数据结构与算法

(一)

2 算法复杂度

杜育根

ygdu@sei.ecnu.edu.cn

主要内容

- 计算的资源
- 算法的定义
- 算法的评估

计算的资源

- **程序运行时需要的资源有两种**

  时间：程序运行需要的时间。

  空间：程序运行需要的存储空间。

✓资源是有限的

Time Limit和Memory Limit

- OJ上的题目中，常常有对运行时间和空间的说明，例如：
- Time Limit: 2000/1000 MS (Java/Others)
- Memory Limit: 65536/65536 K (Java/Others)

- 这2个限制条件非常重要，是检验程序性能的参数。
- 注意：在现场赛中，为了增加迷惑性，可能不会告诉你，需要自己判断。

- 程序必须在限定的时间和空间内运行结束。
- 问题的“有效”解决，不仅在于能否得到正确答案，更重要的是能在合理的时间和空间内给出答案。

李开复：

“1988年，贝尔实验室副总裁亲自来访问我的学校，目的就是为了想了解为什么他们的语音识别系统比我开发的慢几十倍，而且，在扩大至大词汇系统后，速度差异更有几百倍之多.....

在与他们探讨的过程中，我惊讶地发现一个O(n*m)的动态规划居然被他们做成了O(n*n*m).....贝尔实验室的研究员当然绝顶聪明，但他们全都是学数学、物理或电机出身，从未学过计算机科学或算法，才犯了这么基本的错误。”

- 程序运行的时间：时间复杂度
- 程序运行的空间：空间复杂度
- O(n*m)和O(n*n*m)：就是时间复杂度

- 符号’O’表示复杂度，O(n*m)可以粗略地理解为运行次数是n*m。
- O(n*n*m)比O(n*m)运行时间大n倍。

- 上面这个语音识别的例子，假如n=100
- 李开复的系统O(n*m)，设运行时间是1s；
- 贝尔实验室的系统O(n*n*m)，则需要100s。

如何测量程序的运行时间?

用clock函数统计运行时间。

下面程序中的for语句，循环次数是n。

```
#include <bits/stdc++.h>
using namespace std;
int main(){
    int i,k;
    int n=1e8;
    clock_t start, end;
    start = clock();
    for(i = 0; i < n; i++)
       k++;
    end = clock();
    cout << (double)(end - start) /
CLOCKS_PER_SEC << endl;
}
```

- PC机的算力：
  当n = 1e8 = 10^8时，运行时间0.164s。

  当n = 1e9时，运行时间1.645s。

- 如果题目要求“Time Limit: 2000/1000 MS (Java/Others)”，那么内部的循环次数应该满足n<=10^8，即1亿次以内。

例子：不同算法的效率

hdu 1425 sort

Time Limit: 6000/1000MS (Java/Others) Memory Limit: 64M/32M (Java/Others)

给n个整数，按从大到小的顺序输出其中前m大的数。

输入：每组测试数据有两行，第一行有两个数n，m(0 < n, m < 1000000)，第二行包含n个各不相同，且都处于区间[-500000, 500000]的整数。

输出：对每组测试数据按从大到小的顺序输出前m大的数。

Sample Input

5 3

3 -35 92 213 -644

Sample Output

213 92 3

算法1. 冒泡排序

```
#include<bits/stdc++.h>
using namespace std;
int a[1000001];                                    //记录数字
#define swap(a, b) {int temp = a; a = b; b = temp;} //交换
int n, m;
void bubble_sort(){                  //冒泡排序，结果仍放在a[]中
    for(int i = 1; i <= n-1; i++)
        for(int j = 1; j <= n-i; j++)
            if(a[j] > a[j+1])
                swap(a[j], a[j+1]);
}
int main(){
    while(~scanf("%d%d", &n, &m)){
        for(int i=1; i<=n; i++)  scanf("%d", &a[i]);
        bubble_sort();
        for(int i = n; i >= n-m+1; i--){    //打印前m大的数
            if(i == n-m+1)  printf("%d\n", a[i]);
            else            printf("%d ", a[i]);
        }
    }
    return 0;
}
```

冒泡排序的复杂度

```
void bubble_sort(){    //冒泡排序，结果仍放在a[]中
    for(int i = 1; i <= n-1; i++)
        for(int j = 1; j <= n-i; j++)
            if(a[j] > a[j+1])
                swap(a[j], a[j+1]);
}
```

- 在bubble_sort()中有2层循环，循环次数是：
  $n-1 + n-2 + \ldots + 1 \approx n^2/2;$
- 在swap(a, b)中做了3次操作；总的计算次数是 $3n^2/2$，复杂度记为$O(n^2)$。
- n = 100万时，计算1万亿次。

算法2. 快速排序

- **STL的sort()函数，是改良版的快速排序。**
  在上面的程序中，把bubble_sort()；改为：

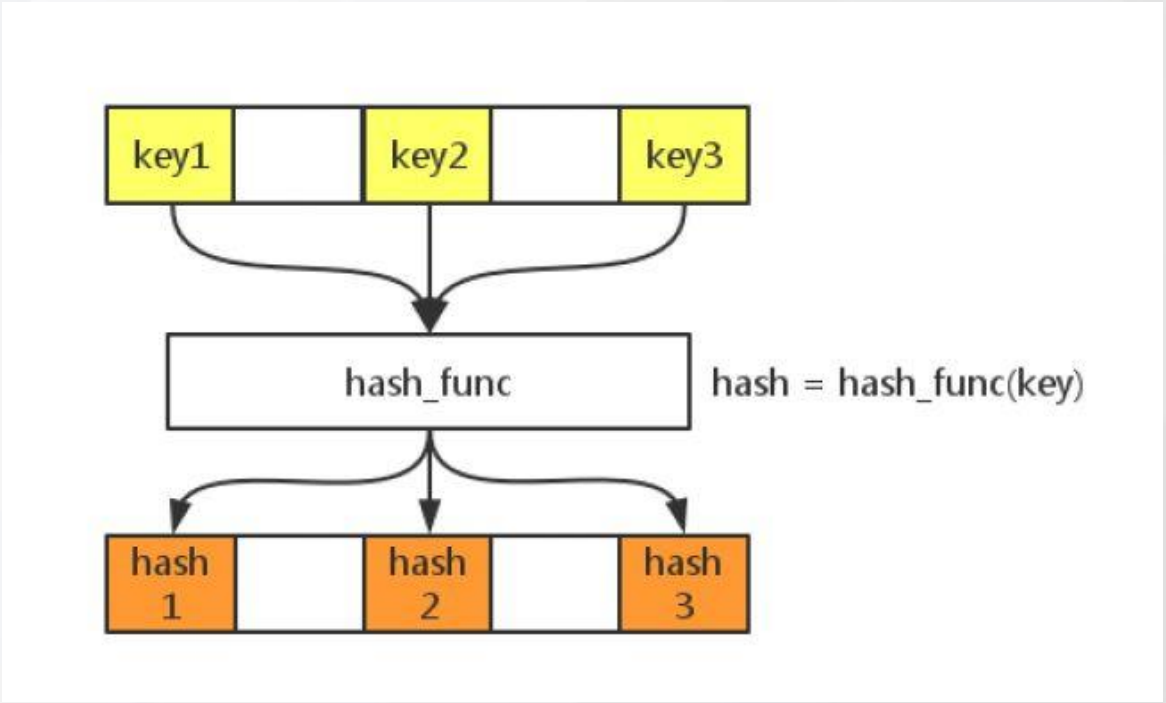
  sort(a + 1, a + n + 1);

  就完成了a[1]到a[n]的排序。

- **算法的时间复杂度是O(nlogn)**
  当n = 100万时，100万 × log2100万 ≈ 2000万。

算法3．哈希

- 哈希算法是一种以空间换取时间的算法。



本题的哈希思路

- 输入一个数字t的时候，对应a[500000 + t]这个位置，记录a[500000 + t] = 1;

- 输出的时候，逐个检查a[i]，如果a[i]等于1，表示这个数存在，打印出前m个。

```
#include<bits/stdc++.h>
using namespace std;
const int MAXN = 1000001;
int a[MAXN];
int main(){
    int n,m;
    while(~scanf("%d%d", &n, &m)){
        memset(a, 0, sizeof(a));
        for(int i=0; i<n; i++){
            int t;
            scanf("%d",&t);//此题数据多，如果用很慢的cin输入，肯定TLE
            a[500000+t]=1; //数字t，登记在500000+t这个位置
        }
        for(int i=MAXN; m>0; i--)
            if(a[i]){
                if(m>1)  printf("%d ", i-500000);
                else      printf("%d\n", i-500000);
                m--;
            }
    }
    return 0;
}
```

哈希的复杂度

- 程序并没有做显式的排序，只是每次把输入的数按哈希直接插入到对应位置，只有1次操作；
- n个数输入完毕，就相当于排好了序。
- 总的时间复杂度是O(n)。

复杂度的表示

- **上述例子中，把n称为问题的数据规模，把程序的复杂度记为O(n)。**
- **复杂度只是一个估计，不需要精确计算。**
  例如在一个有n个数的无序数列中，查找某个数a，可能第一个数就是a，也可能最后一个数才是a。平均查找时间是n/2次，但是仍然把查找的时间复杂度记为O(n)。
- **在算法分析中，规模n前面的系数被认为是不重要的。**

算法的定义

- 算法（Algorithm）：对特定问题求解步骤的一种描述，是指令的有限序列。有5个特征：

(1)输入：一个算法有零个或多个输入。

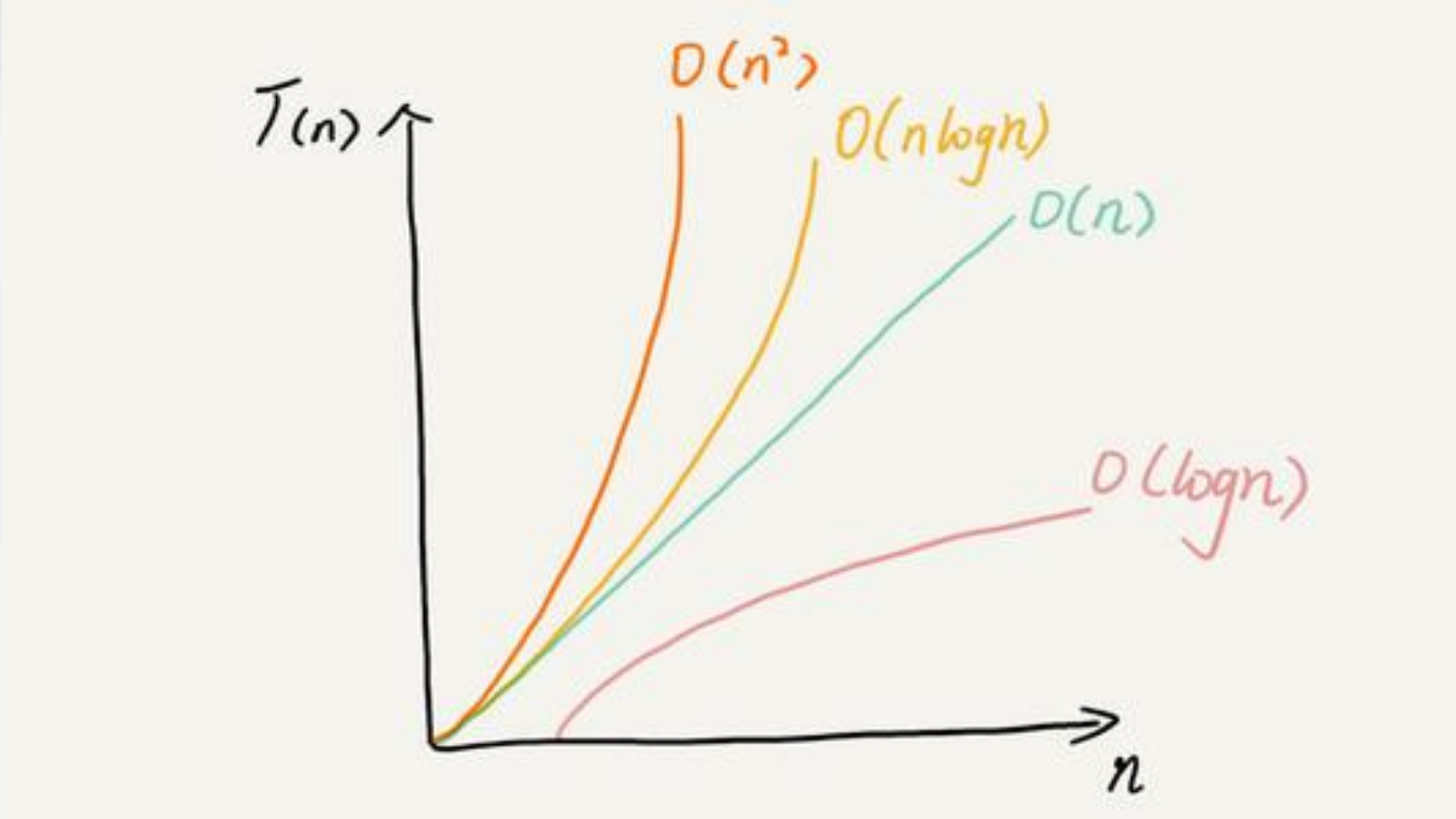
(2)输出：一个算法有一个或多个输出。

(3)有穷性：一个算法必须在执行有穷步之后结束，且每一步都在有穷时间内完成。

(4)确定性：算法中的每一条指令必须有确切的含义，对于相同的输入只能得到相同的输出。

(5)可行性：算法描述的操作可以通过已经实现的基本操作执行有限次来实现。

有哪些复杂度?



算法的评估

- **O(1) 计算时间是一个常数，和问题的规模n无关。**
  用公式计算时，一次计算的复杂度就是O(1)，例如hash算法。在矩阵A[M][N]中查找i行j列的元素，只需要一次访问A[i][j] 。
- **O(logn) 计算时间是对数。**
  通常是以2为底的对数，每一步计算后，问题的规模减小一倍。例如在一个长度为n的有序数列中查找某个数，用折半查找的方法，只需要logn次就能找到。
- **O(n) 计算时间随规模n线性增长。**
  在很多情况下，这是算法能达到的最优复杂度，因为对输入的n个数，程序一般需要处理所有的数，即计算n次。例如查找一个无序数列中的某个数，可能需要检查所有的数。

- O(nlogn) 算法可能达到的最优复杂度。
  快速排序算法是典型例子。
- $O(n^2)$ 一个两重循环的算法，复杂度是$O(n^2)$。
  例如冒泡排序，是典型的两重循环。
- $O(n^3)$、$O(n^4)$等等。
- $O(2^n)$ 一般对应集合问题。
  例如一个集合中有n个数，要求输出它的所有子集。
- O(n!) 在集合问题中，如果要求按顺序输出所有的子集，那么复杂度就是O(n!)

分类

把上面的复杂度分成两类：

- **多项式**复杂度，包括O(1)、O(n)、O(nlogn)、$O(n^k)$等，其中k是一个常数；

$$O(1)<O(\log n)<O(n)<O(n\log n)<O(n^2)<O(n^3)$$

- **指数**复杂度，包括$O(2^n)$、O(n!)等。

$$O(2^n)<O(n!)<O(n^n)$$

易解问题 - - 难解问题：用多项式时间来区分

- 一个算法是多项式复杂度：“高效”算法。
- 指数复杂度：“低效”算法。

- 多项式复杂度的算法，随着规模n的增加，可以通过堆叠硬件来实现，“砸钱”是行得通的；
- 指数复杂度的算法，增加硬件也无济于事，其增长的速度超出了想象力。

多项式函数与指数函数的增长对比

问题规模n	多项式函数					指数函数	
	logn	n	nlogn	n^2	n^3	2^n	n!
1	0	1	0.0	1	1	2	1
10	3.3	10	33.2	100	1000	1024	3628800
20	4.3	20	86.4	400	8000	1048376	2.4E18
50	5.6	50	282.2	2500	125000	1.0E15	3.0E64
100	6.6	100	664.4	10000	1000000	1.3E30	9.3E157

问题规模和可用算法

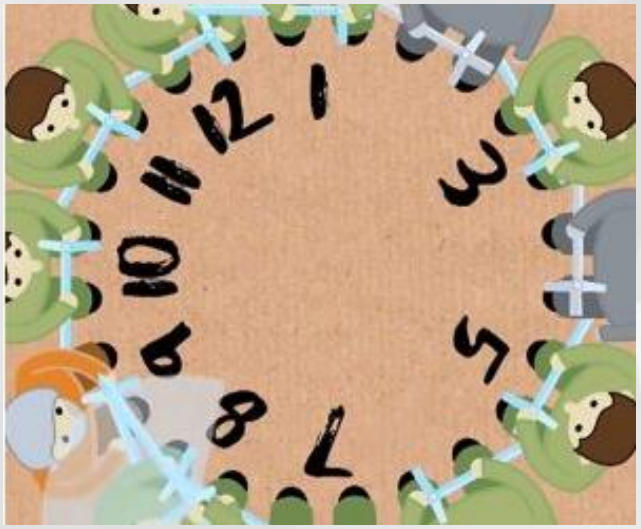
问题规模 n	可用算法的时间复杂度					
	O(logn)	O(n)	O(nlogn)	$O(n^2)$	$O(2^n)$	O(n!)
$n \leq 11$	√	√	√	√	√	√
$n \leq 25$	√	√	√	√	√	×
$n \leq 5000$	√	√	√	√	×	×
$n \leq 10^6$	√	√	√	×	×	×
$n \leq 10^7$	√	√	×	×	×	×
$n > 10^8$	√	×	×	×	×	×

最快的编程方法：打表法

- 一个程序题，你能猜出它所有的输入和输出。（“下界”）
- 如果数据量不大，就可以用“打表”的办法编程。

打表法编程

- Joseph问题http://poj.org/problem?id=1012
- 有k个坏人和k个好人坐成一圈，前k个为好人，后k个为坏人。
- 现在有一个报数m，从编号为1的人开始报数，报到m的人，就杀了他。
- 问当m为什么值时，可以使得在好人死之前，k个坏人先全部死掉?
- 0 < k < 14



一个数一个数地试：

当k=1时，1个好人1个坏人，m=2。

当k=2时，2个好人2个坏人，m=7。

……

当k=14时，m＝13482720。

“打表”：直接输出答案

```
#include <stdio.h>
int a[15]={0, 2, 7, 5, 30, 169, 441, 1872, 7632,
  1740, 93313, 459901, 1358657, 2504881, 13482720};
int main(){
      int k;
      while(scanf("%d", &k)==1){
          if(k==0)
              break;
          printf("%d\n", a[k]);
      }
      return 0;
}
```

课外阅读

算法知识：

- 《算法导论》Thomas H. Cormen，机械工业出版社
- 《算法设计与分析基础》Anany Levitin著，潘彦译

阅读关于计算复杂度的内容