数据结构全书知识梳理总结

第一章 绪论

- 数据就是指能够被计算机识别、存储和加工处理的信息的载体。
- 数据元素是数据的基本单位，有时一个数据元素可以由若干个数据项组成。数据项是具有独立含义的最小标识单位。如整数这个集合中，10 这个数就可称是一个数据元素.又比如在一个数据库(关系式数据库)中，一个记录可称为一个数据元素，而这个元素中的某一字段就是一个数据项。
- 数据结构的定义包括以下三方面内容：逻辑结构、存储结构、和对数据的操作。
- 数据的逻辑结构分为：线性结构、树形结构、复杂结构
- 数据的存储结构分为：顺序表示、链接表示、散列表示、索引表示
- 时间复杂度和渐近时间复杂度：前者是某个算法的时间耗费，它是该算法所求解问题规模 n 的函数，而后者是指当问题规模趋向无穷大时，该算法时间复杂度的数量级。当我们评价一个算法的时间性能时，主要标准就是算法的渐近时间复杂度，因此，在算法分析时，往往对两者不予区分，经常是将渐近时间复杂度 T(n)=O(f(n))简称为时间复杂度，其中的 f(n)一般是算法中频度最大的语句频度。算法中语句的频度不仅与问题规模有关，还与输入实例中各元素的取值相关。但是我们总是考虑在最坏的情况下的时间复杂度。以保证算法的运行时间不会比它更长。

第二章 线性表

- 线性表的逻辑结构特征是很容易理解的，如其名，它的逻辑结构特征就好象是一条线，上面打了一个个结，很形象的，如果这条线上面有结，那么它就是非空表，只能有一个开始结点，有且只能有一个终端结点，其它的结前后所相邻的也只能是一个结点(直接前趋和直接后继)。关于线性表上定义的基本运算，主要有构造空表、求表长、取结点、查找、插入、删除等。
- 线性表的逻辑结构和存储结构之间的关系。在计算机中，如何把线性表的结点存放到存储单元中，就有许多方法，最简单的方法就是按顺序存储。就是按线性表的逻辑结构次序依次存放在一组地址连续的存储单元中。在存储单元中的各元素的物理位置和逻辑结构中各结点相邻关系是一致的。在顺序表中实现的基本运算主要讨论了插入和删除两种运算。对于顺序表的插入和删除运算，其平均时间复杂度均为 O(n)。

- 线性表的链式存储结构。它与顺序表不同，链表是用一组任意的存储单元来存放线性表的结点，这组存储单元可以分布在内存中任何位置上。因此，链表中结点的逻辑次序和物理次序不一定相同。所以为了能正确表示结点间的逻辑关系，在存储每个结点值的同时，还存储了其后继结点的地址信息(即指针或链)。这两部分信息组成链表中的结点结构。对于单链表，其操作运算主要有建立单链表(头插法、尾插法和在链表开始结点前附加一个头结点的算法)、查找(按序号和按值)、插入运算、删除运算等。以上各运算的平均时间复杂度均为 O(n).其主要时间是耗费在查找操作上。
- 循环链表是一种首尾相接的链表。也就是终端结点的指针域不是指向 NULL 空而是指向开始结点(也可设置一个头结点)，形成一个环。采用循环链表在实用中多采用尾指针表示单循环链表。这样做的好处是查找头指针和尾指针的时间都是 O(1)，不用遍历整个链表了。判别链表终止的条件也不同于单链表，它是以指针是否等于某一指定指针如头指针或尾指针来确定。
- 双链表就是双向链表，就是在单链表的每个结点里再增加一个指向其直接前趋的指针域 prior,这样形成的链表就有两条不同方向的链。使得从已知结点查找其直接前趋结点可以和查找其直接后继结点的时间一样缩短为 O(1)。双链表一般也由头指针 head 惟一确定。双链表也可以头尾相链接构成双(向)循环链表。
- 顺序表和链表的比较

具体要求	顺序表	链表
基于空间	适于线性表长度变化不大，易于事先确定其大小时采用。	适于当线性表长度变化大，难以估计其存储规模时采用。
基于时间	由于顺序表是一种随机存储结构，当线性表的操作主要是查找时，宜采用。	链表中对任何位置进行插入和删除都只需修改指针，所以这类操作为主的线性表宜采用链表做存储结构。若插入和删除主要发生在表的首尾两端，则宜采用尾指针表示的单循环链表。

第三章 字符串

- 串就是字符串，是一种特殊的线性表，它的每个结点仅由一个字符组成。
- 空串：是指长度为零的串，也就是串中不包含任何字符(结点)。
- 空白串：指串中包含一个或多个空格字符的串。不同与空串，它的结点就是一个空格字符。
- 在一个串中任意个连续字符组成的子序列称为该串的子串，包含子串的串就称为主串。子串在主串中的序号就是指子串在主串中首次出现的位置。如 A="I love you",B="love", 则 B 在 A 中的序号为 3，注意空格也是字符。
- 空串是任意串的子串，任意串是他自身的子串。
- 串是特殊的线性表(结点是字符)，所以串的存储结构与线性表的存储结构类似。串的顺序存储结构简称为顺序串，顺序串又可按存储分配的不同分为静态存储分配的顺序串和动态存储分配的顺序串。
- 静态的意思可简单地理解为一个确定的存储空间，它的长度是不可变的。如直接使用定长的字符数组来定义一个串。它的优点是涉及串长的操作速度快，因为它的最大长度是不变的。
- 动态存储分配就是在定义串时不分配存储空间，直到需要使用时按所需串的长度分配存储单元给它，并且在运行中还可以根据需要变化串的长度，这就是动态分配。不过这样的串仍是顺序存储的，也就是说指针指向串的首地址，后面的结点是连续存储的。
- 串的链式存储就是用单链表的方式存储串值，串的这种链式存储结构简称为链串。链串与单链表的差异只是它的结点数据域为单个字符。这种存储结构方便于串的插入和删除操作，但是空间利用率不高，因为存放每一个字符要"搭配"一个指向下一字符的地址，而地址所占空间是比较大的。为了解决这种"存储密度"过低的状况，可以让一个结点存储多个字符，事实上这是顺序串和链串的综合(折衷)。
- 子串定位运算又称串的"模式匹配"或"串匹配"，就是在主串中查找出子串出现的位置，这在应用中非常广泛，比如文本编辑中的"查找和替换"用到的就是子串定位运算的算法。

第四章 栈与队列

- 栈的逻辑结构和我们先前学过的线性表相同，如果它是非空的，则有且只有一个开始结点，有且只能有一个终端结点，其它的结点前后所相邻的也只能是一个结点(直接前趋和直接后继),但是栈的运算规则与线性表相比有更多的限制，栈(Stack)是仅限制在表的一端进行插入和删除运算的线性表，通常称插入、删除这一端为栈顶，另一端称为栈底。表中无元素时为空栈。栈的修改是按后进先出的原则进行的，我们又称栈为 LIFO 表(Last In First Out).
- 栈的基本运算有六种：

  构造空栈：InitStack(S)、

  判栈空: StackEmpty(S)、

  判栈满： StackFull(S)、

  进栈： Push(S,x)、可形象地理解为压入，这时栈中会多一个元素

  退栈： Pop(S) 、 可形象地理解为弹出，弹出后栈中就无此元素了。

  取栈顶元素：StackTop(S),不同与弹出，只是使用栈顶元素的值，该元素仍在栈顶不会改变。
- 由于栈也是线性表，因此线性表的存储结构对栈也适用，通常栈有顺序栈和链栈两种存储结构，这两种存储结构的不同，则使得实现栈的基本运算的算法也有所不同。
- 队列也是一种运算受限的线性表，它的运算限制与栈不同，是两头都有限制，插入只能在表的一端进行(只进不出)，而删除只能在表的另一端进行(只出不进)，允许删除的一端称为队尾(rear)，允许插入的一端称为队头 (Front) ,队列的操作原则是先进先出的，所以队列又称作 FIFO 表(First In First Out)
- 队列的基本运算也有六种：

  置空队 ：InitQueue(Q)

  判队空： QueueEmpty(Q)

  判队满： QueueFull(Q)

  入队 ： EnQueue(Q,x)

  出队 ： DeQueue(Q)

  取队头元素： QueueFront(Q),不同与出队，队头元素仍然保留
- 队列也有顺序存储和链式存储两种存储结构，前者称顺序队列，后者为链队。
- 为了克服空间浪费，我们引入循环向量的概念，就好比是把向量空间弯起来，形成一个

头尾相接的环形，这样，当存于其中的队列头尾指针移到向量空间的上界(尾部)时，再加 1 的操作(入队或出队)就使指针指向向量的下界，也就是从头开始。这时的队列就称循环队列。

- 通常我们应用的大都是循环队列。由于循环的原因，光看头尾指针重叠在一起我们并不能判断队列是空的还是满的，这时就需要处理一些边界条件，以区别队列是空还是满。方法至少有三种，一种是另设一个布尔变量来判断(就是请别人看着，是空还是满由他说了算)，第二种是少用一个元素空间，当入队时，先测试入队后尾指针是不是会等于头指针，如果相等就算队已满，不许入队。第三种就是用一个计数器记录队列中的元素的总数，这样就可以随时知道队列的长度了，只要队列中的元素个数等于向量空间的长度，就是队满。
- 队列的链式存储结构称为链队列，一个链队列就是一个操作受限的单链表。为了便于在表尾进行插入(入队)的操作，在表尾增加一个尾指针，一个链队列就由一个头指针和一个尾指针唯一地确定。链队列不存在队满和上溢的问题。在链队列的出队算法中，要注意当原队中只有一个结点时，出队后要同进修改头尾指针并使队列变空。

第五章 二叉树与树

- 树的逻辑结构特征是：树中任一结点都可以有零个或多个直接后继(孩子)结点，但至多只能有一个直接前趋(双亲)结点。树形结构是非线性结构。
- 二叉树的定义：二叉树是 n($n \geq 0$)个结点的有限集，它或者是空集(n=0),或者由一个根结点及两棵互不相交的分别称作这个根的左子树和右子树的二叉树组成。
- 一般二叉树的 3 个重要性质：

  第 i 层至多有 2^{i} 个结点

  高度为 k 的二叉树中，最多有 $2^{k+1}-1$ 个结点

  叶子数＝度 2 结点数－1
- 二叉树的顺序存储结构就是把二叉树的所有结点按照一定次序(从根结点起，从上层到下层，从左往右编号就得到了存放的次序)存储到一片连续的存储单元中
- 用顺序存储方式对于完全二叉树而言其结构简单又节省空间，但是对于一般二叉树并不合适。因此树的存储结构更多的是用链式存储。结点的结构为两个指针域 lchild 和 rchild 分别指向该结点的左孩子和右孩子，另有一个数据域 data 存放结点数据。把所有二叉树的结点，加上一个指向根结点的指针就构成了二叉树的链式存储结构，称为二叉链表。

它就是由根指针 root 唯一确定的。

- 根据访问结点的次序不同可得三种遍历：先序遍历(前序遍历或先根遍历)，中序遍历(或中根遍历)、后序遍历(或后根遍历)。
- 遍历的算法就是一个递归算法，以中序遍历为例，它的定义为

  若二叉树非空，则依次执行如下操作：

  (1) 遍历左子树；

  (2) 访问根结点；

  (3) 遍历右子树。

  将(1)(2)对调则得先序遍历，将(2)(3)对调则得后序遍历。
- 利用二叉链表中的 n+1 个空指针域来存放指向某种遍历次序下的前趋结点和后继结点的指针，这些附加的指针就称为"线索"，加上线索的二叉链表就称为线索链表。
- 二叉树线索化的目的及其实质：利用线索化后的二叉树中的线索就可以直接找到该结点在某种遍历序列中的前趋和后继结点。其实质就是在遍历过程中用线索取代空指针。
- 在中序线索树中查找给定结点的中序前趋和中序后继的方法：若结点*p 的左子树(或右子树非空)，则*p 的中序前趋是从*p 的左孩子开始沿着右指针链往下查找直到找到一个没有右孩子的结点为止，而*p 的中序后继是从它的右孩子开始沿着左指针链往下直到找到一个没有左孩子的结点为止。
- 线索使得查找中序前趋和中序后继变得简单有效，但对于查找指定结点的前序前趋和后序后继并没有什么作用。其原因是查找前序前趋和后序后继结点常常要用到给定结点的双亲结点才能找到，而线索二叉树中的结点没有指向其双亲结点的指针，所以线索对于这两种序列的结点查找并非有效。
- 树和森林及二叉树的转换：三者是唯一对应的，它们之间的转换办法应掌握。（口诀一则）

  树变二叉：兄弟相连留长子。

  林变二叉：树变二叉根相连。

  二叉变树：左孩右右连双亲，去掉原来右孩线。
- 树的存储结构：有双亲链表表示法（就是在每个结点设一指针指向其双亲以唯一地表示任何一棵树，用向量表示），这种表示法中指针是向上链接的，所以对于求指定结点的双亲或祖先十分方便，但不适于求指定结点的孩子及后代。
- 孩子链表表示法（就是为树中每个结点设置一个孩子链表，并将结点及相应的孩子链表

的头指针存放在一个向量中）孩子链表表示便于实现涉及孩子结点及子孙的运算，但不便于实现与双亲有关的运算。因此可以两种表示法结合形成双亲孩子链表表示法。

- 孩子兄弟链表表示法（就是在存储结点信息的同时，附加两个分别指向该结点的最左孩子和右邻兄弟的指针域）这种存储结构的最大优点是，它和二叉树的二叉链表表示完全一样，因此可利用二叉树的算法来实现对树的操作。
- 树的路径长度是从树根到树中每一结点的路径长度之和。在结点数目相同的二叉树中，完全二叉树的路径长度最短。
- 树的代价就是树的带权路径长度，它的值是树中所有叶结点的带权路径长度之和。权就是某结点被赋予的一个实数（这在实际应用中是有某种意义的，比如使用率，数值等）。而树的带权路径长度最小（代价最小）的二叉树就称为最优二叉树（即哈夫曼树）。
- 哈夫曼树的应用最广泛地是在编码技术上，它能够容易地求出给定字符集及其概率分布的最优前缀码。（最优前缀码就是平均码长最小的前缀码）

第六章 集合与字典

- 哈希表的概念

  对于动态查找表而言，1) 表长不确定；2)在设计查找表时，只知道关键字所属范围，而不知道确切的关键字。因此，一般情况需建立一个函数关系，以 H(key)作为关键字为 key 的录在表中的位置，通常称这个函数 H(key)为哈希函数。(注意：这个函数并不一定是数学函数)
- 哈希函数是一个映象，即：将关键字的集合映射到某个地址集合上，它的设置很灵活，只要这个地址集合的大小不超出允许范围即可。
- 由于哈希函数是一个压缩映象，因此，在一般情况下，很容易产生 冲突 现象，即：key1 key2，而 H(key1) = H(key2) 并且，改进哈希函数只能减少冲突，而不能避免冲突。因此，在设计哈希函数时，一方面要考虑选择一个 好 的哈希函数;另一方面要选择一种处理冲突的方法。所谓 好 的哈希函数，指的是对于集合中的任意一个关键字，经哈希函数 映象 到地址集合中任何一个地址的概率是相同的，称这类哈希函数为 均匀的 哈希函数。
- 处理冲突的方法

  处理冲突的实际含义是：为产生冲突的地址寻找下一个哈希地址。

  （1）开放定址法

为产生冲突的地址 H(key)求得一个地址序列： H0, H1, H2, , Hs 1≤s≤m-1

其中：H0 = H(key) Hi = (H(key) + di) MOD m i=1, 2, ,s

增量 di 有三种取法：

1） 性探测再散列

di = c i 最简单的情况 c=1

2） 方探测再散列

di = 12, -12, 22, -22, ,

3） 随机探测再散列

di 是一组伪随机数列

（2）链地址法

将所有哈希地址相同的记录都链接在同一链表中。

线性探测容易产生二次聚集，链地址肯定不会产生二次聚集。一次聚集的产生主要取决于哈希函数，在哈希函数均匀的前提下，可以认为没有一次聚集。

第七章 高级字典结构

- 二叉排序树（二叉查找树）

(1) 定义

二叉排序树或者是一棵空树；或者是具有如下特性的二叉树：

1）若它的左子树不空， 则左子树上所有结点的值均小于根结点的值；

2）若它的右子树不空，则右子树上所有结点的值均大于根结点的值；

3）它的左、右子树也都分别是二叉排序树。通常，取二叉链表作为二叉排序树的存储结构。

(2) 二叉排序树的查找算法

若二叉排序树为空，则查找不成功；否则

1）若给定值等于根结点的关键字，则查找成功；

2）若给定值小于根结点的关键字，则继续在左子树上进行查找；

3）若给定值大于根结点的关键字，则继续在右子树上进行查找。

(3) 二叉排序树的插入算法

对于动态查找表，在查找不成功的情况下，尚需插入关键字等于给定值的记录，并且从查找的过程容易得出插入的算法：

若二叉排序树为空树，则新插入的结点为根结点；否则，新插入的结点必为一个新的叶子结点，其插入位置由查找过程中得到。

(4) 二叉排序树的删除算法

和插入相反，删除在查找成功之后进行，并且要求在删除二叉排序树上某个结点之后，仍然保持二叉排序树的特性。分三种情况：

（1）被删除的结点是叶子；

（2）被删除的结点只有左子树或者只有右子树；

（3）被删除的结点既有左子树，也有右子树。

- 二叉平衡树

这是另一种形式的二叉查找树，其特点为左、右子树深度之差的绝对值不大于 1，称有这种特性的二叉树为平衡树。

- B 树与 B＋树

（1）记忆定义和区别。

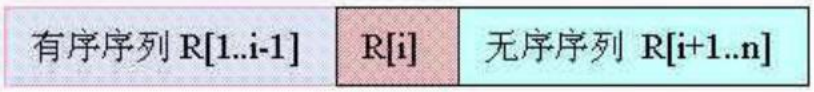
（2）主要理解 B 树的检索、插入和删除。具体请见《数据结构重点图形绘制详细说明》。

第八章 排序

- 插入排序

思想：

假设在排序过程中，记录序列 R[1..n]的状态为：



则一趟直接插入排序的基本思想为：将记录 R[i]插入到有序子序列 R[1..i-1]中，使记录的有序序列从 R[1..i-1]变为 R[1..i]。

完成这个 插入 需分三步进行：

- ◆ 查找 R[i]的插入位置 j+1；
- ◆ 将 R[j+1..i-1]中的记录后移一个位置；
- ◆ 将 R[i]复制到 R[j+1]的位置上

（1）直接插入排序：利用顺序查找实现 在 R[1..i-1]中查找 R[i]的插入位置 的插入排序。

（2）折半插入排序：因为 R[1..i-1]是一个按关键字有序的有序序列，则可以利用折半查

找实现 在 R[1..i-1]中查找 R[i]的插入位置 ，如此实现的插入排序为折半插入排序。

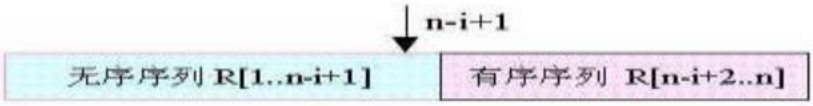
（3）表插入排序：为了减少在排序过程中进行的 移动 记录的操作，必须改变排序过程中采用的存储结构。利用静态链表进行排序，并在排序完成之后，一次性地调整各个记录相互之间的位置，即将每个记录都调整到它们所应该在的位置上。

（4）希尔排序：对待排记录序列先作 宏观 调整，再作 微观 调整。所谓 宏观调整，指的是 跳跃式 的插入排序。即：将记录序列分成若干子序列，每个子序列分别进行插入排序。关键是这种子序列不是由相邻的记录构成的。

- 快速排序

（1）起泡排序

假设在排序过程中，记录序列 R[1..n]的状态为：



思想：借助对无序序列中的记录进行 交换 的操作，将无序序列中关键字最大的记录 交换 到 R[n-i+1]的位置上，实现第 i 趟起泡插入排序。

（2）快速排序

思想：找一个记录，以它的关键字作为 枢轴 ，凡其关键字小于枢轴的记录均移动至该记录之前，反之，凡关键字大于枢轴的记录均移动至该记录之后。致使一趟排序之后，记录的无序序列 R[s..t]将分割成两部分：R[s..i-1]和 R[i+1..t],且 R[j].key≤ R[i].key ≤ R[j].key (s≤j≤i-1) 枢轴 (i+1≤j≤t)。

- 选择排序

（1）简单选择排序

思想：假设排序过程中，待排记录序列的状态为：

并且有序序列中所有记录的关键字均小于无序序列中记录的关键字，则第 i 趟简单选择排序是，从无序序列 R[i..n]的 n-i+1 记录中选出关键字最小的记录加入有序序列。

（2）堆排序

思想：先建一个 大顶堆 ，即先选得一个关键字为最大的记录，然后与序列中最后一个记录交换，之后继续对序列中前 n-1 记录进行 筛选 ，重新将它调整为一个 大顶堆 再将堆顶记录和第 n-1 个记录交换，如此反复直至排序结束。所谓 筛选 指的是对一棵左/右子树均为堆的完全二叉树， 调整 根结点使整个二叉树为堆。

- 归并排序

思想：将两个或两个以上的有序子序列 归并 为一个有序序列，是一种稳定的排序方法。在内部排序中，通常采用的是 2-路归并排序。即将两个位置相邻的有序子序列：

归并为一个有序序列。

第九章 图

- 有向图：顶点对 <x, y> 是有序的，<x, y>≠<y,x>，即 E 中为有向边。
- 无向图：顶点对(x, y)是无序的，(x, y)= (y,x)，即 E 中为无向边。
- 完全图：若无向图中每两个顶点之间都存在着一条边，即有 n(n-1)/2 条边，称完全无向图；若有向图中每两个顶点之间都存在着方向相反的两条边，即有 n(n-1)条边，称完全有向图。
- 权：与图的边或弧相关的数，这些数可以表示从一个顶点到另一个顶点的距离或耗费
- 网：带权的图
- 度：在无向图中是顶点边的数目；在有向图中，顶点的度=入度+出度
- 入度： 有向图中顶点入边的数目
- 出度：有向图中顶点出边的数目
- 路径：从顶点 V 到顶点 V 的一个顶点系列
- 回路：一条路径上的前后两个端点相同

- 邻接点：无向图中一条边(Vi,Vj)的两个端点 Vi,Vj
- 路径长度：路径上经过的边的数目
- 简单路径：一条路径上除了前后端点可以相同外，所有顶点均不相同
- 简单回路：前后两端点相同的简单路径
- 连通图： 无向图中任意两个顶点都连通。如果一个图有 n 个顶点和小于 n-1 条边，则是非连通图。
- 连通分量：无向图的极大连通子图；任何连通图的连通分量只有一个，即其本身；非连通图有多个连通分量。
- 强连通图： 有向图中任意两个顶点都连通
- 强连通分量：有向图的极大强连通子图；任何强连通图的强连通分量只有一个，即其本身；非强连通图有多个强连通分量。
- 生成树： 在一个连通图 G 中，如果取它的全部顶点和一部分边构成一个子图 G ,V(G)=V(G) E(G)∈E(G), 若边集 E（G ）中的边将图中所有的顶点连通又不形成回路，则称子图 G 是图 G 的一棵生成树。一棵有 n 个顶点的生成树有且仅有 n-1 条边，但有 n-1 条边的图不一定是生成树。如果它多于 n-1 条边，则一定有环。
- 最小生成树：具有权最小的生成树
- 有向树：一个有向图恰有一个顶点的入度为 0，其余顶点的入度均为 1。
- 邻接矩阵（Adjacency Matrix）的存储结构，就是用一维数组存储图中顶点的信息，用矩阵表示图中各顶点之间的邻接关系。
- 邻接表(Adjacency List)是图的一种顺序存储与链式存储结合的存储方法。邻接表表示法类似于树的孩子链表表示法。就是对于图 G 中的每个顶点 vi，将所有邻接于 vi 的顶点 vj 链成一个单链表，这个单链表就称为顶点 vi 的邻接表，再将所有点的邻接表表头放到数组中，就构成了图的邻接表。
- 图的遍历：从图中某个顶点出发访遍图中其余顶点，并且使图中的每个顶点仅被访问一次过程。
- 深度优先搜索：从图中某个顶点 V0 出发，访问此顶点，然后依次从 V0 的各个未被访问的邻接点出发深度优 先搜索遍历图，直至图中所有和 V0 有路径相通的顶点都被访问到，若此时图中尚有顶点未被访问，则另选图中一个未曾被访问的顶点作起始点，重复上述过程，直至图中所有顶点都被访问到为止。
- 广度优先搜索：从图中的某个顶点 V0 出发，并在访问此顶点之后依次访问 V0 的所有

未被访问过的邻接点，之　后按这些顶点被访问的先后次序依次访问它们的邻接点，直至图中所有和 V0 有路径相通的顶点都　被访问到。若此时图中尚有顶点未被访问,则另选图中一个未曾被访问的顶点作起始点，重复上　述过程，直至图中所有顶点都被访问到为止。

- 最小生成树算法：

（1）Prim 算法：

Prim 算法构造最小生成树的过程为：在所有　其一个顶点已经落在生成树上，而另一个顶点尚未落在生成树上 的边中取一条权值为最小的边，逐条加在生成树上，直至生成树中含有 n-1 条边为止。

（2）Kruskal 算法：

Kruskal 算法构造最小生成树的过程为：先构造一个只含 n 个顶点、而边集为空的子图，把子图中各个顶点看成各棵树上的根结点，之后，从网的边集 E 中选取一条权值最小的边，若该条边的两个顶点分属不同的树，则将其加入子图，即把两棵树合成一棵树，反之，若该条边的两个顶点已落在同一棵树上，则不可取，而应该取下一条权值最小的边再试之。依次类推，直到森林中只有一棵树，也即子图中含有 n-1 条边为止。

- 有向无环图（directed acycling graph）：一个无环的有向图 简称 DAG 图。有向无环图是描述一项工程进行过程的有效工具。主要进行拓扑排序和关键路径的操作。
- 拓扑排序：由某个集合上的一个偏序得到该集合上的一个全序的操作。拓扑排序实际上是对邻接表表示的图进行遍历的过程。
- AOV 网（Activity On Vertex Network）：顶点表示活动的网，即用顶点表活动，用弧表示活动间优先关系的有向图。
- AOE 网（Activity On Edge Network）：是一个带权的有向无环图，弧表示活动，权表示活动持续的时间，可以用来估算工程的完成时间。
- 关键路径：AOE 网中带权路径最长的路径。
- AOV 网的拓扑排序：在顶点活动网 AOV 网中，若不存在回路，则所有活动可排列成一个线性序列，使得每个活动的所有前驱活动都排在该活动的前面，该序列叫拓扑序列，由 AOV 网构造拓扑序列的过程叫做拓扑排序。AOV 网拓扑序列不是唯一的。
- 拓扑排序的算法：选一个入度为 0 的顶点输出，并将其所有后继顶点的入度　1，重复这两步直至输出所有顶点，或找不到入度为 0 的顶点为止，为便于查找入度为 0 的顶点，

算法中利用顶点的入度域建立一个存放入度为 0 的顶点的栈。