

# GIẢI THUẬT HIỆU NĂNG CAO

## Mục lục

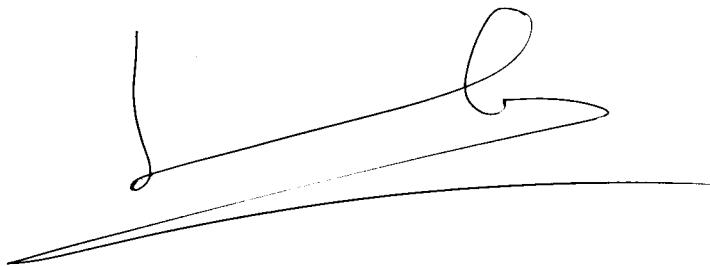
<b>1 – Hàm tiền tố .....</b>	5
1.1 – Hàm tổng tiền tố .....	5
SD09. KHỐI LẬP PHƯƠNG <i>Tên chương trình:</i> CUBICS.???	6
SD02. BÓNG RỔ <i>Tên chương trình:</i> BASKET_B.???	10
UF38. CHÁO VÀ PHỞ <i>Tên chương trình:</i> GRUEL???	16
43L. CÁCH MẠNG <i>Tên chương trình:</i> REVOLUTIONS.???	17
SVC14. BẢN ĐỒ GEN <i>Tên chương trình:</i> GENMAP.???	20
2.1 – Cây chỉ số nhị phân (Binary Indexed Trees) .....	24
EP14. SỐ NGHỊCH THẾ <i>Tên chương trình:</i> INVERS.???	41
EP37. DÂY CUNG <i>Tên chương trình:</i> CHORDS.???	46
50K. CUỘC CHIẾN BẢO VỆ HOGVARTS <i>Tên chương trình:</i> BATLE.???	51
42I. KHÁCH DU LỊCH <i>Tên chương trình:</i> TOURISTS.???	53
UF26. HỌP KẸO <i>Tên chương trình:</i> CANDIES.???	55
VL08. TRÒ CHƠI <i>Tên chương trình:</i> FLASHMOB.???	58
<b>2 – Quản lý cực trị .....</b>	63
2.1 – Tìm cực trị toàn cục .....	63
49K. CHIA THUỐC <i>Tên chương trình:</i> LIQUORS.???	66
46J. TRÌNH TỰ <i>Tên chương trình:</i> ORDER.???	68
38L. HIỆN TƯỢNG ẤM LÊN TOÀN CẦU <i>Tên chương trình:</i> WARMING.???	70
2.2 – Tìm cực trị trên các tập con của tập ổn định .....	73
VU35. ĐỦ CHẤT <i>Tên chương trình:</i> FOOT.???	78
HÀNG RÀO <i>Tên chương trình:</i> FENCE.???	81
2.3- Tìm cực trị trên các tập con thay đổi của tập biến động .....	83
2.3.1 – Khởi tạo cây .....	84
2.3.2 – Cập nhật cây .....	87
2.3.3 – Xử lý các truy vấn tìm cực trị .....	88
VO46. THỨC ĂN NHANH <i>Tên chương trình:</i> FASTFOOT.???	90
13L. BẠCH TUYẾT <i>Tên chương trình:</i> SNOW_WH.???	93
<b>3 – Danh sách mốc nối .....</b>	99
47. KHÓA DÙNG MỘT LẦN <i>Tên chương trình:</i> UNIQUEK.???	100
VL10. THÁM HIỂM <i>Tên chương trình:</i> EXPEDITION.???	103
<b>4 – Kỹ thuật hàm băm .....</b>	106

VM47. MẬT KHẨU	Tên chương trình: PAROLE.???	108
	<i>Giải thuật</i>	109
VN29. BẮT ĐẦU CỦA KẾT THÚC	Tên chương trình: BEGINEND.???	111
	<i>Giải thuật</i>	112
VP13. BA NGƯỜI BẠN	Tên chương trình: FRIENDS.???	114
<b>5 – Cây Van Emde Boas</b>		115
<b>6 – Đồ thị cây RB(Red - Black)</b>		117
6.1 - Tính chất của cây đỏ đen		117
6.2 – Các phép biến đổi trong cây Red – Black		118
<b>7 – Vụn đồng Fibonacci heap</b>		125
7.1 – Cấu trúc của Fibonacci heap		125
7.2 – Các phép xử lý		127
<i>Các phép biến đổi trong Fibonacci heap</i>		129
7.2 – Phạm vi sử dụng của Fibonacci heap		133
<b>Kỹ thuật lập trình</b>		135
1. Cơ sở lập trình		135
VH39. DIỄC	Tên chương trình: HERONS.???	135
<b>16M. HẸN GẶP</b>	Tên chương trình: APPOINTMENT.???	136
VJ18. SÂN VẬN ĐỘNG	Tên chương trình: STADIUM.???	139
VP09. BẢNG LŨY THỪA	Tên chương trình: POWERTAB.???	144
16L. CƠ SỐ	Tên chương trình: BASE.???	149
VL39. CÂN BẰNG	Tên chương trình: BALANCE.???	152
VO48. BÓNG CHUYỀN BÃI BIỂN	Tên chương trình: VOLLEY.???	154
VO40. ĐƯỜNG SẮT TRÊN CAO	Tên chương trình: TRAIN.???	156
VO41. DÂY XÍCH	Tên chương trình: CHAIN.???	157
VO44. LIÊN HOAN PHIM	Tên chương trình: CINEMA.???	158
VM08. TRÁM ĐEN	Tên chương trình: CANARIUM.???	161
VO32. TUYẾN BAY	Tên chương trình: FLY.???	163
VN16. HÌNH CHỮ NHẬT	Tên chương trình: RECTANGLE.???	166
VJ46. TRÌNH TỰ	Tên chương trình: ORDER.???	168
2. Sơ đồ lặp và Quy hoạch động		170
VO10. CÚM GIA CÀM	Tên chương trình: FLU.???	171
VO42. ĐỒ CHƠI	Tên chương trình: TOY.???	174
3. Các bài toán có nội dung hình học		178
KTLT. ĐƯỜNG GẤP KHÚC	Tên chương trình: POLYG.???	178
TE38. ĐƯỜNG GẤP KHÚC 2	Tên chương trình: POLYGON.???	180

VN19. KINH KẾ	Tên chương trình: SUTRAS.???	183
4. Các bài toán đồ thị		187
4.1 – Vai trò của cấu trúc dữ liệu trong các bài toán đồ thị		187
4.1.1 – HÀNG ĐỢI ƯU TIÊN VÀ GIẢI THUẬT DIJSKTRA		187
<b>A - BÀI TOÁN</b>		187
<b>B - GIẢI THUẬT</b>		187
<b>C - KỸ THUẬT CÀI ĐẶT HIỆU QUẢ CAO VỚI ĐỒ THỊ MA TRẬN THƯA</b>		188
4.2 – ĐỒ THỊ HAI MÀU		192
4.2.1 – Khái niệm chung		192
<b>BÀI TOÁN</b>		194
VL07. THI ĐẤU	Tên chương trình: TOURNAMENT.???	200
5. Xử lý xâu		202
VL50. QUAN HỆ HỌ HÀNG	Tên chương trình: RELATIVES. ???	204
VO22. NGÀY NGHỈ	Tên chương trình: HOLYDAYS.???	206
VN44. THƠ CỔ ĐỘNG	Tên chương trình: LYRICS.???	208
VL06. ROBOT	Tên chương trình: ROBOT.???	210
VL19. TIẾN HÓA	Tên chương trình: EVOLUTION.???	212
VN37. DANH SÁCH TRƯỜNG HỌC	Tên chương trình: SCHOOLS.???	213
39M. BẢN ĐỒ	Tên chương trình: MAP.???	215
6. Cấu trúc dữ liệu hàng đợi/stack		218
30M. MATRIOSHKA	Tên chương trình: MATRIOSHKAS.???	218
VO26. TRÒ CHƠI	Tên chương trình: GAMEVO26.???	220
VL20. BỘ SƯU TẬP	Tên chương trình: COLLECTION.???	222
VL18. LỊCH MỚI	Tên chương trình: CALENDAR.???	225
VP11. NÉM ĐÁ	Tên chương trình: STONE.???	227
VJ19. TAM SAO THẤT BỐN	Tên chương trình: SQ.???	230
VJ34. CHỚP SÁNG	Tên chương trình: FLASH.???	233
BM31. TRỒNG CÂY XANH	Tên chương trình: ROWAN.???	238
7. Quản lý khoảng (Segment Control) bằng cây nhị phân		240
VP17. CHỈ SỐ GIÁ TIÊU DÙNG	Tên chương trình: CPI.???	242
VP10. HỆ THỐNG CHẤM ĐIỂM	Tên chương trình: EVALSYS.???	252
8. Cây tiền tố		260
VP08. SỐ LƯỢNG KHOẢNG	Tên chương trình: SEGNUM.???	261
<i>Tài liệu</i>		271
<b>CHIA KEO</b>	Tên chương trình: CANDIES.???	275

Với tất cả mọi người, việc làm hôm nay chính  
là lúc họ "Malting your memory" - Kỷ niệm  
của ngày mai chính là kết quả của Malting  
lấy giờ. Với các bạn OLP, nó là ~~bóng~~ giờ  
sẽ là kỷ ức của hôm sau - Ký ức đó sẽ  
não plus thuộc về nhiều ngày sau ban  
"Typing your memory". Nó là hết mệt,  
chạy hết mệt để 5, 10, ..., năm nữa các  
bạn không bao giờ kéo mót khoảng thời gian  
của tuổi trẻ rồi nón lông H.B.2 (FIT-iUH)

Sài Gòn, mùa code chir 2014



## **1 – Hàm tiền tố**

Hàm tiền tố là hàm xác định trên phần đầu của một cấu trúc dữ liệu (xâu, mảng số, véc tơ, . . .).

Ngoài việc phục vụ cho các bài toán xử lý trực tiếp tiền tố các hàm này thường là công cụ chủ yếu xử lý các bài toán liên quan tới một phần liên tục của đối tượng đang xét. Khi đối tượng đang xét xác định trên một tập vô hạn hàm tiền tố thường được sử dụng để đưa các bài toán 2 đầu di động thành các bài toán một đầu di động và làm cho độ phức tạp của giải thuật giảm hẳn một cách đáng kể.

Hàm tiền tố là công cụ chủ yếu để xử lý nhiều truy vấn thông tin trên các đoạn ( $x, y$ ) khác nhau của đối tượng đang xét. Trong phần lớn các bài toán loại này, độ phức tạp của việc tính hàm tiền tố sẽ là tham số quyết định tác động lên độ phức tạp chung của giải thuật.

Có hai cách tổ chức lưu trữ giá trị hàm tiền tố:

- ◆ Dạng mảng tổng các giá trị từ đầu tới vị trí đang xét (*hàm tổng tiền tố*),
- ◆ Dạng cây thống kê các phân tử thỏa mãn điều kiện cho trước.

### **1.1 – Hàm tổng tiền tố**

Cho dãy số nguyên  $A = (a_1, a_2, \dots, a_n)$ , dãy số  $F = (f_0, f_1, \dots, f_n)$  được gọi là hàm tổng tiền tố của dãy A nếu:

- ◆  $f_0 = 0,$
- ◆  $f_i = \sum_{j=1}^i a_j, i = 1 \div n.$

Dễ dàng thấy rằng  $f_i = f_{i-1} + a_i, i = 1 \div n$ . Việc tính dãy số F ở đây có độ phức tạp  $O(n)$ .

Tổng  $t = a_p + a_{p+1} + \dots + a_q$  ( $p \leq 1 \leq q \leq n$ ) được tính với độ phức tạp  $O(1)$ :

$$t = f_q - f_{p-1}$$

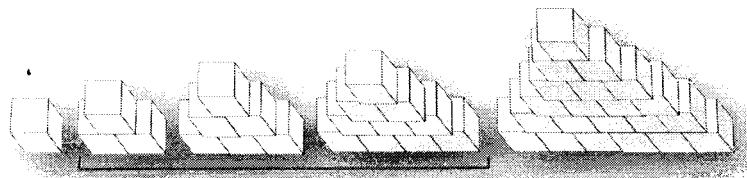
## Bài tập ứng dụng:

### SD09. KHỐI LẬP PHƯƠNG

Tên chương trình: CUBICS.???

Quà sinh nhật của Jimmy là một bộ khối lập phương xếp hình. Jimmy xếp thành  $n$  tháp, tháp thứ  $i$  có độ cao là  $a_i$  ( $1 \leq a_i \leq 10^9$ ,  $1 \leq n \leq 10^5$ ,  $i = 1 \dots n$ ).

Jimmy rất có cảm tình với số nguyên  $k$ , vì vậy dãy liên tục các tháp được coi là hài hòa nếu chúng có độ cao trung bình là  $k$  ( $1 \leq k \leq 10^9$ ).



**Yêu cầu:** Cho  $n$ ,  $k$  và  $a_i$ ,  $i = 1 \dots n$ . Hãy xác định dãy tháp hài hòa dài nhất, chỉ ra tháp đầu tiên và độ dài của dãy tìm được. Nếu tồn tại nhiều dãy cùng độ dài thì chỉ ra dãy tháp có vị trí đầu nhỏ nhất. Nếu không tồn tại dãy tháp thì đưa ra một số 0.

**Dữ liệu:** Vào từ file văn bản CUBICS.INP:

- Dòng đầu tiên chứa 2 số nguyên  $n$  và  $k$ ,
- Dòng thứ 2 chứa  $n$  số nguyên  $a_1, a_2, \dots, a_n$ .

**Kết quả:** Đưa ra file văn bản CUBICS.OUT trên một dòng 2 số nguyên: độ dài của dãy tìm được và số thứ tự của tháp đầu tiên hoặc một số 0 nếu không tồn tại dãy.

**Ví dụ:**

CUBICS.INP
5 3
1 2 3 4 6

CUBICS.OUT
3 2

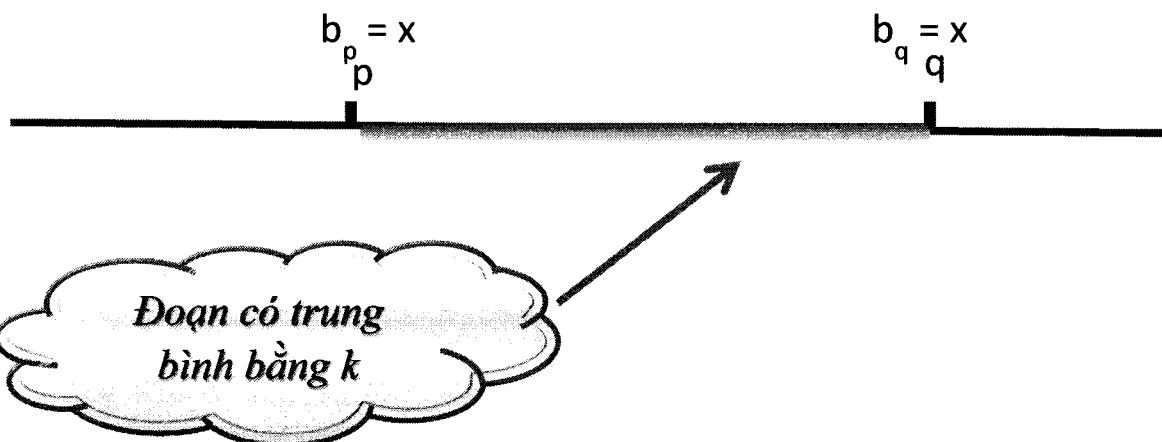
## Gải thuật:

Phân tích giải thuật:

- \* Thay  $a_i \rightarrow a_i - k$
- \* Tính  $b_i = \sum_{j=1}^i a_j$
- \*  $i = 1 \div n$
- \* Nhận xét: nếu  $b_p = b_q$  ( $p < q$ ) thì đoạn  $[p+1, q]$  có trung bình là  $k$ .

Nhiệm vụ:

Tìm  $p, q$  có  $q - p$  là max

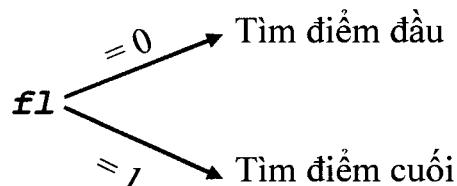


Tổ chức dữ liệu:

- ➔ Mảng  $f[100002]$ : // pair<int,int> f[100002]; dự trữ chỗ cho giá trị hàng rào  $f[n+1]$ ,
- ➔  $f_0.first = 0$ ;  $f_0.second = 0$ ;
- ➔  $f_i.first = \sum_{j=1}^i (a_j - k)$ , //  $\rightarrow f_i = f_{i-1} + a_i - k$ ;
- ➔  $f_i.second = i$ .

Như vậy, không cần thiết phải lưu trữ mảng A.

- ➔ ix: độ dài max của dãy thỏa mãn điều kiện cần tìm, ix – chỉ số phần tử đầu của dãy đó (int ix, ix;), giá trị đầu:  $ix = 0$ ;
- ➔ fl – con lắc



- ➔ Giá trị đầu:  $fl = 0$ ;

Các bước giải thuật:

- ◆ Chuẩn bị, // nhập n, k và khởi tạo các giá trị đầu,
- ◆ Tính giá trị mảng F, giá trị hàng rào  $f[n+1]$  – sẽ xác định sau!
- ◆ Sắp xếp  $f[0..n]$  theo giá trị tăng dần,  $f[n+1].first = f[n].first + 1$ ;
- ◆ Tìm các khoảng thỏa mãn: duyệt  $i = 0 \div n$ :
  - Tiêu chuẩn xác định điểm đầu:  $fl == 0 \&& f[i].first == f[i+1].first$ ,
    - Xử lý điểm đầu:  $ib = f[i].second$ ;
    - $fl^=1$ ;
  - Tiêu chuẩn xác định điểm cuối:  $fl==1 \&& f[i].first != f[i+1].first$ ,
    - Xử lý điểm cuối:  $ie = f[i].second$ ;
    - $d = ie - ib$ ; if ( $d > lx$ ) { $lx = d$ ;  $ix = ib$ ;};
    - $fl^=1$ ;
- ◆ Đưa ra kết quả: nếu  $lx == 0 \rightarrow$  đưa ra 0, trong trường hợp ngược lại: đưa ra  $lx$  và  $ix+1$ .

Độ phức tạp của giải thuật:  $O(n\log n)$  // do có sắp xếp.

*Chương trình giải:*

```
#include <fstream>
using namespace std;
ifstream fi ("CUBICS.INP");
ofstream fo ("CUBICS.OUT");
int n,k,a,d,lx,ix,ib,ie,f1;
pair<int,int> f[100002];

void chbi()
{l1x=0; f1=0;
f[0].first=0; f[0].second=0;
fi>>n>>k;
}

void get_f()
{for(int i=1;i<=n;++i)
{fi>>a; f[i]=make_pair(f[i-1].first+a-k,i);}
}

void xly()
{for(int i=0;i<=n;++i)
if(f1==0 && f[i].first==f[i+1].first){ib=f[i].second;f1^=1;}
else if(f1==1 && f[i].first != f[i+1].first)
{ie=f[i].second; d=ie-ib; f1^=1;
if(d>l1x){lx=d; ix=ib;}
}
}

int main()
{chbi();
get_f();
sort(f,f+n+1); f[n+1].first=f[n].first+1;
xly();
if(lx==0)fo<<0;else fo<<lx<<" "<<ix+1;
}
```

## SD02. BÓNG RỔ

Tên chương trình: BASKET\_B.???

Steve theo dõi một trận đấu bóng rổ trong giải NBA. Trận đấu kéo dài đúng 48 phút. Sau khi kết thúc, trên màn hình điện tử hiện lên bảng thống kê cho biết lúc mỗi đội ghi điểm trong trận đấu. Thông tin mỗi dòng có dạng

***k mm:ss***

trong đó ***k*** – đội ghi điểm, ***k= 1 hoặc 2***, mỗi lần ghi được 1 điểm,  
***mm*** – phút, ***ss*** – giây : thời điểm ghi điểm,  $0 \leq \text{mm} \leq 47$ ,  $0 \leq \text{ss} \leq 59$ .

Steve muốn tính tổng thời gian mỗi đội vươn lên dẫn trước.

**Yêu cầu:** Cho ***n*** ( $1 \leq n \leq 100$ ) và các dòng thông báo. Hãy tính tổng thời gian mỗi đội vươn lên dẫn trước.

**Dữ liệu:** Vào từ file văn bản BASKET\_B.INP:

- Dòng đầu tiên chứa số nguyên ***n***,
- Mỗi dòng trong ***n*** dòng sau chứa một thông báo theo dạng đã nêu.

**Kết quả:** Đưa ra file văn bản BASKET\_B.OUT:

- Dòng thứ nhất: tổng thời gian đội thứ nhất dẫn đầu,
- Dòng thứ hai: tổng thời gian đội thứ hai dẫn đầu.

Thông tin trên mỗi dòng có quy cách ***mm:ss***.

**Ví dụ:**

BASKET_B.INP	BASKET_B.OUT
3	20:00
1 01:10	16:30
2 21:10	
2 31:30	



**Gải thuật:**

**Phân tích giải thuật:**

- ➔ Thời gian cho ở dạng vật lý (phút – giây) – cần chuyển sang dạng tuyệt đối: số giây tính từ đầu trận,
- ➔ Lập bảng đánh dấu thời điểm ghi bàn của mỗi đội,
- ➔ Cập nhật bảng thành bảng cho biết giây thứ *i* đội nào dẫn đầu,
- ➔ Thông kê lại số giây dẫn đầu của mỗi đội và đưa kết quả ra file.

**Tổ chức dữ liệu:**

- ➔ Mảng int f[2880], f<sub>i</sub> cho biết giây thứ *i* đội nào dẫn đầu,
- ➔ Xâu s (chỉ cần độ dài 6) để nhập dữ liệu.

Các bước giải thuật:

- Ghi nhận dữ liệu vào: Nhập và ghi nhận thời điểm ghi bàn:
- Nhập đan xen số nguyên và xâu: **fi>>k>>s**;
- Tính thời điểm tuyệt đối: đổi từ cơ số 60 sang thập phân:  
 $t=((s[0]-48)*10+s[1]-48)*60+(s[3]-48)*10+s[4]-48;$
- Đội 1 ghi bàn ( $k=1$ ):  $f[t]=1$ ; trong trường hợp ngược lại  $-f[t] = -1$ ;
- Xây dựng bảng tích lũy:  $f_i = f_i + f_{i-1}$ ,  $i = 1 \div 2879$ .  $f_i > 0$  nếu đội 1 dẫn bàn,  $f_i < 0$  nếu đội 2 dẫn bàn,
- Đưa ra kết quả: Đổi lại từ cơ số 10 sang cơ số 60.
- Chú ý: *cách đưa ra các số 0 không có nghĩa*.

Độ phức tạp giải thuật:  $O(n)$ .

*Chương trình giải:*

```
#include <fstream>
#include <string>
#include <iomanip>
using namespace std;
ifstream fi ("Basket_b.inp");
ofstream fo ("Basket_b.out");
int f[2882]={0},k,n,t,t1,t2;
string s,s2;

void xl_dg()
{s.reserve(6);
fi>>k>>s;
t=((s[0]-48)*10+s[1]-48)*60+(s[3]-48)*10+s[4]-48;
if(k==1)f[t]=1; else f[t]=-1;
}

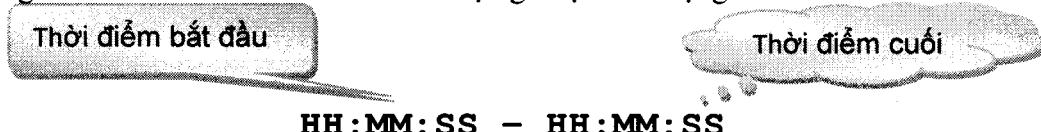
void get_ans(int x)
{int tg1,tg2;
tg1=x/60; tg2=x%60;

fo<<setw(2)<<setfill('0')<<tg1<<':'<<setw(2)<<setfill('0')<<tg2
<<'\n';
}

int main()
{fi>>n;
for(int i=1;i<=n;++i) xl_dg();
for(int i=1;i<=2879;++i)f[i]+=f[i-1];
t1=0;t2=0;
for(int i=0;i<=2879;++i) {if(f[i]>0)++t1; if(f[i]<0)++t2;}
get_ans(t1);
get_ans(t2);
}
```

*Cách đưa ra số 0 không có nghĩa*

Người ta tiến hành điều tra thống kê tình hình xem ti vi trong dân chúng. Mỗi người trong số  $n$  người được hỏi yêu cầu cho biết thời điểm người ta bắt đầu xem ti vi và thời điểm mà hết cuối giây đó người ta rời khỏi ti vi. Câu trả lời được ghi lại dưới dạng:



**HH:MM:SS – HH:MM:SS**

Trong đó  $0 \leq \text{HH} \leq 23$ ,  $0 \leq \text{MM}, \text{SS} \leq 59$ . Hai thời điểm đầu và cuối có thể không cùng một ngày, ví dụ, một người có thể bắt đầu xem từ 23:45:30 cho đến 01:15:00 sáng ngày hôm sau. Dĩ nhiên, không ai xem ti vi liên tục cả một ngày.

Sau khi đã có đủ dữ liệu, các nhà thống kê bắt đầu phân tích.

Độ phổ biến của một giây nào đó được tính bằng tổng số người có xem ti vi ở giây đó. Độ phổ biến của một khoảng thời gian được tính bằng tổng độ phổ biến của các giây trong khoảng đó chia cho tổng số giây trong khoảng.

Hãy tính độ phổ biến của mỗi khoảng trong số  $q$  khoảng cho trước mà các nhà thống kê quan tâm.

**Dữ liệu:** Vào từ file văn bản TV.INP:

- Dòng đầu tiên chứa số nguyên  $n$  ( $1 \leq n \leq 10^5$ ),
- Mỗi dòng trong  $n$  dòng sau chứa một câu trả lời của người được hỏi,
- Dòng thứ  $n+2$  chứa số nguyên  $q$  ( $1 \leq q \leq 10^5$ ),
- Mỗi dòng trong  $q$  dòng sau chứa một khoảng thời gian thống kê, ghi theo quy cách như câu trả lời khi điều tra.

**Kết quả:** Đưa ra file văn bản TV.OUT, kết quả ứng với mỗi khoảng thống kê đưa ra trên một dòng dưới dạng số thực độ chính xác không ít hơn  $10^{-6}$ .

**Ví dụ:**

TV.INP
<b>5</b>
00:00:00 – 00:00:01
00:00:01 – 00:00:03
00:00:00 – 00:00:02
00:00:05 – 00:00:09
00:00:06 – 00:00:06
<b>5</b>
00:00:00 – 00:00:03
00:00:07 – 00:00:09
00:00:06 – 00:00:06
00:00:05 – 00:00:09
00:00:00 – 00:00:09

TV.OUT
<b>2.0000000000</b>
<b>1.0000000000</b>
<b>2.0000000000</b>
<b>1.2000000000</b>
<b>1.4000000000</b>



Chương trình giải:

```
#include <fstream>
#include <string>
#include <iomanip>

using namespace std;
int n,q,b[86400],tb,te;
string s;
ifstream fi ("TV.INP");
ofstream fo ("TV.OUT");

void nhapdl()
{ s.reserve(19);
  getline(fi,s);
  tb=(((s[0]-48)*10+s[1]-48)*60+(s[3]-48)*10+s[4]-48)*60+(s[6]-48)*10+s[7]-48;
  te=(((s[11]-48)*10+s[12]-48)*60+(s[14]-48)*10+s[15]-48)*60+(s[17]-48)*10+s[18]-48;
}

void gn_be()
{b[tb]++;
 if (te==86399) return;
 if (te<tb)b[0]++;
 b[++te]--;
}

void tichluy()
{int t;
 t=b[0];
 for (int i=1;i<86400;i++)
 { t+=b[i];
   b[i]=(b[i-1]+t);
 }
}
float xd_r()
{ float x;
 x=b[te];
 if (tb==0) return(x);
 if (tb<=te) return(x-b[tb-1]);
 return(x+b[86399]-b[tb-1]);
}
void xd_kq()
{//float r,d;
 double r,d;
if (tb>te) d=86400-tb+te+1; else d=te-tb+1;
r= xd_r()/d;
fo<<showpoint<<fixed<<setprecision(10)<<r<<endl;
}

int main()
{memset(b,0,sizeof(b));
fi>>n; getline(fi,s);
```

Cách đưa ra đúng 10 chữ  
số sau dấu chấm thập phân

```
for (int i=1;i<=n;i++)
    {nhapdl();
     gn_be();
    }
tichluy();
fi>>q;  getline(fi,s);
for (int i=1;i<=q;i++)
    {nhapdl();
     xd_kq();
    }
fi.close(); fo.close();
}
```

## UF38. CHÁO VÀ PHỞ

Tên chương trình: GRUEL???

Hội khỏe Phù Đổng năm nay có một môn thi mới do Đoàn thanh niên phụ trách: các trường mở quán ăn sáng giới thiệu món ăn đặc sản vùng miền mình. Quán nào thu hút được nhiều khách đến ăn nhất sẽ thắng.

Quán ăn của một trường có khả năng thắng cuộc cung cấp cho khách hàng 2 món cháo và phở. Theo quy định của Ban Tổ chức, mỗi khách chỉ được ăn một món ở một quán. Mỗi khách ăn cháo chỉ cần dùng một chiếc thìa còn khách ăn phở phải dùng một thìa và một đũa. Vì là quán ăn nghiệp dư nên số thìa và đũa không nhiều lắm: chỉ có  $n$  cái thìa và  $m$  đũa. Nếu một khách nào đó đến gọi món mà không còn đủ thìa hoặc đũa cần cho món đó thì họ sẽ bỏ sang quán khác.

Sáng nay có  $k$  khách đăng ký tới quán. Người thứ  $i$  tới lúc  $t_i$ , gọi món ăn  $a_i$ ,  $a_i = 0$  – gọi cháo,  $a_i = 1$  – gọi phở. Nếu được phục vụ họ sẽ ngồi ăn trong khoảng thời gian  $d_i$ . Không có khách nào cùng đến quán một lúc. Việc rửa thìa đũa được tổ chức rất tốt, nên nếu một khách đi ra đúng vào thời điểm khác mới tới thìa đũa của khách trước được rửa sạch và phục vụ được ngay cho khách mới.

**Yêu cầu:** Hãy xác định những khách nào được phục vụ và khách nào sẽ phải đi nơi khác. Với những khách được phục vụ – đưa ra thông báo “**Yes**”, với khách bị từ chối – đưa ra thông báo “**No**”.

**Dữ liệu:** Vào từ file văn bản GRUEL.INP:

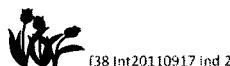
- Dòng đầu tiên chứa 3 số nguyên  $n$ ,  $m$  và  $k$  ( $1 \leq n, m, k \leq 10^4$ ),
- Dòng thứ  $i$  trong  $k$  dòng sau chứa 3 số nguyên  $t_i$ ,  $d_i$  và  $a_i$  ( $1 \leq t_i, d_i \leq 10^5$ ), thông tin được đưa theo thứ tự tăng dần của  $t_i$ .

**Kết quả:** Đưa ra file văn bản GRUEL.OUT đưa ra các thông báo “**Yes**” hoặc “**No**”, mỗi thông báo trên một dòng. Dòng  $i$  tương ứng với khách thứ  $i$  ( $i = 1 \div k$ ).

**Ví dụ:**

GRUEL.INP
3 1 3
1 3 1
2 2 0
3 5 1

GRUEL.OUT
Yes
Yes
No



## 43L. CÁCH MẠNG

Tên chương trình: REVOLUTIONS.???

Trận chiến quyết định giữa Neo và Smith được ấn định vào lúc 19.00. Do sự cố mạng nên Smith đến chậm. Tranh thủ thời gian này, Neo quan sát dãy điệp viên Smith đang đứng dàn trận thành một hàng ngang.

Khi Smith biến một người nào đó thành điệp viên giống mình không phải mọi mã đều bị thay đổi. Với kinh nghiệm quan sát của mình Neo có thể xác định được điệp viên này có xuất xứ là nam hay nữ. Cuộc chiến trước trên tàu đã cho Neo biết là sẽ hết sức vất vả nếu số số điệp viên xuất xứ từ nam là vượt trội. Neo quyết định tìm đoạn phòng thủ có thể dễ chọc thủng hơn cả.

Một đoạn phòng thủ có thể chọc thủng với độ tốt  $k$  là đoạn A bắt đầu bằng nữ và kết thúc là nam, chứa trong đó đoạn có độ tốt  $k-1$  với các điểm đầu – cuối không trùng với điểm đầu – cuối của A và không chứa trong đó các đoạn độ tốt lớn hơn hoặc bằng  $k$ .

**Yêu cầu:** Cho 2 số nguyên  $n$ ,  $k$  và xâu  $s$  độ dài n chỉ chứa các ký tự 0 và 1, trong đó  $n$  – độ dài dãy điệp viên,  $s_i = '0'$  tương ứng với vị trí nữ và  $s_i = '1'$  tương ứng với vị trí nam. Hãy xác định số đoạn có độ tốt  $k$ .

**Dữ liệu:** Vào từ file văn bản REVOLUTIONS.INP:

- ◆ Dòng đầu tiên chứa số nguyên  $n$  – số lượng tests,
- ◆ Mỗi test cho trên 2 dòng:
  - ◆ Dòng thứ nhất chứa 2 số nguyên  $n$  và  $k$  ( $2 \leq n \leq 10^5$ ,  $1 \leq k \leq n/2$ ).
  - ◆ Dòng thứ 2 chứa xâu  $s$ .

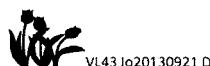
Tổng số điệp viên trong file không vượt quá  $10^5$ .

**Kết quả:** Đưa ra file văn bản REVOLUTIONS.OUT, kết quả mỗi test đưa ra trên một dòng dưới dạng số nguyên.

**Ví dụ:**

REVOLUTIONS.INP
2
3 1
110
3 1
011

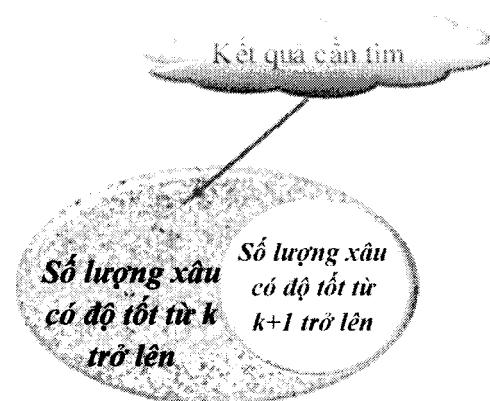
REVOLUTIONS.OUT
0
2



## Giải thuật

Nguyên lý chung giải các bài toán đếm số lượng đối tượng:

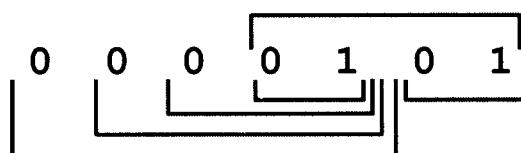
- ◆ Nguyên tắc “*Cá kề đầu, rau kề mớ*”: xác định một khóa nhận dạng – một giá trị biến hoặc một điều kiện nào đó và kiểm tra xem tập đang xét có chứa giá trị hay thỏa mãn điều kiện nhận dạng,
- ◆ Đưa bài toán từ nhiều biến di động về một số bài toán bài toán với số biến di động ít hơn (thông thường là từ bài toán với 2 đầu di động về việc giải 2 lần bài toán một đầu di động).



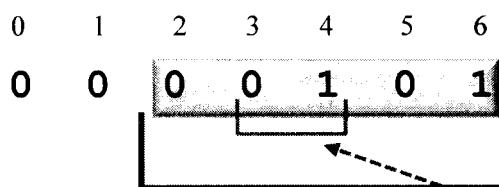
Mô hình toán học của bài toán đang xét:

- ◆ Cho xâu bít  $S$  độ dài  $n$  và số nguyên  $k$ ,
- ◆ Yêu cầu xác định số lượng xâu con các bít liên tiếp thỏa mãn các điều kiện:
  - Bắt đầu bằng 0 và kết thúc bằng 1,
  - Có số lượng bít 0 không ít hơn  $k$ ,
  - Nếu xâu con bắt đầu từ vị trí  $i$  và kết thúc ở vị trí  $j$  ( $i < j$ ) thì khoảng  $[i+1, j-1]$  không chứa các xâu con thỏa mãn 2 điều kiện trên.

Ví dụ: với  $S = 0000101$ ,  $n = 7$  và  $k = 1$  ta có 6 xâu con thỏa mãn:



Xâu con từ vị trí 2 đến 6 không thỏa mãn vì chứa xâu con  $s_3s_4$  độ tốt 1.



*Không thỏa mãn vì chứa xâu con độ tốt 1*

Nhân của giải thuật là hàm  $s1(m)$  xác định số lượng xâu con có độ tốt không nhỏ hơn  $m$ . Gọi  $t1$  là tổng số bít 1 trong xâu,  $t0$  – số lượng bít 0 từ đầu xâu đến vị trí đang xét. Duyệt tất cả các bít từ 0 đến  $n-k$ . Với mỗi bít 0 gấp trên quá trình duyệt:

- ◆ Tăng số lượng  $t0$ ,

- ★ Kiểm tra xem bít này có thể là đầu của một xâu có độ tốt không ít hơn m hay không,
- ★ Nếu bít 0 này có thể là điểm đầu thì tính số lượng điểm cuối tương ứng có thể chọn và tích lũy vào kết quả tìm kiếm,

Lời giải cần tìm của bài toán là hiệu  $sl(k) - sl(k+1)$ .

Chương trình giải:

```
#include <fstream>
#include <iostream>
#include <string>
using namespace std;
int n,m,k,ib,ie,ls,t,t0,t1,res,ans;
string s;
ifstream fi ("Revolutions.inp");
ofstream fo ("Revolutions.out");

int sl(int m)
{int tt1;
tt1=t1;res=0;t0=0;
for(int i=0;i<ls;++i)
    if(s[i]=='0') {++t0; if(t0>=m){if(tt1-m+1>0) res+=(tt1-m+1);}}
    else --tt1;
return res;
}

int main()
{fi>>m;
for(int j=1;j<=m;++j)
{fi>>n>>k;
fi>>s;
ls=s.size();
t0=0;t1=0;
for(int i=0;i<ls;++i) if(s[i]=='1') ++t1;
ans=sl(k)-sl(k+1);
fo<<ans<<'\n';
}
}
```

## SVC14. BẢN ĐỒ GEN

Tên chương trình: GENMAP.???

Các cá thể được tạo ra bằng công nghệ biến đổi gen khi đưa ra nhân giống đại trà bằng phương pháp sinh sản hữu tính dần dần mất đi một số đặc tính quý báu có ở các thế hệ ban đầu. Vấn đề ở chỗ là các cá thể thế hệ mới không giữ được trọn vẹn các gen quý của bố và mẹ. Bản đồ gen của mỗi cá thể được biểu diễn dưới dạng xâu ký tự  $\mathbf{S}$  chỉ chứa các ký tự la tinh in thường, mỗi ký tự đại diện cho một gen.

Nếu bản đồ gen của mẹ / bố là  $\mathbf{S}_p$ , (cá thể thế hệ  $F_1$ ) và bản đồ gen của con sinh ra trực tiếp từ cá thể này (thế hệ  $F_2$ ) là  $\mathbf{S}_c$  thì  $\mathbf{S}_c$  có các tính chất sau:

- ◆  $\mathbf{S}_c$  có  $m$  ký tự đầu giống  $m$  ký tự đầu của  $\mathbf{S}_p$ ,
- ◆  $\mathbf{S}_c$  có  $m$  ký tự cuối giống  $m$  ký tự cuối của  $\mathbf{S}_p$ .

Nói một cách khác  $\mathbf{S}_c$  có tiền tố độ dài  $m$  trùng khớp với tiền tố độ dài  $m$  của  $\mathbf{S}_p$  và  $\mathbf{S}_c$  có hậu tố độ dài  $m$  trùng khớp với hậu tố độ dài  $m$  của  $\mathbf{S}_p$ . Nếu  $k$  là giá trị lớn nhất của các  $m$  thỏa mãn hai điều kiện trên thì cặp bản đồ  $\mathbf{S}_p$  và  $\mathbf{S}_c$  có “độ ổn định di truyền  $k$ ”.

Trên cánh đồng thực nghiệm hiện có  $n$  cây đánh số từ 1 đến  $n$ , cây thứ  $i$  có bản đồ gen là  $\mathbf{S}_i$ .  $i = 1 \div n$ . Người ta cần chọn một cặp cá thể có độ ổn định di truyền  $k$  để nghiên cứu.

Hãy xác định  $q$  – số cặp khác nhau có thể lựa chọn. Hai cặp gọi là khác nhau nếu tồn tại một cây có ở cặp này và không có ở cặp kia.

**Dữ liệu:** Vào từ file văn bản GENEMAP.INP:

- ◆ Dòng đầu tiên chứa 2 số nguyên  $n$  và  $k$  ( $2 \leq n \leq 10^5$ ,  $1 \leq k \leq 200$ ),
- ◆ Dòng thứ  $i$  trong  $n$  dòng sau chứa xâu  $\mathbf{S}_i$ , mỗi xâu có độ dài không quá 200.

**Kết quả:** Đưa ra file văn bản GENEMAP.OUT một số nguyên là phần dư của  $q$  chia cho  $10^9 + 7$ .

**Ví dụ:**

GENEMAP .INP
5 2
aaaaaa
aabdecaa
aaaa
bbcaa
bbaaehaa

GENEMAP.OUT
3



### *Giải thuật:*

◆ Biến đổi xâu, đưa về bài toán chỉ *làm việc với tiền tố* (*Prefix*):

◆ Với phép biến đổi trên từ xâu S ta có xâu U (độ dài gấp đôi):

◆  $S = (s_0, s_1, s_2, \dots, s_{m-2}, s_{m-1})$



◆  $U = (s_0, s_{m-1}, s_1, s_{m-2}, \dots, s_{m-2}, s_1, s_{m-1}, s_0)$

◆ Với 2 xâu X và Y, bằng phép biến đổi trên ta có 2 xâu U và V,

◆ Độ ổn định di truyền bằng một nửa số lượng các ký tự  $u_i$  và  $v_i$  giống nhau ở prefix:  $u_i = v_i$ ,  $i = 0 \div 2 \times k - 1$ .

◆ Từ dãy xâu  $S_i$  ban đầu xây dựng dãy xâu  $U_i$  theo cách biến đổi trên ( $i = 1 \div n$ ),

◆ Sắp xếp  $\{U_i\}$  theo thứ tự tăng dần,

◆ Các xâu ứng với nhóm các cặp có  $k > 0$  đứng thành một nhóm liên tiếp nhau,

◆ Xây dựng hàm  $sl(2^*q)$  xác định số cặp có  $k \geq q$ :

◆ Gọi  $r$  – số lượng cặp cần xác định,

◆ Nhóm các xâu đứng liên tiếp nhau có  $k \geq q$  có ít nhất  $2^*q$  ký tự đầu giống nhau là p thì  $r = p^*(p - 1)/2 \% 1000000007$ ;

◆ Kết quả  $res = sl(2^*k) - sl(2^*k+2)$ .

◆ Việc sắp xếp có độ phức tạp  $O(nlgn)$ ,

◆ Tính  $sl(x)$  – độ phức tạp  $O(n)$ ,

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

```

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <inttypes.h>
#define MAX 100000
#define MAXS 201
#define BASE 1000000007

char fni[]="GeneMap.Inp"; //N: numstring, K: do khop,
char fno[]="GeneMap.Out";

int N, K;

int64_t res;
char aGen[MAX] [2*MAXS];

void Input()
{
    int i;
    FILE *f = fopen(fni, "rt");

    fscanf(f, "%i %i", &N, &K);
    for (i=0; i<N; i++)
        fscanf(f, "%s", aGen[i]);
    fclose(f);
}

void Output()
{
    FILE *f = fopen(fno, "wt");
    fprintf(f, "%ld", (long)res);
    fclose(f);
}

void MakePalindrom(char *s)
{
    int64_t i, j, len=strlen(s);
    int64_t last = len-1;
    char st[2*MAXS];

    for (i=j=0; i<len; i++, j+=2)
    {
        st[j] = s[i];
        st[j+1] = s[last-i];
    }
    st[2*len] = 0;
    strcpy(s, st);
}

void DupData()
{
    for (int64_t i=0; i<N; i++)
        MakePalindrom(aGen[i]);
}

int fcmp(const void* s1, const void* s2)
{
    return strcmp((const char *)s1, (const char *)s2);
}

```

```

}

int64_t CountPairStable(int len)
{
    int64_t i, T=0, subT, m;
    for (i=0; i<N-1; i++)
    {
        if (strlen(aGen[i])>=len)
        {
            for (m = 1; (strlen(aGen[i+1])>=len) && (!strncmp(aGen[i],
aGen[i+1], len)); m++, i++);
            subT = m*(m-1) / 2 % BASE;
            T = (T+subT) % BASE;
        }
    }
    return T;
}

void Solve()
{
    DupData();
    qsort((void *)aGen, N, sizeof(aGen[0]), fcmp);
//    res = CountPairStable(2*K) - CountPairStable(2*K+2);
    int64_t res1 = CountPairStable(2*K);
    int64_t res2 = CountPairStable(2*K+2);
    res = res1 - res2;
}

int main()
{
    Input();
    Solve();
    Output();
    return 0;
}

```

## 2.1 – Cây chỉ số nhị phân (Binary Indexed Trees)

### Mở đầu

Chúng ta thường phải sử dụng một cấu trúc dữ liệu nào đó để giải thuật được thực hiện nhanh hơn. Ở đây chúng ta sẽ xem xét cấu trúc **Cây chỉ số nhị phân**. Theo Peter M. Fenwick, tác giả của cấu trúc này, ban đầu nó được sử dụng để nén dữ liệu. Ngày nay cấu trúc này thường được sử dụng để giải quyết vấn đề lưu trữ tần số xuất hiện và tích lũy tổng tiền tố.

Xét bài toán sau: có  $n$  hộp đựng sỏi. Có thể xuất hiện các truy vấn dạng:

- Bỏ một viên sỏi vào hộp  $i$ ,
- Tính tổng số lượng sỏi trong các hộp từ  $k$  đến  $l$ .

Với cách làm giản đơn thông thường, thời gian xử lý truy vấn loại 1 có độ phức tạp  $O(1)$  và với truy vấn loại 2 – là  $O(n)$ . Giả thiết ta cần thực hiện  $m$  truy vấn và tất cả đều là loại 2. Khi đó thời gian xử lý sẽ là  $O(m * n)$ . Sử dụng một số loại cấu trúc dữ liệu ta có thể giải quyết vấn đề này với độ phức tạp  $O(m \log n)$  trong trường hợp xấu nhất. Ở đây ta sẽ xét cách giải quyết dựa trên Cây chỉ số nhị phân. Tuy cách giải quyết này cũng có độ phức tạp  $O(m \log n)$  trong trường hợp xấu nhất nhưng tốn ít bộ nhớ hơn và rất dễ lập trình.

### Ký hiệu

- **BIT** – Cây chỉ số nhị phân (Binary Indexed Tree),
- **MaxVal** – Giá trị cực đại có tần số xuất hiện khác 0,
- $f[i]$  – tần số xuất hiện của giá trị chỉ số  $i$ ,  $i = 1 \div \text{MaxVal}$ ,
- $c[i]$  – tổng tiền tố tần số xuất hiện của chỉ số  $i$  (tức là  $f[1] + f[2] + \dots + f[i]$ ),
- $\text{tree}[i]$  – tổng tần số lưu trong **BIT** với chỉ số  $i$  (sẽ giải thích khái niệm chỉ số rõ hơn ở phần tiếp theo) và ta sẽ nói một cách ngắn gọn là tần số cây,
- $\text{num}~$  - số đảo bít nhị phân nhận được từ num ( $0 \rightarrow 1, 1 \rightarrow 0$ ).

Lưu ý: thường phải gán  $f[0]=0$ ,  $c[0]=0$ ,  $\text{tree}[0]=0$ , nhưng nói chung ta sẽ bỏ qua việc xét chỉ số 0.

### Nguyên lý cơ sở

Mỗi số nguyên được biểu diễn như một tổng các lũy thừa của 2. Như vậy tổng tiền tố cũng có thể được biểu diễn như tổng các phần tử của tập chứa các tổng tiền tố con. Ở đây ta sẽ xét trường hợp các tổng con này không giao nhau.

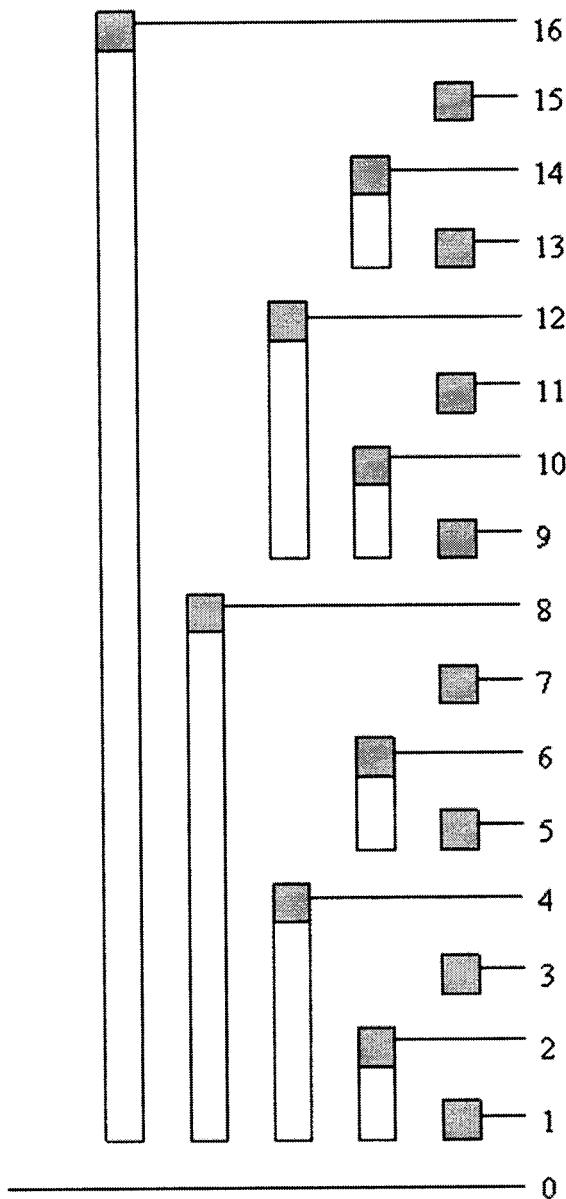
Idx là một chỉ số nào đó của **BIT**, **r** là vị trí bít 1 cuối cùng(tính từ trái sang phải) trong dạng biểu diễn nhị phân của idx. **tree[ idx ]** là tổng các tần số từ chỉ số (**idx**- $2^x+1$ ) cho tới chỉ số **idx** (xem bảng 2.1 để hiểu rõ hơn). Ta nói idx tương ứng với các chỉ số từ (**idx**- $2^x+1$ ) đến **idx** (điều này sẽ phù hợp với việc dùng nó là khóa trong giải thuật và là đối tượng trong quá trình xử lý cây).

	<b>1</b>	<b>2</b>	<b>3</b>	<b>4</b>	<b>5</b>	<b>6</b>	<b>7</b>	<b>8</b>	<b>9</b>	<b>10</b>	<b>11</b>	<b>12</b>	<b>13</b>	<b>14</b>	<b>15</b>	<b>16</b>
<b>f</b>	1	0	2	1	1	3	0	4	2	5	2	2	3	1	0	2
<b>c</b>	1	1	3	4	5	8	8	12	14	19	21	23	26	27	27	29
<b>tree</b>	1	1	2	4	1	4	0	12	2	7	2	11	3	4	0	29

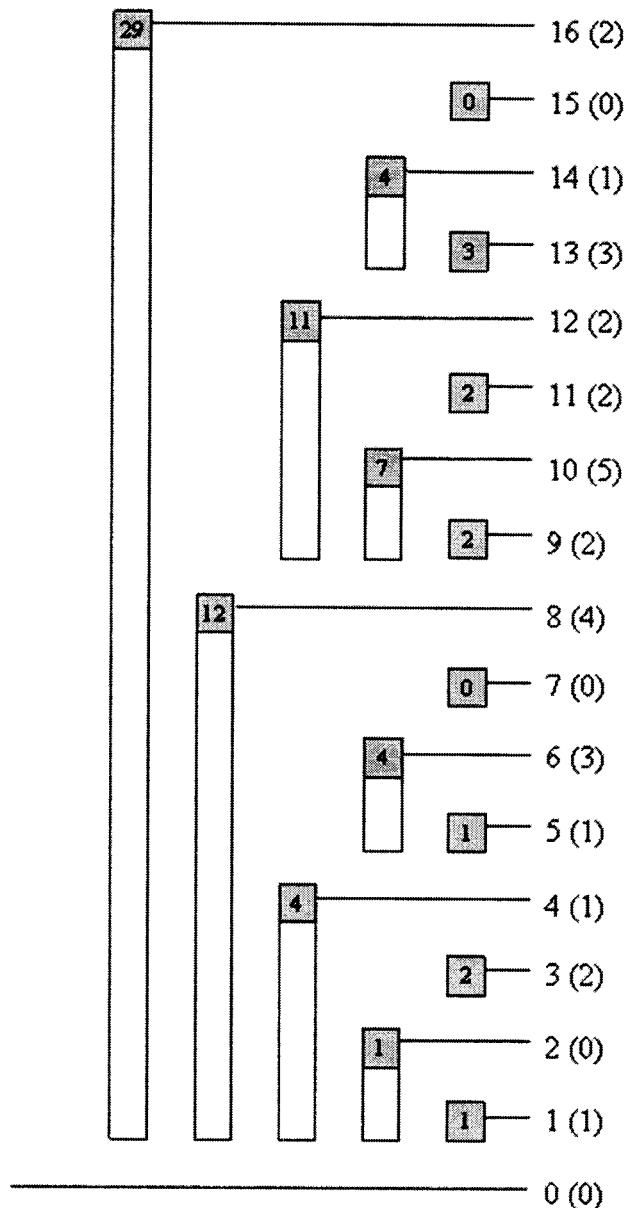
Bảng 2.1

	<b>1</b>	<b>2</b>	<b>3</b>	<b>4</b>	<b>5</b>	<b>6</b>	<b>7</b>	<b>8</b>	<b>9</b>	<b>10</b>	<b>11</b>	<b>12</b>	<b>13</b>	<b>14</b>	<b>15</b>	<b>16</b>
<b>tree</b>	1	1..2	3	1..4	5	5..6	7	1..8	9	9..10	11	9..12	13	13..14	15	1..16

Bảng 2.2 – Bảng phạm vi quản lý của nút BIT



Hình 2.3 – Cây BIT tương ứng (hình chữ nhật chỉ phạm vi quản lý của nút)



Hình 2.4 – Nút cây và tổng tiền tố

Giả thiết ta cần có tổng tiền tố của chỉ số 13 (tổng của 13 phần tử đầu tiên). Do  $13_{10} = 1101_2$ , vì vậy  $c[1101] = \text{tree}[1101] + \text{tree}[1100] + \text{tree}[1000]$ .

### Tách bít 1 cuối cùng

Thuật toán luôn đòi hỏi biết chữ số khác 0 cuối cùng, vì vậy cần xây dựng giải thuật hiệu quả tách bít này. Xét số nguyên dương  $\text{num}$ . Ở dạng biểu diễn nhị phân  $\text{num} = \mathbf{a}1\mathbf{b}$ , trong đó  $\mathbf{a}$  – dãy số nhị phân (có thể rỗng),  $\mathbf{b}$  – dãy số 0 liên tiếp cuối cùng trong biểu diễn nhị phân của  $\text{num}$ .

$$\text{Ta có } -\text{num} = (\mathbf{a}1\mathbf{b})^{\sim} + 1 = \mathbf{a}^{\sim}0\mathbf{b}^{\sim} + 1$$

Do  $b$  là dãy số 0 nên  $\tilde{b}$  là dãy số 1, từ đó có:

$$-\text{num} = (\mathbf{a} \mathbf{l} \mathbf{b})^{\sim} + 1 = \mathbf{a}^{\sim} \mathbf{0} \tilde{\mathbf{b}} + 1 = \mathbf{a}^{\sim} \mathbf{0}(0\dots0)^{\sim} + 1 = \mathbf{a}^{\sim} \mathbf{0}(1\dots1) + 1 = \mathbf{a}^{\sim} \mathbf{1}(0\dots0) = \mathbf{a}^{\sim} \mathbf{1} \mathbf{b}.$$

Như vậy, với  $\text{num}$  và  $-\text{num}$  ta có thể dễ dàng tách được bít 1 cuối cùng bằng phép **AND** (tức là phép **&** trong C/C++, Java):

$$\begin{array}{r} \mathbf{a} \mathbf{l} \mathbf{b} \\ \& \mathbf{a}^{\sim} \mathbf{1} \mathbf{b} \\ \hline = (0\dots0) \mathbf{1}(0\dots0) \end{array}$$

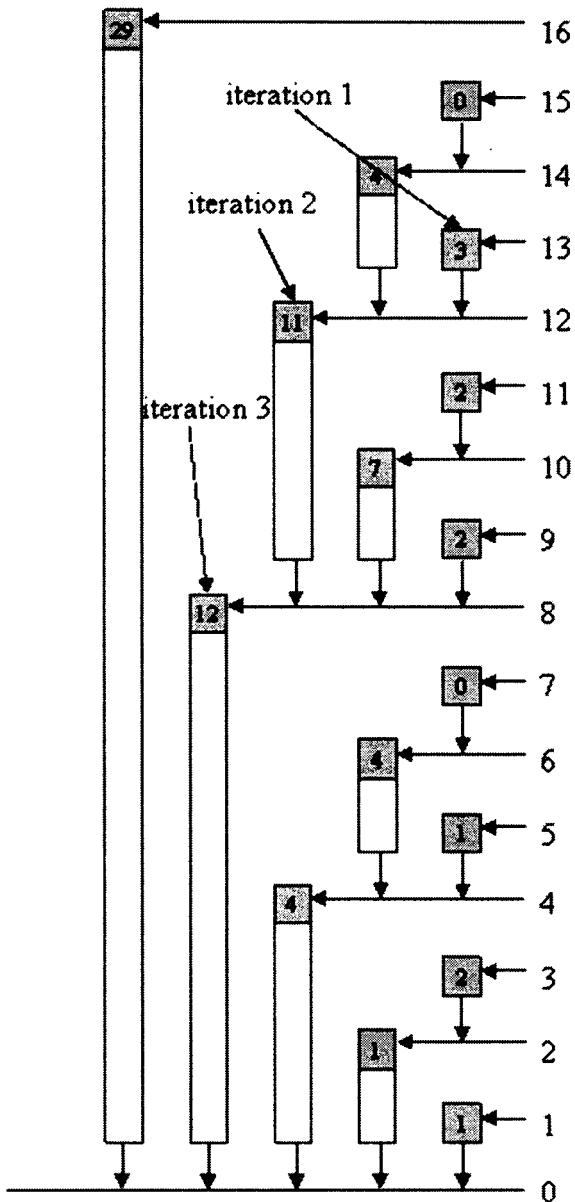
### Tính tổng tiền tố

Để tính tổng tiền tố cho một chỉ số nguyên dương  $\text{idx}$  nào đó ta cần cộng  $\text{tree}[\text{idx}]$  vào biến  $\text{sum}$ , xóa bít 1 cuối cùng của  $\text{idx}$  (tức là thay nó bằng 0) lặp lại việc cộng chừng nào  $\text{idx}$  còn lớn hơn 0. Hàm **read** dưới đây thực hiện công việc đó.

```
int read(int idx){  
  
    int sum = 0;  
  
    while (idx > 0){  
  
        sum += tree[idx];  
  
        idx -= (idx & -idx);  
    }  
  
    return sum;  
}
```

Ví dụ, với  $\text{idx} = 13$ ,  $\text{sum} = 0$  ta có:

Bước lặp	idx	Vị trí bít 1 cuối cùng	idx & -idx	sum
1	13 = 1101	0	1 (2 ^0)	3
2	12 = 1100	2	4 (2 ^2)	14
3	8 = 1000	3	8 (2 ^3)	26
4	0 = 0	---	---	---



Hình 2.5 – Mô hình chỉ phạm vi chỉ số tính tổng (hình minh họa cho chỉ số 13)

Kết quả nhận được sẽ là 26. Số lần lặp tối đa là  $\log \text{MaxVal}$ . Như vậy độ phức tạp của giải thuật là  $O(\log \text{MaxVal})$ , chương trình không đến mươi câu lệnh!

### Thay đổi tần số ở một vị trí và cập nhật cây

Để cập nhật cây khi giá trị một nút nào đó thay đổi, ta cần phải lùi ngược từ nút đó về nút gốc (có giá trị lớn nhất: **MaxVal**). Nếu khi tính tổng tiền tố ta phải đi từ nút đó về 0 thì khi tăng tần số ở một nút, thêm giá trị val thì cũng làm tương tự, nhưng đi từ nút đó về gốc.

```
void update(int idx ,int val){  
    while (idx <= MaxVal){
```

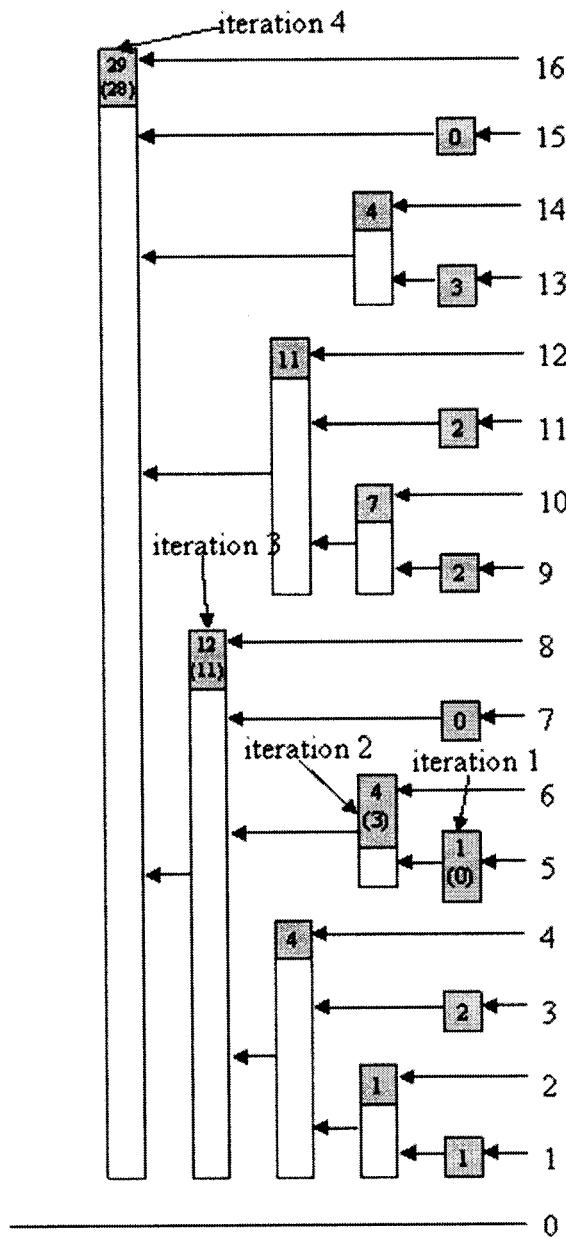
```

        tree[idx] += val;
        idx += (idx & -idx);
    }
}

```

Ví dụ, với **idx** = 5:

Bước lặp	idx	Vị trí 1 cuối cùng	idx & -idx
1	5 = 101	0	1 ( $2^0$ )
2	6 = 110	1	2 ( $2^1$ )
3	8 = 1000	3	8 ( $2^3$ )
4	16 = 10000	4	16 ( $2^4$ )
5	32 = 100000	---	---



Hình 2.6 – Cập nhật cây (trong ngoặc là giá trị trước khi cập nhật); mũi tên chỉ phần cây được cập nhật từ index đến **MaxVal** (hình minh họa cho chỉ số 5)

Hình 2.6 mô tả quá trình cập nhật **BIT**.

Độ phức tạp của giải thuật là  $O(\log \text{MaxVal})$ , số câu lệnh cũng không nhiều!

### Tính giá trị tần số ở một vị trí cụ thể

Giá trị  $\text{tree}[idx]$  không phản ánh giá trị tần số ở điểm  $idx$ . Để có giá trị cần tìm ta có thể lưu trữ riêng, trực tiếp các giá trị này. Nhưng như vậy sẽ tốn thêm bộ nhớ và đòi hỏi phải cập nhật đồng bộ dữ liệu khi xử lý.

Giá trị cần tìm có thể nhận được bằng 2 lần gọi hàm read đã nêu ở trên:

$$f[idx] = \text{read}(idx) - \text{read}(idx-1).$$

Độ phức tạp vẫn là  $O(\log n)$  (chính xác là  $2 \times O(\log n)$ ), tuy lớn hơn  $O(1)$ , nhưng không đáng kể.

Nếu hai đường từ hai chỉ số về gốc có phần chung, thì ta có thể tính tổng cho đến nút chung đầu tiên của 2 đường đường, trừ tổng lưu được và nhận được giá trị tần

số cần tìm giữa hai chỉ số. Đây là việc làm khá đơn giản để tính tổng tần số giữa các chỉ số liền kề, hoặc tính tần số tại một chỉ số nhất định.

Gọi chỉ số cần xác định giá trị là  $x$  và nút cha của nó là  $y$ . Nếu  $y$  ở dạng biểu diễn nhị phân là  $a0b$ , trong đó  $b$  – dãy số 1 thì  $x$  sẽ là  $a1b^{\sim}$  (dễ dàng thấy rằng  $b^{\sim}$  là dãy số 0). Áp dụng thuật toán sum đã nêu với tham số  $x$ . Ở bước đầu tiên tẩy xóa 1 cuối cùng trong  $x$  và  $x$  trở thành  $a0b^{\sim}$ . Gọi giá trị này là  $z$ .

Quá trình tương tự cũng được thực hiện với  $y$ . Áp dụng hàm tính  $\text{sum}$  đã nêu, ta lần lượt xóa từng số 1 cuối cùng, sau một số bước  $y$  từ  $a0b$  trở thành  $a0b^{\sim}$ , tức là bằng  $z$ . Lưu ý là cần xét riêng trường hợp  $x = 0$ .

```

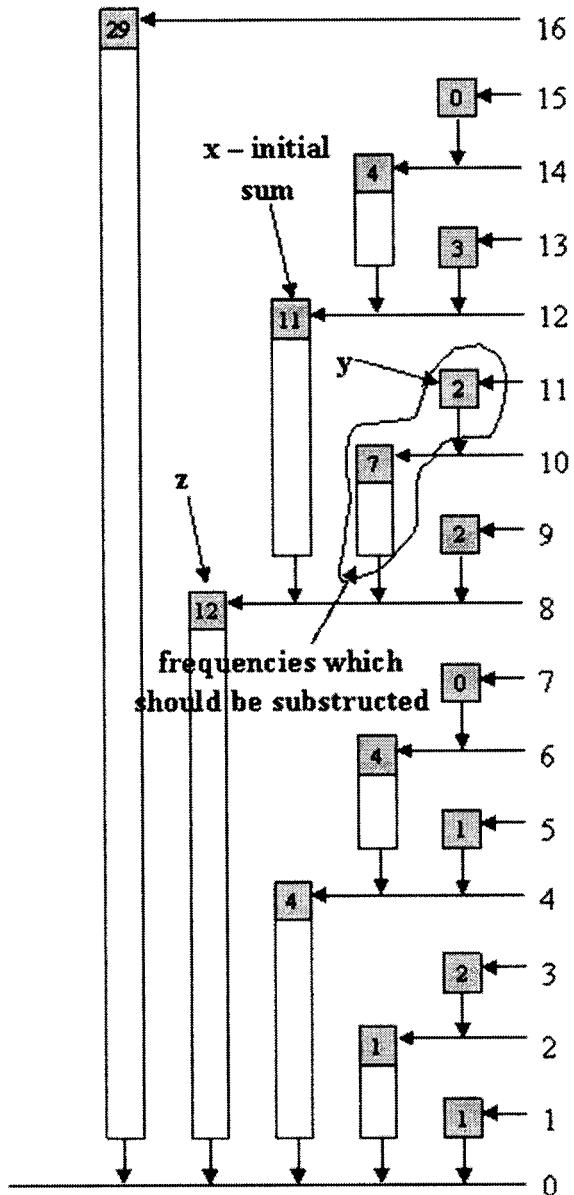
int readSingle(int idx){
    int sum = tree[idx]; // sum will be decreased
    if (idx > 0){ /* special case
        int z = idx - (idx & -idx); // make z first
        idx--; /* idx is no important any more, so instead y, you can use idx
        while (idx != z){ /* at some iteration idx (y) will become z
            sum -= tree[idx];
        /* subtract tree frequency which is between y and "the same path"
            idx -= (idx & -idx);
        }
    }
    return sum;
}

```

Dưới đây là kết quả minh họa cho trường hợp  $idx = 12$ .

Đầu tiên cần tính  $z = 12 - (12 \& -12) = 8$ ,  $\text{sum} = 11$ .

Bước lặp	y	Vị trí 1 cuối cùng	y & -y	sum
1	11 = 1011	0	1 (2 ^0)	9
2	10 = 1010	1	2 (2 ^1)	2
3	8 = 1000	---	---	---



Hình 2.7 – Tính tần số tại một chỉ số cụ thể trong BIT  
(trên hình minh họa trường hợp  $idx = 12$ )

So với cách tính 2 lần gọi hàm `read` đã nêu ở trên thì cách tính trực tiếp này sẽ nhanh hơn. Nếu  $idx$  là số lẻ thì chỉ thuật toán làm việc với độ phức tạp  $O(1)$ , không có lần lặp nào. Còn nếu  $idx$  chẵn thì độ phức tạp sẽ là  $c \times O(\log idx)$ , trong đó  $c < 1$ . Với cách tính  $\text{read}(idx) - \text{read}(idx-1)$  độ phức tạp sẽ là  $c_1 \times O(\log idx)$ , trong đó chắc chắn  $c_1 > 1$ .

Độ phức tạp chung của giải thuật là  $O(\log \text{MaxVal})$ , số câu lệnh của chương trình là không quá 15.

### Tỷ lệ hóa toàn cây theo một thừa số hằng

Nhiều khi ta cần tăng tất cả các giá trị tần số lên  $c$  lần. Khi đó các giá trị của nút trong cây sẽ được cập nhật theo công thức: với nút  $idx$  giá trị mới sẽ là:

$$-(c-1) \times \text{readSingle}(idx)/c$$

bởi vì  $f[idx] - (c-1) \times f[idx]/c = f[idx]/c$ .

```
void scale(int c) {
    for (int i = 1 ; i <= MaxVal ; i++)
        update(-(c - 1) * readSingle(i) / c , i);
}
```

Cách xử lý này có độ phức tạp  $O(\text{MaxVal} \times \log \text{MaxVal})$ . Ta còn có thể làm điều này nhanh hơn nếu lưu ý là phép nhân có tính giao hoán, kết hợp và phân phối. Độ phức tạp của giải thuật sẽ là  $O(\text{MaxVal})$  và chỉ phải viết có vài câu lệnh:

```
void scale(int c) {
    for (int i = 1 ; i <= MaxVal ; i++)
        tree[i] = tree[i] / c;
}
```

## Tìm chỉ số theo tổng tiền tố cho trước

Giải thuật thô là vét cạn, duyệt tất cả các chỉ số, tính tổng tiền tố cho đến khi gặp giá trị cho trước. Nếu có tần số âm thì đây là cách duy nhất. Nhưng trong lớp bài toán này thì tần số là không âm, khi chỉ số tăng, tổng tiền tố không giảm, vì vậy có thể cải tiến việc tìm kiếm nhị phân để có một giải thuật hiệu quả. Ta có thể duyệt tất cả các bít bắt đầu từ bít 1 trái nhất, tạo chỉ số  $idx$ , tính, so sánh kết quả và điều chỉnh chỉ số  $idx$ .

```
// if in tree exists more than one index with a same value
// cumulative frequency, this procedure will return
// some of them (we do not know which one)
// bitMask - initially, it is the greatest bit of MaxVal
```

```

// bitMask store interval which should be searched

int find(int cumFre){

    int idx = 0; // this var is result of function

    while ((bitMask != 0) && (idx < MaxVal)){ // nobody likes overflow :)

        int tIdx = idx + bitMask; // we make midpoint of interval

        if (cumFre == tree[tIdx]) // if it is equal, we just return idx

            return tIdx;

        else if (cumFre > tree[tIdx]){

            // if tree frequency "can fit" into cumFre,
            // then include it

            idx = tIdx; // update index

            cumFre -= tree[tIdx]; // set frequency for next loop

        }

        bitMask >>= 1; // half current interval

    }

    if (cumFre != 0) // maybe given cumulative frequency doesn't exist

        return -1;

    else

        return idx;

}

// if in tree exists more than one index with a same
// cumulative frequency, this procedure will return
// the greatest one.

int findG(int cumFre){

```

```

        int idx = 0;

        while ((bitMask != 0) && (idx < MaxVal)) {
            int tIdx = idx + bitMask;

            if (cumFre >= tree[tIdx]) {
                // if current cumulative frequency is equal to cumFre,
                // we are still looking for higher index (if exists).
                idx = tIdx;
                cumFre -= tree[tIdx];
            }
            bitMask >>= 1;
        }

        if (cumFre != 0)
            return -1;
        else
            return idx;
    }
}

```

Hoạt động của hàm **find** với tổng tiền tố là 21:

<b>Bước lặp 1</b>	tIdx là 16; tree[16] lớn hơn 21; giảm một nữa bitMask và tiếp tục
<b>Bước lặp 2</b>	tIdx là 8; tree[8] nhỏ hơn 21, ta phải tăng 8 chỉ số đầu tiên ở kết quả, lưu idx vì biết chắc đó là một phần của kết quả; trừ cumFre đi một lượng tree[8] (ta không muốn xét lại các tổng tiền tố đã qua và đi duyệt tổng tiền tố khác trong cây); giảm một nữa bitMask và tiếp tục
<b>Bước lặp 3</b>	tIdx là 12; tree[12] lớn hơn 9 (tại đây không có cách tác động lên khoảng 1-8, nhưng tác động được lên khoảng 1-16); giảm một nữa bitMask và tiếp tục
<b>Bước lặp 4</b>	tIdx là 10; tree[10] nhỏ hơn 9, ta phải cập nhật giá trị; giảm một nữa bitMask và tiếp tục
<b>Bước lặp 5</b>	tIdx là 11; tree[11] bằng 2; trả về index (tIdx)

Độ phức tạp của giải thuật là  $O(\log \text{MaxVal})$ . Chương trình chứa không quá 20 dòng lệnh.

## BIT 2 chiều

BIT có thể được tổ chức dưới dạng cấu trúc dữ liệu nhiều chiều. Giả thiết ta có một tập điểm có tọa độ nguyên không âm được đánh dấu trên mặt phẳng. Xét 3 loại truy vấn:

- ◆ Đánh dấu điểm tọa độ  $(x, y)$ ,
- ◆ Xóa đánh dấu điểm tọa độ  $(x, y)$ ,
- ◆ Đếm số lượng điểm được đánh dấu trong hình chữ nhật cạnh song song với trục tọa độ có góc dưới trái ở  $(0, 0)$  và trên phải ở  $(x, y)$ .

Giả thiết có  $m$  truy vấn, tọa độ  $x$  có giá trị lớn nhất là  $\max_x$ , tọa độ  $y$  có giá trị lớn nhất là  $\max_y$ . Vấn đề có thể được giải quyết với độ phức tạp  $O(m \times \log(\max_x) \times \log(\max_y))$ . Trong trường hợp này mỗi phần tử của cây là một mảng, tổng thể ta cần mảng 2 chiều  $\text{tree}[\max_x][\max_y]$ . Việc cập nhật theo mỗi tọa độ được thực hiện theo thuật toán đã xét ở trên. Ví dụ, hàm **update** với 3 tham số cho phép xác lập hay xóa điểm tọa độ  $(x, y)$  có lời gọi **update(x, y, c)**, trong đó  $c = 1$  là xác lập điểm  $(x, y)$ ,  $c = -1$  – xóa điểm đó.

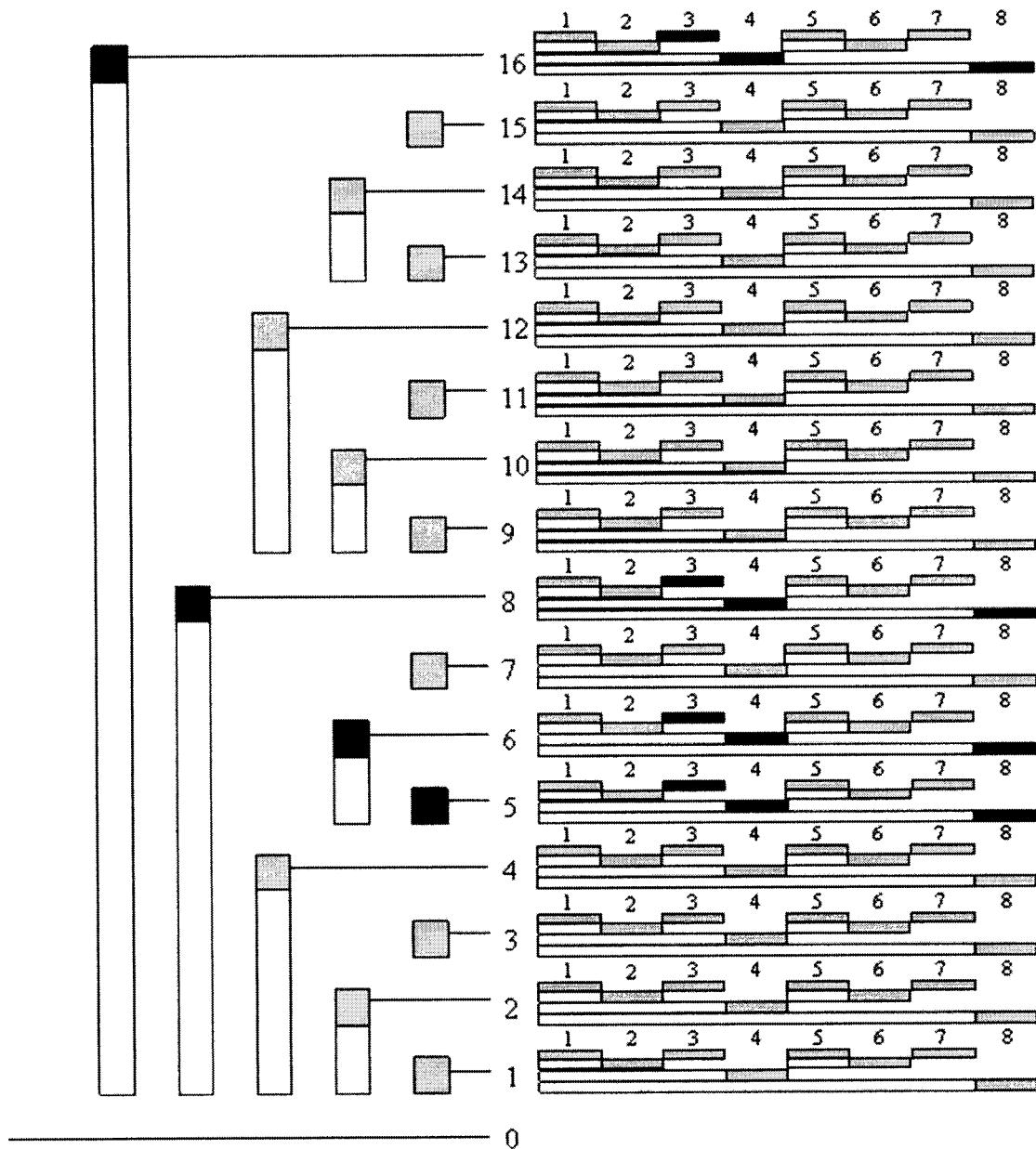
Có thể tổ chức 2 hàm riêng hoặc gộp chung trong một hàm.

Cách tổ chức 2 hàm riêng:

```
void update(int x , int y , int val){  
    while (x <= max_x){  
        tree[x][y] += val;  
        updatey(x , y , val);  
        x += (x & -x);  
    }  
}  
  
void updatey(int x , int y , int val){  
    while (y <= max_y){  
        tree[x][y] += val;  
        y += (y & -y);  
    }  
}
```

Cách tổ chức một hàm chung:

```
void update(int x , int y , int val){  
    int y1;  
  
    while (x <= max_x) {  
  
        y1 = y;  
  
        while (y1 <= max_y) {  
  
            tree[x][y1] += val;  
  
            y1 += (y1 & -y1);  
        }  
  
        x += (x & -x);  
    }  
}
```



Hình 2.8 - BIT BIT 2D (kích thước 16 x 8).  
Màu xanh đánh dấu các trường cần cập nhật khi xử lý chỉ số (5, 3).

Việc sửa đổi các hàm khác cũng được thực hiện tương tự.

Về lý thuyết, có thể tổ chức **BIT n** chiều.

## Bài tập ứng dụng:

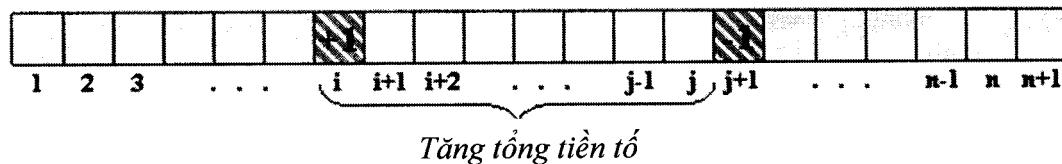
### Bài 1

Có  $n$  lá bài đánh số từ 1 đến  $n$ , sắp thành một dãy trên bàn. Một số lá bài ở trạng thái úp, một số khác – trạng thái ngửa. Có nhiều truy vấn thuộc một trong 2 loại có thể thực hiện:

1.  $T(i, j)$  – đảo trạng thái các lá bài từ  $i$  tới  $j$  (úp thành ngửa và ngược lại),
2.  $Q(i)$  – xác định trạng thái lá bài thứ  $i$  (đưa ra 0 là sấp, 1 là ngửa).

*Giải thuật:*

Với mỗi truy vấn thời gian xử lý là  $O(\log n)$ . Tạo mảng  $f[n+1]$  lưu thông tin truy vấn loại 1. Với mỗi truy vấn  $T(i, j)$  xác lập  $f[i]++$  và  $f[j+1]--$ . Với mỗi lá bài  $k$  trong khoảng  $[i, j]$  tổng  $f[1]+f[2]+\dots+f[k]$  sẽ tăng lên 1, các tổng còn lại – giữ nguyên (xem hình 3.0). Giá trị cần tìm là tổng theo mô đun



2.

Hình 3.0

## EP14. SỐ NGHỊCH THẾ

Tên chương trình: INVERS.???

Xét dãy số nguyên  $\mathbf{A} = (\mathbf{a}_1, \mathbf{a}_2, \dots, \mathbf{a}_n)$  ( $1 \leq n \leq 500\,000$ ). Các số trong dãy  $\mathbf{A}$  khác nhau từng đôi một và nhận giá trị trong phạm vi từ 1 đến  $n$ . Như vậy dãy  $\mathbf{A}$  là một hoán vị các số từ 1 đến  $n$ . Cặp số  $(\mathbf{a}_i, \mathbf{a}_j)$  trong dãy  $\mathbf{A}$  được gọi là một nghịch thế, nếu  $i < j$  và  $\mathbf{a}_i > \mathbf{a}_j$ .

**Yêu cầu:** Cho  $n$  và hoán vị  $\mathbf{A}$ . Hãy xác định số nghịch thế.

**Dữ liệu:** Vào từ file văn bản INVERS.INP:

- Dòng đầu tiên chứa số nguyên  $n$ ,
- Dòng thứ 2 chứa  $n$  số nguyên xác định hoán vị  $\mathbf{A}$ .

**Kết quả:** Đưa ra file văn bản INVERS.OUT một số nguyên – số lượng nghịch thế.

**Ví dụ:**

INVERS.INP	INVERS.OUT
5	5
2 4 3 5 1	



p14 - INVERS

*Đây là bài toán chuẩn. Giải thuật tối ưu giải quyết bài toán này chứa tất cả các đoạn codes mẫu (fragments) cho mọi chương trình sử dụng cấu trúc cây Fenwick. Chính vì vậy, nên bắt đầu từ bài toán cơ sở này, dùng lại khá kỹ với những cách tiếp cận khác nhau để học sinh hiểu phương thức tìm lời giải cho một bài toán tin học, cách nâng cấp, cải tiến giải thuật.*

Nhân tiện trong phần này ta cũng cung cấp cho học sinh các đoạn chương trình tạo tests, qua đó:

- ◆ Khắc phục tâm lý ngại tạo tests và tự kiểm tra chương trình ở học sinh,
- ◆ Làm cho học sinh “tâm phục, khẩu phục” các cấu trúc dữ liệu mới và giải thuật mới!

Các giải thuật tầm thường (trivial) cũng hết sức quan trọng trong quá trình **đào tạo** và sau đó – **bồi dưỡng** học sinh năng khiếu.

*Vai trò của các giải thuật tầm thường:*

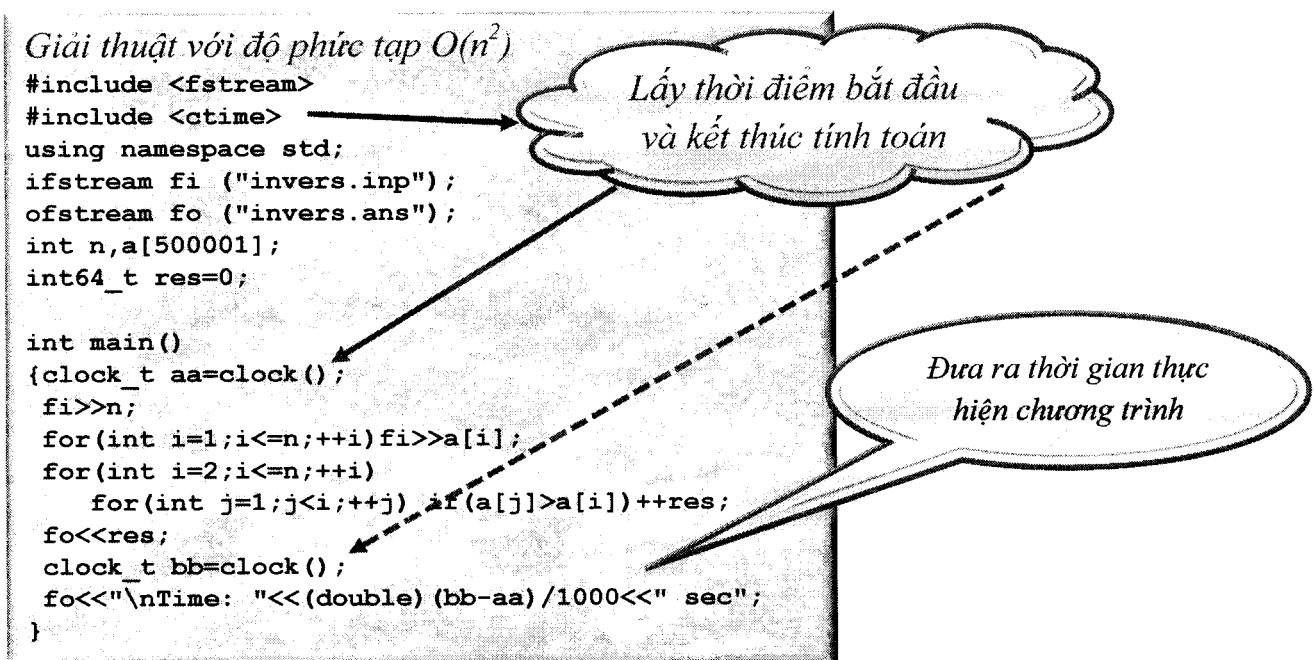
- ◆ Là “phao cứu sinh” khi không nghĩ ra giải thuật tốt hơn hay không đủ thời gian để tìm giải thuật tốt,
- ◆ Chỉ ra các khâu yếu, kém hiệu quả trong giải thuật hiện tại, từ đó biết được trọng tâm cần lưu ý để cải tiến, nâng cấp hiệu quả của giải thuật.

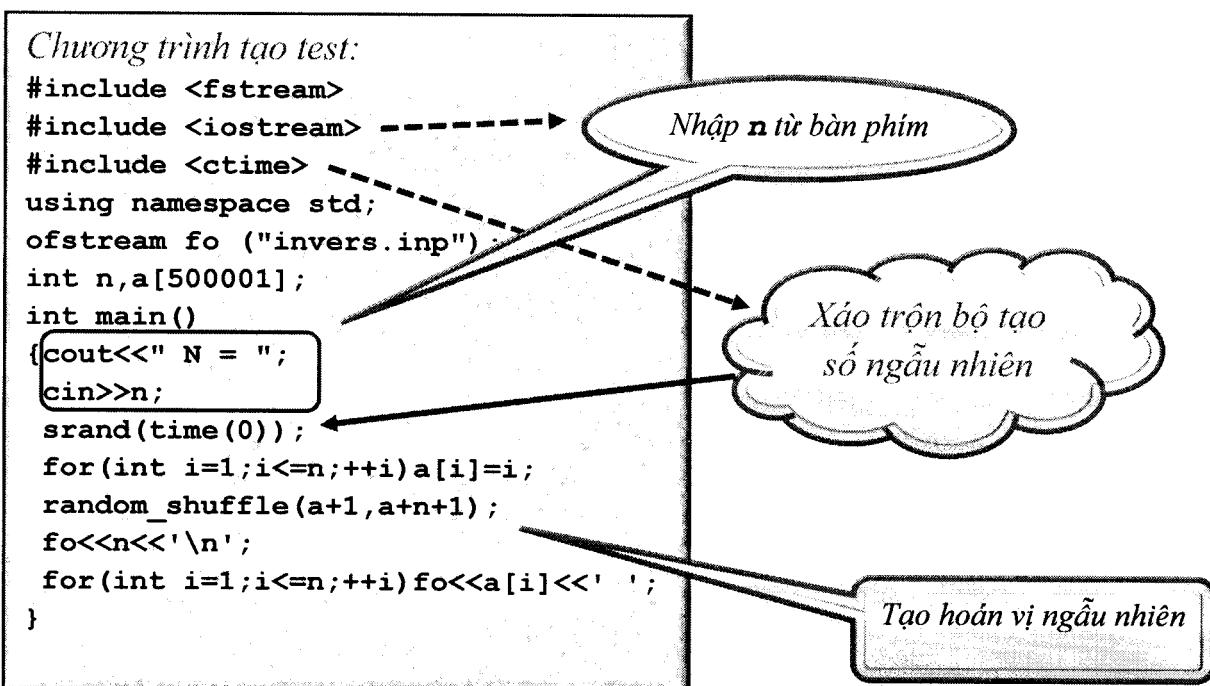
Tuy nhiên trong tài liệu này, trong phần lớn các trường hợp ta sẽ không để cập nhiều tới các giải thuật tầm thường để không làm loãng nội dung chủ yếu là xây dựng các giải thuật hiệu năng cao.

### Giải thuật tầm thường:

Với mỗi  $a_i$ ,  $i = 2 \div n$  kiểm tra xem có bao nhiêu  $j$  thỏa mãn các điều kiện  $j < i$  và  $a_j > a_i$ . Chương trình giải đơn giản, chứa 2 chu trình lồng nhau và có độ phức tạp  $O(n^2)$ . Để tiện so sánh, đánh giá độ phức tạp giải thuật ta sẽ đưa ra thời gian thực hiện chương trình.

Để so sánh với các giải thuật khác, kết quả ở chương trình này sẽ được đưa ra file INVERS.ANS.





Với  $n$  lớn giải thuật trên không hiệu quả vì 2 lý do:

- ◆ Thông tin nhận được khi xử lý mỗi  $a_i$  ( $i = 2 \div n$ ) không được lưu lại hỗ trợ cho việc xử lý giá trị tiếp theo,
- ◆ Kết quả res được tích lũy một cách thô thiển: lần lượt tăng 1 khi gặp cặp nghịch thé.

Muốn nâng cao hiệu quả của giải thuật cần phải quản lý được thông tin về các dữ liệu đã xét và dùng nó để hỗ trợ cho việc xử lý tiếp theo. Trên cơ sở đó kết quả res có thể được tích lũy theo bước lớn hơn 1. Điều này cũng sẽ làm giảm đáng kể thời gian thực hiện giải thuật.

Cấu trúc dữ liệu cây Fenwick giúp chúng ta đồng thời giải quyết được cả 2 yêu cầu trên với hiệu quả đủ cao và chi phí lập trình không lớn. Tồn tại những cấu trúc dữ liệu còn mang lại hiệu quả cao hơn nhưng đòi hỏi bộ nhớ lớn và chi phí lập trình rất cao!

Ứng dụng cây Fenwick – Giải thuật A. Có lưu trữ mảng A :

```
#include <fstream>
#include <ctime>
using namespace std;
ifstream fi ("inv.inp");
ofstream fo ("inv.out");
int n,a[500001];
int64_t t,res=0,s[500001]={0};

void insert_t(int ii)
{while(ii<=n){s[ii]++;ii+=(ii&(-ii));}

int sum_t(int ii)
{int64_t r;
r=0;
while (ii>0){r+=s[ii];ii&=(ii-1);}
return(r);
}

int main()
{fi>>n;
for(int i=1;i<=n;++i)fi>>a[i];
insert_t(a[n]);
for(int i=n-1;i>=1;--i)
{ t=sum_t(a[i]-1);
insert_t(a[i]);
res+=t;
}
fo<<res;
}
```

Hai hàm cơ sở phục vụ xử lý cây BIT

Giải thích nguyên tắc xử lý:

Các phần tử của mảng A được nạp vào cây theo trình tự ngược từ cuối về đầu:

$$a_n \rightarrow a_{n-1} \rightarrow a_{n-2} \rightarrow \dots \rightarrow a_2 \rightarrow a_1$$

Trước khi nạp  $a_i$  ( $i = n-1 \div 1$ ): kiểm tra xem trong cây đã có bao nhiêu phần tử nhỏ hơn  $a_i$  đã có trong cây, đó là số cặp nghịch thế hình thành với  $a_i$  trong



Sử dụng cây Fenwick để tính tổng tiền tố

khoảng từ vị trí i đến vị trí n. Giá trị tìm được cần tích lũy vào kết quả res.

### So sánh hiệu quả

Với mỗi n cố định, thời gian thực hiện chương trình không thay đổi nhiều đối với giải thuật tầm thường. Khi số cặp nghịch thế tăng, thời gian thực hiện chương sẽ tăng do tăng số lần thực hiện phép tích lũy kết quả ++res. Độ phức tạp của giải thuật:  $O(n^2)$ .

Thời gian thực hiện chương trình theo giải thuật sử dụng cây Fenwick chỉ phụ thuộc vào n, không phụ thuộc vào chính hoán vị. Độ phức tạp của giải thuật:  $O(n \ln(n))$ .

Kết quả thực hiện với hoán vị ngẫu nhiên và n = 500 000 trên máy processor

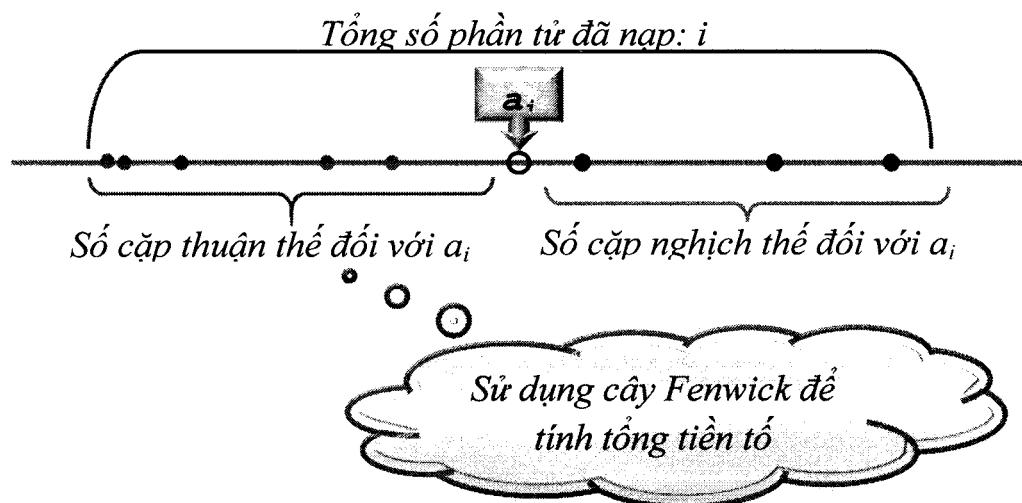
Giải thuật tầm thường	Giải thuật sử dụng cây Fenwick
22152138410	22152138410
Time: 410.024 sec	Time: 0.266 sec
<i>Chênh lệch 1 541 lần</i>	

Co\_i7:

Ứng dụng cây Fenwick – Giải thuật B. Không lưu trữ mảng A :

Các phần tử của mảng A được nạp vào cây theo trình tự lần lượt xuất hiện từ đầu dãy về cuối:

$$a_1 \rightarrow a_2 \rightarrow a_3 \rightarrow \dots \rightarrow a_{n-1} \rightarrow a_n$$



```

#include <fstream>
#include <ctime>
using namespace std;
ifstream fi ("invers.inp");
ofstream fo ("invers.out");
int n,a,s[500001]={0};
int64_t t,res=0;

void insert_t(int ii)
{while(ii<=n){s[ii]++;ii+=(ii&(-ii));}
}

int sum_t(int ii)
{int64_t r;
r=0;
while (ii>0){r+=s[ii];ii&=(ii-1);}
return(r);
}

int main()
{clock_t aa=clock();
fi>>n; fi>>a;insert_t(a);
for(int i=2;i<=n;++i)
{fi>>a;
insert_t(a);t=sum_t(a);
res+=(i-t);
}
fo<<res;
clock_t bb=clock();
fo<<"\nTime: "<<(double)(bb-aa)/1000<<" sec";
}

```

## EP37. DÂY CUNG

Tên chương trình: CHORDS.???

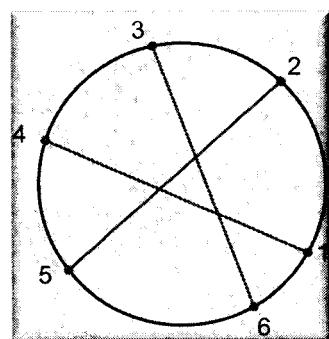
Có  $2n$  điểm khác nhau được đánh dấu trên đường tròn. Các điểm được đánh số từ 1 tới  $2n$  theo chiều ngược kim đồng hồ ( $1 \leq n \leq 100\ 000$ ).

Rسم  $n$  dây cung, dây thứ  $i$  nối hai điểm  $a_i$  và  $b_i$ . Mỗi điểm đã cho chỉ thuộc đúng một dây cung.

*Yêu cầu:* Cho  $n$  và các số  $a_i, b_i$  ( $i = 1 \div n$ ). Hãy xác định số cặp dây cung giao nhau.

*Dữ liệu:* Vào từ file văn bản CHORDS.INP:

- Dòng đầu tiên chứa số nguyên  $n$ ,
- Dòng thứ  $i$  trong  $n$  dòng sau chứa hai số nguyên  $a_i$  và  $b_i$ .



*Kết quả:* Đưa ra file văn bản CHORDS.OUT một số nguyên – số cặp dây cung giao nhau.

*Ví dụ:*

CHORDS.INP	CHORDS.OUT
3 1 4 2 5 3 6	3



p37 - Sách "Các bài tập lập trình số" - Nguyễn Văn Hùng

### *Giải thuật A*

- ➔ Dùng mảng cặp dữ liệu  $p = \{u, v\}$ ,  $u < v$  – xác định cung  $(u, v)$ ,
- ➔ Tổ chức 2 mảng  $b[200001]$  và  $c[200001]$ ,  $b_i/c_i$  – số lượng điểm đầu/cuối trong khoảng  $[1, i]$ , thao tác với mảng: xử lý cây Fenwick,
- ➔ Sắp xếp  $p$  theo thứ tự tăng dần,
- ➔ Với  $i = 1 \div n$ :
  - Xác định số cung cắt cung thứ  $i$ :  $t = \text{sum\_t}(b, p[i].second) - \text{sum\_t}(c, p[i].second);$
  - Tích lũy  $t$  vào kết quả,
  - Xóa cung thứ  $i$ : cập nhật lại  $b$  và  $c$ :  $\text{upd\_t}(b, p[i].first); \text{upd\_t}(c, p[i].second);$
- ➔ Độ phức tạp của giải thuật:  $O(n \log n)$ .

### *Chương trình theo giải thuật A:*

```
#include <iostream>
#include <ctime>
using namespace std;
ifstream fi ("chords.inp");
ofstream fo ("chords.out");
int n,n2,u,v,t,b[200001]={0},c[200001]={0};
pair<int,int> p[100001];
int64_t res=0;

void insert_t(int a[],int x)
{while(x<=n2){a[x]++;x+=(x&(-x));}

int sum_t(int a[],int x)
{int tg=0;
 while(x>0){tg+=a[x]; x&=(x-1);}
 return(tg);
```

```

}

void upd_t(int a[], int x)
{while(x<=n2){a[x]--;x+=(x&(-x));} }

int main()
{clock_t aa=clock();
 fi>>n; n2=n<<1;
 for(int i=1;i<=n;++i)
 {fi>>u>>v;
 if(u>v) {t=u; u=v; v=t; }
 p[i].first=u;p[i].second=v;
 insert_t(b,u); insert_t(c,v);
 }
 sort(p+1,p+n+1);
 for(int i=1;i<=n;++i)
 {
 t=sum_t(b,p[i].second)-sum_t(c,p[i].second);
 res+=t;
 upd_t(b,p[i].first); upd_t(c,p[i].second);
 }
 fo<<res;
 clock_t bb=clock();
 fo<<"\nTime: "<<(double)(bb-aa)/1000<<" sec";
}

```

### *Giải thuật B*

Giải thuật trên đủ hiệu quả cần thiết. Tuy vậy vẫn có những khâu xử lý có thể cải tiến để giảm thời gian thực hiện chương trình (mặc dù *không làm thay đổi* việc đánh giá độ phức tạp của giải thuật!)

Trong nhiều trường hợp, việc sắp xếp dữ liệu có thể vòng tránh bằng một bảng đánh dấu. Thay một khâu xử lý có độ phức tạp  $O(n \log n)$  bằng khâu xử lý với độ phức tạp  $O(n)$  là một việc nên làm, tuy nó có thể không làm thay đổi độ phức tạp chung của toàn giải thuật nhưng vẫn góp phần tăng hiệu quả giải thuật nếu nhìn trên góc độ thời gian thực hiện chương trình. Hơn thế nữa, nếu việc sắp xếp được kích hoạt bằng lời gọi hàm **sort** trong C/C++ thì giải thuật sắp xếp được triển khai là *Quick\_Sort*. Giải thuật sắp xếp này có độ phức tạp trung bình là  $O(n \log n)$ , trong trường hợp xấu nhất, độ phức tạp có thể là  $O(n^2)$ ! Còn bảng đánh dấu luôn làm việc với độ phức tạp  $O(n)$ .

Ở bài toán đang xét có thể thay mảng dữ liệu **pair<int,int> p[100001]**

$$d_i = \begin{cases} 0 & \text{nếu } i \text{ không phải là điểm đầu của một cung,} \\ j & \text{nếu tồn tại cung } (i, j) \text{ với } i < j. \end{cases}$$

bằng mảng đánh dấu **int d[200001]**.

Ta chỉ cần xử lý các  $d_i \neq 0$  với  $i$  chạy từ 1 đến  $ux$ , trong đó  $ux$  – điểm đầu có giá trị lớn nhất. Vai trò của  $p[i].second$  được thay thế bằng  $d[i]$ .

Chương trình theo giải thuật B:

```
#include <fstream>
#include <ctime>
using namespace std;
ifstream fi ("chords.inp");
ofstream fo ("chords.out");
int n,n2,u,ux=0,v,t,t1,t2,b[200001]={0},c[200001]={0},d[200001]={0};
int64_t res=0;
void insert_t(int a[],int x)
{while(x<=n2){a[x]++;x+=(x&(-x));}}

int sum_t(int a[],int x)
{int tg=0;
 while(x>0){tg+=a[x]; x&=(x-1);}
 return(tg);
}

void upd_t(int a[], int x)
{while(x<=n2){a[x]--;x+=(x&(-x));} }

int main()
{clock_t aa=clock();
 fi>>n; n2=n<<1;
 for(int i=1;i<=n;++i)
 {fi>>u>>v;
 if(u>v){t=u; u=v; v=t; }
 if(u>ux)ux=u; d[u]=v;
 insert_t(b,u); insert_t(c,v);
 }
 for(int i=1;i<=ux;++i)
 {if(!d[i])continue;
 t=sum_t(b,d[i])-sum_t(c,d[i]);
 res+=t;
 upd_t(b,i); upd_t(c,d[i]);
 }
 fo<<res;
 clock_t bb=clock();
 fo<<"\nTime: "<<(double)(bb-aa)/1000<<" sec";
 }
```

### **So sánh hiệu quả**

Các bộ dữ liệu tests đều có n = 100 000.

Bộ dữ liệu	Kết quả theo giải thuật A	Kết quả theo giải thuật B
I	4152438412 Time: 0.117 sec	4152438412 Time: 0.104 sec
II	3806221005 Time: 0.113 sec	3806221005 Time: 0.107 sec

III	4539760012 Time: 0.111 sec	4539760012 Time: 0.107 sec
IV	4771703787 Time: 0.119 sec	4771703787 Time: 0.112 sec
V	4976658380 Time: 0.117 sec	4976658380 Time: 0.112 sec
VI	4944550901 Time: 0.108 sec	4944550901 Time: 0.102 sec

Giải thuật B, tuy không nhiều lăm nhung luôn luôn làm việc tốt hơn.

## 50K. CUỘC CHIẾN BẢO VỆ HOGVARTS

Tên chương trình: BATLE.???

Hogwarts chuẩn bị chống trả sự tấn công của các thế lực đen tối. Tuyến tiền tiêu ở ngoài cùng có  $n$  vị trí nằm thành một hàng. Các vị trí được đánh số từ 1 đến  $n$  từ trái qua phải, mỗi vị trí có một người. Năng lực chiến đấu của mỗi người có giá trị nguyên, nằm trong phạm vi từ 1 đến  $n$  và khác nhau từng đôi một. Người ở vị trí  $i$  có năng lực chiến đấu là  $a_i$ . Kết quả bố trí các tuyến phòng thủ theo chiều sâu ở phía trong cho thấy toàn bộ hệ thống phòng thủ sẽ phát huy được tối đa sức mạnh của mình khi ở tuyến tiền tiêu năng lực chiến đấu được bố trí tăng dần từ trái qua phải.

Harry được cử đi tổ chức lại tuyến tiền tiêu. Khả năng đánh giá năng lực chiến đấu của từng người ở Harry là rất tốt. Khi thấy một cặp 2 người ở các vị trí cạnh nhau mà người bên trái có năng lực chiến đấu cao hơn người bên phải Harry cho 2 người này đổi chỗ cho nhau. Về lý thuyết, nếu Harry đi duyệt từ đầu đến cuối  $n-1$  lần thì đảm bảo chắc chắn tuyến tiền tiêu sẽ có cấu hình bố trí tối ưu. Đáng tiếc, thời gian không còn nhiều và Harry chỉ kịp đi duyệt có  $k$  lần.

Hãy xác định năng lực chiến đấu của các vị trí ở tuyến tiền tiêu trước khi trận chiến diễn ra.

**Dữ liệu:** Vào từ file văn bản BATLE.INP:

- ◆ Dòng đầu tiên chứa 2 số nguyên  $n$  và  $k$  ( $1 \leq n \leq 2 \times 10^5$ ,  $0 \leq k \leq n-1$ ),
- ◆ Dòng thứ 2 chứa  $n$  số nguyên  $a_1, a_2, \dots, a_n$ .

**Kết quả:** Đưa ra file văn bản BATLE.OUT trên một dòng  $n$  số nguyên – năng lực chiến đấu ở các vị trí từ trái sang phải trên tuyến tiền tiêu sau khi bố trí lại.

**Ví dụ:**

BATLE.INP
4 1
4 1 3 2

BATLE.OUT
1 3 2 4



## *Giải thuật*

$c_i$  – số nghịch thέ của ai tính từ đầu dãy,

$$r_i = 0 \quad \forall i$$

Nếu  $c_i \leq k - a_i$  về vị trí không có nghịch thέ , ngược lại –  $a_i$  sang trái k vị trí  $r_{i-k}=a_i$ , đánh dấu  $b_{a[i]}=-1$  – đã định vị.

Bắt đầu từ phần tử  $t = 1$ : với  $i = 1 \div n$ : nếu  $r_i = 0$  (chưa có kết quả) – tìm phần tử đầu tiên chưa định vị while( $b[t] < 0$ ) $\rightarrow t$ ; đưa phần tử này vào vị trí  $i$ :  $r_i = t$ ; chuyển sang phần tử tiếp theo:  $\rightarrow t$ ;

*Chương trình giải:*

```
#include<iostream>
#include<ctime>
int a[200001], b[200001]={0}, c[200001], r[200001]={0}, n, k, t;
using namespace std;
ifstream fi ("Battle.inp");
ofstream fo ("Battle.out");

void insert_t(int x)
{while(x<=n){b[x]++;x+=(x&(-x));}
}

int sum_t(int x)
{t=0;while(x>0){t+=b[x]; x&=(x-1);}
 return(t);
}

int main()
{//clock_t aa=clock();
 fi>>n>>k;
 for(int i=1;i<=n;++i)fi>>a[i];
 insert_t(a[1]);
 for(int i=2;i<=n;++)
 {insert_t(a[i]); c[i]=i-sum_t(a[i]);
  for(int i=1;i<=n;++) if(c[i]<=k)c[i]=0;else {c[i]-=k;r[i-k]=a[i];b[a[i]]=-1;}
  t=1;
  for(int i=1;i<=n;++)
   if(r[i]==0){while(b[t]<0)++t;r[i]=t++;}
  for(int i=1;i<=n;++) fo<<r[i]<<" ";
 //clock_t bb=clock();
 //fo<<endl<<(double)(bb-aa)/1000<<" sec.";
 }
```

## 42I. KHÁCH DU LỊCH

Tên chương trình: TOURISTS.???

Một thành phố lớn nổi tiếng với các danh lam thắng cảnh thu hút rất nhiều khách du lịch. Người ta đến thành phố với mục đích chính là tham quan các thắng cảnh và những địa điểm nổi tiếng của thành phố. Khách du lịch là những người rất bận rộn vì vậy không ai muốn thăm quá một lần một loại danh lam thắng cảnh. Ví dụ, để về kề chuyên là tôi đã thăm viện bảo tàng thì chỉ cần vào một viện bảo tàng nào đó là đủ.

Tất cả  $n$  địa điểm nổi tiếng của thành phố đều nằm dọc trên đường phố chính, địa điểm thứ  $i$  thuộc loại  $t_i$  ( $i = 1 \dots n$ ). Các tour du lịch bắt đầu từ một danh lam nào đó và kết thúc ở một danh lam khác. Phụ thuộc vào nơi dừng chân và sau đó là nơi thuận tiện trở về mỗi khách du lịch đăng ký một tour tham quan riêng. Khách thứ  $j$  đăng ký tour  $[l_j, r_j]$  (bắt đầu từ điểm  $l_j$  đến hết điểm  $r_j$ ). Hướng dẫn viên chỉ cần dừng lại giới thiệu kỹ mỗi loại danh lam một lần trên tuyến du lịch. Dĩ nhiên khách muốn được giới thiệu càng nhiều địa điểm càng tốt, nhưng thể loại phải khác nhau từng đôi một.

Hãy xác định số địa điểm mỗi khách du lịch được giới thiệu.

**Dữ liệu:** Vào từ file văn bản TOURISTS.INP:

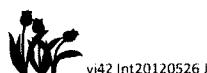
- ◆ Dòng đầu tiên chứa số nguyên  $n$  ( $1 \leq n \leq 10^5$ ),
- ◆ Dòng thứ 2 chứa  $n$  số nguyên  $t_1, t_2, \dots, t_n$  ( $1 \leq t_i \leq 10^9$ ,  $i = 1 \dots n$ ),
- ◆ Dòng thứ 3 chứa số nguyên  $m$  – số khách du lịch ( $1 \leq m \leq 10^5$ ),
- ◆ Dòng thứ  $j$  trong  $m$  dòng sau chứa 2 số nguyên  $l_j$  và  $r_j$  ( $1 \leq l_j \leq r_j \leq n$ ,  $j = 1 \dots m$ ).

**Kết quả:** Đưa ra file văn bản TOURISTS.OUT  $m$  số nguyên, mỗi số trên một dòng. Số nguyên trên dòng thứ  $j$  xác định số địa điểm khách thứ  $j$  được giới thiệu.

**Ví dụ:**

TOURISTS.INP
5
1 1 2 2 1
5
1 5
1 2
2 3
3 4
1 1

TOURISTS.OUT
2
1
2
1
1



*Chuong trinh giải:*

```
#include <fstream>
#include <ctime>
using namespace std;
ifstream fi ("tourists.inp");
ofstream fo ("tourists.out");
int
n,m,k,t[100001],lt,b[100001]={0},res[100001]={0},q[100001]={0},s[1000
01]={0};
pair<int,int> r[100002];

void insert_t(int ii)
{ if(ii==0) s[0]++; else
    while(ii<=n) {s[ii]++;ii+=(ii&(-ii));}
}

int sum(int ii)
{int rr; rr=s[0];
 while(ii>0){rr+=s[ii];ii&=(ii-1); }
 return(rr);
}

int main()
{//clock_t aa=clock();
 fi>>n;
 for(int i=1;i<=n;++i){fi>>r[i].first;r[i].second=i;}
 sort(r+1,r+n+1);
 for(int i=1;i<=n;++i)
    if(r[i].first==r[i-1].first) q[r[i].second]=r[i-1].second;
 fi>>m;
 for(int i =1;i<=m;++i){fi>>b[i]; fi>>r[i].first; r[i].second=i;}
 sort(r+1,r+m+1);
 k=0;
 for(int i=1;i<=m;++i)
 {if (k!=r[i].first)
  { for(int j=k+1;j<=r[i].first;++j) insert_t(q[j]);
    k=r[i].first;
  }
  lt=b[r[i].second]-1; res[r[i].second]=sum(lt)-lt;
 }
 for(int i=1;i<=m;++i) fo<<res[i]<<endl;

 //clock_t bb=clock();
 // fo<<"Time: "<<(double)(bb-aa)/1000<<" seconds.";
}
```

## UF26. HỘP KẸO

Tên chương trình: CANDIES.???

Các bạn gọi điện thoại cho Steve hẹn đến nhà chia vui với kết quả cao mà Steve đã đạt được trong kỳ thi Tin học vừa kết thúc. Steve đi mua  $n$  hộp kẹo để đón bạn, mỗi hộp một loại kẹo và hộp thứ  $i$  có  $a_i$  viên.

Có tất cả  $m$  người tới. Các bạn tới không cùng một lúc mà là lần lượt từng người một. Steve hiểu rất rõ các bạn của mình. Người thứ  $j$  có độ tinh tế  $b_j$ . Điều này có nghĩa là bạn đó sẽ chỉ ăn kẹo ở các hộp có số lượng còn lại không ít hơn  $b_j$  chiếc và sẽ ăn ở những hộp này, mỗi hộp một viên. Nếu một bạn nào đó có độ tinh tế 1 thì bạn đó sẽ ăn ở mỗi hộp một viên kẹo.

Chiều tối, khi các bạn đã về hết, Steve vừa dọn dẹp vừa nhǎm tính xem mỗi bạn đã ăn bao nhiêu viên kẹo.

**Dữ liệu:** Vào từ file văn bản CANDIES.INP:

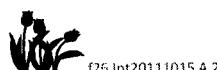
- Dòng đầu tiên chứa số nguyên  $n$  ( $1 \leq n \leq 10^5$ ),
- Dòng thứ 2 chứa  $n$  số nguyên  $a_1, a_2, \dots, a_n$  ( $1 \leq a_i \leq 10^9$ ,  $i = 1 \div n$ ),
- Dòng thứ 3 chứa số nguyên  $m$  ( $1 \leq m \leq 10^5$ ),
- Dòng thứ 4 chứa  $m$  số nguyên  $b_1, b_2, \dots, b_m$  ( $1 \leq b_j \leq 10^9$ ,  $j = 1 \div m$ ).

**Kết quả:** Dưa ra file văn bản CANDIES.OUT  $m$  số nguyên, mỗi số trên một dòng. Số thứ  $j$  là số viên kẹo bạn thứ  $j$  đã ăn.

**Ví dụ:**

CANDIES.INP
3
3 1 1
2
1 2

CANDIES.OUT
3
1



## *Giải thuật*

Sắp xếp  $\{a_i\}$  theo trình tự không tăng: `sort(a+1,a+n+1,greater<int>());`

Cây Fenwick quản lý số lần truy nhập tới các hộp kẹo,

Với mỗi  $b_j, j = 1 \div m$ : tìm kiếm nhị phân trên  $a$  xác định  $k$  max thỏa mãn

$$b_j \leq a_i - t,$$

trong đó  $t$  – số người trước đó đã lấy kẹo từ hộp thứ  $i$ ,

$$t = i - 1 - \text{sum\_t}(i-1) - t_0$$

$t_0$  – số người trước đó không lấy được viên kẹo nào.

```
void b_search(int x, int ii)
{
    int l, r, u;
    l=1; r=n;
    if(x>a[1]-ii+1+c[0]) {k=0; ++c[0]; d[ii]=0; return;}
    while(l<=r)
    {
        u=(l+r)/2;
        if (u>1)t=ii-1-sum_t(u-1)-c[0]; else t=ii-1-c[0];
        if(x<=a[u]-t)l=u+1; else r=u-1;
    }
    k=r;
    d[ii]=k; ++c[k]; if(k>0)insert_t(k);
}
```

Chương trình giải:

```
#include <fstream>
#include <algorithm>
#include <ctime>
using namespace std;
typedef pair<int,int> p2;
int
n,m,k,t,a[100002],c[100001]={0},d[100001],b[100001],s[100001]={0};
ifstream fi ("CANDIES.INP");
ofstream fo ("CANDIES.OUT");

void insert_t(int ii)
{while(ii<=n){s[ii]++;ii+=(ii&(-ii));}
}

int sum_t(int ii)
{int r=0;
 while(ii>0){r+=s[ii];ii&=(ii-1);}
 return(r);
}

void b_search(int x,int ii)
{int l,r,u;
 l=1; r=n;
 if(x>a[1]-ii+1+c[0]) {k=0; ++c[0]; d[ii]=0; return;}
 while(l<=r)
 {u=(l+r)/2;
  if (u>1) t=ii-1-sum_t(u-1)-c[0]; else t=ii-1-c[0];
  if (x<=a[u]-t) l=u+1; else r=u-1;
 }
 k=r;
 d[ii]=k; ++c[k]; if(k>0) insert_t(k);
}

int main()
{clock_t aa=clock();
 fi>>n;
 for(int i=1;i<=n;++i) fi>>a[i];
 fi>>m;
 for(int i=1;i<=m;++i) fi>>b[i];
 sort(a+1,a+n+1,greater<int>()); a[n+1]=0; a[0]=a[1]+1;

for(int j=1;j<=m;++j)
 {b_search(b[j],j);
 }
 for(int j=1;j<=m;++j) fo<<d[j]<<endl;
 clock_t bb=clock();
 fo<<"Time: "<<(double)(bb-aa)/1000<<" sec";
}
```

## VL08. TRÒ CHƠI

Tên chương trình: FLASHMOB.???

Buổi tối trước ngày thi thứ II người ta tổ chức một cuộc chơi thu gom tặng phẩm (flashmob) để các thí sinh giải tỏa tâm lý căng thẳng trước khi vào ngày thi mới.

Quảng trường thành phố được chia thành lưới ô vuông, trên một số ô có để tặng phẩm. Có  $n$  người đăng ký tham gia. Mỗi người nhận được một số xác định trình tự ra sân và tọa độ hai ô ở cùng một hàng hay cột. Khi đến lượt mình, người chơi ra sân thu hết các tặng phẩm ở 2 ô đã nêu và tặng phẩm ở các ô cùng hàng (hoặc cùng cột) giữa 2 ô này. Ban Tổ chức muốn theo số liệu mà mọi người đã nhận được sắp đặt các tặng phẩm sao cho ai ra sân cũng nhặt về được ít nhất một tặng phẩm.

Hãy xác định số lượng tặng phẩm ít nhất cần có và vị trí đặt chúng. Nếu không có cách đặt thì đưa ra số 0. Nếu có nhiều cách đặt thì đưa ra phương án tùy chọn.

**Dữ liệu:** Vào từ file văn bản FLASHMOB.INP:

- ◆ Dòng đầu tiên chứa số nguyên  $n$  ( $1 \leq n \leq 123\,456$ ),
- ◆ Dòng thứ  $i$  chứa 4 số nguyên  $x1_i, y1_i, x2_i, y2_i$  – tọa độ 2 ô của người sẽ ra ở lượt thứ  $i$  ( $1 \leq x1_i, y1_i, x2_i, y2_i \leq 10^9$ ,  $x1_i = x2_i$  hoặc  $y1_i = y2_i$ ).

**Kết quả:** Đưa ra file văn bản FLASHMOB.OUT:

- ◆ Dòng đầu chứa số nguyên  $m$  – số tặng phẩm ít nhất cần có,
- ◆ Nếu  $m > 0$  thì mỗi dòng trong  $m$  dòng tiếp theo chứa 2 số nguyên xác định tọa độ ô cần đặt tặng phẩm.

**Ví dụ:**

FLASHMOB.INP	FLASHMOB.OUT
5	5
2 1 2 4	1 2
2 4 4 4	4 3
5 1 1 1	1 1
4 4 4 2	3 4
4 2 1 2	2 3



Chương trình giải:

```
#include <cassert>
#include <cstdio>
#include <algorithm>
#include <fstream>
#include <vector>
#include <ctime>
using namespace std;
#define pb push_back
#define mp make_pair
#define fs first
#define sc second
#define sz(a) ((int) (a).size())
#define taskname "flashmob"

const int N = 4 * 123456 + 10;
const int inf = (int) 1e9 + 1;
int ansx[N], ansy[N], x[N][2], y[N][2], x_size[N], pos[N], n;
vector< pair< int, pair< int, int > > > x_events[N], y_events[N];
vector< int > allx, ally;
ifstream fi (taskname".inp");
ofstream fo (taskname".out");

struct t_rmq {
    pair<int, int>* rmq;
    int n;
    t_rmq(int max_size = 0) {
        n = max_size;
        while (n & (n - 1))
            ++n;
        if (n > 0) {
            rmq = new pair<int, int>[2 * n - 1];
            init(0, 0, n - 1);
        }
    }
    void init(int t, int l, int r) {
        if (l == r)
            rmq[t] = mp(-inf, 1);
        else {
            int m = (l + r) / 2;
            init(t * 2 + 1, l, m);
            init(t * 2 + 2, m + 1, r);
            rmq[t] = max(rmq[t * 2 + 1], rmq[t * 2 + 2]);
        }
    }
    pair<int, int> get(int l, int r) {
        pair<int, int> res = mp(-inf, -1);
        l += n - 1, r += n - 1;
        while (l < r) {
            if (l % 2 == 0)
                res = max(res, rmq[l++]);
            if (r % 2 == 1)
                res = max(res, rmq[r--]);
            l = (l - 1) / 2, r = (r - 1) / 2;
        }
        return (l <= r) ? max(res, rmq[l]) : res;
    }
}
```

```

void update(int t, int value) {
    t += n - 1;
    rmq[t] = mp(value, t - (n - 1));
    while (t > 0) {
        t = (t - 1) / 2;
        rmq[t] = max(rmq[t * 2 + 1], rmq[t * 2 + 2]);
    }
}
};

struct t_set {
    t_rmq rmq;
    int n;
    t_set(int max_size = 0): rmq(max_size), n(0) {
    }
    ~t_set() {}
    void insert(int x) {
        pos[x] = n++;
        rmq.update(pos[x], -x);
    }
    void erase(int x) {
        rmq.update(pos[x], -inf);
    }
    int get_min() {
        return -rmq.rmq[0].fs;
    }
};
};

t_rmq projection;
t_set current, state[N];

void try_to_find_answer(int i, int l, int r) {
    l = max(l, x[i][0]), r = min(r, x[i][1]);
    if ((ansx[i] == 0) && (ansy[i] == 0) && (l <= r)) {
        pair<int, int> tmp = projection.get(l, r);
        if (tmp.fs >= i) {
            ansx[i] = allx[tmp.sc];
            ansy[i] = ally[y[i][0]];
        }
    }
}

void solve() {
    current = t_set(n);
    projection = t_rmq(sz(allx));
    for (int i = 0; i < sz(allx); ++i) {
        x_events[i].clear();
        projection.update(i, inf);
        x_size[i] = 0;
    }
    for (int i = 0; i < sz(ally); ++i)
        y_events[i].clear();
    for (int i = 0; i < n; pos[i] = -1, ++i)
        if (x[i][0] == x[i][1]) {
            x_events[y[i][0]].pb(mp(x[i][0], mp(i, -1)));
            x_events[y[i][1] + 1].pb(mp(x[i][0], mp(i, 1)));
            x_size[x[i][0]]++;
        } else {

```

```

        y_events[y[i][0]].pb(mp(x[i][0], mp(i, -1)));
        y_events[y[i][0]].pb(mp(x[i][1] + 1, mp(i, 1)));
    }
    for (int i = 0; i < sz(allx); ++i)
        state[i] = t_set(x_size[i]);
    for (int i = 0; i < sz(alloy); ++i) {
        for (int j = 0; j < sz(x_events[i]); ++j) {
            if (x_events[i][j].sc.sc == -1)

state[x_events[i][j].fs].insert(x_events[i][j].sc.fs);
            else

state[x_events[i][j].fs].erase(x_events[i][j].sc.fs);
            projection.update(x_events[i][j].fs,
state[x_events[i][j].fs].get_min());
        }
        sort(y_events[i].begin(), y_events[i].end());
        for (int j = 0; j < sz(y_events[i]); ++j) {
            if (y_events[i][j].sc.sc == -1)
                current.insert(y_events[i][j].sc.fs);
            else
                current.erase(y_events[i][j].sc.fs);
            if (current.get_min() != inf)
                try_to_find_answer(current.get_min(),
y_events[i][j].fs, y_events[i][j + 1].fs - 1);
        }
    }
}

void compress(int (&x)[N][2], int n, vector<int> &allx) {
    for (int i = 0; i < n; ++i)
        for (int t = 0; t < 2; ++t)
            for (int delta = 0; delta <= 1; ++delta)
                allx.pb(x[i][t] + delta);

    allx.pb(inf);
    sort(allx.begin(), allx.end());
    allx.resize(unique(allx.begin(), allx.end()) - allx.begin());
    for (int i = 0; i < n; ++i)
        for (int t = 0; t < 2; ++t)
            x[i][t] = lower_bound(allx.begin(), allx.end(),
x[i][t]) - allx.begin();
}

int main()
{
    fi>>n;
    for (int i = 0; i < n; ++i) {
        for (int t = 0; t < 2; ++t)
            fi>>x[i][t]>>y[i][t];
        sort(x[i], x[i] + 2);
        sort(y[i], y[i] + 2);
    }
    compress(x, n, allx);
    compress(y, n, alloy);
    for (int t = 0; t < 2; ++t) {
        solve();
        swap(x, y);
        swap(ansx, ansy);
        swap(allx, ally);
    }
}

```

```
    }
    for (int i = 0; i < n; ++i)
        if (ansx[i] == 0) {
            printf("0\n");
            return 0;
        }
    fo<<n<<'\n';
    for (int i = 0; i < n; ++i)
        fo<<ansx[i]<<ansy[i]<<'\n';
    return 0;
}
```

## 2 – Quản lý cực trị

Các bài toán tin học phần lớn đều liên quan đến việc tìm kiếm giá trị lớn nhất hoặc nhỏ nhất, tức là tìm các cực trị (min, max) địa phương hoặc toàn cục.

Xác định cực trị trên tập kích thước cố định và các phần tử của tập không thay đổi giá trị trong quá trình tính toán là một bài toán đơn giản, có độ phức tạp O(n).

Vấn đề sẽ phức tạp hơn nhiều khi phải thực hiện nhiều phép tìm kiếm trên các tập con khác nhau của tập cho trước và một số phần tử của tập có thể thay đổi giá trị sau mỗi bước xử lý.

### 2.1 – Tìm cực trị toàn cục

Cực trị toàn cục có thể được xác định thông qua các cấu trúc dữ liệu Vun đống (*Heap*) hoặc Hàng đợi ưu tiên (*Priority\_queue*). Với độ phức tạp O(nlogn) các cấu trúc này đảm bảo phần tử cực trị (min hoặc max) luôn đứng ở đầu cấu trúc.

```
// range heap example
#include <iostream>           // std::cout
#include <algorithm>          // std::make_heap, std::pop_heap, std::push_heap,
std::sort_heap
#include <vector>              // std::vector

int main () {
    int myints[] = {10,20,30,5,15};
    std::vector<int> v(myints,myints+5);

    std::make_heap (v.begin(),v.end());
    std::cout << "initial max heap : " << v.front() << '\n';

    std::pop_heap (v.begin(),v.end()); v.pop_back();
    std::cout << "max heap after pop : " << v.front() << '\n';

    v.push_back(99); std::push_heap (v.begin(),v.end());
    std::cout << "max heap after push: " << v.front() << '\n';

    std::sort_heap (v.begin(),v.end());

    std::cout << "final sorted range :";
    for (unsigned i=0; i<v.size(); i++)
        std::cout << ' ' << v[i];

    std::cout << '\n';

    return 0;
}

Output:
initial max heap : 30
max heap after pop : 20
max heap after push: 99
final sorted range : 5 10 15 20 99
```

Cấu trúc dữ liệu priority\_queue cho phép tạo hàng đợi trong đó các phần tử của hàng đợi đứng theo thứ tự tăng dần hoặc giảm dần, điều này có nghĩa là phần tử cực trị luôn đứng ở đầu hàng đợi. Theo ngầm định hàng đợi ưu tiên được tổ chức theo trình tự giảm dần của giá trị các phần tử.

```

#include <iostream>
#include <queue>
#include <iomanip>

using namespace std;

struct Time {
    int h; // >= 0
    int m; // 0-59
    int s; // 0-59
};

class CompareTime {
public:
    bool operator()(Time& t1, Time& t2)
    {
        if (t1.h < t2.h) return true;
        if (t1.h == t2.h && t1.m < t2.m) return true;
        if (t1.h == t2.h && t1.m == t2.m && t1.s < t2.s) return true;
        return false;
    }
};

int main()
{
    priority_queue<Time, vector<Time>, CompareTime> pq;

    // Array of 4 time objects:

    Time t[4] = { {3, 2, 40}, {3, 2, 26}, {5, 16, 13}, {5, 14, 20} };

    for (int i = 0; i < 4; ++i)
        pq.push(t[i]);

    while (!pq.empty()) {
        Time t2 = pq.top();
        cout << setw(3) << t2.h << " " << setw(3) << t2.m << " " <<
        setw(3) << t2.s << endl;
        pq.pop();
    }

    return 0;
}

```

Chương trình này in thời gian theo trình tự từ muộn nhất đến sớm nhất:

```

5   16   13
5   14   20
3   2   40
3   2   26

```

Nếu muốn in thời gian theo trình tự từ sớm nhất đến muộn nhất thì cần tổ chức hàng đợi ưu tiên như sau:

```

class CompareTime {
public:
    bool operator()(Time& t1, Time& t2) // t2 has highest prio than t1 if t2
is earlier than t1
    {
        if (t2.h < t1.h) return true;
        if (t2.h == t1.h && t2.m < t1.m) return true;
        if (t2.h == t1.h && t2.m == t1.m && t2.s < t1.s) return true;
    }
};

```

```

        return false;
    }
};

// constructing priority queues
#include <iostream>           // std::cout
#include <queue>              // std::priority_queue
#include <vector>              // std::vector
#include <functional>          // std::greater

class mycomparison
{
    bool reverse;
public:
    mycomparison(const bool& revparam=false)
    {reverse=revparam;}
    bool operator() (const int& lhs, const int&rhs) const
    {
        if (reverse) return (lhs>rhs);
        else return (lhs<rhs);
    }
};

int main ()
{
    int myints[] = {10,60,50,20};

    std::priority_queue<int> first;
    std::priority_queue<int> second (myints,myints+4);
    std::priority_queue<int, std::vector<int>, std::greater<int> >
        third (myints,myints+4);

    // using mycomparison:
    typedef std::priority_queue<int, std::vector<int>, mycomparison> mypq_type;

    mypq_type fourth;           // less-than comparison
    mypq_type fifth (mycomparison(true)); // greater-than comparison

    return 0;
}

```

Ở C++ hàng đợi ưu tiên được xây dựng dựa trên cơ sở vun đống và vì vậy, trong nhiều trường hợp, thuận tiện hơn cho việc xử lý dữ liệu.

Khi sử dụng hàng đợi cần lưu ý hai vấn đề:

- ◆ Phần lớn các bài toán đều đòi hỏi biết giá trị cực đại cùng với thông tin về vị trí của giá trị đó trong véc tơ hay ma trận chứa nó, vì vậy phần tử của hàng đợi thường lưu trữ dưới dạng cặp dữ liệu (pair),
- ◆ Bản chất của hàng đợi là loại cấu trúc dữ liệu truy nhập tuyến tính một chiều (trừ hàng đợi 2 đầu dequeue), vì vậy cần lưu ý lọc thông tin “nhiều” trong quá trình xử lý, tức là loại bỏ các thông tin không còn phù hợp khi cấu trúc dữ liệu bị thay đổi trong quá trình xử lý. Vấn đề này nói chung không mất quá nhiều thời gian và không ảnh hưởng đến độ phức tạp của giải thuật.

## Bài tập ứng dụng:

### 49K. CHIA THUỐC

Tên chương trình: LIQUORS.???

Harry và Nevill có một chương trình nâng cao thể lực và kỹ thuật đối kháng. Chương trình gồm  $n$  bài thực hành. Cứ sau mỗi bài học mỗi người lại uống một chén thuốc phục sức lực. Có tất cả  $2n$  chén thuốc, chén thứ  $i$  có khả năng khôi phục  $a_i$ ,  $i = 1 \div 2n$ . Hai người chọn 2 chén có độ chênh lệch về khả năng khôi phục ít nhất, nếu tồn tại nhiều cặp như vậy thì chọn cặp có tổng khả năng khôi phục lớn nhất. Nevill uống chén có khả năng khôi phục lớn hơn, còn Harry uống chén kia.

Hãy chỉ ra khả năng khôi phục ở từng chén mà mỗi người đã uống theo đúng trình tự uống.

**Dữ liệu:** Vào từ file văn bản LIQUORS.INP:

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

**Kết quả:** Đưa ra file văn bản LIQUORS.OUT 2 dòng, mỗi dòng  $n$  số nguyên, dòng thứ nhất là năng lực khôi phục của các chén mà Nevill uống, dòng thứ 2 – năng lực khôi phục của các chén mà Harry uống. Các số được đưa ra theo trình tự uống.

**Ví dụ:**

LIQUORS.INP
2
1 2 4 8

LIQUORS.OUT
2 8
1 4



```
#include <fstream>
#include <ctime>
#include <queue>
#include <algorithm>
using namespace std;
typedef pair<int,int> p2;
typedef pair<int,p2> p3;
int a[200001],p[200001],q[200001],r1[200001],r2[200001],r[200001]={0},
    n,n2,nr=0,it=0,i1,i2;
p3 b[200001],x;
priority_queue<p3,vector<p3>,greater<p3> > d;
ifstream fi ("Liquors.inp");
ofstream fo ("Liquors.out");

void get_new_el()
{int fd;
 fd=0;while(!fd)
 {
    x=d.top();i1=x.second.first;i2=x.second.second;d.pop();
    if(r[i1]==0 && r[i2]==0){++nr;fd=1;return;}
 }
}

void upd_data()
{int ii1,ii2;
 r[i1]=1;r[i2]=2; ii1=p[i1];ii2=q[i2];r1[nr]=a[i1];r2[nr]=a[i2];
 if(ii1!=0 && ii2!=n2+1)
```

```

{q[ii1]=ii2; p[ii2]=ii1;
 x.first=a[ii1]-a[ii2];x.second.first=ii1;x.second.second=ii2;
 d.push(x);
}

int main()
{clock_t aa=clock();

fi>>n; n2=n<<1;
for(int i=1;i<=n2;++i)fi>>a[i];
sort(a+1,a+n2+1,greater<int>());
for(int i=1;i<n2;++i)
 {x.first=a[i]-a[i+1];x.second.first=i;x.second.second=i+1;d.push(x);}

for(int i=1;i<=n2;++i){p[i]=i-1;q[i]=i+1;}
while(nr<n)
 {get_new_el();
  upd_data();
 }
for(int i=1;i<=n;++i)fo<<r1[i]<<" ";
fo<<endl;
for(int i=1;i<=n;++i)fo<<r2[i]<<" ";
fo<<'\
';
clock_t bb=clock();
fo<<"Time: "<<(double)(bb-aa)/1000<<" sec";
}

```

## 46J. TRÌNH TỰ

Tên chương trình: ORDER.???

Steve xây dựng được một chương trình nhận dạng nhanh dãy số có phải là không giảm hay không theo giải thuật riêng của mình. Để chuẩn bị trình bày kết quả trong hội nghị khoa học Steve tạo một chương trình tương tác với chương trình nhận dạng của mình. Chương trình tương tác làm việc với dãy  $n$  số nguyên  $a_1, a_2, \dots, a_n$  và xử lý  $m$  truy vấn, mỗi truy vấn thuộc một trong 2 loại với quy cách:

- ◆ Loại I:  $! \ k \ x$  – Gán cho phần tử  $a_k$  giá trị nguyên  $x$ ,
- ◆ Loại II:  $? - Gọi chương trình nhận dạng, kiểm tra xem dãy số có phải là không giảm hay không và đưa ra câu trả lời tương ứng là YES hoặc NO.$

Hãy xác định các câu trả lời nhận được trong quá trình tương tác.

**Dữ liệu:** Vào từ file văn bản ORDER.INP:

- ◆ Dòng đầu tiên chứa 2 số nguyên  $n$  và  $m$  ( $1 \leq n \leq 10^5$ ,  $0 \leq m \leq 2 \times 10^5$ ),
- ◆ Dòng thứ 2 chứa  $n$  số nguyên  $a_1, a_2, \dots, a_n$  ( $0 \leq a_i \leq 10^9$ ,  $i = 1 \div n$ ),
- ◆ Mỗi dòng trong  $m$  dòng sau chứa một truy vấn.

**Kết quả:** Đưa ra file văn bản ORDER.OUT các câu trả lời đối với truy vấn loại II. Mỗi câu trả lời đưa ra trên một dòng.

**Ví dụ:**

ORDER.INP	ORDER.OUT
5 5	
2 4 6 8 10	
?	
! 2 7	YES
?	NO
! 3 8	YES
?	



VJ46ло20130126 B

```

#include <fstream>
#include <iostream>
using namespace std;
int a[100002],n,m,k,x,v;
string s;
//ifstream fi ("Order.inp");
//ofstream fo ("Order.out");

int main()
{freopen("order.inp","r",stdin);
 freopen("order.out","w",stdout);
 cin>>n>>m;
 for(int i=1;i<=n;++i)cin>>a[i];
 a[0]=-1000000001; a[n+1]=1000000001; v=0;
 for (int i=1;i<=n;++i) if(a[i]<a[i-1])++v;
 for (int i=1;i<=m;++i)
 {cin>>s;
 if(s=="?") {if(v==0) cout<<"YES"<<endl; else cout<<"NO"<<endl;}
 else
 {cin>>k>>x;
 v-= (a[k - 1] > a[k]) + (a[k] > a[k + 1]);
 a[k]=x;
 v+= (a[k - 1] > a[k]) + (a[k] > a[k + 1]);
 }
 }
}
}

```

## 38L. HIỆN TƯỢNG ẨM LÊN TOÀN CẦU

Tên chương trình: WARMING.???

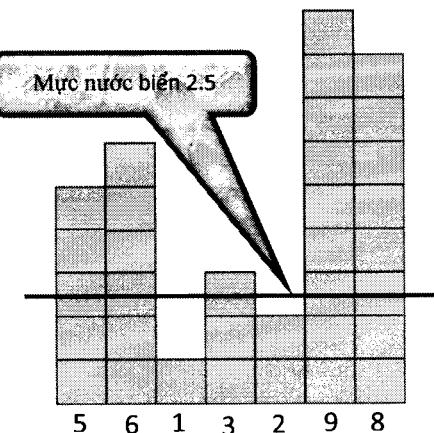
Hiện tượng ẩm lên toàn cầu làm băng ở các cực tan, nước biển sẽ dâng lên, một số vùng đất bị ngập và nơi vốn là đất liền có thể xuất hiện các hòn đảo mới. Các nhà khoa học khảo sát sự biến đổi của một vùng đất qua mô hình một chiều. Địa hình của vùng đất được thể hiện bằng dãy độ cao có giá trị không âm  $h_0, h_1, \dots, h_{n-1}$ . Ở hình bên, với địa hình có các độ cao là 5, 6, 1, 3, 2, 9, 8 thì khi nước biển là 2.5m sẽ hình thành 3 hòn đảo.

Hãy xác định số lượng đảo tối đa có thể hình thành khi nước biển dâng.

**Dữ liệu:** Vào từ file văn bản WARMING.INP:

- ◆ Dòng đầu tiên chứa số nguyên  $n$  ( $1 \leq n \leq 10^6$ ),
- ◆ Dòng thứ  $i$  trong  $n$  dòng sau chứa số nguyên  $h_{i-1}$  ( $0 \leq h_i \leq 2^{30}$ ,  $i = 0 \div n-1$ ).

**Kết quả:** Đưa ra file văn bản WARMING.OUT một số nguyên – số đảo tối đa có thể hình thành.



**Ví dụ:**

WARMING.INP
7
5
6
1
3
2
9
8

WARMING.OUT
3



vL38 NOI.13.1

## *Giải thuật*

B1. Tạo vector  $A = (a_0, a_1, \dots, a_{n-1})$  lưu các cực trị địa phương. Chú ý cách xử lý  $a_0$  và  $a_{n-1}$ . Trong sơ đồ xử lý chung, khi kết thúc, nếu na lẻ thì  $a[+n] = giá trị nhập vào cuối cùng$ . Không cần lưu các giá trị  $h_i$ ,  $i = 0 \div n-1$ .

B2. Nạp vào hàng đợi ưu tiên tăng dần v các cực tiểu địa phương.

**priority\_queue <p2, vector<p2>, greater<p2> > v;** Phần tử của hàng đợi: cặp số nguyên ( $a_j, j$ ).

B3. Tô chúc **priority\_queue <int, vector<int>, greater<int> > d** chứa các cực trị địa phương biên của các đảo.

Duyệt các phần tử của hàng đợi v. Với mỗi ( $a_j, j$ ):

- Loại bỏ khỏi d các giá trị nhỏ hơn hoặc bằng  $a_j$ . Mỗi lần loại bỏ - giảm số đảo 1,
- Nạp các  $a_{j-1}$  và  $a_{j+1}$  chưa được đánh dấu vào d,
- Đánh dấu nạp phần tử  $a_k$ :  $a[k] = -1$ ,
- Tăng số đảo lên 1.

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

*Phương án hiệu quả hơn: WARMING*

```
#include <fstream>
#include <vector>
#include <queue>
#include <ctime>
using namespace std;
int r,res,t,t1,tx,tmn,n,na,k,h,y,km,m;
vector <int>a;
typedef pair<int,int> p2;
vector <p2> b;
ifstream fi ("warming.inp");
ofstream fo ("warming.out");

int main()
{clock_t aa=clock();
 fi>>n; m=0;t1=-1;
 a.push_back(-1); k=1;
 for(int i=1;i<=n;++i)
 {fi>>t;
 switch(k)
 {case 1: if(t>=t1)t1=t; else {a.push_back(t1);t1=t;++m; k^=1;break;}
 case 0: if(t<=t1)t1=t; else {a.push_back(t1);t1=t;++m; k^=1;break;}
 }
 }
 if((m&1)==0){a.push_back(t1);++m;}
 for(int i=1;i<=m;++i)b.push_back(make_pair(a[i],i));
 sort(b.begin(),b.end());
 res=1;r=1; b.push_back(make_pair(b[m].first+1,m+1));
 for(int i=0;i<m;++i)
 {if((b[i].second&1)==0)++r;else --r;
 if(b[i].first!=b[i+1].first && r>res) res=r;
 }
 fo<<res;
 clock_t bb=clock();
 fo<<"\nTime: "<<(double)(bb-aa)/1000<<" sec";
}
```

## 2.2 – Tìm cực trị trên các tập con của tập ổn định

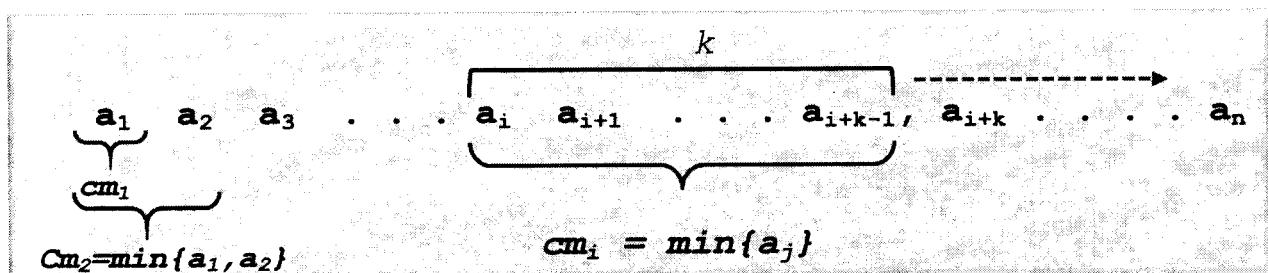
Một tập được gọi là ổn định nếu trong quá trình xử lý kích thước của tập ban đầu và giá trị các phần tử thuộc tập không thay đổi. Trong phần lớn các bài toán thực tế tập ban đầu là một vec tơ hoặc ma trận.

Dữ liệu điều khiển quá trình tìm kiếm cực trị được tổ chức dưới dạng một vec tơ hoạt động theo nguyên tắc hàng đợi – chỉ nạp dữ liệu vào từ một đầu, nhưng dữ liệu lưu trữ trong hàng đợi tạo thành một heap đã sắp xếp. Khi tìm cực tiểu, dữ liệu trong hàng đợi phải có giá trị tăng dần (hoặc không giảm) tính từ đầu hàng đợi, ngược lại, khi tìm cực đại, dữ liệu trong hàng đợi phải có giá trị giảm dần (hoặc không tăng) tính từ đầu hàng đợi. Như vậy, có thể khai báo vec tơ và điều khiển nó như một hàng đợi hoặc trực tiếp sử dụng cấu trúc hàng đợi do STL của hệ thống lập trình cung cấp. Việc sử dụng trực tiếp các công cụ tổ chức hàng đợi do hệ thống lập trình cung cấp sẽ làm cho chương trình sáng sủa, dễ hiểu hơn nhưng hiệu quả giải thuật sẽ giảm sút đôi chút do hệ thống phải thực hiện một số thao tác phục vụ cho hàng đợi. Dùng vec tơ mô phỏng sẽ tránh được các hoạt động dịch vụ không cần thiết này và đảm bảo được hiệu quả xử lý cao nhất.

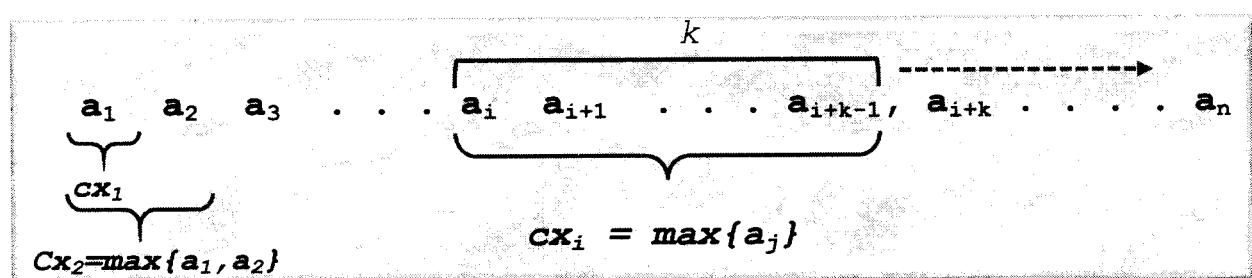
### Mô hình toán học

Cho các số nguyên dương  $n, k$  và dãy số nguyên  $a_1, a_2, \dots, a_n$ , ( $0 < a_i \leq 2^{30}$ ,  $i = 1 \div n$ ,  $1 \leq k \leq n \leq 10^6$ ).

Hãy xác định  $CM = (cm_1, cm_2, \dots, cm_n)$  và  $CX = (cx_1, cx_2, \dots, cx_n)$ , trong đó:  
 $cm_i = \min\{a_j | j = \max\{1, i-k\} \div \min\{i+k-1, n\}\}$



$$cx_i = \max\{a_j | j = \max\{1, i-k\} \div \min\{i+k-1, n\}\}$$



### Giải thuật

a) Giải thuật độ phức tạp  $O(n \times k)$

■ Với mỗi  $i$  ( $i = 1 \div n$ ):

■ Xác định điểm cuối  $ie$  của khoảng cần tìm cực trị:  $ie = \min\{i+k-1, n\}$

■ Tính  $cm_i, cx_i$ : duyệt  $a_j, j = i \div ie$ , tìm min và max.

### b) Giải thuật độ phức tạp O(n)

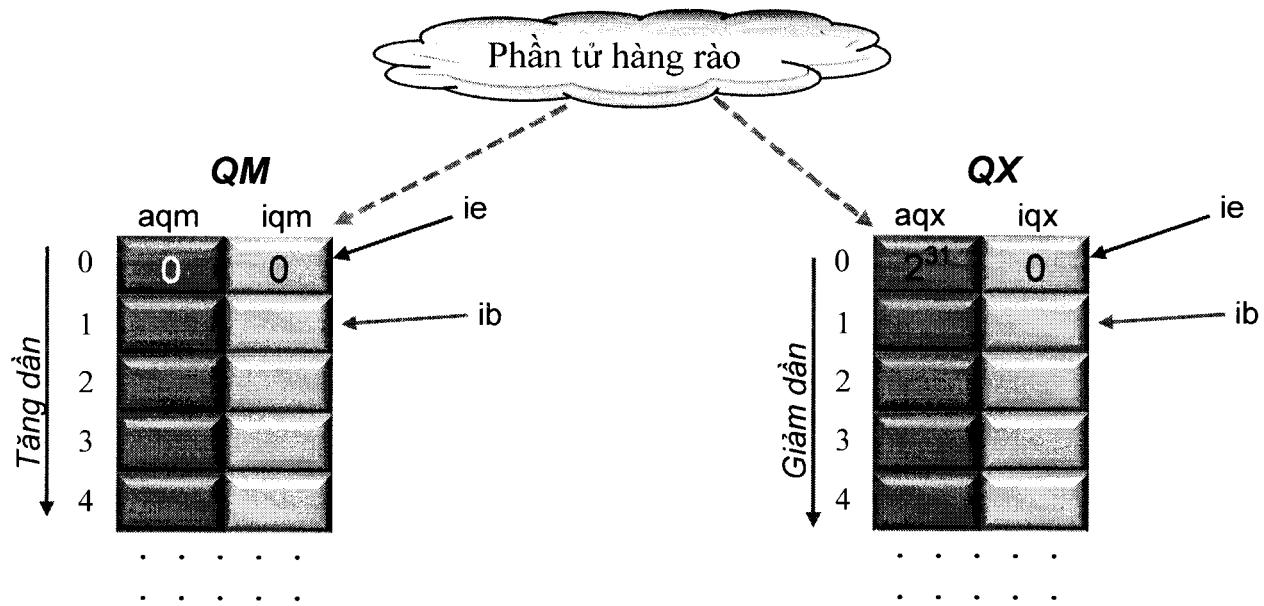
Tổ chức hàng đợi qm mô phỏng heap min và hàng đợi qx mô phỏng heap max lưu trữ các giá trị  $a_i$  trong quá trình duyệt với  $i = 1 \div n$ ,

Sơ đồ tổ chức dữ liệu:

- Mảng A = (a1, a2, ..., an) – lưu trữ dữ liệu vào,
- Hàng đợi QM = {(aqm, iqmq)} và hàng đợi QX = { (aqx, iqx)}, aqm và aqx – lưu giá trị  $a_i$  nạp vào hàng đợi, iqmq và iqx lưu trữ chỉ số i, lưu ý: aqm và aqx không thực sự cần thiết vì có thể tính được từ iqmq và iqx ( $aqm = a_{iqm}$ ),
- Các chỉ số ib và ie xác định vị trí đầu và cuối các phần tử trong hàng đợi tham gia xác định min, max (nếu dùng cấu trúc dữ liệu **deque** trong C++ thì không cần các biến này),
- Các mảng CM và CX để lưu trữ kết quả.

Khởi tạo giá trị đầu:

- Tạo hàng rào: Chỉ cần thiết khi không dùng trực tiếp cấu trúc dữ liệu **deque** trong C++,



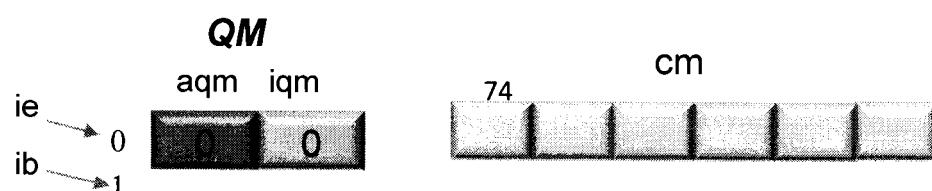
Xử lý: Xét trường hợp tìm min (trường hợp tìm max – xử lý tương tự): với  $i = 1 \div n$ :

- Loại bỏ các phần tử  $aqm_{ie}$  lớn hơn hoặc bằng  $a_i$ : while( $aqm[ie] \geq a[i]$ ) --ie;
- Nạp  $a_i$  vào hàng đợi:  $aqm[++ie] = a[i]$ ;
- Chỉnh lý đầu hàng đợi ib:  

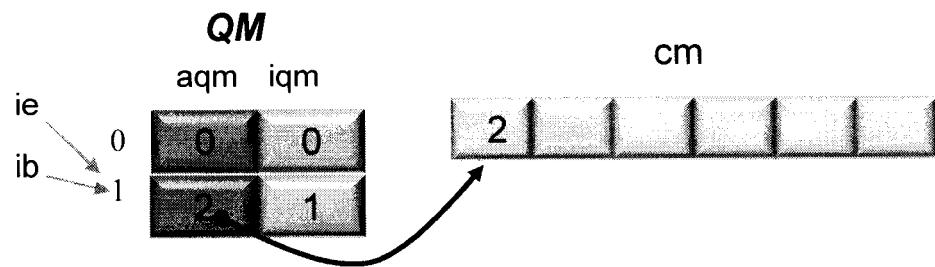
$$\begin{aligned} &\text{if}(ie < ib) ib = ie; \quad // Đảm bảo đuôi ở sau đầu} \\ &\text{if}(iqm[ie] - iqmq[ib] \geq k) ++ib; \quad // Đảm bảo khoảng cách không quá } \\ &k \end{aligned}$$
- Ghi nhận min:  $c[i] = aqm[ib];$

Ví dụ: Với  $n = 6$ ,  $k = 3$  và  $A = (2, 1, 3, 4, 6, 5)$ .

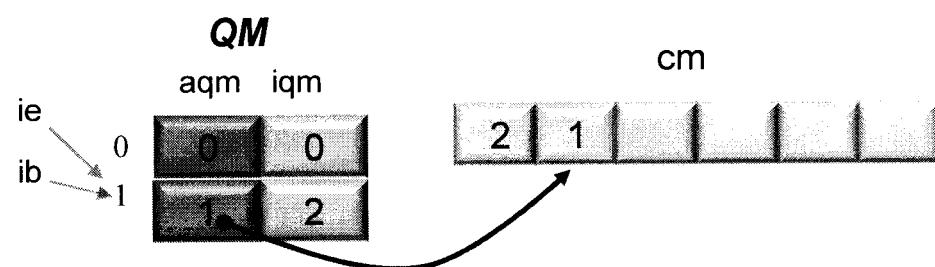
Ban đầu:



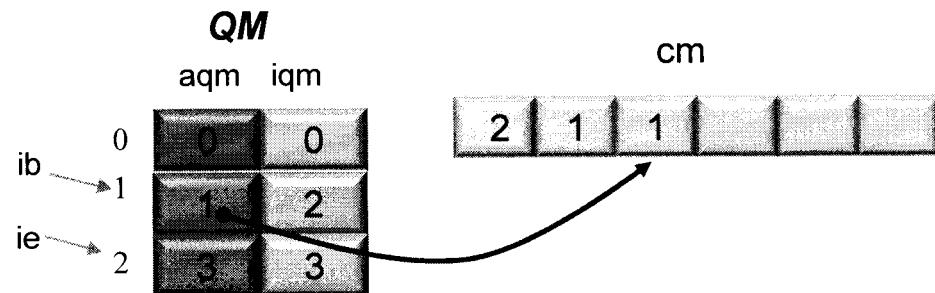
$i = 1:$



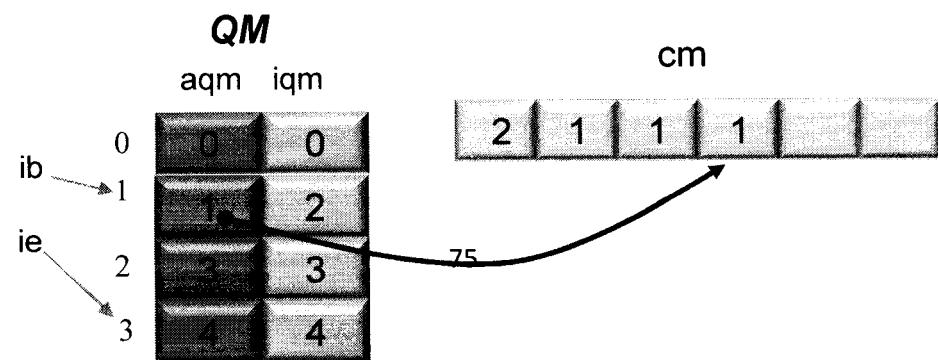
$i = 2:$



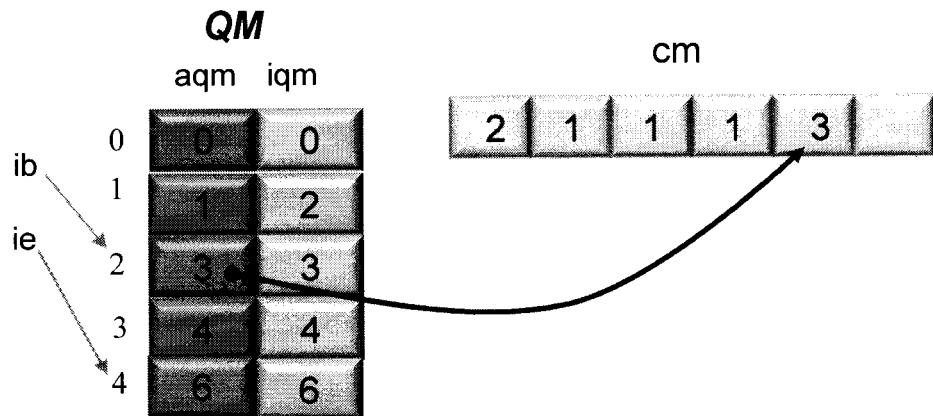
$i = 3:$



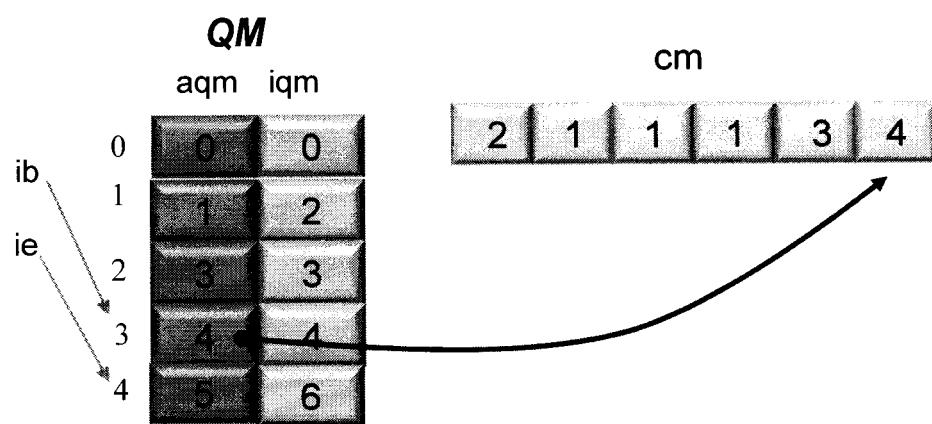
$i = 4:$



i = 5:



i = 6:



Việc tính max (cx) cũng được thực hiện tương tự nhưng luôn luôn đảm bảo tính giảm dần của các giá trị trong hàng đợi aqx.

Phương án dùng hàng đợi **deque** của C++:

Dữ liệu: Vào từ file văn bản MIN\_MAX.INP:

- Dòng đầu tiên chứa 2 số nguyên n và k ( $1 \leq k \leq n \leq 10^6$ ),
- Dòng thứ 2 chứa n số nguyên  $a_1, a_2, \dots, a_n$  ( $0 < a_i \leq 2^{30}$ ,  $i = 1 \div n$ ).

Kết quả: Đưa ra file văn bản MIN\_MAX.OUT:

- Dòng thứ nhất chứa n số nguyên  $cm_1, cm_2, \dots, cm_n$ ,
- Dòng thứ hai chứa n số nguyên  $cx_1, cx_2, \dots, cx_n$ .

Chương trình giải:

```
#include <iostream>
#include <deque>
#include <ctime>
using namespace std;
```



## VU35. ĐỦ CHẤT

Tên chương trình: FOOT.???

Cũng như mọi sinh viên, Steve cố gắng đảm bảo ăn uống điều độ, đủ chất và tiết kiệm. Đã mấy năm rồi, sáng nào Steve cũng ăn hai cái bánh mỳ tròn và uống một cốc sữa đậu nành.

Sữa đậu nành đóng hộp có thể giữ khá lâu, nhưng bánh mỳ thì không để dành được quá  $k$  ngày (kể từ ngày mua). Giá bánh mỳ thường xuyên biến động. Nhờ tính tình vui vẻ cởi mở, Steve có quan hệ rất tốt với người bán hàng và biết được giá bánh trong  $m$  ngày tính từ hôm nay. Từ đó Steve có thể lên kế hoạch để tiết kiệm nhất trong việc mua bánh mỳ.

Ví dụ, bánh có thể giữ được trong hai ngày. Giá bánh hôm nay là 3 đồng/chiếc, giá ngày mai là 1 đồng/chiếc và giá ngày kia sẽ là 2 đồng/chiếc. Kế hoạch chi tiết kiệm của Steve sẽ là: hôm nay mua hai chiếc bánh mỳ tròn, ngày mai – sẽ mua 4 chiếc vừa ăn vừa để dành cho ngày kia. Như vậy Steve phải chi tất cả là  $3 \times 2 + 2 \times 4 = 10$ . Khi có nhiều phương án, Steve luôn lựa chọn cách mua sao cho bánh mỳ phải để dành càng ít ngày càng tốt.

**Yêu cầu:** Cho  $m$ ,  $k$  và  $c_i$ ,  $i = 1 \dots m$ , trong đó  $c_i$  – giá một chiếc bánh mỳ tròn bán ngày thứ  $i$  ( $1 \leq m, k, c_i \leq 10^5$ ). Hãy xác định số tiền tối thiểu cần có và số lượng bánh phải mua ở mỗi ngày.

**Dữ liệu:** Vào từ file văn bản FOOT.INP:

- Dòng thứ nhất chứa 2 số nguyên  $m$  và  $k$ ,
- Dòng thứ 2 chứa  $m$  số nguyên  $c_1, c_2, \dots, c_m$ .

**Kết quả:** Đưa ra file văn bản FOOT.OUT:

- Dòng thứ nhất chứa một số nguyên – chi phí tối thiểu,
- Dòng thứ 2 chứa  $m$  số nguyên, số thứ  $i$  xác định lượng bánh cần mua trong ngày  $i$ .

**Ví dụ:**

FOOT.INP
3 2
3 1 2

FOOT.OUT
10
2 4 0

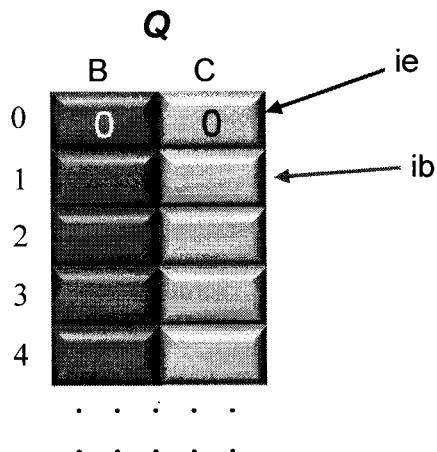


U35\_Inf\_A\_091003A\_E

## *Giải thuật*

*Cấu trúc dữ liệu:*

- ◆ Mảng A = ( $a_1, a_2, \dots, a_m$ ) – lưu dữ liệu vào,
- ◆ Mảng D = ( $d_1, d_2, \dots, d_m$ ),  $d_i$  xác định ngày bánh mỳ được mua cho ngày ăn thứ i,
- ◆ Hàng đợi Q = (B, C) – mô phỏng hoạt động heap min:
  - $b_j$  – giá trị chi phí  $a_i$  được nạp vào Q,
  - $c_j$  – chỉ số i (ngày có giá là  $b_j$ ),
  - ib, ie – các chỉ số quản lý đầu và cuối hàng đợi Q.



*Khởi tạo giá trị đầu:*

*Xử lý:*

- ◆ Luôn luôn đảm bảo  $b_j$  tăng dần,  $j = 0 \div ie$ ,
- ◆ Xác định ngày mua bánh mỳ cho ngày i ( $i = 1 \div m$ ):
  - Loại bỏ các phần tử  $b_j \geq a_i$  và nạp  $a_i$  vào cuối hàng đợi,
  - Chính lý ib và ie, đảm bảo  $ib \leq ie$  và  $ie - ib < k$ ,
  - Ghi nhận ngày mua:  $d_i = c_{ib}$ .

*Dẫn xuất kết quả:*

Dùng mảng C tích lũy số bánh cần mua ở mỗi ngày, tính tổng chi phí và đưa ra các thông tin theo yêu cầu.

```

#include <fstream>
using namespace std;
const int nmax=100001;
ifstream fi ("FOOT.INP");
ofstream fo ("FOOT.OUT");
int n, k,a[nmax],
b[nmax],c[nmax],d[nmax]={0},ib,ie,l,t;
int main()
{fi>>n>>k; ib=1; ie=0;b[0]=0;
for (int i=1; i<=n;++i)
{fi>>t; a[i]=t;
while(t<=b[ie])ie--;
b[++ie]=t; c[ie]=i;
if(ie<ib)ib=ie;
while(i-c[ib]>=k)ib++;d[i]=c[ib];
}
memset(c,0,sizeof(c)); t=0;
for(int i=1;i<=n;++i) if(d[i]>0) c[d[i]]+=2;
for(int i=1;i<=n;++i) t+=a[i]*c[i];
fo<<t<<endl;
for(int i=1;i<=n;++i) fo<<c[i]<<" ";
}

```

**Tạo heap mìn**

**Truy nhập tới đỉnh**

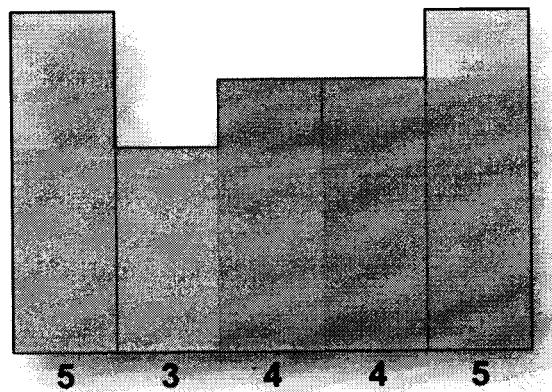
**Ghi nhận ngày mua**

**Tính chi phí**

## HÀNG RÀO

Tên chương trình: FENCE.???

Steve phải sơn lại hàng rào cũ của nhà mình. Hàng rào bao gồm  $n$  thanh ghép khít nhau, mỗi thanh có độ rộng  $1\text{cm}$  và có độ cao có thể khác nhau, thanh thứ  $i$  có độ cao  $h_i$ . Để hoàn thành công việc nhanh chóng và dễ dàng Steve mua con lăn Deluxe. Con lăn có độ rộng  $x \text{ cm}$ . Khi sơn, con lăn phải luôn song song với mặt đất và tiếp xúc đầy đủ với hàng rào, nếu có chỗ trống, sơn sẽ bắn tung tóe làm bẩn mọi thứ xung quanh hoặc tạo ra các giọt sơn gồ ghề trên hàng rào. Vì vậy mỗi lượt, Steve phải chọn  $x$  thanh liên tiếp, đầy con lăn từ đáy hàng rào lên đến đỉnh của thanh thấp nhất trong số các thanh đã chọn. Có thể có trường hợp một thanh được sơn tới vài ba lần, nhưng không sao, con lăn vừa phun sơn vừa xoa đều nên không để lại ánh hưởng gì đáng kể. Steve hoàn thiện nốt phần còn lại chưa sơn của hàng rào bằng chổi sơn – một công việc nhàn chán và tốn thời gian. Vì vậy Steve phải tìm cách dùng con lăn sao cho tổng diện tích sơn bằng chổi là ít nhất.



Ví dụ, hàng rào có 5 thanh với độ cao tương ứng là 5, 3, 4, 4, 5 và con lăn có độ rộng bằng 3. Steve đầy con lăn 2 lượt – lần đầu với các thanh 1, 2, và 3, lần thứ 2 – với các thanh 3, 4 và 5 (thanh thứ 3 được sơn 2 lần). Tổng diện tích phải sơn tay là 4.

**Yêu cầu:** cho  $n$ ,  $x$  và  $h_i$  ( $1 \leq n \leq 10^6$ ,  $1 \leq h_i \leq 10^6$ ,  $i = 1 \dots n$ ,  $1 \leq x \leq 10^5$ ,  $x \leq n$ ). Hãy xác định diện tích tối thiểu phải sơn tay và số lượt đầy con lăn.

**Dữ liệu:** Vào từ file văn bản FENCE.INP:

- Dòng đầu tiên chứa 2 số nguyên  $n$  và  $x$ ,
- Dòng thứ 2 chứa  $n$  số nguyên  $h_1, h_2, \dots, h_n$ .

**Kết quả:** Đưa ra file văn bản FENCE.OUT hai số nguyên:

- Dòng thứ nhất chứa diện tích còn lại phải sơn bằng chổi,
- Dòng thứ 2 – số lượt đầy con lăn.

**Ví dụ:**

FENCE.INP
5 3
5 3 4 4 5

FENCE.OUT
3
2



V39 COCI4.D

```

#include <iostream>
#include <algorithm>
#include <ctime>

using namespace std;
const int nx=1000001;
pair <int, int> a[nx];
int n,x,h[nx],ib,ie,b[nx],c[nx],p[nx],q[nx],t,j;
long long r;
ifstream fi ("FENCE.INP");
ofstream fo ("FENCE.OUT");

int main()
{fi>>n>>x;
 for (int i=1; i<=n; ++i) fi>>h[i];
 ib=1;ie=1; t=nx; memset(p,0,sizeof(p)); b[0]=0;

 for (int i=1;i<x;++i)
 { while (b[ib-1]>=h[i]) ib--;
   b[ib]=h[i];c[ib++]=i;
 }

 for (int i=x;i<=n;++i)
 {while (b[ib-1]>h[i]) {ib--;if(ib<ie) ie=ib;}
   b[ib]=h[i];c[ib++]=i; ;while(i-c[ie]>=x) ie++;
   p[i]=b[ie];
 }

 ib=0; ie=1;b[0]=nx+1;
 for (int i=n; i>=1;--i)
 {while (b[ib]<=p[i]) ib--;
  b[++ib]=p[i]; c[ib]=i; if (ib<ie) ie=ib;
  while (c[ie]-i>=x) ie++;
  q[i]=b[ie];
 }

 r=0;
 for (int i=1; i<=n;++i) r=r+(long long) (h[i]-q[i]);
 t=0; j=0;
 for (int i=1;i<=n;++i)
 { if (q[i]!=q[i-1]) {t++; j=1;}
   else {j++; if (j>x) {j=1;t++;}}
 }

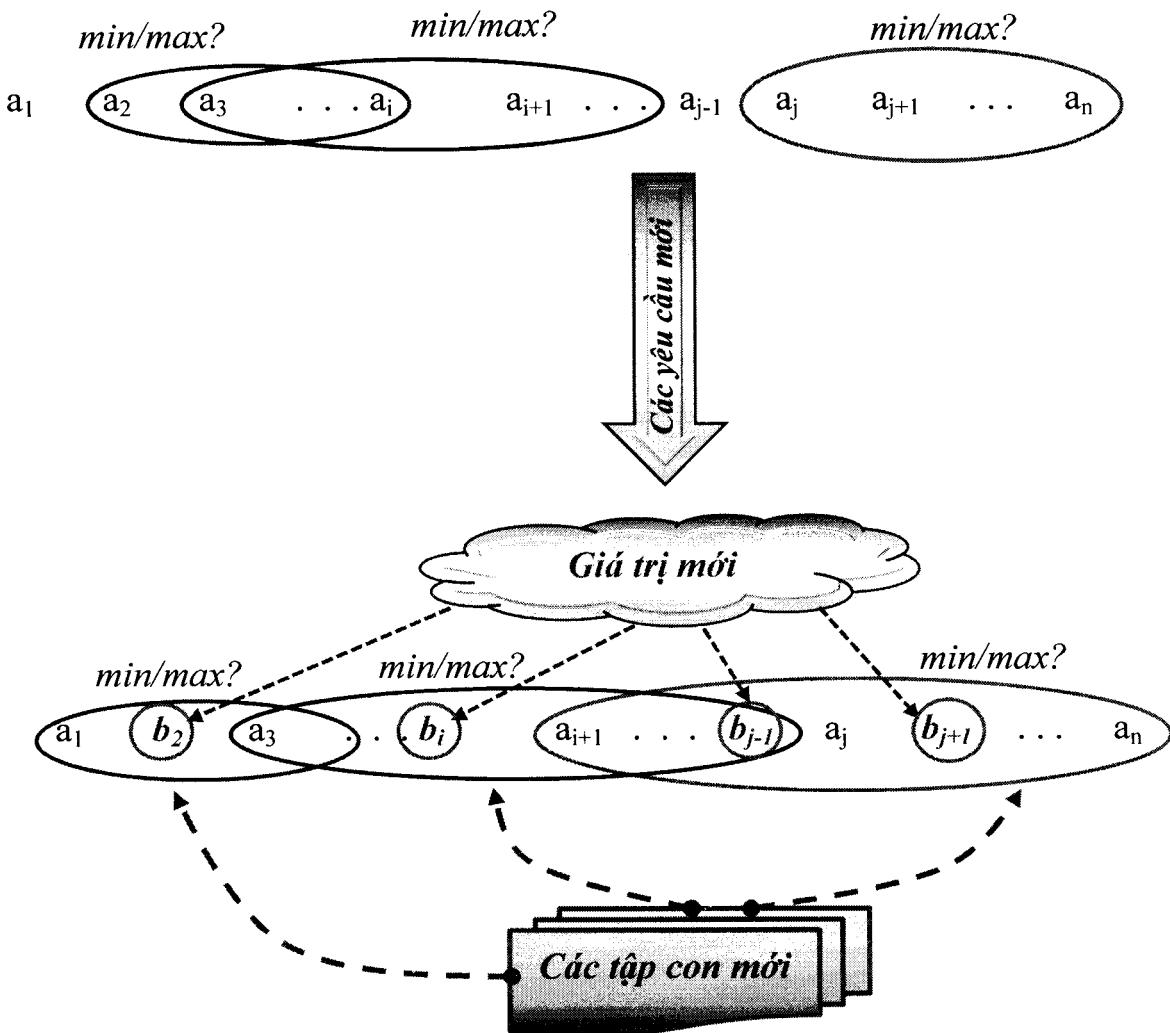
 fo<<r<<endl;
 fo<<t;

 fi.close(); fo.close();
}

```

### 2.3- Tìm cực trị trên các tập con thay đổi của tập biến động

Xét các bài toán trong đó một số giá trị  $a_i$  của mảng  $A = (a_1, a_2, \dots, a_n)$  có thể thay đổi trong quá trình xử lý. Đòi hỏi phải *nhiều lần* tìm min hoặc max của các phần tử nằm ở các vị trí từ  $p$  đến  $q$  với  $p$  và  $q$  chỉ được xác định trước khi phải



tìm min/max.

Các đại lượng  $a_i$  thay đổi giá trị và các tập con cần tìm min/max có thể cho tường minh dưới dạng các truy vấn hoặc có thể phát sinh trong quá trình tính toán, xử lý dữ liệu.

Công cụ vạn năng để quản lý min/max là cây nhị phân.

Cây nhị phân được xét ở đây là cây có nút gốc được đánh số 1. Mỗi nút của cây, trừ các nút lá có một hoặc 2 nút con. Nếu nút được đánh số là  $x$  có 2 nút con thì các nút con của nó có số là  $2 \times x$  và  $2 \times x + 1$ . Nếu nút  $x$  có một nút con thì nút con có số là  $2 \times x$ .

Xét dãy số  $A = (a_0, a_1, \dots, a_{n-1})$ . Gọi  $k$  là số nguyên nhỏ nhất thỏa mãn  $n \leq 2^k$ . Các nút  $2^k, 2^k+1, 2^k+2, \dots, 2^k+n-1$  là các nút lá của cây và quản lý các phần tử

$a_0, a_1, a_2, \dots, a_{n-1}$ . Đường đi từ nút lá bất kỳ về gốc bao giờ cũng qua đúng  $k+1$  nút (kể cả nút lá và nút gốc). Điều này đảm bảo mỗi lần duyệt cây sẽ có độ phức tạp là  $O(\log n)$ . Để thuận tiện trong việc lập trình, nếu cần làm việc với cả min và max thì nên tổ chức 2 cây riêng biệt, một cây quản lý min, cây khác – quản lý max. Mỗi nút của cây cần quản lý 2 đơn vị thông tin: *giá trị min/max* và *nơi đạt giá trị đó*.

Các phép xử lý cây bao gồm:

- ◆ Khởi tạo cây,
- ◆ Cập nhật cây khi giá trị một nút lá thay đổi,
- ◆ Tìm cực trị trong dãy những phần tử ở các vị trí liên tiếp từ  $p$  đến  $q$ .

Việc xử lý 2 cây là tương tự như nhau, vì vậy dưới đây ta chỉ tập trung xem xét cây quản lý min.

### 2.3.1 – Khởi tạo cây

Với  $n$  và  $A = (a_0, a_1, \dots, a_{n-1})$  cho trước, việc đầu tiên là phải xác định vị trí các nút lá trong cây, cụ thể là vị trí nút quản lý  $a_0$ . Các bài toán cần giải quyết thường có  $n$  với bậc  $10^6$  (và triệu đến vài chục triệu), phù hợp với khả năng cấp phát bộ nhớ của các hệ thống lập trình và đủ để kiểm nghiệm trình độ kiến thức cũng như khả năng lập trình của người giải quyết.

#### a – Khai báo dữ liệu

Gọi mảng lưu trữ cây là  $b$ . Nếu khai báo tĩnh, trong trường hợp tổng quát,  $b$  cần có kích thước không nhỏ hơn  $4 \times n$ . Mỗi phần tử của  $b$  là một cặp dữ liệu **pair**<kiểu dữ liệu của  $a$ , int>. Ví dụ, với  $a_i$  nguyên, có giá trị tuyệt đối không vượt quá  $10^{18}$  thì  $b$  phải được khai báo với kiểu dữ liệu **pair**<int64\_t, int>.

#### b – Xác định vị trí nút lá

Như đã nói ở trên, đại lượng  $a_0$  được quản lý bởi nút  $m = 2^k$ , trong đó  $k$  là số nguyên nhỏ nhất thỏa mãn  $n \leq 2^k$ . Giá trị  $m$  chỉ cần xác định một lần, vì vậy

```
for(int i=24;i>=0;--i)
    if(n&(1<<i)) {k=i; break;}
m=1<<(k+1);
```

không cần thiết phải xử lý quá cầu kỳ:

#### c – Gán giá trị cho nút lá

Khi có nhu cầu làm việc với nhiều cây trên cùng một bộ dữ liệu có thể tổ chức cây với nút lá ảo. Giá trị nút lá được cung cấp bằng một hàm riêng. Trong dữ liệu tổ chức các cây không có bộ nhớ dành cho việc lưu trữ giá trị nút lá. Như vậy sẽ

tiết kiệm bộ nhớ và đơn giản hóa khâu cập nhật bộ dữ liệu thuộc nút lá. Giá phải trả là quy trình xử lý phức tạp hơn đôi chút và chương trình sẽ kém tường minh.

Khi không phải sử dụng quá nhiều cây và quy trình cập nhật dữ liệu ở nút lá không quá phức tạp thì thuận tiện nhất là tổ chức nút lá tường minh, lưu trữ giá trị các nút lá ở mỗi cây. Như vậy, khi bộ dữ liệu ban đầu thay đổi cần lặp lại phép cập nhật lá ở tất cả các cây có liên quan.

Việc gán giá trị cho các nút lá nên tiến hành một cách đơn giản nhất. Kỹ thuật tổ chức hàng rào, tuy thuận tiện cho việc xử lý tiếp theo nhưng có thể sẽ tốn rất nhiều thời gian và bộ nhớ lưu trữ.

Gán giá trị cho nút lá:

```
for(int i=0;i<n;++i)b[i+m]=make_pair(a[i],i);
```

#### *d – Tính giá trị các nút còn lại của cây*

Gọi các nút không phải nút lá là nút trong. Nút trong tham gia quản lý dữ liệu có chỉ số lớn nhất  $x = (m+n-1)/2$ .

Các nút của cây chia thành  $k+1$  lớp. Nút gốc (nút chỉ số 1) thuộc lớp 1 và chứa thông tin tổng hợp của các nút lớp 2. Các nút 2 và 3 thuộc lớp 2 và chứa thông tin tổng hợp của các nút lớp 3. Các nút 4, 5, 6, 7 thuộc lớp 3 và chứa thông tin tổng hợp của các nút lớp 4, ... Lớp  $k+1$  chứa các nút có chỉ số từ  $m=2^k$  đến  $m+n-1$  là các nút lá, chứa thông tin về dãy cần xử lý.

Một số bài toán đòi hỏi phải làm việc với nhiều cây hình thành trong quá trình xử lý. Việc *xóa hoặc gán giá trị đầu* cho cây sẽ làm *tăng một cách đáng kể thời gian thực hiện* chương trình. Chính vì vậy giải thuật khởi tạo vạn năng và hợp lý phải dựa trên cơ sở mảng bộ nhớ dùng để chứa cây có thể sẵn chứa giá trị bất kỳ.

Trừ lớp 1 và lớp  $k+1$ , ở mỗi lớp còn lại nút cuối cùng trong lớp (nút có chỉ số lớn nhất thuộc lớp) có thể chỉ có một nút con thực sự. Nút con duy nhất đó sẽ có chỉ số chẵn và việc tổng hợp thông tin đơn thuần chỉ là sao lại thông tin từ nút con duy nhất của nó.

Giải thuật khởi tạo giá trị các nút trong có dạng:

```

y=m+n-1; z=y/2; x=z;

for(int i=x;i>0;--i)

{u=i<<1; v=u+1;

if(i==z)      //Xử lý nút biên

{if(v>y){b[i]=b[u]; y>>=1; z>>=1; continue;} //Trường hợp có 1 nút con

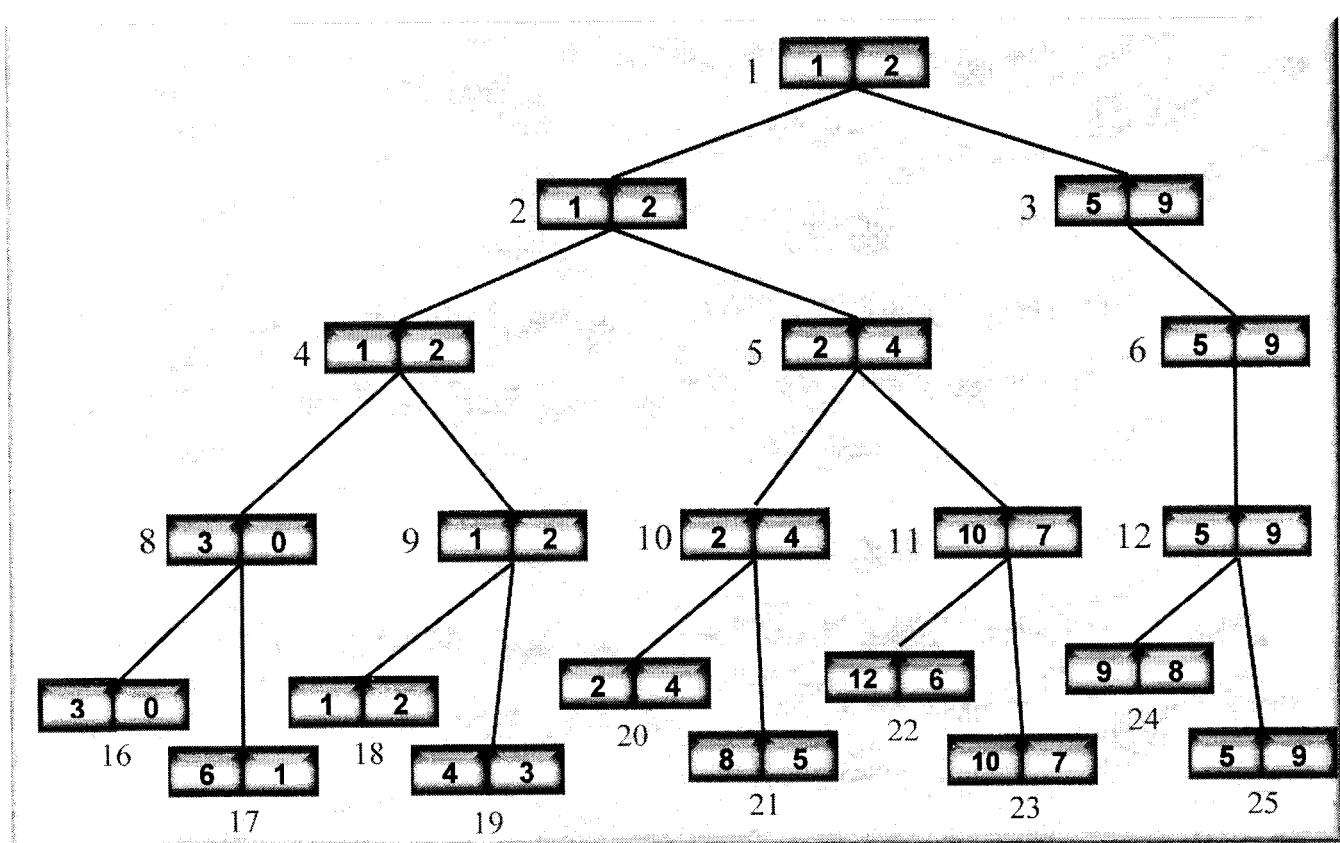
else{y>>=1; z>>=1;}} //Trường hợp có 2 nút con

b[i]=(b[u].first<=b[v].first)?b[u]:b[v];

}

```

Ví dụ, với  $n = 10$  và  $A = (3, 6, 1, 4, 2, 8, 12, 10, 9, 5)$  ta sẽ nhận



được cây:

Nút số 5 với giá trị  $(2, 4)$  cho biết  $\min\{a_4, a_5, a_6, a_7\}$  bằng 2 và đạt được ở phần tử  $a_4$ . Nút số 11 với giá trị  $(10, 7)$  cho biết  $\min\{a_6, a_7\}$  bằng 10 và đạt được ở phần tử  $a_7$ .

### 2.3.2 – Cập nhật cây

Tồn tại hai loại cập nhật:

- Thay đổi giá trị của một phần tử  $a_p$ ,
- Loại bỏ phần tử  $a_h$  khỏi cây.

#### a – Thay đổi giá trị

Phần tử  $a_p$  nhận giá trị mới là  $v$ . Cần phải duyệt lại các nút trên đường đi từ nút  $m+p$  đến gốc, thay đổi giá trị của nút nếu cần thiết.

Quá trình cập nhật sẽ kết thúc khi thỏa mãn một trong 2 điều kiện:

- Đã cập nhật giá trị ở nút gốc,
- Khi giá trị  $v$  không tham gia vào việc xác định cực trị.

Điều cần lưu ý trong cập nhật là nhận biết và xử lý nút biên phải ở mỗi lớp.

```
void upd_v(pair<int,int> pa[],int ps,int newv)
{pa[m+ps]=make_pair(newv,ps);
 int tg=ps+m,kt=0;
 y=m+n-1;z=y/2;
 while(tg!=1 && !kt)
 {tg>>=1; u=tg<<1;v=u+1;
 if(tg==z)
 {if(v>y){pa[tg]=pa[u];y>>=1;z>>=1;continue;}
 else{y>>=1;z>>=1;}
 }
 pa[tg]=(pa[u].first<=pa[v].first)?pa[u]:pa[v];
 kt=(pa[tg].first<newv);
 }
 }
```

Hàm **upd\_v** có 3 tham số:

- $pa$  – cây cần cập nhật,
- $ps$  – vị trí có giá trị thay đổi,
- $newv$  – giá trị mới ở vị trí  $ps$ .

Biến  $kt$  phục vụ nhận biết kết thúc sớm cập nhật. Độ phức tạp của giải thuật cập nhật là  $O(\log n)$ .

#### b – Loại bỏ phần tử

Việc loại bỏ phần tử  $a_h$  ra khỏi cây có thể thực hiện bằng việc hàm **upd\_v** với  $ps=h$  và với  $newv > \max\{a_i, i = 0 \div n-1\}$ . Chỉ số của các phần tử của A đứng sau  $h$  sẽ bị dồn lên.

Để tránh đánh lại chỉ số và làm lại cây ta vẫn giữ nguyên các chỉ số ban đầu, nhưng quy đổi chỉ số mới về chỉ số ban đầu. Bằng cấu trúc dữ liệu *cây Fenwick* ghi nhận các chỉ số đã bị xóa và với giải thuật *tìm kiếm nhị phân*, việc quy đổi chỉ số mới về chỉ số ban đầu có thể thực hiện với độ phức tạp  $O(\log n)$ .

### 2.3.3 – Xử lý các truy vấn tìm cực trị

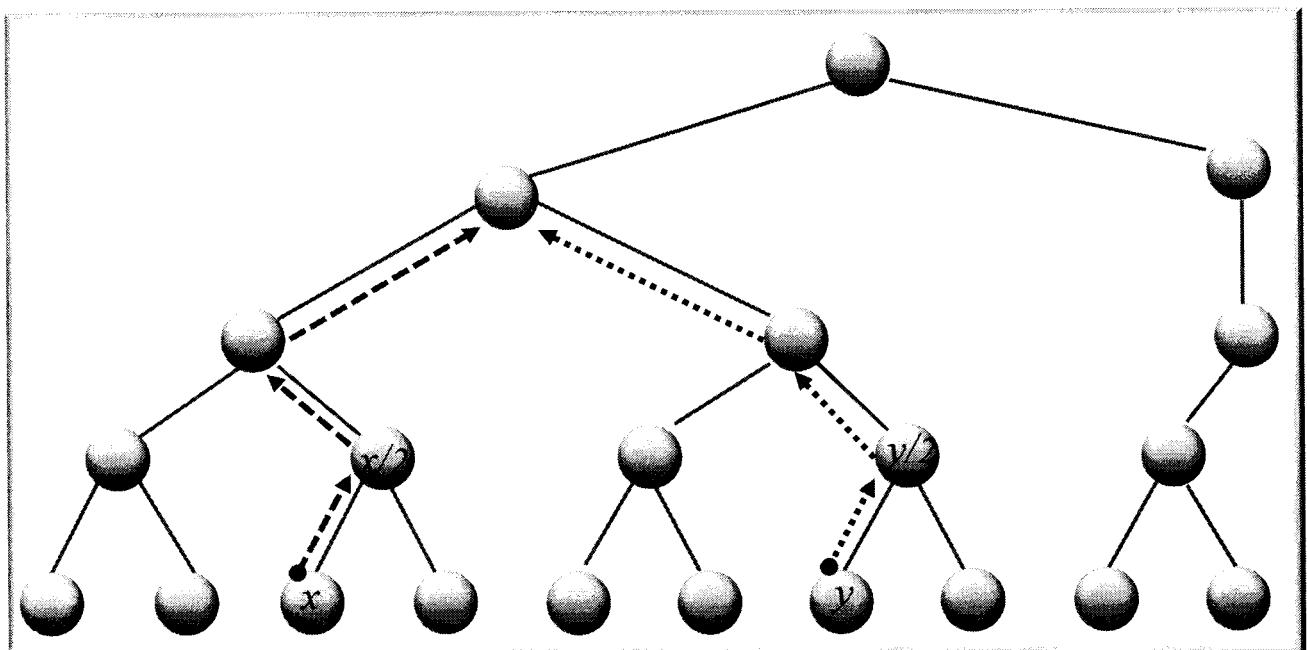
Xét các truy vấn tìm min/max, mỗi truy vấn được xác định bởi cặp số nguyên  $p$  và  $q$  ( $0 \leq p \leq q < n$ ) yêu cầu xác định min/max của tập giá trị  $\{a_p, a_{p+1}, \dots, a_q\}$ . Các truy vấn này có thể cho đan xen với các yêu cầu cập nhật. Nếu trước đó có các yêu cầu xóa thì phải tính lại  $p$  và  $q$  tương ứng với dãy ban đầu.

Các chỉ số  $p, q$  tương ứng với các nút lá  $m+p$  và  $m+q$ . Từ mỗi nút lá nói trên ta sẽ đi ngược về gốc. Tại mỗi nút đi qua cần kiểm tra và cập nhật lại kết quả min/max trung gian. Các min/max trung gian chỉ được cập nhật lại khi thỏa mãn 2 yêu cầu:

- Chỉ số của phần tử ở nút đang xét nằm trong khoảng  $[p, q]$ ,
- Giá trị min (max) lưu ở nút bé hơn (lớn hơn) cực trị trung gian hiện có.

Từ nút  $x$  chuyển sang nút trên nó tương đương với việc từ nút  $x$  chuyển tới nút  $x/2$ .

Xuất phát từ các nút ban đầu  $x = m+p$  và  $y = m+q$  quá trình duyệt ngược về gốc được thực hiện cho đến khi 2 đường duyệt gặp nhau ở nút cha chung gần nhất.



Kết quả cuối cùng có thể dễ dàng nhận được bằng cách so sánh 2 cực trị trung gian.

Hàm `get_min(b,p,q)` dưới đây sẽ trả về chỉ số của phần tử min trong khoảng  $[p, q]$ .

```
int get_min(pair<int,int>pa[],int lt,int rt)
{
    int tl,tr,ir;
    pair<int,int> tgl,tgr;
    tl=m+lt;tr=m+rt; tgl=pa[tl]; tgr=pa[tr];
    while(tl!=tr)
    {
        if(pa[tl].second>=lt && pa[tl].second<=rt && pa[tl].first<tgl.first)
            tgl=pa[tl];
        if(pa[tr].second>=rt && pa[tr].second<=rt && pa[tr].first<tgr.first)
            tgr=pa[tr];
    }
    ir=(tgl.first<=tgr.first)?tgl.second:tgr.second;
    return(ir);
}
```

Độ phức tạp của việc xử lý mỗi truy vấn tìm cực trị là O(logn).

## VO46. THỨC ĂN NHANH

Tên chương trình: FASTFOOT.???

Khuôn viên của trường đại học có hình vuông, được chia thành  $n \times n$  ô. Ở mỗi ô có một tòa nhà, 2 tòa nhà ở cặp ô kề cạnh được nối với nhau bằng hành lang có mái che. Để tạo điều kiện cho sinh viên có nhiều thời gian học tập và nghiên cứu khoa học tại mỗi ô có lắp một máy bán thức ăn tự động, mỗi máy chỉ bán một loại đồ ăn ví dụ chỉ bán cà phê hay chỉ bán bánh mì kẹp thịt. Ký túc xá ở ô  $(1, 1)$  – ô ở góc trên trái. Giảng đường ở ô  $(n, n)$  tại góc dưới phải. Sinh viên luôn đi từ ký túc xá tới giảng đường theo đường đi ngắn nhất. Người ta nhận thấy sinh viên hay mua đồ ăn nhất khi đi lên lầu hoặc khi từ trên lầu về ký túc xá, vì vậy cần xem lại hệ thống đặt máy tự động sao cho trên đường đi số thức ăn có thể mua càng đa dạng càng tốt. Máy ở ô  $(i, j)$  bán thức ăn loại  $a_{i,j}$ . Số lượng máy cùng bán loại thức ăn này trên đường đi ngắn nhất từ ký túc xá đến giảng đường đi qua ô  $(i, j)$  và chứa nhiều máy nhất cùng bán thức ăn  $a_{i,j}$  được gọi là độ lắp của máy tự động ở ô  $(i, j)$ .

Hãy xác định với mỗi giá trị  $k$  trong phạm vi từ 1 đến  $2 \times n - 1$  có bao nhiêu máy tự động có độ lắp  $k$ .

**Dữ liệu:** Vào từ file văn bản FASTFOOT.INP:

- ◆ Dòng đầu tiên chứa số nguyên  $n$  ( $2 \leq n \leq 1500$ ),
- ◆ Dòng thứ  $i$  trong  $n$  dòng sau chứa  $n$  số nguyên  $a_{i,1}, a_{i,2}, \dots, a_{i,n}$  ( $1 \leq a_{i,j} \leq n^2$ ,  $j = 1 \div n$ ).

**Kết quả:** Đưa ra file văn bản FASTFOOT.OUT trên một dòng  $2 \times n - 1$  số nguyên – các giá trị tìm được.

**Ví dụ:**

FASTFOOT.INP	FASTFOOT.OUT
5	2 4 9 0 0 1 1 8 0
1 4 1 3 5	
2 1 4 1 2	
5 1 1 4 5	
3 5 1 1 2	
4 3 5 1 1	



VO46 ROI20140410 G

```

#include <fstream>
#include <ctime>
using namespace std;
typedef pair<int, int> pi2;
typedef pair<int,pi2> pi3;
ifstream fi ("fastfoot.inp");
ofstream fo ("fastfood.out");
int
b[6001]={0},c[6001]={0},cp[6001]={0},bp[6001]={0},bm[6001]={0},cm[6001]={0},
n,n2,nb,k,t,t1,t2,bx,cx,cpx;
pi3 a[2250002],p;
int res[1501][1501]={0},ans[3000]={0},ib,ie,m;

void get_bx(int x)
{int tg,u,v,tgl,tgr;
 tgr=nb+x;tgl=nb+1;
 bx=(bm[tgl]==m)? b[tgl]:0;
 while(tgl!=tgr)
 {u=tgr>>1; v=tgl>>1;
 if(bm[2*u]==m) if( bx<b[2*u] && bp[2*u]<=x)bx=b[2*u];
 if(bm[2*u+1]==m) if( bx<b[2*u+1] && bp[2*u+1]<=x)bx=b[2*u+1];
 if(bm[2*v]==m) if( bx<b[2*v] && bp[2*v]<=x)bx=b[2*v];
 if(bm[2*v+1]==m) if( bx<b[2*v+1] && bp[2*v+1]<=x)bx=b[2*v+1];
 tgl>>=1; tgr>>=1;
 }
}
void get_cx(int x)
{int tg,u,v,tgl,tgr;
 tgl=nb+x;tgr=nb+n;
 cx=(cm[tgr]==m)?c[tgr]:0;
 while(tgr!=tgl)
 {u=tgl>>1; v=tgr>>1;
 if(cm[2*u]==m) if( cx<c[2*u] && cp[2*u]>=x)cx=c[2*u];
 if(cm[2*u+1]==m) if( cx<c[2*u+1] && cp[2*u+1]>=x)cx=c[2*u+1];
 if(cm[2*v]==m) if( cx<c[2*v] && cp[2*v]>=x)cx=c[2*v];
 if(cm[2*v+1]==m) if( cx<c[2*v+1] && cp[2*v+1]>=x)cx=c[2*v+1];
 tgl>>=1; tgr>>=1;
 }
}
void upd_bx(int x,int y)
{int tg,tgl,tgr,u,v;
 tgr=nb+x;tgl=nb+1;
 if(bm[tgr]==m)if(b[tgr]>y || (b[tgr]==y && bp[tgr]<=x))return;
 b[tgr]=y;bp[tgr]=x;bm[tgr]=m;
 while(tgr!=1)
 {u=tgr>>1;
 if(bm[2*u]!=m){b[2*u]=0; bm[2*u]=m;bp[2*u]=n+1;}
 if(bm[2*u+1]!=m){b[2*u+1]=0; bm[2*u+1]=m;bp[2*u+1]=n+1;}
 if(bm[u]!=m){bm[u]=m;b[u]=0;bp[u]=n+1;}
 if(b[u]<b[2*u]) {b[u]=b[2*u];bp[u]=bp[2*u];}else if(b[u]==b[2*u] &&
 bp[u]>bp[2*u])bp[u]=bp[2*u];
 if(b[u]<b[2*u+1]) {b[u]=b[2*u+1];bp[u]=bp[2*u+1];}else if(b[u]==b[2*u+1] &&
 bp[u]>bp[2*u+1])bp[u]=bp[2*u+1];
 tgr=u;tgl>>=1;
 }
}
void upd(cx,int y)
{int tg,tgl,tgr,u,v;
 tgr=nb+n;tgl=nb+x;
 if(cm[tgl]==m)if(c[tgl]>y || (c[tgl]==y && cp[tgr]>=x))return;
 c[tgl]=y;cp[tgl]=x;cm[tgl]=m;
 while(tgl!=1)
 {u=tgl>>1;
 if(cm[2*u]!=m){c[2*u]=0; cm[2*u]=m;cp[2*u]=0;}

```

```

if(cm[2*u+1]!=m){c[2*u+1]=0; cm[2*u+1]=m;cp[2*u+1]=0;}
    if(cm[u]!=m){cm[u]=m;c[u]=0;cp[u]=0;}
    if(c[u]<c[2*u]){c[u]=c[2*u];cp[u]=cp[2*u];}else if(c[u]==c[2*u] &&
cp[u]<cp[2*u])cp[u]=cp[2*u];
    if(c[u]<c[2*u+1]){c[u]=c[2*u+1];
cp[u]=cp[2*u+1];}else if(c[u]==c[2*u+1] && cp[u]<cp[2*u+1])cp[u]=cp[2*u+1];
    tgl=u;tgr>>=1;
}
}

void xly_b(int x,int y)
{int i1,j1;
for(int i=x;i<=y;++i)
{i1=a[i].second.first; j1=a[i].second.second;
get_bx(j1); t1=bx+1;res[i1][j1]=t1;
upd_bx(j1,t1);
}
}

void xly_c(int x, int y)
{int i1,j1,tg;
//memset(c,0,sizeof(c));memset(cp,0,sizeof(cp));
for(int i=y;i>=x;--i)
{i1=a[i].second.first; j1=a[i].second.second;
get(cx(j1); t1=cx+1;res[i1][j1]+=t1;
upd(cx(j1,t1);
}
}

void get_ans()
{for(int i=1;i<=n;++i)
for(int j=1;j<=n;++j)
{t=res[i][j]-1;
++ans[t];
}
for(int i=1;i<=2*n-1;++i) fo<<ans[i]<<' ';
}
int main()
{clock_t aa=clock();
fi>>n; k=0;
for(int i=1;i<=n;++i)
for(int j=1;j<=n;++j)
{fi>>t;p=make_pair(t,make_pair(i,j));
a[++k]=p;
}
n2=n*n;
for(int i=20;i>=0;--i) if(((n>>i) & 1) == 1){t=i+1;break;}
nb=1<<t;--nb; m=0;
sort(a,a+n2+1); a[n2+1].first=0;
ib=1;t=a[1].first;
for(int i=1;i<=n2+1;++i)
if(t!=a[i].first){ie=i-1;t=a[i].first; ++m;
xly_b(ib,ie);xly_c(ib,ie); ib=i;}
get_ans();
clock_t bb=clock();
fo<<"\nTime: "<<(double)(bb-aa)/1000<<" sec.";
}
}

```

### 13L. BẠCH TUYẾT

Tên chương trình: SNOW\_WH.???

Bạch Tuyết sống cùng với  $n$  chú lùn ở một ngôi làng nhỏ có 7 quả đồi và 7 hồ. Các chú lùn suốt ngày chỉ có ăn và chơi trò chơi ghép hình. Muốn thay đổi lối sống không lành mạnh này Bạch Tuyết quyết định tổ chức các hoạt động thể thao. Các chú lùn có chiều cao nguyên, nằm trong phạm vi từ 1 đến  $n$  và không có 2 người nào cùng chiều cao. Ở sân tập các chú phải xếp hàng theo thứ tự cao dần. Tuy nhiên lối sống trì trệ đã làm các chú lùn mất khả năng phân biệt cao thấp và đứng thành một hàng lộn xộn khi tập trung. Để huấn luyện khả năng quan sát Bạch Tuyết đưa ra 2 loại lệnh:

**1  $x$   $y$**  – Hai người ở các vị trí  $x$  và  $y$  đổi chỗ cho nhau,

**2  $a$   $b$**  – Cho biết các người có độ cao  $a$ ,  $a+1$ ,  $a+2$ , . . . ,  $b$  có đứng (không nhất thiết theo trình tự độ cao đã nêu) thành một nhóm liên tiếp nhau hay không?

Có tất cả  $m$  lệnh được đưa ra. Với mỗi lệnh loại thứ 2 hãy đưa ra câu trả lời **YES** hoặc **NO**.

**Dữ liệu:** Vào từ file văn bản SNOW\_WH.INP:

- ◆ Dòng đầu tiên chứa 2 số nguyên  $n$  và  $m$  ( $2 \leq n, m \leq 2 \times 10^5$ ),
- ◆ Dòng thứ 2 chứa  $n$  số nguyên xác định độ cao các chú lùn trong hàng tính từ trái sang phải,
- ◆ Mỗi dòng trong  $m$  dòng sau chứa một lệnh được đưa ra.

**Kết quả:** Đưa ra file văn bản SNOW\_WH.OUT các câu trả lời, mỗi câu trên một dòng.

**Ví dụ:**

SNOW_WH.INP
5 3
2 4 1 3 5
2 2 5
1 3 1
2 2 5

SNOW_WH.OUT
NO
YES



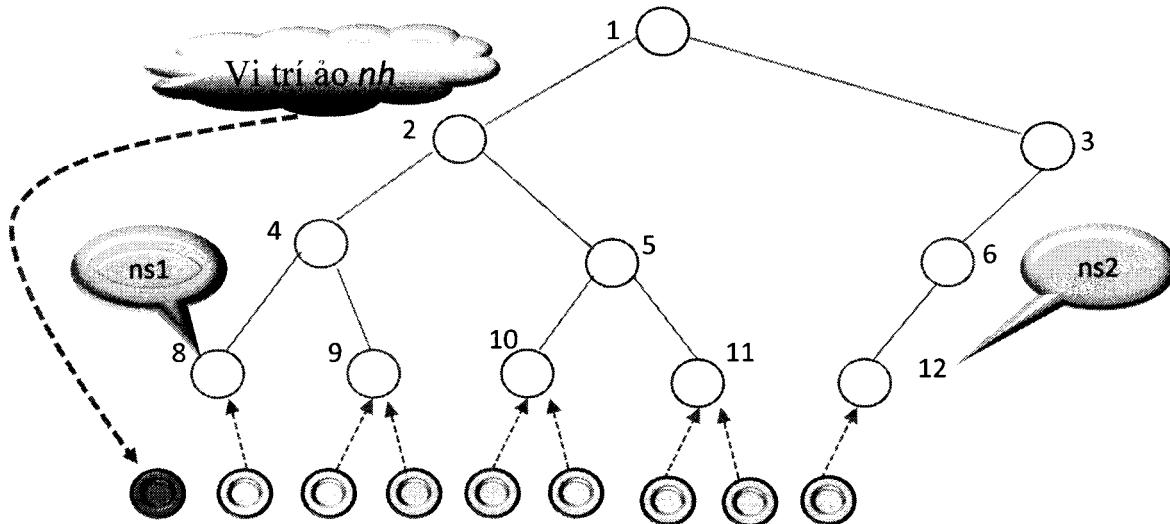
## Giải thuật

Xét động thời hoán vị  $\{h_i\}$  và  $\{p_i\}$ :

```
for(int i=1;i<=n;++i) fi>>h[i];
for(int i=1;i<=n;++i)p[h[i]]=i;p[n+1]=p[n];
```

Sử dụng cây nhị phân quản lý min và max trong các đoạn của P.

```
for(int i=31;i>=0;--i) if((n>>i)&1 ==1){nt=i+1; break;}
nh=1<<nt; ns2=(nh+n)/2; ns1=nh>>1;
dtr_bt();
```



Khởi tạo cây:

```
void dtr_bt()
{nt=(nh+1)/2;
 int u,v;
 for(int i=ns1;i<=ns2;++)
 { u=i*2-nh+1; v=u+1;
   sm[i]=dtr_mn(p[u],p[v]);sx[i]=dtr_mx(p[u],p[v]);
 }
 sm[ns2+1]=sm[ns2];sx[ns2+1]=sx[ns2];
 for(int i=ns1-1;i>=1;--i)
 {u=i*2; v=u+1;sm[i]=dtr_mn(sm[u],sm[v]);
  sx[i]=dtr_mx(sx[u],sx[v]);}
}
```

Dữ liệu để rồi dùng chung cho cả 2 cây.

Đổi chỗ: đồng thời ở cả 2 hoán vị và cập nhật cây quản lý Min-Max:

```

void upd_t(int x,int y)
{int t,t1,t2,u,x0,y0;
 t=p[h[x]];p[h[x]]=p[h[y]];p[h[y]]=t;p[n+1]=p[n];
 t=h[x];h[x]=h[y];h[y]=t;x0=h[x]; y0=h[y];
 t1=(nh+x0-1)/2;t2=(nh+y0-1)/2; if(x0%2==1)u=x0+1; else u=x0-1;
 sm[t1]=dtr_mn(p[u],p[x0]); sx[t1]=dtr_mx(p[u],p[x0]);
 if(y0%2==1)u=y0+1; else u=y0-1;
 sm[t2]=dtr_mn(p[u],p[y0]); sx[t2]=dtr_mx(p[u],p[y0]);
 sm[ns2+1]=sm[ns2];sx[ns2+1]=sx[ns2];
 t1>>=1;t2>>=1;
 while(t1>=1)
 {sm[t1]=dtr_mn(sm[2*t1],sm[2*t1+1]);
  sx[t1]=dtr_mx(sx[2*t1],sx[2*t1+1]);
  t1>>=1;
 }
 while(t2>=1)
 {sm[t2]=dtr_mn(sm[2*t2],sm[2*t2+1]);
  sx[t2]=dtr_mx(sx[2*t2],sx[2*t2+1]);
  t2>>=1;
 }
}

```

Đổi chỗ 2 phần tử vị trí **x**, **y** trong hoán vị **H** tương ứng với việc đổi chỗ 2 phần tử vị trí **h<sub>x</sub>** và **h<sub>y</sub>** trong **P**,

Để an toàn: cần chỉnh lý phần tử hàng rào p<sub>n+1</sub>,

Vị trí trong cây tương ứng với các nút ảo (nút ngoài): t1=(nh+x0-1)/2; t2=(nh+y0-1)/2;

Cập dữ liệu liên hợp: x → (h<sub>x</sub>, u): x0=h[x]; if(x0%2==1)u=x0+1; else u=x0-1;

Cập nhật cây: thông tin nút **t** được cập nhật từ thông tin 2 nút con của nó: 2**t** và 2**t**+1.

Xét đoạn liên tục: các giá trị a ÷ b trong P → [a, b] trong Q chứa các số nguyên ∈ [x, x+b-a].

Xác định t0 – nút cha chung gần nhất của 2 nút dữ liệu x, y:

```

t1=x+nh-1;t2=y+nh-1;
while(t1!=t2){t1/=2;t2/=2;}
t0=t1;

```

Duyệt 2 đường đi: trái từ x và phải từ y (x < y) để xác định cực trị (min, max) trong khoảng [x, y]:

Ở đường duyệt trái: chỉ cập nhật cực trị khi tới nút mới từ nút con phải (t1 chẵn):

```

tml=p[x];txl=tml; if((x&1)==1){tml=sm[t1];txl=sx[t1];}

while(t1>t0){
    if((t1&1)==0 && t1/2!=t0)
        {tml=dtr_mn(tml,sm[t1+1]);txl=dtr_mx(txl,sx[t1+1]);}
    t1>>=1;
}

```

Ở đường duyệt phải: chỉ cập nhật cực trị khi tới nút mới từ nút con trái (t1 lẻ):

```

tmr=p[y];txr=tmr; if((y&1)==0){tmr=sm[t2];txr=sx[t2];}

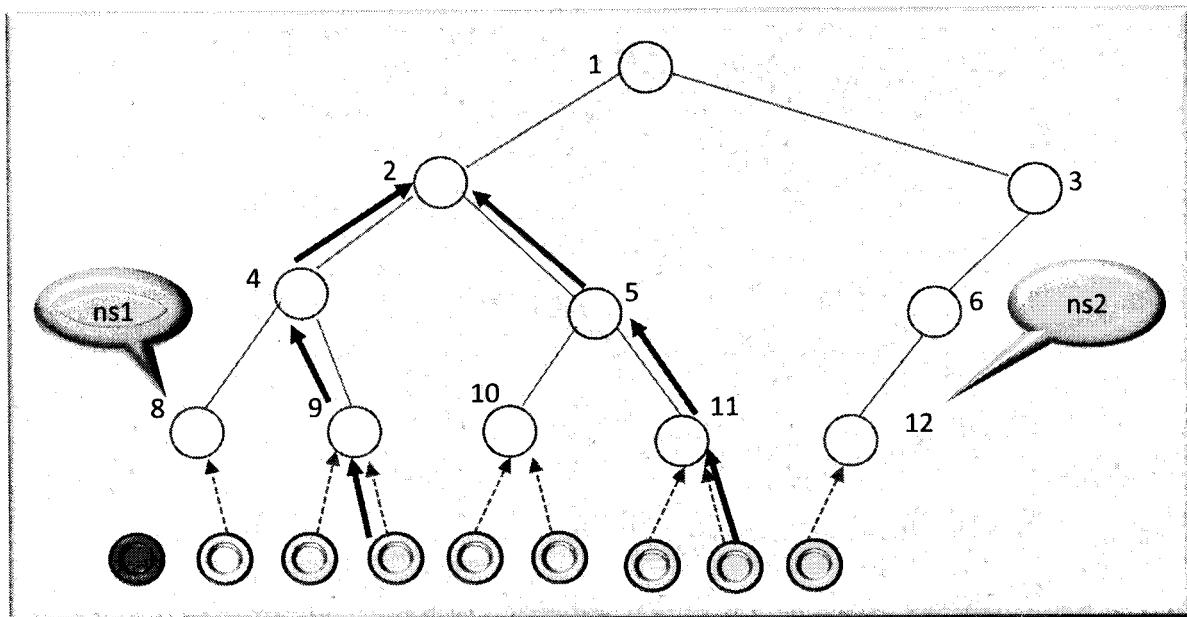
```

```

while(t2>t0){
    if((t2&1)==1 && t2/2!=t0)
        {tmr=dtr_mn(tmr,sm[t2-1]);txr=dtr_mx(txr,sx[t2-1]);}
    t2>>=1;
}

```

Tại nút cha chung: So sánh kết quả hai đường duyệt, xác định cực trị tồn tại trong khoảng [x,



```

void get_ans(int x, int y)
{
int t1,t2,t0,tml,txl,tmr,txr,rm,rx;
t1=x+nh-1;t2=y+nh-1;
while(t1!=t2){t1/=2;t2/=2;}
t0=t1; t1=(x+nh-1)/2;t2=(y+nh-1)/2;
tml=p[x];txl=tml; if((x&1)==1){tml=sm[t1];txl=sx[t1];}
tmr=p[y];txr=tmr; if((y&1)==0){tmr=sm[t2];txr=sx[t2];}
while(t1>t0){
    if((t1&1)==0 && t1/2!=t0)
        {tml=dtr_mn(tml,sm[t1+1]);txl=dtr_mx(txl,sx[t1+1]);}
    t1>>=1;
}
while(t2>t0){
    if((t2&1)==1 && t2/2!=t0)
        {tmr=dtr_mn(tmr,sm[t2-1]);txr=dtr_mx(txr,sx[t2-1]);}
    t2>>=1;
}
if(tml<tmr)rm=tml; else rm=tmr;
if(txl<txr)rx=txr; else rx=txl;
if(y-x == rx-rm) fo<<"YES"<<'\n'; else fo<<"NO"<<'\n';
}

```

y].

Kết quả: Nếu chênh lệch cực trị bằng độ dài khoảng [x, y] thì kết quả là YES.

Độ phức tạp: O(nlogn).

```

#include <fstream>
#include <ctime>
using namespace std;
int n,m,k,a,b,xx,yy,nt,nh,ns2,ns1,h[200002],sm[400002],sx[400002],p[200002];
ifstream fi ("snow_wh.inp");
ofstream fo ("snow_wh.out");

int dtr_mn(int x,int y)
{if(x<y) return x; else return y; }

int dtr_mx(int x,int y)
{if(x<y) return y; else return x; }

void dtr_bt()
{int=(nh+1)/2;
 int u,v;
 for(int i=ns1;i<=ns2;++)
 { u=i*2-nh+1; v=u+1;
   sm[i]=dtr_mn(p[u],p[v]);sx[i]=dtr_mx(p[u],p[v]);
 }
 sm[ns2+1]=sm[ns2];sx[ns2+1]=sx[ns2];
 for(int i=ns1-1;i>=1;--i)
 {u=i*2; v=u+1;sm[i]=dtr_mn(sm[u],sm[v]);sx[i]=dtr_mx(sx[u],sx[v]);}
}

void upd_t(int x,int y)
{int t,t1,t2,u,x0,y0;
 t=p[h[x]];p[h[x]]=p[h[y]];p[h[y]]=t;p[n+1]=p[n];
 t=h[x];h[x]=h[y];h[y]=t;x0=h[x]; y0=h[y];
 t1=(nh+x0-1)/2;t2=(nh+y0-1)/2; if(x0%2==1)u=x0+1; else u=x0-1;
 sm[t1]=dtr_mn(p[u],p[x0]); sx[t1]=dtr_mx(p[u],p[x0]);
 if(y0%2==1)u=y0+1; else u=y0-1;
 sm[t2]=dtr_mn(p[u],p[y0]); sx[t2]=dtr_mx(p[u],p[y0]);
 sm[ns2+1]=sm[ns2];sx[ns2+1]=sx[ns2];
 t1/=2;t2/=2;
 while(t1>=1)
 {sm[t1]=dtr_mn(sm[2*t1],sm[2*t1+1]);
  sx[t1]=dtr_mx(sx[2*t1],sx[2*t1+1]);
  t1/=2;
 }
 while(t2>=1)
 {sm[t2]=dtr_mn(sm[2*t2],sm[2*t2+1]);
  sx[t2]=dtr_mx(sx[2*t2],sx[2*t2+1]);
  t2/=2;
 }
}

void get_ans(int x, int y)
{int t1,t2,t0,tml,txl,tmr,txr,rm,rx;
 t1=x+nh-1;t2=y+nh-1;
 while(t1!=t2){t1/=2;t2/=2;}
 t0=t1; t1=(x+nh-1)/2;t2=(y+nh-1)/2;
 tml=p[x];txl=tml; if((x&1)==1){tml=sm[t1];txl=sx[t1];}
 tmr=p[y];txr=tmr; if((y&1)==0){tmr=sm[t2];txr=sx[t2];}
 while(t1>t0){
  if((t1&1)==0
  t1/2=t0){tml=dtr_mn(tml,sm[t1+1]);txl=dtr_mx(txl,sx[t1+1]);}
  t1/=2;
 }
 while(t2>t0){
  if((t2&1)==1 && t2/2!=t0){tmr=dtr_mn(tmr,sm[t2-1]);txr=dtr_mx(txr,sx[t2-1]);}
  t2/=2;
 }
 if(tml<tmr)rm=tml; else rm=tmr;
}

```

```

if(txl<txr) rx=txr; else rx=txl;
if(y-x == rx-rm) fo<<"YES"<<'\n'; else fo<<"NO"<<'\n';
}

int main()
{clock_t aa=clock();
 fi>>n>>m;
for(int i=1;i<=n;++i)fi>>h[i];
for(int i=1;i<=n;++i)p[h[i]]=i;p[n+1]=p[n];
for(int i=31;i>=0;--i) if((n>>i)&1 ==1){nt=i+1; break;}
nh=1<<nt; ns2=(nh+n)/2; ns1=nh>>1;
dtr_bt();
for(int i=1;i<=m;++i)
{fi>>k>>a>>b;
 if(k==1)upd_t(a,b); else get_ans(a,b);
}
clock_t bb=clock();
fo<<"Time: "<<(double)(bb-aa)/1000<<" sec";
}

```

### **3 – Danh sách mốc nối**

Các dữ liệu có cấu trúc như xâu, mảng, véc tơ, hàng đợi, stack . . . đều là danh sách mốc nối, nhưng ở trong các dữ liệu đó mốc nối được cho dưới dạng không tường minh, thể hiện qua trình tự xuất hiện.

Danh sách mốc nối tường minh trong trường hợp tổng quát được nghiên cứu khá đầy đủ và trình bày chi tiết trong các tài liệu về cấu trúc dữ liệu.

Trong các bài toán thực tế, thông thường chỉ một bộ phận các quan hệ đó trực tiếp tác động lên độ phức tạp của giải thuật. Xác định và thể hiện được các mối quan hệ đó phụ thuộc vào sự linh hoạt và trình độ lập trình của người giải quyết.

Việc kết hợp giữa các cấu trúc dữ liệu chuẩn mà hệ thống lập trình cung cấp với một hoặc một số danh sách mốc nối tường minh thể hiện quan hệ bộ phận do người lập trình thiết kế sẽ tác động rất lớn đến hiệu quả của giải thuật. Ví dụ, nếu chỉ dựa thuần túy trên mảng  $A = (a_1, a_2, \dots, a_n)$  thì việc tìm dãy con dài nhất các phần tử tăng dần sẽ có độ phức tạp  $O(n^2)$ . Việc bổ sung thêm mốc nối tới phần tử cuối nhỏ nhất trong các danh sách con đã làm giảm độ phức tạp xuống còn  $O(n\log n)$ . Đi xa hơn nữa, với hệ thống mốc nối bộ phận xác lập bởi cây vEB (cây Van Emde Boas) độ phức tạp của giải thuật giảm xuống còn  $O(n\log(\log n))$ .

Với mỗi phần tử dữ liệu quan trọng nhất là thông tin về phần tử trước và sau nó. Khái niệm trước, sau phụ thuộc vào tiêu chuẩn phân loại áp dụng trong giải thuật khi giải bài toán. Người lập trình chỉ cần biết vị trí các phần tử này là đủ để giải quyết mọi vấn đề. Như vậy trong trường hợp tổng quát, mỗi phần tử dữ liệu được gắn với một bản ghi mốc nối 2 trường xác định vị trí của các phần tử trước và sau.

Sau khi đã xác định được bản chất của quan hệ và khai báo được dữ liệu cần thiết có hai vấn đề đòi hỏi người lập trình phải cân nhắc và giải quyết:

- ➔ Gán giá trị đầu cho các mốc nối,
- ➔ Cập nhật mốc nối.

Những rắc rối trong việc gán giá trị đầu thường chỉ xảy ra đối với phần tử đầu và cuối danh sách.

## 47. KHÓA DÙNG MỘT LẦN

Tên chương trình: UNIQUEK.???

Độ bảo mật thông tin sẽ cao hơn rất nhiều nếu khóa mã hóa chỉ được sử dụng một lần và không dùng lại ở bất cứ lúc nào khác, bởi vì việc dùng cùng một khóa để mã hóa một số văn bản khác nhau có thể là cho người ta dễ bẻ khóa hơn nhiều.

Có  $m$  văn bản đã mã hóa, văn bản thứ  $i$  được mã hóa bằng khóa có đặc trưng bởi số nguyên  $k_i$  ( $i = 1 \dots m$ ). Cho  $q$  truy vấn, truy vấn thứ  $j$  được xác định bởi 2 số nguyên  $b_j$  và  $e_j$  ( $1 \leq b_j \leq e_j \leq m$ ,  $j = 1 \dots q$ ). Hãy xác định xem khi mã hóa các văn bản từ  $b_j$  đến  $e_j$  có khóa nào bị dùng lại hay không. Nếu không có khóa nào sử dụng quá một lần thì đưa ra thông báo “OK”, trong trường hợp ngược lại – đưa ra số đặc trưng của một khóa nào đó đã sử dụng quá một lần.

**Dữ liệu:** Vào từ file văn bản UNIQUEK.INP, gồm nhiều tests, mỗi test cho trên một nhóm dòng:

- Dòng đầu tiên trong nhóm chứa 2 số nguyên  $m$  và  $q$  ( $1 \leq m \leq 10^6$ ,  $0 \leq q \leq 10^6$ ),
- Dòng thứ  $i$  trong  $m$  dòng tiếp theo chứa số nguyên  $k_i$  ( $0 \leq k_i \leq 2^{30}$ ),
- Dòng thứ  $j$  trong  $q$  dòng tiếp theo chứa 2 số nguyên  $b_j$  và  $e_j$ ,

Sau mỗi test là một dòng trống. Dữ liệu kết thúc bằng dòng chứa 2 số 0.

**Kết quả:** Đưa ra file văn bản UNIQUEK.OUT, mỗi kết quả truy vấn một test đưa ra trên một dòng, chứa thông báo “OK” hoặc số nguyên ứng với khóa dùng lại. Giữa kết quả các tests là một dòng trống.

**Ví dụ:**

UNIQUEK.INP	
10	5
3	
2	
3	
4	
9	
7	
3	
8	
4	
1	
1	3
2	6
4	10
3	7
2	6
5	2
1	
2	
3	
1	
2	
2	4
1	5
0	0

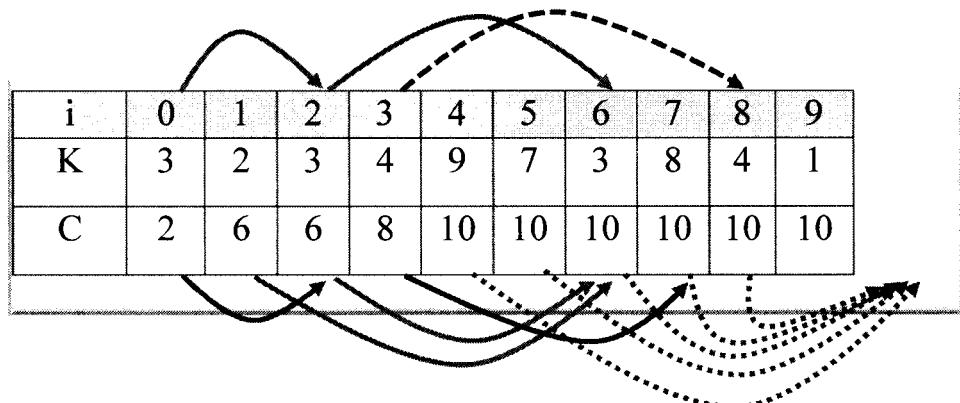
UNIQUEK.OUT	
3	
OK	
4	
3	
OK	
OK	
1	



g47 CERC 11 B

## Giải thuật

Với dãy khóa  $K = (k_0, k_1, \dots, k_{m-1})$  tạo mốc nối  $C = (c_0, c_1, \dots, c_{m-1})$ , trong đó  $c_i$  là chỉ số nhỏ nhất lớn hơn  $i$  để trong khoảng  $[i, c_i]$  tồn tại một cặp giá trị khóa giống nhau,  $c_i = m$  nếu trong khoảng từ  $i$  tới  $m-1$  không có cặp khóa nào giống nhau.

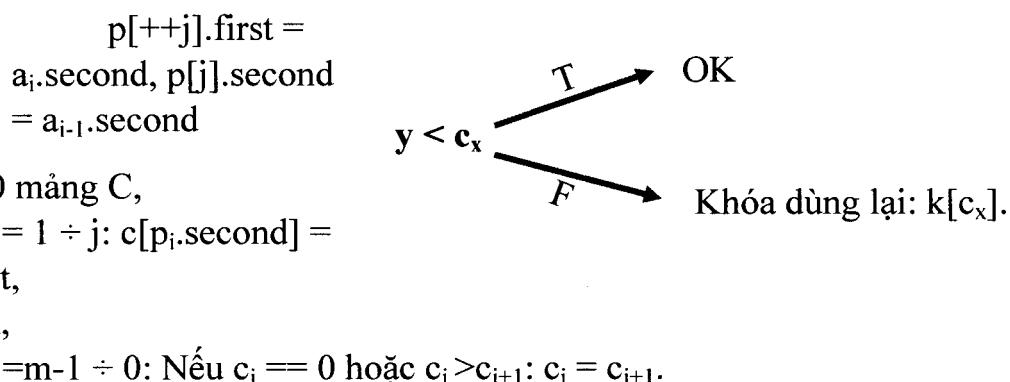


Ví dụ, với  $m = 10$ :

Với khoảng  $[x, y]$  ( $1 \leq x \leq y < m$ ):

Xác định C:

- ↳ Tạo mảng A =  $(a_0, a_1, \dots, a_{m-1})$ , trong đó  $a_i = (k_i, i)$ ,
- ↳ Sắp xếp A theo thứ tự tăng dần,
- ↳ Với  $j = 0$ , nếu  $a_j.\text{first} == a_{j-1}.\text{first}$ : ( $i = 1 \div m-1$ )



Xử lý một truy vấn:

- ↳ Nhập cặp giá trị b và e,
- ↳ Giảm 1 b và e (vì chỉ số đánh trong chương trình từ 0), // --b; --e;
- ↳ Từ vị trí b: phần tử lặp lại gần nhất là ở vị trí  $c_b$ ,  $\rightarrow$  nếu  $e < c_b$  – không có khóa trùng, trong trường hợp ngược lại: tồn tại khóa trùng, ví dụ khóa  $d_{c[b]}$ ,

⌚ Thời gian xử lý một truy vấn:  $O(1)$ .

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

```
#include <fstream>
#include <ctime>
using namespace std;
typedef pair<int,int> p2;
p2 a[1000000],p[1000000];
int n,q,np,b,e,j,c[1000001],d[1000001];
string s;
ifstream fi ("UniqueK.inp");
ofstream fo ("UniqueK.out");
int main()
{ clock_t aa=clock();
    fi>>n>>q;
    while(n>0)
    {
        memset(p,0,sizeof(p)); np=0;
        for (int i=0;i<n;++i){fi>>d[i];a[i].first=d[i];a[i].second=i;}
        sort(a,a+n);
        for(int i=1;i<n;++i)
            if(a[i].first==a[i-1].first)
                {p[++np].first=a[i].second; p[np].second=a[i-1].second;}
        sort(p+1,p+np+1,greater<p2>());
        memset(c,0,sizeof(c));
        for(int i=np;i>=1;--i)
            {j=p[i].second;c[j]=p[i].first;}
        c[n]=n;
        for(int i=n-1;i>=0;--i)
            if(c[i]==0 || c[i]>c[i+1]) c[i]=c[i+1];
        for(int i=1;i<=q;++i)
            {fi>>b>>e;--b;--e;
             if(e<c[b]) fo<<"OK"<<endl; else fo<<d[c[b]]<<endl;
            }
        getline(fi,s,'\
'); fo<<endl;
        fi>>n>>q;
    }
    clock_t bb=clock();
    fo<<endl<<"Time: "<<double(bb-aa)/1000 << " seconds."<< endl;
}
```

## VL10. THÁM HIỂM

Tên chương trình: EXPEDITION.???

Một nhóm cán bộ khoa học được cử đi thám hiểm trên con tàu vũ trụ thế hệ mới. Họ có nhiệm vụ khảo sát  $n$  hành tinh từ Trái đất đến hành tinh Triumf. Các hành tinh được đánh số theo trình tự khảo sát, từ 1 đến  $n$ , Trái đất là 1, còn Triumf – là  $n$ .

Để vượt qua khoảng không tàu có thể dùng nhiên liệu bất kỳ có ở trạm tiếp nhiên liệu trên mỗi hành tinh. Trước khi xuất phát, tàu ở Trái đất và bình nhiên liệu rỗng. Ở hành tinh thứ  $i$  chỉ có nhiên liệu loại  $a_i$ . Khi tới hành tinh thứ  $i$  người ta có thể rút toàn bộ nhiên liệu cũ ra, bơm nhiên liệu loại  $a_i$  vào.

Ở mỗi hành tinh trạm tiếp nhiên liệu hoạt động như sau: bơm vào bình chứa một khối lượng nhiên liệu vừa đủ để đi tới hành tinh tiếp theo có cùng loại nhiên liệu. Nếu như sau đó không có nơi nào có cùng loại nhiên liệu thì không thể nạp nhiên liệu ở đây. Nói một cách khác, nếu nạp được nhiên liệu ở hành tinh  $i$  thì có đủ nhiên liệu để bay đến các hành tinh từ  $(i+1)$  cho đến tận hành tinh  $j$ , trong đó  $j$  là giá trị nhỏ nhất thỏa mãn các điều kiện  $j > i$  và  $a_i = a_j$ .

Hãy xác định số lần nạp nhiên liệu tối thiểu cần thực hiện và chỉ ra những nơi cần nạp. Nếu không có phương án nạp nhiên liệu thì đưa ra số 0. Nếu tồn tại nhiều cách nạp thỏa mãn yêu cầu thì đưa ra một cách theo tùy chọn.

**Dữ liệu:** Vào từ file văn bản EXPEDITION.INP:

- ◆ Dòng đầu tiên chứa số nguyên  $n$  ( $2 \leq n \leq 3 \times 10^5$ ),
- ◆ Dòng thứ 2 chứa  $n$  số nguyên  $a_1, a_2, \dots, a_n$  ( $1 \leq a_i \leq 300\,000$ ,  $i = 1 \div n$ ).

**Kết quả:** Đưa ra file văn bản EXPEDITION.OUT:

- ◆ Dòng đầu tiên chứa số nguyên  $k$  – số lần nạp nhiên liệu tối thiểu cần thực hiện,
- ◆ Nếu  $k > 0$  thì dòng thứ 2 chứa  $k$  số nguyên xác định những nơi cần nạp nhiên liệu, số liệu đưa ra theo trình tự nạp.

**Ví dụ:**

EXPEDITION.INP
7
1 3 2 1 3 2 3

EXPEDITION.OUT
3
1 3 5



### **Giải thuật:**

Đây là bài toán quy hoạch động đòi hỏi tìm giá trị cực tiểu của hàm mục tiêu (*số lần nạp nhiên liệu ít nhất*) và phương án tối ưu (*những nơi nạp*), vì vậy cần:

- ◆ Duyệt từ cuối về đầu,
- ◆ Cần tổ chức dữ liệu để lưu phương án tối ưu đạt được trong quá trình duyệt.

Cấu trúc Heap và bảng mốc nối phần tử cùng giá trị trước đó cho phép cập nhật tại chỗ (*update in place*) phương án tối ưu. Điều này làm giảm độ phức tạp của giải thuật và cho phép tiết kiệm bộ nhớ.

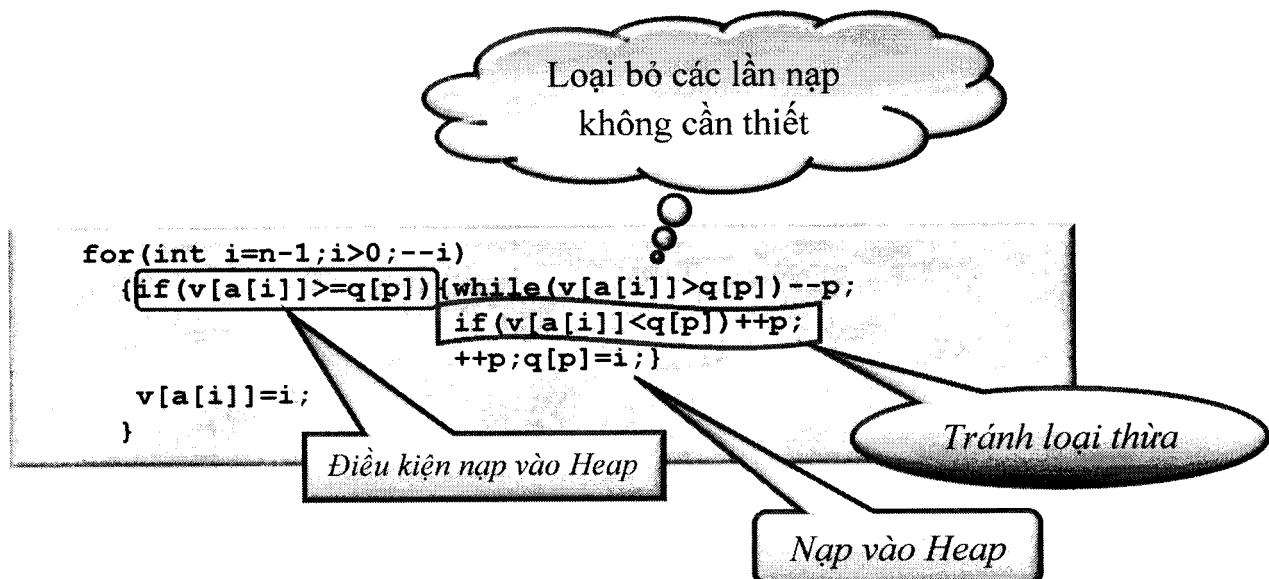
Dữ liệu:

- ◆ Mảng A: Chứa  $a_i$ ,  $i = 1 \div n$ ,
- ◆ Mảng V: Mốc nối tới phần tử cùng giá trị trước đó,  $v[a[i]] = i$ ,
- ◆ Mảng Q: Tổ chức Heap Max, p – xác định vị trí cuối của Heap,

Khởi tạo:  $v[a[n]] = n$ ;  $p = 0$ ;  $q_p = n$ ; – hàng rào.

Điều kiện cập nhật: từ  $i$  với giá trị  $a_i$  phải đi tới được hoặc vượt qua nơi nạp cuối cùng đã xác định, tức là  $v[a[i]] \geq q_p$ .

Cập nhật: Loại bỏ những lần nạp đã xác định bây giờ trở nên không cần thiết.



Dẫn xuất kết quả:

- ◆ Nếu lần nạp cuối cùng không phải ở vị trí 1 – bài toán vô nghiệm (đưa ra 0),
- ◆ Trong trường hợp có nghiệm: đưa ra p (số lần nạp nhiên liệu) và nơi nạp (các giá trị  $q_i$ ,  $i = p \div 1$ ).

Chương trình giải (*Dộ phức tạp  $O(n)$* )

```
#include <iostream>
#include <ctime>
```

```

using namespace std;
int n,p,a[300001],v[300001]={0},q[300001];
ifstream fi ("Expedition.inp");
ofstream fo ("Expedition.out");

int main()
{clock_t aa=clock();
 fi>>n;
 for(int i=1;i<=n;++i)    fi>>a[i];
 v[a[n]]=n;
 p=0; q[0]=n;
 for(int i=n-1;i>0;--i)
 {if(v[a[i]]>=q[p]) {while(v[a[i]]>q[p])--p;
 if(v[a[i]]<q[p])++p; ++p;q[p]=i;}
 v[a[i]]=i;
 }
 if(q[p]!=1)fo<<0;
 else
 {fo<<p<<'\n';
 for(int i=p;i>0;--i)fo<<q[i]<<" ";
 }
 clock_t bb=clock();
 fo<<'\n'<<"Time: "<<(double)(bb-aa)/1000<<" sec";
}

```

*Input:*

16  
1 4 3 2 3 1 4 4 2 4 1 4 3 2 2 3

*Output:*

3  
1 5 13

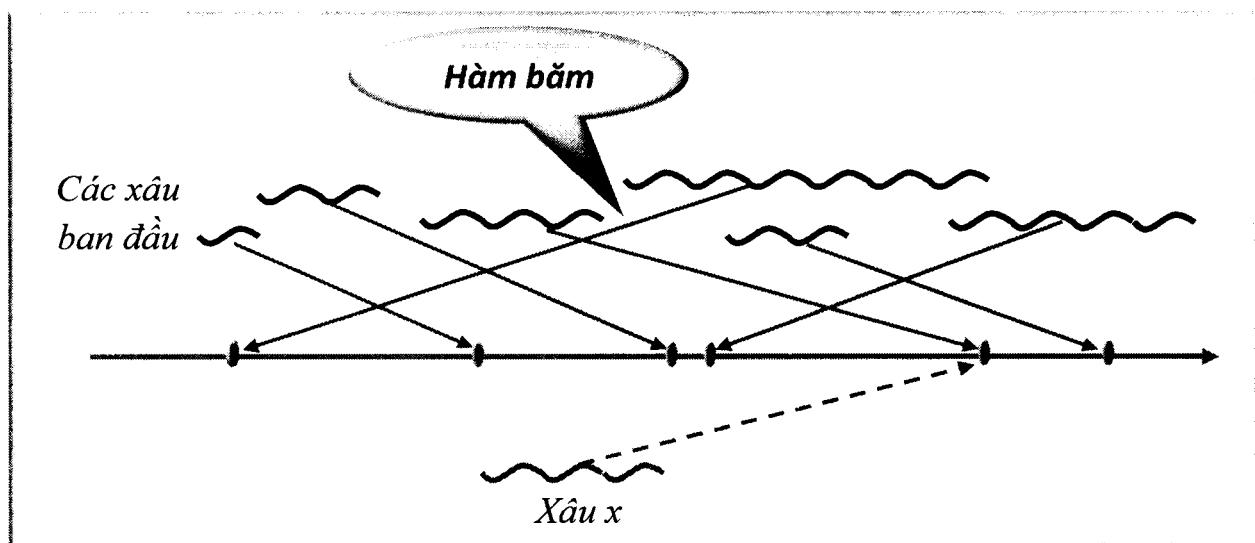
#### 4 – Kỹ thuật hàm băm

Băm (Hashing) là kỹ thuật dẫn xuất việc lưu trữ một tập hợp lớn về một tập nhỏ hơn. Hàm biến đổi khóa của các đối tượng ban đầu về khóa của các đối tượng trong tập nhỏ hơn được gọi là *hàm băm*.

Việc dẫn xuất các đối tượng từ tập lớn sang đối tượng tương đương ở tập nhỏ hơn sẽ làm giảm bớt các phép kiểm tra, so sánh phải thực hiện trong quá trình tìm kiếm, nhận dạng.

Ví dụ, cho tập gồm  $n$  xâu, xâu thứ  $i$  có độ dài  $m_i$  ( $i = 1 \div n$ ). Hãy xác định xem một xâu  $x$  có mặt trong một số tập con của tập đã cho hay không. Một trong các phương pháp khá hiệu quả giải bài toán này là tổ chức một cây tiền tố, trong đó mỗi nút là một ký tự cùng với dấu hiệu cho biết đây có phải là ký tự cuối cùng của một xâu nào đó trong tập ban đầu hay không. Việc nhận dạng sự có mặt của xâu  $x$  sẽ được đưa về việc duyệt cây theo các ký tự của  $x$ . Nếu tồn tại một nhánh cây thuộc một tập con nào đó chứa các ký tự của  $x$  (theo trình tự xuất hiện trong  $x$ ) và nút chứa ký tự cuối cùng của  $x$  có dấu hiệu kết thúc từ thì có nghĩa là  $x$  có mặt trong tập con đang xét, trong trường hợp ngược lại – câu trả lời là phủ định. Gọi độ dài xâu  $x$  là  $lx$ . Công việc kiểm tra sự tồn tại của  $x$  trong tập đã cho có độ phức tạp  $O(lx)$ . Với  $n$ ,  $lx$  và các  $m_i$  đủ lớn, số tập con khá nhiều thì giải thuật nêu trên là chưa đủ hiệu quả. Giải thuật này đặc biệt không thích hợp khi phải liên tiếp kiểm tra với nhiều xâu  $x$  khác nhau. Ngoài ra việc tổ chức cây tiền tố khác phức tạp trong lập trình, tốn bộ nhớ và mất nhiều thời gian.

Với kỹ thuật hàm băm, ta có thể ánh xạ tập xâu ban đầu sang tập số nguyên, ví dụ trong phạm vi từ 1 đến  $10^7$ . Áp dụng ánh xạ tương tự với  $x$ , việc kiểm tra tồn tại có độ phức tạp không quá  $O(\log n)$ . Với một số các tổ chức ánh xạ, độ phức tạp



có thể đạt đến mức  $O(1)$ !.

Nếu hai hay nhiều đối tượng ban đầu khác nhau được hàm băm ánh xạ vào cùng một giá trị thì ta có trường hợp *va chạm*.

Hàm băm tốt là hàm ánh xạ cực tiểu hóa số lượng va chạm và kết quả ánh xạ phân bố đều trên tập kết quả. Hàm băm lý tưởng là hàm ánh xạ không có va chạm.

Có nhiều phương pháp xử lý va chạm, ví dụ tạo danh sách động lưu trữ thông tin về các đối tượng bị va chạm.

Một trong những cách hiệu quả phát hiện và khắc phục va chạm là áp dụng song song hai hàm băm. Xác xuất đồng thời xảy ra va chạm ở cùng một đối tượng với cả hai hàm băm sẽ cực nhỏ, gần như bằng 0 đối với các bài toán thực tế.

Với những bài toán olympic tin học và với cách ánh xạ sẽ xét dưới đây chỉ cần sử dụng một phép ánh xạ. Với  $h$  nguyên đủ lớn, ví dụ  $h = 10^5 + 3$  hoặc  $h = 2^{37}$  việc ánh xạ một xâu hoặc dãy số sang số nguyên được thực hiện như sau:

*Ánh xạ xâu*: xét  $s$  – xâu ký tự độ dài  $m$  cần tạo mảng giá trị  $b_0, b_1, \dots, b_{m-1}$  kiểu int hoặc int64\_t:

- ❖  $b_0 = s_0$ ,
- ❖  $b_1 = b_0 \times h + s_1 = s_0 \times h + s_1$ ,
- ❖  $b_2 = b_1 \times h + s_2 = s_0 \times h^2 + s_1 \times h + s_2$ ,
- ❖ .....
- ❖  $b_i = b_{i-1} \times h + s_i = s_0 \times h^i + s_1 \times h^{i-1} + \dots + s_{i-1} \times h + s_i$ ,
- ❖ .....
- ❖  $b_{m-1} = b_{m-2} \times h + s_{m-1} = s_0 \times h^{m-1} + s_1 \times h^{m-2} + \dots + s_{m-2} \times h + s_{m-1}$ .

Dãy số này cho phép so sánh 2 xâu con các ký tự liên tiếp độ dài bất kỳ với độ phức tạp  $O(1)$ .

Lưu ý rằng hệ thống lập trình C/C++ ngầm định ngắt việc bẫy và xử lý sự kiện tràn ô (*overflow*). Với các hệ thống lập trình có ngầm định việc bẫy và xử lý sự kiện tràn ô ta phải đặt chỉ thị cho chương trình dịch bỏ ngầm định này hoặc phải lấy số dư của phép chia  $b_i$  cho số nguyên đủ lớn  $md$ , như vậy kết quả ánh xạ sẽ nằm trong khoảng  $[0, md-1]$ .

*Ánh xạ dãy số*: Xét dãy số  $a_0, a_1, \dots, a_{m-1}$ . Công thức ánh xạ tương tự trường hợp xâu nếu thay  $s_i$  bằng  $a_i$ ,  $i = 0 \div m-1$ .

## VM47. MẬT KHẨU

Tên chương trình: PAROLE.???

Ngân hàng GreenBank dùng loại mật khẩu sử dụng một lần cho mọi truy nhập tới các dịch vụ của ngân hàng.

Khi có yêu cầu truy nhập ngân hàng sẽ được cung cấp một từ khóa. Người truy nhập chỉ phải nhập vào mật khẩu là một xâu ký tự palindrome độ dài ngắn nhất có chứa từ khóa như một xâu con các ký tự liên tiếp nhau.

Với từ khóa đã cho hãy xác định mật khẩu cần nhập vào. Nếu tồn tại nhiều xâu khác nhau cùng đáp ứng yêu cầu là mật khẩu thì đưa ra xâu bất kỳ trong số đó.

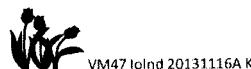
**Dữ liệu:** Vào từ file văn bản PAROLE.INP gồm một dòng chứa từ khóa có độ dài không vượt quá  $3 \times 10^5$  và chỉ bao gồm các ký tự la tinh thường.

**Kết quả:** Đưa ra file văn bản PAROLE.OUT mật khẩu tìm được.

**Ví dụ:**

PAROLE.INP
ab
a

PAROLE.OUT
aba
a



## *Giai thuật*

Sử dụng hàm băm (Hashing Func.): với h nguyên dương và đủ lớn:

$$p_i = \left( \sum_{j=0}^i h^j \right) \bmod d,$$

$$a_0 = s_0,$$

$$a_i = (a_{i-1} \times h + s_i) \bmod d, i = 1 \div n-1,$$

$$b_{n-1} = s_{n-1},$$

$$b_i = (b_{i+1} \times h + s_i) \bmod d, i = n-2 \div 0.$$

$$b_{n-1} = s_{n-1},$$

$$b_{n-2} = b_{n-1} \times h + s_{n-2},$$

.....

$$b_{x+1} = b_{x+2} \times h + s_{x+1} \equiv y,$$

$$b_x = b_{x+1} \times h + s_x = y \times h + s_x,$$

$$b_{x-1} = b_x \times h + s_{x-1} = y \times h^2 + s_x \times h + s_{x-1},$$

$$b_{x-2} = b_{x-1} \times h + s_{x-2} = y \times h^3 + s_x \times h^2 + s_{x-1} \times h + s_{x-2},$$

.....

$$b_0 = b_1 \times h + s_0 = y \times h^{x+1} + s_x h^x + s_{x-1} \times h^{x-1} + \dots + s_2 \times h^2 s_1 \times h^1 + s_0,$$

$$= y \times p_{x+1} + s_x h^x + s_{x-1} \times h^{x-1} + \dots + s_2 \times h^2 s_1 \times h^1 + s_0,$$

$$a_x = s_0 h^x + s_1 \times h^{x-1} + \dots + s_{x-2} \times h^2 s_{x-1} \times h^1 + s_x,$$

```

cc=s[n-1];ixb=n-1;
for(int i=0;i<n;+i)
    if(cc==s[i]) {gethr(i);
        if(t1==t2){ixb=i;break;}}
```

```

void gethr(int x)
{ t1=b[x]; t2 = a[n-1]-a[x-1]*p[n-x];}
```

Xâu S:



Palindrome độ dài m



Palindrome độ dài k

```

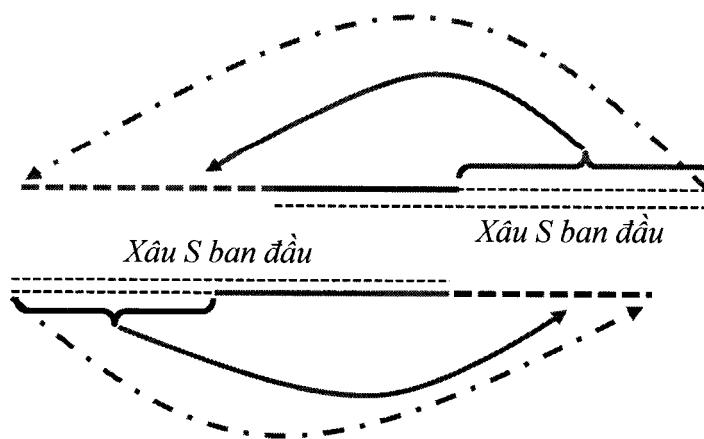
cc=s[0];ixa=0;
for(int i=n-1;i>0;--i)
    if(cc==s[i])
        {geths(i); if(t1==t2){ixa=i;break;}}
```

```

void geths(int x)
{ t1=a[x]; t2 = b[0]-b[x+1]*p[x+1];}
```

Xác định kết quả:



Kéo dài xâu ở đầu có palindrome độ dài lớn hơn.

```

m=ixa+1;q=0; if(m<n-ixb){m=n-ixb; q=1;}
if(q==0){for(int i=n-1;i>ixa;--i) fo<<s[i]; fo<<s;}
else {fo<<s; for(int i=ixb-1;i>=0;--i) fo<<s[i];}
    
```

Độ phức tạp:  $O(n)$ .

```

#include <fstream>
#include <string>
using namespace std;
string s;
char cc;
int n,ixa,ixb,tg1,tg2,q,m;
int64_t a[3000100],b[3000100],p[3000100],d[3000100],ph=500000,t1,t2;
ifstream fi ("parole.inp");
ofstream fo ("parole.out");

void geths(int x)
{ t1=a[x]; t2 = b[0]-b[x+1]*p[x+1]; }

void gethr(int x)
{ t1=b[x]; t2 = a[n-1]-a[x-1]*p[n-x]; }

int main()
{fi>>s; n=s.size();
p[0]=1;
for(int i=1;i<n+100;++i) p[i]=p[i-1]*ph;
a[0]=s[0];
for(int i=1; i<n;++i) a[i]=a[i-1]*ph+s[i];
b[n-1]=s[n-1];
for(int i=n-2;i>=0;--i) b[i]=b[i+1]*ph+s[i];
cc=s[0];ixa=0;
for(int i=n-1;i>0;--i)
    if(cc==s[i]){geths(i); if(t1==t2){ixa=i;break;}}
cc=s[n-1];ixb=n-1;
for(int i =0;i<n;++)
    if(cc==s[i]){gethr(i);if(t1==t2){ixb=i;break;}}
m=ixa+1;q=0; if(m<n-ixb){m=n-ixb; q=1;}
if(q==0){for(int i=n-1;i>ixa;--i) fo<<s[i]; fo<<s;}
else {fo<<s; for(int i=ixb-1;i>=0;--i) fo<<s[i];}
}
    
```

## VN29. BẮT ĐẦU CỦA KẾT THÚC

Tên chương trình: BEGINEND.???

John Connor và Marcus Rice đã thâm nhập được vào Khu liên hợp lắp ráp Kẻ Hủy diệt T-800. Họ chuẩn bị cho nổ tung tổ hợp này và đặt dấu chấm hết cho chuỗi các trận chiến vì tương lai của loài người. Bỗng nhiên hai người phát hiện thấy một điều bất thường: các hòm nhiên nguyên liệu chế tạo T-800 mà họ định cho nổ tung đã bị sắp đặt lại. Trên mỗi hòm trong số  $n$  hòm ở đây đều có sơn một số nguyên. Chính vì vậy hai người mới nhận ra ngay sự thay đổi trình tự. Để đề phòng mọi bất trắc có thể xảy ra Connor và Rice quyết định khôi phục lại trình tự cũ của dãy hòm rồi mới tiến hành thiêu hủy.

Việc bố trí lại các hòm dĩ nhiên là do các T-800 thực hiện. Connor biết rõ cách làm của các rô bốt này:

- ◆ Đảo ngược vị trí của dãy  $k$  hòm bắt đầu từ hòm thứ nhất tính từ trái,
- ◆ Đảo ngược vị trí của dãy  $k$  hòm bắt đầu từ hòm thứ hai tính từ trái,
- ◆ .....
- ◆ Đảo ngược vị trí của dãy  $k$  hòm kết thúc bởi hòm cuối cùng ở bên phải.

Ví dụ, với  $k = 3$  và trình tự ban đầu các hòm là [1, 2, 3, 1, 2], sau khi sắp xếp lại theo kiểu trên trình tự các hòm sẽ là [3, 1, 2, 2, 1].

Chỉ có điều, cả hai đều không biết được giá trị  $k$  được sử dụng ở đây là bao nhiêu.

Cho biết trình tự ban đầu và trình tự hiện tại của dãy hòm. Hãy xác định có bao nhiêu giá trị  $k$  khác nhau có thể đã được áp dụng khi bố trí lại và chỉ rõ các giá trị đó.

**Dữ liệu:** Vào từ file văn bản BEGINEND.INP:

- ◆ Dòng đầu tiên chứa số nguyên  $n$  ( $1 \leq n \leq 10^5$ ),
- ◆ Dòng thứ 2 và dòng thứ 3: mỗi dòng chứa  $n$  số nguyên xác định trình tự ban đầu và trình tự hiện tại của dãy hòm, mỗi số có giá trị trong phạm vi  $[1..10^5]$ .

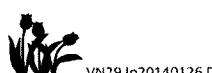
**Kết quả:** Đưa ra file văn bản BEGINEND.OUT:

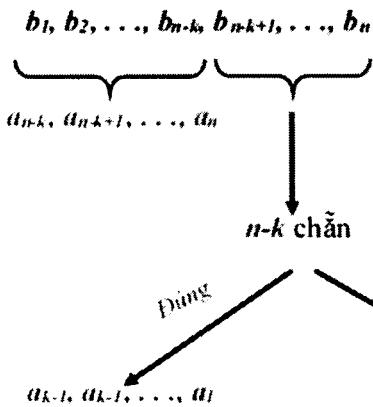
- ◆ Dòng đầu tiên chứa số nguyên  $m$  – số giá trị  $k$  khác nhau có thể được sử dụng,
- ◆ Mỗi dòng trong  $m$  dòng sau chứa một số nguyên dương – các giá trị có thể của  $k$ .

**Ví dụ:**

BEGINEND.INP
5
1 2 3 1 2
3 1 2 2 1

BEGINEND.OUT
1
3





$$P = (p_0, p_1, p_2, \dots, p_n), p_i = h^i, i = 0 \div n.$$

$$C = (c_0, c_1, c_2, \dots, c_n), c_0 = 0, c_i = \sum_{j=1}^i a_j * h^{i-j}, i = 1 \div n.$$

$$D = (d_0, d_1, \dots, d_n), d_0 = 0, d_i = \sum_{j=1}^i a_j * h^{i-j}, i = 1 \div n.$$

$$E = (e_0, e_1, \dots, e_n), e_{n+1} = 0, e_i = \sum_{j=n}^i a_j * h^{i-j}, i = 1 \div n.$$

### Giai thuật

Với k cố định: kết quả biến đổi:

Giải thuật nhận dạng: Dùng hàm băm. *Điều kiện cần*:  $b_1 = a_1$ .

$$c_1 = a_1,$$

$$c_2 = a_1 h + a_2,$$

$$c_3 = a_1 * h^2 + a_2 * h + a_3,$$

.....

$$c_{k-1} = a_1 * h^{k-2} + a_2 * h^{k-3} + \dots + a_{k-2} * h + a_{k-1}, \quad (\equiv uc)$$

.....

$$c_n = a_1 * h^{n-1} + a_2 * h^{n-2} + \dots + a_{k-1} * h^{n-k+1} + a_k * h^{k-1} + \dots + a_{n-1} * h + a_n.$$

$$d_1 = b_1,$$

$$c_n - uc * h^{n-k+1}$$

$$d_2 = b_1 h + b_2,$$

$$d_3 = b_1 * h^2 + b_2 * h + b_3,$$

.....

$$d_{n-k+1} = b_1 * h^{n-k} + b_2 * h^{n-k-1} + \dots + b_{n-k} * h + b_{n-k+1}, \quad (\equiv x)$$

.....

$$d_n = b_1 * h^{n-1} + \dots + b_{n-k} * h^{k-1} + b_{n-k+2} * h^{k-2} + \dots + b_n.$$

$$d_n - x * h^{k-1}$$

Bằng nhau nếu  $n-k$  lẻ

$$e_k = a_k + a_{k+1} * h + \dots + a_n * h^{n-k},$$

Bằng nhau nếu  $n-k$  chẵn

$$e_1 = a_1 + a_2 * h + \dots + a_{k-1} * h^{k-2} + a_k * h^{k-1} + \dots + a_n * h^{n-1}.$$

$$e_1 - e_k * h^{k-1}$$

```

#include <fstream>
#include <vector>
#include <ctime>
using namespace std;
ifstream fi ("beginend.inp");
ofstream fo ("beginend.out");
int64_t a[100001],b[100001],c[100001],d[100001],e[100001],p[100001];
int64_t x,y,uc,ue,vc,ve,h,t;
vector<int> r;
int n,k,good;

void chk_r(int m)
{x=d[n-m+1]; y=d[n]-x*p[m-1];
uc=c[m-1]; vc=c[n]-uc*p[n-m+1];
ue=e[1]-e[m]*p[m-1];
if(((n-m+1)&1)==0) good=(vc==x)&&(uc==y);
else good=(vc==x)&&(y==ue);
}
int main()
{ //clock_t bb=clock();
h=1000003;
fi>>n;
p[0]=1; for(int i=1;i<=n;++i)p[i]=p[i-1]*h;
for(int i=1;i<=n;++i)fi>>a[i];
d[0]=0; for(int i=1;i<=n;++i){fi>>b[i]; d[i]=d[i-1]*h+b[i];}
c[0]=0; for(int i=1;i<=n;++i)c[i]=c[i-1]*h+a[i];
e[n+1]=0; for(int i=n;i>0;--i)e[i]=e[i+1]*h+a[i];
k=0;
for(int i=1;i<=n;++i)
{if(a[i]==b[1]) {chk_r(i);
if(good){++k;r.push_bbck(i);}
}
}
fo<<k<<'\n';
for(size_t i=0;i<r.size();++i)fo<<r[i]<<((i + 1 == r.size()) ? "\n" :
" ");
//clock_t bb=clock();
//fo<<"\nTime: "<<(double)(bb-bb)/1000<<" sec.";
}

```

### VP13. BA NGƯỜI BẠN

Tên chương trình: FRIENDS.???

Ba người bạn cùng ngồi xem truyền hình trực tiếp một trận bóng đá. Ở giờ nghỉ giữa 2 hiệp, không có việc gì làm bạn thứ nhất viết một xâu **s** chỉ chứa các chữ cái la tinh in hoa. Bạn thứ 2 viết tiếp vào cuối xâu này những gì bạn thứ nhất đã viết, tạo thành xâu **t** chứa 2 xâu **s** liên tiếp nhau. Bạn thứ ba viết chèn thêm một ký tự vào đầu hoặc cuối hay một nơi nào đó trong **t** và nhận được xâu **u** độ dài **n**.

Tù xâu **u** hãy xác định xâu **s** ban đầu. Nếu không tồn tại xâu **s** (vì bạn thứ hai có thể nhầm lẫn khi tạo **t**) thì đưa ra thông báo "**NOT POSSIBLE**", nếu xâu **s** tìm được nhưng không đơn trị thì đưa ra thông báo "**NOT UNIQUE**".

**Dữ liệu:** Vào từ file văn bản FRIENDS.INP:

- ◆ Dòng đầu tiên chứa số nguyên **n** ( $2 \leq n \leq 2 \times 10^6 + 1$ ),
- ◆ Dòng thứ 2 chứa xâu **u**.

**Kết quả:** Đưa ra file văn bản FRIENDS.OUT xâu **s** hoặc thông báo thích hợp.

**Ví dụ:**

FRIENDS.INP	FRIENDS.OUT
7	
<b>ABXCABC</b>	<b>ABC</b>

FRIENDS.INP	FRIENDS.OUT
6	
<b>ABCDEF</b>	<b>NOT POSSIBLE</b>

FRIENDS.INP	FRIENDS.OUT
9	
<b>ABABABABA</b>	<b>NOT UNIQUE</b>



## 5 – Cây Van Emde Boas

Cây Van Emde Boas (*vEB Tree*) do Van Emde Boas đề xuất năm 1977 là một loại cấu trúc dữ liệu dùng để lưu trữ các số nguyên trong phạm vi  $[0, m = 2^k]$ , tức là các số có không quá  $k$  bit trong dạng biểu diễn nhị phân.

Tại sao cần tìm hiểu một cấu trúc mới chỉ cho phép lưu trữ các dữ liệu nguyên trong khi đã có hàng loạt các loại cây tìm kiếm nhị phân cân bằng đảm bảo khả năng thực hiện các phép bổ sung, loại bỏ, tìm kiếm, . . . với độ phức tạp  $O(\log n)$ , trong đó  $n$  - số lượng các phần tử của cây?

Nét độc đáo của cấu trúc dữ liệu này là đảm bảo mọi phép xử lý với độ phức tạp  $O(\log(\log m))$  không phụ thuộc vào số lượng phần tử trong cây.

Các phép xử lý thường gặp là:

- ➔ insert( $x$ ) - Bổ sung phần tử  $x$  vào cây,
- ➔ remove( $x$ ) – Loại bỏ phần tử khỏi cây,
- ➔ getmin(), getmax() – Tìm phần tử min và phần tử max trong cây,
- ➔ find( $x$ ) – Tìm phần tử  $x$ ,
- ➔ findnext( $x$ ) – Tìm phần tử tiếp sau  $x$  lưu trong cây,
- ➔ findprevious( $x$ ) – Tìm phần tử trước  $x$  lưu trong cây.

Các kết quả tìm kiếm không đơn thuần cho biết phần tử đó có trong cây hay không mà còn cung cấp trực tiếp thông tin về phần tử tìm được.

Xét cây vEB-k quản lý các số trong khoảng  $[0, 2^k]$ , trong đó  $k$  cũng là một lũy thừa của 2. Cây vEB-1 lưu trữ thông tin về 2 số 0 và 1.

Cây vEB-k chứa các thông tin:

- ➔  $2^{k/2}$  cây vEB- $k/2$ ,
- ➔ Cây vEB- $k/2$  bỗ trợ,
- ➔ Giá trị min và max chứa trong cây này (nếu nó không rỗng).

Giả thiết ta phải nạp số nguyên  $x$  có  $k$  bit trong dạng biểu diễn nhị vào vEB-k, ví

$$x = 93_{10} = \underbrace{0101}_{\text{high}(x)} \underbrace{1101_2}_{\text{low}(x)}$$

dụ,  $x = 93$ .

// Còn tiếp . . .

```

class vEBTraverser {
public:
    uint64_t root() {
        d = 0;
        cix = 1;

        return 1;
    }
    uint64_t left() {
        d++;
        cix <= 1;
        return vEBIndex();
    }
    uint64_t right() {
        d++;
        cix = (cix << 1) + 1;
        return vEBIndex();
    }
private:
    uint64_t cix;
    uint64_t d;

    uint64_t vEBIndex() {
        // Start with largest sub-tree, work down to smallest.
        uint64_t ix = 1;
        uint32_t new_d = d;
        for (char b = 4; b >= 0; --b) {
            const uint64_t b_val = 1L << b;
            if (d & b_val) {
                // Determine sub triangle and add start offset to
index.
                const uint64_t masked_d =      d & (b_val - 1);
                const uint64_t new_node_size = (1L << b_val) - 1;
                uint64_t subtri_ix =
                                (cix >> masked_d) &
new_node_size;
                ix += new_node_size * (1L + subtri_ix);
            }
        }
        return ix;
    }
};

```

## **6 – Đồ thị cây RB(Red - Black)**

Cây *Red - Black* là một dạng cây tìm kiếm nhị phân tự cân bằng, một cấu trúc dữ liệu được sử dụng trong khoa học máy tính. Cấu trúc ban đầu của nó được đưa ra vào năm 1972 bởi Rudolf Bayer. Ông gọi chúng là các "B-cây cân bằng" còn tên hiện nay được đưa ra từ 1978 bởi Leo J. Guibas và Robert Sedgewick. Nó là cấu trúc phức tạp nhưng cho kết quả tốt về thời gian trong trường hợp xấu nhất.

Trong cây *Red – Black*, các nút không có con được gọi là lá (*leaf node*) và được gán giả là null, nghĩa là chúng không chứa bất kỳ dữ liệu nào. Tại đây nó sẽ chịu trách nhiệm thông báo rằng cây đã kết thúc. Việc thêm các nút này làm cho tất cả các nút trong của cây đều chứa dữ liệu và có hai con, hay khác đi cây đỏ đen cùng với các lá null là cây nhị phân đầy đủ.

Cây *Red - Black* cũng giống như các cây tìm kiếm nhị phân khác đều thỏa mãn tính chất: mỗi nút được gán một giá trị sao cho giá trị trên mỗi nút nhỏ hơn hoặc bằng tất cả các giá trị trên các nút thuộc cây con phải và lớn hơn các giá trị nằm trên cây con trái. Điều đó làm cho quá trình tìm kiếm nhanh hơn.

### **6.1 - Tính chất của cây đỏ đen**

Tính chất cây đỏ đen bao gồm:

1. Nút gốc là đen.
2. Mọi nút phải là đỏ hoặc đen.
3. Mỗi con trỏ null mặc định là những nút đen.
4. Nếu một nút là đỏ, những nút con của nó phải là đen.
5. Mọi đường dẫn từ gốc đến nút ngoài phải có cùng số lượng nút đen.

Tính chất 5 còn được gọi là tính chất "cân bằng đen". Số các nút đen trên một đường đi từ gốc tới mỗi lá được gọi là độ dài đen của đường đi đó. Ở đây ta chỉ xét các đường đi từ gốc tới các lá nên ta sẽ gọi tắt các đường đi như vậy là đường đi. Sức mạnh của cây đỏ đen nằm trong các tính chất trên. Từ các tính chất này suy ra trong các đường đi từ gốc tới các lá đường đi dài nhất không vượt quá hai lần đường đi ngắn nhất do số các nút đen trên hai đường đi đó bằng nhau. Do đó cây *RB* là gần cân bằng. Vì các thuật toán chèn, xóa, tìm kiếm trong trường hợp xấu nhất đều tỷ lệ với chiều cao của cây nên cây đỏ đen rất hiệu quả trong các trường hợp xấu nhất không giống như cây tìm kiếm nhị phân thông thường.

Ở cây RB không có đường đi nào từ gốc tới một lá chứa hai nút đỏ liền nhau (theo tính chất 4). Do đó trên mỗi đường số nút đỏ không nhiều hơn số nút đen. Đường đi ngắn nhất là đường đi chỉ có nút đen, đường đi dài nhất có thể là đường đi xen kẽ giữa các nút đỏ và đen.

Các phép toán trên chúng như tìm kiếm (*Search*), chèn (*Insert*), và xóa (*Delete*) trong thời gian  $\mathcal{O}(\log n)$ , trong đó  $n$  là số các phần tử của cây.

## 6.2 – Các phép biến đổi trong cây Red – Black

### a. Phép chèn (*Insert*).

Cây *Red – Black* thực chất là một trường hợp đặc biệt của cây tìm kiếm nhị phân *BST*. Vì vậy ta có thể áp dụng các phép chèn của cây *BST* vào cây *RB*. Nhưng làm vậy sẽ không diễn tả được hiệu quả của cây *RB*. Phép chèn này có độ phức tạp  $O(\log n)$ .

Khi chèn, ta phải tuân thủ đúng các tính chất của cây *Red – Black*. Phép chèn bắt đầu bằng việc bổ sung một nút và gán cho nó màu đỏ.

Gọi  $N$  là nút cần chèn,  $P$  là nút cha của  $N'$ ,  $G$  là nút ông của  $N'$  và  $U$  là nút chú của  $N'$ .

Khi đó, các nút được thể hiện như sau :

```
struct node *grandparent(struct node *n) {
    return n->parent->parent;
}
struct node *uncle(struct node *n) {
    if (n->parent == grandparent(n)->left)
        return grandparent(n)->right;
    else
        return grandparent(n)->left;
}
```

Ta cần xét các trường hợp chèn nút  $N$  như sau :

Trường hợp 1: Nút mới thêm  $N$  ở tại gốc.

Trong trường hợp này, gán lại màu đen cho  $N$ , để bảo toàn tính chất 2 và tính chất 5 của cây *RB*.

```
void insert_case1(struct node *n) {
    if (n->parent == NULL)
        n->color = BLACK;
    else
        insert_case2(n);
}
```

Trường hợp 2 :Nút cha  $P$  của nút mới thêm là đen.

Khi đó cả tính chất 4 và tính chất 5 của cây *RB* đều không bị vi phạm.

```
void insert_case2(struct node *n) {
    if (n->parent->color == BLACK)
```

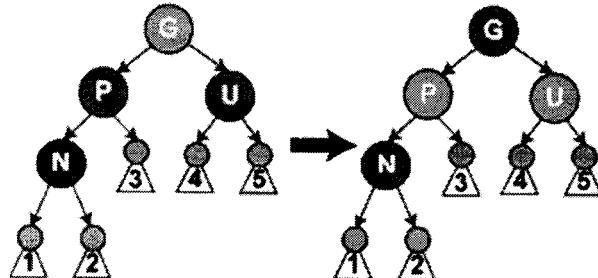
```

        return; /* Tree is still valid */
    else
        insert_case3(n);
}

```

Trong trường hợp, nếu N có ông là nút G, cha P của N là đỏ và P không ở gốc thì G là đen.

Trường hợp 3 :Cả cha P và bác U là đỏ và đổi cả P và U thành đen còn G đổi thành đỏ



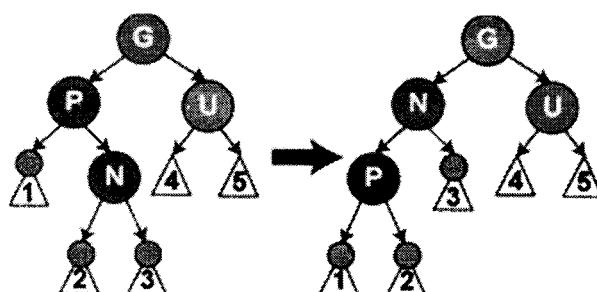
Như vậy sẽ vi phạm tính chất 2 (Gốc là đen) và tính chất 4 (Con của nút đỏ phải là hai nút đen). Trong trường hợp này gọi một thủ tục đệ quy trên G từ trường hợp 1.

```

void insert_case3(struct node *n) {
    if (uncle(n) != NULL && uncle(n)->color == RED) {
        n->parent->color = BLACK;
        uncle(n)->color = BLACK;
        grandparent(n)->color = RED;
        insert_case1(grandparent(n));
    }
    else
        insert_case4(n);
}

```

Trường hợp 4 : Nút cha P là đỏ nhưng nút bác U là đen, nút mới N là con phải của nút P, và P là con trái của nút G



Vì vi phạm tính chất 4 (Con của nút đỏ phải là hai nút đen) nên trong trường hợp này ta sẽ chuyển đổi vị trí hai nút N và P. Sau đó ta định dạng lại nút P như trong trường hợp 5. Cách chuyển đổi được mô tả như sau :

```

void insert_case4(struct node *n) {
    if (n == n->parent->right && n->parent == grandparent(n)->left) {
        rotate_left(n->parent);
    }
}

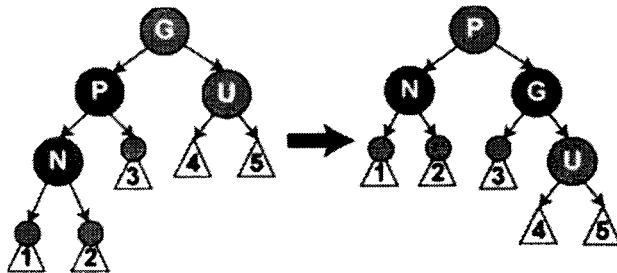
```

```

n = n->left;
} elseif (n == n->parent->left && n->parent == grandparent(n)->right) {
    rotate_right(n->parent);
    n = n->right;
}
insert_case5(n);
}

```

Trường hợp 5 : Nút cha P là đỏ nhưng nút bác U là đen, nút mới N là con trái của P và P là con trái của nút ông G.



Trong trường hợp này, một phép quay phải trên nút ông G được thực hiện, kết quả là trong cây mới nút P trở thành cha của cả hai nút N và G. G là con của P nên G là đen. Đổi màu của P và G thì cây thỏa mãn tính chất 4 và không vi phạm tính chất 5 do các đường đi qua G trước đây bây giờ đi qua P.

```

void insert_case5(struct node *n) {
    n->parent->color = BLACK;
    grandparent(n)->color = RED;
    if (n == n->parent->left && n->parent == grandparent(n)->left) {
        rotate_right(grandparent(n));
    } else {
        /* Here, n == n->parent->right && n->parent == grandparent(n)->right */
        rotate_left(grandparent(n));
    }
}

```

### b. Phép xóa (Delete).

Cũng giống như phép chèn, phép xóa ở cây *Red – Black* cũng giống như trong cây nhị phân, ta cũng có thể áp dụng phép xóa của cây nhị phân vào cây *RB* nhưng sẽ không triệt để và hiệu quả không cao. Xét trường hợp xóa một nút có ít nhất một nút con.

- Trường hợp 1: Nếu ta xóa một nút đỏ thì chắc chắn rằng con của nó là nút đen. Tất cả các đường đi qua nút bị xóa chỉ bớt đi một nút đỏ do đó tính chất 5 không thay đổi. Ngoài ra, cả nút cha và nút con của nút bị xóa đều là nút đen, do đó không vi phạm tính chất 3 và 4.

- Trường hợp 2:Nếu ta xóa một nút đen chỉ có một con là nút đỏ. Khi xóa nút đó các tính chất 4 và 5 bị phá vỡ, nhưng nếu gán lại màu cho nút con là đen thì chúng lại được khôi phục.

- Trường hợp 3:Nếu nút bị xóa và nút con của nó đều là đen. Chúng ta sẽ bắt đầu bằng việc thay nút bị xóa bằng nút con của nó. Gọi nút con trong vị trí mới là N, anh em với nó (con khác của nút cha mới) là S, P là cha mới của N,  $S_L$  chỉ con trái của S và  $S_R$  chỉ con phải của S (chúng tồn tại vì S không thể là lá). Khi đó, anh em của N' được xác định như sau :

```
struct node *sibling(struct node *n) {
    if (n == n->parent->left)
        return n->parent->right;
    else
        return n->parent->left;
}
```

Ta cần hoàn chỉnh các lá *null* sau các phép thay đổi. Nếu nút bị xóa không có con N khác “lá *null*” thì các tính chất được thỏa mãn. Còn nếu N là một “lá *null*” ta sẽ dùng hàm (*function*) *replace\_node* để thay thế cho nút con *child* vào vị trí của nút bị xóa.

```
void delete_one_child(struct node *n) {
    /* Giả thiết: n có ít nhất một con null */
    struct node *child = is_leaf(n->right) ? n->left : n->right;
    replace_node(n, child);
    if (n->color == BLACK) {
        if (child->color == RED)
            child->color = BLACK;
        else
            delete_case1(child);
    }
    free(n);
}
```

Nếu N là một “lá *null*” mà ta không muốn biểu diễn “lá *null*” đó như trên thì ta có thể làm theo giải thuật. Nếu cả N và gốc ban đầu của nó là đen thì sau khi xóa các đường qua N giảm bớt một nút đen vi phạm tính chất 5, cây cần phải cân bằng lại. Khi đó có các trường hợp sau:

- Trường hợp 1 : N là gốc mới. Trong trường hợp này chúng ta dừng lại. Ta đã giải phóng một nút đen khỏi mọi đường đi và gốc mới lại là đen. Không tính chất nào bị vi phạm.

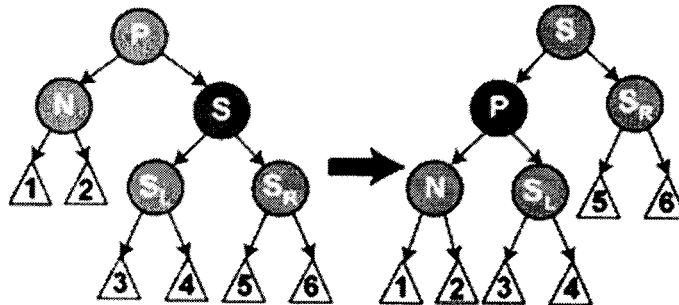
```
void delete_case1(struct node *n) {
    if (n->parent == NULL)
        return;
```

```

        else
            delete_case2(n);
    }
}

```

- Trường hợp 2 : N là con trái của cha P, S là đỏ.



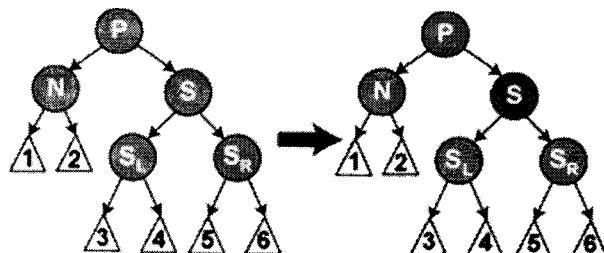
Trong trường hợp này tráo đổi màu của P và S, và sau đó quay trái tại P, nó sẽ làm cho S trở thành nút ông của N (P có màu đen và có một con màu đỏ). Tất cả các đường đi có số các nút đen giống nhau, bây giờ N có một anh em màu đen và cha màu đỏ.

```

void delete_case2(struct node *n) {
    if (sibling(n)->color == RED) {
        n->parent->color = RED;
        sibling(n)->color = BLACK;
        if (n == n->parent->left)
            rotate_left(n->parent);
        else
            rotate_right(n->parent);
        delete_case3(n);
    }
}

```

- Trường hợp 3 :P, S và các con của S là đen.



Lúc này, ta gán S màu đỏ. Kết quả là mọi đường đi qua S (không qua N) có ít hơn một nút đen. Vì việc xóa đi cha trước đây của N' làm tất cả các đường đi qua N bớt đi một nút đen, nên chúng bằng nhau. Tuy nhiên tất cả các đường đi qua P bây giờ có ít hơn một nút đen so với các đường không qua P, do đó tính chất bị vi phạm. Để khắc phục điều này chúng ta tái cân bằng tại P bằng cách bắt đầu lại từ trường hợp 1.

```

void delete_case3(struct node *n) {
    if (n->parent->color == BLACK &&
        sibling(n)->color == BLACK &&
        sibling(n)->left->color == BLACK &&

```

```

        sibling(n)->right->color == BLACK)
    {
        sibling(n)->color = RED;
        delete_case1(n->parent);
    }
else
    delete_case4(n);
}

```

- Trường hợp 4 :S và các con của S là đen nhưng P là đỏ.

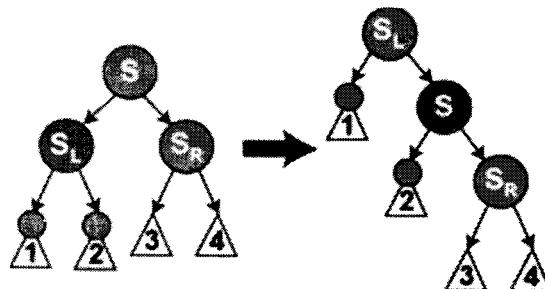
Trong trường hợp này, chúng ta đổi ngược màu của S và P. Như vậy, không ảnh hưởng tới số nút đen trên các đường đi không qua N, nhưng thêm một nút đen trên các đường đi qua N, thay cho nút đen đã bị xóa trên các đường này.

```

void delete_case4(struct node *n) {
if (n->parent->color == RED &&
    sibling(n)->color == BLACK &&
    sibling(n)->left->color == BLACK &&
    sibling(n)->right->color == BLACK)
{
    sibling(n)->color = RED;
    n->parent->color = BLACK;
}
else
    delete_case5(n);
}

```

- Trường hợp 5 :S là đen, con trái của S là đỏ, con phải của S là đen, còn N là con trái của cha nó.



Trong trường hợp này ta đặt nút anh em mới của N là S. Khi đó, ta quay phải tại S, khi đó con trái của S trở thành cha của S. Tiếp theo, ta tráo đổi màu của S và cha mới của nó. Tất cả các đường đi sẽ có số nút đen như nhau, nhưng bây giờ N có một người anh em đen mà con phải của nó lại là đỏ, chúng ta chuyển sang trường hợp 6.

```

void delete_case5(struct node *n) {
if (n == n->parent->left &&
    sibling(n)->color == BLACK &&
    sibling(n)->left->color == RED &&
    sibling(n)->right->color == BLACK)
{
    sibling(n)->color = RED;
}

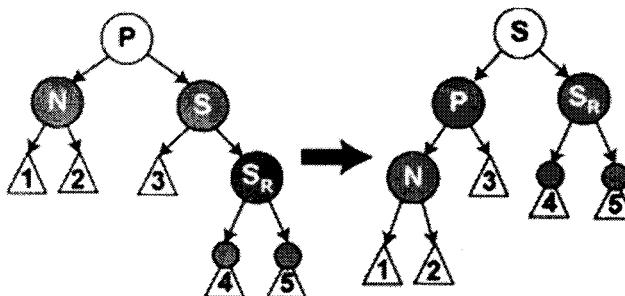
```

```

    sibling(n)->left->color = BLACK;
    rotate_right(sibling(n));
}
else if (n == n->parent->right &&
        sibling(n)->color == BLACK &&
        sibling(n)->right->color == RED &&
        sibling(n)->left->color == BLACK)
{
    sibling(n)->color = RED;
    sibling(n)->right->color = BLACK;
    rotate_left(sibling(n));
}
delete_case6(n);
}

```

- Trường hợp 6: S là đen, con phải của S là đỏ và N là con trái của nút cha P.



Trong trường hợp này chúng ta quay trái tại P, khi đó S trở thành cha của P và con phải của S. Chúng ta hoán đổi màu của P và S, và gán cho con phải của S màu đen. N bây giờ có thêm một nút đen tiền nhiệm: hoặc P mới bị tô đen, nó đã là đen và S là nút ông của nó trở thành đen. Như vậy các đường đi qua N có thêm một nút đen. Trong lúc đó, một đường đi không đi qua N, có hai khả năng:

- Đi qua nút anh em của N. Khi đó cả trước và sau khi quay nó phải đi qua S và P, khi thay đổi màu sắc hai nút này đã tráo đổi màu cho nhau. Như vậy đường đi này không bị thay đổi số nút đen.
- Đi qua nút bác của N', là con phải của S. Khi đó trước khi quay nó đi qua S, cha của S và con phải của S, nhưng sau khi quay nó chỉ đi qua nút S và con phải của S, khi này S đã nhận màu cũ của cha P còn con phải của S đã đổi màu từ đỏ thành đen. Kết quả là số các nút đen trên đường đi này không thay đổi.

Như vậy, số các nút đen trên các đường đi là không thay đổi. Do đó các tính chất 4 và 5 đã được khôi phục. Nút trắng trong hình vẽ có thể là đỏ hoặc đen, nhưng phải ghi lại trước và sau khi thay đổi.

```

void delete_case6(struct node *n) {
    sibling(n)->color = n->parent->color;
    n->parent->color = BLACK;
}

```

```

if (n == n->parent->left) {
    sibling(n)->right->color = BLACK;
    rotate_left(n->parent);
}
else
{
    sibling(n)->left->color = BLACK;
    rotate_right(n->parent);
}

```

## 7 – Vùng đống Fibonacci heap

Đống Fibonacci được phát triển bởi Michael L.Fredman và Robert E.Tarjan vào năm 1984 và lần đầu tiên được công bố trong một tạp chí khoa học vào năm 1987. Tên của Fibonacci heap xuất phát từ dãy số Fibonacci được sử dụng trong phân tích thời gian chạy.

Đống Fibonacci là một cấu trúc dữ liệu đống bao gồm một bộ sưu tập các cây. Vì vậy, nó cũng được ghép vào cấu trúc cây. Thời gian thực hiện giải thuật của nó hơn rất nhiều của một đống nhị thức .

Hoạt động tìm kiếm ít nhất là trong thời gian  $O(1)$ , còn hoạt động chèn, giảm giá trị khóa, và kết nối trong thời gian tối thiểu là  $O(\log n)$ . Điều này có nghĩa là từ một cấu trúc dữ liệu nào, bất kỳ chuỗi các hoạt động  $a$  từ nhóm đầu tiên và các hoạt động  $b$  từ nhóm thứ hai sẽ mất  $O(a+b\log n)$  thời gian. Trong một đống nhị thức các hoạt động sẽ mất  $O((a+b)\log (n))$  thời gian, tốt hơn so với một đống nhị thức khi  $b$ .

Sử dụng Fibonacci đống cho hàng đợi ưu tiên làm cải thiện thời gian xử lý của thuật toán.

### 7.1 – Cấu trúc của Fibonacci heap

Một đống Fibonacci là một tập hợp các cây đáp ứng các điều kiện tối thiểu đống. Điều này ngụ ý rằng khóa của đống luôn luôn là gốc của một trong những cây trong đống. So với một đống nhị thức, cấu trúc của một đống Fibonacci là linh hoạt hơn. Cây không có một hình dạng theo quy định và trong trường hợp cực đoan đống có thể có tất cả các yếu tố trong một cây riêng biệt. Sự linh hoạt này cho phép một số hoạt động sẽ được thực hiện và làm trì hoãn công việc cho các hoạt động sau đó. Ví dụ đống sáp nhập được thực hiện đơn giản bằng cách ghép hai danh sách cây, và hoạt động giảm giá trị khóa đôi khi cắt một nút từ cha mẹ của nó và tạo thành một cây mới.

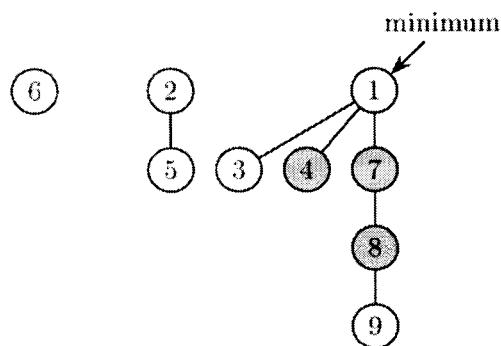
Đặc biệt, mức độ các nút được lưu giữ khá thấp, mỗi nút có bậc nhiều nhất là  $O(\log n)$  và kích thước của một cây con bắt nguồn từ một nút của mức độ k ít nhất là  $F_{k+2}$  trong đó  $F_k$  là thứ k số Fibonacci . Điều này đạt được bởi các quy tắc mà chúng ta có thể cắt giảm nhiều nhất là một con của mỗi nút không phải root. Khi đưa con thứ hai được cắt, nút chính nó cần phải được cắt giảm từ cha mẹ của nó và trở thành gốc rẽ của một cây. Số cây được giảm trong hoạt động tối thiểu xóa, nơi cây được liên kết với nhau.

Là kết quả của một cấu trúc, một số hoạt động có thể mất một thời gian dài trong khi những hoạt động khác được thực hiện rất nhanh chóng. Giả sử rằng các hoạt động xảy ra rất nhanh và mất một ít thời gian. Thêm thời gian này sau đó kết hợp và trừ vào thời gian vận hành thực tế của hoạt động chậm. Số lượng thời gian lưu để sử dụng sau được đo tại bất kỳ thời điểm nào bằng một chức năng tiềm năng. Tiềm năng của một đồng Fibonacci được cho bởi

$$\text{Tiềm năng} = t + 2m$$

Trong đó : t là số cây trong đồng Fibonacci  
m là số lượng các nút được đánh dấu.

Một nút được đánh dấu nếu ít nhất một trong các con của nó đã bị cắt từ nút này đã được thực hiện một đứa con của một nút khác (tất cả các rẽ là đánh dấu). Thời gian thực hiện cho một hoạt động được đưa ra bởi tổng thời gian thực tế và c lần sự khác biệt trong tiềm năng, trong đó c là một hằng số (lựa chọn để phù hợp với các yếu tố liên tục trong các ký hiệu O cho thời gian thực tế).



Trong ví dụ ta thấy nó có ba cây của độ 0, 1 và 3. Ba đỉnh được thể hiện trong màu xanh. Do đó, tiềm năng của heap là 9 (3 cây + 2 \* 3 đỉnh).

Như vậy, gốc rẽ của từng cây trong một đồng có một đơn vị thời gian lưu trữ. Sau này đơn vị thời gian này có thể được sử dụng để liên kết cây này với các cây khác vào thời gian thực hiện 0. Ngoài ra, mỗi nút được đánh dấu có hai đơn vị thời gian lưu trữ.

Người ta có thể được sử dụng để cắt các nút từ cha của nó. Nếu điều này xảy ra, các nút sẽ trở thành một nút khóa và đơn vị thứ hai của thời gian sẽ vẫn lưu giữ trong nó như trong bất kỳ thư mục gốc khác.

## 7.2 – Các phép xử lý

Các phép xử lý trên Fibonacci heap bao gồm :

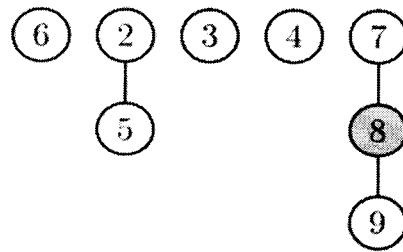
- ◆  $Q.initialize()$ : khởi tạo một hàng đợi rỗng  $Q$ .
- ◆  $Q.isEmpty()$ : trả về giá trị đúng.
- ◆  $Q.insert(e)$ : chèn  $e$  vào hàng đợi  $Q$  và trả về nút có chứa  $e$ .
- ◆  $Q.deleteMin()$ : trả về phần tử tối thiểu của  $Q$  và xóa nó.
- ◆  $Q.min()$ : trả về phần tử tối thiểu của  $Q$ .
- ◆  $Q.decreaseKey(v, k)$ : giảm giá trị của  $v$  thành giá trị  $k$ .

Vun đồng Fibonacci cho phép xóa nhanh và nối các nút khóa của tất cả các cây được liên kết bằng cách sử dụng một vòng tròn, danh sách liên kết kép. Các nhánh con của mỗi nút cũng được liên kết bằng cách sử dụng một danh sách như vậy. Đối với mỗi nút, ta duy trì số lượng các con và các nút được đánh dấu. Hơn nữa ta duy trì một con trỏ ở thư mục gốc.

Hoạt động tìm  $min$  bây giờ là đơn giản bởi vì ta giữ cho con trỏ đến nút chứa nó. Nó không thay đổi tiềm năng của heap, do đó cả chi phí thực tế được phân bổ là không đổi. Như đã đề cập ở trên, phép kết nối được thực hiện đơn giản bằng cách ghép danh sách các nút chính của hai đồng. Điều này có thể được thực hiện trong thời gian liên tục và khả năng không thay đổi, dẫn đến độ phức tạp của bài toán thay đổi.

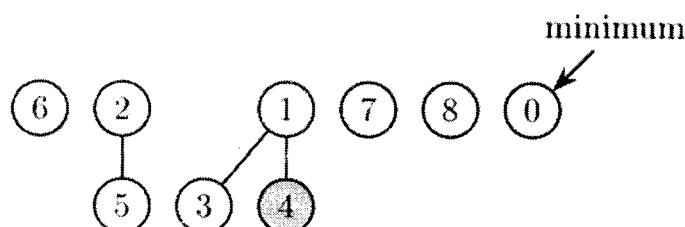
Hoạt động chèn bằng cách tạo ra một đồng mới với một phần tử và làm hợp nhất. Điều này cần có thời gian liên tục, và tăng tiềm năng, bởi vì số lượng của cây tăng lên. Độ phức tạp của giải thuật là không đổi.

Hoạt động xử lý nút cơ sở (tương tự như xóa) hoạt động trong ba giai đoạn. Trước tiên, ta tới tận nút có chứa nút con đó và loại bỏ nó. Con của nó sẽ trở thành nút cha của những cây mới. Nếu có số nút con là  $d$ , phải mất thời gian  $O(d)$  để xử lý tất cả các rẽ mới và tăng tiềm năng bằng cách  $d - 1$ . Do đó, phân bổ thời gian hoạt động của giai đoạn này là  $O(d) = O(\log n)$ . Ở giai đoạn đầu của hoạt động xử lý ví dụ trên sẽ được thể hiện như sau:

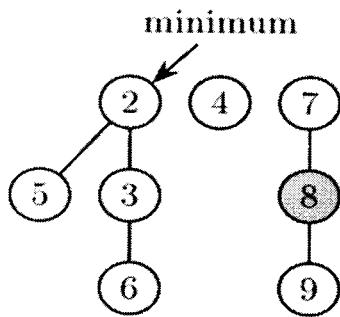


Tuy nhiên để hoàn thiện các hoạt động, ta cần phải cập nhật con trỏ vào thư mục gốc với chính nút cơ sở. Có thể lên đến  $n$  nút ta cần phải kiểm tra. Trong giai đoạn hai, ta giảm số lượng nút của cây và gắn các nút kết với nhau sao cho cùng một mức độ. Khi hai nhân  $u$  và  $v$  có cùng một mức độ, ta thực hiện một trong số họ là con của cây khác để một với phím nhỏ hơn vẫn là nút khóa. Mức độ của nó sẽ tăng một. Điều này được lặp đi lặp lại cho đến khi mọi nút khóa có một mức độ khác nhau. Để tìm cây của cùng một mức độ hiệu quả, ta sử dụng một mảng chiều dài  $O(\log n)$ , trong đó, ta giữ một con trỏ đến một thư mục gốc của mỗi mức độ. Khi một nút thứ hai được tìm thấy trong cùng một mức độ thì liên kết và mảng được cập nhật. Thời gian chạy thực tế là  $O(\log n + m)$  trong đó  $m$  là số khóa vào đầu giai đoạn thứ hai. Cuối cùng, ta sẽ có ít nhất  $O(\log n)$  nút khóa. Do đó sự khác biệt trong chức năng tiềm năng từ trước khi giai đoạn này để sau đó là:  $O(\log n) - m$ , độ phức tạp của bài toán nhiều nhất là  $O(\log n + m) + c(O(\log n) - m)$ . Với một sự lựa chọn đủ lớn của  $c$ , điều này giúp đơn giản hóa đến  $O(\log n)$ .

Ta sẽ thấy, ví dụ trên sau khi giảm nút khóa chính của nút 9-0. Nút này cũng như hai nút khóa chính được đánh dấu và cắt từ một cây có bắt nguồn từ 1 và đặt như nút khóa mới.



Trong giai đoạn thứ ba, ta kiểm tra từng nút khóa chính còn lại. Giai đoạn này thực hiện trong thời gian  $O(\log n)$  và khả năng không thay đổi.



Đây là kết quả đạt được sau khi hoàn thiện xử lý các nút cơ sở.

Hoạt động giảm khóa sẽ mất đi các nút, và nếu khóa mới là nhỏ hơn so với cha của nó (cha không phải là một gốc) nó được đánh dấu. Nếu nó đã được đánh dấu, nó được cắt là tốt nhất. Ta tiếp tục trở lên cho đến khi đạt được một trong hai nút chính hoặc một nút chính được đánh dấu. Mỗi cây mới, ngoại trừ cây đầu tiên được đánh dấu ban đầu như vậy một nút khóa chính sẽ trở nên không rõ ràng. Do đó, số lượng các nút được đánh dấu thay đổi bởi  $-(k-1)+1=-k+2$ . Kết hợp 2 thay đổi, những thay đổi tiềm năng bằng  $2(-k+2)+k=-k+4$ . Thời gian thực tế để thực hiện việc cắt là  $O(k)$ , do đó với một sự lựa chọn đủ lớn  $c$  thời gian thực hiện giải thuật là không đổi.

Hoạt động xóa có thể được thực hiện đơn giản bằng cách giảm giá trị của các nút khóa đó, do đó biến nó thành nút cơ sở của toàn bộ đồng. Thời gian xử lý bài toán của hoạt động này là  $O(\log n)$ .

Các phép xử lý khác trong Fibonacci heap gồm:

$Q.delete(v)$ : xóa  $v$  từ  $Q$  mà không thực hiện tìm kiếm  $v$

$Q.meld(Q')$ : kết hợp  $Q$  và  $Q'$ .

$Q.search(k)$ : tìm kiếm các phần tử  $k$  quan trọng trong  $Q$ .

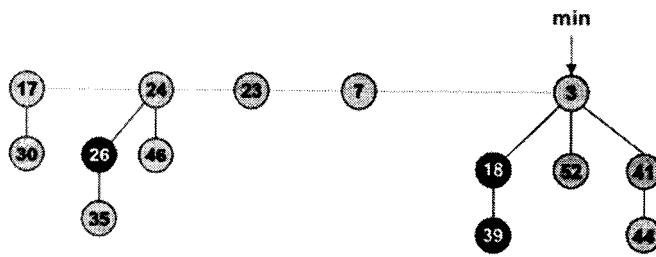
### **Các phép biến đổi trong Fibonacci heap**

#### *a. Phép chèn (Insert)*

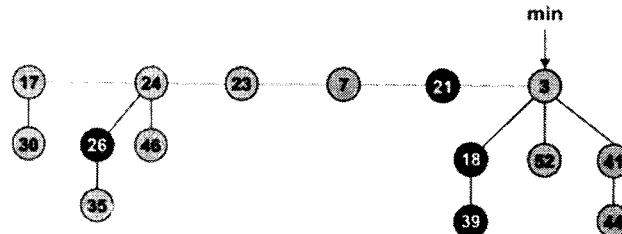
Phép chèn trong cây Fibonacci được thực hiện như sau:

- Tạo ra một cây đơn mới.
- Thêm vào bên trái của con trỏ nhỏ nhất.
- Cập nhật lại con trỏ.

Ví dụ : Chèn phần tử 21 vào cây sau :



Thực hiện phép chèn như đã nêu trên ta được cây mới như sau :

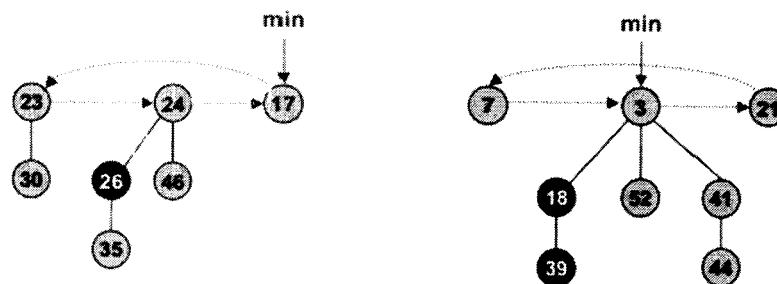


### b. Phép kết hợp(Union)

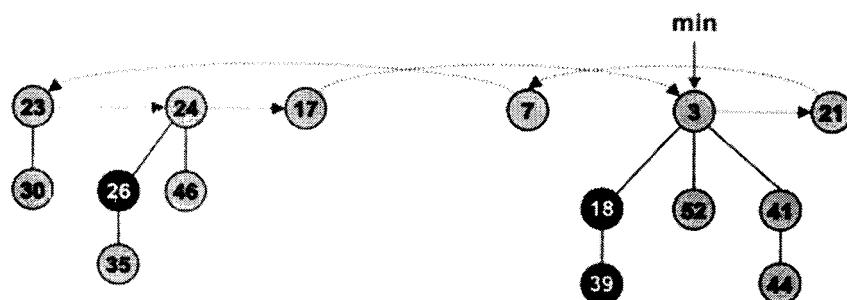
Phép kết hợp dùng để nối hai đống Fibonacci.

Danh sách gốc trước khi nối là hình tròn, danh sách sau khi kết hợp là kép.

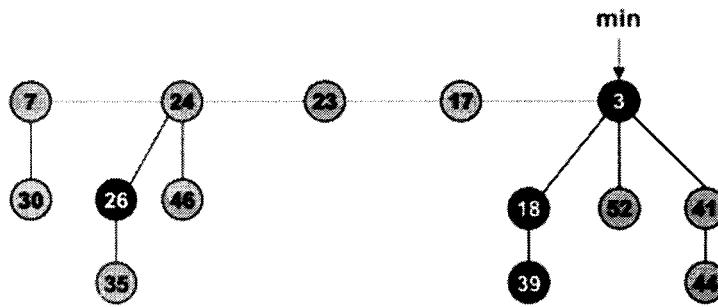
Ví dụ: kết hợp hai cây Fibonacci sau



Ta sẽ thực hiện như sau :



Ta được kết quả như sau :



### c. Phép xóa (Delete)

#### Delete min

- Phép delete min thực hiện xóa phần tử nhỏ nhất và nối các con của nó vào cây gốc.
- Củng cố lại cây để cây không tồn tại hai rễ có cùng một cấp độ.

```

 $A = Q.\text{min}();$ 
if  $Q.\text{size}() > 0$ 
    then  $\text{remove } Q.\text{min}()$  from  $Q.\text{rootlist};$ 
         $\text{add } Q.\text{min}.childlist$  to  $Q.\text{rootlist};$ 
         $Q.\text{consolidate}();$ 
    return  $A;$ 

```

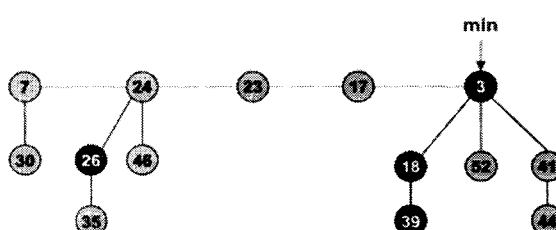
Với A là một mảng có độ dài  $2\log n$  ứng với nút trong đống Fibonacci

```

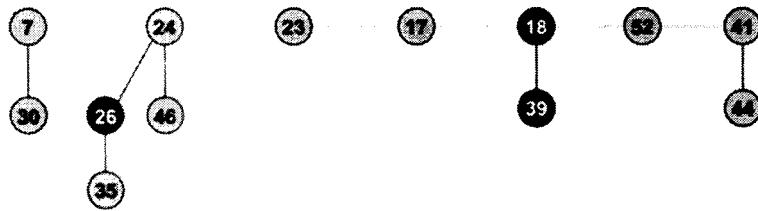
for  $i = 0$  to  $2 \log n$  do  $A[i] = \text{null};$ 
    while  $Q.\text{rootlist} \neq \emptyset$  do
         $B = Q.\text{delete-first}();$ 
        while  $A[\text{rank}(B)]$  is not null do
             $B' = A[\text{rank}(B)];$ 
             $A[\text{rank}(B)] = \text{null};$ 
             $B = \text{link}(B, B');$ 
        end while
         $A[\text{rang}(B)] = B;$ 
    end while
determine  $Q.\text{min};$ 

```

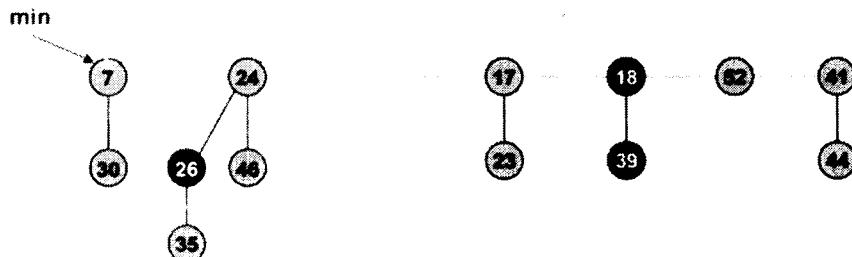
Ví dụ :



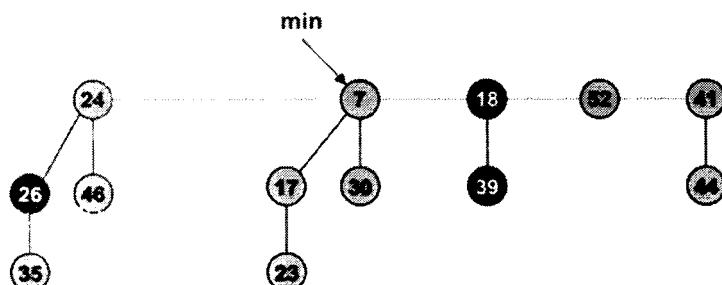
Sau khi xóa phần tử  $min$ , cây được cập nhật như sau:



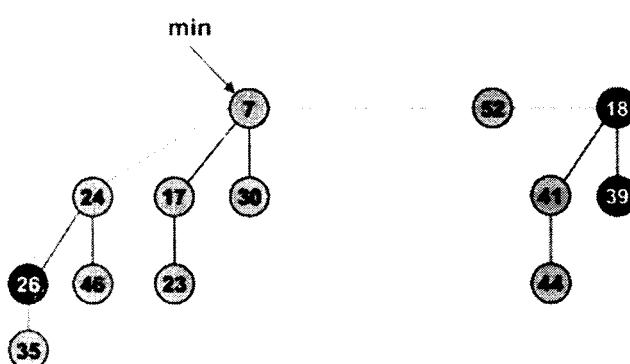
Xét thấy 23 và 17 cùng ở độ sâu 0 nên ghép 23 và 17 thành 1 cây như sau:



Tương tự, 7 và 17 cùng ở độ sâu nên ghép chúng thành 1 cây.



Cứ như vậy ta sẽ được cây hoàn chỉnh sau khi xét tất cả các nút như sau :



#### d. Giảm giá trị khóa (Decrease key).

Khi thực hiện giảm khóa của một yếu tố nào đó từ giá trị  $u$  xuống giá trị  $v$  ta cần xét các trường hợp sau:

- Trường hợp 1:  $min$  của đồng không vi phạm.
  - Giảm khóa từ giá trị  $u$  xuống giá trị  $v$ .
  - Thay đổi con trỏ  $min$  của đồng nếu cần thiết.
- Trường hợp 2 : nút cha của  $u$  không được đánh dấu.
  - Giảm khóa từ giá trị  $u$  xuống giá trị  $v$ .

- Cắt đứt mối quan hệ của  $u$  và cha của nó.
- Đánh dấu cha của  $u$ .
- Thêm cây có gốc là  $u$  vào danh sách đồng, cập nhật giá trị  $\min$  của đồng.
- Trường hợp 3 : nút cha của  $u$  được đánh dấu.
  - Giảm khóa từ giá trị  $u$  xuống giá trị  $v$ .
  - Cắt đứt mối quan hệ giữa  $u$  và cha  $p$  của nó, thêm  $u$  vào danh sách gốc.
  - Cắt đứt mối quan hệ giữa  $p$  và nút cha của  $p$  để nhô tận gốc.

Nếu cha của  $p$  không được đánh dấu : làm theo trường hợp 2.  
 Nếu cha của  $p$  được đánh dấu : làm như trường hợp 3 và cứ lặp lại như vậy.

Đoạn mã lệnh được thể hiện như sau

```

if  $k > v.key$  then return
     $v.key = k;$ 
    update  $Q.\min$ ;
    if  $v \in Q.rootlist$  or  $k \geq v.parent.key$ 
        then return;
do
     $parent = v.parent;$ 
     $Q.cut(v);$ 
     $v = parent;$ 
    while  $v.mark$  and  $v \notin Q.rootlist$ 
    if  $v \notin Q.rootlist$  then  $v.mark = true$ ;
```

Mã lệnh  $Q.cut(v)$

```

if  $v \notin Q.rootlist$ 
then  $rank(v.parent) = rank(v.parent) - 1;$ 
     $v.parent = null;$ 
remove  $v$  from  $v.parent.childlist$ 
add  $v$  to  $Q.rootlist$ 
```

## 7.2 – Phạm vi sử dụng của Fibonacci heap

Việc xử lý bài toán của một đồng Fibonacci phụ thuộc vào mức độ (số lượng con) của bất kỳ gốc cây con  $O(\log n)$ , trong đó  $n$  là kích thước của heap. Ở đây kích thước bắt nguồn từ cây tại bất kỳ nút  $x$  của mức độ  $d$  trong đồng phải có kích thước ít nhất là  $F_{d+2}$ , trong đó  $F_k$  là thứ  $k$  số Fibonacci. Mức độ ràng buộc sau từ này và thực tế mà  $F_{d+2} \geq \varphi_d$  cho tất cả các số nguyên  $d \geq 0$ ,  $\varphi = (1 + \sqrt{5})/2 = 1,618$ .

Xét bất kỳ nút  $x$  ở trong đồng ( $x$  cần không phải là gốc của một trong những cây chính). Xác định kích thước  $x$  là kích thước của cây có gốc là  $x$  (số lượng con của  $x$ ). Chúng ta chứng minh bằng cảm ứng trên chiều cao của  $x$  (chiều dài của một con đường đơn giản nhất từ  $x$  đến một lá hậu duệ), kích thước  $x \geq F_{d+2}$ , trong đó  $d$  là độ  $x$ .

Trường hợp cơ sở: Nếu  $x$  có chiều cao 0,  $d = 0$  và kích thước  $x = l = F_2$ .

Trường hợp quy nạp: Giả sử  $x$  có chiều cao và mức độ tích cực  $d > 0$ . Cho  $y_1, y_2, \dots$  là con của  $x$ , lập chỉ mục theo thứ tự thời gian họ gần đây nhất được thực hiện con của  $x$  ( $y_1$  là sớm nhất và  $y_d$  là muộn nhất) và để cho  $c_1, c_2, \dots$  là độ tương ứng. Chúng tôi cho rằng  $c_i \geq i - 2$  (với  $2 \leq i \leq d$ ). Kể từ đỉnh cao của tất cả các  $y_i$  ít hơn so với  $x$ , chúng ta có thể áp dụng giả thuyết quy nạp cho họ để có được kích thước  $y_i \geq F_{ci+2} \geq F_{(i-2)2} = F_i$ .

Các nút  $x$  và  $y_1$  đóng góp ít nhất 1 kích thước  $x$ , và vì vậy chúng ta có :

$$\text{size}(x_i) \geq 2 + \sum_{i=2}^d \text{size}(y_i) \geq 2 + \sum_{i=2}^d F_i = l + \sum_{i=2}^d F_i$$

Ta cũng chứng minh được rằng  $l + \sum_{i=0}^d F_i = F_{d+2}$  cho bất kỳ  $d \geq 0$  mang đến cho các mong muốn thấp hơn ràng buộc về kích thước  $x$ .

## Kỹ thuật lập trình

### 1. Cơ sở lập trình

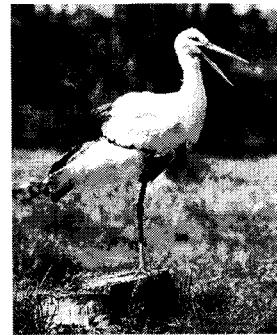
#### VH39. DIỆC

Tên chương trình: HERONS.???

Lần đầu tiên được đi tới vườn bách thú Jimmy thích nhất các con diệc vì nhiều con trong số chúng đứng một chân trông rất ngộ nghĩnh, khi đó chân kia không thấy đâu như vốn sinh ra chúng đã chỉ có một chân. Jimmy đếm được tất cả có **a** chân.

Sau khi đi xem các con thú khác Jimmy lại quay về chỗ chuồng diệc. Một số con đã thay đổi vị trí và cách đứng, Jimmy đếm lại một lần nữa và có số chân là **b**.

Qua số chân thì không thể xác định chính xác có tất cả bao nhiêu con diệc trong chuồng nhưng Gimmy vẫn muốn biết có ít nhất và nhiều nhất là bao nhiêu con.



Hãy xác định số lượng diệc tối thiểu và tối đa.

**Dữ liệu:** Vào từ file văn bản HERONS.INP gồm một dòng chứa 2 số nguyên **a** và **b** ( $1 \leq a, b \leq 10^9$ ).

**Kết quả:** Đưa ra file văn bản HERONS.OUT trên một dòng 2 số nguyên xác định số lượng diệc tối thiểu và tối đa.

**Ví dụ:**

HERONS INP
3 4

HERONS.OUT
2 3



```

#include <iostream>
#include <algorithm>
using namespace std;
ifstream fi ("herons.inp");
ofstream fo ("herons.out");
int a,b,max1,max2,min1,min2,r1,r2;

int main()
{ fi>>a>>b;
  max1=a; min1=(a+1)/2;
  max2=b; min2=(b+1)/2;
  r1=max(min1,min2); r2=min(max1,max2);
  fo<<r1<<' '<<r2;
}

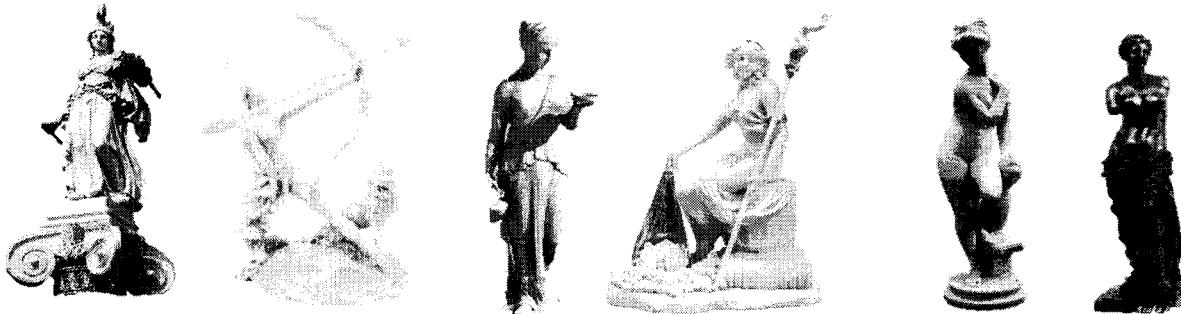
```

Sử dụng hàm toán học

## 16M. HẸN GẶP

Tên chương trình: APPPOINTMENT.???

Thành phố Gloaming (*Hoàng hôn*) nổi tiếng với đường dẫn vào công viên thành phố. Các bức tượng tuyệt đẹp theo chủ đề thần thoại Hy lạp – La mã đặt dọc theo con đường có một sức hút không cưỡng được với mọi khách du lịch. Còn khi những tia nắng cuối cùng trong ngày miến cưỡng rời khỏi bầu trời thì sương mù dày đặc, như một tấm voan trắng mềm mại từ từ rũ xuống. Bây giờ đứng cách quá  $r$  mét là đã không nhìn thấy mặt nhau và các bức tượng trở thành nơi lý tưởng cho các đôi nam nữ thanh niên hẹn hò.



James Bond cần gặp gấp 2 điệp viên nội tuyến của mình để nhận các mật báo khẩn. Không muốn 2 người này nhìn thấy nhau, Bond hẹn gặp mỗi người ở một bức tượng sao cho khoảng cách giữa chúng lớn hơn  $r$ . Trên đường có  $n$  tượng, bức tượng thứ  $i$  ở vị trí cách đầu con đường  $d_i$  mét  $i = 1 \dots n$ ,  $1 \leq d_1 < d_2 < \dots < d_n \leq 10^9$ .

Hãy xác định James Bond có bao nhiêu cách chọn địa điểm.

**Dữ liệu:** Vào từ file văn bản APPPOINTMENT.INP:

- ◆ Dòng đầu tiên chứa 2 số nguyên  $n$  và  $r$  ( $1 \leq n \leq 3 \times 10^5$ ,  $1 \leq r \leq 10^9$ ),
- ◆ Dòng thứ 2 chứa  $n$  số nguyên  $d_1, d_2, \dots, d_n$ .

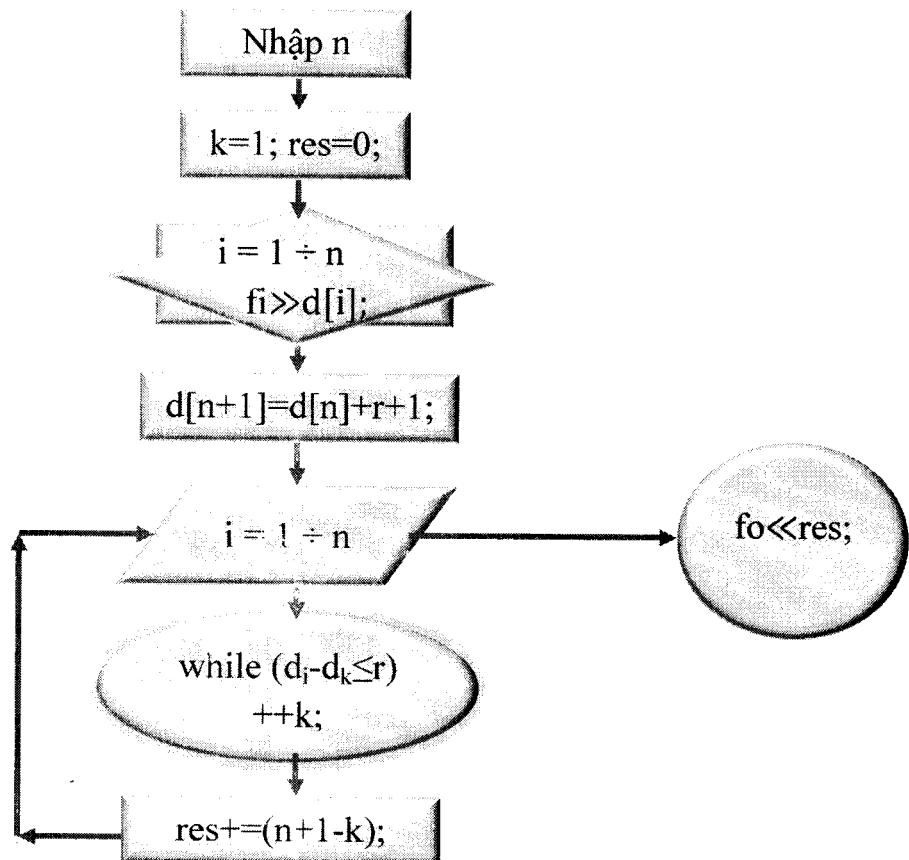
**Kết quả:** Đưa ra file văn bản APPPOINTMENT.OUT một số nguyên – số cách chọn 2 bức tượng để gặp điệp viên.

**Ví dụ:**

APPOINTMENT.INP
4 4
1 3 5 8

APPOINTMENT.OUT
2





### Giải thuật

Độ phức tạp:  $O(n)$ .

## VJ18. SÂN VẬN ĐỘNG

Tên chương trình: STADIUM.???

Để chuẩn bị cho World Cup ở Berland đã xây dựng một sân vận động lớn nhất thế giới. Chỗ ngồi của khán giả được chia thành các sectors giống nhau, mỗi sector có nhiều hàng ghế, mỗi hàng có n ghế đánh số từ 1 đến  $n$  từ trái qua phải và khán giả có thể vào chỗ ngồi của mình từ lối đi ở cả 2 đầu của hàng ghế. Hướng dẫn viên có nhiệm vụ chỉ cho khán giả vào chỗ từ đầu này hay đầu kia để họ không phải đi qua người đã ngồi trên hàng đó. Mỗi người sẽ đi vào chỗ của mình sau khi người trước (nếu có) đã ngồi vào chỗ của mình.

Tất cả các vé trên hàng đều đã được bán hết. Tuy vậy, không thể biết trước khán giả sẽ đến sân theo trình tự nào. Mỗi trường hợp bố trí được cả  $n$  người vào chỗ thỏa mãn yêu cầu đã nêu được gọi là một tình huống tốt.

Hãy xác định số tình huống tốt và đưa ra theo mô đun  $10^9 + 7$ .

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

**Kết quả:** Đưa ra file văn bản STADIUM.OUT kết quả tính được dưới dạng số nguyên.

**Ví dụ:**

STADIUM.INP
3

STADIUM.OUT
4



## Phân tích

Số lượng tình huống tốt khi khán giả vào chỗ ngồi tương đương với số lượng khả năng khi khán giả ra về, trong đó người ra về tiếp theo sẽ là một trong 2 người còn lại ngoài cùng trong dãy ghế.

Giả thiết lần lượt từng người một đứng dậy ra về. Trừ người cuối cùng, có 2 khả năng cho lượt tiếp theo: người ngoài cùng bên phải hoặc bên trái hàng ghế ra về. Với người cuối cùng – chỉ có một khả năng! Như vậy tổng số khả năng cần tìm sẽ là  $2^{n-1}$ .

Vấn đề phức tạp là phải tính số dư của  $2^{n-1}$  chia cho  $p = 10^9 + 7$  với  $n$  rất lớn. Vấn đề này có thể giải quyết bằng việc **ứng dụng sơ đồ nhân Ai Cập tính phần dư của một lũy thừa chia cho một số nguyên**.

Xét bài toán tổng quát: Cho 3 số nguyên dương  $a$ ,  $n$  và  $p$ ,  $1 \leq a, p \leq 10^9$ ,  $1 \leq n \leq 10^{15}$ . Hãy tính số dư nhận được khi chia  $a^n$  cho  $p$ .

*Giải thuật tính số dư:*

$$\begin{cases} u \% p = x \\ v \% p = y \end{cases} \rightarrow (u \times v) \% p = (x \times y) \% p$$

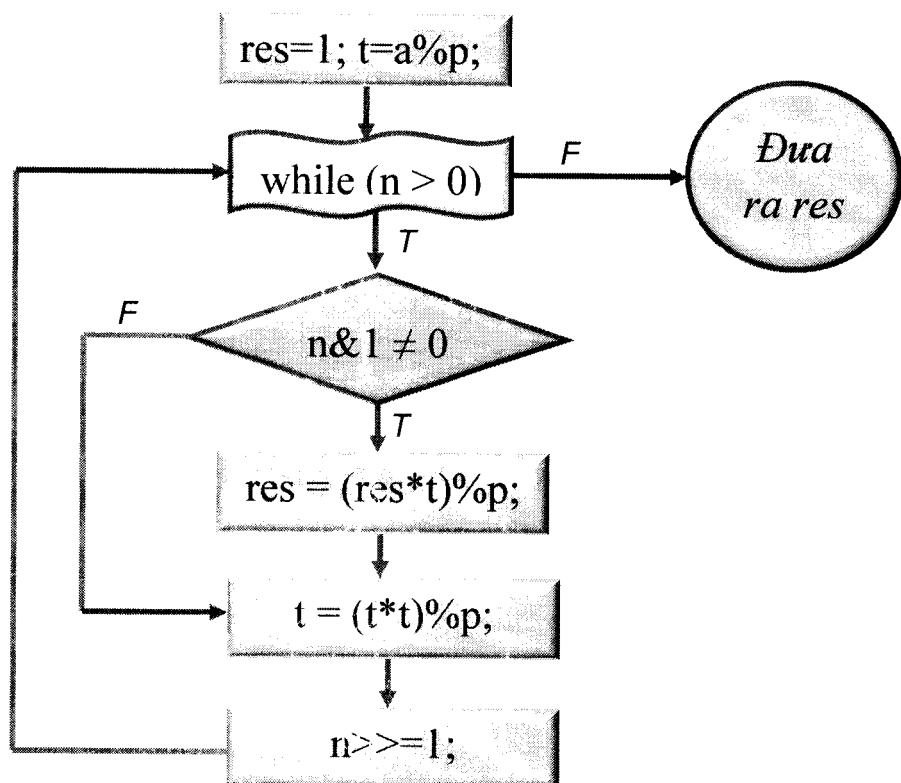
Dễ dàng nhận thấy rằng:

trong đó % là phép chia lấy phần dư.

Với  $u = v$  ta có  $u^2 \% p = x^2 \% p$ .

$a^n$  có thể biểu diễn dưới dạng  $a^n = a^{k_1+k_2+\dots+k_m}$ , trong đó  $k_1 < k_2 < \dots < k_m$  và  $k_i$  là lũy thừa của 2,  $i = 1 \div m$ . Từ  $1 \leq n \leq 10^{15}$  suy ra  $m \leq 50$ .

Biết  $a^1 \% p$  có thể dễ dàng tính ra  $a^2 \% p$ , từ đó  $-a^4 \% p$ ,  $a^8 \% p, \dots, a^{k_m} \% p$ .



Ví dụ, với  $a = 3$ ,  $n = 14$ ,  $p = 100$ , ta có  $3^{14} = 4782969 \rightarrow 3^{14} \% 100 = 69$ .

Kết quả này có thể nhận được theo sơ đồ tính toán sau:

$$n = 14_{10} = 1110_2$$

<i>n</i>	<i>t</i>	<i>tp = t % p</i>	<i>Tích lũy</i> <i>(res*tp)%p</i>	<i>res</i>
0	3	bình phuong 3		1
1	9	bình phuong 9	----->	9
1	81	bình phuong 81	----->	29
1	6561	bình phuong 61	----->	69

Đoạn mã chương trình:

```

res=1; t=a%p;
while(n>0)
{if(n&1) res=(res*t)%p;
 t=(t*t)%p;
 n>>=1;
}

```

Chương trình:

```
#include <fstream>
using namespace std;
ifstream fi ("stadium.inp");
ofstream fo ("stadium.out");
int64_t n,res,r,d=1000000007;

int main()
{fi>>n;--n;
res=1; r=2;
while(n>0)
{if(n&1)res=(res*r)%d;
r=(r*r)%d;n>>-1;
}
fo<<res;
}
```

Tính  $2^n \text{ mod } d$   
với  $n$  rất lớn

Sơ đồ nhân Ai Cập

## VP09. BÀNG LŨY THỬA

Tên chương trình: POWERTAB.???

Cho lưới ô vuông kích thước  $n \times m$  gồm n dòng và m cột. Các dòng được đánh số từ trên xuống dưới bắt đầu từ 1, các cột được đánh số từ trái sang phải bắt đầu từ 1.

Ô ( $i, j$ ) chứa giá trị  $a_{ij} = x^{(i-1) \times m + j}$ .

Cho  $q$  truy vấn, mỗi truy vấn xác định bởi 5 số nguyên  $x1, x2, y1, y2$  và  $p$ , trong đó  $1 \leq x1 \leq x2 \leq n, 1 \leq y1 \leq y2 \leq m, 1 \leq p \leq 10^9 + 7$ .

Với mỗi truy vấn yêu cầu xác định  $(\sum_{i=x1}^{x2} \sum_{j=y1}^{y2} a_{ij}) \bmod p$ .

**Dữ liệu:** Vào từ file văn bản POWERTAB.INP:

- ◆ Dòng đầu tiên chứa 3 số nguyên  $n, m$  và  $x$  ( $1 \leq n, m, x \leq 10^9$ ),
- ◆ Dòng thứ 2 chứa số nguyên  $q$  ( $1 \leq q \leq 10^4$ ),
- ◆ Mỗi dòng trong  $q$  dòng sau chứa 5 số nguyên xác định một truy vấn.

**Kết quả:** Đưa ra file văn bản POWERTAB.OUT, kết quả mỗi truy vấn đưa ra trên một dòng dưới dạng số nguyên.

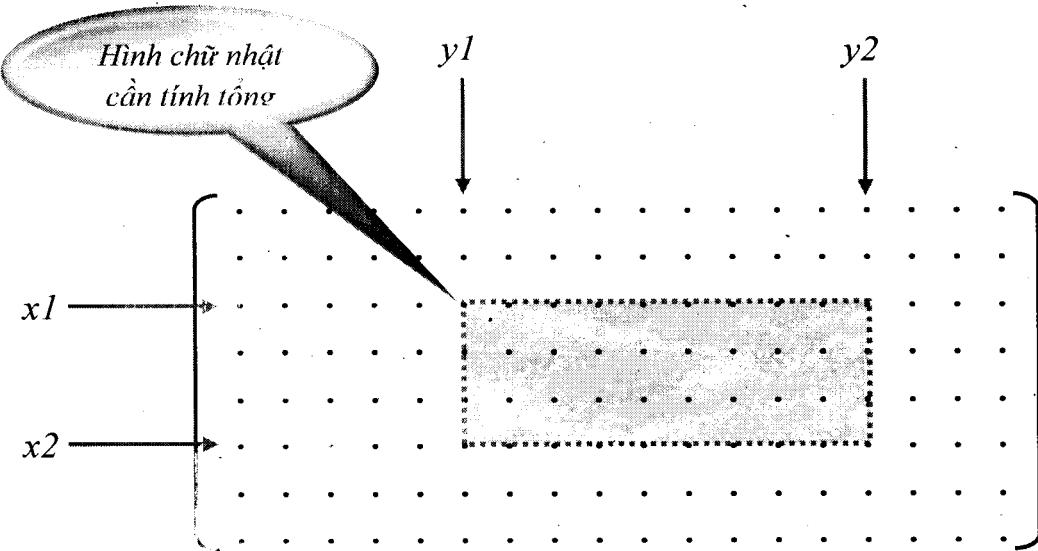
**Ví dụ:**

POWERTAB.INP		
1	10	2
5		
1	1	1
1	1	2
1	1	5
1	1	2
1	1	2

POWERTAB.OUT		
2		
6		
62		
28		
12		



VP09 K2\_R201403 E



Gọi tổng các phần tử cần tính là t, ta có:

$$\begin{aligned}
 t = & x^{(x_1-1)*m+y_1} + x^{(x_1-1)*m+y_1+1} + \dots + x^{(x_1-1)*m+y_2} + \\
 & x^{x_1*m+y_1} + x^{x_1*m+y_1+1} + \dots + x^{x_1*m+y_2} + \\
 & x^{(x_1+1)*m+y_1} + x^{(x_1+1)*m+y_1+1} + \dots + x^{(x_1+1)*m+y_2} + \\
 & \dots \dots \dots \\
 & x^{(x_2-1)*m+y_1} + x^{(x_2-1)*m+y_1+1} + \dots + x^{(x_2-1)*m+y_2} +
 \end{aligned}$$

### *Giải thuật*

Đặt  $k = (x_1-1)*m+y_1$  và nhóm thừa số chung, ta có:

$$\begin{aligned}
 t = & x^k \times (1 + x + x^2 + \dots + x^{y_2-y_1}) + \\
 & x^{k+m} \times (1 + x + x^2 + \dots + x^{y_2-y_1}) + \\
 & \dots \dots \dots + \\
 & x^{k+(x_2-x_1)m} \times (1 + x + x^2 + \dots + x^{y_2-y_1}) \\
 t = & x^k \times (1 + x^m + x^{2m} + \dots + x^{(x_2-x_1)m}) \times (1 + x + x^2 + \dots + x^{y_2-y_1})
 \end{aligned}$$

Đặt  $z = x^m$ , có

$$t = x^k \times \frac{z^{x_2-x_1+1}-1}{z-1} \times \frac{x^{y_2-y_1+1}-1}{x-1} = a \times b \times c,$$

$$\text{trong đó } a = x^k, b = \frac{z^{x_2-x_1+1}-1}{z-1}, c = \frac{x^{y_2-y_1+1}-1}{x-1}$$

Ta cần tính  $\text{ans} = t \bmod p = ((a \bmod p) \times (b \bmod p) \times (c \bmod p)) \bmod p$ .

a) Tính  $a \bmod p$ : áp dụng kỹ thuật nhân Ai Cập (xem giải thuật bài VJ18. Sân vận động),

b) Tính  $b \bmod p$ :  $z = x^m$  có thể rất lớn, vì vậy trong công thức tính cần thay  $z$  bằng  $z^m \bmod p (= u)$ .

Vì  $u^{x^2-x_1+1}-1$  chia hết cho  $u-1$  nên ta có:

$$b \bmod p = ((u^{x^2-x_1+1}-1) \bmod (p \times (u-1))) \bmod (u-1),$$

c) Tính  $c \bmod p$ : tương tự như cách tính  $b \bmod p$ :

$$c \bmod p = ((x^{y^2-y_1+1}-1) \bmod (p \times (x-1))) \bmod (x-1).$$

Lưu ý:

- ◆ Cần xử lý riêng các trường hợp  $x \bmod p = 1$  và  $u \bmod p = 1$ ,
- ◆ Vì  $p \times (u-1)$  và  $p \times (x-1)$  có thể lớn hơn hoặc bằng  $2^{32}$  nên cần phải áp dụng kỹ thuật lưu trữ và xử lý số lớn dạng 16 bytes.

Hàm `uint64_t get_md_t(uint64_t a,uint64_t b,uint64_t y)` phục vụ tính  $a \times b \bmod y$ , trong đó  $a, b, y$  là các số nguyên dương 8 bytes.

```
uint64_t get_md_t(uint64_t a,uint64_t b,uint64_t y)
{
    uint64_t c0,c1,b0,b1,d[4],t,t1;
    c1=a/c32; c0=a%c32; b0=b/c32;b1=b%c32;
    t=c0*b0;t1=t/c32;d[0]=t%c32;
    t=(c0*b1+c1*b0)+t1;
    t1=t/c32; d[1]=t%c32;
    t=c1*b1+t1;
    d[2]=t%c32; d[3]=t/c32;
    t=(d[3]<<c1+d[2])%y;
    for(int i=1;i>=0;--i)
        for(int j=23;j>=0;j-=4)
            {t1=(d[i]>>j & 15); t=(t<<4)|t1;
             t%=y;
            }
    return(t);
}
```

## Chương trình:

```
#include <fstream>
#include <ctime>
using namespace std;
ifstream fi ("Powertab.inp");
ofstream fo ("Powertab.out");
uint64_t n,m,x,p,x1,x2,y1,y2,q,u,v,z;
uint64_t r1,r2,r3,res,c32;

uint64_t get_md_t(uint64_t a,uint64_t b,uint64_t y)
{uint64_t c0,c1,b0,b1,d[4],t,t1;
 c1=a/c32; c0=a%c32; b0=b/c32;b1=b%c32;
 t=c0*b0;t1=t/c32;d[0]=t%c32;
 t=(c0*b1+c1*b0)+t1;
 t1=t/c32; d[1]=t%c32;
 t=c1*b1+t1;
 d[2]=t%c32; d[3]=t/c32;
 t=(d[3]<<32 | d[2])%y;
 for(int i=1;i>=0;--i)
    for(int j=28;j>=0;j-=4)
        {t1=(d[i]>>j & 15); c=(t<<4)|t1;
         t%=y;
        }
    return(t);
}

uint64_t fx(uint64_t a,uint64_t b,uint64_t c)
{uint64_t tw,tg=i;
 tw=a;
 while(b>0)
 /*
 if(b&1) tg=(tg*tw)%c;
 tw=(tw*tw)%c;
 */
 if(b&1) tg=get_md_t(tg,tw,c),
 tw=get_md_c(tw,tw,c);
 b>>=1;
}
return(tg);
}

void get_r()
{uint64_t k,r1,r2,r3;
 int64_t t;
 k=(x1-1)*m+y1;
 r1=fx(z,k,p);
 v=p*(z-1); t=fx(z,y2-y1+1,v)-1; r2=t/(z-1);
 u=fx(z,m,p); v=p*(u-1);
 if(u==1) r3=(x2-x1+1)%p;
 else r3=fx(u,x2-x1+1,v)-1; r3=t/(u-1);
 t=get_md_t(r1,r2,p); t=get_md_t(t,r3,p); res=t;
}

int main()
{clock_t aa=clock();
 fi>>n>>m>>x;
 fi>>q; c32=(uint64_t)1<<32;
 for(int i=1; i<= q; i++)
 {fi>>x1>>x2>>y1>>y2>>p;
 z=x%p;
```

```
if(z==1) res=(x2-x1+1)*(y2-y1+1)%p;else get_r();
fo<<res<<'\n';
}

clock_t bb=clock();
fo<<"Time: "<<(double)(bb-aa)/1000<<" sec";

}
```

## 16L. CƠ SỐ

Tên chương trình: BASE.???

Nội dung của giờ Tin học hôm nay là đổi một số nguyên dương  $x$  ở hệ 10 sang dạng biểu diễn ở cơ số **b**. Thầy giáo viết trên bảng số  $x$  và lần lượt gọi học sinh đọc kết quả biểu diễn ở các cơ số 2, 3, 4, ... Steve bắt đầu thấy buồn chán và không chú ý nghe lầm. Bỗng nhiên một kết quả lạ lùng vang lên làm Steve ngỡ ngàng: các chữ số trong dạng biểu diễn mới đều giống nhau! Chuông reo báo hết giờ. Steve quyết định về nhà phải tìm được cơ số nhỏ nhất cho dạng biểu diễn  $x$  chỉ bằng các chữ số giống nhau.

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

**Kết quả:** Đưa ra file văn bản BASE.OUT một số nguyên – cơ số nhỏ nhất tìm được.

**Ví dụ:**

BASE.INP	BASE.OUT
219	8



## *Giải thuật*

Nhận xét:  $\forall x$  bài toán luôn có nghiệm, ví dụ  $x$  có dạng biểu diễn 11 với cơ số  $x-1$ .

Đang có sẵn dạng biểu diễn nhị phân  $\rightarrow$  xét riêng trường hợp cơ số 2:

```
t=0;r=0;
for(int i=62;i>=0;--i) if((x>>i & 1) == 1){t1=i; break;}
for(int64_t i=t1;i>=0;--i)if((x>>i & 1) == 0) {t=1; break;}
if(t==0) r=2;
```

Các trường hợp còn lại:

$$x = p^*(k^m + k^{m-1} + \dots + k + 1) \quad (*)$$

Với  $m > 1$ :  $k < \sqrt{x} \rightarrow$  duyệt từ  $3 \div \sqrt{x}$  tìm  $k$  thỏa mãn (\*) và có  $p < k$ , xét riêng trường hợp  $x =$

```
void xly1()
{if(x==2){r=3;return;}
 for(int64_t i=3;i<=t2;++i)
 {t=1;
 while(t<=x) {t0=t;t=t*i+1;}
 if(x%t0==0 && x/t0<i){r=i;break;}
 }
}
```

2.

Với  $k \geq \sqrt{x} \rightarrow$  tìm  $p < \sqrt{x}$  cho min  $k$  thỏa mãn  $x = p^*(k+1)$ .

```
void xly2()
{t=x-1;i0=1;
 for(int64_t i=2;i<=t2;++i)
 if(x%i==0){t0=x/i-1;if (i<t0 && t0<t){t=t0;i0=i;}}
 r=t;
}
```

```

#include <fstream>
#include <cmath>
#include <ctime>
using namespace std;
int64_t x,r,t,t1,t2,t0,i0;
double tg;
ifstream fi ("Base.inp");
ofstream fo ('Base.out');

void xly1()
{for(int64_t i=3;i<=t2;++i)
 {t=1;
  while(t<=x){t0=t; t=t*i+1;}
  if(t<=x &&x%t==0){r=i;break;}
 }
}

void xly2()
{t=x-1;i0=1;
 for(int64_t i=2;i<=t2;++i)
  if(x%i==0){t0=x/i-1;if (i<t0 && t0< t){t=t0;i0=i;}}
  r=t;
}

int main()
{fi>>x;
 tg=sqrt(x); t2=tg+1;
 for(int i=62, l>=0;--i, if((x>>i & 1) == 1){t1=i; break;}
 t=0;r=0; if(x==2)r=3;
 else {for(int64_t i=t1;i>=0;--i)if((x>>i & 1) == 0) {t=1; break;}
 if(t==0) r=2;
 if(r==0) xly1();
 if(r==0) xly2();
 }
 fo<<r;
}

```

## VL39. CÂN BẰNG

Tên chương trình: BALANCE.???

Tàu chở gạo xuất khẩu có hai hầm chứa ở đầu và đuôi tàu. Để đảm bảo sự cân bằng tốt nhất cho các tàu chở hàng vượt đại dương người ta xếp vào hai hầm chứa số lượng bao gạo là như nhau. Nhưng ở phút chót, một số hợp đồng xuất khẩu bị hủy. Kết quả là ở một hầm được xếp  $a$  bao gạo, còn hầm kia –  $b$  bao.

Hệ thống điều khiển tự động trên tàu chỉ có khả năng chuyển một số bao gạo từ hầm nhiều hơn sang hầm ít hơn để làm tăng gấp đôi số bao gạo ở hầm trước đó có ít bao hơn. Ví dụ, một hầm hiện chứa 3 bao và hầm kia – 5 bao. Hệ thống tự động sẽ chuyển 3 bao từ hầm nhiều hơn sang hầm ít hơn và ta có kết quả số bao gạo ở mỗi hầm sẽ là 6 và 2. Bước xử lý tiếp theo sẽ cho kết quả 4 – 4. Mỗi hầm cùng chứa một số bao như nhau.

Với  $a$  và  $b$  cho trước hãy xác định có thể đưa số bao gạo ở mỗi hầm về giống nhau hay không.

**Dữ liệu:** Vào từ file văn bản BALANCE.INP:

- ◆ Dòng đầu tiên chứa số nguyên  $n$  – số lượng tests ( $1 \leq n \leq 10^5$ ),
- ◆ Mỗi dòng trong số  $n$  dòng sau chứa 2 số nguyên  $a$  và  $b$  ( $1 \leq a, b \leq 10^9$ ).

**Kết quả:** Đưa ra file văn bản BALANCE.OUT, với mỗi test đưa ra trên một dòng thông báo **YES** hoặc **NO** cho biết xếp được hay không xếp được.

**Ví dụ:**

BALANCE.INP	BALANCE.OUT
3	YES
2 6	NO
1 5	YES
4 4	YES

vL39 IO 20130921 A

```
#include <iostream>
using namespace std;
ifstream fi ("balance.inp");
ofstream fo ("balance.out");
int n,a,b,x,y,z;

int main()
{fi>>n;
 for(int i=1;i<=n;++i)
 {fi>>a>>b;
 if(a>b) {x=a;a=b;b=x;}
 x=a+b; y=b%a; z=x/a;
 if(((x&(-x))==x) || (y==0) && ((z&(-z))==z)) fo<<"YES"<<'\n';
 else fo<<"NO"<<'\n';
 }
}
```

Kiểm tra số thuộc dạng  $2^k$



## VO48. BÓNG CHUYỀN BÃI BIỂN

Tên chương trình: VOLLEY.???

Các bãi biển vùng nhiệt đới thu hút đông đảo khách du lịch quanh năm bởi vì khi đến người ta có thể tắm biển, phơi nắng, thưởng thức các đặc sản biển và được tham gia các hoạt động thể thao mà mình ưa thích. Môn bóng chuyền bãi biển thường thu hút nhiều người nhất, cả người tham gia và người xem. Có  $n$  đội đăng ký thi đấu, đội thứ  $i$  có chỉ số chuyên môn là  $a_i$ ,  $i = 1 \dots n$ . Không có 2 đội nào có cùng chỉ số chuyên môn. Theo danh sách đăng ký hai đội đầu tiên trong danh sách sẽ ra thi đấu với nhau. Đội nào thắng ở lại sân, còn đội thua – quay về xếp hàng ở cuối danh sách. Đội đứng đầu danh sách hiện tại sẽ vào sân. Sau mỗi trận đấu đội thua lại ra sân và xuống xếp hàng ở cuối. Các nhà chuyên môn đánh giá khá chính xác: đội có chỉ số chuyên môn cao hơn luôn thắng!

Dù là sân chơi nghiệp dư nhưng công tác trọng tài cũng được quan tâm chu đáo. Steve tham gia ban trọng tài và theo kết quả bốc thăm sẽ cầm còi ở  $m$  trận có thứ tự là  $b_1, b_2, \dots, b_m$ .

Hãy xác định cặp đội thi đấu ở mỗi trận do Steve cầm còi và chỉ số chuyên môn của các đội đó.

**Dữ liệu:** Vào từ file văn bản VOLLEY.INP:

- ◆ Dòng đầu tiên chứa số nguyên  $n$  ( $2 \leq n \leq 10^5$ ),
- ◆ Dòng thứ 2 chứa  $n$  số nguyên  $a_1, a_2, \dots, a_n$  ( $1 \leq a_i \leq n$ ,  $a_i \neq a_j$  với  $i \neq j$ ,  $i, j = 1 \dots n$ ),
- ◆ Dòng thứ 3 chứa số nguyên  $m$  ( $1 \leq m \leq 10^5$ ),
- ◆ Dòng thứ  $j$  trong  $m$  dòng sau chứa số nguyên  $b_j$  ( $1 \leq b_j \leq 10^{18}$ ).

**Kết quả:** Đưa ra file văn bản VOLLEY.OUT các cặp thi đấu xác định được và các chỉ số chuyên môn tương ứng, mỗi nhóm số trên một dòng, cặp số thứ tự tách cặp chỉ số chuyên môn bởi xâu ” : “, đội có thứ tự nhỏ hơn đứng trước.

**Ví dụ:**

VOLLEY.INP
4
2 1 4 3
3
1
5
2

VOLLEY.OUT
1 2 : 2 1
1 3 : 2 4
1 3 : 2 4



```

#include <fstream>
#include <ctime>
using namespace std;
ifstream fi ("volley.inp");
ofstream fo ("volley.out");
int t,t1,t2,n,m,k,ax,c[100001],aa[100001];
pair<int,int> a[100001],b[100001];

int main()
{clock_t ab=clock();
 fi>>n; ax=0;
 for(int i=0;i<n;++i)
 {fi>>t; if(t>ax){ax=t;k=i;}
  a[i]=make_pair(t,i+1); aa[i]=t;
 }
 for(int i=k;i<n;++i)c[i-k]=a[i].second;
 for(int i=1;i<=k;++i)
 {b[i-1].first=a[0].second; b[i-1].second=a[i].second;
  if(a[i].first<a[0].first)c[k+i]=a[i].second;
  else {c[k+i-1]=a[0].second;a[0]=a[i];}
 }
 fi>>m;
 for(int i=1;i<=m;++i)
 {fi>>t;--t;
  if(t<k){t1=b[t].first;t2=b[t].second;}
  else {t-=k;t%=(n-1); t1=aa[c[0]]; t2=aa[c[t+1]];}
  if(t1>t2){t=t1;t1=t2;t2=t;}
  fo<<t1<<' '<<t2<<" : "<<aa[t1-1]<<' '<<aa[t2-1]<<'\n';
 }
 clock_t ae=clock();
 fo<<"Time: "<<(double)(ae-ab)/1000<<" sec";
}

```

## VO40. ĐƯỜNG SẮT TRÊN CAO

Tên chương trình: TRAIN.???

Thành phố đang xây dựng tuyến đường sắt trên cao phục vụ giao thông nội đô. Tuyến đường sẽ có  $k+1$  ga đánh số từ 0 đến  $k$ . Tàu chạy suốt ngày đêm, từ ga 0 đến ga  $k$  và quay lại. Thời gian đi từ một ga tới ga kế tiếp là 1 phút, thời gian dừng ở mỗi ga là không đáng kể. Hệ thống giao thông này không những nhanh, chuẩn xác về thời gian mà còn là một phương tiện tuyệt vời để ngắm thành phố.

Hội đồng thành phố quyết định sẽ tặng cho tất cả các học sinh đạt giải thưởng thành phố mỗi em một vé đi tàu miễn phí trong năm học.

Minh rất háo hức với thông báo này và quyết tâm phải giành được một tấm vé của thành phố. Phần lớn thời gian rảnh trong ngày Minh đều dùng để rèn luyện kỹ năng Tin học, chuẩn bị cho kỳ thi Học sinh giỏi Tin học sắp tới. Còn ban đêm là thời gian của những giấc mơ đẹp. Minh thấy mình bước lên tàu ở ga số 0, ngồi cạnh cửa sổ say sưa ngắm nhìn quang cảnh thành phố từ trên cao. Thời gian trôi đi khá nhanh. Đồng hồ cho biết Minh đã ngồi trên tàu  $t$  phút và Minh quyết định xuống tàu . . .

Chuông đồng hồ vang lên, Minh tỉnh giấc, vội vàng đi đánh răng, rửa mặt, chuẩn bị đi học. Trên đường tới trường Minh vẫn nghĩ về giấc mơ đêm qua và không thể nhớ được mình đã xuống ở ga số mấy.

Hãy xác định ga mà Minh đã xuống trong giấc mơ sắp thành hiện thực của mình.

Dữ liệu: Vào từ file văn bản TRAIN.INP gồm một dòng chứa 2 số nguyên  $k$  và  $t$  ( $0 < k, t \leq 10^9$ ). Các số ghi cách nhau một dấu cách.

Kết quả: Đưa ra file văn bản TRAIN.OUT một số nguyên – ga xác định được.

Ví dụ:

TRAIN.INP
5 8

TRAIN.OUT
2



*Giải thuật và chương trình*

```
#include <iostream>
using namespace std;
ifstream fi ("Train.inp");
ofstream fo ("Train.out");
int n,k,t,u,v,r;

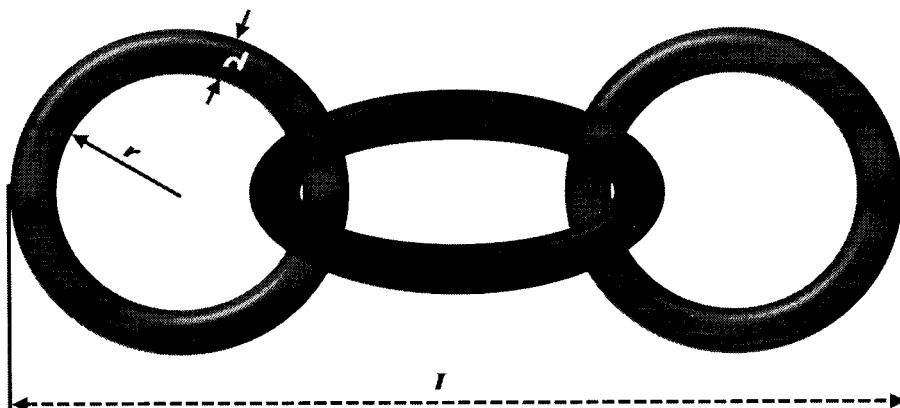
int main()
{
    fi>>k>>t;
    u=t/k;
    v=t%k;
    if((u&1)==1) r=k-v; else r=v;
    fo<<r;
}
```

## VO41. DÂY XÍCH

## Tên chương trình: CHAIN.???

Người ta dùng dây thép tròn độ dày (đường kính thiết diện ngang)  $d$  làm  $n$  vòng tròn bán kính vòng tròn trong là  $r$ , móc nối với nhau thành một dây xích, mỗi vòng tròn là một mắt xích. Hình 1 nêu trường hợp dây có 3 mắt xích. Nếu dây xích có nhiều hơn một mắt xích thì tồn tại hai vòng tròn, mỗi vòng chỉ nối với đúng một vòng tròn khác, đó là các mắt xích đầu và cuối. Cầm 2 mắt xích đầu và cuối, kéo căng ra, ta có dây xích độ dài  $L$ .

Cho  $d$ ,  $r$  và  $n$ . Hãy tính độ dài  $L$  của dây xích.



Hình 1. Dây 3 mắt

**Dữ liệu:** Vào từ file văn bản CHAIN.INP gồm một dòng chứa 3 số nguyên  $d$ ,  $r$  và  $n$  ( $1 \leq d < r \leq 100$ ,  $1 \leq n \leq 10^9$ ). Các số ghi cách nhau một dấu cách.

**Kết quả:** Đưa ra file văn bản CHAIN.OUT một số nguyên – độ dài  $L$  tìm được.

**Ví dụ:**

CHAIN.INP
2 10 3

CHAIN.OUT
64



### Giải thuật và chương trình

```
#include <iostream>
using namespace std;
ifstream fi ("Chain.inp");
ofstream fo ("Chain.out");
int64_t m,n,d,r,ans;

int main()
{
    fi>>d>>r>>n;
    ans=2*(n*r+d);
    fo<<ans;
}
```

## VO44. LIÊN HOAN PHIM

Tên chương trình: CINEMA.???

Liên hoan phim năm nay có  $n$  phim được vào diện xét chọn. Có 2 giải chính là giải đạo diễn hay nhất và giải phim có nội dung hay nhất, mỗi giải được trao cho một phim. Hai giải này được trao cho 2 phim khác nhau.

Kết quả thăm dò ý kiến khán giả và các nhà bình luận điện ảnh xác định được mức độ hân hoan của công chúng khi một phim được giải này hay giải khác hoặc không được giải nào, trong đó  $a_i$  là mức độ hân hoan khi phim thứ  $i$  không được giải nào,  $b_i$  – mức độ hân hoan khi phim được trao giải đạo diễn hay nhất và  $c_i$  – khi phim được giải nội dung hay nhất,  $i = 1 \dots n$ .

Hãy xác định tổng độ hân hoan lớn nhất mà cuộc bầu chọn có thể mang lại cho công chúng và chỉ ra phim nào sẽ được giải gì trong trường hợp này.

**Dữ liệu:** Vào từ file văn bản CINEMA.INP:

- ◆ Dòng đầu tiên chứa số nguyên  $n$  ( $2 \leq n \leq 10^5$ ),
- ◆ Dòng thứ  $i$  trong  $n$  dòng sau chứa 3 số nguyên  $a_i$ ,  $b_i$  và  $c_i$  ( $1 \leq a_i, b_i, c_i \leq 10^9$ ).

**Kết quả:** Đưa ra file văn bản CINEMA.OUT:

- ◆ Dòng thứ nhất chứa một số nguyên – tổng độ hân hoan lớn nhất có thể đạt được,
- ◆ Dòng thứ 2 chứa 2 số nguyên xác định phim được giải đạo diễn hay nhất và phim được giải nội dung hay nhất. Nếu tồn tại nhiều cặp chỉ số cùng thỏa mãn thì đưa ra cặp có thứ tự từ điển nhỏ nhất.

**Ví dụ:**

CINEMA.INP
3
3 6 9
1 5 7
1 3 9

CINEMA.OUT
17
2 3



VO44 ROI20140410 E

## **Giải thuật**

Yêu cầu của bài toán là phải tìm  $i \neq j$  sao cho

$$S = b_i + c_j + \sum_{k \neq i, k \neq j} a_k$$

là lớn nhất.

$$\text{Để dàng thấy rằng } S = (b_i - a_i) + (c_j - a_j) + \sum_{k=1}^n a_k, \quad i \neq j.$$

Việc cực đại hóa  $S$  đưa về việc cực đại hóa  $b_i - a_i + c_j - a_j$ .

Đặt  $ba_i = b_i - a_i$  và  $ca_i = c_i - a_i$ ,  $i = 1 \div n$ , dễ dàng xác định:

$$ba_{ib1} = \max_{1 \leq i \leq n} \{ba_i\}, \quad ba_{ib2} = \max_{1 \leq i \leq n, i \neq ib1} \{ba_i\},$$

$$ca_{ic1} = \max_{1 \leq i \leq n} \{ca_i\}, \quad ca_{ic2} = \max_{1 \leq i \leq n, i \neq ic1} \{ca_i\},$$

Nếu  $ib1 \neq ic1$  thì đó chính là các phim phải tìm.

Nếu  $ib1 = ic1$  thì phải tìm  $\max\{ba_{ib1} + ca_{ic2}, ba_{ib2} + ca_{ic1}\}$ , từ đó xác định được các phim cần tìm và tổng độ hân hoan lớn nhất có thể đạt được. Lưu ý việc xác định cặp chỉ số nhỏ nhất!

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

```

#include <fstream>
#include <ctime>
using namespace std;
ifstream fi ("Cinema.inp");
ofstream fo ("Cinema.out");
int64_t
a,b,c,bx1,bx2,cx1,cx2,sa,ans,mx,ba[100001],ca[100001],r,r1,r2;
int n,ib1,ib2,ic1,ic2;

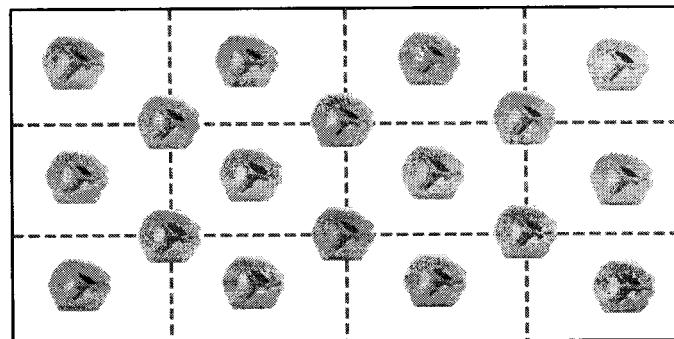
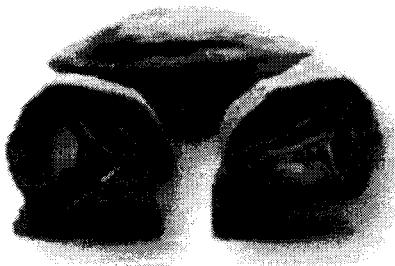
int main()
{//clock_t aa=clock();
 fi>>n;sa=0; mx=-2000000; mx*=1000;
 for(int i=1;i<=n;++i)
 {fi>>a>>b>>c; sa+=a;ba[i]=b-a; ca[i]=c-a; }
 bx1=mx; bx2=mx;cx1=mx;cx2=mx;ib1=0;ic1=0;
 for(int i=1;i<=n;++i)
 {if(bx1<ba[i]) {bx1=ba[i];ib1=i;}
 if(cx1<ca[i]) {cx1=ca[i];ic1=i;}
 }
 if(ib1==ic1)
 {cx2=mx;for(int i=1;i<=n;++i)if(i!=ic1 &&
 cx2<ca[i]) {cx2=ca[i];ic2=i;}
 r1=bx1+cx2;
 bx2=mx;for(int i=1;i<=n;++i)if(i!=ic1 &&
 bx2<ba[i]) {bx2=ba[i];ib2=i;}
 r2=bx2+cx1;
 if(r1>=r2){r=r1;ic1=ic2;} else {r=r2;ib1=ib2;}
 ans=sa+r;
 }
 else ans=sa+bx1+cx1;
 fo<<ans<<'\n'<<ib1<<' '<<ic1;
 /* clock_t bb=clock();
 fo<<"\nTime: "<<(double)(bb-aa)/1000<<" sec."; */
}

```

## VM08. TRÁM ĐEN

Tên chương trình: CANARIUM.???

Hiên, một huyện miền núi phía tây Quảng Nam cũng có trám, tuy không nhiều như ở Bắc Kạn. Các bạn Sinh viên Tình nguyện Mùa hè xanh thấy hột trám vương vãi quanh trường khá nhiều, đã nảy ra sáng kiến “trám hóa” sân trường. Có  $k$  hạt trám được thu thập về. Sân trường có hình chữ nhật. Bằng  $m$  đường cách đều nhau song song với một cạnh của sân trường và  $n$  đường cách đều nhau song song với cạnh kia của sân trường toàn bộ sân được chia thành các hình chữ nhật con giống nhau ( $1 \leq m \leq n$ ). Các hột trám sẽ được chặt đôi. Sau khi ăn nhén bên trong học sinh sẽ đóng nửa hạt này xuống sân tại các điểm giao nhau giữa các được kẻ và ở tâm điểm các hình chữ nhật con. Tại mỗi điểm chỉ đóng nửa hạt trám. Để không lãng phí số hạt trám đã thu nhặt và hạt trám được đóng phân bố đều trên sân các bạn sinh viên quyết định chọn  $m$  và  $n$  sao cho số hạt trám sẽ được dùng hết và hiệu  $n-m$  là nhỏ nhất.



Với  $k$  cho trước hãy xác định  $m$  và  $n$ . Nếu không tồn tại  $m$  và  $n$  thỏa mãn thì đưa ra hai số -1.

**Dữ liệu:** Vào từ file văn bản CANARIUM.INP:

- ↳ Dòng đầu tiên chứa số nguyên  $t$  – số tests ( $1 \leq t \leq 20$ ),
- ↳ Mỗi test cho trên một dòng chứa một số nguyên dương  $k$  ( $1 \leq k \leq 10^{12}$ ).

**Kết quả:** Đưa ra file văn bản CANARIUM.OUT, kết quả mỗi test đưa ra trên một dòng gồm 2 số nguyên  $m$  và  $n$  (có thể là -1 -1), các số cách nhau một dấu cách.

**Ví dụ:**

CANARIUM .INP
2
9
6

CANARIUM .OUT
2 3
-1 -1



*Giải thuật:*

*Mô hình toán học:*

- ◆ Kẻ m, n đường,
- ◆ Đóng trám vào các giao đường kẻ và tâm các hình chữ nhật con.
- ◆ Hãy xác định m, n thỏa m:
  - Sử dụng hết hạt trám,
  - $1 \leq m \leq n$ ,
  - $n - m \rightarrow \min$
- ◆ Số trám đóng trên các nút:  $m^*n$ ,
- ◆ Số trám đóng ở các hình chữ nhật con:  $(m+1)^*(n+1)$ .
- ◆ Có phương trình:  $(m+1)^*(n+1) + m + n = 2*k$ ,
- ◆  $2m^*n + m + n = 2k - 1$
- ◆  $4m^*n + 2m + 2n = 4k - 2$
- ◆  $(2m+1)^*(2n+1) = 4k - 1$
- ◆ Đặt  $x = 2m + 1$ ,  $y = 2n + 1$ , ta có  $x*y = 4k - 1$ .
- ◆ Do  $x \leq y \rightarrow x \leq \sqrt{4*k - 1}$
- ◆  $k \leq 10^{12} \rightarrow y \leq 10^6$ .
- ◆ Kiểm tra mọi y lẻ trong phạm vi  $[\sqrt{4*k - 1} \div 1]$ , dừng khi tìm được y đầu tiên.

```
#include <fstream>
#include <cmath>
using namespace std;
ifstream fi ("CANARIUM.INP");
ofstream fo ("CANARIUM.OUT");
int n,m,u,v,p,q;
int64_t t,k;

int main()
{fi>>q;
for(int j=1;j<=q;++j)
{fi>>k; t=k*4-1;
p=(int)(sqrt(t)+1);
if((p&1)==0)--p;v=0;
for(int i=p;i>=1;i-=2) if(t % i ==0) {v=i;break;}
if (v==1) {m=-1; n=-1;} else {m=(v-1)/2; u=(int)(t/v); n=(u-1)/2;}
fo<<m<<" "<<n<<'\n';
}
}
```

## VO32. TUYẾN BAY

Tên chương trình: FLY.???

Steve sinh hoạt ở câu lạc bộ mô hình máy bay. Hôm nay là ngày kiểm tra kỹ thuật lái máy bay mô hình. Chương trình bay tự động đã được cài sẵn trong máy bay, tại thời điểm  $i$  máy bay sẽ bay ở độ cao  $h_i$  mét,  $i = 1 \div n$ .

Steve phải dùng thiết bị cầm tay điều khiển máy bay bay ở một độ cao ổn định luôn là  $p$ . Do điều kiện địa hình và khoảng cách từ nơi điều khiển đến máy bay nên ở thời điểm  $i$  cứ thay đổi mỗi  $m$  đơn vị độ cao (lên hoặc xuống) so với giá trị định sẵn phải mất  $a_i$  đơn vị năng lượng. Nhiệm vụ của Steve là đảm bảo độ cao ổn định của máy bay với chi phí năng lượng nhỏ nhất.

**Dữ liệu:** Vào từ file văn bản FLY.INP:

- ◆ Dòng đầu tiên chứa số nguyên  $n$  ( $1 \leq n \leq 10^5$ ),
- ◆ Dòng thứ 2 chứa  $n$  số nguyên  $h_1, h_2, \dots, h_n$  ( $1 \leq h_i \leq 10^6$ ,  $i = 1 \div n$ ),
- ◆ Dòng thứ 3 chứa  $n$  số nguyên  $a_1, a_2, \dots, a_n$  ( $1 \leq a_i \leq 10^6$ ,  $i = 1 \div n$ ).

**Kết quả:** Đưa ra file văn bản FLY.OUT trên một dòng số nguyên  $p$  và chi phí năng lượng cần thiết để điều khiển. Nếu tồn tại nhiều độ cao khác nhau cùng thỏa mãn thì đưa ra độ cao nhỏ nhất.

**Ví dụ:**

FLY.INP
6
6 7 8 8 7 7
10 6 3 1 1 4

FLY.OUT
7 14



VO32.lm20140322 A

### **Giải thuật:**

Trình tự xuất hiện độ cao và chi phí thay đổi độ cao không đóng vai trò quan trọng, vì vậy có thể sắp xếp  $\{(h_i, a_i)\}$  theo trình tự tăng dần của các độ cao  $h_i$ .

Gọi  $f(x)$  là chi phí duy trì độ cao không đổi  $x$ . Giả thiết  $h_i \leq x \leq h_{i+1}$ .

Ta có:

$$\begin{aligned} f(x) &= \sum_{j=1}^i (x - h_j) \times a_j + \sum_{j=i+1}^n (h_j - x) \times a_j \\ &= x \times \left( \sum_{j=1}^i a_j - \sum_{j=i+1}^n a_j \right) - \sum_{j=1}^i h_j a_j + \sum_{j=i+1}^n h_j a_j \end{aligned}$$

Đặt  $u_1 = \sum_{j=1}^i a_j$ ,  $u_2 = \sum_{j=i+1}^n a_j$ ,  $v_2 = \sum_{j=i+1}^n h_j a_j$ ,  $v_1 = \sum_{j=1}^i h_j a_j$ , ta có

$$f(x) = h \times (u_1 - u_2) + v_1 - v_2$$

Hàm  $f(x)$  đơn điệu liên tục từng khúc vì vậy giá trị nhỏ nhất toàn bộ sẽ đạt được tại một điểm đầu hay cuối của một khoảng nào đó, tức là  $\exists i \in \mathbb{N} \text{ sao cho } f(h_{ix}) \leq f(x)$ .

Vấn đề còn lại là xác định giá trị đầu của  $u_1$ ,  $u_2$ ,  $v_1$ ,  $v_2$ , cách chỉnh lý các giá trị này khi chuyển từ  $h_i$  sang  $h_{i+1}$  và duyệt giá trị hàm  $f(x)$  với mọi  $x = h_i$ ,  $i = 1 \dots n$ .

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

```

#include <fstream>
using namespace std;
ifstream fi ("fly.inp");
ofstream fo ("fly.out");
int64_t u1=0,u2=0,v1=0,v2=0,ans,r,fx;
pair<int64_t,int64_t> a[100001];
int n;

int main()
{fi>>n;
 for(int i=1;i<=n;++i)fi>>a[i].first;
 for(int i=1;i<=n;++i)fi>>a[i].second;
 sort(a+1,a+n+1);
 for(int i=1;i<=n;++i){u2+=a[i].second; v1+=a[i].first*a[i].second;}
 hx=(int64_t)1000000000;fx*=fx;
 for(int i=1;i<=n;++i)
 {u1+=a[i].second; u2-=a[i].second;
  v1-=a[i].first*a[i].second; v2+=a[i].first*a[i].second;
  r=(u1-u2)*a[i].first+v1-v2; if(r<fx){fx=r; ans=a[i].first;}
 }
 fo<<ans<<" "<<fx;
}

```

## VN16. HÌNH CHỮ NHẬT

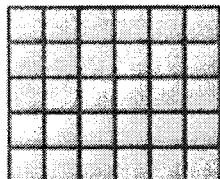
Tên chương trình: RECTANGLE.???

Xét các hình chữ nhật kích thước  $w \times h$ , trong đó  $w, h$  – nguyên và  $w > h$ .

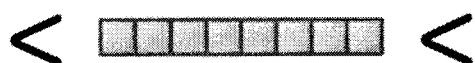
Một hình chữ nhật gọi là nhỏ hơn hình chữ nhật khác nếu thỏa mãn một trong 2 điều kiện:

- ◆ Có đường chéo ngắn hơn,

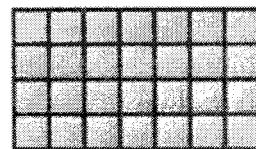
$$h = 5, w = 6$$



$$h = 1, w = 8$$



$$h = 4, w = 7$$



$$\text{Đường chéo} = \sqrt{61}$$

$$\text{Đường chéo} = \sqrt{65}$$

$$\text{Đường chéo} = \sqrt{65}$$

- ◆ Có đường chéo bằng nhau, nhưng có độ cao  $h$  nhỏ hơn.

Với hình chữ nhật cho trước hãy xác định kích thước hình chữ nhật nhỏ nhất lớn hơn hình đã cho.

**Dữ liệu:** Vào từ file văn bản RECTANGLE.INP, gồm nhiều tests, mỗi test cho trên một dòng chứa 2 số nguyên  $w$  và  $h$  ( $0 < w < h \leq 100$ ). Số lượng tests trong file không quá 100. Dữ liệu kết thúc bằng một dòng chứa 2 số 0.

**Kết quả:** Đưa ra file văn bản RECTANGLE.OUT, kết quả mỗi test đưa ra trên một dòng gồm 2 số nguyên  $a$  và  $b$  ( $a < b$ ).

**Ví dụ:**

RECTANGLE.INP
1 2
1 3
2 3
1 4
2 4
5 6
1 8
4 7
98 100
99 100
0 0

RECTANGLE.OUT
1 3
2 3
1 4
2 4
3 4
1 8
4 7
2 8
3 140
89 109

## *Giải thuật*

Xử lý hình chữ nhật  $x \times y$ ,  $x < y$ :  $c = x^2 + y^2$ .

Hình chữ nhật cần tìm:  $a \times b$ ,  $a < b$ .

Tồn tại:  $a < x$  hoặc  $b > y$ .

Chuẩn bị:  $r = 10^6$ .

*Tìm hình lớn hơn có  $a < x$ :*

- ◆ Với mọi  $i = 1 \div x-1$ :
- ◆ Tìm  $q$  nhỏ nhất thỏa mãn:  $q > y$ ,  $tg = i^2 + q^2 \geq c$ ,
- ◆ Nếu  $r > tg$ : lưu hình mới:  $a = i$ ,  $b = q$ ,  $r = tg$ ,
- ◆ Nếu  $r = c \rightarrow$  Đưa ra kết quả và **kết thúc**.

*Tìm hình có  $b > y$ :*

- ◆ Tương tự như trên, nhưng không kết thúc khi  $r = c$ !

*Lưu ý:* tồn tại cách tìm để kết thúc giải thuật khi gặp  $r = c$ .

```
#include <iostream>
using namespace std;
ifstream fi ("Rectangle.inp");
ofstream fo ("Rectangle.out");
int a,b,c,rx,ry,r,tx,ty,t2,tg,p,q,flg;
void xly()
{c=a*a+b*b;r=10000000; flg=0;
 for(int i=1;i<=a;++i)
 {tg=0;q=b;while(tg<c){++q; tg=i*i+q*q;}
 tx=i; ty=q; t2=tg;
 if(t2>c && r>t2){rx=tx;ry=ty;r=t2; if(r==c)break;}
 }
 if(r==c){fo<<rx<<" "<<ry<<endl; flg=1;return;}
 for(int i=1;i<=b;++i)
 { p=a; tg=0;while(tg<c){++p; tg=p*p+i*i;}
 tx=p;ty=i;t2=&
 if(tx<ty && (r>t2 || (r==t2 &&
 tx<rx))){rx=tx;ry=ty;r=t2;}
 }
 fo<<rx<<" "<<ry<<endl;
 return;
}
int main()
{fi>>a>>b;
 while(a>0)
 {xly();
 fi>>a>>b;
 }
}
```

## VJ46. TRÌNH TỰ

Tên chương trình: ORDER.???

Steve xây dựng được một chương trình nhận dạng nhanh dãy số có phải là không giảm hay không theo giải thuật riêng của mình. Để chuẩn bị trình bày kết quả trong hội nghị khoa học Steve tạo một chương trình tương tác với chương trình nhận dạng của mình. Chương trình tương tác làm việc với dãy  $n$  số nguyên  $a_1, a_2, \dots, a_n$  và xử lý  $m$  truy vấn, mỗi truy vấn thuộc một trong 2 loại với quy cách:

- ◆ Loại I: !  $k$   $x$  – Gán cho phần tử  $a_k$  giá trị nguyên  $x$ ,
- ◆ Loại II: ? – Gọi chương trình nhận dạng, kiểm tra xem dãy số có phải là không giảm hay không và đưa ra câu trả lời tương ứng là YES hoặc NO.

Hãy xác định các câu trả lời nhận được trong quá trình tương tác.

**Dữ liệu:** Vào từ file văn bản ORDER.INP:

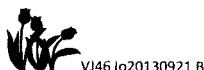
- ◆ Dòng đầu tiên chứa 2 số nguyên  $n$  và  $m$  ( $1 \leq n \leq 10^5$ ,  $0 \leq m \leq 2 \times 10^5$ ),
- ◆ Dòng thứ 2 chứa  $n$  số nguyên  $a_1, a_2, \dots, a_n$  ( $0 \leq a_i \leq 10^9$ ,  $i = 1 \div n$ ),
- ◆ Mỗi dòng trong  $m$  dòng sau chứa một truy vấn.

**Kết quả:** Đưa ra file văn bản ORDER.OUT các câu trả lời đối với truy vấn loại II. Mỗi câu trả lời đưa ra trên một dòng.

**Ví dụ:**

ORDER.INP
5 5
2 4 6 8 10
?
! 2 7
?
! 3 8
?

ORDER.OUT
YES
NO
YES



```
#include <fstream>
#include <iostream>
using namespace std;
int a[100002],n,m,k,x,v;
string s;
int main()
{freopen("order.inp","r",stdin);
 freopen("order.out","w",stdout);
 cin>>n>>m;
 for(int i=1;i<=n;++i)cin>>a[i];
 a[0]=-1000000001; a[n+1]=1000000001; v=0;
 for (int i=1;i<=n;++i) if(a[i]<a[i-1])++v;
 for (int i=1;i<=m;++i)
 {cin>>s;
 if(s=="?") {if(v==0) cout<<"YES"<<endl; else cout<<"NO"<<endl;}
 else
 {cin>>k>>x;
 v-= (a[k - 1] > a[k]) + (a[k] > a[k + 1]);
 a[k]=x;
 v+= (a[k - 1] > a[k]) + (a[k] > a[k + 1]);
 }
 }
}
```

## **2. Sơ đồ lặp và Quy hoạch động**

Với mỗi bài toán ứng dụng giải thuật quy hoạch động, ta phải trả lời rõ ràng, chính xác 6 câu hỏi:

- ➔ Tên và ý nghĩa các biến phục vụ sơ đồ lặp,
- ➔ Cách khai báo các biến đó,
- ➔ Sơ đồ (công thức) lặp chuyển từ một bước sang bước tiếp theo,
- ➔ Giá trị đầu của các biến tham gia tính lặp,
- ➔ Tham số điều khiển lặp: thay đổi từ đâu đến đâu,
- ➔ Kết quả: ở đâu và làm thế nào để dẫn xuất ra.

Các cách trả lời khác nhau sẽ dẫn đến những giải thuật khác nhau cả về cách thực hiện lẫn độ phức tạp.

Để thiêu hủy gà vịt ở các vùng có dịch cúm gia cầm người ta đã chế tạo được những container đặc biệt, nhốt gia cầm vào trong đó, phun một hỗn hợp hóa chất đặc chủng và đóng container. Các con vật trong đó sẽ bị giết và phân hủy thành một hỗn hợp ít ảnh hưởng tới môi trường. Công nghệ mới đang trong giai đoạn thử nghiệm nên chưa hoàn thiện lắm, cá container chuyên dụng chưa được chế tạo, vì vậy phải dùng tạm các container thông thường. Hỗn hợp hóa chất phải được phun trực tiếp vào container trước khi đóng cửa, nếu không hóa chất sẽ bị phát tán và không đảm bảo nồng độ cần thiết cho phát ứng dây chuyền phân hủy sinh học.

Lần thử nghiệm đầu tiên xe chở hóa chất sẽ đi trên một lộ trình có  $n$  điểm, mỗi điểm đặt một container. Ở điểm thứ  $i$  có  $a_i$  con gia cầm và thời điểm thu gom đủ tất cả gà vịt nói trên vào container là  $t_i$ ,  $i = 1 \dots n$ . Theo quy tắc an toàn, sau khi thu gom xong phải đóng ngay container. Thời gian di chuyển của xe chở hóa chất từ một điểm sang điểm ở sau bất kỳ là 1 đơn vị thời gian. Nếu xe đến sớm thì có thể chờ hoặc bỏ qua đi tới điểm tiếp theo và sau này không quay lại nơi đã bỏ qua. Xe đến một nơi vào đúng thời điểm thu gom xong thì vẫn kịp xử lý. Ở thời điểm 1 xe đang đứng tại điểm thứ nhất của lộ trình.

Người ta muốn ngay trong lần thử nghiệm đầu tiên thiêu hủy được càng nhiều gia cầm càng tốt.

Hãy xác định tổng số gia cầm thiêu hủy được, số lượng địa điểm và các địa điểm đã hoàn thành việc thiêu hủy gia cầm.

**Dữ liệu:** Vào từ file văn bản FLU.INP:

- ◆ Dòng đầu tiên chứa số nguyên  $n$  ( $1 \leq n \leq 10^5$ ),
- ◆ Dòng thứ 2 chứa  $n$  số nguyên  $a_1, a_2, \dots, a_n$  ( $1 \leq a_i \leq 10^9$ ,  $i = 1 \dots n$ ),
- ◆ Dòng thứ 3 chứa  $n$  số nguyên  $t_1, t_2, \dots, t_n$  ( $1 \leq t_i \leq 10^5$ ,  $i = 1 \dots n$ ).

**Kết quả:** Đưa ra file văn bản FLU.OUT:

- ◆ Dòng thứ nhất chứa một số nguyên – tổng số gia cầm thiêu hủy được,
- ◆ Dòng thứ 2 chứa số nguyên  $m$  – số lượng địa điểm đã hủy gia cầm,
- ◆ Dòng thứ 3 chứa danh sách (theo thứ tự tăng dần) các điểm đã thiêu hủy gia cầm.

**Ví dụ:**

FLU.INP	FLU.OUT
5	10
4 1 5 9 3	2
9 7 9 8 8	2 4

## Giải thuật

Đây là bài toán quy hoạch động, yêu cầu đưa ra giá trị hàm mục tiêu và phương án tối ưu.

Giải thuật độ phức tạp  $O(n^2)$ :

- $f_i$  – giá trị hàm mục tiêu đạt được khi xuất phát từ  $i$  (`int64_t`),
- $p_i$  – mốc nối từ  $i$  tới điểm tiếp theo.
- $f_n = a_n$ ,  $p_n = 0$ ,
- $f_i = a_i + \max\{f_j | j > i, t_j > t_i\}$ ,
- Giả thiết max đạt ở  $jx$  ( $jx = 0$  nếu không tồn tại  $t_j > t_i$ ):  $p_i = jx$ .

Kết quả: Giá trị hàm mục tiêu:  $\max\{f_i\}$ ,  $i = 1 \dots n$ ,

Phương án tối ưu: giả thiết max nói trên đạt ở  $ix$ : đếm số lượng danh sách mốc nối bắt đầu từ  $p_{ix}$ , đưa ra số lượng và sau đó – danh sách mốc nối.

Giải thuật độ phức tạp  $O(n \log(n))$ :

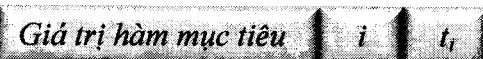
Dùng cây nhị phân quản lý max:

Số phần tử ở lớp lá:  $tx$ ,  $tx = \max\{t_i\}$ ,  $i = 1 \dots n$ ,

$m$  – chỉ số của phần tử lá đầu tiên:

```
for(int i=20; i>=0; --i) if((tx&(1<<i)) != 0) {m=1<<(i+1); break;}
```

$f$  – mảng  $4 \times m$  phần tử, mỗi phần tử: 3 trường:



Các phép xử lý:

- Tìm max với các  $t_j > t_i$ : `i1=m+t[i]-1; get_mx(i1+1, t[i]);`
- Hàm `get_max`:

```
void get_mx(int x, int y)
{
    int u, u1;
    pi3 tu, tv;
    u=x; tu=f[u];
    while(u!=1)
    {u1=u>>1;
     if(f[2*u1].second.second>y && f[2*u1].first>tu.first) tu=f[2*u1];
     if(f[2*u1+1].second.second>y && f[2*u1+1].first>tu.first) tu=f[2*u1+1];
     u=u1;
    }
    d=tu;
}
```

```
void upd_t(int x)
{
    int v1;
    u=x;
    while(u!=1)
    {v1=u>>1; if(f[v1*2].first<f[v1*2+1].first) f[v1]=f[v1*2+1];
     else f[v1]=f[v1*2];
     u=v1;
    }
}
```

- Hàm cập nhật max:

Kết quả: Thông tin chứa trong  $f_i$ .

*Chuong trinh giải*

```
#include <fstream>
using namespace std;
ifstream fi ("flu.inp");
ofstream fo ("flu.out");
int n,m,i1,im,tx,fx,u,v,ru,rv,a[100001], t[100001],p[100001]={0};
typedef pair<int,int> pii ;
typedef pair<int64_t,pii> pi3;
pi3 d,f[4000010]={make_pair(0,make_pair(0,0))};

void upd_t(int x)
{ int v1;
  u=x;
  while(u!=1)
  {v1=u>>1;if(f[v1*2].first<f[v1*2+1].first)f[v1]=f[v1*2+1];
   else f[v1]=f[v1*2];
  u=v1;
  }
}
void get_mx(int x,int y)
{int tg2,u1,v1;
 int64_t tgl;
 pi3 tu,tv;
 u=x; v=im;tu=f[u];
 while(u!=1)
 {u1=u>>1;
  if(f[2*u1].second.second>y && f[2*u1].first>tu.first) tu=f[2*u1];
  if(f[2*u1+1].second.second>y && f[2*u1+1].first>tu.first) tu=f[2*u1+1];
  u=u1;
 }
 d=tu;
}

int main()
{fi>>n; tx=0;
 for(int i=1;i<=n;++i)fi>>a[i];
 for(int i=1;i<=n;++i){fi>>t[i]; if(t[i]>tx) tx=t[i];}
 for(int i=20;i>=0;--i) if((tx&(1<<i)) != 0){m=1<<(i+1); break;}
 i1=m+t[n]-1; im=m+tx-1; d=make_pair(a[n],make_pair(n,t[n]));f[i1]=d;
 upd_t(i1);
 for(int i=n-1;i>0;--i)
 {i1=m+t[i]-1;get_mx(i1+1,t[i]);
  p[i]=d.second.first;
  if(a[i]+d.first>=f[i1].first)
   {d.first+=a[i]; d.second.second=t[i];d.second.first=i;
    f[i1]=d;
    upd_t(i1);}
 }
 u=0;
 fo<<f[1].first<<'\n'; i1=f[1].second.first; u=0;
 while(i1!=0){++u;i1=p[i1];}
 fo<<u<<'\n'; i1=f[1].second.first;
 while(i1!=0){fo<<i1<<' ';i1=p[i1];}
}
```

## VO42. ĐỒ CHƠI

Tên chương trình: TOY.???

Một máy giải trí tự động thế hệ mới được lắp đặt ở công viên thành phố. Máy có  $n$  nút đánh số từ 1 đến  $n$ . Trừ nút đầu tiên (nút số 1) ở các nút còn lại đều có một đường ống dẫn đến. Mỗi nút có không quá 2 đường ống dẫn ra, một ống đi sang trái và một ống – sang phải. Mỗi đường ống có kích thước đường kính ban đầu của mình. Khi người chơi bỏ đồng xu vào nút số 1, đồng xu sẽ theo các đường ống trượt xuống dưới. Tới mỗi nút có 2 đường ống dẫn ra, đồng xu luôn trượt theo đường ống kích thước lớn hơn. Nếu 2 đường ống có cùng kích thước thì đồng xu sẽ trượt theo ống dẫn sang trái. Sau khi đồng xu trượt qua, đường kính của đường ống này sẽ giảm đi 1. Đồng xu không thể trượt theo đường ống kích có đường kính bằng 0. Nếu tại một nút nào đó đồng xu không trượt tiếp xuống dưới được thì máy sẽ dừng lại, chờ người chơi bỏ tiếp đồng xu mới.

Ban đầu ở mỗi nút có một đồ chơi. Khi lần đầu tiên đồng xu tới một nút, đồ chơi ở nút đó được đẩy ra ngoài, rồi vào giỏ quà của người chơi.

Steve rất thích đồ chơi đặt ở nút  $v$ . Hãy xác định số lượng xu tối thiểu cần bỏ vào máy để có được đồ chơi này. Nếu không có cách nào nhận được đồ chơi này thì đưa ra số -1.

**Dữ liệu:** Vào từ file văn bản TOY.INP:

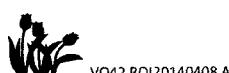
- ◆ Dòng đầu tiên chứa số nguyên  $n$  ( $1 \leq n \leq 10^5$ ),
- ◆ Dòng thứ  $i$  trong  $n$  dòng sau chứa 4 số nguyên  $a_i, u_i, b_i$  và  $w_i$  cho biết từ nút thứ  $i$  đường ống bên trái dẫn tới nút  $a_i$  với đường kính  $u_i$ , đường ống bên phải dẫn tới nút  $b_i$  với đường kính  $w_i$ . Nếu đường ống nào không có thì cặp số tương ứng là 0 và 0 ( $1 \leq a_i, b_i \leq n, 1 \leq u_i, w_i \leq 10^9$ ),
- ◆ Dòng cuối cùng chứa số nguyên  $v$  ( $1 \leq v \leq n$ ).

**Kết quả:** Đưa ra file văn bản TOY.OUT một số nguyên – kết quả tính được.

**Ví dụ:**

TOY.INP
7
2 1 3 2
0 0 6 3
4 1 5 1
0 0 0 0
7 2 0 0
0 0 0 0
0 0 0 0
5

TOY.OUT
3

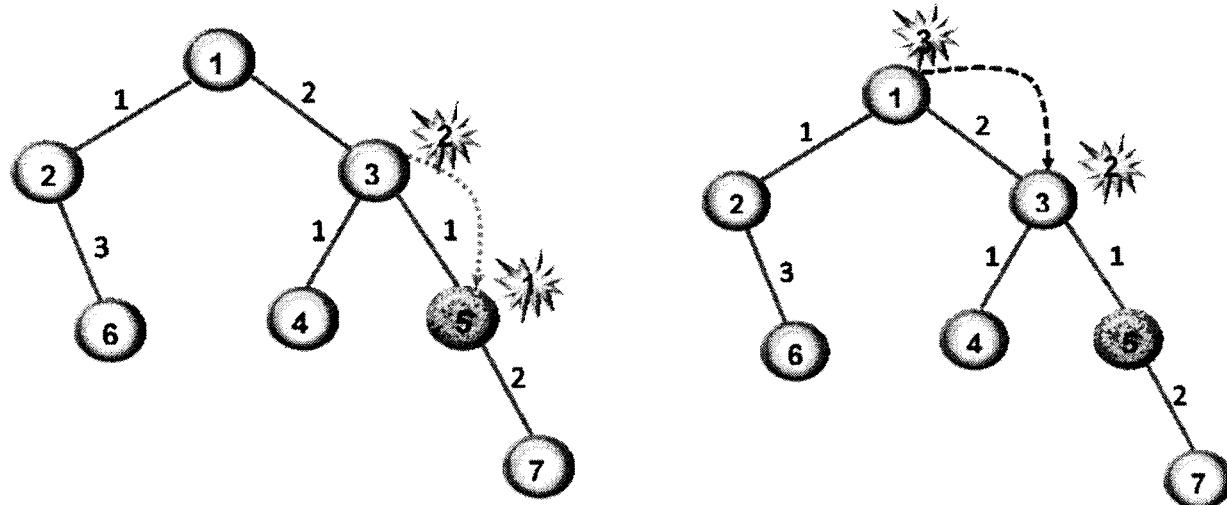


## Giải thuật

Giải thuật tối ưu được xây dựng theo *sơ đồ lấp* tương tự như ở một bài toán quy hoạch động. Đây không phải là một bài toán quy hoạch động vì kết quả ở mỗi bước tiếp theo hoàn toàn xác định đơn trị từ kết quả nhận được ở bước trước.

Gọi  $u$  là nút cha của  $v$ . Ta phải giải quyết bài toán: “để có  $p$  đồng tiền tới nút  $v$  thì cần có bao nhiêu đồng tiền tới nút  $u$ ?” Gọi  $q$  là kết quả của bài toán trên.

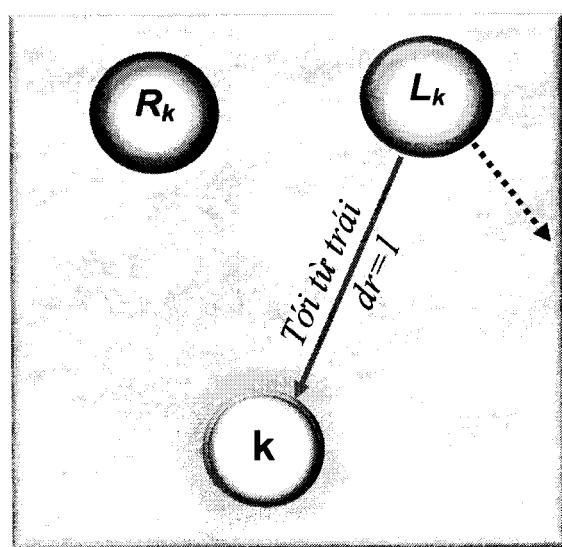
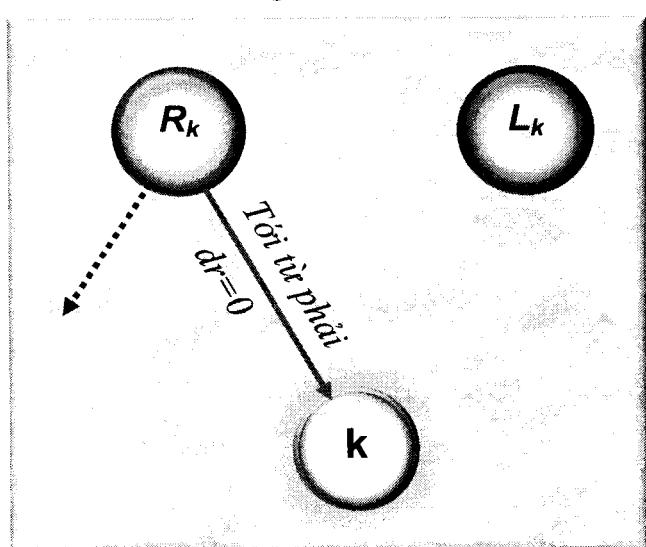
Xuất phát từ  $v$  ban đầu và  $p = 1$ , ta tìm  $u$  và xác định  $q$ . Nếu tồn tại  $q$  thì cho  $u$  nhận vai trò của  $v$ , với  $p = q$  tìm lại  $u$  và  $q$  mới cho đến khi có  $u = 1$  hoặc không



tìm được  $q$ .

Tổ chức dữ liệu:

- ❖ Các mảng  $a$ ,  $b$ ,  $c$ ,  $d$ : kích thước 100 001 loại int – lưu trữ dữ liệu vào,
- ❖ Mảng int  $L[100001]$ ,  $R[100001]$  lưu trữ mốc nối,
- ❖  $k$  – nút đang xét,
- ❖  $dr$  – hướng đi từ nút cha tới  $k$ ,



- ❖  $m$  – số đồng tiền cần dùng.
- Giá trị đầu:  $k = v$ ;  $m = 1$ ;

Xử lý trong một bước lắp:

- ➔ Thay k bằng nút cha của nó,
- ➔ Nếu cạnh di chuyển hẹp hơn m – bài toán vô nghiệm, kết thúc xử lý,
- ➔ Nếu di chuyển được – có 3 trường hợp:
  - Sang trái và kích thước ống sau khi m đồng tiền đi qua ( $b_k - m$ ) lớn hơn hoặc bằng  $d_k - 1 \rightarrow$  không thay đổi m,
  - Sang phải và kích thước ống sau khi m đồng tiền đi qua ( $d_k - m$ ) lớn hơn hoặc bằng  $b_k \rightarrow$  không thay đổi m,
  - Trường hợp còn lại: đường ống to sẽ thu nhỏ dần cho đến khi hai bên bằng nhau, bắt đầu từ đó lắp lại dao động (qua trái, qua phải), dễ dàng tính được số tiền cần thiết đi qua nút k để có m đồng tiền tới nút con đang xét.

Lưu ý:

- Việc quy định giá trị dr như đã nêu chỉ để thuận tiện tính toán,
- Có thể tránh việc tìm min, max khi tính giá trị mới của m bằng giá trị tuyệt đối chênh lệch kích thước của 2 đường ống đi ra.

```

#include <fstream>
#include <ctime>
#include <cmath>
using namespace std;
int
l[100001]={0},r[100001]={0},a[100001],b[100001],c[100001],d[100001];
int n,v,m,k;
ifstream fi ("toy.inp");
ofstream fo ("toy.out");

void xly()
{int dr,x,y,z,tm,tx;
dr=1; if(l[k])k=l[k]; else{k=r[k];dr=0;}
x=b[k];y=d[k];tx=x;
if(dr)z=x-m; else z=y-m;
if(z<0){m=-1; return;}
if((dr==1 && z>=y-1) || (dr==0 && z>x))return;
if(y>tx)tx=y; tm=x;if(tm>y)tm=y;
m=tx-tm+(tm-z)*2-dr;
}

int main()
{clock_t aa=clock();
fi>>n;
for(int i=1;i<=n;++i)
{fi>>a[i]>>b[i]>>c[i]>>d[i];
if(a[i])l[a[i]]=i;
if(c[i])r[c[i]]=i;
}
fi>>v; m=1;k=v;
while(k!=1 && m>0)xly();
fo<<m;
clock_t bb=clock();
fo<<"\nTime: "<<(double)(bb-aa)/1000<<" sec.";
}

```

### 3. Các bài toán có nội dung hình học

#### KTLT. ĐƯỜNG GẤP KHÚC

Tên chương trình: POLYG.???

Cho đường gấp khúc khép kín  $n$  đỉnh không tự cắt, đỉnh thứ  $i$  có tọa độ nguyên  $(x_i, y_i)$  ( $|x_i|, |y_i| \leq 10^6$ ,  $i = 1 \dots n$ ). Các đỉnh được liệt kê theo một trình tự nào đó.

Hãy tính diện tích của đa giác tạo bởi đường gấp khúc này.

*Dữ liệu:* vào từ file POLYG.INP:

- Dòng đầu tiên chữ số nguyên  $n$  ( $2 < n \leq 10^5$ ),
- Dòng thứ  $i$  trong  $n$  dòng sau chứa 2 số nguyên  $x_i$  và  $y_i$ .

*Kết quả:* đưa ra file POLYG.OUT với độ chính xác 6 chữ số sau dấu chấm thập phân.

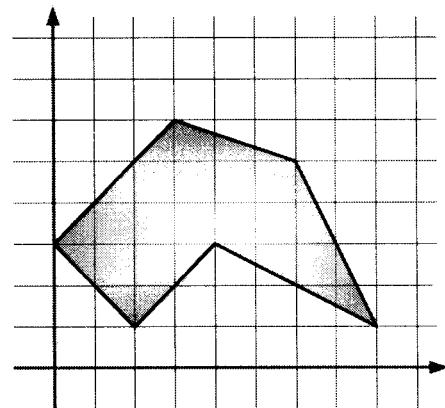
*Ví dụ:*

POLYGON.INP
6
2 1
0 3
3 6
6 5
8 1
4 3

POLYGON.OUT
20.000000



e38 Bait11 F



## Giải thuật

Công thức tính diện tích đa giác:

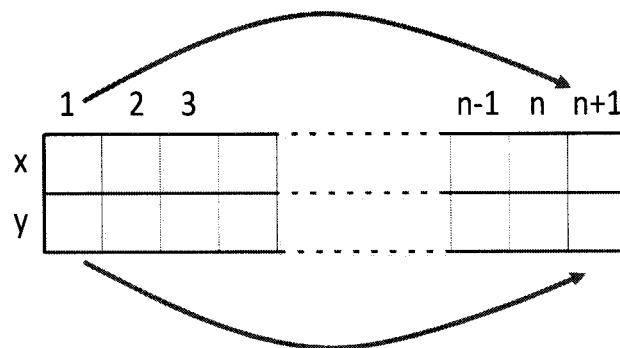
$$S = \frac{1}{2} \left| \sum_{i=1}^n (x_i y_{i+1} - x_{i+1} y_i) \right|$$

Trong đó  $x_{n+1} \equiv x_1$ ,  $y_{n+1} \equiv y_1$ .

Lưu ý: biểu thức  $\sum_{i=1}^n (x_i y_{i+1} - x_{i+1} y_i)$  cho giá trị dương khi các đỉnh được liệt kê theo chiều ngược kim đồng hồ và giá trị âm trong trường hợp ngược lại.

Dữ liệu vòng tròn:

Tạo  $x_{n+1} = x_1$ ,  
 $y_{n+1} = y_1$ .



```
#include <iostream>
#include <iomanip>
#include <cmath>
using namespace std;
int n;
long long d;
float s;
int x[100001], y[100001];
ifstream fi ("polyg.inp");
ofstream fo ("polyg.out");
void nhap()
{
    fi>>n;
    for(int i=0;i<n;++i)
        fi>>x[i]>>y[i];
}

```

```
void ch_bi()
{
    x[n]=x[0];
    y[n]=y[0];
    d=0;
}
void dt()
{
    for(int i=0;i<n;++i)
        d+=(x[i]*y[i+1]-x[i+1]*y[i]);
    s=((float)abs(d))/2;
}
int main()
{
    nhap();
    ch_bi();
    dt();
    fo<<fixed<<setw(10)<<s;
}
```

Cần #include <cmath>

Đổi kiểu dữ liệu

Độ rộng của trường  
kết quả

## TE38. ĐƯỜNG GẤP KHÚC 2

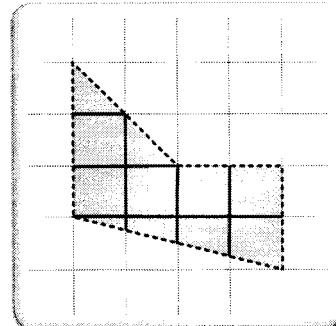
Tên chương trình: POLYGON.???

Trên lưới ô vuông vô hạn với tọa độ các đỉnh nút là nguyên người ta cho một đường gấp khúc đơn khép kín  $n$  đỉnh, tức là đường gấp khúc mà 2 cạnh liên tiếp chỉ có một điểm chung ở đỉnh, một cạnh không cắt các cạnh khác và cũng không có điểm chung nào khác. Đỉnh của đường gấp khúc trùng với điểm chia của lưới.

Hãy tính tổng độ dài các đoạn thẳng của lưới nằm gọn trong đa giác giới hạn bởi đường gấp khúc.

**Dữ liệu:** Vào từ file văn bản POLYGON.INP:

- Dòng đầu tiên chứa số nguyên  $n$  ( $3 \leq n \leq 105$ ),
- Dòng thứ  $i$  trong  $n$  dòng sau chứa 2 số nguyên  $x_i$  và  $y_i$  – tọa độ đỉnh  $i$  ( $|x_i|, |y_i| \leq 5 \times 10^8$ ). Tọa độ các đỉnh cho theo một chiều nào đó (cùng hoặc ngược chiều kim đồng hồ).



**Kết quả:** Đưa ra file văn bản POLYGON.OUT một số thực  $L$  – tổng độ dài tìm được. Gọi  $R$  là độ dài chính xác. Kết quả được coi là đúng nếu thỏa mãn một trong 2 điều kiện sau:

- $|L-R| \leq R \times 10^{-6}$  (sai số tương đối),
- $|L-R| \leq 10^{-6}$  (sai số tuyệt đối).

**Ví dụ:**

POLYGON.INP
5
0 0
-2 2
-2 -1
2 -2
2 0

POLYGON.OUT
12.5



### Giải thuật

Gọi  $S$  là diện tích của đa giác giới hạn bởi đường gấp khúc đã cho, ta có:

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

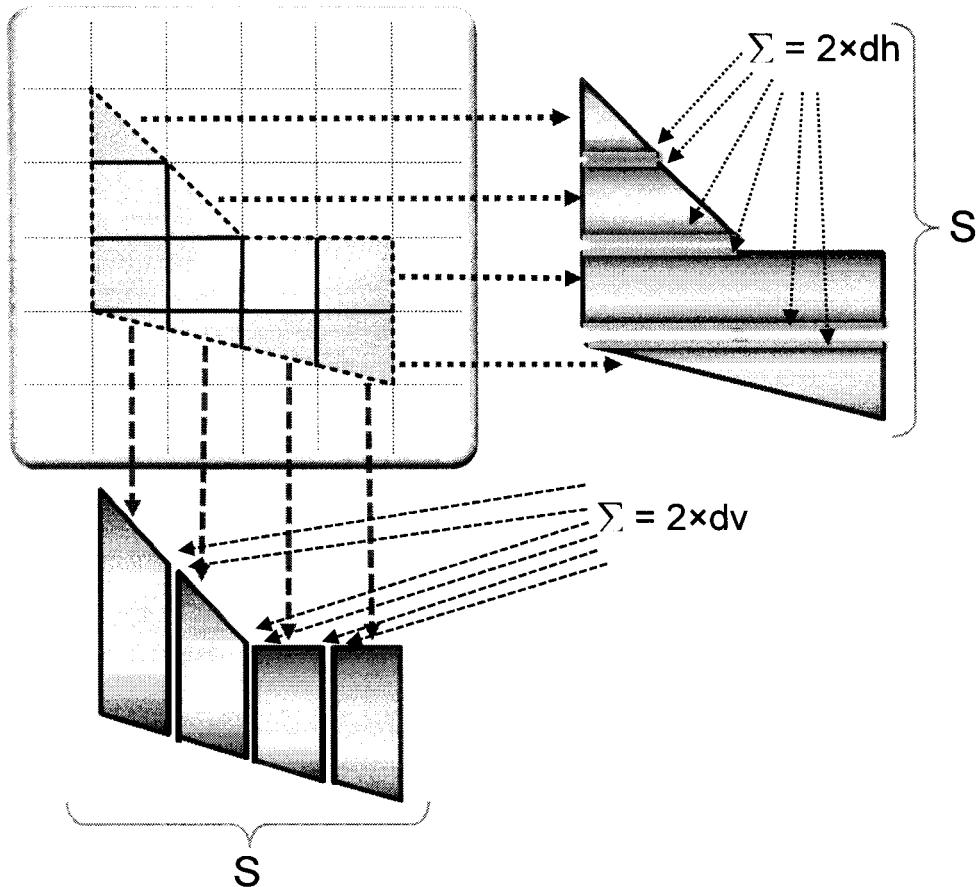
Mặt khác ta có tính  $S$  bằng phương pháp vi phân theo một trong 2 cách:

- ➔ Tổng diện tích các hình thang (có thể suy biến thành tam giác) có độ cao 1 và các cạnh đáy song song với trục  $Ox$ ,
- ➔ Tổng diện tích các hình thang (có thể suy biến thành tam giác) có độ cao 1 và các cạnh đáy song song với trục  $Oy$ .

Gọi:

- ➔  $dh$  là tổng độ dài các đoạn thẳng của lưới song song với trục  $Ox$ , nằm gọn trong đa giác giới hạn bởi đường gấp khúc,
- ➔  $dv$  là tổng độ dài các đoạn thẳng của lưới song song với trục  $Oy$ , nằm gọn trong đa giác giới hạn bởi đường gấp khúc,

- Lh – tổng độ dài các đoạn thẳng song song với trục Ox và thuộc chu vi của đa giác,
- Lv – tổng độ dài các đoạn thẳng song song với trục Oy và thuộc chu vi của



đa giác.

Ta có:

$$2 \times S = (L_h + 2 \times d_h) \times 1,$$

$$2 \times S = (L_v + 2 \times d_v) \times 1.$$

Giá trị cần tìm là  $d_h+d_v$  dễ dàng rút ra từ các công thức tính diện tích S đã nêu.

Lưu ý:

- Chỉ chuyển sang làm việc với số thực khi tính kết quả cuối cùng cần đưa ra,
- $2 \times S$  có thể nhận giá trị thuộc loại `int64_t`.

```

#include <fstream>
#include <cmath>
using namespace std;
int n, lh = 0, lv = 0, S = 0, x[100100], y[100100];
ifstream fi ("POLYGON.INP");
ofstream fo ("POLYGON.OUT");
void input(){
    fi>>n;
    for (int i = 1; i <= n; i++) fi>>x[i]>>y[i];
    x[n+1] = x[1];
    y[n+1] = y[1];
}
void cal_area(){
    for (int i = 1; i <= n; i++)
        S += x[i]*y[i+1] - y[i]*x[i+1];
    S = abs(S);
}
void cal_lh_lv(){
    for (int i = 1; i <= n; i++){
        if (x[i] == x[i+1])
            lv += abs(y[i+1] - y[i]);
        if (y[i] == y[i+1])
            lh += abs(x[i+1] - x[i]);
    }
}
void output(){
    fo << double(2*abs(S) - lv - lh) / double(2);
}
int main(){
    input();
    cal_area();
    cal_lh_lv();
    output();
}

```

## VN19. KINH KẾ

Tên chương trình: SUTRAS.???

Các nhà khảo cổ học tìm thấy một khu di tích **n** hang động của nhà Phật. Các hang động này được thiết lập bên vách đá của một hẻm núi cheo leo sâu thẳm. Để vào khảo sát hiện nay chỉ có một cách duy nhất là treo người thả xuống từ máy bay lên thẳng. Do cây cối rậm rạp và địa hình hẻm núi phức tạp nên cũng chỉ có thể tiếp cận được một hang bằng cách này. Kết quả khảo sát sơ bộ từ xa cho thấy một số hang có cấu trúc đặc biệt để lưu trữ kinh phan cổ. Theo truyền thống, mỗi hàng này sẽ có một bộ kinh. Nếu đánh số các hang từ 1 trở đi bắt đầu từ hang trung tâm và đánh số theo đường xoáy tròn ốc ngược chiều kim đồng hồ (xem hình vẽ) thì những hang và chỉ những hang có số thứ tự là nguyên tố mới chứa kinh kệ.

37	36	35	34	33	32	31
38	17	16	15	14	13	30
39	18	5	4	3	12	29
40	19	6	1	2	11	28
41	20	7	8	9	10	27
42	21	22	23	24	25	26
43	44	45	46			

Khi thả một người vào hang có thể tiếp cận được thì người đó có thể khảo sát hang này và sau đó có thể dùng dây thừng để xuống một trong 3 hang: xuống hang ngay dưới hang mình đang đứng hoặc hang bên phải hay hang bên trái của hang dưới, tức là xuống một trong 3 hang ở dưới kè cạnh hoặc kè đinh với hang đang đứng. Các nhà khảo cổ rất nóng lòng muốn ngay từ lần khảo sát đầu tiên thu được nhiều bộ kinh cổ nhất. Theo cách đánh số đã nêu, hang có thể trực tiếp cận được có số là **k**.

Hãy đưa ra số lượng bộ kinh thu được và số thứ tự của hang cuối cùng thu lượm được kinh. Nếu tồn tại nhiều đường đi cùng thu được số bộ kinh như nhau thì đi theo đường có số thứ tự của hang cuối cùng thu lượm được kinh là lớn nhất. Nếu không thể thu được bộ kinh nào thì đưa ra 2 số 0.

**Dữ liệu:** Vào từ file văn bản SUTRAS.INP, gồm nhiều bộ tests, mỗi bộ test cho trên một dòng chứa 2 số nguyên **n** và **k** ( $1 \leq k \leq m \leq 10^6$ ). Dữ liệu kết thúc bằng dòng chứa 2 số 0.

**Kết quả:** Đưa ra file văn bản SUTRAS.OUT, kết quả mỗi test đưa ra trên một dòng gồm 2 số nguyên xác định các giá trị cần tìm.

**Ví dụ:**

SUTRAS.INP
49 22
46 37
42 23
945 561
1081 681
1056 452
1042 862
973 677
1000000 1000000
0 0

SUTRAS.OUT
0 0
6 43
1 23
20 829
18 947
10 947
13 947
23 947
534 993541



VN19 Jp ACM 2013

## *Giải thuật*

Tạo bảng số nguyên tố  $A = (a_1, a_2, \dots, a_{1000000})$ ,  $a_i = 1$  nếu  $i$  là số nguyên tố và  $a_i = 0$  trong trường hợp ngược lại.

Bố trí các số nguyên theo đường xoáy tròn trên bảng  $c[1002][1002]$ , tọa độ ô – theo tọa độ trên mặt phẳng:

- ◆ Tâm:  $c_{500,500} = 1$ ,
- ◆ Lần lượt 4 cạnh 0, 1, 2 và 3,
- ◆  $d$  – độ dài cạnh vòng  $i$ ,  $d = 2*i$ ,  $i = 1, 2, \dots, 500$ ,

```
c[500][500]=1;u=1;v=0;t=1;
for(int i=1;i<=500;++i)
{d=i<<1;
 for(int j=1;j<=d;++j){c[500+u][500+v]=++t;++v;}
 --v;--u;for(int j=1;j<=d;++j){c[500+u][500+v]=++t;--u;}
 --v;+u;for(int j=1;j<=d;++j){c[500+u][500+v]=++t;--v;}
 ++v;+u;for(int j=1;j<=d;++j){c[500+u][500+v]=++t;+u;}
 }
```

Hàm hỗ trợ: `void get_coord(int x)` cho tọa độ (ik, jk) của số nguyên x.

```
void get_coord(int x)
{v=(int)sqrt(x);
 if(v*v<x)++v; if((v&1)==0)++v;t=x-(v-2)*(v-2)-1;
 v/=2; d=v<<1;
 t1=t/d; t2=t%d;
 switch(t1)
 {case 0:ik=500+v;jk=501-v+t2; break;
 case 1:ik=500+v-1-t2; jk=500+v; break;
 case 2:ik=500-v; jk=500+v-1-t2; break;
 case 3:ik=500-v+1+t2; jk=500-v;
 }
}
```

Với mỗi cặp số nguyên n, k – quy hoạch động:

- ◆ Bản đồ lặp: 2 dòng dữ liệu để lặp: `pair<int, int> b[2][1005]`,
- ◆ Từ  $b_{p,i}$  tính  $b_{q,i}$ ,  $i = 2 \div 1002$ ,
- ◆ Sơ đồ lặp:
  - Bắt đầu từ  $b[p][ik+2]$ ,  $b[p][ik+2].second = 1$  nếu  $k$  – số nguyên tố,
  - Dòng xuất phát:  $jt = jk - 1$ ,
  - $b[q][i] = \max\{b[p][i-1], b[p][i], b[p][i+1]\}$  với các số không vượt quá n,
  - Lưu ý giữ max toàn cục trong quá trình lặp,
  - Dùng kỹ thuật con lắc đảo vai trò các dòng sau mỗi bước lặp,
  - Điều kiện dừng: khi  $jt < 0$  hoặc ở dòng mới không có số nào  $\leq n$ .

```

#include <fstream>
#include <iomanip>
#include <cmath>
#include <ctime>
using namespace std;
ifstream fi ("sutras.inp");
ofstream fo ("sutras_T.out");
int a[1000002],c[1002][1002]={0},t,u,v,d,t1,t2,ik,jk,
    p,q,n,k,f1,ir,il,irn,iln,jt;
pair<int,int> b[2][1005],r,rx,ans;

void chbi()
{for(int i=0;i<=1000000;++i)a[i]=1; a[1]=0;
 for(int i=2;i<=1000;++i)
    if(a[i]) for(int j=i*i;j<=1000000;j+=i)a[j]=0;
 c[500][500]=1;u=1;v=0;t=1;
 for(int i= 1;i<=500;++i)
    {d=i<<1;
     for(int j=1;j<=d;++j){c[500+u][500+v]=++t;++v;}
     --v;--u;for(int j=1;j<=d;++j){c[500+u][500+v]=++t;--u;}
     --v;++u;for(int j=1;j<=d;++j){c[500+u][500+v]=++t;--v;}
     ++v;++u;for(int j=1;j<=d;++j){c[500+u][500+v]=++t;++u;}
    }
}

void get_coord(int x)
{v=(int)sqrt(x);
 if(v*v<x)++v; if((v&1)==0)++v;t=x-(v-2)*(v-2)-1;
 v/=2; d=v<<1;
 t1=t/d; t2=t%d;
 switch(t1)
 {case 0:ik=500+v;jk=501-v+t2; break;
  case 1:ik=500+v-1-t2; jk=500+v; break;
  case 2:ik=500-v; jk=500+v-1-t2; break;
  case 3:ik=500-v+1+t2; jk=500-v;
 }
}

void get_mx(int x)
{rx=b[p][x-1];
 if(rx.first<b[p][x].first || (rx.first==b[p][x].first &&
 rx.second<b[p][x].second))rx=b[p][x];
 if(rx.first<b[p][x+1].first || (rx.first==b[p][x+1].first &&
 rx.second<b[p][x+1].second))rx=b[p][x+1];
}

void trace()
{int flag;
 memset(b,0,sizeof(b));
 p=0;q=1;ir=ik+2;il=ir;jt=jk; t=c[ik][jt];--jt; flag=1;
 if(a[t]==1){b[0][ir].first=1;b[0][ir].second=t;}
 ans=b[0][ir];
 while((flag==1) && (jt>=0))
 {flag=0;f1=0;
  for(int i=il-1;i<=ir+1;++i)
  {t=c[i-2][jt];
   if(t>0 && t<=n)
   {
    get_mx(i);flag=1;
    if(a[t])++rx.first; if(t>rx.second)rx.second=t;
    b[q][i]=rx;
    if(rx.first>ans.first || (rx.first==ans.first &&
 rx.second>ans.second))ans=rx;
   }
}

```

```

}
--jt;if(il>2)--il;if(ir<1003++ir;if(flag){p^=1;q^=1;}      }
fo<<ans.first<<' '<<ans.second<<'\n';
}

int main()
{ clock_t aa=clock();
  chbi();
  fi>>n>>k;
  while(n!=0)
  {if(k==1){ik=500;jk=500;} else get_coord(k);
   trace();
   fi>>n>>k;
  }
  clock_t bb=clock();
  fo<<"\nTime: "<<(double)(bb-aa)/1000<<" sec.";
}

```

## 4. Các bài toán đồ thị

### 4.1 – Vai trò của cấu trúc dữ liệu trong các bài toán đồ thị

Các bài toán tin học ngày nay, ngay trong phạm vi chương trình phổ thông trung học cũng đã có những đặc thù đòi hỏi phải có cách tiếp cận mới trong việc xây dựng và cài đặt giải thuật. Đặc điểm của phần lớn các bài toán hiện nay là:

- ◆ Số đỉnh  $n$  là rất lớn, từ vài trăm ngàn đến vài triệu,
- ◆ Độ dày dàu thấp: số cạnh chỉ vài cho đến vài chục phần trăm so với ma trận dày dàu,
- ◆ Không đơn thuần tìm độ dài đường đi ngắn nhất mà phải xác định cả bản thân đường đi ngắn nhất, thực hiện các phân tích và đánh giá khác nhau về nhóm đường đi có độ dài ngắn nhất,
- ◆ Thực hiện các truy vấn động trên đồ thị: Các thông số về đỉnh, cạnh, điểm xuất phát, điểm đích . . . có thể thay đổi từ truy vấn này sang truy vấn khác.

Các giải thuật cơ sở kinh điển được xem xét và giảng dạy ở thế kỷ XX vẫn rất cần thiết, nhưng là chưa đủ trong việc đào tạo và bồi dưỡng học sinh năng khiếu tin học phù hợp với yêu cầu thực tế hiện tại.

Một giải thuật nói chung và trong các bài toán đồ thị nói riêng, muốn có hiệu quả phải *tận dụng được tối đa và hợp lý các khả năng mà môi trường kỹ thuật cung cấp*, cụ thể là:

- ◆ Cho phép sử dụng khối lượng bộ nhớ lớn,
- ◆ Khai thác được một cách hợp lý các cấu trúc dữ liệu mà các hệ thống lập trình cung cấp,
- ◆ Tốc độ cao của các thiết bị tính toán.

Thông thường các bài toán cần giải cho phép sử dụng không ít hơn 64MB. Người giải bài toán được quyền sử dụng mọi dịch vụ do thư viện chuẩn của hệ thống lập trình cung cấp.

Việc sử dụng các cấu trúc dữ liệu này, khi triển khai, mang lại hiệu quả ngay cả với các bài toán kinh điển.

#### 4.1.1 – HÀNG ĐỢI ƯU TIÊN VÀ GIẢI THUẬT DIJSKTRA

##### A - BÀI TOÁN

Cho đồ thị  $n$  đỉnh và  $m$  cạnh. Đồ thị có thể có hướng hoặc không. Trọng số của mỗi cạnh là không âm. Hãy xác định đường đi ngắn nhất từ đỉnh  $s$  cho trước *tới mỗi đỉnh còn lại* và độ dài của đường đi đó.

##### B - GIẢI THUẬT

Tổ chức mảng  $D = (d_1, d_2, \dots, d_n)$ ,  $d_v$  lưu trữ độ dài đường đi ngắn nhất từ  $s$  tới  $v$ ,  $v = 1 \div n$ . Ban đầu  $d_s = 0$ ,  $d_v = \infty$ ,  $v = 1 \div n$ ,  $v \neq s$ . Ngoài ra, còn cần tới mảng lô gic  $U = (u_1, u_2, \dots, u_n)$  để đánh dấu, cho biết đỉnh  $v$  đã được xét hay chưa. Ban đầu,  $u_v = \text{false}$  với  $v = 1 \div n$ .

Bản thân giải thuật Dijkstra gồm  $n$  bước.

Ở mỗi bước cần chọn đỉnh  $v$  có  $d_v$  là nhỏ nhất trong số các đỉnh  $v$  chưa được đánh dấu, tức là

$$d_v = \min\{d_i \mid u_i = \text{false}, i = 1 \div n\}$$

Công việc tiếp theo trong bước này là điều chỉnh  $D$ : xét tất cả các cạnh  $(v, t)$ . Gọi  $1t$  là trọng số của cạnh  $(v, t)$ . Giá trị  $d_t$  được chỉnh lý theo công thức

$$d_t = \min\{d_t, d_v + 1/t\}$$

Sau  $n$  bước, tất cả các đỉnh đều được đánh dấu và  $d_v$  sẽ là độ dài đường đi ngắn nhất từ  $s$  đến  $v$ . Nếu không tồn tại đường đi từ  $s$  đến  $v$  thì  $d_v$  vẫn nhận giá trị  $\infty$ .

Để khôi phục đường đi có độ dài ngắn nhất cần tổ chức mảng  $P = (p_1, p_2, \dots, p_n)$ , trong đó  $p_v$  lưu đỉnh cuối cùng trước đỉnh  $v$  trong đường đi ngắn nhất từ  $s$  đến  $v$ . Mỗi lần, khi  $d_t$  thay đổi giá trị thì đỉnh đạt min:  $p_t = v$ .

Tính đúng đắn của giải thuật được nêu trong nhiều tài liệu khác nhau và là điều không cần phải trình bày ở đây.

Điều quan trọng là đánh giá độ phức tạp của giải thuật và làm thế nào để giảm độ phức tạp đó. Giải thuật bao gồm  $n$  bước lặp, ở mỗi bước lặp cần duyệt tất cả các đỉnh và sau đó – chỉnh lý  $d_t$ . Như vậy giải thuật có độ phức tạp là  $O(n^2+m)$ .

## C - KỸ THUẬT CÀI ĐẶT HIỆU QUẢ CAO VỚI ĐỒ THỊ MA TRẠN THƯA

Với đồ thị có số cạnh  $m$  nhỏ hơn nhiều so với  $n^2$  thì độ phức tạp của giải thuật có thể giảm xuống bằng việc cải tiến cách duyệt đỉnh ở mỗi bước lặp.

Mục tiêu này có thể đạt được thông qua việc sử dụng Cấu trúc vun đồng Fibonacci (**Fibonacci Heap**), Cấu trúc tập hợp (**Set**) hoặc cấu trúc Hàng đợi ưu tiên (**Priority Queue**).

Cấu trúc vun đồng Fibonacci cho phép giải bài toán tìm đường đi ngắn nhất với độ phức tạp  $O(nlogn+m)$ . Về mặt lý thuyết, đây là độ phức tạp *tối ưu* cho chương trình giải các bài toán dựa trên cơ sở giải thuật Dijkstra. Tuy nhiên việc cài đặt khác phức tạp vì thư viện chuẩn STL của các hệ thống lập trình dựa trên C++ chưa trực tiếp hỗ trợ Fibonacci Heap.

Các giải thuật dựa trên Set hoặc Priority\_Queue tuy không hiệu quả bằng sử dụng Fibonacci heap nhưng cũng cho độ phức tạp khá tốt, đủ chấp nhận được –  $O(mlogn)$ .

Với cấu trúc tập hợp (Set), mỗi đơn vị dữ liệu input cần được tổ chức dưới dạng cặp số nguyên (**pair<int, int>**), phần tử thứ nhất là trọng số và phần tử thứ hai là đỉnh của cạnh. Dữ liệu sẽ được tự động sắp xếp theo trọng số tăng dần – điều mà ta đang cần! Việc tổ chức mảng đánh dấu  $U$  cũng trở nên không cần thiết. Khi điều chỉnh  $D$ , mỗi khi có sự thay đổi, trước hết cần xóa cặp dữ liệu cũ, tính lại  $d_t$  và nạp lại cặp dữ liệu mới ứng với  $d_t$  vừa tính được.

Chương trình làm việc với hàng đợi ưu tiên hoạt động nhanh hơn một chút so với phương án sử dụng tập hợp. Tuy vậy, theo bản chất của cấu trúc dữ liệu, hệ thống không cung cấp dịch vụ xóa thông tin nếu nó không đứng ở đầu hàng đợi. Chính vì vậy phần lớn các giải thuật sử dụng hàng đợi (kể cả hàng đợi ưu tiên) đều phải giải quyết vấn đề *lọc dữ liệu thừa* trong quá trình xử lý. Trong giải thuật này, việc đó đơn thuần là so sánh giá trị lưu trữ ở đầu hàng đợi  $q$  với  $d_v$ . Khi chỉnh lý  $D$ , cặp giá trị ( $d_t, t$ ) được nạp vào hàng đợi. Cặp giá trị mới này sẽ đứng trước các cặp khác có cùng giá trị  $t$ .

Cần lưu ý là với khai báo **priority\_queue < pair<int, int> > q**; việc tổ chức ngầm định sẽ đặt giá trị lớn nhất lên đầu hàng đợi. Muốn có hàng đợi sắp xếp theo thứ tự tăng dần ta cần khai báo:

```
typedef p2 pair<int,int>;
priority_queue <p2,vector<p2>,greater<p2> > q;
```

Trong giải thuật này các giá trị khóa đều không âm, vì vậy ta có thể dùng khai báo đơn giản theo kiểu ngầm định, nhưng nạp giá trị khóa âm. Kết quả là giá trị thực sự nhỏ nhất vẫn đứng ở đầu hàng đợi.

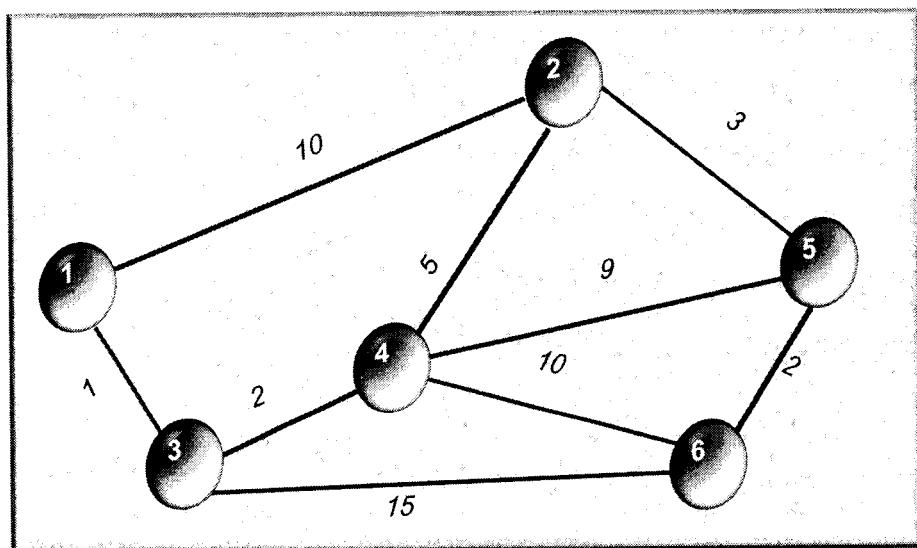
Chương trình sau giải bài toán đã nêu với dữ liệu đưa vào từ file Dijkstra.inp theo quy cách:

- ◆ Dòng đầu tiên chứa 2 số nguyên  $n$  và  $m$  ( $n > 0, m \geq 0$ ),
- ◆ Nếu  $m > 0$  thì mỗi dòng trong  $m$  dòng tiếp theo chứa 3 số nguyên  $a, b$  và  $r$  cho biết có cạnh nối từ  $a$  tới  $b$  với trọng số  $r$  ( $1 \leq a, b \leq n, a \neq b, 0 \leq r \leq 10^6$ ), không có hai dòng nào giống nhau,
- ◆ Dòng  $m+2$  chứa số nguyên  $s$  ( $1 \leq s \leq n$ ),
- ◆ Dòng cuối cùng chứa số nguyên  $k$  và sau đó là  $k$  số nguyên  $c_1, c_2, \dots, c_k$  cho biết phải dẫn xuất đường đi ngắn nhất từ  $s$  tới  $c_i$ ,  $i = 1 \div k, 1 \leq c_i \leq n$ .

Kết quả được đưa ra file Dijkstra.out:

- Dòng đầu tiên chứa  $n$  số nguyên, số thứ  $i$  là độ dài đường đi ngắn nhất từ  $s$  đến đỉnh  $i$ . Độ dài bằng -1 nếu không tồn tại đường đi từ  $s$  đến đỉnh đó,
- Dòng thứ  $i$  trong  $k$  dòng sau chứa thông tin về đường đi ngắn nhất (theo quy cách nêu trong ví dụ).

Ví dụ: xét đồ thị



Các file dữ liệu Input và output:

DIJSKTRA.INP
6 9
1 2 10

DIJSKTRA.OUT
8 0 7 5 3 5
Shortets route from 2 to 1: 2 4 3 1

1	3	1
2	4	5
3	4	2
2	5	3
3	6	15
4	6	10
4	5	9
5	6	2
2		
2	1	6

<b>Shortets route from 2 to 6: 2 5 6</b>
<b>Time: 0 sec</b>

Chương trình trên C++ sử dụng cấu trúc dữ liệu *Hàng đợi ưu tiên*:

```
#include <fstream>
#include <vector>
#include <queue>
#include <ctime>
using namespace std;
const int INF = 1000000000;
ifstream fi ("Dijkstra.inp");
ofstream fo ("Dijkstra.out");
pair<int,int>x;
int a,b,dd,k;
int main()
{ clock_t aa=clock();
  int n,m;
  fi>>n>>m;
  vector < vector < pair<int,int> > > g (n+1);
  for(int i=1;i<=m;++i)
    {fi>>a>>b>>dd;x.first=a;x.second=dd;
     g[b].push_back(x); x.first=b;g[a].push_back(x);
    }
  int s ;
  fi>>s;
  vector<int> d (n+1, INF), p (n+1);
  d[s] = 0;
  priority_queue < pair<int,int> > q;
  q.push (make_pair (0, s));
  while (!q.empty()) {
    int v = q.top().second, cur_d = -q.top().first;
    q.pop();
    if (cur_d > d[v]) continue;

    for (size_t j=0; j<g[v].size(); ++j) {
      int to = g[v][j].first,
          len = g[v][j].second;
      if (d[v] + len < d[to]) {
        d[to] = d[v] + len;
        p[to] = v;
        q.push (make_pair (-d[to], to));
      }
    }
  }
}
```

```

        }
    }
    for(int i=1;i<=n;++i) if(d[i]<INF) fo<<d[i]<<" "; else fo<<-1<<" ";
    fi>>k;
    for(int i=0;i<k;++i)
    {int t;
     fi>>t;
     vector<int> path;
     fo<<"\n Shortets route from "<<s<<" to "<<t<<": ";
     for (int j=t; j!=s; j=p[j])
      path.push_back (j);
     path.push_back (s);
     reverse (path.begin(), path.end());
     for (size_t j=0; j<path.size(); ++j) fo<<path[j]<<" ";
    }
    clock_t bb=clock();
    fo<<"\nTime: "<<(double)(bb-aa)/1000<<" sec";
}

```

### **Ghi chú:**

- ◆ Với mục đích khảo sát và so sánh hiệu quả cài đặt, chương trình có đưa ra thời gian thực hiện với độ chính xác mili giây,
- ◆ Thời gian thực hiện chương trình còn có thể giảm xuống nếu thay việc nhập dữ liệu theo kiểu stream bằng nhập theo quy cách và tổ chức vòng tránh sử dụng dữ liệu dạng cặp (Pair).

Kiểu cài đặt dùng cấu trúc dữ liệu Tập hợp tuy kém hiệu quả hơn đôi chút, nhưng cũng đáng để tham khảo.

Chương trình trên C++ sử dụng cấu trúc dữ liệu kiểu *Tập hợp*:

```

#include <fstream>
#include <vector>
#include <queue>
#include <set>
#include <ctime>
using namespace std;
const int INF = 1000000000;
ifstream fi ("Dijkstra.inp");
ofstream fo ("Dijkstra.out");
pair<int,int>x;
int a,b,dd,k;
int main()
{ clock_t aa=clock();
  int n,m;
  fi>>n>>m;
  vector < vector < pair<int,int> > > g (n+1);
  for(int i=1;i<=m;++i)
   {fi>>a>>b>>dd;x.first=a;x.second=dd;
    g[b].push_back(x); x.first=b;g[a].push_back(x);
   }
  int s ;
  fi>>s;
  vector<int> d (n+1, INF), p (n+1);
  d[s] = 0;

  set < pair<int,int> > q;
  q.insert (make_pair (d[s], s));
  while (!q.empty()) {
   int v = q.begin()->second;
   q.erase (q.begin());

```

```

        for (size_t j=0; j<g[v].size(); ++j) {
            int to = g[v][j].first,
                len = g[v][j].second;
            if (d[v] + len < d[to]) {
                q.erase (make_pair (d[to], to));
                d[to] = d[v] + len;
                p[to] = v;
                q.insert (make_pair (d[to], to));
            }
        }
    }

for(int i=1;i<=n;++i)if(d[i]<INF) fo<<d[i]<<" "; else fo<<-1<<" ";
fi>>k;
for(int i=0;i<k;++i)
{int t;
fi>>t;
vector<int> path;
fo<<"\n Shortets route from "<<s<<" to "<<t<<": ";
for (int j=t; j!=s; j=p[j])
    path.push_back (j);
path.push_back (s);
reverse (path.begin(), path.end());
for (size_t j=0; j<path.size(); ++j) fo<<path[j]<<" ";
}
clock_t bb=clock();
fo<<"\nTime: "<<(double)(bb-aa)/1000<<" sec";
}

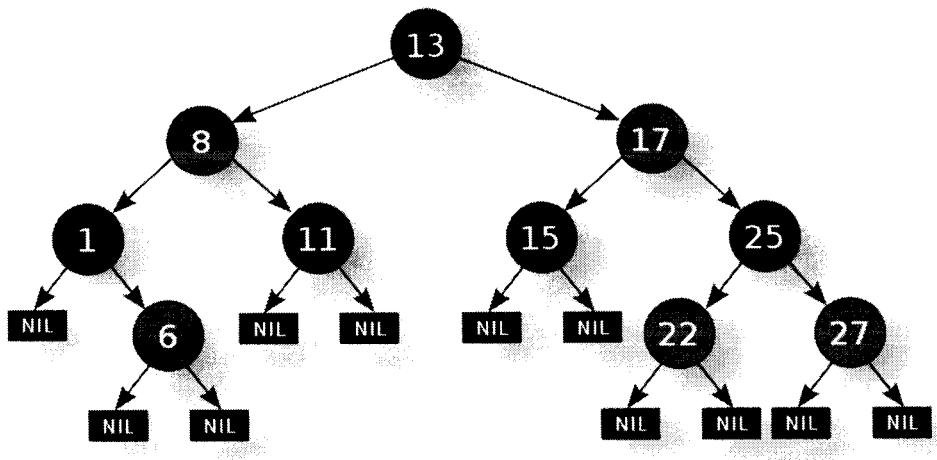
```

Khác với PASCAL, cấu trúc dữ liệu Set trong C/C++ hoạt động khá hiệu quả bởi vì nó (và cả cấu trúc dữ liệu Map) được xây dựng dựa trên cơ sở cây tìm kiếm Đỏ – Đen. Tuy vậy, trong phạm vi báo cáo này, ta sẽ *không dùng lại kỹ* ở cơ sở lý thuyết này, mặc dù đó là mảng kiến thức quan trọng làm nền tảng cho việc xây dựng một loạt các giải thuật hiệu quả cao giải quyết các bài toán có mô hình đồ thị.

## 4.2 – ĐỒ THỊ HAI MÀU

### 4.2.1 – Khái niệm chung

Đồ thị hai màu là loại đồ thị mà mỗi đỉnh được tô bằng một trong hai màu và mỗi cạnh của đồ thị nối 2 đỉnh có màu khác nhau. Đồ thị này còn thường được gọi là đồ thị Đỏ - Đen. Đồ thị này có ứng dụng rộng rãi trong lý thuyết cũng như trong các bài toán thực tế. Nổi tiếng hơn cả là Cây Đỏ - Đen (*Red-Black-Tree, RB-Tree* ). Đó là một trong số các loại cây nhị phân tìm kiếm tự cân bằng có độ dãn nở bậc lô ga rit khi số đỉnh tăng và cho phép thực hiện các phép chỉnh lý cây khi thêm, bớt đỉnh và tìm kiếm với độ phức tạp  $O(\log n)$ .



Cây Đỏ - Đen

Định nghĩa, tính chất, các phép xử lý cơ bản và ứng dụng cây Đỏ - Đen được trình bày ở rất nhiều tài liệu khác nhau và dễ tiếp cận.

Trong phần này ta chỉ xem xét một lớp đồ thị riêng đòi hỏi sự chú ý của người lập trình khi giải các bài toán liên quan tới lớp đồ thị này.

Nhiều giải thuật xử lý đồ thị hai màu được mở rộng và triển khai có hiệu quả cho đồ thị hai phía.

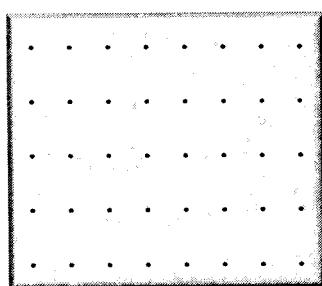
Với đồ thị hai màu, việc xác định cấu trúc và đưa ra dạng biểu diễn cấu trúc, duyệt đồ thị, xây dựng cây khung, . . . đơn giản hơn và có những giải pháp rất hiệu quả. Các vấn đề này sẽ được minh họa thông qua việc xét một bài toán cụ thể – bài toán *Dấu vết trên tuyết* (Olympic Tin học Khu vực Baltic năm 2013 – BOI 2013).

## BÀI TOÁN

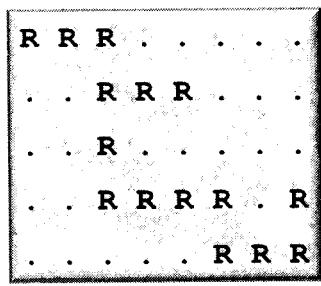
### DẤU VẾT TRÊN TUYẾT

Bài cỏ giữa rừng có hình chữ nhật kích thước  $h \times w$  ô, sau một đêm bị tuyết phủ trắng xóa. Trời sáng, các con thỏ và cáo trong rừng thức dậy đi kiếm ăn và băng qua bài cỏ. Chúng luôn luôn vào bài cỏ từ góc trên trái và rời khỏi bài cỏ ở góc dưới phải. Ở mỗi bước con vật chỉ có thể nhảy sang ô kè cạnh và để lại dấu vết của mình trên ô đó. Dấu vết thỏ để lại khác với dấu vết của cáo. Khi nhảy tới ô đã có dấu vết thì dấu vết mới sẽ đè dấu vết cũ và người ta chỉ quan sát thấy dấu vết mới. Mỗi con vật có thể đi tới đi lui, sang phải sang trái, vui đùa trên bài cỏ, qua lại nhiều lần một số ô đã đi qua. Ở mỗi thời điểm trên bài cỏ có không quá một con vật. Không có con nào đi qua bài cỏ hai lần trở lên. Ví dụ, ban đầu có một chú thỏ đi qua và sau đó là một con cáo. Trạng thái ô không có con vật nào đi qua được đánh dấu bằng ký tự “.”, dấu vết của thỏ - ký tự “R” và dấu vết của cáo - ký tự “F”. Dấu vết quan sát được trên bài cỏ là như sau:

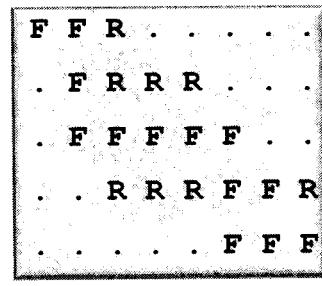
Cho bức tranh dấu vết quan sát được. Hãy xác định số lượng tối thiểu các con vật đã đi qua.



Trạng thái bài cỏ ban đầu



Trạng thái sau khi thỏ đi qua



Trạng thái sau khi cáo đi qua

**Dữ liệu:** Vào từ file văn bản TRACKS.INP:

- Đòng đầu tiên chứa 2 số nguyên  $h$  và  $w$  ( $1 \leq h, w \leq 4000$ ),
- Mỗi dòng trong  $h$  dòng sau chứa xâu độ dài  $w$  từ tập ký tự  $\{., R, F\}$  mô tả một hàng dấu vết trên bài cỏ.

**Kết quả:** Đưa ra file văn bản TRACKS.OUT một số nguyên – số lượng tối thiểu các con vật đã đi qua.

**Ví dụ:**

TRACKS.INP
5 8
FFR.....
.FRRR...
.FFFFF..
..RRRF
....FFF

TRACKS.OUT
2

## CƠ SỞ TOÁN HỌC CỦA GIẢI THUẬT

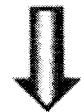
Các ô kề cạnh có cùng một loại dấu vết tạo thành một miền liên thông. Nói một cách khác, trên bảng dữ liệu vào các ô kề cạnh chứa cùng một loại ký tự và khác ký tự dấu chấm ('.') tạo thành một miền liên thông. Giả thiết ta có tất cả  $k$  miền liên thông, đánh số từ 1 đến  $k$ . Miền liên thông có chứa ô ở góc trên trái được đánh số là 1.

Ví dụ, bảng dữ liệu vào có kích thước  $5 \times 13$  với nội dung:

R	F	.	F	R	F	R	F	.	F	R	F	R
R	.	F	R	F	R	.	R	F	R	.	R	F
R	F	.	F	R	F	R	F	.	F	R	F	.
R	.	F	.	F	.	F	.	.	.	.	.	F
R	R	R	R	R	R	R	R	R	R	R	R	R

Ta có các miền liên thông:

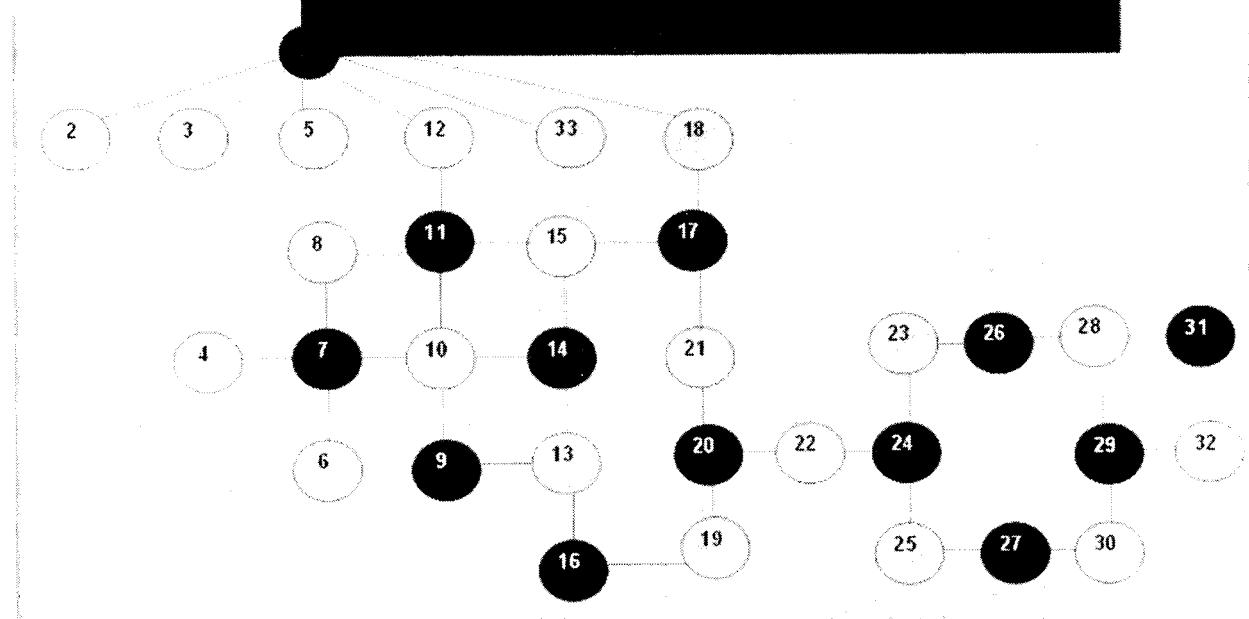
R	F	.	F	R	F	R	F	.	F	R	F	R
R	.	F	R	F	R	.	R	F	R	.	R	F
R	F	.	F	R	F	R	F	.	F	R	F	.
R	.	F	.	F	.	F	.	.	.	.	.	F
R	R	R	R	R	R	R	R	R	R	R	R	R



1	2	.	6	9	13	16	19	.	23	26	28	31
.	4	7	10	14	.	20	22	24	.	29	32	.
3	.	8	11	15	17	21	.	25	27	30	.	.
.	5	.	12	.	18	.	.	.	.	.	.	33

Coi mỗi miền liên thông là một đỉnh của đồ

thông thị và



giữa hai đỉnh có cạnh nối nếu hai miền liên thông này kề nhau, ta có một đồ thị hai màu:

Từ đồ thị 2 màu này ta có thể xây dựng cây khung Đỏ - Đen với nút gốc là 1 sao cho đường đi dài nhất từ nút gốc tới lá là ngắn nhất. Độ dài mỗi cạnh của đồ thị được coi là như nhau và bằng 1. Ta sẽ có cây:

Số lượng nút trên đường đi dài nhất này chính là lời giải bài toán.

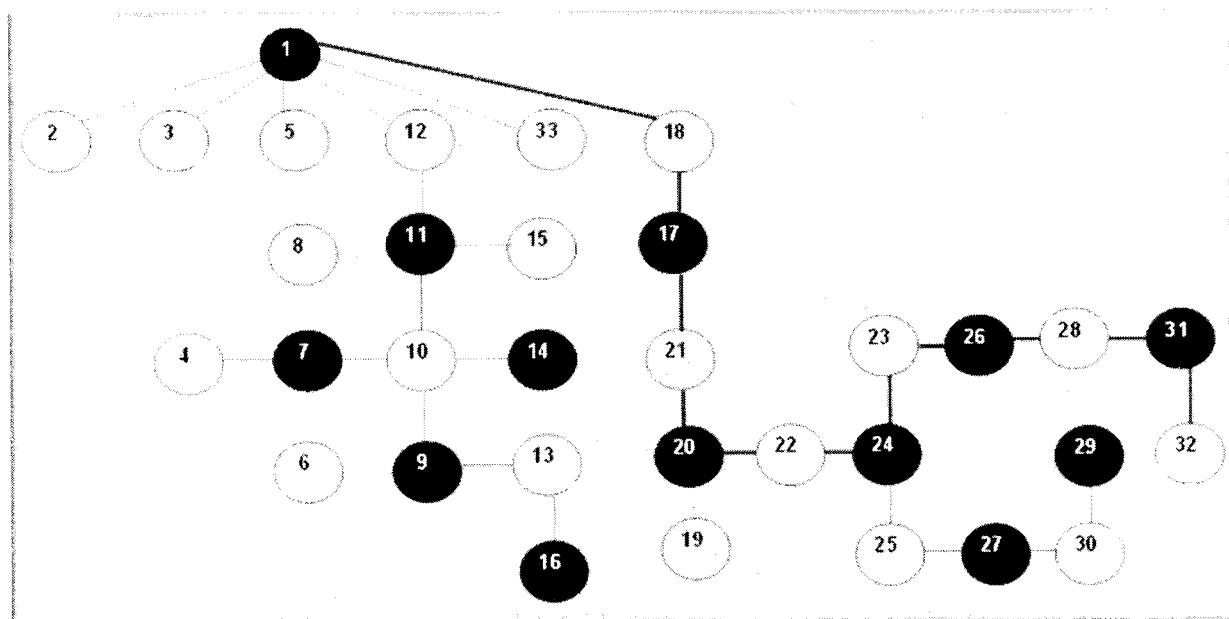
Việc xây dựng cây khung thỏa mãn yêu cầu là khá rắc rối, vì vậy, sau khi xây dựng xong đồ thị có thể dùng *giải thuật Dijkstra tìm độ dài đường đi ngắn nhất từ đỉnh 1 đến các đỉnh còn lại*. Độ dài lớn nhất trong số các độ dài tìm được cộng 1 sẽ là lời giải của bài toán.

## SƠ ĐỒ CÀI ĐẶT TỐI ƯU

Mặc dù có thể cải tiến giải thuật Dijkstra cho trường hợp trọng số của tất cả các cạnh là như nhau nhưng việc xây dựng tường minh đồ thị và sau đó – xử lý nó là không tối ưu bởi vì:

- ➔ Số đơn vị dữ liệu cần xử lý ở đây là rất lớn, có thể lên tới  $4000 \times 4000 = 16 \times 10^6$ ,
- ➔ Số cạnh của đồ thị, trong trường hợp xấu nhất – cũng có thể xấp xỉ  $16 \times 10^6$ ,
- ➔ Chưa khai thác hết mọi tính chất của đồ thị 2 màu.

Việc tổ chức loang theo chiều rộng trực tiếp bằng giải thuật đệ quy, tuy rất ngắn gọn, dễ cài đặt nhưng không đáp ứng được các ràng buộc thực tế về bộ nhớ cần sử dụng và thời gian thực hiện. Mỗi đơn vị dữ liệu input chỉ duyệt có 5 lần nhưng sẽ *không có đủ bộ nhớ để lưu vết* đệ



quy và thời gian quay lui rẽ nhánh là cực lớn.

Chương trình tối ưu cần phải hoạt động với độ phức tạp  $O(m \times n)$ .

Để đạt được hiệu quả này cần sử dụng Hàng đợi hai đầu (**Deque** – Double End Queue). Như đã nói ở phần trên, việc sử dụng Hàng đợi (**Queue** hoặc **Deque**) đều tiềm ẩn nguy cơ thông tin nhiều. Vì vậy, việc lọc dữ liệu trong quá trình xử lý là hết sức cần thiết. Nếu không lọc nhiều, độ phức tạp của giải thuật có thể trở thành  $O(m^2 \times n^2)$ !

### Tổ chức dữ liệu

Các dữ liệu cơ bản là:

- ◆ Mảng  $a[4000][4000]$  kiểu **char** – lưu dữ liệu input,
- ◆ Mảng  $b[4000][4000]$  kiểu **int** – bản đồ đánh dấu đỉnh tiềm năng và phục vụ loang theo đồ thị,  $b_{i,j} = 0$  là điểm chưa xét,  $b_{i,j} = -1$  – điểm xuất phát tiềm năng,  $b_{i,j} > 0$  – điểm đã được xử lý,
- ◆ Hàng đợi  $p$  kiểu **deque** lưu các điểm tiềm năng cần thăm và duyệt tiếp, mỗi phần tử của hàng đợi là một cặp dữ liệu ( $i, j$ ).

### Xử lý

- ◆ Với mỗi vị trí ( $i, j$ ) có  $a_{i,j} \neq '.'$  và cần để lại xử lý sau – phân biệt các trường hợp:
  - ✓ Cùng giá trị với ô đang xử lý – nạp ( $i, j$ ) vào cuối hàng đợi ( $p.push\_back(pp);$ )
  - ✓ Khác giá trị với ô đang xử lý – nạp ( $i, j$ ) vào đầu hàng đợi ( $p.push\_front(pp);$ )
- ◆ Trong mọi trường hợp – cần đánh dấu  $b_{i,j} = -1$
- ◆ Mỗi lần chuyển sang duyệt giá trị mới – tăng bộ đếm lớp lên 1.
- ◆ Bộ đếm lớp sẽ là kết quả cần tìm.
- ◆ Việc kiểm tra giá trị  $b_{i,j} > 0$  cho phép loại ngay khỏi hàng đợi các điểm tiềm năng trở thành nhiễu.

## Chương trình

```
#include <deque>
#include <vector>
#include <algorithm>
#include <ctime>
#include <string>
#include <cstring>
using namespace std;
typedef pair<int,int> p2;
int w,h,k,b[4002][4002]={0},i1,j1;
char c,c1,ct,a[4002][4002]={'.'};
pair<int,int>pp;
deque <p2> p;
string s;
ifstream fi ("tracks.inp");
ofstream fo ("tracks.out");

void swp()
{ct=c;c=c1; c1=ct; }

void upd_p(int x,int y)
{if(x>1 && b[x-1][y]==0 && a[x-1][y]!='.')
 {pp.first=x-1;pp.second=y;if(a[x-1][y]==c)p.push_back(pp); else
 p.push_front(pp);b[x-1][y]==-1;}
 if(x<h && b[x+1][y]==0 && a[x+1][y]!='.')
 {pp.first=x+1;pp.second=y;if(a[x+1][y]==c)p.push_back(pp); else
 p.push_front(pp);b[x+1][y]==-1;}
 if(y>1 && b[x][y-1]==0 && a[x][y-1]!='.')
 {pp.first=x;pp.second=y-1;if(a[x][y-1]==c)p.push_back(pp); else
 p.push_front(pp);b[x][y-1]==-1;}
 if(y<w && b[x][y+1]==0 && a[x][y+1]!='.')
 {pp.first=x;pp.second=y+1;if(a[x][y+1]==c)p.push_back(pp); else
 p.push_front(pp);b[x][y+1]==-1;}
 }

int main()
{clock_t aa=clock();
 k=1;
 fi>>h>>w;
 for(int i=1;i<=h;++i)
 {fi>>s; for(int j=1;j<=w;++j)a[i][j]=s[j-1];}
 c=a[1][1]; if(c=='F')c1='R'; else c1='F';
 pp.first=1; pp.second=1; p.push_back(pp);
 while(!p.empty())
 {pp=p.back();
 i1=pp.first;j1=pp.second;
 if(b[i1][j1]<=0)
 {{if(a[i1][j1]==c)b[i1][j1]=k; else {b[i1][j1]=++k; swp();}}
 p.pop_back();upd_p(i1,j1);
 }
 else p.pop_back();
 }

fo<<k<<'\n';
 clock_t bb=clock();
 fo<<"Time: "<<(double) (bb-aa)/1000<<" sec";
}
```

### **Nhận xét:**

- ◆ Nguyên tắc xử lý trên cho phép thực hiện loang theo chiều rộng một cách hiệu quả đối với đồ thị hai màu cho tường minh, trong trường hợp này cần thêm một hàng đợi phục vụ lưu trữ cấu trúc đồ thị,
- ◆ Giải thuật xử lý đã nêu cho phép tạo cây khung thỏa mãn các yêu cầu đã nêu trong phần cơ sở toán học (phục vụ cho các yêu cầu xử lý khác nếu có),
- ◆ Thời gian chạy chương trình cho trường hợp  $m, n$  bằng 4000 tinh huống đủ xấu –  $\approx 2$  giây trên CPU Cor i5,
- ◆ Có thể tổ chức khai báo động các mảng  $a$  và  $b$  theo kích thước cụ thể cho trong file input.

## VL07. THI ĐẤU

Tên chương trình: TOURNAMENT.???

Để các đội có dịp cọ xát nâng cao tay nghề người ta tổ chức cuộc thi Khúc côn cầu cho toàn vùng Ural. Có  $n$  đội đăng ký tham gia. Các đội đánh số từ 1 đến  $n$ ,  $n$  – chẵn. Cuộc thi đã diễn ra 2 vòng, ở mỗi vòng mỗi đội chơi đúng một trận. Ban tổ chức thấy vẫn còn quá nhiều đội sẽ vào vòng sau theo luật ban đầu và quyết định ở vòng sau chỉ chọn  $k$  đội mà 2 đội bất kỳ trong số đó chưa gặp nhau lần nào ở 2 vòng đầu.

Hãy xác định danh sách các đội được chọn. Nếu có nhiều cách chọn thì đưa ra một trong số các cách đó. Nếu không có cách chọn thì đưa ra số 0.

**Dữ liệu:** Vào từ file văn bản TOURNAMENT.INP:

- ◆ Dòng đầu tiên chứa số nguyên  $n$  ( $2 \leq n \leq 10^5$ ,  $n$  – chẵn),
- ◆  $n/2$  dòng tiếp: theo mỗi dòng chứa 2 số nguyên xác định các độ gặp nhau ở vòng I,
- ◆  $n/2$  dòng sau: theo mỗi dòng chứa 2 số nguyên xác định các độ gặp nhau ở vòng II,
- ◆ Dòng cuối cùng chứa số nguyên  $k$  ( $2 \leq k \leq n$ ).

**Kết quả:** Đưa ra file văn bản TOURNAMENT.OUT số nguyên 0 nếu không có cách chọn hoặc  $k$  số nguyên xác định danh sách các đội được chọn.

**Ví dụ:**

TOURNAMENT.INP
6
1 2
3 5
4 6
2 3
4 5
1 6
3

TOURNAMENT.OUT
1 4 3



```

#include<fstream>
int
a[100001][2]={0},b[100001]={0},c[100001]={0},d[100001]={0},n,k,x,y,jc
;
using namespace std;
ifstream fi ("Tournament.inp");
ofstream fo ("Tournament.out");

void determ_lp()
{int j,p,q;
j=jc;q=1;
while(1)
{ p=a[j][0];
  if(c[p]==0){b[j]=p;c[p]=1;d[j]=q; q=-q;j=p;}
  else {p=a[j][1]; if(c[p]==0){b[j]=p;c[p]=1;d[j]=q; q=-q;j=p;}
         else {b[j]=jc;d[j]=-1;return;}}
}
}

int get_jc()
{int p;
p=0;
for(int i=jc;i<=n;++i) if(c[i]==0){p=i; break;}
return p;
}

int main()
{fi>>n;
for(int i=1;i<=n;++i)
{fi>>x>>y;
 if(a[x][0]==0)a[x][0]=y; else a[x][1]=y;
 if(a[y][0]==0)a[y][0]=x; else a[y][1]=x;
}
fi>>k;
if (k>n/2){fo<<0; return 0;}
jc=1;
while(jc>0)
{ c[jc]=1;d[jc]=1;
  determ_lp();
  jc=get_jc();
}
x=0;
for(int i=1;i<=n;++i)
  if(d[i]==1){fo<<i<<" "; if(++x==k)break;}
}

```

## 5. Xử lý xâu

Một số hàm xử lý xâu thường dùng trong thư viện:

**s.size()** , s.length() – độ dài xâu,

**s.find(s1)** – Vị trí đầu tiên xuất hiện **s1** như xâu con của **s**, nếu **s1** không là xâu con của **s** → kết quả là -1,

**s.rfind(s1)** – Vị trí cuối cùng xuất hiện **s1** như xâu con của **s**, nếu **s1** không là xâu con của **s** → kết quả là -1,

**s.find\_first\_of(c)** – Vị trí đầu tiên xuất hiện ký tự **c** trong **s**, nếu **c** không có trong **s** → kết quả là -1,

**s.substr(i, j)** – cho xâu con độ dài **j** của **s**, bắt đầu từ vị trí **i**.

Tên biến kiểu xâu

Một số hàm khác:  
xem trong các ví dụ.

```
#include <fstream>
#include <string>
using namespace std;
int main()
{ string s="ABCD12345abcdABCDE2";
  string ns="0123456789";
  int l,k,m,p,q,i,j;
  ofstream fo ("xl_kau.out");
  l=s.size();
  k=s.find("BC");
  m=s.rfind("BC");
  p=s.find_first_of('C');
  q=s.find_last_of('C');
  i=s.find_first_of(ns);
  j=s.find_last_not_of(ns);
  fo<<"L = "<<l<<endl;
  fo<<"K = "<<k<<endl;
  fo<<"M = "<<m<<endl;
  fo<<"P = "<<p<<endl;
  fo<<"Q = "<<q<<endl;
  fo<<"I = "<<i<<endl;
  fo<<"J = "<<j<<endl;
  fo.close(); }
```

L	=	19
K	=	1
M	=	14
P	=	2
Q	=	15
I	=	4
J	=	17

fo<<"P = "<<p<<endl;  
fo<<"Q = "<<q<<endl;  
fo<<"I = "<<i<<endl;  
fo<<"J = "<<j<<endl;  
fo.close();

Chuyển đổi kiểu dữ liệu:

◆ Chuyển số nguyên sang dạng xâu:

```
int a = 10;
stringstream ss;
ss << a;
string str = ss.str();
```

```
#include <sstream>
#include <string>
using namespace std;

string myStream = "45";
istringstream buffer(myString);
int value;
buffer >> value; // value = 45
```

◆ Chuyển xâu sang dạng số nguyên:

## VL50. QUAN HỆ HỌ HÀNG

Tên chương trình: RELATIVES. ???

Để theo dõi thực trạng đói sống của động vật hoang dã phục vụ cho công tác bảo tồn thiên nhiên người ta gắn cho mỗi động vật bắt được một con chíp, trong đó có chứa mã phân loại. Theo dõi đường di chuyển và hoạt động hàng ngày của chúng người ta biết được rất nhiều thông tin, trong đó có việc ra đói của các động vật thế hệ F2, F3, ... Mã phân loại là một số thập phân nguyên dương không vượt quá  $10^9$ . Các cặp mã phân loại đảm bảo nếu hai cá thể có quan hệ họ hàng với nhau thì hai mã phải có ít nhất một chữ số giống nhau. Ví dụ 2 con vật có các mã phân loại tương ứng là 47 và 107 có quan hệ họ hàng với nhau, còn hai con vật với các mã 47 và 931 – không có quan hệ.

Cho  $n$  mã phân loại khác nhau từng đôi một. Hãy xác định số cặp động vật có quan hệ họ hàng.

**Dữ liệu:** Vào từ file văn bản RELATIVE.INP:

- ◆ Dòng đầu tiên chứa số nguyên  $n$  ( $2 \leq n \leq 5 \times 10^5$ ),
- ◆ Dòng thứ 2 chứa  $n$  mã phân loại.

**Kết quả:** Đưa ra file văn bản RELATIVE.OUT số cặp động vật xác định được.

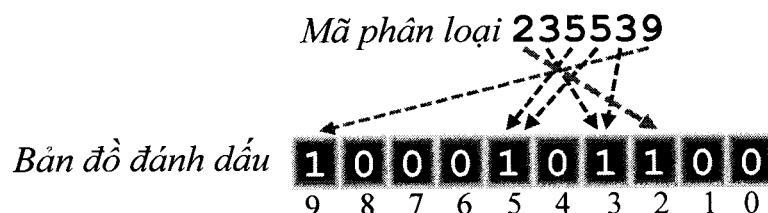
**Ví dụ:**

RELATIVES.INP	RELATIVES.OUT
5	
10 74 47 77 301	4



### Giải thuật

- ◆ Với mỗi mã phân loại cần lọc dữ liệu thừa: nếu một chữ số xuất hiện nhiều lần thì chỉ giữ lại một làm đại diện, mã ban đầu được biến đổi thành mã đại diện tương ứng,
- ◆ Do vị trí xuất hiện chữ số không đóng vai trò quan trọng nên có thể dùng bản đồ để đánh dấu sự có mặt của các chữ số trong mã đại diện,
- ◆ Mỗi mã đại diện sẽ được đặt tương ứng với một số nguyên 10 bit, xác định một lớp tương đương,
- ◆ Tổng cộng có 1024 lớp tương đương với giá trị tương ứng từ 0 đến 1023.



- Với mỗi lớp tương đương  $i$  ( $i = 0 \div 1023$ ) xác định  $b_i$  – số lượng mã thuộc lớp  $i$ ,
- Tính cặp quan hệ họ hàng:
  - Điều kiện 2 lớp  $i$  và  $j$  có quan hệ họ hàng:  $i \text{ and } j \neq 0$ ,
  - Nếu 2 lớp  $i$  và  $j$  có quan hệ thì số cặp quan hệ có thể xác lập sẽ là:
    - $b_i \times b_j$ , nếu  $i \neq j$ ,
    - $(b_i \times (b_i - 1))/2$ , nếu  $i = j$ .

Độ phức tạp giải thuật:  $O(n)$ .

### *Chương trình giải:*

```
#include <fstream>
#include <ctime>
#include <cstring>
using namespace std;
int64_t ans=0,b[1024]={0};
string s;
int n,x,ls;
ifstream fi ("Relatives.inp");
ofstream fo ("Relatives.out");

int main()
{clock_t aa=clock();
 fi>>n;
for(int i=1;i<=n;++i)
{fi>>s; ls=s.size();
 x=0;
 for(int j=0;j<ls;++j)x|=(1<<(s[j]-48));
 ++b[x];
}
for(int i=0;i<1024;++i)
for(int j=i;j<1024;++j)
if((i&j)!=0)
{if(i!=j)ans+=b[i]*b[j];
 else ans+=(b[i]*(b[i]-1))/2;
}
fo<<ans;
clock_t bb=clock();
fo<<"\nTime: "<<(double)(bb-aa)/1000<<" sec";
}
```

## VO22. NGÀY NGHỈ

Tên chương trình: HOLYDAYS.???

Công tác bảo vệ tại các khu công nghiệp được thực hiện 24/24 giờ tất cả các ngày trong năm, vì vậy mỗi nhân viên trong đội bảo vệ có lịch nghỉ riêng quy định ngay từ trước khi năm mới bắt đầu và có dạng là ngày thứ mấy nào đầu tiên hay cuối cùng của một tháng cụ thể nào đó, ví dụ “ngày thứ 2 đầu tiên của tháng giêng” hay “ngày thứ tư cuối cùng của tháng sáu”.

Lịch nghỉ của một người có **n** ngày, mỗi ngày nghỉ được nêu dưới dạng “first *thứ* of *tháng*” hoặc “last *thứ* of *tháng*”, trong đó *thứ* nhận một trong số các giá trị **monday**, **tuesday**, **wednesday**, **thursday**, **friday**, **saturday**, **sunday**, *tháng* có các giá trị lần lượt là **january**, **february**, **march**, **april**, **may**, **june**, **july**, **august**, **september**, **october**, **november**, **december**.

Cho **n**, thứ của ngày đầu tiên trong năm, thông báo “**yes**” nếu đó là năm nhuận và “**no**” trong trường hợp ngược lại, các ngày nghỉ theo dạng đã nêu. Hãy xác định ngày tương ứng trong tháng mà người đó được nghỉ.

**Dữ liệu:** Vào từ file văn bản HOLYDAYS.INP:

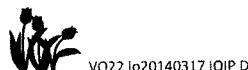
- ◆ Dòng đầu tiên chứa số nguyên **n** ( $1 \leq n \leq 168$ ),
- ◆ Dòng thứ 2 chứa thông tin về thứ của ngày đầu năm và năm đó có nhuận hay không,
- ◆ Mỗi dòng trong **n** dòng sau chứa thông tin về một ngày nghỉ.

**Kết quả:** Đưa ra file văn bản HOLYDAYS.OUT các ngày được nghỉ trong tháng tương ứng, mỗi ngày trên một dòng.

**Ví dụ:**

HOLYDAYS.INP
5
wednesday no
last monday of january
first friday of february
last saturday of
september
first sunday of july
first monday of january

HOLYDAYS.OUT
27
7
27
6
6



VO22 | 020140317 | OIP D

## *Giải thuật*

- ◆ Ngày trong tuần: đánh số từ 0 đến 6,
- ◆ Tạo mảng  $y[367]$ ,  $y_i$  – thứ của ngày  $i$ ,
- ◆ Tạo các mảng hàng phục vụ tìm ngày đầu tiên của tháng bất kỳ, phục vụ chuyển dạng biểu diễn dữ liệu,
- ◆ Duyệt tiến hoặc lùi (không quá 7 giá trị) để tìm lời giải.

```
#include <fstream>
#include <ctime>
using namespace std;
ifstream fi ("holydays.inp");
ofstream fo ("holydays.out");
int sng1[13]={0,31,59,90,120,151,181,212,243,273,304,334,365},
      sng2[13]={0,31,60,91,121,152,182,213,244,274,305,335,366};

string
thu="monday****tuesday***wednesday*thursday**friday****saturday**sunday*****",
thg="january***february**march*****april*****may*****june*****july
*****august****september*october***november**december**";
string s1,s2,s3,s4;
int y[367],ng,th,fl,n,nh,t1,t2,ans;

void get_ans()
{ fl=s1=="last";
  ng=thu.find(s2)/10;
  th=thg.find(s4)/10+1;
  if(nh==0)t1=sng1[th]-sng1[th-1];else t1=sng2[th]-sng2[th-1];
  if(fl==0){if(nh==0)t2=sng1[th-1];else t2=sng2[th-1];}
  else {if(nh==0)t2=sng1[th];else t2=sng2[th];}
  if(fl==0) {for(int i=1; i<=t1;++i) if(y[t2+i]==ng){ans=i; break;}}
  else for(int i=0;i<t1;++i) if(y[t2-i]==ng){ans=t1-i; break;}
  fo<<ans<<'\n';
}
```

## VN44. THƠ CÓ ĐỘNG

Tên chương trình: LYRICS.???

Thể vận hội mùa đông Xô Tri 2014 đã khai mạc. Misa quyết định sáng tác một bài thơ để cả lớp cùng hát ở sân vận động, cổ vũ cho đội nước nhà. Quy tắc gieo vần của Misa là như sau: nếu 2 câu cách nhau đúng  $n$  dòng thì  $k$  ký tự cuối của các câu đó phải giống nhau, bao gồm cả dấu cách, giống nhau cả cách viết hoa hay thường.

Suốt đêm Misa đã sáng tác được  $n$  dòng. Đến giờ đi học Misa để lại bài thơ trên bàn, đề nghị Dima – bạn cùng phòng trong ký túc xá, học ca chiêu sáng tác tiếp. Misa ghi lại quy tắc gieo vần nhưng quên ghi giá trị  $k$ !

Dima cũng muốn giúp Misa, nhưng việc đầu tiên là phải xác định giá trị  $k$  và để cho an toàn, Dima dựa vào phần đã có tìm  $k$  lớn nhất thỏa mãn quy tắc gieo vần.

Hãy xác định giá trị  $k$  mà Dima sử dụng khi kéo dài bài thơ.

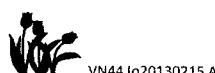
**Dữ liệu:** Vào từ file văn bản LYRICS.INP:

- ◆ Dòng đầu tiên chứa 2 số nguyên  $n$  và  $m$  ( $1 \leq m \leq n \leq 1\,000$ ),
- ◆ Mỗi dòng trong  $n$  dòng sau chứa một câu thơ dưới dạng xâu không quá 1000 ký tự bao gồm chữ cái la tinh (hoa hoặc thường) và dấu cách. Xâu không bắt đầu và không kết thúc bằng dấu cách.

**Kết quả:** Đưa ra file văn bản LYRICS.OUT một số nguyên – giá trị  $k$  tìm được.

**Ví dụ:**

LYRICS.INP	LYRICS.OUT
2 1 Russia Russia go go go Russia Russia lets go	3



```

#include <fstream>
#include<algorithm>
#include<ctime>
#include<string>
using namespace std;

#define name "lyrics"
int n,m,k,r,im,l,lm;
string s[10001],t1,t2;
ifstream fi(name".inp");
ofstream fo(name".out");
int main()
{clock_t aa=clock();
fi>>n>>m; getline(fi,s[0]);
for(int i=1;i<=n;++i)getline(fi,s[i]);
r=2000;
for(int i=1;i<=m;++i)
{l=s[i].size(); if(r>l)r=l;
im=i+m;
while(im<=n)
{lm=s[im].size(); if(r>lm)r=lm;
for(int j=0;j<r;++j)
if(s[i][l-j-1]!=s[im][lm-j-1]) {r=j;break;}
im+=m;
}
fo<<r;
clock_t bb=clock();
fo<<"\nTime: "<<(double)(bb-aa)/1000<<" sec.";
}
}

```

## VL06. ROBOT

Tên chương trình: ROBOT.???

Sinh viên trường Đại học Kỹ thuật hàng không đã thiết kế một robot hỗ trợ quá trình lắp ráp động cơ máy bay.

Quá trình lắp ráp động cơ có 26 loại thao tác khác nhau ký hiệu bằng các chữ cái la tinh thường, các thao tác giống nhau được ghi nhận bằng cùng một ký tự.

Robot được sử dụng một lần để hỗ trợ một phần quá trình lắp ráp gồm dãy các thao tác liên tiếp nhau trong toàn quá trình.

Bộ nhớ robot có  $k$  ô, mỗi ô chứa một lệnh. Các lệnh được thực hiện liên tiếp nhau theo trình tự ghi và sau khi thực hiện lệnh thứ  $k$  robot tiếp tục thực hiện lại chương trình từ đầu. Người ta có thể dừng robot ở bất cứ lúc nào. Việc sử dụng robot được coi là có giá trị kinh tế nếu nó được dùng để thực hiện không ít hơn  $k+1$  thao tác.

Ví dụ, quá trình lắp ráp được mô tả bằng xâu “**zabacabab**” và  $k = 2$ , ta có 5 cách sử dụng robot hiệu quả: nạp vào bộ nhớ robot “**ab**” và thực hiện các công việc từ 2 đến 4 (“**aba**”), nạp vào bộ nhớ robot “**ac**” và thực hiện các công việc từ 4 đến 6 (“**aca**”), nạp vào bộ nhớ robot “**ab**” và thực hiện các công việc từ 6 đến 8 (“**aba**”), nạp vào bộ nhớ robot “**ab**” và thực hiện các công việc từ 6 đến 9 (“**abab**”), nạp vào bộ nhớ robot “**ba**” và thực hiện các công việc từ 7 đến 9 (“**bab**”).

Cho xâu độ dài  $n$  mô tả quá trình lắp ráp ( $1 < n \leq 10^5$ ). Hãy xác định số cách sử dụng robot hiệu quả.

**Dữ liệu:** Vào từ file văn bản ROBOT.INP:

- ◆ Dòng đầu tiên chứa số nguyên  $k$  ( $0 < k < n$ ),
- ◆ Dòng thứ 2 chứa xâu mô tả quá trình lắp ráp.

**Kết quả:** Đưa ra file văn bản ROBOT.OUT một số nguyên – số cách sử dụng robot hiệu quả.

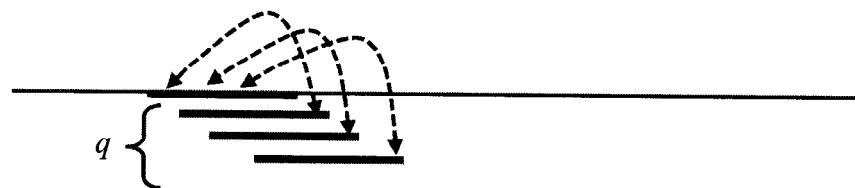
**Ví dụ:**

ROBOT.INP
2
<b>zabacabab</b>

ROBOT.OUT
5



## *Giải thuật*



$$res += q * (q + 1) / 2$$

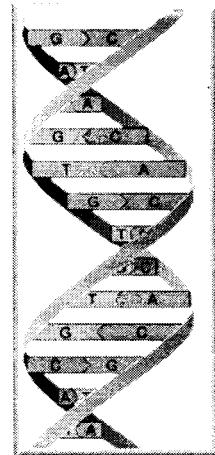
```
#include <fstream>
#include <string>
using namespace std;
string s;
int k,n;
long long res,q;
ifstream fi ("Robot.inp");
ofstream fo ("Robot.out");

int main()
{fi>>k;
 fi>>s; s+="Z";
 n=s.size();
 res=0;q=0;
 for(int i=k;i<n;++i)
 {
 if(s[i]==s[i-k])q++; else{res+=(q*(q+1)/2);q=0; }
 }
 fo<<res;
}
```

## VL19. TIỀN HÓA

Tên chương trình: EVOLUTION.???

Người ta phát hiện thấy sinh vật trên một thiền thạch rơi xuống trái đất. Lập tức thiền thạch được đưa vào phòng thí nghiệm cách ly để nghiên cứu xem sự phát triển của sinh vật có ảnh hưởng đến cuộc sống trên trái đất hay không. Đây là một sinh vật hết sức đơn giản. DNA của nó được mô tả chỉ với một trong số 4 ký tự {**A, C, T, G**}. Tuy nhiên quá trình phát triển thì hết sức độc đáo: sau mỗi khoảng thời gian DNA được kéo dài thêm một ký tự ở bên phải hoặc bên trái chuỗi đã có. Người ta cũng xác định được ký tự mới nào sẽ xuất hiện trong chuỗi DNA ở giai đoạn phát triển tiếp theo từ lúc bắt đầu cho đến khi ngừng phát triển, tuy nhiên việc ký tự mới được gắn vào bên phải hay trái của chuỗi đã có là ngẫu nhiên. Kết quả nghiên cứu cũng cho thấy sinh vật với chuỗi DNA có thứ tự từ điển càng nhỏ bao nhiêu thì sức sống càng mạnh bấy nhiêu.



Hãy xác định DNA của cá thể có sức sống mạnh nhất.

**Dữ liệu:** Vào từ file văn bản EVOLUTION.INP:

- ◆ Dòng đầu tiên chứa một ký tự trong tập {**A, C, T, G**} – chuỗi DNA ban đầu của sinh vật,
- ◆ Dòng thứ 2 chứa xâu **E** gồm các ký tự từ tập trên, trình tự các ký tự là trình tự xuất hiện ký tự mới trong DNA. Xâu **E** có độ dài không quá  $10^6$ .

**Kết quả:** Đưa ra file văn bản EVOLUTION.OUT xâu xác định DNA của cá thể có sức sống mạnh nhất.

**Ví dụ:**

EVOLUTION.INP
C
ACTG

EVOLUTION.OUT
ACCTG



```
#include <fstream>
#include <string>
using namespace std;
char cc, r[2000001];
int p,q,l;
string s;
ifstream fi ("Evolution.inp");
ofstream fo ("Evolution.out");
int main()
{fi>>cc>>s;
p=1000000;q=p; r[p]=cc; l=s.size();
for(int i=0;i<l;++i) if(r[p]>=s[i]) r[--p]=s[i]; else r[++q]=s[i];
for(int i=p;i<=q;++i) fo<<r[i];
}
```

## VN37. DANH SÁCH TRƯỜNG HỌC

Tên chương trình: SCHOOLS.???

Ở vòng thi loại Olympic Tin học mọi học sinh đều có quyền đăng ký tham gia. Mục tên trường là một trong những thông tin bắt buộc phải điền trong phiếu đăng ký. Tên trường bao gồm tên thường gọi và số của trường. Số của trường là số nguyên không bắt đầu bằng 0. Không có hai trường nào có số giống nhau. Tên thường gọi của một trường có thể khác nhau tùy cách gọi của người dự thi, nhưng không chứa số. Người đăng ký có thể nêu tên trường theo cách tùy ý, nhưng bắt buộc phải ghi số của trường, ví dụ “**Liceum 9**”, “**School N 9**” hay “**9ya shkola**”.

Có **n** thí sinh đăng ký dự thi. Hãy xác định số lượng trường có số người đăng ký không quá 5 và chỉ ra số của các trường đó.

**Dữ liệu:** Vào từ file văn bản SCHOOLS.INP:

- ◆ Dòng đầu tiên chứa số nguyên **n** ( $1 \leq n \leq 1\,000$ ),
- ◆ Mỗi dòng trong **n** dòng sau chứa xâu độ dài không quá 100 xác định tên trường, xâu chỉ chứa các ký tự la tinh (thường và hoa), ký tự số và dấu cách.

**Kết quả:** Đưa ra file văn bản SCHOOLS.OUT:

- ◆ Dòng đầu tiên chứa số nguyên **m** – số lượng trường có số người đăng ký không quá 5,
- ◆ Mỗi dòng trong **m** dòng sau chứa một số nguyên – số của các trường có số người đăng ký không quá 5.

**Ví dụ:**

SCHOOLS.INP
<b>9</b>
<b>Physics and Mathematics School 18</b>
<b>9ya shkola imeni Pushkina</b>
<b>Lyceum 9</b>
<b>PaMS 18</b>
<b>Gymnasium 42</b>
<b>School 9</b>
<b>Shkola nomer 9</b>
<b>High school 9</b>
<b>School N 9</b>

SCHOOLS.OUT
<b>2</b>
<b>18</b>
<b>42</b>



VN37 StPt Reg\_Olym 20140203 B

```

#include <fstream>
#include <string>
#include <algorithm>
using namespace std;
ifstream fi ("SCHOOLS.INP");
ofstream fo ("SCHOOLS.OUT");
string s,sc,ns="0123456789",a[1001],b[1001];
int n,i1,i2,m;

int main()
{fi>>n; getline(fi,s);
for(int i=0;i<n;++i)
{getline(fi,s);
i1=s.find_first_of(ns);
i2=s.find_last_of(ns);
//a[i]=s.substr(i1,i2-i1+1);
a[i].assign(s,i1,i2-i1+1);
}
sort(a,a+n); a[n]="****";
i1=1;s=a[0];m=0;
for(int i=0;i<n;++i)
if(a[i]==a[i+1])++i1;
else{if(i1<5){b[m++]=a[i];}i1=1;}
fo<<m<<'\n';
for(int i=0;i<m;++i) fo<<b[i]<<'\n';
}

```

*Chú ý chuyển sang dòng mới trước khi nhập xâu!*

*Hai cách lấy dòng con*

## 39M. BẢN ĐỒ

Tên chương trình: MAP.???

Thuyền trưởng Aleksander Smollet là nỗi ám ảnh kinh hoàng của cướp biển vào những năm của giữa thế kỷ 18. Năm 1744, sau một trận hải chiến ông thu được tấm bản đồ cổ ghi nơi chôn dấu châu báu của một đám cướp. Đáng tiếc việc giải mã thông tin ghi trên bản đồ không phải là chuyện đơn giản. Ông chỉ hiểu lờ mờ là kho báu được chôn cách nơi dựng cây thập tự cỡ **x** bước về phía đông. Cụ thể **x** là bao nhiêu thì không thể tính được. Khi trở về đại lục ông mang bản đồ đến nhờ người bạn già, một nhà thông thái trong Hoàng gia nhờ giúp đỡ. Sau khi tìm hiểu một thời gian mham thông thái cho biết dòng thông báo trên bản đồ là một cách mà cướp biển thời xưa hay dùng để mã hóa một số nguyên. Để tìm được số này cần loại bỏ tất cả các dấu cách, sau đó xác định số cách xóa các ký tự để 3 ký tự còn lại tạo thành một xâu mà đọc từ trái sang phải cũng giống như đọc từ phải sang trái. Số cách xóa chính là số được mã hóa.

Aleksander Smollet đoán đó chính là số **x** cần tìm. Tuy vậy ông lại không thể tính được nó.

Thời gian trôi đi. Sau khi thuyền trưởng qua đời, tất cả bị chìm vào quên lãng.

Mãi tới gần đây, khi thực hiện việc phục chế các văn bản cũ trong kho lưu trữ và tạo phiên bản số hóa để bảo tồn lâu dài người ta lại tình cờ tìm thấy tấm bản đồ này cùng các ghi chép của Aleksander Smollet. Với các công cụ tính toán hiện đại việc xác định số được mã hóa theo cách này không còn là vấn đề lớn.

**Yêu cầu:** Cho xâu **s** ghi trên bản đồ, chỉ chứa các ký tự la tinh thường và dấu cách. Hãy xác định số được mã hóa.

**Dữ liệu:** Vào từ file văn bản MAP.INP gồm một dòng chứa xâu **s** khác rỗng độ dài không quá  $3 \times 10^5$ .

**Kết quả:** Đưa ra file văn bản MAP.OUT số nguyên tìm được.

**Ví dụ:**

MAP.INP	MAP.OUT
<b>treasure</b>	<b>8</b>
<b>you will never find the treasure</b>	<b>146</b>



## Giải thuật

Chuẩn hóa xâu **s** (loại bỏ các dấu cách),

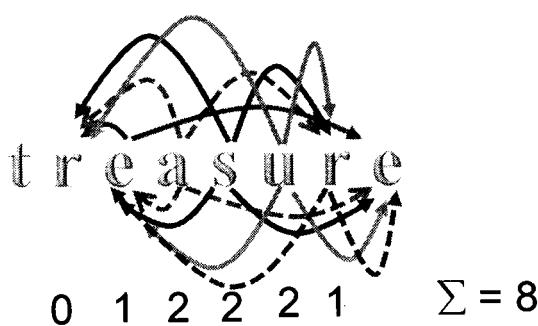
Gọi **n** là độ dài xâu **s** đã chuẩn hóa.

Ký tự  $s_i$  có thể là tâm (ký tự giữa) của xâu palindrome độ dài 3 nếu tồn tại  $j$  và  $k$  thỏa mãn các điều kiện:

- ◆  $j < i$ ,
- ◆  $k > i$ ,
- ◆  $s_j = s_k$ .

Tạo bảng  $p[n][26]$  tích lũy lần số xuất hiện các ký tự kế từ đầu xâu,  $p_{ij}$  – số lần xuất hiện ký tự thứ  $j$  kwwr từ đầu xâu tới vị trí thứ  $i$ .

Bảng này cho phép ta dễ dàng tính được, nếu lấy ký tự  $s_i$  làm tâm ta có thể nhận được bao nhiêu xâu palindrome độ dài 3, từ đó – dễ dàng



nhận được kết quả cần tìm.

	t	r	e	a	s	u	r	e
a	0	0	0	1	1	1	1	1
b	1	0	0	0	0	0	0	0
c	2	0	0	0	0	0	0	0
d	3	0	0	0	0	0	0	0
e	4	0	0	1	1	1	1	2
f	5	0	0	0	0	0	0	0
g	6	0	0	0	0	0	0	0
h	7	0	0	0	0	0	0	0
i	8	0	0	0	0	0	0	0
j	9	0	0	0	0	0	0	0
k	10	0	0	0	0	0	0	0
l	11	0	0	0	0	0	0	0
m	12	0	0	0	0	0	0	0
n	13	0	0	0	0	0	0	0
o	14	0	0	0	0	0	0	0
p	15	0	0	0	0	0	0	0
q	16	0	1	1	1	1	2	2
r	17	0	0	0	0	1	1	1
s	18	0	0	0	1	1	1	1
t	19	1	1	1	1	1	1	1
u	20	0	0	0	0	1	1	1
v	21	0	0	0	0	0	0	0
w	22	0	0	0	0	0	0	0
x	23	0	0	0	0	0	0	0
y	24	0	0	0	0	0	0	0
z	25	0	0	0	0	0	0	0

Bảng P

```

#include <iostream>
#include <vector>
#include <cstdio>
#include <fstream>

using namespace std;

const int MAXN = 3e5 + 7;
int64_t pref_sum[MAXN][26];

int main()
{
    ifstream cin("map.inp");
    ofstream cout("map.out");
    string s, t;
    while (cin >> t) s += t;
    int n = s.length();
    int64_t ans = 0;
    for (int i = 1; i <= n; i++)
    {
        for (int j = 0; j < 26; j++)
            pref_sum[i][j] = pref_sum[i - 1][j];
        pref_sum[i][s[i - 1] - 'a']++;
    }
    for (int i = 1; i < n - 1; i++)
        for (int j = 0; j < 26; j++)
            ans += pref_sum[i][j] * (pref_sum[n][j] - pref_sum[i+1][j]);
    cout << ans;
}

```

## 6. Cấu trúc dữ liệu hàng đợi/stack

### 30M. MATRIOSHKA

Tên chương trình: MATRIOSHKAS.???

Matrioshka là búp bê tiện bằng gỗ, được đặt lồng vào nhau. Kho chứa bán thành phẩm của nhà máy đã chật cứng vì phân xưởng sơn trang trí bị sự cố kỹ thuật. Giám đốc nhà máy quyết định băng mọi giá phải giải phóng kho. Nhưng những con búp bê mộc này thì có thể bán được cho ai?

Cái khó làm ló cái khôn. Cuối cùng Giám đốc nhà máy cũng đã tìm được lối thoát. Nhờ mối quan hệ tốt với Hội Mỹ

thuật thành phố ông cũng  
đã ký được một hợp đồng  
sơn vẽ búp bê trên tinh  
thần tương trợ giúp đỡ.  
Để tránh các rắc rối về thủ  
tục giấy tờ và tài chính có  
thể này sau này các búp



bé được đóng lồng nhau như khi mang bán, mỗi con matrioshka có thể chứa  $m$  búp bê gỗ đường kính  $a_1, a_2, \dots, a_m$ , trong đó  $a_{i+1} = a_i + 1$ ,  $i = 1 \div m-1$ ,  $m$  có thể khác nhau với các matrioshka khác nhau. Mỗi matrioshka nhà máy phải trả một khoản tiền như nhau không phụ thuộc vào  $m$ !

Hiện tại trong kho có  $n$  búp bê gỗ, con thứ  $i$  có kích thước  $s_i$ ,  $i = 1 \div n$ . Hãy xác định số lượng  $t$  nhỏ nhất các matrioshka có thể lấp ráp để chi phí sơn phải trả là tối thiểu. Với mỗi matrioshka thứ  $j$  hãy cho biết  $a_j$  và  $b_j$  – kích thước búp bê nhỏ nhất và lớn nhất tạo thành matrioshka.

**Dữ liệu:** Vào từ file văn bản MATRIOSHKAS.INP:

- ◆ Dòng đầu tiên chứa số nguyên  $n$  ( $1 \leq n \leq 2 \times 10^5$ ),
- ◆ Dòng thứ 2 chứa  $n$  số nguyên  $s_1, s_2, \dots, s_n$  ( $1 \leq s_i \leq 2 \times 10^5$ ,  $i = 1 \div n$ ).

**Kết quả:** Đưa ra file văn bản MATRIOSHKAS.OUT:

- ◆ Dòng đầu tiên chứa số nguyên  $t$ ,
- ◆ Mỗi dòng trong  $t$  dòng sau chứa 2 số nguyên  $a$  và  $b$  – kích thước nhỏ nhất và lớn nhất của búp bê trong matrioshka.

**Ví dụ:**

MATRIOSHKAS.INP
5
3 2 1 2 5

MATRIOSHKAS.OUT
3
2 2
1 3
5 5

## *Giai thuật*

Mảng **int b[200001];**

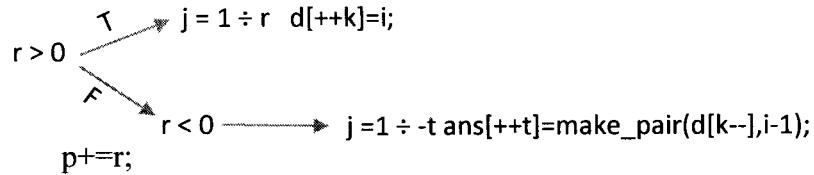
$b_i$  – số búp bê kích thước  $i$ ,  $++b[s[i]]$ ;  $i = 1 \div n$ ,  $nx = \max\{b_i\}$

Biến int  $r=0, k=0, p=0$ ;  $p$  – số búp bê đang có thể lắp ráp thêm.

$d$  – stack  $\langle \text{int} \rangle$ ,

$\text{ans}$  – stack  $\langle \text{pair} \langle \text{int}, \text{int} \rangle \rangle$

Với  $i = 1 \div nx+1$ :  $r = b_i - p$ ,



Đưa ra  $r$  và  $\text{ans}[i]$ ,  $i = 1 \div r$ .

Độ phức tạp của giải thuật:  $O(n)$ .

```

#include <iostream>
#include <ctime>
using namespace std;
int d[200001], b[200002]={0}, n, nx=0, t=0, k=0, p=0, r;
pair<int, int> ans[200001];
ifstream fi("matrioshka.inp");
ofstream fo("matrioshka.out");

int main()
{clock_t aa=clock();
 fi>>n;
 for(int i=1;i<=n;++i)
 {fi>>r; if(r>nx)nx=r; ++b[r];}
 for(int i=1;i<=nx+1;++i)
 {r=b[i]-p;
  if(r>0) for(int j=1;j<=r;++j)d[++k]=i;
  else
   if(r<0) for(int j=1;j<=-r;++j)ans[++t]=make_pair(d[k--], i-1);
  p+=r;
 }
 fo<<t<<'\n';
 for(int i=1;i<=t;++i) fo<<ans[i].first<<' ' <<ans[i].second<<'\n';
 clock_t bb=clock();
 fo<<"Time: "<<(double)(bb-aa)/1000<<" sec";
}
  
```

## VO26. TRÒ CHƠI

Tên chương trình: GAMEVO26.???

Có  $2n-1$  quân bài, trong đó có  $n-1$  lá bài trắng để ở ngoài,  $n$  lá bài còn lại được bày trên bàn. Ở mỗi lá bài này mặt trên có ghi một số nguyên trong có giá trị tuyệt đối không vượt quá  $10^9$ , mặt dưới ghi số 0. Người chơi đoán một số trong phạm vi từ 0 đến  $n$  và thực hiện các nước đi. Mỗi nước đi bao gồm các hành động:

- ◆ Chọn lá bài trên bàn có số nhỏ nhất ở mặt trên, nếu có nhiều lá bài cùng có số ở mặt trên là nhỏ nhất thì chọn lá bài có số ở mặt dưới nhỏ nhất, rút lá bài đó ra khỏi bàn; giả thiết lá bài được chọn có số ở mặt trên là  $a$  và số ở mặt dưới là  $b$ ,
- ◆ Chọn tiếp một lá bài trên bàn có số nhỏ nhất ở mặt trên, nếu có nhiều lá bài cùng có số ở mặt trên là nhỏ nhất thì chọn lá bài có số ở mặt dưới nhỏ nhất, rút lá bài đó ra khỏi bàn; giả thiết lá bài được chọn có số ở mặt trên là  $c$  và số ở mặt dưới là  $d$ ,
- ◆ Lấy một lá bài trắng ghi vào mặt trên số  $a+c$ , mặt dưới – số lớn nhất trong 2 số  $b+1$  và  $d+1$ , đặt lá bài đó lên bàn.

Trò chơi kết thúc sau  $n-1$  nước đi. Khi đó trên bàn chỉ còn một lá bài. Nếu số ghi ở mặt dưới của lá bài này trùng với số đoán ban đầu là thắng.

Hãy xác định số cần đoán để thắng.

**Dữ liệu:** Vào từ file văn bản GAMEVO26.INP:

- ◆ Dòng đầu tiên chứa số nguyên  $n$  ( $1 \leq n \leq 10^5$ ),
- ◆ Dòng thứ 2 chứa  $n$  số nguyên ghi ở mặt trên của các lá bài.

**Kết quả:** Đưa ra file văn bản GAMEVO26.OUT số cần đoán để thắng.

**Ví dụ:**

GAMEVO26.INP
5
1 2 3 4 5

GAMEVO26.OUT
3



### *Giải thuật:*

Sử dụng hàng đợi ưu tiên (nhỏ → lớn) lưu các quân bài.

Độ phức tạp của giải thuật: O(nlogn).

```
#include <fstream>
#include <vector>
#include <queue>
using namespace std;
typedef pair<int64_t, int> pi2;
int n,a,b,c,d,t;
pi2 p1,p2;
priority_queue<pi2,vector<pi2>,greater<pi2>> q;
ifstream fi ("Gamevo26.inp");
ofstream fo ("Gamevo26.out");

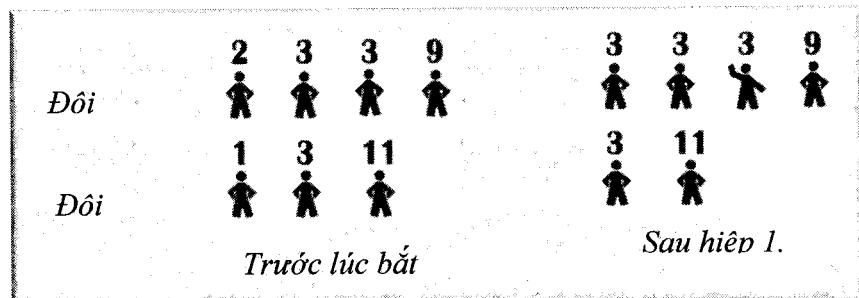
int main()
{fi>>n;
 for(int i=1; i<=n;++i){fi>>p1.first; p1.second=0; q.push(p1);}
 for(int i=1;i<n;++)
 {p1=q.top();q.pop(); p2=q.top();q.pop();
  t=p1.second; if(t<p2.second) t=p2.second;
  p1.first+=p2.first;p1.second=t; q.push(p1);
 }
 fo<<p1.second;
}
```

Dân xứ Byteland và Bitoland rất say mê sưu tầm cốc sứ. Số lượng cốc trong bộ sưu tập quyết định uy tín của người đó trong mắt dân chúng. Ai có càng nhiều thì uy tín càng cao.

Một cuộc thi sưu tập được tổ chức ở Byteland. Đội của Byteland có  $n$  người, còn đội của Bitoland – có  $m$  người. Người chơi ở mỗi đội xếp thành hàng ngang theo thứ tự mức uy tín tăng dần. Hàng thứ nhất của đội Byteland, hàng thứ 2 – của Bitoland. Người thứ  $i$  trong hàng của đội Byteland có mức uy tín là  $a_i$  ( $i = 1 \div n$ ), người thứ  $j$  trong hàng của đội Bitoland có mức uy tín là  $b_j$  ( $j = 1 \div m$ ).

Cuộc thi diễn ra theo từng hiệp chừng nào ở mỗi hàng còn ít nhất một người. Ở mỗi hiệp người đứng đầu ở mỗi hàng (tức là người có mức uy tín thấp nhất trong hàng) ra th đấu. Ai có mức uy tín cao hơn sẽ thắng. Người thua ra khỏi hàng và giao bộ sưu tập của mình cho người thắng. Như vậy uy tín người thắng sẽ tăng nhờ uy tín của người thua. Người thắng quay về hàng của mình và đứng vào vị trí đảm bảo độ uy tín trong hàng vẫn không giảm. Nếu có nhiều vị trí thỏa mãn thì người đó sẽ đứng ở nơi xa đầu nhất có thể. Nếu 2 người ra thi đấu có cùng mức ưu tiên thì sẽ tiến hành thi đấu hiệp phụ. Các hiệp phụ xuất hiện trong quá trình thi đấu được đánh số từ 1 trở đi kể từ đầu cuộc thi (không phụ thuộc vào các hiệp chính). Nếu hiệp phụ là chẵn thì người của đội Bitoland thắng, còn lẻ – người của đội Byteland thắng.

Hệ số hấp dẫn của một hiệp bằng vị trí của người thắng sau khi trở về hàng của mình. Độ hấp dẫn của cuộc thi bằng tổng độ hấp dẫn của các hiệp.



Hãy xác định độ hấp dẫn của cuộc thi.

**Dữ liệu:** Vào từ file văn bản COLLECTION.INP:

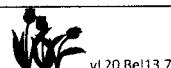
- ★ Dòng đầu tiên chứa 2 số nguyên  $n$  và  $m$  ( $1 \leq n, m \leq 250\,000$ ),
- ★ Dòng thứ 2 chứa  $n$  số nguyên  $a_1, a_2, \dots, a_n$  ( $1 \leq a_i \leq 10^9$ ,  $i = 1 \div n$ ),
- ★ Dòng thứ 3 chứa  $m$  số nguyên  $b_1, b_2, \dots, b_m$  ( $1 \leq b_j \leq 10^9$ ,  $j = 1 \div m$ ).

**Kết quả:** Đưa ra file văn bản COLLECTION.OUT một số nguyên – độ hấp dẫn của cuộc thi.

**Ví dụ:**

COLLECTION.INP
3 2
1 2 5
2 4

COLLECTION.OUT
5



```

#include <fstream>
using namespace std;
int n,m,pa,qa,pb,qb,ia,ib,rd;
int64_t a[250002],b[250002],aw[250002],bw[250002],r,x,y,m15;

ifstream fi ("Collection.inp");
ofstream fo ("Collection.out");
void getxy()
{long long u,v;
 if(ia<=n)u=a[ia];else u=m15;
 if(pa<=qa)v=aw[pa]; else v=m15;
 if(u>v){x=v;++pa;} else {x=u;++ia;}

 if(ib<=m)u=b[ib];else u=m15;
 if(pb<=qb)v=bw[pb]; else v=m15;
 if(u>v){y=v;++pb;} else {y=u;++ib;}}
void insert_ab(int64_t xx[],int64_t yy[],int nn,int ix,int px, int qx)
{int jl,jr,jm,j0;
int64_t t;
t=x+y;j0=0;
yy[++qa]=t;
if(xa<=nn)
{jl=ix;jr=nn;
while(jl<=jr)
{jm=(jr+jl)/2;
 if(t>=xx[jm])jl=jm+1; else jr=jm-1;
}
j0=jl-ix;
}
j0+=(qx-px+1); r+=j0;
}

void insert_a()
{int jl,jr,jm,j0;
long long t;
t=x+y;j0=0;
aw[++qa]=t;
if(ia<=n)
{jl=ia;jr=n;
while(jl<=jr)
{jm=(jr+jl)/2;
 if(t>=a[jm])jl=jm+1; else jr=jm-1;
}
j0=jl-ia;
}
j0+=(qa-pa+1); r+=j0;
}

void insert_b()
{int jl,jr,jm,j0;
long long t;
t=x+y;j0=0;
bw[++qb]=t;
if(ib<=m)
{jl=ib;jr=m;
}

```

```

while(jl<=jr)
  {jm=(jr+jl)/2;
   if(t>=b[jm])jl=jm+1; else jr=jm-1;
  }
  j0=jl-ib;
}
j0+=(qb-pb+1); r+=j0;
}

int main()
{fi>>n>>m;
for(int i=1;i<=n;++i)fi>>a[i];
for(int i=1;i<=m;++i)fi>>b[i];
rd=0; pa=1;pb=1;qa=0;qb=0;ia=1;ib=1;m15=1e15;
getxy();
do
{if(x>y || (x==y && rd==0)){insert_a(); if(x==y)rd^=1;}
 else {insert_b(); if(x==y)rd^=1;}
 getxy();
}while ((x<m15) && (y<m15));
fo<<r;
}

```

## VL18. LỊCH MỚI

Tên chương trình: CALENDAR.???

Các nhà bác học xứ Byteland tranh luận với nhau rất nhiều về cách xây dựng lịch phù hợp với hành tinh của mình, trong đó quan trọng nhất là xác định năm nhuận.

Ở đại hội lần thứ XII có một phương pháp xác định năm nhuận hoàn toàn mới được mọi người hết sức chú ý. Theo cách tính này, số năm được xét dưới dạng nhị phân (không có các số 0 không có nghĩa ở đầu). Các số giống nhau đứng liên tiếp tạo thành một nhóm. Nếu số chứa đúng 3 nhóm thì đó là năm nhuận. Ví dụ các năm  $9 = 1001_2$  và  $13 = 1101_2$  là năm nhuận, còn  $7 = 111_2$  – không phải là nhuận. Năm được đánh số từ 1 trở đi một cách liên tiếp.

Người ta muốn kiểm nghiệm độ chính xác của lịch này và tính số lượng năm nhuận trong khoảng thời gian từ **a** đến **b** (kể cả **a** và **b**).

Hãy xác định số năm nhuận trong khoảng đã cho.

**Dữ liệu:** Vào từ file văn bản CALENDAR.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 CALENDAR.OUT một số nguyên – số năm nhuận.

Ví dụ:

CALENDAR.INP
19 30

CALENDAR.OUT
5



```

#include <fstream>
#include <ctime>
using namespace std;
int64_t a,b,tg,t1,t2,t3,n,r,r1,r2,b1=1;
int
p[65]={0,0,0,1,3,6,10,15,21,28,36,45,55,66,78,91,105,120,136,153,171,
190,210,231, 253, 276, 300, 325, 351, 378,406,4 35, 465, 496, 528,
561, 595, 630,666,703,741,780,820,861,903, 946, 990, 1035, 1081,
1128,1176, 1225,1275,1326,1378,1431,1485,1540,1596,1653,1711,1770,
1830, 1891, 1953},

q[65]={0,0,0,1,4,10,20,35,56,84,120,165,220,286,364,455,560,680,816,9
69,1140,1330, 1540, 1771, 2024, 2300, 2600, 2925, 3276, 3654, 4060,
4495,4960,5456,5984,6545,7140,7770,8436,9139, 9880, 10660, 11480,
12341,13244,14190,15180,16215,17296,18424,19600,20825,22100,23426,
24804, 26235,27720,29260,30856,32509,34220,35990,37820,39711,41664};

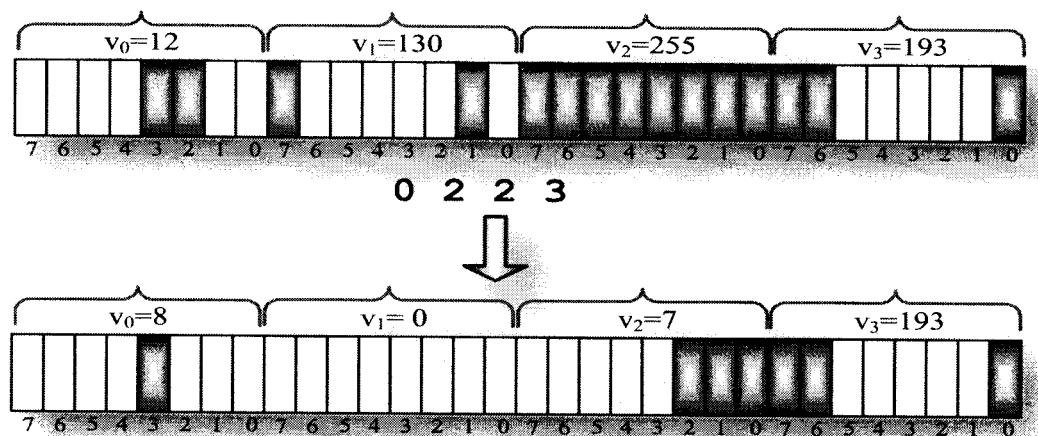
ifstream fi ("calendar.inp");
ofstream fo ("calendar.out");
int f(int64_t x)
{if(x<=4) return 0;
 if((x&1)==0)--x;t1=0;t2=0;tg=0;t3=0;
 for(int i=63;i>=0;--i) if((x&(b1<<i))!=0){n=i;break;}
 for(int i=n;i>=0;--i) if(((x>>i)&1)!=1){t1=i+1;break;}
 for(int i=t1-1;i>=1;--i) if(((x>>i)&1)!=0 && (((x>>(i+1))&1)==0
 )){t2=i;break;}
 for(int i=0;i<=n;++i)if(((x>>i)&1)==1 && (((x>>(i+1))&1)==0
 )){t3=i;break;}
 if(t1==0) return(q[n+1]);
 else
 {tg=t3+q[n];
 if(t2!=t3)tg+=(t2-t3);else tg++;
 for(int i=t1;i<n;++i)tg+=i;
 }
 return tg;
}
int main()
{//clock_t aa=clock();
 fi>>a>>b;
 r2=f(b);
 r1=f(a-1);
 fo<<r2-r1<<endl;
 //clock_t bb=clock();
 //fo<<"Time: "<<(double)(bb-aa)/1000<<" sec";
}

```

## VP11. NÉM ĐÁ

Tên chương trình: STONE.???

Các hệ thống lập trình đều cung cấp phương tiện để khởi tạo giá trị cho một mảng bộ nhớ tinh theo đơn vị byte. Tuy vậy byte là đơn vị quá lớn trong việc xử lý ảnh. Các chương trình xử lý ảnh đòi hỏi có công cụ khởi tạo giá trị cho một vùng bộ nhớ theo đơn vị tinh tế hơn là bít, xác lập giá trị 0 cho dãy bít *liên tiếp nhau từ trái sang phải*. Có câu ắt có cung. Một chương trình như vậy đã được xây dựng. Các byte trong vùng bộ nhớ cần khởi tạo được đánh số từ 0 trở đi, ngoài địa chỉ đầu của vùng cần khởi tạo lời gọi chương trình còn chứa 4 số nguyên  $c$ ,  $p$ ,  $d$  và  $q$  cho biết chương trình sẽ xác lập giá trị 0 cho các bít bắt đầu bít thứ  $p$  của byte  $c$  cho đến bít thứ  $q$  của byte  $d$  (kể cả bít này). Lưu ý rằng trong một byte các bít được đánh số từ 0 đến 7 từ phải sang trái. Một thành viên của VP11.Facebook phát tán trên mạng vài hình ảnh



không đẹp và bị các cư dân mạng “ném đá” tới tấp bằng cách hợp súc tạo lỗ hỏng thông tin trên ảnh bắt đầu từ một vùng thông tin có địa chỉ đã thống nhất, kích thước  $m$  bytes chứa các giá trị  $v_0, v_1, \dots, v_m$  ( $0 \leq v_j \leq 255$ ,  $j = 0 \div m-1$ ). Đã có  $n$  người tham gia tạo lỗ hỏng, người thứ  $i$  kích hoạt chương trình khởi tạo với các tham số  $c_i, p_i, d_i$  và  $q_i$  ( $0 \leq c_i < d_i < m$ , hoặc  $c_i = d_i < m$  và  $q_i < p_i$ ,  $i = 1 \div n$ ). Hoạt động này đã lôi cuốn thêm  $k$  bạn trẻ nữa tham gia, đưa ra các lệnh theo quy tắc trên, nhưng để tiết kiệm thời gian xử lý, một chương trình duyệt đã được cài đặt kiểm tra xem mỗi yêu cầu mới có cần phải thực hiện hay không và chỉ thực hiện khi nó có xóa thêm ít nhất một bít giá trị 1 nếu áp dụng với các giá trị  $v_j$  đã được xử lý bởi  $n$  người đầu tiên, khi đó người đưa ra yêu cầu sẽ nhận được câu trả lời **YES**, trong trường hợp ngược lại – câu trả lời sẽ là **PASS**.

Hãy xác định câu trả lời cho từng người trong số  $k$  người tham gia sau.

**Dữ liệu:** Vào từ file văn bản STONE.INP:

- ➔ Dòng đầu tiên chứa 3 số nguyên  $m, n$  và  $k$  ( $1 \leq m \leq 10^6, 1 \leq n \leq 10^5, 1 \leq k \leq 10$ ),
- ➔ Dòng thứ 2 chứa  $m$  số nguyên  $v_0, v_1, \dots, v_{m-1}$ , ( $0 \leq v_j \leq 255$ ),
- ➔ Dòng thứ  $i$  trong  $n$  dòng tiếp theo chứa 4 số nguyên  $c_i, p_i, d_i$  và  $q_i$ ,
- ➔ Mỗi dòng trong  $k$  dòng tiếp theo chứa 4 số nguyên xác định một yêu cầu xử lý mới theo quy cách như đã nêu.

Các số trên một dòng ghi cách nhau một dấu cách.

**Kết quả:** Đưa ra file văn bản STONE.OUT câu trả lời **YES** hoặc **PASS** cho mỗi yêu cầu mới, mỗi câu trả lời ghi trên một dòng.

*Ví dụ:*

STONE.INP	STONE.OUT
4 1 2 12 130 255 193 0 2 2 3 1 5 1 0 1 1 2 2	PASS YES

```
#include <fstream>
#include <ctime>
using namespace std;
ifstream fi ("Stone.inp");
ofstream fo ("Stone.out");
unsigned int v[1000000],f[8000000];
int n,n8,m,k,a[100],t1,t2,t3,t4,t,tl,tr,tf,fs[8000000];
pair<int,int>r[100000];

void get_fs()
{fs[0]=f[0];
 for(int i=1;i<n8;++i)fs[i]=fs[i-1]+f[i];
}

void upd_tf(int ii)
{int tt=-1;
 for(int i= ii;i>=0;--i)if(f[i]==1){tt=i;break;}
 if(tt== -1)tf=-1;else tf=tt;
 fo<<endl<<"Tf: "<<tf<<endl;
}

void upd_f(int ii)
{int tt;
 tl=r[ii].second; tr=r[ii].first;
 if(tf<=tr)tt= tf; else tt=tr;
 for(int i=tt;i>=tl;--i)f[i]=0;
 tf=tl-1;
}

void get_f()
{int jj=0;
 for(int i=0;i<m;++i)
   for(int j=7;j>=0;--j){t=(v[i]>>j)&1; f[jj++]=t;}
}

int main()
{
 //ios_base::sync_with_stdio(0);
 clock_t aa=clock();
```

```

fi>>m>>n>>k; n8=m*8;
for(int i=0;i<m;++i)fi>>v[i];
for(int i=0;i<n;++i)
{fi>>t1>>t2>>t3>>t4;
 r[i].first=t3*8+7-t4; r[i].second=t1*8+7-t2;
}
sort(r,r+n);tf=r[n-1].first;get_f();
for(int i=n-1;i>=0;--i)if(tf>=r[i].second) upd_f(i);

get_fs();
for(int i=1;i<=k;++i)
{fi>>t1>>t2>>t3>>t4;
 tr=t3*8+7-t4; tl=t1*8+7-t2-1;
 if((tl<0 && fs[tr]==0)|| (fs[tl]==fs[tr]))fo<<"PASS"; else
fo<<"YES";
 fo<<endl;
}
clock_t bb=clock();
fo<<"Time: "<<(double)(bb-aa)/1000<<" sec."<<endl;
}

```

## VJ19. TAM SAO THÁT BỘN

Tên chương trình: SQ.???

Có lẽ ai cũng biết chuyện ngụ ngôn một chị gà mái đang bới đất tìm giun cho đàn gà con bị gió thổi bay một sợi lông. Sự việc được kể từ tai này sang tai khác trở thành chuyện chị gà mái bị gió xoáy vặt trụi không còn chiếc lông nào! Các nhà xã hội học quyết định nghiên cứu một cách nghiêm túc sự biến đổi của các tin đồn. Người ta khảo sát nhiều người thuộc đủ các thành phần xã hội và ngành nghề khác nhau. Dựa vào các thông tin cá nhân người ta tính *Chỉ số lanh lợi SQ (Sagacious Quotient)* cho mỗi người được khảo sát và chốt lại danh sách  $n$  người có SQ là nguyên dương, khác nhau từng đôi một và không vượt quá  $n$ . Các người trong danh sách được đánh số từ 1 đến  $n$ .

Nội dung của công việc khảo sát là chọn một nhóm 4 người, cho người thứ nhất trong nhóm nghe một câu chuyện, sau đó người này phải kể lại cho người thứ 2 trong nhóm, người thứ 2 – kể lại cho người thứ 3 và người này kể lại cho người thứ tư. Các nhà nghiên cứu sẽ so sánh câu chuyện ban đầu với câu chuyện người thứ tư nghe được và rút ra các kết luận cần thiết. Để đề phòng sự phản đối có thể có của Hội bảo vệ quyền phụ nữ người ta quyết định chọn 2 loại nhóm – nhóm A và nhóm B theo các quy tắc sau:

■ Quy tắc chọn nhóm A:

- ◆ Nếu người thứ  $i$  của nhóm có thứ tự  $p_i$  trong danh sách thì  $p_1 < p_2 < p_3 < p_4$ ,
- ◆ Người thứ nhất và người thứ tư phải là nam giới, hai người kia là nữ,
- ◆ Chỉ số SQ của người thứ nhất phải lớn hơn chỉ số SQ của người thứ tư.

■ Quy tắc chọn nhóm B:

- ◆ Nếu người thứ  $i$  của nhóm có thứ tự  $p_i$  trong danh sách thì  $p_1 < p_2 < p_3 < p_4$ ,
- ◆ Người thứ nhất và người thứ tư phải là nữ, hai người kia là nam,
- ◆ Chỉ số SQ của người thứ nhất phải nhỏ hơn chỉ số SQ của người thứ tư.

**Yêu cầu:** Cho  $n$  và các số nguyên  $a_i$ ,  $i = 1 \div n$ , trong đó nếu  $a_i > 0$  thì người thứ  $i$  là nam và có SQ là  $a_i$ , nếu  $a_i < 0$  thì người thứ  $i$  là nữ và có SQ là  $-a_i$ . Hãy xác định có thể chọn được bao nhiêu nhóm khác nhau. Hai nhóm gọi là khác nhau nếu khác nhau người thứ nhất hoặc khác nhau người thứ tư hay khác nhau cả 2 người thứ nhất và thứ tư.

**Dữ liệu:** Vào từ file văn bản SQ.INP:

- ◆ Dòng đầu tiên chứa số nguyên  $n$  ( $4 \leq n \leq 10^6$ ),
- ◆ Dòng thứ 2 chứa  $n$  số nguyên  $a_1, a_2, \dots, a_n$ .

**Kết quả:** Đưa ra file văn bản SQ.OUT trên một dòng 2 số nguyên – số lượng nhóm A khác nhau có thể chọn và số lượng nhóm B khác nhau có thể chọn.

**Ví dụ:**

SQ.INP
8
-2 6 -4 7 8 -3 1 5

SQ.OUT
2 1

```

#include <fstream>
#include <ctime>
using namespace std;
ifstream fi ("sq.inp");
ofstream fo ("sq.out");
int a[1000001],s[1000001]={0},n,t,ts,i0,i1,i2,is,it,kt;
long long res;
long long tt;

void insert_t(int ii)
{while(ii<=n){s[ii]++;ii+=(ii&(-ii));}
}

int sum_t(int ii)
{int r=0;
 while(ii>0){r+=s[ii]; ii&=(ii-1);}
 return(r);
}

void xly2()
{for(int i=1;i<=n;++i)if(a[i]<0){is=i;break;}
 if(is>=n-2){kt=1;return;}
 i1=0;
 for(int i=is;i<=n;++i)if (a[i]<0)insert_t(-a[i]);else {i1=i;i0=i1;
 break;}
 if(i1==0 || i1>=n-1){kt=1;return;}
 i2=0;
 for(int i=i1+1;i<=n;++i)if(a[i]>0){i2=i;break;}
 if(i2==0 || i2==n){kt=1;return;}
}

void xly1()
{for(int i=1;i<=n;++i)if (a[i]>0){is=i; break;}
 if(is>=n-2){kt=1;return;}
 i1=0;
 for(int i=is;i<=n;++i)if(a[i]>0){insert_t(a[i]);ts++;}else
 {i1=i;i0=i1; break;}
 if(i1==0 || i1>=n-1){kt=1;return;}
 i2=0;
 for(int i=i1+1;i<=n;++i)if(a[i]<0){i2=i;break;}
 if(i2==0 || i2==n){kt=1;return;}
}

int main()
{//clock_t aa=clock();
 fi>>n; res=0; ts=0; kt=0;
 for(int i=1;i<=n;++i)fi>>a[i];
 tt=0; xly1();
 if(kt==0)
 { it=i2;tt=2;
 for(int i=it+1;i<=n;++i)
 if(a[i]<0){i1=i2;i2=i;tt++;}
 else {if (i0<i1) {for(int j=i0+1;j<i1;++j)
 if(a[j]>0){insert_t(a[j]);ts++;}
 i0=i1;
 }
 res+=(ts-sum_t(a[i]));
 }
}

```

```

}

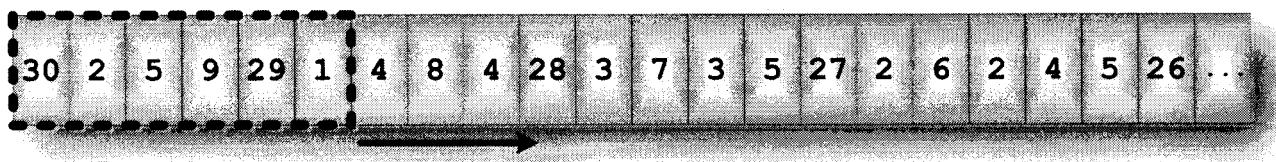
memset(s,0,sizeof(s)); kt=0; xly2();
if(kt==0)
{ it=i2;tt=2;
  if(a[i]>0){i1=i2;i2=i;}
  else
{if (i0<i1) {for(int j=i0+1;j<i1;++j) if(a[j]<0)insert_t(-a[j]);
             i0=i1;
             }
}
fo<<res<<endl;
// clock_t bb=clock();
// fo<<"Time: "<<(double)(bb-aa)/1000<<"sec";
}

```

## VJ34. CHỚP SÁNG

Tên chương trình: FLASH.???

Nhiệm trò chơi điện tử đã trở thành một căn bệnh đòi hỏi y học phải có pháp đồ điều trị riêng. Bệnh viện Thần kinh Trung ương đang thử nghiệm một phương pháp mới đầy triển vọng. Người ta định kỳ cho hiện lên ở trung tâm màn hình một cửa sổ chứa m cột sáng, mỗi cột sáng một màu trong phạm vi từ 1 đến 255 (chế độ 256 màu, trừ màu đen). Thời gian xuất hiện cửa sổ chỉ là  $\frac{1}{30}$  giây, hình ảnh không kịp lưu lại trong mắt, không ảnh hưởng đến công việc bình thường, nhưng não bộ vẫn kịp ghi nhận thông tin. Nếu chọn dãy màu hợp lý thì người chơi sẽ rất mau mệt mỏi hoặc xuất hiện cảm giác ức chế khó chịu, dần dần dẫn đến tâm lý chán hoặc sợ trò chơi điện tử. Các cột sáng hiển thị được chọn như sau. Xuất phát từ dãy  $n$  cột sáng ban đầu, cột sáng thứ  $i$  có màu  $c_i$  ( $0 < c_i \leq 255$ ,  $i = 1 \dots n$ ). Mỗi dãy cột sáng tiếp theo được xác định từ dãy cột sáng trước, trong đó giá trị màu được giảm đi 1, nếu giá trị màu nhận được là 0 thì cột đó bị loại bỏ và dồn các cột bên phải sang vị trí đó, cuối cùng thêm một cột mới có giá trị màu bằng số cột sáng ở dãy trước. Các dãy cột sáng này được viết liên tiếp nhau thành một dãy màu dài vô hạn. Ban đầu cửa sổ hiển thị các cột sáng từ 1 đến một m nào đó. Ở nhịp thời gian tiếp theo cửa sổ chuyển sang phải một vị trí. Ví dụ,  $n = 4$ ,  $m = 6$  và dãy màu ban đầu là



(30, 2, 5, 9). Dãy màu sẽ hiển thị trên cửa sổ sẽ là:

Các cán bộ nghiên cứu phải thử nghiệm một số dãy màu ban đầu để chọn dãy tối ưu, có hiệu quả điều trị lớn nhất. Sự tận tâm và nhiệt tình của nhóm nghiên cứu thật đáng kính trọng. Một khối lượng lớn các dãy ban đầu được khảo sát và đánh giá. Cuối cùng người ta cũng đã chọn được dãy ban đầu tối ưu. Dãy này được ghi lại để chuẩn bị cho báo cáo khoa học ở một hội nghị quốc tế. Đáng tiếc là dãy này quá tốt làm cho chính các bộ trong nhóm cũng mệt mỏi và nhầm lẫn. Người ta bỏ sót số cuối cùng khi ghi! Cũng may là trong biên bản khảo sát còn ghi lại tính chất dãy màu xuất hiện, từ vị trí  $p$  đến vị trí  $q$  trong dãy có đúng  $k$  cặp màu  $c_i$  và  $c_j$  thỏa mãn  $c_j < c_i$ ,  $p \leq i < j \leq q$  và tổng giá trị các màu trong dãy (kể cả  $c_n$ ) không vượt quá 255.

**Yêu cầu:** Cho các số nguyên  $n$ ,  $c_i$ ,  $i = 1 \dots n-1$ ,  $p$ ,  $q$  và  $k$ . Hãy xác định  $c_n$  nhỏ nhất thỏa mãn các điều kiện đã nêu. Nếu không tồn tại giá trị thích hợp thì đưa ra số -1.

**Dữ liệu:** Vào từ file văn bản FLASH.INP:

- ◆ Dòng đầu tiên chứa số nguyên  $t$  – số bộ test trong file ( $1 \leq t \leq 20$ ),
- ◆ Mỗi test gồm một nhóm 1 hoặc 2 dòng:
  - Dòng đầu tiên trong nhóm chứa số 4 nguyên  $n$ ,  $p$ ,  $q$  và  $k$  ( $1 \leq n \leq 255$ ,  $1 \leq p \leq q \leq 10^9$ ,  $0 \leq k \leq (q-p) \times (q-p+1)/2$ ).
  - Với  $n > 1$ : dòng thứ 2 chứa  $n-1$  số nguyên  $c_1, c_2, \dots, c_{n-1}$ ,  $1 \leq c_i \leq 255$ ,  $i = 1 \dots n-1$ .

Các số trên một dòng cách nhau một dấu cách.

**Kết quả:** Đưa ra file văn bản FLASH.OUT , kết quả mỗi test đưa ra trên một dòng chứa số nguyên  $c_n$  tìm được hoặc -1 nếu không tồn tại  $c_n$  thỏa mãn.

**Ví dụ:**

FLASH.INP	FLASH.OUT
2	3
5 10 18 17	4
4 8 5 1	
7 41 50 16	
8 5 1 4 2 6	

```
#include <fstream>
#include <ctime>
using namespace std;
ifstream fi ("flash.inp");
ofstream fo ("flash.out");
int tt,f1,p0,p3,s[256],a0[20000],a2[20000],am[20000],a1[20000];
int mmx=600;
int a[512],b[512],c[512];
long long k,kp,ss,x,y,yy,z,xz,yz,xy,mm,res,
      t,r,is,js,ls,l2,lse,lsb,n,m,ma,mx,p,q,pq;
int cc;
int comp_b()
{if(ma!=m) return(1); else
 {for(int i=1;i<=m;++i) if(a[i]!=b[i]) return(1);}
 return(0);
}

void get_new()
{ k=0;
  for(int i=1;i<=m;++i) if(b[i]>1)b[++k]=--b[i];
  b[++k]=m;m=k;
}

void determ_l()
{m=n;
 for(int i=1;i<=mmx;++i)get_new();
 for(int i=1;i<=m;++i)a[i]=b[i];
 js=mmx; ls=-1; ma=m;
 do
 {++js;
  get_new();
 }
 while(comp_b()==1);
 ls=js-mmx;
 //ls - size loop (par seq) ***
 for(int i=1;i<=n;++i)b[i]=c[i];
 m=n; lsb=m;
 for(int
i=1;i<=mmx;++i){get_new();if(comp_b() ==0){js=i+1;break;}lsb+=m;}
 } //End of Determ_l

void insert_t(int ii)
{while(ii<=255){s[ii]++;ii+=(ii&(-ii));}
```

```

}

int sum_t(int ii)
{int rr=0;
 while(ii>0){rr+=s[ii];ii&=(ii-1);}
 return(rr);
}

void get_a0()
{int pp,jp,ja;
 pp=p;ja=0;
 if(f1==1)
 {
   for(int i=1;i<=n;++i)b[i]=c[i]; m=n;
   while(pp>m){pp-=m;get_new();}
   jp=pp; //pp - new pos
   do
     {a0[++ja]=b[jp++];
      if(jp>m){get_new();jp=1;}
     }
   while(ja<q); // OK ***
 }
 if(f1==2){jp=p0+a0[0]-1; for(int i=p0;i<=jp;++i)a0[++ja]=am[i];}

if(f1==3)
{pp=lsb-p+1; p3=p;

 for(int i=1;i<=n;++i)b[i]=c[i]; m=n;
 while(p3>m){p3-=m;get_new();}
 jp=p3;
 do
   {a0[++ja]=b[jp++];
    if(jp>m){get_new();jp=1;}
   }
   while(ja<a0[0]);
 }
}

void get_a2()
{ for(int i=1;i<=a2[0];++i)a2[i]=am[i];
}

void get_am()
{int jm;
 for(int i=1;i<=ma;++i){b[i]=a[i];am[i]=a[i];}
 jm=ma; m=ma;
 for(int j=2;j<=ls;++j)
   {get_new();for(int i=1;i<=m;++i)am[++jm]=b[i];}
 am[0]=jm;
}

void get_prm()
{int pp,jp;
 get_am();
 lse=am[0]; //size loop in elem ***

 pq=q-p; pp=p;

 if(q <=lsb+lse ) {f1=1;a2[0]=0;a0[0]=q-p+1;get_a0(); }
}

```

```

    else if (p>lsb)
        {pp=(p-lsb)%lse; fl=2; p0=pp;
         if(p0+pq<=lse){a2[0]=0; a0[0]=pq+1;}
         else {a2[0]=(p0+pq)%lse; a0[0]=lse-p0+1;}
         get_a0();

    }
    else {fl=3; a0[0]=lsb-p+1;a2[0]=(pq-a0[0])%lse; get_a0();

    }
    if(a2[0]>0) get_a2();
    mm=(q-p+1-a0[0]-a2[0])/am[0];      //num of cycles
}

void get_data()
{fo<<"Data:"<<endl;
 fo<<n<<" "<<p<<" "<<q<<" "<<ss<<endl;
 for(int i=1;i<=n;++i)fo<<c[i]<<" ";
 fo<<endl<<"a0: ";
 for(int i=0;i<=a0[0];++i)fo<<a0[i]<<" ";
 fo<<endl<<"am: ";
 for(int i=0;i<=am[0];++i)fo<<am[i]<<" ";
 fo<<endl<<"a2: ";
 for(int i=0;i<=a2[0];++i)fo<<a2[i]<<" ";
 fo<<endl<<"mm: ";
 fo<<mm<<endl;
 fo<<"_____"<<endl;
}

void get_inv0()
{
    ss=0; x=0; memset(s,0,sizeof(s));
    m=a0[0]; if(m==0)x=0;
    else
        {insert_t(a0[m]);
         for(int i=m-1;i>0;--i)
             {insert_t(a0[i]);x+=sum_t(a0[i]-1);} // vb x
        }
    if(fl==1){ss=x; return;}

    y=0; memset(s,0,sizeof(s));
    m=am[0];
    insert_t(am[m]);
    for(int i=m-1;i>0;--i)
        {insert_t(am[i]);y+=sum_t(am[i]-1);} //vc y

    m=a2[0]; if(m==0){z=0;xz=0;}
    else
        {z=0; memset(s,0,sizeof(s));insert_t(a2[m]);
         for(int i=m-1;i>0;--i)
             {insert_t(a2[i]);z+=sum_t(a2[i]-1);} //va z
        yy=0; memset(s,0,sizeof(s)); m=am[0];
        for(int i=m;i>0;--i)insert_t(am[i]);
        for(int i=m;i>0;--i)yy+=sum_t(am[i]-1); //dcc yy
        xz=0; m=a2[0];memset(s,0,sizeof(s));
        for(int i=1;i<=m;++i)insert_t(a2[i]);
        r=m;m=a0[0]; for(int i=1;i<=m;++i) xz+=sum_t(a0[i]-1); //dba xz
    }
    yz=0;;memset(s,0,sizeof(s));
}

```

```

if(a2[0]!=0)
{m=a2[0];for(int i=1;i<=m;++i) insert_t(a2[i]);
r=m;m=am[0]; for(int i=1;i<=m;++i)yz+=sum_t(am[i]-1); //dca yz
}
xy=0;memset(s,0,sizeof(s));
if(a0[0]!=0)
{m=am[0];for(int i=1;i<=m;++i) insert_t(am[i]);
r=m;m=a0[0]; for(int i=1;i<=m;++i)xy+=sum_t(a0[i]-1); //dbc xy
}
ss=x+z+xz+mm*(y+yz+xy)+mm*(mm-1)/2*yy;
}
void determ_inv()
{get_prm();
 get_inv0();
}

void xly()
{res=-1;
 tt=0;for(int i=1;i<n;++i)tt+=c[i];
 tt=255-tt;
 for(int ii=1;ii<=tt;++ii)
 {if(n>1)for(int i=1;i<n;++i)b[i]=c[i];
 b[n]=ii; c[n]=ii;
 for(int i=1;i<=n;++i)b[i]=c[i];
 determ_l(); //Determ lengths + loop

 determ_inv();
 if(ss==kp){res=ii; return;}
}
}

int main()
{//clock_t aa=clock();
 fi>>t;
 for(int i=1;i<=t;++i)
 {fi>>n>>p>>q>>kp;
 if(n>1)for(int j=1;j<n;++j)fi>>c[j];
 xly(); fo<<res<<endl;
}
// clock_t bb=clock();
// fo<<"Time: "<<(double)(bb-aa)/1000<<" sec.";
}

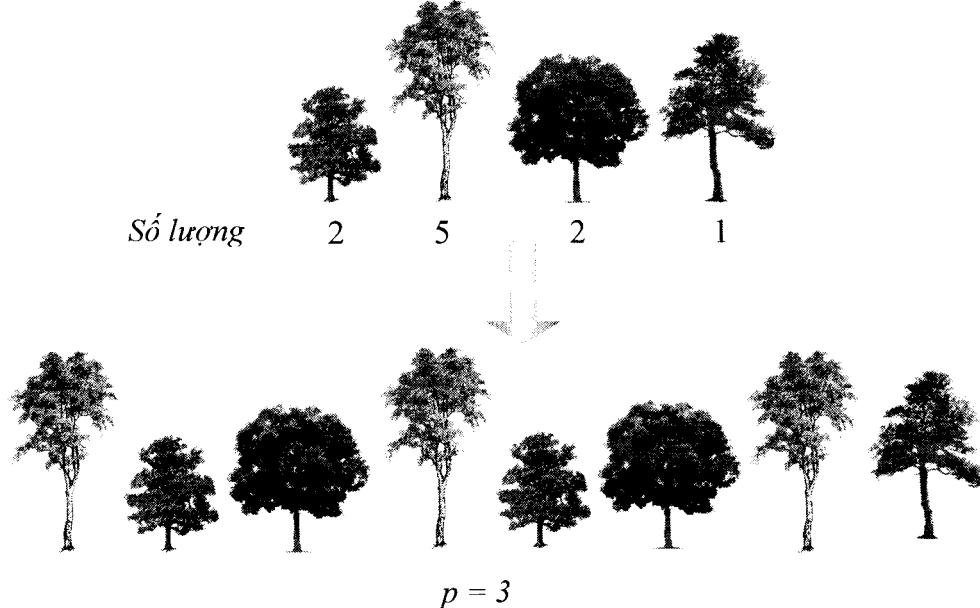
```

## BM31. TRỒNG CÂY XANH

Tên chương trình: ROWAN.???

Để đảm bảo cảnh quan chung và môi trường trong lành chính quyền thành phố quyết định trồng  $k$  loại cây xanh trên một con đường mới của thành phố. Người ta đã mua về  $a_i$  cây non loại cây thứ  $i$ .

Để đảm bảo mỹ thuật dự án trồng cây phải đảm bảo yêu cầu, sao cho trong dãy  $p$  cây bất kỳ liên tiếp mọi cây đều khác nhau. Dĩ nhiên, nếu trên đường trồng ít hơn  $p$  cây thì tất cả chúng phải khác nhau.



**Yêu cầu:** Với số lượng cây con đã mua, hãy xác định số lượng cây tối đa có thể trồng theo các yêu cầu đã nêu.

**Dữ liệu:** Vào từ file văn bản ROWAN.INP:

- Dòng đầu tiên chứa 2 số nguyên  $k$  và  $p$  ( $2 \leq k \leq 100\,000$ ,  $2 \leq p \leq k$ ),  
Dòng thứ  $i$  trong  $k$  dòng sau chứa số nguyên  $a_i$  ( $1 \leq a_i \leq 10^9$ ).

**Kết quả:** Đưa ra file văn bản TREES.OUT một số nguyên là kết quả tìm được.

**Ví dụ:**

ROWAN.INP
4 3
2
5
2
1

ROWAN.OUT
8



m31 ROI 07

```

#include<fstream>
#include<ctime>
using namespace std;
long long n,m,p,q,k,t,l,r,rx,rx;
long long a[1000001],b[1000001];
ifstream fi ("Rowan.inp");
ofstream fo ("Rowan.out");

void get_k(long long x)
{if(x<a[n]) {k=n; return;}
 else if(x>a[1]) {k=0;return;}
 lx=1;rx=n;
 while(lx<=rx)
 {t=(lx+rx)/2;
 if(x>=a[t]) rx=t-1;
 else lx=t+1;
 }
 if(x<a[t]) k=t; else k=rx;
}

int main()
{fi>>n>>p;b[0]=0;
for(int i=1;i<=n;++i) fi>>a[i];
sort(a+1,a+n+1,greater<int>());
for(int i=1;i<=n;++i)b[i]=b[i-1]+a[i];
l=0;r=(b[n]+p-1)/p;
while(l<=r)
{m=(l+r)/2;
get_k(m);
q=b[n]-b[k]-m*(p-k);
if(q>=0) l=m+1; else r=m-1;
}
m=r;
r=m*k+k+b[n]-b[k];
fo<<r;
}

```

## 7. Quản lý khoảng (Segment Control) bằng cây nhị phân

Xét bài toán quảng lý khoảng trên mảng một chiều. Mô hình toán học của lớp bài toán này có dạng: cho mảng số nguyên  $\mathbf{A} = (\mathbf{a}_1, \mathbf{a}_2, \dots, \mathbf{a}_n)$  và một dãy các truy vấn, mỗi truy vấn tác động lên dãy các giá trị từ vị trí  $L$  đến vị trí  $R$  của mảng ( $1 \leq L \leq R \leq n$ ). Các truy vấn cần xử lý theo trình tự xuất hiện. Các dạng truy vấn thường gặp là:

- ◆ Tăng giá trị mỗi phần tử của dãy từ vị trí  $L$  đến vị trí  $R$  một lượng nguyên là  $v$ , tức là thay  $a_i$  bằng  $a_i + v$  với  $i \in [L, R]$ ,  $v$  có thể nhận giá trị âm,
- ◆ Tăng giá trị mỗi phần tử của dãy từ vị trí  $L$  đến vị trí  $R$   $q$  lần, tức là thay  $a_i$  bằng  $q \times a_i$  với  $i \in [L, R]$ ,
- ◆ Giảm giá trị mỗi phần tử của dãy từ vị trí  $L$  đến vị trí  $R$   $q$  lần, tức là thay  $a_i$  bằng  $\frac{a_i}{q}$  với  $i \in [L, R]$ , đảm bảo trong khoảng này mọi  $a_i$  đều chia hết cho  $q$ ,
- ◆ Tính tổng  $\sum_{i=L}^R a_i$  hoặc tích  $\prod_{i=L}^R a_i$  và đưa ra theo mô đun  $p$ ,  $p$  có thể là duy nhất với mọi truy vấn lại này hoặc có thể khác nhau, xác định bởi từng truy vấn,
- ◆ Tìm **max**, **min** trong khoảng  $[L, R]$ .

Thông thường các giá trị  $a_i$  thuộc kiểu **int** (mặc dù điều này không đóng vai trò đặc biệt quan trọng, nhưng cần thiết để khai báo kiểu dữ liệu cho các đại lượng liên quan trong chương trình).

Có thể tồn tại thêm các ràng buộc phụ với mỗi loại truy vấn là thay đổi giá trị của dãy.

Với  $n$  đủ lớn (bậc  $10^4 - 10^6$ ) và số lượng truy vấn cũng lớn thì việc biến đổi trực tiếp giá trị của dãy  $\mathbf{A}$  là không thích hợp vì giải thuật sẽ có độ phức tạp  $O(n^2)$ .

Có nhiều cách tổ chức dữ liệu để xử lý nhưng ở đây ta sẽ xét phương pháp dùng cây nhị phân với các lý do:

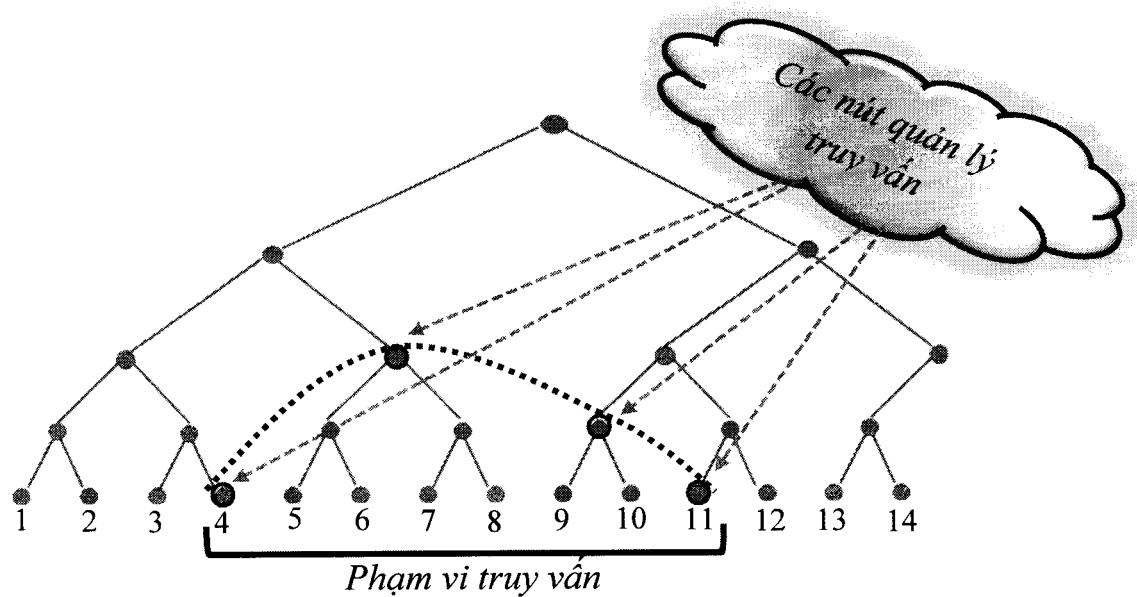
- ◆ Không quá phức tạp trong tổ chức dữ liệu và lập trình,
- ◆ Tính vận năng của giải thuật: cùng một mô hình dữ liệu có thể xử lý các loại truy vấn với nhiều điều kiện phụ đặc thù kèm theo.

Nguyên lý chung để giải các bài toán lớp này là tổ chức 2 cây nhị phân:

- ◆ Một cây để quản lý những truy vấn làm thay đổi giá trị của dãy,
- ◆ Một cây khác để quản lý các giá trị tổng/tích tiền tố của dãy  $\mathbf{A}$ .

Cây quản lý truy vấn ghi nhận thông tin truy vấn làm thay đổi giá trị của dãy có nhiệm vụ ghi lại thông tin về truy vấn ở một *số tối thiểu các nút*, sao cho đường

đi từ mỗi nút lá lưu giá trị  $a_i$  với  $i \in [L, R]$  về gốc đi qua một và chỉ một nút ghi thông tin của truy vấn (nút quản lý) và đường đi từ các nút lá còn lại về gốc không qua nút quản lý nào của truy vấn này. Với mỗi truy vấn số nút quản lý sẽ



không vượt quá độ cao của cây. Một nút có thể đóng vai trò quản lý của nhiều truy vấn.

Các hàm xử lý dữ liệu trong trường hợp tổng quát không khác nhiều so với hàm triển khai trong các bài toán thực tế thuộc lớp này vì vậy ta sẽ khảo sát các hàm đó qua việc xét một số bài toán cụ thể.

## VP17. CHỈ SỐ GIÁ TIÊU DÙNG

Tên chương trình: CPI.???

Có  $n$  mặt hàng được sử dụng để khảo sát và đánh giá chỉ số tiêu dùng (Consumer Price Index), đánh số từ 1 đến  $n$ . Đầu năm, đơn giá mặt hàng thứ  $i$  là  $a_i$ ,  $a_i$  – nguyên dương,  $i = 1 \div n$ . Các mặt hàng được phân loại sắp xếp theo các tiêu chí đảm bảo 2 mặt hàng đứng cạnh nhau trong dãy có cùng một số đặc trưng chung, ví dụ dưa bở đứng cạnh dưa hấu, dưa hấu – cạnh thanh long, thanh long – cạnh vú sữa . . . Chính vì vậy, khi điều kiện xã hội thay đổi như giá điện, giá xăng dầu tăng, mưa nhiều v. v. . . sẽ kéo theo việc thay đổi giá của một loạt mặt hàng đứng liên tiếp nhau.

Để khảo sát sự biến đổi chỉ số tiêu dùng người ta thường xuyên phải tính đơn giá trung bình của một mảng số mặt hàng liên tiếp nhau trong dãy từ thứ  $L$  đến thứ  $R$  ( $1 \leq L \leq R \leq n$ ).

Thông tin gửi về Tổng cục Thống kê dưới dạng bản ghi thuộc một trong 2 loại:

**1  $L$   $R$   $v$**  – đơn giá mỗi mặt hàng các loại từ  $L$  đến  $R$  tăng thêm  $v$  ( có thể  $v < 0$  cho trường hợp giá giảm,  $v$  – nguyên),

**0  $L$   $R$**  – yêu cầu cung cấp đơn giá trung bình hiện tại của các mặt hàng thuộc các loại từ  $L$  đến  $R$ .

Với mỗi yêu cầu cung cấp đơn giá trung bình hiện tại hãy tính và đưa ra trên một dòng với độ chính xác 4 chữ số sau dấu chấm thập phân.

**Dữ liệu:** Vào từ file văn bản CPI.INP:

- ◆ Dòng đầu tiên chứa 2 số nguyên  $n$  và  $m$ , trong đó  $m$  – số bản ghi cần xử lý ( $1 \leq n, m \leq 10^5$ ),
- ◆ Dòng thứ 2 chứa  $n$  số nguyên  $a_1, a_2, \dots, a_n$  ( $1 \leq a_i \leq 10^4$ ,  $i = 1 \div n$ ),
- ◆ Mỗi dòng trong  $m$  dòng sau chứa thông tin về một bản ghi cần xử lý, trong đó  $|v| \leq 10^4$ , dữ liệu đảm bảo không xuất hiện trường hợp đơn giá âm.

**Kết quả:** Đưa ra file văn bản CPI.OUT các đơn giá trung bình tính được theo yêu cầu, mỗi giá trị đưa ra trên một dòng.

**Ví dụ:**

CPI.INP									
11	5								
1	2	3	2	3	2	3	2	4	5
1	2	4	10						
0	1	5							
1	4	10	-1						
0	3	7							
0	6	11							

CPI.OUT									
8.2000									
5.8000									
2.8333									



## *Giải thuật*

Cần tổ chức 2 cây nhị phân **B** và **C**;

- ↳ **B** – lưu trữ tổng tiền tố của dãy,
- ↳ **C** – quản lý các truy vấn loại **L R v**.

### *a) Chuẩn bị dữ liệu:*

Với **n** cho trước, nút lá của các cây nói trên có số thứ tự là  $\text{nb} = 2^k$ , trong đó **k** là số nguyên nhỏ nhất thỏa mãn điều kiện  $2^k > n-1$ . Như vậy **nb** là số nguyên trong dạng biểu diễn nhị phân chỉ có 1 bit 1 và vị trí của bit này là ở bên trái bit có

```

k=n-1;
for(int i=20;i>=0;--i)
    if((k>>i)&1){nb=1<<(i+1);break;}

```

nghĩa đầu tiên của **n-1**.

Các giá trị **a<sub>i</sub>**, **i = 1 ÷ n** được điền vào nút lá tương ứng của **B**:

Chuẩn bị giá trị cho các nút còn lại của **B**: nếu **i** không phải là nút lá thì

```

for(int i=0;i<n;++i) fi>>b[nb+i];

```

$$b_i = b_{2^*i} + b_{2^*i+1}$$

Công thức trên được áp dụng với **i** thay đổi từ **nb-1** tới 1.

### *b) Xử lý truy vấn dạng **L R xx**:*

$$\left\{ \begin{array}{l} u = nb + L - 1, \\ v = nb + R - 1. \end{array} \right.$$

Chỉ số các nút lá tương ứng với khoảng cần xử lý là

Việc xử lý bao gồm hai phần:

- ↳ Xử lý đồng thời 2 đường đi bên trái và bên phải xuất phát tương ứng từ **u** và **v** về nút cha chung gần nhất. Ví dụ, với **n = 11**, **L = 4** và **R = 10**, nút cha chung gần nhất là 1. Việc xử lý thông tin trên hai đường là tương tự, giống nhau ở mức độ “ánh xạ gương”,
- ↳ Xử lý tiếp đường đi chung từ nút cha chung về nút gốc.

#### *b1) Xử lý 2 đường đi:*

Các nút của cây có thể phân loại thành các mức:

- Mức 0 là mức chứa các nút lá, mỗi nút chỉ chứa thông tin của chính nút đó,
- Mức 1: bao gồm các nút nối trực tiếp với nút lá trên đường đi *từ lá về gốc*, mỗi nút thuộc mức 1, có thể trừ nút ở đường đi phải nhất, chứa thông tin tổng hợp của 2 nút,
- Mức *i*: bao gồm các nút nối trực tiếp với nút ở mức *i-1* trên đường đi *từ lá về gốc*, mỗi nút thuộc mức *i*, có thể trừ nút ở đường đi phải nhất, chứa thông tin tổng hợp của  $2^i$  nút,
- Mức cuối cùng: chỉ bao gồm nút gốc và chứa thông tin tổng hợp của n nút.

**Xử lý đường đi bên trái:** xuất phát từ nút lá *u* ( $= nb + L - 1$ ) xử lý nút và đi lên nút cha *u/2*.

Xử lý nút:

**Trường hợp *u* là lẻ:** Gặp nút chỉ số lẻ đầu tiên – ghi nhận thay đổi vào nút *u*:

- Ghi nhận giá trị mới của nút đó trong *B* (nếu đó là nút mức *k* thì giá trị của nút tăng thêm một lượng là *k\*xx*),
- Đánh dấu nút đó là nút quả lý khoảng trong *C*, giá trị nút tương ứng tăng thêm một lượng là *xx*,
- Cập nhật giá trị ở các nút trên đường đi từ *u* về gốc trong *B*.

Để nhận biết đã có nút lẻ được đánh dấu hay chưa: dùng cờ nhận dạng (ví dụ *fxu*, *fxu* = 0 – chưa gặp nút lẻ trước đó, *fxu* = 1 – đã gặp và đã ghi nhận).

**Trường hợp nút *u* là chẵn:** ghi nhận thay đổi vào nút *u+1* nếu thỏa mãn các điều kiện:

- Trên đường đi đã có nút được đánh dấu (*fxu* = 1),
- Nút cùng mức liền kề bên phải (nút *u+1*) không thuộc đường đi bên phải (tức là *u+1* ≠ *v*).

Việc ghi nhận thay đổi được thực hiện như đã nêu ở trường hợp gấp nút lẻ đầu

```
if((u&1) && fxu==0){b[u]+=k*xx;fxu=1;c[u]+=xx;upd_t(u,k*xx);}  
else  
if((u&1)==0 && fxu==1 && u+1!=v)  
{b[u+1]+=k*xx;c[u+1]+=xx;upd_t(u+1,k*xx);}
```

tiên.

**Xử lý đường đi bên phải:** xuất phát từ *v*, xử lý nút và đi lên nút *v/2*. Việc xử lý cũng tiến hành tương tự như với trường hợp đường đi bên trái, nhưng điều kiện kiểm tra là gấp nút lẻ đầu tiên và nút liền kề cùng mức bên trái không thuộc

```
if(((v&1)==0) && fxv==0){b[v]+=k*xx;c[v]+=xx;fxv=1;upd_t(v,k*xx);}  
else  
if((v&1)==1 && fxv==1 && u+1!=v)  
{b[v-1]+=k*xx;c[v-1]+=xx;upd_t(v-1,k*xx);}
```

đường đi bên phải.

Lưu ý: khi chuyển sang mức mới cần tăng  $k$ .

### b2) Xử lý đường đi chung về gốc:

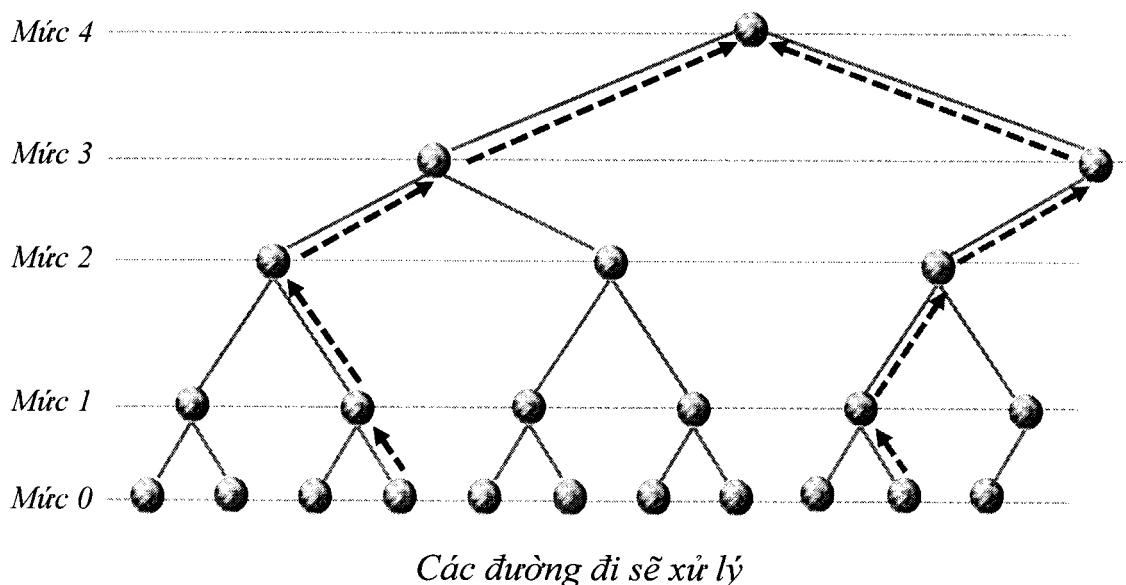
Phân biệt các trường hợp:

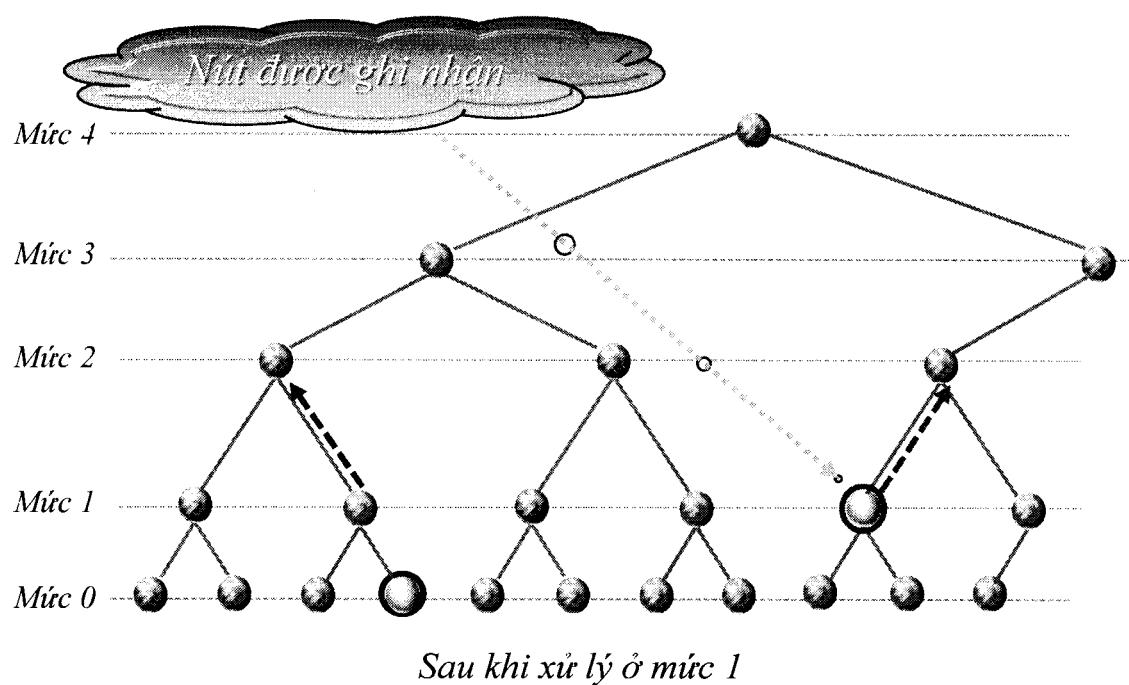
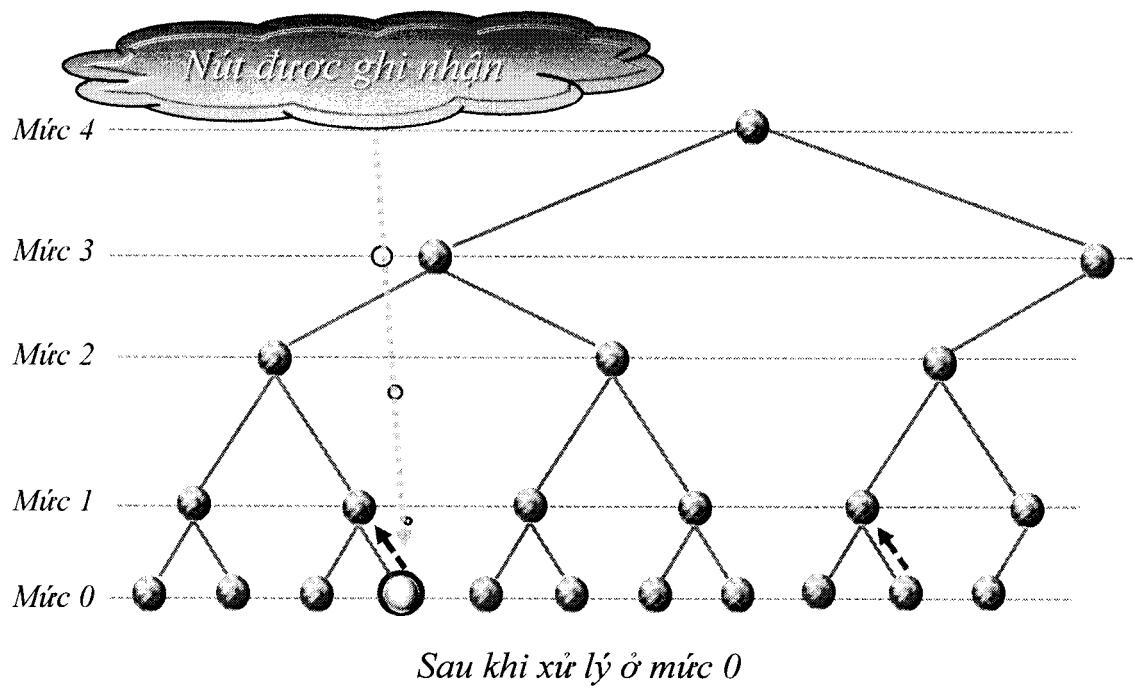
- Chưa có nút nào được ghi nhận ( $\text{fxu} = 0$  và  $\text{fxv} = 0$ ) – ghi nhận thông tin vào nút chung,
- Nếu ở đường nào đó chưa có ghi nhận – ghi nhận vào nút riêng của đầu đường đó.

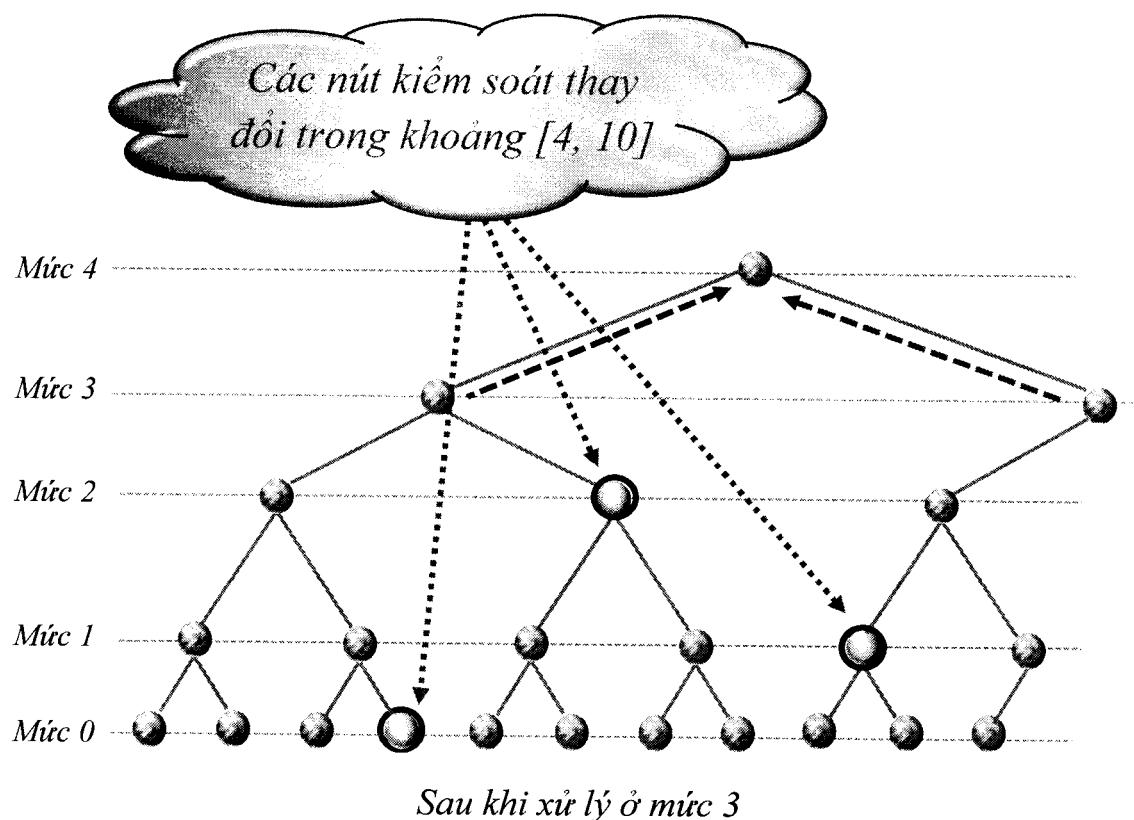
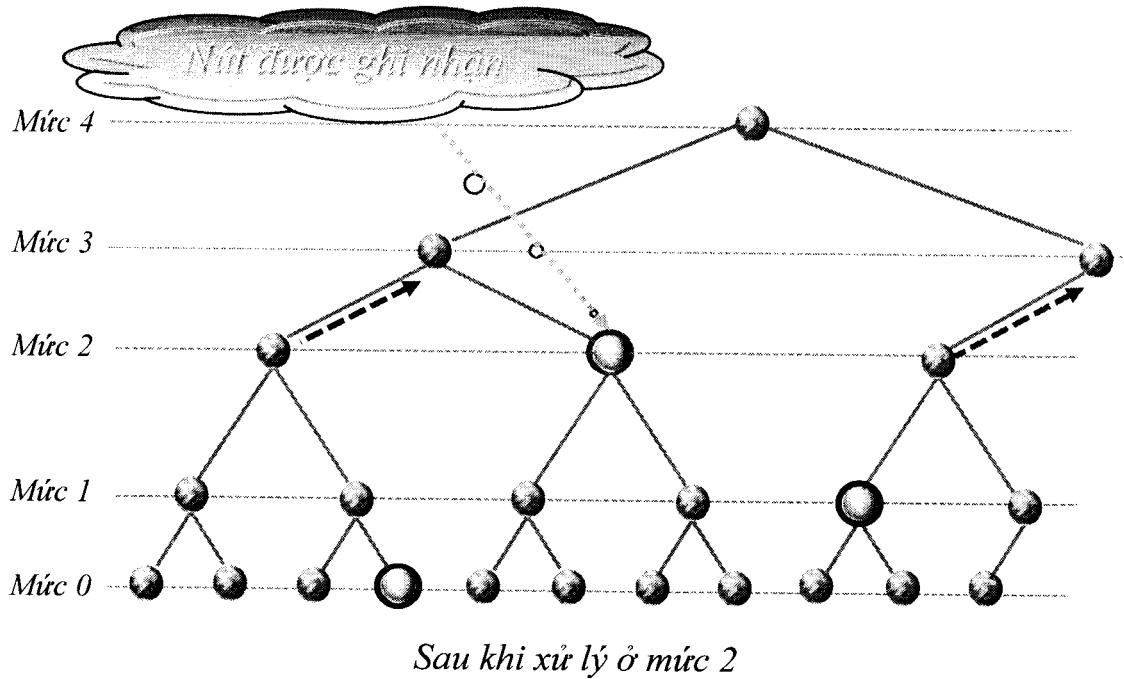
```
if ((fxu==0) && (fxv==0)) {b[u]+=k*xx;c[u]+=xx;upd_t(u,k*xx);return;}
k>>=1;
if (fxu==0) {b[2*u]+=k*xx;c[2*u]+=xx;upd_t(2*u,k*xx);return;}
if (fxv==0) {b[2*u+1]+=k*xx;c[2*u+1]+=xx;upd_t(2*u+1,k*xx);}
```

Lưu ý: cần chỉnh lý k khi cần thiết.

Kết quả các bước xử lý với trường hợp  $n = 11$ ,  $L = 4$  và  $R = 10$ :







Lưu ý: một nút có thể tham gia kiểm soát nhiều khoảng.

### c) Xử lý truy vấn dạng $0 \ L \ R$

Xử lý riêng trường hợp  $L = R$ ,

Xử lý trường hợp  $L \neq R$ :

Duyệt 2 đường đi trái và phải xuất phát tương ứng từ  $u = nb + L - 1$  và  $v = nb + R - 1$  tới nút cha chung gần nhất,

Trong quá trình duyệt: đếm số nút thuộc sự kiểm soát của đường đi. Một nút gọi là thuộc sự kiểm soát của đường đi nếu đường đi từ nút đó về gốc có chứa nút của đường đang xét. Gọi số lượng nút đếm được là  $slu$  và  $slv$  ứng với các đường phải và trái.

Mức 4

Mức 3

Mức 2

Mức 1

Mức 0

Các đường đi sẽ xử lý khi tính đơn giá trung bình trong khoảng [6, 11]

Ở đường đi bên phải: nút đang xét là  $u$  và nút cha của nó là  $u1$ .

Trường hợp  $u$  chẵn: đường đi sẽ kiểm soát cây con có đỉnh là  $u+1$ , tổng đơn giá sẽ  $tu$  sẽ bao gồm cả  $b[u+1]$ , số lượng nút kiểm soát tăng thêm  $k$  – số lượng nút của cây con,

Trường hợp  $u$  lẻ: không có sự tham gia của các đại lượng nói trên!

Trong mọi trường hợp của  $u$ : đơn giá phải tăng thêm một lượng tỷ lệ với số phần tử kiểm soát được và ảnh hưởng bởi các lần tăng giá quyết định bởi  $c[u1]$  – nút

```
if((u&1)==0){slu+=k;tu+=(b[u+1]+slu*c[u1]);}
else tu+=slu*c[u1];
```

mà nó sắp đi qua.

```
if((v&1)==1){slv+=k;tv+=(b[v-1]+slv*c[v1]);}  
else tv+=slv*c[v1];
```

Tương tự như vậy đối với **v** nếu đổi việc kiểm từ chẵn sang lẻ:

Xử lý đường đi từ đỉnh cha chung về gốc: chỉ phụ thuộc vào **sl** – số lượng phần tử trong khoảng (**sl = R - L + 1**) và **c[u1]** với các **u1** trên đường đi.

Tổng đơn giá cần tìm sẽ là tổng phần tính được trên đoạn đường chung với các giá trị tính được trên mỗi đường đi riêng bên phải và bên trái.

```
while(u1!=0)  
{res+=(sl*c[u1]); u1/=2;}  
res+=tu+tv;  
ans=(double)res/(double)(r-l+1);  
fo<<fixed<<setprecision(4)<<ans<<'\n';
```

Lưu ý cách chuyển đổi kiểu dữ liệu và cách đưa ra đúng 4 chữ số sau dấu chấm thập phân.

Độ phức tạp của giải thuật: O(nlogn).

## Chương trình

```
#include <fstream>
#include <cmath>
#include <ctime>
#include <iomanip>
using namespace std;
ifstream fi ("Cpi.inp");
ofstream fo ("Cpi.out");
int64_t b[400010]={0},c[400010]={0},res,x,p,rx,rxp;
int64_t n,nb,pp,m,xx,l,r,g,u,v,k,fxu,fxv,s12=0;
double ans;

void xly_0()
{int64_t tu=0,tv=0;
 res=0; fi>>l>>r;
 int64_t u1,v1,sl,slu=1,slv=0;
 u=nb+l-1; v=nb+r-1; sl=r-l+1;k=1;
 tu=b[u];
 if(u!=v)
 {
    tv=b[v];u1=u/2;v1=v/2;slv=1;
    while(u1!=v1)
    {
      if((u&1)==0){slu+=k;tu+=(b[u+1]+slu*c[u1]);}else tu+=slu*c[u1];
      if((v&1)==1){slv+=k;tv+=(b[v-1]+slv*c[v1]);}else tv+=slv*c[v1];
      u=u1; v=v1;u1=u/2;v1=v/2;k<<=1;
    }
    while(u1!=0)
    {rest+=(sl*c[u1]); u1/=2;}
    rest+=tu+tv;
    ans=(double)res/(double)(r-l+1);
    fo<<fixed<<setprecision(4)<<ans<<'\n';
}

void upd_t(int64_t y,int64_t z)
{int64_t t;
 while(y>1)
  {t=y/2;b[t]+=z;y=t;}
}

void xly_1()
{fxu=0;fxv=0; fi>>l>>r>>xx;
 u=nb+l-1; v=nb+r-1;
 k=1;
 while(u!=v)
 { if((u&1) && fxu==0){b[u]+=k*xx;fxu=1;c[u]+=xx;upd_t(u,k*xx);}
   else if((u&1)==0 && fxu==1 &&
 u+1!=v){b[u+1]+=k*xx;c[u+1]+=xx;upd_t(u+1,k*xx);}
   if(((v&1)==0) && fxv==0){b[v]+=k*xx;c[v]+=xx;fxv=1;upd_t(v,k*xx);}
   else if((v&1)==1 && fxv==1 && u+1!=v){b[v-1]+=k*xx;c[v-
1]+=xx;upd_t(v-1,k*xx);}
   k<<=1;u/=2;v/=2;
 }
 if((fxu==0) && (fxv==0)){b[u]+=k*xx;c[u]+=xx;upd_t(u,k*xx);return;}
 k>>=1;
 if(fxu==0){b[2*u]+=k*xx;c[2*u]+=xx;upd_t(2*u,k*xx);return;}
 if(fxv==0){b[2*u+1]+=k*xx;c[2*u+1]+=xx;upd_t(2*u+1,k*xx);}
}
```

```

void chbi()
{
    for(int i=nb-1;i>0;--i)
        b[i]=b[2*i]+b[2*i+1];

}

int main()
{clock_t aa=clock();
 fi>>n>>m;k=n-1;
 for(int i=20;i>=0;--i)
    if((k>>i)&1){nb=1<<(i+1);break;}

for(int i=0;i<n;++i)fi>>b[nb+i];
    chbi();
    for(int i=1;i<=m;++i)
    {fi>>g;
     if(g)xly_1(); else xly_0();
    }
    clock_t bb=clock();
    fo<<"\nTime: "<<(double)(bb-aa)/1000<<" sec";
}

```

## VP10. HỆ THỐNG CHẤM ĐIỂM

Tên chương trình: EVALSYS.???

Kỳ thi Olympic Tin học năm nay có  $n$  thí sinh tham gia. Hệ thống chấm điểm mới cho thí sinh biết điểm ngay sau khi gửi bài chấm. Kết quả chấm có thể làm tâm trạng thí sinh thay đổi nhiều.

Đầu cuộc tâm trạng của các thí sinh đều bằng 1.

Ban Giám khảo rất quan tâm đến diễn biến tâm trạng của các thí sinh và yêu cầu xử lý truy vấn thuộc các loại:

- **0  $L$   $R$   $P$**  – tính tích tâm trạng tất cả các thí sinh từ  $L$  đến  $R$  và đưa ra theo mô đun  $P$ ,
- **1  $L$   $R$   $X$**  – tâm trạng mỗi thí sinh từ  $L$  đến  $R$  được nhân lên  $X$  sau khi biết kết quả gửi bài,
- **2  $L$   $R$   $X$**  – tâm trạng mỗi thí sinh từ  $L$  đến  $R$  bị chia cho  $X$  sau khi biết kết quả gửi bài, dữ liệu đảm bảo các giá trị tâm trạng đều chia hết cho  $X$ .

Trong mọi truy vấn đều có  $1 \leq L \leq R \leq n$ ,  $1 \leq P \leq 10^9 + 7$ ,  $1 \leq X \leq 100$ .

Với mỗi truy vấn loại 0 hãy đưa kết quả tính được.

**Dữ liệu:** Vào từ file văn bản EVALSYS.INP:

- ◆ Dòng đầu tiên chứa 2 số nguyên  $n$  và  $m$ , trong đó  $m$  – số truy vấn cần xử lý ( $1 \leq n, m \leq 50000$ ),
- ◆ Mỗi dòng trong  $m$  dòng sau chứa các thông tin về một truy vấn.

**Kết quả:** Đưa ra file văn bản EVALSYS.OUT các kết quả ứng với truy vấn loại 0, mỗi kết quả đưa ra trên một dòng.

**Ví dụ:**

EVALSYS.INP	EVALSYS.OUT
5 5	
0 1 5 1000000007	1
1 2 3 6	36
0 1 5 1000000007	
2 2 3 3	4
0 1 5 1000000007	



## Giải thuật

Để giải quyết trọn vẹn mọi yêu cầu đề ra cần:

- ➔ Phân tích số  $x$  trong các truy vấn loại 1 và 2 thành thừa số nguyên tố,
- ➔ Tổ chức lưu trữ tích hình thành trong quá trình xử lý các truy vấn dưới dạng tổng số mũ của các thừa số nguyên tố,
- ➔ Áp dụng sơ đồ nhân Ai Cập tính phần dư của phép chia một lũy thừa cho một số nguyên (*Xem giải thuật bài VJ18. Sân vận động*).

Trong 100 số nguyên đầu tiên có 25 số nguyên tố:

**2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43,**

**47, 53, 59, 61, 67, 71, 73, 79, 83, 89, 97**

Có thể khởi tạo mảng nguyên 25 phần tử với giá trị đầu là các số nguyên tố đã nêu.

Cần thiết phải xây dựng một bảng lưu trữ kết quả phân tích số nguyên  $x$  ( $1 \leq x \leq 100$ ) ra thừa số nguyên tố. Kết quả phân tích nên lưu trữ dưới dạng bảng 2 chiều **int ptab[101][25]**, trong đó **ptab[i][j]** là số mũ của số nguyên tố thứ  $j$  (dãy các số nguyên tố được đánh số bắt đầu từ 0) có mặt trong kết quả phân tích số nguyên  $i$  ra thừa số nguyên tố. Ví dụ, với  $i = 28 = 2^2 \times 7$  dòng 28 của ptab có dạng:

(2, 0, 0, 1, 0)

```
void chbi_b()
{
    int t;
    for(int i=2;i<=100;++i)
    {
        for(int j=0;j<25;++j)
            if(i%pprim[j]==0)
                {t=i/pprim[j];
                 for(int i2=0;i2<25;++i2)
                     ptab[i][i2]=ptab[t][i2];
                  ++ptab[i][j]; break;
                }
    }
}
```

việc xây dựng **ptab** không phải là vấn đề phức tạp:

Hàm tính phần dư của  $x^y$  chia cho  $p$  đã được xét và áp dụng ở một số bài toán trước, cũng là một công việc đơn giản:

```
int64_t fx(int x,int64_t y)
{int64_t tw,tg=1;
 tw=pprim[x];
 while(y>0)
 {if(y&1)tg=(tg*tw)%p;
  tw=(tw*tw)%p; y>>=1;
 }
 return(tg);
}
```

Ở giải thuật độ phức tạp  $O(n^2)$  giá trị của mỗi phần tử của mảng được tính lại khi thuộc vùng biến đổi của truy vấn và lưu dưới dạng mảng 25 phần tử xác định kết quả phân tích ra thừa số nguyên tố. Như vậy, các phép nhân, chia được thay thế bằng phép cộn, từ số mũ và miền giá trị của kết quả không vượt quá khả năng biểu diễn của các dạng dữ liệu chuẩn.

Với độ phức tạp  $O(n^2)$  chương trình chỉ hoạt động có hiệu quả với  $n$  không quá vài ngàn.

*Giải thuật đồ phác tạp*  $O(n^2)$

```
#include <fstream>
#include <cmath>
#include <ctime>
using namespace std;
ifstream fi ("Evalsyst.inp");
ofstream fo ("Evalsyst.out");
int pprim[25]=
{2,3,5,7,11,13,17,19,23,29,31,37,41,43,47,53,59,61,67,71,73,79,83,89,97};
int ptab[101][25]={0};
int64_t a[50001][25]={0},res;
int n,m,g,l,r,x;

void chbi_b()      //Tạo bảng phán tích thwa số nguyên tố
{
    int t;
    for(int i=2;i<=100;++i)
    {
        for(int j=0;j<25;++j)
        if(i%pprim[j]==0)
            {t=i/pprim[j];
             for(int i2=0;i2<25;++i2)
                 ptab[i][i2]=ptab[t][i2];
             ++ptab[i][j]; break;
            }
    }
}

void xly_1()
{for(int i=1;i<=r;++i)
    for(int j=0;j<25;++j)a[i][j]+=ptab[x][j];
}

void xly_2()
{for(int i=1;i<=r;++i)
    for(int j=0;j<25;++j)a[i][j]-=ptab[x][j];
}

int64_t fx(int ii,int64_t y)
{int64_t tw,tg=1;
 tw=pprim[ii];
 while(y>0)
 {if(y&1)tg=(tg*tw)%x;
  tw=(tw*tw)%x; y>>=1;
 }
 return(tg);
}

void xly_0()
{res=1;
 for(int i=1;i<=r;++i)
    for(int j=0;j<25;++j)
        if(a[i][j]>0)res=(res* fx(j,a[i][j]))%x;
 fo<<res<<'\n';
}

int main()
{clock_t aa=clock();
 chbi_b();
 fi>>n>>m;
```

```
for(int i=1;i<=m;++i)
{fi>>g>>l>>r>>x;
 switch(g)
 {case 0:xly_0();break;
 case 1:if(x>1)xly_1();break;
 case 2:if(x>1)xly_2();break;
 }
}
clock_t bb=clock();
fo<<"\nTime: "<<(double)(bb-aa)/1000<<" sec";
}
```

## *Giải thuật độ phức tạp O(nlogn)*

Để có giải thuật hiệu quả cao cần áp dụng kỹ thuật quản lý khoảng bằng cây nhị phân (*Xem VP17. Chỉ số giá tiêu dùng*).

Thay vì 2 bảng một chiều **B** và **C** trong VP17 ở đây ta cần dùng hai mảng 2 chiều

```
int64_t b[25][200010]={0},c[25][200010]={0};
```

Việc các cặp bảng **b[i][2000010]** và **c[i][2000010]** được thực hiện theo đúng quy trình xử lý đã nêu.

## *Chương trình theo giải thuật độ phức tạp O(nlogn)*

```
#include <fstream>
#include <cmath>
#include <ctime>
using namespace std;
ifstream fi ("Evalsyst.inp");
ofstream fo ("Evalsyst.out");
int
pprim[25]={2,3,5,7,11,13,17,19,23,29,31,37,41,43,47,53,59,61,67,71,73,79
,83,89,97};
int ptab[101][25]={0};

int64_t b[25][200010]={0},c[25][200010]={0},res,x,p,rx,RP;
int64_t n,nb,pp,m,xx,l,r,g,u,v,k,f xu,f xv,s12=0,a2[102]={0};

void chbi_b()      //Tạo bảng phân tích thواب số nguyên tố
{
    int t;
    for(int i=2;i<=100;++i)
    {
        for(int j=0;j<25;++j)
        if(i%pprim[j]==0)
            {t=i/pprim[j];
             for(int i2=0;i2<25;++i2)
                ptab[i][i2]=ptab[t][i2];
             ++ptab[i][j]; break;
            }
    }
}

int64_t fx(int ii,int64_t y)  //Tim so du cua luy thua
{
int64_t tw,tg=1;
tw=pprim[ii];
while(y>0)
{if(y&1) tg=(tg*tw)%x;
 tw=(tw*tw)%x; y>>=1;
}
return(tg);
}

void xly_0()
{
int64_t tu[25],tv[25];
res=1;
for(int i=0;i<25;++i)
{
int64_t u1,v1,s1,slu=1,slv=0;
u=nb+l-1; v=nb+r-1; sl=r-l+1;k=1;
tu[i]=b[i][u];
tv[i]=c[i][v];
}
```

```

if(u!=v)
{
    tv[i]=b[i][v];u1=u/2;v1=v/2;slv=1;
    while(u1!=v1)
    {
        if((u&1)==0){slu+=k;tu[i]+=(b[i][u+1]+slu*c[i][u1]);}else
        tu[i]+=slu*c[i][u1];
        if((v&1)==1){slv+=k;tv[i]+=(b[i][v-1]+slv*c[i][v1]);}else
        tv[i]+=slv*c[i][v1];
        u=u1; v=v1;u1=u/2;v1=v/2;k<=1;
    }
}
while(u1!=0)
{
    res=(res*fx(i,sl*c[i][u1]))%x; u1/=2;
}
for(int i=0;i<25;++i)res=(res*fx(i,tu[i]+tv[i]))%x;
fo<<res<<'\n';
}

void upd_t(int ii,int64_t y,int64_t z)
{
int64_t t;
while(y>1)
{
    t=y/2;b[ii][t]+=z;y=t;
}
}

void xly_1b(int ii,int64_t xx)
{
fxu=0;fxv=0;
u=nb+l-1; v=nb+r-1;
k=1;
while(u!=v)
{
    if((u&1) &&
fxu==0){b[ii][u]+=k*xx;fxu=1;c[ii][u]==xx;upd_t(ii,u,k*xx);}
    else if((u&1)==0 && fxu==1 &&
u+1!=v){b[ii][u+1]+=k*xx;c[ii][u+1]==xx;upd_t(ii,u+1,k*xx);}
    if(((v&1)==0) &&
fxv==0){b[ii][v]+=k*xx;c[ii][v]==xx;fxv=1;upd_t(ii,v,k*xx);}
    else if((v&1)==1 && fxv==1 && u+1!=v){b[ii][v-1]+=k*xx;c[ii][v-
1]==xx;upd_t(ii,v-1,k*xx);}
    k<=1;u/=2;v/=2;
}
if((fxu==0) &&
(fxv==0)){b[ii][u]+=k*xx;c[ii][u]==xx;upd_t(ii,u,k*xx);return;}
k>>1;
if(fxu==0){b[ii][2*u]+=k*xx;c[ii][2*u]==xx;upd_t(ii,2*u,k*xx);return;}
if(fxv==0){b[ii][2*u+1]+=k*xx;c[ii][2*u+1]==xx;upd_t(ii,2*u+1,k*xx);}
}

void xly_1()
{
int64_t tx;
for(int i=0;i<25;++i)
{
    if(ptab[x][i]>0)
    {
        tx=ptab[x][i];xly_1b(i,tx);
    }
}

void xly_2()
{
int64_t tx;
for(int i=0;i<25;++i)
{
    if(ptab[x][i]>0){tx=-ptab[x][i];xly_1b(i,tx);}
}
}

```

```
int main()
{clock_t aa=clock();
 chbi_b();
 fi>>n>>m;k=n-1;
 for(int i=20;i>=0;--i)
 if((k>>i)&1){nb=1<<(i+1);break;}

for(int i=1;i<=m;++i)
{fi>>g>>l>>r>>x;
 switch(g)
 {case 0:{xly_0();break;}
 case 1:{if(x>1)xly_1();break;}
 case 2:{if(x>1)xly_2();break;}
 }
}
clock_t bb=clock();
fo<<"\nTime: "<<(double)(bb-aa)/1000<<" sec";
}
```

## 8. Cây tiền tố

Cây tiền tố được sử dụng để quản lý và phục vụ tìm kiếm nhanh các đối tượng có phần đầu giống nhau.

Ví dụ, khi tìm kiếm trên Google.com, nếu ta gõ vào cửa sổ tìm kiếm ký tự **O** hệ thống sẽ đưa ra ra một số đối tượng bắt đầu bằng chữ cái O:

*Ola*

*Ongame*

*One piece*

*Oggy*

Nếu ta gõ tiếp **OI** thì thông tin đưa ra sẽ là:

*Olympus*

*Ola*

*Ola me*

*Ole*

Khi gõ **Oly** sẽ có nội dung gợi ý lựa chọn:

*Olympus*

*Olympic*

*Olympia*

*Olympus has fallen*

Nếu nội dung gõ trong thanh tìm kiếm là **Olympiad in** thì ở cửa sổ dự báo sẽ có nội dung:

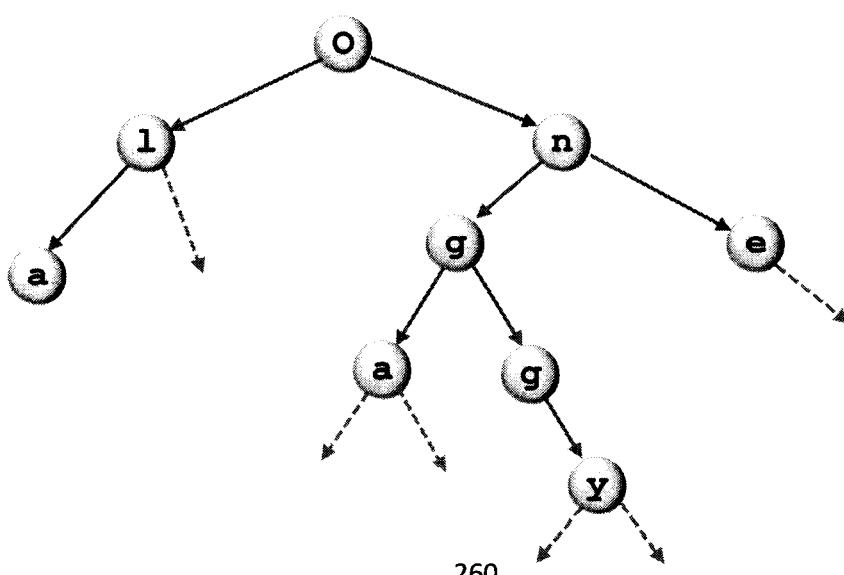
*Olympiad india*

*Olympiad in informatics*

*Olympiad inequalities*

*Olympiad in chippenham*

Để tìm kiếm nhanh các đối tượng có phần đầu giống nhau thông tin được lưu trữ



dưới dạng cây, mỗi nút là một ký tự.

Thông tin lưu trữ ở mỗi nút bao gồm:

- ◆ Ký tự tương ứng với nút,
- ◆ Dấu hiệu kết thúc một xâu độc lập,
- ◆ Các thông tin định vị xâu độc lập,
- ◆ Các mốc nối tới những nút tiếp theo,
- ◆ . . . . .

Việc tạo và xử lý cây tiền tố là nội dung chủ yếu của các hệ quản trị cơ sở dữ liệu văn bản.

Dưới đây ta sẽ xét một số phép xử lý đặc thù trên cây tiền tố nhị phân quản lý các số nhị phân thông qua một bài toán cụ thể.

*Bài toán:*

## VP08. SỐ LƯỢNG KHOẢNG

Tên chương trình: SEGNUM.???

Cho mảng số nguyên không âm  $\mathbf{A} = (\mathbf{a}_1, \mathbf{a}_2, \dots, \mathbf{a}_n)$ . Với số  $x$  nguyên không âm cho trước hãy tính số lượng cặp số ( $i, x$ ) thỏa mãn các điều kiện:

- $1 \leq i \leq x \leq n$ ,
- $x \leq \mathbf{a}_1 \text{ xor } \mathbf{a}_{i+1} \text{ xor } \dots \text{ xor } \mathbf{a}_x$ .

Ở đây  $xor$  là phép tính cộng bit không nhớ (phép **xor** trong PASCAL hay **^** trong C/C++).

**Dữ liệu:** Vào từ file văn bản SEGNUM.INP:

- ◆ Dòng đầu tiên chứa 2 số nguyên  $n$  và  $x$  ( $1 \leq n \leq 10^5$ ,  $0 \leq x \leq 10^9$ ),
- ◆ Dòng thứ 2 chứa  $n$  số nguyên  $\mathbf{a}_1, \mathbf{a}_2, \dots, \mathbf{a}_n$  ( $0 \leq \mathbf{a}_i \leq 10^9$ ,  $i = 1 \div n$ ).

**Kết quả:** Đưa ra file văn bản SEGNUM.OUT một số nguyên – số lượng cặp tìm được.

*Ví dụ:*

SEGNUM.INP
5 0
1 2 3 4 5

SEGNUM.OUT
15

## *Phân tích giải thuật*

Giải thuật tầm thường của bài toán là lần lượt xét đoạn độ dài 1, 2, 3, ... các liên tiếp nhau của dãy A, tính giá trị của dãy phép xor tương ứng, so sánh với x và tích lũy kết quả khi cần thiết. Độ phức tạp của giải thuật sẽ là  $O(n^3)$ .

Để có các giải thuật với độ phức tạp thấp hơn cần xử lý toán học và tổ chức dữ liệu một cách hợp lý.

### *Giải thuật độ phức tạp $O(n^2)$*

Phần lớn các bài toán xem xét đoạn các phần tử liên tiếp nhau đều liên quan tới một loại *hàm tiền tố* nào đó.

Hàm tiền tố đơn giản nhất có thể xây dựng là dãy  $B = (b_1, b_2, b_3, \dots, b_n)$ , trong đó:

- $b_1 = a_1$ ,
- $b_i = a_1 \text{ xor } a_2 \text{ xor } \dots \text{ xor } a_i, i = 2 \div n$ .

Việc tính  $B$  có thể thực hiện với độ phức tạp  $O(n)$ :

- $b_1 = a_1$ ,
- $b_i = b_{i-1} \text{ xor } a_i, i = 2 \div n$ .

So sánh  $b_i$  với  $x$  ta có thể tìm được các khoảng bắt đầu từ  $a_1$  thỏa mãn điều kiện đầu bài ( $i = 1 \div n$ ).

Thực hiện phép biến đổi  $b_i = b_i \text{ xor } b_1, i = 2 \div n$ , so sánh  $b_i$  với  $x$  ( $i = 2 \div n$ ). ta có thể tìm được các khoảng bắt đầu từ  $a_2$  thỏa mãn điều kiện đầu bài.

Thực hiện phép biến đổi  $b_i = b_i \text{ xor } b_2, i = 3 \div n$ , so sánh  $b_i$  với  $x$  ( $i = 3 \div n$ ). ta có thể tìm được các khoảng bắt đầu từ  $a_3$  thỏa mãn điều kiện đầu bài ...

Đây là giải thuật dễ lập trình nhất. Các giá trị ban đầu của A không cần phải lưu trữ.

```

#include <fstream>
#include <cmath>
#include <ctime>
using namespace std;
ifstream fi("Signum.inp");
ofstream fo ("Signum.out");
int x,a[100001],b[100001],n;
int64_t res;

int main()
{clock_t aa=clock();
 fi>>n>>x;b[0]=0;res=0;
 for(int i=1;i<=n;++i)
 {fi>>t; b[i]=b[i-1]^t;
  res+=(b[i]>=x);
 }
 for(int i=2;i<=n;++i)
 {
  for(int j=i;j<=n;++j)
  {b[j]^=b[i-1];
   res+=(b[j]>=x);
  }
 }
 fo<<res;
 clock_t bb=clock();
 fo<<"\nTime: "<<(double)(bb-aa)/1000<<" sec";
}

```

GT độ phức tạp  $O(n^2)$

### *Giải thuật độ phức tạp $O(n)$*

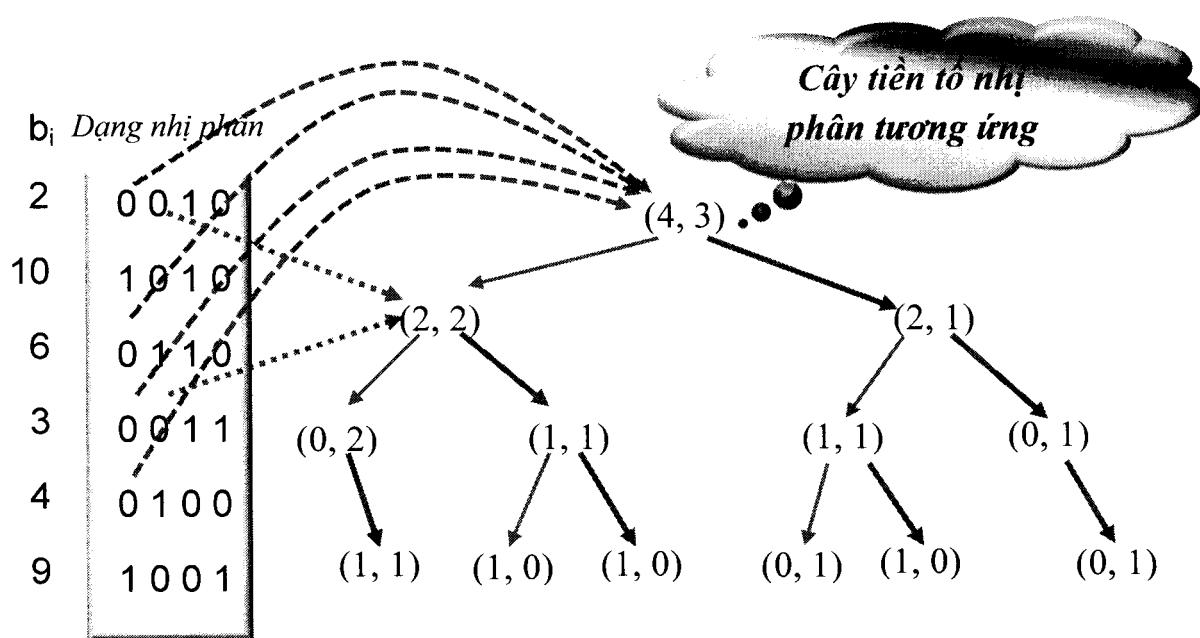
Để nâng cao hiệu quả của giải thuật ta cần phải:

- Vòng tránh việc liên tục cập nhật mảng B,
- Nhận dạng và tích lũy kết quả theo nhóm khoảng.

Các yêu cầu trên có thể thực hiện được dựa vào việc tổ chức cây nhị phân quản lý tiền tố các bit của  $b_i$ ,  $i = 1 \div n$ .

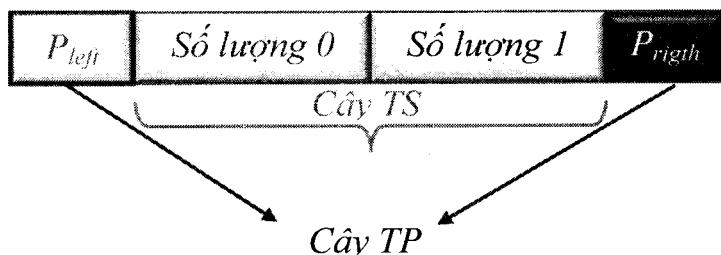
Về bản chất, đây là cây có các nút với trị 0 hoặc 1, các cạnh có trọng số là số lượng các  $b_i$  có bít tiếp theo là 0 hoặc 1. Với bài này, để tiện xử lý ta xem xét cách tổ chức cây ở góc độ hơi khác: cạnh rẽ sang trái tương ứng với giá trị 0, rẽ sang phải – giá trị 1. Mỗi nút của cây lưu trữ 2 giá trị: số lượng nhánh rẽ sang trái và số lượng nhánh rẽ sang phải ở mỗi nút.

Ví dụ, với  $n = 7$ ,  $x = 4$  và  $\mathbf{a} = (2, 8, 12, 5, 7, 13, 6)$  ta có mảng  $\mathbf{B}$  ban đầu và cây



tiền tố nhị phân tương ứng:

Để biểu diễn cây ở mỗi nút cần lưu trữ 4 giá trị:



Để tiện lập trình ta tổ chức 2 cấu trúc dữ liệu: cấu trúc TP lưu trữ các pointers xác định cấu trúc cây tiền tố và cấu trúc TS lưu trữ các số lượng tương ứng ở mỗi nút, các phần tử của cấu trúc thuộc loại `pair<int, int>`.

Vì không biết trước số lượng phần tử trong cây nên các cấu trúc này cần khai báo kiểu vector, bộ nhớ cần thiết để lưu trữ thông tin ở các nút sẽ được cấp phát động trong quá trình tạo lập cây. Chỉ cần quản lý các bít bắt đầu từ  $k$ , trong đó  $k$  – vị trí bít có nghĩa lớn nhất của  $x$  và  $a_1, a_2, \dots, a_n$ . Để dàng thấy rằng đường đi dài nhất từ gốc tới lá là  $k \leq 30$  và số nút lá trong cây không quá  $n$ .

Các công việc cần thực hiện:

- ➔ Nhập dữ liệu và tạo mảng  $\mathbf{B}$ ,
- ➔ Xác định  $k$ ,
- ➔ Khởi tạo cây ban đầu,

- ◆ Duyệt cây để tích lũy số khoảng thỏa mãn,
- ◆ Loại bỏ giá trị  $a_i$  không tham gia vào xử lý ở các bước tiếp theo,
- ◆ Cập nhật cây.

Nhập dữ liệu, tạo B và xác định k là các công việc đơn giản. Có thể không cần lưu trữ mảng A vì ở bước xử lý thứ i ( $i = 0 \div n-1$ ) có  $a_i = b_i$ . Tuy vậy, để thuận

```
fi>>n>>x;fi>>a[0];b[0]=a[0];
for(int i=1;i<n;++i)
{fi>>t; b[i]=b[i-1]^t;a[i]=t;}
```

*Nhập dữ liệu và tạo B*

```
rt=x;
for(int i=0;i<n;++i) if(rt<a[i]) rt=a[i];
k=0;
for(int i=30;i>=0;--i) if(rt&(1<<i)){k=i;break;}
```

*Xác định k*

tiên và dễ hiểu trong xử lý chương trình dưới đây vẫn tổ chức lưu trữ A.

### *Khởi tạo cây ban đầu*

```
m=0;ep=make_pair(-1,-1);es=make_pair(0,0);
tp.push_back(ep);ts.push_back(es);
```

Tạo 2 cây (**tp** và **ts**) rỗng:

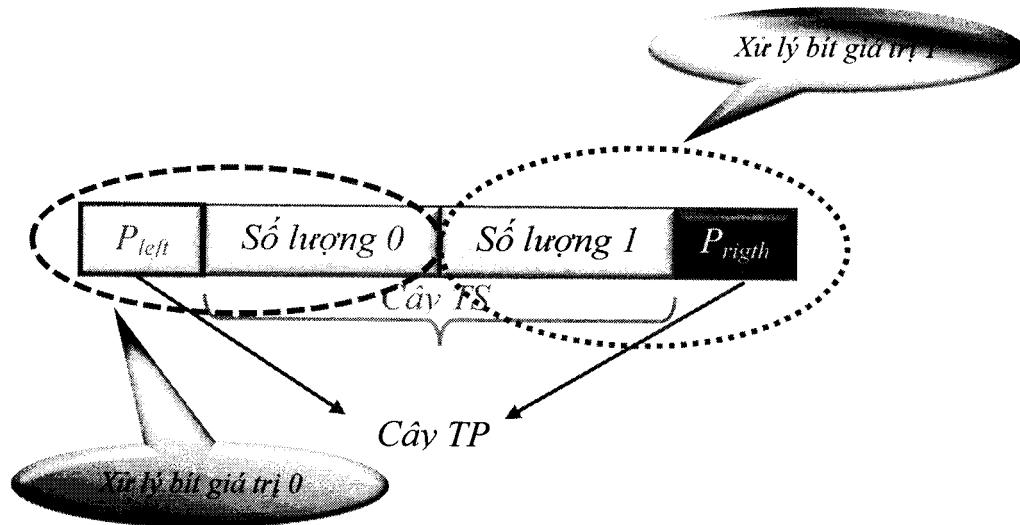
**m** – số thứ tự của nút trong cây.

Lần lượt nạp  $b_i$  vào cây,  $i = 0 \div n-1$ .

Với mỗi giá trị  $b_i$ :

Xuất phát từ nút gốc (**p** = 0),

Tách bit  $j$  ( $j = k \div 0$ ), ghi nhận số lượng vào cây  $TS$  và xác định nút tiếp theo từ



cây  $TP$ .

```

p=0;
for(int j=k;j>=0;--j)
    {tg=b[i]&(1<<j);
     if(tg){q=tp[p].second;++ts[p].second; if(q<0 && (j!=0))
           {new_el();tp[p].second=m;q=m;} p=q;}
      else
      {q=tp[p].first;++ts[p].first;
       if(q<0 && (j!=0)){new_el();tp[p].first=m; q=m;}p=q;}
    }
}

```

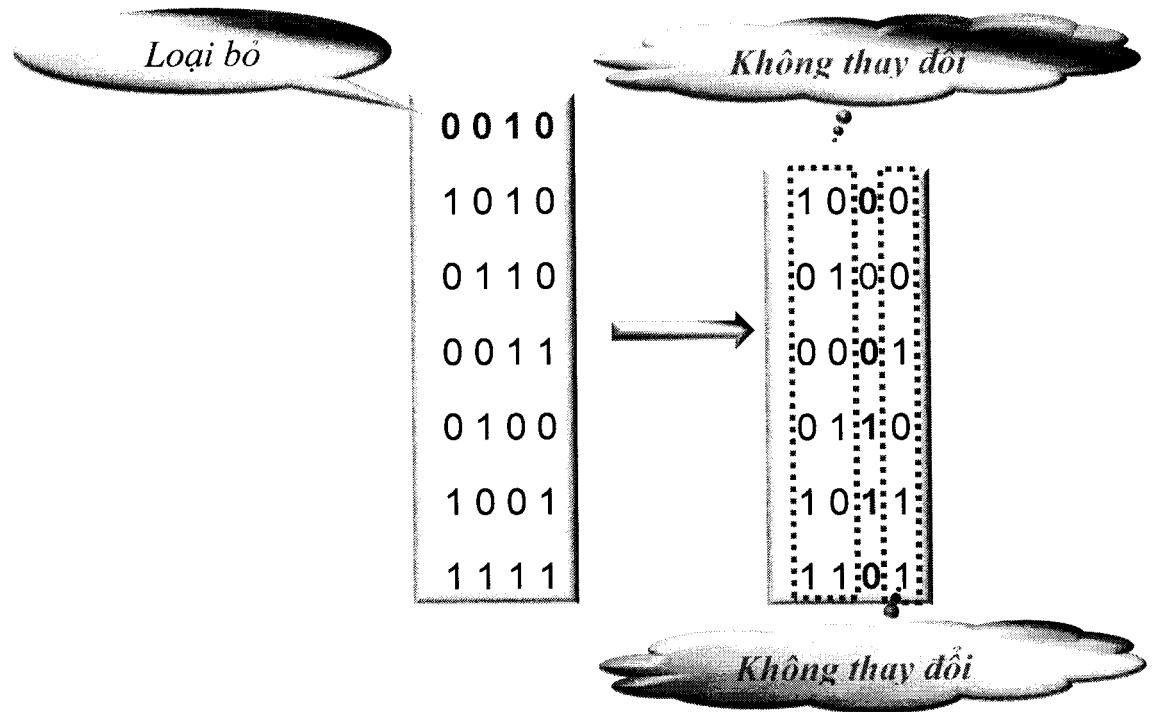
#### *Khởi tạo cây*

Nếu đây là nút lá (**pointer** = -1) thì cần tạo cặp nút mới (cho  $TP$  và  $TS$ ).

#### *Duyệt cây*

Duyệt cây là công việc phải thực hiện khi tích lũy số lượng khoảng thỏa mãn và xóa giá trị  $a_i$  khỏi cây. Công việc này gắn chặt với kỹ thuật tổ chức cập nhật cây.

Lưu ý rằng khi tạo bảng  $\mathbf{B}$  mới tiếp theo, nếu đó là bước  $i$  thì ta phải thực hiện phép xử lý  $b_j = b_j \text{ xor } a_{i-1}$  (tức là  $b[j] \hat{=} a[i]$ ),  $j = i \div n-1$ . Chỉ một phần bảng  $B$  thay đổi, đó là các cột  $u$  tương ứng với bít  $u$  của  $a_{i-1}$  bằng 1. Sự thay đổi



đó là đảo 1 thành 0 và ngược lại.

Các mốc nối khi đi qua cột thay đổi sẽ bị đảo từ trái sang phải và ngược lại. Việc đảo mốc nối có thể được ghi nhận một thanh ghi cờ **flg**. Ban đầu tất cả các bít của flg đều bằng 0. Khi một cột bị thay đổi ta chỉ cần đảo giá trị bít tương ứng của **flg** (từ 0 sang 1 hoặc ngược lại). Như vậy việc cập nhật bảng đơn thuần chỉ là  $\text{flg} \hat{=} a[i];$  !

Khi duyệt cây (để tích lũy khoảng thỏa mãn hay xóa một giá trị khỏi bảng  $\mathbf{B}$ ) cần tra cứu tới **flg** để xác định đúng đường đi. Đường đi được xác định bởi các bảng phương án (Decide Tables) sau:

Bit của $a_i$	Bit của flg	Tổng hợp	Xóa $a_i$
tx	tf	tc	
0	0	0	--ts[p].first; p=tp[p].first;
0	1	1	--ts[p].second; p=tp[p].second;
1	0	2	--ts[p].second; p=tp[p].second;
1	1	3	--ts[p].first; p=tp[p].first;

<i>Bit của x</i>	<i>Bit của flg</i>	<i>Tổng hợp</i>	<i>Tích lũy</i>
<i>tx</i>	<i>tf</i>	<i>tc</i>	
0	0	0	<i>r+=ts[p].second; p=tp[p].first;</i>
0	1	1	<i>r+=ts[p].first; p=tp[p].second;</i>
1	0	2	<i>p=tp[p].second;</i>
1	1	3	<i>p=tp[p].first;</i>

Chú ý:

- ➔ Chương trình sáng sửa, dễ hiểu và dễ hiệu chỉnh hơn nếu thực hiện rẽ nhánh bằng công cụ **switch** với biến **tc**,
- ➔ Lưu ý trường hợp trong cây không tồn tại đường đi đầy đủ theo **x** (khi tích lũy kết quả),
- ➔ Nên *tổ chức xử lý riêng nút cuối cùng* nếu tồn tại đường đi đầy đủ theo **x**,
- ➔ Biến lưu kết quả cần khai báo kiểu **int64\_t**,
- ➔ Chương trình sẽ đơn giản hơn nếu xử lý riêng bảng cuối cùng (chỉ chứa một phần tử **a[n-1]**).

## Chương trình theo giải thuật độ phức tạp $O(n)$

```
#include <iostream>
#include <cmath>
#include <algorithm>
#include <inttypes.h>
#include <vector>
#include <ctime>
using namespace std;
ifstream fi("Segnum.inp");
ofstream fo ("Segnum.out");
int x,a[100001],b[100001],n,m,k,t,flg=0;
int64_t res=0,rt;
vector<pair<int,int>>tp,ts;
pair<int,int> ep,es;

void chbi()
{
    m=0;ep=make_pair(-1,-1);es=make_pair(0,0);
    tp.push_back(ep);ts.push_back(es);
    rt=x;
    for(int i=0;i<n;++i) if(rt<a[i]) rt=a[i];
    k=0;
    for(int i=30;i>=0;--i) if(rt&(1<<i)){k=i;break;}
}

void new_el()
{tp.push_back(ep);
 ts.push_back(es);
 ++m;
}

void trace_t()
{int p=0,tx,tf,tc;
 rt=0;
 for(int i=k;i>0;--i)
 {tx=(x>>i)&1; tf=(flg>>i)&1; tc=tx<<1|tf;
 switch(tc)
 {
 case 0:rt+=ts[p].second;p=tp[p].first;break;
 case 1:rt+=ts[p].first;p=tp[p].second;break;
 case 2:p=tp[p].second;break;
 case 3:p=tp[p].first;break;
 }
 if(p<0)break;
 }
 if(p>=0)
 {
     tx=x&1; tf=flg&1;
     if(tx){if(tf)rt+=ts[p].first;else rt+=ts[p].second;}
     else rt+=(ts[p].first+ts[p].second);
 }
 rest+=rt;
}

void remove_t(int ii)
{int p=0,tx,tf,tc;
 for(int i=k;i>=0;--i)
 {tx=(a[ii]>>i)&1; tf=(flg>>i)&1; tc=tx<<1|tf;
 switch(tc)
 {
```

```

        case 0:--ts[p].first;p=tp[p].first;break;
        case 1:--ts[p].second;p=tp[p].second;break;
        case 2:--ts[p].second;p=tp[p].second;break;
        case 3:--ts[p].first;p=tp[p].first;break;
    }
}

void determ_t()
{int t0,t,t2,t3,p,q,tg;

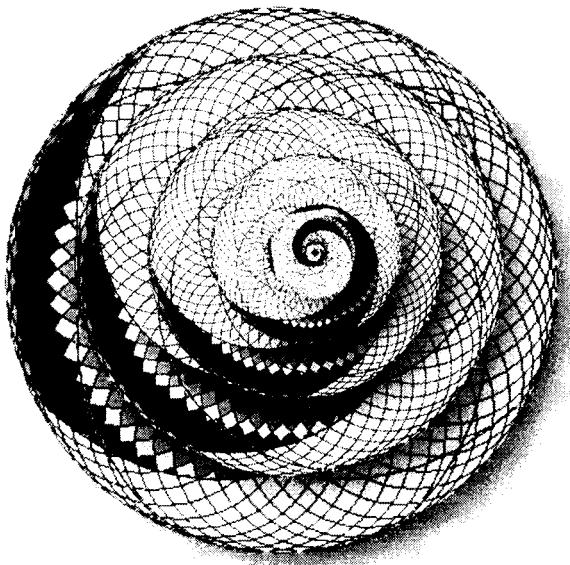
for(int i=0;i<n;++i)
{ p=0;
for(int j=k;j>=0;--j)
{tg=b[i]&(1<<j);
if(tg){q=tp[p].second;++ts[p].second; if(q<0 && (j!=0))
{new_el();tp[p].second=m;q=m;} p=q;}
else
{q=tp[p].first;++ts[p].first;
if(q<0 && (j!=0)){new_el();tp[p].first=m; q=m;}p=q;}
}
}
}

int main()
{clock_t aa=clock();
fi>>n>>x;fi>>a[0];b[0]=a[0];
for(int i=1;i<n;++i)
{fi>>t; b[i]=b[i-1]^t;a[i]=t;}
chbi();
determ_t();
for(int i=0;i<n-1;++i)
{trace_t();
remove_t(i);flg^=a[i];
}
if(x<=a[n-1])++res;
fo<<res<<endl;
clock_t bb=clock();
fo<<"\nTime: "<<(double)(bb-aa)/1000<<" sec";
}
}

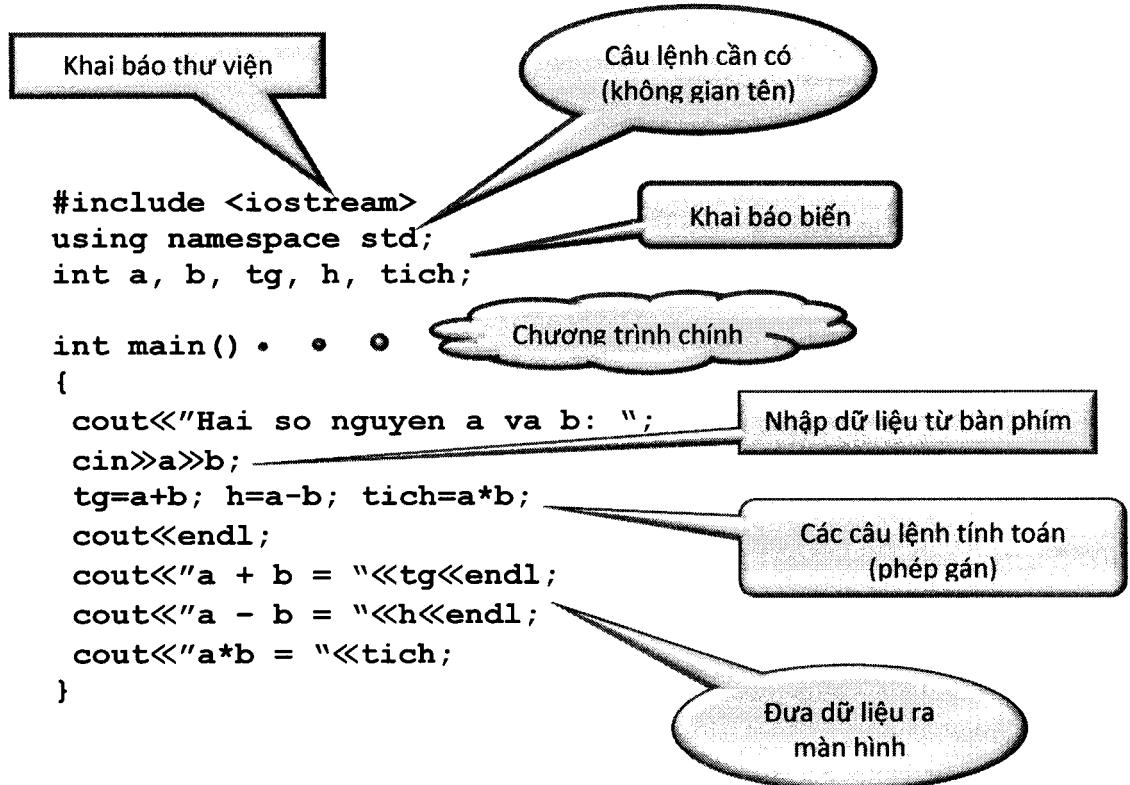
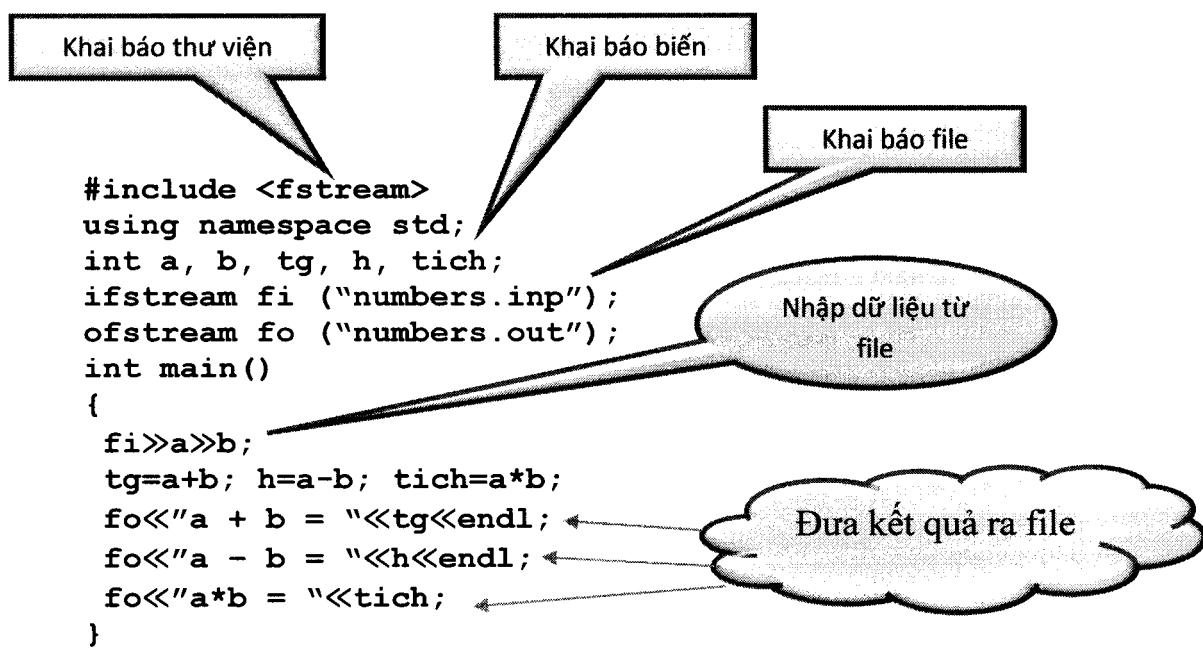
```

## Tài liệu

1. **Topcoder. Boba5551.** *Binary Indexed Trees.*
2. **Lưu Minh Nam** Báo cáo chuyên đề bồi dưỡng học sinh giỏi tin học các trường khối Đồng bằng và duyên hải Bắc bộ 2013. Trường PTTH Chuyên Hưng Yên.
3. Đề thi Olympic Tin học và ICPC các khu vực và quốc gia.







## Mảng số - Lệnh rẽ nhánh – Tổ chức chu trình

Cho mảng số nguyên  $A = (a_1, a_2, \dots, a_n)$ .

Hãy tính và đưa ra:

Tổng các phần tử của mảng  $A$ :  $t = \sum_{i=1}^n a_i$ ,

Phần tử lớn nhất:  $mx = \max\{a_i, i=1 \dots n\}$ ,

Phần tử nhỏ nhất:  $mn = \min\{a_i, i=1 \dots n\}$ .

**Dữ liệu:** nhập vào từ file văn bản DS.INP:

- ↳ Dòng đầu tiên chứa số nguyên  $n$  ( $1 \leq n \leq 1000$ ),
- ↳ Dòng thứ 2 chứa  $n$  số nguyên  $a_1, a_2, \dots, a_n$  ( $0 \leq a_i \leq 10^6$ ).

**Kết quả:** Đưa ra file văn bản DS.OUT các giá trị  $t$ ,  $mx$ ,  $mn$ , mỗi số đưa ra trên một dòng.

**Ví dụ:**

DS.INP	DS.OUT
<p>5</p> <p>4 6 2 3 5</p>	<p>20</p> <p>6</p> <p>4</p>

**Tên chương trình:** TS.CPP.

```
#include <iostream>
using namespace std;
int n,a[1001],t,mx,mn;
ifstream fi ("DS.INP");
ofstream fo ("DS.OUT");

int main()
{fi>>n;
 for(int i=1;i <=n;++i) fi>>a[i];
 t=0;
 for(int i=1;i<= n;++i) t+=a[i];
 mx=a[1]; mn=a[1];
 for(int i =1;i<=n;++i)
 {if(mx<a[i])mx=a[i];
 if(mn>a[i])mn=a[i];
 }
 fo<<t<<endl;
 fo<<mx<<endl;
 fo<<mn<<endl;
 }
```

## CHU TRÌNH và RẼ NHÁNH

### CHIA KẸO

Tên chương trình: CANDIES.???

Đội Trúc Xanh gồm 3 bạn An, Thùy và Minh về đầu trong cuộc thi về ca dao – tục ngữ Việt Nam. Cách trao giải của Ban tổ chức cũng khá độc đáo. Trên bàn bày một dãy  $n$  túi kẹo, trên túi kẹo thứ  $i$  có ghi số nguyên  $a_i$  – số lượng kẹo trong túi ( $a_i \geq 0$ ). Đội thắng cuộc được phép chọn các túi kẹo có số lượng chia hết cho 3.

Đội Trúc Xanh quyết định sẽ chọn hết tất cả các túi có kẹo và được phép lấy. Sau đó từ mỗi túi mỗi người ăn một chiếc kẹo. Phần kẹo còn lại được tập trung và chia đều để mỗi bạn mang về cho em ở nhà.



Hãy xác định, mỗi bạn đã ăn bao nhiêu cái kẹo và mang về nhà bao nhiêu cái.

**Dữ liệu:** Vào từ file văn bản CANDIES.INP:

- Dòng đầu tiên chứa số nguyên  $n$  ( $1 \leq n \leq 10^5$ ),
- Dòng thứ 2 chứa  $n$  số nguyên  $a_1, a_2, \dots, a_n$  ( $0 \leq a_i \leq 10^4$ ,  $i = 1 \div n$ ).

**Kết quả:** Đưa ra file văn bản CANDIES.OUT hai số nguyên: số lượng kẹo mỗi bạn đã ăn và số kẹo mỗi bạn mang về, mỗi số đưa ra trên một dòng.

**Ví dụ:**

CANDIES.INP
9
25 16 11 12 14 0 8 30 21

CANDIES.OUT
3
18

```
#include <fstream>
using namespace std;
int n,t,k,a[100001];
ifstream fi ("CANDIES.INP");
ofstream fo ("CANDIES.OUT");

int main()
{fi>>n;
for(int i=1;i<=n;++i) fi>>a[i];
t=0; k=0;
for(int i=1;i<=n;++i)
    if(a[i]>0 && a[i]%3==0){++t; k+=a[i];}
fo<<t<<endl;
k-=(3*t); k/=3;
fo<<k;
}
```

Kiểm tra chia hết  
cho 3

