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

将大肠杆菌的测序片段比对到参考基因组上

1. 实验原理

通过kmer索引的方式，先将测序片段（70bp）拆分成几个15bp的小段，称为seed.再用这些15bp的小段和原基因组进行比对，如果有两段甚至更多段比对到了参考基因组的同一个位置，那么就有很大的概率说明新的基因组和原基因组在这里是相同的。如果最多只有一段比对到了基因组上的一个位置，那就分别比较这些seed映射到的基因组位置和需要查询的串的编辑距离，选取编辑距离最短的位置。如果有超过两个位置的编辑距离都足够短，我们就很难分清到底应该选取哪个，因而返回：未能确定结果。

2.1 SNAP的优点

局部比对有很多种方法，比如利用动态规划的方法，将经历过的负分置为0，也可以用索引的方法。但是这些算法都不可避免的要有较高的时间复杂度（比如O(n^2)），这是由于问题本身的性质决定的，基因组之间可以发生碱基的突变，插入，删除，和大块区域的移位，倒位，异位等。这导致了很多算法实际上是当作两个不相干的字符串进行对比处理。但实际上对于一些比较相近的基因组，比如人和人的基因组，变异相对来说是比较少的，因而可以采用一种不是那么精确的启发式算法来试图解决一种“建立在基因组很相似的假设上寻找映射和不同的方法 ”，因而SNAP应运而生。

2.2索引

图形用户界面, 应用程序

描述已自动生成

首先建立哈希表，将原基因组拆成15mer，映射到散列表的相应位置（15mer是因为要处理的是大肠杆菌基因组，如果是人的基因组应该是20mer，但是20mer就需要4^20B的空间，我这台普通笔记本是难以承受的）

图形用户界面, 文本, 应用程序

描述已自动生成

节点的结构如图，每个kmer都有可能出现在不同位置，因而采用1个链表来记录每个kmer出现的位置。其中num是用来描述kmer一共出现了几次的。

图形用户界面

中度可信度描述已自动生成

如图为哈希表的结构。

2.3延申选择

日程表

描述已自动生成

一个很巧妙的思维就是鸽笼原理，比如一个两个片段的差异为3，那么如果把这两个片段拆分成4个kmer，则至少有1个kmer一定能对应上，因此SNAP算法就保证了一定能找到70/15=4个编辑距离之内的两个片段。

但是就是有片段匹配上还要考虑到错误匹配的情况，毕竟相似的片段在很多地方可能都有存在。

为了兼顾时间性能与准确性，采用了贪心的方法。

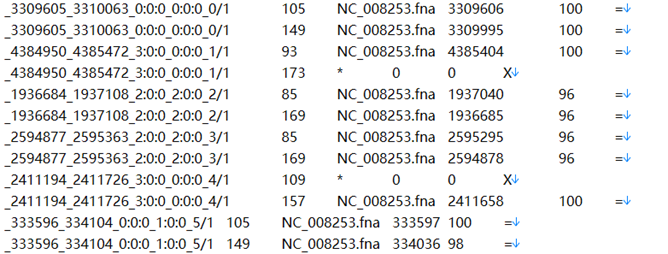
如果有的kmer在基因组出现次数过多，那么直接舍掉不用，因为它会占用比较大的时间开销还非常容易得不到好结果。如果有几个kmer投射到了相同的地方，那就大概率采用这个地方。如果每个kmer都只投射到了一个地方，那就都延申，然后选取最优的结果。但是为了贪心的弊端就是容易出现现在好但是以后不好的结果，于是就保留第二好的结果。

具体算法如下：

图形用户界面, 文本, 应用程序

描述已自动生成

1. 测试结果及分析



如图为输出的sam文件。

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。

1. 经验体会

通过这个实验，我体会到了序列对比的精妙思想，也更加深刻的理解了包括全局对比，拼接等算法的思想，以及彼此之间的不可替代之处，也感受到了snap算法的精髓

1. 附录：源代码（带注释）

#define \_CRT\_SECURE\_NO\_WARNINGS

#include<iostream>

#include<string.h>

#define MAXHASHSIZE 999983

#define confidencenum 2

#define SIZEOFKMER 15

#define SIZEOFREAD 70

#define SEEDTOTRY 6

#define MAXIMUNDIATANCE 4

#define CONFIDENCETHRESHOLD 1

#define MAXHITS 10

#define AVERYLARGENUMBER 100000

typedef struct locate

{

int num;

locate\* nextlocate;

}locate;

typedef struct Node

{

struct Node\* next;

long origndata;

locate\* location;

int num=0;

}node;

typedef struct Hashset

{

node\* location;//kmer在哈希表中的位置

}Hash;

Hash hash[MAXHASHSIZE] = { NULL };

char\* getdata(char\* fname);

char\* str;

void createhash();

char\* getdata(const char\* fname);

long changebasetonum(char a[]);

long pow4(int k);

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

int altersingle(char m);

node\* findposition(char a[20]);

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

int max(int a, int b, int c);

int max2(int a, int b);

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

int\* snap(char\* s);

void SNAP(const char\* file1,const char\* file2,const char\* file3);

int main()

{

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

createhash();

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

}

char\* getdata(const 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;

}

long changebasetonum(char a[])

{

long num=0;

long chengfang=0;

for (int k = 1; k <= 15; k++)

{

chengfang = pow4(15 - k);

num = num + altersingle(a[k]) \* chengfang;

}

return num;

}

long pow4(int k)

{

int i;

int b = 1;

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

{

b = b \* 4;

}

return b;

}

int altersingle(char m)

{

if (m == 'A')

return 0;

if (m == 'C')

return 1;

if (m == 'G')

return 2;

if (m == 'T')

return 3;

}

void createhash()

{

int i, j;

char a[15];

long num=0;

node\* newmer;

long num2=0;

for (i = 0; i < 4938900; i++)

{

for (j = 0; j <15; j++)

{

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

}

num = changebasetonum(a);

num2 = num % MAXHASHSIZE;

newmer = (node\*)malloc(sizeof(node));

newmer->origndata = num;

newmer->next = NULL;

if (hash[num2].location== NULL)

{

hash[num2].location = newmer;

}

else

{

node\* p = hash[num2].location;

node\* q = p;

while (p->next != NULL&&p->origndata!=newmer->origndata)

{

q = p;

p = p->next;

}

if (p->origndata == newmer->origndata)

{

p->num++;

locate\* j = new locate;

if (newmer->location == NULL)

{

j->num = i;

newmer->location = j;

}

else

{

j = newmer->location;

locate\* k = j;

while (j->nextlocate != NULL)

{

k = j;

j = j->nextlocate;

}

j->num = i;

k->nextlocate = j;

}

}

else {

q->next = newmer;

}

}

}

}

node\* findposition(char a[20])

{

long num = 0;

int num2 = 0;

num=changebasetonum(a);

num2 = num % MAXHASHSIZE;

if (hash[num].location == NULL)

{

return NULL;

}

else

{

node\* p1 = hash[num].location;

while (p1->origndata != num)

{

p1 = p1->next;

}

if (p1 == NULL)

{

return NULL;//没找到

}

return p1;

}

}

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;

}

int max(int a, int b, int c)

{

int temp = a;

if (b > a)

{

temp = b;

}

if (c > temp)

{

temp = c;

}

return temp;

}

int max2(int a, int b)

{

if (a > b)

return a;

else return b;

}

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

{

int space[71][71];

int maxnum = 0;

for (int i = 0; i < 101; i++)

{

space[i][0] = 0;

space[0][i] = 0;

}

for (int j = 1; j < 101; j++)

{

for (int k = 1; k < 101; k++)

{

if (a[j] == b[k])

{

space[j][k] = max(space[j - 1][k], space[j][k - 1], space[j - 1][k - 1] + 1);

if (space[j][k] > maxnum)

{

maxnum = space[j][k];

}

}

else

{

space[j][k] = max2(space[j][k - 1], space[j - 1][k]);

}

}

}

if (100 - maxnum > dlimit)

{

return 10000;

}

else return 100 - maxnum;

}

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

{

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

{

if (seedshitting[i] == location)

{

return i;

}

}

return -1;

}

int\* snap(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 = findposition(S);

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

{

nonover++;

}

if (q == NULL)

{

allseedhad = 1;

}

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

{

allseedhad = 1;

locate\* l= q->location;

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

{

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

if (f == -1)

{

seedshitting[numofhit] = l->num - 10 \* i;

seedshittings[numofhit] = 1;

numofhit++;

}

else

{

seedshittings[f]++;

}

l = l->nextlocate;

}

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 SNAP(const char\* file1,const char\* file2,const 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 = snap(s);

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

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

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

f2 = snap(s2);

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

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

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

f3 = snap(rs);

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

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

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

f4 = snap(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);

}