第三章

6. 简述以下算法的功能（栈st和队列qu的元素类型均为ElemType）。

答：先将所有元素出队，之后判断该元素是否为非第i个元素，是则重新入队，不是则不做任何操作，进行下一步循环。

即将qu队列中从front开始的第i个元素出队，其余元素仍按原顺序排列。

8. 环形队列一定优于非环形队列吗？什么情况下使用非环形队列？

答：不一定。虽然环形队列解决了非环形队列假溢出的问题，但是环形队列有覆盖原来数据的风险。如果我们需要使用原来入队的数据时，环形队列就会出现问题，这时我们必须使用非环形队列。

12. 设计一个算法，将一个环形队列（容量为n，元素下标从0到n-1）的元素倒置。例如，图3.2（a）中为倒置前的队列（n=10），图3.2（b）中为倒置后的队列。

答：代码如下

void Reverse(queue<char>&qu) {

stack<char>st;

char temp;

int n = qu.size();

for (int i = 0;i < n;i++) {

temp= qu.front();

qu.pop();

st.push(temp);

}

for (int i = 0;i < n;i++) {

temp =st.top();

st.pop();

qu.push(temp);

}

}

第四章

1. 串是一种特殊的线性表，请从存储和运算两方面分析它的特殊之处。

答：

存储方面：串中每个元素是单个字符，在设计串存储结构时需要每一个单元只存储一个字符的元素。

运算方面：串拥有连接、判断串是否相等、获取子串和子串替换等基本运算，而线性表并没有这些基本运算。

2. 为什么模式匹配中，BF算法是有回溯算法，而KMP算法是无回溯算法？

答：

设目标串为s，模式串为t。

使用BF算法时，当t[j]=s[i]时， i++，j++；当t[j]≠s[i]时， i=i-j+1，j=0。从中看到，一旦两字符不等，目标串指针i会回退，所以BF算法是有回溯算法。

使用KMP算法时，当t[j]=s[i]时， i++，j++；当t[j]≠s[i]时，i不变， j=next[j]。从中看到，目标串指针i不会回退，只会保持位置不变或者向前推进，所以KMP算法是无回溯算法。

3. 在KMP算法中，计算模式串的next时，当j=0时，为什么要置next[0]=-1？

答：t0字符与目标串中某字符si比较不相等时，此时置next[0]=-1表示模式串中已没有字符可与目标串的si比较，目标串当前指针i应后移至下一个字符，再和模式串的t0字符进行比较。

4. KMP算法是简单模式匹配算法的改进，以目标串s="aabaaabc"、模式串t="aaabc"为例说明的next的作用。

答：next数组的作用是用来让模式串回退到第next[j]位和i所指字符进行比较,其保存了一定的数据关系，使得对比效率增高。

以本题为例，next数组如下表所示。

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| j | 0 | 1 | 2 | 3 | 4 |
| t[j] | *a* | *a* | *a* | *b* | *c* |
| next[j] | -1 | 0 | 1 | 2 | 0 |

i=0，j=0开始，当两者对应字符相等时，i++，j++，直到i=2，j=2时对应字符不相等。如果是简单模式匹配，下次从i=1，j=0开始比较。

i=2，j=1的字符比较不相等，保持i=2不变，取j=next[j]=0。

i=2，j=0的字符比较不相等，保持i=2不变，取j=next[j]=-1。

当j=-1时i++、j++，则i=3，j=0，对应的字符均相等，一直比较到j超界，此时表示匹配成功，返回3。

5. 给出以下模式串的next值和nextval值：

（1）ababaa

（2）abaabaab

答：（1）求其next和nextval值如表所示。

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| j | 0 | 1 | 2 | 3 | 4 | 5 |
| t[j] | *a* | *b* | *a* | *b* | *a* | *a* |
| next[j] | -1 | 0 | 0 | 1 | 2 | 3 |
| nextval[j] | -1 | 0 | -1 | 0 | -1 | 3 |

（2）求其next和nextval值如表所示。

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| j | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| t[j] | *a* | *b* | *a* | *a* | *b* | *a* | *a* | *b* |
| next[j] | -1 | 0 | 0 | 1 | 1 | 2 | 3 | 4 |
| nextval[j] | -1 | 0 | -1 | 1 | 0 | -1 | 1 | 0 |

6. 设目标为s="abcaabbabcabaacbacba"，模式串t="abcabaa"。

（1）计算模式串t的nextval数组。

（2）不写算法，给出利用改进的KMP算法进行模式匹配的过程。

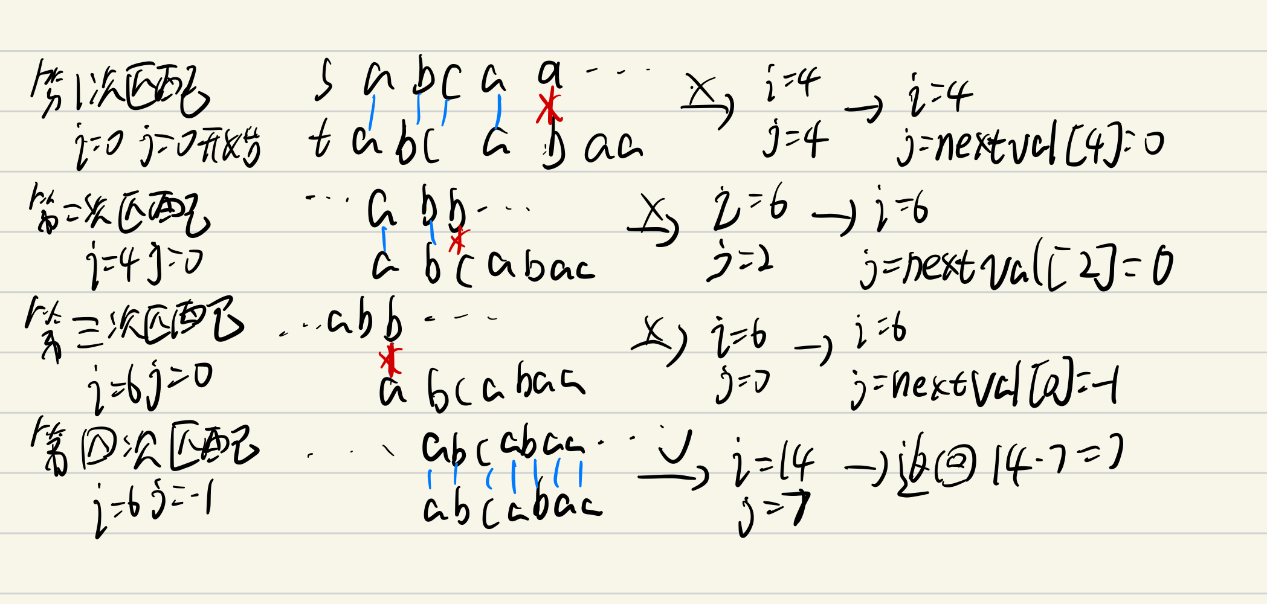
（3）问总共进行了多少次字符比较？

答：

1. 先计算next数组，在此基础上求nextval数组，如表所示。

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| j | 0 | 1 | 2 | 3 | 4 | 5 | 6 |
| t[j] | *a* | *b* | *c* | *a* | *b* | *a* | *a* |
| next[j] | -1 | 0 | 0 | 0 | 1 | 2 | 1 |
| nextval[j] | -1 | 0 | 0 | -1 | 0 | 2 | 1 |

1. 改进后KMP过程如下



（3）从上述匹配过程看出：第1趟到第4趟的字符比较次数分别是5、3、1、7，所以总共进行了16次字符比较。