数据结构考试题（样例三）

一．判断下列各命题是否正确，若正确在（　）内打✓，否则打×。（5分）

1. 使用双向链表存储数据，可以提高查找（定位运算）的速度。（✓）
2. B+ 树是一种特殊的二叉树。（×）
3. 栈可视为一种特殊的线性表。（✓）
4. 最小生成树是边数最少的生成树。（✓）
5. 快速排序总是比其它排序方法快。（×）

二．选则题（10分）

1. 数据结构中，与所使用的计算机无关的是数据的 （3） 结构。

（1）存储　（2）物理　（3）逻辑　（4）物理和存储

2. （4）在链表中进行比在顺序表中进行效率高。

（1）顺序查找　（2）折半查找　（3）分块查找　（4）插入

3. 树结构最适合于用来表示 （3）。

（1）有序数据　（2）无序数据　（3）元素间具有分支层次关系的数据

（4）元素间无关联的数据

4. 散列存储中，碰撞（或称冲突）指的是 （2）。

（1）两个元素具有相同序号　（2）不同的关键字对应于相同的存储地址

（3）两个记录的关键字相同　（4）数据元素过多

5. 直接选择排序的时间复杂性为 （4）（n为元素的个数）。

（1）O(n)　（2）O(log2 n)　（3）O(n log2 n)　（4）O(n2)

三．填空（10分）

1. 抽象数据类型定义由 形式化定义、操作、我我我 部份组成。
2. 循环队列为满的条件是 Q.front = (Q.rear+1) MOD maxsize -1，为空的条件是 Q.front = Q.rear。
3. 设 s = ‘I AM A STUDENT’，t = ‘GOOD WORKER’，CONCAT (Substr (s，6，2)，t) = ‘A　GOOD　WORKER’。
4. 假设二维数组A[1 . . 20，1 . . 10] 按行序优先存储，每个元素占4个单元，A[1，1]的地址是2000，则A[15，10]的地址是 2596 。
5. 对广义表LS = (a, ((b, c), ( ), d), (((e))))分别进行HEAD和TAIL运算，其结果为 HEAD（LS）= （a），TAIL（LS）= (((b, c), ( ), d), (((e)))) 。

6. 深度为k的满二叉树有 $2^k - 1$ 个结点。

7. n个结点的连通图至少有 n - 1 条边。

8. 散列既是一种 存储 方式，又是一种查找方法。

9. 文件的逻辑记录和物理记录的区别是 。

10. ISAM文件由 主索引，柱面索引，磁道索引和主文件组成。

四. 应用题（45 分）

1.（15 分）试编写算法实现：①求二叉的叶子数；②先根遍历二叉树;

答案:

```
struct    btreenode
{     char data;     //数据域
      btreenode *lchild, *rchild;       //左右孩子指针域
}
void    countleaf (btreenode *t, int &count)
{       //先根遍历根指针为 t 的二叉树，以计算其叶子数 count; 假定 count 初值为 0
      if (t != NULL)
      {     if ((t→lchild = = NULL) && (t→rchild = =NULL))
            count++;         //若 t 所指结点是叶子，计数器加 1
            countleaf (t→lchild, count);     //累计左子树上的叶子树
            countleaf (t→rchild, count);     //累计右子树上的叶子树
      }
}
void    preorder (btreenode *t)     //先根遍历指针为 t 的二叉树
{     btreenode *stack [stacksize];         //工作栈
      btreenode *s;
      int top = 0;
      stack[top] = t;       //根指针进栈
      while (top >= 0)
      {     s = stack[top];       //读栈顶元素到 s 中
            top--;       //退栈
            while (s != NULL)
            {     visit(s);
                  top++;
                  stack[top] = s→rchild;          //右指针进栈保存
                  s = s→lchild       //修改 s 以便继续遍历左子树
            }
      }
}
```

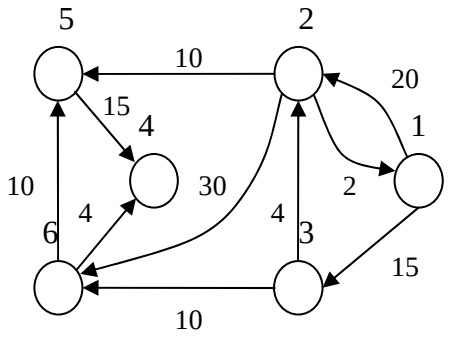
2.（15 分）试编写算法构造一棵哈夫曼树及其哈夫曼编码。

答案: 哈夫曼编码译电算法思想: 假设待译的二进制电文为x_1、x_2…x_m，x_i是‘0’或‘1’。

从已生成的哈夫曼树的根结点出发，按逐个读入的x_i走一条到某个叶子的路经。若x_i是‘0’，则向左走，否则向右走到下一层的结点。一旦到达叶子结点便译出一个相应的字符，然后重新从根出发继续译码直至二进电文结束。

```
void    huffmancode (btreenode *t)
{ //t为指向二叉树根对点的指针，叶子结点的data域中存放了对应的字符，二进制电文x1、x2…xm由函数read( )依次读入
      btreenode *p;
      int k = 0, a;       //以 k 计读入的二进制码的数目
      while (k < m)
      {     p = t;       //p 为指向结点的指针，其初值为指向根结点
            while ((p→lchild != NULL) && (p→rchild != NULL) && (k < m))
            {     read(a);     //读入一个二进制码
                  if (a = = 0)    p = p→lchild;
                  else    p = p→rhcild;
            }
           if ((k == m) && (p→lchild != NULL))
            {     cerr<<”电文有错！”<<endl;   //电文读完，但未到叶子，则电文有错
                  exit(1);
            }
            else    cout<<p→data;
      }
}
```

3.（15 分）对于下面给出的有向图，求从顶点 1 到其余各顶点的最短距离。



答案：要求写出邻接矩阵和下面各条路径的求解过程

1）顶点①→顶点②的最短路径长度：19（①→②，①→③→②）；

2）顶点①→顶点③的最短路径长度：15（①→③）；

3）顶点①→顶点④的最短路径长度：29（①→③→⑥→④，①→③→⑥→⑤→④，①→③→②→⑤→④，①→③→②→⑥→④，①→②→⑥→④，①→②→⑤→④）；

4）顶点①→顶点⑤的最短路径长度：29（①→②→⑤，①→③→②→⑤，①→③→②

→⑥→⑤，①→②→⑥→⑤，①→③→⑥→⑤);

5）顶点①→顶点⑥的最短路径长度：25（①→③→⑥，①→②→⑥，①→③→②→⑥);

五．设计题（30 分）

1.（20 分）试编写插入排序算法实现：①直接插入排序；②折半折入排序；③希字排序。

答案：假定待排序的数据存放在如下定义的存储结构上：

```
struck    records
{     int key;
      othertype otheritem;       //另外定义的数据类型
};
const int n =   · · · ;
```

其中，key 是排序时的关键字，实际上其类型可以为整型、实型或者字符串等。n 为待排序的记录的数目

```
void    straightsort (records a[n+1])
{     int i, j;
      for (i = 2; i <= n; i++)       //从第二个记录起进行插入
      {     a[0] = a[i];
            j = i -1;
            while (a[0].key < a[j].key)
            {     a[j+1] = a[j];
                  j--;
            } //将第 j 个记录赋值给第 j+1 个记录，直至关键字不大于待插入记录的关键字为止
            a[j+1] = a[0];         //将第 i 个记录插入
      }
}
```

上述算法是稳定的，时间复杂性为$O(n^2)$。从空间来看，它只需要一个记录的辅助空间。

假设初始数据文件存放在 a[1]至 a[n]中，low 和 high 分别指向前端和后端，mid 指向当前一次折半的中心位置，x 暂存新插入的记录，折半插入算法如下：

```
void    binary_insertion (records a[n+1])
{     int i, low, high, mid;
      records x;
      for (i = 2; i <= n; i++)
      {     x = a[i];
            low = 1;
            high = i -1;
            while (low <= high)
            {     mid = (low + high) / 2;     //折半
                  if (x.key < a[mid].key)    high = mid - 1;
                  else    low = mid+1;
            }
```

```
            for (j = i -1; j >= low; j--)       //后移记录
               a[j+1] = a[j];
            a[low] = x;
      }
}
```

希尔排序又称缩小增量排序。它是将待分类文件的原始文件分成若干个子文件。例如，当 k=5 时，第一个子文件包含的元素是 a[1], a[6], a[11]，…。而第二个子文件包含的元素是 a[2], a[7], a[12]，…。原文件分成 5 个子文件。先将这 k 个子文件进行排序（通常用直接插入进行），然后再选择较小的 k（例如令 k=3）,又分原文件为 k(=3)个子文件，再对这些子文件进行排序。如此重复下去，直到 k=1 时,子文件变成了一个排序文件，排序完成。增量 k 由始至终组成一个逆减序列，终止为 k=1。

```
void    shellsort (records a[n+1], int *h)
{     int m = 0, j, k, step;
      records x;
      bool found;
      while (h[m] > 1)
      {     step = h[m];     //增量
            for (j = step + 1; j <= n; j++)       //a[j]插入子文件适当位置
            {     x = a[j];
                  k = j - step;
                  found = false;
                  while ((k >= 1) && (! found))
                  if (x.key < a[k].key)
                  {     a[k+step] = a[k];
                        k = k - step;
                  }
                  else    found = true;
                  a[k+step] = x ;
            }
      }
}
```

希字排序法占用的额外空间数量较小，但其比较次数与“增量”的设置有密切的关系。实际推出，当n在某个特定范围内，比较移动次数均为$n^{1.2}$，当$n \to \infty$时，可减少到$n(\log_2 n)^2$。

2.（10 分）如何理解抽象数据型?，逻辑结构与存储结构是什么关系?。试说明线性数据结构和非线性数据结构在逻辑结构、存储结构、基本运算等三个方面的异同。

答案：请参考教科书。