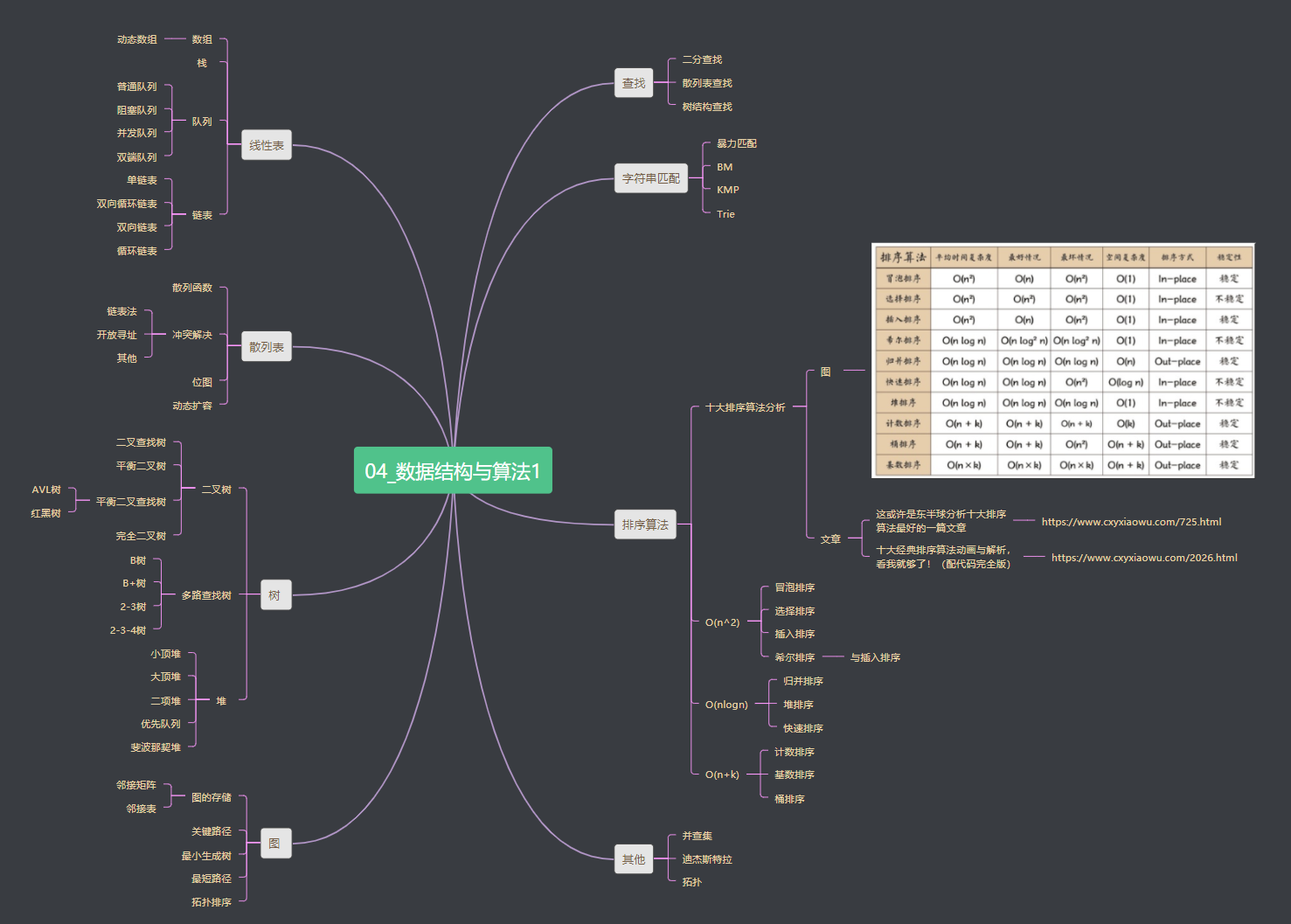
# 数据结构与算法基础

<https://www.processon.com/view/link/5d4eaf62e4b0145255b497a7#map>

<https://www.processon.com/view/link/603da52be401fd7975d5f8be#map>

1. 数据结构与算法入门基础
   1. 算法基础知识
   2. 时间复杂度
   3. 空间复杂度
   4. 入门思维及方法
2. 数据结构与算法基础



判断一个数字是否是2的N次方

while(n>1){

if(n%2==0)

n=n/2

}

1的二进制 01

2的二进制10

4的二进制100

3的二进制011

8的二进制1000

7的二进制0111

if(n&(n-1) == 0) 那么n就是2的次方n>1

算法的特性

五个特征:

1. 有穷性
2. 确定性
3. 可行性
4. 有输入
5. 有输出

设计原则:

1. 正确性
2. 可读性
3. 健壮性
4. 高效率+低存储

评价算法的两个重要指标:

1. 时间复杂度
2. 空间复杂度

时间复杂度计算的意义:在数据量小的时候代码运行效率低无法测试出来，预估时间复杂度可以预估数据量变大之后系统运行速度

时间复杂度表示方法: 大O表示法。O(n),O(nlogn);

O(1), O(n), O(nlogn), O(n^2), O(n+1), O(logn), O(n!)

几种常见的时间复杂度分析：指的是某一段代码。

计算时间复杂度 往往是计算比较大的 而且是不确定的数，如果已经确定了，那么就不用计算了，也是我们说的常量。

1. 常数：O(1) 1表示是常数，所有能确定的数字我们都用O（1），O(1000)=>o(1)
2. 对数：O(logn),O(nlogn)
3. 线性：O(n)
4. 线性对数：O(nlogn)
5. 平方：O(n^2)
6. N次方：O(n^n)

我们怎么找时间复杂度：

1.找到有循环的地方

2.找有网络请求（RPC，远程调用，分布式，数据库请求）的地方。

就是测试时间：log打印，计算平均时间。

几种常见的时间复杂度总结

O(1)>O(logn)>O(n)>O(nlogn)>O(n^2)>O(n^x)

O(1)>O(logn)>O(n)>O(nlogn) 效果都是很好的。几乎优化的空间不是很大。

登录：判断用户名密码O(1)到数据库里查记录；分布式 缓存。

列表的接口：排序，冒泡排序；=>找更优秀的排序算法 快速排序，归并排序，堆排序。

优化的目标就是要往O（1）的方向接近。

空间复杂度分析的意义：找花了内存的地方。数据

如何找出程序的空间复杂度：开了空间的地方， 比如 数组 链表，缓存对象，递归

package algorithm.time;

public class BigO {

public static void main(String[] args) {

int a = 1; //1次 O(1)

for(int i = 0 ;i < 3;i++){//这里会运行几次？4次 在第4次的时候结束 跳出 i=3 (0 1 2 3)

a = a + 1; //这里运行几次？ 3次 O(1)? n+1 n 1 O(3)? => O(1)

}

int n = Integer.MAX\_VALUE; //表示n是未知

int i = 1;

for(int j = 0 ; j < n ;j++){

while ( i <= n){

i = i \* 2;

}

}

while( i <= n){

i = i \* 3; //O(logn)

}

//i的值：2 4 8 16 32,=》2^0,2^1,2^2,2^3,.....2^n

//===> 2^x=n =>求出x就是我们运行的次数 => x=log2n =>计算机忽略掉常数 => x = logn =>O(logn)

//二分查找 为什么是logn的算法？

//1~100 找69这个数

//50:(1+100)/2 = 50

for(i = 0 ; i < n;i++){

a = a +1; //运行了多少次？O(n) n一定是一个未知的，如果n是已知6的

}

for(i = 0 ; i < n;i++){ // 乘法 n次

for(int j = 0 ; j < n ;j ++){ //n次

a = a +1; //运行了多少次？ O(n^2)

}

}

for(i = 0 ; i < n;i++){ // 乘法 n次

for(int j = i ; j < n ;j ++){ //n次

a = a +1; //运行了多少次？ n\*(n+1)/2 => O(n^2); => (n^2+n)/2 => 注意有个规律，有加减法的时候，找次数最高的那个

}

}

/\*

\* 外面的循环次数是确定的 O(n) n次，1 2 3 4 。。。n

\*

\* i=n 运行1次

\* i=n-1 运行2次

\* .

\* .

\* .

\* i=1 运行n次

\*

\* 1,2,3 …… n次 最后里面这层要运行多少次？1+2+3+……+n=n\*(n+1)/2 =>

\*

\*/

/\*int data[100];

List<>; list.add();

Map put

Set add

Queue\*/

}

}