大学生程序竞赛算法基础教程

谈文蓉 主 编
校景中 周绪川 副主编

中国工信出版集团
人民邮电出版社
POSTS & TELECOM PRESS

大学生程序竞赛算法基础教程

谈文蓉 ▌主 编
校景中 周绪川 ▌副主编

人民邮电出版社
北京

图书在版编目（CIP）数据

大学生程序竞赛算法基础教程 / 谈文蓉主编. -- 北京 : 人民邮电出版社, 2019.5
ISBN 978-7-115-50921-5

Ⅰ. ①大… Ⅱ. ①谈… Ⅲ. ①计算机算法－竞赛－高等学校－教材 Ⅳ. ①TP301.6

中国版本图书馆CIP数据核字(2019)第043952号

内 容 提 要

本书共 7 章，内容包括枚举、递归、贪心、二分、动态规划、图论和字符串等大学生程序竞赛中的基本算法。

本书注重理论与实践相结合，书中提供的程序样例较多，以便学生学以致用；内容编排力求循序渐进、由浅入深，以保证教材的易用性和可读性。

本书可作为高等院校理工类相关专业的基础算法类课程教材，也可作为大学生程序竞赛中基础算法的培训教材，也可供对程序设计和算法感兴趣的普通读者学习参考。

◆ 主　　编　谈文蓉
副 主 编　校景中　周绪川
责任编辑　邢建春
责任印制　彭志环

◆ 人民邮电出版社出版发行　　北京市丰台区成寿寺路 11 号
邮编　100164　　电子邮件　315@ptpress.com.cn
网址　http://www.ptpress.com.cn
北京市艺辉印刷有限公司印刷

◆ 开本：700×1000　1/16
印张：10　　2019 年 5 月第 1 版
字数：196 千字　　2019 年 5 月北京第 1 次印刷

定价：49.00 元

读者服务热线：(010)81055488　印装质量热线：(010)81055316
反盗版热线：(010)81055315

前言

云计算、大数据和人工智能技术的发展，对算法能力提出了更高的要求，高等院校计算机类专业越来越重视程序能力和算法能力的培养。以 ACM/ICPC 为代表的程序类竞赛为算法类人才的培养提供了一个很好的平台，本书作者在高校长期从事算法类的竞赛培训工作，《大学生程序竞赛算法基础教程》一书的出版可帮助读者对基础算法快速入门。

本书的特点如下。

本书是作者十余年带队参加 ACM 国际大学生程序竞赛积累下来的讲义资料，贴近实战。

本书注重理论与实践相结合，书中提供的程序样例较多，以便学生学以致用。

本书的内容编排力求循序渐进、由浅入深，以保证教材的易用性和可读性。

全书共 7 章，各章主要内容介绍如下。

第 1 章 C/C++简介。介绍程序竞赛中必备的 C/C++语法。

第 2 章基础算法。介绍算法复杂度、枚举、递归、贪心、二分法等基本算法分析方法和策略。

第 3 章基础数学。介绍最大公约数、素数、欧拉函数、算术基本原理、快速幂等算法所需数学知识。

第 4 章数据结构。讲解栈和队列、优先队列、二叉树、并查集、树状数组、RMQ 和线段树等程序竞赛中常用数据结构。

第 5 章动态规划。讲解基本动态规划、背包、01 背包、完全背包、单调队列、数位 DP、区间 DP 和概率 DP 等主流动态规划算法。

第 6 章图论。介绍建图与遍历、邻接矩阵、Vector 邻接表、链式前向星、搜索、深度优先搜索、广度优先搜索、Prim 算法、Kruskal 算法、Floyed 算法、Dijkstra

算法和拓扑排序等图论算法。

第 7 章字符串。讲述 KMP 和 AC 自动机等字符串匹配算法。

本书是作者长期开展高校教育教学改革的成果，得到了国家民委高等教育教学改革项目（No.17025）、西南民族大学教育教学改革项目（No.2017ZDPY06）、教育部高等教育司产学合作协同育人项目（No.201702108001）的资助。教材在编写的过程中，得到了相关教师和 ACM/ICPC 参赛队员的帮助，在此深表感谢。

本书由谈文蓉教授主编，校景中副教授和周绪川教授副主编，万师敏等同学参与。书中的全部程序均经过调试。由于编者水平所限，加之时间仓促，书中不妥之处在所难免，望广大读者不吝赐教。

编　者

2018 年 10 月

目录

第1章 C/C++简介

首先，阅读一段完整的C语言代码。

```
1. #include <stdio.h>
2. int main()
3. {
4.     int sum = 0 , a, b;
5.     scanf("%d %d",&a,&b);
6.     sum = a + b;
7.     printf("%d\n",sum);
8.     return 0;
9. }
```

#include <stdio.h>表示引用了C语言中标准输入输出头文件。头文件是一个包含功能函数和数据接口声明的载体，是C/C++语言家族中不可缺少的组成部分。编译时，编译器通过头文件找到对应的函数库，进而把已引用函数的实际内容导出来代替原有函数。例如，<stdio.h>包含了C语言中的标准输入输出函数（scanf()函数和 printf()函数）；<math.h>包含了许多数学函数，如求平方根值（sqrt()函数）、求绝对值（abs()函数）等。当我们使用这些函数时，一定要先声明相对应的头文件。

main()是主函数名，函数体用一对大括号表示。在一个程序中，有且仅能有一个以main()命名的函数，程序从main()函数的第一行开始运行，最后一行结束。通常将主函数的返回值设为一个int类型的数据，返回0表示程序无异常，返回其他值表示程序出错。

int sum = 0,a,b; 定义了3个int类型的数据，分别为sum，a，b，并把sum值

初始化为 0。

scanf()是 C 语言中的输入函数，被声明在头文件 stdio.h 中，此行代码表示输入了两个值，并把这两个值分别赋给 a 和 b 这两个变量。函数的第一个参数是格式字符串，它指定了输入的格式。例如，%d 为 int 型数据的格式字符串，%c 为 char 型数据的格式字符串，%f 为 float 型数据的格式字符串。&a、&b 中的&是寻址操作符，&a 表示对象 a 在内存中的地址。

prinf()是 C 语言中的输出函数，同样被声明在头文件 stdio.h 中，此行代码表示输出 sum 的值。

return 0 表示函数的返回值为 0，表示程序无异常，正常结束。

接下来，再来看一段完整的 C++代码。

```
1. #include <iostream>
2. using namespace std;
3.
4. int main()
5. {
6.     int sum = 0 , a, b;
7.     cin >> a >> b;
8.     sum = a + b;
9.     cout << sum << endl;
10.     return 0;
11. }
```

这段代码与上述 C 语言代码功能类似，主要区别在于输入输出的方式不同。在 C++语言中，使用了 iostream 的标准头文件，输入函数改为输入流对象，输出函数改为输出流对象。同时，using namespace std 指明了命名空间为 std，由于 C++标准库中的所有组件都是在一个名为 std 的命名空间中，为了防止许多对象出现同名而混淆的情况，程序都要加上命名空间。

虽然这么看上去，C 语言和 C++语言出入不大，但 C 语言基本上是 C++的一个子集，C++在 C 语言基础上增加了基于对象、面向对象的通用模板化编程以及标准模板库。语法上因此也稍有不同。

（1）在 C 语言中，文件名的后缀为.c，而在 C++中以.cpp 为后缀。

（2）在 C 语言中，变量必须在程序的开始部分集中定义，而在 C++中可以在用到的时候再定义。

（3）C++语言中允许多个函数使用相同的函数名，构成重载，但 C 语言中是万万不行的。

（4）在 C 语言中，struct 类型的定义必须要加上 struct 的前缀，而在 C++中，

struct 可以直接使用其类型名定义。

接下来，对 C/C++语法进行简单介绍。

基本数据类型

C/C++拥有丰富的数据结构，如整数类型、实数类型、字符串类型等。以下是 Windows 平台下 32 位操作系统下的数据。

	类型	字节长度
整数类型	short	2（16 位）
	int	4（32 位）
	long	4（32 位）
	long long	8（64 位）
字符串类型	char	1（8 位）
实数类型	float	4（32 位）
	double	8（64 位）
	long double	16（128 位）
布尔类型	bool	1（8 位）

【注】C 语言中不包括布尔类型，它是包括在 C++语言中的。

整型有无符号（unsigned）和有符号（signed）两种类型，在默认情况下，声明的整型变量都是有符号的类型，如果要声明无符号类型，就需要在类型前加上 unsigned。无符号类型和有符号类型的区别是有符号类型需要使用一个比特来表示数字的正负，如 32 位系统中一个 short 能存储的数据范围为–32768～32767（16 位二进制的最高位作为符号位，“1”为负，“0”为正），而 unsigned 能存储的数据范围则是 0~65535（这个最高位不用作符号位，所以是 2^{16}，共 65536）。使用无符号整型，可使正整数的数据范围扩大一倍。

一个 char 的大小和一个机器字符一样，布尔类型（bool）的取值是真（true）或者假（false）。

变量

在程序的运行过程中，值可以改变的被称为变量。变量命名时，只能由数字、字母和下划线组成，且第一个字符只能为字母或者下划线，如下。

```
1.  int  sum = 0;
2.  char  ch;
```

在一段程序中，每个变量都有作用域和生存周期。在函数或者代码块内部声明的变量称为局部变量，局部变量在函数调用完毕或者代码块运行完之后就结束

了生命周期。在所有函数外声明的变量称为全局变量，全局变量在整个代码结束后才结束其生命周期。

局部变量只在函数内或者代码块内使用，而全局变量可以在整个程序中使用。二者可以同名，但在函数内，局部变量会覆盖全局变量的值。请看下面一段代码。

```
1. #include <iostream>
2. using namespace std;
3.
4. int main()
5. {
6.     int i = 10;
7.     cout <<"全局变量 "<<  i << endl;
8.     for(int i = 0 ; i<3; i++)
9.     {
10.         cout <<"局部变量 "<<  i << endl;
11.     }
12.     i++;
13.     cout <<"全局变量 "<<  i << endl;
14.     return 0;
15. }
```

运行结果如下。

```
全局变量 10
局部变量 0
局部变量 1
局部变量 2
全局变量 11
```

在此段代码中，全局变量和局部变量同名，但在代码块内，局部变量覆盖了全局变量。

常量

在程序的运行过程中，值不能改变的被称为常量，常量可以是任何的基本数据类型，一旦被定义就不能修改。常量的定义方法分为两种：

（1）使用#define 宏定义；

（2）使用 const 关键字。

使用方法如下。

```
1. #define  age  20
2. const int age = 20;
```

【注】使用 define 宏定义，句末不需要加分号；const 定义必须赋值。

宏定义是字符替换，没有数据类型的区别，编译时直接将字段进行替换，容

易产生错误。const 关键字定义，有类型区别，在编译时会进行类型检验。

数组

当变量很少时，可以直接定义，但当有成千上万个相同类型的变量时，逐一定义就显得力不从心，这时就引入了数组的概念。数组是有序数据的集合，可以定义任何数据类型的数组，它在内存中开辟了一段连续的空间。在 C 语言中是不能对数组的长度做动态定义的，但在 C++中可以实现。数组又分为一维数组和多维数组，以二维数组为例，二维数组的每个元素又是一个一维数组。例如，a[4][2]这个二维数组，我们可以看成是一个含有 4 个元素的一维数组，但每个元素是一个包含 2 个元素的二维数组。

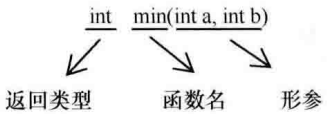
```
#include <iostream>
using namespace std;

const int N = 10;
int main()
{
    int a[N][N];
    for(int i=0;i<N;i++)
    {
        for(int j=0;j<N;j++)
        {
            a[i][j] = i+j;
        }
    }
    //memset(a,0,sizeof(a));
    return 0;
}
```

数组清零的时候，可以遍历一次将每个元素的值设为 0，也可以使用 memset 函数，它是对较大的数组或结构体清零最快的方法，使用方法为 void *memset(void *s, int ch, size_t n)。

函数

为了使程序更清晰易懂，通常将程序模块化，每个模块独立完成自己的功能，这个模块称为函数。在 C/C++程序中至少包括一个函数，程序是由一个主函数和若干个函数构成的，同一个函数可以被不同的函数多次调用，但主函数不能被其他函数调用，只能调用其他函数。

函数是由一个函数头和一个函数主体构成的，函数头又分为返回类型、函数名、参数。



返回类型分为两种：一种为 void 型，即不返回任何数据类型；另一种为其他数据类型，return _type 是其返回的数据类型。参数列表可以为空，也可以传入某些值供函数内使用，这些值称为实参。

函数有 3 种调用方式，分别为传值调用、指针调用和引用调用。

使用传值调用时，把参数的实际值赋值给形式参数，在函数内做的一系列修改对被调用函数中的实际参数没有任何影响。指针调用是把参数的地址复制给形式参数，此时的修改会对实际参数产生影响。引用调用是把参数的引用复制给形式参数，修改也会对实际参数产生影响。

```
1. #include <iostream>
2. using namespace std;
3.
4. void fun1(int a,int b)
5. {
6.     a = 10;
7.     b = 1;
8.     return;
9. }
10. void fun2(int &a,int &b)
11. {
12.     a = 10;
13.     b = 1;
14.     return;
15. }
16. int main()
17. {
18.     int a = 1, b = 10;
19.     cout << a << "  " << b << endl;
20.     fun1(a,b);
21.     cout << a << "  " << b << endl;
22.     fun2(a,b);
23.     cout << a << "  " << b << endl;
24.     return 0;
25. }
```

运行结果如下。

```
1  10
1  10
10  1
```

结构体

结构体通俗讲就像打包封装，把一些有共同特征（如同属于某一类事物的属性，往往是某种业务相关属性的聚合）的变量封装在内部，通过一定方法访问修改内部变量。

结构体的定义如下。

```
1. struct Node{
2.     int a[20];
3.     char b;
4.     float c;
5. }
```

struct 是声明结构体类型时所必须使用的关键字，不能省略。Node 是这个结构体的名称。大括号内是该结构体中的各个成员，由它们组成一个结构体。

结构体有以下几种声明变量方式。

```
struct Node{
    char name[N];
    int b;
    float c;
}node;
```

```
struct Node{
    Char name[N];
    char b;
    float c;
};
struct Node node;
```

```
struct Node{
    int a[N];
    char b;
    float c;
}node={“SWUN”,10,19.5};
```

结构体在定义的时候不能申请内存空间，但如果是结构体变量，声明的时候可以分配，两者关系就像 C++的类与对象，对象才分配内存。结构体的大小通常（只是通常）是结构体所含变量大小的总和。

第2章

基础算法

2.1 算法复杂度

掌握了基本的编程语言，我们就可以用其来解决各种不同类型的问题，但解决问题的途径多种多样，每种途径又对应一种算法。算法是解决问题的步骤，一个高效稳定的算法可以快速让问题得到完美解答，达到事半功倍的效果，然而，算法的复杂度决定了算法的优劣。

算法的复杂度一般从时间复杂度和空间复杂度这两个方面进行评估。

2.1.1 时间复杂度

时间复杂度是指执行算法所需要的计算工作量。在计算机科学中，算法的时间复杂度是一个函数，它定性描述了该算法的执行时间，整个算法的执行时间与基本操作重复执行的次数成正比。

时间复杂度常用大 O 符号表示，如 $O(n)$、$O(\log^n)$等，其中，n 为问题规模，即数据输入的大小。在计算时间复杂度时，先计算基本操作的次数，用 $f(n)$表示，$O(f(n))$代表该算法复杂度是与 $f(n)$成正比的，相当于把 $f(n)$数量级化。

例如，下面这段计算 $1^2+2^2+3^2+4^2+5^2+\cdots+n^2$ 的代码。

```
1. for(int i=1;i<=n;i++)
```

```
2. {
3.     for(int j=1;j<=i;j++)
4.     {
5.         sum += i;
6.      }
7. }
```

循环内的语句执行了 $\frac{n(n+1)}{2}$ 次，取数量级，则这个算法的时间复杂度为 $O(n^2)$。当然，对于一些复杂或庞大的算法，这样精确的计算复杂度显得不太可行，通常，我们采用估算方式，取最大数量级。当运算次数不随着问题规模 n 增长，则时间复杂度为 $O(1)$，称为常数级。当运算次数随着问题规模 n 增长，根据嵌套循环、递归等次数，会有对数级 $O(\log^n)$、线性级 $O(n)$、指数级 $O(C^n)$、阶乘级 $O(n!)$ 等（其中，n 为问题规模大小，C 为一常量）。

常见的几种复杂度的关系为 $c < \log^n < n < n\log^n < n^a < a^n < n!$。

2.1.2　空间复杂度

空间复杂度是指算法在计算机内执行时所需存储空间的度量。算法在执行过程中，本身程序中的变量、数组、结构体和一些数据结构会占用一些空间，也会需要额外的空间，在实际问题中，为了减少空间复杂度，可采用压缩存储的方法。

当数据的输入量不同时，我们可以根据时间和空间复杂度判断算法是否可行，在满足可行的条件下，尽可能使用高效的算法。假设时间限制为 1 s，时间复杂度在 1×10^7 以下一般能流畅运行，达到 1×10^8 且算法结构简单时，或许可勉强运行。而对于空间复杂度，尽量少开一些不必要的空间，数组大约在 1×10^6，具体复杂度预估可根据实际问题判断。

2.2　枚举

把问题所有可能的解一一进行检验，排除后得到正确可行解的过程称为枚举，这种方法是牺牲时间和空间来换取较高的准确性，所以当可能的解范围较大时，一般不建议采取这种方法。枚举的时间复杂度一般为所有可能解的范围，但在绝大多数情况下，可以进行优化处理，缩小可能解的范围，或者根据问题的相关性