第一次习题课

习题讲解

谢强

中国科学技术大学信息学院

2021 年 10 月 19 日

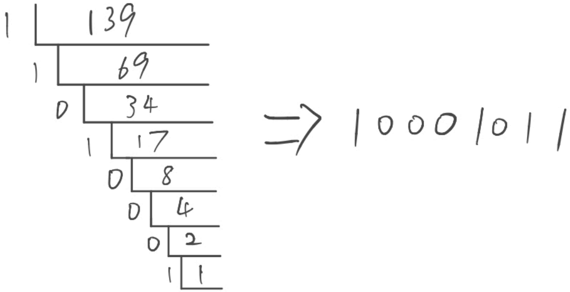
1. 进制转换与补码
   - 进制转换
   - 补码：关于负数的问题
   - `trick`：关于最大值
2. 表达式求值
   - 优先级
   - 表达式的类型判断
3. 关于实验题的一些相关知识
   - 线性代数
   - 线性递推
4. 其他问题
   - 数据类型
   - 悬空 `if`

简单的例子

- 今年将进制转换剔除出了考纲，但是进制转换是计软信方向必须熟练掌握的内容
- 计算机课堂上讲述的进制转换其实是以十进制转二进制为核心的
- 课堂上，我们讲述了如何使用整除取余后倒置的方法来进行十转二的过程

例题

- 例 1. 将 $(139)_{10}$ 转换为二进制、



更快速的方式

- 熟记 2 的整数次方
- 从大到小试减
- 例如：

$$\begin{aligned} 139 &= 128 + 11 \\ &= 128 + 8 + 3 \\ &= 128 + 8 + 2 + 1 \end{aligned} \tag{1}$$

- 因此我们得到，139 在第 7、3、1、0 位为 1，其他位为 0
- 根据经验来说，这样进行进制转换熟练以后更快，大家可以根据自己的喜好尝试

- 同样地，补码也不在今年的考纲中
- 按照往年的情况来看，大家对补码的理解都比较一般
- 对于补码，其实大家都不陌生。在生活中常见的例子是时钟
- 我们按照 12 小时为例子，时针向前拨 7 小时与向后拨 5 小时的效果时一样的

补码：模的意义

- 大家可以用这种方式来理解补码。我们称循环的周期为模，那么钟表的模为 12
- 同样地，一个 int 型整数的模为 2^{32}，因此我们取其中一半为负
- $0 \cdots 2^{31} - 1$ 为非负，$-2^{31} \cdots -1$ 为负数
- 如何计算补码呢，对于负数，我们只需要用模 2^{32}(也就是 1 带 32 个 0) 减去就行
- 对于补码相加，我们只需要正常相加后对模取余数（也就是 32 位开始截断）
- 由于运算在 mod32 下成立，因此乘法也是可以直接运算的

补码：底层计算相关

- 我们尝试从计算的方向来理解补码
- 假设某个十进制的系统，计算的结果不超过 999
- 我们如何来计算 $123-45$ 呢？
- 可以看出，在这次计算中，我们会出现减法的借位，我们希望避开这种计算

如何防止借位

- 因此我们换一种计算方法

$$\underline{123 - 45 = 123} + (999 - 45) + 1 - \underline{1000}$$

- 在这次计算中，我们先计算 999-45。注意这个减法是一定不会借位的。
- 最后，我们加 1 再减去 1000，由于千位数最多溢出 1，因此也不会借位。
- 因此可以类比，999-45 就是我们学习的反码，加上 1 以后也就是补码，最后-1000 的操作相当于高位自动溢出。

- 计算机底层其实并没有所谓的原码和反码，所有的有符号整数运算都是在补码意义下的
- 回顾我们学习的 32 位下如何将一个正数变为负数：
    - 负数反码的值，也就是 `0xffffffff` 减去其正数的二进制值
    - 这个减法在计算机中有更加简单的实现，也就是每一位取反
    - 加上 1 以后得到了我们的补码

以上的计算过程，事实上也就是补码的来源。

int32 有符号的最大值

- 根据上面的表述，我们得到最大值应该为 $2^{31}-1$
- 因此，我们需要快速得到最大值的话，应该有许多写法

各种写法

```
0x7fffffff;
~(1<<31);
(1<<31)-1;
(-1)>>1;
*(int*)"\xff\xff\xff\x7f";
```

优先级

- 运算符号的优先级在表达式的计算中是绝对不能绕过的内容，大家需要记忆一些常见的运算符的相对等级。这里有一些比较一般性的规律
  1. 各种单目 > 乘除 > 加减 > 左右移 > 关系判断 > 位运算 > 逻辑 > 三目 > 赋值
  2. 非 > 且 > 或
  3. 关系判断中，等号和不等号优先级较低
  4. 三目运算和赋值都是右结合

逻辑运算符

- 非运算符相当于负号
- 与运算符在二进制下相当于乘号
- 相对地，或运算相当于加号
- 类比算术运算符号，这几个符号的优先级可以这么记忆

赋值运算符

- 右结合的性质，导致赋值运算同时出现的时候是从右向左运行
- 判断下面的写法：

右结合

```
a*=a+=3;      //结果为(a+3)^2
(a*=a)+=3;    //结果为a^2+3
```

隐式类型转换

- 当两个不同类型的数值进行计算的时候，一般会将它们转换为他们之中更高精度的类型。
- 例如，double 型的变量与 int 型的变量作乘法，会都先提升到 double 型，然后进行运算，结果也为 double 型。
- 观察如下运算：

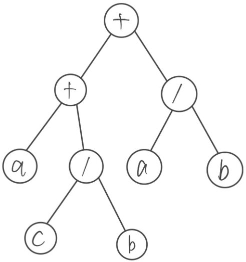
a+c/b+a/b

我们假设 a=1,b=2,c=3.0，前面两者为 int，c 为 double

结果应当为 2.5

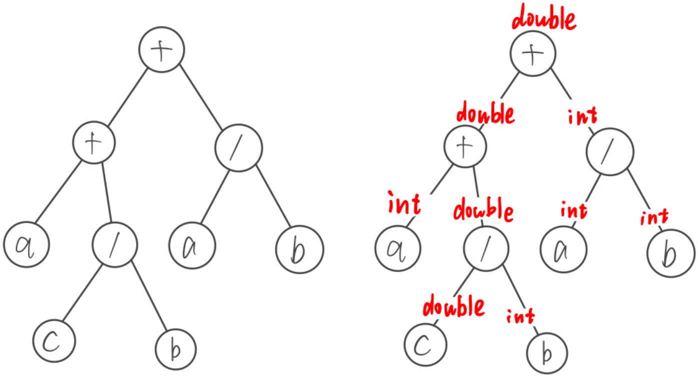
初识表达式树

- 对于上述运算，我们描绘这样的图形：



图：a+c/b+a/b 的表达式树

- 从树枝开始计算，从下到上从左到右



图： 表达式的类型转换顺序

矩阵和向量

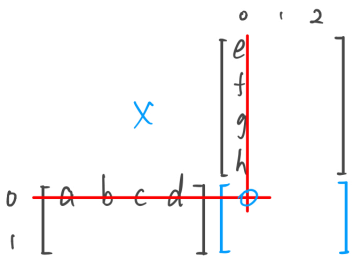
- 二维数组天然地契合了矩阵
- 我们可以自然地用数组元素 a[i][j] 来对应矩阵元素 $A_{i,j}$
- 而向量就是一行或者一列的矩阵，可以用数组 a[1][n] 或者 a[n][1] 来表示
- 注意，矩阵的标号从 0 开始和 1 开始并不影响，只要自己在编程的时候统一即可

矩阵乘法

$$A \times B = C$$
$$C_{i,j} = \sum_{k=0}^{m-1} A_{i,k} \times B_{k,j}$$

- 一句话概括：$C_{i,j}$ 填入 A 的第 i 行与 B 的第 j 列的点乘

图示



- 因此我们知道矩阵乘法不可交换，并且有以下规则
- A 的列数必须等于 B 的行数，1*m 与 m*n 矩阵乘法得到 1*n 矩阵

代码实现

计算矩阵乘法 c++ 语法

```
int a[l][m],b[m][n],c[l][n];
//输入 a,b 清零 c
for(int i=0;i<l;i++)
    for(int j=0;j<n;j++)//开始计算cij
        for(int k=0;k<m;k++)//算点乘
            c[i][j]+=a[i][k]*b[k][j];
```

矩阵与线性递推

大家尝试计算以下矩阵乘法

$$\begin{bmatrix} f_{n-1} & f_n \end{bmatrix} \times \begin{bmatrix} 0 & 1 \\ 1 & 1 \end{bmatrix}$$

这里的 `f` 指 `Fibonacci` 数列

- 数据类型规定了数据在内存中存放的空间和编码方式
- 大家在初学的时候可能会有各种疑问，这里统一一套规范
  - int-32 位，输出用%d
  - char-8 位，输出用%c
  - short-16 位，一般可以用%d 输出
  - long long-64 位，用%lld 输出
  - float-单精度，用%f 输出
  - double-双精度，用%lf 输出

if-else 与花括号

- 观察如下省略的代码：

悬空 if

```
int a=2,b=1;
if(a==1)
    if(b==2) printf("case1");
else printf("case2");
```

感谢

Thank You for Your Attention!