Hå sÜ ®µm (Chñ biªn)

®ç ®øc ®«ng – lª minh hoµng – nguyÔn thanh hïng

tµi liÖu gi¸o khoa

chuyªn tin

quyÓn 1

Nhµ xuÊt b¶n gi¸o dôc viÖt nam

2

C«ng ty Cæ phÇn dÞch vô xuÊt b¶n Gi¸o dôc Hµ Néi - Nhµ xuÊt b¶n Gi¸o dôc ViÖt Nam

gi÷ quyÒn c«ng bè t¸c phÈm.

349-2009/CXB/43-644/GD M4 sè : 8I746H9

3

LỜI NÓI ðẦU

Bộ Giáo dục và ðào tạo ñã ban hành chương trình chuyên tin học cho các

lớp chuyên 10, 11, 12. Dựa theo các chuyên ñề chuyên sâu trong chương trình

nói trên, các tác giả biên soạn bộ sách chuyên tin học, bao gồm các vấn ñề cơ

bản nhất về cấu trúc dữ liệu, thuật toán và cài ñặt chương trình.

Bộ sách gồm ba quyển, quyển 1, 2 và 3. Cấu trúc mỗi quyển bao gồm: phần

lí thuyết, giới thiệu các khái niệm cơ bản, cần thiết trực tiếp, thường dùng nhất;

phần áp dụng, trình bày các bài toán thường gặp, cách giải và cài ñặt chương

trình; cuối cùng là các bài tập. Các chuyên ñề trong bộ sách ñược lựa chọn mang

tính hệ thống từ cơ bản ñến chuyên sâu.

Với trải nghiệm nhiều năm tham gia giảng dạy, bồi dưỡng học sinh chuyên tin

học của các trường chuyên có truyền thống và uy tín, các tác giả ñã lựa chọn,

biên soạn các nội dung cơ bản, thiết yếu nhất mà mình ñã sử dụng ñể dạy học

với mong muốn bộ sách phục vụ không chỉ cho giáo viên và học sinh chuyên

PTTH mà cả cho giáo viên, học sinh chuyên tin học THCS làm tài liệu tham khảo

cho việc dạy và học của mình.

Với kinh nghiệm nhiều năm tham gia bồi dưỡng học sinh, sinh viên tham gia

các kì thi học sinh giỏi Quốc gia, Quốc tế Hội thi Tin học trẻ Toàn quốc,

Olympiad Sinh viên Tin học Toàn quốc, Kì thi lập trình viên Quốc tế khu vực

ðông Nam Á, các tác giả ñã lựa chọn giới thiệu các bài tập, lời giải có ñịnh

hướng phục vụ cho không chỉ học sinh mà cả sinh viên làm tài liệu tham khảo

khi tham gia các kì thi trên.

Lần ñầu tập sách ñược biên soạn, thời gian và trình ñộ có hạn chế nên chắc

chắn còn nhiều thiếu sót, các tác giả mong nhận ñược ý kiến ñóng góp của bạn

ñọc, các ñồng nghiệp, sinh viên và học sinh ñể bộ sách ñược ngày càng hoàn

thiện hơn .

Các tác giả

4

5

Chuyên ñề 1

THUẬT TOÁN

VÀ PHÂN TÍCH THUẬT TOÁN

1. Thuật toán

Thuật toán là một trong những khái niệm quan trọng nhất trong tin học. Thuật ngữ

thuật toán xuất phát từ nhà khoa học Arập Abu Ja'far Mohammed ibn Musa al

Khowarizmi. Ta có thể hiểu thuật toán là dãy hữu hạn các bước, mỗi bước mô tả

chính xác các phép toán hoặc hành ñộng cần thực hiện, ñể giải quyết một vấn ñề.

ðể hiểu ñầy ñủ ý nghĩa của khái niệm thuật toán chúng ta xem xét 5 ñặc trưng sau

của thuật toán:

• ðầu vào (Input): Thuật toán nhận dữ liệu vào từ một tập nào ñó.

• ðầu ra (Output): Với mỗi tập các dữ liệu ñầu vào, thuật toán ñưa ra các

dữ liệu tương ứng với lời giải của bài toán.

• Chính xác: Các bước của thuật toán ñược mô tả chính xác.

• Hữu hạn: Thuật toán cần phải ñưa ñược ñầu ra sau một số hữu hạn (có

thể rất lớn) bước với mọi ñầu vào.

• ðơn trị: Các kết quả trung gian của từng bước thực hiện thuật toán ñược

xác ñịnh một cách ñơn trị và chỉ phụ thuộc vào ñầu vào và các kết quả

của các bước trước.

• Tổng quát: Thuật toán có thể áp dụng ñể giải mọi bài toán có dạng

ñã cho.

ðể biểu diễn thuật toán có thể biểu diễn bằng danh sách các bước, các bước ñược

diễn ñạt bằng ngôn ngữ thông thường và các kí hiệu toán học; hoặc có thể biểu

diễn thuật toán bằng sơ ñồ khối. Tuy nhiên, ñể ñảm bảo tính xác ñịnh của thuật

toán, thuật toán cần ñược viết bằng các ngôn ngữ lập trình. Một chương trình là sự

biểu diễn của một thuật toán trong ngôn ngữ lập trình ñã chọn. Trong tài liệu này,

chúng ta sử dụng ngôn ngữ tựa Pascal ñể trình bày các thuật toán. Nói là tựa

Pascal, bởi vì nhiều trường hợp, ñể cho ngắn gọn, chúng ta không hoàn toàn tuân

6

theo quy ñịnh của Pascal. Ngôn ngữ Pascal là ngôn ngữ ñơn giản, khoa học, ñược

giảng dạy trong nhà trường phổ thông.

Ví dụ: Thuật toán kiểm tra tính nguyên tố của một số nguyên dương 2,

viết trên ngôn ngữ lập trình Pascal.

function is\_prime(n):boolean;

begin

for k:=2 to n-1 do

if (n mod k=0) then exit(false);

exit(true);

end;

2. Phân tích thuật toán

2.1. Tính hiệu quả của thuật toán

Khi giải một bài toán, chúng ta cần chọn trong số các thuật toán một thuật toán mà

chúng ta cho là “tốt” nhất. Vậy dựa trên cơ sở nào ñể ñánh giá thuật toán này “tốt”

hơn thuật toán kia? Thông thường ta dựa trên hai tiểu chuẩn sau:

1. Thuật toán ñơn giản, dễ hiểu, dễ cài ñặt (dễ viết chương trình).

2. Thuật toán hiệu quả: Chúng ta thường ñặc biệt quan tâm ñến thời gian

thực hiện của thuật toán (gọi là ñộ phức tạp tính toán), bên cạnh ñó

chúng ta cũng quan tâm tới dung lượng không gian nhớ cần thiết ñể lưu

giữ các dữ liệu vào, ra và các kết quả trung gian trong quá trình

tính toán.

Khi viết chương trình chỉ ñể sử dụng một số ít lần thì tiêu chuẩn (1) là quan trọng,

nhưng nếu viết chương trình ñể sử dụng nhiều lần, cho nhiều người sử dụng thì

tiêu chuẩn (2) lại quan trọng hơn. Trong trường hợp này, dù thuật toán có thể phải

cài ñặt phức tạp, nhưng ta vẫn sẽ lựa chọn ñể nhận ñược chương trình chạy nhanh

hơn, hiệu quả hơn.

2.2. Tại sao cần thuật toán có tính hiệu quả?

Kĩ thuật máy tính tiến bộ rất nhanh, ngày nay các máy tính lớn có thể ñạt tốc ñộ

tính toán hàng nghìn tỉ phép tính trong một giây. Vậy có cần phải tìm thuật toán

hiệu quả hay không? Chúng ta xem lại ví dụ bài toán kiểm tra tính nguyên tố của

một số nguyên dương 2.

function is\_prime(n):boolean;

begin

7

for k:=2 to n-1 do

if (n mod k=0) then exit(false);

exit(true);

end;

Dễ dàng nhận thấy rằng, nếu là một số nguyên tố chúng ta phải mất \_x0007\_ 2 phép

toán

. Giả sử một siêu máy tính có thể tính ñược trăm nghìn tỉ 10

phép

trong một giây, như vậy ñể kiểm tra một số khoảng 25 chữ số mất khoảng

\_x0010\_

\_x0012\_

\_x0016\_

~3170 năm. Trong khi ñó, nếu ta có nhận xét việc thử từ 2

ñến \_x0007\_ 1 là không cần thiết mà chỉ cần thử từ 2 ñến √ , ta có:

function is\_prime(n):boolean;

begin

for k:=2 to trunc(sqrt(n)) do

if (n mod k=0) then exit(false);

exit(true);

end;

{hàm sqrt(n) là hàm tính √, trunc(x) là hàm làm tròn x }

Như vậy ñể kiểm tra một số khoảng 25 chữ số mất khoảng

‑ ­ !"

­ ­# ~0.03 giây!

2.3. ðánh giá thời gian thực hiện thuật toán

Có hai cách tiếp cận ñể ñánh giá thời gian thực hiện của một thuật toán. Cách thứ

nhất bằng thực nghiệm, chúng ta viết chương trình và cho chạy chương trình với

các dữ liệu vào khác nhau trên một máy tính. Cách thứ hai bằng phương pháp lí

thuyết, chúng ta coi thời gian thực hiện thuật toán như hàm số của cỡ dữ liệu vào

(cỡ của dữ liệu vào là một tham số ñặc trưng cho dữ liệu vào, nó có ảnh hưởng

quyết ñịnh ñến thời gian thực hiện chương trình. Ví dụ ñối với bài toán kiểm tra

số nguyên tố thì cỡ của dữ liệu vào là số cần kiểm tra; hay với bài toán sắp xếp

dãy số, cỡ của dữ liệu vào là số phần tử của dãy). Thông thường cỡ của dữ liệu

vào là một số nguyên dương , ta sử dụng hàm số % trong ñó là cỡ của dữ

liệu vào ñể biểu diễn thời thực hiện của một thuật toán.

Xét ví dụ bài toán kiểm tra tính nguyên tố của một số nguyên dương (cỡ dữ liệu

vào là ), nếu là một số chẵn & 2 thì chỉ cần một lần thử chia 2 ñể kết luận

không phải là số nguyên tố. Nếu & 3 không chia hết cho 2 nhưng lại chia

hết cho 3 thì cần 2 lần thử (chia 2 và chia 3) ñể kết luận không nguyên tố. Còn

nếu là một số nguyên tố thì thuật toán phải thực hiện nhiều lần thử nhất.

8

Trong tài liệu này, chúng ta hiểu hàm số % là thời gian nhiều nhất cần thiết ñể

thực hiện thuật toán với mọi bộ dữ liệu ñầu vào cỡ .

Sử dụng kí hiệu toán học ô lớn ñể mô tả ñộ lớn của hàm %. Giả sử là một số

nguyên dương, % và ' là hai hàm thực không âm. Ta viết % ( )'

nếu và chỉ nếu tồn tại các hằng số dương \* và

, sao cho % + \* ', với

mọi

.

Nếu một thuật toán có thời gian thực hiện % ( )' chúng ta nói rằng

thuật toán có thời gian thực hiện cấp '.

Ví dụ: Giả sử % ( \_x0016\_ , 2, ta có

\_x0016\_ , 2 + \_x0016\_ , 2\_x0016\_ ( 3\_x0016\_

với mọi 1

Vậy % ( )\_x0016\_

, trong trường hợp này ta nói thuật toán có thời gian thực hiện

cấp

\_x0016\_

.

2.4. Các quy tắc ñánh giá thời gian thực hiện thuật toán

ðể ñánh giá thời gian thực hiện thuật toán ñược trình bày bằng ngôn ngữ tựa

Pascal, ta cần biết cách ñánh giá thời gian thực hiện các câu lệnh của Pascal.

Trước tiên, chúng ta hãy xem xét các câu lệnh chính trong Pascal. Các câu lệnh

trong Pascal ñược ñịnh nghĩa ñệ quy như sau:

1. Các phép gán, ñọc, viết là các câu lệnh (ñược gọi là lệnh ñơn).

2. Nếu S1, S2, ..., Sm là câu lệnh thì

Begin S1; S2; …; Sm; End;

là câu lệnh (ñược gọi là lệnh hợp thành hay khối lệnh).

3. Nếu S1 và S2 là các câu lệnh và E là biểu thức lôgic thì

If E then S1 else S2;

là câu lệnh (ñược gọi là lệnh rẽ nhánh hay lệnh If).

4. Nếu S là câu lệnh và E là biểu thức lôgic thì

While E do S;

là câu lệnh (ñược gọi là lệnh lặp ñiều kiện trước hay lệnh While).

5. Nếu S1, S2,…,Sm là các câu lệnh và E là biểu thức lôgic thì

Repeat

S1; S2; …; Sm;

Until E;

là câu lệnh (ñược gọi là lệnh lặp ñiều kiện sau hay lệnh Repeat)

9

6. Nếu S là lệnh, E1 và E2 là các biểu thức cùng một kiểu thứ tự ñếm ñược

thì

For i:=E1 to E2 do S;

là câu lệnh (ñược gọi là lệnh lặp với số lần xác ñịnh hay lệnh For).

ðể ñánh giá, chúng ta phân tích chương trình xuất phát từ các lệnh ñơn, rồi ñánh

giá các lệnh phức tạp hơn, cuối cùng ñánh giá ñược thời gian thực hiện của

chương trình, cụ thể:

1. Thời gian thực hiện các lệnh ñơn: gán, ñọc, viết là )1

2. Lệnh hợp thành: giả sử thời gian thực hiện của S1, S2,…,Sm tương ứng là

)'

, )'\_x0016\_, . . . , )'.. Khi ñó thời gian thực hiện của lệnh hợp

thành là: )/0'

, '\_x0016\_, … , '..

3. Lệnh If: giả sử thời gian thực hiện của S1, S2 tương ứng là

)'

, )'\_x0016\_. Khi ñó thời gian thực hiện của lệnh If là:

)/0'

, '\_x0016\_.

4. Lệnh lặp While: giả sử thời gian thực hiện lệnh S (thân của lệnh While) là

)' và 2 là số lần lặp tối ña thực hiện lệnh S. Khi ñó thời gian thực

hiện lệnh While là )'2.

5. Lệnh lặp Repeat: giả sử thời gian thực hiện khối lệnh

Begin S1; S2;…; Sm; End;

là )' và 2 là số lần lặp tối ña. Khi ñó thời gian thực hiện lệnh

Repeat là )'2.

6. Lệnh lặp For: giả sử thời gian thực hiện lệnh S là )' và 2 là số

lần lặp tối ña. Khi ñó thời gian thực hiện lệnh For là )'2.

2.5. Một số ví dụ

Ví dụ 1: Phân tích thời gian thực hiện của chương trình sau:

var i, j, n :longint;

s1, s2 :longint;

BEGIN

{1} readln(n);

{2} s1:=0;

{3} for i:=1 to n do

{4} s1:=s1 + i;

{5} s2:=0;

10

{6} for j:=1 to n do

{7} s2:=s2 + j\*j;

{8} writeln('1+2+..+',n,'=',s1);

{9} writeln('1^2+2^2+..+',n,'^2=',s2);

END.

Thời gian thực hiện chương trình phụ thuộc vào số 3.

Các lệnh {1}, {2}, {4}, {5}, {7}, {8}, {9} có thời gian thực hiện là )1.

Lệnh lặp For {3} có số lần lặp là , như vậy lệnh {3} có thời gian thực hiện là

). Tương tự lệnh lặp For {6} cũng có thời gian thực hiện là ).

Vậy thời gian thực hiện của chương trình là:

max)1, )1, ), )1, ), )1, )1 ( )

Ví dụ 2: Phân tích thời gian thực hiện của ñoạn chương trình sau:

{1} c:=0;

{2} for i:=1 to 2\*n do

{3} c:=c+1;

{4} for i:=1 to n do

{5} for j:=1 to n do

{6} c:=c+1;

Thời gian thực hiện chương trình phụ thuộc vào số .

Các lệnh {1}, {3}, {6} có thời gian thực hiện là )1.

Lệnh lặp For {2} có số lần lặp là 2, như vậy lệnh {2} có thời gian thực hiện là

).

Lệnh lặp For {5} có số lần lặp là , như vậy lệnh {5} có thời gian thực hiện là

). Lệnh lặp For {4} có số lần lặp là , như vậy lệnh {4} có thời gian thực hiện

là )\_x0016\_

.

Vậy thời gian thực hiện của ñoạn chương trình trên là:

max)1, ), )\_x0016\_

( )\_x0016\_

Ví dụ 3: Phân tích thời gian thực hiện của ñoạn chương trình sau:

{1} for i:=1 to n do

{2} for j:=1 to i do

{3} c:=c+1;

Thời gian thực hiện chương trình phụ thuộc vào số .

Các lệnh {3} có thời gian thực hiện là )1.

11

Khi i = 1, j chạy từ 1 ñến 1 lệnh lặp For {2} lặp 1 lần

Khi i = 2, j chạy từ 1 ñến 2 lệnh lặp For {2} lặp 2 lần

…

Khi i = , j chạy từ 1 ñến lệnh lặp For {2} lặp lần

Như vậy lệnh {3} ñược lặp: 1 , 2,. . , ( 778

\_x0016\_

lần, do ñó lệnh {1} có thời

gian thực hiện là )\_x0016\_

Vậy thời gian thực hiện của ñoạn chương trình trên là: )\_x0016\_

Bài tập

1.1. Phân tích thời gian thực hiện của ñoạn chương trình sau:

for i:=1 to n do

if i mod 2=0 then c:=c+1;

1.2. Phân tích thời gian thực hiện của ñoạn chương trình sau:

for i:=1 to n do

if i mod 2=0 then c1:=c1+1

else c2:=c2+1;

1.3. Phân tích thời gian thực hiện của ñoạn chương trình sau:

for i:=1 to n do

if i mod 2=0 then

for j:=1 to n do c:=c+1

1.4. Phân tích thời gian thực hiện của ñoạn chương trình sau:

a:=0;

b:=0;

c:=0;

for i:=1 to n do

begin

a:=a + 1;

b:=b + i;

c:=c + i\*i;

end;

1.5. Phân tích thời gian thực hiện của ñoạn chương trình sau:

i:=n;

d:=0;

12

while i>0 do

begin

i:=i-1;

d:=d + i;

end;

1.6. Phân tích thời gian thực hiện của ñoạn chương trình sau:

i:=0;

d:=0;

repeat

i:=i+1;

if i mod 3=0 then d:=d + i;

until i>n;

1.7. Phân tích thời gian thực hiện của ñoạn chương trình sau:

d:=0;

for i:=1 to n-1 do

for j:=i+1 to n do d:=d+1;

1.8. Phân tích thời gian thực hiện của ñoạn chương trình sau:

d:=0;

for i:=1 to n-2 do

for j:=i+1 to n-1 do

for k:=j+1 to n do d:=d+1;

1.9. Phân tích thời gian thực hiện của ñoạn chương trình sau:

d:=0;

while n>0 do

begin

n:=n div 2;

d:=d+1;

end;

1.10. Cho một dãy số gồm số nguyên dương, xác ñịnh xem có tồn tại một dãy

con liên tiếp có tổng bằng hay không?

a) ðưa ra thuật toán có thời gian thực hiện )

.

b) ðưa ra thuật toán có thời gian thực hiện )

\_x0016\_

.

c) ðưa ra thuật toán có thời gian thực hiện ).

13

Chuyên ñề 2

CÁC KIẾN THỨC CƠ BẢN

1. Hệ ñếm

Hệ ñếm ñược hiểu là tập các kí hiệu và quy tắc sử dụng tập các kí hiệu ñó ñể biểu

diễn và xác ñịnh giá trị các số. Trong hệ ñếm cơ số 9 9 & 1, các kí hiệu ñược

dùng có các giá trị tương ứng 0, 1, . . , 9 \_x0007\_ 1. Giả sử : có biểu diễn:

\_x0007\_1

\_x0007\_2 …

1

0

,

\_x0007\_1

\_x0007\_2 …

\_x0007\_

trong ñó , 1 số các chữ số bên trái, là số các chữ số bên phải dấu phân chia

phần nguyên và phần phân của số : và các

;

phải thoả mãn ñiều kiện

0 +

< = 9 \_x0007\_ + ; + .

Khi ñó giá trị của số : ñược tính theo công thức:

: (

79

7

,

7>

9

7>

,. . . ,

9

,

>

9

>

, . . . ,

>.9

>. 1

Chú ý: ðể phân biệt số ñược biểu diễn ở hệ ñếm nào người ta viết cơ số làm chỉ

số dưới của số ñó. Ví dụ: :? là biểu diễn : ở hệ ñếm 9.

1.1. Các hệ ñếm thường dùng:

Hệ thập phân (hệ cơ số 10) dùng 10 kí hiệu 0, 1, 2, 3, 4, 5, 6, 7, 8, 9

Ví dụ: 28,910 = 2 × 101

+ 8 × 100

+ 9 × 10-1

Hệ nhị phân (hệ cơ số 2) chỉ dùng hai kí hiệu 0, 1

Ví dụ: 102= 1 × 21

+ 0 × 20 = 210

101,12= 1 × 22

+ 0 × 21 + 1 × 20

+ 1 × 2-1 =5,5

Hệ cơ số mười sáu, còn gọi là hệ hexa, sử dụng các kí hiệu 0, 1, 2, 3, 4, 5, 6, 7, 8,

9, A, B, C, D, E, F, trong ñó A, B, C, D, E, F có các giá trị tương ứng 10, 11, 12,

13, 14, 15 trong hệ thập phân

Ví dụ: AF016 = 10 × 162

+ 15 × 161 + 0 × 160

=280010

14

1.2. Chuyển ñổi biểu diễn số ở hệ thập phân sang hệ ñếm cơ số khác

ðể chuyển ñổi biểu diễn một số ở hệ thập phân sang hệ ñếm cơ số khác, trước hết

ta tách phần nguyên và phần phân rồi tiến hành chuyển ñổi từng phần, sau ñó

ghép lại.

Chuyển ñổi biểu diễn phần nguyên: Từ (1) ta lấy phần nguyên:

@ (

79

7

,

7>

9

7>

,. . . ,

AB 2 đó 0 +

< = 9.

Do 0 +

= 9 nên khi chia @ cho 9 thì phần dư của phép chia ñó là

0

còn

thương số @1 sẽ là:

9

\_x0007\_1 ,

\_x0007\_19

\_x0007\_2 ,. . . ,

1

. Tương tự

1

là phần dư của

phép chia @1 cho 9. Quá trình ñược lặp cho ñến khi nhận ñược thương bằng 0.

Chuyển ñổi biểu diễn phần phân: Từ (1) ta lấy phần sau dấu phẩy:

E (

>

9

>

, . . . ,

>.9

>..

E1 ( E 9 (

>

,

>\_x0016\_9

>

, . . . ,

>.9

>.>

Ta nhận thấy

\_x0007\_1 chính là phân nguyên của kết quả phép nhân, còn phần phân của

kết quả là E2 (

>\_x0016\_9

>

, . . . ,

>.9

>.>

. Quá trình ñược lặp cho ñến khi

nhận ñủ số chữ số cần tìm.

2. Số nguyên tố

Một số tự nhiên F F & 1 là số nguyên tố nếu F có ñúng hai ước số là 1 và F.

Ví dụ các số nguyên tố: 2, 3, 5, 7, 11, 13, 17, 19, 23, …

2.1. Kiểm tra tính nguyên tố

a) ðể kiểm tra số nguyên dương & 1 có là số nguyên tố không, ta kiểm tra

xem có tồn tại một số nguyên 2 + + \_x0007\_ 1) mà là ước của ( chia hết

) thì không phải là số nguyên tố, ngược lại là số nguyên tố.

Nếu & 1 không phải là số nguyên tố, ta luôn có thể tách (

\_x0016\_ à 2 +

+ \_x0016\_ + \_x0007\_ 1. Vì

+

\_x0016\_ ( nên

+ √. Do ñó,

việc kiểm tra với từ 2 ñến \_x0007\_ 1 là không cần thiết, mà chỉ cần kiểm tra từ 2

ñến √.

function is\_prime(n:longint):boolean;

var k :longint;

begin

if n=1 then exit(false);

15

for k:=2 to trunc(sqrt(n)) do

if (n mod k=0) then exit(false);

exit(true);

end;

Hàm is\_prime(n) trên tiến hành kiểm tra lần lượt từng số nguyên trong ñoạn

[2, √], ñể cải tiến, cần giảm thiểu số các số cần kiểm tra. Ta có nhận xét, ñể kiểm

tra số nguyên dương & 1 có là số nguyên tố không, ta kiểm tra xem có tồn

tại một số nguyên tố 2 + + √) mà là ước của thì không phải là số

nguyên tố, ngược lại là số nguyên tố. Thay vì kiểm tra các số là nguyên tố ta

sẽ chỉ kiểm tra các số có tính chất giống với tính chất của số nguyên tố, có thể

sử dụng một trong hai tính chất ñơn giản sau của số nguyên tố:

1) Trừ số 2 và các số nguyên tố là số lẻ.

2) Trừ số 2, số 3 các số nguyên tố có dạng 6 K 1 (vì số có dạng 6 K 2 thì

chia hết cho 2, số có dạng 6 K 3 thì chia hết cho 3).

Hàm is\_prime2(n) dưới ñây kiểm tra tính nguyên tố của số bằng cách kiểm

tra xem có chia hết cho số 2, số 3 và các số có dạng 6 K 1 trong ñoạn [5, √ ].

function is\_prime2(n:longint):boolean;

var k,sqrt\_n:longint;

begin

if (n=2)or(n=3) then exit(true);

if (n=1)or(n mod 2=0)or(n mod 3=0) then exit(false);

sqrt\_n:=trunc(sqrt(n));

k:=-1;

repeat

inc(k,6);

if (n mod k=0)or(n mod (k+2)=0) then break;

until k>sqrt\_n;

exit(k>sqrt\_n);

end;

b) Phương pháp kiểm tra số nguyên tố theo xác suất

Từ ñịnh lí nhỏ Fermat:

nếu F là số nguyên tố và / là số tự nhiên thì /

L

F ( /

Ta có cách kiểm tra tính nguyên tố của Fermat:

16

nếu 2

7

M 2 thì không là số nguyên tố

nếu 2

7

( 2 thì nhiều khả năng là số nguyên tố

Ví dụ:

2

N

9 ( 512

9 ( 8 M 2, do ñó số 9 không là số nguyên tố.

2

3 ( 8

3 ( 2, do ñó nhiều khả năng 3 là số nguyên tố, thực tế 3 là số

nguyên tố.

2

11 ( 2048

11 ( 2, do ñó nhiều khả năng 11 là số nguyên tố, thực

tế 11 là số nguyên tố.

2.2. Liệt kê các số nguyên tố trong ñoạn Q­, RS

Cách thứ nhất là thử lần lượt các số trong ñoạn Q1, :S, rồi kiểm tra tính nguyên

tố của .

procedure generate(N:longint);

var m :longint;

begin

for m:=2 to N do

if is\_prime(m) then writeln(m);

end;

Cách này ñơn giản nhưng chạy chậm, ñể cải tiến có thể sử dụng các tính chất của

số nguyên tố ñể loại bỏ trước những số không phải là số nguyên tố và không cần

kiểm tra các số này.

Cách thứ hai là sử dụng sàng số nguyên tố, như sàng Eratosthene, liệt kê ñược các

số nguyên tố nhanh, tuy nhiên nhược ñiểm của cách này là tốn nhiều bộ nhớ. Cách

làm ñược thực hiện như sau:

Trước tiên xoá bỏ số 1 ra khỏi tập các số nguyên tố. Số tiếp theo số 1 là số 2, là số

nguyên tố, xoá tất cả các bội của 2 ra khỏi bảng. Số ñầu tiên không bị xoá sau số 2

(số 3) là số nguyên tố, xoá các bội của 3... Giải thuật tiếp tục cho ñến khi gặp số

nguyên tố lớn hơn √: thì dừng lại. Tất cả các số chưa bị xoá là số nguyên tố.

{$M 1100000}

procedure Eratosthene(N:longint);

const MAX = 1000000;

var i,j :longint;

Prime :array [1..MAX] of byte;

begin

17

fillchar(Prime,sizeof(Prime),0);

for i:=2 to trunc(sqrt(N)) do

if Prime[i]=0 then

begin

j:=i\*i;

while j<=N do

begin

Prime[j]:=1;

j:=j+i;

end;

end;

for i:=2 to N do

if Prime[i]=0 then writeln(i);

end;

3. Ước số, bội số

3.1. Số các ước số của một số

Giả sử : ñược phân tích thành thừa số nguyên tố như sau:

: ( /< 9T … \*U

Ước số của N có dạng: /

L 9V … \*W

trong ñó

0+ F + ;, 0 + X + Y, … , 0 + B + .

Do ñó, số các ước số của : là ; , 1 Y , 1 … , 1.

Ví dụ:

: ( 100 ( 2\_x0016\_ 5\_x0016\_

, số ước số của 100 là: 2 , 12 , 1 ( 9 ước số (các ước

số ñó là: 1, 2, 4, 5, 10, 20, 25, 50, 100).

: ( 24 ( 2 3, số ước số của 24 là: 3 , 11 , 1 ( 8 ước số (các ước số

ñó là: 1, 2, 3, 4, 6, 8, 12, 24).

3.2. Tổng các ước số của một số

: ( /< 9T … \*U

ðặt :1 ( 9T … \*U

Gọi ZA là tổng các ước của A, ta có,

Z: ( Z:1 , / Z:1 , [ , /< Z:1

18

( \1 , / , [ , /<

] Z:1 (

^\_`\_x0012\_>

^>

Z:1

(

/<8

\_x0007\_ 1

/ \_x0007\_ 1

9T8

\_x0007\_ 1

9 \_x0007\_ 1 …

\*U8

\_x0007\_ 1

\* \_x0007\_ 1

Ví dụ: Tổng các ước của 24 là:

28

\_x0007\_ 1

2 \_x0007\_ 1

3

8

\_x0007\_ 1

3 \_x0007\_ 1 ( 60

3.3. Ước số chung lớn nhất của hai số

Ước số chung lớn nhất (USCLN) của 2 số ñược tính theo thuật toán Euclid

abcd:/, 9 ( abcd:9, /

9

function USCLN(a,b:longint):longint;

var tmp :longint;

begin

while b>0 do begin

a:=a mod b;

tmp:=a; a:=b; b:=tmp;

end;

exit(a);

end;

3.4. Bội số chung nhỏ nhất của hai số

Bội số chung nhỏ nhất (BSCNN) của hai số ñược tính theo công thức:

ebc::/, 9 (

/ 9

abcd:/, 9 (

/

abcd:/, 9 9

4. Lí thuyết tập hợp

4.1. Các phép toán trên tập hợp

1. Phần bù của f trong @, kí hiệu fg , là tập hợp các phần tử của @ không

thuộc f:

fg( h0 i @: 0 k fl

2. Hợp của f và e, kí hiệu f m e, là tập hợp các phần tử hoặc thuộc vào f

hoặc thuộc vào e:

19

f m e ( h0: 0 i f n ặ\* 0 i el

3. Giao của f và e, kí hiệu f o e, là tập hợp các phần tử ñồng thời thuộc cả

f và e

f o e ( h0: 0 i f pà 0 i el

4. Hiệu của f và e, kí hiệu là f\e, là tập hợp các phần tử thuộc tập f

nhưng không thuộc e.

f\e ( h0: 0 i f pà 0 k el

4.2. Các tính chất của phép toán trên tập hợp

1. Kết hợp

f m e m c ( f m e m c

f o e o c ( f o e o c

2. Giao hoán

f m e ( e m f

f o e ( e o f

3. Phân bố

f m e o c ( f m e o f m c

f o e m c ( f o e m f o c

4. ðối ngẫu

f m e rrrrrrr ( fgo er

f o e rrrrrrr ( fgm er

4.3. Tích ðề-các của các tập hợp

Tích ðề-các ghép hai tập hợp:

f e ( h/, 9|/ i f, 9 i el

Tích ðề-các mở rộng ghép nhiều tập hợp:

f

f\_x0016\_ … fU ( h/

, /\_x0016\_, … , /U

|/< i f<

, ; ( 1, 2, . . , l

4.4. Nguyên lí cộng

Nếu f và e là hai tập hợp rời nhau thì

|f m e| ( |f| , |e|

Nguyên lí cộng mở rộng cho nhiều tập hợp ñôi một rời nhau:

20

Nếu hf

, f\_x0016\_, … , fU

l là một phân hoạch của tập @ thì:

|@| ( |f

| , |f\_x0016\_

| , [ , |fU|

4.5. Nguyên bù trừ

Nếu f và e không rời nhau thì

|f m e| ( |f| , |e| \_x0007\_ |f o e|

Nguyên lí mở rộng cho nhiều tập hợp:

Giả sử f

, f\_x0016\_, … , f. là các tập hữu hạn:

|f

m f\_x0016\_ m … m f.| ( :

\_x0007\_ :\_x0016\_ , [ , \_x0007\_1.>

:.

trong ñó :U là tổng phần tử của tất cả các giao của tập lấy từ tập ñã cho

4.6. Nguyên lí nhân

Nếu mỗi thành phần /<

của bộ có thứ tự k thành phần /

, /\_x0016\_, … , /U có <

khả

năng lựa chọn ; ( 1, 2, … , , thì số bộ sẽ ñược tạo ra là tích số của các khả năng

này

\_x0016\_ . . U

Một hệ quả trực tiếp của nguyên lí nhân:

|f

f\_x0016\_ … fU| ( |f

| |f\_x0016\_| … |fU|

4.7. Chỉnh hợp lặp

Xét tập hữu hạn gồm phần tử f ( h/

, /\_x0016\_, … , /7

l

Một chỉnh hợp lặp chập của phần tử là một bộ có thứ tự gồm phần tử của f,

các phần tử có thể lặp lại. Một chỉnh hợp lặp chập của có thể xem như một

phần tử của tích ðềcac f

U

. Theo nguyên lí nhân, số tất cả các chỉnh hợp lặp chập

của sẽ là

U

.

fg

7

U ( U

4.8. Chỉnh hợp không lặp

Một chỉnh hợp không lặp chập của phần tử + là một bộ có thứ tự gồm

thành phần lấy từ phần tử của tập ñã cho. Các thành phần không ñược lặp lại.

ðể xây dựng một chỉnh hợp không lặp, ta xây dựng dần từng thành phần ñầu tiên.

Thành phần này có khả năng lựa chọn. Mỗi thành phần tiếp theo, số khả năng

21

lựa chọn giảm ñi 1 so với thành phần ñứng trước, do ñó, theo nguyên lí nhân, số

chỉnh hợp không lặp chập của sẽ là \_x0007\_ 1 … \_x0007\_ , 1.

f7

U ( \_x0007\_ 1 … \_x0007\_ , 1 (

!

\_x0007\_ !

4.9. Hoán vị

Một hoán vị của phần tử là một cách xếp thứ tự các phần tử ñó. Một hoán vị của

phần tử ñược xem như một trường hợp riêng của chỉnh hợp không lặp khi (

. Do ñó số hoán vị của phần tử là !

4.10. Tổ hợp

Một tổ hợp chập của phần tử + là một bộ không kể thứ tự gồm thành

phần khác nhau lấy từ phần tử của tập ñã cho.

c7

U (

\_x0007\_ 1 … \_x0007\_ , 1

! (

!

! \_x0007\_ !

Một số tính chất

- c7

U ( c7

7>U

- c7

( c7

7 ( 1

- c7

U ( c7>

U>

, c7>

U

( với 0 = = )

5. Số Fibonacci

Số Fibonacci ñược xác ñịnh bởi công thức sau:

u

Z

( 0

Z

( 1

Z7 ( Z7>

, Z7>\_x0016\_ pớ; 2

Một số phần tử ñầu tiên của dãy số Fibonacci:

0 1 2 3 4 5 6 …

wxyz3{||x3 0 1 1 2 3 5 8 …

Số Fibonacci là ñáp án của các bài toán:

a) Bài toán cổ về việc sinh sản của các cặp thỏ như sau:

- Các con thỏ không bao giờ chết;

22

- Hai tháng sau khi ra ñời, mỗi cặp thỏ mới sẽ sinh ra một cặp thỏ con (một ñực,

một cái);

- Khi ñã sinh con rồi thì cứ mỗi tháng tiếp theo chúng lại sinh ñược một cặp con

mới.

Giả sử từ ñầu tháng 1 có một cặp mới ra ñời thì ñến giữa tháng thứ n sẽ có bao

nhiêu cặp.

Ví dụ, n = 5, ta thấy:

Giữa tháng thứ 1:

1 cặp (cặp ban ñầu)

Giữa tháng thứ 2:

1 cặp cặp (ban ñầu vẫn chưa ñẻ)

Giữa tháng thứ 3:

2 cặp (cặp ban ñầu ñẻ ra thêm 1

cặp con)

Giữa tháng thứ 4:

3 cặp (cặp ban ñầu tiếp tục ñẻ)

Giữa tháng thứ 5: 5 cặp.

b) ðếm số cách xếp \_x0007\_ 1 thanh DOMINO có kích thước 2×1 phủ kín bảng có

kích thước 2 \_x0007\_ 1.

Ví dụ: Có tất cả 8 cách khác nhau ñể xếp các thanh DOMINO có kích thước 2x1

phủ kín bảng 2x5 ( 6, Z;9 /\*\*; ( 8.

Hàm tính số Fibonacci thứ bằng phương pháp lặp sử dụng công thức

Z7 ( Z7>

, Z7>\_x0016\_ với 2 và Z

( 0, Z

( 1.

function Fibo(n : longint):longint;

var fi\_1, fi\_2, fi, i :longint;

begin

23

if n<=1 then exit(n);

fi\_2:=0; fi\_1:=1;

for i:=2 to n do begin

fi:=fi\_1 + fi\_2;

fi\_2:=fi\_1;

fi\_1:=fi;

end;

exit(fi);

end;

Công thức tổng quát Z7 (

√

}~

8√

\_x0016\_



7

\_x0007\_ ~

>√

\_x0016\_



7



6. Số Catalan

Số Catalan ñược xác ñịnh bởi công thức sau:

c/A//7 (

1

, 1 c\_x0016\_7

7 (

2!

, 1! !

pớ; 0

Một số phần tử ñầu tiên của dãy số Catalan là:

0 1 2 3 4 5 6 …

{{{33 ­ ­ ! " ­# #! ­! …

Số Catalan là ñáp án của các bài toán:

1) Có bao nhiêu cách khác nhau ñặt dấu ngoặc mở và dấu ngoặc ñóng

ñúng ñắn?

Ví dụ: ( 3 ta có 5 cách sau:

~\ ] , \ ], \ ] , \ ],

2) Có bao nhiêu cây nhị phân khác nhau có ñúng , 1 lá?

Ví dụ: ( 3

24

3) Cho một ña giác lồi , 2 ñỉnh, ta chia ña giác thành các tam giác bằng cách

vẽ các ñường chéo không cắt nhau trong ña giác. Hỏi có bao nhiêu cách chia như

vậy?

Ví dụ: ( 4

7. Xử lí số nguyên lớn

Nhiều ngôn ngữ lập trình cung cấp kiểu dữ liệu nguyên khá lớn, chẳng hạn trong

Free Pascal có kiểu số 64 bit (khoảng 19 chữ số). Tuy nhiên ñể thực hiện các phép

tính với số nguyên ngoài phạm vi biểu diễn ñược cung cấp (có hàng trăm chữ số

chẳng hạn), chúng ta cần tự thiết kế cách biểu diễn và các hàm thực hiện các phép

toán cơ bản với các số nguyên lớn.

7.1. Biểu diễn số nguyên lớn

Thông thường người ta sử dụng các cách biểu diễn số nguyên lớn sau:

• Xâu kí tự: ðây là cách biểu diễn tự nhiên và ñơn giản nhất, mỗi kí tự của

xâu tương ứng với một chữ số của số nguyên lớn tính từ trái qua phải.

• Mảng các số: Sử dụng mảng lưu các chữ số (hoặc một nhóm chữ số), và

một biến ghi nhận số chữ số ñể thuận tiện trong quá trình xử lí.

• Danh sách liên kết các số: Sử dụng danh sách liên kết các chữ số (hoặc

một nhóm chữ số), cách làm này sẽ linh hoạt hơn trong việc sử dụng bộ

nhớ.

Trong phần này, sử dụng cách biểu diễn thứ nhất, biểu diễn số nguyên lớn bằng

xâu kí tự và chỉ xét các số nguyên lớn không âm.

Type bigNum = string;

25

7.2. Phép so sánh

ðể so sánh hai số nguyên lớn a, b ñược biểu diễn bằng xâu kí tự, trước

tiên ta thêm các chữ số 0 vào ñầu số có số chữ số nhỏ hơn ñể hai số có số

lượng chữ số bằng nhau. Sau ñó sử dụng trực tiếp phép toán so sánh trên

xâu kí tự.

Hàm cmp so sánh hai số nguyên lớn a, b. Giá trị hàm trả về

u

0 ế / ( 9

1 ế / & 9

\_x0007\_1 ế / = 9

function cmp(a,b : bigNum): integer;

begin

while length(a)<length(b) do a:='0'+a;

while length(b)<length(a) do b:='0'+b;

if a = b then exit(0);

if a > b then exit(1);

exit(-1);

end;

7.3. Phép cộng

Phép cộng hai số nguyên ñược thực hiện từ phải qua trái và phần nhớ ñược mang

sang trái.

function add(a,b : bigNum): bigNum;

var sum, carry, i, x, y : integer;

c : bigNum;

begin

carry:=0;c:='';

while length(a)<length(b) do a:='0'+a;

while length(b)<length(a) do b:='0'+b;

for i:=length(a) downto 1 do

begin

x:= ord(a[i])-ord('0'); {ord('0')=48}

y:= ord(b[i])-ord('0');

sum:=x + y + carry;

carry:=sum div 10;

26

c:=chr(sum mod 10 +48)+c;

end;

if carry>0 then c:='1'+c;

add:=c;

end;

7.4. Phép trừ

Thực hiện phép trừ ngược lại với việc nhớ ở phép cộng ta phải chú ý ñến việc vay

mượn từ hàng cao hơn. Trong hàm trừ dưới ñây, chỉ xét trường hợp số lớn trừ số

nhỏ hơn.

function sub(a,b:bigNum):bigNum;

var c :bigNum;

s,borrow,i :integer;

begin

borrow:=0;c:='';

while length(a)<length(b) do a:='0'+a;

while length(b)<length(a) do b:='0'+b;

for i:=length(a) downto 1 do

begin

s:=ord(a[i])-ord(b[i])-borrow;

if s<0 then

begin

s:=s+10;

borrow:=1;

end else borrow:=0;

c:=chr(s +48)+c;

end;

while (length(c)>1)and(c[1]='0') do delete(c,1,1);

sub:=c;

end;

7.5. Phép nhân một số lớn với một số nhỏ

Số nhỏ ở ñây ñược hiểu là số nguyên do ngôn ngữ lập trình cung cấp (như:

longint, integer,..). Hàm multiply1(a:bigNum;b:longint):bigNum, trả về

là một số nguyên lớn (bigNum) là kết quả của phép nhân một số nguyên lớn a

(bigNum) với một số b (longint).

27

function multiply1(a:bigNum;b:longint):bigNum;

var i :integer;

carry,s :longint;

c,tmp :bigNum;

begin

c:='';

carry:=0;

for i:=length(a) downto 1 do

begin

s:=(ord(a[i])-48) \* b + carry;

carry:= s div 10;

c:=chr(s mod 10 + 48)+c;

end;

if carry>0 then str(carry,tmp) else tmp:='';

multiply1:=tmp+c;

end;

7.6. Phép nhân hai số nguyên lớn

function multiply2(a,b:bigNum):bigNum;

var sum,tmp :bigNum;

m,i,j :integer;

begin

m:=-1;sum:='';

for i:=length(a) downto 1 do

begin

m:=m+1;

tmp:=multiply1(b,ord(a[i])-48);

{có thể thay câu lệnh tmp:=multiply1(b,ord(a[i])-48);

bằng cách cộng nhiều lần như sau:

tmp:='';

for j:=1 to ord(a[i])-48 do tmp:=add(tmp,b);

như vậy hàm nhân multiply2 chỉ gọi hàm cộng hai số nguyên lớn add}

for j:=1 to m do tmp:=tmp+'0';

sum:=add(tmp,sum);

end;

multiply2:=sum;

end;

28

7.7. Phép toán chia lấy thương nguyên (div)

của một số lớn với một số nhỏ

function bigDiv1(a:bigNum;b:longint):bigNum;

var s,i,hold:longint;

c:bigNum;

begin

hold:=0;s:=0; c:='';

for i:=1 to length(a) do

begin

hold:=hold\*10 + ord(a[i])-48;

s:=hold div b;

hold:=hold mod b;

c:=c+chr(s+48);

end;

while (length(c)>1) and(c[1]='0') do

delete(c,1,1);

bigDiv1:=c;

end;

7.8. Phép toán chia lấy dư (mod) của một số lớn với một số nhỏ

function bigMod1(a:bigNum;b:longint):longint;

var i,hold:longint;

begin

hold:=0;

for i:=1 to length(a) do

hold:=(ord(a[i])-48+hold\*10) mod b;

bigMod1:=hold;

end;

Chú ý: Ta có các công thức sau:

1 A , B mod N ( A mod N , B mod N mod N

2 A Bmod N ( \A mod N B mod N]mod N

7.9. Phép toán chia lấy thương nguyên (div) của hai số lớn

function bigDiv2(a,b:bigNum):bigNum;

var c,hold :bigNum;

29

kb :array[0..10]of bigNum;

i,k :longint;

begin

kb[0]:='0';

for i:=1 to 10 do

kb[i]:=add(kb[i-1],b);

hold:='';

c:='';

for i:=1 to length(a) do

begin

hold:=hold+a[i];

k:=1;

while cmp(hold,kb[k])<>-1 do

inc(k);

c:=c+chr(k-1+48);

hold:=sub(hold,kb[k-1]);

end;

while (length(c)>1)and(c[1]='0') do delete(c,1,1);

bigDiv2:=c;

end;

7.10. Phép toán chia lấy dư (mod) của hai số lớn

function bigMod2(a,b:bigNum):bigNum;

var hold :bigNum;

kb :array[0..10]of bigNum;

i,k :longint;

begin

kb[0]:='0';

for i:=1 to 10 do

kb[i]:=add(kb[i-1],b);

hold:='';

for i:=1 to length(a) do

begin

hold:=hold+a[i];

k:=1;

while cmp(hold,kb[k])<>-1 do

30

inc(k);

hold:=sub(hold,kb[k-1]);

end;

bigMod2:=hold;

end;

7.11. Ví dụ tính số Fibonacci thứ 3 3 + "

Số Fibonacci ñược xác ñịnh bởi công thức sau:

u

Z

( 0

Z

( 1

Z7 ( Z7>

, Z7>\_x0016\_ pớ; 2

Trước tiên ta xây dựng chương trình tính số Fibonacci bằng kiểu dữ liệu Extended

như sau:

function Fibo(n : longint):extended;

var i :longint;

fi\_1, fi\_2, fi : extended;

begin

if n<=1 then exit(n);

fi\_2:=0; fi\_1:=1;

for i:=2 to n do begin

fi:=fi\_1 + fi\_2;

fi\_2:=fi\_1;

fi\_1:=fi;

end;

exit(fi);

end;

var n : longint;

BEGIN

write('Nhap N:'); readln(n);

writeln(Fibo(n));

END.

Chạy chương trình với ( 500 ta nhận ñược kết quả:

1.3942322456169788E+0104, như vậy số Fibonacci thứ 500 có 105 chữ số (có

thể sử dụng cách biểu diễn bằng xâu kí tự), ta xây dựng chương trình tính số

Fibonacci lớn bằng cách sau:

31

- Thay kiểu extended bằng kiểu bigNum.

- Thay các phép toán bằng các hàm tính toán số lớn, xây dựng các hàm tính

toán số lớn cần thiết.

type bigNum = string;

function add(a,b : bigNum): bigNum;

var sum, carry, i : integer;

c : bigNum;

begin

carry:=0;c:='';

while length(a)<length(b) do a:='0'+a;

while length(b)<length(a) do b:='0'+b;

for i:=length(a) downto 1 do

begin

sum:=ord(a[i])-48+ord(b[i])-48+carry;

carry:=sum div 10;

c:=chr(sum mod 10 +48)+c;

end;

if carry>0 then c:='1'+c;

add:=c;

end;

function Fibo(n : longint):bigNum;

var i :longint;

fi\_1, fi\_2, fi : bigNum;

begin

if n<=1 then exit(char(n+48));

fi\_2:='0'; fi\_1:='1';

for i:=2 to n do begin

fi:=add(fi\_1,fi\_2); {fi:=fi\_1 + fi\_2;}

fi\_2:=fi\_1;

fi\_1:=fi;

end;

exit(fi);

end;

var n : longint;

BEGIN

32

write('Nhap N:'); readln(n);

writeln(Fibo(n));

END.

7.12. Ví dụ tính số {{{33 3 + ­

Số Catalan ñược xác ñịnh bởi công thức sau:

c/A//7 (

1

, 1 c\_x0016\_7

7 (

2!

, 1! !

pớ; 0

Rút gọn tử và mẫu cho , 1! ta có:

c/A//7 (

, 2 , 3 . . 2

1 2 . .

Ta sẽ tính tử số bằng cách sử dụng hàm nhân số lớn với số nhỏ, sau ñó sử dụng

hàm chia số lớn cho số nhỏ ñể ñược kết quả cần tính.

type bigNum =string;

function multiply1(a:bigNum;b:longint):bigNum;

var i :integer;

carry,s :longint;

c,tmp :bigNum;

begin

c:='';

carry:=0;

for i:=length(a) downto 1 do

begin

s:=(ord(a[i])-48) \* b + carry;

carry:= s div 10;

c:=chr(s mod 10 + 48)+c;

end;

if carry>0 then str(carry,tmp) else tmp:='';

multiply1:=tmp+c;

end;

function bigDiv1(a:bigNum;b:longint):bigNum;

var s,i,hold:longint;

c:bigNum;

begin

hold:=0;s:=0; c:='';

33

for i:=1 to length(a) do

begin

hold:=hold\*10 + ord(a[i])-48;

s:=hold div b;

hold:=hold mod b;

c:=c+chr(s+48);

end;

while (length(c)>1) and(c[1]='0') do

delete(c,1,1);

bigDiv1:=c;

end;

var n,k :longint;

s :bigNum;

BEGIN

write('Nhap N:'); readln(n);

s:='1';

for k:=(n+2) to 2\*n do s:=multiply1(s,k); {tính tử số}

for k:=1 to n do s:=bigDiv(s,k); {chia cho mẫu số}

writeln(s);

END.

Tuy nhiên, ta có thể rút gọn hoàn toàn mẫu số của phân số trên và chỉ cần sử dụng

hàm nhân số lớn với số nhỏ, chương trình sẽ chạy nhanh hơn.

Bài tập

2.1. Cho s là một xâu chỉ gồm 2 kí tự '0' hoặc '1' mô tả một số nguyên không âm

ở hệ cơ số 2, hãy chuyển số ñó sang hệ cơ số 16 (ñộ dài xâu s không vượt

quá 200).

Ví dụ: 101011002=AC16

1010101111000001001000112=ABC12316

2.2. Cho số nguyên dương N (N+109

)

a) Phân tích N thành thừa số nguyên tố

b) ðếm số ước của N

c) Tính tổng các ước của N

2.3. ðưa ra những số +106

mà cách kiểm tra tính nguyên tố của Fermat bị sai.

34

2.4. Sử dụng sàng số nguyên tố liệt kê các số nguyên tố trong ñoạn Qd, S

2.5. Người ta ñịnh nghĩa một số nguyên dương N ñược gọi là số ñẹp nếu N thoả

mãn một trong hai ñiều kiện sau:

- N bằng 9

- Gọi f(N) là tổng các chữ số của N thì f(N) cũng là số ñẹp

Cho số nguyên dương N (N + 10

, hãy kiểm tra xem N có phải là số ñẹp

không?

2.6. Dùng cách biểu diễn số nguyên lớn bằng xâu và thêm thông tin dấu (sign=1

nếu số lớn là số không âm, sign=-1 nếu số lớn là số âm) ñể xử lí số nguyên

lớn có dấu như sau:

type bigNum = record

sign : longint;

num : string;

end;

Hãy xây dựng các hàm xử lí số nguyên lớn có dấu.

2.7. Dùng cách biểu diễn số nguyên lớn bằng mảng (mỗi phần tử của mảng là một

nhóm các chữ số).

a) Hãy xây dựng các hàm xử lí số nguyên lớn.

b) Sử dụng hàm nhân số nguyên lớn với số nhỏ tính N! với N+2000.

2.8. Tìm K chữ số cuối cùng của MN

(0< K + 9, 0 + M, N + 106

)

Ví dụ: K=2, M=2, N=10, ta có 210=1024, như vậy 2 chữ số cuối cùng của

2

10 là 24

2.9. Cho N (N+ 10) nguyên dương /

, /\_x0016\_, … , / /< = 10N

. Tìm ước số chung

lớn nhất, bội số chung nhỏ nhất của : số trên (chú ý: BSCNN có thể rất

lớn).

2.10. Cho hai số nguyên không âm A, B (0+A+B+10200), tính số lượng số

Fibonacci trong ñoạn [A, B].

2.11. Cho số nguyên dương N (N+10100), hãy tách N thành tổng các số Fibonacci

ñôi một khác nhau.

Ví dụ: N=16=1+5+13

2.12. Cho N là một số nguyên dương không vượt quá 109

. Hãy tìm số chữ số 0 tận

cùng của N!

35

2.13. Cho s là một xâu mô tả số nguyên không âm ở hệ cơ số a, hãy chuyển số ñó

sang hệ cơ số b (1 =a, b+ 16, ñộ dài xâu s không vượt quá 50).

2.14. Xây dựng hàm kiểm tra số nguyên dương N có phải là số chính phương

không? (N<10100)

2.15. Tính c7

U

(0< + + 2000)

2.16. Tính Catalan7 ( + 2000

2.17. Hãy ñếm số cách ñặt quân xe lên bàn cờ sao cho không có quân nào

ăn ñược nhau. 1 + + + 100

2.18. Giả thiết N là số nguyên dương. Số nguyên M là tổng của N với các chữ số

của nó. N ñược gọi là nguồn của M. Ví dụ, N = 245, khi ñó M = 245 + 2 + 4

+ 5 = 256. Như vậy, nguồn của 256 là 245. Có những số không có nguồn và

có số lại có nhiều nguồn. Ví dụ, số 216 có 2 nguồn là 198 và 207.

Cho số nguyên M (M có không quá 100 chữ số) hãy tìm nguồn nhỏ nhất của

nó. Nếu M không có nguồn thì ñưa ra số 0.

2.19. Tính số ước và tổng các ước của N! (N+100)

2.20. Cho một chiếc cân hai ñĩa và các quả cân có khối lượng 30

, 31

, 32

,…

Hãy chọn các quả cân ñể có thể cân ñược vật có khối lượng N (N+10100)

Ví dụ: cần cân vật có khối lượng N=11 ta cần sử dụng các quả cân sau:

- Cân bên trái: quả cân 31

và 32

- Cân bên phải: quả cân 30

và vật N=11

2.21. ðếm số lượng dãy nhị phân khác nhau ñộ dài mà không có 2 số 1 nào

ñứng cạnh nhau?

Ví dụ: ( 3, ta có 5 dãy 000, 001, 010, 100, 101

2.22. Cho xâu s chỉ gồm kí tự từ 'a' ñến 'z' (ñộ dài xâu s không vượt quá 100), hãy

ñếm số hoán vị khác nhau của xâu ñó.

Ví dụ: s='aba', ta có 3 hoán vị 'aab','aba','baa'

2.23. John Smith quyết ñịnh ñánh số trang cho quyển sách của anh ta từ 1 ñến N.

Hãy tính toán số lượng chữ số 0 cần dùng, số lượng chữ số 1 cần dùng,.., số

lượng chữ số 9 cần dùng.

Dữ liệu vào trong file: “digits.inp” gồm 1 dòng duy nhất chứa một số N

(N≤10100).

36

Kết quả ra file “digits.out” có dạng gồm 10 dòng, dòng thứ nhất là số lượng

chữ số 0 cần dùng, dòng thứ hai là số lượng chữ số 1 cần dùng,.., dòng thứ

10 là số lượng chữ số 9 cần dùng.

2.24. TAM GIÁC SỐ (ñề thi học sinh giỏi Hà Tây 2006)

Hình bên mô tả một tam giác số

có số hàng N=5. ði từ ñỉnh (số 7)

ñến ñáy tam giác bằng một ñường

gấp khúc, mỗi bước chỉ ñược ñi

từ số ở hàng trên xuống một

trong hai số ñứng kề bên phải hay

bên trái ở hàng dưới, và tính tích

các số trên ñường ñi lại ta ñược

một tích.

Ví dụ: ñường ñi 7 8 1 4 6 có tích là S=1344, ñường ñi 7 3 1 7 5 có tích là

S=735.

Yêu cầu: Cho tam giác số, tìm tích của ñường ñi có tích lớn nhất

Dữ liệu: Vào từ file văn bản TGS.INP:

• Dòng ñầu tiên chứa số nguyên n, (0<n<101)

• N dòng tiếp theo, từ dòng thứ 2 ñến dòng thứ N+1: dòng thứ i có (i-1) số

cách nhau bởi dấu cách (các số có giá trị tuyệt ñối không vượt quá 100)

Kết quả: ðưa ra file văn bản TGS.OUT một số nguyên – là tích lớn nhất tìm

ñược

TGS.INP TGS.OUT

5

7

3 8

8 1 0

2 7 4 4

4 5 -2 6 5

5880

2.25. HÁI NẤM (bài thi Olympic Sinh viên 2009, khối chuyên)

Một cháu gái hàng ngày ñược mẹ giao nhiệm vụ ñến thăm bà nội. Từ nhà

mình ñến nhà bà nội cô bé phải ñi qua một khu rừng có rất nhiều loại nấm.

Trong số các loại nấm, có ba loại có thể ăn ñược. Cô bé ñánh số ba loại nấm

ăn ñược lần lượt là 1, 2 và 3. Là một người cháu hiếu thảo cho nên cô bé

37

quyết ñịnh mỗi lần ñến thăm bà, cô sẽ hái ít nhất hai loại nấm ăn ñược ñể

nấu súp cho bà. Khu rừng mà cô bé ñi qua ñược chia thành lưới ô vuông

gồm m hàng và n cột. Các hàng của lưới ñược ñánh số từ trên xuống dưới

bắt ñầu từ 1, còn các cột – ñánh số từ trái sang phải, bắt ñầu từ 1. Ô nằm

giao của hàng i và cột j có tọa ñộ (i, j). Trên mỗi ô vuông, trừ ô (1,1) và ô

(m, n) các ô còn lại hoặc có nấm ñộc và cô bé không dám ñi vào (ñánh dấu

là -1), hoặc là có ñúng một loại nấm có thể ăn ñược (ñánh dấu bằng số hiệu

của loại nấm ñó). Khi cô bé ñi vào một ô vuông có nấm ăn ñược thì cô bé

sẽ hái loại nấm mọc trên ô ñó. Xuất phát từ ô (1,1), ñể ñến ñược nhà bà nội

ở ô (m, n) một cách nhanh nhất cô bé luôn ñi theo hướng sang phải hoặc

xuống dưới.

Việc ñi thăm bà và hái nấm trong rừng sâu gặp nguy hiểm bởi có một con

cho sói luôn theo dõi và muốn ăn thịt cô bé. ðể phòng tránh chó sói theo dõi

và ăn thịt, cô bé quyết ñịnh mỗi ngày sẽ ñi theo một con ñường khác nhau

(hai con ñường khác nhau nếu chúng khác nhau ở ít nhất một ô).

Yêu cầu: Cho bảng m×n ô vuông mô tả trạng thái khu rừng. Hãy tính số con

ñường khác nhau ñể cô bé ñến thăm bà nội theo cách chọn ñường ñi ñã nêu

ở trên.

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

- Dòng ñầu chứa 2 số m, n (1 < m, n <101),

- m dòng tiếp tiếp theo, mỗi dòng chứa n số nguyên cho biết thông tin về

các ô của khu rừng. (riêng giá trị ở hai ô (1,1) và ô (m, n) luôn luôn

bằng 0 các ô còn lại có giá trị bằng -1, hoặc 1, hoặc 2, hoặc 3).

Hai số liên tiếp trên một dòng cách nhau một dấu cách.

Kết quả: ðưa ra file văn bản MUSHROOM.OUT chứa một dòng ghi một số

nguyên là kết quả bài toán.

Ví dụ:

MUSHROOM.INP MUSHROOM.OUT

3 4

0 3 -1 2

3 3 3 3

3 1 3 0

3

2.26. HỆ THỐNG ðÈN MÀU (Tin học trẻ bảng B năm 2009)

ðể trang trí cho lễ kỉ niệm 15 năm hội thi Tin học trẻ toàn quốc, ban tổ chức

ñã dùng một hệ thống ñèn mầu gồm ñèn ñánh số từ 1 ñến . Mỗi ñèn có

38

khả năng sáng màu xanh hoặc màu ñỏ. Các ñèn ñược ñiều khiển theo quy

tắc sau:

- Ban ñầu tất cả các ñèn ñều sáng màu xanh.

- Sau khi kết thúc chương trình thứ nhất của lễ kỉ niệm, tất cả các ñèn có số

thứ tự chia hết cho 2 sẽ ñổi màu…Sau khi kết thúc chương trình thứ ;, tất cả

các ñèn có số thứ tự chia hết cho ; , 1 sẽ ñổi màu (ñèn xanh ñổi thành màu

ñỏ còn ñèn ñỏ ñổi thành màu xanh)

Minh, một thí sinh dự lễ kỉ niệm ñã phát hiện ñược quy luật ñiều khiển ñèn

và rất thích thú với hệ thống ñèn trang trí này. Vào lúc chương trình thứ

của buổi lễ vừa kết thúc, Minh ñã nhẩm tính ñược tại thời ñiểm ñó có bao

nhiêu ñèn xanh và bao nhiêu ñèn ñỏ. Tuy nhiên vì không có máy tính nên

Minh không chắc chắn kết quả của mình là ñúng. Cho biết hai số và

, + 10

, em hãy tính lại giúp Minh xem khi chương trình thứ của

buổi lễ vừa kết thúc, có bao nhiêu ñèn màu ñỏ.

Ví dụ với ( 10; ( 3.

Thời ñiểm Trạng thái các ñèn

Bắt ñầu Xanh: 1 2 3 4 5 6 7 8 9 10

ðỏ :

Sau chương trình 1 Xanh: 1 3 5 7 9

ðỏ : 2 4 6 8 10

Sau chương trình 2 Xanh: 1 5 6 7

ðỏ : 2 3 4 8 9 10

Sau chương trình 3 Xanh: 1 4 5 6 7 8

ðỏ : 2 3 9 10

Vậy có 4 ñèn ñỏ sau chương trình thứ 3.

39

Chuyên ñề 3

SẮP XẾP

Sắp xếp là quá trình bố trí lại vị trí các ñối tượng của một danh sách theo một trật

tự nhất ñịnh. Sắp xếp ñóng vai trò rất quan trọng trong cuộc sống nói chung và

trong tin học nói riêng, thử hình dung xem, một cuốn từ ñiển, nếu các từ không

ñược sắp xếp theo thứ tự, sẽ khó khăn như thế nào trong việc tra cứu các từ. Theo

D.Knuth thì 40% thời gian tính toán của máy tính là dành cho việc sắp xếp.

Không phải ngẫu nhiên thuật toán sắp xếp nhanh (Quick Sort) ñược bình chọn là

một trong 10 thuật toán tiêu biểu của thế kỉ 20.

Do ñặc ñiểm dữ liệu (kiểu số hay phi số, kích thước bé hay lớn, lưu trữ ở bộ nhớ

trong hay bộ nhớ ngoài, truy cập tuần tự hay ngẫu nhiên...) mà người ta có các

thuật toán sắp xếp khác nhau. Trong chuyên ñề này, chúng ta chỉ quan tâm ñến

các thuật toán sắp xếp trong trường hợp dữ liệu ñược lưu trữ ở bộ nhớ trong

(nghĩa là toàn bộ dữ liệu cần sắp xếp phải ñược ñưa vào bộ nhớ chính của máy

tính).

1. Phát biểu bài toán

Giả sử các ñối tượng cần sắp xếp ñược biểu diễn bởi bản ghi gồm một số trường.

Một trong các trường ñó ñược gọi là khoá sắp xếp. Kiểu của khoá là kiểu có thứ

tự (chẳng hạn, kiểu số nguyên, kiểu số thực,…)

const

MAX =…;

type

object = record

key : keyType;

[các trường khác]

end;

TArray = array[1..MAX]of object;

var

a : TArray;

n : longint;

40

Bài toán sắp xếp ñược phát biểu như sau: Cho mảng / các ñối tượng, cần sắp xếp

lại các thành phần (phần tử) của mảng / ñể nhận ñược mảng / mới với các thành

phần có các giá trị khoá tăng dần:

/Q1S.  + /Q2S.  + [ + /QS. 

2. Các thuật toán sắp xếp thông dụng

Hai thuật toán hay ñược sử dụng nhiều trong thực tế ñó là thuật toán sắp xếp nổi

bọt (BUBBLE SORT) và thuật toán sắp xếp nhanh (QUICK SORT).

2.1 Thuật toán sắp xếp nổi bọt (Bubble Sort)

Ý tưởng cơ bản của thuật toán là tìm và ñổi chỗ các cặp phần tử kề nhau sai thứ tự

(phần tử ñứng trước có khoá lớn hơn khoá của phần tử ñứng sau) cho ñến khi

không tồn tại cặp nào sai thứ tự (dãy ñược sắp xếp).

Cụ thể:

- Lượt 1: ta xét từ cuối dãy, nếu gặp 2 phần tử kề nhau mà sai thứ tự thì ñổi

chỗ chúng cho nhau. Sau lượt 1, phần tử có khoá nhỏ thứ nhất ñược ñưa về

vị trí 1.

- Lượt 2: ta xét từ cuối dãy (chỉ ñến phần tử thứ 2), nếu gặp 2 phần tử kề

nhau mà sai thứ tự thì ñổi chỗ chúng cho nhau. Sau lượt 2, phần tử có khoá

nhỏ thứ hai ñược ñưa về vị trí 2.

- Lượt i: ta xét từ cuối dãy về (chỉ ñến phần tử thứ i, vì phần ñầu dãy từ 1

ñến i-1 ñã ñược xếp ñúng thứ tự), nếu gặp 2 phần tử kề nhau mà sai thứ tự

thì ñổi chỗ chúng cho nhau. Sau lượt i, phần tử có khoá nhỏ thứ i ñược ñưa

về vị trí i.

Xong lượt thứ n-1 thì dãy ñược sắp xếp xong.

procedure BoubbleSort;

var i, j : integer;

tmp : object;

begin

for i := 1 to n-1 do

for j := n downto i+1 do

if a[j-1].key > a[j].key then

begin

tmp := a[j];

41

a[j]:= a[j-1];

a[j-1]:= tmp;

end;

end;

ðánh giá ñộ phức tạp

Số phép toán so sánh /QY \_x0007\_ 1S.  & /QYS.  ñược dùng ñể ñánh giá hiệu suất

thuật toán về mặt thời gian cho thuật toán sắp xếp nổi bọt. Tại lượt thứ i ta cần

\_x0007\_ ; phép so sánh. Như vậy tổng số phép so sánh cần thiết:

\_x0007\_ 1 , \_x0007\_ 2 , [ , 1 (

\_x0007\_ 1

2

Thuật toán có ñộ phức tạp ):\_x0016\_

Một thuật toán sắp xếp ñơn giản, hay sử dụng khác cũng cho ñộ phức tạp ):

\_x0016\_

for i:=1 to n-1 do

for j:=i+1 to n do

if a[i].key>a[j].key then begin

tmp:=a[i];

a[i]:=a[j];

a[j]:=tmp;

end;

2.2. Thuật toán sắp xếp nhanh (Quick Sort)

Ý tưởng của thuật toán như sau: ðể sắp xếp dãy coi như là sắp xếp ñoạn từ chỉ số

1 ñến chỉ số . ðể sắp xếp một ñoạn trong dãy, nếu ñoạn chỉ có một phần tử thì

dãy ñã ñược sắp xếp, ngược lại ta chọn một phần tử 0 trong ñoạn ñó làm "chốt",

mọi phần tử có khoá nhỏ hơn khoá của “chốt” ñược xếp vào vị trí ñứng trước

chốt, mọi phần tử có khoá lớn hơn khoá của “chốt” ñược xếp vào vị trí ñứng sau

chốt. Sau phép hoán chuyển như vậy thì ñoạn ñang xét ñược chia làm hai ñoạn mà

mọi phần tử trong ñoạn ñầu ñều có khoá ≤ khoá của “chốt” và mọi phần tử trong

ñoạn sau ñều có khoá ≥ khoá của “chốt”. Tiếp tục sắp xếp kiểu như vậy với 2

ñoạn con, ta sẽ ñược ñoạn ñã cho ñược sắp xếp theo chiều tăng dần của khoá.

Cụ thể:

Giả sử phải sắp xếp ñoạn có chỉ số từ L ñến H:

42

- chọn 0 là một phần tử ngẫu nhiên trong ñoạn L..H (có thể chọn 0 là phần

tử ở giữa ñoạn, nghĩa là 0 ( a[(L+H) div 2])

- cho i chạy từ L sang phải, j chạy từ H sang trái; nếu phát hiện một cặp

ngược thứ tự: i ≤ j và /Q;S.  ≥ 0.  ≥ /QYS.  thì ñổi chỗ 2 phần tử

ñó; cho ñến khi i>j. Lúc ñó dãy ở tình trạng: khoá các phần tử ñoạn L..i ≤

khoá của 0; khoá của các phần tử ñoạn j..H ≥ khoá của 0. Tiếp tục sắp xếp

như vậy với 2 ñoạn L..j và i..H.

Thủ tục QuickSort(L,H) sau, sắp xếp ñoạn từ L tới H, ñể sắp xếp dãy số ta

gọi QuickSort(1,n)

procedure QuickSort(L,H:longint);

var i,j :longint;

x,tmp :object;

begin

i:=L;

j:=H;

//x:=a[random(H-L+1)+L];

x:=a[(L+H) div 2];

repeat

while a[i].key<x.key do inc(i);

while a[j].key>x.key do dec(j);

if i<=j then

begin

tmp:=a[i];

a[i]:=a[j];

a[j]:=tmp;

inc(i);

dec(j);

end;

until i>j;

if L<j then QuickSort(L,j);

if i<H then QuickSort(i,H);

end;

ðánh giá ñộ phức tạp

Việc chọn chốt ñể phân ñoạn quyết ñịnh hiệu quả của thuật toán, nếu việc chọn

chốt không tốt rất có thể việc phân ñoạn bị suy biến thành trường hợp xấu (phân

43

thành hai ñoạn mà số phần tử của hai ñoạn chênh lệch nhiều) khiến Quick Sort

hoạt ñộng chậm. Các tính toán ñộ phức tạp chi tiết cho thấy thuật toán Quick sort:

- Có thời gian thực thi cỡ ) 2 trong trường hợp trung bình.

- Có thời gian thực thi cỡ )\_x0016\_

trong trường hợp xấu nhất (2 ñoạn ñược

chia thành một ñoạn n-1 và một ñoạn 1 phần tử). Khả năng ñể xảy ra trường

hợp này là rất ít, còn nếu chọn chốt ngẫu nhiên, hầu như sẽ không xảy ra.

2.3. Nhận xét

Nếu chương trình ít gọi tới thủ tục sắp xếp và chỉ trên tập dữ liệu nhỏ, thì việc sử

dụng một thuật toán phức tạp (tuy có hiệu quả hơn) có thể không cần thiết, khi ñó

có thể sử dụng thuật toán ñơn giản có ñộ phức tạp ):

\_x0016\_

, dễ cài ñặt. Tuy nhiên,

vì ñộ phức tạp )\_x0016\_

, nghĩa là thời gian thực hiện tăng lên gấp 4 khi số lượng

phần tử tăng lên gấp ñôi. Do ñó, trong trường hợp sắp xếp trên tập dữ liệu lớn nên

sử dụng thuật toán sắp xếp nhanh có ñộ phức tạp cỡ ) 2.

3. Sắp xếp bằng ñếm phân phối (Distribution Counting)

Trong trường hợp khoá các phần tử aQ1S, aQ2S , . . . , aQnS là các số nguyên nằm

trong khoảng từ 0 tới  ta có thuật toán ñơn giản và hiệu quả như sau:

Xây dựng dãy \*Q0S, \*Q1S, … , \*QS , trong ñó \*QS là số lần xuất hiện khoá  trong

dãy.

for V := 0 to K do c[V] := 0;{Khởi tạo dãy c}

for i := 1 to n do c[a[i].key] := c[a[i].key] + 1;

Như vậy, sau khi sắp xếp:

- Các phần tử có khoá bằng 0 ñứng trong ñoạn từ vị trí 1 tới vị trí \*Q0S.

- Các phần tử có khoá bằng 1 ñứng trong ñoạn từ vị trí \*Q0S , 1 tới vị trí

\*Q0S , \*Q1S.

- Các phần tử có khoá bằng 2 ñứng trong ñoạn từ vị trí \*Q0S , \*Q1S , 1 tới

vị trí \*Q0S , \*Q1S , \*Q2S.

...

- Các phần tử có khoá bằng  trong ñoạn ñứng từ vị trí \*Q0S , \*Q1S , [ ,

\*Q \_x0007\_ 1S , 1 tới vị trí \*Q0S , \*Q1S , [ , \*Q \_x0007\_ 1S , \*QS.

...

- Các phần tử có khoá bằng  trong ñoạn ñứng từ vị trí \*Q0S , \*Q1S , [ ,

\*Q \_x0007\_ 1S , 1 tới vị trí \*Q0S , \*Q1S , [ , \*QS.

Ví dụ: với dãy gồm 8 phần tử có dãy khoá bằng : 2, 0, 2, 5, 1, 2, 0, 3 ta có

44

\*Q0S \*Q1S \*Q2S \*Q3S \*Q4S \*Q5S

2 1 3 1 0 1

Sau khi sắp xếp, các phần tử có khoá bằng 0 sẽ nằm từ vị trí 1 ñến vị trí 2, phần tử

có khoá bằng 1 nằm ở vị trí 3, các phần tử có khoá bằng 2 nằm từ vị trí 4 ñến vị trí

6, phần tử có khoá bằng 3 nằm ở vị trí 7, các phần tử có khoá bằng 5 nằm ở vị

trí 8.

Dãy khoá sau khi sắp xếp: 0, 0, 1, 2, 2, 2, 3, 5

ðộ phức tạp của thuật toán là: )/0:, 

4. Một số ví dụ ứng dụng thuật toán sắp xếp

Ví dụ 1: Giá trị nhỏ thứ

Cho dãy /

, /\_x0016\_, … , /7, các số ñôi một khác nhau và số nguyên dương 1 + +

. Hãy ñưa ra giá trị nhỏ thứ trong dãy.

Ví dụ dãy gồm 5 phần tử: 5, 7, 1, 3, 4 và ( 3 thì giá trị nhỏ thứ là 4.

Giải

Sắp xếp dãy theo giá trị tăng dần, số ñứng thứ của dãy là giá trị nhỏ thứ . Nếu

+ 5000 có thể sử dụng thuật toán sắp xếp nổi bọt, nhưng nếu & 5000 thì nên

sử dụng thuật toán sắp xếp nhanh.

Ta có thể tìm ñược giá trị nhỏ thứ hiệu quả hơn (không cần phải sắp xếp lại cả

dãy số) cụ thể:

+ Trong thuật toán sắp xếp nổi bọt ta chỉ cần sắp xếp ñến phần tử thứ , khi ñó

phần tử thứ chính là phần tử có khoá nhỏ thứ .

for i:=1 to k do // i chạy ñến k

for j:=n downto i+1 do

if a[j-1]>a[j] then

begin

tmp:=a[j];

a[j]:=a[j-1];

a[j-1]:=tmp;

end;

+ Trong thuật toán Quick Sort, ta thấy rằng:

• Nếu <L<=H thì ñoạn từ L ñến H không cần sắp xếp vì ñoạn này không

ảnh hưởng ñến vị trí thứ .

45

• Nếu L<=H< thì ñoạn từ L ñến H cũng không cần sắp xếp vì ñoạn này

không ảnh hưởng ñến vị trí thứ .

• Nếu L<=<=H thì ta sẽ xử lí tiếp trong ñoạn này.

procedure QuickSort(L,H:longint);

var i,j :longint;

x,tmp :longint;

begin

if (L<=K) and (H>=K) then

begin

i:=L;

j:=H;

x:=a[(L+H) div 2];

repeat

while a[i]<x do inc(i);

while a[j]>x do dec(j);

if i<=j then

begin

tmp:=a[i];

a[i]:=a[j];

a[j]:=tmp;

inc(i);

dec(j);

end;

until i>j;

if L<j then QuickSort(L,j);

if i<H then QuickSort(i,H);

end;

end;

Sau khi gọi và thực hiện thủ tục QuickSort(1,n) thì số ñứng thứ chính là giá trị

nhỏ thứ .

Chú ý: khi @ là số nhỏ thứ (:

;p 2 , 1) của dãy /

, /\_x0016\_, … , /7 thì hàm

Z@ ( |/

\_x0007\_ @| , |/\_x0016\_ \_x0007\_ @| , [ , |/7 \_x0007\_ @|

ñạt giá trị nhỏ nhất

46

Ví dụ 2: Tìm kiếm

Cho dãy ñã ñược sắp tăng dần /

≤ /\_x0016\_ ≤ . . . ≤ /7 và số @. Hãy ñưa ra chỉ số i mà

/< ( @ hoặc ñưa ra i=0 nếu không có phần tử nào có giá trị bằng @.

Giải

Thuật toán tìm kiếm nhị phân có thể tìm phần tử có giá trị bằng @ trên mảng ñã

ñược sắp xếp một cách hiệu quả trong thời gian ) 2. Thuật toán như sau:

Giả sử cần tìm trong ñoạn /QdS, /Qd , 1S, . . , /QS với giá trị cần tìm kiếm là @,

trước hết ta xem xét với giá trị của phần tử nằm giữa dãy, ;

( d , 

;p 2

• Nếu /Q;

S < X thì có nghĩa là ñoạn từ /QdS tới /Q;

S chỉ chứa các

phần tử có giá trị < X, ta tiến hành tìm kiếm tiếp với ñoạn từ /Q;

, 1S

ñến /QS

• Nếu /Q;

S > X thì có nghĩa là ñoạn từ /Q;

S tới fQS chỉ chứa các

phần tử có giá trị > X, ta tiến hành tìm kiếm tiếp với ñoạn từ /QdS ñến

/Q;

\_x0007\_ 1S

• Nếu /Q;

S = X thì việc tìm kiếm thành công (kết thúc quá trình tìm

kiếm).

Quá trình tìm kiếm sẽ thất bại nếu ñến một bước nào ñó, ñoạn tìm kiếm là rỗng

(d & )

function BinarySearch(X: longint): longint;

var L, H, mid: longint;

begin

L := 1; H := n;

while L ≤ H do

begin

mid := (L + H) div 2;

if a[mid] = X then exit(mid);

if a[mid] < X then L := mid + 1

else H := mid - 1;

end;

exit(0);

end;

47

Ví dụ 3: Thống kê

Cho dãy /

, /\_x0016\_, … , /7. Hãy ñếm số lượng giá trị khác nhau có trong dãy và ñưa ra

số lần lặp của giá trị xuất hiện nhiều nhất.

Ví dụ: dãy gồm 8 số: 6, 7, 1, 7, 4, 6, 6, 8 thì dãy có 5 giá trị khác nhau và số lần

lặp của giá trị xuất hiện nhiều nhất trong dãy là 3.

Giải

Các công việc trên sẽ ñược thực hiện ñơn giản nếu mảng ñã ñược sắp xếp, khi ñó

các phần tử có giá trị bằng nhau sẽ ñứng cạnh nhau (liên tiếp nhau).

{ Hàm countValue trả về số giá trị khác nhau trong mảng a có n phần tử ñã sắp

xếp}

function countValue(a : TArray;n : longint):longint;

var i,count :longint;

begin

count:=1;

for i:=2 to n do

if a[i-1]<>a[i] then inc(count);

countValue := count;

end;

{ Hàm highestFrequency trả về số lần lặp của giá trị xuất hiện nhiều nhất

trong mảng a có n phần tử ñã sắp xếp}

function highestFrequency(a:TArray; n:longint):longint;

var i,count,rslt :longint;

begin

rslt:=1;

count:=1;

for i:=2 to n do begin

if a[i] <> a[i-1] then count:=1

else inc(count);

if count>rslt then rslt:=count;

end;

highestFrequency := rslt;

end;

Ví dụ 4. Xét dãy Z gồm 2 = + 10

số nguyên Z ( '

, '\_x0016\_, … , '7

ñịnh

nghĩa như sau:

48

'< ( 1, nếu 1+ ; + 2

'<>

, '<>\_x0016\_ mod 128, nếu 2 = ; +

Hãy cho biết nếu sắp xếp dãy Z theo thứ tự không giảm thì số thứ + có

giá trị là bao nhiêu?

Giải

Ta nhận thấy '<

có giá trị nguyên và 0 + '< + 127, ta sẽ sử dụng thuật toán ñếm

phân phối như sau:

- Xây dựng dãy Z và dãy \*Q0S, \*Q1S, … , \*Q127S , trong ñó \*QS là số lần xuất

hiện giá trị  trong dãy Z.

- Giá trị thứ của dãy Z sau khi sắp xếp là giá trị  nhỏ nhất thoả mãn

\*Q0S , \*Q1S , … , \*QS

Chú ý: Sử dụng / /

2. \_x0007\_ 1 thay cho /

2

., chương trình sẽ chạy

nhanh hơn.

const maxValue =128 – 1;

var fi\_2, fi\_1, fi :longint;

i, v, n, k, cv, P :longint;

c :array[0..maxValue]of longint;

BEGIN

write('Nhap n, k:');readln(n,k);

fi\_1:=1; fi\_2:=1;

fillchar(c,sizeof(c),0);

c[fi\_1]:=c[fi\_1]+1;

c[fi\_2]:=c[fi\_2]+1;

for i:=3 to n do begin

fi:=(fi\_1+fi\_2) and maxValue;

{write(fi:4);}

c[fi]:=c[fi]+1;

fi\_2:=fi\_1;

fi\_1:=fi;

end;

cv:=0;

for v:=0 to maxValue do begin

cv:=cv+c[v];

if cv >= k then begin

P:=v;

49

break;

end;

end;

writeln(P)

END.

Ví dụ 5. Cho dãy gồm : : + 30000 số tự nhiên không vượt quá 10N

, tìm số tự

nhiên nhỏ nhất không xuất hiện trong dãy.

Dữ liệu vào trong file SN.INP có dạng:

- Dòng ñầu là số nguyên :

- Dòng thứ hai gồm : số

Kết quả ra file SN.OUT có dạng: số tự nhiên nhỏ nhất không xuất hiện trong dãy.

SN.INP SN.OUT

5

5 0 3 1 4

2

Giải

Ta có nhận xét sau: số tự nhiên nhỏ nhất không xuất hiện trong dãy sẽ nằm trong

ñoạn [0, n]. Do ñó, ta sử dụng mảng c:array[0..30000]of longint;

với c[x]là số lần xuất hiện của x trong dãy, nếu c[x]=0 tức là x không xuất

hiện trong dãy.

const Limit =30000;

fi ='SN.INP';

fo ='SN.OUT';

var c :array[0..Limit]of longint;

n :longint;

i,x :longint;

f :text;

BEGIN

fillchar(c,sizeof(c),0);

assign(f,fi); reset(f);

readln(f,n);

for i:=1 to n do begin

read(f,x);

if x<=n then inc(c[x]);

end;

50

close(f);

for i:=0 to n do

if c[i]=0 then begin

x:=i;

break;

end;

assign(f,fo); rewrite(f);

write(f,x);

close(f);

END.

Ví dụ 6. Cho xâu s (ñộ dài không vượt quá 106

) chỉ gồm 2 kí tự 'A' và 'B'. ðếm số

cách chọn cặp chỉ số (i,j) mà xâu con liên tiếp từ kí tự thứ i ñến kí tự thứ j của xâu

s có số lượng kí tự 'A' bằng số lượng kí tự 'B'.

Dữ liệu vào trong file “AB.INP” có dạng: gồm một dòng duy nhất chứa xâu s

Kết quả ra file “AB.OUT” có dạng: gồm một dòng duy nhất chứa một số là kết

quả bài toán.

AB.INP AB.OUT

ABAB 4

Giải

const MAX =1000000;

fi ='AB.INP';

fo ='AB.OUT';

var s :ansistring;

c :array[-MAX..MAX]of longint;

f :text;

i, sum :longint;

count :int64;

BEGIN

assign(f,fi); reset(f);

read(f,s);

close(f);

fillchar(c,sizeof(c),0);

c[0]:=1;

sum:=0;

51

count:=0;

for i:=1 to length(s) do begin

if s[i]='A' then sum:=sum - 1

else sum:=sum + 1;

count:=count + c[sum];

inc(c[sum]);

end;

assign(f,fo); rewrite(f);

write(f,count);

close(f);

END.

Bài tập

3.1. Cho một danh sách học sinh (1 ≤ ≤ 200), mỗi học sinh có thông tin sau:

- Họ và tên: Là một xâu kí tự ñộ dài không quá 30 (các từ cách nhau một

dấu cách)

- ðiểm: Là một số thực

A) ðưa ra danh sách họ và tên ñã sắp xếp theo thứ tự abc (ưu tiên tên, họ,

ñệm)

B) Có bao nhiêu tên khác nhau trong danh sách, liệt kê các tên ñó.

C) Chọn những học sinh có thứ hạng 1, 2, 3 ñiểm cao nhất trong danh sách

ñể trao học bổng, hãy cho biết tên những học sinh ñó.

Ví dụ

Dữ liệu vào Kết quả câu A Kết quả

câu B

Kết quả câu

C

6

Vu Anh Quan

8.9

Nguyen Van

Chung

8.7

Hoang Trong

Quynh

8.5

Nguyen Van

Chung

Cong Hoang

Dinh Quang

Hoang

Dinh Quang

Huy

Vu Anh Quan

Hoang Trong

5

Chung

Hoang

Huy

Quan

Quynh

Vu Anh Quan

Dinh Quang

Huy

Dinh Quang

Hoang

Nguyen Van

Chung

52

Dinh Quang

Hoang

8.7

Dinh Quang

Huy

8.8

Cong Hoang

8.0

Quynh

3.2. Cho dãy số gồm : số nguyên /

+ /\_x0016\_ +. . . + /

A) ðưa ra thuật toán có ñộ phức tạp ): 2: ñể tìm 2 chỉ số ; = Y mà

/< , /T ( 0.

B) ðưa ra thuật toán có ñộ phức tạp ):\_x0016\_

 2: ñể tìm 3 chỉ số ; = Y =

mà /< , /T , /U ( 0.

C) ðưa ra thuật toán có ñộ phức tạp ): ñể tìm 2 chỉ số ; = Y mà

/< , /T ( 0.

D) ðưa ra thuật toán có ñộ phức tạp ):\_x0016\_

ñể tìm 3 chỉ số ; = Y = mà

/< , /T , /U ( 0.

3.3. Cho một xâu s (ñộ dài không quá 200) chỉ gồm các kí tự / ñến , ñếm số

lượng xâu con liên tiếp khác nhau nhận ñược từ xâu s.

Ví dụ: s='/9/9', ta có các xâu con liên tiếp khác nhau là:

′/′, ′9′, ′/9′, ′9/′, ′/9/′, ′9/9′, ′/9/9′, số lượng xâu con liên tiếp khác nhau

là 7.

3.4. Viết liên tiếp các số tự nhiên từ 1 ñến : ta ñược một số nguyên . Ví dụ

:=15 ta có =123456789101112131415. Hãy tìm cách xoá ñi  chữ số

của số  ñể nhận ñược số ' là lớn nhất.

3.5. Xét tập Z: tất cả các số hữu tỷ trong ñoạn [0,1] với mẫu số không vượt quá

N 1 = : + 100.

Ví dụ tập Z5: 0/1 1/5 1/4 1/3 2/5 1/2 3/5 2/3 3/4 4/5 1/1

Sắp xếp các phân số trong tập Z: theo thứ tự tăng dần, ñưa ra phân số

thứ .

3.6. Cho xâu s (ñộ dài không vượt quá 106

) chỉ gồm các kí tự ′/′ ñến ′′,

A) Có bao nhiêu loại kí tự xuất hiện trong s

53

B) ðưa ra một kí tự xuất hiện nhiều nhất trong xâu s và số lần xuất hiện của

kí tự ñó.

3.7. Cho 2 dãy /

≤ /\_x0016\_ ≤ . . . ≤ /7 và 9

≤ 9\_x0016\_ ≤ . . . ≤ 9., hãy ñưa ra thuật toán có ñộ

phức tạp ) , ñể có dãy \*

≤ \*\_x0016\_≤ … ≤ \*78. là dãy trộn của hai dãy trên.

3.8. Cho dãy số gồm + 10000 số nguyên /

, /\_x0016\_, … , /7 (|/<

| + 10N

, tìm số

nguyên @ bất kì ñể b ( |/

\_x0007\_ @| , |/\_x0016\_ \_x0007\_ @| , [ , |/7 \_x0007\_ @| ñạt giá trị nhỏ

nhất, có bao nhiêu giá trị nguyên khác nhau thoả mãn.

Ví dụ 1: dãy gồm 5 số 3, 1, 5, 4, 5, ta có duy nhất một giá trị @ ( 4 ñể b ñạt

giá trị nhỏ nhất bằng 6.

Ví dụ 2: dãy gồm 6 số 3, 1, 7, 2, 5, 7 ta có ba giá trị nguyên của @ là 3, 4, 5

ñể b ñạt giá trị nhỏ nhất bằng 13.

3.9. Cho : : + 10000 ñiểm trên mặt phẳng Oxy, ñiểm thứ i có tọa ñộ là

0<

, <. Ta ñịnh nghĩa khoảng cách giữa 2 ñiểm P(xP, yP) và Q(xQ, yQ) bằng

|0 \_x0007\_ 0| , | \_x0007\_ |. Hãy tìm ñiểm f có tọa ñộ nguyên mà tổng khoảng

cách (theo cách ñịnh nghĩa trên) từ f tới N ñiểm ñã cho là nhỏ nhất

(|0<

|, |<

| nguyên không vượt quá10N

)

3.10. Cho : : + 10000 ñoạn thẳng trên trục số với các ñiểm ñầu 0<

và ñộ dài

<

|0<

|,

<

là những số nguyên và không vượt quá 10N

. Tính tổng ñộ dài

trên trục số bị phủ bởi : ñoạn trên.

Ví dụ: có 3 ñoạn 0

( \_x0007\_5,

( 10; 0\_x0016\_ ( 0,

\_x0016\_ ( 6; 0 ( \_x0007\_100,

( 10

thì tổng ñộ dài trên trục số bị phủ bởi 3 ñoạn trên là: 21

3.11. Cho N (N + 300 ñiểm trên mặt phẳng Oxy, ñiểm thứ i có tọa ñộ là (xi

, yi).

Hãy ñếm số cách chọn 4 ñiểm trong N ñiểm trên mà 4 ñiểm ñó tạo thành 4

ñỉnh của một hình chữ nhật. (|xi

|, |yi

| nguyên không vượt quá 1000)

Ví dụ: có 5 ñiểm (0, 0), (0, 1), (1, 0), (-1, 0), (0, -1) có duy nhất 1 cách chọn

4 ñiểm mà 4 ñiểm ñó tạo thành 4 ñỉnh của một hình chữ nhật.

3.12. Cho : : + 10000 ñoạn số nguyên Q/<

, 9<

S, hãy tìm một số mà số ñó

thuộc nhiều ñoạn số nguyên nhất.

Ví dụ: có 5 ñoạn Q0,10S, Q2,3S, Q4,7S, Q3,5S, Q5,8S, ta chọn số 5 thuộc 4 ñoạn

Q0,10S, Q4,7S, Q3,5S, Q5,8S.

3.13. Cho dãy gồm : : + 10000 số /

, /\_x0016\_, . . , /. Hãy tìm dãy con liên tiếp dài

nhất có tổng bằng 0. |/<

| + 10N

54

Ví dụ: dãy gồm 5 số 2, 1, -2, 3, -2 thì dãy con liên tiếp dài nhất có tổng bằng

0 là: 1, -2, 3, -2

3.14. ESEQ

Cho dãy số nguyên A gồm N phần tử A1, A2, .., AN, tìm số cặp chỉ số i, j thoả

mãn:

1

i N

p q

p q j

A A

= =

∑ ∑= với 1 ≤ i < j ≤ N

Dữ liệu vào trong file “ESEQ.INP” có dạng:

- Dòng ñầu là số nguyên dương N (2 ≤ N ≤ 105

)

- Dòng tiếp theo chứa N số nguyên A1, A2, .., AN (|Ai|<109

), các số cách

nhau một dấu cách.

Kết quả ra file “ESEQ.OUT” có dạng: gồm một số là số cặp tìm ñược.

ESEQ.INP ESEQ.OUT

3

1 0 1

3

3.15. GHÉP SỐ

Cho n số nguyên dương a1, a2, . . .,an (1 < n ≤ 100), mỗi số không vượt quá

109

. Từ các số này người ta tạo ra một số nguyên mới bằng cách ghép tất cả

các số ñã cho, tức là viết liên tiếp các số ñã cho với nhau. Ví dụ, với n = 4

và các số 123, 124, 56, 90 ta có thể tạo ra các số mới sau: 1231245690,

1241235690, 5612312490, 9012312456, 9056124123,... Có thể dễ dàng

thấy rằng, với n = 4, ta có thể tạo ra 24 số mới. Trong trường hợp này, số

lớn nhất có thể tạo ra là 9056124123.

Yêu cầu: Cho n và các số a1, a2, . . .,an . Hãy xác ñịnh số lớn nhất có thể tạo

ra khi ghép các số ñã cho thành một số mới.

Dữ liệu vào từ file văn bản NUMJOIN.INP có dạng:

- Dòng thứ nhất chứa số nguyên n,

- Dòng thứ 2 chứa n số nguyên a1 a2 . . . an .

Kết quả ra file văn bản NUMJOIN.OUT gồm một dòng là số lớn nhất có thể

tạo ra khi ghép các số ñã cho thành một số mới.

55

3.16. GIÁ TRỊ NHỎ NHẤT

Cho bảng số A gồm MxN ô, mỗi ô chứa một số nguyên không âm (Aij) có

giá trị không vượt quá 109

. Xét hàng i và hàng j của bảng, ta cần xác ñịnh

Xij nguyên ñể:

ij ij ij

1 1

| | | |

N N

ik jk

k k

S A X A X

= =

   

= − + −         ∑ ∑ ñạt giá trị nhỏ nhất.

Tính ( ∑ ∑ b<T

¢

T£<8

¢>

<£

Dữ liệu vào trong file “WMT.INP” có dạng:

- Dòng ñầu là 2 số nguyên dương M, N (1<M, N<1001)

- M dòng sau, mỗi dòng N số

Kết quả ra file “WMT.OUT” có dạng: gồm một số W

WMT.INP WMT.OUT

2 3

2 3 1

2 3 4

5

3.17. DECIPHERING THE MAYAN WRITING (IOI 2006)

Công việc giải mã chữ viết của người MAIA là khó khăn hơn người ta

tưởng nhiều. Trải qua hơn 200 năm mà người ta vẫn hiểu rất ít về các chữ

viết này. Chỉ trong 3 thập niên gần ñây do công nghệ phát triển việc giải mã

này mới có nhiều tiến bộ.

Chữ viết Maia dựa trên các kí hiệu nhỏ gọi là nét vẽ, mỗi nét vẽ tương ứng

với một âm giọng nói. Mỗi từ trong chữ viết Maia sẽ bao gồm một tập hợp

các nét vẽ như vậy kết hợp lại với nhiều kiểu dáng khác nhau. Mỗi nét vẽ có

thể hiểu là một kí tự ta hiểu ngày nay.

Một trong những vấn ñề lớn khi giải mã chữ Maia là thứ tự ñọc các nét vẽ.

Do người Maia trình bày các nét vẽ này không theo thứ tự phát âm, mà theo

cách thể hiện của chúng. Do vậy nhiều khi ñã biết hết các nét vẽ của một từ

rồi nhưng vẫn không thể tìm ra ñược chính xác cách ghi và ñọc của từ này.

Các nhà khảo cổ ñang ñi tìm kiếm một từ ñặc biệt W. Họ ñã biết rõ tất cả

các nét vẽ của từ này nhưng vẫn chưa biết các cách viết ra của từ này. Vì họ

biết có các thí sinh IOI'06 sẽ ñến nên muốn sự trợ giúp của các sinh viên

này. Họ sẽ ñưa ra toàn bộ g nét vẽ của từ W và dãy S tất cả các nét vẽ có

56

trong hang ñá cổ. Bạn hãy giúp các nhà khảo cổ tính xem có bao nhiêu khả

năng xuất hiện từ W trong hang ñá.

Yêu cầu: Hãy viết chương trình, cho trước các kí tự của từ W và dãy S các

nét vẽ trong hang ñá, tính tổng số khả năng xuất hiện của từ W trong dãy S,

nghĩa là số lần xuất hiện một hoán vị các kí tự của dãy g kí tự trong S.

Các ràng buộc

1 ≤ g ≤ 3 000, số nét vẽ trong W

g ≤ |S| ≤ 3 000 000, |S| là số các nét vẽ của dãy S

Dữ liệu vào:

- Dòng 1: chứa 2 số g và |S| cách nhau bởi dấu cách.

- Dòng 2: chứa g kí tự liền nhau là các nét vẽ của từ W. Các kí tư hợp lệ là

'a'-'z' và 'A'-'Z'. Các chữ in hoa và in thường là khác nhau.

- Dòng 3: Chứa |S| kí tự là dãy các nét vẽ tìm thấy trong hang. Các kí tư hợp

lệ là 'a'-'z' và 'A'-'Z'. Các chữ in hoa và in thường là khác nhau.

Kết quả ra:

Chứa ñúng 1 số là khả năng xuất hiện của từ W trong dãy S.

Dữ liệu vào Kết quả ra

4 11

cAda

AbrAcadAbRa

2

3.18. TRÒ CHƠI VỚI DÃY SỐ (Học sinh giỏi quốc gia, 2007-2008)

Hai bạn học sinh trong lúc nhàn rỗi nghĩ ra trò chơi sau ñây. Mỗi bạn chọn

trước một dãy số gồm n số nguyên. Giả sử dãy số mà bạn thứ nhất chọn là:

9

, 9\_x0016\_, . . . , 97

còn dãy số mà bạn thứ hai chọn là: \*

, \*\_x0016\_, . . . , \*7

Mỗi lượt chơi mỗi bạn ñưa ra một số hạng trong dãy số của mình. Nếu bạn

thứ nhất ñưa ra số hạng 9<

1 + ; + , còn bạn thứ hai ñưa ra số hạng

\*T

1 + Y + thì giá của lượt chơi ñó sẽ là |9< , \*T

|.

Ví dụ: Giả sử dãy số bạn thứ nhất chọn là 1, -2; còn dãy số mà bạn thứ hai

chọn là 2, 3. Khi ñó các khả năng có thể của một lượt chơi là (1, 2), (1, 3), (-

2, 2), (-2, 3). Như vậy, giá nhỏ nhất của một lượt chơi trong số các lượt chơi

có thể là 0 tương ứng với giá của lượt chơi (-2, 2).

57

Yêu cầu: Hãy xác ñịnh giá nhỏ nhất của một lượt chơi trong số các lượt chơi

có thể.

Dữ liệu vào:

Dòng ñầu tiên chứa số nguyên dương + 10

Dòng thứ hai chứa dãy số nguyên 9

, 9\_x0016\_, . . . , 97 |9<

| + 10N

, ; (

1, 2, . . . ,

Dòng thứ hai chứa dãy số nguyên \*

, \*\_x0016\_, . . . , \*7 |\*<

| + 10N

, ; ( 1, 2, . . . ,

Hai số liên tiếp trên một dòng ñược ghi cách nhau bởi dấu cách.

Kết quả ra:

Ghi ra giá nhỏ nhất tìm ñược.

Dữ liệu vào Kết quả ra

2

1 -2

2 3

0

3.19. DÃY SỐ (Học sinh giỏi, Hà Nội 2008-2009)

Cho dãy số nguyên /

, /\_x0016\_, . . /7. Số /L 1 + F + ñược gọi là một số

trung bình cộng trong dãy nếu tồn tại 3 chỉ số ;, Y, 1 + ;, Y, +

ñôi một khác nhau, sao cho /L ( /< , /T , /U/3

Yêu cầu: Cho và dãy số /

, /\_x0016\_, . . /7. Hãy tìm số lượng các số trung bình

cộng trong dãy.

Dữ liệu vào:

- Dòng ñầu ghi số nguyên dương 3 + + 1000

- Dòng thứ hai chứa số nguyên /<

|/<

| = 10¥

Kết quả ra:

Số lượng các số trung bình cộng trong dãy.

Dữ liệu vào Kết quả ra

5

4 3 6 3 5

2

58

3.20. ðẾM SỐ TAM GIÁC (Tin học trẻ, bảng B, năm 2009)

Cho ba số nguyên dương /, 9, và , + 10000 ñoạn thẳng ñánh số

từ 1 tới . ðoạn thẳng thứ ; có ñộ dài

<

(¦;: 1 + ; + ), ở ñây các ñộ dài

,

\_x0016\_, … ,

7

ñược cho như sau:

< ( 9, nếu ; ( 1

/

<>

, 9 mod , 1, nếu 1 = ; +

(\*)

Hãy cho biết có bao nhiêu tam giác khác nhau có thể ñược tạo ra bằng cách

lấy ñúng ba ñoạn trong số ñoạn thẳng ñã cho làm ba cạnh (hai tam giác

bằng nhau nếu chúng có ba cặp cạnh tương ứng bằng nhau, nếu không

chúng ñược coi là khác nhau).

Ví dụ với / ( 6; 9 ( 3; ( 4; ( 5. Ta có 5 ñoạn thẳng với ñộ dài của

chúng tính theo công thức (\*) là 3,2,4,4,4. Với 5 ñoạn thẳng này có thể

tạo ra ñược 4 tam giác với ñộ dài các cạnh ñược chỉ ra như sau:

Tam giác 1: (2, 3, 4)

Tam giác 2: (2, 4, 4)

Tam giác 3: (3, 4, 4)

Tam giác 4: (4, 4, 4)

59

Chuyên ñề 4

THIẾT KẾ GIẢI THUẬT

Chuyên ñề này trình bày các chiến lược thiết kế thuật giải như: Quay lui

(Backtracking), Nhánh và cận (Branch and Bound), Tham ăn (Greedy Method),

Chia ñể trị (Divide and Conquer) vả Quy hoạch ñộng (Dynamic Programming).

ðây là các chiến lược tổng quát, nhưng mỗi phương pháp chỉ áp dụng ñược cho

một số lớp bài toán nhất ñịnh, chứ không tồn tại một phương pháp vạn năng ñể

thiết kế thuật toán giải quyết mọi bài toán. Các phương pháp thiết kế thuật toán

trên chỉ là chiến lược, có tính ñịnh hướng tìm thuật toán. Việc áp dụng chiến lược

ñể tìm ra thuật toán cho một bài toán cụ thể còn ñòi hỏi nhiều sáng tạo. Trong

chuyên ñề này, ngoài phần trình bày về các phương pháp, chuyên ñề còn có

những ví dụ cụ thể, cùng với thuật giải và cài ñặt, ñể có cái nhìn chi tiết từ việc

thiết kế giải thuật ñến xây dựng chương trình.

1. Quay lui (Backtracking)

Quay lui, vét cạn, thử sai, duyệt … là một số tên gọi tuy không ñồng nghĩa nhưng

cùng chỉ một phương pháp trong tin học: tìm nghiệm của một bài toán bằng cách

xem xét tất cả các phương án có thể. ðối với con người phương pháp này thường

là không khả thi vì số phương án cần kiểm tra lớn. Tuy nhiên ñối với máy tính,

nhờ tốc ñộ xử lí nhanh, máy tính có thể giải rất nhiều bài toán bằng phương pháp

quay, lui vét cạn.

Ưu ñiểm của phương pháp quay lui, vét cạn là luôn ñảm bảo tìm ra nghiệm ñúng,

chính xác. Tuy nhiên, hạn chế của phương pháp này là thời gian thực thi lâu, ñộ

phức tạp lớn. Do ñó vét cạn thường chỉ phù hợp với các bài toán có kích thước

nhỏ.

1.1. Phương pháp

Trong nhiều bài toán, việc tìm nghiệm có thể quy về việc tìm vector hữu hạn

, , … , \_x0007\_, … , ñộ dài vector có thể xác ñịnh trước hoặc không. Vector này cần

phải thoả mãn một số ñiều kiện tùy thuộc vào yêu cầu của bài toán. Các thành

phần

ñược chọn ra từ tập hữu hạn

.

60

Tuỳ từng trường hợp mà bài toán có thể yêu cầu: tìm một nghiệm, tìm tất cả

nghiệm hoặc ñếm số nghiệm.

Ví dụ: Bài toán 8 quân hậu.

Cần ñặt 8 quân hậu vào bàn cờ vua 8 8, sao cho

chúng không tấn công nhau, tức là không có hai

quân hậu nào cùng hàng, cùng cột hoặc cùng ñường

chéo.

Ví dụ: hình bên là một cách ñặt hậu thoả mãn yêu

cầu bài toán, các ô ñược tô màu là vị trí ñặt hậu.

Do các quân hậu phải nằm trên các hàng khác nhau, ta ñánh số các quân hậu từ 1

ñến 8, quân hậu i là quân hậu nằm trên hàng thứ i (i=1,2,…,8). Gọi

là cột mà

quân hậu i ñứng. Như vậy nghiệm của bài toán là vector , , … ,

, trong ñó

1

8, tức là

ñược chọn từ tập

\_x0010\_ 1,2, … ,8. Vector , , … ,

là

nghiệm nếu

và hai ô \_x0016\_,

, , \_x0019\_ không nằm trên cùng một ñường chéo.

Ví dụ: (1,5,8,6,3,7,2,4) là một nghiệm.

Tư tưởng của phương pháp quay lui vét cạn như sau: Ta xây dựng vector nghiệm

dần từng bước, bắt ñầu từ vector không ( ). Thành phần ñầu tiên ñược chọn ra

từ tập \_x0010\_

. Giả sử ñã chọn ñược các thành phần , , … , thì từ các ñiều

kiện của bài toán ta xác ñịnh ñược tập

(các ứng cử viên có thể chọn làm thành

phần

,

là tập con của

). Chọn một phần tử

từ

ta mở rộng nghiệm ñược

, , … ,

. Lặp lại quá trình trên ñể tiếp tục mở rộng nghiệm. Nếu không thể

chọn ñược thành phần ( rỗng) thì ta quay lại chọn một phần tử khác của

cho

. Nếu không còn một phần tử nào khác của

ta quay lại chọn một phần

tử khác của làm và cứ thế tiếp tục. Trong quá trình mở rộng nghiệm, ta

phải kiểm tra nghiệm ñang xây dựng ñã là nghiệm của bài toán chưa. Nếu chỉ cần

tìm một nghiệm thì khi gặp nghiệm ta dừng lại. Còn nếu cần tìm tất cả các nghiệm

thì quá trình chỉ dừng lại khi tất cả các khả năng lựa chọn của các thành phần của

vector nghiệm ñã bị vét cạn.

Lược ñồ tổng quát của thuật toán quay lui vét cạn có thể biểu diễn bởi thủ tục

\_x001D\_‑­ !"‑­ sau:

procedure Backtrack;

begin

S1:=A1;

k:=1;

while k>0 do begin

61

while Sk <> # do begin

<chọn xk $ Si>;

Sk:=Sk – {xk};

if (x1, x2,…,xk) là nghiệm then <ðưa ra nghiệm>;

k:=k+1;

<Xác ñịnh Sk>;

end;

k:=k-1; // quay lui

end;

end;

Trên thực tế, thuật toán quay lui vét cạn thường ñược dùng bằng mô hình ñệ quy

như sau:

procedure Backtrack(i);// xây dựng thành phần thứ i

begin

<Xác ñịnh Si>;

for xi $ Si do begin

<ghi nhận thành phần thứ i>;

if (tìm thấy nghiệm) then <ðưa ra nghiệm>

else Backtrack(i+1);

<loại thành phần i>;

end;

end;

Khi áp dụng lược ñồ tổng quát của thuật toán quay lui cho các bài toán cụ thể, có

ba vấn ñề quan trọng cần làm:

- Tìm cách biểu diễn nghiệm của bài toán dưới dạng một dãy các ñối

tượng ñược chọn dần từng bước , , … ,

, … .

- Xác ñịnh tập

các ứng cử viên ñược chọn làm thành phần thứ i của

nghiệm. Chọn cách thích hợp ñể biểu diễn

.

- Tìm các ñiều kiện ñể một vector ñã chọn là nghiệm của bài toán.

1.2. Một số ví dụ áp dụng

1.2.1. Tổ hợp

Một tổ hợp chập k của n là một tập con k phần tử của tập n phần tử.

Chẳng hạn tập {1,2,3,4} có các tổ hợp chập 2 là:

{1,2}, {1,3, {1,4, {2,3}, {2,4}, {3,4}

62

Vì trong tập hợp các phần tử không phân biệt thứ tự nên tập {1,2} cũng là tập

{2,1}, do ñó, ta coi chúng chỉ là một tổ hợp.

Bài toán: Hãy xác ñịnh tất cả các tổ hợp chập k của tập n phần tử. ðể ñơn giản ta

chỉ xét bài toán tìm các tổ hợp của tập các số nguyên từ 1 ñến n. ðối với một tập

hữu hạn bất kì, bằng cách ñánh số thứ tự của các phần tử, ta cũng ñưa ñược về bài

toán ñối với tập các số nguyên từ 1 ñến n.

Nghiệm của bài toán tìm các tổ hợp chập k của n phần tử phải thoả mãn các ñiều

kiện sau:

- Là một vector % \_x0010\_ , , … , &

-

lấy giá trị trong tập 1,2, … '

- Ràng buộc: ( với mọi giá trị i từ 1 ñến ) 1 (vì tập hợp không

phân biệt thứ tự phần tử nên ta sắp xếp các phần tử theo thứ tự tăng

dần).

Ta có: 1

( ( \* ( &

', do ñó tập

(tập các ứng cử viên ñược chọn

làm thành phần thứ i) là từ + 1 ñến ' ) + \_x0016\_. ðể ñiều này ñúng cho cả

trường hợp \_x0016\_ \_x0010\_ 1, ta thêm vào , \_x0010\_ 0.

Sau ñây là chương trình hoàn chỉnh, chương trình sử dụng mô hình ñệ quy ñể sinh

tất cả các tổ hợp chập của '.

program ToHop;

const MAX =20;

type vector =array[0..MAX]of longint;

var x :vector;

n,k :longint;

procedure GhiNghiem(x:vector);

var i :longint;

begin

for i:=1 to k do write(x[i],' ');

writeln;

end;

procedure ToHop(i:longint);

var j:longint;

begin

for j := x[i-1]+1 to n-k+i do begin

x[i] := j;

if i=k then GhiNghiem(x)

else ToHop(i+1);

63

end;

end;

BEGIN

write('Nhap n, k:'); readln(n,k);

x[0]:=0;

ToHop(1);

END.

Ví dụ về Input / Output của chương trình:

n = 4, k=2 1. 1 2

2. 1 3

3. 1 4

4. 2 3

5. 2 4

6. 3 4

Theo công thức, số lượng tổ hợp chập k=2 của n=4 là:

.\_x0007\_

& \_x0010\_

'' ) 1 … ' ) + 1

! \_x0010\_

'!

! ' ) !

\_x0010\_ .1

\_x0010\_ 6

1.2.2. Chỉnh hợp lặp

Chỉnh hợp lặp chập k của n là một dãy k thành phần, mỗi thành phần là một phần

tử của tập n phần tử, có xét ñến thứ tự và không yêu cầu các thành phần khác

nhau.

Một ví dụ dễ thấy nhất của chỉnh hợp lặp là các dãy nhị phân. Một dãy nhị phân

ñộ dài m là một chỉnh hợp lặp chập m của tập 2 phần tử {0,1}. Các dãy nhị phân

ñộ dài 3:

000, 001, 010, 011, 100, 101, 110, 111.

Vì có xét thứ tự nên dãy 101 và dãy 011 là 2 dãy khác nhau.

Như vậy, bài toán xác ñịnh tất cả các chỉnh hợp lặp chập k của tập n phần tử yêu

cầu tìm các nghiệm như sau:

- Là một vector % \_x0010\_ , , … , &

-

lấy giá trị trong tập 1,2, … '

- Không có ràng buộc nào giữa các thành phần.

64

Chú ý là cũng như bài toán tìm tổ hợp, ta chỉ xét ñối với tập n số nguyên từ 1 ñến

n. Nếu phải tìm chỉnh hợp không phải là tập các số nguyên từ 1 ñến n thì ta có thể

ñánh số các phần tử của tập ñó ñể ñưa về tập các số nguyên từ 1 ñến n.

{sử dụng một mảng x[1..n] ñể biểu diễn chỉnh hợp lặp.

Thủ tục ñệ quy sau sinh tất cả chỉnh hợp lặp chập k của n}

procedure ChinhHopLap(i:longint);

var j:longint;

begin

for j := 1 to n do begin

x[i] := j;

if i=k then GhiNghiem(x)

else ChinhHopLap(i+1);

end;

end;

Ví dụ về Input/Output của chương trình:

n = 2, k=3 1. 1 1 1

2. 1 1 2

3. 1 2 1

4. 1 2 2

5. 2 1 1

6. 2 1 2

7. 2 2 1

8. 2 2 2

Theo công thức, số lượng chỉnh hợp lặp chập k=3 của n=2 là:

3

\_x0007\_

& \_x0010\_ '& \_x0010\_ 24 \_x0010\_ 8

1.2.3. Chỉnh hợp không lặp

Khác với chỉnh hợp lặp là các thành phần ñược phép lặp lại (tức là có thể giống

nhau), chỉnh hợp không lặp chập k của tập n (k

n) phần tử cũng là một dãy k

thành phần lấy từ tập n phần tử có xét thứ tự nhưng các thành phần không ñược

phép giống nhau.

Ví dụ: Có n người, một cách chọn ra k người ñể xếp thành một hàng là một chỉnh

hợp không lặp chập k của n.

Một trường hợp ñặc biệt của chỉnh hợp không lặp là hoán vị. Hoán vị của một tập

n phần tử là một chỉnh hợp không lặp chập n của n. Nói một cách trực quan thì

65

hoán vị của tập n phần tử là phép thay ñổi vị trí của các phần tử (do ñó mới gọi là

hoán vị).

Nghiệm của bài toán tìm các chỉnh hợp không lặp chập k của tập n số nguyên từ 1

ñến n là các vector % thoả mãn các ñiều kiện:

- % có k thành phần: % \_x0010\_ , , … , &

-

lấy giá trị trong tập 1,2, … '

- Ràng buộc: các giá trị

ñôi một khác nhau, tức là

với mọi \_x0016\_ .

Sau ñây là chương trình hoàn chỉnh, chương trình sử dụng mô hình ñệ quy ñể sinh

tất cả các chỉnh hợp không lặp chập của ' phần tử.

program ChinhHopKhongLap;

const MAX =20;

type vector =array[0..MAX]of longint;

var x :vector;

d :array[1..MAX]of longint; { mảng d ñể kiểm

soát ràng buộc các giá trị

ñôi một khác nhau,

với mọi \_x0016\_ }

n,k :longint;

procedure GhiNghiem(x:vector);

var i :longint;

begin

for i:=1 to k do write(x[i],' ');

writeln;

end;

procedure ChinhHopKhongLap(i:longint);

var j:longint;

begin

for j := 1 to n do

if d[j]=0 then begin

x[i] := j;

d[j] := 1;

if i=k then GhiNghiem(x)

else ChinhHopKhongLap(i+1);

d[j] := 0;

end;

end;

BEGIN

write('Nhap n, k(k<=n):'); readln(n,k);

66

fillchar(d,sizeof(d),0);

ChinhHopKhongLap(1);

END.

Ví dụ về Input / Output của chương trình:

n = 3, k=3 1. 1 2 3

2. 1 3 2

3. 2 1 3

4. 2 3 1

5. 3 1 2

6. 3 2 1

Theo công thức, số lượng chỉnh hợp không lặp chập k=3 của n=3 là:

\_x0007\_

& \_x0010\_ '' ) 1 … ' ) + 1 \_x0010\_

'!

' ) !

\_x0010\_ 6

1.2.4. Bài toán xếp 8 quân hậu

Trong bài toán 8 quân hậu, nghiệm của bài toán có thể biểu diễn dưới dạng vector

, , … ,

thoả mãn:

1)

là tọa ñộ cột của quân hậu ñang ñứng ở dòng thứ \_x0016\_, $ 1,2, … ,8.

2) Các quân hậu không ñứng cùng cột tức là

với \_x0016\_ .

3) Có thể dễ dàng nhận ra rằng hai ô (x1,y1) và

(x2,y2) nằm trên cùng ñường chéo chính (trên xuống

dưới) nếu: x1−y1=x2−y2, hai ô (x1,y1) và (x2,y2) nằm

trên cùng ñường chéo phụ (từ dưới lên trên) nếu:

x1+y1=x2+y2, nên ñiều kiện ñể hai quân hậu xếp ở

hai ô \_x0016\_,

, , \_x0019\_ không nằm trên cùng một ñường

chéo là: 5

\_x0016\_ ) )

\_x0016\_ + +

Do ñó, khi ñã chọn ñược , , … , & thì & ñược chọn phải thoả mãn các

ñiều kiện:

7

&

) & \_x0016\_ )

+ & \_x0016\_ +

với mọi 1

\_x0016\_ (

67

Sau ñây là chương trình ñầy ñủ, ñể liệt kê tất cả các cách xếp 8 quân hậu lên bàn

cờ vua 8×8.

program XepHau;

type vector =array[1..8]of longint;

var x :vector;

procedure GhiNghiem(x:vector);

var i :longint;

begin

for i:=1 to 8 do write(x[i],' ');

writeln;

end;

procedure XepHau(k:longint);

var Sk :array[1..8]of longint;

xk,i,nSk :longint;

ok :boolean;

begin

{Xác ñịnh tập Sk

là tập các ứng cử viên có thể chọn làm thành phần xk}

nSk:=0;{lực lượng của tập Sk}

for xk:=1 to 8 do {thử lần lượt từng giá trị 1, 2, …,8}

begin

ok:=true;

{kiểm tra giá trị có thể chọn làm ứng cử viên cho xk

ñược hay không}

for i:=1 to k-1 do

if not((xk<>x[i])and(k-xk<>i-x[i])

and(k+xk<>i+x[i])) then

begin

ok:=false;

break;

end;

if ok then begin {có thể chọn làm ứng cử viên cho xk

,kết nạp

vào tập Sk}

inc(nSk);

Sk[nSk]:=xk;

end;

end;

{chọn giá trị xk

từ tập Sk}

for i:=1 to nSk do begin

x[k]:=Sk[i];

68

if k=8 then GhiNghiem(x)

else XepHau(k+1);

x[k]:=0;

end;

end;

BEGIN

XepHau(1);

END.

Việc xác ñịnh tập Sk có thể thực hiện ñơn giản và hiệu quả hơn bằng cách sử dụng

các mảng ñánh dấu. Cụ thể, khi ta ñặt hậu i ở ô (i,x[i]), ta sẽ ñánh dấu cột x[i]

(dùng một mảng ñánh dấu như ở bài toán chỉnh hợp không lặp), ñánh dấu ñường

chéo chính (i-x[i]) và ñánh dấu ñường chéo phụ (i+x[i]).

const n =8;

type vector =array[1..n]of longint;

var cot :array[1..n]of longint;

cheoChinh :array[1-n..n-1]of longint;

cheoPhu :array[1+1..n+n]of longint;

x :vector;

procedure GhiNghiem(x:vector);

var i :longint;

begin

for i:=1 to n do write(x[i],' ');

writeln;

end;

procedure xepHau(k:longint);

var i :longint;

begin

for i:=1 to n do

if (cot[i]=0) and (cheoChinh[k-i]=0) and

(cheoPhu[k+i]=0) then

begin

x[k]:=i;

cot[i]:=1;

cheoChinh[k-i]:=1;

CheoPhu[k+i]:=1;

if k=n then GhiNghiem(x)

else xepHau(k+1);

cot[i]:=0;

69

cheoChinh[k-i]:=0;

CheoPhu[k+i]:=0;

end;

end;

BEGIN

fillchar(cot,sizeof(cot),0);

fillchar(cheoChinh,sizeof(cheoChinh),0);

fillchar(cheoPhu,sizeof(cheoPhu),0);

xepHau(1);

END.

Bài toán xếp hậu có tất cả 92 nghiệm, mười nghiệm ñầu tiên mà chương trình tìm

ñược là:

1. 1 5 8 6 3 7 2 4

2. 1 6 8 3 7 4 2 5

3. 1 7 4 6 8 2 5 3

4. 1 7 5 8 2 4 6 3

5. 2 4 6 8 3 1 7 5

6. 2 5 7 1 3 8 6 4

7. 2 5 7 4 1 8 6 3

8. 2 6 1 7 4 8 3 5

9. 2 6 8 3 1 4 7 5

10. 2 7 3 6 8 5 1 4

1.2.5. Bài toán máy rút tiền tự ñộng ATM

Một máy ATM hiện có ' '

20 tờ tiền có giá !, !, … , !\_x0007\_. Hãy ñưa ra một

cách trả với số tiền ñúng bằng .

Dữ liệu vào từ file “ATM.INP” có dạng:

- Dòng ñầu là 2 số ' và

- Dòng thứ 2 gồm ' số !, !, … , !\_x0007\_

Kết quả ra file “ATM.OUT” có dạng: Nếu có thể trả ñúng thì ñưa ra cách trả,

nếu không ghi -1.

ATM.INP ATM.OUT

10 390

200 10 20 20 50 50 50 50 100 100

20 20 50 50 50 100 100

Nghiệm của bài toán là một dãy nhị phân ñộ dài ', trong ñó thành phần thứ i bằng

1 nếu tờ tiền thứ i ñược sử dụng ñể trả, bằng 0 trong trường hợp ngược lại.

70

% \_x0010\_ , , … , \_x0007\_ là nghiệm nếu: ! + ! + \* + \_x0007\_ !\_x0007\_ \_x0010\_

Trong chương trình dưới ñây có sử dụng một biến ok ñể kiểm soát việc tìm

nghiệm. Ban ñầu chưa có nghiệm, do ñó khởi trị ok=FALSE. Khi tìm ñược

nghiệm, ok sẽ ñược nhận giá trị bằng TRUE. Nếu ok=TRUE (ñã tìm thấy nghiệm)

ta sẽ không cần tìm kiếm nữa.

const MAX =20;

fi ='ATM.INP';

fo ='ATM.OUT';

type vector =array[1..MAX]of longint;

var t :array[1..MAX]of longint;

x,xs :vector;

n,s,sum :longint;

ok :boolean;

procedure input;

var f :text;

i :longint;

begin

assign(f,fi); reset(f);

readln(f,n, s);

for i:=1 to n do read(f,t[i]);

close(f);

end;

procedure check(x:vector);

var i :longint;

f :text;

begin

if sum = s then begin

xs:=x;

ok:=true;

end;

end;

procedure printResult;

var i :longint;

f :text;

begin

assign(f,fo); rewrite(f);

if ok then begin

for i:=1 to n do

71

if xs[i]=1 then write(f,t[i],' ');

end

else write(f,'-1');

close(f);

end;

procedure backTrack(i:longint);

var j :longint;

begin

for j:=0 to 1 do begin

x[i]:=j;

sum:=sum + x[i]\*t[i];

if (i=n) then check(x)

else if sum<=s then backTrack(i+1);

if ok then exit; {nếu ñã tìm ñược nghiệm thì không duyệt nữa}

sum:=sum - x[i]\*t[i];

end;

end;

BEGIN

input;

ok:=false;

sum:=0;

backTrack(1);

PrintResult;

END.

2. Nhánh và cận

2.1. Phương pháp

Trong thực tế, có nhiều bài toán yêu cầu tìm ra một phương án thoả mãn một số

ñiều kiện nào ñó, và phương án ñó là tốt nhất theo một tiêu chí cụ thể. Các bài

toán như vậy ñược gọi là bài toán tối ưu. Có nhiều bài toán tối ưu không có thuật

toán nào thực sự hữu hiệu ñể giải quyết, mà cho ñến nay vẫn phải dựa trên mô

hình xem xét toàn bộ các phương án, rồi ñánh giá ñể chọn ra phương án tốt nhất.

Phương pháp nhánh và cận là một dạng cải tiến của phương pháp quay lui, ñược

áp dụng ñể tìm nghiệm của bài toán tối ưu.

Giả sử nghiệm của bài toán có thể biểu diễn dưới dạng một vector , , … , \_x0007\_

,

mỗi thành phần

\_x0016\_ \_x0010\_ 1,2, . . , ' ñược chọn ra từ tập

. Mỗi nghiệm của bài toán

72

% \_x0010\_ , , … , \_x0007\_

, ñược xác ñịnh “ñộ tốt” bằng một hàm 9% và mục tiêu cần

tìm nghiệm có giá trị 9% ñạt giá trị nhỏ nhất (hoặc ñạt giá trị lớn nhất).

Tư tưởng của phương pháp nhánh và cận như sau: Giả sử, ñã xây dựng ñược k

thành phần , , … , &

của nghiệm và khi mở rộng nghiệm , , … , &,

nếu biết rằng tất cả các nghiệm mở rộng của nó , , … , &, … ñều không tốt

bằng nghiệm tốt nhất ñã biết ở thời ñiểm ñó, thì ta không cần mở rộng từ

, , … , &

nữa. Như vậy, với phương pháp nhánh và cận, ta không phải duyệt

toàn bộ các phương án ñể tìm ra nghiệm tốt nhất mà bằng cách ñánh giá các

nghiệm mở rộng, ta có thể cắt bỏ ñi những phương án (nhánh) không cần thiết, do

ñó việc tìm nghiệm tối ưu sẽ nhanh hơn. Cái khó nhất trong việc áp dụng phương

pháp nhánh và cận là ñánh giá ñược các nghiệm mở rộng, nếu ñánh giá ñược tốt

sẽ giúp bỏ qua ñược nhiều phương án không cần thiết, khi ñó thuật toán nhánh cận

sẽ chạy nhanh hơn nhiều so với thuật toán vét cạn.

Thuật toán nhánh cận có thể mô tả bằng mô hình ñệ quy sau:

procedure BranchBound(i);// xây dựng thành phần thứ i

begin

<ðánh giá các nghiệm mở rộng>;

if(các nghiệm mở rộng ñều không tốt hơn

BestSolution)then exit;

<Xác ñịnh Si>;

for xi $ Si do begin

<ghi nhận thành phần thứ i>;

if (tìm thấy nghiệm) then <Cập nhật BestSolution>

else BranchBound(i+1);

<loại thành phần i>;

end;

end;

Trong thủ tục trên, BestSolution là nghiệm tốt nhất ñã biết ở thời ñiểm ñó. Thủ

tục <cập nhật BestSolution> sẽ xác ñịnh “ñộ tốt” của nghiệm mới tìm thấy,

nếu nghiệm mới tìm thấy tốt hơn BestSolution thì BestSolution sẽ ñược

cập nhật lại là nghiệm mới tìm ñược.

73

2.2. Giải bài toán người du lịch bằng phương pháp nhánh cận.

Bài toán. Cho n thành phố ñánh số từ 1 ñến n và các tuyến ñường giao thông hai

chiều giữa chúng, mạng lưới giao thông này ñược cho bởi mảng C[1..n,1..n], ở

ñây Cij = Cji là chi phí ñi ñoạn ñường trực tiếp từ thành phố i ñến thành phố j.

Một người du lịch xuất phát từ thành phố 1, muốn ñi thăm tất cả các thành phố

còn lại mỗi thành phố ñúng 1 lần và cuối cùng quay lại thành phố 1. Hãy chỉ ra

cho người ñó hành trình với chi phí ít nhất. Bài toán ñược gọi là bài toán người du

lịch hay bài toán người chào hàng (Travelling Salesman Problem - TSP)

Dữ liệu vào trong file “TSP.INP” có dạng:

- Dòng ñầu chứa số n(1<n≤20), là số thành phố.

- n dòng tiếp theo, mỗi dòng n số mô tả mảng C

Kết quả ra file “TSP.OUT” có dạng:

- Dòng ñầu là chi phí ít nhất

- Dòng thứ hai mô tả hành trình

Ví dụ 1:

TSP.INP TSP.OUT Hình minh họa

4

0 20 35 42

20 0 34 30

35 34 0 12

42 30 12 0

97

1->2->4->3->1

Ví dụ 2:

TSP.INP TSP.OUT Hình minh họa

4

0 20 35 10

20 0 90 50

35 90 0 12

10 50 12 0

117

1->2->4->3->1

74

Giải

1) Hành trình cần tìm có dạng (x1 = 1, x2, ..., xn, xn+1 = 1), ở ñây giữa xi

và

xi+1: hai thành phố liên tiếp trong hành trình phải có ñường ñi trực tiếp;

trừ thành phố 1, không thành phố nào ñược lặp lại hai lần, có nghĩa là

dãy (x1, x2, ..., xn) lập thành một hoán vị của (1, 2, ..., n).

2) Duyệt quay lui: x2 có thể chọn một trong các thành phố mà x1 có ñường

ñi trực tiếp tới, với mỗi cách thử chọn x2 như vậy thì x3 có thể chọn một

trong các thành phố mà x2 có ñường ñi tới (ngoài x1). Tổng quát: xi

có

thể chọn 1 trong các thành phố chưa ñi qua mà từ xi-1 có ñường ñi trực

tiếp tới.(2 ≤ i ≤ n).

3) Nhánh cận: Khởi tạo cấu hình BestSolution có chi phí = +∞. Với mỗi

bước thử chọn xi

xem chi phí ñường ñi cho tới lúc ñó có nhỏ hơn chi phí

của cấu hình BestSolution không? nếu không nhỏ hơn thì thử giá trị

khác ngay bởi có ñi tiếp cũng chỉ tốn thêm. Khi thử ñược một giá trị xn

ta kiểm tra xem xn có ñường ñi trực tiếp về 1 không ? Nếu có ñánh giá

chi phí ñi từ thành phố 1 ñến thành phố xn cộng với chi phí từ xn ñi trực

tiếp về 1, nếu nhỏ hơn chi phí của ñường ñi BestSolution thì cập nhật lại

BestSolution bằng cách ñi mới.

program TSP;

const MAX =20;

oo =1000000;

fi ='TSP.INP';

fo ='TSP.OUT';

var c :array[1..MAX,1..MAX]of longint;

x,bestSolution :array[1..MAX]of longint;

d :array[1..MAX]of longint;

n :longint;

sum,best :longint;

procedure input;

var f :text;

i,j,k :longint;

begin

assign(f,fi); reset(f);

read(f,n);

for i:=1 to n do

for j:=1 to n do read(f,C[i,j]);

close(f);

75

end;

procedure update;

begin

if sum+C[x[n],x[1]]<best then begin

best:=sum+C[x[n],x[1]];

bestSolution:=x;

end;

end;

procedure branchBound(i:longint);

var j :longint;

begin

if sum>=best then exit;

for j:=1 to n do

if d[j]=0 then begin

x[i]:=j;

d[j]:=1;

sum:=sum + C[x[i-1],j];

if i=n then update

else branchBound(i+1);

sum:=sum - C[x[i-1],j];

d[j]:=0;

end;

end;

procedure init;

begin

fillchar(d,sizeof(d),0);

d[1]:=1;

x[1]:=1;

best:=oo;

end;

procedure output;

var f :text;

i :longint;

begin

assign(f,fo); rewrite(f);

writeln(f,best);

for i:=1 to n do write(f,bestSolution[i],'->');

write(f,bestSolution[1]);

close(f);

76

end;

BEGIN

input;

init;

branchBound(2);

output;

END.

Chương trình trên là một giải pháp nhánh cận rất thô sơ giải bài toán TSP, có thể

có nhiều cách ñánh giá nhánh cận chặt hơn nữa làm tăng hiệu quả của chương

trình.

2.3. Bài toán máy rút tiền tự ñộng ATM

Bài toán

Một máy ATM hiện có ' '

20 tờ tiền có giá !, !, … , !\_x0007\_. Hãy tìm cách trả ít

tờ nhất với số tiền ñúng bằng .

Dữ liệu vào từ file “ATM.INP” có dạng:

- Dòng ñầu là 2 số ' và

- Dòng thứ 2 gồm ' số !, !, … , !\_x0007\_

Kết quả ra file “ATM.OUT” có dạng: Nếu có thể trả tiền ñúng bằng thì ñưa ra

số tờ ít nhất cần trả và ñưa ra cách trả, nếu không ghi -1.

ATM.INP ATM.OUT

10 390

200 10 20 20 50 50 50 50 100 100

5

20 20 50 100 200

Giải

Như ta ñã biết, nghiệm của bài toán là một dãy nhị phân ñộ dài ', giả sử ñã xây

dựng ñược thành phần , , … , &, ñã trả ñược :;< và sử dụng ­ tờ. ðể

ñánh giá ñược các nghiệm mở rộng của , , … , &, ta nhận thấy:

- Còn phải trả ) :;<

- Gọi !<‑= > là giá cao nhất trong các tờ tiền còn lại (!<‑= > \_x0010\_

?

%!&, . . , !\_x0007\_ thì ít nhất cần sử dụng thêm @ABC

DCEF=&>

tờ nữa.

Do ñó, nếu ­ + @ABC

DCEF=&>

mà lớn hơn hoặc bằng số tờ của cách trả tốt nhất hiện có

thì không cần mở rộng các nghiệm của , , … , & nữa.

77

const MAX =20;

fi ='ATM.INP';

fo ='ATM.OUT';

type vector =array[1..MAX]of longint;

var t,tmax :array[1..MAX]of longint;

x,xbest :vector;

c,cbest :longint;

n,s,sum :longint;

procedure input;

var f :text;

i :longint;

begin

assign(f,fi); reset(f);

readln(f,n, s);

for i:=1 to n do read(f,t[i]);

close(f);

end;

procedure init;

var i :longint;

begin

tmax[n]:=t[n];

for i:=n-1 downto 1 do begin

tmax[i]:=tmax[i+1];

if tmax[i]<t[i] then tmax[i]:=t[i];

end;

sum:=0;

c:=0;

cbest:=n+1;

end;

procedure update;

var i :longint;

f :text;

begin

if (sum = s) and (c<cbest) then begin

xbest:=x;

cbest:=c;

end;

end;

procedure printResult;

78

var i :longint;

f :text;

begin

assign(f,fo); rewrite(f);

if cbest<n+1 then begin

writeln(f,cbest);

for i:=1 to n do

if xbest[i]=1 then write(f,t[i],' ');

end

else write(f,'-1');

close(f);

end;

procedure branchBound(i:longint);

var j :longint;

begin

if c + (s-sum)/tmax[i] >= cbest then exit;

for j:=0 to 1 do begin

x[i]:=j;

sum:=sum + x[i]\*t[i];

c:=c + j;

if (i=n) then update

else if sum<=s then branchBound(i+1);

sum:=sum - x[i]\*t[i];

c:=c - j;

end;

end;

BEGIN

input;

init;

branchBound(1);

PrintResult;

END.

3. Tham ăn (Greedy Method)

Phương pháp nhánh cận là cải tiến phương pháp quy lui, ñã ñánh giá ñược các

nghiệm mở rộng ñể loại bỏ ñi những phương án không cần thiết, giúp cho việc tìm

nghiệm tối ưu nhanh hơn. Tuy nhiên, không phải lúc nào chúng ta cũng có thể

ñánh giá ñược nghiệm mở rộng, hoặc nếu có ñánh giá ñược thì số phương án cần

79

xét vẫn rất lớn, không thể ñáp ứng ñược trong thời gian cho phép. Khi ñó, người

ta chấp nhận tìm những nghiệm gần ñúng so với nghiệm tối ưu. Phương pháp

tham ăn ñược sử dụng trong các trường hợp như vậy. Ưu ñiểm nổi bật của phương

pháp tham ăn là ñộ phức tạp nhỏ, thường nhanh chóng tìm ñược lời giải.

3.1. Phương pháp

Giả sử nghiệm của bài toán có thể biểu diễn dưới dạng một vector , , … , \_x0007\_

,

mỗi thành phần

\_x0016\_ \_x0010\_ 1,2, . . , ' ñược chọn ra từ tập

. Mỗi nghiệm của bài toán

% \_x0010\_ , , … , \_x0007\_

, ñược xác ñịnh “ñộ tốt” bằng một hàm 9% và mục tiêu cần

tìm nghiệm có giá trị 9% càng lớn càng tốt (hoặc càng nhỏ càng tốt).

Tư tưởng của phương pháp tham ăn như sau: Ta xây dựng vector nghiệm X dần

từng bước, bắt ñầu từ vector không ( ). Giả sử ñã xây dựng ñược (k-1) thành phần

, , … , & của nghiệm và khi mở rộng nghiệm ta sẽ chọn &”tốt nhất”

trong các ứng cử viên trong tập & ñể ñược , , … , &

. Việc lựa chọn như thế

ñược thực hiện bởi một hàm chọn. Cứ tiếp tục xây dựng, cho ñến khi xây dựng

xong hết thành phần của nghiệm.

Lược ñồ tổng quát của phương pháp tham ăn.

procedure Greedy;

begin

X:=#;

i:=0;

while (chưa xây dựng xong hết thành phần của nghiệm) do

begin

i:=i+1;

<Xác ñịnh Si>;

xGselect(Si);// chọn ứng cử viên tốt nhất trong tập Si

end;

end;

Trong lược ñồ tổng quát trên, Select là hàm chọn, ñể chọn ra từ tập các ứng cử

viên Si

một ứng cử viên ñược xem là tốt nhất, nhiều hứa hẹn nhất.

Cần nhấn mạnh rằng, thuật toán tham ăn trong một số bài toán, nếu xây dựng

ñược hàm thích hợp có thể cho nghiệm tối ưu. Trong nhiều bài toán, thuật toán

tham ăn chỉ tìm ñược nghiệm gần ñúng với nghiệm tối ưu.

3.2. Bài toán người du lịch

(Bài toán ở mục 2.2)

80

Có nhiều thuật toán tham ăn cho bài này, một thuật toán với ý tưởng ñơn giản như

sau: Xuất phát từ thành phố 1, tại mỗi bước ta sẽ chọn thành phố tiếp theo là thành

phố chưa ñến thăm mà chi phí từ thành phố hiện tại ñến thành phố ñó là nhỏ nhất,

cụ thể:

+ Hành trình cần tìm có dạng (x1 = 1, x2, ..., xn, xn+1 = 1), trong ñó dãy (x1,

x2, ..., xn) lập thành một hoán vị của (1, 2, ..., n).

+ Ta xây dựng nghiệm từng bước, bắt ñầu từ x1=1, chọn x2 là thành phố gần

x1 nhất, sau ñó chọn x3 là thành phố gần x2 nhất (x3 khác x1)... Tổng quát:

chọn xi

là thành phố chưa ñi qua mà gần xi-1 nhất.(2 ≤ i ≤ n).

program TSP;

const MAX =100;

oo =1000000;

fi ='TSP.INP';

fo ='TSP.OUT';

var c :array[1..MAX,1..MAX]of

longint;

x :array[1..MAX]of longint;

d :array[1..MAX]of longint;

n :longint;

sum :longint;

procedure input;

var f :text;

i,j,k :longint;

begin

assign(f,fi); reset(f);

read(f,n);

for i:=1 to n do

for j:=1 to n do read(f,C[i,j]);

close(f);

end;

procedure output;

var f :text;

i :longint;

begin

assign(f,fo); rewrite(f);

writeln(f,sum);

for i:=1 to n do write(f,x[i],'->');

write(f,x[1]);

81

close(f);

end;

procedure Greedy;

var i,j,xi :longint;

best :longint;

begin

x[1]:=1;

d[1]:=1;

i:=1;

while i<n do begin

inc(i);

// chọn ứng cử viên tốt nhất

best:=oo;

for j:=1 to n do

if (d[j]=0) and (c[x[i-1],j]<best) then begin

best:=c[x[i-1],j];

xi:=j;

end;

x[i]:=xi; //ghi nhận thành phần nghiệm thứ i

d[xi]:=1;

sum:=sum+c[x[i-1],x[i]];

end;

sum:=sum+c[x[n],x[1]];

end;

BEGIN

input;

Greedy;

output;

END.

Ví dụ 1. Xuất phát từ thành phố 1, ta xây dựng ñược hành trình 1 2 4 31

với chi phí 97, ñây là phương án tối ưu.

82

Ví dụ 2. Xuất phát từ thành phố 1, ta xây dựng ñược hành trình 1 4 3 21

với chi phí 132, nhưng kết quả tối ưu là 117.

3.3. Bài toán máy rút tiền tự ñộng ATM

(bài toán ở mục 2.3)

Thuật toán với ý tưởng tham ăn ñơn giản, hàm chọn như sau: Tại mỗi bước ta sẽ

chọn tờ tiền lớn nhất còn lại không vượt quá lượng tiền còn phải trả, cụ thể:

- Sắp xếp các tờ tiền giảm dần theo giá trị.

- Lần lượt xét các tờ tiền từ giá trị lớn ñến giá trị nhỏ, nếu vẫn còn chưa lấy

ñủ và tờ tiền ñang xét có giá trị nhỏ hơn hoặc bằng thì lấy luôn tờ tiền

ñó.

const MAX =100;

fi ='ATM.INP';

fo ='ATM.OUT';

type vector =array[1..MAX]of longint;

var t :array[1..MAX]of longint;

x :vector;

c :longint;

n,s :longint;

procedure input;

var f :text;

i :longint;

begin

assign(f,fi); reset(f);

readln(f,n, s);

for i:=1 to n do read(f,t[i]);

close(f);

end;

procedure greedy;

var i,j :longint;

83

tmp :longint;

begin

fillchar(x,sizeof(x),0);

{sắp xếp các tờ theo giá trị giảm dần}

for i:=1 to n-1 do

for j:=i+1 to n do

if t[i]<t[j] then begin

tmp:=t[i];

t[i]:=t[j];

t[j]:=tmp;

end;

c:=0;

for i:=1 to n do

if s>=t[i] then

begin

inc(c); {số lượng tờ lấy}

x[i]:=1; {tờ i ñược lấy}

s:=s-t[i];

end;

end;

procedure printResult;

var i :longint;

f :text;

begin

assign(f,fo); rewrite(f);

if s=0 then begin

writeln(f,c);

for i:=1 to n do

if x[i]=1 then write(f,t[i],' ');

end

else write(f,'-1'); {nếu không lấy ñược ñủ S, S>0}

close(f);

end;

BEGIN

input;

greedy;

PrintResult;

END.

84

Các bộ test thử nghiệm

test Dữ liệu vào Kết quả tìm ñược

1 10 390

200 10 20 20 50 50 50 50 100 100

5

200 100 50 20 20

2 11 100

50 20 20 20 20 20 2 2 2 2 2

8

50 20 20 2 2 2 2 2

3 6 100

50 20 20 20 20 20

-1

Với bộ test (1), thuật toán tham ăn cũng cho ñược nghiệm tối ưu.Tuy nhiên, với

bộ test (2), thuật toán tham ăn không cho nghiệm tối ưu và với bộ test (3), thuật

toán tham ăn không tìm nghiệm mặc dù có nghiệm.

3.4. Bài toán lập lịch giảm thiểu trễ hạn

Bài toán:

Có ' công việc ñánh số từ 1 ñến ' và có một máy ñể thực hiện, biết:

- H

là thời gian cần thiết ñể hoàn thành công việc \_x0016\_.

- I

là thời hạn hoàn thành công việc \_x0016\_.

Máy bắt ñầu hoạt ñộng từ thời ñiểm 0. Mỗi công việc cần ñược thực hiện liên tục

từ lúc bắt ñầu cho tới khi kết thúc, không ñược phép ngắt quãng. Giả sử ­

là thời

ñiểm hoàn thành công việc \_x0016\_. Khi ñó, nếu ­

> I

ta nói công việc \_x0016\_ bị hoàn thành

trễ hạn, còn nếu ­

I

thì ta nói công việc \_x0016\_ ñược hoàn thành ñúng hạn.

Yêu cầu: Tìm trình tự thực hiện các công việc sao cho số công việc hoàn thành trễ

hạn là ít nhất (hay số công việc hoàn thành ñúng hạn là nhiều nhất).

Dữ liệu vào trong file “JS.INP” có dạng:

- Dòng ñầu là số ' '

100 là số công việc

- Dòng thứ hai gồm ' số là thời gian thực hiện các công việc

- Dòng thứ ba gồm ' số là thời hạn hoàn thành các công việc

Kết quả file “JS.OUT” có dạng: gồm một dòng là trình tự thực hiện các công

việc.

Ví dụ: giả sử có 5 công việc với thời gian thực hiện và thời gian hoàn thành

như sau:

\_x0016\_ 1 2 3 4 5

H 6 3 5 7 2

85

I 8 4 15 20 3

Nếu thực hiện theo thứ tự 1, 2, 3, 4, 5 thì sẽ có 3 công việc bị trễ hạn là công việc

2, 4 và 5. Còn nếu thực hiện theo thứ tự 5, 1, 3, 4, 2 thì chỉ có 1 công việc bị trễ

hạn là công việc 2, ñây là thứ tự thực hiện mà số công việc bị trễ hạn ít nhất

(nghiệm tối ưu).

Giải

Ta có hai nhận xét sau:

+ Nếu thứ tự thực hiện các công việc mà có công việc bị trễ hạn ñược xếp trước

một công việc ñúng hạn thì ta sẽ nhận ñược trình tự tốt hơn bằng cách chuyển

công việc trễ hạn xuống cuối cùng (vì ñằng nào công việc này cũng bị trễ hạn).

Ví dụ: thứ tự 1, 2, 3, 4, 5 có công việc 2 bị trễ hạn xếp trước công việc 3 ñúng

hạn, ta chuyển công việc 2 xuống cuối cùng ñể nhận ñược thứ tự: 1, 3, 4, 5, 2, thứ

tự này chỉ có 2 công việc bị quá hạn là công việc 5 và 2.

Như vậy, ta chỉ quan tâm ñến việc xếp lịch cho các công việc hoàn thành ñúng

hạn, còn các công việc bị trễ hạn có thể thực hiện theo trình tự bất kì.

+ Giả sử J: là tập gồm công việc (mà cả công việc này ñều có thể thực hiện

ñúng hạn) và K \_x0010\_ \_x0016\_, \_x0016\_, . . , \_x0016\_& là một hoán vị của các công việc trong J: sao cho

I L

I M

\*

I N

thì thứ tự K là thứ tự ñể hoàn thành ñúng hạn ñược cả

công việc.

Ví dụ: J: gồm 4 công việc 1, 3, 4, 5 (4 công việc này ñều có thể thực hiện ñúng

hạn), ta có thứ tự thực hiện K \_x0010\_ 5, 1, 3, 4 vì IR \_x0010\_ 3

I \_x0010\_ 8

I4 \_x0010\_ 15

I1 \_x0010\_ 20 ñể cả 4 công việc ñều thực hiện ñúng hạn.

Sử dụng chiến lược tham ăn, ta xây dựng tập công việc J: theo từng bước, ban

ñầu J: \_x0010\_ #. Hàm chọn ñược xây dựng như sau: tại mỗi bước ta sẽ chọn công việc

JST mà có thời gian thực hiện nhỏ nhất trong số các công việc còn lại cho vào tập

J:. Nếu sau khi kết nạp JST

, các công việc trong tập J: ñều có thể thực hiện ñúng

hạn thì cố ñịnh việc kết nạp JST

vào tập J:, nếu không thì không kết nạp JST

. ðể

ñơn giản, ta giả sử rằng các công việc ñược ñánh số theo thứ tự thời gian thực

hiện tăng dần H

H

\*

H\_x0007\_. Ta có lược ñồ thuật toán tham ăn như sau:

procedure JobScheduling;

begin

J: U #;

for i:=1 to n do

86

if các công việc trong tập (J: W JST

hoàn thành ñúng

hạn then

J: U J: W \_x0016\_;

for i:=1 to n do

if JST X J: then J: W JST

;

end;

Sau ñây là chương trình hoàn chỉnh:

const MAX =100;

fi ='js.inp';

fo ='js.out';

type TJob =record

p, d :longint;

name :longint;

end;

TArrJobs =array[1..MAX]of TJob;

var jobs,Js :TArrJobs;

d :array[1..MAX]of longint;

n,m :longint;

procedure input;

var f :text;

i :longint;

begin

assign(f,fi);

reset(f);

readln(f,n);

for i:=1 to n do read(f,jobs[i].p);

for i:=1 to n do read(f,jobs[i].d);

close(f);

for i:=1 to n do jobs[i].name:=i;

end;

procedure swap(var j1,j2:TJob);

var tmp :TJob;

begin

tmp:=j1;

j1:=j2;

j2:=tmp;

end;

function check(var Js:TArrJobs; nJob:longint):boolean;

87

var i,j :longint;

t :longint;

begin

for i:=1 to nJob-1 do

for j:=i+1 to nJob do

if Js[i].d>Js[j].d then swap(Js[i],Js[j]);

t:=0;

for i:=1 to nJob do begin

if t+Js[i].p>Js[i].d then exit(false);

t:=t+Js[i].p;

end;

exit(true);

end;

procedure Greedy;

var i,j :longint;

Js2 :TArrJobs;

begin

for i:=1 to n-1 do

for j:=i+1 to n do

if jobs[i].p > jobs[j].p then

swap(jobs[i],jobs[j]);

fillchar(d,sizeof(d),0);

m:=0;

for i:=1 to n do begin

Js2:=Js;

Js2[m+1]:=jobs[i];

if check(Js2,m+1) then begin

m:=m+1;

Js:=Js2;

d[i]:=1;

end;

end;

//writeln(m);

for i:=1 to n do

if d[i]=0 then begin

m:=m+1;

Js[m]:=jobs[i];

end;

end;

88

procedure printResult;

var f :text;

i :longint;

begin

assign(f,fo); rewrite(f);

for i:=1 to n do write(f,Js[i].name,' ');

close(f);

end;

BEGIN

input;

Greedy;

printResult;

END.

Chú ý: Thuật toán tham ăn trình bày trên luôn cho phương án tối ưu.

4. Chia ñể trị (Divide and Conquer)

4.1. Phương pháp

Tư tưởng của chiến lược chia ñể trị như sau: Người ta phân bài toán cần giải thành

các bài toán con. Các bài toán con lại ñược tiếp tục phân thành các bài toán con

nhỏ hơn, cứ thế tiếp tục cho tới khi ta nhận ñược các bài toán con hoặc ñã có thuật

giải hoặc là có thể dễ ràng ñưa ra thuật giải. Sau ñó ta tìm cách kết hợp các

nghiệm của các bài toán con ñể nhận ñược nghiệm của bài toán con lớn hơn, ñể

cuối cùng nhận ñược nghiệm của bài toán cần giải. Thông thường các bài toán con

nhận ñược trong quá trình phân chia là cùng dạng với bài toán ban ñầu, chỉ có cỡ

của chúng là nhỏ hơn.

Thuật toán chia ñể trị có thể biểu diễn bằng mô hình ñệ quy như sau:

procedure DivideConquer(A,x); // tìm nghiệm x của bài toán A

begin

if (A ñủ nhỏ) then Solve(A)

else begin

Phân A thành các bài toán con A1, A2, …, Am;

for i:=1 to m do DivideConquer(Ai, xi);

Kết hợp các nghiệm xi (i=1,2,..,m) của các bài toán

con Ai ñể nhận ñược nghiệm của bài toán A;

end;

end;

89

Trong thủ tục trên, Solve(A) là thuật giải bài toán A trong trường hợp A có cỡ ñủ

nhỏ.

Trong thuật toán tìm kiếm nhị phân và thuật toán sắp xếp nhanh-QuickSort (ở

chuyên ñề sắp xếp) là hai thuật toán ñược thiết kế dựa trên chiến lược chia ñể trị.

Sau ñây, chúng ta sẽ tìm hiểu một số ví dụ minh họa cho phương pháp chia ñể trị.

4.2. Bài toán tính Y

Z

Bài toán: Cho số ‑ và số nguyên dương ', tính ‑

\_x0007\_

Cách 1: Sử dụng thuật toán lặp, mất ' phép nhân ñể tính ‑

\_x0007\_

procedure power(a,n:longint;var p:longint);

{giá trị an

sẽ ñược lưu vào biến p}

var i : longint;

begin

p:=1;

for i:=1 to n do p:=p\*a;

end;

var a, n, p : longint;

BEGIN

write('Nhap a, n:'); readln(a, n);

power(a,n,p);

write(p);

END.

Cách 2: Áp dụng kĩ thuật chia ñể trị, ta tính ‑

\_x0007\_

dựa vào ‑

&

(trong ñó \_x0010\_ ' I\_x0016\_[ 2)

như sau:

- nếu ' chẵn: ‑

\_x0007\_ \_x0010\_ ‑& ‑&

- nếu n lẻ: ‑

\_x0007\_ \_x0010\_ ‑& ‑& ‑

ðể tính ‑

&

ta lại dựa vào ‑

& \ ] , quá trình chia nhỏ cho ñến khi nhận ñược bài

toán tính ‑

thì dừng.

Ví dụ: tính 913

- bài toán ñược tính dựa trên bài toán con 9

\_

, ta có 9

4 \_x0010\_ 9\_ 9`

- bài toán 96

ñược tính dựa trên bài toán con 93

, ta có 9

\_ \_x0010\_ 94 94

- bài toán 93

ñược tính dựa trên bài toán con 91

, ta có 9

4 \_x0010\_ 9 9 9

Thủ tục ñệ quy power(a,n,p) sau thể hiện ý tưởng trên.

procedure power(a,n:longint; var p:longint);

var tmp : longint;

90

begin

if (n=1) then p:=a

else begin

power(a,n div 2,tmp);

if (n mod 2=1) then p:=tmp\*tmp\*a

else p:=tmp\*tmp;

end;

end;

hoặc viết dưới dạng hàm như sau:

function power(a,n:longint):longint;

var tmp : longint;

begin

if (n=1) then exit(a)

else begin

tmp:=power(a,n div 2);

if (n mod 2=1) then exit(tmp\*tmp\*a)

else exit(tmp\*tmp);

end;

end;

ðể ñánh giá thời gian thực hiện thuật toán, ta tính số phép nhân phải sử dụng, gọi

a' là số phép nhân thực hiện, ta có:

7

a1 \_x0010\_ 0

a'

a b'

2

c + 1 + 1 'ế; ' d 1

a'

a b'

2

c + 2

a b '

2

c + 2 + 2

\*

2eSf'

Như vậy, thuật toán chia ñể trị mất không quá 2eSf' phép nhân, nhỏ hơn rất

nhiều so với ' phép nhân.

4.3. Bài toán ghii

Bài toán: Cho mảng số nguyên

=1. . '>, cần tìm j\_x0016\_99

=1. . '> \_x0010\_

=> )

=\_x0016\_>

ñạt giá trị lớn nhất mà 1

\_x0016\_

'.

Ví dụ: mảng gồm 6 số 4, 2, 5, 8, 1, 7 thì ñộ lệch cần tìm là: 6

Cách 1: Thử tất cả các cặp chỉ số \_x0016\_, , ñộ phức tạp kl

procedure find(var maxDiff:longint);

var i,j :longint;

91

begin

maxDiff:=0;

for i:=1 to n do

for j:=i to n do

if a[j]-a[i]>maxDiff then maxDiff:=a[j]-a[i];

end;

Cách 2: Áp dụng kĩ thuật chia ñể trị, ta chia mảng

=1. . '> thành hai mảng con

=1. . > và

= + 1. . '> trong ñó \_x0010\_ ' I\_x0016\_[ 2, ta có:

j\_x0016\_99

=1. . '> \_x0010\_ mj\_x0016\_99

=1. . >

j\_x0016\_99

= + 1. . '>

?

%

= + 1. . '> ) ?nl

=1. . >

Nếu tìm ñược ñộ lệch j\_x0016\_99, giá trị lớn nhất ?

% và giá trị nhỏ nhất ?nl

của hai mảng con

=1. . > và

= + 1. . '>, ta sẽ dễ ràng xác ñịnh ñược giá trị

j\_x0016\_99

=1. . '>). ðể tìm ñộ lệch, giá trị lớn nhất và giá trị nhỏ nhất của hai mảng

con

=1. . > và

= + 1. . '>, ta lại tiếp tục chia ñôi chúng. Quá trình phân nhỏ

bài toán dừng lại khi ta nhận ñược bài toán mảng con chỉ có 1 phần tử. Từ phương

pháp ñã trình bày ở trên, ta xây dựng thủ tục ñệ quy

find2(l,r,maxDiff,maxValue,minValue)

tìm giá trị ñộ lệch, giá trị lớn nhất, giá trị nhỏ nhất trên mảng

=e. . "> với

1

e

"

'.

const MAXN =100000;

fi ='';

fo ='';

var a :array[1..MAXN]of longint;

n :longint;

maxdiff :longint;

tmp1,tmp2 :longint;

procedure find2(l,r:longint;var

maxDiff,maxValue,minValue :longint);

var mid :longint;

maxD1, maxV1, minV1 :longint;

maxD2, maxV2, minV2 :longint;

begin

if l=r then begin

maxDiff:=0;

maxValue:=a[r];

92

minValue:=a[r];

end

else begin

mid:=(l+r) div 2;

find2(l, mid, maxD1, maxV1,minV1);

find2(mid+1, r, maxD2, maxV2, minV2);

maxDiff:=maxV2 - minV1;

if maxDiff < maxD1 then maxDiff := maxD1;

if maxDiff < maxD2 then maxDiff := maxD2;

if maxV1 > maxV2 then

maxValue:=maxV1 else maxValue:=maxV2;

if minV1 < minV2 then

minValue:=minV1 else minValue:=minV2;

end;

end;

procedure input;

var f :text;

i :longint;

begin

assign(f,fi); reset(f);

readln(f,n);

for i:=1 to n do read(f,a[i]);

close(f);

end;

BEGIN

input;

find2(1,n,maxDiff,tmp1,tmp2);

writeln(maxDiff);

END.

Gọi a' là số phép toán cần thực hiện trên mảng ' phần tử

=1. . '>, ta có:

a' \_x0010\_ 7 0 'ế; ' \_x0010\_ 1

a b'

2

c + a b'

2

c + o 'ế; ' d 1

Giả sử, ' \_x0010\_ 2&

, bằng phương pháp thế ta có:

a' \_x0010\_ a2

&

\_x0010\_ 2a2

& + o \_x0010\_ 22a2

& + o + o

\_x0010\_ 2a2

& + 2o + o \_x0010\_ \* \_x0010\_ 24a2

&4 + 2 + 2 + 1

\_x0010\_ 2&a1 + 2&o + \* + 2o + o \_x0010\_ 2& ) 1o \_x0010\_ ' ) 1o

93

ðộ phức tạp thuật toán là: kl

4.4. Lát nền

Hãy lát nền nhà hình vuông cạnh ' \_x0010\_ 2&

2

10 bị khuyết một phần tư tại

góc trên phải (khuyết phần 2) bằng những viên gạch hình thước thợ tạo bởi 3 ô

vuông ñơn vị.

Nền nhà p \_x0010\_ q Gạch hình thước thợ Một cách lát nền

Giải. Ta chia nền nhà thành 4 phần (hình bên

phải), mỗi phần có hình dạng giống với hình

ban ñầu nhưng có cạnh giảm ñi một nửa. Như

vậy, nếu có thể lát nền với kích thước 2

&

thì ta

hoàn toàn có thể lát nền với kích thước 2

&

.

Thủ tục ñệ quy cover(x,y,s,t) dưới ñây

sẽ lát nền có kích thước :, bị khuyết phần

! ! \_x0010\_ 1,2,3,4 có tọa ñộ trái trên là , r.

const MAXSIZE =1 shl 10;

var a :array[1..MAXSIZE,1..MAXSIZE]of longint;

count,k :longint;

procedure cover(x,y,s,t:longint);

begin

if s = 2 then begin

inc(count);

if t<>1 then a[x,y]:=count;

if t<>2 then a[x,y+1]:=count;

if t<>3 then a[x+1,y]:=count;

if t<>4 then a[x+1,y+1]:=count;

exit;

94

end;

if t=1 then begin

cover(x,y+s div 2,s div 2,3);

cover(x+s div 2,y,s div 2,2);

cover(x+s div 2,y+s div 2,s div 2,1);

cover(x+s div 4,y+s div 4,s div 2,1);

end;

if t=2 then begin

cover(x,y,s div 2,4);

cover(x+s div 2,y,s div 2,2);

cover(x+s div 2,y+s div 2,s div 2,1);

cover(x+s div 4,y+s div 4,s div 2,2);

end;

if t=3 then begin

cover(x,y,s div 2,4);

cover(x,y+s div 2,s div 2,3);

cover(x+s div 2,y+s div 2,s div 2,1);

cover(x+s div 4,y+s div 4,s div 2,3);

end;

if t=4 then begin

cover(x,y,s div 2,4);

cover(x+s div 2,y,s div 2,2);

cover(x,y+s div 2,s div 2,3);

cover(x+s div 4,y+s div 4,s div 2,4);

end;

end;

procedure output;

var i,j :longint;

f :text;

begin

assign(f,'cover.out'); rewrite(f);

for i:=1 to 1 shl k do

begin

for j:=1 to 1 shl k do write(f,a[i,j],' ');

writeln(f);

end;

95

close(f);

end;

BEGIN

write('k=');readln(k);

cover(1,1,1 shl k,2);

output;

END.

Ví dụ về Input / Output của chương trình:

k = 3 1 1 3 3 0 0 0 0

1 4 4 3 0 0 0 0

2 4 13 13 0 0 0 0

2 2 13 16 0 0 0 0

5 5 14 16 16 15 9 9

5 8 14 14 15 15 12 9

6 8 8 7 10 12 12 11

6 6 7 7 10 10 11 11

4.5. Tháp Hà Nội

Cho 3 cái cọc và ' ñĩa có kích thước

khác nhau. Ban ñầu cả ' ñĩa ñều ở cọc 1

và ñược xếp theo thứ tự ñĩa to ở dưới,

ñĩa nhỏ ở trên. Hãy di chuyển cả ' ñĩa từ

cọc 1 sang cọc 3 theo quy tắc sau:

- Một lần chỉ ñược chuyển một ñĩa

- Trong quá trình chuyển ñĩa, có thể sử

dụng cọc 2 làm cọc trung gian và một

ñĩa chỉ ñược ñặt lên một ñĩa lớn hơn.

Giải

ðể chuyển ' ñĩa từ cọc 1 sang cọc 3 ta sẽ thực hiện như sau:

- chuyển ' ) 1 ñĩa từ cọc 1 sang cọc 2, sử dụng cọc 3 làm cọc trung gian

- chuyển 1 ñĩa từ cọc 1 sang cọc 3

- chuyển ' ) 1 ñĩa từ cọc 2 sang cọc 3, sử dụng cọc 1 làm cọc trung gian

96

procedure move(n :longint;src,dst,tmp:longint);

begin

if n = 1 then writeln('move ',src,' ',dst)

else begin

move(n-1,src,tmp,dst);

move(1,src,dst,tmp);

move(n-1,tmp,dst,src);

end;

end;

4.6. Bài toán sắp xếp mảng bằng thuật toán trộn (Merge Sort)

Bài toán: Cho mảng số nguyên

=1. . '>, cần sắp xếp các phần tử của mảng theo

thứ tự tăng dần.

Giải: Ta chia mảng

=1. . '> thành hai mảng con

=1. . > và

= + 1. . '> trong

ñó \_x0010\_ ' I\_x0016\_[ 2. Giả sử, hai mảng con

=1. . > và

= + 1. . '> ñã ñược sắp xếp

tăng dần, ta sẽ trộn hai mảng con ñể ñược mảng

=1. . '> cũng sắp xếp tăng dần.

ðể sắp xếp hai mảng con

=1. . > và

= + 1. . '> ta lại tiếp tục chia ñôi chúng.

Thủ tục ñệ quy MergeSort(i,j) sắp xếp tăng dần mảng con

=\_x0016\_. . > với 1

\_x0016\_

'. ðể sắp xếp cả mảng

=1. . '>, ta chỉ cần gọi thủ tục này với \_x0016\_ \_x0010\_ 1, \_x0010\_ '.

procedure MergeSort(i,j:longint);

var k : longint;

begin

if (i<j) then begin

k:=(i+j) div 2;

MergeSort(i,k);

MergeSort(k+1,j);

Merge(i,k,j);

{thủ tục Merge(i,k,j) trộn hai mảng con A[i..k], A[(k+1)..j] ñã ñược

sắp xếp thành mảng A[i..j] cũng ñược sắp xếp}

end;

end;

97

Việc trộn hai mảng con ñã ñược sắp xếp

=\_x0016\_. . > và

= + 1. . > thành mảng

=\_x0016\_. . > cũng ñược sắp xếp có thể thực hiện trong thời gian k ) \_x0016\_ + 1, là bài tập

3.8 ở chuyên ñề Sắp xếp. Thuật toán MergeSort có ñộ phức tạp là k'eSf'.

5. Quy hoạch ñộng (Dynamic programming)

5.1. Phương pháp

Trong chiến lược chia ñể trị, người ta phân bài toán cần giải thành các bài toán

con. Các bài toán con lại ñược tiếp tục phân thành các bài toán con nhỏ hơn, cứ

thế tiếp tục cho tới khi ta nhận ñược các bài toán con có thể giải ñược dễ dàng.

Tuy nhiên, trong quá trình phân chia như vậy, có thể ta sẽ gặp rất nhiều lần cùng

một bài toán con. Tư tưởng cơ bản của phương pháp quy hoạch ñộng là sử dụng

một bảng ñể lưu giữ lời giải của các bài toán con ñã ñược giải. Khi giải một bài

toán con cần ñến nghiệm của bài toán con cỡ nhỏ hơn, ta chỉ cần lấy lời giải ở

trong bảng mà không cần phải giải lại. Chính vì thế mà các thuật toán ñược thiết

kế bằng quy hoạch ñộng sẽ rất hiệu quả.

ðể giải quyết một bài toán bằng phương pháp quy hoạch ñộng, chúng ta cần tiến

hành những công việc sau:

- Tìm nghiệm của các bài toán con nhỏ nhất.

- Tìm ra công thức (hoặc quy tắc) xây dựng nghiệm của bài toán con thông

qua nghiệm của các bài toán con cỡ nhỏ hơn.

- Tạo ra một bảng lưu giữ các nghiệm của các bài toán con. Sau ñó tính

nghiệm của các bài toán con theo công thức ñã tìm ra và lưu vào bảng.

- Từ các bài toán con ñã giải ñể tìm nghiệm của bài toán.

Sau ñây, chúng ta sẽ tìm hiểu một số ví dụ minh họa cho phương pháp quy

hoạch ñộng.

5.2. Số Fibonacci

Số Fibonacci ñược xác ñịnh bởi công thức:

7

s, \_x0010\_ 0

s \_x0010\_ 1

s\_x0007\_ \_x0010\_ s\_x0007\_ + s\_x0007\_ [ớ\_x0016\_ ' t 2

Hãy xác ñịnh số Fibonacci thứ '.

Cách 1: Áp dụng phương pháp chia ñể trị, ta tính s\_x0007\_ dựa vào s\_x0007\_ và s\_x0007\_.

98

function F(n: longint): int64;

begin

if n <=1 then F := n

else F := F(i - 1) + F(i - 2);

end;

BEGIN

readln(n);

Writeln(F(n));

END.

Hàm ñệ quy F(n) ñể tính số Fibonacci thứ '. Ví dụ ' \_x0010\_ 6, chương trình chính gọi

F(6), nó sẽ gọi tiếp F(5) và F(4) ñể tính ... Quá trình tính toán có thể vẽ như cây

dưới ñây. Ta nhận thấy ñể tính F(6) nó phải tính 1 lần F(5), hai lần F(4), ba lần

F(3), năm lần F(2), ba lần F(1).

Cách 2: Phương pháp quy hoạch ñộng.

Ta sử dụng mảng S[0..MaxN], S[i] ñể lưu lại lời giải cho bài toán tính số

Fibonacci thứ i.

const MaxN =50;

var S :array[0..MaxN]of int64;

n,k :longint;

function F(n:longint):int64;

begin

if S[n]=-1 then

begin

{bài toán chưa ñược giải thì sẽ tiến hành giải}

if n<=1 then S[n]:=n

else S[n]:=F(n-1) + F(n-2);

end;

{nếu bài toán ñã ñược giải thì không cần giải nữa mà lấy luôn kết quả}

F(6)

F(5) F(4)

F(3) F(2)

F(2) F(1)

F(4)

F(3) F(2)

F(2) F(1)

F(3)

F(2) F(1)

99

F:=S[n];

end;

BEGIN

readln(n);

for k:=0 to MaxN do S[k]:=-1;

writeln(F(n));

END.

Ta nhận thấy, mỗi bài toán con chỉ ñược giải ñúng một lần. Hãy cài ñặt cả 2

chương trình trên và thử chạy với ' \_x0010\_ 40 ñể thấy ñược sự khác biệt!

Ta cũng có thể cài ñặt phương pháp quy hoạch ñộng cho bài toán như sau:

const maxN =50;

var S : array[0..maxN] of Int64;

i : longint;

BEGIN

readln(n);

S[0] := 1; S[1] := 1;

for i := 2 to n do

S[i] := S[i - 1] + S[i - 2];

Writeln(S[n]);

END.

Trước hết nó tính sẵn S[0] và S[1], từ ñó tính tiếp S[2], lại tính tiếp ñược S[3],

S[4],.., S[n]. ðảm bảo rằng mỗi giá trị Fibonacci chỉ phải tính 1 lần.

5.3. Dãy con ñơn ñiệu tăng dài nhất

Cho dãy số nguyên A = a1, a2, ..., an. (n ≤ 1000, -10000 ≤ ai ≤ 10000). Một dãy

con của A là một cách chọn ra trong A một số phần tử giữ nguyên thứ tự. Như vậy

A có 2n

dãy con.

Yêu cầu: Tìm dãy con ñơn ñiệu tăng của A có ñộ dài lớn nhất.

Ví dụ: A = (1, 2, 3, 4, 9, 10, 5, 6, 7, 8). Dãy con ñơn ñiệu tăng dài nhất là: (1, 2, 3,

4, 5, 6, 7, 8).

Giải

Bổ sung vào A hai phần tử: a0 = -∞ và an+1 = +∞. Khi ñó dãy con ñơn ñiệu tăng

dài nhất chắc chắn sẽ bắt ñầu từ a0 và kết thúc ở an+1.

Với ∀ i: 0 ≤ i ≤ n + 1. Ta sẽ tính L[i] = ñộ dài dãy con ñơn ñiệu tăng dài nhất bắt

ñầu tại ai

.

100

1. Bài toán nhỏ nhất

L[n + 1] = ðộ dài dãy con ñơn ñiệu tăng dài nhất bắt ñầu tại an+1 = +∞. Dãy con

này chỉ gồm mỗi một phần tử (+∞) nên L[n + 1] = 1.

2. Công thức

Giả sử với i từ n ñến 0, ta cần tính L[i]: ñộ dài dãy con tăng dài nhất bắt ñầu tại ai

.

L[i] ñược tính trong ñiều kiện L[i + 1], L[i + 2], ..., L[n + 1] ñã biết:

Dãy con ñơn ñiệu tăng dài nhất bắt ñầu từ ai

sẽ ñược thành lập bằng cách lấy ai

ghép vào ñầu một trong số những dãy con ñơn ñiệu tăng dài nhất bắt ñầu tại vị trí

aj

ñứng sau ai

.

Ta sẽ chọn dãy nào ñể ghép ai

vào ñầu? Tất nhiên là chỉ ñược ghép ai

vào ñầu

những dãy con bắt ñầu tại aj

nào ñó lớn hơn ai

(ñể ñảm bảo tính tăng) và dĩ nhiên

ta sẽ chọn dãy dài nhất ñể ghép ai

vào ñầu (ñể ñảm bảo tính dài nhất). Vậy L[i]

ñược tính như sau:

Xét tất cả các chỉ số j trong khoảng từ i + 1 ñến n + 1 mà aj

> ai

, chọn ra chỉ số

jmax có L[jmax] lớn nhất. ðặt L[i] := L[jmax] + 1.

3. Truy vết

Tại bước xây dựng dãy L, mỗi khi tính L[i] := L[jmax] + 1, ta ñặt T[i] = jmax.

ðể lưu lại rằng: Dãy con dài nhất bắt ñầu tại ai

sẽ có phần tử thứ hai kế tiếp là

ajmax. Sau khi tính xong hay dãy L và T, ta bắt ñầu từ 0. T[0] là phần tử ñầu tiên

ñược chọn,

T[T[0]] là phần tử thứ hai ñược chọn,

T[T[T[0]]] là phần tử thứ ba ñược chọn ...

Quá trình truy vết có thể diễn tả như sau:

i := T[0];

while i <> n + 1 do

{Chừng nào chưa duyệt ñến số an+1=+∞ ở cuối}

begin

<Thông báo chọn ai>

i := T[i];

end;

Ví dụ: với A = (5, 2, 3, 4, 9, 10, 5, 6, 7, 8).

Hai dãy Length và Trace sau khi tính sẽ là:

101

i 0 1 2 3 4 5 6 7 8 9 10 11

ai -∞ 5 2 3 4 9 10 5 6 7 8 +∞

Length[i] 9 5 8 7 6 3 2 5 4 3 2 1

Trace[i] 2 8 3 4 7 6 11 8 9 10 11

Const max = 1000;

var

a, L, T: array[0..max + 1] of longint;

n: longint;

procedure Enter; {Nhập dữ liệu}

var

i: longint;

begin

Write('n = '); Readln(n);

for i := 1 to n do

begin

Write('a[', i, '] = '); Readln(a[i]);

end;

end;

procedure Optimize; {Quy hoạch ñộng}

var

i, j, jmax: longint;

begin

a[0] := -32768; a[n + 1] := 32767; {Thêm hai phần tử canh hai

ñầu dãy a}

L[n + 1] := 1; {ðiền cơ sở quy hoach ñộng vào bảng phương án}

for i := n downto 0 do

begin

{Chọn trong các chỉ số j ñứng sau i thoả mãn aj > ai ra chỉ số jmax có L[jmax] lớn nhất}

jmax := n + 1;

for j := i + 1 to n + 1 do

if (a[j] > a[i]) and (L[j] > L[jmax]) then jmax

:= j;

L[i] := L[jmax] + 1; {Lưu ñộ dài dãy con tăng dài nhất bắt ñầu tại ai}

T[i] := jmax; {Lưu vết: phần tử ñứng liền sau ai

trong dãy con tăng

dài nhất ñó là ajmax}

end;

Truy vết

102

Writeln('Length of result : ', L[0] - 2);{Chiều dài dãy con

tăng dài nhất}

i := T[0]; {Bắt ñầu truy vết tìm nghiệm}

while i <> n + 1 do

begin

Writeln('a[', i, '] = ', a[i]);

i := T[i];

end;

end;

begin

Enter;

Optimize;

end.

5.4. Dãy con chung dài nhất

Cho hai số nguyên dương ?, l 0 ( ?, l

100 và hai dãy số nguyên: A1,

A2,..., AM và B1, B2,..., B N. Tìm một dãy dài nhất C là dãy con chung dài nhất của

hai dãy A và B, nhận ñược từ A bằng cách xoá ñi một số số hạng và cũng nhận

ñược từ B bằng cách xoá ñi một số số hạng.

Dữ liệu vào trong file LCS.INP có dạng:

+ Dòng thứ nhất chứa M số A1, A2,..., AM

+ Dòng thứ hai chứa N số B1, B2,..., B N.

Dữ liệu ra trong file LCS.OUT có dạng:

+ Dòng thứ nhất ghi số k là số số hạng của dãy C.

+ Dòng thứ hai chứa k số là các số hạng của dãy C.

Giải

Cần xây dựng mảng L[0..M, 0..N] với ý nghĩa: L[i, j] là ñộ dài của dãy chung dài

nhất của hai dãy A[0.. i] và B[0..j].

ðương nhiên nếu một dãy là rỗng (số phần tử là 0) thì dãy con chung cũng là rỗng

vì vậy L[0, j] = 0 ∀j, j = 1.. N, L[i, 0] = 0 ∀i, i = 1.. M. Với M ≥ i > 0 và N ≥ j >

0 thì L[i, j] ñược tính theo công thức truy hồi sau:

L[i,j] = Max{L[i, j-1], L[i-1, j], L[i-1, j-1] + x}

(với x = 0 nếu A[i] ≠ B[j] , x=1 nếu A[i]=B[j])

103

const fi = 'LCS.INP';

fo = 'LCS.OUT';

MaxMN = 100;

var f : text;

a,b : array[0..MaxMN] of longint;

l : array[0..MaxMN,0..MaxMN] of longint;

m,n : longint;

p : array[0..MaxMN] of longint;

count : longint;

procedure Enter;

var f : text;

begin

m := 0;

n := 0;

assign(f,fi);

reset(f);

while not eoln(f) do

begin

inc(m);

read(f,a[m]);

end;

readln(f);

while not eoln(f) do

begin

inc(n);

read(f,b[n]);

end;

close(f);

end;

function max(x,y : longint) : longint;

begin

if x>y then max := y

else max := y;

end;

procedure Optimize;

var i,j : longint;

begin

for i:=1 to m do l[i,0] := 0;

for j:=1 to n do l[0,j] := 0;

for i:=1 to m do

for j:=1 to n do

begin

104

if a[i]=b[j] then l[i,j] := l[i-1,j-1] + 1

else l[i,j] := max(l[i,j-1],l[i-1,j]);

end;

end;

procedure Trace;

var f : text;

i,j : longint;

begin

assign(f,fo);

rewrite(f);

writeln(f,l[m,n]);

i := m;

j := n;

fillchar(p,sizeof(p),0);

count:= 0;

while (i>0) and (j>0) do

begin

if a[i]=b[j] then

begin

inc(count);

p[count] := a[i];

dec(i);

dec(j);

end

else if l[i,j]=l[i,j-1] then dec(j)

else dec(i);

end;

for i:=count downto 1 do write(f,p[i],' ');

close(f);

end;

BEGIN

Enter;

Optimize;

Trace;

END.

5.5. Bài toán cái túi

Trong siêu thị có n gói hàng (n ≤ 100), gói hàng thứ i có trọng lượng là Wi ≤ 100

và trị giá Vi ≤ 100. Một tên trộm ñột nhập vào siêu thị, sức của tên trộm không thể

mang ñược trọng lượng vượt quá M ( M ≤ 100). Hỏi tên trộm sẽ lấy ñi những gói

hàng nào ñể ñược tổng giá trị lớn nhất.

105

Giải

Nếu gọi B[i, j] là giá trị lớn nhất có thể có bằng cách chọn trong các gói {1, 2, ...,

i} với giới hạn trọng lượng j. Thì giá trị lớn nhất khi ñược chọn trong số n gói với

giới hạn trọng lượng M chính là B[n, M].

1. Công thức tính B[i, j].

Với giới hạn trọng lượng j, việc chọn tối ưu trong số các gói {1, 2, ...,i - 1, i} ñể

có giá trị lớn nhất sẽ có hai khả năng:

• Nếu không chọn gói thứ i thì B[i, j] là giá trị lớn nhất có thể bằng cách chọn

trong số các gói {1, 2, ..., i - 1} với giới hạn trọng lượng là j. Tức là B[i, j] =

B[i - 1, j]

• Nếu có chọn gói thứ i (tất nhiên chỉ xét tới trường hợp này khi mà Wi ≤ j) thì

B[i, j] bằng giá trị gói thứ i là Vi

cộng với giá trị lớn nhất có thể có ñược bằng

cách chọn trong số các gói {1, 2, ..., i - 1} với giới hạn trọng lượng j - Wi

. Tức

là về mặt giá trị thu ñược: B[i, j] = Vi

+ B[i - 1, j - Wi]

Vì theo cách xây dựng B[i, j] là giá trị lớn nhất có thể nên nó sẽ là max trong hai

giá trị thu ñược ở trên.

2. Cơ sở quy hoạch ñộng:

Dễ thấy B[0, j] = giá trị lớn nhất có thể bằng cách chọn trong số 0 gói = 0.

3. Tính bảng phương án:

Bảng phương án B gồm n + 1 dòng, M + 1 cột, trước tiên ñược ñiền cơ sở quy

hoạch ñộng: Dòng 0 gồm toàn số 0. Sử dụng công thức truy hồi, dùng dòng 0 tính

dòng 1, dùng dòng 1 tính dòng 2, v.v... ñến khi tính hết dòng n.

0 1 ... M

0 0 0 0 0

1

2

... ...

n

106

4. Truy vết:

Tính xong bảng phương án thì ta quan tâm ñến b[n, M] ñó chính là giá trị lớn nhất

thu ñược khi chọn trong cả n gói với giới hạn trọng lượng M. Nếu b[n, M] = b[n -

1, M] thì tức là không chọn gói thứ n, ta truy tiếp b[n - 1, M]. Còn nếu b[n, M] ≠

b[n - 1, M] thì ta thông báo rằng phép chọn tối ưu có chọn gói thứ n và truy tiếp

b[n - 1, M - Wn]. Cứ tiếp tục cho tới khi truy lên tới hàng 0 của bảng phương án.

const

max = 100;

var

W, V : array[1..max] of longint;

B : array[0..max, 0..max] of longint;

n, M : longint;

procedure Enter;

var

i: longint;

begin

Write('n = '); Readln(n);

for i := 1 to n do

begin

Writeln('Pack ', i);

Write(' + Weight : '); Readln(W[i]);

Write(' + Value : '); Readln(V[i]);

end;

Write('M = '); Readln(M);

end;

procedure Optimize;

var

i, j: longint;

begin

FillChar(B[0], SizeOf(B[0]), 0);

for i := 1 to n do

for j := 0 to M do

begin

B[i, j] := B[i - 1, j];

if (j >= W[i]) and (B[i, j] < B[i-1,j-W[i]] + V[i])

then

B[i, j] := B[i - 1, j - W[i]] + V[i];

end;

107

end;

procedure Trace;

begin

Writeln('Max Value : ', B[n, M]);

Writeln('Selected Packs: ');

while n <> 0 do

begin

if B[n, M] <> B[n - 1, M] then

begin

Writeln('Pack ', n, ' W = ', W[n], ' Value = ', V[n]);

M := M - W[n];

end;

Dec(n);

end;

end;

BEGIN

Enter;

Optimize;

Trace;

END.

Bài tập

4.1. Cho danh sách tên của ' '

10 học sinh (các tên ñôi một khác nhau) và

một số nguyên dương

'. Hãy liệt kê tất cả các cách chọn học sinh

trong ' học sinh.

Ví dụ:

Dữ liệu vào Kết quả ra

' \_x0010\_ 4, \_x0010\_ 2, danh sách

tên học sinh như sau:

An

Binh

Hong

Minh

Có 6 cách chọn 2 học sinh trong

4 học sinh:

1. An Binh

2. An Hong

3. An Minh

4. Binh Hong

5. Binh Minh

6. Hong Minh

108

4.2. Một dãy nhị phân ñộ dài ' '

10 là một dãy \_x0010\_ … \_x0007\_ trong ñó

$ 0,1, \_x0016\_ \_x0010\_ 1,2, . . , '. Hãy liệt kê tất cả các dãy nhị phân ñộ dài '

Dữ liệu vào Kết quả ra

' \_x0010\_ 3 Có 8 dãy nhị phân ñộ dài 3

1. 000

2. 001

3. 010

4. 011

5. 100

6. 101

7. 110

8. 111

4.3. Cho xâu S (ñộ dài không vượt quá 10) chỉ gồm các kí tự u

u ñến uvu (các kí

tự trong xâu S ñôi một khác nhau). Hãy liệt kê tất cả các hoán vị khác nhau

của xâu S.

Dữ liệu vào Kết quả ra

S='XYZ' Có 6 hoán vị khác nhau của 'XYZ'

1. XYZ

2. XZY

3. YXZ

4. YZX

5. ZXY

6. ZYX

4.4. Cho số nguyên dương ' '

20,hãy liệt kê tất cả các xâu ñộ dài ' chỉ gồm

2 kí tự u

u hoặc u\_x001D\_u mà không có 2 kí tự u\_x001D\_u nào ñứng cạnh nhau.

Dữ liệu vào Kết quả ra

' \_x0010\_ 4 Có 8 xâu ñộ dài 4

1. AAAA

2. AAAB

3. AABA

4. ABAA

5. ABAB

109

6. BAAA

7. BAAB

8. BABA

4.5. Cho dãy số

gồm ll

10 số nguyên ‑, ‑, … , ‑w và một số nguyên

dương x 1 ( x ( l). Hãy ñưa ra một cách chia dãy số thành x nhóm mà

các nhóm có tổng bằng nhau.

Dữ liệu vào Kết quả ra

N=5, S=3

Dãy số a:

1, 4, 6, 9, 10

nhóm 1: 4, 6

nhóm 2: 1, 9

nhóm 3: 10

4.6. Một xâu X = x1x2..xM ñược gọi là xâu con của xâu Y = y1y2..yN nếu ta có thể

nhận ñược xâu X từ xâu Y bằng cách xoá ñi một số kí tự, tức là tồn tại một

dãy các chỉ số:

1

\_x0016\_ ( \_x0016\_ ( \* ( \_x0016\_y

l ñể \_x0010\_ r L

, \_x0010\_ r M

, … , y \_x0010\_ r z

Ví dụ: X='adz' là xâu con của xâu Y='baczdtz'; \_x0016\_ \_x0010\_ 2 ( \_x0016\_ \_x0010\_ 5 ( \_x0016\_4 \_x0010\_ 7.

Nhập vào một xâu S (ñộ dài không quá 15, chỉ gồm các kí tự 'a' ñến 'z'), hãy

liệt kê tất cả các xâu con khác nhau của xâu S.

Dữ liệu vào Kết quả ra

S='aba' Có 6 xâu con khác nhau của 'aba'

1. a

2. b

3. aa

4. ab

5. ba

6. aba

4.7. Cho số nguyên dương ' '

10, liệt kê tất cả các cách khác nhau ñặt '

dấu ngoặc mở và ' dấu ngoặc ñóng ñúng ñắn?

Dữ liệu vào Kết quả ra

' \_x0010\_ 3 Có 5 cách

b \_x0019\_c , \_x0019\_, \_x0019\_ , \_x0019\_,

110

4.8. Cho ' '

10 số nguyên dương ‑, ‑, … , ‑\_x0007\_ ‑

10|

. Tìm số nguyên

dương < nhỏ nhất sao cho < không phân tích ñược dưới dạng tổng của một

số các số (mỗi số sử dụng không quá một lần) thuộc ' số trên.

Dữ liệu vào Kết quả ra

n=4

Dãy số a:

1, 2, 3, 6

13

4.9. Cho xâu S (ñộ dài không vượt quá 10) chỉ gồm các kí tự u

u ñến uvu (các kí

tự trong xâu S không nhất thiết phải khác nhau). Hãy liệt kê tất cả các hoán

vị khác nhau của xâu S.

Dữ liệu vào Kết quả ra

S='ABA' Có 3 hoán vị khác nhau của 'ABA'

1. AAB

2. ABA

3. BAA

4.10. Bài toán mã ñi tuần

Cho bàn cờ ' ' ô, tìm cách di chuyển một quân mã (mã di chuyển theo

luật cờ vua) trên bàn cờ xuất phát từ ô (1,1) ñi qua tất cả các ô, mỗi ô qua

ñúng một lần.

Ví dụ: N=5

1 24 13 18 7

14 19 8 23 12

9 2 25 6 17

20 15 4 11 22

3 10 21 16 5

4.11. Số siêu nguyên tố là số nguyên tố mà khi bỏ một số tuỳ ý các chữ số bên

phải của nó thì phần còn lại vẫn tạo thành một số nguyên tố.

Ví dụ: 2333 là một số siêu nguyên tố có 4 chữ số vì 233, 23, 2 cũng là các

số nguyên tố.

111

Cho số nguyên dương N (0< N <10), ñưa ra các số siêu nguyên tố có N chữ

số cùng số lượng của chúng.

Ví dụ: Với N=4

Có 16 số: 2333 2339 2393 2399 2939 3119 3137 3733 3739 3793 3797

5939 7193 7331 7333 7393

4.12. Cho một xâu S (chỉ gồm các kí tự '0' ñến '9', ñộ dài nhỏ hơn 10) và số

nguyên M, hãy ñưa ra một cách chèn vào S các dấu '+' hoặc '-' ñể thu ñược

số M cho trước (nếu có thể).

Ví dụ: M = 8, S='123456789' một cách chèn: '-1+2-3+4+5-6+7';

4.13. Trong cờ vua quân tượng chỉ có thể di chuyển theo

ñường chéo và hai quân tượng có thể chiếu nhau

nếu chúng nằm trên ñường di chuyển của nhau.

Trong hình bên, hình vuông tô ñậm thể hiện các vị

trí mà quân tượng B1 có thể ñi tới ñược, quân tượng

B1 và B2 chiếu nhau, quân B1 và B3 không chiếu

nhau. Cho kích thước N của bàn cờ và K quân

tượng, hỏi có bao nhiêu cách ñặt các quân tượng

vào bàn cờ mà các quân tượng không chiếu nhau.

Dữ liệu vào trong file: “bishops.inp” có dạng:

- Dòng ñầu là số t là số test (t≤10)

- t dòng sau mỗi dòng chứa 2 số nguyên dương N, K (2≤N≤10, 0<K≤N2

)

Kết quả ra file: “bishops.out” gồm t dòng, mỗi chứa một số duy nhất là số

cách ñặt các quân tượng vào bàn cờ tương ứng với dữ liệu vào.

4.14. N-mino là hình thu ñược từ N hình vuông 1×1 ghép lại (cạnh kề cạnh). Hai

n-mino ñược gọi là ñồng nhất nếu chúng có thể ñặt chồng khít lên nhau.

Cho số nguyên dương N (1<N<8), tính và vẽ ra tất cả các N-mino trên màn

hình.

Ví dụ: Với N=3 chỉ có hai loại N-mino sau ñây:

3-mino thẳng 3-mino hình thước thợ

4.15. Trong mục 2.2, lời giải bài toán TSP là một giải pháp nhánh cận rất thô sơ.

Hãy thử chạy chương trình với trường hợp như sau: số thành phố ' \_x0010\_ 20,

112

khoảng cách giữa các thành phố bằng 1 (nghĩa là .=\_x0016\_, > \_x0010\_ 1 với \_x0016\_ ). Hãy

rút ra nhận xét và có thể ñánh giá nhánh cận chặt hơn nữa làm tăng hiệu quả

của chương trình.

4.16. Cho bàn cờ quốc tế 8×8 ô, mỗi ô ghi một số nguyên dương không vượt quá

32000.

Yêu cầu: Xếp 8 quân hậu lên bàn cờ sao cho không quân nào khống chế

ñược quân nào và tổng các số ghi trên các ô mà quân hậu ñứng là lớn nhất.

Dữ liệu vào: gồm 8 dòng, mỗi dòng ghi 8 số nguyên dương, giữa các số

cách nhau một dấu cách.

Kết quả ra: một số duy nhất là ñáp số của bài toán.

Dữ liệu vào Kết quả ra

1 2 4 9 3 2 1 4

6 9 5 4 2 3 1 4

3 6 2 3 4 1 8 3

2 3 7 3 2 1 4 2

1 2 3 2 3 9 2 1

2 1 3 4 2 4 2 8

2 1 3 2 8 4 2 1

8 2 3 4 2 3 1 2

66

4.17. Một chiếc ba lô có thể chứa ñược một khối lượng }. Có ' '

20 ñồ vật

ñược ñánh số 1, 2, . . , '. ðồ vật \_x0016\_ có khối lượng ‑

và có giá trị ­

. Cần chọn

các ñồ vật cho vào ba lô ñể tổng giá trị các ñồ vật là lớn nhất.

4.18. Dominoes

Có N quân Domino xếp thành một hàng

như hình vẽ

Mỗi quân Domino ñược chia làm hai

phần, phần trên và phần dưới. Trên mặt

mỗi phần có từ 1 ñến 6 dấu chấm.

Ta nhận thấy rằng:

Tổng số dấu chấm ở phần trên của N quân Domino bằng: 6+1+1+1=9, tổng

số dấu chấm ở phần dưới của N quân Domino bằng 1+5+3+2=11, ñộ chênh

lệch giữa tổng trên và tổng dưới bằng |9-11|=2

113

Với mỗi quân, bạn có thể quay 180o

ñể phần trên trở thành phần dưới, phần

dưới trở thành phần trên, và khi ñó ñộ chênh lệch có thể ñược thay ñổi. Ví dụ

như ta quay quân Domino cuối cùng của hình trên thì ñộ chênh lệch bằng 0

Bài toán ñặt ra là: Cần quay ít nhất bao nhiêu quân Domino nhất ñể ñộ

chênh lệch giữa phần trên và phần dưới là nhỏ nhất.

Dữ liệu vào trong file: “DOMINO.INP” có dạng:

- Dòng ñầu là số nguyên dương N (1≤N≤20)

- N dòng sau, mỗi dòng hai số ai

, bi

là số dấu chấm ở phần trên, số dấu chấm

ở phần dưới của quân Domino thứ i (1≤ ai

, bi

≤6)

Kết quả ra file: “DOMINO.OUT” có dạng: Gồm 1 dòng duy nhẩt chứa 2 số

nguyên cách nhau một dấu cách là ñộ chênh lệch nhỏ nhất và số quân

Domino cần quay ít nhất ñể ñược ñộ chênh lệch ñó.

4.19. Cho một lưới MxN (M, N

10) ô, mỗi ô ñặt một bóng ñèn bật hoặc tắt. Trên

mỗi dòng và mỗi cột có một công tắc. Nếu tác ñộng vào công tắc dòng i

(i=1..M) hoặc công tắc cột j (j=1..N) thì tất cả các bóng ñèn trên dòng i hoặc

cột j sẽ thay ñổi trạng thái. Hãy tìm cách tác ñộng vào các công tắc ñể ñược

nhiều ñèn sáng nhất.

4.20. Có 16 ñồng xu xếp thành bảng 4x4, mỗi ñồng xu có thể úp hoặc ngửa như

hình vẽ sau:

Màu ñen thể hiện ñồng xu úp, màu trắng thể hiện ñồng xu ngửa.

Tại mỗi bước ta có phép biến ñổi sau: Chọn một

ñồng xu và thay ñổi trạng thái của ñồng xu ñó và

tất cả các ñồng xu nằm ở các ô chung cạnh (úp

thành ngửa, ngửa thành úp). Cho trước một trạng

thái các ñồng xu, hãy lập trình tìm số phép biến

ñổi ít nhất ñể ñưa về trạng thái tất cả các ñồng xu

hoặc ñều úp hoặc ñều ngửa.

Dữ liệu vào trong file “COIN.INP” có dạng: Gồm 4 dòng, mỗi dòng 4 kí tự

'w' - mô tả trạng thái ngửa hoặc 'b'- mô tả trạng thái úp.

Kết quả ra file “COIN.OUT” có dạng: Nếu có thể biến ñổi ñược ghi số

phép biến ñổi ít nhất nếu không ghi “Impossible”

COIN.INP COIN.OUT COIN.INP COIN.OUT

bwbw

wwww

Impossible bwwb

bbwb

4

114

bbwb

bwwb

bwwb

bwww

4.21. Có N file chương trình với dung lượng S1, S2,…,Sn và loại ñĩa CD có dung

lượng D. Hỏi cần ít nhất bao nhiêu ñĩa CD ñể có thể copy ñủ tất cả các file

chương trình (một file chương trình chỉ nằm trong một ñĩa CD).

a) Giải bài toán bằng phương pháp nhánh cận với N

10.

b) Giải bài toán bằng một thuật toán tham ăn với N

100.

Dữ liệu vào Kết quả ra

N=5, D=700

320, 100, 300, 560, 50

Cần ít nhất 2 ñĩa CD

ðĩa 1: 320, 300, 50

ðĩa 2: 100, 560

4.22. Chương trình giải bài toán lập lịch giảm thiểu trễ hạn (ở mục 3.4) có ñộ

phức tạp kl4

, hãy cải tiến hàm check ñể nhận ñược chương trình với ñộ

phức tạp kl

.

4.23. Cho một xâu S (ñộ dài không quá 200) chỉ gồm 3 loại kí tự u

~

, u\_x001D\_~

, u.u. Ta

có phép ñổi chỗ hai kí tự bất kì trong xâu, hãy tìm cách biến ñổi ít bước nhất

ñể ñược xâu theo thứ tự tăng dần.

Dữ liệu vào Kết quả ra

S='CBABA' Cần ít nhất 2 phép biến ñổi

CBABA ABABC AABBC

4.24. Cho l l

1000 ñoạn số nguyên =‑

, T

>, hãy chọn một tập gồm ít số

nhất mà mỗi ñoạn số nguyên trên ñều có ít nhất 2 số thuộc tập.

|‑

|, |T

|

10|

Ví dụ: có 5 ñoạn =0,10>, =2,3>, =4,7>, =3,5>, =5,8>, ta chọn tập gồm 4 số

{2, 3, 5, 7}

4.25. Cho phân số M/N (0<M<N, M,N nguyên). Hãy phân tích phân số này thành

tổng các phân số có tử số bằng 1, càng ít số hạng càng tốt.

Dữ liệu vào từ file “PS.IN” chứa 2 số M, N

Kết quả ra file “PS.OUT”

- Dòng ñầu là số lượng số tách

- Các dòng sau mỗi dòng chứa mẫu số của các số hạng

115

Dữ liệu vào Kết quả ra

5 6 2

2 3

4.26. Cho một số tự nhiên N. Hãy tìm cách phân tích số N thành các số nguyên

dương p1, p2,..,pk (với k>1) sao cho:

- p1, p2,…, pk ñôi một khác nhau

- p1 + p2 + …+ pk = N

- S=p1 \* p2 \* …\* pk ñạt giá trị lớn nhất

Dữ liệu vào trong file: “PT.INP” có dạng: Gồm nhiều test, mỗi dòng là một

test chứa một số N (5≤N≤1000)

Kết quả ra file: “PT.OUT” có dạng: Gồm nhiều dòng, mỗi dòng là tích lớn

nhất ñạt ñược (số S) cho test ñó

Dữ liệu vào Kết quả ra

5 6

7 12

4.27. Cho hai phép toán \*2 (nhân với 2) và /3 (chia nguyên cho 3). Cho trước số

1, bằng cách sử dụng hai phép toán trên ta xây dựng ñược biểu thức có giá

trị bằng N.

Ví dụ N=6 thì 1\*2\*2\*2\*2\*2/3/3\*2=6 (thực hiện từ trái qua phải)

Dữ liệu vào từ file “BT.INP” chứa số N (N có không quá 100 chữ số)

Ghi kết quả ra file “BT.OUT” biểu thức ngắn nhất có thể

4.28. Cho số nguyên dương N (N

10100), hãy tách N thành tổng ít các số

Fibonacci nhất.

Ví dụ: N=16=1+5+13

4.29. Cần phải tổ chức việc thực hiện N chương trình ñánh số từ 1 ñến N trên một

máy tính. Mỗi chương trình thứ i ñòi hỏi thời gian tính là 1 giờ, và nếu nó

ñược hoàn thành trước thời ñiểm d[i] (giả sử thời ñiểm bắt ñầu thực hiện các

chương trình là 0) thì người chủ máy tính sẽ ñược trả tiền công là w[i] (i =

1,2,...,N). Việc thực hiện mỗi chương trình phải ñược tiến hành liên tục từ

lúc bắt ñầu cho ñến khi kết thúc không cho phép ngắt quãng, ñồng thời tại

mỗi thời ñiểm máy chỉ có thể thực hiện một chương trình).

Hãy tìm trình tự thực hiện các chương trình sao cho tổng tiền công nhận

ñược là lớn nhất.

116

Dữ liệu vào ñược cho trong JOB.INP:

- Dòng ñầu tiên chứa số N (N ≤ 5000),

- Dòng thứ i trong N dòng tiếp theo chứa 2 số d[i], w[i] ñược ghi cách nhau

bởi dấu cách.

Kết quả ñưa ra file JOB.OUT:

- Dòng ñầu tiên chứa tổng tiền công nhận ñược theo trình tự tìm ñược.

- Dòng tiếp theo ghi trình tự thực hiện các chương trình.

4.30. Tìm K chữ số cuối cùng của MN

(0< K

9, 0

M, N

109

)

Ví dụ: K=2, M=2, N=10, ta có 210=1024, như vậy 2 chữ số cuối cùng của

2

10 là 24

4.31. Viết hàm kiểm tra tính nguyên tố của số l l

10|

theo Fermat.

4.32. Lát gạch

Cho một nền nhà hình vuông có kích thước 2

&

bị khuyết một ô, hãy tìm

cách lát nền nhà bằng loại gạch hình thước thợ (tạo bởi 3 hình vuông

ñơn vị).

nền nhà (ô màu ñen là ô khuyết) một cách lát nền

4.33. Cho dãy ‑, ‑, … , ‑\_x0007\_, các số ñôi một khác nhau và số nguyên dương

1

'. Hãy ñưa ra giá trị nhỏ thứ trong dãy.

Ví dụ: dãy gồm 5 phần tử: 5, 7, 1, 3, 4 và \_x0010\_ 3 thì giá trị nhỏ thứ là 4.

4.34. Dãy con lồi

Dãy số nguyên A1, A2, …, AN ñược gọi là lồi, nếu nó giảm dần từ A1 ñển

một Ai

nào ñó, rồi tăng dần tới AN.

Ví dụ dãy lồi: 10 5 4 2 –1 4 6 8 12

Yêu cầu: Cho một dãy số nguyên, bằng cách xóa bớt một số phần tử của

dãy và giữ nguyên trình tự các phần tử còn lại, ta nhận ñược dãy con lồi dài

nhất.

117

Dữ liệu vào trong file: DS.INP

- Dong ñầu là N (N<=10000)

- Các dòng sau là N số nguyên của dãy số (các số kiểu longint)

Kết quả ra file: DS.OUT

- Ghi số phần tử của dãy con tìm ñược

- Các dòng tiếp theo ghi các số thuộc dãy con

4.35. Palindrome

Một xâu ñược gọi là xâu ñối xứng nếu ñọc từ trái qua phải cũng giống như

ñọc từ phải qua trái. Ví dụ xâu “madam” là một xâu ñối xứng. Bài toán ñặt

ra là cho một xâu S gồm các kí tự thuộc tập ['a'..'z'], hãy tìm cách chèn vào

xâu S ít nhất các kí tự ñể xâu S thành xâu ñối xứng.

Ví dụ: xâu “adbhbca” ta sẽ chèn thêm 2 kí tự (c và d) ñể ñược xâu ñối xứng

“adcbhbcda”.

Dữ liệu vào trong file PALIN.INP có dạng: Gồm một dòng chứa xâu S. (ñộ

dài mỗi xâu không vượt quá 200)

Kết quả ghi ra file PALIN.OUT có dạng: Gồm một dòng là một xâu ñối

xứng sau khi ñã chèn thêm ít kí tự nhất vào xâu S.

Palin.inp Palin.out

acbcd adcbcda

4.36. Stones

Có N ñống sỏi xếp thành một hàng, ñống thứ i có Ai

viên sỏi. Ta có thể

ghép hai ñống sỏi kề nhau thành một ñống và mất một chi phí bằng tổng hai

ñống sỏi ñó.

Yêu cầu: Hãy tìm cách ghép N ñống sỏi này thành một ñống với chi phí là

nhỏ nhất.

Ví dụ: Có 5 ñống sỏi

4 1 2 7 5

4 3 7 5

7 7 5

7 12

19

Phạt = 3 + 7 + 12 + 19 = 41

Dữ liệu vào trong file “STONES.INP” có dạng:

118

- Dòng ñầu là số N (N < 101) là số ñống sỏi

- Dòng thứ 2 gồm N số nguyên là số sỏi của N ñống sỏi. (0 < Ai < 1001)

Kết quả ra file “STONES.OUT” có dạng: gồm một số là chi phí nhỏ nhất ñể

ghép N ñống thành một ñống.

STONES.INP STONES.OUT

5

4 1 2 7 5

41

4.37. Cắt hình 1

Có một hình chữ nhật M×N ô, mỗi lần ta ñược phép cắt một hình chữ nhật

thành hai hình chữ nhật con theo chiều ngang hoặc chiều dọc và lại tiếp tục

cắt các hình chữ nhật con cho ñến khi ñược hình vuông thì dừng.

Hỏi có thể cắt hình chữ nhật MxN thành ít nhất bao nhiêu hình vuông.

Dữ liệu vào trong file HCN.INP:

Gồm 1 số dòng, mỗi dòng là 1 test là một cặp số M, N (1<=M,N<=100)

Kết quả ra file HCN.INP:

Gồm 1 số dòng là kết quả tương ứng với dữ liệu vào

4.38. Cắt hình 2

Cho một bảng số A gồm M dòng, N cột, các giá trị của bảng A chỉ là 0

hoặc 1. Ta muốn cắt bảng A thành các hình chữ nhật con sao cho các hình

chữ nhật con có giá trị toàn bằng 1 hay toàn bằng 0. Một lần cắt là một nhát

cắt thẳng theo dòng hoặc theo cột của một hình chữ nhật thành hai hình chữ

nhật riêng biệt. Cứ tiếp tục cắt cho ñến khi hình chữ nhật toàn bằng 1 hay

toàn bằng 0. Hãy tìm cách cắt ñể ñược ít hình chữ nhật nhất mà các hình chữ

nhật con có giá trị toàn bằng 1 hay toàn bằng 0.

Ví dụ: Bảng số 5×5 sau ñược chia thành 8 hình chữ nhật con.

Dữ liệu vào trong file HCN2.INP

119

- Dòng ñầu là 2 số nguyên dương M, N (M,N≤30)

- M dòng tiếp theo, mỗi dòng N số chỉ gồm 0 hoặc 1 thể hiện bảng số A

Kết quả ra file HCN2

Gồm 1 dòng duy nhất chứa một số duy nhất là số hình chữ nhật ít nhất

4.39. TKSEQ

Cho dãy số A gồm N số nguyên và số nguyên K. Tìm dãy chỉ số

1≤i1<i2<…<i3K≤N sao cho:

\_x0010\_ ‑ L ) ‑ M + ‑ 

\_x0019\_ + ‑  ) ‑  + ‑ 

\_x0019\_+. . +‑ M ) ‑ L + ‑ \_x0019\_

ñạt giá trị lớn nhất.

Dữ liệu vào trong file “TKSEQ.INP” có dạng:

- Dòng ñầu là gồm 2 số nguyên N, K (0<3K≤N≤500)

- Dòng 2 gồm N số nguyên a1, a2,…, aN (|ai

|<109

)

Kết quả ra file “TKSEQ.OUT” có dạng: gồm một số duy nhất S lớn nhất

tìm ñược

TKSEQ.INP TKSEQ.OUT

5 1

1 2 3 4 5

4

4.40. Least-Squares Segmentation

Ta ñịnh nghĩa trọng số của ñoạn số từ số ở vị trí thứ i ñến vị trí thứ j của dãy

số nguyên A[1], A[2], ..., A[N] là:

∑

= > ) <‑'

& trong ñó <‑' \_x0010\_ ∑

= >

& / ) \_x0016\_ + 1

Yêu cầu: Cho dãy số nguyên A gồm N số A[1], A[2], ..., A[N] và số nguyên

dương G (1 < G2

< N). Hãy chia dãy A thành ñúng G ñoạn ñể tổng trọng số

là nhỏ nhất.

Dữ liệu vào trong file văn bản “LSS.INP” có dạng:

- Dòng ñầu gồm hai số N và G (1 < G2

< N < 1001)

- N dòng tiếp theo, mỗi dòng một số nguyên mô tả dãy số A (0<A[i]<106

)

Kết quả ra file văn bản “LSS.OUT” có dạng: gồm một dòng chứa một số

thực duy nhất là ñáp án của bài toán. (ñưa ra theo quy cách :0:2)

LSS.INP LSS.OUT

5 2

3

0.50

120

3

3

4

5

4.41. Phân trang (ðề thi chọn ñội tuyển quốc gia 1999)

Văn bản là một dãy gồm N từ ñánh số từ 1 ñến N. Từ thứ i có ñộ dài là wi

(i=1, 2,... N). Phân trang là một cách xếp lần lượt các từ của văn bản vào

dãy các dòng, mỗi dòng có ñộ dài L, sao cho tổng ñộ dài của các từ trên

cùng một dòng không vượt quá L. Ta gọi hệ số phạt của mỗi dòng trong

cách phân trang là hiệu số (L-S), trong ñó S là tổng ñộ dài của các từ xếp

trên dòng ñó. Hệ số phạt của cách phân trang là giá trị lớn nhất trong số các

hệ số phạt của các dòng.

Yêu cầu: Tìm cách phân trang với hệ số phạt nhỏ nhất.

Dữ liệu vào từ tệp văn bản PTRANG.INP

- Dòng 1 chứa 2 số nguyên dương N, L (N<=4000, L<=70)

- Dòng thứ i trong số N dòng tiếp theo chứa số nguyên dương wi (wi<=L),

i=1, 2,.., N

Kết quả ghi ra file văn bản PTRANG.OUT

- Dòng 1 ghi 2 số P, Q theo thứ tự là hệ số phạt và số dòng theo cách phân

trang tìm ñược

- Dòng thứ i trong số Q dòng tiếp theo ghi chỉ số của các từ trong dòng thứ i

của cách phân trang.

4.42. Chọn số

Cho mảng A có kích thước NxN gồm các số nguyên không âm. Hãy chọn ra

K số sao cho mỗi dòng có nhiều nhất 1 số ñược chọn, mỗi cột có nhiều nhất

1 số ñược chọn ñể tổng K số là lớn nhất.

Dữ liệu vào từ tệp văn bản SELECT.INP

- Dòng thứ nhất gồm 2 số N và x x

l

15

- N dòng sau, mỗi dòng N số nguyên không âm

( 10000

Kết quả ghi ra file văn bản SELECT.OUT

Tổng lớn nhất chọn ñược và số cách chọn (cách nhau ñúng một dấu cách)

121

Select.inp Select.out

3 2

1 2 3

2 3 1

3 1 2

6 3

4.43. Puzzle of numbers

Khi một số phần chữ số trong ñẳng thức ñúng của tổng hai số nguyên bị mất

(ñược thay bởi các dấu sao “\*”). Có một câu ñố là: Hãy thay các dấu sao bởi

các chữ số ñể cho ñẳng thức vẫn ñúng.

Ví dụ bắt ñầu từ ñẳng thức sau:

9334

789

--------

10123 (9334+789=10123)

Các ví dụ các chữ số bị mất ñược thay bằng các dấu sao như sau:

\*3\*4 hay \*\*\*\*

78\* \*\*\*

10123 \*\*\*\*\*

Nhiệm vụ của bạn là viết chương trình thay các dấu sao thành các chữ số ñể

ñược một ñẳng thức ñúng. Nếu có nhiều lời giải thì ñưa ra một trong số ñó.

Nếu không có thì ñưa ra thông báo: “No Solution”.

Chú ý các chữ số ở ñầu mỗi số phải khác 0.

Dữ liệu vào trong file “REBUSS.INP”: gồm 3 dòng, mỗi dòng là một xâu kí

tự gồm các chữ số hoặc kí tư “\*” . ðộ dài mỗi xâu không quá 50 kí tự.

Dòng 1, dòng 2 thể hiện là hai số ñược cộng, dòng 3 thể hiện là tổng hai số.

Kết quả ra file “REBUSS.OUT”: Nếu có lời giải thì file kết quả gồm 3

dòng tương ứng với file dữ liệu vào, nếu không thì thông báo “No Solution”

REBUSS.INP REBUSS.OUT

\*3\*4

78\*

10123

9334

789

10123

122

4.44. Xếp lịch giảng

Một giáo viên cần giảng ' vấn ñề ñược ñánh số từ 1 ñến ' '

10000.

Mỗi một vấn ñề \_x0016\_ cần có thời gian là !

\_x0016\_ \_x0010\_ 1. . '. ðể giảng ' vấn ñề ñó

thì giáo viên có các buổi ñã ñược phân có ñộ dài là  

500.

• Một vấn ñề thì phải giải quyết trong một buổi .

• Vấn ñề \_x0016\_ phải ñược giảng trước vấn \_x0016\_ + 1 với mọi \_x0016\_ \_x0010\_ 1. . ' ) 1 .

Học sinh có thể ra về sớm nếu như buổi giảng ñã kết thúc, tuy nhiên nếu

thời gian ra về ñó quá sớm so với buổi giảng thì thật là phí. Chính vì thế

người ta ñánh giá buổi lên lớp bằng giá trị jn như sau :

jn \_x0010\_ 7 0 ' ; ! \_x0010\_ 0

). ' ; 1

!

10

! ) 10

' ; ! d 10

Trong ñó ! là thời gian thừa của buổi lên lớp ñó, . là một hằng số .

Yêu cầu: Hãy xếp lịch dạy sao cho tổng số các buổi là cần ít nhất có thể

ñược. Trong các lịch dạy ít nhất ñó, hãy tìm lịch dạy sao cho tổng số jn là

nhỏ nhất có thể ñược.

Dữ liệu vào từ file SCHEDULING.INP

- Dòng ñầu là số n (số vấn ñề cần giảng) .

- Dòng tiếp theo là L và C

- Dòng cuối cùng là N số thể hiện cho !, !, … , !\_x0007\_ .

Kết quả ra file SCHEDULING.OUT

- Dòng ñầu tiên là số buổi .

- Dòng tiếp theo là tổng jn nhỏ nhất ñạt ñược .

SCHEDULING.INP SCHEDULING.OUT

10

120 10

80 80 10 50 30 20 40 30 120 100

6

2700

4.45. Khu vườn (IOI 2008)

Ramsesses II thắng trận trở về. ðể ghi nhận chiến tích của mình ông quyết

ñịnh xây một khu vườn tráng lệ. Khu vườn phải có một hàng cây chạy dài từ

cung ñiện của ông tại Luxor tới thánh ñường Karnak. Hàng cây này chỉ

chứa hai loại cây là sen và cói giấy, bởi vì chúng tương ứng là biểu tượng

của miền Thượng Ai Cập và Hạ Ai Cập.

123

Vườn phải có ñúng N cây. Ngoài ra, phải có sự cân bằng: ở mọi ñoạn cây

liên tiếp của vườn, số lượng sen và số lượng cói giấy phải không lệch nhau

quá 2.

Vườn cây ñược biểu diễn dưới dạng xâu các kí tự 'L' (lotus – sen) và 'P'

(papyrus – cói giấy). Ví dụ, với N = 5 có tất cả 14 vườn ñảm bảo cân bằng.

Theo thứ tự từ ñiển, các vườn ñó là: LLPLP, LLPPL, LPLLP, LPLPL,

LPLPP, LPPLL, LPPLP, PLLPL, PLLPP, PLPLL, PLPLP, PLPPL, PPLLP

và PPLPL.

Các vườn cân bằng với ñộ dài xác ñịnh cho trước ñược sắp xếp theo thứ tự

từ ñiển và ñược ñánh số từ 1 trở ñi. Ví dụ, với N=5, vườn số 12 sẽ là vườn

PLPPL.

NHIỆM VỤ

Cho số cây N và xâu biểu diễn một vườn cân bằng, hãy lập trình tính số thứ

tự của vườn này theo moñun M, trong ñó M là số nguyên cho trước.

Lưu ý rằng giá trị của M không ñóng vai trò quan trọng trong việc giải bài

toán, nó chỉ làm cho việc tính toán trở nên ñơn giản.

HẠN CHẾ

1 <= N <= 1 000 000

7 <= M <= 10 000 000

CHẤM ðIỂM

Có 40 ñiểm dành cho các dữ liệu vào với N không vượt quá 40.

INPUT

Chương trình của bạn phải ñọc từ file “GARDEN.INP” các dữ liệu sau:

• Dòng 1 chứa số nguyên N, số cây trong vườn,

• Dòng 2 chứa số nguyên M,

• Dòng 3 chứa xâu gồm N kí tự 'L' (sen) hoặc 'P' (cói giấy) biểu diễn

vườn cân bằng.

OUTPUT

Chương trình của bạn phải ghi ra file “GARDEN.OUT” một dòng chứa một

số nguyên trong phạm vi từ 0 ñến M-1, là số thứ tự tự theo môñun M của

vườn ñược mô tả trong ñầu vào.

Input ví

dụ 1

Output ví

dụ 1

Giải thích

124

5

7

PLPPL

5 Số thứ tự của PLPPL là 12. Như vậy

output là 12 theo môñun 7, tức là

5.

Input ví dụ 2 Output ví dụ 2

12

10000

LPLLPLPPLPLL

39

4.46. Số rõ ràng

Bờm mới tìm ñược một tài liệu ñịnh nghĩa số rõ ràng như sau: Với số

nguyên dương ', ta tạo số mới bằng cách lấy tổng bình phương các chữ số

của nó, với số mới này ta lại lặp lại công việc trên. Nếu trong quá trình ñó,

ta nhận ñược số mới là 1, thì số ' ban ñầu ñược gọi là số rõ ràng. Ví dụ, với

n = 19, ta có:

19 82 (= 12

+92

) 68 100 1

Như vậy, 19 là số rõ ràng.

Không phải mọi số ñều rõ ràng. Ví dụ, với n = 12, ta có:

12 5 25 29 85 89 145 42 20 4 16 37 58

89 145

Bờm rất thích thú với ñịnh nghĩa số rõ ràng này và thách ñố phú ông: Cho

một số nguyên dương ', tìm số ' là số rõ ràng liền sau số ', tức là '

là số rõ ràng nhỏ nhất lớn hơn '. Tuy nhiên, câu hỏi ñó quá dễ với phú ông

và phú ông ñã ñố lại Bờm: Cho hai số nguyên dương ' và k (1

',

10R), hãy tìm số

&

' \_x0010\_ … '\_x0019\_ là số rõ ràng liền sau thứ

của '.

Bạn hãy giúp Bờm giải câu ñố này nhé!

Dữ liệu vào từ file văn bản CLEAR.INP có dạng:

- Dòng ñầu là số ! 0 ( !

20

- ! dòng sau, mỗi dòng chứa 2 số nguyên ' và .

Kết quả ra file văn bản CLEAR.OUT gồm ! dòng, mỗi dòng là kết quả

tương ứng với dữ liệu vào.

125

CLEAR.INP CLEAR.OUT

2

18 1

1 145674807

19

1000000000

4.47. Hái nấm

Bé Bông ñi hái nấm trong N khu rừng ñánh số từ 1 ñến N, nhưng chỉ có M

khu rừng có nấm. Việc di chuyển từ khu rừng thứ i sang khu rừng thứ j tốn

tij ñơn vị thời gian. ðến khu rừng i có nấm, cô bé có thể dừng lại ñể hái

nấm. Nếu tổng số ñơn vị thời gian cô bé dừng lại ở khu rừng thứ i là di

(di>0), thì cô bé hái ñược: 

h



 + h



 + . . +  h



h

 cây nấm tại khu rừng ñó

(trong ñó Si

là số lượng nấm có tại khu rừng i, [x] là phần nguyên của x).

Giả thiết rằng ban ñầu cô bé ở khu rừng thứ nhất và ñi hái nấm trong thời

gian không quá P ñơn vị.

Yêu cầu: Hãy tính số lượng cây nấm nhiều nhất mà cô bé có thể hái ñược.

Dữ liệu vào từ file văn bản MUSHROOM.INP:

• Dòng ñầu tiên chứa ba số nguyên dương M (M ≤ 10), N (0<M≤ N ≤

100) và P (P ≤ 10000);

• M dòng tiếp theo, mỗi dòng chứa 2 số nguyên dương r và Sr

nghĩa là

khu rừng r có Sr

nấm (Sr

≤ 109

);

• Dòng thứ i trong N dòng cuối cùng chứa N số nguyên dương tij (tij ≤

10000), (i, j=1, ..., N ).

Kết quả ghi ra file văn bản MUSHROOM.OUT: số lượng cây nấm nhiều

nhất bé Bông có thể hái ñược.

MUSHROOM.INP MUSHROOM.OUT

2 2 2

1 5

2 10

0 3

3 0

3

126

Chuyên ñề 5

CÁC THUẬT TOÁN

TRÊN ðỒ THỊ

Trên thực tế có nhiều bài toán liên quan tới một tập các

ñối tượng và những mối liên hệ giữa chúng, ñòi hỏi

toán học phải ñặt ra một mô hình biểu diễn một cách

chặt chẽ và tổng quát bằng ngôn ngữ kí hiệu, ñó là ñồ

thị: một mô hình toán học gồm các ñỉnh biểu diễn các

ñối tượng và các cạnh biểu diễn mối quan hệ giữa các

ñối tượng.

Những ý tưởng cơ bản của ñồ thị ñược ñưa ra từ thế kỉ

thứ XVIII bởi nhà toán học Thuỵ Sĩ Leonhard Euler,

năm 1736, ông ñã dùng mô hình ñồ thị ñể giải bài toán về bảy cây cầu

Königsberg (Seven Bridges of Königsberg). Bài toán này cùng với bài

toán mã ñi tuần (Knight Tour) ñược coi là những bài toán ñầu tiên của lí

thuyết ñồ thị.

Rất nhiều bài toán của lí thuyết ñồ thị ñã trở thành nổi tiếng và thu hút

ñược sự quan tâm lớn của cộng ñồng nghiên cứu. Ví dụ bài toán bốn

màu, bài toán ñẳng cấu ñồ thị, bài toán người du lịch, bài toán người

ñưa thư Trung Hoa, bài toán ñường ñi ngắn nhất, luồng cực ñại trên

mạng v.v… Trong phạm vi một chuyên ñề, không thể trình bày tất cả

những gì ñã phát triển trong suốt gần 300 năm, chúng ta sẽ xem xét lí

thuyết ñồ thị dưới góc ñộ người lập trình, tức là khảo sát những thuật

toán cơ bản nhất có thể dễ dàng cài ñặt trên máy tính một số ứng dụng

của nó. Công việc của người lập trình là ñọc hiểu ñược ý tưởng cơ bản

của thuật toán và cài ñặt ñược chương trình trong bài toán tổng quát

cũng như trong trường hợp cụ thể.

Leonhard Euler

1707-1783

127

1. Các khái niệm cơ bản

1.1. ðồ thị

ðồ thị là mô hình biểu diễn một tập các ñối tượng và mối quan hệ hai ngôi giữa

các ñối tượng:

\_x0007\_

\_x0010\_

\_x0012\_\_x0010\_

,

Có thể ñịnh nghĩa ñồ thị G là một cặp , : , . Trong ñó là tập các

ñỉnh (vertices) biểu diễn các ñối tượng và gọi là tập các cạnh (edges) biểu diễn

mối quan hệ giữa các ñối tượng. Chúng ta quan tâm tới mối quan hệ hai ngôi

(pairwise relations) giữa các ñối tượng nên có thể coi là tập các cặp , với

và là hai ñỉnh của biểu diễn hai ñối tượng có quan hệ với nhau.

Một số hình ảnh của ñồ thị:

Hình 5-1. Ví dụ về mô hình ñồ thị

Có thể phân loại ñồ thị , theo ñặc tính và số lượng của tập các cạnh :

• ñược gọi là ñơn ñồ thị (hay gọi tắt là ñồ thị) nếu giữa hai ñỉnh , có

nhiều nhất là 1 cạnh trong nối từ tới .

• ñược gọi là ña ñồ thị (multigraph) nếu giữa hai ñỉnh , có thể có

nhiều hơn 1 cạnh trong nối và (Hiển nhiên ñơn ñồ thị cũng là ña ñồ thị).

Nếu có nhiều cạnh nối giữa hai ñỉnh , thì những cạnh ñó ñược gọi là

cạnh song song (parallel edges)

• ñược gọi là ñồ thị vô hướng (undirected graph) nếu các cạnh trong là

không ñịnh hướng, tức là cạnh nối hai ñỉnh , bất kì cũng là cạnh nối

hai ñỉnh , . Hay nói cách khác, tập gồm các cặp , không tính thứ tự:

, , .

• ñược gọi là ñồ thị có hướng (directed graph) nếu các cạnh trong là có

ñịnh hướng, tức là có thể có cạnh nối từ ñỉnh tới ñỉnh nhưng chưa chắc ñã

có cạnh nối từ ñỉnh tới ñỉnh . Hay nói cách khác, tập gồm các cặp ,

128

có tính thứ tự: , \_x001D\_ , . Trong ñồ thị có hướng, các cạnh còn ñược gọi

là các cung (arcs). ðồ thị vô hướng cũng có thể coi là ñồ thị có hướng nếu

như ta coi cạnh nối hai ñỉnh , bất kì tương ñương với hai cung , và

, .

Hình 5-2 là ví dụ về ñơn ñồ thị/ña ñồ thị có hướng/vô hướng.

Hình 5-2. Phân loại ñồ thị

1.2. Các khái niệm

Như trên ñịnh nghĩa ñồ thị , là một cấu trúc rời rạc, tức là các tập và

là tập không quá ñếm ñược, vì vậy ta có thể ñánh số thứ tự 1, 2, 3… cho các

phần tử của tập V và và ñồng nhất các phần tử của tập và với số thứ tự của

chúng. Hơn nữa, ñứng trên phương diện người lập trình cho máy tính thì ta chỉ

quan tâm ñến các ñồ thị hữu hạn ( và là tập hữu hạn) mà thôi, chính vì vậy từ

ñây về sau, nếu không chú thích gì thêm thì khi nói tới ñồ thị, ta hiểu rằng ñó là

ñồ thị hữu hạn.

a) Cạnh liên thuộc, ñỉnh kề, bậc

ðối với ñồ thị vô hướng , . Xét một cạnh

, nếu

, thì ta

nói hai ñỉnh và là kề nhau (adjacent) và cạnh

này liên thuộc (incident) với

ñỉnh và ñỉnh .

Với một ñỉnh trong ñồ thị vô hướng, ta ñịnh nghĩa bậc (degree) của , kí hiệu

deg là số cạnh liên thuộc với . Trên ñơn ñồ thị thì số cạnh liên thuộc với

cũng là số ñỉnh kề với .

Vô hướng

ðơn ñồ thị ða ñồ thị

Có hướng Vô hướng Có hướng

129

ðịnh lí 5-1

Giả sử , là ñồ thị vô hướng, khi ñó tổng tất cả các bậc ñỉnh

trong sẽ bằng hai lần số cạnh:

!deg

"#

2||

(1)

Chứng minh

Khi lấy tổng tất cả các bậc ñỉnh tức là mỗi cạnh

, sẽ ñược tính một lần

trong deg và một lần trong deg. Từ ñó suy ra kết quả.

Hệ quả

Trong ñồ thị vô hướng, số ñỉnh bậc lẻ là số chẵn.

ðối với ñồ thị có hướng , . Xét một cung

, nếu

, thì ta

nói nối tới và nối từ , cung

là ñi ra khỏi ñỉnh và ñi vào ñỉnh . ðỉnh

khi ñó ñược gọi là ñỉnh ñầu, ñỉnh ñược gọi là ñỉnh cuối của cung

.

Với mỗi ñỉnh trong ñồ thị có hướng, ta ñịnh nghĩa: Bán bậc ra (out-degree) của

kí hiệu deg& là số cung ñi ra khỏi nó; bán bậc vào (in-degree) kí hiệu

deg' là số cung ñi vào ñỉnh ñó.

ðịnh lí 5-2

Giả sử , là ñồ thị có hướng, khi ñó tổng tất cả các bán bậc ra

của các ñỉnh bằng tổng tất cả các bán bậc vào và bằng số cung của ñồ thị

!deg&

"#

!deg' ||

"#

(2)

Chứng minh

Khi lấy tổng tất cả các bán bậc ra hay bán bậc vào, mỗi cung , sẽ ñược tính

ñúng một lần trong deg& và cũng ñược tính ñúng một lần trong deg'. Từ ñó

suy ra kết quả.

b) ðường ñi và chu trình

Một dãy các ñỉnh:

( )\*, +, … , -

.

sao cho /'+, /

, 0\_x0012\_: 1 3 \_x0012\_ 3 4 ñược gọi là một ñường ñi (path), ñường ñi

này gồm 4 1 ñỉnh \*, +, … , - và 4 cạnh \*, +

, +, 5

, … , -'+, -

. Nếu

có một ñường ñi như trên thì ta nói - ñến ñược (reachable) từ \* hay \* ñến ñược

-, kí hiệu \* 6 -. ðỉnh \* ñược gọi là ñỉnh ñầu và ñỉnh - gọi là ñỉnh cuối của

ñường ñi (. Các ñỉnh +, 5, … , -'+ ñược gọi là ñỉnh trong của ñường ñi (

130

Một ñường ñi gọi là ñơn giản (simple) hay ñường ñi ñơn nếu tất cả các ñỉnh trên

ñường ñi là hoàn toàn phân biệt (dĩ nhiên khi ñó các cạnh trên ñường ñi cũng hoàn

toàn phân biệt). ðường ñi ( )\*, +, … , -

. trở thành chu trình (circuit) nếu

\* -. Trên ñồ thị có hướng, chu trình ( ñược gọi là chu trình ñơn nếu nó có ít

nhất một cung và các ñỉnh +, 5, … , - hoàn toàn phân biệt. Trên ñồ thị vô

hướng, chu trình ( ñược gọi là chu trình ñơn nếu 4 7 3 và các ñỉnh +, 5, … , -

hoàn toàn phân biệt.

c) Một số khái niệm khác

ðẳng cấu

Hai ñồ thị , và

9

9

, 9

ñược gọi là ñẳng cấu (isomorphic) nếu tồn

tại một song ánh :: ; < sao cho số cung nối với trên bằng số cung nối

: với : trên

9

.

ðồ thị con

ðồ thị

9

9

, 9

là ñồ thị con (subgraph) của ñồ thị , nếu < = và

< = .

ðồ thị con > ?, > ñược gọi là ñồ thị con cảm ứng (induced graph) từ ñồ thị

bởi tập ? = nếu > @, : , ?A trong trường hợp này chúng ta

còn nói > là ñồ thị hạn chế trên ?.

Phiên bản có hướng/vô hướng

Với một ñồ thị vô hướng , , ta gọi phiên bản có hướng (directed

version) của là một ñồ thị có hướng

9 , < tạo thành từ bằng cách thay

mỗi cạnh , bằng hai cung có hướng ngược chiều nhau: , và , .

Với một ñồ thị có hướng , , ta gọi phiên bản vô hướng (undirected

version) của là một ñồ thị vô hướng

9 , < tạo thành bằng cách thay mỗi

cung , bằng cạnh vô hướng , . Nói cách khác,

9

tạo thành từ bằng

cách bỏ ñi chiều của cung.

Tính liên thông

Một ñồ thị vô hướng gọi là liên thông (connected) nếu giữa hai ñỉnh bất kì của ñồ

thị có tồn tại ñường ñi. ðối với ñồ thị có hướng, có hai khái niệm liên thông tuỳ

theo chúng ta có quan tâm tới hướng của các cung hay không. ðồ thị có hướng

gọi là liên thông mạnh (strongly connected) nếu giữa hai ñỉnh bất kì của ñồ thị có

tồn tại ñường ñi. ðồ thị có hướng gọi là liên thông yếu (weakly connected) nếu

phiên bản vô hướng của nó là ñồ thị liên thông.

131

ðồ thị ñầy ñủ

Một ñồ thị vô hướng ñược gọi là ñầy ñủ (complete) nếu mọi cặp ñỉnh ñều là kề

nhau, ñồ thị ñầy ñủ gồm ñỉnh kí hiệu là BC. Hình 5-3 là ví dụ về các ñồ thị ñầy

ñủ BD, BE và BF.

Hình 5-3. ðồ thị ñầy ñủ

ðồ thị hai phía

Một ñồ thị vô hướng gọi là hai phía (bipartite) nếu

tập ñỉnh của nó có thể chia làm hai tập rời nhau G,

H sao cho không tồn tại cạnh nối hai ñỉnh thuộc G

cũng như không tồn tại cạnh nối hai ñỉnh thuộc H.

Nếu |G| I và |H| và giữa mọi cặp ñỉnh

J, K trong ñó J G, K H ñều có cạnh nối thì ñồ

thị hai phía ñó ñược gọi là ñồ thị hai phía ñầy ñủ,

kí hiệu BL,C. Hình 5-4 là ví dụ về ñồ thị hai phía

ñầy ñủ B5,D.

Hình 5-4. ðồ thị hai phía ñầy ñủ

ðồ thị phẳng

Một ñồ thị ñược gọi là ñồ thị phẳng (planar graph) nếu chúng ta có thể vẽ ñồ thị

ra trên mặt phẳng sao cho:

• Mỗi ñỉnh tương ứng với một ñiểm trên mặt phẳng, không có hai ñỉnh cùng

toạ ñộ.

• Mỗi cạnh tương ứng với một ñoạn ñường liên tục nối hai ñỉnh, các ñiểm nằm

trên hai cạnh bất kì là không giao nhau ngoại trừ các ñiểm ñầu mút (tương

ứng với các ñỉnh)

Phép vẽ ñồ thị phẳng như vậy gọi là biểu diễn phẳng của ñồ thị

Ví dụ như ñồ thị ñầy ñủ BE là ñồ thị phẳng bởi nó có thể vẽ ra trên mặt phẳng như

Hình 5-5

BD BE BF

132

Hình 5-5. Hai cách vẽ ñồ thị phẳng của MN

ðịnh lí 5-3 (ðịnh lí Kuratowski)

Một ñồ thị vô hướng là ñồ thị phẳng nếu và chỉ nếu nó không chứa ñồ thị

con ñẳng cấu với BD,D hoặc BF.

ðịnh lí 5-4 (Công thức Euler)

Nếu một ñồ thị vô hướng liên thông là ñồ thị phẳng và biểu diễn phẳng

của ñồ thị ñó gồm ñỉnh và

cạnh chia mặt phẳng thành : phần thì

O

: 2.

ðịnh lí 5-5

Nếu ñơn ñồ thị vô hướng , là ñồ thị phẳng có ít nhất 3 ñỉnh thì

|| 3 3|| O 6. Ngoài ra nếu không có chu trình ñộ dài 3 thì

|| 3 2|| O 4.

ðịnh lí 5-5 chỉ ra rằng số cạnh của ñơn ñồ thị phẳng là một ñại lượng

|| Ο|| ñiều này rất hữu ích ñối với nhiều thuật toán trên ñồ thị thưa (có ít

cạnh).

ðồ thị ñường

Từ ñồ thị vô hướng , ta xây dựng ñồ thị vô hướng

9

như sau: Mỗi ñỉnh của

9

tương ứng với một cạnh của , giữa hai ñỉnh J, K của

9

có cạnh nối nếu và chỉ

nếu tồn tại ñỉnh liên thuộc với cả hai cạnh J, K trên . ðồ thị

9

như vậy ñược gọi

là ñồ thị ñường của ñồ thị . ðồ thị ñường ñược nghiên cứu trong các bài toán

kiểm tra tính liên thông, tập ñộc lập cực ñại, tô màu cạnh ñồ thị, chu trình Euler và

chu trình Hamilton v.v…

2. Biểu diễn ñồ thị

Khi lập trình giải các bài toán ñược mô hình hoá bằng ñồ thị, việc ñầu tiên cần

làm tìm cấu trúc dữ liệu ñể biểu diễn ñồ thị sao cho việc giải quyết bài toán ñược

thuận tiện nhất.

133

Có rất nhiều phương pháp biểu diễn ñồ thị, trong bài này chúng ta sẽ khảo sát một

số phương pháp phổ biến nhất. Tính hiệu quả của từng phương pháp biểu diễn sẽ

ñược chỉ rõ hơn trong từng thuật toán cụ thể.

2.1. Ma trận kề

Với , là một ñơn ñồ thị có hướng trong ñó || , ta có thể ñánh số

các ñỉnh từ 1 tới và ñồng nhất mỗi ñỉnh với số thứ tự của nó. Bằng cách ñánh số

như vậy, ñồ thị có thể biểu diễn bằng ma trận vuông S T/UV

CWC

. Trong ñó:

/U X1, nếu \_x0012\_,

0, nếu \_x0012\_, \

Với 0\_x0012\_, giá trị của các phần tử trên ñường chéo chính ma trận S: @//A có thể ñặt

tuỳ theo mục ñích cụ thể, chẳng hạn ñặt bằng 0. Ma trận S xây dựng như vậy

ñược gọi là ma trận kề (adjacency matrix) của ñồ thị . Việc biểu diễn ñồ thị vô

hướng ñược quy về việc biểu diễn phiên bản có hướng tương ứng: thay mỗi cạnh

\_x0012\_, bởi hai cung ngược hướng nhau: \_x0012\_, và , \_x0012\_.

ðối với ña ñồ thị thì việc biểu diễn cũng tương tự trên, chỉ có ñiều nếu như \_x0012\_,

là cung thì /U là số cạnh nối giữa ñỉnh \_x0012\_ và ñỉnh .

Hình 5-6. Ma trận kề biểu diễn ñồ thị

Trong trường hợp là ñơn ñồ thị, ta có thể biểu diễn ma trận kề S tương ứng là

các phần tử lôgic:

/U XTrue, nếu \_x0012\_,

False, nếu \_x0012\_, \

Có một cách khác biểu diễn ñồ thị vô hướng bằng ma trận S T/UV

CWC

như sau:

/U ddeg\_x0012\_ , nếu \_x0012\_

O1, nếu \_x0012\_,

0, TH khác

Cách biểu diễn này có ứng dụng trong một số bài toán ñồ thị, gọi là biểu diễn bằng ma trận

Laplace (Laplacian matrix hay Kirchhoff matrix)

1

2 3

4 5

1

2 3

e 4 5

f

g

0 1 1 0 0

1 0 1 1 1

1 1 0 0 1

0 1 0 0 1

0 1 1 1 0h

i

j

e

f

g

0 1 1 0 0

0 0 1 1 1

0 0 0 0 1

0 0 0 0 1

0 0 0 0 0h

i

j

134

Ma trận kề có một số tính chất:

• ðối với ñồ thị vô hướng , thì ma trận kề tương ứng là ma trận ñối xứng

/U U/, ñiều này không ñúng với ñồ thị có hướng.

• Nếu là ñồ thị vô hướng và S là ma trận kề tương ứng thì trên ma trận S,

tổng các số trên hàng \_x0012\_ bằng tổng các số trên cột \_x0012\_ và bằng bậc của ñỉnh \_x0012\_:

deg \_x0012\_

• Nếu là ñồ thị có hướng và S là ma trận kề tương ứng thì trên ma trận S,

tổng các số trên hàng \_x0012\_ bằng bán bậc ra của ñỉnh \_x0012\_: deg&\_x0012\_, tổng các số trên

cột \_x0012\_ bằng bán bậc vào của ñỉnh \_x0012\_: deg'\_x0012\_

Ưu ñiểm của ma trận kề:

• ðơn giản, trực quan, dễ cài ñặt trên máy tính

• ðể kiểm tra xem hai ñỉnh , của ñồ thị có kề nhau hay không, ta chỉ việc

kiểm tra bằng một phép so sánh: k" \_x001D\_ 0

Nhược ñiểm của ma trận kề

• Bất kể số cạnh của ñồ thị là nhiều hay ít, ma trận kề luôn luôn ñòi hỏi

5

ô

nhớ ñể lưu các phần tử ma trận, ñiều ñó gây lãng phí bộ nhớ.

• Một số bài toán yêu cầu thao tác liệt kê tất cả các ñỉnh kề với một ñỉnh

cho trước. Trên ma trận kề việc này ñược thực hiện bằng cách xét tất cả các

ñỉnh và kiểm tra ñiều kiện k" \_x001D\_ 0. Như vậy, ngay cả khi ñỉnh là ñỉnh cô

lập (không kề với ñỉnh nào) hoặc ñỉnh treo (chỉ kề với 1 ñỉnh) ta cũng buộc

phải xét tất cả các ñỉnh và kiểm tra giá trị tương ứng k".

2.2. Danh sách cạnh

Hình 5-7. Danh sách cạnh

Với ñồ thị , có ñỉnh, I cạnh, ta có thể liệt kê tất cả các cạnh của ñồ

thị trong một danh sách, mỗi phần tử của danh sách là một cặp J, K tương ứng

với một cạnh của , trong trường hợp ñồ thị có hướng thì mỗi cặp J, K tương

1

2 3

4 5

(1,2) (1,3) (2,3) (2,4) (2,5) (3,5) (4,5)

1 2 3 4 5 6 7

135

ứng với một cung, J là ñỉnh ñầu và K là ñỉnh cuối của cung. Cách biểu diễn này

gọi là danh sách cạnh (edge list).

Có nhiều cách xây dựng cấu trúc dữ liệu ñể biểu diễn danh sách, nhưng phổ biến

nhất là dùng mảng hoặc danh sách móc nối.

Ưu ñiểm của danh sách cạnh:

• Trong trường hợp ñồ thị thưa (có số cạnh tương ñối nhỏ), cách biểu diễn bằng

danh sách cạnh sẽ tiết kiệm ñược không gian lưu trữ, bởi nó chỉ cần ΟI ô

nhớ ñể lưu danh sách cạnh.

• Trong một số trường hợp, ta phải xét tất cả các cạnh của ñồ thị thì cài ñặt trên

danh sách cạnh làm cho việc duyệt các cạnh dễ dàng hơn. (Thuật toán Kruskal

chẳng hạn)

Nhược ñiểm của danh sách cạnh:

• Nhược ñiểm cơ bản của danh sách cạnh là khi ta cần duyệt tất cả các ñỉnh kề

với ñỉnh nào ñó của ñồ thị, thì chẳng có cách nào khác là phải duyệt tất cả

các cạnh, lọc ra những cạnh có chứa ñỉnh và xét ñỉnh còn lại.

• Việc kiểm tra hai ñỉnh , có kề nhau hay không cũng bắt buộc phải duyệt

danh sách cạnh, ñiều ñó khá tốn thời gian trong trường hợp ñồ thị dày

(nhiều cạnh).

2.3. Danh sách kề

ðể khắc phục nhược ñiểm của các phương pháp ma trận kề và danh sách cạnh,

người ta ñề xuất phương pháp biểu diễn ñồ thị bằng danh sách kề (adjacency list).

Trong cách biểu diễn này, với mỗi ñỉnh của ñồ thị, ta cho tương ứng với nó một

danh sách các ñỉnh kề với .

Với ñồ thị có hướng , . gồm ñỉnh và gồm I cung. Có hai cách cài

ñặt danh sách kề phổ biến:

• Forward Star: Với mỗi ñỉnh , lưu trữ một danh sách l mn chứa các ñỉnh

nối từ : l mn @: , A.

• Reverse Star: Với mỗi ñỉnh , lưu trữ một danh sách l mn chứa các ñỉnh

nối tới : l mn @: , A

Tùy theo từng bài toán, chúng ta sẽ chọn cấu trúc Forward Star hoặc Reverse Star

ñể biểu diễn ñồ thị. Có những bài toán yêu cầu phải biểu diễn ñồ thị bằng cả hai

cấu trúc Forward Star và Reverse Star.

136

Việc biểu diễn ñồ thị vô hướng ñược quy về việc biểu diễn phiên bản có hướng

tương ứng: thay mỗi cạnh , bởi hai cung có hướng ngược nhau: , và

, .

Bất cứ cấu trúc dữ liệu nào có khả năng biểu diễn danh sách (mảng, danh sách

móc nối, cây…) ñều có thể sử dụng ñể biểu diễn danh sách kề, nhưng mảng và

danh sách móc nối ñược sử dụng phổ biến nhất.

a) Biểu diễn danh sách kề bằng mảng

Dùng một mảng l m1 … In chứa các ñỉnh, mảng ñược chia làm ñoạn, ñoạn thứ

trong mảng lưu danh sách các ñỉnh kề với ñỉnh . ðể biết một ñoạn nằm từ chỉ

số nào ñến chỉ số nào, ta có một mảng

lm1 … 1n ñánh dấu vị trí phân

ñoạn:

lmn sẽ bằng chỉ số ñứng liền trước ñoạn thứ , quy ước

lm

1n I. Khi ñó các phần tử trong ñoạn:

l o

lmn 1 …

lm 1np

là các ñỉnh kề với ñỉnh .

Nhắc lại rằng khi sử dụng danh sách kề ñể biểu diễn ñồ thị vô hướng, ta quy nó về

ñồ thị có hướng và số cung I ñược nhân ñôi (Hình 5-8).

Hình 5-8. Dùng mảng biểu diễn danh sách kề

b) Biểu diễn danh sách kề bằng các danh sách móc nối

Hình 5-9. Biểu danh sách kề bởi các danh sách móc nối

1 2

3

4

5

q\_x0012\_

m1n 2 3

q\_x0012\_

m2n 1 3 4 5

q\_x0012\_

m3n 1 2 5

q\_x0012\_

m4n 2 5

q\_x0012\_

m5n 2 3 4

1 2

3

4

5

1

2

2

3

3

1

4

3

5

4

6

5

7

1

8

2

9

5

10

2

11

5

12

2

13

3

14

l : 4

1 2 3 4 5

1

0

2

2

3

6

4

9

5

11

6

l: 14

137

Trong cách biểu diễn này, ta cho tương ứng mỗi ñỉnh của ñồ thị với q\_x0012\_

mn là

chốt của một danh sách móc nối gồm các ñỉnh kề với .

Ưu ñiểm của danh sách kề

• ðối với danh sách kề, việc duyệt tất cả các ñỉnh kề với một ñỉnh cho trước là

hết sức dễ dàng, cái tên “danh sách kề” ñã cho thấy rõ ñiều này.

• Việc duyệt tất cả các cạnh cũng ñơn giản vì một cạnh thực ra là nối một ñỉnh

với một ñỉnh khác kề nó.

Nhược ñiểm của danh sách kề

• Danh sách kề yếu hơn ma trận kề ở việc kiểm tra , có phải là cạnh hay

không, bởi trong cách biểu diễn này ta sẽ phải việc phải duyệt toàn bộ danh

sách kề của hay danh sách kề của .

2.4. Danh sách liên thuộc

Danh sách liên thuộc (incidence lists) là một mở rộng của danh sách kề. Nếu như

trong biểu diễn danh sách kề, mỗi ñỉnh ñược cho tương ứng với một danh sách các

ñỉnh kề thì trong biểu diễn danh sách liên thuộc, mỗi ñỉnh ñược cho tương ứng với

một danh sách các cạnh liên thuộc. Chính vì vậy, những kĩ thuật cài ñặt danh sách

kề có thể sửa ñổi một chút ñể cài ñặt danh sách liên thuộc.

ðặc biệt trong trường hợp ñồ thị có hướng, ta có thể xây dựng danh sách liên

thuộc từ danh sách cạnh tương ñối dễ dàng bằng cách bổ sung các con trỏ liên kết.

Giả sử ñồ thị có hướng , có ñỉnh và I cung ñược biểu diễn bởi danh

sách cạnh

m1 … In. Vì ñồ thị có hướng, nếu ta cho tương ứng mỗi ñỉnh một

danh sách các cung ñi ra khỏi (forward star) thì sẽ có tổng cộng danh sách liên

thuộc và mỗi cung chỉ xuất hiện trong ñúng một danh sách liên thuộc. Vì vậy ta có

thể bổ sung hai mảng

lm1 … n và s\_x0012\_4m1 … In trong ñó:

•

lmn là chỉ số cung ñầu tiên trong danh sách liên thuộc của ñỉnh . Nếu

danh sách liên thuộc ñỉnh là t,

lmn ñược gán bằng 0.

• s\_x0012\_4m\_x0012\_n là chỉ số cung kế tiếp cung

m\_x0012\_n trong danh sách liên thuộc chứa cung

m\_x0012\_n. Trường hợp

m\_x0012\_n là cung cuối cùng của một danh sách liên thuộc, s\_x0012\_4m\_x0012\_n

ñược gán bằng 0

ðể duyệt tất cả những cung ñi ra khỏi một ñỉnh nào ñó, ta có thể thực hiện dễ

dàng bằng thuật toán sau:

i := head[u];

while i ≠ 0 do

begin

138

«Xử lí cung e[i]»;

i := link[i];

end;

2.5. Chuyển ñổi giữa các cách biểu diễn ñồ thị

Có một số thuật toán mà tính hiệu quả của nó phụ thuộc rất nhiều vào cách thức

biểu diễn ñồ thị, do ñó khi bắt tay vào giải quyết một bài toán ñồ thị, chúng ta

phải tìm cấu trúc dữ liệu phù hợp ñể biểu diễn ñồ thị sao cho hợp lý nhất. Nếu ñồ

thị ñầu vào ñược cho bởi một cách biểu diễn bất hợp lí, chúng ta cần chuyển ñổi

cách biểu diễn khác ñể thuận tiện trong việc triển khai thuật toán.

Ta xét bài toán chuyển ñối các cách biểu diễn ñơn ñồ thị có hướng , có

ñỉnh và I cạnh. Có thể biểu diễn ñồ thị này bởi:

Ma trận kề:

var

a: array[1..n, 1..n] of Boolean;

Danh sách cạnh:

type

TEdge = record

x, y: Integer;

end;

var

e: array[1..m] of TEdge;

Danh sách kề (forward star) (biểu diễn bằng mảng):

var

adj: array[1..2 \* m] of Integer;

head: array[1..n + 1] of Integer;

Danh sách liên thuộc (forward star) (biểu diễn bằng cấu trúc liên kết)

type

TEdge = record

x, y: Integer;

end;

var

e: array[1..m] of TEdge; //Danh sách cạnh

link: array[1..m] of Integer; //link[i]: chỉ số cạnh kế tiếp trong danh sách

liên thuộc

head: array[1..n] of Integer; //head[i]: chỉ số cạnh ñầu

tiên trong danh sách liên thuộc

139

a) Chuyển ñổi giữa ma trận kề và danh sách cạnh

Nếu ñồ thị ñược cho bởi ma trận kề S T/UV

CWC

trong ñó /U Trueu\_x0012\_,

, ta có thể xây dựng danh sách cạnh tương ứng bằng cách duyệt tất cả các cặp

\_x0012\_, , nếu /U True thì ñưa cặp này vào danh sách cạnh

.

k := 0;

for i := 1 to n do

for j := 1 to n do

if a[i, j] then

begin

k := k + 1;

e[k].x := i; e[k].y := j;

end;

Ngược lại, nếu ñồ thị cho bởi danh sách cạnh

, ta có thể xây dựng ma trận kề S

bằng cách khởi tạo các phần tử của S là False rồi duyệt danh sách cạnh, mỗi khi

duyệt qua cung J, K, ta ñặt vw x True.

for i := 1 to n do

for j := 1 to n do a[i, j] := False;

for k := 1 to m do

with e[k] do

a[x, y] := True;

b) Chuyển ñổi giữa ma trận kề và danh sách kề

Từ ma trận kề S T/UV

CWC

, ta có thể xây dựng hai mảng l m1 … In và

lm1 … 1n sao cho các phần tử trong mảng l từ chỉ số

lmn 1 tới

chỉ số

lm 1n chứa danh sách kề của ñỉnh . Hai mảng này là danh sách kề

dạng forward star của ñồ thị.

head[n + 1] := m;

for i := n downto 1 do

begin

head[i] := head[i + 1];

for j := n downto 1 do

if a[i, j] then

begin

adj[head[i]] := j;

head[i] := head[i] - 1;

end;

end;

140

Ngược lại, chúng ta có thể xây dựng ma trận kề từ danh sách kề theo cách: ðặt

các phần tử của ma trận kề S bằng ys

, sau ñó với mỗi ñỉnh , duyệt các ñỉnh

thuộc danh sách kề của nó và ñặt k" x z

.

for i := 1 to n do

for j := 1 to n do a[i, j] := False;

for u := 1 to n do

for k := head[u] + 1 to head[u + 1] do

a[u, adj[k]] := True;

c) Chuyển ñổi giữa danh sách cạnh và danh sách kề

Từ danh sách cạnh

, ta có thể xây dựng hai mảng l và

l tương ứng với

danh sách kề dạng forward star bằng thuật toán ñếm phân phối.

Trước hết, ta tính các

lmn là bậc của ñỉnh (0 ):

for u := 1 to n do head[u] := 0;

for i := 1 to m do

with e[i] do

head[x] := head[x] + 1;

Sau ñó, ta chia mảng l thành ñoạn, ñoạn thứ sẽ chứa các ñỉnh kề với

ñỉnh . ðể xác ñịnh vị trí các ñoạn này, ta ñặt mỗi

lmn trỏ tới vị trí cuối ñoạn

thứ :

for u := 2 to n do

head[u] := head[u - 1] + head[u];

Tiếp theo là duyệt lại danh sách cạnh, mỗi khi duyệt tới cạnh J, K ta ñưa K vào

mảng l tại vị trí

lmJn, ñưa J vào mảng l tại vị trí

lmKn ñồng thời

giảm hai con trỏ

lmJn và

lmKn ñi 1.

for i := m downto 1 do

with e[i] do

begin

adj[head[x]] := y; head[x] := head[x] - 1;

end;

ðến ñây, chúng ta có mảng l phân làm ñoạn, trong ñó

lmn là vị trí ñứng

liền trước ñoạn thứ . Việc cuối cùng là ñặt:

head[n + 1] := m;

Việc chuyển ñổi từ danh sách kề sang danh sách cạnh ñược thực hiện ñơn giản

hơn: Với mỗi ñỉnh , ta xét các ñỉnh thuộc danh sách kề của nó và ñưa ,

vào danh sách cạnh

.

141

i := 0;

for u := 1 to n do

for k := head[u] + 1 to head[u + 1] do

begin

v := adj[k];

i := i + 1;

e[i].x := u; e[i].y := v;

end;

d) Chuyển ñổi giữa danh sách cạnh và danh sách liên thuộc

Bởi danh sách liên thuộc ñược ñặc tả ñã bao gồm danh sách cạnh, ta chỉ quan tâm

tới vấn ñề chuyển ñổi từ danh sách cạnh thành danh sách liên thuộc.

Trước hết với mọi ñỉnh ta ñặt

lmn x 0 ñể khởi tạo danh sách liên thuộc

của bằng t.

for u := 1 to n do head[u] := 0;

Tiếp theo ta duyệt danh sách cạnh, mỗi khi duyệt qua một cung J, K ta móc nối

cung này vào danh sách liên thuộc các cung ñi ra khỏi J:

for i := m downto 1 do

with e[i] do

begin

link[i] := head[x];

head[x] := i;

end;

Với các bài toán mà chúng ta sẽ khảo sát, cũng có một số thuật toán không phụ

thuộc nhiều và cách biểu diễn ñồ thị, trong trường hợp này tôi sẽ chọn cấu trúc dữ

liệu dễ cài ñặt và trình bày nhất ñể việc ñọc hiểu thuật toán/chương trình ñược

thuận tiện hơn.

Bài tập

5.1. Cho một ñồ thị có hướng ñỉnh, I cạnh ñược biểu diễn bằng danh sách kề,

trong ñó mỗi ñỉnh sẽ ñược cho tương ứng với một danh sách các ñỉnh nối

từ . Cho một ñỉnh , hãy tìm thuật toán tính bán bậc ra và bán bậc vào của

. Xác ñịnh ñộ phức tạp tính toán của thuật toán

5.2. ðồ thị chuyển vị của ñồ thị có hướng , là ñồ thị

{ , {

,

trong ñó:

{ @, : , A

142

Hãy tìm thuật toán xây dựng

{

từ trong hai trường hợp: và

{

ñược

biểu diễn bằng ma trận kề; và

{

ñược biểu diễn bằng danh sách kề.

5.3. Cho ña ñồ thị vô hướng , ñược biểu diễn bằng danh sách kề, hãy

tìm thuật toán Ο|| || ñể xây dựng ñơn ñồ thị

9 , < và biểu

diễn < bằng danh sách kề, biết rằng ñồ thị < gồm tất cả các ñỉnh của ñồ thị

và các cạnh song song trên ñược thay thế bằng duy nhất một cạnh trong

<.

5.4. Cho ña ñồ thị ñược biểu diễn bằng ma trận kề S T/UV trong ñó /U là số

cạnh nối từ ñỉnh \_x0012\_ tới ñỉnh . Hãy chứng minh rằng S

-

là ma trận | T/UV

trong ñó /U là số ñường ñi từ ñỉnh \_x0012\_ tới ñỉnh qua ñúng 4 cạnh. Gợi ý: Sử

dụng chứng minh quy nạp.

5.5. Cho ñơn ñồ thị , , ta gọi bình phương của một ñồ thị là ñơn

ñồ thị

5 , 5

sao cho , 5

nếu và chỉ nếu tồn tại một ñỉnh } sao cho , }

và }, ñều thuộc . Hãy tìm thuật toán Ο||

D

ñể xây dựng

5

từ

trong trường hợp cả và

5

ñược biểu diễn bằng ma trận kề, tìm thuật toán

Ο|| ||

5

ñể xây dựng

5

từ trong trường hợp cả và

5

ñược biểu

diễn bằng danh sách kề.

5.6. Xây dựng cấu trúc dữ liệu ñể biểu diễn ñồ thị vô hướng và các thao tác:

• Liệt kê các ñỉnh kề với một ñỉnh cho trước trong thời gian Ο||

• Kiểm tra hai ñỉnh có kề nhau hay không trong thời gian Ο1

• Loại bỏ một cạnh trong thời gian Ο1

5.7. Với ñồ thị , ñược biểu diễn bằng ma trận kề, ña số các thuật toán

trên ñồ thị sẽ có ñộ phức tạp tính toán Ω||

5

, tuy nhiên không phải không

có ngoại lệ. Chẳng hạn bài toán tìm “bồn chứa” (universal sink) trong ñồ

thị: bồn chứa trong ñồ thị có hướng là một ñỉnh nối từ tất cả các ñỉnh khác

và không có cung ñi ra. Hãy tìm thuật toán \_x0007\_|| ñể xác ñịnh sự tồn tại và

chỉ ra bồn chứa trong ñồ thị có hướng.

5.8. Người ta còn có thể biểu diễn ñồ thị bằng ma trận liên thuộc (incidence

matrix): Với ñồ thị có hướng , có ñỉnh và I cung, ma trận liên

thuộc | T/UV của kích thước I W , trong ñó:

143

/U dO1, nếu cung thứ đi ra khỏi đỉnh \_x0012\_

1, nếu cung thứ ñi vào ñỉnh \_x0012\_

0, nếu cung thứ không liên thuộc với ñỉnh \_x0012\_

Xét |

{

là ma trận chuyển vị của ma trận |, hãy cho biết ý nghĩa của ma trận

tích ||{

3. Các thuật toán tìm kiếm trên ñồ thị

3.1. Bài toán tìm ñường

Cho ñồ thị , và hai ñỉnh

, .

Nhắc lại ñịnh nghĩa ñường ñi: Một dãy các ñỉnh:

( )

\*, +, … , - ., 0\_x0012\_: /'+, /

ñược gọi là một ñường ñi từ

tới , ñường ñi này gồm 4 1 ñỉnh \*, +, … , - và

4 cạnh \*, +

, +, 5

, … , -'+, -

. ðỉnh

ñược gọi là ñỉnh ñầu và ñỉnh

ñược gọi là ñỉnh cuối của ñường ñi. Nếu tồn tại một ñường ñi từ

tới , ta nói

ñến ñược và ñến ñược từ

:

6 .

Hình 5-10: ðồ thị và ñường ñi

Trên cả hai ñồ thị ở Hình 5-10, )1,2,3,4. là ñường ñi từ ñỉnh 1 tới ñỉnh 4.

)1,6,5,4 . không phải ñường ñi vì không có cạnh (cung) 6,5.

Một bài toán quan trọng trong lí thuyết ñồ thị là bài toán duyệt tất cả các ñỉnh có

thể ñến ñược từ một ñỉnh xuất phát nào ñó. Vấn ñề này ñưa về một bài toán liệt kê

mà yêu cầu của nó là không ñược bỏ sót hay lặp lại bất kì ñỉnh nào. Chính vì vậy

mà ta phải xây dựng những thuật toán cho phép duyệt một cách hệ thống các ñỉnh,

những thuật toán như vậy gọi là những thuật toán tìm kiếm trên ñồ thị (graph

traversal). Ta quan tâm ñến hai thuật toán cơ bản nhất: thuật toán tìm kiếm theo

chiều sâu và thuật toán tìm kiếm theo chiều rộng.

Trong những chương trình cài ñặt dưới ñây, ta giả thiết rằng ñồ thị ñược cho là ñồ

thị có hướng, số ñỉnh không quá 10F

, số cung không quá 10

, các ñỉnh ñược ñánh

1

2

6

3

4

5

1

2

6

3

4

5

G1 G2

144

số từ 1 tới và ñồng nhất với số hiệu của chúng. Khuôn dạng Input/Output quy

ñịnh cụ thể như sau:

Input

• Dòng 1 chứa số ñỉnh , ñỉnh xuất phát

và ñỉnh cần ñến .

• dòng tiếp theo, dòng thứ \_x0012\_ chứa một danh sách các ñỉnh, mỗi ñỉnh trong

danh sách tương ứng với một cung \_x0012\_, của ñồ thị, ngoài ra có thêm một số 0

ở cuối dòng ñể báo hiệu kết thúc.

Output

• Danh sách các ñỉnh có thể ñến ñược từ

• ðường ñi từ

tới nếu có

Sample Input Sample Output

8 1 5

2 3 0

3 4 0

1 5 0

6 0

0

2 0

8 0

0

Reachable vertices from 1:

1, 2, 3, 5, 4, 6,

The path from 1 to 5:

5<-3<-2<-1

3.2. Biểu diễn ñồ thị

ðồ thị ñược biểu diễn bằng danh sách kề dạng forward star, mỗi ñỉnh sẽ ñược

cho tương ứng với một danh sách các ñỉnh nối từ . Nếu ñồ thị có ñỉnh thì có

tổng cộng danh sách kề, gọi I là tổng số phần tử trên tất cả các danh sách kề.

Khi ñó I ||, như ñã quy ước, I 3 10

.

Cấu trúc dữ liệu ñược cài ñặt bằng mảng l m1 … In mảng này ñược chia làm

ñoạn liên tiếp, ñoạn thứ chứa danh sách các ñỉnh nối từ . Vị trí của các ñoạn

ñược xác ñịnh bởi mảng

lm0 … n trong ñó

lmn là vị trí cuối ñoạn thứ ,

quy ước

lm0n 0. Như vậy các ñỉnh nối từ sẽ nằm liên tiếp trong mảng l

từ chỉ số

lm O 1n 1 tới chỉ số

lmn.

1

2 3

4 5

6

7

8

145

3.3. Thuật toán tìm kiếm theo chiều sâu

a) Ý tưởng

Tư tưởng của thuật toán tìm kiếm theo chiều sâu (Depth-First Search – DFS) có

thể trình bày như sau: Trước hết, dĩ nhiên ñỉnh

ñến ñược từ

, tiếp theo, với mọi

cung

, J của ñồ thị thì J cũng sẽ ñến ñược từ

. Với mỗi ñỉnh J ñó thì tất nhiên

những ñỉnh K nối từ J cũng ñến ñược từ

… ðiều ñó gợi ý cho ta viết một thủ tục

ñệ quy y\_x0012\_

\_x0012\_ mô tả việc duyệt từ ñỉnh bằng cách thăm ñỉnh và tiếp tục

quá trình duyệt y\_x0012\_

\_x0012\_ với là một ñỉnh chưa thăm nối từ .

Kĩ thuật ñánh dấu ñược sử dụng ñể tránh việc liệt kê lặp các ñỉnh: Khởi tạo

\_x0012\_smn x True, 0 , mỗi lần thăm một ñỉnh, ta ñánh dấu ñỉnh ñó lại

(\_x0012\_smn x False) ñể các bước duyệt ñệ quy kế tiếp không duyệt lại ñỉnh ñó nữa

ðể lưu lại ñường ñi từ ñỉnh xuất phát

, trong thủ tục y\_x0012\_

\_x0012\_, trước khi gọi

ñệ quy y\_x0012\_

\_x0012\_ với là một ñỉnh chưa thăm nối từ (chưa ñánh dấu), ta lưu

lại vết ñường ñi từ tới bằng cách ñặt

mn x , tức là

mn lưu lại

ñỉnh liền trước trong ñường ñi từ

tới . Khi thuật toán DFS kết thúc, ñường ñi

từ

tới sẽ là:

)+  5

m+

n  D

m5

n   

.

procedure DFSVisit(uV); //Thuật toán tìm kiếm theo chiều sâu từ ñỉnh u

begin

avail[u] := False; //avail[u] = False ⇔ u ñã thăm

Output ← u; //Lit kê u

for ∀vV:(u, v)E do //Duyệt mọi ñỉnh v chưa thăm nối từ u

if avail[v] then

begin

trace[v] := u; //Lưu vết ñường ñi, ñỉnh liền trước v trên ñường ñi

từ s tới v là u

DFSVisit(v); //Gọi ñệ quy ñể tìm kiếm theo chiều sâu từ ñỉnh v

end;

end;

begin //Chương trình chính

Input → ðồ thị G, ñỉnh xuất phát s, ñỉnh ñích t;

for ∀vV do avail[v] := True; //Đánh dấu mọi ñỉnh ñều chưa thăm

DFSVisit(s);

if avail[t] then //s đi t\_x0007\_i ñ

c t

«Truy theo vết từ t ñể tìm ñường ñi từ s tới t»;

end.

146

b) Cài ñặt

DFS.PAS Tìm ñường bằng DFS

{$MODE OBJFPC}

{$M 4000000}

program DepthFirstSearch;

const

maxN = 100000;

maxM = 1000000;

var

adj: array[1..maxM] of Integer; //Các danh sách kề

head: array[0..maxN] of Integer; //Mảng ñánh dấu vị trí cắt ñoạn trong

adj

avail: array[1..maxN] of Boolean;

trace: array[1..maxN] of Integer;

n, s, t: Integer;

procedure Enter; //Nhập dữ liệu

var

u, v, i: Integer;

begin

ReadLn(n, s, t);

i := 0;

for u := 1 to n do

begin //Đọc danh sách kề của u

repeat

read(v);

if v <> 0 then //Thêm v vào mảng adj

begin

Inc(i); adj[i] := v;

end;

until v = 0;

head[u] := i; //Đọc hết một dòng, ñánh dấu vị trí cắt ñoạn thứ u

ReadLn;

end;

head[0] := 0; //Cầm canh

end;

procedure DFSVisit(u: Integer); //Thuật toán tìm kiếm theo chiều sâu bắt

ñầu từ u

var

i: Integer;

begin

avail[u] := False;

147

Write(u, ', '); //Liệt kê u

for i := head[u - 1] + 1 to head[u] do //Duyệt các ñỉnh adj[i] nối từ u

if avail[adj[i]] then

begin

trace[adj[i]] := u;

DFSVisit(adj[i]);

end;

end;

procedure PrintPath; //In đường ñi từ s tới t

begin

if avail[t] then //Từ s không có ñường tới t

WriteLn(' There is no path from ', s, ' to ', t)

else

begin

WriteLn('The path from ', s, ' to ', t, ':');

while t <> s do //Truy vết ngược từ t về s

begin

Write(t, '<-');

t := trace[t];

end;

WriteLn(s);

end;

end;

begin

Enter;

FillChar(avail[1], n \* SizeOf(avail[1]), True);

WriteLn('Reachable vertices from ', s, ': ');

DFSVisit(s);

WriteLn;

PrintPath;

end.

Có thể không cần mảng ñánh dấu \_x0012\_sm1 … n mà dùng luôn mảng

m1 … n

ñể ñánh dấu: Khởi tạo các phần tử mảng

m1 … n là:



m

n \_x001D\_ 0

mn 0, 0 \_x001D\_

Khi ñó ñiều kiện ñể một ñỉnh chưa thăm là

mn 0, mỗi khi từ ñỉnh

thăm ñỉnh , phép gán

mn x sẽ kiêm luôn công việc ñánh dấu ñã thăm

mn \_x001D\_ 0.

148

Một vài tính chất của DFS

Cây DFS

Nếu ta sắp xếp danh sách kề của mỗi ñỉnh theo thứ tự tăng dần thì thuật toán DFS

luôn trả về ñường ñi có thứ tự từ ñiển nhỏ nhất trong số tất cả các ñường ñi từ

tới tới .

Quá trình tìm kiếm theo chiều sâu cho ta một cây DFS gốc

. Quan hệ cha–con

trên cây ñược ñịnh nghĩa là: nếu từ ñỉnh tới thăm ñỉnh (y\_x0012\_

\_x0012\_ gọi

y\_x0012\_

\_x0012\_) thì là nút cha của nút . Hình 5-11 là ñồ thị và cây DFS tương

ứng với ñỉnh xuất phát

1.

Hình 5-11: ðồ thị và cây DFS

Mô hình duyệt ñồ thị theo DFS

Cài ñặt trên chỉ là một ứng dụng của thuật toán DFS ñể liệt kê các ñỉnh ñến ñược

từ một ñỉnh. Thuật toán DFS dùng ñể duyệt qua các ñỉnh và các cạnh của ñồ thị

ñược viết theo mô hình sau:

procedure DFSVisit(uV); //Thuật toán tìm kiếm theo chiều sâu từ ñỉnh u

begin

Time := Time + 1;

d[u] := Time;

Output ← u; //Liệt kê u

for ∀vV:(u, v)E do //Duyệt mọi ñỉnh v nối từ u

if d[v] = 0 then DFSVisit(v); //Nếu v chưa thăm, gọi ñệ quy ñể tìm

kiếm theo chiều sâu từ ñỉnh v

Time := Time + 1;

f[u] := Time;

end;

begin //Chương trình chính

Input → ðồ thị G

1

2 3

4 5

6

7

8

1

2 3

4 5

6

7

8

149

for ∀vV do d[v] := 0; //Mọi ñỉnh ñều chưa ñược duyệt ñến

Time := 0;

for ∀vV do

if d[v] = 0 then DFSVisit(v);

end.

Thuật toán này sẽ thăm tất cả các ñỉnh và các cạnh của ñồ thị và thứ tự thăm ñược

gọi là thứ tự duyệt DFS. Như ví dụ ở ñồ thị trong bài, thứ tự thăm DFS với các

ñỉnh là:

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

Thứ tự thăm DFS với các cạnh là:

1,2; 2,3; 3,1; 3,5; 2,4; 4,6; 6,2; 1,3; 7,8

Thời gian thực hiện giải thuật của DFS có thể ñánh giá bằng số lần gọi thủ tục

y\_x0012\_

\_x0012\_ (|| lần) cộng với số lần thực hiện của vòng lặp for bên trong thủ tục

y\_x0012\_

\_x0012\_. Chính vì vậy:

• Nếu ñồ thị ñược biểu diễn bằng danh sách kề hoặc danh sách liên thuộc, vòng

lặp for bên trong thủ tục y\_x0012\_

\_x0012\_ (xét tổng thể cả chương trình) sẽ duyệt

qua tất cả các cạnh của ñồ thị (mỗi cạnh hai lần nếu là ñồ thị vô hướng, mỗi

cạnh một lần nếu là ñồ thị có hướng). Trong trường hợp này, thời gian thực

hiện giải thuật DFS là Θ|| ||

• Nếu ñồ thị ñược biểu diễn bằng ma trận kề, vòng lặp for bên trong mỗi thủ

tục y\_x0012\_

\_x0012\_ sẽ phải duyệt qua tất cả các ñỉnh 1 … . Trong trường hợp này

thời gian thực hiện giải thuật DFS là Θ|| ||

5

Θ||

5

.

• Nếu ñồ thị ñược biểu diễn bằng danh sách cạnh , vòng lặp for bên trong thủ

tục y\_x0012\_

\_x0012\_ sẽ phải duyệt qua tất cả danh sách cạnh mỗi lần thực hiện thủ

tục. Trong trường hợp này thời gian thực hiện giải thuật DFS là Θ||||.

Thứ tự duyệt ñến và duyệt xong

Hãy ñể ý thủ tục y\_x0012\_

\_x0012\_:

• Khi bắt ñầu vào thủ tục ta nói ñỉnh ñược duyệt ñến hay ñược thăm

(discover), có nghĩa là tại thời ñiểm ñó, quá trình tìm kiếm theo chiều sâu bắt

ñầu từ sẽ xây dựng nhánh cây DFS gốc .

• Khi chuẩn bị thoát khỏi thủ tục ñể lùi về , ta nói ñỉnh ñược duyệt xong

(finish), có nghĩa là tại thời ñiểm ñó, quá trình tìm kiếm theo chiều sâu từ

kết thúc.

Trong mô hình duyệt DFS ở trên, chúng ta sử dụng một biến ñếm z\_x0012\_I

ñể xác

ñịnh thời ñiểm duyệt ñến lk và thời ñiểm duyệt xong :k của mỗi ñỉnh . Thứ tự

150

duyệt ñến và duyệt xong này có ý nghĩa rất quan trọng trong nhiều thuật toán có

áp dụng DFS, chẳng hạn như các thuật toán tìm thành phần liên thông mạnh, thuật

toán sắp xếp tô pô…

ðịnh lí 5-6

Với hai ñỉnh phân biệt , :

• ðỉnh ñược duyệt ñến trong thời gian từ lk ñến :k: lmn mlk, :kn nếu

và chỉ nếu là hậu duệ của trên cây DFS.

• ðỉnh ñược duyệt xong trong thời gian từ lk ñến :k:

:mn mlk, :k

n nếu và chỉ nếu là hậu duệ của trên cây DFS.

Ch

ng minh

Bản chất của việc ñỉnh ñược duyệt ñến (hay duyệt xong) trong thời gian từ lk ñến :k

chính là thủ tục y\_x0012\_

\_x0012\_ ñược gọi (hay thoát) khi mà thủ tục y\_x0012\_

\_x0012\_ ñã bắt ñầu

nhưng chưa kết thúc, nghĩa là thủ tục y\_x0012\_

\_x0012\_ ñược dây chuyền ñệ quy từ

y\_x0012\_

\_x0012\_ gọi tới. ðiều này chỉ ra rằng nằm trong nhánh DFS gốc , hay nói cách

khác, là hậu duệ của .

Hệ quả

Với hai ñỉnh phân biệt , thì hai ñoạn mlk, :kn và ml", :"n hoặc rời

nhau hoặc chứa nhau. Hai ñoạn mlk, :kn và ml", :"n chứa nhau nếu và chỉ

nếu và có quan hệ tiền bối–hậu duệ.

Ch

ng minh

Dễ thấy rằng nếu hai ñoạn mlk, :kn và ml"

, :"

n không rời nhau thì hoặc lk ml"

, :"

n hoặc

l" mlk, :kn, tức là hai ñỉnh , có quan hệ tiền bối–hậu duệ, áp dụng ðịnh lí 5-6, ta có

ðPCM.

ðịnh lí 5-7

Với hai ñỉnh phân biệt \_x001D\_ mà , thì phải ñược duyệt ñến

trước khi ñược duyệt xong:

,  l"  :k (0.1)

Ch

ng minh

ðây là một tính chất quan trọng của thuật toán DFS. Hãy ñể ý thủ tục y\_x0012\_

\_x0012\_, trước

khi thoát (duyệt xong ), nó sẽ quét tất cả các ñỉnh chưa thăm nối từ và gọi ñệ quy ñể

thăm những ñỉnh ñó, tức là phải ñược duyệt ñến trước khi ñược duyệt xong: l"  :k.

ðịnh lí 5-8 (ñịnh lí ñường ñi trắng)

ðỉnh là hậu duệ thực sự của ñỉnh trong một cây DFS nếu và chỉ nếu

tại thời ñiểm lk mà thuật toán thăm tới ñỉnh , tồn tại một ñường ñi từ

151

tới mà ngoại trừ ñỉnh , tất cả các ñỉnh khác trên ñường ñi ñều chưa

ñược thăm.

Ch

ng minh

“⇒”

Nếu là hậu duệ của , ta xét ñường ñi từ tới dọc trên các cung trên cây DFS. Tất cả

các ñỉnh } nằm sau trên ñường ñi này ñều là hậu duệ của , nên theo ðịnh lí 5-6, ta có

lk  l, tức là vào thời ñiểm lk, tất cả các ñỉnh } ñó ñều chưa ñược thăm

“⇐”

Nếu tại thời ñiểm lk, tồn tại một ñường ñi từ tới mà ngoại trừ ñỉnh , tất cả các ñỉnh

khác trên ñường ñi ñều chưa ñược thăm, ta sẽ chứng minh rằng mọi ñỉnh trên ñường ñi này

ñều là hậu duệ của . Thật vậy, giả sử phản chứng rằng



là ñỉnh ñầu tiên trên ñường ñi

này mà không phải hậu duệ của , tức là tồn tại ñỉnh } liền trước



trên ñường ñi là hậu

duệ của . Theo ðịnh lí 5-7,



phải ñược thăm trước khi duyệt xong }: l"

  :; } lại là

hậu duệ của nên theo ðịnh lí 5-6, ta có : 3 :k, vậy l"

  :k. Mặt khác theo giả thiết

rằng tại thời ñiểm lk thì



chưa ñược thăm, tức là lk  l"

, kết hợp lại ta có l"

mlk, :kn, vậy thì



là hậu duệ của theo ðịnh lí 5-6, trái với giả thiết phản chứng.

Tên gọi “ñịnh lí ñường ñi trắng: white-path theorem” xuất phát từ cách trình bày

thuật toán DFS bằng cơ chế tô màu ñồ thị: Ban ñầu các ñỉnh ñược tô màu trắng,

mỗi khi duyệt ñến một ñỉnh thì ñỉnh ñó ñược tô màu xám và mỗi khi duyệt xong

một ñỉnh thì ñỉnh ñó ñược tô màu ñen: ðịnh lí khi ñó có thể phát biểu: ðiều kiện

cần và ñủ ñể ñỉnh là hậu duệ thực sự của ñỉnh trong một cây DFS là tại thời

ñiểm ñỉnh ñược tô màu xám, tồn tại một ñường ñi từ tới mà ngoại trừ ñỉnh

, tất cả các ñỉnh khác trên ñường ñi ñều có màu trắng.

3.4. Thuật toán tìm kiếm theo chiều rộng

a) Ý tưởng

Tư tưởng của thuật toán tìm kiếm theo chiều rộng (Breadth-First Search – BFS) là

“lập lịch” duyệt các ñỉnh. Việc thăm một ñỉnh sẽ lên lịch duyệt các ñỉnh nối từ nó

sao cho thứ tự duyệt là ưu tiên chiều rộng (ñỉnh nào gần ñỉnh xuất phát

hơn sẽ

ñược duyệt trước). ðầu tiên ta thăm ñỉnh

. Việc thăm ñỉnh

sẽ phát sinh thứ tự

thăm những ñỉnh +, 5, … nối từ

(những ñỉnh gần

nhất). Tiếp theo ta thăm

ñỉnh +, khi thăm ñỉnh + sẽ lại phát sinh yêu cầu thăm những ñỉnh +, 5, … nối

từ +. Nhưng rõ ràng các ñỉnh này “xa”

hơn những ñỉnh nên chúng chỉ ñược

thăm khi tất cả những ñỉnh ñã thăm. Tức là thứ tự duyệt ñỉnh sẽ là:

, +, 5, … , +, 5, …

152

Hình 5-12: Thứ tự thăm ñỉnh của BFS

Thuật toán tìm kiếm theo chiều rộng sử dụng một danh sách ñể chứa những ñỉnh

ñang “chờ” thăm. Tại mỗi bước, ta thăm một ñỉnh ñầu danh sách, loại nó ra khỏi

danh sách và cho những ñỉnh chưa “xếp hàng” kề với nó xếp hàng thêm vào cuối

danh sách. Thuật toán sẽ kết thúc khi danh sách rỗng.

Vì nguyên tắc vào trước ra trước, danh sách chứa những ñỉnh ñang chờ thăm ñược

tổ chức dưới dạng hàng ñợi (Queue): Nếu ta có 

là một hàng ñợi với thủ tục

(

ñể ñẩy một ñỉnh vào hàng ñợi và hàm (\_x0010\_ trả về một ñỉnh lấy ra từ

hàng ñợi thì mô hình của giải thuật BFS có thể viết như sau:

Queue := (s); //Khởi tạo hàng ñợi chỉ gồm một ñỉnh s

for ∀vV do

avail[v] := True;

avail[s] := False; //ðánh dấu chỉ có ñỉnh s ñược xếp hàng

repeat //Lặp tới khi hàng ñợi rỗng

u := Pop; //Lấy từ hàng ñợi ra một ñỉnh u

Output ← u; //Liệt kê u

for ∀vV:avail[v] and (u, v)E do //Xét những ñỉnh v kề u

chưa ñược ñẩy vào hàng ñợi

begin

trace[v] := u; //Lưu vết ñường ñi

Push(v); //ðẩy v vào hàng ñợi

avail[v] := False; //ðánh dấu v ñã xếp hàng

end;

until Queue = t;

if avail[t] then //s ñi tới ñược t

«Truy theo vết từ t ñể tìm ñường ñi từ s tới t»;

+ 5 ……

+ 5

Thăm trước tất cả các ñỉnh

…… Thăm sau tất cả các ñỉnh

153

b) Cài ñặt

BFS.PAS Tìm ñường bằng BFS

{$MODE OBJFPC}

program Breadth\_First\_Search;

const

maxN = 100000;

maxM = 1000000;

var

adj: array[1..maxM] of Integer; //Các danh sách kề

head: array[0..maxN] of Integer; //Mảng ñánh dấu vị trí cắt ñoạn trong adj

avail: array[1..maxN] of Boolean;

trace: array[1..maxN] of Integer;

n, s, t: Integer;

Queue: array[1..maxN] of Integer;

front, rear: Integer;

procedure Enter; //Nhập dữ liệu

var

u, v, i: Integer;

begin

ReadLn(n, s, t);

i := 0;

for u := 1 to n do

begin //Đọc danh sách kề của u

repeat

read(v);

if v <> 0 then //Thêm v vào mảng adj

begin

Inc(i); adj[i] := v;

end;

until v = 0;

head[u] := i; //Đọc hết một dòng, ñánh dấu vị trí cắt ñoạn thứ u

ReadLn;

end;

head[0] := 0; //Cầm canh

end;

procedure BFS; //Thuật toán tìm kiếm theo chiều rộng

var

u, i: Integer;

begin

front := 1; rear := 1; //front: chỉ số ñầu hàng ñợi; rear: chỉ số cuối hàng ñợi

Queue[1] := s; //Khởi tạo hàng ñợi ban ñầu chỉ có mỗi một ñỉnh s

154

FillChar(avail[1], n \* SizeOf(avail[1]), True); //Các đỉnh ñều

chưa xếp hàng

avail[s] := False; //ngoại trừ ñỉnh s ñã xếp hàng

repeat

u := Queue[front]; Inc(front); //Lấy từ hàng ñợi ra một ñỉnh u

Write(u, ', '); //Liệt kê u

for i := head[u - 1] + 1 to head[u] do //Duyệt những ñỉnh adj[i]

nối từ u

if avail[adj[i]] then //Nếu ñỉnh ñó chưa thăm

begin

Inc(rear); Queue[rear] := adj[i]; //Đẩy vào hàng ñợi

avail[adj[i]] := False;

trace[adj[i]] := u; //Lưu vết ñường ñi

end;

until front > rear;

end;

procedure PrintPath; //In đường ñi từ s tới t

begin

if avail[t] then //Từ s không có ñường tới t

WriteLn(' There is no path from ', s, ' to ', t)

else

begin

WriteLn('The path from ', s, ' to ', t, ':');

while t <> s do //Truy vết ngược từ t về s

begin

Write(t, '<-');

t := trace[t];

end;

WriteLn(s);

end;

end;

begin

Enter;

WriteLn('Reachable vertices from ', s, ': ');

BFS;

WriteLn;

PrintPath;

end.

Tương tự như thuật toán tìm kiếm theo chiều sâu, ta có thể dùng mảng

z

m1 … n kiêm luôn chức năng ñánh dấu.

155

c) Một vài tính chất của BFS

Cây BFS

Nếu ta sắp xếp các danh sách kề của mỗi ñỉnh theo thứ tự tăng dần thì thuật toán

BFS luôn trả về ñường ñi qua ít cạnh nhất trong số tất cả các ñường ñi từ

tới .

Nếu có nhiều ñường ñi từ

tới ñều qua ít cạnh nhất thì thuật toán BFS sẽ trả về

ñường ñi có thứ tự từ ñiển nhỏ nhất trong số những ñường ñi ñó.

Quá trình tìm kiếm theo chiều rộng cho ta một cây BFS gốc

. Khi thuật toán kết

thúc z

mn chính là nút cha của nút trên cây. Hình 5-13 là ñồ thị và cây BFS

tương ứng với ñỉnh xuất phát

1.

Hình 5-13: ðồ thị và cây BFS

Mô hình duyệt ñồ thị theo BFS

Tương tự như thuật toán DFS, trên thực tế, thuật toán BFS cũng dùng ñể xác ñịnh

một thứ tự trên các ñỉnh của ñồ thị và ñược viết theo mô hình sau:

procedure BFSVisit(sV);

begin

Queue := (s); //Khi to hàng ñ  
i ch gm m\_x0019\_t ñnh s

Time := Time + 1;

d[s] := Time; //Duyệt ñến ñỉnh s

repeat //Lặp tới khi hàng ñợi rỗng

u := Pop; //Lấy từ hàng ñợi ra một ñỉnh u

Time := Time + 1;

f[u] := Time; //Ghi nhận thời ñiểm duyệt xong ñỉnh u

Output ← u; //Liệt kê u

for ∀vV:(u, v)E do //Xét những ñỉnh v kề u

if d[v] = 0 then //Nếu v chưa duyệt ñến

begin

Push(v); //ðẩy v vào hàng ñợi

1

2 3

4 5

6

7

8

1

2 3

4 5

6

7

8

156

Time := Time + 1;

d[v] := Time; //Ghi nhận thời ñiểm duyệt ñến ñỉnh v

end;

until Queue = t;

end;

begin //Chương trình chính

Input → ðồ thị G;

for ∀vV do d[v] := 0; //Mọi ñỉnh ñều chưa ñược duyệt ñến

Time := 0;

for ∀vV do

if avail[v] then BFSVisit(v);

end.

Thời gian thực hiện giải thuật của BFS tương tự như ñối với DFS, bằng Θ||

|| nếu ñồ thị ñược biểu diễn bằng danh sách kề hoặc danh sách liên thuộc, bằng

Θ||

5

nếu ñồ thị ñược biểu diễn bằng ma trận kề, và bằng Θ|||| nếu ñồ thị

ñược biểu diễn bằng danh sách cạnh.

Thứ tự duyệt ñến và duyệt xong

Tương tự như thuật toán DFS, ñối với thuật toán BFS người ta cũng quan tâm tới

thứ tự duyệt ñến và duyệt xong: Khi một ñỉnh ñược ñẩy vào hàng ñợi, ta nói ñỉnh

ñó ñược duyệt ñến (ñược thăm) và khi một ñỉnh ñược lấy ra khỏi hàng ñợi, ta nói

ñỉnh ñó ñược duyệt xong. Trong mô hình cài ñặt trên, mỗi ñỉnh sẽ tương ứng

với thời ñiểm duyệt ñến lk và thời ñiểm duyệt xong l"

Vì cách hoạt ñộng của hàng ñợi: ñỉnh nào duyệt ñến trước sẽ phải duyệt xong

trước, chính vì vậy, việc liệt kê các ñỉnh có thể thực hiện khi chúng ñược duyệt

ñến hay duyệt xong mà không ảnh hưởng tới thứ tự. Như cách cài ñặt ở trên, mỗi

ñỉnh ñược ñánh dấu mỗi khi ñỉnh ñó ñược duyệt ñến và ñược liệt kê mỗi khi nó

ñược duyệt xong.

Có thể sửa ñổi một chút mô hình cài ñặt bằng cách thay cơ chế ñánh dấu duyệt

ñến/chưa duyệt ñến bằng duyệt xong/chưa duyệt xong:

Input → ðồ thị G;

for ∀vV do avail[v] := True; //ðánh dấu mọi ñỉnh ñều chưa duyệt xong

Queue := t;

for ∀vV do Push(v); //Khởi tạo hàng ñợi chứa tất cả các ñỉnh

repeat //Lặp tới khi hàng ñợi rỗng

u := Pop; //Lấy từ hàng ñợi ra một ñỉnh u

if avail[u] then //Nếu u chưa duyệt xong

begin

157

Output ← u; //Lit kê u

avail[u] := False; //Đánh du u ñã duyt xong

for ∀vV: avail[v] and ((u, v)E) do //Xét nh‑ng ñnh v k u ch

a

duyt xong

begin

trace[v] := u; //L

u v$t ñ

%ng ñi

Push(v); //Đâ&y v vào hàng ñ  
i

end;

until Queue = t;

Kết quả của hai cách cài ñặt không khác nhau, sự khác biệt chỉ nằm ở lượng bộ

nhớ cần sử dụng cho hàng ñợi 

: Ở cách cài ñặt thứ nhất, do cơ chế ñánh

dấu duyệt ñến/chưa duyệt ñến, mỗi ñỉnh sẽ ñược ñưa vào 

ñúng một lần và

lấy ra khỏi 

ñúng một lần nên chúng ta cần không quá ô nhớ ñể chứa các

phần tử của 

. Ở cách cài ñặt thứ hai, có thể có nhiều hơn ñỉnh ñứng xếp

hàng trong 

vì một ñỉnh có thể ñược ñẩy vào 

tới 1 deg lần

(tính cả bước khởi tạo hàng ñợi chứa tất cả các ñỉnh), có nghĩa là khi tổ chức dữ

liệu, chúng ta phải dự trù ∑ 1 deg "# 2I ô nhớ cho 

. Con

số này ñối với ñồ thị có hướng là I ô nhớ.

Rõ ràng ñối với BFS, cách cài ñặt như ban ñầu sẽ tiết kiệm bộ nhớ hơn. Nhưng có

ñiểm ñặc biệt là nếu thay cấu trúc hàng ñợi bởi cấu trúc ngăn xếp trong cách cài

ñặt thứ hai, ta sẽ ñược thứ tự duyệt ñỉnh DFS. ðây chính là phương pháp khử ñệ

quy của DFS ñể cài ñặt thuật toán trên các ngôn ngữ không cho phép ñệ quy.

Bài tập

5.9. Viết chương trình cài ñặt thuật toán DFS không ñệ quy.

5.10. Xét ñồ thị có hướng , , dùng thuật toán DFS duyệt ñồ thị . Cho

một phản ví dụ ñể chứng minh giả thuyết sau là sai: Nếu từ ñỉnh có ñường

ñi tới ñỉnh và ñược duyệt ñến trước , thì nằm trong nhánh DFS gốc .

5.11. Cho ñồ thị vô hướng , , tìm thuật toán Ο|| ñể phát hiện một

chu trình ñơn trong .

5.12. Cho ñồ thị có hướng , có ñỉnh, và mỗi ñỉnh \_x0012\_ ñược gán một

nhãn là số nguyên /

, tập cung của ñồ thị ñược ñịnh nghĩa là ,

u k 7 ". Giả sử rằng thuật toán DFS ñược sử dụng ñể duyệt ñồ thị,

hãy khảo sát tính chất của dãy các nhãn nếu ta xếp các ñỉnh theo thứ tự từ

ñỉnh duyệt xong ñầu tiên ñến ñỉnh duyệt xong sau cùng.

5.13. Mê cung hình chữ nhật kích thước I W gồm các ô vuông ñơn vị I, 3

1000. Trên mỗi ô ghi một trong ba kí tự:

158

• O: Nếu ô ñó an toàn

• X: Nếu ô ñó có cạm bẫy

• E: Nếu là ô có một nhà thám hiểm ñang ñứng.

Duy nhất chỉ có 1 ô ghi chữ E. Nhà thám hiểm có thể từ một ô ñi sang một

trong số các ô chung cạnh với ô ñang ñứng. Một cách ñi thoát khỏi mê cung

là một hành trình ñi qua các ô an toàn ra một ô biên. Hãy chỉ giúp cho nhà

thám hiểm một hành trình thoát ra khỏi mê cung ñi qua ít ô nhất.

4. Tính liên thông của ñồ thị

4.1. ðịnh nghĩa

a) Tính liên thông trên ñồ thị vô hướng

ðồ thị vô hướng , ñược gọi là liên thông (connected) nếu giữa mọi cặp

ñỉnh của luôn tồn tại ñường ñi. ðồ thị chỉ gồm một ñỉnh duy nhất cũng ñược coi

là ñồ thị liên thông.

Cho ñồ thị vô hướng , và ? là một tập con khác rỗng của tập ñỉnh . Ta

nói ? là một thành phần liên thông (connected component) của nếu:

• ðồ thị hạn chế trên tập ?: > ?, > là ñồ thị liên thông.

• Không tồn tại một tập  chứa ? mà ñồ thị hạn chế trên  là liên thông

(tính tối ñại của ?).

(Ta cũng ñồng nhất khái niệm thành phần liên thông ? với thành phần liên thông

> ?, >).

Hình 5-14: ðồ thị và các thành phần liên thông

Một ñồ thị liên thông chỉ có một thành phần liên thông là chính nó. Một ñồ thị

không liên thông sẽ có nhiều hơn 1 thành phần liên thông. Hình 5-14 là ví dụ về

ñồ thị và các thành phần liên thông +, 5, D của nó.

+

5

D

159

ðôi khi, việc xoá ñi một ñỉnh và tất cả các cạnh liên thuộc với nó sẽ tạo ra một ñồ

thị con mới có nhiều thành phần liên thông hơn ñồ thị ban ñầu, các ñỉnh như thế

gọi là ñỉnh cắt (cut vertices) hay nút khớp (articulation nodes). Hoàn toàn tương

tự, những cạnh mà khi ta bỏ nó ñi sẽ tạo ra một ñồ thị có nhiều thành phần liên

thông hơn so với ñồ thị ban ñầu ñược gọi là cạnh cắt (cut edges) hay cầu

(bridges).

Hình 5-15: Khớp và cầu

b) Tính liên thông trên ñồ thị có hướng

Cho ñồ thị có hướng , , có hai khái niệm về tính liên thông của ñồ thị có

hướng tuỳ theo chúng ta có quan tâm tới hướng của các cung không.

gọi là liên thông mạnh (strongly connected) nếu luôn tồn tại ñường ñi (theo các

cung ñịnh hướng) giữa hai ñỉnh bất kì của ñồ thị, gọi là liên thông yếu (weakly

connected) nếu phiên bản vô hướng của nó là ñồ thị liên thông.

Hình 5-16: Liên thông mạnh và liên thông yếu

4.2. Bài toán xác ñịnh các thành phần liên thông

Một bài toán quan trọng trong lí thuyết ñồ thị là bài toán kiểm tra tính liên thông

của ñồ thị vô hướng hay tổng quát hơn: Bài toán liệt kê các thành phần liên thông

của ñồ thị vô hướng.

ðể liệt kê các thành phần liên thông của ñồ thị vô hướng , , phương

pháp cơ bản nhất là bắt ñầu từ một ñỉnh bất kì, ta liệt kê những ñỉnh ñến ñược từ

ñỉnh ñó vào một thành phần liên thông, sau ñó loại tất cả các ñỉnh ñã liệt kê ra

Liên thông Liên thông

Khớp

Cớ

160

khỏi ñồ thị và lặp lại, thuật toán sẽ kết thúc khi tập ñỉnh của ñồ thị trở thành t .

Việc loại bỏ ñỉnh của ñồ thị có thể thực hiện bằng cơ chế ñánh dấu những ñỉnh bị

loại:

procedure Scan(uV)

begin

«Dùng BFS hoặc DFS liệt kê và ñánh dấu những ñỉnh có thể

ñến ñược từ u»;

end;

begin

for 0uV do

«Khởi tạo v chưa bị ñánh dấu»;

Count := 0;

for 0uV do

if «u chưa bị ñánh dấu» then

begin

Count := Count + 1;

Output ← «Thông báo thành phần liên thông thứ Count

gồm các ñỉnh :»;

Scan(u);

end;

end.

Thời gian thực hiện giải thuật ñúng bằng thời gian thực hiện giải thuật duyệt ñồ

thị bằng DFS hoặc BFS.

4.3. Bao ñóng của ñồ thị vô hướng

a) ðịnh nghĩa

ðồ thị ñầy ñủ với ñỉnh, kí hiệu BC, là một ñơn ñồ thị vô hướng mà giữa hai ñỉnh

bất kì của nó ñều có cạnh nối. ðồ thị ñầy ñủ BC có ñúng 

C

-

 CC'+

5

cạnh, bậc

của mọi ñỉnh ñều là O 1

Hình 5-17: ðồ thị ñầy ñủ

BD BE BF

161

b) Bao ñóng ñồ thị

Với ñồ thị , , người ta xây dựng ñồ thị ,  cũng gồm những ñỉnh

của còn các cạnh xây dựng như sau:

Giữa hai ñỉnh , của có cạnh nốiuGiữa hai ñỉnh , của có ñường ñi

ðồ thị xây dựng như vậy ñược gọi là bao ñóng của ñồ thị .

Từ ñịnh nghĩa của ñồ thị ñầy ñủ, và ñồ thị liên thông, ta suy ra:

• Một ñơn ñồ thị vô hướng là liên thông nếu và chỉ nếu bao ñóng của nó là ñồ

thị ñầy ñủ

• Một ñơn ñồ thị vô hướng có 4 thành phần liên thông nếu và chỉ nếu bao ñóng

của nó có 4 thành phần ñầy ñủ.

Hình 5-18: ðơn ñồ thị vô hướng và bao ñóng của nó

Bởi việc kiểm tra một ñơn ñồ thị có phải ñồ thị ñầy ñủ hay không có thể thực hiện

khá dễ dàng (ñếm số cạnh chẳng hạn) nên người ta nảy ra ý tưởng có thể kiểm tra

tính liên thông của ñồ thị thông qua việc kiểm tra tính ñầy ñủ của bao ñóng. Vấn

ñề ñặt ra là phải có thuật toán xây dựng bao ñóng của một ñồ thị cho trước và một

trong những thuật toán ñó là:

c) Thuật toán Warshall

Thuật toán Warshall – gọi theo tên của Stephen Warshall, người ñã mô tả thuật

toán này vào năm 1960, ñôi khi còn ñược gọi là thuật toán Roy-Warshall vì

Bernard Roy cũng ñã mô tả thuật toán này vào năm 1959. Thuật toán ñó có thể

mô tả rất gọn:

Giả sử ñơn ñồ thị vô hướng , có ñỉnh ñánh số từ 1 tới , thuật toán

Warshall xét tất cả các ñỉnh 4 , với mỗi ñỉnh 4 ñược xét, thuật toán lại xét tiếp

tất cả các cặp ñỉnh \_x0012\_, : nếu ñồ thị có cạnh \_x0012\_, 4 và cạnh 4, thì ta tự nối thêm

cạnh \_x0012\_, nếu nó chưa có. Tư tưởng này dựa trên một quan sát ñơn giản như sau:

162

Nếu từ \_x0012\_ có ñường ñi tới 4 và từ 4 lại có ñường ñi tới thì chắc chắn từ \_x0012\_ sẽ có

ñường ñi tới .

Thuật toán Warshall yêu cầu ñồ thị phải ñược biểu diễn bằng ma trận kề S

T/UV, trong ñó /U True u \_x0012\_, . Mô hình cài ñặt thuật toán khá ñơn giản:

for k := 1 to n do

for i := 1 to n do

for j := 1 to n do

a[i, j] := a[i, j] or a[i, k] and a[k, j];

Việc chứng minh tính ñúng ñắn của thuật toán ñòi hỏi phải lật lại các lí thuyết về

bao ñóng bắc cầu và quan hệ liên thông, ta sẽ không trình bày ở ñây. Tuy thuật

toán Warshall rất dễ cài ñặt nhưng ñòi hỏi thời gian thực hiện giải thuật khá lớn:

ΘD

. Chính vì vậy thuật toán Warshall chỉ nên sử dụng khi thực sự cần tới bao

ñóng của ñồ thị, còn nếu chỉ cần liệt kê các thành phần liên thông thì các thuật

toán tìm kiếm trên ñồ thị tỏ ra hiệu quả hơn nhiều.

Dưới ñây, ta sẽ thử cài ñặt thuật toán Warshall tìm bao ñóng của ñơn ñồ thị vô

hướng sau ñó ñếm số thành phần liên thông của ñồ thị:

Việc cài ñặt thuật toán sẽ qua những bước sau:

• Dùng ma trận kề S biểu diễn ñồ thị, quy ước rằng /U z

, 0\_x0012\_

• Dùng thuật toán Warshall tìm bao ñóng, khi ñó S là ma trận kề của ñồ thị bao

ñóng

• Dựa vào ma trận kề S, ñỉnh 1 và những ñỉnh kề với nó sẽ thuộc thành phần

liên thông thứ nhất; với ñỉnh nào ñó không kề với ñỉnh 1, thì cùng với

những ñỉnh kề nó sẽ thuộc thành phần liên thông thứ hai; với ñỉnh nào ñó

không kề với cả ñỉnh 1 và ñỉnh , thì cùng với những ñỉnh kề nó sẽ thuộc

thành phần liên thông thứ ba v.v…

Input

• Dòng 1: Chứa số ñỉnh 3 200 và số cạnh I của ñồ thị

• I dòng tiếp theo, mỗi dòng chứa một cặp số và tương ứng với một cạnh

,

Output

Liệt kê các thành phần liên thông của ñồ thị

163

Sample

Input

Sample Output

12 10

1 4

2 3

3 6

4 5

6 7

8 9

8 10

9 11

11 8

11 12

Connected Component 1: 1, 4, 5,

Connected Component 2: 2, 3, 6, 7,

Connected Component 3: 8, 9, 10,

11, 12,

WARSHALL.PAS Thuật toán Warshall liệt kê các thành phần liên thông

{$MODE OBJFPC}

program WarshallAlgorithm;

const

maxN = 200;

var

a: array[1..maxN, 1..maxN] of Boolean; //Ma trận kề của ñồ thị

n: Integer;

procedure Enter; //Nhập ñồ thị

var

i, j, k, m: Integer;

begin

ReadLn(n, m);

for i := 1 to n do

begin

FillChar(a[i][1], n \* SizeOf(a[i][1]), False);

a[i, i] := True;

end;

for k := 1 to m do

begin

ReadLn(i, j);

a[i, j] := True;

a[j, i] := True; //Đồ thị vô hướng: (i, j) = (j, i)

end;

end;

procedure ComputeTransitiveClosure; //Thuật toán Warshall

var

1

4 5

2

6 7

3

9

11 12

10

8

164

k, i, j: Integer;

begin

for k := 1 to n do

for i := 1 to n do

for j := 1 to n do

a[i, j] := a[i, j] or a[i, k] and a[k, j];

end;

procedure PrintResult;

var

Count: Integer;

avail: array[1..maxN] of Boolean; //avail[v] = True ↔ v chưa ñược liệt kê

vào thành phần liên thông nào

u, v: Integer;

begin

FillChar(avail, n \* SizeOf(Boolean), True); //Mọi ñỉnh ñều chưa

ñược liệt kê vào thành phần liên thông nào

Count := 0;

for u := 1 to n do

if avail[u] then //Với một ñỉnh u chưa ñược liệt kê vào thành phần liên thông

nào

begin //Liệt kê thành phần liên thông chứa u

Inc(Count);

Write('Connected Component ', Count, ': ');

for v := 1 to n do

if a[u, v] then //Xét những ñỉnh v kề u (trên bao ñóng)

begin

Write(v, ', '); //Liệt kê ñỉnh ñó vào thành phần liên thông

chứa u

avail[v] := False; //Liệt kê ñỉnh nào ñánh dấu ñỉnh ñó

end;

WriteLn;

end;

end;

begin

Enter;

ComputeTransitiveClosure;

PrintResult;

end.

4.4. Bài toán xác ñịnh các thành phần liên thông mạnh

ðối với ñồ thị có hướng, người ta quan tâm ñến bài toán kiểm tra tính liên thông

mạnh, hay tổng quát hơn: Bài toán liệt kê các thành phần liên thông mạnh của ñồ

165

thị có hướng. Các thuật toán tìm kiếm thành phần liên thông mạnh hiệu quả hiện

nay ñều dựa trên thuật toán tìm kiếm theo chiều sâu Depth-First Search.

Ta sẽ khảo sát và cài ñặt hai thuật toán liệt kê thành phần liên thông mạnh với

khuôn dạng Input/Output như sau:

Input

• Dòng ñầu: Chứa số ñỉnh 3 10F

và số cung I 3 10

của ñồ thị.

• I dòng tiếp theo, mỗi dòng chứa hai số nguyên , tương ứng với một cung

, của ñồ thị.

Output

Các thành phần liên thông mạnh.

Sample

Input

Sample Output

11 15

1 2

1 8

2 3

3 4

4 2

4 5

5 6

6 7

7 5

8 9

9 4

9 10

10 8

10 11

11 8

Strongly Connected

Component 1:

7, 6, 5,

Strongly Connected

Component 2:

4, 3, 2,

Strongly Connected

Component 3:

11, 10, 9, 8,

Strongly Connected

Component 4:

1,

a) Phân tích

Xét thuật toán tìm kiếm theo chiều sâu:

procedure DFSVisit(uV);

begin

«Thêm u vào cây T»

for ∀vV:(u, v)E do

if v∉T then

begin

1

2

3 4

5 6

7

8 11

9 10

166

«Thêm v và cung (u, v) vào cây T»;

DFSVisit(v);

end;

end;

begin

Input → ðồ thị G;

for ∀vV do avail[v] := True;

for ∀vV do

if avail[v] then

begin

«Tạo ra một cây rỗng, gọi là T»

DFSVisit(v);

end;

end.

ðể ý thủ tục thăm ñỉnh ñệ quy y\_x0012\_

\_x0012\_. Thủ tục này xét tất cả những ñỉnh

nối từ :

Hình 5-19: Ba dạng cung ngoài cây DFS

• Nếu chưa ñược thăm thì ñi theo cung ñó thăm , tức là cho ñỉnh trở thành

con của ñỉnh trong cây tìm kiếm DFS, cung , khi ñó ñược gọi là cung

DFS (Tree edge).

• Nếu ñã thăm thì có ba khả năng xảy ra ñối với vị trí của và trong cây

tìm kiếm DFS:

− là tiền bối (ancestor) của , tức là ñược thăm trước và thủ tục

y\_x0012\_

\_x0012\_ do dây chuyền ñệ quy từ thủ tục y\_x0012\_

\_x0012\_ gọi tới.

Cung , khi ñó ñược gọi là cung ngược (back edge)

1

2

3

5

4

6

7

2 là tiền bối của 4

→4,2 là cung ngược

7 là hậu duệ của 5

→5,7 là cung xuôi

Với 6, 4 thuộc nhánh cây DFS ñã duyệt trước ñó

→6,4 là cung chéo

167

− là hậu duệ (descendant) của , tức là ñược thăm trước , nhưng thủ

tục y\_x0012\_

\_x0012\_ sau khi tiến ñệ quy theo một hướng khác ñã gọi

y\_x0012\_

\_x0012\_ rồi. Nên khi dây chuyền ñệ quy lùi lại về thủ tục

y\_x0012\_

\_x0012\_ sẽ thấy là ñã thăm nên không thăm lại nữa. Cung ,

khi ñó gọi là cung xuôi (forward edge).

− thuộc một nhánh DFS ñã duyệt trước ñó, cung , khi ñó gọi là cung

chéo (cross edge)

Ta nhận thấy một ñặc ñiểm của thuật toán tìm kiếm theo chiều sâu, thuật toán

không chỉ duyệt qua các ñỉnh, nó còn duyệt qua tất cả những cung nữa. Ngoài

những cung nằm trên cây DFS, những cung còn lại có thể chia làm ba loại: cung

ngược, cung xuôi, cung chéo (Hình 5-19).

b) Cây DFS và các thành phần liên thông mạnh

ðịnh lí 5-9

Nếu J và K là hai ñỉnh thuộc thành phần liên thông mạnh

thì với mọi

ñường ñi từ J tới K cũng như từ K tới J. Tất cả ñỉnh trung gian trên

ñường ñi ñều phải thuộc

.

Ch

ng minh

Vì J và K là hai ñỉnh thuộc

nên có một ñường ñi từ J tới K và một ñường ñi khác từ K tới

J. Nối tiếp hai ñường ñi này lại ta sẽ ñược một chu trình ñi từ J tới K rồi quay lại J trong

ñó là một ñỉnh nằm trên chu trình. ðiều này chỉ ra rằng nếu ñi dọc theo chu trình ta có

thể ñi từ J tới cũng như từ tới J, nghĩa là và J thuộc cùng một thành phần liên thông

mạnh.

ðịnh lí 5-10

Với một thành phần liên thông mạnh

bất kì, sẽ tồn tại duy nhất một ñỉnh

sao cho mọi ñỉnh của

ñều thuộc nhánh cây DFS gốc .

Ch

ng minh

Trong số các ñỉnh của

, chọn là ñỉnh ñược thăm ñầu tiên theo thuật toán DFS. Ta sẽ

chứng minh

nằm trong nhánh DFS gốc . Thật vậy: với một ñỉnh bất kì của

, do

liên thông mạnh nên phải tồn tại một ñường ñi từ tới :

) J\*

, J+

, … , J- .

Từ ðịnh lí 5-9, tất cả các ñỉnh J+

, J5

, … , J- ñều thuộc

, lại do cách chọn nên chúng sẽ

phải thăm sau ñỉnh . Lại từ ðịnh lí 5-8 (ñịnh lí ñường ñi trắng), tất cả các ñỉnh

J+

, J5

, … , J- phải là hậu duệ của tức là chúng ñều thuộc nhánh DFS gốc .

ðỉnh trong chứng minh ñịnh lí – ñỉnh thăm trước tất cả các ñỉnh khác trong

–

gọi là chốt của thành phần liên thông mạnh

. Mỗi thành phần liên thông mạnh có

duy nhất một chốt. Xét về vị trí trong cây DFS, chốt của một thành phần liên

168

thông mạnh là ñỉnh nằm cao nhất so với các ñỉnh khác thuộc thành phần ñó, hay

nói cách khác: là tiền bối của tất cả các ñỉnh thuộc thành phần ñó.

ðịnh lí 5-11

Với một chốt không là tiền bối của bất kì chốt nào khác thì các ñỉnh

thuộc nhánh DFS gốc chính là thành phần liên thông mạnh chứa .

Ch

ng minh

Với mọi ñỉnh nằm trong nhánh DFS gốc , gọi

là chốt của thành phần liên thông mạnh

chứa . Ta sẽ chứng minh

. Thật vậy, theo ðịnh lí 5-10, phải nằm trong nhánh DFS

gốc

. Vậy nằm trong cả nhánh DFS gốc và nhánh DFS gốc

, nghĩa là và

có quan

hệ tiền bối–hậu duệ. Theo giả thiết không là tiền bối của bất kì chốt nào khác nên phải

là hậu duệ của

. Ta có ñường ñi

6 6 , mà

và thuộc cùng một thành phần liên

thông mạnh nên theo ðịnh lí 5-9, cũng phải thuộc thành phần liên thông mạnh ñó. Mỗi

thành phần liên thông mạnh có duy nhất một chốt mà và

ñều là chốt nên

.

Theo ðịnh lí 5-10, ta ñã có thành phần liên thông mạnh chứa nằm trong nhánh DFS gốc

, theo chứng minh trên ta lại có: Mọi ñỉnh trong nhánh DFS gốc nằm trong thành phần

liên thông mạnh chứa . Kết hợp lại ñược: Nhánh DFS gốc chính là thành phần liên

thông mạnh chứa .

c) Thuật toán Tarjan

Ý tưởng

Hình 5-20: Thuật toán Tarjan “bẻ” cây DFS

Thuật toán Tarjan [40] có thể phát biểu như sau: Chọn là chốt không là tiền bối

của một chốt nào khác, chọn lấy thành phần liên thông mạnh thứ nhất là nhánh

DFS gốc . Sau ñó loại bỏ nhánh DFS gốc ra khỏi cây DFS, lại tìm thấy một

chốt

khác mà nhánh DFS gốc

không chứa chốt nào khác, lại chọn lấy thành

1

2

3 4

5 6

7

8 11

9 10

1

2

3 4

5 6

7

8 11

9 10

169

phần liên thông mạnh thứ hai là nhánh DFS gốc

… Tương tự như vậy cho thành

phần liên thông mạnh thứ ba, thứ tư, v.v… Có thể hình dung thuật toán Tarjan

“bẻ” cây DFS tại vị trí các chốt ñể ñược các nhánh rời rạc, mỗi nhánh là một

thành phần liên thông mạnh.

Mô hình cài ñặt của thuật toán Tarjan:

procedure DFSVisit(uV);

begin

«ðánh dấu u ñã thăm»

for ∀vV: (u, v)E do

if «v chưa thăm» then DFSVisit(v);

if «u là chốt» then

begin

«Liệt kê thành phần liên thông mạnh tương ứng với chốt u»

«Loại bỏ các ñỉnh ñã liệt kê khỏi ñồ thị và cây DFS»

end;

end;

begin

«ðánh dấu mọi ñỉnh ñều chưa thăm»

for ∀vV do

if «v chưa thăm» then DFSVisit(v);

end.

Trình bày dài dòng như vậy, nhưng bây giờ chúng ta mới thảo luận tới vấn ñề

quan trọng nhất: Làm thế nào kiểm tra một ñỉnh nào ñó có phải là chốt hay

không ?

ðịnh lí 5-12

Trong mô hình cài ñặt của thuật toán Tarjan, việc kiểm tra ñỉnh có phải

là chốt không ñược thực hiện khi ñỉnh ñược duyệt xong, khi ñó là chốt

nếu và chỉ nếu trong nhánh DFS gốc không có cung tới ñỉnh thăm

trước .

Ch

ng minh

Ta nhắc lại các tính chất của 4 loại cung:

Cung DFS và cung xuôi nối từ ñỉnh thăm trước ñến ñỉnh thăm sau, hơn nữa chúng ñều

là cung nối từ tiền bối tới hậu duệ

Cung ngược và cung chéo nối từ ñỉnh thăm sau tới ñỉnh thăm trước, cung ngược nối từ

hậu duệ tới tiền bối còn cung chéo nối hai ñỉnh không có quan hệ tiền bối–hậu duệ.

Nếu trong nhánh DFS gốc không có cung tới ñỉnh thăm trước thì tức là không tồn tại

cung ngược và cung chéo ñi ra khỏi nhánh DFS gốc . ðiều ñó chỉ ra rằng từ , ñi theo các

cung của ñồ thị sẽ chỉ ñến ñược những ñỉnh nằm trong nội bộ nhánh DFS gốc mà thôi.

Thành phần liên thông mạnh chứa phải nằm trong tập các ñỉnh có thể ñến từ , tập này lại

chính là nhánh DFS gốc , vậy nên là chốt.

170

Ngược lại, nếu từ ñỉnh của nhánh DFS gốc có cung , tới ñỉnh thăm trước thì cung

ñó phải là cung ngược hoặc cung chéo.

Nếu cung , là cung ngược thì là tiền bối của , mà cũng là tiền bối của nhưng

thăm sau nên là hậu duệ của . Ta có một chu trình 6 6 ; nên cả và thuộc

cùng một thành phần liên thông mạnh. Xét về vị trí trên cây DFS, là tiền bối của nên

không thể là chốt

Nếu cung , là cung chéo, ta gọi

là chốt của thành phần liên thông mạnh chứa . Tại

thời ñiểm thủ tục y\_x0012\_

\_x0012\_ xét tới cung , , ñỉnh ñã ñược duyệt ñến nhưng chưa

duyệt xong (do là tiền bối của ), ñỉnh

cũng ñã duyệt ñến (

ñược thăm trước do

là

chốt của thành phần liên thông mạnh chứa , ñược thăm trước theo giả thiết, ñược thăm

trước vì là chốt của thành phần liên thông mạnh chứa ) nhưng chưa duyệt xong (vì nếu

ñược duyệt xong thì thuật toán ñã loại bỏ tất cả các ñỉnh thuộc thành phần liên thông mạnh

chốt

trong ñó có ñỉnh ra khỏi ñồ thị nên cung , sẽ không ñược tính ñến nữa), ñiều

này chỉ ra rằng khi y\_x0012\_

\_x0012\_ ñược gọi, hai thủ tục y\_x0012\_

\_x0012\_ và y\_x0012\_

\_x0012\_

ñều ñã

ñược gọi nhưng chưa thoát, tức là chúng nằm trên một dây chuyền ñệ quy, hay và

có

quan hệ tiền bối–hậu duệ. Vì

ñược thăm trước nên

sẽ là tiền bối của , ta có chu trình

6 6 ; 6

nên và

thuộc cùng một thành phần liên thông mạnh, thành phần này

ñã có chốt

rồi nên không thể là chốt nữa.

Từ ðịnh lí 5-12, việc sẽ kiểm tra ñỉnh có là chốt hay không có thể thay bằng

việc kiểm tra xem có tồn tại cung nối từ một ñỉnh thuộc nhánh DFS gốc tới một

ñỉnh thăm trước hay không?.

Dưới ñây là một cách cài ñặt hết sức thông minh, nội dung của nó là ñánh số thứ

tự các ñỉnh theo thứ tự duyệt ñến. ðịnh nghĩa I

mn là số thứ tự của ñỉnh

theo cách ñánh số ñó. Ta tính thêm q\_x0010\_}mn là giá trị I

m. n nhỏ nhất trong

các ñỉnh có thể ñến ñược từ một ñỉnh nào ñó của nhánh DFS gốc bằng một

cung. Cụ thể cách tính q\_x0010\_}mn như sau:

Trong thủ tục y\_x0012\_

\_x0012\_, trước hết ta ñánh số thứ tự thăm cho ñỉnh :

I

mn và khởi tạo q\_x0010\_}mn x ∞. Sau ñó xét các ñỉnh nối từ u, có hai

khả năng:

Nếu ñã thăm thì ta cực tiểu hoá q\_x0010\_}mn theo công thức:

q\_x0010\_}mnmới x minq\_x0010\_}mncũ, I

mn

Nếu chưa thăm thì ta gọi ñệ quy y\_x0012\_

\_x0012\_, sau ñó cực tiểu hoá q\_x0010\_}mn theo

công thức:

q\_x0010\_}mnmới x minq\_x0010\_}mncũ, q\_x0010\_}mn

Khi duyệt xong một ñỉnh (chuẩn bị thoát khỏi thủ tục y\_x0012\_

\_x0012\_), ta so sánh

q\_x0010\_}mn và I

mn, nếu như q\_x0010\_}mn 7 I

mn thì là chốt, bởi không

có cung nối từ một ñỉnh thuộc nhánh DFS gốc tới một ñỉnh thăm trước . Khi

ñó chỉ việc liệt kê các ñỉnh thuộc thành phần liên thông mạnh chứa chính là

nhánh DFS gốc .

171

ðể công việc dễ dàng hơn nữa, ta ñịnh nghĩa một danh sách   
4 ñược tổ chức

dưới dạng ngăn xếp và dùng ngăn xếp này ñể lấy ra các ñỉnh thuộc một nhánh nào

ñó. Khi duyệt ñến một ñỉnh , ta ñẩy ngay ñỉnh ñó vào ngăn xếp, thì khi duyệt

xong ñỉnh , mọi ñỉnh thuộc nhánh DFS gốc sẽ ñược ñẩy vào ngăn xếp   
4

ngay sau . Nếu là chốt, ta chỉ việc lấy các ñỉnh ra khỏi ngăn xếp   
4 cho tới

khi lấy tới ñỉnh là sẽ ñược nhánh DFS gốc cũng chính là thành phần liên

thông mạnh chứa .

Mô hình

Dưới ñây là mô hình cài ñặt ñầy ñủ của thuật toán Tarjan

procedure DFSVisit(uV);

begin

Count := Count + 1;

Number[u] := Count; //ðánh s) u theo th

t+ duyt ñ$n

Low[u] := +∞;

Push(u); //ð,y u vào ngăn x$p

for ∀vV:(u, v)E do

if Number[v] > 0 then //v ñã thăm

Low[u] := min(Low[u], Number[v])

else // v ch

a thăm

begin

DFSVisit(v); //ði thăm v

Low[u] := min(Low[u], Low[v]);

end;

//ð$n ñây u ñ

c duyt xong

if Low[u] ≥ Number[u] then //N$u u là ch)t

begin

«Thông báo thành phần liên thông mạnh với chốt u gồm có

các ñỉnh:»;

repeat

v := Pop; //Ly t2 ngăn x$p ra m\_x0019\_t ñnh v

Output ← v;

«Xoá ñỉnh v khỏi ñồ thị: V := V - {v}»;

until v = u;

end;

end;

begin

Count := 0;

Stack := t; //Khi to m\_x0019\_t ngăn x$p r5ng

for ∀vV do Number[v] := 0; //Number[v] = 0 ↔ v ch

a thăm

172

for ∀vV do

if Number[v] = 0 then DFSVisit(v);

end.

Bởi thuật toán Tarjan chỉ là sửa ñổi của thuật toán DFS, các phép vào/ra ngăn xếp

ñược thực hiện không quá lần. Vậy nên thời gian thực hiện giải thuật vẫn là

Θ|| || trong trường hợp ñồ thị ñược biểu diễn bằng danh sách kề hoặc danh

sách liên thuộc, là Θ||

5

nếu dùng ma trận kề và là Θ|||| nếu dùng danh

sách cạnh.

Cài ñặt

Chương trình cài ñặt dưới ñây biểu diễn ñồ thị bởi danh sách liên thuộc kiểu

forward star: Mỗi ñỉnh sẽ ñược cho tương ứng với một danh sách các cung ñi ra

khỏi , như vậy mỗi cung sẽ xuất hiện trong ñúng một danh sách liên thuộc. Nếu

các cung ñược lưu trữ trong mảng

m1 … In, danh sách liên thuộc ñược xây dựng

bằng hai mảng.

•

lmn là chỉ số cung ñầu tiên trong danh sách liên thuộc của ñỉnh . Nếu

danh sách liên thuộc ñỉnh là t,

lmn ñược gán bằng 0.

• s\_x0012\_4m\_x0012\_n là chỉ số cung kế tiếp cung

m\_x0012\_n trong danh sách liên thuộc chứa cung

m\_x0012\_n. Trường hợp

m\_x0012\_n là cung cuối cùng của một danh sách liên thuộc, s\_x0012\_4m\_x0012\_n

ñược gán bằng 0

TARJAN.PAS Thuật toán Tarjan

{$MODE OBJFPC}

{$M 4000000}

program StronglyConnectedComponents;

const

maxN = 100000;

maxM = 1000000;

type

TStack = record

Items: array[1..maxN] of Integer;

Top: Integer;

end;

TEdge = record //Cấu trúc cung

x, y: Integer; //Hai đỉnh ñầu mút

end;

var

e: array[1..maxM] of TEdge; //Danh sách cạnh

link: array[1..maxM] of Integer; //link[i]: chỉ số cung tiếp theo e[i] trong

danh sách liên thuộc

173

head: array[1..maxN] of Integer; //head[u]: chỉ số cung ñầu tiên trong

danh sách liên thuộc các cung ñi ra khỏi u

avail: array[1..maxN] of Boolean;

Number, Low: array[1..maxN] of Integer;

Stack: TStack;

n, Count, SCC: Integer;

procedure Enter; //Nhập dữ liệu

var

i, u, v, m: Integer;

begin

ReadLn(n, m);

for i := 1 to m do //Đọc danh sách cạnh

with e[i] do ReadLn(x, y);

FillChar(head[1], n \* SizeOf(head[1]), 0); //Khởi tạo các danh sách

liên thuộc rỗng

for i := m downto 1 do //Xây dựng các danh sách liên thuộc

with e[i] do

begin

link[i] := head[x]; //Móc nối e[i] = (x, y) vào danh sách liên thuộc

những cung đi ra khỏi x

head[x] := i;

end;

end;

procedure Init; //Khởi tạo

begin

FillChar(Number, n \* SizeOf(Number[1]), 0); //Mọi ñỉnh ñều chưa

thăm

FillChar(avail, n \* SizeOf(avail[1]), True); //Chưa ñỉnh nào bị

loại

Stack.Top := 0; //Ngăn xếp rỗng

Count := 0; //Biến ñếm số thứ tự duyệt ñến, dùng ñể ñánh số

SCC := 0; //Biến ñánh số các thành phần liên thông

end;

procedure Push(v: Integer); //Đẩy một ñỉnh v vào ngăn xếp

begin

with Stack do

begin

Inc(Top); Items[Top] := v;

end;

end;

function Pop: Integer; //Lấy một ñỉnh v khỏi ngăn xếp, trả về trong kết quả

hàm

begin

174

with Stack do

begin

Result := Items[Top]; Dec(Top);

end;

end;

//Hàm cực tiểu hoá: Target := Min(Target, Value)

procedure Minimize(var Target: Integer; Value: Integer);

begin

if Value < Target then Target := Value;

end;

procedure DFSVisit(u: Integer); //Thuật toán tìm kiếm theo chiều sâu bắt

ñầu từ u

var

i, v: Integer;

begin

Inc(Count); Number[u] := Count; //Trước hết ñánh số cho u

Low[u] := maxN + 1; //khởi tạo Low[u]:=+∞ rồi sau cực tiểu hoá dần

Push(u); //Đẩy u vào ngăn xếp

i := head[u]; //Duyệt từ ñầu danh sách liên thuộc các cung ñi ra khỏi u

while i <> 0 do

begin

v := e[i].y; //Xét những ñỉnh v nối từ u

if avail[v] then //Nếu v chưa bị loại

if Number[v] <> 0 then //Nếu v ñã thăm

Minimize(Low[u], Number[v]) //cực tiểu hoá Low[u] theo công thức này

else //Nếu v chưa thăm

begin

DFSVisit(v); //Tiếp tục tìm kiếm theo chiều sâu bắt ñầu từ v

Minimize(Low[u], Low[v]); //Rồi cực tiểu hoá Low[u] theo công thức này

end;

i := link[i]; //Chuyển sang xét cung tiếp theo trong danh sách liên thuộc

end;

//Đến ñây thì ñỉnh u ñược duyệt xong, tức là các ñỉnh thuộc nhánh DFS gốc u ñều ñã

thăm

if Low[u] >= Number[u] then //Nếu u là chốt

begin //Liệt kê thành phần liên thông mạnh có chốt u

Inc(SCC);

WriteLn('Strongly Connected Component ', SCC, ': ');

repeat

v := Pop; //Lấy dần các ñỉnh ra khỏi ngăn xếp

Write(v, ', '); //Liệt kê các ñỉnh ñó

avail[v] := False; //Rồi loại luôn khỏi ñồ thị

175

until v = u; //Cho tới khi lấy tới ñỉnh u

WriteLn;

end;

end;

procedure Tarjan; //Thuật toán Tarjan

var

v: Integer;

begin

for v := 1 to n do

if avail[v] then DFSVisit(v);

end;

begin

Enter;

Init;

Tarjan;

end.

d) Thuật toán Kosaraju-Sharir

Mô hình

Có một thuật toán khác ñể liệt kê các thành phần liên thông mạnh là thuật toán

Kosaraju-Sharir (1981). Thuật toán này thực hiện qua hai bước:

• Bước 1: Dùng thuật toán tìm kiếm theo chiều sâu với thủ tục y\_x0012\_

\_x0012\_,

nhưng thêm vào một thao tác nhỏ: ñánh số lại các ñỉnh theo thứ tự duyệt

xong.

• Bước 2: ðảo chiều các cung của ñồ thị, xét lần lượt các ñỉnh theo thứ tự từ

ñỉnh duyệt xong sau cùng tới ñỉnh duyệt xong ñầu tiên, với mỗi ñỉnh ñó, ta lại

dùng thuật toán tìm kiếm trên ñồ thị (BFS hay DFS) liệt kê những ñỉnh nào

ñến ñược từ ñỉnh ñang xét, ñó chính là một thành phần liên thông mạnh. Liệt

kê xong thành phần nào, ta loại ngay các ñỉnh của thành phần ñó khỏi ñồ thị.

ðịnh lí 5-13

Với là ñỉnh duyệt xong sau cùng thì là chốt của một thành phần liên

thông mạnh không có cung ñi vào.

Ch

ng minh

Dễ thấy rằng ñỉnh duyệt xong sau cùng phải là gốc của một cây DFS nên sẽ là chốt của

một thành phần liên thông mạnh, kí hiệu

.

Gọi

là chốt của một thành phần liên thông mạnh

khác. Ta chứng minh rằng không

thể tồn tại cung ñi từ

sang

, giả sử phản chứng rằng có cung , trong ñó

và

. Khi ñó tồn tại một ñường ñi (+

:

6 trong nội bộ

và tồn tại

176

một ñường ñi (5

: 6 nội bộ

. Do tính chất của chốt,

ñược thăm trước mọi ñỉnh

khác trên ñường (+

và ñược thăm trước mọi ñỉnh khác trên ñường (5

. Nối ñường ñi

(+

:

6 với cung , và nối tiếp với ñường ñi (5

: 6 ta ñược một ñường ñi

(:

6 (Hình 5-21)

Hình 5-21

Có hai khả năng xảy ra:

Nếu

ñược thăm trước thì vào thời ñiểm

ñược thăm, mọi ñỉnh khác trên ñường ñi (

chưa thăm. Theo ðịnh lí 5-8 (ñịnh lí ñường ñi trắng),

sẽ là tiền bối của và phải ñược

duyệt xong sau . Trái với giả thiết là ñỉnh duyệt xong sau cùng.

Nếu

ñược thăm sau , nghĩa là vào thời ñiểm ñược duyệt ñến thì

chưa duyệt ñến, lại

do ñược duyệt xong sau cùng nên vào thời ñiểm duyệt xong thì

ñã duyệt xong. Theo

ðịnh lí 5-13,

sẽ là hậu duệ của . Vậy từ

có ñường ñi tới và ngược lại, nghĩa là và

thuộc cùng một thành phần liên thông mạnh. Mâu thuẫn.

ðịnh lí ñược chứng minh.

ðịnh lí 5-13 chỉ ra tính ñúng ñắn của thuật toán Kosaraju-Sharir: ðỉnh duyệt

xong sau cùng chắc chắn là chốt của một thành phần liên thông mạnh và thành

phần liên thông mạnh này gồm mọi ñỉnh ñến ñược . Việc liệt kê các ñỉnh thuộc

thành phần liên thông mạnh chốt ñược thực hiện trong thuật toán thông qua thao

tác ñảo chiều các cung của ñồ thị rồi liệt kê các ñỉnh ñến ñược từ .

Loại bỏ thành phần liên thông mạnh với chốt khỏi ñồ thị. Cây DFS gốc lại

phân rã thành nhiều cây con. Lập luận tương tự như trên với ñỉnh duyệt xong sau

cùng (Hình 5-22)

Ví dụ:

u v

s r (+ (5

(

177

Hình 5-22. ðánh số lại, ñảo chiều các cung và thực hiện thuật toán tìm kiếm trên ñồ thị với

cách chọn các ñỉnh xuất phát ngược lại với thứ tự duyệt xong (thứ tự 11, 10… 3, 2, 1)

Cài ñặt

Trong việc lập trình thuật toán Kosaraju–Sharir, việc ñánh số lại các ñỉnh ñược

thực hiện bằng danh sách: Tại bước duyệt ñồ thị lần 1, mỗi khi duyệt xong một

ñỉnh thì ñỉnh ñó ñược ñưa vào cuối danh sách. Sau khi ñảo chiều các cung của ñồ

thị, chúng ta chỉ cần duyệt từ cuối danh sách sẽ ñược các ñỉnh ñúng thứ tự ngược

với thứ tự duyệt xong (cơ chế tương tự như ngăn xếp)

ðể liệt kê các thành phần liên thông mạnh của ñơn ñồ thị có hướng bằng thuật

toán Tarjan cũng như thuật toán Kosaraju-Sharir, cách biểu diễn ñồ thị tốt nhất là

sử dụng danh sách kề hoặc danh sách liên thuộc. Tuy nhiên với thuật toán

Kosaraju-Sharir, việc cài ñặt bằng dach sách liên thuộc là hợp lí hơn bởi nó cho

phép chuyển từ cách biểu diễn forward star sang cách biểu diễn reverse star một

cách dễ dàng bằng cách chỉnh lại mảng s\_x0012\_4 và

l. Cấu trúc forward star ñược

sử dụng ở pha ñánh số lại các ñỉnh, còn cấu trúc reverse star ñược sử dụng khi liệt

kê các thành phần liên thông mạnh (bởi cần thực hiện trên ñồ thị ñảo chiều)

KOSARAJUSHARIR.PAS Thuật toán Kosaraju–Sharir

{$MODE OBJFPC}

{$M 4000000}

program StronglyConnectedComponents;

const

maxN = 100000;

maxM = 1000000;

1

2

3 4

5 6

7

8 11

9 10

11

6

5 4

3 2

1

10 7

9 8

178

type

TEdge = record //Cấu trúc cung

x, y: Integer; //Hai đỉnh ñầu mút

end;

var

e: array[1..maxM] of TEdge; //Danh sách cạnh

link: array[1..maxM] of Integer; //link[i]: Chỉ số cung kế tiếp e[i] trong

danh sách liên thuộc

head: array[1..maxN] of Integer; //head[u]: Chỉ số cung ñầu tiên trong

danh sách liên thuộc

avail: array[1..maxN] of Boolean;

List: array[1..maxN] of Integer;

Top: Integer;

n, m, v, SCC: Integer;

procedure Enter; //Nhập dữ liệu

var

i, u, v: Integer;

begin

ReadLn(n, m);

for i := 1 to m do

with e[i] do

ReadLn(x, y);

end;

procedure Numbering; //Liệt kê các ñỉnh theo thứ tự duyệt xong vào danh sách List

var

i, u: Integer;

procedure DFSVisit(u: Integer); //Thuật toán DFS từ u

var

i, v: Integer;

begin

avail[u] := False;

i := head[u];

while i <> 0 do //Xét các cung e[i] đi ra khỏi u

begin

v := e[i].y;

if avail[v] then DFSVisit(v);

i := link[i];

end;

Inc(Top); List[Top] := u; //u duyệt xong, ñưa u vào cuối danh sách List

end;

179

begin

//Xây dựng danh sách liên thuộc dạng forward star: Mỗi ñỉnh u tương ứng với danh sách

các cung ñi ra khỏi u

FillChar(head[1], n \* SizeOf(head[1]), 0);

for i := m downto 1 do

with e[i] do

begin

link[i] := head[x];

head[x] := i;

end;

FillChar(avail[1], n \* SizeOf(avail[1]), True);

Top := 0; //Khởi tạo danh sách List rỗng

for u := 1 to n do

if avail[u] then DFSVisit(u);

end;

procedure KosarajuSharir;

var

i, u: Integer;

procedure Enum(u: Integer); //Thuật toán DFS từ u trên đồ thị ñảo chiều

var

i, v: Integer;

begin

avail[u] := False;

Write(u, ', ');

i := head[u];

while i <> 0 do //Xét các cung e[i] đi vào u

begin

v := e[i].x;

if avail[v] then Enum(v);

i := link[i];

end;

end;

begin

//Xây dựng danh sách liên thuộc dạng reverse star: mỗi ñỉnh u tương ứng với danh sách

các cung ñi vào u

FillChar(head[1], n \* SizeOf(head[1]), 0);

for i := m downto 1 do

with e[i] do

begin

link[i] := head[y];

head[y] := i;

end;

180

FillChar(avail[1], n \* SizeOf(avail[1]), True);

SCC := 0;

for u := n downto 1 do

if avail[List[u]] then //Liệt kê thành phần liên thông chốt List[u]

begin

Inc(SCC);

WriteLn('Strongly Connected Component ', SCC, ': ');

Enum(List[u]);

WriteLn;

end;

end;

begin

Enter;

Numbering;

KosarajuSharir;

end.

Thời gian thực hiện giải thuật có thể tính bằng hai lượt DFS, vậy nên thời gian

thực hiện giải thuật sẽ là Θ|| || trong trường hợp ñồ thị ñược biểu diễn

bằng danh sách kề hoặc danh sách liên thuộc, là Θ||

5

nếu dùng ma trận kề và

là Θ|||| nếu dùng danh sách cạnh.

4.5. Sắp xếp tô pô

Hình 5-23. ðồ thị có hướng và ñồ thị các thành phần liên thông mạnh

Xét ñồ thị có hướng , , ta xây dựng ñồ thị có hướng



 ,  như sau: Mỗi ñỉnh thuộc

 tương ứng với một thành phần liên

thông mạnh của . Một cung ,

 nếu và chỉ nếu tồn tại một cung

, trên trong ñó ;

.

{1}

{2,3,4} {8,9,10,11}

{5, 6, 7}

1

2

3 4

5 6

7

8 11

9 10

181

ðồ thị

 gọi là ñồ thị các thành phần liên thông mạnh

ðồ thị

 là ñồ thị có hướng không có chu trình (directed acyclic graph-DAG)

vì nếu

 có chu trình, ta có thể hợp tất cả các thành phần liên thông mạnh

tương ứng với các ñỉnh dọc trên chu trình ñể ñược một thành phần liên thông

mạnh lớn trên ñồ thị , mâu thuẫn với tính tối ñại của một thành phần liên thông

mạnh.

Trong thuật toán Tarjan, khi một thành phần liên thông mạnh ñược liệt kê, thành

phần ñó sẽ tương ứng với một ñỉnh không có cung ñi ra trên

 . Còn trong

thuật toán Kosaraju–Sharir, khi một thành phần liên thông mạnh ñược liệt kê,

thành phần ñó sẽ tương ứng với một ñỉnh không có cung ñi vào trên

 . Cả hai

thuật toán ñều loại bỏ thành phần liên thông mạnh mỗi khi liệt kê xong, tức là loại

bỏ ñỉnh tương ứng trên



.

Nếu ta ñánh số các ñỉnh của

 từ 1 trở ñi theo thứ tự các thành phần liên thông

mạnh ñược liệt kê thì thuật toán Kosaraju–Sharir sẽ cho ta một cách ñánh số gọi là

sắp xếp tô pô (topological sorting) trên

 : Các cung trên

 khi ñó sẽ chỉ

nối từ ñỉnh mang chỉ số nhỏ tới ñỉnh mang chỉ số lớn. Nếu ñánh số các ñỉnh của

 theo thuật toán Tarjan thì ngược lại, các cung trên

 khi ñó sẽ chỉ nối từ

ñỉnh mang chỉ số lớn tới ñỉnh mang chỉ số nhỏ.

Bài tập

5.14. Chứng minh rằng ñồ thị có hướng , là không có chu trình nếu và

chỉ nếu quá trình thực hiện thuật toán tìm kiếm theo chiều sâu trên không

có cung ngược.

5.15. Cho ñồ thị có hướng không có chu trình , và hai ñỉnh

, . Hãy

tìm thuật toán ñếm số ñường ñi từ

tới (chỉ cần ñếm số lượng, không cần

liệt kê các ñường).

5.16. Trên mặt phẳng với hệ toạ ñộ Decattes vuông góc cho ñường tròn, mỗi

ñường tròn xác ñịnh bởi bộ 3 số thực J, K, ở ñây J, K là toạ ñộ tâm và

là bán kính. Hai ñường tròn gọi là thông nhau nếu chúng có ñiểm chung.

Hãy chia các ñường tròn thành một số tối thiểu các nhóm sao cho hai ñường

tròn bất kì trong một nhóm bất kì có thể ñi ñược sang nhau sau một số hữu

hạn các bước di chuyển giữa hai ñường tròn thông nhau.

5.17. Cho một lưới ô vuông kích thước I W gồm các số nhị phân @0,1A

I, 3 1000. Ta ñịnh nghĩa một hình là một miền liên thông các ô kề

cạnh mang số 1. Hai hình ñược gọi là giống nhau nếu hai miền liên thông

tương ứng có thể ñặt chồng khít lên nhau qua một phép dời hình. Hãy phân

182

loại các hình trong lưới ra thành một số các nhóm thỏa mãn: Mỗi nhóm gồm

các hình giống nhau và hai hình bất kì thuộc thuộc hai nhóm khác nhau thì

không giống nhau:

5.18. Cho ñồ thị có hướng , , hãy tìm thuật toán và viết chương trình ñể

chọn ra một tập ít nhất các ñỉnh  = ñể mọi ñỉnh của ñều có thể ñến

ñược từ ít nhất một ñỉnh của  bằng một ñường ñi trên .

5.19. Một ñồ thị có hướng , gọi là nửa liên thông (semi-connected) nếu

với mọi cặp ñỉnh , thì hoặc có ñường ñi ñến , hoặc có ñường ñi

ñến .

a) Chứng minh rằng ñồ thị có hướng , là nửa liên thông nếu và

chỉ nếu trên tồn tại ñường ñi qua tất cả các ñỉnh (không nhất thiết phải là

ñường ñi ñơn)

b) Tìm thuật toán và viết chương trình kiểm tra tính nửa liên thông của

ñồ thị.

5. Vài ứng dụng của DFS và BFS

5.1. Xây dựng cây khung của ñồ thị

Cây là ñồ thị vô hướng, liên thông, không có chu trình ñơn. ðồ thị vô hướng

không có chu trình ñơn gọi là rừng (hợp của nhiều cây). Như vậy mỗi thành phần

liên thông của rừng là một cây.

Xét ñồ thị , và z , {

là một ñồ thị con của ñồ thị ({ = ),

nếu z là một cây thì ta gọi z là cây khung hay cây bao trùm (spanning tree) của

ñồ thị . ðiều kiện cần và ñủ ñể một ñồ thị vô hướng có cây khung là ñồ thị ñó

phải liên thông.

Dễ thấy rằng với một ñồ thị vô hướng liên thông có thể có nhiều cây khung

(Hình 5-24).

1 1 1 0 1 1 0 0 1

1 0 0 0 1 0 0 1 1

1 1 0 0 0 0 0 0 0

1 0 0 1 0 0 0 0 0

1 0 0 1 0 0 0 0 0

0 0 1 1 0 1 0 0 0

1 0 0 0 0 1 0 0 1

1 0 1 0 0 1 1 0 1

1 1 1 1 1 0 0 1 1

1 1 1 0 2 2 0 0 2

1 0 0 0 2 0 0 2 2

1 1 0 0 0 0 0 0 0

1 0 0 3 0 0 0 0 0

1 0 0 3 0 0 0 0 0

0 0 3 3 0 3 0 0 0

1 0 0 0 0 3 0 0 3

1 0 1 0 0 3 3 0 3

1 1 1 1 1 0 0 3 3

183

Hình 5-24: ðồ thị và một số ví dụ cây khung

ðịnh lí 5-14 (Daisy Chain Theorem)

Giả sử z , là ñồ thị vô hướng với ñỉnh. Khi ñó các mệnh ñề sau

là tương ñương:

1. z là cây

2. z không chứa chu trình ñơn và có O 1 cạnh

3.z liên thông và mỗi cạnh của nó ñều là cầu

4.Giữa hai ñỉnh bất kì của z ñều tồn tại ñúng một ñường ñi ñơn

5.z không chứa chu trình ñơn nhưng hễ cứ thêm vào một cạnh ta thu ñược

một chu trình ñơn.

6. z liên thông và có O 1 cạnh

Chứng minh:

1⇒2:

Từ z là cây, theo ñịnh nghĩa z không chứa chu trình ñơn. Ta sẽ chứng minh cây z

có ñỉnh thì phải có O 1 cạnh bằng quy nạp theo số ñỉnh . Rõ ràng khi 1

thì cây có 1 ñỉnh sẽ chứa 0 cạnh. Nếu 1, gọi ( )+, 5, … , -

. là ñường ñi

dài nhất (qua nhiều cạnh nhất) trong z. ðỉnh + không thể kề với ñỉnh nào trong

số các ñỉnh D, E, … , -, bởi nếu có cạnh +, ¡ 3 3 3 4, ta sẽ thiết lập

ñược chu trình ñơn )+, 5, … , ¡, +

.. Mặt khác, ñỉnh + cũng không thể kề với

ñỉnh nào khác ngoài các ñỉnh trên ñường ñi ( trên bởi nếu có cạnh \*, +

,

\* \ ( thì ta thiết lập ñược ñường ñi )\*, +, 5, … , -

. dài hơn (. Vậy ñỉnh +

chỉ có ñúng một cạnh nối với 5, nói cách khác, + là ñỉnh treo. Loại bỏ + và

cạnh +, 5

khỏi z, ta ñược ñồ thị mới cũng là cây và có O 1 ñỉnh, cây này

theo giả thiết quy nạp có O 2 cạnh. Vậy cây z có O 1 cạnh.

z+ z5 zD

184

2⇒3:

Giả sử z có 4 thành phần liên thông z+, z5, … , z-. Vì z không chứa chu trình ñơn

nên các thành phần liên thông của z cũng không chứa chu trình ñơn, tức là các

z+, z5, … , z- ñều là cây. Gọi +, 5, … , - lần lượt là số ñỉnh của z+, z5, … , z- thì

cây z+ có + O 1 cạnh, cây z5 có 5 O 1 cạnh…, cây z- có - O 1 cạnh. Cộng lại

ta có số cạnh của z là O 4 cạnh. Theo giả thiết, cây z có O 1 cạnh, suy ra

4 1, ñồ thị chỉ có một thành phần liên thông là ñồ thị liên thông.

Bây giờ khi z ñã liên thông, kết hợp với giả thiết z không có chu trình nên nếu bỏ

ñi một cạnh bất kì thì ñồ thị mới vẫn không chứa chu trình. ðồ thị mới này không

thể liên thông vì nếu không nó sẽ phải là một cây và theo chứng mình trên, ñồ thị

mới sẽ có O 1 cạnh, tức là z có cạnh. Mâu thuẫn này chứng tỏ tất cả các cạnh

của z ñều là cầu.

3⇒4:

Gọi J và K là 2 ñỉnh bất kì trong z, vì z liên thông nên sẽ có một ñường ñi ñơn từ

J tới K. Nếu tồn tại một ñường ñi ñơn khác từ J tới K thì nếu ta bỏ ñi một cạnh

, nằm trên ñường ñi thứ nhất nhưng không nằm trên ñường ñi thứ hai thì từ

vẫn có thể ñến ñược bằng cách: ñi từ ñi theo chiều tới J theo các cạnh thuộc

ñường thứ nhất, sau ñó ñi từ J tới K theo ñường thứ hai, rồi lại ñi từ K tới theo

các cạnh thuộc ñường ñi thứ nhất. ðiều này chỉ ra việc bỏ ñi cạnh , không

ảnh hưởng tới việc có thể ñi lại ñược giữa hai ñỉnh bất kì. Mâu thuẫn với giả thiết

, là cầu.

4⇒5:

Thứ nhất z không chứa chu trình ñơn vì nếu z chứa chu trình ñơn thì chu trình ñó

qua ít nhất hai ñỉnh , . Rõ ràng dọc theo các cạnh trên chu trình ñó thì từ có

hai ñường ñi ñơn tới . Vô lí.

Giữa hai ñỉnh , bất kì của z có một ñường ñi ñơn nối với , vậy khi thêm

cạnh , vào ñường ñi này thì sẽ tạo thành chu trình.

5⇒6:

Gọi và là hai ñỉnh bất kì trong z, thêm vào z một cạnh , nữa thì theo giả

thiết sẽ tạo thành một chu trình chứa cạnh , . Loại bỏ cạnh này ñi thì phần còn

lại của chu trình sẽ là một ñường ñi từ tới . Mọi cặp ñỉnh của z ñều có một

ñường ñi nối chúng tức là z liên thông, theo giả thiết z không chứa chu trình ñơn

nên z là cây và có O 1 cạnh.

6⇒1:

185

Giả sử z không là cây thì z có chu trình, huỷ bỏ một cạnh trên chu trình này thì z

vẫn liên thông, nếu ñồ thị mới nhận ñược vẫn có chu trình thì lại huỷ một cạnh

trong chu trình mới. Cứ như thế cho tới khi ta nhận ñược một ñồ thị liên thông

không có chu trình. ðồ thị này là cây nhưng lại có  O 1 cạnh (vô lí). Vậy z là

cây.

ðịnh lí 5-15

Số cây khung của ñồ thị ñầy ñủ BC là

C'5

.

Ta sẽ khảo sát hai thuật toán tìm cây khung trên ñồ thị vô hướng liên

thông , .

a) Xây dựng cây khung bằng thuật toán hợp nhất

Trước hết, ñặt z , t; z không chứa cạnh nào thì có thể coi z gồm || cây

rời rạc, mỗi cây chỉ có 1 ñỉnh. Sau ñó xét lần lượt các cạnh của , nếu cạnh ñang

xét nối hai cây khác nhau trong z thì thêm cạnh ñó vào z, ñồng thời hợp nhất hai

cây ñó lại thành một cây. Cứ làm như vậy cho tới khi kết nạp ñủ || O 1 cạnh vào

z thì ta ñược z là cây khung của ñồ thị. Trong việc xây dựng cây khung bằng

thuật toán hợp nhất, một cấu trúc dữ liệu biểu diễn các tập rời nhau thường ñược

sử dụng ñể tăng tốc phép hợp nhất hai cây cũng như phép kiểm tra hai ñỉnh có

thuộc hai cây khác nhau không.

b) Xây dựng cây khung bằng các thuật toán tìm kiếm trên ñồ thị.

Hình 5-25: Cây khung DFS và cây khung BFS trên cùng một ñồ thị (mũi tên chỉ chiều ñi thăm

các ñỉnh)

Áp dụng thuật toán BFS hay DFS bắt ñầu từ ñỉnh

nào ñó, tại mỗi bước từ ñỉnh

tới thăm ñỉnh , ta thêm vào thao tác ghi nhận luôn cạnh , vào cây khung. Do

ñồ thị liên thông nên thuật toán sẽ xuất phát từ

và tới thăm tất cả các ñỉnh còn

lại, mỗi ñỉnh ñúng một lần, tức là quá trình duyệt sẽ ghi nhận ñược ñúng || O 1

cạnh. Tất cả những cạnh ñó không tạo thành chu trình ñơn bởi thuật toán không

1

2 3

4 5 6 7

1

2 3

4 5 6 7

Cây DFS Cây BFS

186

thăm lại những ñỉnh ñã thăm. Theo mệnh ñề tương ñương thứ hai, ta có những

cạnh ghi nhận ñược tạo thành một cây khung của ñồ thị.

5.2. Tập các chu trình cơ sở của ñồ thị

Xét một ñồ thị vô hướng liên thông , ; gọi z , {

là một cây khung

của nó. Các cạnh của cây khung ñược gọi là các cạnh trong, còn các cạnh khác là

các cạnh ngoài cây.

Nếu thêm một cạnh ngoài

O { vào cây khung z, thì ta ñược ñúng một chu

trình ñơn trong z, kí hiệu chu trình này là

¢

. Chu trình

¢

chỉ chứa duy nhất một

cạnh ngoài cây còn các cạnh còn lại ñều là cạnh trong cây z

Tập các chu trình:

Ψ @

¢

|

O {

A

ñược gọi là tập các chu trình cơ sở của ñồ thị .

Các tính chất quan trọng của tập các chu trình cơ sở:

• Tập các chu trình cơ sở là phụ thuộc vào cây khung, hai cây khung khác nhau

có thể cho hai tập chu trình cơ sở khác nhau.

• Cây khung của ñồ thị liên thông , luôn chứa || O 1 cạnh, còn lại

|| O || 1 cạnh ngoài. Tương ứng với mỗi cạnh ngoài có một chu trình cơ

sở, vậy số chu trình cơ sở của ñồ thị liên thông là || O || 1.

• Tập các chu trình cơ sở là tập nhiều nhất các chu trình thoả mãn: Mỗi chu

trình có ñúng một cạnh riêng, cạnh ñó không nằm trong bất cứ một chu trình

nào khác. ðiều này có thể chứng minh ñược bằng cách lấy trong ñồ thị liên

thông một tập gồm 4 chu trình thoả mãn ñiều ñó thì việc loại bỏ cạnh riêng

của một chu trình sẽ không làm mất tính liên thông của ñồ thị, ñồng thời

không ảnh hưởng tới sự tồn tại của các chu trình khác. Như vậy nếu loại bỏ

tất cả các cạnh riêng thì ñồ thị vẫn liên thông và còn || O 4 cạnh. ðồ thị liên

thông thì không thể có ít hơn || O 1 cạnh nên ta có || O 4 7 || O 1 hay

4 3 || O || 1.

• Mọi cạnh trong một chu trình ñơn bất kì ñều phải thuộc một chu trình cơ sở.

Bởi nếu có một cạnh , không thuộc một chu trình cơ sở nào, thì khi ta bỏ

cạnh ñó ñi ñồ thị vẫn liên thông và không ảnh hưởng tới sự tồn tại của các

chu trình cơ sở. Lại bỏ tiếp || O || 1 cạnh ngoài của các chu trình cơ sở

thì ñồ thị vẫn liên thông và còn lại || O 2 cạnh. ðiều này vô lí.

ðối với ñồ thị , có 4 thành phần liên thông, ta có thể xét các thành phần

liên thông và xét rừng các cây khung của các thành phần ñó. Khi ñó có thể mở

187

rộng khái niệm tập các chu trình cơ sở cho ñồ thị vô hướng tổng quát: Mỗi khi

thêm một cạnh không nằm trong các cây khung vào rừng, ta ñược ñúng một chu

trình ñơn, tập các chu trình ñơn tạo thành bằng cách ghép các cạnh ngoài như vậy

gọi là tập các chu trình cơ sở của ñồ thị . Số các chu trình cơ sở là || O || 4.

5.3. Bài toán ñịnh chiều ñồ thị

Bài toán ñặt ra là cho một ñồ thị vô hướng liên thông , , hãy thay mỗi

cạnh của ñồ thị bằng một cung ñịnh hướng ñể ñược một ñồ thị có hướng liên

thông mạnh. Nếu có phương án ñịnh chiều như vậy thì ñược gọi là ñồ thị ñịnh

chiều ñược. Bài toán ñịnh chiều ñồ thị có ứng dụng rõ nhất trong sơ ñồ giao thông

ñường bộ. Chẳng hạn như trả lời câu hỏi: Trong một hệ thống ñường phố, liệu có

thể quy ñịnh các ñường phố ñó thành ñường một chiều mà vẫn ñảm bảo sự ñi lại

giữa hai nút giao thông bất kì hay không.

Có thể tổng quát hoá bài toán ñịnh chiều ñồ thị: Với ñồ thị vô hướng ,

hãy tìm cách thay mỗi cạnh của ñồ thị bằng một cung ñịnh hướng ñể ñược ñồ thị

mới có ít thành phần liên thông mạnh nhất. Dưới ñây ta xét một tính chất hữu ích

của thuật toán thuật toán tìm kiếm theo chiều sâu ñể giải quyết bài toán ñịnh chiều

ñồ thị

Xét mô hình duyệt ñồ thị bằng thuật toán tìm kiếm theo chiều sâu, tuy nhiên trong

quá trình duyệt, mỗi khi xét qua cạnh , thì ta ñịnh chiều luôn cạnh ñó thành

cung , . Nếu coi một cạnh của ñồ thị tương ñương với hai cung có hướng

ngược chiều nhau thì việc ñịnh chiều cạnh , thành cung , tương ñương

với việc loại bỏ cung , của ñồ thị. Ta có một phép ñịnh chiều gọi là phép

ñịnh chiều DFS.

Hình 5-26. Phép ñịnh chiều DFS

Thuật toán thực hiện phép ñịnh chiều DFS có thể viết như sau:

1

2 3

4 5 6

7 8 9 10

1

2 3

4 5 6

7 8 9 10

188

procedure DFSVisit(u V);

begin

«Thông báo thăm u và ñánh dấu u ñã thăm»;

for ∀v:(u, v) E do

begin

«ðịnh chiều cạnh (u, v) thành cung (u, v) xoá cung

(v, u) khỏi ñồ thị»;

if «v chưa thăm» then

DFSVisit(v);

end;

end;

begin

«ðánh dấu mọi ñỉnh ñều chưa thăm»;

for ∀vV do

if «v chưa thăm» then DFSVisit(v);

end;

Thuật toán DFS sẽ cho ta một rừng các cây DFS và các cung ngoài cây. Ta có các

tính chất sau:

ðịnh lí 5-16

Sau quá trình duyệt DFS và ñịnh chiều, ñồ thị sẽ chỉ còn cung DFS và

cung ngược.

Ch

ng minh

Xét một cạnh , bất kì, không giảm tính tổng quát, giả sử rằng ñược duyệt ñến trước

. Theo ðịnh lí 5-8 (ñịnh lí ñường ñi trắng), ta có là hậu duệ của . Nhìn vào mô

hình cài ñặt thuật toán, có nhận xét rằng việc ñịnh chiều cạnh , chỉ có thể ñược thực

hiện trong thủ tục y\_x0012\_

\_x0012\_ hoặc trong thủ tục y\_x0012\_

\_x0012\_.

Nếu cạnh , ñược ñịnh chiều trước khi ñỉnh ñược duyệt ñến, nghĩa là việc ñịnh chiều

ñược thực hiện trong thủ tục y\_x0012\_

\_x0012\_, và ngay sau khi cạnh , ñược ñịnh chiều

thành cung , thì ñỉnh sẽ ñược thăm. ðiều ñó chỉ ra rằng cung , là cung DFS.

Nếu cạnh , ñược ñịnh chiều sau khi ñỉnh ñược duyệt ñến, nghĩa là khi thủ tục

y\_x0012\_

\_x0012\_ ñược gọi thì cạnh , chưa ñịnh chiều. Vòng lặp bên trong thủ tục

y\_x0012\_

\_x0012\_ chắc chắn sẽ quét vào cạnh này và ñịnh chiều thành cung ngược , .

Trong ñồ thị vô hướng ban ñầu, cạnh bị ñịnh hướng thành cung ngược chính là

cạnh ngoài của cây DFS. Chính vì vậy, mọi chu trình cơ sở của cây DFS trong ñồ

thị vô hướng ban ñầu vẫn sẽ là chu trình trong ñồ thị có hướng tạo ra. ðây là một

phương pháp hiệu quả ñể liệt kê các chu trình cơ sở của cây khung DFS: Vừa

duyệt DFS vừa ñịnh chiều, nếu duyệt phải cung ngược , thì truy vết ñường ñi

của DFS ñể tìm ñường từ ñến , sau ñó nối thêm cung ngược , ñể ñược

một chu trình cơ sở.

189

ðịnh lí 5-17

ðiều kiện cần và ñủ ñể một ñồ thị vô hướng liên thông có thể ñịnh chiều

ñược là mỗi cạnh của ñồ thị nằm trên ít nhất một chu trình ñơn (hay nói

cách khác mọi cạnh của ñồ thị ñều không phải là cầu).

Ch

ng minh

Gọi , là một ñồ thị vô hướng liên thông.

"⇒"

Nếu là ñịnh chiều ñược thì sau khi ñịnh hướng sẽ ñược ñồ thị liên thông mạnh <. Với

một cạnh , ñược ñịnh chiều thành cung , thì sẽ tồn tại một ñường ñi ñơn trong <

theo các cạnh ñịnh hướng từ về . ðường ñi ñó nối thêm cung , sẽ thành một chu

trình ñơn có hướng trong <. Tức là trên ñồ thị ban ñầu, cạnh , nằm trên một chu trình

ñơn.

"⇐"

Nếu mỗi cạnh của ñều nằm trên một chu trình ñơn, ta sẽ chứng minh rằng: phép ñịnh

chiều DFS sẽ tạo ra ñồ thị < liên thông mạnh.

Lấy một cạnh , của , vì , nằm trong một chu trình ñơn, mà mọi cạnh của một

chu trình ñơn ñều phải thuộc một chu trình cơ sở của cây DFS, nên sẽ có một chu trình cơ

sở chứa cạnh , . Có thể nhận thấy rằng chu trình cơ sở của cây DFS qua phép ñịnh

chiều DFS vẫn là chu trình trong < nên theo các cung ñã ñịnh hướng của chu trình ñó ta

có thể ñi từ tới và ngược lại.

Lấy J và K là hai ñỉnh bất kì của , do liên thông, tồn tại một ñường ñi

)J \*

, +

, … , - K.

Vì /

, /&+ là cạnh của nên theo chứng minh trên, từ /

có thể ñi ñến ñược /&+ trên <,

0\_x0012\_: 1 3 \_x0012\_  4, tức là từ J vẫn có thể ñi ñến K bằng các cung ñịnh hướng của <. Suy ra <

là ñồ thị liên thông mạnh

Với những kết quả ñã chứng minh trên, ta còn suy ra ñược: Nếu ñồ thị liên thông

và mỗi cạnh của nó nằm trên ít nhất một chu trình ñơn thì phép ñịnh chiều DFS sẽ

cho một ñồ thị liên thông mạnh. Còn nếu không, thì phép ñịnh chiều DFS sẽ cho

một ñồ thị ñịnh hướng có ít thành phần liên thông mạnh nhất, một cạnh không

nằm trên một chu trình ñơn nào (cầu) của ñồ thị ban ñầu sẽ ñược ñịnh hướng

thành cung nối giữa hai thành phần liên thông mạnh.

5.4. Liệt kê các khớp và cầu của ñồ thị

Nếu trong quá trình ñịnh chiều ta thêm vào ñó thao tác ñánh số các ñỉnh theo thứ tự

duyệt ñến của thuật toán DFS, gọi I

mn là số thứ tự của ñỉnh theo cách

ñánh số ñó. ðịnh nghĩa thêm q\_x0010\_}mn là giá trị I

m. n nhỏ nhất của những

ñỉnh ñến ñược từ nhánh DFS gốc bằng một cung ngược. Tức là nếu nhánh DFS

gốc có nhiều cung ngược hướng lên phía gốc thì ta ghi nhận lại cung ngược

hướng lên cao nhất. Nếu nhánh DFS gốc không chứa cung ngược thì ta cho

190

q\_x0010\_}mn x ∞. Cách tính các giá trị I

m. n và q\_x0010\_}m. n tương tự như trong

thuật toán Tarjan: Trong thủ tục y\_x0012\_

\_x0012\_, trước hết ta ñánh số thứ tự thăm cho

ñỉnh (I

mn) và khởi tạo q\_x0010\_}mn x ∞, sau ñó xét tất cả những ñỉnh kề

, ñịnh chiều cạnh , thành cung , . Có hai khả năng xảy ra:

• Nếu chưa thăm thì ta gọi y\_x0012\_

\_x0012\_ ñể thăm , khi thủ tục y\_x0012\_

\_x0012\_

thoát có nghĩa là ñã xây dựng ñược nhánh DFS gốc nằm trong nhánh DFS

gốc , những cung ngược ñi từ nhánh DFS gốc cũng là cung ngược ñi từ

nhánh DFS gốc ⇒ ta cực tiểu hoá q\_x0010\_}mn theo công thức: q\_x0010\_}mnmới x

minq\_x0010\_}mncũ, q\_x0010\_}mn

• Nếu ñã thăm thì , là một cung ngược ñi từ nhánh DFS gốc ⇒ ta cực

tiểu hoá q\_x0010\_}mn theo công thức: q\_x0010\_}mnmới x minq\_x0010\_}mncũ, I

mn

Hình 5-27. Cách ñánh số và ghi nhận cung ngược lên cao nhất

Hãy ñể ý một cung DFS , ( là nút cha của nút trên cây DFS)

• Nếu từ nhánh DFS gốc không có cung nào ngược lên phía trên có nghĩa

là từ một ñỉnh thuộc nhánh DFS gốc ñi theo các cung ñịnh hướng chỉ ñi

ñược tới những ñỉnh nội bộ trong nhánh DFS gốc mà thôi chứ không thể tới

ñược , suy ra , là một cầu. Cũng dễ dàng chứng minh ñược ñiều ngược

lại. Vậy , là cầu nếu và chỉ nếu q\_x0010\_}mn 7 I

mn. Như ví dụ ở

Hình 5-27, ta có C, F và E, H là cầu.

• Nếu từ nhánh DFS gốc không có cung nào ngược lên phía trên , tức là nếu

bỏ ñi thì từ không có cách nào lên ñược các tiền bối của . ðiều này chỉ

ra rằng nếu không phải là nút gốc của một cây DFS thì là khớp. Cũng

không khó khăn ñể chứng minh ñiều ngược lại. Vậy nếu không là gốc của

A

B C

D E F

G H I J

1

2 3

7 8 4

10 9 5 6

low=1

low=1 low=1

low=2

low=2

low=2 low=+∞ low=4 low=4

low=4

191

một cây DFS thì là khớp nếu và chỉ nếu q\_x0010\_}mn 7 I

mn. Như ví dụ

ở Hình 5-27, ta có B, C, E và F là khớp.

• Gốc của một cây DFS thì là khớp nếu và chỉ nếu nó có từ hai 2 nhánh con trở

lên. Như ví dụ ở Hình 5-27, gốc A không là khớp vì nó chỉ có một nhánh con.

ðến ñây ta ñã có ñủ ñiều kiện ñể giải bài toán liệt kê các khớp và cầu của ñồ thị:

ñơn giản là dùng phép ñịnh chiều DFS ñánh số các ñỉnh theo thứ tự thăm và ghi

nhận cung ngược lên trên cao nhất xuất phát từ một nhánh cây DFS, sau ñó dùng

ba nhận xét kể trên ñể liệt kê ra tất cả các cầu và khớp của ñồ thị.

Input

• Dòng 1: Chứa số ñỉnh 3 1000, số cạnh I của ñồ thị vô hướng .

• I dòng tiếp theo, mỗi dòng chứa hai số , tương ứng với một cạnh ,

của

Output

Các khớp và cầu của

Sample Input Sample Output

11 14

1 2

1 3

1 4

3 4

3 5

3 6

3 8

4 7

5 6

5 8

6 9

7 10

7 11

10 11

Bridges:

(1, 2)

(4, 7)

(6, 9)

Articulations:

1

3

4

6

7

Về kĩ thuật cài ñặt, ngoài các mảng ñã ñược nói tới khi trình bày thuật toán, có

thêm một mảng (

m1 … n, trong ñó (

mn chỉ ra nút cha của nút trên

cây DFS, nếu là gốc của một cây DFS thì (

mn ñược ñặt bằng O1. Công

dụng của mảng (

m1 … n là ñể duyệt tất cả các cung DFS và kiểm tra một

ñỉnh có phải là gốc của cây DFS hay không.

CUTVE.PAS Liệt kê các khớp và cầu của ñồ thị

{$MODE OBJFPC}

program ArticulationsAndBridges;

1

3 4

5 6 7

8 9 10 11

2

192

const

maxN = 1000;

var

a: array[1..maxN, 1..maxN] of Boolean;

Number, Low, Parent: array[1..maxN] of Integer;

n, Count: Integer;

procedure Enter; //Nhập dữ liệu

var

i, m, u, v: Integer;

begin

FillChar(a, SizeOf(a), False);

ReadLn(n, m);

for i := 1 to m do

begin

ReadLn(u, v);

a[u, v] := True;

a[v, u] := True;

end;

end;

//Hàm cực tiểu hoá: Target := min(Target, Value)

procedure Minimize(var Target: Integer; Value: Integer);

begin

if Value < Target then Target := Value;

end;

procedure DFSVisit(u: Integer); //Thuật toán tìm kiếm theo chiều sâu bắt ñầu

từ u

var

v: Integer;

begin

Inc(Count);

Number[u] := Count; //Đánh số u theo thứ tự duyệt ñến

Low[u] := maxN + 1; //Đặt Low[u] := +∞

for v := 1 to n do

if a[u, v] then //Xét các đỉnh v kề u

begin

a[v, u] := False; //Định chiều cạnh (u, v) thành cung (u, v)

if Parent[v] = 0 then //Nếu v chưa thăm

begin

Parent[v] := u; //cung (u, v) là cung DFS

DFSVisit(v); //Đi thăm v

Minimize(Low[u], Low[v]); //Cực tiểu hoá Low[u] theo Low[v]

end

193

else

Minimize(Low[u], Number[v]); //Cực tiểu hoá Low[u] theo

Number[v]

end;

end;

procedure Solve;

var

u, v: Integer;

begin

Count := 0; //Khởi tạo bộ ñếm

FillChar(Parent, SizeOf(Parent), 0); //Các đỉnh đều chưa thăm

for u := 1 to n do

if Parent[u] = 0 then

begin

Parent[u] := -1;

DFSVisit(u);

end;

end;

procedure PrintResult; //In kết quả

var

u, v: Integer;

nChildren: array[1..maxN] of Integer;

IsArticulation: array[1..maxN] of Boolean;

begin

WriteLn('Bridges: '); //Liệt kê các cầu

for v := 1 to n do

begin

u := Parent[v];

if (u <> -1) and (Low[v] >= Number[v]) then

WriteLn('(', u, ', ', v, ')');

end;

WriteLn('Articulations:'); //Liệt kê các khớp

FillChar(nChildren, n \* SizeOf(Integer), 0);

for v := 1 to n do

begin

u := Parent[v];

if u <> -1 then Inc(nChildren[u]);

end;

//Đánh dấu các gốc cây có nhiều hơn 1 nhánh con

for u := 1 to n do

IsArticulation[u] := (Parent[u] = -1) and (nChildren[u]

>= 2);

194

for v := 1 to n do

begin

u := Parent[v];

if (u <> -1) and (Parent[u] <> -1) and (Low[v] >=

Number[u]) then

IsArticulation[u] := True; //Đánh dấu các khớp không phải gốc

cây

end;

for u := 1 to n do //Liệt kê

if IsArticulation[u] then

WriteLn(u);

end;

begin

Enter;

Solve;

PrintResult;

end.

Trong bài toán liệt kê các khớp và cầu của ñồ thị, ta biểu diễn ñồ thị bằng ma trận

kề ñể tiện lợi cho thao tác ñịnh chiều. Nếu ñồ thị có số ñỉnh lớn (không thể biểu

diễn ñược bằng ma trận kề) và số cạnh I nhỏ (ñồ thị thưa), chúng ta phải tìm một

cấu trúc dữ liệu khác ñể biểu diễn ñồ thị ñể chi phí về bộ nhớ và thời gian phụ

thuộc chủ yếu vào I thay vì

5

như ma trận kề. Trong các cấu trúc dữ liệu biểu

diễn ñồ thị phổ biến, chỉ có danh sách kề và danh sách liên thuộc cho phép thực

hiện ñiều này, tuy nhiên việc thực hiện ñịnh chiều cạnh vô hướng thành cung có

hướng sẽ trở nên khá phức tạp.

Error! Reference source not found. yêu cầu bạn sửa ñổi thuật toán ñể bỏ ñi thao

tác ñịnh chiều, từ ñó có thể biểu diễn ñồ thị thưa bởi danh sách kề mà không còn

gặp khó khăn trong việc ñịnh chiều ñồ thị nữa.

5.5. Các thành phần song liên thông

a) Các khái niệm và thuật toán

ðồ thị vô hướng liên thông ñược gọi là ñồ thị song liên thông nếu nó không có

khớp, tức là việc bỏ ñi một ñỉnh bất kì của ñồ thị không ảnh hưởng tới tính liên

thông của các ñỉnh còn lại. Ta quy ước rằng ñồ thị chỉ gồm một ñỉnh và không có

cạnh nào cũng là một ñồ thị song liên thông.

Cho ñồ thị vô hướng , , xét một tập con < § . Gọi < là ñồ thị hạn

chế trên <. ðồ thị < ñược gọi là một thành phần song liên thông của ñồ thị nếu

< song liên thông và không tồn tại ñồ thị con song liên thông nào khác của

195

nhận < làm ñồ thị con. Ta cũng ñồng nhất khái niệm < là thành phần song liên

thông với khái niệm < là thành phần song liên thông.

Cần phân biệt hai khái niệm ñồ thị ñịnh chiều ñược (không có cầu) và ñồ thị song

liên thông (không có khớp). Nếu như ñồ thị không ñịnh chiều ñược thì tập ñỉnh

của có thể phân hoạch thành các tập con rời nhau ñể ñồ thị hạn chế trên các

tập con ñó là các ñồ thị ñịnh chiều ñược. Còn nếu ñồ thị không phải ñồ thị song

liên thông thì tập cạnh của có thể phân hoạch thành các tập con rời nhau ñể trên

mỗi tập con, các cạnh và các ñỉnh ñầu mút của chúng trở thành một ñồ thị song

liên thông. Hai thành phần song liên thông có thể có chung một ñiểm khớp nhưng

không có cạnh nào chung

Hình 5-28. ðồ thị và hai thành phần song liên thông có chung khớp

Xét mô hình ñịnh chiều ñồ thị ñánh số ñỉnh theo thứ tự duyệt ñến và ghi nhận

cung ngược lên cao nhất...

procedure DFSVisit(uV);

begin

Count := Count + 1;

Number[u] := Count; //Đánh s) u theo th

t+ duyt ñ$n

Low[u] := +∞;

for ∀vV:(u, v)E do

begin

«ðịnh chiều cạnh (u, v) thành cung (u, v)»;

if Number[v] > 0 then //v đã thăm

Low[u] := min(Low[u], Number[v])

else // v ch

a thăm

begin

DFSVisit(v); //Đi thăm v

Low[u] := min(Low[u], Low[v]); //C+c ti<u hoá Low[u]

end;

end;

1

2 3

5

4

6

196

end;

begin

Count := 0;

for ∀vV do Number[v] := 0; //Number[v] = 0 ↔ v ch

a thăm

for ∀vV do

if Number[v] = 0 then DFSVisit(v);

end.

Trong thủ tục y\_x0012\_

\_x0012\_, mỗi khi xét các ñỉnh kề chưa ñược thăm, thuật

toán sẽ gọi y\_x0012\_

\_x0012\_ ñể ñi thăm sau ñó cực tiểu hoá q\_x0010\_}mn theo q\_x0010\_}mn.

Tại thời ñiểm này, nếu q\_x0010\_}mn 7 I

mn thì hoặc là khớp hoặc là gốc

của một cây DFS. ðể tiện, trong trường hợp này ta gọi cung , là cung chốt

của thành phần song liên thông.

Thuật toán tìm kiếm theo chiều sâu không chỉ duyệt qua các ñỉnh mà còn duyệt và

ñịnh chiều các cung nữa. Ta sẽ quan tâm tới cả thời ñiểm một cạnh ñược duyệt

ñến, duyệt xong, cũng như thứ tự tiền bối–hậu duệ của các cung DFS: Cung DFS

, ñược coi là tiền bối thực sự của cung DFS 9

, < (hay cung

9

, 9

là hậu

duệ thực sự của cung , ) nếu cung

9

, 9

nằm trong nhánh DFS gốc . Xét

về vị trí trên cây, cung

9

, 9

nằm dưới cung , .

Có thể nhận thấy rằng nếu , là một cung chốt thỏa mãn: Khi y\_x0012\_

\_x0012\_

gọi y\_x0012\_

\_x0012\_ và quá trình tìm kiếm theo chiều sâu tiếp tục từ không thăm

tiếp bất cứ một cung chốt nào (tức là nhánh DFS gốc không chứa cung chốt

nào) thì cung , hợp với tất cả các cung hậu duệ của nó sẽ tạo thành một

nhánh cây mà mọi ñỉnh thuộc nhánh cây ñó là một thành phần song liên thông.

Chính vì vậy thuật toán liệt kê các thành phần song liên thông có tư tưởng khá

giống với thuật toán Tarjan tìm thành phần liên thông mạnh. Việc cài ñặt thuật

toán liệt kê các thành phần song liên thông chính là sự sửa ñổi ñối ngẫu của thuật

toán Tarjan: Thay khái niệm “chốt” bằng “cung chốt” và thay vì dùng ngăn xếp

chứa chốt và các ñỉnh hậu duệ của chốt ñể liệt kê các thành phần liên thông mạnh,

chúng ta sẽ dùng ngăn xếp chứa cung chốt và các hậu duệ của cung chốt ñể liệt kê

các thành phần song liên thông.

Vấn ñề rắc rối duy nhất gặp phải là quy ước một ñỉnh cô lập của ñồ thị cũng là

một thành phần song liên thông. Nếu thực hiện thuật toán trên, thành phần song

liên thông chỉ gồm duy nhất một ñỉnh sẽ không có cung chốt nào cả và như vậy sẽ

bị sót khi liệt kê. Ta sẽ phải xử lí các ñỉnh cô lập như trường hợp riêng khi liệt kê

các thành phần song liên thông của ñồ thị.

procedure DFSVisit(uV);

197

begin

Count := Count + 1;

Number[u] := Count; //Đánh s) u theo th

t+ duyt ñ$n

Low[u] := +∞;

for ∀vV:(u, v)E do

begin

«ðịnh chiều cạnh (u, v) thành cung (u, v)»;

if Number[v] > 0 then //v đã thăm

Low[u] := min(Low[u], Number[v])

else // v ch

a thăm

begin

Push((u, v)); //Đ,y cung (u, v) vào ngăn x$p

DFSVisit(v); //Đi thăm v

Low[u] := min(Low[u], Low[v]); //C+c ti<u hoá Low[u]

if Low[v] ≥ Number[u] then //(u, v) là cung ch)t

begin

«Thông báo thành phần song liên thông với cung

chốt (u, v):»;

repeat

(p, q) := Pop; //Ly t2 ngăn x$p ra m\_x0019\_t cung (p, q)

Output ← q; //Lit kê các ñnh nên ch cCn xut ra m\_x0019\_t ñCu mút

until (p, q) = (u, v);

Output ← u; //Còn thi$u ñnh u, lit kê n)t

end;

end;

end;

end;

begin

Count := 0;

for ∀vV do Number[v] := 0; //Number[v] = 0 ↔ v chưa thăm

Stack := ∅;

for ∀vV do

if Number[v] = 0 then

begin

DFSVisit(v);

if «v là ñỉnh cô lập» then

«Liệt kê thành phần song liên thông chỉ gồm một

ñỉnh v»

end;

end.

198

b) Cài ñặt

Về kĩ thuật cài ñặt không có gì mới, có một chú ý nhỏ là chúng ta chỉ dùng ngăn

xếp   
4 ñể chứa các cung DFS, vì vậy   
4 không bao giờ phải chứa quá

O 1 cung

Input

• Dòng 1: Chứa số ñỉnh 3 1000 và số cạnh I của một ñồ thị vô hướng

• I dòng tiếp theo, mỗi dòng chứa hai số , tương ứng với một cạnh ,

của ñồ thị.

Output

Các thành phần song liên thông của ñồ thị

Sample

Input

Sample Output

9 10

1 3

1 4

3 4

3 6

3 7

4 8

4 9

5 9

6 7

8 9

Biconnected

component: 1

5, 9

Biconnected

component: 2

9, 8, 4

Biconnected

component: 3

7, 6, 3

Biconnected

component: 4

4, 3, 1

Biconnected

component: 5

2

BCC.PAS Liệt kê các thành phần song liên thông

{$MODE OBJFPC}

program BiconnectedComponents;

const

maxN = 1000;

type

TStack = record

x, y: array[1..maxN - 1] of Integer;

Top: Integer;

end;

var

a: array[1..maxN, 1..maxN] of Boolean;

1

2 3 4

6 7 8 9

5

199

Number, Low: array[1..maxN] of Integer;

Stack: TStack;

BCC, PrevCount, Count, n, u: Integer;

procedure Enter; //Nhập dữ liệu

var

i, m, u, v: Integer;

begin

FillChar(a, SizeOf(a), False);

ReadLn(n, m);

for i := 1 to m do

begin

ReadLn(u, v);

a[u, v] := True;

a[v, u] := True;

end;

end;

procedure Push(u, v: Integer); //Đẩy một cung (u, v) vào ngăn xếp

begin

with Stack do

begin

Inc(Top);

x[Top] := u;

y[Top] := v;

end;

end;

procedure Pop(var u, v: Integer); //Lấy một cung (u, v) khỏi ngăn xếp

begin

with Stack do

begin

u := x[Top];

v := y[Top];

Dec(Top);

end;

end;

//Hàm cực tiểu hoá: Target := min(Target, Value)

procedure Minimize(var Target: Integer; Value: Integer);

begin

if Value < Target then Target := Value;

end;

procedure DFSVisit(u: Integer); //Thuật toán tìm kiếm theo chiều sâu

var

v, p, q: Integer;

200

begin

Inc(Count);

Number[u] := Count;

Low[u] := maxN + 1;

for v := 1 to n do

if a[u, v] then //Xét mọi cạnh (u, v)

begin

a[v, u] := False; //Định chiều luôn

if Number[v] <> 0 then //v đã thăm

Minimize(Low[u], Number[v])

else //v chưa thăm

begin

Push(u, v); //Đẩy cung DFS (u, v) vào Stack

DFSVisit(v); //Tiếp tục quá trình DFS từ v

Minimize(Low[u], Low[v]);

if Low[v] >= Number[u] then //Nếu (u, v) là cung chốt

begin //Liệt kê thành phần song liên thông với cung chốt (u, v)

Inc(BCC);

WriteLn('Biconnected component: ', BCC);

repeat

Pop(p, q); //Lấy một cung DFS (p, q) khỏi Stack

Write(q, ', '); //Chỉ in ra một ñầu cung, tránh in lặp

until (p = u) and (q = v); //Đến khi lấy ra cung (u, v)

thì dừng

WriteLn(u); //In nốt ra ñỉnh u

end;

end;

end;

end;

begin

Enter;

FillChar(Number, n \* SizeOf(Integer), 0);

Stack.Top := 0;

Count := 0;

BCC := 0;

for u := 1 to n do

if Number[u] = 0 then

begin

PrevCount := Count;

DFSVisit(u);

if Count = PrevCount + 1 then //u là đỉnh cô lập

begin

201

Inc(BCC);

WriteLn('Biconnected component: ', BCC);

WriteLn(u);

end;

end;

end.

Bài tập

5.20. Hãy sửa ñổi thuật toán liệt kê khớp và cầu của ñồ thị, sửa ñổi thuật toán liệt

kê các thành phần song liên thông sao cho không cần phải thực hiện việc

ñịnh chiều ñồ thị nữa (Bởi vì việc ñịnh chiều một ñồ thị tỏ ra khá cồng kềnh

và không hiệu quả nếu ñồ thị ñược biểu diễn bằng danh sách kề hay danh

sách cạnh)

5.21. Tìm thuật toán ñếm số cây khung của ñồ thị (Hai cây khung gọi là khác

nhau nếu chúng có ít nhất một cạnh khác nhau)

6. ðồ thị Euler và ñồ thị Hamilton

6.1. ðồ thị Euler

a) Bài toán

Bài toán về ñồ thị Euler ñược coi là bài toán ñầu tiên của lí thuyết ñồ thị. Bài toán

này xuất phát từ một bài toán nổi tiếng: Bài toán bảy cây cầu ở Königsberg:

Thành phố Königsberg thuộc ðức (nay là Kaliningrad thuộc Cộng hoà Nga), ñược

chia làm 4 vùng bằng các nhánh sông Pregel. Các vùng này gồm 2 vùng bên bờ

sông (B, C), ñảo Kneiphof (A) và một miền nằm giữa hai nhánh sông Pregel (D).

Vào thế kỉ XVIII, người ta ñã xây 7 chiếc cầu nối những vùng này với nhau.

Người dân ở ñây tự hỏi: Liệu có cách nào xuất phát tại một ñịa ñiểm trong thành

phố, ñi qua 7 chiếc cầu, mỗi chiếc ñúng 1 lần rồi quay trở về nơi xuất phát không

?

Nhà toán học Thụy sĩ Leonhard Euler ñã giải bài toán này và có thể coi ñây là ứng

dụng ñầu tiên của Lí thuyết ñồ thị, ông ñã mô hình hoá sơ ñồ 7 cái cầu bằng một

ña ñồ thị, bốn vùng ñược biểu diễn bằng 4 ñỉnh, các cầu là các cạnh. Bài toán tìm

ñường qua 7 cầu, mỗi cầu ñúng một lần có thể tổng quát hoá bằng bài toán: Có

tồn tại chu trình trong ña ñồ thị ñi qua tất cả các cạnh và mỗi cạnh ñúng một lần.

202

Hình 5-29: Mô hình ñồ thị của bài toán bảy cái cầu

Chu trình qua tất cả các cạnh của ñồ thị, mỗi cạnh ñúng một lần ñược gọi là chu

trình Euler (Euler circuit/Euler circle/Euler tour). ðường ñi qua tất cả các cạnh

của ñồ thị, mỗi cạnh ñúng một lần gọi là ñường ñi Euler (Euler path/Euler

trail/Euler walk). Một ñồ thị có chu trình Euler ñược gọi là ñồ thị Euler (Eulerian

graph/unicursal graph). Một ñồ thị có ñường ñi Euler ñược gọi là ñồ thị nửa

Euler (Semi-Eulerian graph/Traversable graph).

b) Các ñịnh lí và thuật toán

ðịnh lí 5-18 (Euler)

Một ñồ thị vô hướng liên thông , có chu trình Euler khi và chỉ

khi mọi ñỉnh của nó ñều có bậc chẵn.

Ch

ng minh

Nếu có chu trình Euler thì khi ñi dọc chu trình ñó, mỗi khi ñi qua một ñỉnh thì bậc của

ñỉnh ñó tăng lên 2 (một lần vào + một lần ra). Chu trình Euler lại ñi qua tất cả các cạnh nên

suy ra mọi ñỉnh của ñồ thị ñều có bậc chẵn.

Ngược lại nếu liên thông và mọi ñỉnh ñều có bậc chẵn, ta sẽ chỉ ra thuật toán xây dựng

chu trình Euler trên .

Xuất phát từ một ñỉnh bất kì, ta ñi sang một ñỉnh tùy ý kề nó, ñi qua cạnh nào xoá luôn

cạnh ñó cho tới khi không ñi ñược nữa, có thể nhận thấy rằng sau mỗi bước ñi, chỉ có ñỉnh

ñầu và ñỉnh cuối của ñường ñi có bậc lẻ còn mọi ñỉnh khác trong ñồ thị ñều có bậc chẵn.

Cạnh cuối cùng ñi qua chắc chắn là ñi tới một ñỉnh bậc lẻ, vì nếu là cạnh ñi tới một ñỉnh

bậc chẵn thì ñỉnh này sẽ có ít nhất 2 cạnh liên thuộc, và như vậy khi ñi tới ñỉnh này và xoá

cạnh vào ta vẫn còn một cạnh ñể ra, quá trình ñi chưa kết thúc. ðiều này chỉ ra rằng cạnh

cuối cùng bắt buộc phải ñi về nơi xuất phát tức là chúng ta có một chu trình

. Cũng dễ

dàng nhận thấy rằng khi quá trình này kết thúc, mọi ñỉnh của vẫn có bậc chẵn.

Nếu còn lại cạnh liên thuộc với một ñỉnh nào ñó trên

thì lại bắt ñầu từ , ta ñi một

cách tùy ý theo các cạnh còn lại của ta sẽ ñược một chu trình

< bắt ñầu từ và kết thúc

A

B

C

D

A

B

C

D

203

tại . Thay thế một bước ñi qua ñỉnh trên

bằng cả chu trình

<, ta sẽ ñược một chu

trình mới lớn hơn. Quy trình ñược lặp lại cho tới khi

không còn ñỉnh nào có cạnh liên

thuộc nằm ngoài

. Do tính liên thông của , ñiều này có nghĩa là

chứa tất cả các cạnh

của hay

là chu trình Euler trên ñồ thị ban ñầu.

Hệ quả

Một ñồ thị vô hướng liên thông , có ñường ñi Euler khi và chỉ

khi nó có ñúng 2 ñỉnh bậc lẻ.

Ch

ng minh

Nếu có ñường ñi Euler thì chỉ có ñỉnh bắt ñầu và ñỉnh kết thúc ñường ñi có bậc lẻ còn

mọi ñỉnh khác ñều có bậc chẵn. Ngược lại nếu ñồ thị liên thông có ñúng 2 ñỉnh bậc lẻ thì ta

thêm vào một cạnh giả nối hai ñỉnh bậc lẻ ñó và tìm chu trình Euler. Loại bỏ cạnh giả khỏi

chu trình, chúng ta sẽ ñược ñường ñi Euler.

ðịnh lí 5-19

Một ñồ thi có hướng liên thông yếu , có chu trình Euler thì mọi

ñỉnh của nó có bán bậc ra bằng bán bậc vào: l

¨& l

¨' , 0

; Ngược lại, nếu liên thông yếu và mọi ñỉnh của nó có bán bậc ra

bằng bán bậc vào, thì có chu trình Euler (suy ra sẽ là liên thông

mạnh).

Ch

ng minh

Tương tự như phép chứng minh ðịnh lí 5.18.

Hệ quả

Một ñồ thị có hướng liên thông yếu , có ñường ñi Euler nhưng

không có chu trình Euler nếu tồn tại ñúng hai ñỉnh s, sao cho:

l

¨&

O l

¨'

l

¨' O l

¨& 1

còn tất cả những ñỉnh còn lại của ñồ thị ñều có bán bậc ra bằng bán bậc

vào.

Việc chứng minh ðịnh lí 5-18 (Euler) cho ta một thuật toán hữu hiệu ñể chỉ ra chu

trình Euler trên ñồ thị Euler. Thuật toán này hoạt ñộng dựa trên một ngăn xếp

  
4 và ñược mô tả cụ thể như sau: Bắt ñầu từ ñỉnh 1, ta ñi thoải mái theo các

cạnh của ñồ thị cho tới khi không ñi ñược nữa, ñi tới ñỉnh nào ta ñẩy ñỉnh ñó vào

ngăn xếp và ñi qua cạnh nào thì ta xoá cạnh ñó khỏi ñồ thị. Khi không ñi ñược

nữa thì ngăn xếp sẽ chứa các ñỉnh trên một chu trình

bắt ñầu và kết thúc ở ñỉnh

1. Sau ñó chúng ta lấy lần lượt các ñỉnh ra khỏi ngăn xếp tương ñương với việc ñi

ngược chu trình

. Nếu ñỉnh ñược lấy ra () không có cạnh nào còn lại liên thuộc

với nó thì sẽ ñược ghi ra chu trình Euler, ngược lại, nếu vẫn còn có cạnh liên

thuộc thì ta lại ñi tiếp từ theo cách trên và ñẩy thêm vào ngăn xếp một chu trình

204

< bắt ñầu và kết thúc tại , ñể khi lấy các ñỉnh ra khỏi ngăn xếp sẽ tương ñương

với việc ñi ngược lại chu trình

< rồi tiếp tục ñi ngược phần còn lại của chu trình

trong ngăn xếp…Có thể hình dung là thuật toán lần ngược chu trình

, khi ñến

ñỉnh thì thay bằng cả một chu trình

<…

Khi cài ñặt thuật toán, chúng ta cần trang bị ba phép toán trên ngăn xếp   
4:

• (

: ðẩy một ñỉnh vào   
4

• (\_x0010\_: Lấy ra một ñỉnh khỏi   
4

•

: ðọc phần tử ở ñỉnh   
4

Stack := (1); //Ngăn xếp ban ñầu chỉ chứa một ñỉnh bất kì,

chẳng hạn ñỉnh 1

repeat

u := Get; //ðọc phần tử ở ñỉnh ngăn xếp

if ∃(u, v) E then //Từ u còn ñi tiếp ñược

begin

Push(v);

E := E – {(u, v)}; //Xoá cạnh (u, v) khỏi ñồ thị

end;

else //Từ u không ñi ñâu ñược nữa

begin

u := Pop; //Lấy u khỏi ngăn xếp

Output ← u; //In ra u

end;

until Stack = ∅; //Lặp tới khi ngăn xếp rỗng

c) Cài ñặt

Dưới ñây chúng ta sẽ cài ñặt thuật toán tìm chu trình Euler trên ña ñồ thị Euler vô

hướng , . Dữ liệu vào luôn ñảm bảo ñồ thị liên thông, có ít nhất một

ñỉnh và mọi ñỉnh ñều có bậc chẵn.

Input

• Dòng 1 chứa số ñỉnh 3 10F

và số cạnh I 3 10

• I dòng tiếp, mỗi dòng chứa số hiệu hai ñầu mút của một cạnh.

Output

Chu trình Euler

205

Sample Input Sample Output

5 9

1 2

1 3

2 3

2 4

2 5

3 4

3 5

4 5

4 5

1 2 4 5 4 3 5 2 3 1

Ngoài các thao tác ñối với ngăn xếp, thuật toán tìm chu trình Euler còn yêu cầu

cài ñặt hai thao tác sau ñây một cách hiệu quả:

• Với mỗi ñỉnh kiểm tra xem có tồn tại cạnh liên thuộc với nó hay không, nếu

có thì chỉ ra một cạnh liên thuộc.

• Loại bỏ một cạnh khỏi ñồ thị

Các cạnh của ñồ thị ñược ñánh số từ 1 tới I, sau ñó mỗi cạnh vô hướng J, K sẽ

ñược thay thế bởi hai cung có hướng ngược chiều: J, K và K, J. Mỗi cung là

một bản ghi gồm hai ñỉnh ñầu mút và chỉ số cạnh vô hướng tương ứng.

const

maxM = 1000000;

type

TArc = record

x, y: Integer; //cung (x, y)

edge: Integer; //chỉ số cạnh vô hướng tương ứng

end;

var

a: array[1..2 \* maxM] of TArc;

Danh sách liên thuộc ñược xây dựng theo kiểu reverse star: Mỗi ñỉnh cho tương

ứng với một danh sách các cung ñi vào . Các danh sách này ñược cho bởi hai

mảng

lm1 … n và s\_x0012\_4m1 … 2In trong ñó:

•

lmn là chỉ số cung ñầu tiên trong danh sách liên thuộc các cung ñi vào ,

trường hợp ñỉnh không còn cung ñi vào,

lmn ñược gán bằng 0.

• s\_x0012\_4m\_x0012\_n là chỉ số cung kế tiếp cung /

trong cùng danh sách liên thuộc chứa

cung /

, trường hợp /

là cung cuối cùng trong một danh sách liên thuộc,

s\_x0012\_4m\_x0012\_n ñược gán bằng 0.

1

2 3

4 5

1 2

3

4

5 6

7

8

9

206

ðể thực hiện thao tác xoá cạnh, ta duy trì một mảng ñánh dấu l

s

lm1 … In

trong ñó l

s

lm\_x0012\_n True nếu cạnh vô hướng thứ \_x0012\_ ñã bị xoá. Mỗi khi cạnh vô

hướng bị xoá, cả hai cung có hướng tương ứng ñều không còn tồn tại, việc kiểm

tra một cung có hướng /

còn tồn tại hay không có thể thực hiện bằng việc kiểm

tra: l

s

lm/

.

l¨

n

?

False.

Chúng ta sẽ cài ñặt các thao tác sau trên cấu trúc dữ liệu:

• Hàm

: Trả về phần tử nằm ở ñỉnh ngăn xếp.

• Hàm Pop: Trả về phần tử nằm ở ñỉnh ngăn xếp và rút phần tử ñó khỏi ngăn

xếp.

• Thủ tục Pushv: ðẩy một ñỉnh vào ngăn xếp.

Tất cả các thao tác trên trên ngăn xếp có thể cài ñặt ñể thực hiện trong thời gian

¬1. Thuật toán tìm chu trình Euler có thể viết cụ thể hơn:

Stack := (1); //Khởi tạo ngăn xếp chỉ chứa một ñỉnh

repeat

u := Get; //ðọc ñỉnh u từ ngăn xếp

i := head[u]; //Xét cung a[i] ñứng ñầu danh sách liên thuộc

các cung ñi vào u

while (i > 0) and (deleted[a[i].edge]) do //cung a[i] ứng

với cạnh vô hướng ñã xoá

i := link[i]; //Dịch sang cung kế tiếp

head[u] := i; //Những cung ñã duyệt qua bị loại ngay, cập

nhật lại chỉ số ñầu danh sách liên thuộc

if i > 0 then //u còn cung ñi vào ứng với cạnh vô hướng

chưa xoá

begin

Push(a[i].x); //ðẩy ñỉnh nối tới u vào ngăn xếp (ñi

ngược cung a[i])

Deleted[a[i].edge] := True; //Xoá ngay cạnh vô hướng

ứng với cung a[i]

end

else

Output ← Pop;

until Top = 0; //Lặp tới khi ngăn xếp rỗng

Xét vòng lặp repeat…until, mỗi bước lặp có một thao tác (

hoặc (\_x0010\_ ñược

thực hiện. Mỗi lần thao tác (

ñược thực hiện phải có một cạnh vô hướng bị

xoá và ngăn xếp có thêm một ñỉnh. Mỗi lần thao tác (\_x0010\_ ñược thực hiện thì ngăn

xếp bị bớt ñi một ñỉnh. Vì thuật toán in ra I 1 ñỉnh trên chu trình Euler nên sẽ

phải có tổng cộng I 1 thao tác (\_x0010\_. Trước khi vào vòng lặp ngăn xếp có một

207

ñỉnh và khi vòng lặp kết thúc ngăn xếp trở thành rỗng, suy ra số thao tác (

phải là I. Từ ñó, vòng lặp repeat…until thực hiện 2I 1 lần.

Tiếp theo ta ñánh giá số thao tác duyệt danh sách liên thuộc của ñỉnh . Bởi sau

vòng lặp while có lệnh cập nhật

lmn x \_x0012\_ nên có thể thấy rằng lệnh gán

\_x0012\_ x s\_x0012\_4m\_x0012\_n ñược thực hiện bao nhiêu lần thì danh sách liên thuộc của bị giảm

ñi ñúng chừng ñó cung. Tổng số phần tử của các danh sách liên thuộc là 2I và

khi thuật toán kết thúc, các danh sách liên thuộc ñều rỗng. Suy ra tổng thời gian

thực hiện phép duyệt danh sách liên thuộc (vòng lặp while) trong toàn bộ thuật

toán là ΘI.

Suy ra thời gian thực hiện giải thuật là ΘI.

EULER.PAS Tìm chu trình Euler trong ña ñồ thị Euler vô hướng

{$MODE OBJFPC}

program EulerTour;

const

maxN = 100000;

maxM = 1000000;

type

TArc = record //Cấu trúc một cung

x, y: Integer; //Đỉnh ñầu và ñỉnh cuối

edge: Integer; //Chỉ số cạnh vô hướng tương ứng

end;

var

n, m: Integer;

a: array[1..2 \* maxM] of TArc; //Danh sách các cung

link: array[1..2 \* maxM] of Integer; //link[i]: Chỉ số cung kế tiếp a[i]

trong cùng danh sách liên thuộc

head: array[1..maxN] of Integer; //head[u]: chỉ số cung ñầu tiên trong

danh sách các cung đi vào u

deleted: array[1..maxM] of Boolean; //Đánh dấu cạnh vô hướng bị xoá

hay chưa

Stack: array[1..maxM + 1] of Integer; //Ngăn xếp

Top: Integer; //Phần tử ñỉnh ngăn xếp

procedure Enter; //Nhập dữ liệu và xây dựng danh sách liên thuộc

var

i, j, u, v: Integer;

begin

ReadLn(n, m);

j := 2 \* m;

for i := 1 to m do

208

begin

ReadLn(u, v); //Đọc một cạnh vô hướng, thêm 2 cung có hướng tương ứng

a[i].x := u; a[i].y := v; a[i].edge := i;

a[j].x := v; a[j].y := u; a[j].edge := i;

Dec(j);

end;

FillChar(head[1], n \* SizeOf(head[1]), 0); //Khởi tạo các danh

sách liên thuộc rỗng

for i := 2 \* m downto 1 do

with a[i] do //Duyệt từng cung (x, y)

begin //Đưa cung ñó vào danh sách liên thuộc các cung ñi vào y

link[i] := head[y];

head[y] := i;

end;

FillChar(deleted[1], n \* SizeOf(deleted[1]), False); //Các

cạnh vô hướng đều chưa xoá

end;

procedure FindEulerTour;

var

u, i: Integer;

begin

Top := 1; Stack[1] := 1; //Khởi tạo ngăn xếp chứa ñỉnh 1

repeat

u := Stack[Top]; //ðọc phần tử ở ñỉnh ngăn xếp

i := head[u]; //Cung a[i] ñang ñứng ñầu danh sách liên thuộc

while (i > 0) and (deleted[a[i].edge]) do

i := link[i]; //Dịch chỉ số i dọc danh sách liên thuộc ñể tìm cung ứng với

cạnh vô hướng chưa xoá

head[u] := i; //Cập nhật lại head[u], "nhảy" qua các cung ứng với cạnh vô

hướng ñã xoá

if i > 0 then //u còn cung đi vào ứng với cạnh vô hướng chưa xoá

begin

Inc(Top); Stack[Top] := a[i].x; //Đi ngược cung a[i], đẩy ñỉnh

nối tới u vào ngăn xếp

Deleted[a[i].edge] := True; //Xoá cạnh vô hướng tương ứng với a[i]

end

else //u không còn cung đi vào

begin

Write(u, ' '); //In ra u trên chu trình Euler

Dec(Top); //Lấy u khỏi ngăn xếp

end;

until Top = 0; //Lặp tới khi ngăn xếp rỗng

WriteLn;

209

end;

begin

Enter;

FindEulerTour;

end.

d) Vài nhận xét

Bằng việc quan sát hoạt ñộng của ngăn xếp, chúng ta có thể sửa mô hình cài ñặt

của thuật toán nhằm tận dụng chính ngăn xếp của chương trình con ñệ quy chứ

không cần cài ñặt cấu trúc dữ liệu ngăn xếp ñể chứa các ñỉnh:

procedure Visit(u: Integer);

var

i: Integer;

begin

i := head[u];

while i ≠ 0 do

begin //Xét cung a[i] đi vào u

if not deleted[a[i].edge] then //Cạnh vô hướng tương ứng chưa bị xoá

begin

deleted[a[i].edge] := True; //Xoá cạnh vô hướng tương ứng

Visit(a[i].x); //ði ngược chiều cung a[i] thăm ñỉnh nối tới u

end;

end;

Output ← u; //Từ u không thể ñi ngược chiều cung nào nữa, in ra u trên chu trình Euler

end;

begin

«Nhập ñồ thị và xây dựng danh sách liên thuộc»;

Visit(1); //Khởi ñộng thuật toán tìm chu trình Euler

end.

Cách cài ñặt này khá ñơn giản vì thao tác trên ngăn xếp ñược thực hiện tự nhiên

qua cơ chế gọi và thoát thủ tục ñệ quy. Tuy nhiên cần chú ý rằng ñộ sâu của dây

chuyền ñệ quy có thể lên tới I 1 cấp nên với một số công cụ lập trình cần ñặt

lại dung lượng bộ nhớ Stack1

.

Chúng ta có thể liên hệ thuật toán này với thuật toán tìm kiếm theo chiều sâu: Từ

mô hình DFS, nếu thay vì ñi thăm ñỉnh chúng ta ñi thăm cạnh (một cạnh có thể ñi

tiếp sang cạnh chung ñầu mút với nó). ðồng thời ta ñánh dấu cạnh ñã qua/chưa

1

Trong Free Pascal 32 bit, dung lượng bộ nhớ Stack dành cho biến ñịa phương và tham số chương trình con

mặc ñịnh là 64 KiB. Có thể ñặt lại bằng dẫn hướng biên dịch {$M…}

210

qua thay cho cơ chế ñánh dấu một ñỉnh ñã thăm/chưa thăm. Khi ñó thứ tự duyệt

xong (finish) của các cạnh cho ta một chu trình Euler.

Thuật toán không có gì sai nếu ta xây dựng danh sách liên thuộc kiểu forward star

thay vì kiểu reverse star. Tuy nhiên ta chọn kiểu reverse star bởi cách biểu diễn

này thích hợp ñể tìm chu trình Euler trên cả ñồ thị vô hướng và có hướng.

Người ta còn có thuật toán Fleury (1883) ñể tìm chu trình Euler bằng tay: Bắt ñầu

từ một ñỉnh, chúng ta ñi thoải mái theo các cạnh theo nguyên tắc: xoá bỏ các cạnh

ñi qua và chỉ ñi qua cầu khi không còn cách nào khác ñể chọn. Khi không thể ñi

tiếp ñược nữa thì ñường ñi tìm ñược chính là chu trình Euler.

Bằng cách “lạm dụng thuật ngữ”, ta có thể mô tả ñược thuật toán tìm Fleury cho

cả ñồ thị Euler có hướng cũng như vô hướng:

• Dưới ñây nếu ta nói cạnh , thì hiểu là cạnh , trên ñồ thị vô hướng,

hiểu là cung , trên ñồ thị có hướng.

• Ta gọi cạnh , là “một ñi không trở lại” nếu như từ ñi tới , sau ñó xoá

cạnh này ñi thì không có cách nào từ quay lại .

Thuật toán Fleury tìm chu trình Euler: Xuất phát từ một ñỉnh, ta ñi một cách tuỳ ý

theo các cạnh tuân theo hai nguyên tắc: Xoá bỏ cạnh vừa ñi qua và chỉ chọn cạnh

“một ñi không trở lại” nếu như không còn cạnh nào khác ñể chọn.

Thuật toán Fleury là một thuật toán thích hợp cho việc tìm chu trình Euler bằng

tay (với những ñồ thị vẽ ra ñược trên mặt phẳng thì việc kiểm tra cầu bằng mắt

thường là tương ñối dễ dàng). Tuy vậy khi cài ñặt thuật toán trên máy tính thì

thuật toán này tỏ ra không hiệu quả.

6.2. ðồ thị Hamilton

a) Bài toán

Khái niệm về ñường ñi và chu trình Hamilton ñược ñưa ra bởi William Rowan

Hamilton (1856) khi ông thiết kế một trò chơi trên khối ña diện 20 ñỉnh, 30 cạnh,

12 mặt, mỗi mặt là một ngũ giác ñều và người chơi cần chọn các cạnh ñể thành

lập một ñường ñi qua 5 ñỉnh cho trước (Hình 5-30).

ðồ thị , ñược gọi là ñồ thị Hamilton (Hamiltonian graph) nếu tồn tại

chu trình ñơn ñi qua tất cả các ñỉnh. Chu trình ñơn ñi qua tất cả các ñỉnh ñược gọi

là chu trình Hamilton (Hamiltonian Circuit/Hamiltonian Circle). ðể thuận tiện,

người ta quy ước rằng ñồ thị chỉ gồm 1 ñỉnh là ñồ thị Hamilton, nhưng ñồ thị gồm

2 ñỉnh liên thông không phải là ñồ thị Hamilton.

211

Hình 5-30

ðồ thị , ñược gọi là ñồ thị nửa Hamilton (traceable graph) nếu tồn tại

ñường ñi ñơn qua tất cả các ñỉnh. ðường ñi ñơn ñi qua tất cả các ñỉnh ñược gọi là

ñường ñi Hamilton (Hamiltonian Path).

Hình 5-31

Trong Hình 5-31, ðồ thị + có chu trình Hamilton ), ,   
, l,

, .. 5 không có

chu trình Hamilton nhưng có ñường ñi Hamilton ), ,   
, l.. D không có cả chu

trình Hamilton lẫn ñường ñi Hamilton.

b) Các ñịnh lí liên quan

Từ ñịnh nghĩa ta suy ra ñược ñồ thị ñường của ñồ thị Euler là một ñồ thị

Hamilton. Ngoài ra những ñịnh lí sau ñây cho chúng ta vài cách nhận biết ñồ thị

Hamilton.

ðịnh lí 5-20

ðồ thị vô hướng G, trong ñó tồn tại 4 ñỉnh sao cho nếu xoá ñi 4 ñỉnh này

cùng với những cạnh liên thuộc của chúng thì ñồ thị nhận ñược sẽ có

nhiều hơn 4 thành phần liên thông thì khẳng ñịnh là G không phải ñồ thị

Hamilton

a

b

e

c

d

a b

d c

b c

e d

a

f

+ 5 D

212

ðịnh lí 5-21 (ðịnh lí Dirak, 1952)

Xét ñơn ñồ thị vô hướng , có 7 3 ñỉnh. Nếu mọi ñỉnh ñều có

bậc không nhỏ hơn /2 thì là ñồ thị Hamilton.

ðịnh lí 5-22 (ðịnh lí Ghouila-Houiri, 1960)

Xét ñơn ñồ thị có hướng liên thông mạnh , có ñỉnh. Nếu trên

phiên bản vô hướng của , mọi ñỉnh ñều có bậc không nhỏ hơn thì là

ñồ thị Hamilton.

ðịnh lí 5-23 (ðịnh lí Ore, 1960)

Xét ñơn ñồ thị vô hướng , có 7 3 ñỉnh. Với mọi cặp ñỉnh

không kề nhau có tổng bậc 7 thì là ñồ thị Hamilton.

ðịnh lí 5-24 (ðịnh lí Meynie, 1973)

Xét ñơn ñồ thị có hướng liên thông mạnh , có ñỉnh. Nếu trên

phiên bản vô hướng của , với mọi cặp ñỉnh không kề nhau có tổng bậc

7 2 O 1 thì là ñồ thị Hamilton.

ðịnh lí 5-25 (ðịnh lí Bondy-Chvátal, 1972)

Xét ñồ thị vô hướng , có ñỉnh, với mỗi cặp ñỉnh không kề

nhau , mà l

¨ l

¨ 7 ta thêm một cạnh nối và , cứ làm

như vậy cho tới khi không thêm ñược cạnh nào nữa ta thu ñược ñồ thị mới

kí hiệu   
s. Khi ñó là ñồ thị Hamilton nếu và chỉ nếu   
s là ñồ thị

Hamilton.

Nếu ñồ thị thỏa mãn ñiều kiện của ðịnh lí 5-21 hoặc ðịnh lí 5-23thì   
s là

ñồ thị ñầy ñủ, khi ñó   
s chắc chắn có chu trình Hamilton. Như vậy ñịnh lí

Bondy-Chvátal là mở rộng của ñịnh lí Dirak và ñịnh lí Ore.

c) Cài ñặt

Mặc dù chu trình Hamilton và chu trình Euler có tính ñối ngẫu, người ta vẫn chưa

tìm ra phương pháp với ñộ phức tạp ña thức ñể tìm chu trình Hamilton cũng như

ñường ñi Hamilton trong trường hợp ñồ thị tổng quát. Tất cả các thuật toán tìm

chu trình Hamilton hiện nay ñều dựa trên mô hình duyệt, có thể kết hợp với một

số mẹo cài ñặt (heuristics).

Chúng ta sẽ lập trình tìm một chu trình Hamilton (nếu có) trên một ñơn ñồ thị vô

hướng với khuôn dạng Input/Output như sau:

Input

• Dòng 1 chứa số ñỉnh và số cạnh I của ñơn ñồ thị (2 3 3 1000)

213

• I dòng tiếp theo, mỗi dòng chứa hai số , tương ứng với một cạnh ,

của ñồ thị

Output

Một chu trình Hamilton nếu có

Sample Input Sample Output

5 8

1 2

1 3

1 4

2 3

2 4

3 4

3 5

4 5

1 2 3 5 4 1

Tìm chu trình Hamilton trên ñồ thị vô hướng

{$MODE OBJFPC}

program HamiltonCycle;

const

maxN = 1000;

var

a: array[1..maxN, 1..maxN] of Boolean; //Ma trận kề

avail: array[2..maxN] of Boolean;

x: array[1..maxN] of Integer;

Found: Boolean;

n: Integer;

procedure Enter; //Nhập dữ liệu và khởi tạo

var

m, i, u, v: Integer;

begin

FillChar(a, SizeOf(a), False);

ReadLn(n, m);

for i := 1 to m do

begin

Read(u, v);

a[u, v] := True;

a[v, u] := True;

end;

FillChar(avail, SizeOf(avail), True); //Mọi ñỉnh 2...n ñều chưa ñi

qua

Found := False; //Found = False: Chưa tìm ra nghiệm

1 3

2 4

5

214

x[1] := 1;

end;

procedure Attempt(i: Integer); //Thuật toán quay lui

var

v: Integer;

begin

for v := 2 to n do

if avail[v] and a[x[i - 1], v] then //Xét các ñỉnh v chưa ñi qua kề

với x[i - 1]

begin

x[i] := v; //Thử ñi sang v

if i = n then //Nếu ñã qua ñủ n ñỉnh, ñến ñỉnh thứ n

begin

if a[v, 1] then Found := True; //ðỉnh thứ n quay về ñược

1 thì tìm ra nghiệm

Exit; //Thoát luôn

end

else //Qua chưa ñủ n ñỉnh

begin

avail[v] := False; //ðánh dấu ñỉnh ñã qua

Attempt(i + 1); //ði tiếp

if Found then Exit; //Nếu ñã tìm ra nghiệm thì thoát ngay

avail[v] := True;

end;

end;

end;

procedure PrintResult; //In kết quả

var

i: Integer;

begin

if not Found then

WriteLn('There is no Hamilton cycle')

else

begin

for i := 1 to n do

Write(x[i], ' ');

WriteLn(1);

end;

end;

begin

Enter;

215

Attempt(2);

PrintResult;

end.

6.3. Hai bài toán nổi tiếng

a) Bài toán người ñưa thư Trung Hoa

Bài toán người ñưa thư Trung Hoa (Chinese Postman) ñược phát biểu ñầu tiên

dưới dạng tìm hành trình tối ưu cho người ñưa thư: Anh ta phải ñi qua tất cả các

quãng ñường ñể chuyển phát thư tín và mong muốn tìm hành trình ngắn nhất ñể ñi

hết các quãng ñường trong khu vực mà anh ta phụ trách. Chúng ta có thể phát biểu

trên mô hình ñồ thị như sau:

Bài toán: Cho ñồ thị , , mỗi cạnh

có ñộ dài (trọng số)

. Hãy

tìm một chu trình ñi qua tất cả các cạnh, mỗi cạnh ít nhất một lần sao cho tổng ñộ

dài các cạnh ñi qua là nhỏ nhất.

Dĩ nhiên nếu là ñồ thị Euler thì lời giải chính là chu trình Euler, nhưng nếu

không phải ñồ thị Euler thì sao?. Người ta ñã có thuật toán với ñộ phức tạp ña

thức ñể giải bài toán người ñưa thư Trung Hoa nếu là ñồ thị vô hướng hoặc có

hướng. Một trong những thuật toán ñó là kết hợp thuật toán tìm chu trình Euler

với một thuật toán tìm bộ ghép cực ñại trên ñồ thị. Tuy nhiên nếu là ñồ thị hỗn

hợp (có cả cung có hướng và cạnh vô hướng) thì bài toán người ñưa thư Trung

Hoa là bài toán NP-ñầy ñủ, trong trường hợp này, việc chỉ ra một thuật toán ña

thức cũng như việc chứng minh không tồn tại thuật toán ña thức ñể giải quyết hiện

vẫn ñang là thách thức của ngành khoa học máy tính.

Thật ñáng tiếc, sơ ñồ giao thông của hầu hết các thành phố trên thế giới ñều ở

dạng ñồ thị hỗn hợp (có cả ñường hai chiều và ñường một chiều) và như vậy chưa

thể có một thuật toán ña thức tối ưu dành cho các nhân viên bưu chính.

b) Bài toán người du lịch

Bài toán người du lịch (Travelling Salesman) ñặt ra là có thành phố và chi phí di

chuyển giữa hai thành phố bất kì trong thành phố ñó. Một người muốn ñi du

lịch qua tất cả các thành phố, mỗi thành phố ít nhất một lần và quay về thành phố

xuất phát, sao cho tổng chi phí di chuyển là nhỏ nhất có thể. Chúng ta có thể phát

biểu bài toán này trên mô hình ñồ thị như sau:

Bài toán: Cho ñồ thị , , mỗi cạnh

có ñộ dài (trọng số)

. Hãy

tìm một chu trình ñi qua tất cả các ñỉnh, mỗi ñỉnh ít nhất một lần sao cho tổng ñộ

dài các cạnh ñi qua là nhỏ nhất.

216

Thực ra yêu cầu ñi qua mỗi ñỉnh ít nhất một lần hay ñi qua mỗi ñỉnh ñúng một lần

ñều khó như nhau cả. Bài toán người du lịch là NP-ñầy ñủ, hiện tại chưa có thuật

toán ña thức ñể giải quyết, chỉ có một số thuật toán xấp xỉ hoặc phương pháp

duyệt nhánh cận mà thôi.

Bài tập

5.22. Trên mặt phẳng cho

hình chữ nhật có các

cạnh song song với các

trục toạ ñộ. Hãy chỉ ra

một chu trình:

• Chỉ ñi trên cạnh

của các hình chữ

nhật

• Trên cạnh của mỗi

hình chữ nhật,

ngoại trừ những

giao ñiểm với

cạnh của hình chữ

nhật khác có thể

qua nhiều lần,

những ñiểm còn

lại chỉ ñược qua

ñúng một lần.

5.23. Trong ñám cưới của Persée và Andromède có 2 hiệp sĩ. Mỗi hiệp sĩ có

không quá O 1 kẻ thù. Hãy giúp Cassiopé, mẹ của Andromède xếp 2

hiệp sĩ ngồi quanh một bàn tròn sao cho không có hiệp sĩ nào phải ngồi cạnh

kẻ thù của mình. Mỗi hiệp sĩ sẽ cho biết những kẻ thù của mình khi họ ñến

sân rồng.

5.24. Gray code: Một hình tròn ñược chia thành 2

hình quạt ñồng tâm. Hãy xếp tất cả các xâu nhị

phân ñộ dài vào các hình quạt, mỗi xâu vào

một hình quạt sao cho bất cứ hai xâu nào ở hai

hình quạt cạnh nhau ñều chỉ khác nhau ñúng 1

bit. Ví dụ với 3:

100 000

001

011

110 010

111

101

A B

D C

E F

H G

I J

L K

M

N

O P Q

R

A B M F G R H P N E M C Q R K L O I N J Q P O D A

217

5.25. Bài toán mã ñi tuần: Trên bàn cờ tổng quát kích

thước I W ô vuông 5 3 I, 3 1000. Một

quân mã ñang ở ô J+, K+

có thể di chuyển sang

ô J5, K5

nếu |J+ O J5

|.|K+ O K5

| 2 (Xem

hình vẽ).

Hãy tìm hành trình của quân mã từ ô xuất phát từ một ô tùy chọn, ñi qua tất

cả các ô của bàn cờ, mỗi ô ñúng 1 lần.

Ví dụ với 8

Hướng dẫn: Nếu coi các ô của bàn cờ là các ñỉnh

của ñồ thị và các cạnh là nối giữa hai ñỉnh tương

ứng với hai ô mã giao chân thì dễ thấy rằng hành

trình của quân mã cần tìm sẽ là một ñường ñi

Hamilton. Tuy vậy thuật toán duyệt thuần túy là

bất khả thi với dữ liệu lớn, bạn có thể thử cài ñặt

và ngồi xem máy tính vẫn toát mồ hôi ☺.

ðể giải quyết bài toán mã ñi tuần, có một mẹo nhỏ

ñược Warnsdorff ñưa ra cách ñây gần 2 thế kỉ (1823). Mẹo này không chỉ

áp dụng ñược vào bài toán mã ñi tuần mà còn có thể kết hợp vào thuật toán

duyệt ñể tìm ñường ñi Hamilton trên ñồ thị bất kì nếu biết chắc ñường ñi ñó

tồn tại (duyệt tham phối hợp).

Với mỗi ô J, K ta gọi bậc của ô ñó, degJ, K, là số ô kề với ô J, K chưa

ñược thăm (kề ở ñây theo nghĩa ñỉnh kề chứ không phải là ô kề cạnh). ðặt

ngẫu nhiên quân mã vào ô J, K nào ñó và cứ di chuyển quân mã sang ô kề

có bậc nhỏ nhất. Nếu ñi ñược hết bàn cờ thì xong, nếu không ta ñặt ngẫu

nhiên quân mã vào một ô xuất phát khác và làm lại.

Thuật toán này ñã ñược thử nghiệm và nhận thấy rằng việc tìm ra một bộ

I, : 5 3 I, 3 1000 ñể chương trình chạy ® 10 giây cũng là một

chuyện…bất khả thi.

15 26 39 58 17 28 37 50

40 59 16 27 38 51 18 29

25 14 47 52 57 30 49 36

46 41 60 31 48 53 56 19

13 24 45 62 1 20 35 54

42 61 10 23 32 55 2 5

9 12 63 44 7 4 21 34

64 43 8 11 22 33 6 3

218

MỤC LỤC

CHUYÊN ðỀ 1. THUẬT TOÁN VÀ PHÂN TÍCH THUẬT TOÁN ........................................ 5

1. Thuật toán ........................................................................................................ 5

2. Phân tích thuật toán .......................................................................................... 6

Bài tập ................................................................................................................ 11

CHUYÊN ðỀ 2. CÁC KIẾN THỨC CƠ BẢN ........................................................................... 13

1. Hệ ñếm ........................................................................................................... 13

2. Số nguyên tố .................................................................................................. 14

3. Ước số, bội số ................................................................................................ 17

4. Lí thuyết tập hợp ............................................................................................ 18

5. Số Fibonacci .................................................................................................. 21

6. Số Catalan ...................................................................................................... 23

7. Xử lí số nguyên lớn ........................................................................................ 24

Bài tập ................................................................................................................ 33

CHUYÊN ðỀ 3. SẮP XẾP ........................................................................................................... 39

1. Phát biểu bài toán ........................................................................................... 39

2. Các thuật toán sắp xếp thông dụng ................................................................ 40

3. Sắp xếp bằng ñếm phân phối (Distribution Counting) .................................. 43

Bài tập ................................................................................................................ 51

CHUYÊN ðỀ 4. THIẾT KẾ GIẢI THUẬT ............................................................................... 59

1. Quay lui (Backtracking) ................................................................................. 59

2. Nhánh và cận ................................................................................................. 71

3. Tham ăn (Greedy Method)............................................................................. 78

4. Chia ñể trị (Divide and Conquer) .................................................................. 88

5. Quy hoạch ñộng (Dynamic programming) .................................................... 97

Bài tập .............................................................................................................. 107

CHUYÊN ðỀ 5. CÁC THUẬT TOÁN TRÊN ðỒ THỊ ........................................................ 126

1. Các khái niệm cơ bản ................................................................................... 127

2. Biểu diễn ñồ thị ............................................................................................ 132

3. Các thuật toán tìm kiếm trên ñồ thị .............................................................. 143

4. Tính liên thông của ñồ thị ............................................................................ 158

5. Vài ứng dụng của DFS và BFS .................................................................... 182

6. ðồ thị Euler và ñồ thị Hamilton ................................................................... 201

HƯỚNG DẪN GIẢI BÀI TẬP ................................................................................................ thiếu

219

ChÞu tr¸ch nhiÖm xuÊt b¶n :

Chñ tÞch H§QT kiªm Tæng Gi¸m ®èc Ng« TrÇn ¸i

Phã Tæng Gi¸m ®èc kiªm Tæng biªn tËp nguyÔn quý thao

Tæ chøc b¶n th¶o vµ chÞu tr¸ch nhiÖm néi dung:

Phã tæng biªn tËp phan xu©n Thµnh

Gi¸m ®èc C«ng ty CP. DÞch vô XuÊt b¶n Gi¸o dôc Hµ Néi phan kÕ th¸i

Biªn tËp vµ söa b¶n in:

NguyÔn thÞ thanh xu©n

Tr×nh bµy b×a:

L¦¥NG QUèC HIÖP

ChÕ b¶n:

NguyÔn thÞ thanh xu©n

Tài liệu giáo khoa chuyên Tin –Quyển 1

M· sè : 8I746H9

In ................... b¶n, khæ 17 × 24 cm t¹i ...............................................................................

Sè in ................. ; Sè xuÊt b¶n : ...............................................

In xong vµ nép l−u chiÓu th¸ng .... n¨m 2009.