Technology Tarena 达内科技

数据结构与算法

C/C++教学体系

Technology Tarena 达内科技

“

个人简介

闵卫
minwei@tarena.com.cn

”

Technology Tarena 达内科技

“

数据结构

”

全程目标

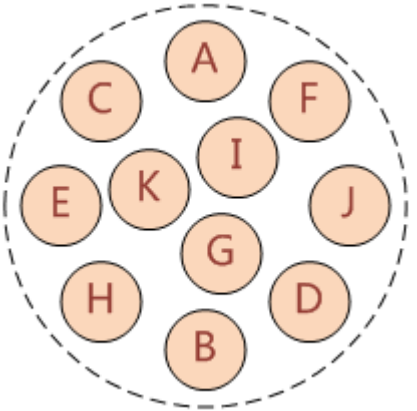
- 数据结构的基本概念
  - 逻辑结构
  - 物理结构
  - 运算结构
- 数据结构的基本实现
  - 堆栈
  - 队列
  - 链表
  - 二叉树

数据结构的基本概念

- 数据结构是相互之间存在一种或多种特定关系的数据的集合
- 数据结构是计算机存储、组织数据的方式
- 数据结构的选择直接影响计算机程序的运行效率(时间复杂度)和存储效率(空间复杂度)
- 计算机程序设计=算法+数据结构
- 数据结构的三个层次
  - 抽象层——逻辑结构
  - 结构层——物理结构
  - 实现层——运算结构

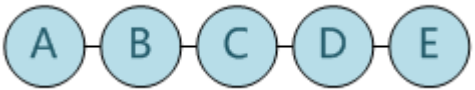
逻辑结构

- 集合结构(集)
  - 结构中的数据元素除了同属于一个集合外没有其它关系



逻辑结构

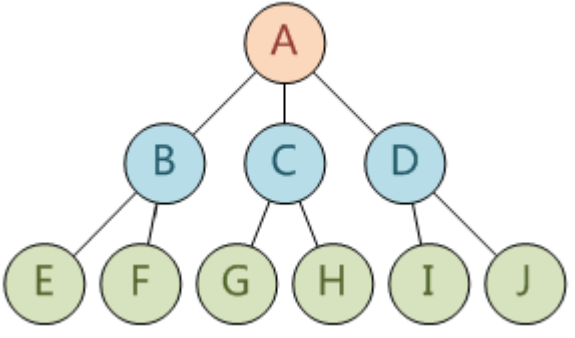
- 线性结构(表)
  - 结构中的数据元素具有一对一的前后关系



逻辑结构

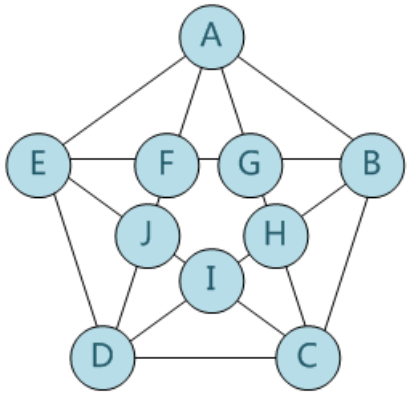
- 树型结构(树)
  - 结构中的数据元素具有一对多的父子关系

```
/
+-- bin
|   +-- alsaunmute
|   +-- arch
|   +-- zcat
+-- boot
|   +-- efi
|   |   +-- EFI
|   +-- grub
|   |   +-- splash.xpm.gz
|   +-- vmlinuz-3.6.7-4.fc16.i686
+-- dev
|   +-- autofs
|   +-- console
|   +-- zero
+-- etc
    +-- abrt
    |   +-- plugins
    +-- tmp
    +-- yp
```



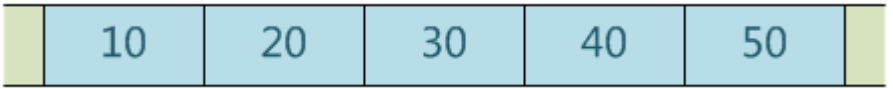
逻辑结构

- 网状结构(图)
  - 结构中的数据元素具有多对多的交叉映射关系



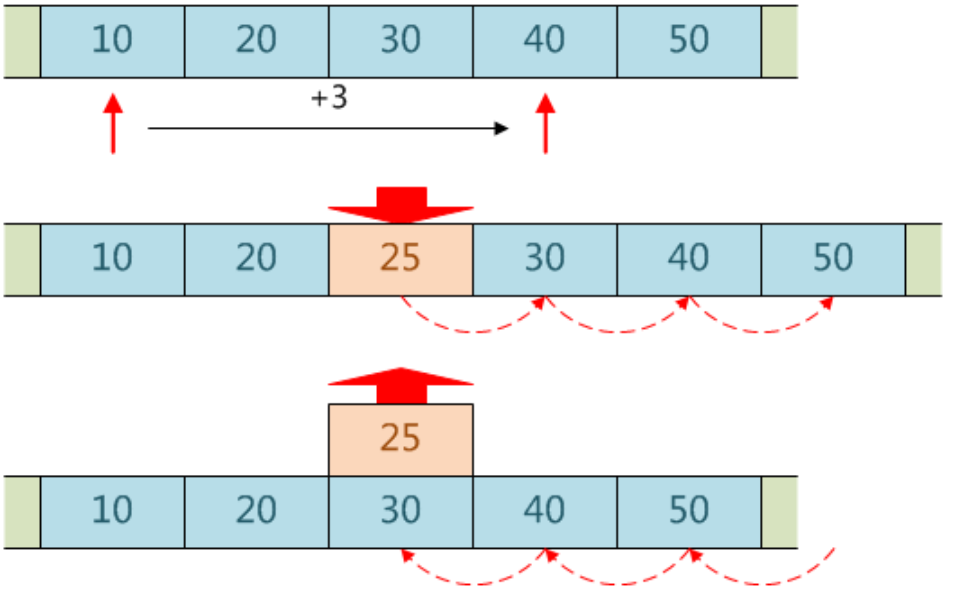
物理结构

- 顺序结构
  - 结构中的数据元素存放在一段连续的地址空间中



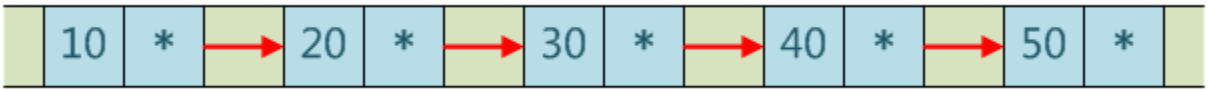
物理结构

- 顺序结构
  - 随机访问方便，空间利用率低，插入删除不方便



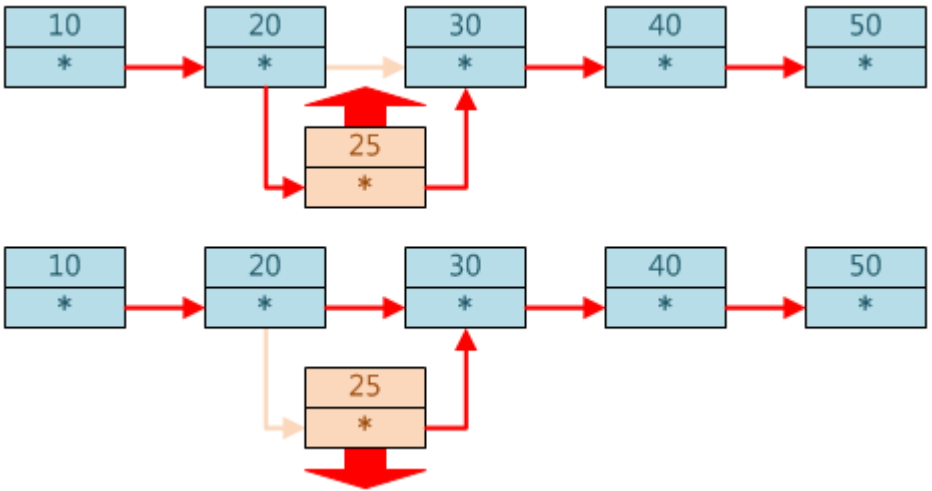
物理结构

- 链式结构
  - 结构中的数据元素存放在彼此独立的地址空间中
  - 每个独立的地址空间称为节点
  - 节点除保存数据外，还需要保存相关节点的地址



物理结构

- 链式结构
  - 插入删除方便，空间利用率高，随机访问不方便



逻辑结构与物理结构的关系

数据结构	表	树	图
顺序	顺序表(数组)	顺序树	链表数组
链式	链式表(链表)	链式树	

- 每种逻辑结构采用何种物理结构实现，并没有一定之规，通常根据实现的难易程度，以及在时间和空间复杂度方面的要求，选择最适合的物理结构，亦不排除复合多种物理结构实现一种逻辑结构的可能

运算结构

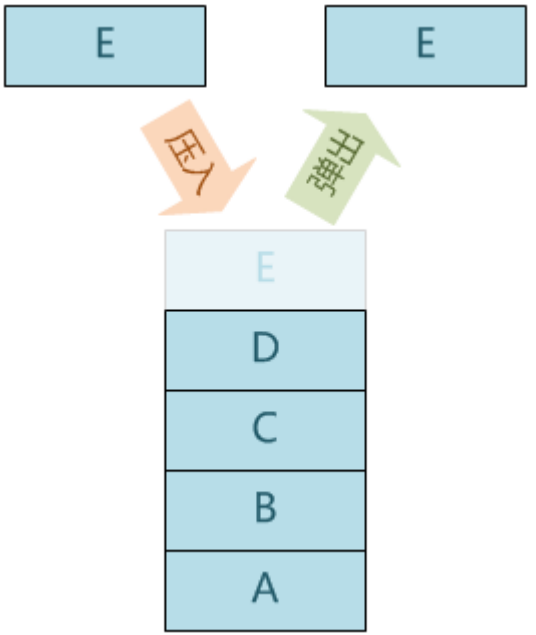
- 创建与销毁
  - 分配资源、建立结构、释放资源
- 插入与删除
  - 增加、减少数据元素
- 获取与修改
  - 遍历、迭代、随机访问
- 排序与查找
  - 算法应用

数据结构的基本实现

- 堆栈
  - 基于顺序表的实现
  - 基于链式表的实现
- 队列
  - 基于顺序表的实现
  - 基于链式表的实现
- 链表
  - 双向线性链表的实现
- 二叉树
  - 有序二叉树(二叉搜索树)的实现

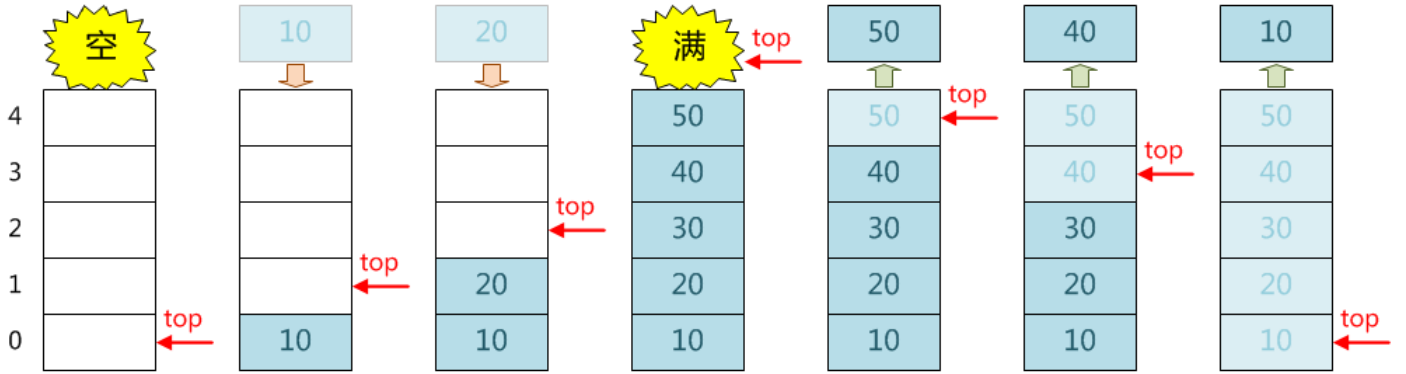
堆栈

- 后进(压入/push)先出(弹出/pop)



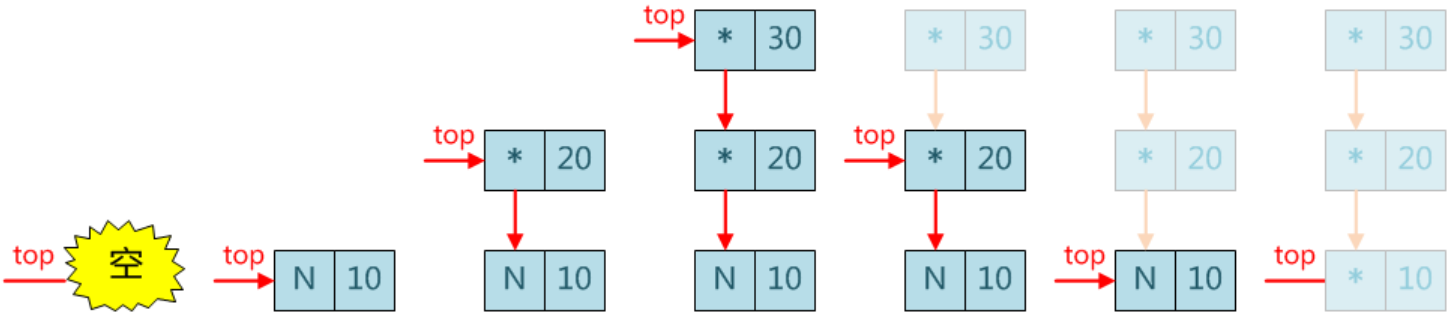
实现基于顺序表的堆栈

- 初始化空间、栈顶指针、判空判满



实现基于链式表的堆栈

- 动态分配、栈顶指针、注意判空

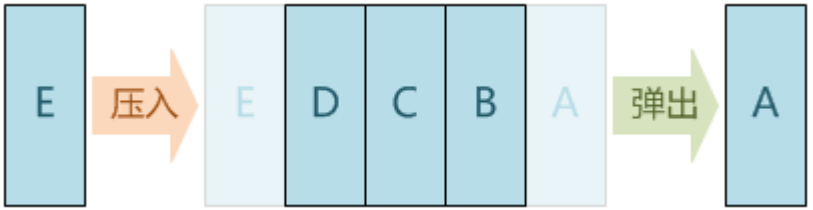


练习时间

输入十进制整数和进制数，利用堆栈以指定进制格式打印该十进制数

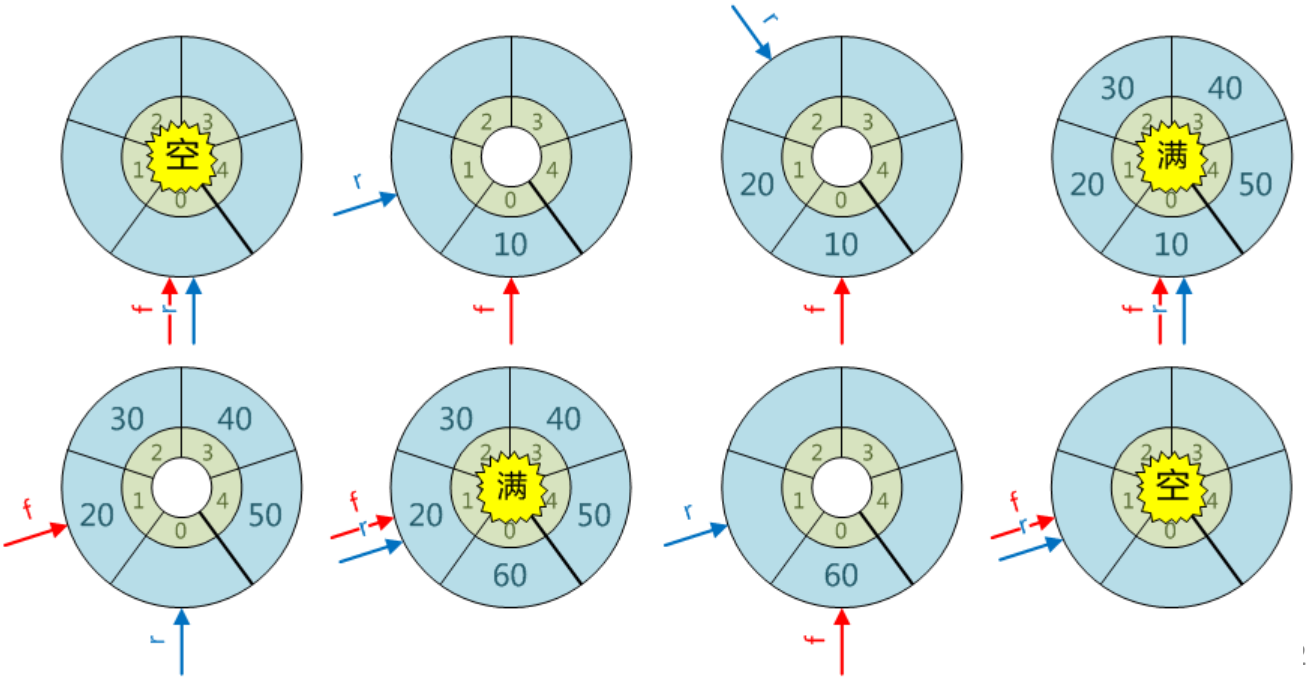
队列

- 先进(压入/push)先出(弹出/pop)



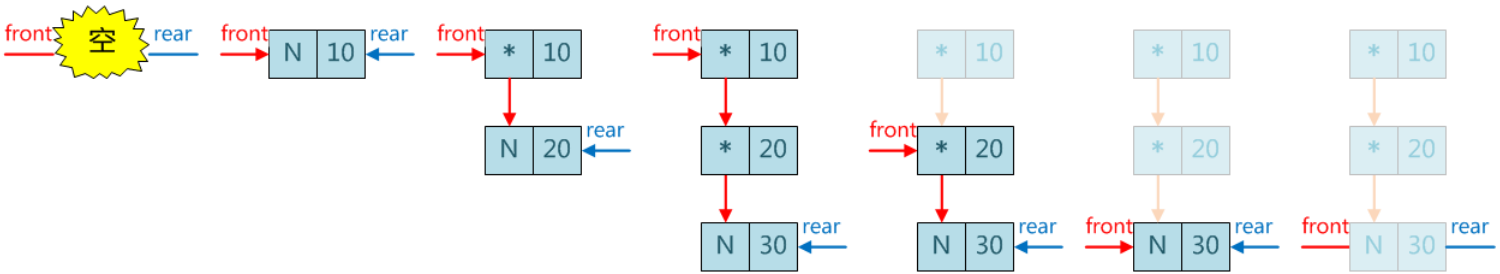
实现基于顺序表的队列

- 初始化空间、前弹后压、循环使用、判空判满



实现基于链式表的队列

- 动态分配、前后指针、注意判空



Technology Tarena 达内科技

练习时间

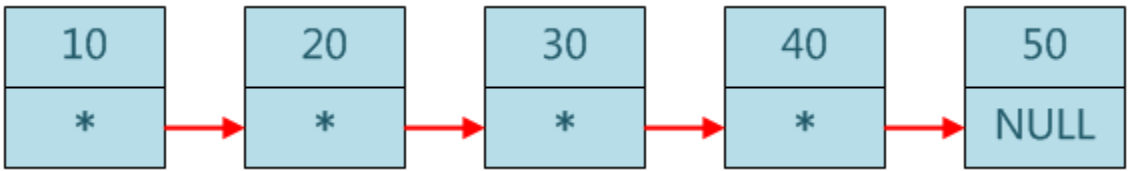
利用堆栈实现队列

链表

- 地址不连续的节点序列，彼此通过指针相互连接
- 根据不同的结构特征，将链表分为：
  - 单向线性链表
  - 单向循环链表
  - 双向线性链表
  - 双线循环链表
  - 数组链表
  - 链表数组
  - 二维链表

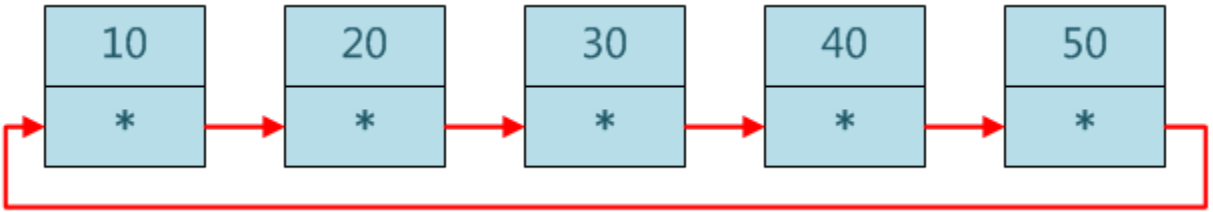
链表

- 单向线性链表



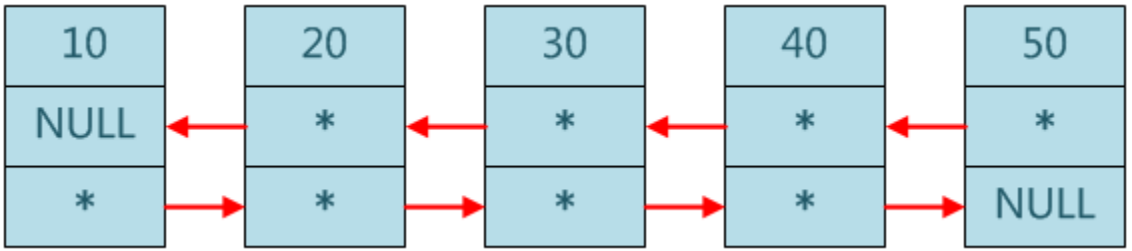
链表

- 单向循环链表



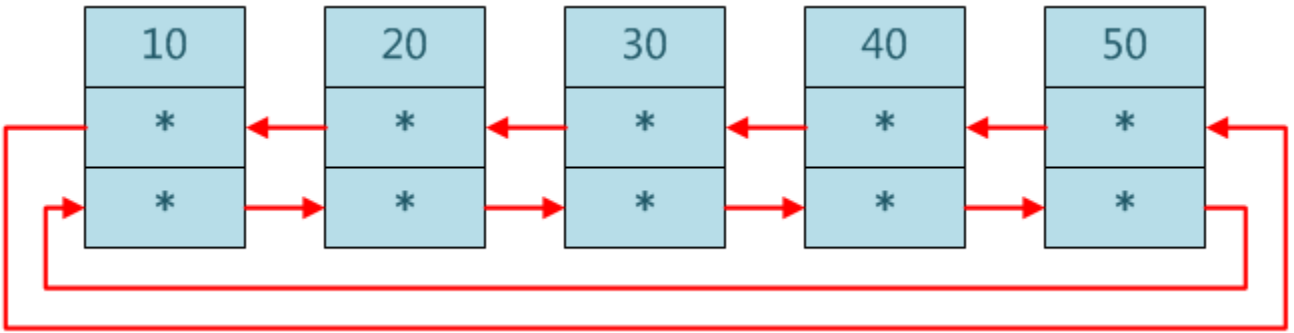
链表

- 双向线性链表



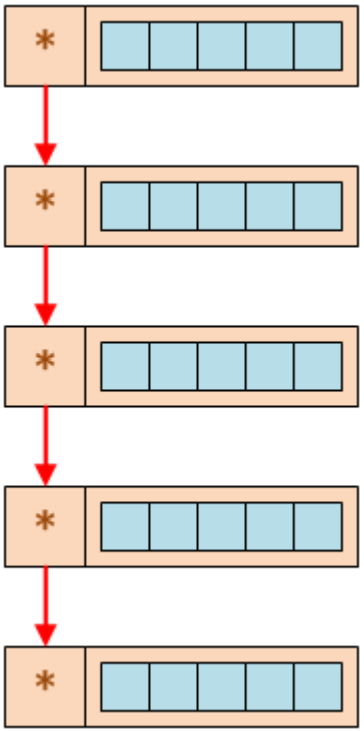
链表

- 双向循环链表



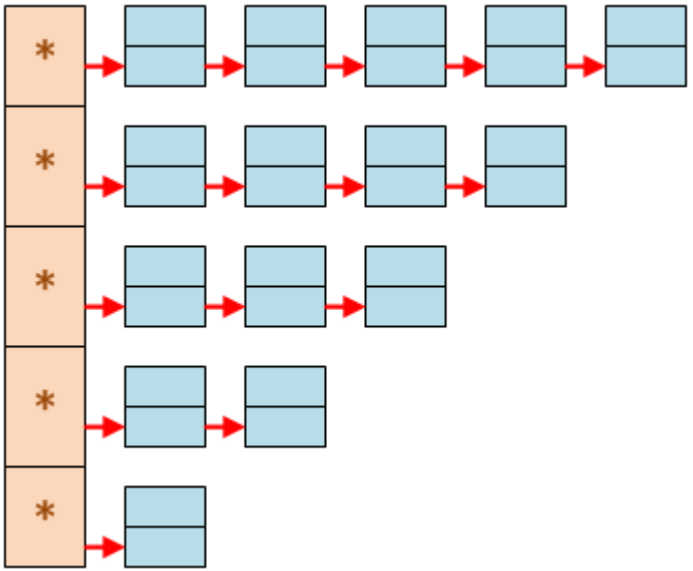
链表

- 数组链表



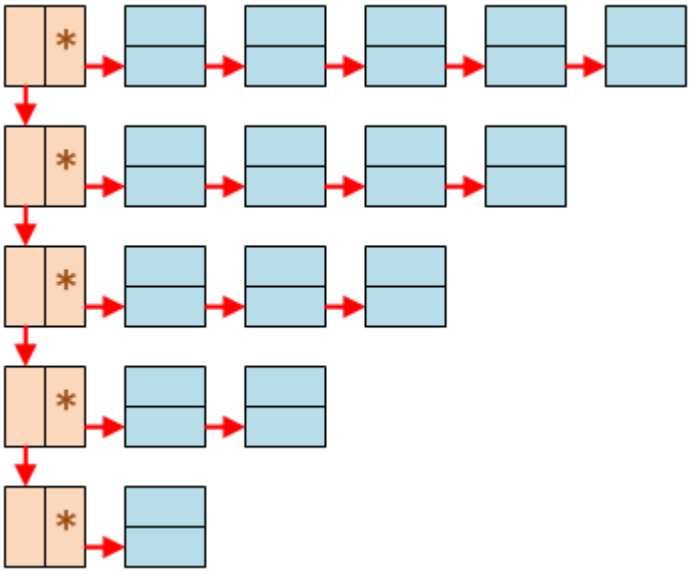
链表

- 链表数组



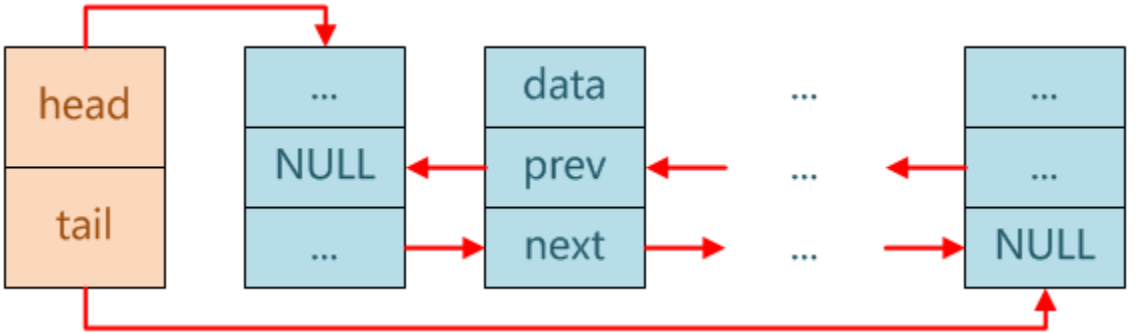
链表

- 二维链表



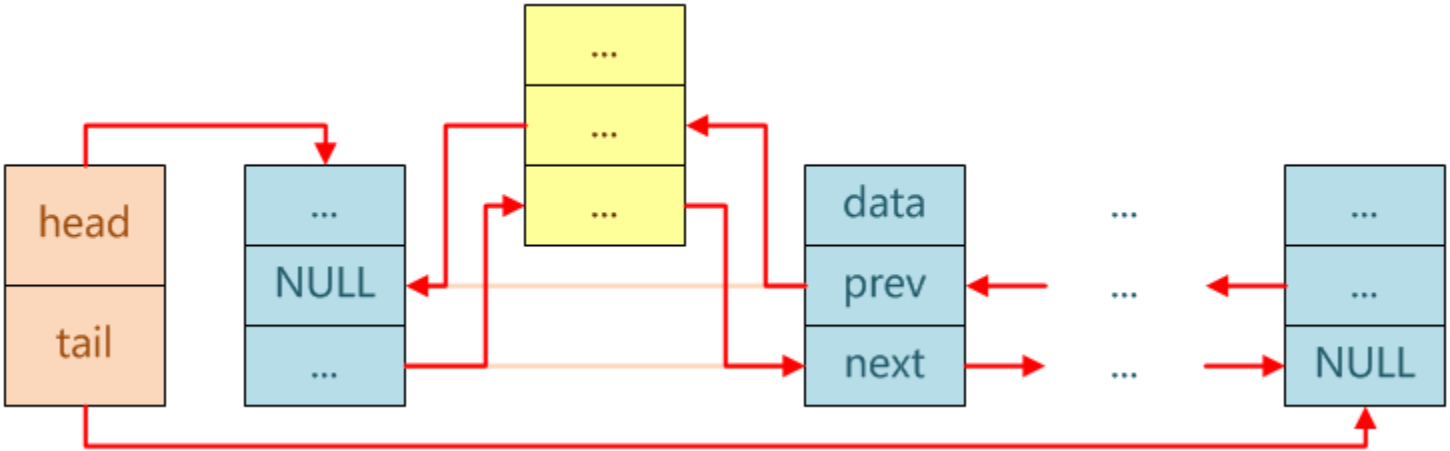
实现双向线性链表

- 结构模型



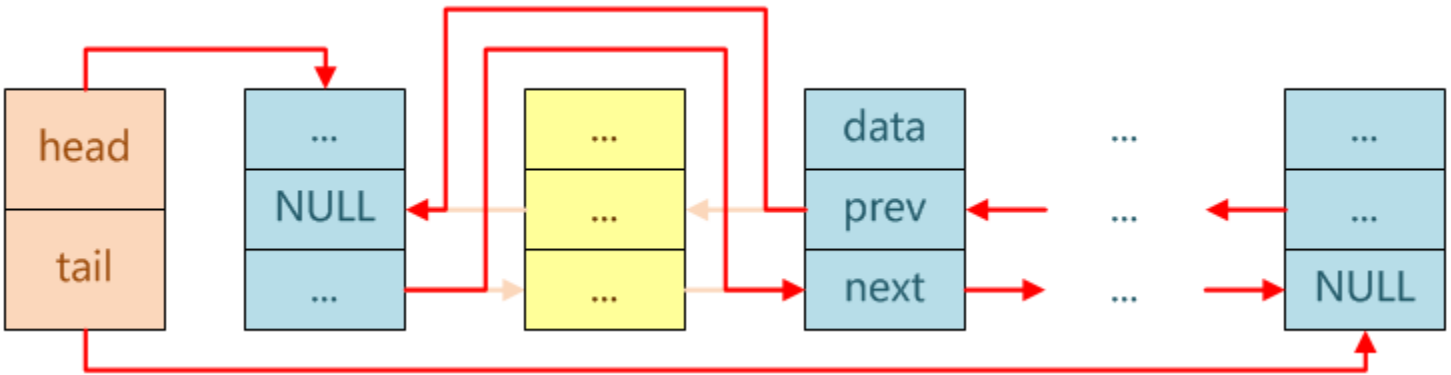
实现双向线性链表

- 插入节点



实现双向线性链表

- 删除节点



练习时间

单向链表

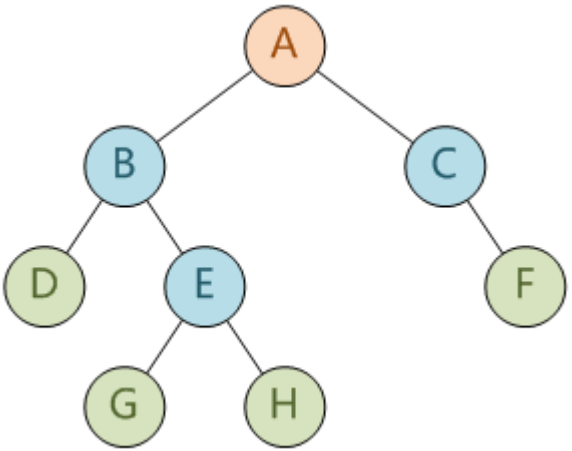
1. 实现list_append()、list_size()、list_print()和list_rprint()函数，分别用于单向链表的追加、测长、正向打印和反向打印
2. 实现list_reverse()函数，用于将单向链表逆转
3. 实现list_middle()函数，用于获取单向链表的中间值，其平均时间复杂度不得超过O(N)级

二叉树

- 树形结构的最简模型，每个节点最多有两个子节点
- 每个子节点有且仅有一个父节点，整棵树只有一个根节点
- 具有递归的结构特征，用递归的方法处理，可以简化算法
- 三种遍历序
  - 前序遍历：D-L-R
  - 中序遍历：L-D-R
  - 后序遍历：L-R-D

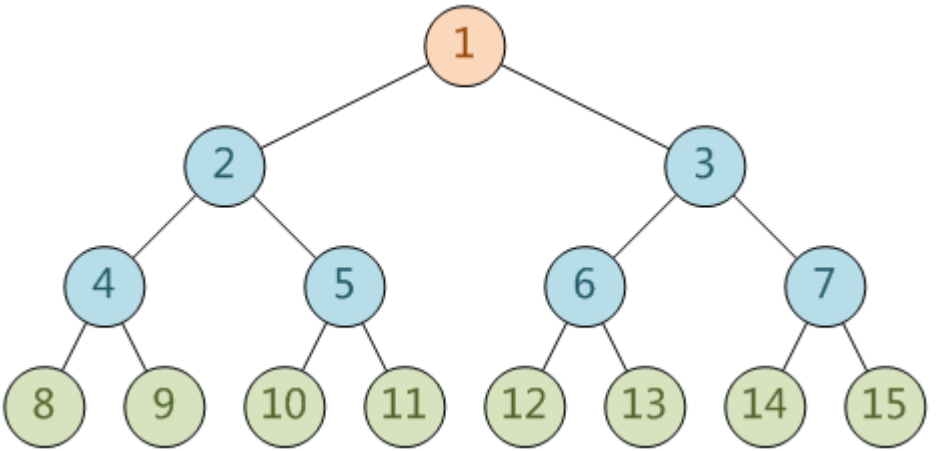
二叉树

- 二叉树的一般形式
  - 根节点、枝节点和叶节点
  - 父节点和子节点
  - 左子节点和右子节点
  - 左子树和右子树
  - 大小和高度(深度)



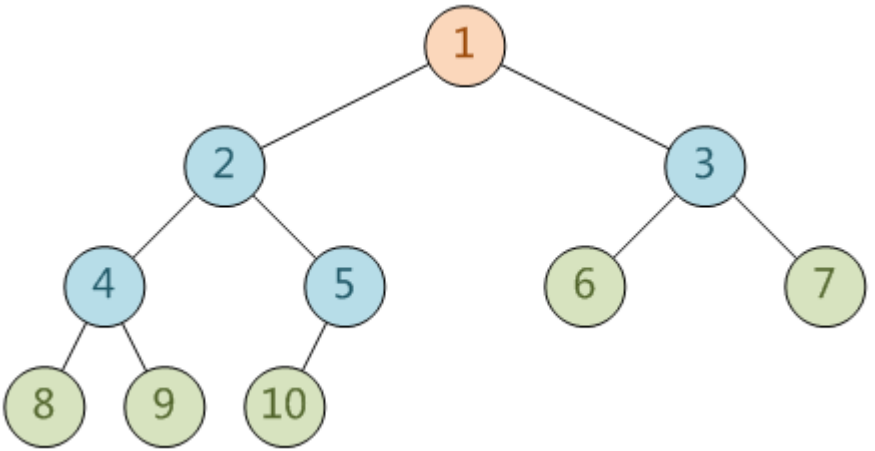
二叉树

- 满二叉树
  - 每层节点数均达到最大值
  - 所有枝节点均有左右子树



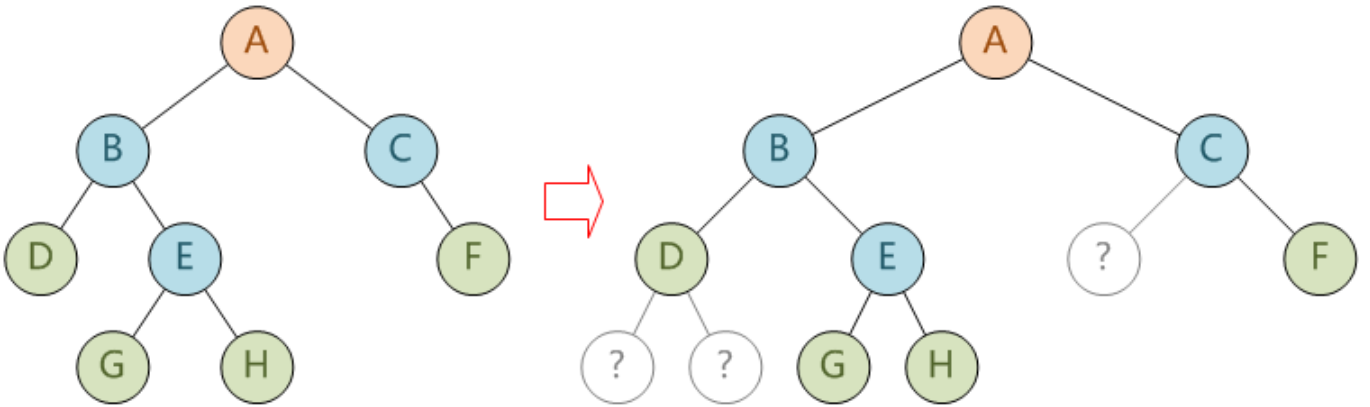
二叉树

- 完全二叉树
  - 除最下层外，各层节点数均达到最大值
  - 最下层的节点都连续集中在左边



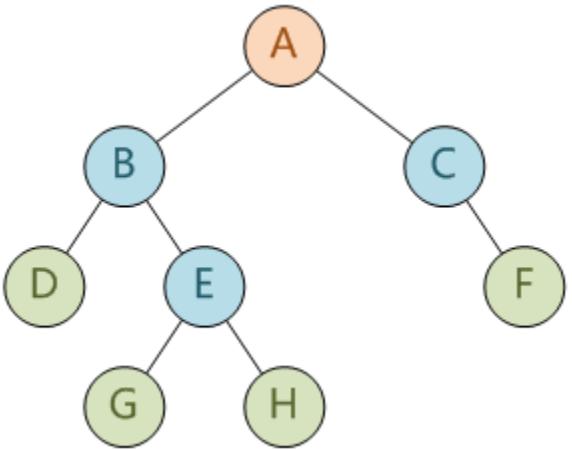
二叉树的存储结构

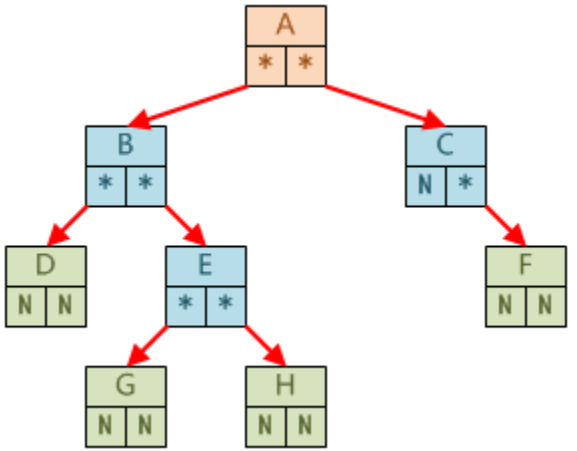
- 顺序存储
  - 从上到下、从左到右，依次存放
  - 非完全二叉树需用虚节点补成完全二叉树



二叉树的存储结构

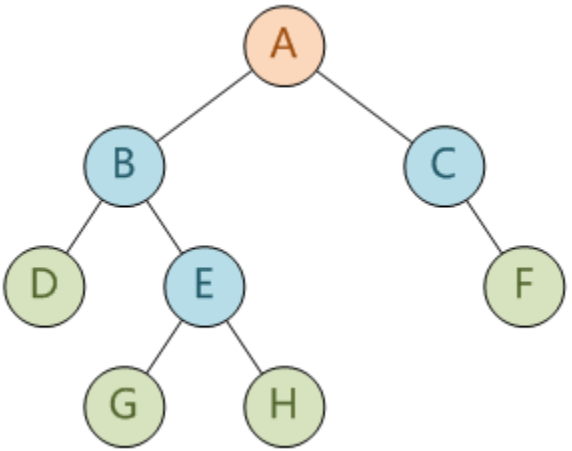
- 链式存储
  - 二叉链表，每个节点包括三个域，一个数据域和两个分别指向其左右子节点的指针域

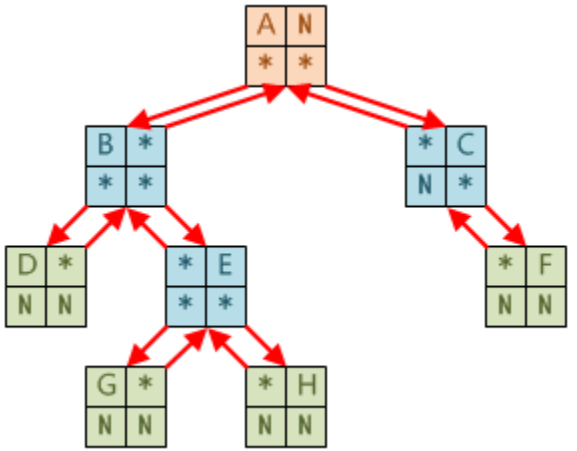




二叉树的存储结构

- 链式存储
  - 三叉链表，每个节点包括四个域，一个数据域、两个分别指向其左右子节点的指针域和一个指向其父节点的指针域



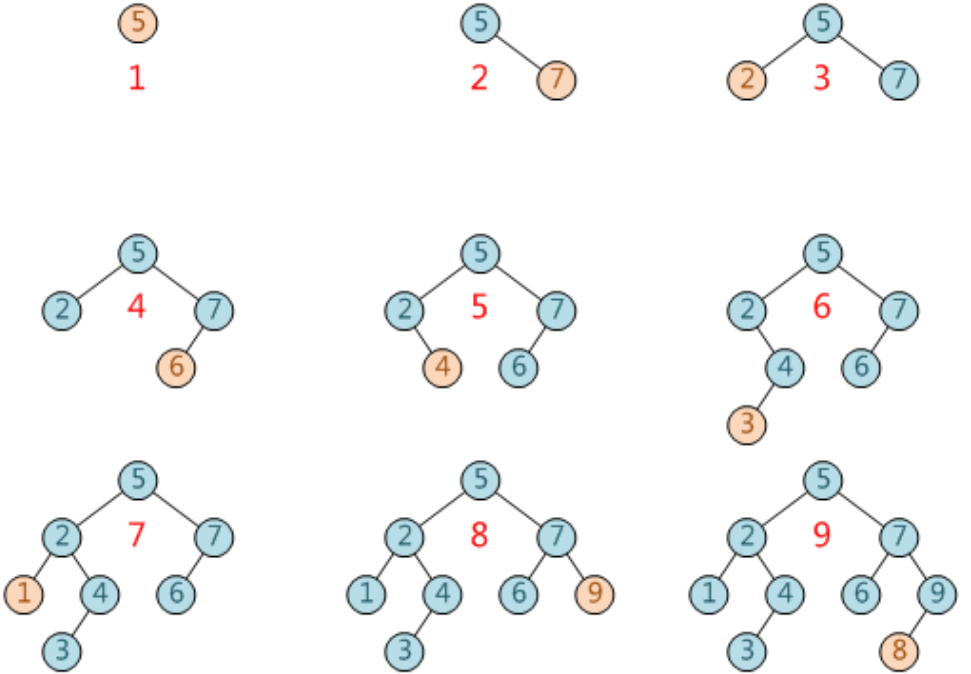


实现有序二叉树

- 有序二叉树亦称二叉搜索树，若非空树则满足：
  - 若左子树非空，则左子树上所有节点的值均小于等于根节点的值
  - 若右子树非空，则右子树上所有节点的值均大于等于根节点的值
  - 左右子树亦分别为有序二叉树
- 基于有序二叉树的排序和查找，可获得O(logN)级的平均时间复杂度

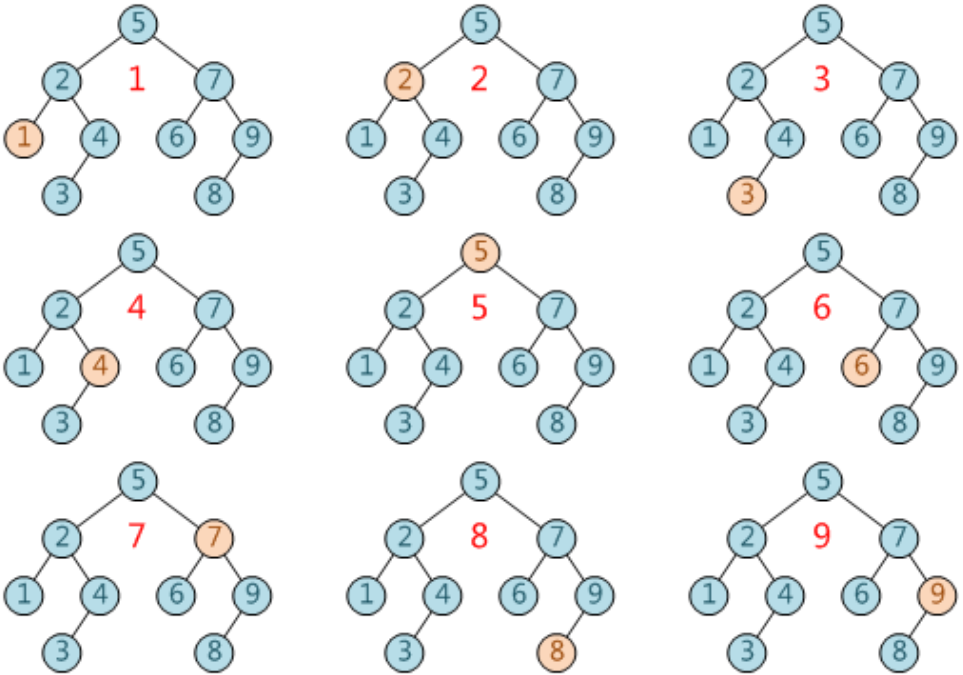
实现有序二叉树

- 构建过程



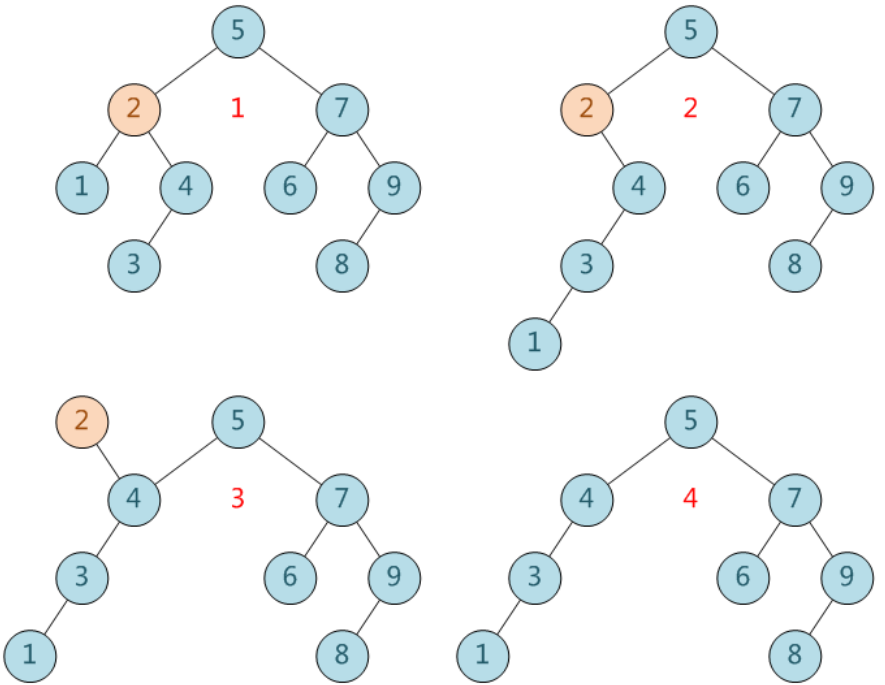
实现有序二叉树

- 中序遍历



实现有序二叉树

- 删除节点



“

算法

”

全程目标

- 排序算法
  - 冒泡排序
  - 插入排序
  - 选择排序
  - 快速排序
  - 归并排序
- 查找算法
  - 线性查找
  - 二分查找

冒泡排序

第一趟

30 20 50 40 10

20 30 50 40 10

20 30 50 40 10

20 30 40 50 10

20 30 40 10 50

第二趟

20 30 40 10 50

20 30 40 10 50

20 30 40 10 50

20 30 10 40 50

第三趟

20 30 10 40 50

20 30 10 40 50

20 10 30 40 50

第四趟

20 10 30 40 50

10 20 30 40 50

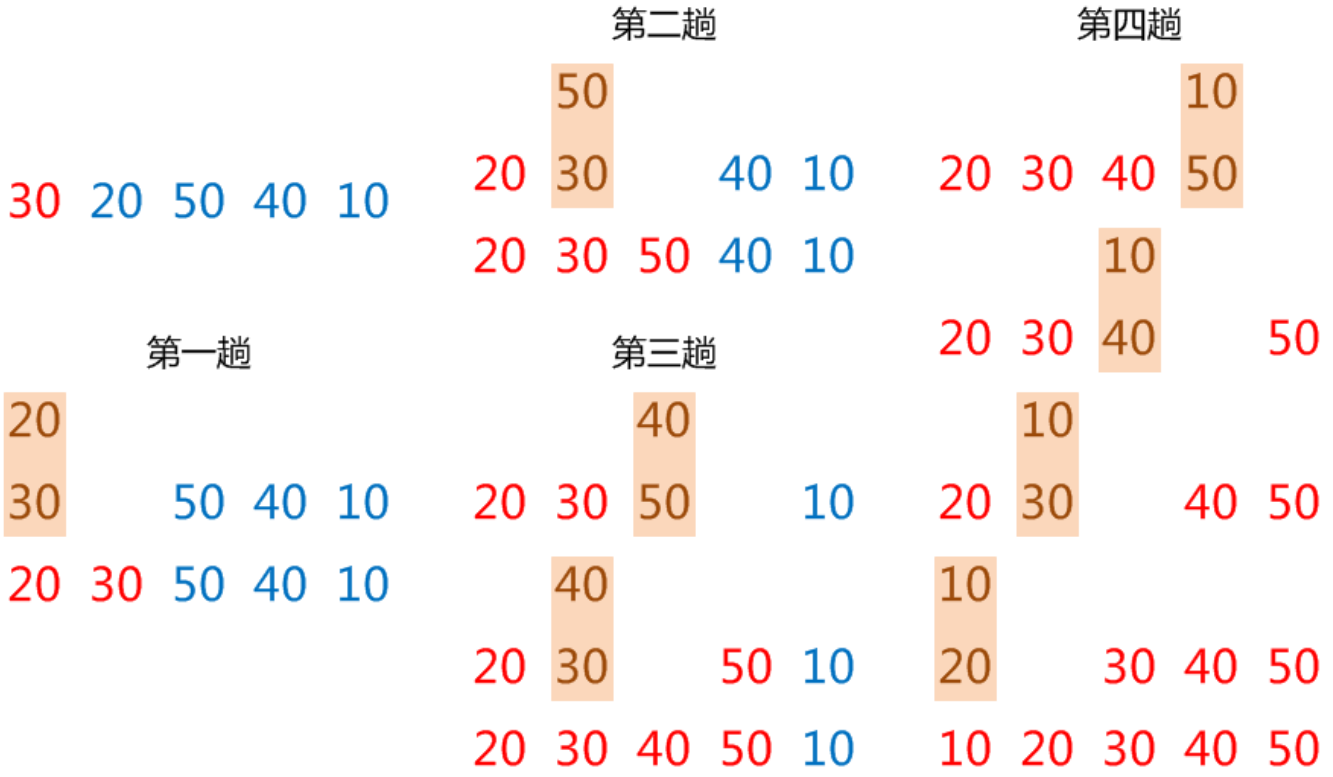
结果

10 20 30 40 50

冒泡排序

- 算法
  1. 相邻元素两两比较，前者大于后者，彼此交换
  2. 从第一对到最后一对，最大的元素沉降到最后
  3. 针对未排序部分，重复以上步骤，沉降次大值
  4. 每次扫描越来越少的元素，直至不再发生交换
- 评价
  - 平均时间复杂度：$O(N^2)$
  - 稳定排序
  - 对数据的有序性非常敏感

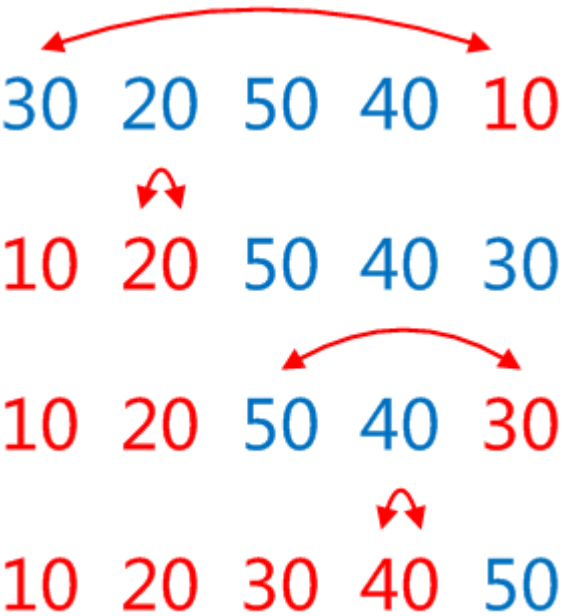
插入排序



插入排序

- 算法
  1. 首元素自然有序
  2. 取出下一个元素，对已排序序列，从后向前扫描
  3. 大于被取出元素者后移
  4. 小于等于被取出元素者，将被取出元素插入其后
  5. 重复步骤2，直至处理完所有元素
- 评价
  - 平均时间复杂度：$O(N^2)$
  - 稳定排序
  - 对数据的有序性非常敏感
  - 不交换只移动，优于冒泡排序

选择排序



选择排序

- 算法
  1. 在整个序列中寻找最小元素，与首元素交换
  2. 在剩余序列中寻找最小元素，与次元素交换
  3. 以此类推，直到剩余序列中仅包含一个元素
- 评价
  - 平均时间复杂度：$O(N^2)$
  - 非稳定排序
  - 对数据的有序性不敏感
  - 交换次数少，优于冒泡排序

快速排序

10 80 30 60 50 40 70 20 90

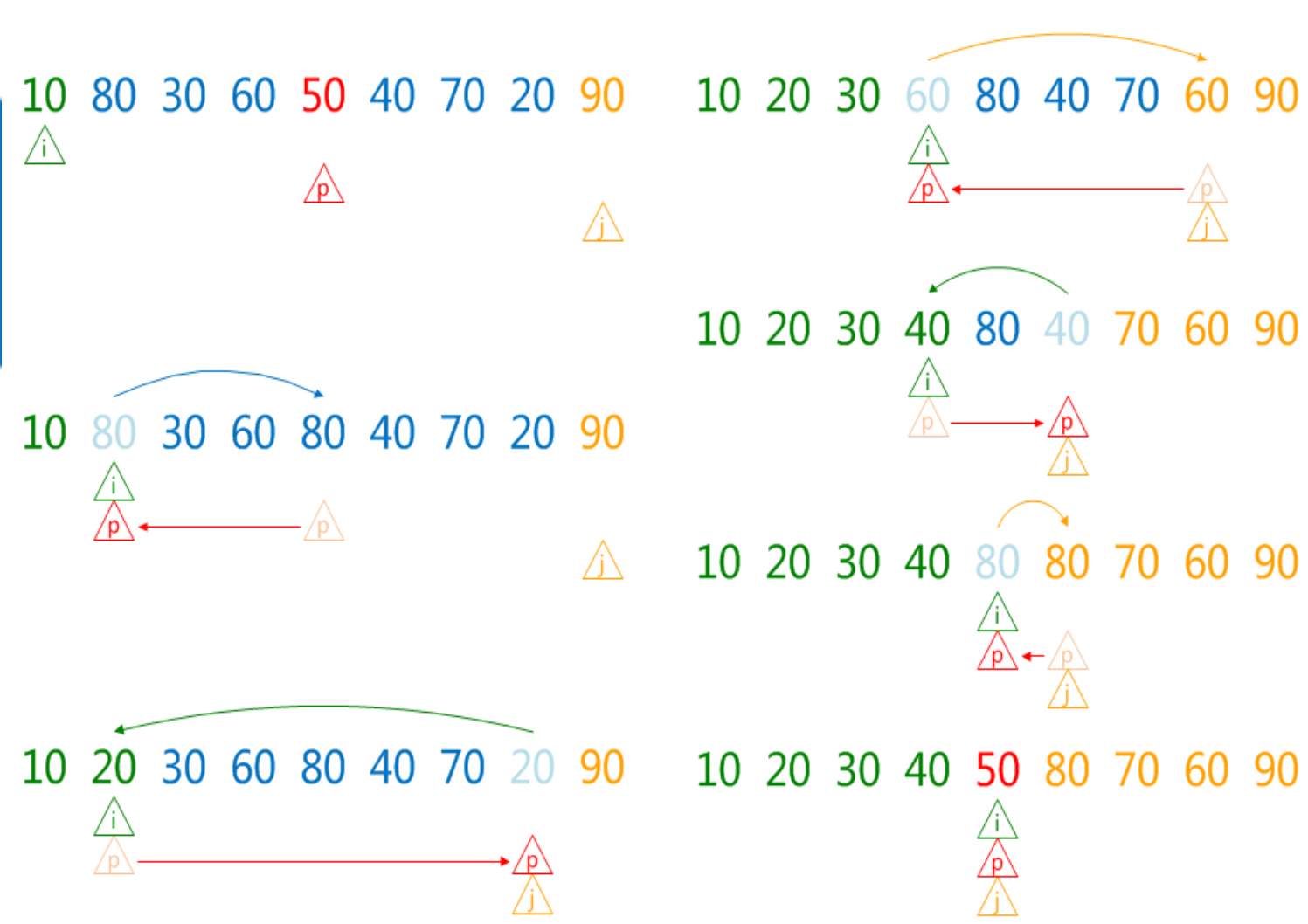
10 20 30 40 50 80 70 60 90

10 20 30 40 50 60 70 80 90

10 20 30 40 50 60 70 80 90

快速排序

- 算法
  1. 从待排序序列中任意挑选一个元素，作为基准
  2. 将所有小于基准的元素放在基准之前，大于基准的元素放在基准之后，等于基准的元素放在基准之前或之后，这个过程称为分组
  3. 以递归的方式，分别对基准之前和基准之后的分组继续进行分组，直到每个分组内的元素个数不多于1个——自然有序——为止


10 80 30 60 50 40 70 20 90
i
p
j
10 20 30 60 80 40 70 60 90
i
p
p
j
10 80 30 60 80 40 70 20 90
i
p
p
j
10 20 30 40 80 40 70 60 90
i
p
p
j
10 20 30 60 80 40 70 20 90
i
p
p
j
10 20 30 40 80 80 70 60 90
i
p
p
j
10 20 30 40 50 80 70 60 90
i
p
j

快速排序

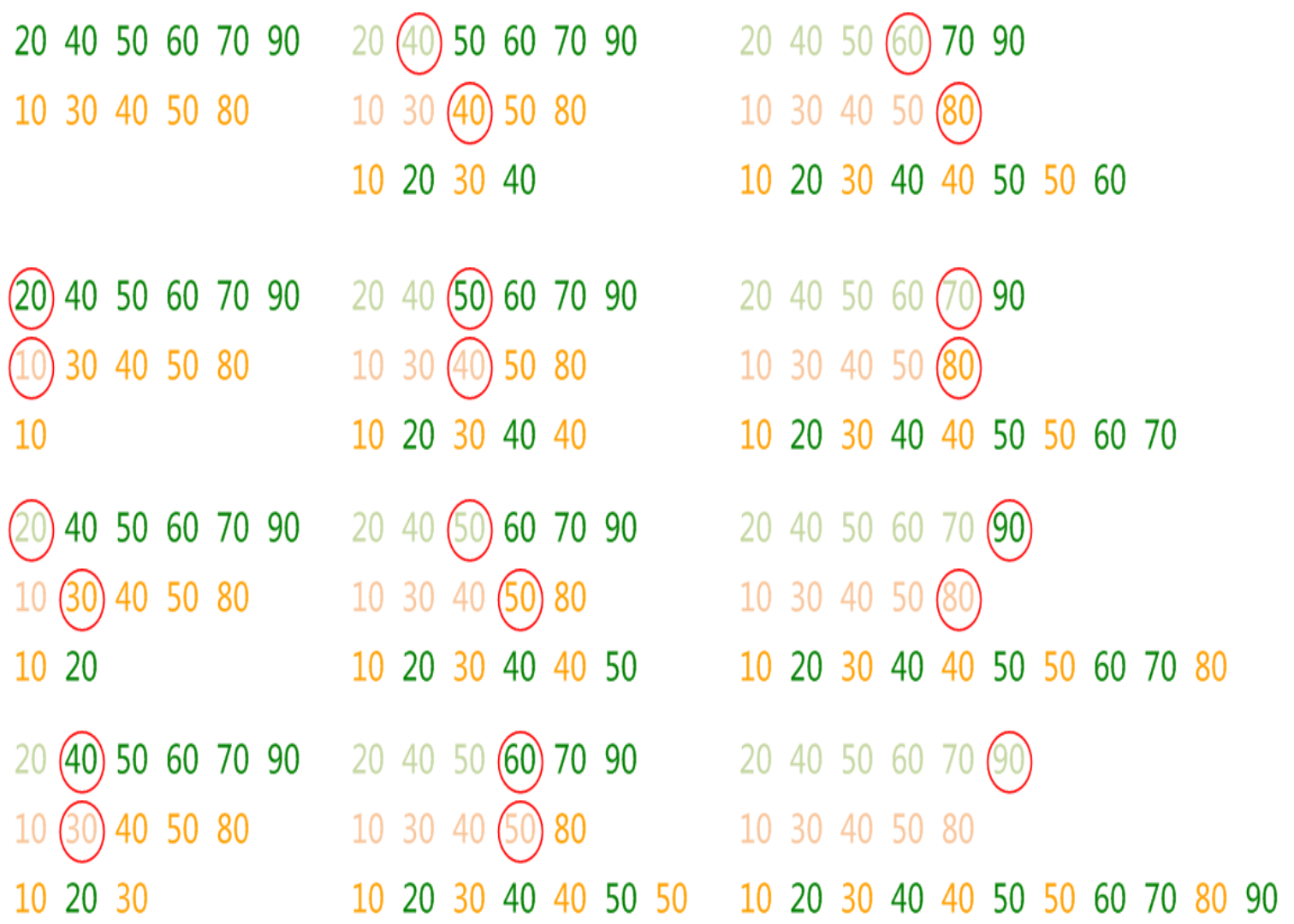
- 就地分组
  - 在不额外分配内存空间的前提下，实现以基准为中心的分组
- 评价
  - 平均时间复杂度：O(NlogN)
  - 非稳定排序
  - 若每次都能均匀分组，则排序速度最快

练习时间

基于快速排序算法，实现类似标准C库qsort函数的功能

归并排序

- 合并
  1. 分配合并序列，其大小为两个有序序列大小之和
  2. 设定两个指针，分别指向两个有序序列的首元素
  3. 比较指针目标，较小者进入合并序列，指针后移
  4. 重复步骤3，直到某一指针到达序列末尾
  5. 将另一序列的剩余元素直接复制到合并序列末尾


20 40 50 60 70 90
10 30 40 50 80
20 40 50 60 70 90
10 30 40 50 80
10 20 30 40
20 40 50 60 70 90
10 30 40 50 80
10 20 30 40 40 50 50 60
20 40 50 60 70 90
10 30 40 50 80
10
20 40 50 60 70 90
10 30 40 50 80
10 20 30 40 40
20 40 50 60 70 90
10 30 40 50 80
10 20 30 40 40 50 50 60 70
20 40 50 60 70 90
10 30 40 50 80
10 20
20 40 50 60 70 90
10 30 40 50 80
10 20 30 40 40 50
20 40 50 60 70 90
10 30 40 50 80
10 20 30 40 40 50 50 60 70 80
20 40 50 60 70 90
10 30 40 50 80
10 20 30
20 40 50 60 70 90
10 30 40 50 80
10 20 30 40 40 50 50
20 40 50 60 70 90
10 30 40 50 80
10 20 30 40 40 50 50 60 70 80 90

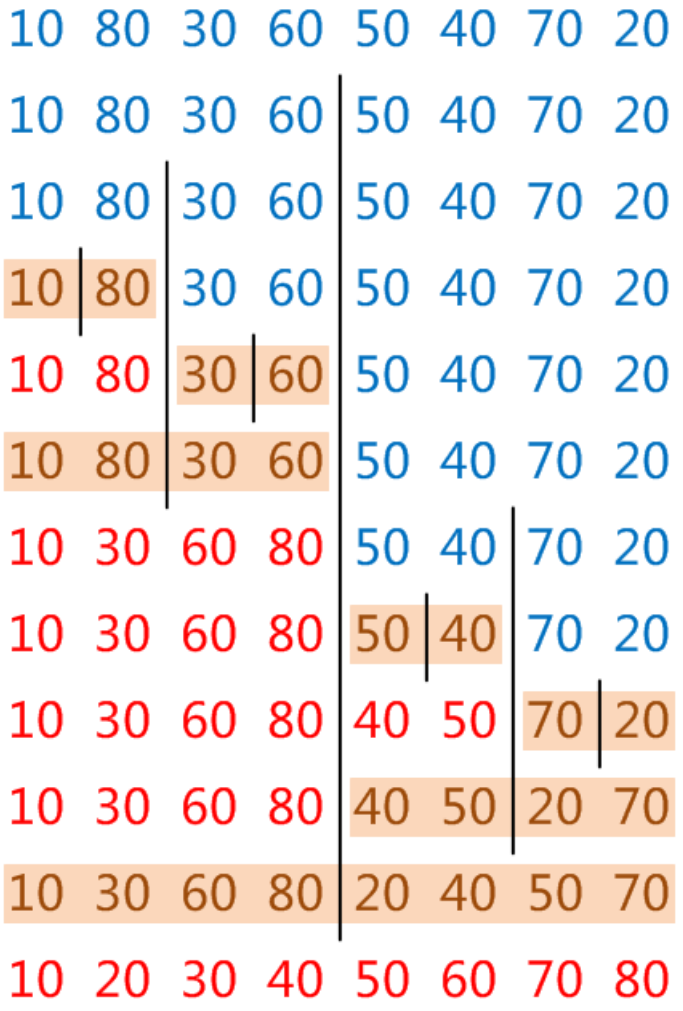
归并排序

- 算法
  1. 将待排序序列从中间划分为两个相等的子序列
  2. 以递归的方式分别对两个子序列进行排序
  3. 将两个有序的子序列合并成完整序列
- 评价
  - 平均时间复杂度：O(2NlogN)
  - 稳定排序
  - 对数据的有序性不敏感
  - 非就地排序，需要辅助空间，不适合处理海量数据

知识讲解

归并排序

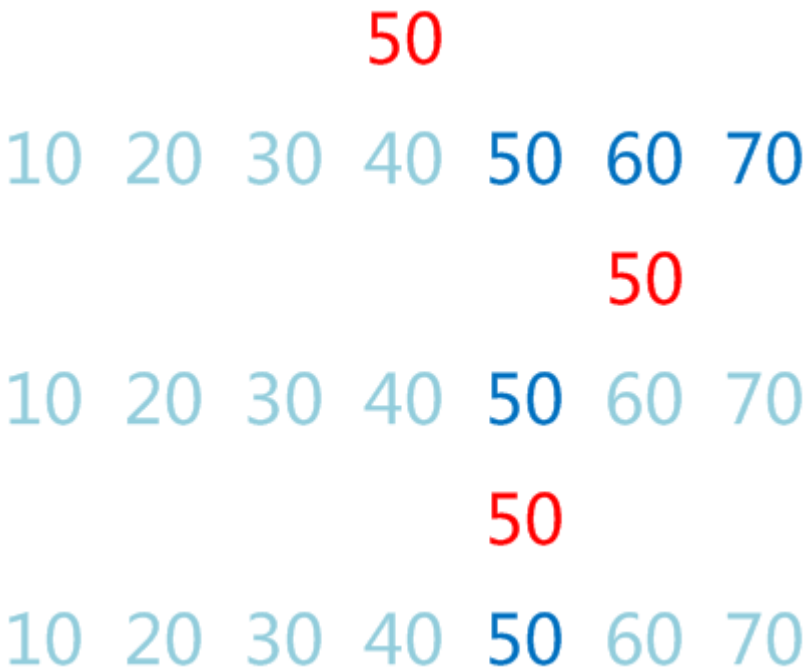
- 在递归过程中逐层合并



线性查找

- 算法
  1. 从头开始，依次将每一个元素与查找目标进行比较
  2. 或者找到目标，或者找不到目标
- 评价
  - 平均时间复杂度：O(N)
  - 对数据的有序性没有要求

二分查找



二分查找

- 算法
  1. 假设表中的元素按升序排列
  2. 若中间元素与查找目标相等，则查找成功，否则利用中间元素将表划分成前后两个子表
  3. 若查找目标小于中间元素，则在前子表中查找，否则在后子表中查找
  4. 重复以上过程，直到查找成功，或者因子表不存在而宣告失败
- 评价
  - 平均时间复杂度：O(logN)
  - 数据必须有序

练习时间

设有整数矩阵，行从左到右递增，列从上到下递增，且没有重复元素。实现一个函数，求特定元素在矩阵数组中的索引，返回-1表示该元素不存在

Technology Tarena 达内科技

“

再见

”