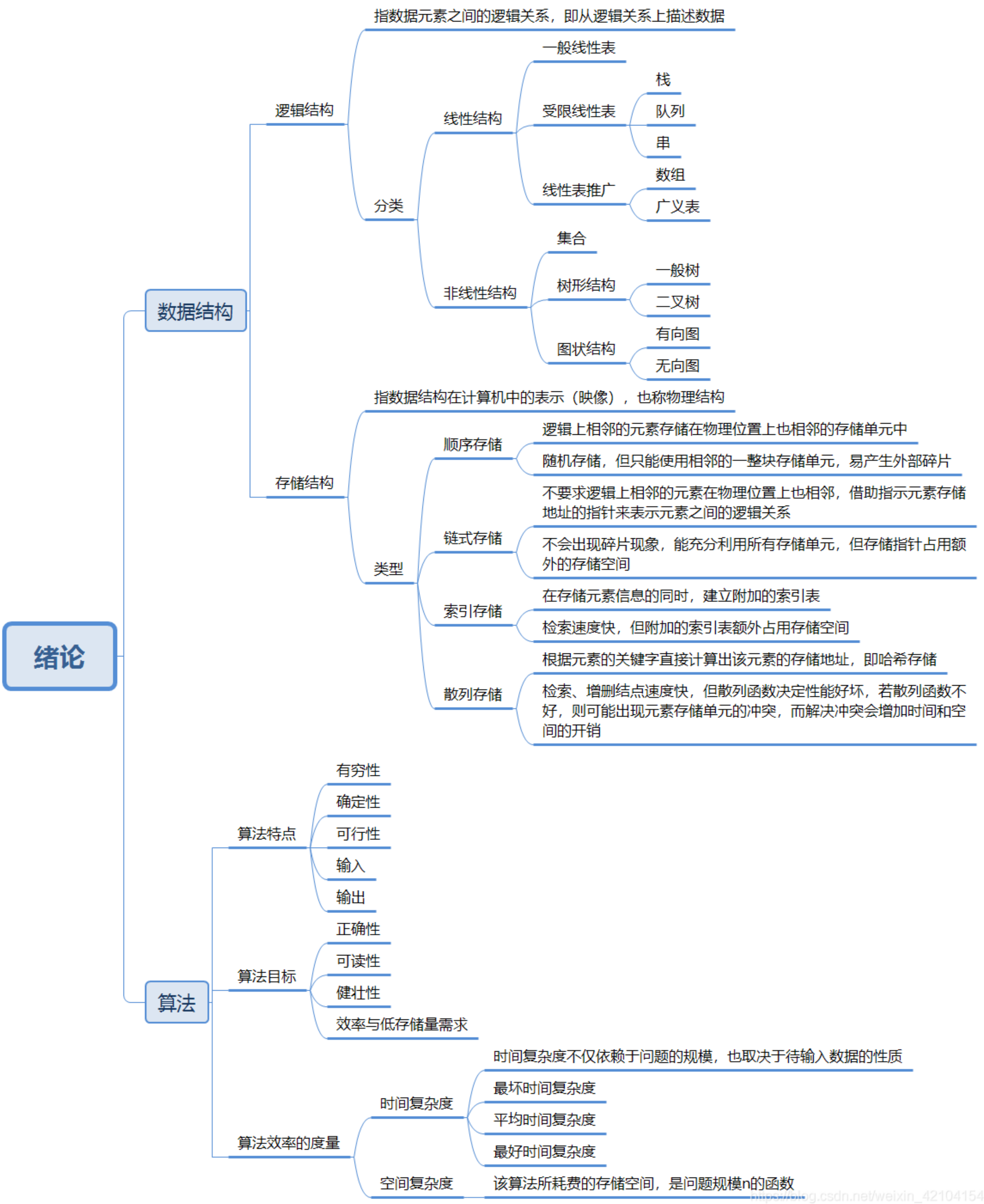
@TOC

相关课程链接

> 数据结构总结与知识网图 计算机网络知识总结及知识网图 操作系统总结及知识网图 计算机组成原理总结及知识网图

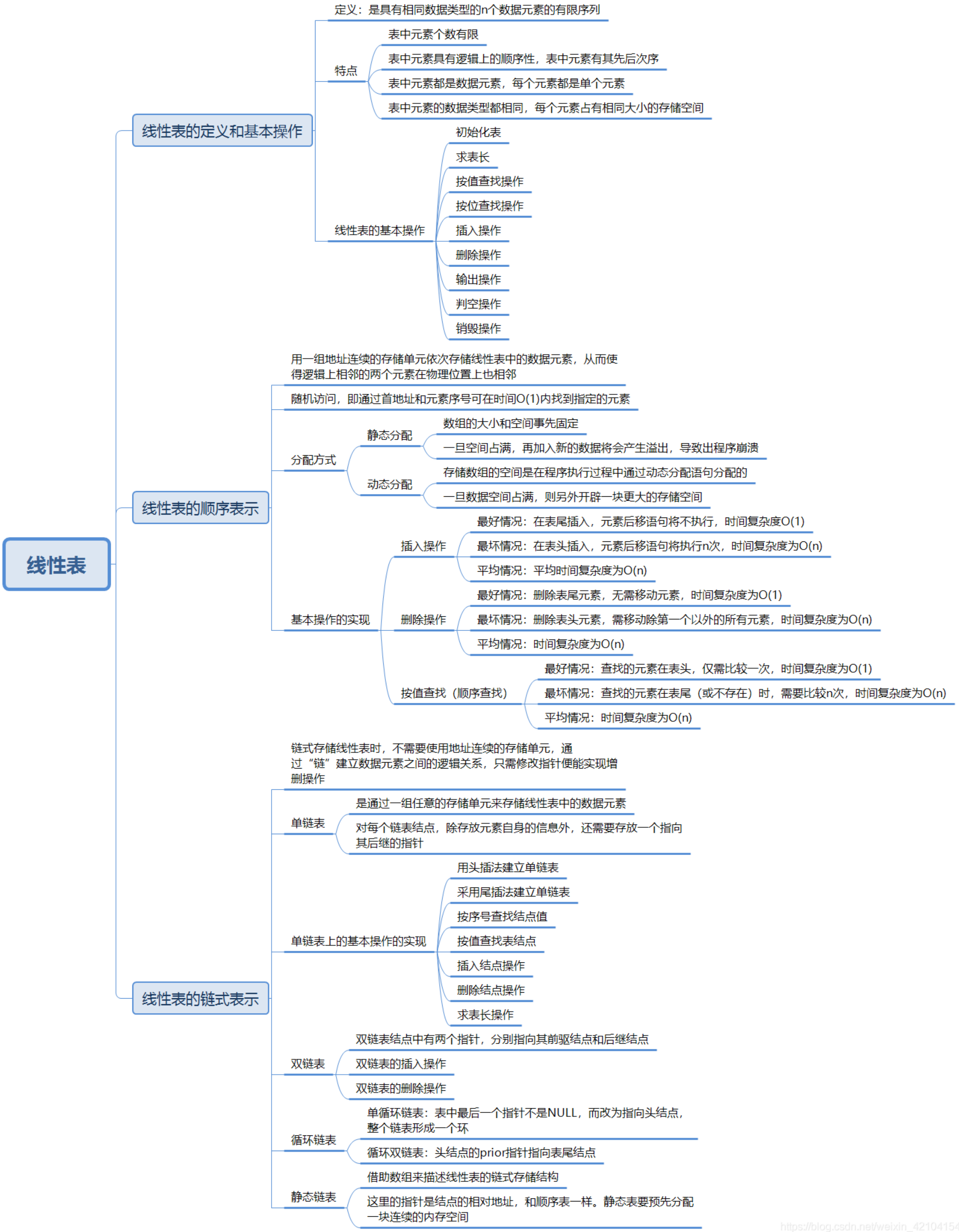
第一章 绪论

知识网图


绪论
数据结构
逻辑结构
指数据元素之间的逻辑关系，即从逻辑关系上描述数据
分类
线性结构
一般线性表
受限线性表
栈
队列
串
线性表推广
数组
广义表
非线性结构
集合
树形结构
一般树
二叉树
图状结构
有向图
无向图
存储结构
指数据结构在计算机中的表示（映像），也称物理结构
类型
顺序存储
逻辑上相邻的元素存储在物理位置上也相邻的存储单元中
随机存储，但只能使用相邻的一整块存储单元，易产生外部碎片
链式存储
不要求逻辑上相邻的元素在物理位置上也相邻，借助指示元素存储地址的指针来表示元素之间的逻辑关系
不会出现碎片现象，能充分利用所有存储单元，但存储指针占用额外的存储空间
索引存储
在存储元素信息的同时，建立附加的索引表
检索速度快，但附加的索引表额外占用存储空间
散列存储
根据元素的关键字直接计算出该元素的存储地址，即哈希存储
检索、增删结点速度快，但散列函数决定性能好坏，若散列函数不好，则可能出现元素存储单元的冲突，而解决冲突会增加时间和空间的开销
算法
算法特点
有穷性
确定性
可行性
输入
输出
算法目标
正确性
可读性
健壮性
效率与低存储量需求
算法效率的度量
时间复杂度
时间复杂度不仅依赖于问题的规模，也取决于待输入数据的性质
最坏时间复杂度
平均时间复杂度
最好时间复杂度
空间复杂度
该算法所耗费的存储空间，是问题规模n的函数

第二章 线性表

知识网图



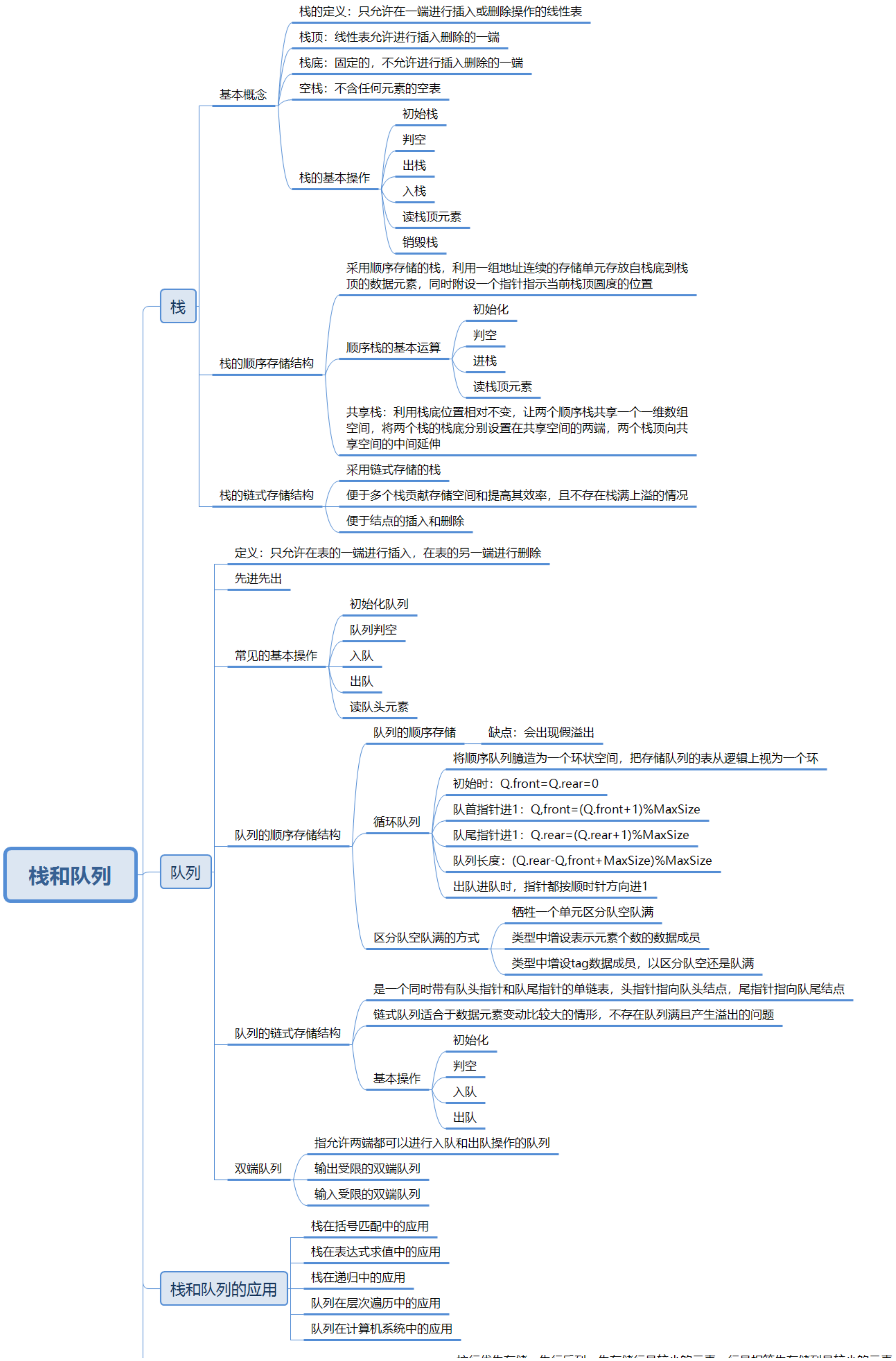
顺序表和链表的比较

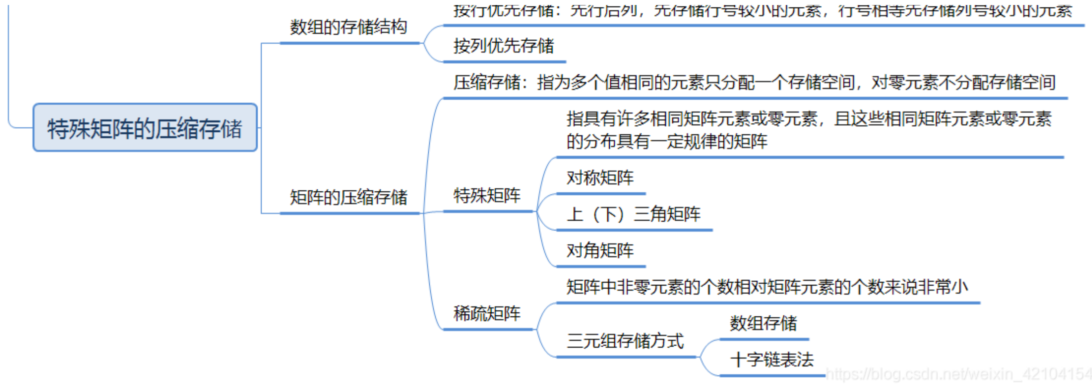
> **1）存取方式** 顺序表可以顺序存取，也可随机存取，链表只能从表头顺序存取元素 **2）逻辑结构与物理结构** 采用顺序存储时，逻辑上相邻的元素，对应的物理存储位置也相邻。采用链式存储时，逻辑上相邻的元素，物理存储位置不一定相邻，对应的逻辑关系通过指针链接来表示。 **3）查找、插入和删除操作** 对于**按值查找**，顺序表无序时，两者的时间复杂度均为O(1)，顺序表有序时，可采用折半查找，时间复杂度为O(log2n).

> 对于**按序号查找**，顺序表支持随机访问，时间复杂度为O(1),而链表的平均时间复杂度为O(n). 对于**插入、删除**操作，顺序表需要移动半个表长的元素，而链表只需修改相关结点的指针域即可。

第三章

知识网图


栈和队列
栈
基本概念
栈的定义：只允许在一端进行插入或删除操作的线性表
栈顶：线性表允许进行插入删除的一端
栈底：固定的，不允许进行插入删除的一端
空栈：不含任何元素的空表
栈的基本操作
初始栈
判空
出栈
入栈
读栈顶元素
销毁栈
栈的顺序存储结构
采用顺序存储的栈，利用一组地址连续的存储单元存放自栈底到栈顶的数据元素，同时附设一个指针指示当前栈顶圆度的位置
顺序栈的基本运算
初始化
判空
进栈
读栈顶元素
共享栈：利用栈底位置相对不变，让两个顺序栈共享一个一维数组空间，将两个栈的栈底分别设置在共享空间的两端，两个栈顶向共享空间的中间延伸
栈的链式存储结构
采用链式存储的栈
便于多个栈贡献存储空间和提高其效率，且不存在栈满上溢的情况
便于结点的插入和删除
队列
定义：只允许在表的一端进行插入，在表的另一端进行删除
先进先出
常见的基本操作
初始化队列
队列判空
入队
出队
读队头元素
队列的顺序存储结构
队列的顺序存储
缺点：会出现假溢出
循环队列
将顺序队列臆造为一个环状空间，把存储队列的表从逻辑上视为一个环
初始时：Q.front=Q.rear=0
队首指针进1：Q.front=(Q.front+1)%MaxSize
队尾指针进1：Q.rear=(Q.rear+1)%MaxSize
队列长度：(Q.rear-Q.front+MaxSize)%MaxSize
出队进队时，指针都按顺时针方向进1
区分队空队满的方式
牺牲一个单元区分队空队满
类型中增设表示元素个数的数据成员
类型中增设tag数据成员，以区分队空还是队满
队列的链式存储结构
是一个同时带有队头指针和队尾指针的单链表，头指针指向队头结点，尾指针指向队尾结点
链式队列适合于数据元素变动比较大的情形，不存在队列满且产生溢出的问题
基本操作
初始化
判空
入队
出队
双端队列
指允许两端都可以进行入队和出队操作的队列
输出受限的双端队列
输入受限的双端队列
栈和队列的应用
栈在括号匹配中的应用
栈在表达式求值中的应用
栈在递归中的应用
队列在层次遍历中的应用
队列在计算机系统中的应用



区分队空或队满的方式

> 1)牺牲一个单元来区分队空和队满，入队时少用一个队列单元。即约定以“队头指针在队尾指针的下一位置作为队满的标志”。 2）类型中增设表示元素个数的数据成员。 3）类型中增设tag数据成员，以区分队满或队空。tag=0时，若因删除导致Q.front=Q.rear，则队空；若因插入导致Q.front=Q.rear，则队满。

栈在括号匹配中的应用

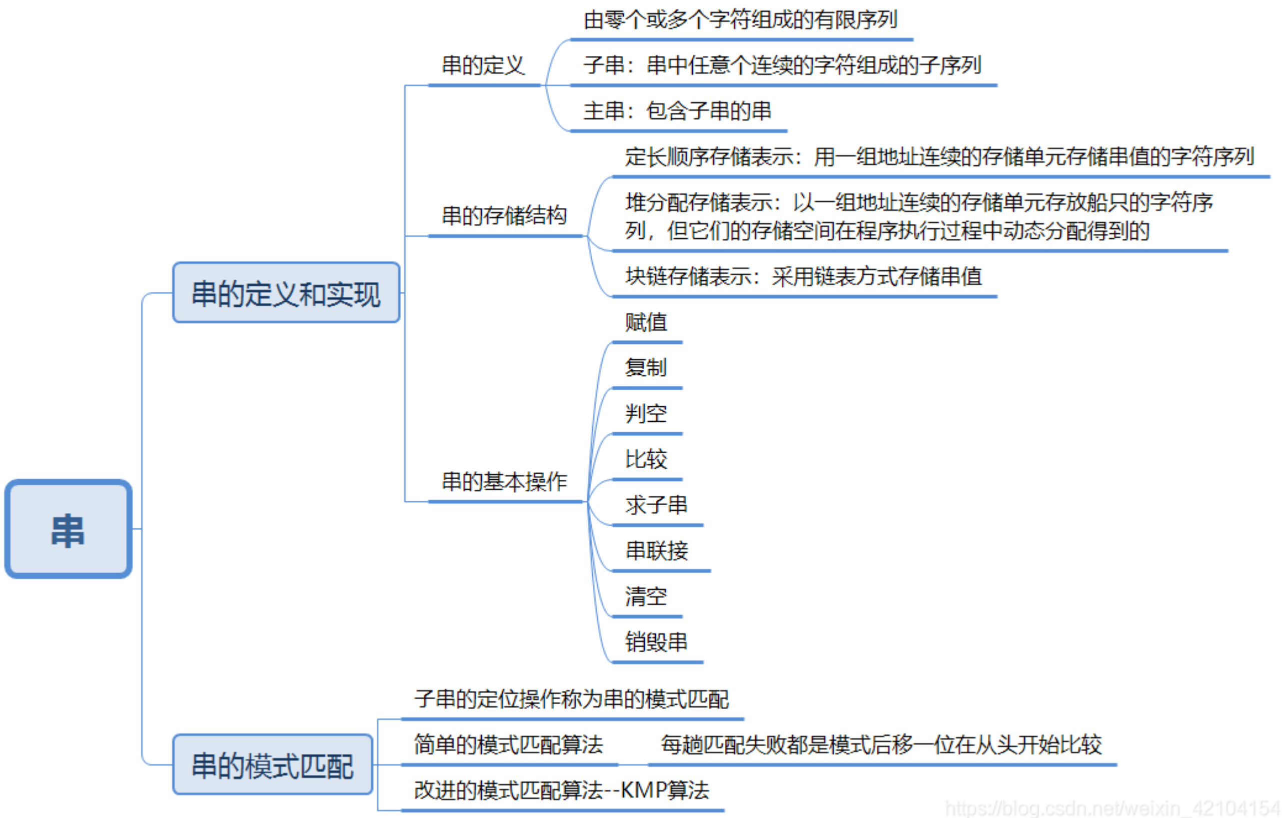
> **算法思想：** 1）初始设置一个空栈，顺序读入括号。 2）若是右括号，则或者使置于栈顶的最急迫期待得以消解，或者是不合法的情况（括号序列不匹配，则退出程序。 3）若是左括号，则作为一个新的更急迫的期待压入栈中，使原有的在栈中的所有未消解的期待的急迫性降了一级。算法结束时，栈为空，否则括号序列不匹配。

队列在计算机系统中的应用

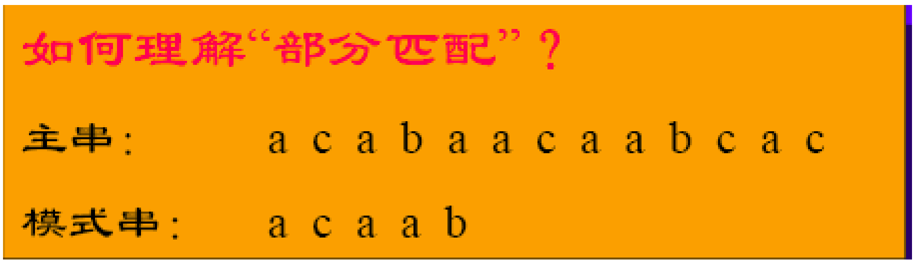
> 队列经常被应用于计算机系统中。 **1）解决主机与设备之间的速度不匹配的问题 2）解决由多用户引起的资源竞争问题**

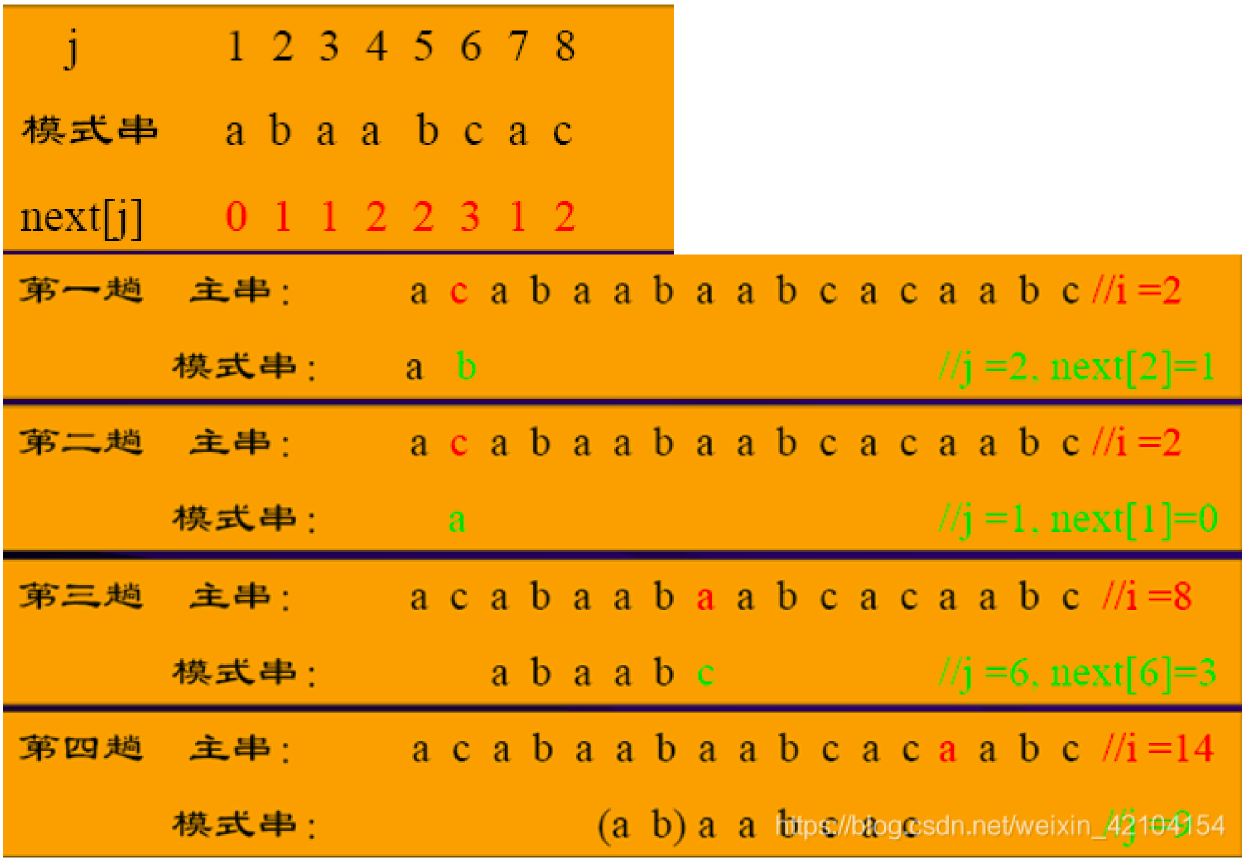
第四章 串

知识网图



KMP算法

> KMP的时间复杂度为O(m+n)，直观地看，是因为在匹配过程中指针 i 没有回溯。KMP算法的核心思想是利用已经得到的部分匹配信息来进行后面的匹配过程。 从主串s的第pos个字符起和模式的第一个字符比较之，若相等，继续逐个比较后继字符。当一趟匹配过程中出现字符比较不等时，不回溯指针，而是利用已经得到的"部分匹配"的结果将模式串向右"滑动"尽可能远的一段距离后，继续进行比较。
>
> 
>
>
> **KMP算法匹配过程**



串的特点

> **串的两个显著特点是：**

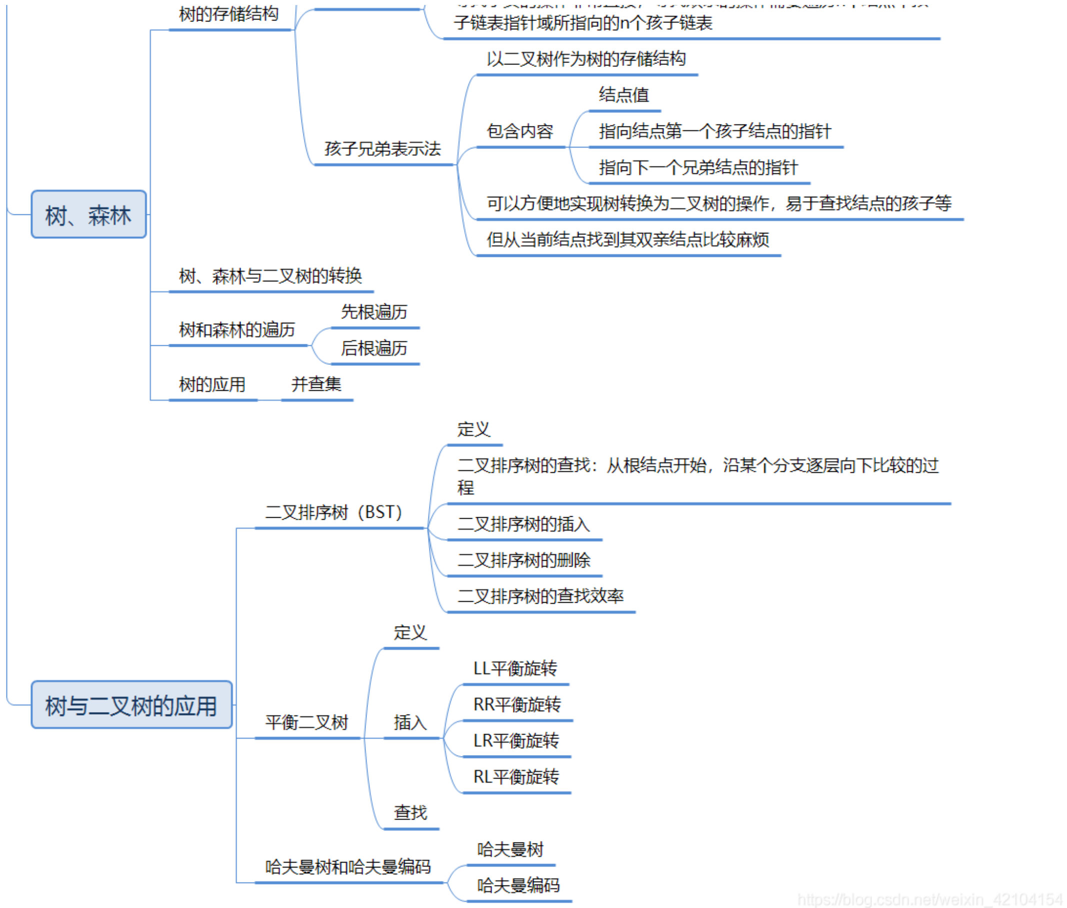
```
1、它的数据元素都是字符，因此它的存储结构和线性表有很大不同，例如多数情况下，实现串类型采用的是“堆分配”的存储结构，而当用链表存储串值时，结点中数据域的类型不是“字符”，而是“串”，这种块链结构通常只在应用程序中使用；
2、串的基本操作通常以"串的整体"作为操作对象，而不像线性表是以"数据元素"作为操作对象。
```

第五章 树与二叉树

知识网图

树与二叉树

- 树的基本概念
  - 树的定义
    - 树是一种递归的数据结构
    - 树的根结点没有前驱，除根结点外的所有结点有且仅有一个前驱
    - 树中的所有结点可以有零个或多个后继
  - 基本术语
    - 祖先：根A到结点K的唯一路径上的任意结点，称为结点K的祖先
    - 度：树中一个结点的孩子的个数称为结点的度，树中结点的最大度数称为树的度
    - 结点的深度：从根结点开始自顶向下逐层累加的
    - 结点的高度：从叶节点开始自底向上逐层累加的
    - 树的高度：树中结点的最大层数
    - 有序树：树中结点的各子树从左到右是有序的，不能互换
    - 无序树：树中结点的各子树无序，可以互换
    - 路径：树中两个结点之间的路径是由两个结点之间所经过的结点序列构成的
    - 路径长度：路径上所经过的边的个数
    - 森林：是m(m>=0)棵互不相交的树的集合
  - 树的性质
    - 树中的结点数等于所有结点的度数+1
    - 度为m的树中第i层上至多有m^(i-1)个结点
    - 高度为h的m叉树最多有m^(i-1)/(m-1) 个结点
    - 具有n个结点的m叉树的最小高度为logm(n(m-1)+1)
- 二叉树
  - 概念
    - 二叉树的定义：每个结点至多有两棵子树，并且二叉树的子树有左右之分，其次序不能任意颠倒
    - 特殊的二叉树
      - 满二叉树
        - 一棵高度为h，且含有2^h-1个结点的二叉树
        - 除叶子结点外的每个结点度数均为2
      - 完全二叉树
        - 高度为h，有n个结点的二叉树，当且仅当其每个结点都与高度为h的满二叉树编号为1~n的结点一一对应时，称为完全二叉树
      - 二叉排序树
        - 左子树上所有结点的关键字均小于根结点的关键字；右子树上所有结点的关键字均大于根结点的关键字
        - 左右子树又各是一颗二叉排序树
      - 平衡二叉树
        - 树上任一结点的左右子树的深度之差不超过1
    - 二叉树的性质
      - 非空二叉树上的叶子结点数等于度为2的结点数+1
      - 非空二叉树上的第k层上至多有2^(k-1)个结点
      - 高度为h的二叉树至多有2^(h-1)个结点
    - 二叉树的存储结构
      - 顺序存储结构
        - 用一组地址连续的存储单元依次自上而下、自左而右存储完全二叉树上的结点元素
        - 完全二叉树和满二叉树采用顺序存储比较合适
      - 链式存储结构
        - 用链表节点存储二叉树中的每个结点
        - 二叉链表的域
          - 数据域
          - 左指针域
          - 右指针域
  - 二叉树的遍历
    - 二叉树的遍历是指按照某条搜索路径访问树中每个结点，使得每个结点均被访问一次，且仅被访问一次
    - 先序遍历：根左右
    - 中序遍历：左根右
    - 后序遍历：左右根
    - 层次遍历：从上到下，从左到右依次访问每个结点
  - 线索二叉树
    - 为了加快查找结点前驱和后继的速度，利用空指针存放指向其前驱和后继的指针
    - 先序线索二叉树
    - 中序线索二叉树
    - 后序线索二叉树
- 双亲表示法
  - 采用一组连续空间来存储每个结点，同时在每个结点中增设一个伪指针，指示其双亲结点在数组中的位置
  - 利用每个结点只有唯一双亲的性质，可以很快得到每个结点的双亲结点
  - 求结点的孩子时需要遍历整个结构
- 孩子表示法
  - 将每个结点的孩子都用单链表链接起来形成一个线性结构
  - 寻找子女的操作非常直接，寻找双亲的操作需要遍历n个结点中孩



树的性质

> 1)树中的结点数等于所有结点的度数加1. 2）度为m的树中的第i层上至多m^(i-1)个结点。 3）高度为h的m叉树至多有(m^h-1)/(m-1)个结点。 4）具有n个结点的m叉树的最小高度为(logm(n(m-1)+1))取上限。

二叉树与度为2的有序树的区别

> 1）度为2的树至少有三个结点，而二叉树可以为空。 2）度为2的有序树的孩子的左右次序是相对于另一孩子而言，若某一结点只有一个孩子，则无左右次序之分，而二叉树有左右次序之分。

二叉树的性质

> 1)在二叉树的第i层上至多有2i-1个结点（i>0）。 2)深度为k的二叉树至多有2k-1个结点（k>0）。（满二叉树） 3)对于任何一棵二叉树，若2度的结点数有n2个，则叶子数（n0）必定为n2＋1 （即n0=n2+1） 4)具有n个结点的完全二叉树的深度必为□log2n□ （取下限） ＋1 5)：对完全二叉树，若从上至下、从左至右编号，则编号为i 的结点，其左孩子编号必为2i，其右孩子编号必为2i＋1；其双亲的编号必为□ i/2 □ （取下限） （i＝1 时为根,除外）

第六章 图

知识网图

- 图
  - 图的基本概念
    - 有向图：有向边的有限集合
    - 无序图：无向边的有限集合
    - 简单图
      - 不存在重复边
      - 不存在顶点到自身的边
    - 多重图：某两个结点之间的边数多于一条，又允许顶点通过同一条边和自己关联
    - 完全图
    - 子图
    - 连通、连通图和连通分量
      - 图G中任意两个顶点都是连通的。，则图G为连通图，否则为非连通图
      - 无向图中的极大连通子图称为连通分量
    - 强连通图、强连通分量
      - 在有向图中，若从顶点v到顶点w和从顶点w到顶点v之间都有路径，则称这两个顶点为强连通的。
      - 若图中任何一对顶点都是强连通的，则称此图为强连通图
      - 有向图的极大连通子图称为有向图的强连通分量
    - 生成树、生成森林
      - 包含图中的全部顶点的一个极小子图称为连通图的生成树
    - 顶点的度、入度和出度
      - 图中每个顶点的度定义为以该顶点为一个端点的边的数目
    - 边的权和网
      - 在一个图中，每条边都可以标上具有某种含义的数值，该数值称为权值
      - 边上带有权值的图称为带权图，也称为网
    - 稠密图、稀疏图
    - 路径、路径长度和回路
    - 简单路径、简单回路
    - 距离
    - 有向树
  - 图的存储及基本操作
    - 邻接矩阵法
      - 用一个一维数组存储图中的顶点的信息，用一个二维数组存储图中边的信息，存储顶点之间邻接关系的二维数组称为邻接矩阵
    - 邻接表法
      - 子主题 1
    - 十字链表法
    - 邻接多重表
  - 图的遍历
    - 图的遍历是从图中的某一顶点出发，按照某种搜索方法沿着图中的边对图中的所有顶点访问一次且仅访问一次
    - 遍历算法
      - 广度优先搜索
      - 深度优先搜索
    - 图的遍历算法可以用来判断图的连通性
  - 图的应用
    - 最小生成树
      - 最小生成树的性质
        - 最小生成树不是唯一的
        - 最小生成树的边的权值之和是唯一的
        - 最小生成树的边数为顶点数-1
      - 最小生成树算法
        - Prim算法
        - Kruskal算法
    - 最短路径
      - Dijkstra算法求单源最短路径问题
      - Floyd算法求各顶点之间最短路径问题
    - 有向无环图描述表达式
    - 拓扑排序
    - 关键路径

图的存储结构

根据图的自身特性给，可采取以下五种存储方式： 1）邻接矩阵 2）邻接表 3）邻接多重表 4）十字链表 5）边集数组 **1）邻接矩阵** 用两个数组来表示图，一个一维数组存储图中的顶点信息，一个二维数组（邻接矩阵）存储图中的边或弧的信息。

- 若图G有n个顶点，则邻接矩阵是一个n*n的方阵，定义为：

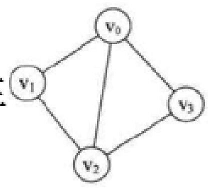
$$arc[i][j] = \begin{cases} 1, 若(v_i,v_j) \in E 或 < v_i,v_j > \in E \\ 0, 反之 \end{cases}$$

- 若图G是网图（**带权**），有n个顶点，则邻接矩阵是一个n*n的方阵，定义为：

$$arc[i][j]= \begin{cases} W_{ij}, 若(v_i,v_j) \in E 或 < v_i,v_j > \in E \\ 0, \quad 若\ i = j \\ \infty, \quad 反之 \end{cases}$$

①图的邻接矩阵

- 顶点集合：一维数组
- 边（弧）集合：二维矩阵
  - 0 – 无边（即没有关系）
  - 1 – 有边（即存在逻辑关系）
  - 无向图的邻接矩阵为对称矩阵（可压缩存储）



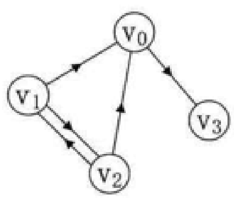
顶点数组： | v_0 | v_1 | v_2 | v_3 |

边数组：

$$\begin{array}{c|c} & \begin{matrix} v_0 & v_1 & v_2 & v_3 \end{matrix} \\ \begin{matrix} v_0 \\ v_1 \\ v_2 \\ v_3 \end{matrix} & \begin{bmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 \end{bmatrix} \end{array}$$

v_1的度为2

主对角线



顶点数组： | v_0 | v_1 | v_2 | v_3 |

边数组：

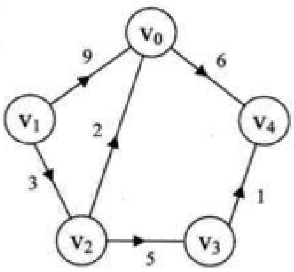
$$\begin{array}{c|c} & \begin{matrix} v_0 & v_1 & v_2 & v_3 \end{matrix} \\ \begin{matrix} v_0 \\ v_1 \\ v_2 \\ v_3 \end{matrix} & \begin{bmatrix} 0 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \end{array}$$

v_1的出度为2

v_1的入度为1

②网图（带权）的邻接矩阵

- 特殊符号 ∞ 表示不可达（即不存在边）
  - 在实现时，可用权值取值范围外的任意值表示，比如整型数的最大值
  - 主对角线上值为零，可理解为自身到自身代价为零



顶点数组：

v_0	v_1	v_2	v_3	v_4

边数组：

$$
\begin{array}{c}
 \\ v_0 \\ v_1 \\ v_2 \\ v_3 \\ v_4
\end{array}
\begin{array}{c}
\begin{array}{ccccc} v_0 & v_1 & v_2 & v_3 & v_4 \end{array} \\
\begin{bmatrix}
0 & \infty & \infty & \infty & 6 \\
9 & 0 & 3 & \infty & \infty \\
2 & \infty & 0 & 5 & \infty \\
\infty & \infty & \infty & 0 & 1 \\
\infty & \infty & \infty & \infty & 0
\end{bmatrix}
\end{array}
$$

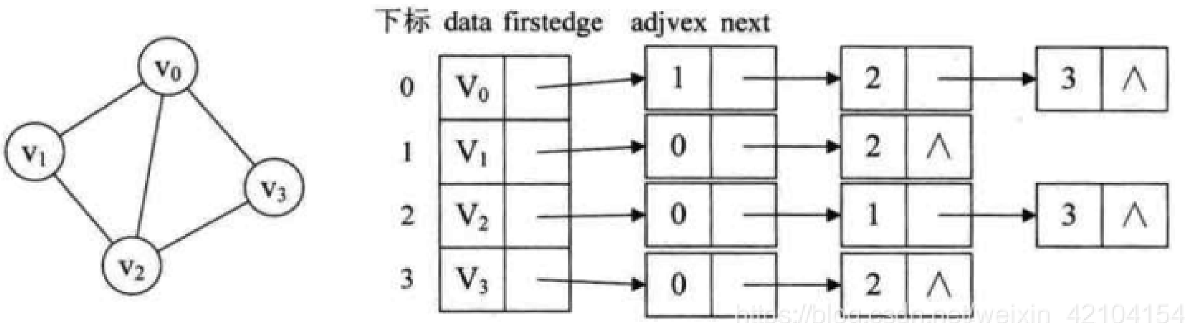
③ 邻接矩阵的实现

- 主要类型：顶点数据类型、边的权值类型
- 主要变量：最多有多少个顶点、无穷的具体值
- 图结构：
  - 顶点集合、边集合
  - 顶点数量、边数量

```
typedef char VertexType; /* 图的顶点类型 */
typedef int EdgeType; /* 边（弧）上权值类型 */

typedef struct{ /* 图的邻接矩阵表示 */
    VertexType vexs[MAXVEX];  /* 图的顶点集合 */
    EdgeType edges[MAXVEX][MAXVEX]; /* 图的边集合（邻接矩阵） */
    int numVertexs, numEdges; /* 图的顶点数、边数 */
}GraphAMatrix; /* 采用邻接矩阵存储的图的结构类型 */
```

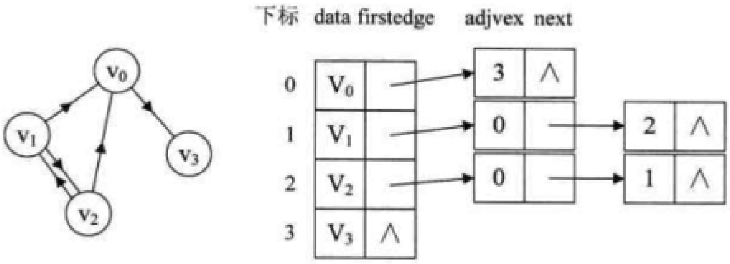

2）**邻接表** 数组与链表相结合的存储方法 一维数组存放所有顶点信息 每个顶点Vi的所有邻接点构成一个线性表，用链表存储

- 无向图：顶点Vi 的边表
- 有向图：顶点Vi 作为弧尾的出边表

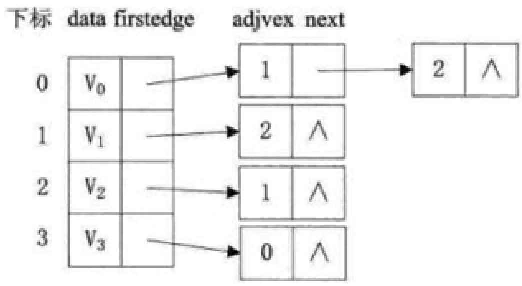


① 邻接表与逆邻接表

- 边表为出边的称为邻接表
- 边表为入边的称为**逆邻接表**



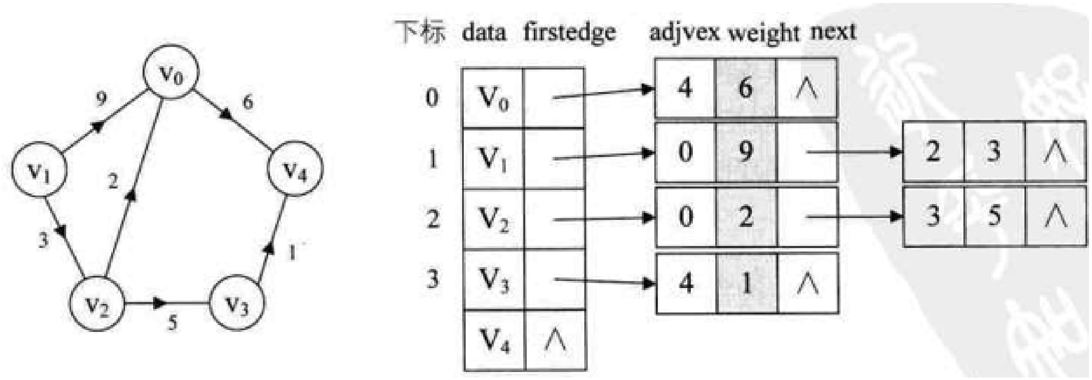
邻接表



逆邻接表

② 有向网图的邻接表

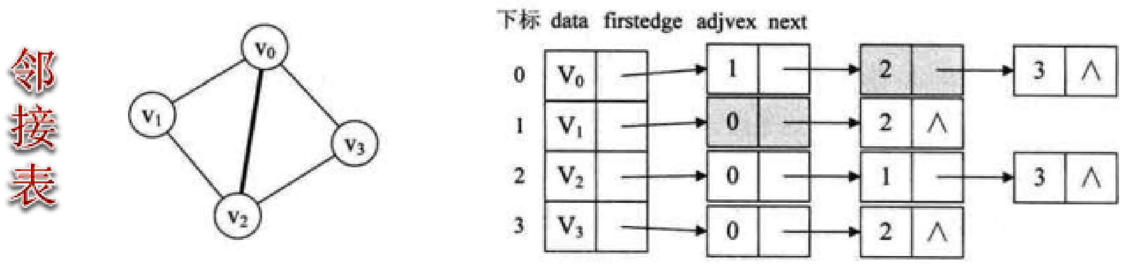
- 在邻接链表的结点中，添加一个权值属性

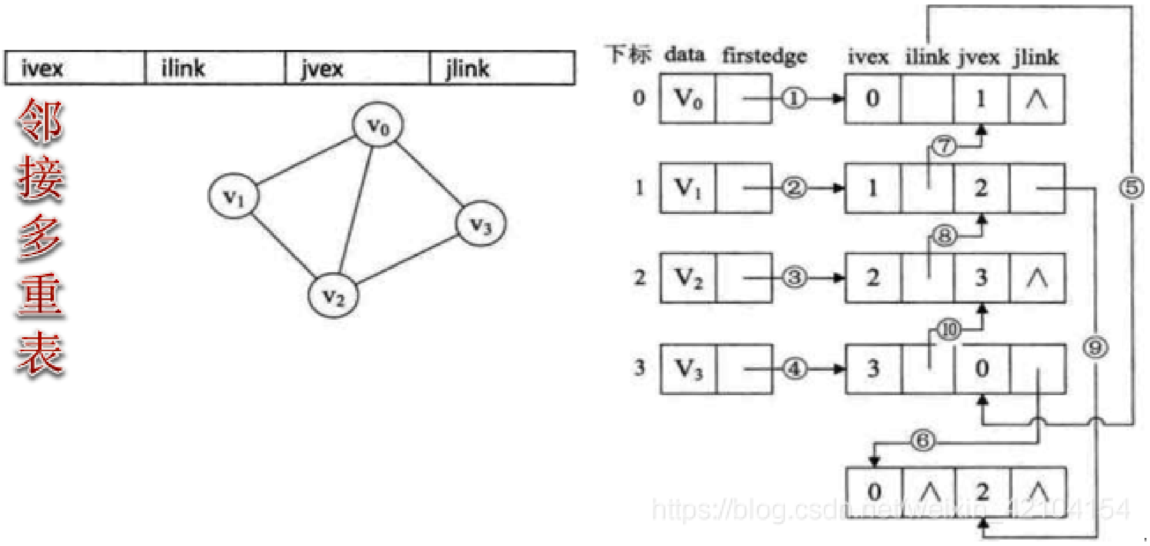


③ 邻接表的实现(代码描述)

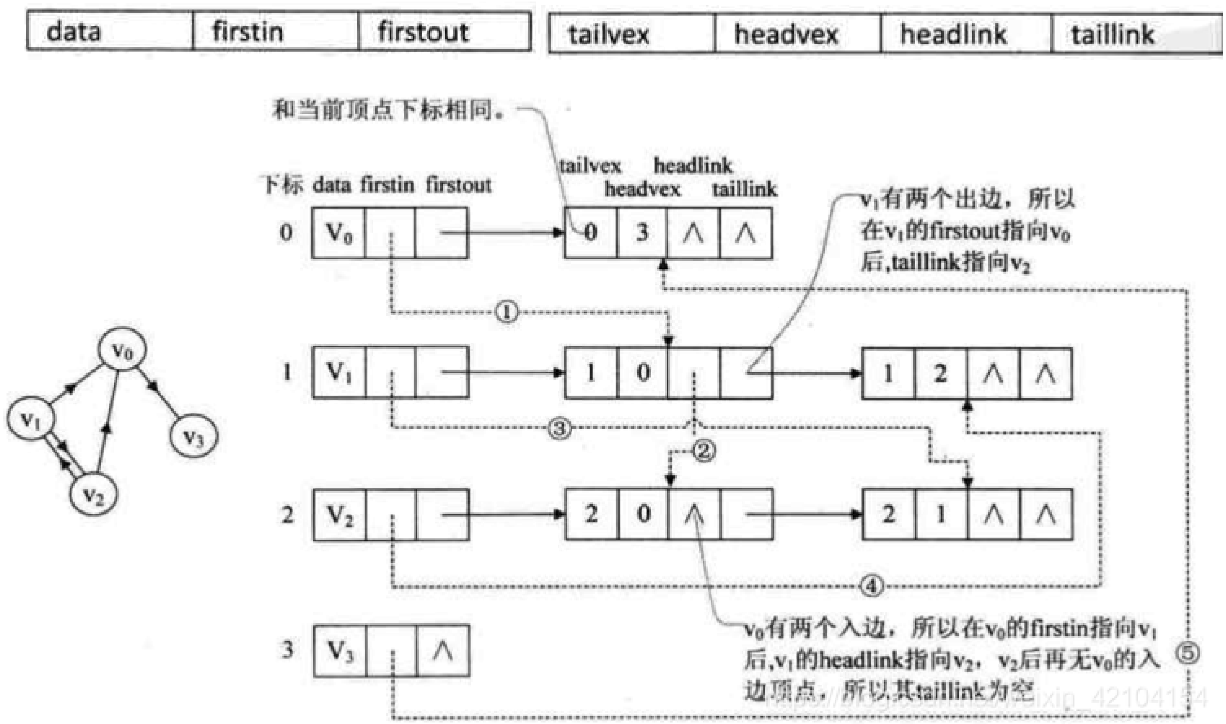
```
#define MAXVEX 100 /* 顶点数量的最大值 */
#define INFINITY 65545 /* 自定义的不可达值，整型数的最大值 */
typedef char VertexType; /* 顶点类型 */
typedef int EdgeType; /* 边上权值的类型 */

typedef struct EdgeNode{ /* 邻接链表中结点类型，及边结点 */
    int adjvex; /* 对应顶点序号，即有向图中的弧头顶点 */
    EdgeType weight; /* 此边对应的权值 */
    struct EdgeNode *next; /* 相同弧尾的下一条边 */
}EdgeNode;

typedef struct{ /* 顶点类型，除了数据域以外，还需要包含对应的第一个边结点的地址 */
    VertexType data; /* 数据域，存放顶点信息 */
    EdgeNode *firstedge; /* 指针域，指向邻接链表中的第一个结点地址，即第一条边 */
}VertexNode, AdjList[MAXVEX]; /* AdjList 即为邻接表中含边信息的顶点集合 */

typedef struct{ /* 邻接表表示的图的结构类型 */
    AdjList adjList; /* 含边信息的顶点集合 */
    int numVextexes, numEdges; /* 图中顶点数和边数 */
}GraphADJList;
```


3）邻接多重表 对于无向图，一条边对应的两个顶点互为邻接点，一条边会产生两个邻接表结点，则会造成空间浪费，操作繁琐，变得删除修改均要找到两个顶点操作两次。 改造边结点类型，形成一种新的结构：邻接多重表



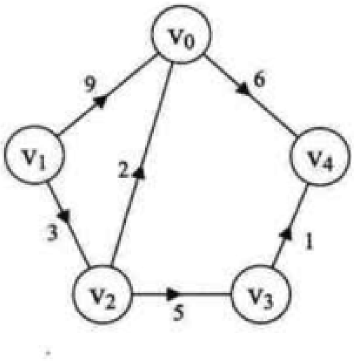


4）十字链表 对于有向图，邻接表关注了顶点的出边，易于计算出度，入读则效率低下。而逆邻接表关注了顶点的入边，易于计算入度，但出度效率低下。 将二者综合考虑，邻接表+逆邻接表，改造顶点和边结点的类型，形成一种十字链表



5）**边集数组** 由两个一维数组组成，一个存放顶点的信息，即顶点集合，一个存放边的信息，即边集数组

- 边数组元素由边的起点下标、终点下标及权值组成



顶点数组：

v_0	v_1	v_2	v_3	v_4

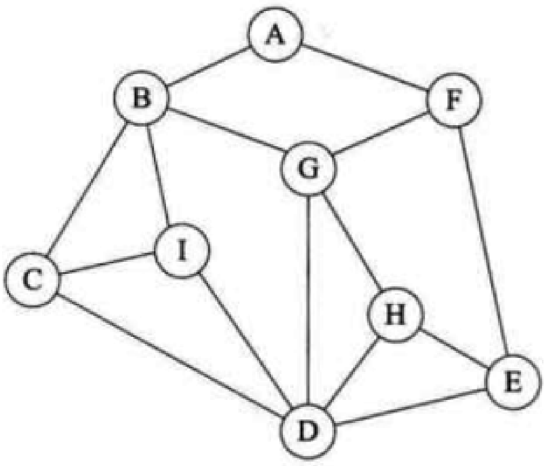
边数组：

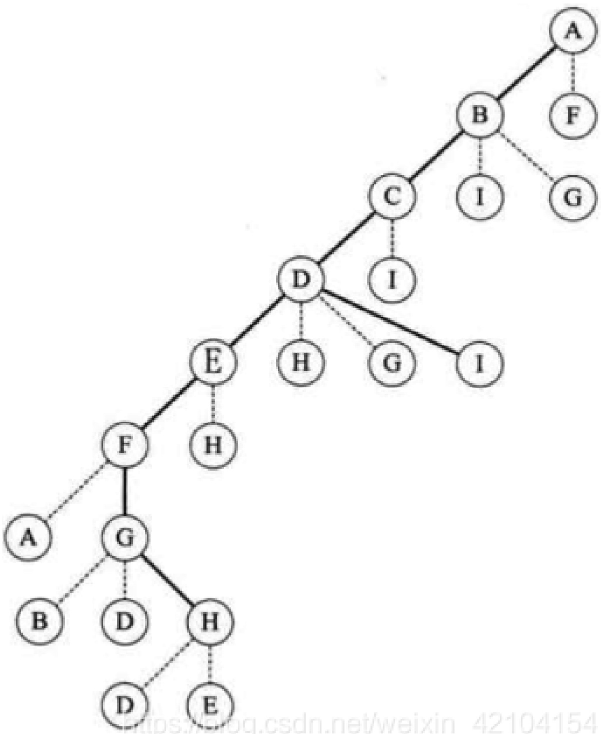
	begin	end	weight
edges[0]	0	4	6
edges[1]	1	0	9
edges[2]	1	2	3
edges[3]	2	3	5
edges[4]	3	4	1
edges[5]	2	0	2

图的遍历

从图中某一顶点出发访遍图中其余顶点，每个顶点仅被访问一次。 两种遍历方式 1）深度优先遍历 2）广度优先遍历 **1）深度优先遍历**

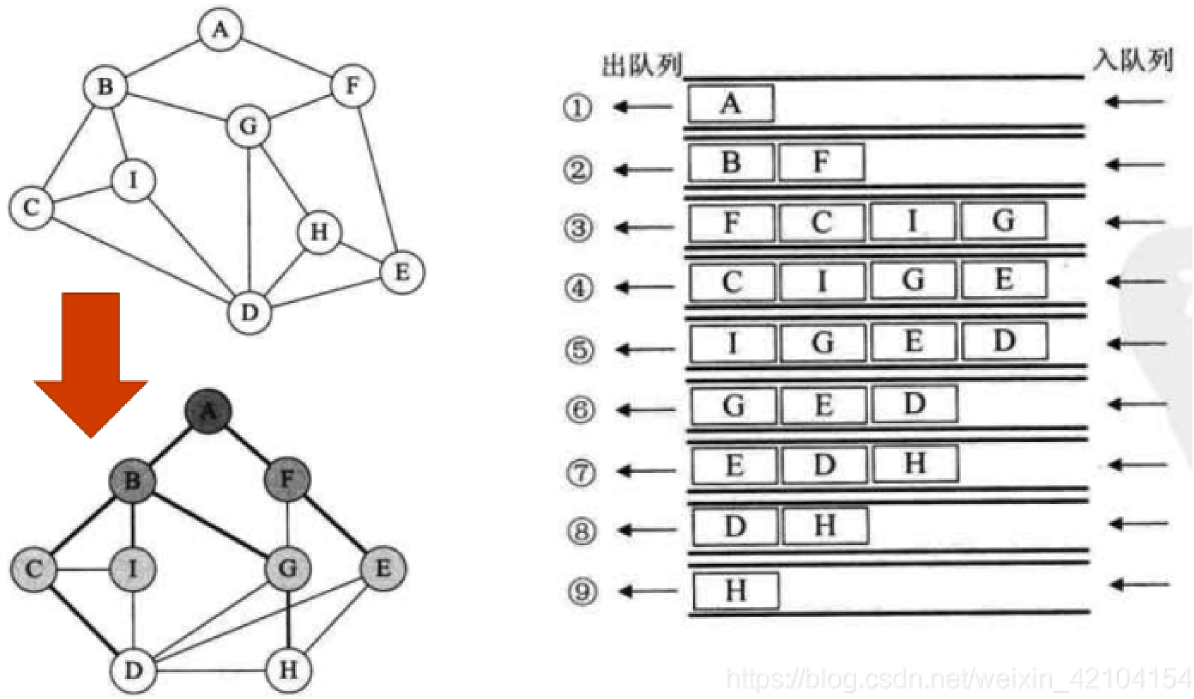
- 深度优先遍历，也叫深度优先搜索，简称DFS





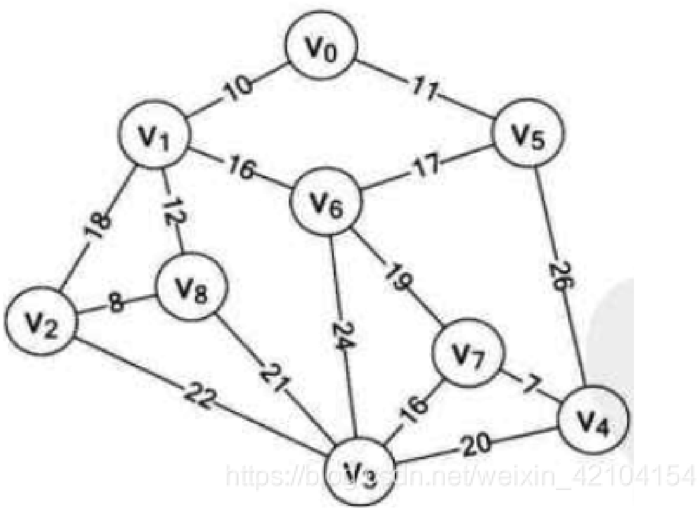
2）广度优先遍历

- 广度优先遍历又称为广度优先搜索，简称BFS



最小生成树

- 一个连通图的生成树是一个**极小连通子图**，它含有图中全部的 **n 个顶点**，但只有足以构成一颗树**的 n-1 条边**



最小生成树性质 1）最小生成树不是唯一的。当图G中的各边权值互不相等时，G的最小生成树是唯一的；若无向连通图的边数比顶点数少1，即G本身是一棵树时，则G的最小生成树就是它本身。 2）最小生成树的边的权值之和总是唯一的，且是最小的。 3）最小生成树的边数为顶点数-1。 **实现算法：** 1）Prim算法 2）Kruskal算法

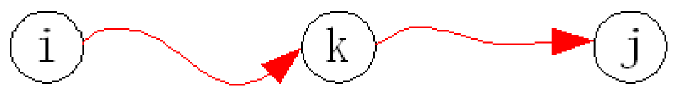
最短路径

> 对于网图，最短路径是两个顶点之间经过的边上权值之和最小的路径 路径上的第一个顶点是原点 路径上最后一个顶点是终点 **最短路径问题** 1）单源最短路径问题 **Dijkstra算法**

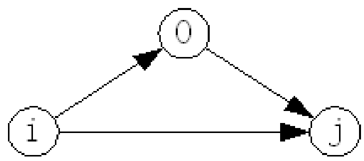
- Dijkstra： 一种按路径长度递增次序求解的算法
- **算法思想：** 设 $M = \{P_{0,i}, P_{0,i}$是v_0到v_i的最短路径 $| i=1\ldots n-1 \}$，
  - 设已知权值$W_{0,k}=\min(\{W_{0,i} \mid <v_0,v_i> \in E\})$， 即$<v_0,v_k>$是$v_0$到其他各顶点的直达弧中最短的；则$(v_0,v_k) \in M$，且$(v_0,v_k)$是M中最短的一条路径！
  - 设$P_{0,k}=(v_0,v_k)$是M中的最短路径，$P_{0,i}$是M中的次短路径，则$P_{0,i}=(v_0,v_i)$或 $P_{0,i}=(v_0,v_k,v_i)$
  - 设已知M的一个子集U，且U中的路径均短于M-U中的路径，设$P_{0,i}$是M-U中的最短的路径，则$P_{0,i}=(v_0,v_i)$或$P_{0,i}=P_{0,k}+\{v_{k,i}\}$，其中$P_{0,k} \in U$

> 2）任意两点间最短路径问题 **Floyd算法**

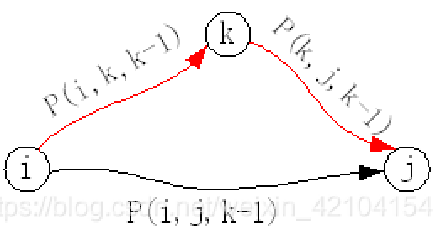
- Floyd 算法思想：
  - 定义 P(i,j,k) 为：从v_i到v_j，由序号不大于 k 的顶点为中间点（或直达）可构成的最短路径



P(i,j,0)



P(i,j,k)



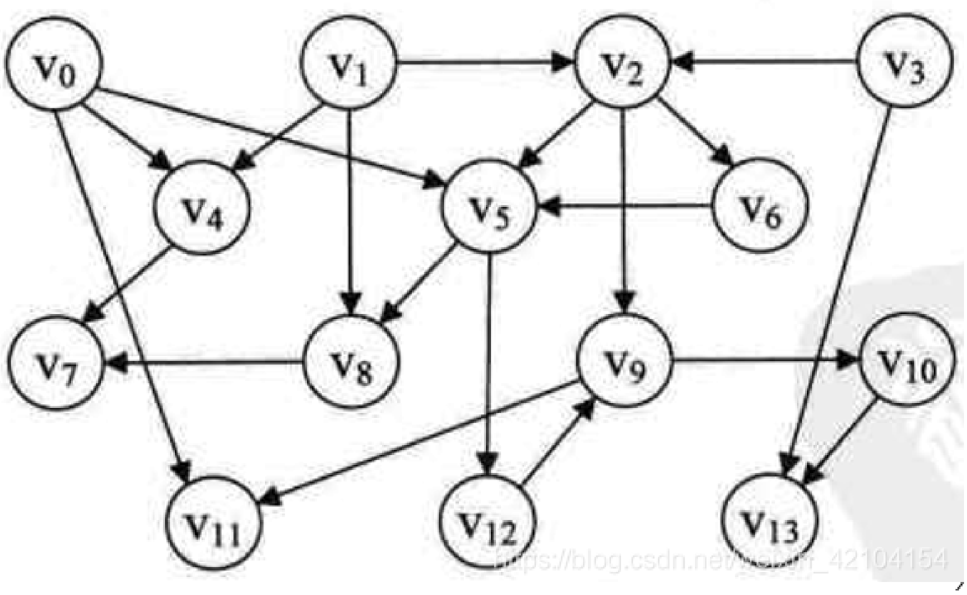
拓扑排序

- 在一个表示工程的有向图中，用顶点表示活动，用弧表示活动之间的优先关系，这样的有向图为顶点表示活动的网，我们称之为**AOV网**（Activity On Vertex Network）

- **拓扑序列**：是具有n个顶点的有向图G=(V,E) 中的顶点序列 v1,v2,…,vn，满足若从顶点 vi 到 vj 有一条路径，则在顶点序列中顶点 vi 比在 vj 之前
- **拓扑排序**：其实就是对一个有向图构造拓扑序列的过程

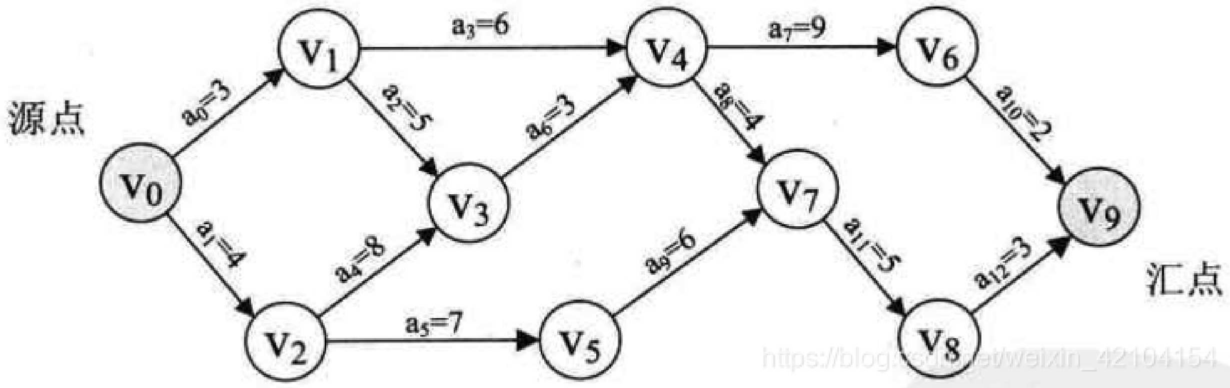
拓扑排序算法

- 从AOV网中选择一个入度为0的顶点输出，然后删去此顶点及以它尾的弧，重复步骤直至输出图中全部顶点。



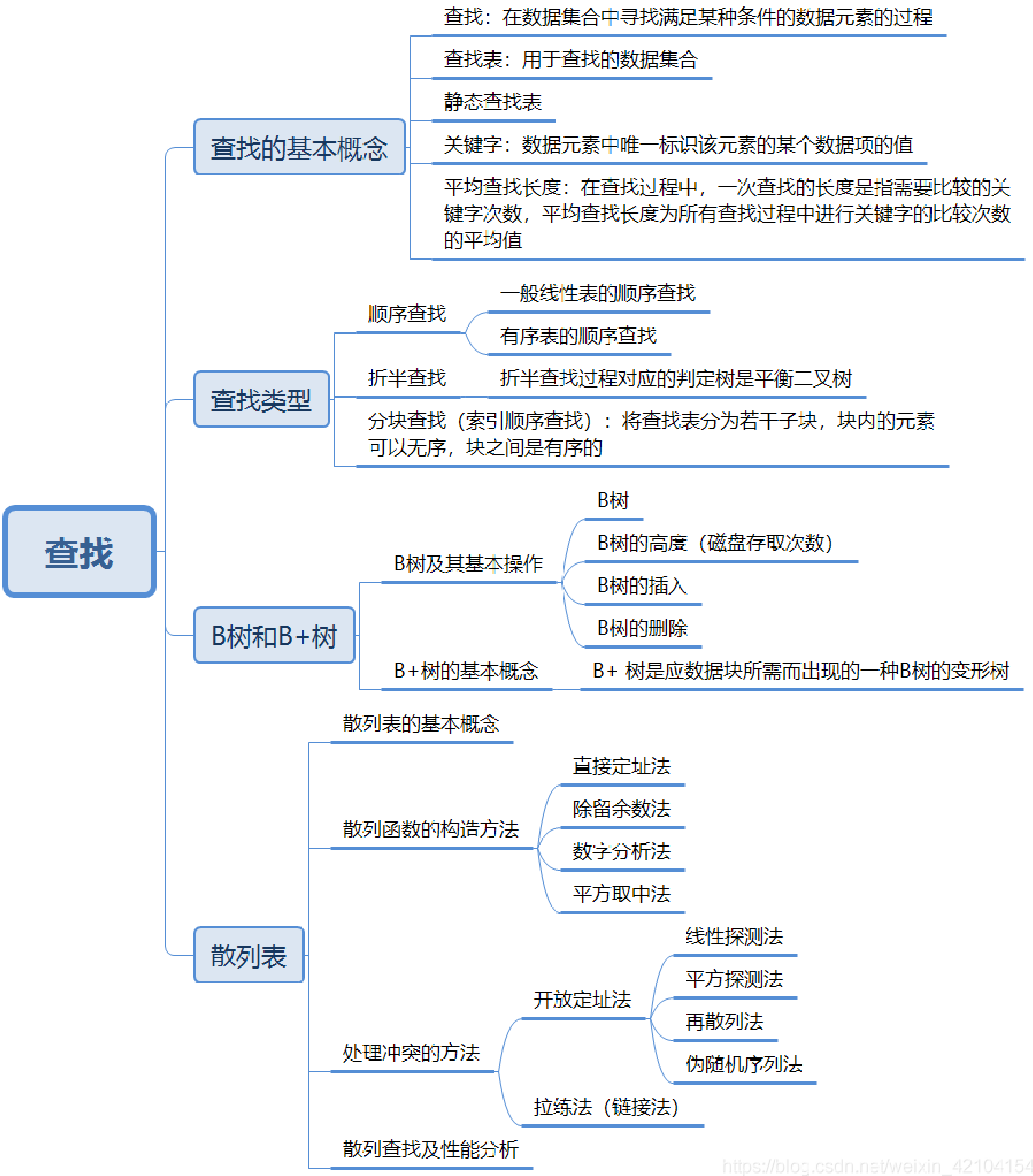
关键路径

- 与AOV对应的是**AOE网**(Activity On Edge Network)，是一个带权的有向无环图，其中顶点表示事件，弧表示活动，权值代表活动持续的时间
  - AOV：顶点代表活动，边代表约束关系，不带权
  - AOE：边代表活动，顶点代表事件或时间点，带权且权值表示活动持续的时间



- **路径长度**：路径上各个活动所持续的时间之和
- **关键路径**：从源点到汇点具有最大路径长度的路径
- **关键活动**：关键路径上的活动

第七章 查找


查找
查找的基本概念
查找：在数据集合中寻找满足某种条件的数据元素的过程
查找表：用于查找的数据集合
静态查找表
关键字：数据元素中唯一标识该元素的某个数据项的值
平均查找长度：在查找过程中，一次查找的长度是指需要比较的关键字次数，平均查找长度为所有查找过程中进行关键字的比较次数的平均值
查找类型
顺序查找
一般线性表的顺序查找
有序表的顺序查找
折半查找
折半查找过程对应的判定树是平衡二叉树
分块查找（索引顺序查找）：将查找表分为若干子块，块内的元素可以无序，块之间是有序的
B树和B+树
B树及其基本操作
B树
B树的高度（磁盘存取次数）
B树的插入
B树的删除
B+树的基本概念
B+ 树是应数据块所需而出现的一种B树的变形树
散列表
散列表的基本概念
散列函数的构造方法
直接定址法
除留余数法
数字分析法
平方取中法
处理冲突的方法
开放定址法
线性探测法
平方探测法
再散列法
伪随机序列法
拉练法（链接法）
散列查找及性能分析

B树

B树的性质

- B树是一种平衡的**多路查找树**，节点最大的孩子数目成为B树的**阶**
- m阶B树的性质（可为空树）
  - 每个结点最多有 m 颗子树
  - 如果根节点不是叶子结点，则至少具有两颗子树
  - 每个非根的分支结点都有k-1个元素和k颗子树，其中$\lceil m/2 \rceil \le k \le m$，叶子节点只有k-1个元素
  - 所有分支结点的结构：$n \mid A_0 \mid K_0 \mid A_1 \mid K_1 \mid \ldots \mid A_{n-1} \mid K_{n-1} \mid A_n$
    - K_i为关键字，且$K_i<K_{i+1}$，A_i为子树指针
    - A_{i-1}子树的结点关键字均小于K_i
    - A_n子树上的结点关键字均大于 K_n
  - 所有叶子结点均在同一层，且不存放任何信息，可看作查找失败的标志，可用空指针实现

B树的查找 从根结点出发，先在结点上折半查找k，若找到，则查找成功，否则 若Ki<K<Ki+1，则沿Ai子树继续查找； 若K>Kn,则沿An子树继续查找； 若已到达叶子结点，则查找失败。 **B树的插入（分裂调整）** 通过查找操作，定位待插入的结点，再向结点上插入新记录。若结点中记录个数超过m-1个，则进行分裂调整。 将结点的中间记录抽出，将结点分裂为两个结点 将中间记录加入到其父结点中 继续对父结点做出调整，当根结点被分裂调整后，生成新的根结点 **B树的删除**

- **设被删除的记录的关键字为K_i，若被删除记录的结点不是最后一层非叶子结点，则可从A_i所指向的子结点中取最大的记录来顶替K_i的位置（相当于从下一层中删除了最大记录）；依此递推，直到最下一层非叶子结点；**
- **当在最下层非叶子结点上删除记录K_i后，记录个数不少于$\lceil m/2 \rceil - 1$时，合并A_i和A_{i+1}，删除完成；**
- **当在最下层非叶子结点上删除记录后，记录个数少于$\lceil m/2 \rceil - 1$时：**
  1. **若其左兄弟（或右兄弟）中的记录个数大于$\lceil m/2 \rceil - 1$时，则将其左兄弟中的最大记录（或右兄弟中的最小记录）移至其父结点中，而将其父结点中的小于（或大于）上移记录的最大记录（或最小记录）下移至被删除记录的结点中；即从其兄弟结点中“借”记录；**
  2. **若兄弟结点无可“借”记录（仅有$\lceil m/2 \rceil - 1$条记录），则将该结点与其兄弟结点及父结点上的相关记录合并成一个结点（相当于从父结点中删除了相关记录）；若父结点因此记录个数少于$\lceil m/2 \rceil - 1$时，则重复步骤1，对父结点做出调整；**

B+树

> B+树是应数据库所需而出现的一种B树的变形树。 一棵m阶B+树所满足的条件： 1）每个分支结点最多有m颗子树（孩子结点）。 2）非叶根结点至少有两棵子树，其他每个分支结点至少有[m/2]颗子树。 3）结点的子树个数与关键字个数相等。 4）所有叶结点个数与关键字个数相等。 5）所有分支结点中仅包含它的各个子结点中关键字的最大值及指向其子结点的指针。

m阶B树与m阶B+树的区别

> 1）在B+树中，具有n个关键字的结点只含有n颗子树，即每个关键字对应一棵子树；在B树中，具有n个关键字的结点含有n+1颗子树。 2）在B+树中，每个结点（非根内部结点）的关键字个数n的范围是[m/2]<=n<=m（根结点：1<=n<=m);在B树中，每个结点（非根内部结点）的关键字n的范围为[n/2]-1<=n<=m)(根结点：1<=n<=m-1)。 3）在B+树中，叶结点包含信息，所有非叶结点仅起索引作用，非叶结点中的每个索引项只含有对应子树的最大关键字和指向该子树的指针，不含有该关键字对应记录的存储地址。 4）在B+树中，叶结点包含全部关键字，即在非叶结点中出现的关键字也会出现在叶结点中；在B树中，叶结点包含的关键字和其他结点包含的关键字是不重复的。 5）B+树可以进行顺序查找，B树不支持顺序查找。

第八章 排序

知识网图

知识网图 链接: https://download.csdn.net/download/weixin_42104154/16486739.