# project：

实现在一组DNA序列中寻找转座子的精确匹配的算法。

# motivation：

1. 转座子：转座子（Transposon）亦称为转座元件，跳跃子，是一类DNA序列，它们能够在基因组中通过转录或逆转录，在内切酶（Nuclease）的作用下，在其他基因座上出现。转座子可以分为Ⅰ型转座子和Ⅱ型转座子两类：前者又称为逆元件（Retro element），转座中间体为RNA。该型转座子会先被转录为RNA，然后该RNA被逆转录再次成为DNA，才被插入到目标位点中；后者被称为不复制转座子，被内切酶切下后直接插入到目标位点中。
2. 转座子在生物DNA的主要组成部分（如人类基因组有大约44-45%的转座子），在研究生物的遗传变异中有重要作用。
3. 高通量测序方法的应用，给我们带来了大量的基因序列信息。通过基因组序列比对寻找某一生物基因组中含有的转座子，对研究该生物基因的遗传变异有着重要意义。

# background：

1. 通过Burrows-wheeler变换压缩序列大小（块排序压缩）
2. 简洁的哈希索引
3. 建立后缀树或后缀数组

分析：

1. BWT算法将输入的序列进行“全循环排列”，再对全循环字符串进行字典序排序，排序后输出全循环字符串的最后一个字母连成的字符串。BWT算法的结果是原序列中相同的字符在输出字符串中很大程度上聚集到了一起。BWT算法能够在不需要额外数据的前提下，仅通过牺牲额外的计算，提高文本压缩效率。对经过Burrows-wheeler变换后的序列再使用通用的统计压缩模型（Huffman编码，PPM算法等）进行压缩能够得到理想的压缩率。
2. 哈希索引的原理是通过滑动窗口的方式建立哈希表，将序列位置信息储存于一个哈希表内,建立哈希数据索引的方法，能够达到较快的索引速度，但是需要耗费大量的内存。例如在目前公认的人类基因组版本HG19中，哈希表已经达到了15G，这是一个很大的空间。
3. 后缀树通过建立后缀树这样一个数据结构储存序列位置信息，支持非常快速的对齐。但是后缀树的建立麻烦，计算耗时长，同时消耗很大的空间。后缀树组相对于后缀树来说计算更加简单，同时更节省空间。建立后缀树（或后缀树组）后，可以通过二进制搜索算法进行快速查询，但需要大范围的内存访问。

综合讨论下来，选择了利用建立后缀数组实现序列的精确匹配。

# 后缀数组算法：

什么是后缀数组?

后缀：字符串s从某个位置i到字符串末尾的子串，定义以s的第i个字符为第一个元素的后缀为suff(i)。

后缀数组sa[i]：把s的每个后缀按照字典序排列，后缀数组sa[i]就表示排名为i的后缀的起始位置的下标；它的映 射数组rank[i]就表示suff(i)的字典序排名。 易知性质：sa[rank[i]]=rank[sa[i]]=i

后缀数组的辅助工具：

LCP：最长公共前缀。 LCP(i,j)为排名为i的后缀suff(sa[i])与排名为j的后缀suff(sa[j])的最长公共前缀。

定义height[i]=LCP(i,i-1)，即排名为i的后缀与排名为i-1的后缀的字符串的最长公共前缀。

定义H[i]=height[rank[i]]，即后缀suff(i)与其字典序排名前一名的后缀的最长公共前缀。而本算法利用的正是此 数组。

---

建立后缀数组：

倍增基数排序：

排序依据：ASCII码值（如A<C<T<G）

基数排序：可以在O(n)的时间内对一个二元组(x,y)进行排序，其中x是第一关键字，y是第二关键字

倍增法：每次将排序长度*2

---

求LCP（height数组和H数组）：

利用性质：sa[rank[i]]=rank[sa[i]]=i，可以通过线性计算简单求出

---

序列精确匹配的后缀树组算法：

有长基因组序列s1（长度m）和短转座子序列s2（长度n），用`@`符号连接（`@`的ASCII码最接近且小于`A`,保证从`@`开始的后缀rank=1）成字符串s。求s的后缀树组及LCP。如果能在s1中找到s2的精确匹配，那么

Height[rank[m+2+1]=n

即 存在target：rank[target]=rank[m+2]（即序列s2的字典序）+1 （target<=m）且H[target]=n

如s1=TGG , m=3; s2=TG , n=2; 那么s=TGG@TG ; suff(m+2)=s2=TG

字典序：@TG 、G 、G@TG 、GG@TG 、TG 、TGG@TG

rank[m+2]=5 = rank[1]-1

target=1 对应后缀 TGG@TG ； H[target]=2=n ， 也就是说存在s的后缀字典序在suff(m+2)后且它们之间最长公共前缀与s2的长度相同，那么也就找到了精确匹配，起始位置为target=1。

```
s1:
TGG
s2:
TG
TGG@TG
sa[i]: 4 6 3 2 5 1
rank[i]: 6 4 3 1 5 2
Height[i]: -1 0 1 1 0 2
H[i]: 2 1 1 -1 0 0
```

# 伪代码：

```
void SuffixSort()
{
    基数排序，将由rank和tp组成的二元组排序；
    把子串长度翻倍,更新rank，直到长度大于等于字符串长度；
    基数排序，更新sa数组，并用tp暂时存下上一轮的rank；
    用已经完成的SA来更新与它互逆的rank,并离散rank；
}

void GetHeight()
{
    通过H[i] ≥ H[i - 1] - 1的性质以及sa数组、rank数组得到H数组；
}

int main()
{
    将要比对的字符串用@拼接得到字符串s;
      SuffixSort()//得到sa、rank数组；
      GetHeight()//得到H数组；
      if(存在H[i]>=s2的长度) 判断是否分别来自字符串s1;
        if(来自s1) 得出所有的位置；
}
```

# 复杂度分析：

- 倍增基数排序：
  - 基数排序时间复杂度为O(N*M) （M为字符集的大小，也是口袋的个数，本程序中为常数69，所以实际为O(N)）
  - 倍增基数排序相对于在基数排序的基础上加了二分，所以时间复杂度进一步减少到O(logN)
  - 比较两个字符串的时间复杂度为O(N)
  - 所以总时间复杂度为O(N*logN)
- 求LCP：
  - 计算height数组为线性运算：时间复杂度为O(N)
  - 结果匹配：也是线性运算，时间复杂度为O(N)
- 空间复杂度：开辟了5个数组（sa，rank，tp，tax，height），空间复杂度为O(M+5*N)->O(N)

# result：

```
[ywt@localhost project]$ ./1.3  seq.fastq seq1.fastq seq2.fastq -n
./1.3: option requires an argument -- 'n'
Usage():         [-a]  [-n amount](required)
[ywt@localhost project]$ ./1.3  seq.fastq seq1.fastq seq2.fastq -n 2
[ywt@localhost project]$ cat result.txt
Result:

>seq1 in >seq :
The location of the firstcharacter is 4

>seq2 in >seq :
The location of the firstcharacter is 14
The location of the firstcharacter is 5
The location of the firstcharacter is 7
```

一共完成了三个版本的迭代更新：

版本1.1：交互式单序列比对版本

版本1.2：交互式多序列比对版本

版本1.3：命令行输入版本

最新版本1.3：

准确地实现了在一组DNA序列中寻找转座子的精确匹配的算法，且能够同时匹配多个DNA序列，并且通过命令行输入，结果文件输出的模式适应了在服务器运行。

# issue：

1. 交互式版本的结果可视化
2. MPI多线程运行加快计算速度
3. 利用LCP实现新功能：实现部分匹配的算法并计算匹配度

# reference：


https://blog.csdn.net/weixin_44044714/article/details/101128489

https://www.cnblogs.com/victorique/p/8480093.html#autoid-1-3-0

https://www.cnblogs.com/zwfymqz/p/8413523.html#_label4

赵雅男. 生物序列比对中BWT索引技术及其算法研究[D]. 安徽:中国科学技术大学,2015.


# group work：

版本1.1、后缀数组和LCP算法：唐志轩

版本更新1.2、1.3，开题报告，小组展示：余文韬

文献搜寻、程序检查和算法优化、report：刘澄

# 源代码：

```
//1.1.c
#include<stdio.h>
#include<string.h>
#define MAXN 100000
int rank[MAXN], sa[MAXN], tax[MAXN], tp[MAXN], Height[MAXN];
//sa[i] ：排名为i的后缀的位置
//rank[i]：从第i个位置开始的后缀的排名
//tax[i]：i号元素出现了多少次，用于基数排序
//tp[i]:第二关键字排名为i的后缀的位置,用于基数排序
void GetHeight(int N,int rank[],int sa[],char s[]);
void SuffixSort(int N,char s[]);
int main() {
char s1[MAXN],s2[MAXN];
char s[MAXN];
printf("s1:\n");
scanf("%s", s1);
int lenofs1=strlen(s1);
printf("s2:\n");
scanf("%s", s2);
int lenofs2=strlen(s2);
strcat(s+1,s1);
strcat(s+1,"@");
strcat(s+1,s2);
int N; //N：字符串s的长度
N = strlen(s+1);
//printf("s:\n%s\n",s+1);
SuffixSort(N,s);
GetHeight(N,rank,sa,s);
```

```
int i,link,target; //判断是否分别来自s1,s2
for(i=1;i<=N;i++)
if(rank[i]==1) {link=i; break;}
for(i=1;i<=N;i++)
if(rank[i]==rank[link+1]+1) {target=i; break;}
int j=0;
if(Height[rank[target]]==lenofs2) printf("The location of the firstcharacter is
%d\n",target);
else {j++;printf("Not found!\n");}
while(j==0&&rank[target]<N)
{
    for(i=1;i<=N;i++)
    if(rank[i]==rank[target]+1) {target=i;break;}
    if(Height[rank[target]]>=lenofs2) printf("The location of the firstcharacter
is %d\n",target);
    else j++;
}
return 0;
}
void GetHeight(int N,int rank[],int sa[],char s[]) {
int i, j, k = 0;
printf("Height[rank[i]]: ");
for( i = 1; i <= N; i++) {
if(k) k--;
j = sa[rank[i] - 1];
while(s[i + k] == s[j + k]) k++;
Height[rank[i]] = k;}
for( i = 1; i <= N; i++)
if(rank[i]==1) Height[rank[i]] = -1;
for( i = 1; i <= N; i++)
printf("%d ",Height[rank[i]]);
printf("\n");
}
void SuffixSort(int N,char s[]) {
int i,w,p; //p：排名，用于基数排序
//w:当前倍增长度
int M = 69; //M ：字符集的大小，即排名的个数，用于基数排序。
for ( i = 1; i <= N; i++) {rank[i] = s[i] - '0' + 1, tp[i] = i;}
//以下4行为基数排序
for (i = 0; i <= M; i++) tax[i] = 0;
for (i = 1; i <= N; i++) tax[rank[i]]++;
for (i = 1; i <= M; i++) tax[i] += tax[i - 1];
for (i = N; i >= 1; i--) sa[ tax[rank[tp[i]]]-- ] = tp[i];
for ( w = 1, p = 0; p < N; M = p, w <= 1) {
p = 0;
for (i = 1; i <= w; i++) tp[++p] = N - w + i;
for (i = 1; i <= N; i++) if (sa[i] > w) tp[++p] = sa[i] - w;
for (i = 0; i <= M; i++) tax[i] = 0;
for (i = 1; i <= N; i++) tax[rank[i]]++;
for (i = 1; i <= M; i++) tax[i] += tax[i - 1];
for (i = N; i >= 1; i--) sa[ tax[rank[tp[i]]]-- ] = tp[i];
memcpy(tp,rank,sizeof(rank));
rank[sa[1]] = p = 1;
for (i = 2; i <= N; i++)
rank[sa[i]] = (tp[sa[i - 1]] == tp[sa[i]] && tp[sa[i - 1] + w] ==tp[sa[i] + w])
? p : ++p;
}
printf("sa[i]: ");
```

```
for ( i = 1; i <= N; i++)
printf("%d ", sa[i]);
printf("\n");
printf("rank[i]: ");
for ( i = 1; i <= N; i++)
printf("%d ", rank[i]);
printf("\n");
}
```

```
//1.2.c
#include<stdio.h>
#include<string.h>
#define MAXN 100000
#define MAX 5000
int rank[MAXN], sa[MAXN], tax[MAXN], tp[MAXN], Height[MAXN];
//sa[i] ：排名为i的后缀的位置
//rank[i]：从第i个位置开始的后缀的排名
//tax[i]：i号元素出现了多少次，用于基数排序
//tp[i]:第二关键字排名为i的后缀的位置,用于基数排序
void GetHeight(int N,int rank[],int sa[],char s[]);
void SuffixSort(int N,char s[]);
int main() {
char s[MAXN];//后缀数组字符串
int amount,i; //amount 要比对的序列个数
char ss[MAXN]; int lenof;//主序列（如基因组序列）
char s0[amount][MAX];int lenofs[amount]; //比对的序列

printf("Please input the amount of sequence:");
scanf("%d",&amount);

printf("s:\n");
scanf("%s",ss);
lenof=strlen(ss);
strncat(s+1,ss,lenof);

for(i=0;i<amount;i++){
    printf("s%d:\n",i+1);
    scanf("%s",s0[i]);//printf("s%d:YES\n",i+1);
    lenofs[i]=strlen(s0[i]);
    //printf("s%d:YES\n",i+1);
    int ii;
    for(ii=0;ii<=i;ii++) strcat(s+1,"@");
    strncat(s+1,s0[i],lenofs[i]);//printf("%s\n",s+1);
}
//printf("s:\n%s\n",s+1);
int N; //N：字符串s的长度
N = strlen(s+1);//printf("N=%d\n",N);

SuffixSort(N,s);
GetHeight(N,rank,sa,s);

int link=lenof+1,target; //判断是否分别来自s1,s2
//for(i=1;i<=N;i++)
for(i=0;i<amount;i++)
{
```

```
    printf("s%d:\n",i+1);

 target=sa[rank[link+1]+1];//printf("link=%d,target=%d,lenofs=%d,Height[rank[tar
get]]=%d\n",link,target,lenofs[i],Height[rank[target]]);
    int j=0,jj=0;
    //if(Height[rank[target]]==lenofs[i]) printf("The location of the
firstcharacter is %d\n",target);
    //else {j++;printf("Not found!\n");}
    while(j==0)
    {
        if(Height[rank[target]]>=lenofs[i]) {if(target<=lenof) {printf("The
location of the firstcharacter is %d\n",target);jj++;}}
        else j++;
        target=sa[rank[target]+1];
    }
    if(jj==0) printf("Not found!\n");
    link=link+lenofs[i]+i+2;
}
return 0;
}
void GetHeight(int N,int rank[],int sa[],char s[]) {
int i, j, k = 0;
//printf("Height[rank[i]]: ");
for( i = 1; i <= N; i++) {
if(k) k--;
j = sa[rank[i] - 1];
while(s[i + k] == s[j + k]) k++;
Height[rank[i]] = k;}
for( i = 1; i <= N; i++)
if(rank[i]==1) Height[rank[i]] = -1;
/*for( i = 1; i <= N; i++)
printf("%d ",Height[rank[i]]);
printf("\n");*/
}
void SuffixSort(int N,char s[]) {
int i,w,p; //p: 排名，用于基数排序
//w:当前倍增长度
int M = 69; //M : 字符集的大小，即排名的个数，用于基数排序。
for ( i = 1; i <= N; i++) {rank[i] = s[i] - '0' + 1, tp[i] = i;}
//以下4行为基数排序
for (i = 0; i <= M; i++) tax[i] = 0;
for (i = 1; i <= N; i++) tax[rank[i]]++;
for (i = 1; i <= M; i++) tax[i] += tax[i - 1];
for (i = N; i >= 1; i--) sa[ tax[rank[tp[i]]]-- ] = tp[i];
for ( w = 1, p = 0; p < N; M = p, w <= 1) {
p = 0;
for (i = 1; i <= w; i++) tp[++p] = N - w + i;
for (i = 1; i <= N; i++) if (sa[i] > w) tp[++p] = sa[i] - w;
for (i = 0; i <= M; i++) tax[i] = 0;
for (i = 1; i <= N; i++) tax[rank[i]]++;
for (i = 1; i <= M; i++) tax[i] += tax[i - 1];
for (i = N; i >= 1; i--) sa[ tax[rank[tp[i]]]-- ] = tp[i];
memcpy(tp,rank,sizeof(rank));
rank[sa[1]] = p = 1;
for (i = 2; i <= N; i++)
rank[sa[i]] = (tp[sa[i - 1]] == tp[sa[i]] && tp[sa[i - 1] + w] ==tp[sa[i] + w])
? p : ++p;
}
```

```
/*printf("sa[i]: ");
for ( i = 1; i <= N; i++)
printf("%d ", sa[i]);
printf("\n");
printf("rank[i]: ");
for ( i = 1; i <= N; i++)
printf("%d ", rank[i]);
printf("\n");*/
}
```

```
//1.3.c
#include<stdio.h>
#include<string.h>
#include<unistd.h>
#include<stdlib.h>
#define MAXN 100000

void usage(){
    printf("Usage():\t[-a] [-n amount](required)\n");
}
int rank[MAXN], sa[MAXN], tax[MAXN], tp[MAXN], Height[MAXN];
//sa[i] ：排名为i的后缀的位置
//rank[i]：从第i个位置开始的后缀的排名
//tax[i]：i号元素出现了多少次，用于基数排序
//tp[i]:第二关键字排名为i的后缀的位置,用于基数排序
void GetHeight(int N,int rank[],int sa[],char s[]);
void SuffixSort(int N,char s[]);
int main(int argc,char *argv[])
{
    char s[MAXN];//后缀数组字符串
    int amount,i; //amount 要比对的序列个数
    int a=0,b=0;//bool a ,输出形式;bool b,是否传入参数n

    int o;
    const char *optstring="an:";

    while((o=getopt(argc,argv,optstring))!=-1)
    {

        switch(o){
            case 'a':
                a=1;
                break;
            case 'n':
                b=1;
                amount=atoi(optarg);
                break;
        }
    }

    if(b==0)
    {
        usage();
        return 0;
    }
```

```
char ss[MAXN]; int lenof;//主序列（如基因组序列）
char s0[amount];int lenofs[amount]; //比对的序列


FILE *fp;char name[amount+1][50];
if((fp=fopen(argv[optind],"r"))==NULL)
{
    printf("error when open file %s",argv[optind]);
}
else
{
    fgets(name[0],50,fp);
    name[0][strlen(name[0])-1]='\0';
    fgets(ss,MAXN,fp);
    lenof=strlen(ss)-1;ss[lenof]='\0';
    strncat(s+1,ss,lenof);
}
fclose(fp);


for(i=0;i<amount;i++){
    if((fp=fopen(argv[optind+i+1],"r"))==NULL)
    {
        printf("error when open file %s",argv[optind]);
    }
    else
    {
        fgets(name[i+1],50,fp);
        name[i+1][strlen(name[i+1])-1]='\0';
        fgets(s0,MAXN,fp);
        lenofs[i]=strlen(s0)-1;s0[lenofs[i]]='\0';
        int ii;
        for(ii=0;ii<=i;ii++) strcat(s+1,"@");
        strncat(s+1,s0,lenofs[i]);
    }
    fclose(fp);
}

int N; //N：字符串s的长度
N = strlen(s+1);

SuffixSort(N,s);
GetHeight(N,rank,sa,s);

int link=lenof+1,target; //判断是否分别来自s1,s2

fp=fopen("result.txt","w+");
fprintf(fp,"Result:\n\n");

if (a==1)
{
    fprintf(fp,"sa[i]: ");
    for ( i = 1; i <= N; i++)
    fprintf(fp,"%d ", sa[i]);
    fprintf(fp,"\n");
    fprintf(fp,"rank[i]: ");
    for ( i = 1; i <= N; i++)
```

```
        fprintf(fp,"%d ", rank[i]);
        fprintf(fp,"\n");
        fprintf(fp,"Height[rank[i]]: ");
        for( i = 1; i <= N; i++)
        fprintf(fp,"%d ",Height[rank[i]]);
        fprintf(fp,"\n\n");
    }

    for(i=0;i<amount;i++)
    {
        fprintf(fp,"%s in %s :\n",name[i+1],name[0]);//
        target=sa[rank[link+1]+1];
        int j=0,jj=0;

        while(j==0)
        {
            if(Height[rank[target]]>=lenofs[i]) {if(target<=lenof)
{fprintf(fp,"The location of the firstcharacter is %d\n",target);jj++;}}
            else j++;
            target=sa[rank[target]+1];
        }
        if(jj==0) printf("Not found!\n");
        link=link+lenofs[i]+i+2;
        fprintf(fp,"\n");
    }
    fclose(fp);
return 0;
}
void GetHeight(int N,int rank[],int sa[],char s[])
{
    int i, j, k = 0;

    for( i = 1; i <= N; i++) {
    if(k) k--;
    j = sa[rank[i] - 1];
    while(s[i + k] == s[j + k]) k++;
    Height[rank[i]] = k;}
    for( i = 1; i <= N; i++)
    if(rank[i]==1) Height[rank[i]] = -1;
}

void SuffixSort(int N,char s[])
{
    int i,w,p; //p: 排名，用于基数排序 //w:当前倍增长度
    int M = 69; //M : 字符集的大小，即排名的个数，用于基数排序。
    for ( i = 1; i <= N; i++) {rank[i] = s[i] - '0' + 1, tp[i] = i;}
    //以下4行为基数排序
    for (i = 0; i <= M; i++) tax[i] = 0;
    for (i = 1; i <= N; i++) tax[rank[i]]++;
    for (i = 1; i <= M; i++) tax[i] += tax[i - 1];
    for (i = N; i >= 1; i--) sa[ tax[rank[tp[i]]]-- ] = tp[i];
    for ( w = 1, p = 0; p < N; M = p, w <= 1) {
    p = 0;
    for (i = 1; i <= w; i++) tp[++p] = N - w + i;
    for (i = 1; i <= N; i++) if (sa[i] > w) tp[++p] = sa[i] - w;
    for (i = 0; i <= M; i++) tax[i] = 0;
    for (i = 1; i <= N; i++) tax[rank[i]]++;
    for (i = 1; i <= M; i++) tax[i] += tax[i - 1];
```

```
    for (i = N; i >= 1; i--) sa[ tax[rank[tp[i]]]-- ] = tp[i];
    memcpy(tp,rank,sizeof(rank));
    rank[sa[1]] = p = 1;
    for (i = 2; i <= N; i++)
    rank[sa[i]] = (tp[sa[i - 1]] == tp[sa[i]] && tp[sa[i - 1] + w] ==tp[sa[i] +
w]) ? p : ++p;
    }
}
```