STL

- set
  - 平衡树
    - 慢
  - 不支持排名
  - 可以查询前趋后继
    - it =find(x) it--;
    - it=find(x) it++;
  - function
    - insert
    - lower_bound(x) >=
    - upper_bound(x) >
    - erase()
    - count()
    - find()
- map
  - 使用
    - map<xx,xx>
    - 可以用pair的相同类型来使用迭代器
  - 实现
    - 平衡树
  - 有[]操作
    - key
    - key value
  - 可能比set快
  - 根据key值快速查找记录，查找的复杂度基本是Log(N)
  - 用法：
    - map<int, string> mapStudent;
    - 插入：
      - mapStudent.insert(pair<int, string>(1, "student_one"));
      - mapStudent[1] = "student_one"; 【会覆盖】
    - 迭代器：
      - map<int, string>::iterator iter;
      - for(iter = mapStudent.begin(); iter != mapStudent.end(); iter++)
      - cout<<iter->first<<' '<<iter->second<<endl;
    - 查找：
      - map<int, string>::iterator iter;
      - iter = mapStudent.find(1);

        - iter = mapStudent.lower_bound(2);
        - iter = mapStudent.upper_bound(2);
        - Equal_range函数返回一个pair，pair里面第一个变量是Lower_bound返回的迭代器，pair里面第二个迭代器是Upper_bound返回的迭代器，如果这两个迭代器相等的话，则说明map中不出现这个关键字
    - 删除：
        - iter = mapStudent.find(1);
        - mapStudent.erase(iter);
        - int n = mapStudent.erase(1);//如果删除了会返回1，否则返回0
        - mapStudent.erase();
        - mapStudent.begin(), mapStudent.end());
- hash table
    - Hash表实现
        - 拉链 分散地址
    - STL中hash_map扩容
        - (1) 创建一个新桶，该桶是原来桶两倍大最接近的质数(判断n是不是质数的方法：用n除2到sqrt(n)sqrt(n)范围内的数)；
        - (2) 将原来桶里的数通过指针的转换，插入到新桶中(注意STL这里做的很精细，没有直接将数据从旧桶遍历拷贝数据插入到新桶，而是通过指针转换)
        - (3) 通过swap函数将新桶和旧桶交换，销毁新桶
- tree
    - 红黑树
        - 节点为红色或者黑色；
        - 根节点为黑色；
        - 从根节点到每个叶子节点经过的黑色节点个数的和相同；
        - 如果父节点为红色，那么其子节点就不能为红色。
        - 叶节点是黑色
    - 红黑树与AVL树的区别
        - 红黑树与AVL树都是平衡树，但是AVL是完全平衡的(平衡就是值树中任意节点的左子树和右子树高度差不超过1)；
        - 红黑树效率更高，因为AVL为了保证其完全平衡，插入和删除的时候在最坏的情况下要旋转logN次，而红黑树插入和删除的旋转次数要比AVL少。
    - Trie树(字典树)
        - 每个节点保存一个字符
        - 根节点不保存字符
        - 每个节点最多有n个子节点(n是所有可能出现字符的个数)
        - 查询的复杂度为O(k)，k为查询字符串长度
- 链表
    - 链表和插入和删除，单向和双向链表都要会
    - 链表的问题考虑多个指针和递归
        - (1) 反向打印链表(递归)
        - (2) 打印倒数第K个节点(前后指针)

        - (3) 链表是否有环(快慢指针)等等。
- 栈和队列
    - 队列和栈的区别?
        - (从实现，应用，自身特点多个方面来阐述，不要只说一个先入先出，先入后出，这个你会别人也会，要展现出你比别人掌握的更深)
        - 线性表：线性表是一种线性结构，它是一个含有n≥0个结点的有限序列，同一个线性表中的数据元素数据类型相同并且满足“一对一”的逻辑关系。
        - “一对一”的逻辑关系指的是对于其中的结点，有且仅有一个开始结点没有前驱但有一个后继结点，有且仅有一个终端结点没有后继但有一个前驱结点，其它的结点都有且仅有一个前驱和一个后继结点。）
        - 这种受限表现在：栈的插入和删除操作只允许在表的尾端进行（在栈中成为“栈顶”），满足“FIFO：First In Last Out”；队列只允许在表尾插入数据元素，在表头删除数据元素,满足“First In First Out”。
        - 栈与队列的相同点：
        - 1.都是**线性结构。**
        - 2.**插入操作都是限定在表尾进行。**
        - 3.都可以通过**顺序结构和链式结构**实现。
        - 4.插入与删除的时间复杂度都是O（1），在空间复杂度上两者也一样。
        - 5.多链栈和多链队列的管理模式可以相同。
        - 栈与队列的不同点：
        - 1.**删除数据元素的位置不同，栈的删除操作在表尾进行，队列的删除操作在表头进行。**
        - 2.应用场景不同；常见栈的应用场景包括**括号问题的求解**，**表达式的转换和求值**，**函数调用和递归实现**，**深度优先搜索遍历**等；常见的队列的应用场景包括计算机系统中各种资源的管理，**消息缓冲器的管理**和**广度优先搜索遍历**等。
        - 3.顺序栈能够实现多栈空间共享，而顺序队列不能。
    - 典型的应用场景
- string
    - 特性
        - 直接赋值
- algorithm
    - 功能
        - sort
            - 自定义比较方法
        - 二分查找
            - 返回迭代器
        - 第k大元素
            - 不用stl
                - 分治
            - stl
                - nth_element(a,a+3,a+5);
        - 堆的操作
            - make_heap(a,a+n)

        - push_heap(a,a+n)
        - pop_heap(a,a+n)
        - 大根堆
        - 直接用数组实现
    - 最大最小绝对值
        - max
        - min
        - abs
    - 交换
        - swap
    - unique
        - unique(a,a+5)-a;
            - 长度
        - 实现离散化
            - 先sort
            - 再去重
            - 最后二分

- 海量数据问题
    - 十亿整数（随机生成，可重复）中前K最大的数
    - 类似问题的解决方法思路：首先哈希将数据分成N个文件，然后对每个文件建立K个元素最小/大堆（根据要求来选择）。最后将文件中剩余的数插入堆中，并维持K个元素的堆。最后将N个堆中的元素合起来分析。可以采用归并的方式来合并。在归并的时候为了提高效率还需要建一个N个元素构成的最大堆，先用N个堆中的最大值填充这个堆，然后就是弹出最大值，指针后移的操作了。当然这种问题在现在的互联网技术中，一般就用map-reduce框架来做了。
    - 大数据排序相同的思路：先哈希（哈希是好处是分布均匀，相同的数在同一个文件中），然后小文件装入内存快排，排序结果输出到文件。最后建堆归并。
- 布隆过滤器
    - 几十亿个数经常要查找某一个数在不在里面，使用布隆过滤器，布隆过滤器的原理。布隆过滤器可能出现误判，怎么保证无误差？