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. 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
16. CC: for each vertex, if unmarked, depth first search a connected component.