习题课讲义1

报告人 李达天

中国科学技术大学
University of Science and Technology of China

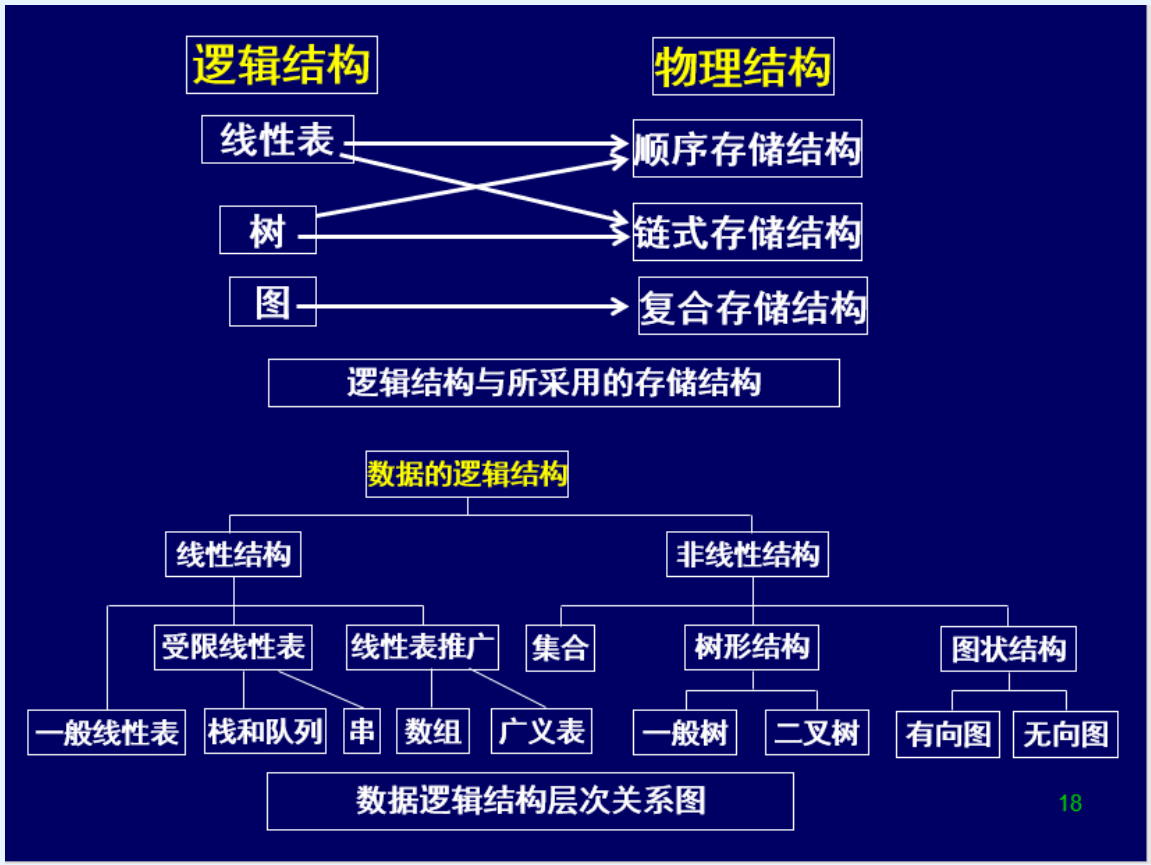
01

HW1

绪论（预备知识）

HW1-1

- 数据的逻辑结构？数据的物理结构？逻辑结构与物理结构的区别和联系是什么？



HW1-2

- 时间复杂度

(1)

```
Sum1( int n )
    {   int p=1, sum=0, m ;
        for (m=1; m<=n; m++)
            {  p*=m ; sum+=p ;  }
        return (sum) ;
    }
```

一层循环，O(n)

HW1-2

- 时间复杂度

(3) 递归函数

```
fact( int n )
   {   if (n<=1)  return(1) ;
       else return( n*fact(n-1)) ;
   }
```

递归树，O(n)

(2)

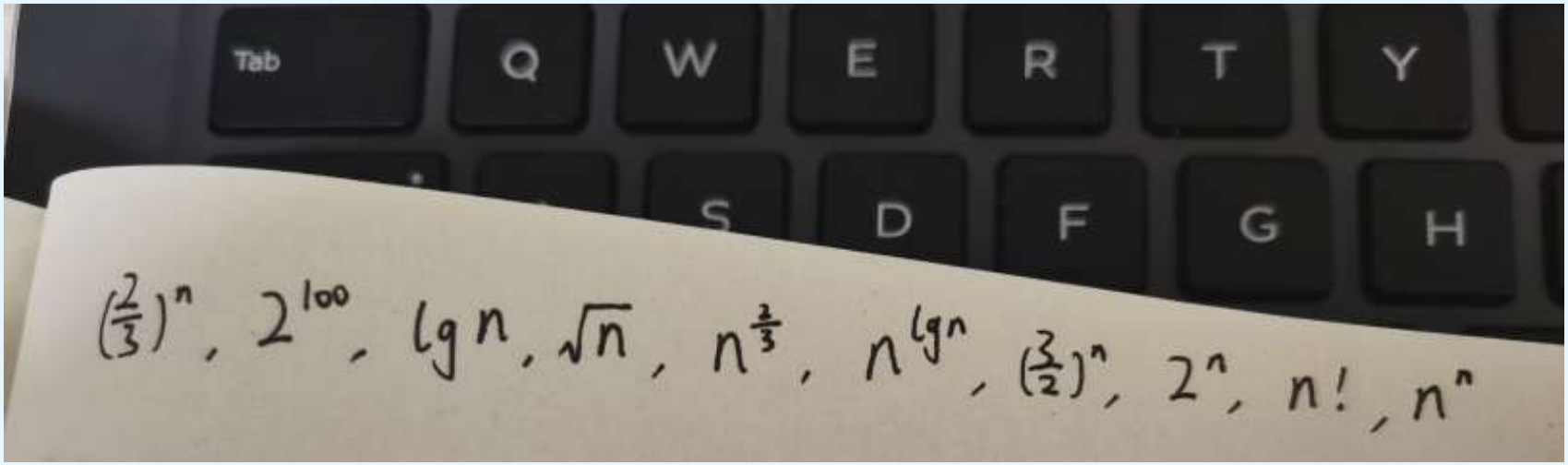
```
Sum2( int n )
   {   int sum=0, m, t ;
       for (m=1; m<=n; m++)
           {   p=1 ;
               for (t=1; t<=m; t++)  p*=t ;
               sum+=p ;
           }
       return (sum) ;
   }
```

两层循环，$O(n^2)$

HW1-3

- 按增长率由小至大的顺序排列下列各函数：

$$2^{100}, (3/2)^n, (2/3)^n, n^n, \sqrt{n}, n!, 2^n, \lg n, n^{\lg n}, n^{3/2}$$



HW1-3

指出当n足够大时，哪一个较优，哪一个较劣？

(1) $T_1(n)=5n^2-3n+60\lg n$ (2) $T_2(n)=3n^2+1000n+3\lg n$

(3) $T_3(n)=8n^2+3\lg n$ (4) $T_4(n)=1.5n^2+6000n\lg n$

- 答：4较优，3较劣（比较最高项系数即可）

01

HW2

线性表

HW2-1

- 为什么在单循环链表中设置尾指针比设置头指针更好?
- 答：尾指针是指向终端结点的指针，用它来表示单循环链表可以使得查找链表的开始结点和终端结点都很方便，查找时间都是O(1)。 若用头指针来表示该链表，则查找终端结点的时间为O(n)。

HW2-2 环形链表

- 如何判断单链表是否存在环？

```
bool hasCycle(struct ListNode* head) {
    if (head == NULL || head->next == NULL) {
        return false;
    }
    struct ListNode* slow = head;
    struct ListNode* fast = head->next;
    while (slow != fast) {
        if (fast == NULL || fast->next == NULL) {
            return false;
        }
        slow = slow->next;
        fast = fast->next->next;
    }
    return true;
}
```

HW2-2 相交链表

- 如何判断两个单链表相交，以及相交点?

```
struct ListNode *getIntersectionNode(struct ListNode *headA, struct ListNode *headB) {
    if (headA == NULL || headB == NULL) {
        return NULL;
    }
    struct ListNode *pA = headA, *pB = headB;
    while (pA != pB) {
        pA = pA == NULL ? headB : pA->next;
        pB = pB == NULL ? headA : pB->next;
    }
    return pA;
}
```

HW2-3

- 删除单链表中值重复的结点

```
void DeleteSame(Linklist *L)
{
    LNode *p,*q,*s;
    p = (*L)->next;

    for(p;p!=NULL;p=p->next)
    {
        s=p;                        //s指向要删除结点的前驱
        for(q=p->next;q!=NULL; )
        {
            if(q->data==p->data)
            {
                s->next=q->next;
                free(q);
                q=s->next;
            }
            else
            {
                s=q;
                q=q->next;
            }
        }
    }
}
```

快慢指针拓展

- 怎么找到单链表入环的第一个结点?

```
struct ListNode* detectCycle(struct ListNode* head) {
    struct ListNode *slow = head, *fast = head;
    while (fast != NULL) {
        slow = slow->next;
        if (fast->next == NULL) {
            return NULL;
        }
        fast = fast->next->next;
        if (fast == slow) {
            struct ListNode* ptr = head;
            while (ptr != slow) {
                ptr = ptr->next;
                slow = slow->next;
            }
            return ptr;
        }
    }
    return NULL;
}
```

拓展

- 思考：如何在时间复杂度为O(n)，空间复杂度为O(1)的情况下判断一个链表是否为回文链表？

- 思路一：栈
- 思路二：快慢指针

备考经验

- 去年题型：选择、判断、填空、简答题（包含较短的编程题）、编程题（两三道，手写代码）
- 考试范围：可能会有一些细碎的知识点考察，时间允许的话建议完整过一遍知识点（隔壁马老师名言：计算机是文科orz）
- 重点数据结构要熟练掌握，如线性表、栈和队列、二叉树

谢谢！

祝大家考试顺利，生活愉快~

中国科学技术大学
University of Science and Technology of China

习题课

hw 3,4

hw3

1. 已知从1至n的数字序列，按顺序入栈，每个数字入栈后即可出栈，也可在栈中停留，请设计算法验证给定的出栈的数字队列是否合法

- 从 1 到 n 依次入栈
- 每当一个数入栈后
  - While(栈非空&&栈顶元素==array[i])　　// a 为要求验证的数字队列
    - { S.pop(); i++; }
- 若最后栈非空或未将队列全部遍历, 则输出非法, 否则合法

hw3

1 2 3 4 5 6 7 8 9 = 110;

请看上边的算式，为了使等式成立，需要在数字间填入加号或者减号（可以不填，但不能填入其它符号）。之间没有填入符号的数字组合成一个数，例如：12+34+56+7-8+9 就是一种合格的填法；123+4+5+67-89 是另一个可能的答案。请你利用计算机的优势，帮助警察同志快速找到所有答案。每个答案占一行。形如：

12+34+56+7-8+9

123+4+5+67-89

47

hw3

- 可能的方式有 $3^8 = 6561$种, 可以直接穷举

- 方法一: 8层循环
- 方法二: 使用单层循环, 循环 3^8 次, 每轮循环中循环控制变量 i 唯一地确定一种符号插入方式
- 方法三: 递归

hw3

```
int main()
{
    int i, j, k, prev_oper, s = 0;
    int cur_number, sum, count = 0;
    for (i = 0; i < pow(3, N - 1); i++)
    {
        int tmp = i;
        sum = 0;
        prev_oper = 1; // 当前处理数字之前的符号
        for (j = 0; j < N - 1; j++)
        {
            oper[j] = tmp % 3; // 0: 无符号, 1: 加号+, 2: 减号
            tmp /= 3;
        }
        for (j = 0; j < N - 1; j++)
        {
            cur_number = 0;
            if (oper[j] == 0 && j + 1 < N - 1)
            {
                // 若无符号且未遍历到最后一个符号位, 当前数字位数加 1
                count++;
                continue;
            }
            else
            {
                for (k = 0; k <= count; k++)
                {
                    cur_number = cur_number * 10;
                    cur_number += nums[j - count + k];
                }
                if (prev_oper == 1)
                    sum += cur_number;
                else
                    sum -= cur_number;
                count = 0;
                prev_oper = oper[j];
            }
        }
        if (sum == 110)
        {
            output_result();
            s++;
        }
    }
    printf("%d", s);
}
```

hw4

1. 已知模式串t = ‘abcaab*abc’，其中“*”为单一字符的通配符，请写出用KMP算法所得的next数组（数组下标从1开始）。

hw4

$$next[j]=\begin{cases}0 & j=1\\ P[1\ldots j-1]\text{中前后缀相等的最大真子串的长度加1(包括空串),即:} & \\ \text{Max}\{k\mid 1\le k<j \text{ 且” } p_1\cdots p_{k-1}\text{”=“}p_{j-k+1}\cdots p_{j-1}\text{”}\} \text{ //k=1时,为空串} & \end{cases}$$

例:

j	1	2	3	4	5	6	7	8
P	a	b	a	a	b	c	a	c
next[j]	0	1	1	2	2	3	1	2

25

hw4

j	1	2	3	4	5	6	7	8	9	10
P	a	b	c	a	a	b	*	a	b	c
next[j]	0	1	1	1	2	2	3	4	5	3

谢谢！

请提宝贵意见

中国科学技术大学
University of Science and Technology of China

Data Structure Tree & Graph

赵雅馨

zhaoyaxin@mail.ustc.edu.cn

https://a6zmqofujm.feishu.cn/docs/doccnglPlvnsBFigEh4n9y0nzif

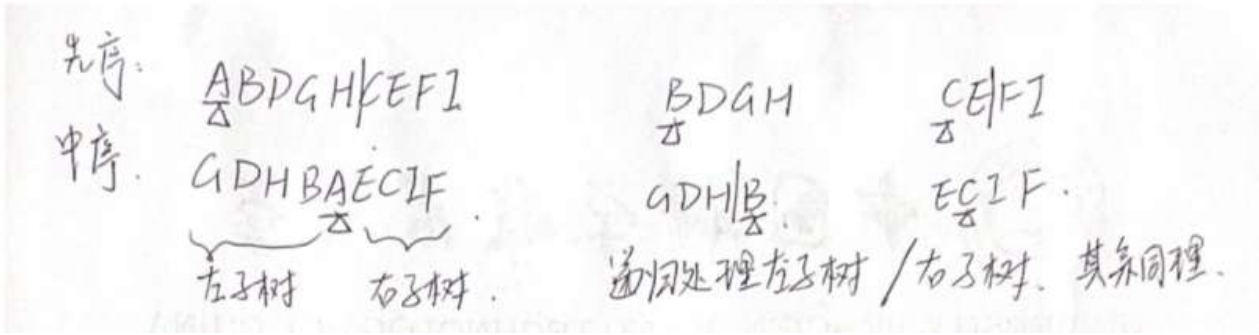
OUTLINE

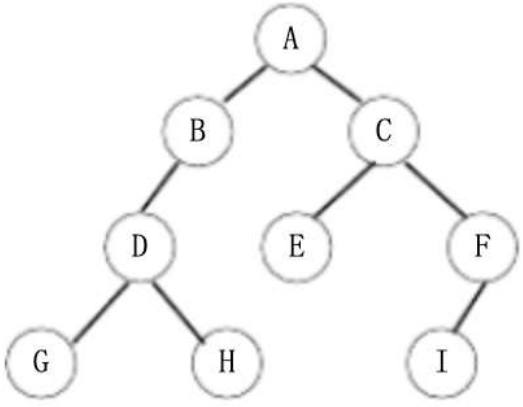
- HW5
  - **Binary Tree**
  - Huffman Tree
- HW6
  - Kruskal

HW5

已知一棵二叉树的先序遍历序列和中序遍历序列分别为ABDGHCEFI和GDHBAECIF，请画出这棵二叉树对应的中序线索树，然后给出该树的后序遍历序列。

step1: 根据先序遍历序列和中序遍历序列构建二叉树





HW5

已知一棵二叉树的先序遍历序列和中序遍历序列分别为ABDGHCEFI和GDHBAECIF，请画出这棵二叉树对应的中序线索树，然后给出该树的后序遍历序列。

§6.4 线索二叉树

1. 基本概念

在一基本数据结构上常常需要扩充，增加辅助信息，其目的是：

①开发新操作；②加速已有操作。

- 线索 —— 利用空指针域(n+1个)存放指向结点在某种遍历次序下的前驱和后继指针
- 线索链表 —— 加上线索的二叉链表
- 线索二叉树 —— 相应的二叉树称为线索二叉树
- 目的：加速遍历操作

  加速查找任一结点在某种遍历次序下的前驱和后继操作
- 结点结构

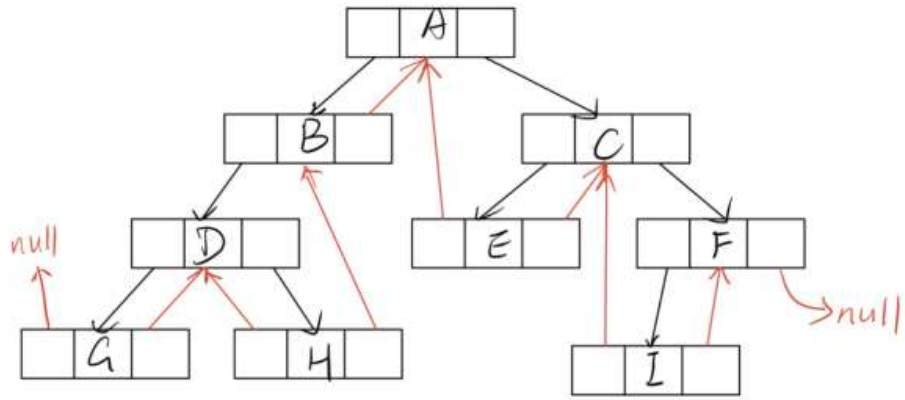
lchild	ltag	data	rtag	rchild

- 线索标志

$$\text{左标志ltag} = \begin{cases} 0: \text{lchild为左指针，指向孩子} \\ 1: \text{lchild为左线索，指向前驱} \end{cases}$$

$$\text{右标志rtag} = \begin{cases} 0: \text{rchild为右指针，指向孩子} \\ 1: \text{rchild为右线索，指向后继} \end{cases}$$

step2: 画出这棵二叉树对应的中序线索树



step3: 给出该树的后序遍历序列

GHDBEIFCA

OUTLINE

- HW5
  - Binary Tree
  - **Huffman Tree**
- HW6
  - Kruskal

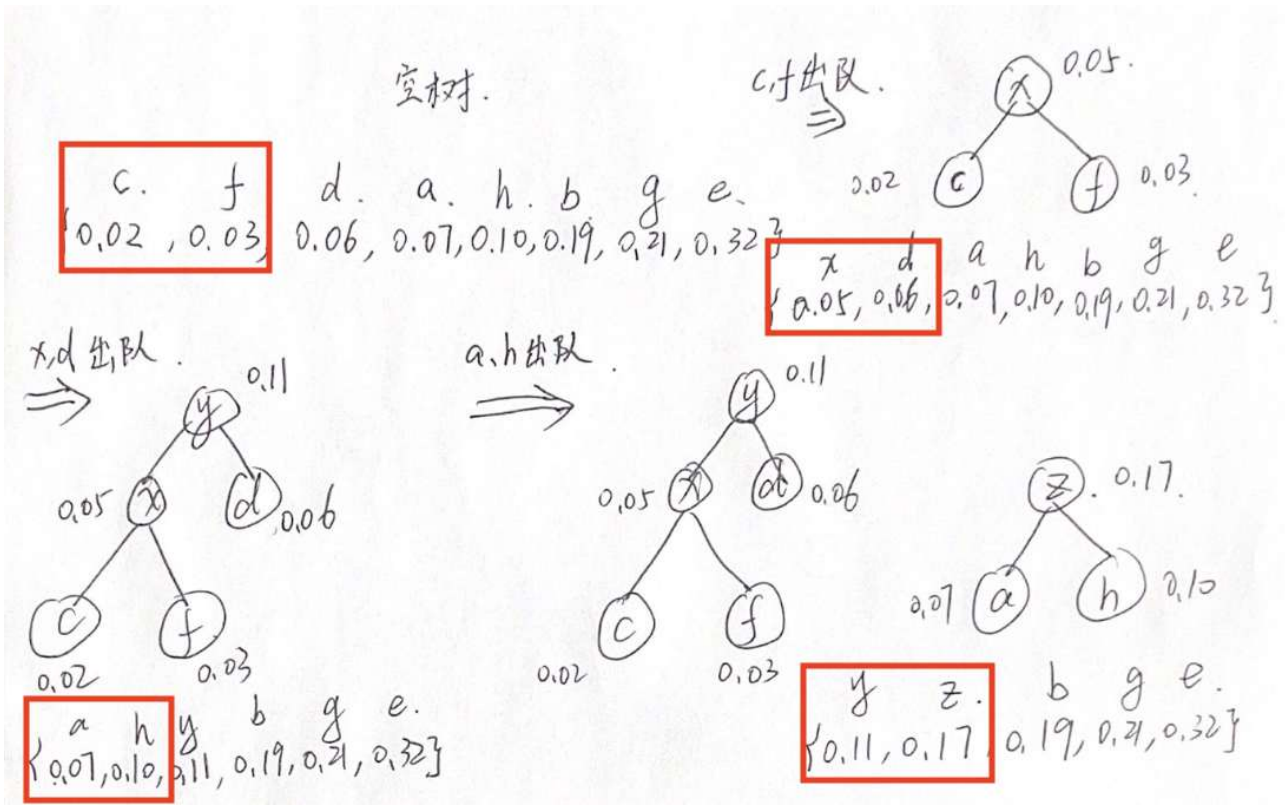
HW5

假设用于通信的电文是由字符集{a, b, c, d, e, f, g, h}中的字符构成， 这8个字符在电文中出现的概率分别为{0.07, 0.19, 0.02, 0.06, 0.32, 0.03, 0.21, 0.10} 。
① 请画出对应的huffman树(按左子树根结点的权小于等于右子树根结点的权的次序构造)。
② 求出每个字符的huffman编码。

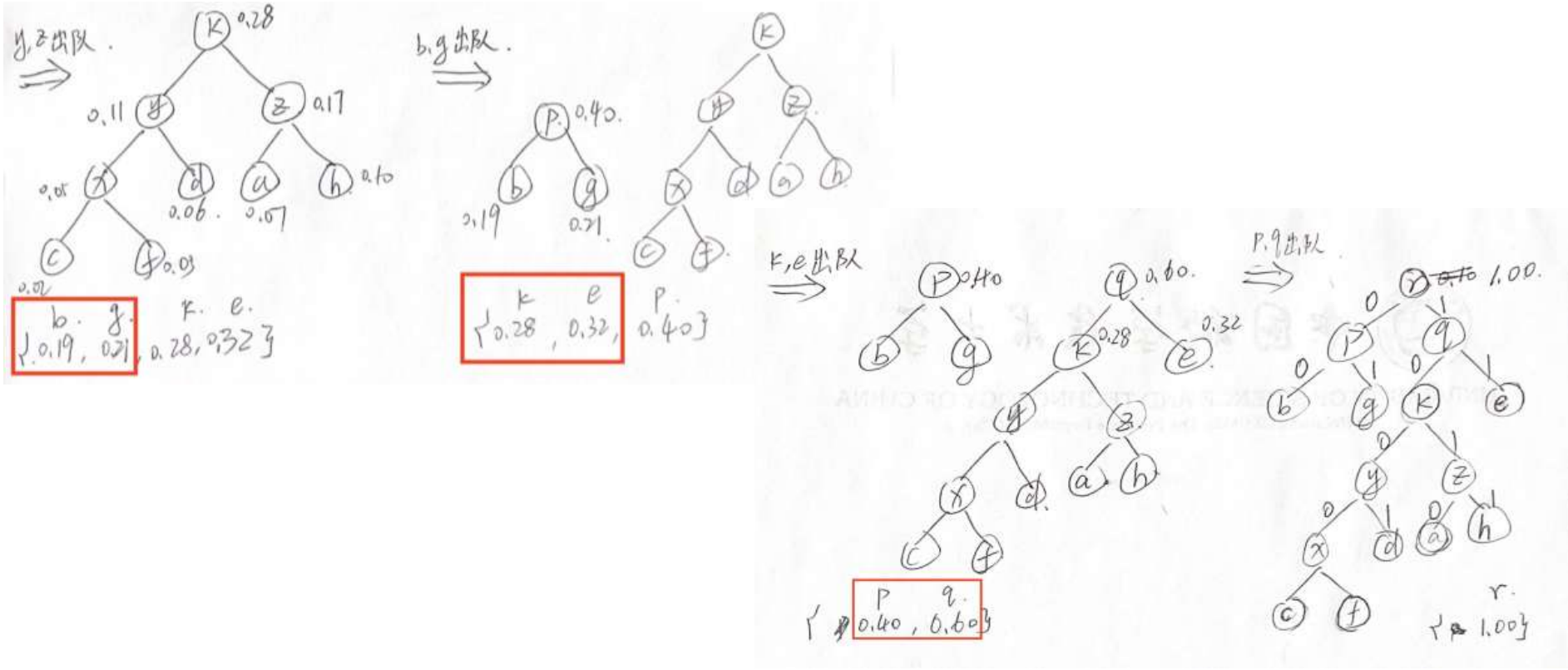
贪心思想：出现频率比较多的字符，用短一些的编码；出现频率比较少的字符，用长一些的编码。

我们把每个字符看作一个节点，并且辅带着把频率放到优先级队列中。我们从队列中取出频率最小的两个节点 A、B，然后新建一个节点 C，把频率设置为两个节点的频率之和，并把这个新节点 C 作为节点 A、B 的父节点。最后再把 C 节点放入到优先级队列中。重复这个过程，直到队列中没有数据。

HW5


空树.
c,f出队.
0.05
x
0.02
c
f
0.03
c. f
{0.02, 0.03}
d. a. h. b g e.
0.06, 0.07, 0.10, 0.19, 0.21, 0.32}
x d
{0.05, 0.06,
a h b g e
0.07, 0.10, 0.19, 0.21, 0.32}
x,d出队.
0.11
y
0.05
x
d
0.06
c
f
0.02
0.03
a h
{0.07, 0.10,
y b g e.
0.11, 0.19, 0.21, 0.32}
a,h出队.
0.11
y
0.05
x
d
0.06
c
f
0.02
0.03
z. 0.17.
0.07
a
h
0.10
y z.
{0.11, 0.17,
b g e.
0.19, 0.21, 0.32}

HW5


y, z出队.
K 0.28
y 0.11
z 0.17
x 0.08
d 0.06
a 0.07
h 0.10
c
f 0.03
0.02
b. g.
{0.19, 0.21}
K. e.
{0.28, 0.32}
b, g出队.
P 0.40.
b
g
0.19
0.21
K
H
Z
x
d
a
h
c
f
{0.28, 0.32, 0.40}
K, e出队
P 0.40
q 0.60.
0.28
0.32
P, q出队
r 1.00.
0
1
{0.40, 0.60}
{1.00}

OUTLINE

- HW5
  - Binary Tree
  - Huffman Tree
- HW6
  - **Kruskal**

HW6

实现Kruskal算法，即将课堂上PPT中算法伪代码补充完整（允许调用已有的排序算法，或假设边集中的边已按边权增序排列）

Part1. 解题思路

贪心思想

Kruskal算法与prim算法不同：

- Prim算法是以 顶点 为基础（每次寻找离Vnew最近的顶点）；
- 而Kruskal算法是以 边 为基础，每次从 边 集合中寻找最小的边（不管两个顶点属于V还是Vnew），然后判断该边的两个顶点是否同源（属于同一个连通分量）。

📌 Kruskal需要对所有的边进行排序，然后从小到大，依次遍历每条边，同时判断每条边是否同源，如果同源，跳过；如果不同源，将两个连通分量合并，直到所有顶点属于同一个连通分量，算法结束。

数据结构很明显了 —— 我们需要用到并查集。

HW6

实现Kruskal算法，即将课堂上PPT中算法伪代码补充完整（允许调用已有的排序算法，或假设边集中的边已按边权增序排列）

Part2. 数据结构

因为算法要求我们对所有边进行排序，同时需要知道每条边的两个端点

所以可以建立一个结构体/类，保存以上三个属性<start,end,len>（这里称为点-边式）。

其中start和end分别为两个顶点， len为两顶点的权值，即两点之间的距离

C++ ▾ 自动换行

```
1  struct VP {
2   int start; // 顶点1
3   int end; // 顶点2
4   int len; // 长度
5  };
```

HW6

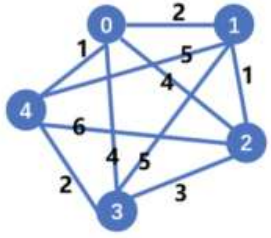
实现Kruskal算法，即将课堂上PPT中算法伪代码补充完整（允许调用已有的排序算法，或假设边集中的边已按边权增序排列）

Part3. 步骤

1. 初始化：将图（邻接矩阵或邻接表）转换成点-边式，并对点-边式按边的长度进行排序。同时，初始化并查集；
2. 依次遍历所有的点-边式，每次取最小值。
3. 作如下判断：如果该点-边式的两个顶点同源，跳过; 如果该点-边式的两个顶点不同源，则将这两个源（连通分量）合并
4. 重复步骤2, 直到存在一个连通分量，包含了图中所有的节点
5. 算法结束

HW6

Part4. 举例



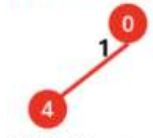
Kruskal没有起点

1.先对所有边进行排序

1：节点0-节点4
1：节点1-节点2
2：节点0-节点1
2：节点3-节点4
3：节点2-节点3
4：节点0-节点2
4：节点0-节点3
5：节点1-节点3
5：节点1-节点4
6：节点2-节点4

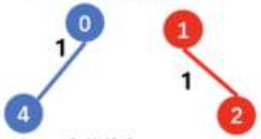
2.依次选取最短的边构成连通分量

1：节点0-节点4



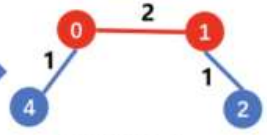
合并节点0，4

1：节点1-节点2



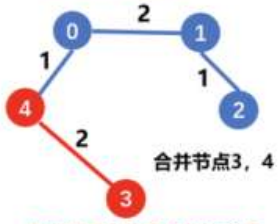
合并节点1，2

2：节点0-节点1



合并节点0，1

2：节点3-节点4



合并节点3，4

连通分量已经包含所有节点
计算长度1+1+2+2=6
结束

HW6

2、Kruskal算法

- 特点：当前形成的集合T除最终结果外，始终是森林，无环。
- 算法：

```
KruskalMST(G){
  T=(V, φ)；//包含n个顶点，不含边的森林；
  依权的增序对E(G)中边排序，设结果在E[0..e-1];
  for (i=0; i<e; i++) {
      取E中第i条边(u,v);
      if （u和v分属森林T中两棵不同的树） then
          T=T∪{（u,v）}; //否则产生回路，放弃此边
      if （T已是一棵树）then return T;
  }//endfor
}
```

15

```cpp
class Solution {
public:
    int minCostConnectPoints(vector<vector<int>>& points) {
        int res = 0;
        int n = points.size();
        Djset ds(n);
        vector<Edge> edges;
        // step1 初始化，建立点-边式数据结构
        for (int i = 0; i < n; i++) {
            for (int j = i + 1; j < n; j++) {
                // 假设cost定义为两点之间的曼哈顿距离，当然也可以作为自定义的输入
                Edge edge = {i, j, abs(points[i][0] - points[j][0]) + abs(point
                edges.emplace_back(edge);
            }
        }
        // step2 按边长度排序
        sort(edges.begin(), edges.end(), [](const auto& a, const auto& b) {
            return a.len < b.len;
        });

        // step3 连通分量合并
        for (auto& e : edges) {
           res = ds.merge(e.start, e.end, e.len);
           if (res != -1) return res;
        }
        return 0;
    }
};
```

HW6

2、Kruskal算法

- 特点：当前形成的集合T除最终结果外，始终是森林，无环。
- 算法：

```
KruskalMST(G){
  T=(V, φ)；//包含n个顶点，不含边的森林；
  依权的增序对E(G)中边排序，设结果在E[0..e-1];
  for (i=0; i<e; i++) {
      取E中第i条边(u,v);
      if （u和v分属森林T中两棵不同的树） then
          T=T∪{（u,v）}; //否则产生回路，放弃此边
      if （T已是一棵树）then return T;
  }//endfor
}
```

```cpp
int find(int x) {
    // 路径压缩：压缩方式是，直接指向根节点
    if (x != parent[x]) {
        parent[x] = find(parent[x]);
    }
    return parent[x];
}

int merge(int x, int y, int length) {
    int rootx = find(x);
    int rooty = find(y);
    if (rootx != rooty) {
        // 按秩合并
        if (rank[rootx] < rank[rooty]) {
            swap(rootx, rooty);
        }
        parent[rooty] = rootx;
        if (rank[rootx] == rank[rooty]) rank[rootx] += 1;
        // rooty的父节点设置为rootx,同时将rooty的节点数和边长度累加到rootx,
        size[rootx] += size[rooty];
        len[rootx] += len[rooty] + length;
        // 终止条件：如果某个连通分量的节点数 包含了所有节点，直接返回边长度
        if (size[rootx] == num) return len[rootx];
    }
    return -1;
}
```

```cpp
class Djset {
public:
    vector<int> parent; // 记录节点的根
    vector<int> rank;   // 记录根节点的深度（用于优化）
    vector<int> size;   // 记录每个连通分量的节点个数
    vector<int> len;    // 记录每个连通分量里的所有边长度
    int num;            // 记录节点个数
```

预祝大家考试顺利!


World Champion
TOM
争做认真学习冠军


我不许你停止学习


世界纷纷扰扰与我无关
我的心里只有看书做题