1. 用BF算法编写一程序，从键盘输入主串和子串，如果匹配则返回主串中匹配的位置，否则返回无法匹配的消息。
2. 用Word 文档保存源程序，并在源程序中注释该算法。
3. 把运行结果截图显示在文档中。
4. 附BF算法，自己编完整的程序

#include <iostream>

#include <cstring>

using namespace std;

int BF\_Search(char\* s, char\* p) {

//传入主串和子串，获取各自长度信息

int s\_len = strlen(s);

int p\_len = strlen(p);

//在主串中遍历每一个可能的起始位置，直到最后一个可能的位置为止。

//由于子串长度不超过主串长度，因此可以保证这些起始位置是合法的

for (int i = 0; i <= s\_len - p\_len; i++) {

//在每个起始位置上，使用循环遍历子串的每一个字符，并比较它与主串对应位置的字符是否相等。

//如果在某个位置发现不匹配的字符，那么就退出循环。

int j;

for (j = 0; j < p\_len; j++) {

if (s[i + j] != p[j]) {

break;

}

}

//如果在子串的每个位置上都找到了与主串对应位置相等的字符，则表示子串匹配成功，返回当前起始位置 i。

if (j == p\_len) {

return i;

}

}

//如果在主串中没有找到匹配的子串，则返回 -1，表示匹配失败。

return -1;

}

int main() {

char s[100], p[100];

cout << "请输入主串: ";

cin >> s;

//忽略输入流中的回车符

cin.ignore();

cout << "请输入子串: ";

cin.getline(p, 100);

int pos = BF\_Search(s, p);

if (pos == -1) {

cout << "无法匹配" << endl;

}

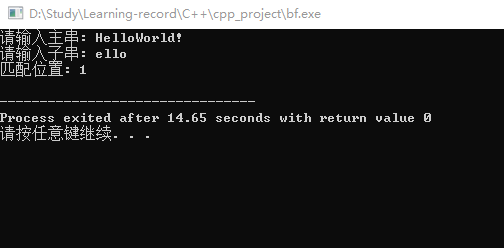
else {

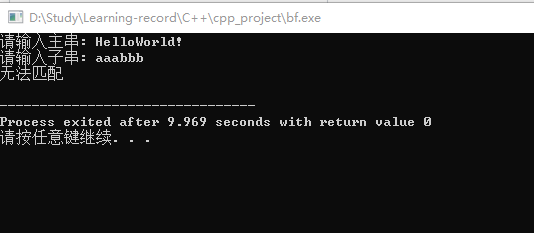
cout << "匹配位置: " << pos << endl;

}

return 0;

}





1. 用KMP算法完成上述目标（源程序加详细注释+运行结果截图）。

#include <iostream>

#include <cstring>

#include <vector>

using namespace std;

//GetNext 函数是一个计算子串 next 数组的函数，它的参数包括一个指向子串的指针 p 和一个存储计算结果的 vector 对象 next。

void GetNext(char\* p, vector<int>& next) {

//首先计算子串的长度，并将 next 数组的大小设置为子串的长度。

int len = strlen(p);

next.resize(len);

//接下来，定义两个变量 i 和 j，并将 j 的初值设置为 -1。

int i = 0, j = -1;

//同时，将 next[0] 的值设置为 -1，因为当 j 为 -1 时，我们需要将 i 向右移动一位，即 i++。

next[0] = -1;

//接下来，我们进入一个 while 循环，该循环的终止条件是 i 的值小于子串的长度减 1。

while (i < len - 1) {

//在每次循环中，我们首先判断 j 的值是否为 -1 或者当前字符和子串的第 j 个字符相等。

if (j == -1 || p[i] == p[j]) {

//如果是的话，就将 i 和 j 同时向右移动一位，即 i++ 和 j++，并将 next[i] 的值设置为 j

i++;

j++;

next[i] = j;

}else {

//否则，将 j 的值设为 next[j]，继续进行下一次匹配。

//最终，next 数组中存储的就是子串的 next 数组。

j = next[j];

}

}

}

//KMP\_Search 函数的参数包括一个指向主串的指针 s 和一个指向子串的指针 p，它的返回值是匹配的位置，如果找不到匹配则返回 -1。

int KMP\_Search(char\* s, char\* p) {

//定义一个 vector 对象 next，并调用 GetNext 函数计算出子串的 next 数组。

vector<int> next;

GetNext(p, next);

//定义两个变量 i 和 j，并将它们的初值都设置为 0。同时，我们计算出主串和子串的长度。

int i = 0, j = 0;

int s\_len = strlen(s);

int p\_len = strlen(p);

//接下来，我们进入一个 while 循环，该循环的终止条件是 i 的值小于主串的长度且 j 的值小于子串的长度。

while (i < s\_len && j < p\_len) {

//在每次循环中，我们首先判断 j 的值是否为 -1 或者当前字符和子串的第 j 个字符相等。

if (j == -1 || s[i] == p[j]) {

//如果是的话，就将 i 和 j 同时向右移动一位，即 i++ 和 j++。

i++;

j++;

}else {

//否则，将 j 的值设为 next[j]，继续进行下一次匹配。

j = next[j];

}

}

//判断 j 的值是否等于子串的长度

if (j == p\_len) {

//如果是的话，就返回当前匹配的位置 i-j

return i - p\_len;

}else {

//否则，说明没有匹配成功，就返回 -1。

return -1;

}

}

int main() {

char s[100], p[100];

cout << "请输入主串: ";

cin >> s;

cout << "请输入子串: ";

cin >> p;

int pos = KMP\_Search(s, p);

if (pos == -1) {

cout << "无法匹配" << endl;

}

else {

cout << "匹配位置: " << pos << endl;

}

return 0;

}

