BWT基因位置检索软件设计
实验报告

班级：1503401
学号：1150340111
姓名：吴宛真

目录

一、 软件需求分析

1.1 项目背景：

序列比对法是生物信息学中一种基本的信息处理方法。它对发现生物序列中的功能，结构和进化的信息具有非常重要的意义。序列比对算法就是运用某种特定的算法，将测序仪测出的序列片段比对到已知生物序列的参考序列中，根据比对结果可以进一步判断序列间相似性关系以及它的生物学特征，从而对生物的生长状态作出预测。伴随着测序技术的发展，各个基因组中的基因数量与长度都在持续增长。采用传统的线性比对方法，时间复杂度为o(mn)级别（m为字串长度，n为参考基因串长度），对处理大规模基因串非常不合适，所以对参考基因串的压缩和索引就显得尤为重要。

1.2 实现意义：

采用BWT对原串进行轮转变换后，将非精确比对转换为精确比对。利用后缀数组或者FW-index的方法，使得每个针对参考基因组的子串的匹配在每次比对中只需要进行一次。算法的时间复杂度为o(m)级别，在长子串与庞大基因库比对中，大大缩短了匹配的时间。

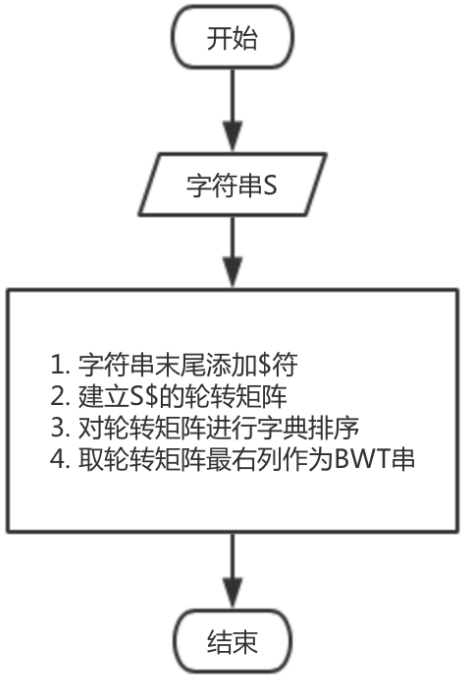
二、 BWT检索原理

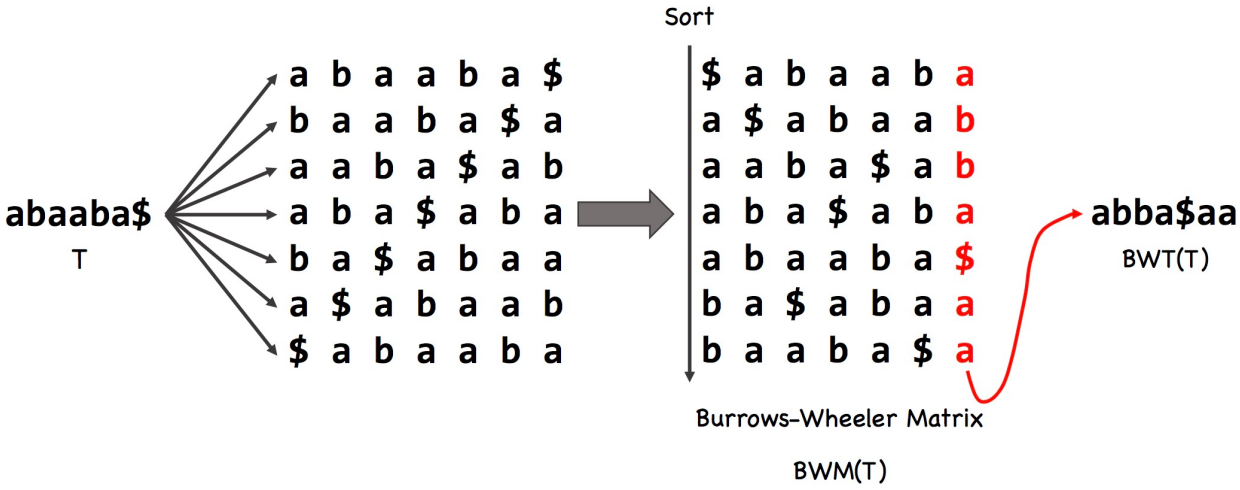
2.1 原理概要介绍

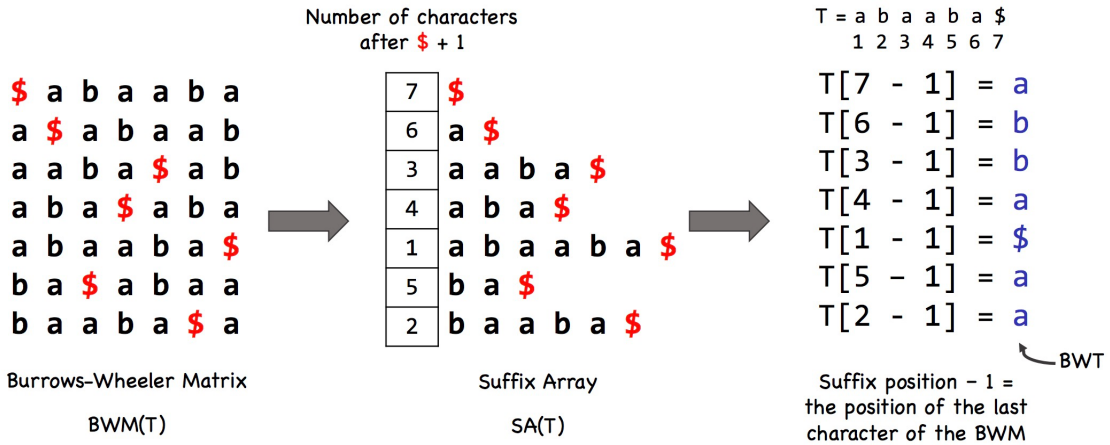
BWT是The Burrows-Wheeler Transform的缩写，是对参考基因组的变换方式，也就是对参考基因组进行了一次有规律的重新排序，变换的目的就是为了方便后续进行查找。

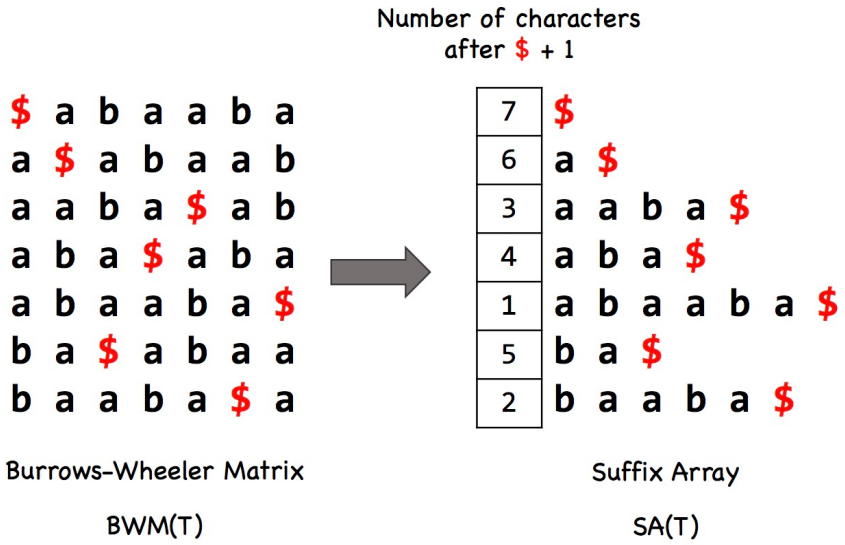
这种算法的核心思想是对字符串轮转后得到的字符矩阵进行排序和变换，下面通过 几个示意图进行说明:

2.2 BWT串生成流程图


开始
字符串S
1. 字符串末尾添加$符
2. 建立S$的轮转矩阵
3. 对轮转矩阵进行字典排序
4. 取轮转矩阵最右列作为BWT串
结束


Sort
abaaba$
T
a b a a b a $
b a a b a $ a
a a b a $ a b
a b a $ a b a
b a $ a b a a
a $ a b a a b
$ a b a a b a
$ a b a a b a
a $ a b a a b
a a b a $ a b
a b a $ a b a
a b a a b a $
b a $ a b a a
b a a b a $ a
abba$aa
BWT(T)
Burrows-Wheeler Matrix
BWM(T)


$ a b a a b a
a $ a b a a b
a a b a $ a b
a b a $ a b a
a b a a b a $
b a $ a b a a
b a a b a $ a
Burrows-Wheeler Matrix
BWM(T)
Number of characters after $ + 1
7 $
6 a $
3 a a b a $
4 a b a $
1 a b a a b a $
5 b a $
2 b a a b a $
Suffix Array
SA(T)
T = a b a a b a $
1 2 3 4 5 6 7
T[7 - 1] = a
T[6 - 1] = b
T[3 - 1] = b
T[4 - 1] = a
T[1 - 1] = $
T[5 - 1] = a
T[2 - 1] = a
BWT
Suffix position - 1 = the position of the last character of the BWM



生成后缀数组SA[]，可根据SA可直接将T转化为BWT串

2.1 BWT串生成

(1)首先，BWT先对需要转换的文本块，进行循环右移，每次循环一位。可以知道长度为n的文本块，循环n次后重复，这样就得到看n个长度为n的字符串。如下图中的“Rotate Right”列。（其中‘#’作为标识符，不在文本块的字符集中，这样保证n个循环移位后的字符串均布相同。并且定义'#'小于字符集中的任意字符）。

(2)对循环移位后的n个字符串按照字典序排序。如下图中的“Sorted (M)”列。

(3)记录下“Sorted (M)”列中每个字符串的最后一个字符，组成了“L”列。(其中"F"列是“Sorted (M)”列中每个字符串的前缀)

BWT（T）重要性质(理解LF-mapping，FM-index关键)：
在Burrows-Wheeler Matrices中，最后一列第i个出现的字符x，与第一列中第i个出现的x是同一个。第一列其实就是T的一个字典排序。

2.2BWT(T)的还原

BWT（T）可还原出T，原理为Last First (LF)Mapping，使用算法为UNPERMUTE algorithm。

LF-Mapping，关键为将最后一列的某个字符x映射到第一列中，看x在第一列中的位置。只需要知道，字典序排在x之前的所有字符的个数，和在最后一列中这个x在所有x中是第几个。因此定义两个参数C[c],和Occ[c,r]。

c代表某字符，r代表矩阵的行位置。

C[c]表示字典序小于字符c的所有字符个数。

Occ[c,r]表示在BWT（T）中第r行之前出现字符c的个数。从第0行开始算。

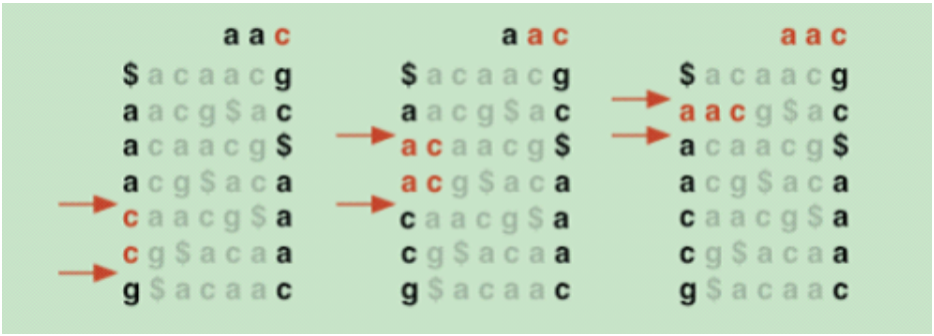
图示与算法

```
// C[c] 字典序小于字符c的所有字符个数
// cc[c,r] BWT (T) 中第r行之前出现字符c的个数
LF(r,c)
    return C[c] + Occ(c,r) + 1
    // 返回最右列位置r元素映射到第一列的位置


UnPremute() // LF mapping的算法
    r <- 1
    T = ""
    while BWT[r] != $ do
        T <- predend BWT[r] to T
        r <- LF(r)
    end while
    return T // 返回原字符串T
```

2.3 read匹配 (match)

图示



算法

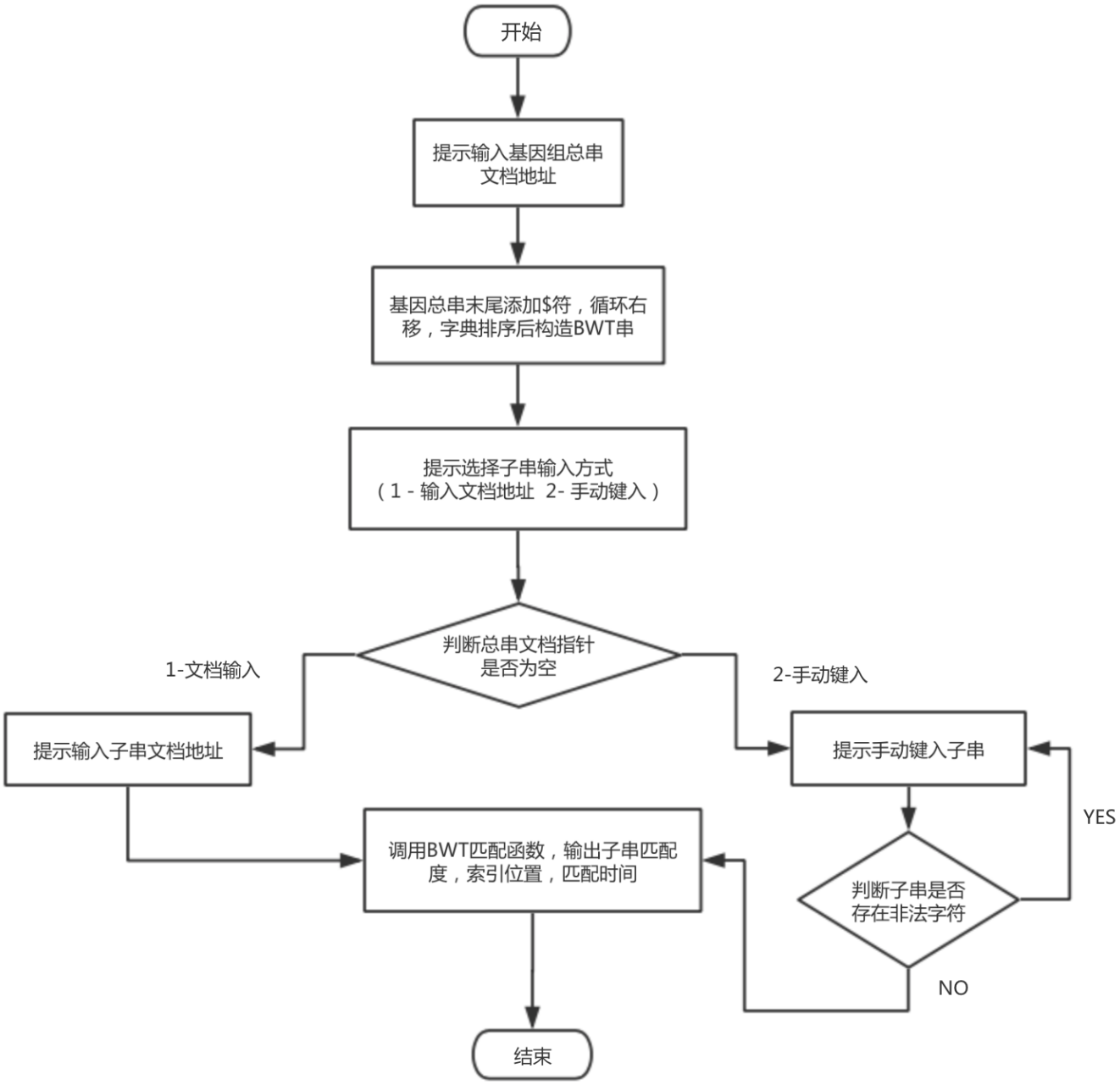
```
Match(P[1,p])  // 子串匹配
    c <- P[p]  // 从子串末尾开始，从后往前匹配
    sp <- C[c] + 1
    ep <- C[c+1] + 1
    i <- p -1
    while sp < ep and i >= 1 do // 不断缩小[sp,ep]可能匹配区间
        c <- P[i]
        sp <- LF(c,sp)
        ep <- LF(c,ep)
        i <- i - 1
    end while // 直到找到read的第一个字符结束
    return sp,ep // [sp,ep]区间为精确匹配区间
```

2.4 read 定位

前面讲述如何查找一个read，但我们需要知道read映射到参考基因组上的位置，只需构造数组SA[r]，记录BWT矩阵第r行在原轮转矩阵中的位置。

三、 软件架构

3.1 软件设计流程图



四、 功能模块设计

```
void FileInRefe(string file); //文件输入总串模块
void FileInSub(string file);  //文件输入子串模块
void HandInRefe(); // 手动输入总串模块
void HandInSub(); //手动输入子串模块
void MainMenu(); //主菜单模块
void loopMove(char *str, int steps);// 循环位移模块
void BWTprint(); // BWT串输出模块
void ExactMatch(); // 精确匹配模块
void Unpremute(); // BWT串还原为原始串模块
int LF(int r); // LF mapping模块 找到BWT串r行字符在首列的位置
int LFC(int r,char c); // LFC模块
void FormC(); // C[c]构造模块 字典序小于c的所有字符个数
int GetOcc(char c,int r){ // Occ构造模块 BWT串r行前字符c出现的次数
```

4.1 - 主菜单模块

4.2 - 总串文件输入模块

4.3 - 子串文件输入模块

4.4 - 总串手动输入模块

4.5 - 子串手动输入模块

4.6 - 循环位移模块

4.7 - BWT串输出模块

4.8 - BWT串还原模块

4.9 - LF mapping模块

4.10 - LFC模块

4.11 - C[c]构造模块

4.12 - GetOcc模块

4.13 - 精确匹配模块

五、 测试结果

文件输入测试结果:

生成BWT串 还原为原始字符串 对子串进行匹配

```
请选择总串输入方式 (1 - 默认位置文件读入 2 - 输入文件路径 其他 - 手动输入)
1
请选择子串输入方式 (1 - 默认位置文件读入 2 - 输入文件路径 其他 - 手动输入)
1
正在生成BWT矩阵...
所用时间 : 0.274
BWT(T) :
tcaaaaaaacgaaacaataacatataaacaaacgtccaagataaagatgaagactcatagcgttgtaaagttcacgtgggcgggccaggc
atatacgccgggtattgaaaatgtagataaatgaaaaaaagcgccgaatgctacaggtcaggccctgtaaagctgagaaaagaggtcccc
caaagactaccacacccgcgatccacgctccggccagagatccgggataaggaccccatacacccatcagacaagtgaccttccgctaac
aaattgttgctaggacgccgttttcgcggttaggtggcagaatagcaaaataaaaaaggccctaacgtttttatttctcagaagtgcata
ccgcggactgattgcgcattctttaggctccctccattgaaagacctacagtaactcgtcggacccgtggactcagctccaggaccctga
acggaagacatctgccaccatcacgactatccctgtgcctgacagcgaccccgtctatcccacagacccctcacgtcacgcccgctag
agaccacccgcatcagaatttataacccccttctcgcccccctctttccctccctgtggcgtttcggtaacagttacttactattctgg
```

```
------------------------------
原始字符串为:
gcaggcggatcacgaggttgggagatcgagaccatcctggctaacggtgaaaccccgtctctactgaaaaatacaaaaaaa
tactcgggaggctgaggaaggagaatggcgtgaacctgggcggtggagcttgcagtgagctgagatcacgccactgcact
aaaaaaaaggcctcccctgcttgccacaggtctccccaaggcgcactggcctcatcttgggcctgtgttatctcctaggt
aacagttcctgcatgggcggcatgaaccggaggcccatcctcaccatcatcacactggaagactccaggtcaggagccac
cttgcctctgacccctgggcccacctcttaccgatttcttccatactactacccatccacctctcatcacatccccggcg
ggctttgggacctcttaacctgtggcttctcctccacctacctggagctggagcttaggctccagaaaggacaagggtgg
aggacctgatttccttactgcctcttgcttctcttttcctatcctgagtagtggtaatctactgggacggaacagctttg
agaggaagagaatctccgcaagaaaaggggagcctcaccacgagctgccccccagggagcactaagcgaggtaagcaagcag
tcagattcacttttatcacctttccttgcctctttcctagcactgcccaacaacaccagctcctctccccagccaaagaa
```

```
------------------------------
子串aaccccgtctctactgaaaaatacaaaaaaaaattagccgggcgtggtgctgggcacctgtagtcccagctactcgggag
匹配成功
匹配区间为 199 - 200
子串所在位置为 50
------------------------------
子串cctcccctgcttgccacaggtctccccaaggcgcactggcctcatcttgg
匹配成功
匹配区间为 2717 - 2718
子串所在位置为 250
------------------------------
子串ggcaggctcttccccatcacccactgctctcttggagccagtgcttgaag
匹配成功
匹配区间为 4605 - 4606
子串所在位置为 6900
------------------------------
子串caccg
匹配成功
匹配区间为 1853 - 1856
子串所在位置为 1766
子串所在位置为 4347
子串所在位置为 5323
------------------------------
子串gccccaaacccggcgcgctgctctccccccaaaaaa
匹配失败
在位置2242失配
匹配率为 :21.6216%
------------------------------
```

手动输入测试结果

```
请选择总串输入方式  (1 - 默认位置文件读入 2 - 输入文件路径 其他 - 手动输入)
3
请输入总串并以$结束
accacacactactgaca$
请选择子串输入方式  (1 - 默认位置文件读入 2 - 输入文件路径 其他 - 手动输入)
3
请输入子串并以$结束
cac$
正在生成BWT矩阵...
所用时间 : 0
BWT(T) :
acgcc$ctacaaaaatcc
----------------------------
原始字符串为:
accacacactactgaca
----------------------------
匹配成功
匹配区间为 9 - 12
子串所在位置为 2
子串所在位置为 4
子串所在位置为 6
----------------------------
```

六、 源代码附录

```
#include <iostream>
#include<fstream>
#include<string.h>
#include<map>
#include<stdlib.h>
#include<assert.h>
#include<stdio.h>
#include<windows.h>
#include <time.h>
#define N 10000
using namespace std;
char Refer[N]; // refenence 总串
char BWT[N];
map<char,int>C;
map<string,int>SA;
char SubQue[N]; // subquence 子串
char BWTMatrix[N][N]; //BWT矩阵
char T[N];
```

```
string S[N]; // 子串
int SubQueNum=0;

void FileInRefe(string file){ //文件输入总串模块
    ifstream infile;
    infile.open(file.data());
    assert(infile.is_open());
    char c;
    int i=0;
    while (!infile.eof()){
        infile>>c;
        Refer[i]=c;
        i++;
    }
    infile.close();
    Refer[i]='$';
    Refer[i+1]='\0';
}


void FileInSub(string file){ //文件输入子串模块
    ifstream infile;
    infile.open(file.data());
    assert(infile.is_open());
    string s;
    while(getline(infile,s)){
        S[SubQueNum]=s;
        SubQueNum++;
    }
    infile.close();
}
void HandInRefe(){ // 手动输入总串模块
    cout<<"请输入总串并以$结束"<<endl;
    char c;
    int i=0;
    do{
        cin>>c;
        if(c!='\n') Refer[i]=c;
        i++;
    }while(c!='$');
    Refer[i]='\0';
}
void HandInSub(){ //手动输入子串模块
    cout<<"请输入子串并以$结束"<<endl;
    char c;
    int i=0;
    do{
```

```
        cin>>c;
        if(c!='\n') SubQue[i]=c;
        i++;
    }while(c!='$');
    SubQue[i-1]='\0';
    SubQueNum = 1;
    S[0] = SubQue;

}

void MainMenu(){ //主菜单模块
    char mode;
    string adr_refe = "C:\\Users\\Chaser\\Documents\\Tencent Files\
\1340338392\\FileRecv\\res.txt";//默认总串文件地址
    string adr_sub = "C:\\Users\\Chaser\\Documents\\Tencent Files\
\1340338392\\FileRecv\\read.txt";//默认子串文件地址
    cout<<"请选择总串输入方式  (1 - 默认位置文件读入 2 - 输入文件路径 其他 - 手动
输入)"<<endl;
    // 选择总串输入方式 1-从默认位置读入文件 2-从给定位置读入文件 其他-手动输入
    cin>>mode;
    if (mode == '1'){
        FileInRefe(adr_refe);
    }else if (mode == '2'){
        cout<<"请输入总串文件路径 : "<<endl;
        cin>>adr_refe;
        FileInRefe(adr_refe);
    }else{
        HandInRefe();
    }

    cout<<"请选择子串输入方式  (1 - 默认位置文件读入 2 - 输入文件路径 其他 - 手动
输入)"<<endl;
    // 选择子串输入方式 1-从默认位置读入文件 2-从给定位置读入文件 其他-手动输入
    cin>>mode;
    if (mode == '1'){
        FileInSub(adr_sub);
    }else if (mode == '2'){
        cout<<"请输入子串文件路径 : "<<endl;
        cin>>adr_sub;
        FileInSub(adr_sub);
    }else{
        HandInSub();
    }
}

void loopMove(char *str, int steps)// 把str串后长度step的字符移动到前面
```

```
{
    char temp[N];
    char*p;
    int n = strlen(str) - steps;
    p = str + n;
    memcpy(temp, p, steps);
    memcpy(temp + steps,str, n);
    memcpy(str, temp, strlen(str));
}
int compare(const void *a, const void *b)
{
    return strcmp((char *)a, (char *)b);
}

void BWTprint(){ // 输出BWT串
    cout<<"BWT(T) :"<<endl;
    Sleep(500);
    for( int i=0;i<strlen(Refer);i++){
        BWT[i]=BWTMatrix[i][strlen(Refer)-1];
        cout<<BWTMatrix[i][strlen(Refer)-1];
    }
    cout<<endl;
    BWT[strlen(Refer)]='\0';
}

int GetOcc(char c,int r){ //Occ：BWT串r行前字符c出现的次数
    if(r==0){
        return 0;
    }else{
        int cnt=0;
        for(int i=0;i<r;i++){
            if(BWT[i]==c) cnt++; // 遍历BWT串 对出现的c进行计数
        }
        return cnt;
    }
}

void FormC(){ // 构造C[c] 字典序小于c的所有字符个数
    C['a'] = C['c'] = C['g'] = C['t'] = 0;//初始化
    for(int i = 0;i < strlen(Refer);i++){
        char c = BWT[i];
        if (c =='a'){
            C['c']++;  C['g']++;  C['t']++;
        }else if (c == 'c'){
            C['g']++;  C['t']++;
        }else if (c == 'g'){
            C['t']++;
```

```
        }
    }
}

int LF(int r){ //LF mapping 找到BWT串r行字符在首列的位置
    int c=BWT[r];
    return GetOcc(c,r)+C[c]+1;
}

int LFC(int r,char c){ // 匹配时调用
    return C[c]+GetOcc(c,r)+1;
}

void Unpremute(){ // 由BWT串还原为原始串
    int len=strlen(Refer);
    for(int i=0;i<len;i++){
        T[i]='a';
    }
    T[len]='\0';
    int r=0;
    int p=int(strlen(Refer))-1;
    while(BWT[r]!='$'){ //遇到$说明循环结束
        T[p]=BWT[r];
        r=LF(r); //通过LF映射函数进行还原
        p=p-1;
    }
    cout<<"原始字符串为: "<<endl;
    Sleep(500);
    for(int i=1;i<len;i++){
        cout<<T[i];
    }
    cout<<endl;
}

void ExactMatch(){ // 精确匹配
    int p = strlen(SubQue) - 1; //从子串最后一位字符开始遍历
    char c = SubQue[p];
    int sp = C[c] + 1;
    int num = 0;
    for(int i = 0;i < strlen(Refer);i++){
        if(Refer[i] == c) num++;
    }
    int ep = C[c] + num + 1;
    int i = p-1;
    while(sp < ep && i>=0){ //根据算法逐渐缩小sp ep 匹配区间
        c = SubQue[i];
        sp = LFC(sp,c);
```

```
        ep = LFC(ep,c);
        i = i-1;
    }
    if((sp < ep)){ //没有越界
        cout<<"匹配成功 "<<endl; //匹配成功
        cout<<"匹配区间为 "<<sp<<" - "<<ep<<""<<endl; //输出匹配区间
        for(int i = sp;i < ep;i++){
            string current = BWTMatrix[i]; //对区间范围的每行进行遍历 通过SA找到其在原始总串中的位置
            cout<<"子串所在位置为 "<<(strlen(Refer) - SA.find(current)->second)%(strlen(Refer))<<endl;
        }
    }
    else{
        cout<<"匹配失败"<<endl;
        cout<<"在位置"<<sp-1<<"失配"<<endl;
        cout<<"匹配率为 :"<<(1-float((i+1))/strlen(SubQue))*100<<"%"<<endl; //匹配率 = 匹配字符/子串总长度
    }
}

int main(int argc, char const *argv[])
{
    MainMenu();
    clock_t start, finish;
    start = clock();

    int len = strlen(Refer);
    for (int i = 0; i < len; i++) {
        strcpy(BWTMatrix[i], Refer);
        loopMove(Refer, 1);// 对总串进行循环位移
        string str=BWTMatrix[i]; //复制到所在行
        SA[str]=i;// 记录未排序矩阵每行对应的原始位置,
    }
    qsort(&BWTMatrix, len, N, compare);//字典排序

    finish = clock();
    float d1 = (double)(finish - start) / CLOCKS_PER_SEC;
    cout<<"正在生成BWT矩阵..."<<endl<<"所用时间 : "<<d1<<endl;
    BWTprint();
    cout<<"----------------------------"<<endl;
    FormC();
    Unpremute();
    cout<<"----------------------------"<<endl;

    for(int i=0;i<SubQueNum;i++){//对每个子串分别进行匹配
```

```
      S[i].copy(SubQue, S[i].length());
      SubQue[S[i].length()] = '\0';
      cout<<"子串"<<S[i]<<endl;
      ExactMatch();
     cout<<"----------------------------"<<endl;
    }

     return 0;
}
```