单项选择题

1. 程序的时间复杂度 A

```
int i = 0, s = 0;

while (s < n) {
    s = s + i;
    i++;
}
```

显然s = 0 + 1 + 2 + ... + (i-1) = i*(i-1)/2，当s >= n时，i*(i-1)/2 >= n，解得i约等于$\sqrt{2n}$，因此该程序的时间复杂度为O($\sqrt{n}$)。

2. 操作序列 B

q指向节点A，p指向A的后继B，s指向被插入的节点X，则在A和B之间插入X的操作序列为：

$q \rightarrow next = s$
$s \rightarrow next = p$

3. 不可能的出栈序列 D

abcdef依次入栈，不允许连续三次出栈，以下出栈序列不可能出现的是：

afedcb，此序列为a入栈出栈，bcdef入栈后fedcb出栈，违反了不允许连续三次出栈的规则。

4. 二维对应一维的下标 A

A[1...m][1...n]是一个二维数组，按行优先存储在一维数组B[1...m*n]中，则A[i][j]对应B中的下标为：

下标 = (i-1)*n + j

- 建议带入特殊值验证，例如A[0][0]

5. 最小生成树的正确叙述 A

- 最小生成树的代价唯一
  - 正确，最小生成树是有最优解的，最优解的代价是唯一的
- 所有权值最小的边一定会出现在所有的最小生成树中
  - 错误，给出反例，一个三角形ABC其边等长，总有一条边不会出现在最小生成树中
- Prim算法从不同顶点开始得到的结果相同
  - 错误，给出反例同上
- Prim算法和Kruskal算法得到的最小生成树总不同
  - 错误，给出反例，一个只有两个顶点一条边的图，两种算法得到的结果相同

6. 不是前缀编码 D

D中110是1100的前缀。

7. 时间复杂度 C

```
n个节点e条边的有向图，使用邻接表存储，进行BFS
```

时间复杂度为O(n+e)，因为每个节点和每条边都要访问一次。

8. 二分法查找比较次数 C

```
有序表(13, 18, 24, 35, 47, 50, 62)中查找24，使用二分法
```

第一次比较：中间元素35 > 24，继续在左半部分查找(13, 18, 24)
第二次比较：中间元素18 < 24，继续在右半部分查找(24)
第三次比较：找到24

9. 拓扑排序 A

```
有向无环图，边(1,2),(2,3),(3,4),(1,4)
```

拓扑排序结果为1,2,3,4

10. 二叉排序树的深度 A

二叉排序树中插入以下结点的序列：(34, 76, 45, 18, 26, 54, 92)

插入顺序如下：

```
      34
     /  \
   18    76
     \   /  \
     26 45  92
          \
          54
```

该二叉排序树的深度为4。

填空题

1.

栈 是一种仅在表尾进行插入和删除操作的线性表，遵循后进先出。

2.

顺序循环队列Q[0 ... M-1]其头尾分别为F和R，F指向头元素前一位置，R指向尾元素位置，则队列中元素个数为 (R - F + M) % M 。

3.

广义表((a), ((b), c), (((d))))的深度为 4 。

- 建议使用栈辅助计算，遇到左括号深度加1，遇到右括号深度减1，记录最大深度。

4.

高度为5的完全二叉树中至少有 16 个节点。

- 课本上对于完全二叉树的定义是：深度为K的，有n个结点的二叉树， 当且仅当其每一个结点都与深度为K的满二叉树中编号从1至n的结点一一对应时，称之为完全二叉树。

5.

二叉树的先序遍历序列为ABDGHCEFI，中序遍历序列为GDHBAECIF，则该二叉树的后序遍历序列为 GHDBEIFCA 。

生成的树如下：

```
        A
      /   \
     B     C
    /     / \
   D     E   F
  / \       /
 G   H     I
```

- 先序遍历的第一个节点为根节点A，在中序遍历中找到A，左边为左子树BDGH，右边为右子树CEFI。依次类推。

6.

含有n个节点的二叉链表(无头节点)中共有 n+1 个空指针，如果加上中序线索，则共有 2 个空指针。

- 对于课本定义的线索二叉树，如果节点没有左右孩子，则其左右指针均指向前驱和后继，因此总共只有最前和最后一个节点的指针为空。

7.

一棵哈夫曼树共有2001个节点，则该哈夫曼树有 1001 个叶子节点。

- 总节点数 = 2 * 叶子节点数 - 1

8.

图的邻接表如下，则从v1开始的DFS序列为 v1, v2, v3, v6, v4, v5 ， BFS序列为 v1, v2, v4, v5, v3, v6 。

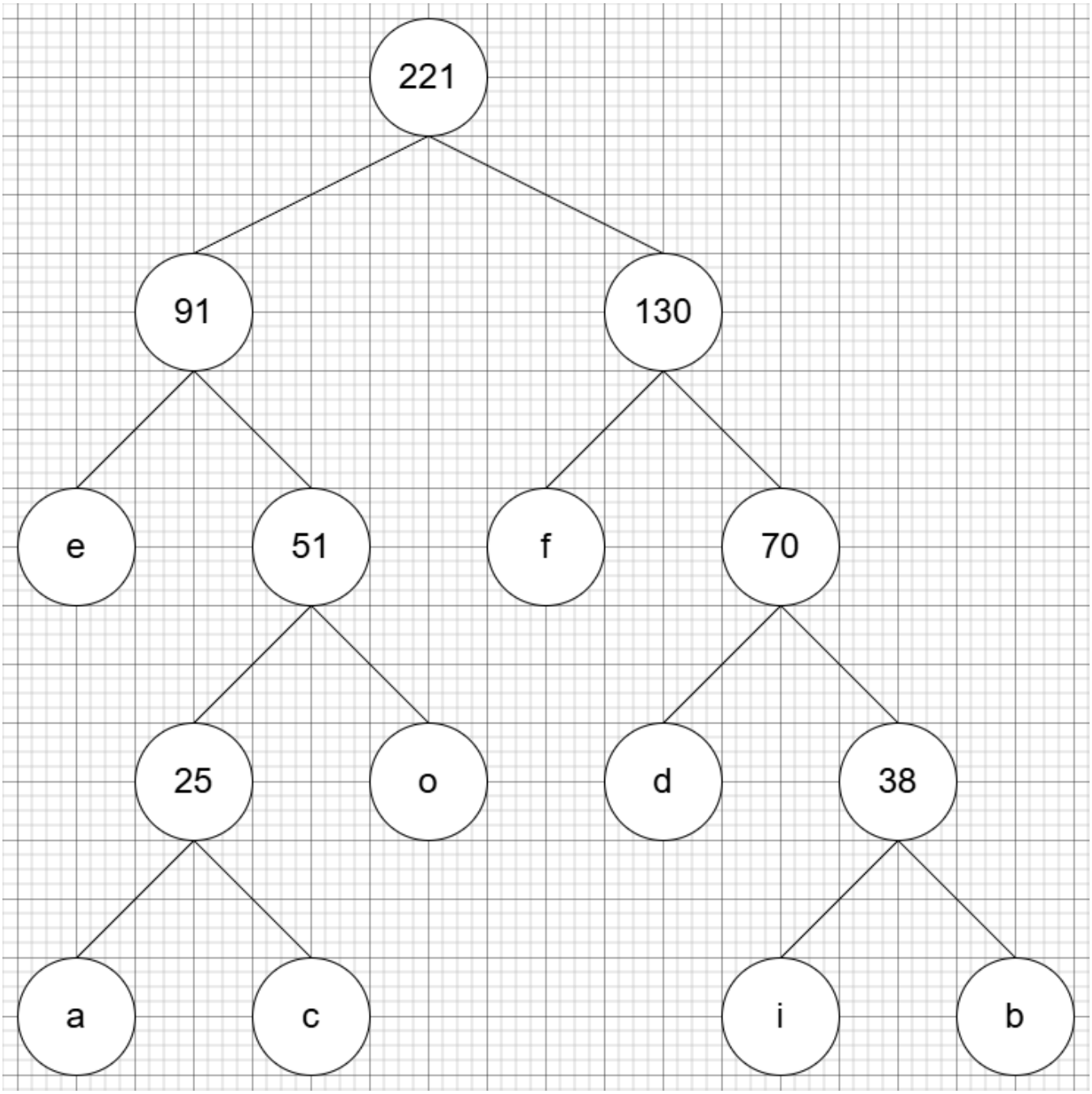
```
v1 -> v2 -> v5 -> v4
v2 -> v3 -> v5
v3 -> v6
v5 -> v4 -> v6 -> v3
```

- 部分顺序不唯一，视具体实现而定，此处遵循从小到大访问顺序

应用题

1. 霍夫曼树

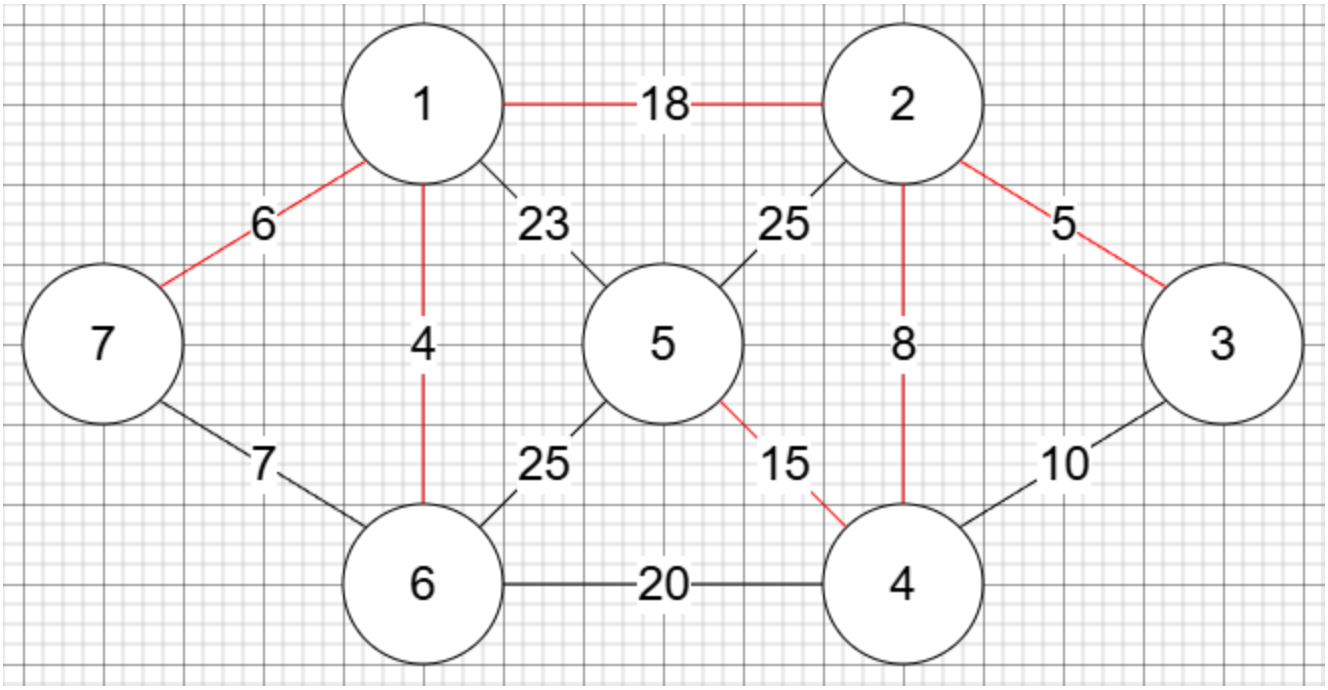
字符	a	b	c	d	e	f	o	i
频率	10	20	15	32	40	60	26	18



a	b	c	d	e	f	o	i
0100	1111	0101	110	00	10	011	1110

- 霍夫曼树的左右顺序可以不同，因此编码可能不同，但编码长度一定相同，此处遵循左小右大的规则

2. Prim算法



- Prim算法从不同顶点开始可能得到不同的最小生成树，此处从A开始，注意如果边长互不相等，则最小生成树唯一

3. B-Tree

见附件gif

- 数据结构可视化
  https://www.cs.usfca.edu/~galles/visualization/Algorithms.html

---

算法设计题

1. 给定连通的无向无权图，打印两个不同顶点s和t之间的所有简单路径，图以邻接矩阵存储(简单路径：不包含重复顶点的路径)。

思考思路：

- 如何确定两个点之间的路径：
  - 枚举所有路径

- 如何枚举所有路径：
  - BFS/DFS遍历
- BFS如何实现：
  - 使用队列实现BFS
  - 使用Visited数组记录访问过的节点
  - 遇到t节点时，输出路径
  - 如何记录路径：
    - 对于搜索中的节点，记录其路径
    - 如果遇到重复节点，难以处理多种情况，并且队列本身记录路径比较复杂
    - 考虑转向DFS实现
- DFS如何实现：
  - 使用递归实现DFS
  - 使用Visited数组记录访问过的节点
  - 遇到t节点时，输出路径
  - 递归调用时，传入当前路径
  - 回溯时，删除当前节点
- 算法是否具有正确性：
  - DFS遍历所有可能路径，遇到t节点时输出路径，保证了所有简单路径均被输出
- (可选步骤)算法复杂度分析：
  - 时间复杂度：O(V + E) * P，其中P为简单路径数量
  - 空间复杂度：O(V)用于Visited数组和递归栈
  - 是否可以优化算法?

```
def DFS(graph, s, t, visited, path):
    visited[s] = True
    path.append(s)

    if s == t:
        print("->".join(map(str, path)))
    else:
        for neighbor in range(len(graph)):
            if graph[s][neighbor] == 1 and not visited[neighbor]:
                DFS(graph, neighbor, t, visited, path)

    path.pop()
    visited[s] = False

DFS(graph, start, target, visited, path)
```

- 考试时需要使用C语言进行编写，注意语法差异，使用其他语言可能导致扣分
- 答题时注意注释和代码格式，优先保证展示你的思路和逻辑，万不得已时可以使用伪代码，可能会扣分
- 注意书写，如果助教无法辨认你的代码，可能会扣分

2. 在两个有序递增数组A[m]B[n]中寻找中位数(提示：时间复杂度可以达到O(log(m + n)))

思考思路：

- 如果不考虑时间复杂度，如何寻找中位数：
  - 合并两个数组，然后找到中位数
- 如何合并两个有序数组：
  - 使用双指针法，分别指向两个数组的起始位置
  - 比较两个指针所指元素，较小的元素加入合并后的数组，并移动对应指针
- 如何找到中位数：
  - 如果合并后的数组长度为奇数，中位数为中间元素
  - 如果合并后的数组长度为偶数，中位数为中间两个元素的平均值
- 时间复杂度分析：
  - 合并两个数组的时间复杂度为O(m + n)
  - 找到中位数的时间复杂度为O(1)

    - 总时间复杂度为O(m + n)，不满足要求
- 如何进行初步优化?
    - 可以考虑不完全合并，只合并到中位数位置
    - 使用双指针法，直到合并到中位数位置
    - 时间复杂度为O((m + n)/2)，仍然不满足要求(注意，时间复杂度仍然是O(m + n)，因为常数项不影响复杂度)
- 提示中提到可以达到O(log(m + n))，考虑使用二分法：
    - 是否可以先合并再二分?
    - 不能，因为合并本身是O(m + n)
- 是否可以在不合并的情况下使用二分法?
    - 转变思路：中位数作用，将数组均匀划分为两部分。两个数组的中位数，左边部分的元素均小于右边部分的元素
    - 即找到第一组的最大值≤第二组的最小值
    - 假定划分点在A数组的i位置，在B数组的j位置
    - 找到$\max\left(a_i, b_j\right) \leq \min\left(a_{i+1}, b_{j+1}\right)$
    - 即找到$a_i \leq b_{j+1}$且$b_j \leq a_{i+1}$
- 如何使用二分法找到划分点：
    - 已知$i + j = (m + n + 1)/2$，只需要在A数组中使用二分法找到i，然后计算j
    - 初始化i的范围为[0, m-1]
    - 计算$i = (left + right)/2$
      $j = (m + n + 1)/2 - i$
    - 检查条件：
        - 如果$a_i < b_{j+1}$，说明i太小，增加i，left = i + 1
        - 如果$b_j < a_{i+1}$，说明i太大，减少i，right = i - 1
    - 找到划分点，再根据奇偶进行微调

- 详细题解
  https://leetcode.cn/problems/median-of-two-sorted-arrays/solutions/2950686/tu-jie-xun-xu-jian-jin-cong-shuang-zhi-z-p2gd/
- 压轴题考点
    - 高级算法(二分，贪心，动态规划等)
    - 高级数据结构(堆，平衡树，并查集等)
    - https://oi-wiki.org/