2018 北航991 真题参考答案

一、

1.D　　2.B　　3.C　　4.D　　5.D

6.A　　7.C　　8.n*(n-1)/2　　9.D　　10.D

二、

1.

选择链式存储结构，若多个线性表需要长度动态变化，意味着要多次进行插入和删除，那么采用链式存储结构效率更高。

2.

第 7 层有 10 个叶结点有两种情况：

第一种：第 7 层为完全二叉树最后一层，第 7 层 10 个叶结点是在第 7 层从左到右排列开来的，这种情况完全二叉树结点最少，按照公式推导，总结点数为：

2^6-1+10=73（前 6 层总结点数加上第 7 层 10 个叶结点）；

第二种：第 7 层为完全二叉树倒数第 2 层，第 7 层 10 个叶结点为第 7 层最右侧 10 个节点，因其没有孩子节点形成，此时第 7 层有孩子节点的结点个数为：

2^{7-1}-10=54，若这 54 个结点在第 8 层均有左右孩子结点，此时结点总数最多，故第 8 层应有 54*2=108 个结点，按照公式总结点数为：2^7-1+108=235（前 7 层总结点数加上第 8 层 108 个叶结点）。

3.

（1）判断邻接矩阵中非零元素总个数，用其除以二即无向图边的数目；

（2）设该节点为第 i 个结点，邻接矩阵中数第 i 行(第 i 列也可以)非零元素的个数即该顶点的度；

（3）判断邻接矩阵的第 i 行第 j 列（或者第 j 行第 i 列）的元素是否非零，如果非零即两个顶点之间有边，否则，就没有边。

4.

(1)使用折半插入排序所要进行的关键字比较次数，与待排序元素的初始状态无关。无论待排序序列是否有序，已形成的部分子序列是有序的。折半插入排序首先查找插入位置，

插入位置是判定树失败的位置（对应外部结点），失败位置只能在判定树的最下面两层上。

(2)待排序序列正序的情况下，折半插入排序比直接插入排序需要更多的关键字比较。

三.

1.实现双向链表中两个相邻结点位置的调换。

2.

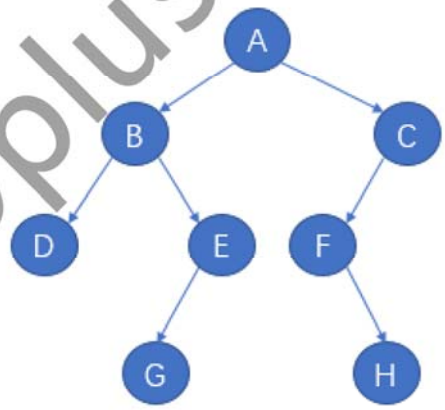
至少有三个元素空间。

原因：进栈顺序从 a1 到 a6，通过观察出栈顺序发现，a1 第一个进栈却最后一个出栈，因此在整个过程中，a1 从进栈后一直在栈中，需要一个固定的空间存放 a1；通过观察出栈顺序前三位发现是连续且与进栈顺序相同，因此至少需要一个空间，使得这三个元素进栈后就出栈，直到 a4 出栈完毕此位置空缺；出栈顺序第 4、5 位是 a6 和 a5，与进栈顺序相反，因此说明，a5 进栈后未出栈，a6 又进栈、出栈后 a5 出栈，原来 a4 剩下的一个空间被 a5 占用，再需要一个空间容纳 a6，最后 a1 出栈。故总共需要三个元素空间。

（图示可根据以下进栈顺序自行画出即可：a1 push、a2 push、a2 pop、a3 push、a3 pop、a4 push、a4 pop、a5 push、a6 push、a6 pop、a5 pop、a1 pop）

3.

已知二叉树采用顺序存储结构，因此恢复该二叉树：

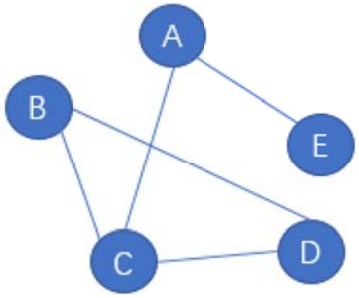


前序遍历序列：ABDEGCFH

后序遍历序列：DGEBHFCA

4.

已知无向图邻接表表示方法，恢复原无向图：



深度优先遍历序列为：ACBDE

广度优先遍历序列为：ACEBD

四.

```
int INSERT_BINSEARCH(ElemType a[], int &n, ElemType item)
{
    int low=0, high=n-1, mid, i;
    if(n==MaxSize || i<1 || i>n+1)
        return -1;
    while(low<=high)
    {
        mid=(low+high)/2;
        if(a[mid]<=item)
            low=mid+1;
        else
            high=mid-1;
    }
    for(i=n-1;i>=mid;i--)
        a[i+1]=a[i];
    a[mid]=itcm;
    n++;
    return 1;
}
```

五.

1.A　　2.B　　3.C　　4.B　　5.D

6.B　　7. C　　8.A　　9.D　　10.B

六.

1.

```
int i = 0,j = 100,count = 0;
while(i < 100){
    while(j >= i){
```

```
            count += j-i;
            j = j-2;
        }
        i = i+1;
}
```

2.

char *msg1 声明的是字符类型的指针变量，指向字符类型变量，指针变量存储单元内存储其指向的字符类型变量地址，这种声明字符串的方式使得指针变量 msg1 指向该字符串常量的首地址，即存储第一个字符变量地址。

char msg2[20]声明了一个长度为 20 的字符数组，msg2 是数组名同时也代表数组首地址。使用时需要注意 msg1 是指针变量，可以移动位置（在此字符串常量内进行加减数字移动对应长度位移量）去寻找字符串中某一字符执行操作，移动之前需要另一个指针记录字符串首元素地址，以便后续操作；msg2 是数组名，虽然也可以表示数组首元素地址，但属于常量指针，即固定值，同样可以在数组名基础上进行加减数字去定位数组中某元素，但不可对 msg2 进行自加或者自减以及其他修改数组名本身值的运算。

3. ①不同点：

fread 是用二进制方式向文件读入一组数据，将磁盘文件中若干字节的内容一批读入内存，参数有四个，分别为存储区地址，读写字节数，数据项个数以及文件指针；

fscanf 是用格式化的方式向终端进行格式化的输入，参数有三个，分别为文件指针，格式字符串，以及输入表列。

②适用情况：

fread 适用于需要内存与磁盘进行频繁交换数据的情况，用二进制方式读取数据；

fscanf 适用于向终端进行输入，不适用于内存与磁盘频繁交换数据的情况。

4.1248@

七.

```
int freqs(char *source, char *destination){
        char *p, *q;
        int num=0;
        while(*source != '\0'){
                for(p=source, q=destination; *p!='\0' && *q!='\0' && *p==*q; p++,q++);
                if(*q=='\0'){
                        num++;
                }
                source++;
```

```
        }
        return num;
}
```

八.

```
#include<stdio.h>
#include<string.h>
//第一行内容合法性检测
int Rule1(char *s){
        if(*s != '@'){
                return 0;
        }else{
                return 1;
        }
}
//第二行内容合法性检测
int Rule2(char *s, int len){
        int i;
        for(i = 0; i < len; i++){
                if(s[i] != 'A' &&s[i] != 'T' && s[i] != 'G' && s[i] != 'C'&&s[i] != 't' && s[i] != 'g'
                && s[i] != 'c'){
                        return 0;
                }
        }
        return 1;
}
//第三行内容合法性检测
int Rule3(char *s){
        if(*s != '+'){
                return 0;
        }else{
                return 1;
        }
}
//检查第四行的内容合法性
int Rule4(char *s, int len){
        int i;
        for(i = 0; i < len; i++){
                if(s[i] < '!' || s[i] > '~'){
                        return 0;
                }
        }
        return 1;
}
//检查第二行与第四行是否长度相同
```

```
int Rule5(int len1, int len2){
	if(len1 != len2){
		return 0;
	}
	return 1;
}
int main(int argc, char **argv){
	if(argc == 1){
		printf("请输入文件名");
	}else{
		printf("正在处理文件%s\n", argv[1]);
	}
	int tindex = 0, len1 = 0, len2 = 0, index = 1; //定义生物数量计数器，第二行内容长度，
	第四行内容长度，每一个生物内容行数计数
	char content[101];
	FILE *fp;
	if((fp = fopen(argv[1], "r")) == NULL){
		printf("文件不存在\n");
		return 0;
	}else{
		int flag = 0; //读完一行的标识符
		while(!feof(fp)){
			fgets(content, 101, fp);
			flag++;
			if(index == 1 && flag == 1){
				int result1 = Rule1(&content[0]);
				if(!result1){
					printf("第%d 个生物的第一行内容错误", tindex+1);
					return 0;
				}
			}
			if(index == 2){
				int tmp_len = 0;
				int j;
				for(j = 0; content[j] != '\0' && content[j] != '\n'; j++){
					len1 += 1;
					tmp_len++;
				}
				int result2 = Rule2(content, tmp_len);
				if(!result2){
					printf("第%d 个生物的第二行内容错误", tindex+1);
					return 0;
				}
			}
```

```
        if(index == 3 && flag == 1){
            int result3 = Rule3(&content[0]);
            if(!result3){
                printf("第%d 个生物的第三行内容错误", tindex+1);
                return 0;
            }
        }
        if(index == 4){
            int tmp_len = 0;
            int j;
            for(j = 0; content[j] != '\0' && content[j] != '\n'; j++){
                len2 += 1;
                tmp_len++;
            }
            int result4 = Rule4(content, tmp_len);
            if(!result4){
                printf("第%d 个生物的第二行内容错误", tindex+1);
                return 0;
            }
        }
        //判断是否读完一行
        if(strlen(content) != 100){
            //判断是否读完一个生物的内容
            if(index == 4){
                //判断第二行，第四行内容长度是否相同
                int result5 = Rule5(len1, len2);
                if(!result5){
                    printf("第%d 个生物的第二行,第四行长度不一致",
                        tindex+1);
                    return 0;
                }
                index = 1;
                tindex += 1;
                len1 = 0;
                len2 = 0;
            }else{
                index++;
            }
            flag = 0;
        }

    }
    printf("检测通过");
}
```

```
}
```