# Quy hoạch động

## Thuật toán qui hoạch động

**Bá Hiệp**

Trong quá trình học tập, chúng ta gặp rất nhiều các bài tập về Toán-Tin. Các bài tập dạng này rất phong phú và đa dạng. Thực tế chưa có thuật toán hoàn chỉnh có thể áp dụng cho mọi bài toán. Tuy nhiên người ta đã tìm ra một số thuật toán chung như chia để trị, tham ăn, quay lui,... Các thuật toán này có thể áp dụng để giải một lớp khá rộng các bài toán hay gặp trong thực tế. Trong bài viết này, tôi muốn đề cập với các bạn một thuật toán khác, đó là thuật toán quy hoạch động. Tư tưởng cơ bản của thuật toán là:
Để giải một bài toán ta chia bài toán đó thành các bài toán nhỏ hơn có thể giải một cách dễ dàng. Sau đó kết hợp lời giải các bài toán con, ta có được lời giải bài toán ban đầu. Trong quá trình giải các bài toán con đôi khi ta gặp rất nhiều kết quả trùng lặp của các bài toán con. Để tăng tính hiệu quả, thay vì phải tính lại các kết quả đó, ta lưu chúng vào một bảng. Khi cần lời giải của một bài toán con nào đó ta chỉ cần tìm trong bảng, không cần tính lại.
Tư tưởng của thuật toán quy hoạch động khá đơn giản. Tuy nhiên khi áp dụng thuật toán vào trường hợp cụ thể lại không dễ dàng (điều này cũng tương tự như nguyên tắc Dirichlet trong toán vậy). Khi giải bài toán bằng phương pháp này, chúng ta phải thực hiện hai yêu cầu quan trọng sau:
- Tìm công thức truy hồi xác định nghiệm bài toán qua nghiệm các bài toán con nhỏ hơn. - Với mỗi bài toán cụ thể, ta đề ra phương án lưu trữ nghiệm một cách hợp lý để từ đó có thể truy cập một cách thuận tiện nhất.
Để minh hoạ thuật toán, ta xét một vài ví dụ.
***Ví dụ 1:***Cho hai dãy số nguyên ($a_1,a_2,...,a_m$), ($b_1,b_2,...,b_n$). Tìm dãy con chung có độ dài lớn nhất của hai dãy trên (coi dãy không có số nguyên nào là dãy con của mọi dãy và có độ dài bằng 0).
**Lời giải**
Chúng ta có thể thấy ngay rằng độ phức tạp của bài toán trên phụ thuộc vào hai số m, n. Xét hai trường hợp:
+*Trường hợp1:* m=0 hoặc n=0.
Đây là trường hợp đặc biệt, có duy nhất một dãy con chung của hai dãy có độ dài? bằng 0. Vì vậy dãy con chung có độ dài lớn nhất của chúng có độ dài bằng 0.
+*Trường hợp 2:* m0 và n0.
Trong trường hợp này, ta xét các bài toán nhỏ hơn là tìm dãy con chung có độ dài lớn nhất của hai dãy ($a_1,a_2,...,a_i$), ($b_1,b_2,...,b_j$) với $0 \le i \le m$, $0 \le j \le n$. Go.i l[i,j] là độ dài của dãy con chung lớn nhất của hai dãy ($a_1,...,a_i$), ($b_1,...,b_j$). ; Như vậy ta phải tính tất cả các l[i,j] trong đó 0 <= i <= m, 0 <= j <= n.
Chúng ta có thể thấy ngay rằng l[0,0]=0. Giả sử ta tính được l[s,t] với 1<S<t<j.<>
-Nếu $i_i$ $b_j$ thì l[i,j]=max{l[i-1,j], l[i,j-1]}.
-Nếu $i_{i</SUB>=Bj}$ thì l[i,j]= 1+l[i-1,j-1].
Với những nhận xét trên, ta hoàn toàn tính được l[m,n] chính là độ dài dãy con chung dài nhất của (a1,..am), (b1,..bn).
Để tìm phần tử của dãy con, ta xuất phát từ ô l[m,n] tới ô l[0,0]. Giả sử ta đang ở ô l[i,j]. Nếu ai</SUB>=Bj thì ta thêm ai vào dãy con rồi nhảy tới ô l[i-1,j-1]. Nếu aibj thì l[i,j]=l[i-1,j] hoặc l[i,j]=l[i,j-1]. Nếu l[i,j]=l[i-1,j] thì nhảy tới ô l[i-1,j], ngược lại thì nhảy tới ô l[i,j-1].

Sau đây là lời giải của bài toán. Chương trình được viết bằng ngôn ngữ Pascal:

```
uses crt;
const fi='b2.inp';
var
a:array[1..10] of integer;
b:array[1..10] of integer;
kq:array[0..10,0..10] of integer;
i,j,maxa,maxb:integer;
f:text;
procedure init;
begin
assign(f,fi);reset(f);i:=0;
while not(eoln(f)) do begin inc(i);read(f,a[i]);end;maxa:=i;
readln(f);i:=0;
while not(eoln(f)) do begin inc(i);read(f,b[i]);end;maxb:=i;
close(f);
end;
function max(a,b:integer):integer;
begin
if a>b then max:=a else max:=b;
end;
begin
init;
kq[0,0]:=0;
for i:=1 to maxa do for j:=1 to maxb do
if a[i]<>b[j] then kq[i,j]:=max(kq[i-1,j],kq[i,j-1])
else kq[i,j]:=kq[i-1,j-1]+1;
writeln('Do dai day con chung lon nhat:',kq[maxa,maxb]);
i:=maxa;j:=maxb;
while (i>0)or(j>0) do
if a[i]=b[j] then begin write(a[i]);dec(i);dec(j);end
else if kq[i-1,j]=kq[i,j] then dec(i) else dec(j);
end.
```

Với nội dung file?b2.inp? chứa 2 dãy (a1,a2,..am) ,(b1,b2,..bn) sau:

1 2 3 2 3 4 6

6 9 8 7

Xét bài toán kinh điển về tối ưu tổ hợp:

***Ví dụ 2:***Cho cái túi chứa được trọng lượng tối đa là w. Có n đồ vật, đồ vật thứ i có khối lượng a[i] và giá trị c[i], $1 \le i \le n$. Tìm cách xếp đồ vật vào túi sao cho đạt giá trị lớn nhất.

**Lời giải**

Gọi f(k,v) là giá trị lớn nhất của túi đựng trọng lượng v và chỉ chứa các đồ vật từ 1 đến k.

Nếu k=1 thì f(k,v)=(v div a[1])*c[1]. Giả sử tính được f(s,t) với 1

Đặt: tg=v div a[k], f(k,v)=max{f(k-1,u)+x*c[k]}? (*) ,với x=0,1,2,...,tg, u=v-x*a[k]

Giá trị lớn nhất là f(n,w). Ta dùng mảng bản ghi a[1..n,1..w] chứa kết quả trung gian. Mỗi bản ghi a[k,v] chứa giá trị f(k,v) và giá trị x thoả mãn công thức (*).

Để xác định số lượng x[i] đồ vật i thoả mãn điều kiện tối ưu, ta xuất phát từ a[n,w] xác định được x[n]. Nhảy tới a[n-1,w-x[n]*a[n]] xác định được x[n-1]. Cứ như vậy tới x[1].

Sau đây là lời giải, chương trình được viết bằng ngôn ngữ Pascal:

```
uses crt;
const n=5;w=17;
fi='b3.inp';
type kq=record
num,val:integer;
end;
var
a:array[1..10] of integer;{khoi luong}
c:array[1..10] of integer;{Gia tri}
i,j,tg,k,max,save:integer;
f:text;
b:array[1..n,1..w] of kq;
procedure init;
begin
assign(f,fi);reset(f);
for i:=1 to n do begin read(f,a[i],c[i]);end;
close(f);
end;
begin
init;
for j:=1 to w do? for i:=1 to n do
begin
tg:=j div a[i];max:=0;
for k:=0 to tg do if (b[i-1,j-k*a[i]].val+k*c[i])>max then
begin max:=b[i-1,j-k*a[i]].val+k*c[i];save:=k;end;
b[i,j].val:=max;
b[i,j].val:=max;
b[i,j].num:=save;
for i:=1 to n do
begin
for j:=1 to w do write(b[i,j].val:3);
writeln;
end;
writeln('Max:',b[n,w].val);
i:=n;j:=w;
while i>=1 do
begin
if b[i,j].num>0 then writeln('Co ',b[i,j].num,' do vat ',i);
j:=j-a[i]*b[i,j].num;dec(i);
end;
readln;
end.
```

Với nội dung file?b3.inp? :hàng i chứa khối lượng a[i], giá trị c[i]:

3 4

4 5

7 10

8 11

9 13

Qua hai ví dụ trên chắc các bạn đã nắm được tư tưởng của thuật toán qui hoạch động cũng ; như cách cài đặt cho nó. ; Như các bạn thấy, cách phát biểu thuật toán rất đơn giản. Nếu biết cách vận dụng thuật toán một cách hợp lý, ta có thể giải được một lớp khá rộng các bài toán trong thực tế. Hi vọng thuật toán sẽ là công cụ tốt của các bạn trong quá trình học tập môn tin học. Chúc các bạn thành công.

Bá Hiệp

## Thuật toán quy hoạch động trên mảng một chiều

**Trần Minh Quang**

**Bài toán 1:** Cho một dãy số nguyên dương $a_1$, $a_2$,... $a_N$. Hãy tỉa bớt một số ít nhất các phần tử của dãy số nguyên đó và giữ nguyên thứ tự các phần tử còn lại sao cho dãy số còn lại là một dãy tăng dần. Ta gọi dãy số nguyên tăng dần còn lại sau khi đã tỉa bớt một số phần tử là dãy con của dãy đã cho.

***Input:*** Dữ liệu vào được cho bởi tệp văn bản với quy cách:

- Dòng đầu ghi số N là số phần tử
- Dòng tiếp theo ghi N số là các số nguyên của dãy.

***Output:***

-Ghi ra màn hình: Số lượng phần tử của dãy con cực đại và chỉ số các phần tử trong dãy con đó (theo thứ tự tăng dần).

Ví dụ:

- Với Input trong file DAYSO.INP như sau:

10

10 100 20 1 2 50 70 80 3 60

- thì Output phải là:

1 2 50 70 80

ý tưởng của thuật toán quy hoạch động ở đây là: Để xây dựng dãy con dài nhất của dãy đã cho chúng ta sẽ xây dựng dãy con dài nhất của đoạn phần tử đầu $a_1$, $a_2$,... $a_i$.

Để làm được điều đó: ta gọi S[i] là số lượng phần tử nhiều nhất của dãy con tăng dần, trong đó $a_i$ cũng thuộc dãy con trên (nó là phần tử cuối cùng).

Chúng ta sẽ tính S[i] ở từng bước dựa vào các S[i-1],... S[1] như sau:

Ban đầu S[i] với i = 1, 2,... N được gán bằng 1 vì trường hợp xấu nhất thì dãy con chỉ là một phần tử.

Với mỗi i >= 2 thì S[i] được tính bằng công thức truy hồi sau:

S[i]:=Max(S[j]+1) với j=i-1,... 1 mà $a_j < a_i$.

Để lấy lại dãy con cực đại ta dùng một mảng Truoc với ý nghĩa: Truoc[i] là chỉ số của phần tử trước phần tử i trong dãy con cực đại lấy trong dãy $a_1$,$a_2$,... $a_i$.

Bây giờ chúng ta phải tìm vị trí i sao cho S[i] đạt max. Ta lưu vị trí đó vào biến Luu.

Như vậy: S[Luu] chính là số lượng phần tử của dãy con cực đại của dãy đã cho. Và bằng mảng Truoc ta có thể lấy lại chỉ số các phần tử thuộc dãy con đó.

Đến đây ta gặp một vấn đề: Mảng Truoc chỉ cho phép ta lần ngược từ cuối về đầu dó đó để in ra các chỉ số theo thứ tự tăng dần ta phải dùng thêm một mảng phụ P và in ngược lại của mảng P:

```
dem:=0;
i:=Luu;
While i<>0 do
Begin
Inc(dem);P[dem]:=i; i:=Truoc[i];
End;
```
Chỉ số theo thứ tự tăng dần của dãy con cực đại được in ra bằng dòng lệnh:
```
For i:=dem downto 1 do Write(P[i],' ');
```
Tuy nhiên làm như trên có vẻ dài dòng trong khi chúng ta đã nhận ra tính đệ quy trong việc lấy ngược lại. Và thủ tục in ra dãy con đã rất ngắn gọn và sáng sủa:
```
Procedure Print(i:Integer);
Begin
If i>0 then
Begin
Print(Truoc[i]);Write(i,' ');
End;
End;
```
Công việc in ra chỉ cần một lời gọi: Print(Luu);
Ta có toàn văn chương trình:
```
{$A+,B-,D+,E+,F-,G-,I+,L+,N-,O-,P-,Q-,R-,S+,T-,V+,X+,Y+}
{$M 65500,0,655360}
Uses Crt;
Const fi = 'DAYSO.INP';
MaxN=5000;
Var A : Array[1..MaxN] of Integer;
S : Array[1..MaxN] of Integer;
Truoc : Array[1..MaxN] of Integer;
i,j,Luu : Word;
N : Word;
Procedure Init;
Begin
Fillchar(S,SizeOf(S),1);
Fillchar(Truoc,SizeOf(Truoc),0);
End;
Procedure Readfile;
Var f:Text;
Begin
Assign(f,fi);
Reset(f);
Readln(f,N);
For i:=1 to N do Read(f,A[i]);
Close(f);
End;
Procedure Find;
Begin
For i:=2 to N do
```

```
Begin
For j:=i-1 downto 1 do
If (A[j]< (S[i]
Begin
S[i]:=S[j]+1;
Truoc[i]:=j;
End;
End;
End;
Procedure Print(i:Word);
Begin
If i >0 then
Begin
Print(Truoc[i]);
Write(a[i],' ');
End;
End;
Procedure OutputResult;
Begin
Luu:=N;
For i:=N-1 downto 1 do
If S[i]>S[Luu] then Luu:=i;
Print(Luu);
End;
BEGIN
Clrscr;
Init;
Readfile;
Find;
OutputResult;
Readln;
END.
```

Qua ví dụ trên chúng ta đã hiểu cách mà thuật toán thể hiện. Bây giờ chúng ta sẽ xét tiếp một bài toán sắp xếp trình tự phục vụ khách hàng mà cách giải đều sử dụng thuật toán Quy hoạch động trên mảng một chiều.

Ta xét tiếp một ví dụ sau:

**Bài toán 2:** Tại thời điểm 0, ông chủ một máy tính hiệu năng cao nhận được đơn đặt hàng thuê sử dụng máy của n khách hàng. Các khách hàng được đánh số từ 1 đến n. Khách hàng i cần sử dụng máy từ thời điểm $d_i$ đến thời điểm $c_i$ ($d_i$, $c_i$ là các số nguyên và $0 < d_i < c_i < 1000000000$) và sẽ trả tiền sử dụng máy là $p_i$ ($p_i$ nguyên, $0 < p_i \leq 10000000$). Bạn cần xác định xem ông chủ cần nhận phục vụ những khách hàng nào sao cho khoảng thời gian sử dụng máy của hai khách được nhận phục vụ bất kỳ không được giao nhau đồng thời tổng tiền thu được từ việc phục vụ họ là lớn nhất.

***Dữ liệu vào:*** Từ file văn bản THUE.INP

Dòng đầu tiên ghi số n ($0 < n =< 1000$);

- Dòng thứ i+1 trong số n dòng tiếp theo ghi 3 số $d_i$, $c_i$, $p_i$ cách nhau bởi dấu trắng (i = 1, 2,... n).

***Kết quả:*** Ghi ra file văn bản THUE.OUT

-Dòng đầu tiên ghi hai số nguyên dương theo thứ tự là số lượng khách hàng nhận phục vụ và tổng tiền thu được từ việc phục vụ họ.

-Dòng tiếp theo ghi chỉ số của các khách hàng được nhận phục vụ.

Ví dụ:

| THUE.INP | THUE.OUT | | THUE.INP | THUE.OUT |
|---|---|---|---|---|
| 3<br>150 500 150<br>1 200 100<br>400 800 80 | 2 180<br>2 3 | | 4<br>400 821 800<br>200 513 500<br>100 325 200<br>600 900 600 | 2 1100<br>2 4 |

Bài toán này chúng ta phải chú ý ở chỗ: Để dùng thuật toán Quy hoạch động tối ưu từng bước thì trước hết chúng ta phải sắp xếp các $c_i$ theo thứ tự tăng dần:

Giả sử $c_1 \le c_2 \le ... \le c_N$.

Tương tự bài toán trên: Gọi F[k] là số tiền lớn nhất khi phục vụ một số khách hàng từ 1 đến k.

Với mỗi F[k] ta có:

- Nếu chấp nhận phục vụ khách k thì F[k]:=F[t]+$p_k$ (với t là chỉ số max thoả mãn khoảng thời gian [$d_t$, $c_t$ [$d_k$,$c_k$] = ).
- Nếu không chấp nhận phục vụ k thì F[k]:=F[k-1].

Như vậy hàm quy hoạch động của F[k] sẽ là:

F[k]:=Max{F[t]+$p_k$ ,F[k-1]} với k = 2, 3,... N và t có ý nghĩa như trên.

Để lấy lại chỉ số các khách hàng được phục vụ chúng ta lại dùng mảng Truoc như ví dụ trên.

Trên đây là những gì tôi muốn trình bày với các bạn. Theo tôi, thuật toán tuy đơn giản nhưng tầm ứng dụng của nó rất phong phú mà nếu nắm vững nó là rất có lợi cho tư tưởng thuật toán của các bạn.

## Giải thuật quy hoạch động

**CongHiep_87@yahoocom**

Đối với các bạn yêu thích môn lập trình thì có lẽ giải thuật qui hoạch động tương đối quen thuộc trong việc giải quyết các vấn đề tin học. Tuy nhiên, sẽ thật là khó để có thể tìm được cơ cở và công thức cho việc sử dụng qui hoạch động. Chính vì vấn đề này, qui hoạch động lại trở thành không phổ biến. Đối với những bài toán như vậy, chúng ta lại cố gắng đi tìm cách giải khác ví dụ như vét cạn hay tham lam....điều đó thật là dở! Chính vì vậy, tôi muốn đưa ra một số bài toán áp dụng qui hoạch động để mong rằng sau bài báo này, các bạn sẽ yêu thích giải thuật này hơn.

Trước hết các bạn phải luôn nhớ rằng, giải thuật qui hoạch động được xuất phát từ nguyên lí Bellman: nếu 1 cấu hình là tối ưu thì mọi cấu hình con của nó cũng là tối ưu. Chính vì vậy để xây dựng 1 cấu hình tối ưu, ta hãy xây dựng dần các cấu hình con sao cho các cấu hình con này cũng phải tối ưu Đây chính là đường lối chủ đạo cho mọi bài toán qui hoạch động. Sau đây là một số bài toán được giải quyết bằng qui hoạch động.

**I. Các bài toán**

**Bài 1:** Trước tiên chúng ta hãy xét 1 bài toán thật đơn giản và quen thuộc đó là tìm giá trị lớn nhất trong n số là $a_1$, $a_2$, ..., $a_n$. Giải quyết bài toán này, ta sẽ xây dựng các cấu hình con tối ưu bằng cách lần lượt tìm số lớn nhất trong k số đầu tiên với k chạy từ 1 đến n:

K=1: max1:=a1;
K=2: max2:=max(max1,a2);
K=3: max3:=max(max2,a3);
.....
K=n: maxn:=max(maxn-1,an);

Như vậy khi k đạt tới n thì $max_n$ chính là giá trị lớn nhất trong n số đã chọ Việc cài đặt chương trình hết sức đơn giản như sau:

```
Uses crt;
Var a: array[1..100] of integer;
n,k,max: integer;
Begin
Write('Cho so luong phan tu: ');readln(n);
For i:=1 to n do begin write('a[',i,']= ');readln(a[i]);end;
Max:=a[1];
For k:=2 to n do
If a[k]>max then max:=a[k];
Write('Gia tri lon nhat cua day cac so da cho la: ',max);
Readln
End.
```

Bây giờ chúng ta xét đến bài toán 2 có phần hấp dẫn hơn. Đây chính là một trong những bài toán điển hình cho giải thuật qui hoạch động:

**Bài 2:** *Bài toán cái túi*: Cho n loại đồ vật ($1 \le n \le 100$) với một đồ vật loại thứ i ($1 \le i \le n$) có trọng lượng là a[i] và giá trị sử dụng là c[i]. Một nhà thám hiểm cần mang theo một số đồ vật vào túi của mình sao cho tổng trọng lượng các đồ vật đem theo không vượt quá sức chịu đựng của túi là w ($1 \le w \le 250$) và tổng giá trị sử dụng từ các đồ vật đem theo là lớn nhất. Hãy tìm một phương án mang cho nhà thám hiểm với giả sử rằng số lượng đồ vật của mỗi loại là luôn đủ dùng.

* Thuật giải bằng qui hoạch động được mô tả như sau:

Ta xây dựng một mảng 2 chiều f với f[i,j] là giá trị sử dụng lớn nhất có được bởi j vật từ 1 đến j mà tổng trọng lượng không vượt quá j.

Khởi tạo : f[i,1]:=0 với i < a[1]
F[i,1]:=c[1]*(i div a[1]) với i > =a[1]; (i = 1..w);

Ta lần lượt cho i đạt tới w và j đạt tới n bằng cách sau:

```
For j:=2 to n do
For i:=1 to w do
If i >= a[i] then f[i,j]:=Max(f[i-a[j],j]+ c[j],f[i-1,j])
Else f[i,j]:=f[i-1,j].
```

Như vậy cho đến f[w,n] ta sẽ thu được giá trị lớn nhất có thể đạt được từ n loại đồ vật đã cho sao cho trọng lượng không vượt quá w. Hệ thức toán trên được gọi là hệ thức Dantzig. Có thể rất dễ hiểu được thuật toán như sau:

Phần khởi tạo: f[i,1] có nghĩa là giá trị lớn nhất nếu chỉ có 1 loại vật (ở đây là vật 1) mà trọng lượng không quá ị Như vậy nếu i < a[1] thì rõ ràng không thể mang theo vật nào và giá trị f=0. Ngược lại nếu i ≥ a[1] thì số vật được phép mang theo đi sẽ là i div a[1] và giá trị đạt

được là f= c[1]*(i div a[1]).
Phần xây dựng: chúng ta xét đến f[i,j] có nghĩa là xét đến giá trị lớn nhất có thể đạt được từ j loại đồ vật (1,,j) mà trọng lượng không qúa i. Vậy thì rõ ràng là nếu i < a[j] thì có nghĩa là đồ vật j không thể mang đi hay với trọng lượng là i thì ta vẫn không thể cải thiện được giá trị f và f vẫn nhận giá trị f[i,j-1]. Ngược lại nếu i ≥a[j] thì chúng ta xét việc nếu mang thêm vật j thì sẽ có lợi hơn việc không mang hay không, điều đó có nghĩa là xét Max(f[i-a[j],j]+ c[j],f[i-1,j]).
Chương trình cài đặt giải quyết bài toán cái túi rất đơn giản như sau:

```
Uses crt;
Var value,weight:array[1..30]of 0..500;{value: gia tri;weight: trong luong}
f:array[0..500,0..30] of 0..10000;
w,w1,sl:integer;
fi:text;
Procedure Init;
Var i:byte;
Begin
clrscr;
assign(fi,'tuitxt');reset(fi);
readln(fi,w,sl);w1:=w;
for i:=1 to sl do readln(fi,weight[i],value[i]);
End;
{***********************************************}
Procedure Solve;
Var i,j:word;
Begin
for j:=1 to sl do f[0,j]:=0;
for i:=1 to w do f[i,1]:=(i div weight[1])*value[1];
for j:= 2 to sl do
for i:=1 to w do
begin
if i else begin
f[i,j]:=f[i,j-1];
if (value[j]+f[i-weight[j],j])>f[i,j] then
f[i,j]:=(value[j]+f[i-weight[j],j]);
end;
end;
(***********************************************}
Procedure Print_rerult;
Var i:byte;
Begin
write('* Gia tri cao nhat dat duoc la: ',f[w,sl]);writeln;
End;
(***********************************************)
Begin
Init;
Solve;
```

```
Print_result;
Readln;
End.
```
Chú ý: chương trình trên được đọc dữ liệu từ file.

**II. Vấn đề công thức truy hồi**

Đối với một bài toán qui hoạch động thì công thức truy hồi cũng là một phần rất quan trọng. Nếu chúng ta chỉ xây dựng được giá trị tối ưu thì đôi khi vẫn là chưa đủ. Vấn đề được đặt ra là làm thế nào để xác định được cấu hình tối ưu Để giải quyết vấn đề này ta lại phải xác định được công thức truy hồi Thực tế là để xác định được công thức truy hồi này thì cũng không phải quá khó bởi từ công thức qui hoạch động chúng ta cũng có thể suy ngay ra được công thức truy hồi

Tôi xin trở lại với bài toán cái túi đã nêu ở trên để xây dựng cấu hình tối ưu cho bài toán cái túi có nghĩa là phải mang những loại vật nào và mỗi loại vật là bao nhiêu để có được giá trị sử dụng max: Xây dựng hàm phụ choose[i,k] với ý nghĩa để đạt được giá trị tốt nhất tại f[i,k] thì cần phải sử dụng đến loại đồ vật nào (i=1..w,k=1..n) bằng cac công thức sau:

Choose[i,1]:=0 nếu i

Ta lần lượt cho k chạy tới n và i chạy tới w để xây dựng mảng choose như sau:

Nếu f[i,k]=f[i,k-1] thì choose[i,k]:=choose[i,k-1] (do không mang vật k)

Nếu không thì n choose[i,k]:=k (có nghĩa mang theo vật k)

Khi xây dựng đến choose[w,n] thì ta chỉ cần chú ý đến cột cuối cùng của mảng choose và bắt đầu truy hồi. Giả sử mảng number[i] (i=1..n) sẽ cho ta số lượng loại vật i được mang theo. Ta sẽ cải thiện chương trình giải bài toán cái túi ở trên như sau:

```
Program Bai_toan_cai_tui;
Uses crt;
Var value,weight,number:array[1..20]of 0..1000;{value:gia tri}
f,choose:array[0..1200,0..12]of 0..10000;
w,w1,sl:0..2000;
fi:text;
Procedure Init;
Var i:byte;
Begin
clrscr;
assign(fi,'tui.txt');reset(fi);
readln(fi,w,sl);w1:=w;
for i:=1 to sl do readln(fi,weight[i],value[i]);
End;
{***********************************************}
Procedure Solve;
Var i,j:word;
Begin
for j:=1 to sl do begin f[0,j]:=0;choose[0,j]:=0;end;
for i:=1 to w do
begin
f[i,1]:=(i div weight[1])*value[1];
if i>=weight[1] then choose[i,1]:=1
else choose[i,1]:=0;
```

```
end;
for j:= 2 to sl do
for i:=1 to w do
begin
choose[i,j]:=choose[i,j-1];
if i else begin
f[i,j]:=f[i,j-1];
if (value[j]+f[i-weight[j],j])>f[i,j] then
begin
f[i,j]:=(value[j]+f[i-weight[j],j]);
choose[i,j]:=j;
end;
end;
end;
for i:=1 to sl do number[i]:=0;
while choose[w1,sl]<>0 do
begin
number[choose[w1,sl]]:=number[choose[w1,sl]]+1;
w1:=w1-weight[choose[w1,sl]];
end;
End;
{************************************************}
Procedure Print;
Var i:byte;
Begin
write('* Gia tri cao nhat dat duoc la: ',f[w,sl]);writeln;
write('* Khoi luong da dung la: ',w-w1);writeln;writeln;
writeln('* Nha tham hiem can dem nhu sau: ');
for i:=1 to sl do
if number[i]<>0 then
begin write(' - ',number[i],' vat ',i, ' voi trong luong ',number[i]*weight[i],' va gia tri:
',number[i]*value[i]);
writeln;
end;
End;
{************* Main **********************}
Begin
Init;
Solve;
Print;
Readln;
End.
```

**III. Bàn luận**

Về bài toán cái túi còn rất nhiều lời giải Ta cũng có thể giải quyết bài toán cái túi bằng thuật toán nhánh cận. Ưu điểm lớn nhất của thuật toán nhánh cận là có thể chỉ ra được mọi cấu hình tối ưu của bài tóan, tuy nhiên trong trường hợp xấu nhất, nhánh cận lại chính là vét cạn.

Chính vì vậy, thời gian để thực hiện chương trình bằng nhánh cận sẽ rất lâu
Rất tiếc rằng, giải thuật qui hoạch động luôn luôn chỉ nêu ra được một cấu hình tối ưu. Nếu chúng ta giải bằng qui hoạch động như trên, thời gian chạy chương trình rất nhanh chóng.
Chương trình trên hoàn toàn có thể cải thiện được bằng cách thay vì dùng mảng 2 chiều f và choose ta có thể chỉ dùng 4 mảng 1 chiều đó là f1, f2, choose1, choose2 bởi thực chất tại cột j của f thì ta chỉ có thể liên quan đến cột j-1 của f. Chính vì vậy, 2 mảng f1,f2 có thể dùng thế lần lượt cho nhau tương đương dùng mảng 2 chiều f. Khi đó chương trình sẽ có thể chạy với bộ dữ liệu cỡ vài nghìn!
Thuật toán qui hoạch động còn được ứng dụng trong rất nhiều bài toán, tôi xin được nêu ra thêm một số bài toán khác nữa :
*Bài 3:* Một tam giác được tạo bởi các số x và sắp xếp như hình bên
Hãy tìm đường đi từ đỉnh xuống đáy sao cho: tổng các số đi qua là
lớn nhất. Cho biết:
- x là các số nguyên bất kì từ 0 đến 99.
- tam giác có số hàng <=20.
- mỗi bước đi: xuống 1 hàng tới số gần nhất bên trái hay phải.
* Dữ liệu: đọc từ file 'vaoinp' có dạng:
- Dòng đầu: số lượng dòng của tam giác.
- Từ dòng 2: các số cụ thể.
* Output: in ra màn hình
- Hình tam giác cân được tạo từ các số.
- Giá trị tổng các số đã gặp trên đường đi.
- Các số đã gặp trên đường đi
( Câu 2 trong đề thi chọn đội tuyển Tin học Hà Nội 2001-2002)
*Bài 4:* Chúng ta hãy giải quyết bài toán cái túi nhưng được thay đổi đi một số chi tiết như sau: Một nhà thám hiểm cần đem theo một số đồ vật vào cái túi có trọng tải không quá w của ông. Có tất cả n đồ vật, mỗi đồ vật i có trọng lượng là a[i] và giá trị sử dụng là c[i]. Hãy giúp nhà thám hiểm cách mang các đồ vật sao cho tổng giá trị sử dụng là lớn nhất có thể được (mỗi đồ vật chỉ có thể mang 1 lần hoặc không mang).
Một bài báo không thể nói hết được tất cả những ưu việt của cả một thuật toán. Tuy nhiên, sau bài báo này, tôi hy vọng các bạn sẽ hay sử dụng qui hoạch động hơn trong việc giải toán. Nếu bạn nào muốn lời giải cụ thể của tất cả các bài toán trên, hãy liên hệ với tôi theo địa chỉ:

## Quy hoạch tối ưu một bảng hai chiều - Bài toán tổng quát

**Đỗ Sơn Huỳnh**

*Có rất nhiều bài toán tối ưu trên một bảng cho trước gồm M dòng, N cột như các dạng bài tìm một hành trình đi từ dòng thứ nhất tới dòng thứ M thoả mãn một điều kiện tối ưu nào đó. Nhưng cũng có những bài toán tối ưu với số liệu ban đầu là các mảng phần tử một chiều đều có thể đưa về bài toán quy hoạch tối ưu trên một bảng hai chiều. Một ví dụ dễ thấy và dễ gặp nhất là bài toán tìm xâu con lớn nhất, tìm đoạn dãy con đơn điệu dài nhất, bài toán cây xăng, và điển hình nhất là bài toán cái túi (với dữ liệu đầu vào là nguyên).*
*Tất cả các bài toán đó chúng ta đều có thể đưa về một dạng tổng quát mà tôi tạm gọi là* ***"Bài toán tổng quát quy hoạch tối ưu trên một bảng hai chiều"****. Bài viết này là sự tổng hợp của bản thân tôi trong quá trình học môn tin học PASCAL. Xin nêu ra để các bạn có thể tham khảo và cho những ý kiến quý báu.*

**Phát biểu bài toán**

Cho một bảng gồm M dòng, N cột. Hãy tìm một phương án tối ưu để ″đi ″ từ dòng thứ nhất đến hết dòng thứ M với các nguyên tắc sau:

**1. Điều kiện tối ưu:**

Là điều kiện bài toán đưa ra. Đường đi tối ưu được tính bằng tổng trọng số các ô đi qua. Trọng số của một ô phụ thuộc quy tắc tính trọng số của bài toán.

**2. Quy tắc tính trọng số:**

- Trọng số bằng trị số chính số liệu tại ô.
- Trọng số được tính bằng quy tắc do ô đứng trước quy định tuỳ theo từng bài toán.
- Trọng số phụ thuộc vào ô đứng trước ô đang xét.

**3. Quy tắc ″Đi từ trên xuống dưới ″:**

Từ dòng thứ i bạn có thể đi ngang sang trái hoặc sang phải trên dòng đó và đi xuống dưới dòng thứ (i+1) theo các hướng chéo hoặc thẳng đứng.

**Thuật giải chung**

1. Bước 0: Mô hình hoá:

Nếu bài toán không phải là dạng tối ưu trên một bảng hai chiều, ta phải tìm cách mô hình hoá để đưa nó về dạng này.

2. Bước 1: Xây dựng các quy tắc tính trọng số:

Xin lưu ý rằng điều kiện tối ưu ở đây đã có sẵn ngay từ đầu.

Tuỳ theo dạng của bài toán ta sẽ có các quy tắc tính trọng số khác nhau. Khi đi xem xét với các bài toán cụ thể ta sẽ rõ hơn điều này.

3. Bước 2: Xây dựng quy tắc ″đi ″:

Đôi khi quy tắc đi chưa có sẵn mà phải tự người lập trình đặt ra cho phù hợp với cách mô hình hoá của mình. Vấn đề này thuộc vào tư duy của mỗi người nên rất phong phú và phức tạp.

4. Bước 3: Xây dựng công thức tối ưu:

Đây là bước quan trọng nhất của bài toán. Để xây dựng được công thức, ta cần phải dựa vào các quy tắc đi và tính trọng số.

5. Bước 4: Duyệt phương án tối ưu:

Đây là bước cuối cùng để ghi dữ liệu tìm được ra FILE kết quả.

Bước này tương đối dễ dàng vì trong qúa trình quy hoạch, Chúng ta đã lưu các trạng thái của từng ô đi qua, đa phần là lưu vị trí của ô đứng trước ô này trên đường đi tối ưu.

**Một số bài toán**

Trước khi đi xét các bài toán cụ thể, chúng ta quy ước rằng mảng A[1..M,1..N] là mảng lưu dữ liệu ban đầu. Mảng B[1..M,1..N] là mảng dùng để quy hoạch.

Với những bài toán với dữ liệu đầu vào là các mảng một chiều thì ta sẽ dùng ngay các dữ liệu đó mà không cần xây dựng mảng A.

Các bài toán quen thuộc như bài toán cái túi,bài toán tìm đoạn dãy con đơn điệu dài nhất,bài toán cây xăng,..v…v. ta sẽ không xét đến ở đây nữa.

**1. Bài toán ″Con kiến ″:**

Trên một sân hình chữ nhật MxN, được chia thành các ô vuông đơn vị, mỗi ô chứa một lượng thức ăn. Một con kiến xuất phát từ ô (1,1) muốn đi qua sân để đến dòng thứ M. Con kiến chỉ có thể đi theo một dòng chia nhỏ trên sân ứng với một dòng của bảng chữ nhật hoặc đi theo trên một cột của sân. Hãy chỉ ra đường đi giúp con kiến có được nhiều thức ăn nhất.

FOOD.INP
3 5
(Trong tất cả các bài toán dưới đây, dòng đầu bao giờ cũng là hai giá trị M và N)

FOOD.OUT
45 (lượng thức ăn Max)
(1,1) (2,1) (2,2) (2,3) (3,3)

**Thuật giải**
- Bước 0: Bỏ qua vì đây là bài toán đúng dạng
- Bước 1: Trọng số ở đây là lượng thức ăn trên mỗi ô.
- Bước 2: Quy tắc đi:

Bước 3: Công thức quy hoạch
B[i,j] là lượng thức ăn lớn nhất đi từ ô (1,1) đến ô (i,j)
B[1,j] = A[1,j] với j = 1..N
B[i,1] = A[i,1]+B[i-1,1] với i = 2..M
B[i,j] =Max{B[i-1,j],B[i,j-1]} + A[i,j] với i = 2..M và j = 2..N

**2. Bài toán "Sa mạc ":**
Một bãi sa mạc có dạng hình chữ nhật MxN. Mỗi ô vuông đơn vị trên sa mạc có một độ cao nào đó. Một người muốn đi từ bờ đầu này sang bờ cuối cùng bên kia. Người đó chỉ có thể đi từ ô đang đứng tới một ô mới theo hướng thẳng đứng chéo trái hoặc chéo phải. Giả thiết rằng người đó không được vượt ra hai mép trái và phải của sa mạc.
Hãy tìm đường đi sao cho người đó phải vượt qua quãng đường ngắn nhất. Mỗi lần đi từ một ô sang ô mới tiếp theo người đó phải đi hết quãng đường bằng độ chênh cao giữa hai ô đó.

SAMAC.INP

SAMAC.OUT
12 (Quãng đường Min)
(1,3) (2,4) (3,3) (4,2) (5,2)
**Thuật giải** -Bước 0: Bỏ qua
- Bước 1: Trọng số là độ chênh cao giữa hai ô liên tiếp.
- Bước 2: Quy tắc đi.

- Bước 3: Công thức tối ưu:
B[i,j] là quãng đường nhỏ nhất đi từ bờ đầu tiên đến ô (i,j).
B[1,j] = 0 với j = 1..N
B[i,1] = Min { B[i-1,1] + abs(A[i,1] - A[i-1,1]);
B[i-1,2] + abs(A[i,1] - A[i-1,2])}
Với i = 2..M
B[i,j] = Min { B[i-1,j -1] + abs(A[i,j] - A[i-1,j -1]);
B[i-1,j] + abs(A[i,j] - A[i-1,j]);
B[i-1,j+1] + abs(A[i,j] - A[i-1,j+1])}

Với i = 2..M, j = 2..N-1
B[i,N] = Min { B[i-1,N] + abs(A[i,N] - A[i-1,N]);
B[i-1,N-1] + abs(A[i,N] - A[i-1,N-1])}
Với i = 2..M.

**3. Bài toán ″Quầy bán hàng ″:**

Một siêu thị có M gian hàng, mỗi gian hàng gồm N ngăn chứa, mỗi ngăn chứa được bố trí ở mỗi phòng. Giám đốc siêu thị quyết định mở một đợt khuyến mãi cho khách hàng với các quy tắc sau:
Mỗi gian hàng được bố trí trên từng tấng tương ứng từ tầng 1 đến M. Mỗi tầng có N thang máy đi lên ứng với mỗi phòng.
Một khách hàng có thể mua sản phẩm tại một gian hàng nhưng chỉ có thể đi theo một hướng (không được mua xong rồi quay trở lại nơi đã mua).
Khách hàng có thể đi thang máy lên tầng tiếp theo, nhưng phải mua ít nhất tại một ngăn chứa ở tầng đó thì mới được phép đi lên tầng trên nữa.
Khách hàng mua hàng tại một ngăn chứa. Mỗi ngăn chứa quy định một số lượng hàng mà người khách buộc phải mua khi đến ngăn chứa đó.
Nếu độ chênh số lượng hàng giữa hai ngăn chứa liên tiếp của một khách hàng là một số may mắn đã biết trước. Khách hàng đó sẽ được khuyến mãi thêm một số hàng bằng chính số may mắn đó.
Đến tầng thứ M, khách hàng chỉ có thể mua hàng tại duy nhất một ngăn chứa.
Hãy giúp khách hàng lựa chọn điểm xuất phát và hướng đi sao cho mua được nhiều hàng nhất (Kể cả số hàng được khuyến mãi).
Dữ liệu đầu vào cho trong FILE văn bản SHOP.INP có cấu trúc như sau:
Dòng đầu là 3 số M, N, K (1<M, N, K ≤100), K là số các con số may mắn.
Dòng thứ hai ghi K con số may mắn.
M dòng tiếp theo ghi số lượng hàng quy định tại mỗi ngăn chứa. Mỗi dòng gồm N số cách nhau bởi ít nhất một dấu trắng.
Kết quả ghi ra FILE văn bản SHOP.OUT như sau:
Dòng một là số lượng hàng nhiều nhất.
- Dòng hai điểm xuất phát và quá trình mua hàng. Mỗi ngăn chứa đi qua được biểu diễn theo dạng ″(x,y) ″ trong đó (x,y) là vị trí của ngăn chứa.
SHOP.INP

SHOP.OUT
80 (Lượng hàng mua được Max)
(1,3) (1,2) (2,2) (2,1) (3,1) (3,2) (3,3) (3,4) (4,4)

**Thuật giải:**

Bước 0: Bỏ qua.
Bước 1: Trọng số ở đây là số lượng hàng tại các ngăn chứa. Đồng thời khi thoả mãn điều kiện ″khuyến mãi ″ thì trọng số sẽ được tăng thêm số lượng bằng số các con số may mắn (phụ thuộc vào ô đứng trước).
Bước 2: Quy tắc đi
Bước 3: Công thức
B[i,j] là lượng hàng max khi đi từ tầng 1 cho đến ngăn chứa (i,j)
B[0,j]= 0 với j =1..N

B[i,0] = 0 với i =1..M
B[i,j] = Max{B[i,j-1]+KM1, B[i-1,j] + KM2,
B[i-1,k]+ SA[i,u]+KM3}+A[i,j]
Với i =1..M , j =1..(N-1) , k =(j+1)..M , u = j+1..k
KM1 là lượng hàng khuyến mãi nếu Abs(A[i,j]-A[i,j-1]) là con số may mắn.
KM2 là lượng hàng khuyến mãi nếu Abs(A[i,j]-A[i-1,j]) là con số may mắn.
KM3 là tổng số lượng hàng khuyến mãi nếu Abs(A[i,k]-A[i-1,k]) và Abs(A[i,t]-A[i,t+1]) với t = j..(u-1) là các con số may mắn.
B[i,N] = Max{B[i,N-1]+KM1,B[i-1,N]+KM2}+A[i,N]
Với i =1..M
KM1 là lượng hàng khuyến mãi nếu abs(A[i,N]-A[i,N-1]) là con số may mắn.
KM2 là lượng hàng khuyến mãi nếu abs(A[i,N]-A[i-1,N]) là con số may mắn.

**4.Bài toán ″Tách từ ″:**

Đây là bài số 2 trong đề thi OLYMPIC Tin học sinh viên lần thứ XII, 2003, khối không chuyên. Mời các bạn xem đề bài trong số báo 5(44) của Tạp chí ISM.

**Thuật giải:**

Bước 0: Bài toán này thực chất là bài toán ″xâu con lớn nhất ″. Ta xây dựng bảng B[i,j] là xâu con lớn nhất giữa 2 xâu S1[1..i] và S[1..j].Gọi ll = length(S1), l =length(S).
Nhận xét thấy rằng với B[ll,i]=ll với i = ll..l thì ta có một phương án để tách từ. Bằng phương pháp duyệt dựa trên bảng lưu trạng thái qua quá trình quy hoạch, ta xét xem những vị trí còn lại trong xâu S (chưa thuộc S1) có tạo ra xâu S2 không. Nếu thoả mãn điều kiện này thì bài toán đã giải quyết xong. Lưu ý rằng bài toán luôn có lời giải.
Bước 1: Trọng số là 0 hoặc 1 tuỳ xem S1[i] khác hoặc bằng S[j].
Bước 2: Quy tắc đi:

Bước 3: Công thức
B[0,j] = 0 với j =1..l
B[i,0] = 0 với i =0..ll
B[i,j] = min{B[i,j-1],B[i-1,j],B[i-1,j-1]+gt}
với i = 1..ll , j =1..l
gt là 0 hoặc 1 tuỳ theo S1[i] khác hoặc bằng S[j].

**.Bài toán ″Cắt hình chữ nhật ″:**

Đây là bài 146 trong mục ″Đề ra kì này ″. Xin các bạn xem đề bài trong số 6(45) của Tạp chí ISM.
Bước 0: Ta xây dựng bảng B[i,j] là số lần cắt ít nhất để cắt một hình chữ nhật có kích thước [1..i ,1.. j].
Bước 1: Trọng số ở đây là 1 thể hiện một nhát cắt.
Bước 2: Quy tắc đi: Bài toán này có quy tắc đi tương đối phức tạp.

Bước 3: Công thức
B[i,1] = i với i = 1..M
B[1,j] = j với j = 1..N
B[i,j] = min{B[i,q]+B[i,j-q],
B[k,j]+B[i-k,j]}
với q = 1..(j-1) , k =1..(i-1)

**Tổng kết**

Còn rất nhiều bài toán khác có dạng như bài toán tổng quát này nhưng chung quy lại chúng ta đều có thể đưa nó về một dạng chung. Sau đó dựa vào những nguyên tắc giải chung, ta đều có thể giải quyết dễ dàng.
Các dạng bài toán tổng quát này khi dữ liệu cho quá giới hạn khai báo bảng hai chiều đều có thể giải quyết bằng cách quy hoạch liên tục trên 2 mảng một chiều. Sau mỗi bước quy hoạch phải thay đổi 2 mảng này sao cho phù hợp với bước quy hoạch tiếp theo. Cái khó của bài toán có dữ liệu lớn này là việc lưu trữ trạng thái để sau khi quy hoạch toàn bộ ta còn có thể in ra file kết quả ″quá trình đi ″ của phương án tối ưu.
Rất mong nhận được những ý kiến đóng góp quý báu của các bạn đọc ISM. Mọi thắc mắc xin gửi cho tôi theo địa chỉ:

# Phương pháp quy hoạch động

**Phạm Hải Minh**

*Quy hoạch động là một phương pháp rất hay và mạnh của tin học. Nhưng để giải được các bài toán bằng phương pháp quy hoạch động thật chẳng dễ dàng chút nào. Chủ yếu học sinh hiện nay sử dụng quy hoạch động theo kiểu làm từng bài cho nhớ mẫu và áp dụng vào những bài có dạng tương tự.*
*Qua quá trình học tập tôi đã tự rút ra cho mình một số kinh nghiệm về cách giải các bài toán bằng quy hoạch động, xin đưa ra để mọi người cùng tham khảo và góp ý.*

## 1. Lí thuyết:

Phương pháp quy hoạch động gồm 6 bước:
- *Bước 1*: Chia nhỏ bài toán
Lập vectơ P có các thành phần x1,x2,..,xn. Mỗi vectơ P ứng với một bài toán con của bài toán. Ban đầu ta xây dựng P với 1 thành phần duy nhất.

- *Bước 2*: Lập hệ thức quy hoạch động
Xây dựng hàm f(P) là hàm tối ưu của vectơ P (hay hàm tối ưu cho mỗi bài toán con)
f(P) = g(f(P1),f(P2),..,f(Pn))
g có thể là hàm Max,Min hoặc tổng tuỳ yêu cầu của bài toán là tìm Max,Min hay tính tổng.
P gọi là vectơ cha
P1,P2,P3,..,Pn gọi là vectơ con

- *Bước 3*: Kiểm tra
Nếu không xây dựng được hàm f thì thêm tiếp hoặc bỏ đi từng thành phần của vectơ P rồi quay lại bước 2. Nếu được thì làm tiếp bước 4.

- *Bước 4*: Tối ưu hoá hệ thức
Tối ưu vectơ P bằng cách xét từng thành phần x của vectơ P:
Chọn vectơ PBest trong P1,P2,P3,..Pn chỉ khác nhau thành phần x sao cho có thể đưa PBest vào thay P1,P2,P3..,Pn trong hàm g mà không làm thay đổi giá trị của hàm g thì có thể đơn giản thành phần x của vectơ P.

- *Bước 5*: Chọn kiểu quy hoạch động

+ Kiểu 1: Nếu các thành phần của vectơ con P1 luôn ≤ hay ≥ các thành phần của vectơ cha P thì ta có thể dùng các vòng lặp for lồng nhau để cài đặt.
+ Kiểu 2: Nếu vectơ P và vectơ P1 luôn có mối quan hệ cha con một chiều thì ta có thể dùng phương pháp đệ quy có nhớ để cài đặt.
+ Kiểu 3: Nếu vectơ P và vectơ P1 luôn có mối quan hệ cha con hai chiều nhưng không rõ đâu là vectơ cha , đâu là vectơ con vì còn phụ thuộc vào từng bài toán thì ta có thể dùng phương pháp repeat.. until để cài đặt.

- *Bước 6*: Tối ưu hoá bộ nhớ (chỉ dùng cho cài đặt kiểu 1)
Đơn giản vectơ P bằng cách xét từng thành phần x của vectơ P:
Nếu f(P(..,x,.. ))=g(f(P1(..,x1,..)),f(P2(..,x2,..)),..,f(Pn(..,xn,..)))
và x-x1, x-x2,.., x-xn≤T nào đó thì ta chỉ cần đưa vòng lặp của x lên đầu tiên và bỏ x ra khỏi vectơ P và lưu T+1 vectơ P.

**2. Ví dụ:**
*Ví dụ 1*: Bài toán tìm đường đi ngắn nhất:

*** Bài toán**: Cho đồ thị 1 chiều có trọng số được biểu diễn bởi ma trận kề a. Tìm đường đi ngắn nhất từ đỉnh S đến đỉnh T.

*** Cách giải bằng quy hoạch động**:
- Bước 1: Vectơ P (đỉnh hiện tại)
- Bước 2: f(P(u)) = f(P1(v))+a[v,u] (f là đường đi ngắn nhất từ S đến u)
- Bước 3: Đúng.
- Bước 4: Không cần đơn giản.
- Bước 5: Vì P(u) và P1(v) có thể là vectơ cha hay vectơ con tuỳ thuộc bài toán nên ta phải dùng kiểu cài đặt 3.
- Bước 6: Không có.

*Ví dụ 2*: Bài toán cái túi:

*** Bài toán**: Cho n đồ vật, đồ vật thứ i có giá trị V[i] và trọng lượng W[i], số lượng không hạn chế. Ta có một túi đựng được trọng lượng không quá T. Cần chọn các đồ vật để bỏ vào túi sao cho tổng giá trị là lớn nhất.
*** Cách giải bằng quy hoạch động**:
- Bước 1: Vectơ P (trọng lượng hiện tại)
- Bước 2: f(P(m))=Max(f(P1(m-W[i]))+V[i]) (f là tổng giá giá trị lớn nhất khi dùng các vật có tổng trọng lượng m)
- Bước 3: Đúng.
- Bước 4: Không cần đơn giản.
- Bước 5: Vì nếu P1(m1) con của P(m) <=> m1< m nên ta có thể dùng kiểu cài đặt 1.
- Bước 6: Không cần đơn giản.

*Ví dụ 3*: Bài toán chia kẹo:

*** Bài toán**: Cho n gói kẹo, gói kẹo thứ i có a[i] cái kẹo. Cần chọn ra một số gói kẹo sao cho

số kẹo là lớn nhất và không vượt quá W cái.

*** Cách giải bằng quy hoạch động**:
- Bước 1: Vectơ P (tổng số kẹo hiện tại)
- Bước 2 (1): Do không biết những gói kẹo nào đã dùng, những gói kẹo nào chưa dùng nên không thể lập được công thức quy hoạch động
- Bước 3 (1): Chưa đúng nên thêm thành vectơ P (tổng số kẹo, số gói kẹo đã dùng)
- Bước 2 (2): Do vẫn không biết những gói kẹo nào đã dùng, những gói kẹo nào chưa dùng nên không thể lập được công thức quy hoạch động
- Bước 3 (2): Thành phần thêm vào không giải quyết được vấn đề nên bỏ đi và thêm thành phần khác vào P (tổng số kẹo, gói keo cuối cùng đã dùng)
- Bước 2(3): f(P(u,i))=0 nếu không tồn tại P1(u-a[i],j]) sao cho f(P1)=1 (với j<i)
f(P(u,i))=1 nếu tồn tại P1(u-a[i],j) sao cho f(P1)=1 (với j<i)
- Bước 3(3): Đúng.
- Bước 4: Xét thành phần (gói kẹo cuối cùng đã dùng) của vectơ P. Ta thấy chỉ cần chọn P(u,i) trong P1(u,i1),P2(u,i2),..,Pn(u,in) sao cho i min và f(P)=1. Vì thế ta chỉ cần lưu i min này vào mảng IMin và như vậy hàm f không hề bị thay đổi. Hàm f lúc này thành:
f(P(u))=0 nếu không tồn tại P1(u-a[i]) sao cho f(P1)=1 (với IMin[u-a[i]]<i)
f(P(u))=1 nếu tồn tại P1(u-a[i]) sao cho f(P1)=1 (với IMin[u-a[i]]<i)
- Bước 5: Vì nếu P1(u1) là con của P(u) tương đương u1<u nên ta có thể dùng kiểu cài đặt 1.

*Ghi chú:* Có chương trình mẫu của các ví dụ kèm theo.

**3. Bài tập:**

*Bài tập 1:* Trên đoạn đường AB dài n km cần đặt k trạm xăng 1 tại A, 1 tại B và sao cho khoảng cách giữa các trạm xăng sau không lớn hơn khoảng cách giữa các trạm xăng trước. Tính số cách đặt các trạm xăng.
**Input:**
n k (n≤100,k≤n)
**Output:**
P số cách đặt các trạm.
*Ví dụ:*
**Input:**
4 3
**Output:**
2

**Bài tập 2:** Cho ma trận n*m chỉ gồm 0 và 1.Hỏi cần ít nhất bao nhiêu nhát cắt thẳng (mỗi nhát cắt thẳng phải chia ma trận ra làm 2 phần) để chia ma trận thành những hình chữ nhật chỉ gồm 0 hoặc 1.
**Input:**
n m (n,m≤20)
ma trận n*m
**Output:**
P số nhát cắt ít nhất.
*Ví dụ:*
**Input:**

3 3
1 1 1
0 0 1
0 0 1
**Output:**
3
**4. Mở rộng:**
Với phương pháp trên nếu không thể giải được bài toán bằng quy hoạch động do khó khăn trong dữ liệu hay trong việc lập hệ thức, ta có thể nhanh chóng chuyển sang một thuật toán duyệt hoặc tham lam. Nếu hệ thức đúng đắn nhưng không thể lưu trữ ta có thể sử dụng thuật toán duyệt bằng cách đệ qui (bỏ đi phần có nhớ ) sẽ vẫn cho kết quả đúng mặc dù có chậm hơn. Nếu không thể lập được hệ thức đúng ta vẫn có thể cài đặt như một thuật toán tham lam cho kết quả gần đúng.
Hi vọng qua bài viết này các bạn có thể tự rút ra những kinh nghiệm riêng cho bản thân.
Chúc các bạn thành công với phương pháp quy hoạch động.

## Thuật toán Dijkstra trên cấu trúc Heap

**Trần Đỗ Hùng**

**1.** ***Nhắc lại thuật toán Dijkstra tìm đường đi ngắn nhất***
**Bài toán:** Cho đồ thị có hướng với trọng số các cung (i,j) là C[i,j] không âm, tìm đường đi ngắn nhất từ đỉnh s đến đỉnh t.
**Thuật toán Dijkstra:**
*Bước 1- Khởi trị:*
- Khởi trị nhãn đường đi ngắn nhất từ đỉnh s tới đỉnh i là d[i]:= C[s,i] (nếu không có đường đi trực tiếp từ s đến i thì C[s,i] bằng vô cùng). Lưu lại đỉnh trước khi tới i trên hành trình ngắn nhất là Tr[i] := s
- Khởi trị nhãn đỉnh s là d[s] =0
- Đánh dấu mọi đỉnh i là tự do (nhãn d[i] chưa tối ưu): DX[i]:=false
*Bước 2 (vòng lặp vô hạn):*
- Tìm đỉnh $i_0$ tự do có nhãn d[$i_0$] nhỏ nhất.
- Nếu không tìm được $i_0$ ($i_0$ =0) hoặc $i_0$ =t thì thoát khỏi vòng lặp còn không thì
+ Đánh dấu $i_0$ đã được cố định nhãn DX[$i_0$]:=True (gọi $i_0$ là đỉnh được cố định nhãn)
+ Sửa nhãn cho các đỉnh j tự do kề với $i_0$ theo công thức d[j] = Min{d[j], d[$i_0$]+C[$i_0$,j] và ghi lưu lại đỉnh trước j là $i_0$: Tr[j]:= $i_0$
*Bước 3 &minus Tìm và ghi kết quả:*
Dựa vào giá trị d[t] và mảng Tr để kết luận thích hợp

***2. Cấu trúc Heap và một số phép xử lí trên Heap***
a) *Mô tả Heap:* Heap được mô tả như một cây nhị phân có cấu trúc sao cho giá trị khoá ở mỗi nút không vượt quá giá trị khoá của hai nút con của nó (suy ra giá trị khoá tại gốc Heap là nhỏ nhất).
*b) Hai phép xử lí trên Heap*
- Phép cập nhật Heap
Vấn đề: Giả sử nút v có giá trị khoá nhỏ đi, cần chuyển nút v đến vị trí mới trên Heap để bảo toàn cấu trúc Heap

Giải quyết:
+ Nếu nút v chưa có trong Heap thì tạo thêm nút v thành nút cuối cùng của Heap (*hình 1*)
+ Chuyển nút v từ vị trí hiện tại đến vị trí thích hợp bằng cách tìm đường đi ngược từ vị trí hiện tại của v về phía gốc qua các nút cha có giá trị khoá lớn hơn giá trị khoá của v. Trên đường đi ấy dồn nút cha xuống nút con, nút cha cuối cùng chính là vị trí mới của nút v (*hình 2*).
*Chú ý*: trên cây nhị phân, nếu đánh số các nút từ gốc đến lá và từ con trái sang con phải thì dễ thấy: khi biết số hiệu của nút cha là i có thể suy ra số hiệu hai nút con là 2*i và 2*i+1, ngược lại số hiệu nút con là j thì số hiệu nút cha là j div 2.

- Phép loại bỏ gốc của Heap
Vấn đề: Giả sử cần loại bỏ nút gốc khỏi Heap, hãy sắp xếp lại Heap (gọi là phép vun đống)
Giải quyết:
+ Tìm đường đi từ gốc về phía lá, đi qua các nút con có giá trị khoá nhỏ hơn trong hai nút con cho đến khi gặp lá.
+ Trên dọc đường đi ấy, kéo nút con lên vị trí nút cha của nó.
Ví dụ trong hình vẽ 2 nếu bỏ nút gốc có khoá bằng 1, ta sẽ kéo nút con lên vị trí nút cha trên đường đi qua các nút có giá trị khoá là 1, 2, 6, 8 và Heap mới như hình 3

***3. Thuật toán Dijkstra tổ chức trên cấu trúc Heap (tạm kí hiệu là Dijkstra_Heap)***
Tổ chức Heap: Heap gồm các nút là các đỉnh i tự do (chưa cố định nhãn đường đi ngắn nhất), với khoá là nhãn đường đi ngắn nhất từ s đến i là d[i]. Nút gốc chính là đỉnh tự do có nhãn d[i] nhỏ nhất. Mỗi lần lấy nút gốc ra để cố định nhãn của nó và sửa nhãn cho các đỉnh tự do khác thì phải thức hiện hai loại xử lí Heap đã nêu (phép cập nhật và phép loại bỏ gốc).

Vậy thuật toán Dijkstra tổ chức trên Heap như sau:
Cập nhật nút 1 của Heap (tương ứng với nút s có giá trị khoá bằng 0)
Vòng lặp cho đến khi Heap rỗng (không còn nút nào)
Begin
+ Lấy đỉnh u tại nút gốc của Heap (phép loại bỏ gốc Heap)
+ Nếu u= t thì thoát khỏi vòng lặp
+ Đánh dấu u là đỉnh đã được cố định nhãn
+ Duyệt danh sách cung kề tìm các cung có đỉnh đầu bằng u, đỉnh cuối là v
Nếu v là đỉnh tự do và d[v] > d[u] + khoảng cách (u,v) thì
Begin
Sửa nhãn cho v và ghi nhận đỉnh trước v là u
Trên Heap, cập nhật lại nút tương ứng với đỉnh v.
End;
End;

***4. Đánh giá***
+ Thuật toán Dijkstra tổ chức như nêu ở mục 1. Có độ phức tạp thuật toán là $O(N^2)$, nên không thể thực hiện trên đồ thị có nhiều đỉnh.
+ Các phép xử lí Heap đã nêu (cập nhật Heap và loại bỏ gốc Heap) cần thực hiện không quá 2.lgM phép so sánh (nếu Heap có M nút). Số M tối đa là N (số đỉnh của đồ thị) và ngày càng

nhỏ dần (tới 0). Ngoài ra, nếu đồ thị thưa (số cung ít) thì thao tác tìm đỉnh v kề với đỉnh u là không đáng kể khi ta tổ chức danh sách các cung kề này theo từng đoạn có đỉnh đầu giống nhau (dạng Forward Star). Do đó trên đồ thị thưa, độ phức tạp của Dijkstra_Heap có thể đạt tới O(N. k.lgN) trong đó k không đáng kể so với N
+ *Kết luận*: Trên đồ thị nhiều đỉnh ít cung thì Dijkstra_Heap là thực hiện được trong thời gian có thể chấp nhận.

***5. Chương trình***

```
uses crt;
const maxN = 5001;
maxM = 10001;
maxC = 1000000000;
fi = &rquo;minpath.in&rquo;;
fo = &rquo;minpath.out&rquo;;

type k1 = array[1..maxM] of integer;
k2 = array[1..maxM] of longint;

k3 = array[1..maxN] of integer;
k4 = array[1..maxN] of longint;
k5 = array[1..maxN] of boolean;

var ke : ^k1; {danh sách đỉnh kề}
c : ^k2; {trọng số cung tương ứng với danh sách kề}
p : ^k3; 1 {vị trí đỉnh kề trong danh sách kề}
d : k4; {nhãn đường đi ngắn nhất trong thuật toán Dijkstra}
tr : k3; {lưu đỉnh trước của các đỉnh trong hành trình ngắn nhất }
dx : k5; {đánh dấu nhãn đã cố định, không sửa nữă}
h, {heap (Đống)}
sh : k3; {số hiệu của nút trong heap}
n,m,s,t, {số đỉnh, số cạnh, đỉnh xuất phát và đỉnh đích}
shmax : integer; {số nút max trên heap}

procedure doc_inp;
var i,u,v,x : integer;
f : text;
begin
assign(f,fi);

{Đọc file input lần thứ nhất}

reset(f);
readln(f,n,m,s,t);
new(p);
new(ke);
```

```
new(c);
fillchar(p^,sizeof(p^),0);
for i:=1 to m do
begin
readln(f,u);
inc(p^[u]); {p^[u] số lượng đỉnh kề với đỉnh u}
end;
for i:=2 to n do
p^[i] := p^[i] + p^[i-1]; {p[i]^ dùng để xây dựng chỉ số của mảng kê}
close(f); {p[i]^ là vị trí cuối cùng của đỉnh kề với đỉnh i trong mảng kê}

{Đọc file input lần thứ hai}
reset(f);
readln(f);
for i:=1 to m do
begin
readln(f,u,v,x);
kê[p^[u]] := v; {xác nhận kề với đỉnh u là đỉnh v}
c^[p^[u]] := x; {xác nhận trọng số của cung (u,v) là x}
dec(p^[u]); {chuyển về vị trí của đỉnh kề tiếp theo của u}
end;
p^[n+1] := m; {hàng rào}
close(f);
end;

procedure khoitri;
var i : integer;
begin
for i:=1 to n do d[i] := maxC; {nhãn độ dài đường đi ngắn nhất từ s tới i là vô cùng}
d[s] := 0; {nhãn độ dài đường đi ngắn nhất từ s tới s là 0}
fillchar(dx,sizeof(dx),False); {khởi trị mảng đánh dấu: mọi đỉnh chưa cố định nhãn }
fillchar(sh,sizeof(sh),0); {khởi trị số hiệu các nút của Heap là 0}
shmax := 0; {khởi trị số nút của heap là 0}
end;

procedure capnhat(v : integer);
{đỉnh v vừa nhận giá trị mới là d[v], do đó cần xếp lại vị trí của đỉnh v trong heap, bảo đảm
tính chất heap}

var cha,con : integer;
begin
con := sh[v]; {con là số hiệu nút hiện tại của v}
if con=0 then {v chưa có trong heap, thì bổ sung vào nút cuối cùng của heap}
begin
inc(shmax);
con := shmax;
```

```
end;
cha := con div 2; {cha là số hiệu hiện tại của nút cha của nút v hiện tại}
while (cha>0) and (d[h[cha]] > d[v]) do
{nếu nhãn của nút cha (có số hiệu là cha) lớn hơn nhãn của nút v thì đưa dần nút v về phía
gốc tới vị trí thoả mãn điều kiện của heap bằng cách: kéo nút cha xuống vị trí của nút con của
nó }
begin
h[con] := h[cha];
sh[h[con]] := con;
con := cha;
cha := con div 2;
end;
h[con] := v; {nút con cuối cùng trong quá trình "kéo xuống" nêu trên, là vị trí mới của v}
sh[v] := con;
end;

function lay: integer;
{lấy khỏi heap đỉnh gốc, vun lại heap để hai cây con hợp thành heap mới}
var r,c,v : integer;
begin
lay := h[1]; {lấy ra nút gốc là nút có nhãn nhỏ nhất trong các nút chưa cố định nhãn}
v := h[shmax]; {v: đỉnh cuối cùng của heap}
dec(shmax); {sau khi loại đỉnh gốc, số nút của heap giảm đi 1}
r := 1; {bắt đầu vun từ nút gốc}
while r*2 <= shmax do {quá trình vun heap}
begin
c := r*2; {số hiệu nút con trái của r}
if (c
inc(c); {so sánh nhãn của hai nút con, chọn c là con có nhãn nhỏ hơn}
if d[v]<=d[h[c]] then break; { dừng khi nhãn v không vượt quá nhãn hai nút con}
h[r] := h[c]; {chuyển nút có số hiệu là con lên nút có số hiệu là cha}
sh[h[r]] := r; {xác nhận lại số hiệu trong heap của nút mới chuyển lên}
r := c; {xác nhận cha mới để quá trình lặp lại}
end;
h[r] := v; {đỉnh v được đặt vào vị trí r cuối cùng để bảo đảm điều kiện của heap}
sh[v] := r; {xác nhận lại số hiệu của nút v trong heap}
end;

procedure dijkstra;
var i,u,j,v,min : integer;
begin
capnhat(1); {tạo nút thứ nhất cho heap}
repeat
u := lay; {u: đỉnh chưa cố định nhãn, có nhãn nhỏ nhất}
if u=t then break; {tới đích thì dừng}
dx[u] := True; {đánh dấu u được cố định nhãn}
```

```
for j:= p^[u]+1 to p^[u1] do {j: chỉ số trong mảng ke, của các đỉnh kề với u}
begin
v := kê[j]; {v kề với u}
if (not dx[v]) and (d[v]>d[u]+c^[j]) then {điều kiện sửa nhãn v}
begin
d[v] := d[u] + c^[j]; {sửa lại nhãn của v}
tr[v] := u; {ghi nhận lại đỉnh trước của v là u}
capnhat(v); {cập nhật lại v trong heap để bảo đảm cấu trúc heap }
end;
end;
until shmax = 0; {dừng khi không còn đỉnh tự do (số nút của heap bằng 0)}
end;

procedure inkq;
var f : text; i,j : integer;
kq : k3;
begin
assign(f,fo);
rewrite(f);
if d[t]=maxc then
writeln(f,-1) {ghi kết quả: vô nghiệm}
else
begin
writeln(f,d[t]); {ghi độ dài đường đi ngắn nhất từ đỉnh s đến đỉnh t vào file output}
i := 0;
while t<>s do {lần ngược các đỉnh liên tiếp của hành trình ngắn nhất lưu vào mảng kq}
begin
inc(i);
kq[i] := t;
t := tr[t];
end;
inc(i);
kq[i] := s;
for j:=i downto 1 do write(f,kq[j],' '); {ghi hành trình vào file output}
end;
close(f);
end;

BEGIN
doc_inp;
khoitri;
dijkstra;
inkq;
END.
```

**Cách khác để tính giá trị của một biểu thức**

**Quắch Nguyễn**

Tính giá trị của một biểu thức là một bài toán không khó. Có nhiều cách để giải quyết bài toán này chẳn hạn như: đưa biểu thức số học theo ký pháp hậu tố và tiền tố, hoặc đưa về dạng cây biểu thức…Tuy nhiên ở đây tôi muốn đưa ra một phương pháp giải khác với những phương pháp trên nhằm mục đích để cho các bạn đọc gần xa tham khảo thêm và cũng rất mong nhận được ý kiến phản hồi của bạn đọc gần xa.

**Phương pháp giải**

***1.*** Biểu thức chỉ là một số thực thì giá trị của biểu (gtbt) thức bằng chính số đó

***2.*** Biểu thức chỉ chứa phép toán '*' thì gtbt = gtbt(trước phép nhân đầu tiên ) * gtbt(dãy sau phép nhân đầu tiên)

***3.*** Biểu thức có chứa 2 phép toán '+, *' thì gtbt = gtbt(dãy trước dấu cộng đầu tiên)+ gtbt(dãy sau dấu cộng đầu tiên)

```
function gtbt(s:string):real;
begin
if isnumber(s) then gtbt:=strtonumber(s)
else
if all_mull_ope(s) then tinh:=gtbt(head(s,'*'))*gtbt(tail(s,'*'))
else
gtbt:=gtbt(head(s,'+'))+ gtbt(tail(s,'+'));
end;
```

***4.*** Biểu thức có chứa 3 phép toán '+ , −, *'trong trường hợp này ta chèn thêm dấu '+' vào trước những dấu '−' (ngoại trừ dấu ' −' đứng tại ví trí thứ 1 : s1=' −' ) sau đó vẫn tính như (3)

***5***. Biểu thức có chứa 4 phép toán ' +, −, *, /' tương tự như (4) và chèn thêm dấu ' *' trước những dấu ' /' sau đó tính như (3).

***Các hàm:***

```
Function Isnumber(s:string):boolean; {hàm này trả về giá trị đúng nếu s là một số }
var r:real; code:word;
begin
val(s,r,code); Isnumber:=(code=0)or (s[1]='/');
end;

Function Strtonumber(s:string):real; {hàm này trả về một số tương ứng với s hoặc 1/s nếu
s[1] ='/'}
var r:real; code:word;
begin
val(s,r,code);
if code 0 then
begin val(copy(s,2,length(s)-1),r,code); r:=1/r; end;
Strtonumber:=r;
end;

Function All_mull_ope(s:string):boolean; { hàm trả về giá trị true khi trong s có ' * ' và không
có '+' }
begin
All_mull_ope:= (pos('*',s)>0) and (pos('+',s)=0);
```

```
end;

Function Head(s:string;c:char):string; { hàm trả về chuỗi đứng trước dấu c đầu tiên }
begin
Head:=copy(s,1,pos(c,s)-1);
end;

Function Tail(s:string;c:char):string; { hàm trả về chuỗi đứng sau dấu c đầu tiên }
begin
Tail:=copy(s,pos(c,s)+1,length(s)-pos(c,s));
end;

Function Insert_ope(s:string):string; { hàm xử lí chuỗi trước khi tính }
var i:byte;
begin
while pos('--',s)>0 do begin i:= pos('--',s); delete(s,i,1); s[i]:='+'; {đổi 2 dấu '-' thành
dấu'+'}
while pos(' ',s)>0 do delete(s,pos(' ',s),1);
i:=1;
repeat
inc(i);
case s[i] of '-': begin insert('+',s,i); inc(i); end;
'/': begin insert('*',s,i); inc(i); end;
end;
until i>= length(s);
Insert_ope:=s;
end;

Function Tinh_gia_tri_bieu_thuc_khong_dau_ngoac(s:string) : real;
var s:string;
begin
Tinh_gia_tri_bieu_thuc_khong_dau_ngoac:= gtbt(insert_ope(s));
end;
```

***6.*** Biểu thức có chứa 4 phép toán trên và dấu ngoặc:
Trước tiên ta đi tìm dấu ngoặc đóng đầu tiên tính từ trái sang phải trong b iểu thức. Nếu tồn tại thì ta lui về trái tìm dấu ngoặc mở đầu tiên mà ta gặp (tính từ dấu ngoặc đóng đầu tiên). Sau đó trích biểu thức con trong khoảng trên ra tính giá trị của biểu thức con đó (5) và đưa giá trị ấy về dạng chuỗi ghép vào đúng vị trí của nó ban đầu trong biểu thức mẹ. Lặp lại bước 6. Ngược lại nếu không tồn tại dấu ngoặc đóng thì biểu thức đã cho là biểu thức không có chứa dấu ngoặc (5)

```
Function Numbertostr(r:real):string;
var s:string;
begin
str(r:0:5,s);
```

```
Numbertotr :=s;
end;
Function Tinh_gia_tri_bieu_thuc_co_ngoac(s:string):real;
begin
i:=pos(')',s);
while i>0 do
begin
repeat
dec(i);
until s[i]='(';
s1:=copy(s,i+1,pos( ')',s)-i-1);
delete(s,i,pos( ')',s) -i+1;
insert(numbertostr(Tinh_gia_tri_bieu_thuc_khong_dau_ngoac(s1)),s,i);
i:=pos( '-',s);
end;
Tinh_gia_tri_bieu_thuc_khong_co_ngoac:=Tinh_gia_tri_bieu_thuc_khong_dau_ngoac(s);
end;
```

***7.***Xử lí lỗi của biểu thức: ta chỉ cần sửa lại chút ít các đoạn mã lệnh trên thì ta sẽ phát hiện được tất các các lỗi mà biểu thức đó có chẵn hạn như: sai cú pháp, xuất hiện kí tự lạ, chia cho 0…

***8.*** Mở rộng phép toán: với phương pháp nêu trên các bạn dễ dàng bổ sung thêm một số phép toán vào biểu thức như: ^, log, Ln, sin, cos,…

***9.*** Tính biểu thức dài: ta có thể dùng mảng để lưu trữ biểu thức thay vì chuỗi.

Trong quá trình viết bài có thể có xuất hiện một số sai sót. Mong bạn đọc thông cảm và tự khắc phục (nếu được) hoặc liên hệ với tác giả của bài viết để bổ sung.

## Dijtra - thuật toán tốt để giải các bài toán tối ưu

**Đỗ Giang Lâm**

*Bài toán tối ưu là một bài toán muôn thuở trong tin học bởi nó có ứng dụng thực tế cao. Ví dụ như làm thế nào để hoàn thành một công viêc nào đó với chi phí ít nhất, hay đi như thế nào để đến đích sớm nhất, vv... Cùng với sự trợ giúp của máy tính và các thuật toán hiệu quả đã giúp chúng ta giải quyết bài toán tối ưu một cách dễ dàng và chính xác. Một trong số các thuật toán hiệu quả đó là thuật toán Dijkstra- thuật toán tìm đường đi ngắn nhất trên đồ thị có trọng số không âm.*

Chắc hẳn các bạn đều không xa lạ gì với thuật toán Dijkstra. Đây là một cải tiến từ thuật toán Ford_Bellman. Trong truờng hợp trọng số trên các cung là không âm, thuật toán do Dijkstra đề nghị để giải bài toán tìm đường đi ngắn nhất từ đỉnh s đến các đỉnh còn lại của đồ thị làm việc hiểu quả hơn nhiều so với thuật toán Ford_Bellman. Thuật toán được xây dựng trên cơ sở gán cho các đỉnh các nhãn tạm thời. Nhãn của một đỉnh cho biết cận trên của độ dài đường đi ngắn nhất từ s đến nó. Các nhãn này sẽ được biến đổi theo một thủ tục lặp, mà ở mỗi một bước lặp có một nhãn tạm thời trở thành nhãn cố định. Nếu nhãn của một đỉnh nào đó trở

thành nhãn cố định thì nó sẽ cho ta không phải là cận trên mà là đường đi ngắn nhất từ đỉnh s đến nó. Thuật toán được mô tả cụ thể như sau:

**Đầu vào:** Đồ thị có hướng G = (V,E) với n đỉnh, s thuộc V là đỉnh xuất phát, ma trận trọng số a[u,v], a[u,v] ≥ 0, u,v thuộc V.
**Đầu ra:** Độ dài đường đi ngắn nhất từ s đến tất các đỉnh còn lại: d[v], v thuộc V.
truoc[v], v thuộc V, ghi nhận đỉnh đi trước v trong đường đi ngắn nhất từ s đến v

```
Procedure Dijkstra;
Begin
for v thuộc V do begin d[v] := a[s,v]; truoc[v] := s; end; (* Khởi tạo *)
d[s] := 0; T := V {s} (* T là tập các đỉnh có nhãn tạm thời *)
(* Bước lặp *)
while T ≠ ∅ do
Begin
Tìm đỉnh u thuộc T thoả mãn d[u] = Min {d[z] : z thuộc T}
T := T{u} (* Cố định nhãn của đỉnh u *)
for v thuộc T do (* Cập nhật lại nhãn cho các đỉnh trong T *)
if d[v] > d[u] + a[u,v] then
begin d[v] := d[u] + a[u,v]; truoc[v] := u; end;

end;
end;
```

Thuật toán Dijkstra tìm được đường đi ngắn nhất trên đồ thị sau thời gian cỡ $O(n^2)$. Nếu chỉ cần tìm đường đi ngắn nhất từ s đến một đỉnh t nào đó thì có thể kết thúc thuật toán khi đỉnh t trở thành đỉnh có nhãn cố định. Việc cài đặt thuật toán chắc không có gì khó khăn. Điều khó nhất là trong các bài toán cụ thể ta phải đưa được nó về mô hình đồ thị như thế nào, đỉnh là gì, 2 đỉnh có cung nối khi nào, trọng số là gì, đỉnh xuất phát là gì, đỉnh kết thúc là gì, vv..Từ đó mới áp dụng thuật toán cơ bản để giải. Sau đây là một số bài toán hay ứng dụng thuật toán Dijkstra để giải.

**Bài 1: Chuyển Hàng**
Bản đồ một kho hàng hình chữ nhật kích thước mxn được chia thành các ô vuông đơn vị (m hàng, n cột: các hàng đánh số từ trên xuống dưới, các cột được đánh số từ trái qua phải). Trên các ô vuông của bản đồ có một số ký hiệu:
- Các ký hiệu # đánh dấu các ô đã có một kiện hàng xếp sẵn.
- Một ký hiệu *: Đánh dấu ô đang có một rôbốt.
- Một ký hiệu $: Đánh dấu ô chứa kiện hàng cần xếp.
- Một ký hiệu @: Đánh dấu vị trí mà cần phải xếp kiện hàng vào $ vào ô đó.
- Các ký hiệu dấu chấm ".": Cho biết ô đó trống.

Tại môt thời điểm, rô bốt có thể thực hiện một trong số 6 động tác ký hiệu là:
- L, R, U, D: Tương ứng với phép di chuyển của rô bốt trên bản đồ: sang trái, sang phải, lên trên, xuống dưới. Thực hiện một phép di chuyển mất 1 công
- +, − : Chỉ thực hiện khi rôbốt đứng ở ô bên cạnh kiện hàng $. Khi thực hiện thao tác +, rôbốt đứng yên và đẩy kiện hàng $ làm kiện hàng này trượt theo hướng đẩy, đến khi chạm một kiện hàng khác hoặc tường nhà kho thì dừng lại. Khi thực hiện thao tác − , rô bốt kéo

kiện hàng $ về phía mình và lùi lại 1 ô theo hướng kéo. Thực hiện thao tác đẩy hoặc kéo mất C công. Rô bốt chỉ được di chuyển vào ô không chứa kiện hàng của kho.

Hãy tìm cách hướng dẫn rôbốt thực hiện các thao tác để đưa kiện hàng $ về vị trí @ sao cho số công phải dùng là ít nhất.
**Dữ liệu:** Vào từ file văn bản CARGO.INP
- Dòng 1: Ghi ba số nguyên dương m, n, C ( m, n ≤ 100; C ≤ 100)
- m dòng tiếp theo, dòng thứ i ghi đủ n ký kiệu trên hàng i của bản đồ theo đúng thứ tự trái qua phải.
Các ký hiệu được ghi liền nhau.
**Kết quả:** Ghi ra file văn bản CARGO.OUT
- Dòng 1: Ghi số công cần thực hiện
- Dòng 2: Một dãy liên tiếp các ký tự thuộc {L, R, U, D, +, -} thể hiện các động tác cần thực hiện của rô bốt.
Rằng buộc: Luôn có phương án thực hiện yêu cầu đề bài.
**Ví dụ:**

***Phân tích:***
*Thuật toán:* Ta sẽ dùng thuật toán Dijkstra để giải bài toán này.
* **Mô hình đồ thị:**
Mỗi đỉnh của đồ thị ở đây gồm 3 trường để phân biệt với các đỉnh khác:
- i: Tọa độ dòng của kiện hàng (i = 1..m)
- j: Tọa độ cột của kiện hàng (j = 1..n)
- h: Hướng của rô bốt đứng cạnh kiện hàng so với kiện hàng (h = 1..4: Bắc, Đông, Nam, Tây).

Bạn có thể quan niệm mỗi đỉnh là (i,j,u,v): trong đó i,j: tọa độ của kiện hàng; u,v: tọa độ của rôbốt đứng cạnh kiện hàng. Nhưng làm thế sẽ rất lãng phí bộ nhớ và không chạy hết được dữ liệu. Ta chỉ cần biết hướng h của rôbốt so với kiện hàng là có thể tính được tọa độ của rôbốt bằng cách dùng 2 hằng mảng lưu các số ra:
dx : array[1..4] of integer = (-1,0,1,0)
dy : array[1..4] of integer = (0,1,0,-1)
Khi đó, tọa độ(u,v) của rôbốt sẽ là : u := i + dx[h]; v := j + dy[h];

- Hai đỉnh (i1,j1,h1) và (i2,j2,h2) được gọi là kề nhau nếu qua 1 trong 2 thao tác + hoặc - kiện hàng được rôbốt đẩy hoặc kéo từ ô (i1, j1) đến ô (i2, j2) và rôbốt có thể di chuyển được từ ô (u1,v1) đến ô (u2,v2) ( u1 = i1+dx[h1]; v1=j1+dy[h1]; u2=i2+dx[h2]; v2= j2+dy[h2]). Tất nhiên các ô (i2,j2) và (u2,v2) phải đều không chứa kiện hàng.
- Trọng số giữa 2 đỉnh là C (số công mà rô bốt đẩy kiện hàng từ ô (i1,j1) đến ô (i2,j2) ) cộng với công để rô bốt di chuyển từ ô (u1,v1) đến ô (u2,v2).
Giả sử kiện hàng cần xếp đang ở ô (is,js) và hướng của rôbốt đứng cạnh kiện hàng là hs và ô cần xếp kiện hàng vào là ô (ie, je). Khi đó, ta sẽ dùng thuật toán Dijkstra để tìm đường đi ngắn nhất từ đỉnh (is,js,hs) đến đỉnh (ie,je,he) với he thuộc {1..4}.
Mảng d sẽ là 1 mảng 3 chiều: d[i,j,h]: Độ dài đường đi ngắn nhất từ đỉnh xuất phát (is,js,hs) đến đỉnh (i,j,h). Kết quả của bài toán sẽ là d[ie,je,he] với he thuộc {1..4}.
Để ghi nhận phương án ta sẽ dùng 3 mảng 3 chiều tr1, tr2, tr3. Khi ta di từ đỉnh (i1,j1,h1) đến

đỉnh (i2,j2,h2) thì ta sẽ gán: tr1[i2,j2,h2]:= i1; tr2[i2,j2,h2]:= j1; tr3[i2,j2,h2] := h1 để ghi nhận các thông tin: tọa độ dòng, cột, hướng của dỉnh trước đỉnh (i2,j2,h2). Từ 3 mảng này ta có thể dễ dàng lần lại đường đi.

**Bài 2: Hành trình rẻ nhất:**
Thành phố Peace vừa đưa vào áp dụng một hệ thống điện tử tự động tính lệ phí sử dụng lệ phí đường giao thông. Một hệ thống được triển khai để phát hiện xe của bạn rẽ trái, rẽ phải, đi thẳng hoặc quay đầu và mỗi thao tác như vậy phải trả một lệ phí tương ứng. Đó là 1 $ cho mỗi lần rẽ trái, 5$ cho mỗi lần rẽ phải, đi thẳng về phía trước là miễn phí, quay đầu xe là bị cấm, ngoại trừ tình huống ở cuối phố khi không còn có thể đi thẳng, rẽ trái, hoặc rẽ phải được nữa. Trong trường hợp ngoại lệ bắt buộc phải quay đầu xe, bạn phải trả lệ phí 10$. Bạn được mời để thiết kế và đưa ra chỉ dẫn cho người đi xe đi sao cho phải trả lệ phí là ít nhất giữa 2 điểm bất kỳ trong thành phố. Rất may hệ thống đường giao thông của Peace có dạng bàn cờ.
**Ví dụ:**

ở ví dụ trên ký tự '#' để chỉ ra đoạn đường phố, còn ký tự '.' chỉ ra đoạn đường không là đường. Các đoạn đường phố đều có thể đi cả 2 chiều. Ký tự 'E' chỉ ra vị trí xuất phát của xe ô tô có đầu xe hướng về phía Đông, còn ký tự 'F' chỉ ra vị trí kết thúc. Lộ trình phải trả là 8$, trong đó ta phải thực hiên 3 lần rẽ trái và 1 lần rẽ phải để đến đích. Ta có thể thực hiện cách đi rẽ phải 2 lần để đến đích, nhưng cách đó lệ phí phải trả là 10$.

Chiều cao và chiều rộng của bản đồ ít nhất là 4 và không quá 30. Bản đồ có đúng 1 điểm xuất phát và 1 điểm kết thúc. Luôn có đường đi từ điểm xuất phát đến điểm kết thúc. Luôn có một khung gồm toàn ký tự '.' viền bản đồ để không thể vượt ra ngoài bản đồ.
**Dữ liệu:** Vào từ file văn bản ERP.INP gồm các dòng:
- Dòng thứ nhất chứa 2 số nguyên dương h, w theo thứ tự là chiều cao h và chiều rộng của bản đồ.
- Mỗi dòng trong h dòng tiếp theo chứa w ký tự. Mỗi ký tự chỉ là một trong số các ký tự sau:
- '.': vị trí không có đường
- '#': vị trí có đường của bản đồ.
- 'E': vị trí xuất phát, xe hướng đầu về phía Đông.
- 'W': vị trí xuất phát, xe hướng đầu về phía Tây.
- 'N': vị trí xuất phát, xe hướng đầu về phía Bắc.
- 'S': vị trí xuất phát, xe hướng đầu về phía Nam.
- 'F': vị trí kết thúc.
Trong bản đồ có đúng 1 trong 4 ký tự 'E', 'W', 'N', 'S'. Và đúng 1 ký tự F.
Kết quả ghi ra file văn bản ERP.OUT duy nhất một số là lệ phí của lộ trình rẻ nhất đi từ vị trí xuất phát đến vị trí kết thúc.
***Phân tích:***

*Mô hình đồ thị:*
Mỗi đỉnh của đồ thị ở đây gồm 3 trường để phân biệt với các đỉnh khác:
- i: Tọa độ dòng của nút mà ô tô đang đứng( i = 1..h)
- j: Tọa độ cột của nút mà ô tô đang đứng (j = 1..w)
- h: Hướng của đầu ô tô (h=1..4)

Ta sẽ xét xem 1 ô (i,j) có phải là 1 nút không. Để dễ hình dung, ta sẽ quan niệm 1 nút ở đây cũng giống như 1 nút giao thông như trong thực tế. Ta sẽ xây dựng một mảng nut (nut(i,j)=true: ô (i,j) là 1 nút ) như sau:
- Điều kiện cần để ô (i,j) là 1 nút là ô (i,j) đó phải là đường.
- Các ô xuất phát và kết thúc đều là nút.
- Nếu một ô (i,j) có 2 ô kề theo hướng 1 và 3 (hoặc 2 và 4) là đường và 2 ô kề theo hướng 2 và 4 (hoặc 1 và 3) không là đường thì ô (i,j) đó không là 1 nút. Ngược lại ô (i,j) đó là 1 nút.
- Khi ta đang đứng ở ô (i1,j1) và ô tô đang quay đầu về hướng h1 ta sẽ tìm đỉnh kề bằng cách đi theo 4 hướng(1..4).
Khi đi theo hướng h2 ta gặp một ô (i2,j2) là 1 nút. Khi đó đỉnh (i1,j1,h1) và (i2,j2,h2) được gọi là kề nhau. ở mỗi hướng ta sẽ đi cho đến khi gặp ô không là đường.
- Trọng số ở đây sẽ được lưu trữ trong 1 hằng mảng cp(u,v) với ý nghĩa cp(u,v) là chi phí phải trả để ô tô đang quay đầu theo hướng u chuyển sang quay đầu theo hướng v. Khi đó, trọng số giữa 2 đỉnh (i1,j1,h1) và (i2,j2,h2) là cp(h1,h2).
Hằng mảng chi phí như sau:
```
cp :array[1..4,1..4] of integer= ( (0,5,10,1),
(1,0,5,10),
(10,1,0,5),
(5,10,1,0) );
```
- Mảng d(i,j,h) sẽ lưu trữ chi phí ít nhất để đi từ đỉnh xuất phát (is,js,hs) đến đỉnh (i,j,h). Kết quả của bài toán sẽ là d(ie,je,he) với ô (ie,je) là ô kết thúc, he=1..4.

**Bài tập:**
**Bài 1: Đường đi trong mê cung.**
Một mê cung gồm MxN ô vuông (M dòng, N cột, M, N ≤ 100). Mỗi ô vuông có thể có từ 0 đến 4 bức tường bao quanh. Cần phải đi từ bên ngoài vào mê cung, bắt đầu từ phía Tây, qua các ô của mê cung và thoắt ra khỏi mê cung về phía Đông. Chỉ được phép di chuyển theo 4 hướng Đông, Tây, Nam, Bắc và đi qua nơi không có tường chắn.
1. Trong trường hợp có đường đi, hãy tìm đường đi qua ít ô nhất.
2. Trong trường hợp trái lại, hãy tìm cách phá bỏ ít nhất một số bức tường để có đường đi. Nếu có nhiều phương án như vậy, hãy chỉ ra một phương án để đường đi qua ít ô nhất.
Trạng thái của 1 ô được cho bởi một số nguyên trong khoảng 0 đến 15 theo quy tắc: bắt đầu là 0 (Không có tường), cộng thêm 1(nếu có bức tường phía Tây), cộng thêm 2 (nếu có bức tường phía Bắc), cộng thêm 4 (nếu có bức tường phía Đông), cộng thêm 8 (nếu có bức tường phía Nam).

Dữ liệu vào được cho bởi file văn bản MECUNG.INP bao gồm:
- Dòng đầu ghi 2 số M, N
- Các dòng tiếp theo ghi ma trận trạng thái của các ô gồm M dòng, N cột, trong đó giá trị dòng i cột j mô tả trạng thái của ô [i,j].
Các giá trị ghi trên 1 dòng cách nhau ít nhất 1 dấu cách.
Kết quả ghi ra file văn bản MECUNG.OUT:
1. Trường hợp tìm thấy đường đi:
- Dòng đầu ghi sô ô mà đường đi đi qua
- Dòng tiếp theo ghi thông tin về đường đi gồm tọa độ dòng xuất phát, sau đó cách 1 dấu trắng, là xâu kí tự mô tả đường đi( theo hướng) viết liên tiếp nhau, gồm các ký tự D(đông),

B(bắc), N(nam), T(tây).
2. Trường hợp không tìm thấy đường đi:
- Dòng đầu ghi số 0
- Dòng thứ 2 ghi số bức tường phải phá
- Dòng thứ ba ghi số ô mà đường đi đi qua
- Dòng cuối ghi thông tin về đường di theo quy cách giống như trường hợp 1.

**Vidụ 1:**

Các bạn thấy đấy, một bài toán tưởng chừng như rất khó, nhưng sau khi phân tích và đưa nó về mô hình đồ thị ta có thể giải nó dễ dàng bằng thuật toán Dijkstra. Phần cài đặt các bài toán trên, tôi để các bạn tự làm. Nếu muốn có chương trình hoặc có gì không hiểu, xin liên hệ với toà soạn hoặc tôi và tôi sẽ cho bạn thấy sức mạnh của thuật toán Dijkstra! Chúc các bạn thành công

## Kỹ thuật tìm kiếm nhị phân giải một số bài toán tối ưu

**Nguyễn Thanh Tùng**

Có lẽ ai trong chúng ta cũng biết về thuật toán tìm kiếm nhị phân và sự hiệu quả của nó. Sử dụng kỹ thuật tìm kiếm tương tự trong một số bài toán ta cũng đạt được kết quả rất khả quan. Sau đây là một số bài toán như vậy.

**I. Bài toán ví dụ.**

Thông tin về mạng giao thông gồm n thành phố cho bởi mảng A kích thước n?n gồm các phần tử $a_{ij}$ là một giá trị nguyên dương chỉ tải trọng của tuyến đường nối 2 thành phố i,j. Nếu $a_{ij}$ =0 thì giữa i,j không có đường nối.

Tải trọng của một lộ trình từ thành phố s đến thành phố t (có thể đi qua một số thành phố trung gian) là tải trọng của tuyến đường có tải trọng nhỏ nhất thuộc lộ trình.

Cho trước 2 thành phố s và t. Giả thiết luôn có lộ trình từ s đến t, hãy tìm một lộ trình từ s đến t có tải trọng lớn nhất.

**Thuật giải**

Đặt $k_{min} = \min \{a_{ij} : a_{ij} >0\}$; $k_{max} = \max \{a_{ij}: a_{ij} >0\}$

Với k thuộc $k_{min}..k_{max}$, ta xây dựng đồ thị G(k) gồm:

n đỉnh ứng với n thành phố.

2 đỉnh i,j có cạnh nối nếu $a_{ij} \geq k$

Nếu G(k) có một đường đi từ s đến t thì ta nói mạng giao thông có "lộ trình tối thiểu k" (Vì mọi cạnh của G(k) đều có trọng số ≥ k nên nếu G(k) có một đường đi từ s đến t thì đường đi đó có tải trọng ≥k).

Bài toán được chuyển thành việc tìm giá trị k lớn nhất thuộc kmin..kmax sao cho mạng giao thông có "lộ trình tối thiểu k". Khi đó đường đi từ s đến t tìm được chính là đường đi có tải trọng lớn nhất cần tìm.

Ta sử dụng kỹ thuật tìm kiếm nhị phân dựa trên nhận xét: *nếu mạng có "lộ trình tối thiểu p" và k là giá trị lớn nhất để có "lộ trình tối thiểu k" thì k thuộc p..$k_{max}$. Ngược lại (nếu mạng không có "lộ trình tối thiểu p") thì k thuộc $k_{min}..p-1$.*

Thủ tục Search thực hiện việc tìm kiếm nhị phân có dạng như sau:

```
procedure search(x);
begin
```

```
l := kmin; r := kmax;
repeat
k := (l + r) div 2;
check(k);
if ok then l := k else r := k - 1;
until l>=r;
end;
```

Trong đó thủ tục Check (k) thực hiện:

1. Xây dựng đồ thị G(k) như đã mô tả ở trên.
2. Dùng thuật toán DFS để kiểm tra xem có đường đi từ s đến t không. Nếu có thì đặt Ok là true, ngược lại thì Ok là false.

**Chương trình mẫu**

Để bạn đọc tiện theo dõi, tôi xin cung cấp chương trình mẫu của bài toán này. Để xử lí lỗi, một số đoạn chương trình hơi phức tạp so với mẫu trên.

```
program weight;
const
inp = ’weight.inp’;
out = ’weight.out’;
max = 100;
type
mang1 = array[1..max] of integer;
mang2 = array[1..max, 1..max] of LongInt;
var
n,s,t,z : integer;
k,l,r : LongInt;
a : mang2;
đ,kq : mang1;
ok : boolean;
(**********************)
procedure nhap;
var
i,j : integer;
f : text;
begin
assign(f, inp);
reset(f);
readln(f, n, s, t);
for i := 1 to n do begin
for j := 1 to n do read(f,a[i,j]);
readln(f);
end;
close(f);
end;
(**********************)
procedure chbi;
var
```

```
i,j : integer;
begin
l := maxLongInt; r := 0;
for i := 1 to n do
for j := 1 to n do
if a[i,j] > 0 then begin
if l > a[i,j] then l := a[i,j];
if r < a[i,j] then r := a[i,j];
end;
end;
(**********************)
procedure dfs(i : integer);
var
j : integer;
begin
for j := 1 to n do
if (đ[j] = 0) and (a[i,j] > = k) then begin
đ[j] := i;
dfs(j);
end;
end;
(**********************)
procedure check;
begin
fillchar(đ,sizeof(đ),0);
đ[s] := -1;
dfs(s);
if đ[t] = 0 then ok := false
else ok := true;
end;
(**********************)
procedure search;
begin
repeat
k := (l+r) div 2;
check;
if ok then l := k
else r := k-1;
until (l=r) or (l = r-1);
if l = r-1 then begin
k := r;
check;
if ok then exit;
end;
k := l;
check;
```

```
end;
procedure trace;
var
i : integer;
begin
if not ok then exit;
z := 0; i := t;
repeat
inc(z); kq[z] := i;
i := đ[i];
until i = -1;
end;
(**********************)
procedure xuly;
begin
search;
trace;
end;
(**********************)
procedure inkq;
var
f : text;
i : integer;
begin
assign(f, out);
rewrite(f);
writeln(f, k);
for i := z downto 1 do
write(f,' ',kq[i]);
close(f);
end;
(**********************)
begin
nhap;
chbi;
xuly;
inkq;
end.
```

**Nhận xét**

a. **Với kỹ thuật tìm kiếm nhị phân, giải thuật trên chỉ cần thực hiện cỡ $\log_2(k_{max} - k_{min})$ lần kiểm tra (gọi thủ tục check)**. Do hạn chế $a_{ij}$ là nguyên dương $\leq$ maxLongInt nên $k_{max} - k_{min} < 2^{32}$. Thủ tục check sử dụng thuật toán DFS có độ phức tạp tính toán là $O(n^2)$ nên giải thuật có thời gian thực thi cỡ $C.O(n^2)$ với $C < 32$.

b. **Ta không cần phải xây dựng G(k) một cách tường minh** (tính hẳn thành ma trận kề) mà chỉ cần thay biểu thức kiểm tra có cạnh (i,j) không bằng biểu thức $a_{ij} \geq k$ (trong thủ tục DFS).

c. **Giá trị tối ưu là một trong các phần tử của A**. Trong bài này do aij là số nguyên nên ta

xác định được khoảng tìm kiếm là miền nguyên $k_{min}..k_{max}$ và thực hiện việc tìm kiếm nhị phân trên miền đó.

Nếu aij là số thực không thể kĩ thuật tìm kiếm nhị phân không áp dụng được trên miền thực $[k_{min}, k_{max}]$. Để áp dụng được ta phải *sắp xếp tăng dần các phần tử dương của A (tối đa có $n^2$ phần tử) rồi thực hiện tìm kiếm nhị phân trên dãy tăng dần đó*. Khi đó thủ tục search cần thay đổi: l khởi tạo bằng 1, r khởi tạo bằng $n^2$ và thủ tục check được gọi với tham số là d[k]: check(d[k]) trong đó d dãy tăng dần chứa $n^2$ phần tử của A.

Cũng có thể làm thế khi aij là số nguyên, tuy nhiên sẽ không hiệu quả vì sẽ tốn thời gian sắp xếp dãy và tốn không gian lưu trữ dãy đã sắp.

**II. Một số bài toán áp dụng**

**1. Bài toán 1 (đề thi HSGQG năm học 1999-2000)**

Có n công nhân và n công việc. Nếu xếp công nhân i nếu làm việc j thì phải trả tiền công là $a_{ij}$. Hãy tìm một cách xếp mỗi người một việc sao cho tiền công lớn nhất cần trả trong cách xếp việc đó là nhỏ nhất.

**Thuật giải**

Nếu bài toán yêu cầu là tìm cách xếp việc sao cho tổng tiền công phải trả là nhỏ nhất thì đó là bài toán tìm cặp ghép đầy đủ trọng số cực tiểu. Tuy nhiên bài này là tìm cách xếp việc sao cho tiền công lớn nhất là nhỏ nhất. Ta có ý tưởng như sau: tìm số k bé nhất sao cho tồn tại một cách sắp xếp đủ n người, n việc và các yêu cầu về tiền công đều $\leq k$.

Dễ thấy việc tìm kiếm đó có thể thực hiện bằng kĩ thuật tìm kiếm nhị phân, và việc kiểm tra số k có thoả mãn không chính là việc kiểm tra đồ thị 2 phía G(k) có bộ ghép đầy đủ hay không. Đồ thị đồ thị 2 phía G(k) được xác định như sau:

G(k) = (X,Y,E) Trong đó: X là tập n đỉnh ứng với n công nhân, Y là tập n đỉnh ứng với n công việc. Với i thuộc X, j thuộc Y nếu $a_{ij} \geq k$ thì cho (i,j) thuộc E (2 đỉnh i,j chỉ được nối với nhau nếu $a_{ij} \geq k$) .

Nếu k là số nhỏ nhất mà G(k) có bộ ghép đầy đủ thì bộ ghép đó chính là cách xếp việc cần tìm.

Ta cũng có một số bài toán dạng tương tự:

**Bài toán 2. Thời gian hoàn thành**

Có n công nhân và n công việc. Nếu xếp công nhân i nếu làm việc j thì thời gian hoàn thành là $T_{ij}$. Hãy tìm một cách xếp mỗi người một việc sao tất cả các công việc hoàn thành trong thời gian sớm nhất (các công việc được tiến hành song song).

**Bài toán 3. Năng suất dây truyền**

Dây truyền sản xuất có n vị trí và n công nhân (đều được đánh số từ 1..n). $a_{ij}$ là năng suất (số sản phẩm sản xuất được trong một đơn vị thời gian) của công nhân i khi làm việc tại vị trí j. Với mỗi cách bố trí dây truyền (công nhân nào làm ở vị trí nào) năng suất của một vị trí là năng suất của công nhân làm việc tại vị trí đó. Năng suất chung của dây truyền là năng suất của vị trí kém nhất trên dây truyền. Hãy tìm cách bố trí dây truyền để có năng suất cao nhất.

Chú ý: trong bài này ta phải tìm số k lớn nhất để G(k) có bộ ghép đầy đủ và 2 đỉnh i,j chỉ được nối với nhau nếu $a_{ij} \geq k$.

**2. Bài toán 4 (đề thi HSGQG năm học 1998-1999)**

Một đoạn đường quốc lộ có n cửa hàng, pi là khoảng cách của nó so với đầu đường. Nếu một cửa hàng có kho thì không cần phải đi lấy hàng, ngược lại thì phải đến lấy hàng ở cửa hàng có kho gần nhất. Hãy chọn k cửa hàng để đặt kho sao cho quãng đường đi lấy hàng dài nhất trong số các các cửa hàng còn lại là ngắn nhất.

**Thuật giải**

Bài này có thể làm bằng vét cạn (duyệt các tổ hợp). Ngoài ra còn có phương pháp quy hoạch động. Tuy nhiên chúng hoàn toàn không hiệu quả khi n lớn. Ta có thể áp dụng kỹ thuật tìm kiếm nhị phân kết hợp tham lam như sau.
Thủ tục search tìm kiếm nhị phân giá trị d trong miền $d_{min}..d_{max}$ tương tự bài toán 1. Riêng thủ tục check(d) sẽ thực hiện khác. Thay vì kiểm tra xem có thể bố trí k kho sao cho quãng đường đi lấy hàng của mọi cửa hàng không có kho đều ≤d không, ta sẽ làm ngược lại: *lần lượt bố trí các kho sao cho quãng đường đi lấy hàng của mỗi cửa hàng không bao giờ vượt quá d*.
Cách làm như sau:
Gọi $d_{ij}$ là khoảng cách giữa 2 cửa hàng i,j: $d_{ij} = |p_i - p_j|$
Dùng 2 cửa hàng giả có chỉ số là 0 và t = n+1 làm biên sao cho $d_{i0} = d_{it} = \infty$ với mọi i. Đặt 2 kho (giả) tại 2 cửa hàng đó.
Với mỗi cửa hàng i từ 1 đến n: nếu i đã có kho thì chuyển sang cửa hàng tiếp theo, ngược lại thì:
1. Tìm cửa hàng x có kho gần i nhất về phía bên trái (x thuộc 0..i).
2. Tìm cửa hàng y có kho gần i nhất về phía bên phải (y thuộc i..t).
Nếu dix hoặc $d_{iy} \le d$ thì quãng đường đi lấy hàng của i ≤ d (thoả mãn). Ngược lại tìm cửa hàng j chưa có kho xa i nhất về phía phải (j thuộc i..y) mà $d_{ij} \le d$. Đặt kho tại j.
Sau quá trình trên đếm số kho (tất nhiên không tính 2 kho 0 và t). Nếu số kho ≤ k thì đặt Ok là true.

**3. Bài toán 5 (đề thi Olympic Tin học SV 2004)**

Có N hành khách, M khách sạn và K xe bus. Mỗi hành khách cần về một khách sạn và mỗi xe bus sẽ đi qua một số khách sạn nào đó. Xe bus i có q[i] chỗ ngồi. Hãy sắp xếp hành khách lên xe bus sao cho:
1. Mỗi xe bus không chứa nhiều hành khách hơn số ghế của nó.
2. Tất cả hành khách đều lên xe bus có đi qua khách sạn mình cần đến.
3. Số xe bus cần dùng là ít nhất.

**Thuật giải**

Bài này có một thuật giải áp dụng kĩ thuật tìm kiếm nhị phân như sau: ta sẽ tìm số T nhỏ nhất sao cho: chỉ dùng T xe bus là chở được hết khách thoả mãn 3 điều kiện trên.
T sẽ được tìm bằng phương pháp nhị phân trong miền từ 1 đến K. Để kiểm tra giá trị T có thoả mãn không, ta sẽ tìm một tổ hợp T xe bus trong số K xe bus sao cho có thể sắp xếp N hành khách lên T xe bus đó thoả mãn 3 điều kiện trên.
Ta chọn T xe bus bằng phương pháp duyệt tổ hợp và kiểm tra tổ hợp có được có thoả mãn không bằng thuật toán cặp ghép (hoặc luồng trên đồ thị 2 phía). Thuật toán luồng thực thi nhanh hơn, được mô tả như sau:
1. Xây dựng đồ thị 2 phía G(X,Y,E). Trong đó: X là các khách sạn, Y là các xe bus được chọn (T xe bus). Khả năng thông qua của mỗi đỉnh thuộc X là số hành khách đến khách sạn đó. Khả năng thông qua của mỗi đỉnh thuộc Y là số chỗ ngồi của xe bus đó. Nếu xe bus j đi qua khách sạn i thì đặt khả năng thông qua trên cạnh (i,j) là N, ngược lại thì đặt bằng 0.
2. Tìm luồng cực đại trên đồ thị G. Nếu giá trị luồng qua mỗi đỉnh ở X bằng khả năng thông qua của nó thì việc lựa chọn tổ hợp T xe bus đó là thoả mãn yêu cầu. Từ giá trị luồng ta cũng dễ dàng tìm được cách sắp xếp hành khách.

*Nếu các bạn không quen với thuật toán luồng thì có thể cài bằng thuật toán cặp ghép:*

1. Xây dựng đồ thị 2 phía G(X,Y,E). Trong đó: X là các hành khách, Y là các chỗ ngồi trên các xe bus được chọn (có T xe bus được chọn, xe t có q[t] chỗ thì ta sinh q[t] đỉnh trong Y).

Nếu xe bus tương ứng của j đi qua khách sạn của i thì đưa cạnh (i,j) vào E.
2. Tìm bộ ghép đầy đủ trên đồ thị G. Nếu có thì việc lựa chọn tổ hợp T xe bus đó là thoả mãn yêu cầu và bộ ghép đó chính là cách sắp xếp hành khách.

**Ta rút ra một số nhận xét sau**:

1. Việc tìm kiếm nhị phân làm giảm số tổ hợp phải duyệt rất nhiều. Nếu vét cạn thuần tuý thì phải duyệt tối đa $2^K$ tổ hợp. Tuy nhiên dùng phương pháp tìm kiếm nhị phân, nếu đã duyệt xong giá trị T thì ta không cần phải kiểm tra các tổ hợp nhiều hơn T phần tử (nếu T thoả mãn) hoặc ít hơn T phần tử (nếu T không thoả mãn).
2. Ta chỉ cần tìm một tổ hợp thoả mãn, nên số tổ hợp cần duyệt còn ít hơn nữa (tìm thấy một tổ hợp thoả mãn thì không cần tìm các tổ hợp khác cùng số phần tử nữa). Và ta có thể áp dụng một số kĩ thuật tham lam trong khi duyệt như: ưu tiên chọn các xe bus chở được nhiều khách, đi qua nhiều khách sạn, không xét các xe bus không đi qua khách sạn nào trong số các khách sạn có hành khách cần đến…
3. Cần giảm miền tìm kiếm $k_{min}…k_{max}$ xuống càng nhiều càng tốt (kết hợp với phương pháp tham lam chẳng hạn). Mặt khác ghi nhận lại những giá trị T đã xét để tránh phải xét lại một cách vô ích.

**Ta cũng có một bài toán giải bằng kĩ thuật tương tự:**

**Bài toán 6. Mạng máy tính**

Mạng gồm N máy tính, một số cặp máy được nối với nhau bằng cáp. Có một chương trình điều khiển, máy nào được cài đặt chương trình đó thì có thể điều khiển tất cả các máy khác có cáp nối trực tiếp với nó (tất nhiên có thể điều khiển chính nó). Cần chọn ra một số ít máy nhất để cài chương trình sao tất cả các máy đều được điều khiển.

**Thuật giải**

Bài này chỉ giải được bằng cách duyệt tất cả các tổ hợp. Tuy nhiên áp dụng phương pháp tìm kiếm nhị phân sẽ giúp giảm số tổ hợp cần duyệt rất nhiều (lập luận như bài 5). Bạn đọc có thể cài theo cả 2 phương pháp để so sánh.

Trên đây là một số bài toán áp dụng được kỹ thuật tìm kiếm nhị phân. Lời giải không kèm chương trình mẫu mà dành cho bạn đọc tự cài đặt vừa để tránh bài viết dài một cách không cần thiết vừa để bạn đọc rèn luyện kỹ năng lập trình. Bạn đọc nào có nhu cầu về chương trình mẫu để tham khảo, xin liên hệ với tác giả qua email: *TungNT31@yahoo.com*

## Độ phực tạp thuật toán và tổ chức dữ liệu

**Trần Đỗ Hùng**

*Trong Tin học, khi phân tích một bài toán người ta cũng tìm giả thiết và kết luận. Giả thiết là những dữ liệu đưa vào máy tính xử lí kèm theo các điều kiện ràng buộc chúng (gọi là input) và kết luận là những thông tin thu được sau xử lí (gọi là output). Bài toán trong Tin học có thể hiểu là bất cứ công việc gì có thể giao cho máy tính thực hiện: vẽ một chữ, một hình trên màn hình cũng là bài toán! ...*

Phương pháp giải bài toán có thể cài đặt thành chương trình máy tính (viết bằng ngôn ngữ lập trình) gọi là thuật toán. Thuật toán được thể hiện là dãy các thao tác có thứ tự và hữu hạn để sau khi thực hiện dãy thao tác này, từ input của bài toán sẽ nhận được output của bài toán.

Một bài toán có thể được giải bằng một vài thuật toán khác nhau. Người ta cần lựa chọn thuật toán thích hợp và do đó cần đánh giá thuật toán. Để đánh giá thuật toán người ta dựa vào khái niệm độ phức tạp thuật toán. Độ phức tạp của thuật toán là đại lượng đánh giá lượng thời gian và không gian bộ nhớ dành cho thực hiện thuật toán.

Từ ý nghĩa thực tiễn của các bài toán khác nhau, có khi người ta quan tâm tới thuật toán đòi hỏi ít thời gian thực hiện, nhưng cũng có khi lại quan tâm nhiều hơn tới thuật toán cho phép cài đặt dữ liệu chiếm ít không gian bộ nhớ.
Độ phức tạp về không gian bộ nhớ của thuật toán phụ thuộc phần lớn vào cấu trúc dữ liệu được sử dụng khi cài đặt thuật toán.
Độ phức tạp về thời gian thực hiện (*còn gọi là độ phức tạp tính toán*) được đánh giá sơ bộ dựa vào số lượng các thao tác cơ bản (gán, so sánh 2 số nguyên, cộng, nhân 2 số nguyên …). Số lượng các thao tác này là một hàm số của kích cỡ dữ liệu input. Nếu kích cỡ dữ liệu input là N thì thời gian thực hiện thuật toán là một hàm số của N. Với một thuật toán trên các bộ dữ liệu input khác nhau (cùng kích cỡ N) có thể các hàm số này là khác nhau; song điều quan tâm hơn cả là *mức độ tăng* của chúng như thế nào khi *N tăng đáng kể*.
*Ví dụ:* Xét thuật toán tìm giá trị lớn nhất trong dãy N số sau đây:
*Input*. Số nguyên dương N, dãy N số $A_1, A_2, \ldots, A_N$
*Output* max$\{A_1, A_2, \ldots, A_N\}$
*Bước 1*. max ← $A_1$
*Bước 2*. **for** i ← 2 **to** N **do**
**If** max < $A_i$ **then** max ← $A_i$
*Bước 3*. Hiện max
Trường hợp xấu nhất: dữ liệu vào là dãy sắp tăng, thì số thao tác cơ bản là f(N)=2N+1
Trường hợp tốt nhất: giá trị lớn nhất ở ngay đầu dãy, thì thao tác cơ bản là g(N)= N.
Khi N lớn đáng kể thì f(N) và g(N) coi như xấp xỉ nhau và xấp xỉ hàm bậc nhất của N. Vậy trong trường hợp này ta kí hiệu độ phức tạp thời gian của thuật toán trên là O(N).
Người ta phân lớp các bài toán theo độ phức tạp thuật toán. Có thể liệt kê một số lớp sau có độ phức tạp tăng dần:
- Độ phức tạp hằng O(1)
- Độ phức tạp lôgarit O(logN)
- Độ phức tạp tuyến tính O(N)
- Độ phức tạp NlogN O(NlogN)
- Độ phức tạp đa thức $O(N^k)$ k: hằng nguyên
- Độ phức tạp luỹ thừa $O(a^N)$ a: cơ số nguyên dương khác 1
- Độ phức tạp giai thừa O(N!)

Tính hiệu quả (*về thời gian*) của thuật toán là đánh giá về thực hiện thuật toán trong một khoảng thời gian cho phép. Tính hiệu quả được nhận xét gián tiếp qua độ phức tạp tính toán của thuật toán. Độ phức tạp lớn thì thời gian thực hiện lâu hơn.
Chúng ta xét hai bài toán quen thuộc sau đây làm ví dụ về lựa chọn thuật toán và cài đặt dữ liệu:

**Bài toán 1. Dãy đơn điệu tăng dài nhất**
Cho mảng ĂN) gồm N phần tử nguyên. Hãy xoá đi một số ít nhất các phần tử của mảng để những phần tử còn lại lập thành dãy tăng dài nhất.
**Dữ liệu** vào từ file văn bản DAYTANG.IN
Dòng đầu là số nguyên N ($1 \le N \le 30000$)
Tiếp theo là N số nguyên lần lượt từ phần tử đầu đến phần tử cuối của mảng.
**Kết quả** ghi ra file văn bản DAYTANG.OUT
Dòng đầu là số K là số lượng các phần tử còn giữ lại
Tiếp theo là K dòng, mỗi dòng ghi 2 số: số thứ nhất là giá trị phần tử giữ lại, số thứ hai là chỉ

số (trong mảng ban đầu) của phần tử được giữ lại này. **DAYTANG.IN**
**10**
**1 2 8 10 5 9 4 3 6 7**
**DAYTANG.OUT**
**5**
**1 1**
**2 2**
**5 5**
**6 9**
**7 10**
Bài toán này thường được giải bằng Qui hoạch động. Có thể cài đặt dữ liệu như sau:
Xây dựng 2 mảng một chiều T và mảng D với ý nghĩa sau:
D[i] là độ dài của dãy kết quả khi bài toán chỉ xét dãy $A_1$ , $A_2$ ,…, $A_i$ và nó được tính theo công thức truy hồi:
D[i] = Max { D[i], D[j] +1 với mọi j mà j < i và $A_j \leq A_i$ } (1)
T[i]=j là chỉ số (trong dãy A ban đầu) của phần tử đứng ngay trước $A_i$ trong dãy kết quả.
Cách tìm T[i]: duyệt mảng A từ vị trí 1 đến vị trí i-1, vị trí j thoả mãn có D[j] lớn nhất và A[j]≤A[i].
Khởi trị: T[1]:= 0 ; D[1]:= 1; Ví dụ:

Khi duyệt ngược tìm kết quả ta tiến hành như sau:
Tìm phần tử lớn nhất trong D, giả sử đó là D[i], ta đổi dấu của D[i] coi như đánh dấu nó, tìm tiếp phần tử j = T[i], lại đổi dấu D[j], quá trình cứ như thế lùi chỉ số j đến khi j=0

Kết qủa được dãy con dài nhất là : 3 4 7 9 10
Độ phức tạp tính toán của thuật toán trên là $O(N^2)$. Với N=30000 thì mặc dù có thể tổ chức các mảng động một chiều để cài đặt dữ liệu thực hiện thuật toán nhưng cũng không thể chấp nhận được vì thời gian thực hiện thuật toán quá lâu!

Ta tìm kiếm một thuật toán khác (vẫn bằng qui hoạch động):
*Về cài đặt dữ liệu:*
+ Mảng một chiều A là dãy số đã cho ban đầu.
+ Mảng một chiều L dùng để lưu các chỉ số (*trong dãy A ban đầu*) của một số phần tử của A xếp theo giá trị tăng dần (tạo thành một dãy trung gian là H) để dựa vào H ta có thể tìm ra giá trị mảng Tr.
+ Mảng Tr có ý nghĩa như sau: Tr^[k] là chỉ số (*trong dãy A ban đầu*) của phần tử đứng trước phần tử $A_k$ trong dãy kết quả. Dựa vào mảng Tr sẽ tìm ra dãy kết quả.
+ Dùng biến d để theo dõi độ dài dãy kết quả
*Thuật toán:*
- Ban đầu dãy kết quả có độ dài d=1, phần tử đầu của dãy H là $A_1$. Xác nhận chỉ số (*trong dãy A ban đầu*) của phần tử đầu dãy H là L[1]:=1, chỉ số (*trong dãy A ban đầu*) của phần tử cuối dãy H là L[d]=1. Chưa có phần tử trước $A_1$ nên tạm coi chỉ số của phần tử trước $A_1$ là 0 (gán Tr^[1] :=0;)
- Duyệt tuyến tính các phần tử từ $A_2$ đến $A_N$. Giả sử đã xét đến $A_k$.

+ Nếu $A_k$ < A thì thay phần tử đầu của dãy H bằng $A_k$ nhờ gán lại L[1]:=k; Tr^[k]:= 0;
+ Nếu $A_k$ > = $A_{L[d]}$ thì thêm $A_k$ vào cuối dãy H(điều này tương ứng với sự kiện dãy kết quả được kéo dài thêm 1 phần tử nữa vì phần tử $A_k$ có chỉ số k > L[d]. Tuy nhiên, chú ý rằng dãy H không đồng nhất với dãy kết quả). Gán Tr^[k]:= L[d]; Inc(d); L[d]=k;
+ Trường hợp còn lại: dùng phương pháp tìm kiếm nhị phân để biết vị trí $A_k$ trong dãy H={$A_{L[1]}$, $A_{L[2]}$, …, $A_{L[d]}$ ). Giả sử tìm được chỉ số t sao cho $A_{L[t-1]} \le A_k \le A_{L[t]}$ thì ta gán Tr^[k] :=L[t-1]; L[t] := k;
- Dựa vào mảng Tr^ tìm ngược dãy kết quả (dài bằng d). Chú ý rằng phần tử cuối cùng của dãy kết quả là $A_{L[d]}$

Thủ tục chính của chương trình theo thuật toán trên là:

```
procedure lam;
var l : mg;
i,j,k,t : integer;
f : text;
begin
new(tr); { tr^[i]=j nghĩa là trong dãy tăng kết quả: liền trước A[i] là A[j] }
fillchar(l,sizeof(l),0);
fillchar(tr^,sizeof(tr^),0);
l[1]:=1; {Xác nhận phần tử đầu của dãy H có chỉ số là 1 }
d:=1; {d: số lượng phần tử của dãy kết qủa}
for k:=2 to n do {duyệt dãy A, từ phần tử thứ 2}
begin
if â[k]<>
begin
l[1]:=k; {phần tử đầu của dãy H là â[k]}
tr^[k]:=0; {â[k] có thể là phần tử đầu của dãy kết quả}
end
else
if â[k]>=â[l[d]] then
begin
tr^[k]:=l[d];{xác nhận l[d] là chỉ số của phần tử có thể đứng trước â[k] trong dãy kết quả }
inc(d);
l[d]:=k; {thêm phần tử â[k] vào cuối dãy H}
end
else {bằng tìm kiếm nhị phân vị trí của â[k] trong dãy H}
begin
i:=1;j:=d;
while i< do>
begin
t:=(i+j)div 2;
if â[k]>=â[l[t]] then i:=t+1
else j:=t;
end;
t:=(i+j) div 2;
```

```
tr^[k]:=l[t-1];
l[t]:=k; {phần tử thứ t trong H là Â[k]}
end;
end;

k:=l[d]; {tìm vết dãy kết quả, lần từ cuối dãy}
fillchar(l,sizeof(l),0);
assign(f,fo);
rewrite(f);
writeln(f,d);
while k>0 do {ghi lại vết dãy kết quả}
begin
l[k]:=1;
k:=tr^[k];
end;
for k:=1 to n do
if l[k]=1 then writeln(f,â[k],#32,k); {ghi kết quả vào file output}
close(f);
end;
```

Độ phức tạp thời gian của thuật toán này là O(NlogN) nên có thể chấp nhận về thời gian cho phép khi N=30000.

**Bài toán 2. Palindrome (time limit 2s)**
Một xâu được gọi là xâu đối gương nếu đọc từ trái qua phải cũng giống như đọc từ phải qua trái. Ví dụ xâu "madam" là một xâu đối gương. Bài toán đặt ra là cho một xâu S gồm các kí tự thuộc tập M=['a'..'z'], hãy tìm cách chèn vào xâu S các kí tự thuộc tập M tại các vị trí bất kì với số lượng kí tự chèn vào là ít nhất để xâu S thành xâu đối gương. Ví dụ: xâu: "adbhbca" ta sẽ chèn thêm 2 kí tự ( **c** và **d**) để được xâu đối gương "ad**c**bhbc**d**a".
Dữ liệu vào trong file PALIN.IN có dạng:
Dòng thứ nhất là một số nguyên dương T (T <=10) là số bộ test
T dòng sau, mỗi dòng chứa một xâu S, độ dài mỗi xâu không vượt quá 500.
Kết quả ghi ra file PALIN.OUT có dạng: Gồm nhiều dòng, mỗi dòng là một xâu đối gương sau khi đã chèn thêm ít kí tự nhất vào xâu S tương ứng ở file input.

**Thuật toán.**
Dùng Qui hoạch động:
Gọi L[i,j] là số kí tự cần chèn thêm vào đoạn S[i..j] để đoạn này thành đối gương (các đoạn S[1..i-1] và S[j+1..n] đã đối xứng nhau rồi) thì:

L[i,j] = Min {L[i+1,j-1}, L[i+1,j]+1, L[i,j-1]+1} (2)

L[i,j] = L[i+1,j-1] chỉ khi S[i]=S[j]
L[i,j] = L[i+1,j]+1khi S[i]<>S[j] và ta chèn thêm S[i] vào bên phải S[j]. Đánh dấu hiện tượng này bằng bít 1 trên mảng đánh dấu D.

L[i,j] = L[i,j-1]+1 khi S[i]<>S[j] và ta chèn thêm S[j] vào bên trái S[i]. Đánh dấu hiện tượng này bằng bít 0 trên mảng đánh dấu D.
Nhãn L[1,N] chính là số kí tự chèn thêm cần tìm.

Do kích thước đề bài, không thể tổ chức mảng hai chiều L (kể cả cách tổ chức theo kiểu mảng động cũng không thể được, vì mảng hai chiều kích thước 500?500 phần tử kiểu Integer là quá lớn).
Vì vậy điều chủ chốt để thực hiện thuật toán này là vấn đề cài đặt dữ liệu theo cách khác hợp lí hơn.
Ta có nhận xét: theo công thức truy hồi (2) thì L[i,j] chỉ liên quan tới L[i+1,j-1] , L[i,j-1] và L[i+1,j].
Gọi độ dài đoạn S[i..j] là k=j-i+1. (hay là j=i+k-1)
Thay cho L[i, j] ta dùng C3[i] (ứng với khoảng cách từ i tới j là k).
Thay cho L[i+1, j] ta dùng C2[i+1] (ứng với khoảng cách từ i+1 tới j là k-1).
Thay cho L[i, j-1] ta dùng C2[i] (dùng C2 vì ứng với khoảng cách từ i tới j-1 vẫn là k-1).
Thay cho L[i+1, j-1] ta dùng C1[i+1] (ứng với khoảng cách từ i+1 tới j-1 là k-2)
Công thức truy hồi (2) có dạng mới là:
C3[i] = Min {C2[i] +1, C2[i+1] +1, C1[i+1]} (2.b)
C3[i] = C1[i+1] khi S[i]=S[j]
C3[i] = C2[i+1] +1 khi S[i]<>S[j] và chèn thêm S[i] vào bên phải S[j]. Dùng mảng D để đánh dấu hiện tượng này bằng bít là 1 (d[i,v] := d[i,v] or (1 SHL k), v=j div 8, k=j mod 8).

C3[i] = C2[i] +1 khi S[i]<>S[j] và chèn thêm S[j] vào bên trái S[i]. Hiện tượng này đã được đánh dấu bằng bít là 0 khi khởi trị D.
Vậy ta chỉ cần dùng 3 mảng một chiều là C1(500), C2(500), C3(500) và một mảng hai chiều là D (500, 500 div 8).
*Sau đây là toàn bộ chương trình giải bài toán 2*

```
const maxn = 501;
fi = 'palin.in';
fo = 'palin.out';
type m1 = array[1..maxn]of char;
m2 = array[1..2*maxn]of char;
m3 = array[0..maxn div 8]of byte;
m4 = array[0..maxn]of m3;
m5 = array[0..maxn]of integer;
var f,g : text;
a : m1;
d : m4; {đánh dấu }
c1,c2,c3 : m5; {3 mảng 1 chiều thay cho mảng hai chiều L}
kq : m2; {xâu đối gương cần tìm}
dau,cuoi,sol,n,test,sotest :integer;
procedure readinp;
begin
n:=1;
while not seekeoln(f) do
begin
```

```
read(f,a[n]);
if a[n]=' ' then
if n>1 then break
else continue;
inc(n);
end;
readln(f);
dec(n);
end;
procedure batbit(i,j :integer);
var v,k :integer;
begin
v:=j div 8;
k:=j mod 8;
d[i,v]:=d[i,v] or (1 shl k);
end;
procedure init_data;
var i,j,k :integer;
begin
fillchar(d,sizeof(d),0);
fillchar(c1,sizeof(c1),0);
fillchar(c2,sizeof(c2),0);
fillchar(c3,sizeof(c3),0);
for i:=1 to n-1 do {xét các xâuchỉ gồm 2 kí tự liên tiếp a[i] và a[i+1]}
begin
j:=i+1;
if a[i]<>a[j] then
begin
c3[i]:=1; {phải chèn vào 1 kí tự}
batbit(i,j); {đánh dấu hiện tượng chèn này: chèn a[i] vào bên trái a[i+1]}
end;
end;
end;
procedure process;
var k,i,j :integer;
begin
init_data;
for k:=3 to n do
begin
{c1 lưu trạng thái của c2}
move(c2,c1,sizeof(c2));
{ c2 lưu trạng thái của c3}
move(c3,c2,sizeof(c3));
for i:=1 to n-k+1 do
begin
j:=i+k-1;
```

```
if a[i]=a[j] then
begin
c3[i]:=c1[i+1]; {không cần chèn thêm kí tự nào}
end
else
begin
if c2[i]
begin
c3[i]:=c2[i]+1; {chèn a[i] vào bên trái a[j]}
batbit(i,j); {đánh dấu hiện tượng chèn này}
end
else
c3[i]:=c2[i+1]+1; {chèn a[j] vào bên phải a[i] }
end;
end;
end;
end;
function getbit(i,j :integer):byte;
var p,k,v :integer;
begin
v:=j div 8;
k:=j mod 8;
p:=d[i,v] and (1 shl k);
getbit:=ord( p>0 );
end;
procedure print;
var i :integer;
procedure find(left,right :integer);
var k :byte;
begin
if left>right then exit;
if a[left]=a[right] then
begin
inc(dau); kq[dau]:=a[left];
if left
begin
dec(cuoi);
kq[cuoi]:=a[right];
end;
find(left+1,right-1);
exit;
end;
k:=getbit(left,right);
if k=1 then
begin
inc(dau);
```

```
kq[dau]:=a[right];
dec(cuoi);
kq[cuoi]:=a[right];
find(left,right-1);
end
else begin
dec(cuoi);
kq[cuoi]:=a[left];
inc(dau);
kq[dau]:=a[left];
find(left+1,right);
end;
end;
begin
fillchar(kq,sizeof(kq),0);
sol:=c3[1];
dau:=0;
cuoi:=n+sol+1;
find(1,n); {Tìm lại kết quả bằng đệ qui }
for i:=1 to n+sol do write(g,kq[i]);
writeln(g);
end;
BEGIN
assign(f,fi); reset(f);
assign(g,fo); rewrite(g);
readln(f,sotest);
for test:=1 to sotest do
begin
readinp;
process;
print;
end;
close(f); close(g);
END.
```

Qua hai bài toán trên, nhận thấy rằng khi cố gắng tìm tòi, lựa chọn thuật toán và cài đặt dữ liệu có thể giải được những bài toán đã quen thuộc đạt yêu cầu tốt hơn trước: thời gian thực hiện nhanh và thích ứng được với các bộ dữ liệu vào có kích cỡ lớn hơn.

*Trần Đỗ Hùng*
*giáo viên PTTH Nguyễn Huệ Hà Tây*

## Qui hoạch động với các bài toán có dữ liệu lớn

**Cao Minh Anh**

*Như chúng ta đã biết khi giải một bài toán thì vấn đề thời gian và dữ liệu là cốt lõi. Vì thế trong các kì thi chúng ta thường gặp các bài có dữ liệu lớn. Những bài nay thường rất khó*

*bởi vì ta vừa phải giải quyết vấn đề dữ liệu, vừa phải giải quyết vấn đề thời gian. Vì thế khi có được một thuật toán hay tối ưu vẫn chưa chắc giải được những bài này, tối ưu chưa đủ mà cần phải có phương pháp cài đặt tốt để có thể giải quyết về mặt dữ liệu. Quy hoạch động tượng trưng cho sự tối ưu hoá về thời gian vì thế tôi xin trình bày một số bài toán dữ liệu lớn dùng quy hoạch động để giải và cách thức xử lý của từng bài toán.*

**Bài 1: Tìm số lớn nhất.**
Cho n số nguyên ($n<=2^{32}$) trong file **max.inp**, hãy tìm số lớn nhất trong n số nguyên đã cho. Kết quả được viết vào file **max.out** gồm 2 số M, P − trong đó M là số lớn nhất và P là vị trí của nó trong n số nguyên. Nếu có nhiều kết quả đưa ra số lớn nhất có vị trí lớn nhất.
**Ví dụ:**

Đây là một bài quy hoạch động rất đơn giản, tôi muốn đề cập để các bạn thấy rằng bài toán tìm max là một bài toán rất cơ bản, ai học cũng biết bài toán này nhưng chắc các bạn sẽ không nghĩ rằng đây là một bài toán qui hoạch động đơn giản mà ai cũng có thể làm được. Tôi muốn giới thiệu với các bạn bài toán tìm max với dữ liệu lớn dùng file để chứa các phần tử thay vì dùng mảng, như thế ta vừa tiết kiệm được dữ liệu vừa giải quyết bài toán nhanh nhất.

```
uses crt;
const
fi='max.inp';
go='max.out';

var max,n,k :longint;
f,g :text;

procedure Openf;
begin
assign(f,fi);
reset(f);
assign(g,go);
rewrite(g);
end;

procedure Closef;
begin
close(f);
close(g);
end;

procedure Main;
var i:integer;
begin
max:=-maxlongint;
readln(f,n);
```

```
for i:=1 to n do
begin
readln(f,k);
if k>max then max:=k;
end;
end;

procedure Print;
begin
writeln(g,max);
end;

begin
clrscr;
Openf;
Main;
Print;
Closef;
end.
```

**Bài 2:** Cho N số tự nhiên nằm trong khoảng [0..9], hãy tìm dãy tăng dài nhất dài nhất.
**Dữ liệu vào:** File **daytang.inp** gồm dòng đầu tiên là số N.N dòng tiếp theo là N số tự nhiên.
**Dữ liệu ra:** File **daytang.out** gồm 2 số M là chiều dài của dãy tăng dài nhất.
**Ví dụ:**

- Nếu bài trên với dữ liệu N khoảng 10000 thì ta có thể giải quyết theo cách sau:
+ Nhập N số trên vào mảng a.
+ Gọi Fx[i] là chiều dài dài nhất của dãy tăng kết thúc là phần tử a[i].
Như vậy ta có chương trình quy hoạch đơn giản như sau:

```
procedure Quyhoach;
var i,j:integer;
begin
for i:=1 to n do fx[i]:=1;
for i:=2 to n do
for j:=i-1 downto 1 do
if a[j]<=a[i] then
if fx[i]
```

Muốn xuất kết quả ta chỉ cần writeln(Max(fx[1],fx[2],..,fx[n]));
Nhưng nếu bài toán trên với dữ liệu lớn không thể bỏ vào mảng thì ta không thể giải quyết theo phương pháp trên được.
Ta chú ý các số chỉ nằm trong khoảng [0..9] vì thế thay vì dùng mảng Fx[i] là chiều dài dãy tăng lớn nhất khi a[i] cuối dãy ta có thể dùng mảng Fx[i] là chiều dài dãy tăng lớn nhất khi i đứng cuối dãy. Như thế chỉ cần khai báo:

```
Var fx : array[0..9] of longint;
```

Như vậy khi nhập một số k. Thì số k có thể đặt sau dãy tăng có tận cùng là 0,1,2..k. Như vậy

thì fx[k] sẽ bằng
Fx[k]:=max(Fx[0]+1,Fx[1]+1,Fx[2]+1,..,fx[k]+1)
Với cách làm như thế thì bài toán trở nên đơn giản hơn rất nhiều vài có thể giải quyết với N rất lớn.
Đây là toàn bộ chương trình:

```
uses crt;
const
fi='daytang.inp';
go='daytang.out';

var fx :array[0..9] of longint;
n,k,max :longint;
f,g :text;

procedure Openf;
begin
assign(f,fi);
reset(f);
assign(g,go);
rewrite(g);
end;

procedure Closef;
begin
close(f);
close(g);
end;

procedure Quyhoach;
var i,j:integer;
begin
readln(f,n);
for i:=1 to n do
begin
readln(f,k);
for j:=k downto 0 do
if fx[k]
end;
end;

procedure Findmax;
var i:integer;
begin
max:=0;
for i:=0 to 9 do
```

```
if fx[i]>max then max:=fx[i];
end;

procedure Xuat;
var i:integer;
begin
writeln(g,max);
end;

begin
clrscr;
openf;
Quyhoach;
Findmax;
Xuat;
Closef;
end.
```

Chúng ta có thể dùng với các số tự nhiên lớn hơn không nhất thiết là từ [0..9] chỉ cần khai báo mảng Fx lớn hơn là được.Sau đây chúng ta chuyển qua một bài toán dùng qui hoạch động rất hay đó là bài Cấp số cộng.

**Bài 3: Cấp số cộng (Đề thi quốc gia bảng Bm).**

Cho một tệp văn bản gồm N (N rất lớn) số nguyên a1, a2, a3,..,an với Abs(ai)<=30 000. Hãy tìm trong dãy A đó một dãy con dài nhất lập thành một dãy cấp số cộng có công sai là d. Với 0<D<=100.

**Dữ liệu vào:** Csc. INP

- Dòng đầu ghi số N
- N dòng tiếp theo ghi các số ứng với dãy A

Dữ liệu ra: Csc.OUT

- Dòng đầu ghi số M là số phần tử và công sai của dãy cấp số cộng đó
- M dòng tiếp theo ghi số chỉ số của các số thuộc cấp số cộng.

**Ví dụ:**

Nếu dữ liệu nhỏ N=10000 thì ta có thể dùng phương pháp duyệt đơn thuần để giải bài toán trên một cách dễ dàng, nhưng với dữ liệu N<=100000 thì quả là lớn,việc lưu các số này vào mảng là một chuyện không thể, huống chi ta phải duyệt với độ phức tạp N*(N+1 )/2. Nếu duyệt thì ta sẽ không giải quyết được dữ liệu cũng như thời gian.

Các bạn hãy chú ý rằng nếu biết công sai và số đầu hay số cuối thì ta có thể biết được dãy cấp số cộng nó như thế nào, tại sao ta không tìm dãy cấp số cộng dài nhất ứng với một công sai nào đó.

Giả sử cần tìm chiều dài lớn nhất của dãy cấp số cộng có công sai là d

+ Ta gọi Fx[i] là chiều dài dãy cấp số cộng dài nhất kết thúc bằng i.

Suy ra:Fx[i]=Fx[i-d]+1;

Vì đã biết công sai là d nên khi nhập được số k nào đó thì muốn tạo thành cấp số cộng tận cùng bằng k thì số trước đó phải là (k-d). Như vậy chiều dài dài nhất của dãy cấp số cộng tận cùng bằng k sẽ bằng chiều dài dài nhất của dãy cấp số cộng tận cùng là k-d cộng 1.

Ta có thể khai báo như sau:
Type arr=array[0..30000] of word;
var a,b :^arr;
Nhận xét chiều dài tối đa của dãy cấp số cộng là 60001 nên ta khai báo word là vừa đủ.
+ Với mảng a dùng cho các số không âm,mảng b dành cho các số âm.
A[i] là chiều dài dài nhất của dãy cấp số cộng tận cùng bằng i,
B[i] là chiều dài dài nhất của dãy cấp số cộng tận cùng bằng -i,
Vậy để giải quyết bài toán Cấp số cộng ta chỉ cần duyệt với từng công sai từ 1?0 là được.
Sau đây là bài giải:

```
uses crt;

const
fi='csc.inp';
go='csc.out';

Type arr=array[0..30000] of word;

var a,b :^arr;
max,k,s,d,luu,sc,sd,n :longint;
f,g :text;

procedure Openf;
begin
assign(f,fi);
reset(f);
assign(g,go);
rewrite(g);
end;

procedure Closef;
begin
close(f);
close(g);
end;

procedure Main;
var i:integer;
begin
max:=0;
for i:=1 to 100 do
begin
openf;
New(a);
New(b);
readln(f);
```

```
fillchar(â,sizeof(â),0);
fillchar(b^,sizeof(b^),0);
while not eof(f) do
begin
d:=max;
readln(f,k);
s:=k-i;
if s>=0 then â[k]:=â[s]+1
else
begin
if k>=0 then â[k]:=b^[abs(s)]+1
else b^[abs(k)]:=b^[abs(s)]+1;
end;
if max
if max
if d<>max then
begin
luu:=i;
sc :=k;
end;
end;
dispose(a);
dispose(b);
closef;
end;
end;

procedure Print;
var i:integer;
begin
Openf;
writeln(g,max,luu:4);
sd:=sc-(max-1)*luu;
readln(f,n);
for i:=1 to n do
begin
readln(f,k);
if k=sd then
begin
writeln(g,i);
sd:=sd+luu;
end;
end;
Closef;
end;
```

```
begin
clrscr;
Main;
Print;
end.
```

**Bài 4: Xâu fibonacci.**

***Định nghĩa:*** Dãy xâu **fibo** được xây dựng theo nguyên tắc sau:

+ $F_1$ =’B’;$F_2$ =’A’;
+ Fk = $F_{k-1}$ + $F_{k-2}$.

***Yêu cầu:*** Tính số lần xuất hiện của SR trong Fn(tức là số xâu con các kí tự liên tiếp nhau bằng SR). Hai xâu con được gọi là khác nhau nếu khác nhu ít nhất một ký tự.(N<=100).

***Dữ liệu vào:*** File FSTR.inp

+ Dòng đầu tiên là số N.
+ Dòng tiếp theo là xâu SR.(length(SR)<=15).

***Dữ liệu ra:*** File FSTR.out

Số lần xuất hiện tìm được.

*+ Phương pháp 1:* Thuật toán lùa bò vào chuồng. Với phương pháp này chỉ giải quyết với dữ liệu không lớn lắm (N<=35) nhưng cũng là một cách để cách bạn tham khảo.

+ Tìm Fn. Ta sẽ tìm Fn và đưa đó vào file để chứa.

Chương trình tìm và xuất Fn rất đơn giản như sau:

```
Function Fibo(N:integer):integer;
Begin
If n=1 then write(g,’B’)
Else
If n=2 then write(g,’A’)
Else
Fibo:=Fibo(n-2)+Fibo(n-1);
End;
```

+ Nhập xâu SR. Ta tưởng tượng rằng ’A’ chính là 1,’B’ chính là 0. Lúc này SR là biểu diễn nhị phân của số K. Vấn đề tìm k khá đơn giản. Sau khi tìm được k, ta mở file chứa Fn ra,sao đó lấy từng đoạn liên tiếp có chiều dài length(SR) ra chuyển ra nhị phân (với ’A’ chính là ’1’,’B’ là ’0’), nếu chuyển ra nhị phân bằng đúng k thì tăng đếm lên 1. Sau khi đọc hết file thì biến đếm chính là kết quả cần tìm.

**Ví dụ:**

SR=’ABA’ SR=’101’ đây là biểu diễn của số K=5.

Nếu dịch từng đoạn 3 ta sẽ đươc các xâu sau:’101’,011’,110’,’101’,’010’,’101’,’011’,’110’,’101’,’011’,’110’. Ta sẽ chuyển các xâu này ra số nếu bằng k thì tăng dem lên. Để tiết kiệm thời gian ta sẽ dùng bit để khỏi dùng chương trình chuyển nhị phân.

+ Có thể có bạn sẽ hỏi là tại sao ta không dùng xâu luôn cho nó nhanh khỏi phải dùng nhị phân chi cho nó mệt. Ta đọc từng đoạn xâu con rồi kiểm tra có bằng SR không là xong. Tôi muốn giới thiệu cho các bạn phương pháp trên để giải quyết bài có thể hỏi nhiều xâu Sr chứ không phải là một xâu. Nếu như dùng xâu thì ta tốn đến 15 byte, nhưng dùng số thì chỉ tốn 2 byte thôi.

Tôi đã giới thiệu xong phương pháp 'Lùa bò vào chuồng', hi vọng sẽ giúp các bạn có thêm một kinh nghiệm nào đó. Sau đây tôi xin bàn đến một phương pháp tối ưu.
*Phương pháp 2:* Quy hoạch động.
Gọi Fx[i] là số lần xuất hiện của Sr trong Fi.
Nhận xét:
Nếu biết Fx[k-2],Fx[k-1]
Suy ra: Fx[k]:=Fx[k-2]+Fx[k-1].
Nhưng công thức trên vẫn chưa đủ: Để ý rằng Đoạn đầu của $F_{k-2}$ và đoạn cuối của $F_{k-1}$ nối với nhau có thể có xâu SR.
Như vậy để hoàn chỉnh ta phải làm như sau:
Fx[k]:=Fx[k-2]+Fx[k-1].
If đoạn nối của $F_{k-2}$và $F_{k-1}$ có Sr then Fx[k]:=Fx[k]+1;
Như vậy bài làm đơn giản như sau:
+ Tìm xâu fibo có chiều dài nhỏ nhất nhưng lớn hơn xâu Sr, giả sử là Fk.
+ Ta xét xem xâu Sr xuất hiện bao nhiêu lần trong Fk và $F_{k+1}$.
+ Vấn đề là làm sao tìm đoạn nối. Khi n càng lớn thì xâu Fn cũng càng lớn không thể lưu trữ được. Ta có nhận xét sau:
- Đặt length(Sr)=p;
Khi xét xâu $F_k$ chỉ cần quan tâm đến p -1 phần tử đầu (?Giả sử cho vào xâu SA?, xét xâu $F_{k+1}$ chỉ cần xét p -1 phần tử cuối (?Giả sử cho vào xâu SB?).
→ Fk=SA……$F_{k+1}$ =……SB
$F_{k+2}$ =……SBSA……&hllip;
Nhưng nếu xét tiếp $Fk_{+3}$ thì :
$F_{k+3}$ =……SBSA……/……SB
Lúc này thì khác trước một chút: Đoạn giữa là đoạn tiếp súc giữa p-1 phần tử cuối của Fk ('Giả sử cho vào xâu YA'), và p -1 phần tử đầu của $F_{k+1}$ ('Giả sử cho vào xâu YB').
Bây giờ ta hình dung lại một chút:
Fk=SA.YA;$F_{k+1}$ =YB.SB;
$F_{k+2}$ = YB.(SBSA).YA;
$F_{k+3}$ = YB.SBSA.(YAYB).SB;
$F_{k+4}$ = YB.SBSA.YAYB.(SBYB).SBSA.YA
$F_{k+5}$ = YB.SBSA.YAYB.SBYB.SBSA.(YAYB).SBSA.YAYB.SB
$F_{k+6}$ = YB.SBSA.YAYB.SBYB.SBSA.YAYB.SBSA.YAYB.(SBYB)……
Các bạn đã thấy quy luật của nó chưa. Rất dễ nhận thấy SB và YA thay đổi luân phiên cho nhau, nhưng còn SA và YB thì khác, lúc đầu là SA nhưng về sau là YB hết. Từ những nhận xét trên ta có thể giải quyết bài này với dữ liệu có thể lên đến n<=5000 (chỉ cần làm thêm chương trình cộng số lớn là xong, và mảng động) vì độ phức tạp của cách làm này chỉ là n. Nếu dùng quy hoạch động thì bài toán này trở nên cực kỳ đơn giản phải không các bạn. Sau đây là chương trình của bài trên:

```
{$n+}
uses crt;
const
fi='fibo.inp'';
go='fibo.out';

var S,R,S1,S2,X,SA,SB,YA,YB :string;
```

```
a :array[1..200] of extended;
n,dem :integer;
f,g :text;

procedure Openf;
begin
assign(f,fi);
reset(f);
assign(g,go);
rewrite(g);
end;

procedure Closef;
begin
close(f);
close(g);
end;

procedure Swap(var A,B:string);
var T:string;
begin
T:=A;
A:=B;
B:=T;
end;

procedure Solve;
begin
readln(f,S);
readln(f,n);
S1:='b';S2:='á;
dem:=2;
while length(S1)
begin
inc(dem);
R:=S2;
S2:=S2+S1;
S1:=R;
end;
if pos(S,S1)<>0 then a[dem-1]:=1;
if pos(S,S2)<>0 then a[dem]:=1;
end;

procedure Main;
var i,k:integer;
begin
```

```
k :=length(S);
SA:=copy(S1,1,k-1);YA:=copy(S1,length(S1)-k+2,k-1);
SB:=copy(S2,1,k-1);YB:=copy(S2,length(S2)-k+2,k-1);
for i:=dem+1 to n do
begin
a[i]:=a[i-1]+a[i-2];
X:=SB+SA;
if pos(S,X)<>0 then a[i]:=a[i]+1;
SA:=YB;
Swap(SB,YA);
end;
writeln(g,a[n]:0:0);
end;

begin
clrscr;
Openf;
Solve;
Main;
Closef;
end.
```

Nếu các bạn muốn làm dữ liệu lớn hơn chỉ cần làm thêm chương trình cộng số lớn là xong ngay. Còn rất nhiều bài toán dùng quy hoạch động để giải rất hay mong rằng sẽ nhận được những đóng góp ý kiến, những bài giải hay để tạp chí Tin học nhà trường trở thành sân chơi, học hỏi cho tất cả các bạn yêu môn Pascal nói riêng và các bạn yêu tin học nói chung. Cuối cùng xin chúc các bạn thành công và nhận được thêm một kinh nghiệm nào đó về quy hoạch động. Mọi thắc mắc xin liên hệ địa chỉ

## Duyệt với độ ưu tiên và duyệt với độ sâu hạn chế

**Trần Đỗ Hùng**

Chúng ta rất quen thuộc với thuật toán Back-Tracking (duyệt có quay lui). Chúng ta hãy cùng nhau nhìn nhận lại vấn đề này một lần nữa trước khi đi vào một vài khía cạnh đặc biệt của vấn đề này: *duyệt với độ ưu tiên* và *duyệt với độ sâu hạn chế*.

Thường là chúng ta sử dụng Back-Tracking trong trường hợp nghiệm của bài toán là dãy các phần tử được xác định không theo một luật tính toán nhất định; muốn tìm nghiệm phải thực hiện từng bước, tìm kiếm dần từng phần tử của nghiệm. Để tìm mỗi phần tử, phải kiểm tra: *các khả năng* có thể chấp nhận được của phần tử này (gọi là kiểm tra 'đúng và sai). Nếu khả năng nào đó không dẫn tới giá trị có thể chấp nhận được thì phải loại bỏ và chọn khả năng khác (chưa được chọn). Sau khi chọn được một khả năng ta phải xác nhận lại trạng thái mới của bài toán; vì thế trước khi chuyển sang chọn khả năng khác ta phải trả lại trạng thái bài toán như trước khi chọn đề cử (nghĩa là phải quay lui lại trạng thái cũ). Nếu phần tử vừa xét chưa là phần tử cuối cùng thì duyệt tiếp phần tử tiếp theo. Nếu là phần tử cuối cùng của một nghiệm thì ghi nhận nghiệm. Nếu bài toán yêu cầu chỉ tìm một nghiệm thì sau khi ghi nhận nghiệm cần điều khiển thoát khỏi thủ tục đệ qui.

Thuật toán BackTracking xây dựng trên cơ sở tìm kiếm dần từng bước theo cùng một cách

thức, nên thường dùng các hàm, thủ tục đệ qui thực hiện thuật toán. Ví dụ một dàn bài như sau:

**Procedure Tim(k:Integer);** : *{Tìm khả năng cho bước thứ k}*
Begin
*Vòng lặp* < đề cử mọi khả năng cho bước khử thứ k >
Begin
< Thử chọn một đề cử >
if < đề cử này chấp nhận được > then
Begin
< Lưu lại trạng thái trước khi chấp nhận đề cử >
< Ghi nhận giá trị đề cử cho bước thứ k >
< Xác lập trạng thái mới của bài toán sau khi chấp nhận đề cử >
If < Chưa phải bước cuối cùng > then
Tim(k+1)
Else {là bước cuối cùng}
Ghi_nhan_Nghiem;
< Trả lại trạng thái của bài toán trước khi chấp nhận đề cử > *{Quay lui}*
< Xóa giá trị đã đề cử >
End;
End;
End;

Thuật toán cho phép tìm được mọi nghiệm (nếu có) khi điều kiện về thời gian cho phép và bộ nhớ còn đủ. Song trong thực tế, yêu cầu về thời gian và kích thước bộ nhớ bị hạn chế, nên việc áp dụng Back-Tracking cho một số bài toán có thể không dẫn tới kết quả mong muốn. Trong những trường hợp như thế, để khắc phục : *phần nào* những hạn chế này, chúng ta kết hợp với phương pháp duyệt với độ ưu tiên và phương pháp duyệt với độ sâu hạn chế.

**1. Duyệt với độ ưu tiên.**

Trước hết chúng ta nhớ lại phương pháp duyệt có cận: Cận là một điều kiện để kiểm tra đề cử có được chấp nhận hay không. Nhờ có cận chúng ta có thể : *loại bỏ* một số đề cử không thoả mãn điều kiện, làm cho quá trình duyệt nhanh hơn. Việc thử mọi khả năng đề cử cho một phần tử của nghiệm cũng giống như tình trạng 'dò dẫm chọn đường' của một người đi đường, mỗi khi đến ngãN-đường phải lần lượt chọn từng con đường đi tiếp trong các con đường xuất phát từ ngãN-đường đó. Nếu có được những điều 'chỉ dẫn' bảo đảm chắc chắn những con đường nào là đường 'cụt' không thể đi tới đích thì người đi đường sẽ loại ngay những con đường đó.

Trong phương pháp duyệt với độ ưu tiên chúng ta lại chú ý đến tình huống ngược lại: tìm những 'chỉ dẫn' cho biết chỉ : *cần đi theo* một số con đường nhất định trong N đường, coi những chỉ dẫn ấy như 'la bàn' chỉ phương hướng tìm kiếm đích của mình. Tất nhiên việc đưa ra những lời chỉ dẫn, những dự đoán và khẳng định điều này là 'đúng', điều kia là 'sai là việc thận trọng. Những khẳng định tưởng là ' chắc chắn' nếu thực sự chỉ là điều 'ngộ nhận' thì có thể bỏ sót một số con đường tới đích, hoặc chệch hướng không thể tới đích!

Nhưng nói chung, những chỉ dẫn hợp lí sẽ đi tới đích hoặc gần với đích nhanh nhất. Điều này rất thích hợp với những bài toán thực tế chỉ yêu cầu tìm lời giải 'gần sát với nghiệm'. Khi đó người ta thường duyệt với độ ưu tiên. Nội dung duyệt với độ ưu tiên xuất phát từ ý tưởng heuristic (tìm kiếm): tìm một : *'chỉ dẫn'* sao cho xác suất tới đích có thể chấp nhận. Công việc cụ thể là:

+ Sắp xếp các đề cử theo một 'khoá' (key) nào đó,
+ Sau khi sắp xếp, chỉ chọn một số đề cử ở các vị trí đầu (hoặc cuối) của danh sách đã sắp.
Những đề cử này theo suy luận hợp lí về tính chất và giá trị của khoá sẽ bảo đảm là một : *'chỉ dẫn'* cho xác suất tới đích có thể chấp nhận.
Nhược điểm của phương pháp này là có thể bỏ sót nghiệm hoặc chỉ tìm được lời giải 'gần sát với nghiệm'.
Để khắc phục, chúng ta có thể áp dụng nó nhiều lần, mỗi lần mở rộng thêm số lượng đề cử trong danh sách đề cử đãsắp. Đôi khi cũng phải thay đổi hoàn toàn, làm lại từ đầu: điều chỉnh và mở rộng thêm điều kiện chọn làm khoá sắp xếp cho hợp lí hơn.
*Ví dụ:* **(Bài toán lập lịch cho sinh viên chọn môn học)**
Sinh viên theo học các trường đại học thường rất bối rối bởi các quy tắc phức tạp đánh giá hoàn thành chương trình học. Việc hoàn thành chương trình học đòi hỏi sinh viên có một số kiến thức trong một số lĩnh vực nhất định. Mỗi một đòi hỏi có thể được đáp ứng bởi nhiều môn học. Một môn học cũng có thể đáp ứng đồng thời nhiều đòi hỏi khác nhau. Các quy tắc này thường được phổ biến cho các sinh viên ngay khi họ mới vào trường. Do thời gian còn lại quá ít nên mỗi sinh viên đều muốn theo học ít môn học nhất mà vẫn đáp ứng được tất cả các đòi hỏi của chương trình học. Có M môn học được đánh số từ 1 đến M. Có N đòi hỏi được đánh số từ 1 đến N. Với mỗi môn học cho biết danh sách các đòi hỏi được thoả mãn khi học môn này.
Cần viết một chương trình tìm ra một số ít nhất các môn học mà một sinh viên cần học để có thể hoàn thành chương trình học (nghĩa là đáp ứng được N đòi hỏi).
**Dữ liệu vào** từ file văn bản **MONHOC.INP** có cấu trúc:
ã Dòng đầu là 2 số nguyên dương M và N (M, N ≤ 200);
ã Trong M dòng sau, dòng thứ i là N số nguyên phân cách nhau bởi dấu cách mà số thứ j là 1 nếu môn học i đáp ứng được đòi hỏi j, là 0 trong trường hợp ngược lại.
**Kết quả** ghi ra file văn bản **MONHOC.OUT** có cấu trúc:
ã Dòng đầu tiên ghi số lượng ít nhất các môn học;
ã Dòng tiếp theo ghi số hiệu các môn học đó.

*Ví dụ:*

*Cách giải:*
Duyệt chọn các môn học với độ ưu tiên phù hợp: mỗi lần duyệt các môn ta chỉ đề cử một số môn học chưa chọn có nhiều yêu cầu chưa thỏa mãn sẽ được thoả mãn. Số lượng các môn học được đề cử càng nhiều thì thời gian thực hiện chương trình càng lâu và có thể dẫn tới tràn stack, do đó khi lập trình cần điều chỉnh số lượng này cho phù hợp thời gian cho phép trong đề bài. Ví dụ mỗi lần ta sắp giảm các môn học còn lại theo số lượng yêu cầu còn cần phải đáp ứng mà mỗi môn có thể đáp ứng được, sau đó chỉ chọn ra một vài môn học đầu dãy đã sắp để làm đề cử khi duyệt. Ngoài ra với M, N 200 cũng cần đặt ngắt thời gian trong khi duyệt để thoát khỏi duyệt trong phạm vi thời gian còn cho phép.
***Chương trình:***
```
const fi = 'monhoc.inp';
fo = 'monhoc.out';
max = 200;
lim = 3; {số lượng môn học được đề cử tối đa là lim, lim có thể điều chỉnh cho phù hợp}
type km1 = array[1..max,1..max] of byte;
```

```
km2 = array[1..max] of byte;
var time : longint;
a : km1; {a[i,j]=1 : môn i đáp ứng được yêu cầu j}
m,n : byte;
kq,lkq, {kq[i] là môn được chọn thứ i trong phương án}
dx_mon, {dx_mon[i]=1: môn i đãđược chọn}
dx_yeucau: km2; {dx_yeucau[j]=k: yêu cầu j đãđược đáp ứng bởi môn được chọn thứ k}
so_mon_chon, {số môn đãđược chọn}
lso_mon_chon : byte;
so_yc_dx : byte; {số yêu cầu đãđược đáp ứng}
f : text;

procedure read_in;
begin
{Đọc file input lấy giá trị cho các biến M, N và mảng A[1..M, 1..N]}
end;

procedure toi_uu;
begin
lkq:=kq;
lso_mon_chon:= so_mon_chon;
end;

function bac(i: byte): byte;
begin
{Hàm cho biết số yêu cầu chưa được đáp ứng sẽ được đáp ứng nếu chọn môn i}
end;

procedure tao_decu(var sl: byte; var danhsach: km2);
var i,j,k : byte;
b : km2;
begin
{Dùng mảng danhsach để chứa các môn chưa chọn, sau đó sắp giảm mảng này theo số lượng
yêu cầu chưa được đáp ứng mà mỗi môn có thể đáp ứng. Cuối cùng chỉ giữ lại không quá lim
môn (3 môn)}
sl:=0;
for i:=1 to m do
if dx_mon[i]=0 then
begin
inc(sl);
b[sl]:=bac(i);
danhsach[sl]:=i;
if b[sl]=0 then dec(sl);
end;
if sl>1 then
begin
```

```
for i:=1 to sl-1 do
for j:=i+1 to sl do
if b[i]
begin
k:=b[i]; b[i]:=b[j]; b[j]:=k;
k:=danhsach[i];
danhsach[i]:=danhsach[j];
danhsach[j]:=k;
end;
end;
if sl>lim then sl:=lim;
end;

procedure nap(i: byte); {chọn môn i, xác nhận những yêu cầu do môn i đáp ứng}
var j : byte;
begin
inc(so_mon_chon);
kq[so_mon_chon]:=i;
dx_mon[i]:=1;
for j:=1 to n do
if (a[i,j]=1) and (dx_yeucau[j]=0) then
begin
dx_yeucau[j] := so_mon_chon;
inc(so_yc_dx);
end;
end;

procedure bo(i: byte); {Không chọn môn i, xác nhận lại những yêu cầu chưa được đáp ứng}
var j : byte;
begin
for j:=1 to n do
if dx_yeucau[j]=so_mon_chon then dx_yeucau[j]:=0;
dec(so_mon_chon);
dx_mon[i]:=0;
end;

procedure try; {duyệt với các đề cử được ưu tiên}
var i,j,sl,lso_yc_dx : byte;
danhsach : km2;
begin
if (meml[$0:$046c]-time)/18.2>30 then exit;
if so_mon_chon >= lso_mon_chon then exit;
if so_yc_dx=n then
begin toi_uu; exit; end;

tao_decu(sl,danhsach);
```

```
lso_yc_dx:=so_yc_dx; {lưu lại số yêu cầu đãđáp ứng trước khi chọn đề cử, để phục vụ bước quay lui}
for i:=1 to sl do {chỉ duyệt với số lượng đề cử là sl}
begin
nap(danhsach[i]); {xác nhận đề cử thứ i trong danh sách ưu tiên}
try;
bo(danhsach[i]); {Quay lui: xoá xác nhận đề cử}
so_yc_dx:=lso_yc_dx; {Lấy lại số yêu cầu đãđáp ứng trước khi chọn đề cử }
end;
end;

procedure hien_kq;
begin
{Ghi vào file output số môn chọn ít nhất là lso_mon_chon và số hiệu các môn được chọn là giá trị các phần tử của mảng lkq}
end;

BEGIN
clrscr;
read_in;
lso_mon_chon:=m+1; {Khởi trị số môn được chọn ít nhất là m+1}
time:= meml[$0:$046C];
try;
hien_kq;
END.
```

**2. Duyệt với độ sâu hạn chế.**

Phương pháp này đã được các tác giả Đinh Quang Huy và Đỗ Đức Đông trình bày rất rõ ràng trong bài 'Chiến lược tìm kiếm sâu lặp', số báo Tin học và Nhà trường tháng 8/2003. Sau đây chúng tôi chỉ minh hoạ bằng một chương trình giải bài toán nêu trong bài báo đó:

*Bài toán. (**Biến đổi bảng số)*** Xét bảng A có NxN ô vuông (Nmso-char-type: Ê5), trong bảng có một ô chứa số 0, các ô còn lại mỗi ô chứa một số nguyên dương tuỳ ý. Gọi P là phép đổi giá trị của ô số 0 với ô kề cạnh với nó.

**Yêu cầu đặt ra là**: Cho trước bảng A và bảng B (B nhận được từ A sau một số phép biến đổi P nào đó). Hãy tìm lại số phép biến đổi P ít nhất để từ bảng A có thể biến đổi thành bảng B.

Dữ liệu vào từ file văn bản 'BDBANG.IN':

Dòng đầu là số nguyên dương N

N dòng sau, mỗi dòng N số nguyên không âm thể hiện bảng A

N dòng tiếp theo, mỗi dòng N số nguyên không âm thể hiện bảng B

Kết quả ghi ra file văn bản 'BDBANG.OUT': Nếu không thể biến đổi được (do điều kiện thời gian hoặc bộ nhớ) thì ghi -1. Nếu biến đổi được thì ghi theo qui cách sau:

Dòng đầu là số nguyên không âm K đó là số phép biến đổi ít nhất để có dãy biến đổi $A=A_0 \rightarrow A_1 \rightarrow A_2 \rightarrow ? \rightarrow A_K = B$

Tiếp theo là một dòng trắng

Tiếp theo là K+1 nhóm dòng, mỗi nhóm là một bảng $A_i$ ($0 \le i \le K$), giữa hai nhóm cách nhau một dòng trắng.

Ví dụ:
BDBANG.IN
3
2 8 3
1 6 4
7 0 5
1 2 3
8 0 4
7 6 5
BDBANG.OUT
5

2 8 3
1 6 4
7 0 5

2 8 3
1 0 4
7 6 5

2 0 3
1 8 4
7 6 5
0 2 3
1 8 4
7 6 5

1 2 3
0 8 4
7 6 5

1 2 3
8 0 4
7 6 5

***Chương trình***.

```
uses crt;
const fi = 'bdbang.in';
fo = 'bdbang.out';
max = 5;
dktamchapnhanvn = 25;
limit = 200;
dxy : array[0..3,0..1] of integer = ((0,-1),(-1,0),(1,0),(0,1));
type item = array[0..max,0..max] of integer;
m1 = array[0..limit] of item;
m2 = array[0..limit] of integer;
var a,b : item;
```

```
l,kq : m1;
depth,pre : m2;
top,n,ok,t : integer;

function cmp(x,y : item): integer; {so sánh hai bảng x và y}
var i,j : byte;
begin
cmp := 0;
for i:=0 to n-1 do
for j:=0 to n-1 do
if x[i,j]<>y[i,j] then exit;
cmp := 1;
end;

procedure ađ(x : item); {Nạp thêm bảng x vào stack L }
begin
inc(top);
l[top] := x;
end;

procedure get(var x : item); {Lấy khỏi đỉnh stack L một bảng gán cho x}
begin
x := l[top];
dec(top);
end;

procedure read_inp;
var i,j : integer;
f : text;
begin
assign(f,fi);
reset(f);
readln(f,n);

for i:=0 to n-1 do {Đọc bảng nguồn}
begin
for j:=0 to n-1 do read(f,a[i,j]);
readln(f);
end;

for i:=0 to n-1 do {Đọc bảng đích}
begin
for j:=0 to n-1 do read(f,b[i,j]);
readln(f);
end;
close(f);
```

```
end;

procedure pos0(w: item;var x,y : integer); {Tìm toạ độ x và y của ô 0 trong bảng}
var i,j : integer;
begin
for i:=0 to n-1 do
for j:=0 to n-1 do
if w[i,j]=0 then
begin
x := i;
y := j;
exit;
end;
end;

procedure swap(var x,y : integer); {Tráo đổi hai giá trị x và y}
var c : integer;
begin
c := x;
x := y;
y := c;
end;

{Duyệt theo chiều sâu, với độ sâu hạn chế tối đa là d}
procedure DLS(d : integer);
var c : item;
pre_c,k,x,y,u,v : integer;
begin
top := -1;
ađ(a); {coi bảng xuất phát là đỉnh gốc của cây tìm kiếm}
depth[top]:=0; {coi độ sâu của đỉnh xuất phát là 0}
kq[0] := a; {mảng ghi nhận các bảng thu được trong quá trình biến đổi }
while (top>=0) do
begin
if top=-1 then break;
t := depth[top]; {bước biến đổi thứ t = độ sâu của bảng đang xét}
pre_c := pre[top]; {hướng của biến đổi bảng trước (bảng 'chá) thành bảng đang xét}
get(c); {c: bảng đang xét}

kq[t] := c; {ghi nhận bảng thu được ở bước thứ t}
if (cmp(c,b)=1) then {nếu c là bảng đích thì dừng tìm kiếm}
begin
ok := 1;
break;
end;
if (t<=d) then {nếu độ sâu t chưa vượt quá giới hạn độ sâu là d thì duyệt tiếp}
```

```
begin
pos0(c,x,y); {tìm ô 0 }
for k:=0 to 3 do {Khởi tạo các hướng đi tiếp}
if (t=0) or (k<>3-pre_c) then {nếu là đỉnh gốc thì chọn cả 4 hướng, nếu không là đỉnh gốc thì
không được chọn hướng đi về bảng 'chá của bảng đang xét để tránh lặp lại bảng đã qua}
begin
u := x + dxy[k, 0];
v := y + dxy[k, 1];
if (u>=0) and (v>=0) and (u
begin
swap(c[x,y],c[u,v]); {thực hiện biến đổi: đổi chỗ ô 0 và ô kề nó theo hướng k }
ađ(c); {nạp bảng mới sau khi biến đổi}
depth[top] := t+1; {độ sâu của bảng vừa nạp vào stack}
pre[top] := k; {hướng biến đổi đãsinh ra bảng mới này}
swap(c[x,y],c[u,v]); {quay lui, trả lại bảng trước khi biến đổi}
end;
end;
end;
end;
end;
{Thực hiện duyệt theo chiều sâu với độ sâu hạn chế được tăng dần cho tới khi vượt độ sâu
giới hạn thì đành chấp nhận là vô nghiệm}
procedure depth_deepening_search;
var d : integer;
begin
d := -1;
repeat
inc(d);
dls(d);
until (ok=1) or (d>dktamchapnhanvn);
end;

procedure print_item(var f: text;w : item); {ghi một bảng vào file output}
var i,j : integer;
begin
for i:=0 to n-1 do
begin
for j:=0 to n-1 do write(f,w[i,j],' ');
writeln(f);
end;
end;

procedure output; {ghi kết quả vào file output}
var f : text;
k : integer;
begin
```

```
assign(f,fo);
rewrite(f);
if ok<>1 then
begin
write(f,-1);
close(f);
halt;
end;
writeln(f,t);
writeln(f);
for k:=0 to t do
begin
print_item(f,kq[k]);
writeln(f);
end;
close(f);
end;

BEGIN
read_inp;
Depth_deepening_search;
output;
END.
```

Nhược điểm của duyệt với độ sâu hạn chế là: nếu nghiệm chỉ có thể gặp ở độ sâu khá sâu thì sẽ phí mất nhiều thời gian cho các phép duyệt lại với độ sâu mới chưa đến độ sâu của nghiệm. Đặc biệt trường hợp vô nghiệm thì phương pháp này tồi tệ nhất không nên sử dụng. Trường hợp tốt đẹp nhất với duyệt độ sâu hạn chế là nghiệm ở độ sâu nhỏ, trong khi cây tìm kiếm theo DFS có nhiều lá rất xa gốc (độ sâu lớn) ở vị trí phải duyệt trước nghiệm.
Bài toán còn có thể giải bằng duyệt với tổ chức Heap (để lấy đề cử nhanh chóng hơn) hoặc duyệt theo phương pháp 'leo đồi': khi chọn đề cử dựa vào hàm ước lượng đánh giá sự gần gũi của đề cử với bảng đích B, chọn lấy những đề cử tốt nhất theo dự báo của hàm ước lượng để thử duyệt tiếp.
Tuy nhiên việc đánh giá hơn kém giữa các cách giải khác nhau của bài toán này còn phụ thuộc nhiều vào bộ dữ liệu được chọn. Chúng tôi rất mong được trao đổi cùng các độc giả những chương trình hữu hiệu giải bài toán này với kích thước lớn hơn.

## Những nhận xét bất ngờ khi giải một bài toán tin

**Nguyễn Xuân Huy**

Người Pháp rất tự hào với ngôn ngữ của mình. Họ nhất quyết không chịu mượn thuật ngữ Computer theo tiếng Anh để nói về máy tính. Họ gọi máy tính là Ordinateur (đọc là or-đi-na-tơ). Từ này có nghĩa là *người tổ chức trật tự*. Càng ngẫm càng thấy người Pháp có lý. Hầu hết các vấn đề của tin học đều được giải quyết trên cơ sở xây dựng *một quy trình*? *một trật tự xử lý*. Trật tự đó có thể là dãy dữ liệu được sắp tăng hoặc giảm, trật tự xử lý các đỉnh hoặc cạnh trong đồ thị, thí dụ như ưu tiên theo chiều rộng hoặc chiều sâu, hoặc một trật tự xử lý

mang lại kết quả tối ưu trong các bài toán tham lam. Để có được trật tự tối ưu chúng ta cần quan sát và phân tích chu đáo để đưa ra những nhận xét về tính chất của các đối tượng và quan hệ giữa các đối tượng cần xử lý. Đôi khi ta phát hiện ra những nhận xét hết sức bất ngờ và thú vị. Dưới đây là một số nhận xét tác giả thu lượm được từ các bạn học sinh phổ thông. Xin chia sẻ với bạn đọc.

**Bài 1 - Độ cao**.

*Độ cao* của một số tự nhiên là *tổng các chữ số* của số đó. Cho trước hai giá trị *n* và *h*. ký hiệu *S*(*h,n*) là số lượng các số tự nhiên độ cao *h* và có không quá *n* chữ số. Thí dụ, *S*(2,3) = 6 với các số 2, 11, 020, 101, 110, 200.

***Phân tích**:* Ta thấy *S* là *hàm 2 biến*. Trước hết hãy cho *h* và *n* các giá trị cận dưới. Ta thấy,

Nhận xét 1: *S*(0, n) = 1, nếu n > 0; *S*(*h*,0) = 0.

Nhận xét 2: Độ cao tối đa của số *n* chữ số là 9**n*, do đó ta đặt *S*(*h,n*)=0 nếu *h* > 9*n.

Nhận xét 3: *S*( *h*,1) = 1 với $0 \le h \le 9$.

Nhận xét 4: Nếu 101 là số độ cao h = 2 thì 999-101 = 898 là số độ cao h = 27-2 = 25. Tổng quát hóa nhận xét này ta thu được công thức *S*(*h,n*) = *S*(9**n*-*h,n*). Như vậy, theo thí dụ trên ta tính được

*S*(25,3) = *S*(3*9-25,3) = *S*(2,3) = 6 với các số 997, 988, 979, 898, 889, 799 (bạn nhớ thêm 0 vào bên trái các số cho đủ *n* chữ số).

Rõ ràng là tính *S*(2,3) sẽ dễ chịu hơn *S*(25,3). Vậy thì trước hết ta chỉnh lại giá trị của tham số *h* trong *S*(*h,n*) như sau:

**if h > 9*n div 2 then h := 9*n - h**;

Nhận xét 5: Để tính *S*( *h,n*) ta qui định bù các số 0 vào bên trái sao cho mọi số đều có đúng *n* chữ số và dĩ nhiên chúng có cùng độ cao *h*. Ta chia tập toàn thể các số đó thành 10 lớp không giao nhau căn cứ vào chữ số cuối cùng? chữ số thứ *n*, đó là chữ số hàng đơn vị. Gọi chữ số đó là *i*, ta thấy, tổng *n-1* chữ số còn lại sẽ là *h-i*. Như vậy số lượng các số thuộc lớp *i* (có chữ số cuối cùng là *i*) sẽ là *S*(*h-i,n-1*). Dĩ nhiên *S*(*h-i,n-1*) sẽ có nghĩa khi $h\text{-}i \ge 0$ và *n-1* > 0. Ta thu được công thức sau:

Vậy là chỉ cần dùng một mảng một chiều S[0..MN] xử lý theo n bước lặp: bước lặp thứ j sẽ tính cho các số có không quá j chữ số. Nên chọn MN = 9*9 = 81, đó là độ cao tối đại của số có 9 chữ số. Tại bước j ta cần tính, với mỗi k = h..1 giá trị S[k] là số lượng các số độ cao k có không quá j chữ số. Bạn đọc tự lý giải vì sao phải tính ngược các giá trị S [k]. Nguyên tắc chung để xác định trật tự xử lý là: *không ghi vào nơi chưa xử lý*. Bạn thấy giống y chang với nguyên tắc lau nhà: *Không bước vào nơi đã lau*.

Hàm S(h,n) sẽ được triển khai như sau:

```
if h > 9*n div 2 then h := 9*n - h;
fillchar(S,sizeof(S),0);
{Khởi trị cho các số có 1 chữ số}
for i:=0 to 9 do S[i]:=1;
for j:=2 to n do
for k:=h downto 1 do
for i:=1 to min(k,9) do
S[k]:=S[k]+S[k-i];
```

**Bài 2 - Số đẹp**

Số tự nhiên hệ đếm *b* có 2*k* chữ số được gọi là số đẹp loại (*b,k,h*) nếu tổng *k* chữ số đầu bằng tổng *k* chữ số cuối và bằng *h*. Với ba giá trị *b, k* và *h* cho trước hãy cho biết có bao nhiêu số

đẹp loại (*b,k,h*). Thí dụ, [1][0 ][19][0 ][0][20 ] là một số đẹp loại (25,3,20), trong đó mỗi chữ số * của hệ đếm 25 được ghi trong cặp ngoặc: [*].
**Gợi ý**: Mỗi số đẹp loại ( *b,k,h*) được ghép bởi 2 số *x* và *y*, trong đó *x* là số độ cao *h* có đúng *k* chữ số, *y* là số độ cao *h* có không quá *k* chữ số (nếu cần ta thêm 0 ở đầu trái số *y*). Hãy thực hiện đoạn trình trên một lần duy nhất để thu được ngay kết quả.
**Bài 3 - Chia đoạn** *(Đề thi Tin học Quốc gia Bulgary 2004)*
Cho hai dãy số nguyên dương *a*[1..*n*] và *b*[1.. *m*]. Hãy chia mỗi dãy thành *k* đoạn, ký hiệu lần lượt từ trái qua phải là 1, 2,..., *k*, mỗi đoạn phải chứa ít nhất một phần tử của dãy tương ứng. Với mỗi đoạn i = 1, 2,..., *k* mso-ansi-language: VI'>hãy tính hàm
$C(i) = (t(a,i) ? s(a,i))*(t(b,i)-s(b,i))$
Trong đó
$t(a,i)$ là tổng các phần tử của dãy *a* trong đoạn *i*,
$s(a,i)$ là số lượng phần tử của dãy *a* trong đoạn *i*,
$t(b,i)$ là tổng các phần tử của dãy *b* trong đoạn *i*,
$s(b,i)$ là số lượng phần tử của dãy *b* trong đoạn *i*.
Cuối cùng ta tính trị

Hãy cho biết giá trị nhỏ nhất của *C*?

Thí dụ, với *a* = (1,2,3), *b* = (1,2), nếu chọn *k* =2 và chia hai dãy *a* và *b* như sau:
*a* = ((1,2),(3)), *b* = ((1),(2)) thì
$C(1) = ((1+2)-2)*(1-1) = 1*0 = 0$,
$C(2) = (3-1)*(2-1) = 2*1 = 2$,
$C = C(1)+C(2) = 0+2 = 2$.
***Phân tích***:
Nhận xét 1: Nếu giảm mỗi phần tử của hai dãy a và b đi 1 đơn vị thì công thức tính C(i) sẽ được rút gọn như sau:
$C(i) = t(a,i) *t( b,i)$
Bạn hãy tự giải thích điều này.
Nhận xét 2: Kí hiệu F(i,j) là trị tối ưu của hàm C đối với hai dãy a[1..i] và b[1..j]. Ta xét đoạn cuối cùng, đoạn thứ k. Giả sử đoạn cuối cùng này chứa p1 phần tử của dãy a với tổng là s1 và p2 phần tử của dãy b với tổng là s2. Khi đó
F(i,j) = min {F(i-p1,j-p2) + s1*s2}
Cho i, j, p1 và p2 biến thiên ta tính đươc F. Tiếp cận này đòi hỏi độ phức tạp $O(N^4)$.
Nhận xét 3: Với mọi đoạn k, p1 và p2 không thể đồng thời lớn hơn 1. Thật vậy, nếu p1 > 1 và p2 > 1 thì ta có thể tách đoạn k thành hai đoạn con k1 và k2 với các tổng các phần tử lần lượt cho a và b là s1 = s11+s12 và s2 = s21+s22. Khi đó
C(k) = s1*s2 = (s11+s12)*(s21+s22) = s11*s21 + s12*s22 +... ≥ s11*s21 + s12*s22 = C(k1)+C(k2).
Theo nhận xét này, mỗi đoạn trong phép chia tối ưu chỉ có thể là một trong 3 dạng sau:
i) chứa 1 phần tử của a và 1 phần tử của b,
ii) chứa 1 phần tử của a và p2 > 1 phần tử của b,
iii) chứa p1 > 1 phần tử của a và 1 phần tử của b,
Vậy ta có
F(i,j) = min {F(i-1,j-1) + a[i]*b[j], F(i-1,j-p2)+ a[i]*s2, F(i-p1,j-1)+ s1*b[j] }
Nhận xét 4: Ta có a[i]*s2 = a[i]*(b[i-p2+1]+...+b[j-1]+b[j]) = a[i]*(b[i-p2+1]+...+b[j-1]) +

a[i]*b[j], do đó, F(i-1,j-p2)+a[i]*s2 ≥ F(i,j-1) + a[i]*b[j].
Và tương tự, F(i-p1,j-1)+s1*b[j] ≥ F(i-1,j) + a[i]*b[j].
Vậy ta có,
F(i,j) = min {F(i-1,j-1) + a[i]*b[j], F(i,j-1)+ a[i]*b[j], F(i-1,j)+ a[i]*b[j] }
Đưa hạng tử amso-ansi-language: VI'>[i]*b[j] ra ngoài ta được một hệ thức thật đẹp và dễ tính vì chỉ cần độ phức tạp $O(N^2)$.
F(i,j) = min {F(i-1,j-1), F(i,j-1), F(i-1,j)} + a[i]*b[j]
Bạn thấy trật tự tính tóan đã giảm đáng kể độ phức tạp, từ bậc 4 xuống bậc 2.
**Gợi ý**: Để tính hàm F theo công thức trên cần một mảng hai chiều. Liệu bạn có cách nào thay bằng hai mảng một chiều?
**Bài 4 - Số duy nhất** *(Đề thi Tin học các quốc gia Baltic, BOI 2004)*
Cho N số tự nhiên trong đó có duy nhất một số x xuất hiện đúng 1 lần, các số còn lại đều xuất hiện đúng K > 1 lần. Tìm số duy nhất x?
Thí dụ, với 7 số 123, 123, 45, 123, 45, 405, 45 ta phải tìm được x = 405.
Nhận xét 1: Nếu K là số chẵn thì ta dùng phép toán XOR bit N số đã cho, kết quả sẽ cho ngay số x cần tìm. Tuy nhiên bài toán không cho biết K là bao nhiêu.
Nhận xét 2: Ta có thể tìm được số K bằng cách đếm số lần xuất hiện của số đầu tiên (kí hiệu là x) trong dãy. Nếu K=1 thì ta có ngay x là số cần tìm. Ngược lại, nếu K > 1 thì đó chính là số lần xuất hiện của các số còn lại. Vậy là yên tâm.
Nhận xét 3: Với hai nhận xét đầu tiên chúng ta dễ bị tắc trong mê cung tìm kiếm. Hãy thử suy nghĩ về trật tự xem sao. Trật tự gì? Trật tự các số không cho ta thêm giải pháp nào. Ta thử tìm cách giải theo trật tự của các chữ số. Đúng vậy, từ hàng ngàn năm trước các nhà toán học cổ Hy Lạp như ácsimet và Ơclid đã nói rằng cùng là chữ số 1 nhưng nếu đặt ở vị trí khác nhau thì chữ số đó mang các giá trị khác nhau, khi thì là 1, khi là 10, khi là 100. Như vậy chúng ta phải đếm xem trong N số đã cho các chữ số trong từng hàng xuất hiện bao nhiêu lần. Ta dùng một mảng hai chiều a để ghi nhận các số đếm, a[c,j] cho biết số lần xuất hiện của chữ số c tại vị trí j. Ta thấy a[c,j] chỉ có thể chia hết cho K hoặc chia cho K dư 1. Bạn hãy giải thích điều này.
Vì chỉ quan tâm đến các chữ số nên ta có thể đọc các dòng của tệp **input** vào các biến string. Ngoài ra ta cũng có thể xử lý các chữ số theo các vị trí tính từ phải qua trái.
Ta giả thiết là số dài nhất trong dãy đã cho có MN=20 chữ số.

```
MN = 20;
var a: array['0'..'9',1..MN] of word;
x,y: string;
c: char;
N,K: word;
f,g: text;
1. Mở các tệp inpyt f và output g.
fillchar(a,sizeof(a),0);
readln(f,N);
readln(f,x); K:=1;
for j:=1 to length(x) do inc(a[x[j],j]);
for i:=2 to N do
begin
readln(f,y);
if y=x then inc(K);
```

```
for j:=1 to length(y) do inc(a[y[j],j]);
end;
if K > 1 then
begin
x:='';
for j:=1 to MN do
for c:='0' to '9' do
if a[c,j] mod K <> 0 then
begin
x:=x+c;
break;
end;
end;
writeln(g,x);
close(f); close(g);
```

**Bài 5 - Dãy ký tự duy nhất**

Cho N xâu kí tự trên bảng chữ cái a..z trong đó có duy nhất một xâu x xuất hiện đúng M lần, các xâu còn lại đều xuất hiện đúng K > M lần. Tìm xâu x?

*Bài sau đây các bạn chỉ nên giải vào chiều 30 tết.*

**Bài 6 - Qua cầu** *(Tổng quát hóa để thi tuyển nhân viên của hãng Microsoft)*

Một đàn con nít đi chơi đêm giao thừa với 1 chiếc đèn lồng. Gặp một cây cầu mỏng manh. Các em phải đưa nhau qua cầu. Cầu yếu, chỉ chịu được sức nặng của 2 em mỗi lượt, ngoài ra, cầu trơn và gập gềnh nên buộc phải soi đèn mới đi được. Mỗi em qua cầu với thời gian khác nhau. Hãy tìm cách để các em qua cầu nhanh nhất. Giả thiết cho n em với thời gian vượt cầu của mỗi em lần lượt là t1, t2,?,tn.

Thí dụ, có 4 em với thời gian vượt cầu lần lượt là 1, 2, 5 và 10 phút. Theo phương án sau đây ta phải cần 19 phút: Cho em nhanh nhất? em số 1 cầm đèn dẫn tường người qua cầu:

1. Em 1 dẫn em 2: 2 ph.
2. Em 1 cầm đèn về: 1 ph.
3. Em 1 dẫn em 3: 5 ph.
4. Em 1 cầm đèn về: 1 ph.
5. Em 1 dẫn em 4: 10 ph.

Nếu theo phương án sau đây ta được lời giải tối ưu: 17 ph.

1. Em 1 dẫn em 2: 2 ph.
2. Em 1 cầm đèn về: 1 ph.
3. Em 1 giao đèn cho em 3 và 4 qua cầu: 10 ph.
4. Em 2 cầm đèn về: 2 ph.
5. Em 1 dẫn em 2: 2 ph.

*Nhận xét 1*: Một em quá chậm đi cùng em quá nhanh sẽ làm 'hại em đi nhanh, do đó khi qua cầu nên cử hai em chậm nhất.

*Nhận xét 2*: Khi cầm đèn chạy về nên giao cho em đi nhanh nhất.

Chỉ với hai nhận xét trên chúng ta lập được trật tự giải bài toán trên như sau:

Gọi nơi tập hợp các em nhỏ trước khi qua cầu là T (bên trái cầu), nơi cần đến là P (bên phải cầu).

Trước hết tìm hai em đi nhanh nhất đòan là a và b.

Lặp các bước 1-6 cho đến khi T rỗng:

1. a và b qua P,
2. a cầm đèn về T,
3. a giao đền cho 2 em chậm nhất x và y ở T,
4. x và y qua P,
5. x và y giao đèn cho b ở P,
6. b trở về T.

Bạn cần lưu ý trường hợp n=1. Ngoài ra mỗi khi đến P bạn cần kiểm tra ngay xem còn ai ở T không, mỗi khi ở T bạn lại phải kiểm tra xem còn bao nhiêu người chưa qua cầu, có như vậy vòng lặp mới kết thúc đúng lúc.

Chúc các bạn một năm mới khỏe mạnh, thông minh, bấm máy là chạy.

## Cách nhìn khác đối với một số lớp bài toán quen thuộc

**Cao Minh Anh**

Các bạn đã bao giờ thích thú khi tìm ra một cách nhìn nhận mới cho những bài mình đã biết? Thật là thú vị nếu ta tìm cho mình một phương pháp khác mà tính tối ưu vẫn không giảm.

Sau đây tôi sẽ trình bày với các bạn một cách nhìn nhận khác đối với lớp bài toán như xác định hoán vị thứ k, nhị phân thứ k, …

**Tư tưởng thuật toán**

- Giả sử cần tìm 1 trạng thái nào đó gồm n phần tử của một dạng xác định. Đề bài cho biết đặc điểm của trạng thái trên là M.

- Với M ta có thể xác định phần tử đầu tiên của trạng thái, nhưng n-1 phần tử còn lại thì không xác định được. Vấn đề bây giờ là làm sao tìm được n-1 trạng thái còn lại. Rất đơn giản, bởi vì nếu biết được M ta sẽ xác định được phần tử đầu tiên, tại sao ta không thử n-1 phần tử còn lại có phải thuộc một trạng thái nào đó gồm n-1 phần tử có đặc điểm M1. Từ M1 này ta suy được phần tử đầu tiên của trạng thái mới, tức là đã tìm được phần tử thứ 2 của trạng thái ban đầu. Cứ làm như thế ta sẽ tìm được hết các phần tử của trạng thái ban đầu.

- Cái khó của thuật toán trên là đòi hỏi người lập trình phải tìm được cách xác định phần tử đầu tiên dựa vào đặc điểm M, thứ hai là phải tìm được các đặc điểm $M_1$, $M_2$,..., $M_{n-1}$ tương ứng với các trạng thái mới có ít phần tử hơn.

**Bài toán 1**
Xác định dãy nhị phân thứ k gồm n phần tử (n≤200).
File nhiphan.inp
n k
File nhiphan.out
$a_1$ $a_2$ $a_3$.. $a_n$ (*dãy nhị phân thứ k*)
***Ví dụ***
nhiphan.inp
4 7
nhiphan.out
0 1 1 0

***Thuật toán***

Dựa vào ví dụ dưới ta nhận thấy rằng:

- Có 1 nửa trạng thái đầu tiên có số 0 đứng đầu, còn nửa còn lại là số 1 đứng đầu. Hay có $2^{n-1}$ trạng thái đầu tiên bắt đầu là số 0, còn lại $2^{n-1}$ trạng thái bắt đầu là số 1. Như vậy chỉ cần so sánh k với $2^{n-1}$ thì ta có thể biết được phần tử đầu tiên của dãy nhị phân thứ k là 0 hay 1.

If k>$2^{n-1}$ then a[1]=1 else a[1]=0.

- Vấn đề còn lại là làm sao xác định được n-1 phần tử còn lại.

- Ta nhận thấy rằng nếu phần tử đầu tiên là 0 thì n-1 phần tử còn lại giống với các phần tử của dãy nhị phân thứ k có n-1 phần tử, còn nếu phần tử đầu tiên là 1 thì n-1 phần tử còn lại giống với các phần tử của dãy nhị phân thứ (k-$2^{n-1}$) có n-1 phần tử. Bây giờ ta lại làm bài toán nhỏ hơn là xác định dãy nhị phân thứ k' nào đó gồm n-1 phần tử. Cứ làm như thế ta sẽ tìm được hết các phần còn lại.

Từ những nhận xét trên ta có chương trình đơn giản như sau:
```
For I:=n downto 1 do
Begin
If k>1 shl (I-1) then
Begin
Write('1');
K:=k-(1 shl (I-1));
End
Else write('0');
End;
```
Vì bài này dữ liệu lớn, ta chỉ cần làm thêm chương trình nhân số lớn để tính $2^h$, và hàm so sánh 2 xâu k với $2^h$.

**Bài 2**
Tìm hoán vị thứ k gồm n phần tử (n≤200).

***Thuật toán***

***Nhận xét***
- (n-1)! Trạng thái đầu tiên có phần tử đầu tiên là 1, tiếp theo (n-1)! Tiếp là 2, tiếp tới là 3, và cuối cùng là 4.
- Dựa vào k ta có thể xác định phần tử đầu tiên. Nếu tìm được phần tử đầu tiên là h thì dùng mảng b để đánh dấu h, để n-1 phần tử còn lại không có h.
- Bây giờ cần xác định vị trí k' của n-1 phần tử còn lại.
- K' = k-v (với v là vị trí đầu tiên bắt đầu bằng số h). Việc tính v rất dễ dàng.
Đây là phần giải tạm với dữ liệu $k \leq 2^{32}$.

```
Fillchar(b,sizeof(b),1);
T:=1;
For I:=1 to n do T:=T*I;
For I:=n downto 1
Begin
T:=T div I;
For j:=1 to n do
If b[j] then
Begin
if K>T then k:=k-T
else
begin
write(j:4);
b[j]:=false;
break;
end;
end;
end;
```

Ta có thể nâng dữ liệu lên $n \leq 1000$, chỉ cần làm chương trình nhân số lớn và hàm so sánh hai số lớn là xong.

**Bài 3**
(*Đề 2 - dãy nhị phân olympic tin học sinh viên khối chuyên tin*)

Xét tập S các dãy nhị phân độ dài N trong đó không có hai bit 1 nào kề nhau. Các dãy này được xếp theo chiều tăng dần của số nguyên mà nó biểu diễn, theo thứ tự đó mỗi dãy có một số hiệu, chẳng hạn n=5.

Cho số nguyên dương $N \leq 100$ hãy nhị phân có số hiệu M.
Ví dụ
BINSEQ.inp
5 5

BINSEQ.out
0 0 1 0 1

***Thuật toán***
Đây là một bài rất khó để ta xác định cách xác định phần tử đầu tiên.

*Nhận xét* :
Với n=5 có 13 dãy tất cả. Trong đó có 8 dãy đầu tiên bắt đầu bằng 0,5 dãy còn lại bắt đầu bằng 1. Với n=6 thì có tất cả 21 dãy, 13 cái đầu là 0,8 cái sau là 1. Ta thấy nó có liên quan đến số fibonaci.

Chỉ cần kiểm tra M với a[n-1], nếu M>a[n-1] thì đó là số 0 đầu tiên, ngược lại là số 1 đầu tiên. Nếu số đầu tiên là 0 thì trạng thái mới cần tìm gồm n-1 phần tử có số hiệu vẫn là M, ngược lại có số hiệu là M-a[n-1].

Bài làm rất đơn giản như sau:

```
a[0]:=1;a[1]:=1;
For I:=2 to n do a[I]:=a[I-1]+a[I-2];(Tạo các số fibo)
For I:=n downto 1 do
Begin
If M>a[I-1] then
Begin
Write('1');
M:=M-a[I-1];
End
Else write('0');
End;
```

Thật là đơn giản phải không các bạn? Vấn đề ở chỗ phải tìm ra đặc điểm để xác định phần tử đầu tiên và đặc điểm của các phần tử còn lại. Hi vọng các bạn sẽ nhận thêm được kinh nghiệm nào đó từ bài viết này.

## Quan hệ sinh dữ liệu và tiếp cận Quy hoạch động

**Lê Đình Thanh**

Chúng ta đã biết quy hoạch động (QHĐ) là một phương pháp giải toán rất hiệu quả một khi nó được sử dụng. Đúng như tác giả Nguyễn Thanh Tùng - trong bài "Một số bài toán quy hoạch động kinh điển", Tạp chí Tin học và Nhà trường Số 1, Năm 2005- nhận xét: điều khó nhất để giải một bài toán bằng QHĐ là biết rằng nó có thể giải theo QHĐ và tìm được công thức QHĐ của nó. Nói cách khác, vấn đề khó nhất của việc giải một bài toán QHĐ là tìm *dấu hiệu nhận biết QHĐ* và tìm*quy luật quy hoạch dữ liệu* của bài toán đó. Trong bài báo này, tôi đưa ra một cách tiếp cận theo *quan hệ sinh dữ* liệu để giải quyết hai vấn đề nói trên. Đây không phải là cách tiếp cận tối ưu nhưng nó giải quyết được một lớp lớn các bài toán quy hoạch động.

Các góp ý xin gửi về địa chỉ: thanhld_fit_hdu@yahoo.com

**1. Quan hệ sinh dữ liệu tuyến tính và khả năng quy hoạch động**

Nhận xét sau xuất phát từ chính định nghĩa về quy hoạch động:

Giả sử P là bài toán đang cần giải quyết trên mẫu dữ liệu D, và giả thiết từ D ta có thể tạo ra các mẫu dữ liệu $D_i$, (i = 1, 2...) là mẫu con của D và đồng dạng với D. Ví dụ, D là dãy số tự nhiên và Di là các dãy con của nó, D là mã vạch có chiều dài L và $D_i$ là các mã vạch có chiều dài nhỏ hơn L.

Cũng áp dụng yêu cầu của bài toán trên các mẫu $D_i$ ta được các bài toán $P_i$ (i = 1, 2...) cùng dạng với P và nhỏ hơn P. $P_i$ là các bài toán con của P. Gọi $O_i$ (i = 1, 2...) là *dữ liệu được sinh ra* bởi $P_i$, và gọi O là dữ liệu được sinh ra bởi P. Đặt G = {O} U {$O_i$, i = 1, 2...}. Nếu trên G ta lập được 2 quan hệ là *quan hệ sinh* và *quan hệ thứ tự* thì bài toán P có thể giải quyết bằng QHĐ, trong đó quan hệ sinh là khả năng dữ liệu được sinh của một bài toán này có thể tạo ra từ các dữ liệu sinh của các bài toán khác và quan hệ thứ tự là tính khả sinh (hiểu theo nghĩa toán học).

Ta hình dung mỗi $O_i$ hay O như một đỉnh của đồ thị và $O_j$ là một trong các dữ liệu ra được dùng để sinh $O_k$ ($O_k$ đứng sau $O_j$ trong quan hệ thứ tự) được biểu diễn bằng một cung có hướng từ đỉnh j đến đỉnh k, thì đồ thị được tạo là đồ thị có hướng, phi chu trình. Dữ liệu được quy hoạch trên đồ thị bắt đầu từ các đỉnh không có cung vào. Khi một đỉnh đã được xét (một bài toán con đã được giải quyết), ta xóa luôn các cung ra từ đỉnh đó. Một số đỉnh mới lại trở thành đỉnh không có cung vào. Quá trình được lặp lại cho đến khi đỉnh tương ứng với bài toán P được giải quyết.

Một trường hợp khác, từ D không tạo được các Di là con của D nhưng có thể tạo ra các *bài toán tương tự* P trên D. Gọi các bài toán tương tự đó là $Q_j$ (j = 1,2...). Nếu trên các dữ liệu sinh của các bài toán tương tự cũng có *quan hệ sinh* và *quan hệ thứ tự* thì bài toán đã cho cũng có thể giải quyết bằng giải thuật QHĐ.

**2. Phương pháp tiếp cận**

Từ phân tích ở trên, ta đưa ra các bước tiếp cận để nhận biết một bài toán QHĐ và tìm công thức quy hoạch cho nó như sau:

b1. Tạo và chọn các mẫu dữ liệu con đồng dạng với mẫu dữ liệu cần xử lý, áp dụng cùng yêu cầu của bài toán đối với những mẫu dữ liệu con đó để được các bài toán con, nếu được, hoặc sử dụng yêu cầu của bài toán trên các thể hiện khác nhau của dữ liệu để được các bài toán tương tự.

b2. Giả sử tất cả các bài toán con hoặc tương tự (kể cả bài toán ban đầu) đã được giải quyết và biết được dữ liệu sinh của chúng, tìm quan hệ sinh và quan hệ thứ tự giữa các dữ liệu sinh. Nếu tồn tại 2 quan hệ vừa nêu, giải bài toán đã cho bằng QHĐ như mô tả ở bước 3. Ngược lại, xét lại bước 1 hoặc tìm lời giải theo phương pháp khác.

b3. Phân tích và thu gọn quy tắc sinh dữ liệu. Lập quy tắc QHĐ dựa trên quy tắc sinh đã thu gọn.

**3. Các ví dụ**

***Ví dụ 1. Mã vạch***

Một mã vạch là một dãy gồm M vạch trắng và vạch đen liên tiếp, trong đó vạch đầu tiên và vạch cuối cùng là các vạch đen, độ rộng các vạch bằng nhau, đồng thời không có quá N vạch cùng một màu liền nhau. Hãy tìm chiều dài tối thiểu M để số mã vạch sinh ra không nhỏ hơn K.

***Tiếp cận***

b1. Với  mỗi số tự nhiên i, Pi là yêu cầu tính số các mã vạch có chiều dài i. Ta được một lớp bài toán tương tự. Thực hiện tiếp bước 2.
b2. Giả sử V là một mã vạch có chiều dài i, bằng cách thêm t vạch nữa vào một đầu của V, ta được một đoạn vạch có chiều dài i+t. Đoạn vạch này là một mã vạch nếu đoạn mới thêm và đoạn tiếp nối giữa mã vạch đã có với đoạn thêm thõa mãn tính chất của mã vạch. Bằng cách thêm vạch vào một đầu mã vạch để sinh mã vạch mới ta có quan hệ sinh và quan hệ sinh này thể hiện quan hệ thứ tự. Thực hiện tiếp bước 3.
b3. Ký hiệu Vi là tập các mã vạch có chiều dài i. Yêu cầu của bài toán là tìm i nhỏ nhất để card($V_i$) ≥ K, trong đó card(Vi) là lực lượng của Vi. Giả sử v là một mã vạch thuộc Vi, và giả sử sau khi thêm t vạch vào một đầu của v ta được một mã vạch u thuộc $V_{i+t}$ (hình).

Nếu cắt u tại vạch đen đầu tiên tính từ đầu vừa thêm và trừ vạch ngoài cùng ta sẽ được một mã vạch s thuộc $V_{i+j}$ (j < k). Nhu+ va^.y, u cu~ng co' the^? ddu+o+.c ta.o tu+` s ba(`ng ca'ch the^m mo^.t so^' va.ch tra('ng va` mo^.t va.ch dden ngoa`i cu`ng.
*Nhận xét trên cho ta quy tắc sinh:*
Một mã vạch có chiều dài i được tạo ra từ một và chỉ một mã vạch ngắn hơn bằng cách thêm vào một đầu một số vạch trắng cùng một vạch đen ngoài cùng.
*Tìm quy tắc QHĐ:*
Rõ ràng số vạch trắng được thêm chỉ có thể là 0, 1, …, N. Nếu có vạch trắng được thêm vào thì cách thêm luôn hợp lệ (được mã vạch mới). Trường hợp chỉ thêm một vạch đen, nếu N vạch đầu tiên ở  đầu  được  thêm của mã vạch đã có đều đen thì phép thêm không hợp lệ. Bởi vậy ta có quy hoạch:
card($V_i$) = ∑card($V_{i-1-t}$) – I, I là số các mã vạch có chiều dài i-1 và có N vạch đen ở đầu được thêm,
(t = 0, 1, …, N)
Bây giờ ta xét các mã vạch có chiều dài là i -1 và có N vạch đen ở đầu được thêm. Rõ ràng, vạch tiếp theo là vạch trắng. Số vạch trắng liên tiếp chỉ có thể là 1, 2, …, N. Giả sử có k vạch trắng liên tiếp, nếu cắt N vạch đen ở đầu và k vạch trắng kế tiếp, ta  được một mã vạch có chiều dài i-1-N-k. Nhận xét đó cho ta quy hoạch:
I = ∑card($V_{i-1-N-k}$), với k = 1, 2, …, N.
Bởi vậy, ta có công thức quy hoạch động:
card($V_i$) = ∑card($V_{i-1-t}$) - ∑card($V_{i-1-N-k}$), với t = 0, 1, …, N và k = 1, 2, …, N.
***Ví dụ 2. Dãy con đơn điệu dài nhất***
Cho dãy $a_1$,$a_2$,..$a_n$. Hãy tìm một dãy con tăng có nhiều phần tử nhất của dãy.
***Tiếp cận***
b1. Gọi L là dãy đã cho và $L_i$ (i = 1,..., n) là dãy liên tiếp các phần tử bắt đầu từ $a_1$ đến $a_i$ (L = $L_n$). P và $P_i$ là yêu cầu tìm dãy con tăng có nhiều phần tử nhất của dãy L, $L_i$ tương ứng.
b2. Giải sử S, $S_i$ là các xâu con tìm được khi giải P, $P_i$. Ta thấy $S_i$ được tạo từ $S_j$ nào đó với 0 < j < i va` $a_j \le a_i$.
b3. $S_i$ = max($S_{ij}$), trong đó $S_{ij}$ = $S_j$ + [$a_i$], 0 < j < i va` $a_j \le a_i$ và max tính theo chiều dài dãy là quy tắc sinh, đồng thời là công thức quy hoạch động.
***Ví dụ 3. Chia kẹo***
Cho dãy $a_1$, $a_2$,.. $a_n$. Tìm một dãy con của dãy đó có tổng bằng S.
***Tiếp cận***
b1. Gọi L là dãy đã cho và $L_i$ (i = 1,..., n) là dãy liên tiếp các phần tử bắt đầu từ $a_1$ đến $a_i$ (L = $L_n$). P và $P_i$ là yêu cầu tìm dãy con của dãy L, $L_i$, tương ứng, có tổng bằng S.

b2. Giả sử $A_i$ là các tổng có thể tạo ra từ dãy $L_i$. $E+a_{i+1}$ sẽ là một tổng có thể tạo từ $L_{i+1}$ nếu E thuộc $A_i$. $a_{i+1}$ cũng là một phần tử thuộc $A_{i+1}$.
b3. $A_1 = \{a_1\}$, từ $A_i$ sinh $A_{i+1}$, i = 1,..., n-1, nếu S chưa có trong $A_i$ là quy tắc sinh cũng là quy tắc quy hoạch động.

***Ví dụ 4. Tìm đường đi ngắn nhất trong đồ thị***

Cho đơn đồ thị có trọng G gồm N đỉnh. Hãy tìm đường đi ngắn nhất từ đỉnh u đến đỉnh v.

***Tiếp cận***

b1. Giả sử các đỉnh được đánh số từ 1 đến N và hai đỉnh u, v có thứ tự tương ứng là i, j. Nếu gọi bài toán tìm đường đi ngắn nhất từ đỉnh i đến đỉnh j là $P_{ij}$ hoặc đơn giản là $P_j$ thì các bài toán tìm đường đi ngắn nhất từ đỉnh i đến đỉnh k là $P_{ik}$ hoặc $P_k$. Rõ ràng ta được N bài toán tương tự $P_t$, t = 1,.., N.
b2. Giả sử $C_t$ là tổng chiều dài của đường đi ngắn nhất từ đỉnh i đến đỉnh t và k là đỉnh liền trước t trong đường đi đó. Dễ thấy đường đi ngắn nhất từ i đến t bỏ đi cung cuối cùng (cung nối đỉnh k với đỉnh t) là đường đi ngắn nhất từ đỉnh i đến đỉnh k. Bởi vậy, $C_t = C_k + w(k, t)$, trong đó w(k, t) là trọng số cung (k,t).
b3. Quy tắc sinh và quy hoạch
$C_i = 0$,
Từ tập các đỉnh đã biết đường đi ngắn nhất xuất phát từ i, tìm đỉnh kế tiếp các đỉnh đó có tổng đường đi ngắn nhất đến đỉnh đã biết và độ dài cung nối tương ưng nhỏ nhất:
T – tập các $C_k$ đã biết, $C_t = \min(C_k + w(k, t))$.

## Chia sẻ một thuật toán hay

**Nguyễn Duy Hàm**

**1. Đặt vấn đề.**

Trong rất nhiều bài toán khi đọc lên yêu cầu ta đều có ngay thuật toán để giải quyết nó, song các thuật toán phát sinh đó nếu không phân tích kĩ và tinh tế nhận dạng thì chỉ giải quyết được khi kích cỡ dữ liệu đầu vào là khá nhỏ có nghĩa là chỉ giải quyết được bài toán trong những trường hợp rất hạn chế. Trong lớp các bài toán như vậy ở đây tôi xin giới thiệu một số bài toán khác nhau song cách giải quyết lại khá giống nhau.

**2. Bài toán.**

*Bài toán 1:*

Cho file văn bản gồm rất nhiều dòng, mỗi dòng 80 kí tự. Các kí tự của file văn bản gồm "*" và "." Các kí tự "*" trong văn bản tạo nên các chữ cái T,E,F,I. hãy đếm các số lượng các chữ cái T,E,F,I có trong văn bản giả sử dữ liệu vào là đúng.

Ví dụ file input như sau:

…….********……*……..**********……*********
…….*……..*……*……*……..
…….*******...……..*……*……******….******…..
…….*……..*……*……..*……..
…….*…..……*……..**********

Thì có 2 chữ T,1 chữ E, 1 chữ F, và 1 chữ I.

*Bài toán 2:*

Bài toán tương tự mà tôi muốn giới thiệu ở đây là bài toán trong kì thi OLIMPIC Tin học Sinh viên Việt Nam lần thứ 14 năm 2005 tại TP.Hồ Chí Minh vừa qua. Đây là bài toán không khó, nhưng để vượt qua hết test của BGK lại là một điều không dễ. Bài toán đại ý như sau:

Ảnh chụp mặt người sau khi đã xử lý là 1 bảng vuông A kích thước NxN (10 < N < 800). Với mỗi ô (i,j) có giá trị từ 0 đến 255 là mức xám của ảnh tại ô này (trong đó 0 là màu nền). Để xác định vị trí là mặt người, ta cần thống kê các đặc trưng có dạng hình vuông kích thước k x k (1 < K ≤ 40) trong đó các giá trị trong hình vuông đều phải khác 0.
**Yêu cầu** : Từ 1 ảnh chụp mặt người đếm tất cả các đặc trưng có trong ảnh đó.
**3. Cách giải quyết.**
Với bài toán thứ nhất ta thấy rằng việc đọc dữ liệu vào một mảng 2 chiều là điều không thể, vì không biết trước số dòng của văn bản sẽ là bao nhiêu? bộ nhớ có đủ chỗ hay không? Mà nếu có đọc được đi chăng nữa thi chắc gì đã có cách nào hay để giải! Ta hãy đi tìm các đặc trưng của các chữ T,E,F,I. Ta thấy rằng các chữ cái này đều có 1 trục thẳng đứng. Cụ thể với chữ I có 1 trục thẳng không có gạch nằm ngang nào, với chữ T có 1 gạch nằm ngang, chữ F có 2 gạch nằm ngang, chữ E có 3 gạch… như vậy chữ I có 1 nét, chữ T có 2 nét, chữ F có 3 nét và chữ E có 4 nét. Ta sẽ đếm số nét của chữ để xác định đó là chữ gì!
Để đếm các nét của chữ ta sử dụng 1 mảng 1 chiều d[1..80], ban đầu ta clear các giá trị của mảng này=0, ta sử dụng 1 mảng dem[1..4] để đếm số lượng các chữ cái, trong đó dem[1] là số lượng chữ I, dem[2] là số lượng chữ T, dem[3] là số lượng chữ F và dem[4] là số lượng chữ E. Ta sẽ xử lý 2 dòng (st1,st2) liền nhau trong văn bản:
Ta xét các trường hợp sau:
1.(St1[i]='*'^st1[i+1]='*'^St2[i]='*'^st2[i+1]='.')->d[i]:=d[i]+1
2.(St1[i-1]='.'^St1[i]='*'^st1[i+1]='*'^St2[i]='.'^d[i]<>0)->d[i]:=d[i]+2;
3.(St1[i-1]='.'^St1[i]='*'^st1[i+1]='.'^St2[i]='.') -> d[i]:=d[i]+1
Trường hợp 2 và 3 là các trường hợp kết thúc của 1 chữ cái, do vậy ta sẽ tăng dem[d[i]] lên 1 trong các trường hợp đó, đồng thời d[i]:=0;Cứ như vậy cho đến khi kết thúc file và giá trị của dem[1..4] sẽ cho ta số lượng các chữ I,T,F,E đếm được. Phải chú ý rằng khi đọc hết file input thì st1 chứa dòng cuối cùng và st2 chứa 1 chuỗi các kí tự '.' để xét lần cuối cùng. Như vậy bài toán 1 đã được giải quyết.
Với bài toán thứ 2: Cách đơn giản nhất mà ta nghĩ đến khi đọc đề bài có thể là cách áp tất cả các hình vuông KxK lên bảng vuông NxN rồi xét và đếm. cách tốt hơn nữa là trước khi áp hình vuông ta có thể đánh dấu các ô có giá trị 0 và các ô nằm trên cùng 1 hình vuông KxK với ô đó, và cách tốt hơn cách này nữa là hình vuông kế tiếp được xét bằng cách bớt cột bên trái thêm cột bên phải, hoặc bớt hàng bên trên thêm hàng bên dưới như vậy ta chỉ cần xét trên các cột thêm hoặc hàng thêm này mà thôi…song chúng ta phải chú ý rằng bài này đòi hỏi chạy trong 2 giây và dữ liệu tối đa là 800x800. Thực tế dữ liệu 800x800 đã là 1 gợi ý của bài toán để chúng ta biết rằng không thể đọc toàn bộ bảng vuông vào mảng 2 chiều được, nhưng như vậy lại có bạn đọc từng phần 1 để xử lý, tuy nhiên nếu thế lại không đáp ứng được thời gian chạy của chương trình.Và cách giải quyết ở đây là hoàn toàn đơn giản, chúng ta áp dụng như các giải quyết của bài TEFI ở trên.
Như vậy ta sẽ đọc từng dòng của bảng A vào chuỗi st và sử dụng mảng d[1..801] (phần tử thứ 801 dùng làm lính canh) kiểu word để xử lý. Giá trị d[i] sẽ là số các ô ***khác 0 liên tục tính từ dòng đang xét trở lên ở cột i*** của bảng A. Ban đầu ta clear các gía trị của mảng d[1..801].
//Khởi tạo ban đầu
**Procedure** Khoitao;
**begin**
Fillchar(a,sizeof(d),0);
dem:=0;
d[n+1]:=0;//phần tử lính canh

```
end;
//Xử lý dữ liệu đọc vào
Procedure Xuly;
Var x,j,i : word;
Begin
Assign(f,'xulyanh.out');
Reset(f);
For j:=1 to n do
Begin
For i:= 1 to n do
Read(f,x);
If x<>0 then inc(d[i])
Else d[i]:=0;
End;
If j>=k then Duyet;
readln(f);
End;
Close(f);
End;
```

//Xử lý mảng trên d được viết như sau:

```
Procedure Duyet;
Var j,i : word;
Begin
For i:= 1 to n-k do
Begin
J:=0;
While (d[i+j]>k ) and (j<=k) do inc(j);
If j>k then inc(dem);
End;
End;
```

Xử lý dữ liệu trên mảng d theo cách viết trên là chưa tối ưu, chỉ mang tính mô tả để ta dễ hiểu cách làm. Ta nên viết lại Procedure Duyet như sau:

```
Procedure Duyet;
Var j,i : word;
Begin
i:=1;
Repeat
While d[i] < k do inc(i);
if i < n-k then
Begin
j:=0;
While (d[i+j]>=k) and (j<=k) do inc(j);
if j>k then
Begin
inc(dem); i:=i+k+1;
while (d[i]>=k) do
```

**Begin**
inc(i);
inc(dem);
**End;**
End
Else i:=i+j;
**End;**
Until i>n-k
**End;**
Thoát khỏi vòng lặp này biến dem sẽ trả về số các đặc trưng tìm thấy của ảnh! Như vậy bài toán đã được giải quyết 1 cách trọn vẹn.

**4. Bài toán mở rộng.**

1. Cho file văn bản chứa các kí tự '0' và '1' file gồm rất nhiều dòng, mỗi dòng có 1000 kí tự. Hãy xác định toạ độ hình vuông lớn nhất trong văn bản chỉ chứa hoặc là kí tự '0' hoặc là kí tự '1'. Yêu cầu đưa ra toạ độ đầu và độ dài của cạnh hình vuông tìm được.
2. Quay lại bài toán 1 nếu các chữ cái T,E,F,I của chúng ta có thể quay $90^0$,$180^0$, $270^0$ thì chúng ta sẽ đếm các chữ cái này như thế nào?

Xin hẹn các bạn ở một bài toán khác. Mọi góp ý, trao đổi xin liên hệ qua email *duyhaman@yahoo.com.*. Xin chân thành cảm ơn!

## Tìm kiếm ưu tiên chiều rộng - Một số bài tập áp dụng

**Ngô Minh Đức**

**Trình bày sơ lược**

Tìm kiếm ưu tiên chiều rộng , hay còn gọi là "loang", là một trong những thuật toán duyệt đồ thị đơn giản nhất. Ý tưởng của nó được sử dụng trong nhiều thuật toán, chẳng hạn thuật toán Prim tìm cây khung nhỏ nhất, thuật toán Dijkstra tìm đường đi ngắn nhất, v.v...

Loang chủ yếu được sử dụng để tìm đường đi ngắn nhất theo số cạnh giữa hai đỉnh của một đồ thị. Ta hình dung từ một đỉnh nguồn s, ban đầu thuật toán loang khám phá các đỉnh đến được từ s, đó là lớp thứ nhất, sau đó lại khám phá các đỉnh chưa thăm và đến được từ lớp thứ nhất, đó là lớp thứ hai, v.v... Nghĩa là các đỉnh đến từ có khoảng cách k từ s luôn được khám phá trước các đỉnh có khoảng cách k+1 từ s.

Sau đây là mã giả của thuật toán loang: (thực ra là mã Pascal)

```
For i:=1 to n do {n là số đỉnh}
Trace[i]:=0;
Trace[s]:=-1; {s là đỉnh nguồn}
d[s]:=0; {d[i] là khoảng cách từ nguồn đến đỉnh i}
i:=1; j:=1; q[i]:=s; {q là hàng đợi}
While i<=j do
Begin
For k Adj[q[i]] do
If Trace[k]=0 then
Begin
Trace[k]:=q[i];
D[k]:=D[q[i]]+1;
Inc(j);
```

```
q[j]:=k;
End;
Inc(i);
End;
```

Về mặt trực quan, ta thấy thuật toán loang luôn tìm được đường đi ngắn nhất theo số cạnh giữa hai đỉnh của một đồ thị. Nhưng thực ra, cũng cần phải chứng minh điều này. Dưới đây là một số bổ đề, hướng chứng minh:

Ký hiệu d(s,v) là số cạnh ít nhất trên một đường đi nào đó giữa s và v, giá trị này còn được gọi là khoảng cách giữa s và v. Nếu không có đường đi thì d(s,v)=¥. Một đường đi từ s đến v có số cạnh là d(s,v) được gọi là đường đi ngắn nhất (theo số cạnh) giữa s và v.

Bổ đề 1: Với mọi cạnh (u,v) thuộc G, ta có d(s,v)= d(s,u)+1

Bổ đề 2: Sau khi kết thúc thuật toán loang, với mọi đỉnh v giá trị d[v] trả về thỏa d[v]³ d(s,v)

Chứng minh: có thể quy nạp theo số phép toán đẩy vào hàng đợi

Bổ đề 3: Giả sử trong qúa trình thực hiện thuật toán loang, hàng đợi Q chứa các đỉnh v1, v2,... , vr, với v1 ở đầu hàng đợi và vr ở cuối. Thế thì d[vr] ≤ d[v1] + 1 và d[vi ] ≤ d[vi+1] với mọi i = 1, 2,..., r - 1.

Chứng minh: có thể quy nạp theo số phép toán hàng đợi

Hệ qủa 1: Giả sử đỉnh vi và vj được đưa vào hàng đợi trong qúa trình thực hiện thuật toán loang, và vi được đưa vào trước vj, thế thì d[vi] ≤ d[vj] ngay khi vj được đưa vào hàng đợi

Chứng minh: trực tiếp từ bổ đề 3 và tính chất mỗi đỉnh chỉ nhận giá trị d nhiều nhất một lần

Đinh lý: Sau khi kết thúc thuật toán loang, d[v]= d(s,v) với mọi đỉnh v thuộc G

Chứng minh:

Phản chứng. Gọi v là đỉnh có giá trị d(s,v) nhỏ nhất mà bị gán sai nhãn d[v], từ đó suy ra điều mâu thuẫn

Câu hỏi:

Cho G=(V,E) là một đồ thị vô hướng liên thông. Hãy viết chương trình tìm một đường đi trong G qua mỗi cạnh đúng một lần theo mỗi hướng.

Một số bài tập áp dụng

Biến đổi từ

Cho một từ điển (bao gồm một số từ). Từ một từ ta có thể thay đổi một chữ cái để thu được một từ khác cũng trong từ điển. Như vậy từ này có thể biến thành từ kia bằng cách thực hiện một số phép biến đổi. Ví dụ từ “spice” có thể biến đổi thành từ “stock” như sau: spice, slice, slick, stick, stock.

Yêu cầu: cho một số cặp từ, gồm từ nguồn và từ đích. Với mỗi cặp từ, hãy xác định số phép biến đổi ít nhất để từ từ nguồn thu được từ đích.

Giới hạn: từ điển chứa không qúa 200 từ

Hướng dẫn:đây là bài toán loang đơn giản. Mỗi từ là một đỉnh của đồ thị, nhưng không cần xây dựng đồ thị một cạnh tường minh mà dùng một hàm để kiểm tra trực tiếp (i,j) có phải là cạnh của đồ thị hay không

Bộ sưu tập (Đề thi quốc gia bảng B 2005)

Một bộ sưu tập tiền xu cổ là có giá trị phải gồm không ít hơn Zo đồng vàng, S0 đồng bạc, M0 đồng đồng. Cho biết các qui tắc đổi gói tiền (Z1, S1, M1) sang (Z2, S2, M2). Mỗi hội viên (hội sưu tập tiền cổ) không được giữa qúa 4 đồng tiền mỗi loại. Các đồng tiền nhận được sau mỗi lần đổi được gộp lại với các đồng tiền mà hội viên đang có để thành một bộ sưu tập mới và có thể được sử dụng để đổi trong những lần sau nếu cần.

Yêu cầu: Cho số lượng Z,S,M các đồng tiền mà Alibaba có ban đầu và các quy tắc đổi tiền. Hãy chỉ ra một phương án đổi tiền nào đó để Alibaba có được một bộ sưu tập có giá trị.
Hướng dẫn: đây cũng là một bài toán loang đơn giản. Mỗi đỉnh của đồ thị là một bộ (Z,S,M), do điều kiện $0 \le Z,S,M \le 4$ nên chỉ có 53=125 đỉnh. Từ các quy tắc đổi tiền giúp ta xác định được các cạnh của đồ thị. Chú ý cài đặt cẩn thận để đạt kết qủa tốt.

Đường kính của cây

Đường kính của cây T=(V,E) được cho bởi giá trị
max(d(u,v)), với u,v T
nghĩa là giá trị lớn nhất trong các khoảng cách ngắn nhất trên cây đó
Chỉ ra một thuật toán hiệu qủa để tính đường kính của một cây.
Hướng dẫn: Loang từ một đỉnh bất kỳ s. Giả sử u là đỉnh xa nhất khi đi từ s. Lại tiến hành loang từ u. Giả sử v là đỉnh xa nhất khi đi từ u. Thế thì u->v chính là đường kính của cây

Trạng thái xa nhất

Trò chơi 8-puzzle gồm một khay hình vuông với 8 mảnh vuông được đặt lên 8 ô vuông. Ô vuông còn lại rỗng. Mỗi mảnh có ghi một con số. Một mảnh kề với ô rỗng có thể được đẩy sang ô rỗng này. Một ván chơi bao gồm trạng thái bắt đầu và trạng thái kết thúc. Người chơi phải biến đổi đến trạng thái kết thúc bằng cách di chuyển các mảnh vuông. Bài toán 8-puzzle yêu cầu phải biến đổi với số bước ít nhất

Nhưng trong bài toán này (bài toán trạng thái xa nhất), bạn được cho một trạng thái bắt đầu. Hãy tìm trạng thái xa nhất (theo nghĩa số bước đi) trong tất cả các trạng thái đến được.
Hướng dẫn: Đây cũng là một bài toán loang. Nhưng hơi khó khăn ở chỗ lưu vết, vì có đến 9! =362880 trạng thái => không đủ bộ nhớ. Có thể khắc phục bằng cách dùng kỹ thuật băm. Cách làm như sau: cho tương ứng (1-1) giữa mỗi trạng thái với một số nguyên trong khoảng 1->362880, sau đó dùng một bảng băm với kích thước sao cho đủ bộ nhớ. Tiếp đó ánh xạ mỗi trạng thái vào một khe trong bảng băm (có thể dùng phép đồng dư) để lưu.

Mê cung

Cho một mê cung có kích thước WxH (1<=W<=38, 1<=H<=100). Trong mê cung đầy những hàng rào. Tuy nhiên trên cạnh của mê cung có hai vị trí không có hàng rào, giúp "thoát" ra khỏi mê cung, và từ bất kỳ vị trí nào trong mê cung cũng có đường thoát ra ngoài.
Yêu cầu: Tính số bước cần thiết để thoát ra khỏi mê cung từ vị trí "tệ nhất" trong mê cung (vị trí mà muốn thoát ra phải đi xa nhất)
Ví dụ, với W=5, H=3

Kết qủa: 9
Hướng dẫn: đây cũng là bài toán loang đơn giản. Từ cửa thoát 1 ta loang để thu được mảng khoảng cách d1. Từ cửa thoát 2 ta loang để thu được mảng khoảng cách d2. Với mỗi ô i,j đặt d[i,j]=min(d1[i,j],d2[i,j]). Thế thì giá trị cần tìm chính là giá trị d lớn nhất.

Hình tròn màu

Cho n hình tròn, mỗi hình tròn i được tô bởi một màu $c_i$. Nối từ hình tròn i đến hình tròn j là một cung có màu là $e_{i,j}$. Ban đầu người ta đặt hai quân cờ tại 2 vị trí (1,2) và di chuyển theo quy tắc sau:
Nếu hai quân cờ đang đứng tại hai ô(x,y) thì
Có thể di chuyển quân cờ đang đứng tại ô y đến ô y' nếu $e_{y,y'}$</SUB>=Cx

Có thể di chuyển quân cờ đang đứng tại ô x đến ô x' nếu ex,x'</SUB>=Cy

Mỗi lần di chuyển như vậy tốn 1 đơn vị thời gian.

Bài toán đặt ra là: tìm cách di chuyển sao cho trong thời gian nhanh nhất có một quân cờ đến ô n.

Dữ liệu vào: số n, c1, c2,... , cn, ei,j

Dữ liệu ra: thời gian cần thiết để di chuyển và cách di chuyển. Trong trường hợp không di chuyển được thì đưa ra số -1.

Hướng dẫn: Xây dựng đồ thị G=(V,E) với mỗi đỉnh là một cặp (x,y) thể hiện vị trí đang đứng của 2 quân cờ. Tập cạnh có dạng ((x,y),(x,y')) nếu ey,y'</SUB>=Cx hoặc ((x,y),(x',y)) nếu ex,x'</SUB>=Cy. Với mô hình đồ thị trên thì bài toán của chúng ta sẽ là: tìm đường đi ngắn nhất (theo số cạnh) từ đỉnh (1,2) đến đỉnh có dạng (p,n) hoặc (n,q). Đến đây ta có thể dùng thuật toán loang để giải quyết bài toán.

**Một số bài tập khác**

1.Mã trên bàn cờ 5x5

Có các quân mã trắng và đen trên một bàn cờ 5x5. Có 12 quân mỗi loại và chỉ có một ô rỗng. Tại mỗi thời điểm, một quân mã có thể di chuyển đến một ô rỗng (cách đi của quân mã như luật cờ vua thông thường).

Cho một trạng thái ban đầu của bàn cờ, hãy xác định số bước đi ít nhất để đạt được trạng thái sau:

**2.Lâu đài**

Cho sơ đồ một lâu đài gồm các: #: bức tường, -, | không có tường

Yêu cầu: đếm số phòng và kích thước mỗi phòng, sau đó phá bỏ đi một bức tường sao cho căn phòng mới thu được là rộng nhất

Trong ví dụ trên vị trí mũi tên là bức tường cần phá bỏ

**Gặp gỡ (Thi quốc gia 98-99)**

Trên một lưới ô vuông M*N (M,N < 100), người ta đặt robot A ở góc trái trên, robot B ở góc phải dưới. Mỗi ô của lưới có thể đặt một vật cản hoặc không (ô trái trên và phải dưới không có vật cản). Hai robot bắt đầu di chuyển đồng thời với tốc độ như nhau và không robot nào được dừng lại trong khi robot kia di chuyển trừ khi nó không thể di chuyển (trừ khi nó không thể đi được nữa). Tại mỗi bước, robot chỉ có thể di chuyển theo 4 hướng - đi lên, đi xuống, sang trái, sang phải - vào các ô kề cạnh. Hai robot sẽ gặp nhau nếu chúng đứng trong cùng một ô vuông. Bài toán đặt ra là tìm cách di chuyển ít nhất mà 2 robot phải thực hiện để có thể gặp nhau.

Dữ liệu vào trong file Meet.inp :

- dòng đầu ghi 2 số M,N.
- M dòng tiếp theo, mỗi dòng ghi N số 0 hoặc 1 mô tả trạng thái của các ô vuông: 1-có vật cản, 0-không có vật cản.

Các số trên cùng một dòng của file dữ liệu cách nhau ít nhất một dấu trắng.

Kết quả ghi ra file Meet.out :

- nếu 2 robot không thể gặp nhau thì ghi ký tự #.
- Ngược lại, ghi hai dòng, mỗi dòng là một dãy các lý tự viết liền nhau mô tả các bước đi của robot : U-đi lên, D-đi xuống, L- sang trái, R-sang phải Dòng đầu là các bước đi của A, dòng sau là của B.

Ví dụ:

Meet.inp

4 6

0 1 1 0 0 0

0 0 0 0 0 1

0 0 1 0 0 1

0 1 0 1 0 0

Meet.out

DRRR

LULU

**Kết luận**

Đây chỉ là một số bài tập cơ bản áp dụng thuật toán loang. Còn nhiều bài tập hay khác áp dụng kỹ thuật đơn giản này. Mong được có dịp trao đổi với các bạn nhiều hơn.

# Kỹ năng tối ưu hoá thuật toán

**Nguyễn Duy Hàm**

Tối ưu hoá là một đòi hỏi thường xuyên không chỉ trong quá trình giải quyết các bài toán tin học, mà ngay trong giải quyết các công việc thường ngày của chúng ta. Tối ưu hoá thuật toán là một công việc yêu cầu tư duy thuật toán rất cao, cùng với khả năng sử dụng thuần thục các cấu trúc dữ liệu. Lần này tôi xin được chia sẻ với các bạn về tối ưu hoá thuật toán trong giải quyết một bài toán tưởng chừng rất đơn giản, nhưng việc tìm ra thuật toán tối ưu cho nó lại không dễ chút nào. Tối ưu hoá thường được tiến hành theo 2 góc độ đó là tối ưu theo không gian có nghĩa là tối ưu không gian lưu trữ (bộ nhớ), và tối ưu theo thời gian có nghĩa là giảm độ phức tạp thuật toán, giảm các bước xử lý trong chương trình…Tuy nhiên không phải lúc nào ta cũng có thể đạt được đồng thời cả 2 điều đó, trong nhiều trường hợp tối ưu về thời gian sẽ làm tăng không gian lưu trữ, và ngược lại.

***Phát biểu bài toán***

Cho dãy số $a_0, a_1, \ldots, a_{n-1}$ ($0 < n < 10.000$), hãy tìm dãy con có tổng lớn nhất của dãy số đó. Yêu cầu đưa ra tổng lớn nhất và chỉ số đầu và cuối của dãy con tìm được. Hạn chế của bài toán là thời gian chạy 0.005s trên máy P4 2.4Ghz.

Chúng ta thấy rằng bài toán khá đơn giản, song nếu không được giải quyết tốt sẽ không đáp ứng được yêu cầu về thời gian. Với các bài toán có hạn chế về thời gian như thế, đòi hỏi ta phải đưa ra được thuật toán tối ưu nhất.

*Cách giải thứ nhất:*

Cách làm sơ khai nhất là xét tất cả các đoạn con có thể theo đoạn mã sau:

```
int Findmax(int a[],int &dau,int &cuoi){
max=a[0];dau=cuoi=0;
for(i=0;i<N;I++){< p>
for(j=i;j<N;J++){< p>
s=0;
for(k=i;k<=j;k++)
s+=a[k];
if (max<S){< p>
max=s;
dau=i;
cuoi=j;
}
}
}
return max;
}
```

Ta dễ thấy rằng cách làm trên có độ phức tạp $O(n^3)$ với 3 vòng for lồng nhau. Đoạn mã trên có quá nhiều cái để ta có thể tối ưu, ví dụ như khi tính tổng từ i->j đã không tận dụng được tổng từ i->(j-1) đã được tính trước đó! Do vậy đây là 1 đoạn mã khó có thể chấp nhận được về mặt kĩ thuật lập trình lẫn kĩ năng viết chương trình.

*Cách làm thứ 2:*

Với 1 cải tiến nhỏ ta sẽ xét hết tất cả các dãy con có thể của dãy số bắt đầu từ vị trí thứ i (i=0,1,2…n-1), với việc tận dụng lại các giá trị đã được tính trước đó. Chúng ta cùng theo dõi đoạn mã sau:

```
int Findmax(int a[],int max,int &dau,int &cuoi){
max=a[0];dau=cuoi=0;
for(i=0;i<N-1;I++){< p>
s=0;
for(j=i;j<N;J++){< p>
s+=j;
if (s>max){
max=s;
cuoi=j;
dau=i;
}
}
return max;
}
```

Thuật toán trên sử dụng 2 vòng lặp lồng nhau, vòng đầu tiên lặp n lần, vòng thứ 2 lặp tối đa n lần, nên dễ thấy độ phức tạp của thuật toán này là $O(n^2)$. Tôi tin rằng nhiều người sẽ đồng ý sử dụng cách làm trên, nhưng chắc chắn 1 điều là nó không đáp ứng được đòi hỏi về thời gian, không tin các bạn cứ thử xem!

*Cách giải thứ 3:*

Ta cùng tìm hiểu một cải tiến khác chắc chắn sẽ tốt hơn. Ta sử dụng thêm 2 mảng k,c mỗi mảng gồm n phần tử. Trong đó k[i] sẽ lưu giá trị lớn nhất của tổng dãy con mà a[i] là cuối dãy, c[i] sẽ lưu chỉ số đầu của dãy đó. Như vậy k[i]=max(a[i],a[i]+k[i-1]). Hãy theo dõi đoạn mã khá lí thú sau, nó được xây dựng trên tư tưởng quy hoạch động trên:

```
int Findmax(int a[],int &dau,int &cuoi){
k[0]=max=a[0];c[0]=dau=cuoi=0;
for(i=1;i<N;I++){< p>
s=k[i-1]+a[i];
k[i]=a[i];
c[i]=i;
if (s>k[i]){
k[i]=s;
c[i]=c[i-1];//đầu dãy
}
//tìm tổng lớn nhất
if (k[i]>max){
max=k[i];
cuoi=i;//cuối dãy
}
}
dau=c[cuoi];
return max;
}
```

Với thuật toán trên độ phức tạp là O(n) với chỉ 1 vòng lặp n lần duy nhất. Như vậy, có thể nói ta đã tối ưu xong về mặt thời gian từ độ phức tạp $O(n^3)$ xuống còn O(n). Về không gian bộ nhớ, ở cách làm trên ta đã sử dụng thêm 2 mảng k và c mỗi mảng n phần tử, nếu không gian đầu vào không phải hạn chế 10.000 mà là 20.000 thậm chí 30.000 thì sao? Chắc chắn sẽ không đủ không gian bộ nhớ để sử dụng 2 mảng k và c như trên. Vậy giải quyết thế nào? Ta sẽ dễ dàng bỏ mảng k đi khi sử dụng tính toán hoàn toàn trên mảng a, vậy có thể bỏ nốt mảng c đi không? nếu bỏ đi thì ta sẽ xác định giá trị đầu của mỗi dãy như thế nào?

*Cách thứ 4:*

Tại mỗi vị trí đang xét ta sẽ so sánh để tìm ra dãy có tổng lớn nhất ngay như trên, và như vậy ta sẽ sử dụng 1 biến để lưu giá trị đầu của dãy tìm được! Ý tưởng đó thể hiện qua đoạn mã sau:

```
int Findmax(int a[],int &dau,int &cuoi){
dau=cuoi=luu=0;max=a[0];
for(i=1;i<N;I++){< p>
a[i]+=a[i-1];
if (a[i]<A[I]-A[I-1]){< p>
a[i]-=a[i-1];
dau=i;
}
//tìm tổng lớn nhất
if (a[i]>max){
max=a[i];
cuoi=i;//cuối dãy
luu=dau;//đầu dãy
}
}
dau=luu;
return max;
}
```

Cách làm trên quả thật rất hiệu quả, tuy nhiên nó đã làm biến đổi dãy số ban đầu(mảng a). Làm sao có thể giữ nguyên dãy số, không dùng thêm mảng phụ, vẫn đáp ứng được yêu cầu đề bài. Với một cải tiến nhỏ ta sẽ thấy ở đoạn mã sau thật tuyệt vời:

```
int Findmax(int a[],int &dau,int &cuoi){
dau=luu=cuoi=0;max=s=a[0];
for(i=1;i<N;I++){< p>
s+=a[i];
if (s<A[I]){< p>
s=a[i];
dau=i;
}
//tìm tổng lớn nhất
if (s>max){
max=s;
cuoi=i;//cuối dãy
luu=dau;//đầu dãy
}
```

```
}
dau=luu;
return max;
}
```
Với cải tiến cuối cùng này mọi yêu cầu của bài toán đã được giải quyết triệt để. Chúc các bạn thành công. Hẹn gặp lại ở một vấn đề khác, mọi trao đổi, góp ý xin liên hệ qua Email *duyhaman@yahoo.com*.

## Ứng dụng phương pháp quy nạp toán học

**Nguyễn Duy Khương**

Trong toán học, quy nạp là một phương pháp đơn giản nhưng hiệu quả để chứng minh các bài toán. Ở bài viết này tôi xin đưa ra một ứng dụng nhỏ của nó trong việc giải các bài toán tin học:

**1. Thuật toán quy nạp quy nạp(nhắc lại):**

Giả sử có bài toán F cần chứng minh đúng với mọi $n \in N$. Ta chứng minh bài toán đúng bằng cách quy nạp, cần tiến hành các bước sau:

- n = 1: mệnh đề cần chứng minh đúng.
- Giả sử n = k: mệnh đề cần chứng minh đúng.
- n = k + 1: ta cần chứng nó cũng đúng. Vậy theo nguyên lý quy nạp bài toán đúng với mọi N.

Trong tin học, thuật toán nay cũng được áp dụng. Tuy thuật toán đơn giản nhưng nó lại được áp dụng một cách rất linh động và khéo léo trong các bài toán tin.

**2. Phát biểu bài toán tổng quát giải bằng quy nạp:**

Thông thường bài toán giải bằng quy nạp không phải là một bài toán tối ưu hoá. Nó chỉ đơn giản là bài toán cần chỉ ra cách biến đổi theo quy luật cho trước để thu được kết quả mong đợi. Và bài toán đó thường được phát biểu như sau: Cho N đối tượng và một số thao tác biến đổi. Yêu cầu sử dụng các thao tác biến đổi để thu được kết mong đợi.

Cách làm thông thường:

- Nếu n = 0; 1: ta luôn có cách biến đổi đúng.
- Nếu có n > 1 mà ta luôn chỉ ra một cách biến đổi sao cho giản bớt được số đối tượng mà điều kiện bài toán vẫn không thay đổi.
- Như vậy vì số đối tượng trong một bài toán tin học luôn là hữu hạn nên sau một số hữu hạn bước thì số lượng đối tương bằng 1 hoặc 0. Suy ra bài toán được giải quyết một cách hoàn toàn.

**3. Các ví dụ cụ thể:**

P1. Cho N vecto v1, v2, …, vn độ dài nhỏ hơn L cho trước. Cần đặt trước các vecto một dấu cộng hoặc trừ sao cho vecto tổng thu được có độ dài nhỏ hơn căn 2 nhân L. Input: Segment.In

- Dòng đầu ghi số N ( $1 < N \leq 10000$) và L ($0 < L \leq 15000.00$)
- N dòng tiếp theo mỗi dòng ghi một cặp số xi, yi là toạ độ của vecto thứ i. ($|xi|, |yi| \leq 10000$).

Ouput: Segment.out

- Dòng thứ nhất ghi YES nó có thể đặt các dấu cộng trừ thoả mãn yêu cầu bài toán, NO trong trường hợp ngược lại.
- Nếu là YES dòng thứ hai ghi N kí tự cộng hoặc trừ cần đặt trước các vecto sao cho tổng

vecto nhỏ hơn sprt(2)*L.
**Ví dụ:**

Thuật giải:
- n = 1 bài toán đã đúng.
- n = 2 có trường hợp xảy ra với hai vecto:
+ góc giữa hai vecto nhỏ hơn 90 độ, lấy vecto 1 trừ cho vecto 2 lúc đó ta thu được 1 vecto co độ dài nhỏ hơn sqrt(2) * L.
+ góc giữa hai vecto lớn hơn hoặc bài một 90 độ, lấy vecto 1 cộng cho vecto 2 lúc đó ta thu được 1 vecto có độ dài nhỏ hơn sqrt(2) * L.
- n ≥ 3 suy ra luôn có 2 vecto mà góc giữa hai phương của vecto nhỏ hơn hoặc 60 bằng độ có hai trường xảy ra:
+ góc giữa hai vecto đó nhỏ hơn hoặc bằng 60 độ, trừ vecto 1 cho vecto 2 thay hai vecto bằng vecto mới nhận được.
+ góc giữa hai vecto đó lớn hơn hoặc bằng 120 độ, cộng vecto 1 cho vecto 2 thay hai vecto bằng vecto mới nhận được. Lưu ý sau này nếu trừ vecto mới nhận được phải trừ (đổi dấu) toàn vecto bộ hình thành lên nó, vecto mới nhận được có độ dài nhỏ hơn L, số đoạn thẳng giảm 1.
Như vậy đây là bài toán luôn có lời giải, độ phức tạp O(N^2) nếu xử lý khéo có thể giản xuống O(n*log(n)). Chương trình mô tả thuật giải:

```
Procedure Xuly (n: integer //* số đối tượng *//);
Begin
If n <= 1 then exit else
If n = 2 then
Begin
If goc (v1, v2) <= 90 độ Then đổi dấu (v2);
//* đổi dấu tức là đổi dấu toàn bộ vecto thành phần *//
Thay hai vecto bằng 1 vecto (v1 + v2);
//* vecto mới gồm các thành phần của v1 và v2 *//
//* lúc này v2 nếu cần thiết đã đổi dấu rồi *//
Exit;
End else
Begin
lấy 3 vecto bất kỳ;
chọn ra được 2 vecto v1, v2 sao cho
góc nhỏ giữa phương 2 vec to nhỏ 60;
if goc (v1, v2) <= 60 then đổi dấu (v2);
Thay hai vecto này bằng (v1 + v2);
End;
End;
```

**P2: Utopia (IOI 2002)**
Đất nước Utopia tươi đẹp bị chiến tranh tàn phá. Khi chiến tranh tạm ngừng, đất nước bị chia cắt thành bốn miền bởi một đường kinh tuyến (đường theo hướng Bắc-Nam) và một đường vĩ tuyến (đường theo hướng Đông-Tây). Giao điểm của hai đường này xem như điểm (0, 0). Tất cả các miền này đều muốn mang tên Utopia, theo thời gian các miền này được gọi là Utopia

1 (phía Đông Bắc), Utopia 2 (phía Tây Bắc), Utopia 3 (phía Tây Nam), Utopia 4 (phía Đông Nam). Một điểm trong bất kỳ miền nào cũng được xác định bởi khoảng cách theo hướng Đông và khoảng cách theo hướng Bắc của nó so với điểm (0, 0). Bộ hai khoảng cách này được gọi là toạ độ của điểm. Các thành phần của toạ độ có thể là số âm; do đó một điểm miền Utopia 2 có toạ độ (âm, dương), một điểm miền Utopia 3 có toạ độ (âm, âm), một điểm miền Utopia 4 có toạ độ (dương, âm), một điểm miền Utopia 1 có toạ độ (dương, dương) (Giống như các góc phần tư trong hệ toạ độ đề các).

Một vấn đề là các công dân không được phép đi qua biên giới giữa các miền. May thay, một số thí sinh tham dự IOI của Utopia sáng chế ra một loại máy an toàn để vận chuyển từ xa. Máy này cần các số mã, mỗi số mã chỉ có thể được dùng một lần. Bây giờ, nhóm thí sinh tham dự và bạn phải hướng dẫn máy vận chuyển từ xa xuất phát từ vị trí ban đầu (0, 0) đến các miền của Utopia theo một trình tự cho trước. Khi bạn nhận được một dãy N số hiệu miền mà theo trình tự đó máy vận chuyển từ xa phải hạ cánh, với mỗi miền, bạn không cần phải quan tâm đến việc hạ cánh tại điểm nào của miền đó miễn là điểm đó thuộc miền yêu cầu. Người ta có thể yêu cầu máy hạ cánh liên tiếp hai lần hay nhiều lần hơn ở cùng một miền. Sau khi rời khỏi điểm ban đầu (0, 0), bạn không được hạ cánh trên các đường biên giới.

Bạn sẽ nhận được input là một dãy 2N số mã và phải viết chúng thành một dãy gồm N cặp mã, sau khi đặt một dấu + hoặc một dấu - thích hợp trước mỗi số. Nếu hiện tại bạn ở điểm (x,y) và sử dụng cặp mã số (+u,-v), bạn sẽ được chuyển tới điểm (x+u,y-v). Bạn có 2N số và bạn có thể sử dụng các số này theo thứ tự bạn muốn, mỗi số kèm theo một dấu + hoặc một dấu -.

Giả sử bạn có các số mã 7, 5, 6, 1, 3, 2, 4, 8 và phải hướng dẫn máy vận chuyển từ xa lần lượt đến các miền thuộc dãy miền 4, 1, 2, 1. Dãy các cặp mã (+7, -1), (-5, +2), (-4, +3), (+8, +6) đáp ứng yêu cầu này vì nó đưa bạn từ (0,0) lần lượt đến các vị trí (7, -1), (2, 1), (-2, 4) và (6, 10). Những điểm này nằm tương ứng ở Utopia 4, Utopia 1, Utopia 2 và Utopia 1.

**Nhiệm vụ:**

Cho 2N số mã khác nhau và một dãy N số hiệu miền là dãy các miền mà máy vận chuyển từ xa phải lần lượt hạ cánh. Hãy xây dựng một dãy các cặp mã từ các số đã cho để hướng dẫn máy vận chuyển từ xa lần lượt đi đến các miền thuộc dãy đã cho.

**Input**

Chương trình của bạn phải đọc từ input chuẩn. Dòng đầu gồm một số nguyên dương N ($1 \leq N \leq 10000$). Dòng thứ hai gồm 2N số mã nguyên khác nhau, tất cả đều là các số nguyên dương không lớn hơn 100000. Dòng cuối cùng gồm một dãy N số hiệu miền, mỗi một trong các số đó là 1, 2, 3 hoặc 4. Hai số liên tiếp trên một dòng cách nhau đúng một dấu trống.

**Output**

Chương trình của bạn phải viết ra output chuẩn. Ouput gồm N dòng, mỗi dòng chứa một cặp số mã mà trước mỗi số có dấu + hoặc dấu -. Đó là các cặp mã sẽ hướng dẫn cho máy vận chuyển đến dãy miền đã cho. Chú ý rằng không được có dấu trống sau dấu + hoặc dấu - nhưng phải có một dấu trống sau số mã thứ nhất. Nếu có nhiều lời giải, chương trình của bạn có thể đưa ra một lời giải bất kỳ trong số đó. Nếu không có lời giải, chương trình của bạn phải đưa ra một số 0 duy nhất.

**Ví dụ:**

Thuật giải:

Đây cũng là bài toàn luôn giải được để giải bài toán trên ta cần giải bài toán nhỏ sau: Cho một gồm N số nguyên dương khác nhau a1 < a2 < … < an. Cần sắp xếp lại và thêm các dấu

vào các số ai sao cho tổng các số từ 1 đến vị trí i là dương hoặc âm biết trước:
Ví dụ:
Dãy 3 số: 1 2 3
Dấu cần xây dựng: +-+
Một cách thêm dấu và sắp xếp: +1 -2 + 3
Bổ đề:
Nếu dãy có dãy {ai} dương tăng và một chuỗi dấu cần biến đổi thì ta luôn có cách thêm dấu và thay đổi vị trí sao cho tổng cách số từ 1 đến i mang đấu của chuỗi dấu biết trước.
Chứng minh:
- số an mang dấu cuối cùng của chuỗi trên, các số còn lại đan dấu. (điều kiện về dấu để luôn xây được dãy) Ví dụ: {ai} = {1, 5, 10}; S = '++-'; suy ra dãy {ai} được đánh dấu như sau: -1, +5, -10;
- nhận xét rằng: tổng của n số trên cuối cùng sẽ mang dấu của S[n]
- với n = 1 ta nhận được dãy cần tìm.
- Giả sử n =k ta xây dựng được chuỗi như trên.
- n = k + 1 chia hai trường hợp:
+ s[n-1] cùng dấu s[n]: đặt a[1] vào vị trí cuối cùng và lấy phần tử a[1] và s[n] ra khỏi dãy, nhưng còn là (n - 1) số và (n-1) dấu và dấu cuối cùng với số cuối, dãy đan dấu (vẫn thoả mãn điều kiện về dấu)
+ s[n - 1] và s[n] khác dấu: đặt a[n] vào vị trí cuối cùng và lấy phần tử a[n] và s[n] ra khỏi dãy, nhưng còn là (n - 1) số và (n-1) dấu và dấu cuối cùng với số cuối, dãy đan dấu. Theo giả thiết quy nạp thì dãy còn lại luôn xây dựng được.
Chương trình:

```
Procedure Xuly (Var a, dau, q: array [1…10000] of integer; n, cd, ct : integer);
//*ta đang xử lý dãy các số từ a[cd] đến a[ct] có dấu từ dấu[1] đến dấu[n] *//
Begin
If n <= 1 then exit;
If dau[n - 1] = dau[n] then
Begin
q[n] := a[cd];
Xuly (a, dau, q, n - 1, cd + 1, ct);
End else
Begin
q[n] := a[ct];
Xuly (a, dau, q, n - 1, cd, ct - 1);
End;
End;
//* mảng q thu được là kết quả cần tìm*//
```

Ở bài toán trên các dấu tọa độ x, y là đã xác định: Chia dãy 2N phần tử khác nhau thành hai gồm N phần tử làm hoành độ và tung độ, sau đó sắp tăng. Tiến hành thuật giải trên ta thu được hai dãy toạ độ thoả mãn điều kiện về dấu, đó cũng chính là dãy toạ độ cần tìm.

**P3: (Bài tập tự giải) 2N point (POI)**

Cho 2N diểm trên mặt phẳng gồm N diểm màu xanh và N điểm màu trắng sao cho không có ba điểm nào thẳng hàng. Hãy tìm N đoạn thẳng mà mỗi đoạn thẳng nối giữa một điểm màu xanh và một điểm màu trắng và các doạn nay không cắt nhau.

Input: 2Npoint.Int
- Dòng đầu ghi số N. ($1 \le N \le 5000$)
- N dòng tiếp theo mỗi dòng ghi toạ độ xi, yi của điểm màu xanh ($|xi|, |yi| \le 10000$).
- N dòng tiếp theo mỗi dòng ghi toạ độ xi, yi của điểm màu trắng ($|xi|, |yi| \le 10000$).
Output: 2Npoint.Out
- Gồm N dòng mỗi dòng ghi hai số a, b để chỉ điểm màu xanh thứ a, nối với điểm màu trắng thứ b.

**4. Lời kết:**

. Mọi tư duy toán học đều có thể vận dụng một cách linh động trong tin học. Tôi hy vọng sau bài viết này, ngoài việc học thêm được một hướng giải mới, các bạn sẽ tìm tòi thêm các ứng dụng toán học khác vào tin học.

## Bàn về một bài toán hay

**Nguyễn Hiển**

Các bạn thân mến! Việc nghiên cứu thuật toán và rút ra kinh nghiệm từ các bài toán hay luôn là một công việc thích thú với các bạn đam mê lập trình. Trong bài viết này, tôi muốn giới thiệu với các bạn các cách giải quyết khác nhau cho một bài toán tưởng như đơn giản.

**A - BÀI TOĂN**

Hôm nay, bạn có nhiệm vụ tân trang lại ô tô của mình. Tất nhiên, có quá nhiều việc phải làm, vì vậy, bạn muốn thuê các xưởng để sửa chữa làm các công việc đó. Không may, các xưởng đó lại rất chuyên môn hoá, cho nên bạn phải thuê các xưởng khác nhau cho các công việc khác nhau. Hơn nữa, họ có xu hướng đòi nhiều tiền hơn với các xe ở tình trạng tốt hơn. Chẳng hạn, một người thợ sơn có thể đòi tiền nhiều hơn khi sửa 1 ô tô mà toàn bộ nội thất đã làm bằng da, anh ta sẽ không đòi nhiều tiền cho xe chưa bọc da. Do đó, tiền trả thêm phụ thuộc vào công việc nào được làm và các công việc làm trước đó. Tất nhiên, bạn luôn muốn tiết kiệm tiền cho mình bằng một thứ tự tốt nhất cho các công việc.
Các công việc được đánh số từ 1 tới n. Cho giá gốc p của mỗi công việc và tiền trả thêm s (tính theo đơn vị đồng) đối với một cặp công việc (i,j) là $s_{i,j}$ (với $i \ne j$), nghĩa là bạn phải trả thêm $s_{i,j}$ đồng cho công việc i nếu và chỉ nếu công việc j được làm xong trước đó. Bạn cần tính tổng giá tiền nhỏ nhất để hoàn thành các công việc này. Dữ liệu vào: file INP.DAT Dòng đầu tiên của file vào chứa số nguyên n ($1 \le n \le 14$). Tiếp theo có n dòng, mỗi dòng chứa đúng n số nguyên: số nguyên thứ i trên dòng này là giá gốc của công việc thứ i và số nguyên thứ j trên dòng này ($i \ne j$) là số tiền $s_{i,j}$ phải trả thêm cho công việc i nếu công việc j được làm trước đó. Các giá tiền là các số nguyên không âm, không vượt quá 100000.
**Dữ liệu ra:** file OUT.DAT
Ghi ra file 1 số nguyên duy nhất là tổng số tiền nhỏ nhất để hoàn thành các công việc. Ví dụ:

**B - CĂCH GIẢI QUYẾT**

Đây là bài toán có nhiều cách giải, với mỗi cách tiếp cận và suy nghĩ về bài toán, chúng ta có các cách giải quyết khác nhau:

**I. CĂCH GIẢI QUYẾT THỨ NHẤT:** *Duyệt*

Dễ thấy, yêu cầu chính của bài toán là sắp xếp n công việc cho trước theo 1 thứ tự xác định để có cực tiểu chi phí. Vì thế, cách giải quyết đơn giản nhất là duyệt mọi hoán vị của các công việc và tính ra chi phí nhỏ nhất. Rất dễ tính toán ra độ phức tạp của thuật toán là: n!.

Với cách giải quyết này, theo thử nghiệm của tôi có thể qua được khoảng 8/14 test (một con số không nhỏ dù không tìm được thuật toán tối ưu!).

**II. CÁCH GIẢI QUYẾT THỨ 2:** *Dùng lý thuyết đồ thị*

Nếu xem xét mỗi công việc như 1 đỉnh của đồ thị, số tiền trả thêm cho công việc i nếu công việc j được làm trước đó là trọng số của cung (có hướng) từ đỉnh j đến đỉnh i. Như vậy, bài toán quy về việc xác định 1 đường đi Hamilton có chi phí cực tiểu. Dễ dàng cài đặt thuật toán tìm đường đi Hamilton trên đồ thị có hướng, nhưng thực tế thuật toán tìm đường đi Hamilton cũng là thuật toán duyệt, độ phức tạp không khác phương pháp 1.

**III. CÁCH GIẢI QUYẾT THỨ 3:** *Quy hoạch động*

Cách giải quyết đúng đắn nhất cho bài toán này chính là thuật toán tôi muốn đưa ra để bàn luận với các bạn: Thuật toán Quy hoạch động trên tập hợp.

Chúng ta có một cách tiếp cận khác về bài toán như sau:

- Xét 1 tập hợp a công việc (có thứ tự) đã được làm (ta đánh thứ tự là 1..a), giả sử chi phí cực tiểu lúc đó là $Q_a$, như vậy khi ta chọn một công việc mới để làm sau các công việc trên thì chi phí mới sẽ là:

Với một cách tổ chức dữ liệu hợp lí, ta có thể thực hiện bài toán qui hoạch động khá dễ dàng.

- Xét mỗi số nguyên h, coi như 1 tập hợp các công việc: Bit thứ i của số nguyên h sẽ là trạng thái của công việc i, cụ thể: Bit thứ i của số nguyên h mang giá trị 1 nếu công việc i đã được chọn làm và ngược lại, mang giá trị 0 nếu công việc i chưa được làm. Như vậy, cần 1 số nguyên n bit để lưu 1 tập hợp và có $2^n$ tập hợp.
- Xây dựng mảng best, với ý nghĩa: best[h] = số tiền cực tiểu để thực hiện các công việc trong tập h.

Như vậy, ta có công thức quy hoạch động để tính giá trị cực tiểu khi kết nạp thêm công việc i: $Best[h] = Min\{best[h_0] + p_i + \sum s_{i,j}\}$. Với:

+ $h_0$: tập các công việc có công việc đã được làm và không có công việc i.
+ h: tập các công việc của h0 và kết nạp thêm công việc h được làm cuối cùng. Dễ thấy, $h_0$ chính là số lấy trực tiếp từ h bằng việc tắt đi bit thứ i.
+ $p_i$: số tiền chi cho công việc i.
+ $s_{i,j}$: số tiền trả thêm cho công việc i nếu công việc j được làm trước đó. Dễ thấy, nếu bit thứ j của h là 1 thì $s_{i,j} \neq 0$, ngược lại $s_{i,j} = 0$.

Chương trình có thể viết đơn giản:

Nếu đã xác định được thuật toán thì việc viết chương trình trên trở nên đơn giản, trong khuôn khổ bài báo, tôi không muốn đưa ra phần cài đặt của chương trình, bạn nào có nhu cầu, xin liên hệ trực tiếp qua email.

**C - BÀN LUẬN THÊM**

- Độ phức tạp của thuật toán trên là: $n*2^n$.
- Với thuật toán trên, chương trình chạy rất nhanh với n = 14, thậm chí là n = 20.
- Sở dĩ đề bài chỉ cho n = 14 vì để làm bài toán trên, yêu cầu bộ nhớ là $2^n$ byte.
- Nếu bạn cấp phát 2, 3 mảng động trong TP, bạn hoàn toàn có thể chạy với n = 18.
- Nếu viết bằng Free Pascal, do bộ nhớ không hạn chế nên chương trình có thể chạy với n = 20, thậm chí lớn hơn nữa.

**Kết luận** Như vậy, với 1 bài toán tưởng như đơn giản, nhưng với cách tiếp cận khác nhau, ta

có các phương án giải quyết khác nhau, do đó thu được các kết quả rất khác nhau. Chỉ có đào sâu suy nghĩ, so sánh, cân đối phương pháp và quá trình thực hiện mới giúp ta tìm ra thuật toán tối ưu cho chương trình của mình. Chúc các bạn luôn có các thuật toán tốt nhất!

## Phương pháp quy hoạch động

**Vũ Văn Thoả**

Bài toán 1. Giả sử đường đi từ điểm thứ 1 đến điển thứ 10 có thể xây dựng qua các điểm trung gian đưthuộcc đánh số từ 2 đến 9. Chi phí làm các đoạn đường đưthuộcc cho trong bảng dưới.

| ? | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|
| 1 | 4 | 11 | 3 | - | - | - | - | - | - |
| 2 | 0 | - | - | 3 | 4 | - | - | - | - |
| 3 | - | 0 | - | 1 | 6 | - | - | - | - |
| 4 | - | - | 0 | 4 | 6 | 4 | - | - | - |
| 5 | 3 | 1 | 4 | 0 | - | - | 9 | 8 | - |
| 6 | 4 | 6 | 6 | - | 0 | - | - | 5 | - |
| 7 | - | - | 4 | - | - | 0 | 7 | 12 | - |
| 8 | - | - | - | 9 | - | 7 | 0 | - | 5 |
| 9 | - | - | - | 8 | 5 | 12 | - | 0 | 3 |

Trong đó dấu ? có nghĩa là giữa các điểm i và j không thể xây dựng đưthuộcc đường.
Hãy tìm hành trình của con đường cần xây dựng để chi ohí nhỏ nhất.
Giải. Ta chia tập các điểm thành các tập con như sau:
S1={1} gồm điểm 1.
S2={2,3,4} gồm các điểm có thể xây dựng đường từ điểm 1 đến chúng.
S3={5,6,7} gồm các điểm có thể xây dựng đường từ điểm của S1 đến chúng.
S4={8,9} gồm các điểm có thể xây dựng đường từ điểm của S2 đến chúng.
S5={10} gồm các điểm có thể xây dựng từ điểm của S4 đến chúng.
Bất kỳ hành trình nào để xây dựng đường từ điểm 1 đến điểm 10 đều chứa 4 đoạn đường mà mỗi đoạn noói 2 điểm của hai tập con tương ứng. Vì vậy, quá trình tìm hành trình tối ưu hoá của bài toán có thể chia ra 4 giai đoạn. Ta đưa vào các ký kiệu sau đây.
N=1,2,3,4 là ký hiệu gia đoạn; f(n,i) là chi phí nhỏ nhất để xây dựng toàn bộ đường. Đặt f(0,10)=0, ta có chương trình truy toán:
F(n,i)= min{c(i,j)+f(n-1,j),jthuộc S(5-n+1)}, ithuộc S(5-n).
Ta đặt d(n,i)=j, trong đó j là điểm ứng với cực tiểu vế trái.
Các kết quả tính cụ thể từng bước như sau.
N=1

| ij | 10 | f(1,i) | d(1,j) |
|---|---|---|---|
| 8 | 5+0 | 5 | 10 |

| | | | |
|---|---|---|---|
| 9 | 3+0 | 3 | 10 |

N=2

| ij | 8 | 9 | f(2,i) | d(2,i) |
|---|---|---|---|---|
| 5 | 9+5 | 8+3 | 11 | 9 |
| 6 | ? | 5+3 | 8 | 9 |
| 7 | 7+5 | 12+3 | 12 | 8 |

N=3

| ij | 5 | 6 | 7 | f(3,i) | d(3,i) |
|---|---|---|---|---|---|
| 2 | 3+11 | 4+8 | ? | 12 | 6 |
| 3 | 1+11 | 6+8 | ? | 12 | 5 |
| 4 | 4+11 | 6+8 | 4+12 | 14 | 6 |

N=4

| ij | 2 | 3 | 4 | f(4,i) | d(4,i) |
|---|---|---|---|---|---|
| 1 | 4+12 | 11+12 | 3+14 | 16 | 2 |

Như vậy chi phí nhỏ nhất là f(4,1)=16 và hành trình là (1,2,6,9,10).
Phương pháp giải bài toán 1 là phương pháp QHĐ. Đó là phương pháp đặc biệt để tìm lời giải tối ưu của các quá trình (hệ thống) nhiều giai đoạn (bước). Tư tưởng chủ đạo của phương pháp QHĐ là chuyển việc quy hoạch của cả quá trình nhiều giai đoạn (tĩnh) sàg qui hoạch từng giai đoạn động dựa trên phương trình truy toán. Trong mỗi giai đoạn ta chỉ tối ưu hoá một bước. Tính đúng đắn của phương pháp QHĐ suy ra từ nguyên tắc sau.
Nguyên tắc tối ưu của Bellman.
Trong mọt dãy tối ưu các lựa chọn thì mọi dãy con của nó cũng tối ưu.
Điều kiện tối ưu không phụ thuộc vào tiền sử của hệ điều khiển mà chỉ phụ thuộc vào trạng thái tức thời và mục tiêu điều khiển.
Nội dung của phương pháp QHĐ là xây dựng phương trình truy toán của quá trình nhiều giai đoạn. Sơ đồ tính toán đượoc thực hiện từ duới lên và lưu trữ tất cả các kết quả cần thiết. Từ đó tìm ra lời giải tôpí ưu của bài toán. Ta dễ dàng lập trình để gải các bài toán nhờ QHĐ trên máy tính điện tử. Các bạn tự lập chương trình cho bài toán 1.
Bài toán 2.( Đề thi tin học quốc tế lần thứ 10-1998) Polygon
Xét đa giác (polygon) N đỉnh và N cạnh đưthuộcc đánh số từ 1 đến N. Tại mỗi đỉnh của đa giác cho một số nguyên, còn mỗi cạnh cho nhãn của phép toán cộng:+ hoặc nhân:*. Trên hình vẽ cho đa giác với N=4:

-7 2 4

+

1 + * 3

?

*

5 2

Có trò chơi 1 người như sau. Nước đi đầu tiên là xoá một cạnh bất kì của đa giác. ở mỗi nước đi tiếp theo ta chọn một cạnh E với cách đỉnh V1, V2 rồi thay cạnh này và các đỉnh của nó bởi đỉnh mới có giá trị bằng kết quả thực hiện phép toán xác định bởi nhãn của E trên các giá trị của V1, V2. Trò chơi kết thúc khi đa giác chỉ còn 1 đỉnh và giá trị của nó là kết quả của trò chơi. Chẳng hạn, đối với đa giác đã cho ta xoá cạnh 3, sau đó lần lượtthuộcc các cạnh 1, 4, 2 ta có kết quả trò chơi là 0.

Hãy viết chương trình để tính giá trị cựuc đại và đưa ra danh sách tất cả các cạnh mà ở nước đi đầu tiên khi xoá chúng sẽ nhận đượthuộcc gá trị cực đại đó.

Dữ liệu vào đượoc cho trong file polygon.in gồm 2 dòng: dòng đầu cho số N đỉnh, daong sau chứa nhãn của các cạnh từ 1 đến N, giữa chúng cách ô viết giá trị của các đỉnh (số đàu tiên ứng với đỉnh giữa cạnh N và 1). Nhãn của các cạnh là các chữ cái: chữ t chỉ phép +, chữ x chỉ phép *. Đối với đa giác đã cho, tệp vào là:

4

t -7 t 4 x 2 x 5

Dữ liệu ra là tệp polygon.out gồm hai dòng: dong đàu ghi giá trị cực đại của trò chơi, dong sau ghi số hiệu tất cả các cạnh mà ở nướoc đi đầu tiên xoá chúng cho phép nhận đượthuộcc giá trị cực đại đó. Số liệu các cạnh ghi theo thứ tự tăng và cách nhau 1 ô. Chẳng hạn với đa giác đã cho tệp là:

33

1 2

Hạn chế 0 <= N<= 50

Với dãy nước đi bất kì giá trị của đỉnh thuộc [-32767;32767]

Giải. Ta giải bài toán này nhờ phương pháp QHĐ. Khi xoá đi một cạnh, đa giác trở thành đường gấp khúc (ĐGK) N đỉnh và N-1 cạnh. Sau mỗi bước đi ĐGK giảm đi 1 đỉnh (và 1 cạnh). Để nhận đượthuộcc kết quả, ta cần tính và điền vào bảng các kết quả của trò chơi đối với các phần của ĐGK gồm từ 1 đến N đỉnh. Do có phép * nên trong mỗi bước ta cần tính cả giá trị cực đại lẫn giá trị cực tiểu của trò chơi.

Bảng đối với ĐGK có 1 đỉnh sẽ gồm giá trị của các đỉnh nên đã biết. Để lập bảng đối với ĐGK có m đỉnh, 2<= m<= N, ta làm như sau:

Ký hiệu rMax(i,m) bag rmin(i,m) là cực đại và cực tiểu của trò chơi đối với ĐGK có m đỉnh (2<= m<= N) bắt đàu từ đỉnh i (1<= i<= N). Giả sử i+j (1<= j<= m-1) là cạnh bất kì của ĐGK đó. Ta có phương trình truy toán:

Nếu nhãn (j)=+ thì rmax(i,m)= rmax(i,j)+rmax(i+j,m-j), rmin(i,m)=rmin(i,j)+ rmin(i,j,m-j).

Nếu nhãn (j)=* thì rmax=max{rmax(i,j)*rmax(i+j,m-j),rmax(i,j)*rmin(i+j,m-j), rmin(i,j)*rmax(i+j,m-j),rmin(i,j)*rmin(I+j,m-j)}, rmin=min{rmax(i,j)*rmax(i+j,m-j), rmax(i,j)*rmin(i+j,m-j), rmin(i,j)*rmax(i+j,m-j),rmin(i,j)*rmin(I+j,m-j)}

Đthuộc tiện tính toán ta đặt rmax(n+i,1)=rmax(i,1), rmin(N+i,1)=rmin(i,1), nhãn (N+i)= nhãn (i) (1<= i<= N)

Ký hiệu Max_Res là kết quả của trò chơi thì Max_Res=max{rmaxx(i,n),1<= i<= N}

**Thuật toán quy hoạch động**

**Quang Tùng**

Thuật toán quy hoạch động là một trong những phương pháp giải bài toán tối ưu nhanh và hiệu quả lại rất dễ làm. Sau đây xin giới thiệu lời giải một bài toán quy hoạch động khá cơ bản. **Đó là bài PalinDrone trong đề thi IOI2000.**

**Nội dung:** Cho một xâu ký tự có độ dài không quá 5000 ký tự. Hãy tìm cách thêm vào chuỗi tại bất kỳ vị trí nào trong chuỗi sao cho chuỗi mới tạo thành là một chuỗi đối xứng (Khi đảo ngược chuỗi này ta vẫn được chuỗi đó).

**Yêu cầu:** Số ký tự thêm là ít nhất và thời gian chạy không quá 2 giây. Lời giải được sử dụng các dẫn hướng {$r-},{$c-},{$s-}.

**Input:** file text có tên palin.inp gồm:

- Dòng 1 số nguyên n: độ dài của chuỗi. (1<=n<=5000)
- Dòng 2 là chuỗi ký tự. Không có dấu cách.

Ví dụ : 5

Abcdba

**Output:** file text có tên là palin.out gồm: Một số nguyên chỉ số ký tự cần thêm.

VD: 1 (Vì cần thêm ký tự d vào chuỗi để có abdcdba).

**Thuật giải.**

Nếu xét kỹ bài toán thì thêm vào một số ký tự để đạt được tính đối xứng thì những ký tự này phải giống những ký tự chưa đối xứng ở trong chuỗi. Như vậy những ký tự mà không cần thêm để đối xứng thì nó đã đối xứng rồi. Vì vậy chúng ta chỉ cần tìm chuỗi con đối xứng dài nhất (CCD) rồi chỉ cần thêm số ký tự để tạo tính đối xứng cho những ký tự ngoài chuỗi con này là xong. Điều này đảm bảo tính đối xứng của chuỗi và số ký tự cần thêm là ít nhất.

Nếu ta gọi F[i,j] là độ dài CCD của chuỗi nằm từ vị trí i đến vị trí j. Vậy độ dài CCD của cả chuỗi là F[1,n]. Đơn giản ta nhận thấy:

Nếu S[i]=S[j] thì F[i,j]:=F[i+1,j-1]+2;(1) (S: là mảng lưu chuỗi)

Nếu S[i]<>S[j] thì F[i,j]:=max(f[i,j-1],f[i+1,j]);(2)

Nhưng nếu sử dụng mảng f[i,j] thì cần 5000*5000*2 byte(quá lớn).

Ta hãy chú ý vào công thức, F[i,j] chỉ phụ thuộc vào f[i+1,j-1] hoặc f[i,j-1] hoặc f[i+1,j-1].

Vậy ta sẽ sử dụng mảng:

D,d1,db:array[1..5000] of integer;

Trong đó tại lần lặp thứ i, d[j] chứa độ dài CCD của chuỗi đó có vị trí là j và i + j. d1[j] chứa độ dài CCD của chuỗi có vị trí là j và i+j-2. db là mảng phụ để gán giữa d và d1;

**Như vậy ta có sơ đồ thuật toán sau:**

*Khởi tạo:* Gán mảng d giá trị 1 và d1 giá trị 0.

*Xử lý:* {độ dài CCD của cả chuỗi sẽ là d[1] tại lần lặp thứ n-1}.

```
For i:=1 to n-1 do
Begin
Db:=d;{Lưu lại mảng d ở lần lặp i-1}
For j:=1 to n-i do
If s[j]=s[i+j] then d[j]:=d1[i+1]+2 {Tương tự như (1)}
Else
d[j]:=max(d[j],d[j+1]);
{Tương tự như (2)}
d1:=db; {lưu lại d1 băng lần lặp thứ i-2 của d}
end;
Writeln('Số ký tự cần thêm:',n-d[1]);
```

Nếu bạn muốn biết thêm đề và lời giải cũng như test các bài IOI, hãy gửi email theo địa chỉ: **quatang6**

## Cùng trao đổi về đề thi chọn học sinh giỏi quốc gia - Bảng B năm 2002-2003

**Dương Bích Ngọc**

*Tiếp tục bài viết về "Đôi điều về đề thi chọn học sinh giỏi quốc gia bảng B năm 2002 − 2003" của tác giả Đinh Xuân Hải số báo 5 (năm 2003), Trong số báo này tôi muốn cùng các bạn trao đổi về 2 bài: Bài số 1 và Bài số 2. Bài số 2 đã được tác giả đề cập đến bằng thuật toán ghép cặp, vậy chúng ta cùng xem còn cách nào khác để giải quyết bài toán thú vị này không?*

Đề bài các bạn có thể xem lại trong cùng số báo tháng 5 (năm 2003), ở đây tôi không trình bày lại nữa.

**Bài số 1: Hướng đi**

Trước khi vào giải bài toán, chúng ta cùng phân tích những nhận xét sau:

*Nhận xét 1*: Từ một xâu nguồn S chỉ gồm các kí tự (L,R,C), ta chỉ thay đổi vị trí các kí tự trong S được xâu S'. Rôbốt sau khi chuyển động theo chỉ dẫn của S,S' lần lượt có hướng là h,h' thì h và h' giống nhau.

*Ví dụ:* Ban đầu rôbôt đang hướng về phía Đông (E)

Sau khi chuyển động theo chỉ dẫn của S, S', robot đều có cùng hướng N

Như vậy ta không cần quan tâm đến thứ tự các kí tự L,R,C trong S mà chỉ cần quan tâm tới số lần xuất hiện của chúng trong S (hay cũng là số lần rẽ trái, rẽ phải của rôbôt).

*Nhận xét 2:*

Giả sử rôbốt xuất phát từ một hướng u thid sau 4 lần rẽ trái hay rẽ phải liên tiếp nó lại trở về hướng u ban đầu. Nhận xét này giúp chúng ta giảm một cách tối đa việc thực hiện các di chuyển của rôbốt.

*Ví dụ:* từ nhận xét 1, ta tìm được số lần rẽ trái, rẽ phải lần lượt của rôbốt là t và p.

Vậy để xác định hướng của rôbôt, ta chỉ cần xét (t mod 4) lần rẽ trái và (p mod 4) lần rẽ phải.

Số lần xét luôn ≤ 6

Chương trình dưới đây đã sử dụng kết hợp hai nhận xét trên và có cấu trúc tương đối đơn giản.

```
Program Huongdi;
Const fi= 'whatdir.inp';
fo= 'whatdir.out';
c1:array [1..8] of char= ('E', 'E','S','S','N','N','W','W');
c2:array [1..8] of char= ('L', 'R','L',' R','L','R','L','R');
c3:array [1..8] of char= ('N', 'S','E',' W','W','E','S','L');
var x: string;
t,p,d: integer;
tich: array[0..200] of integer;
xp: char;
f: text;
Procedure doc_file;
```

```
Begin
Assign(f,fi); reset(f);
Readln(f,xp);
Readln(f,x);
Close(f) ;
End;
Procedure khoi_tao;
Begin
x:= '('+x+')';
Fillchar(tich,sizeof(tich),0);
T:=0; p:=0; d:=0; tich[d]:=1;
End;
Procedure Tinh_t_p;
Var c: char; y: string;
L,i,k,so,code: integer;
Begin
L: =length(x); i:=1;
While (i<=l) do
Begin
c:=x[i];
if c='L' then
begin
t:=(t+tich[d]) mod 4;
if (x[i-1] in ['0'…'9']) then dec(d);
end
else
if c='R' then
begin
p:=(p+tich[d]) mod 4;
if (x[i-1] in ['0'…'9']) then dec(d);
end
else
if c=')' then dec(d)
else
if (c in ['0'…'9']) then
begin
for k:=i to l do
if not (x[k] in ['0'…'9']) then break;
y:=copy(x,i,k-i);
val(y,so, code);
inc(d);
tich[d]:=(tich[d-1]*so) mod 4;
i:=k-1;
end;
inc(i);
end;
```

```
end;
Procedure ghinhan;
Var i,k: integer;
Begin
Assign(f,fo); rewrite(f);
For i:=1 to t do
For k:=1 to 8 do
If (c1[k]=xp) and (c2[k]='L') then
Begin
xp:=c3[k]; break;
End;
For i:=1 to p do
For k:=1 to 8 do
If (c1[k]=xp) and (c2[k]='R') then
Begin
xp:=c3[k];
break;
end;
writeln(f,xp);
close(f);
end;
BEGIN
docfile;
khoitao;
Tinh_t_p;
Ghinhan;
END.
```

**Bài số 2. Truyền tin**

+) ***Trước tiên ta khởi tạo một mảng a (100x200)***

- a[i,j]=1 với i:= 1..N;j:=1..N;
- ngoài ra a[i,j]=0.

Vì m ≤10 000 nên ta phải vừa đọc vừa xử lý. Ta đọc từng dòng 2 xâu x,y (x là xâu xuất phát, y là xâu nhận). Nếu (y[i]<>x[j]) thì a[i,j]=0 với 0<i,j≤n. Trừ trường hợp này ra a[i,j] giữ nguyên.

Các bạn chú ý cách xây dung mảng a này có khác so với bài của tác giả Đinh Xuân Hải.

+) ***Trước khi vào tìm kết quả, ta có một số nhận xét sau:***

- Xét dòng thứ i của mảng a: Nếu tìm thấy duy nhất 1 chỉ số j mà a[i,j]=1, thì nếu tồn tại đường truyền tin ta có bit i phải tương ứng với bít j. Nên ta có vòng lặp

```
for t:=1 to n do
if (t<>i) then a[t,j]:=0;
```

ta làm tương tự với cột j (j:=1..n).

- Nhận xét thứ 2 là:

Nếu tồn tại một dòng i (hay 1 cột j) mà a[i,t]=0 (hay a[t,j]=0) t=1..n thì xuất −1.

+) ***ở bài tập này tôi muốn giới thiệu với các bạn về thuật giải tìm luồng cực đại trong mạng.***

- Trước tiên ta thay đổi gt mảng a bằng cách: Nếu a[i,j]=1 thì gán a[i,j+n]=1 và a[i,j]=0.

- Thêm vào tập đỉnh hai đỉnh S và T.
S nối với các đỉnh i:=1..n.
T nối với các đỉnh i= (n+1)…2*n
Vì thế mà mảng a có kích thước 100x200.
- Cung có trọng số là 1 nếu i được nối với j hay a[i,j]=1 đ c[i,j]=1. ngược lại thì là 0: c[i,j]=0.
- Ta tìm giá trị luồng cực đại lần đầu tiên. Nếu giá trị <n thì xuất −1.
Trong trường hợp ngược lại ta được mảng px có n phần tử. Trong đó f[i,px[i]]=1 với f là giá trị của luồng trên các cung của đồ thị.
*Sau đó ta thực hiện:*

```
For i:=1 to n do
Begin
c[i,px[i]]:=0;
Tìm giá trị của luồng cực đại;
If gia_tri=n then
Begin
Xuat −1;
Halt;
End
else
c[i,px[i]]=1;
end;
```

Đoạn chương trình này để xem còn quy luật nào không? Nó cũng tương tự như đoạn tìm quy luật của tác giả Đinh Xuân Hải. Nếu ta tìm được duy nhất1 quy luật thì mảng px cần phải trả lại giá trị: px[i]:=px[i]-n (hay bit i sẽ tương ứng với bit (px[i]-n)).
Tư tưởng của cách làm là như vậy. Dựa vào cấu trúc trên và cách sử dụng thuật toán tìm luồng cực đại trong mạng, các bạn hoàn toàn có thể viết được chương trình. Tôi hy vọng từ thuật toán nêu trên và thuật toán cặp ghép của tác giả Đinh Xuân Hải, các bạn có thể tìm thấy một cách giải hay cho riêng mình.
Nếu có gì góp ý xi liên hệ với tôi qua địa chỉ sau: *Dương Bích Ngọc* − Lớp 11Tin Trường THPT Chuyên Thái Nguyên hoặc qua địa chỉ mail: SVDuongNgoc@Yahoo.com.

## Kỳ thi chọn học sinh giỏi quốc gia - Tin học Bảng A

**TH&NT**

***Hãy lập trình giải các bài toán sau:***

**Bài 1. Phân đoạn *Tên file chương trình*: SEGPAR.???**

Cho dãy số nguyên $a_1, a_2, \ldots, a_n$ và số nguyên dương $k$. Ta gọi $k$-phân đoạn của dãy số đã cho là cách chia dãy số đã cho ra thành $k$ đoạn, mỗi đoạn là một dãy con gồm các phần tử liên tiếp của dãy. Chính xác hơn, một $k$-phân đoạn được xác định bởi dãy chỉ số

Đoạn thứ $i$ là dãy con . Ở đây quy ước $n_0 = 0$

**Yêu cầu:** Hãy xác định số $M$ nhỏ nhất để tồn tại $k$-phân đoạn sao cho tổng các phần tử trong mỗi đoạn đều không vượt quá $M$.

**Dữ liệu:**

Vào từ file văn bản SEGPAR.INP.

- Dòng đầu tiên chứa hai số nguyên $n$ và $k$ ($1 \le k \le n \le 15000$);

- Dòng thứ $i$ trong số $n$ dòng tiếp theo chứa số nguyên $a_i$ ($|a_i| \le 30000$), $i$ =1, 2, …, $n$.
Các số cạnh nhau trên một dòng trong file dữ liệu cách nhau ít nhất một dấu cách.
**Kết quả:** Ghi ra file SEGPAR.OUT một số nguyên duy nhất là giá trị $M$ tìm được.
**Ví dụ:**

**Bài 2: Pháo hoa**
***Tên chương trình*: FIREWK.???**
Nhằm chào mừng các ngày lễ lớn trong năm 2005 người ta đã chế tạo một loại đạn pháo hoa mới, khi bắn, đạn nổ thành bông hoa $2n$ cánh màu ( $1 \le n \le 30$). Nguyên vật liệu cho phép tạo được $m$ màu khác nhau, đánh số từ 1 đến $m$ ($2 \le m \le 32$).
Để đảm bảo tính mỹ thuật, việc chuyển tiếp màu giữa 2 cánh hoa kề nhau phải tuân theo quy tắc chuyển màu cầu vồng sau đây:
- Bên cạnh cánh hoa màu $i$ phải là cánh hoa màu $i$-1 hoặc $i$+1, với $1 < i < m$,
- Bên cạnh cánh hoa màu 1 chỉ có thể là cánh hoa màu 2,
- Bên cạnh cánh hoa màu $m$ chỉ có thể là cánh hoa màu $m$-1.
Một bông hoa không nhất thiết phải có đầy đủ $m$ màu. Mỗi bông hoa tương ứng với một vòng tròn $2n$ số thể hiện màu của các cánh hoa. Ví dụ, hình 1 là bông hoa 24 cánh ($n$ = 12) và hình 2 là vòng tròn số tương ứng với nó. Mỗi bông hoa được mô tả bằng dãy $2n$ số nguyên liệt kê các chỉ số màu của các cánh hoa theo chiều kim đồng hồ. Ví dụ, bông hoa ở hình 1 có thể được mô tả bằng dãy số

***3 4 3 2 1 2 3 4 3 2 1 2 3 4 3 2 1 2 3 4 3 4 3 2***
Dãy có thứ tự từ điển nhỏ nhất trong các dãy có thể dùng để mô tả hoa được gọi là mã hoa. Khi đó, mã hoa của bông hoa ở hình 1 sẽ là
***1 2 3 4 3 2 1 2 3 4 3 2 1 2 3 4 3 4 3 2 3 4 3 2***
Trong các ngày lễ, Ban tổ chức yêu cầu bắn các đạn pháo hoa $2n$ cánh có đúng $k$ cánh màu C ($0 \le k \le 2$). Các mã hoa thỏa mãn yêu cầu vừa nêu cũng được sắp xếp theo thứ tự từ điển và đánh số bắt đầu từ 1. Hơn nữa, nhằm tạo ra các hoa không giống nhau, đội bắn pháo hoa cần đảm bảo hai viên đạn pháo hoa bắn liên tiếp phải có mã khác nhau. Do vậy, người ta đã thiết kế một Hệ thống chụp ảnh và phân tích tự động để báo cho đội bắn pháo hoa biết số thứ tự của viên đạn pháo hoa vừa nổ trên trời. Em được giao viết chương trình giải quyết nhiệm vụ chính trong phần mềm phân tích tự động này.
**Yêu cầu**:
Cho biết $n$, $m$, $k$ và C. Gọi X là tập tất cả các mã hoa $2n$ cánh có đúng $k$ cánh màu C.
Hãy xác định số lượng $p$ các phần tử của X;
Cho một mã hoa nào đó trong tập X. Hãy xác định số thứ tự từ điển của nó trong X.
**Dữ liệu**: Vào từ file văn bản FIREWK.INP. Dòng đầu tiên chứa 4 số nguyên $n$, $m$, $k$ , C; dòng tiếp theo chứa $2n$ số nguyên mô tả một mã hoa.
Các số trên một dòng của file dữ liệu cách nhau ít nhất một dấu cách.
**Kết quả**: Đưa ra file văn bản FIREWK.OUT. Dòng đầu tiên ghi số nguyên $p$; dòng tiếp theo ghi số thứ tự tìm được của mã hoa.
**Ví dụ**:

**Ngày 2**
**Bài 3. Bộ sưu tập**
*Tên chương trình: COLLECT.???*

Một bộ sưu tập tiền xu cổ được coi là có giá trị phải gồm không ít hơn $Z_0$ đồng tiền vàng, $S_0$ đồng tiền bạc và $M_0$ đồng tiền đồng. Bộ sưu tập ban đầu của Alibaba có một số lượng nhất định các đồng tiền vàng, bạc và đồng nhưng chưa phải là một bộ sưu tập có giá trị. Tại Trụ sở của Hiệp hội những người sưu tầm tiền cổ có đặt một máy đổi tiền để giúp hội viên đổi được các bộ sưu tập có giá trị. Tuy nhiên, máy đổi chỉ hỗ trợ việc đổi tiền trọn gói theo quy tắc đổi gói ($Z_1$,$S_1$,$M_1$) lấy gói ($Z_2$,$S_2$,$M_2$) đồng tiền. Các quy tắc đổi tiền khác nhau từng đôi một, được gán số hiệu tuần tự 1,2,3,... và được công bố trước. Hội viên có thể tạo gói tiền thích hợp từ bộ sưu tập của mình để thực hiện việc đổi tiền. Các đồng tiền nhận được sau mỗi lần đổi được gộp lại với các đồng tiền mà hội viên đang có để thành một bộ sưu tập mới và có thể được sử dụng để đổi trong những lần sau nếu cần. Số lần đổi không hạn chế, tuy nhiên, là người thực dụng, Alibaba luôn cố gắng giảm tới mức tối đa số lần đổi tiền. Mặt khác, để ngăn chặn việc đầu cơ, Hiệp hội quy định, trong mọi thời điểm, mỗi hội viên không được giữ quá 4 đồng tiền mỗi loại và không được phép đổi tiếp khi đã đổi được một bộ sưu tập có giá trị.
**Yêu cầu:** Cho biết số lượng Z, S, M các đồng tiền vàng, bạc, đồng mà Alibaba có ban đầu và các quy tắc đổi tiền. Hãy chỉ ra tất cả các bộ sưu tập tiền cổ có giá trị mà Alibaba có thể có được sau một số lần đổi không vượt quá k cho trước.
**Dữ liệu:** Vào từ file văn bản COLLECT.INP.
- Dòng đầu ghi số nguyên dương k ($k \le 1000$); dòng thứ 2 ghi 6 số nguyên không âm Z, S, M, $Z_0$, $S_0$, $M_0$ ($0 \le Z, S, M, Z_0, S_0, M_0 \le 4$.
- Các dòng tiếp theo mỗi dòng ghi 6 số nguyên không âm $Z_1$, $S_1$, $M_1$, $Z_2$,$S_2$, $M_2$ xác định một quy tắc đổi tiền ($0 \le Z_1, Z_2, S_1, S_2, M_1, M_2 \le 4$).
**Kết quả:** Đưa ra file văn bản COLLECT.OUT.
- Nếu không tồn tại cách đổi để có được bộ sưu tập có giá trị, file kết quả chỉ gồm một số -1. Trong trường hợp ngược lại, dòng đầu ghi số v là số các bộ tiền cổ có giá trị mà Alibaba có thể đổi được.
- Dòng thứ i trong v dòng tiếp theo ghi 4 số nguyên $Z_i$, $S_i$, $M_i$, $k_i$ mô tả bộ sưu tập có giá trị thứ i và số lần đổi $k_i$ ít nhất không vượt quá k cần thực hiện để có được bộ sưu tập ấy.
Các số trên một dòng của file dữ liệu và file kết quả đặt cách nhau ít nhất một dấu cách.
***Ví dụ:***

**Bài 4. Khuôn thép**
***Tên chương trình: STEEL.???***
Để chuẩn bị cho Lễ hội kỷ niệm 30 năm ngày Chiến dịch Hồ Chí Minh toàn thắng, giải phóng miền Nam, thống nhất đất nước, người ta cần gia công các loại khuôn thép có hình dạng là các hình đa giác lồi M đỉnh. Mỗi khuôn thép được thiết kế trên một tấm thép cũng có hình dạng là một hình đa giác lồi N đỉnh, không có cạnh nào của khuôn thép nằm gọn trên một cạnh của tấm thép. Để tiện cho việc gia công, khuôn thép được vẽ sao cho hai đường thẳng chứa hai cạnh không kề nhau của nó không cắt nhau ở bên trong tấm thép.
Công việc chính cần làm trong quá trình gia công là sử dụng máy cắt để cắt được khuôn thép từ tấm thép ra. Rõ ràng là cần phải thực hiện M nhát cắt. Mỗi nhát cắt được thực hiện bằng cách chọn một cạnh nào đó của khuôn thép và cắt theo đường thẳng chứa cạnh ấy chia tấm thép thành hai phần, một phần chứa khuôn thép cần gia công. Chi phí cắt khuôn thép là tổng chiều dài của các đường cắt.

Trên hình 1 và 2, tấm thép là tứ giác được tô nhạt, khuôn thép là hình vuông được tô bằng các gạch đậm. Các nét gạch đứt là các đường cắt với tổng chi phí bằng 6.5 đơn vị.

**Yêu cầu**: Cho biết hình dạng tấm thép và khuôn thép cần gia công. Hãy tìm phương án cắt khuôn thép có chi phí nhỏ nhất.
**Dữ liệu**: Vào từ file văn bản STEEL.INP: Dòng đầu ghi số N ($3 \le N \le 2000$) là số đỉnh của tấm thép; N dòng tiếp theo, mỗi dòng ghi 2 số thực x và y ($-10^4 < x, y < 10^4$), là toạ độ N đỉnh của tấm thép được liệt kê theo chiều kim đồng hồ bắt đầu từ một đỉnh nào đó; dòng tiếp theo ghi số M ($3 \le M \le 2000$) là số đỉnh của khuôn thép; cuối cùng là M dòng, mỗi dòng ghi 2 số thực x và y ($-10^4 < x, y < 10^4$) là toạ độ M đỉnh của khuôn thép được liệt kê theo chiều kim đồng hồ bắt đầu từ một đỉnh nào đó. Các số trên một dòng cách nhau ít nhất một dấu cách.
**Kết quả:** Đưa ra file văn bản STEEL.OUT chi phí nhỏ nhất tìm được với độ chính xác tới 4 chữ số sau dấu chấm thập phân.
***Ví dụ:***

## Hội thi Tin học trẻ không chuyên toàn quốc lần thứ XI

**TH&NT**

***Lập trình thực hiện các công việc sau đây***
**Bài 1. Tập bắn**
Đội tuyển bắn cung Việt nam gồm 6 xạ thủ đang tích cực luyện tập tại trung tâm huấn luyện thể thao QG để chuẩn bị cho SEAGAME 23. Một buổi tối nọ, với mục đích luyện tập thêm để nâng cao thành tích cá nhân, một thành viên của đội lén đến phòng tập một mình và bắn vào 6 cái bia trong phòng tập mỗi bia một phát. Sau đó anh ta đến quán cà phê giải khát. Thật bất ngờ là trong quán cà phê anh ta gặp cả 5 xạ thủ còn lại, và họ cũng đã vừa thực hiện việc làm như vậy. 6 thành viên vui vẻ trò chuyện về kết quả bắn tập thêm của mình, mỗi thành viên cho biết số lượng phát tên trúng bia của mình. Không ngờ, chủ quán lại là bạn thân của huấn luyện viên đội tuyển bắn cung và ông đã ghi lại kết quả đó rồi báo ngay cho huấn luyện viên đội tuyển bắn cung. Huấn luyện viên là một người rất có trách nhiệm với công việc, vì thế, ngay lập tức trong đêm đó ông đã đến phòng tập ghi lại vết tích còn để lại trên 6 bia trong phòng tập. Mặc dù các xạ thủ đã rút mũi tên trúng đích ra khỏi bia tập, nhưng vết cắm của mũi tên còn để lại trên bia (bạn được biết thêm rằng không có 2 hoặc nhiều hơn mũi tên để lại cùng một vết). Huấn luyện viên đội tuyển rất băn khoăn không biết là có xạ thủ nào nói sai thành tích của mình tại quán cà phê hay không?
**Yêu cầu**: Giúp huấn luyện viên đội tuyển quốc gia xác định xem: hoặc là chắc chắn có xạ thủ nói sai sự thật về kết quả luyện tập, hoặc nếu trái lại có bao nhiêu khả năng bắn (tức là, một cách chỉ ra mỗi xạ thủ đã bắn trúng những bia nào) đạt được kết quả trùng khớp với các thông tin về kết quả bắn tập từ các xạ thủ cũng như từ vết tích để lại trên các bia.
**Dữ liệu:** Vào từ file văn bản TAPBAN.INP
- Dòng đầu tiên gồm 6 số nguyên là số lượng mũi tên trúng bia của 6 xạ thủ;
- Dòng thứ hai gồm 6 số nguyên là số lượng mũi tên trúng đích trên 6 bia mà huấn luyện ghi nhận được dựa theo vết tích để lại trên bia.
**Kết quả:** Ghi ra file văn bản TAPBAN.OUT:
- Dòng thứ nhất ghi số nguyên k là số lượng khả năng tìm được;
- Nếu k = 1 thì mỗi dòng trong 6 dòng tiếp theo ghi kết quả bắn bia của một xạ thủ bao gồm 6 số nguyên, số thứ i là 1 nếu xạ thủ bắn trúng bia thứ i và bằng 0 nếu trái lại, i = 1,2,…,6.
**Ví dụ:**

**Bài 2. Robot**

Bên lề cuộc thi Robocon năm nay, các đội còn được mời tham dự cuộc chơi: "Robot tìm đường đi". Thể lệ cuộc chơi như sau: Robot của các đội sẽ phải tìm đường đi trên một sa bàn hình chữ nhật được chia thành lưới m x n ô vuông đều nhau gồm m dòng và n cột. Các dòng của lưới được đánh số từ 1 đến m, từ trên xuống dưới. Các cột của lưới được đánh số từ 1 đến n, từ trái qua phải. Ô ở vị trí giao của dòng i và cột j gọi là ô (i,j). Có hai loại ô vuông: ô tiêu thụ năng lượng và ô nạp năng lượng. Di chuyển qua ô tiêu thụ năng lượng robot sẽ mất một lượng năng lượng nhất định. Đi vào ô nạp năng lượng robot sẽ được nạp đầy năng lượng và không mất năng lượng. Bình năng lượng của robot chỉ chứa được một lượng năng lượng nhất định và robot không thể di chuyển nếu không còn năng lượng. Robot chỉ có thể di chuyển từ một ô sang ô có cạnh liền kề.

**Yêu cầu:** Hãy lập trình giúp robot tìm đường đi từ ô (1,1) đến ô (m,n) sao cho tổng năng lượng tiêu thụ trên đường đi là nhỏ nhất. Biết rằng ô xuất phát và ô đích cũng là các ô nạp năng lượng. Robot có thể di chuyển qua một ô bất kỳ nhiều lần.

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

- Dòng đầu tiên chứa 2 số nguyên dương m và n là số dòng và số cột của sa bàn, hai số cách nhau một dấu trắng ($m,n \leq 100$);
- Dòng thứ hai chứa số nguyên p ($1 \leq p \leq 32767$) là dung lượng của bình năng lượng của robot.
- Dòng thứ i trong số m dòng tiếp theo chứa n số nguyên không âm $a_{i1}, a_{i2}, \ldots, a_{im}$ ($a_{ij} \leq 20000$, i = 1,2,…,m; j=1,2,…,n) tương ứng với các ô trên dòng i của sa bàn. Nếu aij là số dương thì ô (i,j) là ô tiêu thụ năng lượng và khi đi qua ô này robot tốn hao một lượng năng lượng là $a_{ij}$. Nếu $a_{ij} = 0$ thì ô (i,j) là ô nạp năng lượng.

**Kết quả:** Ghi ra file văn bản ROBOT.OUT số nguyên s là tổng năng lượng tiêu thụ trên đường đi tìm được. Ghi s = -1, nếu không tồn tại cách đi để robot đạt đến đích.

**Ví dụ:**

## Hội thi Tin học trẻ không chuyên toàn quốc

**TH&NT**

**Lập trình giải các bài toán sau:**

**Bài 1. Vòng trang sức**

Tên file chương trình: Jewel.???

Người ta xâu N viên đá quý kích thước giống nhau thành một vòng đeo cổ ($5 \leq N \leq 120$), mỗi viên có một màu trong số các màu đánh số từ 1 đến 9. Để tăng tính độc đáo cho vòng trang sức quý này, người ta định lắp khoá đeo vào vị trí ở giữa hai viên ngọc nào đó sao cho khi mở vòng ra, không phụ thuộc vào việc cầm đầu dây nào bên tay phải, ta đều được một chuỗi hạt giống nhau, tức là màu của viên đá thứ i từ trái sang không phụ thuộc vào cách cầm.

Yêu cầu: Cho xâu S độ dài N ký tự, chỉ chứa các ký tự số từ 1 đến 9 xác định cấu hình của vòng đã được xâu. Hãy xác định số vị trí khác nhau có thể lắp khoá đeo. Dữ liệu: Vào từ file văn bản JEWEL.INP một dòng chứa xâu S.

Kết quả: Đưa ra file văn bản JEWWL.OUT số nguyên K - số vị trí khác nhau tìm được. K = 0 nếu không thể có vị trí nào thích hợp.

**Ví dụ:**

**Bài 2. Bộ chuyển hướng**

Tên file chương trình: Switch.???

Bộ chuyển hướng tín hiệu quang có dạng hình hộp chữ nhật kích thước M*N*1, được lắp ráp từ M*N phần tử, mỗi phần tử là một khối lập phương cạnh đơn vị. Bốn mặt quanh của một phần tử ký hiệu là 1,2,3 và 4 (xem hình 2). Tia la de luôn được chiếu vuông góc với một mặt bên nào đó và khi thoát ra ở mặt bên nào đó - cũng vuông góc với mặt đó. Có 3 loại phần tử:

- Loại 0 – phần tử trong suốt, tia la de chiếu qua nó không đổi hướng,
- Loại 1 – Chiếu vào mặt 1 thì ra ở mặt 2 và ngược lại, chiếu vào mặt 3 thì ra mặt 4 và ngược lại,
- Loại 2 – Chiếu vào mặt 1 thì ra ở mặt 4 và ngược lại, chiếu vào mặt 2 thì ra mặt 3 và ngược lại.

Các hàng của hộp được đánh số từ 1 đến M từ trên xuống dưới, các cột được đánh số từ 1 đến N từ trái qua phải. Người ta đã thiết kế một bộ chuyển hướng trong đó có K+1 phần tử loại khác 0, đảm bảo khi chiếu tia la de từ ngoài vào hộp tại mặt hv của phần tử (iv,jv) trên biên thì nó sẽ ra khỏi hộp qua mặt hr của phần tử (ir,jr) trên biên (phần tử (i,j) là phần tử biên nếu i bằng 1 hoặc M hay j bằng 1 hoặc N). Do sơ xuất ở khâu lắp ráp, người ta đã đặt nhầm một phần tử loại 0 vào vị trí đáng lẽ phải là phần tử loại khác 0 (xem hình 2, hình 3).

Yêu cầu: Cho biết M,N ( $0 < M,N \leq 150$), tọa độ (iv,jv),(ir,jr), hv , hr, tọa độ (ip,jp) và loại tp của các phần tử loại khác 0 đã lắp đúng theo thiết kế (p = 1, 2,..., K). Hãy xác định:

- Vị trí phần tử biên (ix,jx) nơi tia la de thoát ra khỏi hộp đã lắp,

Vị trí (i0,j0) của phần tử bị lắp nhầm và loại phần tử t0 cần thay thế vào đó.

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

- Dòng đầu tiên chứa 3 số nguyên M N K,
- Dòng thứ 2 chứa 3 số nguyên iv jv hv,
- Dòng thứ 2 chứa 3 số nguyên ir jr hr,

Dòng thứ p trong K dòng sau chứa 3 số nguyên ip jp tp.

Kết quả: Đưa ra file văn bản SWITCH.OUT:

- Dòng thứ nhất chứa 2 số nguyên ix jx,
- Dòng thứ 2 chứa 3 số nguyên i0 j0 t0.

Trong trường hợp có nhiều lời giải - chỉ cần đưa ra một trong số đó.

Các số trên một dòng cách nhau ít nhất một dấu cách.

**Ví dụ:**

## Nhận xét - Hướng dẫn - Lời giải

**TH&NT**

**BẢNG B**

**Bài 1. Vòng trang sức**

Đây là một bài toán khá đơn giản, chỉ đòi hỏi khả năng xử lý mảng một chiều và sử dụng thuần thục các giải thuật lặp. Bạn cứ thử đặt tại mọi nơi của chiếc vòng này và lần lượt đi về hai đầu chuỗi hạt kiểm tra sự giống nhau, việc này không có gì phức tạp trừ việc chỉ số của mảng chạy "lung tung" khi đến cuối mảng – đang kiểm tra trên chuỗi hạt nhưng có khi tăng, lại có khi giảm. Đây chính là điều làm các bạn bối rối nhất. Do các bạn học sinh bảng B đa số chưa có được tư duy sắc sảo nên đã mất thời gian xử lý việc này – trong khi chỉ cần có một

cách tổ chức dữ liệu "hai vòng" thì việc xử lý này không còn nữa. Cách tổ chức đó như sau: Bạn khai báo xâu S chứa chuỗi hạt rộng lên đến 250 ký tự. Cho chuỗi hạt "dài gấp đôi", nghĩa là giả sử S có chiều dài d, cho S thành chiều dài 2d mà S[d+i] = S[i] (i=1..d), khi đó có thể tưởng tượng chuỗi hạt được đánh chỉ số giả đến 2 lần:

Khi cần kiểm tra nửa chuỗi hạt từ S[10] đến S[3] bạn khỏi lo phải tăng hay giảm chỉ số mà cứ đi từ S[10] đến S[15] là được. Lợi ích của phép "gấp đôi" là như thế. Để rõ hơn về các giải thuật thú vị như thế này mời bạn tìm đọc cuốn "Bắn tàu trên biển" - TS Nguyễn Xuân Huy (có thể liên hệ qua tòa soạn để có cuốn sách này).

**Bài 2. Bộ chuyển hướng**

Đề bài toán có vẻ phức tạp nhưng thực chất chỉ là việc thử chọn đặt các phần tử vào các vị trí khác nhau rồi tiến hành "bắn tia lade" xem kết quả thế nào. Việc phức tạp nhất ở việc "bắn tia lade", tùy vào giá trị của mảng thiết bị mà tia lade chuyển sang các vị trí khác nhau – việc này chỉ đơn giản là các phép kiểm tra. Bạn chỉ cần tỉnh táo xem xét các vị trí biên là bài toán trở thành hoàn hảo.

Với tư duy của các bạn bảng B, bài toán có vẻ phức tạp, để những bài toán như thế này trở nên đơn giản các bạn hãy áp dụng phương pháp mô hình hóa bài toán thành lược đồ các đầu vào và yêu cầu đầu ra dạng "Tin học" – nghĩa là bài toán cho các số nào, cho mảng gì, quan hệ với nhau ra sao, ta cần đưa ra số nào, theo quy luật nào,… từ đó giải quyết bài toán trên các "đối tượng Tin học" sẽ dễ dàng hơn.

**BẢNG C**

**Bài 1. Tập bắn**

Ta mô hình hoá bài toán thành một bài toán đơn giản đối với ma trận 2 chiều như sau:

Xây dựng một ma trận hai chiều A[1..6,1..6], mỗi hàng i của ma trận gồm 6 phần tử thể hiện 6 bia của người thứ i bắn, do mỗi người chỉ bắn vào một bia nên A[i,j] chỉ có giá trị là 0 (bắn trượt) hoặc 1 (bắn trúng). Đây chính là ma trận thể hiện một khả năng có thể xảy ra mà người Huấn luận viên dự đoán, và theo đề bài bạn phải xuất ra ma trận này nếu chỉ có một khả năng duy nhất.

Gọi số phát trúng của 6 người là T[1..6], số phát trúng trên 6 bia là B[1..6]. Nhận xét: T chính là ma trận thể hiện tổng các hàng của A. B là ma trận thể hiện tổng các cột của A.

Đề bài đã cho mảng T và mảng B, nhiệm của bạn là tìm số ma trận A như vậy, nếu chỉ có 1 ma trận duy nhất thì cho biết mảng A đó.

Thuật toán đệ quy là thuật toán áp dụng tốt ở bài toán này để thử và tìm ra ma trận A, với nhánh cận hợp lý nhờ 2 mảng T và B sẽ làm cho đệ quy dễ dàng và nhẹ nhàng hơn. Chẳng hạn khi thử chọn đến phần tử A[i,j] (bằng 0 hoặc 1) bạn kiểm tra tổng hàng và tổng cột trước nó đã bằng T[i] hay B[j] chưa, nếu đã bằng, đương nhiên A[i,j] và các phần tử "sau" hay "dưới" nó đều bằng 0.

Với tư tưởng như vậy bạn có thể viết chương trình thực hiện dễ dàng.

**Bài 2. Robot**

Kích thước của bài toán khá lớn nên bạn nào sử dụng Đệ quy, Quay lui nói chung đều không đạt.

Bạn nào sử dụng phương pháp Duyệt theo chiều rộng (Breadth First Search - BFS) cổ điển không có cải biến – nghĩa là làm khá giống như bài toán "Mã đi tuần trên bàn cờ vua" cũng không đạt, nhưng nhiều bạn đã sử dụng cách này. Xét test sau đây:

Nếu sử dụng BFS cổ điển sẽ tìm ra lối đi đến ô (2,2) là (1,1) → (1,2) → (2,2) và hết năng lượng buộc phải sang (2,3) sau đó sang (3,3) và không tìm được đường đi. Tuy nhiên có một cách đi khác đến ô (2,2) là: (1,1) → (1,2) → (1,3) → (2,3) → (2,2) mà vẫn còn năng lượng là 1, tiếp tục đi về đích theo đường: (2,2) → (3,2) → (4,2) → (5,2) → (5,3). Như vậy để đi đến (2,2) phương pháp BFS nguyên bản không đảm bảo tối ưu, từ đó cho thấy đó là phương pháp không đúng đắn khi giải bài toán này.

Để giải bài toán này bạn có thể sử dụng phương pháp quy hoạch động, hoặc áp dụng thuật toán tìm đường đi ngắn nhất trên một đồ thị giả một cách khéo léo theo kiểu thuật toán Dijkstra. Dù là sử dụng cách nào thì thực chất cũng là việc Loang (tức là duyệt theo chiều rộng) và thay đổi các giá trị tối ưu tại các ô đã đi đến theo quy tắc sau đây:

Những ô nằm sau ô nạp năng lượng trên hành trình của Robot là những ô có thể đánh dấu việc tối ưu để không cho Robot quay lại ô đó, những ô không nằm sau ô nạp năng lượng nhưng nằm sau ô đã đánh dấu tối ưu thì cũng được đánh dấu tối ưu.

Từ đó bạn có thể hình dung ra một giải thuật, có thể nói là tựa như BFS, tựa như Dijkstra, tựa như Quy hoạch động… - như thế không có nghĩa là quá khó, đó đơn giản là giải thuật dựa trên tư tưởng đã nói ở trên và áp dụng với một cấu trúc dữ liệu hợp lý là bài toán được giải quyết trọn vẹn. Trên số tháng sau sẽ đăng chi tiết về giải thuật "lai tạo" này và chương trình thực hiện bài toán.

**Một số ý kiến về đề thi năm nay**

- Bạn Trần Lê Quân (Đoàn Đồng Tháp, bảng A): Đề thi với học sinh tiểu học như vậy là khó, kiến thức quá rộng, có những "thứ" chưa được nghe đến bao giờ.
- Nguyễn Thị Thùy Linh (Đoàn Nghệ An, bảng B): Đề khó hơn mọi năm (năm nay bạn đã thi Toàn quốc lần thứ 2), phần trắc nghiệm có nhiều câu hỏi hay.
- Xa Thành Nam (Đoàn Hòa Bình, bảng B): Đề khó, đặc biệt là bài 2 rất khó.
- Đỗ Văn Trung (Đoàn Ninh Bình, bảng C): Nhìn chung đề thi ở mức độ tương đối, không khó và cũng không dễ, cả hai bài đều sử dụng giải thuật quay lui, bài Robot bạn chưa làm được một số trường hợp. Phần thi trắc nghiệm vừa phải.
- Trần Thiện Khiêm (Đoàn Quảng Trị, bảng C): Đề tương đối khó, bài "Tập bắn" chỉ làm được một trường hợp.
- Võ Thị Thùy Linh (Đoàn Hòa Bình, bảng C): Đề thi khó hơn năm ngoái (năm nay bạn đã thi Toàn quốc lần thứ 2) khiến bạn chỉ làm được một nửa. Bài Robot khó hơn bài "Tập bắn".
- Huỳnh Ngọc Ân (Đoàn BCVT, bảng C): Bạn cho rằng bài tập bắn rắc rối, đề thi vừa với bạn nhưng khó hơn một chút thì hay hơn, thời gian làm bài không thừa nên test chưa kỹ. Bạn Ân thi Tin Học trẻ không chuyên lần này là lần thứ 4 với thành tích khá cao.
- Chú Đạo (Đoàn Bến Tre - 2 thí sinh bảng A và C): Các em đều cho rằng đề thi khó, phần trắc nghiệm kiến thức quá rộng.
- Phụ huynh bạn Linh (Đoàn Nghệ An): Các cháu nói đề thi quá rộng so với chương trình học và khả năng tự học của các cháu, nhiều kiến thức chưa được phổ cập rộng.

BBT tạp chí đã có cơ hội gặp gỡ, trò chuuyện và tặng tạp chí cho các bạn học sinh, các thầy cô, phụ huynh,… trong thời gian diễn ra hội thi. Các bạn học sinh từ khắp mọi miền tổ quốc bạn nào cũng sáng sủa thông minh, rất xứng đáng đại diện cho Đoàn, cho Tỉnh mình. Trên đây chỉ là một trong số ít ý kiến mà BBT được nghe, có thể thấy sơ bộ Đề thi là khá khó và rắc rối. Phần thi trắc nghiệm có kiến thức rộng khiến rất nhiều bạn phải "trả lời ngẫu nhiên" hay "đoán mò"! Tuy nhiên, rõ ràng là do tư duy phân tích của các bạn có khác nhau (và sự thực là cũng chưa tốt) nên có bạn thấy bài "Tập bắn" quá khó, có bạn lại thấy bài Robot quá

khó (bảng C) còn bài kia thì dễ (!). Nếu phân tích đề như Olypiad đã làm thì bài nào cũng đơn giản cả phải không các bạn? Chúc các bạn thành công hơn nữa ở các kỳ thi sau!

## Các kỳ thi Tin học trên thế giới

**Ngô Minh Đức**

*** Quốc tế**

Ngoài kỳ thi IOI nổi tiếng nhất thế giới mà trong số tháng 9/2005 này tạp chí đã đăng tin chi tiết còn có:

**ACM International Collegiate Programming Contest – ICPC**

ACM International Collegiate Programming Contest (info.acm.org/contest) được tổ chức hằng năm bởi tổ chức ACM (info.acm.org), là một kì thi có uy tín dành cho sinh viên các trường đại học diễn ra lần đầu tiên vào năm 1970.

Trong vòng 5 giờ, mỗi đội dùng một máy tính để giải các bài toán tin học. Một đội được phép nộp một bài toán nhiều lần. Bài nộp trong khi thi và sẽ được chấm ngay; tuy nhiên bài nộp sai sẽ bị trừ điểm.

Hơn 60 đội sẽ tham dự kì thi chung kết ICPC World Finals, thông thường được tổ chức vào tháng 2, 3 hoặc đầu tháng 4 hằng năm. Để dành được quyền tham dự kì thi chung kết, các đội phải trải qua vòng loại khu vực (regional contests) chia làm hơn 25 khu vực trên khắp thế giới. Trước đó, thông thường còn có các kì tuyển chọn nội bộ (local contests) để chọn ra các đội tham dự vòng loại khu vực.

**Internet Problem Solving Contest – IPSC**

IPSC (ipsc.ksp.sk) là kì thi lập trình đồng đội trên internet. Mục đích của cuộc thi là để so sánh khả năng giải toán tin học của mọi người trên khắp thế giới, tạo ra một sân chơi vui vẻ, bổ ích. Hàng năm có hàng trăm đội tham gia.

*** Khu vực**

**Balkan Olympiad on Informatics – BOI**

Kì thi BOI - Olympic tin học khu vực Balkan (http://www.csd.auth.gr/contests/boi.html) là một kì thi lập trình cho học sinh, sinh vịn các nước khu vực Balkan (bao gồm Albenia, Bulgaria, Cyprus, FYROM, Hy Lạp, Romania, Thổ Nhĩ Kỳ, Nam Tư).

BOI lần đầu tiên tổ chức tại Romania: http://skyblue.csd.auth.gr/boi/boi.html. BOI gần nhất là lần thứ 13 tại Bulagria: http://www.boi2004-plovdiv.org/ Thông tin chung về các kì thi BOI: http://infoman.musala.com/contests/boi/main.html

**Baltic Olympiad in Informatics – BOI**

Một kì thi nổi tiếng khác là Olympic tin học khu vực Baltic (cũng gọi tắt là BOI) được mô phỏng theo Olympic tin học quốc tế IOI. Các nước tham dự trong kì thi này bao gồm Estonia, Phần Lan, Latvia, Lithuania, Ba Lan và Thụy Điển

BOI gần nhất là lần thứ 11 tại Lithuania: http://ims.mii.lt/olimp/tin/boi2005/

**Central-European Olympiad in Informatics – CEOI**

Olympic tin học khu vực Trung Âu, gọi tắt là CEOI (http://ceoi.inf.elte.hu/intro.html) được tổ chức bởi Bộ Văn Hóa, Giáo Dục hoặc các tổ chức tương đương của một trong tám nước khu

vực Trung Âu. Theo điều lệ đã được chấp nhận bởi những người khởi xướng CEOI (http://ceoi.inf.elte.hu/rules.html), các đội của tám nước Ăo, Croatia, Cộng Hòa Séc, Hungary, Ba Lan, Romania, Cộng Hòa Slovak và Slovenia sẽ được mời tham dự với tư cách chính thức. Ngoài ra, nước chủ nhà có thể mời thêm các đội khác với tư cách khách mời.
CEOI2005 tổ chức tại Sárospatak, Hungary (28/7 – 5/8/2005): http://ceoi.inf.elte.hu/ceoi2005/
* Quốc gia Hầu như các nước đều có kì thi Olympic Tin học Quốc Gia để tuyển chọn học sinh tham dự kì thi tin học quốc tế IOI. Xin giới thiệu website về kỳ thi này của một số quốc gia:
Croatia: http://pubwww.srce.hr/hsin/;
Ba Lan: http://oi.edu.pl;
Anh: http://www.olympiad.org.uk;
Mỹ: http://usaco.uwp.edu/;
Singapore: http://www.comp.nus.edu.sg/~noi/
Để biết được đầy đủ chi tiết hơn về những kì thi tin học trên thế giới, mời các bạn truy cập website:
http://olympiads.win.tue.nl/ioi/misc/other.html và http://olympiads.win.tue.nl/ioi/noi/index.html. Hầu hết website của những kì thi đều có lưu giữ lại đề bài, lời giải và bộ test; là những phương tiện rất tốt để chúng ta có thể luyện tập kỹ năng giải bài toán tin học.

# Trần Minh Tâm

2

**Xin thầy hãy dạy cho con tôi**

**Một chuyện tình**


Đang tải

Gửi phản hồi