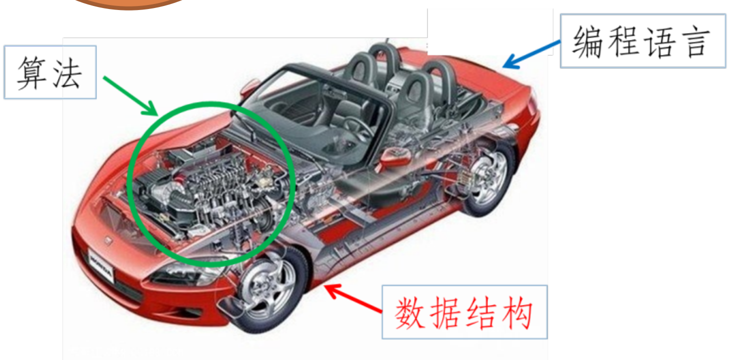
第一章(续)：算法

数据结构与算法

赵妍妍, 哈尔滨工业大学

数据结构

算法



两种算法的比较

- 思考： 求1+2+3+……+100结果的程序

```
int i, sum = 0, n = 100;
for ( i = 1; i <= n; i++)
{
    sum = sum + i;
}
printf ( " %d ", sum );
```

- 思考：这样是否是最高效的？

```
  sum = 1    + 2    + 3    + … + 99  + 100
  sum = 100 + 99   + 98   + … +2    + 1
2*sum = 101 + 101 + 101 + … + 101 +101
```

两种算法的比较

- 思考： 求1+2+3+……+100结果的程序

```
int i, sum = 0, n = 100;
for ( i = 1; i <= n; i++)
{
    sum = sum + i;
}
printf ( " %d ", sum );
```

很大的数

- 思考：这样是否是最高效的？

```
int i, sum = 0, n = 100;
sum = ( 1 + n ) *n /2;
printf ( " %d ", sum );
```

学习目标

- 了解算法、算法复杂性，掌握算法性能的评价方法
- 了解解决问题的一般过程和算法的逐步求精方法，掌握问题求解的基本过程和方法

提纲

- 1. 算法定义
- 2. 算法的特性
- 3. 算法设计的要求
- 4. 算法效率的度量方法
- 5. 算法的时间复杂度
- 6. 算法的空间复杂度

1.算法的定义

- 算法
  - 解决特定问题求解步骤的描述，在计算机中表现为指令的有限序列，并且每条指令表示一个或多个操作。
  - 程序是算法的一种实现，计算机按照程序逐步执行算法，实现对问题的求解。
- 思考：有没有通用的算法？

2.算法的特性

- 五个基本特性
  - 输入
    - 算法具有零个或多个输入
  - 输出
    - 算法至少有一个或多个输出
  - 有穷性
    - 算法在执行有限的步骤之后，自动结束而不会出现无限循环，并且每一个步骤在可接受的时间内完成
  - 确定性
  - 可行性

2.算法的特性

- 五个基本特性
  - 输入
  - 输出
  - 有穷性
  - 确定性
    - 算法的每一步骤都具有确定的含义，不会出现二义性
  - 可行性
    - 算法的每一步都必须是可行的，也就是说，每一步都能够通过执行有限次数完成。

3.算法设计的要求

什么是好的算法?

3.算法设计的要求

- 正确性
  - 至少应该具具有输入、输出和加工处理无歧义性、能正确反映问题的需求，能够得到问题的正确答案
  - 四个层次
    - 算法程序无语法错误
    - 对于合法的输入数据能够产生满足要求的输出结果
    - 对于非法的输入数据能够得出满足规格说明的结果
    - 对于精心选择的，甚至刁难的测试数据都有满足要求的输出结果

算法的正确性在大部分情况下不可能用程序来证明，而是用数学方法证明。

3.算法设计的要求

- 可读性
  - 算法设计的另一目的是为了便于阅读、理解和交流

```
<!doctype html><html><head></head><body>
<div id="box" style="width:252px;font:25px/25px 宋体;background:#000;color:#9f9;border:#999 20px ridge;text-shadow:2px 3px 1px #0f0;"></div>
<script>
var domain="www.zuidaima.com";
var author="zuidaima";
var map=eval("["+Array(23).join("0x801,")+"0xfff]");
var tatris=[[0x6600],[0x2222,0xf00],[0xc600,0x2640],[0x6c00,0x4620],[0x4460,0x2e0,0x6220,0x740],[0x2260,0xe20,0x6440,0x4700],[0x2620,0x720,0x2320,0x2700]];
var keycom={"38":"rotate(1)","40":"down()","37":"move(2,1)","39":"move(0.5,-1)"};
var dia, pos, bak, run;
function start(){
    dia=tatris[~~(Math.random()*7)];
    bak=pos={fk:[],y:0,x:4,s:~~(Math.random()*4)};
    rotate(0);
}
function over(){
    document.onkeydown=null;
    clearInterval(run);
```

3.算法设计的要求

- 健壮性
  - 当输入数据不合法时，算法也能做出相关处理，而不是产生异常或莫名其妙的结果
    - 输入的时间或者距离不应该是负数等

3.算法设计的要求

- 时间效率高和存储量低
  - 花最少的钱，用最短的时间，办最大的事儿

一个好的算法

正确性	可读性
健壮性	时间效率高和存储量低

4.算法效率的度量方法

- 事后统计方法
  - 通过设计好的测试程序和数据，利用计算机计时器对不同算法编制的程序的运行时间进行比较，从而确定算法效率的高低。
- 缺点：
  - 必须事先编好程序
  - 时间依赖计算机硬件和软件等环境因素
  - 算法的测试数据设计困难
    - 排序，数据小/数据大

4.算法效率的度量方法

- 事前分析估算方法
  - 在计算机程序编制之前，依据统计方法对算法进行估算
  - 一个高级程序语言编写的程序在计算机上运行时所消耗的时间取决于下列因素：
    - 算法采用的策略、方法
    - 编译产生的代码质量（软件）
    - 问题的输入规模
    - 机器执行指令的速度（硬件）

4.算法效率的度量方法

- 事前分析估算方法
  - 算法采用的策略、方法
  - 编译产生的代码质量（软件）
  - 问题的输入规模
  - 机器执行指令

```
int i, sum = 0, n = 100;      /*执行了1次*/
for ( i = 1; i <= n; i++)     次*/
{
    sum = sum + i;
}
printf ( “ %d ”, sum );       /*执行了1次*/
```

2n+3

在分析程序的运行时间时，最重要的是把程序看成是独立于程序设计语言的算法或一系列步骤。

4.算法效率的度量方法

- 算法分析——时间复杂度分析

算法的执行时间 = 每条语句执行时间之和

每条语句执行次数之和

基本语句的执行次数

执行次数×执行一次的时间

单位时间

指令系统、编译的代码质量

5.算法的时间复杂度

- 算法的时间复杂度定义
  - 在进行算法分析时，语句总的执行次数T(n)是关于问题规模n的函数，进而分析T(n)随n的变化情况并确定T(n)的数量级。记作：T(n)=O(f(n))

```
int i, sum = 0, n = 100;      /*执行了1次*/
for ( i = 1; i <= n; i++)     /*执行了n+1次*/
{
    sum = sum + i;            /*执行了n次*/
}
printf ( " %d ", sum );       /*执行了1次*/
```

O(n) 线性阶

```
int i, sum = 0, n = 100;      /*执行了1次*/
sum = ( 1 + n ) *n /2;        /*执行了1次*/
printf ( " %d ", sum );       /*执行了1次*/
```

O(1)常数阶

5.算法的时间复杂度

- 算法分析----最好情况、最坏情况、平均情况
  - 例：在一维整型数组A[n]中顺序查找与给定值k相等的元素（假设该数组中有且仅有一个元素值为k）。

```
int Find(int A[ ], int k, int n)
{
    for (i=0; i<n; i++)
        if (A[i]= =k) break;
    return i;
}
```

  - 基本语句的执行次数是否只与问题规模有关？
  - 结论：如果问题规模相同，时间代价与输入数据（的分布）有关，则需要分析最好情况、最坏情况、平均情况

5.算法的时间复杂度

- 常见的时间复杂度

执行次数函数	阶	非正式术语
12	O(1)	常数阶
2n+3	O(n)	线性阶
$3n^2+2n+1$	$O(n^2)$	平方阶
$5\log_2 n+20$	O(logn)	对数阶
$2n+3n\log_2 n+19$	O(nlogn)	nlogn阶
$6n^3+2n^2+3n+4$	$O(n^3)$	立方阶
2^n	$O(2^n)$	指数阶

常见阶的比较：

$$O(1) < O(\log_2 n) < O(n) < O(n\log_2 n) < O(n^2) < O(n^3) < \ldots < O(2^n) < O(n!)$$

5.算法的时间复杂度

- 算法分析----时间复杂性分析的基本方法
- 时间复杂性的运算法则

  设T1(n)=O(f(n))，T2(n)=O(g(n))，则
  - ①加法规则:T1(n)+T2(n)= O(max{ f(n), g(n)})
  - ②乘法规则:T1(n)*T2(n) = O(f(n) · g(n))

- 时间复杂性的分析方法
  - 首先求出程序中各语句、各模块的运行时间,
  - 再求整个程序的运行时间。
  - 各种语句和模块分析应遵循的规则是：

5.算法的时间复杂度

- 算法分析----各种语句和模块分析应遵循的规则
  - (1)赋值语句或读/写语句:
    - 运行时间通常取O(1) .有函数调用的除外，此时要考虑函数的执行时间。
  - (2)语句序列:
    - 运行时间由加法规则确定，即该序列中耗时最多的语句的运行时间。
  - (3)分支语句：
    - 运行时间由条件测试（通常为O(1) ）加上分支中运行时间最长的语句的运行时间

5.算法的时间复杂度

- (4)循环语句:
  - 运行时间是对输入数据重复执行n次循环体所耗时间的总和
  - 每次重复所耗时间包括两部分：一是循环体本身的运行时间；二是计算循环参数、测试循环终止条件和跳回循环头所耗时间。后一部分通常为O(1)。
  - 通常，将常数因子忽略不计，可以认为上述时间是循环重复次数n和m的乘积，其中m是n次执行循环体当中时间消耗最多的那一次的运行时间(乘法规则)
  - 当遇到多重循环时，要由内层循环向外层逐层分析。因此，当分析外层循环的运行时间是，内层循环的运行时间应该是已知的。此时，可以把内层循环看成是外层循环的循环体的一部分。

5.算法的时间复杂度

- (5)函数调用语句:
  - ①若程序中只有非递归调用，则从没有函数调用的被调函数开始，计算所有这种函数的运行时间。然后考虑有函数调用的任意一个函数P，在P调用的全部函数的运行时间都计算完之后，即可开始计算P的运行时间
  - ②若程序中有递归调用，则令每个递归函数对应于一个未知的时间开销函数T(n)，其中n是该函数参数的大小，之后列出关于T的递归方程并求解之。

6.算法空间复杂度

- 空间换取时间
- 思考
  - 判断某年是不是闰年
    - 算法，每次给一个年份，进行运算
  - 建立2050个元素的数组
    - 闰年1，非闰年0
    - 变为查找该数据的某一项的值是多少
  - 时间变短，但是空间变大
- 一般而言，“复杂度”指时间复杂度

时空资源的折中原理

- 同一个问题求解，一般会存在多种算法，这些算法在时空开销上的优劣往往表现出“时空折中”（trade-off）的性质
  - 即，为了改善一个算法的时间开销，往往以增大空间开销为代价，而设计出一个新算法来
  - 有时，为了缩小算法的空间开销，也可以牺牲计算机的运行时间，通过增大时间开销来换取存储空间的节省

6.算法空间复杂度

- 空间复杂度(Space Complexity)是对一个算法在运行过程中临时占用存储空间大小的量度。
- 一个算法在计算机存储器上所占用的存储空间，包括存储算法本身所占用的存储空间，算法的输入输出数据所占用的存储空间和算法在运行过程中临时占用的存储空间这三个方面。
- 算法的空间复杂性是指算法在执行过程中的最大存储量需求
- 空间复杂性的渐近表示----空间复杂度

  S（n）= O（ f（n）） 其中，n为问题的输入规模

实例

- 算法分析--例:分析下述“冒泡”排序程序的时间复杂性。

```
    void BubbleSort( int A[], int n )
    {   int i, j, temp;
(1)     for (i=0; i<n-1; i++)
(2)         for (j=n-1; j>=i+1; j--)
(3)             if (A[j-1]>A[j]) {
(4)                 temp=A[j-1];
(5)                 A[j-1]=A[j];
(6)                 A[j]=temp;
                }
    }
```

•时间复杂度：

$$O(\sum_{i=0}^{n-2}(n-i-1)) \le O(n(n-1)/2) = O(n^2)$$

• 空间复杂度：

O(1)

几点课程说明

- 编程语言的使用
  - C、C++
- 教材
- 习题
- 助教（3人）

四、算法设计题：（共 25 分）

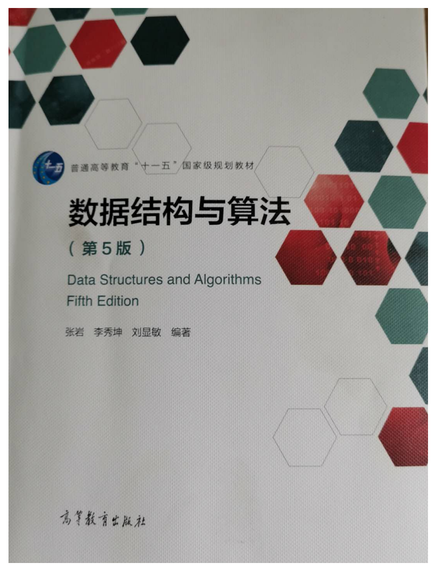
按以下要求设计算法：

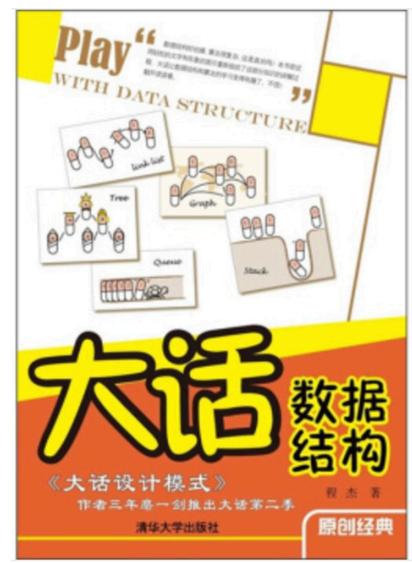
（1）给出算法的基本设计思想。

（2）使用 C 或 C++或 Java 语言，给出相关的数据类型定义。

（3）根据设计思想，采用 C 或 C++或 Java 语言描述算法，关键之处给出注释。

（4）说明你所设计算法的时间复杂度。





例:编写求n!的程序，并分析其时间复杂性。

- 求n!的递归算法

```
long  fact ( int  n)
{   if ( n==0 ) || ( n ==1 )
          return( 1 );
     else
          return( n * fact(n – 1));
}
```

- 时间复杂性的递归方程

$$T(n) = \begin{cases} C & \text{当 } n=0,\ n=1 \\ G + T(n-1) & \text{当 } n > 1 \end{cases}$$

- 空间复杂度：O（n）

- 解递归方程:

$$\begin{aligned} T(n) &= G + T(n-1) \\ T(n-1) &= G + T(n-2) \\ T(n-2) &= G + T(n-3) \\ &\cdots\ \cdots \\ T(2) &= G + T(1) \\ +\quad T(1) &= C \\ \hline T(n) &= G(n-1) + C \end{aligned}$$

$\Downarrow$

取 $f(n) = n$

$\therefore T(n) = O(f(n))$

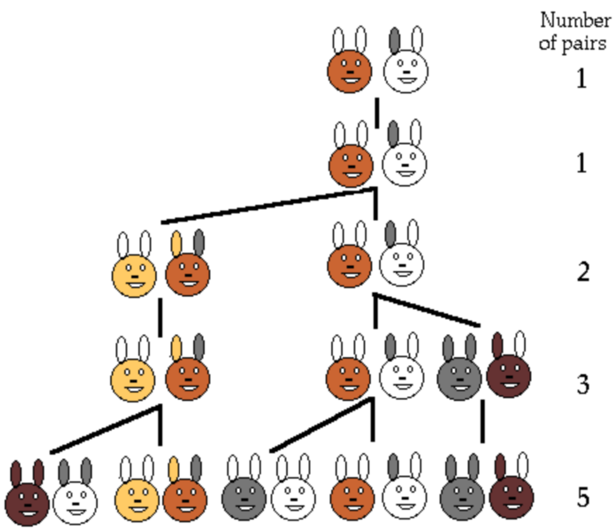
$= O(n)$

思考题

https://blog.csdn.net/lxf_style/article/details/80458519

求这两种算法的时间复杂度和空间复杂度

- 斐波那契数列： 1、1、2、3、5、8、13、21、34



$$F(n)=\begin{cases}0, & 当n=0\\ 1, & 当n=1\\ F(n-1)+F(n-2), & 当n>1\end{cases}$$

两种常见算法：

- 递归
- 循环

方法一：递归

```c
#include<stdio.h>
#include<stdlib.h>
long long Fib(long long N)
{
  if (N < 3)   //当N<3时，斐波那契数为1
  return 1;
  return Fib(N - 1) + Fib(N - 2);//函数递归
}
int main()
{
  int n = 0;
   scanf("%d",&n);
   printf("%d",Fib(n));
   system("pause");
   return 0;
```

方法二：循环

```c
#include<stdio.h>
#include<stdlib.h>
long Fib(long N)
{
  int result = 0;    //前两个数之和
  int pre_result = 1;  //前一个数
  int next_older_result = 1; //前前一个数-_-!
   result = pre_result;
   while (N > 2)
   {
    N--;
     next_older_result = pre_result;
     pre_result = result;
     result = pre_result + next_older_result;//结果为前两个数
    }
   return result;
}
int main()
{
    int n = 0;
    scanf("%d",&n);
    printf("%d",Fib(n));
    system("pause");
   return;
```

算法的分类

- 算法设计与算法分析是计算机科学的核心问题
- 常用的设计方法
  - 穷举法(百钱买百鸡)
  - 贪心法(Huffman树、Prim等)
  - 递归法, 分治法(二分检索、快速排序等)
  - 回溯法(树、图等的深度优先搜索)
  - 动态规划法(最佳二叉排序树)
  - α-β裁剪和分枝界限法
  - 并行算法

穷举法

- 百钱买百鸡
  - 中国古代算书《张丘建算经》中有一道著名的百鸡问题：公鸡每只值5 文钱，母鸡每只值3 文钱，而3 只小鸡值1 文钱。用100 文钱买100 只鸡，问：这100 只鸡中，公鸡、母鸡和小鸡各有多少只？

设公鸡、母鸡、小鸡分别为x、y、z 只，由题意得：

① $x + y + z = 100$

② $5x + 3y + (1/3)z = 100$

作业要求：

1. 提交程序
2. 思考时间复杂度，以注释形式写在程序里

时间：下周一上课前交

小结

- 1. 算法定义
- 2. 算法的特性
- 3. 算法设计的要求
- 4. 算法效率的度量方法
- 5. 算法的时间复杂度
- 6. 算法的空间复杂度