算法基石

[Foundation] of Algorithm

Ruan Xingzhi

洛谷网校

2020 年 1 月 18 日

intro

- 这个直播间**现场打代码**，可能会翻车
- 有问题可以立刻踢出
- 可能会突然发起调查，在聊天区发 1 表示认同，0 表示反对

例

rxz：你会不会线段树？
发 1：我会线段树
发 0：我不会线段树

目录

复杂度

算法：计算方法

一个问题可能有很多种算法来解决，但耗时有显著差异。

冒泡排序一个长度为 100000 的数组，耗时几分钟；快速排序耗时不到一秒钟。

如何衡量算法的运行时间？

例 1

来看下面一份代码。它会执行多少次操作？

```
int getSum(int n)
{
    sum = 0;
    for(i=1; i<=n; i++)
        for(j=1; j<=n; j++)
            sum += i*j;
    return sum;
}
```

例 1

来看下面一份代码。它会执行多少次操作？

```
1 int getSum(int n)
2 {
3     sum = 0;
4     for(i=1; i<=n; i++)
5         for(j=1; j<=n; j++)
6             sum += i*j;
7     return sum;
8 }
```

第 5 行的 for 每次执行需要 n 次操作；这个 for 一共被执行 n 遍。$n + n + \cdots + n = n^2$

例 2

这个例子呢？

```
int getSum(int n)
{
    sum = 0;
    for(i=1; i<=n; i++)
        for(j=i; j<=n; j++)
            sum += i*j;
    return sum;
}
```

例 2

这个例子呢？

```cpp
int getSum(int n)
{
    sum = 0;
    for(i=1; i<=n; i++)
        for(j=i; j<=n; j++)
            sum += i*j;
    return sum;
}
```

$$1+2+\cdots+n=\frac{n(n+1)}{2}$$

O 记号

n 很大的时候，n^2 与 $\frac{n(n+1)}{2}$ 区别不太大。

O 记号是“最坏情况下，程序执行操作的**总次数**”。

- 如果是多个项相加，只取**最大的项**。
- 省略掉所有的常数。

例

$\frac{n(n+1)}{2}$ 是 $O(n^2)$ 的。
$3n-10$ 是 $O(n)$ 的。
n^2+n+1 是 $O(n^2)$ 的。
233 是 $O(1)$ 的。

O 记号

可见，O 记号为我们提供了一种非常简便的方式来衡量复杂程度。

来做点题：

$233n^{100} - 1000000n + 1$ 是

O 记号

可见，O 记号为我们提供了一种非常简便的方式来衡量复杂程度。

来做点题：

$233n^{100} - 1000000n + 1$ 是 $O(n^{100})$

$1 + 2 + 4 + 8 + \cdots + 2^n$ 是

O 记号

可见，O 记号为我们提供了一种非常简便的方式来衡量复杂程度。
来做点题：
$233n^{100} - 1000000n + 1$ 是$O(n^{100})$
$1 + 2 + 4 + 8 + \cdots + 2^n$ 是$O(2^n)$
$123\log_2(n) + 566\ln(n) + 10\lg(n)$ 是

O 记号

可见，O 记号为我们提供了一种非常简便的方式来衡量复杂程度。

来做点题：

$233n^{100} - 1000000n + 1$ 是$O(n^{100})$

$1 + 2 + 4 + 8 + \cdots + 2^n$ 是$O(2^n)$

$123\log_2(n) + 566\ln(n) + 10\lg(n)$ 是$O(\log n)$

log 的问题

log（对数）是指数的逆运算。
$\log_a b$ 是询问“a 的多少次方等于 b”

例子：$\log_2 1024 = 10$，因为 2 的 10 次方是 1024.
来算一算：
$\log_3 9 =$

log 的问题

log（对数）是指数的逆运算。
$\log_a b$ 是询问“a 的多少次方等于 b”

例子：$\log_2 1024 = 10$，因为 2 的 10 次方是 1024.
来算一算：
$\log_3 9 =$2
$\log_{233} 1 =$

log 的问题

log（对数）是指数的逆运算。
$\log_a b$ 是询问“a 的多少次方等于 b”

例子：$\log_2 1024 = 10$，因为 2 的 10 次方是 1024.
来算一算：
$\log_3 9 =$2
$\log_{233} 1 =$0
$\log_{10} 100000 =$

log 的问题

log（对数）是指数的逆运算。
$\log_a b$ 是询问“a 的多少次方等于 b”

例子：$\log_2 1024 = 10$，因为 2 的 10 次方是 1024.
来算一算：
$\log_3 9$ =2
$\log_{233} 1$ =0
$\log_{10} 100000$ =5
由于非常常用，故以 $\ln$ 来表示以 e 为底的对数；用 $\lg$ 来表示以 10 为底的对数。

log 的问题

复杂度的式子中，一般选择省略掉 log 的底数。
这是因为换底公式：

$$\frac{\log_c b}{\log_c a} = \log_a b$$

我们说 $O(\log_2 n)$ 和 $O(\log_3 n)$ 没有区别，因为

$$\log_2 n = \frac{\log_3 n}{\log_3 2}$$

$\frac{1}{\log_3 2}$ 是一个常数，所以 $O(\log_2 n)$ 和 $O(\log_3 n)$ 没区别。
基于这个理由，复杂度中的 $\log$，底数一般都省略掉。

例 3

问时间复杂度:

```
int getSum(int n)
{
    sum = 0;
    for(i=1; i<=n; i++)
        for(j=n; j<=n+5; j++)
            sum += i*j;
    return sum;
}
```

例 3

问时间复杂度：

```
1 int getSum(int n)
2 {
3     sum = 0;
4     for(i=1; i<=n; i++)
5         for(j=n; j<=n+5; j++)
6             sum += i*j;
7     return sum;
8 }
```

注意到 5-6 行的复杂度是 $O(1)$，一共执行 n 次，所以是 $O(n)$.

例 4

问 calc 的时间复杂度：

```
void play()
{
    for(i=1; i<=n; i++)
        printf("I am Happy!!!");
}

void calc()
{
    for(int i=1; i<=n; i++)
        play();
}
```

例 4

问 calc 的时间复杂度：

```
void play()
{
    for(i=1; i<=n; i++)
        printf("I am Happy!!!");
}

void calc()
{
    for(int i=1; i<=n; i++)
        play();
}
```

play 函数复杂度是 $O(n)$，一共被调用 n 次，所以总复杂度是 $O(n^2)$.

例 5

```cpp
void play1() ... // 复杂度是O(1)的
void play2() ... // 复杂度是O(n^2)的
void play3() ... // 复杂度是O(n)的

void calc()
{
    for(int i=1; i<=n; i++)
    {
        play1();
        play2();
        play3();
    }
}
```

例 5

```
1  void play1() ... // 复杂度是O(1)的
2  void play2() ... // 复杂度是O(n^2)的
3  void play3() ... // 复杂度是O(n)的
4
5  void calc()
6  {
7      for(int i=1; i<=n; i++)
8      {
9          play1();
10         play2();
11         play3();
12     }
13 }
```

9-11 行的复杂度是 $O(n^2)$，一共被执行 n 次，所以是 $O(n^3)$.

例 6

问时间、空间复杂度：

```
#define n 100

void calc()
{
    int temp[n+5], sum=0;

    for(int i=1; i<=n; i++)
        temp[n-i+1] = a[i];
    for(int i=1; i<=n; i++)
        sum += a[i]*temp[i];
}
```

例 6

问时间、空间复杂度：

```
#define n 100

void calc()
{
    int temp[n+5], sum=0;

    for(int i=1; i<=n; i++)
        temp[n-i+1] = a[i];
    for(int i=1; i<=n; i++)
        sum += a[i]*temp[i];
}
```

时间 $O(n)$，空间 $O(n)$.

复杂度理论总结

- 我们以 O 记号来表示程序的复杂程度。
- 时间复杂度是**操作次数**，空间复杂度是使用的**内存大小**。
- 复杂度只考虑**数量级**，例如 $2^n + 233n^2$ 是 $O(2^n)$.
- 时间复杂度越大，程序跑的时间就越长。

关于时间复杂度

可以估计，电脑一秒之内执行 1 **亿次运算**。
根据经验，如果时间复杂度算出来在 2000w 以下，一般可以认为 1s 之内能跑出来。

概述

排序是最常见的任务之一，目标是把一个无序数组排好序。（一般是从小到大排序）

例

数组 A:[1, 3, 4, 2, 5] 在排序之后变成 [1, 2, 3, 4, 5]

排序可以用来做什么？

把全班的成绩排序，以便您知道自己有多强；
把洛谷的题目按照通过率排序，然后选通过率最低的题做。

选择排序

最朴素的思路是**枪打出头鸟**：
选出数组中最小值，作为排序结果的第一个元素；再选出次小值，作为结果的第二个元素⋯⋯

如何代码实现？

算法描述

1. c←1
2. 在 A[c, n] 这一段区间内选出最小值，记为 A[x]
3. 将 A[x] 与 A[c] 交换，c++
4. 重复 2-3 步，直到 c=n

无内鬼，一起敲代码。

复杂度分析

选择排序的复杂度：每次选择耗时 $O(n)$，一共执行了 n 次选择。因此，总复杂度是 $O(n^2)$.

冒泡排序

冒泡排序的思想是：
每一趟操作（称为冒泡），我们从左往右扫描这个数组，如果 $a[i] > a[i+1]$，就把它们交换；
执行上述操作 n 次，数组就有序了。

冒泡排序

冒泡排序的思想是：
每一趟操作（称为冒泡），我们从左往右扫描这个数组，如果 $a[i] > a[i+1]$，就把它们交换；
执行上述操作 n 次，数组就有序了。

正确性证明

很容易发现，第一趟冒泡时，最大值会一路被换到最右边；第二趟时，会把次大值换到右数第二个。以此类推，每趟冒泡都会使得一个数归位，因此 n 趟冒泡之后，所有 n 个数都去了该去的位置。

插入排序

想象你是个赌徒，打牌的时候需要把自己的牌排序。
牌是一张一张摸来的，我们只需要**维持手上的牌总是有序**，然后把新来的牌插入到手牌中。

如何代码实现？

算法描述

开一个数组 w，保存目前有序的数组。

1. 枚举 a 中的每个数 x，执行下述操作：
2. 在 w 中找到第一个大于 x 的位置 p
3. 把 w 数组从 p 开始的元素全都往后挪一位，空出 w[p] 这个位置
4. w[p] $\leftarrow$ x

最后 w 数组即排序结果。

简单排序总结

- 选择排序：时间 $O(n^2)$，空间 $O(1)$
- 冒泡排序：时间 $O(n^2)$，空间 $O(1)$
- 插入排序：时间 $O(n^2)$，空间 $O(n)$，但是可以改造为 $O(1)$

课后习题

改造插入排序，使之只使用 $O(1)$ 的额外空间。

归并排序

归并排序是基于分治思想的排序方法。

归并排序 a[l,r] 的算法，简要描述为：

1. 将 a[l,r] 拆成 a[l, mid]、a[mid+1, r] 两个部分
2. 调用 MergeSort(l, mid)、MergeSort(mid+1, r)，将这两个小部分各自排好序
3. 将两个有序数组，合并成一个大的有序数组，覆盖 a[l,r]

思路

算法的正确性是显然的。来看一个例子：

例

目标：排序 [1,8,2,7,3,6,4,5]

- 划分为 [1,8,2,7] 和 [3,6,4,5]
- 各自排序，得到 [1,2,7,8] 和 [3,4,5,6]
- 将 [1,2,7,8] 和 [3,4,5,6] 合并成 [1,2,3,4,5,6,7,8]

那现在问题来了，如何快速实现“将两个有序数组合并起来”？

合并有序数组

现在假设我们要合并 A,B 这两个数组。
思路：如果我们从左到右生成结果数组，那么可以看成是从 A,B 里面一个一个取元素。

- 若 A 已经取完，则从 B 中取。
- 若 B 已经取完，则从 A 中取。
- 若 A,B 都没有取完，则比较 A,B 头部，选择更小的那一个。

例子：合并 [1,2,7,8] 和 [3,4,5,6]

代码实现

框架：

```cpp
void MergeSort(int l, int r)
{
    int mid = (l+r)/2;

    MergeSort(l, mid);
    MergeSort(mid+1, r);

    // todo: merge a[l,mid], a[mid+1,r] -> a[l,r]

}
```

我们来具体敲一遍代码。

复杂度分析

Latex 不好画图，看我用数位板画吧。

归并排序的时间复杂度是 $O(n\log n)$，空间复杂度是 $O(n)$.

排序模板题

https://www.luogu.com.cn/problem/P1177
直接使用归并排序即可。

估测程序时间

考虑冒泡排序和归并排序两个算法。

- 冒泡：$O(n^2) = 10^{10}$，高于 10 亿，肯定跑不过。
- 归并：$O(n \log_2 n) = 1660964$，低于 2000 万，故 1s 内可以跑过。

快速排序

假设一群人站在您面前，您需要把他们按照身高排序。

您可以这样做：

1. 随便选一个人。
2. 比他矮的人站他左边去。
3. 比他高的人站他右边去。
4. 把他左边、右边的人分别排序。

上述第 4 步是递归进行的。干完了之后，整个队伍也就有序了。

快速排序

举个例子。假设我们要排序 [3,2,5,4,8,7,6,1]:

例

- 随便选了个人，身高为 4
- 站队，序列变成 [3,2,1] 4 [5,8,7,6]
- 分别排序，序列变成 [1,2,3] 4 [5,6,7,8]

因此，快速排序的核心在于实现第二步的“站队”。

代码框架

```
void QuickSort(int l, int r)
{
    int flag = a[l];

    // todo: 以flag划分a[l, r]
    // 比flag小的放在a[l, p]
    // 比flag大的放在a[q, r]
    // a[p+1, q-1]是恰好等于flag的

    QuickSort(l, p);
    QuickSort(q, r);
}
```

复杂度分析

如果我们每次都恰好取到中位数作为 flag，那么快速排序是 $O(n\log n)$ 的。
假设我们随机取 flag，那么复杂度大概是有保证的。
复杂度可以证明是 $\Theta(n\log n)$.

在实践上，快速排序是最快的排序方法。因此得名。

STL sort

algorithm 库提供了快速排序：sort
从小到大排序一个数组。参数为起始地址、结束地址。
左闭右开区间！！！

```cpp
#include <algorithm>

int main(void)
{
    int a[10] = {1,4,2,3,5};

    sort(a, a+5);    // 排序a[0]到a[4]
}
```

排序总结

回顾刚刚讲的几种排序方法：

- 简单排序：复杂度普遍为 $O(n^2)$，慢成狗。
- 归并排序：复杂度稳定 $O(n\log n)$
- 快速排序：复杂度 $O(n\log n)$

需要注意依据 n 的大小来选择排序方法。

模拟

模拟是一类算法的统称：题目要你干什么，你就照着用最暴力的方式干一遍，判断结果。
也没有什么特别的技巧。下面来看点题目。

陶陶摘苹果

https://www.luogu.com.cn/problem/P1046

陶陶摘苹果

https://www.luogu.com.cn/problem/P1046

把高度都读进来，数有多少个苹果高度不超过 $h + 30$.

校门外的树

https://www.luogu.com.cn/problem/P1047

校门外的树

https://www.luogu.com.cn/problem/P1047
开一个数组 w，w[x] 为 1 表示 x 有树，为 0 表示没有树。
每次读一个区间，把这个区间内的数全都标成 0. 最后，数 1 的个数，即为答案。这一步 $O(n)$.

复杂度：最坏情况下，每次需要标记整个数组，所以标记操作每次是 $O(n)$. 一共有 M 个区间，所以标记的总复杂度是 $O(nm)$.

故总复杂度：$O(nm+n)=O(nm)$

明明的随机数

https://www.luogu.com.cn/problem/P1059

明明的随机数

https://www.luogu.com.cn/problem/P1059

解法 1

按照题目要求，真的去排序，然后去重。

解法 2

以 w[x] 记录 x 是否出现过，然后从小到大扫描 w 数组。

津津的储蓄计划

https://www.luogu.com.cn/problem/P1089

津津的储蓄计划

https://www.luogu.com.cn/problem/P1089
纯模拟。

车厢重组

https://www.luogu.com.cn/problem/P1116

车厢重组

https://www.luogu.com.cn/problem/P1116
注意到只允许交换相邻元素。那么我们对数组执行冒泡排序，统计交换次数，即为答案。

正确性

冒泡排序过程中，每一个元素都会向自己该去的地方走。
冒泡排序的每一次交换，都会使得数组“更加”有序。也就是说，每一次交换都是有效的。

第 k 小整数

https://www.luogu.com.cn/problem/P1138

第 k 小整数

https://www.luogu.com.cn/problem/P1138
手法和《明明的随机数》如出一辙。我们采用算法 2.

硬币翻转

https://www.luogu.com.cn/problem/P1146

硬币翻转

https://www.luogu.com.cn/problem/P1146
观察样例我们可以发现，先翻转除了第一个以外的；再翻转除了第二个以外的……以此类推。
用位运算可以优雅地实现。

铺地毯

https://www.luogu.com.cn/problem/P1003

铺地毯

https://www.luogu.com.cn/problem/P1003

每次去模拟铺地毯：把对应矩阵内所有元素标记为 id，最后输出目标点的标记值即可。

铺地毯

https://www.luogu.com.cn/problem/P1003
每次去模拟铺地毯：把对应矩阵内所有元素标记为 id，最后输出目标点的标记值即可。
复杂度：每次覆盖耗时 $O(m^2)$，其中 m 为地图大小，是 2×10^5；一共有 $n = 10^4$ 次铺地毯。
所以总复杂度是 $O(nm^2) = 10^4 \times (2 \times 10^5)^2 = 4 \times 10^{14}$，凉了！

铺地毯

所以不能朴素模拟了，需要改进。

铺地毯

所以不能朴素模拟了，需要改进。

从后往前考虑每一块地毯。如果它覆盖了 (x, y)，就直接输出它的编号。

正确性是显然的。复杂度是 $O(n)$.

独木桥

https://www.luogu.com.cn/problem/P1007

独木桥

https://www.luogu.com.cn/problem/P1007
所以可以假装没有碰撞。相当于两个人穿模而过。
则所求 max 为士兵离岸最坏距离的最大值，min 为士兵离岸最好距离的最大值。

三连击

https://www.luogu.com.cn/problem/P1008

三连击

https://www.luogu.com.cn/problem/P1008
枚举第一个数，生成第二、第三个数，看他们是否用尽了 1,2,3...9 即可。

Cantor 表

https://www.luogu.com.cn/problem/P1014

Cantor 表

https://www.luogu.com.cn/problem/P1014
注意到整个 Cantor 表是分层的（斜方向）。
找出它是第几层，即可推断。

跳马问题

https://www.luogu.com.cn/problem/P1644

跳马问题

https://www.luogu.com.cn/problem/P1644
纯模拟即可。

信封问题

https://www.luogu.com.cn/problem/P1595 假装 $n \leq 10$.

信封问题

https://www.luogu.com.cn/problem/P1595 假装 $n \leq 10$.
本题需要**枚举排列**，然后判断枚举出来的排列要不要统计进答案。

组合的输出

https://www.luogu.com.cn/problem/P1157

组合的输出

https://www.luogu.com.cn/problem/P1157

本题需要**枚举子集**。

总结

- 模拟是最朴素的算法。几乎所有题目都可以通过模拟拿到一些分数。
- 枚举排列：采用递归方法
- 枚举子集：利用二进制