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. Skip table:possibility P, random number <= P, add height.
11. Dynamic programming: optimal subproblems & subproblems override
12. Greedy algorithm:choose the best choice for now&hope it works for all.
13. Heap sort quick sort merge sort insertion sort selection sort bucket sort counting sort radix sort
14. Positon: static relative absolute fixed; 普通流 浮动 绝对定位
15. 后缀表达式：
16. 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
17. CC: for each vertex, if unmarked, depth first search a connected component.
18. No cycle: if a connected component has no cycle, then the number of its vertices == the number of edge + 1
19. Bipartite：depth search & color
20. 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）
21. 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].
22. 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。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。
23. 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。
24. 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、
25. 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！