数据结构实验报告4

学号：2022114467 姓名：林泽宇 专业：软件工程

知识范畴：串 完成日期：2023年 月 日

实验题目：基于KMP算法的子串查找与替换

|  |  |  |  |
| --- | --- | --- | --- |
| 课程目标1（60%） | 课程目标2（40%） | 得分(100分)  期末成绩占比5% | 批阅人签字 |
| 程序代码及测试 | 写作、排版、代码注释等 |  |
|  |  |  |

实验内容及要求：

从键盘输入主串s以及子串t1和t2。编写程序，将主串s中所有t1子串替换为t2子串，输出替换后得到的串以及t1被替换的次数。要求子串查找采用KMP算法。

实验目的：掌握KMP算法。

数据结构设计简要描述：

typedef struct

{

    char \*idx;

    int len;

} str;

算法设计简要描述：

采用KMP算法匹配模式串和匹配串, 通过字符串的按长度删除字符和在指定位置插入字符等操作实现替换.

输入/输出设计简要描述：

根据输入提示, 分别从键盘输入主串s, 模式串t1, 替换串t2.

编程语言说明：

使用C++编程。 主要代码采用C语言实现 ；动态存储分配采用C++的new和delete操作符实现；输入与输出采用printf()和scanf()操作；程序注释采用C/C++规范。

主要函数说明：

//初始化字符串

void initStr(str &s)

//用字符数组给字符串赋值

void assignStr(str &s, const char \*chars)

//从键盘输入给字符串赋值

void inputStr(str &s)

//从pos起删除s中len个字符

void trimStr(str &s, int pos, int len)

//在s的pos处插入t1

void insertStr(str &s, int pos, const str &t)

//遍历字符串

void traverseStr(const str &s)

//获取next数组

void getNext(const str &s, int nextval[])

//KMP算法匹配字符串

int kmp(str &s, const str &t1, const str &t2)

程序测试简要报告：

文本, 信件

描述已自动生成

文本, 信件

描述已自动生成

结论: 代码可以运行得到正确结果

源程序代码：

#include <iostream>

#include <cstring>

using namespace std;

typedef struct

{

    char \*idx;

    int len;

} str;

//初始化字符串

void initStr(str &s)

{

    s.idx = nullptr;

    s.len = 0;

}

//用字符数组给字符串赋值

void assignStr(str &s, const char \*chars)

{

    initStr(s);

    s.len = strlen(chars);

    s.idx = new char[s.len + 1];

    strcpy(s.idx, chars);

}

//从键盘输入给字符串赋值

void inputStr(str &s)

{

    printf("请输入字符串(不支持空格):\n");

    char chars[100];

    scanf("%s", chars);

    assignStr(s, chars);

}

//从pos起删除s中len个字符

void trimStr(str &s, int pos, int len)

{

    if (pos < 0)

    {

        printf("pos can't be nagetive!\n");

        exit(-1);

    }

    if (pos + len > s.len)

    {

        printf("Downflow!\n");

        exit(-1);

    }

    char \*tmp = s.idx;

    s.idx = new char[s.len - len + 1];

    for (int i = 0; i < pos; ++i)

        s.idx[i] = tmp[i];

    for (int i = pos; i < s.len - len; ++i)

        s.idx[i] = tmp[i + len];

    s.len -= len;

    delete[] tmp;

    tmp = nullptr;

}

//在s的pos处插入t1

void insertStr(str &s, int pos, const str &t)

{

    if (pos < 0)

    {

        printf("pos can't be nagetive!\n");

        exit(-1);

    }

    char \*tmp = s.idx;

    s.idx = new char[s.len + t.len + 1];

    for (int i = 0; i < pos; ++i)

        s.idx[i] = tmp[i];

    for (int i = pos; i < pos + t.len; ++i)

        s.idx[i] = t.idx[i - pos];

    for (int i = pos + t.len; i < s.len + t.len; ++i)

        s.idx[i] = tmp[i - t.len];

    s.len += t.len;

    delete[] tmp;

    tmp = nullptr;

}

//遍历字符串

void traverseStr(const str &s)

{

    printf("Traverse Str:\n");

    if (!s.len)

        return (void)printf("The str is empty.\n");

    for (int i = 0; i < s.len; ++i)

        printf("%c", s.idx[i]);

    printf("\n");

}

//获取next数组

void getNext(const str &s, int next[])

{

    int j = 0, k = -1;

    next[0] = -1;

    while (s.idx[j])

    {

        if (k == -1 || s.idx[j] == s.idx[k])

        {

            ++j, ++k;

            next[j] = k;

        }

        else

            k = next[k];

    }

}

// KMP算法匹配字符串

int kmp(str &s, const str &t1, const str &t2)

{

    int n = s.len, nextval[t1.len], i = 0, j = 0, cnt = 0;

    getNext(t1, nextval); //获取nextval数组

    while (i < s.len)

    {

        if (j == -1 || s.idx[i] == t1.idx[j])

            ++i, ++j;

        else

            j = nextval[j];

        if (j == t1.len)

        {

            j = 0, ++cnt;

            i -= t1.len;

            trimStr(s, i, t1.len); //删除t1

            insertStr(s, i, t2);   //插入t2, 完成替换

            i += t2.len;

        }

    }

    return cnt;

}

int main()

{

    str s, t1, t2;

    inputStr(s), inputStr(t1), inputStr(t2);

    traverseStr(s);

    printf("本次kmp一共进行了%d次匹配\n", kmp(s, t1, t2));

    printf("匹配替换后的主串为:\n");

    traverseStr(s);

    printf("\n");

    return 0;

}