保定市信息学竞赛公益大讲堂 第七节

CCF CSP-1 更多知识

6.1 数据结构

6.1.1 什么是数据结构

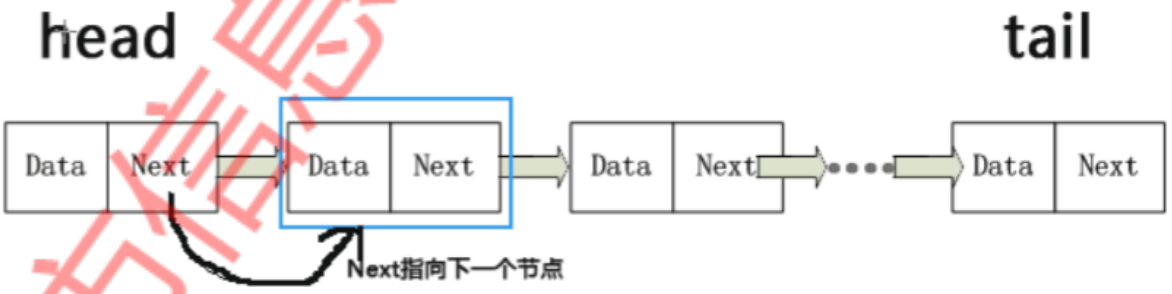
概念：数据结构是以某种形式将数据组织在一起的集合

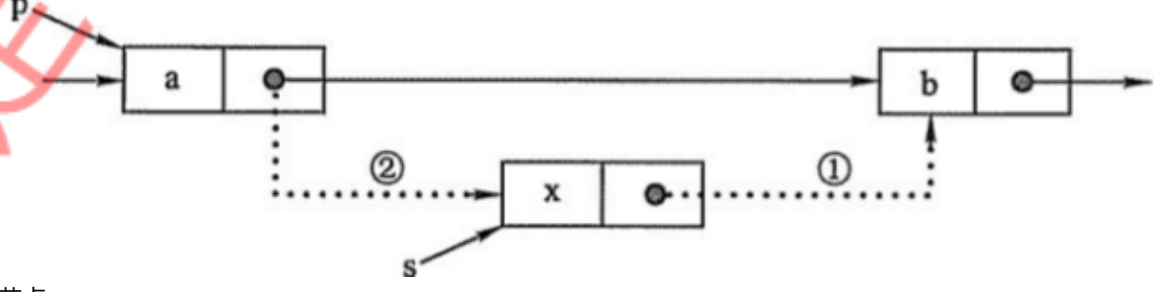
- 常见数据关系类型
  - 线性
    - 一对一关系
    - 每一个结点只有一个前驱和后继，头结点没有前驱，尾结点没有后继
  - 树
    - 一对多关系
  - 图
    - 多对多关系

6.1.2 线性数据结构

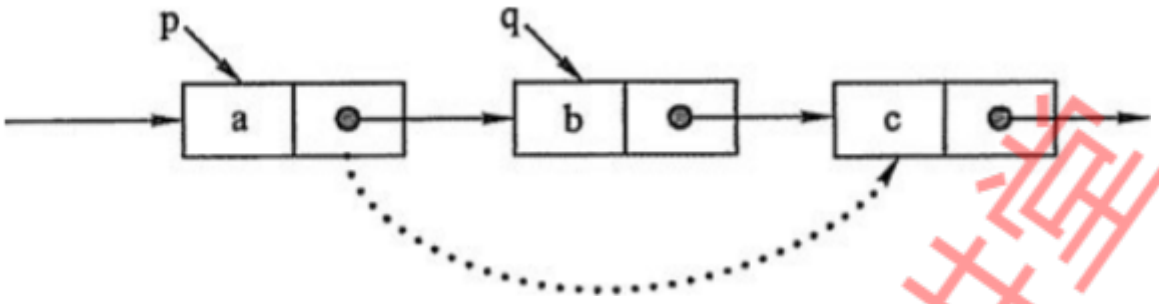
线性表

- 数组存储
  - 即用一组连续的存储单元依次存储线性表的数据元素
  - 数组大小是固定的，不可以动态添加
  - 查找速度快，但是插入数据、删除数据复杂度高
- 链表存储
  - 用一组任意的存储单元存储线性表的数据元素，通过指针关联
  - 链表是一种物理存储单元上非连续、非顺序的存储结构，数据元素的逻辑顺序是通过链表中的指针链接次序实现的
  - 链表由一系列节点组成，这些节点不必在内存中相连。每个节点由数据部分Data和链部分Next，Next指向下一个节点，这样当添加或者删除时，只需要改 变相关节点的Next的指向，效率很高



  - 添加节点
    - `S->Next=P->Next` `P->Next=S`
    - 

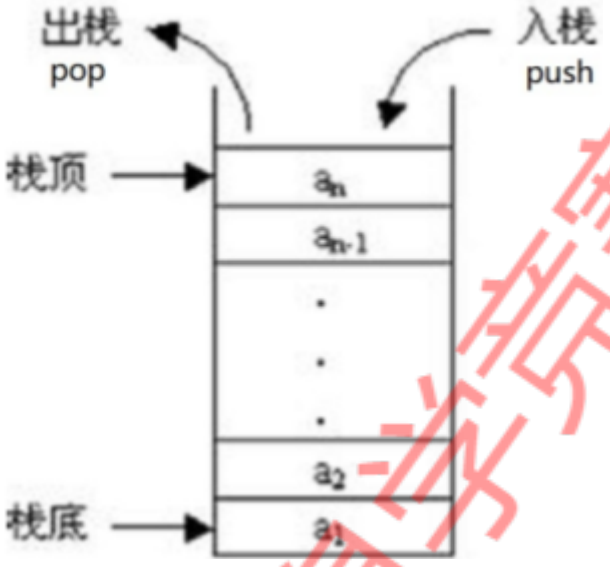
  - 删除节点
    - `P->Next=P->Next->Next`



- 链表的分类
  - 单链表，链表最后一个节点指向空
  - 循环单链表，主要是链表的最后一个节点指向第一个节点，整体构成一个链环。
  - 双向链表，主要是节点中包含两个指针部分，一个指向前驱， 一个指向后继。
  - 循环双向链表，是最后一个节点指向第一个节点

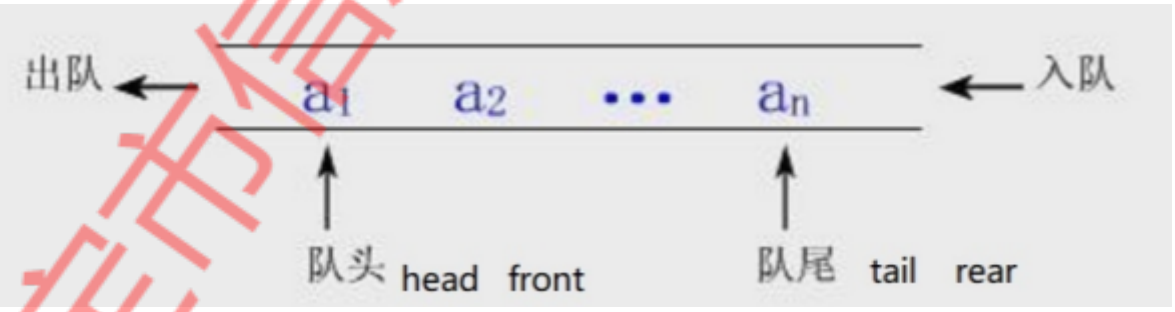
栈

- 栈只能在栈顶进行访问、插入和删除元素
- 先进后出FILO



队列

- 队列只能从队列尾插入元素，从队列头访问和删除元素
- 先进先出FIFO

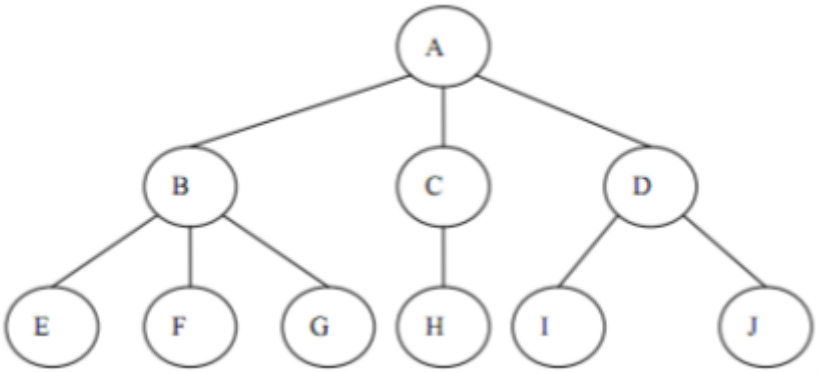


6.1.3 树

什么是树

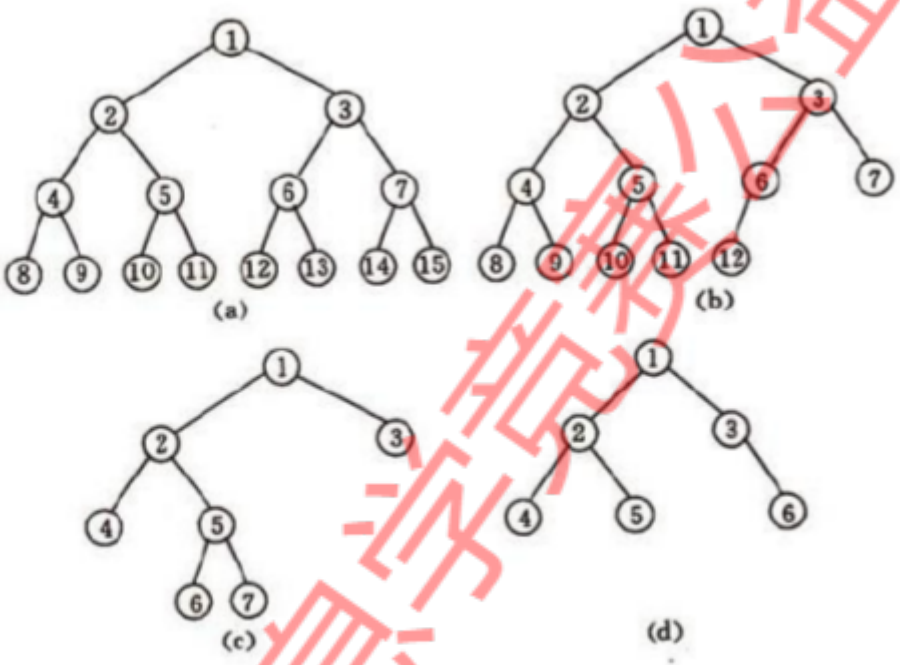
- 概念：是由n（n>=1）个有限节点组成一个具有层次关系的集合。
- 它具有以下特点：
  - 每个节点有零个或多个子节点；
  - 没有父节点的节点称为根节点；
  - 每一个非根节点有且只有一个父节点；

- 除了根节点外，每个子节点可以分为多个不相交的子树。



二叉树

- 概念：二叉树是每个节点最多有两棵子树的树结构。通常子树被称作“左子树”和“右子树”。二叉树常被用于实现二叉查找树和二叉堆。
- 性质：

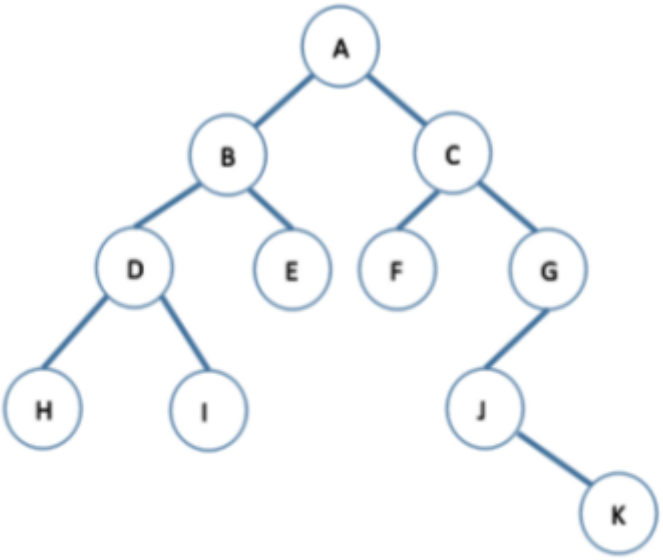


特殊形态的二叉树

（a）满二叉树；（b）完全二叉树；（c）和（d）非完全二叉树

  - 二叉树的每个结点至多只有2棵子树(不存在度大于2的结点)，二叉树的子树有左右之分，次序不能颠倒。
  - 二叉树的第i层至多有2^(i-1)个结点；深度为k的二叉树至多有2^k-1个结点。
  - 一棵深度为k，且有2^k-1个节点的二叉树称之为 满二叉树
  - 深度为k，有n个节点的二叉树，当且仅当其每一个节点都与深度为k的满二叉树中，序号为1至n的节点对应时，称之为 完全二叉树

- 三种遍历方法

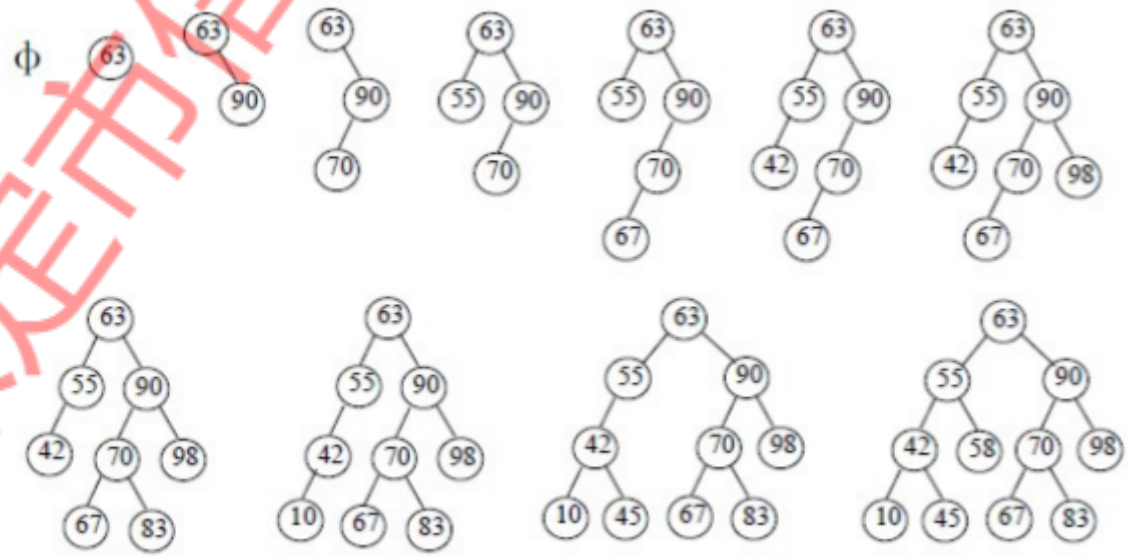


先序遍历 A B D H I E C F G J K
中序遍历 H D I B E A F C J K G
后序遍历 H I D E B F K J G C A

  - 先序遍历：若二叉树为空，则空操作，否则先访问根节点，再先序遍历左子树，最后先序遍历右子树。
  - 中序遍历：若二叉树为空，则空操作，否则先中序遍历左子树，再访问根节点，最后中序遍历右子树。
  - 后序遍历：若二叉树为空，则空操作，否则先后序遍历左子树，再后序遍历右子树，最后访问根节点。
- 树和二叉树的区别
  - 二叉树每个节点最多有2个子节点，树则无限制。
  - 二叉树中节点的子树分为左子树和右子树，即使某节点只有一棵子树，也要指明该子树是左子树还是右子树，即二叉树是有序的。
  - 树决不能为空，它至少有一个节点，而一棵二叉树可以是空的。

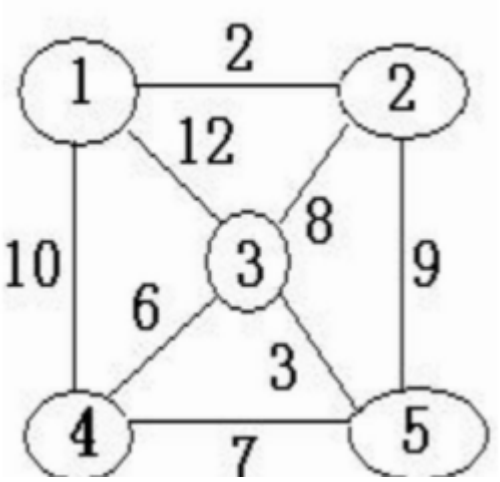
二叉排序树BST

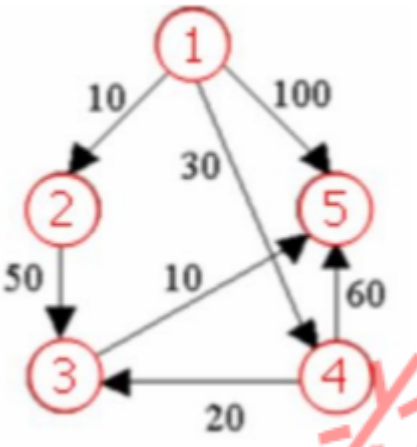
- 性质
  - 若左子树不空，则左子树上所有结点的值均小于它的根结点的值；
  - 若右子树不空，则右子树上所有结点的值均大于它的根结点的值；
  - 左、右子树也分别为二叉排序树；
  - 没有键值相等的结点。
- 示例：



6.1.4 图

基本概念



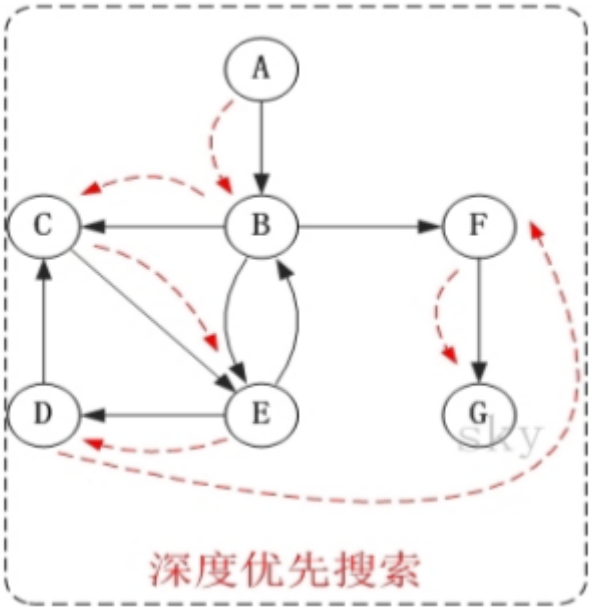


- 概念：图是一个由边和用边连接在一起的顶点组成的集合
- 图根据有向边和无向边的分类可分为有向图和无向图
- 条边赋予一个表示大小的值，这个值就称为权值
- 图的度数
  - 无向图中，一个点连接边的数量就是这个点的度数；无向图中，度数为奇数的顶点为奇点，度数为偶数则为偶点
  - 有向图中，以某顶点为弧头，终止于该顶点的弧的数目称为该顶点的入度；有向图中，以某顶点为弧尾，起始于该顶点的弧的数目称为该顶点的出度
  - 有向图中，在某顶点的入度和出度的和称为该顶点的度数
- 图的性质
  - 一个图中，所有定点的度数和是所有边数的2倍
  - 一个有向图中，所有入度之和等于所有出度之和
  - 任意一个无向图中，奇点的个数一定是偶数个
- 欧拉回路
  - 概念：通过图中每条边一次且仅有一次，并且过每个点的回路
  - 欧拉回路存在条件：图是联通的，存在0个奇点
- 欧拉通路
  - 概念：通过图中每条边一次且仅有一次，并且过每个点的通路
  - 欧拉通路存在条件：图是联通的，存在0个或者2个奇点；如果是2个奇点，则欧拉路一定是从一个奇点出发，到另一个奇点结束

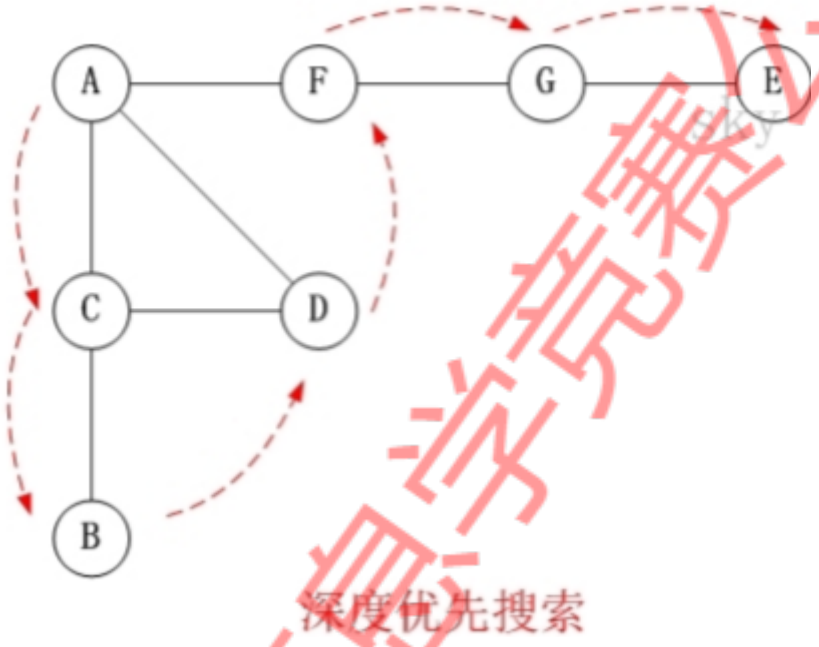
深度优先搜索 遍历 Depth First Search

- 遍历是指沿着某条搜索路线，依次对图中每个结点均做一次且仅做一次访问

- 访问顺序是：A → B → C → E → D → F → G

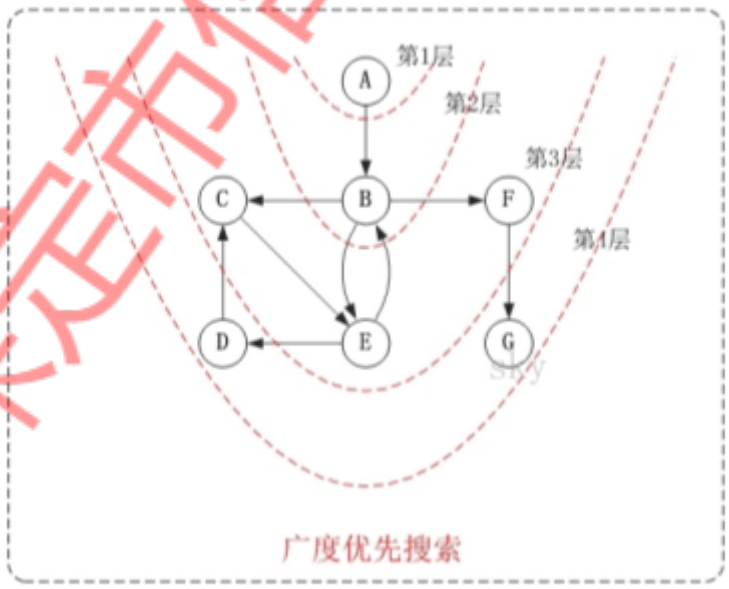


- 访问顺序是：A → C → B → D → F → G → E

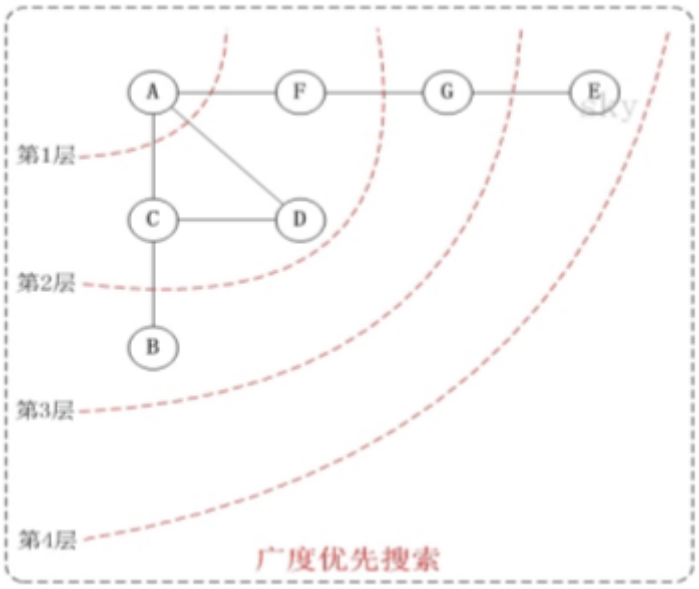


广度优先搜索 遍历 Breadth First Search

- 访问顺序是：A → B → C → E → F → D → G



- 访问顺序是：A → C → D → F → B → G → E



最小生成树 MST

例如：要在n个城市之间铺设光缆，主要目标是要使这 n 个城市的任意两个之间都可以通信，但铺设光缆的费用很高，且各个城市之间 铺设光缆的费用不同，怎样铺设可以使铺设光缆的总费用最低。

最短路径

- 单源点最短路径：Dijkstra算法
- 多源点最短路径：Floyd算法

6.2 算法

6.2.1 什么是算法

- 算法是指可以解决问题的方案的准确而完整的描述。
- 算法是一系列解决问题的清晰指令，代表着用系统的方法描述解决问题的策略机制。
- 对于同一个问题的解决，可能会存在着不同的算法，为了衡量一个算法的优劣，提出了空间复杂度与时间复杂度这两个概念。

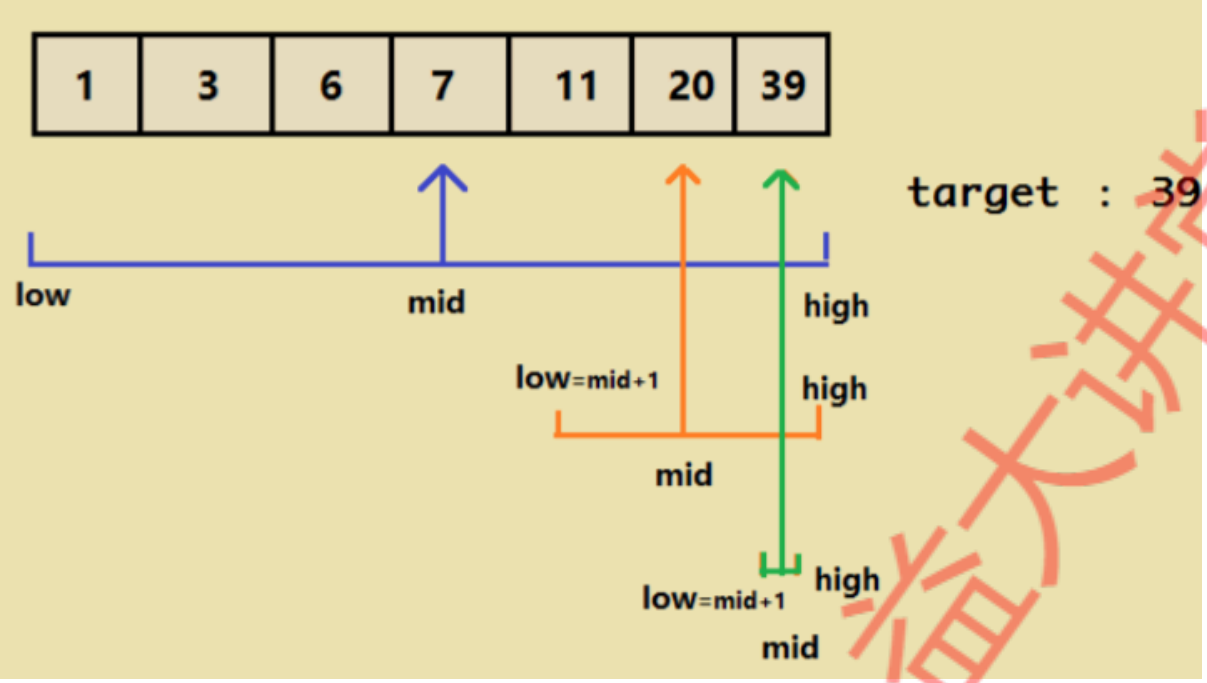
6.2.2 查找算法

顺序查找

顺序查找，又称线性查找，从头至尾逐个比较关键词，直至找到或者到末尾，时间复杂度为O(N)

二分查找

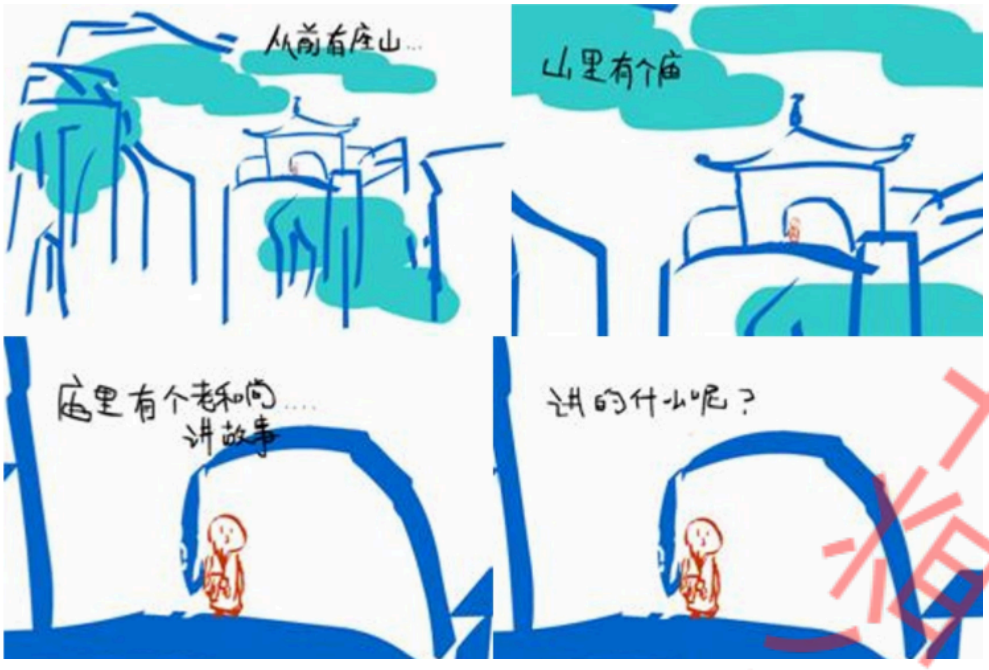
二分查找，又称折半查找，对于有序表来说，它的优点是比较次数少，查找速度快，平均性能好，时间复杂度为O(LogN)



6.2.3 排序算法

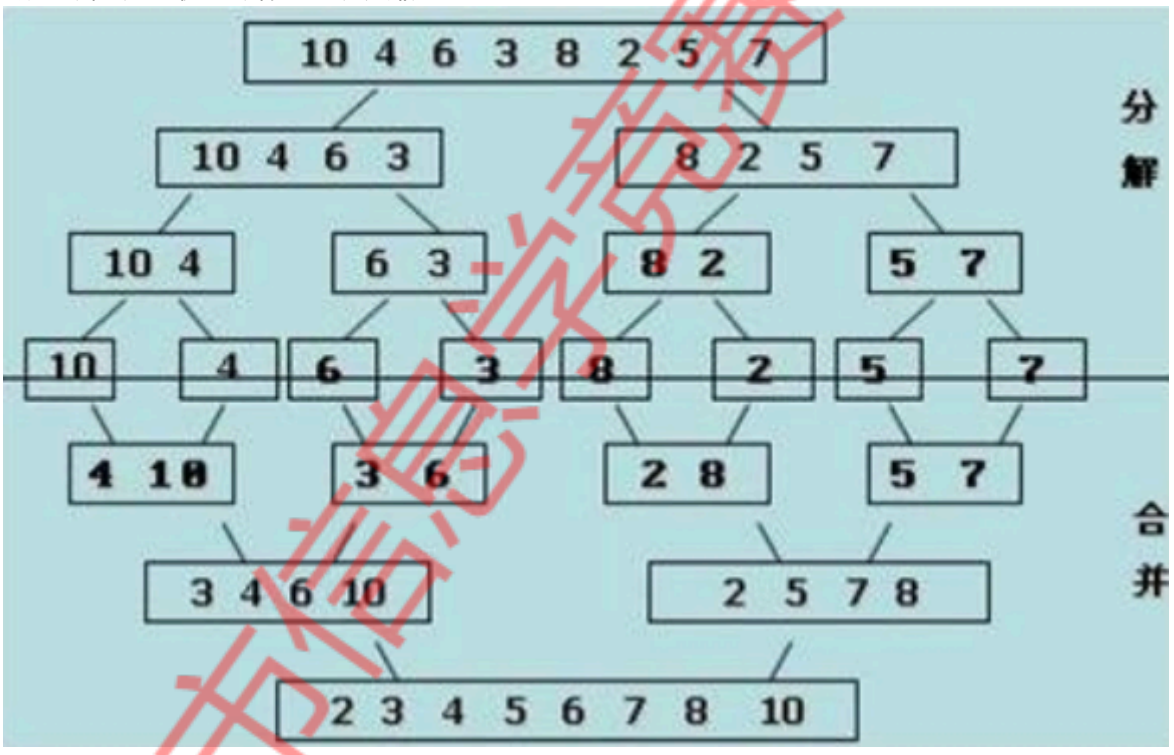
各种常用排序算法						
类别	排序方法	时间复杂度			空间复杂度	稳定性
		平均情况	最好情况	最坏情况	辅助存储	
插入排序	直接插入	$O(n^2)$	$O(n)$	$O(n^2)$	$O(1)$	稳定
	shell排序	$O(n^{1.3})$	$O(n)$	$O(n^2)$	$O(1)$	不稳定
选择排序	直接选择	$O(n^2)$	$O(n^2)$	$O(n^2)$	$O(1)$	不稳定
	堆排序	$O(n\log_2 n)$	$O(n\log_2 n)$	$O(n\log_2 n)$	$O(1)$	不稳定
交换排序	冒泡排序	$O(n^2)$	$O(n)$	$O(n^2)$	$O(1)$	稳定
	快速排序	$O(n\log_2 n)$	$O(n\log_2 n)$	$O(n^2)$	$O(n\log_2 n)$	不稳定
归并排序		$O(n\log_2 n)$	$O(n\log_2 n)$	$O(n\log_2 n)$	$O(n)$	稳定
基数排序		$O(d(r+n))$	$O(d(n+rd))$	$O(d(r+n))$	$O(rd+n)$	稳定
注：基数排序的复杂度中，r代表关键字的基数，d代表长度，n代表关键字的个数						

6.2.4 递归算法



6.2.5 分治算法

分治算法的思想是将待解决的问题分解为几个规模较小但类似于原问题的子问题，递归地求解这些子问题，然后合并这些子问题的解来建立最终的解，例如归并排序



6.2.6 动态规划

最优化原理，把多阶段过程转化为一系列单阶段问题，利用各阶段之间的关系，逐个求解

6.3 排列组合与集合运算

6.3.1 排列与组合

概念

- 排列：从给定的一组元素中取出若干个元素按照一定顺序排列起来，称为排列。例如，从3个不同的元素 {a, b, c} 中取出2个元素进行排列，可能的排列有 (a, b), (b, a), (a, c), (c, a), (b, c), (c, b)。

- **组合**：从给定的一组元素中取出若干个元素不考虑顺序进行组合，称为组合。例如，从3个不同的元素 {a, b, c} 中取出2个元素进行组合，可能的组合有 {a, b}, {a, c}, {b, c}。

公式

- **排列数公式**：给定 (n) 个不同的元素，从中取出 (m) 个元素进行排列的数目为：

$$P(n, m) = \frac{n!}{(n-m)!}$$

- **组合数公式**：给定 (n) 个不同的元素，从中取出 (m) 个元素进行组合的数目为：

$$C(n, m) = \frac{n!}{m!(n-m)!}$$

举例

- 假设从5个不同的元素 {1, 2, 3, 4, 5} 中取出3个元素进行排列和组合：
  - 排列：

$$P(5, 3) = 5 \times 4 \times 3 = 60$$

  - 组合：

$$C(5, 3) = \frac{5 \times 4 \times 3}{3 \times 2 \times 1} = 10$$

6.3.2 集合运算

集合的基本概念

- **集合**：集合是由若干元素组成的整体。集合中的元素是互不相同的。
- **子集**：若集合A的所有元素都属于集合B，则A称为B的子集。
- **并集**：两个集合A和B的并集是包含A或B中所有元素的集合，记作A ∪ B。
- **交集**：两个集合A和B的交集是同时包含在A和B中的元素集合，记作A ∩ B。
- **差集**：集合A和B的差集是属于A但不属于B的元素组成的集合，记作A - B。

集合运算的性质

- **交换律**：A ∪ B = B ∪ A，A ∩ B = B ∩ A
- **结合律**：A ∪ (B ∪ C) = (A ∪ B) ∪ C，A ∩ (B ∩ C) = (A ∩ B) ∩ C
- **分配律**：A ∪ (B ∩ C) = (A ∪ B) ∩ (A ∪ C)，A ∩ (B ∪ C) = (A ∩ B) ∪ (A ∩ C)

示例

- 给定集合A = {1, 2, 3} 和 B = {3, 4, 5}：
  - **并集**：A ∪ B = {1, 2, 3, 4, 5}
  - **交集**：A ∩ B = {3}
  - **差集**：A - B = {1, 2}

6.3.3 应用场景

- **排列与组合**：排列与组合广泛应用于密码学、概率论、统计学等领域，例如计算可能的密码组合数、确定某事件发生的概率等。
- **集合运算**：集合运算用于数据库查询、逻辑运算、集合操作等场景，例如在数据库中查找满足某些条件的记录，或者在程序中处理多组数据的交集与并集。