**資訊學奧林匹克競賽高級培訓教程**

**第一章 演算法概述**

1、什麼是演算法

演算法是指一組（有限個）規則，為解決一個問題而採用的方法和步驟，通俗點說：就是電腦解題的過程。在這個過程中，無論是形成解題思路還是編寫程式，都是在實施某種演算法。前者是推理實現的演算法，後者是操作實現的演算法。一般演算法應該具有以下五個重要特徵：

①、有窮性：一個演算法應包括有限的運算步驟，執行了有窮的操作後將終止運算，不能是個閉環。

②、確切性：演算法的每一步驟必須有確切的定義，讀者理解時不會產生二義性。並且，在任何條件下，演算法只有唯一的一條執行路徑，對於相同的輸入只能得出相同的輸出。如在演算法中不允許有“計算8/0”或“將7或8與x相加”之類的運算，因為前者的計算結果是什麼不清楚，而後者對於兩種可能的運算應做哪一種也不知道。

③、輸入：一個演算法有0個或多個輸入，以描述運算物件的初始情況，所謂0個輸入是指演算法本身定義了初始條件。如在5個數中找出最小的數，則有5個輸入。

④、輸出：一個演算法有一個或多個輸出，以反映對輸入資料加工後的結果，這是演算法設計的目的。它們是同輸入有著某種特定關係的量。如上述在5個數中找出最小的數，它的出輸出為最小的數。如果一個程式沒有輸出，這個程式就毫無意義了。

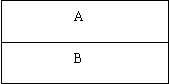
⑤、可行性：演算法中每一步運算應該是可行的。演算法原則上能夠精確地運行，而且人能用筆和紙做有限次運算後即可完成。

2、演算法的表示方法

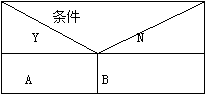
演算法通常有三種表示方法：自然語言法、程式流程圖法、程式法。

結構化程式設計三種程式結構的流程圖（N-S圖）如下：

1.順序結構

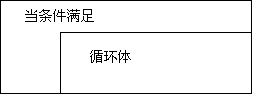


2.選擇結構

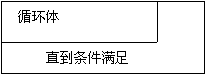


3.迴圈結構

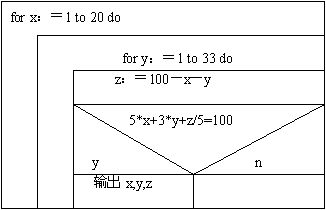
當型迴圈



直到型迴圈



【**例1-1**】 百錢買百雞問題



3、如何來評價一個演算法的好壞呢？主要是從兩個方面：

一是看演算法運行所佔用的時間；我們用時間複雜度來衡量，例如：在以下3個程式中，

（1）x:=x+1;

（2）for i:=1 to n do

x:=x+1;

（3）for i:=1 to n do

for j:=1 to n do

x:=x+1;

含基本操作“x增1”的語句x:=x+1的出現的次數分別為1，n和n2，則這三個程式段的時間複雜度分別為O（1），O（n），O（n2），分別稱為常量階、線性階和平方階。在演算法時間複雜度的表示中，還有可能出現的有：對數階o(log n)，指數階o(2n)等。在n很大時，不同數量級的時間複雜度有：**O(1)< O(log n)<O(n)< O(nlog n)<O(n2) <O(n3) <O(2n)**，很顯然，指數階的演算法不是一個好的演算法。

二是看演算法運行時所佔用的空間，既空間複雜度。由於當今電腦硬體技術發展很快，程式所能支配的自由空間一般比較充分，所以空間複雜度就不如時間複雜度那麼重要了，有許多問題人們主要是研究其演算法的時間複雜度，而很少討論它的空間耗費。

時間複雜性和空間複雜性在一定條件下是可以相互轉化的。在中學生資訊學奧賽中，對程式的執行時間作出了嚴格的限制，如果執行時間超出了限定就會判錯，因此在設計演算法時首先要考慮的是時間因素，必要時可以以犧牲空間來換取時間，動態規劃法就是一種以犧牲空間換取時間的有效演算法。對於空間因素，視題目的要求而定，一般可以不作太多的考慮。

我們通過一個簡單的數值計算問題，來比較兩個不同演算法的效率（在這裡只比較時間複雜度）。

【**例1-2**】 求n！所產生的數後面有多少個0（中間的0不計）。

【**演算法一**】　從1乘到n，每乘一個數判斷一次，若後面有0則去掉後面的0，並記下0的個數。為了不超出數的表示範圍，去掉與生成0無關的數，只保留有效位數，當乘完n次後就得到0的個數。

【參考程式】

var i,t,n,sum:longint;

begin

　t:=0; sum:=1;

readln(n);

　for i:=1 to n do

　 begin

　 sum:=sum\*i;

　 while sum mod 10=0 do

　 begin

　 sum:=sum div 10;

　 inc(t);{計數器增加1}

　 end;

　 sum:=sum mod 1000;{舍去與生成0無關的數}

　 end;

　writeln(t:6);

end.

【**演算法二**】　此題中生成0的個數只與含5的個數有關，n！的分解數中含5的個數就等於末尾0的個數，因此問題轉化為直接求n！的分解數中含5的個數。

【**參考程式**】

var t,n:integer;

begin

　readln(n);

　t:=0;

　repeat

　 n:=n div 5 ;

　 inc(t,n); {計數器增加n}

　until n<5;

　writeln(t:6);

end.

分析對比兩種演算法就不難看出，它們的時間複雜度分別為O（n）、O（log n）,演算法二的執行時間遠遠小於演算法一的執行時間。

在資訊學奧賽中，其主要任務就是設計一個有效的演算法，去求解所給出的問題。如果僅僅學會一種程式設計語言，而沒學過演算法的選手在比賽中是不會取得好的成績的，選手水準的高低在於能否設計出好的演算法。

**第二章 線性表、棧、佇列、鏈表**

2．1 線性表的定義及應用

線性表(linear list)是最常用且比較簡單的一種資料結構，它是由有限個資料元素組成的有序集合，每個資料元素由一個或多個資料項目組成。比如:26個英文字母的字母表（a，b，c，……z）是一個線性表，表中每個元素由單個字母字元組成資料項目。

線性表具有如下的結構特點：

①、均勻性：雖然不同資料表的資料元素可以是各種各樣的，但對於同一線性表的各資料元素必定具有相同的資料類型。

②、有序性：各資料元素在線性表中的位置只取決於它們的序列，資料元素之前的相對位置是線性的，即存在唯一的“第一個“和“最後一個“的資料元素，除了第一個和最後一個外，其它元素前面均只有一個資料元素（直接前趨）和後面均只有一個資料元素（直接後繼）。

在實現線性表資料元素的存儲方面，一般可用順序存儲結構和鏈式存儲結構兩種方法。本章主要介紹用陣列實現線性表資料元素的順序存儲及其應用。另外棧．佇列和串也是線性表的特殊情況，又稱為受限的線性結構。

**7**

**t**

**1**

**0**

**data**

|  |
| --- |
| **10** |
| **6** |
| **10** |
| **102** |
| **5** |
| **21** |
| **11** |

如左圖，定義：m為線性表的最大長度

type

　 list=record

　　data:array[1..m] of 資料類型;

　　　 t:0..m;

　　　 end;

var:l:list;

i:integer;

線性表的一些性質：

①、第一個結點無前序結點，最後一個結點無後序結點。

②、插入結點。l.data[i+1]:=l.data[i];l.t:=l.t+1。

③、刪除結點。 l.data[i-1]:=l.data[i];l.t:=l.t-1。

2．2 棧的定義及應用

“棧”的應用很廣泛，大家在pascal程式設計中，常遇的一種錯誤就是“棧”超界，那麼，“棧”為何物呢？

|  |
| --- |
| **4** |
| **0** |
| **8**  **t=1**  **data**  **t=0** |
| **5** |
| **9** |

以上就是一個棧。

①、棧的特點：

棧是一種線性表，對於它所有的插入和刪除都限制在表的同一端進行，這一端叫做棧的“頂”，另一端則叫做棧的“底”，其操作特點是“後進先出”或“先進後出”。棧也稱為後進先出表（lifo表）。

②、棧的一般定義：

　　type

　　　stack=record

　　　　　　　data:array[1..m] of 資料類型;

　　　　　　　t:0..m

　　　end;

　　var

　　　s:stack;

**為了方便，我們通常採用以下簡易方式定義：**

**type stack=array[1..m] of 資料類型**

**【棧的基本運算】**

①、棧的插入 push(s,x)：往棧s中推入一個值為x的項目；

　　若t=m則print(‘overflow’)

　　否則t:=t+1;data[t]:=x;

②、棧的彈出 pop(s)：從棧s中彈出一個項目；

　　若t=0則print(‘underflow’)

　　否則x:=data[t];t:=t-1;

③、讀棧頂元素top(s,x)：把棧頂元素的值讀到變數x中，

棧保持不變；

　　若t=0則print(‘error’)

　　否則x:=data[t];

④、判棧是否為空sempty(s)：這是一個布耳函數，當棧s中沒有元素(即t=0)時，稱它為空棧，函數取真值，否則值為假。

若t=0則sempty:=true

否則sempty:=false

對於出棧運算中的“下溢”，程式中僅給出了一個標誌資訊，而在實際應用中，下溢可用來作為控制程式轉移的判斷標誌，是十分有用的。對於入棧運算中的“上溢”，則是一種致命的錯誤，將使程式無法繼續運行，所以要設法避免。

【**例2-2-1**】　試設計一個進棧、出棧的pascal實現過程程式。

program zhan;

const m=10;

type stack=array[1..m] of integer;

var s:stack;

top,x,y,i:integer;

procedure push(var s:stack;var top,x:integer);

begin

if top=m then writeln('overflow')

else begin top:=top+1;s[top]:=x;end;

end;

procedure pop(var s:stack;var top,y:integer);

begin

if top=0 then writeln('underflow')

else begin y:=s[top];top:=top-1;write(y,' ');end;

end;

begin

for i:=1 to 5 do

begin

read(x);

push(s,top,x);

end;

for i:=1 to 5 do pop(s,top,y);

end.棧的用途極為廣泛，在來源程式編譯中運算式的計算、過程的嵌套調用和遞迴呼叫等都要用到棧。

【**棧的應用之一**】——計算運算式的值

①、運算式的三種形式：

中綴運算式：運算子放在兩個運算物件中間，如：(2+1)\*3；

尾碼運算式：不包含括弧，運算子放在兩個運算物件的後面，所有的計算按運算子出現的順序，嚴格從左向右進行（不再考慮運算子的優先規則，如：2 1 + 3 \*；

首碼運算式：同尾碼運算式一樣，不包含括弧，運算子放在兩個運算物件的前面，如：\* + 2 1 3。

【**演算法分析**】 將中綴運算式轉換為尾碼運算式的演算法思想

· 當讀到數位直接送至輸出佇列中

· 當讀到運算子t時

　 第一步：比較棧頂元素與t的優先順序別，如果棧頂元素的優先順序別高於或等於t，則棧頂元素彈出，送到輸出佇列中，否則不彈出。**（運算子的優先順序別定義為：“\* /”＞“+ -”＞“(”）**

第二步：t進棧

· 讀到左括弧時總是將它壓入棧中

· 讀到右括弧時，將靠近棧頂的第一個左括弧上面的運算子全部依次彈出，送至輸出佇列後，再丟棄左括弧

【**例2-2-2**】　把中綴運算式（（a+b）×c／d+10）／2 轉化成尾碼運算式？

示例：12+（3\*（20/4）-8）\*6@ → 12 3 20 4 /\* 8-6\*+

中綴運算式轉換為尾碼運算式的參考程式如下：輸入運算式以“@”為結束標誌。輸出時用空格隔開。

【**參考程式**】

program biaodashi;

const m=10;

type stack=array[1..m] of char; 定義棧

var s:stack;

i,t:integer;x,:real;str,h:string;ch,w:char;

fp:text;

procedure push; 進棧

begin

if t=m then writeln('overflow') else begin t:=t+1;s[t]:=ch;end;

end;

function pop:char; 出棧

begin

if t=0 then writeln('underflow') else begin pop:=s[t];t:=t-1;end;

end;

function top(var s:stack):char; 取棧頂元素

begin

if t=0 then writeln('underflow') else top:=s[t];

end;

function rank(ch:char):integer; 判斷運算子的優先順序別

begin

case ch of

'+','-':rank:=1;

'\*','/':rank:=2;

'(':rank:=0;

end;

end;

procedure readdata;

begin

assign(fp,'input.txt');

reset(fp);

read(fp,str);

close(fp);

end;

begin

h:='';t:=0;

readdata;

i:=1;ch:=str[i];

while ch<>'@' do

begin

case ch of

'0'..'9':h:=h+ch;

'(':push;

')':begin

w:=pop;

while w<>'(' do

begin

h:=h+' '+w;w:=pop;

end;

end;

'+','-','\*','/':

if t=0 then begin h:=h+' ';push;end

else begin

w:=top(s);

if rank(w)>=rank(ch) then

begin

w:=pop;h:=h+' '+w;

end;

push;

h:=h+' ';

end;

end;

i:=i+1;ch:=str[i];

end;

while t<>0 do h:=h+' '+pop;

writeln(h);

end.

②、運算式的計算：

由於尾碼運算式中沒有括弧，不需判別優先順序，計算嚴格從左向右進行，故計算一個尾碼運算式要比電腦一個中綴運算式簡單得多。

【**演算法分析**】 運用尾碼運算式進行計算的具體做法

· 建立一個棧s

· 從左到右讀尾碼運算式，讀到數字就直接將它壓入棧s中，讀到運算子則從棧中依次彈出兩個數分別到y和x，然後以“x 運算子 y”的形式電腦出結果，再壓加棧s中

· 如果尾碼運算式未讀完，就重複上面過程，最後輸出棧頂的數值則為結束。

由尾碼運算式計算數值，也就是計算運算式的值的全過程。

尾碼運算式求值，運算式13/25+61的尾碼運算式格式為： 13,25/61,+,@，運算式以“@”為結束標誌。

【**部分參考程式**】

function computer(a:string):real; 尾碼運算式由上面的程式得到，需定義一個實數型棧。

begin

while i<=length(a) do

begin

ch:=a[i];

if ch<>' ' then

begin

case ch of

'0'..'9':begin

x:=0;

while ch<>' 'do

begin

x:=x\*10+ord(ch)-ord('0');

i:=i+1;ch:=a[i];

end;

end;

'+': x:=popdata+popdata;

'-': begin x:=popdata;x:=popdata-x;end;

'\*': x:=popdata\*popdata;

'/': begin x:=popdata;x:=popdata/x;end;

end;

pushdata;

end;

i:=i+1;

end;

computer:=popdata;

end;

2．3 佇列的定義及應用

佇列是限定在一端進行插入，另一端進行刪除的特殊線性表。正象排列買東西，排在前面的人買完東西後離開隊伍（刪除），而後來的人總是排在隊伍未尾（插入）。通常把佇列的刪除和插入分別稱為出隊和入隊。允許出隊的一端稱為隊頭，允許入隊的一端稱為隊尾。所有需要進隊的資料項目，只能從隊尾進入，佇列中的資料項目只能從隊頭離去。由於總是先入隊的元素先出隊（先排隊的人先買完東西），這種表也稱為先進先出（fifo）表。

佇列可以用陣列q[1…ｍ]來存儲，陣列的上界ｍ即是佇列所容許的最大容量。在佇列的運算中需設兩個指針：

head：隊頭指標，指向實際隊頭元素的前一個位置

tail：隊尾指標，指向實際隊尾元素所在的位置

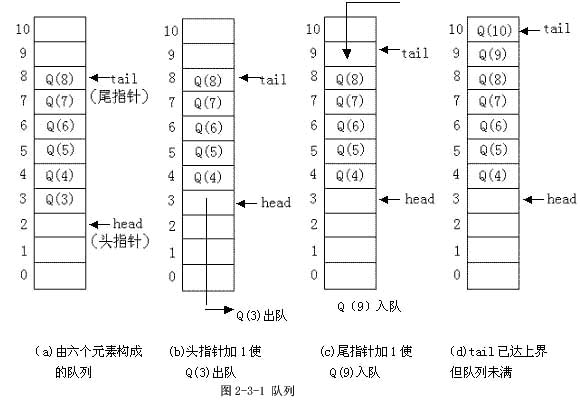
一般情況下，兩個指標的初值設為０，這時佇列為空，沒有元素。

圖2-3-1 ( a)畫出了一個由６個元素構成的佇列，陣列定義q[1…10]。q(i) i=3,4,5,6,7,8頭指針head＝2，尾指針tail＝8。佇列中擁有的元素個數為num:=tail-head

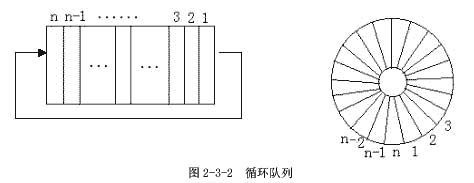
現要讓排頭的元素出隊，則需將頭指針加１。即head=head+1，這時頭指標向上移動一個位置，指向q(3)，表示q(3)已出隊。見圖2-3-1 (b)。

如果想讓一個新元素入隊，則需尾指標向上移動一個位置。即tail=tail+1，這時q(9)入隊，見圖2-3-1 (c)。

當隊尾已經處在最上面時，即tail=10，見圖2-3-1 (d)，如果還要執行入隊操作，則要發生"上溢"，但實際上佇列中還有三個空位置，所以這種溢出稱為"假溢出"。



克服假溢出的方法有兩種。一種是將佇列中的所有元素均向低位址區移動，顯然這種方法是很浪費時間的；另一種方法是將陣列存儲區看成是一個首尾相接的環形區域。當存放到n位址後，下一個位址就"翻轉"為１。在結構上採用這種技巧來存儲的佇列稱為迴圈佇列，見圖2-3-2



**迴圈隊的基本操作**

①、隊的定義

type queue=array[1..100] of integer;

var q:queue;　　　　{定義陣列}

head,tail:integer; 　{隊首、隊尾指針}

②、入隊操作

procedure addq(var q:queue;var tail:integer; n:integer; x: 資料類型);

begin

　if tail=n

　then begin tail:=0; end

　else begin tail:=tail+1; q[tail]:=x end

end;

③、出隊操作

procedure deleteq((var q:queue;var head;tail ,n:integer;var x: 資料類型);

begin

　 if head=tail

　 then begin writeln(‘queue empty!’); halt end

　 else begin head:=head+1; x:=q[head];end

end;

**【佇列應用舉例】**

【**例2-3-1**】 設有ｎ個人依次圍成一圈，從第１個人開始報數，數到第ｍ個人出列，然後從出列的下一個人開始報數，數到第ｍ個人又出列，…，如此反復到所有的人全部出列為止。設ｎ個人的編號分別為1，2，…，n，列印出出列的順序。

**【演算法分析】**

本題我們可以用陣列建立標誌位元等方法求解，但如果用上資料結構中迴圈鏈的思想，則更貼切題意，解題效率更高。

ｎ人圍成一圈，把一人看成一個結點，ｎ人之間的關係採用連結方式，即每一結點有一個前繼結點和一個後繼結點，每一個結點有一個指標指向下一個結點，最後一個結點指標指向第一個結點。這就是單迴圈鏈的資料結構。

當ｍ人出列時，將ｍ結點的前繼結點指標指向ｍ結點的後繼結點指標，即把ｍ結點驅出迴圈鏈。

①、建立迴圈鏈表。

當用陣列實現本題鏈式結構時，陣列a[i]作為"指標"變數來使用，a[i]存放下一個結點的位置。設立指標j指向當前結點，則移動結點過程為j:=a[j]，當數到m時，m結點出鏈，則a[j]:=a[a[j]]。

當直接用鏈來實現時，則比較直觀，每個結點有兩個域：一個數值域，一個指標域，當數到m時，m出鏈，將m結點的前繼結點指標指向其後繼結點；

②、設立指標，指向當前結點，設立計數器，計數數到多少人；

③、沿鏈移動指標，每移動一個結點，計數器值加１，當計數器值為ｍ時，則ｍ結點出鏈。計數器值置為１；

④、重複③、直到n個結點均出鏈為止。

**【參考程式】** 用陣列實現鏈式結構

program ex11-6a;

　const n=14;m=4;{設有10個人,報到4的人出列}

　var a:array[1..n] of integer;

　　i,j,k,p:integer;

　begin

　　for i:=1 to n-1 do a[i]:=i+1;{建立鏈表}

　　a[n]:=1;j:=n;k:=1;p:=0;{第n人指向第1人,並置初始}

　　repeat

　　　j:=a[j];k:=k+1;{報數,計數器加1}

　　　if k=m then {數到m,m人出隊,計數器置1}

　　　　begin

　　　　　write(a[j]:4);p:=p+1;a[j]:=a[a[j]];k:=1;

　　　　end

　　　until p=n;{直到n個人均出隊為止}

　end.

【**例2-3-2**】 求兩個一元多項式的和。輸入多項式方式為：多項式項數，每項係數和指數，按指數從大到小的順序輸入。

**【演算法分析】**

多項式的算數運算是表處理的一個經典問題。建立兩張表ａ、ｂ分別存放兩個多項式的內容，建立表指標ta、tb，指向表ａ和表ｂ的元素，根據表ａ、ｂ元素中的指數大小合併輸出。

1、比較ta、tb指向元素的大小，若ta的指數大於tb的指數，輸出ta元素，改變指標ta；

2、若ta的指數小於tb的指數，輸出tb元素，改變指標tb；

3、若ta的指數等於tb的指數，ta、tb元素的係數相加輸出，同時改變指針ta和tb；

4、若有一表取空，則輸出另一表剩餘的內容。

**【參考程式】** 多項式相加的順序表實現

program muti;

type

node=record

zhi,xi:integer;

end;

ar=array[1..1000] of node;

var

a,b:ar;

ta,tb,n:integer;

begin

write('one : '); readln(n);{輸入第一個多項式的係數和指數}

for ta:=n downto 1 do readln(a[ta].xi,a[ta].zhi);

ta:=n;

write('two : '); readln(n);{輸入第二個多項式的係數和指數}

for tb:=n downto 1 do readln(b[tb].xi,b[tb].zhi);

tb:=n;

write('result is ');

while (ta>0) and (tb>0) do {當兩個表均不空時}

begin {比較兩表指標指向的項指數,輸出指數小的項係數和指數, 同時改變該表指針}

if a[ta].zhi>b[tb].zhi then

begin

if a[ta].xi<0 then write(#8' '#8);write(a[ta].xi,'x',a[ta].zhi,'+');dec(ta);

end

else

if a[ta].zhi<b[tb].zhi then

begin

if b[tb].xi<0 then write(#8' '#8);write(b[tb].xi,'x',b[tb].zhi,'+');dec(tb);

end

else

begin {若兩表指標指向的項指數相等,則兩係數相加輸出, 兩表指針同時改變}

if b[tb].xi+a[ta].xi<>0 then

begin

if b[tb].xi+a[ta].xi<0 then write(#8' '#8);write(b[tb].xi+a[ta].xi,'x',b[tb].zhi,'+');

end;

dec(ta);dec(tb);

end;

end;

while ta>0 do {若有一表空,則輸出另一表的剩餘項}

begin

if a[ta].xi<0 then write(#8' '#8);write(a[ta].xi,'x',a[ta].zhi,'+');dec(ta);

end;

while tb>0 do

begin

if b[tb].xi<0 then write(#8' '#8);write(b[tb].xi,'x',b[tb].zhi,'+');dec(tb);

end;

writeln(#8' '#8);

readln;

end.

【**例2-3-3**】 一個旅行家想駕駛汽車以最少的費用從一個城市到另一個城市（假設出發時油箱是空的）。給定兩個城市之間的距離d1、汽車油箱的容量c（以升為單位）、每升汽油能行駛的距離d2、出發點每升汽油價格p和沿途油站數n（n可以為零），油站i離出發點的距離di、每升汽油價格pi（i=1，2，……n）。

計算結果四捨五入至小數點後兩位。

如果無法到達目的地，則輸出“no solution”。

【**樣例輸入**】

d1=275.6 c=11.9 d2=27.4 p=2.8 n=2

|  |  |  |
| --- | --- | --- |
| 油站號i | 離出發點的距離di | 每升汽油價格pi |
| 1 | 102.0 | 2.9 |
| 2 | 220.0 | 2.2 |

【**樣例輸出**】

26.95（該資料表示最小費用）

**【演算法分析】**

看到這道題，許多人都馬上判斷出窮舉是不可行的，因為資料都是以實數的形式給出的。但是，不用窮舉，有什麼方法是更好的呢？遞推是另一條常見的思路，但是具體方法不甚明朗。

既然沒有現成的思路可循，那麼先分析一下問題不失為一個好辦法。由於汽車是由始向終單向開的，我們最大的麻煩就是無法預知汽車以後對汽油的需求及油價變動；換句話說，前面所買的多餘的油只有開到後面才會被發覺。

提出問題是解決的開始。為了著手解決遇到的困難，取得最優方案，那就必須做到兩點，即只為用過的汽油付錢；並且只買最便宜的油。如果在以後的行程中發現先前的某些油是不必要的，或是買貴了，我們就會說：“還不如當初不買。”由這一個想法，我們可以得到某種啟示：假設我們在每個站都買了足夠多的油，然後在行程中逐步發現哪些油是不必要的，以此修改我們先前的購買計畫，節省資金；進一步說，如果把在各個站加上的油標記為不同的類別，我們只要在用時用那些最便宜的油並為它們付錢，其餘的油要麼是太貴，要麼是多餘的，在最終的計畫中會被排除。要注意的是，這裡的便宜是對於某一段路程而言的，而不是全程。

由此，我們得到如下演算法：從起點起(包括起點)，每到一個站都把油箱加滿（終點除外）；每經過兩站之間的距離，都按照從便宜到貴的順序使用油箱中的油，並計算花費，因為這是在最優方案下不得不用的油；如果當前站的油價低於油箱中仍保存的油價，則說明以前的購買是不夠明智的，其效果一定不如購買當前加油站的油，所以，明智的選擇是用本站的油代替以前購買的高價油，留待以後使用，由於我們不是真的開車，也沒有為備用的油付過錢，因而這樣的反悔是可行的；當我們開到終點時，意味著路上的費用已經得到，此時剩餘的油就沒有用了，可以忽略。

資料結構採用一個佇列：存放由便宜到貴的各種油，一個頭指標指向當前應當使用的油（最便宜的油），尾指標指向當前可能被替換的油（最貴的油）。在一路用一路補充的過程中同步修改資料，求得最優方案。

注意：每到一站都要將油加滿，以確保在有解的情況下能走完全程。並假設出發前油箱裡裝滿了比出發點貴的油，將出發點也看成一站，則程式迴圈執行換油、用油的操作，直到到達終點站為止。

本題的一個難點在於認識到油箱中油的可更換性，在這裡，突破現實生活中的思維模式顯得十分重要。

**【參考程式】**

program shortoil(input,output);

const max=1000;

type recordtype=record

　　　 price,content:real

　　　 end;

var

　 i,j,n,point,tail:longint;

　 content,change,distance2,money,use:real;

　 price,distance,consume:array[0..max] of real;

　 oil:array [0..max] of recordtype;

begin

　 write("input d1,c,d2,p:");

　 readln(distance[0],content,distance2,price[0]);

　 write("input n:");

　 readln(n);

　 distance[n+1]:=distance[0];

　 for i:=1 to n do

begin

write("input d[",i,"],","p[",i,"]:");readln(distance[i],price[i])

　 end;

　 distance[0]:=0;

　 for i:=n downto 0 do consume[i]:=(distance[i+1]-distance[i])/distance2;

　 for i:=0 to n do

　　 if consume[i]>content then　begin writeln("no solution"); halt end;

　 money:=0; tail:=1; change:=0;

　 oil[tail].price:=price[0]\*2; oil[tail].content:=content;

　 for i:=0 to n do

　 begin

　　 point:=tail;

while (point>=1) and (oil[point].price>=price[i]) do

　　 begin

　　　 change:=change+oil[point].content;

　　　 point:=point-1

　　 end;

　　 tail:=point+1;

　　 oil[tail].price:=price[i];

　　 oil[tail].content:=change;

　　 use:=consume[i]; point:=1;

　　 while (use>1e-6) and (point<=tail) do

　　　 if use>=oil[point].content

　　　 then begin

　　　　 use:=use-oil[point].content;

　　　　 money:=money+oil[point].content\*oil[point].price;

　　　　 point:=point+1 end

　　　 else begin

　　　　 oil[point].content:=oil[point].content-use;

　　　　 money:=money+use\*oil[point].price;

　　　　 use:=0

　　　 end;

　　 for j:=point to tail do oil[j-point+1]:=oil[j];

　　 tail:=tail-point+1;

　　 change:=consume[i]

　 end;

　 writeln(money:0:2)

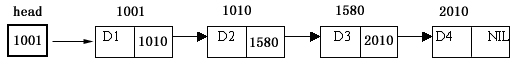
end.

2．4 鏈表的定義及應用

**（1）建立鏈表**

鏈表是動態資料結構的一種基本形式。如果我們把指標所指的一個存貯單元叫做結點，那麼鏈表就是把若干個結點鏈成了一串。我們把鏈表叫做動態資料結構，是因為鏈表中的結點可增可減，可多可少，可在中間任意一個位置插入和刪除。

下面就是一個鏈表：



每個鏈表有兩個域，一個是資料欄，一個是指向下一個結點的指標域。最後一個結點的指標域為nil，nil表示空指針。鏈表的第一個結點稱為表頭，最後一個結點稱為表尾。指向表頭的指標（head）稱為頭指標。

要定義一個鏈表，每個結點要定義成記錄型，而且其中有一個域為指標。

type

   point=↑node；

   node=record

       data:string[5]；

       next:point；

       end;

var  p1,p2:point；

在這個定義中，定義了兩個指標變數p1，p2。每個結點有兩個域，一個是資料欄，一個是指標域，資料欄是字串型。這種定義中，指標中有記錄，記錄中有指標，形成一種遞迴呼叫。

下面我們來看一下如何定義上圖的鏈表：

有：p1，p2，p3，p4屬於point類型。

①、建立第一個結點

new（p1）；

p1↑.data:=d1;

②、建立第二個結點,並把它接在p1的後面

new(p2);

p2↑.data:=d2;

p1↑.next=p2;

③、建立第三個結點,並把它接在p2的後面.

new(p3);

p3↑.data:=d3;

p2↑.next:=p3;

④、建立最後一個結點,並把它接在p3的後面.

new（p4）；

p4↑．data：＝d4；

p4↑．next：＝nil；

p3↑．next：＝p4；

一個鏈表的建立過程簡單地說分三步：

第一步：申請新結點；

第二步：給結點的資料欄和指針域賦值；

第三步：將結點連結到鏈表中的某一位置。

【**例2-4-1**】 建立一個有10個結點的鏈表，最後輸出該鏈表。

program  creattable（input，output）；

type point=^node;

node=record

data:string[5];

next:point;

end;

var

p1,p2,head:point;

i:integer;

begin

new(p1); {產生新結點p1，作為鏈表的頭}

writeln('input data');

readln(p1^.data);

head:=p1; {指標head指向鏈表頭}

for i:=1 to 9 do {用迴圈產生9個新結點，每個結點都接在上一個結點之後}

begin

new(p2);

writeln('input data');

readln(p2^.data);

p1^.next:=p2;

p1:=p2;

end;

p2^.next=nil; {給最後一個結點的link域賦空值nil}

while head^.next<>nil do {從鏈表頭開始依次輸出鏈表中的data域}

begin

write(head^.data,'->');

head:=head^.next;

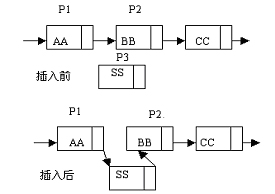
end;

end.

在本程式裡，輸出鏈表的過程就是一個鏈表的遍歷。給出一個鏈表的頭結點，依次輸出後面每一個結點的內容，指標依次向後走的語句用head：＝head↑．next  來實現。

**（2）在鏈表中插入結點**

在一個已建好的鏈表中插入一個結點，這是一種常見演算法。當我們找到插入點後，就斷開原先的連結，將新結點插入進來。



如圖所求示：要把p3的結點插入到p2之前，應該這樣操作：

①、p1，p2之間的鏈斷開

改p1指向p3。

p1↑．next：＝p3；

②、將p2，p3連接起來：

p3↑．next：＝p2；

於是，p3所指向的結點便插入進來了。

【**例2-4-2**】 插入一結點的過程

procedure insert(x:real;var head:point); {插入一結點的過程}

var

q,last,link:point;

begin

  new(q); {建立新結點}

q^.data:=x;

if x<=head^.data then {插入前表}

       begin

         q^.next:=head;

         head:=q;

       end;

   else begin   {找出表中合適的位置}

       link:=head;

       while (x>link^.data) and (link^.next<>nil) do

          begin

           last:=link;

           link:=link^.next

          end;

       if x<=link^.data then  {插入中間表}

              begin

                last^.next:=q;

                q^.next:=link

              end

        else   {插入表尾}

              begin

                link^.next:=q;

                q^.next:=nil

              end

        end

  end;

【**例2-4-3**】　建立一個有序的整數鏈表，再將一任意整數插入進鏈表，使鏈表仍然有序。

program lintable(input,output);

type point=^node;

node=record

data:integer;

next:point;

end;

var

p1,p2,p3,k:point;

begin {建立一個整數的鏈表，要求輸入的整數依次是有序的，以數9999結束}

new(p1);

writeln('input data');

readln(p1^.data);

k:=p1;

repeat

new(p2);

writeln('input data');

readln(p2^.data);

p1^.next:=p2;

p1:=p2;

until p2^.data=9999;

p2^.next:=nil;{有序鏈表建立完畢}

writeln('input p3');

readln(p3^.data);

p1:=k;p2:=k;{p1，p2都指向表頭}

while p2^.data<p3^.data do{p2找到插入點後一個結點}

p2:=p2^.next;

while p1^.next<>p2 do{p1找到插入點以前的結點}

p1:=p1^.next;

{將p3接入p1，p2之間}

p1^.next:=p3;

p3^.next:=p2;

repeat{將整個插入後的鏈表依次列印出來}

write(k^.data,'->');

k:=k^.next;

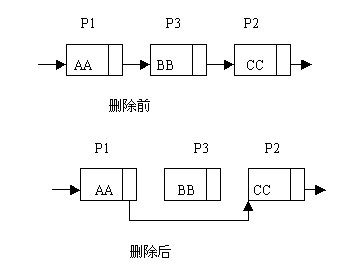
until k^.data=9999;

write(k^.data);

end.

**（3）刪除一個結點**

要在一個鏈表中刪去一個結點，首先在鏈表中找到該結點，將其前面的指針與該點斷開，直接接到後面的結點，再用dispose  命令將p3結點空間釋放。如圖，刪除p3結點：



刪除語句如下：

p1↑．link：＝p3↑．link

dispose（p3）；

【**例2-4-4**】　刪除結點的過程

procedure delete(x:real;var head;point;var deleted:boolean);{刪除結點的過程}

var

  last,link:point;

begin {遍歷表直到找到目標或到達表末}

link:=head;

  while(link^.data<>x)and(link^.next<>dil) do

   begin

    last:=link;

    link:link^.next

   end;

if link^.data=x then {如果目的檔案找到，刪除包含它的結點，設置deleted}

begin

   deleted:=true;

   if link=head then

   head:=head^.next

   else last^.link:=link^.next

  end

else deleted:=false

end;

**第三章 樹 森林**

3．1 樹及森林的定義

樹（tree）是一種非線性資料結構，用它能很好地描述有分支和層次特性的資料集合。在樹型結構中，二叉樹是最常用的結構，一般樹型結構常常轉換成二叉樹進行處理。

**（1）樹的定義**

樹是由n(n>=0)元素組成的有限集合以及在該集合上定義的一種關係構成的。集合中的每個元素稱為樹的**結點**，所定義的關係稱為**父子關係**。父子關係在樹的結點之間建立了一個層次結構。在這種層次結構中有一個結點具有特殊的地位，這個結點稱為該樹的根結點，或簡稱為**樹根**。我們可以形式地給出樹的遞迴定義如下:

①、單個結點是一棵樹，樹根就是該結點本身。

②、設t1,t2,..,tk是樹，它們的根結點分別為n1,n2,..,nk。用一個新結點n作為n1,n2,..,nk的父親，則得到一棵新樹，結點n就是新樹的根。我們稱n1,n2,..,nk為一組兄弟結點，它們都是結點n的兒子結點。我們還稱n1,n2,..,nk為結點n的子樹。

空集合也是樹，稱為**空樹**。空樹中沒有結點。

一棵典型的樹如圖3-1-1所示：

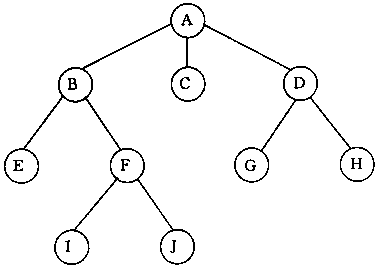


圖3-1-1 樹的層次結構

由圖3-1-1可以看出樹的形狀就像一棵現實中的樹，只不過是倒過來的。

**（2）樹的相關術語**

①、一個結點的子樹的個數稱為該結點的**度**。一棵**樹的度（寬度）**是指該樹中結點的最大度數。

②、樹中度為零的結點稱為**葉結點或終端結點。**

③、樹中度不為零的結點稱為**分枝結點或非終端結點**。除根結點外的分枝結點統稱為**內部結點**。

例如在圖3-1-1中，結點A，B和E的度分別為3，2，0。其中A為根結點,B為內部結點，E為葉結點，樹的度為3。

④、對於一棵子樹中的任意兩個不同的結點，如果從一個結點出發，按層次自上而且下沿著一個個樹枝能到達另一結點，稱它們之間存在著一條**路徑**。可用路徑所經過的結點序列表示路徑，路徑的長度等於路徑上的結點個數減1。

例如，在圖3-1-1中，結點A到結點I有一條路徑ABFI，它的長度為3。

⑤、對兩個用樹枝連接的相關聯的結點，稱上端的結點為下端結點的**父結點**，稱下端的結點為上端結點的**子結點**，同一父結點的多個子結點稱為**兄弟結點**。從根結點到某個子結點所經過的所有結點稱為這個子結點的**祖先**，以某個結點為根的子樹中的任一結點都是該結點的**子孫**。

例如在圖3-1-1中，結點F的祖先有A，B，而它的子孫有I,J，E和F是兄弟。

⑥、定義一棵樹的根結點的層次為1，其它結點的層次等於它的父結點的層次數加1。一棵樹中所有結點的層次的最大值稱為樹的**深度（高度）**。

例如，在圖3-1-1中，這棵樹的深度為4。

⑦、樹的定義在某些結點之間確定了父子關係，我們又將這種關係延拓為**祖先子孫關係**。但是樹中的許多結點之間仍然沒有這種關係。例如兄弟結點之間就沒有祖先子孫關係。如果我們在樹的每一組兄弟結點之間定義一個從左到右的次序，則得到一棵**有序樹**；否則稱為**無序樹**。設結點n的所有兒子按其從左到右的次序排列為n1,n2,..,nk，則我們稱n1是n的**最左兒子**，或簡稱**左兒子**，並稱ni是ni-1的**右鄰兄弟**，或簡稱**右兄弟**(i=2,3,..k)。

圖3-1-2中的兩棵樹作為無序樹是相同的，但作為有序樹是不同的，因為結點a的兩個兒子在兩棵樹中的左右次序是不同的。後面，我們只關心有序樹，因為無序樹總可能轉化為有序樹加以研究。

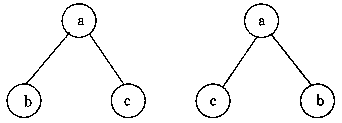


圖3-1-2 兩棵不同的有序樹

⑧、**森林**是m(m>0)棵互不相交的樹的集合。如果我們刪去一棵樹的樹根，留下的子樹就構成了一個森林。

**（3）樹的遍歷**

樹的遍歷是樹的一種重要的運算。所謂遍歷是指對樹中所有結點的系統的訪問，即依次對樹中每個結點訪問一次且僅訪問一次。樹的3種最重要的遍歷方式分別稱為前序遍歷、中序遍歷和後序遍歷。

樹的這3種遍歷方式可遞迴地定義如下：

如果t是一棵空樹，那麼對t進行前序遍歷、中序遍歷和後序遍歷都是空操作，得到的列表為空表。

如果t是一棵單結點樹，那麼對t進行前序遍歷、中序遍歷和後序遍歷都只訪問這個結點。這個結點本身就是要得到的相應列表。

否則，設t如圖3-1-3所示，它以n為樹根，樹根的子樹從左到右依次為t1,t2,..,tk，那麼有：

①、對t進行前序遍歷是先訪問樹根n,然後依次前序遍歷t1,t2,..,tk。

②、對t進行中序遍歷是先中序遍歷t1，然後訪問樹根n，接著依次對t2,t2,..,tk進行中序遍歷。

③、對t進行後序遍歷是先依次對t1,t2,..,tk進行後序遍歷，最後訪問樹根n。

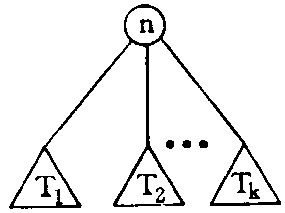


圖3-1-3 樹t

為了將一棵樹中所有結點按某種次序列表，只須對樹根調用相應過程。例如對圖3-1-4中的樹進行前序遍歷、中序遍歷和後序遍歷將分別得到前序列表：A B E F I J C D G H；中序列表：E B I F J A C G D H；後序列表：E I J F B C G H D A。

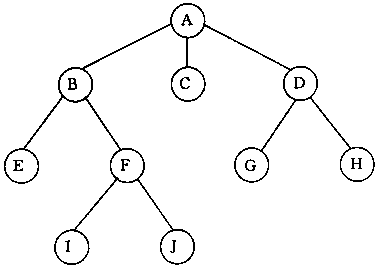


圖3-1-4 一棵樹

3．2 二叉樹及其遍歷

**二叉樹**是一類非常重要的特殊的樹形結構，它的特點是每個結點至多只有二棵子樹，即二叉樹中不存在度大於2的結點，而且二叉樹的子樹有左子樹、右子樹之分，孩子有左孩子、右孩子之分，其次序不能顛倒，所以二叉樹是一棵有序樹。 二叉樹的根可以有空的左子樹或空的右子樹，或者左、右子樹均為空。

二叉樹有5種基本形態，如圖3-2-1所示。

A

A

B

A

B

A

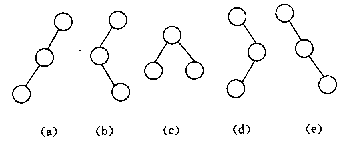
B

C

圖3-2-1 二叉樹的5種基本形態(其中□表示空)

在二叉樹中，每個結點至多有兩個兒子，並且有左、右之分。因此任一結點的兒子不外4種情況：沒有兒子；只有一個左兒子；只有一個右兒子；有一個左兒子並且有一個右兒子。

含有3個結點的不同二叉樹如下：



**（1）二叉樹的數學性質**

二叉樹具有以下的重要性質：

①、在二叉樹中，第i層的結點總數不超過2 (i-1)；

②、對於任意一棵二叉樹，如果其葉結點數為N0，而度數為2的結點總數為N2，則N0=N2+1；

③、深度為h的二叉樹最多有2 h-1個結點(h>=1)，最少有h個結點；

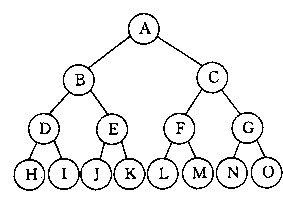
④、具有n個結點的完全二叉樹的深度為 img8log2nimg9+1

## （2）特殊形態的二叉樹

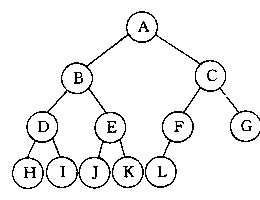
**滿二叉樹**和**完全二叉樹**是二叉樹的兩種特殊情形。

一棵高度為h≥1且有2h-1個結點的二叉樹稱為**滿二叉樹**。

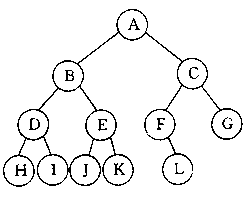
若一棵二叉樹至多只有最下面的兩層結點的度數小於2，並且最下面一層結點都集中在該層的最左邊，則稱這種二叉樹為**完全二叉樹**。



(a) 滿二叉樹



(b) 完全二叉樹



(c) 非完全二叉樹

圖3-2-2 特殊形態的二叉樹

例如圖3-2-2(a)是一棵高度為3的滿二叉樹。滿二叉樹的特點是每一層上的結點數都達到最大值，即對給定的高度，它是具有最多結點數的二叉樹。滿二叉樹中不存在度數為1的結點，每個分枝結點均有兩棵高度相同的子樹，且葉結點都在最下面一層上。圖7(b)是一棵近似滿二叉樹。顯然滿二叉樹是近似滿二叉樹，但近似滿二叉樹不一定是滿二叉樹。在滿二叉樹的最下層上，從最右結點開始連續往左刪去若干個結點後得到的二叉樹是一棵近似滿二叉樹。因此，在近似滿二叉樹中，若某個結點沒有左兒子，則它一定沒有右兒子，即該結點是一個葉結點。圖7(c)中，結點f沒有左兒子而有右兒子l，故它不是一棵近似滿二叉樹。

**（3）二叉樹的存儲結構**

**順序存儲結構**

此結構是將二叉樹的所有結點，按照一定的次序，存儲到一片連續的存儲單元中。因此，必須將結點排成一個適當的線性序列，使得結點在這個序列中的相應位置能反映出結點之間的邏輯關係。這種結構特別適用於[完全二叉樹](http://www.zsqz.com/jsbase/datastructure/basic/binary_tree/chapter3.htm)。

在一棵具有n個結點的近似滿二叉樹中，我們從樹根起，自上層到下層，逐層從左到右給所有結點編號，就能得到一個足以反映整個二叉樹結構的線性序列，如圖3-2-3所示。其中每個結點的編號就作為結點的名稱。

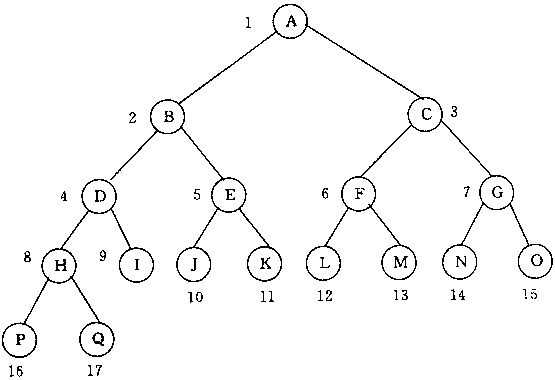


圖3-2-3 近似滿二叉樹的結點編號

因此，我們可以對樹的類型作如下說明：

數據值：data

父結點編號：parent

左孩子結點編號：lchild

右孩子結點編號：rchild

用pascal語言定義如下：

const m=樹中結點數的上限；

type

node=record

data::資料類型

parent,lchild,rchild:0..m;

end;

treetype=array[1..m] of node;

var

tree:treetype

將陣列下標作為結點名稱(編號)，就可將二叉樹中所有結點的標號存儲在一維陣列中。例如，圖3-2-3中的二叉樹的順序存儲結構如圖3-2-4所示。

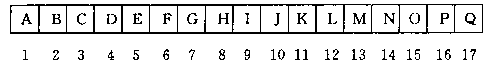


圖3-2-4 近似滿二叉樹的順序存儲結構

在二叉樹的這種表示方式下，各結點之間的邏輯關係是隱含表示的。完全二叉樹中，除最下面一層外，各層都充滿了結點。可能除最底層外，每一層的結點個數恰好是上一層結點個數的2倍。因此，從一個結點的編號就可推知其父親，左、右兒子，和兄弟等結點的編號。例如，對於結點i我們有：

①、僅當i=1時，結點i為根結點；

②、當i>1時，結點i的父結點為img8i/2img9；

③、結點i的左兒子結點為2i；

④、結點i的右兒子結點為2i+1；

⑤、當i為奇數且不為1時，結點i的左兄弟結點為i-1；

⑥、當i為偶數時，結點i的右兄弟結點為i+1。

由上述關係可知，近似滿二叉樹中結點的層次關係足以反映結點之間的邏輯關係。因此，對近似滿二叉樹而言，順序存儲結構既簡單又節省存儲空間。

對於一般的二叉樹，採用順序存儲時，為了能用結點在陣列中的位置來表示結點之間的邏輯關係，也必須按近似滿二叉樹的形式來存儲樹中的結點。顯然，這將造成存儲空間的浪費。在最壞情況下，一個只有k個結點的右單枝樹卻需要2 k-1個結點的存儲空間。例如，只有3個結點的右單枝樹，如圖3-2-5(a)所示，添上一些實際不存在的虛結點後，成為一棵近似滿二叉樹，相應的順序存儲結構如圖3-2-5(b)所示。

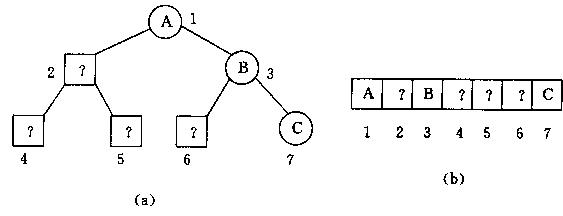


圖3-2-5 一般二叉樹的順序存儲結構

**二叉樹的鏈式存儲結構**

對於二叉樹的每個結點需要保存三種資訊：資料元素、左子樹和右子樹，故可以增設兩個分別左、右兒子結點的指標來表達樹的結構。因此，在二叉樹對應的二重鏈表中，每個結點包含三個域：資料欄、左指針域和右指針域，如下圖：

左指針域 資料欄 右指針域

|  |  |  |
| --- | --- | --- |
| lchild | data | rchild |

其中左指標域lchild、右指標域rchild分別指向結點的左、右兒子。

這種鏈表稱為二叉鏈表。二叉表的表頭指標bt指向根結點。

例：如圖3-2-6所示的二叉樹，

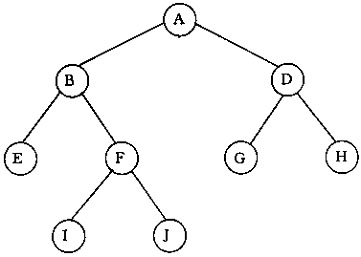


圖3-2-6

採用二叉鏈表進行存儲，如圖3-2-7

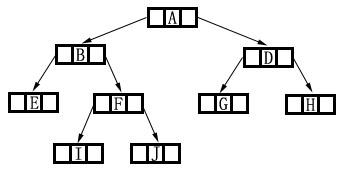


圖3-2-7

二叉樹中結點的定義如下：

type

tree=^node;

node=record

data:資料類型;

lchild,rchild:tree;

end;

var bt:tree;

**（4）二叉樹的遍歷**

在二叉樹的應用中，常常要求在樹中查找具有某種特徵的結點，或者對全部結點逐一進行某種處理，這就是二叉樹的遍歷問題。所謂二叉樹的遍歷是指按一定的規律和次序訪問樹中的各個結點，而且每個結點訪問一次且僅訪問一次。

二叉樹的遍歷運算（遞迴定義）

1. 前序遍歷（DLR）

訪問根；按先序遍歷左子樹；按先序遍歷右子樹

1. 中序遍歷（LDR）

按中序遍歷左子樹；訪問根；按中序遍歷右子樹

1. 後序遍歷（LRD）

按後序遍歷左子樹；按後序遍歷右子樹；訪問根

問題：對如圖3-2-8所示的二叉樹寫出其前序、中序、後序遍歷結果，並考慮它們各屬於什麼運算式？

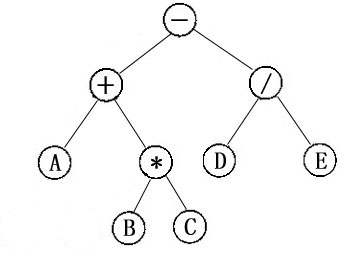


圖3-2-8

前序遍歷結果：– + a \* b c / d e 又稱首碼運算式

中序遍歷結果：a + b \* c – d / e 又稱中綴表述式

後序遍歷結果：a b c \* + d e / – 又稱尾碼運算式

下面給出中序遍歷的程式過程

procedure inorder(bt:btree);{根結點為bt的二叉樹的遞迴演算法}

begin

if bt<>nil then

begin

inorder(bt^.lchild);

write(bt^.data);

inorder(bt^.rchild);

end;

end;

同理可寫出先序遍歷和後序遍歷的過程。

如何建立一棵如上圖所示的二叉樹？

將上面二叉樹以前序遍歷的結果作為字串輸入來建立這棵樹。並在每個葉子結點後添加兩空格字元來表示，每個結點有空子樹時則須輸入空格。如：- + a ￠ ￠ \* b ￠ ￠ c ￠ ￠ / d ￠ ￠ e ￠ ￠，其中￠表示空格。下麵是實現程式。如何判斷結點是葉子結點？

Program buildtree(input,output);

type tree=^node;

node=record

data:char;

lchild,rchild:tree;

end;

var root:tree;

ch:char;

Procedure inorder(bt:tree); {中序遍歷二叉樹}

begin

if bt<>nil then

begin

inorder(bt^.lchild);

write(bt^.data);

inorder(bt^.rchild);

end;

end;

procedure createtree(var t:tree);

begin

read(ch);

if ch=' ' then t:=nil

else begin

new(t);t^.data:=ch;

createtree(t^.lchild);

createtree(t^.rchild);

end;

end;

Begin{main}

writeln;

write('Create Tree--preorder:');

createtree(root);

writeln;

write('Print Tree--inorder:');

inorder(root)

End.

**（5）二叉樹的其他重要操作**

①、求二叉樹的樹高

procedure treedepth(root:tree;h:integer;var depth:integer);

begin

if root<>nil then

begin

if h>depth then depth:=h;

treedepth(root^.lchild,h+1,depth);

treedepth(root^.rchild,h+1,depth);

end;

end;

②、由遍歷結果確定一棵二叉樹

【**基本思路**】　用前、中序兩個遍歷結果可以唯一地確定一棵二叉樹。下面用前序序列和中序序列確定一棵二叉樹。（注意：用前、後序遍歷卻不能唯一地確定一棵二叉樹。為什麼？）

【**演算法思想**】　首先用前序序列的第一個結點作為根結點；然後在中序序列中查找根結點位置，並以此為界，將中序序列劃分為左右兩個子序列，它們分別是左右子樹的中序序列；接著，根據左右子樹的中序序列中的結點個數，將前序序列去掉根結點後的序列，再劃分為左右兩個子序列，它們也分別是左右子樹的前序序列；最後，對左右子樹的前序序列和中序序列遞迴地實現同樣的方法，直至所的左右子樹為空時終止。

【**例題3-2-1**】　某一棵二叉樹的前序序列為**abcdefgh**，中序序列為**bdceaghf**，試求其後序序列，並畫出其二叉樹的結構圖。前序遍歷和中序遍歷序列分別由pre[1..n]和ino[1..n]存儲。根據前、中序序列確定二叉樹的遞迴演算法：

【**參考程式**】

program buildtree(input,output);

type tree=^node;

node=record

data:char;

lchild,rchild:tree;

end;

var root:tree;

ch:char;

pre,ino:string[255];

l1,h1,l2,h2:integer; {l1,h1,l2,h2分別表示前序與中序序列的陣列的上界和下界}

procedure post(bt:tree); {後序遍歷過程}

begin

if bt<>nil then

begin

post(bt^.lchild);

post(bt^.rchild);

write(bt^.data);

end;

end;

procedure copying(l1,h1,l2,h2:integer;var bt:tree); {建樹}

var j:integer;

begin

if l2=h2 then begin

new(bt);bt^.data:=ino[l2];

bt^.lchild:=nil;bt^.rchild:=nil;end

else

begin

j:=pos(pre[l1],ino);

new(bt);bt^.data:=pre[l1];

if j>l2 then copying(l1+1,l1+j-l2,l2,j-1,bt^.lchild); {j-l2是指ino的子字串的長度}

if j<h2 then copying(l1+j-l2+1,h1,j+1,h2,bt^.rchild);

end;

end;

begin

readln(pre);read(ino);l1:=1;l2:=1;h1:=length(pre);h2:=length(ino);

copying(l1,h1,l2,h2,root);post(root);

end.

③、按層次遍歷二叉樹

【**演算法思想**】　根結點為第一層，左右兒子作為下一層，並借用“先進先出”的佇列作為輔助手段，從而得到按層次遍歷的序列。

【**演算法描述**】

procedure breadthways(t);

begin {cq[0..m-1]是迴圈佇列}

head:=0;tail:=1;cq[tail];=t; {入隊}

while (head<>tail) do

begin

head:=(head+1) mod m;

p;=cq[head];write(p^.data); {出隊}

if p^.lchild<>nil then {左子樹入隊}

begin

tail:=(tail+1) mod m;

cq[tail]:=p^lchild;

end;

if p^.rchild<>nil then {右子樹入隊}

begin

tail:=(tail+1) mod m;

cq[tail]:=p^rchild;

end;

end;

end;

3．3 二叉排序樹

**二叉排序樹的性質**

①、若它的左子樹不空，則左子樹上所有結點的值均小於它的根結點的值；

②、若它的右子樹不空，則右子樹上所有結點的值均大於它的根結點的值；

③、它的左、右子樹也分別為二叉樹排序樹。

如圖3-3-1所示是一棵二叉排序樹

從上述二叉排序樹的性質可看出，當採用中序遍歷就可以生成一個有序的序列。該二叉排序樹的中序遍歷結果為：10、20、25、30、40、50、60、70、80

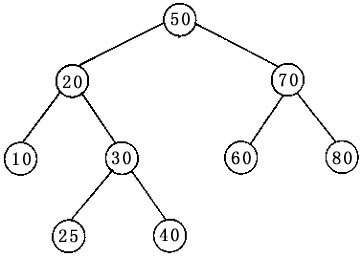


圖3-3-1

下面介紹如何生成這樣的一棵二叉排序樹？

【**問題分析**】首先生成一個結點，再根據大小決定這個結點是插在左子樹還是右子樹，如此重複直到結束。

假設資料登錄以前序遍歷的結果作為字串輸入，以0為資料序列結束標誌，

50 20 10 30 25 4070 60 80 0，下麵是建立圖3-3-1所示二叉排序樹的參考程式

【**參考程式**】

program b-ordertree(input,output);

type tree=^node;

node=record

data:integer;

lchild,rchild:tree;

end;

var bt:tree;

procedure insert(var t:tree;x:integer);{ 二叉排序樹的插入}

begin

new(s);s^.data:=x;s^.lchild:=nil;s^.rchild:=nil;

if t=nil then t:=s

else if s^.data<t^.data then insert(t^.lchild,x)

else if s^.data>t^.data then insert(t^.rchild,x);

end;

procedure inorder(bt:tree);{ 二叉排序樹的中序遍歷輸出}

begin

if bt<>nil then

begin

inorder(bt^.lchild); write(bt^.data); inorder(bt^.rchild);

end;

end;

begin

bt:=nil;

repeat

read(n);

if n<>0 then insert(bt,n);

until n=0;

inorder(bt);

end.

3．4 哈夫曼樹（huffman tree）

**哈夫曼樹**，又稱最優二叉樹，是一類帶權路徑長度最短的樹。

**路徑長度**：從樹中一個結點到另一個結點之間的分支構成這兩個結點之間的路徑，路徑上的分支數目稱做路徑長度。

例如：在如圖3-4-1中的樹中，結點A到結點E之間的路徑長度為2，結點A到結點J之間的路徑長度為3。

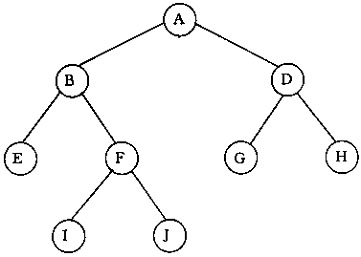


圖3-4-1

**樹的路徑長度**：從樹根到每一結點的路徑長度之和。

如圖3-4-1所示樹中根結點到每個結點的路徑長度如下表。

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 結點 | A | B | D | E | F | G | H | I | J |
| 路徑長度 | 0 | 1 | 1 | 2 | 2 | 2 | 2 | 3 | 3 |

樹的路徑長度為：1+1+2+2+2+2+3+3=16。

在幾個結點數相同的二叉樹中，完全二叉樹的路徑長度最短。

若將上述概念推廣到一般情況，考慮帶權的結點。結點的帶權路徑長度為從該結點到樹根之間的路徑長度與結點上權的乘積。樹的帶權路徑長度為樹中所有葉結點的帶權路徑長度之和。用ki表示葉結點，li表示ki到根結點的路徑長度，wi表示ki的權，那麼二叉樹的帶權路徑長度（wpl）有公式：

wpl=∑ wi × li (1≤ i ≤n)

當wi =1時，顯然，二叉樹的帶權路徑長度就是二叉樹的路徑長度。如果wi ≠1時，帶權路徑長度最小的是否還是完全二叉樹呢？如圖3-4-2所示三種情況，它們的帶權路徑長為分別為：

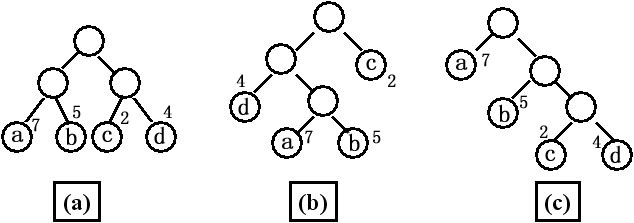


圖3-4-2

(a):wpl=7×2+5×2+2×2+4×2=36

(b): wpl =7×3+5×3+4×2+2×1=46

(c): wpl =7×1+5×2+4×3+2×3=35

由此可見，樹的帶權路徑最小的是(c)，而不是(a)。使二叉樹帶權路徑長度取最小值的樹稱為哈夫曼樹或最優二叉樹。哈夫曼樹最主要的特點是，葉子的權越大就越靠近根結點，權越小就越遠離根結點，因此哈夫曼樹又稱為最優葉子二叉樹。

如何構造哈夫曼樹呢？

設有n個結點k1，k2，…，kn，權值分別為w1，w2，…，wn，以這n個結點為葉結點構造最優二叉樹，演算法如下：

①、以n個帶權的結點作為根結點構造n棵二叉樹的集合f={t1，t2，…，tn}，其中每棵二叉樹中均只含一個帶權值為wi的根結點，其左、右子樹為空樹；

②、在f中選取其根結點的權值為最小的兩棵二叉樹，分別作為左、右子樹構造一棵新的二叉樹，並置這棵新的二叉樹根結點的權值為其左、右子樹根結點的權值之和；

③、從f中刪去這兩棵樹，同時加入生成的新樹；

④、重複②和③兩步，直到f中只含一棵樹為止。

從上述演算法中可以看出，f實際上是森林，演算法的目的是不斷地對森林中的二叉樹進行“合併”，最終得到哈夫曼樹。

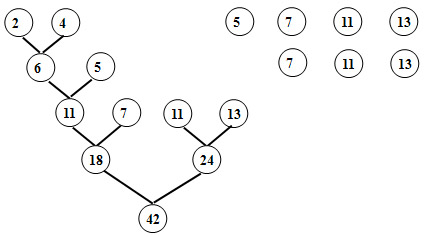
如何用程式實現哈夫曼演算法呢？這與實際問題所採用的存儲結構有關，現假設用陣列f來存儲哈夫曼二叉樹，其中第i個陣列元素f[i]是哈夫曼二叉樹中的一個結點，其位址為i，有3個域，Data域存放該結點的權值，lchild域和rchild域分別存放該結點左、右子樹的根結點的地址（下標）。在初始狀態下：

f[i].data:=wi，f[i].lchild:=0;f[i].rchild:=0;（i=1,2,…,n）

即先構造好n個葉子結點，以後每步構造一棵新的二叉樹時，都對森林中所有二叉樹的根結點進行排序，因此可用陣列a作為排序暫存空間，其中第i個陣列元素a[i]是森林中第i棵二叉樹的根結點，有2個域，data是根結點所對應的權值，addr是根結點在f中的位址（下標），在初始狀態下：a[i].data:=wi；a[i].addr:=i；（i=1,2,…,n）

【**例3-4-1**】　已知結點：未标题-2 拷贝，試求其哈夫曼樹？（圓圈內的數字為該結點的權值）

依據上述演算法，該樹的建立過程如下：



wpl=2×4+4×4+5×3+7×2+11×2+13×2=101

注：6+11+18+24+42=101，建樹過程中所得根的總和也等於wpl

哈夫曼樹具有不唯一性,可能出現在：

①．構造新樹時，左、右孩子未作規定。

②．當有多個權值相同的樹，可作有候選樹時，選擇誰未作規定。

【**參考程式**】

program huffmantree;

const m=樹中結點數的上限；

type

node=record

data::資料類型

addr,lchild,rchild:0..m;

end;

treetype=array[1..m] of node;

var

f,a:treetype

i,t,temp,tempL,tempR:integer;

procedure insort(a,n);

begin

for i:= 1 to n-1 do

for j:= i+1 to n do

if a[i].data<a[j].data then

begin

temp:=a[j].data;a[j].data:=a[i].data;a[i].data:=temp;

tempL:= a[j].lchild;a[j].lchild:=a[i].lchild;a[i].lchild:=tempL

tempR:= a[j].rchild;a[j].rchild:=a[i].rchild;a[i].rchild:=tempR

end;

end;

procedure createhuffmantree(f,t,a,n); {已知n個權值a[i]，構造哈夫曼樹f，且根結點地址為t}

var i:integer;

begin

for i:=1 to n do {初始化}

begin

f[i].data:=a[i].data; f[i].lchild:=0;f[i].rchild:=0;a[i].addr:=i;

end;

t:=n+1; {t指向下一個可利用單元}

i:=n; {當前森林中的二叉樹是i}

while i>=2 do

begin

insort(a,i); {定義一個排序函數insort，對a的前i個元素按data域進行排序}

f[t].data:=a[1].data+a[2].data; {生成新的二叉樹}

f[t].lchild:=a[1].addr;f[t].rchild:=a[2].addr;

a[1].data:=f[t].data; {修改森林}

a[1].addr:=t;

a[2].data:=a[i].data; {修改森林}

a[2].addr:=a[i].addr; {節省空間}

i:=i-1; {二叉樹數目減1}

t:=t+1;

end;

begin

createhuffmantree(f,t,a,n);

end.

【**例3-4-2**】 **合併果子**（提高組2004屆複賽題）

【問題描述】

    在一個果園裡，多多已經將所有的果子打了下來，而且按果子的不同種類分成了不同的堆。多多決定把所有的果子合成一堆。

    每一次合併，多多可以把兩堆果子合併到一起，消耗的體力等於兩堆果子的重量之和。可以看出，所有的果子經過n-1次合併之後，就只剩下一堆了。多多在合併果子時總共消耗的體力等於每次合併所耗體力之和。

    因為還要花大力氣把這些果子搬回家，所以多多在合併果子時要盡可能地節省體力。假定每個果子重量都為1，並且已知果子的種類數和每種果子的數目，你的任務是設計出合併的次序方案，使多多耗費的體力最少，並輸出這個最小的體力耗費值。

    例如有3種果子，數目依次為1，2，9。可以先將1、2堆合併，新堆數目為3，耗費體力為3。接著，將新堆與原先的第三堆合併，又得到新的堆，數目為12，耗費體力為12。所以多多總共耗費體力=3+12=15。可以證明15為最小的體力耗費值。

【輸入檔】

    輸入檔fruit.in包括兩行，第一行是一個整數n(1<＝n<=10000)，表示果子的種類數。第二行包含n個整數，用空格分隔，第i個整數ai(1<＝ai<=20000)是第i種果子的數目。

【輸出檔】

    輸出檔fruit.out包括一行，這一行只包含一個整數，也就是最小的體力耗費值。輸入資料保證這個值小於231。

樣例輸入

3

1 2 9

【樣例輸出】

15

【資料規模】

對於30％的資料，保證有n<=1000：

對於50％的資料，保證有n<=5000；

對於全部的資料，保證有n<=10000。

【分析】

咋一看，覺得此題目和經典的石子合併很相似，其實不然，石子的合併必須是相鄰的2堆，要按照一定得順序進行合併，而果子合併則不需要是相鄰的，可以採取以下的步驟進行合併：

①．先將所有的數按從大到小的順序排序，取最後兩個數合併；（之所以從大到小排列，是為了方便後面的陣列維護）

②．將兩個數的和插入到陣列中，（插入排序）隨時保持陣列有序；（如果在整理陣列的時候仍然調用快速排序，則速度會降慢，因為此時的陣列是有序的，只需要將合併後的數放入到有序數組的適當位置，使用插入排序更合適。）

③．合併過程中做陣列維護，合併到一堆時結束。

【參考程式】

program fruit;

var n,i,s,t,j,k:longint;

a:array[1..10000] of longint;

procedure quicksort(l,r:longint)；

var i,j,x,t:longint;

begin

  i:=l; j:=r; x:=a[(l+r)div 2];

  repeat

  while a[i]<x do inc(i);

  while a[j]>x do dec(j);

  if (a[i]>=a[j])and(i<=j) then

begin

   t:=a[i]; a[i]:=a[j]; a[j]:=t;

   inc(i); dec(j);

  end;

  until i>j;

 if j>l then quicksort(l,j);

 if i<r then quicksort(i,r);

end;

begin

 readln (n);

 for i:=1 to n do read (a[i]);

 quicksort (1,n);

 for i:=1 to n-1 do

begin

  t:=a[1]+a[2]; {合併最小的兩堆}

  s:=s+t;

  j:=3;

  while (t>a[j]) and (j<=n) do j:=j+1; {找到應插入的位置，邊界特殊判斷一下}

  j:=j-1; n:=n-1;

  for k:=1 to j-2 do a[k]:=a[k+2]; {這道題較特殊，前面的往前移兩位}

  a[j-1]:=t; {塞進去}

  for k:=j to n do a[k]:=a[k+1]; {後面的依然前移，不過是一位}

  end;

 writeln (s);

end.

**哈夫曼編碼**

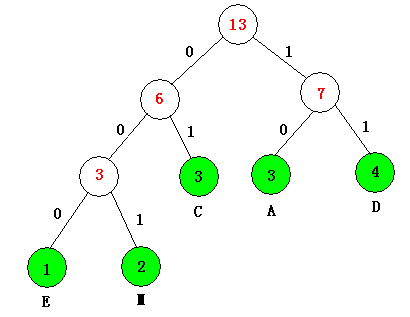
從哈夫曼樹根結點開始，對左子樹分配代碼“0”，右子樹分配代碼“1”，一直到達葉子結點為止，然後將從樹根沿每條路徑到達葉子結點的代碼排列起來，便得到了哈夫曼編碼。

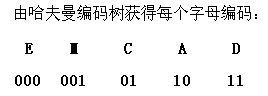
例，對電文 EMCAD 編碼。若等長編碼，則

shu-2-11

EMCAD 的編碼為：000001010011100 共15位

設各字母的使用頻度為 {E,M,C,A,D}={1,2,3,3,4}。用頻度為權值生成哈夫曼樹，並在葉子上標注對應的字母，樹枝分配代碼“0”或“1”:





各字母的編碼即為哈夫曼編碼：  EMCAD 的編碼為：000001011011 共12位

**【習題】**

1、用一維陣列存放的一棵完全二叉樹：abcdefghijkl。試寫出後序遍歷該二叉樹的訪問結點的順序。

2、輸入一組關鍵字:49,38,05,97,76,13,27,38,44,82,35,50，畫出由此而生成的二叉排序樹。

3、已知一棵二叉樹的前序遍歷為HGEDBFCA，中序遍歷為EGBDHFAC,請畫出該棵二叉樹？

4、二叉樹T，已知其先序遍歷是1 2 4 3 5 7 6（數字為節點編號，以下同），後序遍歷是4 2 7 5 6 3 1，則該二叉樹的中根遍歷是（ ABD ）

A．4 2 1 7 5 3 6 B．2 4 1 7 5 3 6 C．4 2 1 7 5 6 3 D．2 4 1 5 7 3 6

5、最優首碼編碼，也稱Huffman編碼。這種編碼組合的特點是對於較頻繁使用的元素給與較短的唯一編碼，以提高通訊的效率。下面編碼組合哪一組不是合法的首碼編碼：

A）（00，01，10，11） B）（0，1，00，11）

C）（0，10，110，111） D）（1，01，000，001）

6、平衡二叉樹的定義是，或者是一棵空樹，或者是具有如下性質的二叉樹：它的左、右子樹都是平衡二叉樹，且左、右子樹的深度之差的絕對值不超過1。設計遞迴演算法，判斷一棵給定的樹是否為平衡二叉樹。

**第四章 圖及其應用**

4．1圖的定義

**定義**：圖是由頂點集合及頂點間的關係集合組成的一種資料結構，graph=(v，e)，其中

    1)v是頂點的有窮非空有限集合；

    2)e是頂點之間關係的有窮集合，也稱為邊集合，e={(a,b)|a ∈ v，b ∈ v} 。

    如圖1所示

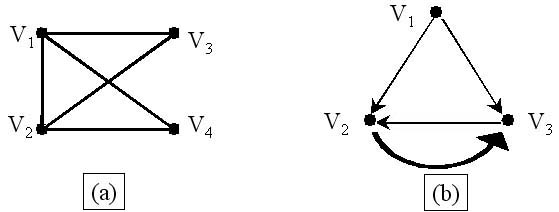


圖4-1-1

以上兩個均為圖。

(a)的頂點集合為：v={v1,v2,v3,v4}

(a)的邊集合為：e={(v1,v2), (v1,v3), (v1,v4), (v2,v3), (v2,v4)}

(b)的頂點集合為：v={v1,v2,v3}

(b)的邊集合為：v={<v1,v2>, <v1,v3>, <v2,v3>, <v3,v2>}

**無向圖與有向圖**：邊的表示方式是用該邊的兩個頂點來表示的，如果邊的表示無方向，那麼，對應的圖就是無向圖，否則稱為有向圖，如圖1中，(a)是無向圖，(b)是有向圖。

在無向圖中，邊的兩個頂點在邊的表示中可以互換，如邊（v1,v4）與邊（v4,v1）是等價的，表示的是同一條邊。（無向圖中邊的表示用圓括號）

在有向圖中，邊的走向不同就認為是不同的邊。如在邊的集合e={<v2,v3>, <v3,v2>}中，<v2,v3>表示該邊是由頂點v2出發，到頂點v3結束，即邊<v2,v3>表明了該邊的方向性，且兩個頂點的順序不能顛倒。（有向圖中邊的表示用尖括弧）

**完全圖**（每一對不同的頂點都有一條邊相連）**：**若有n個頂點的無向圖有n(n-1)/2條邊，則此圖為完全無向圖。有n個頂點的有向圖有n(n-1)條邊，則此圖為完全有向圖。

**頂點的度**：與頂點關聯的邊的條數，有向圖中等於該頂點的入度與出度之和。

**入度**——以該頂點為終點的邊的條數和。

**出度**——以該頂點為起點的邊的條數和。

**圖的階**：圖中頂點的個數。例如圖1中的(a)的階是4，(b)的階是3。

**帶權圖**：圖的每一邊附加一個具有代表性的資料。

**路徑**：從某個起點結點到某個終點結點的一條通路的結點序列。

**路徑長度：**非帶權圖的路徑長度是指此路徑上邊的條數。帶權圖的路徑長度是指路徑上各邊的權之和。

**簡單路徑**：若路徑上各頂點v1，v2，…，vn均不互相重複，則稱這樣的路徑為簡單路徑。

**回路**：若路徑上起點與終點重合，則此路徑稱為回路或環。

**連通圖與連通分量**：在無向圖中，若從頂點v1到頂點v2有路徑，則稱頂點v1與v2是連通的。如果圖中任意一對頂點都是連通的，則稱此圖是連通圖。非連通圖的極大連通子圖叫做連通分量。

**強連通圖與強連通分量**：在有向圖中，若對於每一對頂點vi和vj，都存在一條從vi到vj和從vj到vi的路徑，則稱此圖是強連通圖。非強連通圖的極大強連通子圖叫做強連通分量。

**網路**：帶權的連通圖。

4．2圖的存儲結構

圖在電腦中的表示，就是要把圖中點和邊存放到一些結構中。圖的存儲方法常用的有鄰接矩陣和鄰接表兩種。對於資料量比較小的情況，常用鄰接矩陣，在資料量大的情況下採用鄰接表。

**（1）鄰接矩陣**（陣列）

鄰接矩陣是表示頂點之間相鄰關係的矩陣，實質上是一個二維陣列。設g=(v,e)是具有n(n>=1)個頂點的圖，則g的鄰接矩陣是具有如下性質的n階方陣：

0或∞ 若(vi,vj)或< vj,vi >不是e(g)中的邊

a[i,j]=

1或權值 若(vi,vj)或< vj,vi >是e(g)中的邊

鄰接矩陣表示：

const n=100;

var v:array[1..n] of boolean; {頂點信息}

g:array[1..n,1..n] of integer {鄰接矩陣}

圖4-1-1（a）所示的鄰接矩陣表示為：

A（G）=

0 1 1 1

1 0 1 1

1 1 0 0

1 1 0 0

下面給出帶權無向圖的鄰接矩陣的建立過程：

for i:=1 to n do

for j:=1 to n do

g[i,j]:=32767; {初始化，對不帶權的圖置為32767，看成∞}

readln(e); {讀入邊的數目}

for k:=1 to e do

begin

read(i,j,w); {讀入兩個頂點的序號及權值}

g[i,j]:=w; {w表示權值}

g[j,i]:=w;

end.

從上圖也可看出，各頂點的度數總和是3+3+2+2=10，邊數是5，即各頂點的度數總和是邊數的2倍。同時也可看出，用鄰接矩陣表示的無向圖是一個對稱的矩陣。

**（2）鄰接表**（鏈表）

鄰接表是圖的一種鏈式存儲結構。在鄰接表中，對圖的每一個頂點vi（1≤i≤n）建立一個鄰接關係的單鏈表，為每個頂點vi（1≤i≤n）建立的單鏈表，是表示以該頂點為起點的所有邊的資訊。第i個 單鏈表中的結點表示依附於頂點vi（1≤i≤n）的邊。每個結點由3個域組成：

|  |  |  |
| --- | --- | --- |
| 頂點域 | 資料欄 | 連結域 |

頂點域——指鄰接點的序號；

資料欄----權值（注：有權值才定義，否則不定義）；

連結域——指向依附於頂點vi的下一條邊所對應的結點。

圖4-1-1（a）用鄰接表表示為：

|  |  |
| --- | --- |
| 1 |  |
| 2 |  |
| 3 |  |
| 4 |  |

|  |  |
| --- | --- |
| 2 |  |

|  |  |
| --- | --- |
| 3 |  |

|  |  |
| --- | --- |
| 4 |  |

|  |  |
| --- | --- |
| 1 |  |

|  |  |
| --- | --- |
| 3 |  |

|  |  |
| --- | --- |
| 4 |  |

|  |  |
| --- | --- |
| 1 |  |

|  |  |
| --- | --- |
| 2 |  |

|  |  |
| --- | --- |
| 1 |  |

|  |  |
| --- | --- |
| 2 |  |

一個圖的鄰接表存儲結構說明如下：

type

   arcptr=^arcnode; {邊結點信息}

   arcnode=record

     adjvex:1..num;  {邊上的終點（鄰接點域）}

weight:integer; {該邊上的權，無權圖可以省去}

     next:atrcptr;   {指向下一條邊的連結（鏈域）}

     end;

   vexnode=record {表頭結點，包括一個起點資訊和一個連結}

     data:integer; {起點資料類型}

     link:arcptr {指向下一個頂點的連結資訊}

     end

 var adj1ist=array[1..num] of vexnode;

4．3 圖的遍歷

從圖中的某個頂點出發系統地訪問圖中所有頂點，使每個頂點恰好被訪問一次，這種運算操作被稱為圖的遍歷。遍歷的結果是將頂點排成一定的序列。為了避免重複訪問某個頂點，可以設一個標誌陣列visit[i]，未訪問時置初值為false(0)，訪問一次後就修改其值為true(1)。

圖的遍歷分為深度優先遍歷和廣度（寬度）優先遍歷兩種。

**（1）深度優先遍歷（dfs）**

從圖中的某個頂點vi出發，訪問此頂點並作已訪問標記，然後依次從vi的一個未被訪問過的鄰接頂點vj出發再進行深度優先遍歷。當vi的所有鄰接點都被訪問過時，則退回到上一個頂點vk，再從vk的另一個未被訪問過的鄰接點出發進行深度優先遍歷，直至圖中所有頂點都被訪問到為止。

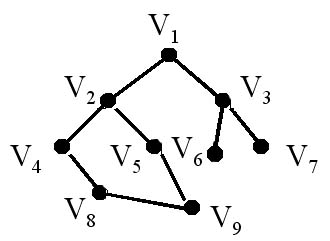


圖4-3-1

如圖4-3-1的深度優先遍歷的結點序列：v1， v2， v4， v8， v9， v5， v3， v6， v7

深度優先遍歷的實現（用鄰接矩陣）：

program graphdfs;

const num=100;

var visit:array[1..num] of boolean; {記錄頂點是否訪問的標誌，訪問過則置1，否則置0}

a:array[1..num,1..num] of integer;

i,j,n:integer;

procedure dfs(i:integer);

var j:integer;

begin

vist[i]:=true;

for j:=1 to n do

if (not visit[j]) and (a[i,j]=1) then dfs(j); {如果該頂點沒被訪問且鄰接}

end

begin

readln(n);

fillchar(visit,sizeof(visit),false); {初始化vist陣列}

for i:=1 to n do

for j:=1 to n do

read(a[i,j]); {鄰接矩陣讀入初值}

dfs(1);

end.

**（2）廣度優先遍歷（bfs）**

從圖中的某個結點vi出發，訪問此結點，然後依次訪問與vi鄰接的、未被訪問的過的所有結點，然後再分別從這些結點出發進行廣度優先遍歷，直到圖中所有被訪問過的結點的相鄰結點都被訪問到。若此時圖中尚有結點未被訪問，則另選圖中一個未被訪問的結點作起始點重複上述過程，直至圖中所有結點都被訪問到為止。

如圖4-3-1的廣度優先遍歷的結點序列：v1， v2， v3， v4， v5， v6， v7， v8， v9

廣度優先搜索基本演算法：

program bfs;

初始化；建立佇列data;

設佇列首指標head:=0;佇列尾指標tail:=1;

while head <tail

begin

head 增1，取出head所指結點進行擴展；

for i:=1 to r do begin

if 子結點符合條件then begin

tail增1，並把新結點存入資料庫隊尾；

if新結點與原有結點有重複 then 刪於該結點(tail減1)

else if 新結點即目標 then 輸出並退出 ;

end;

end;

end;

廣度優先遍歷的實現（用鄰接矩陣）：

program grahpbfs;

const num=100;

var visit:array[1..num] of boolean;

a:array[1..num,1..num] of integer;

i,j,k,n:integer;

head,tail:integer;

procedure bfs(x:integer);

var i,j:integer;

begin

head:=1;tail:=1;

fillchar(visit,sizeof(visit),false);

visit[head]:=true;

while head<>tail do

begin

for i:=1 to n do

if (a[head,i]=1) and (visit[i]=false)

then

begin

inc(tail);visit[i]:=true;

end;

end;

end;

begin

read(n);

for i:=1 to n do

for j:1 to n do

read(a[i,j]);

bfs[1];

end.

4．4 最短路徑

最短路徑問題：

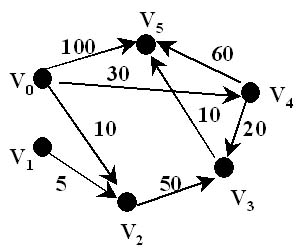


圖4-4-1

如圖4-4-1所示，圖中頂點表示城市，邊表示城市間的交通聯繫，邊上是數值是該兩城市的距離。假如某旅客想從城市v0到城市v5，那他如何選擇路線，使之最短呢？這就是生活中常見的路徑最短問題。

**（1）單源最短路徑**（從某個源點到其餘各頂點的最短路徑）

給定帶權有向圖g=（v，e），其中每條邊的權為非負實數，另外，再給定v中的一個頂點，我們稱之為源點，要求從源點到其他各頂點的最短路徑長度。這個問題稱為圖論單源最短路徑問題。

dijkstra（迪傑斯特拉）演算法是解決這類問題的貪心演算法。其基本思想如下：

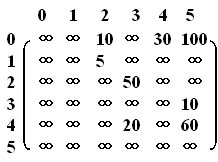
①假設用帶權的鄰接矩陣cost來表示帶權有向圖，cost[i,j]表示邊<vi,vj>上的權值。若<vi,vj>不存在，則置cost[i,j] 為∞（在電腦上可用允許的最大值表示）。s為已找到從源點出發的最短路徑的終點的集合，它的初始狀態為空集。那麼，從源點出發到圖上其餘各頂點（終點）vi可能達到的最短路徑長度的初值為：dist[i]=cost[源點,i ]。

②選擇vj， 使得dist[j]=min{dist[i]|vi∈v-s}，vj就是當前求得的一條從源點出發的最短路徑的終點。令s=s∪{j}。

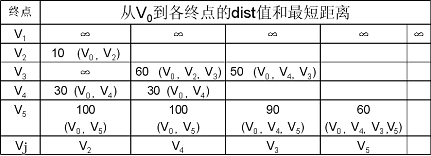
③修改從源點出發到集合v-s上任一頂點vk可達的最短路徑長度。如果dist[j]+ cost[j,k] ＜dist[k]，則修改dist[k]為: dist[k]= dist[j]+ cost[j,k]。

④重複操作② ③共n-1次，由此求得從源點到圖上其餘各頂點的最短路徑是依路徑長度遞增的序列。

上圖所示有向圖的帶權鄰接矩陣為：



演算法實施過程中dist向量的變化狀況如下：



dijkstra（迪傑斯特拉）演算法程式實現如下：

program minroadpath;

const

max=30000;

n=5;

cost:array[0..n,0..n]of integer=((max,max,10,max,30,100),(max,max,5,max,max,max),(max,max,max,50,max,max),(max,max,max,max,max,10),(max,max,max,20,max,60),(max,max,max,max,max,max));{陣列賦初值}

var dist:array[1..n] of integer;

i,j,k,wm:integer;

s:set of 0..n;

path:array[1..n] of set of 0..n;{path為一個集合類型的陣列，用來記錄路徑}

begin

for i:=1 to n do

begin

dist[i]:=cost[0,i];

if dist[i]<>max then path[i]:=path[i]+[i];

end;

s:=[0];

for k:=1 to n do

begin

wm:=max;j:=0;

for i:=1 to n do {尋找當前從源點出發路徑最短的結點}

if not (i in s) and (dist[i]<wm) then

begin j:=i;wm:=dist[i];end;

s:=s+[j];

for i:=1 to n do {修改選擇vj結點後各結點的最短路徑}

if not (i in s) and (dist[j]+cost[j,i]<dist[i]) then

begin

dist[i]:=dist[j]+cost[j,i];

path[i]:=path[j]+[i];

end;

end;

for i:=1 to n do {輸出}

begin

write(dist[i],' ');

for j:=1 to n do

if j in path[i] then write('v',j,' ');

writeln;

end;

end.

**（2）每一對頂點的最短路徑**

只要對以上演算法進行 n 次求解，便可求得每一對頂點的最短路徑。

4．5 最小生成樹

**（1）生成樹的概念**

 設圖g＝(v，e)是一個連通圖，當從圖中任一頂點出發遍歷圖g時，將邊集e(g)分成兩個集合a(g)和b(g)。其中a(g)是遍歷圖時所經過的邊的集合，b(g)是遍歷圖時未經過的邊的集合。顯然，g1＝(v，a)是圖g的子圖，則稱子圖g1是連通圖g的生成樹。圖的生成樹不是惟一的。如對圖4-5-1(a)，當按深度和廣度優先搜索法進行遍歷就可以得到圖4-5-1中(b)和(c)的兩棵不同的生成樹，並分別稱之為深度優先生成樹和廣度優先生成樹。

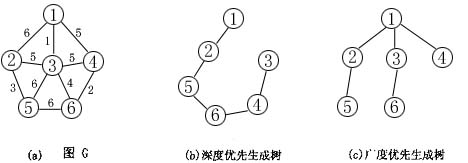


圖4-5-1

對於有n個頂點的連通圖，至少有n-1條邊，而生成樹中恰好有n-1條邊，所以連通圖的生成樹是該圖的極小連通子圖。若圖g的生成樹中任意加一條邊屬於邊集b(g)中的邊，則必然形成回路。

 求解生成樹在許多領域有實際意義。例如，對於供電線路或煤氣管道的鋪設問題，即假設要把n個城市聯成一個供電或煤氣管道網路，則需要鋪設n－1條線路。任意兩城市間可鋪設一條線路，n個城市間最多可能鋪設n(n－1)/2條線路，各條線路的造價一般是不同的。一個很實際的問題就是如何在這些可能的線路中選擇n-1條使該網路的建造費用最少，這就是下面要討論的最小生成樹問題。

**（2）圖的最小生成樹**

在前面我們已經給出圖的生成樹的概念。這裡來討論生成樹的應用。

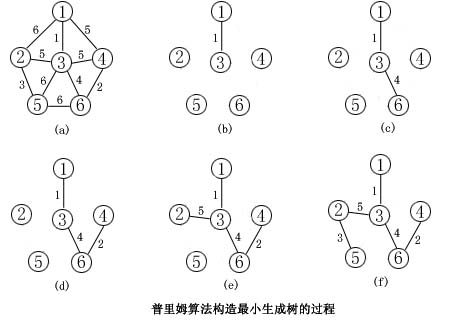
假設，要在n個居民點之間敷設煤氣管道。由於，在每一個居民點與其餘n－1個居民點之間都可能敷設煤氣管道。因此，在n個居民點之間，最多可能敷設n(n-1)/2條煤氣管道。然而，連通n個居民點之間的管道網路，最少需要n-1條管道。也就是說，只需要n-1條管道線路就可以把n個居民點間的煤氣管道連通。另外，還需進一步考慮敷設每一條管道要付出的經濟代價。這就提出了一個優選問題。即如何在n(n-1)/2條可能的線路中優選n-1條線路，構成一個煤氣管道網路，從而既能連通n個居民點，又能使總的花費代價最小。

解決上述問題的數學模型就是求圖的最小生成樹問題。把居民點看作圖的頂點，把居民點之間的煤氣管道看作邊，而把敷設各條線路的代價當作權賦給相應的邊。這樣，便構成一個帶權的圖，對於一個有n個頂點的圖可以生成許多互不相同的生成樹，每一棵生成樹都是一個可行的敷設方案。現在的問題是應尋求一棵所有邊的權總和為最小的生成樹。

如何構造這種圖的最小生成樹呢?下面有兩種常用的演算法。

【**prim（普裡姆）演算法**】

任取一個頂點加入生成樹，然後對那些一個端點在生成樹中，而另一個端點不在生成樹中的邊進行排序，取權值最小的邊，將它和另一個端點加進生成樹中。重複上述步驟直到所有頂點都進入了生成樹為止。



【**演算法實現**】

在實際的程式設計中，關鍵在於如何找出滿足條件的頂點和邊，以便加入到生成樹中。i∈s（已經加入的頂點的集合），j∈v－s（未加入的頂點），且cost[i,j ]是權最小的邊（i,j）。設置兩個陣列closed和mincost。對每一個j∈v－s，closed[j]是j在s中的鄰接頂點，它與j在s中的所有其他頂點k相比較cost[j,closed[j]]<=cost[j,k]。mincost[j]的值就是cost[j,closed[j]]。在程式的初始，先將mincost[i]設置為頂點i到第一個頂點的權值，closed[i]設置為1，在prim演算法的執行過程中，先找出v－s中是mincost值最小的頂點j，然後根據陣列closed選取邊（j,closed[j]），最後將頂點j添加到s中，並對陣列closed和mincost進行修改。

【**參考程式**】

program prim-mintree;

const maxn=50;

var cost:array[1..maxn,1..maxn] of integer;

mincost,closed:array[1..maxn] of integer; {mincost陣列用來更新已加入頂點和未加入頂點的權值，closed陣列用來更新與已加入頂點的關係}

min,sum,k,i,j,n,x,y:integer;

begin

assign(input,'prim.in');

reset(input);

assign(output,'prim.out');

rewrite(output);

read(n);

for i:=1 to n do

for j:=1 to n do

begin

read(cost[i,j]);

if (i<>j) and (cost[i,j]=0) then cost[i,j]:=maxint;

end;

close(input);

for i:=1 to n do {將第一個頂點加入}

begin

mincost[i]:=cost[1,i];closed[i]:=1;

end;

for i:=2 to n do {選取最小的邊，並將頂點加入}

begin

min:=maxint;

for j:=1 to n do

if (mincost[j]<min) and (mincost[j]<>0) then

begin

min:=mincost[j];k:=j;

end;

mincost[k]:=0;

for j:=1 to n do {更新}

if (mincost[j]>cost[k,j]) and (mincost[j]<>0) then

begin

mincost[j]:=cost[k,j];closed[j]:=k;

end;

sum:=sum+min;

end;

for i:=2 to n do

writeln(i,'--->',closed[i]);

writeln(sum);

close(output);

end.

輸入:

6

0 6 1 5 0 0

6 0 5 0 3 0

1 5 0 5 6 4

5 0 5 0 0 2

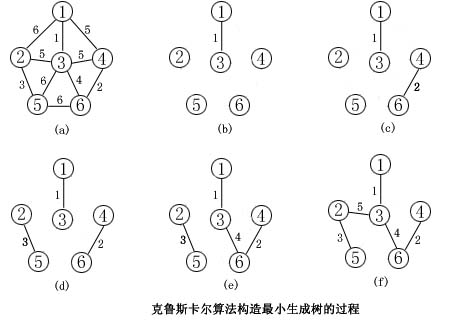
0 3 6 0 0 6

0 0 4 2 6 0

【**kruskal(克魯斯卡爾)演算法**】

對給定圖的所有邊從小到大排序，依次試探將邊和它的端點加入生成樹。如果加入的邊不產生環，則繼續將邊和它的端點加入，否則，將它刪去。直到生成樹中含有了n-1條邊。

【分析】這是一種貪心法。要使生成樹最小，可以再每次都在剩餘邊中挑選最小的邊加入到生成樹，直到所有頂點都在生成樹中。要實現這一目的，要解決三個問題。首先要對所有邊按權從小到大排序，我們可以選用高效的快速排序來實現。其實，要淘汰一些邊，即使它們的權比較小，但在生成樹中它們是多餘的（形成環），我們用**並查集**來解決。每取一條邊，我們就用find函數判斷這條邊的兩個端點是否在同一個集合中，如果是則說明這條邊不能再添加入到生成樹中了。第三，演算法什麼時候結束呢？我們可以設置一個計數器p，初始值為邊數-1。當p=0時，表示所有的邊都處理完畢，演算法結束。



【**參考程式**】

program kruskal\_mintree;

const vnum=6; {頂點的個數}

enum=10; {邊的條數}

type node=record

len:integer;

v1,v2:integer;

end;

edge=array[1..enum] of node;

var

g:array[1..vnum,1..vnum] of integer;

vset:array[1..vnum] of set of 1..vnum;

i,j,k,n,m,p,q,tot:integer;

e:edge; {tot:記最小生成樹的權值}

function find(v:integer):integer; {返回頂點v所在的集合}

var i:integer;

begin

i:=1;

while (i<=n) and ( not (v in vset[i])) do inc(i);

if i<=n then find:=i else find:=0;

end;

procedure quicksort(var b:edge; left,right:integer);

var i,j,x,t1,vtemp:integer;

begin

i:=left;j:=right;x:=b[i].len;

repeat

while (b[j].len>=x) and (j>i) do j:=j-1;

if j>i then begin

t1:=b[i].len; b[i].len:=b[j].len;b[j].len:=t1;

vtemp:=b[i].v1; b[i].v1:=b[j].v1;b[j].v1:=vtemp;

vtemp:=b[i].v2; b[i].v2:=b[j].v2;b[j].v2:=vtemp;

end;

while (b[i].len<=x) and (i<j) do i:=i+1;

if i<j then begin

t1:=b[j].len;b[j].len:=b[i].len;b[i].len:=t1;

vtemp:=b[j].v1; b[j].v1:=b[i].v1;b[i].v1:=vtemp;

vtemp:=b[j].v2; b[j].v2:=b[i].v2;b[i].v2:=vtemp;

end

until i=j;

b[i].len:=x;

i:=i+1;j:=j-1;

if left<j then quicksort(b,left,j);

if i<right then quicksort(b,i,right);

end;

begin

assign(input,'kruskal.in');

reset(input);

assign(output,'kruskal.out');

rewrite(output);

readln(n,m);

for i:=1 to n do

begin

for j:=1 to n do read(g[i,j]);

readln;

end;

close(input);

k:=1;

while k<=m do

for i:=1 to n do

for j:=1 to n do

if g[i,j]<>0 then

begin

e[k].len:=g[i,j];e[k].v1:=i;

e[k].v2:=j;g[j,i]:=0;k:=k+1;

end;

quicksort(e,1,m); {對所有邊按權值遞增排序，存於e[i]中，e[i].v1與e[i].v2為邊i所連接的兩個頂點的序號，e[i].len為第 i務邊的長度}

for i:=1 to n do vset[i]:=[i]; {初始化定義n個集合，第i個集合包含一個元素i}

p:=n-1; q:=1; tot:=0; {p為待加入的邊數，q為邊集指針}

while p>0 do

begin

i:=find(e[q].v1);j:=find(e[q].v2);

if i<>j then

begin

inc(tot,e[q].len);

vset[i]:=vset[i]+vset[j];vset[j]:=[];

dec(p);

end;

inc(q);

end;

writeln('tot=',tot);

end.

kruskal.in

6 10

0 6 1 5 0 0

6 0 5 0 3 0

1 5 0 5 6 4

5 0 5 0 0 2

0 3 6 0 0 6

0 0 4 2 6 0

kruskalout

tot=15

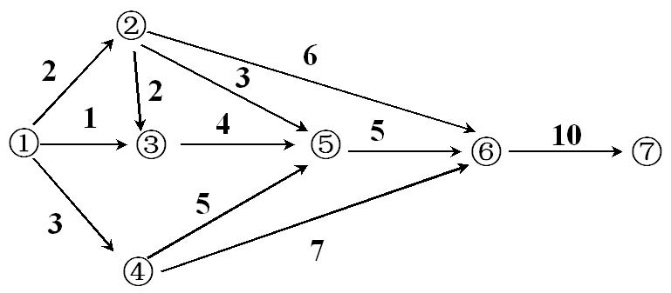
4．6 關鍵路徑

**（1）拓撲排序**

【**案例**】　電腦機房的裝修是這樣進行的：裝水（2天），裝電（5天），可同時進行，水電裝完後開始牆壁（6天）、地面（3天）的粉刷（也可同時進行），最後是安裝電腦設備（3天）。問總共至少要多少天完工？

在日常生活中，一項大的工程可以看作是由若干個子工程（這些子工程稱為“活動”）組成的集合，這些子工程（活動）之間必定存在一些先後關係，即某些子工程必須在其他一些子工程完成之後才能開始，我們可以用有向圖來形象地表示這些子工程之間的先後關係，子工程為頂點，子工程之間的先後關係為有向邊，這種有向圖稱為“aov網”。在aov網中，有向邊代表子工程的先後關係，即有向邊的起點活動是終點活動的前驅活動，只有當起點活動完成之後終點活動才能進行。一個aov網應該是一個有向無環圖，即不帶有回路。

把不帶回路的aov網中的所有活動排成一個線性序列，使得每個活動的所有前驅活動都排在該活動的前面，這個過程稱為“拓撲排序”，所得到的活動序列稱為“拓撲序列”。



aov網的拓撲序列是不惟一的。如上圖的拓撲序列至少有：①②③④⑤⑥⑦，①④②③⑤⑥⑦。

拓撲排序的演算法思想：首先選擇一個無前驅的頂點（即入度為0的頂點，圖中至少應有一個這樣的頂點，否則肯定存在回路），然後從圖中移去該頂點以及由他發出的所有有向邊，如果圖中還存在無前驅的頂點，則重複上述操作，直到操作無法進行。如果圖不為空，說明圖中存在回路，無法進行拓撲排序；否則移出的頂點的順序就是對該圖的一個拓撲排序。

拓撲排序的演算法實現：

網採用鄰接矩陣a表示,若a[i，j]=1，表示活動i先於j，a[i，j]=0，表示活動i與j不存在先後關係。

（1）計算各頂點的入度

（2）找入度為零的點輸出之，刪除該點，且與該點關聯各點的入度減1

（3）若所有頂點都輸出完畢。

程式如下：

program toupusort;

const maxn=100;

var

map:array[1..maxn,1..maxn] of byte;

into:array[1..maxn] of byte; {存儲結點的入度}

n,i,j,k:byte;

procedure init;

var

i,j:integer;

begin

read(n);

fillchar(into,sizeof(into),0);

for i:=1 to n do

for j:=1 to n do

begin

read(map[i,j]);

if map[i,j]<>0 then inc(into[j]); {計算結點的入度}

end;

end;

begin

init;

for i:=1 to n do

begin

j:=1;

while (j<=n) and (into[j]<>0) do inc(j);

write(j,' ');

into[j]:=255;

for k:=1 to n do {調整結點的入度}

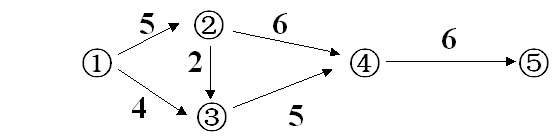
if map[j,k]=1 then dec(into[k]);

end;

end.

**（2）關鍵路徑**

設有一個工程網路如下圖表示(無環路的有向圖)，若結點的排列已經過拓撲排序，即序號前面的結點會影響序號後面結點的活動，



其中，頂點表示活動，①表示工程開始，⑤表示工程結束(可變，用n表示)，邊上的數字表示活動延續的時間。

如上圖中，活動①開始5天后活動②才能開始工作，而活動③則要等①、②完成之後才能開始，即最早也要7天后才能工作。

在工程網路中，延續時間最長的路徑稱為**關鍵路徑**。

上圖中的關鍵路徑為：①—②—③—④—⑤共18天完成。

如果將① ③之間的活動時間改為14天，則關鍵路徑是怎樣的呢？

**【演算法分析】**

①、資料結構：

r[1..n，1..n]of integer； 表示活動的延續時間，若無連線，則用-1表示；

eet[1..n]　　　　　　　 表示活動最早可以開始的時間

et[1..n]　　　　　　 　 表示活動最遲應該開始的時間

關鍵路徑通過點j，具有如下的性質：eet[j]=et[j]

②、約定：

 結點的排列已經過拓撲排序，即序號前面的結點會影響序號後面結點的活動。

【**參考程式**】

program gao;

　var i,j,n,max,min,w,x,y:integer;

　　　r:array[1..20,1..20]　of integer;

　　　eet,et:array[1..20]　of integer;

　begin

　　readln(n);

　　for i:=1 to n do

　　　for j:=1 to n do r[i,j]:= -1;

　　 readln(x,y,w);{輸入從活動x到活動y的延續時間,以0為結束}

　 while x<>0 do　begin　r[x,y]:=w;readln(x,y,w)

;end;

　　eet[1]:=0;{認為工程從0天開始}

　　for i:=2 to n do

　　 begin

max:=0;

　　　　for j:=1 to n do

　　　　　if r[j,i]<>-1 then

　　　　　　　 if r[j,i]+eet[j]>max

then max:=r[j,i]+eet[j];

　　　　eet[i]:=max;

end;

　 et[n]:=eet[n]

　　for i:=n-1 downto 1 do

　　　begin

　　　　min:=10000;

　　　　for j:=1 to n do

　　　　　if r[i,j]<>-1 then

　　　　　　　　if et[j]-r[i,j]<min

then min:==et[j] - r[i,j];

　　　　et[i]:==min;

　　　end;

　　writeln(eet[n]);

　　for i:=1 to n -1 do

　　　　　if eet[i]=et[i]

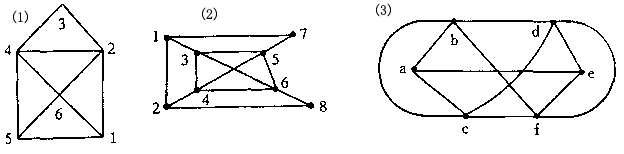
then write(i,'→');

　　write(n);readln

end.

4．6 一筆劃問題

【**例4-6-1**】 程式設計找出下圖4-6-1中（1）的一筆劃路線。



由數學知識可知：當一個圖的頂點全是偶點或僅有兩個奇點時才能一筆劃出，而且此圖必須是連通圖。

【**演算法分析**】

①、首先用鄰接矩陣來表示圖：如果i,j兩點間有線段連接則值為1，否則為0。

0 1 0 0 1 1

1 0 1 1 0 1

0 1 0 1 0 0

0 1 1 0 1 1

1 0 0 1 0 1

1 1 0 1 1 0

可用陣列常量links[i,j] 表示，即：

const links:array[1..n, 1..n] of integer= ((0,1,0,0,1,1), (1,0,1,1,0,1),(0,1,0,1,0,0),

(0,1,1,0,1,1),(1,0,0,1,0,1), (1,1,0,1,1,0));

②、計算每個點的度數存dgr[i]中，如對於第i個頂點，可這樣統計：

for j:=1 to n do dgr[i]:=dgr[i]+links[i,j];

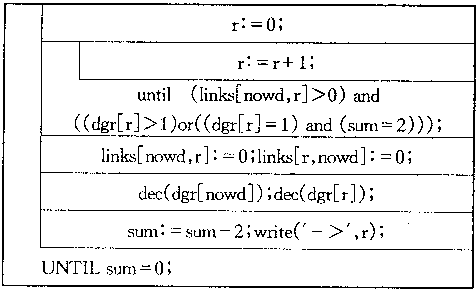
sum:=sum+dgr[i];

③、統計奇點的個數，即dgr中有幾個奇點，存odt中。

if odd(dgr[i]) then odt:=odt+1;

start:=i;

④、如果odt>2則無解，否則從一個奇點開始搜索路線，其演算法如下



【**程式清單**】

program postal\_route;

const n = 6;

links:array[1..n, 1..n] of integer= ((0,1,0,0,1,1), (1,0,1,1,0,1), (0,1,0,1,0,0), (0,1,1,0,1,1), (1,0,0,1,0,1), (1,1,0,1,1,0));

var

dgr: array[1..n] of integer;

i, j, r, sum, odt, start, nowd: integer;

procedure find\_degree; {求每個結點的度}

begin

sum:=0; odt:=0; start:=1;

for i:= 1 to n do

begin

dgr[i]:= 0;

for j:= 1 to n do dgr[i]:= dgr[i] + links[i,j];

sum:= sum + dgr[i]; {記錄總的度數}

if odd(dgr[i]) then

begin odt := odt + 1; start:= i end

end;

end;

begin

find\_degree

if odt>2 then

begin writeln ('no sulution . ');exit end;

nowd:= start;

write(start);

repeat {從起點開始輸出結點}

r:=0;

repeat

r:=r+1;

until (links[nowd,r]>0) and ((dgr[r]>1) or ((dgr[r]=1) and (sum=2)));

links[nowd,r]:=0; links[r,nowd]:=0; sum:=sum-2;

dec(dgr[nowd]); dec(dgr[r]);

nowd :=r;

write('->',r);

until sum = 0;

writeln; readln

end.

【**運行結果**】

5->1->2->3->->4->2->6->4->5->6->1

**【思考練習】**試程式設計找出圖4-6-1中第（2）和第（3）圖一筆劃的路線。

4．7 網路流問題

在實際生活中有許多流量問題，例如在交通運輸網路中的人流、車流、貨物流，供水網路中的水流，金融系統中的現金流，通訊系統中的資訊流，等等。50年代以福特(ford)、富克遜(fulkerson)為代表建立的“網路流理論”，是網路應用的重要組成部分。在近年的奧林匹克資訊學競賽中，利用網路流演算法高效地解決問題已不是什麼稀罕的事了。本節著重介紹最大流(包括最小費用)演算法，並通過實際例子，討論如何在問題的原型上建立—個網路流模型，然後用最大流演算法高效地解決問題。

**【問題描述】**如圖4-7-1所示是聯結某產品地v1和銷售地v4的交通網，每一弧(vi,vj)代表從vi到vj的運輸線，產品經這條弧由vi輸送到vj，弧旁的數表示這條運輸線的最大通過能力。產品經過交通網從v1到v4。現在要求制定一個運輸方案使從v1到v4的產品數量最多。

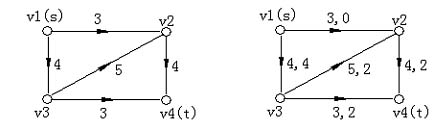
****

圖4-7-1

**（1）基本概念及相關定理**

**網路與網路流**

**定義1** 給一個有向圖n=(v，e)，在v中指定一點，稱為源點(記為vs，和另一點，稱為匯點(記為vt)，其餘的點叫中間點，

對於e中每條弧(vi，vj)都對應一個正整數c(vi，vj)≥o(或簡寫成cij)，稱為f的容量，則賦權有向圖n=(v，e，c，vs，vt)稱為一個網路。如圖4-1所給出的一個賦權有向圖n就是一個網路，指定v1是源點，v4為匯點，弧旁的數字為cij。

所謂網路上的流，是指定義在弧集合e上一個函數f={f(vi，vj)}，並稱f(vi，vj)為弧(vi，vj)上的流量(下麵簡記為fij)。如圖4-7-1所示的網路n，弧上兩個數，第一個數表示容量cij，第二個數表示流量fij。

**可行流與最大流**

在網路的一個可行流中，即任意一個實際運輸情況中，實際運輸量必須滿足一些關係：

1、實際運輸量不能是負的

2、每個弧上的流量不能超過該弧的最大通過能力(即弧的容量)；

3、中間點的流量為0（即指向vi的弧上的運輸量之和，應該等於所有從vi出發的弧上的運輸量之和），源點的淨流出量和匯點的淨流入量必相等且為這個方案的總輸送量。因此有：

**定義2** 滿足下列條件

(1)容量約束：0≤fij≤cij，(vi,vj)∈e

(2)守恆條件

對於中間點：流入量=流出量；對於源點與匯點：源點的淨流出量vs(f)=匯點的淨流入量（-vt(f)）的流f，稱為網路n上的**可行流**，並將源點s的淨流出量稱為流f的流值v(f)。

網路n中流值最大的流f\*稱為n的最大流。

**可增廣路徑**

所謂可增廣路徑，是指這條路徑上的流可以修改，通過修改，使得整個網路的流值增大。

若給定一個可行流f=(fij),我們把網路中fij=cij的弧稱作飽和弧， fij<cij的弧稱作非飽和弧， fij=0的弧稱作零流弧， fij>0的弧稱作非零流弧。

若p是網路中聯結源點s和匯點t的的一條路(不用管邊的有向性)，我們定義路的方向是從vs到vt，則路上的弧被分為兩類—— 一類與路的方向一致，稱為前向弧；另一類和路的方向相反，稱為後向弧。

如果對於p上的每一條前向弧都是非飽和弧，每一條後向弧都是非零流弧，則稱p為一條可增廣路。因為可以通過修正p上各弧的流量來使得總流量變得更大。修正的方法是：

w1－不屬於p上的弧一概不變

w2－對於p上的所有前向弧加上a，後向弧減去a。這裡a是一個可改進量。

**（2）最大流演算法**

【**演算法思想**】 :最大流問題實際上是求一可行流{fij}，使得v(f)達到最大。若給了一個可行流f，只要判斷網路n中有無關於f的增廣路徑，如果有增廣路徑，改進f， 得到一個流量增大的新的可行流；如果沒有增廣路徑，則得到最大流。

1、尋求最大流的標號法(ford,fulkerson)

從一個可行流(一般取零流)開始，不斷進行以下的標號過程與調整過程，直到找不到關於f的可增廣路徑為止。

①標號過程

在這個過程中，網路中的點分為已標號點和未標號點，已標號點又分為已檢查和未檢查兩種。每個標號點的標號資訊表示兩個部分：第一標號表明它的標號從哪一點得到的，以便從vt開始反向追蹤找出可增廣路徑；第二標號是為了表示該頂點是否已檢查過。

標號開始時，給vs標上(s，0)，這時vs是已標號但末檢查的點，其餘都是未標號的點，記為(0，0)。

取一個已標號而未檢查的點vi，對於一切未標號的點vj：

a．對於弧(vi，vj)，若fij<cij，則給vj標號(vi，0)，這時，vj點成為已標號而未檢查的點。

b．對於弧(vi，vj)，若fji>0，則給vj標號(-vi，0)，這時，vj點成為已標號而未檢查的點。

於是vi成為已標號且已檢查的點，將它的第二個標號記為1。重複上述步驟，一旦vt被標上號，表明得到一條從vi到vt的增廣路徑p，轉入調整過程。若所有標號都已檢查過，而標號過程進行不下去時，則演算法結束，這時的可行流就是最大流。

②調整過程

從vt點開始，通過對每個點的第一個標號，反向追蹤，可找出增廣路徑p。例如設vt的第一標號為vk(或-vk)，則弧(vk，vt)(或 相應地(vt，vk))是p上弧。接下來檢查vk的第一標號，若為vi(或-vi)，則找到(vi，vk)(或相應地(vk，vi))。再檢查vi的第一 標號，依此類推，直到vs為止。這時整個增廣路徑就找到了。在上述找增廣路徑的同時計算q：

q=min{min(cij-fij)，minf\*ij}

對流f進行如下的修改：

f'ij =  fij+q   (vi，vj)∈ p的前向弧的集合

f'ij =  fij-q   (vi，vj)∈ p的後向弧的集合

f'ij =  f\*ij     (vi，vj)不屬於p的集合

接著，清除所有標號，對新的可行流f’，重新進入標號過程。

例4-7-1:下圖表示一個公路網,v1是某原材料產地,v6表示港口碼頭,每段路的通過能力(容量)如圖4-7-2上的各邊上的資料,找一運輸方案,使運輸到碼頭的原材料最多?

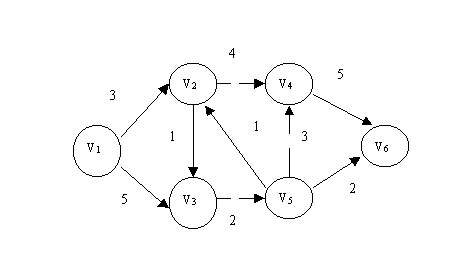


圖4-7-2

【**參考程式**】

program max\_stream;

const maxn=20;

type

  net=record

    c,f:integer; {c表示容量,f表示流量}

    end;

  node=record

    l,p:integer;{l記錄是否標號,p記錄是否檢查}

    end;

var

  lt:array[0..maxn] of node;

  g:array[0..maxn,0..maxn] of net;

  n,s,t:integer;

  f:text;

procedure init;{初始化過程，讀入有向圖，並設置流為0}

var fn :string;

  i,j :integer;

begin

  write( 'graph file = ' ); readln(fn);

  assign(f,fn);

  reset(f);

  readln(f,n);

  fillchar(g,sizeof(g) ,0);

  fillchar(lt,sizeof(lt),0);

  for i:=1 to n do

    for j:=1 to n do read(f,g[i,j].c);

  close(f);

end;

function find: integer; {尋找已經標號未檢查的頂點}

var i: integer;

begin

  i:=1;

  while (i<=n) and not((lt[i].l<>0)and(lt[i].p=0)) do inc(i);

  if i>n then find:= 0 else find:= i;

end;

function ford(var a: integer):boolean; {用標號法找增廣路徑，並求修改量ａ}

var

  i,j,m,x:integer;

begin

  ford:=true;

  fillchar(lt,sizeof(lt),0);

  lt[s].l:=s;

  repeat

    i:= find;

    if i=0 then exit;

    for j:=1 to n do

      if (lt[j].l= 0)and((g[i,j].c<>0)or(g[j,i].c<>0)) then

      begin

if (g[i,j].f<g[i,j].c) then lt[j].l:= i;

if (g[j,i].f>0) then lt[j].l:=-i;

    end;

    lt[i].p:=1;

  until (lt[t].l<>0);

  m:=t;a:=maxint;

  repeat

    j:=m;m:=abs(lt[j].l);

    if lt[j].l<0 then x:= g[j,m].f;

    if lt[j].l>0 then x:= g[m,j].c- g[m,j].f;

    if x<a then a:= x;

  until m= s;

  ford:=false;

end;

procedure change(a: integer);{調整過程}

var m, j: integer;

begin

  m:= t;

  repeat

    j:=m;m:=abs(lt[j].l);

    if lt[j].l<0 then g[j,m].f:=g[j,m].f-a;

    if lt[j].l>0 then g[m,j].f:=g[m,j].f+a;

  until m=s;

end;

procedure print; {列印最大流及其方案}

var

  i ,j: integer;

  max: integer;

begin

  max:=0;

  for i:=1 to n do

  begin

    if g[i,t].f<>0 then max:= max + g[i,t].f;

    for j:= 1 to n do

if g[i,j].f<>0 then writeln( i, '-> ' ,j,' ' ,g[i,j].f);

  end;

  writeln('the max stream=',max);

end;

procedure process;{求最大流的主過程}

var del:integer;

    success:boolean;

begin

  s:=1;t:=n;

  repeat

    success:=ford(del);

    if success then print

    else change(del);

  until success;

end;

begin {main program}

  init;

  process;

end.

測試資料檔案(zdl.txt)如下:

6

0 3 5 0 0 0

0 0 1 4 0 0

0 0 0 0 2 0

0 0 0 0 0 5

0 1 0 3 0 2

0 0 0 0 0 0

運行結果如下:

graph file=zdl.txt

1->2  3

1->3  2

2->4  3

3->5  2

4->6  3

5->6  2

the max stream=5

**最小費用最大流及演算法**

上面的網路，可看作為輔送一般貨物的運輸網路，此時，最大流問題僅表明運輸網路運輸貨物的能力，但沒有考慮運送貨物的費用。在實際問題中，運送同樣數量貨物的運輸方案可能有多個，因此從中找一個輸出費用最小的的方案是一個很重要的問題，這就是最小代價流所要討論的內容。

1.最小費用最大流問題的模型

給定網路n=(v，e，c，w，s，t)，每一弧(vi,vj)屬於e上，除了已給容量cij外，還給了一個單位流量的費用w(vi,vj)≥o(簡記為wij)。所謂最小費用最大流問題就是要求一個最大流f，使得流的總輸送費用取最小值。

w(f)=∑wijfij

2.用對偶法解最小費用最大流問題

定義：已知網路n=(v，e，c，w，s，t)，f是n上的一個可行流，p為vs到vt(關於流f)的可增廣路徑，稱w(p)=∑wij（p+）-∑wij（p-）為路徑p的費用。

若p\*是從vs到vt所有可增廣路徑中費用最小的路徑，則稱p\*為最小費用可增廣路徑。

定理：若f是流量為v(f)的最小費用流，p是關於f的從vs到vt的一條最小費用可增廣路徑，則f經過p調整流量q得到新的可行流f’(記為f’=f+q)，一定是流量為v(f)+q的可行流中的最小費用流。

3.對偶法的基本思路

①取f={0}

②尋找從vs到vt的一條最小費用可增廣路徑p。

若不存在p，則f為n中的最小費用最大流，演算法結束。

若存在p，則用求最大流的方法將f調整成f\*，使v(f\*)=v(f)+q，並將f\*賦值給f，轉②。

4.反覆運算法求最小費用可增廣路徑

在前一節中，我們已經知道了最大流的求法。在最小費用最大流的求解中，每次要找一條最小費用的增廣路徑，這也是與最大流求法唯一不同之處。於是，對於求最小費用最大流問題餘下的問題是怎樣尋求關於f的最小費用增廣路徑p。為此，我們構造一個賦權有向圖b(f)，它的頂點是原網路n中的頂點，而把n中每一條弧(vi，vj)變成兩個相反方向的弧(vi，vj)和(vj，vi)。定義w(f)中的弧的權如下：

如果(fij<cij)，則bij=wij ；如果(fij=cij)，則bij=+oo

如果(fij>0)，則bji=-wij ；如果(fij=cij)，則bji=+oo

於是在網路n中找關於f的最小費用增廣路徑就等價于在賦權有向圖b(f)中，尋求從vs到vt的最短路徑。求最短路有三種經典演算法，它們分別是dijkstra演算法、floyd演算法和反覆運算演算法(ford演算法）。由於在本問題中，賦權有向圖b(f)中存在負權，故我們只能用後兩種方法求最短路，其中對於本問題最高效的演算法是反覆運算演算法。為了程式的實現方便，我們只要對原網路做適當的調整。將原網路中的每條弧(vi，vj)變成兩條相反的弧：

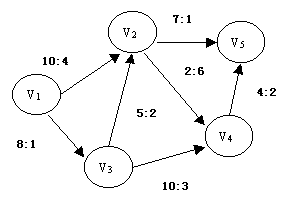
前向弧(vi,vj)，其容量cij和費用wij不變，流量為fij；

後向弧(vj,vi)，其容量0和費用-wij，流量為-fij。

事實上，對於原網路的資料結構中，這些單元已存在，在用標號法求最大流時是閒置的。這樣我們就不必產生關於流f的有向圖b(f)。同時對判斷(vi,vj)的流量可否改時，可以統一為判斷是否“fij<cij”。因為對於後向弧(vj，vi)，若fij>0，則fji=-fij<0=cji。

例2：求輸送量最大且費用最小的運輸方案？

如下圖是一公路網，v1是倉庫所在地（物資的起點），v5是某一工地（物資的終點）每條弧旁的兩個數位分別表示某一時間內通過該段路的最多噸數和每噸物資通過該段路的費用。



【**參考程式**】

program maxflow\_with\_mincost;

const

  name1='flow.in';

  name2='flow.out';

  maxn=100;{最多頂點數}

type

  tbest=record  {陣列的結構}

      value,fa:integer;

    end;{到源點的最短距離，父輩}

  nettype=record

    {網路鄰接矩陣類型}

    c,w,f:integer;

    {弧的容量，單位通過量費用，流量}

    end;

var

  net:array[1..maxn,1..maxn] of nettype;

     {網路n的鄰接矩陣}

  n,s,t:integer;{頂點數，源點、匯點的編號}

  minw:integer;{最小總費用}

  best:array[1..maxn] of tbest;{求最短路時用的陣列}

procedure init;{數據讀人}

var inf:text;

    a,b,c,d:integer;

begin

  fillchar(net,sizeof(net),0);

  minw:=0;

  assign(inf,name1);

  reset(inf);

  readln(inf,n);

  s:=1;t:=n;{源點為1，匯點n}

  repeat

    readln(inf,a,b,c,d);

    if a+b+c+d>0 then

    begin

    net[a,b].c:=c;{給弧(a,b)賦容量c}

    net[b,a].c:=0;{給相反弧(b,a)賦容量0}

    net[a,b].w:=d;{給弧(a,b)賦費用d}

    net[b,a].w:=-d;{給相反孤(b，a)賦費用-d}

    end;

  until a+b+c+d=0;

  close(inf);

end;

function find\_path:boolean;

{求最小費用增廣路函數,若best[t].value<>maxint,

則找到增廣路，函數返回true，否則返回false}

var quit:boolean;

    i,j:integer;

begin

  for i:=1 to n do best[i].value:=maxint;best[s].value:=0;

  {尋求最小費用增廣路徑前，給best陣列賦初值}

  repeat

    quit:=true;

    for i:=1 to n do

      if best[i].value < maxint then

        for j:=1 to n do

          if (net[i,j].f < net[i,j].c) and

          (best[i].value + net[i,j].w <

          best[j].value) then

          begin

            best[j].value:=best[i].value + net[i,j].w;

            best[j].fa := i;

            quit:= false;

          end;

  until quit;

  if best[t].value<maxint then find\_path:=true else find\_path:=false;

end;

procedure add\_path;

var i,j: integer;

begin

  i:= t;

  while i <> s do

    begin

      j := best[i].fa;

      inc(net[j,i].f); {增廣路是弧的數量修改，增量1}

      net[i,j].f:=-net[j,i].f;{給對應相反弧的流量賦值-f}

      i:=j;

    end;

    inc(minw,best[t].value); {修改最小總費用的值}

end;

procedure out;{輸出最小費用最大流的費用及方案}

var ouf: text;

    i,j: integer;

begin

  assign(ouf,name2);

  rewrite(ouf);

  writeln(ouf,'min w = ',minw);

  for i := 1 to n do

    for j:= 1 to n do

      if net[i,j].f > 0 then

        writeln(ouf,i, ' -> ',j,': ',

         net[i,j].w,'\*',net[i,j].f);

   close(ouf);

end;

begin {主程序}

   init;

   while find\_path do add\_path;

   {當找到最小費用增廣路，修改流}

   out;

end.

flow.in如下：

5

1 2 10 4

1 3 8 1

2 4 2 6

2 5 7 1

3 2 5 2

3 4 10 3

4 5 4 2

0 0 0 0

運行結果flow.out如下：

min w = 55

1 -> 2: 4\*3

1 -> 3: 1\*8

2 -> 5: 1\*7

3 -> 2: 2\*4

3 -> 4: 3\*4

4 -> 5: 2\*4

**【練習】**

1、平面上有n個點（n<=100），每個點的座標均在－1000~1000之間，其中的一些點之間有連線。若有連線，則表示可從一個點到達另一個點，即兩點間有通路，通路的距離為兩點間的直線距離。現在的任務是找出一點到另一點之間的最短路徑。

輸入：輸入檔為short.in，共n+m+3行，其中：第一行為整數n。第二行到第n+1行（共n行），每行兩個整數x和y，描述了一個點的座標（以一個空格分隔）。第n+2行為一個整數m，表示圖中連線的個數。此後的m行，每行描述一條連線，由兩個整數i和j組成，表示第i個點和第j個點之間有連線。最後一行：兩個整數s和t，分別表示源點和目標點。

輸出：輸出檔為short.out，僅一行，一個實數（保留兩位小數），表示s從t到的最短路徑長度。

樣例輸入：

5

0 0

2 0

2 2

1. 2
2. 1

5

1 2

1 3

1 4

2 5

3 5

1 5

樣例輸出：

3.41

2、某工廠發現廠裡的機器在生產產品時需要消耗大量的原材料,現在廠裡想找出消耗原材料最大的一條生產線路加以改進以降低成本。已知廠裡的生產線路是一個有向的無環網路，有n個機器分別代表網路中的n 個結點，弧〈i，j〉（i〈j）表示原材料從機器i傳到機器j的損耗數量。

輸入檔：第一行兩個整數n，m（n〈=100，m〈=100）分別表示網路中的結點的個數與弧數，第二行至m+1行，每行三個整數a,b,c，表示弧〈a,b〉的損耗為c。

  輸出檔：只一個整數，損耗最大線路的損耗量。

3、

**第五章 資料排序及其複雜度**

**排序**是電腦程式設計中的一種重要運算。它的功能是將一個資料元素（或記錄）的任意序列，重新排列成一個按關鍵字有序的序列。

假設含有n個記錄的序列為 {R1，R2，…，Rn}

其相應的關鍵字序列為 {K1，K2，…，Kn}

上述排序中定義的關鍵字Ki可以是記錄Ri（i=1,2,…,n）的主關鍵字，也可以是記錄Ri的次要索引機碼，甚至是若干資料項目的組合。若Ki是主關鍵字，則任何一個記錄的無序序列經排序後得到的結果是唯一的；若Ki是次要索引機碼，則排序的結果是不唯一的；因為待排序的記錄序列中可能存在兩個或兩個以上關鍵字相等的記錄。假設Ki=Kj(1≤i≤n,1≤j≤n,i≠j)，且在排序前的序列中Ri領先於Rj（即i<j），若在排序後的序列中Ri仍領先於Rj，則稱所用的排序方法是穩定的，反之則是不穩定的。

5．1 冒泡排序

【**基本思想**】　大數下沉，小數上冒的一種排序，一趟排好一個，共進行n-1次

【**參考程式**】

program mppx;

const n=7;

var a:array[1..n] of integer;

    i,j,k,temp:integer;

begin

   for i:= 1 to n do read(a[i]);

for i:= 1 to n-1 do

for j:= i+1 to n do

if a[i]<a[j] then

begin temp:=a[j];a[j]:=a[i];a[i]:=temp;end;

     for i:= 1 to n do write(a[i]:6);

     writeln;

end.

5．2 選擇排序

【**基本思想**】　對待排序的序列進行n-1遍的處理，第1遍處理是將A[1..n]中最小者與A[1]交換位置，第2遍處理是將A[2..n]中最小者與A[2]交換位置，......，第i遍處理是將A[i..n]中最小者與A[i]交換位置。這樣，經過i遍處理之後，前i個數的位置就已經按從小到大的順序排列好了。

【**示例**】

24 56 48 36 12 92 86 34

i=1:[12][56 48 36 24 92 86 34]

i=2:[12 24] [48 36 56 92 86 34]

【**參考程式**】

program xzpx;

const n=7;

var a:array[1..n] of integer;

i,j,k,t:integer;

begin

for i:= 1 to n do read(a[i]);

for i:=1 to n-1 do

begin

k:=i;

for j:=i+1 to n do{尋找最小元素的位置}

if a[j]<a[k] then k:=j;

if k<>i then

begin t:=a[i];a[i]:=a[k];a[k]:=t;end;

end;

for i:= 1 to n do write(a[i]:6);

writeln;

end.

5．3 插入排序

【**基本思想**】　經過i-1遍處理後,A[1..i-1]己排好序。第i遍處理僅將A[i]插入A[1..i-1]的適當位置p，原來p後的元素一一向右移動一個位置,使得A[1..i]又是排好序的序列。

【**示例**】

24 56 48 36 12 92 86 34

第一趟 [24 56]

第二趟 [24 48 56]

……

【**參考程式**】

program crpx;

const n=7;

var a:array[1..n] of integer;

i,j,k,t:integer;

begin

for i:= 1 to n do read(a[i]);

for i:=2 to n do

begin

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

while (k<a[j]) and (j>0) do

begin a[j+1]:=a[j];j:=j-1 end;

{尋找要插入的位置，該位置後的元素向右移動一個位置}

a[j+1]:=k;{插入該位置}

end;

write('output data:');

for i:= 1 to n do write(a[i]:6);

end.

5．4 快速排序

【**基本思想**】　是冒泡排序的一種改進，通過部分排序來完成整個資料序列的排序，先從資料序列中選一個元素，並將序列中所有比該元素小的元素都放到它左邊(右邊)，將所有比該元素大的元素都放到其右邊(左邊)，這樣就為該元素在表中找到了一個恰當的位置，然後再以同樣的方法處理兩邊……(這裡用到遞迴)。

【**示例**】

i j

初始關鍵字 [49 38 65 97 76 13 27 49］

j向左掃描

第一次交換後 ［27 38 65 97 76 13 49 49］

i向右掃描

第二次交換後 ［27 38 49 97 76 13 65 49］

j向左掃描，位置不變，第三次交換後 ［27 38 13 97 76 49 65 49］

i向右掃描，位置不變，第四次交換後 ［27 38 13 49 76 97 65 49］

j向左掃描 ［27 38 13 49 76 97 65 49］

（以上為一次劃分過程）

初始關鍵字 ［49 38 65 97 76 13 27 49］

一趟排序之後 ［27 38 13］ 49 ［76 97 65 49］

二趟排序之後 ［13］ 27 ［38］ 49 ［49 65］76 ［97］

三趟排序之後 13 27 38 49 49 ［65］76 97

最後的排序結果 13 27 38 49 49 65 76 97

【**參考程式**】

program kspx;

const n=8;

type

arr=array[1..n] of integer;

var

a:arr;

i:integer;

procedure quicksort(var b:arr; left,right:integer);

var i,j,x,t1:integer;

begin

i:=left;j:=right;x:=b[i];

repeat

while (b[j]>=x) and (j>i) do j:=j-1;

if j>i then begin t1:=b[i]; b[i]:=b[j];b[j]:=t1;end;

while (b[i]<=x) and (i<j) do i:=i+1;

if i<j then begin t1:=b[j];b[j]:=b[i];b[i]:=t1; end

until i=j;

b[i]:=x;

i:=i+1;j:=j-1;

if left<j then quicksort(b,left,j);

if i<right then quicksort(b,i,right);

end;

begin

for i:=1 to n do read(a[i]);

quicksort(a,1,n);

for i:=1 to n do write(a[i]:6);

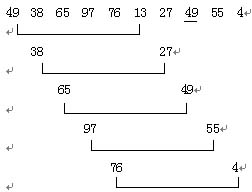
end.

5．5 希爾排序

【**基本思想**】　將整個無序序列分割成若干小的子序列分別進行插入排序或冒泡排序。

序列分割方法：將相隔某個增量d的元素構成一個子序列。在排序過程中，逐次減小這個增量，最後當d減到1時，進行一次插入排序或冒泡排序，排序就完成。增量序列一般採用：d1=n div 2 ,di=di-1 div 2 ；i=2,3,4.....其中n為待排序序列的長度。

【示例】



一趟排序結果：13 27 49 55 4 49 38 65 97 76

【**參考程式1**】{子序列是插入排序}

program shellins;

const n=10;

type

arr=array[1..n] of integer;

var

a:arr;

i,j,t,d:integer;

bool:boolean;

begin

write('input data:');

for i:=1 to n do read(a[i]);

writeln;

d:=n;

while d>1 do

  begin

  d:=d div 2;

  for j:=d+1 to n do

    begin

    t:=a[j];i:=j-d;

    while (i>0) and (a[i]>t) do

       begin a[i+d]:=a[i];i:=i-d;end;

    a[i+d]:=t;

    end;

  end;

  write('output data:');

for i:=1 to n do write(a[i]:6);

writeln;

end.

【**參考程式2**】 {子序列是冒泡排序}

program shellins;

const n=10;

type

arr=array[1..n] of integer;

var

a:arr;

i,temp,d:integer;

bool:boolean;

begin

write('input data:');

for i:=1 to n do read(a[i]);

writeln;

d:=n;

while d>1 do

  begin

  d:=d div 2;

  repeat

   bool:=true;

   for i:=1 to n-d do

     if a[i]>a[i+d] then

     begin temp:=a[i];a[i]:=a[i+d];a[i+d]:=temp; bool:=false end;

  until bool;

  end;

write('output data:');

for i:=1 to n do write(a[i]:6);

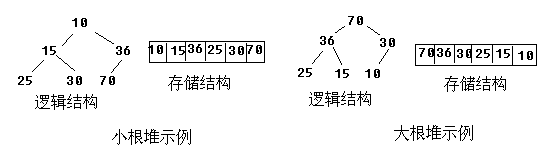
writeln;

end.

5．6 堆排序與二叉樹排序

**堆**的定義: N個元素的序列K1,K2,K3,...,Kn.稱為堆，當且僅當該序列滿足如下特性：

6



堆實質上是滿足如下性質的完全二叉樹，可用一維陣列連續存儲：

（1）樹中任一非葉子結點的關鍵字均小於等於其孩子結點的關鍵字。例如序列10,15,36,25,30,70就是一個堆，它對應的完全二叉樹如上圖所示。這種堆中根結點（稱為堆頂）的關鍵字最小，我們把它稱為小根堆。反之，若完全二叉樹中任一非葉子結點的關鍵字均大於等於其孩子的關鍵字，則稱之為大根堆。

（2）在堆對應的完全二叉樹中，若非終端結點的位址為K，則它的父親結點的位址為K/2，它的左兒子結點的位址為2K，右兒子結點的地址為2K+1。

（3）堆頂元素（即根結點）必為序列中n個元素的最小值（或最大值）。

【**堆排序的思想**】

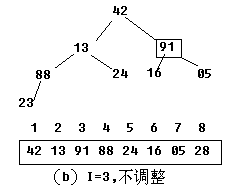
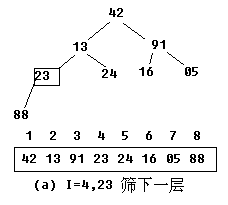
（1）建初始堆（將結點**[n/2],[ n/2]-1,...3,2,1**分別調成堆）

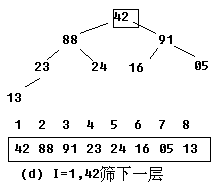
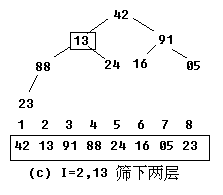
（2）當未排序完時，輸出堆頂元素，刪除堆頂元素，將剩餘的元素重新建堆。

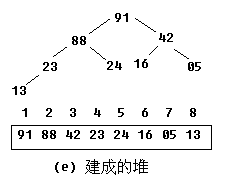
【**演算法實現**】　用“篩選法”來調整堆

以序列42，13，91，23，24，16，05，88建一個大根堆為例，資料存放在a[1..n]陣列中。

（1）如何建初始堆：我們可以這樣理解，首先所有葉子結點（編號為**trunc(n/2)+1到n**）都各自成堆，我們只要從最後一個分支結點(編號為**trunc(n/2))**開始，不斷“調整”每個分支結點與孩子結點的值，使它們滿足堆的要求，直到根結點為止，這樣一定能確保根（堆頂元素）的值最大。“調整”的思想如下：即如果當前結點編號i為，則它的左孩子為2\*i，右孩子為2\*i+1，首先比較a[i]與max(a[2\*i],a[2\*i+1])；如果a[i]大，說明以結點i為根的子樹已經是堆，不用再調整。否則將結點i和左右孩子中值大的那個結點j互換位置，互換後可能破壞以j為根的堆，所以必須再比較a[j]與max(a[2\*j],a[2\*j+1])，依此類推，直到父結點的值大於等於兩個孩子或出現葉結點為止。這樣以i為根的子樹就被調整成為一個堆。







（2）經過第一步驟建立好一個初始堆後，可以確定堆頂元素值最大，我們就把它（a[1]）與最後一個元素(a[n])交換，然後再對a[1..n-1]進行調整，得到次大值與a[n-1]交換，如此下去，所有元素便有序存放了。

【**參考程式**】

program duipx;{建立小根堆，從小到大輸出}

const n=8;

type arr=array[1..n] of integer;

var a:arr;i:integer;

procedure heap(var a:arr;head,tail:integer);

  var i,j, temp:integer;

  begin

  i:=head;j:=2\*i;temp:=a[i];

  while j<=tail do

  begin

   if (j<tail) and (a[j]>a[j+1]) then j:=j+1;{尋找左右孩子中較小者}

   if temp>a[j] then

      begin a[i]:=a[j];i:=j;j:=2\*i; end

          else exit;

  a[i]:=temp;

  end;

end;

begin

  for i:=1 to n do read(a[i]);

  for i:=(n div 2) downto 1 do

  heap(a,i,n);

  for i:=n downto 2 do

  begin

  write(a[1]:4);

  a[1]:=a[i];

  heap(a,1,i-1);

  end;

  writeln(a[1]:4);

end.

5．7 歸併排序

【**基本思想**】 就是將兩個或兩個以上的有序序列合成一個新的有序序列。將兩個有序序列合併為一個有序序列叫二路歸併(merge)，歸併排序的思想是把含有n個元素的序列看成n個有序的子序列，每個子序列的長度為1，然後兩兩歸併，……，如此重複，直至最後得到一個長度為n的有序序列為止。時間複雜度：O（m+n），假設兩個有序序列的長度分別為m和n。

位置： 1 2 3 4 5 6 7

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 49 | 38 | 65 | 97 | 76 | 13 | 27 |

原始資料：

數據： 長度1

49

38

65

97

76

13

27

38

49

65

97

13

76

38

49

65

97

13

27

76

13

27

38

49

65

76

97

數據： 長度2

數據： 長度4

數據： 長度7

program gbpx;

const maxn=7;

type arr=array[1..maxn] of integer;

var a,b,r2:arr;

i:integer;

procedure merge(r:arr;low,m,n:integer;var r2:arr);{已知r[low..m]和r[m+1..n]分別按關鍵字有序，將它們歸併為一個有序序列並存放在r2[low..n]}

var i,j,k,p:integer;

begin

i:=low;j:=m+1;k:=low-1;

while (i<=m) and (j<=n) do

begin

k:=k+1;

if r[i]<=r[j] then begin r2[k]:=r[i];i:=i+1 end

else begin r2[k]:=r[j];j:=j+1 end

end;

if i<=m then

for p:=i to m do begin k:=k+1;r2[k]:=r[p] end;

if j<=n then

for p:=j to n do begin k:=k+1;r2[k]:=r[p] end;

{將r[low..m]或r[m+1..n]中的剩餘序列複製到r2中}

end;

procedure mergesort( var r,r1:arr;s,t:integer);{對r[s..t]中序列進行歸併排序，使 r[s..t]中的序列按關鍵字有序}

var k:integer;

begin

if s=t then r1[s]:=r[s] else

begin

k:=(s+t) div 2;

mergesort(r,r2,s,k);

mergesort(r,r2,k+1,t);

merge(r2,s,k,t,r1)

end;

end;

begin

write('Enter data:');

for i:= 1 to maxn do read(a[i]);

mergesort(a,b,1,maxn);

for i:=1 to maxn do write(b[i]:9);

writeln;

end.

**【各種排序演算法的比較】**

（1）穩定性比較：

 插入排序、冒泡排序、二叉樹排序、歸併排序、基數排序是穩定的；選擇排序、快速排序、堆排序、希爾排序是不穩定的。

（2）時間複雜性比較

 插入排序、冒泡排序、選擇排序的時間複雜度為O(n2)，其它非線形排序的時間複雜度為O(nlogn)，希爾排隊序的時間複雜度為O(n3/2)。

（3）其它比較

插入、冒泡排序的速度較慢，但參加排序的序列局部或整體有序時，這種排序能達到較快的速度。反而在這種情況下，快速排序反而慢了。

當n較小時，對穩定性不作要求時宜用選擇排序，對穩定性有要求時宜用插入或冒泡排序。

當n較大時，關鍵字元素比較隨機，對穩定性沒要求宜用快速排序。

當n較大時，關鍵字元素可能出現本身是有序的，對穩定性沒有要求時宜用堆排序。

元素交換次數最少的排序為選擇排序，元素比較次數最少的排序為歸併排序。

(4) 逆序對

定義：對於一個包含N個非負整數的陣列A[1..n]，如果有i < j，且A[ i ]>A[ j ]，則稱( i , j )為陣列A中的一個逆序對。

例如，陣列（3，1，4，5，2）的逆序對有(3,1),(3,2),(4,2),(5,2)，共4個。

【**問題**】 如果給定一個長度為n的序列，每次可以交換相鄰的元素。求將通過交換使原序列變成不下降序列的最少交換次數？

【**解答**】 求序列的逆序對個數就是問題的解

【**證明**】 假設 a 和 b 相鄰，若 a<b 交換會增加逆序數，所以最好不要做此交換；若 a=b 交換無意 義，也不要進行此交換；a>b 時，交換會減少逆序，使序列更順序，所以做交換。

由上可知，所謂的移動只有一種情況，即a>b，且一次移動的結果是逆序減 1 。

假設初始逆序是 n， 每次移動減 1，那麼就需要 n 次移動時序列變為順序。所以題目轉化為直接求序列的逆序便可以了。

【**求解方法**】

方法一：最原始的方法，利用兩重迴圈進行枚舉。該演算法的時間複雜度為O(n^2)。

方法二：利用歸併排序的思想求解逆序對的個數，這是目前解決該問題的一種較為高效的演算法。該演算法的時間複雜度為O(nlogn)。

**第六章 高精度演算法**

由於電腦輸入計算結果的精度通常受到電腦的限制，如：在雙精度方式下，電腦最多只能輸出16位元有效數字，如果超過16位，則只能按浮點形式輸出，另外，一般電腦實數表示的範圍為1038，如果超過這個範圍，電腦就無法表示了。但是我們可以通過一些簡單的辦法來解決這個問題。這就是我們要說的高精度電腦。

**【基本方法】**

在電腦上進行高精度計算，首先要處理好以下幾個基本問題：

①、資料的接收與存儲；

②、計算結果位數的確定；

③、進位處理和借位處理；

④、商和餘數的求法；

下面我們逐一介紹一下這幾個問題的解決方法。

**【資料的接收與存儲】**

要在電腦上進行高精度計算，首先就應該有精確的輸入，即電腦要精確地接收和存儲資料。通常：

①、當輸入的數值在電腦允許的範圍內時，可以用數值型變數來接收資料。

②、當輸入的資料超過電腦允許顯示的精度範圍時，採用字元來接收資料。

③、分離各位數字。

**【計算結果位數的確定】**

①、兩數之和的位數最大為較大的數的位數加1。

②、乘積的位數最大為兩個因數的位數之和。

③、階乘與乘方的位數可以採用對數運算來確定計算結果的位數。

**【進位處理和借位處理】**

①、加法的進位處理

進行加法處理時，先設置一個加法進位元標誌T，並將T的初值設為0。當兩數相加時，從低位元到高位，各位數位分別相加，如果相加後某個單元中的數大於10，則將該單元中的數減去10，並將進位元標誌T設為1，當對下一單元進行相加時，還要再加上前一個單元的進位元標誌T。同時將T再次置為0，不斷重複，直到最高位為止。

具體演算法為：

T:=0;

對變數i從1到n，重複下列步驟：

c[i]:=a[i]+b[i]+T;T:=0;

if c[i]>=10 then begin

c[i]:=c[i]-10;T:=1;

end;

②、乘法的進位處理

y:=a[i]\*b[i]+T;T:=y div 10;c[i]:=y-T\*10

③、減法的借位處理

if a[i] <b[i] then

begin a[i+1]:=a[i+1]-1;a[i]:=a[i]+10;end;

c[i]:=a[i]-b[i];

**【實例6-1】 求任意位數的加法運算**

【**問題分析**】

①、資料的接收和存儲

採用字串輸入的方式，設參與運算的兩組數分別為A和B，利用字串函數把字串轉化為數值，將A、B中的每一位元數位分別存儲在A、B兩個陣列中，最低位元在第一個單元中。(PASCAL語言中可以直接採用字元讀取的方式來接收資料，而後通過ORD(x)-48的方式轉化成數值。)

②、確定和的位數

設LA為A的位數，LB為B的位數，則兩數之和的位數最大為較大加數位數加1，即如果LA>LB，則和的位數最大為LA+1。

③、進位處理

進行加法處理時，先設置一個加法進位元標誌 T，並將 T 的初值設為 0。當兩數相加時，從低位元到高位，各位數位分別相加，如果相加後某個單元中的數大於 10，則將該單元中的數減去10，並將進位元標誌 T 設為 1，當對下一單元進行相加時，還要再加上前一個單元的進位元標誌 T。同時將 T 再次置為 0，不斷重複，直到最高位為止。

【**參考程式**】

program add;

const n=100;

type arrtype=array[1..n] of integer;

var

a,b:arrtype;

t,s,j,l:integer;

procedure readdata(var int:arrtype);

var

ch:char;

i,k:integer;

begin

writeln('Input a number:');

read(ch);k:=0;

while ch in['0'..'9'] do

begin

inc(k);

int[k]:=ord(ch)-48;

read(ch);

end;

for i:=k downto 1 do begin

int[n+i-k]:=int[i];

int[i]:=0;

end;

end;

begin

readdata(a);readln;

readdata(b);writeln;

t:=0;

for j:=n downto 1 do begin

s:=a[j]+b[j]+t;

a[j]:=s mod 10;

t:=s div 10;

end;

j:=1;writeln('output:');

while a[j]=0 do j:=j+1;

while j<=n do begin

write(a[j]);

inc(j);

end;

writeln;

end.

**【實例6-2】 求任意位數的減法運算**

①、資料的接收和存儲

採用字串輸入的方式，設參與運算的兩個數分別為A和B，利用字串函數把字串轉化為數值，將A、B中的每一位元數位分別存儲在A、B兩個陣列中，最低位元在第一個單元中。(PASCAL語言中可以直接採用字元讀取的方式來接收資料，而後通過ORD(x)-48的方式轉化成數值。)

②、確定差的位數

設LA為A的位數，LB為B的位數，則兩數之差的位數最大為較大數的位數，即如果LA>LB，則差的位數最大為LA。

③、借位處理

做減法運算時，要先判斷是否需要借位，如果需要借位，從上一位借過一個10，上一位的數減去1，處理完之後再相減。

④、負數的處理

如果減數大於被減數，則交換A、B的值，並令負數標誌 T=-1。當列印計算結果時，先判斷 T 的值是否為-1，如果T=-1，則在數值前面先輸出一個負號。

【**參考程式**】

program subtration;

const n=100;

type arrtype=array[1..n] of integer;

var

a,b:arrtype;

g,t,j,l:integer;

s:char;

f:boolean;

procedure readdata(var int:arrtype);

var

ch:char;

i,k:integer;

begin

writeln('Input a number:');

read(ch);k:=0;

while ch in['0'..'9'] do begin

inc(k);

int[k]:=ord(ch)-48;

read(ch);

end;

for i:=k downto 1 do begin

int[n+i-k]:=int[i];

int[i]:=0;

end;

end;

begin

readdata(a);readln;

readdata(b);writeln;

f:=true;j:=1;

while (a[j]=b[j]) and (j=b[j]+g then begin

a[j]:=a[j]-b[j]-g;

g:=0;

end

else begin

a[j]:=10+a[j]-b[j]-g;

g:=1;

end;

j:=1;writeln('output:');write(s);

while (a[j]=0) and (j<=n do begin

write(a[j]);

inc(j);

end;

　writeln;

end.

**【實例6-3】 求任意位數的乘法運算**

①、資料的接收和存儲

採用字串輸入的方式，設參與運算的兩個數分別為A和B，利用字串函數把字串轉化為數值，將A、B中的每一位元數位分別存儲在A、B兩個陣列中，最低位元在第一個單元中。(PASCAL語言中可以直接採用字元讀取的方式來接收資料，而後通過ORD(x)-48的方式轉化成數值。)

②、確定積的位數

設LA為A的位數，LB為B的位數，乘積的位數最多為LA+LB，最少為LA+LB-1。所以，乘積的位數上限為LA+LB。

③、演算法

首先計算被乘數與乘數的個位數字的乘積，把結果保存到積陣列中，然後再用被乘數去乘以乘數的十位元數字，把結果退一位元加到積陣列中。每加一次乘積結果就進行一次進位處理，其方法與加法中的進位處理一樣。

【**參考程式**】

program multiplication;

const n=100;

type arrtype=array[0..n] of integer;

arrtype2=array[1..2\*n] of integer;

var

a,b,d:arrtype;

c:arrtype2;

g,i,j,s:integer;

procedure readdata(var int:arrtype);

var

ch:char;

i,k:integer;

begin

writeln('Input a number:');

read(ch);k:=0;

while ch in['0'..'9'] do begin

inc(k);

int[k]:=ord(ch)-48;

read(ch);

end;

for i:=k downto 1 do begin

int[n+i-k]:=int[i];

int[i]:=0;

end;

end;

begin

readdata(a);readln;

readdata(b);writeln;

for i:=n downto 1 do begin

g:=0;

for j:=n downto 1 do begin

s:=a[j]\*b[i]+g;

d[j]! :=s mod 10;

g:=s div 10;

end;

d[0]:=s;

g:=0;

for j:=n downto 0 do begin

s:=d[j]+c[i+j]+g;

c[i+j]:=s mod 10;

g:=s div 10

end;

end;

end.

**【實例6-4】 求A÷B的精確值**

由於A和B都是電腦允許的顯示精度，故A、B均可使用數值變數來接收與存儲。A÷B精確值有兩種情況：

①、A能被B整除，沒有餘數。

②、A不能被B整除，對餘數進行處理。首先，我們知道，在做除法運算時，有一個不變的量和三個變化的量，不變的量是除數，三個變化的量分別是：被除數、商和餘數。做除法運算時，每次都是用被除數減去商與除數的積，如果所得餘數不為零，則將其擴大10倍再次作為被除數，繼續試除，直至餘數為零或達到要求的精確度為止。最後，必須對精確度的後一位求商，然後判斷其值而四捨五入得出最後的結果。

**【參考程式】**

program divide

const n=100;

type arrtype=array[0..n] of integer;

var

a,b,c:arrtype;

i,j,k,l,m:integer;

begin

write('Input a number:');readln(m);

write('Input a number:');readln(k);

w! rite('Input a accuracy:');readln(l);

write(m,'/',k,'=');

a[0]:=m;

b[0]:=m div k;

c[0]:=m mod k;

if c[0]=0 then write(b[0])

else begin

for i:=1 to l+1 do begin

a[i]:=c[i-1]\*10;

b[i]:=a[i] div k;

c[i]:=a[i] mod k;

if c[i]=0 then i:=l+1;

end;

if b[l+1]>4 then b[l]:=b[l]+1;

for i:=l downto 1 do

if b[i]>=10 then begin

b[i]:=b[i]-10;

b[i-1]:=b[i-1]+1;

end

else

i:=1;

write(b[0],'.');

for i:=1 to l do write(b[i]);

end;

readln;

end.

**【實例6-5】 求高精度A÷B的商和餘數**

①、資料的接收和存儲

採用字串輸入的方式，設參與運算的兩個數分別為A和B，利用字串函數把字串轉化為數值，將A中的每一位元數位分別存儲在A陣列中，最低位元在第一個單元中。(PASCAL語言中可以直接採用字元讀取的方式來接收資料，而後通過ORD(x)-48的方式轉化成數值。)。B則可以用一般的整數變數來接收和存儲。

②、演算法

首先，我們知道，在做除法運算時，有一個不變的量和三個變化的量，不變的量是除數，三個變化的量分別是：被除數、商和餘數。做除法運算時，每次都是用被除數減去商與除數的積，如果所得餘數不為零，則將其擴大10倍再次作為被除數，繼續試除，直至餘數為零或達到要求的精確度為止。

【**參考程式**】

program div1;

const n=100;

type arrtype=array[0..n] of integer;

var

a:arrtype;

g,b,i,j,s:integer;

procedure readdata(var int:arrtype);

var

ch:char;

i,k:integer;

begin

write('Input a number:');

read(ch);k:=0;

while ch in['0'..'9'] do begin

inc(k);

int[k]:=ord(ch)-48;

read(ch);

end;

for i:=k downto 1 do begin

int[n+i-k]:=int[i];

int[i]:=0;

end;

end;

begin

readdata(a);writeln;

write('Input the numbr b:');

readln(b);writeln;

g:=0;

for i:=1 to n do begin

s:=a[i]+g\*10;

a[i]:=s div b;

g:=s mod b;

end;

j:=1;writeln('output:');

while (a[j]=0) and (j<=n do begin

write(a[j]);

inc(j);

end;

if g<>0 then writeln('......',g);

writeln;

end.

**【實例6-6】 求高精度A÷高精度B的商和餘數**

①、資料的接收和存儲

採用字串輸入的方式，設參與運算的兩個數分別為A和B，利用字串函數把字串轉化為數值，將A、B中的每一位元數位分別存儲在A和B陣列中，最低位元在第一個單元中。(PASCAL語言中可以直接採用字元讀取的方式來接收資料，而後通過ORD(x)-48的方式轉化成數值。)。

②、演算法

可以用減法代替除法運算：不斷比較A[1..n]與B[1..n]的大小，如果A[1..n]>=B[1..n]，商C[1..n]+1→C[1..n]，然後就是一個減法過程：A[1..n]-B[1..n]→A[1..n]。由於簡單的減法速度太慢，故必須進行優化。設置一個位置值J，當A[1..n]>B[1..n]時。B[1..n]左移→B[0..n]，j:=j+1，即令B[1..n]增大10倍。這樣就減少了減法的次數。當j>0且A[1..n]<B[0..n]時，B[0..n]右移→B[0..n]，j:=j-1，即令B[1..n]縮小10倍。

【**參考程式**】

Program div2;

const n=100;

type arrtype=array[1..n] of integer;

var

a,b,c:arrtype;

g,s,i,j,k,l:integer;

procedure readdata(var int:arrtype);

var

ch:char;

i,k:integer;

begin

write('Input a number:');

read(ch);k:=0;

while ch in['0'..'9'] do begin

inc(k);

int[k]:=ord(ch)-48;

read(ch);

end;

for i:=k downto 1 do begin

int[n+i-k]:=int[i];

int[i]:=0;

end;

end;

function f(a,b:arrtype):boolean;

var j:integer;

begin

f:=true;j:=1;

while (a[j]=b[j]) and (j=b[i]+g then

begin

a[i]:=a[i]-b[i]-g;

g:=0;

end

else

begin

a[i]:=10+a[i]-b[i]-g;

g:=1;

end;

end;

procedure ine(n:integer);

var i,s:integer;

begin

g:=0;

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

for i:=n downto 1 do begin

s:=c[i]+g;

c [i]:=s mod 10;

g:=s div 10;

end;

end;

begin

readdata(a);readln;

readdata(b);writeln;

j:=1;

while f(a,b) do begin

j:=j+1;

for i:=2 to n do b[i-1]:=b[i];

b[n]:=0;

end;

while j>0 do begin

while f(a,b) do begin

ine(n-j+1);

end;

end;

j:=j-1;

for i:=n downto 2 do begin

b[i]:=b[i-1];

b[i-1]:=0;

end;

end;

j:=1;writeln('output:');

while (c[j]=0) and (j<=n do begin

write(c[j]);

inc(j);

end;

j:=1;

while (a[j]=0) and (j<=n) do j:=j+1;

if j<=n then write('......');

while j<=n do begin

write(a[j]);

inc(j);

end;

writeln;

end.

**第七章 窮舉法**

【**窮舉法**】（窮舉法）是基於電腦的特點而進行解題的思維方法。一般是在一時找不出解決問題的更好途徑（即從數學上找不到求解的公式或規則）時，可以根據問題中的部分條件（約束條件）將所有可能解的情況列舉出，然後通過一一驗證是否符合整個問題的求解要求，而得到問題的解。這種解決問題的方法我們稱之為窮舉演算法。

窮舉演算法模式：

（1）題解的可能搜索的範圍：用迴圈或迴圈嵌套結構實現；

（2）寫出符合問題解的條件；

（3）能使程式優化的語句，以便縮小搜索範圍，減少程式運行的時間。

下面我們就從窮舉演算法的的優化、窮舉物件的選擇以及判定條件的確定，這三個方面來探討如何用窮舉法解題。

【**例7-1**】　百錢買百雞問題：有一個人有一百塊錢，打算買一百隻雞。到市場一看，大雞三塊錢一隻，小雞一塊錢三隻，不大不小的雞兩塊錢一隻。現在，請你編一程式，幫他計畫一下，怎麼樣買法，才能剛好用一百塊錢買一百隻雞？

【**演算法分析**】　此題很顯然是用窮舉法，我們以三種雞的個數為窮舉物件（分別設為x,y,z）,以三種雞的總數（x+y+z）和買雞用去的錢的總數(x\*3+y\*2+z)為判定條件，窮舉各種雞的個數。

下面是解這個百雞問題的程式

var x,y,z:integer;

begin

for x:=0 to 100 do

for y:=0 to 100 do

for z:=0 to 100 do{窮舉所有可能的解}

if (x+y+z=100)and(x\*3+y\*2+z div 3=100)and(z mod 3=0)then writeln('x=',x,'y=',y,'z=',z); {驗證可能的解，並輸出符合題目要求的解}

end.

上面的條件還有優化的空間，三種雞的和是固定的，我們只要窮舉二種雞（x,y），第三種雞就可以根據約束條件求得（z=100-x-y），這樣就縮小了窮舉範圍，請看下面的程式：

var x,y,z:integer;

begin

for x:=0 to 100 do

for y:=0 to 100-x do

begin

　　　z:=100-x-y;

　　　if (x\*3+y\*2+z div 3=100)and(z mod 3=0)then writeln('x=',x,'y=',y,'z=',z);

end;

end.

未經優化的程式迴圈了1013 次，時間複雜度為O(n3)；優化後的程式只迴圈了（102\*101/2）次 ，時間複雜度為O(n2)。從上面的對比可以看出，對於窮舉演算法，加強約束條件，縮小窮舉的範圍，是程式優化的主要考慮方向。

在窮舉演算法中，窮舉物件的選擇也是非常重要的，它直接影響著演算法的時間複雜度，選擇適當的窮舉物件可以獲得更高的效率。如下例：

【**例7-2**】　將1,2...9共9個數分成三組,分別組成三個三位數,且使這三個三位數構成1:2:3的比例,試求出所有滿足條件的三個三位數.

例如:三個三位數192,384,576滿足以上條件.(NOIP1998pj)

【**演算法分析**】　這是1998年全國分區聯賽普及組試題（簡稱NOIP1998pj，以下同）。此題資料規模不大，可以進行窮舉，如果我們不加思地以每一個數位為窮舉物件，一位元一位元地去窮舉：

for a:=1 to 9 do

for b:=1 to 9 do

………

for i:=1 to 9 do

這樣下去，窮舉次數就有9９次，如果我們分別設三個數為x,2x,3x，以x為窮舉對象，窮舉的範圍就減少為９３，在細節上再進一步優化，窮舉範圍就更少了。程式如下：

var

t,x:integer;

s,st:string;

c:char;

begin

for x:=123 to 321 do{窮舉所有可能的解}

　begin

　t:=0;

　str(x,st);{把整數x轉化為字串，存放在st中}

　str(x\*2,s); st:=st+s;

　str(x\*3,s); st:=st+s;

　for c:='1' to '9' do{窮舉9個字元，判斷是否都在st中}

　if pos(c,st)<>0 then inc(t) else break;{如果不在st中，則退出迴圈}

if t=9 then writeln(x,' ',x\*2,' ',x\*3);

　end;

end.

在窮舉法解題中，判定條件的確定也是很重要的，如果約束條件不對或者不全面，就窮舉不出正確的結果， 我們再看看下面的例子。

【**例7-3**】　 一元三次方程求解(noip2001tg)

【問題描述】 有形如：ax3+bx2+cx+d=0 這樣的一個一元三次方程。給出該方程中各項的係數(a，b，c，d 均為實數)，並約定該方程存在三個不同實根(根的範圍在-100至100之間)，且根與根之差的絕對值>=1。

要求由小到大依次在同一行輸出這三個實根(根與根之間留有空格)，並精確到小數點後2位。

提示：記方程f(x)=0，若存在2個數x1和x2，且x1<x2，f(x1)\*(x2)<0，則在(x1，x2)之間一定有一個根。

【**樣例輸入**】　1 -5 -4 20

【**樣例輸出**】　-2.00 2.00 5.00

【**演算法分析**】　由題目的提示很符合二分法求解的原理，所以此題可以用二分法。用二分法解題相對於窮舉法來說要複雜很多。此題是否能用窮舉法求解呢？再分析一下題目，根的範圍在-100到100之間，結果只要保留兩位小數，我們不妨將根的值域擴大100倍（-10000<=x<=10000），再以根為窮舉對象，窮舉範圍是-10000到10000，用原方程式進行一一驗證，找出方程的解。

有的同學在比賽中是這樣做：

var

k:integer;

a,b,c,d,x :real;

begin

read(a,b,c,d);

for k:=-10000 to 10000 do

begin

x:=k/100;

if a\*x\*x\*x+b\*x\*x+c\*x+d=0 then write(x:0:2,' ');

end;

end.

用這種方法，很快就可以把程式編出來，再將樣例資料代入測試也是對的，等成績下來才發現這題沒有全對，只得了一半的分。

這種解法為什麼是錯的呢？錯在哪裡？前面的分析好象也沒錯啊，難道這題不能用窮舉法做嗎？ 看到這裡大家可能有點迷惑了。

在上面的解法中，窮舉範圍和窮舉對象都沒有錯，而是在驗證窮舉結果時，判定條件用錯了。因為要保留二位小數，所以求出來的解不一定是方程的精確根，再代入ax3+bx2+cx+d中，所得的結果也就不一定等於0，因此用原方程ax3+bx2+cx+d=0作為判斷條件是不準確的。

我們換一個角度來思考問題，設f(x)=ax3+bx2+cx+d，若x為方程的根，則根據提示可知，必有f(x-0.005)\*f(x+0.005)<0，如果我們以此為窮舉判定條件，問題就逆刃而解。另外，如果f(x-0.005)=0，哪麼就說明x-0.005是方程的根，這時根據四舍5入，方程的根也為x。所以我們用f（x-0.05）\*f（x+0.05）<0和 (f(x-0.005)=0)作為判定條件。為了程式設計的方便，我們設計一個函數f(x)計算ax3+bx2+cx+d的值，程式如下：

var

k:integer;

a,b,c,d,x:extended;

function f(x:extended):extended; {計算ax3+bx2+cx+d的值}

begin

　　f:=((a\*x+b)\*x+c)\*x+d;

end;

begin

　read(a,b,c,d);

　　for k:=-10000 to 10000 do

　　begin

x:=k/100;

　if (f(x-0.005)\*f(x+0.005)<0) or (f(x-0.005)=0) then write(x:0:2,' ');

{若x兩端的函數值異號或x-0.005剛好是方程的根，則確定x為方程的根}

　　end;

end.

【**例7-4**】　古希臘人認為因數的和等於它本身的數是一個完全數（自身因數除外），例如28的因數是1、2、4、7、14，且1+2+4+7+14=28，則28是一個完全數，編寫一個程式求2~1000內所有的完全數。

【**演算法分析**】

①、本題是一個搜索問題，搜索範圍2~1000，找出該範圍內的完全數；

②、完全數必須滿足的條件：因數的和等於該數的本身；

③、問題關鍵在於將該數的因數一一尋找出來，並求出因數的和：分解因數的方法比較簡單，採用迴圈完成來分解因數並求因數的和。

for a:=2 to 1000 do

begin

s:=0;

　for b:=1 to a-1 do

　　if a mod b =0 then s:=s+b;{分解因數並求和}

if a=s then begin write(a,’=’,1,);

for b:=2 to a-1 do

　　if a mod b=0 then write(‘+’,b);

writeln;

end.

【**例7-5**】　郵局發行一套票面有四種不同值的郵票，如果每封信所貼郵票張數不超過三枚，存在整數r，使得用不超過三枚的郵票，可以貼出連續的整數1、2、3、…、r來，找出這四種面值數，使得r值最大。

【**演算法分析**】

①、面值不同的四種郵票，每封信所貼郵票不超過3張；

②、用這四種郵票貼出連續的整數，並且使r值最大；

③、用窮舉法，找出所有符合條件的解；

④、本題用集合的方法統計郵票的面值，提高判斷重複的速度。{程式優化}

設四種郵票的面值分別為：a、b、c、d，根據題意設：

a<b<c<d，因此a=1，用迴圈語句完成搜索。

【**參考程式**】

program stamp;

var a,b,c,d:integer;

x,x0,x1,x2,x3,x4:integer;

st1:set of 1..100;

function number(a,b,c,d:integer):integer;

var n1,n2,n3,n4,sum:integer;

begin

st1:=[];

for n1:=0 to 3 do

for n2:=0 to 3-n1 do

for n3:=0 to 3-n1-n2 do

for n4:=0 to 3-n1-n2-n3 do

if n1+n2+n3+n4<=3 then

begin sum:=n1\*a+n2\*b+n3\*c+n4\*d;{計算信封的郵票面值}

st1:=st1+[sum];end;{利用集合判斷重複}

sum:=1;

while sum in st1 do

sum:=sum+1;

number:=sum-1;

end;

begin{main}

a:=1;x0:=0;

for b:=a+1 to 3\*a+1 do

for c:=b+1 to 3\*b+1 do{每種郵票的可取值的範圍}

for d:=c+1 to 3\*c+1 do

begin

x:=number(a,b,c,d);{調用函數求每封信的郵票總面值}

if x>x0 then

begin

x0:=x;x1:=a;x2:=b;x3:=c;x4:=d;{保存最大面值郵票}

write(x1:5,x2:5,x3:5,x4:5);

writeln(‘’:10,’x0=’,x0);

end;

end;

end.

用窮舉法解題的最大的缺點是運算量比較大，解題效率不高，如果窮舉範圍太大（一般以不超過兩百萬次為限），在時間上就難以承受。但窮舉演算法的思路簡單，程式編寫和調試方便，比賽時也容易想到，在競賽中，時間是有限的，我們競賽的最終目標就是求出問題解，因此，如果題目的規模不是很大，在規定的時間與空間限制內能夠求出解，那麼我們最好是採用窮舉法，而不需太在意是否還有更快的演算法，這樣可以使你有更多的時間去解答其他難題。

【**練習**】

【**練習7-1**】鈔票換硬幣，把一元鈔票換成一分、二分、五分硬幣（每種至少一枚），有哪些種換法？

【**練習7-2**】發獎品問題：學期末班裡評出5位三好生。老師要將5件互不相同的獎品發給這5位同學。一共有哪幾種不同的發法？

**第八章 遞推、遞迴**

【**遞推**】

遞推關係是一種簡潔高效的常見數學模型，比如我們熟悉的Fibonacci數列問題，F（1）=0，F（2）=1，在n>2時有：F（n）=F（n-1）+F（n-2）。在這種類型的問題中，每個資料都和它前面的若干個資料項目（或後面的若干個資料項目）有一定的關聯，這種關聯一般是通過一個遞推關係式來表示的。求解問題時我們就從初始的一個或若干個資料項目出發，通過遞推關係式逐步推進，從而得到最終結果。這種求解問題的方法叫“遞推法”。其中，初始的若干資料項目稱為“邊界”。

用遞推法求解Fibonacci數列問題的參考程式如下：

var f0,f1,f2:real;

    i,n:byte;

begin

     readln(n);

     f0:=1;f1:=2;

     for i:=2 to n do

     begin

          f2:=f0+f1;

          f0:=f1;

          f1:=f2

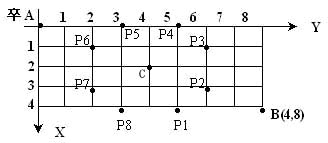
     end;

     writeln(f2:1:0)

end.

【**例8-1**】 過河卒問題

棋盤上A點有一個過河卒，需要走到目標B點。卒行走的規則：可以向下、或者向右。同時在棋盤上的任一點有一個對方的馬（如C點），該馬所在的點和所有跳躍一步可達的點稱為對方馬的控制點（如圖中的C點和P1，P2，…，P8）。卒不能通過對方馬的控制點。棋盤用座標表示，A點（0，0）、B（n，m）（n，m為不超過20的整數），同樣馬的位置座標是需要給出的，C≠A且C≠B。現在從鍵盤輸入n，m，要你計算出卒從A點能夠到達B點的路徑的條數。



【**問題分析**】 跳馬是道老得不能再老的題目，每位程式設計者都學過，一般是在學習回溯或搜索等演算法的時候，很多書上也有類似的題目，一些比賽中也經常出現這一問題的變形，有些同學一看到這種類型的題目就去盲目搜索，但事實證明：當n,m=15就會超時。

其實，對本題稍加分析就能發現，要到達棋盤上的一個點，只能從左邊過來（我們稱之為左點）或是從上面過來（我們稱之為上點）。根據加法原理，到達某一點的路徑數目，就等於到達其相鄰的上點和左點的路徑數目之和，因此我們可以使用逐列（或逐行）遞推的方法來求出從起點到終點的路徑數目。障礙點（馬的控制點）也完全適用，只要將到達該點的路徑數目設置為0即可。

假設用F[i,j]表示到達點（i,j）的路徑數目，用g[i,j]表示點（i,j）是否是對方馬的控制點，g[i,j]=0表示不是對方馬的控制點，g[i,j]=1表示是對方馬的控制點。則我們可以得到如下的遞推關係式：

F[i,j]=0 { g[x,y]=1}

F[i,0]= F[i-1,0] { i>0，g[x,y]=0}

F[0,j]= F[0,j-1] { j>0，g[x,y]=0}

F[i,j]= F[i-1,j]+ F[i,j-1] { i>0，j>0， g[x,y]=1}

上述遞推關係式的邊界為：。考慮到最大情況下：n=20，m=20，路徑條數可能會超出長整數範圍，所以要使用comp類型或高精度運算。

【**參考程式**】

program chess;

var n,m,x,y:integer;

    a:array[-2..22,-2..22] of boolean;  {b表示棋盤上所有點是否是馬控制的點，如果是控制點，則a[?,?]為true，否則為false，為什麼下標範圍是-2..22，而不是0..22？}

    b:array[0..20,0..20] of int64;{b表示小卒子到達棋盤上每一個點的走法，問題的結果就在a[n.m]中，這裡用的int64資料類型，只有FP中才有！因為當棋盤夠大時，結果會超過longint的表示範圍}

    i,j,p:longint;

begin

   assign(input,'chess.in');

   assign(output,'chess.out');

   reset(input);

   rewrite(output);

   read(n,m, x,y); {x，y為馬的座標}

   for i:=-2 to 22 do   {初始棋盤上各點小卒子都允許通過}

        for j:=-2 to 22 do a[i,j]:=true;

   {馬可以控制9個點}

   a[x,y]:=false;

   a[x-1,y-2]:=false;

   a[x-2,y-1]:=false;

   a[x-2,y+1]:=false;

   a[x-1,y+2]:=false;

   a[x+1,y+2]:=false;

   a[x+2,y+1]:=false;

   a[x+2,y-1]:=false;

   a[x+1,y-2]:=false;

   for i:=0 to 22 do  {初始化所有點小卒子的走法為0}

      for j:=0 to 22 do b[i,j]:=0;

  {以下8 行屬於遞推}

   for i:=0 to n do

       for j:=0 to m do begin

          if not a[i,j] then b[i,j]:=0             {馬控制的點，走法為0}

          else if i+j=0 then b[i,j]:=1          {座標原點，走法為1}

          else if i=0 then b[i,j]:=b[i,j-1]     {在x軸上，走法等於前一個點的}

          else if j=0 then b[i,j]:=b[i-1,j]     {在y軸上，走法等於前一個點的}

          else b[i,j]:=b[i,j-1]+b[i-1,j];       {別的情況，等於上面相鄰點的加左邊相鄰點的}

       end;

  輸出結果;{結果在a[n,m]中}

   close(input);

   close(output);

end.

解決遞推類型問題有三個重點：一是如何建立正確的遞推關係式，二是遞推關係有何性質，三是遞推關係式如何求解。其中第一點是基礎，也最重要，不是所有題目都會像上面的題目那麼簡單。遞推通常分為順推法和倒推法。所謂順推法，就是從問題的邊界條件（初始狀態，已知的、隱含的、推導出的）出發，通過遞推關係式依次從前往後遞推出問題的解；所謂倒推法，就是在不知道問題的初始狀態（邊界條件）下，從問題的最終解（目標狀態或某個中間狀態，已知的或者經過簡單推理得到的）出發，反過來推導出問題的初始狀態。

【**遞迴**】

遞迴演算法通常是把規模較大的、較難解決的問題變為規模較小的、易解決的同一問題。規模較小的問題又變成規模更小的同一問題，並且小到一定程度便可以直接得出它的解，從而得到原來問題的解。

一個過程(或函數)直接或間接調用自己本身,這種過程(或函數)叫遞迴過程(或函數)。

如:

procedure a;

  begin

  .

  .

  .

  a;

  .

  .

  .

end;

這種方式是直接調用.

又如:

 procedure b;   procedure c;

  begin             begin

  .                 .

  .                 .

  .                 .

  c;                b;

  .                 .

  .                 .

  .                 .

end;              end;

這種方式是間接調用。

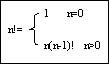
應用遞迴演算法始終要考慮以下兩個方面：

1. 遞迴的形式
2. 遞迴結束條件

顯然，如果沒有第一點，就不可能通過不斷地遞迴接近於目標；如果沒有第二點，遞迴就不會結束（一直遞迴到記憶體空間用完為止）。

【**例8-3**】 階乘函數

階乘函數可遞迴地定義為：



階乘函數的引數n的定義域是非負整數。遞迴式的第一式給出了這個函數的初始值，是非遞迴地定義的。每個遞迴函數都必須有非遞迴定義的初始值，否則，遞迴函數就無法計算。遞迴式的第二式是用較小引數的函數值來表達較大引數的函數值的方式來定義n的階乘。定義式的左右兩邊都引用了階乘記號，是遞迴定義式，可遞迴地計算如下。

function fac(n:integer):integer;

begin

if n=0 then fac:=1

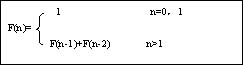
else

fac:=n\*fac(n-1)

end.

【**例8-4**】 Fibonacci數列

無窮數列1，1，2，3，5，8，13，21，34，55，…，稱為Fibonacci數列。它可以遞迴地定義為：



這是一個遞迴關係式，它說明當n大於１時，這個數列的第n項的值是它前面兩項之和，它用兩個較小的引數的函數來定義較大引數的函數值，所以需要兩個初始值F(0)和F(1)。

第n個Fibonacci數可遞迴地計算如下：

function Fibonacci(n:integer):integer;

begin

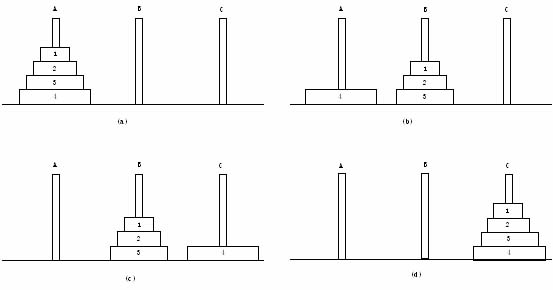
if n<=1 then Fibonacci:=1

else

Fibonacci:= Fibonacci(n-1)+Fibonacci(n-2)

end.

**【例8-5】** 梵塔問題



 如圖：設有3根標號為A，B，C的柱子，在A柱上放著n個盤子，每一個都比下面的略小一點，要求把A柱上的盤子全部移到C柱上，移動的規則是：（1）一次只能移動一個盤子；（2）移動過程中大盤子不能放在小盤子上面；（3）在移動過程中盤子可以放在A，B，C的任意一個柱子上。找出移動次數最小的方案。

【**參考程式**】

program fanta;

var

n:integer;

procedure move(n,a,b,c:integer);

begin

if n=1 then writeln(a,'--->',c)

else

begin

move(n-1,a,c,b);

writeln(a,'--->',c);

move(n-1,b,a,c);

end;

end;

begin

write('Enter n=');

read(n);

move(n,1,2,3);

end.

【**練習**】

1、已知Ａckerman函數，其定義如下：

n+1 m=0

Ack(m,n)= Ack(m-1,1) n=0

Ack(m-1,Ack(m,n-1)) m≠0且n≠0

試求A(3,5)的值？

2、已知： 求f（1.4，９）的值？

3、閱讀下列程式，求DS(9)的值？

Program ds;

Var a,s:integer;

procedure ds(x:integer);

begin

if x=1 then s:=1;

else

begin

x:=x-2;ds(x);s:=s\*2;

end;

begin

readln(a);ds(a);write(s);

end.

【**例8-6**】 排列問題

輸出n個元素的無重複的全排列。N個元素有n！種不同排列。

1 個元素直接輸出；

2 個元素有兩種排列，如（a,b）（b,a）;

3 個元素以（a,b,c）為例，有6種不同的排列：

a b c, a c b, b a c, b c a, c b a, c a b

分析這些排列，一個簡單的演算法是：

（1）a後隨（b,c）的所有排列。

（2）b後隨（a,c）的所有排列。

（3）c後隨（b,a）的所有排列。

上面（2）是將（1）中的a，b互換位置；上面（3）是將（1）中的a，c互換位置。這裡意味著用迴圈的方法來重複執行“交換位置，後隨剩餘序列的所有排列”；而對剩餘序列可以再使用這個方法，這就成了遞迴呼叫，後隨的元素沒有時，就到了遞迴的邊界。

對於n個元素a=(a1，a2，a3，…，ak，…，an)，設過程perm(a,k,n)是求a的第k個到n個元素的全排列，設swap(a,k,i)是將a的第k個元素和第i個元素對換，i=k,…,n。有偽代碼寫的演算法如下：

procedure perm(a,k,n)

if k=n then [print a,返回] {當k指向最後的元素時，遞迴終止，輸出A}

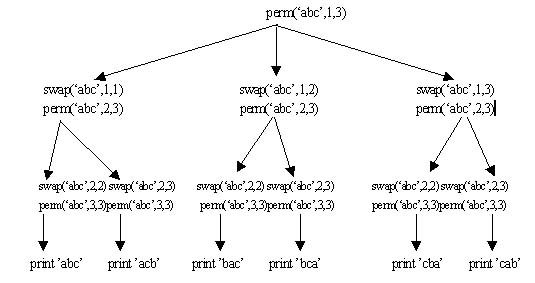
else

for i=k to n do

[call swap(a,k,i) { ak依次與ak，…，an的元素交換位置}

call perm(a,k+1,n)] { 遞迴呼叫，求ak之後序列的全排列}

如求a=(a b c)的全排列，具體層次如下：



program pailei;

var a:string;

k,n:integer;

procedure swap(var a:string;k,i:integer);

var t:char;

begin

t:=a[k];a[k]:=a[i];a[i]:=t;

end;

procedure perm(a:string;k,n:integer);

var i:integer;

begin

if k:=n then writeln(a)

else for i:=k to n do

begin swap(a,k,i);perm(a,k+1,n);end;

end;

begin

write(‘input a string:’);

readln(a);n:=length(a);perm(a,1,n);

end.

【**例8-7**】 自然數拆分。任何一個大於1的自然數n，總可以拆分成若干個小於n的自然數之和。試求n的所有拆分。

【**演算法分析**】

n=2 可拆分成 2=1+1

n=3 可拆分成 3=1+2

3=1+1+1

n=4 可拆分成 4=1+3

4=1+1+2

4=1+1+1+1

4=2+2

……

取=7來說明：

第一步：將n拆分為2：

7=1+6

7=2+5

7=3+4

拆分的特點是第2個加數總大於等於第1個加數，因此第1個加數從1變化到n div 2；

第二步：只要第2個加數仍大於1，就可按第一步重複拆分，例如：

對於 7=1+6

7=1+1+5

7=1+2+4

7=1+3+3

其中7=1+1+5又可繼續拆分獲得：

7=1+1+1+4

7=1+1+2+3

……

最後獲得如下結果（共14種拆分）：

7=1+6

7=1+1+5

7=1+1+1+4

7=1+1+1+1+3

7=1+1+1+1+1+2

7=1+1+1+1+1+1+1

7=1+1+1+2+2

7=1+1+2+3

7=1+2+2+2

7=1+3+3

7=2+5

7=2+2+3

7=3+4

【**演算法實現**】 用陣列a[0..100](假設操作中輸入的n<=100)存儲已完成的一種拆分。

例如：7=1+6 表示為a[0]=7,a[1]=1,a[2]=6,k=2；繼續拆分從a[k]開始；a[k]能否再拆分取決於a[k] div 2是否大於等於a[k-1]；遞迴過程有一個參數kx，它指向要拆分的數a[k]。

program chaifen;

var n,i,j;integer;

a:array[0..100] of integer;

procedure sum(kx:integer);

var k,m,L:integer;

begin

j;=j+1;

write(‘sum No.’,j:3’:’,a[0],’=’);

for k:=1 to kx-1 do

write(a[k],’+’);writeln(a[kx]);

k:=kx;L:=a[k];

for m:=a[k-1] to L div 2 do

begin a[k]:=m;a[k+1]:=L-m;sum(k+1);end;

end;

begin

write(‘input n:’);read(a[0]);j:=0

if a[0]>=2 then

for i:=1 to a[0] div 2 do

begin a[1]:=I;a[2]:=a[0]-a[1];sum(2);end;

writeln(‘program end.’);

end.

我們在編寫程式時是否使用遞迴演算法，關鍵是看問題是否適合用遞迴演算法來求解。由於遞迴演算法編寫的程式邏輯性強，結構清晰，正確性易於證明，程式調試也十分方便，在NOIP中，資料的規模一般也不大，只要問題適合用遞迴演算法求解，我們還是可以大膽地使用遞迴演算法。

【**練習**】

1、環問題

有N(2<=N<=9)個環,拆裝這些環的規則:第一個環可以隨意拆裝,第二個環只有在第一環已裝上時可以拆裝;第I個環只有在第i-1環已裝上,且第i-2， 第i-3……第1環都拆下時可以裝拆.編程式描述拆下N個環的過程.

2、用遞迴演算法完成：有52張牌，使它們全部正面朝上，第一輪是從第2張開始，凡是2的倍數位置上的牌翻成正面朝下；第二輪從第3張牌開始，凡是3的倍數位置上的牌，正面朝上的翻成正面朝下,正面朝下的翻成正面朝上；第三輪從第4張牌開始，凡是4的倍數位置上的牌按上面相同規則翻轉，以此類推，直到第一張要翻的牌超過52為止。統計最後有幾張牌正面朝上，以及它們的位置號。

**第九章 回溯法**

回溯法是一種既帶有系統性又帶有跳躍性的搜索法，它的基本思想是：在搜索過程中，當探索到某一步時，發現原先的選擇達不到目標，就退回到上一步重新選擇。它主要用來解決一些要經過許多步驟才能完成的，而每個步驟都有若干種可能的分支，為了完成這一過程，需要遵守某些規則，但這些規則又無法用數學公式來描述的一類問題。下面通過實例來瞭解回溯法的思想及其在電腦上實現的基本方法。

回溯法的基本思想是窮舉搜索。一般適用于尋找解集或找出滿足某些約束條件的最優解的問題。這些問題所具有的共性是順序性，即必須先探求第一步，確定第一步採取的可能值，再探求第二步採取的可能值，然後是第三步，……，直到達到目標狀態。

【**例9-1**】 某人要從A路口經過4個路口（含起始路口和目的路口）到達B路口，已知該地區的道路有一個特點：除起點和終點外，每個路口都有三條叉路。請找出一條可行的路線。

首先，應該分析出我們所能找到的有關資訊：

（1）從起點地到目的地一共要經過且只能經過4個路口；

（2）除起點和終點外，每個路口都有三條叉路；

要解決這一個問題，首先應該將地形圖（如圖9-1-1中的圖a）轉換成較規範的路徑圖（如圖9-1-1中的圖b），其中路徑圖中的結點對應地圖的路口，路徑圖中的連線對應地圖的路；然後嘗試按照某種規則進行路的探求。當在某個路口走不通時就回頭另找一條新路，當發現了目標地就可以停止尋找。

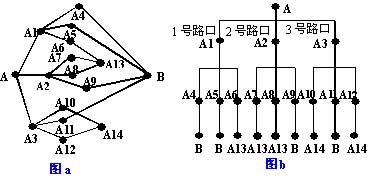


圖9-1-1

根據回溯法，我們很容易找出問題所涉及到的資料與路徑之間的對應關係：路口所在的層數表示探求步數對應的編號，每個路口的分叉表示從該情況出發所有可能的情況。

在資料的具體描述中，為體現探求步驟的順序性，可以將層數用整數來表示；為表示每個出發點的規律性，我們通常將情況歸納一下，並用順序出現的整數表示。

對於每個路口而言，其分叉的形式應是一致的。因此通常情況下我們用整數來表示，如圖b，由於每個路口都有三個分叉，那麼我們可以根據分叉出現的順序從左向右依次用整數1，2，3進行編號就可以了。

在程式中我們可以用陣列a存放路徑，其中下標表示探求的步數編號，陣列元素a[k]的值表示在第k步所採取的可能值的編號。這樣對路的探求方法都可以用以下的代碼表示。

procedure find(k:integer); {找第k步的可能性}

begin

if 到目的地 then {表示一條路已找出}

begin 輸出路線；結束探求；end;

else

if k<=4 then

for i:=1 to 3 do {窮舉三種可能的方向並記錄下來}

begin a[k]:=i;find(k+1);end;

end.

由此可知，根據每個結點探求具有共性這個特點，在實現中我們通常採用遞迴的演算法來實現回溯策略。下面給出的是對於該遞迴演算法的非遞迴代碼：

begin

a[1]:=1;k:=1; {確定起步的路口號及尋找的初始方向}

while 未到目的地 do

begin

while a[k] 無路可走 do {回溯，返回第k-1個路口，另找一條路}

begin k:=k-1;a[k]:=a[k]+1;end;

k:=k+1;a[k]:=1;

end;

輸出路線；

end.

通過觀察，我們可以發現實現回溯演算法的特性：在解決過程中首先必須要先為問題定義一個解的空間，這個空間必須包含問題的一個解。在搜索路的同時也就產生了新的解的空間。在搜索期間的任何時刻，僅保留從起點到當前點的路徑。因此，回溯演算法的空間需求為（從開始節點起最長的路徑的長度）。

回溯的策略是一種常見的策略。它可以避免對規模很大的候選解進行一次性檢查，因此適用于一些求解規模很大的問題。在具體實現過程中我們應該以找出解題方法與路徑圖對應的資料關係為切入點，使用遞迴的演算法加以實現。回溯策略，通常用來解決自然數的排列、n皇后問題、迷宮問題、數的拆分、0/1背包問題、旅行商問題、貨船裝貨和圖形覆蓋正方形等問題。

【**例9-2**】 n皇后問題

【**問題描述**】 在一個國際棋盤上，放置n個皇后（n<10），使她們相互之間不能進攻。求出所有佈局。

輸入：n

輸出：每行輸出一種方案，每種方案順序輸出皇后所在的列號，各個數之間用空格隔開。

【**樣例輸入**】 4

【**樣例輸出**】

2 4 1 3

2 1 4 2

【**問題分析**】

這是一個古老的搜索問題。由於每人皇后都必須要確定，因此用回溯的方法來解決是最合適的。最常見的方法是首先將每個皇后的可取值都訪問一遍，然後以衝突條件作為“剪去”分叉的依據，就可以得到結果。解決問題所需的資料結構為陣列a:array[1..100] of integer；其中下標表示棋盤行號，同時表示皇后的編號；陣列元素的值表示皇后所在的列號。

衝突條件解決如下：

條件一：所有皇后不能在同一行上；

解決方案：可將皇后進行編號，並將編號對應陣列的下標；

條件二：所有皇后不能在同一列上；

解決方案：可將皇后所處的列進行編號，並確保任意兩個皇后的列號不相等。對第k個皇后而，a[k]<>a[i]（其中i的值是從1到k-1）。

條件三：所有皇后都不能出現在對角線上；

解決方案：對角線有兩種，一種是左上到右下的，一種是右上到左下的。

通過分析對角線上的兩皇后的行與列的關係可得出她們處在對角線上的數學運算式為：

abs(a[i]-a[k])=abs(i-k)

過程描述：

procedure try(k:integer);

begin

if k>n then 輸出a[1]~a[n]

else begin

for i:=1 to n do

if i 的值不衝突 then begin a[k]:=I;try(k+1);end;

end;

end.

【**例9-3**】 0/1字串問題

問題描述：輸出僅由0和1組成的長度為n的字串，並且其中不可含有三個連續的相同子串。

輸入：字串長度n（n<=40）。

輸出：所有滿足條件的字串的個數。

【**樣例輸入**】 2

【**樣例輸出**】 4

【**問題分析**】

1、對三個連續的相同子串的理解：

首先我們必須明確，所謂三個連續的相同子串是指：

1. 三個子串長度相等且對應位置的數字相同；
2. 三個子序列順序銜接；

設0/1序列的長為L，字串的起始指標為1，其中有三個長度同為M的子序列，那麼

三個連續字串序列的起始指標分別為：L-M+1、L-2\*M+1、L-3\*M+1，其中M的範圍應從1到1/3 L。

2、對於某種長度為n的字串進行搜索的方法：

x——第三個子序列的首指針；

y——第二個子序列的首指針；

z——第一個子序列的首指針；

初始時x=L，y=L-1，z=L-2，即三個子序列各含一個數字。若三個數位不同，則x←x-1，y←y-2，z←z-3，使得三個子序列各含兩個數位；若三個子序列各不相同，則x←x-1，y←y-2，z←z-3，使得各子序列的數位個數擴充為3…，依次類推，直至az…ay-1= ay…ax-1= ax…a1或者z<=0（即序列中不存在三個連續的相同子序列）為止。

為了判別三個分別以z、y、x為指標的子序列是否相同，我們設r、q、p分別為第一個子序列，第二個子序列、第三個子序列的搜索指針。判別前設r=z、q=y、p=x。若r、q、p所指的三個字元相同，則r、q、p指針分別右移一位，再比較三個右移後位置上的數字，若相同再右移……依此類推。若比較過程中出現不同數字，則說明三個子序列非連續相同；若右移至r=y，則說明三個子序列連續相同。

例如：

z y x

↓ ↓ ↓

序列a ： 0 1 0 0 1 0 0 1 0

位 ：1 2 3 4 5 6 7 8 9

初始時x=9、y=8、z=7，第一個子序列為a7，第2個子序列為a8，第3個子序列為a9。由於a7 ≠a8 ≠a9，因此x←x-1=8，y=8-2=6，z=z-3=4，使得第1個子序列為a4 a5=01，第2個子序列為a6 a7=00，第3個子序列為a8 a9=10。這三個子序列的搜索指針分別為r=4，q=6，p=8。

由於ap≠aq≠ar，三個子序列不同，因此x←x-1=7，y←y-2=4，z←z-3=1，使得第1個子序列為a1a2a3=010，第2個子序列為a4a5a6=010，第3個子序列為a7a8a9=010。搜索前。r=1，q=4，p=7。

由於ap=aq=ar，因此r、q、p指針分別右移一位，右移後所指的數學相同，三個指針再移，…，直至移至後，三個指標移動過程中還未出現數位不同的情況，可以斷定a1a2a3= a4a5a a6=a7a8a9=010，它們是a中三個連續的相同子序列。

3、對於判斷的優化演算法：

考慮序列中0/1的對稱性。在滿足條件的方案中，是對稱的。

例如時，中不含3個連續的相同子串；那麼100110也必然不含3個連續的相同子串；

因此，我們只要查找第1位為0的序列，將得出的方案數乘以2即為所求的方案總數，這樣可減少一半的搜索量。

4、字串的生成方法：

我們可以使用回溯生成所有第1位為0長度為n、且不含三個連續的相同子串的0/1序列。初始時設為0，然後試填。填數的順序為先填0後填1。若當前位填0後，不存在以當前位為最後的一位的三個連續的相同子串，則填下一位；否則當前位改填1。若填1後出現以當前位為最後一位的三個連續的相同子串，則回溯至前一位，重新填數；否則填下一位。當生成滿足條件的一個字串時，方案數=方案數+2。這樣反復搜索，直至所有滿足條件的0/1序列產生為止。

【**資料結構**】

L——位指標，調用前L←1；

a——01序列，調用前a[1]←0

tot——方案總數，調用前tot←0

【**例9-5**】 求N個數的全排列。

【**分析**】求N個數的全排列，可以看成把N個不同的球放入N個不同的盒子中，每個盒子中只能有一個球。解法與八皇后問題相似。

【**參考過程**】

procedure try(i:integer);

var j:integer;

begin

for j:=1 to n do

if a[j]=0 then

begin

x[i]:=j;

a[j]:=1;

if i<n then try(i+1)

else print;

a[j]=0;

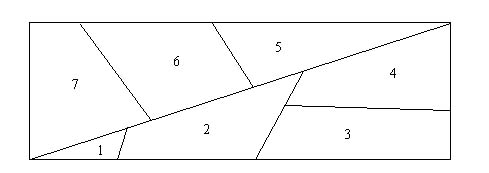
end;

end;

【**練習**】

1、四色問題

設有如下圖所示的地圖，每個區域代表一個省，區域中的數位代表省的編號，現在要求給每個省塗上紅（R），藍（B），白（W），黃（Y）四種顏色之一，同時使用相鄰的省份以不同的顏色區分。



輸入：用鄰接矩陣表示地圖。從文件中讀入，檔案格式如下：

N （有N個省）

N行用空格隔開的0/1串 （1表示相鄰，0表示不相鄰）

輸出：RBWY串

2、在n\*n棋盤上（1<=n<=10）填入1，2，3，…，n\*n，共有n\*n個數，使得任意兩個相鄰數的和為素數。

例如：當n=2時，有：

|  |  |
| --- | --- |
| 1 | 2 |
| 4 | 3 |

當n=4時，一種可以填寫的方案如下表，在這裡我們約定：左上角的格子必須填數位1。程式要求：①輸入n；②若輸出有多個解，則輸出第一行、第一列之和為最小的排列方案；若無解，則輸出“NO！”

|  |  |  |  |
| --- | --- | --- | --- |
| 1 | 2 | 11 | 12 |
| 16 | 15 | 8 | 5 |
| 13 | 4 | 9 | 14 |
| 6 | 7 | 10 | 3 |

**第十章 分治法**

**分治演算法**

分治演算法的基本思想是將一個規模為N的問題分解為K個規模較小的子問題，這些子問題相互獨立且與原問題性質相同。求出子問題的解，就可得到原問題的解。

分治法解題的一般步驟：

（1）分解，將要解決的問題劃分成若干規模較小的同類問題；

（2）求解，當子問題劃分得足夠小時，用較簡單的方法解決；

（3）合併，按原問題的要求，將子問題的解逐層合併構成原問題的解。

根據分治法的分割原則，應把原問題分為多少個子問題才比較適宜？每個子問題是否規模相同或怎樣才為適當？這些問題很難給予肯定的回答。但人們從大量實踐中發現，在用分治法設計演算法時，最好使子問題的規模大致相同。即將一個問題分成大小相等的k個子問題的處理方法是行之有效的。許多問題可以取k=2。這種使子問題規模大致相等的做法是出自一種平衡子問題的思想，它幾乎總是比子問題規模不等的做法要好。

【**例10-1**】 二分搜索演算法是運用分治策略的典型例子。給定已排好序的n個元素a[1:n]，現要在這n個元素中找出一特定的元素x。二分搜索演算法的基本思想是將n個元素分成個數大致相同的兩半，取a[n/2]與x進行比較。如果x= a[n/2]，則找到x，演算法終止。如果x< a[n/2]，則只要在陣列a的左半部繼續搜索x。如果x> a[n/2]，則只要在陣列a的右半部繼續搜索x。

具體演算法可描述如下

function binarysearch(a:array of [1..n];:integer;n:integer):integer;

var left,right,middle:integer;

begin

left:=1;right:=n;

while (left<=right) do

begin

middle:=(left+right) div 2

if x=a[middle] then begin binarysearch:=middle;break;end;

if x>a[middle] then left:=middle+1 else right:=middle-1;

end;

if find=false then write(“not find”);

end;

【**例10-2**】 循環賽日程表

設有n=2 k個運動員要進行網球循環賽。現要設計一個滿足以下要求的比賽日程表：

（1）、每個選手必須與其他n-1個選手各賽一次；

（2）、每個選手一天只能賽一次；

（3）、循環賽一共進行n-1天。

按此要求可將比賽日程表設計成有n行n-1列的表。在表中第i行和第j列處填入第i個選手在第j天所遇到的選手。

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
| 2 | 1 | 4 | 3 | 6 | 5 | 8 | 7 |
| 3 | 4 | 1 | 2 | 7 | 8 | 5 | 6 |
| 4 | 3 | 2 | 1 | 8 | 7 | 6 | 5 |
| 5 | 6 | 7 | 8 | 1 | 2 | 3 | 4 |
| 6 | 5 | 8 | 7 | 2 | 1 | 4 | 3 |
| 7 | 8 | 5 | 6 | 3 | 4 | 1 | 2 |
| 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 |

按分治策略，可以將所有的選手分為兩半，n個選手的比賽日程表就可以通過為n/2個選手設計的比賽日程表來決定。遞迴地用這種一分為二的策略對選手進行分割，直到只剩下2個選手時，比賽日程表的制定就變得很簡單。這時只要讓這2個選手進行比賽就可以了。

上圖所列出的表格就是8個選手的比賽日程表。其中左上角與左下角的2小塊分別為選手1至選手4以及選手5至選手8前3天的比賽日程。據此，將左上角小塊中的所有數字按其相對位置抄到右下角，將左下角小塊中的所有數字按其相對位置抄到右上角，這樣不分別安排好了選手1至選手4以及選手5至選手8後4天的比賽日程。依時思想容易將這個比賽日程表推廣到具有任意多個選手的情形。

在演算法設計中，用陣列a記錄2^n個球隊的迴圈比賽表，整個迴圈比賽表從最初的1\*1方陣按上述規則生成2\*2的方陣，再生成4\*4的方陣，……直到生成出整個迴圈比賽表為止。變數h表示當前方陣的大小,也就是要生成的下一個方陣的一半。

【**參考程式**】

var

i,j,h,m,n:integer;

a:array[1..32,1..32]of integer;

begin

readln(n);

m:=1;a[1,1]:=1;h:=1;

for i:=1 to n do m:=m\*2;

repeat

for i:=1 to h do

for j:=1 to h do begin

a[i,j+h]:=a[i,j]+h;{構造右上角方陣}

a[i+h,j]:=a[i,j+h];{構造左下角方陣}

a[i+h,j+h]:=a[i,j];{構造右下角方陣}

end;

h:=h\*2;

until h=m;

for i:=1 to m do

begin

for j:=1 to m do write(a[i,j]:4);

writeln;

end;

end.

【**例10-3**】 求方程f(x)=2x+3x-4x=0在[1,2]內的根。

【**參考程式**】

e0=1e-38;

function f(x:real):real;

begin

f:=exp(x\*ln(2))+ exp(x\*ln(3))- exp(x\*ln(4));

end;

begin

a:=1;b:=2;

repeat

x:=(a+b)/2;fa:=f(a);fx:=f(x);

if fa\*fx<0 then b:=x {[a,x]內有解}

else a:=x;

until (abs(b-a)<e0);

【**練習**】

1、設有n個運動員要進行網球循環賽。設計一個滿足以下要求的比賽日程表：

（1）、每個選手必須跟其他n-1個選手各賽一次；

（2）、每個選手一天只能賽一次；

（3）、當n是偶數時，循環賽一共進行n-1天；當n是奇數時，循環賽一共進行n天。

2、一元三次方程求解

問題描述：有形如：ax3+bx2+cx+d=0  這樣的一個一元三次方程。給出該方程中各項的係數(a，b，c，d  均為實數)，並約定該方程存在三個不同實根(根的範圍在-100至100之間)，且根與根之差的絕對值>=1。要求由小到大依次在同一行輸出這三個實根(根與根之間留有空格)，並精確到小數點後2位。

提示：記方程f(x)=0，若存在2個數x1和x2，且x1<x2，f(x1)\*f(x2)<0，則在(x1，x2)之間一定有一個根。

樣例

輸入：1   -5   -4   20

輸出：-2.00   2.00   5.00

**第十一章 貪心法**

在實際生活中，我們經常會遇到求一個問題的可行解和最優解的要求，這就是所謂的最優化問題。每個最優化問題都包含一組限制條件和一個優化函數，符合限制條件的問題求解方案稱為可行解，使優化函數取得最佳值的可行解稱為最優解。

貪心法是求解這類問題的一種常用演算法，它是從問題的某一個初始解出發，採用逐步構造最優解的方法向給定的目標前進。在每個局部階段，都做出一個看上去最優的決策（即某種意義下的、或某個標準下局部最優解），並期望通過每次所做的局部最優選擇產生出一個全域最優解。做貪心決策的依據稱為貪心準則（策略），但要注意決策一旦做出，就不可再更改。貪心與遞推不同的是，推進的每一步不是依據某一固定的遞推式，而是做一個當時看似最佳的貪心選擇（操作），不斷地將問題實例歸納為更小的相似子問題。所以，在有些最優化問題中，採用貪心法求解不能保證一定得到最優解，這時我們可以選擇其他最優化問題的演算法，如動態規劃等。歸納、分析、選擇貪心準則是正確解決貪心問題的關鍵。

【**例11-1**】 刪數問題

鍵盤輸入一個高精度的正整數n（≤240位元），去掉其中任意s個數位後剩下的數位按原左右次序將組成一個新的正整數。程式設計對給定的n和s，尋找一種方案，使得剩下的數位組成的新數最小。

輸入：n,s

輸出：最後剩下的最小數。

【**樣例輸入**】 178543,4

【**樣例輸出**】 13

【**問題分析**】

由於正整數n的有效數位為240位，所以很自然地採用字串來存貯n。那麼如何決定哪s位刪除呢？是不是最大的s個數字呢？為了盡可能逼近目標，我們選取的貪心策略為：每一步總是選擇一個使剩下的數最小的數字刪去，即按高位到低位元的循序搜尋，若各位數字遞增，則刪除最後一個數字；否則刪除第一個遞減區間的首字元，這樣刪一位元便形成了一個新數字串。然後回到串首，按上述規則再刪下一個數位。重複以上過程s次為止，剩下的數字串便是問題的解了。

例如：n=178543，s=4

刪數過程如下：

n=178543 {刪掉8}

17543 {刪掉7}

1543 {刪掉5}

143 {刪掉4}

13 {解}

這樣，刪數問題就與如何尋找遞減區間首字元這樣一個簡單的問題對應起來。不過還要注意一個細節性的問題，就是可能會出現字串串首有若干個0的情況，甚至整個字串都是0的情況。程式如下：

輸入n，s

while s>0 do

begin

i:=1; {從串首開始找}

While (i<length(n)) and (n[i]<=n[i+1]) do i:=i+1;

Delete(n,i,1); {刪除字串n的第i個字元}

s:=s-1;

end;

While (length(n)>1) and (n[1]=’0’) do delete(n,1,1); {刪去串首可能產生的無用0}

輸出n。

【**例11-2**】 0/1背包問題

有一容量為weight的背包。現在要從n件物品中選取若干裝入背包中，每件物品i的重量為w[i]，價值為p[i]。定義一種可行的背包裝載為：背包中物品的總重量不能超過背包的容量，並且一個物品要麼全部選取，要麼不選取，定義最佳裝載是指所裝入的物品價值最高，並且是可行的背包裝載。

【**樣例輸入**】

weight 11

n 4

w[i] 2 4 6 7

p[i] 6 10 12 13

【**樣例輸出**】

0 1 0 1

23

【**問題分析**】

設陣列choice[1..n]，若choice[i]=1表示物品i裝入背包中，choice[i]=0表示物品不裝入背包中，則choice=[0 1 0 1]是一種可行的背包裝載方案，也是一種最佳裝載方案，此時總價值為23。

【**演算法分析**】

0/1背包問題有好幾種貪心準則，每種貪心準則都是採用多步過程來完成背包的裝入，在每一步過程中利用某種固定的貪心準則選擇一個物品裝入背包。

一種貪準則為：從剩餘的物品中，選出可以裝入背包的價值最大的物品，利用這種規則，價值最大物品首先被裝入（假定有足夠容量），然後是下一個價值最大的物品，如此繼續下去。這種貪心準則不能保證得到最優解。如：weight=105;n=3;w[i]=100,10,10;p[i]=20,15,15。按以上準則得出的結果為：choice[1,0,0]=20；而最優解為choice[0,1,1]=30。

另一種方案是“重量貪心準則”，即從剩下的物品中選擇可裝入背包的重量最小的物品，雖然這種規則對於前面的例子能產生最優解，但在一般情況下不一定得到最優解。如：weight=25;n=2;w[i]=10,20;p[i]=5,100。按以上準則得出的結果為：choice[1,0]=5；而最優解為choice[0,1]=100。

本題的另一種方案為“單位價值貪心準則”，這種方案是從剩餘物品中選擇可裝入背包的p[i]/w[i]值最大的物品，這種策略也不能保證得到最優解。如：weight=30;n=3;w[i]=20,15,15;p[i]=40,25,25。按以上準則得出的結果為：choice[1,0,0]=40；而最優解為choice[0,1,1]=50。

我們不必因所考察的幾個貪婪演算法都不能保證得到最優解而沮喪， 0 / 1背包問題是一個N P-完全問題（NPC）。對於這類問題，也許根本就不可能找到具有多項式時間的演算法。雖然按pi /wi 非遞（增）減的次序裝入物品不能保證得到最優解，但它是一個直覺上近似的解。我們希望它是一個好的啟發式演算法，且大多數時候能很好地接近最後演算法。在6 0 0個隨機產生的背包問題中，用這種啟發式貪婪演算法來解有2 3 9題為最優解。有5 8 3個例子與最優解相差1 0 %，所有6 0 0個答案與最優解之差全在2 5 %以內。該演算法能在O (nlogn)時間內獲得如此好的性能。我們也許會問，是否存在一個x (x<1 0 0 )，使得貪婪啟發法的結果與最優值相差在x%以內。答案是否定的。為說明這一點，考慮例子weight=y，n =2, w = [ 1 ,y], p= [ 1 0 , 9y],貪婪演算法結果為choice[1,0]=10。對於y≥1 0 / 9，最優解的值為9 y。因此，貪婪演算法的值與最優解的差對最優解的比例為( ( 9y - 1 0)/9y\* 1 0 0 ) %，對於大的y，這個值趨近於1 0 0 %。但是可以建立貪婪啟發式方法來求解，使解的結果與最優解的值之差在最優值的x% (x<100) 之內。首先將最多k 件物品放入背包，如果這k 件物品重量大於weight，則放棄它。否則，剩餘的容量用來考慮將剩餘物品按pi /wi 遞減的順序裝入。通過考慮由啟發法產生的解法中最多為k 件物品的所有可能的子集來得到最優解。

【**具體解法**】

當k= 0時，背包按物品價值密度遞減順序裝入，首先將物品1放入背包，然後是物品2，背包剩下的容量為5個單元，剩下的物品沒有一個合適的，因此解為choice [ 1 , 1 , 0 , 0 ]=16。此解獲得的價值為1 6。

現在考慮k = 1時的貪婪啟發法。最初的子集為{ 1 } , { 2 } , { 3 } , { 4 }。子集{ 1 } , { 2 }產生與k= 0時相同的結果，考慮子集{ 3 }，置choice[3]=1。此時還剩5個單位的容量，按價值密度遞減順序來考慮如何利用這5個單位的容量。首先考慮物品1，它適合，因此取choice[1] 為1，這時僅剩下3個單位容量了，且剩餘物品沒有能夠加入背包中的物品。通過子集{ 3 }開始求解得結果為choice [ 1 , 0 , 1 , 0 ]=18，獲得的價值為1 8。若從子集{ 4 }開始，產生的解為choice [ 1 , 0 , 0 , 1 ]=19，獲得的價值為1 9。考慮子集大小為0和1時獲得的最優解為[ 1 , 0 , 0 , 1 ]。這個解是通過k= 1的貪婪啟發式演算法得到的。

若k= 2，除了考慮k< 2的子集，還必需考慮子集{ 1 , 2 } , { 1 , 3 } , { 1 , 4 } , { 2 , 3 } , { 2 , 4 }和{ 3 , 4 }。首先從最後一個子集開始，它是不可行的，故將其拋棄，剩下的子集經求解分別得到如下結果：[ 1 , 1 , 0 , 0 ] , [ 1 , 0 , 1 , 0 ] , [ 1 , 0 , 0 , 1 ] , [ 0 , 1 , 1 , 0 ]和[ 0 , 1 , 0 , 1 ]，這些結果中最後一個價值為2 3，它的值比k= 0和k= 1時獲得的解要高，這個答案即為啟發式方法產生的結果。

這種修改後的貪婪啟發方法稱為k階優化方法（k - o p t i m a l）。也就是，若從答案中取出k 件物品，並放入另外k 件，獲得的結果不會比原來的好，而且用這種方式獲得的值在最優值的( 1 0 0 / (k + 1 ) ) %以內。當k= 1時，保證最終結果在最佳值的5 0 %以內；當k= 2時，則在3 3 . 3 3 %以內等等，這種啟發式方法的執行時間隨k 的增大而增加，需要測試的子集數目為O (nk )，每一個子集所需時間為O (n)，因此當k >0時總的時間開銷為O (nk+1)。實際觀察到的性能要好得多。

【**搜索方法**】

    procedure search(k,v:integer); {搜索第k個物品，剩餘空間為v}

    var i,j:integer;

    begin

       if v<best then best:=v;

       if v-(s[n]-s[k-1])>=best then exit; {s[n]為前n個物品的重量和}

       if k<=n then begin

         if v>w[k] then search(k+1,v-w[k]);

         search(k+1,v);

       end;

    end;

【**例11-3**】 均分紙牌（NOIP2002tg）

【**問題描述**】 有 N 堆紙牌，編號分別為 1，2，…, N。每堆上有若干張，但紙牌總數必為 N 的倍數。可以在任一堆上取若干張紙牌，然後移動。移牌規則為：在編號為 1 堆上取的紙牌，只能移到編號為 2 的堆上；在編號為 N 的堆上取的紙牌，只能移到編號為 N-1 的堆上；其他堆上取的紙牌，可以移到相鄰左邊或右邊的堆上。現在要求找出一種移動方法，用最少的移動次數使每堆上紙牌數都一樣多。例如 N=4，4 堆紙牌數分別為：

　　①　9　②　8　③　17　④　6

移動3次可達到目的：

　　從 ③ 取 4 張牌放到 ④ （9 8 13 10）→從 ③ 取 3 張牌放到 ②（9 11 10 10）→ 從 ② 取 1 張牌放到①（10 10 10 10）。

輸 入：鍵盤輸入檔案名。

檔案格式：N（N 堆紙牌，1 <= N <= 100）

　　 A1 A2 … An （N 堆紙牌，每堆紙牌初始數，l<= Ai <=10000）

輸 出：輸出至螢幕。格式為：所有堆均達到相等時的最少移動次數。

【**輸入樣例**】

a.in：

　4

　9 8 17 6

【**輸出樣例**】 3

【**演算法分析**】 設a[i]為第i堆紙牌的張數（0<=i<=n），v為均分後每堆紙牌的張數，s為最小移到次數。

我們用貪心法，按照從左到右的順序移動紙牌。如第i堆(0<i<n)的紙牌數a[i]不等於平均值，則移動一次(即s加1)，分兩種情況移動：

（1）若a[i]>v，則將a[i]-v張紙牌從第i堆移動到第i+1堆；

（2） 若a[i]<v，則將v -a[i]張紙牌從第i+1堆移動到第i堆；

為了設計的方便，我們把這兩種情況統一看作是將a[i]-v張牌從第i堆移動到第i+1堆；移動後有：a[i]:=v；a[i+1]:=a[i+1]+a[i]-v；

在從第i+1堆中取出紙牌補充第i堆的過程中，可能會出現第i+1堆的紙牌數小於零(a[i+1]+a[i]-v<0 )的情況。

如n=3，三堆紙牌數為（1，2，27）這時v=10，為了使第一堆數為10，要從第二堆移9張紙牌到第一堆，而第二堆只有2張紙牌可移，這是不是意味著剛才使用的貪心法是錯誤的呢？

我們繼續按規則分析移牌過程，從第二堆移出9張到第一堆後，第一堆有10張紙牌，第二堆剩下-7張紙牌，再從第三堆移動17張到第二堆，剛好三堆紙牌數都是10，最後結果是對的，從第二堆移出的牌都可以從第三堆得到。我們在移動過程中，只是改變了移動的順序，而移動的次數不變，因此此題使用貪心法是可行的。

【**參考程式**】：

var

i,n,s:integer;v:longint;

a:array[1..100]of longint;

f:text;fil:string;

begin

readln(fil);

assign(f,fil);reset(f);

readln(f,n);v:=0;

for i:=1 to n do begin

read(f,a[i]); inc(v,a[i]);

end;

v:=v div n; {每堆牌的平均數}

for i:=1 to n-1 do

if a[i]<>v then {貪心選擇}

begin

inc(s);{移牌步數計數}

a[i+1]:=a[i+1]+a[i]-v;{使第i堆牌數為v}

end;

writeln(s);

end.

**總結**

利用貪心演算法解題，需要解決兩個問題：

一是問題是否適合用貪心法求解。我們看一個找幣的例子，如果一個貨幣系統有3種幣值，面值分別為一角、五分和一分，求最小找幣數時，可以用貪心法求解；如果將這三種幣值改為一角一分、五分和一分，就不能使用貪心法求解。用貪心法解題很方便，但它的適用範圍很小，判斷一個問題是否適合用貪心法求解，目前還沒有一個通用的方法，在資訊學競賽中，需要憑個人的經驗來判斷何時該使用貪心演算法。

二是確定了可以用貪心演算法之後，如何選擇一個貪心標準，才能保證得到問題的最優解。在選擇貪心標準時，我們要對所選的貪心標準進行驗證才能使用，不要被表面上看似正確的貪心標準所迷惑，如下面的列子。

【**例11-4**】 （NOIP1998tg）設有n個正整數，將他們連接成一排，組成一個最大的多位元整數。例如:n=3時，3個整數13,312,343,連成的最大整數為:34331213

又如:n=4時,4個整數7,13,4,246連接成的最大整數為7424613

輸入：N

N個數

輸出:連接成的多位數

【**演算法分析**】 此題很容易想到使用貪心法，在考試時有很多同學把整列數按從大到小的順序連接起來，測試題目的例子也都符合，但最後測試的結果卻不全對。按這種貪心標準，我們很容易找到反例：12，121 應該組成12121而非12112,那麼是不是相互包含的時候就從小到大呢？也不一定，如：12，123 就是12312而非12123,這樣情況就有很多種了。是不是此題不能用貪心法呢？

其實此題是可以用貪心法來求解，只是剛才的貪心標準不對，正確的貪心標準是：先把整數化成字串，然後再比較a+b和b+a，如果a+b>b+a，就把a排在b的前面，反之則把a排在b的後面。

【**參考程式**】：

var

s:array[1..20] of string;

t:string;i,j,k,n:longint;

begin

readln(n);

for i:=1 to n do begin

read(k);

str(k,s[i]);

end;

for i:=1 to n-1 do

for j:=i+1 to n do

if s[i]+s[j]<s[j]+s[i] then

begin{交換}

t:=s[i];

s[i]:=s[j];

s[j]:=t;

end;

for i:=1 to n do write(s[i]);

end.

**【練習】**

1、排隊接水

來源程式名：water.pas 可執行檔名：water.exe

輸入檔案名：water.in 輸出檔案名：water.out

問題描述：

有n個人在一個水龍頭前排隊接水，假如每個人接水的時間為Ti，請程式設計找出這n個人排隊的一種順序，使得n個人的平均等待時間最小。

輸入：輸入檔共兩行，第一行為n；第二行分別表示第1個人到第n個人每人的接水時間T1，T2，…，Tn，每個資料之間有1個空格。

輸出：輸出檔有兩行，第一行為一種排隊順序，即1到n的一種排列；第二行為這種排列方案下的平均等待時間（輸出結果精確到小數點後兩位）。

樣例：

water.in

10

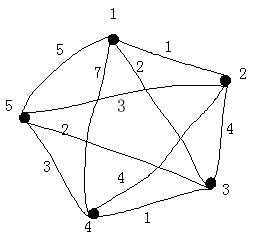
56 12 1 99 1000 234 33 55 99 812

water.out

3 2 7 8 1 4 9 6 10 5

291.90

2、已知5個城市之間有班機傳遞郵件，目的是為了尋找一條耗油量較少的飛行路線。5個城市的聯繫網路如圖所示。圖中編號的結點表示城市，兩個城市之間的連線上的值表示班機沿該航線已行的耗油量，並假定從城市i到j和城市j到i之間的耗油量是相同的。



分析：

① 運用貪心思想：

在每一步前進的選擇上，選取相對當前城市耗油量最小的航線；

② 圖解：若從1出發，有圖：

總耗油量=14 1-2-5-3-4-1

但若路線改為：1-5-3-4-2-1，則總耗油量=13，所以，這樣的貪心法並不能得出最佳解。

③ 改善方案：

從所有城市出發的貪心過程，求最優的。

程式設計：

① 資料結構：

城市聯繫網路圖的描述(圖的鄰接矩陣的描述)：

const

c=array[1..5,1..5] of integer==((0,1,2,7,5), (1,0,4,4,3),(2,4,0,1,2),(7,4,1,0,3), (5,3,2,3,0));

② 貪心過程：

begin

初始化所有城市的算途徑標誌；

設置出發城市V；

for i:=1 to n-1 do {n-1個城市}

begin

s:=從V至所有未曾到過的城市的邊集中耗油量最少的那個城市；

累加耗油量；

V:=s；

設V城市的訪問標誌；

end;

最後一個城市返回第一個城市，累加耗油量；

end;

③ 主過程：實現改善方案

begin

for i:=1 to n do

begin

cost1:=maxint; {初始化}

調用貪心過程，返回本次搜索耗油量cost；

if cost<cost1 then 替換；

end;

輸出；

end.

貪心演算法所作的選擇可以依賴於以往所作過的選擇，但決不依賴於將來的選擇，也不依賴於子問題的解，因此貪心演算法與其它演算法相比具有一定的速度優勢。如果一個問題可以同時用幾種方法解決，貪心演算法應該是比較好的選擇之一。

**第十二章 搜索策略**

深度優先搜索和廣度優先搜索法，以樹的搜索為例，深度優先搜索法是優先擴展尚未擴展的且具有最大深度的結點；廣度優先搜索法是在擴展完第K層的結點以後才擴展K+1層的結點。

深度優先搜索法與前面講的回溯法差不多，主要的區別是回溯法在求解過程中不保留完整的樹結構，而深度優先搜索則記下完整的搜尋樹，搜尋樹起記錄解路徑和狀態判重的作用。為了減少存儲空間，在深度優先搜索中，用標誌的方法記錄訪問過的狀態，這種處理方法使得深度優先搜索法與回溯法沒什麼區別了。在回溯法中，我們己分析了非遞迴的實現過程，在這裡就只討論深度優先的遞迴實現方法。

深度優先搜索的遞迴實現過程：

procedure dfs(i);

for i:=1 to r do

if 子結點mr符合條件then 產生的子結點mr入棧;

if 子結點 mr 是目標結點 then 輸出

else dfs(i+1);

棧頂元素出棧（即刪去mr）;

endif;

endfor;

**【例12-1】** 騎士遊歷

設有一個n\*m的棋盤，在棋盤上任一點有一個中國象棋馬,馬走的規則為:1.馬走日字 2.馬只能向右走。當N,M 輸入之後,找出一條從左下角到右上角的路徑。例如:輸入 N=4,M=4，輸出:路徑的格式:(1,1)->(2,3)->(4,4)，若不存在路徑,則輸出"no"

【**演算法分析**】 我們以4×4的棋盤為例進行分析，馬的4種可能走法（稱為跳法如下圖）設定一個順序，如當前位置在棋盤的（i,j）方格，下一個可能位置依次為（i+2,j+1），（i+2,j-1），（i+1,j+2），（i+1,j-2），而實際上可以走的位置僅限於還未走過的和不越出棋盤邊界的那些位置。為了便於程式統一處理，引入兩個一維陣列dx和dy，分別存儲各種走法相對當前位置的縱橫方向的增量。

|  |  |  |  |
| --- | --- | --- | --- |
|  |  | 3 |  |
|  |  |  | 1 |
|  | 馬 |  |  |
|  |  |  | 2 |
|  |  | 4 |  |

馬從（1，1）開始，按深度優先搜索法，走一步到達（2，3），判斷是否到達終點，若沒有，則繼續往前走，再走一步到達（4，4），然後判斷是否到達終點，若到達則退出，搜索過程結束。為了減少搜索次數，在馬走的過程中，判斷下一步所走的位置是否在棋盤上，如果不在棋盤上，則另選一條路徑再走。

【**參考程式**】

const

dx:array[1..4]of integer=(2,2,1,1);

dy:array[1..4]of integer=(1,-1,2,-2);

type

map=record

x,y:integer;

end;

var

i,n,m:integer;

a:array[0..50]of map;

procedure dfs(i:integer);

var j:integer;

begin

for j:=1 to 4 do

if a[i-1].x+dx[j]>0)and(a[i-1].x+dx[j]<=n)and(a[i-1].y+dy[j]>0)

and(a[i-1].y+dy[j]<=n) then {判斷是否在棋盤上}

begin

a[i].x:=a[i-1].x+dx[j];

a[i].y:=a[i-1].y+dy[j];{入棧}

if (a[i].x=n)and(a[i].y=m)then

begin

write('(',1,',',1,')');

for j:=2 to i do write('->(',a[j].x,',',a[j].y,')');

halt;{輸出結果並退出程式}

end;

dfs(i+1);{搜索下一步}

a[i].x:=0;a[i].y:=0;{出棧}

end;

end;

begin

a[1].x:=1;a[1].y:=1;

readln(n,m);

dfs(2);

writeln('no');

end.

從上面的例子我們可以看出，深度優先搜索演算法有兩個特點：

1、已產生的結點按深度排序，深度大的結點先得到擴展，即先產生它的子結點。

2、深度大的結點是後產生的，但先得到擴展，即“後產生先擴展”，與棧的工作原理

相同，因此用堆疊作為該演算法的主要資料結構，存儲產生的結點。

對於不同的問題，深度優先搜索演算法基本上是一樣的，但在具體處理方法和程式設計技巧上又都不相同，甚至會有很大的差別。我們再看看另一個例子。

**【例12-2】** 選數（存檔名：NOIP2002pj）

**【問題描述】** 已知 n 個整數 x1,x2,…,xn，以及一個整數 k(k＜n＝。從n 個整數中任選 k 個整數相加，可分別得到一系列的和。例如當 n=4，k＝3，4 個整數分別為 3，7，12，19 時，可得全部的組合與它們的和為：3＋7＋12=22　　3＋7＋19＝29　　7＋12＋19＝38　　3＋12＋19＝34。現在，要求你計算出和為素數共有多少種。例如上例，只有一種的和為素數：3＋7＋19＝29。

【**樣例輸入**】 鍵盤輸入，格式為：

n , k (1<=n<=20，k＜n

x1,x2，…,xn （1<=xi<=5000000）

【**樣例輸出**】 螢幕輸出，格式為：-747.2.2

一個整數（滿足條件的種數）。

**【輸入輸出樣例】**

輸入：4 3

　3 7 12 19

輸出：1

【**演算法分析**】本題是求從n個數中選k個數的組合，並使其和為素數。求解此題時，先用深度優先搜索法生成k個數的組合，再判斷k個數的和是否為素數，若為素數則總數加1。

在程式實現過程中，用陣列a存放輸入的n個數，用s表示k個數的和，ans表示和為素數的個數。為了避免不必要的搜索，程式對搜索過程進行了優化，限制搜索範圍，在搜索過程dfs(i,m)中，參數m為第i個數的上限，下限為n-k+i。

來源程式：

var

n,k,i: byte;

ans,s:longint;

a: array[1 .. 20] of Longint;

procedure prime(s:longint);{判斷K個數的和是否為素數}

var

i:integer;

begin

i:=2;

while (sqr(i)<=s)and(s mod i<>0) do inc(i);

if sqr(i)>s then inc(ans){若為素數則總數加1}

end;

procedure dfs(i,m:byte);{搜索第i個數， }

var

j:byte;{j表示第i個數的位置

begin

for j:=m to n-k+i do{枚舉第i個數}

begin

inc(s,a[j]);{入棧}

if i=k then prime(s)

else dfs(i+1,j+1);{繼續搜第i+1個數}

dec(s,a[j]){出棧}

end

end;

begin

readln(n,k);

for i:=1 to n do read(a[i]);

ans:=0; s:=0;

dfs(1,1);

writeln(ans);

end.

從上面的兩個例子我們可以看出，用遞迴實現深度優先搜索比非遞迴更加方便。

在使用深度搜索法解題時，搜索的效率並不高，所以要重視對演算法的優化，盡可能的減少搜索範圍，提高程式的速度。

在深度優先搜索演算法中，深度越大的結點越先得到擴展，若把它改為深度越小的結點越先得到擴展，就是廣度優先搜索法。

廣度優先搜索法

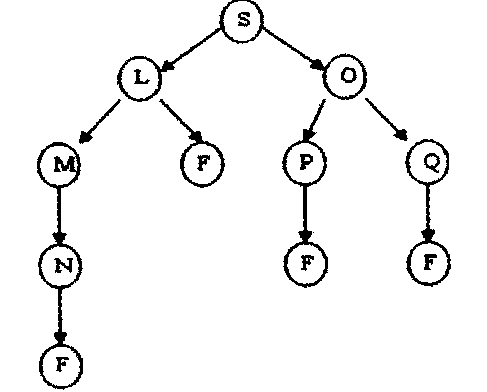
廣度優先搜索的適用範圈

在解決某些最優化問題(如最短路徑)時，用深度優先搜索找最優解必須搜索完所有路徑，即使一個目標結點在搜尋樹很淺的樹枝上，也得等到它所有結點均被搜索後才能找到它。用這種方法求某些最優解時，效率比較低，而且有時一個有解的問題樹可能含有無窮分枝，深度優先搜索如果誤入無窮分枝(即深度無限)，則不可能找到目標結點，因此深度優先搜索策略在解決此類問題時是有局限性的。而廣度優先搜索，能較好地解決這類問題。廣度優先搜索是最簡便和常用的圖形搜索演算法之一，從對圖形的遍歷來看，遵循“從淺入深”的搜索策略。在這種搜索過程中，樹上的結點擴展是沿著深度的“斷層”進行的，所以這種方法一定能保證找到最短(步數最少)的解答序列。在不少題中需要找到經歷步驟最少而達到目標的方案時，多採用此種搜索方法。

廣度優先搜索的原理及演算法實現

為了體現先生成先擴展的執行方式又能保留所有生成的結點以待進一步擴展，廣度優先搜索在資料結構上使用了“佇列”結構。佇列是一種線性表，具有“先進先出”的特點，在佇列中所有的插入和刪除操作分別僅在佇列尾和佇列首進行。定義用一維陣列father來保存“先進先出”的佇列結構，兩個“指標”變數head和tail，分別用來指向佇列的頭和尾。我們將擴展出的狀態存儲在一個稱為state的陣列裡，陣列容量大小為可能的結點數量，定義為maxn。其中state[1..head-1]存儲已擴展狀態(即這些狀態的兒子狀態已擴展出)；state[head..tail]存儲待擴展狀態(即這些狀態的兒子狀態尚待擴展)。初始結點先入隊，首指標指向待擴展結點，每生成一個子結點，則尾指標tail增加1，並且提供指向它們父親結點的指標保存在father陣列中，當前結點的所有子結點均生成後，首指標向後移動(即加1)，位於head指標之前的(已被刪除)為已擴展結點，tail指向所有已生成結點的最後一個。當擴展的子結點state[tail]為目標結點時，根據father教組中所保存的每個元素的父結點在佇列中的序號，通過逆向跟蹤，找到從根結點到目標結點的一條路徑。其次，對新生成的結點還應與已產生的所有結點進行比較，相同結點則不入隊。若head指針大於tail指針，表示所有解答樹上的結點已產生。如果目標結點仍未出現，說明“無解”。

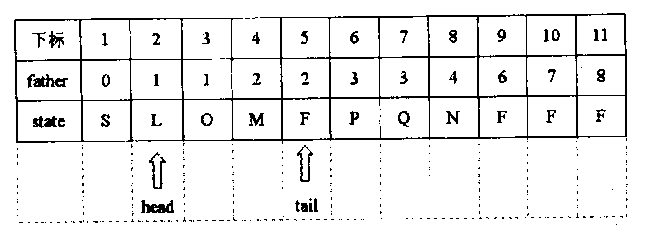
如下圖所示的一棵搜尋樹：



廣度優先搜索結點擴展次序：S L O M F P Q N F F F

深度優先搜索結點擴展次序：S L M N F F O P F Q F

廣度優先搜索的擴展情況如下表：



例如：我們要找的目標結點為F，採用深度優先需要遍歷完所有的結點通過比較才能知道那個目標結點距離根結點最近，而採用廣度優先，首先遍歷的目標結點必定是距離根結點最近的結點，這是廣度優先搜索的重要特點。當父結點L擴展到F時，即表示目標結點已找到，我們可以根據father保存的佇列指標遞迴找出一條路徑，F(2)一>L(1)一>s(0)，當指針為O時，為根結點，遞迴結束，我們要找的路徑為sLF。如果我們要找結點K，擴展完所有結點都無法找到時，這時會出現tail=head，表示“無解”。

廣度優先搜索的演算法描述：

其中，max為產生子結點的規則數：

program BFS

初始化，初始狀態存入state表中；佇列首指標head:=0；尾指針tail:=1；

repeat

指標head後移一位，指向待擴展結點(head:=head+1)；

for i：=1 to max do

begin

if子結點符合條件then

begin

tail指標增1，把新結點存入佇列尾；

if 新結點與原已產生的結點重複then 刪去該結點(取消入隊，tail減1)

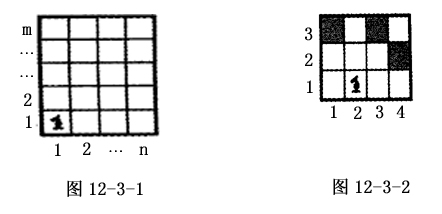
else if 新結點是目標結點 then 輸出並退出；

end

until(head>=tail)；

【**例12-3**】　騎士旅行

【**問題描述**】　在一個n\*m格子的棋盤上，有一隻國際象棋的騎士在棋盤的左下角(1：1)(如圖12-3-1)，騎士只能根據象棋的規則進行移動，要麼橫向跳動一格縱向跳動兩格，要麼縱向跳動一格橫向跳動兩格。例如，n=4，m=3時，若騎士在格子(2：1)(如圖12-3-2)，則騎士只能移入下麵格子：(1：3)，(3：3)或(4：2)；對於給定正整數n，m，i，j值(m，n≤50，i≤n，j≤m)，你要測算出從初始位置(1:1)到格子(i:j)最少需要多少次移動。如果不可能到達目標位置，則輸出“NEVER”。



【**輸入**】　輸入檔的第一行為兩個整數n與m，第二行為兩個整數i與j。

【**輸出**】　輸出檔僅包含一個整數為初始位置(1:1)到格子(i:j)最少移動次數。

【**樣例輸入**】

5 3

1 2

【**樣例輸出**】

3

分析：本題是一個最優化問題(最短路徑)，從題目的資料規模來看，用深度優先搜索在時間複雜度上難以承受，本題的初始結點(1,1)和目標結點(i,j)以及從當前結點擴展到下一結點的規則題目都是明確給出的。由於目標點是隨機給出的，因此無法找到計算最少步數的數學規律，只能通過廣度優先搜索的辦法求解。

【**資料結構**】

設f表示佇列，用來存儲當前結點i的父結點的位置。佇列的首指標為h，尾指針為t。初始時，f[1]=O。

s[i]記錄擴展出的結點i的位置。(s[i,1])表示橫坐標，s[i,2]表示縱坐標。當(s[i,1]，s[i,2])為目標結點時，通過逆向跟蹤，找到從根結點到目標結點的一條路徑。初始時，s[1,1]=1，s[1,2]=1，為初始結點的位置。

a[i,j]用來表示當前位置是否有馬到達過，如a[i,j]=0表示沒有，a[i,j]=1表示已經到達過。

dx、dy——移動後的位置增量陣列。馬有8種不同的擴展方向：

(x-2,y-1)(x-1,y-2)(x-2,y+1)(x-1,y+2)(x+2,y-1)(x+1,y-2)(x+2,y+1)(x+1,y+2)

我們將i方向上的位置增量存八常量陣列dx[i]、dy[i]中(1≤i≤8)

dx：array[1..8]of integer=(-2,-1,-2,-1,2,1,2,1)；

dy：array[1..8]of integer=(-1,-2,1,2,-1,-2,1,2)；

【**約束條件**】

1、馬不能跳出界外，當前結點生成的子結點必須確保在棋盤內。

2、該點在以前的擴展中沒有到達過。如果曾經到達過，則根據廣度優先搜索的原理，先前到達該點所需的步數一定小於當前步數，因此完全沒有必要再擴展下去。

Procedure bfs;

Var

h,t,i:integer;

begin

h:=0;t:=1;s[1,1]:=1;s[1,2]:=1;f[1]:=0;a[1,1]:=1; {初始化}

repeat

inc(h);{指標指向待擴展結點}

for i:=1 to 8 do

if check(s[h,1]+dx[i],s[h,2]+dy[i]) then {判斷當前結點是否可以生成}

begin

inc(t);

s[t,1]:=s[h,1]+dx[i];

s[t,2:=s[h,2]+dy[i];{生成新的子結點}

a[s[t,1],s[t,2]]:=1;{標記當前生成的子結點}

f[t]:=h;{保存父結點指針}

if (s[t,1]=x) and (s[t,2]=y) then {判斷是否為目標結點}

begin

print(t);writeln(total);exit;

end;

until h>=t;

if h=t then write(‘NEVER’);

end.

我們可以用非常簡單的遞迴程式來計算出最小步數。

procedure print(x:integer);

begin

if x=0 then exit;

print(f[x]);

inc(total);

end;

【**例12-4**】 翻幣問題

【**問題描述**】 有N個硬幣(6≤N≤20000)全部正面朝上排成一排，每次將其中5個硬幣翻過來放在原位置，直到最後全部硬幣翻成反面朝上為止。試程式設計找出步數最少的翻法，輸出最少步數。

【**輸入資料**】 輸入一個正整數N(6≤N≤20000)，表示硬幣的數量。

【**輸出資料**】：第1行：一個整數，表示最少步數。

【**輸入**】

6

【**輸出**】

6

【**分析**】本題的初始結點和目標結點都很明確，即正面個數為N，用s[1]=N來表示，目標結點為正面個數為0個，即s[i]=O；而解題的關鍵是找出從當前狀態如何變化到下一狀態(即變化的規律)。

任意翻轉5個硬幣，正反面的個數變化為：

5正0反 正-5 反+5

4正1反 正-3 反+3

3正2反 正-1 反+1

2正3反 正+1 反-1

1正4反 正+3 反-3

0正5反 正+5 反-5

即有6種變化，用s[i]表示結點i正面的個數，完成翻轉即正面的個數為0，在執行上面6種翻轉時要檢查是否符合翻轉條件，即正面的個數和反面的個數要大於其對應的翻轉數，生成新結點時要判斷此結點是否出現過，否則就會出現相同的5個硬幣翻來翻去的情況。

procedure BFS;

var w,k,h,t:integer;

begin

h:=O;t:=1;f[i]:=0;s[1]:=n:

repeat

inc(h);

for w:=0 to 5 do

if(s[h]>=w) and (n-s[h]>=5-w) then

begin

inc(t); f[t]:=h;s[t]:=s[h]-w+5-w; {生成子結點並八佇列}

for k:=1 to t-1 do {判斷當前結點是否出現過,如果出現則刪除}

if s[k]=s[t] then

begin

dec(t):

break;

end;

if s[t]=0 then {判斷當前結點是否為目標結點}

begin

print(t);exit;

end;

end;

until h>=t;

end;

如果需要輸出每次翻幣後的結點狀態，只需把保留的結點狀態輸出即可。

procedure print(v:integer);

begin

if V=0 then exit；

print(f[v]);

writeln(step,’：’，s[v]);

inc(step);

end.

應用廣度優先搜索的注意事項

廣度優先搜索的效率還有賴於目標結點所在位置情況，如果目標結點處在比較深的層上時，需搜索的結點數基本上以指數增長，由於生成的結點數比較多，使用的空間大，容易造成空間溢出，所以在描述結點狀態時要設計好資料結構，儘量做到節省空間。另外可以採用雙向搜索和A演算法對廣度優先的搜索方式進行改良或改造，加入一定的“智慧因素”，使搜索能儘快接近目標結點，減少在空間和時間上的複雜度。我們也可以採用滾動廣搜來節約空間消耗量。影響廣度優先搜索的時間複雜度的另一個重要因素是在生成新結點時要進行判重，花在判重的時間很可觀，所以在狀態總數可以承受的情況下，我們可以採用HasH函數來進行判重，這樣可大大加快搜索速度。

**【例12-5】** 字串變換（NOIP2002tg）

**【問題描述】** 已知有兩個字串 A$, B$ 及一組字串變換的規則（至多6個規則）：

A1$ -> B1$　A2$ -> B2$ 規則的含義為：在 A$中的子串 A1$ 可以變換為 B1$、A2$ 可以變換為 B2$ …。例如：A$＝'abcd'　B$＝'xyz' 變換規則為：‘abc’->‘xu’　‘ud’->‘y’　‘y’->‘yz’　則此時，A$ 可以經過一系列的變換變為 B$，其變換的過程為：‘abcd’->‘xud’->‘xy’->‘xyz’ 共進行了三次變換，使得 A$ 變換為B$。

**【輸入】** 鍵盤輸人檔案名。檔案格式如下：

　　A$ B$

　　A1$ B1$ \

　　A2$ B2$  |-> 變換規則

　　... ... /

　　所有字串長度的上限為 20。

**【輸出】** 輸出至螢幕。格式如下：

　　若在 10 步（包含 10步）以內能將 A$ 變換為 B$ ，則輸出最少的變換步數；否則輸出"NO ANSWER!"

**【輸入輸出樣例】**

b.in:

abcd xyz

abc xu

ud y

y yz

【**螢幕顯示**】 3

【**演算法分析**】 此題是求變換的最少步數，很顯然可以使用廣度優先搜索法，如果直接從初狀態搜到目標狀態，最壞情況下存儲的結點數超過6 10，搜索空間過大，因此我們考慮使雙向搜索，同時從初始狀態和目標狀態向中間狀態搜索，當相遇時搜索結束。採用雙向搜索，存儲的結點數還有可能超限，我們在前向搜索佇列中存儲5步內變換的結點，在後向搜索佇列中，由於第5步產生的結點只是用來與前向佇列中的結點比較，所以可以不存儲在佇列中，後向搜索佇列只需存儲4步內的結點，這樣就解決了存儲空間問題。

為了使用方便，在程式設計中用一個陣列a[1..max]存儲兩個佇列，前向搜索佇列為a[1..mid]，後向搜索佇列為a[mid..max]，用st存儲搜索方向，st=0表示前向搜索，st=1表示後向搜索，用op[st]和cl[st]分別表示佇列尾指標和首指標，用be表示佇列起始位置，迴圈產生每一個結點，若在10內無解退出迴圈，若在10內找到解則輸出解並退出程式。

【**參考程式**】

const mid=12000;max=16000;

type

node=record s:string;x:byte;end;

var

i,mark:integer;

a:array [1..max]of ^node;

x:array[0..6,0..1]of string[20];

d,fil:string;

op,cl:array [0..1] of integer;

procedure Init;{讀取數據，初始化}

var f:text;t:string;

begin

readln(fil);

assign(f,fil);reset(f);i:=0;

while not eof(f) do begin

readln(f,t);

x[i,0]:=copy(t,1,pos(' ',t)-1);

x[i,1]:=copy(t,pos(' ',t)+1,length(t));

inc(i);

end;{while}

mark:=i-1;close(f);

end;

{判斷是否到達目標狀態}

procedure bool(be,st:integer);

begin

for i:=mid-be+1 to cl[1-st] do

if a[cl[st]]^.s=a[i]^.s then begin

writeln(a[cl[st]]^.x+a[i]^.x);

halt;

end;{if}

end;

{判斷節點是否與前面的結點重複}

procedure check(be,st:integer);

begin

for i:=be+1 to cl[st]-1 do

if a[i]^.s=a[cl[st]]^.s then

begin dec(cl[st]);exit; end;

bool(be,st);

end;

{擴展產生新節點}

procedure expand(be,st:integer);

var i,j,k,lx,ld:integer;

begin

inc(op[st]);d:=a[op[st]]^.s;

k:=a[op[st]]^.x;ld:=length(d);

for i:=1 to mark do begin

lx:=length(x[i,st]);

for j:=1 to ld do begin

if (copy(d,j,lx)=x[i,st]) then begin

if (st<>1)or(k<>4)then begin

inc(cl[st]);

new(a[cl[st]]);

end;{if}

a[cl[st]]^.s:= copy(d,1,j-1)+ x[i,1-st]+ copy(d,j+lx,ld);

a[cl[st]]^.x:=k+1;

check(be,st);{檢查是否重複}

end;{if}

end;{for}

end;{for}

end;

procedure bfs;

var be,k,st:integer;

Begin

for st:=0 to 1 do begin

if st=0 then be:=0 else be:=mid;

op[st]:=be+0;cl[st]:=be+1;

new(a[cl[st]]);

a[cl[st]]^.s:=x[0,st];

a[cl[st]]^.x:=0;

end;{for}

repeat

if (op[0]<cl[0])and(a[cl[0]]^.x<=5)then expand(0,0);

if (op[1]<cl[1])and(a[cl[1]]^.x<=5)then expand(mid,1);

until(op[0]>=cl[0])or(a[cl[0]]^.x>5)or(op[1]>=cl[1])or (a[cl[1]]^.x>5);

End;

BEGIN

init;bfs;writeln('NO ANSWER!')

END.

兩種搜索演算法的比較：

|  |  |  |  |
| --- | --- | --- | --- |
| 搜索方式 | 擴展方式 | 資料結構 | 適合求解的問題 |
| 深度優先 | 後產生先擴展 | 棧 | 可行解或所有解 |
| 廣度優先 | 先產生先擴展 | 佇列 | 最優解 |

在選擇搜索方式時，並不是完全遵循以上原則，具體還是要根據題目的要求而定。在求最優解時，如果搜索的深度不大，我們也可以考慮使用深度優先搜索；在求解可行解時，如果搜索的深度沒有限制，或者搜索的代價與搜索的深度成正比,我們也應該使用廣度優先搜索。

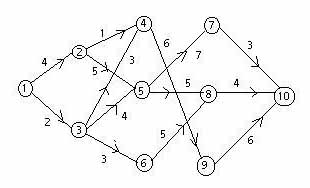
**第十三章 動態規劃**

13．1動態規劃問題的數學描述

我們先來看一個簡單的多階段決策問題。

【**例13-1**】 現有一張地圖，各結點代表城市，兩結點間連線代表道路，線上數字表示城市間的距離。如圖1所示，試找出從結點1到結點10的最短路徑。

第一階段 第二階段 第三階段 第四階段 第五階段

 圖1

本問題的解決可採用一般的窮舉法，即把從結點1至結點10的所有道路列舉出來，計算其長度，再進行比較，找出最小的一條。雖然問題能解決，但採用這種方法，當結點數增加，其運算量將成指數級增長，故而效率是很低的。

分析圖1可知，各結點的排列特徵：

1、可將各結點分為5個階段；

2、每個階段上的結點隻跟相鄰階段的結點相連，不會出現跨階段或同階段結點相連的情況，如不會出現結點1與結點4連、結點4與結點5連的情況。

3、除起點1和終點10外，其它各階段的結點既是上一階段的終點，又是下一階段的起點。例如第三階段的結點4、5、6，它即是上一階段結點2、3中某結點的終點，又是下一階段結點7、8、9中某結點的起點。

根據如上特徵，若對於第三階段的結點5，選擇1-2-5和1-3-5這兩條路徑，後者的費用要小於前者。那麼考慮一下，假設在所求的結點1到結點10最短路徑中要經過結點5，那我們在結點1到結點5之間會取那條路徑呢？顯然，無論從結點5出發以後的走法如何，前面選擇1-3-5這條路都總是會優於1-2-5的。也就是說，當某階段結點一定時，後面各階段路線的發展不受這點以前各階段的影響。反之，到該點的最優決策也不受該點以後的發展影響。

由此，我們可以把原題所求分割成幾個小問題，從階段1開始，往後依次求出結點1到階段2、3、4、5各結點的最短距離，最終得出答案。在計算過程中，到某階段上一結點的決策，只依賴於上一階段的計算結果，與其它無關。例如，已求得從結點1到結點5的最優值是6，到結點6的最優值是5，那麼要求到下一階段的結點8的最優值，只須比較min{6+5,5+5}即可。這樣，運用動態規劃思想大大節省了計算量。可以看出，動態規劃是解決此類多階段決策問題的一種有效方法。

13．2動態規劃中的主要概念，名詞術語

1、階段：把問題分成幾個相互聯繫的有順序的幾個環節，這些環節即稱為階段。

2、狀態：某一階段的出發位置稱為狀態。通常一個階段包含若干狀態。如圖1中，階段3就有三個狀態結點4、5、6。

3、決策：從某階段的一個狀態演變到下一個階段某狀態的選擇。

4、策略：由開始到終點的全過程中，由每段決策組成的決策序列稱為全過程策略，簡稱策略。

5、狀態轉移方程：前一階段的終點就是後一階段的起點，前一階段的決策選擇匯出了後一階段的狀態，這種關係描述了由k階段到k+1階段狀態的演變規律，稱為狀態轉移方程。

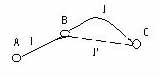
6、目標函數與最優化概念：目標函數是衡量多階段決策過程優劣的準則。最優化概念是在一定條件下找到一個途徑，經過按題目具體性質所確定的運算以後，使全過程的總效益達到最優。

13．3、運用動態規劃需符合的條件

任何思想方法都有一定的局限性，超出了特定條件，它就失去了作用。同理，動態規劃也並不是萬能的。那麼使用動態規劃必須符合什麼條件呢？必須滿足最優化原理和無後效性。

1、最優化原理

最優化原理可這樣闡述：一個最優化策略具有這樣的性質，不論過去狀

 圖2

態和決策如何，對前面的決策所形成的狀態而言，餘下的諸決策必須構成最優策略。簡而言之，一個最優化策略的子策略總是最優的。

如圖2中，若路線I和J是A到C的最優路徑，則根據最優化原理，路線J必是從B到C的最優路線。這可用反證法證明：假設有另一路徑J’是B到C的最優路徑，則A到C的路線取I和J’比I和J更優，這與原名題矛盾。從而證明J必是B到C的最優路徑。

最優化原理是動態規劃的基礎，任何問題，如果失去了最優化原理的支援，就不可能用動態規劃方法計算。

2、無後效性

“過去的步驟只能通過當前狀態影響未來的發展，當前的狀態是歷史的總結”。這條特徵說明動態規劃只適用於解決當前決策與過去狀態無關的問題。狀態，出現在策略任何一個位置，它的地位相同，都可實施同樣策略，這就是無後效性的內涵。

由上可知，最優化原理，無後效性，是動態規劃必須符合的兩個條件。

4、動態規劃的計算方法

對於一道題，怎樣具體運用動態規劃方法呢？

（1）首先，分析題意，考察此題是否滿足最優化原理與無後效性兩個條件。

（2）接著，確定題中的階段，狀態，及約束條件。

（3）推導出各階段狀態間的函數基本方程，進行計算。

13．4 怎樣用動態規劃法解題

用動態規劃法解題主要有兩種方法：順推動態規劃法和逆推動態規劃法。

所謂順推動態規劃法，指的是從起點出發，層層遞推，直到終點，逆推動態規劃法指的是從終點出發，逆向求解。這兩種方法本質上是一樣的，具體解題時，可根據實際情況來選用。

【**例13-2**】 排隊買票

問題描述：一場演唱會即將舉行。現有N（O〈N〈=200〉個歌迷排隊買票，一個人買一張，而售票處規定，一個人每次最多只能買兩張票。假設第I位元歌迷買一張票需要時間Ti（1〈=I〈=n〉，隊伍中相鄰的兩位歌迷（第j個人和第j+1個人）也可以由其中一個人買兩張票，而另一位就可以不用排隊了，則這兩位歌迷買兩張票的時間變為Rj，假如Rj〈Tj+Tj+1，則這樣做就可以縮短後面歌迷等待的時間，加快整個售票的進程。現給出N，Tj和Rj，求使每個人都買到票的最短時間和方法。

解決問題：本題的階段十分明顯，只要按排隊的先後順序劃分即可。而買票的方式只有兩種，要麼一人買一張，要麼一人買兩張，整個過程呈線性排列。要使前I個人買票時間最短，只需考慮前I個人的買票方式，與佇列以後的人無關。而且顯而易見，在最優策略中，任意m個連續的決策也肯定是最優的。這樣，問題就符合了最優化原理及無後效性，能運用動態規劃。那如何構造函數遞推式呢？可以以到每個人為止所需的最短時間為狀態值，設為f（i），於是有f（i）=min{f（i-1）+Ti，f（i-2）+Ri-1}，起步時f（0）=0，f（1）=T1 。如此從前往後，只需遍歷一次即可。

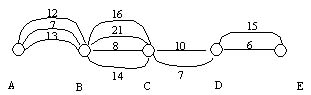
上面的分析是從前往後進行的。其實倒過來逆推也一樣。設f（I）表示當票賣到第I個人時，最少還需多少時間才能賣完。則函數遞推式為f（I）=min{f（I+1）+Ti，f（I+2）+Ri}，從後往前逆推，起步時f（n）=Tn，f（n+1）=0 。

採用順推還是逆推動態規劃，要看實際情況而定，哪一種直觀、簡便，就運用哪一種。

13．5、動態規劃適用的原則

動態規劃適用的原則為優化原則，即整體優化可以分解為若干個局部優化。下面通過例子來說明動態規劃適用的原則。

【**例13-3**】 求城市間最短通路問題。設有n個城市分佈有一條幹道上，相鄰的城市之間有若干條通路，每條通路上的數字表示通咱的距離。如下圖。求出從A到E的最短距離。



從上圖可知，從A到B共有3條通路，其距離分別為12、7和13；從B到C共有4條通路，其距離分別為16、21、8、14；從C到D共有2條通路，其距離分別為10和7；從D到E共有2條通路，其距離分別為15和6。

首先考慮用窮舉法求解。窮舉法的過程為先求出A到E的所有路徑，再找出一條最短的路徑，其最短路徑的距離即為所求。

從A到E路徑的條數，根據乘法公式為3×4×2×2=48條。對於簡單的情況，窮舉法是可能的，但是對複雜的情況，窮舉法就不可能了。例如，從A到E之間經過20個城市，每2個城市之間平均有5條通路，此時，路徑的條數為215，這是一個非常大的數，因此窮舉法在這種情況下求解是不可能的。

下面考慮用動態規劃求解，有D（A，E）表示城市A、E之間的最短距離，由加法原理可知

D（A，E）= D（A，B）+D（B，C）+D（C，D）+D（D，E）

而相鄰城市之間的通路是有限的，上式的意義為城市A、E間的最短距離，等於相鄰城市間最短距離之和。假設從A到E中間有50個城市，每2個相鄰城市之間的平均通路有100條，總的查找工作量為51×100，這是一比較小的數，在電腦上運行時幾乎不用時間。

【**例13-4**】 求最長不下降序列。

【**問題描述**】 設有一個正整數的序列：b1，b2，…，bn，對於下標i1＜i2＜…＜iL，若有bi1≤bi2≤…≤biL則稱存在一個長度為L的不下降序列。

例如，下列數列

13 7 9 16 38 24 37 18 44 19 21 22 63 15

對於下標i1=1，i2=4，i3=5，i4=9，i5=13，且滿足13＜16＜38＜44＜63，則存在長度為5的不下降序列。但是，我們看到還存在其他的不下降序列。

i1=2，i2=3，i3=4，i4=8，i5=10，i6=11，i7=12，i8=13，且滿足7＜9＜16＜18＜19＜21＜22＜63，則存在長度為8的不下降序列。

問題為：當b1，b2，…，bn給出之後，求出最長的不下降序列。

【**資料結構**】

n：表示數列的個數。

b:array[1..n,1..3] of integer;

其中：

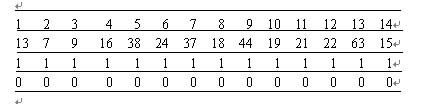
b[i,1]表示第i項的數值本身。

b[i,2]表示從第i項到最後一項最長不下降序列的長度。

b[i,3]連結字，表示最長不下降序列經過此項之後，後面繼續的項。當b[i,3]=0時表示連結結束。

【**演算法分析**】

以上面的數列為例，構造出一個二維陣列。



上表第一行表示下標，即資料項目的位置；第二行表示資料項目本身的數值；第三行為長度，長度為1表示最長的長度為1，即數本身；第四行為連結字，0表示無連結。

下面給出從後向前的求解過程：

①、首先從倒數第二項開始計算，由於在它的後面僅有一項，因此僅僅作一次比較，由於63>15，所以從63出發，不能作任何連結，長度仍為1。

②、再看倒數第三項22，在它的後面有2項，因此，必須在後面的2項中，找出比它大的，最長的作為連結，修改22的長度和連結指標。

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | 11 | 12 | 13 | 14 |
|  |  | 22 | 63 | 15 |
|  |  | 2 | 1 | 1 |
|  |  | 13 | 0 | 0 |

③、再到數項21，在它的後面有3項，因此必須在後面的3項中，找出比它大的，且為最長的作為連結，由於22，63都大於21，但22的長度為2，63的長度為1，所以選擇22作為它的連結。連結後，21的長度為3，連結字為12，表示到21之後，還有資料項目22，到了22之後，還有63，到了63之後，由於連結字為0，所以連結結束。

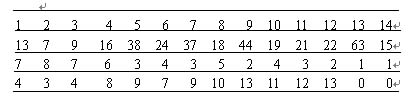
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | 11 | 12 | 13 | 14 |
|  | 21 | 22 | 63 | 15 |
|  | 3 | 2 | 1 | 1 |
|  | 12 | 13 | 0 | 0 |

【**一般項i過程為**】

①在i+1,i+2,…,n項中，找出比b[i,1]大的最長長度L，及最長長度的位置K；

②L>0，則b[i,2]:=L+1; b[i,3]:=K。

經過上面的計算之後，前面的表成為



【**結果輸出**】

①首先在所有的b[i,2]中找出最大者(上表中的b[2,2]=8)，輸出最大長度，並用L：=2，記錄最大長度位置。

②輸出最長的序列，演算法為：

while L<>0 do begin write(b[L,1]);L:= b[L,3];end;

program buxiajiang;

const n=10;

var i,j,k,L:integer;

b:array[1..n,1..3] of integer;

begin

for i:=1 to n do begin readln(b[i,1]); b[i,2]:=1; b[i,3]:=0;end;

for i:=n-1 downto1 do

begin L:=0;k:=0;

for j:=i+1 to n do

if (b[j,1])> b[i,1]) and (b[j,2]>L) then begin L:= b[j,2];k:=j;end;

if L>0 then begin b[i,2]:=L+1; b[i,3]:=k;end;

end;

L:=1;

for j:=2 to n do

If b[j,2]>b[L,2] then L:=j;

writeln(‘max=’, b[L,2]);

while L<>0 do begin write(b[L,1]);L:= b[L,3];end;

end.

【**例13-5**】 合唱隊形（noip2004tg）

【**問題描述**】 N位同學站成一排，音樂老師要請其中的(N-K)位同學出列，使得剩下的K位同學排成合唱隊形。合唱隊形是指這樣的一種隊形：設K位同學從左到右依次編號為1, 2, …, K，他們的身高分別為T1, T2, …, TK，則他們的身高滿足T1 < T2 < … < Ti , Ti > Ti+1 > … > TK (1 <= i <= K)。你的任務是，已知所有N位同學的身高，計算最少需要幾位元同學出列，可以使得剩下的同學排成合唱隊形。

【**輸入檔**】 輸入檔chorus.in的第一行是一個整數N（2 <= N <= 100），表示同學的總數。第一行有n個整數，用空格分隔，第i個整數Ti（130 <= Ti <= 230）是第i位同學的身高（釐米）。

【**輸出檔**】 輸出檔chorus.out包括一行，這一行只包含一個整數，就是最少需要幾位元同學出列。

【**樣例輸入**】 8

186 186 150 200 160 130 197 220

【**樣例輸出**】 4

【**演算法分析**】 此題採用動態規劃法求解。先分別從左到右求最大上升子序列，從右到左求最大下降子序列，再枚舉中間最高的一個人。演算法實現起來也很簡單，時間複雜度O(N2 )。

我們先考慮如何求最大上升子序列的長度，設f1(i )為前i個同學的最大上升子序列長度。若要求f1(i)，必須先求得f1(1),f1(2),…,f1(i-1)，再選擇一個最大的f1(j)（j<i），在前j個數中的最大上昇冪後添加Ti，就可得到前i個數的最大上升子序列f1(i)。這樣就得到狀態轉移方程：

f1(i)=max{f1(j)+1} (j<i,Tj<Ti)

邊界條件：f1(1)=1;

設f2(i)為後面N-i+1位排列的最大下降子序列長度，用同樣的方法可以得到狀態轉移方程：f2(i)=max{f2(j)+1} (i<j,Tj<Ti)；邊界值為f2(N)=1;

有了狀態轉移方程，程式實現就非常容易了。

【**參考程式**】

var

t,f1,f2:array[1..100]of byte;

i,j,n,max:integer;

begin

assign(input,'chorus.in');

reset(input); readln(n);

for i:=1 to n do begin

read(t[i]);f1[i]:=1;f2[i]:=1;

end;{for}

close(input); max:=0;

for i:=2 to n do

for j:=1 to i-1 do begin

if (t[i]>t[j])and(f1[j]>=f1[i]) then f1[i]:=f1[j]+1; {從左到右求最大上升子序列}

if (t[n-i+1]>t[n-j+1])and(f2[n-j+1]>=f2[n-i+1]) then f2[n-i+1]:=f2[n-j+1]+1; {從右到左求最大下降子序列}

end;{for}

for i:=1 to n do if max<f1[i]+f2[i] then max:=f1[i]+f2[i]; {枚舉中間最高的}

assign(output,'chorus.ans');

rewrite(output);

writeln(n-max+1);

close(output);

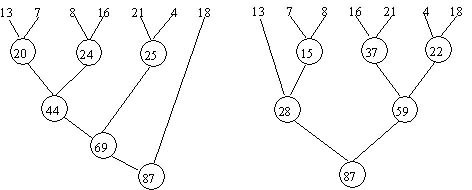
end.

運用動態規劃法求解問題的關鍵是找出狀態轉移方程，只要找出了狀態轉移方程，問題就解決了一半，剩下的事情就是解決如何把狀態轉移方程用程式實現。

【**例13-6**】 最小代價子母樹

【**問題描述**】設有n堆沙子排成一排，其編號為1，2，3，…，n（n≤100）。每堆沙子有一定的數量，如下表：13 7 8 16 21 4 18

現在要將n堆沙子歸併成為一堆。歸併的過程為每次只能將相鄰的兩堆沙子堆成一堆，這樣經過n-1次歸併之後最後成為一堆，如上面7堆沙子，可以有多種方法歸併成一堆，其中的2種方法如下圖。



歸併的代價是這樣定義的，將兩堆沙子歸併為一堆時，兩堆沙子數量的和稱為歸併兩堆沙子的代價。如上圖，將13和7歸併為一堆的代價為20。歸併總代價指的是將沙子全部歸併為一堆沙子的代價的和，如上面兩種歸併方法中，第一種的總代價為：20+24+25+44+69+87=267；第二種的總代價為：15+37+22+28+59+87=248。

由此可見，不同歸併過程得到的總的歸併代價是不一樣的。

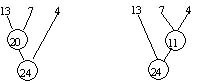
問題：當n堆沙子的數量給出之後，找出一種合理的歸併方法，使總的歸併代價為最小。

【**演算法分析**】

為了說明演算法的過程，我們先分析一些簡單的情況。

（1）n=2，僅有一種堆法，因此總的歸併代價為2堆沙子數量的和。

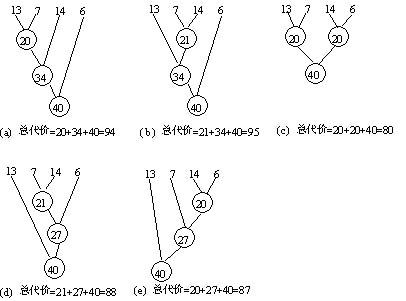
（2）n=3



有兩種堆法。從上圖可看到它們的總的歸併代價分別為44和35。下面我們分析產生這種現象的原因。

第一種堆法的總代價為20+24=44，第二種堆法的總代價為11+24=35。由此可見，最後一次歸併的代價為全部沙子數量的和，對任何歸併方案都是相同的，因此，總代價的大小將取決於第一次歸併，第一種方法的第一次歸併代價為20，第二種方法的第一次歸併代價為11，因此第二種方法比第一種方法好。

（3）n=4



當n=4時，共有5種歸併的方法，它們的總代價分別如上圖所示，最小的歸併總代價為(c)80。上面的5種方法可以分為三類。

第一類包括（a），（b），基本方法是歸併前面的三堆，再歸併最後的一堆，由於最後一堆歸併的代價是相同的，所以在歸併前面三堆時不同的方法將產生不同的結果。（a）中歸併三堆的代價為54，（b）中歸併三堆的代價為55，所以第一類方法中的（a）比（b）優。

第二類僅有一種方法（c），分別歸併2堆的代價為20，20，相加為40。

第三類包括（d），（e），基本方法是先歸併後面的三堆，再歸併第一堆，由於最後一次歸併代價是相同的，所以歸併後三堆的方法不同將產生不同的結果，由上面的分析可知，方法（e）比方法（d）優。

引入記號f(i,j)表示從i堆到j堆沙子的歸併最小代價數。從上面討論可知：

f（1，4）=min{ f（1，3），f（1，2）+f（3，4），f（2，4）}+40

而f（1，3）=min{ f（1，2），f（2，3）}+34

f（1，2）=20

f（3，4）=20

f（2，3）=21

f（2，4）=min{ f（2，3），f（3，4）}+27

引入記號g（i，j）表示第i堆沙子到j堆沙子的數量和，如圖，g（1，1）=13，g（1，2）=20，…，g（1，4）=40。

一般情況的分析

f（1，n）=min{ f（1，n-1），f（1，2）+ f（3，n），f（1，3）+ f（4，n），…，f（2，n）}+g（1，n）

【**資料結構**】

說明：

h:array[1..n] of integer;表示每堆沙子重量。

g:array[1..n,1..n] of integer;

g[i,j]表示從i堆到j堆沙子重量的和。

f:array[1..n,1..n] of integer;

f[i,j]表示第i層從第j個位置開始j堆沙子最小歸併代價，並約定f[1,i]=0。

演算法流程：

①初始化

for i:=1 to n do readln(h[i]);{輸入沙子數量}

for i:=1 to n do

for j:=1 to n do

begin s:=0;

for L:=i to j do s:=s+h[L];

g[i,j]:=s;

end;

for i:=1 to n do

for j:=1 to n do f[i,j]:=0;

②求解

for i:=2 o n do {i展示層次，即歸併沙子的堆數}

for j=1 to n-i+1 do {j表示歸併沙子的起始位置}

begin

min:=f[i-1,j];

for L:=i-2 downto 1 do

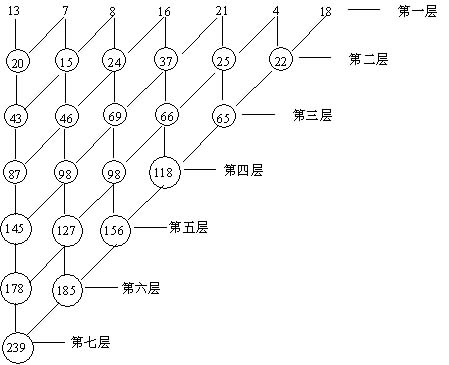
if min>f[L,j]+f[i-L,j+L] then min:= f[L,j]+f[i-L,j+L];

f[i,j]:=g[i,j]+min;

end;

③輸出

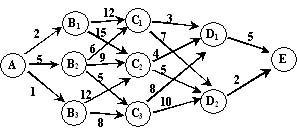
writeln(f[n,1]);



注：數值為最小代價

【**例13-7**】 最短路徑問題

下圖表示城市之間的交通路網，線段上的數字表示費用，單向通行由A->E。試用動態規劃的最優化原理求出A->E的最省費用。



如圖從A到E共分為4個階段，即第一階段從A到B，第二階段從B到C，第三階段從C到D，第四階段從D到E。除起點A和終點E外，其它各點既是上一階段的終點又是下一階段的起點。例如從A到B的第一階段中，A為起點，終點有B1，B2，B3三個，因而這時走的路線有三個選擇，一是走到B1，一是走到B2，一是走到B3。若選擇B2的決策，B2就是第一階段在我們決策之下的結果，它既是第一階段路線的終點，又是第二階段路線的始點。在第二階段，再從B2點出發，對於B2點就有一個可供選擇的終點集合(C1，C2，C3)；若選擇由B2走至C2為第二階段的決策，則C2就是第二階段的終點，同時又是第三階段的始點。同理遞推下去，可看到各個階段的決策不同，線路就不同。很明顯，當某階段的起點給定時，它直接影響著後面各階段的行進路線和整個路線的長短，而後面各階段的路線的發展不受這點以前各階段的影響。故此問題的要求是：在各個階段選取一個恰當的決策，使由這些決策組成的一個決策序列所決定的一條路線，其總路程最短。如何解決這個問題呢？

【**決策過程**】

（1）由目標狀態E向前推，可以分成四個階段，即四個子問題。如上圖所示。

（2）策略：每個階段到E的最省費用為本階段的決策路徑。

（3）D1，D2是第一次輸人的結點。他們到E都只有一種費用，在D1框上面標5，D2框上面標2。目前無法定下，那一個點將在全程最優策略的路徑上。第二階段計算中，5，2都應分別參加計算。

（4）C1，C2，C3是第二次輸入結點，他們到D1，D2各有兩種費用。此時應計算C1，C2，C3分別到E的最少費用。

C1的決策路徑是 min{（C1D1），（C1D2）}。計算結果是C1+D1+E，在C1框上面標為8。

同理C2的決策路徑計算結果是C2+D2+E，在C2框上面標為7。

同理C3的決策路徑計算結果是C3+D2+E，在C3框上面標為12。

此時也無法定下第一，二階段的城市那二個將在整體的最優決策路徑上。

（5）第三次輸入結點為B1，B2，B3，而決策輸出結點可能為C1，C2，C3。仿前計算可得Bl，B2，B3的決策路徑為如下情況。

Bl:B1C1費用 12+8=20， 路徑:B1+C1+D1+E

B2:B2C1費用 6+8=14， 路徑:B2+C1+D1+E

B3:B2C2費用 12+7=19， 路徑:B3+C2+D2+E

此時也無法定下第一，二，三階段的城市那三個將在整體的最優決策路徑上。

（6）第四次輸入結點為A，決策輸出結點可能為B1，B2，B3。同理可得決策路徑為

A：AB2，費用5+14=19，路徑 A+B2+C1+D1+E。

此時才正式確定每個子問題的結點中，那一個結點將在最優費用的路徑上。19將結果顯然這種計算方法，符合最優原理。子問題的決策中，只對同一城市（結點）比較優劣。而同一階段的城市（結點）的優劣要由下一個階段去決定。

【**參考程式**】

採用逆推法。

program shortroad(input,output);

const max=100;

var

n,st,en, {頂點數，起點，終點}

i,j,x : integer;

way : array[1..max,1..max] of integer; {線路網路的帶權矩陣}

netdot : array[1..max] of record {netdot[j].ne--從j點出發的決策}

ne,cost:integer {netdot[j].cost--從j點至en點的最短距離}

end;

f : text; {檔變數}

str : string; {檔案名串}

begin

write('file name=',str); {讀入檔案名串}

readln(str);

assign(f,str); {檔案名與檔變數連接}

reset(f); {讀文件準備}

readln(f,n,st,en); {讀入頂點數，起點，終點}

writeln('n=',n:4,'st=',st:4,'en=',en:4);

for i:= 1 to n do {讀入各點間連線的距離，無邊相連用－1表示}

for j:= 1 to n do read(f,way[i,j]);

close(f);

for i:= 1 to n do netdot[i].cost:=maxint; {st至各點的最短路徑長度初始化}

netdot[en].cost:=0; netdot[en].ne :=0; {從最後一段開始，由後何前逐步遞推}

for j:= n downto 1 do {以下雙重迴圈逆推的求解}

for x:= n downto j do

if ( way[j,x] >0 ) and ( netdot[x].cost <> maxint )

and (way[j,x]+netdot[x].cost<netdot[j].cost ) then

{若en至x點的最短路己經求出，且加入邊(j，x)後使得en至j的路徑目前最短}

begin

netdot[j].cost:=netdot[x].cost+way[j,x] ; {則記下最短路徑長度}

netdot[j].ne:=x;

end;

if netdot[st].cost=maxint then writeln('noway')

else

begin {否則從起點出發，按計算順序反推最短路徑}

x:=st;

writeln(netdot[x].cost); {輸出最短路徑長度}

while x<>0 do begin

write(x:3,' ');

x:=netdot[x].ne;

end;

end;

end.

【**求解示意圖**】

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 結點 | A1 | B1 | B2 | B3 | C1 | C2 | C3 | D1 | D2 | E |
| 編號 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| 最短長度 | 18 | 20 | 14 | 19 | 8 | 7 | 12 | 5 | 2 | 0 |
| 決策結點 | 3 | 5 | 5 | 7 | 8 | 9 | 9 | 10 | 10 | 0 |

輸入檔案名為q01.txt其內容如下：

10 1 10

0 2 5 1 -1 -1 -1 -1 -1 -1

-1 0 -1 -1 12 14 -1 -1 -1 -1

-1 -1 0 -1 6 10 4 -1 -1 -1

-1 -1 -1 0 -1 12 11 -1 -1 -1

-1 -1 -1 -1 0 -1 -1 3 9 -1

-1 -1 -1 -1 -1 0 -1 6 5 -1

-1 -1 -1 -1 -1 -1 0 -1 10 -1

-1 -1 -1 -1 -1 -1 -1 0 -1 5

-1 -1 -1 -1 -1 -1 -1 -1 0 2

-1 -1 -1 -1 -1 -1 -1 -1 -1 0

運行結果如下：

file name=q01.txt

1  3  5  8  10

【**例13-8**】 背包問題

背包問題有三種：

1、部分背包問題（每件物品只有一件，但物品可分割）

一個旅行者有一個最多能裝m公斤的背包，現在有n種物品，它們的總重量分別是W1，W2，...,Wn,它們的總價值分別為C1,C2,...,Cn。若從這n件物品中任取若干件（某件物品裝不下時可分割，部分裝入），問背包中裝入哪些物品可使得所裝物品的價值和最大？

解決問題的方法是貪心演算法:將C1/W1,C2/W2,...Cn/Wn,從大到小排序,不停地選擇價值與重量比最大的放人背包直到放滿為止。

2、0/1背包 （每件物品只有一件，要麼選取，要麼不選取）

一個旅行者有一個最多能裝m公斤的背包，現在有n件物品，它們的重量分別是W1，W2，...,Wn,它們的價值分別為C1,C2,...,Cn。若從這n件物品中任取若干件（這些物品要麼被裝入要麼被留下）。問背包中裝入哪些物品可使得所裝物品的價值和最大？

【**決策過程**】

顯然這個題可用深度優先方法對每件物品進行枚舉(選或不選用0、1控制)。程式簡單,但是當n的值很大的時候不能滿足時間要求，時間複雜度為O（2n）。按遞迴的思想我們可以把問題分解為子問題,使用遞迴函數。

設 f(i，x)表示前i件物品，總重量不超過x的最優價值

則 f（i，x）=max(f(i－1，x-w[i])+c[i]，f（i－1，x））

f（n，m）即為最優解，邊界條件為f（0，x）＝0 ，f(i,0)=0;

這個方程非常重要，基本上所有跟背包相關的問題的方程都是由它衍生出來的。所以有必要將它詳細解釋一下：“將前i件物品放入容量為x的背包中”這個子問 題，若只考慮第i件物品的策略（放或不放），那麼就可以轉化為一個隻牽扯前i-1件物品的問題。如果不放第i件物品，那麼問題就轉化為“前i-1件物品放入容量為x的背包中”，價值為f（i-1，x）；如果放第i件物品，那麼問題就轉化為“前i-1件物品放入剩下的容量為x-w[i]的背包中”，此時能獲得的最大價值就是f(i－1，x-w[i])再加上通過放入第i件物品獲得的價值c[i]。

【**輸入資料**】

n 4 m 11

w[i] 2 4 6 7

c[i] 6 10 12 13

【**輸出資料**】

0 1 0 1 (選取用1表示，沒選取用0表示)

23 （最大總價值）

【**求解示意圖**】

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 重量w[i] | 2 | 4 | 6 | 7 |
| 價值c[i] | 6 | 10 | 12 | 13 |
| 狀態（放入） | 6 | 16 | 22 | 23 |
| 狀態（不放入） | 0 | 6 | 16 | 22 |

【**參考程式**】(順推法)

program 0-1beibao;

const maxm=200;maxn=30;

type ar=array[1..maxn] of integer;

var m,n,j,i:integer;

c,w:ar;

f:array[0..maxn,0..maxm] of integer;

function max(x,y:integer):integer;

begin

if x>y then max:=x else max:=y;

end;

begin

readln(n,m);

for i:=1 to n do read(w[i]);readln;

for i:=1 to n do read(c[i]);

for i:=1 to m do f[0,i]:=0;

for i:=1 to n do f[i,0]:=0;

for i:=1 to n do

for j:=1 to m do

begin

if j>=w[i] then f[i,j]:=max(f[i-1,j-w[i]]+c[i],f[i-1,j])

else f[i,j]:=f[i-1,j];

end;

writeln(f[n,m]);

補上輸出“取最大價值時所選取的物品編號”；

end.

使用二維陣列存儲各子問題時方便，但當maxm較大時如maxn=2000時不能定義二維陣列f,怎麼辦,其實可以用一維陣列,但是上述中j:=1 to m 要改為j:=m downto 1,為什麼?請大家自己解決。

3、完全背包問題（物品有無數件）

一個旅行者有一個最多能裝m公斤的背包，現在有n種物品，每件的重量分別是W1，W2，...,Wn,

每件的價值分別為C1,C2,...,Cn。若從這n件物品中任取若干件（每種物品的件數足夠多）。問背包中裝入哪些物品可使得所裝物品的價值和最大？

【**輸入資料**】

第一行兩個數：物品總數n，背包載重量m；兩個數用空格分隔；

第二行n個數,為n種物品重量；兩個數用空格分隔；

第三行n個數,為n種物品價值; 兩個數用空格分隔；

【**輸出資料**】

第一行總價值；

以下n行，每行兩個數，分別為選取物品的編號及數量；

【**輸入樣例**】

n 4 m 13

w[i] 2 3 4 7

c[i] 1 3 5 9

【**輸出樣例**】

15 （最大總價值）

2 2

4 1

【**數學模型**】

設 f(x)表示重量不超過x公斤的最大價值，則 f(x)=max{f(x-w[i])+c[i]}  當x>=w[i]， 1<=i<=n

【**求解示意圖**】

i=1時 →f[1]=0；

i=2時 f[0]+1 →f[2]=1

i=3時 max{f[1]+1,f[0]+3} →f[3]=3

i=4時 max{f[2]+1,f[1]+3,f[0]+5} →f[4]=5

i=5時 max{f[3]+1,f[2]+3,f[1]+5} →f[5]=5

i=6時 max{f[4]+1,f[3]+3,f[2]+5} →f[6]=6

i=7時 max{f[5]+1,f[4]+3,f[3]+5,f[0]+9} →f[7]=9

i=8時 max{f[6]+1,f[5]+3,f[4]+5,f[1]+9} →f[8]=10

i=9時 max{f[7]+1,f[6]+3,f[5]+5,f[2]+9} →f[9]=10

i=10時 max{f[8]+1,f[7]+3,f[6]+5,f[3]+9} →f[10]=12

i=11時 max{f[9]+1,f[8]+3,f[7]+5,f[4]+9} →f[11]=14

i=12時 max{f[10]+1,f[9]+3,f[8]+5,f[5]+9} →f[12]=15

i=13時 max{f[11]+1,f[10]+3,f[9]+5,f[6]+9} →f[13]=15

【**參考程式**】（順推法）

program allbeibao;

const maxm=2000;maxn=30;

type ar=array[1..maxn] of integer;

var m,n,j,i,t:integer;

c,w:ar;

f:array[1..maxm] of integer;

begin

readln(n,m);

for i:=1 to n do read(w[i]);readln;

for i:=1 to n do read(c[i]);

for i:=1 to n do f[i]:=0;

t:=0;

for i:=1 to m do

for j:=1 to n do

begin

if i>=w[j] then t:=f[i-w[j]]+c[j];

if t>f[i] then f[i]:=t

end;

writeln(f[m]);

補上輸出“取最大價值時所選取的物品編號及所選選該物品的件數”；

end.

【**例13-9**】 乘積最大（noip2000提高組）

【**問題描述**】

今年是國際數學聯盟確定的“2000——世界數學年”，又恰逢我國著名數學家華羅庚先生誕辰90周年。在華羅庚先生的家鄉江蘇金壇，組織了一場別開生面的數學智力競賽的活動，你的一個好朋友XZ也有幸得以參加。活動中，主持人給所有參加活動的選手出了這樣一道題目:

設有一個長度N的數字串，要求選手使用K個乘號將它分成K+1個部分，找出一種分法，使得這K+1個部分的乘積能夠為最大。同時，為了幫助選手能夠正確理解題意，主持人還舉了如下的一個例子：

有一個數字串: 312，當N=3，K=1時會有以下兩種分法：

1）3\*12=36

2）31\*2=62

這時，符合題目要求的結果是：31\*2=62

現在，請你幫助你的好朋友XZ設計一個程式，求得正確的答案。

輸入：

程式的輸入共有兩行:

第一行共有2個自然數N,K (6<=N<=40，1<=K<=6)

第二行是一個K度為N的數字串。

輸出:

結果顯示在螢幕上，相對于輸入，應輸出所求得的最大乘積（一個自然數）。

樣例:

輸入

4 2

1231

輸出

62

【**演算法分析**】

此題滿足動態規劃法的求解標準，我們把它按插入的乘號數來劃分階段，若插入K個乘號，可把問題看做是K個階段的決策問題。

設數位字串為a1a2…an

K=1 時，一個乘號可以插在a1a2…an中的n-1個位置，這樣就得到n-1個子串的乘積：

a1\*a2…an, a1a2\*a3…an, …, a1a2…a n-1\*an

此時的最大值= max{a1\*a2…an, a1a2\*a3…an, …, a1a2…a n-1\*an }

K=2時，二個乘號可以插在a1a2…an中n-1個位置的任兩個地方，這樣總共會產生



個乘積。把這些乘積分個類，便於觀察規律。

Case1:a1a2 …\*a n-1\*an , a1a2 …\*a n-2 a n-1\*an , a1\*a2 …a n-3 a n-2 a n-1\*an ,

因後一個乘號位置不變，要使這些乘積最大，就要找出在前n-1個數中插入一個乘號的最大值。設符號F[n-1,1]為在前n-1個數中插入一個乘號的最大值,則Case1的最大值為F[n-1,1]\*an

Case2:a1a2 …\*a n-2\*a n-1 an , a1a2 …\*a n-3 a n-2\* a n-1 an , a1\*a2 …a n-3 a n-2\* a n-1 an ,

因後一個乘號位置不變，要使這些乘積最大，就要找出在前n-2個數中插入一個乘號的最大值。設符號F[n-2,1]為在前n-2個數中插入一個乘號的最大值,則Case2的最大值為F[n-2,1]\*a n-1 an

同理，Case3的最大值為F[n-3,1]\* a n-2 a n-1 an

Case n-2的最大值為F[2,1]\*a3…an

下面以n=9, k=4, s=‘321044105’為例，說明計算過程。

n 1 2 3 4 5 6 7 8 9

K=0 3 32 321 3210 32104 321044 3210441 32104410 321044105

1 - 6 63 630 12840 128416 - - -

2 - - 6 60 2520 51360 526440 - -

3 - - - 0 240 10080 103320 1033200 -

4 - - - - 0 960 10080 100800 5166000

其實在上面顯示的計算過程中，我們未顯示出右子串是什麼，但它實際上是要參加運算的。我們引入符號f[i,k]表示從a1~ai中插入k個乘號取得的最大值，g[i,j]表示從ai~aj的子串的數值。則上式可表示為：

f[9,4] = max{f[8,3]\*g[9,9], f[7,3]\*g[8,9], f[6,3]\*g[7,9], f[5,3]\*g[6,9], f[4,3]\*g[5,9]}

F[8,3] = max{f[7,2]\*g[8,8], f(6,2]\*g[7,8], f[5,2]\*g[6,8], f[4,2]\*g[5,8],f[3,2]\*g[4,8]}

F[7,3] = max{f[6,2]\*g[7,7], f[5,2]\*g[6,7], f[4,2]\*g[5,7], f[3,2]\*g[4,7]}

……

F[7,2] = max{f[6,1]\*g[7,7], f[5,1]\*g[6,7], f[4,1]\*g[5,7], f[3,1]\*g[4,7], f[2,1]\*g[3,7]}

……

F[6,1] = max {f[5,0]\*g[6,6], f[4,0]\*g[5,6], f[3,0]\*g[4,6], f[2,0]\*g[3,6], f[1,0]\*g[2,6]}

設f[i,k]表示在前i位數中插入K個乘號所得的最大值，a[i,j]表示從第i位到第j位所組成的自然數。用f[i,k]存儲階段K的每一個狀態，可以得狀態轉移方程：

f[i,k]=max{f[j,k-1]\*g[j+1,i]}（k<=j<=i）

邊界值：f[j,0]=a[1,j] (1<j<=i)

根據狀態轉移方程，我們就很容易寫出動態規劃程式：

for k:=1 to r do

for i:=k+1 to n do

for j:=k to I do

if f[i,k]<f[j,k-1]\*g[j+1,I] then f[i,k]:=f[j,k-1]\*g[j+1,i]

參考程式（略）。

【**例13-10**】 最長公共子序列問題

假設有：X= x1 x2 x3 ... xm；Y = y1 y2 y3 ... yn

1、最長公共子序列(不必連續)

定義f(m, n)為X和Y之間最長的子序列的長度；於是有f(m, 0) = f(0, n) = 0。

①如果xm≠yn, 則f(m, n) = max{ f(m-1, n), f(m, n-1) }，也就是說，如果兩個字串最後一位元不等，則它們的最長公共子串Z的最後一位不等於X的最後一位，則Z是X的首碼與Y的最長公共子串或是X與Y的首碼的最長公共子串。

【**舉例說明**】

X = <A, B, E, …, G, M>

Y= <C, D, H, …, K, S>

Z= <……, F>

因X和Y最後一位不等，X和它的子串的最後一位也不等，則Z是X’=<A,B,E…,G>和Y的最長公共子串。

X = <A, B, E, …, G, W>

Y= <C, D, H, …, K, I>

Z= <……, F>

因X和Y最後一位不等，Y和它的子串的最後一位也不等，則Z是X和Y’=<C,D,H, …K>的最長公共子串。

②如果xm = yn，則f(m, n) = f(m-1, n-1) + 1，也就是說，如果兩個字串最後一位元字元相等，則它們的最長公共子串的最後一位元就是這個字元，並且公共子串的第k-1個首碼是兩個字串首碼的最長公共子串。

【**舉例說明**】

X = <A, B, E, …, G, F>

Y= <C, D, H, …, K, F>

因X和Y的最後一位相等，則它們的最長公共子串Z的最後一位必為F，即有：

Z = <……, F>的樣式。並且 Z’=<……>是X’=<A, B, E, …,G> 和Y’=<C,D,H, …,K>的最長公共子串。

【**求解示意圖**】

設X＝＜A，B，C，B＞，Y＝＜B，D，C＞。

1) X4 = B, Y3=C , X4 <> Y3，因此轉而求“ X’=<A,B,C>與Y的LCS” 和 “Y’= <B,D>與X的LCS”。誰長誰就是X和Y的LCS。

2a) X=<A,B,C>, Y=<B,D,C>, X3=Y3, 因此轉而求X=<A,B>與Y=<B,D>的LCS。

2b) X=<A,B,C,B>, Y=<B,D>, X4<>Y2, 因此轉而求X=<A,B,C>與Y=<B,D> 和 X=<A,B,C,B>與Y=<B>的LCS。

3a) X=<A,B>, Y= <B,D>, X2<>Y2, 因此轉而求<A>與<B,D> 和<A,B>與<B>的LCS。

3b1)X=<A,B,C>, Y=<B,D>, X3<>Y2, 因此轉而求<A,B,C>與<B>的LCS和<A,B>與<B,D>的LCS。

3b2) X=<A,B,C,B>, Y=<B>, X4=Y1, 轉而求 <A,B,C>與<>的LCS。在下一次計算中，就會遇到<A,B,C>與<>。顯然，一個字串與一個空串的公共子串仍是一個空串，長度為0。因此X=<A,B,C,B>, Y=<B>的LCS為<B>，長度為1。

4a)同理， <A>與<B,D> 的LCS為<>; <A,B>與<B>的LCS為<B>, 因此<A,B>與<B,D>的LCS為<B>。

4b)因<A,B,C>與<B>的LCS為<B>; <A,B>與<B,D>的LCS為<B> 故<A,B,C,B>與<B,D>的LCS為<B>。

5a)因此<A,B>與<B,D>的LCS為<B> 故<A,B,C>與<B,D,C>的LCS為<B,C>。

6) 因<A,B,C>與<B,D,C>的LCS為<B,C>； <A,B,C,B>與<B,D>的LCS為<B> 故

<A,B,C,B>與<B,D,C>的LCS為<B,C>。

【**檔輸入**】

文件：lcs.in。檔包含兩行，分別為兩個字串。

【**檔輸出**】

文件：lcs.out。檔包含兩行，第一行為公共子序列的長度，第二行為公共子序列的字元組成。

【**參考程式**】

program lcs;

const maxlen=200;

var i,j:longint;

f:array[0..maxlen,0..maxlen] of byte;

x,y,z:string;

begin

assign(input,'lcs.in');

reset(input); readln(x); readln(y);

close(input);

fillchar(f,sizeof(f),0);

for i:=1 to length(x) do

for j:=1 to length(y) do

if x[i]=y[j] then f[i,j]:=f[i-1,j-1]+1

else if f[i-1,j]>f[i,j-1] then f[i,j]:=f[i-1,j]

else f[i,j]:=f[i,j-1];

z:='';

i:=length(x);j:=length(y);

assign(output,'lcs.out');

rewrite(output);

writeln(f[i,j]);

while (i>0) and (j>0) do

if x[i]=y[j]

then begin z:=x[i]+z;i:=i-1;j:=j-1 end

else if f[i-1,j]>f[i,j-1] then i:=i-1

else j:=j-1;

if z<>'' then writeln(z);

close(output);

end.

2、最長公共子序列(必須是連續的)

定義f(m, n)為X和Y之間最長的公共子序列的長度並且該子字串結束於Xm & Yn。因為要求是連續的，所以定義f的時候多了一個要求字串結束於Xm & Yn。

於是有f(m, 0) = f(0, n) = 0

①、如果xm≠yn, 則f(m, n) = 0

②、如果xm = yn, 則f(m, n) = f(m-1, n-1) + 1

因為最長公共子序列不一定結束於X / Y末尾，所以這裡必須求得所有可能的f(p, q) | 0 < p < m, 0 < q < n, 最大的f(p, q)就是解。

【**應用動態規劃要注意**】

1．階段的劃分是關鍵，必須依據題意分析，尋求合理的劃分階段(子問題)方法。而每個子問題是一個比原問題簡單得多的優化問題。而且每個子問題的求解中，均利用它的一個後部子問題的最優化結果，直到最後一個子問題所得最優解，它就是原問題的最優解。

2．變數太多，同樣會使問題無法求解。

3．最優化原理應在子問題求解中體現。有些問題也允許順推。

【**練習**】

1、防衛導彈

一種新型的防衛導彈可截擊多個攻擊導彈。它可以向前飛行，也可以用很快的速度向下飛行，可以毫無損傷地截擊進攻導彈，但不可以向後或向上飛行。但有一個缺點，儘管它發射時可以達到任意高度，但它只能截擊比它上次截擊導彈時所處高度低或者高度相同的導彈。現對這種新型 防衛導彈進行測試，在每一次測試中，發射一系列的測試導彈（這些導彈發射的間隔時間固定，飛行速度相同），該防衛導彈所能獲得的資訊包括各進攻導彈的高度，以及它們發射次序。現要求編一程式，求在每次測試中，該防衛導彈最多能截擊的進攻導彈數量，一個導彈能被截擊應滿足下列兩個條件之一：

①、是該次測試中第一個被防衛導彈截擊的導彈；

②、是在上一次被截擊導彈的發射後發射，且高度不大於上一次被截擊導彈的高度的導彈。

2、石子合併

在一個圓形操場的四周擺放著N堆石子(N<= 100),現要將石子有次序地合併成一堆.規定每次只能選取相鄰的兩堆合併成新的一堆,並將新的一堆的石子數,記為該次合併的得分.編一程式,由檔讀入堆疊數N及每堆疊的石子數(<=20).

①選擇一種合併石子的方案,使用權得做N－1次合併,得分的總和最小;

②選擇一種合併石子的方案,使用權得做N－1次合併,得分的總和最小;

輸入資料:

第一行為石子堆數N;

第二行為每堆的石子數,每兩個數之間用一個空格分隔.

輸出資料:

從第一至第N行為得分最小的合併方案.第N+1行是空行.從第N+2行到第2N+1行是得分最大合併方案.每種合併方案用N行表示,其中第i行(1<=i<=N)表示第i次合併前各堆的石子數(依順時針次序輸出,哪一堆先輸出均可).要求將待合併的兩堆石子數以相應的負數表示.

輸入輸出範例:

輸入:

4

4 5 9 4

輸出:

－４ ５ ９ －４

－８ －５ ９

－１３ －９

２２

４ －５ －９ ４

４ －１４ －４

－４ －１８

２２

3、複製書稿

【問題描述】

影印機發明之前，複製一本書是非常困難的，書中的所有內容只能請抄寫員用手工重新抄寫．一個抄寫員要完成複製一本書的工作．往往需要幾個月的時間。歷史記載曾有一個著名的抄寫員名叫Xerox。無論如何，複製書稿這種工作是非常令人討厭的，而提高複製速度的唯一方法只能是雇傭更多的抄寫員。

很久以前，有一個戲院的藝術團準備演出一場著名的古代悲劇，這個古代悲劇的原著被分成了許多本書。顯然，表演者需要這些書稿的副本，所以，他們雇傭了許多抄寫員來複製這些書稿。現在請你主持複製工作，假設你有m本書(編號為l，2，…，m)，想將這些書每本複製一份，m本書的頁數有可能不同(頁數分別是P1，P2，…，Pm )。你的任務是將這m本書分給k個抄寫員(k<=m)，每本書只能分配給一個抄寫員進行複製，而每個抄寫員所分配到的書必須是連續版序的。意思是說。存在一個連續昇冪數0=b0<b1<b2<…<bk-1<bk=m，這樣，第i號抄寫員得到的書稿是從第bi-1+1本書到第bi本書．複製工作是同時開始進行的。並且每個抄寫員複製的速度都是一樣的，所以，複製完所有書稿所需的時問決定於分配得列最多工作的那個抄寫員的複製時間。因此，你的任務是找到一個最優分配方案，儘量平均分配工作量，使得分配給每一個抄寫員的頁數的最大值盡可能的小(如果存在多個最優分配方案，只用輸出其中一種)。

【輸入】

檔的笫一行是兩個整數m和k(1<=k<=m<=500)。

第二行有m個整數P1，P2，…，Pm，這m個整數均為正整數且都不超過1000000．每兩個整數之間用空格分開。

【輸出】

檔應該有k行，每行有兩個正整數．整數之問用空格分開．

第i行的兩個整數ai和bi，表示第i號抄寫員所分配得到的書稿的起始編號和終止編號，即從第ai本書到第bi本書是第i號抄寫員複製．

【輸入輸出樣例1】

|  |  |
| --- | --- |
| books.in | books.out |
| 9 3  100 200 300 400 500 600 700 800 900 | 1 5  6 7  8 9 |

【輸入輸出樣例2】

|  |  |
| --- | --- |
| books.in | books.out |
| 5 4  100 100 100 100 100 | 1 1  2 2  3 3  4 5 |

**排列與組合**

加法原理與乘法原理

1、加法原理：

完成一件事有n類辦法，在第一類辦法中有m1 種不同的方法，在第二類辦法中有 m2種不同的方法，……，在第n類辦法中有 mn種不同的方法。那麼完成這件事共有 N= m1+m2+...+mn 種不同的方法。

2、乘法原理：

完成一件事需要分成n個步驟，做第一步有m1 種不同的方法，做第二步有 m2種不同的方法，……，做第n步有 種mn不同的方法，那麼完成這件事有 N=m1\*m2\*...\*mn 種不同的方法。

3.兩個原理的聯繫與區別：

分類計數原理與分步計數原理的共同點是它們完成一件事情，共有多少種不同的方法，區別在於完成一件事情的方式不同：分類計數原理是“分類完成”，即任何一類辦法中用任何一個方法都能獨立完成這件事；分步計算原理是“分步完成”，即這些方法需要分步驟順次相依，且每一個步驟都完成了，才能完成這件事情。

排列與組合的概念與計算公式

1、排列及計算公式

從n個不同元素中，任取m(m≤n)個元素按照一定的順序排成一列，叫做從n個不同元素中取出m個元素的一個排列；從n個不同元素中取出m(m≤n)個元素的所有排列的個數，叫做從n個不同元素中取出m個元素的排列數，用符號p表示。

p=n(n-1)(n-2)……(n-m+1)= n!/(n-m)!(規定0!=1)。

2、組合及計算公式

從n個不同元素中，任取m(m≤n)個元素並成一組，叫做從n個不同元素中取出m個元素的一個組合；從n個不同元素中取出m(m≤n)個元素的所有組合的個數，叫做從n個不同元素中取出m個元素的組合數。用符號C表示。

C= p/m!=n!/((n-m)!\*m!)；

組合的性質：

（1）C= C；

（2）C= C+C；

３、其他排列與組合公式

從n個不同的元素中取出r個，按照一定的順序排在一個封閉曲線上，叫做環形排列（或迴圈排列，圓排列），其排列數＝p/r=n!/r(n-r)!。

若n個元素中，有n1個a1，n2個a2，…，nk個ak，且n1+ n2+…+nk=n，則這n個不盡相異元素的全排列數為 n!/(n1!\*n2!\*...\*nk!)。

從m個不同的元素中，允許重複取出k個元素，不管怎樣的順序並成一組，稱為m個相異元素允許重複的k元組合，其計算公式為C。

解排列組合應用問題的基本思路：

（1）以元素為主考慮，即先滿足特殊元素的要求，再考慮其他元素；

（2）以位置為主考慮，即先滿足特殊位置的要求，再考慮其他位置；

（3）先不考慮附加條件，計算出排列或組合數，再減去不合要求的排列或組合數。

“分步相乘，分類相加，有序排列，無序組合”，合理地分類、正確地分步是應用分類計數原理和分步計數原理的關鍵，分清是否與順序有關是區別排列與組合的依據。

例：某天的課程表要排入語文、數學、英語、物理、化學、體育6六課程，如果第一節不排體育，最後一節不排數學，一共有多少種不同的排法？

用排除法：

不考慮任何限制條件有p種，其中包括不符合條件的有：

①．數學排在最後一節，有p種；

②．體育排在第一節，有p種；

但這兩種情況中都包含著數學排在最後一節，體育排在第一節的情況，有p種，故以上共有p－2 p+ p=21 p種。

例：一個口袋有4個不同的紅球，6個不同的白球，求：

（1）從中任取4個球，紅球的個數不比白球少的取法有多少種？

（2）若取1個紅球記2分，取1個白球記1分，從中任取5個球，使總分不少於7分的取法有多少種？

解：

（1）可以分以下三種情況：

①．取4個紅球不取白球，有C種；

②．取3個紅球1個白球，有CC種；

③．取2個紅球2個白球，有CC種。

所以共有C+ CC+ CC=115種。

（2）設紅球取x個，白球取y個，則有：

x+y=5

2x+y=7

由此可得三組解：x=2、y=3；x=3、y=2； x=4、y=1

於是可求出取法種數為：CC+ CC+ CC=186種

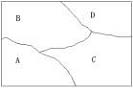
練習：

1、由數位1，2，3，4，5可以組成多少個三位數（各位上的數字允許重複）？

2、由數位0、1，2，3，4，5可以組成多少個三位數（各位上的數字允許重複）？

3、由數位0，1，2，3，4，5可以組成多少個十位元數位大於個位數位的兩位元數？

4、一個三位元密碼鎖,各位上數位由0,1,2,3,4,5,6,7,8,9十個數位組成,可以設置多少種三位元數的密碼(各位上的數位允許重複)？首位數位不為0的密碼數是多少種？首位數位是0的密碼數又是多少種？

5、如圖，要給地圖A、B、C、D四個區域分別塗上3種不同顏色中的某一種，允許同一種顏色使用多次，但相鄰區域必須塗不同的顏色，不同的塗色方案有多少種？

6、某班有22名女生，23名男生。

① 選一位學生代表班級去領獎，有幾種不同選法？

② 選出男學生與女學生各一名去參加智力競賽，有幾種不同的選法？

7、105有多少個約數？並將這些約數寫出來.。

8、從5幅不同的國畫、2幅不同的油畫、7幅不同的水彩畫中選不同畫種的兩幅畫佈置房間，有幾種選法？

9、若x、y可以取1，2，3，4，5中的任一個，則點(x,y)的不同個數有多少？

10、一個口袋內裝有5個小球另一個口袋內裝有4個小球，所有這些小球的顏色各不相同① 從兩個口袋內任取一個小球，有 種不同的取法；

11、從兩個口袋內各取一個小球，有 種不同的取法。

12、乘積（a1+a2+a3）(b1+b2+b3+b4)(c1+c2+c3+c4+c5)展開共有 個項。

13、有四位考生安排在5個考場參加考試.有 種不同的安排方法。

(答案:125；180；15；1000，900，100；6；45，506；8；59；25；9；20；60；625)

練習：

1、（1）用0，1，2，3，4組合多少無重複數位的四位元數？（96）

　 （2）這四位數中能被4整除的數有多少個？（30）

　 （3）這四位數中能被3整除的數有多少個？（36）

2、用0，1，2，3，4五個數位組成無重複數位的五位元數從小到大依次排列。

（1）第49個數是多少？（30124）

（2）23140是第幾個數？（40）

３、求下列不同的排法種數：

（1）6男２女排成一排，2女相鄰；(p(7,7)\*p(2,2))

（2）6男２女排成一排，2女不能相鄰；(p(6,6)\*p(7,2))

（3） 5男3女排成一排，3女都不能相鄰;(p(5.5)\*p(6,3))

（4） 4男4女排成一排，同性者相鄰；(p(4,4)\*p(4,4)\*p(2,2))

（5）4男4女排成一排，同性者不能相鄰。(p(4,4)\*p(4,4)\*p(2,2))

４、有四位醫生、六位護士、五所學校。

(1)  若要選派三位醫生到五所學校之中的三所學校舉辦健康教育講座，每所學校去一位醫生有多少種不同的選派方法？(c(5,3)\*p(4,3))

(2)  在醫生或護士中任選五人，派到五所學校進行健康情況調查，每校去且僅去一人，有多少種不同的選派方法？(p(10,5))

(3)  組成三個體檢小組，每組一名醫生、兩名護士，到五所學校中的三所學校為老師體檢，有多少種不同的選派方法？(c(5,3)\*p(4,3)\*c(6,2)\*c(4,2)\*c(2,2))

５、平面上有三條平行直線，每條直線上分別有7，5，6個點，且不同直線上三個點都不在同一條直線上。問用這些點為頂點，能組成多少個不同四邊形？(2250)

６、平面上有三條平行直線，每條直線上分別有7，5，6個點，且不同直線上三個點都不在同一條直線上。問用這些點為頂點，能組成多少個不同三角形？(751)

７、將N個紅球和M個黃球排成一行。例如:N=2,M=2可得到以下6種排法:

紅紅黃黃 紅黃紅黃 紅黃黃紅 黃紅紅黃 黃紅黃紅 黃黃紅紅

問題:當N=4,M=3時有多少種不同排法?(不用列出每種排法)(35)

8.用20個不同顏色的念珠穿成一條項鍊,能做多少個不同的項鍊.(20!/20)

9．在單詞MISSISSIPPI 中字母的排列數是(11!/(1!\*4!\*4!\*2!)

10．求取自1,2,...k的長為r的非減序列的個數為(c(r+k-1,r))

**排列的產生演算法**

用PascaI語言程式設計計算排列數，當元素數目n較小時，可以採用窮舉法計算排列總數，並輸出所有不同的排列。請讀者自行上機驗證。

但是，當元素數日n較大時，窮舉法便受到制約。這是因為利用窮舉法保存所有的全排列需要佔用較大的空間。例如n=20，要保存每個排列需要20!的空間，這是不現實的。所以需要尋求一種通用的解題方法。

**【鄰位交換法】**

為了求出元素1，2，3，…，n的所有不同的排列，特制定如下規則：

(1)寫出從小到大的初始排列，在每個數上方加上向左的箭頭，即

1 2 … n

若某元素在箭頭所指的方向上存在數值比它小的數，則認定該元素為活動元素，找出當前最大的活動元素d；

(2)令最大活動元素d沿箭頭方向前進一步，即交換一次資料，得到一個新排列；

(3)改變比最大活動元素d大的所有元素的箭頭方向；

(4)反復執行(2)、(3)步，直到數列中沒有活動元素為止。

**【演算法分析】**

設n=2：

(1)初始排列加上左箭頭得：1 2

(2)按照箭頭所指方向(向左)看，數位2的相鄰元素1比它小，稱2為活動元素。令活動元素向箭頭方向前進一步，即1與2交換一次，可得一個新的排列：

2 1

此時數列己不存在活動元素。因為2個不同元素的全排列種數為2，且至此已經得到2不同的排列，所以運算到此結束。

所有的不同排列為： 1 2 2 1

設n=3：

(1)初始排列：1 2 3

(2)根據上述規則可知當前數列的活動元素有2和3，最大的活動元素是3；

(3)先讓最大的活動元素3進行兩次交換，分別得到兩個排列：

1 3 2

3 1 2

這時最大的活動元素變成2，交換2與l後可得

3 2 1

此時雖沒有活動元素，但是只構造出3個排列。由於3 ！=6，說明上述操作並沒有構造出全部排列。

(4)繼續執行第3步，改變3的箭頭方向，得

3 2 1

(5)按照最大數3的箭頭所指方向（向右）看，相鄰元素2比它小，3再次作為活動元素，與2交換，得

2 3 1

再進行3與1的交換，得

2 1 3

此時不再有活動元素，即可獲得全部排列：

1 2 3 1 3 2 2 1 3 2 3 1 3 1 2 3 2 1

設n=4：

下面討論生成全排列的全部過程

|  |  |
| --- | --- |
| 狀態 | 操作注釋 |
| 1 2 3 4 | 初始排列，4是最大活動元素 |
| 1 2 4 3 | 4 與 3 交換 |
| 1 4 2 3 | 4 與 2 交換 |
| 4 1 2 3 | 4 與 1 交換，3 是最大活動元素 |
| 4 1 3 2 | 4 變更方向為 4 |
| 1 4 3 2 | 4與1交換 |
| 1 3 4 2 | 4與3交換 |
| l 3 2 4 | 4與2交換，3是最大活動元素 |
| 3 1 2 4 | 4變更方向為4 |
| 3 l 4 2 | 4與2交換 |
| 3 4 1 2 | 4與1交換 |
| 4 3 1 2 | 4與3交換，2是最大活動元素 |
| 4 3 2 l | 4 3變更方向為4 3 ；4是最大活動元素 |
| 3 4 2 1 | 4與3交換 |
| 3 2 4 1 | 4與2交換 |
| 3 2 1 4 | 4與1交換，3成為最大活動元素 |
| 2 3 1 4 | 4變更方向為4 |
| 2 3 4 1 | 4與1交換 |
| 2 4 3 1 | 4與3交換 |
| 4 2 3 1 | 4與2變換，3是最大活動元素 |
| 4 2 1 3 | 4變更方同為4，4是最大活動元素 |
| 2 4 1 3 | 4與2交換 |
| 2 1 4 3 | 4與l交換 |
| 2 1 3 4 | 4與3交換，已沒有活動元素，結束 |

下面為實現上述演算法編寫的Pascal程式，其中陣列p[1.. n]表示排列pl，p2，…，pn；d[1..n] 表示最大活動元素在排列中的位置變化；e[1..n]表示箭頭方向資料，用“-1”表示左箭頭；“1”表示在箭頭。

例如n=4的前5組輸出資料為：

|  |  |  |
| --- | --- | --- |
| p[1..4] | d[1..4] | e[1..4] |
| 1 2 3 4 | l 2 3 4 | -1-1-1-1 |
| 1 2 4 3 | l 2 3 3 | -1-1-1-1 |
| 1 4 2 3 | 1 2 3 2 | -1-1-1-1 |
| 4 1 2 3 | 1 2 3 1 | -1-1-1-1 |
| 4 1 3 2 | 1 2 2 O | -1-1-1 1 |

從陣列d[1..n]中d[4]的變化，可以看出最大活動元素4從p4(d[4]=4)的位置連續變換到p1(d[4]=1)位置，然後成為不活動元素(d[4]=0)。這時3成為最大活動元素，d[3]由3變為2表示向左交換一次，同時e[4]由-1變為1，表示元素4上面的箭頭由向左變為向右。

程式中並沒有設定變數d[1]和 e[1]，因為它們分別表示最小元素的活動狀態和箭頭，其值始終分別為1和-1。輸出時為了顯示完整的結果，直接添加了d[1]和e[1]的值。

從程式的運行結果可以看出變化規律是：d[4]變化4次後，d[3]才變化一次；同樣d[3]變化3次後，d[2]變化一次。

【參考程式】

program pailei;

const n=4;

label 8;

var

p:array[1..n] of integer;

e,d:array[2..n]of integer;

j,t,k,l,m:integer;

begin

for j:=1 to n do {排列初始化}

begin

p[j]:=j;

d[j]:=j;

e[j]:=-1;

end;

writeln;

writeln('p[1..n]':7,'d[1..n]':14,'e[1..n]':14);{輸出表頭}

8:m:=0;

for j:=1 to n do write(p[j],' '); {輸出當前的排列}

write(' ':6);

write('1 '); {添加d[1]的值}

for j:=2 to n do write(d[j],' '); {輸出當前的活動元素位置}

write(' ':6);

write('-1'); {添加e[1]的值}

for j:=2 to n do write(e[j],' '); {輸出當前的箭頭方向資料}

writeln;

for k:=n downto 2 do {資料交換過程}

begin

d[k]:=d[k]+e[k];

l:=d[k];

if l=k then e[k]:=-1 {箭頭方向向左}

else

if l=0 then

begin

e[k]:=1; {箭頭方向向右}

m:=m+1;

end

else

begin

l:=l+m;

t:=p[l]; {交換相鄰元素}

p[l]:=p[l+1];

p[l+1]:=t;

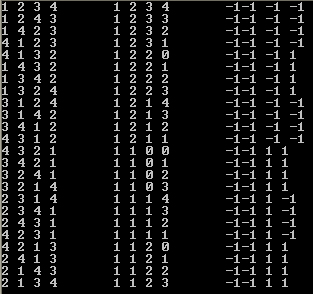
goto 8

end;

end;

end.

程式的運行結果如下：



**組合的產生演算法**

生成組合有哪些規律呢?下面通過例題討論如何程式設計計算組合的問題。

例：從1、2、3、4四個數中任取三個數，程式設計輸出所有的組合。

解：由組合的定義容易得出：從1，2，3，4中任取三個數的4種組合是

1 2 3 1 2 4 l 3 4 2 3 4

分析以上組合結果可以看出幾條規律：

若將已知的四個數由小到大排列成1 2 3 4，自左至右生成三個數的組合時，每一個組合中的三個數都按從小到大的順序排列，並且每一位數都按遞增變化。左側第一位數最大到2，左側第二位數最大到3，左側第三位數最大到4。可以歸納為：從1，2，…，n這n個數中任取m個數的組合，左側第i位元數最大到n-m+i(i=1,2,…,m)。

生成下一個組合時，首先考慮第i位數按遞增變化，當第i位數變化到最大值時，再考慮第i-l位數，第i-2位數，…，第1位數的變化。在上例中先由123變化到124，此時左側第三位數4已達到最大(n-m+i=4-3+3=4)，然後考慮左側第二位數2變成3，組合由124變化到134。

當三個數中右側連續2位元數都達到本位元素的最大值時，在上例中的134，左側第三位(即右側第一位)數4和左側第二位(即右側第二位)數3都已達到最大值，而左側第一位(即右側第三位)數1小於最大值2，則下一個組合的生成規律是

左側第一位數在原數字基礎上加1，即1+1=2；

左側第二位數在變化後的左側第一位數上加l，即2+1=3；

左側第三位數在變化後的左側第二位數上加l，即3+1=4；

所以，134的下一個組合是234。

這條規律可歸納為：若i是滿足條件Ci<n-m+i的最大位置（即當組合的m個數中右側連續m-i位元數都達到本位元素的最大值)時，下一個組合中

Ci：=Ci+1：

Ci+l：=Ci+l；

Ci+2：=Ci+1+l；

…

Cm：=Cm-l+1；

從這個具體的例子擴展到一般，求從n個不同元素中取出m個元素的所有組合可以按以下規則生成：

(1)設生成的組合為C1C2…Cm(C1＜C2＜…＜Cm)，則1≤Cj≤n-m+j (j=1，2，…，m)

如上例中：n=4，m=3，有C1≤4-3+1=2，C2≤4-3+2=3，C3≤4-3+3=4。

(2)令i=max{j|Cj<n-m+j}，賦值計算：

Ci：=Ci+1：

Ci+l：=Ci+l；

Ci+2：=Ci+1+l；

…

Cm：=Cm-l+1；

輸出C1C2…Cm，即為生成的一個組合。

(3)當C1C2…Cm都己滿足Ci=n-m+i(i=l，2，…，m)，則生成過程結束。

下面給出計算組合數及輸出所有組合的參考程式。

在程式中，a陣列存放生成的組合，b陣列存放每個Ci的取值上界(n-m+i)，k是陣列下標指標。

program comb;

var

n,m,i,k,s:integer;

a,b:array[0..100] of integer;

begin

write('input n,m=');

read(n,m);

for i:=1 to m do

begin

a[i]:=i; {前m個元素存入a陣列 }

b[i]:=n-m+i; {記錄a陣列中各元素的取值上界}

end;

k:=m;

s:=0;

repeat

if k=m then

begin

s:=s+1; {計算組合數}

for i:=1 to m do write(a[i]);

write(' ');

end;

if a[k]<b[k] then {判斷Ci是否達到上界}

begin

a[k]:=a[k]+1;

if k<m then

for k:=k+1 to m do a[k]:=a[k-1]+1;

end

else k:=k-1; {當Ci達到上界後，處理Ci-1，Ci-2，…}

until k=0;

writeln;

writeln('s=',s); {輸出組合數S=}

end.

排列與組合的產生演算法

1、排列的產生

方法１：（遞迴，深度優先產生）

程式如下：

program pailei;

const m=4;

var a:array[1..m] of integer ;

    b:array[1..m] of boolean;

procedure print;

var i:integer;

begin

  for i:=1 to m do

  write(a[i]);

  writeln;

end;

procedure try(dep:integer);

var i:integer;

begin

  for i:=1 to m do

  if  b[i] then

   begin

   a[dep]:=i; b[i]:=false;

   if dep=m then print else try(dep+1);

   b[i]:=true;

   end;

end;

begin

fillchar(b,sizeof(b),true);

try(1);

end.

方法２．根據上一個排列產生下一個排列．

程式如下：

program pailei;

const m=5;

var a:array[1..m] of integer ;

i,j,temp,k,l:integer;

procedure print;

var i:integer;

begin

  for i:=1 to m do

  write(a[i]);

  writeln;

end;

begin

for i:=1 to m do a[i]:=i;

repeat

  print;

  i:=m-1;

  while (i>0) and (a[i]>a[i+1]) do i:=i-1;

  if i>0 then

  begin

  j:=m;

  while  a[j]<a[i] do j:=j-1;

  temp:=a[i];a[i]:=a[j];a[j]:=temp;

  k:=i+1;l:=m;

  while k<l do

    begin

    temp:=a[k];a[k]:=a[l];a[l]:=temp;

     k:=k+1;l:=l-1

    end;

  end;

until i=0;

end.

２．組合的產生演算法

演算法１：（遞迴，深度優先產生）

程式如下：

program zuhe;

const n=6;m=4;

var a:array[0..m] of integer;

  i,j:integer;

procedure print;

var i:integer;

begin

  for i:=1 to m do write(a[i]);

  writeln;

end;

procedure try(dep:integer);

var i:integer;

begin

  for i:=a[dep-1]+1  to n-(m-dep) do

  begin

  a[dep]:=i;

  if dep=m then print else try(dep+1);

  end

end;

begin

a[0]:=0;

try(1);

end.

演算法２：根據前一個組合產生下一個組合

程式如下：

program zuhe;

const n=6;m=4;

var a:array[1..m] of integer;

  i,j:integer;

procedure print;

var i:integer;

begin

  for i:=1 to m do write(a[i]);

  writeln;

end;

begin

  for i:=1 to m do

  a[i]:=i;

  repeat

  print;

  i:=m;

  while (i>0) and (a[i]=n-(m-i)) do dec(i);

  if i>0 then

   begin

    a[i]:=a[i]+1;

    for j:=i+1 to m do a[j]:=a[j-1]+1;

  end;

  until i=0;

end.

練習：

１、已知n（1<=n<=20）個整數x1,x2,…,xn（1<=xi<=5000000），以及一個整數k（k<n）。從n個整數中任選k個整數相加，可分別得到一系列的和。現在，要求你計算出和為素數共有多少種。

２、n個部件,每個部件必須經過先A後B兩道工序。已知部件i在A,B 機器上的時間分別為ai，bi。如何安排加工順序，總加工時間最短？

輸入：

5

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 部件 | 1 | 2 | 3 | 4 | 5 |
| ai | 3 | 5 | 8 | 7 | 10 |
| bi | 6 | 2 | 1 | 4 | 9 |

輸出：

34

1 5  4  2  3

計算幾何

4.1 基礎知識

1.兩點間的距離公式：

已知：平面上的兩點的直角坐標分別為P1(x1,y1)，P2(x2,y2)，則P1和P2兩點間的距離為

      d=sqrt((x1-x2)\*(x1-x2)+(y1-y2)\*(y1-y2))

2.線段的中點座標公式：

已知：平面上的兩點的直角坐標分別為P1(x1,y1)，P2(x2,y2)，則線段P1P2的中點座標為：

      x=(x1+x2)/2     y=(y1+y2)/2

3.直線的斜率公式：

已知：平面上的兩點的直角坐標分別為P1(x1,y1)，P2(x2,y2),則線段P1P2所在的直線的斜率為：k=（y2-y1)/(x2-x1)

4.直線的點斜式方程:

已知：直線過點P0(x0,y0)，斜率為k,則該直線所在的方程為：

       y=k(x-x0)+y0=kx+y0-kx0=kx+b(與y軸交點的縱坐標:縱截距)

5、點到直線的距離：

已知點P0(x0,y0)，直線Ax+By+C=0，則點P0到直線的距離為：D=A\*x0+B\*y0+C/sqrt(A\*A+B\*B)

練習：

1.已知:矩形上三點的座標p1(x1,y1),p2(x2,y2),p3(x3,y3)

(1)求矩形另外一點的座標p4(x4,y4)。

(2)判斷點p(x,y)是在矩形內、矩形外還是在矩形的邊上。

4. 2 線段的相交判斷

1.叉積

已知：平面上的兩點的直角坐標分別為p1(x1,y1)，p2(x2,y2)則

（1）該兩點相對座標原點(0,0)的叉積為m=x1\*y2-x2\*y1

    若m>0 則相對座標原點,點p1在點p2的順時針方向

    若m<0 則相對座標原點,點p1在點p2的逆時針方向

    若m=0 則原點和p1、p2在一條直線上

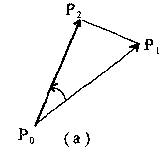
（2）該兩點相對點p0(x0,y0)的叉積為m=(x1-x0)\*(y2-y0)-(x2-x0)\*(y1-y0)

    若m>0 則相對p0點,點p1在點p2的順時針方向

    若m<0 則相對p0點,點p1在點p2的逆時針方向

    若m=0 則p0和p1、p2在一條直線上

2.確定兩條連續的有向線段p0p1和p1p2在pl點是向左轉還是向右轉



  (1)計算叉積m=(x1-x0)\*(y2-y0)-(x2-x0)\*(y1-y0)

 （2）判斷m

     若m>0 則p1點向左拐

     若m<0 則p1點向右拐

     若m=0 則點p0、p1、p2在一條直線上

 3.確定兩條線段p1p2、p3p4是否相交

程式如下：

program xdxj;

type p=record

     x, y:real

end;

var p1,p2,p3,p4:p;

function m(p1,p2,p0:p):real;

begin

  m:=(p1.x-p0.x)\*(p2.y-p0.y)-(p2.x-p0.x)\*(p1.y-p0.y);

end;

function max(a,b:real):real;

begin

  if a>b then max:=a else max:=b;

end;

function min(a,b:real):real;

begin

  if a<b then min:=a else min:=b;

end;

function across(p1,p2,p3,p4:p):boolean;

begin

if (max(p1.x,p2.x)>=min(p3.x,p4.x)) and (max(p3.x,p4.x)>=min(p1.x,p2.x)) and (max(p1.y,p2.y)>=min(p3.y,p4.y)) and (max(p3.y, p4.y)>=min(p1.y,p2.y)) and (m(p2,p3,p1)\*m(p2,p4,p1)<0) and (m(p4,p1,p3)\*m(p4,p2,p3)<0) then across:=true

else across:=false;

end;

begin

 readln(p1.x,p1.y,p2.x,p2.y);

 readln(p3.x,p3.y,p4.x,p4.y);

 if across(p1,p2,p3,p4) then writeln('yes') else writeln('no');

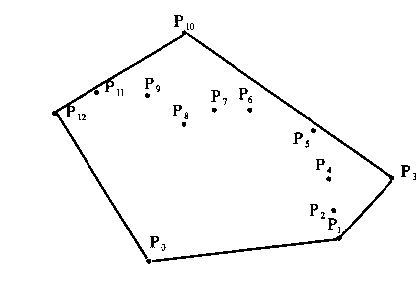
end.

4.３尋找凸包演算法

1.凸包的概念

一個點集Q＝（p0,p1,p2...pn-1)，它的凸包是一個最小的凸多邊形P，且滿足Q中的每個點或者在P的邊界上，或者在P的內部。在直觀上，我們可以把Q中的每個點看作露在板外的鐵釘。那麼凸包就是包含所有鐵釘的一個拉緊的橡皮繩所構成的形狀。

如圖：



2.尋找凸包演算法

演算法如下(Graham演算法):

1)求q中y座標最小的點p0,若具有最小座標的點有多個,則取最左邊的點作為p0。

2)對q中剩餘的點按逆時針相對p0的極角排序,若有數個保留其中距p0最遠的點得到序列(p1,p2,...pn-1)；

3)p0,p1,p2相繼入棧；

4)for i=3 to n-1 do

    1) while 由次棧頂元素、棧頂元素和Pi所形成角不是向左轉do棧頂元素出棧s；

    2)pi入棧；

5)列印按逆時針排列的棧中的各頂點。

程式如下:

program tubao;

const maxn=500;

type p=record

     x,y:real;

     end;

var n,top:integer;

list:array[0..maxn]of p;

s:array[1..maxn] of integer;

f:text;

procedure swap(var a,b:p);

var t:p;

begin t:=a;a:=b;b:=t end;

function m(p1,p2,p0:p):real;

begin

 m:=(p1.x-p0.x)\*(p2.y-p0.y)-(p2.x-p0.x)\*(p1.y-p0.y);

end;

function comp(p1,p2:p):boolean;

var t:real;

begin

 t:=m(p1,p2,list[0]);

 if (t>0) or (t=0) and (sqr(p1.x-list[0].x)+sqr(p1.y-list[0].y)<

  sqr(p2.x-list[0].x)+sqr(p2.y-list[0].y))

   then comp:=true else  comp:=false;

end;

procedure sort(l,r:integer);

var i,j:integer;

x:p;

begin

 if r-l+1<=5 then begin

  for j:=l+1 to  r do

   begin

    i:=j;

    while (i>1) and comp(list[i],list[i-1]) do

     begin  swap(list[i],list[i-1]); dec(i) end

   end;

  end else

   begin

    x:=list[l+random(r-l+1)];

    i:=l;j:=r;

    repeat

     while comp(list[i],x) do inc(j);

     while comp(x,list[j]) do dec(j);

     if i<j then swap(list[i],list[j])

    until i>=j;

    sort(l,j);

    sort(j+1,r);

    end

end;

procedure init;

var i:integer;

begin

 assign(f,'input.txt');

 reset(f);

 readln(f,n);

 for i:=0 to n-1 do

  begin

   readln(f,list[i].x,list[i].y);

   if (list[i].y<list[0].y) or

    (list[i].y=list[0].y) and (list[i].x<list[0].x)

    then swap(list[0],list[i]);

   end ;

 sort(1,n-1)

 end;

 procedure graham;

 var i:integer;

 begin

  for i:=1 to 3 do s[i]:=i-1;

  top:=3;

  for i:=3 to n-1 do

   begin

    while m(list[i],list[s[top]],list[s[top-1]])>=0 do dec(top);

    inc(top);

    s[top]:=i;

   end;

  for i:=1 to top do

   write('(',list[s[i]].x:7:2,',',list[s[i]].y:7:2,')');

  writeln

 end;

 begin

 init;

 graham;

 readln

 end.

練習:

1.巳知:平面上有n個點(n<=10000)，用Graham演算法找出彼此間最遠的兩個點.

其它與程式設計相關的數學知識

5.1 鴿巢原理

1.簡單形式

如果n+1個物體被放進n個盒子，那麼至少有一個盒子包含兩個或更多的物體。

例1：在13個人中存在兩個人，他們的生日在同一月份裡。

例2：設有n對已婚夫婦。為保證有一對夫婦被選出，至少要從這2n個人中選出多少人？（n+1）

2.加強形式

令q1,q2,...qn為正整數。如果將q1+q2+...+qn-n+1個物體放入n個盒子內,那麼或者第一個盒子至少含有q1個物體,或者第二個盒子至少含有q2個物體,...,或者第n個盒子含有qn個物體。

例3:一籃子水果裝有蘋果、香蕉和橘子。為了保證籃子內或者至少8個蘋果或者至少6個香蕉或者至少9個橘子，則放入籃子中的水果的最小件數是多少？（21件）

5. 2   容斥原理及應用

 原理：集S的不具有性質P1,P2,...，Pm的物體的個數由下式給出：

|A1∩A2∩...∩Am|=|S|-∑|Ai|+∑|Ai∩Aj|-∑|Ai∩Aj∩Ak|+...+(-1)m|A1∩A2∩...∩Am|

如:m=3,時上式為：

|A1∩A2∩A3|=|S|-(|A1|+|A2|+|A3|)+(|A1∩A2|+|A1∩A3|+|A2∩A3|)－|A1∩A2∩A3|

推論:至少具有性質P1,P2,...Pm之一的集合S的物體的個數有：

| A1∪A2∪....∪Am|=|S|—|A1∩A2∩...∩Am|=

∑Ai|-∑|Ai∩Aj|+∑|Ai∩Aj∩Ak|+...+(-1)m+1|A1∩A2∩...∩Am|

例4：求從1到1000不能被5，6，和8整除的整數的個數？

 （1000-（200+166+125）+（33+25+41）-8=600）

5.３ 常見遞推關係及應用

1.算術序列

每一項比前一項大一個常數d；若初始項為h0：則遞推關係為 hn=hn-1+d=h0+nd；對應的各項為：h0，h0+d，h0+2d，....,h0+nd；前n項的和為(n+1)h0+dn(n+1)/2 。

例5: 1,2,3,...

例6: 1,3,5,7...等都是算術序列。

2.幾何序列

每一項是前面一項的常數q倍；若初始項為h0：則遞推關係為 hn=h0qn-1q=h0qn；對應的各項為: h0，h0q1，h0q2，....,h0qn

例7: 1,2,4,8,16,...

例8: 5,15,45,135,...等都是幾何序列；前n項和為((qn+1-1)/(q-1) )h0

3.Fibonacci序列

除第一、第二項外每一項是它前兩項的和；

若首項為f0為0,則序列為0，1，1，2，3，5，8...遞推關係為(n>=2)fn=fn-1+fn-2，前n項的和Sn=f0+f1+f2+...+fn=fn+2-1

例9:以下是Fibonacci的示例:

1.樓梯有n階臺階,上樓可以一步上1階,也可以一步上2階,編一程式計算共有多少種不同的走法?

2.有一對雌雄兔,每兩個月就繁殖雌雄各一對兔子.問n個月後共有多少對兔子?

3.有n\*2的一個長方形方格,用一個1\*2的骨牌鋪滿方格。求鋪法總數？

4.錯位排列

首先看例題：

例10：在書架上放有編號為1,2,....n的n本書。現將n本書全部取下然後再放回去,當放回去時要求每本書都不能放在原來的位置上。

例如:n=3時:原來位置為:123

放回去時只能為:312或231這兩種

問題:求當n=5時滿足以上條件的放法共有多少種?(不用列出每種放法) (44)

{1，2，3，....,n}錯位排列是{1,2,3,..,n}的一個排列i1i2...in,使得i1<>1,i2<>2,i3<>3,...in<>n

錯位排列數列為 0,1,2,9,44,265,....

錯位排列的遞推公式是:dn=(n-1)(dn-2+dn-1)(n>=3)=ndn-1+(-1)n-2

5.分平面的最大區域數

1.直線分平面的最大區域數的序列為：2，4，7，11，....,

    遞推公式是: fn=fn-1+n=n(n+1)/2+1

2.折線分平面的最大區域數的序列為:2, 7, 16,29, ...,

    遞推公式是:fn=(n-1)(2n-1)+2n;

3.封閉曲線(如一般位置上的圓)分平面的最大區域數的序列為:2, 4, 8, 14,...,

    遞推公式是:fn=fn-1+2(n-1)=n2-n+2

6.Catalan 數列

先看下麵兩個例題:

例11:將一個凸多邊形區域分成三角形區域的方法數?

令hn表示具有n+1條邊的凸多邊形區域分成三角形的方法數。同時令h1=1，則hn滿足遞推關係

hn=h1hn-1+h2hn-2+...+hn-1h1(n>=2)(想一想,為什麼?)

該遞推關係的解為hn=c(2n-2,n-1)/n (n=1,2,3,...)

其對應的序列為1,1,2,5,14,42,132,....從第二項開始分別是三邊形,四邊形,...的分法數

即k邊形分成三角形的方法數為hk=c(2k-4,k-2)/(k-1)(k>=3)

例12:n個+1和n個-1構成2n項 a1,a2,...,a2n

其部分和滿足a1+a2+...+ak>=0(k=1,2,3,..,2n)對與n該數列為

 Cn=c(2k,k)/(k+1)  (k>=0)    對應的序列為        1,1,2,5,14,42,132,...

 序列1,1,2,5,14,42,132,....叫Catalan數列。

例13：下列問題都是Catalan數列。

1.有2n個人排成一行進入劇場。入場費5元。其中只有n個人有一張5元鈔票，另外n人只有10元鈔票， 劇院無其它鈔票，問有多少中方法使得只要有10元的人買票，售票處就有5元的鈔票找零？

2.一位大城市的律師在她住所以北n個街區和以東n個街區處工作。每天她走2n個街區去上班。如果他從不穿越（但可以碰到）從家到辦公室的對角線，那麼有多少條可能的道路？

3.在圓上選擇2n個點,將這些點成對連接起來使得所得到的n條線段不相交的方法數?

4.n個結點可夠造多少個不同的二叉樹?

5.一個棧(無窮大)的進棧序列為1,2,3,..n,有多少個不同的出棧序列?