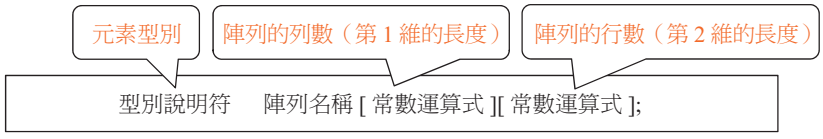
1.7.2 二維陣列

1．靜態定義

二維陣列的靜態定義格式如下圖所示。



其中，陣列的行數和列數必須是整數常數，不能是變數，該數值必須是已知的數值。

- 可以在定義時對陣列進行初始化。

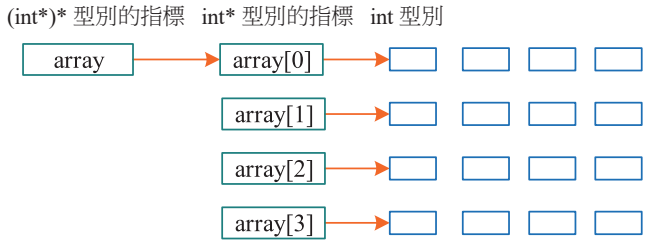
```
int a[2][4]={{0,1,2,3},{7,2,9,5}};
int a[2][4]={0,1,2,3,7,2,9,5};
int a[2][4]={{0,1,2},{0}};
```

- 將二維陣列作為參數時，可以省略其列數，但必須指定其行數。

```
int sum(int a[][5],int n);
```

2．動態定義

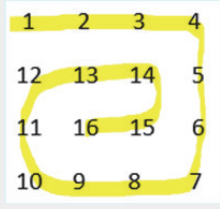
一個 *m* 列 *n* 行的二維陣列相當於 *m* 個長度為 *n* 的一維陣列。




```
int **array=new int*[m];
for(int i=0;i<m;++i){
    array[i]=new int[n];// 按列分配記憶體空間
}
for(int i=0;i<m;i++){
    delete[] array[i]; // 按列釋放記憶體空間
```

```
    }
    delete[] array;
```

訓練 18（P5731）：蛇形填數，輸入一個不大於 9 的正整數 n，以蛇形填寫 $n \times n$ 的矩陣。



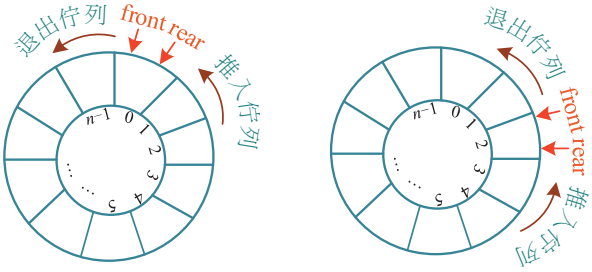

```
#include<bits/stdc++.h> // 萬能標頭檔
using namespace std;
int a[20][20];
int main(){
    int n,x,y,total=1;
    scanf("%d",&n);
    x=y=1;
    a[1][1]=1;
    while(total<n*n){
        while(y+1<=n&&!a[x][y+1])// 向右
            a[x][++y]=++total;
        while(x+1<=n&&!a[x+1][y])// 向下
            a[++x][y]=++total;
        while(y-1>0&&!a[x][y-1])// 向左
            a[x][--y]=++total;
        while(x-1>0&&!a[x-1][y])// 向上
            a[--x][y]=++total;
    }
    for(int i=1;i<=n;i++){
        for(int j=1;j<=n;j++)
            printf("%3d",a[i][j]);
        if(i<n) printf("\n");
    }
    return 0;
}
```

1.8 熟悉字串

字串指儲存在記憶體空間的連續位元組中的一系列字元。C++ 中的字串分為兩種形式：C 風格的字串、C++ string 型別的字串。

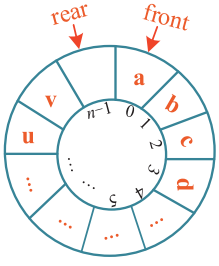
1．空佇列

無論佇列開頭和佇列結尾在什麼位置，只要 rear 和 front 指向同一位置，就視為空佇列。若將循環佇列中的一維陣列畫成環形圖，則空佇列的情況如下圖所示，front=rear。

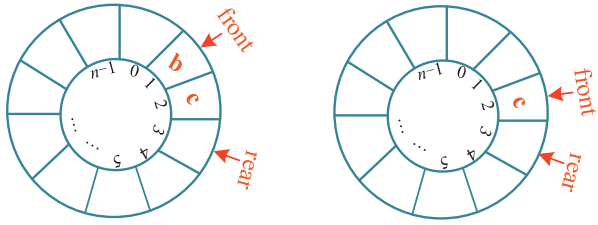


2．滿佇列

在此採用浪費一個記憶體空間的方法，當 rear 的下一個位置是 front 時，就視為滿佇列。但是 rear 向後移動一個位置（rear+1）後，很可能超出了陣列的最大索引值，這時它的下一個位置應該為 0，滿佇列（臨界狀態）的情況如下圖所示。其中，佇列的最大記憶體空間數 Maxsize=*n*，當 rear=Maxsize−1 時，rear+1=Maxsize。而根據循環佇列的規則，rear 的下一個位置為 0 才對，怎麼才能變為 0 呢？可以考慮取餘運算，即 (rear+1)%Maxsize=0，而此時 front=0，即 (rear+1)%Maxsize=front，為滿佇列的臨界狀態。



對滿佇列的一般狀態的判斷是否也適用此方法呢？例如，循環佇列滿佇列（一般狀態）的情況如下圖所示。假如最大記憶體空間數 Maxsize=100，當 rear=1 時，rear+1=2。進行取餘後，(rear+1)%Maxsize=2，而此時 front=2，即 (rear+1)%Maxsize=front。對滿佇列的一般狀態也可以採用此公式進行判斷，因為一個不大於 Maxsize 的數，與 Maxsize 進行取餘運算，其結果仍然是該數本身，所



對於退出佇列操作，當 front 後移一位時，為了處理臨界狀態（front+1=Maxsize），需要將 front 加 1 後進行取餘運算。

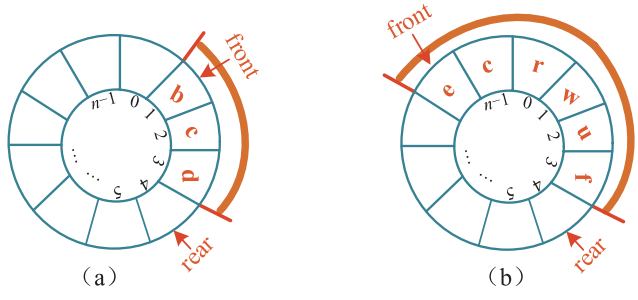
```
e=Q[front];    // 用變數記錄 front 指向的元素
front=(front+1)%Maxsize; //front 後移一位
```

注意

在循環佇列中無論是推入佇列還是退出佇列，在將 rear、front 加 1 後都要進行取餘運算，主要是為了處理臨界狀態。

5．佇列中的元素數量計算

在循環佇列中到底儲存了多少個元素呢？在循環佇列中儲存的實際上是從 front 到 rear−1 這一區間的資料元素，但是不可以直接用兩個索引值相減得到元素資料。因為佇列是循環的，所以存在兩種情況：rear≥front，如下圖（a）所示；rear<front，如下圖（b）所示。



在上圖（b）中，rear=4，front=Maxsize−2，rear−front=6−Maxsize。但是可以看到循環佇列中的元素實際上為 6 個，那怎麼辦呢？當兩者之差為負數時，可以將差值

演算法程式碼：

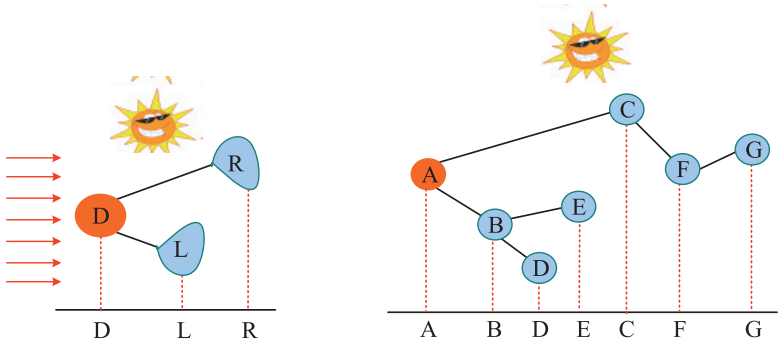
```
void posorder(Btree T) {//後序遍歷
    if(T){
       posorder(T->lchild);
       posorder(T->rchild);
       cout<<T->data<<"  ";
    }
}
```

二元樹遍歷程式碼簡單明瞭，「cout<<T->data;」述句在前面的就是前序遍歷，在中間的就是中序遍歷，在後面的就是後序遍歷。

若不要求遵循程式執行流程，只寫出二元樹遍歷序列，則還可以使用投影法快速得到該遍歷序列。

1．前序遍歷

前序遍歷就像在左邊刮大風的情況下遍歷，將二元樹的樹枝刮向右方，遍歷順序為根、左子樹、右子樹，太陽直射，將所有節點都投影到地上。一棵二元樹，其前序遍歷投影如下圖所示，前序遍歷序列為 ABDECFG。



2．中序遍歷

中序遍歷就像在無風的情況下遍歷，遍歷順序為左子樹、根、右子樹，太陽直射，將所有節點都投影到地上。一棵二元樹，其中序遍歷投影如下圖所示，中序遍歷序列為 DBEAFGC。

CHAPTER

09

動態規劃入門

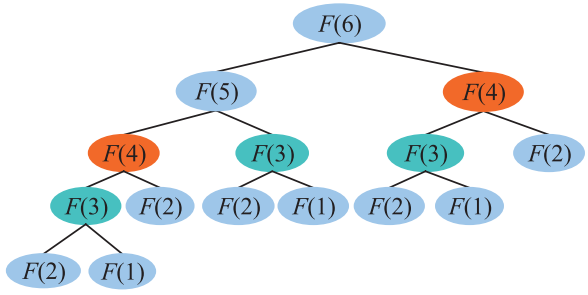
動態規劃是一種表格處理方法，它把原問題分解為若干子問題，由下而上先求最小子問題的解，把結果儲存在表格中，在求大的子問題的解時，直接從表格查詢小的子問題的解，避免重複計算，以提升效率。

9.1 動態規劃祕笈

對什麼樣的問題可以使用動態規劃求解呢？首先要分析問題是否具有以下 3 種性質。

（1）最佳子結構。最佳子結構指問題的最佳解包含其子問題的最佳解，這是使用動態規劃的基本條件。

（2）子問題重疊。子問題重疊指求解過程中每次產生的子問題並不總是新問題，有大量子問題是重複的。例如，在遞迴求解費氏數列時，有大量子問題被重複求解，如下圖所示。動態規劃利用了子問題重疊的性質，由下而上對每個子問題都只求解一次，將其結果儲存在一個表格中，當再次需要求解該子問題時，直接在表格中查詢，無須再次求解，進而提升效率。子問題重疊不是使用動態規劃解決問題的必要條件，但更能突出動態規劃的優勢。

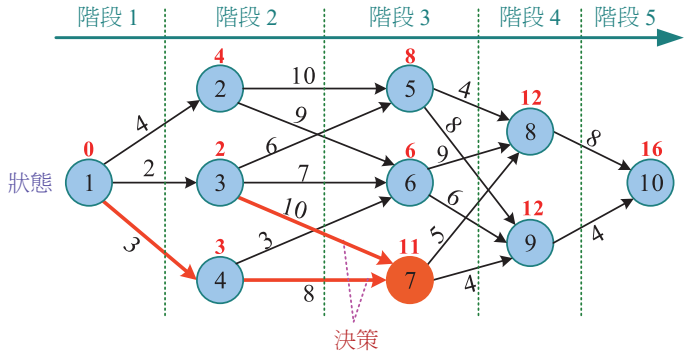


（3）無後效性。在動態規劃中會將原問題分解為若干子問題，將每個子問題的求解過程都作為一個階段，在完成前一階段後，根據前一階段的結果求解後一階段。並且，對當前階段的求解只與之前階段有關，與之後階段無關，這叫做「無後效性」。若一個問題有後效性，則需要將其轉換或逆向求解來消除後效性，之後才可以使用動態規劃。

9.1.1 動態規劃的三個要素

在現實生活中有一類活動，可以將活動過程按順序分解為若干個相互聯繫的階段，在每個階段都要做出決策，對全部過程的決策是一個決策序列。對每個階段決策的選擇都不是隨意確定的，它依賴當前狀態，又影響以後的發展。這種把問題看作一個前、後關聯的具有鏈狀結構的多階段的過程叫做「多階段決策過程」，這種問題就叫做「多階段決策問題」。

根據無後效性，動態規劃的求解過程構成一個有向非循環圖，求解遍歷的順序就是該有向非循環圖的一個拓撲序。在有向非循環圖中，節點對應問題的狀態，有向邊對應狀態之間的轉移，如何進行狀態轉移對應動態規劃中的決策。所以，**狀態、階段、決策**就是動態規劃的三個要素。



在求解動態規劃問題時，如何確定狀態和狀態轉移方程是關鍵，也是困難點。不同的狀態和狀態轉移方程可能產生不同的演算法複雜性。動態規劃問題靈活多變，在各類演算法競賽中層出不窮，需要多練習、多總結，累積豐富的經驗且發揮創造力。

9.2 背包問題

背包問題是動態規劃的經典問題之一，本節講解 01 背包問題、完全背包問題及其最佳化。背包問題指在一個有容積或重量限制的背包內裝入物品，物品有體積或重量、價值等屬性，要求在滿足背包容量或重量限制的情況下裝入物品，使背包內的物品價值之和最大。根據物品限制條件的不同，背包問題可分為 01 背包問題、完全背包問題、多重背包問題、分組背包問題和混合背包問題等。

9.2.1 01 背包問題

給定 n 種物品，每種物品都有重量 w_i 和價值 v_i，每種物品都只有一個。另外，背包容量為 W。求解在不超過背包容量的前提下將哪些物品裝入背包，才可以