sort排序：

ascending order 升序

descending order 降序

each entry 每个条目

sorting algorithm is stable 排序算法稳定

Insertion sort and Shell sort 插入排序与

Selection Sort and HeapSort 选择排序与堆排序

MergeSort and Bucket Sort

the number of comparisons of keys 键值的比较次数

the number of times entries must be moved inside a list 条目在列表中移动的次数

a sorted sublist and a un-sorted sublist 一个排序子表，一个未排序的子表

Continue in this manner 持续这种方式

interchange 交换位置

strategy策略

on each pass 在每次遍历中

be regard考虑

Convert

Percolate-Down algorithm

minimum spanning tree 最小生成树

Courseware 课件

**TAKE PRIDE IN THE WORK YOU DO!!! DON'T CHEAT.**

**entry**

**DS can be defined in mathematics**

The estimation of the running time of algorithms算法运行时间的估计

platform 平台

upper bound上限

A is asymptotically faster than B a比b快

p(N) grows strictly faster than T(N) p(N)严格地比T(N)增长得快

if T1(N) = O(f(N)) and T2(N) = O(g(N)) then

a) T1(N) + T2(N) = max (O(f(N)), O(g(N)))

b) T1(N) \* T2(N) = O(f(N) \* g(N))

infinitely fast

evaluate

number of iterations 迭代次数

Nested Loops嵌套循环

Consequtive Statements

recursive call 递归调用

Algorithm for fibonacci series (斐波纳契数列)

finite sequence有限序列

**consecutive** storage units连续的存储单元

**predecessor**

**successor**

* + store items "sequentially" without restrictions on location
* Cursor operation simulates the features
* Polynomials(多项式相加)

Applications of stacks  
 (I) Balancing Symbols (Bracket Matching)  
 (II) Postfix Expressions  
 (III) Infix to Postfix Conversion

栈的应用

(I)平衡符号(括号匹配)

(2)后缀表达式

(3)中缀到后缀的转换

(4)进制数转换

(5)函数调用

**recursion**

parenthesis are necessary

* Recursion should be used freely in the initial design of algorithms.It is especially appropriate where the main step toward solution consists of reducing a problem to one or more smaller cases.

在算法的初始设计中应充分利用递归。当解决问题的主要步骤是将问题简化为一个或多个更小的案例时，这种方法尤其适用。

**Q.rear+1)%Q.queuesize==Q.front(front和rear都为int类型**

**4. 在具有n个单元的循环队列中，队满时共有 n-1 个元素。**

**finite sequence of characters**

**Concat(&T,S1,S2)//String Concatenation**

**Initial：String S1 and S2。**

**Output：Concatenate S1 and S1 and put the new string into T .**

**SubString (&Sub, S, pos, len)**

**SubString(&sub, ‘data structure’, 6, 9) Sub=‘structure’**

**Index (S, T, pos)**

**S = ′abcaabcaaabc′, T = ′bca′ Index(S, T, 3) = 6**

**Replace (&S, T, V)**

**S=‘abcaabc’, T=‘ab’, V=‘x’ S=‘xcaxc’**

**StrDelete (&S, pos, len)**

**S= ‘structure’ StrDelete( ＆S, 1, 5) S=‘ture’**

**index(S,T,pos)**

* **设n = StrLength(S);m = StrLength(T);**
* **最好情况的复杂度为O(n+m),如**
  + **T = “STING”**
  + **S = “A STRING SEARCHING EXAMPLE CONSISTING OF SIMPLE TEXT”**
* **最坏情况的复杂度为O(n\*m),如**
  + **T = “00000001”**
  + **S = “00000000000000000000000000000000000000000000000001”**

**4. 子串的定位运算称为串的模式匹配； 被匹配的主串 称为目标串， 子串 称为模式串。**

**Level of the root of a tree is 1,**

* **A *full binary tree* (满二叉树） of depth *k* is a binary tree of depth *k* having 2k -1 nodes.**
* Complete Binary Tree 对于深度为K的，有n个结点的二叉树，当且仅当其每一个结点都与深度为K的满二叉树中编号从1至n的结点一一对应时称之为完全二叉树。

若设二叉树的深度为h，除第 h 层外，其它各层 (1～h-1) 的结点数都达到最大个数，第 h 层所有的结点都连续集中在最左边，这就是完全二叉树。

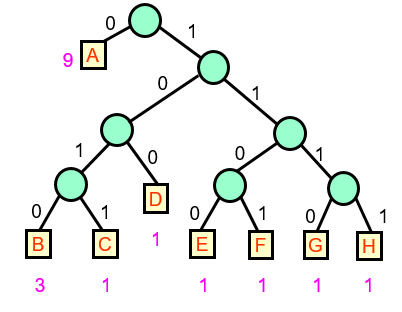
**For any nonempty binary tree, T, if n0 is the number of leaf nodes and n2 is the number of nodes of degree 2, then n0 = n2 + 1.**

* **A Threaded Binary Tree**
* **Tree traverse in preorder 🡸🡺 Corresponding binary tree traverse in preorder.**
* **Tree traverse in postorder 🡸🡺 Corresponding binary tree traverse in Inorder.**

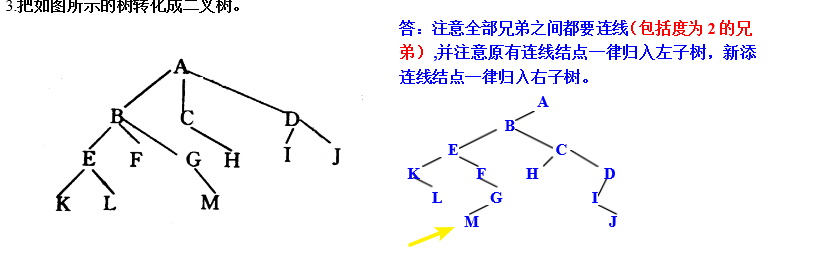
***Huffman Tree***

***Path Length: The length of path is the number of edges on the path.***

**the binary tree with the *minimum* weighted path length.**

**路径唯一,加权平均长度最小**

**用5个权值{3, 2, 4, 5, 1}构造的哈夫曼（Huffman）树的带权路径长度是 33**



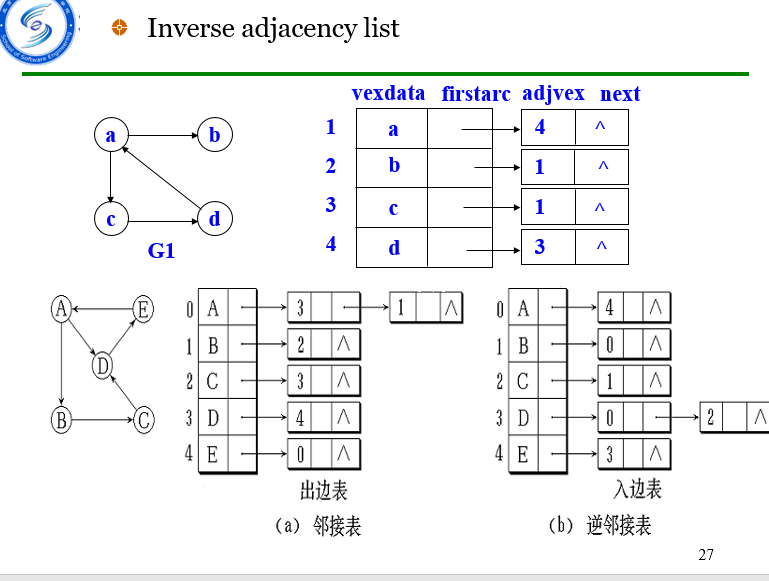
**set of vertices V(G)**

**A directed graph**

**for undirected graph with n vertices, the maximum number of edges is n(n-1)/2**

**for directed graph with n vertices, the maximum number of edges is n(n-1)**

* **If the edges of a graph is e<nlog(n), the graph is called Sparse graph ,otherwise, it is called as Dense Graph**
* **the adjacency matrix for a digraph need not be symmetric**
* **the adjacency matrix for a digraph need not be symmetric**



* + **depth-first search**
  + **breadth-first search**
* ***level by level.***

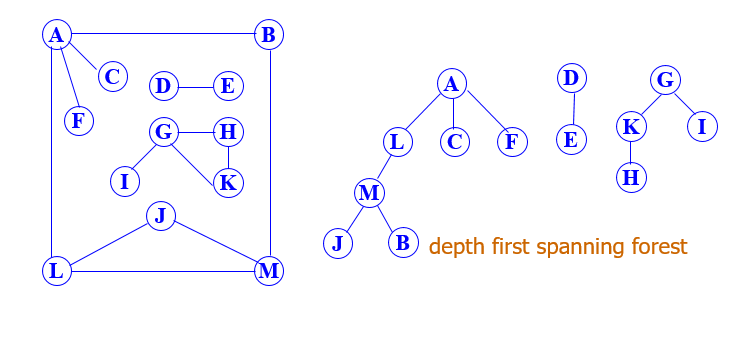
**Visit the start vertex *v***

**connected graph**

***Minimal Cost Spanning Trees　  
(最小代价生成树）***

**When DFS is used, the resulting spanning tree is known as a depth first spanning tree**

**When BFS is used, the resulting spanning tree is known as a breadth first spanning tree**

****

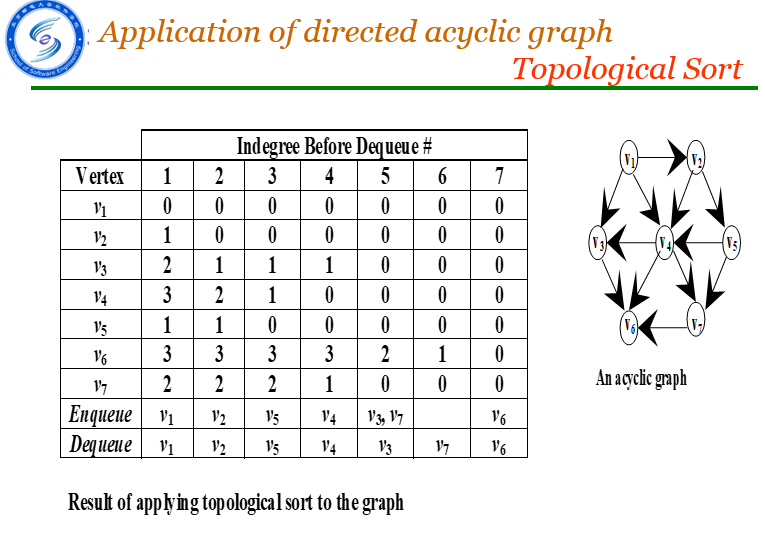
**Prim**

**Kruskal 生成最小代价生成树 图变成树**

**Application of directed acyclic graph**

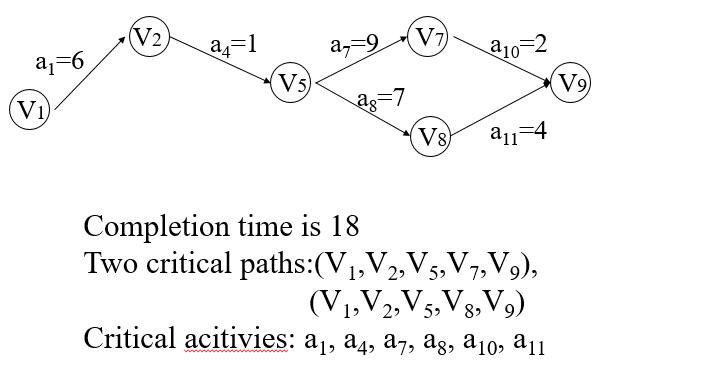
* + **Topological sort(判断图中是否有循环,检查入度为0点,遍历,删除,再找,**

**Decrement the indegree of all adjacent vertices.** **Running time is Ο(|*E*|+|*V*|).**

****

* + **Critical path. AOE (Activity on Edge)**

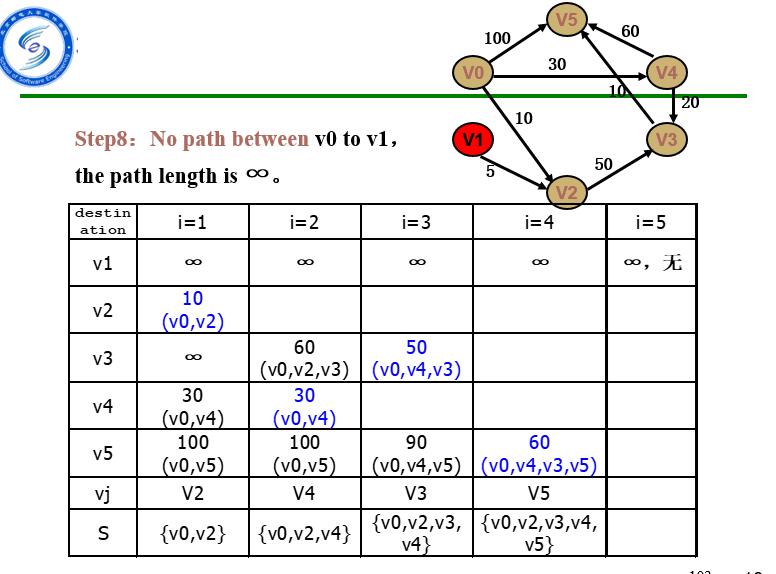
**Duration time of an activity.**

****

**the Time Complexity of whole algorithm is O(n+e)**

**# optimal path *Shortest Path***

***Dijkstra Algorithm***



**（ d ）13. 广度优先遍历类似于二叉树的**

**A．先序遍历 B. 中序遍历 C. 后序遍历 D. 层次遍历**

**2. 有向图G用邻接表矩阵存储，其第i行的所有元素之和等于顶点i的 出度 。**

**10. 用普里姆(Prim)算法求具有n个顶点e条边的图的最小生成树的时间复杂度为 n平方 ；用克鲁斯卡尔(Kruskal)算法的时间复杂度是 nlog2n 。**

***Sequential Search Analysis 顺序搜索分析***

***Indexing Search索引搜索***

**outcomes of the comparison.**

**nonempty left subtree 二叉搜索树最糟糕的情况 (n+1)/2 最好情况 log2(n+1)-1**

* + **Binary search tree**
* **AVL tree 有平衡因子**

**Four standard types of rotations are performed on AVL trees**

**左左型:: 把孩子给祖先,自己顶替祖先,执行一次从-+1开始,**

**左右型::****把孩子给祖先,自己顶替祖先,执行两次 从0开始**

**handle collisions affect the performance of searching**

***Linear Probing***

**3. 假设在有序线性表a[20]上进行折半查找，则比较一次查找成功的结点数为1；比较两次查找成功的结点数为 ；比较四次查找成功的结点数为 2的k-1次方 ；平均查找长度为(\_\_\_\_\_\_) 。(1+4+12+32+25)/20**

**（ c ）5. 折半搜索与二叉搜索树的时间性能**

**A. 相同 B. 完全不同 C. 有时不相同 D. 数量级都是O（log2n）**