**周报**

本周学习了《零基础学算法》的第一第二章。学习的主要内容终结如下：

1. 基础算法思想
2. 递推算法思想：通过通过已知条件，利用特定关系得出中间推论，，逐步推论得到结果
3. 顺推法（斐波那契数列）
4. 逆推法（该存多少钱）

2、每局 （穷举）算法思想：从所有的答案中逐一去除候选答案，并验证该候选答案是否为正确的解。

1. 实例：填数游戏，运算字符
   1. 递归算法思想：一种直接或间接地调用自身的算法。
2. 条件：要有一个递归结束条件
3. 递归算法解题的运算效率较低
4. 实例：求阶乘、数制转换
   1. 分治算法思想：将大规模N的问题，分解成M个规模小的子问题，这些子问题相互独立，并且与原问题形式相同，递归的解这些子问题，然后将各子问题的解合并到原问题的解，这种解法叫做分治法。
5. 分解设计，一般步骤：

分解：将求解的问题划分成若干较小规模的问题

求解：当子问题划分得到足够小时，得到子问题的解

合并：按求解问题的要求，将子问题的解逐层合并，构成最终的解

1. 实例：乒乓球比赛日程安排。
   1. 贪婪算法思想：总是做出当前最好的选择。也就是说从整体最优考虑，它所作出选择只是局部最优选择。
2. 特点和思想：

不能保证最后的解释最优解

不能用来求最大或者最小值

只能满足某些约束条件的可行解的范围

1. 实例：换零钱
   1. 试探算法思想（回溯法）：他是一种系统的搜索问题的方法
2. 算法思路：为例求得问题的解，先选择某一种可能情况进行试探，在探索过程中，一旦发现原来选择的假设情况是错误的就返回一步重新选择，继续向另一个方向试探，如此反复进行，直至到解或证明无解。
3. 试探算法一般采用递归形式编写程序
4. 实例：成成彩票号码。
   1. 模拟算法：适用于不能创建数学模型
5. 算法思路：可使用随机函数来模拟自然界发生的不可能预测的情况。（C语言中使用srand()和rand()函数生成随机数）
6. 实例：猜数游戏、模拟筛子游戏
7. 简单数据结构（线性结构、树形结构、网状结构）
8. 线性表：顺序线性、链式线性

（1） 线性表结构特征

* 1. 有且只有一个“首元素”
  2. 有且只有一个“末元素”
  3. 除末元素之外，其余元素均有唯一的后续元素
  4. 除首元素之外，其余元素均有唯一的前驱元素

（2） 对于线性表有一下操作：

① 添加节点

② 插入节点

③ 删除节点

* 1. 查找节点
  2. 遍历节点
  3. 统计节点

（3） 定义顺序表结构：

typedef struct //定义顺序表结构

{

DATA ListData[MAXSIZE+1]; //保存顺序表的数组

int ListLen; //顺序表已存结点 的数量

}SeqListType;

（4） 添加节点：（添加到中间）

for(i=SL->ListLen;i>=n;i--) //将顺序表中的数据向后移动

SL->ListData[i+1]=SL->ListData[i];

SL->ListData[n]=data; //插入结点

SL->ListLen++; //顺序表结点数量增加1

（5）添加数据到末尾：

SL->ListData[++SL->ListLen]=data;

（6）删除节点：

for(i=n;i<SL->ListLen;i++) //将顺序表中的数据向前移动

SL->ListData[i]=SL->ListData[i+1];

SL->ListLen--; //顺序表元素数量减1

（7）按关键之查找节点：

for(i=1;i<=SL->ListLen;i++)

if(strcmp(SL->ListData[i].key,key)==0) //如果找到所需结点

return i; //返回结点序号

1. 操作链表

单链表的每一个节点包括两个部分：一部分是实际数据，另一个部分是下一个节点的地址。

1. 定义链表的结构：

typedef struct Node

{

DATA data; //数据

struct Node \*next; //下一个节点的地址

}ChainListType;

1. 添加节点至尾部：

node->data=data; //保存数据

node->next=NULL; //设置结点指针为空，即为表尾

if(head==NULL) //是头指针

{

head=node;

return head;

}

h=head;

while(h->next!=NULL) //查找链表的末尾

h=h->next ;

h->next=node;

1. 添加节点至首部：

node->data=data; //保存数据

node->next=head; //指向头指针所指结点

head=node; //头指针指向新增结点

1. 插入节点：

node->data=data; //保存结点中的数据

node1=ChainListFind(head,findkey);//查找节点的位置，返回关键点的指针。

if(node1) //若找到要插入的结点

{

node->next=node1->next; //新插入结点指向关键结点的下一结点

node1->next=node; //设置关键结点指向新插入结点

}

1. 根据关键字查找节点：

while(h) //若结点有效，则进行查找

{

if(strcmp(h->data.key,key)==0) //若结点关键字与传入关键字相同

return h; //返回该结点指针

h=h->next; //处理下一结点

}

1. 实例：用链表制作通讯录
2. 队列（队列是线性表的一种特殊形式）

只允许在表的前端进行删除操作，而在表的后端进行插入操作。

1. 原则：先来先服务（先进先出）
2. 常用的操作：
   1. 初始化：创建一个队列，置队头和队尾的序号
   2. 进队列：将一个元素添加到对尾
   3. 出队列：把对头元素去除，同事删除该元素
   4. 获取队列第一个元素，不删除该元素
   5. 获取该队列的长度：根据队头、队尾计算出队列中元素的数量
3. 定义顺序队列结构

typedef struct

{

DATA data[QUEUEMAX]; //队列数组

int head; //队头

int tail; //队尾

}SeqQueue;

1. 初始化顺序队列：创建一个队列，申请保存队列内存，置队头和队尾的序号

if(q=(SeqQueue \*)malloc(sizeof(SeqQueue))) //申请保存队列的内存

{

q->head = 0;//设置队头

q->tail = 0;//设置队尾

return q;

}

1. 获取队列的状态：
   1. 队列是否为空：(q->head==q->tail);
   2. 对列是否已满：(q->tail==QUEUEMAX);
   3. 获取队列长度：(q->tail-q->head);
2. 入队操作：加入队列，修改队尾序号

q->data[q->tail++]=data; //执行完语句后再自加1

1. 出队操作：将对头元素取出。

return &(q->data[q->head++]);

1. 获取对头元素：至获取元素不删除队列中的元素

return &(q->data[q->head]);

1. 循环队列的操作：

（1）实例：银行排号程序

1. 后进先出结构：栈
2. 栈的操作只有两个：入栈与出栈
3. 栈的存储方法有两种：顺序栈和链式栈（以下都是顺序栈）
4. 定义顺序栈的结构：

typedef struct stack

{

DATA data[SIZE+1]; //数据元素

int top; //栈顶

}SeqStack;

1. 初始化栈：
   1. 申请内存空间,用来保存栈中的数据
   2. 设置栈顶指针值为0，表示是一个空栈

if(p=(SeqStack \*)malloc(sizeof(SeqStack))) //申请栈内存

{

p->top=0; //设置栈顶为0

return p;//返回指向栈的指针

}

1. 判断栈的状态：
   1. 判断栈是否是空栈：(s->top==0);
   2. 清空栈：s->top=0;
   3. 判断栈是否已满：(s->top==SIZE);
2. 入栈操作（Push）：s->data[++s->top]=data;//将元素入栈，先执行加1，后执行后面的语句
3. 出栈操作（Pop）：使栈顶指针减一，指向栈的下一个元素，原来元素被弹出。执行的顺序是：先弹出数据后指针减一。

出栈的代码：(s->data[s->top--]);

1. 获取（读取）栈顶元素（不删除栈的元素）：(s->data[s->top]);
2. 实例：算术表达式求值

**2016-9-25**

**系统控制与信息处理重点实验室**