针对性训练题（Typst 版）

—

1（链表指针操作 — 6 分）

已知双向链表用结构

```
struct Node {
  Node *prev, *next;
  int val;
};
```

链表中有节点序列 `... <-> A <-> p <-> q <-> B <-> ...`，其中 `p->next == q`，`p->prev == A`（A 可能为 `NULL` 表示头），`q->next == B`（B 可能为 `NULL` 表示尾）。已知 `p->prev != NULL` 且 `q->next != NULL`（即 A 和 B 都存在），且此链表不使用哨兵结点。

请完成以下任务（要求给出精确的赋值语句及顺序，并尽量使赋值数最少且无中间悬空指针导致的临时失效——不得使用额外内存分配，只能赋指针）：

(a) 写出将 `p` 与其后继 `q` 在链中交换位置 的最短序列（按 C 语句逐行列出，例如 `t = p->next;`）。交换后链表局部应变为 `... <-> A <-> q <-> p <-> B <-> ...` 且其他节点关系保持不变。

(b) 证明你给出的语句序列在执行过程中的每一步都保持了链表其它部分可达性（即不会丢失对 A、B 及其余节点的引用），并说明若 `A==NULL` 或 `B==NULL`（极端边界）时需要如何修改你的序列（简要列出条件分支要改的语句）。

评分要点：语句顺序正确性（3 分）、最少赋值（1 分）、中间状态安全性证明（2 分）。

—

2（最小堆 — 6 分）

给定初始数组（下标从 1 开始）：

```
索引: 1  2  3  4  5  6  7  8  9
值:   20 15 30 40 18 50 35 25 10
```

以二叉堆的数组表示（完全二叉树），要求：

(a) 按“自底向上 SiftDown”的标准过程把该数组整理成最小堆。请逐步写出在每次对某个下标执行 SiftDown 后数组的完整状态（从最后一个非叶子节点到根，按下标顺序处理，每一步都写出变化后的数组）。注明对哪个下标做了 SiftDown 及比较/交换的关键动作。

(b) 在得到最小堆后，从该堆插入新元素 7，并按 SiftUp（上浮）过程调整至堆序。写出插入后每次交换后的数组状态，直到完成。并统计（i）SiftUp 总共进行了多少次父子交换，(ii) 插入完成后的最终数组表示。

要求：每一步变化都要写出数组（完整 9 或 10 个元素），并标注被交换的索引对。答案只写过程与最终数组，不要写实现代码。

—

3（并查集 — 6 分）

对集合 `{0,1,2,3,4,5,6,7,8,9}`（初始每个元素单独成一个等价类），采用并查集（Union-Find）。合并规则明确如下：

- 使用 按规模（size）合并：把节点数较小的树并入节点数较大的树（大树根作为新根）；
- 若两个集合规模相同，则将根值较大的并入根值较小（根值小的成为新根）；
- `find(x)` 要实现 路径压缩（即每次 find 后路径全部指向根）。

现在按下列顺序执行 `union(a,b)`（每步合并时先调用 `find(a)` 和 `find(b)`），请在每一步写出当前的 **`parent[]` 与 `size[]`**（以数组形式列出 0..9），并画出合并后每个树的根结构（用 **`root: children...`** 的形式）：

合并序列（按此顺序执行）：

`(2,5), (1,2), (3,4), (0,9), (2,8), (7,6), (1,3), (0,1), (9,7)`

此外，在全部合并完成后，再依次对节点 8、5、3 调用 `find` 并说明这些 `find` 操作如何影响 `parent[]`（即路径压缩带来的变化）。

评分要点：每步 parent/size 正确（4 分），最后路径压缩效果正确展示（2 分）。

—

4（KMP — 6 分）

给定模式串（长度 15）：

`P = "abacababababacaba"`

采用 KMP 的常见 `next[]` 构造（这里使用 `next[0] = -1` 的 convention），并考虑“优化版 next”（记作 `nextval[]`），优化规则为：当 `P[i] == P[next[i]]` 时令 `nextval[i] = nextval[next[i]]`，否则 `nextval[i] = next[i]`。

任务：

(a) 请计算并写出 `next[i]`（对 i=0..14）与对应的 `nextval[i]`（对 i=0..14）。格式为两行对齐表格或数组表示均可。说明你使用的构造细节（例如 `next[0]=-1`，循环不变量等）。

(b) 选取模式下某一典型失配下标（你认为能体现优化收益的位置），举例说明改用 `nextval` 能在匹配过程中避免多少次字符比较（给出原本用 `next` 的情形与用 `nextval` 的情形各进行一次示例比较计数）。

要求：计算要精确，示例计数需说明比较序列；不写代码实现。

—

5（栈操作序列 — 6 分）

对输入序列为 `1,2,3,4,5,6,7`（按这个顺序依次入栈），设 S 表示 `push(next input)`，X 表示 `pop()`（输出弹出元素）。初始栈空，最终栈也空。

(a) 判断下面给出的 输出序列 是否可以通过某个合法的 S/X 序列得到（如果可以，给出对应的 S/X 序列；如果不可以，给出严谨的反证）：

输出序列 1：**`2,1,4,3,6,5,7`** 输出序列 2：`3,2,1,5,4,7,6`

(b) 对于任意输出序列，给出一个必要且充分的判别条件（简明数学表述），并简要证明该条件的正确性（证明不必过长，但需有关键论证思路）。

评分要点：单个序列正确判定与 S/X 序列给出（或不可能证明）（各题 2.5 分），充要条件陈述与证明（1 分）。

—

6（哈夫曼编码与阈值分析 — 8 分）

文件中包含字符集合 {a,b,c,d,e,f,g}。当前已知频率（归一化，总和为 1）为：

p(a)=0.05, p(b)=0.08, p(c)=0.12, p(d)=0.15, p(e)=0.20, p(f)=0.10, p(g)=0.30

规则：构造哈夫曼树时若两节点权值相等，请把“权值较小的放在左子树（或按题约定左 ⩽ 右）”。固定长度编码（若对 7 个字符使用二进制无前缀码）需要 ceil(log2 7) = 3 位/字符。

任务：

(a) 按哈夫曼算法（每步合并权值最小的两个结点）给出每一步的合并操作与新权值，并由此写出每个字符的哈夫曼编码长度 ℓ(x)（不需要给出实际 0/1 编码，只要写出每字符的长度）。然后计算平均编码长度 L_H = Σ_x p(x) ℓ(x)（保留小数到 4 位）。

(b) 假设其他字符的相对频率（除了 g）保持相对比例不变（即若 g 的频率从 0.30 变为 x，则其它字符的频率按常数因子缩放，使总和为 1），且哈夫曼树构造的合并顺序与编码方案保持不变（即树结构没有因为频率微调而变化）。求：当 g 的频率下降到多少（求 x 的临界值），会出现固定长度编码（3 bits/字符）比当前哈夫曼编码更节省空间（即当 L_H ≥ 3 时，固定长度反而更优）。请列出建立不等式的推导过程并求出数值解（精确到小数点后 4 位）。

说明：这道题考察哈夫曼树构造、平均长度计算以及参数敏感性分析。你可以把第（b）中“哈夫曼树结构保持不变”的假设作为题设前提（即便在实际中频率微调可能改变合并顺序），按此前提进行推导。

—