2019 华中科技大学 887 数据结构与算法分析

【皮皮版】 卷一

仅供测试之用

一、名词解释（25 分）

1.渐近时间复杂度

【解析】如果一个问题的规模是 n，解这一问题的某一算法所需要的时间为 T(n)，它是 n 的某一函数。T(n)称为这一算法的“时间复杂度”。当输入量 n 逐渐加大时，时间复杂度的极限情形称为算法的“渐近时间复杂度”。

2. 就地算法

【解析】就地（In-place）算法指的是直接修改输入数据而不是将输入数据复制一份处理之后再覆盖回去，这个名称和时间复杂度没什么关系，纯粹是指算法处理数据的方式。比如一个冒泡排序就是就地算法。

3.DAG

【解析】如果一个有向图无法从某个顶点出发经过若干条边回到该点，则这个图是一个有向无环图（DAG 图）。

4. Complete Graph

【解析】在图中，每两个顶点之间都存在边，则称为完全图。

5. 同义词

【解析】散列函数可能会把两个或以上的不同关键字映射到同一地址，这种情况为冲突。（同义词）

二、选择题（25 分）

2.1

```
int frog {
    if(n==0)
      return n/2;
   else  return （n%2+frog（n-1）/2）;
 }
```

上述算法时间复杂度是多少（ B ）

A. logn　　B.n　　C. nlogn　　D. (n)`2

2.2 对于逆波兰式 0！1+23！4+^*56!7*8!?/-11+的值等于 2019，则？处的运算符为(D)。

A. 加号　　B. 减号　　C. 乘号　　D. 除号　　E. 乘方　　F. 阶乘

2.3 现有字符串 s 为”aabaabaabaac”，模式串 t 为”aabaac”，那么采用 KMP 算法，在第(A)趟匹配时串 t 在串 s 中匹配成功。

A. 3　　B. 4　　C. 6　　D. 7

2.4 设一棵 m 叉树中度数为 0 的结点数为 N0，度数为 1 的结点数为 N1，……，度数为 m 的结点数为 Nm，则 N0=(B)。

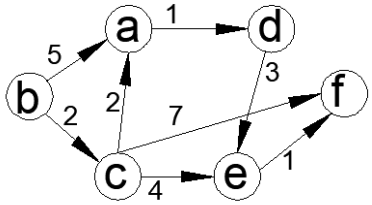
A. N1+N2+……+Nm

B. 1+N2+2N3+3N4+……+(m-1)Nm

C. N2+2N3+3N4+……+(m-1)Nm

D. 2N1+3N2+……+(m+1)Nm

2.5 使用 Dijkstra 算法计算有如下有向带权图从源节点 b 到达各节点的最短路径则求得最短路径的节点（b 除外）依次是(A)。



A. c, a, d, e, f　　B. c, a, e, f, d　　C. c, e, a, d, f　　D. a, c, e, f, d

三、应用题（60 分）

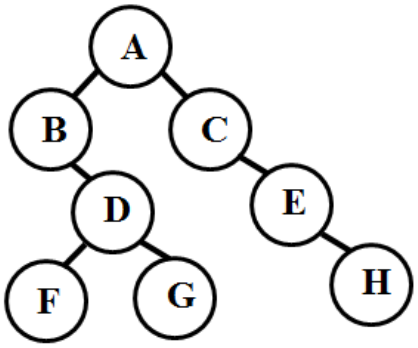
3.1 给出一个顺序存储的二叉树，ABC0D0E00FG0H,0 为空

（1）画出这个二叉树

（2）求前中后序结果

【解析】

（1）



（2）

前序：ABDFGCEH

中序：BFDGACEH

后序：FGDBHECA

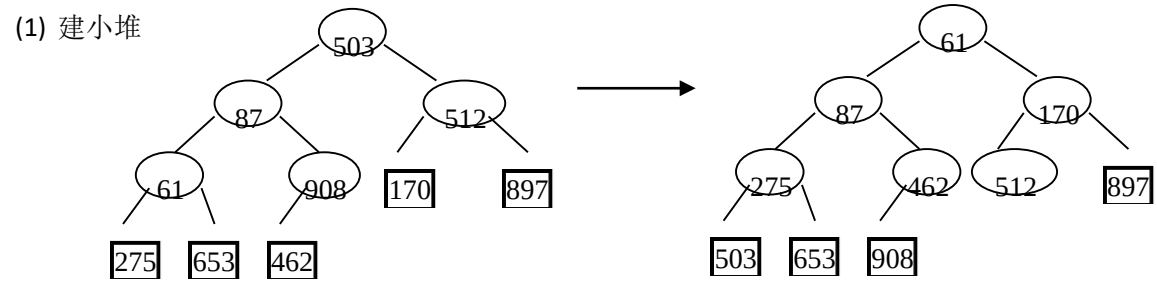
3.2 已知待排序的序列为（503，87，512，61，908，170，897，275，653，462），试完成下列各题。

(1) 根据以上序列建立一个堆（画出第一步和最后堆的结果图），希望先输出最小值。

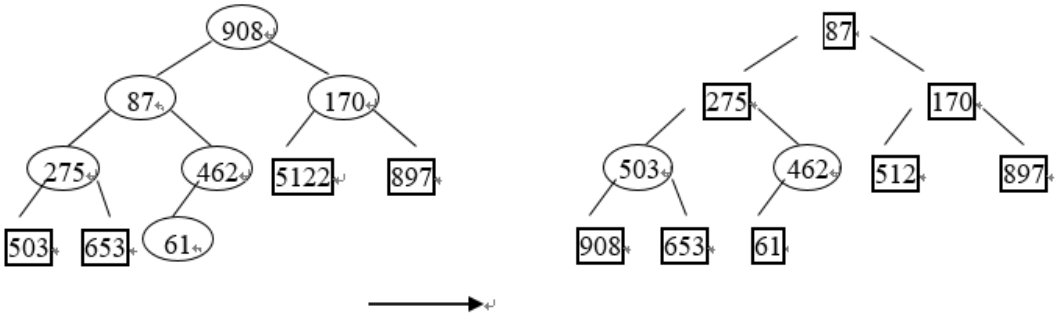
(2) 输出最小值后，如何得到次小值。（并画出相应结果图）

【解析】

(1) 建小堆



(2) 求次小值



3.3 对于下列七种排序算法：冒泡排序、插入排序、选择排序、基数排序、堆排序、归并排序、快速排序。请给出：

（1）在最佳情况下，哪些算法是 O(n^2),哪些算法是 O(n log2n) ，哪些算法是 O(n)?

（2）在最差情况下，哪些算法是 O(n^2),哪些算法是 O(n log2n) ，哪些算法是 O(n)?

（3）在平均情况下，哪些算法是 O(n^2),哪些算法是 O(n log2n) ，哪些算法是 O(n)?

【解析】

（1）在最佳情况下，选择排序是 O(n^2)；快速排序、堆排序、归并排序是 O（nlog2n）；插入排序、起泡排序是 O(n)。

（2）在最差情况下，快速排序、插入排序、起泡排序、选择排序是 O(n^2)；堆排序、归并排序是 O（nlog2n）；没有一种排序是 O(n)。

（3）在平均情况下，插入排序、起泡排序、选择排序是 O(n^2)；快速排序、堆排序、归并排序是 O（nlog2n）；没有一种排序是 O(n)。

3.4 全源最短路径问题采用 Floyd 算法进行求解。下面给出了一个由 4 个顶点构成的有向图邻接矩阵 Dist[4][4]和路径矩阵 Path[4][4]。约定 Dist 中用∞表示不能到达，Path 中用-1 表示没有前驱的情况。请计算并给出每一次迭代的结果。

	$Dist^{(-1)}$				$Dist^{(0)}$				$Dist^{(1)}$				$Dist^{(2)}$				$Dist^{(3)}$			
	0	1	2	3	0	1	2	3	0	1	2	3	0	1	2	3	0	1	2	3
0	0	1	4	∞																
1	∞	0	2	5																
2	∞	∞	0	1																
3	2	∞	∞	0																

	$Path^{(-1)}$				$Path^{(0)}$				$Path^{(1)}$				$Path^{(2)}$				$Path^{(3)}$			
	0	1	2	3	0	1	2	3	0	1	2	3	0	1	2	3	0	1	2	3
0	-1	0	0	-1																
1	-1	-1	1	1																
2	-1	-1	-1	2																
3	3	-1	-1	-1																

【解析】

Dist (-1)

	0	1	2	3
0	0	1	4	INF
1	INF	0	2	5
2	INF	INF	0	1
3	2	INF	INF	0

Path (-1)

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

Dist (0)

	0	1	2	3
0	0	1	4	INF
1	INF	0	2	5
2	INF	INF	0	1
3	2	3	6	0

Path (0)

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

Dist (1)

	0	1	2	3
0	0	1	3	6
1	INF	0	2	5
2	INF	INF	0	1
3	2	3	5	0

Path (1)

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

Dist (2)

	0	1	2	3
0	0	1	3	4
1	INF	0	2	3
2	INF	INF	0	1
3	2	3	5	0

Path (2)

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

Dist (3)

	0	1	2	3
0	0	1	3	4
1	5	0	2	3
2	3	4	0	1
3	2	3	5	0

Path (3)

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

3.5 一带权连通图含有五个顶点，采用邻接矩阵存储方式，并且邻接矩阵采用三元组表示，每个三元组的格式为：（顶点 1 的行号，顶点 2 的列号，边权重）。已知邻接矩阵含有 16 个非零元素，依次为（1,2,7）、（1,3,5）、（1,4，9）、（2,1,7）、（2,3,8）、（2,4,5）、（2,5,4）、（3,1,5）、（3,2,8）、（3,4,6）、（4,1,9）、（4,2,5）、（4,3,6）、（4,5,2）、（5,2,4）、（5,4,2）。

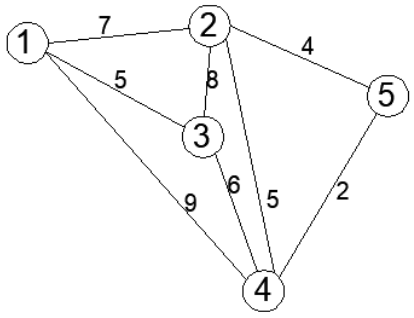
（1）请画出此连通图

（2）使用 Kruskal 方法，画出求该连通图最小生成树的具体过程。

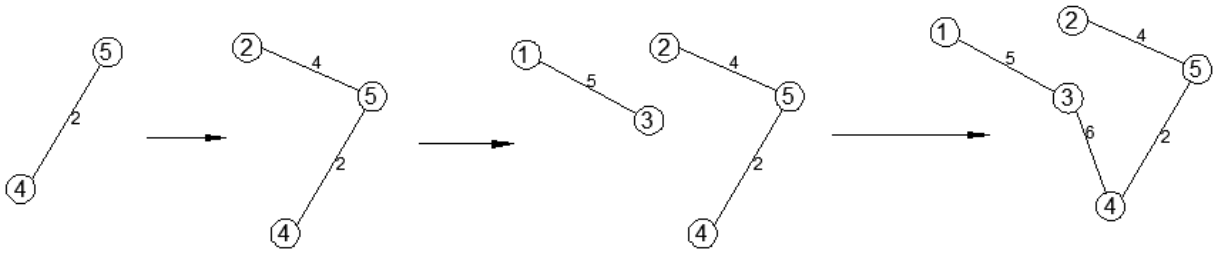
（3）使用 Prim 方法，画出求该连通图最小生成树的具体过程。

【解析】

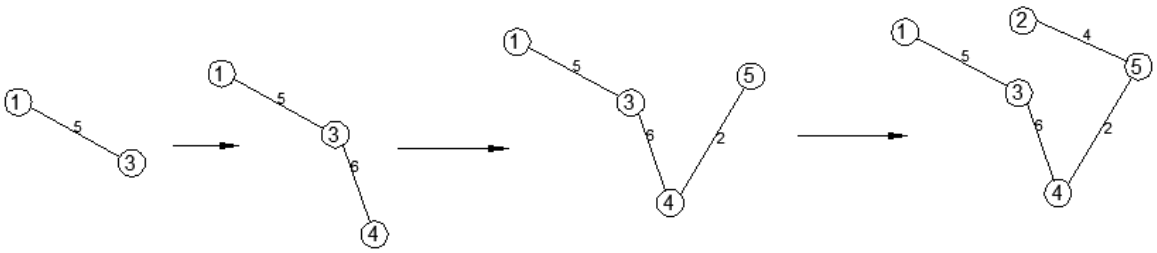
（1）该连通图如下图所示



（2）运用 Kruskal 算法生成最小生成树：



（3）运用 Prim 算法生成最小生成树（假设以顶点 1 为起点）：



四、算法设计（40 分）

4.1 试编写算法，求一棵以孩子—兄弟链表标识的树的高度。给定树中结点的定义为：

```
typedef struct CSNode {  //孩子-兄弟节点的定义
    char data;
    struct CSNode *firstChild;
    struct CSNode *nextSibling;
} CSNode, *CSTree;
```

函数的声明是：

int height(CSNode* T)//T 为根结点，返回树的高度。

【解析】

递归求解

```
int height (CSTree *T) {                    //求孩子兄弟链表表示的树 T 的深度
    int maxd, d;
    CSTree *p;
    if(T==NULL) return 0; //空树
    else {
      for(maxd=0,p=T->firstChild; p; p=p->nextSibling)
        if((d= height (p))>maxd)
              maxd = d;                     //子树的最大深度
      return maxd + 1;
  }//end if
}//end height
```

4.2 在古老的一维模式识别中,常常需要计算连续子向量的最大和,当向量全为正数的时候,问题很好解决。但是,如果向量中包含负数,是否应该包含某个负数,并期望旁边的正数会弥补它呢?

例如:{6,-3,-2,7,-15,1,2,2},连续子向量的最大和为 8(从第 0 个开始,到第 3 个为止)。

各位小皮皮有什么好的解法呢?(子向量的长度至少是 1)

【解析】

```
int FindGreatestSumOfSubArray(vector<int> array) {

    int max = INT_MIN, sum = 0;

    for(int i = 0; i < array.size(); ++i){
          sum += array[i];                  //累加
          if(sum > max) max = sum;  //
          if(sum < 0) sum = 0;
//当 sum 小于 0 时，就不要在继续往后累加了，因为这样只会越来越小。
                }

    return max;
}
```