# 排序

快排和归并

# 二分

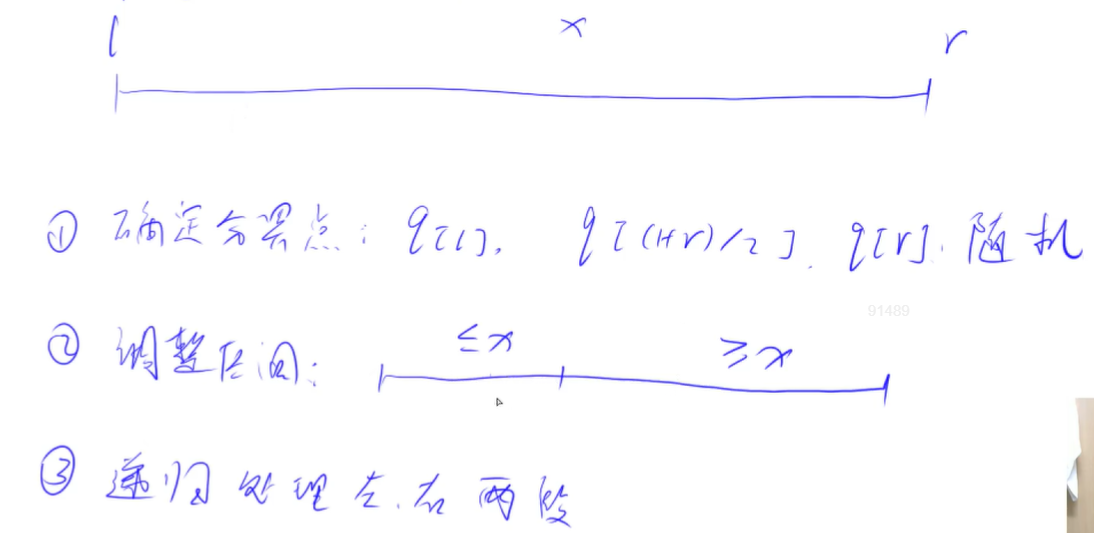
整数和浮点数

快排的主要思想是分治。

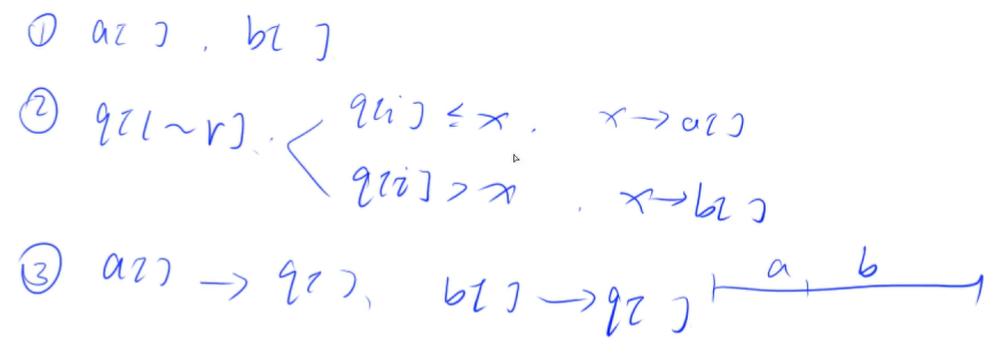
1、确定分界点（随机）

**2、调整范围**

3、递归处理左右两端



暴力做法：开两个新数组，分别放小的和大的：



推荐做法：不需要额外开辟空间，用两个指针，从两头往中间走

# 归并（分治）

1、确定分界点mid

2、递归排序left,right

**3、归并，合二为一**

两个指针分别指向两个数组的最小值，每次拿出二者中的较小值。

**（排序稳定：两个相同的值，交换位置排序后顺序不变则稳定。快排不稳定，归并稳定）**

快排：nlogn（平均，最差n^2） 归并：n

# 高精度（仅限c++，python不吃这一套）

主要四种情况：

1、A+B (10^6

2、A-B

3、A\*a

4、A/a

大整数存储方法：存入数组，第0位存个位比较好（考虑到进位问题），剩下的依此类推。

加减：用一个t表示进位或借位。

除法：注意从高位到低位，输出前需要reverse.

# 前缀和与差分：

挺简单的，没啥可写的。

有些有意思的性质可以看课程。

# 位运算

n的第k位数字：n>>k右移k位后&1.

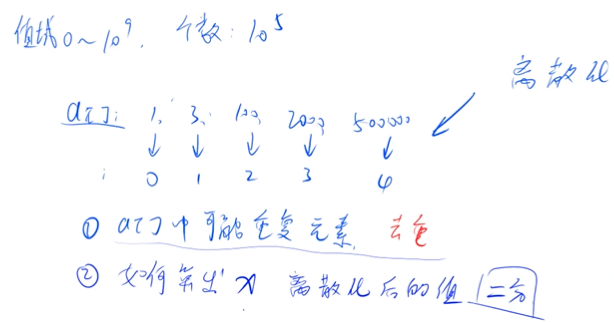
lobit(x):返回x的最后一位1

Eg: x=1010 >>10

x=101000 >>1000

实现：return x&-x

# 离散化（整数）



1、vector去重的方法（直接背就行）：

sort(alls.begin(),alls.end())

alls.erase(unique(alls.begin(),alls.end()),alls.end()) //去掉重复元素

2、二分，离散化

int find(int x){ //找到第一个大于等于x的位置，返回下标+1

int l=0,r=all.size()-1;

while(l<r){

int mid=l+r>>1;

if(alls[mid]>=x) r=mid;

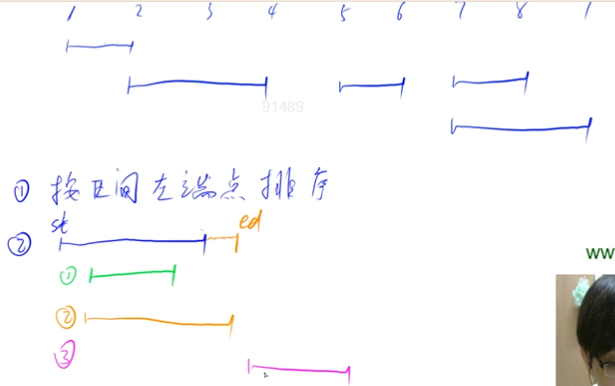
else l=mid+1;

}

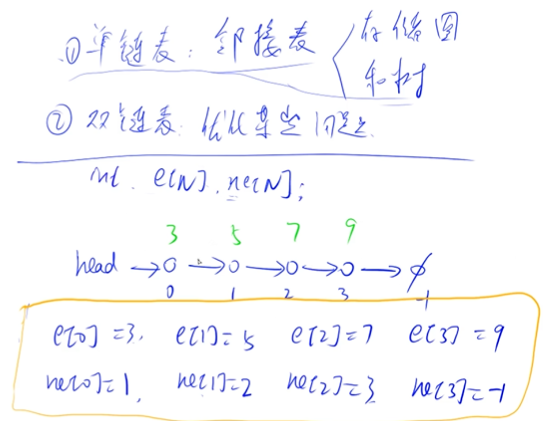
return r+1;

}

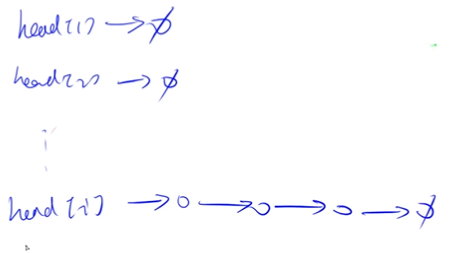
# 区间合并：



# 单链表与邻接表



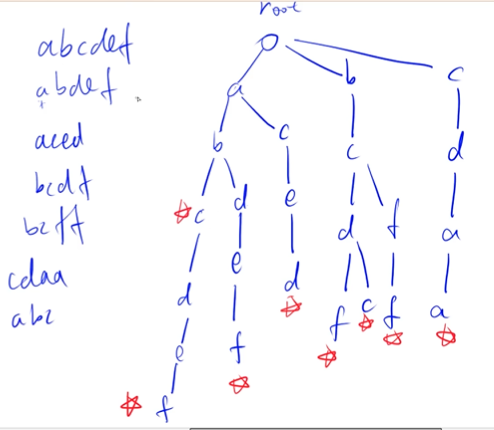
邻接表：n个单链表



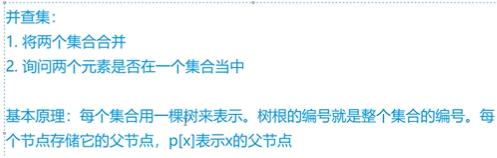
# Trie树

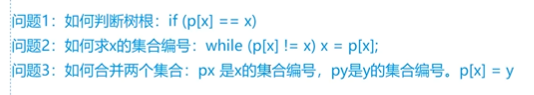
用于快速存储和查找字符串集合的数据结构。

存储如下：



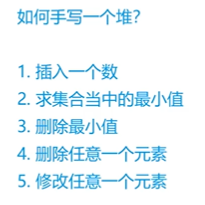
# 并查集





Tips:用字符串读取字母可以避免空格回车的干扰%s

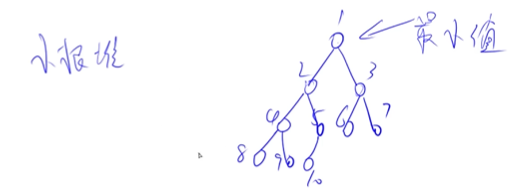
# 堆排序

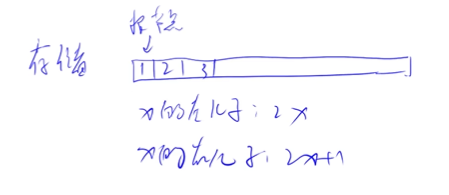


堆是一个完全二叉树。

每个点都小于左右子节点的值。

下标要从1开始。





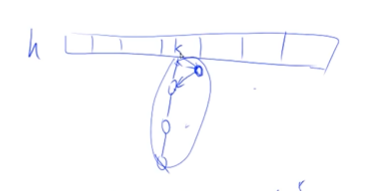
下移：每次和三个点里的最小值交换。

上移：和父节点比较大小。

# 哈希表

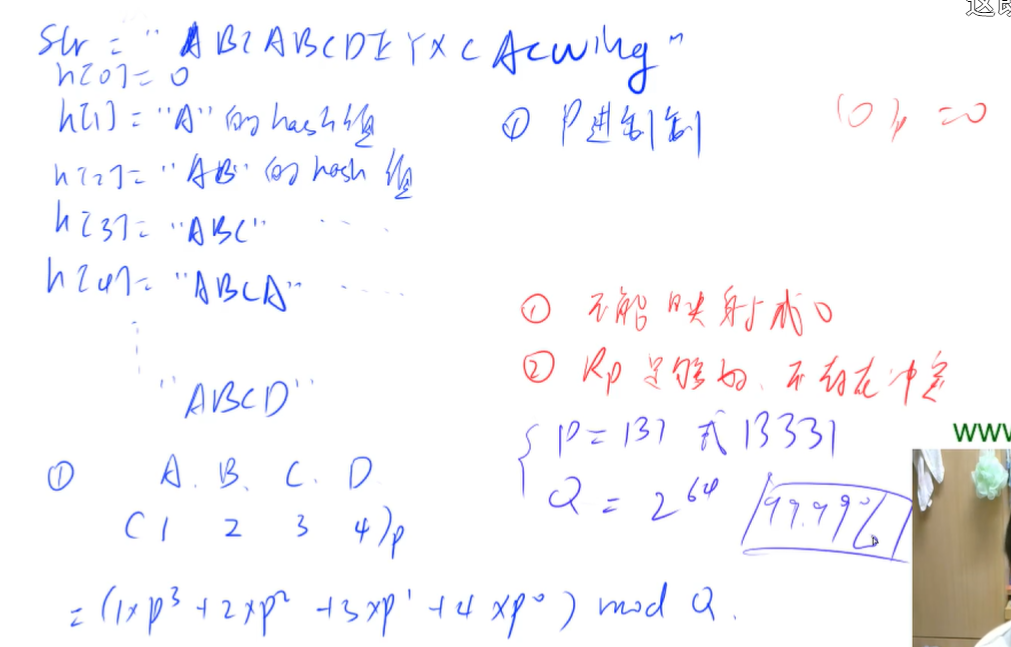
存储结构：开放寻址法和拉链法。

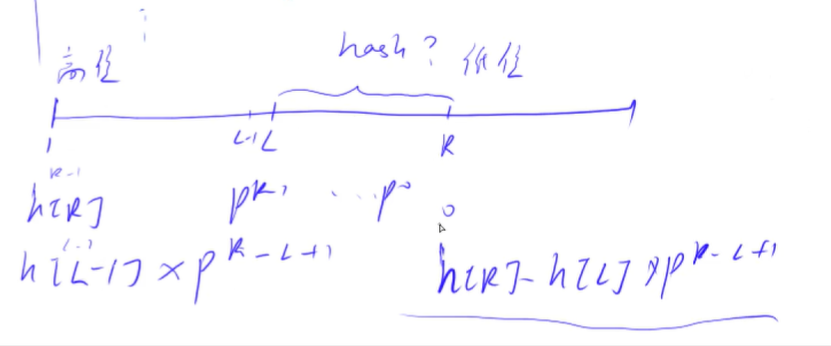
拉链法：



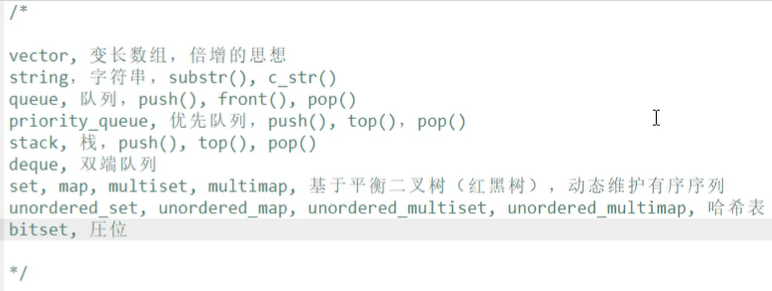
开放寻址法：相当于把拉链法中的多个链表平铺了。

**字符串哈希：**



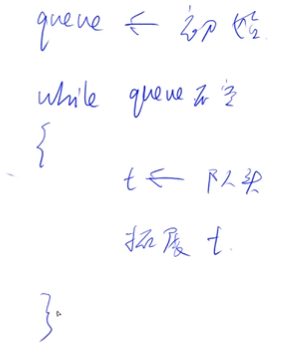


# STL

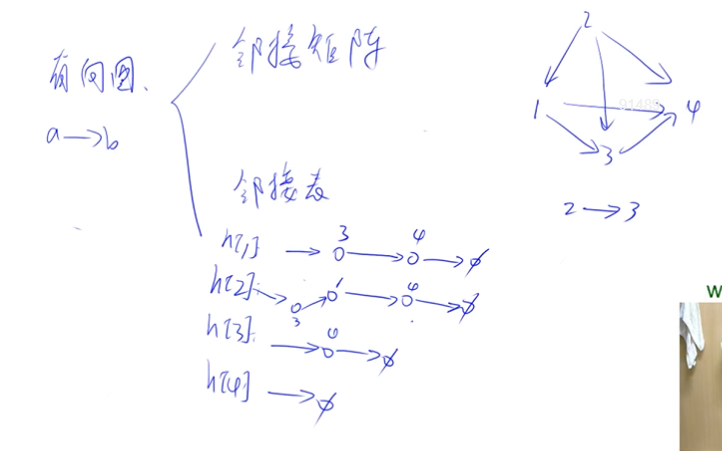


# 搜索

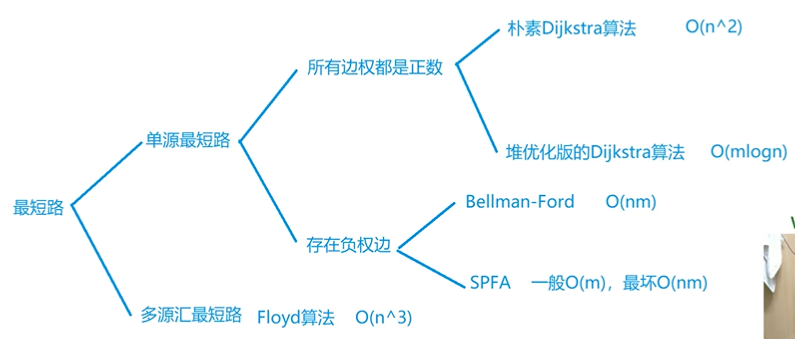
BFS能搜到最短路

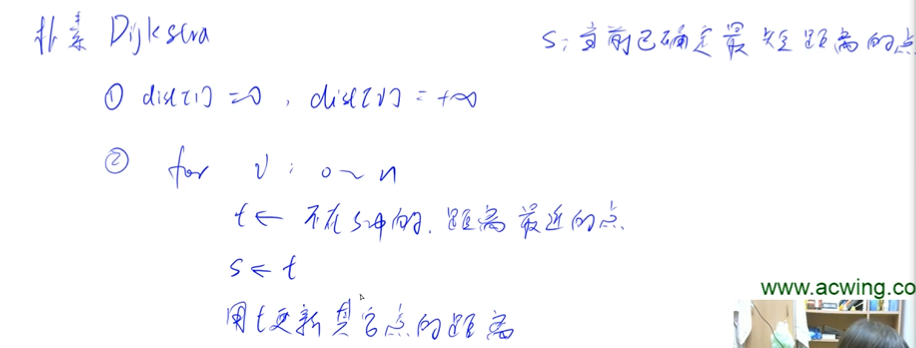


# 树的存储

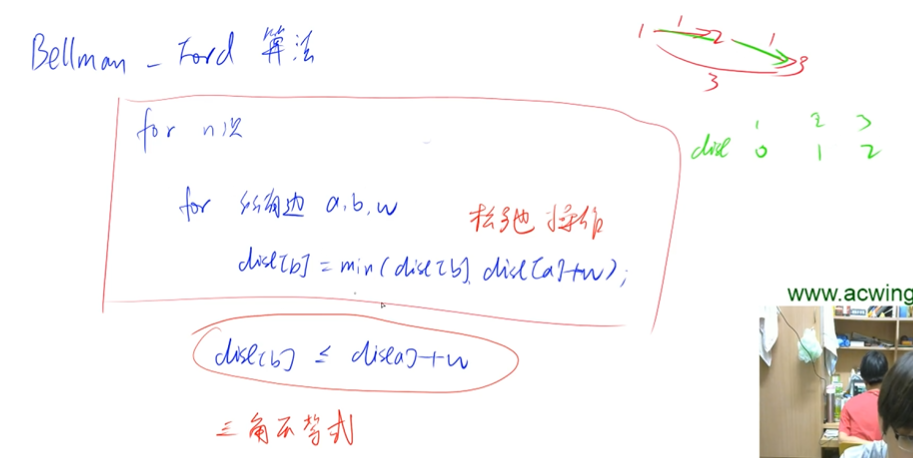


# 最短路问题

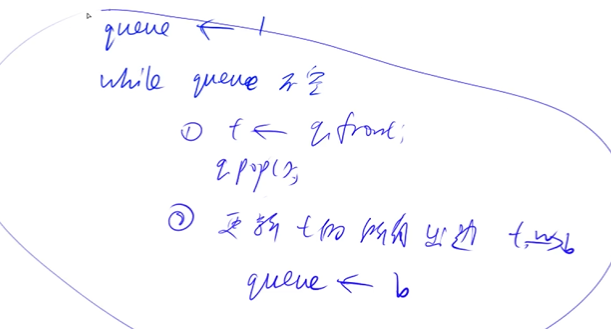




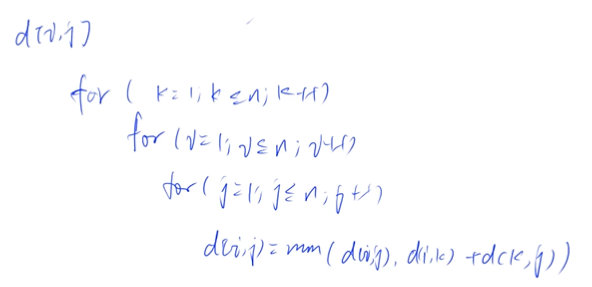
**Bellman-ford算法：**

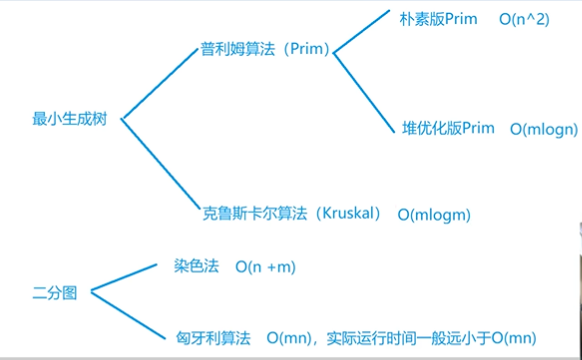


**Spfa:**

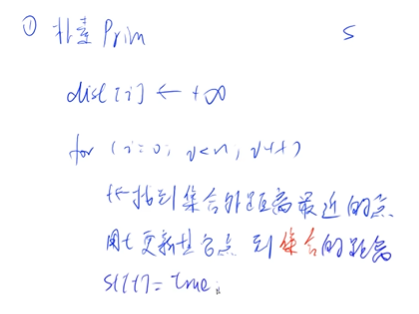


**Floyd:**

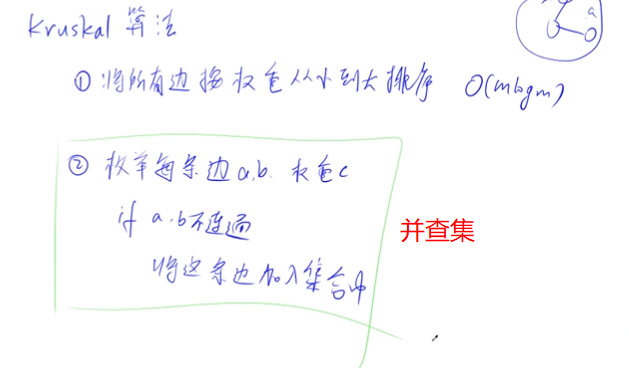




**Prim:**

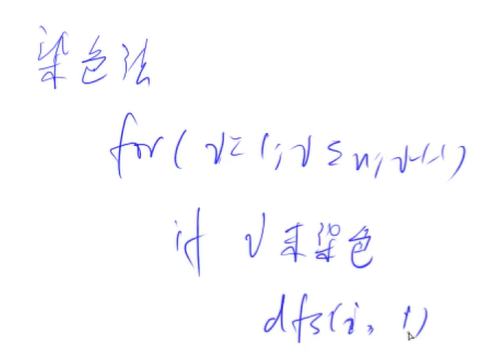


**Kruskal:**

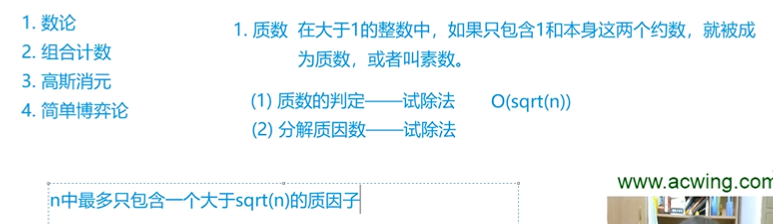


**二分图染色法：**

二分图当且仅当图中不含奇数环



# 数学知识



40.24