# 北京邮电大学

# 2023 年硕士研究生招生考试试题

## 考试科目：809 数据结构

# 北京邮电大学

# 2023 年硕士研究生招生考试试题

## 考试科目：809 数据结构

**请考生注意：①所有答案(包括选择题和填空题)一律写在答题纸上，否则不计成绩。**

**②不允许使用计算器**

## 一、填空题（每空 1 分，共 35 分）

1. 数据结构包含四种逻辑结构，其中 _____适合应用于图书管理系统、手机通信录等；_____适合应用于对弈问题等；_____更适合应用于路由问题、导航问题等。

2. 如下所示的 C++代码，其中①的时间复杂度是_____，②的时间复杂度是_____。

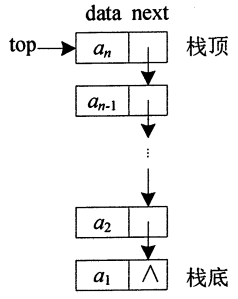
```
①int x=0;
②while (x*x <n) x++;
```

3. 单循环链表中，尾指针是 rear，每个结点有一个数据域 data 和指针域 next，则判断循环链表是非空的判定条件是_____。

4. 表长为 n 的顺序表，当在任何位置上插入或删除一个元素的概率相等时，插入一个元素所需移动元素的平均个数为_____，删除一个元素需要移动元素的平均个数为_____。

5. 设循环队列数组大小为 100，队头指针为 front，队尾指针为 rear；约定 front 指向队头元素的前一个位置，该位置永远不存放数据。则队满的判别条件为_____。若 front=20，rear=60，则队列长度为_____，若 front=60，rear=20，则队列长度为_____。

6. 已知栈的存储结构如下图所示，预设临时指针 p，可以指向任意一个结点，则出栈的指针操作是_____；_____；delete p；。




7. 设 9 行 10 列的二维数组起始元素为 A[0][0]，按行优先存储到起始元素下标为 0 的一维数组 B 中，则元素 B[35]对应于原二维数组 A 中_____元素。

8. 已知二叉树中叶子数为100，仅有一个孩子的结点数为51，则总结点数是_____。

9. 含有100个结点的完全二叉树，度为0，1，2的结点数量分别为_____、_____、____，树的高度为_____。

10. 100个结点若用三叉链表存储，则所有结点中的空指针共有_____个。

11.已知下列字符{A,B,C,D,E,F,G}的权值分别为{3,12,7,4,2,8,11}，若按照权值“左小右大”的方式合成结点，构建哈夫曼树；编码时左分支标0，右分支标1。请给出：(1)哈夫曼树的带权路径长度WPL=_____；(2)字符A的编码_____；(3)字符B的编码_____。

12. 有n个结点并且其高度为n的二叉树的形态数目是 _____。

13. 具有n个顶点的无向连通图，最多有_____边，由该图生成的生成树有 ___个顶点，____条边。

14. 由15个结点构成的二叉排序树，在等概率查找的条件下，计算查找成功时的平均查找长度，其最大值是_____，最小值是_____(第二个空小数点后保留1位数字)。

15. 在各种查找方法中，平均查找长度与结点个数无关的查找方法是_____。

16 .对于序列{25,43,62,31,48,1}，采用的散列函数为H(k)=k%7，则元素43的同义词是_____。

17. 对于键值序列{12,13,11,18,60,15,7,18,25,100}，用筛选法建堆，必须从键值为_____的结点开始进行调整。

18. 非比较排序中，_____排序是对桶排序的改进和扩展，长度为n的非比较排序的时间复杂度一般接近_____。

19.对序列{45,83,96,32,61,27,16,54,38}按升序进行快速排序， 若以“尾元素”为基准，一趟快速排序的结果为_____。

## 二、单选题（每小题 1 分，共 25 分）

1. 软件开发过程一般至少可以分成四个阶段，需求分析、系统设计、系统开发、系统维护，数据结构和算法是在（ ）阶段使用最多的知识和方法。

A. 需求分析　　B. 系统设计　　C. 系统开发　　D. 系统维护

2. 以下结构中，（ ）不属于线性结构。

A. 线性表　　B. 数组　　C. 串　　D. 二叉树

3.已知双链表L，长度n、头指针first，在其某个值为x的结点前插入一个结点的时间

复杂度是(　　)。

A. O(n)　　B. O(1)　　C. $O(\log_2 n)$　　D. $O(n^2)$

4. 以顺序表 a[]存储的线性结构，表长为 n，在该表中第 i 个位置插入元素 x 的核心代码应为（　　）。（备注：数组 a[]足够大，从 a[0]开始存储元素。）

A.
```
for(int j = n; j>i; j--)
    a[j] = a[j-1];
a[i] = x;
```

B.
```
for(int j = n; j>=i; j--)
    a[j] = a[j-1];
a[i-1] = x;
```

C.
```
for(int j = n; j>=i; j--)
    a[j] = a[j-1];
a[i] = x;
```

D.
```
for(int j = i; j<=n; j++)
    a[j] = a[j+1];
a[i-1] = x;
```

5. 将一个递归算法改为对应的非递归算法时，通常需要使用（　　）。

A. 数组　　B. 队列　　C. 二叉树　　D. 栈

6. 四个元素 1、2、3、4 依次进栈，出栈次序不可能出现（　　）情况。

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

7. 设push(k)表示整数k入栈，pop表示栈顶元素出栈。对于空栈S，在操作序列push(11), push(22), pop, push(12), push(25), pop, push(66)之后，栈顶元素、栈底元素及栈内元素数目分别是（　　）。

A. 66, 11, 3　　B. 22, 25, 2　　C. 66, 12, 5　　D. 22, 11, 2

8. 为解决计算机主机和键盘输入之间速度不匹配问题，设置键盘缓冲区，该缓冲区应该是一个（　　）结构。

A. 线性表　　B. 栈　　C. 队列　　D. 数组

9. 将一棵树 t 转换为二叉树 h，则 t 的后序遍历是 h 的（　　）。

A. 前序遍历　　B. 中序遍历　　C. 后序遍历　　D. 层序遍历

10. 设 F 是一个森林，B 是由 F 转换得到的二叉树，F 中有 n 个非叶结点，则 B 中右指针域为空的结点有(　　)个。

A. n-1　　B. n　　C. n+1　　D. n+2

11. 某二叉树的前序遍历结点访问顺序是 abcdefg，中序遍历的结点访问顺序是 abcdefg，则其后序遍历的结点访问顺序是(　　)。

A. bdgcefa　　B. gfedcba　　C. bdgaecf　　D. gdbefca

12. 高度为 8 的平衡二叉树的结点数至少是(　　)。

A. 20　　B. 33　　C. 54　　D. 62

13. 若从二叉树的任一结点出发到根的路径上所经过的结点序列按其关键字有序，则该二叉树是（　　）。

A. 二叉排序树　　B. 哈夫曼树　　C. 堆　　D. 完全二叉树

14.完全二叉树最简单的存储结构为（　　）。
A.二叉链表　　B.顺序结构　　C.线索链表　　D.三叉链表

15.以下说法错误的是（　　）。
A.由树转换成二叉树，其根结点的右子树总是空的。
B.存在这样的二叉树，对其采用任何次序的遍历其结点访问序列均相同。
C.连通网的最小生成树是唯一的。
D.若二叉排序树中的某个结点存在右子树，则该树的中序遍历序列中，该结点的后继结点没有左孩子。

16.若度为m的哈夫曼树中，其叶子结点个数为n，则非叶子结点的个数为（　　）。
A.n-1　　B. $\left\lfloor \frac{n}{m} \right\rfloor - 1$　　C. $\left\lceil \frac{n-1}{m-1} \right\rceil$　　D. $\left\lfloor \frac{n}{m-1} \right\rfloor - 1$　　E. $\left\lceil \frac{n+1}{m+1} \right\rceil - 1$

17. 图结构中结点之间的关系是(　　),即每个结点都可以有0个或多个前趋和后继。
A. 一对一的关系　　B. 一对多的关系
C. 多对多的关系　　D. 无关系

18. 如果某图的邻接矩阵是对角线元素均为零的上三角矩阵，则此图是（　　）。
A. 有向完全图　　B. 连通图　　C. 强连通图　　D. 有向无环图

19.设G是一个非连通图，共有28条边，则该图至少有（　　）个顶点。
A.7　　B.8　　C.9　　D.10

20.一个连通无向网的最小生成树可以使用（　　）生成。
A. Huffman算法　　B.Dijkstra算法　　C.Prim算法　　D. Floyd算法

21. 二叉排序树中，最大值结点的(　　)
A.左指针一定为空　　B.右指针一定为空
C.左、右指针均为空　　D.左右指针均不为空

22.一个长度为100的有序表进行折半查找，最多比较(　　)次就能找到表中的元素。
A.7　　B.8　　C.9　　D.10

23. 设H(x)是一个哈希函数，有K个不同的关键字（$x_1, x_2, \ldots, x_K$）满足$H(x_1) = H(x_2) = \cdots = H(x_K)$，若用线性探测法将这K个关键字存入哈希表中，至少要探测（　　）次。
A. K-1　　B.K　　C.K+1　　D.K(K-1)/2

24. 就平均性能而言，比较排序中最快的排序方法是(　　)。
A.冒泡排序　　B.希尔排序　　C.快速排序　　D.插入排序

25.设序列中有1000个元素，若只想得到其中前5个最大元素，则采用(　　)方法最合适。

A. 快速排序　　B. 归并排序　　C. 直接插入排序　　D. 堆排序

## 三、简答题（28 分）

1.（6 分，每空 2 分）已知 p 结点是某双向链表的中间结点，试从下面语句((1)～(18))中选择合适的语句序列，完成（1）～（3）要求的操作。

（1）在 p 结点后插入 s 结点的语句序列是_____。

（2）删除 p 结点的直接后继结点的语句序列是_____。

（3）删除 p 结点的直接前驱结点的语句序列是_____。

| | | | |
|---|---|---|---|
| (1) | p->next = p->next->next; | (10) | p->prior->next = p; |
| (2) | p->prior = p->prior->prior; | (11) | p->next->prior = p; |
| (3) | p->next = s; | (12) | s->next->prior = s; |
| (4) | p->prior = s; | (13) | s->prior->next = s; |
| (5) | p->prior->next = p->next; | (14) | p->next->prior = p->prior; |
| (6) | s->prior = p; | (15) | q=p->next; |
| (7) | s->next = p->next; | (16) | p=p->prior; |
| (8) | s->prior = p->prior; | (17) | delete p; |
| (9) | s->next = p; | (18) | delete q; |

2.（8 分） 设叶子结点的权值集合：W={0.17，0.05，0.01，0.04，0.06，0.20，0.33，0.12，0.02}，请完成以下任务：

（1）以 W 为权值集合，构造一棵 Huffman 树,要求所有分支结点的左孩子权值小于等于右孩子权值，请画出该树；（4 分）

（2）若规定 Huffman 树左支编码为 0，右支编码为 1，则最长编码和最短编码分别是什么；（2 分）

（3）计算其带权路径长度 WPL，小数点后保留 2 位小数。（2 分）

3.（6 分）设散列表长 12，按散列函数 H(x) = x % 11 计算散列地址，存储序列{41，13，23，1，32，20，10，54}，如果发生冲突，使用线性探测法处理冲突。

（1）填上散列表的存储结果。（4 分）

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | | | | | | | |

（2）若等概率查找序列中的每个元素，请计算上述方式查找成功的平均查找长度。（2 分）

4.（8 分，每空 2 分）设一组关键码序列为{65，98，56，35，42，84，17，25，71}，分别采用下列算法进行升序排序，请写出各算法的第一趟排序结果。

（1）简单选择排序_____；

（2）希尔排序（间隔 d=4）_____；

（3）二路归并排序_____；

（4）快速排序（第 1 个元素为轴值）_____。

## 四、综合题（62 分）

1.（12 分）对于如下稀疏矩阵：

（1）请写出对应的三元组顺序表；（5 分）

（2）若采用顺序取，直接存的算法进行转置运算，需要引入辅助数组 number[] 和 position[]，分别表示矩阵各列的非零元素个数和矩阵中各列第一个非零元素在转置矩阵中的位置，请写出这 2 个辅助数组的值；（2 分）

（3）请编程实现计算 number[]和 position[]。（所有数组起始元素下标为 0）（5 分，每空 1 分）

原矩阵：

| | | | |
|---|---|---|---|
| 0 | 0 | 2 | 0 |
| 0 | 0 | 0 | 0 |
| 7 | 0 | 0 | 5 |
| 0 | 0 | -4 | 0 |

| row | col | item |
|---|---|---|
| | | |
| | | |
| | | |
| | | |
| 矩阵行数: | | |
| 矩阵列数: | | |
| 非零元个数: | | |

| Col | 0 | 1 | 2 | 3 |
|---|---|---|---|---|
| number[col] | | | | |
| position[col] | | | | |

已知：

```
const int N=128;
struct MatrixNode //定义三元组结构
{
    int row; //非零元素的行号
    int col;  //非零元素的列号
    int value;//非零元素的值
};
MatrixNode data[N];   //三元组
```

i. 计算 number[]，已知原矩阵 n 行 m 列 t 个元素。

```
int *number = new int[m];
memset(number,0, sizeof(int)*m);   //置 0
_____;
_____;
```

ii. 计算 position

```
int *position= new int[m];
```

_____;
_____;
_____;

2. （10 分）已知序列：{ 8, 6, 2, 4, 12, 10, 5, 16, 11}
（1）画出该序列对应的二叉排序树；（3 分）
（2）若基于该二叉排序树进行等概率查找，计算查找成功的平均查找长度；（2 分）
（3）用 C++代码补全下面的算法，完成判定一棵二叉树是否为排序二叉树的代码实现。（5 分，每空 1 分）

```
struct BiNode
{
    int data;
    BiNode* lch,*rch;
    BiNode():lch(NULL),rch(NULL){}   //构造函数
};
bool IsBST(BiNode* R)   //pre已知，为记录当前结点的前驱结点值，初值为-∞
{
    if (_____)   return true;
    else
    {
        bool b = _____;
        if (_____)       return false;
        pre = R->data;
        b = _____;
        _____;
    }
}
```

3. （14 分）已知带权无向图 G 的邻接矩阵如下，完成以下问题。
（1）结点编号按照从上到下的顺序依次为 0~5，请根据邻接矩阵画出图 G；（3 分）
（2）画出该图的最小生成树；（2 分）
（3）分别写出从 0 号结点开始深度和广度优先遍历结果；（2 分）
（4）补全下面得 C++代码实现广度优先遍历算法。（7 分，每空 1 分）

$$\begin{Bmatrix} 0 & 6 & 1 & 5 & \infty & \infty \\ 6 & 0 & 5 & \infty & 3 & \infty \\ 1 & 5 & 0 & 5 & 6 & 4 \\ 5 & \infty & 5 & 0 & \infty & 2 \\ \infty & 3 & 6 & \infty & 0 & 7 \\ \infty & \infty & 4 & 2 & 7 & 0 \end{Bmatrix}$$

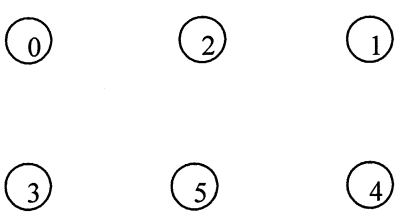



图 G（左）邻接矩阵 arc[N][N]（N 为结点总数）　　（右）画图结点布局

```
const MAX = 9999;       //这里表示无穷大
const int N = 6;
int visited[N]={0};     //表示结点是否被访问过，0 为未访问，1 已访问。
void BFS (int v)        //从 v 出发广度优先
{
    int queue[N];       //定义队列
    int f, r;           //定义队头和队尾指针
    _____;  //队列初始化
    if(visited[v]==1) return;  //非连通图的处理

    _____;    //结点 v 入队
    visited[v] = 1;
    while( _____)//队列不空
    {   v= _____;
        cout<<v;
        for (int i=0;i<N;i++)   // N 为结点总数
            if (_____) //找未访问过的邻接点
            {
                _____;
                _____;
            }
    }
}
```

4. （11 分）已知序列{12，47，10，18，60，15，7，13，25，100}

（1）使用最少的调整次数将其调整为大根堆(画图)，写出调整后的序列；（3 分）

（2）补全下面的代码，完成建大根堆的算法，包括筛选算法和调用筛选算法完成建大根堆的 main()函数。（8 分，每空 1 分）

```
void Sift (int r[], int k, int m)   //k 是被“筛选”结点的编号，m 是最后一个结点的编号
{
    int i=k;
    int j=_____;            //i 是要筛选的结点，j 是 i 的左孩子
    while (_____)           //j 存在
    {
        if ( _____)  j++;    //j 是左右孩子中较大者
        if (r[i]>r[j])
            _____;                  //符合大根堆的条件，结束
        else
        {   int t = r[i];   r[i] = r[j] ;  r[j] = t;     //根结点与孩子结点交换
            _____; //迭代
            _____;
        }
    }
}
```

```
int main()
{
    int r[]= {0, 12，47，10，18，60，15，7，13，25，100}; //下标 0 号的元素不用
    int n = 10;
    for (_____) //从最后一个分支结点开始建堆
        _____;
    for (int i=1;i<=n;i++) //输出大根堆序列
        cout<<r[i]<<endl;
}
```

5.（15 分）.以邻接表为存储结构实现从源点到其余各顶点最短路径的 Dijkstra 算法，邻接表的弧结点、顶点的结构定义分别为 ArcNode 和 VertexNode，基于邻接表的图类定义为 ALGraph；请完成下列代码。

```
struct ArcNode
{   int adjvex; //边的终点在顶点表中下标
    int weight; //网的边或弧之权值
    ArcNode* next; //指向边表中下一个结点
};
struct VertexNode
{   int vertex;   //数据域
    ArcNode* firstArc; //指向边表第一个结点
};
const int INF_VALUE = 0x3f3f3f3f; //定义无穷大值
const int MAXSIZE = 5;       //图的最大顶点数

class ALGraph
{
public:
    ALGraph(int a[],   int n, int e);
    ~ALGraph();
    void DijkstraBasedOnAdjList( int v);
private:
    VertexNode adjList[MAXSIZE]; //邻接表
    int vertexNum, arcNum; //顶点数、边数
};
// 基于邻接表的Dijkstra算法
// 参数：srcVex--源点在顶点表中下标
void ALGraph::DijkstraBasedOnAdjList(int srcVex)
{
    int dist[MAXSIZE] = {0};   //源点到各个顶点距离数组
    int path[MAXSIZE] = {0}; //源点到各个顶点路径数组
    int s[MAXSIZE] = {0};//顶点是否找到最短路径的标志数组，0未找到，1已找到;
```

```
 int i, j, k, pre;

 // 初始化距离与路径值
 for ( i = 0; i < vertexNum; i++) // 首先初始化dist[i]为无穷；
 {   dist[i] = INF_VALUE;
     path[i] = -1;            //表示在最短路径中，结点i的前驱结点还未找到
 }

 ArcNode* p = adjList[srcVex].firstArc;
 while (p) //根据邻接表修正距离和路径
 {   dist[p->adjvex] = _____;
     path[p->adjvex] =_____;
     p =_____;
 }
 // 设置源点相关信息
 s[srcVex] = 1;
 dist[srcVex] = _____;
 path[srcVex] = _____;   //源点无前驱结点

 // 寻找单源最短路径
 int min_dist = INF_VALUE;
 for (i = 0; i < vertexNum - 1; i++)   //扩充s集
 {    min_dist = INF_VALUE;
      for (j = 0; j < vertexNum; j++)   // 找最短路径的顶点k
      {
           if (_____&&_____   )
           {
               min_dist =_____;
               k = ____;
           }
      }
      s[k] = 1; //将k加入s集
      p = _____;
      while (p )//调整未找到最短路径的各顶点的距离值
      {
           j = p->adjvex;
           int d = dist[k] + p->weight;
           if (_____)
           {   dist[j] =  _____;
               path[j] =  _____;
           }
           p = _____;
      }
}
```

```
    //所有顶点均已扩充到s集中，打印结果；
    for (i = 0; i < vertexNum; i++)
    {
        cout << dist[i]<<", " << i;
        pre = path[i];
        while (_____)//继续找路径上前驱顶点
        {   cout << "<-" << pre;
            pre = path[pre];
        }
        cout << endl;
    }
}
```