**哈尔滨工业大学计算学部**

**课程实验报告**

课程名称：生物信息学

课程类型：选修

项目名称：基因组测序片段比对与映射

项目名称：系统设计与实现

班级：2103601

学号：2021112845

姓名：张智雄

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

1. **实验目的**

给定基因组序列（大肠杆菌E. Coli, 长度~5Mbp）作为参考基因组和一定数量来自于大肠杆菌的基因组测序片段（1000条，70 bp/read）作为输入，建立算法和系统，完成基因组测序片段比对，重构片段位置（基因组坐标）和基因组片段与参考序列的差异。

1. **实验原理**

2.1 双端测序（pair end reads）

双端测序(pair end reads)是一种广泛应用于DNA测序的技术，其特点在于同时从DNA分子的两端进行测序，从而获得两个相互重叠的读取片段。这种测序策略不仅能够提高测序覆盖度和准确度，还能够增强在基因组组装和变异检测等应用中的结果可靠性。通过配对两端的读取片段，可以更有效地捕获DNA分子的完整信息，减少测序误差，并提高对基因组中复杂区域的解析能力。

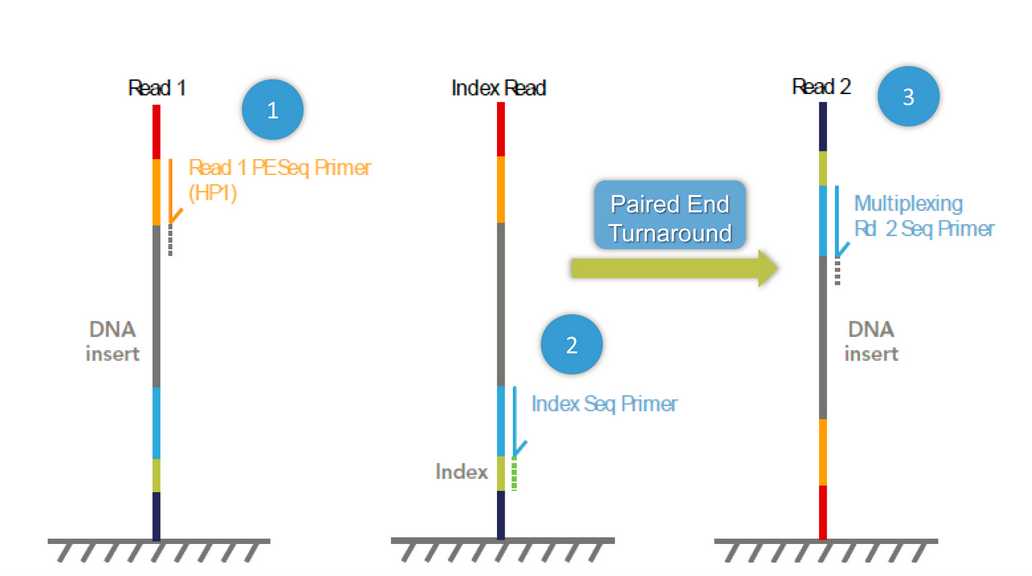


图1 双端测序

此外，Pair-end reads技术的应用范围还包括转录组学和表观遗传学领域，为研究基因表达和表观遗传变化提供了更为全面的数据支持。

2.2 Bowtie绝对比对

Bowtie是一种常用的快速和高效的序列比对算法，主要用于将短序列（例如短于50-100个碱基对的DNA片段）与参考基因组进行比对。Bowtie绝对比对是指利用Bowtie进行DNA序列的绝对匹配比对，即精确地找到参考基因组中与查询序列完全匹配的区域。

该算法的基本原理是在Burrows-Wheeler转换后的序列中寻找与查询序列完全匹配的子序列。这一过程通过迭代计算查询序列的后缀来实现。首先，算法从查询序列的最后一个字符开始，并在转换后的序列中寻找以该字符为前缀的行的范围。随着迭代的进行，逐渐增加查询序列后缀的长度，并不断更新匹配的范围。这一过程在每一步中要么缩小匹配范围，要么保持不变。

**2.2.1 Bowtie算法具体过程**

这个算法的核心思想是利用LF-mapping性质，即在Burrows-Wheeler转换后的序列中，第一列中字符的顺序与最后一列中相同字符的顺序一致。这一性质保证了通过转换矩阵的排列，我们能够准确地确定最后一列中每个字符的第i次出现对应于第一列中第i次出现的相同字符。这是因为Burrows-Wheeler转换的本质是对后缀数组进行排序，所以这种对应关系是显而易见的。

* 函数在这一背景下的作用是，给定BWT中第r个字符是某字符c的第i次出现，该函数返回SA（Suffix Array，后缀数组）中第i次出现的字符c的位置。
* 函数则用于进行DNA序列的绝对匹配，它通过输入字符c和位置r来返回新区间，从而实现高效的搜索。

文本

描述已自动生成图形用户界面, 应用程序

描述已自动生成

而后我们使用EXACTMATCH的函数来实现绝对匹配的功能，它接受一个字符串作为输入，并返回一个包含两个整数的元组，它们分别表示在Burrows- Wheeler转换中Q的绝对匹配的起始位置和结束位置，具体实现原理如下：

* 首先，函数通过调用字符串长度函数来获取输入字符串Q的长度，并将其存储在变在中。
* 然后，函数将变量设置为的最后一个字符，并使用预先计算的变量获取转换矩阵中以开头的行的数量。
* 接下来，函数使用变量和来表示可能包含的匹配的区间的起始和结束位置，其中初始化为，ep初始化为。
* 然后，函数通过一个循环来从的最后一个字符开始，向前遍历所有字符，并使用函数（由和当前位置决定的函数）来缩小可能包含的匹配的区间。在每次迭代中，函数将设置为当前位置的字符，并使用和分别更新和。如果，并且，则继续循环，否则函数返回元组，其中和表示可能包含的匹配的区间的起始和结束位置。

该函数的主要功能是**找到在Burrows-Wheeler转换中Q的绝对匹配的起始位置和结束位置**，从而实现对DNA序列的精确匹配。它使用了LF-Mapping算法和LFC函数来提高查询效率，并且对于大规模DNA序列数据具有高效性和可扩展性。

图片包含 图形用户界面

描述已自动生成

最后我们需要找到这个序列的起始位置，找到对应的后缀起始位置有多种策略。一种简单的策略是对结果集合中的每一个元素进行回溯，直到回溯到为止，就可以得到相应的起始位置。另一种方法是利用一个额外的数组，即数组来进行存储，该数组将后缀数组中的每个位置映射回其对应的后缀的起始位置。这种方法需要额外的空间来存储数组，但可以提高查找效率。

而本实验则采用了一种更节省空间的算法，即**每隔一段距离记录一次起始位置**。在这种算法中，如果搜索结果没有与记录的起始位置相符，就进行回溯，并记录回溯的次数，直到搜索结果与记录的位置相符为止。虽然这种算法需要更少的时间，但可能需要更多的额外空间，具体如下：

图片包含 日历

描述已自动生成

选择合适的策略取决于具体的应用场景和资源限制。

2.3 含有替换错误的比对

Burrows-Wheeler范围适用于逐步扩展的查询后缀。当搜索到达一个空的Burrows-Wheeler范围时（这表明相应的后缀在文本中不存在），算法可能选择一个之前已检查的查询位置并在此处替换不同的基础，从而在该位置产生一个假设的不匹配。这一过程被称为“回溯”。执行回溯后，精确匹配搜索从替换位置左侧继续。搜索仅执行与用户可配置的对齐策略相一致的回溯操作。

举例来说，如果对齐策略规定整个对齐过程中最多存在两处不匹配，搜索过程中在搜索空间的位置已经假设了两个不匹配，且在处获得一个空的Burrows -Wheeler范围，则搜索不会尝试再假设第三个不匹配，因为这将违反对齐策略。这被称为“策略剪枝”。另外，搜索过程永远不会回溯至与空的Burrows-Wheeler范围关联的搜索空间点。这是因为范围的空缺意味着参考文献中不可能存在假设类型的对齐。这被称为“空范围剪枝”。

首先，应用精确查找算法来确定序列在目标基因组中的出现位置。如果在某个碱基位置上找不到任何候选匹配位置，那么算法将回退到前一个碱基，并尝试用其他碱基替换当前碱基。接着，算法重新进行精确查找，重复这一过程直到找到一个完全匹配的序列。在搜索过程中，如果某个分支的累计错误数量超过了设定的上限阈值，那么算法将放弃继续探索该分支，从而提高搜索效率。

图示, 工程绘图

描述已自动生成

图 2 含有替换错误的比对

因此为了修正是上述的替换错误，使用函数在给定的字符串范围内，最多进行 max\_changes 次字符更改，以找到一个与目标字符串匹配的字符串，其返回类型为 ，即包含两个整数的 pair。

在这一过程中，函数会利用名为 EXACTMATCH 的函数来检查当前字符串与目标字符串是否匹配。如果找到了匹配的字符串，函数将返回该字符串在原始字符串中的起始和结束位置。在函数的实现中，它会遍历指定范围内的字符串，并尝试将当前字符替换为 DNA 序列中的四种碱基（A, C, G, T）。对于每种可能的字符替换，函数会递归调用自身来处理下一个字符。如果在递归过程中找到了匹配的字符串，函数会立即返回匹配的起始和结束位置。如果在允许的最大更改次数内未找到匹配的字符串，函数将返回 。

文本

描述已自动生成

1. **测试结果及分析**

本实验尝试了两种不同的方法来实现 occ 函数。一种是通过一直回溯的方式，另一种是设置 checkpoints 的方法。两种方法的目标都是计算字符 c 在 Burrows-Wheeler Transform（BWT）中位置 r 之前出现的次数。

**方法一：**简单回溯这种方法的实现相对简单，直接遍历 BWT，统计位置 r 之前字符 c 出现的次数。这种方法的时间复杂度较高，需要遍历整个 BWT，导致运行速度较慢。

**方法二：**设置 checkpoints这种方法在算法实现中使用了 checkpoints，通过预先计算并存储某些位置上字符出现的次数，以空间换时间，降低时间复杂度。当需要计算位置 r 之前字符 c 出现的次数时，可以直接查找最近的一个 checkpoint，然后从 checkpoint 开始计算字符 c 的出现次数。这种方法避免了完全遍历 BWT，大大提高了运行速度。

通过设置 checkpoints，牺牲部分存储空间，但换取了更高的运行效率，从而在实际应用中提高了算法的性能。

以下是测试的结果以及分析，本实验在约500w个碱基中搜索约14w对70碱基长的read pairs，基本上除了编辑距离过大的序列，其他序列都能模糊匹配到参考基因组，编辑距离0-5不等。

文本

描述已自动生成

图 3 实验结果

1. **经验体会**

这次实验让我深入了解了 Bowtie 算法和 BWT（Burrows-Wheeler Transform）的原理。通过学习这种高效的基因组比对算法，我加深了对生物信息学的认识。

同时，实验中的编程任务不仅提高了我的编程能力，还让我学会了如何调试代码。在解决实验中遇到的问题时，我逐步培养了耐心和细心，并通过不断尝试和改进，掌握了解决问题的方法和技巧。

最后，实验中遇到的困难和挑战促使我积极寻求解决方案。在搜索国外论坛的过程中，我发现了很多有价值的信息和经验分享。这些资料对于解决问题和拓宽视野都有很大帮助。

1. **附录：源代码（带注释）**

**main.cpp**

|  |
| --- |
| #include <iostream>  #include <fstream>  #include <string>  #include <vector>  #include <bitset>  #include <cstdlib>  #include "LFindex.h"  int\* BWT = NULL;  int C[5] = {0, 0, 0, 0, 0};  using namespace std;  int num[4] = {0, 0, 0, 0};  int mark[5000][4];  /\* i: || 0 -- A||1 -- C||2 -- G|| 3--T || \*/  /\* C[i]: the index just 1 less than the start position, that is to say C[0] = 0, C[1] = 1 + num of A and so forth \*/  int main() {  ifstream BWT\_file;  char cc;  BWT\_file.open("F:\\bioinfo\\exp2\\BWT.txt");  int cnt = 0;  //count the number of each character  while(BWT\_file >> cc){  cnt++;  if(cc == '$'){  continue;  }else{  if(cc == 'A') num[0]++;  else if(cc == 'C') num[1]++;  else if(cc == 'G') num[2]++;  else if(cc == 'T') num[3]++;  }  if(cnt % 1000 == 0){  mark[cnt / 1000][0] = num[0];  mark[cnt / 1000][1] = num[1];  mark[cnt / 1000][2] = num[2];  mark[cnt / 1000][3] = num[3];  }  }  BWT\_file.close();  //calculate the start position of each character  for(int i = 1; i <= 4; i++){  C[i] = C[i - 1] + num[i - 1];  }  //read the BWT file again and store the BWT in the array BWT  BWT\_file.open("./NC\_008253.se.5end.forwatd.fq");  if(BWT\_file.is\_open())  cout << "BWT file is open" << endl;  else{  cout << "BWT file is not open" << endl;  return 1;  }  BWT = new int[cnt];  int i = 0;  while(BWT\_file >> cc){  if(cc == 'A') BWT[i] = 0;  else if(cc == 'C') BWT[i] = 1;  else if(cc == 'G') BWT[i] = 2;  else if(cc == 'T') BWT[i] = 3;  else BWT[i] = 4;  i++;  }  BWT\_file.close();  ifstream Q\_file;  Q\_file.open("./map.sam");  PII res = {0, 0};  string Q ;  EXACTMATCH("AGCTTTTCATTCTGACTGCAACGGGCAATATGTCTCTGTGTGGATTAAAAAAAGAGTGTCTGATAGCAGC");  cout << res.first << " " << res.second << endl;  return 0;  } |

**LFindex.cpp**

|  |
| --- |
| #include <iostream>  #include <fstream>  #include <string>  #include <vector>  #include <bitset>  #include <cstdlib>  #include <algorithm>  #include <utility>  using namespace std;  bool flag = false;  typedef std::pair<int, int> PII;  extern int mark[5000][4];  extern int \*BWT;//the original declaration is in main.cpp(extern int\* BWT;)  extern int C[5]; //the original declaration is in main.cpp(extern int C[5] = {0, 0, 0, 0,0};)  int Occ(int c, int r){  //naive way to implement Occ, need to be improved  int count = 0;  int index = r / 1000;  count = mark[index][c];  for(int i = index \* 1000 + 1; i < r - 1; i++){  if(BWT[i] == c){  count++;  }  }  return count;  }  int LF(int r){  int c = BWT[r];  return C[c] + Occ(c, r) + 1;  }  int LFC(int r, int c){  return C[c] + Occ(c, r) + 1;  }  PII EXACTMATCH(string Q){  int q = Q.length();  int\* QN = new int [q];  for(int i = 0; i < q; i++){  if(Q[i] == 'A') QN[i] = 0;  else if(Q[i] == 'C') QN[i] = 1;  else if(Q[i] == 'G') QN[i] = 2;  else if(Q[i] == 'T') QN[i] = 3;  }  int c = QN[q - 1];  int sp = C[c] + 1;  int ep = C[c + 1];  int i = q - 2;  while(sp <= ep && i >= 0){  c = QN[i];  sp = LFC(sp, c);  ep = LFC(ep, c);  if(sp == ep && i != 0){  delete [] QN;  flag = false;  return{i, i} ;  }  i--;  }  delete [] QN;  flag = true;  return {sp, ep};  }  const char bases[] = {'A', 'C', 'G', 'T'};  vector<PII> modified\_positions;  PII replace\_char(string &s, int index, int max\_changes, int range\_start, int range\_end, int changes = 0) {  if (changes == max\_changes) {  return {-1, -1};  }  PII res = EXACTMATCH(s);  if (res.first != -1 && res.second != -1) {  return res;  }  char bases[] = {'A', 'C', 'G', 'T'};  for (int i = range\_start; i <= range\_end; i++) {  for (int j = 0; j < 4; j++) {  if (s[i] != bases[j]) {  char tmp = s[i];  s[i] = bases[j];  modified\_positions.push\_back({i, j});  PII result = replace\_char(s, index + 1, max\_changes, range\_start, range\_end, changes + 1);  if (result.first != -1 && result.second != -1) {  return result;  }  s[i] = tmp;  modified\_positions.pop\_back();  }  }  }  return {-1, -1};  }  PII blurred\_match1(string Q) {  int a, b, c = 0;  int aj, bj, cj = 0;  char CH[4] = {'A', 'C', 'G', 'T'};  int n = Q.length();  vector<PII > wrong\_pos;  PII res = {0, 0};  res = EXACTMATCH(Q);  if (flag == true) {  return res;  }  int ed = res.second;  if (!flag) {  int i = 0;  int mid = n << 2;  /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  /\*Case 1: the left part has up to 3 errors, while the right part has no errors\*/  /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  replace\_char(Q, 0, 3, 0, n - 1);  /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  /\*Case 2: the left part has more than 1 errors and up to 3 errors while the right part has no errors\*/  /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  replace\_char(Q, 0, 3, 0, n - 2);  /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  /\*Case 3: the left part has more than 1 errors and up to 2 errors while the right part has exactly one error\*/  /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  replace\_char(Q, 0, 2, 0, n - 2);  /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  /\*Case 4: the left part has exactly 2 errors while the right part has exactly one error\*/  /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  replace\_char(Q, 0, 2, 0, n - 3);  }  }  string read\_reverse(string read) {  //reverse the read  reverse(read.begin(), read.end());  for(int i = 0; i < read.length(); i++) {  switch (read[i]) {  case 'A':  read[i] = 'T';  break;  case 'C':  read[i] = 'G';  break;  case 'G':  read[i] = 'C';  break;  case 'T':  read[i] = 'A';  break;  default:  cout << "只有AGCT被支持!" << endl;  }  }  return read;  } |