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： 实验 1

实验题目：构建 CSA

班级：1803401

学号：1183710102

姓名：张欣玥

设计成绩	报告成绩	指导老师

一、 实验目的

构建压缩后缀数组的索引结构。

二、 实验原理

增量构建算法

把该基因组（长 n）划分为 k 段，每段长 l（除最后一段以外），整个构建过程可分为基础部分和合并部分。
基因组用 vector<char>存储。

1. 基础部分

先排最后一段并求这最后一段的 ψ 值，因此需要给这(n-k*l)个后缀排序，求 SA 的值以及 SA^{-1} 的值。具体实现：

将最后一段中的各个后缀转换成 string 类型，给各个后缀及其开始的位置构建 pair 组对，并放入一个 vector 容器 T 中，可以直接使用 sort()函数给 vector 依据后缀的字典序进行排序，这样就得到了 SA 的值。

构建数组 RSA 求 SA^{-1}，从头遍历 T，读取由小到大排好序的后缀对应的 SA 的值即 pair 的第一个值，并赋予 RSA 中的对应位置其字典序的大小的值：

```
//求SA-1
int tl = T.size();
int RSA[tl];//SA-1
for(int i=0;i<tl;i++){
    RSA[T[i].first] = i;
}
```

求 ψ，

$$\psi(i) = SA^{-1}[SA[i]+1]$$

直接利用前面两个函数来求即可。这里有一个注意点，规定（最后一个值+1=0）

2. 合并部分

目标：在已知 T' 的 CSA 时，合并其上一段中包含的 l 个后缀。

算法：

A. 计算待加入的l个后缀如果插入已知列表中应该的位置即 order：

$$order(cX,T') = \begin{cases} l_c - 1 & if\ B\ is\ empty \\ \max\{b \mid b \in B\} & otherwise \end{cases}$$

此处需要知道已知 CSA 中各个字母开头的分界线l_c以及 X 在T'中的 order，因为现在只知道最短的后缀对应的 X 的 order 值（**为T'中最长的后缀在T'中的字典序**），所以只需要从后往前迭代计算由短到长的各个后缀的 order 值即可，每次都用到上一次的计算结果。又因为每次都需要计算l个 order 值，所以用一个定长数组来表示 order 即可。

另外，l_c的第一次计算（基础部分中的）可以直接遍历然后记录各个值最后出现的位置，后面每次合并需要更新，可以在计算 order 时顺便记录在该分组中以各个字母开头的后缀有多少个，最后把它们加到原来的l_c值即可。

```
switch(c){
    case 'A':
        depart_pos_increase[1]++;
        if(CSA[depart_pos[0]+1]>=order[j]){//B=空集
        for(int k=depart_pos[0]+1;k<=depart_pos[1];k++){
        break;
    case 'C':
        depart_pos_increase[2]++;
        if(CSA[depart_pos[1]+1]>=order[j]){//B=空集
        for(int k=depart_pos[1]+1;k<=depart_pos[2];k++){
        break;
    case 'G':
        depart_pos_increase[3]++;
```

改进方案：遍历记录直到找到最大的位置的时间复杂度是**线性**的，但是由于以各个字母开头的后缀区的Ψ值都是**有序**的，所以可以用**二分查找**，使得该查找的时间复杂度降为 **log** 级别，具体实现代码如下：

```
left=depart_pos[0]+1;
r=depart_pos[1];
do{
    mid = (left+r)/2;
    if(CSA[mid]<=st)left=mid;
    else if(CSA[mid]>st)r=mid;
}while(left!=r && left!=r-1);
if(st>=CSA[r])order[j-1] = r;
else order[j-1] = left;
```

B. 计算$f(j)$

$f(j)$表示在合并之前即在T'中第 j 小的序列在合并以后的位置（即在T^iT'中第$f(j)$小），因此为原来的位置 j 加上待加入的后缀中应该插入的位

置小于 j 的个数：

$$f(j) = j + \#(order(suf_k, T') \le j),$$

其中#(order(suf k, T')<j)表示 order(suf k, T')小于 j 的个数, 1≤k≤l，$0 \le j < |T'|$。用一个数组 order_num 来表示不同的 j 对应的#(order(suf k, T')≤j)，遍历一遍 order，为所有 order_num 中下标大于该 order 的位置的数量加一。

```
//建立f(j)原来T中现在的值
//遍历order，计算order小于各个值的数量
int order_num[T_size] = {0};
for(int x=0;x<l;x++){
    for(int y=T_size-1;y>order[x];y--){
        order_num[y]++;
    }
}
```

则 f[k] = k + order_num[k]。

但是上述计算 order_num 的方法太过 naive，后面大于的值都是每次只增长 1 涨上去的，这样每个数字都相当于被多次计数，浪费 CPU 时间，拉低性能，所以把计算 order_num 的过程分为两步，先计算 order 等于不同的值的数目，再相加：

```
for(int x=0;x<l;x++){
    order_num[order[x]]++;
    order_map[order[x]].insert(x);
}
int incr = order_num[0];
order_num[0] = 0;
for(int y=0;y<T_size-1;y++){
    int tmp = order_num[y+1];
    order_num[y+1]=order_num[y] + incr;
    incr = tmp;
}
```

C. 计算 $g(j)$

$g(j)$ 表示该 l 个后缀在 T^iT' 中的位置。

$$g(j) = order(suf_j, T') + \#(suf_k \le suf_j),$$

order 已经有了，需要计算在该分组中各个后缀的排序，和基础步骤的排序方法一样。

实现方案 1：直接在每组中构造出该组中的所有后缀的集合，然后使用 sort 函数直接排序：

```
int sa = l;//该组中最后一个字符的位置0-l
pair<int,string> Ti[l+1];
int d = 0;
for(int i=basic;i>basic-l;i--){
    s.insert(s.begin(),whole_gene[i]);
      cout << s << endl;
    Ti[d] = make_pair(sa,s);
    sa--;
    d++;
}
sort(Ti,Ti+l+1,compare);//排的是原来本组新加的l个后缀
for(int k=1;k<=l;k++){
    g[Ti[k-1].first] = order[Ti[k-1].first-1] + k;
}
```

遇到的问题是，如果每组长度为l，则该组中所有的后缀需要占用的空间为$O(l^2)$，而$l=O(\log n)$，空间占用过大，很容易就造成段错误（栈溢出或内存不足），另外，排到后期，各个后缀的长度过长，用 sort 函数直接比较两个后缀的代价也随着增大，时间开销也很大。

实现方案 2：利用已经计算过的 order 值来减少两个后缀字符串直接比较的代价，根据 order 的意义可知，如果一个后缀的 order 值小于另一个后缀的 order 值，则该后缀也一定小于另一个后缀，反之依然成立。所以需要直接比较的后缀范围就限制在了所有 order 相等的后缀集合之内：

```
for(int k=1;k<=l;k++){
    //计算第k个后缀在该组中的排名
    int r = order_num[order[k-1]];
    for(int x=0;x<l;x++){
        if(order[x]==order[k-1]){
            //x和k直接比对
            int minsize = l-max(x,k-1)+T_size;
            int x_start = basic-l+x+1;
            int k_start = basic-l+k;
            for(int y=0;y<minsize;y++){
                if(whole_gene[x_start+y]<whole_gene[k_start+y]){
                    r++;
                    break;
                }
                else if(whole_gene[x_start+y]>whole_gene[k_start+y])
                    break;
            }
        }
```

该方案的问题在于只减少了后缀字符串之间比较的代价，但是由于直接比较引入了比较次数的增加，即每两个后缀之间都要比，并且还要比两次，就造成了比较次数的时间复杂度增加为$O(n^2)$，运行过慢，但是内存占用很小。

实现方案 3：

综合上述两种方案，可以在计算 order_num 的阶段顺便把有着各个不同 order 值的后缀分组，建立数据结构：

```
map<int,set<int>> order_map;
```

其中，key 为 order 值，value 为 order 值为 key 的所有的后缀的起始位置的集合。构建好 order_map 后，遍历所有 order 值对应的集合，并在集合内部排序，因为每次只构建某个 order 值对应的部分后缀的集合，所以内存占

用并不大，在集合内部采用 sort 函数排序，所以排序效率也能得到保证：

```
for(iter=order_map.begin();iter!=order_map.end();iter++){//对每个order值对应的后缀集合排序
    pair<int,string> r_set[1000];
    set<int> s = iter->second;
    set<int>::iterator iter_set;
    int m=0;
    for(iter_set=s.begin();iter_set!=s.end();iter_set++){
        char *s = (char*)whole_gene+basic_start+(*iter_set);
        if(s==NULL) cout << "no" <<endl;
        string str(s);
        //cout << str << endl;
        r_set[m]=make_pair(*iter_set,str);//为该后缀在该分组中起始位置
        m++;
    }
    sort(r_set,r_set+m,compare);
    for(int k=0;k<m;k++){
        int r = order_num[iter->first]+k;
        Ti[r] = r_set[k].first+1;//Ti是该组中排序后的SA值
        g[r_set[k].first+1] = order[r_set[k].first] + r + 1;//1是加上自己
    }
```

```
top - 01:08:03 up  3:24,  1 user,  load average: 0.40, 0.15, 0.10
任务: 318 total,   2 running, 251 sleeping,   0 stopped,   0 zombie
%Cpu(s): 64.2 us, 35.8 sy,  0.0 ni,  0.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
KiB Mem :  2006900 total,   196364 free,  1525640 used,   284896 buff/cache
KiB Swap:   630548 total,   552468 free,    78080 used.   308264 avail Mem

进程 USER      PR  NI    VIRT    RES    SHR ◆ %CPU %MEM     TIME+ COMMAND
  3602 zxy1183+  20   0   79728  31812   3400 R 97.4  1.6   0:12.20 CSA_linux2
  1666 zxy1183+  20   0 3025256 155764  35284 S  1.0  7.8   2:06.56 gnome-shell
  3001 zxy1183+  20   0   46120   3728   2848 R  0.7  0.2   0:19.41 top
  1886 zxy1183+  20   0  272440  11916   6396 S  0.3  0.6   0:19.00 vmtoolsd
  2229 zxy1183+  20   0 1249644 157576  26972 S  0.3  7.9   4:30.69 codeblocks
```

D. 更新ψ

$$
\begin{aligned}
&1.\ \Phi'[f(j)] = f(\Phi[j])\ \textit{for}\ j = 1, 2, \ldots, |T'|.\\
&2.\ \Phi'[g(\ell)] = f(\Phi[0]).\\
&3.\ \Phi'[g(j)] = g(j+1)\ \textit{for}\ j = 1, 2, \ldots, \ell - 1.
\end{aligned}
$$

因为$f(j)$和$g(j)$都是严格递增的函数，$f(j)$：原来在T'中小的在T^iT'还是小的，$g(j)$：如果$suf_i < suf_j$，则$g(i) < g(j)$，所以只要待加入的所有后缀按照字典序排序，它们对应的位置也会依次加入。

算法流程为：

$j_f \leftarrow 1, \quad j_g \leftarrow 1.$
for $t = 0, 1, .., |T^iT'|$
  if $t = g(\ell)$
    $\Phi'[t] \leftarrow f(\Phi[0]);$
  else if $t = f(j_f)$
    $\Phi'[t] \leftarrow f(\Phi[j_f]), \quad j_f$++;
  else $\Phi'[t] \leftarrow g(j_g + 1), \quad j_g$++;

具体实现为：

```
int jf=1;int jg=1;
for(int t=0;t<T_size_new;t++){
    if(t==0)
        CSA_new[t] = g[1];
    else if(t==g[l]){
        CSA_new[t] = f[CSA[0]];jg++;
    }
    else if(t==f[jf]){
        CSA_new[t] = f[CSA[jf]]; jf++;
    }
    else{
        CSA_new[t] = g[Ti[jg-1]+1];
        //cout << Ti[jg-1].first << " " <
        jg++;
    }
}
```

因为在计算中原来串的最后一位的 CSA 值是新加进来的最长的后缀的字典序排序（即 g[1]），而新加进来的最短的后缀的后一位正好是原来串中最长的后缀。

三、 测试结果及分析

1. 算法的正确性：

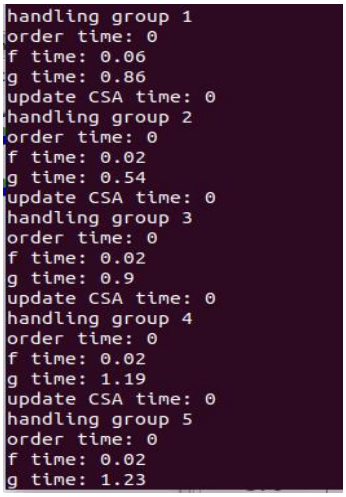
AGCTTTTCATTCTGACTGCAACGGGCA

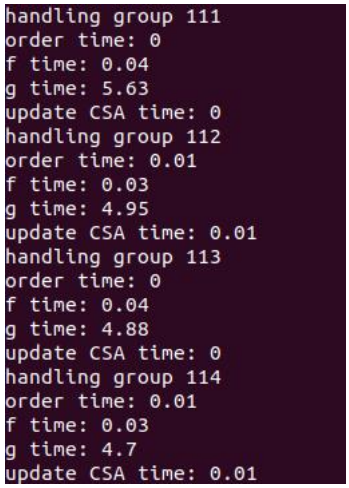
```
$
A$
CA$
GCA$
l = 6
k = 4
num = 4
n = 28
handling group 1
order time: 0
f time: 0
g time: 0
update CSA time: 0
handling group 2
order time: 0
f time: 0
g time: 0
update CSA time: 0
handling group 3
order time: 0
f time: 0
g time: 0
update CSA time: 0
handling group 4
order time: 0
f time: 0
g time: 0
update CSA time: 0

Process returned 0 (0x0)   execution time : 0.007 s
Press ENTER to continue.
```

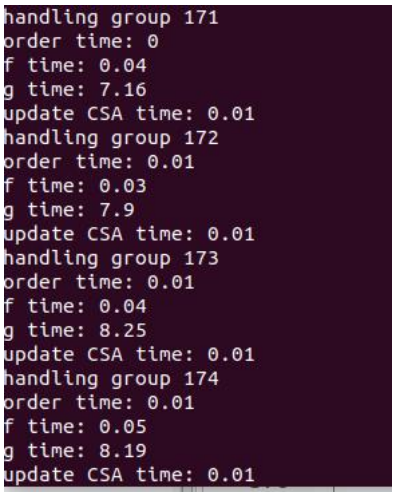
result.txt

```
5 0 3 10 12 17 25 1 2 6 19 22 23 27 4 7 8 13 15 18 9 11 14 16 20 21 24 26
```

2. 性能： 下面是处理各个不同的分组的各个部分的运行时间，可以看出计算每一组的压缩后缀数组φ值的时间主要是在计算 g 函数上，即给各个新加入的后缀排序的过程中。另外，随着基础后缀的变长，g 函数的计算时间也逐渐增长，因为后缀长度增加，比较任意两个后缀之间的大小关系的代价也增加，所需时间也就更多。







四、 经验体会

1. 能用数组就不要用 vector，你可能觉得如果是变长数组，用 vector 可以省内存，但是 vector 建在堆上，每次增加项或者删除项都要根据当前 vector 的容量进行二倍扩张或者收缩，这样会导致消耗时间大幅增长。
2. 在栈里或者在静态存储区的变量读取比在堆里的变量速度会快。
3. Map,set,vector,string 等 C++ STL 的容器的空间的释放是系统随程序运行调整，clear()函数只清空当前数据，并不释放空间。swap()函数可以释放空间，但是系统仍然会为该容器保留该部分空间，以便下次扩容不用再重新申请。另外，malloc_trim(0)可以释放堆顶层不用的空间。
4. Segmentation fault 在 Ubuntu 中可以用 gdb 查看错误位置，在编译时加上 -g，然后运行可执行文件，之后用 gdb 分析 core 文件，即可检查发生错误的行数。如果没有生成 core 文件可能是 ulimit 限制了 core 文件长度。
5. Set 和 map 都是自身可排序的数据结构，并且插入删除的效率都很高。
6. 论文中的 f 计算公式和实际编写的代码有些出入，只计算了 order 值小于的，不加等于的。
7. 最后合并的情况要考虑‘$’后缀，即 t=0 的情况，它一直保持在字典序中最小的位置。
8. 还有一定要从小的数据集开始检验正确性，不要盲目自信，否则等半天结果错了。

五、 附录：源代码（带注释）

```
#include <iostream>
#include <fstream>
#include <vector>
#include <cstdio>
```

```
#include <string>
#include <math.h>
#include <algorithm>
#include<ctime>
#include <stdio.h>
#include <sys/times.h>
#include <unistd.h>
#include <stdlib.h>
#include <map>
#include <set>
#include <malloc.h>

#define _GLIBCXX_USE_CXX11_ABI 0
#define n_real 4938921

using namespace std;

bool compare(pair<int,string> p1,pair<int,string> p2);
int main()
{
	ifstream f;
	string tmp;
	static char whole_gene[n_real];
	f.open("NC_008253.fna",ios::in);
	//f.open("testtext.fna",ios::in);

	if(f.fail()){
		cout << "can not open the file." << endl;
		exit(1);
	}
	//去掉第一行
	if(f.good())
		getline(f,tmp);
	int z=0;
	do{
		if(f.good()){
			getline(f,tmp);
			for(string::iterator it=tmp.begin();it<tmp.end();it++){
				whole_gene[z]=(*it);
				z++;
			}
		}
	}while(!f.eof());
	f.close();
```

```
        whole_gene[n_real-1]='$';
        //cout << whole_gene.size() << endl;

        int n=n_real;//加上最后的$
        int k=(int)20*log2(n);//组数
        int l=n/k;//每组的字符数
//      int l=6;
//      int k=4;

        int last_num = n-k*l;

        //给最后一组 n-l(k-1) : n 排序并计算 CSA
        vector<pair<int,string> > T;
        int Ti[l+1];
        string s("");
        int sa = last_num-1;
        for(int i=n-1;i>=n-last_num;i--){
            s.insert(s.begin(),whole_gene[i]);
            cout << s << endl;
             T.push_back(make_pair(sa,s));
             sa--;
        }
        sort(T.begin(),T.end(),compare);
        int tl = T.size();

        cout << "l = " << l << endl;
        cout << "k = " << k << endl;
        cout << "num = " << last_num << endl;
        cout << "n = " << n << endl;


//      cout << "SA" << endl;
//      for(int i=0;i<tl;i++){
//              cout << T[i].first << " ";
//      }
//      cout << endl;
        //求 SA-1
        int RSA[tl];//SA-1
        for(int i=0;i<tl;i++){
              RSA[T[i].first] = i;
        }
//      cout << "SA-1" << endl;
//      for(int i=0;i<tl;i++){
//              cout << RSA[i] << " ";
```

```
//      }
//      cout << endl;
        //求 CSA 值
        static int CSA[n_real];
        static int CSA_new[n_real];
        static int order_num[n_real] = {0};
        for(int i=0;i<tl;i++){
                int a = T[i].first;
                if(a<tl-1)
                        CSA[i] = RSA[a+1];
                else
                        CSA[i] = RSA[0];
        }
//      cout << "CSA" << endl;
//      for(int i=0;i<tl;i++){
//              cout << CSA[i] << " ";
//      }
//      cout << endl;

        //求分界点
        int depart_pos[4] = {-1};//用-1 来标记
        depart_pos[0] = 0;
        for(vector<pair<int,string>>::iterator iter=T.begin();iter!=T.end();iter++){
                if((*iter).second.at(0)=='A')
                        depart_pos[1] = iter - T.begin();
                if((*iter).second.at(0)=='C')
                        depart_pos[2] = iter - T.begin();
                if((*iter).second.at(0)=='G')
                        depart_pos[3] = iter - T.begin();
        }
        vector<pair<int,string>>().swap(T);


        int clktck = 0;
        clktck = sysconf(_SC_CLK_TCK);
        struct tms    tmsstart, tmsend;
        clock_t         Start, End;
        //merge step
        int order[l+1];//最后一个存的是 T‘的 order
        order[l] = RSA[0];
        //cout << order[l] << endl;
        //vector<pair<int,string> > Ti;
        for(int i=0;i<k;i++){//对每一组，从后往前编号
```

```
//              cout << "depart_pos" << endl;
//              for(int i=0;i<4;i++)
//                    cout << depart_pos[i] << " ";
//              cout << endl;

            int basic = n - i*l-last_num-1;//每一组最后一个字符在 whole_gene 中的位
置
            int basic_start = basic - l + 1;
            cout << "handling group "<<i+1<<endl;
            int T_size = i*l + last_num;//用 T 表示已经排好序的部分
            int depart_pos_increase[4]={0};

            Start = times(&tmsstart);
            //对该组中的每一个后缀，从后往前计算 order 值
             for(int j=l;j>0;j--){
                 char c = whole_gene[basic-l+j];
                 int st=order[j];
                 int left,r,mid;
                 switch(c){
                       case 'A':
                             depart_pos_increase[1]++;
                             //depart_pos_increase[2]++;
                             //depart_pos_increase[3]++;
                             if(depart_pos[1]==-1                                    ||
CSA[depart_pos[0]+1]>order[j]){//B=空集
                                   order[j-1]= depart_pos[0];
                                   break;
                             }
//                             for(int k=depart_pos[0]+1;k<=depart_pos[1];k++){
//                                   if(CSA[k]<=order[j]){
//                                         order[j-1] = k;
//                                   }
//                             }

                             left=depart_pos[0]+1;
                             r=depart_pos[1];
                             do{
                                   mid = (left+r)/2;
                                   if(CSA[mid]<=st)left=mid;
                                   else if(CSA[mid]>st)r=mid;
                             }while(left!=r && left!=r-1);
                             if(st>=CSA[r])order[j-1] = r;
                             else order[j-1] = left;
```

```
                    break;
                case 'C':
                    depart_pos_increase[2]++;
                    //depart_pos_increase[3]++;
                    if(depart_pos[2]==-1 ||
CSA[depart_pos[1]+1]>order[j]){//B=空集
                        order[j-1]= depart_pos[1];
                        break;
                    }
//                    for(int k=depart_pos[1]+1;k<=depart_pos[2];k++){
//                        if(CSA[k]<=order[j]){
//                            order[j-1] = k;
//                        }
//
                    left=depart_pos[1]+1;
                    r=depart_pos[2];
                    do{
                        mid = (left+r)/2;
                        if(CSA[mid]<=st)left=mid;
                        else if(CSA[mid]>st)r=mid;
                    }while(left!=r && left!=r-1);
                    if(st>=CSA[r])order[j-1] = r;
                    else order[j-1] = left;
                    break;
                case 'G':
                    depart_pos_increase[3]++;
                    if(depart_pos[3]==-1 ||
CSA[depart_pos[2]+1]>order[j]){//B=空集
                        order[j-1]= depart_pos[2];
                        break;
                    }
//                    for(int k=depart_pos[2]+1;k<=depart_pos[3];k++){
//                        if(CSA[k]<=order[j]){
//                            order[j-1] = k;
//                        }
//                    }
                    left=depart_pos[2]+1;
                    r=depart_pos[3];
                    do{
                        mid = (left+r)/2;
                        if(CSA[mid]<=st)left=mid;
                        else if(CSA[mid]>st)r=mid;
                    }while(left!=r && left!=r-1);
                    if(st>=CSA[r])order[j-1] = r;
```

```
                    else order[j-1] = left;
                    break;
                case 'T':
                    depart_pos_increase[0]++;
                    if(depart_pos[3]+1==T_size){//之前没有以 T 开头的后缀
                        order[j-1]= depart_pos[3];
                        break;
                    }
                    if(CSA[depart_pos[3]+1]>order[j]){//B=空集
                        order[j-1]= depart_pos[3];
                        break;
                    }
//                    for(int k=depart_pos[3]+1;k<=i*l+last_num-1;k++){
//                        if(CSA[k]<=order[j]){
//                            order[j-1] = k;
//                        }
//                    }
                    left=depart_pos[3]+1;
                    r=i*l+last_num-1;
                    do{
                        mid = (left+r)/2;
                        if(CSA[mid]<=st)left=mid;
                        else if(CSA[mid]>st)r=mid;
                    }while(left!=r && left!=r-1);
                    if(st>=CSA[r])order[j-1] = r;
                    else order[j-1] = left;
                    break;
            }
        }

//        cout << "depart_pos_increase" << endl;
//        for(int i=0;i<4;i++)
//            cout << depart_pos_increase[i] << " ";
//        cout << endl;
//
//        cout << "order" << endl;
//        for(int z=0;z<l;z++){
//            cout << order[z] << "    ";
//        }
//        cout << endl;
        End = times(&tmsend);
        cout << "order time: " << (End - Start)/(double) clktck << endl;
```

```
        //建立 f(j)原来 T 中现在的值
        //遍历 order，计算 order 小于各个值的数量
        map<int,set<int>> order_map;
//        for(int x=0;z<l;z++){
//                set<int> st;
//            order_map.insert(x,st);
//        }
        Start = times(&tmsstart);
        for(int x=0;x<n_real;x++){
            order_num[x]=0;
        }
        for(int x=0;x<l;x++){
            order_num[order[x]]++;
            order_map[order[x]].insert(x);
        }
        int incr = order_num[0];
        order_num[0] = 0;
        for(int y=0;y<T_size-1;y++){
            int tmp = order_num[y+1];
            order_num[y+1]=order_num[y] + incr;
            incr = tmp;
        }
//        cout << "order_num" << endl;
//        int num_test=0;
//        for(int z=0;z<T_size;z++){
//            cout << order_num[z] << endl;
//            num_test += order_num[z];
//        }
//        cout << "num_test = " << num_test << endl;


//        cout << "f" << endl;
        int f[T_size];
        for(int k=0;k<T_size;k++){
            f[k] = k + order_num[k];
            //cout << f[k] << " ";
        }
//        cout << endl;
        End = times(&tmsend);
        cout << "f time: " << (End - Start)/(double) clktck << endl;

        //建立 g(j)新加的后缀的值
        Start = times(&tmsstart);
        int g[l+1];
```

```
//            cout << "g" << endl;
//            int sa = l;//该组中最后一个字符的位置 0-l
//            pair<int,string> Ti[l+1];
//            int d = 0;
//            for(int i=basic;i>basic-l;i--){
//                  s.insert(s.begin(),whole_gene[i]);
////                  cout << s << endl;
//                  Ti[d] = make_pair(sa,s);
//                  sa--;
//                  d++;
//            }
//            sort(Ti,Ti+l+1,compare);//排的是原来本组新加的 1 个后缀
//            for(int k=1;k<=l;k++){
//                  g[Ti[k-1].first] = order[Ti[k-1].first-1] + k;
//            }


            map<int,set<int>>::iterator iter;
            for(iter=order_map.begin();iter!=order_map.end();iter++){//对每个 order 值
对应的后缀集合排序
                  pair<int,string> r_set[1000];
                  set<int> s = iter->second;
                  set<int>::iterator iter_set;
                  int m=0;
                  for(iter_set=s.begin();iter_set!=s.end();iter_set++){
                        char *s = (char*)whole_gene+basic_start+(*iter_set);
                        if(s==NULL) cout << "no" <<endl;
                        string str(s);
                        //cout << str << endl;
                        r_set[m]=make_pair(*iter_set,str);//x 为该后缀在该分组中起始
位置
                        m++;
                  }
                  sort(r_set,r_set+m,compare);
                  for(int k=0;k<m;k++){
                        int r = order_num[iter->first]+k;
                        Ti[r] = r_set[k].first+1;//Ti 是该组中排序后的 SA 值
                        g[r_set[k].first+1] = order[r_set[k].first] + r + 1;//1 是加上自己
                  }
            }


//            for(int x=1;x<=l;x++){
//                  cout << g[x] << " ";
```

```
//              }
//              cout << endl;
            End = times(&tmsend);
            cout << "g time: " << (End - Start)/(double) clktck << endl;


            //计算新的 CSA 的值
            Start = times(&tmsstart);
            int T_size_new = T_size+l;
            //vector<int> CSA_new(T_size_new);
            int jf=1;int jg=1;
            for(int t=0;t<T_size_new;t++){
                if(t==0)
                    CSA_new[t] = g[1];
                else if(t==g[l]){
                    CSA_new[t] = f[CSA[0]];jg++;
                }
                else if(t==f[jf]){
                    CSA_new[t] = f[CSA[jf]]; jf++;
                }
                else{
                    CSA_new[t] = g[Ti[jg-1]+1];
                    //cout << Ti[jg-1].first << " " << jg <<endl;
                    jg++;
                }
            }
            End = times(&tmsend);
            cout << "update CSA time: " << (End - Start)/(double) clktck << endl;
            //CSA = CSA_new;
            for(int x=0;x<T_size_new;x++){
                CSA[x]=CSA_new[x];
            }

//              cout << "Ti" << endl;
//              for(int i=0;i<l;i++)
//                  cout << Ti[i].first << " " << Ti[i].second;
//              cout << endl;
//
//              cout << "CSA" << endl;
//              for(int i=0;i<T_size_new;i++)
//                  cout << CSA[i] << " ";
//              cout << endl;

            order[l] = g[1];
```

```
            //更新 depart_pos
            for(int x=1;x<4;x++){
                if(depart_pos[x]==-1 && depart_pos_increase[x]!=0){
                    depart_pos[x] = depart_pos[x-1];
//                    for(int y=1;y<=x;y++){
//                        depart_pos[x] += depart_pos_increase[x];
//                    }
                    depart_pos[x] += depart_pos_increase[x];
                }
                else if(depart_pos[x]!=-1){
                    for(int y=1;y<=x;y++){
                        depart_pos[x] += depart_pos_increase[y];
                    }
                }
            }
            for(iter=order_map.begin();iter!=order_map.end();iter++){
                    set<int>().swap(iter->second);
            }
            map<int,set<int>>().swap(order_map);
            //order_map.clear();
            malloc_trim(0);




        }

        ofstream outfile;
        outfile.open("result.txt");
        for(int i=0;i<n;i++)
            outfile << CSA[i] << " ";
        outfile.close();
        return 0;
}

bool compare(pair<int,string> p1, pair<int,string> p2){
        string s1 = p1.second;
        string s2 = p2.second;
       return s1+s2 < s2+s1;
}
```