进度信息

Datalab 差3个浮点数，别的完成（我算法菜。。。有不少都看了wp才会。。。哭了），视频看了南京大学计算机系统基础1-15，深入理解计算机系统看到了第一部分的第二章整数运算

👉bitXor

使用～和&来实现^

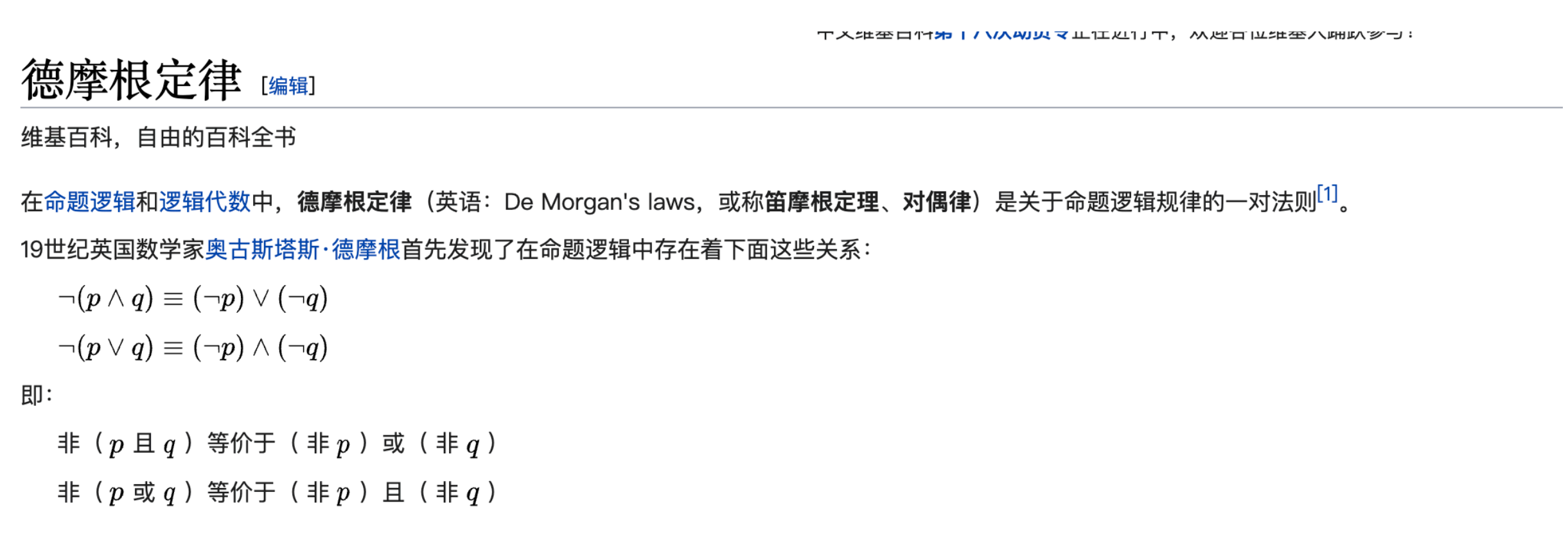
离散数学： a ^ b = (~a ∧ b) V (a ∧ ~b) , 使用得摩根律来去掉 V

```c
int bitXor(int x,int y)
{
    return ~(~(~x&y)&~(x&~y));
}
```

还有一种是使用同或的非来实现

```c
int bitXor(int x, int y)
{
    return ~(x&y)&~(~x&~y);//同或的非
}
```

德摩根律：



德摩根定律 [编辑]

维基百科，自由的百科全书

在命题逻辑和逻辑代数中，**德摩根定律**（英语：De Morgan's laws，或称**笛摩根定理**、**对偶律**）是关于命题逻辑规律的一对法则[1]。

19世纪英国数学家奥古斯塔斯·德摩根首先发现了在命题逻辑中存在着下面这些关系：

$\neg(p \wedge q) \equiv (\neg p) \vee (\neg q)$

$\neg(p \vee q) \equiv (\neg p) \wedge (\neg q)$

即：

非（p 且 q）等价于（非 p）或（非 q）

非（p 或 q）等价于（非 p）且（非 q）

👉isTmax(x)

```c
1  /*
2   * isTmax - returns 1 if x is the maximum, two's complement number,
3   *     and 0 otherwise
4   *   Legal ops: ! ~ & ^ | +
5   *   Max ops: 10
6   *   Rating: 1
7   */
8  int isTmax(int x) {
9    int i = x+1;//Tmin,1000...
10   x=x+i;//-1,1111...
11   x=~x;//0,0000...
12   i=!i;//exclude x=0xffff...
13   x=x+i;//exclude x=0xffff...
14   return !x;
15 }
```

思路

这个里面最秀的就是排除0xffffffffffffffff的情况，当输入补码的最大值得时候，如果没有12， 13行的判断，其实0xffffffffffffffff会出现被当做补码最大值得误判情况，因为在判断补码最大值得时候我们先将其转换为0xffffffffffffffff，然后求出全0得情况，进行判断的。

其中：

！ 符号是对最低位进行求反，逻辑非。

👉negate(x)

【💡点击最左侧的“+”，选择“附件音频与录制”功能，一边听课一边实时录音，遇到重点随时打标记】

```c
/*
 * negate - return -x
 *   Example: negate(1) = -1.
 *   Legal ops: ! ~ & ^ | + << >>
 *   Max ops: 5
 *   Rating: 2
 */
int negate(int x) {
  return ~x+1;
}
```

思路：

补码其实是一个阿贝尔群（可交换群），对于x来说，-x是其补码，所以-x可以通过x取反+1得到。

👉isAsciiDight(x)

```c
/*
 * isAsciiDigit - return 1 if 0x30 <= x <= 0x39 (ASCII codes for characters '0' to '9')
 *   Example: isAsciiDigit(0x35) = 1.
 *            isAsciiDigit(0x3a) = 0.
 *            isAsciiDigit(0x05) = 0.
 *   Legal ops: ! ~ & ^ | + << >>
 *   Max ops: 15
 *   Rating: 3
 */
int isAsciiDigit(int x) {
  int sign = 0x1<<31;
  int upperBound = ~(sign|0x39);
  int lowerBound = ~0x30;
  upperBound = sign&(upperBound+x)>>31;
  lowerBound = sign&(lowerBound+1+x)>>31;
  return !(upperBound|lowerBound);
}
```

思路：

求出两个操作数使得一个数是加上比0x39大的数后符号由正变负，另一个数是加上比0x30小的值时是负数。然后获取他们的符号位进行与操作，就可以了

👉conditional

```c
/*
 * conditional - same as x ? y : z
 *   Example: conditional(3,4,5) = 4
 *   Legal ops: ! ~ & ^ | + << >>
 *   Max ops: 16
 *   Rating: 3
 */
int conditional(int x, int y, int z) {
  x = !!x;
  x = ~x+1;
  return (x&y)|(~x&z);
}
```

思路：

首先对于逻辑运算来说，如果a不是0，那么无论他是多少，都会被看成1，所以如果 a = 2,那么 !a = 0，所以x = !!x 得到的是0或者1，那么x = ~x + 1就会求得他们的相反数。那么利用下面的运算可以得到，如果x为0的话就会得到z的值，如果x的值不为0的话就会得到y的值，good。

👉logicalNeg

```c
/*
 * logicalNeg - implement the ! operator, using all of
 *              the legal operators except !
 *   Examples: logicalNeg(3) = 0, logicalNeg(0) = 1
 *   Legal ops: ~ & ^ | + << >>
 *   Max ops: 12
 *   Rating: 4
 */
int logicalNeg(int x) {

  return ((x|(~x+1))>>31)+1;
}
```

思路：

除了，0和最小值（符号位为1，其余为0）外，其他数字的补码和原来都是相反数的关系，所以，我们可以通过求补码啊然后进行|操作使得其为0，然后结果再加1，就可以是非0情况，那么对于0和最小值，我们可以通过他们的符号位来判断，0的补码和符号位相与为0，但是最小值的符号位与其补码相与为1，所以我们就可以实现 !操作。

注意点：

在c语言里面的>>都是带符号的算术右移。

👉howManyBits

```c
int howManyBits(int x) {
  int n = 0 ;
  x^=(x<<1);
  n +=  (!!( x & ((~0) << (n + 16)) )) << 4;
  n +=  (!!( x & ((~0) << (n + 8)) )) << 3;
  n +=  (!!( x & ((~0) << (n + 4)) )) << 2;
  n +=  (!!( x & ((~0) << (n + 2)) )) << 1;
  n +=  (!!( x & ((~0) << (n + 1)) ));
  return n+1;
}
```

(!!(x & ((~0) << (n + 16))))这个操作是为了判断特定位置范围内是否存在1，我们从高16位开始往下检查看是否存在1，同时， x^=(x<<1) 的结果是如果是正数的话，那么比最高位还高一位被置1，如果是负数的话，那么他会被转化为比最初负数的最高0所在位多一位的正数，比如 -3 - > 7,那么补码所需要的最小位数，就转化为求最高位所在位数的情况了。

👉BitsfloatFloat2Int(f) + 另外几个

```c
unsigned floatScale2(unsigned uf)
{
    int exp = (uf & 0x7f800000)>>23;//求出阶码
    int sign = uf&(1<<31);
    if(exp==0)//求出符号位
    {
        return uf<<1|sign; //阶码为0，直接左移然后加上符号位
    }
    if(exp==255)
    {
        return uf; //NaN
    }
    exp++;
    if(exp==255)
    {
        return 0x7f800000|sign; //返回原符号无穷大
    }
    return (exp<<23)|(uf&0x807fffff); //返回指数加一后的原符号数，指数+1等于乘以2
}



int floatFloat2Int(unsigned uf) {
  int s_    = uf>>31;
  int exp_  = ((uf&0x7f800000)>>23)-127;
  int frac_ = (uf&0x007fffff)|0x00800000;
  if(!(uf&0x7fffffff)) return 0;

  if(exp_ > 31) return 0x80000000;
  if(exp_ < 0) return 0;

  if(exp_ > 23) frac_ <<= (exp_-23);
  else frac_ >>= (23-exp_);

  if(!((frac_>>31)^s_)) return frac_;
  else if(frac_>>31) return 0x80000000;
  else return ~frac_+1;
}


unsigned floatPower2(int x) {

  int INF = 0xff<<23;
  int exp = x + 127;
  if(exp <= 0) return 0;
  if(exp >= 255) return INF;
  return exp << 23;
}
```

关键在于理解浮点数的几个特殊值。

👉课堂记录

👉拓展资料

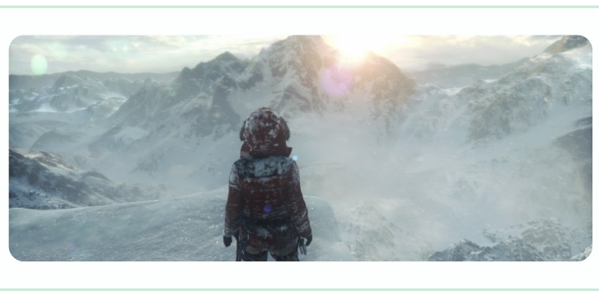
网页书签

> 添加网页书签，了解更多信息，拓展视野，方便查阅。

CSAPP 之 DataLab详解，没有比这更详细的了

纸上得来终觉浅，绝知此事要躬行。 因为md转换不够好，所以后续文章都没有转移到知乎来，可以在我的博客中查看。为

https://zhuanlan.zhihu.com/p/59534845



【💡点击最左侧的“+”，选择“网页书签”功能，将相关课程资料链接粘贴在对话框中，即可插入链接，便于随时查看】