1. 深度为 h 的平衡二叉 (AVL) 树的最少结点数为 N_h(N_h 个结点的 AVL 树最大深度为 h)，则 N_h 和 h 满足：$N_0 = 0$，$N_1 = 1$，$N_2 = 2$；$N_h = N_{h-1} + N_{h-2} + 1$。

2. 顺序表设计算法对每个元素写出位移量 (或目标索引) 并观察规律。和微积分一样，不要纠结起点，应关注中间过程和一般情形。

3. 折半查找要注意：

```
while(low <= high)
low = mid+1;
high = mid - 1;
```

high 后插入。

4. 插入链表有尾插法和头插法两种。尾插法 (表尾插入，q 为表尾)：

```
q->next = p; q = q->next;
```

头插法 (表头插入，q 为表头)：

```
tmp = q->next; q->next = p; p->next = tmp;
```

5.& 会改变地址中的内容和地址本身，如

```
void change(Type &adress) {
    adress = xxx;
}
```

6. 出栈序列一个元素右侧元素中值比它小 (优先顺序在它前面) 的为尚未出栈的元素，这些元素之间一定满足与优先顺序相反的顺序排序，如 a<b<c(a 先入栈)，则 cab 不可能，因 a<c，b<c，ab 顺序与优先顺序相同，不符合要求。

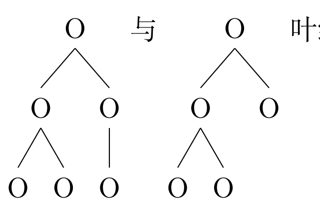
7. 循环队列队尾元素在队头元素之前时，因队尾指针指向队尾下一个元素位置，所以队头指针队尾指针指向同一位置，此时队空 (同理也可使队头指针指向前一个增加头结点，队尾指针指队尾)。

8. 输入受限的双端队列：$\leftarrow$ 1 2 3 4 $\leftrightarrow$ 若第一个输出 4，即 (4 x x x 形式)，则此时 1，2，3 一定还在队里，因此下一个输出的只能是 1 或 3。因此，输入受限队列夹心的中心元素不能先输出，如 123 中 2 为夹心元素，因此 4213 中比 4 小的元素中有 2 夹心先出，不满足要求。输出受限输出序列不能有山峰，如 4132 中比 4 小的元素中 3 为山峰，不满足要求。

9. 只有度为 0 和 2 的结点的二叉树最大深度为 $\frac{1+n}{2}$，最少结点个数为 2h+1。

10. 满二叉树最底层结点数比其余结点数之和多 1，第 i 层有 2^{i-1} 个结点。

11. 因 与 叶结点数相同，完全二叉树最多结点数应在右图基础上加 1。



12.n 个结点的 m 叉树最小高度为 $\lceil log_m(n(m-1)+1) \rceil$。

13. 有 n 个结点的完全二叉树高度为 $\lfloor log_2 n \rfloor + 1$。

14. 先序遍历根和左儿子 (或唯一儿子) 相邻，右儿子为第一个非左子树结点。

15. 如果一个名词能确定数据具体存储方式，则其为存储 (物理) 结构。

16. 先序线索树前驱，后序线索树后继不能有效求解。

17. 前序序列为入栈次序，中序序列为出栈次序。

18. 后序非递归遍历二叉树：

```
typedef struct{ BinaryTree data; int tag; } Stack;
void PostOrderPrint(BinaryTree t) {
        Stack s[MAX_SIZE]; int top = 0;
        BinaryTree p = t;
        while( p || top > 0) {
                if(p) {
                        s[++top].data = p;
                        s[top].tag = 0;
                        p = p->left;
                }
                else {
                        p = s[top].data;
                        if(p->right && s[top].tag!=1) {
                                s[top].tag = 1;
                                p = p->right;
                        }
                        else {
                                cout<<s[top--].data->data;
                                p = NULL; }
                }
        }
}
```

19. 使用栈或队列的算法设计：a. 考虑好工作栈，工作队列和工作指针的取值范围 (空间，所有可能值)，再从取值范围中划分出不同情况用作条件判断。如 18 中 p 应遍历所有结点的左右子树 (包括叶)，因此应考虑 p 指空 (空子树) 的情况。s 应存放回溯时要访问的根结点且所有结点都应入栈一次，只访问从 s 中弹出的结点，因此入栈元素应为 p 第一次指向的结点，出栈条件应为左右子树都遍历过 p 指空时。b. 考虑变量范围后再考虑变量取值顺序。18 中出栈顺序为后序序列，入栈顺序应为先序序列。

20. 简易栈：初始化：

```
ElemType stack[MAX_SIZE]; int top = 0;
```

入栈：

```
stack[++top]=element;
```

出栈：

```
element = stack[top--];
```

栈空：

```
top == 0
```

栈顶:

```
element = stack[top];
```

简易队列: 初始化:

```
ElemType queue[MAX_SIZE]; int front = 0, rear = 0;
```

入队:

```
queue[rear++] = element;
```

出队:

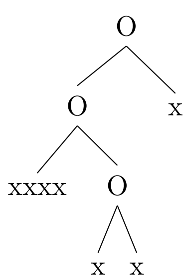
```
element = queue[front++];
```

队空:

```
front == rear
```

队头/尾:

```
element1 = queue[front];
element2 = queue[rear];
```

21. 二叉排序树插入时只插结点处。
22. 平衡因子绝对值大于等于 1 的结点互换高度，一正一负中间插，中间子树根转移。
23. 平衡二叉树取最少结点数即非叶结点平衡因子均为 1 的情况。
24. 由哈夫曼树的构造过程可得哈夫曼树有 n 个叶结点，n-1 个非叶结点，无度为 1 的结点。
25. 前缀码所有编码结点均为叶结点，没有一个编码是另一个编码的前缀。
26. 叶结点权值组可能对应多个哈夫曼树 (带权路径相同但形状不同)。
27. 

其中 xxxx 为省略的子树，x 为查找失败处。查找失败要进行三次比较，因此查找失败的平均查找长度为 $\frac{\cdots+3\times 2}{\cdots+2}$，其中 2 为两个失败结点 (x)。
28. 堆是一个完全二叉树。
29. 中序非递归遍历二叉树:

```
void InOrderPrint(BinaryTree t) {
        BinaryTree stack[MAX_SIZE]; int top = 0;
        BinaryTree p = t;
        while(p || top>0) {
                if(p) {
```

```
                stack[++top] = p;
                p = p->left;
            }
            else {
                p = stack[top--];
                cout<<p->data;
                p = p->right;
            }
        }
}
```

30. 树的根到叶结点的路径长度 = 深度-1

31. 合并长度为 m 和 n 的有序表最多需 m+n-1 次比较 (大小大小...)

32. 图的遍历就是从一个结点出发遍访其余结点。错误! 非连通图无法实现。

33. 非连通图可以有单个度为 0 的结点，不占用任何一条边。

34. 边数最少的连通图和边数最少的强连通图都是简单回路。

35.n 个顶点无向图，要有 $\frac{(n-1)(n-2)}{2}$(完全图) + 1 个边以确保其为连通图 (完全图加一边)。

36. 极大连通子图 = 连通分量，极小连通子图 = 最小生成树 (MST)

37. 当某个顶点只有出弧没有入弧时，其他顶点无法到达这个顶点，不可能与其他顶点和边构成强连通分量 (这个顶点为一个单独强连通分量)。

38. 顶点 (事件)v_k 的最早发生时间 $ve(k)$ 为从源点到 v_k 最长路径 (取最大值)。

顶点 (事件)v_k 的最迟发生时间 $vl(k)$ 为从源点到汇点最长路径长度减去从 v_k 到汇点最长路径长度 (取最小值)。

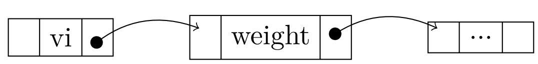
边 (弧，活动)a_i 的最早开始时间 $e(i)$ 为从源点到 a_i 的尾顶点 (活动起点) 的最长路径。

边 (弧，活动)a_i 的最迟开始时间 $l(i)$ 为从源点到汇点的最长路径长度减去从 a_i 的头顶点 (活动终点) 路径长度再减去 a_i 权，即 $l(i) = vl(k) - weight(a_i)$，其中 $a_i = v_j\vec{v}_k$。

若 $ve(k) = vl(k)$，则 v_k 为关键路径上的点。

39.Prim 算法：新边连旧边，无环。Kruskal 算法：最小边，无环。

40.Dijkstra 算法：a. 选择离出发点最近的点，确定这个点的最短路径。b. 修改其它点的最短路径 (广度优先)。

41. 带权有向图邻接表：

42.AOE 网一定从一个事件 (顶点) 出发止于一个事件 (汇点)。

43.

算法	图的类型	算法结果
Prim,Kruskal	带权无向连通图	最小生成树
Dijkstra,Floyd	带权有向图	最短路径

44. 折半查找比较次数例：1 2 3 4 5 (找 4)，$\frac{1+5}{2} = 3$，$\frac{4+5}{2} = 4$，因此共比较两次。

45.B+ 树：非失败，非根结点关键字个数 n 满足 $\lceil \frac{m}{2} \rceil \leq n \leq m$；根结点关键字个数满足 $1 \leq n \leq m$

B 树：非失败，非根结点关键字个数 n 满足 $\lceil \frac{m}{2} \rceil - 1 \leq n \leq m-1$，子树个数 $\lceil \frac{m}{2} \rceil \leq n_h \leq m$；根结点关键字个数满足 $1 \leq n \leq m-1$，子树个数 $2 \leq n_h \leq m$

m 阶关键字 (非叶结点) 个数为 n 的 B 树，高度 h 满足 $log_m(n+1) \le h \le log_{\lceil \frac{m}{2} \rceil}(\frac{n+1}{2})+1$，失败结点个数为 n+1。

m 阶高为 h 的 B 树关键字最多为 m^h-1

B 树失败结点为空，不占高度，插入访问失败结点，插在叶结点上。

B 树插入溢出时中间结点上游父结点并增加分支。

B 树删除时缺的位置可由同一棵子树根上的关键字向下滑动填上，若下滑后父结点关键字过少，则借兄弟或父结点反方向下潜。

46. 表项：散列表中的记录个数。

47. 查找失败时最后还要和空元素比较一次。要用散列函数值域来计算查找失败平均查找长度，成功用元素个数算。(成个失域)

48.KMP 算法 Next 数组求法：

a. next[1]=0,next[2]=1

b. 从当前计算的元素之前截字符串，根据左侧 next 值判断从几位开始比较，若成功 +1，失败-1，直到成功或减到 1 为止。

c. 前 n 位与倒数后 n 位比较。

d. next 数组输入子串匹配失败的位置，输出要滑动到的位置。如 a b c a c 在 j=5(c) 匹配失败，由 next[5]=2，应从 b 开始匹配。

e. 统一 next[] sstring[] 的下标，都从 0 开始时 next[0]=-1

49. 对任意的 n 个关键字的序列进行基于比较的排序，至少要进行 $\lceil log_2(n!) \rceil$ 次两两比较。

50. 除了直插，冒泡，归并，基数外都不稳定。(直基冒归)

51. 第 i 趟快排，冒泡，选择后，会有 $\ge i$ 个元素在最终位置。

52. 涉及稳定性时，哪个数据项要求全局有序后排哪个 (保证有序)。

53. 堆的删除要将堆尾元素置顶，一次下游最多比较 2 次。

54.k 路归并，排序趟数 m 满足 $k^m \ge N$，$m=\lceil log_k N \rceil$

55.m 叉哈夫曼树，度 0 结点数为 n_0，度为 m 结点数为 n_m，则有：$n_m=\frac{n_0-1}{m-1}$ 多余结点数 $u=(n_0-1)\%(m-1)$，应增加空归并段数为 $m-u-1$

56. 并行 m 路归并要设置 2m 个输入缓冲区，2 个输出缓冲区。

57.Hash 函数 H(key)=key%p，N 为表长，n 为关键字个数，$\alpha=\frac{n}{N}$ 为填装因子，p 应取不大于表长的最大素数，$N=\lceil \frac{n}{\alpha} \rceil$ 链地址法 (即链表)$\neq$ 线性探测再散列法。

58.Floyd 算法 $A^{(-1)}[i][j]=arcs[i][j]$，$A^{(k)}[i][j]=Min\{A^{(k-1)}[i][j],A^{(k-1)}[i][k]+A^{(k-1)}[k][j]\}$，$k=0,1,\cdots,n-1$。$A^{(k)}_{n\times n}=Min\{A^{(k-1)}_{n\times n},\begin{bmatrix}a_{1k}\\ \cdots\\ a_{nk}\end{bmatrix}\oplus\begin{bmatrix}a_{k1} & \cdots & a_{kn}\end{bmatrix}\}$，其中 $\oplus$ 表示将矩阵乘法中元素间的运算由乘改为加。Min{} 表示将矩阵对应元素取最小值作为新矩阵值。

59. 邻接表：第 i 结点的第 j 邻接点:

```
g.vertices[g.vertices[i]->first ->...(->next (j-1 nexts))->adjvex]
```

对应网的权 Aij:

```
g.vertices[i]->first(->next(j nexts))->info
```

60. 平均查找长度 $\neq$ 最大查找长度，看清“平均”“最坏情况”。

61. 有向图 v_i 到 v_j 所有路径：令 visited[j]=true，从 v_i 对图进行深度优先遍历，若某次 NextAdj 操作得到 v_j，则逆序输出 (从 stack[0] 开始) 栈和 v_j。

62. 邻接表深度优先非递归遍历：

```
void DFS(ALGraph &g,int vi){
        //从vi顶点开始进行深度优先遍历
        ArcNode *stack[g.arcnum];
        int top=0;
        bool visited[g.vexnum];
        for(int i=0; i < g.vexnum; i++)
                visited[i]=false;
        ArcNode *p=g.vertices[vi]->firstarc;
        cout<<vi<<"  ";
        visited[vi]=true;
        while(p||top>0) {
                if(p) {
                        if(!visited[p->adjvex]) {
                                cout<<p->adjvex<<"  ";
                                visited[p->adjvex]=true;
                                stack[++top]=p;
                                p=g.vertices[p->adjvex]->firstarc;
                        }
                        else
                                p = p->nextarc;
                        }
                        else
                                p = stack[top--]->nextarc;
                }
                cout<<endl;
}
```

63. 基于交换排序思想：low 指向左侧第一个要交换的，high 指向右侧第一个要交换的，当 low>=high 时结束，若使用 A[0] 保存 A[low] 第一次交换改为

```
A[low]=A[high];
```

，与之对应的交换改为 A[high]=A[low]，最后恢复 A[low] 可减少空间使用。

64. 顶点 k 的偏心度为所有其余点到 k 最短路径中的最大值，最小偏心度的顶点为中心点。

65.Warshall 算法判断 i 到 j 是否有一条路径：$A_{ij}^{(k+1)} = A_{ij}^{(k)} \cup (A_{ik}^{(k)} \cap A_{kj}^{(k)})$

66. 基数排序 = 队列 + 散列

67. 每个判断条件下都应改变状态以避免死循环 (输出不算改变状态，赋值算)。

68. 算法中的循环往往对应一个数学归纳过程。
69. 题目中非形式的数据结构要用伪码形式化的定义出来。
70. 最后不要忘了写算法复杂度。