2024 年 PKU 数算 A 期中参考解答（From Qiyu Zhang）

1. ✗ -2 D(?)

解：C，抽象数据类型（ADT）的核心思想是定义数据的逻辑特性和操作行为，而不关心其底层的具体实现细节，它的主要组成部分包括：

- 数据对象 (B 选项)：指具有相同性质的数据元素的集合
- 数据关系 (D 选项)：描述了数据元素之间的逻辑关系（如线性、树状、图状等）
- 一组操作 (A 选项)：定义在数据对象上的一系列基本操作（如插入、删除、查找等），并规定了这些操作的功能和约束

2. ✓ A

$$\Sigma_{i=1}^{N}\left(\frac{N}{i}-1\right) \leq \Sigma_{i=1}^{N}\left[\frac{N}{i}\right] \leq N\Sigma_{i=1}^{N}\frac{1}{i}$$

$$\Sigma_{i=1}^{N}\frac{N}{i}-N \leq O \leq N\left(1+\frac{1}{2}+\frac{1}{3}+\ldots+\frac{1}{N}\right)$$

而

$$\left(1+\frac{1}{2}+\frac{1}{3}+\ldots+\frac{1}{N}\right) \sim \ln N$$

原因是

$$\ln\left(1+\frac{1}{n}\right) < \frac{1}{n}, \ln(n+1) < S$$

并且

$$\left(1+\frac{1}{2}+\frac{1}{3}+\ldots+\frac{1}{n}\right) < \ln 2n$$

由数学归纳法，只需证

$$\frac{1}{n+1} < \ln\left(1+\frac{1}{n}\right)$$

而

$$\ln n > 1-\frac{1}{n}$$

得证

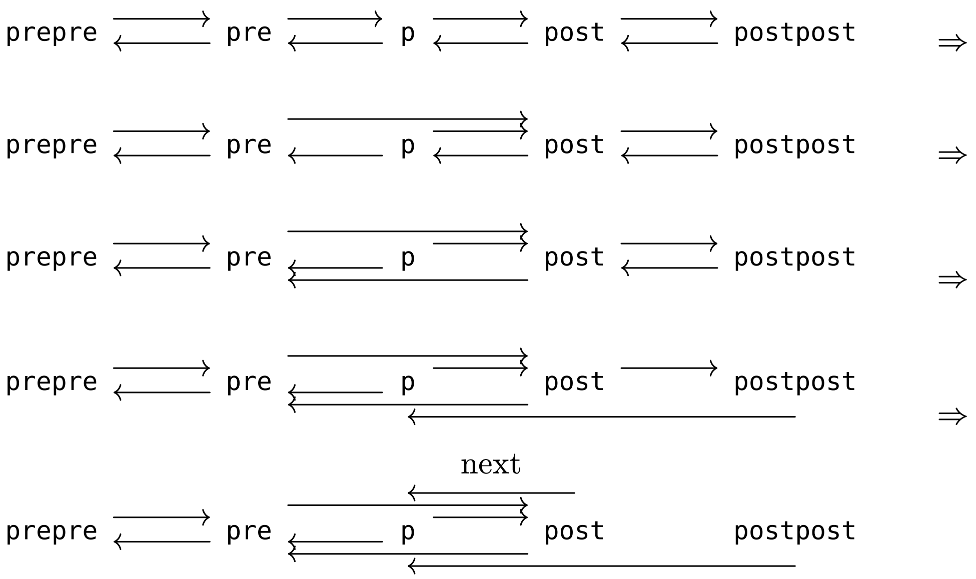
故

$$O(N\log N)$$

3. ✗ -2 B(? 双向链表数据结构不熟)

解：C，

- 选项 A：操作序列中，p->next->next=p；会将 q->next 指向 p，随后 p->next=p->next->next；会导致 p->next 指向自身（因为 p->next 是 q，而 q->next 刚被设为 p），造成循环引用，错误。



- 选项 B：与选项 A 类似，p->next->next=p；设置 q->next 指向 p 后，p->next=p->next->next；同样使 p->next 指向自身，错误。
- 选项 C：操作序列正确且完整地处理了所有指针调整：

1. p->next->next->prev=p；设置 q 的后继结点的 prev 指向 p（即 next_next->prev = p）
2. p->next->prev=p->prev；设置 q 的 prev 指向 p 的前驱结点（即 q->prev = prev）
3. p->prev=p->next；设置 p 的 prev 指向 q（即 p->prev = q）
4. p->prev->prev->next=p->prev；设置 p 的前驱结点的 next 指向 q（即 prev->next = q）
5. p->next=p->prev->next；设置 p 的 next 指向 q 的后继结点（即 p->next = next_next）
6. p->prev->next=p；设置 q 的 next 指向 p（即 q->next = p）

4. ✗ -2 C(? 不熟悉线性表链表基本概念和特点)

解：D，

- 选项 A:顺序表存储密度为 1（100% 利用率），而链表每个结点都需额外存储指针，存储密度小于 1，空间利用率更低
- 选项 B:找到插入位置可用二分查找（O(log n)），但插入操作本身需要移动后续所有元素，平均需移动 n/2 个元素，时间复杂度为 O(n)
- 选项 C:删除元素后，需要移动该元素后面的所有元素以保持连续性。平均仍需移动 n/2 个元素，时间复杂度为 O(n)
- 选项 D:顺序表支持随机存取，按索引访问元素的时间复杂度为 O(1)。而链表访问元素需遍历，时间复杂度为 O(n)

5. ✗ -3 3(?BST 允许右节点为空吗)

解：6， 通过动态规划方法计算（类似卡特兰数计算但加上约束），定义 f(n) 为 n 个节点满足条件的 BST 数量：

- $f(0) = 1$(空树)
- $f(1) = 1$
- $f(2) = 1$
- $f(3) = 2$
- $f(4) = 3$
- $f(5) = 6$

对于 n=5，根节点可能为 3、4 或 5（因为根节点必须满足左子树节点数 ≥ 右子树节点数）：

- 根节点为 3：左子树有 2 个节点（值 1,2），右子树有 2 个节点（值 4,5），方式数为 f(2)×f(2)=1×1=1
- 根节点为 4：左子树有 3 个节点（值 1,2,3），右子树有 1 个节点（值 5），方式数为 f(3)×f(1)=2×1=2
- 根节点为 5：左子树有 4 个节点（值 1,2,3,4），右子树空，方式数为 f(4)×f(0)=3×1=3
- 总数为 6

6. ✗ -6 (?忘记最小堆怎么构建了)

解：[5, 8, 15, 12, 28, 30, 17, 23]；[4, 5, 15, 8, 28, 30, 17, 23, 12]

```
        23                         23                         23
      /    \                     /    \                     /    \
    12      17                 12      17                 12      *15*
   /  \    /  \               /  \    /  \               /  \    /  \
  5   28  30  15           *5*   28  30  15             5   28  30  *17*
 /                          /                          /
8                ⇒      *8*              ⇒          8                   ⇒

        23                         23                        *5*
      /    \                     /    \                     /    \
     5      15                *5*      15               *23*      15
    /  \   /  \               /  \    /  \              /  \     /  \
 *12*  28 30  17           *8*   28  30  17            8   28   30  17
  /                         /                         /
*8*              ⇒        12             ⇒          12                  ⇒

         5                          5                         5
      /    \                     /    \                     /    \
    *8*     15                  8      15                  8      15
    /  \   /  \               /  \    /  \               /  \    /  \
 *23*  28 30  17           *12*  28  30  17            12  28  30  17
  /                         /                          / \
12               ⇒       *23*            ⇒          23 *4*          ⇒   …

              *4*
            /     \
         *5*       15
         /  \     /  \
      *8*   28   30  17
       / \
⇒   23*12*
```

7. ✓ AD

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

8.

(1) ✓任意一个 X 及以前的序列中 X 的个数不大于 S 的个数（或：对任意从头开始的子序列，$|S| \geq |X|$）

(2) ✓不可能；✗ -2 理由（？不知道）

（对于两个序列T_1, T_2，假定前n个操作都相同，第$n+1$个操作T_1为S，T_2为X，假定此时栈顶元素为a，即将输入元素为b，则对于T_1，a将在b之后出栈；对于T_2，a将在b之前出栈。故输出序列不同）

9.

(1) ✗ -2(? strcpy 是啥（字符串复制函数）？地址为啥可以用<（16 进制数）？)

解：意味着目标内存的起始地址位于源字符串的内存中间，形成了内存重叠；应该采用从后向前反向拷贝的方式。

(2) ✗ -4（？优化是啥）

	a	b	a	b	a	b	a
前	0✗ -1	0	1✗ 0	2✗ 1	3✗ 2	4✗ 3	5✗ 4
后	-1	0	-1	0	-1	0	-1

解：第 i+1 个元素~next[i]表示模式串 P[0..i]中，最长相等真前缀和真后缀的长度
优化：优化后的 nextval 数组旨在解决基础 next 数组在某些情况下存在的不必要的回溯。
假设模式串为“ABAB”，其 next 数组（方法 1）为[-1,0,0,1]：

- 当在索引 3（字符 B）失配时，跳转到 next[3]=1（字符 B）
- 但此时原 B 已经和文本不匹配，跳转后的 B 必然再次失败

优化的核心思想：若跳转后的字符与原字符相同，则继续向前跳转

```
if pattern[i] == pattern[next[i]]:
    nextval[i] = nextval[next[i]]  # 递归向前查找
```

```
else:
    nextval[i] = next[i]
```

10.

(1)✓

前：ABDEGCFHI

中：DBGEACHFI

后：DGEBHIFCA

(2)✓ 先看前序第一个 A 为根；再看中序，根左为左子树，根右为右子树；如此递归操作

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

故后序：DGHEBFCA

11.✗ -8

(1)(? Huffman 树构建忘记)

解：

```
          100
        /      \
   44             56
  /  \           /  \
19   25        26   30
/ \            / \
9 10          14 12
              / \
             6   8
```

(2)

解：$\frac{[4\times(6+8)+3\times(12+9+10)+2\times(25+30)]}{100}=2.59$

(3)

解：设$g=x$,其它字符频率扩大$\frac{1-x}{0.7}$倍，列式令平均长度=3，解出$x=-\frac{11}{59}$

12✗ -3.(? 路径压缩优化未学)

解：用并查集（Union-Find）数据结构进行合并操作。

合并规则：

- 重量权衡合并：合并时，将节点数较少的树并入节点数较多的树（节点数多的树根作为新根）。若两棵树节点数相同，则将根值较大的树并入根值较小的树（根值小的作为新根）。

- 路径压缩：在 find 操作中，将路径上的所有节点直接指向根节点，以扁平化树结构，减少后续查找时间。
- 维护数组：
  - parent[i]：元素 i 的父节点索引。
  - size[i]：以 i 为根的树的节点数（仅对根节点有效）。

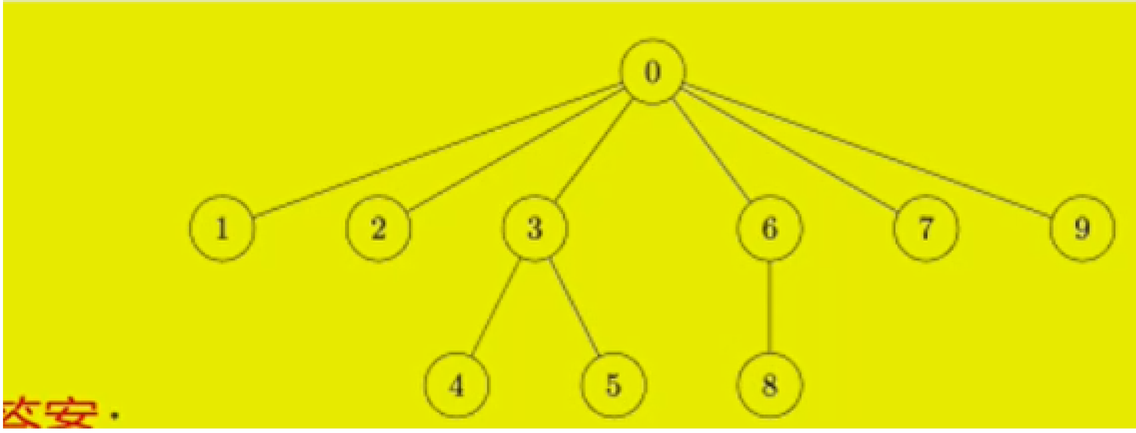
初始化：

- parent = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]（每个元素初始时自身为根）
- size = [1, 1, 1, 1, 1, 1, 1, 1, 1, 1]

合并过程：

步骤	等价对	操作说明	合并后 `parent` 数组（索引0
初始	-	每个元素独立	[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
1	(0, 2)	`find(0)=0`, `find(2)=2`。大小相同（size=1），根值0<2，故2并入0。	[0, 1, 0, 3, 4, 5, 6, 7, 8, 9]
2	(1, 2)	`find(1)=1`, `find(2)=0`（路径压缩：2直接指向0）。大小：size[0]=2 > size[1]=1，故1并入0。	[0, 0, 0, 3, 4, 5, 6, 7, 8, 9]
3	(3, 4)	`find(3)=3`, `find(4)=4`。大小相同（size=1），根值3<4，故4并入3。	[0, 0, 0, 3, 3, 5, 6, 7, 8, 9]
4	(4, 5)	`find(4)=3`, `find(5)=5`。大小：size[3]=2 > size[5]=1，故5并入3。	[0, 0, 0, 3, 3, 3, 6, 7, 8, 9]
5	(2, 5)	`find(2)=0`, `find(5)=3`。大小相同（size[0]=3, size[3]=3），根值0<3，故3并入0。	[0, 0, 0, 0, 3, 3, 6, 极, 8, 9]
6	(6, 7)	`find(6)=6`, `find(7)=7`。大小相同（size=1），根值6<7，故7并入6。	[0, 0, 0, 0, 3, 3, 6, 6, 8, 9]
7	(8, 7)	`find(8)=8`, `find(7)=6`（路径压缩：7直接指向6）。大小：size[6]=2 > size[8]=1，故8并入6。	[0, 0, 0, 0, 3, 3, 6, 6, 6, 9]
8	(2, 8)	`find(2)=0`, `find(8)=6`。大小：size[0]=6 > size[6]=3，故6并入0。	[0, 0, 0, 0, 3, 3, 0, 6, 6, 9]
9	(9, 7)	`find(9)=9`, `find(7)=0`（路径压缩：7→6→0，压缩后7直接指向0）。大小：size[0]=9 > size[9]=1，故9并入0。	[0, 0, 0, 0, 3, 3, 0, 0, 6, 0]

最终：



13. ✓

```
queue2.push(x)
!queue1.empty()
queue2.push(queue1.front())//???
queue1.front()
queue1.empty()
```

解:

```
class MyStack{
    queue<int> queue1,queue2;
    ......
    void push(int x){
        queue2.push(x);//1 最后用排除法，不能是queue1.push(x)，否则下面while语句无意义
        //只有1，3可以填进栈的语句
        while(!queue1.empty()){//2  这里其实可以凭语感，肯定是!queue.empty()；结合第三个突破点，必然为queue1
            queue2.push(queue1.front());//3 结合第三个突破点，必然是queue2.push(queue1.?)，也只有front了
            queue1.pop();//第三个突破点，此时queue1，queue2都要有元素
        }
        swap(queue1,queue2);//第二个突破点，最后进的元素在queue2顶部
    }
    int pop(){
        int r=queue1.front();
        queue1.pop();
        return r;
    }//第一个突破点，说明最后输入的元素在queue1的顶部，由此直接解决4，5
    int top(){
        return queue1.front();//4
    }
    bool empty(){
        return queue1.empty();//5
    }
};
```

14. ✓

- 求 input 的字符串的 next 数组，从而获取最长公共前后缀长度l_1
- 用字符串长度L减去最长公共前后缀长度l_1，获得最小循环节长度l_2
- 判断字符串长度L是否为最小循环节长度l_2的倍数，是，可；不是，不可

15.

(1) ✗ -2 遍历 `L[i],R[i]`, 让 `T[L[i]]=T[R[i]]=i`（未分析时间复杂度-1；未判断 L[i]!=0 -1）

(2) ✗ -3 从U开始一直向上搜索（依据T）至根节点，若此过程中遇到了V，是；否则不是（未写出转换为判断 V 是否为 U 的祖先-2；未分析时间复杂度-1）

16.（? 绕晕了）解：

- 1. 通过输入序列建树（4’）：根据输入序列还原二叉树（2’）；根据左子右兄弟还原树（2’）
- 2. BFS，从右到左即为镜面顺序（4’）：正确 BFS（1’）；输出时从左到右入队（3’）
- 3. 分析时间复杂度（2’）：$\boldsymbol{O(N)}$，建二叉树访问各结点 1 次，还原时 1 次，BFS 1 次

17.

证：设叶节点数目为m，共有$h(h \geq 1, k \geq 2)$层，则

$$n = 1 + k + k^2 + \ldots + k^{h-1}$$

$$m = k^h$$

那么

$$n = \frac{1 - k^h}{1 - k}$$

故

$$n(k-1) + 1 = k^h - 1 + 1 = k^h = m$$

证毕。