哈基米工业大学（深圳）

2025 年春树橘桔枸鱼蒜发试题　答案

题目来源：HITSZ 23 级对答案群　题解：Chi. Ya.

回忆（有效编辑字数序）：动量子，蜡笔小新、哆啦 A 梦，RegGniy，Ma-nx

本试题只能反映考察知识点，因为回忆不出来了所以部分试题与原题不同。

一、选择题（每题 2 分）

1. 要压缩储存一个规模为 $n*n$ 的对称矩阵，至少要分配线性表长度为 _____.

   A. $n*n/2$

   B. $n*(n+1)/2$

   C. $n*(n-1)/2$

   D. n

   对于一个 $n*n$ 的对称矩阵 A，其元素满足 $A_{ij}=A_{ji}$。因此只需要存储矩阵的下三角部分以及主对角线上的元素即可。存储这些元素所需的数量为：

   $$\sum_{i=1}^{n} i = 1+2+3+\cdots+n = \frac{n(n+1)}{2}$$

   故答案为：B

2. 哪种排序的平均复杂度是最优的 _____.

   A. 气泡排序

   B. 快速排序

   C. 插入排序

   D. 选择排序

   以上常见排序算法的平均时间复杂度：$O(n^2)$，$O(n\log n)$，$O(n^2)$，$O(n^2)$：

   故答案为：B

3. 关于动态规划的说法，不正确的是 _____.

   A. 动态规划算法适用于具有优化子结构的问题

   B. 动态规划算法能解决的问题不能用递归解决

   C. 动态规划算法适用于具有重叠子问题的问题

   D. 动态规划的核心思想是记录并重用子问题的解，以避免重复计算

   动态规划适用的一个关键性质就是如果一个问题的最优解包含了其子问题的最优解，则称此问题具有优化子结构。重叠子问题是应用动态规划的另一个关键特征。这意味着在递归求解过程中，许多相同的子问题会被反复计算。

   动态规划本质上是带有记忆的递归。任何可以用动态规划解决的问题，都可以用一个朴素的递归函数来解决。

   故答案为：B

4. 用贪心法安排任务，每个任务的（开始时间，结束时间）分别是 {(1, 3), (2, 5), (4, 6), (3, 8), (7, 9), (9, 10)}，一个人同时只能执行一个任务，执行任务时不能半途而废。这个人最多能执行完 _____ 个任务.

A. 2

B. 3

C. 4

D. 5

活动选择问题使用贪心算法求解。最优策略是：

将所有任务按照结束时间从早到晚排序。选择第一个任务（结束时间最早的）。从余下的任务中，选择与上一个已选任务不冲突（即开始时间晚于或等于上一个任务的结束时间）的第一个任务。重复直到没有更多可选择的任务。

故答案为：C

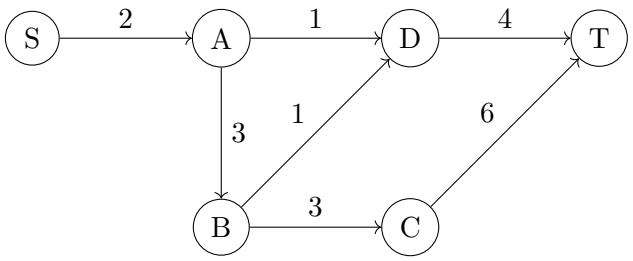
5. 图中，使用 A* 算法寻找从起点 S 到终点 T 的路径。图中边上的数字代表移动的实际代价。对于 B 点，其到终点 T 的最优估计代价 h(B) 应该是 _____.

A. 9

B. 5

C. 3

D. 1



A* 算法的评估函数是 $f(n) = g(n) + h(n)$。为了保证算法找到最优解，启发函数 $h(n)$ 必须是可采纳的，即 $h(n)$ 的值不能超过从节点 n 到目标 T 的实际最短路径成本。一个最优的启发函数就是 $h(n)$ 直接等于 n 到 T 的实际最短路径成本。

要求 B 点到 T 的最优估计代价 h(B)，即计算从节点 B 到终点 T 的实际最短路径长度。

故答案为：B

6. $g(n) = n + 3 \cdot g(n/4)$, $f(n) = n^2$, 那么 _____.

A. $g(n) = \Theta(f(n))$

B. $g(n) = \Omega(f(n))$

C. $g(n) = \omega(f(n))$

D. $g(n) = O(f(n))$

可以使用主定理来求解递归式 $g(n) = 3g(n/4) + n$ 的时间复杂度。主定理的形式为 $T(n) = aT(n/b) + f(n)$。在此题中，$a = 3$, $b = 4$, $f(n) = n^2$。比较 $f(n)$ 与 $n^{\log_b a}$。

$$n^{\log_b a} = n^{\log_4 3} \approx n^{0.792}$$

这意味着 $g(n) = O(f(n))$。故答案为：D

7. 下列程序段的时间复杂度是 _____。

```
int sum=0;
for(int i=1; i<n; i*=2)
    for(int j=0; j<i; j++)
        sum++;
```

A. $O(\log n)$

B. $O(n)$

C. $O(n \log n)$

D. $O(n^2)$

变量 i 的初始值为 1，每次循环都乘以 2。直到 $2^k \geq n$ 时循环终止。外层循环执行的次数大约为 $\log_2 n$ 次。对于外层循环的每一次迭代，内层循环执行 i 次。

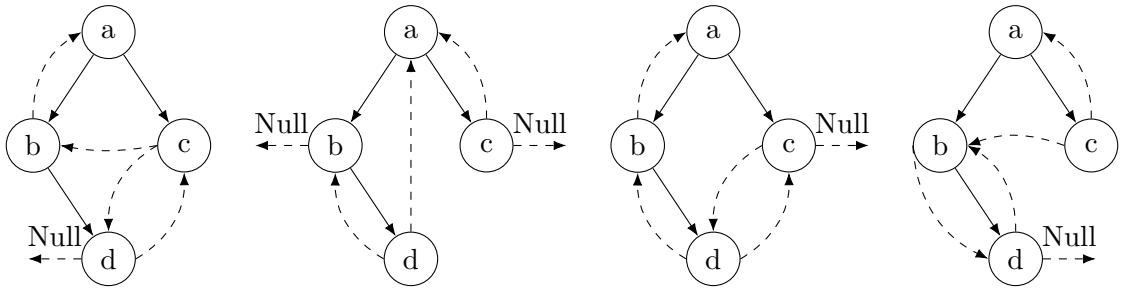
假设 n 恰好是 2 的幂，即 $n = 2^k$，则 i 的取值为 $1, 2, 4, \ldots, 2^{k-1}$。总操作次数为：

$$S = \sum_{p=0}^{k-1} 2^p = 1 + 2 + 4 + \cdots + 2^{k-1} = \frac{1(2^k - 1)}{2 - 1} = 2^k - 1$$

因为 $n = 2^k$，所以总操作次数为 $n - 1$。

故答案为：B

8. 下列线索二叉树中 (用虚线表示线索), 符合后序线索树定义的是 _____。



在后序线索二叉树中，对任意结点：若左子空，则其左子指针应指向它在后序遍历的前驱。若右子空，则其右子指针应指向它在后序遍历的后继。

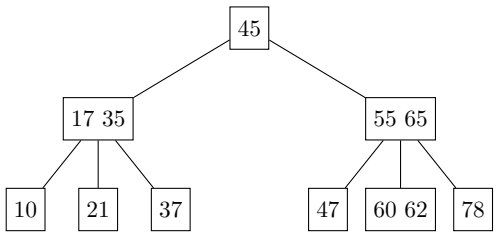
确定该二叉树的后序遍历序列。根据左右根，序列为：d, b, c, a。

- **d**: 左子空，右子空。左线索应为 NULL。右线索应指向 b。

- **b**: 左子空，右子 d。左线索应指向 d。无右线索。
- **c**: 左子空，右子空。左线索应指向 b。右线索应指向 a。
- **a**: 左右子非空，无左右线索。

故答案为：D

9. 已知一棵 3 阶 B 树, 如下图所示。删除关键字 78 得到一棵新 B 树, 其最右叶结点中的关键字是 _____。



A. 60

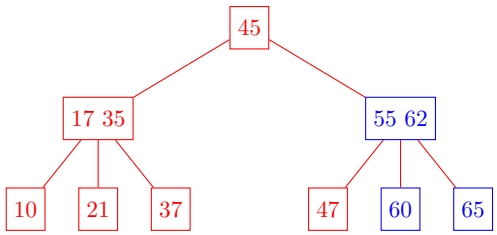
B. 60, 62

C. 62, 65

D. 65

B 树的阶数 $m = 3$。根据定义，每个结点最多有 $m - 1 = 2$ 个关键字。除根结点外，每个非叶结点至少有 $\lceil m/2 \rceil - 1 = \lceil 1.5 \rceil - 1 = 1$ 个关键字。

删除关键字 78 的步骤如下：关键字 78 位于最右侧的叶结点中。删除。删除后，该叶结点变为空，其关键字数量小于要求的最小值 1。检查其兄弟结点。它的左兄弟有 2 个关键字，是最大数量，因此可以借一个关键字。通过值旋转完成。



故答案为：D

10. 关于程序优化和正确性的说法 _____.

A. 对于部分输入具有正确的输出结果，算法就具有正确性

B. 调试程序就能保证算法的正确性

C. 对于每一个输入，都有正确的输出结果，算法才具有正确性

D. 算法只要程序能运行就正确

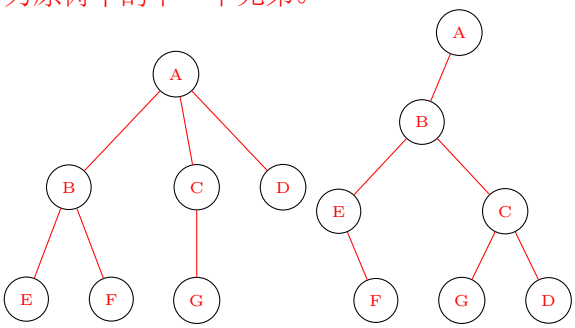
算法正确性的标准定义：算法必须对问题域内的每一个合法输入，都能在有限时间内终止并产出符合问题要求的正确输出。

故答案为：C

二、填空题（每空 1 分）

1. 多叉树的后序遍历序列与其通过孩子兄弟表示法转换而来的二叉树的 _____ 遍历序列相同。

   根据孩子兄弟表示法的转换规则，对任一结点，其左子树为原树中的第一个孩子，其右子树为原树中的下一个兄弟。



   遍历: E, F, B, G, C, D, A

   故答案为: 中序

2. 二维数组 `A` 按行优先方式存储，每个元素占用 1 个存储单元。若元素 `A[0][0]` 的存储地址是 100，`A[3][3]` 的存储地址是 220，则元素 `A[5][5]` 的存储地址是 _____。

   设数组的列数为 `c`。按行优先存储的地址计算公式为 $\text{Addr}(\texttt{A[i][j]}) = \text{Base} + (\texttt{i} \times \texttt{c} + \texttt{j}) \times \text{size}$。

   根据题意，$\text{Base} = 100$，$\text{size} = 1$。$\text{Addr}(\texttt{A[3][3]}) = 100 + (3 \times \texttt{c} + 3) \times 1 = 220$。

   所以 $\texttt{c} = 39$。

   因此，$\text{Addr}(\texttt{A[5][5]}) = 100 + (5 \times 39 + 5) \times 1 = 100 + 195 + 5 = 300$。

   故答案为: 300

3. 一个包含 7 个顶点的边不重的无向图，至少需要 _____ 条边，才能确保在点间任意分配这些边后，该图始终是连通的。

   要保证 n 个顶点的图无论如何连接都是连通的，其边数必须多于任何可能的不连通图的最大边数。对于 n 个顶点，边数最多的不连通图是一个包含 $n-1$ 个顶点的完全图（K_{n-1}）和 1 个孤点。

   对于 $n = 7$，这个不连通图是 K_6 和一个孤立点。K_6 的边数为 $C_6^2 = \frac{6 \times 5}{2} = 15$ 条。

   当图有 15 条边时，它可能是 K_6 和一个孤立点，这是不连通的。为保证连通，只需再增加 1 条边，即 $15 + 1 = 16$ 条边。

   故答案为: 16

4. 包含 50 个结点的二叉树，其高度至少为 _____。（注：根结点在第 0 层，高度定义为最大层数）

   高度为 h 的二叉树最多有 $2^{h+1} - 1$ 个结点。为了使 50 个结点的二叉树高度最小，应使其尽可能成为一棵完全二叉树。需要找到最小的 h 使得 $2^{h+1} - 1 \geq 50$。

   $2^{h+1} \geq 51$。即 $h \geq 5$。

   故答案为: 5

5. 假设在一张地图上进行寻路，各点的高度值如下表所示。若采用爬山法，从起点 (1, 2) 开始，最终抵达山顶的路径为 _____。

x\y	1	2	3	4	5
1	4	5	3	2	1
2	5	6	7	6	5
3	7	8	9	8	7
4	6	7	8	7	6
5	4	6	6	5	4

采用爬山法，从起点开始不断选择上下左右的最高点前进。

故答案为：$\boxed{(1,2) \to (2,2) \to (3,2) \to (3,3)}$

6. 在某通信系统中，六个字符出现的频数分别为 {10, 8, 6, 5, 4, 3}。若对这些字符进行哈夫曼编码，则平均码长为 _____。

将频数作为权值构建哈夫曼树。每次选取权值最小的两个结点合并。

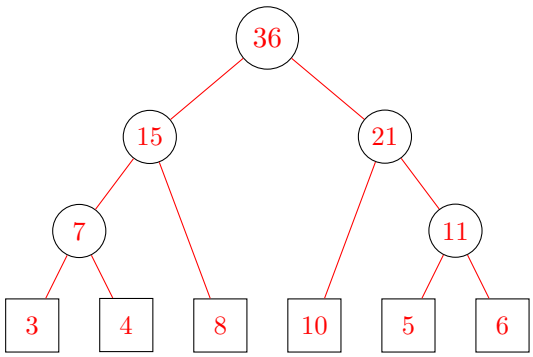
合并 3, 4 得 7 → {5, 6, 7, 8, 10}

合并 5, 6 得 11 → {7, 8, 10, 11}

合并 7, 8 得 15 → {10, 11, 15}

合并 10, 11 得 21 → {15, 21}

合并 15, 21 得 36



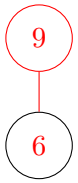
频数 10、8 的码长为 2，频数 6、5、4、3 的码长为 3。

计算平均码长：$\frac{\sum(\text{频数}\times\text{码长})}{\text{总频数}} = \frac{10\times2+8\times2+6\times3+5\times3+4\times3+3\times3}{10+8+6+5+4+3} = \frac{20+16+18+15+12+9}{36} = \frac{90}{36} = 2.5$。

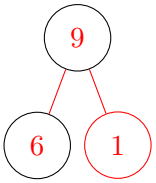
故答案为：$\boxed{2.5}$

7. 将关键字序列 {6, 9, 1, 5, 8, 4, 7} 依次插入一个初始为空的大根堆中，构建完成后，该大根堆的广度优先遍历序列为 _____。

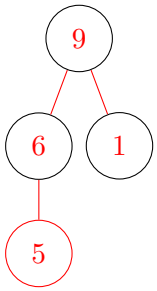
插入 9



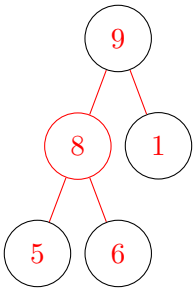
插入 1



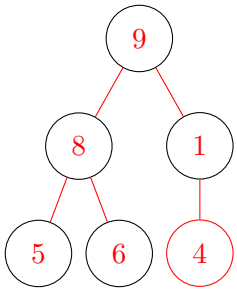
插入 5



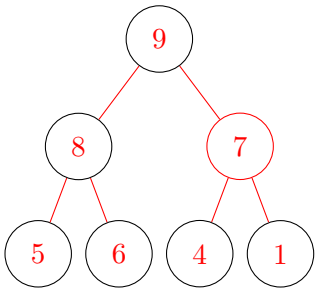
插入 8



插入 4



插入 7 (最终)



故答案为：$\boxed{9,8,7,5,6,4,1}$

8. 在一个栈上定义了两种操作，一种是将一个元素入栈，另外一种是连续出栈 k 个元素。在栈上执行一系列操作，入栈操作的代价为 1，k-出栈操作的代价为 $\min(k,s)$，其中 k 是一个正常量，s 是执行出栈操作时栈中的元素个数。那么，在最坏情况下，摊还代价复杂度是 _____。

出栈元素的个数至多与入栈操作的次数相等，由这两种操作组成的操作的时间代价最多为 n，分摊到每一个操作的时间代价就为 $n/n=1$。也就是 $O(1)$

故答案为：$\boxed{O(1)}$

9. 已知广义表 $A=(a,b,(c,d),(e,(f,g)))$，则该广义表的长度为 _____，深度为 _____。

广义表的长度是指其最外层包含的元素个数。A 的最外层元素是 a, b, (c,d) 和 $(e,(f,g))$，共 4 个。广义表的深度是其括号嵌套的最大层数。

故答案为：$\boxed{4;\ 3}$

三、简答题（10 分）

1. 设一棵二叉树的先序遍历序列为 M，F，C，A，G，E，J，H，L，T，Q，R，W，V，中序遍历序列为 G，A，C，E，F，H，J，L，M，Q，R，T，V，W。

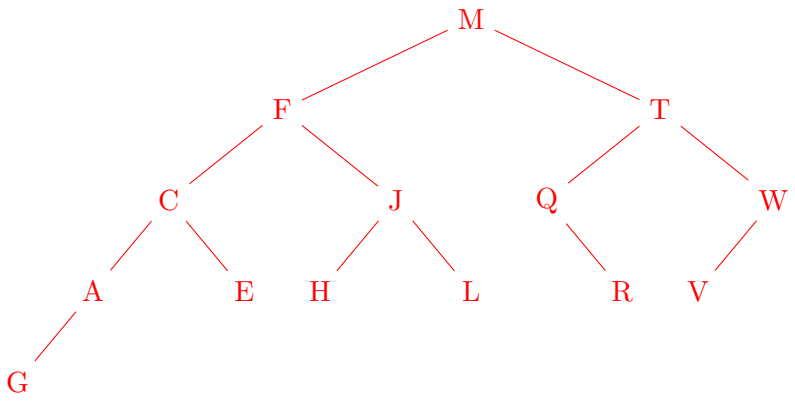
（1）求该二叉树的高度（只有一个结点的二叉树高度为 0）。

（2）写出该二叉树的后序遍历序列。

（3）请画出该二叉树的顺序存储表示。

根据先序和中序序列构造二叉树。

- 先序序列的第一个元素 M 是根节点。
- 在中序序列中找到 M，其左侧 G，A，C，E，F，H，J，L 是左子树，右侧 Q，R，T，V，W 是右子树。
- 对左子树，其在先序序列中对应的部分是 F，C，A，G，E，J，H，L，所以左子树的根是 F。
- 对右子树，其在先序序列中对应的部分是 T，Q，R，W，V，所以右子树的根是 T。
- 以此类推，递归地构建左右子树，最终还原的二叉树如下：



从图中可以看出，树的最长路径包含 5 个节点。

二叉树的高度为 4。 对还原的二叉树进行后序遍历（左右根）得到：

G，A，E，C，H，L，J，F，R，Q，V，W，T，M。

二叉树的顺序存储结构需要空位来维持完全二叉树的编号规则。这里只展示到最后一个非空节点所需的位置。

索引	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
数据	M	F	T	C	J	Q	W	A	E	H	L	-	R	V	-	G

2. 给定一个关键字序列 {47, 7, 29, 11, 16, 92, 22, 10}，要将其存入一个哈希表中。该哈希表的长度为 12，哈希函数为 $H(key) = key \mod 12$，采用线性探测再散列解决冲突。

（1）求出装载因子。

（2）画出该哈希表的终态。

装载因子 α 的计算公式为：$\alpha = \frac{\text{表中元素个数}}{\text{哈希表长度}}$。即$\alpha = \frac{8}{12} = \frac{2}{3} \approx 0.67$

关键字插入过程如下：

- $H(47) = 47 \pmod{12} = 11$. 存入地址 11。
- $H(7) = 7 \pmod{12} = 7$. 存入地址 7。
- $H(29) = 29 \pmod{12} = 5$. 存入地址 5。
- $H(11) = 11 \pmod{12} = 11$. **冲突**。线性探测下一个地址 $(11+1) \pmod{12} = 0$. 存入地址 0。
- $H(16) = 16 \pmod{12} = 4$. 存入地址 4。
- $H(92) = 92 \pmod{12} = 8$. 存入地址 8。
- $H(22) = 22 \pmod{12} = 10$. 存入地址 10。
- $H(10) = 10 \pmod{12} = 10$. **冲突**。线性探测三个地址，存入地址 1。

地址	关键字
0	11
1	10
2	
3	
4	16
5	29
6	
7	7
8	92
9	
10	22
11	47

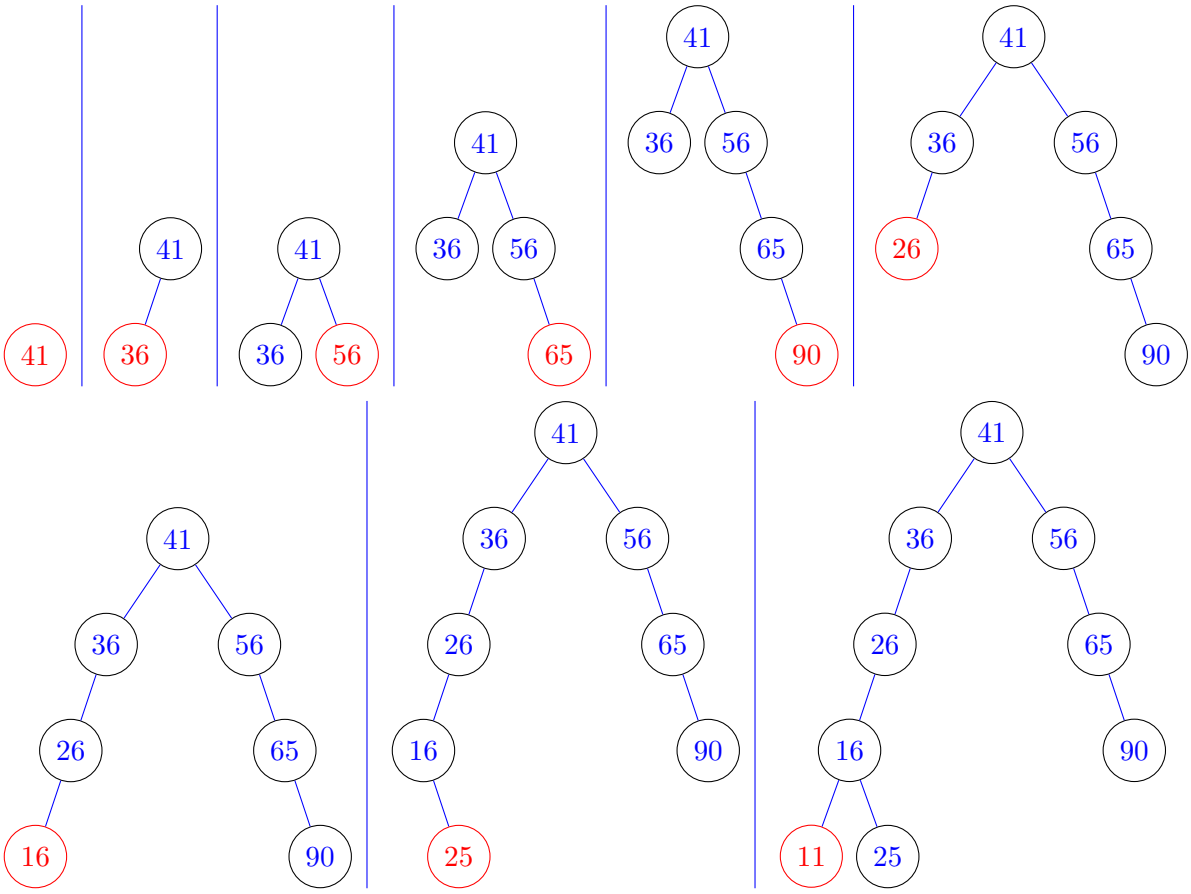
四、分析题（30 分）

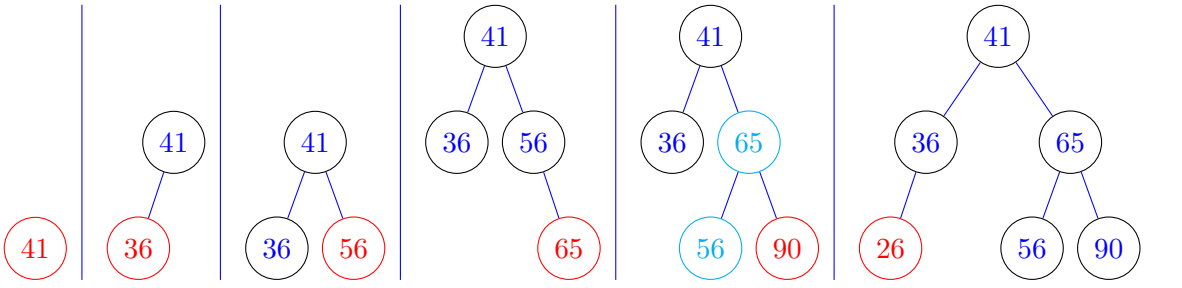
1. 给定如下序列 {41, 36, 56, 65, 90, 26, 16, 25, 11}

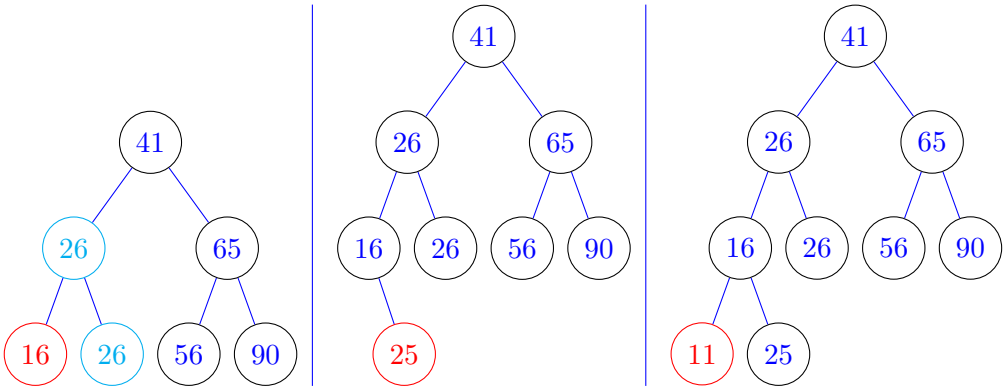
（1）将序列中的元素依次插入初始为空的二叉搜索树，画出每一步对应的树的状态。

（2）将序列中的元素依次插入初始为空的二叉平衡树，画出每一步对应的树的状态。

（3）求两棵树的平均查找长度。

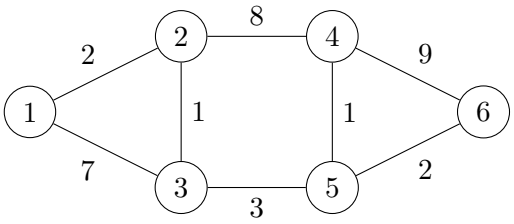






对于二分探索树，平均查找长为 $(1\times1+2\times2+3\times2+4\times2+5\times2)/9=3.22$。对于平衡二分木，平均查找长为 $(1\times1+2\times2+3\times4+4\times2)/9=2.78$。

2. 根据给出的无向图回答下列问题。



（1）用三元顺序组表示该图的邻接矩阵。

（2）从结点 1 开始进行广度优先遍历，写出遍历序列。若广度优先遍历的实现是先出队列后入队列，则遍历过程中队列最长的时候有多长?

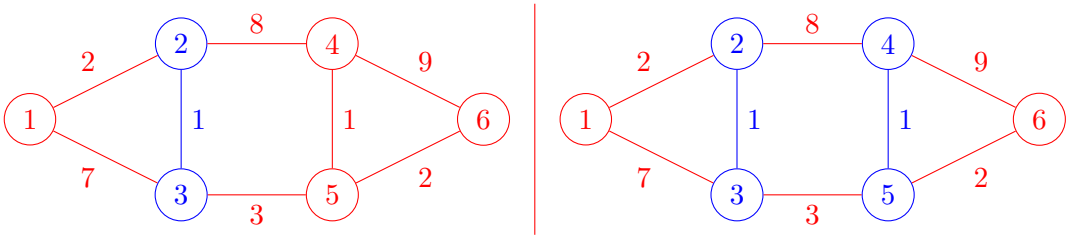
（3）求最小生成树，并说明算法思想。

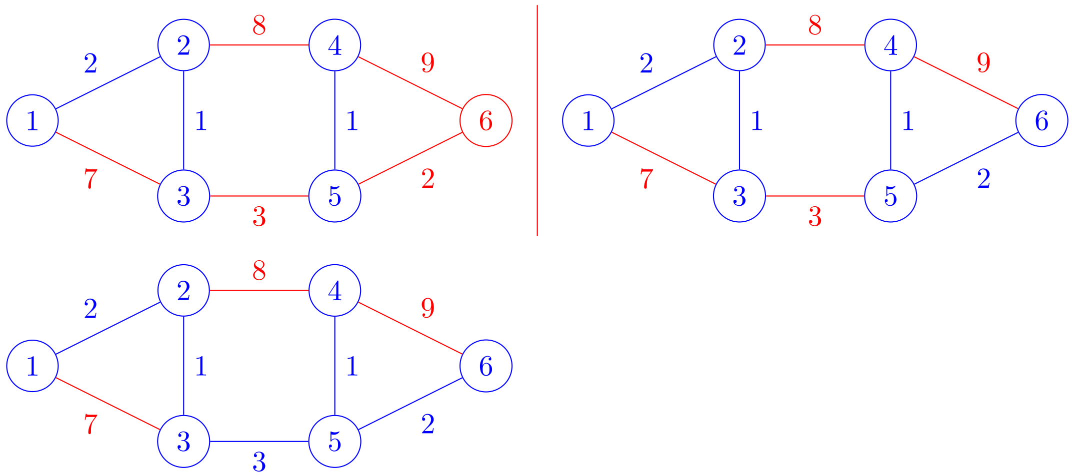
无向图一共有 8 条边，每条边使用 (From, To, Cost) 来表示即可满足题目所求。

(1, 2, 2), (1, 3, 7), (2, 3, 1), (2, 4, 8), (3, 5, 3), (4, 5, 1), (4, 6, 9), (5, 6, 2)

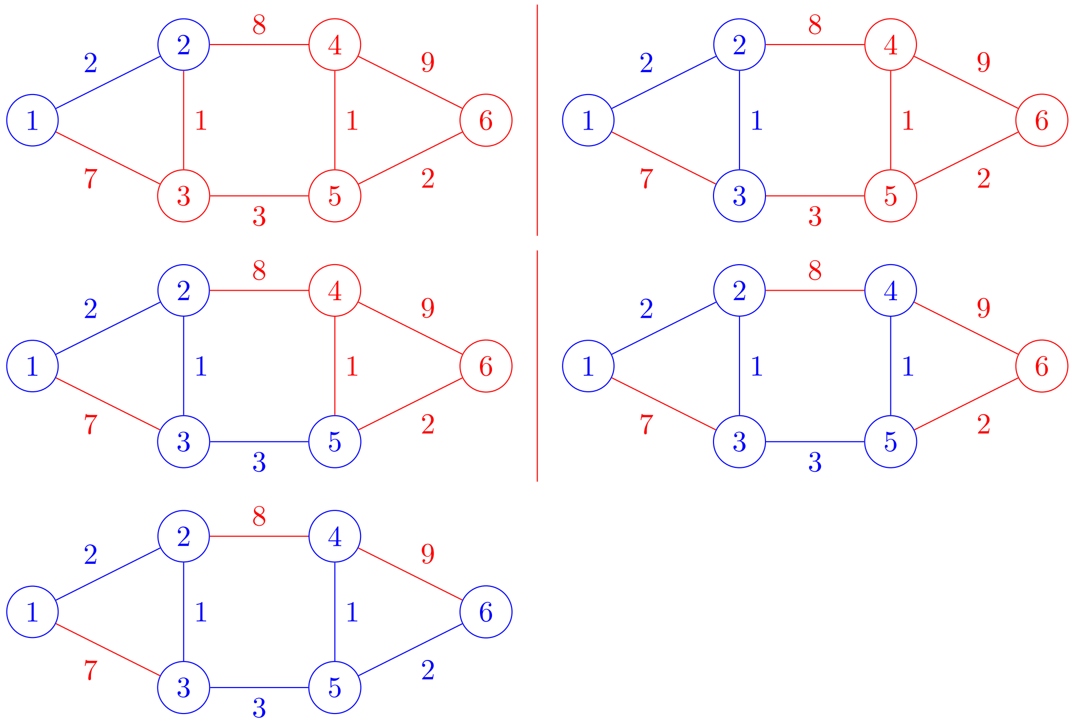
可能的广度优先遍历序列如下：1, 2, 3, 4, 5, 6 或 1, 3, 2, 5, 4, 6。遍历过程中队列最长的时候只有两个元素。这个结论不解释。

使用 Kruskal 法求最小生成树。思想是贪心，引入不会成环的最小权边。流程如下：

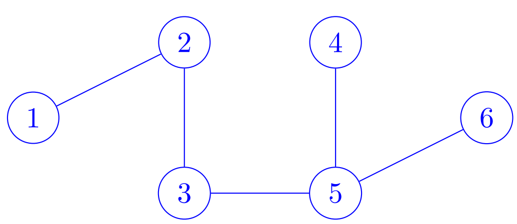




使用 Prim 法求最小生成树。思想是贪心，引入到连通分支最小权点。流程如下（从 1 开始）：



反正最后是这个：



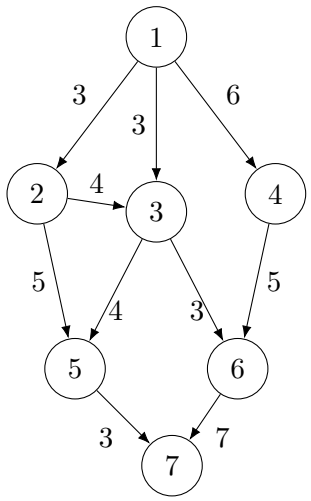
，最小权为 9。

3. 给定下图所示的有向带权图，其中顶点表示事件，边表示活动，权重表示活动的持续时间。

（1）写出该图的四个不同的拓扑排序序列。

（2）计算从源点 1 到其他各个顶点的最长路径长度。给出所有顶点的事件最早发生时间 v_e。

（3）找出该图的关键路径，并指出其长度。



拓扑排序思路：

计算所有顶点的入度。将所有入度为 0 的顶点放入一个队列中。当队列不为空时，从队列中取出一个顶点，将其添加到拓扑序列的末尾。对于该顶点的每一个邻接点，将其入度减 1。如果邻接点的入度变为 0，则将其入队。重复此过程，直到队列为空。

由于在排序过程中，队列中可能同时存在多个入度为 0 的顶点，会产生不同的拓扑序列。以下是所有可能的拓扑排序序列：

$1 \to 2 \to 4 \to 3 \to 5 \to 6 \to 7$

$1 \to 4 \to 2 \to 3 \to 6 \to 5 \to 7$

$1 \to 2 \to 3 \to 4 \to 5 \to 6 \to 7$

$1 \to 2 \to 3 \to 5 \to 4 \to 6 \to 7$

$1 \to 2 \to 3 \to 4 \to 6 \to 5 \to 7$

$1 \to 2 \to 4 \to 3 \to 6 \to 5 \to 7$

$1 \to 4 \to 2 \to 3 \to 5 \to 6 \to 7$

事件的最早发生时间 $v_e(k)$ 指的是从源点到点 k 的最长路径的长度。定义源点的 $v_e(1) = 0$。对于其他任意顶点 k，其计算公式为：

$$v_e(k) = \max\{v_e(j) + w(j,k)\}$$

其中，j 是所有指向 k 的直接前驱顶点，$w(j,k)$ 是边 (j,k) 的权重。

计算过程（按给出的第一拓扑顺序）：

$$
\begin{aligned}
&v_e(1) = 0 \\
&v_e(2) = v_e(1) + w(1,2) = 0 + 3 = 3 \\
&v_e(4) = v_e(1) + w(1,4) = 0 + 6 = 6 \\
&v_e(3) = \max\{v_e(1) + w(1,3), v_e(2) + w(2,3)\} = \max\{0 + 3, 3 + 4\} = 7 \\
&v_e(5) = \max\{v_e(2) + w(2,5), v_e(3) + w(3,5)\} = \max\{3 + 5, 7 + 4\} = 11 \\
&v_e(6) = \max\{v_e(3) + w(3,6), v_e(4) + w(4,6)\} = \max\{7 + 3, 6 + 5\} = 11 \\
&v_e(7) = \max\{v_e(5) + w(5,7), v_e(6) + w(6,7)\} = \max\{11 + 3, 11 + 7\} = 18
\end{aligned}
$$

关键路径是从源点到汇点的最长路径。为了得到它，还需要计算每个事件的最迟发生时间 $v_l(k)$。定义汇点的最迟发生时间等于其最早发生时间，即 $v_l(7) = v_e(7) = 18$。对于其他任意顶点 j，其计算公式为：

$$v_l(j) = \min\{v_l(k) - w(j,k)\}$$

其中，k 是所有由 j 指向的直接后继顶点。

最迟发生时间 v_l 计算过程（按某个逆拓扑序，这里选用第三个）：

$$
\begin{aligned}
&v_l(7) = 18 \\
&v_l(6) = v_l(7) - w(6,7) = 18 - 7 = 11 \\
&v_l(5) = v_l(7) - w(5,7) = 18 - 3 = 15 \\
&v_l(4) = v_l(6) - w(4,6) = 11 - 5 = 6 \\
&v_l(3) = \min\{v_l(5) - w(3,5), v_l(6) - w(3,6)\} = \min\{15 - 4, 11 - 3\} = 8 \\
&v_l(2) = \min\{v_l(3) - w(2,3), v_l(5) - w(2,5)\} = \min\{8 - 4, 15 - 5\} = 4 \\
&v_l(1) = \min\{v_l(2) - w(1,2), v_l(3) - w(1,3), v_l(4) - w(1,4)\} = \min\{4 - 3, 8 - 3, 6 - 6\} = 0
\end{aligned}
$$

关键事件是那些最早发生时间与最迟发生时间相等的事件（即时间余量 $v_l(i) - v_e(i) = 0$ 的顶点）。

顶点	$v_e(i)$	$v_l(i)$	时间余量 $(v_l - v_e)$	**关键事件**
1	0	0	0	是
2	3	4	1	否
3	7	8	1	否
4	6	6	0	是
5	11	15	4	否
6	11	11	0	是
7	18	18	0	是

由关键事件（1, 4, 6, 7）构成的路径即为关键路径。长 18。

五、算法设计题（30 分）

1. 数组 nums 中仅有 0、1、2、3 四种 int 元素，希望实现对此数组的最高效率的逆序排序。

   (1) 设计并描述排序思路，给出伪代码。

   (2) 求出你设计的排序算法对应的时空复杂度

   对仅包含 0, 1, 2, 3 四种确定整数的数组，最高效的排序方法是**计数排序**。该算法不基于比较，而是统计每个元素的出现次数，然后根据统计结果直接重构数组。

   创建一个大小为 4 的辅助数组 counts。遍历原始数组 nums，当遇到一个元素时，就在 counts 数组中对应索引的位置上加一。根据 counts 数组的统计结果，按顺序将正确数量的元素填充回原数组 nums 中。

```
function Sort0123(nums):
  // 1. 初始化计数数组
  let counts = [0, 0, 0, 0]

  // 2. 统计0, 1, 2, 3的出现次数
  for each element in nums:
    counts[element] = counts[element] + 1

  // 3. 根据计数结果，重构原数组
  let index = 0
  for value from 3 to 0:
    for count from 1 to counts[value]:
      nums[index] = value
      index = index + 1

  return nums
```

   算法包含两次遍历：一次是遍历长度为 n 的原数组进行计数，另一次是根据计数值重构数组（总共写入 n 个元素）。总时间复杂度为 $O(n)+O(n)=O(n)$。算法需要一个大小固定为 4 的辅助数组 counts。其空间占用不随输入数组 nums 的规模 n 变化，因此空间复杂度为定数级，即 $O(1)$。

2. 现欲在 3 阶 B 树中查找在给定的 [ts, te] 区间内的关键字个数。B 树节点的结构如下：

```
struct node{
    int key[2];
    struct node* children[3];
    int n;
}
```

(1) 设计并描述你的求解思路，给出伪代码。

(2) 求出你设计的算法对应的时间复杂度。

B 树的结构是递归的，设计一个递归遍历算法当然是舒服的。思路如下：

从根节点开始递归搜索。在当前节点中，从左到右检查每个关键字 key[i]。

递归左子：若 ts < key[i]，说明其左子树 children[i] 可能包含目标，递归搜索该子树。

统计数目：若 ts ≤ key[i] ≤ te，则计数器加一。

提前剪枝：若 key[i] > te，则该节点后续的关键字及其右子树均无需搜索，提前终止。

递归右子：遍历完节点所有关键字后，若 te ≥ 最后一个关键字，则需要递归搜索最右子树。

```
function countInRange(node, ts, te):
  if node is NULL: return 0

  let total_count = 0
  let i = 0

  while i < node->n:
    // 提前剪枝
    if te < node->key[i]: return total_count
    // 统计数目
    if ts <= node->key[i] <= te: total_count += 1
    // 递归左子
    if ts < node->key[i]:
      total_count += countInRange(node->children[i], ts, te)

    i = i + 1
  // 递归右子，注意以下内容不在while循环中
  if te >= node->key[i-1]:
    total_count += countInRange(node->children[i], ts, te)

  return total_count
```

假定 N 是 B 树中关键字的总数，k 是位于区间 [ts,te] 内的关键字总数。那么时间复杂度: $O(\log N + k)$。算法首先花费 $O(\log N)$ 的时间找到搜索区间的起点，然后遍历所有包含这 k 个结果的节点和关键字。

3. 给定正整数 n，设计动态规划算法求出满足以下条件的序列总数。

条件：序列中的元素不是 1 就是 2；不能出现相邻的 2；序列中所有元素之和为 n。

(1) 给出动态规划算法中优化子结构。

(2) 给出动态规划算法中的递推关系。

(3) 给出动态规划算法对应的伪代码。

(4) 给出动态规划算法的时空复杂度。

该问题有最优子结构。一个满足条件且和为 n 的序列总数，可以由满足条件的和更小的序列（如 $n-1$）的总数推导得出。具体地，将问题分解为求解所有和为 n 且以 1 结尾的序列数量，以及所有和为 n 且以 2 结尾的序列数量。这两部分之和即为最终解。

定义两个函数，

- $I(i)$: 表示和为 i 且以 1 结尾的合法序列的总数。
- $II(i)$: 表示和为 i 且以 2 结尾的合法序列的总数。

则该问题的递推关系如下：

$$I(i) = I(i-1) + II(i-1) \qquad \text{(在和为}i-1\text{ 的任意合法序列后添加 1)}$$

$$II(i) = I(i-2) \qquad \text{(在和为}i-2\text{ 且以 1 结尾的序列后添加 2)}$$

基础情况：$I(1) = 1, II(1) = 0$。为方便计算，可定义 $I(0) = 1, II(0) = 0$。

```
function countSequences(n):
  if n == 0: return 1
  if n == 1: return 1

  // DP数组
  let I = array of size (n + 1)
  let II = array of size (n + 1)

  // 基础情况
  let I[0] = 1, II[0] = 0, I[1] = 1, II[1] = 0

  // 迭代计算
  for i from 2 to n:
    I[i] = I[i-1] + II[i-1]
    II[i] = I[i-2]

  return I[n] + II[n]
```

算法通过一个循环计算结果，循环体内是常数时间操作，因此总时间复杂度为 $O(n)$。上述伪代码使用了两个大小为 $n+1$ 的数组，空间复杂度为 $O(n)$。

事实上，这个算法的空间复杂度可以压得更紧。由于计算当前状态仅需前两项的状态，因此无需存储整个数组，只需用几个变量来记录前序状态即可。

```
function countSequencesCompact(n):
  if n == 0: return 1
  if n == 1: return 1

  // 初始化 i=1 和 i=0 时的状态变量
  let I_prev1 = 1   // 代表 I(1)
  let II_prev1 = 0  // 代表 II(1)
  let I_prev2 = 1   // 代表 I(0)

  for i from 2 to n:
    let I_curr = I_prev1 + II_prev1
    let II_curr = I_prev2
    I_prev2 = I_prev1
    I_prev1 = I_curr
    II_prev1 = II_curr

  // 循环结束后，I_prev1 和 II_prev1 中存储的是 I(n) 和 II(n) 的值
  return I_prev1 + II_prev1
```

空间复杂度为 $O(1)$。