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：限选

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

班级：1803401

学号：1180300906

姓名：张飞扬

|  |  |  |
| --- | --- | --- |
| 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）作为输入

建立算法和系统，可以为基因组序列建立自索引表示。

实验要求

自索引结构可以选择CSA或者BWT

自索引构建算法可以参照以下论文之一

Liu and Zhu, 2016（Advised）

Hon et al., 2007（Advised）

Karkkainen, 2007

其他相关论文

高分要求：

完整地完成算法所有模块

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. 实验原理

2.1实验原理概述

本次实验采用增量构建的方法来构建 CSA。

首先，定义字母表 ∑，在DNA序列中，∑ = {A, C, G, T}。定义 $ 为字符串末尾字符。

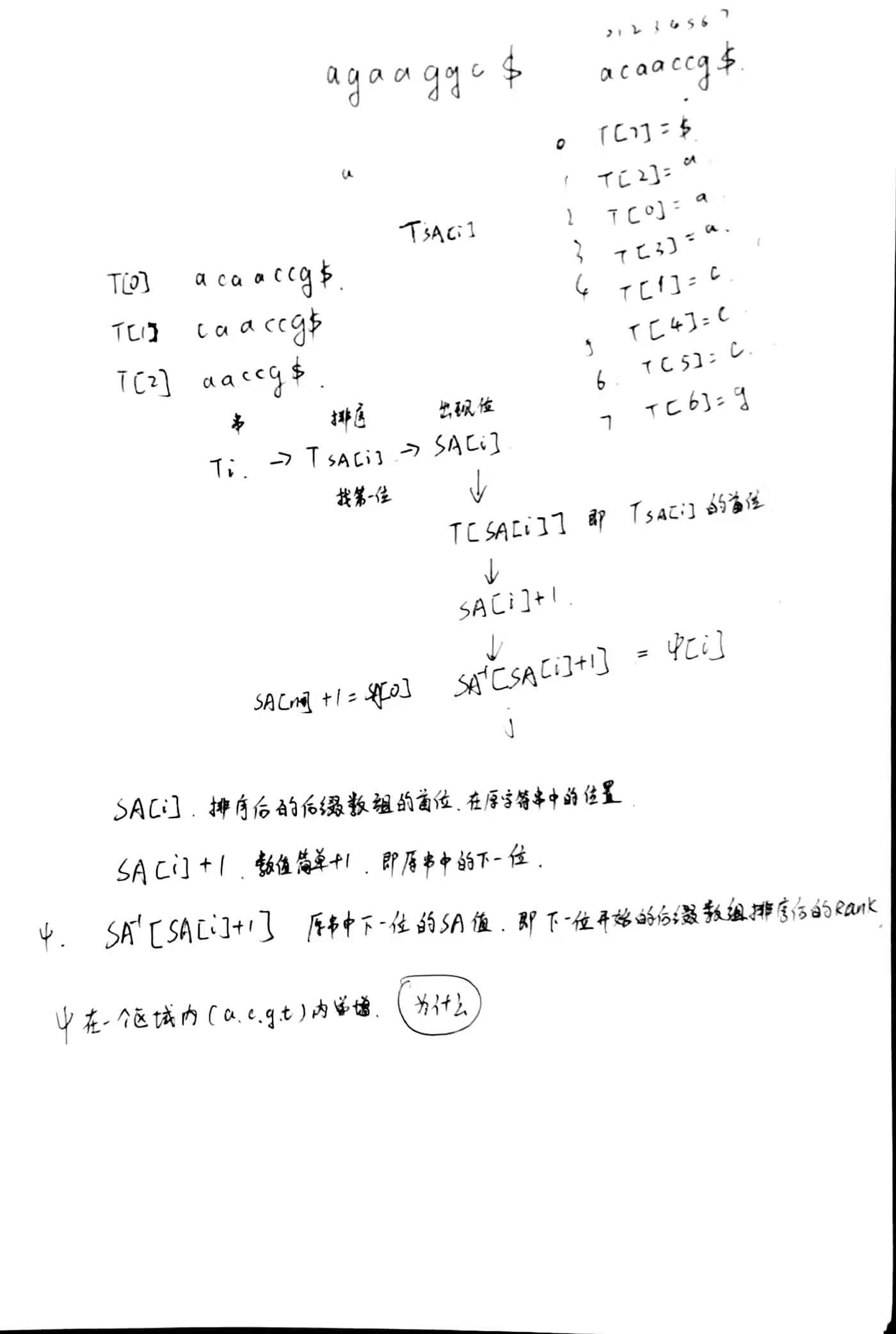
规定字典序如下：$<A<C<G<T。 设 T 是一个 n 长的定义在 ∑ 上的字符串。T[0..n] = T[0]T[1] . . . T[n], 并规定 T[n] = $。规定 Ti = T[i..n] = T[i]T[i + 1] . . . T[n] ，即 T 的一个从第 i 位开始至最后一位的后缀。

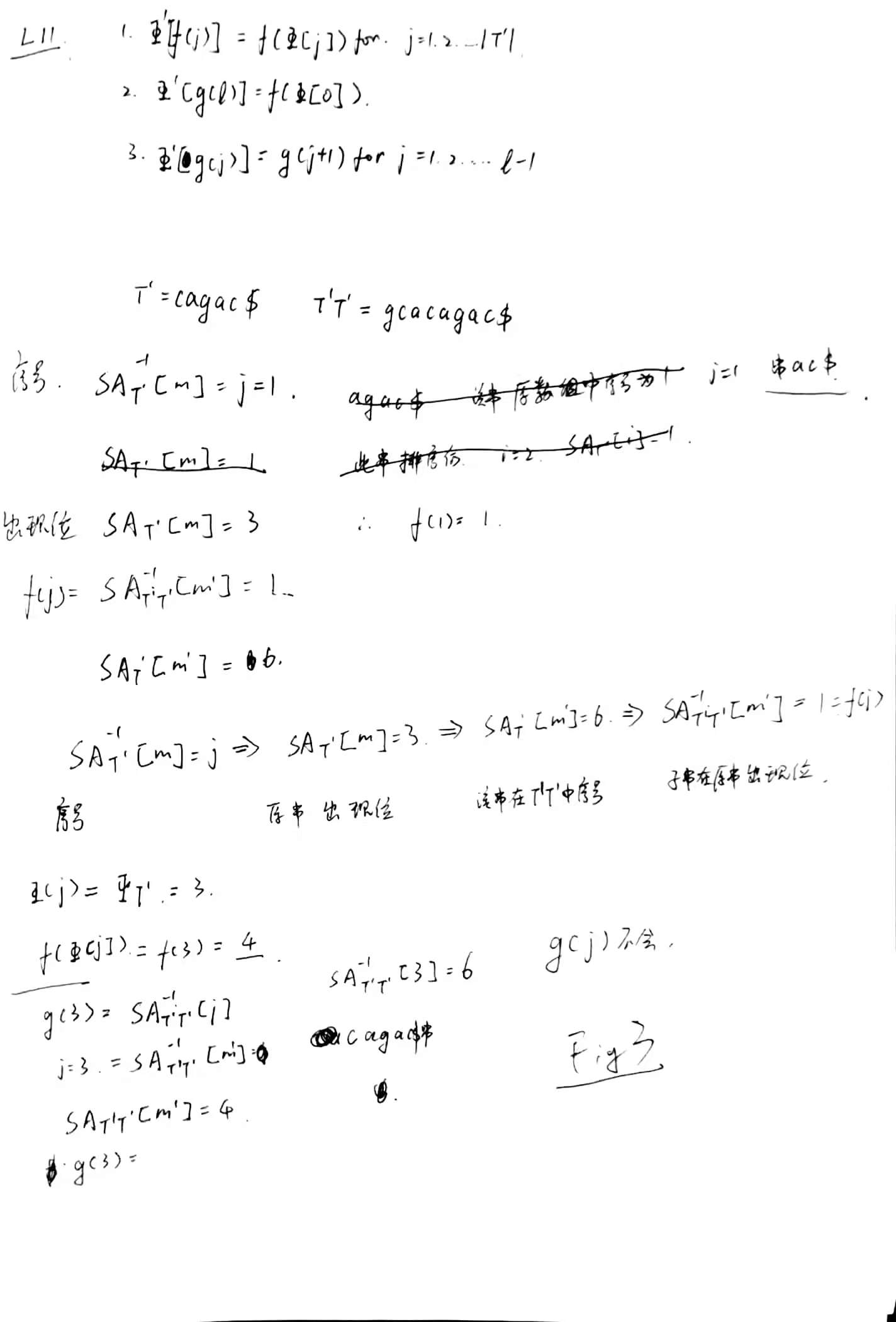
首先得到后缀数组 SA ，SA是将 T 的 n 个后缀排序后的序列，数组SA[i] 存储 T 的第 i 小的后缀在后缀数组中的位置。

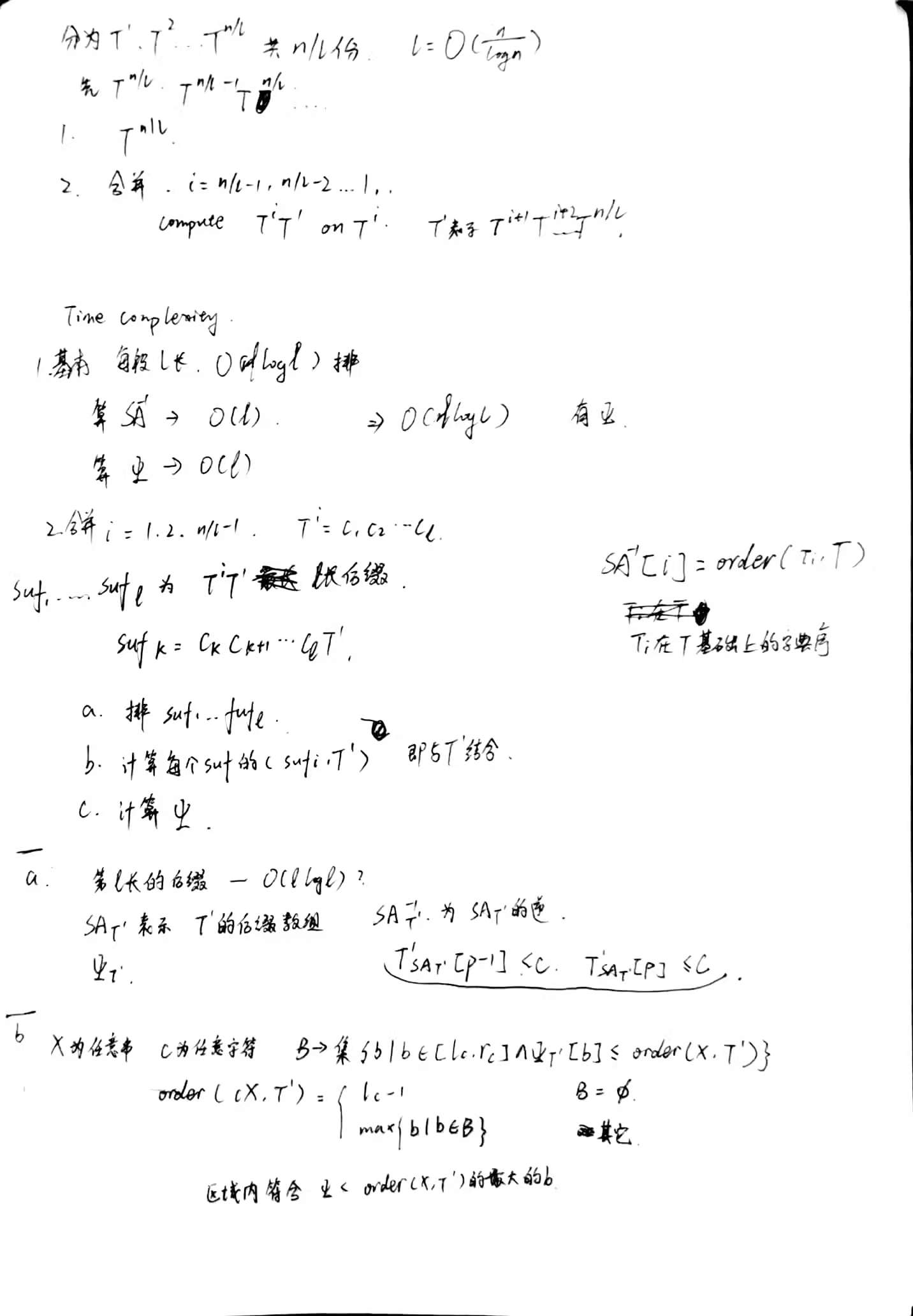
定义 SA^-1[i] 为 T 的第 i 个后缀在后缀数组中存储的位置，即 Ti 的顺序。

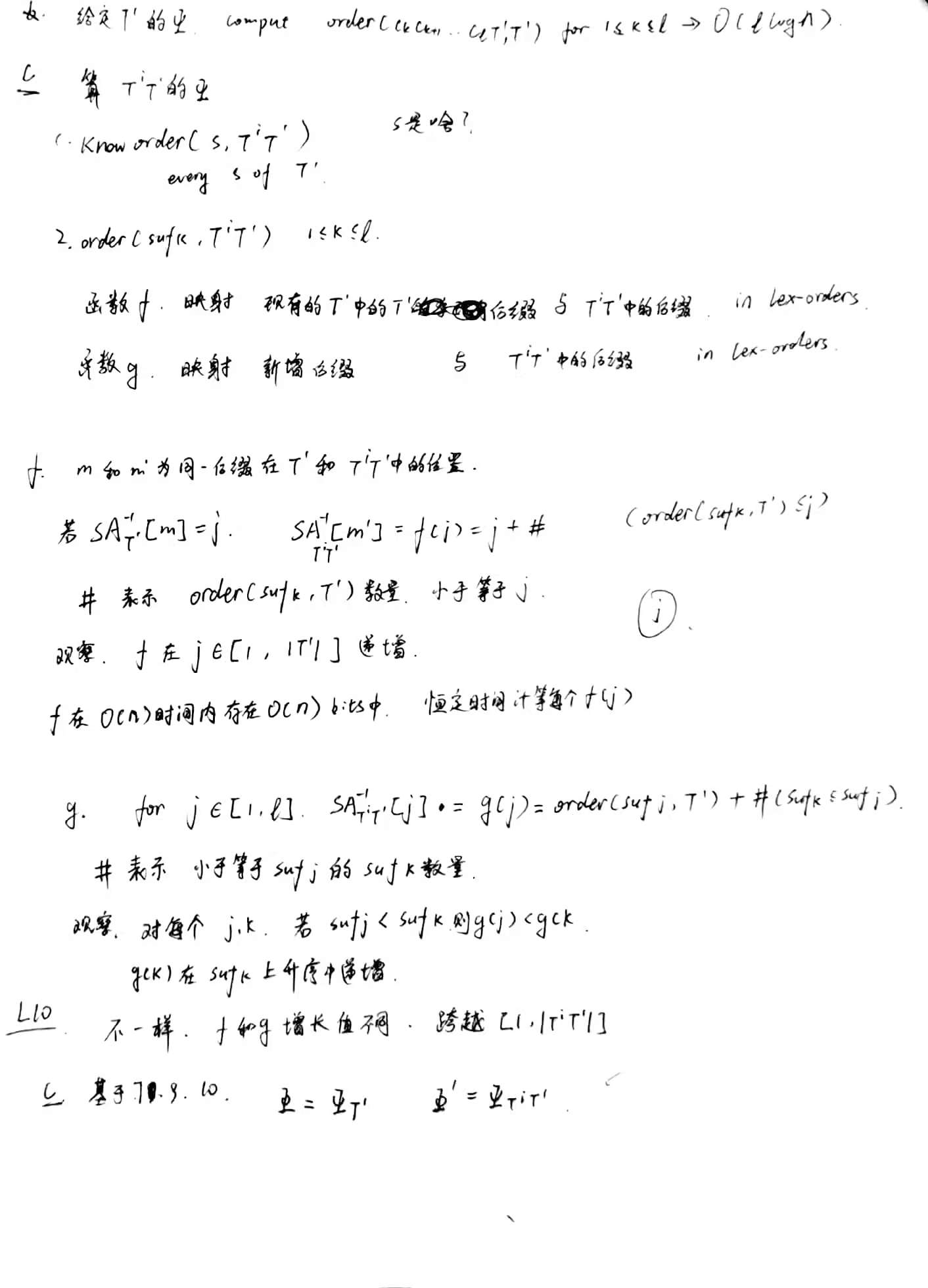
定义函数 Ψ：Ψ[i] = SA^-1[SA[i] + 1] for i = 1, 2, . . . , n，并规定 Ψ[0] = SA^-1[0]。不难看出 Ψ[i] 存储的是第 i 小的后缀在 T 上的下一个后缀在 SA 中存储的位置，或者说是第 i 小的后缀去掉首字符之后在 SA 的位置，我们可以把 Ψ 理解为一个指向 SA 的指针。CSA 中存储就的是数组 Ψ[0..n]，共 n+1 个整数。

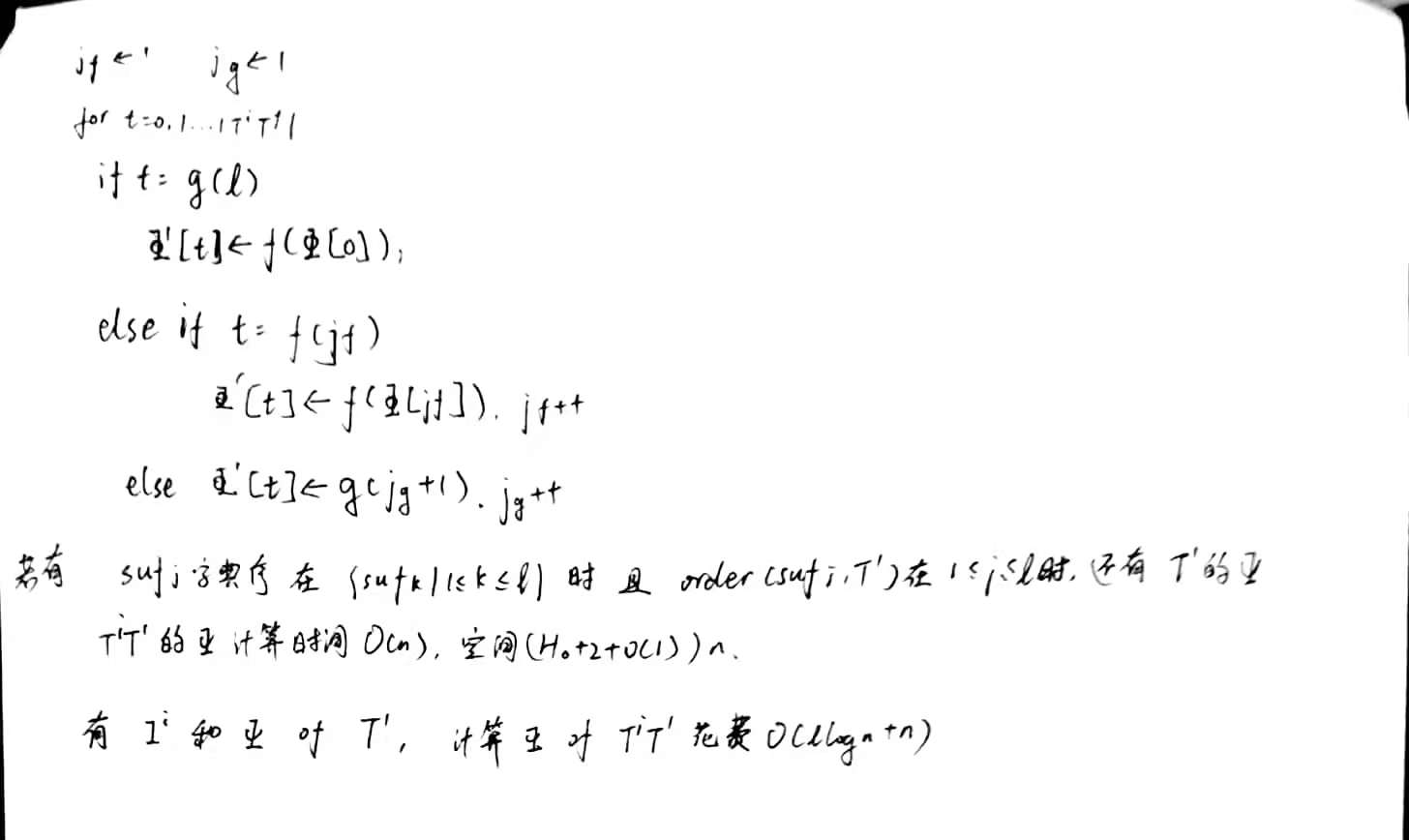
2.2论文学习











1. 算法设计

将 T 划分为 n/l 个连续的区域，记为 T1 , T2 , . . . , T n/l，l = O( log nn )。通过 Base Step 和 Merge Step 两个部分来逐步构建整个 T 上的 CSA。

3.1 Base Step

主要构建 T^n/l 的 Ψ。首先通过排序得到 T^n/l 的后缀数组，这个操作耗费 O(l log l) 的时间。计算得到 SA^-1 耗费 O(l) 时间，然后得到 Ψ 需要 O(l) 时间。因此 Base Step 的时间复杂度是 O(l log l) 的。

3.2 Merge Step

这一步的主要思想是基于之前的排序结果，将新的串插入进来。记 Ti = c1c2 · · · cl , i ∈ 1, 2, . . . , n/l-1，设 suf1, suf2, . . . , sufl 是 TiT′ 的 l 个最长的后缀，显然 sufk = ckck+1 · · · clT′。

首先，将这 l 个最长后缀排序，这一步和 Base Step 中的排序并无区别。然后对每一个 sufi，要得到其在 T′ 中的顺序 order(sufi , T′)。然后就可以构造 TiT′ 的 Ψ。

1. 测试结果分析

程序正确运行，时间复杂度分析合理。

1. 经验体会
2. 从计算机方面来说，要有足够好的c/c++语言能力，这门语言对于高效实现新算法来说十分重要。
3. 从数学基础来说，我觉得应该：

（1）学好线性代数

（2）学好高等数学，或者数学分析

（3）学好统计学

（4）学好离散数学

（5）学好计算机算法和数据结构

3. 从生物学来说要学好基因组学、遗传学等，有助于判断结果的合理性，或者对结果进行解释。

4. 对自己遇到的问题，一定要把文献上的数据拿来，自己独立分析一遍，自己去体会分析的过程，才能对这些这些问题有更深入的理解。

1. 附录：源代码

Main函数

#include<stdlib.h>

#include<string.h>

#include"CSA.h"

#include<ctime>

#include<fstream>

#include<iostream>

using namespace std;

void usage();

void helpbuild();

void helpload();

void helpsave();

void helpcount();

void helplocate();

void splitcommand(string command,string result[]);

void showpos( int \* pos,int num);

int main(int argc, char\* argv[])

{

usage();

string command;

string result[2];

char filename[100]={'\0'};

char indexname[100]={'\0'};

CSA \*csa=NULL;

while(1)

{

result[0]="";

result[1]="";

command="";

cout<<">";

getline(cin,command);

splitcommand(command,result);

if(result[0]=="quit")

break;

else if(result[0]=="build")

{

if(csa!=NULL)

delete csa;

csa=NULL;

csa=new CSA(result[1].data());

}

else if(result[0]=="count")

{

int num=0;

if(csa!=NULL)

{

csa->counting(result[1].data(),num);

cout<<"occs: "<<num<<endl;

}

else

{

cout<<"build a csa first"<<endl;

}

}

else if(result[0]=="locate")

{

int \* pos;

int num=0;

if(csa!=NULL)

{

pos=csa->locating(result[1].data(),num);

showpos(pos,num);

delete [] pos;

}

else

cout<<"build a csa first"<<endl;

}

else if(result[0]=="del")

{

if(csa!=NULL)

delete csa;

csa=NULL;

}

else if(result[0]=="load")

{

if(csa!=NULL)

delete csa;

csa=new CSA();

csa->load(result[1].data());

}

else if(result[0]=="save")

{

csa->save(result[1].data());

}

else if(result[0]=="size")

{

if(csa!=NULL)

cout<<csa->compressRatioForCount()\*8<<endl;

else

cout<<"build a csa first"<<endl;

}

else if(result[0]=="help")

{

if(result[1]=="build")

helpbuild();

if(result[1]=="count")

helpcount();

if(result[1]=="locate")

helplocate();

if(result[1]=="load")

helpload();

if(result[1]=="save")

helpsave();

}

else

usage();

}

return 0;

}

void showpos(int \* pos,int num)

{

cout<<"occs:"<<num<<endl;

for(long int i=0;i<num;i++)

{

cout<<pos[i]<<endl;

if((i+1)%20==0)

{

char command;

cout<<"-----------------more---------------------";

system("stty raw");

command=getchar();

cout<<endl<<'\r';

system("stty cooked");

if(command==27)

{

cout<<endl;

return ;

}

}

}

}

void splitcommand(string command,string result[])

{

int i=0;

int start=0;

int len=command.length();

result[0]=command;

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

{

if(command.at(i)!=' ')

continue;

result[0]=command.substr(0,i);

start=i+1;

break;

}

result[1]=command.substr(start,len);

}

void usage()

{

cout<<"--------------------------------------------------------------------------------------"<<endl;

cout<<"The flowing commands are supported "<<endl;

cout<<" help XX: show the details for the command XX"<<endl;

cout<<" build XX:build the index of file XX"<<endl;

cout<<" load XX: load the index file XX"<<endl;

cout<<" save XX: write the csa to a index file XX"<<endl;

cout<<" count XX: count the pattern XX's occs" <<endl;

cout<<" locate YY: enum eyery position of the pattern"<<endl;

cout<<" size: the size of the csa"<<endl;

cout<<" quit: say goodbye"<<endl;

}

void helpbuild()

{

cout<<"build XX"<<endl;

cout<<" XX:the source file,it's you responsibility to provide a correct path"<<endl;

}

void helpcount()

{

cout<<"count XX"<<endl;

cout<<" XX:the pattern.you have the responsibility to ensure the index csa is nearby,otherwise,nothing you will get"<<endl;

}

void helplocate()

{

cout<<"locate XX"<<endl;

cout<<" XX: the pattern.it's like count, and you can scan the positions of all the occs in a manner like the system command -more-"<<endl;

}

void helpload()

{

cout<<"load XX"<<endl;

cout<<" XX: the csa index file, the command will read the index file and build a csa secretly"<<endl;

}

void helpsave()

{

cout<<"save XX"<<endl;

cout<<" XX: the csa index file, the command will save the csa in file XX"<<endl;

}

CSA函数

#include<stdlib.h>

#include "CSA.h"

#include<cstdio>

#include<cmath>

#include"divsufsort.h"

#include<iostream>

using namespace std;

#define SearchKind 2

integer CSA::save(const char \* indexfile){

savekit s(indexfile);

s.writeu64(198809102510);

s.writeinteger(n);

s.writeinteger(alphabetsize);

s.writeinteger(SL);

s.writeinteger(L);

s.writeinteger(D);

s.writeinteger(RD);

//sal

SAL->write(s);

//ranl

RankL->write(s);

//code

s.writeinteger(256);

s.writeintegerarray(code,256);

//start

s.writeinteger(alphabetsize+1);

s.writeintegerarray(start,alphabetsize+1);

//incode

s.writeinteger(alphabetsize);

s.writeintegerarray(incode,alphabetsize);

//phi0

Phi0->write(s);

s.close();

return 0;

}

integer CSA::load(const char \* indexfile){

loadkit s(indexfile);

u64 magicnum;

s.loadu64(magicnum);

if(magicnum!=198809102510)

{

cerr<<"Not a CSA file"<<endl;

exit(0);

}

s.loadinteger(this->n);

s.loadinteger(this->alphabetsize);

s.loadinteger(SL);

s.loadinteger(L);

s.loadinteger(D);

s.loadinteger(RD);

SAL=new InArray();

SAL->load(s);

RankL=new InArray();

RankL->load(s);

//code

integer len=0;

s.loadinteger(len);

this->code=new integer[len];

s.loadintegerarray(this->code,len);

//start

s.loadinteger(len);

this->start=new integer[len];

s.loadintegerarray(this->start,len);

//incode

s.loadinteger(len);

this->incode=new integer[len];

s.loadintegerarray(this->incode,len);

Phi0=new Phi();

Phi0->load(s);

s.close();

return 0;

}

CSA::CSA(const char \* sourcefile,integer L,integer D,integer phitype){

this->SL=L\*18;

this->L =L;

this->D =D;

this->RD=D\*16;

this->code =new integer[256];

for(integer i=0;i<256;i++)

code[i]=0;

this->alphabetsize =0;

uchar \*T=NULL;

T=GetFile(sourcefile);

statics(T);

integer \*SA=new integer[n];

divsufsort(T,SA,n);

parmaters p={alphabetsize,n,SL,L,start,lastchar,SA,T,code,phitype};

CreateSupportStructer(&p);

if(SA!=NULL)

delete [] SA;

if(T!=NULL)

delete [] T;

// cout<<"CSA is done"<<endl;

/\*

integer start=0;

integer num=n-0;

integer \* phiarray=Phi0->GetPhiArray();

integer \* phipiece=new integer[num];

Phi0->GetPhiPiece(start,num,phipiece);

for(integer i=0;i<num;i++)

if(phiarray[i+start]!=phipiece[i]){

cout<<"fuck "<<phiarray[i+start]<<" "<<phipiece[i]<<" "<<i<<" "<<i+start<<endl;

break;

}

\*/

}

integer CSA::getN(){

return n;

}

integer CSA::getAlphabetsize(){

return this->alphabetsize;

}

CSA::~CSA(void){

delete SAL;

delete RankL;

delete [] start;

delete Phi0;

delete [] code;

delete [] incode;

}

bool CSA::existential(const char \*Pattern){

integer L=0;

integer R=0;

Search2(Pattern,L,R);

if(R>=L)

return true;

else

return false;

}

void CSA::counting(const char \*Pattern,integer &num){

integer L=0;

integer R=0;

Search(Pattern,L,R);

// cout<<L<<" "<<R<<endl;

num=R-L+1;

}

void CSA::statics(uchar \* T){

for(integer i=0;i<n;i++)

code[T[i]]++;

for(integer i=0;i<256;i++)

if(code[i]!=0)

this->alphabetsize ++;

this->start =new integer[this->alphabetsize +1];

this->start [this->alphabetsize ]=n;

this->start [0]=0;

integer k=1;

integer pre=0;

for(integer i=0;i<256;i++)

if(code[i]!=0)

{

start[k]=pre+code[i];

pre=start[k];

k++;

}

this->incode =new integer[this->alphabetsize];

k=0;

for(integer i=0;i<256;i++)

if(code[i]!=0)

{

code[i]=k;

incode[k]=i;

k++;

}

else

code[i]=-1;

lastchar=T[n-1];

}

uchar\* CSA::GetFile(const char \* filename){

FILE \*fp=fopen(filename,"r+");

if(fp==NULL)

{

cout<<"Be sure that the file is available"<<endl;

exit(0);

}

fseek(fp , 0, SEEK\_END);

this->n = ftell(fp);

uchar \* T=new uchar[n];

fseek(fp , 0, SEEK\_SET);

integer e=0;

integer num=0;

while((e=fread(T+num,sizeof(uchar),n-num,fp))!=0)

num=num+e;

if(num!=n)

exit(0);

fclose(fp);

return T;

}

integer CSA::sizeInByte(){

return (SAL->GetMemorySize ()+Phi0->Size()+RankL->GetMemorySize ());

}

integer CSA::sizeInByteForCount(){

return Phi0->Size();

}

double CSA::compressRatio(){

return sizeInByte()/(1.0\*n);

}

double CSA::compressRatioForCount(){

return sizeInByteForCount()/(1.0\*n);

}

void CSA::CreateSupportStructer(parmaters \*csa){

integer i=0;

integer j=0;

integer step1=D;

integer step2=RD;

SAL=new InArray(n/step1+1,blog(n));

RankL=new InArray(n/step2+1,blog(n));

for(i=0,j=0;i<n;i=i+step1,j++){

SAL->SetValue (j,csa->SA[i]);

}

for(i=0;i<n;i++){

if(csa->SA[i]%step2==0)

RankL->SetValue (csa->SA[i]/step2,i);

}

Phi0=new Phi(csa);

}

void CSA::Search2(const char \*Pattern, integer &L, integer &R){

integer len=strlen(Pattern);

if(len==0){

L=1;

R=0;

return;

}

unsigned char c=Pattern[len-1];

integer coding=code[c];

if(coding <0||coding>alphabetsize-1){

L=1;

R=0;

return ;

}

integer Left=start[coding];

integer Right=start[coding+1]-1;

integer l0=0;

integer r0=0;

for(integer i=len-2;i>=0;i--){

c=Pattern[i];

coding=code[c];

if(coding<0){

Left=1;

Right=0;

break;

}

l0=start[coding];

r0=start[coding+1]-1;

Right=Phi0->RightBoundary(Right,l0,r0);

Left=Phi0->LeftBoundary(Left,l0,r0);

if(Left>Right){

Left=1;

Right=0;

break;

}

}

L=Left;

R=Right;

return ;

}

void CSA::Search(const char \*Pattern, integer &L, integer &R){

integer templeft;

integer tempright;

integer jj;

integer mleft;

integer mright;

integer middle;

integer Right;

integer left;

integer right;

integer Left;

integer i;

integer len=strlen(Pattern);

unsigned char c=Pattern[len-1];

integer coding=code[c];

if(coding>alphabetsize-1){

L=1;

R=0;

return ;

}

Left=start[coding];

Right=start[coding+1]-1;

for(i=len-2;i>=0;i--){

c=Pattern[i];

coding =code[c];

if(coding>alphabetsize-1){

L=1;

R=0;

return ;

}

left=start[coding];

right=start[coding+1]-1;

if(coding==code[lastchar])

left=left+1;

if(left>right || Phi0->GetValue (left)>Right || Phi0->GetValue (right)<Left){

L=1;

R=0;

return ;

}

else{

mleft=left;

mright=right;

while(mright-mleft>1){

middle=(mleft+mright)/2;

jj=Phi0->GetValue (middle);

if(jj<Left)

mleft=middle;

else

mright=middle;

}

jj=Phi0->GetValue (mleft);

if(jj>=Left && jj<=Right)

templeft=mleft;

else

templeft=mright;

mleft=left;

mright=right;

while(mright-mleft>1){

middle=(mleft+mright)/2;

jj=Phi0->GetValue (middle);

if(jj>Right)

mright=middle;

else

mleft=middle;

}

jj=Phi0->GetValue (mright);

if(jj>=Left && jj<=Right)

tempright=mright;

else

tempright=mleft;

Left=templeft;

Right=tempright;

if(Left>Right){

L=1;

R=0;

return;

}

}

}

if(Left>Right)

L=1,R=0;

else

L=Left,R=Right;

}

integer CSA::lookup(integer i){

integer D=this->D;

integer step=0;

while(i%D!=0){

i=Phi0->GetValue(i);

step++;

}

i=i/D;

return (n+SAL->GetValue (i)-step)%n;

}

/\*

void CSA::SelfTesting()

{

cout<<"CSA is Selftesting..."<<endl;

integer k=0;

integer value=0;

for(integer i=0;i<n;i++)

{

value=lookup(i);

if(SA[i]!=value)

k++;

}

if(k!=0)

cout<<"CSA is wrong "<<k<<endl;

else

cout<<"CSA is right"<<endl;

}

\*/

//寰楀埌浣嶇疆i鐨勬帓鍚?

integer CSA::Inverse(integer i){

integer RD=this->RD ;

integer anchor=i/RD;

integer p=anchor\*RD;

integer sa=this->RankL ->GetValue (anchor);

while(p<i){

sa=Phi0->GetValue(sa);

p++;

}

return sa;

}

uchar \* CSA::extracting(integer i, integer len){

if(i+len>n){

cout<<"overshoot the length of the Doc."<<endl;

return NULL;

}

uchar \*p=new uchar[len+1];

p[len]='\0';

integer k=0;

i=this->Inverse (i);

for(integer j=0;j<len;j++){

k=this->Phi\_list (i);

p[j]=this->Character (k);

i=Phi0->GetValue(i);

}

return p;

}

integer CSA::Phi\_list(integer i){

integer l=0;

integer r=this->alphabetsize ;

integer m=0;

while(l<r){

m=(l+r)/2;

if(start[m]<=i)

l=m+1;

else

r=m;

}

return r-1;

}

integer CSA::Character(integer i){

return incode[i];

}

integer CSA::blog(integer x){

integer ans=0;

while(x>0){

ans++;

x=x>>1;

}

return ans;

}

integer \* CSA::locating(const char \*Pattern, integer &num){

integer L=0;

integer R=0;

this->Search2(Pattern,L,R);

num=R-L+1;

if(L>R)

return NULL ;

integer \*pos=new integer[num];

Enumerative1(L,R,pos);

return pos;

}

void CSA::Enumerative2(integer L,integer R, integer \*&pos){

integer D=this->D;

InArray \*SAL=this->SAL ;

integer \* distance=new integer[R-L+1];

integer \*pred=new integer[R-L+1];

integer f=0;

integer step=0;

integer q=0;

integer s=0;

integer i=0;

for(integer i=0;i<R-L+1;i++){

pos[i]=0;

pred[i]=-1;

}

for(integer j=L;j<=R;j++){

f=0;

i=j;

step=0;

while(i%D!=0){

i=this->Phi0->GetValue(i);

step++;

if(L<=i&&i<=R){

distance[j-L]=step;

pred[i-L]=j;

f=1;

break;

}

}

if(f==0){

i=i/D;

pos[j-L]=SAL->GetValue(i)-step;

}

}

integer fu=0;

for(integer j=L;j<=R;j++){

if(pos[j-L]!=0){

q=j;

while(pred[q-L]!=-1){

fu++;

s=pos[q-L];

i=pred[q-L];

step=distance[i-L];

pos[i-L]=s-step;

pred[q-L]=-1;

q=i;

}

}

}

delete [] pred;

delete [] distance;

}

void CSA::Enumerative1(integer L,integer R,integer \*&pos){

integer i;

for(i=L;i<=R;i++){

pos[i-L]=lookup (i);

}

}

Save函数

#include"savekit.h"

void savekit::close()

{

if(w!=NULL)

fclose(w);

w=NULL;

}

savekit::~savekit()

{

if(w!=NULL)

fclose(w);

}

savekit::savekit(const char \* file)

{

this->w=fopen(file,"wb");

if(w==NULL)

{

cout<<"Fopen error"<<endl;

exit(0);

}

}

integer savekit::writei64(i64 value)

{

fwrite(&value,sizeof(i64),1,w);

return 1;

}

integer savekit::writeu64(u64 value)

{

fwrite(&value,sizeof(u64),1,w);

return 1;

}

integer savekit::writeinteger( integer value)

{

fwrite(&value,sizeof( integer),1,w);

return 1;

}

integer savekit::writeu32(u32 value)

{

fwrite(&value,sizeof(u32),1,w);

return 1;

}

integer savekit::writei16(i16 value)

{

fwrite(&value,sizeof(i16),1,w);

return 1;

}

integer savekit::writeu16(u16 value)

{

fwrite(&value,sizeof(u16),1,w);

return 1;

}

integer savekit::writei64array(i64 \* value,integer len)

{

fwrite(value,sizeof(i64),len,w);

return 1;

}

integer savekit::writeu64array(u64 \* value,integer len)

{

fwrite(value,sizeof(u64),len,w);

return 1;

}

integer savekit::writeintegerarray(integer \* value,integer len)

{

fwrite(value,sizeof(integer),len,w);

return 1;

}

integer savekit::writeu32array(u32\* value,integer len)

{

fwrite(value,sizeof(u32),len,w);

return 1;

}

integer savekit::writei16array(i16 \* value,integer len)

{

fwrite(value,sizeof(i16),len,w);

return 1;

}

integer savekit::writeu16array(u16 \* value,integer len)

{

fwrite(value,sizeof(u16),len,w);

return 1;

}