学习情况表

|  |  |  |  |
| --- | --- | --- | --- |
| **姓名** | 牛钰欢 | **学号** | 2020905105 |
| **学院** | 信息学院 | **专业** | 通信工程 |

（请在下面表格本周学习情况）

|  |
| --- |
| 学习情况简述 |
| 1. 串：是由零个或多个字符组成的有限序列，又名叫字符串。一般记为s=”a1 a2......an”(n≥0)。s是串的名称，用双引号括起来的字符序列是串的值。 2. 零个字符的串称为空串，它的长度为零，可以直接用两双引号“""”表示，也可以用希腊字母“Φ”来表示。P.S.空格串，是只包含空格的串。注意它与空串的区别，空格串是有内容有长度的，而且可以不止一个空格。 3. 子串与主串，串中任意个数的连续字符组成的子序列称为该串的子串，相应地，包含子串的串称为主串。 4. 单词字符串在计算机中的大小取决于它们挨个字母的前后顺序。（“silly”<“stupid”）事实上，串的比较是通过组成串的字符之间的编码来进行的，而字符的编码指的是字符在对应字符集中的序号。计算机中的常用字符是使用标准的ASCII编码。 5. 给定两个串：s="a1 a2......an"，t="b1 b2......bm"，当满足以下条件之一时，s<t： 6. n<m，且ai=bi（i=1，2，……，n）。例如当s=“hap”，t=“happy” 7. 存在某个k≤min（m，n），使得ai=bi（i=1，2，……，k-1），ak<bk。例如当s="happen"，t="happy" 8. 串的存储结构与线性表相同，分为两种:   1)串的顺序存储结构:用一组地址连续的存储单元来存储串中的字符序列的。按照预定义的大小，为每个定义的串变量分配一个固定长度的存储区。    字符串操作可能导致上溢提示报错和对多出来的字符串截尾。于是对于串的顺序存储，有一些变化，串值的存储空间可在程序执行过程中动态分配而得。比如在计算机中存在一个自由存储区，叫做“堆”。  2）串的链式存储结构：一个结点可以存放一个字符，也可以考虑存放多个字符，最后一个结点若是未被占满时，可以用“#”或其他非串值字符补全。    但串的链式存储结构除了在连接串与串操作时有一定方便之外，总的来说不如顺序存储灵活，性能也不如顺序存储结构好。  7，模式匹配算法:对主串的每一个字符作为子串开头，与要匹配的字符串进行匹配。对主串做大循环，每个字符开头做T的长度的小循环，直到匹配成功或全部遍历完成为止。  8,根据等概率原则，平均是(n+m)/2次查找，时间复杂度为O(n+m)。  9,KMP模式匹配算法:在我们知道T串中首字符“a”与T中后面的字符均不相等的前提下:    对于在子串中有与首字符相等的字符:    我们把T串各个位置的j值的变化定义为一个数组next：     1. next数组值推导： 2. T="abcdex" next[j] 011111   1）当j=1时，next[1]=0；  2）当j=2时，j由1到j-1就只有字符“a”，属于其他情况next[2]=1；  3）当j=3时，j由1到j-1串是“ab”，显然“a”与“b”不相等，属其他情况，next[3]=1；  4）以后同理，所以最终此T串的next[j]为011111。  2）T="abcabx" next[j] 011123  1）当j=1时，next[1]=0；  2）当j=2时，同上例说明，next[2]=1；  3）当j=3时，同上，next[3]=1；  4）当j=4时，同上，next[4]=1；  5）当j=5时，此时j由1到j-1的串是“abca”，前缀字符“a”与后缀字符“a”相等（前缀用下划线表示，后缀用斜体表示），因此可推算出k值为2（由‘p1...pk-1’=‘pj-k＋1...pj-1’，得到p1=p4）因此next[5]=2；  6）当j=6时，j由1到j-1的串是“abcab”，由于前缀字符“ab”与后缀“ab”相等，所以next[6]=3。  P.S.我们可以根据经验得到如果前后缀一个字符相等，k值是2，两个字符k值是3，n个相等k值就是n+1。  3）T="ababaaaba" next[j] 011234223  1）当j=1时，next[1]=0；  2）当j=2时，同上next[2]=1；  3）当j=3时，同上next[3]=1；  4）当j=4时，j由1到j-1的串是“aba”，前缀字符“a”与后缀字符“a”相等，next[4]=2；  5）当j=5时，j由1到j-1的串是“abab”，由于前缀字符“ab”与后缀“ab”相等，所以next[5]=3；  6）当j=6时，j由1到j-1的串是“ababa”，由于前缀字符“aba”与后缀“aba”相等，所以next[6]=4；  7）当j=7时，j由1到j-1的串是“ababaa”，由于前缀字符“ab”与后缀“aa”并不相等，只有“a”相等，所以next[7]=2；  8）当j=8时，j由1到j-1的串是“ababaaa”，只有“a”相等，所以next[8]=2；  9）当j=9时，j由1到j-1的串是“ababaaab”，由于前缀字符“ab”与后缀“ab”相等，所以next[9]=3。  11,KMP模式匹配算法改进  1)T="ababaaaba" next[j] 011234223 nextval[j] 010104210  1）当j=1时，nextval[1]=0；  2）当j=2时，因第二位字符“b”的next值是1，而第一位就是“a”，它们不相等，所以nextval[2]=next[2]=1，维持原值。  3）当j=3时，因为第三位字符“a”的next值为1，所以与第一位的“a”比较得知它们相等，所以nextval[3]=nextval[1]=0；  4）当j=4时，第四位的字符“b”next值为2，所以与第二位的“b”相比较得到结果是相等，因此nextval[4]=nextval[2]=1；  5）当j=5时，next值为3，第五个字符“a”与第三个字符“a”相等，因此nextval[5]=nextval[3]=0；  6）当j=6时，next值为4，第六个字符“a”与第四个字符“b”不相等，因此nextval[6]=4；  7）当j=7时，next值为2，第七个字符“a”与第二个字符“b”不相等，因此nextval[7]=2；  8）当j=8时，next值为2，第八个字符“b”与第二个字符“b”相等，因此nextval[8]=nextval[2]=1；  9）当j=9时，next值为3，第九个字符“a”与第三个字符“a”相等，因此nextval[9]=nextval[3]=0。  总结改进过的KMP算法，它是在计算出next值的同时，如果a位字符与它next值指向的b位字符相等，则该a位的nextval就指向b位的nextval值，如果不等，则该a位的nextval值就是它自己a位的next的值。  12，树：树（Tree）是n（n≥0）个结点的有限集。n=0时称为空树。  注：1.n>0时根结点是唯一的，不可能存在多个根结点  2.m>0时，子树的个数没有限制，但它们一定是互不相交的  13，树的结点包含一个数据元素及若干指向其子树的分支。结点拥有的子树数称为结点的度。度为0的结点称为叶结点或终端结点；度不为0的结点称为非终端结点或分支结点。除根结点之外，分支结点也称为内部结点。树的度是树内各结点的度的最大值。  14，结点的子树的根称为该结点的孩子，相应地，该结点称为孩子的双亲。  15，如果将树中结点的各子树看成从左至右是有次序的，不能互换的，则称该树为有序树，否则称为无序树。  16，森林是m（m≥0）棵互不相交的树的集合。对树中每个结点而言，其子树的集合即为森林。  17，线性结构与树结构的比较    18，树的存储结构：利用顺序存储和链式存储结构的特点可以实现对树的存储结构的表示。我们这里要介绍三种不同的表示法：双亲表示法、孩子表示法、孩子兄弟表示法。  1）双亲表示法：我们假设以一组连续空间存储树的结点，同时在每个结点中，附设一个指示器指示其双亲结点在数组中的位置。    这样的存储结构，我们可以根据结点的parent指针很容易找到它的双亲结点，所用的时间复杂度为O(1)，直到parent为-1时，表示找到了树结点的根。可如果我们要知道结点的孩子是什么需要遍历整个结构才行。  改进：增加一个结点最左边孩子的域，不妨叫它长子域，这样就可以很容易得到结点的孩子。如果没有孩子的结点，这个长子域就设置为-1。    再改进：增加一个右兄弟域来体现兄弟关系，也就是说，每一个结点如果它存在右兄弟，则记录下右兄弟的下标。同样的，如果右兄弟不存在，则赋值为-1。     1. 孩子表示法：一种是指针域的个数就等于树的度，即树各个结点度的最大值。第二种方案每个结点指针域的个数等于该结点的度，我们专门取一个位置来存储结点指针域的个数。     改进：把每个结点的孩子结点排列起来，以单链表作存储结构，则n个结点有n个孩子链表，如果是叶子结点则此单链表为空。然后n个头指针又组成一个线性表，采用顺序存储结构，存放进一个一维数组中，如上右图。  其中child是数据域，用来存储某个结点在表头数组中的下标。next是指针域，用来存储指向某结点的下一个孩子结点的指针。表头数组的表头结点中，data是数据域，存储某结点的数据信息。firstchild是头指针域，存储该结点的孩子链表的头指针。  改进：双亲孩子表示法     1. 孩子兄弟表示法：我们设置两个指针，分别指向该结点的第一个孩子和此结点的右兄弟   其中data是数据域，firstchild为指针域，存储该结点的第一个孩子结点的存储地址，right-sib是指针域，存储该结点的右兄弟结点的存储地址。这个表示法的最大好处是它把一棵复杂的树变成了一棵二叉树。     1. 二叉树的特点有：每个结点最多有两棵子树，所以二叉树中不存在度大于2的结点。左子树和右子树是有顺序的，次序不能任意颠倒。 2. 二叉树具有五种基本形态： 1）空二叉树。 2）只有一个根结点。 3）根结点只有左子树。 4）根结点只有右子树。 5）根结点既有左子树又有右子树。 3. 特殊二叉树：1）斜树：所有的结点都只有左子树的二叉树叫左斜树。所有结点都是只有右子树的二叉树叫右斜树。这两者统称为斜树。   2）满二叉树：在一棵二叉树中，如果所有分支结点都存在左子树和右子树，并且所有叶子都在同一层上，这样的二叉树称为满二叉树。  3）完全二叉树：对一棵具有n个结点的二叉树按层序编号，如果编号为i（1≤i≤n）的结点与同样深度的满二叉树中编号为i的结点在二叉树中位置完全相同，则这棵二叉树称为完全二叉树。完全二叉树的特点：（1）叶子结点只能出现在最下两层。（2）最下层的叶子一定集中在左部连续位置。（3）倒数二层，若有叶子结点，一定都在右部连续位置。（4）如果结点度为1，则该结点只有左孩子，即不存在只有右子树的情况。（5）同样结点数的二叉树，完全二叉树的深度最小。   1. 完全二叉树的性质：性质1：在二叉树的第i层上至多有2i-1个结点（i≥1）   性质2：深度为k的二叉树至多有2k-1个结点（k≥1）。  性质3：对任何一棵二叉树T，如果其终端结点数为n0，度为2的结点数为n2，则n0=n2 +1。  性质4：具有n个结点的完全二叉树的深度为|log2n+1|（|x|表  示不大于x的最大整数）。  性质5：如果对一棵有n个结点的完全二叉树（其深度为）的结点按层序编号（从第1层到第层，每层从左到右），对任一结点i（1≤i≤n）有：  1）如果i=1，则结点i是二叉树的根，无双亲；如果i>1，则其双亲是结点。  2）如果2i>n，则结点i无左孩子（结点i为叶子结点）；否则其左孩子是结点2i。  3）如果2i+1>n，则结点i无右孩子；否则其右孩子是结点2i+1。   1. 二叉树的存储结构：1）二叉树的顺序存储结构就是用一维数组存储二叉树中的结点，并且结点的存储位置，也就是数组的下标要能体现结点之间的逻辑关系。（顺序存储结构一般只用于完全二叉树。）2）二叉链表：      1. 遍历二叉树：指从根结点出发，按照某种次序依次访问二叉树中所有结点，使得每个结点被访问一次且仅被访问一次。 2. 二叉树的遍历方法：     1）前序遍历：规则是若二叉树为空，则空操作返回，否则先访问根结点，然后前序遍历左子树，再前序遍历右子树。遍历的顺序为：ABDGH-CEIF。  2）中序遍历：规则是若树为空，则空操作返回，否则从根结点开始（注意并不是先访问根结点），中序遍历根结点的左子树，然后是访问根结点，最后中序遍历右子树。遍历的顺序为：GDHBAE-ICF。  3）后序遍历：规则是若树为空，则空操作返回，否则从左到右先叶子后结点的方式遍历访问左右子树，最后是访问根结点。遍历的顺序为：GHDBIEFCA。  4）层序遍历：规则是若树为空，则空操作返回，否则从树的第一层，也就是根结点开始访问，从上而下逐层遍历，在同一层中，按从左到右的顺序对结点逐个访问。遍历的顺序为：ABCDEFGHI。 |
| 本周练习过的代码（例） |
| *代码1：拼接*  name=input(**"please enter your username"**) new\_str1=**"hello"**+name print(new\_str1)  *代码2：占位符%s的使用*  name=input(**"请输入您的名字:"**) age=int(input(**"请输入您的年龄："**)) day=int(input(**"请输入您加入的天数："**)) print(**"hello,%s,your age is %d,you have been the vip for %d days"**%(name,age,day))  *代码3：format()*  name=input(**"请输入您的名字:"**) age=int(input(**"请输入您的年龄："**)) day=int(input(**"请输入您加入的天数："**)) print(**"hello,{},your age is {},you have been the vip for {} days"**.format(name,age,day))  *代码4：字符串内置方法*  new\_str=**" welcome to xf1433.com.xf1433"** print(new\_str.find(**"xf"**)) *#找所要寻找的字符串的位置* print(new\_str.find(**"xf"**,13)) *#从13位开始，找所要寻找的字符串的位置* print(new\_str.count(**"com"**)) *#查找字符串出现了几次* print(new\_str.replace(**"xf"**,**"xiaofeng"**,1)) *#加1代表只替换一个，不写默认全部替换* print(new\_str.split(**"."**,1)) *#加1代表只切割一次，不写默认全切* print(new\_str.startswith(**"welcome"**)) *#判断是否以该字符串开头* print(new\_str.upper()) *#全部转换为大写，小写为lower* print(new\_str.strip()) *#去掉开头空格*    代码5：KMP算法的实现  def find(s,t):  p=0  while p<=len(s)-len(t):  i=p  j=0  while j<len(t):  if s[i]!=t[j]:break  i+=1  j+=1  else:  return p  p+=1  else:  return -1 print(find(**'1234567'**,**'345'**)) print(find(**'1234567'**,**'567'**))    代码6：KMP算法的优化  def get\_next(t):  a=[0]\*len(t)  a[0]=-1  k=-1  j=0  while j<len(t)-1:  if k==-1 or t[j]==t[k]:  j+=1  k+=1  if t[j]!=t[k]:  a[j]=k  else:  a[j]=a[k]  else:  k=a[k]  return a  def kmp(s,t):  aa=get\_next(t)  i=0  j=0  while i<len(s) and j<len(t):  if s[i]==t[j]:  i+=1  j+=1  else:  j=aa[j]  if j<0:  i+=1  j=0  if j==len(t):return i-len(t)  return -1 print(get\_next(**'12345'**)) print(get\_next(**'11111'**)) print(get\_next(**'121212'**)) print(kmp(**'aaaaaaabxxxy'**,**'aaab'**)) print(kmp(**'aaaaaaabxxxy'**,**'aaac'**)) print(kmp(**'kaaaaaaabxxxy'**,**'aaa'**)) print(kmp(**'aaaabaaac'**,**'aaac'**))    代码7：树的各种遍历  class Node(object):  def \_\_init\_\_(self,item):  self.elem=item  self.lchild=None  self.rchild=None class Tree(object):  def \_\_init\_\_(self):  self.root=None  def add(self,item):  node=Node(item)  if self.root is None:  self.root=node  return  queue=[self.root]  while queue:  cur\_node=queue.pop(0)  if cur\_node.lchild is None:  cur\_node.lchild=node  return  else:  queue.append(cur\_node.lchild)  if cur\_node.rchild is None:  cur\_node.rchild=node  return  else:  queue.append(cur\_node.rchild)  def breadth\_travel(self): *#广度遍历* if self.root is None:  return  queue=[self.root]  while queue:  cur\_node=queue.pop(0)  print(cur\_node.elem,end=**""**)  if cur\_node.lchild is not None:  queue.append(cur\_node.lchild)  if cur\_node.rchild is not None:  queue.append(cur\_node.rchild)  def preorder(self,node): *#先序遍历* if node is None:  return  print(node.elem,end=**""**)  self.preorder(node.lchild)  self.preorder(node.rchild)  def inorder(self,node): *#中序遍历* if node is None:  return  self.inorder(node.lchild)  print(node.elem, end=**""**)  self.inorder(node.rchild)   def postorder(self, node): *# 后序遍历* if node is None:  return  self.postorder(node.lchild)  self.postorder(node.rchild)  print(node.elem, end=**""**) if \_\_name\_\_==**"\_\_main\_\_"**:  tree=Tree()  tree.add(0)  tree.add(1)  tree.add(2)  tree.add(3)  tree.add(4)  tree.add(5)  tree.add(6)  tree.add(7)  tree.add(8)  tree.add(9)  tree.breadth\_travel()  print(**""**)  tree.preorder(tree.root)  print(**""**)  tree.inorder(tree.root)  print(**""**)  tree.postorder(tree.root)  print(**""**) |
| 本周练习过的代码 |
|  |

1. 该表信息将会作为你报名申请的重要依据，请认真仔细填写。
2. 培训班有严格的制度，请认真阅读规则并结合自身情况填写该表。
3. 完成该表填写后以“xxx学习情况”命名，及时上传作业。

最后希望大家能够加入我们，一起努力，共同进步！