大O表示法：$\exists C>0, \forall n\geq C_1, f(n)\leq Cg(n)$，则 $f(n)=O(g(n))$

大Ω表示法：$\exists C_1>0, \forall n\geq C_2, f(n)\geq C_2 g(n)$，则 $f(n)=\Omega(g(n))$

大Θ表示法：有 $f(n)=O(g(n))$，且 $f(n)=\Omega(g(n))$，则 $f(n)=\Theta(g(n))$

Master Theorem: recurrance equation: $T(n)=\alpha T(\frac{n}{b})+f(n)$

$T(1)=O(1)$, $T(n)=\alpha T(\lceil \frac{n}{b} \rceil)+O(n^\gamma)$ for $n\geq 2$ $(\alpha\geq 1, \beta\geq 1, \gamma\geq 0)$

① $\log_\beta \alpha < \gamma \rightarrow T(n)=O(n^\gamma)$

② $\log_\beta \alpha = \gamma \rightarrow T(n)=O(n^\gamma \log n)$

③ $\log_\beta \alpha > \gamma \rightarrow T(n)=O(n^{\log_\beta \alpha})$

二分查找：O(logn)

```
bool BinarySearch(int A[], int t, int len){
  int l=0, r=len-1, mid=0;
  do{
    mid=l+(r-l)/2;
    if(t==A[mid]) return ture;
    else if(t<A[mid]) r=mid-1;
    else l=mid+1;
  }while(l>r);
  return false;
}
```

快速排序：O(nlogn)(平均)，O(n²)(最坏)，不稳定，易退化成冒泡

```
void Quick(int A[], int l, int r){
  if(l>=r) return;
  int i=l-1, j=r+1, x=A[(l+r)/2];
  while(1){ i<j
    do i++; while(A[i]<x);
    do j--; while(A[j]>x);
    if(i<j) swap(A[i],A[j]);
  }
  Quick(A,l,j); Quick(A,j+1,r);
}
```

选择排序O(n²)不稳定，从未排序中找最小的往前排

```
void Selection(int A[], int len){
  for(int i=0; i<len-1; i++){
    int k=i;
    for(int j=i+1; j<len; j++){
      if(A[k]>A[j]) k=j;
    }
    swap(A[i], A[k]);
  }
}
```

希尔排序：无确定时间复杂度，不稳定，易退化成插排

```
void Shell(int A[], int len){
  int N=len, h=1;
  while(h<N/2) h=h*2+1;
  while(h>=1){
    for(int i=h; i<N; i++){
      for(int j=i; j>=h; j-=h){
        if(A[j-h]>A[j]) swap(A[j-h], A[j]);
        else break;
      }
    }
    h/=2;
  }
```

冒泡排序：O(n²)稳定

```
void Bubble(int A[], int len){
  for(int i=0; i<len-1; i++){
    for(int j=1; j<len; j++){
      if(A[j-1]>A[j]) swap(A[j-1], A[j]);
    }
  }
}
```

Array：易寻找/修改，难增删；Linked List：易增删，难寻找/修改

链表的插入：Time O(n), Space O(1)；删除：Time O(n), Space O(1).

查找：Time O(n), Space O(n). 修改：Time O(n), Space O(1).

Dubble Linked List：双向链表. Circular Linked List：环状链表

判断是否为环状链表：快慢指针：if(tmp.next==tmp.next.next) return ture;

操作Time都是O(1).　慢指针next　快每次next.next.

插入排序：O(n²)，稳定，插并比较

```
void Insertion(int A[], int len){
  for(int i=0; i<len; i++){
    for(int j=i; j>=1; j--){
      if(A[j-1]>A[j]) swap(A[j-1], A[j]);
    }
  }
}
```

Stack：FILO；Queue：FIFO (linear)　a+b*c：prefix expression: +a*bc

用处：Stack：Function Runtime　infix expression: a+b*c

Queue：OS Scheduling　(用stack)　postfix expression: abc*+

环形队列：队空：front=rear，队满：(rear+1)%m=front

进队：rear=(rear+1)%m，出队：front=(front+1)%m

大小：(rear-front+m)%m.

归并排序：O(nlogn)，稳定，常数大

```
void Merge(int A[], int l, int r){
  if(l>=r) return;
  int mid=l+(r-l)/2;
  Merge(A,l,mid); Merge(A,mid+1,r);
  int k=0, i=l, j=mid+1;
  while(i<=mid && j<=r){
    if(A[i]<=A[j]) tmp[k++]=A[i++];
    else tmp[k++]=A[j++];
  }
  while(i<=mid) tmp[k++]=A[i++];
  while(j<=r) tmp[k++]=A[j++];
  for(i=l, j=0; i<=r; i++, j++) A[i]=tmp[j];
}
```

String算法：用txt去匹配pat

Brute-Force: O(mn)

```
int BruteForce(String pat, String txt){
  int m=pat.length(); int n=txt.length();
  for(int i=0; i<=n-m; i++){
    int j=0;
    for(j=0; j<m; j++) if(txt[i+j]!=pat[j]) break;
    if(j==m) return i; ←返回移位i后开始匹配
  }
  return -1; ←返回未能发现匹配
}
```

Rabin-Karp (Hash) O(mn) 但如果有 O(m+n).

Horner's Rule: $A(x)=a_nx^n+a_{n-1}x^{n-1}+\cdots+a_1x+a_0$ 在 x_0 处

的解为 $A(x_0)=((a_nx_0+a_{n-1})x_0+\cdots+a_1)x_0+a_0$

基数 模数

```
int RabinKarp(String pat, String txt, int d, int q){
  int m = pat.length(); int n = txt.length();
  int h = d^(m-1) (mod q), p=0, t=0;
  for (int j=0; j<=m-1; j++){
    p = (d*p + pat[j]) mod q;
    t = (d*t + txt[j]) mod q;
  }
  for(int i=0; i<=n-m; i++){
    if(p!=t) t = (d*(t-txt[i]*h)+txt[i+m]) mod q;
    else { if(pat[0...m-1]=txt[i...i+m-1]) return i;
           else t = (d*(t-txt[i]*h)+txt[i+m]) mod q; }
  }
}
```

如:避免哈希冲突

Finite State Automata: Q:状态集; q_0:$q_0\in Q$ 开始状态; A:$A\subseteq Q$ 接受态;

Σ:输入字符集; δ:状态转移函数 $Q\times\Sigma\to Q$.

```
int[][] Transition(String pat, char Σ[]){   // ← 预处理
  int m = pat.length;
  int dfa[0][pat[0]] = 1;
  for(int x=0, j=1; j<M; j++){
    for(int c=0; c<Σ.length(); c++){
      dfa[j][c] = dfa[x][c];
    }
    dfa[j][pat[j]] = j++;
    x = dfa[x][pat[j]];
  }
  return dfa;
}
```

FSA算法: $O(|\Sigma|*m+n)=O(C*m+n)=O(m+n)$

```
int FSA(String pat, String txt){
  int m = pat.length(); int n = txt.length();
  int q=0; int δ[][] = Trasition(pat, Σ);
  for(int i=0; i<n-1; i++){
    q = δ[q][txt[i]];
    if(q==m) return (i-m);
  }
  return -1;
}
```

Search Pattern: aabaaabb

j	pattern[1...j]	x
0		0
1	a	1
2	a b	0
3	a b a	1
4	a b a a	2
5	a b a a a	2
6	a b a a a b	3
7	a b a a a b a	0

Substring 子字符串, 不包括空串, prefix 前缀, suffix 后缀.

(P[1...]) (P[...m]).

prefix function (next array):

给定字符串 P[1...m], 其前缀函数 $\pi: \{1,2,\dots m\}\to\{0,1,\dots,m-1\}$ 是

$\pi[i]=\max\{k, k<i$ 且 $P[1...k]=P[i-k+1,\dots i]\}$ ← 公共前后缀

```
int[] NextArray(String pat){
  int m = pat.length(); int π[m+1]; π[1]=0;
  int k=0;
  for(int q=2; q<=m; q++){
    while(k>0 && pat[k+1]!=pat[q]) k=π[k];
    if(pat[k+1]==pat[q]) k=k+1;
    π[q]=k;
  }
  return π;
}
```

KMP: O(m+n)

```
int KMP(String pat, String txt){
  int m = pat.length(); int n = txt.length();
  int[] π = NextArray(pat); int q=0;
  for(int i=1; i<=n; i++){
    while(q>0 && pat[q+1]!=txt[i]) q=π[q];
    if(pat[q+1]==txt[i]) q=q+1;
    if(q==m) return (i-m);
  }
  return -1;
}
```

Tree:

① n nodes → n-1 edges: 对每个非根节点U，都有且仅有一条边指向它，而根节点无边指向，而对有n个节点的树，非根节点有(n-1)个，故有(n-1)条边.

② 对每个内节点都有两个子节点的树，若共有m个叶节点，则内节点最多有(m-1)个.
设每个内节点平均有x个子节点，共有m个叶节点 ($x \geq 2$)，则叶节点的父节点共最多$\frac{m}{x}$个.
对$\frac{m}{x}$个内节点，共最多$\frac{m}{x^2}$个父节点，以此类推有 $\frac{m}{x}+\frac{m}{x^2}+\frac{m}{x^3}+\cdots+1 = m(\sum_{i}\frac{1}{x^i}) = m\cdot\frac{1}{x}\cdot\frac{1}{1-\frac{1}{x}} = \frac{m}{x-1}$
$\because x \geq 2$ ∴该式最大值为(m-1).

③ 有$n \geq 2$个节点的完全二叉树有高度$O(\log n)$.
设树高为h，对每一层：$l_0=2^0=1$, $l_1=2^1=2$, $l_2=2^2=4$, $l_3=2^3=8$ … $l_{h-1}=2^{h-1}$, $l_h=x\ (x\geq 1)$
因此，$2^0+2^1+2^2+\cdots+2^{h-1}+x=n \Rightarrow 2^h-1=n-x \Rightarrow 2^h<n+1 \Rightarrow h<1+\log_2 n \Rightarrow h=O(\log n)$

Traversal: ① preorder: 从root开始，先递归遍历左子树，再递归遍历右子树 (root→left→right)
(①③或②③决定一棵树，只有①②不能求定) ② postorder: 从左子树leaf开始，先递归遍历左子树，再递归遍历右子树，最后到root (left→right→root)
③ inorder: 从左子树leaf开始，先递归遍历左子树，再到root，最后递归遍历右子树 (left→root→right)
④ level: 自顶向下，自左向右 (top→bottom, left→right)

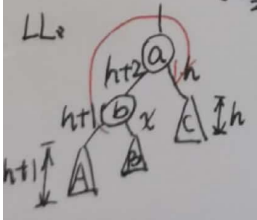
Huffman Tree.
(假设至少有2个节点的Huffman tree中节点U，V有最小的频率，则 ① U，V有共同的父节点 ② $\min[\text{depth}(U),\text{depth}(V)] \geq \text{depth}(x)$, x为Huffman tree的任意叶.
Huffman编码是最优的前缀码，空间复杂度最低.
设字符表C，$\forall c \in C$，有频度$f(c)$，且$|C|=n$，设T为C的最优编码树，x，y为C中频率最低的字符，字符表$C'=C\cup\{z\}-\{x,y\}$，其中$f(z)=f(x)+f(y)$，且$d(x)=d(y)=d(z)+1$，C'的编码树为T'.
建树代价 $B(T')=B(T)-f(x)d(x)-f(y)d(y)+f(z)d(z)=B(T)-[f(x)+f(y)]$
设C'中构建的Huffman tree为$H_{C'}$，令H_C为$H_{C'}$中z由x和y合并形成的新Huffman tree，则有：$H_{C'}=H_C-f(x)+f(y)$
由于H_C最优，则$B(H_C)\leq B(T')$，于是$B(H_C)=B(H_{C'})+[f(x)+f(y)] \leq B(T')+[f(x)+f(y)]=B(T)$
又由于T最优，则$B(T)\leq B(H_C)$，故$B(H_C)=B(T)$，即Huffman编码是最优的.

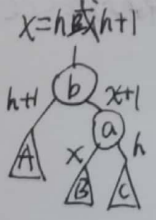
Priority Queue: 是完全二叉树，Space O(n)，Time. 插入$O(\log n)$，删除(最大/最小)$O(\log n)$

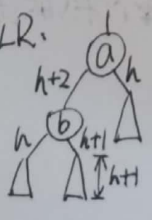
Binary Heap: 是完全二叉树，若U是内点，则其小于/大于其子结点.
找right-most leaf: $O(\log n)$: 设binary heap有n个node，将n转化为二进制，并删去most significant bit，对剩下的bits，从左到右数，从根节点开始，若为0往左走；若为1，往右走
parent(i) $=\lfloor \frac{i}{2} \rfloor$, left(i) $=2i$, right(i) $=2i+1$. rightmost leaf(i) $=A[n]$.
堆排序: $O(n\log n)$ = build heap $O(n\log n)$ + n × delete-min $O(\log n)$
root-fix operation: $O(\log n)$: 从rightmost leaf开始下沉操作. 这样建堆$(O(n))$比插入建堆$(O(n\log n))$快
设Binary heap高为h，则$n=2^{h+1}-1$，故 $\sum_{i=0}^{h} O(i\cdot 2^{h-i}) = O(n)$ $(n=2^{h+1}-1)$

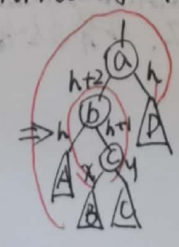
BST: Space O(n). Time: Successor O(h), insert O(h), delete O(h). BST不一定是完全二叉树，但右子树>本结点>左子树
($n=|S|$, h为树高) Predecessor
Predecessor Query: 先向左子树找一个结点，再一直往右子树找到尽头，即为前驱，Successor Query类似.

BBST: 左右子树高度差不超过1. Space. O(n), predecessor $O(\log n)$. insert $O(\log n)$, delete $O(\log n)$.
有n个node的BBST高为$O(\log n)$.
设树高为h，最小树高为填满各层：$n=2^{h+1}-1 \Rightarrow h=\log_2(n+1)-1$；树高一定时，左右子树高低差1时，node最少.
此时有：$n(h)=n(h-1)+n(h-2)+1$, $n(0)=1$, $n(1)=2$ ($n(k)$即为k高树共有的node数).
$\Rightarrow [n(h)+1]=[n(h-1)+1]+[n(h-2)+1]$，斐波那契数列 $\Rightarrow n(h)+1=f(h+3) \Rightarrow n(h)=\frac{1}{\sqrt{5}}\left[\left(\frac{1+\sqrt{5}}{2}\right)^{h+3}-\left(\frac{1-\sqrt{5}}{2}\right)^{h+3}\right]-1$
$\Rightarrow h<\log_{\frac{1+\sqrt{5}}{2}}[\sqrt{5}(n(h)+2)]-3$ 综上：n个node的BST高为$O(\log n)$. $>\frac{1}{\sqrt{5}}\left(\frac{1+\sqrt{5}}{2}\right)^{h+3}-2$

LL:


⇒


LR:


⇒


⇒


Graph: $G=(V,E)$

无向图：$\sum_{u\in V} d(u)=2|E|$

有向图：$\sum_{u\in V} d^+(u)=\sum_{u\in V} d^-(u)=|E|$

Adjacency List: Space: $O(|V|+|E|)$, Adjacency Matrix: Space $O(|V|^2)$ ← 无向图是对称的，有向图不一定

Single Source Shortest Path (SSSP): 从源点 s 到终点 t 的最短路径，或 s 不能到 t.

Breadth First Search (BFS): 层次遍历的推广.

最初图中全部节点都是白色，并建构一个空的 BFS Tree T。建构一个空的队列 Q，并将源点 s 入队，上色为黄色，使其为 T 的 root. 然后重复以下步骤直至 Q 为空：① 从 Q 中离队 v ② 对 v 的所有白色出节点 u（① 入队 Q，上黄色 ② 作为 T 中 v 的子节点）③ 将 v 上色为红色.

复杂度：对每个出队的节点 v，需 $O(1+d^+(v))$ 处理，则 BFS 总用时为 $O(\sum_{v\in V}(1+d^+(v)))=O(|V|+|E|)$

由 BFS 构建出的 BFS Tree 中就是单源最短路径了.

Depth First Search (DFS): 各种序遍历的推广.

最初图全部节点都是白色，并建构一个空的 DFS Tree T。建构一个空的栈 S. 随机选择一个节点 v 入栈，上色为黄色，使其为 T 的 root. 然后重复以下步骤直至 S 为空：① 查看 S 的栈顶 v，并看其是否有白色出节点（有，使其为 u，上黄色，使 u 在 T 中为 v 的子结点，入栈. 若无，弹出 v，并上红色.）

复杂度：$O(|V|+|E|)$

Edge Classification: 对 G 中的有向边 (u,v) 可分为
- Forward edge: 在 DFS-森林中 u 是 v 的祖先.
- Backward edge: 在 DFS-森林中 u 是 v 的后代.
- Cross edge: 不是以上两种.

利用一个计数器 c，每当入栈和出栈时，c++. 对每个节点 v 有：

① discovery time: d-tm(v) 是 v 被入栈时的 c 值. ② finish time f-tm(v): 是 v 被出栈时的 c 值

定义 $I(v)=[d\text{-}tm(v), f\text{-}tm(v)]$. 该值能在 DFS 运行时多用 $O(|V|)$ 时间来获得.

Parenthesis Theorem: ① 若 u 是 v 的真祖先，则 $I(v)\subseteq I(u)$ ② 若 u 是 v 的真后代，则 $I(u)\subseteq I(v)$ ③ 否则，$I(u)$ 和 $I(v)$ 不相交.

Cycle Theorem: G 中有环当且仅当 T 中有 Backward edge.

Cycle Detection 用时: $O(|V|+|E|)$

Topological Sort: 用于 DAG，返回该图中所有节点的线性序列，在前面的节点是后面的祖先. 总用时 $O(|V|+|E|)$ ← 有不只一种

建构一个空列表 L，对 G 进行 DFS，每当节点 v 变红（即出栈）时，将其加入 L 中. 最后，逆序输出列表 L 中元素.

Shortest Path with Weighted Graph

若 $v_1\to v_2\to\cdots\to v_{k+1}$ 是从 v_1 到 v_{k+1} 的最短路径，则对 $\forall i,j$ 且 $1\le i\le j\le k+1$，$v_i\to v_{i+1}\to\cdots\to v_j$ 是 v_i 到 v_j 的最短路径.

假设该命题为假，即 v_i 到 v_j 有更短的路径，令 $i=1, j=k+1$，则有从 v_1 到 v_{k+1} 的更短的路径，有悖条件.

Dijkstra's Algorithm (需非负权值)

$\forall v\in V$, parent(v)=null, dist(v)=∞. 取源点 s, dist(s)=0. 令 S=V. 重复以下步骤直至 S 为空.

① 从 S 中弹出 dist(u) 最小的节点 u. ② 对 u 的所有出边 (u,v)，若 $dist(v)>dist(u)+w(u,v)$，则 $dist(v)=dist(u)+w(u,v)$ 且 parent(v)=u.

最终会获得 SP tree 即为单源最短路径树. 复杂度：$O((|V|+|E|)*\log|V|)$

Minimum spanning Tree: 对无向图

Prim's Algorithm:

设 (u,v) 是 E 中权值最小的边，令 $S=\{u,v\}$，初始化 T_{mst} 只有一条边 (u,v).

$\forall z\in V/S$：① 若 z 是 u 的邻边，但不是 v 的，best-ext(z)=edge(z,u) ② 若 z 是 v 的邻边，但不是 u 的，best-ext(z)=edge(z,v). ③ 若 z 是 u 和 v 的邻边，best-ext(z)= edge(z,u) 和 edge(z,v) 中权值最小的.

重复以下步骤直至 S=V：① $u\in S$，获得有最小权值的外延边 (u,v). ② 将 v 加入 S 中，将 (u,v) 加入 T_{mst} 中. ③ 对 v 的所有邻边 (v,z). 若 $z\notin S$，且 best-ext(z) 权重大于 edge(v,z)，则 best-ext(z)=edge(v,z)

复杂度：若用最小堆实现的 Priority Queue，为 $O((|V|+|E|)*\log|V|)$；若用斐波那契堆，为 $O(|V|\log|V|+|E|)$.

Strongly Connected Components: 对有向图.

强连通分支即一个最大的节点集 $S\subseteq V$，且 $\forall u,v\in S$，有 $u\to v$ 和 $v\to u$，即 u 和 v 互相可达. 可由 DFS 运行两次获得.

设 S_1 和 S_2 都是 G 的 SCC，则 $S_1\cap S_2=\emptyset$.

假设 $\exists v\in S_1$ 且 $\in S_2$，则对 $\forall u_1\in S_1$ 和 $\forall u_2\in S_2$，有一条路径 $u_1\xrightarrow{S_1} v\xrightarrow{S_2} u_2$，同理有 $u_2\to v\to u_1$，有悖条件.

寻找 SCC：① 将原图 G 的所有路的方向都调转成 G^R. ② 对 G^R 做 DFS，得到拓扑序 L^R，并逆序 L^R 得 L.

③ 从 L 的第一个节点开始 DFS，若该 DFS 结束后仍未遍历完所有节点，则按顺序在 L 中下一个白色节点开始做 DFS，直至全部节点都变红. ④ 最终每个 DFS 的结果就是 SCC.

复杂度：$O(|V|+|E|)$