1. **数据结构**：数据结构是相互之间存在的一种或多种特定关系的数据元素的集合。数据结构包括三方面的内容:**逻辑结构**、**存储结构**和**数据的运算**。
2. **逻辑结构**：逻辑结构是指数据元素之间的逻辑关系，即从逻辑关系描述数据。它与数据的存储无关，是独立于计算机的。数据的逻辑结构分为线性结构和非线性结构，线性表是典型的线性结构；集合，树和图是典型的非线性结构。
3. **存储结构**：存储结构是指数据结构在**计算机内中的表示**（又称映像），也称**物理结构**。它包括**数据元素的表示**和**关系**的表示。主要有：**顺序存储**，**链接存储**，**索引存储**和**散列存储**。
4. **索引存储**：在存储元素信息的同时，还建立附加的索引表。
5. **算法**：算法是对**特定问题求解步骤的一般描述**，他是指令的有限序列，其中每一条指令表示一个或多个操作，具有五大特性：有穷性，确定性，可行性，输入，输出。
6. **时间复杂度**：算法的时间复杂度是一个函数，它**定性**描述了该算法的运行时间。常用大 O 表示
7. **空间复杂度**：是对一个算法在运行过程中临时占用存储空间大小的量度，记做 S(n)=O(f(n))
8. **线性表**：线性表是具有相同数据类型的 n（n>=0）个数据元素的有限序列。其中 n 为表长，当 n=0 时该线性表是一个空表。若用 L 命名线性表，则其一般表示如下：

   L=（a1，a2，…，ai，ai+1，…，an）

   其中，a1 是唯一一个“第一个”数据元素，又称为表头元素；an 是唯一的“最后一个”数据元素，又称为表尾元素。除第一个元素外，每个元素有且仅有一个直接前驱。除最后一个元素外，每个元素有且仅有一个直接后继。
9. **顺序表**：线性表的顺序存储。

10. **栈**：只允许在一端进行插入或删除的线性表。栈顶为线性表允许插入和删除的那一端，栈底是固定的，不允许进行插入和删除的另一端。
11. **队列**：是一种操作受限的线性表，只允许在表的一端进行插入，而在表的另一端进行删除。向队列中插入元素称为入队或进队；删除元素称为出队或离队。其操作的特性是先进先出。
12. **数组**：数组是由 n 个相同类型的数据元素构成的有限序列，每个数据元素称为一个数组元素，每个元素受 n 个线性关系的约束，每个元素在 n 个线性关系中的序号称为该元素的下标，并称该数组为 n 维数组。
13. **树的定义**：树是 N（N>=0）个结点的有限集合，N=0 时，称为空树，这是一种特殊情况。在任意一棵非空树中应满足：

    1）有且仅有一个特定的称为根的结点。

    2）当 N>1 时，其余结点可分为 m（m>0）个互不相交的有限集合 T1，T2，…，Tm，其中每一个集合本身又是一棵树，并且称为根节点的子树。

    具有以下两个特点：

    1）树的根节点没有前驱结点，除根节点之外的所有结点有且只有一个前驱结点。

    2）树中所有结点可以有零个或多个后继结点。
14. **二叉树**：二叉树是一种树形结构，其特点是每个结点至多只有两颗子树（即二叉树中不存在度大于 2 的点），并且，二叉树的子树有左右之分，其次序不能任意颠倒。

    二叉树以递归的形式定义：二叉树是 n（n>=0）个结点的有限集合：

    1）或者为空二叉树，即 n=0。

    2）或者由一个根节点和两个互不相交的被称为根的左子树和右子树。左子树和右子树又分别是一棵二叉树。

15. 满二叉树：一棵高度为 h，并且含有2^h-1 个结点的二叉树称为满二叉树，即树中的每一层都含有最多的结点。
16. 完全二叉树：设一个高度为 h,有 n 个结点的二叉树，当且仅当其每一个结点都与高度为 h 的满二叉树中编号为 1~n 的结点一一对应时，称为完全二叉树。
17. 对于 n 个结点的二叉树，在二叉链存储结构中有 n+1 个空链域，利用这些空链域存放在某种遍历测序下该结点的前驱结点和后继结点的指针，这些指针称为线索，加上线索的二叉树称为**线索二叉树**。
18. **并查集**：并查集是一种简单的集合表示，它支持以下 3 种操作：
    1）Union（S，Root1，Root2）：把集合 S 中的子集合 Root2 并入子集合 Root1 中，要求 Root1 和 Root2 互不相交，否则不执行合并。
    2）Find（S，x）：查找集合 S 中单元素 x 所在的子集合，并返回该子集合的名字。
    3）Initial（S）：将集合 S 中每一个元素都初始化只有一个单元素的子集合。
19. **BST（二叉排序树）**：也称为二叉查找树。二叉排序树或者是一棵空树，或者是一棵具有下列特性的非空二叉树：
    1）若左子树非空，则左子树上所有结点关键字值均小于根结点的关键字值。
    2）若右子树非空，则右子树上所有结点关键字值均大于根结点的关键字值。
    3）左、右子树本身也分别是一棵二叉排序树。
20. **AVL 树（平衡二叉查找树）**：平衡二叉树是一棵空树，或者是具有下列性质的二叉树：它的左子树和右子树都是平衡二叉树，且左子树和右子树的高度差的绝对值不超过 1
21. **Huffman 树**（哈夫曼树）：在含有 N 个带权叶子结点的二叉树中，其中带权路径长度（WPL）最小的二叉树称为哈夫曼树。
22. **带权路径长度**（WPL）：树中所有叶结点的带权路径长度之和称为该树的带权路径长度。

23. **图** G 由顶点集 V 和边集 E 组成，记为 G=（V,E），其中 V（G）表示图 G 中顶点的有限非空集；E（G）表示图 G 中顶点之间的关系（边）集合。

24. **强连通图：**在有向图中，若从顶点 v 到顶点 w 和从顶点 w 到顶点 v 之间都有路径，则称这两个顶点是强连通的。若图中任何一对顶点都是强连通的，则称此图为强连通图。有向图中的极大强连通子图称为有向图的强连通分量。

25. **生成树：**一个图的生成树是指一极小连通子图，他含有图中全部顶点，n-1 条边

26. **简单图**：一个图 G 如果满足：1.不存在重复边；2.不存在顶点到自身的边，则称图 G 为简单图。

27. **多重图**：若图 G 中某两个结点之间的边数多于一条，又允许顶点通过同一条边和自己关联，则 G 为多重图。

28. **完全图**：在无向图中，如果任意两个顶点之间都存在边，则称该图为无向完全图。含有 n 个顶点的无向完全图有 n（n-1）/2 条边。在有向图中，如果任意两个顶点之间都存在**方向相反**的两条弧，则称该图为有向完全图。含有 n 个顶点的有向完全图有 n（n-1）条有向边。

29. **图的邻接矩阵存储**：用一个**一维数组**存储图中顶点的信息，用一个**二维数组**存储图中边的信息（即各顶点的邻接关系）。

30. **图的邻接表法**：邻接表就是**对图 G 中的每个顶点 Vi 建立一个单链表**，第 i 个单链表中的结点表示依附于顶点 Vi 的边（对于有向图则是以顶点 Vi 为尾的弧），这个单链表就**称为顶点 Vi 的边表。边表的头指针和顶点的数据信息采用顺序存储。**

31. **有向无环图**：**一个有向图中不存在环**，则称为有向无环图，简称 **DAG** 图。

32. **AOV 网**：如果用 DAG 图 表示一个工程，其**顶点表示活动**，用**有向边<Vi,Vj>表示活动 Vi 必须先于活动 Vj 进行的这样一种关系**，则将这种有向图称为顶点表示活动的网络。

33. **拓扑排序**：拓扑排序是对有向无环图的顶点的一种排序，它使得如果存在一条从顶点 A 到顶点 B 的路径，那么在排序中顶点 B 出现在顶点 A 的后面。每个 DAG 图都有一个或多个拓扑排序序列。

    步骤

    1）从 DAG 图中选择一个没有前驱的顶点并输出

    2）从图中删除该顶点和所有以它为顶点的有向边

    重复 1 和 2 直到当前的 DAG 图为空或当前图中不存在无前驱的顶点为止。而后一种情况则说明有向图中必定有环。

34. **AOE 网**：在带权有向图中，**以顶点表示事件**，**有向边表示活动**，边上的权值表示完成该活动的开销（如完成活动所需时间），则称这种有向图为用边表示活动的网络，简称为 AOE 网。

    性质：

    1）只有在某顶点所代表的事件发生后，从该顶点出发的各有向边所代表的活动才能开始；

    2）只有在进入某一顶点的各有向边所代表的活动都已经结束时，该顶点所代表的事件才能发生。

    3）在 AOE 网中仅有一个入度为 0 的顶点，称为源点。网中也仅存在一个出度为 0 的顶点，称为入点。

35. **关键路径**：AOE 网中，从源点到汇点的所有路径中，**具有最大路径长度的路径称为关键路径。把关键路径上的活动称为关键活动**。

36. **B 树**：又称为多路平衡查找树，B 树中所有结点的孩子结点数的最大值称为 B 树的阶，通常用 m 表示。一棵 m 阶 B 树或为空树，或为满足如下特性的 m 叉树：

1）树中每个结点至多有 m 棵子树（即至多含有 m-1 个关键字）

2）若根结点不是终端结点，则至少有两棵子树

3）除根结点外的所有非叶结点至少有 m/2 棵子树（即至少含有 m/2-1 个关键字）

4）结点表示：

$$C_0\ k_1\ C_1\ k_2\ \ldots\ldots\ k_p\ C_p$$

37. **B+树**：是 B-树的一种变形

1）树中每个非叶结点最多有 m 棵子树

2）根结点（非叶结点）至少有 2 棵子树

3）除根结点外，每个非叶结点至少有 m/2 棵子树；

有 n 棵子树的非叶结点有 n（南大-1）个关键码

4）所有叶结点都处于同一层次上，包含了全部关键码及指向相应数据对象存放地址，关键码按关键码从小到大顺序链接

5）每个叶结点中子树棵树 n 可以＞m，也可以＜m。假设叶结点可容纳的最大关键码数为 m1，则指向对象的指针数也有 m1，这时子树棵数 n 应满足（ m1/2 ，m1 ）

6）根结点本身又是叶结点，则结点格式同叶结点

38. **散列函数**：一个把查找表中的关键字映射成该关键字对应的地址的函数，**记作 Hash (key) =Addr**

39. **散列表**：**是根据关键字而直接进行访问的数据结构**。散列表建立了关键字和存储地址之间的一种**直接映射**关系

40. 广度优先搜索（BFS）：类似于二叉树的层次遍历算法，它的基本思想是：首先访问起始顶点 v，接着由 v 出发，依次访问 v 的各个未访问过的邻接顶点 w1，w2，…，wi，然后再依次访问 w1，…，wi 的所有未被访问过的邻接顶点；再从这些访问过的顶点出发，再访问它们所有未被访问过的邻接顶点…依次类推，直到图中所有顶点都被访问过为止。

    空间复杂度 O(n)，邻接表存储时时间复杂度 O（n+e），邻接矩阵存储时时间复杂度 O（n^2）

41. 深度优先搜索（DFS）：首先访问图中某一起始顶点 v，然后由 v 出发，访问与 v 邻接且未被访问的任一顶点 w1，再访问与 w1 邻接且未被访问的任一顶点 w2，…重复上述过程。当不能再继续向下访问时，依次退回到最近被访问的顶点，若它还有邻接顶点未被访问过，则从该点开始继续上述搜索过程，直到图中所有顶点均被访问过为止。

    空间复杂度 O（n），邻接矩阵时间复杂度 O（n^2），邻接表时间复杂度 O(n+e)

42. Prim 算法：时间复杂度 O（n^2），不依赖于 e，因此它适用于求解边稠密的图的最小生成树。

43. Kruskal 算法：时间复杂度 O（e$\log e$），因此 Kruskal 算法适合与边稀疏而顶点较多的图。

44. Dijkstras 算法求单源最短路径：时间复杂度 O（n^2），如果要找出所有结点对之间的最短距离，则需要对每个结点运行一次 Dijkstra 算法，时间复杂度为 O（n^3）

45. Floyd 算法：时间复杂度 O（n^3）

46. 堆：