北京大学信息科学技术学院试题

考试科目：数据结构与算法 B 姓名：_____ 学号：_____

考试时间：2024 年 6 月 18 日 任课教师：_____

请将答案撰写正在答题纸上！考试结束后将试卷和答题纸一并上交！

一. 选择题（30 分，每小题 2 分）

1. 设栈的输入序列是 1 2 3 4 5，则（ B ）不可能是其出栈序列。

   A. 23415　　B. 54132　　C. 23145　　D. 15432

2. 对线性表进行二分法查找，其前提条件是（ C ）。

   A. 线性表以链接方式存储，并且按关键码值排好序

   B. 线性表以链接方式存储，并且按关键码值的检索频率排好序

   C. 线性表以顺序方式存储，并且按关键码值排好序

   D. 线性表以顺序方式存储，并且按关键码值的检索频率排好序

3. 对于二叉树中的一个结点 N，如果其左子树的高度为 h1，右子树的高度为 h2，则以 N 为根结点子树高度为（ C ）。

   A. max(h1, h2)　　B. min(h1, h2)　　C. max(h1, h2) + 1　　D. min(h1, h2) + 1

4. 在插入排序算法中，如果待排序的序列已经是有序的，则插入排序算法的时间复杂度为（ C ）。

   A. $O(n^2)$　　B. $O(n \log n)$　　C. $O(n)$　　D. $O(1)$

5. 下列哪个概念属于存储结构？（ B ）。

   A. 线性表　　B. 链表　　C. 栈　　D. 队列

6. 在二分查找算法中，每次比较后都将搜索范围缩小一半，若要在长度为 n 的数组中查找一个元素，最坏情况下需要比较多少次（ D ）。

   A. n　　B. $\lfloor n/2 \rfloor$　　C. $\lfloor \log n \rfloor$　　D. $\lfloor \log n \rfloor + 1$

7. 在一个具有 n 个结点的有序单链表中插入一个新结点并仍保持其有序，其平均时间复杂度为（ C ）。

   A. $O(n \log n)$　　B. $O(1)$　　C. $O(n)$　　D. $O(n^2)$

8. 假设有 4 棵结点关键码为整数的二叉树，若它们的中序遍历序列分别如下， 请问其中可能是二叉搜索树（二叉排序树）的是（ B ）。

   A. 5, 3, 1, 2, 4　　B. 1, 2, 3, 4, 5　　C. 5, 3, 4, 2, 1　　D. 1, 3, 5, 4, 2

9. 假设线性表有 2n（n>100）个元素，以下操作（ D ）在单链表上实现比在顺序表上实现效率更高。

   A. 在表中最后一个元素的后面插入一个新元素

   B. 顺序输出表中的前 i 个元素

C. 交换表中第 i 个元素和第 n+i 个元素的值 (i=0, …, n-1)

D. 删除表中第 1 个元素

10. 用数组 Q[n] 实现循环队列，设队头的前一个元素的下标为 f ，队尾的下标为 r， 则该队列中元素总数是（ D ）。

A. r - f　　B. (n + f - r) % n　　C. n + r - f　　D. (n + r - f) % n

11. 对于一个拥有 21 条边的非连通无向图，其至少包括（ D ）个顶点。

A.5　　B.6　　C.7　　D.8

12. 若使用分治算法解决某一问题，每次把规模为 n 的问题分解为 2 个规模为 n/2 的子问题，将两个子问题的结果合并的时间开销为 O(n)，那么这个算法的总时间复杂度为（ B ）。

A. O(n)　　B. O(n log n)　　C. O(n^2)　　D. O(n^2 log n)

13. 有 n 个顶点 m 条边的无向图，下列说法中错误的是（ D ） 。

A 采用邻接表储存，储存该图的空间复杂度为 O (m+n) 。

B 若为稠密图，更倾向于采用邻接矩阵存储。

C 采用邻接表储存，删除一个顶点的最坏情况的时间复杂度为 O (n+m) 。

D 若为稀疏图，深度优先周游的时间复杂度低于广度优先周游的时间复杂度。

14. 利用栈将表达式 3*2^(4+2*2-6*3)-5 （其中^为乘幂）转换为后缀表达式过程中，当扫描到 6 时，运算符栈为 （ D ）。

A. *^(+*-　　B. *^-　　C. *^(+　　D. *^(-

15. 定义一棵没有 1 度结点的二叉树为满二叉树。对于一棵包含 k 个结点的满二叉树，其叶子结点的个数为（ C ）。

A. [k/2]　　B. [k/2]-1　　C.[k/2]+1　　D.以上三个都有可能

二． 判断（10 分，每小题 1 分；对填写“Y”，错填写“N”）

1. （ Y ）分治法和动态规划法都运用了将问题分解为规模较小的子问题的思想。
2. （ N ）在链表中查找和删除一个元素的时间复杂度都是 O(1)。
3. （ N ）相比于顺序存储，完全二叉树更适合用链式表示存储。
4. （ Y ）通常不能通过一棵二叉树的前序遍历结点序列和后序遍历结点序列来确定这颗二叉树的完整结构。
5. （ N ）二叉搜索树一定是满二叉树。
6. （ N ）归并排序算法一定比简单插入排序算法的执行效率高。
7. （ Y ）在待排序元素为正序情况下，直接插入排序可能比快速排序的时间复杂度更小。
8. （ Y ）一个有 n 个结点的无向图，最少有一个连通分量。
9. （ N ）图的遍历算法（如深度优先搜索 DFS 和广度优先搜索 BFS）都只能用于无向图。

10. （ Y ）若连通无向图的边的权值互不相同，其最小生成树只有一个。

三. 填空（20 分，每题 2 分）

1. 按照简单且高效的原则，如果经常需要在线性表头部进行插入和删除操作，最合适的存储结构是__链表__；如果需要随机存取，最合适的存储结构是 __顺序表__。

2. 假设顺序表中包含 6 个数据元素{a，b，c，d，e，f},他们的查找概率分别为{0.12，0.25，0.23，0.2，0.05，0.15}，顺序查找时为了使查找成功的平均比较次数最少，则表中的数据元素的存放顺序应该是 b,c,d,f,a,e 。

3. 已知某棵完全二叉树中有 120 个结点，则该二叉树的结点一共有__7__层，有__60__个叶子结点。

4. 已知一棵树的中序遍历序列为 DBGEACF，后序遍历序列为 DGEBFCA，则这棵树的前序遍历结果为__ABDEGCF__。

5. 某段电文中只有 a,b,c,d 四种字符，各种字符出现的次数为：a 出现 1000 次，b 出现 2000 次，c 出现 6000 次，d 出现 1000 次，采用哈夫曼编码该电文的长度为__2000*8__个比特。

6. 一组记录的关键字为 45，80，55，40，42，85，利用堆排序的方法，从最后一个非叶子结点开始调整，建立的初始最大堆为__85 80 55 40 42 45__ 。

7. 设一棵 m 叉树中有 N_1 个度数为 1 的结点（度数表示子结点个数），N_2 个度数为 2 的结点，……，N_m 个度数为 m 的结点，则该 m 叉树中共有____$1+\sum_{i=2}^{m}(i-1)N_i$____个终端结点（即叶结点）。

8. 设散列表的表长 m=15，散列函数 H(key)=key%11，表中已有 4 个结点：addr(16)=5，addr(37)=4，addr(50)=6，addr(70)=7，如果用线性探查处理冲突，关键字为 38 的结点的地址是____8____ 。

9. 设森林 F 中有 4 棵树，第 1、2、3、4 棵树的结点个数分别为 10、9、11、7，当把森林 F 转换成一棵二叉树后，其根结点的右子树中有____27____个结点。

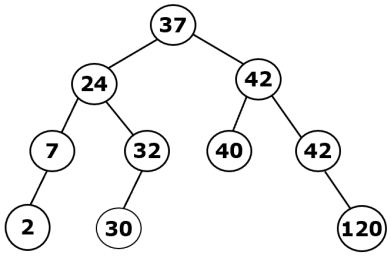
10. 一个无向图，如果边的数量 m ___>=___ （填写数量关系）结点个数 n，那么该图一定存在回路。

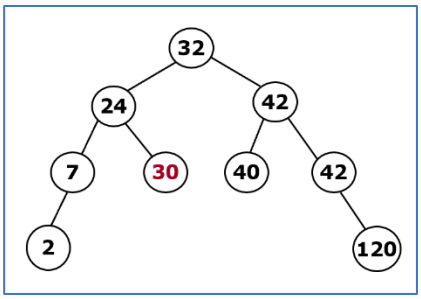
四. 简答（3 题，共 14 分）

1. 对序列{H, E, B, L, G, A, F, J, I, C, D, K}中的关键码按字母序的升序重新排列，则：
   a) 冒泡排序第一趟交换结束后的结果是？（1 分）
   b) 二路归并排序第一趟归并后的结果是？（1 分）
   c) 初始步长为 4 的希尔（shell）排序第一趟后的结果是 ？（2 分）

答：前两问送分，第三问需要熟悉 shell sort 思想。
1、两个给出一个就对。从左往右冒泡排序的结果是 {E, B, H, G, A, F, J, I, C, D, K, L}；从右往左冒泡排序的结果是 {A, H, E, B, L, G, C, F, J, I, D, K}。 ；
2、二路归并排序一趟扫描的结果是 (E H B L A G F J C I D K)；
3、初始步长为 4 的希尔（shell）排序一趟的结果是 (G A B J H C D K I E F L) ；

2. 在下列二叉排序树中删除结点“37”的基本过程是：首先寻找该结点左子树上的最大者 r，并利用 r 辅助删除该结点。请画出删除该结点后的二叉排序树结构。（3 分）





答案及解释：

基本答案如右图所示，但也可以有第二种方式

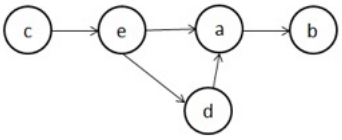
3. Dr. Stranger 的电脑感染了一种病毒。该病毒会将文档中的每种字母固定替换为另一种字母，且互不相同，可以看作文档所涉及字母集合上的双射函数。例如，文档 D = {ab, ac, bc}，涉及的字母集合为 X = {a, b, c}，病毒函数 f:X→X，假设其中 f(a) = b，f(b) = c，f(c) = a，那么感染病毒后的文档 D' = {bc, ba, ca}。现在 Dr. Stranger 有一个重要文档 D，内容为 6 个按字典序排列的单词，涉及的字母集合为 X = {a, b, c, d, e}。该文档感染了病毒后，内容变为 D' = {cebdbac, cac, ecd, dca, aba, bac}。请你破解出该病毒规则 f，并还原出原文档 D 的内容。给出思路（2 分）、具体步骤（3 分）、最终答案（2 分）。

答案及解释：

主要思路：就是推断病毒发作后文档 D 中的字母顺序，与字典序的字母顺序进行一一对应即可。每相邻的两个字符串进行比较；遇到不同的字母就建立边。

具体方法就是从相邻的两个字符串之间推断字母之间的顺序，如题目中单词 cac 在单词 ecd 之前，所以可以推断被替换后的字母顺序中字母 c 在字母 e 之前，将字母之间的顺序关系看成有向边，画出字母关系图，然后对图进行拓扑排序。得出的序列就是病毒发作后文档 D 中的字母顺序。

该题中可求得字母的顺序关系图如下：



通过拓扑排序，得到的病毒发作后文档 D 中的字母顺序{c，e，d，a，b}，而字典序为{a，b，c，d，e}，所以可以判断病毒将字母 a 替换为字母 c，将字母 b 替换为字母 e，将字母 c 替换为字母 d，将字母 d 替换为字母 a，将字母 e 替换为字母 b。

最终文档 D 的原内容如下：{abeceda，ada，bac，cad，ded，eda}。

五. 算法填空（4 题，共 26 分）

请注意：每个空最多填一条语句，不得使用逗号连接多个表达式的写法。

1. (6 分) 链表操作。

下面的程序描述的是一个只带表尾指针的循环单链表的操作(表尾指针指向链表最后一个结点，最后一个结点的 next 指针指向链表的第一个结点)。先是将 n 个整数依次插入链表头部，然后删掉链表尾部 m 个元素，再往链表前部插入 k 个整数。请填空。

输入：

第 1 行是整数 n(0 < n <= 100)

第 2 行是 n 个非负整数

第 3 行是整数 m(0 <m <= 200)

第 4 行是整数 k(0 < k <= 100)

第 5 行是 k 个非负整数

输出：

第 1 行：将链表中的元素依次输出（结果会是输入第 2 行的倒序）

第 2 行：将链表删除尾部 m 个元素后的结果输出。如果链表为空，输出 NULL

第 3 行：依次输出链表前部又插入输入中的第 5 行的 k 个整数后的结果

样例输入：

```
4
1 2 3 4
3
2
100 200
```

样例输出：

```
4 3 2 1
4
200 100 4
```

```
class Node:
    def __init__(self, data, next=None):
        self.data, self.next = data, next

class LinkedList:
    def __init__(self):
        self.tail = None  # 表尾指针，指向链表最后一个结点

    def pushFront(self, data):  # 在链表头部插入元素
        nd = Node(data)
        if self.tail is None:
            self.tail = nd
            nd.next = nd
        else:
            nd.next = self.tail.next  # 2 分
```

```
            self.tail.next = nd

    def popBack(self):  # 在链表尾部删除元素
        if self.tail is not None:
            if self.tail.next is self.tail:  # 1分
                self.tail = None
            else:
                ptr = self.tail.next
                while ptr.next != self.tail:  # 1分
                    ptr = ptr.next
                ptr.next = ptr.next.next
                self.tail = ptr  # 1分

    def print(self):
        if self.tail is not None:
            ptr = self.tail.next
            print(ptr.data, end=" ")
            ptr = ptr.next
            while ptr != self.tail.next:  # 1分
                print(ptr.data, end=" ")
                ptr = ptr.next
        else:
            print("NULL")
        print()

n = input()
a = list(map(int, input().split()))
Lst = LinkedList()
for x in a:
    Lst.pushFront(x)
Lst.print()
n = int(input())
for i in range(n):
    Lst.popBack()
Lst.print()
n = input()
b = list(map(int, input().split()))
for x in b:
    Lst.pushFront(x)
Lst.print()
```

2. (6 分) 求二叉树的宽度。

给定一棵二叉树，求该二叉树的宽度。二叉树宽度定义：结点最多的那一层的结点数目。

输入：第一行是一个整数 n，表示二叉树的结点个数。二叉树结点编号从 0 到 n-1， n <= 100。接下来有 n

行，依次对应二叉树的编号为 0,1,2....n-1 的结点。
每行有两个整数，分别表示该结点的左儿子和右儿子的编号。如果第一个（第二个）数为-1 则表示没有左（右）儿子
输出:输出 1 个整数，表示二叉树的宽度
样例输入

```
3
-1 -1
0 2
-1 -1
```

样例输出

```
2
```

```
class BinaryTree:
    def __init__(self, data, left=None, right=None):
        self.data, self.left, self.right = data, left, right

    def countWidth(self):
        width = [0 for i in range(200)]  # width[i]记录第 i 层宽度

        def traversal(root, level):
            if root is None:
                return
            width[level] += 1
            traversal(root.left, level + 1)  # 1 分
            traversal(root.right, level + 1)  # 1 分

        traversal(self, 0)
        return max(width)


def buildTree():
    n = int(input())
    nodes = [BinaryTree(None) for i in range(n)]
    for nd in range(n):
        L, R = map(int, input().split())
        if L != -1:
            nodes[nd].left = nodes[L]
            nodes[L].data = L  # 1 分
        if R != -1:
            nodes[nd].right = nodes[R]
            nodes[R].data = R  # 1 分
    for i in range(n):
        if nodes[i].data is None:  # 2 分
            return nodes[i]
    return None
```

```
tree = buildTree()
print(tree.countWidth())
```

3. (7 分) 用 Prim 算法求无向图最小生成树。

输入：第一行两个整数 n,m (n< 100)，表示图有 n 个结点，m 条边。结点编号从 0 开始算。接下来有 m 行，每行三个数 s e w, 表示边(s,e)的权值是 w(w<1000)。

输出：输出最小生成树的所有边

输入样例：

```
4 4
0 1 8.5
1 2 4
2 3 9
3 1 7
```

输出样例：

```
(1,0)(2,1)(3,1)
```

```
def prim(G):  # Prim 算法求图 G 最小生成树，返回最小生成树的边的列表
    INF = 1 << 30  # 无穷大
    # G 是邻接矩阵,矩阵中 None 表示没有边，顶点编号从 0 开始
    n = len(G)  # n 是顶点数目，顶点编号 0 – (n-1)
    dist = [INF for i in range(n)]  # 各顶点到已经建好的那部分树的距离
    used = [False for i in range(n)]  # 标记顶点是否已经被加入最小生成树
    prev = [None for i in range(n)]
    # 顶点 i 是通过边(i,prev[i])被加入最小生成树的
    doneNum = 0  # 已经被加入最小生成树的顶点数目
    edges = []  # 最小生成树的边的列表
    while doneNum < n:  # 1 分
        if doneNum == 0:
            x = minDist = 0  # 顶点 0 最先被加入最小生成树
        else:
            x, minDist = None, INF
            for i in range(n):
                if not used[i] and dist[i] < minDist:  # 1 分
                    x, minDist = i, dist[i]
        used[x] = True  # 1 分
        doneNum += 1
        if doneNum > 1:
            edges.append((x, prev[x]))  # 1 分
        for v in range(n):
            if not used[v] and G[x][v] is not None and G[x][v] < dist[v]:  #2 分
```

```
                dist[v] = G[x][v]
                prev[v] = x  # v 是通过 x 连接到最小生成树的    #1 分
    return edges


n, m = map(int, input().split())
G = [[None for i in range(n)] for j in range(n)]  # 邻接矩阵
for i in range(m):
    tmp = input().split()
    s, e, w = int(tmp[0]), int(tmp[1]), float(tmp[2])
    G[s][e] = G[e][s] = w
edges = prim(G)  # edges 的元素是一个元组(u,v)，表示一条边
for e in edges:
    print(f"({e[0]},{e[1]})", end="")
```

4. (7 分) 完成下列算法，计算一个无向图中所有连通分量的结点个数。
输入：图的结点数、边数和边的列表。
输出：每个连通分量的结点个数。

输入样例：
5
[(0, 1), (1, 2), (3, 4)]
输出样例：
[3, 2]

```
def dfs(node, visited, graph, n):
    count = 1
    visited[node] = True

    for i in range(n):
        if i in graph[node] and not visited[i]:  # (1) 1 分
            count += dfs(i, visited, graph, n)  # (2) 2 分
    return count

def count_components(n, edges):
    graph = {i: [] for i in range(n)}
    for u, v in edges:
        graph[u].append(v)  # (3)1 分
        graph[v].append(u)  # (4)1 分

    visited = [False] * n
    components = []
```

```
    for i in range(n):
        if not visited[i]: #(5) 1分
            components.append(dfs(i, visited, graph, n))  # (6)2分

    return components

n = int(input())
edges = eval(input())
print(count_components(n, edges))
```