哈尔滨工业大学计算机科学与技术学院

实验报告

课程名称：生物信息学

课程类型：专业核心

实验项目名称：大规模基因组序列表示与索引

系统设计与实现

实验题目：大规模基因组序列表示与索引  
系统设计与实现

班级：

学号：

姓名：

|  |  |  |
| --- | --- | --- |
| 设计成绩 | 报告成绩 | 指导老师 |
|  |  |  |

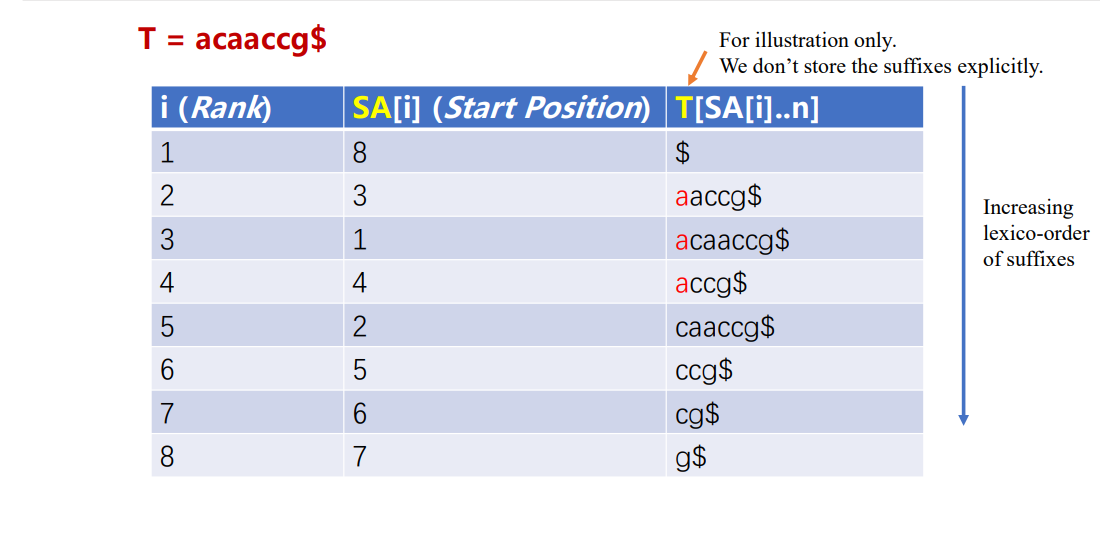
1. 实验目的

为基因组建立索引，为序列查询等操作打下基础

1. 实验原理

采用增量压缩后缀数组的方法

2.1压缩后缀数组



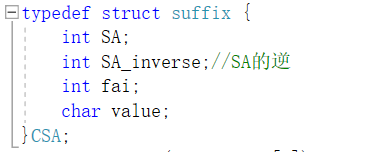
如图，对于一个串T=acaaccg$，那么可将它的每一个后缀（即从每一个字母开始一直到最后一个字母的序列）进行排序。对于一个给定的序列，这个串排到第几个位置只和它从第几个元素开始有关，因而只存储每个串开始的位置即可。如上图，实际上操作是将T[SA[i]…n]（SA[i]为每个后缀的起始位置，T[SA[i]…n]就表示每个后缀串）按字典顺序排序，但表现在程序里，可以将每个T[SA[i]]映射到SA[i]，即保存SA[i]就完全可以知道T[SA[i]]。

但是这样带来了一个问题，串原有的顺序被打乱了，即在这种情况下，通过任意SA[i]位的元素寻找SA[i+1]位的元素都只能从头找起，时间复杂度大大增加。因而引入了Ψ[i]项，Ψ[i]表示根据字典序排在第i位的后缀在原字符串上的下一位所排在的字典序中的位置。即SA[i]的Ψ[i] 就指向SA[i+1]。如下图所示，其中最后一个串的Ψ指向第一个串。

表格

描述已自动生成

写到程序里，可以用结构体来实现



这里我写的比较麻烦，应该可以不存sa\_inverse的，因为sa\_inverse就是排序后的i。

2.2.1 CSA的构建

那么如何将这些后缀串来排序呢

最基本的想法是硬排，用一些比如快排，归并排序的算法，可以将复杂度缩小到O(n\*logn)，但是如果这样的两个串：S1：AAAAAAAC，S2：AAAAAAAT，判断字典序的时候就不可避免的要比较很多次，而且DNA索引的一个特殊性在于，进行的是后缀的比较，即上述S1,S2串比较之后，不可避免的S1的后缀还要和S2及S2的后缀进行比较，而且都要比较到不同字母为止，这样需要比较的次数就更多了。但是其实这些比较大多都是无用而且重复的比较。那么就有了这样的一个想法，可不可以先进行后缀的比较，然后再比较原串呢。

在S1和S2的字典序比较中，先比较最后的一个字符，发现S1的是C，S2的是T，那么suffix1[8]（指的是从第8个元素开始的后缀）排在了suffix2[8]之前，然后进行suffix1[7]与suffix2[7]的比较。看第7个字母，S1[7]==S2[7],那么比较suffix1[8]与suffix2[8]。由于suffix1[8]< suffix2[8],所以suffix1[7]< suffix2[7],那么再比较suffix1[6]与suffix2[6]，由于S1[6]==S2[6]，问题变成了比较suffix1[7]与suffix2[7]，由于已知suffix1[7]< suffix2[7],所以suffix1[6]< suffix2[6]。依次类推，我们可以比较出suffix 1[1]< suffix 2[2]，还顺带比较出了S1和S2的后缀。

**因而或许可以在建立索引的时候，先将后面的字符进行排序，再据此对之前的字符排序。**

可以采用分块的方式，假设需要索引的DNA串一共有M个字符，将N个字符取为一组，那么就有了L=（M/N）+1组，分别命名为到。先将组排好序建好索引，然后再依次向前，合并…组，从而最终实现对整个序列的索引。之后重要的是合并的步骤了，即如何将合到已经排好的Tˊ（即）上。按照上面说的方法，需要借助后缀，先从Ti[N]（即串的最后一个字符）开始，Ti[N]的后缀（记为Si[N]）是Tˊ，因而Ti[N]的Ψ就是Tˊ的字典序。想把Ti[N]插进去，那么就是像上面说的，把Ti[N]和Tˊ中的串对比。先比较第一个字符，再比较第二个字符及其后缀。比较第一个字符就是把Ti[N]的位置映射到Si[N]所在的区域，比较第二个字符及其后缀就是把Tˊ与和Si[N]相同首字母的串的Ψ进行比较。由于之前Tˊ排好序的，因而Tˊ中首字母相同的串，其后缀也是按字段序排列的，因而和Si[N]相同首字母的串的Ψ值呈递增排列。采用二分查找法就能判断出串Ti[N]应该插在Tˊ的什么位置。Ti[N]确定后，Ti[N-1]的Ψ值就是Ti[N]的位置，因而可以用相同方法确定Ti[N-1]的位置。以此类推，就可以将串并入Tˊ串。

写成数学语言：

定义为任意的基因序列，为任意碱基，为基因序列在组的顺序即大小，为在组中与碱基相同区段且小于等于的位置的集合，即，于是有：

特别的，如果发现新的串比任意一个串都小，就排到这些串的最前面。

在插入一个串之后，理论上就可以将这个串之后的所有串位置都更新一下子（位置+1，Ψ值大于该串位置的串Ψ加一），但是这样时间开销会比较大，因此可以将整个串先排完序再插入到原来的串中。但是需要注意的是，后插入的串的位置会不准，因为之前插入的串也会占位，比如第5个插入的串，在它之前有3个串插入的位置在它的前面，因而它在串中的实际位置就应该是order+3。同时Ψ也要相应更新。

所以采取一个很巧妙的办法，有点类似于归并排序的思想。先将排序，然后将和合入一个新串（用s表示）。将Ti[k]在中的顺序（即上文中的order）存入一个数组buffer(其实也可以不存，这里是方便理解)，buffer[N]={1，5，13…156,178…}(举的例子，大概就是表示长成什么样子，实际上不会是这些数)，对于任一个s[i]中的i，如果i不在buffer里，那就将原来的中的串放入s[i]中，否则放入Ti[k]。由于两个串都是升序排列的，因而可以设置两个标记变量来实现这个事情，即flag1用于记录中已经放到了第几位，再放进去的只能是buffer[flag1+1], 同理。

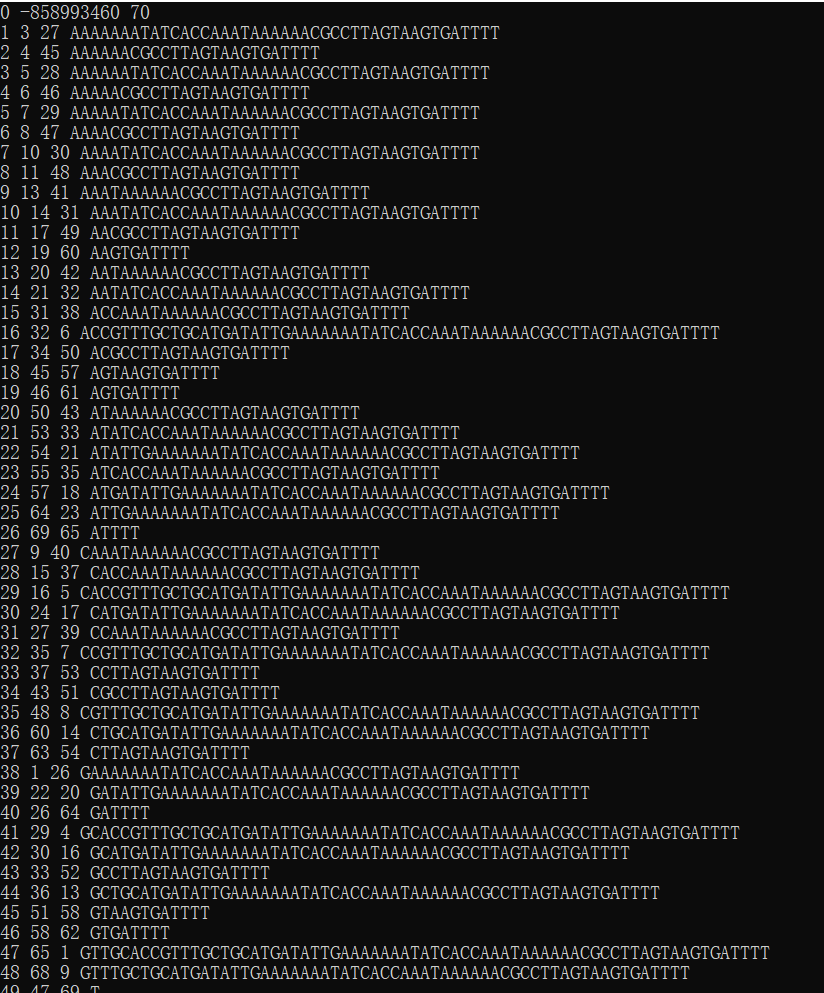
排完之后Ψ值也需要相应的更改，对于中的串，它的下一位后缀只会因为中在它之前位次的串的插入而相应的后移。因而将它的Ψ值放到buffer中进行二分查找（此过程时间复杂度O(\*log)，原Ψ值加上得到的它在buffer中的序数就可。对于中的Ψ，其受影响也是因为在它前面插入的元素，因而直接将它的Ψ加它在中的排序减一即可

2.3一些需要处理的东西

算法中很有可能出现中元素在中排序相同的结果（比如都为lc-1）此时需要先找出这些串并将他们排序，这可能就需要手动排序，而且值得注意的是如果不能在中完全比较出来还需要在中比较。这会导致时间复杂度的一个提升，不过所幸DNA中除了很少数的长串单碱基序列以外还是没有这种情况的。由于在最后归并的过程中，我们先考虑中的串，所以可以将相同元素排到后面（比如有3个串都是第50位，那么将它们排出顺序后分别放到第50，51，52位，这样最后归并的时候也是取完50然后取51，52，然后再从中取串）。这一步骤应该在标记元素位置之后，合并之前进行。

在将串重排之后，由于元素位置发生了改变，导致无法通过SA[i]找到SA[i+1]，这时可以采取一个巧妙的方法，建立从i到SA[i]的映射。将中的串SA位的字典序（order）放到一个新的数组（F）的第SA位中。这样子每当想查询一个串的后缀的位置查询F[SA+1]。

1. 测试结果及分析



如图，是最后一组元素的排序，我们可以看出是非常正确的

表格

中度可信度描述已自动生成

如图是最后生成的文件的样子

1. 经验体会

经过此次实验，我深刻体会了CSA算法的精妙，并通过种种学习方式将它成功弄明白，培养了我的学习能力。此外，此实验中我学会了文件的读写，qsort函数等使用方法。也通过编写程序体会到了一些生物学中的奇妙思想，收获很大。

1. 附录：源代码（带注释）

#define \_CRT\_SECURE\_NO\_WARNINGS

#include<stdio.h>

#include<stdlib.h>

#include <string.h>

#define N 70

typedef struct suffix {

int SA;

int SA\_inverse;//SA的逆

int fai;

char value;

}CSA;

void firstsort(char orign[N]);

int compare(const void\* a, const void\* b);

int compare1(const void\* a, const void\* b);

void calculateACGT();

void refreshACGT(char a[]);

int merge(char a[N],int n);

void move(CSA\* a,CSA\* b);

void printa(char\* fname, CSA arr[], int l);

int findposition(char arr[N],int i, int fai,int oral, int orar, int orcl, int orcr, int orgl, int orgr, int ortl, int ortr);

void checksame(char a[N]);

void newsort(int i, int j, char a[N]);

int F(int j, CSA arr[], int l);

int compare3(int g, int k, char a[N]);

typedef struct suffixarra {

char T[N];

int SA;

int SA\_inverse;//SA的逆

int fai;

char value;

}SA\_T;

CSA dna[5000000];

int totalnum = 0;

int al = 1, ar = 0, cl = 0, cr = 0, gl = 0, gr = 0, tr = 1, tl = 1;

int newnum[5];

int main()

{

FILE\* fp;

int row=0;

char ch;

char line[N + 2];

if (fopen\_s(&fp,"C:\\Users\\hp\\Desktop\\VS CODE\\shengxin1\\NC\_008253.txt","r")!=0) {

printf("Fail to open file!\n");

exit(0); //退出程序（结束程序）

}

if (fgetc(fp) == '>')

{

while (fgetc(fp) != '\n')

{

continue;

}

}

while ((ch = fgetc(fp)) != EOF)

{

if(ch!='\n')

totalnum++;

}

row = totalnum / N;

fseek(fp, -(N+1), SEEK\_CUR);

fgets(line, N, fp);

//printf("%c", ch);

firstsort(line);

for (int m = 2; m<5; m++)

{

fseek(fp, -(2 \* N-1), SEEK\_CUR);

fgets(line, N, fp);

for (int y = 0; y < N; y++)

{

// printf("%c", line[y]);

}

merge(line,m);

//printf("%d \n", m);

}

printa("liuyaojia.txt", dna, totalnum);

fclose(fp);

}

SA\_T suffixarray[N];

int lastseq;

int buffer[N];//存储当前Ti一组的个数排序

CSA bufferc[N];//存储当前被取出来的N个字符

void firstsort(char orign[N])

{

int i;

int t = 1;

for (i = 0; i < N; i++)

{

strcpy\_s(suffixarray[i].T,N, orign);

orign++;

suffixarray[i].SA = i+1;

suffixarray[i].value = orign[i];

}

qsort(suffixarray, N, sizeof(suffixarray[1]), compare);

int SA1[N + 1];

for(int i=0;i<N;i++)

{

suffixarray[i].SA\_inverse = i+1;

}

for (int k = 0; k < N; k++)

{

SA1[suffixarray[k].SA] = k;

if (suffixarray[k].SA == 1)

{

SA1[N] = k;

lastseq = k;

}

}

for (int i = 0; i < N; i++)

{

suffixarray[i].fai = SA1[suffixarray[i].SA + 1];

}

for (i = 0; i < N; i++)

{

//printf("%d ", i);

suffixarray[i].SA\_inverse = i;

//printf("%d ", suffixarray[i].fai);

//printf("%d ", suffixarray[i].SA);//排第i的后缀是从第SA个字母开头的

for (int j = 0; j < N; j++)

{

// printf("%c", suffixarray[i].T[j]);

}

// printf("\n");

}

for (i = 0; i < N; i++)

{

dna[i].fai = suffixarray[i].fai;

dna[i].SA\_inverse = suffixarray[i].SA\_inverse;

dna[i].SA = suffixarray[i].SA;

dna[i].value = suffixarray[i].value;

}

calculateACGT();

}

int compare(const void\* a, const void\* b)

{

SA\_T\* p = (SA\_T\*)a;

SA\_T\* q = (SA\_T\*)b;

return strcmp(p->T, q->T);

}

int compare1(const void\* a, const void\* b)

{

CSA\* p = (CSA\*)a;

CSA\* q = (CSA\*)b;

return p->SA\_inverse - q->SA\_inverse;

}

void calculateACGT()

{

int i = 1;

while (suffixarray[i].T[0] == 'A')

{

ar++;

i++;

}

cl = ar + 1;

cr = cl-1;

while (suffixarray[i].T[0]== 'C')

{

cr++;

i++;

}

gl = cr + 1;

gr = gl-1;

while (suffixarray[i].T[0] == 'G')

{

gr++;

i++;

}

tl = gr + 1;

tr = tl-1;

while (suffixarray[i].T[0] == 'T')

{

tr++;

i++;

}

}

int findposition(char arr[N], int i, int fai, int oral, int orar,int orcl,int orcr,int orgl, int orgr, int ortl, int ortr)

{

int start = 0, end = 0;

int tmp = -1;

int flag=0;

int w1[5] = {0, al,cl,gl,tl};

int w2[5] = { 0,ar,cr,gr,tr };

if (arr[i] == 'A')

{

start = oral;

end = orar;

flag = 1;

}

if (arr[i] == 'C')

{

start = orcl;

end = orcr;

flag = 2;

}

if (arr[i] == 'G')

{

start = orgl;

end = orgr;

flag = 3;

}

if (arr[i] == 'T')

{

start = ortl;

end = ortr;

flag = 4;

}

tmp=binarysearch(fai, dna, start, end);

if (tmp == -1)

{

bufferc[i].SA\_inverse = w1[flag];

}

else {

bufferc[i].SA\_inverse = tmp+w1[flag]-start;

}

bufferc[i].SA =i;

bufferc[i].fai = fai;

return bufferc[i].SA\_inverse;

}

int merge(char a[N],int n)//n表示倒数第几个串

{

int count = 0;

int x\_fai = lastseq;

int p = 0;

int sa[N];

int flag1 = 0, flag2 = 0;

int l = (n - 1) \* N;

int oral = al, orar = ar, orcl = cl, orcr = cr, orgl = gl, orgr = gr, ortr = tr, ortl = tl;

refreshACGT(a);

CSA\* s =(CSA\*) malloc(sizeof(CSA) \* n \* N);

x\_fai = findposition(a, N , lastseq, oral, orar, orcl, orcr, orgl, orgr, ortl, ortr);

x\_fai=findposition(a, N - 1, x\_fai,oral,orar,orcl,orcr,orgl,orgr,ortl,ortr);

for (int d = 2; d< N + 1; d++)

{

x\_fai = findposition(a, N - d, x\_fai, oral, orar, orcl, orcr, orgl, orgr, ortl, ortr);

}

qsort(bufferc, N, sizeof(CSA), compare1);

checksame(a);

//printf("?");

printf("\n");

lastseq =x\_fai;

for (int k = 0; k < N; k++)

{

buffer[k] =bufferc[k].SA\_inverse;

}

for (int m = 0; m < N; m++)

{

sa[bufferc[m].SA] = bufferc[m].SA\_inverse;

}

for (int w = 0; w < N-1; w++)

{

bufferc[w].fai = sa[bufferc[w].SA + 1];

}

for (int r = 0; r < N; r++)

{

//printf("%d %d %d \n", bufferc[r].SA\_inverse, bufferc[r].SA, bufferc[r].fai);

}

for (int k = 0; k < N; k++)

{

printf("%d ", buffer[k]);

}

printf("\n");

s[0].SA = n \* N;

s[0].fai = lastseq;

s[0].SA\_inverse = 0;

s[0].value = '$';

for (int q = 1; q < n \* N; q++)

{

if (q == buffer[flag1])

{

s[q].SA = bufferc[flag1].SA;

s[q].fai = bufferc[flag1].fai+flag1;

s[q].SA\_inverse = bufferc[flag1].SA\_inverse;

s[q].value = bufferc[flag1].value;

flag1++;

printf("!");

}

else {

s[q].SA\_inverse = dna[flag2].SA\_inverse;

s[q].value = dna[flag2].value;

s[q].SA = dna[flag2].SA;

s[q].fai = dna[flag2].fai +binarysearch2(buffer, dna[flag2].fai);

flag2++;

}

//printf("flag1=%d,flag2=%d,s[q].SA=%d,S[q].fai=%d\n", flag1, flag2, s[q].SA, s[q].fai);

}

for (int g = 0; g < n \* N; g++)

{

dna[g].SA = s[g].SA;

dna[g].fai = s[g].fai;

dna[g].SA\_inverse = s[g].SA\_inverse;

dna[g].value = s[g].value;

}

for (int g = 0; g < n \* N; g++)

{

//printf("%d ", s[g].SA\_inverse);

}

free(s);

}

void checksame( char a[N])

{

int q=0;

int i = 0,j=1;

while(i<N)

{

q = bufferc[i].SA\_inverse;

while (bufferc[j].SA\_inverse == q)

{

j++;//j永远指向相同的串的后一个

}

newsort(i, j,a);

i = j;

j = i + 1;

}

qsort(bufferc, N, sizeof(CSA), compare1);

}

void newsort(int i, int j,char a[N])

{

int temp = i;

int num = j - i ;

int a1[N];

int a2[N];

for (int g = i; g < j ; g++)

{

for (int k =i; k < j; k++)

{

if (a1[k-i] == -1)

{

continue;

}

temp=compare3(temp, k,a);//返回大的

}

a1[temp - i] = -1;

a2[temp-i] = num;

num--;

for (int k = i; k < j; k++)

{

if (a1[k-i] == -1)

{

continue;

}

else

{

temp = k;

}

}

}

for (int s = i; s < j; s++)

{

bufferc[s].SA\_inverse = bufferc[s].SA\_inverse + a2[s-i];

}

}

int compare3(int g, int k,char a[N])

{

int point1 = bufferc[g].SA;

int point2 = bufferc[k].SA;

CSA \*r1=NULL;

CSA \*r2=NULL;

char value1=1;

char value2=1;

while (point1 < N && point2 < N)

{

if (a[point1] > a[point2])

{

return g;

}

if (a[point1] < a[point2])

{

return k;

}

if (a[point1] == a[point2])

{

point1++;

point2++;

}

}

return k;

}

void move(CSA\* a, CSA\* b)//b赋值给a

{

a->fai = b->fai;

a->SA = b->SA;

a->SA\_inverse = b->SA\_inverse;

}

int binarysearch(int key, CSA arr[], int start, int end)

{

while (start < end)

{

if (start == end - 1)

{

if (arr[end].fai <= key)

return end + 1;

else if (arr[start].fai > key)

return start;

else if (arr[start].fai <= key && arr[end].fai > key)

return start + 1;

else

return end + 1;

}

int mid = (start + end) / 2;

if (arr[mid].fai <= key)

{

start = mid;

}

else if (arr[mid].fai > key)

{

end = mid;

}

}

}

//计算key值在数组A的位置

int binarysearch2( CSA arr[], int key)

{

int start = 0;

int end = N - 1;

while (start < end)

{

if (start == end - 1)

{

if (arr[end].SA\_inverse <= key)

return end + 1;

else if (arr[start].SA\_inverse > key)

return start;

else if (arr[start].SA\_inverse <= key && arr[end].SA\_inverse > key)

return start + 1;

else

return end + 1;

}

int mid = (start + end) / 2;

if (arr[mid].SA\_inverse <= key)

{

start = mid;

}

else if (arr[mid].SA\_inverse > key)

{

end = mid;

}

}

}

int F(int j, CSA arr[], int l) {

int k = j;

k += binarysearch2(0, l -1, arr, j);

return k;

}

void printa(char\* fname, CSA arr[], int l)

{

printf("开始写入数据\n");

FILE\* R = fopen(fname, "w");

int i = 0;

for (i = 0; i < 10\*N; i++)

{

char ch[100];

\_itoa\_s(i, ch,20, 10);

fputs(ch, R);

fputc('\t', R);

\_itoa\_s(arr[i].SA, ch,20, 10);

fputs(ch, R);

fputc('\t', R);

\_itoa\_s(arr[i].SA\_inverse, ch,20, 10);

fputs(ch, R);

fputc('\t', R);

\_itoa\_s(arr[i].fai, ch, 20, 10);

fputs(ch, R);

fputc('\t', R);

ch[0] = '\0';

ch[1] = '\0';

fputs(ch, R);

fputc('\n', R);

}

fclose(R);

printf("写入数据完毕\n");

}

void refreshACGT(char a[])

{

int counta=0,countc=0,countg=0,countt=0;

newnum[0] = 0;

for (int i = 0; i < N; i++)

{

if (a[i] == 'A')

counta++;

if (a[i] == 'C')

countc++;

if (a[i] == 'G')

countg++;

if (a[i] == 'T')

countc++;

}

ar = ar + counta;

cl = ar + 1;

cr = cr + counta + countc;

gl = cr + 1;

gr = gr + counta + countc + countg;

tl = gr + 1;

tr = gr + counta + countc + countg + countt;

newnum[1] = counta;

newnum[2] = countc;

newnum[3] = countg;

newnum[4] = countt;

}