蓝桥杯青少年创意编程 C++组
赛前集训教程包

版本-190919

蓝桥杯大赛组

目录

第 01 课 基本数据类型及运算符

1.1、基本数据类型及类型转换

1. 基本数据类型

整型：int、longlong

int 与 long long 的存储范围不同，一般情况下我们用 int 即可，但是如果题目中给的数据范围较大，则选择使用 long long。

布尔型：bool

布尔类型只有两个值，false 和 true。通常用来判断条件是否成立。

字符型：char

char 类型的变量代表的是单个字符，

例如：

char a;//定义一个字符型变量

cin>>a;//从键盘输入一个字符存入变量 a 中。

a= ‘*’ ；//给变量 a 赋一个字符’ *’ 。

注意：字符是用单引号来表示’ ’ 。

实型：float、double

float 与 double 的精确度不同 ,如果题目没有特别要求 ,我们一般使用 double，如果题目明确告诉你使用单精度浮点数数据类型或者 float，则要使用 float。

2. 数据类型的转换

（1）自动类型转换（隐式类型转换）

在不同数据类型的混合运算中，编译器会隐式地进行数据类型转换，称为自动类型转换。

自动类型转换遵循下面的规则：

①若参与运算的数据类型不同，则先转换成同一类型，然后进行运算。

②转换按数据长度增加的方向进行，以保证精度不降低。例如int类型和long类型运算时，先把 int 类型转成 long 类型后再进行运算。

③在赋值运算中，赋值号两边的数据类型不相同时，将把右边表达式值的类型转换为左边变量的类型。如果右边表达式值的数据类型长度比左边长时，将丢失一部分数据。

④在赋值语句中，赋值号两边数据类型一定是相兼容的类型，如果等号两边数据类型不兼容，语句在编译时会报错。

（2）强制类型转换（显示类型转换）

自动类型转换不能实现目的时，可以显式进行类型转换，称为强制类型转换。

一般形式：（数据类型）（表达式）

注意：数据类型加小括号这个整体才是强制类型转换的符号，后面的表达式可以加小括号，也可以不加，如果不加的话则遵循就近原则，谁离得强制类型转换符号近，系统则强制类型转换谁。

例：int a;a=(int)1.5;a 的值为 1。

1.2、变量与常量

1. 变量

变量可以看作是存储数据的小盒子，一个小盒子里面只能存放一个具体的值，而且这个值可以改变。

例如：inta= 3；a 就是一个变量

2. 常量

常量是固定值，在程序执行期间不会改变。这些固定的值，又叫做字面量。

常量可以是任何的基本数据类型，可分为整型数字、浮点数字、字符、字符串和布尔值。

常量就像是常规的变量，只不过常量的值在定义后不能进行修改。

1.3、字符与字符串

1、字符就是单个字符，字符串就是多个字符的集合。

2、单个空白字符和空白字符串是两个概念，在 C++中字符就是单个字符，字符串是用\0 结尾的，字符和字符串在操作上也不同，复制等等是不一样的。

字符串简介：

字符串或串(String)是由数字、字母、下划线组成的一串字符。一般记为 s="a1a2…an"(n>=0)。它是编程语言中表示文本的数据类型。在程序设计中，字符串(string)为符号或数值的一个连续序列 ,如符号串(一串字符)或二进制数字串(一串二进制数字)。

1.4、运算符

1. 赋值运算符

在 C++里面，一个等号“=”代表的是赋值，赋值运算符是将赋值运算符右侧

的值赋值给左侧的变量。

2. 算术运算符

C++中有 5 个基本的算术运算符：+（加）-（减）、*（乘）、/（除）、%（取余）。

注意：

① 两个整数相除，得到的结果值保留整数位

② 取余运算符两边的数字必须都得是整数

3. 逻辑运算符

C++中有三个基本的逻辑运算符：&&（与）、||（或）、!（非）

逻辑非：经过逻辑非运算，其结果与原来相反。

逻辑与：若参加运算的某个条件不成立，其结果就不成立，只有当参加运算的所有条件都成立，其结果才成立。

逻辑或：若参加运算的某个条件成立，其结果就成立，只有当参加运算的所有条件都不成立，其结果才不成立。

4. 关系运算符

C++中有六个基本的关系运算符：>（大于）、>=（大于等于）、<（小于）、<=（小于等于）、 ==（等于）、 !=（不等于）

符号	**意义**	**举例**
>	大于	10>5
>=	大于等于	10>=10
<	小于	10<5
<=	小于等于	10<=10

==	等于	10==5
!=	不等于	10!=5

第 02 课 基本程序结构

2.1、顺序结构程序设计

1. 输入语句：

cin 是 C++的输入语句，与 cout 语句一样，为了叙述方便，常常把由 cin 和运算符”>>”实现输入的语句称为输入语句或 cin 语句。

cin 语句的一般格式为：

cin>>变量 1>>变量 2>>……>>变量 n；

与 cout 类似，一个 cin 语句可以分写成若干行，如

```
cin>>a>>b>>c>>d;
```

也可以写成

```
cin>>a;
cin>>b;
cin>>c;
cin>>d;
```

以上书写变量值均可以以从键盘输入：1 2 3 4

也可以分多行输入数据：

```
1
2 3
```

4

2. 输出语句：

cout 语句一般格式为：

cout<<项目 1<<项目 2<<…<<项目 n；

功能：

（1）如果项目是表达式，则输出表达式的值。

（2）如果项目加引号，则输出引号内的内容。

输出总结：

cout<<项目 1<<项目 2<<…<<项目 n;

①输出语句可以输出多项内容，用<<连接;

②如果输出项目是"2+3"，则输出 2+3;

③如果输出项目是 2+3，则输出 5;

④如果输出遇到 endl，则换行。

3. 输出格式控制：

2.2、分支结构程序设计

1. if-else 语句

一般形式：if（表达式）{

语句块 1

}else{

语句块 2

}

判断表达式的逻辑值，当表达式的值非 0，则执行语句块 1，当表达式的值为 0 的时候，执行语句块 2.

2. switch 语句

switch 语句是多分支选择语句，也叫开关语句。switch 语句基本格式及框架图如下：

```
switch（表达式）{
        case 常量表达式 1：[语句组 1] [break;]
        case 常量表达式 2：[语句组 2] [break;]
                ……
        case 常量表达式 n：[语句组 n] [break;]
        [default:语句组 n+1]
}
```

功能：首先计算表达式的值，case 后面的常量表达式值逐一与之匹配，当某一个 case 分支中的常量表达式值与之匹配时，则执行该分支后面的语句组，然后顺序执行之后的所有语句 ,直到遇到 break 语句或 switch 语句的右括号“}”为止。如果 switch 语句中包含 default，default 表示表达式与各分支常量表达式的值都不匹配时，执行其后面的语句组，通常将 default 放在最后。

规则：

（1）合法的 switch 语句中的表达式，其取值只能是整型、字符型、布尔型或者枚举型

（2）常量表达式是由常量组成的表达式，值的类型与表达式类型相同。

（3）任意两个 case 后面的常量表达式值必须各不相同，否则将引起歧义

（4）“语句组”可以使一个语句也可以是一组语句

（5）基本格式中的[ ]表示可选项

3. 分支语句嵌套

在 if 语句中又包含一个或多个 if 语句称为 if 语句的嵌套。一般形式如下：

```
if( )
        if( )语句 1
        else  语句 2
    else
        if( )语句 3
        else  语句 4
```

应当注意 if 与 else 的配对关系。else 总是与它上面最近的、且未配对的 if 配对。假如写成：

```
if( )
        if( )语句 1
    else
        if( )语句 2
        else  语句 3
```

编程序者把第一个 else 写在与第一个 if(外层 if)同一列上，希望 else 与第一个 if 对应，但实际上 else 是与第二个 if 配对，因为它们相距最近，而且第二个 if 并未与任何 else 配对。为了避免误用，最好使每一层内嵌的 if 语句都包含 else 子句(如本节开头列出的形式)，这样 if 的数目和 else 的数目相同，从内层到外

层一一对应，不致出错。

如果 if 与 else 的数目不一样，为实现程序设计者的企图，可以加花括号来确定配对关系。例如：

```
if( )
    {
        if ( ) 语句 1
    } //这个语句是上一行 if 语句的内嵌 if
    else  语句 2//本行与第一个 if 配对
```

这时{ }限定了内嵌 if 语句的范围，{ }外的 else 不会与{ }内的 if 配对。关系清楚，不易出错。

2.3、循环结构程序设计

1. while 语句

while 死循环结构:

（1）格式：

```
while(1) {
        循环语句;
}
```

（2）功能

不断地执行循环体中的语句。

代码有两部分：

（1）while(1)

（2）花括号{　循环语句; }

while（表达式）语句的格式与功能

（1）格式

格式 1：

while(表达式)

语句;

格式 2：

while(表达式){

语句 1;

语句 2;

……

}

（2）功能

当表达式的值非 0 时，不断地执行循环体中的语句。所以，用 while 语句实现的循环被称为“当型循环”。

2. for 语句

for 循环格式

格式 1:

for(循环变量初始化;循环条件;循环变量增量)

语句;

格式 2:

for(循环变量初始化;循环条件;循环变量增量) {

语句 1;

语句 2;

……

}

3. do-while 语句

do{

语句;

}while(表达式);

do-while 循环与 while 循环的不同在于：它先执行循环中的语句，然后再判断表达式是否为真 ;如果为真则继续循环 ,如果为假 ,则终止循环。因此 ,do-while 循环至少要执行一次循环语句。

4、循环结构嵌套

循环的嵌套：

（1）一个循环体内包含着另一个完整的循环结构，就称为嵌套循环。

（2）内嵌的循环又可以嵌套循环，从而构成多重循环。

（3）三种循环可以相互嵌套。下面都是合法的嵌套。

①while()　　②do　　③for(; ;)

{　…　{　…

{ …

while()　　do

for(;　;)

{…..　　{…

{ …

} }while(); }

… …

…

} }while();

}

④while() ⑤do

⑥for(; ;)

{ … { …

{ …

do for(; ;)

while()

{….. {…

{ …

}while(); } }

… …

…

} }while();

}

注意：

①嵌套的循环控制变量不应同名，以免造成混乱。

②内循环变化快，外循环变化慢。

③正确确定循环体。

④循环控制变量与求解的问题挂钩。

5. break 语句

break 语句的格式和用法

（1）格式

break；

（2）功能

中断所在循环体，跳出本层循环。

第 03 课 数组

3.1、一维数组及二维数组

1. 一维数组

申请 10 个整数数据类型的变量可以这么写：int a[10];

int a[10];这行语句代表同时定义了 10 个整型变量，就如同 10 个“小房子”并排放在了一起。

那么我们如何使用这些变量呢？

首先，[ ]里的数字表示需要定义的变量的个数，我们这里定义了 10 个。这 10 个变量分别用 a[0]、a[1]、a[2]、a[3]、a[4]、a[5]、a[6]、a[7]、a[8]、a[9]来表示。

注意：我们要表达数组中某一个元素的格式是：数组名[下标]。在 C++中，下标是从 0 开始的，所以一个大小为 n 的数组，它的有效下标是 0~n-1。

0 是下标，a[0]用来存值

数组：由具有相同数据类型的固定数量的元素组成的结构。

例如：int a[10];

double b[10], c[5];

注意：数组定义时的一个关键点是数组的长度如何选择。

数组元素的引用：

(1)下标可以是整型常量或整型表达式；

a[3]=3;

或: int i=3;

a[i]=3;

(2)下标在 0~4 之内, 即 a[0]~a[4]，

注意：下标不要超范围

(3)可以单独针对每个元素赋值，

如：a[4] = 5;

也可以这么用：

int i = 4;

a[i] = 5;

(4)每个元素都是一个变量，数组是“一组变量”，而不是一个变量。

2. 二维数组

二维数组定义的一般格式：

类型名 数组名[常量表达式 1][常量表达式 2];

通常二维数组中的第一维表示行下标，第二维表示列下标。

行下标和列下标都是从 0 开始的。例如：

int num[4][6];

二维数组的使用与一维数组类似，引用的格式为：

数组名[下标 1][下标 2]

使用数组时特别注意下标不能越界。

使用二维数组时，需要区分是处理行数据、列数据，还是处理所有数据的行列下标。

遍历一个二维数组要使用二重循环。

3.2、数组的输入和输出

1. 一维数组的输入输出。

利用一层 for 循环实现控制下标的变化从而实现对一维数组元素的输入以及输出。

例如：for(int i=0; i<5; ++i){

```
    cin>> a[i];
```

}

利用循环实现了对一维数组元素的输入。

2. 二维数组的输入输出

二维数组的输入方式有两种：

（1）按行输入：

输入 n 行 m 列数据(n、m 均小于 100)：

int n, m, a[105][105];

```
cin>> n >> m;
for(int i=1; i<=n; i++){          //行数变化
    for(int j=1; j<=m; j++){   //列数变化
        cin>> a[i][j];            //按行输入
    }
}
```

（2）按列输入：

输入 n 行 m 列数据(n、m 均小于 100) ：

```
int n, m;
cin>> n >> m;
for(int j=1; j<=m; j++){          //列数变化
    for(int i=1; i<=n; i++){    //行数变化
        cin>> a[i][j];            //按列输入
    }
}
```

二维数组的输出我们只需要掌握住按行输出即可：

```
for(int i=1;i<=n;i++){//控制行数
for(int j=1;j<=m;j++){
    cout<<a[i][j]<<" ";
    }
    cout<<endl;
}
```

3.3、数组元素的遍历

1. 一维数组的遍历

将存放在一维数组中的元素依次查看一遍并寻找符合条件的数的过程就是对一维数组的遍历。

2. 二维数组的遍历

二维数组遍历和一维数组遍历类似，只不过在遍历到一维元素时，由于元素是一维数组还需要遍历，构成双重循环。使用双重循环遍历二维数组时，外层循环的次数使用数组元素的行数来进行控制，内层循环的次数是使用每个一维数组的元素的，也就是二维数组的列数来进行控制。

3.4、数组元素排序

1. 选择排序

选择排序：是一种简单直观的排序算法。它的工作原理是每一次从待排序的数据元素中选出最小（或最大）的一个元素，存放在序列的起始位置，然后，再从剩余未排序元素中继续寻找最小（大）元素，然后放到已排序序列的末尾。以此类推，直到全部待排序的数据元素排完。

例如，6 个数按照从小到大排序：

```
#include <iostream>
using namespace std;
int main( ) {
    int a[6] , i , j , t;
    for ( i=0 ; i<6 ; i++)
```

```
        cin>> a[i];
    for ( i=0 ; i<5; i++)
        for ( j=i+1 ; j<6; j++)
            if ( a[i]>a[j] ) {
                t=a[i] ;
                a[i]=a[j] ;
                a[j]=t ;
            }
    for ( i=0 ; i<6 ; i++)
        cout<< a[i] << " ";
    return 0 ;
}
```

可推广到 n 个数的选择排序

2. 冒泡

冒泡排序是一种计算机科学领域的较简单的排序算法。它重复地走访过要排序的元素列，依次比较两个相邻的元素，如果他们的顺序错误就把他们交换过来。走访元素的工作是重复地进行直到没有相邻元素需要交换，也就是说该元素已经排序完成。

这个算法的名字由来是因为越大（小）的元素会经由交换慢慢“浮”到数列的顶端（升序或降序排列），就如同碳酸饮料中二氧化碳的气泡最终会上浮到顶端一样，故名“冒泡排序”。

例如：5 个数按照从小到大排序：

```
#include <iostream>
using namespace std;
int main( ) {
    int a[5] , i , j , t;
    for ( i=0 ; i<5 ; i++)
        cin>> a[i];
    for ( i=0 ; i<4 ; i++)//比较了四轮
        for ( j=0 ; j<4 ; j++)//每轮需要四次比较
            if ( a[j]>a[j+1] ) {
                t=a[j] ;
                a[j]=a[j+1] ;
                a[j+1]=t ;
            }
    for ( i=0 ; i!=5 ; i++)
        cout<< a[i] << " ";
    return 0;
}
```

可推广到 n 个数的冒泡排序。

3. 桶排序

现在如果需要将输入的 5 个数（范围是 0~9）从小到大排序，该怎么办

例如输入 2 5 2 1 8 ,则输出 1 2 2 5 8。首先我们先申请一个大小为 10 的数组,int a[10],编号为 a[0]~[9]，并初始化为 0。

我们现在只需将 “小房间的值加 1”就可以了，例如：2 出现了，就将 a[2]的值加 1。

实现过程：

```
#include<iostream>
using namespace std;
int main(){
    int  i,j,t, a[10]={0};      //数组初始化为 0
    for(i=1;i<=5;i++){ //循环读入 5 个数
    cin>>t;      //把每一个数读到变量 t 中
        a[t]++;      // t 所对应小房子中的值增加 1
    }
    for(i=0;i<=9;i++){ //依次判断 0~9 这个 10 个小房子
        for(j=1;j<=a[i];j++)  //出现了几次就打印几次
    cout<<i<<  ‘  ‘;
    }
    return 0;
}
```

这种形式的排序就是桶排序，桶排序是要借助于数组来实现的，我们将每个数组元素看作一个桶，每出现一个数，就在对应编号的桶里面放一个小旗子，最后只要数一数每个桶里面有几个小旗子就 OK 了。

3.5、字符数组

1. 一维字符数组

我们将 char 类型定义的数组叫做字符数组，也可以叫它字符串。

字符数组的定义格式如下：char 数组名[元素个数];

cin 和 scanf 输入方式是没法读入空格，回车，tab 的，如果我们的字符串中需要空格，可以用 gets()来读入一行字符串

例如：

```
char a[10];

gets(a);

cout<<a;
```

```
gets 函数已经被淘汰，这里应该使用
getline(cin, a);
```

注意：gets()函数可以保留空格，遇到回车结束，需要一个头文件<cstdio>

字符数组初始化的两种方式：

char a[10]；

（1）char a[10]={ ‘c’ ,’ o’ ,’ d’ 。,’ u’ ,’ c’ ,’ k’ };//一些单个字符

（2）char a[10]={ “coduck"}; //一个字符串

输入：

char a[10]；

（1）无空格的字符串输入：

1.利用循环输入：

```
for(int i=0;i<=9;i++){
   cin>>a[i];  //输入固定长度为 10 的字符数组。
```

}

2.直接 cin;

cin>>a; //可以输入小于等于 10 的字符数组

有空格的字符串输入：

1.gets(a); //gets 函数可以保留空格，遇到回车结束！加头文件<cstdio>

说明：遇到没有空格的字符串，可以用 cin>>数组名;来输入。遇到有空格的字符串，可以用 gets(数组名);来输入

输出：

cout<<a;

说明：cout 输出遇到‘\0’结束。

2. 二维字符数组

我们可以借助一个二维数组来存储多个字符串，这个二维字符数组的每一行都是一个单独的字符串。

第 04 课 函数

4.1、函数的定义和使用

函数定义

前面我们用过了很多 C++标准函数，但是这些标准函数并不能满足所有需求。当我们需要特定的功能函数时，这就需要我们要学会自定义函数,根据需求定制想要的功能。

函数定义的语法：

返回类型　　函数名（参数列表）

{

　　函数体

 }

关于函数定义的几点说明：

（1）自定义函数符合“根据已知计算未知”这一机制，参数列表相当于已知，是自变量，函数名相当于未知，是因变量。如 1.1 参考代 2 中 max 函数的功能是找出两个数的最大数，参数列表中 x 相当于已知——自变量，max 函数的值相当于未知——因变量。

（2）函数名是标识符，一个程序中除了主函数名必须为 main 外，其余函数的名字按照标识符的取名规则命名。

（3）参数列表可以是空的，即无参函数，也可以有多个参数，参数之间用逗号隔开，不管有没有参数，函数名后的括号不能省略。参数列表中的每个参数，由参数类型说明和参数名组成。如 max 函数的参数列表数有两个参数，两个参数类型分别是 int，int，两个参数名分别是 a，b。

（4）函数体是实现函数功能的语句，除了返回类型是 void 的函数，其他函数的函数体中至少有一条语句是“return 表达式；”用来返回函数的值。执行函数过程中碰到 return 语句，将在执行完 return 语句后直接退出函数，不去执行后面的语句。

（5）返回值的类型一般是前面介绍过的 int、double、char 等类型，也可以是数组。有时函数不需要返回任何值，例如函数可以只描述一些过程用 printf 向屏幕输出一些内容，这时只需定义的数返回类型为 void，并且无须使用 return

返回函数的值。

函数使用时，函数名必须与函数定义时完全一致，实参与形参个数相等，类型一致，按顺序一一对应。被调用函数必须是已存在的函数，如果调用库函数，一定要包含相对应的头文件。

4.2、函数的递归调用

函数之间有三种调用关系：主函数调用其他函数、其他函数互相调用、函数递归调用

C++程序从主函数开始执行，主函数由操作系统调用，主函数可以调用其他函数，其他函数之间可以互相调用，但不能调用主函数，所有函数是平行的，可以嵌套调用，但不能嵌套定义。

4.3、变量的作用域：局部变量和全局变量

作用域描述了名称在文件的多大范围内可见可使用。C++程序中的变量按作用域来分，有全局变量和局部变量

全局变量：定义在函数外部没有被花括号括起来的变量称为全局变量。全局变量的作用域是从变量定义的位置开始到文件结束。由于全局变量是在函数外部定义的，因此对所有函数而言都是外部的，可以在文件中位于全局变量定义后面的任何函数中使用。

局部变量：定义在函数内部作用域为局部的变量局部变量。函数的形参和在该函数里定义的变量都被称为该函数的局部变量。

注意：全局变量和局部变量都有生命周期，变量从被生成到被撤销的这

段时间就称为变量的生存期，实际上就是变量占用内存的时间。局部变量的生命周期是从函数被调用的时刻开始到函数结束返回主函数时结束。而全局变量的生命周期与程序的生命周期是一样的。若程序中全局变量与局部变量同名，且同时有效，则以局部变量优先。即在局部变量的作用范围内，全局变量不起作用。

第 05 课 简单算法

5.1、进制转换

1、r 进制数(非十进制数)转化成十进制数：

各种进位制转换为十进制的方法：分别写出二进制数、八进制数和十六进制数的按权展开式，再按十进制运算规则求和得到的值，即为转换后的十进制数。

2、十进制数转化成 r 进制数：

分整数和小数两部分分别转换处理，最后再求和。

整数部分的转换方法是：不断的除以 r 取余数，直到商为 0，余数从下到上排列（除 r 取余，逆序排列）；小数部分的转换方法是：不断乘以 r 取整数，整数从上到下排列（乘 r 取整，顺序排列）。

3、二进制、八进制、十六进制数间的转换：

二进制、八进制、十六进制之间的对应规则如下：每 3 位二进制对应 1 位八进制数；每 4 位二进制对应 1 位十六进制数。

（1）二进制转化成八(十六)进制：分为如下三个步骤

整数部分：小数点为基准从右向左按三(四)位进行分组

小数部分：小数点为基准从左向右按三(四)位进行分组

不足补零

（2）八（十六）进制转换为二进制

八进制数转换成二进制数：只需将 1 位八进制数转为 3 位二进制数。

十六进制数转换成二进制数 :只需将 1 位十六进制数转为 4 位二进制数。

5.2、模拟算法

在我们所解决的题目中，有一类问题是模拟一个游戏的对弈过程，或者模拟一项任务的操作过程，进行统计计分，判断输赢等。这些问题很难建立数学模型用特定算法解决，一般只能采用“模拟”法。用模拟法解决必须关注以下几个问题：审题要仔细，游戏规则不能错；分析要全面，各种特例不能丢；编程要细心，测试运行要到位。

例如：

有一天，一只蚱蜢像往常一样在草地上愉快地跳跃，它发现了一条写满了英文字母的纸带。蚱蜢只能在元音字母（A、E、I、O、U）间跳跃，一次跳跃所需的能力是两个位置的差。纸带所需的能力值为蚱蜢从纸带开头的前一位置根据规则调到纸带结尾的后一个位置的过程中能力的最大值。

蚱蜢想知道跳跃纸带所需能力值（最小）是多少。如下图所示，纸带所需能力值（最小）是 4.

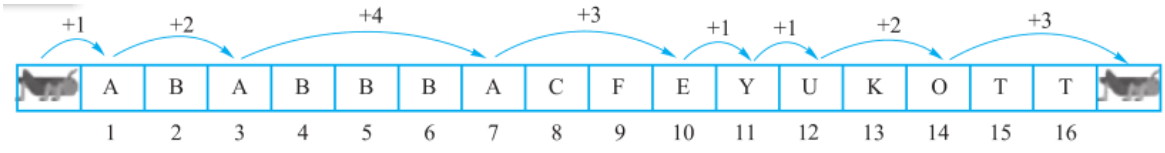

图 9.3-1　蚱蜢跳跃示意图

【输入格式】

一行一个字符串，字符串场不超过 100.

【输出格式】

一行一个整数，代表（最小）能力值。

【输入样例】

KMLPTGFHNBVCDRFGHNMBVXWSQFDCVBNHTJKLPMNFVCKMLPTGFH

NBVCDRFGHNMBVXWSQFDCVBNHTJKLPMNFVC

【输出样例】

85

【问题分析】

从头到尾枚举纸带的每一个字母，按照规则模拟蚱蜢在元音字母间跳跃的过程，打擂台记录能力值。

【示例代码】

```
#include<bits/stdc++.h>
using namespace std;
int main(){
    freopen("grasshopper.in","r",stdin);
    freopen("grasshopper.out","w",stdout);
    string a;
    cin>>a;
    int ans=0,pre=0;
    for(int i=0;i<a.length();i++){
        if(a[i]=='A'||a[i]=='E'||a[i]=='I'||a[i]=='O'||a[i]=='U'||a[i]=='Y'){
```

```
            if(ans<=(i+1-pre)){
                ans=i+1-pre;
            }
            pre=i+1;
        }
    }
    if(ans<=a.length()+1-pre){
        ans=a.length()+1-pre;
    }
    cout<<ans<<endl;
    return 0;
}
```

5.3、枚举算法

计算机的特点之一就是运算速度快，善于重复做一件事。“穷举”正是基于这一特点的最古老的算法。它一般是在一时找不出解决问题的更好途径，即从数学上找不到求解的公式或规则时，根据问题中的“约束条件”，将解的所有可能情况一一列举出来，然后再逐个验证是否符合整个问题的求解要求，从而求得问题的可行解或者最优解。

例题：火柴棒等式

【问题描述】

给出 n 根火柴棒，可以拼出多少个形如“A+B=C”的等式？

等式中的 A、B、C 是用火柴棒拼出的整数（若该数非零，则最高位不能是 0）。

用火柴棒拼数字 0~9 的拼法如下图所示。

需要注意以下几点：

（1） 加号与等号各自需要两根火柴棒。

（2） 如果 A ≠ B，则 A+B=C 与 B+A=C 视为不同的等式（A、B、C 均大于或等于 0）。

（3） n 根火柴棒必须全部用上（n≤24）。

【输入样例】

14

【输出样例】

2

【样例说明】

两个等式分别为：0+1=1 和 1+0=1。

【问题分析】

首先，预处理每个数字（0~9）需要用几根火柴棒，存储在数组 f 中。然后，穷举 a 和 b，算出它们的和 c，再判断下列约束条件是否成立：f（a）+ f（b）+ f （c）= n-4。现在的问题是：a 和 b 的范围有多大？可以发现尽量用数字 1 拼成的数比较大，分析可知最多不会超过 1111。程序实现时，分别用三个循环语句预处理好所有两位数、三位数、四位数构成所需要的火柴棒数量。

【示例代码】

```
#include<bits/stdc++.h>
```

```
using namespace std;
int f[10000];
int main(){
    freopen("matches.in","r",stdin);
    freopen("matches.out","w",stdout);
    f[0]=6;f[1]=2;f[2]=5;f[3]=5;f[4]=4;
    f[5]=5;f[6]=6;f[7]=3;f[8]=7;f[9]=6;
    for(int i=10;i<=99;++i){
        f[i]=f[i/10]+f[i%10];
    }
    for(int i=100;i<=999;++i){
        f[i]=f[i/100]+f[i/10%10]+f[i%10];
    }
    for(int i=1000;i<=9999;++i){
        f[i]=f[i/1000]+f[i/100%10]+f[i/10%10]+f[i%10];
    }
    int n,total=0;
    cin>>n;
    for(int a=0;a<=1111;++a){
        for(int b=0;b<=1111;++b){
            int c=a+b;
            if(f[a]+f[b]+f[c]==n-4){
```

```
                total++;
            }
        }
    }
    cout<<total<<endl;
    return 0;
}
```

第 06 课 基本数据结构

6.1、结构体

1、结构体的定义：

在存储和处理大批量数据时，一般会使用数组来实现，但是每一个数据的类型及含义必须一样。如果需要把不同类型、不同含义的数据当作一个整体来处理，比如 1000 个学生的姓名、性别、年龄、体重、成绩等，怎么办？C++提供了结构体。

C++中的结构体是由一系列具有相同类型或不同数据类型的数据构成的数据集合。使用结构体，必须要先声明一个结构体类型，再定义和使用结构体变量。结构体类型的声明格式如下：

```
struct    类型名{
    数据类型 1      成员名 1；
    数据类型 2      成员名 2；
```

…

};

也可以把结构体类型声明和变量定义在一起，格式如下：

struct 类型名{

数据类型 1 成员名 1；

数据类型 2 成员名 2；

…

}变量名；

2、结构体变量的使用：

结构体变量具有以下特点：

（1）可以对结构体变量的整体进行操作。例如，swap（a[i],a[j]）。

（2）可以对结构体变量的成员进行操作。

引用结构体变量中的成员的格式为：结构体变量名.成员名

（3）结构体变量的初始化方法与数组类似。

student s={"xiaomming",'f',16,169};

6.2、栈

1、栈的基本概念：

栈（Stack）是限制在表的一端进行插入和删除操作的线性表

后进先出 LIFO（Last In First Out）

先进后出 FILO（First In Last Out）

栈顶（Top）:允许进行插入、删除操作的一端，又陈伟表尾。用栈顶指针（top）

来指示栈顶元素。

空栈：当表中没有元素时称为空栈。

2、栈的顺序表示与实现：

栈的顺序存储结构简称为顺序栈，和线性表类似，用一维数组来存储栈。

3、栈的应用

（1）括号匹配检查

【题目描述】

假设表达式中允许包含圆括号和方括号两种括号 ,其嵌套的顺序随意 ,如([]())或[([][])]等为正确的匹配，[(])或([]()或(()))均为错误的匹配。本题的任务是检验一个给定的表达式中的括号是否匹配正确 。

输入一个只包含圆括号和方括号的字符串，判断字符串中的括号是否匹配，匹配就输出“OK”，不匹配就输出“Wrong”。

【输入】

一行字符，只含有圆括号和方括号，个数小于 255

【输出】

匹配就输出一行文本“OK“，不匹配就输出一行文本”Wrong“

【样例输入】

[(])

【样例输出】

Wrong

【解决思路】

使用栈来实现，首先输入字符串，遍历字符串：

如果是左括号，进栈；

如果是右括号，跟栈顶数据比较

如果和栈顶的左括号匹配，出栈

如果不匹配，输出“Wrong"

遍历结束后，栈中还有括号，输出“Wrong”；没有，输出“Ok”

【代码示例】

```
#include<iostream>
#include<string>
using namespace std;
int main()
{
    char a[256];
    string s;
    int i,top;
    cin>>s;
    top=0;
    for(i=0;i<s.size();i++)
    {
        if(s[i]=='('||s[i]='[')
          {
            a[++top]=s[i];;
          }
```

```
        if(s[i]==')')
        {
            if(a[top]=='(') top--;
            else
             {    top++;    }
         }
         if(s[i]==']')
        {
            if(a[top]=='[') top--;
            else
             {    top++;    }
         }
    }
    if(top==0) cout<<"OK"<<endl;
    else cout<<"Wrong"<<endl;
    return 0;
}
```

（2）铁轨问题

6.3、队列

1、队列定义

队列是一种先进先出（FIFO）的线性表。只能在线性表的一端进行插入操作，

在另一端进行删除操作。类似与生活中的排队购票，先来先买，后来后买。

在不断入队、出队的过程中，队列将会呈现初以下几种状态：

队满：队列空间已被全被占用

队空：队列中没有任何元素。

溢出：当队列满时，却还有元素要入队，就会出现“上溢”；当队列已空，却还要做“出队”操作，就会出现“下溢”。两种情况合在一起称为队列的“溢出”。

2、队列的应用

【题目描述】

假设在周末舞会上，男士们和女士们进入舞厅时，各自排成一队。跳舞开始时，依次从男队和女队的队头上各出一人配成舞伴。规定每个舞曲能有一对跳舞者。若两队初始人数不相同，则较长的那一队中未配对者等待下一轮舞曲。现要求写一个程序，模拟上述舞伴配对问题。(0<m,n,k<1000)

【输入】

第一行男士人数 m 和女士人数 n；

第二行舞曲的数目 k。

【输出】

共 k 行，每行两个数，表示配对舞伴的序号，男士在前，女士在后。

【样例输入】

2 4

6

【样例输出】

1 1

2 2

1 3

2 4

1 1

2 2

【解题思路】

显然，舞伴配对的顺序符合“先进先出”，所以用两个队列分别存放男士队伍和女士队伍，然后模拟 k 次配对：每次取各队队头元素“配对”输出，并进行“出队”和重新“入队”的操作。

【代码示例】

```
#include<iostream>
using namespace std;
int main(){
    int a[10001],b[10001],f1=1,f2=1,r1,r2,k1=1;
    int m,n,k;
    cin>>m>>n>>k;
    r1=m;r2=n;
    for(int i=1;i<=m;i++) a[i]=i;
    for(int j=1;j<=n;j++) b[j]=j;
    while(k1<=k){
```

```
            cout<<a[f1]<<" "<<b[f1]<<endl;
            r1++;a[r1]=a[f1];f1++;
            r2++;b[r2]=b[f2];f2++;
            k1++;
        }
        return 0;
}
```

6.4、树

1、树

（1）树的定义

树(Tree)是 n(n≥0)个结点的有限集,它或为空树(n=0) ;或为非空树,对于非空树 T:

有且仅有一个称之为根的结点;

除根结点以外的其余结点可分为 m(m> 0)个互不相交的有限集 T1，T2, ...Tm, 其中每一个集合本身又是一棵树，并且称为根的子树(SubTree)。

（2）基本术语

根——即根结点(没有前驱)

叶子——即终端结点(没有后继)

森林——指 m 棵不相交的树的集合（例如删除 A 后的子树个数)

有序树——结点各子树从左至右有序，不能互换(左为第一)

无序树——结点各子树可互换位置。

双亲——即上层的那个结点(直接前驱)

孩子——即下层结点的子树的根(直接后继)

兄弟——同一双亲下的同层结点(孩子之间互称兄弟)

堂兄弟——即双亲位于同一层的结点（但并非同一双亲)

祖先——即从根到该结点所经分支的所有结点

子孙——即该结点下层子树中的任一结点

结点——即树的数据元素

结点的度——结点挂接的子树数

结点的层次——从根到该结点的层数（根结点算第一层)

终端结点——即度为 0 的结点 ,即叶子

分支结点——即度不为 0 的结点(也称为内部结点)

树的度——所有结点度中的最大值

树的深度——指所有结点中最大的层数

（3）树的存储结构

方法 1:数组,称为“父亲表示法”

```
const int m = 10;//树的结点数
struct node
{
    int data, parent;//数据域、指针域
};
node tree[m];
```

优缺点:利用了树中除根结点外每个结点都有唯一的父结点这个性质。很容易找到树根,但找孩子时需要遍历整个线性表。

方法 2:树型单链表结构,称为“孩子表示法”。每个结点包括一个数据域和一个指针域(指向若干子结点)。称为”孩子表示法”。假设树的度为 10 ,树的结点仅存放字符,则这棵树的数据结构定义如下:

```
const int m = 10;   / /树的度
typedef   struct node ;
typedef node *tree;
    struct node
   {
        char data;//数据域
        tree child[m];//指针域，指向若干孩子结点
   }
tree t;
```

缺陷:只能从根(父)结点遍历到子结点,不能从某个子结点返回到它的父结点。但程序中确实需要从某个结点返回到它的父结点时;就需要在结点中多定义一个指针变量存放其父结点的信息。这种结构又叫带逆序的树型结构。

方法 3 :树型双链表结构,称为”父亲孩子表示法”。每个结点包括一个数据域和二个指针域(一个指向若干子结点，一个指向父结点)。假设树的度为 10 ,树的结点仅存放字符,则这棵树的数据结构定义如下:

```
    const int m= 10; / /树的度
   typedef struct node;
```

```
typedef node *tree;//声明 tree 是指向 node 的指针类型
struct node
{
      char data;   // 数据域
     tree   child[m]; //指针域 ,指向若干孩子结点
     tree father ;   / /指针域,指向父亲结点
};
tree t;
```

方法 4 :二叉树型表示法,称为”孩子兄弟表示法”。也是一种双链表结构,但每个结点包括一个数据域和二个指针域(一个指向该结点的第一个孩子结点,一个指向该结点的下一个兄弟结点)。称为”孩子兄弟表示法”。假设树的度为 10 , 树的结点仅存放字符,则这棵树的数据结构定义如下:

```
typedef struct node; ,
typedef node *tree;
struct node
{
      char data;   //数据域
      tree firstchild, next;
      //指针域,分别指向第一个孩子结点和下一个兄弟结点};
} ;
tree t;
```

（4）树的遍历

在应用树结构解决问题时,往往要求按照某种次序获得树中全部结点的信息,这种操作叫作树的遍历。遍历的方法有多种,常用的有:

A、先序(根)遍历:先访问根结点,再从左到右按照先序思想遍历各棵子树。如上图先序遍历的结果为：125634789 ;

B、后序(根)遍历:先从左到右遍历各棵子树,再访问根结点。如上图后序遍历的结果为: 562389741 ;

C、层次遍历:按层次从小到大逐个访问,同一层次按照从左到右的次序。如上图层次遍历的结果为: 123456789 ;

D、叶结点遍历:有时把所有的数据信息都存放在叶结点中,而其余结点都是用来表示数据之间的某种分支或层次关系,这种情况就用这种方法。如上图按照这个思想访问的结果为: 56389 ;

2、二叉树

（1）二叉树的定义

二叉树(Binary Tree)是 n(n≥0)个结点所构成的集合,它或为空树(n= 0) ;或为非空树,对于非空树 T:

a)有且仅有一个称之为根的结点;

b)除根结点以外的其余结点分为两个互不相交的子集T和T2,分别称为的左子树和右子树，且和 72 本身又都是二叉树。

（2）二叉树的基本特点

a)结点的度小于等于 2

b)有序树（子树有序，不能颠倒）

二叉树的五种不同的形态

（3）二叉树的性质

性质 1：在二叉树的第 i 层上之多有 2^(i-1)个结点

性质 2：深度为 k 的二叉树至多有 2^k-1 个结点

性质 3：对于任何一棵二叉树，若 2 度的结点数有 n_2 个，则叶子数 n_0 必定为 n_2+1（即 n_0=n_2+1）

特殊形态的二叉树

a)满二叉树：一颗深度为 k 且有 2^k-1 个结点的二叉树。

b)完全二叉树：深度为 k 的，有 n 个结点的二叉树，当且仅当其每一个结点都与深度为 k 的满二叉树中编号从 1 至 n 的结点一一对应

性质 4：具有 n 个结点的完全二叉树的深度必为[log_2n]+1

性质 5：对完全二叉树，若从上至下、从左至右编号，则编号为 i 的结点，其左孩子编号必为 2i，其右孩子编号必为 2i+1;其双亲的编号为 i/2;

（4）二叉树的顺序存储

实现：按满二叉树的结点层次编号，依次存放二叉树中的数据元素

（5）遍历二叉树

遍历定义——指按某条搜索路线遍访每个结点且不重复(又称周游)。

遍历用途——它是树结构插入、删除、修改、查找和排序运算的前提，是二叉树一切运算的基础和核心。

DLR——先序遍历,即先根再左再右

LDR——中序遍历,即先左再根再右

LRD——后序遍历,即先左再右再根

6.5、图

1、图的相关概念

（1）图是由一个顶点的集合 V 和一个顶点间关系的集合 E 组成:记为 G=(V,E)

V :顶点的有限非空集合。

E :顶点间关系的有限集合(边集)。

存在一个结点 v，可能含有多个前驱结点和后继结点。

（2）图中顶点之间的连线若没有方向,则称这条连线为边,称该图为无向图。

（3）图中顶点之间的连线若有方向,则称这条连线为弧,则称该图为有向图。

（4）完全图稠密图稀疏图

在一个无向图中,如果任意两顶点都有一条直接边相连接,则称该图为无向完全图。一个含有 n 个顶点的无向完全图中，n(n-1)/2 条边。

在一个有向图中,如果任意两顶点之间都有方向互为相反的两条弧相连接,则称该图为有向完全图。在一个含有 n 个顶点的有向完全图中，n(n-1)条边。

若一个图接近完全图,称为稠密图。边数很少的图被称为稀疏图。

（5）度

a)顶点的度(degree)是指依附于某顶点 v 的边数,通常记为 TD(v)

结论:图中所有顶点的度=边数的两倍

b)出度入度

对有向图来说

顶点的出度:以顶点 v 为始点的弧的数目,记为 OD(v)。顶点的入度:以顶点 v 为终点的弧的数目,记为 ID(v)。顶点的度: TD(v)=ID(v) + OD(v).

（6）权 网络

与边有关的数据信息称为权

边上带权的图称为网图或网络

弧或边带权的图分别称为有向网或无向网

（7）路径

简单路径：序列中顶点不重复出现的路径

简单回路（简单环）：除第一个顶点与最后一个顶点之外，其他顶点不重复出现的回路。

2、图的存储结构

（1）邻接矩阵（有向图，无向图，网图分别如何存储）

（2）邻接表（了解）

邻接矩阵：代码书写简单，找邻接点慢

邻接表：代码书写较复杂，找邻接点快

第 07 课 指针

7.1、概念

（1）指针也是一个变量。和普通变量不同的是，指针变量李存储的数据是一个内存地址。

（2）指针变量的定义：

类型名　*指针变量名

int *p;

（3）指针变量赋值

a）先定义变量，再赋值

int a=3,*p;

p=&a;

b）定义变量的同时，进行赋值。

int a=3,*p=&a;

注意：只能用同类型变量的地址进行赋值

7.2、引用与运算

1、指针的引用

引用指针时，首先要理解指针变量与普通变量的区别和对应关系。例如，定义一个指针变量“int *p;”和一个普通变量“int a;”，关于两者之间的各种引用方式对应关系如下：

（1）“p”等同于“&a”,表示的是内存地址

（2）“*p”等同于“a”,表示变量里存储的实际数据

（3）“*p=3;”等同于“a=3;”，表示变量的赋值方式。

2、指针的运算

如果定义的是局部指针变量，其地址就是随机的，直接操作会引发不可预测的错误。所以，指针变量一定要初始化后才能引用。

由于指针变量存储的是内存地址，所以也可以执行加法、减法运算，一般用

来配合数组进行寻址操作。

7.3、指针与数组。

在 C++中，数组名在一定意义上可以被看成指针。“数组的指针”是指整个数组在内存中的起始地址，“数组元素的指针”是数组中某个元素所占存储单元的地址。例如，"int a[10]，*p；p=a；“就表示定义了一个指针 p，指向 a 数组在内存中的起始地址 a[0]。一般可以使用“下标法”访问数组元素,如 a[5]；也可以使用“地址法”访问数组元素 ,因为数组名就代表数组在内存中的起始地址，也就是 a[0]的地址，如 a+4 就表示 a[4]的地址；当然，也可以通过“指针法”访问数组元素，通过数组的指针或者数组元素的指针访问数组元素,能使目标程序质量更高，占用内存更少，运行速度更快。

四、函数指针及扩展

程序中需要处理的数据都保存在内存空间，而程序以及函数同样也保存在内存空间。C++支持通过函数的人口地址(指针)访问函数。另一方面，有些函数在编写时要调用其他的辅助函数，但是尚未确定，在具体执行时，再为其传递辅助函数的地址。比如排序函数 sort(a，a+n，cmp)，其中的比较函数 cmp 是根据需要传递给 sort 的，就是传递了一个函数指针。

函数指针就是指向函数的指针变量，定义格式如下:

类型名(*函数名) (参数);

例如，"int(*f) (int a，int b)；"，规范来说，此处的“函数名”应该称为“指针的变量名”。

第 08 课 基本算法

8.1、高精度算法

计算机最初、也是最重要的应用就是数值运算。在编程进行数值运算时,有时会遇到运算的精度要求特别高,远远超过各种数据类型的精度范围;有时数据又特别大 ,远远超过各种数据类型的极限值。这种情况下,就需要进行“高精度运算”。高精度运算首先要处理好数据的接收和存储问题，其次要处理好运算过程中的“进位”和“借位”问题。

【题目描述】

输入 n，输出 n!的精确值，n!=1x2x3……xn,1<n<1000

【输入】

一个整数 n

【输出】

n！的值

【样例输入】

2

【问题分析】

假设已经计算好（n-1）!，那么对于求 n!，就是用一个整数去乘以一个高精度数。只要用 n 乘以（n-1）!的每一位（从低位开始），同时不断处理进位。

8.2、递推算法

“递推”是计算机解题的一种常用方法。利用“递推法"解题首先要分析归纳出“递推关系”。比如经典的斐波那契问题，用 f(i)表示第 i 项的值，则 f(1)=0,f(2)=1，在 n>2 时,,存在递推关系:f(n)=f(n-1)+f(n-2) 。

在递推问题模型中 ,每个数据项都与它前面的若干数据项(或后面的若干数据项）存在一定的关联，这种关联一般是通过一 个“递推关系式”来描述的。求解问题时，需要从初始的一个或若干个数据项出发，通过递推关系式逐步推进，从而推导出最终结果。这种求解问题的方法叫“递推法”。其中，初始的若干数据项称为“递推边界”。

解决递推问题有三个重点:一是建立正确的递推关系式;二是分析递推关系式的性质;三是根据递推关系式编程求解。

8.3、分治算法

“分治”是一种常用的解题策略。它是将一个难以直接解决的大问题 ,分解成若干规模较小的、相互独立的 、相同或类似的子问题，分而治之，再合成得到问题的解。根据“平衡子问题”的思想，一般会把问题分解成两个规模相等的子问题，也就是“二分法” ，比如经典的二分查找(由半查找)问题。

例：找伪币。

【问题描述】

给出 16 个一模一样的硬币,其中有 1 个是伪造的 ,并且那个伤造的硬币比真的硬币要轻一些，本题的任务是找出这个伪造的硬币。为了完成这一任务，将提供一合可用来比较两组硬币重量的仪器，可以指导两组硬币孰轻孰重。

【问题分析】

方法 1　穷举法

依次比较硬币 1 与硬币 2、硬币 3 和硬币 4、硬币 5 和硬币 6……最多通过 8 次比较来判断伪造币的存在并找出这个伪币。

方法 2　二分法

把 16 个硬币的情况看成一个大问题。 第一步,把这一大问题分成两个小问题，随机选择 8 个硬币作为第一组(A 组),剩下的 8 个硬币作为第二组(B 组)。第二步，利用仪器判断伪币在 A 组还是在 B 组中,如果在 A 组中则再把 A 组中的 8 个硬币随机分成 2 组，每组 4 个再去判断...这样,只要(必须)4 次比较一定能找出伪币。

方法 3　三分法

把 16 个硬币分成 3 组(A 组 5 个、B 组 5 个、C 组 6 个),利用仪器比较 A、B 两组，一次就可以判断出伪币在 A、B 、C 哪一组中。假如在 C 组中,则再把 C 组中的 6 个分成 3 组(2 个、2 个、2 个)，再用仪器比较一次判断出在哪一组。然后再比较 1 次就能判断出 2 个硬币中哪个是伪币。这样，只要 2~3 次比较便能找出伪币。

8.4、贪心算法

贪心法的基本思想

贪心法是从问题的某个初始解出发，采用逐步构造最优解的方法，向给定的目标前进。在每一个局部阶段， 都做一个“看上去最优的决策，并期望通过每一次所做的局部最优选择产生出一个全局最优解。做出贪心决策的依据称为“贪心策略”。要注意的是，贪心策略一旦做出，就不可再更改。

与递推不同的是，贪心严格意义上说只是一种策略或方法，而不是算法。推进的每一步不是依据某一个固定的递推式 ,而是做一个当时“看似最佳”的贪心选择(操作),不断将问题归纳为更小的相似子问题。所以，归纳、分析、选择正确合适的贪心策略，是解决贪心问题的关键。

8.5、搜索算法（宽度优先搜索、深度优先搜索）

1、宽度优先搜索

宽度优先搜索的基本思想

宽度优先搜索（Breadth First Search, BFS），简称宽搜，又称为广度优先搜索。它是从初始结点开始，应用产生式规则和控制策略生成第一层结点，同时检查目标结点是否在这些生成的结点中。若没有，再用产生式规则将所有第一层结点逐一拓展，得到第二层结点，并逐一检查第二层结点是否包含目标结点。若没有，再用产生式规则拓展第二层结点。如此依次拓展，检查下去，直至发现目标结点为止。如果拓展完所有结点，都没有发现目标结点，则问题无解。

在搜索的过程中，宽度优先搜索对于结点是沿着深度的断层拓展的。如果要拓展第 n+1 层结点，必须先全部拓展完第 n 层结点。对于同层结点来说，它们对于问题的解的价值是相同的。所以，这种搜索方法一定能保证找到最短的解序列。也就是说，第一个找到的目标结点，一定是应用产生式规则最少的。因此，宽度优先搜索算法适合求最少步骤或者最短解序列这类最优解问题。

2、深度优先搜索

深度优先搜索(Depth First Search，DFS),简称深搜，其状态“退回一步”的顺序符合“后进先出”的特点 ,所以采用“栈”存储状态。深搜的空间复杂度较小，

因为它只存储了从初始状态到当前状态的条搜索路径。但是深搜找到的第一个解不一定是最优解，要找最优解必须搜索整棵“搜索树”。所以，深搜适用于要求所有解方案的题目。

深搜可以采用直接递归的方法实现，其算法框架如下:

```
void dfs (dep:integer，参数表);{
    自定义参数;
    if(当前是目标状态){
        输出解或者作计数、评价处理;
    }else
        for(i = 1; i <=状态的拓展可能数; i++)
            if(第 i 种状态拓展可行){
                维护自定义参数;
                dfs(dep+1,参数表);
            }
}
```

8.6、动态规划算法

动态规划是一种将问题实例分解为更小的、相似的子问题,并存储子问题的解而避免计算重复的子问题,以解决最优化问题的算法策略。动态规划实际上就是一种排除重复计算的算法,更具体的说,就是用空间换取时间。

能采用动态规划求解的问题的一般要具有 3 个性质:

(1)最优化原理:问题的最优解所包含的子问题的解也是最优的

(2)无后效性:即某阶段状态一旦确定,就不受这个状态以后决策的影响。

(3)有重叠子问题:即子问题之间是不独立的,一个子问题在下一阶段决策中可能被多次使用到。

动态规划问题的一般解题步骤

1、判断问题是否具有最优子结构性质,若不具备则不能用动态规划。

2、把问题分成若干个子问题(分阶段)

3、建立状态转移方程(递推公式)。

4、找出边界条件。

5、将已知边界值带入方程。

6、递推求解。