理解数据结构

程序和算法

性能分析与度量

什么是数据结构？

- 数据：数据是信息的载体，是描述客观事物的数、字符、以及所有能输入到计算机中并被计算机程序识别和处理的符号的集合。

  ——数值数据, 非数值性数据

- 数据对象：数据的子集。具有相同性质的数据成员（数据元素）的集合。

  ——整数数据对象 N = { 0, 1, 2, … }

  ——学生数据对象

什么是数据结构？

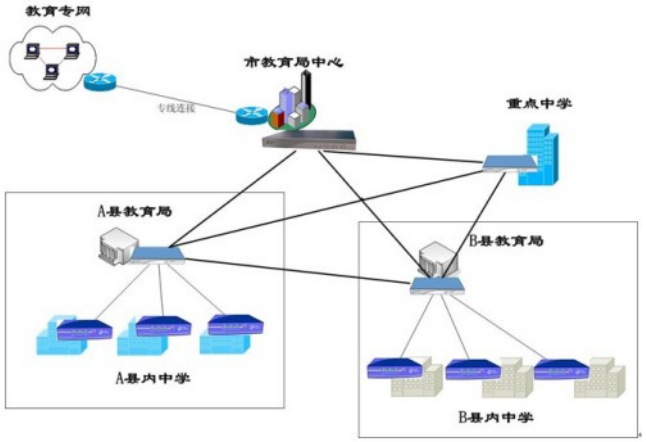
定义：　由某一数据对象及该对象中所有数据成员之间的关系组成。记为：

Data_Structure = {D, R}

其中，D是某一数据对象，R是该对象中所有数据成员之间的关系的有限集合

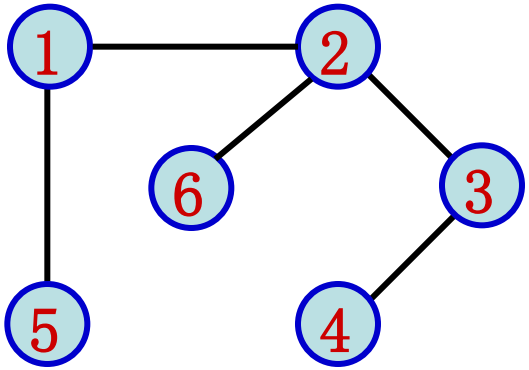
例子

- N个网站之间的连接关系
- 复数的组成

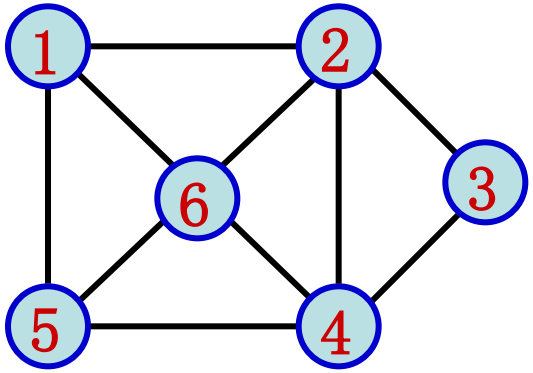


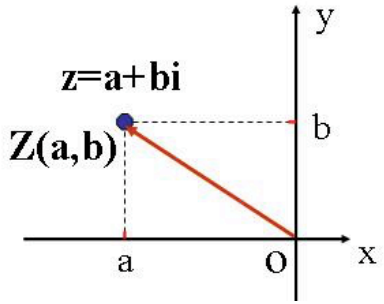
- *N*个网站之间的连通关系

树形关系



网状关系





- 复数的数据结构定义如下：

Complex=(C，R)

C是包含两个实数的集合｛C1，C2}

R={P}，P是定义在集合上的一种关系{〈C1，C2〉}

数据结构是数据的组织形式

- 包括三个方面：
  - 数据元素间的逻辑关系，即数据的逻辑结构；
  - 数据元素及其关系在计算机存储内的表示，即数据的存储表示；
  - 数据的运算，即对数据元素施加的操作。

相关：

- 逻辑结构
- 物理结构
- 相关操作
- 实现

数据的逻辑结构

- 从逻辑关系上描述数据，与数据的存储无关；
- 看作是从具体问题抽象出来的数据模型；
- 与数据元素本身的形式、内容无关；
- 与数据元素的相对存储位置无关。

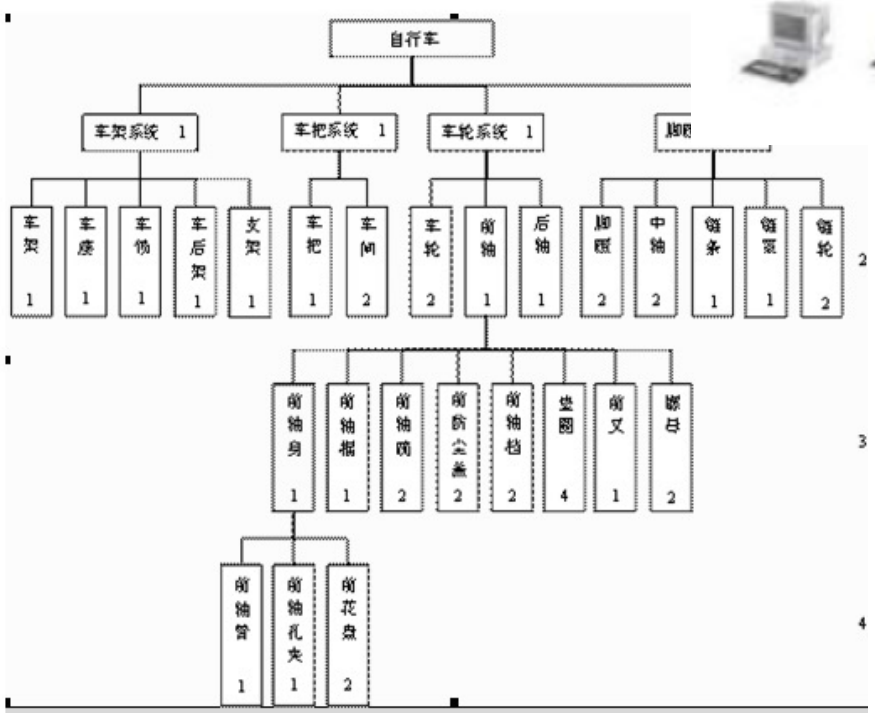
数据的逻辑结构分类

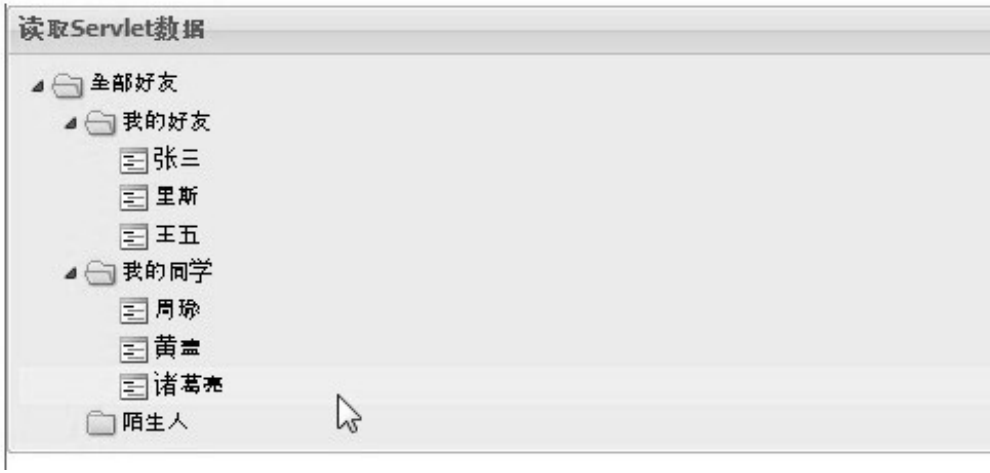
- 线性结构
  - 线性表
- 非线性结构
  - 树
  - 图（或网络）

线性结构

树型结构

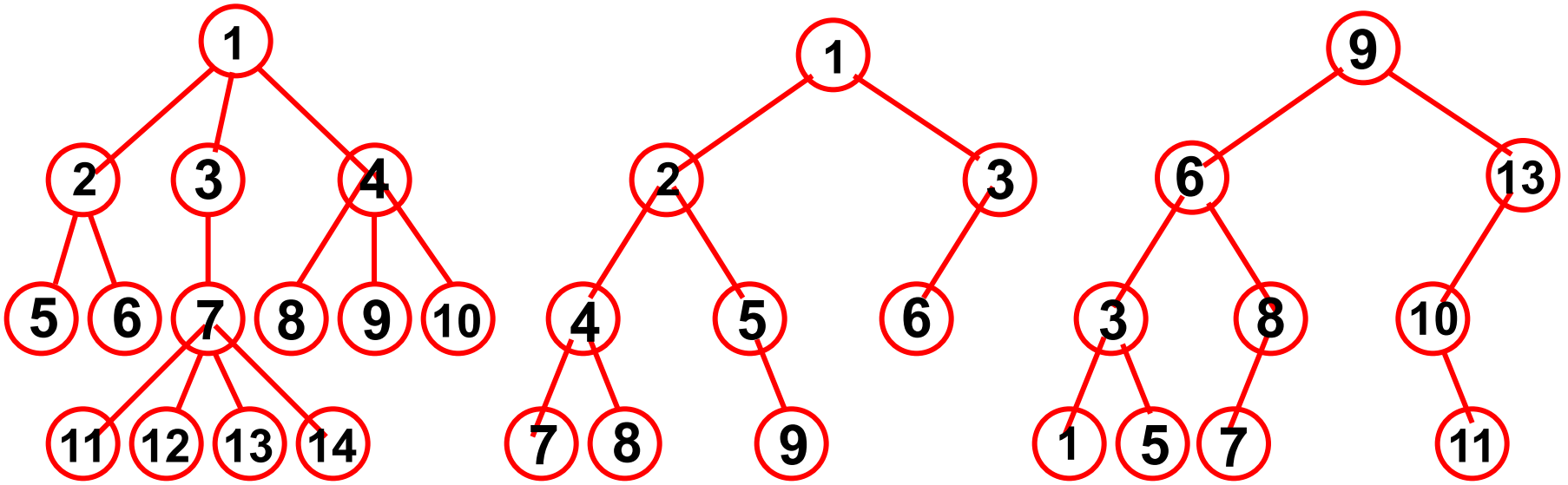

核心交换机
服务器
服务器
汇聚层
交换机
边缘层
交换机
工作站


自行车
车架系统 1
车把系统 1
车轮系统 1
车架 1
车座 1
车铃 1
车后架 1
支架 1
车把 1
车闸 2
车轮 2
前轴 1
后轴 1
脚踏 2
中轴 2
链条 1
链罩 1
链轮 2
前轴身 1
前轴握 1
前轴碗 2
前防尘盖 2
前轴档 2
垫圈 4
前叉 1
螺母 2
前轴管 1
前轴孔夹 1
前花盘 2

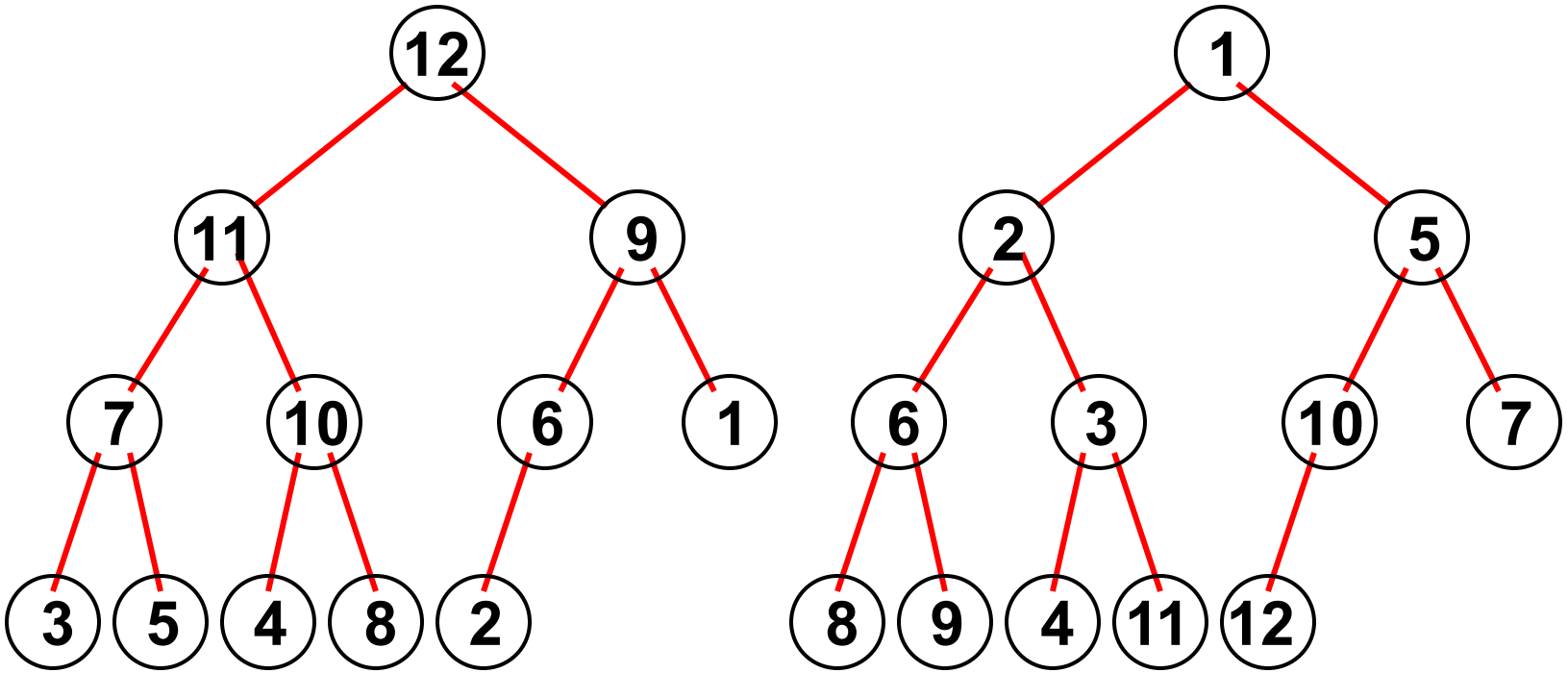

读取Servlet数据
全部好友
我的好友
张三
里斯
王五
我的同学
周瑜
黄盖
诸葛亮
陌生人

树型结构

树　　二叉树　　二叉搜索树

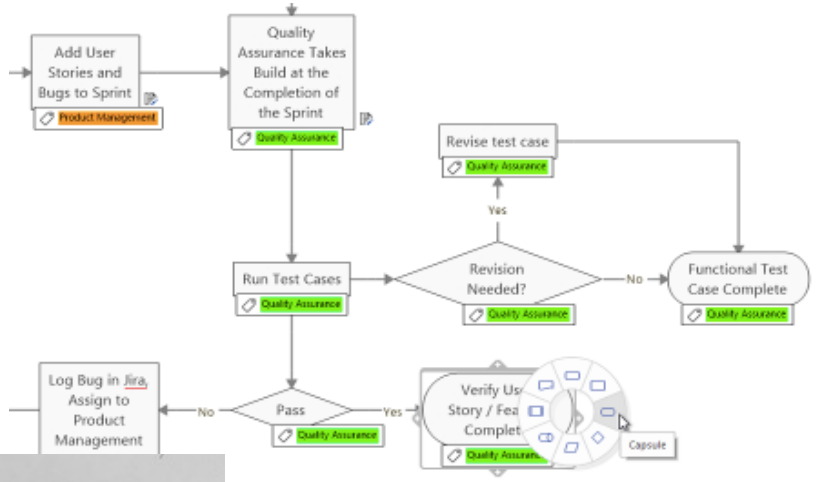


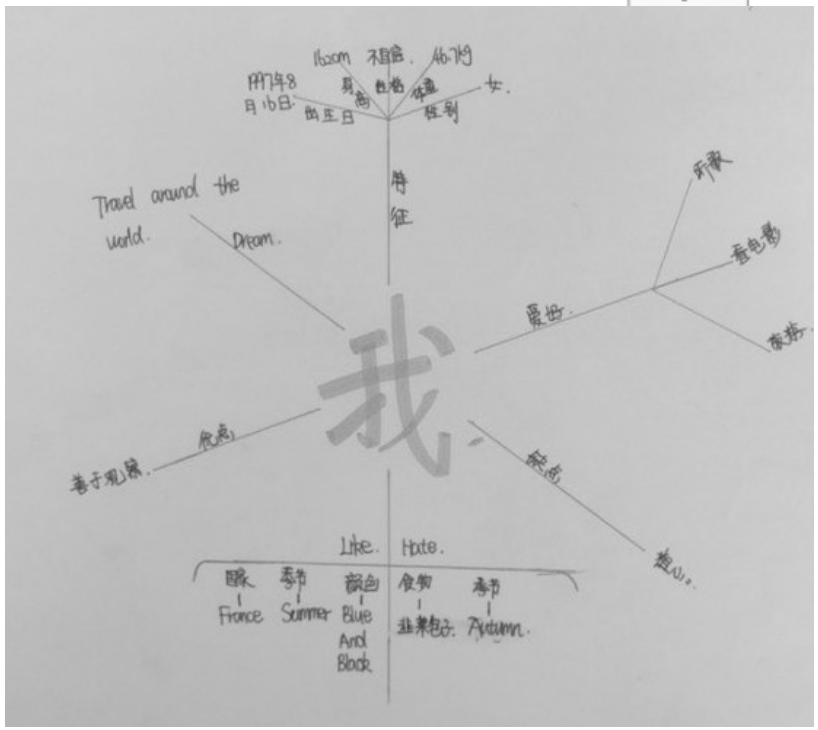
堆结构

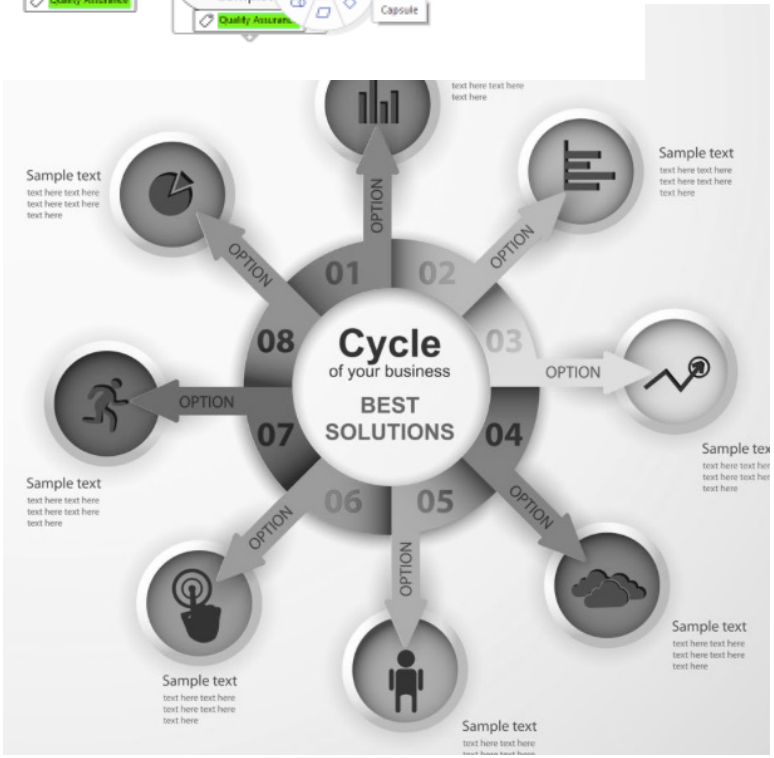


“最大”堆　　“最小”堆

图结构

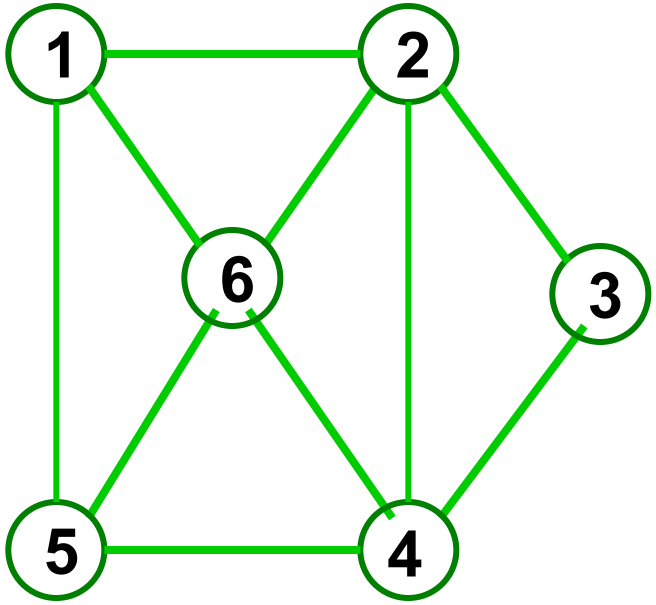

Add User Stories and Bugs to Sprint
Product Management
Quality Assurance Takes Build at the Completion of the Sprint
Quality Assurance
Revise test case
Quality Assurance
Yes
Run Test Cases
Quality Assurance
Revision Needed?
Quality Assurance
No
Functional Test Case Complete
Quality Assurance
Log Bug in Jira, Assign to Product Management
No
Pass
Quality Assurance
Yes
Capsule


我
Travel around the world.
Dream.
Like.
Hate.
France
Summer
Blue And Black
Autumn.

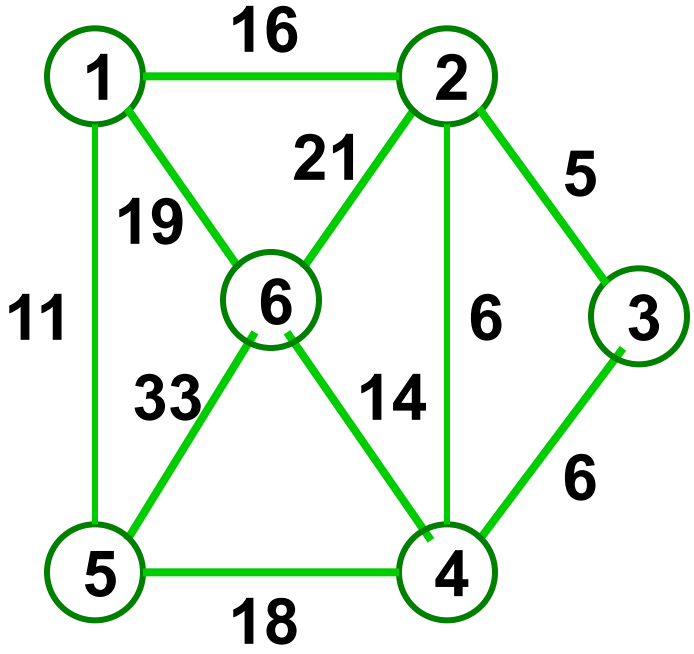

Cycle
of your business
BEST SOLUTIONS
01
02
03
04
05
06
07
08
OPTION
Sample text
text here text here text here text here

图结构

图结构



网络结构

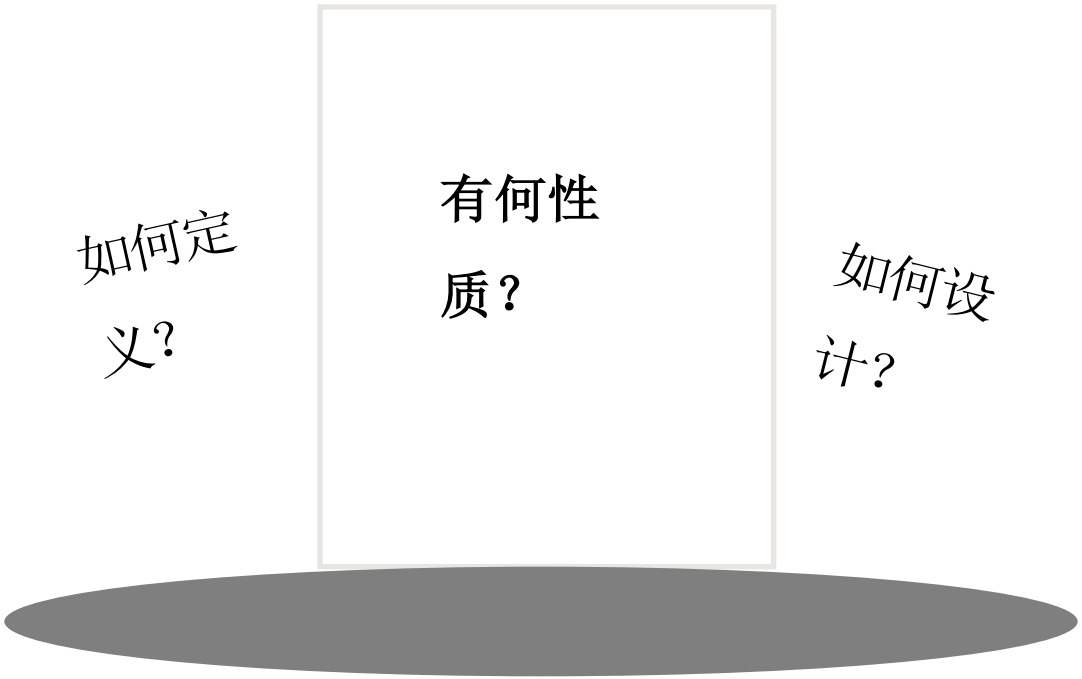


- 数据的存储结构是逻辑结构用计算机语言的实现；
- 数据的存储结构依赖于计算机语言。
  - 顺序存储表示
  - 链接存储表示
  - 索引存储表示

  } 主要用于内存的存储表示

  - 散列存储表示

  } 主要用于外存（文件）的存储表示


如何定义?
有何性质?
如何设计?

- 定义： 是对特定问题求解步骤的一种描述，算法是指令的有限序列，其中每一条指令表示一个或多个操作。
- 特性：
  - 输入
  - 输出
  - 确定性
  - 有穷性
  - 有效性

算法设计： 自顶向下，逐步求精

例：选择排序问题

算法框架：

初始值a[k]

```
for ( int i = 0; i < n-1; i++ ) {     //n-1趟
        从a[i]检查到a[n-1];
    若最小的整数在a[i]，交换a[i]与a[k];
}
```

- 细化程序：程序 SelectSort

```
void selectSort ( int a[ ], const int n ) {
//对n个整数a[0],a[1],…,a[n-1], 按非递减顺序排序
  for ( int i = 0; i < n-1; i++ ) {
        int k = i;
        //从a[i]检查到a[n-1], 找最小的整数, 在a[k]
        for ( int j = i+1; j < n; j++ )
           if ( a[j] < a[k] ) k = j;
              //k指示当前找到的最小整数
    int temp = a[i];  a[i] = a[k];   a[k] = temp;
              //交换a[i]与a[k]
     }
}
```

- 正确性

四个层次：

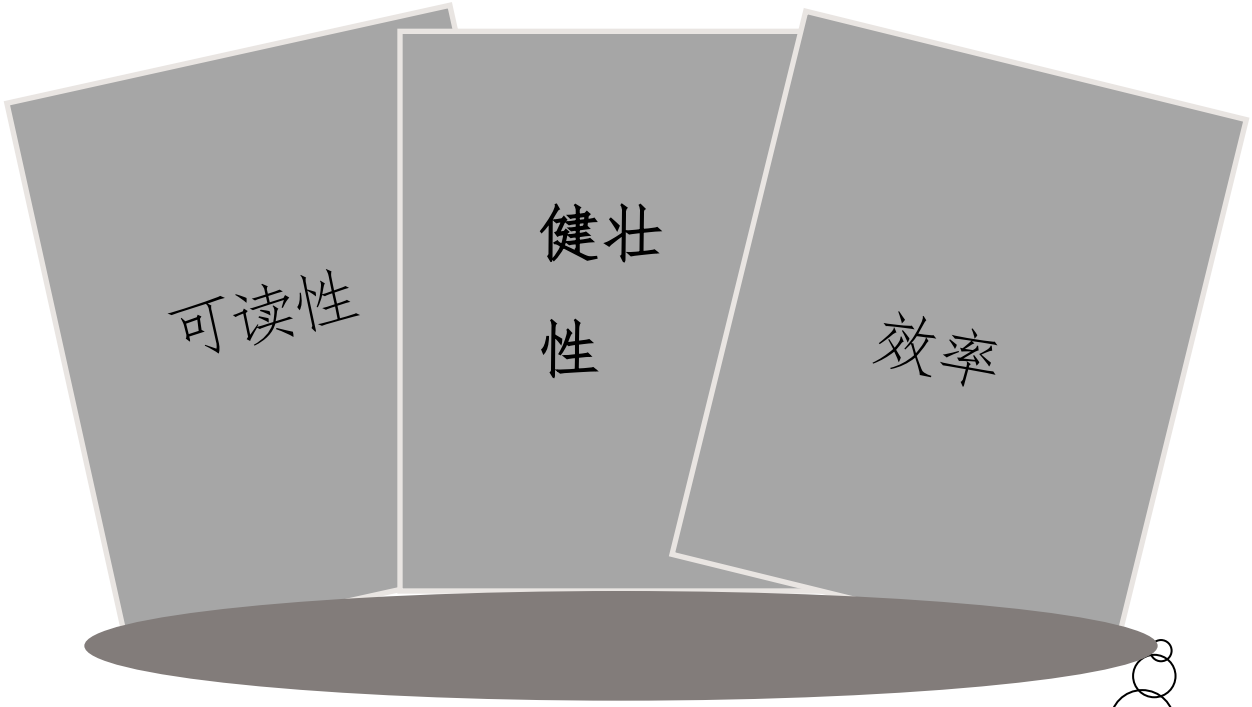
–程序不含语法错误；

–程序对于几组输入数据能够得出满足规格说明要求的结果；

–程序对于精心选择的典型、苛刻而带有刁难性的几组输入数据能够得出满足规格说明要求的结果；

–程序对于一切合法的输入数据都能产生满足规格说明要求的结果。

算法...



时间代价

空间代价

算法效率的度量...

- 后期测试
- 事前估计

—— 空间复杂性度量

—— 时间复杂性度量

算法的空间复杂度度量...

我们对空间复杂度感兴趣的原因主要是：

- 如果一个程序要运行在一个多用户计算机系统中我们需要指明该程序所需内存的大小。
- 运行程序，是否有足够的内存。
- 若干种解决方法，对内存的需求不同。
- 估算一个程序所解决的问题是什么规模的。

例1 求表达式a+b+b*c+(a+b-c)/(a+b)+4.0

```
float abc (float a, float b,  float c ) {
return a+b+b*c+(a+b-c)/(a+b)+4.0
}
```

例2 以迭代方式求累加和

```
float sum ( float a[ ], const int n )
    {
      float s = 0.0;
      for ( int i = 0; i < n; i++ )
          s += a[i];
      return s;
    }
```

```
float rsum(float a[ ],const int n) {
  if (n<=0) return 0;
  else return rsum(a,n-1)+a[n-1]
)
```

空间复杂度是指当问题的规模以某种单位从1增加到n时，解决这个问题的算法在执行时所占用的存储空间也以某种单位由1增加到S（n）

空间复杂度的组成

程序所需的空间

1、指令空间

- 编译之后程序指令所需的存储空间

2、数据空间

- 常量
- 动态空间

3、环境栈空间

- 调用和恢复函数或方法时所需要的空间

运行时间

估算运行时间

- 程序步

  语法上或语义上有意义的一段指令序列

  执行时间与实例特性无关

例如：

- 注释：程序步数为0
- 声明语句：程序步数为0
- 表达式：程序步数为1， return a+b+b*c
- 赋值语句 <变量>=<表达式>

- 循环语句（循环控制部分，执行语句）

for(<初始化语句>;<表达式1>;<表达式2>)…
while <表达式> do…
do …while <表达式>

- **switch 语句**
- **if then**
- 函数执行语句：
- 动态存储管理：**new delete sizeof( )**
- 转移语句：**continue,break,goto,return**

例 以迭代方式求累加和的函数

```
行   float sum ( float a[ ], const int n )
1    {
2       float s = 0.0;
3       for ( int i = 0; i < n; i++ )
4           s += a[i];
5       return s;
6    }
```

```
float sum ( float a[ ], const int n ) {
    float s = 0.0;
    count+2;  //count统计执行语句条数
    for ( int i = 0; i < n; i++ ) {
        count+2;      //针对for语句
      s += a[i];
      count++;
    }       //针对赋值语句
    count+2;  //针对for的最后一次
    count++; //针对return语句
    return s;
```

注意：

一个语句本身的程序步数可能不等于该语句一次执行所具有的程序步数。

例如：赋值语句

x = sum (R, n);

—建表，列出逐个语句的程序步

例：计算累加和程序

程 序 语 句	一次执行所需程序步数	执行频度	程序步数
{	0	1	0
float s = 0.0;	1	1	1
for (**int** i=0; i<n; i++)	2	n+1	2n+2
s += a[i];	1	n	n
return s;	1	1	1
}	0	1	0
	总程序步数		3n+4

```
void selectSort ( int a[ ], const int n ) {
//对n个整数a[0],a[1],…,a[n-1], 按非递减顺序排序
    for ( int i = 0; i < n-1; i++ ) {
        int k = i;
        //从a[i]检查到a[n-1], 找最小的整数, 在a[k]
        for ( int j = i+1; j < n; j++ )
            if ( a[j] < a[k] ) k = j;
                //k指示当前找到的最小整数
        int temp = a[i];  a[i] = a[k];   a[k] = temp;
                //交换a[i]与a[k]
    }
}
```

时间复杂度的渐进表示法

描述最坏情况下的时间代价

定义：如果存在正整数c和n_0, 使得当n>= n_0 时， T(n)<=cf(n), 则记为T(n)=O(f(n))

例: (线性函数)

考察$f(n)=3n+2$

例: 平方函数

考察$f(n)=10n^2+4n+2$

例：设有两个算法在同一机器上运行，其执行时间分别为 $100n^2$ 和 2^n，问：要使前者快于后者，n 至少要取多大？

解答：

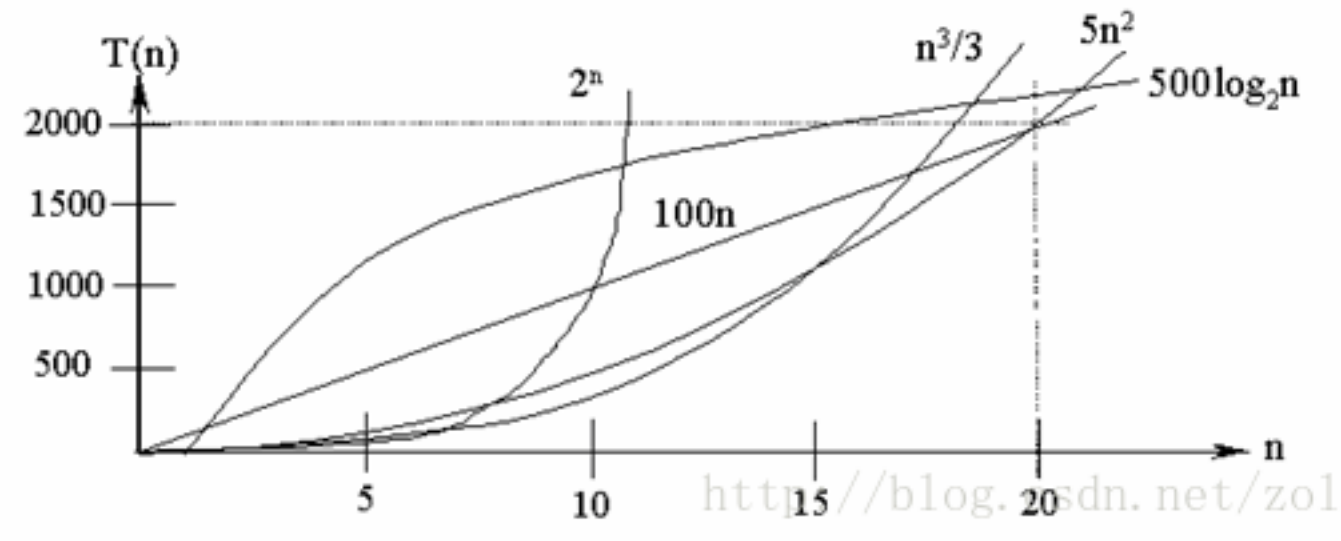
问题是找出满足 $100n^2 <= 2^n$ 的最小的n。用试探法：

$n = 13$时，$100n^2 = 16900 > 2^n = 8192$

$n = 14$时，$100n^2 = 19600 > 2^n = 16384$

$n = 15$时，$100n^2 = 22500 < 2^n = 32764$

取 $n = 15$ 满足要求。



大O表示法:

$$c < \log_2 n < n < n\log_2 n < n^2 < n^3 < 2^n < 3^n < n!$$

使用大（O）表示法：

对于多项式，我们只保留最高次幂的项，常数系数和低阶项可以不要。

$\log_2 n \longrightarrow O(\log_2 n)$

加法法则，针对并行程序

$$T(n, m) = T1(n) + T2(m)$$

$$= O(\max(f(n), g(m)))$$

$$c < \log_2 n < n < n\log_2 n < n^2 < n^3 < 2^n < 3^n < n!$$

乘法规则 针对嵌套程序段

$$T(n, m) = T1(n) * T2(m)$$
$$= O(f(n) * g(m))$$

特例： 若T1 (n)=O(c), c与n无关的任意常数， T2 (n)=O(f(n)), 则T (n)=O(f(n))

任何非0正常数都属于同一数量级，记为O(1)

```
void example (float x[ ][ ], int m, int n, int k) {
    float sum [ ];
    for ( int i = 0; i < m; i++ ) {
        sum[i] = 0.0;
        for ( int j=0; j<n; j++ )
            sum[i] += x[i][j];
    }
    for ( i = 0; i < m; i++ )
        cout << "Line" << i <<
                " : " <<sum [i] << endl;
}
```

```
template <class Type>
 void dataList<Type>::bubbleSort ( ) {
int i = 1;  int exchange = 1;
        //当exchange为0则停止排序
      while ( i < ArraySize && exchange ) {
         BubbleExchange ( i, exchange );
         i++;
    }  //一趟比较
}
```

```
template <class Type>
void dataList<Type>::
BubbleExchange(const int i, int & exchange ){
    exchange = 0;                    //假定元素未交换
    for ( int j = ArraySize-1; j>=i; j--)
        if ( Element[j-1] > Element[j] ) {
            Swap ( j -1, j );   //发生逆序
                  //交换Element[j-1]与Element[j]
            exchange = 1;    //做“发生了交换”标志
        }
}
```

BubbleSort **n-1趟**

BubbleExchange ( )
n-i次比较

$$\because \sum_{i=1}^{n-1}(n-i)=\frac{n(n-1)}{2}$$

渐进时间复杂度

$$O(f(n)*g(n)) = O(n^2)$$

渐进的空间复杂度

$$S(n) = O(f(n))$$

复习

1、数据抽象与封装

- *抽象*
- *封装*
- *面向对象程序设计*

2、类

- *类*
- *数据成员*
- *成员函数*

3、对象

- *对象的创建和标识*
- *对象的初始化*

讨论环节

Google面试题

有一个random number generator，是生成真实的随机数，而不是伪随机数，会生成几千亿个32位整数，打印出现次数100的整数。