1. #id .class element element#id element.class [attr] [attr=”value”] [attr\*=”value”] element[attr] element[attr=”value”] element[attr\*=”value”] ,group descendant >children ~all siblings +first sibling
2. snbao: String Number Boolean Array Object
3. fwghso:From Where Group Having Select Orderby
4. avl: |size(node.left)-size(node.right)|<=1
5. treap: assign node a priority, node.priority<=node.left.priority node.priority<=node.right.priority; node.right.key>=node.key>=node.left.key; the node are inserted as if they comes in sequence of priority.
6. Sbt: size(node.left)>=size(node.right.left) size(node.right.right) size(node.right)>=size(node.left.right) size(node.left.left)
7. Splay tree: move the lasted visited node to the root
8. Rbtree:each node can have one color:red or black; if a node is red, then its children or parents can’t be red. All paths from a node to its leaf have same black depth.
9. B tree: define t (t>=2) as min degree; each nonroot node has [t-1,2t-1] keys, has [t,2t] children. All leaf are at same depth.
10. 霍夫曼树：为使不等长编码为前缀编码(即要求一个字符的编码不能是另一个字符编码的前缀)，可用字符集中的每个字符作为叶子结点生成一棵编码二叉树，为了获得传送报文的最短长度，可将每个字符的出现频率作为字符结点的权值赋予该结点上，显然字使用频率越小权值越小，权值越小叶子就越靠下，于是频率小编码长，频率高编码短，这样就保证了此树的最小带权路径长度效果上就是传送报文的最短长度。因此，求传送报文的最短长度问题转化为求由字符集中的所有字符作为叶子结点，由字符出现频率作为其权值所产生的哈夫曼树的问题。利用哈夫曼树来设计二进制的前缀编码，既满足前缀编码的条件，又保证报文编码总长最短。
11. Skip table:possibility P, random number <= P, add height.
12. Dynamic programming: optimal subproblems & subproblems override
13. Greedy algorithm:choose the best choice for now&hope it works for all.
14. Heap sort quick sort merge sort insertion sort selection sort bucket sort counting sort radix sort
15. Positon: static relative absolute fixed; 普通流 浮动 绝对定位
16. 后缀表达式: 计算一个表达式（中缀表达式）1）创建一个栈和后缀队列，读取中缀表达式序列，如果是（，加入到栈，2）如果是数字，加入后缀表达式3）如果是）（右括号），如果栈非空，弹栈并将栈顶加入到后缀队列，直到遇到(左括号（注：左括号弹栈，但不加入后缀表达式），4)如果是操作符， 如果栈为空，加入栈，否则查看栈顶，如果栈顶是(或者新扫描的元素优先级高（需要先计算），则入栈；否则弹出所有大于等于当前优先级的所有元素加入到后缀表达式，直到遇到比当前优先级低的元素或者（
17. 计算后缀表达式，建一个栈，读取到操作数压入栈，如果读取到n元运算符，（后弹栈-先弹栈），取出栈顶n个操作数运算，将结果存入栈。重复，直至扫描完整个表达式，弹出栈顶数值为结果。
18. Graph:adj matrix, adj table; depth first search: marked it, check all edges it connect, if unmarked, recursively search it. Breadth first search:create a queue, add the start search point&marked it; while the queue is not empty, dequeue a vertex, search all the edges it connects to, if a edge unmarked, mark it & add to the queue
19. CC: for each vertex, if unmarked, depth first search a connected component.
20. No cycle: if a connected component has no cycle, then the number of its vertices == the number of edge + 1
21. Bipartite：depth search & color
22. Directed graph:adjs[u] 所有从u离开的边的点的集合（所有u可以到达的点的集合）
23. 深度优先搜索中，u.d表示u被搜索起始的时间点，u.f表示搜索的终止点。[U.d,u.f]和[v.d,v.f]要么包含，要么不相交。如果v是u的后代，在[v.d,v.f]被包含在[u.d,u.f].
24. 深度优先搜索中的边分为a)树边，(u,v)是树边，如果搜索u的邻接矩阵时，v是白色的。B)back edge(回边) （u,v）是u连接它的祖先v，自环认为是回边。回边创造了环。可以通过edgeto,找到这个环。如果搜索u的邻接矩阵时，v是灰色的。（u,v是回边）c)前边 forward edge, 是非树边，但u连接到后代v d)cross edge 叉边 在同一个树中，u,v没有祖先后代关系 或者在不同的树中，边跨树只能出现在有向图中，a树先被搜索了，b树的某个点指向a树的点。如果搜索u的邻接矩阵时，v是黑色的。那么(u,v)是前向边或叉边。如果u.d<v.d，那么（u,v）是前边，如果u.d>v.d，那么是叉边。深度搜索时灰点的数目是深度搜素的深度+1。
25. 深度搜索一个无向图，所有的边是树边或者回边。不可能是前向边或者叉边。
26. 拓扑排序只能出现在有向无环图中。如果存在（u,v）,那么u出现在v之前。深度优先搜索，如果one vertex is finished, add it to the head of linked list.按照u.f反向排序、
27. 一个图是无环的，如果没有回边。

计算有向图的强链接组件：1）调用dfs计算u.f for each vertex u, （可使用栈）2) compute g的转置矩阵，3）计算dfs(G**T**),但是以u.f的降序来调用dfs,4) output the vertices of each tree in the depth-first forest formed in 3 as a strong connected component.

证明：一：当前栈顶元素X与其搜到的所有的点Yi构成了一个强连通分量。

假设在反图的dfs中（也就是第二次dfs）从X搜到了Y，那么说明原图中有Y到X的路径。

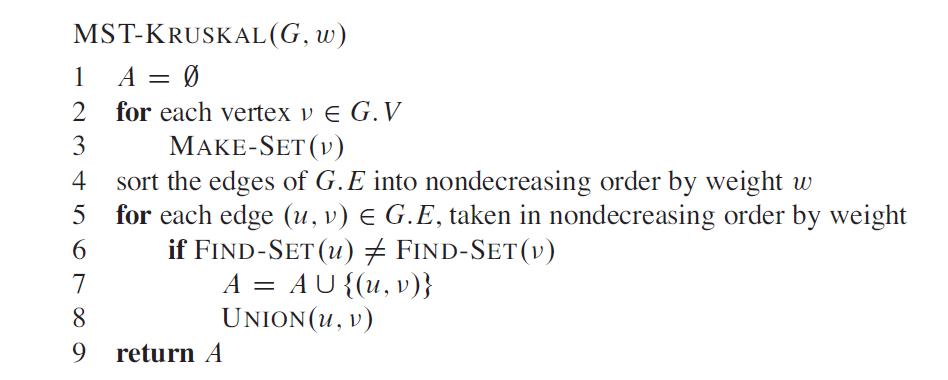
既然X搜到了Y，说明，X比Y后出栈，只能有这样的解释：Y是X搜索树中的一点（原图中），也就是说原图中有X到Y的一点，再由上得X Y在一个强连通分量中。

其实X比Y后出栈理论上还有一种可能，就是Y 和 X是并行的两棵搜索树的根（原图中），Y先搜完了再去搜X的子树，但是由于已经知道了原图中有Y到X的路径，因此这种情况是不可能出现的。

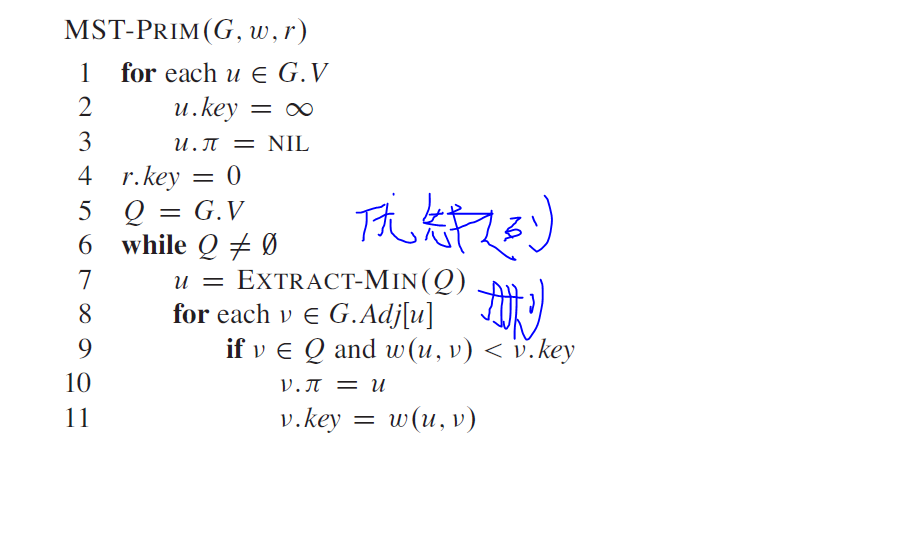
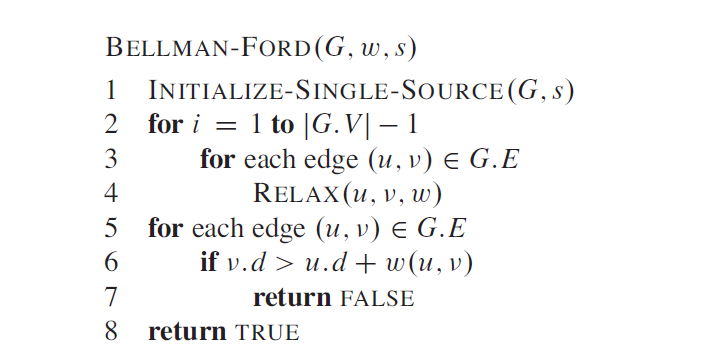
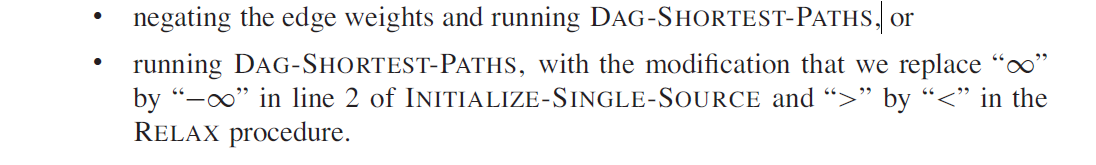
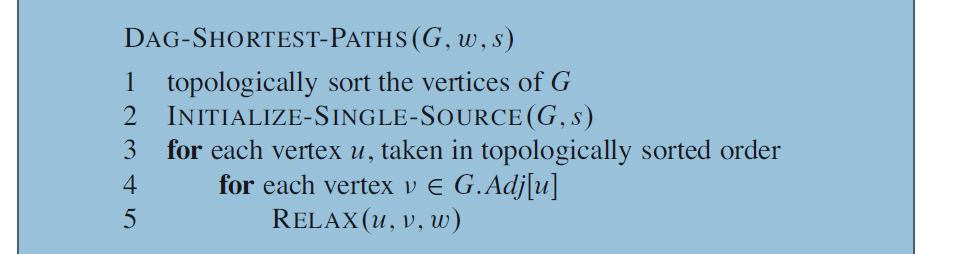
二：不存在这样的两个点X、Y，他们能互相到达，但是分别位于两个不同的连通分量，也就是说上述算法搜出来的所有连通分量都是极大的强连通分量。

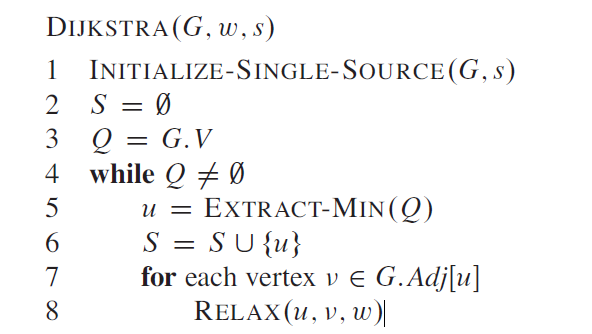
这个证明我感觉超简单，因为第二次dfs不管X与Y谁先出来搜，一个点肯定能搜到另外一个点，他们肯定就会在一个连通分量内了！

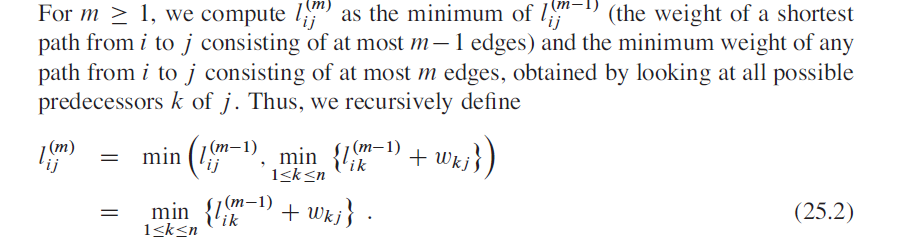
1. 最小生成树：在无向有权图中，最大的连通分量且权最小。G是无向连接图，权函数w。A是边E的子集，A是最小生成树的子集。（s,V-s）是一个G的分割并且respect A，（u,v）是分割中最小的边。（u,v）is safe for A.
2. Kruskal 最小生成树算法：



It runs at OElgv

1. Prim 算法：
2. Prim’s algorithm runs at Elg(v)
3. 单源最短路径，在有向有权图中，查找最短路径。单源最短路径的变种：a)单源目的路径；b) Single-pair shortest-path problem:c) **All-pairs shortest-paths problem:**
4. 单源最短路径的最优结果：a-🡪 …🡪b🡪c是最短路径，则a🡪…🡪b也是最短路径。
5. Bellman-Ford算法：可以存在负边，并且可以判断是否存在负环，如果存在负环，则最短路径不存在，如果不存在负环，则可以计算出最短路径。It runs at O(VE)
6. 有向无环图的单源最短路径。It runs at O(V+E).可以有负边。这个算法既可以找最短路径，也可以找最长路径。
7. Dijkstra算法适用于所有的边是非负的的有向图。Dijkstra’s algorithm maintains a set of S of vertices whose final shortest-path weights from source s have already been determined. The algorithm repeatly select the vertex u 属于 V-Swith the minimum shortest-path eatimate, adds u to S, and relaxes all edges leaving u. use a min priority queue Q of vertices, keyed by their d values.

将[图](http://baike.baidu.com/view/143347.htm" \t "_blank)G中所有的顶点V分成两个顶点集合S和T。以v为源点已经确定了最短路径的终点并入S[集合](http://baike.baidu.com/view/15216.htm)中，S初始时只含顶点v,T则是尚未确定到源点v最短路径的顶点集合。然后每次从T集合中选择S集合点中到T路径最短的那个点，并加入到集合S中，并把这个点从集合T删除。直到T集合为空为止。

1. 多源最短路径，算法的结果W=(wij)，i=j，则wij=0；从i到j的最短距离；或者∞，如果从i到j没有路径。先驱矩阵M=(mij),mij是null,如果i=j，或者从i到j没有路径。或者从i到j的最短路径中j的前驱。
2. let (l)^m ij be the minimum weight of any path from vertex i to vertex j that contains at most m edges. (L)^1 ij = (wij)当m等于n-1时，计算出最短路径。因为最短路径最多包含|V|-1个边。
3. 