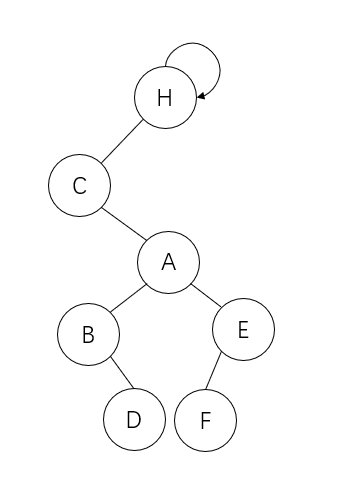
**2020秋数据结构模拟-参考答案**

一、选择题：

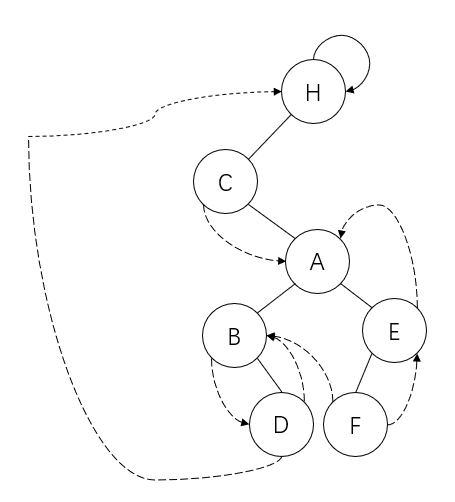
二、填空题：

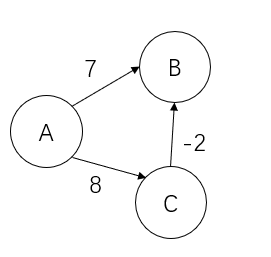
三、简答题：

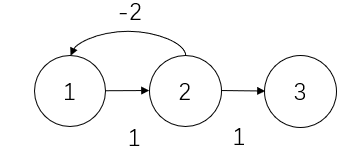
1.（1）：二叉树T如图所示：（3分）

（头节点的右孩子指向自己）

（2）：构造后续线索如图所示：（3分）

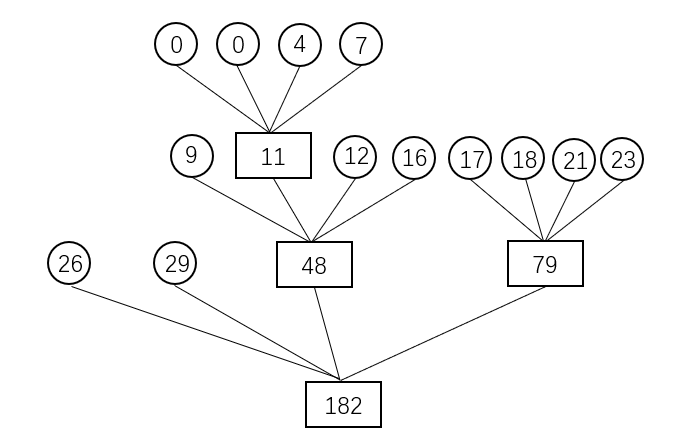


2：（1）：如图所示：，以A为源点，根据Dijkstra算法，贪心地将A至B的当前最短路确定为最短路，显然这个结果并不正确，存在负边权的边后，贪心地选取当前最短路并不一定是最终的最短路。

（2）：如图所示：，以1为源点，根据Floyd算法，修正矩阵第一行时，A[1][3]=2，但1至3的最短路不存在，因为1与2形成负边权回路，可无限环绕降低总边权。

3.(1)：{8,13,16,21,30},{10} （2分）

(2)：最佳归并树如图所示：（4分）



读写次数 = {（4+7）\*3+（9+12+16+17+18+21+23）\*2+（26+29）\*1}\*2 = 640（次） （2分）

解析：三、简答题

1.（1）由后序遍历策略左-右-根可知：C为根节点，再由中序遍历策略左-根-右可知，C无左子树。类似推断可知：A为子树的根节点，且BD位于A的左子树上，FE位于A的右子树上。考虑A的左子树，B必为根，最后结合中序序列可知，D在B的右子树上，同理可得，E是根，F位于E的左子树上。根据头节点定义，画出头节点，便于后序线索树的构造。

（2）根据题目给出的后序遍历序列，结合后续线索定义构造线索即可。

2.略。

3.（1）选择置换法生成初始归并段的基本算法。

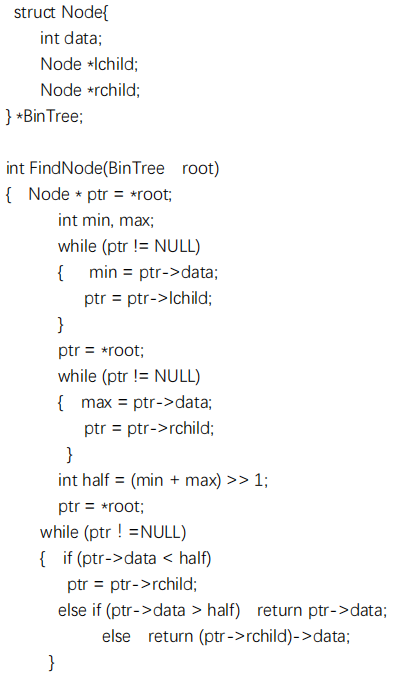
（2）归并路数K=4；总结点数m=11，u=（m-1）%（B-1）=10%3=1，虚段数C=K-u-1=2，最佳归并树即构造以归并段长度为权的哈夫曼树，叶子节点到根节点的路径长即为数据需要归并次数，一次归并=两次IO。

四、算法设计题：

1. 参考答案：

算法思想：在 BST 中，最小值就是最左边的结点，最大值就是最右边的结点。在分别求出min 和 max 后，求出 half。然后利用查找，找出第一个大于 half 的结点即为所求。O（logn） 或中序遍历最左最右即为所求，顺序或折半找出大于 half 的结点，复杂度为O（n）。

数据结构:



2.答：思路：

首先建立一个长度为10的计数数组（HASH表），存放每个数字的出现次数，初始时各单元（桶）均为0；

然后，扫描字符串，每碰到一个字符（数字），在Hash表中找到对应的项并把出现的次数增加一次。最后，扫描计数数组，得到每个字符出现的次数。

算法描述：

void FindChar(char\* str){  
 const int tableSize = 10;  
 unsigned int hashTable[tableSize];  
 **if**(str == NULL){  
 **return**;  
 }  
 int hashtable[tableSize]={0};  
 **for**(int i = 0; i < strlen(str) ; i++){  
 hashTable[str[i]]++;  
 }  
 **for**(int j = 0;j<strlen(str);j++){  
 **if**(hashTable[str[j]]==1){  
 printf("%c\n",str[j]);  
 **return**;  
 }  
 }  
 **if**(j == strlen(str)){  
 printf("不存在这样的字符\n");  
 }  
}

时间复杂度：

算法的基本操作时扫描字符串的每个字符并对其出现的次数进行计数，因此时间复杂度是O（n）

3.答：思路：

当森林采用二叉链表表示法时，若结点没有孩子，则它必是叶子结点，总的叶子结点个数是孩子子树上的叶子树和兄弟子树上叶节点个数之和

代码：

**typedef** **struct** node{  
 datatype data;  
 **struct** node\* firstchild,rightsib;  
}\*Forest;  
int LeavesCounter(Forest f){  
 **if**(t==NULL) **return** 0;  
 **else** **if**(f->firstchild == NULL) **return** (1+LeavesCounter(f->rightsib));  
 **else** **return** LeavesCounter(f->firstchild)+LeavesCounter(f->rightsib);  
}

时间复杂度分析：

采用树的递归遍历算法，所以时间复杂度与二叉树的遍历算法相同，分别为O（n）和O（logn）.