位运算, 布隆过滤器, LRU, 排序

笔记本:	程序设计与算法		
创建时间:	2020/10/22 2:14	**更新时间:**	2020/10/27 1:22
作者:	Jie Zhong		

位运算

按位与“&”

1&1 为1
1&0 为0
0&1 为0
0&0 为0

通常用来将变量中的某些位清0且同时保留其他位不变。也可以用来获取变量中的某一位。

按位或“|”

1|1 为1
1|0 为1
0|1 为1
0|0 为0

通常用来将某变量中的某些位置1且保留其他位不变。

按位异或“^”

1^1 为0
1^0 为1
0^1 为1
0^0 为0

***一个比特跟1进行异或，则会被反转，而跟0进行异或，则不会被反转
通常用来将变量中的某些位取反，且保留其他位不变。

按位非“~” （单目）

~0 为1
~1 为0

左移运算符"<<"

a << b

- 将a的各二进制位全部左移b位，左移时，高位丢弃，低位补0，而得到的值。
- 实际上，左移1位，就等于乘以2，左移n位，就等于乘以2^n。而左移操作比乘法快得多。

右移运算符">>"

a >> b

- 将a的各二进制位全部右移b位，右移时，低位丢弃，高位补进来。
  - 对于无符号的数，高位补0
  - 对于有符号的数，符号位将一起移动，并且大多数C/C++编译器规定，如果原符号位为1，则右移时高位补1，如果原符号位为0，则高位补0。
- 实际上，右移n位，就相当于除以2^n，并将结果往小里取整（因为可能除不进）。

位运算实战

```
x ^ 0 = x
```

x ^ 1s = ~x //1s = ~0
x ^ (~x) = 1s
x ^ x = 0
c = a ^ b => a ^ c = b, b ^ c = a //交换a, b两个数
a ^ b ^ c = a ^ (b ^ c) = (a ^ b) ^ c // associative
将 x 最右边的 n 位清零： x & (~0 << n)
获取 x 的第 n 位值（0 或者 1）：(x >> n) & 1
获取 x 的第 n 位的幂值： x & (1 << n)
仅将第 n 位置为 1 x | (1 << n)
仅将第 n 位置为 0 x & (~ (1 << n))
将 x 最高位至第 n 位（含）清零： x & ((1 << n) 1

判断奇偶 • x % 2 == 1 -----> (x & 1) == 1 • x % 2 == 0 -----> (x & 1) == 0
x >> 1 -----> x / 2. • 即： x = x / 2; -----> x = x >> 1
X = X & (X - 1) 清零最低位的 1
X & -X => 得到最低位的 1
X & ~X => 0

布隆过滤器 (Bloom Filter)

实现 - 一个很长的二进制向量和一系列随机映射函数。布隆过滤器可以用来检索一个元素是否在一个集合中。
优点：空间复杂度和时间复杂度远超一般算法
缺点：有一定的误识别率，删除困难
应用

- 分布式系统（Map Reduce）- Hadoop，搜索引擎
- Redis缓存
- 垃圾邮件，评论等过滤
- 比特币网络

LRU (Least Recent Used)

- 缓存大小 capacity
- 替换测量

实现
　　Hash Map + Double LinkedList
时间复杂度

- 查询 O(1)
- 修改，更新 O(1)

初级排序

- 选择排序

```
public void selectionSort(int[] arr) {
    int len = arr.length;
    for (int i = 0; i < len - 1; i++) { //每次循环后将第i小的元素放好
        int min_idx = i;
        //用来记录第i个到第len-1个元素中，最小的那个元素的下标
        for (int j = i; j < len; j++) {
            if (arr[j] < arr[min_idx]) min_idx = j;
        }
        //将第i小的元素放在第i个位子上，并将原来占着第i个位置的元素挪到后面
        int tmp = arr[i];
        arr[i] = arr[min_idx];
        arr[min_idx] = tmp;
    }
}
```

- 插入排序

```
public void insertionSort(int[] arr) {
    int size = arr.length;
    for (int i = 1; i < size; i++) {
        //arr[i]最左的无序元素，每次
        int curr = arr[i];
        //循环前面的有序元素，将arr[i]放到合适的位置
        int j = i - 1;
        //arr[i] 和前面的有序部分[0, i - 1]比较，放到维持有序的位置
        while (j >= 0 && arr[j] > curr) {
            arr[j + 1] = arr[j];
            j--;
        }
        arr[j + 1] = curr;
    }
}
```

- 冒泡排序

```
public void bubbleSort(int[] arr) {
    int size = arr.length;
    for (int i = size - 1; i > 0; i--) {
        for (int j = 0; j < i; j++) {
            if (arr[j] > arr[j + 1]) {
                int tmp = arr[j];
                arr[j] = arr[j + 1];
                arr[j + 1] = tmp;
            }
        }
    }
}
```

高级排序

- 归并排序

```
//归并排序
public void mergeSort(int[] arr) {
    if (arr == null || arr.length <= 1) return;
    mergeSort(arr, 0, arr.length - 1);
}
public void mergeSort(int[] arr, int s, int e) {
    if (s >= e) return;
```

```
        int mid = s + (e - s)/2;
        mergeSort(arr, s, mid);
        mergeSort(arr, mid + 1, e);
        merge(arr, s, mid, e);
    }
    private void merge(int[] arr, int s, int mid, int e) {
        int[] tmp = new int[e - s + 1];
        int p = 0, p1 = s, p2 = mid + 1;
        while (p1 <= mid && p2 <= e) {
            tmp[p++] = arr[p1] <= arr[p2] ? arr[p1++] : arr[p2++];
        }
        while (p1 <= mid) {
            tmp[p++] = arr[p1++];
        }
        while (p2 <= e) {
            tmp[p++] = arr[p2++];
        }


        for (int i = 0; i < e - s + 1; i++) {
            arr[s + i] = tmp[i];
        }
    }
```

- 快速排序

```
public void quickSort(int[] arr) {
        if (arr == null || arr.length <= 1) return;
        quickSort(arr, 0, arr.length - 1);
    }


    private void quickSort(int[] arr, int s, int e) {
        if (s >= e) return;
        /**
         * 从第K个元素开始分，把K挪到适当的位置，使得比K小的元素都在K左边，比K大的元素都在K右边
         * 把K左边的部分快速排序
         * 把K右边的部分快速排序
         */
        int K = arr[s];
        int i = s, j = e;
        int tmp = 0;
        while (i != j) {
            while (j > i && arr[j] >= K) --j;
            //swap
            tmp = arr[i]; arr[i] = arr[j]; arr[j] = tmp;
            while (i < j && arr[i] <= K) ++i;
            //swap
            tmp = arr[i]; arr[i] = arr[j]; arr[j] = tmp;
        } //处理完之后，arr[i] == K
        quickSort(arr, s, i - 1);
        quickSort(arr, i + 1, e);
    }
```

- 堆排序

非比较排序

- 计数排序
- 桶排序
- 基数排序