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：必修

实验项目名称：基于SNAP的基因组测序片段比对与映射系统设计与实现

实验题目：基因组测序片段比对与映射系统设计与实现

班级：

学号：

姓名：

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

1. 实验目的

给定基因组序列（大肠杆菌E. Coli, 长度~5Mbp）作为参考基因组。给定一定数量来自于大肠杆菌的基因组测序片段（共282224条，pair-end，70 bp/read）作为输入。建立算法和系统，可以完成对基因组测序片段的比对，重构片段位置（基因组坐标，正/反链）和基因组片段与参考序列的差异。

1.完整地完成算法所有模块

2.支持编辑距离3以上的片段比对

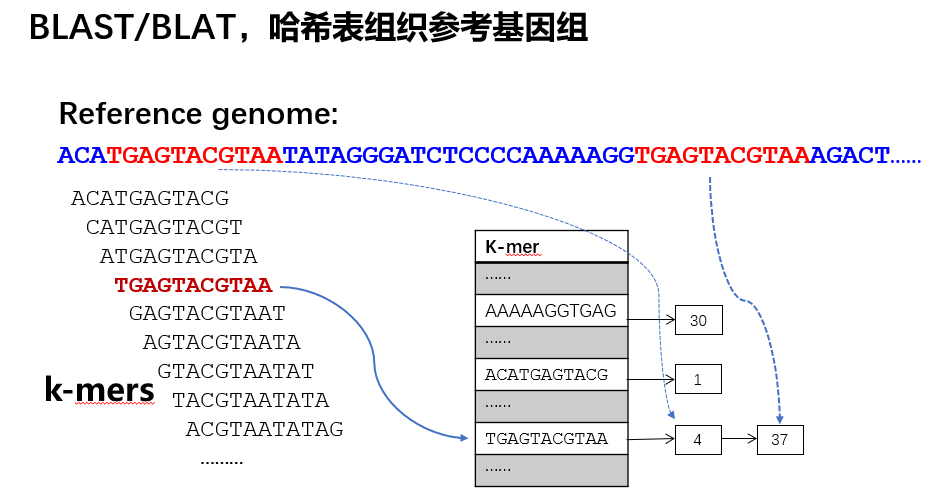
3.正确重构片段基因组位置与局部差异

4.支持SAM格式输出

5.程序运行正确，程序代码优化较好

6.所有文档完整且写作规范

1. 实验原理



SNAP算法是基于一个seed的基因组（或其他参考数据库）短子串的

哈希索引。给定要对齐的读取，SNAP从中绘制多个长度为s（seed长度）的子字符串，并在哈希索引中执行精确查找，以查找数据库中包含相同子字符串的位置。然后计算读取位置和每个候选位置之间的编辑距离，以找到最佳对齐方式。

其中有两个特性使SNAP不同于以前的算法，并提高了它的速度：使用更大的索引来交换内存以减少计算，以及对局部对齐步骤的一组优化，这大大降低了根据候选位置测试读取的成本。

SNAP的索引是一个哈希表，从固定长度的seed字符串到引用数据库中出现这些字符串的位置列表。它在两个方面与BLAST中的参数不同：

1. 更长的seed，SNAP的索引的哈希表的seed的长度典型值为20个碱基对，较长的种子意味着从read中提取的seed与参考数据库精确匹配的可能性较小，但也大大减少了假阳性种子匹配的数量。通过相应概率计算最后将seed的长度设计成为20。

但是在论文中作者是以人类基因组且较大的read长度（100bp）分析的，在该实验中用的是大肠杆菌的基因组而且read长度不是很大（70bp），理论上不需要这么长的seed索引，但是为了能尽可能还原这个算法，我仍然将seed的长度设计为20。



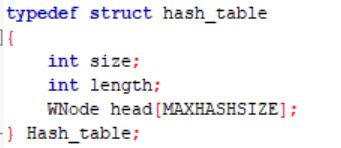
1. 重叠种子：为了节省内存，许多索引只索引参考数据库的非重叠s-mer（例如，前10个碱基，然后是下10个碱基，等等），因此当种子长度为s时，索引中只出现数据库中1/s的位置。这种方法的缺点是每次读取必须尝试更多seed，以找到与索引中使用的步幅匹配的seed。作者从我看不懂的计算机组成原理得出了一些关于查找代价的结论，最终确定了SNAP通过索引重叠的s-mer（即，在数据库上每一次滑动一个长度作为一个seed）来避免这一成本。这需要一个更大的索引，但它仍然在现代服务器的能力范围内，但好像是不在我的计算机的能力范围内，为了能尽可能还原这个算法，我将实验中的hash表索引做了一些改动，使其节约了很多的空间但又不违反seed 20bp长度的索引，虽然这样做会比标准的SNAP索引慢一些，但影响的速度为可以接受的，而且可以在我的计算机运行下去：

若seed大小为20bp，则一个20bp的hash表索引，需要大小420这么大的表头，这很明显很大，于是我这样改进了实验的hash表索引：

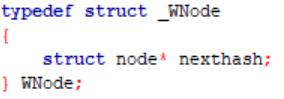
在哈希表中允许碰撞产生，在计算机的能力范围内设置一个很大的质数尽量减少碰撞同时防止占用空间过大。



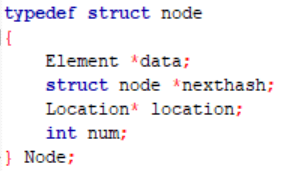
Hash表中有一个大小，长度，WNode类型（表头）的数组head[]。



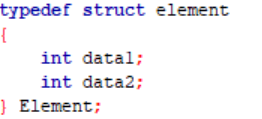
表头只包括一个node类型（节点）的指针nexthash。



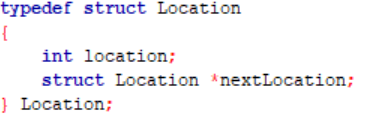
节点包括一个Element类型data（seed数据）的结构体指针，指向同索引的下一个节点的指针nexthash。一个Location类型（位置数据）的指针location，代表该节点位置数量的num。



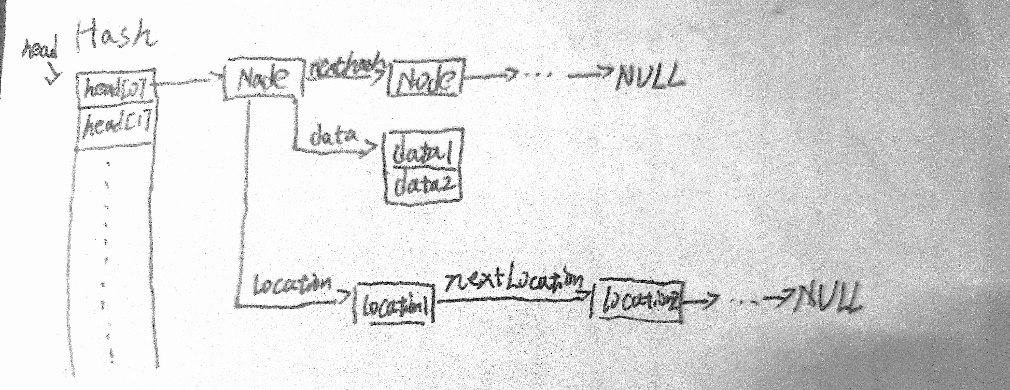
Element结构体用于存储seed的信息，将20bp按A-00,C-01,G-10, T-11的方式编码，并取前30位转化为data1（int类型），后10位转化位data2（int类型）。



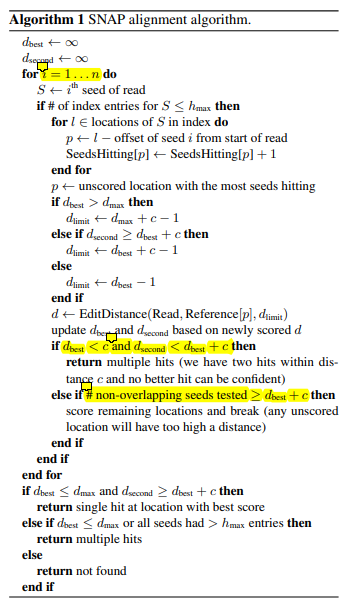
Location结构体用于存储位置信息，包括一个location（int类型）代表该seed在参考基因组的一个位置，Location类型的指针nextLocation指向下一个位置信息。



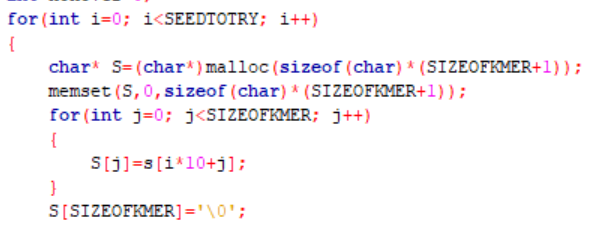
最终hash表结构如下图所示：



SNAP的伪代码：



算法的第三行第四行：尝试n个read的seed从尽可能多的连续非重叠种子开始，然后用s/2、s/4、3s/4等偏移下一组种子。在实验中我在一个read中选取了6个seed，从开始开始，每隔10bp选择一个。

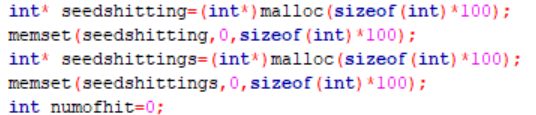


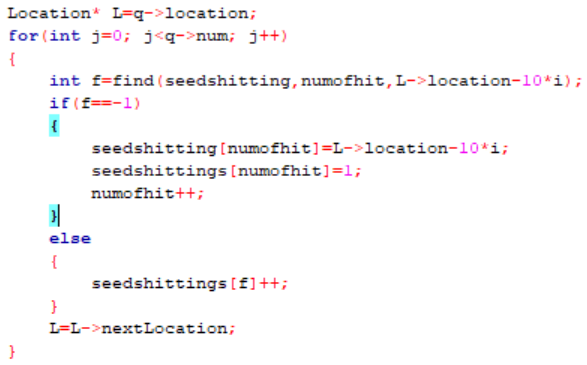
算法的第五行，判断seed的位置数量是否小于等于hmax，在hash表中找到seed对应的节点，节点上的num即seed的位置数量。



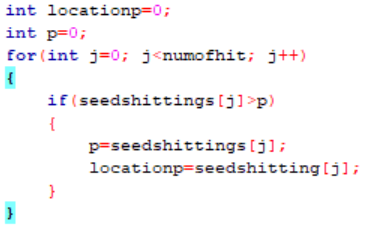


算法第六行第七行第八行，记录不同的位置的数量，将其存储位两个数组，可以减少时间与空间的消耗。其中seedhitting记录位置信息，seedhittings记录对应位置的hit次数信息，每次有新hit都遍历一下seedhitting。若hit的位置不存在，则将其记录在seedhitting中，并设置seedhittings对应的次数为1；否则，直接在对应的次数上加一。

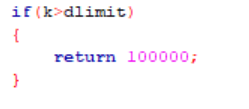




算法第十行，寻找最多hit的位置与次数，遍历seedhittings即可。

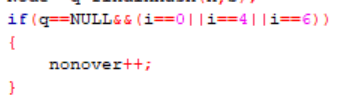


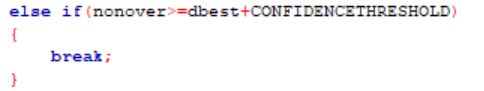
算法第18行，计算编辑距离，实验中采用的是动态规划的方法，不同的是当运算过程中一旦中途结果大于dlomit，立刻跳出。



算法第23行，利用鸽笼原理当hash表无法索引到的非重叠seed大于等于dbest+c时，直接忽略未计算的seed直接退出该条read的计算。



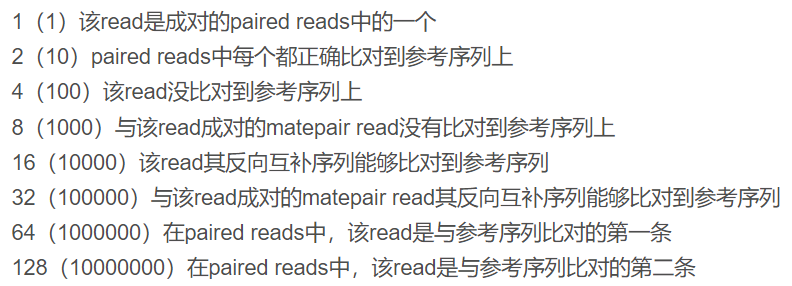




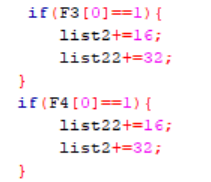
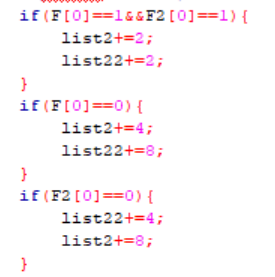
SAM：格式输出

第一列QNAME直接在给的文件中读出。

第二列FLAG，同时对read，pair read以及它们两个的反向互补运行SNAP算法，将符合约束的数组加起来。







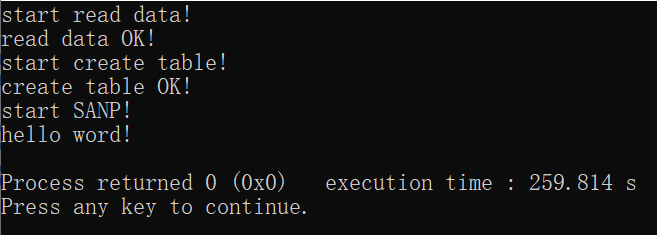
第三列RNAME，read对比的序列名称，由第一个实验的文件给出。

第四列POS，SANP算法输出，从1开始数的read或其反向互补序列对比到对比序列的位置。

第五列MAPQ，质量值代表匹配的可信度，在这次实验中以100-编辑距离代表。

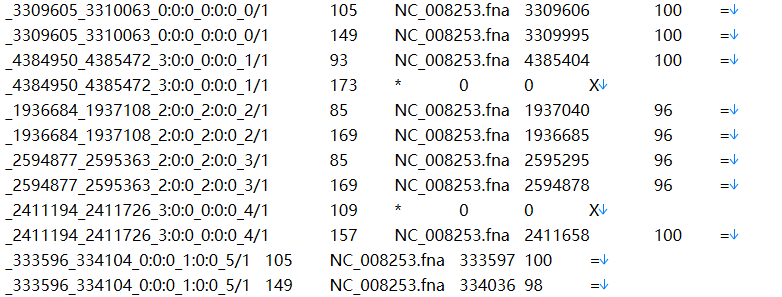
第六列CIGAR，"="表示正确匹配到序列上，"X"表示错误匹配到序列上。

1. 测试结果及分析



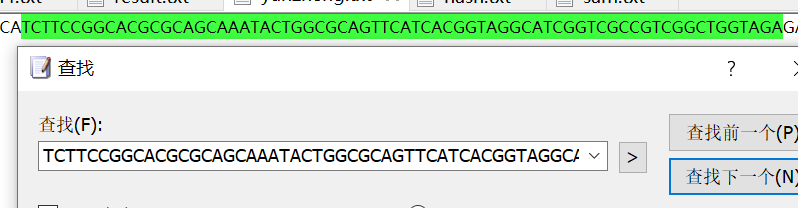
读入对比数据，构建hash表，SANP算法，SAM文件输出一共用时大约250s。

SAM文件的部分输出：



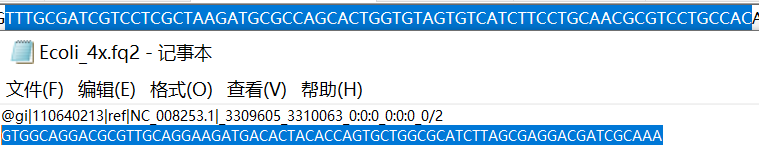
分析一下第一行与第二行，两行read名相同，第一行为read1，第二行为read2，分析两行的FLAG可知105=64(read1)+32(pair read反向互补匹配)+8(pair read不匹配)+1(read) ，149=128(read2)+16(read2 反向互补匹配)+4(read2不匹配)+1(read)。可以得到read1匹配，read2反向互补匹配，匹配到的序列都为NC\_008253.fna。read1匹配的位置为对比序列的3309606位置，编辑距离为0。read2反向互补匹配位置为3309995，编辑距离为0。可以通过搜索验证一下。

Read1:





Read2反向互补：

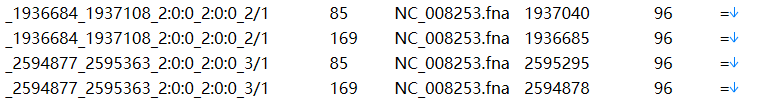




其他的read的情况同理。可以按照上述模板分析。

完整地完成算法所有模块：对比序列读入模块，hash索引构造模块，SANP算法模块，SAM文件写入模块。

支持编辑距离3以上的片段比对，调整参数MAXIMUNDIATANCE的数值，可以找到编辑距离为MAXIMUNDIATANCE之内的比对，如下为一些编辑距离为4的匹配。



正确重构片段基因组位置与局部差异，位置在第4列，局部差异在第五列。

支持SAM格式输出，生成的SAM文件部分如上。

程序运行正确，程序代码优化较好。所有文档完整且写作规范。

1. 经验体会

通过次此实验，使我大致了解了SANP算法怎么回事，还让我学会了怎么构建hash表，还让我了解了SAM文件大体格式是什么。

1. 附录：源代码（带注释）

注释见第二部分实验原理

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <math.h>

#define SIZEOFKMER 20

#define SIZEOFREAD 70

#define MAXHASHSIZE 999983

#define SEEDTOTRY 6

#define MAXIMUNDIATANCE 4

#define CONFIDENCETHRESHOLD 1

#define MAXHITS 10

#define AVERYLARGENUMBER 100000

char \*str;

typedef struct element

{

int data1;

int data2;

} Element;

typedef struct Location

{

int location;

struct Location \*nextLocation;

} Location;

typedef struct node

{

Element \*data;

struct node \*nexthash;

Location\* location;

int num;

} Node;

typedef struct \_WNode

{

struct node\* nexthash;

} WNode;

typedef struct hash\_table

{

int size;

int length;

WNode head[MAXHASHSIZE];

} Hash\_table;

Hash\_table \*h;

Hash\_table\* Creat\_Table()

{

Hash\_table\* h;

h = (Hash\_table\*)malloc(sizeof(Hash\_table));

memset(h, 0, sizeof(Hash\_table));

h->size = MAXHASHSIZE;

h->length = 0;

for(int i=0 ; i<h->size; i++)

{

h->head[i].nexthash = NULL;

}

return h;

}

int CaculateHash(int data1)

{

return data1%MAXHASHSIZE;

}

char \*datatosubdata(char \*data,int start,int end)

{

char\* subdata=(char \*)malloc((sizeof(char))\*(end-start+1+1));

memset(subdata,0,(sizeof(char))\*(end-start+1+1));

for(int i=start; i<=end; i++)

{

subdata[i-start]=data[i];

}

subdata[end-start+1]='\0';

return subdata;

}

char \*subdatatobindata(char \*subdata)

{

int L=strlen(subdata);

char\* bindata=(char \*)malloc(sizeof(char)\*2\*L+1);

memset(bindata,0,sizeof(char)\*2\*L+1);

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

{

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

{

bindata[2\*i]='0';

bindata[2\*i+1]='0';

}

else if(subdata[i]=='C')

{

bindata[2\*i]='0';

bindata[2\*i+1]='1';

}

else if(subdata[i]=='G')

{

bindata[2\*i]='1';

bindata[2\*i+1]='0';

}

else if(subdata[i]=='T')

{

bindata[2\*i]='1';

bindata[2\*i+1]='1';

}

else if(subdata[i]=='N')

{

bindata[2\*i]='N';

bindata[2\*i+1]='N';

}

}

bindata[2\*L]='\0';

return bindata;

}

int bindatatoint(char \*bindata)

{

int num=0;

int L=strlen(bindata);

for(int j=L-1; j>=0; j--)

{

if(bindata[j]=='1')

{

num+=(int)pow(2,L-1-j);

}

}

return num;

}

int chartodata1(char \*data)

{

char\* subdata=datatosubdata(data,0,14);

char\* bindata=subdatatobindata(subdata);

int data1=bindatatoint(bindata);

free(subdata);

free(bindata);

return data1;

}

int chartodata2(char \*data)

{

char\* subdata=datatosubdata(data,15,19);

char\* bindata=subdatatobindata(subdata);

int data2=bindatatoint(bindata);

free(subdata);

free(bindata);

return data2;

}

char \*inttobin(int data,int length)

{

char\* c=(char \*)malloc(sizeof(char)\*(length\*2+1));

memset(c,0,sizeof(char)\*(length\*2+1));

for(int i=0; i<length\*2; i++)

{

if(data/(int)pow(2,2\*length-1-i)==1)

{

data-=pow(2,2\*length-1-i);

c[i]='1';

}

else

{

c[i]='0';

}

}

c[length\*2]='\0';

return c;

}

char\* dattochar(Element\* data)

{

char\* a=inttobin(data->data1,15);

char\* b=inttobin(data->data2,5);

char\* ch=(char\*)malloc(sizeof(char)\*(SIZEOFKMER+1));

memset(ch,0,sizeof(char)\*(SIZEOFKMER+1));

for(int i=0; i<15; i++)

{

if(a[2\*i]=='0'&&a[2\*i+1]=='0')

{

ch[i]='A';

}

else if(a[2\*i]=='0'&&a[2\*i+1]=='1')

{

ch[i]='C';

}

else if(a[2\*i]=='1'&&a[2\*i+1]=='0')

{

ch[i]='G';

}

else if(a[2\*i]=='1'&&a[2\*i+1]=='1')

{

ch[i]='T';

}

}

for(int i=0; i<5; i++)

{

if(b[2\*i]=='0'&&b[2\*i+1]=='0')

{

ch[i+15]='A';

}

else if(b[2\*i]=='0'&&b[2\*i+1]=='1')

{

ch[i+15]='C';

}

else if(b[2\*i]=='1'&&b[2\*i+1]=='0')

{

ch[i+15]='G';

}

else if(b[2\*i]=='1'&&b[2\*i+1]=='1')

{

ch[i+15]='T';

}

}

free(a);

free(b);

ch[SIZEOFKMER]='\0';

return ch;

}

Node \* findinhash(Hash\_table \*h,char\* data)

{

int data1=chartodata1(data);

int data2=chartodata2(data);

int position=CaculateHash(data1);

Node \*q=h->head[position].nexthash;

while(q!=NULL)

{

if(q->data->data2==data2&&q->data->data1==data1)

{

return q;

}

q=q->nexthash;

}

return NULL;

}

void InsertHash(Hash\_table \*h,Element\* k,int location)

{

Node \*q = (Node \*)malloc(sizeof(Node));

memset(q,0,sizeof(Node));

q->data=k;

Location \*L=( Location \*)malloc(sizeof(Location));

memset(L,0,sizeof(Location));

L->location=location;

L->nextLocation=NULL;

q->location=L;

//q->location[0]=location;

q->num=1;

int position;

position = CaculateHash(k->data1);

q->nexthash = h->head[position].nexthash;

h->head[position].nexthash = q;

h->length += 1;

}

void print\_Hash(Hash\_table \*h)

{

printf("HASH:\n");

int i = 0;

for (i = 0; i < h->size ; i++)

{

Node \* p = h->head[i].nexthash;

if(p)

{

printf("hash:%d\t",CaculateHash(p->data->data1));

}

while (p)

{

printf("[value:%d-%d-",p->data->data1,p->data->data2);

Location\* L=p->location;

for(int i=0; i<p->num; i++)

{

printf("%d\t",L->location);

L=L->nextLocation;

}

printf("]");

p = p->nexthash;

if(!p)

{

printf("\n");

}

}

}

printf("HASHOK:\n");

}

void print\_hash(Hash\_table \*h,char \*file)

{

FILE \* r=fopen(file,"w");

char k[100];

int i = 0;

for (i = 0; i < h->size ; i++)

{

Node \* p = h->head[i].nexthash;

if(p)

{

itoa(CaculateHash(p->data->data1),k,10);

fputs(k,r);

fputs("\t",r);

}

while (p)

{

fputs("[",r);

itoa(p->data->data1,k,10);

fputs(k,r);

fputs("\t",r);

itoa(p->data->data2,k,10);

fputs(k,r);

fputs("\t",r);

Location\* L=p->location;

for(int i=0; i<p->num; i++)

{

itoa(L->location,k,10);

fputs(k,r);

fputs("\t",r);

//fputs(L->l,r);

fputs("\t",r);

//printf("%d %c\t",L->location,L->l);

L=L->nextLocation;

}

fputs("]",r);

p = p->nexthash;

if(!p)

{

fputs("\n",r);

}

}

}

fclose(r);

}

char \*turn(char\* str)

{

char\* s=(char\*)malloc(sizeof(char)\*(SIZEOFKMER+1));

memset(s,0,sizeof(char)\*(SIZEOFKMER+1));

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

{

s[i]=str[SIZEOFKMER-i-1];

}

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

{

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

{

s[i]='T';

}

else if(s[i]=='C')

{

s[i]='G';

}

else if(s[i]=='G')

{

s[i]='C';

}

else if(s[i]=='T')

{

s[i]='A';

}

}

s[SIZEOFKMER]='\0';

return s;

}

char \*Turn(char\* str,int L)

{

char\* s=(char\*)malloc(sizeof(char)\*(L+1));

memset(s,0,sizeof(char)\*(L+1));

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

{

s[i]=str[L-i-1];

}

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

{

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

{

s[i]='T';

}

else if(s[i]=='C')

{

s[i]='G';

}

else if(s[i]=='G')

{

s[i]='C';

}

else if(s[i]=='T')

{

s[i]='A';

}

}

s[L]='\0';

return s;

}

void CreateHashTable()

{

int L=strlen(str);

char \*kmeans=(char \*)malloc(sizeof(char)\*(SIZEOFKMER+1));

memset(kmeans,0,sizeof(char)\*(SIZEOFKMER+1));

for(int i=0; i<=L-SIZEOFKMER; i++)

{

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

{

kmeans[j]=str[i+j];

}

kmeans[SIZEOFKMER]='\0';

if(findinhash(h,kmeans)==NULL)

{

Element\* ele=(Element\*)malloc(sizeof(Element));

ele->data1=chartodata1(kmeans);

ele->data2=chartodata2(kmeans);

InsertHash(h,ele,i);

}

else

{

Node\* p=findinhash(h,kmeans);

Location\* L=(Location\*)malloc(sizeof(Location));

memset(L,0,sizeof(Location));

L->location=i;

L->nextLocation=p->location;

p->location=L;

p->num++;

}

}

}

char \*getdata(char\* fname)

{

char s[100];

char \*str=(char \*)malloc(sizeof(char)\*4938921);

memset(str,0,sizeof(char)\*4938921);

FILE \* r=fopen(fname,"r");

while(fgets(s,100,r)!=NULL)

{

if(s[0]!='>')

{

s[strlen(s)-1]='\0';

strcat(str,s);

}

}

fclose(r);

return str;

}

int EditDistance(char\* a,char\*b,int dlimit)

{

int k=0;

int s[strlen(a)+1][strlen(b)+1];

int ss[strlen(a)+1][strlen(b)+1];

for(int i=0; i<strlen(a)+1; i++)

{

for(int j=0; j<strlen(b)+1; j++)

{

s[i][j]=0;

ss[i][j]=0;

}

}

for(int i=1; i<strlen(a)+1; i++)

{

for(int j=1; j<strlen(b)+1; j++)

{

if(a[i-1]==b[j-1])

{

int max=0;

if(s[i-1][j]>max)

{

max=s[i-1][j];

}

if(s[i][j-1]>max)

{

max=s[i][j-1];

}

if(s[i-1][j-1]+1>max)

{

max=s[i-1][j-1]+1;

}

s[i][j]=max;

}

else

{

int max=0;

if(s[i-1][j]>max)

{

max=s[i-1][j];

}

if(s[i][j-1]>max)

{

max=s[i][j-1];

}

s[i][j]=max;

}

if(s[i][j]==s[i][j-1])

{

ss[i][j]=1;

}

else if(s[i][j]==s[i-1][j])

{

ss[i][j]=2;

}

else

{

ss[i][j]=3;

}

}

}

int i=strlen(a);

int j=strlen(b);

while(i!=0&&j!=0)

{

if(ss[i][j]==3)

{

i--;

j--;

}

if(ss[i][j]==1)

{

j--;

k++;

if(k>dlimit)

{

return 100000;

}

}

if(ss[i][j]==2)

{

i--;

k++;

if(k>dlimit)

{

return 100000;

}

}

}

return k;

}

int find(int\* seedshitting,int numofhit,int location)

{

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

{

if(seedshitting[i]==location)

{

return i;

}

}

return -1;

}

int\* sanp(char \*s)

{

static int result[10];

for(int i=0;i<10;i++){

result[i]=0;

}

int dbest=100000;

int dsecond=100000;

int dlimit=0;

int locationdbest=-1;

int locationdsecond=-1;

int flag=0;

int allseedhad=0;

int\* seedshitting=(int\*)malloc(sizeof(int)\*100);

memset(seedshitting,0,sizeof(int)\*100);

int\* seedshittings=(int\*)malloc(sizeof(int)\*100);

memset(seedshittings,0,sizeof(int)\*100);

int numofhit=0;

int nonover=0;

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

{

char\* S=(char\*)malloc(sizeof(char)\*(SIZEOFKMER+1));

memset(S,0,sizeof(char)\*(SIZEOFKMER+1));

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

{

S[j]=s[i\*10+j];

}

S[SIZEOFKMER]='\0';

Node\* q=findinhash(h,S);

if(q==NULL&&(i==0||i==4||i==6))

{

nonover++;

}

if(q==NULL)

{

allseedhad=1;

}

if(q!=NULL&&q->num<=MAXHITS)

{

allseedhad=1;

Location\* L=q->location;

for(int j=0; j<q->num; j++)

{

int f=find(seedshitting,numofhit,L->location-10\*i);

if(f==-1)

{

seedshitting[numofhit]=L->location-10\*i;

seedshittings[numofhit]=1;

numofhit++;

}

else

{

seedshittings[f]++;

}

L=L->nextLocation;

}

int locationp=0;

int p=0;

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

{

if(seedshittings[j]>p)

{

p=seedshittings[j];

locationp=seedshitting[j];

}

}

if(dbest>MAXIMUNDIATANCE)

{

dlimit=MAXIMUNDIATANCE+CONFIDENCETHRESHOLD-1;

}

else if(dsecond>dbest+CONFIDENCETHRESHOLD)

{

dlimit=dbest+CONFIDENCETHRESHOLD-1;

}

else

{

dlimit=dbest-1;

}

char\* Str=(char\*)malloc(sizeof(char)\*(SIZEOFREAD+1));

memset(Str,0,sizeof(char)\*(SIZEOFREAD+1));

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

{

Str[j]=str[locationp+j];

}

Str[SIZEOFREAD]='\0';

int d=EditDistance(s,Str,dlimit);

free(Str);

if(dbest>d&&locationdbest!=locationp)

{

dsecond=dbest;

locationdsecond=locationdbest;

dbest=d;

locationdbest=locationp;

}

else if(dsecond>d&&locationdbest!=locationp)

{

dsecond=d;

locationdsecond=locationp;

}

if(dbest<CONFIDENCETHRESHOLD&&dsecond<dbest+CONFIDENCETHRESHOLD)

{

flag=1;

result[0]=1;

result[1]=locationdbest+1;

result[2]=100-dbest;

free(seedshitting);

free(seedshittings);

return result;

}

else if(nonover>=dbest+CONFIDENCETHRESHOLD)

{

break;

}

}

free(S);

}

if(flag==0)

{

if(dbest<=MAXIMUNDIATANCE&&dsecond>=dbest+CONFIDENCETHRESHOLD)

{

result[0]=1;

result[1]=locationdbest+1;

result[2]=100-dbest;

free(seedshitting);

free(seedshittings);

return result;

}

else if(dbest<=MAXIMUNDIATANCE)

{

result[0]=1;

result[1]=locationdbest+1;

result[2]=100-dbest;

free(seedshitting);

free(seedshittings);

return result;

}

else if(allseedhad==0)

{

result[0]=0;

free(seedshitting);

free(seedshittings);

return result;

}

else

{

result[0]=0;

free(seedshitting);

free(seedshittings);

return result;

}

}

}

void SANP(char \* file1,char\* file2,char\* file3)

{

FILE \* ra=fopen(file1,"r");

FILE \* rb=fopen(file2,"w");

FILE \* rc=fopen(file3,"r");

char s[100];

char s2[100];

char \*rs;

char \*rs2;

char ss[100];

char \*temp;

char \*te=(char\*)malloc(sizeof(char)\*30);

while(fgets(s,100,ra)!=NULL&&fgets(s2,100,rc)!=NULL)

{

if(s[0]=='@')

{

s[strlen(s)-1]='\0';

temp = strtok(s,"|");

int i=0;

while(temp)

{

temp = strtok(NULL,"|");

if(i==3)

{

fputs(temp,rb);

fputs("\t",rb);

strcpy(te,temp);

break;

}

i++;

}

}

if(s[0]!='@'&&s[0]!='2'&&s[0]!='+')

{

int list2=65;

int list22=129;

s[strlen(s)-1]='\0';

s2[strlen(s2)-1]='\0';

rs=Turn(s,70);

rs2=Turn(s2,70);

int \*f;

int \*f2;

int \*f3;

int \*f4;

int F[3];

int F2[3];

int F3[3];

int F4[3];

f=sanp(s);

F[0]=\*(f+0);

F[1]=\*(f+1);

F[2]=\*(f+2);

f2=sanp(s2);

F2[0]=\*(f2+0);

F2[1]=\*(f2+1);

F2[2]=\*(f2+2);

f3=sanp(rs);

F3[0]=\*(f3+0);

F3[1]=\*(f3+1);

F3[2]=\*(f3+2);

f4=sanp(rs2);

F4[0]=\*(f4+0);

F4[1]=\*(f4+1);

F4[2]=\*(f4+2);

//printf("%d\n",F4[1]);

if(F[0]==1&&F2[0]==1){

list2+=2;

list22+=2;

}

if(F[0]==0){

list2+=4;

list22+=8;

}

if(F2[0]==0){

list22+=4;

list2+=8;

}

if(F3[0]==1){

list2+=16;

list22+=32;

}

if(F4[0]==1){

list22+=16;

list2+=32;

}

itoa(list2,ss,10);

fputs(ss,rb);

fputs("\t",rb);

if(F[0]==0&&F3[0]==0){

fputs("\*\t",rb);

fputs("0\t",rb);

fputs("0\t",rb);

fputs("X",rb);

}

else if(F[0]==1){

fputs("NC\_008253.fna\t",rb);

itoa(F[1]+1,ss,10);

fputs(ss,rb);

fputs("\t",rb);

itoa(F[2],ss,10);

fputs(ss,rb);

fputs("\t",rb);

fputs("=",rb);

}

else if(F3[0]==1){

fputs("NC\_008253.fna\t",rb);

itoa(F3[1]+1,ss,10);

fputs(ss,rb);

fputs("\t",rb);

itoa(F3[2],ss,10);

fputs(ss,rb);

fputs("\t",rb);

fputs("=",rb);

}

fputs("\n",rb);

fputs(te,rb);

fputs("\t",rb);

itoa(list22,ss,10);

fputs(ss,rb);

fputs("\t",rb);

if(F2[0]==0&&F4[0]==0){

fputs("\*\t",rb);

fputs("0\t",rb);

fputs("0\t",rb);

fputs("X",rb);

}

else if(F2[0]==1){

fputs("NC\_008253.fna\t",rb);

itoa(F2[1]+1,ss,10);

fputs(ss,rb);

fputs("\t",rb);

itoa(F2[2],ss,10);

fputs(ss,rb);

fputs("\t",rb);

fputs("=",rb);

}

else if(F4[0]==1){

fputs("NC\_008253.fna\t",rb);

itoa(F4[1]+1,ss,10);

fputs(ss,rb);

fputs("\t",rb);

itoa(F4[2],ss,10);

fputs(ss,rb);

fputs("\t",rb);

fputs("=",rb);

}

fputs("\n",rb);

}

}

fclose(ra);

fclose(rb);

fclose(rc);

}

int main()

{

printf("start read data!\n");

str=getdata("NC\_008253.fna");

printf("read data OK!\n");

h = Creat\_Table();

printf("start create table!\n");

CreateHashTable();

printf("create table OK!\n");

//print\_hash(h,"hash.txt");

printf("start SANP!\n");

SANP("Ecoli\_4x.fq1","sam.txt","Ecoli\_4x.fq2");

printf("hello word!\n");

}