VT43. GIẢM GIÁ <i>Tên chương trình: SALE.CPP</i>	55
VT44. ĐỊA LAN <i>Tên chương trình: CYMBIDIUM.CPP</i>	59
VT45. CÓC THÍ NGHIỆM <i>Tên chương trình: GLASSES.CPP</i>	62
VT46. SÓ KHẢ NĂNG <i>Tên chương trình: CASES.CPP</i>	65
VT47. BÀN PHÍM <i>Tên chương trình: KEYBOARD.CPP</i>	69
VT48. SỔ HÓA TÀI LIỆU <i>Tên chương trình: DOCUMENTS.CPP</i>	73
VT49. SỐ BUỒN TẺ <i>Tên chương trình: BORING.CPP</i>	77
VT50. TẢO BIỂN <i>Tên chương trình: SEAWEED.CPP</i>	80
VU01. ẢO THUẬT <i>Tên chương trình: TRICK.CPP</i>	83
VU02. CẤP ĐIỂM <i>Tên chương trình: PAIRS.CPP</i>	89
VU03. NHện <i>Tên chương trình: SPIDERS.CPP</i>	92
VU05. SỐ DƯ NHỎ NHẤT <i>Tên chương trình: MINRMD.CPP</i>	95
VU07. TUYẾN Ô TÔ BUÝT NHANH <i>Tên chương trình: BUS.CPP</i>	102
VU08. CÁ HỒI <i>Tên chương trình: SALMON.CPP</i>	107
VU09. CAMERA BAY <i>Tên chương trình: DRONE.CPP</i>	111
VU10. SỐ NHỎ NHẤT <i>Tên chương trình: MINNUM.CPP</i>	118
VU11. ƯỚC SỐ <i>Tên chương trình: DIVISORS.CPP</i>	120

VU12. NÓI CHUYỆN	<i>Tên chương trình: TALK.CPP</i>	124
VU13. ĐỖ XE	<i>Tên chương trình: PARKING.CPP</i>	129
VU14. XE BUÝT NHANH	<i>Tên chương trình: BRT.CPP</i>	132
VU15. BONSAI BAY	<i>Tên chương trình: BONSAI.CPP</i>	135
VU16. HÀNH LÝ	<i>Tên chương trình: BAGGAGE.CPP</i>	137
VU17. TOUR DU LỊCH	<i>Tên chương trình: TOUR.CPP</i>	139
VU18. PHÂN LỚP	<i>Tên chương trình: SUBCLASS.CPP</i>	142
VU19. DÂY HÌNH VUÔNG	<i>Tên chương trình: SQUARES.CPP</i>	144
VU20. PHÂN SỐ	<i>Tên chương trình: FRACTION.CPP</i>	147
VU21. MỨC ĐỘ GIÓNG NHAU	<i>Tên chương trình: SAMENESS.CPP</i>	150
VU22. MÚA TẬP THỂ	<i>Tên chương trình: DANCERS.CPP</i>	153
VU23. BẢNG GIÁ	<i>Tên chương trình: PRICES.CPP</i>	156
VU24. GIẢI THƯỞNG	<i>Tên chương trình: PRIZE.CPP</i>	159
VU25. ĐƯỜNG VÀNH ĐAI	<i>Tên chương trình: CIRCLE.CPP</i>	163
VU26. DI CỤ	<i>Tên chương trình: MIGRATION.CPP</i>	166
VU27. Ô ĐỎ CHỮ	<i>Tên chương trình: CROSSW.CPP</i>	173
VU28. CƠ HỘI CUỐI CÙNG	<i>Tên chương trình: OPPORTUNITY.CPP</i>	176
VU29. CHỮ THẬP	<i>Tên chương trình: PLUSES.CPP</i>	179
VU30. LÀNG CỔ	<i>Tên chương trình: VILLAGE.CPP</i>	182
VU31. KHUNG BIỂN QUẢNG CÁO	<i>Tên chương trình: FRAME.CPP</i>	185
VU32. NGUỒN TIN	<i>Tên chương trình: SOURCES.CPP</i>	188
VU33. KHÓA SỐ	<i>Tên chương trình: MAXSTR.CPP</i>	190
VU34. MÃ SỐ	<i>Tên chương trình: SAFEKEY.CPP</i>	193
Quy hoạch động		197
Nguyên lý Bellman		197
Bài 1a. VỀ ĐÍCH	<i>Tên chương trình: FINISH.CPP</i>	197
Các bước xử lý		202
Bài 1b.		205
Các bài tập ứng dụng giải thuật quy hoạch động		207
Bài 2. NÓI ĐIỂM	<i>Tên chương trình: JOIN.CPP</i>	207
Bài 3. DÂY KÝ TỰ CON CHUNG DÀI NHẤT	<i>Tên chương trình: LCS.CPP</i>	210
Quy hoạch động đơn giản		213

SÓ CATALAN	<i>Tên chương trình: CATALAN.CPP</i>	214
LÁT ĐƯỜNG VIỀN	<i>Tên chương trình: PAVE.CPP</i>	217
HAI SỐ 1	<i>Tên chương trình: TWOONE.CPP</i>	220
MÁY ATM	<i>Tên chương trình: ATM.CPP</i>	223

TREAP

Treap là cấu trúc dữ liệu kết hợp giữa cây tìm kiếm nhị phân với vun đống nhị phân, vì vậy tên gọi là sự kết hợp tên của hai cấu trúc trên (*Treap = Tree + Heap*).

Cấu trúc dữ liệu này lưu trữ các cặp (x, y) dưới dạng tạo thành cây tìm kiếm nhị phân theo x và là vun đống nhị phân theo y . Giả thiết tất cả các x và tất cả các y khác nhau. Khi đó nếu trong cấu trúc có nút lưu trữ cặp (x_0, y_0) thì ở các nút của cây con trái của nút này đều có x nhỏ hơn x_0 , ở các nút của cây con phải của nút này đều có x lớn hơn x_0 , ngoài ra, ở cả cây con phải lẫn cây con trái giá trị y ở các nút đều nhỏ hơn y_0 .

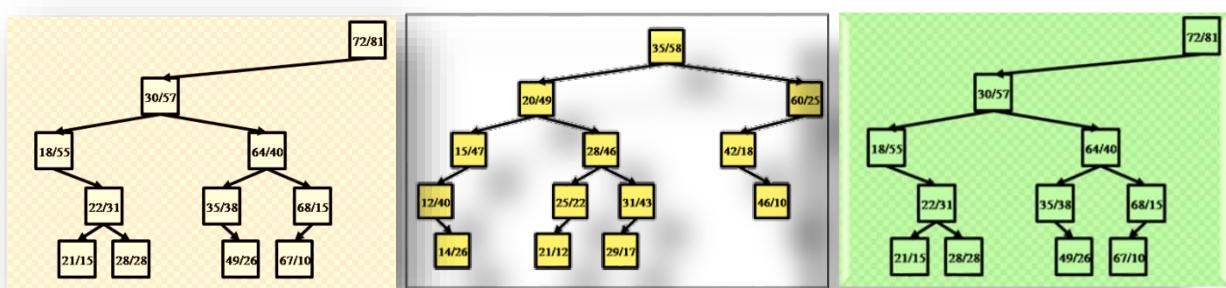
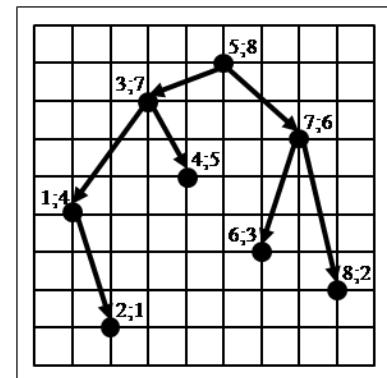
Treap do Siedel và Argon đề xuất năm 1996.

Trong phần lớn các ứng dụng Treap, x là khóa (*key*), đồng thời cũng là giá trị lưu trữ trong cấu trúc, còn y là độ ưu tiên (*priority*). Nếu như không có độ ưu tiên y , với bộ dữ liệu x cho trước ta sẽ có một tập các cây tìm kiếm nhị phân, trong số đó một số cây suy biến thành nhánh và làm chậm một cách đáng kể quá trình tìm kiếm.

Việc đưa mức độ ưu tiên vào sẽ cho phép xác định cây một cách đơn giản, không phụ thuộc vào trình tự bổ sung các nút. Ta có thể xây dựng cây nhị phân không suy biến, đảm bảo thời gian tìm kiếm trung bình là $O(\log n)$, bởi vì khi bổ sung nút với độ ưu tiên ngẫu nhiên xác xuất nhận được cây có độ cao lớn hơn $[4\log_2 n]$ là vô cùng bé.

Cặp giá trị (x, y) có thể xét như tọa độ một điểm trên mặt phẳng. Khi đó Treap là một cây đồ thị phẳng (các cạnh không giao nhau) nối các điểm trên mặt phẳng. Vì vậy Treap còn được gọi là Cây Đề các (*Carsian Tree*).

Ví dụ: Một số cấu trúc Treap:



Các phép xử lý với Treap

Treap cho phép tổ chức xử lý như với cây tìm kiếm nhị phân bình thường, nhưng trong phần lớn các trường hợp thời gian thực hiện mỗi phép xử lý đều là $O(\log n)$:

- ♣ **void insert(T1 key, T2 value)** – bổ sung cặp giá trị (**key, value**) vào Treap,
- ♣ **void remove(T1 key)** – xóa các nút có khóa bằng **key** khỏi Treap,
- ♣ **T2 find(T1 key)** – tìm giá trị xác định bởi khóa **key**.

Cấu trúc Treap cũng cho phép dễ dàng thực hiện các phép chọn và cập nhật theo nhóm:

- ♠ **T2 query(T1 key1, T1 key2)** – xác định giá trị của một hàm (tổng, tích, max, . . .) với các phần tử trong khoảng [**key1, key2**],
- ♠ **void modify(T1 key1, T1 key2)** – cập nhật giá trị (tăng thêm một lượng, nhân với một số, xác lập một thuộc tính, . . .) với tất cả các phần tử trong khoảng [**key1, key2**].

Tồn tại nhiều phép xử lý phức tạp hơn như hợp nhất các Treap, phân chia một Treap thành vài Treaps riêng biệt, . . . đều được thực hiện dựa trên các phép xử lý cơ sở sẽ xét chi tiết dưới đây.

Các phép xử lý cơ sở

Khai báo cấu trúc phần tử của Treap

```
struct TreapNode
{
    int k, p, v;
    TreapNode *l, *r;
    TreapNode(int key, int value)
    {
        k = key;
        v = value;
        p = rand();
        l = r = NULL;
    }
};
```

Merge (Hợp nhất Treaps)

Xét Treap **A** có gốc là **a** và Treap **B** có gốc là **b**. Giả thiết các khóa của **A** đều nhỏ hơn khóa bất kỳ của **B** (hoặc không lớn hơn nếu tổ chức Treap với các phần tử có thể có khóa giống nhau). Cần tạo Treap **T** mới từ các phần tử của **A** và **B**, trả về con trỏ chỉ tới gốc của **T**:

```

TreapNode *merge(TreapNode *a, TreapNode *b)
{
    if (!a || !b)
        return a ? a : b;
    if (a->p > b->p)
    {
        a->r = merge(a->r, b);
        return a;
    } else
    {
        b->l = merge(a, b->l);
        return b;
    }
}

```

Giả thiết khóa của các phần tử 2 cây đều khác nhau. Khi đó gốc của **T** sẽ là phần tử của **A** hoặc **B** có độ ưu tiên lớn nhất. Để dàng thấy rằng phần tử đó là **a** hoặc **b**. Nếu **a->p > b->p** thì **a** là gốc của cây hợp nhất. Các phần tử của **B** đều có khóa lớn hơn **a->k** vì vậy chúng sẽ được gắn vào cây con **a->r**. Như vậy bài toán ban đầu được dẫn về bài toán hợp nhất 2 cây **a->r** với **B**. Không khó để nhận thấy là quy trình xử lý trên có thể thể hiện bằng sơ đồ đệ quy. Đệ quy kết thúc khi một trong 2 cây là rỗng. Trường hợp gốc là **b** được xử lý tương tự.

split (Tách cây)

Cho số **k** và cây **T** với gốc là **t**. Yêu cầu tạo 2 cây: cây **A** gồm các phần tử của **T** có khóa nhỏ hơn **k** và cây **B** – chứa các phần tử còn lại của **T**. Hàm trả về giá trị 2 con trỏ chỉ tới gốc các cây.

Nếu **t->k < k** thì **t** thuộc **A**, trong trường hợp ngược lại – **t** thuộc **B**. Giả thiết **t** thuộc **A**, khi đó **t** sẽ là gốc của **A**. Khi đó các phần tử của cây con **t->l** sẽ là một phần của **A** vì có khóa nhỏ hơn **t->k** và do đó – nhỏ hơn **k**. Với cây con **t->r** tình hình phức tạp hơn: cây con có thể chứa các phần tử có khóa nhỏ hơn **k** lẫn các phần tử có khóa không nhỏ hơn **k**. Ta có thể xử lý theo sơ đồ đệ quy: tách cây con **t->r** theo **k**, nhận được 2 cây **A'** và **B'**. Treap **A'** được đặt thế chỗ cho **t->r**, khi đó **T** sẽ chỉ chứa các phần tử có khóa nhỏ hơn **k**. **B'** chứa các phần tử có khóa không nhỏ hơn **k**. Như vậy kết quả tách cây sẽ là **T** và **B'**. Trường hợp **t** thuộc **B** – xử lý tương tự. Đệ quy kết thúc khi **T** trở thành rỗng.

```

void split(TreapNode *t, int k, TreapNode *&a, TreapNode *&b)
{
    if (!t)
        a = b = NULL;
    else if (t->k < k)
    {
        split(t->r, k, t->r, b);
        a = t;
    } else
    {
        split(t->l, k, a, t->l);
        b = t;
    }
}

```

Tìm kiếm trên Treap

Việc tìm kiếm trên Treap được thực hiện theo đúng sơ đồ tìm kiếm trên cây nhị phân. Dễ dàng thấy rằng độ phức tạp của quá trình tìm kiếm không vượt quá $O(\log n)$. Như vậy Treap có thể coi như cây tìm kiếm nhị phân cân bằng (*tuy bản thân Treap có cấu trúc phức tạp hơn*).

Luôn luôn giả thiết rằng con trỏ `TreapNode *t` chỉ tới gốc của Treap.

Bổ sung phần tử mới

Xét việc bổ sung phần tử với khóa **key** và giá trị **value** vào cây **T**. Phần tử này có thể xem như một Treap **Tn** hoàn chỉnh bao gồm một nút. Như vậy có thể tiến hành hợp nhất hai cây. Tuy vậy, đầu tiên phải xử lý sơ bộ, chia **T** thành 2 phần: **T1** chứa các phần tử có khóa nhỏ hơn key và **T2** – chứa các phần tử còn lại, sau đó thực hiện hai phép hợp nhất và hoàn thành việc bổ sung phần tử mới vào cây.

```

void insert(int key, int value)
{
    TreapNode *tn = new TreapNode(key, value), *t1, *t2;
    split(t, key, t1, t2);
    t = merge(t1, tn);
    t = merge(t, t2);
}

```

Xóa phần tử của cây

Giả thiết phải loại bỏ khỏi **T** tất cả các phần tử có khóa bằng **key**. Tách **T** thành 2 cây **T1** và **T2** theo khóa **key**, **T1** chứa các phần tử có khóa nhỏ hơn **key**, **T2** chứa các phần tử còn lại. Tách **T2** theo khóa **key+1**, nhánh phải (**T3**) của cây nhận được

chứa các phần tử có khóa lớn hơn key, còn trong **T2** chỉ có các phần tử có khóa bằng **key**. Hợp nhát **T1** và **T3** ta có kết quả cần tìm.

Nếu cần giải phóng bộ nhớ của các phần tử bị xóa ta cần gọi hàm **dispose**:

```
void dispose(TreapNode *n)
{
    if (n == NULL)
        return;
    dispose(n->l);
    dispose(n->r);
    delete n;
}
```

```
void remove(int key)
{
    TreapNode *t1, *t2, *t3;
    split(t, key, t1, t2);
    split(t2, key + 1, t2, t3);
    t = merge(t1, t3);
    dispose(t2);
}
```

Tìm phần tử

Việc tìm kiếm một phần tử với khóa cho trước trên Treap có thể thực hiện theo đúng sơ đồ tìm kiếm trên cây nhị phân bình thường.

Mặt khác có thể sử dụng các công cụ **merge()** và **split()** đã xây dựng ở trên để tìm kiếm. Phần của cây có thể chứa phần tử cần tìm với khóa cho trước có thể xác định bằng hai lát cắt. Sau khi trích hoặc xử lý kết quả cần khôi phục lại cây (bằng hai phép ghép). Cách xử lý này đặc biệt có hiệu quả với các truy vấn xử lý nhóm phần tử. Nếu phần tử cần tìm không tồn tại kết quả sẽ là 0.

```
int find(int key)
{
    int res = 0;
    TreapNode *t1, *t2, *t3;
    split(t, key, t1, t2);
    split(t2, key + 1, t2, t3);
    if (t2 != NULL)
        res = t2->v;
    t1 = merge(t1, t2);
    t = merge(t1, t3);
    return res;
}
```

Truy vấn theo nhóm phần tử

Các truy vấn đòi hỏi phải làm việc với nhiều phần tử như tính tổng hoặc tích, đếm số lượng, tìm giá trị max hoặc min, . . . có thể dễ dàng thực hiện trên Treap.

Xử lý truy vấn trên toàn bộ cây

Giả thiết cần tính tổng các phần tử, đồng thời xác định số lượng phần tử.

Trong cấu trúc phần tử cần thay đổi: bổ sung thêm 2 trường **cnt** và **sum** để lưu số lượng và tổng của các phần tử thuộc cây con có gốc là nút đang xét. Trong khai báo cấu trúc các trường này cần được khởi tạo là 1 và giá trị của phần tử.

```
struct TreapNode
{
    int k, p, v;
    int sum, cnt;
    TreapNode *l, *r;
    TreapNode(int key, int value)
    {
        k = key;
        v = value;
        p = rand();
        l = r = NULL;
        sum = value;
        cnt = 1;
    }
};
```

Ngoài ra, cần có thêm các hàm **get()**, **sum()** và **update()** phục vụ cập nhật các trường này trong quá trình xây dựng cây.

```
int getSum(TreapNode *n)
{
    return n != NULL ? n->sum : 0;
}
```

```
int getCnt(TreapNode *n)
{
    return n != NULL ? n->cnt : 0;
}
```

```

void update(TreapNode *n)
{
    if (n == NULL) return;
    n->sum = v + getSum(n->l) + getSum(n->r);
    n->cnt = 1 + getCnt(n->l) + getCnt(n->r);
}

```

Các phép **merge()** và **split()** cũng cần được thay đổi thích hợp để phục vụ cho các trường mới đưa vào. Với mỗi nhánh cây nhận được cần gọi hàm **update()** để chỉnh lý giá trị các trường **cnt** và **sum**.

```

TreapNode *merge(TreapNode *a, TreapNode *b)
{
    if (!a || !b)
        return a ? a : b;
    if (a->p > b->p) {
        a->r = merge(a->r, b);
        update(a);
        return a;
    } else {
        b->l = merge(a, b->l);
        update(b);
        return b;
    }
}

```

```

void split(TreapNode *t, int k,
           TreapNode *&a, TreapNode *&b)
{
    if (!t)
        a = b = NULL;
    else if (t->k < k) {
        split(t->r, k, t->r, b);
        a = t;
    } else {
        split(t->l, k, a, t->l);
        b = t;
    }
    update(a); update(b);
}

```

Nếu cần tìm số lượng hay tổng các phần tử chỉ cần truy nhập tới trường tương ứng của phần tử gốc.

Xử lý truy vấn trên đoạn

Giả thiết cần đếm số phần tử hoặc tổng các phần tử có khóa nằm trên đoạn $[k_1, k_2]$. Bằng hai lát cắt (tương tự như khi tìm hoặc xóa phần tử) ta có thể xác định nhánh cây chứa các phần tử cần tìm. Việc xác định giá trị quan tâm được thực hiện như đã nêu ở mục trên.

```
int rangeSum(int k1, int k2)
{
    TreapNode *t1, *t2, *t3;
    split(t, k1, t1, t2);
    split(t2, k2 + 1, t2, t3);
    int s = getSum(t2);
    t1 = merge(t1, t2);
    t = merge(t1, t3);
    return s;
}
```

Cập nhật theo nhóm phần tử

Các phép cập nhật theo nhóm phần tử có thể là thay các giá trị cũ bằng một giá trị mới, nhân một số với các giá trị ở nút, bổ sung thêm một thuộc tính nào đó, ...

Dưới đây ta sẽ xét việc cộng thêm một số vào các giá trị của nút trên toàn cây hoặc trên một đoạn. Các loại cập nhật khác được thực hiện theo kiểu tương tự.

Cập nhật trên toàn cây

Xét việc cộng thêm một số vào các giá trị ở tất cả các nút của cây.

Cần bổ sung vào cấu trúc **TreapNode** trường **add** lưu giá trị cần tăng. Trong khai báo cần gán giá trị đầu là 0 cho trường mới này.

```
struct TreapNode
{
    int k, p, v;
    int sum, cnt, add;
    TreapNode *l, *r;
    TreapNode(int key, int value)
    {
        k = key;
        v = value;
        p = rand();
        l = r = NULL;
        sum = value;
        cnt = 1;
        add = 0;
    }
};
```

Khi đó giá trị thực lưu ở nút gốc **t** của cây **T**:

Không đơn thuần là $t \rightarrow v$ mà sẽ là $t \rightarrow v + t \rightarrow add$.

Giá trị tròn **sum**: thay $t \rightarrow sum$ bằng $t \rightarrow sum + t \rightarrow add * t \rightarrow cnt$.

```
int getSum(TreapNode *n)
{
    return n != NULL ? n->sum + n->add * n->cnt : 0;
}
```

Để các hàm **merge()** và **split()** hoạt động đúng cần đảm bảo việc chuyển giao giá trị thực cho các tham số. Sự tồn tại của trường **add** có thể phá vỡ tính đúng đắn. Ví dụ $t \rightarrow add$ bằng 10, còn $t \rightarrow l \rightarrow add$ bằng 0, thì sau khi ngắt $t \rightarrow l$ khỏi t thông tin xác định là giá trị các phần tử của cây phải được cộng thêm 10 sẽ bị mất. Vì vậy trước khi thực hiện các phép **merge()** và **split()** cần chuyển giao thông tin này cho các nhánh sẽ nhận được bằng hàm **down()**.

```
void down(TreapNode *n)
{
    if (n == NULL)
        return;
    n->v += n->add;
    if (n->l != NULL)
        n->l->add += n->add;
    if (n->r != NULL)
        n->r->add += n->add;
    n->add = 0;
}
```

Các hàm **merge()** và **split()** sẽ có dạng:

```
TreapNode *merge(TreapNode *a, TreapNode *b)
{
    if (!a || !b)
        return a ? a : b;
    if (a->p > b->p)
    {
        down(a);
        a->r = merge(a->r, b);
        update(a);
        return a;
    } else
    {
        down(b);
        b->l = merge(a, b->l);
        update(b);
        return b;
    }
}
```

```

void split(TreapNode *t, int k,
           TreapNode *&a, TreapNode *&b)
{
    down(t);
    if (!t)
        a = b = NULL;
    else if (t->k < k)
    {
        split(t->r, k, t->r, b);
        a = t;
    } else
    {
        split(t->l, k, a, t->l);
        b = t;
    }
    update(a);
    update(b);
}

```

Với Treap được tổ chức như trên, nếu muốn cộng thêm một giá trị nào đó vào tất cả các phần tử của cây thì cần gán giá trị này cho trường **add** của phần tử gốc.

Cập nhật trên đoạn

Nếu như đã thiết kế Treap với khả năng cập nhật toàn bộ cây thì việc cập nhật trên đoạn [**k1**, **k2**] cũng sẽ dễ dàng thực hiện, tương tự như việc xử lý truy vấn trên đoạn.

```

void rangeAdd(int k1, int k2, int addVal)
{
    TreapNode *t1, *t2, *t3;
    split(t, k1, t1, t2);
    split(t2, k2 + 1, t2, t3);
    if (t2 != NULL)
        t2->add += addVal;
    t1 = merge(t1, t2);
    t = merge(t1, t3);
}

```

Phạm vi ứng dụng và hạn chế của cấu trúc Treap

Ứng dụng

- ✚ Cấu trúc Treap tương đối dễ lập trình, vì vậy nó là công cụ hữu hiệu giải nhiều loại bài toán đòi hỏi xử lý truy vấn ở các dạng:
- ✚ Các bài toán đòi hỏi tổ chức cây đủ cân bằng để tìm kiếm và cập nhật (ví dụ, tổ chức tập hợp, tổ chức từ điển),

- ✚ Các loại bài toán xử lý truy vấn trên một phạm vi nào đó, các bài toán cập nhật trên đoạn. Khác với cây quản lý đoạn, Treap cho phép xử lý truy vấn trên các tập thay đổi,
- ✚ Tính toán các số liệu thống kê với độ phức tạp $O(\log n)$,
- ✚ Phục vụ xây dựng một cấu trúc dữ liệu mạnh hơn – Treap với khóa ẩn.

Hạn chế

- ✓ Tốn bộ nhớ: mỗi nút của cây phải lưu trữ khá nhiều trường dữ liệu trung gian cho các loại truy vấn khác nhau,
- ✓ Việc thay đổi cấu trúc nút sẽ kéo theo sự thay đổi trong các phép xử lý cơ bản, điều này làm tăng độ phức tạp lập trình.

Treap với khóa ẩn

Ý tưởng chính

Cấu trúc dữ liệu vector trong C/C++ cho phép tổ chức mảng động với khả năng bổ sung phần tử mới vào cuối mảng hoặc xóa phần tử cuối trong danh sách. Cấu trúc này cũng cho phép xác định hoặc cập nhật giá trị của phần tử theo vị trí của nó.

Giả thiết ta cần một cấu trúc dữ liệu như vậy cùng với các khả năng mới để có thể bổ sung phần tử vào vị trí bất kỳ hoặc xóa một phần tử cùng với việc đánh số tự động lại các phần tử của mảng.

Cấu trúc dữ liệu với các tính chất đã nêu có thể xây dựng dựa trên cấu trúc Treap và được gọi là *Treap với khóa ẩn* (*Treap with implicit key*).

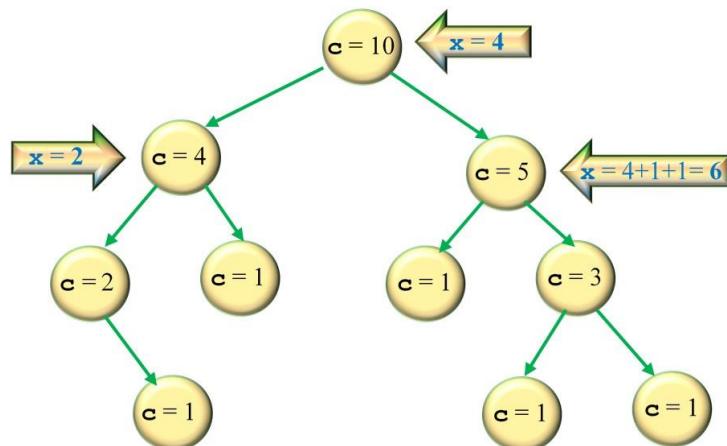
Khóa X

Như đã nói ở trên, Treap là cấu trúc dữ liệu kết hợp giữa cây nhị phân và vun đong nhị phân. Trong Treap khóa **x** phục vụ tổ chức cây nhị phân được lưu trữ tường minh. Nhưng người ta cũng có thể vòng tránh việc lưu trữ khóa tường minh bằng cách lấy *số lượng phần tử ở nhánh trái* của phần tử đang xét làm *khóa* tổ chức cây nhị phân. Như vậy, trong mỗi phần tử chỉ lưu trữ tường minh độ ưu tiên **y**, còn khóa sẽ là *số thứ tự* của phần tử trong cây *trù 1*.

Vấn đề phải giải quyết ở đây là khi bổ sung hay loại bỏ phần tử, số thứ tự các phần tử trong cây có thể bị thay đổi. Nếu không có cách tiếp cận hợp lý, độ phức tạp của các phép bổ sung, loại bỏ sẽ có độ phức tạp $O(n)$ và làm mất ưu việt của cây tìm kiếm nhị phân.

Đại lượng thay thế C

Lối thoát khỏi tình huống rắc rối trên là khá đơn giản. Thay vì lưu trữ tường minh **x** người ta lưu trữ **c** – *số lượng nút của cây con* có gốc là phần tử đang xét (bao gồm cả nút gốc). Đây là một đại lượng có miền xác định đủ nhỏ và dễ dàng dẫn xuất khi cần thiết. Lưu ý là các phép xử lý trong Treap đều thực hiện theo chiều từ trên xuống dưới. Nếu trong quá trình duyệt, đi từ gốc tới một đỉnh nào đó ta cộng số lượng nút và tăng thêm 1 của các nhánh trái bị bỏ qua thì khi tới nút cần xét sẽ có trong tay khóa của phần tử đó.



Các phép xử lý cấu trúc

Hai phép xử lý cấu trúc cây:

- ⊕ **merge()** – hợp nhất 2 cây, trong đó ở một cây các khóa **x** có giá trị nhỏ hơn giá trị khóa ở cây kia,
- ⊕ **split()** – tách một cây thành 2 cây, trong đó ở một cây các khóa **x** có giá trị nhỏ hơn giá trị khóa ở cây kia.

Trong Treap với khóa ẩn tham số cho hàm merge là gốc của 2 cây bất kỳ cần hợp nhất: **merge(root1, root)**. Lời gọi hàm **split** sẽ là **split(root, t)** xác định việc tách cây được thực hiện sao cho nhánh trái có **t** nút.

Hàm split

Xuất phát từ nút gốc, xét yêu cầu cắt **k** nút của một cây. Giả thiết nhánh trái của cây có **1** nút và nhánh phải – **r** nút. Có 2 trường hợp xảy ra:

- ♣ **$1 \geq k$** : khi đó cần gọi đệ quy split từ nút con trái của gốc với cùng tham số **k**, đồng thời biến phần phải của kết quả thành con trái của gốc, còn gốc của cây đang xử lý sẽ là phần phải của kết quả,
- ♣ **$1 < k$** : xử lý tương tự như trên nếu đổi vai trò của trái và phải, split được gọi đệ quy từ nút con phải với tham số **$k-1-1$** , phần trái của kết quả là nút con trái phải, còn gốc – thuộc phần trái kết quả.

Để dễ hiểu tư tưởng thuật toán, các bước xử lý được trình bày ở đoạn mã giả sau (giải thuật xử lý chính xác trên C++ với đầy đủ các hàm sẽ được xét ở phần cuối).

```
<Treap, Treap> split(Treap t, int k)
    int l = t.left.size
    if l >= k
        <t1, t2> = split(t.left, k)
        t.left = t2
        update(v)
        r = v
        return <t1, t2>
    else
        <t1, t2> = split(t.right, k - l - 1)
        t.right = t1
        update(v)
        l = v
        return <t1, t2>
```

Hàm merge

Trong quá trình hợp nhất không có sử dụng giá trị khóa x vì vậy hàm **merge** được xử lý như ở Treap bình thường.

Đảm bảo tính nhất quán của c

Sau mỗi thao tác xử lý đỉnh con cần ghi vào trường lưu trữ c tổng giá trị tương ứng của các nút con cộng thêm 1.

```
void update(Treap t)
t.size = 1 + t.left.size + t.right.size
```

Ứng dụng Treap với khóa ẩn

- ✚ Bổ sung phần tử mới vào bất kỳ vị trí nào trong cây đang xét đồng thời duy trì mọi tính chất của cây ban đầu,
- ✚ Di chuyển đoạn các phần tử của mảng sang nơi mới bất kỳ,
- ✚ Thực hiện các phép xử lý đổi với nhóm phần tử, kích thước nhóm thay đổi và được xác định ở mỗi lần xử lý, điều mà cấu trúc quản lý đoạn không đáp ứng được,
- ✚ Có thể thực hiện việc đổi chỗ các phần tử ở vị trí chẵn sang vị trí lẻ và ngược lại,
- ✚ Thực hiện các phép xử lý nêu trên một cách hiệu quả đối với xâu (*cấu trúc Rope*).

Tổ chức Treap với khóa ẩn trên C++

```
struct TN {
    int v, p, c, maxV;
    TN *l, *r;
    TN(int V) : v(V), p(rand()), c(1), maxV(V), l(0), r(0) {}
};

class ITreap {
    TN *root;
    int getMaxV(TN *n) {
        return n ? n->maxV : -(1 << 30);
    }
    int getC(TN *n) {
        return n ? n->c : 0;
    }
    void update(TN *n) {
        if (!n)
            return;
        n->c = 1 + getC(n->l) + getC(n->r);
        n->maxV = max(n->v, max(getMaxV(n->l), getMaxV(n->r)));
    }
    TN *merge(TN *a, TN *b) {
        if (!a || !b)
            return a ? a : b;
        if (a->p > b->p) {
            a->r = merge(a->r, b);
            update(a);
            return a;
        } else {

```

```

        b->l = merge(a, b->l);
        update(b);
        return b;
    }
}

void split(TN *t, int k, TN *&a, TN *&b) {
    if (!t) {
        a = b = 0;
        return;
    }
    if (getC(t->l) < k) {
        split(t->r, k - getC(t->l) - 1, t->r, b);
        a = t;
    } else {
        split(t->l, k, a, t->l);
        b = t;
    }
    update(a);
    update(b);
}
public:
    ITreap() : root(0) {}
    int size() {
        return getC(root);
    }
    void insert(int pos, int v) {
        TN *a, *b;
        split(root, pos, a, b);
        root = merge(a, merge(new TN(v), b));
    }
    int getMax(int l, int r) {
        TN *a, *b, *c;
        split(root, l, a, b);
        split(b, r - l + 1, b, c);
        int res = getMaxV(b);
        root = merge(a, merge(b, c));
        return res;
    }
}
};

```

Bài tập ứng dụng

VU04. HAI BẢN ĐỒ

Tên chương trình: TWOMAPS.CPP

Vết đứt gãy lớn trên vỏ trái đất *Banda Detachment* nằm ở phía tây Thái bình dương, độ sâu đáy biển ở đó đạt tới 7 km. Nhiều bức ảnh địa hình đã được chụp.

Để khảo sát vết đứt gãy lớn này người ta dùng một máy thăm dò tự động. Một trong số các khe nứt tạo thành một đường thẳng và là vùng thuộc kế hoạch thăm dò. Theo dự kiến máy thăm dò sẽ tiến hành đo đạc khảo cứu đoạn có độ dài s . Bộ nhớ của thiết bị thăm dò chỉ có thể chứa 2 bản đồ cho tổng độ dài là s . Nếu *một phần nào đó có cả 2 bản đồ thì chỉ tính là một*. Bộ phận chuẩn bị có 2 thao tác cơ bản:

- Thao tác **A** có dạng **1 1 x** – xin cung cấp bản đồ của khe nứt đoạn từ điểm 1 đến điểm **x**, $1 < x$,
- Thao tác **B** có dạng **2 k** – trả về bản đồ đã xin ở thao tác thứ k. Đảm bảo là thao tác thứ k thuộc loại A và không có việc một bản đồ bị trả về 2 lần.

Các thao tác được đánh số từ 1, có thể tồn tại nhiều bản đồ cùng chụp một đoạn (có **1** và **x** giống nhau). Có **n** thao tác được thực hiện. Sau mỗi thao tác hãy xác định số cách khác nhau chọn 2 bản đồ khác nhau cho tổng độ dài đúng bằng s . Hai cách chọn gọi là khác nhau nếu một bản đồ (xác định theo trình tự xin cung cấp) có ở cách chọn thứ nhất và không có ở cách chọn thứ 2. Ban đầu bộ phận chuẩn bị chưa có bản đồ nào trong tay.

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

- ✚ Dòng đầu tiên chứa 2 số nguyên **s** và **n** ($1 \leq s \leq 10^9$, $1 \leq n \leq 10^5$),
- ✚ Mỗi dòng trong **n** dòng tiếp theo chứa 2 hoặc 3 số nguyên xác định thao tác dạng **A** hoặc **B**, trong đó **1**, **x** có giá trị tuyệt đối không vượt quá 5×10^8 .

Kết quả: Đưa ra file văn bản TWOMAPS.OUT số cách chọn tính được sau mỗi thao tác, các kết quả đưa ra dưới dạng số nguyên, mỗi số trên một dòng.

Ví dụ:

TWOMAPS.INP	TWOMAPS.OUT
10 6	0
1 0 8	1
1 7 10	2
1 5 15	0
2 2	2
1 12 14	0
2 5	



VU04 Io20161105 J

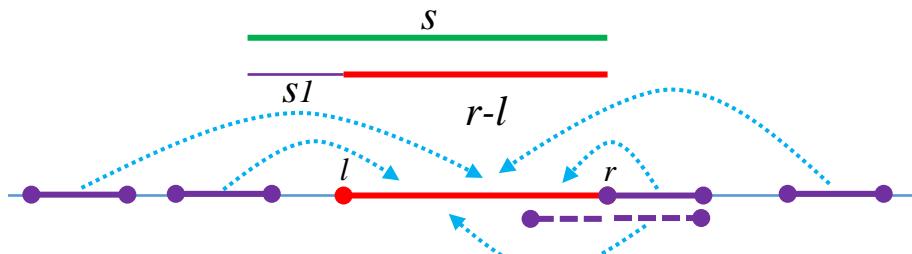
Giải thuật: Phương án A: Không sử dụng cấu trúc dữ liệu Treap.

Nhận xét:

Xét việc bổ sung bản đồ $[l, r]$,

Đặt $s1 = s - (r - l)$, có 3 trường hợp xảy ra:

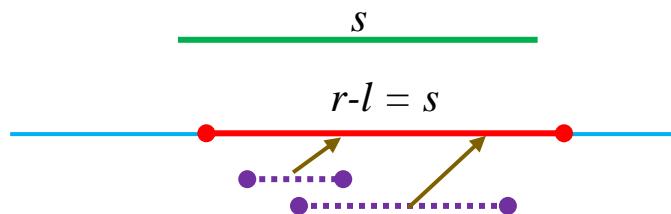
- ✚ $s1 < 0$: không có thêm lựa chọn mới,
- ✚ $s1 = 0$: Số lựa chọn bổ sung bằng số đoạn nằm gọn trong $[l, r]$,
- ✚ $s1 > 0$: Số lựa chọn bổ sung bằng tổng số đoạn có độ dài $s1$ nằm ngoài đoạn $[l, r]$, cộng với số lượng đoạn có giao khác rỗng với $[l, r]$ và tổng độ dài ngoài phần chung bằng $s1$.



Để quản lý các đoạn phục vụ tìm kiếm cần lưu trữ:

- ▣ Điểm đầu của đoạn và số lượng điểm đầu trùng với nó *theo khóa điểm cuối*,
- ▣ Điểm cuối của đoạn và số lượng điểm cuối trùng với nó *theo khóa điểm đầu*,
- ▣ Điểm đầu của đoạn và số lượng điểm đầu trùng với nó *theo khóa độ dài đoạn*,

Khi $r - l$ bằng s các thông tin lưu trữ nêu trên mới cho phép thống kê các đoạn cần tìm có điểm đầu hoặc điểm cuối trùng với đoạn đang xét.



Cần lưu trữ *tổng tiền tố* số lượng điểm đầu của các đoạn có độ dài không vượt quá $s - 2$, tức là các đoạn có thể nằm gọn trong đoạn độ dài s và không có chung điểm đầu và điểm cuối với đoạn độ dài s .

Tổ chức dữ liệu:

Để thuận tiện lập trình và dễ hiểu hơn khi chuyển sang phương án dùng Treap ta khai báo một *cấu trúc dữ liệu hướng đối tượng (OOP)*, gắn các phép xử lý với dữ liệu. Điều này cho phép dù dữ liệu được lồng vào cấu trúc lưu trữ chuẩn nào, dạng lời gọi tới phép xử lý vẫn giữ nguyên.

```

class MSet
{
    vector < pair < int , int > > a;
public:
    void add(int p, int v)
    {
        for (auto & q : a)
            if (q.first == p)
            {
                q.second += v;
                return;
            }
        a.emplace_back(p, v);
    }
    int get(int p)
    {
        int s = 0;
        for (auto q : a)
            if (q.first < p)
                s += q.second;
        return s;
    }
    int get(int l, int r)
    {
        int s = 0;
        for (auto q : a)
            if (l <= q.first && q.first <= r)
                s += q.second;
        return s;
    }
};

```

Tạo cặp dữ liệu mới
và bổ sung vào cuối

Hai kiểu truy xuất
kết quả

Khai báo dữ liệu:

Lưu điểm cuối và số lượng
(khóa: điểm đầu)

```
map < int , MSet > fs, fl, fr;
```

Lưu điểm đầu và số lượng
(khóa: độ dài đoạn)

Lưu điểm đầu và số lượng
(khóa: điểm cuối)

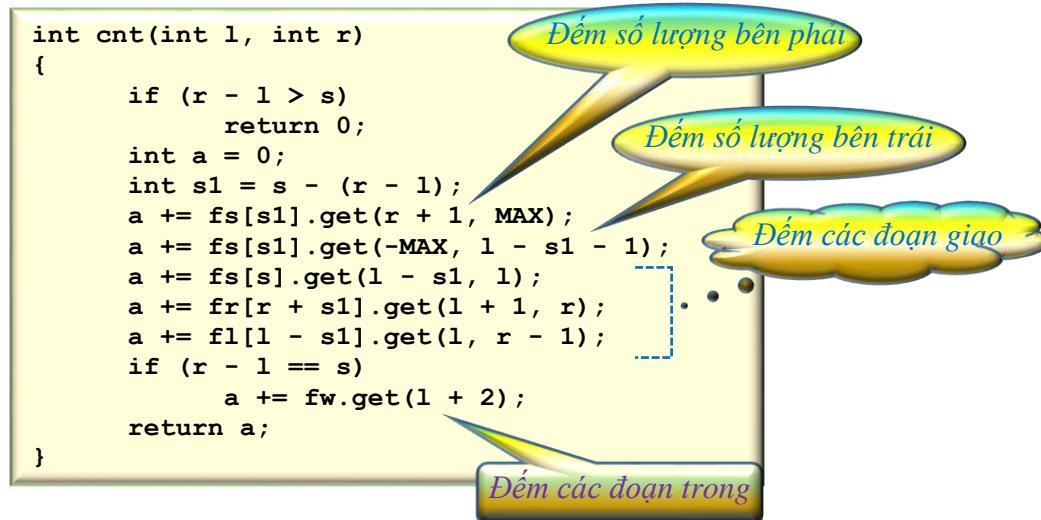
Tích lũy tổng tiền tố
(khóa: điểm đầu, cuối)

```
MSet fw;
```

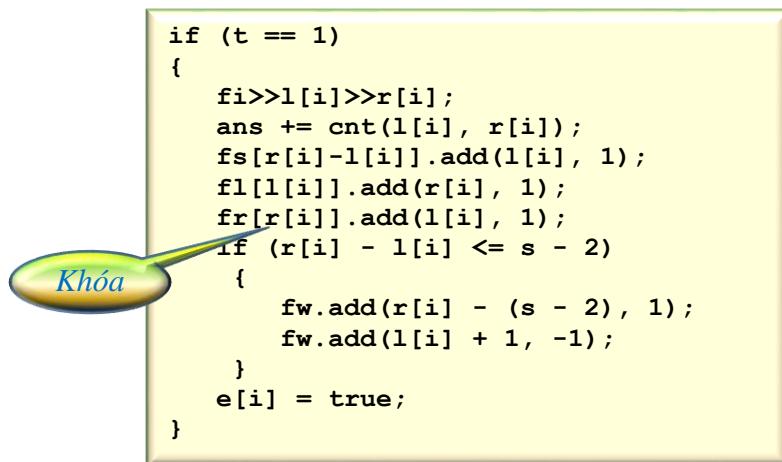
Xử lý:

Thêm bản đồ mới $[l_i, r_i]$:

- Cập nhật kết quả: **ans+=cnt(l[i], r[i]);**



- Cập nhật các mảng ghi nhận dữ liệu:



Việc loại bỏ bản đồ: được thực hiện tương tự, nhưng cập nhật thông tin trước, sau đó tính lại kết quả.

Dộ phức tạp của giải thuật: $O(n^2)$.

Chương trình: Giải thuật A (Không sử dụng Treap)

```
#include <bits/stdc++.h>
#define NAME "twomaps."
using namespace std;
typedef long long ll;
ifstream fi (NAME"inp");
ofstream fo (NAME"out");
const int MAX = 5e8;
const int N = 100002;

class MSet
{
    vector < pair < int , int > > a;
public:
    void add(int p, int v)
    {
        for (auto & q : a)
            if (q.first == p)
            {
                q.second += v;
                return;
            }
        a.emplace_back(p, v);
    }
    int get(int p)
    {
        int s = 0;
        for (auto q : a)
            if (q.first < p)
                s += q.second;
        return s;
    }
    int get(int l, int r)
    {
        int s = 0;
        for (auto q : a)
            if (l <= q.first && q.first <= r)
                s += q.second;
        return s;
    }
};

map < int , MSet > fs, fl, fr;
MSet fw;

int n, s, l[N], r[N];
ll ans = 0;
bool e[N];

int cnt(int l, int r)
{
    if (r - l > s)
        return 0;
    int a = 0;
    int s1 = s - (r - l);
    a += fs[s1].get(r + 1, MAX);
    a += fs[s1].get(-MAX, l - s1 - 1);
    a += fs[s].get(l - s1, l);
}
```

```

a += fr[r + s1].get(l + 1, r);
a += fl[l - s1].get(l, r - 1);
if (r - l == s)
    a += fw.get(l + 2);
return a;
}

int main()
{
    fi>>s>>n;
    for (int i = 0; i < n; ++i)
    {
        int t;
        fi>>t;
        if (t == 1)
        {
            fi>>l[i]>>r[i];
            ans += cnt(l[i], r[i]);
            fs[r[i] - l[i]].add(l[i], 1);
            fl[l[i]].add(r[i], 1);
            fr[r[i]].add(l[i], 1);
            if (r[i] - l[i] <= s - 2)
            {
                fw.add(r[i] - (s - 2), 1);
                fw.add(l[i] + 1, -1);
            }
            e[i] = true;
        } else {
            int k;
            fi>>k;
            --k;
            e[k] = false;
            fs[r[k] - l[k]].add(l[k], -1);
            fl[l[k]].add(r[k], -1);
            fr[r[k]].add(l[k], -1);
            if (r[k] - l[k] <= s - 2)
            {
                fw.add(r[k] - (s - 2), -1);
                fw.add(l[k] + 1, 1);
            }
            ans -= cnt(l[k], r[k]);
        }
        fo<<ans<<'\\n';
    }
    fo<<"\\nTime: "<<clock() / (double)1000<<" sec";
    return 0;
}

```

Giải thuật: Phương án B: Sử dụng cấu trúc dữ liệu Treap.

Nhận xét:

Độ phức tạp của phương án A cao vì không thể chèn dữ liệu mới vào chỗ cần thiết trong các vectors để đảm bảo các dữ liệu luôn luôn được sắp xếp, từ đó có thể áp dụng các phương pháp tìm kiếm nhanh,

Giải pháp:

- Tạo chức Treap với trường **sum** phục vụ tính tổng tiền tố,
- Gắn các phép xử lý Treap với cấu trúc.
- Cấu trúc Treap và các phép xử lý: theo đúng sơ đồ lý thuyết đã nêu.

Lớp dữ liệu Mset được xác định như sau:

```
class MSet
{
    Treap * t;
public:
    MSet() : t(Treap::null) {}
    void add(int p, int v)
    {
        if (!tadd(t, p, v))
        {
            Treap * t1, * t2;
            split(t, p, t1, t2);
            t = merge(merge(t1, new Treap(p, v)), t2);
        }
    }
    int get(int p)
    {
        Treap * t1, * t2;
        split(t, p, t1, t2);
        int w = t1->sum;
        t = merge(t1, t2);
        return w;
    }
    int get(int l, int r)
    {
        Treap * t1, * t2, * t3;
        split(t, l, t1, t2);
        split(t2, r + 1, t2, t3);
        int w = t2->sum;
        t = merge(t1, merge(t2, t3));
        return w;
    }
};
```

Khai báo dữ liệu trong chương trình và sơ đồ xử lý: giữ nguyên như ở phương án A.

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

Chương trình: Giải thuật B – Sử dụng cấu trúc dữ liệu Treap.

```
#include <bits/stdc++.h>
#define NAME "twomaps."
using namespace std;
typedef long long ll;
ifstream fi (NAME"inp");
ofstream fo (NAME"out");
const int MAX = 5e8;
const int N = 100179;

struct Treap
{
    static Treap * null;
    int x, y, val, sum;
    Treap * l, * r;
    Treap(const char *): x(0), y(0), val(0), sum(0), l(this), r(this) {}
    Treap(int nx, int nval = 0):x(nx), y(rand()), val(nval), sum(nval),
    l(null), r(null) {}

    void update()
    {
        sum = l->sum + val + r->sum;
    }
};

Treap * Treap::null = new Treap("...");

Treap * merge(Treap * l, Treap * r)
{
    if (l == Treap::null) return r;
    if (r == Treap::null) return l;
    if (l->y > r->y)
    {
        l->r = merge(l->r, r);
        l->update();
        return l;
    } else {
        r->l = merge(l, r->l);
        r->update();
        return r;
    }
}

void split(Treap * t, int x, Treap *& l, Treap *& r)
{
    if (t == Treap::null)
    {
        l = r = t;
        return;
    }
    if (t->x < x)
    {
        l = t;
        split(t->r, x, t->r, r);
    } else {
        r = t;
        split(t->l, x, l, t->l);
    }
    t->update();
}
```

```

bool tadd(Treap * t, int x, int v)
{
    if (t == Treap::null)
        return false;
    if (t->x == x)
    {
        t->val += v;
        t->sum += v;
        return true;
    } else if (x < t->x)
    {
        if (tadd(t->l, x, v) )
        {
            t->sum += v;
            return true;
        }
        return false;
    } else {
        if (tadd(t->r, x, v) )
        {
            t->sum += v;
            return true;
        }
        return false;
    }
}
class MSet
{
    Treap * t;
public:
    MSet() : t(Treap::null) {}
    void add(int p, int v)
    {
        if (!tadd(t, p, v) )
        {
            Treap * t1, * t2;
            split(t, p, t1, t2);
            t = merge(merge(t1, new Treap(p, v)), t2);
        }
    }
    int get(int p)
    {
        Treap * t1, * t2;
        split(t, p, t1, t2);
        int w = t1->sum;
        t = merge(t1, t2);
        return w;
    }
    int get(int l, int r)
    {
        Treap * t1, * t2, * t3;
        split(t, l, t1, t2);
        split(t2, r + 1, t2, t3);
        int w = t2->sum;
        t = merge(t1, merge(t2, t3));
        return w;
    }
};

```

```

map < int , MSet > fs, fl, fr;
MSet fw;
int n, s, l[N], r[N];
ll ans = 0;
bool e[N];
int cnt(int l, int r)
{
    if (r - l > s)
        return 0;
    int a = 0;
    int s1 = s - (r - l);
    a += fs[s1].get(r + 1, MAX);
    a += fs[s1].get(-MAX, l - s1 - 1);
    a += fs[s].get(l - s1, l);
    a += fr[r + s1].get(l + 1, r);
    a += fl[l - s1].get(l, r - 1);
    if (r - l == s)
        a += fw.get(l + 2);
    return a;
}
int main()
{
    fi>>s>>n;
    for (int i = 0; i < n; ++i)
    {
        int t;
        fi>>t;
        if (t == 1)
        {
            fi>>l[i]>>r[i];
            ans += cnt(l[i], r[i]);
            fs[r[i] - l[i]].add(l[i], 1);
            fl[l[i]].add(r[i], 1);
            fr[r[i]].add(l[i], 1);
            if (r[i] - l[i] <= s - 2)
            {
                fw.add(r[i] - (s - 2), 1);
                fw.add(l[i] + 1, -1);
            }
            e[i] = true;
        } else {
            int k;
            fi>>k; --k;
            e[k] = false;
            fs[r[k] - l[k]].add(l[k], -1);
            fl[l[k]].add(r[k], -1);
            fr[r[k]].add(l[k], -1);
            if (r[k] - l[k] <= s - 2)
            {
                fw.add(r[k] - (s - 2), -1);
                fw.add(l[k] + 1, 1);
            }
            ans -= cnt(l[k], r[k]);
        }
        fo<<ans<<'\n';
    }
    fo<<"\nTime: "<<clock() / (double)1000<<" sec";
    return 0;
}

```



Cấu trúc ROPE

Rope là cấu trúc dữ liệu lưu trữ xâu dưới dạng cây nhị phân cân bằng cho phép thực hiện các phép bổ sung, loại bỏ, kết nối xâu với độ phức tạp $O(\log n)$.

Trong một số các trường hợp, khi làm việc với xâu ta cần:

- ✚ Thực hiện nhanh các phép ghép nối, trích xâu con,...
- ✚ Với các xâu dài: thời gian thực hiện các phép xử lý trên không tỷ lệ với độ dài của xâu,
- ✚ Lưu trữ trạng thái và quay về trạng thái cũ khi cần thiết.

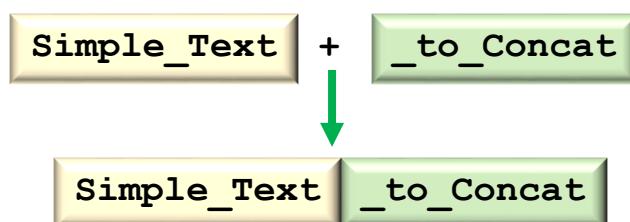
Việc lưu trữ tuyến tính các ký tự của xâu không đáp ứng được các yêu cầu nêu trên.

Mô tả cấu trúc

Xâu được lưu trữ theo mô hình Treap với khóa ẩn. Mỗi lá của cây sẽ lưu đoạn liên tục các ký tự của xâu và độ dài đoạn đó. Ở các nút trong – lưu tổng độ dài các lá của cây con. Ban đầu cây chỉ bao gồm một đỉnh lưu xâu đang xét. Dựa vào thông tin ở các đỉnh trong có thể tìm các ký tự của xâu theo chỉ số. Không cần lưu thông tin đánh dấu lá vì mỗi nút trong có đúng 2 nút con, còn lá – không có nút con. Vì vậy để nhận dạng lá chỉ cần kiểm tra một nút có nút con hay không.

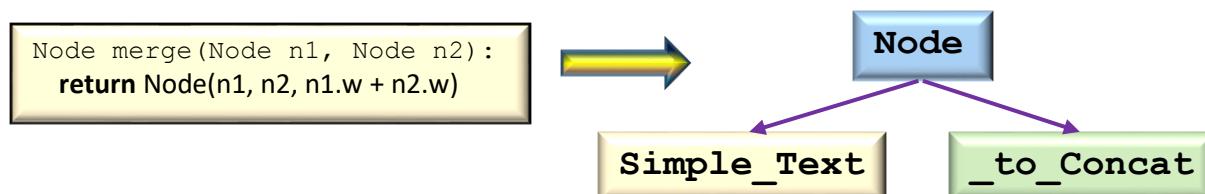
Cộng xâu – merge

Giả thiết ta cần hợp nhất 2 xâu, tạo xâu mới kết quả ghép liên tiếp 2 xâu ban đầu:

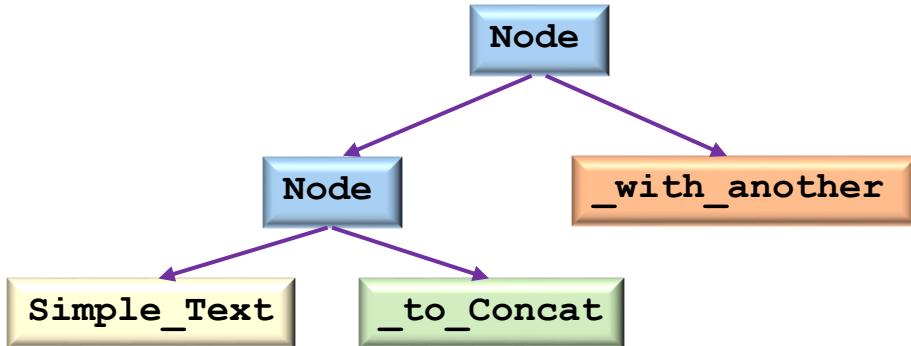


Việc hợp nhất xâu bình thường đòi hỏi dự trữ bộ nhớ cho xâu kết quả, lần lượt copy các ký tự của xâu thứ nhất và sau đó – các ký tự của xâu thứ 2 vào xâu kết quả. Độ phức tạp của giải thuật là $O(n)$.

Với cấu trúc cây, kết nối được thực hiện với độ phức tạp $O(1)$ và chỉ cần thêm bộ nhớ hỗ trợ tổ chức cây.



Độ phức tạp xử lý vẫn là O(1) nếu ta cần kết nối thêm:



Xác định ký tự theo chỉ số - hàm get

Để tìm ký tự theo chỉ số i cần duyệt cây bắt đầu từ nút gốc, dựa vào trọng số ở mỗi nút để quyết định đi sang cây con trái hay phải:

- ♣ Nếu nút đang xét không phải là lá:
 - ♠ Nếu nút trái có trọng số w và $w \geq i \rightarrow$ chuyển sang cây con trái,
 - ♠ Nếu $w < i \rightarrow i = w$, chuyển sang cây con phải,
- ♣ Nếu nút đang xét là nút lá: truy nhập tới két quả là ký tự thứ i .

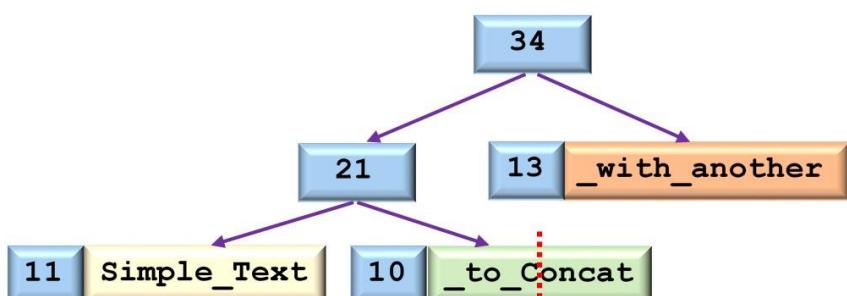
```
char get(int i, Node node):  
    if node.left != Ø  
        if node.left.w >= i  
            return get(i, node.left)  
        else  
            return get(i - node.left.w, node.right)  
    else  
        return node.s[i]
```

Độ phức tạp của giải thuật: $O(h)$, trong đó h – độ cao của cây.

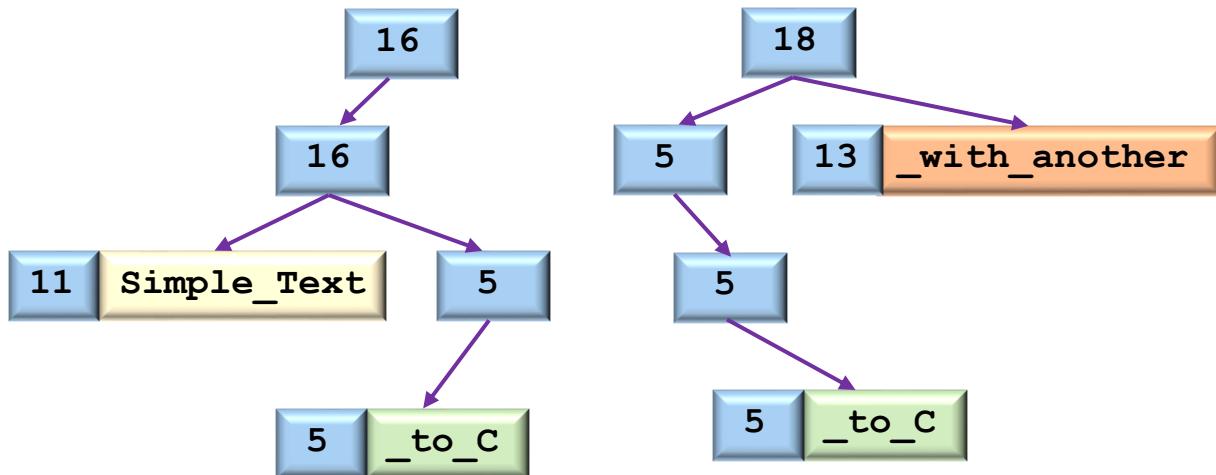
Tách xâu – hàm split

Để tách xâu thành 2 phần với ký tự cuối của phần đầu ở vị trí i : tìm phần xâu chứa ký tự thứ i (xử lý tương tự hàm **get**), mỗi nút gấp trên đường đi được chia thành hai để gắn với 2 cây, sau khi chia định cần tính lại trọng số.

Ví dụ dưới đây xét việc tách xâu ở vị trí 16.



Kết quả duyệt và tách đỉnh cho 2 cây:



Với những nút trong chỉ có một đỉnh con: cần thay đỉnh đó bằng đỉnh con của nó.



Giải thuật trên ngôn ngữ giả định:

```

Pair<Node, Node> split(Node node, int i):
    Node tree1, tree2
    if node.left != ∅
        if node.left.w >= i
            res = split(node.left, i)
            tree1 = res.first
            tree2.left = res.second
            tree2.right = node.right
            tree2.w = tree2.left.w + tree2.right.w
        else
            res = split(node.right, i - node.left.w)
            tree1.left = node.left
            tree1.right = res.first
            tree1.w = tree1.left.w + tree1.right.w
            tree2 = res.second
    else
        tree1.s = node.s.substr(0, i)
        tree2.s = node.s.substr(i, node.s.len)
        tree1.w = i
        tree2.w = node.s.len - i
    return (tree1, tree2)
  
```

Độ phức tạp của giải thuật: $O(h)$, trong đó h – độ cao của cây.

Các phép xóa, bổ sung – delete và insert

Các phép **delete** và **insert** dễ dàng thực hiện thông qua các hàm **merge** và **split** đã xét ở trên.

Giải thuật xóa các ký tự từ vị trí **beginIndex** cho tới trước vị trí **endIndex**:

```
Node delete(Node node, int beginIndex, int endIndex):  
    (tree1, tree2) = split(node, beginIndex)  
    tree3 = split(tree2, endIndex - beginIndex).second  
    return merge(tree1, tree3)
```

Giải thuật bổ sung xâu **s** vào xâu ban đầu từ vị trí **insertIndex**:

```
Node insert(Node node, int insertIndex, string s):  
    (tree1, tree3) = split(node, insertIndex)  
    tree2 = Node(s)  
    return merge(merge(tree1, tree2), tree3)
```

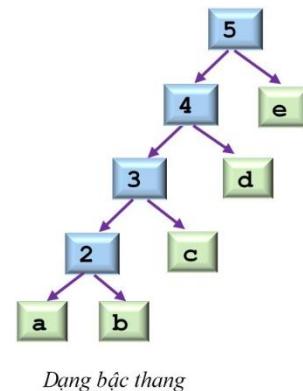
Độ phức tạp của giải thuật: $O(h)$.

Cân bằng cây

Mỗi lần ghép xâu đều cần kiểm tra xem cây có bị rơi vào dạng “bậc thang” hay không, trong trường hợp cần thiết phải tiến hành cân bằng cây.

Một cây được cân bằng tốt nếu độ dài của xâu không nhỏ hơn số Fibonacci thứ $(h+2)$.

Việc cân bằng cây được thực hiện theo các giải thuật áp dụng với cây nhị phân cân bằng.



Các giải pháp nâng cao hiệu quả

Việc tồn tại các xâu quá ngắn dễ làm cây mất cân bằng và làm giảm hiệu quả của cấu trúc, vì vậy người ta thường trực tiếp *ghép các xâu con ngắn* thành một xâu có độ dài không nhỏ hơn một giá trị m nào đó (ví dụ, $m = 32$). Giải pháp này giúp tiết kiệm đáng kể bộ nhớ và vẫn đảm bảo các ưu việt của cấu trúc rope.

Trong phần lớn các trường hợp xử lý cần duyệt các ký tự liên tiếp hoặc có vị trí gần nhau. Xác xuất của 2 ký tự ở các vị trí liên tiếp hoặc gần nhau thuộc cùng một xâu trong cấu trúc là rất cao. Vì vậy ở các bài toán thực tế, khi áp dụng cấu trúc này người ta thường tạo *phòng đệm vòng tròn* lưu vị trí kết quả tìm kiếm của các truy vấn. Quá trình tìm kiếm bắt đầu từ việc kiểm tra các kết quả đã nhận được ở các truy vấn trước.

BÀI TẬP

VT37. GIÁ ĐÈ DÀY

Tên chương trình: CUPBOARD.CPP

Khách đến tham quan một ngôi đền cổ phải để dày dép ở ngoài, trên các giá của một tủ tường lớn. Tủ tường có n giá, giá thứ nhất ở độ cao h_1 tính từ sàn, giá thứ 2 - ở độ cao h_2, \dots , giá thứ n - độ cao h_n .

Người trực ở lối vào thường phải chỉ giúp tìm lại dày dép của mình khi đi ra. Kinh nghiệm làm việc lâu năm giúp ông nhìn người là đoán ra họ để dày ở đâu. Người có chiều cao **height** không để dày dép của mình ở giá cao hơn **height** và thấp hơn **height/k**. Ngoài ra, người có chiều cao **height** đi dày dép có **size** không nhỏ hơn **height/m1** và không lớn hơn **height/m2**. Tuy vậy, thỉnh thoảng cũng gặp những trường hợp ngoại lệ - có những đôi dày không nằm đúng chỗ theo quy luật. Có thể ai đó đã để lại sang nơi khác để giải phóng chỗ cho người cùng đi theo mình.

Cần phải đặc biệt chú ý tới các đôi dày này để giúp khách ra khỏi lúng túng khi tìm dày.

Hãy xác định số đôi dày không nằm đúng vị trí theo quy luật.

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

- ✚ Dòng đầu tiên chứa một số nguyên n ($1 \leq n \leq 10^5$),
- ✚ Dòng thứ 2 chứa 3 số nguyên $K, m1$ và $m2$ ($1 \leq k \leq 100, 1 \leq m2 \leq m1 \leq 100$),
- ✚ Dòng thứ i trong n dòng sau chứa số nguyên h_i ($1 \leq h_1 \leq 10^7, h_{i-1} < h_i \leq 10^7$ với mọi i), sau đó là số nguyên k_i - số đôi dày/dép ở ngăn thứ i ($1 \leq k_i \leq 10^5$), tiếp theo là k_i số nguyên $size_{i,j}$ - kích thước các đôi dày/dép trên giá ($1 \leq size_{i,j} \leq 10^7$).

Tổng tất cả các ki không vượt quá 10^5 .

Kết quả: Đưa ra file văn bản CUPBOARD.OUT một số nguyên – số trường hợp ngoại lệ.

Ví dụ:

CUPBOARD.INP
3
2 1 1
1 2 1 2
2 2 1 4
4 2 3 4

CUPBOARD.OUT
2



VT37 IO20161015 B

Giải thuật: Kỹ thuật tổ chức dữ liệu.

Nhận xét:

Từ điều kiện $\frac{height}{K} \leq h_i \leq height \Leftrightarrow h_i \leq height \leq h_i \cdot K$

Từ $\frac{height}{m_1} \leq size \leq \frac{height}{m_2}$

Ta có:

$$\frac{h_i}{m_1} \leq size \Leftrightarrow h_i \leq size \cdot m_1$$

$$size \leq \frac{h_i \cdot k}{m_2} \Leftrightarrow size \cdot m_2 \leq h_i \cdot K$$

Kiểm tra các điều kiện trên với từng vị trí đặt dày/dép ta sẽ tính được kết quả cần tìm.

Tổ chức dữ liệu: Tổ chức các mảng động:

- `vector<int> k(n)` – lưu số lượng dày/dép ở mỗi giá,
- `vector<int> h(n)` – lưu độ cao của giá đế,
- `vector<vector<int>> shoes(n)` – lưu kích thước dày/dép trên giá (mảng 2 chiều).

Lưu ý: Cần ép kiểu dữ liệu để nhận được kết quả kiểm tra chính xác.

Độ phức tạp của giải thuật: O(n).

Chương trình:

```
#include <bits/stdc++.h>
#define NAME "cupboard."
using namespace std;
ifstream fi (NAME"inp");
ofstream fo (NAME"out");
int n,m1,m2,K;

int main()
{
    fi>>n>>K>>m1>>m2;
    vector<int> k(n);
    vector<int> h(n);
    vector<vector<int>> shoes(n);
    for (int i = 0; i < n; i++)
    {
        fi>>h[i]>>k[i];
        shoes[i].resize(k[i]);
        for (int j = 0; j < k[i]; j++) fi>>shoes[i][j];
    }
    int ans = 0;
    for (int i = 0; i < n; i++) {
        for (size_t j = 0; j < shoes[i].size(); j++)
        {
            int size = shoes[i][j];
            if (size * 1LL * m2 <= h[i] * 1LL * K && size * 1LL * m1 >=
h[i]) continue;
            ans++;
        }
    }
    fo<<ans;
    fo<<"\nTime: "<<clock() / (double)1000<<" sec";
    return 0;
}
```



VT38. ĐƯỜNG BẬC THANG

Tên chương trình: STAIRWAY.CPP

Các bậc thang của đường dẫn lên đỉnh núi được đánh số từ 1 đến n , 1 là bậc thang đầu tiên, n – bậc cuối cùng trên đỉnh núi. Ở các bậc có số chẵn 10 người ta ghi số của bậc trên bậc đó. Ngoài ra bậc đầu tiên và bậc cuối cùng cũng được ghi số. Chi phí để sơn mỗi chữ số là như nhau và được coi là bằng 1.

Hãy xác định tổng chi phí của việc đánh dấu trên toàn bộ đường đi.

Dữ liệu: Vào từ file văn bản STAIRWAY.INP gồm một dòng chứa số nguyên n ($1 \leq n \leq 10^{12}$).

Kết quả: Đưa ra file văn bản STAIRWAY.OUT một số nguyên – tổng chi phí tìm được.

Ví dụ:

STAIRWAY.INP
23

STAIRWAY.OUT
7



Giải thuật: Tổ chức dữ liệu.

Nhận xét:

- ✚ Không kể bậc đầu tiên và bậc cuối cùng, số lượng bậc thang được đánh dấu sẽ là $m = n / 10$,
- ✚ Nếu bỏ chữ số 0 cuối cùng ở các số viết ở các bậc thang nói trên thì các số đã viết làm thành dãy số từ 1 đến m ,
- ✚ Số lượng chữ số 0 bị bỏ qua là m .
- ✚ Như vậy cần phải tính số lượng các chữ số trong dãy các số tự nhiên liên tiếp từ 1 đến m , cộng thêm 1 chữ số ở bậc đầu tiên, m chữ số 0 ở cuối mỗi số và số lượng chữ số ở bậc cuối cùng nếu n không chia hết cho 10.

Tính số lượng các chữ số trong dãy số liên tiếp từ 1 đến m :

Gọi d_i – số lượng chữ số của các số dương có không quá i chữ số, $p_i = 10^i$, $i = 0, 1, 2, \dots$. Ta có:

i	d_i
0	0
1	9
2	$9+90\times2 = 189$
3	$189+900\times3 = 2889$
...
i	$d_{i-1} + 9 \times p_{i-1} \times i$
...

Tổ chức dữ liệu: các mảng `int64_t d[13], p[13]`.

Xử lý:

- Chuẩn bị: các mảng p và d,
- Tính các chữ số của m,
- Tính số chữ số ghi trên các bậc (trừ bậc cuối),
- Kiểm tra và chỉnh lý kết quả nếu bậc cuối chưa được ghi theo quy luật chung.

Độ phức tạp của giải thuật: O(1).

Chương trình:

```
#include <bits/stdc++.h>
#define NAME "stairway."
using namespace std;
ifstream fi (NAME"inp");
ofstream fo (NAME"out");
int64_t n,ans,m,p[13],d[13],k=0,v;

int main()
{
    fi>>n; m=n/10;
    p[0]=1; d[0]=0;
    for(int i=1;i<=12;++i) {p[i]=p[i-1]*10;d[i]=d[i-1]+p[i-1]*9*i;}
    v=m; while(v>0) ++k, v/=10;
    v=m-p[k-1]+1;
    ans=d[k-1]+v*k+m+1;
    if(n%10) ans+=k+1;
    fo<<ans;
    fo<<"\nTime: "<<clock() / (double)1000<<" sec";
}
```



VT39. IPHONE

Tên chương trình: IPHONE.CPP

Trong khi chờ đợi vào dự buổi giới thiệu một sản phẩm mới các khách mời được bố trí ngồi uống nước và ăn hoa quả quanh bàn tròn. Jeniffier phụ trách tiếp khách và cũng ngồi một chỗ bên bàn. Có tất cả n người kể cả Jeniffier. Sự chú ý của mọi người đều tập trung vào Jeniffier vì muôn qua cô biết thêm về đặc trưng của sản phẩm sắp được giới thiệu. Về phần mình, Jeniffier tỏ ra là một chủ nhà hiếu khách. Cô liên tục nói chuyện với khách mời về những vấn đề mà họ quan tâm. Để giữ không khí tĩnh lặng trang nghiêm trước một sự kiện lớn và phép lịch sự không nói chen qua mặt người khác cô xin đổi chỗ với người ngồi bên cạnh. Dĩ nhiên khách mời rất vui lòng đáp ứng yêu cầu của cô.

Khi phòng giới thiệu mở cửa, mọi người cùng Jeniffier đứng dậy đi vào. Lát sau cô mới phát hiện ra là mình đã để quên Iphone ở vị trí ngồi cuối cùng trước khi rời đi. Jeniffier không nhớ rõ đó là nơi nào nhưng nhớ vị trí ngồi ban đầu của mình và nhớ là mình đã k lần đổi chỗ với người ngồi bên cạnh. Cô nhờ một nhân viên bảo vệ ra tìm hộ Iphone của mình, chỉ cho anh ta những nơi có thể có.

Hãy xác định số lượng các vị trí mà nhân viên bảo vệ phải tới kiểm tra theo yêu cầu của Jeniffier.

Dữ liệu: Vào từ file văn bản IPHONE.INP gồm một dòng chứa 2 số nguyên n và k ($3 \leq n \leq 10^9$, $0 \leq k \leq 10^9$).

Kết quả: Đưa ra file văn bản IPHONE.OUT một số nguyên – số lượng vị trí cần tới tìm.

Ví dụ:

IPHONE.INP	IPHONE.OUT
5 2	3

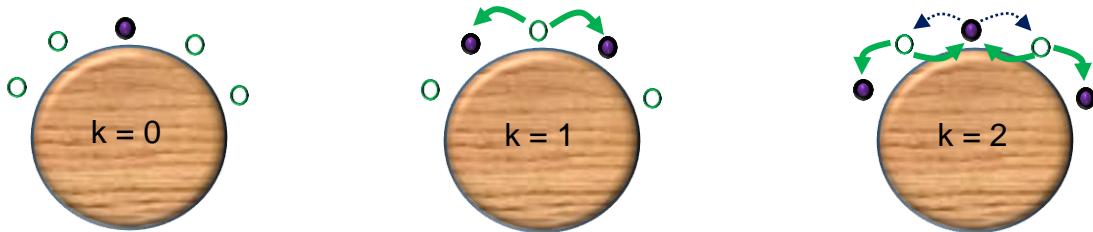


VT39 IO20161023 A

Giải thuật: Khả năng phân tích và đoán nhận giải thuật.

Nhận xét:

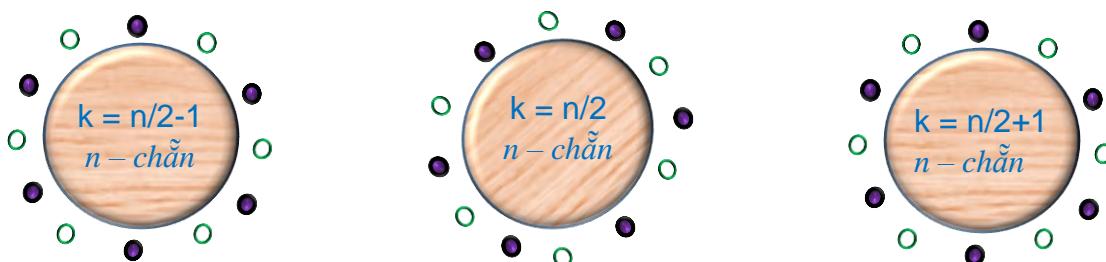
Trước hết xét một số trường hợp riêng, khi k đủ nhỏ:



Các vị trí cuối cùng có thể được đánh dấu đậm.

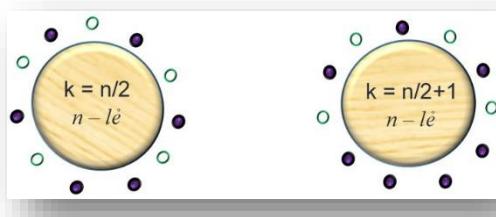
Dễ dàng thấy rằng số vị trí cần tìm là $k+1$.

Khi $k > n/2$ vị trí cuối luôn luôn lặp lại một trong các vị trí cuối có thể khác nếu n là chẵn và tồn tại $n/2$ vị trí không thể là cuối:



Với n lẻ, số vị trí cuối mới sẽ tăng dần khi k tăng, mọi vị trí đều có thể là vị trí cuối khi $k \geq n-1$:

Kết quả sẽ là $\min\{n/2, k+1\}$ nếu n chẵn và $\min\{n, k+1\}$ với n lẻ.



Chương trình:

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

int main()
{
    fi>>n>>k; ++k;
    if(n&1) ans=min(n,k); else ans=min(n/2,k);
    fo<<ans;
    fo<<"\nTime: "<<clock()/(double)1000<<" sec";
}
```

Độ phức tạp của giải thuật:
 $O(1)$.



VT40. ĐIỂM THƯỞNG

Tên chương trình: BONUS.CPP

Cách thưởng điểm của một trò chơi điện tử mới là như sau: trên màn hình có n quả bóng bay đánh số từ 1 đến n . Khi bắt đầu vào chơi ở mức 1 số điểm thưởng b của người chơi là 0. Nếu người chơi vượt qua được mức thứ i thì quả bóng có số i sẽ bay lên, lần lượt chạm với các quả bóng còn lại, mỗi khi quả bóng i chạm với quả bóng j , giá trị $i \text{ xor } j$ sẽ được cộng vào b . Sau khi chạm vào quả bóng n quả bóng i sẽ bị nổ và làm cho giá trị b trở thành $b \bmod 10^9 + 7$.

Toàn bộ trò chơi có $n-1$ mức. Hãy xác định số điểm thưởng b của người chơi sau khi vượt qua mức $n-1$.

Dữ liệu: Vào từ file văn bản BONUS.INP gồm một dòng chứa số nguyên n ($2 \leq n \leq 10^9$).

Kết quả: Đưa ra file văn bản BONUS.OUT một số nguyên – số điểm thưởng b ở cuối trò chơi.

Ví dụ:

BONUS.INP
3

BONUS.OUT
6



VT40 IO20161023 K

Giải thuật: Xử lý bít.

Nhận xét:

Gọi c_i là số lượng số trong phạm vi từ 1 đến n có bít thứ i bằng 1 ($i = 0, 1, 2, \dots$). Như vậy số lượng số có bít thứ i bằng 0 sẽ là $n - c_i$.

Nếu k là số lượng bít có nghĩa của n thì kết quả cần tính (chưa lấy modun 10^9+7) sẽ là:

$$\sum_{i=0}^k 2^i c_i (n - c_i)$$

Cách tính nhanh c_i : Xét các số 0, 1, 2, ..., n theo thứ tự tăng dần. 2^i số đầu tiên có bít ở vị trí i là 0, 2^i số tiếp theo bít ở vị trí i là 1, 2^i số tiếp theo nữa bít ở vị trí i là 0 và cứ như thế lặp đi lặp lại với chu kỳ 2^{i+1} . Như vậy dễ dàng suy ra c_i từ số chu kỳ cộng thêm số lượng ở phần còn lại cuối cùng (nếu có).

Tổ chức dữ liệu: mảng **int c[31]** lưu các c_i đã nói ở trên.

Xử lý:

Tính c_i :

Tính theo số chu kỳ
đầy đủ

```

for(int i=0;i<31;++i)
{
    c[i] += (n >> (i + 1)) << i;
    if(n % (1 << (i + 1)) >= (1 << i))
        c[i] += n % (1 << (i + 1)) - (1 << i) + 1;
    c[i] %= p;
}

```

Phần còn lại

Tính kết quả: theo công thức đã nêu.

Độ phức tạp của giải thuật: O(1).

Chương trình:

```
#include <bits/stdc++.h>
#define NAME "bonus."
#define mk make_pair
using namespace std;
ifstream fi (NAME"inp");
ofstream fo (NAME"out");
const int IX=31;
const int p=1000000007;
int64_t a[IX],ans=0,x,y,z;
int n;

int main()
{
    fi >> n;
    for(int i=0;i<IX;++i)
    {
        a[i] += (n >> (i + 1)) << i;
        if(n % (1 << (i + 1)) >= (1 << i)){
            a[i] += n % (1 << (i + 1)) - (1 << i) + 1;
        }
        a[i] %= p;
    }

    for(int i=0;i<IX;++i)
    {
        x = (1ll << i) % p;
        y = a[i] % p;
        z = (n - a[i]) % p;
        ans += x * y % p * z % p;
        ans %= p;
    }
    fo << ans % p;
}

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



VT41. NGÃ TƯ

Tên chương trình: CROSSY.CPP

Đèn điều khiển giao thông ở ngã tư giao giữa 2 đường phố chính Bắc – Nam và Đông – Tây bị hỏng. Ở ngã tư này các xe không được rẽ, tức là chỉ được chạy thẳng theo các hướng **U**, **D**, **L**, **R** (Xem hình bên).

Trong thời gian sửa chữa hệ thống tín hiệu đèn đường người ta quan sát thấy có **n** xe đến ngã tư, xe thứ **i** đến vào thời điểm **t_i** và đi theo hướng **d_i** ($d_i \in \{L, R, U, D\}$). Các ô tô đến ngã tư và đứng vào dòng xếp hàng chờ qua đường. Các lái xe đều tuân thủ luật lệ giao thông “Nhường đường cho xe bên phải”. Như vậy có 4 dòng xếp hàng tương ứng với các hướng đi **U**, **D**, **L**, **R**. Tại thời điểm đến và cứ mỗi giây sau đó xe đứng đầu dòng xếp hàng sẽ vượt qua ngã tư nếu dòng xếp hàng bên phải mình không có xe. Các xe bên dưới sẽ dồn lên trong dòng xếp hàng của mình. Cứ mỗi giây theo mỗi hướng có không quá một ô tô tới ngã tư.

Với mỗi xe đến hãy xác định thời điểm xe đó qua ngã tư. Có thể xe phải xếp hàng mãi, khi đó thời điểm qua ngã tư được coi là bằng -1.

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

- ✚ Dòng đầu tiên chứa một số nguyên **n** ($1 \leq n \leq 10^5$),
- ✚ Dòng thứ **i** trong **n** dòng sau chứa 2 đại lượng **t_i** và **d_i** ($0 \leq t_i \leq 10^9$).

Kết quả: Đưa ra file văn bản CROSSY.OUT n số nguyên – thời điểm qua ngã tư của mỗi ô tô (liệt kê theo trình tự thời điểm tới ngã tư nêu trong input, mỗi số trên một dòng).

Ví dụ:

CROSSY.INP
4
0 R
0 U
0 L
5 D

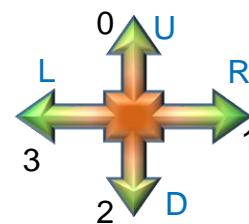
CROSSY.OUT
2
1
0
5



Giải thuật: Tổ chức dữ liệu, xử lý đồng bộ thời gian.

Nhận xét:

- ✚ Cân tổ chức 4 hàng đợi cho các hướng,
- ✚ Thông tin cần lưu trữ: Thời điểm đến ngã tư và *số thứ tự trong input*,
- ✚ Để tiện xử lý: Đánh số các hướng từ 0 đến 3,
- ✚ Những xe tới ngã tư cùng một lúc được nạp vào các hàng đợi trước khi xét khả năng vượt ngã tư,
- ✚ Thời điểm xét: tăng dần sang giây tiếp theo hoặc thời điểm gần nhất có xe tới.



Tổ chức dữ liệu:

- ▣ Mảng **car** `t[100001]` – lưu dữ liệu vào kèm tên số thứ tự,
- ▣ Bốn hàng đợi **queue<int>** `q[4]` – lưu các xe đang chờ ở mỗi hướng,
- ▣ Mảng động **vector<int>** `v` – lưu các xe qua được ngã tư trong khoảng thời gian xét,
- ▣ Mảng **int** `out[100001]` – lưu kết quả.

Xử lý:

Sử dụng cờ **bool b = false** để ngắt quá trình xét khi tất cả các xe đều phải chờ, ít nhất 3 hàng đợi khác rỗng,

Ghi nhận dữ liệu vào:

```

for (int i = 0; i < n; i++)
{
    char c;
    fi>>t[i].t>>c;
    t[i].dir = dir[c];
    t[i].num = i;
}

```

Đưa dữ liệu vào hàng đợi xét vượt ngã tư:

```

if (b)break;
while (t[j].t == t[i].t)
{
    q[t[j].dir].push(t[j].num);
    j++;
}

```

Với hàng đợi **z**, hàng đợi cần trả nó là **(z+3) & 3**,

Ghi nhận xe qua được ngã tư:

```
curt++;
vector<int> v;
b = true;
for (int z = 0; z < 4; z++)
    if (q[(z + 3) & 3].empty() && !q[z].empty())
    {
        b = false;
        v.push_back(z);
    }
if (b) break;
for (int z : v)
{
    out[q[z].front()] = curt;
    q[z].pop();
}
```



Đưa kết quả cuối cùng:

```
for (int i = 0; i < n; i++)
    fo << out[i] - 1 << "\n";
```

Độ phức tạp của giải thuật: $O(n)$.

Chương trình:

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

ofstream fo (NAME"out");
struct car{
    int t, dir, num;
    car() {}
};

/*
bool operator < (car a, car b){
    return a.t < b.t;
}
*/
car t[100001];
int dir[256];
int out[100001];

int main()
{
    int n;
    fi>>n;
    dir['U'] = 0;
    dir['R'] = 1;
    dir['D'] = 2;
    dir['L'] = 3;
    for (int i = 0; i < n; i++)
    {
        char c;
        fi>>t[i].t>>c;
        t[i].dir = dir[c];
        t[i].num = i;
    }
    queue<int> q[4];
    int j = 0;
    bool b = false;
    for (int i = 0; i < n; i++)
    {
        if (b)break;

        while (t[j].t == t[i].t)
        {
            q[t[j].dir].push(t[j].num);
            j++;
        }
        int curt = t[i].t;
        while ((j == n || curt < t[j].t) && !(q[0].empty() && q[1].empty() &&
q[2].empty() && q[3].empty()))
        {
            curt++;
            vector<int> v;
            b = true;
            for (int z = 0; z < 4; z++)
                if (q[(z + 3) & 3].empty() && !q[z].empty())
                {
                    b = false;
                    v.push_back(z);
                }
            if (b) fo<<v;
        }
    }
}
```

```

        }
        if (b) break;
        for (int z : v)
        {
            out[q[z].front()] = curt;
            q[z].pop();
        }
        i = j - 1;
    }

    for (int i = 0; i < n; i++)
    {
        fo << out[i] - 1 << "\n";
    }
    fo << "\nTime: " << clock() / (double)1000 << " sec";
    return 0;
}

```



VT42. RUỘNG BẬC THANG

Tên chương trình: TERRACES.CPP

Ruộng bậc thang ở Tây Bắc Việt Nam, là địa điểm có phong cảnh tuyệt đẹp, thu hút nhiều khách du lịch. Các bậc thang được đánh số từ 1 đến n bắt đầu từ dưới chân núi. Ở các bậc có số chia hết cho 8 (tức là các bậc 8, 16, 24, ...) và ở bậc cuối cùng (bậc n) người ta dùng đá khảm vào thành ruộng số của bậc đó để khách du lịch biết mình đã leo cao bao nhiêu. Theo quan niệm của nhiều người, 6 và 8 là các chữ số may mắn vì vậy vì vậy người ta dùng đá quý để khảm các chữ số này.



Có m tuyến du lịch, tuyến thứ i tới vùng ở đó khu ruộng bậc thang có n_i bậc và có giá bằng số lượng các chữ số được khảm bằng đá quý khách du lịch nhìn thấy khi leo hết tất cả các bậc ruộng.

Cho m và các số n_1, n_2, \dots, n_m . Hãy xác định giá của mỗi tuyến du lịch.

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

- ✚ Dòng đầu tiên chứa số nguyên m ($1 \leq m \leq 1000$),
- ✚ Dòng thứ i trong m dòng tiếp theo chứa số nguyên n_i ($8 \leq n_i \leq 10^{15}$).

Kết quả: Đưa ra file văn bản TERRACES.OUT m số nguyên – giá của mỗi tuyến du lịch, mỗi số đưa ra trên một dòng.

Ví dụ:

TERRACES.INP
4
9
32
56
18

TERRACES.OUT
1
2
4
3



Giải thuật: Phân tích tình huống lô gic, bảng phưong án.

Nhận xét:

Ba chữ số cuối của bậc thang được đánh dấu lặp lại theo chu kỳ 1000, tức là bậc **xxx** được đánh dấu thì các bậc **1xxx, 11xxx, 111xxx, 5634xxx, ...** cũng sẽ được đánh dấu,

Số lượng các số 6 và 8 (**các chữ số đặc biệt**) ở các bậc được đánh dấu trong các khoảng [0,199], [100,299], [200,399], [300,499], [400,599] giống nhau và bằng 16 ở mỗi khoảng,

Số lượng các chữ số đặc biệt ở các bậc được đánh dấu trong các khoảng [600,799], [800,999] giống nhau và bằng $16 + 13$ ở mỗi khoảng,

Gọi d_i – số các chữ số đặc biệt ở các bậc được đánh dấu trong phạm vi tất cả các bậc thang có số thứ tự không quá $i+1$ chữ số, ta có:

$$d_2 = 106,$$

$$d_i = d_{i-1} \times 10 + 2 \times 125 \times 10^{i-3}, i \geq 3.$$

Như vậy với số **n** có **k+1** chữ số ta chỉ còn tìm thêm số các chữ số đặc biệt được đánh dấu trong khoảng $[10^{k+1}, n]$,

$$\text{Nếu } n = x_k 10^k + x_{k-1} 10^{k-1} + \dots + x_1 10^1 + x_0$$

$$\begin{array}{ccccccccc}
 & x_k 10^k & + & x_{k-1} 10^{k-1} & + & \dots & + & x_1 10^1 & + & x_0 \\
 & \downarrow & + & \downarrow & + & & & & & \\
 & (x_k - 1) \times d_{k-1} & & & & \dots & & & \\
 & & \downarrow & & & & & & \\
 & & (x_{k-1} - 1) \times d_{k-2} & & & & & &
 \end{array}$$

Nếu $x_i \geq 6$: cần bổ sung thêm số lượng các chữ số đặc biệt chưa tính theo công thức trên, cần phân biệt các trường hợp $x_i = 6, x_i = 7, x_i = 8$ và $x_i = 9$.

Lưu ý là 999 bậc thang đầu tiên có quy luật riêng vì vậy với các bậc thang này nên tạo bảng giá trị riêng và khi $n < 10^3$ – chỉ cần tra bảng và cách tính nếu trên chỉ áp dụng chừng nào **k** còn lớn hơn 3.

Tổ chức dữ liệu:

Mảng **int64_t b[1000]** – **b_i** lưu kết quả cần tìm khi $i < 10^3$,

Mảng **int64_t d[16]** – lưu các giá trị **d_i** đã nói ở trên,

Mảng **int64_t p[16]** – lưu các giá trị **p_i = 10ⁱ**.

Xử lý:

Tính bảng **b**: Chỉ thực hiện một lần vì vậy không cần tối ưu hóa giải thuật

```
void calc_b()
{int64_t m, tp, r=0;
 m=1000; b[0]=0;
 for(int i=1; i<m; ++i)
 {
     b[i]=b[i-1]; tp=i; if(tp%8) continue;
     while(tp>0)
     {
         if(tp%10 ==6 || tp%10==8) ++b[i];
         tp/=10;
     }
 }
```

Chỉnh lý kết quả khi khoảng xét bắt đầu chữ số 6 hoặc lớn hơn:

*Khoảng chỉ chia một bộ phận
số bắt đầu bằng 6 hoặc 8*

```
void correct_ans()
{int64_t tg;
 if(q==6) {tg=m; ans+=tg/8+1; return;}
 if(q==7) {tg=p[k]; ans+=tg/8; return;}
 if(q==8) {tg=p[k]+m; ans+=tg/8+1; return;}
 if(q==9) {tg=p[k]*2; ans+=tg/8; return;}
 }
```

Độ phức tạp của giải thuật: Với mỗi số n giải thuật có độ phức tạp $O(\lg n)$.

Chương trình:

```
#include <bits/stdc++.h>
#define NAME "terraces."
using namespace std;
ifstream fi (NAME"inp");
ofstream fo (NAME"out");
int64_t ts,n,d[16],p[16],q,t,k=0,m,ans;
int64_t b[1000];

void calc_b()
{
    int64_t m, tp, r=0;
    m=1000; b[0]=0;
    for(int i=1; i<m; ++i)
    {
        b[i]=b[i-1]; tp=i; if(tp%8) continue;
        while(tp>0)
        {
            if(tp%10 ==6 || tp%10==8) ++b[i];
            tp/=10;
        }
    }
}

void correct_ans()
{
    int64_t tg;
    if(q==6) {tg=m; ans+=tg/8+1; return;}
    if(q==7) {tg=p[k]; ans+=tg/8; return;}
    if(q==8) {tg=p[k]+m; ans+=tg/8+1; return;}
    if(q==9) {tg=p[k]*2; ans+=tg/8; return;}
}

void solve()
{
    fi>>n;
    m=n; ans=0;
    while(m>=1000)
    {
        int64_t tm;
        tm=m; k=0; while(tm>0) ++k, tm/=10; --k;
        ans+=d[k-1]; tm=m; q=tm/p[k];
        for(int i=1; i<q; ++i) ans+=d[k-1];
        m%=p[k]; if(q>=6) correct_ans();
        if(m<1000) break;
    }
    ans+=b[m];
    m=n;
    if(n%8!=0)
        while(m>0)
    {
        t=m%10; if(t==6 || t==8) ++ans;
        m/=10;
    }

    fo<<ans<<endl;
}

int main()
{
    fi>>ts;
    d[2]=106; t=125; p[0]=1; d[0]=0;
```

```
for(int i=1;i<16;++i)p[i]=p[i-1]*10;
for(int i=3;i<16;++i){d[i]=d[i-1]*10+2*t; t*=10;}
calc_b();
for(int i=0;i<ts;++i) solve();
fo<<"\nTime: "<<clock()/(double)1000<<" sec";
}
```



VT43. GIẢM GIÁ

Tên chương trình: SALE.CPP

Cuối năm, tất cả các cửa hàng đều giảm giá một số mặt hàng để thanh lý hàng tồn kho và thu hút khách hàng. Không muốn là trường hợp ngoại lệ, hiệu sách trung tâm thành phố cũng có chính sách giảm giá của mình. Nếu khách hàng đặt mua sách qua mạng và với mỗi đơn đặt hàng mua từ $w+q$ cuốn trở lên, w cuốn giá thấp nhất trong đơn sẽ được miễn phí. Giá vận chuyển giao hàng vẫn là e đồng với mỗi đơn đặt hàng không phụ thuộc vào số lượng sách đặt mua. Giá trị q sẽ được công bố trước thời điểm giảm giá và chỉ có hiệu lực 1 giờ kể từ thời điểm công bố. Một người có thể gửi nhiều đơn đặt hàng và không hạn chế số lượng sách đặt trong mỗi đơn.

Steve dự định mua n cuốn sách, cuốn thứ i có giá là a_i . Tổng số tiền phải trả sẽ phụ thuộc vào giá trị q cụ thể và Steve nhầm tính số tiền phải chi trả với các q bằng $1, 2, \dots, n$.

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

✚ Dòng đầu tiên chứa 3 số nguyên n , e và w ($1 \leq w \leq n \leq 3 \times 10^5$, $1 \leq e \leq 10^6$),

✚ Dòng thứ i trong n dòng sau chứa số nguyên a_i ($1 \leq a_i \leq 10^6$).

Kết quả: Đưa ra file văn bản SALE.OUT trên một dòng n số nguyên các tổng số tiền phải trả ứng với q bằng $1, 2, \dots, n$.

Ví dụ:

SALE.INP
5 1 1
1
2
3
4
5

SALE.OUT
12 14 15 15 16



Giải thuật: Tổng tiền tố.

Nhận xét:

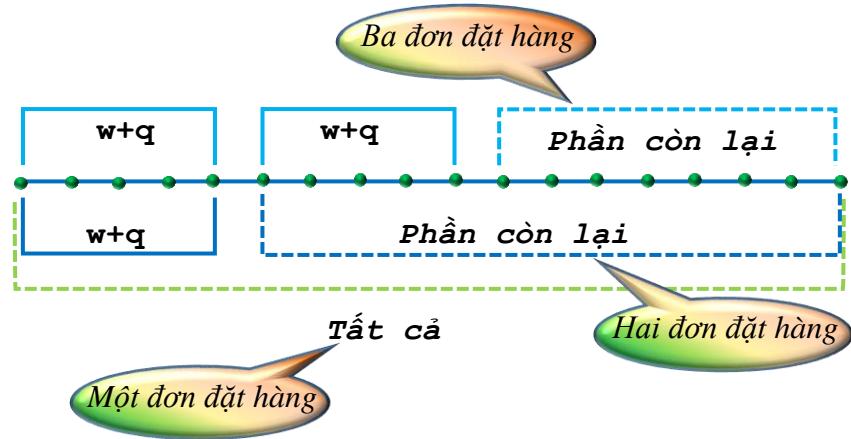
Với q xác định, mỗi đơn đặt hàng chỉ nên đặt $w+q$ cuốn sách (trừ đơn đặt hàng cuối cùng – mua các quyền còn lại),

Mỗi đơn đặt hàng: đặt mua những cuốn có giá cao nhất trong số các cuốn còn chưa mua, như vậy sẽ bớt được nhiều nhất số tiền phải trả,

Các sách cần được sắp xếp giảm dần theo giá để lựa chọn,

Phải tính tổng số tiền ứng với một đoạn liên tục các cuốn sách, vì vậy cần chuẩn bị tổng tiền tố,

Với mỗi q xác định lần lượt xét các trường hợp mua sách bằng 1, 2, 3, . . . đơn đặt hàng và chọn phương án với chi phí nhỏ nhất. Số phương án xét sẽ không vượt quá $n / (w+q) + 1$.

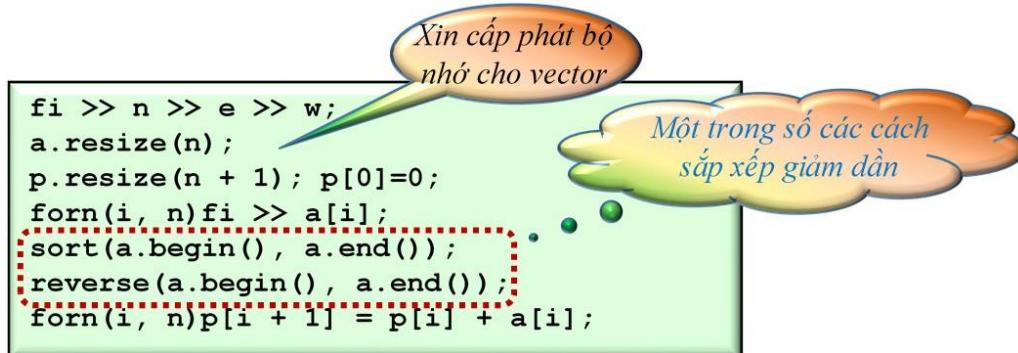


Tổ chức dữ liệu:

Các mảng a và p lưu dữ liệu vào và tổng tiền tố.

Xử lý:

Nhập dữ liệu, sắp xếp và tính tổng tiền tố:



Tính chi phí đơn đặt hàng mua các cuốn sách từ **l** đến **r** với một **q** xác định:

```
int64_t buy(int l, int r, int q)
{
    int s = (r - l + 1);
    if(s <= 0) return 0;
    if(s >= q + w) r -= w;
    return p[r + 1] - p[l] + e;
}
```

Duyệt các cách mua với các **q**:

```
for(int q = 1; q <= n; q++)
{
    int64_t s = 0, ans = buy(0, n - 1, q);
    int l = 0;
    while(l < n)
    {
        int r = min(l + q + w, n);
        s += buy(l, r - 1, q);
        ans = min(ans, s + buy(r, n - 1, q));
        l = r;
    }
    fo << ans << ' ';
}
```

Độ phức tạp của giải thuật: $O(nlnn)$.

Chương trình:

```
#include <bits/stdc++.h>
#define NAME "sale."
#define forn(i, n) for(int i = 0; i < n; i++)
using namespace std;
ifstream fi (NAME"inp");
ofstream fo (NAME"out");

int n, w, e;
vector<int64_t> a, p;

int64_t buy(int l, int r, int q)
{
    int s = (r - l + 1);
    if(s <= 0) return 0;
    if(s >= q + w) r -= w;
    return p[r + 1] - p[l] + e;
}

int main()
{
    fi >> n >> e >> w;
    a.resize(n);
    p.resize(n + 1); p[0]=0;
    forn(i, n) fi >> a[i];
    sort(a.begin(), a.end());
    reverse(a.begin(), a.end());
    forn(i, n) p[i + 1] = p[i] + a[i];
    for(int q = 1; q <= n; q++)
    {
        int64_t s = 0, ans = buy(0, n - 1, q);
        int l = 0;
        while(l < n) {
            int r = min(l + q + w, n);
            s += buy(l, r - 1, q);
            ans = min(ans, s + buy(r, n - 1, q));
            l = r;
        }
        fo << ans << ' ';
    }
    fo<<"\nTime: "<<clock() / (double)1000<<" sec";
}
```



VT44. ĐỊA LAN

Tên chương trình: CYMBIDIUM.CPP

Giáo sư Braun đã lai giống tạo ra n loại địa lan mới độc đáo. Mỗi loại địa lan được trồng trong một chậu riêng và được ghi một số thể hiện đặc điểm gene của loại hoa đó, chậu thứ i có số là a_i , $i = 1 \div n$.

Hội chợ hoa địa lan thế giới mời giáo sư tham gia và giới thiệu các thành quả nghiên cứu của ông. Các chậu hoa do ông mang tới sẽ được đặt trên k đôn dọc theo đường đi dẫn tới gian hàng chính của hội chợ.

Muốn gây ấn tượng mạnh cho những người tham dự, đặc biệt là các nhà khoa học Giáo sư yêu cầu trợ lý của mình chọn k chậu sao cho khi thực hiện phép **AND** số trên các chậu hoa tới (nếu xét chúng ở dạng biểu diễn nhị phân) sẽ có kết quả bằng 0.

Hãy xác định xem người trợ lý có thể chọn được k chậu như vậy hay không và đưa ra thông báo “**YES**” hoặc “**NO**”.

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

- ✚ Dòng đầu tiên chứa 2 số nguyên n và k ($1 \leq n \leq 2 \times 10^4$, $1 \leq k \leq n$),
- ✚ Dòng thứ 2 chứa n số nguyên a_1, a_2, \dots, a_n ($0 \leq a_i < 2^{12}$, $i = 1 \div n$).

Kết quả: Đưa ra file văn bản CYMBIDIUM.OUT thông báo tìm được.

Ví dụ:

CYMBIDIUM.INP
3 2
5 4 3

CYMBIDIUM.OUT
YES



VT44 IO20161105 B

Giải thuật: Quy hoạch động, cập nhật tại chỗ, bảng phương án.

Nhận xét:

Mặt nạ m là chuỗi bít để tách từ một chuỗi bít bất kỳ các bít tương ứng với 1 trong m bằng phép **AND**,

Để xác định có tồn tại **k** số trong dãy số đã cho có kết quả các phép **AND** bằng 0 ta chỉ cần xác định từ các số đã cho có tạo được mặt nạ 0 với sự tham gia của **k** hoặc ít hơn các số đã cho hay không,

Nếu tạo được mặt nạ 0 với đúng **k** số hạng thì đó chính là các số phải chọn,

Nếu mặt nạ 0 được tạo ra bởi ít số lượng số hạng ít hơn **k** thì các số còn lại có thể chọn bất kỳ (số nào nhân với 0 cũng cho kết quả 0),

Ở đây ta không phải chỉ ra các số cần chọn vì vậy vấn đề trở nên đơn giản hơn.

Các số đã cho nhỏ hơn 2^{12} , vì vậy ta chỉ cần tìm *mặt nạ 12 bít*,

Việc tìm mặt nạ được thực hiện tương tự *Bài toán đổi tiền*,

Gọi dp_j là số lượng số ít nhất trong số các số a_0, a_1, \dots, a_{i-1} cần chọn để tạo ra mặt nạ j ($j = 0, 1, 2, \dots, 2^{12}-1$) .

Giá trị đầu của dp_j :

$$dp_j = \begin{cases} 1 & \text{với } j = a_0, \\ 0 & \text{với } j = 2^{12}-1, \\ +\infty & \text{trong các trường hợp còn lại.} \end{cases}$$

Với mỗi $i = 1 \div n-1$ duyệt mọi $j = 0 \div 2^{12}-1$ tính:

$$dp_{a_i \& j} = \min\{dp_j + 1, dp_{a_i \& j}\}$$

Kết quả: nếu $dp_0 \leq k \rightarrow$ có cách chọn, trong trường hợp ngược lại – không có cách chọn thỏa mãn điều kiện đã nêu.

Tổ chức dữ liệu: Hai mảng **a** (kích thước **n**) và **dp** (kích thước 2^{12}).

Độ phức tạp của giải thuật: $O(n \times 2^{12})$.

Chương trình:

```
#include <bits/stdc++.h>
#define NAME "cymbidium."
using namespace std;
ifstream fi(NAME"inp");
ofstream fo (NAME"out");
const int N = 1 << 12;
int ans[N];

int main()
{
    int n, k;
    fi >> n >> k;
    vector <int> a(n);
    for (int i = 0; i < n; i++) fi >> a[i];
    for (int i = 0; i < N - 1; i++) ans[i] = n + 1; ans[N-1]=0;
    ans[a[0]]=1;
    for (int i = 1; i < n; i++)
        for (int j = 0; j < N; j++)
            ans[a[i] & j] = min(ans[j] + 1, ans[a[i] & j]);
    ans[0] > k ? fo << "NO" : fo << "YES";
    fo<<"\nTime: "<<clock() / (double)1000<<" sec";
    return 0;
}
```



VT45. CỐC THÍ NGHIỆM

Tên chương trình: GLASSES.CPP

Để chuẩn bị thí nghiệm mới, giáo sư Braun đặt lên bàn n cốc thủy tinh, mỗi cốc có hình lăng trụ đứng, đáy là một hình đa giác lồi và yêu cầu người trợ lý rót tất cả dung môi từ một bình chứa v đơn vị thể tích vào các cốc sao cho độ cao dung môi ở các cốc là như nhau.

Hãy xác định độ cao dung môi ở cốc sau khi rót và đưa ra với độ chính xác 8 chữ số sau dấu chấm thập phân.

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

- ✚ Dòng đầu tiên chứa 2 số nguyên n và v ($1 \leq n \leq 10^5, 1 \leq v \leq 10^{18}$),
- ✚ Tiếp theo là n nhóm dòng mô tả các cốc:
 - ❖ Dòng thứ nhất trong nhóm chứa số nguyên k – số đỉnh của đa giác lồi ($3 \leq k \leq 10^4$),
 - ❖ Dòng thứ j trong k dòng tiếp theo chứa 2 số nguyên xj và yj – tọa độ của một đỉnh đa giác lồi, các đỉnh liệt kê theo chiều ngược kim đồng hồ, các tọa độ có giá trị tuyệt đối không vượt quá 10^9 .

Tổng số đỉnh trong tất cả các cốc không vượt quá 10^6 .

Các tọa độ đỉnh chỉ phục vụ mô tả hình dáng đa giác, vì vậy các đa giác có thể giao nhau.

Kết quả: Đưa ra file văn bản GLASSES.OUT một số thực với độ chính xác 10^{-8} – độ cao tìm được.

Ví dụ:

GLASSES.INP
2 10
3
0 0
1 0
0 1
4
1 1
0 2
0 0
1 0

GLASSES.OUT
5.00000000



VT45 IO20161105 C

Giải thuật: Diện tích đa giác lồi.

Nhận xét:

Thể tích lăng trụ đứng bằng diện tích đáy nhân chiều cao: $\mathbf{V} = \mathbf{S} \times \mathbf{h}$,

Lượng nước rót vào các cốc tương đương lượng nước rót vào một cốc có diện tích đáy bằng tổng diện tích đáy các cốc,

Gọi tổng diện tích đáy các cốc là **sum_area**, ta có chiều cao cần tìm $\mathbf{h} = \mathbf{v} / \mathbf{sum_area}$,

Như vậy vấn đề chính là tìm diện tích đáy mỗi cốc,

Công thức tính diện tích đa giác có cạnh không tự cắt:

$$\mathbf{S} = \frac{1}{2} \left| \sum_{i=0}^{n-1} (x_i y_{i+1} - x_{i+1} y_i) \right|$$

Trong đó $\mathbf{x}_n \equiv \mathbf{x}_0, \mathbf{y}_n \equiv \mathbf{y}_0$.

Để thuận tiện tính toán cần bổ sung thêm phần tử $(\mathbf{x}_n, \mathbf{y}_n)$ với $\mathbf{x}_n = \mathbf{x}_0, \mathbf{y}_n = \mathbf{y}_0$,

Để chống tích lũy sai số làm tròn ta sẽ tính $2 \times \mathbf{sum_area}$ và chỉ chia cho 2 khi tính kết quả cuối cùng.

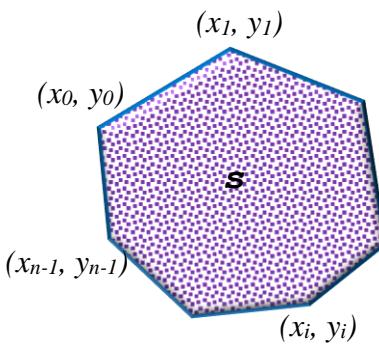
Tổ chức dữ liệu:

Hai mảng động `vector<int64_t>x(k+1), y(k+1)` để lưu trữ tọa độ đỉnh của đa giác đáy cốc.

Xử lý:

Không cần lưu trữ thông tin của tất cả các đa giác đáy cốc.

Độ phức tạp của giải thuật: $O(k)$, trong đó k – tổng số đỉnh của các cốc.



Chương trình:

```
#include <bits/stdc++.h>
#define NAME "glasses."
using namespace std;
ifstream fi (NAME"inp");
ofstream fo (NAME"out");
int n;
uint64_t v,sum_area;
double h;

int main()
{
    fi >> n >> v;
    sum_area=0;
    for(int i=0;i<n;++i)
    {int k;
        fi>>k;
        vector<int64_t>x(k+1),y(k+1);
        for(int j=0;j<k;++j) fi>>x[j]>>y[j];
        x[k]=x[0]; y[k]=y[0];
        int64_t s=0;
        for(int j=0;j<k;++j) s+=x[j]*y[j+1]-x[j+1]*y[j];
        sum_area+=abs(s);
    }
    h=(double)v*2/sum_area;
    fo<<setprecision(8)<<fixed<<h;
    fo<<"\nTime: "<<clock()/(double)1000<<" sec";
    return 0;
}
```



VT46. SỐ KHẢ NĂNG

Tên chương trình: CASES.CPP

Giáo sư Braun làm thí nghiệm và đọc các quả quan sát được cho trợ lý của mình ghi chép. Buổi tối ông ngồi đọc kết quả và rất thất vọng vì người trợ lý đã ghi các kết quả liên tiếp nhau thành một số **x** lớn!

Có tìm cách khôi phục lại giá trị thực Giáo sư nhớ lại 3 điều quan trọng:

- Ông đã đọc đúng **k** lần cho trợ lý ghi,
- Các số đã đọc đè không bắt đầu bằng 0 (trừ giá trị 0),
- Hai số liên tiếp nhau có chênh lệch không nhỏ hơn **a** và không lớn hơn **b**.

Với chứng đó thông tin Giáo sư thấy rằng khó có thể khôi phục lại giá trị thực, nhưng ông vẫn muốn biết từ **x** có thể có bao nhiêu cách tạo bộ dữ liệu thỏa mãn các điều kiện đã nêu.

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

- Dòng đầu tiên chứa một số nguyên **t** – số bộ dữ liệu cần xử lý ($1 \leq t \leq 100$),
- Mỗi dòng trong **t** dòng tiếp theo chứa 4 số nguyên **x**, **a**, **b** và **k** mô tả một bộ dữ liệu ($1 \leq x \leq 10^{18}$, $0 \leq a \leq b \leq 10^{18}$, $1 \leq k \leq 18$).

Kết quả: Đưa ra file văn bản CASES.OUT **t** số nguyên, mỗi số trên một dòng – số cách tách x thành bộ dữ liệu thỏa mãn các điều kiện đã nêu.

Ví dụ:

CASES.INP	CASES.OUT
3	1
248 16 45 2	2
248 16 46 2	0
4444 1 5 2	



VT46 IO20161105 D

Giải thuật: Quy hoạch động.

Nhận xét:

Việc phân chia một dãy các chữ số thành các số thỏa mãn những điều kiện đã cho là một toán quy hoạch động chuẩn,

Gọi $dp_{i,j,m}$ là số cách chia đoạn $x[0..j]$ thành m số và số cuối cùng là đoạn $x[i..j]$, $1 \leq i \leq j \leq n$ – độ dài xâu x .

Số tiếp theo cần chọn là đoạn $x[j+1..p]$ ứng với số có chênh lệch với số cuối cùng trước đó theo giá trị tuyệt đối nằm trong khoảng $[a, b]$, $p = 1, 2, \dots, n - j$.

Tổ chức dữ liệu:

Mảng 3 chiều `int dp[20][20][20]`, với giá trị đầu bằng 0.

Số x có thể lưu dưới dạng xâu hoặc số nguyên.

Xử lý:

Xử lý độc lập các bộ dữ liệu,

Cần có hàm tính và kiểm tra độ chênh lệch của 2 số, hàm biến đổi xâu thành số,

Kết quả: Tổng các $dp_{i,j,k}$ (số cuối bắt đầu từ i đến hết xâu và là số thứ k).

Độ phức tạp của giải thuật: $O(n^4)$.

Chương trình:

```
#include <bits/stdc++.h>
#define NAME "cases."
using namespace std;
const int MOD = static_cast<int>(1e9) + 7;
ifstream fi (NAME"inp");
ofstream fo (NAME"out");
int64_t a, b;

bool divisible(int64_t x, int64_t y)
{
    if (y == -1) return true;
    return a <= abs(x - y) && abs(x - y) <= b;
}

int64_t stoll(string y)
{
    int m;
    int64_t z=0;
    m=y.size();
    for(int i=0; i<m; ++i) z=z*10+y[i]-48;
    return z;
}

int main()
{
    int tests;
    fi >> tests;
    while (tests--)
    {
        string x;
        int k;
        fi >> x >> a >> b >> k;
        int dp[20][20][20]={0};
        dp[0][0][0] = 1;
        for (size_t i = 0; i <= x.size(); i++)
            for (size_t j = i; j <= x.size(); j++)
                for (int t = 0; t < k; t++)
                {
                    if (dp[i][j][t] == 0) continue;
                    if (i > 0 && x[i - 1] == '0' && j - i + 1 > 1) continue;
                    int64_t last = (i == 0 || j == 0) ? -1 : stoll(x.substr(i - 1, j - i + 1));
                    for (size_t p = j + 1; p <= x.size(); p++)
                    {
                        if (x[j] == '0' && p - j > 1) continue;
                        int64_t number = stoll(x.substr(j, p - j));
                        if (divisible(number, last))
                        {
                            dp[j + 1][p][t + 1] += dp[i][j][t];
                            if (dp[j + 1][p][t + 1] >= MOD)
                                dp[j + 1][p][t + 1] -= MOD;
                        }
                    }
                }
        int cnt = 0;
        for (size_t i = 0; i <= x.size(); i++)
    }
```

```
    cnt += dp[i][x.size()][k];
    if (cnt >= MOD) cnt -= MOD;
}
fo << cnt << '\n';
}
fo<<"\nTime: "<<clock() / (double)1000<<" sec";
return 0;
}
```



VT47. BÀN PHÍM

Tên chương trình: KEYBOARD.CPP

Trong một đợt tập huấn Tin học Steve đã có dịp làm quen với nhiều bạn bè mới. Về nhà, ngay lập tức Steve lén mạng trao đổi thông tin với các bạn mới và dĩ nhiên đầu tiên là với Ira, cô bạn xinh nhất trại tập huấn.

Thật đáng ngạc nhiên, thông tin không hiển thị đúng như khi gõ vào. Không hiểu hệ thống bị nhiễm virus hay bàn phím bị hỏng do va đập. Sau một thời gian khảo sát, Steve thấy rằng phím *Home* bị dính với một số ký tự. Sau khi gõ ký tự bị dính con trỏ tự động về đầu dòng. Ví dụ, Steve đánh vào xâu “**i****railikeyou**” thì nhận được kết quả “**ouilikeyira**” do các phím **a** và **y** bị dính. Trước khi tìm cách khắc phục Steve muốn biết bàn phím chỉ bị lỗi dính phím *Home* hay không.

Hãy dựa vào kết quả gõ, xác định nếu bàn phím chỉ bị lỗi dính phím thì đưa ra thông báo “**Yes**”, trong trường hợp còn lỗi khác – đưa ra thông báo “**No**”.

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

- + Dòng đầu tiên chứa một số nguyên **n** – độ dài thông báo ($1 \leq n \leq 5000$),
- + Dòng thứ 2 chứa xâu **s** chỉ gồm các ký tự la tinh thường, độ dài **n** – xâu gõ vào từ bàn phím,
- + Dòng thứ 3 chứa xâu **t** chỉ gồm các ký tự la tinh thường, độ dài **n** – xâu gõ nhận được trên màn hình.

Kết quả: Đưa ra file văn bản KEYBOARD.OUT thông báo “**Yes**” hoặc “**No**” nhận được theo kết quả khảo sát.

Ví dụ:

KEYBOARD.INP
11
i railikeyou
ouilikeyira

KEYBOARD.OUT
Yes



VT47 IO20161023 J

Giải thuật: Hàm Z.

Nhận xét:

Giả thiết **s** là tổng các xâu **c₁**, **c₂**, . . . , **c_k**: **s** = **c₁** + **c₂** + . . . + **c_{k-1}** + **c_k** và sau khi gõ xâu **c_i** (**i** = 1 ÷ **k**) con trỏ màn hình quay về đầu dòng ta sẽ nhận được xâu **t**:

$$t = c_k + c_{k-1} + \dots + c_2 + c_1$$

Ta sẽ dùng bảng đánh dấu để khảo sát lỗi dính phím *Home* có phải là duy nhất hay không.

Xét mảng **dp_i**, **i** = 0 ÷ **n**, trong đó **dp_i** = **true** nếu từ phần cuối của xâu **s** bắt đầu từ ký tự **i** có thể nhận được phần đầu của xâu **t** và **dp_i** = **false** trong trường hợp ngược lại, **dp_n** = **true**.

Bắt đầu với **i** = **n**, tìm tiền tố ngắn nhất của xâu **s[0..i-1]** là hậu tố của **t**, nếu độ dài tiền tố tìm được là **k** thì cho **dp_j** = **true** với **j** = **n-k** và lặp lại xử lý tương tự với cặp xâu **s[k..n-1]** và **t[0..n-k-2]**.

Quá trình xử lý kết thúc khi không tìm được tiền tố khác rỗng hoặc khi cả 2 xâu đều rỗng.

Nếu **dp₀** có giá trị true thì kết quả là “**Yes**”, trong trường hợp ngược lại – “**No**”.

Tổ chức dữ liệu:

Mảng **bool dp[5002]** – đánh dấu các đoạn thỏa mãn điều kiện chỉ tồn tại một loại lỗi dính phím *Home*,

Mảng động **vector<int> F = Z(w)** – lưu giá trị hàm Z với cặp xâu đang xét.

Xử lý:

Ví dụ:

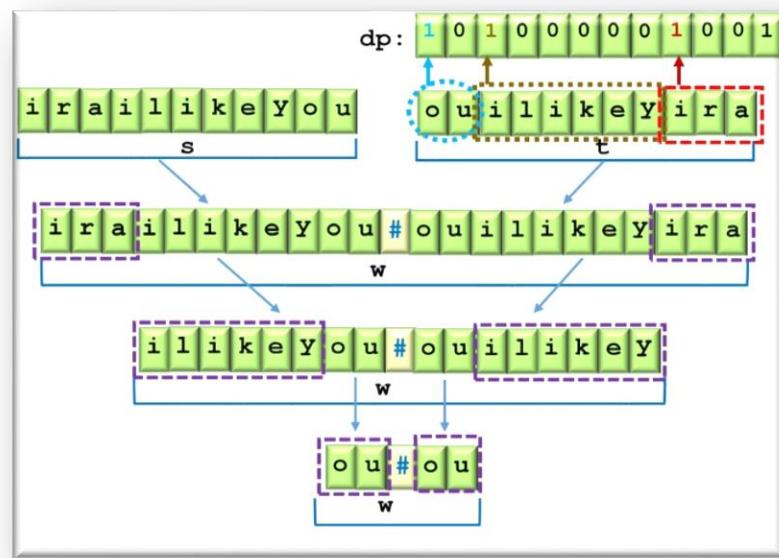
$s = \text{"irailikeyou"}$,
 $t = \text{"ouilikeyira"}$

Các bước xử lý và kết quả được mô tả ở hình bên.

Các xâu s và t được ghép lại thành xâu w với ký tự “#” phân tách 2 xâu cần xử lý.

Việc tạo xâu w cho phép ta sử dụng [Hàm Z](#) để tìm tiền tố và hậu tố chung một cách nhanh chóng.

Giải thuật xây dựng Hàm Z đã được đề cập tới ở các phần trước:



```
vector<int> Z(string s)
{
    int n = s.size();
    vector<int> z(n);
    for (int i = 1, l = 0, r = 0; i < n; ++i)
    {
        if (i <= r)      z[i] = min(r - i + 1, z[i - 1]);
        while(i+z[i] < n && s[z[i]] == s[i + z[i]]) ++z[i];
        if (i + z[i] - 1 > r){l = i; r = i + z[i] - 1;}
    }
    return z;
}
```

Sau khi có giá trị Hàm Z, chỉ cần tìm và đánh dấu các vị trí đầu của hậu tố có độ dài bằng tiền tố:

```
for(int j=0;j<i;++j)
{
    int s = j + i + 1;
    if(F[s] == i - j)dp[j] = 1;
}
```

Độ phức tạp của giải thuật: $O(n^2)$.

Chương trình:

```
#include <bits/stdc++.h>
#define NAME "keyboard."
using namespace std;
ifstream fi (NAME"inp");
ofstream fo (NAME"out");
const int dd = 5e3 + 7;
bool dp[dd];

vector<int> Z(string s)
{
    int n = s.size();
    vector<int> z(n);
    for (int i = 1, l = 0, r = 0; i < n; ++i)
    {
        if (i <= r) z[i] = min(r - i + 1, z[i - l]);
        while (i + z[i] < n && s[z[i]] == s[i + z[i]]) ++z[i];
        if (i + z[i] - 1 > r)
        {
            l = i; r = i + z[i] - 1;
        }
    }
    return z;
}

int main()
{
    int n;
    string a, b;
    fi >> n >> a >> b;
    dp[n] = 1;
    for (int i = n; i > 0; i--)
    {
        if (dp[i])
        {
            string w;
            for(int j=n-i;j<n;++j)w.push_back(a[j]);
            w.push_back('#');
            for(int j=0;j<i;++j) w.push_back(b[j]);
            vector<int> F = Z(w);
            for(int j=0;j<i;++j)
            {
                int s = j + i + 1;
                if(F[s] == i - j) dp[j] = 1;
            }
        }
    }
    if(dp[0]) fo << "Yes\n"; else fo << "No\n";
    fo<<"\nTime: "<<clock() / (double)1000<<" sec ";
}
```



VT48. SỐ HÓA TÀI LIỆU

Tên chương trình: DOCUMENTS.CPP

Viện lưu trữ Quốc gia quyết định số toàn bộ tài liệu lưu trữ. Tài liệu sẽ được cung cấp cho người tham khảo dưới dạng file kiểu Djvu. Như vậy sẽ không hạn chế số người truy cập đồng thời tới cùng một tài liệu và cũng sợ hư hỏng mất mát.

Phòng số hóa của Steve có rất nhiều máy scanner tự động chuyên dụng để copy tài liệu đưa nội dung vào file dạng Djvu. Mỗi tài liệu nhận được, sau khi scan xong phải chuyển sang bộ phận xử lý bảo quản và lưu trữ dài hạn. Bộ phận này ít người và chỉ có thể tiếp nhận xử lý mỗi ngày một tài liệu.

Steve sẽ phải xử lý n tài liệu. Khi tài liệu thứ i được chuyển tới, hệ thống sẽ tự động khảo sát sơ bộ và ghi biên bản dưới dạng 3 số nguyên s_i – ngày tài liệu được chuyển đến, f_i – ngày cuối cùng mà muộn nhất cuối ngày đó phải chuyển tài liệu này sang bộ phận xử lý bảo quản và c_i – số ngày cần thiết để scan xong tài liệu.

Ban Giám đốc Viện rất quan tâm đến tiến độ thực hiện và yêu cầu các bộ phận báo cáo khả năng hoàn thành công việc đúng tiến độ được hay không, dựa vào đó – tìm biện pháp bổ trợ phù hợp.

Hãy xác định xem Phòng số hóa có thể thực hiện đúng tiến độ được hay không và đưa ra thông báo “**YES**” hoặc “**NO**” tương ứng.

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

- ✚ Dòng đầu tiên chứa một số nguyên n ($1 \leq n \leq 10^5$),
- ✚ Dòng thứ i trong n dòng sau chứa 3 số nguyên s_i , f_i và c_i ($1 \leq s_i, f_i, c_i \leq 10^9$).

Kết quả: Đưa ra file văn bản DOCUMENTS.OUT thông báo xác định được.

Ví dụ:

DOCUMENTS.INP
3
2 7 3
3 8 1
1 4 2

DOCUMENTS.OUT
YES



Giải thuật: Xử lý đồng bộ thời gian.

Nhận xét:

Với mỗi tài liệu cần lưu giữ 2 tham số chủ chốt: ngày hoàn tất scan s_i+c_i và hạn cuối cùng phải trả f_i ,

Các dữ liệu này sẽ lưu trữ trong mảng cặp giá trị và sắp xếp theo tham số thứ nhất đã nêu,

Bắt đầu từ ngày hoàn tất sớm nhất j , với các tài liệu có cùng ngày hoàn thiện j nạp hạn cuối và số thứ tự của nó vào tập hợp tạo thành một vun đống theo chiều tăng dần của thời hạn hoàn thiện.

Nếu thời điểm hoàn thiện của phần tử đầu trong vun đống nhỏ hơn hoặc bằng thời điểm xét j – xóa nó khỏi vun đống và cập nhật lại thời điểm xét j : j sẽ tăng *lên 1* nếu vun đống còn khác rỗng hoặc bằng *thời điểm hoàn tất gần nhất chưa xét* trong mảng trong trường hợp ngược lại,

Trong quá trình xét nếu gặp phần tử có ngày hoàn tất nhỏ hơn thời điểm đang xét thì đưa ra thông báo **NO** và kết thúc xử lý,

Việc duyệt các phần tử ở mảng lưu trữ để đưa vào vun đống *thực hiện song song* với quá trình nhận biết thời điểm trả tài liệu.

Sau khi các phần tử ở mảng ban đầu đã được đưa vào vun đống, nếu vun đống chưa rỗng thì cần *xử lý kết thúc*: xác định thời điểm trả tài liệu với thời gian xét tăng dần 1. Nếu trả hết được mọi tài liệu – kết quả sẽ là **YES**.

Tổ chức dữ liệu:

- Mảng **pair <int, int>** **p[100005]** lưu các cặp giá trị (s_i+c_i, f_i),
- Tập **set <pair<int,int>>** **s** lưu vun đống.

Xử lý:

- Nhập dữ liệu, tạo mảng **p** và sắp xếp,
- Tạo vun đống song song với việc lọc dữ liệu khỏi vun đống,
- Xử lý kết thúc.

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

Chương trình:

```
#include <bits/stdc++.h>
#define NAME "documents."
using namespace std;
ifstream fi (NAME"inp");
ofstream fo (NAME"out");
set<pair<int, int>> s;
pair<int, int> p[100005];

int main()
{
    int n;
    fi >> n;

    for (int i = 0; i < n; i++)
    {
        int a, b, c;
        fi >> a >> b >> c;
        if (a + c > b)
        {
            fo << "NO" << endl;
            return 0;
        }
        p[i].first = a + c;
        p[i].second = b;
    }
    sort(p, p + n);

    int j = p[0].first;
    int cnt = 0;
    for (int i = 0; i < n; i++)
    {
        while (i < n && p[i].first == j)
        {
            s.insert(make_pair(p[i].second, i));
            cnt++;
            i++;
        }
        pair<int, int> v = *s.begin();
        if (v.first < j)
        {
            fo << "NO" << endl;
            return 0;
        }
        s.erase(v);
        cnt--;
        if (cnt == 0 && i != n)
            j = p[i].first;
        else
            j++;
        i--;
    }
    while (cnt > 0)
    {
        pair<int, int> v = *s.begin();
        if (v.first < j)
        {
            fo << "NO" << endl;
            return 0;
```

```
        }
        s.erase(v);
        j++;
        cnt--;
    }
fo << "YES" << endl;
fo<<"\nTime: "<<clock() / (double)1000<<" sec";
return 0;
}
```



VT49. SỐ BUỒN TẺ

Tên chương trình: BORING.CPP

Sự đơn điệu dễ mang tới cảm giác nhảm chán. Nếu phải nghe một diễn giả trình bày một vấn đề nào đó bằng một giọng đều đều vô cảm thì chỉ 30 phút sau cả hội trường đã im phăng phắc: không một ai trong số thính giả có thể gượng nỗi trước cảm dỗ của con buồn ngủ chợt ập tới và mỗi người sẽ chìm vào một giấc mơ đẹp của riêng mình. Nhưng trong cuộc sống tất bật, đa dạng nhiều khi sự đơn điệu lại rất có giá trị. Những số điện thoại, biển số xe chỉ toàn chữ số 1 hay toàn chữ số 6, mặc dù đó là những số rất buồn tẻ - chỉ chứa có một loại chữ số, nhưng lại được bán rất đắt!

Hãy xác định trong khoảng $[a, b]$ ($a \leq b$) có bao nhiêu số buồn tẻ. Ví dụ, trong khoảng $[300, 500]$ có 2 số buồn tẻ, đó là 333 và 444.

Dữ liệu: Vào từ file văn bản BORING.INP gồm một dòng chứa 2 số nguyên a và b ($1 \leq a \leq b \leq 10^{18}$).

Kết quả: Đưa ra file văn bản BORING.OUT số lượng số buồn tẻ tìm được.

Ví dụ:

BORING.INP	BORING.OUT
300 500	2



Giải thuật: Tổng tiền tố.

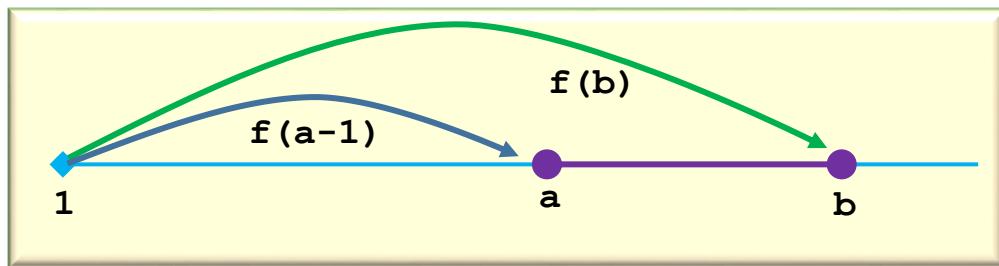
Nhận xét:

Công cụ hiệu quả để tính tích lũy một đại lượng nào đó trong một khoảng cho trước là tổng tiền tố,

Tổng tiền tố có thể biểu diễn dưới dạng mảng số hoặc hàm,

Ở bài toán này ta sẽ sử dụng hàm $f(x)$ – xác định số lượng số buồn tẻ trong khoảng từ 1 đến x ,

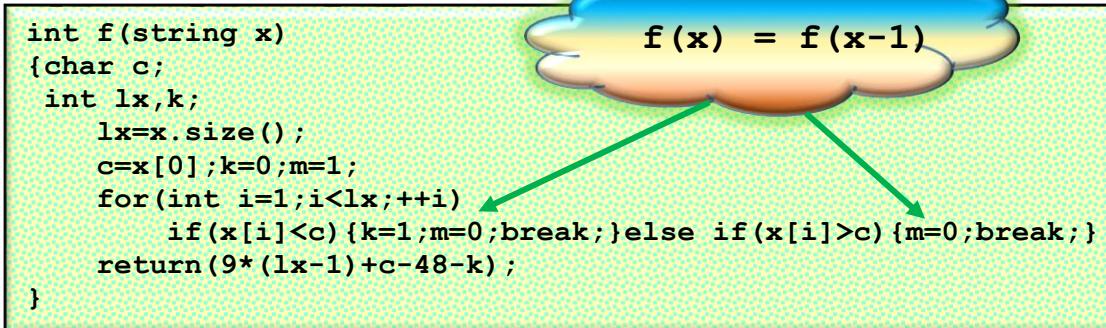
Để dàng thấy rằng kết quả cần tìm $ans = f(b) - f(a-1)$.



Tổ chức dữ liệu: Các số a, b nên lưu trữ dưới dạng xâu.

Xử lý:

Ta sẽ vòng tránh việc thực hiện phép $a-1$ đối với xâu bằng cách nhận dạng x có phải là một số buồn tẻ hay không khi tính $f(x)$:



Độ phức tạp của giải thuật: $O(1)$.

Chương trình:

```
#include <bits/stdc++.h>
#define NAME "boring."
using namespace std;
ifstream fi (NAME"inp");
ofstream fo (NAME"out");
string a, b;
int m,ans;
int f(string x)
{char c;
 int lx,k;
 lx=x.size();
 c=x[0];k=0;m=1;
 for(int i=1;i<lx;++i) if(x[i]<c) {k=1;m=0;break;} else
if(x[i]>c) {m=0;break;}
 return(9*(lx-1)+c-48-k);
}

int main()
{
    fi>>a>>b;
    ans=f(b)-f(a)+m;
    fo<<ans;
}
```



VT50. TẢO BIỂN

Tên chương trình: SEAWEED.CPP

Tảo biển sinh sản rất nhanh khi có môi trường thuận lợi với chúng và có những loài còn tiết ra môi trường những chất độc hại.

Một loại tảo nâu trong môi trường nước bị ô nhiễm nặng sinh sản theo quy luật sau:

- Ngày đầu tiên (ngày 0) có n cá thể ở mức 1,
- Ở mỗi ngày tiếp theo, mỗi cá thể mức i sinh ra i cá thể mức 1, các cá thể mới sinh sẽ sinh sôi, phát triển từ ngày hôm sau.
- Bản thân các cá thể mức i phát triển thành mức $i+1$ và chu kỳ phát triển trong ngày chấm dứt.

Hãy xác định sau k ngày trong nước biển có bao nhiêu cá thể.

Dữ liệu: Vào từ file văn bản SEAWEED.INP gồm một dòng chứa 2 số nguyên n và k ($1 \leq n \leq 1000$, $1 \leq k \leq 10^5$).

Kết quả: Đưa ra file văn bản SEAWEED.OUT một số nguyên – số lượng cá thể tảo theo mô đun 10^9+7 .

Ví dụ:

SEAWEED.INP
3 2

SEAWEED.OUT
15



Giải thuật: Nhận dạng giải thuật, công thức lặp.

Nhận xét:

Các cá thể sinh trưởng và phát triển độc lập với nhau, vì vậy ta chỉ cần xét từ 1 cá thể ban đầu, sau k ngày nhận được bao nhiêu cá thể, gọi số cá thể nhận được là **res1** thì kết quả cần tìm sẽ là **ans = res1 × n**,

Gọi a_i – số cá thể ngày thứ i , $a_0 = 1$,

Ta có $a_1 = 2, a_2 = 5, a_3 = 13, \dots$

Tổng quát: $a_n = a_{n-1} + \sum_{i=1}^{n-1} a_i$.

Gọi f_i là số Fibonacci thứ i , $f_0 = f_1 = 1, f_i = f_{i-1} + f_{i-2}, i \geq 2$.

Ta có:

i	a _i	i	f _i
0	1	0	1
1	2	1	1
2	5	2	2
3	13	3	3
4	34	4	5
		5	8
		6	13
		7	21
		8	34

Số lượng cá thể tạo

Số Fibonacci

Bằng quy nạp có thể dễ dàng chứng minh được rằng $a_k = f_{2k}$.

Độ phức tạp của giải thuật: $O(k)$.

Chương trình:

```
#include <bits/stdc++.h>
#define NAME "seaweed."
using namespace std;
ifstream fi (NAME"inp");
ofstream fo (NAME"out");
const int mod = 1e9 + 7;

int main()
{
    int n, k;
    fi >> n >> k;

    int f0, f1, f;
    f0 = f1 = 1;
    for (int i = 2; i <= 2 * k; i++)
    {
        f = (f0 + f1) % mod;
        f0 = f1;
        f1 = f;
    }
    f1 = ((int64_t) f1 * n) % mod;
    fo << f1;
    fo<<"\nTime: "<<clock() / (double)1000<<" sec";
    return 0;
}
```

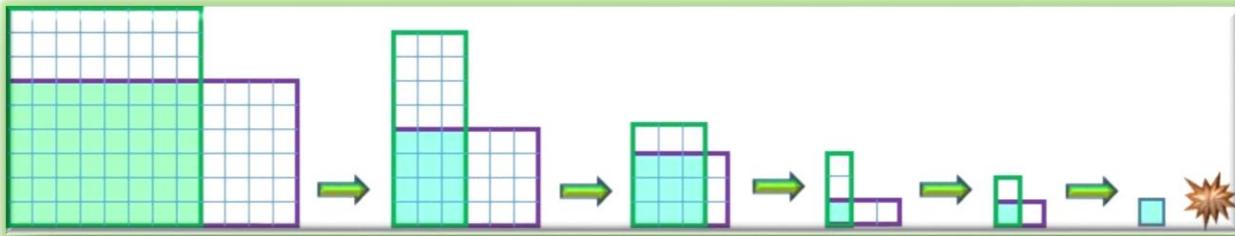


VU01. ẢO THUẬT

Tên chương trình: TRICK.CPP

Giáo sư Braun kể cho cô trợ lý trẻ của mình một trò ảo thuật hồi nhỏ đã xem trên TV đã làm ông kinh ngạc và nhớ mãi.

Nhà ảo thuật lấy ra 2 tấm bìa carton hình chữ nhật, một tấm kích thước $a \times b$ cm, tấm kia – $c \times d$ cm, đặt tấm bìa đầu lên bàn sao cho cạnh dài hơn song song với mép ngang của bàn, sau đó đặt tấm bìa thứ 2 đè lên tấm bìa thứ nhất, cạnh dài hơn song song với cạnh dọc của bàn và 2 tấm bìa có góc chung trùng nhau. Ông bật 2 ngón tay, sau tiếng tách, phần chung biến mất một cách bí ẩn. Các thao tác đó được lặp lại với 2 tấm bìa còn lại trên bàn vài lần cho đến khi có 2 tấm bìa hình vuông kích thước giống nhau và cuối cùng – tất cả biến mất!



Cô trợ lý cũng rất hứng thú với câu chuyện, nhưng ở khía cạnh khác: có phải với hai tấm bìa hình chữ nhật bất kỳ cũng làm cho chúng biến mất hoàn toàn hay không và nếu được thì phải qua bao nhiêu lần đặt chồng hình?

Với các đại lượng a, b, c, d cho trước hãy xác định xem có thể cuối cùng làm cho 2 tấm bìa biến mất, đưa ra câu trả lời “YES” hoặc “NO”. Trong trường hợp có thể làm biến mất cả 2 tấm bìa thì đưa ra trên dòng tiếp theo một số nguyên – số lần chồng hình trong quá trình biểu diễn.

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

Kết quả: Đưa ra file văn bản TRICK.OUT kết quả tìm được.

Ví dụ:

TRICK.INP
12 6 8 9

TRICK.OUT
YES 6



Giải thuật: Tìm kiếm nhị phân.

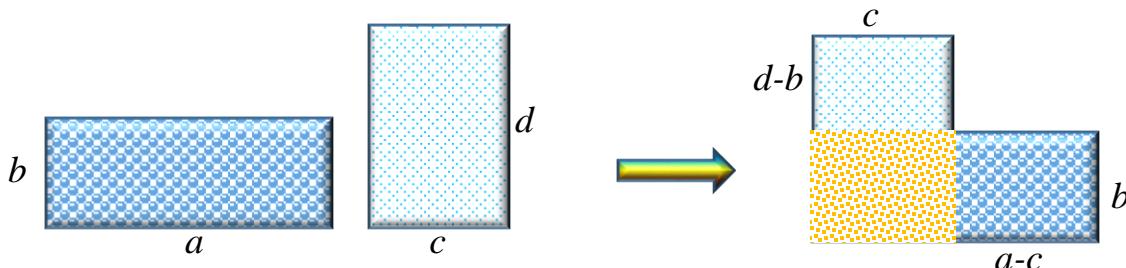
Nhận xét:

Cần và đủ để kết quả chèo hình làm cả hình chữ nhật ban đầu biến mất là hai tấm bìa có cùng diện tích: $\mathbf{a} \times \mathbf{b} = \mathbf{c} \times \mathbf{d}$,

Việc tính diện tích đòi hỏi phải xử lý số 128 bit,

Khi có diện tích bằng nhau: tìm số bước biến đổi,

Để thuận tiện xử lý cần chuẩn hóa dữ liệu theo tiêu chuẩn $\mathbf{a} \geq \mathbf{b}$ và $\mathbf{c} \leq \mathbf{d}$,



Sau một lần biến đổi độ dài cạnh \mathbf{a} trở thành $\mathbf{a}-\mathbf{c}$, độ dài cạnh \mathbf{d} trở thành $\mathbf{d}-\mathbf{b}$,

Chuẩn hóa dữ liệu và lặp lại phép biến đổi trên cho đến khi có $\mathbf{a} = \mathbf{b} = \mathbf{c} = \mathbf{d}$, ảo thuật kết thúc sau khi thêm một lần đặt chèo hình nữa.

Nếu một cạnh nào đó của một tấm bìa quá nhỏ so với các cạnh của tấm bìa khác quá trình tính toán mô phỏng trên sẽ làm việc rất chậm,

Để giảm độ phức tạp của giải thuật cần tìm số lượng phép biến đổi mà trong quá trình đó *kích thước các tấm bìa vẫn giữ nguyên điều kiện chuẩn hóa* đã nêu, số lượng phép biến đổi có thể xác định bằng phương pháp *tìm kiếm nhị phân*,

Nếu việc tìm kiếm nhị phân cho kết quả 0 – cần *thực hiện một phép biến đổi* trước khi tiếp tục tìm kiếm nhị phân.

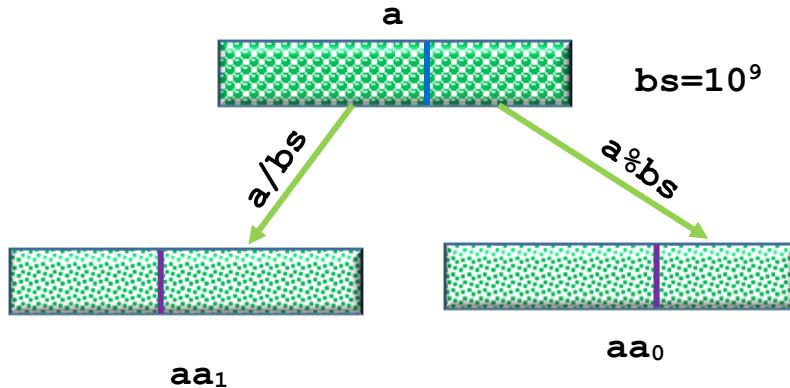
Tổ chức dữ liệu: chỉ cần tổ chức mảng phục vụ nhân số lớn, các mảng này được cục bộ hóa trong hàm kiểm tra.

Xử lý:

Hàm kiểm tra diện tích:

Tích của 2 số kiểu `int64_t` là một số 128 bit, ở cơ số 10 là số có 37 – 38 chữ số,

Dữ liệu trong bài toán có kích thước không vượt quá 10^{18} , vì vậy mỗi số kiểu `int64_t` có thể lưu trữ dưới dạng 2 thành phần, để tiện tính toán, mỗi thành phần cũng có kiểu `int64_t`:



Dữ liệu:

```
array<int64_t,5>aa={0},bb={0},cc={0},dd={0},t,ta={0},tc={0};
```

*Để thuận tiện
lập trình*

Nhân 2 số có giá trị không vượt quá 10^{18} :

```
for(int i=0;i<5;++i)t[i]=0,ta[i]=0;
nh=0;
for(int i=0;i<5;++i)
{ tg=aa[i]*bb[0]+nh;ta[i]=tg%bs;nh=tg/bs; }
t[0]=0;
for(int i=0;i<4;++i)
{ tg=aa[i]*bb[1]+nh;t[i+1]=tg%bs;nh=tg/bs; }
for(int i=0;i<5;++i)
{ tg=t[i]+ta[i]+nh;ta[i]=tg%bs;nh=tg/bs; }
```

So sánh 2 diện tích:

Diện tích hình 1

Diện tích hình 2

```
r=1;
for(int i=0;i<5;++i)r&=(ta[i]==tc[i]);
return r;
```

Tính số lần chồng hình khi có nghiệm:

```
int k=0;
while(clock() / (double) CLOCKS_PER_SEC <= 1.00)
{
    if (a < b) swap(a, b);
    if (c > d) swap(c, d);

    int64_t l = 0, r = 0;
    if (b && c) {l = 0; r = min(d / b + 7, a / c + 7);}
    bool good = 0;
    while (l < r - 1)
    {
        int64_t m = (l + r) / 2;
        bool fa = (a - (m - 1) * c > c);
        bool fd = (d - (m - 1) * b > b);
        if (fa && fd) {good = 1;l = m;}
        else r = m;
    }
    if (good){ a -= l * c; d -= l * b; ans += l;}
    if (a == b && a == c && a == d)
    {
        fo << ans + 1 << '\n';
        fo << "\nTime: " << clock() / (double)1000000 << " sec";
        return 0;
    }
    if(k!=ans)k=ans; else a-=c, d-=b, ++ans;
}
```

Trường hợp kết quả tìm kiếm nhị phân bằng 0

Độ phức tạp của giải thuật: $\approx O(64)$.

Chương trình:

```
#include <bits/stdc++.h>
#define NAME "trick."
using namespace std;
ifstream fi (NAME"inp");
ofstream fo (NAME"out");
const uint64_t bs=(uint64_t)1000000000;
int64_t a, b, c, d;

bool check_s()
{array<int64_t,5>aa={0},bb={0},cc={0},dd={0},t,ta={0},tc={0};
 int64_t tg;
 int64_t nh;
 bool r;
 aa[0]=a%bs; aa[1]=a/b;
 bb[0]=b%bs; bb[1]=b/b;
 cc[0]=c%bs; cc[1]=c/b;
 dd[0]=d%bs; dd[1]=d/b;
 for(int i=0;i<5;++i)t[i]=0,ta[i]=0;
 nh=0;
 for(int i=0;i<5;++i)
 {
     tg=aa[i]*bb[0]+nh;ta[i]=tg%bs;nh=tg/b;
 } nh=0;t[0]=0;
 for(int i=0;i<4;++i)
 {
     tg=aa[i]*bb[1]+nh;t[i+1]=tg%bs;nh=tg/b;
 }
 for(int i=0;i<5;++i)
 {
     tg=t[i]+ta[i]+nh;ta[i]=tg%bs;nh=tg/b;
 }

 for(int i=0;i<5;++i)t[i]=0,tc[i]=0;
 for(int i=0;i<5;++i)
 {
     tg=cc[i]*dd[0]+nh;tc[i]=tg%bs;nh=tg/b;
 }t[0]=0;
 for(int i=0;i<4;++i)
 {
     tg=cc[i]*dd[1]+nh;t[i+1]=tg%bs;nh=tg/b;
 }
 for(int i=0;i<5;++i)
 {
     tg=t[i]+tc[i]+nh;tc[i]=tg%bs;nh=tg/b;
 }
 r=1; for(int i=0;i<5;++i)r&=(ta[i]==tc[i]);
 return r;
}

int main()
{
    fi >> a >> b >> c >> d;
    int64_t ans = 0;

    if(!check_s())
    {
        fo << "NO\n";
    }
    else
        fo << "YES\n";
}
```

```

    fo<<"\nTime: "<<clock() / (double)1000<<" sec";
    return 0;
} fo << "YES\n";
int k=0;
while(clock() / (double) CLOCKS_PER_SEC <= 1.00)
{
    if (a < b) swap(a, b);
    if (c > d) swap(c, d);
    int64_t l = 0, r = 0;

    if (b && c) {l = 0; r = min(d / b + 7, a / c + 7);}
    bool good = 0;
    while (l < r - 1)
    {
        int64_t m = (l + r) / 2;
        bool fa = (a - (m - 1) * c > c);
        bool fd = (d - (m - 1) * b > b);
        if (fa && fd) {good = 1;l = m;}
        else r = m;
    }
    if (good) { a -= l * c; d -= l * b; ans += l;}
    if (a == b && a == c && a == d)
    {
        fo << ans + 1 << '\n';
        fo<<"\nTime: "<<clock() / (double)1000<<" sec";
        return 0;
    }
    if(k!=ans) k=ans; else a-=c, d-=b, ++ans;
}
fo<<"\nTime: "<<clock() / (double)1000<<" sec";
}

```



Khoảng cách $d(\mathbf{A}, \mathbf{B})$ giữa 2 điểm \mathbf{A} tọa độ (\mathbf{p}, \mathbf{q}) và \mathbf{B} tọa độ (\mathbf{u}, \mathbf{v}) trên mặt phẳng có thể định nghĩa theo nhiều cách khác nhau, ví dụ 2 trong số các định nghĩa đó là *khoảng cách Euclidean* $d(\mathbf{A}, \mathbf{B}) = \sqrt{(p-u)^2 + (q-v)^2}$ hoặc *khoảng cách Manhattan* $d(\mathbf{A}, \mathbf{B}) = |\mathbf{p}-\mathbf{u}| + |\mathbf{q}-\mathbf{v}|$. Không phụ thuộc vào định nghĩa, các tính chất toán học vẫn được bảo toàn! Tuy vậy, về mặt định lượng thì không.

Nhưng cũng có các trường hợp riêng. Xét định nghĩa khoảng cách theo 2 cách nêu trên. Với một số cặp điểm \mathbf{A} và \mathbf{B} , giá trị của khoảng cách tính theo 2 định nghĩa đều như nhau. Với các cặp điểm này các tính chất định lượng vẫn được bảo toàn.

Cho n điểm trên mặt phẳng, điểm thứ i có tọa độ $(\mathbf{x}_i, \mathbf{y}_i)$, $i = 1 \dots n$. Các điểm không nhất thiết khác nhau. Hãy xác định *trong số các điểm đã nêu* có bao nhiêu cặp điểm mà khoảng cách giữa chúng theo cả 2 định nghĩa nêu trên đều có giá trị như nhau.

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

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

Kết quả: Đưa ra file văn bản PAIRS.OUT một số nguyên – số cặp điểm tìm được.

Ví dụ:

PAIRS.INP	PAIRS.OUT
<pre> 6 0 0 0 1 0 2 -1 1 0 1 1 1 </pre>	<pre> 11 </pre>



VU02 Mos20160307 E

Giải thuật: Khoảng giá trị và đếm cấu hình.

Nhận xét:

Một cặp điểm trên mặt phẳng có giá trị khoảng cách giống nhau theo 2 định nghĩa
đã nêu khi và chỉ khi có cùng tọa độ x hay cùng tọa độ y,

Như vậy ở mỗi dòng / cột có điểm đã cho cần đếm dòng / cột đó chứa bao nhiêu
điểm trong số đã cho, xác định số cặp có thể chọn và tích lũy vào kết quả,

Việc tính riêng từng dòng / cột sẽ không dẫn đến việc đếm lặp,

Nhóm các điểm trùng nhau bị tính hai lần vì vậy loại bỏ một lần đếm.

Tổ chức dữ liệu:

Dùng mảng động `vector<pair<int, int>> a` để lưu tọa độ các điểm.

Xử lý:

Nhập dữ liệu và sắp xếp theo tọa độ **x** để tìm các điểm cùng hàng,

Tính số cặp điểm cùng hàng:

```

int64_t count()
{
    int j;
    int64_t ans = 0;
    for (int i = 0; i < n;)
    {
        for (j=i; (j<n) && (a[j].first== a[i].first); ++j)
            if(j-i>1) ans += (int64_t)(j - i)*(j-i-1)/2;
        i = j;
    }
    return ans;
}

```

Cách tìm kiếm nêu trên cho phép vòng tránh phần tử hàng rào.

Đảo vai trò 2 tọa độ **x**, **y** và lặp lại việc sắp xếp, tìm kiếm,

Tìm các đoạn chứa điểm giống nhau, chỉnh lý lại kết quả.

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

Chương trình:

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

/*
bool cmpX(const Pt &a, const Pt &b)
{
    if (a.first != b.first) return a.first < b.first;
    return a.second < b.second;
}
*/
int n;
int64_t count()
{
    int j;
    int64_t ans = 0;
    for (int i = 0; i < n;) {
        for (j = i; (j < n) && (a[j].first == a[i].first); ++j);
        if(j-i>1)ans += (int64_t)(j - i) * (j - i - 1) / 2;
        i = j;
    }
    return ans;
}

int main()
{
    fi>>n;
    a.resize(n);
    for (int i = 0; i < n; ++i)
        fi>>a[i].first>>a[i].second;
    int ans = 0;
    sort(a.begin(), a.end());
    ans += count();
    for (int i = 0; i < n; ++i)
        swap(a[i].first, a[i].second);
    sort(a.begin(), a.end());
    ans += count();
    for (int i = 0; i < n;)
    {
        int j;
        for (j = i; (j < n) && (a[j].first == a[i].first) && (a[j].second ==
a[i].second); ++j);
        if(j-i>1)ans -= (int64_t)(j - i) * (j - i - 1) / 2;
        i = j;
    }
    fo<<ans;
    fo<<"\nTime: "<<clock() / (double)1000<<" sec";
}
```



VU03. NHỆN

Tên chương trình: SPIDERS.CPP

Một loài nhện sinh sôi nảy nở rất nhanh trong môi trường mới có nguy cơ gây mất cân bằng sinh thái. Vì vậy người ta đã bỏ nhiều công sức nghiên cứu quá trình phát triển của chúng để tìm cách kiểm soát trong môi trường thực.

Từ một nhện cái ban đầu người ta đã phát triển được thành một đàn gồm n nhện cái. Các nhện đực đều bị ăn thịt sau khi hoàn thành xong vai trò truyền giống. Nhện thứ i có tuổi là a_i và có tiềm năng sinh ra c_i nhện cái con. Dĩ nhiên, tuổi của nhện con phải nhỏ hơn tuổi nhện mẹ.

Hãy xác định xem dữ liệu ghi lại ở trên có hợp lý hay không và đưa ra thông báo “NO” nếu không hợp lý. Nếu hợp lý thì đưa ra thông báo “YES” và với con nhện thứ i – chỉ ra số thứ tự tương ứng với nhện mẹ ($i = 1 \div n$). Nhện được đánh số từ 1 trở đi. Với nhện cái ban đầu số thứ tự tương ứng với nhện mẹ của nó là 0.

Dữ liệu: Vào từ file văn bản SPIDERS.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 a_i và c_i ($1 \leq a_i \leq 10^9$, $0 \leq c_i \leq n-1$).

Kết quả: Đưa ra file văn bản SPIDERS.OUT, dòng đầu tiên chứa thông báo “NO” hoặc “YES”. Trong trường hợp thông báo tìm được là “YES” – dòng thứ 2 chứa n số nguyên, số thứ i cho biết nhện mẹ của nhện i xác định trong file input.

Ví dụ:

SPIDERS.INP	SPIDERS.OUT
5	YES
5 4	0 1 2 1 1
4 1	
1 0	
2 0	
3 0	



VU03_lo20161126_A

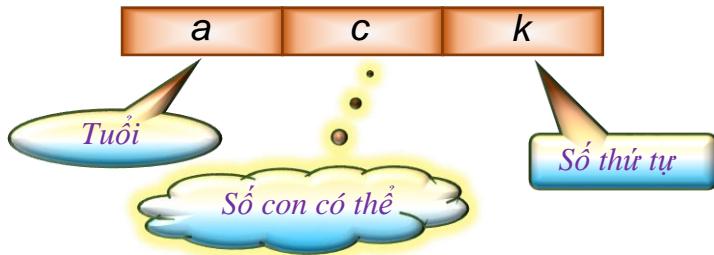
Giải thuật: Tổ chức dữ liệu.

Nhận xét:

- Có nhiều cây trực hệ khác nhau thỏa mãn điều kiện cần tìm,
- Xử lý theo giải thuật tham lam: gắn **tối đa có thể** cho các nhện mẹ **bắt đầu từ tuổi thấp nhất** trở lên, nếu mọi nhện đều nhện mẹ thì có kết quả **YES**.

Tổ chức dữ liệu:

Thông tin về mỗi nhện được lưu trong cấu trúc **Spd** chứa 3 trường nguyên:



Mảng **Spd s [N]** – ghi nhận dữ liệu vào.

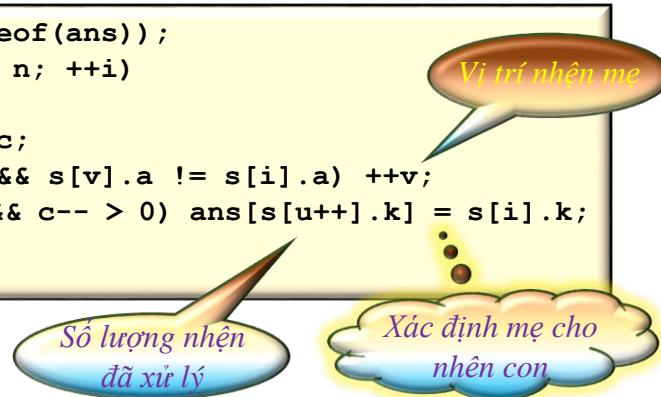
Xử lý:

Ghi nhận dữ liệu,

Sắp xếp các bản ghi theo trình tự tuổi tăng dần,

Duyệt mảng từ đầu về cuối, xác định mẹ cho các nhện con:

```
memset(ans, -1, sizeof(ans));
for (int i = 0; i < n; ++i)
{
    int c = s[i].c;
    while (v < i && s[v].a != s[i].a) ++v;
    while (u < v && c-- > 0) ans[s[u++].k] = s[i].k;
}
```



Dựa vào số nhện đã xử lý đưa ra các kết quả tương ứng.

Độ phức tạp của giải thuật: $O(nlnn)$.

Chương trình:

```
#include <bits/stdc++.h>
#define NAME "spiders."
using namespace std;
ifstream fi (NAME"inp");
ofstream fo (NAME"out");
const int N = 200002;

struct Spd
{
    int a, c;
    int k;
};

int n;
Spd s[N];
int ans[N];

int main()
{
    fi>>n;
    for (int i = 0; i < n; ++i)
    {
        fi>>s[i].a>>s[i].c;
        s[i].k = i;
    }
    sort(s, s + n, [] (const Spd & x, const Spd & y) { return x.a < y.a; });
    int u = 0, v=0;
    memset(ans, -1, sizeof(ans));
    for (int i = 0; i < n; ++i)
    {
        int c = s[i].c;
        while (v < i && s[v].a != s[i].a) ++v;
        while (u < v && c-- > 0) ans[s[u++].k] = s[i].k;
    }
    if (u == n - 1)
    {
        fo<<"YES\n";
        for (int i = 0; i < n; ++i)
            fo<<ans[i] + 1<<' ';
    } else
        fo<<"NO";
    fo<<"\nTime: "<<clock() / (double)1000<<" sec";
}
```



VU05. SỐ DƯ NHỎ NHẤT

Tên chương trình: MINRMD.CPP

Thầy giáo Tin học thu hút cảm tình lớn của học sinh toàn lớp bởi tính tình vui vẻ, khiếu hài hước và khả năng ứng xử linh hoạt. Các bài tập khô khan trở nên sinh động và hấp dẫn dưới sự trình bày của thầy.

“Hôm nay ta sẽ làm bài tập 3 trong I” – Thầy thông báo đâu giờ học: “Trên màn hình sẽ có 2 số nguyên n và m . Các chữ số của n sẽ có một trong 2 màu đen hoặc xanh. Với số n đã cho, số nhận được bằng cách thay các chữ số màu xanh bằng số 0 và số nhận được bằng việc thay các chữ số màu đen bằng 0 hãy tìm số dư khi chia cho m ”. Máy chiếu được bật lên và các số n , m hiện lên trên màn hình. Thật không may, chiếc máy chiếu cũ lại trở chứng đồng đảnh, không hiển thị tốt một số màu, trong đó có màu xanh! Chỉ còn nhìn rõ các chữ số màu đen, còn ở vị trí các chữ số màu xanh là một vệt hồng tím! Có lẽ bảng giải mã các tia B và G bị hỏng.

Sau một hồi chỉnh lý máy chiếu mà không được, thầy giáo nói với cả lớp: “Đã vậy, chúng ta phải làm bài tập mà máy chiếu yêu cầu: điền vào chỗ bị nhòe các chữ số thích hợp để được số n có số dư nhỏ nhất khi chia cho m và đưa ra số dư nhỏ nhất này. Số n nhận được không bắt đầu bằng các số 0 không có nghĩa”.

Dữ liệu: Vào từ file văn bản MINRMD.INP gồm một dòng chứa 2 số nguyên n và m , các chữ số bị nhòe được thay bằng ký tự ‘?’ , n có không quá 14 chữ số, $1 \leq m \leq 10^9$.

Kết quả: Đưa ra file văn bản MINRMD.OUT một số nguyên – số dư nhỏ nhất tìm được.

Ví dụ:

MINRMD.INP	MINRMD.OUT
?1? 730	80

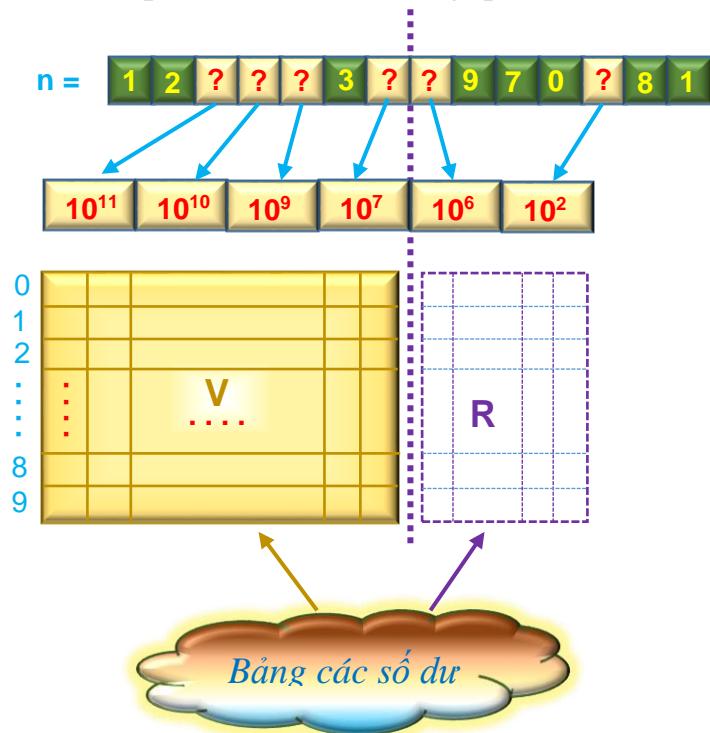


VU05 IO20161126 H

Giải thuật: Kỹ thuật bảng phương án

Nhận xét:

- ✚ Số n ban đầu lưu trữ dưới dạng xâu, gọi k là độ dài của n ,
- ✚ Cần lưu ý các trường hợp đặc biệt:
 - ✳ $k = 1$,
 - ✳ Ký tự $n_0 = '?'$, ký tự này chỉ có thể thay thế bằng một chữ số trong phạm vi từ 1 đến 9,
- ✚ Nếu thay tất cả các ‘?’ bằng chữ số 0 ta có số t_0 ,
- ✚ Nếu thay dấu ‘?’ ở vị trí i trong xâu bằng chữ số d , giá trị số sẽ tăng thêm một lượng là $c = d \times 10^{k-i-1}$,
- ✚ Ta có thể lập bảng lưu các giá trị $c \% m$ có thể có và chọn giá trị nhỏ nhất, nhưng điều này là không thể vì số lượng các giá trị cần lưu trữ hoặc xét có thể đạt tới 10^{14} – vượt quá khả năng lưu trữ và thời gian cho phép xử lý,
- ✚ Để giảm thời gian xử lý và có đủ bộ nhớ lưu trữ dữ liệu trung gian ta chia xâu n thành 2 phần, mỗi phần có độ dài không quá 7,



Các số dư ở bảng V được sắp xếp theo thứ tự tăng dần,

Với mỗi số dư r_i (ở bảng bên phải) và v_j (ở bảng bên trái) ta có $r_i + v_j < 2 \times m$, vì vậy $r_i + v_j$ đạt min ở v_0 hoặc ở v_k thỏa mãn điều kiện $r_i + v_k \geq m$ và $1_i + v_{k-1} < m$.

Tổ chức dữ liệu:

- Mảng `int64_t p10[14]` – $p10_i = 10^i$,
- Mảng động `vector <int64_t> x` – lưu $p10_i$ tương ứng với các ký tự ‘?’,
- Mảng động `vector <int> v` – lưu bảng số dư v .

Xử lý:

Tính mảng **X**:

```
for (int i = 0; i < k; ++i)
    if (n[i] == '?') x.push_back(p10[k - i - 1]);
        else t0 += p10[k - i - 1] * (n[i] - '0');
```

Tính mảng **V**:

```
vector < int > v;  k=x.size();
int t1 = k / 2;
for (int mask = 0; mask < p10[t1]; ++mask)
{
    int64_t csum = t0;
    int cmask = mask;
    if (n[0] == '?' && mask % 10 == 0) continue;
    for (int i = 0; i < t1; ++i)
    {
        csum += x[i] * (cmask % 10);
        cmask /= 10;
    }
    v.push_back(csum % m);
}
sort(v.begin(), v.end());
```



Tìm số dư nhỏ nhất:

```
int ans = m;
for (int mask = 0; mask < p10[k-t1]; ++mask)
{
    int64_t csum = 0;
    int cmask = mask;
    for (int i = t1; i < k; ++i)
    {
        csum += x[i] * (cmask % 10);
        cmask /= 10;
    }
    int y = csum % m;
    auto it = lower_bound(v.begin(), v.end(), m - y);
    ans = min(ans, (y + (it == v.end() ? v[0] : *it)) % m);
}
```

Cập nhật kết quả
với mỗi r_i

Độ phức tạp của giải thuật: $O(nlnn)$.

Chương trình:

```
#include <bits/stdc++.h>
#define NAME "minrmd."
#define sz(x) ((int)((x).size()))
using namespace std;
ifstream fi (NAME"inp");
ofstream fo (NAME"out");
const int N = 14;
string n;
int m,k;
vector<int64_t> x;
int64_t t0 = 0;
int64_t p10[N];

int main()
{
    p10[0] = 1;
    for (int i = 1; i < N; ++i)
        p10[i] = p10[i-1]*10;
    fi >> n >> m; k=n.size();
    if (n == "?") {fo << "0\n";return 0;}
    for (int i = 0; i < k; ++i)
        if (n[i] == '?')
            x.push_back(p10[k - i - 1]);
        else
            t0 += p10[k - i - 1]*(n[i]-'0');
    vector<int> v;k=x.size();
    int t1 = k / 2;
    for (int mask = 0; mask < p10[t1]; ++mask)
    {
        int64_t csum = t0;
        int cmask = mask;
        if (n[0] == '?' && mask % 10 == 0)
            continue;
        for (int i = 0; i < t1; ++i)
        {
            csum += x[i] * (cmask % 10);
            cmask /= 10;
        }
        v.push_back(csum % m);
    }
    sort(v.begin(), v.end());
    int ans = m;
    for (int mask = 0; mask < p10[k-t1]; ++mask)
    {
        int64_t csum = 0;
        int cmask = mask;
        for (int i = t1; i < k; ++i)
        {
            csum += x[i] * (cmask % 10);
            cmask /= 10;
        }
        int y = csum % m;
        auto it = lower_bound(v.begin(), v.end(), m - y);
        ans = min(ans, (y + (it == v.end() ? v[0] : *it)) % m);
    }
}
```

```

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

```

VU06. CHIA ĐÔI

chương trình: BISECT.CPP

Tên



Nhóm tìm kiếm kho báu khảo sát một chiếc thuyền cổ bị đắm và thu được 3 thỏi vàng trọng lượng tương ứng là **a**, **b** và **c**. Tổng trọng lượng 3 thỏi là một số chẵn. Theo quy định hiện hành, một nửa số hiện vật tìm được là của nhà nước, nửa kia – thuộc người tìm được.

Người ta cố gắng tìm cách chia đều vàng tìm được thành 2 phần có trọng lượng bằng nhau mà không cần cắt, nếu không thể được – sẽ cắt một thỏi thành 2 phần để chia.

Nếu có thể chia đều mà không cần cắt thì đưa ra số 0, trong trường hợp ngược lại – chỉ ra thỏi vàng cần cắt (1, 2 hoặc 3) và trọng lượng mỗi phần sau khi cắt.

Dữ liệu: Vào từ file văn bản BISECT.INP gồm một dòng chứa 3 số nguyên **a**, **b** và **c** ($1 \leq a, b, c \leq 10^8$).

Kết quả: Đưa ra file văn bản BISECT.OUT, dòng thứ nhất chứa số 0 hoặc số nguyên cho biết thỏi vàng cần cắt, nếu cần cắt thì dòng thứ 2 chứa hai số nguyên xác định trọng lượng mỗi phần sau khi cắt. Đưa ra phương án tùy chọn nếu có nhiều cách chia.

Ví dụ:

BISECT.INP
2 3 3

BISECT.OUT
2
2 1



VU06 vkop20161211 A

Giải thuật: Phân tích tình huống.

Nhận xét:

Do tổng các số là chẵn nên bài toán luôn có nghiệm,

Đặt **sum** = (**a** + **b** + **c**) / 2, nếu **sum** bằng một trong 3 số đã cho thì không cần phân chia thỏi vàng,

Xét trường hợp cần phân chia:

Cần phân chia thỏi 1 khi **a** > **sum**,

Phân chia thỏi 2 khi **b** > **sum - a**,

Trường hợp còn lại: phân chia thỏi 3.

Chương trình:

```
#include "bits/stdc++.h"
#define NAME "bisect."
using namespace std;
ifstream fi (NAME"inp");
ofstream fo (NAME"out");
int a,b,c,sum;

int main()
{
    fi>>a>>b>>c;
    sum = (a+b+c)/2;

    if (a == sum || b == sum || c == sum)
    {
        fo << 0 << "\n"; return 0;
    }

    if (a > sum)
    {
        fo << 1 << "\n" << sum << " " << a - sum << "\n";
        return 0;
    }
    sum -= a;
    if (b > sum)
    {
        fo << 2 << "\n" << sum << " " << b - sum << "\n";
        return 0;
    }
    sum -= b;
    fo << 3 << "\n" << sum << " " << c - sum << "\n";
}
```



VU07. TUYẾN Ô TÔ BUÝT NHANH

Tên chương trình: BUS.CPP

Tuyến ô tô buýt nhanh có n chỗ dừng cho khách lên xuống. Các chỗ dừng đánh số từ 1 đến n . Xe chạy từ chỗ dừng 1 đến chỗ dừng n và cần 1 phút để đi từ một chỗ dừng tới chỗ dừng tiếp theo. Thời gian dừng đón khách là không đáng kể. Trên xe có 1 hàng ghế ngồi, ghế được đánh số từ 1 đến m , ghế 1 gần cửa ra nhất và ghế m – xa cửa ra nhất. Mỗi ghế chỉ để một người ngồi. Bên cạnh mỗi ghế có treo một tay cầm để một hành khách có thể đứng. Khi một hành khách lên xe, nếu còn ghế trống họ sẽ tới ghế trống gần cửa ra nhất để ngồi. Nếu không còn chỗ ngồi họ sẽ tìm đến chỗ đứng trống gần cửa ra nhất và sẽ đứng ở đó cho tới khi ra không phụ thuộc vào việc xuất hiện các ghế ngồi hay chỗ đứng trống. Khi lên xe, nếu không còn chỗ (cả ngồi và đứng) hành khách sẽ ra khỏi xe chờ chuyến sau. Ở mỗi chỗ đỗ, các hành khách cần ra sẽ ra hết sau đó mới nhận hành khách lên xe.

Giáo sư Braun sử dụng ô tô buýt nhanh để đi làm. Hàng sáng ông là người đầu tiên lên xe ở chỗ đỗ 1 và sẽ xuống xe ở chỗ đỗ n . Là người đầu tiên lên xe ông có thể chọn chỗ ngồi bất kỳ. Ông muốn chọn một chỗ sao cho tổng thời gian bên cạnh mình có người đứng là ít nhất để đỡ bị phân tâm trong suy nghĩ về công việc của mình. Là một nhà khoa học ông quan sát và nhớ rõ ai sẽ lên xe lúc nào và xuống xe ở đâu.

Hãy xác định tổng thời gian nhỏ nhất có người đứng bên cạnh và chỗ ngồi mà giáo sư Braun cần chọn. Nếu có nhiều cách chọn khác nhau thì đưa ra chỗ ngồi gần cửa ra vào nhất.

Dữ liệu: Vào từ file văn bản BUS.INP:

- + Dòng đầu tiên chứa 3 số nguyên n , m và k , trong đó k – tổng số hành khách chờ lên xe ($2 \leq n \leq 10^9$, $1 \leq m \leq 2 \times 10^5$, $0 \leq k \leq 2 \times 10^5$),
- + Dòng thứ i trong k dòng tiếp theo chứa 2 số nguyên a_i và b_i xác định nơi vào và ra của một hành khách. Nếu một nơi có nhiều người chờ vào thì hành khách lên xe theo trình tự nêu trong input ($1 \leq a_i < b_i \leq n$).

Kết quả: Đưa ra file văn bản BUS.OUT trên một dòng 2 số nguyên – tổng thời gian nhỏ nhất có người đứng bên cạnh và chỗ ngồi mà giáo sư Braun cần chọn.

Ví dụ:

BUS.INP
10 2 3
1 10
3 9
7 10

BUS.OUT
3 2



Giải thuật: Mô phỏng tiến trình.

Nhận xét:

- ✚ Việc một hành khách mới vào phải đứng hay được ngồi chỉ phụ thuộc vào tập các vị trí trống và không phụ thuộc vào nơi giáo sư Braun ngồi,
- ✚ Một hành khách vào phải đứng khi không còn chỗ ngồi trống và vị trí đứng không phụ thuộc vào nơi giáo sư Braun ngồi,
- ✚ Như vậy có thể đặt giáo sư Braun vào chỗ ngồi bất kỳ, mô phỏng toàn bộ quá trình hành khách chọn chỗ và tìm ra nơi thỏa mãn điều kiện của bài toán.

Tổ chức dữ liệu:

- ⌚ Mảng động **vector<pii> inp** – lưu dữ liệu vào,
- ⌚ Mảng **vector<tuple<int, int, int>> s** – lưu sự kiện cần xử lý,
- ⌚ Mảng **vector<pii> where(k)** – quản lý vị trí hành khách,
- ⌚ Mảng **vector<int> here(m)** – quản lý trạng thái các vị trí đứng,
- ⌚ Tập **set<int> place[2]** – quản lý trạng thái tự do của các chỗ.

Xử lý:

Ghi nhận thông tin, sự kiện và chuẩn bị dữ liệu:

```

    fi>>n>>m>>k;
    vector<pii> inp;
    vector<tuple<int, int, int>> s;
    for (int i = 0; i < k; ++i)
    {
        int a, b;
        fi>>a>>b;
        inp.push_back({a, b});
        s.push_back(make_tuple(a, 1, i));
        s.push_back(make_tuple(b, 0, i));
    }

    vector<pii> where(k);
    vector<int> here(m);
    set<int> place[2];
    for (int i = 0; i < m - 1; ++i)
    {
        place[0].insert(i);
        place[1].insert(i);
    }
    place[1].insert(m - 1);
    sort(bend(s));

```

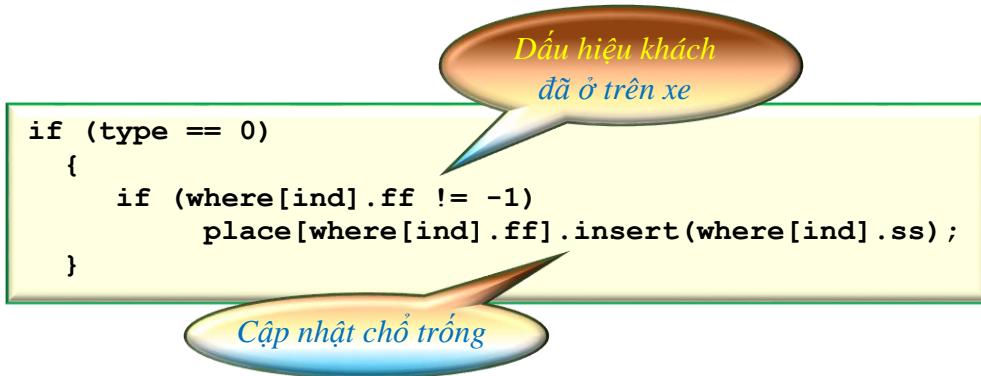
Ghi nhận sự kiện

Xin cấp phát bộ nhớ

*Để dành chỗ ngồi m
cho g.s. Braun*

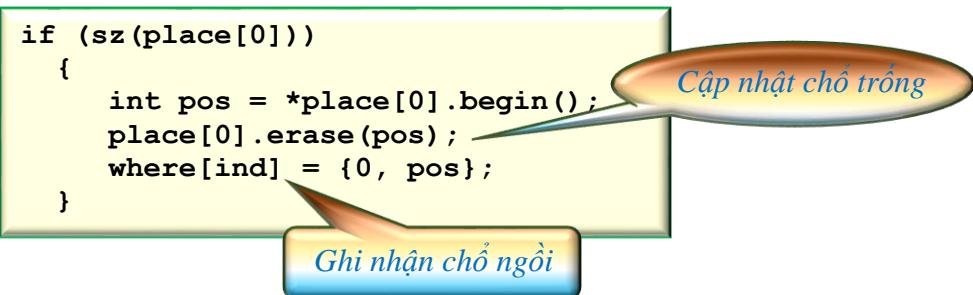
Xử lý sự kiện:

Sự kiện khách ra:

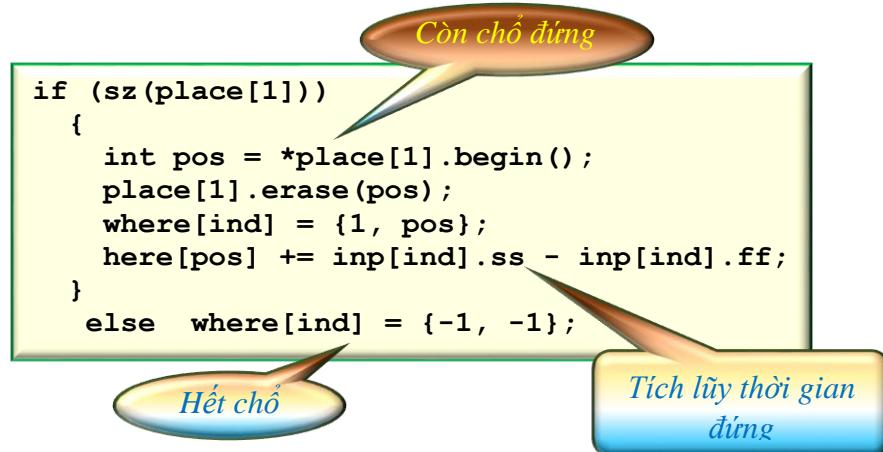


Sự kiện khách lên xe:

Trường hợp còn chỗ ngồi trống:



Trường hợp hết chỗ ngồi:



Tìm khoảng thời gian nhỏ nhất và vị trí:

```
int pos = 0;
for (int i = 0; i < m; ++i)
    if (here[i] < here[pos]) pos = i;
```

Độ phức tạp của giải thuật: $O(k(\ln k + \ln m))$.

Chương trình:

```
#include "bits/stdc++.h"
#define puba push_back
#define ff first
#define ss second
#define bend(_x) begin(_x), end(_x)
#define sz(_x) ((int) (_x).size())
#define NAME "bus."

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

int main()
{
    int n, m, k;
    fi>>n>>m>>k;
    vector<pii> inp;
    vector<tuple<int, int, int>> s;
    for (int i = 0; i < k; ++i)
    {
        int a, b;
        fi>>a>>b;
        inp.push_back({a, b});
        s.push_back(make_tuple(a, 1, i));
        s.push_back(make_tuple(b, 0, i));
    }

    vector<pii> where(k);
    vector<int> here(m);
    set<int> place[2];
    for (int i = 0; i < m - 1; ++i)
    {
        place[0].insert(i);
        place[1].insert(i);
    }
    place[1].insert(m - 1);
    sort(bend(s));

    for (auto act : s)
    {
        int t, type, ind;
        tie(t, type, ind) = act;
        if (type == 0)
        {
            if (where[ind].ff != -1)
                place[where[ind].ff].insert(where[ind].ss);
        } else
        {
            if (sz(place[0]))
            {
                int pos = *place[0].begin();
                place[0].erase(pos);
                where[ind] = {0, pos};
            }
        }
    }
}
```

```

    else
        if (sz(place[1]))
        {
            int pos = *place[1].begin();
            place[1].erase(pos);
            where[ind] = {1, pos};
            here[pos] += inp[ind].ss - inp[ind].ff;
        } else where[ind] = {-1, -1};
    }
}

int pos = 0;
for (int i = 0; i < m; ++i)
    if (here[i] < here[pos]) pos = i;

fo << here[pos] << " " << pos + 1 << "\n";

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

```



Cá hồi và cá tầm là các loại cá vùng ôn đới, có giá trị kinh tế cao. Ngày nay đã có công nghệ nuôi dưỡng chúng ở các nước nhiệt đới. Viện Thủy sản có 2 trung tâm, một để nuôi và nhân giống cá hồi, trung tâm khác – nuôi và nhân giống cá tầm. Trường Dự án phát triển cá đặc sản phải chạy đi chạy lại như con thoi giữa 2 trung tâm, nhất là khi có những đợt lạnh về, lúc thích hợp nhất để kích thích cá hồi để trứng và nhân giống. Mỗi lần lên trại cá hồi ông ở đó đúng k ngày và quay trở lại trại cá tầm ngay cuối ngày thứ k . Cũng có thể, nếu cần thiết ông quay lại trại cá hồi ngay ngày hôm sau! Đợt lạnh này kéo dài n ngày. Khi ở trại cá hồi, vào một số ngày lạnh ông cá mẹ lên đẻ đặc, chụp ảnh và tiêm thuốc kích thích cho cá để trứng. Ngày chụp được ghi tự động trên ảnh. Quá bận rộn nên ông cũng không nhớ rõ mình đã lên trại cá hồi bao nhiêu đợt.

Sau đợt lạnh, những ngày bận rộn với cá hồi qua đi. Bây giờ trọng tâm chú ý chuyển sang cá tầm. Để lên kế hoạch cho những ngày tiếp theo ông muốn biết tối đa mình đã có bao nhiêu ngày trong đợt lạnh mình đã làm việc ở trại cá tầm.

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

- ✚ Dòng đầu tiên chứa 3 số nguyên n , k và m , trong đó m – số lượng các ngày được ghi lại trên ảnh ($1 \leq k \leq n \leq 10^9$, $1 \leq m \leq 2 \times 10^5$, $m \leq n$),
- ✚ Dòng thứ 2 chứa m số nguyên d_1, d_2, \dots, d_m ($1 \leq d_i \leq n$, $d_i \neq d_j$ với $i \neq j$, $i, j = 1 \dots m$).

Kết quả: Đưa ra file văn bản SALMON.OUT một số nguyên – số ngày tối đa làm việc ở trại cá tầm.

Ví dụ:

SALMON.INP
7 4 3
4 3 5

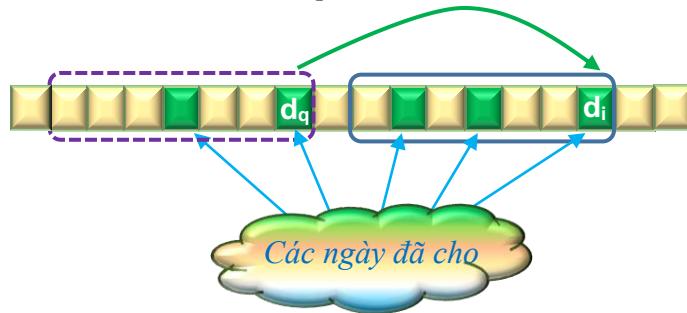
SALMON.OUT
3



Giải thuật: Quy hoạch động.

Nhận xét:

- ✚ Một chuyến đi tới trại cá hồi có thể bắt đầu trước đợt lạnh và cũng có thể kết thúc sau đợt lạnh,
- ✚ Để giảm thiểu số ngày ở trại cá hồi mỗi chuyến đi phải *kết thúc bằng ngày cuối cùng ở trại, trừ chuyến cuối cùng* có thể kết thúc ở một ngày sau đợt lạnh,
- ✚ Sắp xếp các ngày đã biết ở trại cá hồi theo thứ tự tăng dần,
- ✚ Gọi dp_i là số ngày lạnh nhiều nhất ở trại cá tầm nếu *chuyến đi lên trại cá hồi kết thúc ở ngày i* ,
- ✚ Theo sơ đồ giải thuật quy hoạch động, các giá trị dp_i cần tính theo trình tự từ cuối về đầu, tức là với *i thay đổi từ m lùi về 1*,
- ✚ Chuyến đi kết thúc ở ngày i sẽ bao gồm các d_j thỏa mãn điều kiện $d_i - d_j + 1 \leq k$,
- ✚ Như vậy mỗi giá trị dp_i sẽ có quan hệ tới giá trị dp_q , trong đó $q < i$ và là chỉ số lớn nhất thỏa mãn điều kiện $d_q - d_i + 1 > k$ nếu tồn tại q ,



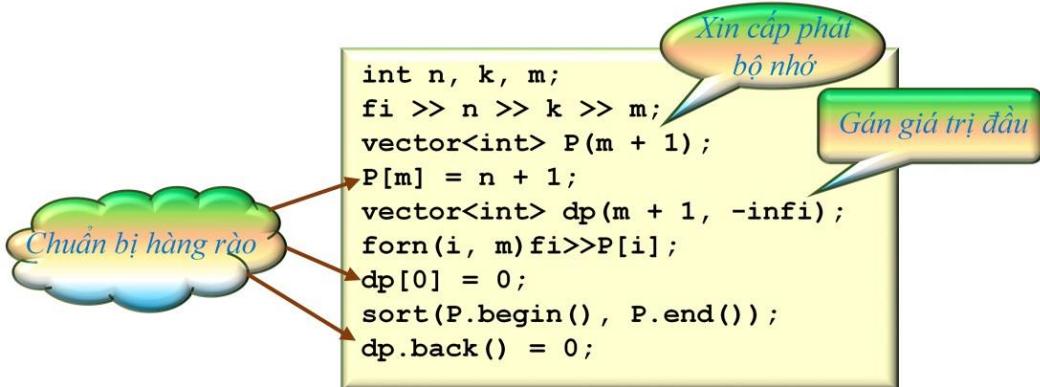
- ✚ Chỉ số q có thể xác định bằng phương pháp tìm kiếm nhị phân,
- ✚ Kết quả cần tìm là giá trị *max* trong dp .

Tổ chức dữ liệu:

- ▀ Mảng `vector<int> p(m+1)` – lưu các ngày được ghi nhận,
- ▀ Mảng `vector<int> dp` – lưu giá trị tối ưu theo sơ đồ quy hoạch động.

Xử lý:

Nhập dữ liệu và chuẩn bị:



Tính dp_i cho các đợt đi có ngày đầu nằm trong đợt lạnh:

The diagram shows a yellow rectangular box containing C++ code. A speech bubble on the right says "Tính q tương ứng với i" (Calculate q corresponding to i). The code uses `upper_bound` to find the index q such that $P[q] \leq e < P[q+1]$.

```
for(int i = m - 1; i >= 0; i--)
{
    int e = P[i] + k - 1;
    int q = upper_bound(P.begin(), P.end()-1, e) - P.begin();
    dp[i] = max(dp[i], dp[q]+P[q]-min(e, n) - 1);
}
```

Tính các đợt đi có ngày bắt đầu ngoài đợt lạnh:

The diagram shows a yellow rectangular box containing C++ code. The code initializes $dp[0]$ and res , then iterates through the array P . If $P[i] - k \leq 0$, it updates res with the maximum value between res and $dp[i+1] + P[i+1] - P[i] - 1$. Otherwise, it breaks out of the loop.

```
dp[0] += P[0] - 1;
int res = dp[0];
forn(i, m)
    if(P[i] - k <= 0)
        res = max(res, dp[i+1] + P[i+1] - P[i] - 1);
    else break;
```

Độ phức tạp của giải thuật: $O(mlnm)$.

Chương trình:

```
#include <bits/stdc++.h>
#define NAME "salmon."
#define forn(i, n) for(int i = 0; i < (int)n; i++)
#define fornn(i, q, n) for(int64_t i = (int64_t)q; i < (int64_t)n; i++)
#define times clock() * 1.0 / CLOCKS_PER_SEC
//#pragma comment(linker, "/STACK:667772160")

using namespace std;

ifstream fi (NAME"inp");
ofstream fo (NAME"out");
const int infi = 1e9 + 7;

int main()
{
    int n, k, m;
    fi >> n >> k >> m;
    vector<int> P(m + 1);
    P[m] = n + 1;
    vector<int> dp(m + 1, -infi);
    forn(i, m) fi >> P[i];
    dp[0] = 0;
    sort(P.begin(), P.end());
    dp.back() = 0;
    for(int i = m - 1; i >= 0; i--)
    {
        int e = P[i] + k - 1;
        int w = upper_bound(P.begin(), P.end() - 1, e) - P.begin();
        dp[i] = max(dp[i], dp[w] + P[w] - min(e, n) - 1);
    }
    dp[0] += P[0] - 1;
    int res = dp[0];
    forn(i, m)
        if(P[i] - k <= 0) res = max(res, dp[i + 1] + P[i + 1] - P[i] - 1);
    else break;
    fo << res;
    fo << "\nTime: " << times << " sec";
}
```



Một thiết bị nhận dạng mới được lắp trên camera bay (*Drone*). Để thử nghiệm khả năng nhận dạng người ta tạo một trường thử nghiệm hình chữ nhật kích thước $n \times m$ ô vuông (n dòng và m cột), mỗi ô được sơn một trong số 26 màu, mỗi màu được ký hiệu bằng một chữ cái la tinh thường. Các chữ cái khác nhau tương ứng với những màu khác nhau. Camera treo lơ lửng trên ô tọa độ (x, y) ở độ cao đủ bao quát toàn trường thử nghiệm. Nhiệm vụ của máy bay là phải báo về tọa độ 2 ô khác màu nhau, một ô ở cùng hàng và ô khác – cùng cột với ô tọa độ (x, y) phía dưới và khoảng cách 2 ô được chọn phải là nhỏ nhất. Nếu 2 ô được chọn có tọa độ là $(r1, c1)$ và $(r2, c2)$ thì khoảng cách được xác định là $|r1-r2| + |c1-c2|$. Các ô cần chọn không được trùng với ô (x, y) . Nếu không tồn tại cặp ô thỏa mãn điều kiện đã nêu thì báo về giá trị -1.



Để có đủ số liệu thông kê đáng tin cậy, người ta di chuyển camera sang ô khác và phát tín hiệu yêu cầu chuyển thông tin về. Có tất cả q lần yêu cầu cung cấp thông tin, lần thứ i camera treo trên ô tọa độ (x_i, y_i) , $i = 1 \dots q$.

Hãy xác định thông tin được chuyển về. Với mỗi lần, nếu có nhiều cặp ô cùng thỏa mãn điều kiện đã nêu thì váo về tọa độ cặp ô tùy chọn.

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

- ✚ Dòng đầu tiên chứa 2 số nguyên n và m ($2 \leq n, m \leq 5 \times 10^5$, $n \times m \leq 10^6$),
- ✚ Mỗi dòng trong n dòng tiếp theo chứa xâu ký tự la tinh thường độ dài m mô tả một dòng của trường thử nghiệm,
- ✚ Dòng thứ $n+2$ chứa số nguyên q ($1 \leq q \leq 2 \times 10^5$),
- ✚ Dòng thứ i trong n dòng tiếp theo chứa 2 số nguyên x_i, y_i ($1 \leq x_i \leq n, 1 \leq y_i \leq m$).

Kết quả: Đưa ra file văn bản DRONE.OUT q dòng, mỗi dòng chứa thông tin chuyển về theo yêu cầu cung cấp, thông tin trên mỗi dòng là số -1 nếu không tồn tại cặp ô thỏa mãn điều kiện cần tìm hoặc 4 số nguyên $r1, c1, r2, c2$ xác định tọa độ ô thứ nhất và ô thứ 2 trong cặp ô tìm được.

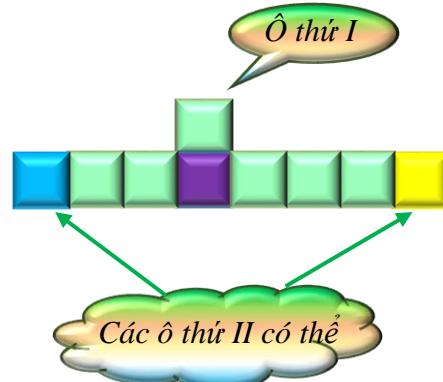
Ví dụ:

DRONE.INP	DRONE.OUT
3 4	-1
abbb	1 1 2 4
baab	3 2 2 1
babb	2 4 3 2
4	
1 1	
1 4	
3 1	
3 4	

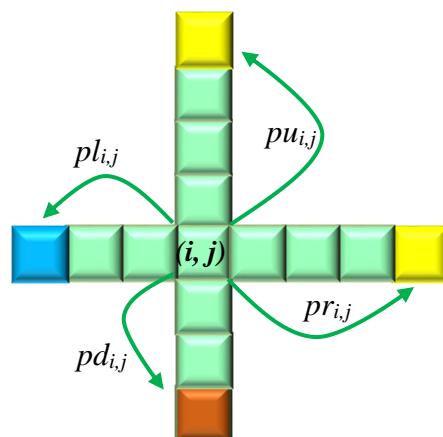
Giải thuật: Kỹ thuật tổ chức dữ liệu và nhận dạng.

Nhận xét:

- ✚ Bằng phương pháp phản chứng có thể dễ dàng chứng minh một trong 2 ô cần chọn phải kề cạnh với ô (x, y) ,
- ✚ Với ô (x, y) đang xét có tối đa 4 khả năng lựa chọn ô đầu tiên,
- ✚ Duyệt mọi khả năng có thể của ô đầu tiên, xác định ô thứ 2 tương ứng có thể chọn để có khoảng cách tới ô đã chọn là nhỏ nhất,
- ✚ So sánh các cách chọn cặp ô là giữ lại kết quả tối ưu,
- ✚ Gọi d_{\min} là khoảng cách nhỏ nhất của cặp ô được chọn, ta thấy $d_{\min} \geq 2$, vì vậy kết quả duyệt có thể kết thúc khi có $d_{\min} = 2$,
- ✚ Sau khi chọn ô thứ nhất, nếu ở hướng vuông góc các ô kề cạnh với (x, y) đều cùng màu với ô thứ nhất thì trên hướng vuông góc phải duyệt sang phải và trái, tìm ô gấp đầu tiên khác màu để xác định ô thứ 2 trong cặp,



- ✚ Như vậy, với mỗi ô (i, j) trong bảng đã cho cần xác định vị trí ô gần nhất khác màu ở 4 hướng:



- ✚ Có thể tồn tại các cách chọn cặp ô khác nhau cùng cho khoảng cách nhỏ nhất,
- ✚ Cần tổ chức lưu trữ động vì một trong 2 giá trị n hoặc m có thể rất lớn.

Tổ chức dữ liệu:

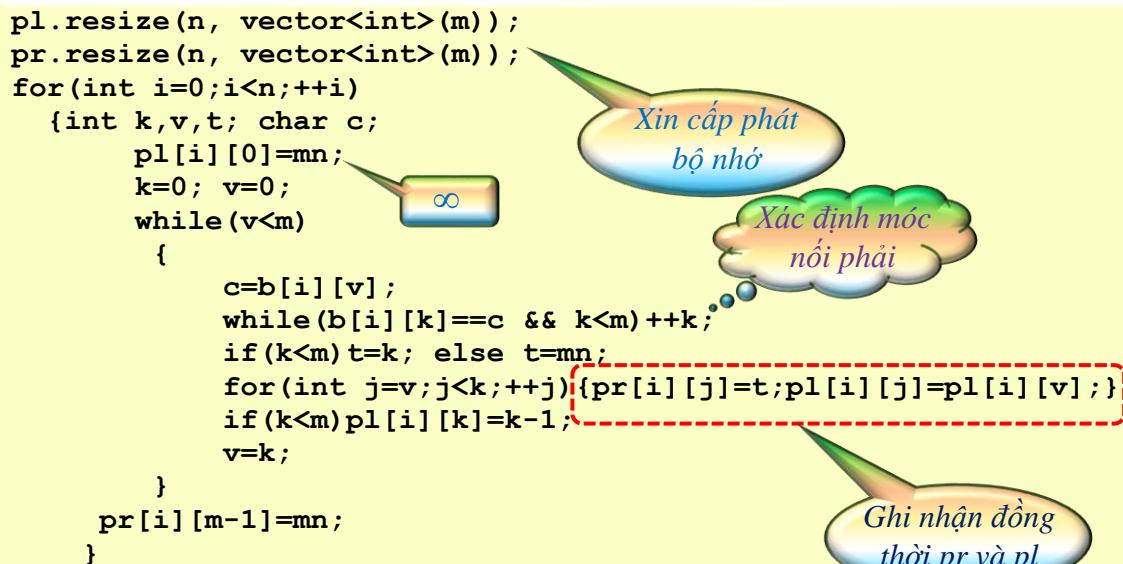
- Mảng `vector<string>` **b** – lưu trữ bảng dữ liệu của trường thử nghiệm,
- Các mảng `vector<vector<int>>` **pl, pr, pu, pd** – lưu trữ mốc nối tới ô khác màu gần nhất.

Xử lý:

Nhập bảng mô tả trường thử nghiệm,

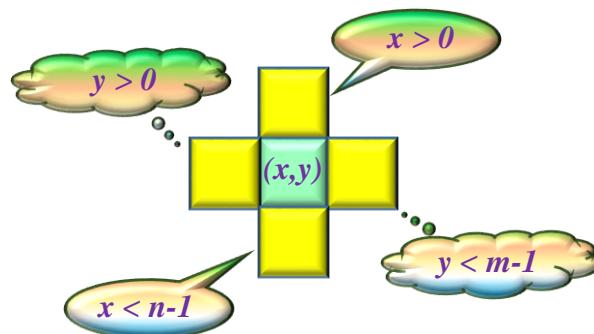
Xác định các mốc nối trái/phải tới ô khác màu gần nhất:

```
pl.resize(n, vector<int>(m));
pr.resize(n, vector<int>(m));
for(int i=0;i<n;++i)
    {int k,v,t; char c;
     pl[i][0]=mn;
     k=0; v=0;
     while(v<m)
        {
            c=b[i][v];
            while(b[i][k]==c && k<m)++k;
            if(k<m) t=k; else t=mn;
            for(int j=v;j<k;++j){pr[i][j]=t;pl[i][j]=pl[i][v];}
            if(k<m)pl[i][k]=k-1;
            v=k;
        }
     pr[i][m-1]=mn;
    }
```

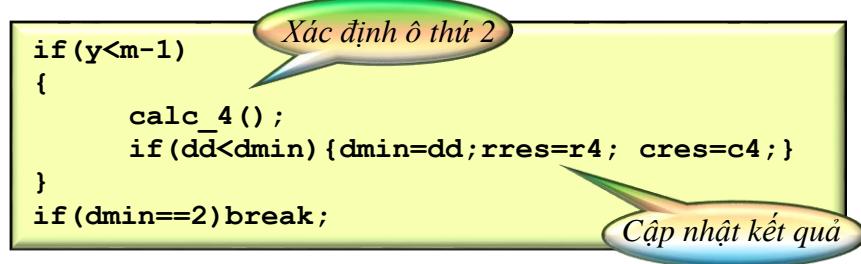


Xác định các mốc nối trên/dưới tới ô khác màu gần nhất: tương tự như trên,

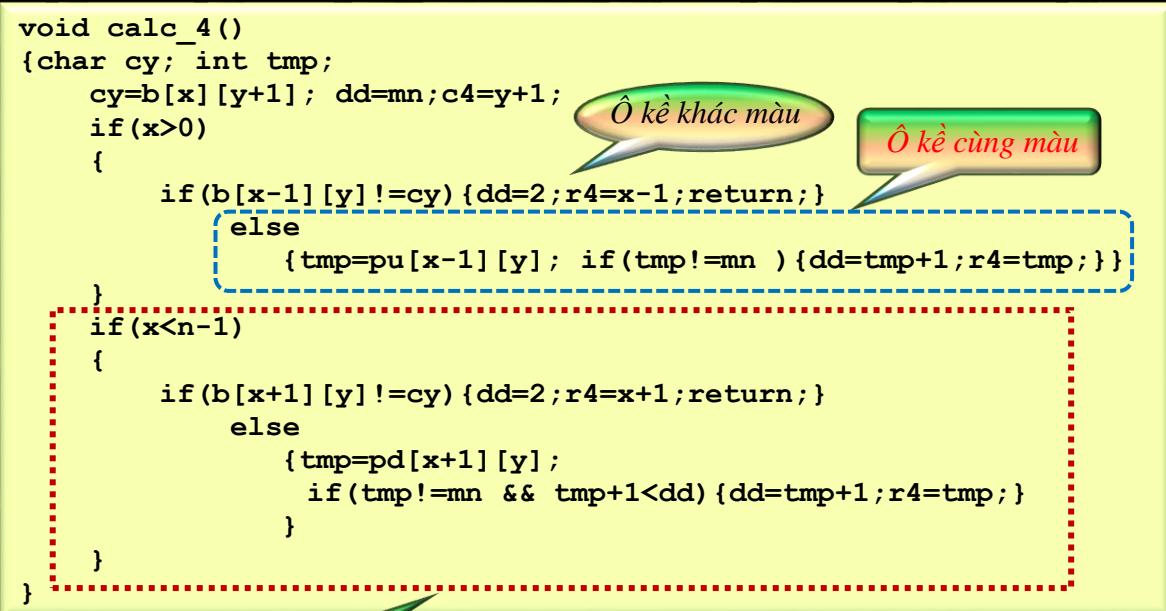
Điều kiện tồn tại ô thứ nhất trong cặp (*chỉ số bắt đầu từ 0*):



Xử lý trường hợp $y < m-1$:



Chọn ô thứ 2:



Các trường hợp còn lại: *Xử lý tương tự.*

Độ phức tạp của giải thuật: $O(m \times n) + O(q)$.

Chương trình:

```
#include <bits/stdc++.h>
#define NAME "drone."
#define times clock() * 1.0 / CLOCKS_PER_SEC
using namespace std;
ifstream fi (NAME"inp");
ofstream fo (NAME"out");
vector<string>b;
vector<vector<int> >pl,pr,pu,pd;
string s;
int n,m,q,x,y,dmin,mn,cres,rres;
int dd,r1,r2,r3,r4,c1,c2,c3,c4;

void calc_4 ();
void calc_3 ();
void calc_2 ();
void calc_1 ();

int main()
{
    fi>>n>>m;
    mn=m+n;

    for(int i=0;i<n;++i)
    {
        fi>>s; b.push_back(s);
    }
    pl.resize(n, vector<int>(m));
    pr.resize(n, vector<int>(m));
    for(int i=0;i<n;++i)
    {int k,v,t; char c;
        pl[i][0]=mn;
        k=0; v=0;
        while(v<m)
        {
            c=b[i][v];
            while(b[i][k]==c && k<m) ++k;
            if(k<m) t=k; else t=mn;
            for(int j=v;j<k;++j) {pr[i][j]=t; pl[i][j]=pl[i][v];}
            if(k<m) pl[i][k]=k-1;
            v=k;
        }
        pr[i][m-1]=mn;
    }

    pu.resize(n, vector<int>(m));
    pd.resize(n, vector<int>(m));

    for(int j=0;j<m;++j)
    {int k,z,t; char c;
        vector<int>qu(n),qd(n);
        qu[0]=mn;
        k=0; z=0;
        while(z<n)
        {
            c=b[z][j];
            while( (k<n) && (c==b[k][j]) ) ++k;
            if(k<n) t=k; else t=mn;
```

```

        for (int i=z; i<k; ++i) {qd[i]=t; qu[i]=qu[z];}
        if (k<n) qu[k]=k-1;
        z=k;
    } qd[n-1]=mn;
    for (int i=0; i<n; ++i) {pu[i][j]=qu[i]; pd[i][j]=qd[i];}
}

fi>>q;
for (int i=0; i<q; ++i)
{
    fi>>x>>y; --x; --y;
    dmin=mn;
    while (1)
    {
        if (y<m-1) {calc_4(); if (dd<dmin) {dmin=dd; rres=r4; cres=c4;} }
        if (dmin==2) break;
        if (y>0) {calc_2(); if (dd<dmin) {dmin=dd; rres=r2; cres=c2;} }
        if (dmin==2) break;
        if (x<n-1) {calc_3(); if (dd<dmin) {dmin=dd; rres=r3; cres=c3;} }
        if (dmin==2) break;
        if (x>0) {calc_1(); if (dd<dmin) {dmin=dd; rres=r1; cres=c1;} }
        break;
    }

    if (dmin==mn) fo<<"-1\n";
    else fo<<x+1<<' '<<cres+1<<' '<<rres+1<<' '<<y+1<<' \n';
}
fo<<"\nTime: "<<times<<" sec";
}

void calc_4()
{char cy; int tmp;
cy=b[x][y+1]; dd=mn; c4=y+1;
if (x>0)
{
    if (b[x-1][y]!=cy) {dd=2; r4=x-1; return;}
    else {tmp=pu[x-1][y]; if (tmp!=mn) {dd=tmp+1; r4=tmp;}}
}
if (x<n-1)
{
    if (b[x+1][y]!=cy) {dd=2; r4=x+1; return;}
    else {tmp=pd[x+1][y]; if (tmp!=mn && tmp+1<dd) {dd=tmp+1; r4=tmp;}}
}
}

void calc_3()
{char cy; int tmp;
cy=b[x+1][y]; dd=mn; r3=x+1;
if (y>0)
{
    if (b[x][y-1]!=cy) {dd=2; c3=y-1; return;}
    else {tmp=pl[x][y-1]; if (tmp!=mn && tmp+1<dd) {dd=tmp+1; c3=tmp;}}
}
if (y<m-1)
{
    if (b[x][y+1]!=cy) {dd=2; c3=y+1; return;}
    else {tmp=pr[x][y+1]; if (tmp!=mn && tmp+1<dd) {dd=tmp+1; c3=tmp;}}
}
}

void calc_2()

```

```

{char cy; int tmp;
cy=b[x][y-1]; dd=mn;c2=y-1;
if(x>0)
{
    if(b[x-1][y]!=cy) {dd=2;r2=x-1;return; }
    else {tmp=pu[x-1][y]; if(tmp!=mn ) {dd=tmp+1;r2=tmp; } }
}
if(x<n-1)
{
    if(b[x+1][y]!=cy) {dd=2;r2=x+1;return; }
    else {tmp=pd[x+1][y]; if(tmp!=mn && tmp+1<dd) {dd=tmp+1;r2=tmp; } }
}
}

void calc_1()
{char cy; int tmp;
cy=b[x-1][y]; dd=mn;r1=x-1;
if(y>0)
{
    if(b[x][y-1]!=cy) {dd=2;c1=y-1;return; }
    else{tmp=pl[x][y-1]; if(tmp!=mn && tmp+1<dd) {dd=tmp+1;c1=tmp; } }
}
if(y<m-1)
{
    if(b[x][y+1]!=cy) {dd=2;c1=y+1;return; }
    else{tmp=pl[x][y+1]; if(tmp!=mn && tmp+1<dd) {dd=tmp+1;c1=tmp; } }
}
}
}

```



VU10. SỐ NHỎ NHẤT

Tên chương trình: MINNUM.CPP

Cho 2 số nguyên dương a và n . Hãy tìm số nguyên không âm b nhỏ nhất thỏa mãn điều kiện $a \wedge b$ chia hết cho n , trong đó \wedge là phép tính **xor**. Khi thực hiện phép tính \wedge có thể bổ sung thêm các số 0 bên trái của toán hạng để a và b có cùng số lượng bít như nhau. Ví dụ, $a = 25$, $b = 100$, khi đó kết quả $a \wedge b$ sẽ là 125.

$$\begin{aligned}a &= 25_{10} = 0011001 \\b &= 100_{10} = 1100100 \\a \wedge b &= 1111101 = 125_{10}.\end{aligned}$$

Dữ liệu: Vào từ file văn bản MINNUM.INP

- ⊕ Dòng đầu tiên chứa một số nguyên t ($1 \leq t \leq 10^4$) – số lượng tests,
- ⊕ Mỗi dòng trong t dòng tiếp theo chứa 2 số nguyên a và n ($1 \leq a, n \leq 10^{18}$).

Kết quả: Đưa ra file văn bản MINNUM.OUT các số nguyên tìm được, kết quả mỗi test đưa ra trên một dòng.

Ví dụ:

MINNUM.INP
3
10 5
3 2
98 100

MINNUM.OUT
0
1
6



VU10 vkop20161211 K

Giải thuật: Số học đơn giản..

Nhận xét:

Với mỗi cặp số a và n bài toán luôn có nghiệm vì $a^a = 0$ – chia hết cho n , hơn thế nữa, số b cần tìm sẽ không vượt quá a ,

Do a^b chia hết cho n nên a^b phải bằng $a - a \% n$ hoặc bằng $a - a \% n + n$. từ đó suy ra

$$b = \min\{a^{(a-a \% n)}, a^{(a-a \% n+n)}\}$$

Độ phức tạp của giải thuật: $O(t)$.

Chương trình:

```
#include <bits/stdc++.h>
#define NAME "minnum."
#define times clock() * 1.0 / CLOCKS_PER_SEC
using namespace std;
ifstream fi (NAME"inp");
ofstream fo (NAME"out");

int64_t a,b,n,t,tmp;

int main()
{
    fi>>t;
    for(int i=0;i<t;++)
    {
        fi>>a>>n;
        tmp=a%n;
        b=min(a^(a-tmp),a^(a-tmp+n));
        fo<<b<<'\\n';
    }
    fo<<"\\nTime: "<<times<<" sec";
}
```



Helene coi việc nghiên cứu kỹ nội dung lý thuyết và làm hết các bài tập như một việc đương nhiên đối với học sinh, chính vì vậy Helene luôn vui vẻ ở vị trí đầu trong bảng thành tích của lớp. Các thầy cô thường phải giao thêm bài tập trên lớp cho Helene trong khi các bạn khác vẫn còn đang đánh vật với những bài tập chung.

Giờ Tin học hôm nay Helene nhận được một bài tập bổ sung khá khó nhằn: "Cho số nguyên dương x , hãy tìm số nguyên $y \geq x$ có không ít hơn 100 ước số và y không vượt quá 1% so với x , tức là $x \leq y \leq 1.01x$ ".

Cũng may, tuy mệt nhoài người nhưng Helene đã kịp hoàn thành xong chương trình trước khi tiếng chuông hết giờ vang lên.

Hãy xác định kết quả mà chương trình Helene đưa ra.

Dữ liệu: Vào từ file văn bản DIVISORS.INP gồm một dòng chứa số nguyên x ($1 \leq x \leq 10^{16}$).

Kết quả: Đưa ra file văn bản DIVISORS.OUT số -1 nếu không tồn tại y thỏa mãn điều kiện đã nêu hoặc một số nguyên y tùy chọn thỏa mãn điều kiện bài toán.

Ví dụ:

DIVISORS.INP
510000

DIVISORS.OUT
510510



Giải thuật: Kỹ thuật đoán nhận giải thuật, tính số lượng ước.

Nhận xét:

- ✚ Nếu t là một số nguyên và số nguyên z có không ít hơn 100 ước thì $t \times z$ cũng có không ít hơn 100 ước,
- ✚ Một số nguyên phải đủ lớn mới có thể có 100 ước,
- ✚ Để dàng xây dựng *chương trình khảo sát*, tìm số lượng ước của một số nguyên z , sử dụng chương trình này để *tìm số nguyên nhỏ nhất có 100 ước*,
- ✚ Phần *chủ yếu của chương trình khảo sát* (hàm tìm số lượng ước) *sẽ được sử dụng* trong chương trình giải bài toán ban đầu,
- ✚ Kết quả khảo sát cho thấy **45360** là số nhỏ nhất có 100 ước,
- ✚ Với các số x nhỏ hơn $45360 \times 100 - y$ được tìm bằng cách duyệt vét cạn, độ phức tạp của giải thuật trong trường hợp này là $O(x/100 \times x^{0.5})$,
- ✚ Nếu $1.01x < 45360 -$ kết quả là **-1**,
- ✚ Nếu $x \geq 4536000$ thì kết quả là **(x/45360+1) × 45360**.

Xử lý:

Trường hợp $x \geq 4536000$:

```
const int64_t P = 45360;

int64_t y = x + x / 100;
if (y / P != x / P) fo<< P * (y / P)<<endl;
```

Trường hợp $x < 4536000$:

```
for (; x <= y; ++x)
    if (check(x))
    {
        fo<<x<<endl;
        return 0;
    }
    fo<<"-1\n";
```

Kiểm tra từ x đến $1.01x$

Hàm kiểm tra số lượng ước:

Số lượng ước tính theo phương pháp phân tích ra thừa số nguyên tố,

Việc phân tích ra thừa số nguyên tố cho phép xử lý với độ phức tạp $O(\sqrt{x})$

Xuất phát từ ước hiến nhiên: 1, số lượng ước xuất phát $cnt = 1$,

Nếu x chứa thừa số p^k thì số ước sẽ tăng thêm $(k+1)$ lần.

```

bool check(int64_t x)
{
    int cnt = 1;
    for (int64_t i = 2; i * i <= x; ++i)
    {
        int cur = 0;
        while (x % i == 0)
        {
            x /= i; cur++;
        }
        cnt *= cur + 1;
    }
    if (x != 1) cnt *= 2;
    return cnt >= 100;
}

```

Tuy nhiên, với **x** không lớn (**x** < 4536000) và chỉ cần kiểm tra số lượng ước có vượt quá 100 hay không việc tính số ước có thể thực hiện đơn giản hơn:

```

for (int64_t i = x; i * 100 <= 101 * x; i++)
{
    int numDivs = 0;
    for (int64_t j = 1; j * j <= i; j++)
    {
        if (numDivs >= 100) break;
        if (i % j == 0)
        {
            numDivs++;
            if (j * j != i) numDivs++;
        }
    }
    if (numDivs >= 100)
    {
        fo << i << endl;
        return 0;
    }
}

```

Chương trình:

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

bool check(int64_t x)
{
    int cnt = 1;
    for (int64_t i = 2; i * i <= x; ++i)
    {
        int cur = 0;
        while (x % i == 0)
        {
            x /= i;
            cur++;
        }
        cnt *= cur + 1;
    }
    if (x != 1) cnt *= 2;
    return cnt >= 100;
}

int main()
{
    const int64_t P = 45360;

    int64_t x;
    fi>>x;
    int64_t y = x + x / 100;
    if (y / P != x / P) fo<< P * (y / P)<<endl;
    else
    {
        for (; x <= y; ++x)
            if (check(x))
            {
                fo<<x<<endl;
                fo<<"\nTime: "<<clock() / (double)1000<<" sec";
                return 0;
            }
        fo<<"-1\n";
    }

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



Ban thư ký gồm n người của một hội nghị quốc tế đang nghiên cứu các tài liệu chuẩn bị cho Đoàn minh thảo luận ở cuộc họp chung. Mỗi người trong Ban thư ký thường biết nhiều thứ tiếng. Số lượng ngôn ngữ khác nhau là m . Người thứ i biết k_i thứ tiếng là $a_{i,1}, a_{i,2}, \dots, a_{i,k_i}$, $i = 1 \div n$. Thỉnh thoảng có những cuộc trao đổi ý kiến ngắn giữa 2 người nào đó trong Ban. Việc trao đổi có thể thực hiện trực tiếp giữa 2 người này hoặc qua các người trung gian thứ 3, thứ 4, . . . Nếu trao đổi trực tiếp thì 2 người sử dụng ngôn ngữ mà cả 2 đều biết, nếu trao đổi qua các người trung gian thì người thứ nhất nói với người thứ 2 bằng ngôn ngữ chung cả hai đều biết, sau đó người thứ 2 nói với người thứ 3 bằng ngôn ngữ chung cả hai đều biết và cứ như thế cho đến người cần nhận thông tin. Nội dung trao đổi thường là khá nhạy cảm về mặt chính trị hay kinh tế vì vậy khi trao đổi ai cũng muốn càng ít người biết nội dung càng tốt. Khi nói trên một ngôn ngữ nào đó thì những ai biết tiếng ấy đều nghe và biết được nội dung.

Hãy xác định với mỗi cặp (A , B) trong Ban thư ký số người ít nhất hiểu nội dung trao đổi giữa 2 người này. Nếu A và B không có cách trao đổi thì coi số lượng cần tìm là -1.

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

- ✚ Dòng đầu tiên chứa 2 số nguyên n và m ($2 \leq n \leq 300$, $1 \leq m \leq 300$),
- ✚ Dòng thứ i trong n dòng sau chứa các số nguyên $k_i, a_{i,1}, a_{i,2}, \dots, a_{i,k_i}$ ($1 \leq k_i, a_{i,j} \leq m$, $a_{i,j} < a_{i,j+1}$).

Kết quả: Đưa ra file văn bản TALK.OUT n dòng, dòng thứ i chứa n số $f_{i,1}, f_{i,2}, \dots, f_{i,n}$, trong đó $f_{i,j}$ – số người hiểu được thông báo khi i nói với j , $f_{i,i} = 0$.

Ví dụ:

TALK.INP	
6	4
2	1 2
2	2 3
1	2
1	1
2	3 4
1	4

TALK.OUT	
0	3 3 2 4 5
3	0 3 4 2 3
3	3 0 4 4 5
2	4 4 0 5 6
4	2 4 5 0 2
5	3 5 6 2 0



Giải thuật: Quy hoạch động, Giải thuật Floyd.

Nhận xét:

Xét một ngôn ngữ L và 2 người A, B . Có thể xác định $d_L(A, B)$ – số người biết thông tin khi A và B trao đổi trực tiếp với nhau bằng ngôn ngữ L :

$$d_L(A, B) = \begin{cases} 0 & \text{với } A = B, \\ \text{Số người biết } L \text{ khi } A \text{ và } B \text{ cùng biết } L, \\ \infty & \text{khi } A \text{ hoặc } B \text{ hoặc cả hai không biết } L. \end{cases}$$

Gọi $\text{distance}(A, B)$ là số người biết ít nhất khi A và B trao đổi trực tiếp với nhau, ta có:

$$\text{distance}(A, B) = \min_{\forall L} \{ d_L(A, B) \}$$

$\text{distance}(A, B)$ tạo thành ma trận kè với mọi cặp $(A, B), A, B = 1 \div n$,

Bằng phương pháp quy hoạch động ta có thể dễ dàng tìm khoảng cách ngắn nhất giữa A và B bất kỳ, trong đó *khoảng cách là số người biết thông tin*.

Kích thước bài toán không lớn vì vậy có thể sử dụng phương pháp *đơn giản* và *dễ lập trình* nhất – *Giải thuật Floyd*,

Để tránh hiệu ứng “*bùng nổ bộ nhớ*” (*thuật ngữ của Bellman*) trong các bài toán quy hoạch động nhiều chiều, thông tin về người biết mỗi ngôn ngữ có thể lưu trữ dưới dạng bít đánh dấu.

Tổ chức dữ liệu:

- Mảng động `vector<vector<int>> langs(n)` – lưu dữ liệu vào,
- Mảng động `vector<bitset<300>> humans(k)` – đánh dấu những người biết với từng ngôn ngữ,
- Mảng động `vector<vector<int>> dist(n, vector<int>(n, INF))` – lưu *số người ít nhất* biết thông tin khi A và B trao đổi với nhau (trực tiếp hoặc gián tiếp qua các người khác) với giá trị khởi tạo là ∞ ,
- Mảng động đánh dấu người biết thông tin trong quá trình trao đổi:

```
vector<vector<human_set>>visited(n, vector<bitset<300>>(n)).
```

Xử lý:

Ghi nhận dữ liệu:

```
for (int i = 0; i < n; ++i)
{
    int x;
    fi>>x;
    langs[i].resize(x);
    for (int j = 0; j < x; ++j)
    {
        fi>>langs[i][j];
        --langs[i][j];
        humans[langs[i][j]].set(i);
    }
}
```

Xin cấp phát
bộ nhớ

Đánh dấu người
biết ngôn ngữ

Khởi tạo giá trị đầu để tìm kiếm:

```
for (int i = 0; i < n; ++i) dist[i][i] = 0;
for (int i = 0; i < n; ++i)
{
    for (int lang : langs[i])
    {
        const auto & curSet = humans[lang];
        int curD = curSet.count();
        for (int j = 0; j < n; ++j)
        {
            if (curSet[j] && dist[i][j] > curD)
            {
                dist[i][j] = curD;
                visited[i][j] = curSet;
            }
        }
    }
}
```

Hai người i, j biết
chung ngôn ngữ

Giải thuật Floyd:

```
for (int k = 0; k < n; ++k)
{
    human_set curSet;
    for (int i = 0; i < n; ++i)
    {
        for (int j = 0; j < n; ++j)
        {
            int maxD = max(dist[i][k], dist[k][j]);
            if (maxD >= dist[i][j]) continue;
            curSet = visited[i][k] | visited[k][j];
            int curD = curSet.count();
            if (curD < dist[i][j])
            {
                dist[i][j] = curD;
                visited[i][j] = curSet;
            }
        }
    }
}
```

Độ phức tạp của giải thuật: $O(\max(n,k)^3)$.

Chương trình:

```
#include <bits/stdc++.h>
#define NAME "talk."
#define times clock() / (double)1000
using namespace std;
ifstream fi (NAME"inp");
ofstream fo (NAME"out");

typedef bitset<300> human_set;
int n, k;

void solve(const vector <vector<int>> & langs,
           const vector <human_set> & humans) {
    int n = int(langs.size());
    const int INF = n + 1;
    vector <vector<int>> dist(n, vector<int>(n, INF));
    vector <vector<human_set>> visited(n, vector<human_set>(n));
    for (int i = 0; i < n; ++i) {
        dist[i][i] = 0;
    }
    for (int i = 0; i < n; ++i)
    {
        for (int lang : langs[i])
        {
            const auto & curSet = humans[lang];
            int curD = curSet.count();
            for (int j = 0; j < n; ++j)
            {
                if (curSet[j] && dist[i][j] > curD)
                {
                    dist[i][j] = curD;
                    visited[i][j] = curSet;
                }
            }
        }
    }
    for (int k = 0; k < n; ++k)
    {
        human_set curSet;
        for (int i = 0; i < n; ++i)
        {
            for (int j = 0; j < n; ++j)
            {
                int maxD = max(dist[i][k], dist[k][j]);
                if (maxD >= dist[i][j]) { continue; }
                curSet = visited[i][k] | visited[k][j];
                int curD = curSet.count();
                if (curD < dist[i][j])
                {
                    dist[i][j] = curD;
                    visited[i][j] = curSet;
                }
            }
        }
    }
    for (int i = 0; i < n; ++i)
    {
        for(int j=0;j<n;++j) fo<<(dist[i][j]== INF ? -1 : dist[i][j])<<' ';
    }
}
```

```

        fo<<'\n';
    }
}
int main()
{
    fi>>n>>k;
    vector <vector<int>> lang(n);
    vector <human_set> humans(k);
    for (int i = 0; i < n; ++i)
    {
        int x;
        fi>>x;
        lang[i].resize(x);
        for (int j = 0; j < x; ++j)
        {
            fi>>lang[i][j];
            --lang[i][j];
            humans[lang[i][j]].set(i);
        }
    }
    n = int(lang.size());
    solve(lang, humans);
    fo<<"\nTime: "<<times<<" sec";
    return 0;
}

```



Giáo sư Braun được mời tới giảng dạy ở một trường đại học danh tiếng ở một thành phố lớn. Theo hợp đồng ông phải làm việc ở nơi mới n ngày. Gần nơi ông ở có khu vực trông giữ xe. Người gửi xe có thể mua vé giữ chỗ riêng theo ngày, theo tuần (7 ngày) hoặc 4 tuần (28 ngày). Ai muốn mua bao nhiêu vé và bao nhiêu loại khác nhau đều được, miễn là có tiền. Dĩ nhiên, có thể mua vé giữ chỗ nhiều hơn số ngày sử dụng thực tế. Giá một vé giữ chỗ ngày là a đồng, một vé giữ chỗ tuần là b đồng và một vé cho 4 tuần là c đồng.

Giáo sư Braun không thiêu tiền, nhưng ông không có thói quen ném tiền qua cửa sổ. Ông lên kế hoạch mua vé thuê chỗ đỗ xe không ít hơn n ngày với chi phí thấp nhất.

Hãy xác định số tiền GS Braun phải chi.

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

- ✚ Dòng đầu tiên chứa 3 số nguyên a , b và c ($1 \leq a \leq b \leq c \leq 1\,000$),
- ✚ Dòng thứ 2 chứa số nguyên n ($1 \leq n \leq 10^{15}$).

Kết quả: Đưa ra file văn bản PARKING.OUT một số nguyên – số tiền phải chi trả.

Ví dụ:

PARKING.INP	PARKING.OUT
4 7 20 10	14



Giải thuật: Số học đơn giản, khả năng phân tích lô gic.

Nhận xét:

- ⊕ Có thể chỉ mua một loại vé (vé ngày, vé tuần hay vé 4 tuần) hoặc mua hỗn hợp nhiều loại vé,
- ⊕ Trường hợp chỉ mua một loại vé: chi phí tương ứng sẽ là
 - ✿ Chỉ mua loại vé ngày: $x_a = n \times a$,
 - ✿ Chỉ mua loại vé tuần: $x_b = (n+6) / 7 \times b$,
 - ✿ Chỉ mua loại vé 4 tuần: $x_c = (n+27) / 28 \times c$,
- ⊕ Trường hợp mua nhiều loại vé:
 - ✿ Mua 2 loại vé tuần và vé ngày: dễ dàng thấy rằng *chỉ cần xét khi $n \% 7 \neq 0$* , chi phí sẽ là $n / 7 \times b + n \% 7 \times a$,
 - ✿ Mua nhiều loại vé, trong đó có sử dụng loại vé 4 tuần: *chỉ cần xét khi $n \% 28 \neq 0$* , sau khi mua $n / 28$ vé loại 4 tuần, số ngày còn lại $r = n \% 28$ có thể mua bằng một hoặc 2 loại vé ngày, tuần.
- ⊕ So sánh chi phí trong các trường hợp và lựa chọn kết quả nhỏ nhất.

Độ phức tạp của giải thuật: O(1).

Chương trình:

```
#include <bits/stdc++.h>
#define NAME "parking."
#define times clock() / (double)1000
using namespace std;
ifstream fi (NAME"inp");
ofstream fo (NAME"out");
int a,b,c;
int64_t n,t,r,trb,trab,xa,xb,xc,ans;

int main()
{
    fi>>a>>b>>c>>n;
    xa=n*a;
    xb=(n+6)/7*b;
    t=n/7*b+n%7*a;
    if(xb>t) xb=t;
    xc=(n+27)/28*c;
    t=n/28; r=n%28;
    trb=(r+6)/7*b;
    trab=r/7*b+r%7*a;
    if(trb>trab) trb=trab;
    t=t*c+trb;
    if(xc>t) xc=t;
    ans=xa; if(ans>xb) ans=xb; if(ans>xc) ans=xc;
    fo<<ans;
}
```



Tuyến xe buýt nhanh được đưa vào khai thác thực nghiệm trên trục lộ chính của thành phố. Toàn tuyến có n điểm đỗ, đánh số từ 0 đến $n-1$. Điểm đỗ số 0 ở đầu trục lộ, điểm đỗ $n-1$ – ở cuối trục lộ. Cạnh một số điểm đỗ là cụm dân cư. Ở mỗi điểm đỗ cạnh cụm dân cư có một ô tô buýt xuất phát. Các ô tô buýt chạy từ điểm xuất phát của mình về phía cuối trục lộ. Mỗi ô tô buýt sau, khi xuất phát gặp cụm dân cư thứ k trên đường đi của mình hoặc điểm cuối của trục lộ thì quay lại điểm xuất phát. Riêng ô tô buýt ở cụm dân cư cuối cùng từ trái sang tạm thời không tham gia vào kế hoạch khai thác thực nghiệm này.

Thời gian ô tô buýt đi từ một điểm đỗ tới điểm đỗ tiếp theo là 1 phút.

Với mỗi ô tô buýt hãy xác định thời gian quay vòng – khoảng thời gian chạy kể từ khi xuất phát cho đến khi xe quay lại điểm xuất phát.

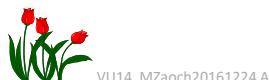
Dữ liệu: Vào từ file văn bản BRT.INP:

- ✚ Dòng đầu tiên chứa một số nguyên k ($1 \leq k \leq 3 \times 10^5$),
- ✚ Dòng thứ 2 chứa xâu s độ dài n mô tả các điểm đỗ, s_i bằng ‘0’ nếu điểm đỗ i không cạnh cụm dân cư và bằng ‘1’ trong trường hợp ngược lại, ($1 \leq n \leq 3 \times 10^5$).

Kết quả: Đưa ra file văn bản BRT.OUT trên một dòng m số nguyên xác định thời gian quay vòng của mỗi xe, trong đó m – số cụm dân cư ở điểm đỗ trên trục lộ.

Ví dụ:

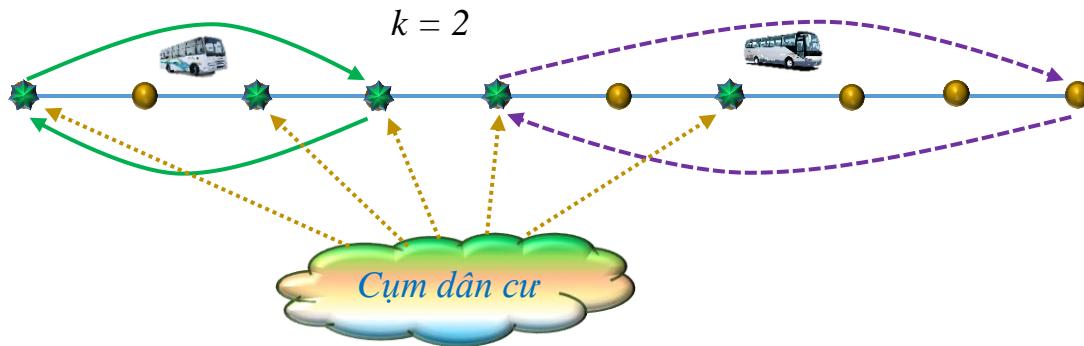
BRT.INP	BRT.OUT
<pre>1 10010001000</pre>	<pre>6 8 0</pre>



Giải thuật: Quản lý khoảng tuyến tính.

Nhận xét:

- Sử dụng 2 con trỏ p và q quản lý điểm đầu và cuối tuyến đường đi mỗi ô tô buýt,



- Dễ dàng xác định p và q cho ô tô buýt đầu tiên,
- Với các ô tô buýt tiếp theo: *cập nhật* p và q,
- Thời gian quay vòng là $(q-p) \times 2$,
- Việc cập nhật và đưa ra kết quả được thực hiện chừng nào p còn nhỏ hơn vị trí cụm dân cư cuối cùng,
- Để thuận tiện cập nhật q nên thêm giá trị hàng rào sn='1'.

Độ phức tạp của giải thuật: $O(n)$.

Chương trình:

```
#include <bits/stdc++.h>
#define NAME "brt."
#define times clock() / (double)1000
using namespace std;
ifstream fi (NAME"inp");
ofstream fo (NAME"out");
int n,k,m,p,q,d;
string s;

int main()
{
    fi>>k>>s;
    p=0; n=s.size();s+='+';
    for(int i=0;i<n;++i) if(s[i]=='1') {p=i; break;}
    q=n-1; m=0;
    for(int i=p+1;i<n;++i)
    {
        m+=s[i]-48;
        if(m==k) {q=i;break;}
    }
    while(p<n)
    {
        d=(q-p)*2;
        fo<<d<<' ';
        ++p; while(s[p]=='0' && p<n)++p;
        if(q<n-1) ++q; while(s[q]=='0' && q<n-1)++q;
    }
    fo<<"\nTime: "<<times<<" sec";
}
```



Bonsai bay là mặt hàng tết rất được ưa chuộng trên thị trường. Tùy theo loại cây trồng, các chậu hoa được chia thành những lớp khác nhau, giá mỗi chậu trong cùng một lớp là như nhau. Công đoàn của một bệnh viện quyết định duyệt chi **s** đồng mua các chậu bonsai loại F về trang trí phòng khách đón tết để mang lại niềm vui cho các bệnh nhân phải ở lại trong dịp tết. Giá mỗi chậu loại F là **k** đồng.

Nhưng giá xăng lại tăng và kéo theo sự tăng giá của các mặt hàng khác. Giá mỗi chậu bonsai định mua tăng thêm **p** phần trăm.



Hãy xác định với số tiền đã duyệt bây giờ có thể mua được bao nhiêu chậu bonsai bay.

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

- ⊕ Dòng đầu tiên chứa một số nguyên **k** ($1 \leq k \leq 10^9$),
- ⊕ Dòng thứ 2 chứa số nguyên **p** ($0 \leq p < 100$),
- ⊕ Dòng thứ 3 chứa số nguyên **s** ($1 \leq s \leq 10^9$).

Kết quả: Đưa ra file văn bản BONSAI.OUT một số nguyên – số chậu hoa có thể mua.

Ví dụ:

BONSAI.INP
33
5
100

BONSAI.OUT
2



Giải thuật: Kiến thức cơ sở, vòng tránh số thực (chống tích lũy sai số làm tròn).

Nhận xét:

Về mặt toán học, kết quả cần tìm là phần nguyên của biểu thức $s/(k \times (1+p/100))$,

Mẫu số là số thực, nói chung là một giá trị gần đúng, việc chia cho một số thực sẽ cho một kết quả gần đúng,

Ảnh hưởng của sai số làm tròn sẽ đặc biệt lớn khi số bị chia và số chia có giá trị gần giống nhau.

Cách đơn giản nhất vòng tránh ảnh hưởng của sai số làm tròn là chuyển về bài toán tương đương chỉ làm việc với số nguyên, trong trường hợp này – tính đơn giá và số tiền theo đơn vị xu, tức là tăng lên 100 lần.

Độ phức tạp của giải thuật: O(1).

Chương trình:

```
#include <bits/stdc++.h>
#define NAME "bonsai."
#define times clock() / (double)1000
using namespace std;
ifstream fi (NAME"inp");
ofstream fo (NAME"out");
int64_t k,s,p,ans;

int main()
{
    fi>>k>>p>>s;
    ans=(s*100) / (k*(100+p));
    fo<<ans;
    fo<<"\nTime: "<<times<<" sec";
}
```



Giáo sư Braun chuẩn bị tham gia vào một đoàn thám hiểm. Mỗi thành viên được mang theo người không quá s kg hành lý trong ba lô, còn lại phải gửi theo đường hàng hóa vận chuyển tới sau. Quá bận bịu với các công việc chuẩn bị cho đoàn, ông không còn nhiều thời gian đóng gói đồ đạc của mình. Ông bày ra sàn tất cả những gì cần mang theo và sắp xếp chúng theo thứ tự giảm dần của mức quan trọng. Tất cả có n đồ vật, vật thứ i có trọng lượng a_i . Lần lượt duyệt từ đầu đến cuối, nếu gặp đồ vật còn cho được vào ba lô ông gạt lại sang phần mang theo người, nếu không cho được vào ba lô – ông bỏ vào valy hàng hóa gửi theo sau.

Hãy xác định có bao nhiêu kg hàng hóa được chọn mang theo người và bao nhiêu kg hàng gửi theo sau.

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

- ✚ Dòng đầu tiên chứa một số nguyên s ($1 \leq s \leq 10^9$),
- ✚ Dòng thứ 2 chứa số nguyên n ($1 \leq n \leq 10^5$),
- ✚ Dòng thứ 3 chứa n số nguyên a_1, a_2, \dots, a_n ($1 \leq a_i \leq 2 \times 10^9$, $i = 1 \dots n$).

Kết quả: Đưa ra file văn bản BAGGAGE.OUT hai số nguyên, mỗi số trên một dòng xác định trọng lượng hàng mang theo người và trọng lượng hàng gửi chuyển tới sau.

Ví dụ:

BAGGAGE.INP	BAGGAGE.OUT
20	18
5	8
6 10 5 2 3	



Giải thuật: Kiến thức cơ sở, tránh lưu dữ liệu thừa.

Nhận xét:

Dữ liệu đã được sắp xếp theo mức quan trọng, tức là theo trình tự cần xử lý, Mỗi dữ liệu nhập vào: sau khi xử lý, có thể “quên”,

Quy trình xử lý: nếu trọng lượng vật tiếp theo nhỏ hơn trọng lượng còn có thể mang theo – tích lũy vào phần mang theo và giảm tương ứng trọng lượng mang theo còn lại, nếu khác – tích lũy vào phần gửi chậm.

Độ phức tạp của giải thuật: O(n).

Chương trình:

```
#include <bits/stdc++.h>
#define NAME "baggage."
#define times clock() / (double)1000
using namespace std;
ifstream fi (NAME"inp");
ofstream fo (NAME"out");
int64_t ans2=0;
int n,s,t,ans1=0;

int main()
{
    fi>>s>>n;
    for(int i=0;i<n;++i)
    {
        fi>>t;
        if(t<=s)ans1+=t, s-=t;else ans2+=t;
    }

    fo<<ans1<<'n'<<ans2;
    fo<<"\nTime: "<<times<<" sec";
}
```



VU17. TOUR DU LỊCH

Tên chương trình: TOUR.CPP

Du lịch là ngành công nghiệp không khói, đóng góp nhiều cho ngân sách nhà nước. Tiền thuế cho mỗi tour du lịch được tính theo số trạm dừng chân trên tuyến của tour du lịch.

Tour du lịch sinh thái xuyên rừng đầy đủ đi từ tây sang đông có n trạm dừng, tính từ tây sang đông trạm thứ i ở độ cao h_i . Để thu hút khách du lịch người ta tổ chức thêm một tour tiết kiệm gồm một đoạn đường ngắn nhất có thể để giảm thiểu ảnh hưởng thuế lên giá vé, trong đó có đoạn lên dốc và đoạn xuống dốc. Dĩ nhiên, tour phải bắt đầu từ một trạm dừng chân nào đó và kết thúc ở trạm dừng chân khác và cũng đi từ tây sang đông.

Hãy xác định trạm đầu và trạm cuối của tour tiết kiệm.

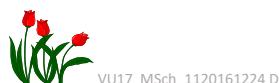
Dữ liệu: Vào từ file văn bản TOUR.INP:

- ✚ Dòng đầu tiên chứa một số nguyên n ($1 \leq n \leq 10^6$),
- ✚ Dòng thứ 2 chứa n số nguyên h_1, h_2, \dots, h_n ($0 \leq h_i \leq 10^9$, $i = 1 \dots n$).

Kết quả: Đưa ra file văn bản TOUR.OUT hai số nguyên, mỗi số trên một dòng xác định các trạm đầu và cuối. Trường hợp không thể chọn tour theo yêu cầu đã nêu – đưa ra một số 0.

Ví dụ:

TOUR.INP	TOUR.OUT
7	3
18 10 15 20 20 10 3	6



Giải thuật: Phương pháp 2 con trỏ, tránh lưu dữ liệu thừa.

Nhận xét:

Đoạn tiềm năng có thể chọn kéo dài từ một trạm trước đỉnh núi tới một trạm sau đỉnh núi,

Nếu không tồn tại đỉnh núi – kết quả là 0,

Tình hình phức tạp khi đỉnh núi là một số điểm có cùng độ cao,

Gọi **llen** là số trạm cùng độ cao trên đỉnh ($llen \geq 1$), ta chỉ cần tìm đoạn có **llen** nhỏ nhất, để làm được việc đó cần lưu **prh** – độ cao trước khi tới đỉnh.

Tổ chức dữ liệu:

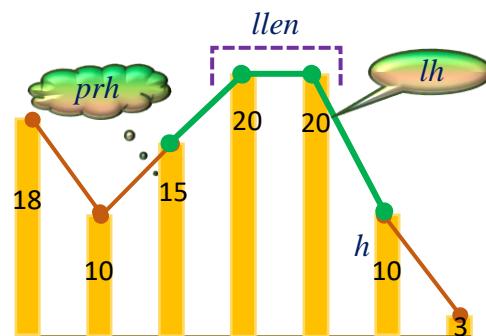
llen – là số trạm cùng độ cao trên đỉnh,

prh – độ cao trước khi tới đỉnh,

lh – độ cao của đỉnh,

h – độ cao trạm đang xét,

lmin – số lượng nhỏ nhất các trạm cùng độ cao trên đỉnh.



Xử lý:

Với mỗi độ cao **h** tiếp theo phân biệt 3 trường hợp:

- ✿ Cùng độ cao đỉnh,
- ✿ Bắt đầu xuống dốc,
- ✿ Các trường hợp còn lại.

Mỗi khi bắt đầu xuống dốc: cần cập nhật kết quả đoạn tiềm năng có thể chọn.

Độ phức tạp của giải thuật: $O(n)$.

Chương trình:

```
#include <bits/stdc++.h>
#define NAME "tour."
#define times clock() / (double)1000
using namespace std;
ifstream fi (NAME"inp");
ofstream fo (NAME"out");
const int INF=1e9;
int n,h,lmin=INF, lh=INF, ans1=0, ans2=0, prh=INF, llen=0;

int main()
{
    fi>>n;
    for(int i=1;i<=n;++i)
    {
        fi>>h;
        if(h==lh)++llen;
        else
        {
            if(prh<lh && lh>h && llen<lmin)
            {
                lmin=llen;
                ans1=i-llen-1; ans2=i;
                if(lmin==1)break;
            }
            prh=lh; lh=h; llen=1;
        }
    }
    if(ans1==0) fo<<0;else fo<<ans1<<'\
'<<ans2;
    fo<<"\nTime: "<<times<<" sec";
}
```



VU18. PHÂN LỚP

Tên chương trình: SUBCLASS.CPP

Giờ Tin học. Sau khi giảng về cách nhận biết một số nguyên có chia hết cho số nguyên khác hay không, thầy giáo viết trên bảng số nguyên n và yêu cầu mọi người chia các số nguyên từ 1 tới n vào các lớp, mỗi lớp chứa các số không chia hết cho nhau, tức là nếu một số chia hết cho số kia thì 2 số đó phải ở 2 lớp khác nhau.

Ví dụ, với $n = 10$, các lớp có thể là {1}, {2, 7, 9}, {3, 4, 10} và {5, 6, 8}, tất cả có 4 lớp. Có rất nhiều cách phân lớp khác nhau.

Hãy xác định số lượng lớp nhỏ nhất phân chia được các số theo điều kiện đã nêu.

Dữ liệu: Vào từ file văn bản SUBCLASS.INP gồm một dòng chứa số nguyên n ($1 \leq n \leq 10^{15}$).

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

Ví dụ:

SUBCLASS.INP	SUBCLASS.OUT
10	4



Giải thuật: Kỹ năng phân tích, đoán nhận giải thuật.

Nhận xét:

Gọi \mathbf{k} là số nguyên lớn nhất thỏa mãn điều kiện $2^k \leq n$,

Rõ ràng thấy 2^q chia hết cho 2^p với $0 \leq p < q \leq k$, vì vậy các số $2^0, 2^1, 2^2, \dots, 2^k$ phải ở những lớp khác nhau,

Số lớp tối thiểu phải có là $k+1$,

Bằng việc chỉ ra một cách phân lớp cụ thể, ta sẽ chứng minh là chỉ cần $k+1$ lớp là đủ:

- ✚ Lớp 0 chứa số $2^0 = 1$,
- ✚ Lớp i chứa các số từ 2^i đến $2^{i+1}-1$, $i = 1 \div k-1$,
- ✚ Lớp k chứa các số từ 2^k tới n .
- ✚ Xét lớp có nhiều hơn một số: gọi a là số nhỏ nhất trong lớp và x là một số bất kỳ trong lớp, $x \neq a$, ta có $a < x < 2 \times a$ vì vậy x không chia hết cho a và càng không chia hết cho số nào lớn hơn a ở trong lớp đó.

Độ phức tạp của giải thuật: $O(1)$.

Chương trình:

```
#include <bits/stdc++.h>
#define NAME "subclass."
#define times clock() / (double)1000
using namespace std;
ifstream fi (NAME"inp");
ofstream fo (NAME"out");
int64_t n;
int ans;

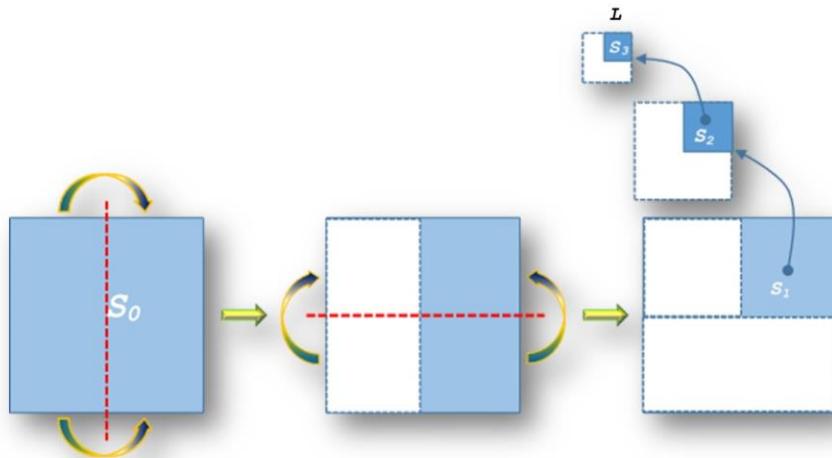
int main()
{
    fi>>n;
    for(int i=62;i>0;--i)
        if( (n>>i) & 1) {ans=i+1;break;}
    fo<<ans;
    fo<<"\nTime: "<<times<<" sec";
}
```



VU19. DÃY HÌNH VUÔNG

Tên chương trình: SQUARES.CPP

Xét hình vuông s_0 , gấp s_0 theo 2 đường trung bình sẽ tạo được hình vuông s_1 . Quá trình gấp được thực hiện lại tương tự với hình vuông s_1 để tạo hình vuông s_2 , và cứ thế tạo được dãy các hình vuông: s_0, s_1, \dots, s_N .



Cho L là độ dài cạnh của s_N , hãy tính T là tổng diện tích các hình vuông của dãy và đưa ra số dư của T chia cho (10^9+7)

Dữ liệu: Vào từ file văn bản SQUARES.INP gồm 1 dòng ghi 2 số nguyên N và L ($0 \leq N, L \leq 10^9$)
Kết quả: Đưa ra file văn bản SQUARES.OUT số dư tìm được.

Ví dụ:

SQUARES.INP	SQUARES.OUT
3 1	85



Giải thuật: Tính nhanh lũy thừa, tìm số dư của phân số.

Nhận xét:

Tính từ cuối về đầu, sau mỗi lần biến đổi kích thước hình vuông tăng 2 lần, diện tích sẽ tăng 4 lần,

Ta có $T = L^2 + 4L^2 + 4^2L^2 + \dots + 4^{n-1}L^2$

$$= L^2 \times \frac{4^n - 1}{3}$$

Dễ dàng chứng minh được là $T \% p = (L^2 \times (4^n - 1)) \% (3 \times p) / 3$,

Cách tính nhanh $4^n \% p$, trong đó $p = 3 \times 3$:

```
t=4;p3=p*3;a=(a%p*a%p3)%p3;ta=a;
ans=1;
while(n)
{
    if(n&1)ans=(ans*t)%p3;
    t=(t*t)%p3;
    n>>=1;
}
```

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

Chương trình:

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

int64_t a,n,nn;
int64_t ans,t,ta,p3,p=1e9+7;

int main()
{
    fi>>n>>a;++n;
    t=4;p3=p*3;a=(a%p*a%p3)%p3;ta=a;
    ans=1;
    while(n)
    {
        if(n&1)ans=(ans*t)%p3;
        t=(t*t)%p3;
        n>>=1;
    }

    ans=(ans-1)*ta%p3/3;
    fo<<ans;
}
```



Cho 2 phân số đúng và tối giản $\frac{a}{b}$, $\frac{c}{d}$. Mỗi phép biến đổi là tăng a và b lên 1, sau đó giản ước phân số nhận được.

Hãy xác định sau bao nhiêu bước biến đổi từ phân số thứ nhất ban đầu nhận được phân số thứ 2 đã cho.

Dữ liệu: Vào từ file văn bản FRACTION.INP gồm một dòng chứa 4 số nguyên a , b , c , d , $0 < a < b \leq 10^5$, $0 < c < d \leq 10^5$, a và b nguyên tố cùng nhau, c và d nguyên tố cùng nhau $\frac{a}{b} \neq \frac{c}{d}$.

Kết quả: Đưa ra file văn bản FRACTION.OUT số 0 nếu không có cách biến đổi hoặc một số nguyên – số lượng phép biến đổi.

Ví dụ:

SUBCLASS.INP	SUBCLASS.OUT
1 6 2 3	5



Giải thuật: Kỹ thuật phân tích, đoán nhận giải thuật.

Nhận xét:

Xét $\frac{a+1}{b+1} - \frac{a}{b}$, sau khi quy đồng mẫu số, tử số sẽ là $(a+1) \times b - a \times (b+1) = b - a > 0$,

Như vậy, sau mỗi phép biến đổi *phân số sẽ tăng* và *tiến dần tới 1* khi số phép biến đổi tăng vô hạn,

Từ đó suy ra: không thể biến đổi nếu $\frac{a}{b} > \frac{c}{d}$.

Xét trường hợp $\frac{a}{b} < \frac{c}{d}$.

Ta không thể tìm số phép biến đổi bằng cách giải phương trình $\frac{a+x}{b+x} = \frac{c}{d}$ vì việc tối giản sau mỗi phép biến đổi sẽ làm thay đổi điểm xuất phát của bước tiếp theo,

Với lý do tương tự – giải thuật tìm kiếm nhị phân cũng không thích hợp!

Ràng buộc dữ liệu vào cho phép giải bài toán bằng cách mô phỏng biến đổi, lần lượt kiểm tra các phân số nhận được.

Lưu ý: Việc kiểm tra điều kiện dừng của giải thuật phụ thuộc vào cách khai báo dữ liệu.

Độ phức tạp của giải thuật: Trong trường hợp xấu nhất – không vượt quá 10^8 phép biến đổi..

Chương trình:

```
#include <bits/stdc++.h>
#define NAME "fraction."
#define times clock()/(double)1000
using namespace std;
ifstream fi (NAME"inp");
ofstream fo (NAME"out");
int64_t a,b,c,d,ans=0,k;

int main()
{
    fi>>a>>b>>c>>d;
    while(a*d<b*c)
    {
        ++a; ++b; ++ans; k=__gcd(a,b);
        a/=k; b/=k;
    }
    if(a!=c || b!=d) ans=0;
    fo<<ans;
    fo<<"\nTime: "<<times<<" sec";
}
```



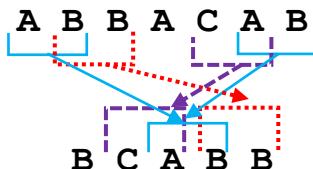
VU21. MỨC ĐỘ GIỐNG NHAU

Tên chương trình: SAMENESS.CPP

Một quần thể sinh vật ngoài hành tinh có gên tạo thành từ 26 cơ sở ký hiệu từ **A** tới **Z**. Mỗi gene là một xâu ký tự các chữ cái la tinh hoa. Hai cá thể khác nhau có gene khác nhau. Mức độ giống nhau của 2 gene sẽ xác định độ huyết thống của chúng.

Độ giống nhau của gene thứ nhất với gene thứ 2 là số lượng cặp 2 cơ sở liên tiếp nhau trong gene thứ nhất đồng thời là cặp 2 cơ sở liên tiếp nhau trong gene thứ hai.

Ví dụ, gene thứ nhất là **ABBACAB**, gene thứ 2 là **BCABB** thì mức độ giống nhau là 4.



Cho hai genes. Hãy xác định mức độ giống nhau của gene thứ nhất với gene thứ 2.

Dữ liệu: Vào từ file văn bản SAMENESS.INP bao gồm 2 dòng chứa 2 xâu khác rỗng xác định gene thứ nhất và gene thứ 2, mỗi xâu chỉ chứa các ký tự chữ cái in hoa và có độ dài không quá 10^5 .

Kết quả: Đưa ra file văn bản SAMENESS.OUT một số nguyên – mức độ giống nhau của 2 genes.

Ví dụ:

SAMENESS.INP	SAMENESS.OUT
ABBACAB BCABB	4



VU21 MOkr_1120161224 3

Giải thuật: Thống kê tần số.

Nhận xét:

Nguyên lý chung giải quyết các bài toán thống kê tần số là xác định các mẫu thống kê, đếm số lần xuất hiện của các mẫu thống kê trong đối được cần thống kê.

Ở bài toán cụ thể này số lượng mẫu thống kê không vượt quá 26×26 , đó là các cặp ký tự **AA**, **AB**, **AC**, ..., **ZZ**.

Bảng mẫu được cho ở xâu thứ 2, dựa vào xâu này đánh dấu các mẫu xuất hiện bằng bảng **f[26][26]**, $f_{i,j} = 1$ nếu trong mẫu có cặp liên tiếp các ký tự thứ **i** và thứ **j**, $f_{i,j} = 0$ trong trường hợp ngược lại.

Do bảng mẫu được cho sau, để tiết kiệm bộ nhớ ta sẽ [xây dựng ngay trong quá trình thống kê](#),

Với xâu cần thống kê và đánh giá (xâu thứ nhất): xây dựng bảng tần số các cặp ký tự liên tiếp trong xâu (lưu vào mảng **d[26][26]**).

Dựa vào **f** và **d** tích lũy các tần số của xâu thứ nhất được đánh dấu trong **f**.

Tổ chức dữ liệu:

Xâu s để lần lượt lưu trữ các xâu ở dữ liệu vào,

Mảng **int f[26][26]={0}** – lưu đánh dấu mẫu,

Mảng **int d[26][26]={0}** – lưu tần số các cặp cơ sở của gene thứ nhất.

Xử lý:

Đọc gene thứ nhất và xác định tần số xuất hiện các cặp cơ sở,

Đọc gene thứ hai, kiểm và tích lũy kết quả với mỗi mẫu cơ sở đang xử lý, đánh dấu sự tồn tại và đã xử lý của cặp cơ sở.

Độ phức tạp của giải thuật: $O(n+m)$, trong đó n, m – độ dài các xâu.

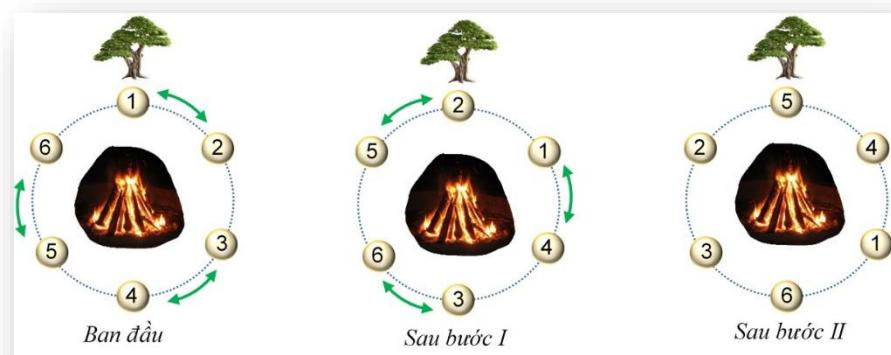
Chương trình:

```
#include <bits/stdc++.h>
#define NAME "sameness."
#define times clock() / (double)1000
using namespace std;
ifstream fi (NAME"inp");
ofstream fo (NAME"out");
int n,d[26][26]={0},f[26][26]={0};
int64_t ans=0;
string s;

int main()
{
    fi>>s; n=s.size();
    for(int i=0;i<n-1;++i)++d[s[i]-65][s[i+1]-65];
    fi>>s; n=s.size();
    for(int i=0;i<n-1;++i)
    {
        if(f[s[i]-65][s[i+1]-65]==0)ans+=d[s[i]-65][s[i+1]-65];
        f[s[i]-65][s[i+1]-65]=1;
    }
    fo<<ans;
    fo<<"\nTime: "<<times<<" sec";
}
```



Đêm đốt lửa trại kết thúc bằng điệu múa tập thể. Mọi người nắm tay nhau thành một vòng tròn n người quanh đống lửa (n chẵn), đánh số từ 1 đến n theo chiều kim đồng hồ, bắt đầu từ người đứng cạnh gốc cây to. Ở bước thứ nhất và các bước lẻ tiếp theo người đứng cạnh cây và người bên cạnh theo theo chiều kim đồng hồ tạo thành một cặp, hai người tiếp theo – thành một cặp, . . . Ở các bước chẵn người đứng cạnh cây và người bên cạnh theo theo chiều ngược kim đồng hồ tạo thành một cặp, hai người tiếp theo – thành một cặp, . . . Cứ mỗi bước nhảy cặp 2 người cạnh nhau lại đổi chỗ.



Hình trên tương ứng với $n = 6$ và trạng thái vòng tròn sau 2 bước nhảy.

Hãy xác định 2 người đứng cạnh người p sau k bước nhảy.

Dữ liệu: Vào từ file văn bản DANCERS.INP gồm một dòng chứa 3 số nguyên n, p và k ($2 \leq n \leq 10^9$, n – chẵn, $1 \leq k \leq 10^9$, $1 \leq p \leq n$).

Kết quả: Đưa ra file văn bản DANCERS.OUT trên một dòng 2 số nguyên theo thứ tự tăng dần, xác định những người đứng cạnh p .

Ví dụ:

DANCERS.INP	DANCERS.OUT
6 5 2	2 4



Giải thuật: Nhận dạng và xử lý chu kỳ.

Nhận xét:

Khảo sát, tìm quy luật biến đổi của dữ liệu: xét $n = 8$ và các trạng thái sau quá trình 8 bước nhảy:

<i>Trạng thái ban đầu</i>	1	2	3	4	5	6	7	8
<i>Trạng thái sau 1 bước</i>	2	1	4	3	6	5	8	7
<i>Trạng thái sau 2 bước</i>	7	4	1	6	3	8	5	2
<i>Trạng thái sau 3 bước</i>	4	7	6	1	8	3	2	5
<i>Trạng thái sau 4 bước</i>	5	6	7	8	1	2	3	4
<i>Trạng thái sau 5 bước</i>	6	5	8	7	2	1	4	3
<i>Trạng thái sau 6 bước</i>	3	8	5	2	7	4	1	6
<i>Trạng thái sau 7 bước</i>	8	3	2	5	4	7	6	1
<i>Trạng thái sau 8 bước</i>	1	2	3	4	5	6	7	8

Từ bảng khảo sát trên ta thấy những người ở vị trí lẻ (1, 3, 5, 7) luôn di chuyển sang phải, những người số chẵn (2, 4, 6, 8) – sang trái, như vậy sau n bước mọi người trở về vị trí ban đầu của mình. Trạng thái sau k bước giống trạng thái sau $k \% n$ bước.

Ban đầu 2 người bên cạnh p là $p-1$ và $p+1$ (0 tương ứng với n , $n+1$ – tương ứng với 1). Nếu p lẻ thì sau mỗi bước, số của mỗi bên cạnh tăng lên 2, nếu p chẵn, sau mỗi bước số của các người bên cạnh giảm 2.

Từ các quy luật đã quan sát thấy ta dễ dàng lập trình tìm nghiệm của bài toán.

Độ phức tạp của giải thuật: O(1).

Chương trình:

```
#include <bits/stdc++.h>
#define NAME "dancers."
#define times clock() / (double)1000
using namespace std;
ifstream fi (NAME"inp");
ofstream fo (NAME"out");
int n,k,p,ans1,ans2;

int main()
{
    fi>>n>>p>>k;
    k%=n;
    ans1=p-1; ans2=p+1;
    if(p&1) ans1+=2*k, ans2+=2*k;
    else ans1-=2*k, ans2-=2*k;
    ans1%=n; if(ans1<0) ans1+=n;
    ans2%=n; if(ans2<0) ans2+=n;
    if(ans1>ans2) swap(ans1,ans2);
    fo<<ans1<<' '<<ans2;
    fo<<"\nTime: "<<times<<" sec";
}
```



Quy luật của cuộc sống là “*Mode tới rồi rời đi*”. Những kiểu tóc, quần áo, dày dép hôm nay đang được cực kỳ ưa chuộng, nhưng ngày mai có thể chẳng còn được ai ngó ngàng tới. Chủ một cửa hàng thời trang rất có uy tín nắm vững quy luật này. Ông quyết định bán hạ giá tất cả các mặt hàng sắp hết mode để chuyển sang các mặt hàng khác mang lại lợi nhuận cao hơn. Giá các mặt hàng hiện có sẽ được bán với giá mới, giảm 25% so với giá hiện tại. Thật may mắn là giá cũ và giá mới đều là nguyên. Sau khi đóng cửa hàng, ông tháo biển giá cũ xuống và dựa vào đó làm biển giá mới, xếp từng cặp trên quầy để ngày mai treo bảng giá mới.

Sáng hôm sau, khi chuẩn bị mở cửa hàng ông phát hiện ra một điều cười được mà khóc cũng không xong: các nhân viên dọn dẹp đã quá nhiệt tình và chu đáo, sắp xếp tất cả n biển giá trên quầy thành một dãy tăng dần theo giá trị.

Ông phải gấp rút tìm ra các biển giá mới để treo.

Hay xác định các biển giá mới cần tìm.

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

- ✚ Dòng đầu tiên chứa một số nguyên n ($2 \leq n \leq 10^5$, n chẵn),
- ✚ Dòng thứ i trong n dòng sau chứa số nguyên a_i – số trên biển giá i ($1 \leq a_i \leq 10^9$).

Kết quả: Đưa ra file văn bản PRICES.OUT số trên các biển giá được chọn, mỗi số trên một dòng.

Ví dụ:

PRICES.INP	PRICES.OUT
6	30
30	42
40	45
42	
45	
56	
60	



Giải thuật: Nguyên lý cúc trị, Phương pháp hai con trỏ.

Nhận xét:

Phương pháp 2 con trỏ:

Dữ liệu đã được sắp xếp tăng dần, vì vậy số đầu tiên của dãy (a_1) phải là một biến giá mới,

Gọi p là số thứ tự trong dãy đã cho của biến giá mới tìm được, ban đầu $p = 1$, giá cũ của mặt hàng này là $t = a_p / 3 * 4$, đánh dấu loại bỏ phần tử p trong dãy và tìm q nhỏ nhất trong số các phần tử chưa bị loại bỏ thỏa mãn điều kiện $a_q = t$, đánh dấu loại bỏ phần tử này,

Cập nhật p và q : tìm tiếp từ p và q hiện tại,

Trong quá trình xử lý: đưa ra các giá trị a_p tìm được,

Quá trình xử lý kết thúc khi tìm được $n/2$ giá trị.

Tổ chức dữ liệu:

Mảng `int a[100000]` – lưu dữ liệu input,

Mảng `int f[100000]` – đánh dấu loại bỏ.

Độ phức tạp của giải thuật: $O(n)$.

Phương pháp trực tiếp áp dụng nguyên lý cúc trị:

Số nhỏ nhất trong dãy (ban đầu là a_1) là giá mới, đưa ra và loại bỏ số này cùng với số tương ứng với giá cũ của nó ($a_1 / 3 * 4$) ra khỏi dãy,

Lặp lại công việc trên cho đến khi tất cả các số bị loại bỏ.

Tổ chức dữ liệu: Dùng `multiset <int> a` để lưu dãy dữ liệu cần xử lý.

Lô gic thuật toán đơn giản, nhưng thời gian thực hiện sẽ lớn hơn đôi chút ☺.

Chương trình: Phương pháp 2 con trỏ.

```
#include <bits/stdc++.h>
#define NAME "prices."
#define times clock() / (double)1000
using namespace std;
ifstream fi (NAME"inp");
ofstream fo (NAME"out");
const int N = 200000;
int n,m=0,n2,p,q,t,a[N],f[N]={0};

int main()
{
    fi>>n; n2=n/2;
    for(int i=0;i<n;++i) fi>>a[i];
    p=0; f[0]=1; t=a[0]/3*4; fo<<a[p]<<'\\n';
    for(int i=1;i<n;++i)
        if(f[i]==0 && a[i]==t) {q=i; f[i]=1; break;}
    while(m<n2-1)
    {
        while(f[p])+p; t=a[p]/3*4; f[p]=1; fo<<a[p]<<'\\n'; ++m;
        if(m<n2-1)while(f[++q]==1 || a[q]!=t);
    }
    fo<<"\\nTime: "<<times<<" sec";
}
```

Chương trình: Phương pháp trực tiếp áp dụng nguyên lý cực trị

```
#include <bits/stdc++.h>
#define NAME "prices."
#define times clock() / (double)1000
using namespace std;
ifstream fi (NAME"inp");
ofstream fo (NAME"out");
const int N = 200000;
int n,t,p;

int main()
{
    fi>>n;
    multiset<int>a;
    for(int i=0;i<n;++i)
    {
        fi>>t; a.insert(t);
    }
    while(!a.empty())
    {
        p=*a.begin();
        fo<<p<<'\\n'; p=p/3*4;
        a.erase(a.begin());
        a.erase(a.find(p));
    }
    fo<<"\\nTime: "<<times<<" sec";
}
```



Làng Gốm thủ công mỹ nghệ có n nghệ nhân. Mỗi nghệ nhân sáng tạo ra rất nhiều mẫu hàng mới giúp hàng hóa của làng chiếm lĩnh thị trường trong và ngoài nước, người thứ i sáng tạo được a_i mẫu hàng mới, $i = 1 \dots n$. Hàng năm Liên hiệp Hợp tác xã Thủ công mỹ nghệ đều tiến hành xét trao giải thưởng Sáng tạo cho các nghệ nhân xuất sắc. Giá trị giải thưởng tỷ lệ với số mẫu hàng mới do nghệ nhân sáng tạo. Để tránh việc bị chảy máu chất xám bởi sự lôi kéo của các làng nghề khác, tập thể nghệ nhân Làng Gốm quyết định sẽ đề xuất một danh sách xét lĩnh giải Sáng tạo, trong đó mỗi người đều trình lên một số lượng mẫu hàng mới như nhau. Mỗi nghệ nhân trong diện đề xuất nhận giải sẽ lấy một số mẫu của người khác trong số những người ngoài danh sách đề xuất thành mẫu sáng tạo của mình trong báo cáo. Số lượng lấy thêm của mỗi người cũng không được vượt quá k để tránh sự xoi mói của phương tiện thông tin đại chúng.

Những người nằm ngoài danh sách đề cử xét giải sẽ được Làng trao giải riêng, tuy nhiên họ vẫn có phần nào bị thiệt thòi về mặt danh tiếng, vì vậy mọi người cố gắng tìm cách để số người thuộc diện này là ít nhất có thể.

Hãy xác định danh sách những người nằm ngoài danh sách đề cử xét giải.

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

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

Kết quả: Đưa ra file văn bản PRIZE.OUT, dòng đầu tiên chứa số nguyên m – số lượng người ngoài danh sách đề cử, dòng thứ 2 chứa m số nguyên – danh sách những người ngoài đề cử.

Ví dụ:

PRIZE.INP	PRIZE.OUT
6 8	3
8 15 38 2 1 25	2 3 6



Giải thuật: Tổng tiền tố, Tìm kiếm nhị phân.

Nhận xét:

Sắp xếp các a_i theo thứ tự không giảm,

Giả thiết đã xác định được một số người đề cử và số sản phẩm cá nhân lớn nhất của những người trong số danh sách này là x , người được chọn tiếp theo có lợi nhất là người có số sản phẩm y , nhỏ hơn hoặc bằng x và gần x nhất, tức là người đứng trước trong danh sách đã sắp xếp,

Như vậy danh sách những người được đề cử tạo thành một khoảng liên tục p phần tử,

Gọi s – tổng số lượng sản phẩm của những người trong danh sách được chọn, z – số sản phẩm của người ít nhất trong đó, ta có $s - z \times p \leq k$,

Lần lượt lấy từng người một trong danh sách làm điểm đầu, bằng phương pháp tìm kiếm nhị phân ta có thể xác định được điểm cuối và từ đó – xác định được danh sách dài nhất.

Tổ chức dữ liệu:

- `vector <pair <int, int> > a(n)` – lưu trữ (a_i, i), $i = 1 \div n$,
- `int64_t sum[100001]` – lưu trữ tổng tiền tố các a_i ,
- `vector <int> res` – lưu trữ danh sách người ngoài đề cử,
- `pair <int, int> ans` – lưu trữ biên của đoạn được chọn.

Xử lý:

Ghi nhận dữ liệu và sắp xếp,

Tính tổng tiền tố các a_i ,

Tìm kiếm đoạn cần chọn:

```

ans.first = 0;
for (int i = 0; i < n; i++)
{
    int l = -1, r = n - i + 1;
    while (l < r - 1)
    {
        int m = (l + r) >> 1;
        if (sum[i+m] - sum[i] - (int64_t)m*a[i].first <= k)
            l = m; else r = m;
    }
    if (l > ans.first)
    {
        ans.first = l;
        ans.second = i;
    }
}

```

Ghi nhận danh sách người ngoài đè cử:

```
vector <int> res;
for (int i = 0; i < n; i++)
{
    if(i>=ans.second && i <= ans.second + ans.first - 1) continue;
    res.push_back(a[i].second);
}
```

Đưa ra kết quả: kích thước vector **res** và các phần tử của **res**.

Dộ phức tạp của giải thuật: O(nlnn).

Chương trình:

```
#include <bits/stdc++.h>
#define NAME "prize."
using namespace std;
ifstream fi (NAME"inp");
ofstream fo (NAME"out");
int64_t sum[100001];

int main()
{
    int n, k;
    fi>>n>>k;

    vector <pair <int, int> > a(n);
    for (int i = 0; i < n; i++)
    {
        fi>>a[i].first;
        a[i].second = i;
    }
    sort(a.begin(), a.end());

    sum[0] = 0;
    for (int i = 1; i <= n; i++)
        sum[i] = sum[i - 1] + a[i - 1].first;

    pair <int, int> ans;
    ans.first = 0;
    for (int i = 0; i < n; i++)
    {
        int l = -1, r = n - i + 1;
        while (l < r - 1)
        {
            int m = (l + r) >> 1;
            if (sum[i + m] - sum[i] - (int64_t) m * a[i].first <= k)
                l = m; else r = m;
        }
        if (l > ans.first)
        {
            ans.first = l;
            ans.second = i;
        }
    }

    vector <int> res;
    for (int i = 0; i < n; i++)
    {
        if (i >= ans.second && i <= ans.second + ans.first - 1) continue;
        res.push_back(a[i].second);
    }

    fo<<res.size()<<'\n';
    sort(res.begin(), res.end());
    for (int i = 0; i < res.size(); i++)
        fo<<res[i] + 1<< ' ';
    fo<<"\nTime: "<<clock() / (double) 1000<< " sec";
    return 0;
}
```



Thành phố có 2 đường vành đai là 2 đường tròn lấy trung tâm thành phố làm tâm, đường thứ nhất có bán kính 10 km, đường thứ 2 – 20 km. Các đường vành đai cho phép lưu thông 2 chiều. Có n đường nối 2 vành đai và chỉ cho phép đi từ vành đai trong ra vành đai ngoài. Mỗi đường nối là một đoạn bán kính. Bán kính chứa đường nối thứ i tạo thành góc angl_i độ với trực OX.

Có q taxi hoạt động. Xe thứ j đón khách ở đường vành đai trong, tại điểm mà bán kính đi qua điểm đó tạo thành góc a_j độ với trực OX và trả khác ở đường vành đai ngoài, tại điểm mà bán kính đi qua điểm đó tạo thành góc b_j độ với trực OX. Các taxi đều đi theo đường ngắn nhất.

Hãy xác định độ dài đường đi của mỗi xe và đưa ra với độ chính xác 10 chữ số sau dấu chấm thập phân.

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

- ✚ Dòng đầu tiên chứa 2 số nguyên n và q ($1 \leq n, q \leq 10^5$),
- ✚ Dòng thứ i trong n dòng sau chứa số thực angl_i ($0 \leq \text{angl}_i < 360$),
- ✚ Dòng thứ j trong q dòng tiếp theo chứa 2 số thực a_j và b_j ($0 \leq a_j, b_j < 360$).

Kết quả: Đưa ra file văn bản .OUT các độ dài đường đi tìm được, mỗi độ dài trên một dòng.

Ví dụ:

CIRCLES.INP
1 2
180
0 0
60 300

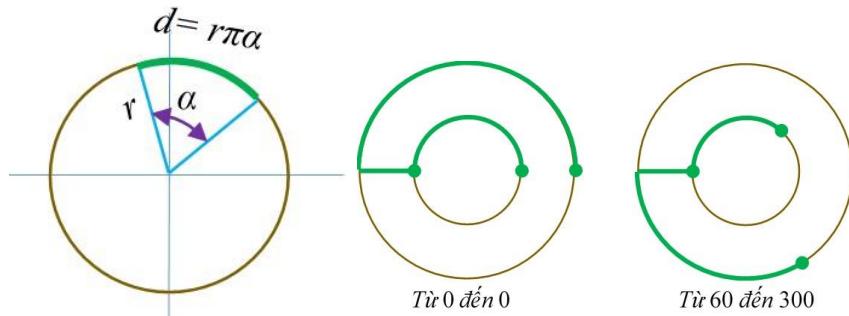
CIRCLES.OUT
104.2477796077
72.8318530718



Giải thuật: Nội dung hình học, Tìm kiếm nhị phân.

Nhận xét:

Độ dài cung α radian của đường tròn bán kính r là $d = r\pi\alpha$,



Tìm hai đường nối gần nhất với điểm đích, đi theo đường trong tới đường nối và từ đó – tới đích,

Tính độ dài các đường đi và chọn đường ngắn hơn,

Tìm hai đường nối gần nhất: áp dụng giải thuật tìm kiếm nhị phân,

Để tránh phải xét đường đi theo 2 chiều (theo kim đồng hồ và ngược chiều kim đồng hồ) cùng với góc α lưu trữ các góc $\alpha+2\pi$ và $\alpha+2\pi$.

Việc tìm kiếm nhị phân có thể thực hiện thông qua hàm **lower_bound** sau khi sắp xếp các góc theo thứ tự tăng dần.

Tổ chức dữ liệu:

Mảng **vector<double>** **arr** – lưu trữ các góc (theo số đo radian).

Độ phức tạp của giải thuật: O(nlgn).

Chương trình:

```
#include <bits/stdc++.h>
#define puba push_back
#define ff first
#define ss second
#define bend(_x) begin(_x), end(_x)
#define szof(_x) ((int) (_x).size())
#define NAME "circles."

using namespace std;
ifstream fi (NAME"inp");
ofstream fo (NAME"out");
typedef int64_t ll;
typedef pair<int, int> pii;
const int INF = 1e9 + 7;
const double PI = atan2(0, -1);
const double PI2 = PI*2;

double dist(double ang1, double ang2)
{
    double tmp = abs(ang1 - ang2);
    while (tmp > PI2) tmp -= PI2;
    double d = min(tmp, PI2 - tmp);
    return d;
}

int main()
{
    int n, q;
    fi>>n>>q;
    vector<double> arr;
    for (int i = 0; i < n; ++i)
    {
        double num;
        fi>>num;
        num = num / 180 * PI;
        arr.push_back(num);
        arr.push_back(num + PI2);
        arr.push_back(num - PI2);
    }
    sort(bend(arr));
    double r1 = 10, r2 = 20;
    for (int i = 0; i < q; ++i)
    {
        double from, to;
        fi>>from>>to;
        from = from / 180 * PI;
        to = to / 180 * PI;
        auto it = lower_bound(bend(arr), to);
        double ans = dist(*it, from) * r1 + r1 + dist(*it, to) * r2;
        --it;
        ans = min(ans, dist(*it, from) * r1 + r1 + dist(*it, to) * r2);
        fo<<fixed<<setprecision(10)<<ans<<'\n';
    }
    fo<<"\nTime: "<<setprecision(4)<<clock() / (double)1000<<" sec";
    return 0;
}
```

{}

VU26. DI CỨ MIGRATION.CPP



Tên chương trình:

Tuyến du lịch trong Khu bảo tồn Quốc gia chạy dọc theo miền có n động vật hoang dã sinh sống. Các động vật thường xuyên di cư, đổi chỗ cho nhau: cả đoàn từ vị trí l_1 đến r_1 chuyển sang sống ở các vị trí từ l_2 đến r_2 và ngược lại, các động vật từ vị trí l_2 đến r_2 chuyển chỗ trống tương ứng từ l_1 đến r_1 , nói cách khác, động vật ở vị trí i đổi chỗ cho động vật ở vị trí $i-l_1+l_2$, $i = l_1 \div r_1$, $r_1-l_1 = r_2-l_2$ và 2 đoạn này không giao nhau. Động vật thứ j có độ nguy hiểm t_j đối với khách du lịch.

Tour du lịch được bố trí khách thăm các vị trí từ l đến r và độ hấp dẫn của tour được xác định bằng số động vật có độ nguy hiểm không nhỏ hơn a và không lớn hơn b ($a \leq b$).

Trong năm có m sự kiện xảy ra, mỗi sự kiện là một lần động vật di cư hoặc có một tour du lịch. Với mỗi tour du lịch hãy xác định độ hấp dẫn của tour.

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

- ✚ Dòng đầu tiên chứa 2 số nguyên n và m ($1 \leq n \leq 10^6$, $1 \leq m \leq 2 \times 10^3$),
- ✚ Dòng thứ 2 chứa n số nguyên t_1, t_2, \dots, t_n ($1 \leq t_j \leq 10^9$, $j = 1 \div n$),
- ✚ Mỗi dòng trong m dòng sau chứa thông tin về một sự kiện và có dạng **1** l_1 r_1 l_2 r_2 cho hiện tượng di cư ($1 \leq l_1 \leq r_1 \leq n$, $1 \leq l_2 \leq r_2 \leq n$, $r_1-l_1=r_2-l_2$) hoặc **2** l r a b cho tour du lịch ($1 \leq l \leq r \leq n$, $1 \leq a \leq b \leq 10^9$).

Kết quả: Đưa ra file văn bản MIGRATION.OUT các số nguyên xác định độ hấp dẫn của mỗi tour du lịch, mỗi số trên một dòng.

Ví dụ:

MIGRATION.INP
6 5
1 2 3 4 5 6
1 1 3 4 6
1 2 3 5 6
2 3 6 2 5
1 1 1 5 5
2 2 5 2 6

MIGRATION.OUT
2
3

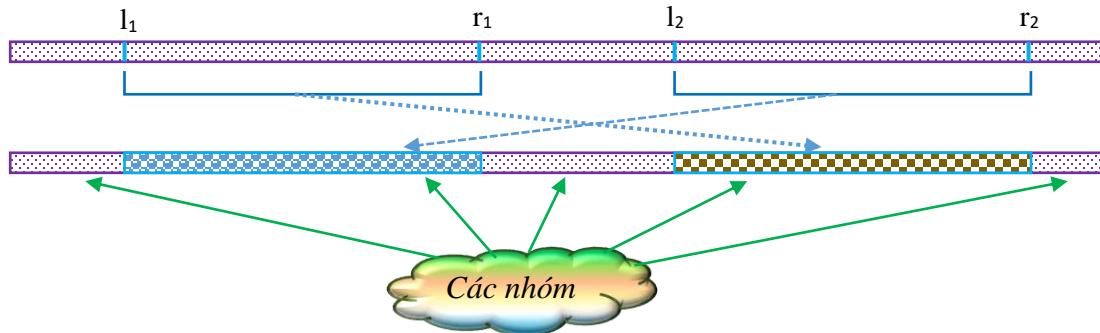


VU26 Io20161126 B

Giải thuật: Quản lý hoán vị, Tìm kiếm nhị phân.

Nhận xét:

Kết quả di cư sẽ chia các động vật thành các nhóm, mỗi nhóm giữ nguyên trình tự như ban đầu:



Mỗi lần di cư sẽ làm phát sinh không quá 4 nhóm mới, ta có thể xác định các nhóm này và quản lý động vật dưới dạng các nhóm,

Mỗi tour du lịch bao gồm một số nguyên nhóm và có thể thêm 2 phần của 2 nhóm (đầu và cuối tour),

Như vậy tổng số nhóm cần quản lý không vượt quá $4 \times m + 1$.

Sắp xếp động vật trong từng nhóm theo thứ tự tăng dần của độ nguy hiểm,

Việc xác định độ hấp dẫn trong từng nhóm có thể dễ dàng thực hiện bằng giải thuật tìm kiếm nhị phân.

Tổ chức dữ liệu:

- ▀ Mảng `vector<int> inp` – lưu độ nguy hiểm của mỗi động vật,
- ▀ Mảng `vector<pii> segs` – lưu thông tin quản lý các nhóm, mỗi phần tử là một cặp dữ liệu xác định các điểm đầu và cuối của nhóm,
- ▀ Mảng `vector<vector<int>> reqs` – lưu các truy vấn,
- ▀ Một số mảng trung gian hỗ trợ việc phân nhóm.

Xử lý:

Đọc tất cả các truy vấn và lần lượt tiến hành phân nhóm dựa vào truy vấn đọc được,

Sắp xếp các nhóm theo điểm đầu và độ nguy hiểm trong từng nhóm,

Xử lý lại các truy vấn:

- ✓ Với truy vấn loại 1: Đổi chỗ các khối,

- ✓ Với truy vấn loại 2: thông qua các hàm `upper_bound` và `lower_bound` tính độ hấp dẫn.

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

Chương trình:

```
#include "bits/stdc++.h"
#define puba push_back
#define ff first
#define ss second
#define bend(_x) begin(_x), end(_x)
#define szof(_x) ((int) (_x).size())
#define NAME "migration."

using namespace std;
ifstream fi (NAME"inp");
ofstream fo (NAME"out");
typedef long long ll;
typedef pair<int, int> pii;
const int INF = 1e9 + 7;
const double PI = atan2(0, -1);

int n, m;
vector<pii> segs;

void split(int l, int r)
{
    int nowl = 0, nowr = 0;
    for (int i = 0; i < szof(segs); ++i)
    {
        auto& s = segs[i];
        nowr = nowl + s.ss - s.ff;
        if (r <= nowl || nowr <= l || (l <= nowl && nowr <= r))
        {
            nowl = nowr;
            continue;
        }
        if (l > nowl)
        {
            int shift = s.ff - nowl;
            s.ss = l + shift;
            if (r < nowr)
            {
                segs.insert(segs.begin() + i + 1, {l + shift, r + shift});
                segs.insert(segs.begin() + i + 2, {r + shift, nowr + shift});
                break;
            } else
            {
                segs.insert(segs.begin() + i + 1, {l + shift, nowr + shift});
                i += 1;
            }
        } else
        {
            int shift = s.ff - nowl;
            s.ss = r + shift;
            segs.insert(segs.begin() + i + 1, {r + shift, nowr + shift});
            i += 1;
        }
        nowl = nowr;
    }

void my_swap(int l1, int r1, int l2, int r2)
```

```

{
    int now = 0;
    int pl1 = -1, pr1 = -1, pl2 = -1, pr2 = -1;

    for (int j = 0; j < szof(segs); ++j)
    {
        if (now == l1) pl1 = j;
        if (now == r1) pr1 = j;
        if (now == l2) pl2 = j;
        if (now == r2) pr2 = j;
        now += segs[j].ss - segs[j].ff;
    }
    if (now == l1) pl1 = szof(segs);
    if (now == r1) pr1 = szof(segs);
    if (now == l2) pl2 = szof(segs);
    if (now == r2) pr2 = szof(segs);
    vector<pii> s1, s2;
    for (int j = pl1; j < pr1; ++j) s1.puba(segs[j]);
    for (int j = pl2; j < pr2; ++j) s2.puba(segs[j]);
    vector<pii> between, after;
    for (int j = pr1; j < pl2; ++j) between.puba(segs[j]);
    for (int j = pr2; j < szof(segs); ++j) after.puba(segs[j]);
    segs.resize(pl1);
    for (pii j : s2) segs.puba(j);
    for (pii j : between) segs.puba(j);
    for (pii j : s1) segs.puba(j);
    for (pii j : after) segs.puba(j);
}

vector<int> inp;

int get(int l, int r, int a, int b)
{
    return upper_bound(inp.begin() + l, inp.begin() + r, b) -
lower_bound(inp.begin() + l, inp.begin() + r, a);
}

int main()
{
    fi>>n>>m;
    for (int i = 0; i < n; ++i)
    {
        int num;
        fi>>num;
        inp.puba(num);
    }

    vector<vector<int>> reqs;
    segs = {{0, n}};
    for (int i = 0; i < m; ++i)
    {
        int type;
        fi>>type;
        if (type == 1)
        {
            int l1, r1, l2, r2;
            fi>>l1>>r1>>l2>>r2;
            --l1; --l2;

            if (l1 > l2)

```

```

    {
        swap(l1, l2);
        swap(r1, r2);
    }

    reqs.pubba({type, l1, r1, l2, r2});

    split(l1, r1);
    split(l2, r2);
    my_swap(l1, r1, l2, r2);
} else
{
    int l, r, a, b;
    fi>>l>>r>>a>>b;
    --l;
    split(l, r);
    reqs.pubba({type, l, r, a, b});
}
}

sort(bend(segs));
for (auto s : segs)
    sort(inp.begin() + s.ff, inp.begin() + s.ss);
for (int i = 0; i < m; ++i)
{
    if (reqs[i][0] == 1)
    {
        int l1 = reqs[i][1], r1=reqs[i][2],l2=reqs[i][3],r2 = reqs[i][4];
        my_swap(l1, r1, l2, r2);
    } else
    {
        int l = reqs[i][1], r=reqs[i][2],a=reqs[i][3],b = reqs[i][4];
        int ans = 0;
        int now = 0;
        int pl = -1, pr = -1;
        for (int j = 0; j < szof(segs); ++j)
        {
            if (now == l)pl = j;
            if (now == r)pr = j;
            now += segs[j].ss - segs[j].ff;
        }
        if (now == l)pl = szof(segs);
        if (now == r)pr = szof(segs);

        for (int j = pl; j<pr;++j)ans+=get(segs[j].ff,segs[j].ss,a, b);
        fo << ans << "\n";
    }
}
fo<<"\nTime: "<<clock() / (double)1000<<" sec";
return 0;
}

```



VU27. Ô ĐỒ CHỮ

Tên chương trình: CROSSW.CPP

Steve phụ trách mục giải trí của một tờ báo địa phương. Để chuẩn bị cho số mới Steve dự định sẽ đưa ra một ô đồ chữ gồm 4 từ, 2 từ đọc theo chiều ngang từ trái sang phải, 2 từ đọc theo chiều dọc từ trên xuống dưới. Các từ ngang nằm ở 2 dòng khác nhau và các từ dọc - ở 2 cột khác nhau. Mỗi từ ngang và từ dọc có một ký tự chung.

Với 4 từ được chọn có thể xây dựng nhiều ô đồ thỏa mãn điều kiện đã nêu. Hai bảng đó gọi là khác nhau nếu không thể bằng cách tịnh tiến toàn bảng để từ một bảng nhận được bảng kia. Ví dụ, với 4 từ **internet**, **ifmo**, **rampage** và **olympiad** ta có thể xây dựng 2 bảng đó:

i	n	t	e	r	n	e	t
f				a			
m				m			
o	l	y	m	p	i	a	d
				a			
				g			
				e			

i	f	m	o				
n			l				
t			y				
e			m				
r	a	m	p	a	g	e	
n			i				
e			a				
t			d				

Trong số báo này, thay vì việc đưa ra ô đồ truyền thống, Steve yêu cầu bạn đọc tự xây dựng các bảng đồ khác nhau và gửi lời giải tới tòa soạn.

Hãy xác định số bảng đồ khác nhau nhiều nhất mà một bạn đọc có thể gửi.

Dữ liệu: Vào từ file văn bản CROSSW.INP gồm 4 dòng, mỗi dòng chứa một từ độ dài lớn hơn 1 và không quá 30 ký tự la tinh thường.

Kết quả: Đưa ra file văn bản CROSSW.OUT một số nguyên – số lượng bảng đồ chữ khác nhau có thể xây dựng.

Ví dụ:

CROSSW.INP
internet
ifmo
rampage
olympiad

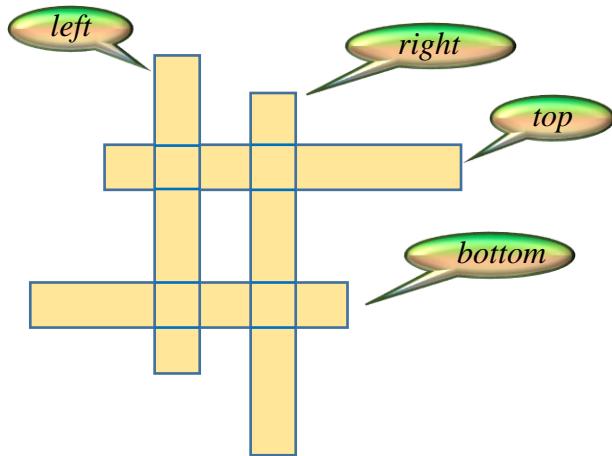
CROSSW.OUT
2



Giải thuật: Duyệt vét cạn.

Nhận xét:

Để tiện xử lý ta gọi 4 xâu trong bảng là **top**, **bottom**, **left** và **right**,



Lần lượt xét mỗi xâu ở một trong 4 vai trò trên, ta có tất cả $4! = 24$ khả năng,

Với độ phức tạp $O(len^2)$ xét các vị trí **left** và **right** giao với **top**,

Với độ phức tạp $O(len)$ để trượt vị trí **bottom** so với **top**,

Với độ phức tạp $O(len)$ tính khoảng cách giữa **bottom** và **top**,

Với độ phức tạp $O(len)$ xét các vị trí tương ứng **left** và **right** giao với **bottom**,

Nếu ở 4 vị trí chung các ký tự của các xâu giao nhau là trùng nhau ta sẽ có một câu hình bảng cần tìm,

Các xâu **left** và **right** di chuyển độc lập vì vậy có thể xét riêng từng xâu và nhân các kết quả nhận được,

Độ phức tạp chung của giải thuật: $O(len^5)$ – chấp nhận được với $len \leq 30$.

Chương trình:

```
#include <bits/stdc++.h>
#define NAME "crossw."
using namespace std;
ifstream fi (NAME"inp");
ofstream fo (NAME"out");
int solve(string &top, string &bottom, string &left, string &right)
{ int ans = 0;
  for (int i = 0; i + 1 < top.size(); i++)
  {
    for (int j = i + 1; j < top.size(); j++)
      for (int shift = j - bottom.size() + 1; shift <= i; shift++)
      {
        char c1 = bottom[i - shift];
        char c2 = bottom[j - shift];
        for (int l = 0; l + 1 < min(left.size(), right.size()); l++)
        {
          int ansl = 0;
          int ansr = 0;
          for (int k = 0; k + l + 1 < left.size(); k++)
          {
            int kk = k + l + 1;
            if (top[i] == left[k] && c1 == left[kk]) ansl++;
          }
          for (int k = 0; k + l + 1 < right.size(); k++)
          {
            int kk = k + l + 1;
            if (top[j] == right[k] && c2 == right[kk]) ansr++;
          }
          ans += ansl * ansr;
        }
      }
    }
  return ans;
}
int main()
{
  vector<string> words(4);
  for (int i = 0; i < 4; i++) fi >> words[i];
  int ans = 0;
  for (int i = 0; i < 4; i++)
    for (int j = i + 1; j < 4; j++)
    {
      vector<int> other(2);
      int c = 0;
      for (int k = 0; k < 4; k++)
        if (k != i && k != j) other[c++] = k;
      int x = other[0];
      int y = other[1];
      ans += solve(words[i], words[j], words[x], words[y]) +
             solve(words[j], words[i], words[x], words[y]) +
             solve(words[i], words[j], words[y], words[x]) +
             solve(words[j], words[i], words[y], words[x]);
    }
  fo << ans << '\n';
  fo<<"\nTime: "<<clock() / (double)1000<<" sec";
  return 0;
}
```

{}

VU28. CƠ HỘI CUỐI CÙNG

trinh: OPPORTUNITY.CPP



Tên chương

Trong một trò chơi điện tử nếu cuối ván người chơi vẫn chưa đạt số điểm cần thiết để chuyển sang ván sau, nhưng số điểm cũng đủ lớn thì có thể có một cơ hội cuối cùng để tăng điểm và qua ván. Hệ thống sẽ đưa ra một dãy n giỏ hàng, giỏ thứ i có a_i đồ vật, $i = 1 \div n$. Người chơi được quyền một lần sắp xếp lại các giỏ hàng bằng cách đánh dấu *một số giỏ hàng liên tiếp cuối cùng* trong dãy. Dãy các giỏ hàng được đánh dấu, giữ nguyên trình tự đã có, sẽ tự động chuyển lên đầu dãy đã cho. Sau đó người chơi chỉ định một giỏ hàng làm điểm xuất phát. Giỏ hàng này sẽ bị gạt sang băng chuyền chờ đi. Tiếp theo, lần lượt giỏ hàng *gần nhất bên phải* giỏ hàng bị chờ đi và có *số lượng hàng lớn hơn* giỏ đã vào băng chuyền ngay trước đó cũng sẽ bị gạt vào băng chuyền. Số lượng các giỏ hàng được chờ đi sẽ được cộng vào số điểm của người chơi.

Ví dụ, với dãy các giỏ hàng 2, 4, 1, 3, 3 người chơi có thể đánh dấu 3 giỏ cuối và nhận được dãy giỏ hàng mới là 1, 3, 3, 2, 4, sau đó chỉ định điểm xuất phát là giỏ hàng đầu tiên thì sẽ nhận thêm một số điểm bổ sung là 3: 1, 3, 3, 2, 4.

Hãy xác định số điểm bổ sung lớn nhất người chơi có thể nhận.

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

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

Kết quả: Đưa ra file văn bản OPPORTUNITY.OUT một số nguyên – số điểm bổ sung lớn nhất người chơi có thể nhận.

Ví dụ:

OPPORTUNITY.INP
5
2 4 1 3 3

OPPORTUNITY.OUT
3



VU28.l020161126.G

Giải thuật: Dữ liệu vòng tròn, Kỹ thuật 2 con trỏ (Hàng đợi hai đầu – Deque)..

Nhận xét:

Việc mang một đoạn cuối của dãy **a** lên đầu tương ứng với việc đẩy vòng tròn dãy số,

Việc lưu trữ liên tiếp 2 lần dãy **a** sẽ chứa thông tin toàn bộ các phép đẩy vòng,

Tính số điểm nhận được lần lượt với các điểm đầu là **a_{2n}**, **a_{2n-1}**, ...

Giả thiết đã xét điểm đầu là **a_{i+1}**, chỉ số các phần tử bị đẩy vào băng chuyền được lưu ở các phần tử **b_p**, **b_{p+1}**, ..., **b_q** (hàng đợi **b**),

Chuyển sang xét điểm đầu là phần tử **a_i**:

- ✚ Với các phần tử **a[b_j] ≤ a_i – b_j** bị loại khỏi hàng đợi **b**, **j = p, p-1, ...**
- ✚ Nếu **b_k – i > n → b_k** bị loại khỏi hàng đợi **b**, **k = q, q+1, ...** (khoảng cách 2 giở không quá **n**),
- ✚ Bổ sung **i** vào hàng đợi (**b[++p]=i**),
- ✚ Số điểm nhận được sẽ là **p-q+1**,
- ✚ Cập nhật kết quả tối ưu.

Việc lập trình sẽ đơn giản hơn đôi chút nếu dùng hàng đợi 2 đầu (**deque**), nhưng thời gian thực hiện sẽ lớn hơn việc trực tiếp quản lý **b** bằng 2 con trỏ **p** và **q**.

Tổ chức dữ liệu:

- ❑ Mảng **vector<int> a** – lưu dữ liệu vòng tròn,
- ❑ Mảng **vector<int> b** – tổ chức hàng đợi.

Độ phức tạp của giải thuật: O(n).

Chương trình:

```
#include "bits/stdc++.h"
#define puba push_back
#define ff first
#define ss second
#define bend(_x) begin(_x), end(_x)
#define szof(_x) ((int)(_x).size())
#define NAME "opportunity."

using namespace std;
ifstream fi (NAME"inp");
ofstream fo (NAME"out");
typedef long long ll;
typedef pair<int, int> pii;
vector<int> a;
vector<pii> b;
int n,n2,ans=1,p=0,q=0,t;

int main()
{
    fi>>n; n2=n<<1;
    a.resize(n2); b.resize(n2+1);
    for(int i=0;i<n;++i)
    {
        fi>>t;
        a[i]=t;a[n+i]=t;
    }
    b[p]={a[n2-1],n2-1};
    for(int i=n2-2;i>=0;--i)
    {
        while(p>=q && b[p].ff<=a[i]) --p;
        b[++p]={a[i],i};
        while(q<p && b[q].ss-i>=n) ++q;
        if(ans<p-q+1) ans=p-q+1;
    }
    fo << ans << "\n";
    fo<<"\nTime: " << clock() / (double)1000 << " sec";
    return 0;
}
```



Ông kính của Huble được quay về phía vốn được coi là vùng trống của vũ trụ và thật ngạc nhiên, các ảnh chụp cho thấy ở đó cũng dày đặc các ngôi sao, tạo thành vô số ngân hà với đủ loại hình dáng, trước đây không quan sát được vì chúng ở quá xa.

Các nhà nghiên cứu nghiệp dư khai thác các khía cạnh thông tin khác nhau từ những bức ảnh do NASA cung cấp. Một số người muốn tìm biểu tượng chữ thập trong vũ trụ.

Trên ảnh kích thước $n \times m$, mỗi ngôi sao hoặc cụm sao tạo thành một điểm sáng, đánh dấu là 1, vùng tối – đánh dấu 0. Chữ thập kích thước $4k+1$ với tâm ở ô tọa độ (x, y) là vùng mà hàng và cột đi qua ô (x, y) có $2k+1$ ô 1 liên tiếp nhau và nhận (x, y) làm ô giữa của miền liên tiếp. Có thể k bằng 0.

Hãy xác định kích thước và tâm của hình chữ nhật kích thước lớn nhất.

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

- ⊕ Dòng đầu tiên chứa 2 số nguyên n và m ($1 \leq n, m \leq 5\,000$),
- ⊕ Mỗi dòng sau n dòng sau chứa xâu các ký tự 0, 1 kích thước m .

Kết quả: Đưa ra file văn bản PLUSES.OUT số -1 nếu không tồn tại chữ thập, trong trường hợp tồn tại đưa ra một số nguyên – kích thước lớn nhất tìm được, ở dòng tiếp theo – 2 số nguyên xác định tâm của hình chữ nhật. Nếu tồn tại nhiều hình thì đưa ra tọa độ (x, y) có thứ tự từ điển nhỏ nhất trong số các tọa độ thỏa mãn điều kiện tìm kiếm. Dòng và cột được đánh số bắt đầu từ 1.

Ví dụ:

PLUSES.INP	PLUSES.OUT
<pre>6 7 0000000 0001000 0001000 0111110 0001000 0001000</pre>	<pre>9 4 4</pre>



VU29 Io20161126 I

Giải thuật: Thống kê đơn giản..

Nhận xét:

Duyệt theo hàng từ trên xuống dưới và từ dưới lên trên ta có thể tính được số lượng 1 liên tiếp theo cột kết thúc ở ô (*i*, *j*) theo chiều duyệt,

Tương tự, duyệt các cột từ trái sang phải và ngược lại ta có thể tính được số lượng 1 liên tiếp theo hàng kết thúc ở ô (*i*, *j*) theo chiều duyệt,

Có thể giảm số lần duyệt: chỉ cần duyệt từ trên xuống dưới và từ trái qua phải, nhưng thời gian xử lý sẽ giảm không đáng kể và độ phức tạp lập trình sẽ cao hơn,

Với các ô (*i*, *j*) chứa 1 có thể dễ dàng tính kích thước hình chữ thập nhận (*i*, *j*) làm tâm, từ đó cập nhật kết quả tối ưu,

Để đảm bảo có kết quả với thứ tự từ điển của tọa độ tâm nhỏ nhất cần ưu tiên duyệt theo hàng và chỉ cập nhật kết quả khi có kích thước mới lớn hơn kết quả hiện có.

Không tồn tại chữ thập khi và chỉ khi bảng chứa toàn số 0.

Tổ chức dữ liệu:

- ▀ Mảng **bool a[MAX][MAX]** – đánh dấu 0, 1 cho bởi dữ liệu vào,
- ▀ Các mảng **short up_[MAX][MAX], down_[MAX][MAX], left_[MAX][MAX], right_[MAX][MAX]** – lưu kết quả đoạn 1 liên tục theo các chiều duyệt.

Độ phức tạp của giải thuật: O(n×m).

Chương trình:

```
#include "bits/stdc++.h"
#define puba push_back
#define ff first
#define ss second
#define bend(_x) begin(_x), end(_x)
#define szof(_x) ((int)(_x).size())
#define NAME "opportunity."

using namespace std;
ifstream fi (NAME"inp");
ofstream fo (NAME"out");
typedef long long ll;
typedef pair<int, int> pii;
vector<int> a;
vector<pii> b;
int n,n2,ans=1,p=0,q=0,t;

int main()
{
    fi>>n; n2=n<<1;
    a.resize(n2); b.resize(n2+1);
    for(int i=0;i<n;++i)
    {
        fi>>t;
        a[i]=t;a[n+i]=t;
    }
    b[p]={a[n2-1],n2-1};
    for(int i=n2-2;i>=0;--i)
    {
        while(p>=q && b[p].ff<=a[i]) --p;
        b[++p]={a[i],i};
        while(q<p && b[q].ss-i>=n) ++q;
        if(ans<p-q+1) ans=p-q+1;
    }
    fo << ans << "\n";
    fo<<"\nTime: " << clock() / (double)1000 << " sec";
    return 0;
}
```



VU30. LÀNG CỔ

Tên chương trình: VILLAGE.CPP

Khu làng cổ có n ngôi nhà độc đáo, mỗi nhà có không quá một lối ra, từ nhà i có lối ra tới nhà a_i , $i = 1 \dots n$. Qua nhiều lần cải tạo, đôi khi lối ra từ một nhà lại dẫn tới chính nhà đó. Để phục vụ mục đích du lịch người ta quyết định sửa lại lối đi sao cho từ ngôi nhà 1 khách du lịch đi theo lối ra có thể lần lượt đến tất cả các nhà và quay trở lại ngôi nhà ban đầu. Không muốn thay đổi quá nhiều cảnh quan hiện tại, Ủy ban Bảo tồn văn hóa cổ cho phép chỉ thay đổi lối ra từ đúng một ngôi nhà.

Hãy xác định ngôi nhà và lối ra dẫn đến nhà mới cần có để thỏa mãn điều kiện đã nêu.

Dữ liệu: Vào từ file văn bản VILLAGE.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 ($1 \leq a_i \leq n$, $i = 1 \dots n$).

Kết quả: Đưa ra file văn bản VILLAGE.OUT đưa ra trên mtj dòng 2 số nguyên xác định nhà và lối ra mới. Trong trường hợp không cần sửa lại đường đi hoặc không thể sửa một đường đi để đáp ứng yêu cầu đã nêu thì đưa ra 2 số -1.

Ví dụ:

VILLAGE.INP	VILLAGE.OUT
3	1 2
1 3 1	



VU30 Io20161126 J

Giải thuật: Tìm chu trình.

Nhận xét:

Dễ dàng nhận thấy rằng chỉ khi có đúng một ngôi nhà không có đường dẫn tới,
Bằng phương pháp đánh dấu có thể xác định những ngôi nhà không có đường dẫn
đến,

Xuất phát từ ngôi nhà không có đường đến lần lượt chuyển sang n-1 nhà khác,

Nếu trên đường đi gặp lại một nhà đã thăm hoặc không còn đường đi tiếp – bài
tính vô nghiệm,

Mốc nối nhà tới cuối cùng với nhà xuất phát ta có nghiệm cần tìm.

Tổ chức dữ liệu:

- Mảng `vector<int> a` – lưu mốc nối ban đầu,
- Mảng `bool is[N]={0}` – đánh dấu nhà có mốc nối tới,
- Mảng `bool us[N]={0}` – đánh dấu nhà trong quá trình duyệt xác định chu
trình.

Độ phức tạp của giải thuật: O(n).

Chương trình:

```
//#pragma comment(linker, "/STACK:667772160")
#include <bits/stdc++.h>
using namespace std;
#define name "village."
#define forn(i, n) for(int i = 0; i < (int)n; i++)
#define fornn(i, q, n) for(ll i = (ll)q; i < (ll)n; i++)
#define times clock() * 1.0 / CLOCKS_PER_SEC

ifstream fi ("name.inp");
ofstream fo ("name.out");

const int N = 1e5 + 1;

vector<int> a;
bool us[N]={0}, is[N]={0};

int main()
{
    int n;
    fi >> n;
    a.resize(n);
    forn(i, n)
    {
        fi >> a[i];
        a[i]--;
        is[a[i]] = 1;
    }
    int v = -1, s, r = -1;
    forn(i, n)
        if(!is[i]) {v = i; break;}
    s = v;
    if (v == -1)
    {
        fo << "-1 -1";
        return 0;
    }

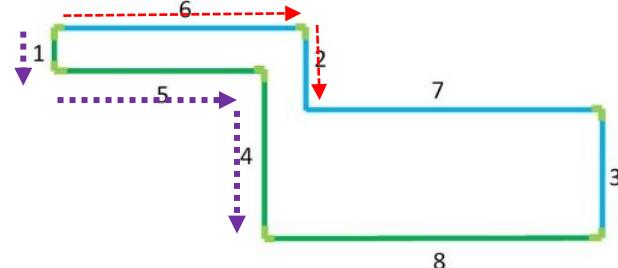
    forn(i, n)
    {
        if(us[v])
        {
            fo << "-1 -1";
            return 0;
        }
        us[v] = 1;
        r = v;
        v = a[v];
    }
    fo << r + 1 << " " << s + 1;
    fo<<"\nTime: "<<clock()/(double)1000<<" sec";
}
```



VU31. KHUNG BIỂN QUẢNG CÁO

Tên chương trình: FRAME.CPP

Để tăng hiệu ứng cho bảng quảng cáo khai trương cửa hàng người quản lý mang ra một bộ đèn quảng cáo, mỗi cái đèn có dạng óng dài. Bộ đèn gồm n chiếc, có độ dài khác nhau từ 1 đến n . Ông phân công cho 2 nhóm thợ dùng các đèn lắp thành một khung sáng khép kín có cạnh không tự cắt, mỗi đoạn của khung phải song song với cạnh hình chữ nhật của biển quảng cáo, mỗi đoạn gồm một hoặc một số đèn. Dĩ nhiên không thể uốn cong hoặc bẻ gãy lấy một phần của đèn. Để mau chóng hoàn thành, 2 nhóm thợ phải thi công song song. Xuất phát từ góc trên trái, nhóm thứ nhất lắp các đèn theo đường ngang từ trái sang phải, sau đó lắp đèn chạy dọc tiếp từ trên xuống dưới, rồi tiếp đến – chạy ngang sang phải, . . . Cũng xuất phát từ góc trên trái, nhóm thứ 2 măc đèn dọc xuống dưới, sau đó sang ngang từ trái sang phải, rồi xuống dưới, .. cứ như thế cho đèn khi 2 nhóm gặp nhau ở một điểm. Hai nhóm phải chia nhau sử dụng đèn để có khung đèn độ dài lớn nhất có thể. Ví dụ, với bộ 8 đèn, độ dài của toàn khung là 36 với một trong số các cách măc nêu ở hình trên.



Với n cho trước, hãy xác định độ dài lớn nhất có thể của khung đèn. Nếu không tồn tại cách lắp đèn thì đưa ra số 0.

Dữ liệu: Vào từ file văn bản FRAME.INP gồm một dòng chứa số nguyên n ($1 \leq n \leq 10^9$).

Kết quả: Đưa ra file văn bản FRAME.OUT một số nguyên – độ dài lớn nhất tìm được.

Ví dụ:

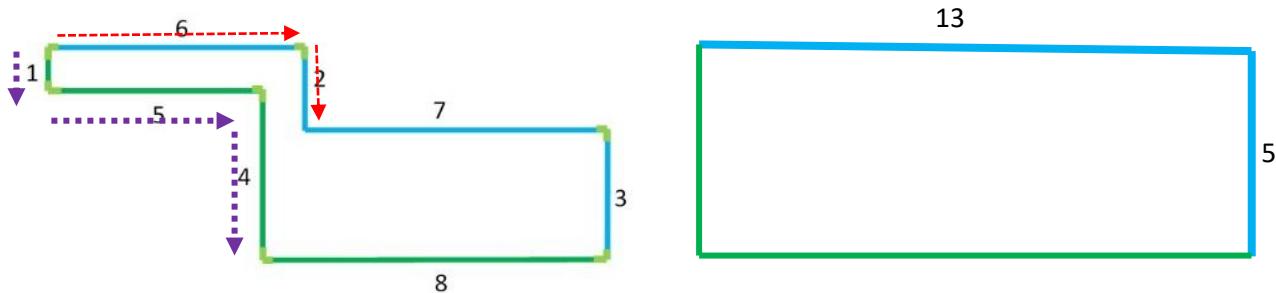
FRAME.INP	FRAME.OUT
8	36



Giải thuật: Phân tích, đoán nhận giải thuật.

Nhận xét:

Việc tồn tại 2 đường bậc thang xây dựng theo kiểu đã nêu tương đương với việc tồn tại hình chữ nhật cùng chu vi:



Với $n < 7$: không tồn tại cách xây dựng hình chữ nhật,

Với $n \geq 7$: phân biệt các trường hợp:

- ⊕ $n = 4k$ – một trong số các cách xác định cạnh hình chữ nhật:
 - Hai cạnh đối có kích thước $\{1, 4\}$ và $\{3, 2\}$,
 - Hai cạnh kia: $\{5, 8, 9, 12, 13, 16, \dots\}$ và $\{6, 7, 10, 11, 14, 15, \dots\}$,
- ⊕ $n = 4k+1$: tổng độ dài tất cả các đèn lè vì vậy không dùng đèn độ dài 1 và một trong số các cách xác định cạnh hình chữ nhật:
 - Hai cạnh đối có kích thước $\{2, 5\}$ và $\{3, 4\}$
 - Hai cạnh kia: $\{5, 8, 9, 12, 13, 16, \dots\}$ và $\{6, 7, 10, 11, 14, 15, \dots\}$,
- ⊕ $n = 4k+2$: tổng độ dài tất cả các đèn lè vì vậy không dùng đèn độ dài 1 và một trong số các cách xác định cạnh hình chữ nhật:
 - Hai cạnh đối có kích thước $\{2, 3, 5\}$ và $\{4, 6\}$
 - Hai cạnh kia: $\{7, 10, 11, 14, 15, 18, \dots\}$ và $\{8, 9, 12, 13, 16, 17, \dots\}$,
- ⊕ $n = 4k+3$: có thể sử dụng hết các đèn và một trong số các cách xác định cạnh hình chữ nhật:
 - Hai cạnh đối có kích thước $\{1, 2\}$ và $\{3\}$,
 - Hai cạnh kia: $\{4, 7, 8, 11, 12, 15, \dots\}$ và $\{5, 6, 9, 10, 13, 14, \dots\}$.

Độ phức tạp của giải thuật: $O(1)$.

Chương trình:

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

int main()
{
    int64_t n;
    fi >> n;
    if (n < 7)
    {
        fo << "0\n";
        return 0;
    }

    long long sum = n * (n + 1) / 2;
    if (n % 4 == 3 || n % 4 == 0)
    {
        fo << sum;
        return 0;
    }
    if (n % 4 == 2 || n % 4 == 1)
        sum -= 1;

    fo << sum;
    return 0;
}
```



Để bạn đọc biết được độ tin cậy của thông tin, các bài đăng trên một trang WEB được gắn kèm hệ số tin cậy dưới dạng số nguyên. Thông tin được kiểm tra từ n nguồn tin độc lập, đánh số từ 1 đến n . Thông tin có hệ số tin cậy k có nghĩa là nó được kiểm chứng từ các nguồn tin i và j , trong đó $1 \leq i, j \leq n$, $i < j$ và $i+j=k$. Các cặp nguồn kiểm chứng càng nhiều thì độ tin cậy của thông tin càng cao. Ví dụ, với $n = 5$ và $k = 3$ thì độ tin cậy của thông tin sẽ là 1 vì chỉ có một cặp $1 + 2 = 3$.

Cho n và k . Hãy xác định độ tin cậy của thông tin.

Dữ liệu: Vào từ file văn bản SOURCES.INP gồm một dòng chứa 2 số nguyên n và k ($1 \leq n, k \leq 10^{15}$).

Kết quả: Đưa ra file văn bản SOURCES.OUT một số nguyên – độ tin cậy của thông tin.

Ví dụ:

SOURCES.INP	SOURCES.OUT
5 3	1



VU32 Io20170128 A

Giải thuật: Phân tích, đoán nhận giải thuật.

Nhận xét:

Xét cặp số i, j ($i < j, 1 \leq i, j \leq n$) thỏa mãn điều kiện $i + j = k$, ta thấy:

- ❖ $i < (k-1)/2$,
- ❖ $j > (k-1)/2$.

Số lượng i có thể chọn là $a = (k-1)/2$,

Số lượng j có thể chọn là $b = n - k/2$,

Với mỗi i phải có một j tương ứng và ngược lại, vì vậy kết quả cần tìm là

$$\min\{a, b\}$$

Lưu ý: giá trị $n - k/2$ có thể âm, khi đó có $b = 0$ (không tồn tại j).

Độ phức tạp của giải thuật: $O(1)$.

Chương trình:

```
#include <bits/stdc++.h>
#define NAME "sources."
using namespace std;
ifstream fi (NAME"inp");
ofstream fo (NAME"out");
int64_t n, k, b, ans;
int main()
{
    fi>>n>>k;
    b=n-k/2; if (b<0) b=0;
    ans=min((k-1)/2,b);
    fo << ans << endl;
    return 0;
}
```



Để tăng độ an toàn chống hiện tượng cướp ngân hàng ngày càng phổ biến người ta dùng khóa số với mã mở khóa đơn giản nhưng rất hiệu quả. Trên cửa ra vào hiển thị một xâu khá dài các ký tự số. Các chữ số có thể di chuyển đổi chỗ cho nhau hoặc bị xóa. Muốn mở khóa người ta phải di chuyển các chữ số và trong trường hợp cần thiết – xóa vài chữ số để nhận được *xâu lớn nhất* thỏa mãn điều kiện đã cài đặt. Điều kiện này được thay đổi thường xuyên. Hôm nay điều kiện đó là “Số nhận được phải chia hết cho 3”. Số nhận được có thể bắt đầu bằng các chữ số 0. Xâu “000” sẽ lớn hơn xâu “00”.

Hãy xác định khóa mở cửa.

Dữ liệu: Vào từ file văn bản MAXSTR.INP gồm một xâu ký tự số có độ dài lớn hơn 2 và không vượt quá 10^5 .

Kết quả: Đưa ra file văn bản MAXSTR.OUT xâu khóa mở cửa.

Ví dụ:

MAXSTR.INP
105

MAXSTR.OUT
510



Giải thuật: Thống kê tần số.

Nhận xét:

Các việc đổi chỗ để nhận được kết quả theo một tính chất nào đó thường có thể thực hiện một cách có hiệu quả thông qua bảng tần số các đối tượng đổi chỗ,

Từ xâu **s** có thể xây dựng bảng tần số xuất hiện các chữ số: d_i – số lần xuất hiện chữ số **i** trong **s**, $i = 0 \div 9$.

Gọi **sum** là phần dư chia cho 3 của tổng các chữ số.

Nếu **sum** bằng 0 ta không phải xóa chữ số nào,

Nếu **sum** khác 0 ta phải xóa một hoặc 2 chữ số khác 0 để đưa **sum** về 0. Việc xóa 2 chữ số sẽ làm cho kết quả nhận được nhỏ hơn khi xóa một chữ số, vì vậy chỉ xóa 2 chữ số khi không thể đưa **sum** về 0 bằng cách xóa một chữ số.

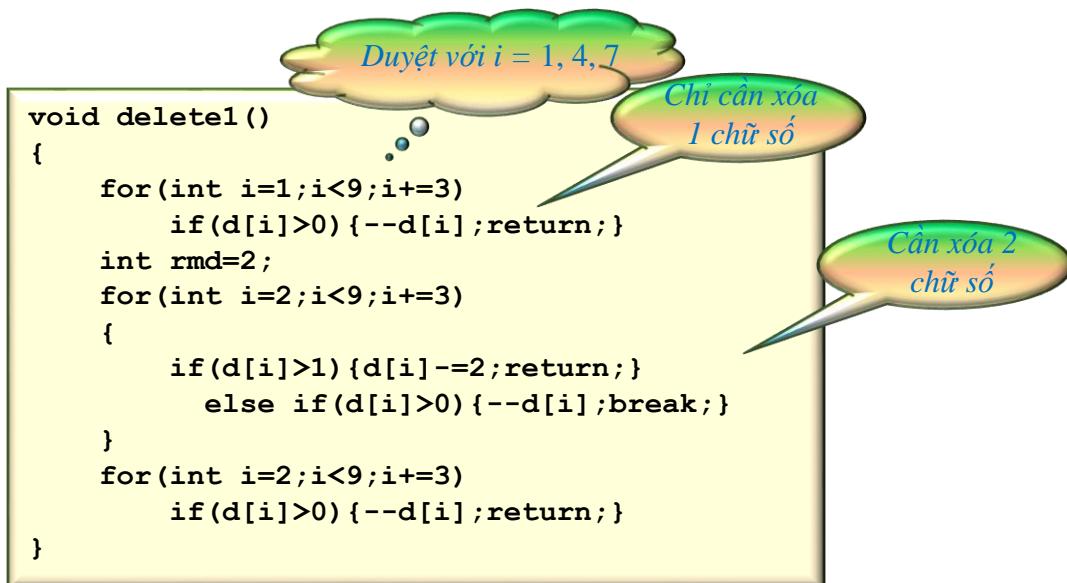
Đưa ra kết quả: dựa vào bảng tần số, đưa ra từ các chữ số lớn đến bé. Số sẽ bắt đầu bằng 0 nếu chỉ còn lại các chữ số 0.

Tổ chức dữ liệu:

Mảng **int d[10]={0}** – lưu tần số xuất hiện các chữ số.

Xử lý:

Hàm **void delete1()** xử lý trường hợp **sum = 1**:



Hàm **void delete2()** xử lý trường hợp **sum = 2**: Xử lý tương tự.

Độ phức tạp của giải thuật: $O(n)$.

Chương trình:

```
#include <bits/stdc++.h>
#define NAME "maxstr."
using namespace std;
ifstream fi (NAME"inp");
ofstream fo (NAME"out");
string s;
int n,sum=0,t,d[10]={0};

void delete1()
{
    for(int i=1;i<9;i+=3)
        if(d[i]>0){--d[i];return;}
    int rmd=2;
    for(int i=2;i<9;i+=3)
    {
        if(d[i]>1){d[i]-=2;return;}
        else if(d[i]>0){--d[i];break;}
    }
    for(int i=2;i<9;i+=3)
        if(d[i]>0){--d[i];return;}
}

void delete2()
{
    for(int i=2;i<9;i+=3)
        if(d[i]>0){--d[i];return;}
    int rmd=2;
    for(int i=1;i<9;i+=3)
    {
        if(d[i]>1){d[i]-=2;return;}
        else if(d[i]>0){--d[i];break;}
    }
    for(int i=1;i<9;i+=3)
        if(d[i]>0){--d[i];return;}
}

int main()
{
    fi>>s; n=s.size();
    for(int i=0;i<n;++i)
    {
        t=s[i]-48;
        d[t]++;
    }
    for(int i=0;i<=9;++i) sum+=d[i]*i;
    sum%=3;
    if(sum==2) delete2();else if(sum==1) delete1();
    for(int i=9;i>=0;--i)
        if(d[i]>0) for(int j=0;j<d[i];++j) fo<<i;
    fo<<"\nTime: "<<clock() / (double)1000<<" sec";
}
```



VU34. MÃ SỐ

Tên chương trình: SAFEKEY.CPP

Tủ đựng hồ sơ mật lưu trữ tài liệu về UFO (*Unidentified Flying Object*) được bảo vệ bằng khóa điện tử. Người khóa nhập vào bộ nhớ của tủ hồ sơ 2 số nguyên $n1$ và nr ($n1 \leq nr$). Muốn mở khóa phải nhập vào số lượng số “*dạng cấp số cộng*” trong khoảng $[n1, nr]$.

Số $\mathbf{A} = a_{n-1}a_{n-2}\dots a_1a_0$ được gọi là có dạng cấp số cộng nếu các chữ số của nó tạo thành cấp số cộng theo mô đun 10, tức là tồn tại d nguyên sao cho $a_1 = (a_0 + d) \% 10$, $a_2 = (a_1 + d) \% 10$, . . .

Với $n1$ và nr cho trước hãy xác định khóa mở tủ hồ sơ.

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

- ✚ Dòng đầu tiên chứa một số nguyên $n1$ ($1 \leq n1 \leq 10^{10^5}$),
- ✚ Dòng thứ 2 chứa số nguyên nr ($n1 \leq nr \leq 10^{10^5}$).

Kết quả: Đưa ra file văn bản SAFEKEY.OUT một số nguyên – mã mở tủ hồ sơ.

Ví dụ:

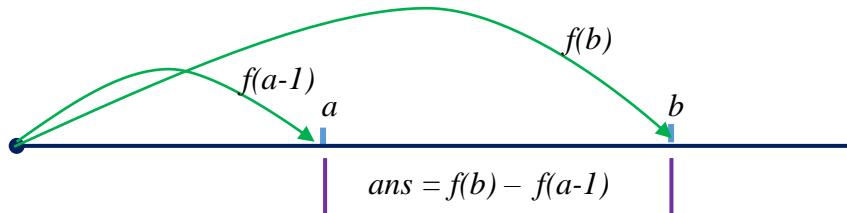
SAFEKEY.INP	SAFEKEY.OUT
90	9
110	11



Giải thuật: Tổng tiền tố.

Nhận xét:

Việc thống kê các đối tượng trong một khoảng liên tục $[a, b]$ thường dựa trên hàm tổng tiền tố $f(x)$ xác định số lượng đối tượng thỏa mãn điều kiện cần tìm trong khoảng $[0, x]$, khi đó kết quả cần tìm sẽ là $f(b) - f(a-1)$.



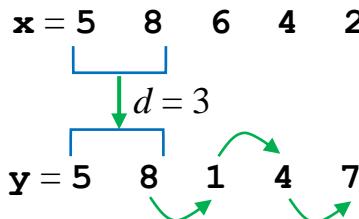
Để tìm các số dạng cấp số cộng chỉ xét các công bội $d = 0, 1, 2, \dots, 8, 9$,

Dễ dàng thấy rằng tất cả các số có 1 chữ số đều là số dạng cấp số cộng (có tất cả 10 số từ 0 đến 9),

Xét số có đúng k ($k > 1$) chữ số bắt đầu bằng 1: với các d khác nhau ($d = 0, 1, \dots, 9$) ta có 10 số khác nhau dạng cấp số cộng, tương tự như vậy với số bắt đầu bằng 2, bằng 3, ...

Như vậy trong số các số có đúng k chữ số ($k > 1$) có 90 số dạng cấp số cộng.

Từ 2 chữ số *trái nhất* của số x có k chữ số ta có thể xác định d và từ đó – tìm được số y dạng cấp số cộng có k chữ số và bắt đầu bằng 2 chữ số của số đã cho, ví dụ $x = 58642$, ta có $y = 58147$.



Có 2 trường hợp: $x \geq y$ hoặc $x < y$.

Cách tính hàm $f(x)$ với $x = x_0x_1\dots x_{k-1}$:

Với $k = 1$: có x_0+1 số,

Với $k > 1$: $f(x)$ bao gồm:

- ▣ Các số dạng cấp số cộng có ít hơn k chữ số: $10 + 90 \times (k-2)$,
- ▣ Các số dạng cấp số cộng có k chữ số và chữ số đầu nhỏ hơn x_0 : $(x_0-1) \times 10$,

- ▣ Các số dạng cấp số cộng có **k** chữ số bắt đầu bằng x_0 và chữ số tiếp theo nhỏ hơn x_1 : x_1 số,
- ▣ Thêm **1** số nếu $x \geq y$.

Dữ liệu vào cần lưu trữ dưới dạng xâu (có tới 5 000 chữ số),

Cần có hàm tính xâu **n1-1**, cần lưu ý xóa số 0 không có nghĩa có thể xuất hiện!

Tổ chức dữ liệu: Không cần lưu trữ bảng giá trị phục vụ tính $f(x)$ như ở phần lớn các bài toán sử dụng tổng tiền tố vì ở đây $f(x)$ có thể dẫn xuất theo công thức.

Độ phức tạp của giải thuật: $O(n)$, trong đó n – độ dài xâu số của dữ liệu vào.

Chương trình:

```
#include<bits/stdc++.h>
#define NAME "safekey."
using namespace std;
ifstream fi (NAME"inp");
ofstream fo (NAME"out");
string sl,sr;
int ans,Last,n;

int calc(string &s)
{int d,t=10;
 n=s.size(); Last=1;
 if(n==1) return s[0]-47;
 t+=90*(n-2)+(s[0]-49)*10+(s[1]-48);
 d=(s[1]-s[0]+10)%10;
 for(int i=1;i<n;++i)
 {
     if(s[i]-48<(s[i-1]-48+d)%10) {Last=0;break;}
     if(s[i]-48>(s[i-1]-48+d)%10) break;
 }
 
 return t+Last;
}

void decr()
{
    n = sl.size();--n;
    while (sl[n] == '0')
    {
        sl[n] = '9';
        --n;
    }
    --sl[n];
    if (n>0 && sl[0] == '0') sl = sl.erase(0,1);
}
int main()
{
    fi>>sl>>sr;
    decr();
    ans=calc(sr)-calc(sl);
    fo<<ans;
    fo<<"\nTime: "<<clock() / (double)1000<<" sec";
}
```



Quy hoạch động

Nhiều bài toán tối ưu có thể giải theo sơ đồ sau:

- ✚ Phân rã bài toán ban đầu thành k bài toán con tương tự như bài toán ban đầu nhưng có kích thước nhỏ hơn,
- ✚ Kết quả cần tìm của bài toán ban đầu được tổng hợp từ kết quả của mỗi bài toán con,
- ✚ Với mỗi bài toán nhận được: lặp lại hai bước trên,
- ✚ Quá trình phân rã được thực hiện cho đến khi nhận được bài toán con đủ đơn giản, có thể dễ dàng tìm ra lời giải.

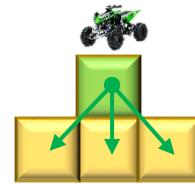
Nguyên lý Bellman

Xét một số bài toán ví dụ.

Bài 1a. VỀ ĐÍCH

Tên chương trình: FINISH.CPP

Chặng cuối cùng trước khi cán đích của một cuộc đua xe địa hình là một vùng chướng ngại hình chữ nhật kích thước $n \times m$ ô (n hàng và m cột). Các tay đua có thể và chỉ có thể vào ô bất kỳ ở hàng đầu tiên và ra khỏi vùng chướng ngại từ ô bất kỳ ở hàng cuối cùng. Xe không được ra khỏi vùng chướng ngại trước khi tới được hàng cuối cùng. Từ một ô (i, j) xe có thể di chuyển sang một trong số các ô $(i+1, j-1), (i+1, j), (i+1, j+1)$ nếu ô đó tồn tại. Thời gian vượt qua ô (i, j) là $a_{i,j}$, $i = 1 \div n$, $j = 1 \div m$.



Hãy xác định thời gian nhỏ nhất một tay đua có thể vượt qua vùng chướng ngại.

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

- ✚ Dòng đầu tiên chứa 2 số nguyên n và m ($1 \leq n, m \leq 2000$, $n \times m \leq 3 \times 10^6$),
- ✚ Dòng thứ i trong n dòng sau chứa m số nguyên $a_{i,1}, a_{i,2}, \dots, a_{i,m}$ ($0 \leq a_{i,j} \leq 10^9$, $j = 1 \div m$).

Kết quả: Đưa ra file văn bản FINISH.OUT một số nguyên – thời gian nhỏ nhất tìm được.

Ví dụ:

FINISH.INP
3 4
6 5 1 3
2 8 7 8
9 4 6 4

FINISH.OUT
11



Phân tích:

Các khái niệm cơ bản:

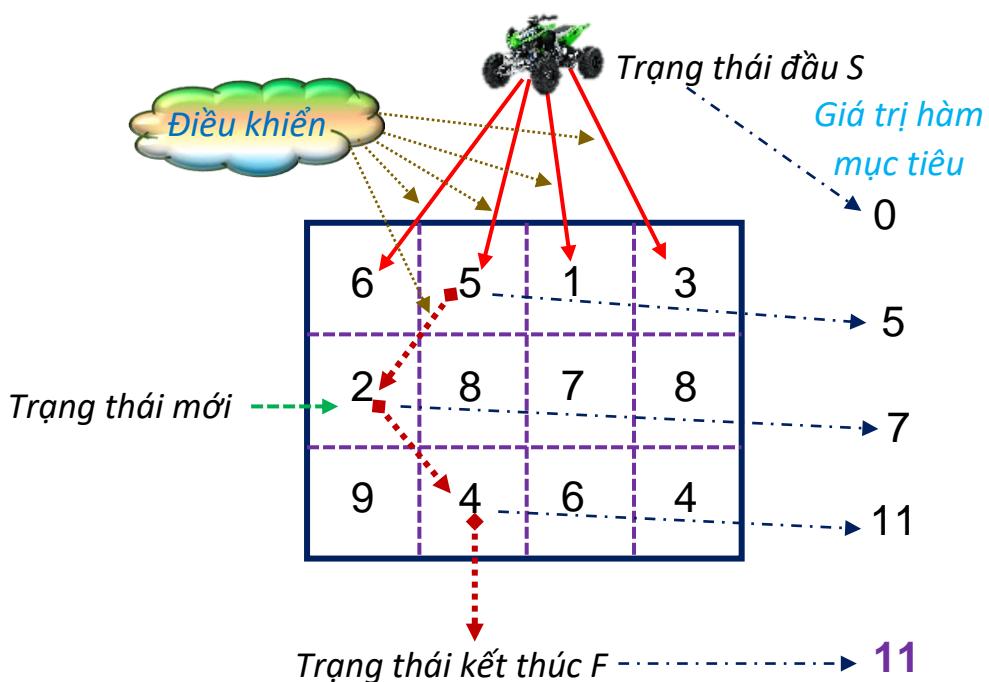
Trạng thái: Vị trí của xe được gọi là **trạng thái**,

Hàm mục tiêu: Chi phí thời gian để xe *tới và vượt qua được một trạng thái* nào đó (tức là tới một nào đó và đi qua khỏi ô đó) được gọi là **giá trị hàm mục tiêu**,

Điều khiển: Từ một trạng thái xe phải di chuyển tới trạng thái mới, việc lựa chọn một cách di chuyển được gọi là **điều khiển**.

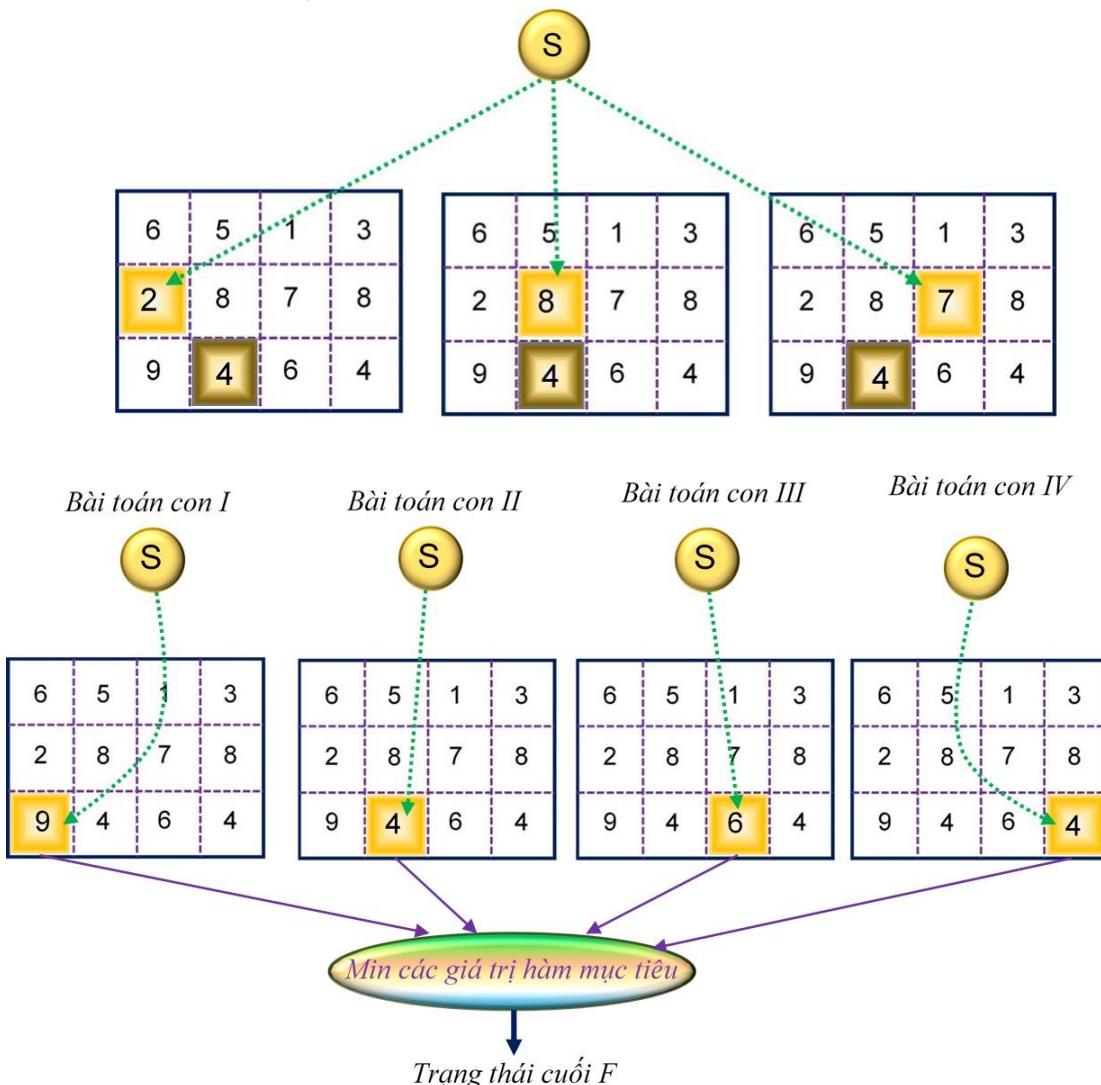
Trạng thái đầu S – xe ở ngoài vùng chướng ngại và chuẩn bị vượt qua vùng này,

Trạng thái kết thúc F – xe đã vượt qua vùng chướng ngại và đang ở ngoài vùng này.



Để giải quyết bài toán ban đầu ta xét m bài toán con (trong ví dụ cụ thể này, $m = 4$), bài toán con thứ j xác định giá trị nhỏ nhất có thể đạt được của hàm mục tiêu khi xuất phát từ trạng thái S tới trạng thái kết thúc là ô (m, j) .

Mỗi bài toán con có kích thước $(n-1) \times m$ – nhỏ hơn kích thước bài toán ban đầu. Việc tìm lời giải cho bài toán con với trạng thái kết thúc ở ô (i, j) được đưa về tìm lời giải cho các trường hợp trạng thái kết thúc lần lượt là các ô $(i-1, j-1)$, $(i-1, j)$ và $(i-1, j+1)$, trong đó *có một số lời giải đã nhận được trước đó trong quá trình xử lý!* Mỗi bài toán con này có kích thước $(i-1) \times m$.



Cứ như thế kích thước các bài toán con giảm dần cho đến khi có kích thước $1 \times m$ và có lời giải hiển nhiên: thời gian chuyển từ S tới trạng thái $(1, j)$ là $a_{1,j}$.

Lần ngược theo quá trình phân rã ta sẽ có lời giải tất cả các bài toán con và từ đó – có lời giải của bài toán ban đầu.

Tuy vậy, xử lý theo quy trình trên ta sẽ có sơ đồ giải thuật đệ quy, đòi hỏi rất **nhiều bộ nhớ và thời gian xử lý**, vượt quá khả năng cung cấp của hệ thống và ràng buộc thời gian của đề bài.

Nếu đứng trên quan điểm dữ liệu ta thấy:

- ✚ Đường đi tối ưu phải qua một trong các ô ở hàng cuối cùng, nếu ô đó là (n, j) thì hàm mục tiêu sẽ tăng thêm một số gia là $a_{n, j}$ khi về đích,
- ✚ Vì không biết trên thực tế đó là ô nào nên ta phải chuẩn bị xử lý cho mọi khả năng: lưu trữ bảng số gia có thể sẽ cần dùng đến $P = (p_1, p_2, \dots, p_m)$, trong đó $p_j = a_{n, j}$,

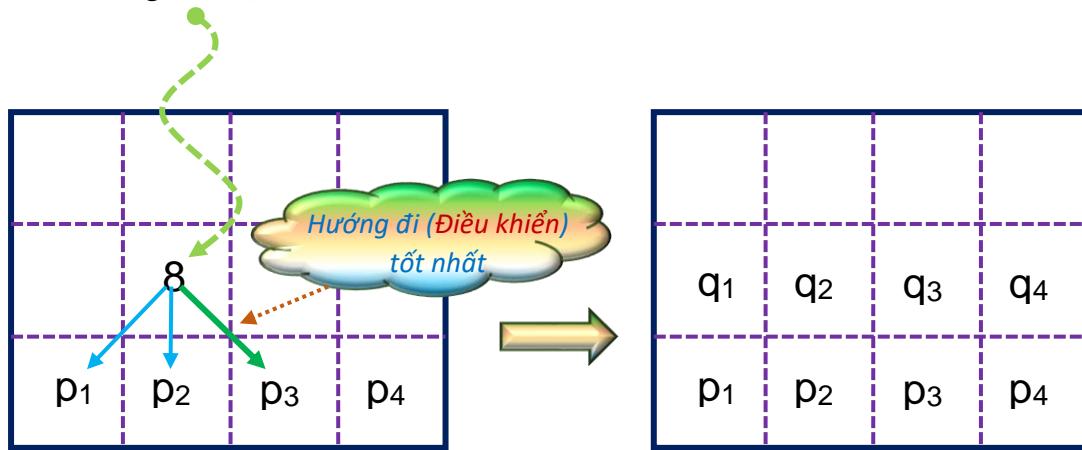
Trước khi đến hàng cuối cùng, đường đi tối ưu phải qua một ô ở hàng trên – hàng **$n-1$** , nếu đường đi tối ưu qua ô $(n-1, k)$ thì chi phí tối ưu về đích sẽ là một trong các khả năng:

- ✚ Với chi phí a_{n-1}, k tới ô $(n, k-1)$ và về đích một cách tối ưu với chi phí p_{k-1} ,
- ✚ Với chi phí a_{n-1}, k tới ô (n, k) và về đích một cách tối ưu với chi phí p_k ,
- ✚ Với chi phí a_{n-1}, k tới ô $(n, k+1)$ và về đích một cách tối ưu với chi phí p_{k+1} ,

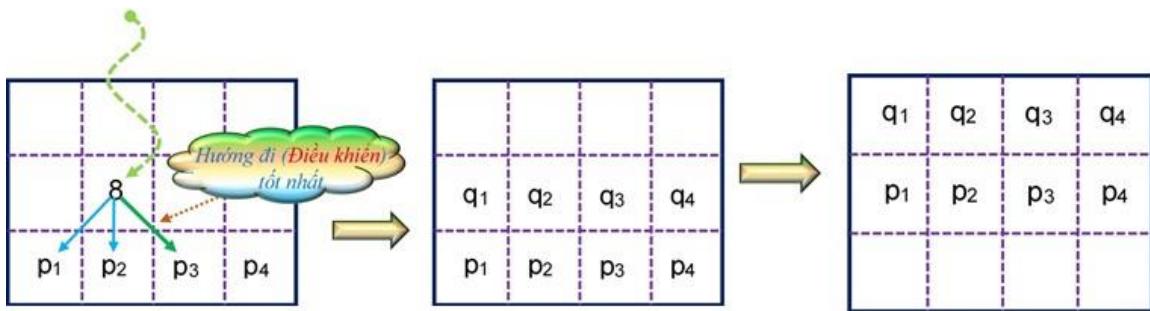
Lựa chọn đường đi với số gia nhỏ nhất vào hàm mục tiêu, ta có

$$q_k = a_{n-1, j} + \min\{p_{k-1}, p_k, p_{k+1}\}$$

Có thể dễ dàng tính q_k với mọi $k = 1, 2, \dots, m$.



Biến q thành p ta có thể tính q mới cho hàng trên tiếp theo cho đến khi được q tương ứng với các ô ở hàng trên cùng.



Giá trị hàm mục tiêu cần tìm sẽ là

$$f = \min_{j=1 \dots m} \{q_j\}$$

Nhận xét:

- ✚ Quá trình tính toán đệ quy (*recursive*) được thay bằng *sơ đồ lặp* (*recurrence*),
- ✚ Trong quá trình lặp chỉ cần làm việc với 2 mảng dữ liệu trung gian **p** và **q**,
- ✚ Khi tính giá trị **q_k** ta phải xác định điều khiển (trong trường hợp này là hướng đi) tối ưu mốc nối với **p_j** và không phụ thuộc vào việc bằng bộ điều khiển nào có được **p_j**,
- ✚ Việc dẫn xuất kết quả phải *bắt đầu từ trạng thái cuối F* và lùi dần về trạng thái đầu **S**,
- ✚ Nhiều bài toán có tính chất “đối xứng”: việc đi từ **S** đến **F** tương đương việc đi từ **F** đến **S**, quá trình tìm lời giải có thể bắt đầu từ **F** hoặc từ **S** phụ thuộc vào sơ đồ điều khiển dữ liệu.

Sơ đồ xử lý trên được gọi là *Quy hoạch động* và dựa trên nguyên lý được R. Bellman phát biểu năm 1953(*Nguyên lý Bellman*):

Không phụ thuộc vào trạng thái ban đầu và điều khiển ban đầu, điều khiển ở bước tiếp theo phải là tối ưu đối với trạng thái nhận được bởi điều khiển ở bước trước.

Điểm quan trọng cần lưu ý là việc chọn điều khiển ở bước tiếp theo *không phụ thuộc vào lịch sử điều khiển trước đó*. Điều này có nghĩa là ở bước nào đó điều khiển bị chọn sai, thì sai lầm này không thể khắc phục được trong tương lai!

Trong trường hợp có yêu cầu dẫn xuất phương án tối ưu thì cần phải tổ chức một ma trận lưu giữ các điều khiển nhận được trong quá trình tính toán.

Các giải thuật tính toán dựa trên sơ đồ lặp đơn thuần, không có việc phải lựa chọn tối ưu cũng được xếp vào lớp quy hoạch động và được gọi là *Quy hoạch động đơn giản*.

Trong nhiều trường hợp quy hoạch động được sử dụng để tạo *Bảng phương án* (*Decide Table*) cho phép phân tích và tìm lời giải theo chiều *từ trên xuống* (từ S tới F) cực kỳ hiệu quả.

Các bước xử lý

Dấu hiệu nhận dạng có thể áp dụng giải thuật Quy hoạch động:

- ✚ Bài toán ban đầu có thể chia thành các bài toán con tương tự,
- ✚ Lời giải mỗi bài toán con chỉ phụ thuộc vào giá trị hàm mục tiêu của những bài toán con khác đã giải và không phụ thuộc vào phương án tối ưu ứng với giá trị của hàm mục tiêu.

Các vấn đề cần giải quyết khi giải bài toán theo giải thuật Quy hoạch động:

1. Tổ chức dữ liệu: Khai báo các biến phục vụ sơ đồ lặp, đặc biệt lưu ý cách chọn kiểu dữ liệu phù hợp,
2. Khởi tạo giá trị đầu,
3. Xác định công thức lặp,
4. Xác định giá trị tham số quản lý miền tính lặp,
5. Xác định vị trí hoặc cách tính kết quả cần tìm của hàm mục tiêu.

Các công việc nêu trên có thể thực hiện theo *trình tự tùy ý*.

Phương pháp quy hoạch động cũng thường được sử dụng như *công cụ phân tích, đoán nhận giải thuật* trong các bài toán điều khiển, trò chơi, bài toán tương tác người – máy.

Quy hoạch động còn được sử dụng như *công cụ xây dựng bảng phương án* (*Decide Table*) trong các bài toán lô gic phức tạp.

Áp dụng với bài toán đang xét:

Bài toán đang xét có tính chất đối xứng: việc tìm đường đi từ trên xuống dưới tương đương với việc tìm đường đi từ dưới lên trên,

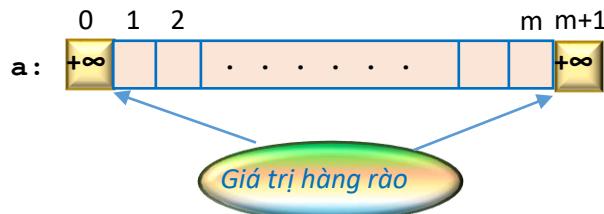
Quy trình xử lý từ trên xuống dưới cho phép vòng tránh việc lưu trữ toàn bộ dữ liệu input, thay vào đó có thể đọc từng dòng và xử lý.

Tổ chức dữ liệu:

- Mảng động `vector<int64_t> a` – lưu một dòng của ma trận dữ liệu vào,
- Mảng động `vector<int64_t> f[2]` – lưu 2 mảng phục vụ tính lặp.

Khởi tạo giá trị đầu:

Với mảng `a`:



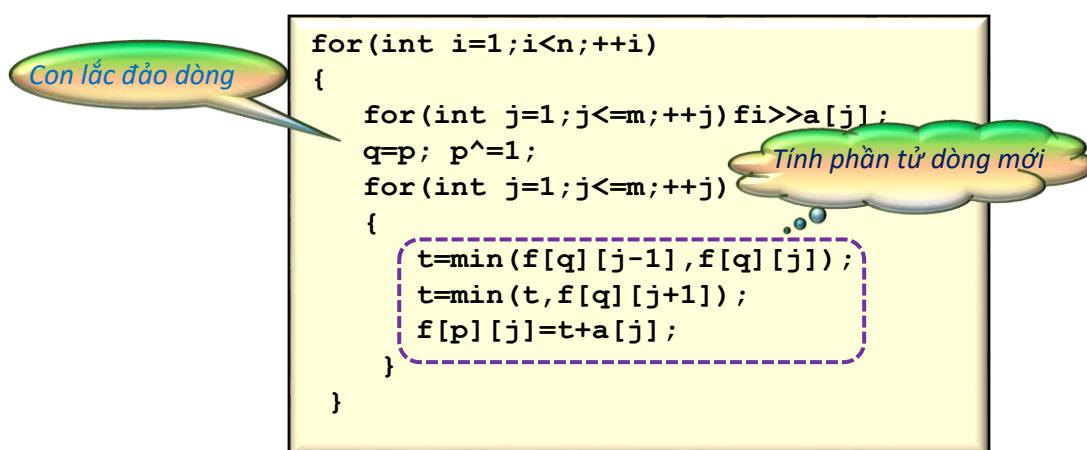
Tương tự như vậy với các mảng `f[0]` và `f[1]`,

Khởi tạo giá trị để lặp:

```
p=0; for(int j=1;j<=m;++j)
{
    fi>>t; f[p][j]=t;
}
```

Miền xử lý lặp: các dòng từ 1 đến $n-1$, các cột từ 1 đến m ,

Công thức lặp:



Kết quả: Giá trị \min ở dòng mới.

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

Chương trình: Chỉ yêu cầu đưa ra thời gian nhỏ nhất.

```
#include <bits/stdc++.h>
#define NAME "finish."
using namespace std;
ifstream fi (NAME"inp");
ofstream fo (NAME"out");
const int64_t INF=(int64_t)1e17;
int n,m,p,q;
int64_t t;
vector<int64_t>f[2];
int main()
{
    fi>>n>>m; p=0;
    vector<int64_t>a(m+2);
    f[0].resize(m+1);
    f[1].resize(m+1);
    a[0]=INF; a[m]=INF;
    f[0][0]=INF; f[0][m]=INF;
    f[1][0]=INF; f[1][m]=INF;

    for(int j=1;j<=m;++j)
    {
        fi>>t;
        f[p][j]=t;
    }
    for(int i=1;i<n;++i)
    {
        for(int j=1;j<=m;++j) fi>>a[j];
        q=p; p^=1;
        for(int j=1;j<=m;++j)
        {
            t=min(f[q][j-1],f[q][j]);
            t=min(t,f[q][j+1]);
            f[p][j]=t+a[j];
        }
    }
    int64_t ans=INF;
    for(int j=1;j<=m;++j) ans=min(ans,f[p][j]);
    fo<<ans;
    fo<<"\nTime: "<<clock()/(double)1000<<" sec";
}
```



Bài 1b.

Xét bài toán trên với yêu cầu bổ sung:

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

- ⊕ Dòng đầu tiên chứa một số nguyên – thời gian nhỏ nhất tìm được,
- ⊕ Dòng thứ 2 chứa n số nguyên xác định các cột lần lượt phải tới theo đường đi tối ưu. Trường hợp có nhiều đường đi tối ưu – đưa ra đường đi tùy chọn.

Ví dụ:

FINISH.INP
3 4
6 5 1 3
2 8 7 8
9 4 6 4

FINISH.OUT
11
2 1 2



Phân tích:

Cần tổ chức thêm các mảng:

- ▀ Mảng dữ liệu `vector<int>res (n)` – lưu vết đường đi của phương án tối ưu,
- ▀ Mảng `vector<vector<int>>route` – lưu điều khiển tối ưu ở từng ô.

Chương trình: Yêu cầu đưa ra thời gian nhỏ nhất và cách đi.

```
#include <bits/stdc++.h>
#define NAME "finish."
using namespace std;
ifstream fi (NAME"inp");
ofstream fo (NAME"out");
const int64_t INF=(int64_t)1e17;
int n,m,p,q;
int64_t t,v;
vector<int64_t>f[2];
vector<vector<int>>route;
int main()
{
    fi>>n>>m; p=0;
    vector<int64_t>a(m+2);
    vector<int>res(n);
    f[0].resize(m+1);
    f[1].resize(m+1);
    route.resize(n, vector<int>(m+2));
    a[0]=INF; a[m]=INF;
    f[0][0]=INF; f[0][m]=INF;
    f[1][0]=INF; f[1][m]=INF;
```

```

for(int j=1;j<=m;++j)
{
    fi>>t;
    f[p][j]=t; route[0][j]=0;
}
for(int i=1;i<n;++i)
{
    for(int j=1;j<=m;++j) fi>>a[j];
    q=p; p^=1;
    for(int j=1;j<=m;++j)
    {
        t=f[q][j-1];v=j-1;
        if(t>f[q][j]) t=f[q][j],v=j;
        if(t>f[q][j+1]) t=f[q][j+1],v=j+1;
        f[p][j]=t+a[j]; route[i][j]=v;
    }
}

int64_t ans=INF;
for(int j=1;j<=m;++j) if(ans>f[p][j]) ans=f[p][j],v=j;
fo<<ans<<'\n';
res[n-1]=v;
for(int i=n-1;i>0;--i)v=route[i][v],res[i-1]=v;
for(int i=0;i<n;++i) fo<<res[i]<<' ';
fo<<"\nTime: "<<clock()/(double)1000<<" sec";
}

```



Các bài tập ứng dụng giải thuật quy hoạch động

Bài 2. NÓI ĐIỂM

Tên chương trình: JOIN.CPP

Người ta kẻ một đường thẳng và đánh dấu n điểm trên đường đã kẻ, điểm thứ i ở vị trí tọa độ x_i , $i = 1 \div n$. Không có 2 điểm nào có cùng một tọa độ. Sau đó người ta dùng sơn phản quang tô một số đoạn thẳng, mỗi đoạn đều có độ dài khác 0. Các điểm đã đánh dấu trở thành điểm phản quang nếu nó nằm trong miền được sơn, kể cả là điểm đầu hoặc cuối đoạn được sơn.

Hãy xác định tổng độ dài ngắn nhất các đoạn được sơn để mọi điểm đã chọn đều trở thành điểm phản quang.

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

- ✚ Dòng đầu tiên chứa một số nguyên n ($2 \leq n \leq 10^5$),
- ✚ Dòng thứ 2 chứa n số nguyên x_1, x_2, \dots, x_n ($|x_i| < 10^9$, $i = 1 \div n$).

Kết quả: Đưa ra file văn bản JOIN.OUT một số nguyên – tổng độ dài nhỏ nhất tìm được.

Ví dụ:

JOIN.INP
6
4 30 0 10 22 2

JOIN.OUT
16



Giải thuật: Quy hoạch động.

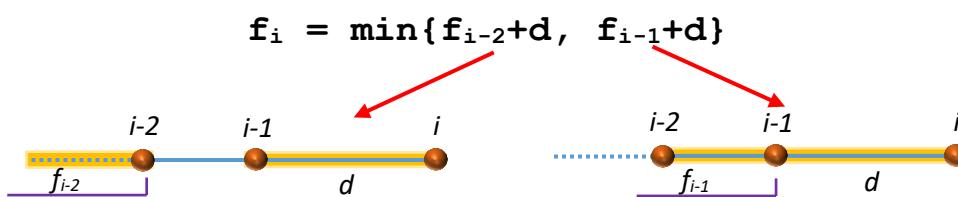
Nhận xét:

Để tổng độ dài đoạn cần sơn là nhỏ nhất thì mỗi đoạn phải bắt đầu và kết thúc bằng điểm đã đánh dấu,

Khi $n = 2$ hoặc bằng 3 – chỉ có một cách sơn: từ điểm trái nhất đến điểm phải nhất và có $\text{ans} = |\mathbf{x}_1 - \mathbf{x}_2|$ hoặc $\text{ans} = \max\{\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3\} - \min\{\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3\}$,

Sắp xếp các điểm theo thứ tự tăng dần của tọa độ,

Gọi f_i là tổng độ dài nhỏ nhất cần sơn để các điểm từ 1 đến i ($i > 3$) đều là điểm phản quang và $d = \mathbf{x}_i - \mathbf{x}_{i-1}$, ta có



Công thức lặp trên được tính với $i = 4 \div n$,

Kết quả cần tìm sẽ là f_n .

Các giá trị đầu cần chuẩn bị: $f_1 = +\infty$, $f_2 = \mathbf{x}_2 - \mathbf{x}_1$, $f_3 = \mathbf{x}_3 - \mathbf{x}_1$.

Tổ chức dữ liệu:

Hai mảng `int x[100000], f[100000]` với các chức năng như đã nói ở trên.

Lưu ý:

- ✿ Bài toán có tính chất đối xứng: duyệt từ cuối về đầu hoặc ngược lại là như nhau,
- ✿ Cách chuẩn bị trên tương ứng với việc các điểm được đánh số từ 0 , nếu đánh số từ 1 – có thể chuẩn bị đơn giản hơn.

Độ phức tạp của giải thuật: $O(n!n)$.

Chương trình:

```
#include <bits/stdc++.h>
#define NAME "join."
using namespace std;
ifstream fi (NAME"inp");
ofstream fo (NAME"out");
const int64_t INF=(int)1e9;
int n,t;

int x[100000],f[100000];
int main()
{
    fi>>n;
    for(int i=0;i<n;++i)fi>>x[i];
    if(n==2){fo<<abs(x[0]-x[1]); return 0;}
    sort(x,x+n);
    f[0]=INF;f[1]=x[1]-x[0]; f[2]=f[1]+x[2]-x[1];
    for(int i=3;i<n;++i)
        {t=x[i]-x[i-1];f[i]=min(f[i-1]+t,f[i-2]+t);}

    fo<<f[n-1];
    fo<<"\nTime: "<<clock()/(double)1000<<" sec";
}
```



Bài 3. DÃY KÝ TỰ CON CHUNG DÀI NHẤT

Tên chương trình: LCS.CPP

Cho hai xâu **a** và **b**. Xâu **z** được gọi là dãy con của **a** nếu có thể gạch bỏ một số(có thể là 0) ký tự trong **a** để nhận được **z**. Ví dụ, **a** = “**abcdef**” và **z** = “**bde**” , thì **z** là dãy con của **a**, nhưng “**bed**” – không phải là dãy con của **a**.

Nếu **z** đồng thời là dãy con của **a** và của **b** thì **z** được gọi là dãy con chung.

Hãy xác định độ dài dãy con chung lớn nhất của hai xâu **a** và **b**.

Dữ liệu: Vào từ file văn bản LCS.INP:

- ⊕ Dòng đầu tiên chứa xâu **a** các ký tự la tinh thường độ dài không quá 10^4 ,
- ⊕ Dòng thứ hai chứa xâu **b** các ký tự la tinh thường độ dài không quá 10^4 .

Kết quả: Đưa ra file văn bản LCS.OUT một số nguyên – độ dài lớn nhất tìm được.

Ví dụ:

LCS.INP	LCS.OUT
abcdabcdef cabgad	4



Giải thuật: Quy hoạch động.

Nhận xét:

Gọi n và m là độ dài các xâu a và b ,

Vấn đề cần phải giải quyết là tìm độ dài của dãy con chung dài nhất (**LCS** – *Largest Common Subsequence*),

Bài toán có tính chất đối xứng (xét từ cuối xâu về đầu hay ngược lại là như nhau), nhưng vì C++ lưu các ký tự bắt đầu từ vị trí 0 và không cho phép sử dụng chỉ số âm, để tiện lưu trữ giá trị hàng rào ta sẽ xử lý theo đúng sơ đồ lý thuyết tổng quát, xét các xâu từ cuối về đầu.

Gọi $\text{lcs}_{i,j}$ là LCS của hậu tố xâu a bắt đầu từ ký tự i và hậu tố xâu b bắt đầu từ ký tự j ,

Ta có:

$$\text{lcs}_{i,j} = \begin{cases} 0 & i=n \text{ hoặc } j=m \\ \text{lcs}_{i+1,j+1} + 1 & x_i = y_j \\ \max\{\text{lcs}_{i+1,j+1}, \text{lcs}_{i+1,j}\} & x_i \neq y_j \end{cases}$$

Việc tính dòng i và cột j của bảng **lsc** chỉ cần dựa trên dòng $i+1$ và cột $j+1$ vì vậy không cần thiết phải lưu trữ mảng hai chiều mà chỉ cần 2 mảng một chiều, lưu trữ dưới dạng `int f[2][10000]` với kỹ thuật con lắc để hoán đổi vị trí các mảng giá trị mới và giá trị cũ,

Các giá trị đầu có thể khởi tạo trước mỗi bước lặp,

Phạm vi duyệt: $i = n \div 0$, $j = m \div 0$ (bắt đầu từ n , m để khởi tạo giá trị đầu),

Kết quả: ở đầu dòng mới (biến chỉ số 0).

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

Chương trình:

```
#include <bits/stdc++.h>
#define NAME "lcs."
using namespace std;
ifstream fi (NAME"inp");
ofstream fo (NAME"out");
const int64_t INF=(int)1e9;
int n,m;
string a,b;
int f[2][10000];

int main()
{
    fi>>a>>b;p=1,q=0;
    n=a.size(); m=b.size();
    for(int i=n;i>=0;--i)
    {
        p^=1;q^=1;
        for(int j=m;j>=0;--j)
            if(i==n||j==m)f[p][j]=0;
            else if(a[i+1]==b[j+1])f[p][j]=f[q][j]+1;
            else f[p][j]=max(f[q][j],f[p][j+1]);
    }
    fo<<f[p][0];
    fo<<"\nTime: "<<clock()/(double)1000<<" sec";
}
```



Quy hoạch động đơn giản

Quy hoạch động đơn giản là quy trình tính toán theo sơ đồ lặp *không đòi hỏi lựa chọn tối ưu*.

Phần lớn giải thuật giải các bài toán lặp đều có mô hình toán học đệ quy. Có thể lập trình trực tiếp theo mô hình đệ quy, chương trình rất ngắn gọn, nhưng độ phức tạp của giải thuật sẽ rất cao và tốn rất nhiều bộ nhớ, nhiều khi vượt quá khả năng cung cấp của hệ thống.

Thông thường tính toán theo sơ đồ lặp là việc quản lý hai trạng thái: *Trạng thái cũ* và *Trạng thái mới*.

Sơ đồ giải thuật lặp:

```
Chuẩn bị Trạng thái mới,  
while (!Điều kiện kết thúc)  
{  
    Biến trạng thái mới thành trạng thái cũ,  
    Tính lại trạng thái mới,  
}  
Đưa ra kết quả: Trạng thái mới.
```

Việc chuẩn bị trạng thái mới cho phép nhận được kết quả đúng ngay khi số lần lặp bằng 0.

Khi trạng thái có cấu trúc dữ liệu phức tạp, việc biến *trạng thái mới* thành *trạng thái cũ* có thể thực hiện bằng *kỹ thuật con lắc*.

Trong một số trường hợp sơ đồ tính toán sẽ ngắn gọn hơn nhiều nếu có thể *cập nhật tại chỗ trạng thái* mà không cần phải ghi sang nơi mới. Việc cập nhật tại chỗ đặc biệt có hiệu quả khi trạng thái cần được lưu trữ không phải bằng các đại lượng vô hướng, ví dụ như mảng, véc tơ, tập hợp, . . .

Sơ đồ tính toán sẽ có dạng:

```
Chuẩn bị Trạng thái ,  
while (!Điều kiện kết thúc) Cập nhật Trạng thái,  
Đưa ra kết quả: Trạng thái nhận được.
```

Xét một số bài toán quy hoạch động đơn giản minh họa cho sơ đồ lặp.

Xét biểu thức chỉ chứa các ngoặc ‘(‘ và ‘)’.

Biểu thức ngoặc đúng được định nghĩa như sau:

- ✿ $()$ là biểu thức ngoặc đúng,
- ✿ Nếu \mathbf{A} là một biểu thức ngoặc đúng thì $(\mathbf{A}, \mathbf{A}()$ và (\mathbf{A}) cũng là những biểu thức ngoặc đúng.

Cho n ngoặc mở ($\text{và } n$ ngoặc đóng) (gọi tắt là n cặp ngoặc). Số lượng biểu thức ngoặc đúng chứa n cặp ngoặc là số Catalan thứ n , ký hiệu C_n .

Ví dụ, với $n = 3$ ta có $C_3 = 5$:

$$() () () \quad () (()) \quad ((()) \quad ((()) \quad ((()))$$

Cho số nguyên n . Hãy xác định 9 chữ số cuối cùng của C_n , không dẫn xuất các số 0 không có nghĩa ở đầu.

Dữ liệu: Vào từ file văn bản CATALAN.INP gồm một dòng chứa số nguyên n ($1 \leq n \leq 10^4$).

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

Ví dụ:

CATALAN.INP	CATALAN.OUT
3	5



Giải thuật: sơ đồ lặp (Quy hoạch động đơn giản).

Số Catalan có nhiều ứng dụng trong lý thuyết cũng như trong thực tế và có nhiều cách tính khác nhau:

$$C_0 = 1 \quad \text{và} \quad C_n = \sum_{i=0}^{n-1} C_i C_{n-1-i}$$

$$C_0 = 1 \quad \text{và} \quad C_n = \frac{2(2n-1)}{n+1} C_{n-1}$$

$$C_n = \frac{1}{n+1} \binom{2n}{n} = \frac{1}{2n+1} \binom{2n+1}{n} = \binom{2n}{n} - \binom{2n}{n-1}$$

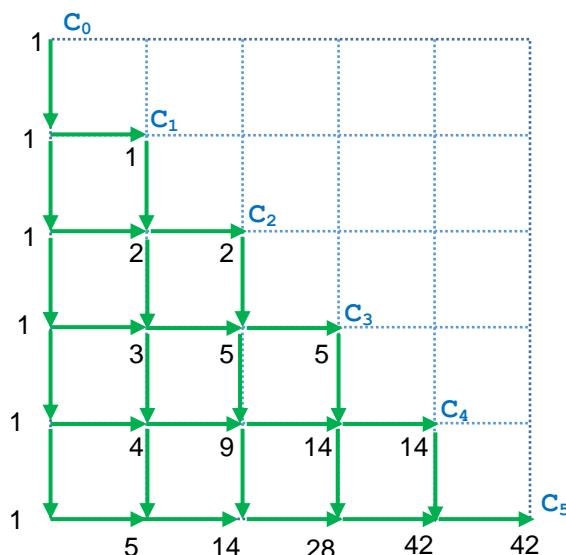


Với yêu cầu của đầu bài, có thể sử dụng công thức lặp thứ nhất hoặc thứ 3 nhưng việc dẫn xuất kết quả khá rắc rối với cùng độ phức tạp $O(n^2)$.

Ta có thể xây dựng sơ đồ tính toán đơn giản hơn ở mỗi bước lặp.

Trên lưới ô vuông kích thước $n \times n$ xét các đường đi từ góc trên trái xuống góc dưới, mỗi bước đi xuống tương ứng với (, bước đi ngang – tương ứng với), các đường đi phải thỏa mãn 2 tính chất sau:

- Từ mỗi nút chỉ có thể đi xuống dưới hoặc sang phải,
- Số bước đi sang phải không được vượt quá số bước đi xuống dưới tính từ đầu đường đi.



Để dễ dàng thấy rằng từ dòng ($i-1$) ta có thể tính ra các số trên dòng i . Gọi các số trên dòng $i-1$ là $D = (d_0, d_1, d_2, \dots, d_{i-2}, d_{i-1}, d_i, \dots)$, $d_j = 0$ với $j > i-1$.

Ta có công thức lặp cập nhật tại chỗ thông tin:

$$d_j = d_j + d_{j-1}, \quad j = 1 \div i.$$

Công thức trên chỉ sử dụng phép cộng, vì vậy việc dẫn xuất kết quả sẽ đơn giản hơn so với các công thức lặp khác.

Kết quả: biến d_n .

Chương trình:

```
#include <bits/stdc++.h>
#define NAME "catalan."
#define times clock()/(double)1000
using namespace std;
ifstream fi (NAME"inp");
ofstream fo (NAME"out");
const int p=1e9;
int n,d[10001]={0};

int main()
{
    fi>>n;
    d[0]=1;
    for(int i=1;i<=n;++i)
        for(int j=1;j<=i;++j)d[j]=(d[j]+d[j-1])%p;
    fo<<d[n];
    fo<<"\nTime: "<<times<<" sec";
}
```



LÁT ĐƯỜNG VIỀN

Tên chương trình: PAVE.CPP

Người ta dùng 2 loại gạch có kích thước 1×2 và 2×2 để lát một đường viền kích thước $n \times 2$. Số lượng mỗi loại gạch là vô hạn. Khi dùng, các viên gạch kích thước 1×2 có thể đặt ngang hay dọc tùy ý. Hai cách lát gọi là khác nhau nếu tìm thấy một vị trí i trên đường viền được lát bằng các cách khác nhau.

Ví dụ với $n = 3$ ta có 5 cách lát khác nhau:



Hãy đưa ra số cách lát theo mô đun $10^9 + 7$.

Dữ liệu: Vào từ file văn bản PAVE.INP gồm một dòng chứa số nguyên n ($1 \leq n \leq 10^5$).

Kết quả: Đưa ra file văn bản PAVE.OUT một số nguyên – số cách lát theo mô đun $10^9 + 7$.

Ví dụ:

PAVE.INP	PAVE.OUT
3	5



Giải thuật: sơ đồ lặp (Quy hoạch động đơn giản).

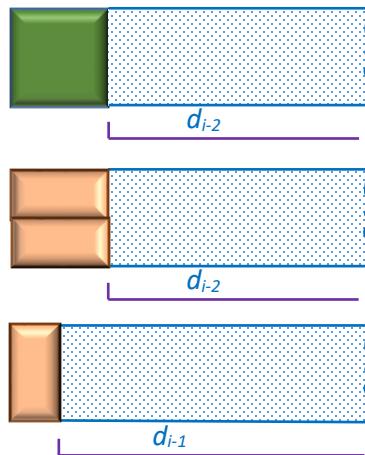
Với $n = 1$ có một cách lát,

Với $n = 2$ có 3 cách lát.



Xét $n \geq 3$: Gọi d_i là số cách lát đường viền độ dài i , ta có:

$$d_i = d_{i-1} + 2 \times d_{i-2}$$



Độ phức tạp của giải thuật: $O(n)$.

Chương trình:

```
#include <bits/stdc++.h>
#define NAME "pave."
#define times clock() / (double)1000
using namespace std;
ifstream fi (NAME"inp");
ofstream fo (NAME"out");
const int p=1e9+7;
int64_t d2=1,d3=3,d1;
int n;
int main()
{
    fi>>n;
    if(n==1)d3=1;
    else
        if(n>2) for(int i=2;i<n;++i)
            {d1=d2; d2=d3; d3=(d2+2*d1)%p; }
    fo<<d3;
    fo<<"\nTime: "<<times<<" sec";
}
```



HAI SỐ 1

Tên chương trình: TWOONE.CPP

Cho số nguyên dương n . Hãy xác định số lượng số nguyên dương không lớn hơn n và trong dạng biểu diễn nhị phân của các số đó không có hai số 1 đứng liên tiếp nhau.

Ví dụ, với $n = 6$ ta có 4 số cần tìm: $1_{10} = 1_2$, $2_{10} = 10_2$, $4_{10} = 100_2$, $5_{10} = 101_2$.

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

- ✚ Dòng đầu tiên chứa một số nguyên t ($1 \leq t \leq 10^5$), trong đó t – số lượng tests,
- ✚ Dòng thứ i trong t dòng sau chứa số nguyên n_i ($1 \leq n_i \leq 10^{18}$).

Kết quả: Đưa ra file văn bản TWOONE.OUT t số nguyên – các kết quả tìm được, mỗi số trên một dòng.

Ví dụ:

TWOONE.INP	TWOONE.OUT
4	4
6	7
10	13
32	88
446	



Giải thuật: Sơ đồ lặp (Quy hoạch động đơn giản), Số Fibonacci.

Xét các số có i bít có nghĩa, gọi a_i – số xâu bít độ dài m bắt đầu bằng 0 và không có hai ký tự 1 đứng liên tiếp, b_i – tổng số xâu bít không có hai ký tự 1 đứng liên tiếp, bắt đầu bằng 1 và có độ dài không vượt quá i ,

Đặt $a_0 = 1, b_0 = 0$, ta có: $a_1 = 1, b_1 = 1$, với $i > 1$ có:

$$\text{a}_i = \text{a}_{i-1} + \text{a}_{i-2},$$

$$\text{b}_i = \text{b}_{i-1} + \text{a}_{i-1}.$$

Các số nguyên n cần xét không vượt quá 10^{18} vì vậy các xâu bít tương ứng có độ dài không quá 64. Gọi k là vị trí bít 1 trái nhất của n .

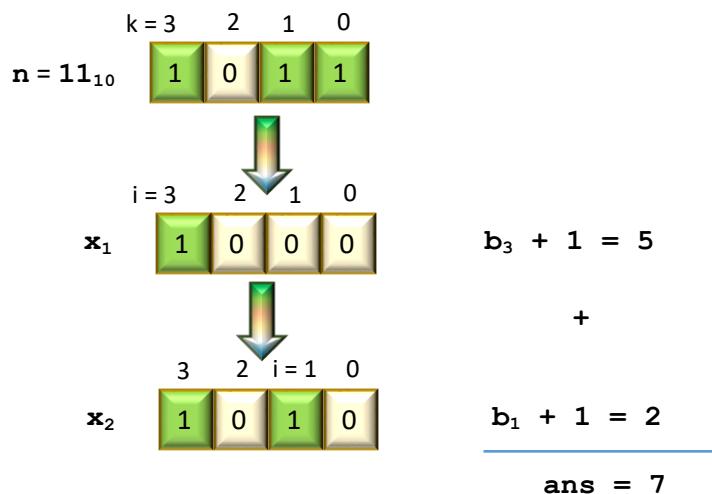
i	a_i	b_i
0	1	0
1	1	1
2	2	2
3	3	4
4	5	7
5	8	12

Tạo các số mới bằng cách xóa hết các bít 1 trong n , chỉ giữ lại 1 bít 1 trái nhất, giữ lại 2 bít 1 trái nhất, giữ lại 3 bít 1 trái nhất, ... Quá trình tạo số mới chấm dứt khi không còn bít 1 để xóa hoặc lần đầu tiên gặp bít 1 thứ 2 trong cặp 2 bít 1 liên tiếp.

Giả thiết x_j là một trong các số mới nhận được, trong đó i là vị trí bít 1 phải nhất, khi đó số lượng số thỏa mãn yêu cầu để bài với x_j sẽ là b_{i+1} .

Kết quả là tổng các số lượng tìm được.

Ví dụ với $n = 11_{10}$:



Độ phức tạp của giải thuật: $O(t)$.

Chương trình:

```
#include <bits/stdc++.h>
#define NAME "twoone."
#define times clock()/(double)1000
using namespace std;
ifstream fi (NAME"inp");
ofstream fo (NAME"out");
int64_t n,ans;
int64_t a[64],b[64];
int k,t,d;

int main()
{
    a[0]=1;b[0]=0; a[1]=1; b[1]=1;
    for(int i=2;i<=63;++i)
    {
        a[i]=a[i-1]+a[i-2];
        b[i]=b[i-1]+a[i-1];
    }
    fi>>t;
    for(int j=1;j<=t;++j)
    {
        fi>>n;
        for(int i=63;i>=0;--i) if((n>>i)&1){k=i;break;}
        ans=b[k];
        d=1;
        for(int i=k-1;i>=0;--i)
        {
            if((n>>i)&1)
            {

                if((n>>(i+1))&1){ans+=b[i];break;}
                ans+=b[i];++d;
            }
        }
        fo<<ans+d<<'\n';
    }
    fo<<"\nTime: "<<times<<" sec";
}
```



Trong máy ATM có k ngăn lưu trữ các tờ tiền mệnh giá a_1, a_2, \dots, a_k , mỗi mệnh giá có số lượng tờ đủ nhiều.

Khi khách hàng có yêu cầu rút n đồng, chương trình điều khiển sẽ xác định xem có cách trả đúng n đồng hay không. Nếu có cách trả thì xác định số tờ tiền ít nhất cần sử dụng và số lượng tờ mỗi mệnh giá cần đưa ra.

Ví dụ với 3 mệnh giá 10, 60, 100 và số tiền cần rút là 130 thì số lượng tờ tiền ít nhất cần sử dụng là 3, phương án đưa ra là 2 tờ mệnh giá 60 và 1 tờ mệnh giá 10.

Dữ liệu: Vào từ file văn bản ATM.INP:

- ✚ Dòng đầu tiên chứa một số nguyên k ($1 \leq k \leq 20$),
- ✚ Dòng thứ 2 chứa k số nguyên a_1, a_2, \dots, a_k theo thứ tự tăng dần ($1 \leq a_i < a_j \leq 10^5$, $1 \leq i < j \leq k$),
- ✚ Dòng thứ 3 chứa số nguyên n ($1 \leq n \leq 10^6$).

Kết quả: Đưa ra file văn bản ATM.OUT nếu không có cách trả thì đưa ra số -1, trong trường hợp có cách trả: dòng thứ nhất chứa một số nguyên – số lượng tờ tiền cần sử dụng, dòng thứ 2 chứa k số nguyên, số thứ i cho biết phải sử dụng bao nhiêu tờ tiền mệnh giá a_i . Nếu tồn tại nhiều cách chi trả – đưa ra cách tùy chọn.

Ví dụ:

ATM.INP	ATM.OUT
3	3
10 60 100	1 2 0
130	



Giải thuật: Quy hoạch động.

Gọi $f[j]$ – cách chi trả tối ưu số tiền j ($0 \leq j \leq n$), nếu không có cách chi trả thì $f[j] = +\infty$, $f[0]=0$, $f[j] = +\infty$ với $0 < j < a_1$.

Nếu $j \geq a_i$: ta có thể sử dụng một lần tờ mệnh giá a_i và số tờ tiền phải sử dụng sẽ là $f[j-a[i]] + 1$,

Ghi nhận các cách chi trả tối ưu đối với j :

$$f[j] = \min\{f[j-a[1]], f[j-a[2]], \dots, f[j-a[k]]\} + 1$$

Lần lượt duyệt với j từ a_1 đến n và có số lượng tờ cần dùng ít nhất ở $f[n]$.

Xác định cách chi trả (khi $f[n] < +\infty$):

Điều kiện có sử dụng đồng tiền mệnh giá a_i : $f[n]=f[n-a[i]]+1$,

Nếu có sử dụng đồng tiền mệnh giá a_i : giảm n ($n=a[i]$), tích lũy tàn số sử dụng và tiếp tục kiểm tra.

Độ phức tạp của giải thuật: O(k×n).

Chương trình:

```
#include <bits/stdc++.h>
#define NAME "atm."
#define times clock()/(double)1000
using namespace std;
ifstream fi (NAME"inp");
ofstream fo (NAME"out");
const int INF=1e9;
int n,k;

int main()
{
    fi>>k;
    vector<int>a(k);
    for(int i=0;i<k;++i)fi>>a[i];
    fi>>n;
    vector<int>f(n+1); f[0]=0;
    for(int j=1;j<a[1];++j)f[j]=INF;
    for(int j=a[1];j<=n;++j)
    {
        f[j]=INF;
        for(int i=0;i<k;++i)
            if(j>=a[i] && f[j-a[i]]+1<f[j])f[j]=f[j-a[i]]+1;
    }
    if(f[n]==INF){fo<<"-1";return 0;}
    vector<int>b(k,0);
    fo<<f[n]<<'\n';
    while(n>0)
        for(int i=0;i<k;++i)
            if(f[n-a[i]]+1==f[n]) {++b[i]; n-=a[i]; break;}
    for(int i=0;i<k;++i) fo<<b[i]<<' ';
    fo<<"\nTime: "<<times<<" sec";
}
```

