串

字符串。ASCII码为基本数据所形成的一维线性结构。

|  |
| --- |
| int num = 3;  scanf(“%d”, &num); //这种做法是有巨大漏洞的。 |
| char s[1024];  int num;  gets(s);  //对s的内容进行详尽的判断！  num = atoi(s); |

串是一个线性结构：

串的存储形式：

typedef struct STRING{

CHARACTER \*head;

int length;

} STRING;

struct CHARACTER{

char ch[32];

struct CHARACTER \*next;

};

串匹配算法：

在string中定位substring第一次出现的位置；若substring不存在，则返回NULL；

朴素的串匹配算法：

int search(const char \*str, const char \*subStr){

“ababcabcababcdababcdeabc”

“babcde”

int strLen = strlen(str);

int subStrLen = strlen(subStr);

int i;

int j;

for(i = 0; i <= strlen - subStrLen; i++){

for(j = 0; j < subStrLen; j++){

if(str[i+j] != subStr[j]) break;

}判断subStrLen是否比较完成

}

}

next数组:表示对应下标的字符失配之后，应该使用该数组的哪一个下标的字符进行比较

aadaadaaaadaa

下标 0 1 2 3 4 5 6 7 8 9 10 11 12

子串 a a d a a d a a a a d a a

next数组 0 0 1 0 1 2 3 4 5 2 2 3 4

1、next数组的前两个元素的值一定为0；

2、用当前串元素ch的前一个元素chpre和chcmp = next[Ichpre]比较，若chcmp == chpre，则next[Ich] = next[Ichcmppre] + 1；

下标 0 1 2 3 4 5 6 7 8 9 10 11 12

chcmp chpre ch

chcmppre

子串substr a a d a a d a a a a d a a

next数组 0 0 1

3、若chpre != chcmp，则chcmppre = chcmp, chcmp = substr[next[Ichcmp]]，若chpre != chcmp，重复上述操作，若chcmp == chpre， 则next[Ich] = next[Ichcmppre]+1，若全部比较完都不相等，则为next[Ichpre] = 0。

下标 0 1 2 3 4 5 6 7 8 9 10 11 12

chcmp chpre ch

chcpppre

子串substr a a d a a d a a a a d a a

next数组 0 0 1

由于chcmp != chpre;

则 chcmp = substr[next[1]] = a;

下标 0 1 2 3 4 5 6 7 8 9 10 11 12

chcmp chpre ch

chcmppre

子串substr a a d a a d a a a a d a a

next数组 0 0 1 0

由于chcmp != chpre;

则 next[Ich] = 0;

下标 0 1 2 3 4 5 6 7 8 9 10 11 12

chcmp chpre ch

chcmppre

子串 a a d a a d a a a a d a a

next数组 0 0 1 0 1 2 3 4 5 2

由于chcmp != chpre;

则 chcmp = substr[next[2]] = d;

下标 0 1 2 3 4 5 6 7 8 9 10 11 12

chcmp chcmppre chpre ch

子串 a a d a a d a a a a d a a

next数组 0 0 1 0 1 2 3 4 5 2

由于chcmp != chpre;

则 chcmp = substr[next[1]] = a;

下标 0 1 2 3 4 5 6 7 8 9 10 11 12

chcmp chcmppre chpre ch

子串 a a d a a d a a a a d a a

next数组 0 0 1 0 1 2 3 4 5 2

chcmp == chpre;

next[Ich] = next[Ichcmppre]+1;

KMP算法过程如下：

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22

a a d a a d a a d a a d a a a a d a a d a a d

a a d a a d a a a a d a a

0 0 1 0 1 2 3 4 5 2 2 3 4

当前失配点为8，因此用下标为8的substr对应的next数组的元素也就是5作为下标，与当前失配点进行比较，相当于将子串下标为5的元素与原串下标为8的元素对齐：

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22

a a d a a d a a d a a d a a a a d a a d a a d

a a d a a d a a a a d a a

0 0 1 0 1 2 3 4 5 2 2 3 4

当前失配点为11，因此用下标为11的substr对应的next数组的元素也就是5作为下标，与当前失配点进行比较，相当于将子串下标为5的元素与原串下标为11的元素对齐：

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22

a a d a a d a a d a a d a a a a d a a d a a d

a a d a a d a a a a d a a

0 0 1 0 1 2 3 4 5 2 2 3 4

找到子串第一次出现的位置。

下标 0 1 2 3 4 5 6 7 8 9 10 11 12

子串 a a d a a d a a a a d a a

next数组 0 0 1 0 1 2 3 4 5 2 2 3 4

i: 2/3/4/5/6/7/8/9/10/11/12

j: 0/1/0/1/2/3/4/5/2/1/2/1/2/3/4

|  |
| --- |
| void getNext(const char \*str, int \*next){  int i = 0;  int j = 0;  boolean flag = TRUE;  next[0] = next[1] = 0;//next数组的前两个元素为0  for(i = 2; str[i]; i++){  for(flag = TRUE; flag;){  if(str[i-1] == str[j]){  next[i] = ++j;  flag = FALSE;  }  else if(j == 0){  next[i] = 0;  flag = FALSE;  }  else{  j = next[j];  }  }  }  } |

|  |
| --- |
| i = j = 0;  while(strLen -i + j >= subLen){  while(subStr[j] && str[i] == subStr[j]){  i++; j++;  }  if(subStr[j] == 0){  free(next);  return i - subLen;  }  else if(j == 0){  i++;  j = 0;  }  else{  j = next[j];  }  } |

KMP算法时间复杂度：O(m+n)，其中n表示原串长度，m表示字串长度。