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： 基于de Bruijn graph的

基因组片段拼接系统

实验题目： 基于de Bruijn graph的

基因组片段拼接系统

班级：

学号：

姓名：

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

1. 实验目的

将给定的测序片段进行拼接，拼接成一些contigs

1. 实验原理

测序产生的是长度100bp左右的小片段，因而需要把小片段拼接成长度更长些的大片段（contig），进而把大片段拼成基因组,不能直接拼成基因组是因为经常会有某些片段难以被测序得到。

拼接的理论基础是overlap,即同一基因组被随机在不同的位置切断成小片段，所形成的小片段之间大概率会有重合部分，如下图所示

图片包含 图形用户界面

描述已自动生成

利用这些重合的片段，就能找到一条连接起这些测序产生的小片段的正确的路径。

然而，实际上每次测序都有测错的可能，假如两个100bp长度的序列，中间有30个bp的重合，那就需要这两个序列的这30bp序列都没有错误，才能将这两个片段接合起来，但是很明显这样准确率就会很低，因为很难保证测错不发生，所以可以采用动态规划的方法，但动态规划的复杂度就是O（l^2）了，所以可以采取另一种方法 划分成Kmer,然后用debruijn图找到kmer之间的联系

其中，每个kmer作为debruijn图的顶点，该kmer的前k-1mer当作入边，后k-1mer为出边。如果一个kmer的后k-1mer和另一个kmer的前k-1mer相同，那么这两个点就可通过边相连。

数据结构表示如下：

文本

描述已自动生成

我选取长度为15的kmer，数组多了1位大概是因为字符串后面需要’/0’。

为了方便的找到每个kmer的位置，我采用了散列的方法存储kmer，即将其化为四进制的整数，再表示为10进制的整数。理想情况下，散列表可以有4的15次方比特的内存，但是由于笔记本电脑的功能限制，我选取散列大小为99999983。

如果一个kmer被投射到一个已经被占据的位置，它就被投射到该位置链接到的下一个位置，即nextnode指向的位置。

之后便是debruijn图的构建过程，对每个kmer找到能指向它的kmer和它指向的kmer。特别的，如果一个kmer没有其他kmer能指向它，那么就表示它大概率是一个contig的起始kmer。

之后从每一个起始kmer开始，寻找与它相邻的kmer，最后形成一条哈密顿路，就是contigs。

一些问题：

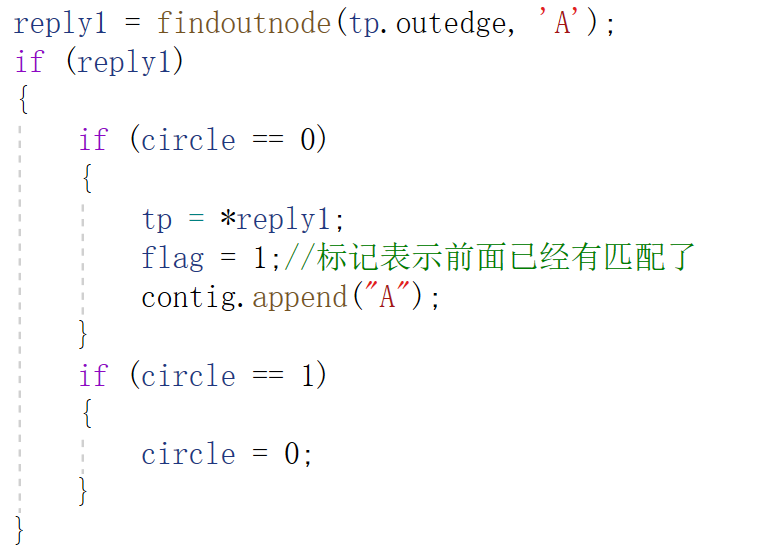
1. 从原始数据到kmer

利用jellyfish软件进行处理，将原始测序片段切成15mer的大小。并统计15mer出现的频率。

1. 重复片段的影响

在生物基因组中，存在着很多重复片段，这些片段在debruijn图中会形成环路，从而容易使程序陷入死循环。最简单的例子，对于序列AAAAAAAAAAAAAAAAA 20个A，那么就会有5个kmer都为全A 因而表现在debruijn图中就是该kmer具有指向自己的入边和出边。假如我们的程序设定遍历顺序是先沿下一个字母为A的出边遍历，那程序就会一直做全是A的死循环，所以为了避免出现这样的情况，我设置一个容器来存放遍历到的多入边的顶点。如果遍历到的多入边的顶点在容器中，则换条路径；如果不在容器中，则加进来。

代码实现如下：



r

1. tips

tips是指由于测序错误等因素引起的从正确的contigs分支出较短的旁路，这一般都是由测序错误引起的，所以应该去掉。具体实施的方法呢，就是在每个具有多条出边的顶点处，走其中一条路的时候将另外的路压入栈，等到遍历完这条路的时候再取出其他路来走。如果某条路上顶点个数过少，那么就舍掉这个顶点。

1. 测试结果及分析



生成的contigs如图

1. 经验体会

通过这次实验，我体会到了用kmer构建debruijn图并完成基因组拼接的过程，尤其意识到了重复序列对于拼接过程的重大影响，同时，也深刻体会到了采用计算kmer的工具对实验的巨大便捷作用。

1. 附录：源代码（带注释）

#define \_CRT\_SECURE\_NO\_WARNINGS

#include<cstdlib>

#include<stdio.h>

#include <iostream>

#include<stack>

#include <fstream>

#include <string>

#include <set>

using namespace std;

#define kmerlength 15

#define hashsize 99999983

#define N 268435456

void buildhash();

void builddbg();

void searchdbg();

long chartodata(char a[15]);

void char\_int(char a[15], int num[15]);

void mystrncpy1(char m[14], char n[15]);

void mystrncpy2(char m[14], char n[15]);

void mystrncpy3(char m[15], char n[15]);

int findinnode(char k[14], char a);

void save(string contig);

typedef struct node

{

char inedge[15];

char outedge[15];

char nodeinfo[16];

//注：这个有很多冗余，不过能比较好的体现出debrujin图的形式

//边的长度正常应该是14，但是运行结果不对，我也不知道为啥，特别邪门

struct node\* nextnode=NULL;

int ifmultiin=0;

}node;

node\* findoutnode(char k[14], char a);

node\* find(char a[15]);

typedef struct stateinfo

{

long order;

node\* position=NULL;

}stateinfo;

node\* hashset[hashsize];

stack <node> s;

stack <node> start;

int main()

{

buildhash();

printf("finished buildhash!\n");

builddbg();

printf("finished builddbg!\n");

searchdbg();

}

string contig;

ofstream f;

void searchdbg()

{

node first;

node tp,\*reply1=NULL,\*reply2=NULL,\*reply3=NULL,\*reply4=NULL;

int flag = 0;

int endmark = 0;

int lun=0;

int circle = 0;

f.open("result.txt", ios::out);

while (!start.empty())

{

first = start.top();

start.pop();

contig = first.nodeinfo;

tp = first;

long ordernumber = 15;

set<long> bubbleend;

set<long> outnode;

stack<stateinfo>savestack;

stateinfo sd = { 0,NULL };//这个变量名没啥意义，瞎起的

int inbubble = 0;

long bubblestart;

int stackdeal = 0;

endmark = 0;

flag = 0;

while (!endmark)

{

flag = 0;

ordernumber++;

if (tp.ifmultiin == 1)

{

bubbleend.insert(chartodata(tp.nodeinfo));

}

if (savestack.size() > 1000)

{

stackdeal = 1;

break;

}

reply1 = findoutnode(tp.outedge, 'A');

if (reply1)

{

if (circle == 0)

{

tp = \*reply1;

flag = 1;//标记表示前面已经有匹配了

contig.append("A");

}

if (circle == 1)

{

circle = 0;

}

}

reply2= findoutnode(tp.outedge, 'C');

if (reply2)

{

if (circle == 0)

{

if (flag == 0||stackdeal==1)

{

flag = 1;

contig.append("C");

outnode.insert(chartodata(tp.nodeinfo));

tp = \*reply2;

}

else

{

stateinfo q;

q.order = ordernumber;

q.position = reply2;

savestack.push(q);

}

}

if (circle == 1)

{

circle = 0;

}

}

reply3 = findoutnode(tp.outedge, 'G');

if (reply3)

{

if (circle == 0)

{

if (flag == 0||stackdeal==1)

{

flag = 1;

contig.append("G");

outnode.insert(chartodata(tp.nodeinfo));

tp = \*reply3;

}

else

{

stateinfo q;

q.order = ordernumber;

q.position = reply3;

savestack.push(q);

}

}

if (circle == 1)

{

circle = 0;

}

}

reply4 = findoutnode(tp.outedge, 'T');

if (reply4)

{

if (flag == 0||stackdeal==1)

{

flag = 1;

contig.append("T");

outnode.insert(chartodata(tp.nodeinfo));

tp = \*reply4;

}

else

{

stateinfo q;

q.order = ordernumber;

q.position = reply4;

savestack.push(q);

}

}

if (strcmp(tp.nodeinfo, "TTTTTTTTTTTTTTT")== 0)

{

endmark = 1;

}

if (!reply1 && !reply2 && !reply3 && !reply4)

{

if (!savestack.empty())

{

sd = savestack.top();

savestack.pop();

tp = \*(sd.position);

}

endmark = 1;

if (inbubble == 0)

{

if (contig.length() > 100)

{

lun++;

save(contig);

}

//保存contig

string().swap(contig);

inbubble = 1;

}

else

{

//保存tips

string().swap(contig);

}

}

if (inbubble == 1&&outnode.find(chartodata(tp.nodeinfo))==outnode.end())

{

if (bubbleend.find(chartodata(tp.nodeinfo)) != bubbleend.end())

{

//保存bubble

}

}

if (inbubble == 1 && outnode.find(chartodata(tp.nodeinfo)) != outnode.end())

{

circle = 1;

}

}

f << lun << endl;

}

}

node\* findoutnode(char k[14], char a)

{

char b[15];

b[14] = a;

node\* x;

for (int j = 0; j < 14; j++)

{

b[j] = k[j];

}

x = find(b);

return x;

}

node\* find(char a[15])

{

long b;

node\* k = NULL;

b=chartodata(a);

b = b % hashsize;

k = hashset[b];

if (k == NULL)

{

return NULL;

}

else

{

if (strcmp(k->nodeinfo, a))

{

return k;

}

else

{

while (!strcmp(k->nodeinfo, a) && k)

{

k = k->nextnode;

}

if (k == NULL)

{

return NULL;

}

else

{

return k;

}

}

}

}

int findinnode(char k[14], char a)

{

char b[15];

b[0] = a;

node\* x;

for (int j = 0; j < 14; j++)

{

b[j + 1] = k[j];

}

x=find(b);

if (x == NULL)

{

return 0;

}

else

{

return 1;

}

}

void builddbg()

{

node temp;

int flag1,flag2,flag3,flag4;

while (!s.empty())

{

temp = s.top();

s.pop();

flag1=findinnode(temp.inedge, 'A');

flag2=findinnode(temp.inedge, 'C');

flag3=findinnode(temp.inedge, 'G');

flag4=findinnode(temp.inedge, 'T');

if(!flag1&&!flag2&&!flag3&&!flag4)

{

start.push(temp);

}

if (flag1 + flag2 + flag3 + flag4 >= 2)

{

temp.ifmultiin = 1;

}

}

}

void buildhash()

{

string line;

char buffer[16];

string filename("kmerdata.txt");

ifstream input\_file(filename);

long value = 0;

long hashvalue;

hashvalue = value;

while (getline(input\_file, line))

{

strcpy(buffer,line.data());

value = chartodata(buffer);

hashvalue = value % hashsize;

node\* newnode = new node;

if (hashset[hashvalue] != NULL)

{

node\* p = hashset[hashvalue];

while (p->nextnode)

{

p = p->nextnode;

}

p->nextnode = newnode;

mystrncpy1(newnode->inedge, buffer);

mystrncpy2(newnode->outedge, buffer);

mystrncpy3(newnode->nodeinfo, buffer );

}

else {

mystrncpy1(newnode->inedge, buffer);

mystrncpy2(newnode->outedge, buffer);

mystrncpy3(newnode->nodeinfo, buffer);

hashset[hashvalue] = newnode;

}

s.push(\*newnode);

}

}

void mystrncpy1(char m[14], char n[15])

{

for (int i = 0; i < 14; i++)

{

m[i] = n[i];

}

}

void mystrncpy2(char m[14], char n[15])

{

for (int i = 0; i <14; i++)

{

m[i] = n[i+1];

}

}

void mystrncpy3(char m[15], char n[15])

{

for (int i = 0; i < 16; i++)

{

m[i] = n[i];

}

}

void char\_int(char a[15], int num[15])

{

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

{

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

num[i] = 0;

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

num[i] = 1;

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

num[i] = 2;

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

num[i] = 3;

}

}

long chartodata(char a[15])

{

long temp = 0;

int num[kmerlength];

char\_int(a, num);

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

{

temp = temp + num[i] \* pow(4, 14 - i);

}

return temp;

}

void save(string contig)

{

f << contig << endl;

}