MỘT SỐ KỸ THUẬT
TRIỂN KHAI PHƯƠNG PHÁP DUYỆT

LỜI NÓI ĐẦU

Hàng ngày, chúng ta thường dùng từ "*duyệt*" trong những tình huống, sự việc khác nhau như: Ban văn nghệ của Đoàn trường duyệt các tiết mục văn nghệ chào mừng 20 tháng 11; đầu năm học Ban giám hiệu duyệt kế hoạch công tác năm của trường và các tổ; Ban chỉ huy quân sự duyệt các phương án tác chiến; người thủ kho kiểm kê (duyệt) các vật liệu, máy móc trong kho,…

Trong các bài toán tin học, việc "*duyệt*" cũng xảy ra thường xuyên. Vậy "*duyệt*" là gi? Duyệt

có thể được coi là việc xem xét (để xử lý) các thành phần của một đại lượng nào đó hoặc các khả năng (trạng thái) có thể xảy ra của một hiện tượng trong một quá trình nào đó. Chẳng hạn, duyệt dãy số, duyệt các cấu hình tổ hợp,…

Duyệt để tìm nghiệm của bài toán nào đó là phương pháp tự nhiên đầu tiên mà người ta nghĩ đến. Các thuật toán duyệt luôn cho ta kết quả đứng nếu tìm thấy hoặc cho phép khẳng định bài toán vô nghiệm.

Trong tình hình hiện nay, với các bài toán có dữ liệu lớn thì thuật toán duyệt nhiều khi không đáp ứng được về thời gian thực hiện chương trình, nhưng nó vẫn vô cùng hữu ích. Đó là nó có thể giúp định hướng thuật toán, sau đó có thể cải tiến để đáp ứng yêu cầu về thời gian chạy chương trình. Mặt khác, nếu tổ chức duyệt tốt, có thể giúp khâu kiểm thử vì kết quả của thuật toán duyệt là đáng tin cậy.

Trong quá trình dạy chương trình chuyên tin, các thuật toán duyệt được dạy đầu tiên. Vì vậy, đến với Hội thảo khoa học lần này, chúng tôi xin tham góp một số vấn đề về phương pháp duyệt, chủ yếu là các kỹ thuật duyệt tuần tự để với anh chị em đồng nghiệp cùng tham khảo.

Xin trân trọng cảm ơn!

PHẦN NỘI DUNG CHUYÊN ĐỀ

---

1. Duyệt xuôi và duyệt ngược

Giai đoạn duyệt xuôi: từ điểm xuất phát, dựa trên những nhận xét hợp lý nào đó sẽ lần lượt tìm ra những điểm trung gian cần phải qua trước khi tới đích trên hành trình ngắn nhất. Trong giai đoạn tìm kiếm theo chiều thuận này, người ta lưu lại vết của hành trình vào một mảng *trace* mà trace[i]=j có ý nghĩa điểm j là điểm cần phải qua ngay trước điểm i trong hành trình ngắn nhất.

Giai đoạn duyệt ngược: Với mảng *trace,* từ trace[kt]=i_1 biết được trước khi đến điểm đích *kt* phải qua điểm i_1. Tương tự, từ trace[i_1]=i_2 biết được trước khi đến điểm i_1 phải qua điểm i_2…, cuối cùng, từ trace[i_k]=xp biết được trước khi đến điểm i_k phải qua điểm xuất phát *xp.* Suy ra hành trình ngắn nhất là $xp \to i_k \to \ldots \to i_2 \to i_1 \to kt$.

Ví dụ 1. Phân tích số thành tổng hai lập phương

Phân tích một số nguyên dương N thành tổng hai lập phương của hai số nguyên dương. Có bao nhiêu cách khác nhau?

Input: Số N nhập từ bàn phím

Output: Đưa kết quả ra màn hình, mỗi cách trên một dòng

Phân tích.

Bình thường có thể xét mọi cặp số nguyên dương i và j có giá trị tăng dần để chọn ra những cặp (i,j) mà $i^3+j^3=N$.

Tuy nhiên nhận thấy nếu i tăng thì j giảm nên ta có thể dùng phương pháp duyệt đồng thời ngược và xuôi (i: tăng dần, j: giảm dần) như sau: Ban đầu chọn j có giá trị cao nhất, còn i =1. Kiểm tra $k=i^3+j^3$, nếu k=N thì cặp (i,j) này là một kết quả, tiếp tục tăng i và giảm j, nếu k>N thì giảm j, nếu k<N thì tăng i. Công việc này được lặp cho đến khi i>j. Cách duyệt này hiệu quả hơn cách bình thường.

Văn bản chương trình.

```
uses   Crt;

var     i,j,count,N,k  : Integer;

BEGIN
    write('Nhập N = '); Readln(N);

    count:= 0;

    i       := 1;
```

```
    j       := 1;
    while (j*j*j+1<N) do Inc(j);
    repeat
        k :=i*i*i+j*j*j;
        if k=N then begin
             Inc(count);
             Write(i:4,j:4); Inc(i) ; Dec(j);
        end;
        if k < N then Inc(i);
        if k > N then Dec(j);
    until i >j;
    writeln('Co ',count,' Cach phan tich ' );
    readln
END.
```

Ví dụ 2. Thời điểm hai kim đồng hồ trùng nhau

Các kim đồng hồ chuyển động với các vận tốc góc không đổi, thời điểm hiện thời là *h* giờ *m* phút. Tìm thời gian sớm nhất để hai kim giờ và phút trùng nhau (tính theo số nguyên phút).

Input. Hai số nguyên *h* và *m* (nhập từ bàn phím)

Output. Số nguyên phút thể hiện thời gian sớm nhất để hai kim giờ và phút trùng nhau (hiện trên màn hình)

Phân tích

Để thuận tiện chúng ta kí hiệu thời điểm h giờ, m phút là **h:m**. Rõ ràng hai kim giờ và phút trùng nhau tại 0:0, sau đó thời gian sớm nhất để hai kim lại trùng nhau là t_0 giờ thì kim giờ quay được một góc là $\alpha = \frac{t_0}{12}$ vòng, kim phút quay được góc là $\beta = 1 + \frac{t_0}{12}$ vòng. Do kim phút chạy nhanh hơn kim giờ 12 lần nên có: $\beta = 12\alpha$ suy ra: $t_0 = \frac{12}{11}$ (giờ). Vậy kim phút đã chạy được $\frac{12 \times 60}{11}$ (phút)= $\frac{720}{11}$ (phút). Vậy cứ sau $\frac{720}{11}$ phút hai kim lại trùng nhau do tốc độ quay của các kim không thay đổi.

Trong các thời điểm trùng nhau: $\frac{720}{11}$, $2\times\frac{720}{11}$, $3\times\frac{720}{11}$,...ta cần chọn ra thời điểm sớm nhất sau thời điểm h:m. Nghĩa là cần tìm số k nguyên nhỏ nhất sao cho $k\times\frac{720}{11}\geq 60\times$ h+m . Khi đó khoảng thời gian sớm nhất để hai kim gặp nhau là: $\Delta = k\times\frac{720}{11}$ - (60×h+m)= $\frac{720\times k-11\times(60\times h+m)}{11}$

Đặt **t** =11×(60×h+m). Bình thường, để tìm Δ ta ***tăng dần số nguyên k*** cho đến khi 720×k bắt đầu lớn hơn 11×(60×h+m) như chương trình sau:

Văn bản chương trình 1:

```
uses  crt;
var   h, m, t, k : longint;
BEGIN
    write('Nhap thoi diem h:m '); readln(h,m);
    t := 11*(60*h+m);
    k := 0;
    while (k*720<t) do
        k := k+1;
    writeln('Dap so : ',(k*720-t) div 11);
    readln;
END.
```

Tuy nhiên, chúng ta có thể viết lại chương trình trên dưới dạng khác bằng cách làm ngược lại: biến đổi thực hiện giảm dần t từng lượng 720 cho đến khi t≤720 khi đó $\frac{720-t}{11}$ là đáp số

Văn bản chương trình 2

```
uses  crt;
var   h, m, t : longint;
BEGIN
    write('Nhap thoi diem h:m '); readln(h,m);
    t := 11*(60*h+m);
    t := t mod 720;
    writeln('Dap so : ',(720-t) div 11);
    readln;
END.
```

Ví dụ 3. Trò chơi với dãy số - Seqgame **(Thi HSG 2008)**

Hai bạn học sinh trong lúc nhàn rỗi nghĩ ra trò chơi sau đây. Mỗi bạn chọn trước một dãy số gồm N số nguyên. Giả sử dãy số mà bạn thứ nhất chọn là:

$b_1, b_2, ..., b_n$

còn dãy số mà bạn thứ hai chọn là:

$c_1, c_2, ..., c_n$

Mỗi lượt chơi mỗi bạn đưa ra một số hạng trong dãy số của mình. Nếu bạn thứ nhất đưa ra số hạng b_i ($1 \le i \le N$), còn bạn thứ hai đưa ra số hạng c_j ($1 \le j \le N$) thì giá của lượt chơi đó sẽ là $|b_i+c_j|$.

Ví dụ: Giả sử dãy số bạn thứ nhất chọn là 1, -2; còn dãy số mà bạn thứ hai chọn là 2, 3. Khi đó các khả năng có thể của một lượt chơi là (1,2), (1,3), (-2,2), (-2,3). Như vậy, giá nhỏ nhất của một lượt chơi trong số các lượt chơi có thể là 0 tương ứng với giá của lượt chơi (-2,2).

Yêu cầu

Hãy xác định giá nhỏ nhất của một lượt chơi trong số các lượt chơi có thể.

Dữ liệu

- Dòng đầu tiên chứa số nguyên dương N ($N \le 10^5$)
- Dòng thứ hai chứa dãy số nguyên $b_1, b_2, ..., b_n$ ($|b_i| \le 10^9$,i=1, 2, …, N)
- Dòng thứ ba chứa dãy số nguyên $c_1, c_2, ..., c_n$ ($|c_i| \le 10^9$,i=1, 2, …, N)

Hai số liên tiếp trong một dòng được ghi cách nhau bởi dấu cách.

Kết quả

Ghi ra giá nhỏ nhất tìm được.

Ràng buộc

60% số test ứng với 60% số điểm của bài có $1 \le N \le 1000$

Ví dụ

SEQGAME.IN	SEQGAME.OUT
2 1 -2 2 3	0

Phân tích

+ Sắp xếp tăng từng dãy số

+ Dùng 2 biến chạy xuôi và ngược là *i* và *j*. Biến *i* dùng duyệt xuôi mảng b(N) bắt đầu từ vị trí 1, biến *j* dùng duyệt ngược mảng c(N) bắt đầu từ vị trí N.

+ Vòng lặp thực hiện trong khi (i<=N) và (j>=1):

- Tính $v=b_i+c_j$
- Nếu v>=0 thì giảm j (để |v| giảm đi)

Ngược lại nếu v<0 thì tăng i (vì số âm khi được tăng lên thì trị tuyệt đối của nó giảm đi.

Văn bản hương trình.

```
const fi     = 'seqgame.in';
      fo     = 'seqgame.out';
```

```
      max    = 100000;
type  tarray = array[1..max] of Integer;
var   b, c   : tarray;
      n, min : integer;
procedure read_input;
var   f        : text;
      i, temp  : Integer;
begin
  assign(f, fi); reset(f);
  readln(f, n);
  for i := 1 to n do read(f, b[i]);
  readln(f);
  for i := 1 to n do begin
    read(f, c[i]);
  end;
  close(f);
end;
procedure quicksort(var k: TArray; l, r: Integer);
var i, j: integer;
    temp, key: Integer;
begin
  if l >= r then exit;
  key := k[(l + r) div 2];
  i := l; j := r;
  repeat
    while k[i] < key do inc(i);
    while k[j] > key do dec(j);
    if i <= j then begin
      if i < j then begin
        temp := k[i]; k[i] := k[j]; k[j] := temp;
      end;
      inc(i); dec(j);
    end;
  until i > j;
  QuickSort(k, l, j);
  QuickSort(k, i, r);
end;
procedure solve;
var v, i, j: integer;
begin
  min := maxInt;
  i:=1; j:= n;
  while (i<=n) and (j>1) do begin
    v := b[i]+c[j];
    if v>=0 then dec(j)
    else if v<0 then inc(i);
    if abs(b[i]+c[j])<min then min:=abs(b[i]+c[j]);
  end;
end;
procedure write_out;
var f: text;
begin
  assign(f, fo); rewrite(f);
  write(f, min);
  close(f);
end;
```

```
BEGIN
  read_input;
  quicksort(b, 1, n);
  quicksort(c, 1, n);
  solve;
  write_out;
END.
```

2. Sử dụng biến "lính canh"

Mọi quốc gia đều có biên giới và lính biên phòng ngày đêm canh gác. Vượt qua biên giới là sang nước láng giềng.

Tin học cũng "học" đời thường ở cách bố trí lính canh này. Để quản lý phạm vi cần duyệt, dùng biến canh vị trí đầu và biến canh vị trí cuối của phạm vi này.

Tổ chức hàng đợi (Queue) bằng mảng một chiều là một minh họa điển hình về dùng biến lính canh. Người ta thường dùng biến *first* chỉ vào vị trí đầu hàng đợi, biến *last* chỉ vào vị trí cuối hàng đợi. Các phần tử trong hàng đợi chờ xử lý chỉ nằm từ vị trí *first* đến vị trí last. Biến *first* còn làm nhiệm vụ định ra vị trí của các phần tử lấy ra khỏi hàng đợi để xử lý; biến *last* còn làm nhiệm vụ xác định được vị trí phần tử nạp vào hàng đợi để chờ xử lý.

Ví dụ 1. Dãy con có tổng chia hết cho n

Cho số nguyên dương n và dãy A gồm n số nguyên dương a_1, a_2, …, a_n, có thể tạo ra một dãy con gồm các phần tử liên tiếp của A mà tổng các phần tử chia hết cho n hay không. Nếu có hãy viết ra dãy con này.

Dữ liệu vào từ tệp input.txt, dòng đầu tiên của mỗi test là số n ($1 \le n \le 10^4$), dòng thứ hai là các số nguyên dương a_1, a_2, …, a_n cách nhau dấu trống.

Kết quả ghi ra tệp output.txt dãy con tìm được hoặc thông báo "No" nếu không tìm được dãy con nào thỏa mãn. Ví dụ:

Input.txt	Output.txt
1 1 3 1 4 2	1 1 2

Phân tích

Giả sử các tổng $s_{1=} a_1$, $s_2= a_1+a_2$, $s_{3=}a_1+a_2+a_3$, …, $s_n=a_1+a_2+…+a_n$ khi chia cho n có các dư tương ứng là r_1, r_2, …, r_n. Nếu trong chúng có $r_i=0$ thì dãy $\{a_1, a_2, …, a_i\}$ là dãy con cần tìm. Do có n số r_1, r_2, …, r_n mà chỉ thuộc n-1 loại dư (từ 0 đến n-1) nên tồn tại r_i và r_j bằng nhau (i<j) (theo nguyên lý Đi-rich-lê). Suy ra $s_j-s_i=a_{i+1}+a_{i+2}+…+a_j$ chia hết cho n vậy $\{a_{i+1},a_{i+2},…,a_j\}$ là dãy con cần tìm. Bài toán luôn luôn có nghiệm. Khi lập trình, ta dùng mảng S để lưu lại dãy

{s_1, s_2, …, s_n} và mảng B với ý nghĩa B[r] là vị trí i mà S_i chia cho n có dư r. Đồng thời chúng ta dùng 2 biến "lính canh": *bs* và *es* để đánh dấu vị trí i+1 và j khi gặp s_j-s_i chia hết cho n, đó là vị trí đầu và cuối của dãy con cần tìm.

Văn bản chương trình

```
Program Day_con_co_tong_chia_het_cho_n;
const  maxn = 10000;
       fi   = 'input.txt';
       fo   = 'output.txt';
var    f, g : text;
       a    : array[1..maxn] of longint;
       b    : array[0..maxn] of longint;
       s, r : longint;
       n, i, bs, es  : longint;
begin
    assign(f,fi); reset(f);
    assign(g,fo); rewrite(g);
    while true do begin
         if eof(f) then break;
         readln(f,n);
         if n=0 then break;
         for i:=1 to n do read(f,a[i]);
         s := 0;
         fillchar(b, sizeof(b),0);
         for i:=1 to n do begin
             s := s + a[i];
             r := s mod n;
             if r=0 then begin
               bs := 1; es := i;
                break;
             end
             else  {r <>0}
             if b[r]<>0 then begin
                  bs := b[r] +1; es := i;
                  break;
             end;
             b[r]:=i;
         end;
         for i:=bs to es do write(g,a[i]:6);
         writeln(g);
    end;
    close(f); close(g);
end.
```

Ví dụ 2. Dãy con lớn nhất

Cho dãy số A gồm N số nguyên khác 0. Tìm một dãy con gồm các phần tử liên tiếp của A mà tổng các số trong dãy con là lớn nhất.

Dữ liệu vào từ tệp input.txt, trong đó ghi dãy số A, kết thúc bởi số 0 (không thuộc dãy A). Bảo đảm rằng dãy A không rỗng và tổng của số lượng bất kỳ các số của A có thể biểu diễn là số nguyên kiểu longint.

Kết quả ra ghi vào tệp output.txt chỉ số của số đầu và số cuối của dãy con và tổng các số của dãy con. Ví dụ

Input.txt	Output.txt	Input.txt	Output.txt
-2 -1 0	2 2 -1	1 2 -3 3 0	1 2 3

Văn bản chương trình

```
uses  crt;
const fi = 'input.txt';
      fo = 'output.txt';
      maxn = 10000;
var   a    : longint;
      maxS,dau,cuoi:longint
      S, d, c,P         : longint;
      f, g : text;
begin
    assign(f,fi); reset(f);
    assign(g,fo); rewrite(g);
    read(f,a); p := 1;
    maxS := a; dau := p; cuoi := p;
        while a<0 do begin
      if a>maxS then begin
          maxS := a; dau := p; cuoi := p;
      end;
      read(f,a);
      if a=0 then exit;
      inc(p);
    end;
    S := a; d := p; c := p;
    maxS := a; dau := p; cuoi := p;
    while true do begin
      read(f,a);
      if a=0 then break else inc(p);
      if S>=0 then begin
         S := S + a;
         if a>0 then begin
            c := p;
            if S>maxS then begin
               dau := d; cuoi := c; maxS := S;
            end;
         end;
      end
      else {S<0}
      if a>0 then begin
         S := a; d := p; c := p;
         if S>maxS then begin
             dau := d;  cuoi := c;  MaxS := S;
         end;
      end;
    end;
    close(f);
    write(g,dau,' ',cuoi,' ',maxS);
    close(g);
end.
```

3. Cộng dồn

Trong quá trình tính toán, nếu biết tổ chức dữ liệu có tính toán kế thừa thì số lượng phép tính giảm đi rõ rệt.

Ví dụ 1. Tổng k số nguyên liên tiếp lớn nhất

Cho một mảng A gồm *N* số nguyên và số nguyên dương *k*. Hãy tìm tổng k số nguyên liên tiếp của mảng A lớn nhất.

Input: nhập từ file dayso.inp, dòng đầu là hai số n, k; Dòng dau là dãy A

Output: Đưa ra file dayso.out ba số nguyên i,j,T, trong đó i là chỉ số đầu, j là chỉ số cuối của đoạn k phần tử, T là tổng lớn nhất.

Phân tích

Bình thường ta phải tính tổng tất cả các đoạn con có *k* phần tử liên tiếp, rồi so sánh các tổng này để tìm ra tổng lớn nhất. Công việc này đòi hỏi *(N-k)* $\times$*(k-1)* phép cộng và tổ hợp C_{N-k}^{2} phép so sánh hai tổng. Nếu N và k tương đối lớn thì số lượng phép tính này rất lớn. Độ phức tạp tính toán trung bình cỡ O(*N*$\times$*k*).

Để giải bài toán này, còn cách sau đây có độ phức tạp tính toán trung bình là O(*N*): Ta tạo các tổng $S_i = A_1+A_2+\ldots+A_i = S_{i-1}+A_i$. Sau đó muốn tìm các tổng *k* phần tử liên tiếp bắt đầu từ *j* ta sử dụng công thức:

$A_j+A_{j+1}+\ldots+A_{j+k-1}=(A_1+A_2+\ldots+A_{j+k-1})-(A_1+A_2+\ldots+A_{j-1})=S_{j+k-1}-S_{j-1,}$ với $1 \le j \le N-k+1$.

Văn bản chương trình.

```
Program Tong_K_so_nguyên;

const fi  = 'dayso.in';
      fo  = ‘dayso.out’;
      nmax=10000;
var   n, k, be, en : integer;
      max : longint;
      a   : array[1..nmax] of integer;
      s   : array[0..nmax] of longint;
Procedure doc_input;
var f : text; i : integer;
begin
  assign(f,fi); reset(f);
  read(f,n,k);
  for i:=1 to n do read(f,a[i]);
  close(f);
  s[0] := 0;
  for i:=1 to n do s[i] := s[i-1] + a[i];
end;
var j : integer;
BEGIN
```

```
  doc_input;
  max := -maxlongint;
  for j:= 1 to n-k+1 do
  if max<s[j+k-1]-s[j-1] then begin
     max := s[j+k-1]-s[j-1];
     be :=j;
     en := j+k-1;
  end;
  write(be,' ',en,' ',max);
  readln;
END.
```

Ví dụ 2. Hình chữ nhật lớn nhất

Cho bảng hai chiều M dòng, N cột gồm M×N ô vuông. Mỗi ô vuông chứa một số nguyên. Tìm trong bảng một hình chữ nhật có tổng các số trên các ô là lớn nhất (hình chữ nhật này được gọi là hình chữ nhật lớn nhất trong các hình chữ nhật thuộc bảng). Một hình chữ nhật có thể gồm một số ô 1×1 kề nhau hoặc chiếm toàn bộ bảng.

Ví dụ trong bảng sau, hình chữ nhật lớn nhất nằm ở góc trái-dưới của bảng:

0	-2	-7	0
9	2	-6	2
-4	1	-4	1
-1	8	0	-2

và có tổng bằng15.

Dữ liệu vào trong file HCN.IN chứa bảng hai chiều M×N số nguyên. Dòng đầu tiên chứa hai số nguyên dương M và N là kích thước dòng và cột của bảng. Tiếp theo là M×N số nguyên cách nhau bởi các dấu trắng (dấu xuống dòng hoặc dấu trống). M×N số nguyên này theo thứ tự lần lượt là các số thuộc các ô của bảng tính theo hàng từ trên xuống dưới và theo cột từ trái qua phải. M và N không quá 100. Các số nguyên trong bảng thuộc đoạn [-127,127].

Kết quả ra file HCN.OUT: Dòng đầu là tổng các số thuộc các ô của hình chữ nhật lớn nhất. Dòng thứ hai là 4 số nguyên thể hiện toạ độ của ô ở góc trái-trên và toạ độ ô ở góc phải-dưới của hình chữ nhật lớn nhất (toạ độ dòng trước, toạ độ cột sau).

HCN.IN

```
4 4
0 -2 -7  0 9  2 -6  2
-4  1 -4  1 -1
```

```
8  0 -2
```

HCN.OUT

```
15
2 1 4 2
```

Phân tích

Cần phải duyệt tất cả các hình chữ nhật nằm trong bảng chữ nhật đã cho. Đó là các hình chữ nhật (d1, c1, d2, c2) có tọa độ góc trái trên là (d1, c1) và tọa độ góc phải dưới là (d2, c2), sử dụng 4 vòng lặp lồng nhau sẽ duyệt được hết các hình chữ nhật trong bảng:

For d_1:=1 to M do

For c_1:=1 to N do

For d_2:=d_1 to M do

For c_2:=c_1 to N do

Mỗi lần tạo một hình chữ nhật, phải tính tổng các số trong hình chữ nhật đó. Như vậy độ phức tạp tính toán cỡ $O(N^3.M^3)$. Chương trình sẽ chạy rất chậm khi N và M cỡ 100. Vì vậy cần biết kỹ thuật ghi lưu kết quả thích hợp để tính tổng các số trong hình chữ nhật đó cỡ O(1), làm cho độ phức tạp tính toán chỉ còn cỡ $O(N^2.M^2)$.

Ta có thể tiến hành như sau:

Trước hết từ bảng hai chiều A[1..M,1..N] chúng ta tạo ra bảng B[1..M,1..N] sao cho mỗi ô B[i,j] là tổng các số nằm trong hình chữ nhật của bảng A, có góc trái-trên là ô (1,1) và góc phải dưới là ô (i,j).

Dùng bảng B để tính tổng các số nằm trong hình chữ nhật (d1, c1, d2, c2) của bảng A bằng công thức:

Tong(d1, c1, d2, c2)=B[d2,c2]-B[d1-1,c2]-B[d2,c1-1]+B[d1-1,c1-1]

Văn bản chương trình.

```
Program Hinh_chu_nhat;
const  fi    = 'hcn.in';
       fo    = 'hcn.out';
       max   = 101;
var    m,n   : integer;
       a     : array[0..max,0..max] of longint;
       sumMax: longint;
       ld1,lc1,ld2,lc2 : integer;
Procedure read_input;
var f   : text;
    i,j : integer;
begin
    assign(f,fi);    reset(f);
    readln(f,m,n);
```

```
    for i:=1 to m do
    for j:=1 to n do read(f,a[i,j]);
    close(f);
end;
Procedure Tao_B;
var i,j : integer;
begin
    for j:=2 to n do  a[1,j] := a[1,j] + a[1,j-1];
    for i:=2 to m do
    begin
        for j:= 1 to n do
            a[i,j] := a[i,j] + a[i,j-1];
        for j:=1 to n do
            a[i,j] := a[i,j] + a[i-1,j];
    end;
end;
function Tong(d1,c1,d2,c2 : integer) : longint;
var p : longint;
begin
    p := a[d2,c2] - a[d1-1,c2] - a[d2,c1-1] + a[d1-1,c1-1];
    Tong := p;
end;
Procedure find;
var  dong1,cot1,dong2,cot2 : integer; sum : longint;
begin
    sumMax := -maxlongint;
    for dong1:=1 to m do
    for cot1:=1 to n do
    for dong2:=dong1 to m do
    for cot2:=cot1 to n do
    begin
        sum := tong(dong1,cot1,dong2,cot2);
        if sum > sumMax then
        begin
            sumMax := sum;
            ld1 := dong1;
            lc1 := cot1;
            ld2 := dong2;
            lc2 := cot2;
        end;
    end;
end;
Procedure write_output;
var f : text;
begin
    assign(f,fo);    rewrite(f);
```

```
    writeln(f,sumMax);
    write(f,ld1,' ',lc1,' ',ld2,' ',lc2);
    close(f);
end;
BEGIN
    read_input;
    Tao_B;
    find;
    write_output;
END.
```

4. Chơi Ô ăn quan.

Trò chơi "Ô ăn quan" là trò chơi dân gian thú vị gắn với tuổi thơ của nhiều người. Trò chơi thật đơn giản nhưng chứa nhiều yếu tố bất ngờ: bạn bốc một ô sỏi của mình rồi rải đều trên các ô (mỗi ô 1 viên) theo một chiều nào đó, khi hết sỏi nếu gặp ô tiếp theo có sỏi thì lại bốc sỏi của ô này và rải tiếp, nếu cách một ô mới gặp ô có sỏi thì được "ăn" số sỏi ở ô có sỏi này…

Một số bài toán tin học cũng học cách rải sỏi của trò chơi này để phân bố các giá trị của các biến đếm vào các ô nhớ.

Ví dụ 1. Tạo các số Hexa

Hãy tạo tất cả các số nguyên tăng dần từ 1 đến 10000 trong hệ đếm Hexa (là hệ đếm có cơ số 16).

Phân tích

Mỗi số x<1000 trong hệ đếm Hecxa có thể biểu diễn dưới dạng: $x=x_1.16^3 + x_2.16^2 + x_3.16 + x_4$ (mà $0 \leq x_i<16$, i=1, 2, 3, 4). Để hình thành các số x_1, x_2, x_3, x_4 chúng ta tạm tạo ra 4 ô là A_1, A_2, A_3 và A_4 theo thứ tự xếp từ trái qua phải. Sau đó chúng ta lấy 10000 viên sỏi rải vào các ô này theo nguyên tắc: Mỗi lần rải 1 viên cho đầy ô A_4 (nghĩa là đủ 16 viên, viên thứ nhất ứng với số 0, viên cuối ứng với số 15). Khi A_4 đầy, thì "ăn" hết quân của A_4, và rải một quân vào A_3, tiếp theo lại quay về rải cho đầy A_4, rồi lại "ăn" hết quân của A_4 và rải thêm một quân vào A_3, quá trình này lặp đến khi A_3 đầy (đủ 16 quân) thì "ăn" hết quân ở A_3, đồng thời rải một quân vào A_2, lại quay về rải quân bắt đầu từ ở A_4… quá trình tiếp tục làm đầy dần A_2…, cứ thế lặp cho đến khi hết 10000 viên sỏi. Tập hợp các số lượng sỏi ở các ô A_1, A_2, A_3 và A_4 sau mỗi lần rải 1 viên sỏi là các số x_1, x_2, x_3, x_4 trong biểu diễn một số nguyên x dưới dạng Hexa. Khi x tăng dần từ 1 đến 10000 ta biểu diễn được các số nguyên từ 1 đến 10000 trong hệ Hexa. Chỉ lưu ý một điều là x_i=10 biểu diễn bằng 'A', x_i=11 biểu diễn bằng 'B', … , x_i=15 biểu diễn bằng 'F',

Văn bản chương trình

```
Program Tao_so_Hexa;

const fo = 'hexa.out';
      L  = 4;
      M  = 15;
      KT : array[0..15] of char =
('0','1','2','3','4','5','6','7','8','9','A','B','C','D','E','F');
var   a    : array[1..L] of integer;
      i    : integer;
      g    : text;
      OK   : boolean;
Procedure hien;
var i : integer; value : longint;
begin
   for i:=1 to L do write(g,KT[a[i]]);
   value := 0;
   for i:= 1 to L do value := a[i] + value*(M+1);
   writeln(g,'   = ',value);
   if value = 10000 then Ok := true;
end;
Procedure tao(L : byte);
var i,j : integer;
begin
   inc(a[L]);
   if a[L]>M then begin
      a[L] := 0;
      if L>1 then tao(L-1)
      else exit;
   end;
end;

BEGIN
   for i:=1 to L do a[i]:= 0;
   assign(g,fo); rewrite(g);
   OK := false;
   repeat
      tao(L);
      hien;
   until OK;
   close(g);
END.
```

Ví dụ 2. Thiết bị ACM

ACM là một thiết bị gồm L máy đo các thông số không phụ thuộc vào nhau. Mỗi máy đo ghi nhận một số nguyên trong đoạn từ 1 đến M. Không phải tất cả các thông số đó đều được sử dụng để tính toán. Một vài số trong chúng dùng để kiểm tra lỗi. Các giá trị trong máy đo được chọn một cách máy móc sao cho tổng giá trị trong tất cả các máy đo luôn chia hết cho số K cho trước. Giá trị trong các máy đo của ACM hoàn toàn xác định trạng thái của thiết bị. Những người sáng chế đã thực hiện hiệu quả hơn việc đưa thông tin về trạng thái của thiết bị: Cụ thể là, thay vì ghi dãy số gồm tất cả các số trong các máy đo, người ta đưa ra một số duy nhất gọi là mã trạng thái. Mã trạng thái phải hoàn toàn mô tả được trạng thái của thiết bị nên các nhà sáng chế đã quyết định tính toán như sau:

Trạng thái của thiết bị thể hiện bởi một dãy số có chiều dài L (là số lượng các số trong các máy đo) thỏa mãn các điều kiện ở trên. Do vậy, để biểu diễn mã trạng thái người ta chọn chỉ số của dãy số trạng thái hiện tại của thiết bị theo thứ tự từ điển (tăng dần) trong danh sách tất cả các trạng thái có thể có (trạng thái đầu tiên có chỉ số bằng 0).

Nhưng bây giờ lại phát sinh thêm một vấn đề:

Gỉa sử đã biết một mã trạng thái, ta cần phải xác định các giá trị ghi nhận được ở các máy đo.

Để giải quyết vấn đề trên cần có sự giúp đỡ của bạn.

Dữ liệu vào cho trong tệp văn bản ACM.IN: dòng đầu tiên có 3 số nguyên là L, M và K ($1 \le L \le 100$; $2 \le M \le 50$; $1 \le K \le 50$). Trên dòng thứ hai là số nguyên N (là mã trạng thái)

Kết quả ra ghi vào tệp văn bản ACM.OUT các trạng thái của các máy đo ứng với mã N, tức là L số nguyên đứng cách nhau bởi các khoảng trắng. Ví dụ:

```
ACM.IN  ACM.OUT

3 10 4  9 6 1

213
```

Phân tích

Sử dụng kỹ thuật rải sỏi.

Văn bản chương trình.

```
Program Thietbi_ACM;
const fi = 'acm.in';
      fo = 'acm.out';
var   status : array[1..101] of byte;
      k, L, m : byte;
      n       : longint;
      index   : longint;
      i       : byte;
Procedure readfile;
var f : text;
begin
  assign(f,fi); reset(f);
  readln(f,L,M,k); readln(f,n);
  close(f);
end;
function test : boolean;
var i : integer; s : longint;
begin
  s := 0;
  for i:=1 to L do s := s + status[i];
  if s mod k=0 then
     test:=true
  else test:=false;
end;
Procedure raigianh(i: byte);
begin
  inc(status[i]);
```

```
  if status[i]>M then begin
     status[i] := 1;
     if i>1 then raigianh(i-1)
     else exit;
  end;
end;
Procedure writefile;
var f : text;i : byte;
begin
  assign(f,fo); rewrite(f);
  for i:=1 to L do write(f,status[i],' ');
  close(f); halt;
end;
BEGIN
  readfile;
  index := 0;
  for i:=1 to L do status[i]:=1;
  repeat
    if test then begin
       if index=n then writefile;
       inc(index);
    end;
    raigianh(L);
  until count>n;
END.
```

5. Kỹ thuật đánh dấu

Để không xét lại những phần tử đã duyệt, người ta thường đánh dấu các phần tử đã duyệt.

Ví dụ khi thực hiện thuật toán sàng **Eratosthenes,** để tìm các số nguyên tố không vượt quá số nguyên dương N, chúng ta đánh dấu trên trục số các bội của 2, rồi các bội của 3 chưa đánh dấu, các bội của 5 chưa đánh dấu,…. Các số còn lại chưa bị đánh dấu chính là các số nguyên tố.

Ví dụ 1. Chia kẹo

Có n gói kẹo ($n \leq 200$), các gói kẹo được đánh số từ 1 đến n. Gói kẹo thứ i có A_i cái kẹo ($1 \leq i \leq n$, $0 < A_i \leq 200$). Hãy xếp các gói kẹo thành hai phần sao cho tổng số kẹo của hai phần chênh lệch nhau ít nhất.

Input cho trong File văn bản `CHIAKEO.IN`

Dòng thứ nhất ghi số n,

Các dòng sau ghi lần lượt các giá trị từ A_1 đến A_n

Output.

Kết quả ghi ra file văn bản `CHIAKEO.OUT`

Dòng thứ nhất ghi số kẹo chênh lệch giữa hai phần,

Dòng thứ hai ghi số hiệu các gói kẹo thuộc phần thứ nhất,

Dòng thứ ba ghi số hiệu các gói kẹo thuộc phần thứ hai.

Các số cách nhau ít nhất một dấu trống.

Phân tích.

Dùng một trục số với các điểm chia nguyên từ 0 đến 40000 để đánh dấu các tổng số kẹo có thể sinh ra khi gộp một số gói kẹo nào đó lại với nhau. Ta lần lượt xét từng gói kẹo từ gói A_1 đến gói A_n. Giả sử bây giờ xét đến gói A_i, để tạo ra các tổng mới ta cộng A_i vào từng tổng đã có (cộng từ tổng lớn nhất về tổng nhỏ nhất là 0)

Mỗi khi sinh ra một tổng mới, cần ghi lưu số hiệu gói kẹo vừa gộp vào để sinh ra tổng mới này.

Từ điểm X là điểm nửa tổng tất cả số kẹo (nếu X đã đánh dấu) hoặc điểm đánh dấu gần điểm X nhất (nếu điểm X không được đánh dấu) biết được gói cuối cùng vừa cho vào một phần, trừ đi số kẹo của gói này ta đến tổng mới (điểm đánh dấu mới) và lại biết số hiệu gói kẹo nào vừa gộp vào để tạo ra tổng mới này…quá trình kết thúc khi đi đến điểm đánh dấu 0.

Ví dụ. Có 5 gói kẹo với số kẹo lần lượt là: A_1= 3, A_2=15, A_3=2, A_4=6, A_5=19. Tổng tất cả các gói có số kẹo là 45. Hy vọng rằng một phần sẽ là 23. Nếu không, cần chọn số kẹo một phần là số gần với số 23 nhất. Số này phải là tổng có thể sinh ra do xếp một số gói kẹo lại với nhau.

Đầu tiên chưa xét gói kẹo nào thì tổng là 0.

Xét gói A_1: có tổng mới là **3**

Xét các gói A_1 và A_2: các tổng mới có thể sinh ra là: 15+3=1**8**, 15+0=1**5**, và tổng cũ là **3**.

Xét các gói A_1, A_2 và A_3: các tổng mới có thể sinh ra là: 2+18=**20**, 2+15=1**7**, 2+3=5 (đã có trước), 2+0=**2**, và các tổng cũ là 1**8**, 1**5**, **3**

Xét các gói A_1, A_2, A_3 và A_4 tương tự sẽ đánh dấu thêm các tổng: **7, 9, 21, 23, 24, 26**

Đến đây thấy đã xuất hiện tổng một số gói kẹo là 23. Tìm ngược lại quá trình sinh ra tổng 23 ta được đáp số: Một phần sẽ gồm các gói kẹo A_4, A_3, A_2, phần thứ hai gồm các gói kẹo còn lại.

Văn bản chương trình

```
Program chiakeo;
const   fi   = 'chiakeo.in';
        fo   = 'chiakeo.out';
        maxn = 200;
        maxk = 200;
        maxs = 40000;
type    m1   = array[0..maxs] of byte;
        m2   = array[1..maxn] of byte;
var     a   : m2;
        t   : m1;
        sum : word;
        n   : byte;
```

```
    Procedure nhap;
    var f: text;
        i: byte;
    begin
        sum:= 0;
        assign(f,fi);    reset(f);
        readln(f,n);
        for i:=1 to n do
        begin
            read(f,a[i]);
            sum:= sum + a[i];
        end;
        close(f);
    end;
    Procedure danhdau;
    var i: byte; max,j: word;
    begin
        fillchar(t,sizeof(t),0);
        t[0]:= 1;
        max := 0;
        for i:=1 to n do
        begin
            for j:=max downto 0 do
            if t[j]>0 then
            if t[j+a[i]]=0 then
                t[j+a[i]]:=i;
            max:= max + a[i];
        end;
    end;
    Procedure timkq;
    var i,tong: integer;
        f     : text;
        kq    : m2;
    begin
        fillchar(kq,sizeof(kq),0);
        assign(f,fo);
        rewrite(f);
        tong:= sum div 2;
        while t[tong]=0 do
                        dec(tong);
        writeln(f,sum-tong-tong));
        repeat
             kq[t[tong]]:= 1;
             tong:= tong - a[t[tong]];
        until tong=0;
        for i:=1 to n do
        if kq[i] = 0 then write(f,i,' ');
        writeln(f);
        for i:=1 to n do
        if kq[i] = 1 then write(f,i,' ');
        close(f);
    end;
BEGIN
    nhap;
    danhdau;
```

```
    timkq;
END.
```

6. Kỹ thuật lùa bò vào chuồng

Ngoài việc dùng kĩ thuật đánh dấu trong khi duyệt, người ta còn hay dùng kĩ thuật "*lùa bò vào chuồng*". Nội dung của kĩ thuật này là: khi duyệt, những thành phần nào có đặc điểm giống nhau thì lưu vào cùng một chỗ. Kĩ thuật này có cái tên ngộ nghĩnh như vậy vì quá trình thực hiện tương tự như công việc lùa những con bò cùng loại vào cùng một chuồng.

Ví dụ xét bài toán: cho mảng A(1..N) một chiều gồm N phần tử là các số nguyên không âm đánh số từ 1 đến N, có giá trị không vượt quá M, hãy tìm số nguyên không âm nhỏ nhất chưa có mặt trong mảng A.

Giải bài toán này ta làm như sau: Dùng mảng một chiều B[0..M+1] để làm vai trò dãy chuồng bò. Duyệt mảng A[1..N], cho "con bò" A[i] vào "chuồng" B[j] với chỉ số j=A[i], nghĩa là tăng B[A[i]] lên một đơn vị. Sau đó duyệt lại B theo chỉ số tăng dần từ 0, gặp phần tử đầu tiên bằng 0 thì chỉ số của phần tử này chính là số nguyên không âm nhỏ nhất chưa có mặt trong mảng A (loại bò này không có mặt trong đàn bò). Kỹ thuật này có tốc độ tuyến tính (chỉ cần duyệt mảng A một lần).

Ví dụ 1. Mã nhân viên.

Tổng Giám đốc công ty X nổi tiếng là một người kĩ lưỡng. Ông ta thực hiện việc quản lý nhân viên của mình bằng cách gán cho mỗi nhân viên một mã số khác nhau. Công ty có N nhân viên, nhân viên i (i=1, 2, …N) có mã số A_i. Do bận đi công tác nước ngoài một thời gian dài nên ông trao quyền quản lý công ty cho một người khác. Khi ông trở về, công ty đã có sự thay đổi về số lượng nhân viên. Khi tiếp nhận thêm nhân viên mới, ông yêu cầu muốn biết mã số nhỏ nhất có thể gán cho nhân viên mới.

Yêu cầu: Cho N mã số của nhân viên trong công ty. Hãy tìm mã số nhỏ nhất chưa xuất hiện trong N mã số đã cho

Dữ liệu vào cho trong tệp văn bản MASO.IN: Dòng đầu là số N ($1<N\leq 30000$). N dòng tiếp theo, dòng thứ i ghi số Ai ($i=1..N; 1\leq Ai\leq 109$)

Kết quả ghi vào tệp văn bản MASO.OUT một số duy nhất là mã số tìm được.

Ví dụ

```
MASO.IN  MASO.OUT
6        2
7
5
6
1
3
4
```

Phân tích

Đây là bài sử dụng kỹ thuật "lùa bò vào chuồng".

Văn bản chương trình

```
const fi='MASO.in';
      fo='MASO.out';
var   n, k : longint;
      b : array[0..30001] of longint;
      ai : longint;
Procedure readfile;
var f : text; i : longint;
begin
  assign(f,fi); reset(f);
  readln(f,n);
  for i:=1 to n do b[i] := 0;
  for i:=1 to n do begin
    read(f,ai);
    if ai<=n+1 then inc(b[ai]);
  end;
  close(f);
end;
Procedure process;
var i : longint;
begin
  k := 1;
  for i:=1 to n+1 do
  if b[i]=0 then begin
     k := i;
     exit;
  end;
end;
Procedure writefile;
var f : text; i : longint;
begin
  assign(f,fo); rewrite(f);
  writeln(f,k);
  close(f);
end;
BEGIN
  readfile;
  process;
  writefile;
END.
```

Ví dụ 2. Tìm tổng Nim

Cho bảng G hai chiều có dòng tiêu đề (dòng đầu tiên) là các giá trị của x (nguyên không âm) và cột tiêu đề (cột bên trái) là các giá trị của y (nguyên không âm). Giao giữa dòng giá trị của y và của x là một số nguyên z không âm gọi là tổng Nim của x và y. Quy tắc tìm tổng Nim z của x và y như sau: z là số nguyên không âm nhỏ nhất khác với các số bên trái nó và phía trên nó. Hãy lập trình hiện trên màn hình bảng G có các giá trị z là tổng Nim của x và y với $0 \le x, y \le 9$.

Phân tích

Dùng kỹ thuật "lùa bò vào chuồng".

Văn bản chương trình

```
Program tim_tong_NIM;
var   g  : array[0..9,0..9] of integer;
      c  : array[0..100] of integer;
      i,j,k : integer;
begin
    fillchar(g,sizeof(g),0);
    g[0,0] := 0;
    for i:=0 to 9 do g[i,0] := i;
    for j:=0 to 9 do g[0,j] := j;
    for i:=1 to 9 do
    for j:=1 to 9 do begin
        fillchar(c,sizeof(c),0);
        for k:=0 to i-1 do c[g[k,j]] := 1;
        for k:=0 to j-1 do c[g[i,k]] := 1;
        k:=0;
        while c[k]=1 do inc(k);
        g[i,j] := k;
    end;
    for i:=0 to 9 do begin
        for j:=0 to 9 do write(g[i,j]:3);
        writeln;
    end;
    readln
end.
```

KẾT LUẬN

Phương pháp duyệt được xem là một chiến lược tổng quát, có tính định hướng tìm thuật toán. Chẳng thế mà các đề thi học sinh giỏi môn Tin học từ cấp cơ sở đến cấp quốc gia năm nào cũng có những bài toán có thể giải bằng phương pháp duyệt. Song với yêu cầu ngày càng cao, để có được một chương trình đáp ứng được đòi hỏi về bộ nhớ, về thời gian thực hiện cần có nhiều sáng tạo. Đó là sự kết hợp giữa giải thuật, cấu trúc dữ liệu và kỹ thuật lập trình. Giải thuật và cấu trúc dữ liệu có thể học trong các tài liệu, còn về kỹ thuật lập trình thì rất ít thấy nói đến. Với suy nghĩ như vậy, cùng với thực tế giảng dạy, chúng tôi thu thập, sưu tầm một số kỹ thuật duyệt có thể áp dụng với một số lớp bài toán. Rất mong được các đồng nghiệp tham góp ý kiến.

Xin trân trọng cảm ơn!