Binary Search

```
left ← 1 , right ← n
repeat
  mid ← (L+r)/2
  if (t = A[mid]) return "True"
  else if (t < A[mid])
       right = mid-1
  else
       left ← mid+1
until left > right
return False
```

Tree

1. Full level [binary]
第 L 级上有 2^L 个 nodes

2. Complete Binary tree
0~h-1 全满, h 层尽量靠左.

3. n 个节点的 Com.. b. 的高为 O(logn)

排序

[n^2] Selection-Sort

```
for integer i ← 1 to n-1
    k ← i
    for int j ← i+1 to n
        if A[k] > A[j]
            k ← j
    swap A[i], A[k]
```

[n^2] Insertion Sort (A[1...n])

```
for int i ← 1 to n
    for int j ← i to 1 (j-1>0)
        if A[j-1] > A[j] then
            Swap A[j-1], A[j]
```

[n^2] Bubble-Sort (A[1...n])

```
for int i ← 1 to n-1
    for int j ← 2 to n
        if A[j-1] > A[j] then
            swap A[j-1], A[j]
```

Merge_Sort (A, n) [nlogn]

```
if n > 1
    p ← ⌊n/2⌋
    B[1...p] ← A[1...p]
    C[1...n-p] ← A[p+1...n]
    Merge-Sort (B, P)
    Merge-Sort (C, n-p)
    A[1...n] ← Merge (B, p, C, n-p)
```

Merge (L, n_L, R, n_R)

```
n ← n_L + n_R
Let A[1...n] be a new Array
i ← 1   j ← 1
for k ← 1 to n
    if i ≤ n_L and (j > n_R or L(i) ≤ R(j))
        A[k] ← L[i] ; i ← i+1
    else
        A[k] ← R[j] ; j ← j+1
return A
```

Quick_Sort (A[1...n], lo=1, hi=m)

```
p ← Partition (A, lo, p-1)
Quick-Sort (A, lo, p-1)
Quick-Sort (A, p+1, hi)
```

O(nlogn) 最坏 O(n²)

Partition (A, lo, hi)

```
p ← Random (lo, hi)
pivot ← A[p]
L ← lo , R ← hi
for int i ← lo to hi
    if (A[i] < pivot) A'[L++] ← A[i]
    else A'[R--] ← A[i]
A[lo, hi] ← A'
return L ;
```

Link List

```
Traverse (A):
node trav = A
while (trav != null)
    print trav.value
    trav = trav.next
```

Queue FIFO

```
enQueue (item):
    if (rear < MAXSize)
        S[rear] = item
        rear ++
    else
        Queue is Full

deQueue
    if (front < rear)
        front ++
    else
        Queue is empty
```

Ring Queue

```
enQueue (item):
    if (!isFull())
        array[rear] = item
        rear = (rear + 1)
               % (MaxSize + 1)

deQueue ( )
    if (!isEmpty())
        front = (front + 1)
                % (MaxSize + 1)

isFull ( )
    return (rear+1) % (Max+1) == front

isEmpty ( )
    return rear == front
```

Stack FILO

用法一: 判断括号是否对称
前括号入栈, 后括号将栈顶弹出匹配, 若不配则为 false

用法二: 用 stack 实现算术的中缀转后缀
之后再用 stack 将后缀求计算出结果

★★ 中缀转后缀表达式

```
InfixToPostfix (String Infix)
n ← len[s]
String Postfix ← empty String
Operator stack ops ← empty stack
for i ← 0 to n-1
    token ← Infix[i]
    if (token is operand) then
        Postfix += token
    else
        if (ops is empty) then
            ops.push(token)
        else if (token is '(')
            ops.push(token)
        else if (token is ')')
            topS ← tops.top()
            repeat
                add to topS to Postfix String
                ops.pop()
                topS ← ops.top()
            until (topS is "(")
        else
            topS ← ops.top()
            while (topS precedence over token)
                add to topS to Postfix String
                ops.pop()
                topS ← ops.top()
            ops.push(token)
while (ops is not empty)
    topS ← ops.top()
    add to topS to postfix String
    ops.pop()
return Postfix
```

String Matching

Rabin-Karp (T, P, d, q)

```
n ← len(T) , m ← len(P)
h ← d^(m-1) (mod q) , p ← 0  t0 ← 0
for j ← 0 to m-1
    p ← (dp + P[j]) mod q
    t0 ← (d t0 + T[j]) mod q
for i ← 0 to n-m
    if p != t_i then
        t_{i+1} ← (d(t_i - T[i] h) + T[i+m]) mod q
    else
        if P[0...m-1] = T[i, i+m-1]
            pattern occurs with shift i
```

d 进制 q 取模
• 构建第一个
for i ← 0 to m
    hash ← (d × hash + t[i]) % q
    target ← (... target + P[i]) ...
    if i != m, then h ← (h × d) % q

• 之后匹配时 (如果后)

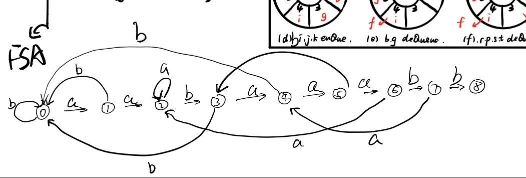
hash ← (d × (hash − h × t[i]) + t[i+m]) % q

FSA: Σ: 字集 X: P[1...] 的状态

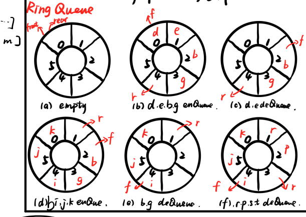
```
m ← len(p) , x ← 0
for i ← 0 to m-2
    for each c in Σ
        if p[i+1] = c
            δ[i, c] ← i+1
        else δ[i, c] ← δ[x, c]
    x ← δ[x, P[i+1]]
```

subString: P[a, b]
prefix: P[0, ...]
Suffix: P[..., m]

a a b a a a b b



Queue front 前部, rear 尾部
Ring Queue



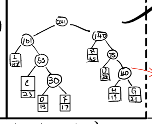
KMP (String txt, String pattern)

```
int m = txt.length(), n = pattern.length();
int [] next = nextArray(pattern);
int p = 0, i = 0;
while (i < m) {
    if (txt.charAt(i) == pattern.charAt(p)) {
        i++ ; p++ ;
    }
    else if (p != 0) {
        p = next[p-1]
    }
    else i++ ;
    if (p == n) return true ;
}
return false
```

nextArray (String pattern)

```
int next[] = next int [pattern.length()] ;
next[0] = 0
int j = 0
for int i = 1 to pattern.length
    while (j > 0 && pattern.charAt(j) != pattern.charAt(i)) {
        j = next[j-1];
    }
    if (pattern.charAt(i) == pattern.charAt(j))
        j++
    next[i] = j ;
return next[] ;
```

Huffman tree



1. 一个哈夫曼树至少有 2 个节点, 设节点 u, v have the top-2 lowest frequencies, then
(1) u, v have same parents
(2) depth(u), (v) ≥ depth(x) [x 为任意叶节点]
2. 哈夫曼树编码是最佳前缀编码, 空间成本最低

F	G	D	H	U	C	L	E
0111	1111	0110	1110	110	010	00	10

traversing Binary Tree

先 根左右

```
preorderprint (treeNode tree) {
    print (root)
    if (root.left != null)
        preorderprint (root.left)
    if (root.right != null)
        preorderprint (root.right)
```

后 左右根

```
PostorderPrint (treeNode tree)
    if (root.left != null)
        PostorderPrint (root.left)
    if (root.right != null)
        PostorderPrint (root.right)
    print (root)
```

中 左根右

```
InorderPrint (treeNode tree)
    if (root.left != null)
        InorderPrint (root.left)
    print (root)
    if (root.right != null)
        InorderPrint (root.right)
```

层次 (Level order)

```
LevelOrderPrint (treeNode tree)
queue is a new Queue
queue.enqueue (tree)
while (!queue.isEmpty())
    treeNode c = queue.top();
    print (c)
    queue.dequeue ;
    if (c != null && c.left != null)
        queue.enqueue (c.left)
    if (c != null && c.right != null)
        queue.enqueue (c.right)
```

AVL Tree

```
insert (x, t)   O(h)
if (t == null)
    return t = new TreeNode(x)
else if (x < t.value) {
    t.left = insert (x, t.left) ;
    if (high(t.left) - high(t.right) == 2) {
        if (x < t.left.value)
            t = LL(t)
        else t = LR(t) ;
    }
} else if (x > t.value) {
    t.right = insert (x, t.right);
    if (high(t.right) - high(t.left) == 2) {
        if (x < t.right.value)
            t = RL(t);
        else t = RR(t);
    }
t.height = Math.max(high(t.left), high(t.right)) + 1 ;
t.size = size(t.left) + size(t.right) + 1 ;
return t;
}
```

```
high (t)   return t == null ? -1 : t.high ;
size (t)   return t == null ? -1 : t.size ;
```

```
findMin (t) {
    if (t == null) return null;
    if (t.left == null) return t;
    return findMin(t.left)
```

```
findMax (t) {
    if (t == null) return null;
    if (t.right == null) return t;
    return findMax(t.right)
```

```
LL (X)
TreeNode w = X.left;
X.left = w.right ;
w.right = X ;
X.high = max(high(X.left), high(X.right))
w.high =
X.size =
w.size =
return w
```

```
RR (W)
Tree Node X = w.right
w.right = X.left ;
X.left = w ;
X.high =
w.high =
X.size =
w.size ;
return X ;
```

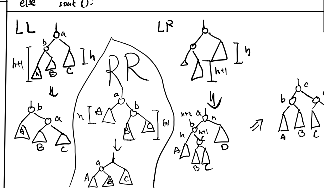
```
LR (X)
X.left = RR(X.left) ;
return LL (X) ;
```

```
RL (X)
X.right = LL(X.right) ;
return RR(X) ;
```

```
remove (X, t)   O(h)
if (t == null) return null
if (x < t.value) {
    t.value = remove (X, t.left) ;
    if (high(t.right) - high(t.left) == 2) {
        TreeNode currentNode = t.right ;
        if (high(CN.left) > high(CN.right))
            t = RL(t);
        else t = RR(t);
else if (x > t.value) {
    t.right = remove(x, t.right);
    if (high(t.left) - high(t.right) == 2) {
        TreeNode CN = t.left;
        if (high(CN.right) > high(CN.left))
            t = LR(t);
        else t = LL(t);
else if (t.right != null && t.left != null) {
    t.value = find Min(t.right).value;
    t.right = remove(t.value, t.right);
else t = (t.left != null) ? t.left : t.right;
if (t != null) {
    t.high = Math.max( high(t.left), high(t.right)) + 1;
    t.size = size(t.left) + size(t.right) + 1;
}
return t
```

```
findX (node, key)
if (node == null) return null;
else {
    TreeNode temp;
    if (node.value == key) return node;
    else if (node.value > key) {
        if (node.left == null)
            return node;
        else temp = findX(node.left, key);
    }
    else {
        if (node.right == null)
            return node;
        else temp = find X (node.right, key);
    }
    if (node.value < key && temp.value > key ||
        node.value > key && temp.value < key) {
        a = Math.abs(node.value - key);
        b = Math.abs(temp.value - key);
        if (b < a) return temp;
        else return node;
    }
    else return temp
```

```
findKthElement (node, K) {
    if (node != null)
        int s = size (node.left);
        if (s == k-1)    sout (node.value);
        else if (k < s)  find KthElement (node.left, k);
        else find KthElement (node.right, k-s-1);
    else sout ();
```



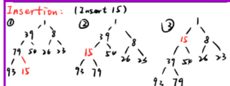
Predessor 前驱 → 比它小的最大值
Successor 后继 → 大于它的最小值

```
K 叉树和叉树找最远
max_dep = 0 ; max_node = null ;
root = Nodes [1]
Sou (tree.root, used, 0)
max_dep = 0, used = new boolean[n+1];
Sou (max_node, used, 0)
max_dep is max
```

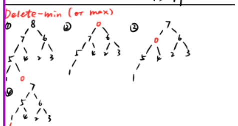
```
Sou (root, boolean[] used, deep)
    root.deep = deep;
    if (root.haspeople && deep > max_deep) {
        max_deep = deep ;
        max_node = root ;
    }
    used [root.value] = true;
    for (int i = 0 ; i < root.sons.size() ; i++) {
        if (!used [root.sons.get(i).value] {
            sou (root.sons.get(i), used, deep+1);
        }
    }
}
```

求在数组 A[n] 中 k 在 A[n] 和 A[2n] 间, 复杂度 O[logk]

Insertion: (Insert 15)



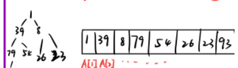
Delete-min (or max)



Find rightmost leaf

n = 9 → 1001
skip the first 1
0 → left , 1 → right

Binary Heaps in Dynamic Arrays



Property I: for A[i] → left child A[2i]
→ right child A[2i+1]
→ parent: A[i/2]

Binary Heap

```
insert (int input)
TreeNode temp = new TreeNode(input)
if (length == 1) {
    nodes[length] = temp
    length ++;
} else if (length < nodes.length) {
    int p = length ;
    nodes[length] = temp
    while (p > 1) {
        if (nodes[p/2].value < nodes[p].value)
            break;
        else {
            swap (nodes, p, p/2)
            p /= 2
        }
    }
    length ++;
}
else {
    nodes[1] = temp;
    length = 1
    change ( );
    length = nodes.length ;
```

O(log n)
树用数组时
可有以比较
插入的新数
直接加到 O(n)

```
getTop ( ) {
    return nodes[1].value;
}
```

Change ( )

```
while ((length*2 < nodes.length && nodes[length].value >=
nodes[length*2].value) || (length*2+1 < nodes.length &&
nodes[length].value >= nodes[length*2+1].value)) {
    long a = 0, b = 0, min ;
    if (length*2 < nodes.length)
        a = nodes[length*2].value;
    if (length*2+1 < nodes.length)
        b = nodes[length*2+1].value;
    if (a != 0 && b != 0)
        min = Math.min(a, b);
    else if (a != 0)
        min = a;
    else min = b
    if (min == nodes[length*2].value) {
        swap (nodes, length, length*2);
        length = length*2
    else {
        swap (nodes, length, length*2+1)
        length = length*2+1
        if (length >= nodes.length)
            break;
```

Property: A tree with n nodes has n-1 edges

Property II: Let T be a tree where every internal node has at least 2 child nodes.
If m is the number of leaf nodes, then the number of internal nodes is at most m-1.

Property III: A complete binary tree with n ≥ 2 has height O(log n)

I proof:
For each non-root node v, it has one and only one edge point to itself.
A tree with n nodes, thus the number of non-root nodes is n-1.
Thus, this tree has n-1 edges.

II proof
Let T be a tree where every internal node has at least 2 child nodes. If m is the number of leaf nodes, then the number of internal nodes is at most m-1.
Suppose internal node v has x_v child nodes
The average child nodes of each internal node is x
It has m leaf nodes, thus it has m/x parent nodes at most, i.e., they are parent of leaf nodes.
For m/x internal nodes, it has at most m/x² parents.
For m/x² internal nodes, it has at most m/x³ parents.
The total number of internal nodes is
m/x + m/x² + ... + 1
It is at most m-1.

Suppose the height is h → III proof
The number of nodes at each level:
Level 0: $2^0 = 1$, Level 1: $2^1 = 2$
Level 2: $2^2 = 4$, Level 3: $2^3 = 8$
...
Level h-1: $2^{(h-1)}$, Level h: x (x ≥ 1)
Thus, $2^0 + 2^1 + ... + 2^{h-1} + x = n$
→ $(1-2^{h-1})/(1-2) = n - x$ → $2^{h-1} < n$
Thus, h = O(log n)

Space = O(|V|+|E|)

Adjacency Matrix . Space = O(|V|²)

Shortest Path

Single Source Shortest Path (SSSP)

Solve SSSP : BFS

BFS: O(|E|+|V|) (Breadth first search)

queue Q:

Repeat the following until Q is empty

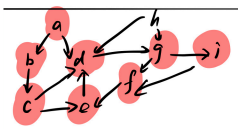
De-queue from Q the first vertex V

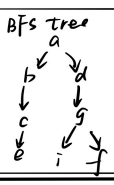
For every out-neighbor u of v that is still white

2.1 Enqueue u into Q, and color u yellow

2.2 Make u a child of v in the BFS tree T

Color V red (meaning V is visited)





O(|V|+|E|)

```
判断DAG
DFS(Node begin, ArrayList<Node>[] edge, Node[] point){
    Stack<Node> s = new Stack<>();
    s.push(begin);
    begin.color = 1
    while(!s.isEmpty){
        Node temp = s.peak();
        boolean pushed = false;
        for(Node t : edge[temp.value]){
            if(pushed && t.color == 0){
                s.push(t);
                t.color = 1;
                pushed = true;
            }
        }
        if(!pushed){
            Node t = s.pop();
            t.color = 2
        }
    }
}
```

DFS (Depth First) 判断 DAG (Directed acyclic graph)

Stack S Complexity : O(|E|+|V|)

At the beginning, color all vertices in the graph white, and create an empty DFS tree T.

Create a stack S. Pick an arbitrary vertex v. Push v into S, and color it yellow (which means "in the stack")

Make v the root of T

Repeat the following until S is empty

Let v be the vertex that currently tops the stack S (do not remove v from S)

Does v still have a white out-neighbor

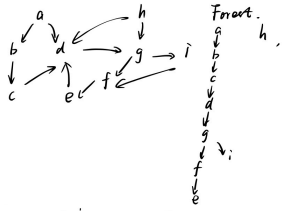
2.1 If yes: let it be u.

Push u into S, and color u yellow

Make u a child of v in the DFS-tree T

2.2 If no, pop v from S, and color v red (meaning v is visited)

If there are still white vertices, repeat the above by restarting from an arbitrary white vertex v', creating a new DFS tree rooted at v'.



Forest.


```
Topological(ArrayList<Node>[] edge, Node[] point){
    Queue S = new Queue();
    for(Node t : point){
        if(t.in == 0)
            S.enQueue(t);
    }
    while(!S.isEmpty()){
        Node temp = S.deQueue();
        temp.color = 2;
        for(Node te: edge[temp.value]){
            if(te.color == 0){
                te.in--;
                if(te.in == 0){
                    te.color = 1;
                    S.enQueue(te);
                }
            }
        }
    }
}
```

BFS

```
DiJi(int begin, Node[] point, ArrayList<ArrayList<edge>> edges){
    Queue<Node> queue = new PriorityQueue<>(new Comparator<Node>(){
        @Override
        public int compare(Node o1, Node o2){return o1.length - o2.length;}
    });
    point[begin].len = 0 ;
    queue.add(point[begin]);
    while(!queue.isEmpty()){
        Node temp = queue.poll();
        for(int i=0; i<edges.get(temp.value).size(); i++){
            int path = temp.len + edges.get(temp.value).get(i).weight;
            int len = point[edges.get(temp.value).get(i).next.value].len;
            if(len > path){
                point[edges.get(temp.value).get(i).next.value].len = path;
                point[edges.get(temp.value).get(i).next.value].parent = temp;
                queue.add(point[edges.get(temp.value).get(i).next.value]);
            }
        }
    }
}
```

O((|V|+|E|)*log|V|)

```
static class edge{ int weight; Node first;
    Node next;
    public ····· (构造函数)
}
```

```
long MST(Node begin, ArrayList<ArrayList<Edge>>edges){
    Long sum = 0;
    Queue<Node> queue = new PriorityQueue<>(new Comparator<Node>(){
        @Override
        public int compare(Node o1, Node o2){return o1.length - o2.length;}
    });
    queue.add(begin);
    while(!queue.isEmpty()){
        Node temp = queue.poll();
        temp.color = 2;
        sum += temp.best.weight;
        temp.best.next.color = 2;
        int value = temp.value;
        for(Edge t : edges.get(value)){
            if(t.next.best.weight <= t.weight){
                t.next.best = t;
                if(t.next.color == 0){
                    t.next.color = 1;
                    queue.add(t.next);
                }
                else{
                    if(queue.remove(t.next))
                        queue.add(t.next)
                }
            }
        }
    }
}
```

```
static class Edge{
    Long weight;
    Node first;
    Node next
    public ··· (构造函数)
}
```

```
static class Node{
    int value;
    long len =
    long coefficient;
    Node parent;
    Edge best =
new Edge(null, 1, null);
    int color = 0;
    int in;
    int out;

Node(int value,
long coefficient){
    :
}
```

```
SCC(ArrayList<ArrayList<Node>> edges, ··· reverseEdge, Node[] point){
    int n = point.length;
    int[] inWhichSet = new int[n+1];
    Node begin = point[1];
    ArrayList<Node> LR = new A···;
    DFS(LR, begin, reverseEdge, point);
    for(Node node: point) node.color = 0;
    Arr··· L = reverse(LR);   (LR 转序)
    SecondDFS(1, inWhichSet, L, L.get(0), edges, reverseEdge, point);
    long sum=0
    boolean[] isUsed = new boolean[point.length];
    for(int i=1 ; i<point.length; i++){
        if(point[i].in == 0 && !isUsed[point[i].value]){
            sum += point.t ;
            isUsed[point[i].value]= true;
        }
    }
    print(sum);
}
```

```
SecondDFS(int setIndex, int[] inWhichSet, Arr···<Node> L, Node begin, Arr···<Arr···<Node>> edge, ··· reE, Node[] point){
    Stack<Node> s = new Stack<>();
    Arr··· <Node> SSC = new Arr···L···<>();
    Node min = null;
    s.push(begin);
    begin.color = 1;
    while(!s.isEmpty){
        Node temp = s.peak();
        boolean pushed = false;
        for(Node t : edge[temp.value]){
            if(pushed && t.color == 0){
                s.push(t);
                t.color = 1;
                pushed = true;
            }
        }
        if(!pushed){
            Node t = s.pop();
            t.color = 2
            SSC.add(t);
            inWhichSet[t.value] = setIndex;
            if(min == null)  min = t;
            else if(min.t > t.t)
                min = t;
        }
    }
    int setIn = 0;
    for(Node node: SSC){
        for(Node t : rev···E···get(node.value)){
            if(inWhichSet[t.value] != setIndex){
                setIn t=1;
            }
        }
        point[node.value] = min;
    }
    min.in = setIn;
    boolean hasWhite = false;
    Node beginning = null;
    for(int i=1; i<L.size(); i++){
        if(L.get(i).color == 0){
            hasWhite = true;
            beginning = point[i];
            break;
        }
    }
    if(hasWhite)
        SecondDFS(setIndex+1, inWhichSet, L, beginning, edges, reverseEdge, point);
}
```

红色字强力缩点..

```
DFS(L, begin, edge, point){
    stack<Node> s = new Stack<>();
    begin.color = 1;
    while(!s.isEmpty){
        Node temp.peak();
        boolean pushed = false;
        for(Node t : edge[temp.value]){
            if(pushed && t.color == 0){
                s.push(t);
                t.color = 1;
                pushed = true;
            }
        }
        if(!pushed){
            Node t = s.pop();
            t.color = 2
        }
    }
}
```

```
boolean hasWhite = false;
Node beginning = null;
for(int i=1; i<point.length; i++){
    if(point[i].color == 0){
        hasWhite = true;
        beginning = point[i];
    }
}
if(hasWhite)  DFS(L, beginning, edge, point);
```