EDU
CSDN学院 IT实战派

图解数据结构和算法

数据结构和算法初体验

讲师：Samuel

课程概述

1 数据结构和算法的意义

- 数据结构是所有的计算机领域研究绕不去的一个话题
- 数据结构和算法是进入大厂的面试门槛
- 数据结构是算法的基础，数据结构+算法=程序
- 现实世界中，数据结构和算法也是无处不在的

课程概述

2 课程内容设置

数组

栈和队列

链表

二叉搜索树

堆

哈希表

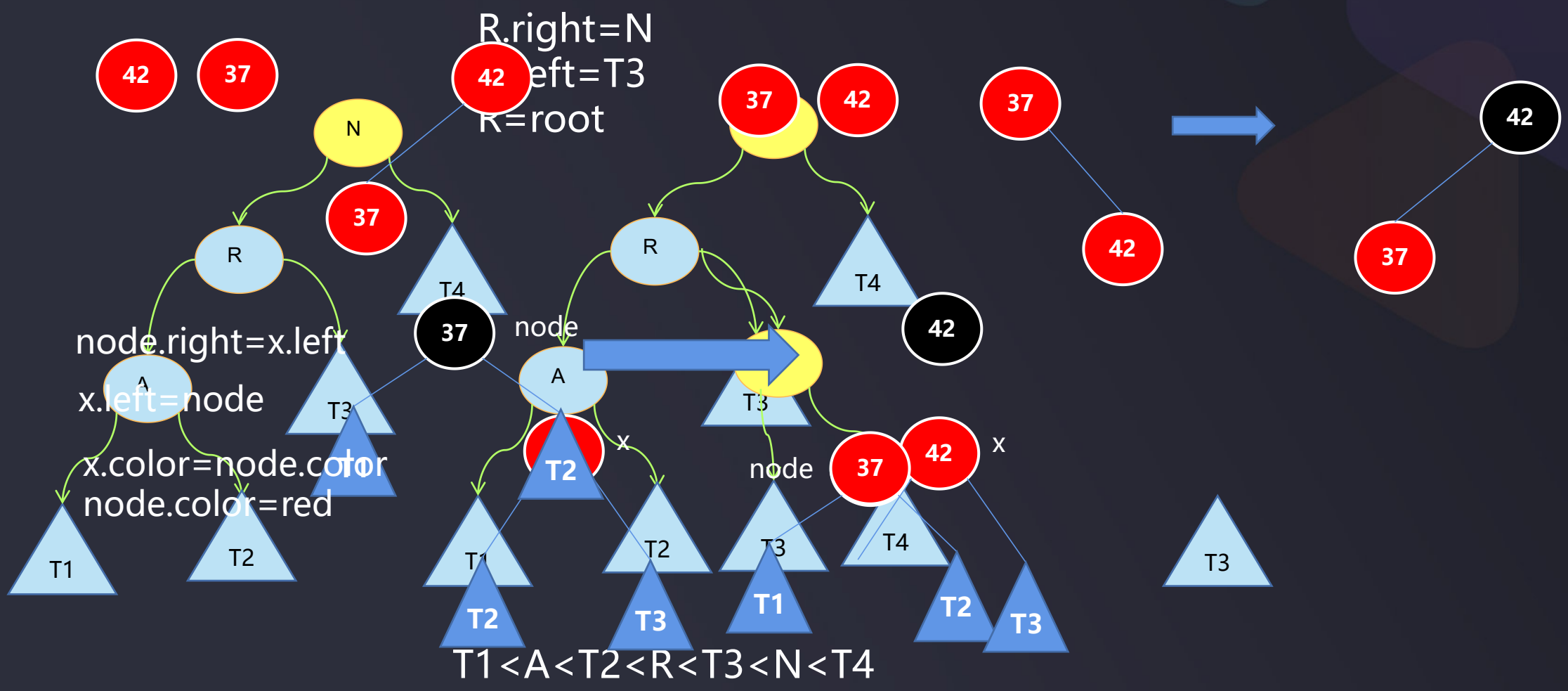
红黑树

AVL树

图

课程概述

3 课程特点及答疑



Content

1. 初识数据结构和算法
2. 基本概念
3. 时间复杂度和空间复杂度

数据结构和算法的魅力

1. ArrayList和LinkedList

ArrayList底层是数组，查询快，插入慢，有移动的动作。

LinkedList 底层链表，插入快 查询慢

2. -1^n问题

编程求(-1)^0+ (-1)^1+ (-1)^2+.....+ (-1)^n

思路一：循环遍历求sum

思路二：当n为奇数时sum=-1,当n为偶数时sum=0

3. 1到100求和问题

编程求1+2+3+.....+100

思路一：循环遍历求sum

思路二：(1+n)*n/2

基本概念

1 数据结构(data structure)

- 数据结构是相互之间存在一种或多种特定关系的数据元素的集合
- 研究的是数据的逻辑结构和数据的物理结构以及它们之间的相互关系
- 数据结构包括：线性结构和非线性结构

2 算法（Algorithm）

- 是解题方案的准确而完整的描述，是一系列解决问题的清晰指令
- 算法代表着用系统的方法描述解决问题的策略机制
- 一个算法的优劣可以用空间复杂度与时间复杂度来衡量

基本概念

1 度量算法的执行时间

- 度量一个程序(算法)执行时间的两种方法
  - 事后统计法
  - 事前估算法
- 统计某个算法的时间复杂度来度量方法的优越性
- 时间复杂度统计法属于事前估算法

2 时间频度T(n)

- 时间频度：一个算法花费的时间与算法中语句的执行次数成正比
- 一个算法中的语句执行次数称为语句频度或时间频度，记为T(n)， n称为问题的规模

3 时间复杂度O(n)

- 某个函数f(n),使得当n趋近于无穷大时，T(n)/f(n)的极限值为不等于零的常数，则称f(n)是T(n)的同数量级函数。记作T(n)=O(f(n))
- T(n) 不同，但时间复杂度可能相同
- 称**O(f(n))** 为算法的渐进时间复杂度，简称时间复杂度

基本概念

4 最坏时间复杂度

- 最坏情况下的时间复杂度称最坏时间复杂度
- 一般讨论的时间复杂度均是最坏情况下的时间复杂度
- 最坏情况下的时间复杂度是算法在任何输入实例上运行时间的界限

5 平均时间复杂度

- 平均时间复杂度是指所有可能的输入实例均以等概率出现的情况下，该算法的运行时间
- 平均时间复杂度和最坏时间复杂度是否一致，和算法有关

基本概念

6 空间复杂度

- 一个算法的空间复杂度(Space Complexity)定义为该算法所耗费的存储空间，它也是问题规模n的函数
- 空间复杂度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
- 在做算法分析时，主要讨论的是时间复杂度
- 从用户使用体验上看，更看重的程序执行的速度
- 一些缓存产品(redis, memcache)和算法(基数排序)本质就是用空间换时间

时间复杂度

1 分析1+2+….+100时两种算法的时间频度

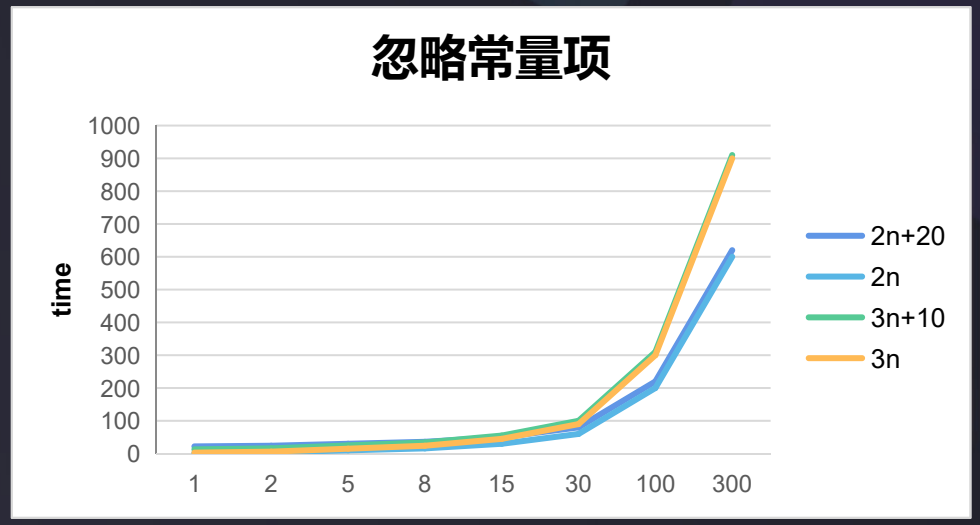
```
public static void main(String[] args)
{
    //算法一
    int sum = 0 ;
    int end = 100;
    for(int i=1;i<=end;i++){
        sum += i;
    }
    //算法二
    sum = (1+end)*end/2;
}
```

- 算法一的时间频度T(n) = n+1
- 算法二的时间频度T(n) =1
- 根据时间频度可以推算时间复杂度

计算时间复杂度原则

1 忽略常量项

	T(n)=2n+20	T(n)=2*n	T(3n+10)	T(3n)
1	22	2	13	3
2	24	4	16	6
5	30	10	25	15
8	36	16	34	24
15	50	30	55	45
30	80	60	100	90
100	220	200	310	300
300	620	600	910	900



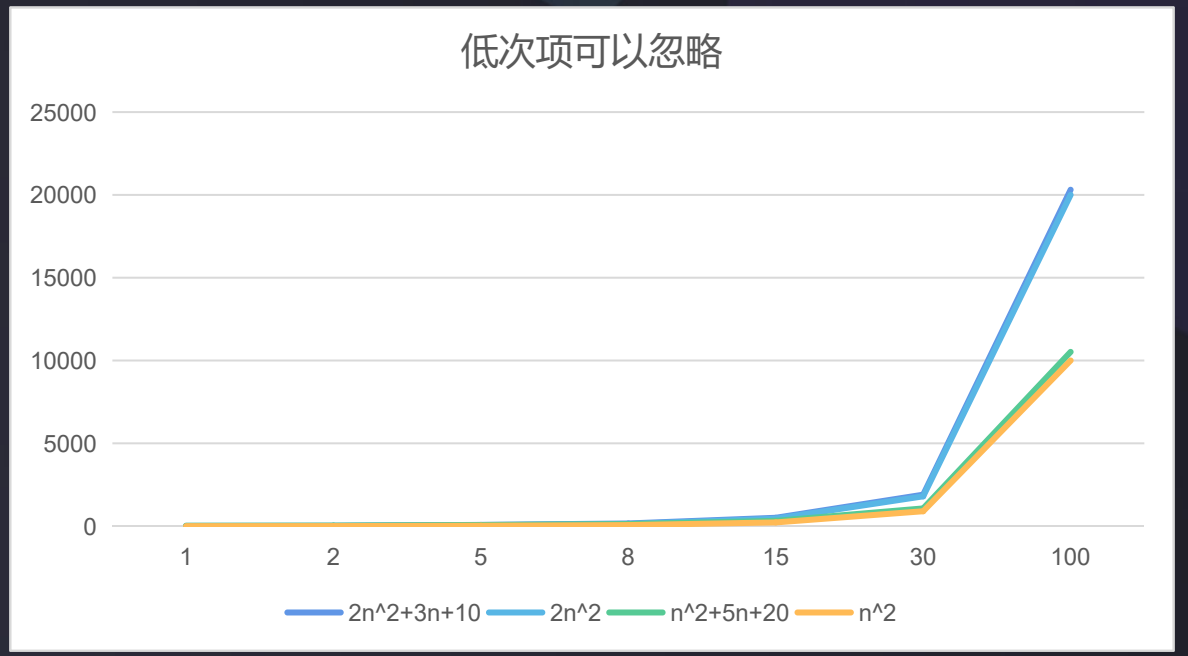
结论:

- 2n+20 和 2n 随着n 变大，执行曲线无限接近, 20可以忽略
- 3n+10 和 3n 随着n 变大，执行曲线无限接近, 10可以忽略

计算时间复杂度原则

2 忽略低次项

	T(n)=2n^2+3n+10	T(2n^2)	T(n^2+5n+20)	T(n^2)
1	15	2	26	1
2	24	8	34	4
5	75	50	70	25
8	162	128	124	64
15	505	450	320	225
30	1900	1800	1070	900
100	20310	20000	10520	10000



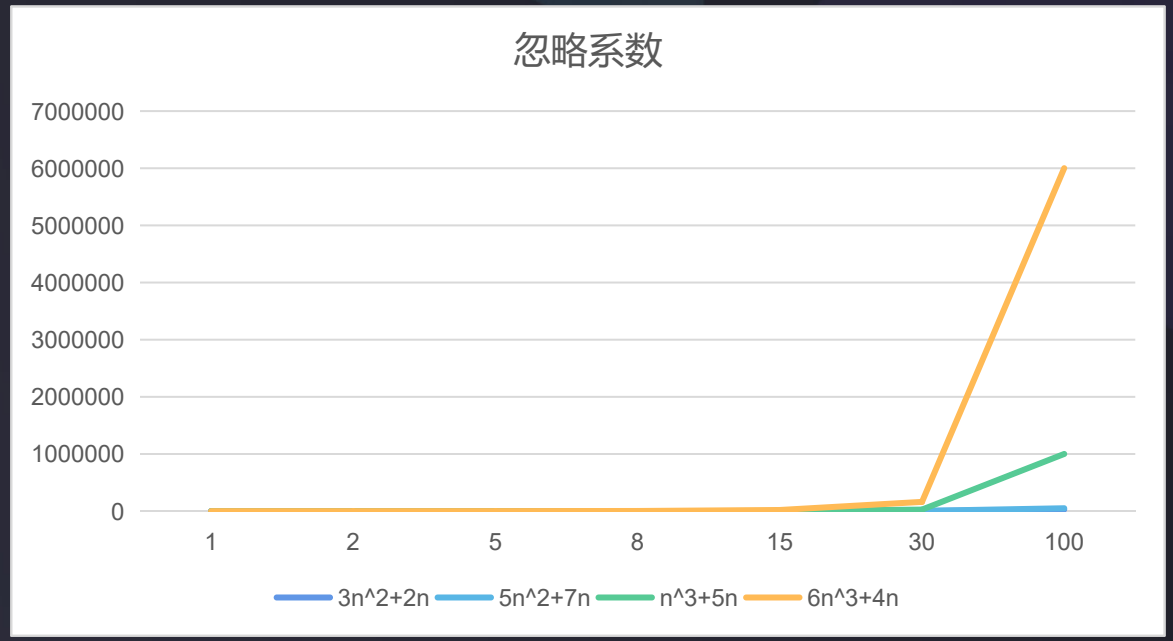
结论:

- 2n^2+3n+10 和 2n^2 随着n 变大, 执行曲线无限接近, 可以忽略 3n+10
- n^2+5n+20 和 n^2 随着n 变大,执行曲线无限接近, 可以忽略 5n+20

计算时间复杂度原则

3 忽略系数

	T(3n^2+2n)	T(5n^2+7n)	T(n^3+5n)	T(6n^3+4n)
1	5	12	6	10
2	16	34	18	56
5	85	160	150	770
8	208	376	552	3104
15	705	1230	3450	20310
30	2760	4710	27150	162120
100	30200	50700	1000500	6000400



结论:

- 随着n值变大，5n^2+7n 和 3n^2 + 2n ，执行曲线重合, 说明 这种情况下, 5和3可以忽略。
- 而n^3+5n 和 6n^3+4n ，执行曲线分离，说明多少次方是关键

计算时间复杂度原则

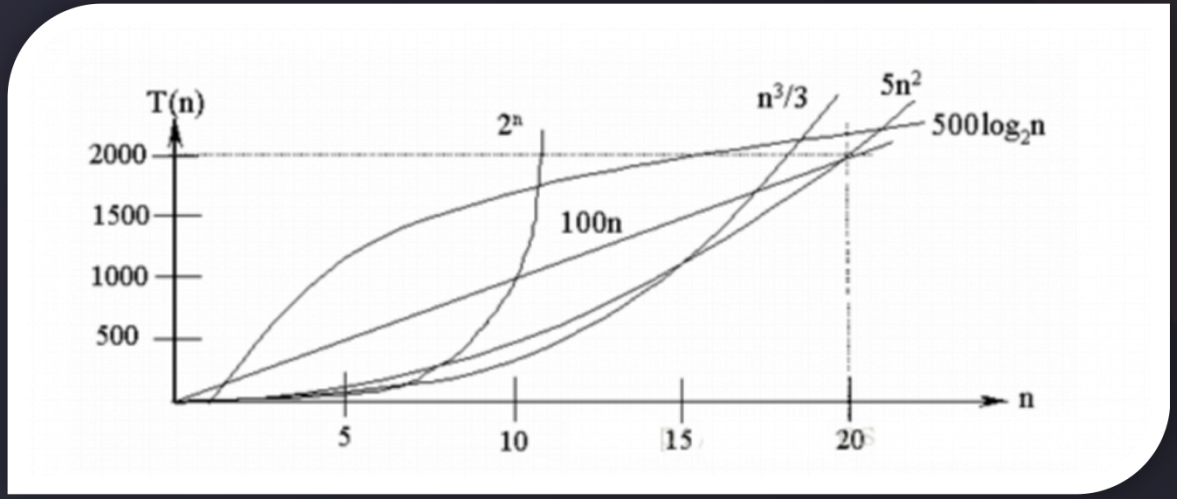
4 计算时间频度

- 用常数1代替运行时间中的所有加法常数 $T(n)=n^2+7n+6$ => $T(n)=n^2+7n+1$
- 修改后的运行次数函数中，只保留最高阶项 $T(n)=n^2+7n+1$ => $T(n) = n^2$
- 去除最高阶项的系数 $T(n) = n^2$ => $T(n) = n^2$ => $O(n^2)$

计算时间复杂度原则

5 常见的时间复杂度

- 常数阶O(1)
- 对数阶O($\log_2 n$)
- 线性阶O(n)
- 线性对数阶O($n\log_2 n$)
- 平方阶O(n^2)
- 立方阶O(n^3)
- k次方阶O(n^k)
- 指数阶O(2^n)



时间复杂度由小到大依次为：$O(1) < O(\log_2 n) < O(n) < O(n\log_2 n) < O(n^2) < O(n^3) < O(n^k) < O(2^n)$

常见的时间复杂度

1 常数阶O(1)

- 无论代码执行了多少行，只要是没有循环等复杂结构，那这个代码的时间复杂度就都是O(1)

```
int i = 1;
int j = 100;
i++;
++j;
int k = i*j;
```

```
int i=1;
int n=100;
while (i<n){
    i = i*2;
}
```

2 对数阶O($\log_2 n$)

- 如果N= a^x （a>0，且a≠1），那么数x叫做以a为底N的对数，记作x=$\log_a n$
- a叫做对数的底数，N叫做真数，x叫做“以a为底N的对数”

常见的时间复杂度

3 线性阶O(n)

- for循环里面的代码会执行n遍，因此它消耗的时间是随着n的变化而变化的

```
int i=1,j=0,n=100;
for(i=1;i<=n;++i){
    j=i;
    j++;
}
```

```
int i=1,j=0,n=100;
for(i=1;i<=n;++i){
    while (i<n){
        j=i;
        j++;
    }
}
```

4 线性对数阶O(n**log*n***)

- 将时间复杂度为O(logn)的代码循环N遍的话，那么它的时间复杂度就是 n * O(logN)，也就是了O(nlogN)

常见的时间复杂度

5 平方阶O(n)

◆ 如果把 O(n) 的代码再嵌套循环一遍，它的时间复杂度就是 $O(n^2)$

```
int i=1,j=0,n=100,res=0;
for(i=1;i<=n;++i){
   for (j=0;j<n;j++){
        res=i;
        res++;
    }
}
```

6 K次方阶O(n^**k**)

$O(n^3)$相当于三层n循环， O(n^k)也就是k次循环

EDU

CSDN学院 IT实战派

下节课再见，记得关注公众号