第四章 串

2022年9月30日 星期五 14:06

串的定义和实现【2023年考点不考这个，了解即可】

定义	• 串：字符串简称串，是由零个或多个字符组成的有限序列 • 子串：串中任意多个连续的字符组成的子序列 • 主串：包含子串的串 • 某个字符在串中的序号称为该字符在串中的位置 • 空格串：由一个或多个空格组成的串

定长顺序存储表示	`#define MAXLEN 255`
	`typedef struct`
	`{`
	`  char ch[MAXLEN];`
	`  int length;`
	`} SString;`

堆分配存储表示	`typedef struct`
	`{`
	`  char *ch;`
	`  int length;`
	`} SString;`

块链存储表示	• 类似于线性表的链式存储结构 • 也可采用链表方式存储串值其中节点称为块 • 最后一个节点占不满时，用#填充
基本操作	• StrAssign(&T,chars):赋值操作 • StrCopy(&T,S):赋值操作 • StrEmpty(S):判空操作 • StrCompare(S,T):比较操作 • StrLength(S):求串长 • StrString(&Sub,S,pos,len):求子串 • Concat(&T,S1,S2):串联接 • Index(S,T):定位操作 • ClearString(&S):清空操作 • DestoyString(&S):销毁串

串的模式匹配

- 模式匹配：子串的定位操作，求的是子串（模式串）在主串中的位置

	解释说明	代码	例题【后续补充】

暴力匹配算法	• 时间复杂度为O(nm)
	• m为模式串的长度，n为串的长度
		`int Index(SString S, SString T)`
		`{`
		`  int i = 1, i = 1;`
		`  while (i <= S.length && j <= T.length)`
		`  {`
		`  if (S.ch[i] == T.ch[j])`
		`  {`
		`  ++i;`
		`  ++j;`
		`  }`
		`  else`
		`  {`
		`  i = i - j + 2;`
		`  j = 1;`
		`  }`
		`  }`
		`  if (j > T.length)`
		`  return i - T.length;`
		`  else`
		`  return 0;`
		`}`	

模式匹配算法-KMP算法	字符串的前缀、后缀和部分匹配值
	前缀：除最后一个字符外，字符串的所有头部子串
	后缀：除第一个字符外，字符串的所有尾部子串
	部分匹配值：字符串的前缀和后缀的最长想定前后缀长度
	移动位数：子串需要向后移动的位数
	值 = 已匹配字符数-对应部分匹配值（首尾重合）
	KMP算法的原理
	• next[j]=PM[j-1]
	• PM[j-1]是对应的部分匹配值
	• 在子串的第j个字符与主串发生失配时
	• 则跳到子串的next[j]位置重新与主串当前位置进行比较
	时间复杂度为O(m+n)
		`void get_next(String T, int next[])`
		`{`
		`  int i = 1, j = 0;`
		`  next[1] = 0;`
		`  while (i < T.length)`
		`  {`
		`  if (j == 0 || T.ch[i] == T.ch[j])`
		`  {`
		`  ++i;`
		`  ++j;`
		`  next[i] = j;`
		`  }`
		`  else`
		`  j = next[j];`
		`  }`
		`}`
		`int Index_KMP(String S, String T, int next[])`
		`{`
		`  int i = 1, j = 1;`
		`  while (i <= S.length && j <= T.length)`
		`  {`
		`  if (j == 0 || S.ch[i] == T.ch[j])`
		`  {`
		`  ++i;`
		`  ++j;`
		`  }`
		`  else`
		`  j = next[j];`
		`  }`
		`  if (j > T.length)`
		`  return i - T.length;`
		`  else`
		`  return 0;`
		`}`	

KMP算法的进一步优化	无	`void get_nextval(String T, int next[])`
		`{`
		`  int i = 1, j = 0;`
		`  nextval[1] = 0;`
		`  while (i < T.length)`
		`  {`
		`  if (j == 0 || T.ch[i] == T.ch[j])`
		`  {`
		`  ++i;`
		`  ++j;`
		`  if (T.ch[i] != T.ch[j])`
		`  nextval[i] = j;`
		`  else`
		`  nextval[i] = nextval[j]`
		`  }`
		`  else`
		`  j = nextval[j];`
		`  }`
		`}`
		`int Index_KMP(String S, String T, int next[])`
		`{`
		`  int i = 1, j = 1;`
		`  while (i <= S.length && j <= T.length)`
		`  {`
		`  if (j == 0 || S.ch[i] == T.ch[j])`
		`  {`
		`  ++i;`
		`  ++j;`
		`  }`
		`  else`
		`  j = next[j];`
		`  }`
		`  if (j > T.length)`
		`  return i - T.length;`
		`  else`
		`  return 0;`
		`}`	