**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：大规模基因组序列表示与索引

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, 长度约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，并且掌握文件操作、位操作。

* 完成对所有后缀的直接排序；
* 自索引结构可以选择CSA或者BWT；
* 将直接排序与其他自索引构建算法比较计算速度、内存等。

1. **实验原理**

2.1 碱基序列的存储

由于基因组序列通常仅由四种碱基组成：腺嘌呤(A)、胞嘧啶(C)、鸟嘌呤(G)和胸腺嘧啶(T)，所以我们只需使用两个比特位(bit)来存储碱基，从而节省存储空间。在本实验中，我们分配了一个无符号长整型数组，将每32个碱基共64个比特位存储在一个无符号长整型变量（uint64\_t）中。

此过程由uint64\_t encode(char c)函数完成，其具体功能即将一个给定的字符c（取值只能为A、C、G、T）按下表转换为对应的二进制码表示，否则返回-1表示非法字符。

表格 1 碱基编码

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 字符 | A | C | G | T | 其他 |
| 对应编码 | 00 | 01 | 10 | 11 | 非法字符 |

整个的读取过程由void C2B(uint64\_t\* addr, std::ifstream &infile)函数完成，其接收一个指向存储结果的数组指针addr和一个输入文件流infile。函数从输入文件流中读取DNA序列，然后将序列转换为二进制表示，存储到addr数组中。

具体来说，使用std::getline()函数从输入文件流中读取一行DNA序列，将其存储在名为cur\_row的字符串变量中。然后，通过一个嵌套的for循环遍历cur\_row中的每个字符，使用encode()函数转换为2位二进制表示，存储在cur\_char变量中。然后将cur\_char从高位向低位依次填充到一个uint64\_t类型的变量cur中，如果当cur中64位都已被cur\_char填充，则将其存储到addr数组中。否则将当前字符的二进制编码左移两位后与cur进行按位或操作。

而在处理完整个DNA序列后，如果cur仍然保留有未处理的二进制位数，则会左移相应的位数，然后存储到addr数组中。

而要从数组中提取两个比特位对应的碱基，可以通过掩码操作来实现。首先，该函数计算出存储要提取字符的uint64\_t类型变量在addr数组中的索引，以及在该变量中存储要提取字符的位置。具体来说，由于一个uint64\_t类型变量可以存储32个字符的二进制表示，因此和可以计算：

然后，函数从addr数组中读取存储要提取字符的uint64\_t类型变量，并将其存储到变量code中。接下来，函数将code左移个二进制位，再将其右移62个二进制位，清除code中除了要提取的字符之外的二进制位，以获得存储要提取字符的二进制码，按同样的规则即可将二进制码还原为一个碱基字符。

2.2 后缀直接排序

给定一个DNA序列，其中DNA序列以二进制形式存储在数组addr中，同时给定该DNA序列对应的后缀数组SA。该算法旨在对后缀数组进行排序，使得后缀按字典序从小到大排列。

函数extract\_val(uint64\_t\* addr,int idx)的作用是从数组addr中提取一段二进制数值，并将其转换为一个无符号整数。该函数接受两个参数：数组指针addr和整数idx，表示需要提取的二进制数值在DNA序列中的位置。函数首先计算出存储要提取的二进制数值的uint64\_t类型变量在addr数组中的索引以及在该变量中存储要提取的二进制数值的位置。

1. 如果，则函数直接返回；
2. 否则，函数分别从和中提取要求的二进制数值，并将它们合并成一个uint64\_t类型的变量res。

函数compare(uint64\_t\* addr,int i,int j)的作用是比较后缀数组中两个后缀在字典序上的大小关系。函数首先调用extract\_val(addr, i)和extract\_val(addr, j)函数，分别提取后缀数组中第个和第个后缀所表示的二进制数值。

1. 如果这两个二进制数值相等，那么函数递归地调用compare(addr, i+1, j+1)，比较后缀数组中第个和第个后缀的大小关系；
2. 否则，函数直接比较这两个二进制数值的大小关系，并返回结果。

函数my\_quicksort(int SA[],int left,int right,uint64\_t\* addr)使用快速排序算法对后缀数组SA进行排序。

**ALGORITHM 1** QuickSort in SA(快速排序)

1：选择待排序数组中的一个元素作为，;

2：**for**  **to**  **do**

2： **if** **then** ；

3：**end for**

4：；

5：

6：

函数首先选择后缀数组中的一个元素作为枢轴，将后缀数组分成两个部分：左边的部分包含所有小于枢轴的元素，右边的部分包含所有大于枢轴的元素。然后，函数递归地对这两个部分分别进行排序，直到排序完成。在递归的过程中，函数调用compare(uint64\_t\* addr,int i,int j)以判断后缀数组中两个后缀的大小关系，并使用std::swap()函数将它们交换位置。最后，函数将枢轴元素放回到分界点上，然后递归地对左右两个部分继续进行排序。

2.3 简易**BWT**的构建

Burrows-Wheeler Transform (BWT) 是一种数据压缩算法，它通过对输入字符串进行变换，生成一种可被更高效压缩的形式。BWT的主要思想是通过对字符串的循环旋转（cyclic rotation）来重新排列字符顺序，然后按照一定规则选择出相应的最后一列字符。

BWT的关键在于，生成的结果具有一定的**局部性**，即相似字符倾向于聚集在一起，这有利于后续的压缩算法的效率。

表格

描述已自动生成

图 1 一个变换的例子

具体而言，将需要转换的字符串，进行循环右移，每次循环一位，得到一组字符串，记录排序之后每个字符串的第一个字符和最后一个字符，F和L两列字符，因为F列是有序的，而L列是无序的，并且两列中**字符集合与相对顺序**均相同，所以只记录L列即可，而L列**实际上就是就是排序后的结果中第个后缀的前一个字符**。

图表, 图示

描述已自动生成

图 2 BWT示意图

从这个序列中可以得到以下信息：

1. L列中第一个字符是初始字符串的最后一个字符。
2. L列字符在初始字符串中的下一个元素为同行的F列字符；

所以解码过程是倒序来找原始字符串的，以图1为例：通过上面的第一条信息，得到原始字符串的最后一个字符，即字符C；确定字符C的位置后，需要通过F和L列还有相同字符的相对信息记录来找字符C的前一个元素。因为这个字符C是L列中第一个字符C，即相同字符的相对位置为1，所以去F列中找第一个字符C，即找到表中的第三行，通过第二行，可以得到倒数第二个字符为T，依次类推。

而在本实验中，根据BWT的定义可知，当对所有的后缀进行排序了之后，我们得到了这个序列的SA数组，SA数组中的每一位都是这个后缀开始的位置，而后缀开始位置的前一个位置我们可以由原串得出，因此只需要遍历一遍SA就可以得到BWT，完成简易版本BWT的构建。

2.4 **CSA**的原理

CSA（Compressed Suffix Array）是一种数据结构，用于存储字符串的后缀数组并支持高效的查询操作。后缀数组是一个将字符串的所有后缀按字典序排序的数组，CSA 主要解决的问题是在占用较少空间的情况下，提供对后缀数组的高效访问，其核心思想是利用了后缀数组中**后缀之间的重复性和局部性**。通常，后缀数组中相邻后缀的前缀会有很多重复的部分，因此 CSA 通过对这些重复部分进行压缩来节省空间。

CSA的主要过程就是对序列构建，其中，满足，主要过程如下：

1. 初始处理步骤：首先要将序列分成段连续的区域即为
2. 基本步骤：从开始构建函数
3. 合并步骤：对于，根据和来构建的函数，其中

将合并到已经排好的上，需要借助后缀依次向前排序。具体地，定义为任意的基因序列，为任意碱基，为基因序列在组的顺序大小，为在组中与碱基相同区段且的位置集合，即，于是有：

实际上就是先将排序，然后将和合入一个新串，由于和都是升序排列，直接根据在的顺序(order)合并即可（归并）。

排完之后值也需要相应的更改，对于中的串，它的下一位后缀只会因为中在它之前位次的串的插入而相应的后移。因而将它的值放到记录的顺序数组中进行二分查找（时间复杂度）即可计算得到。

1. **测试结果及分析**

如下图3所示，将排好序的SA对应后缀打印前25个字符（在字符串长度内）；而后打印简易BWT构建后前50个字符打印如下：

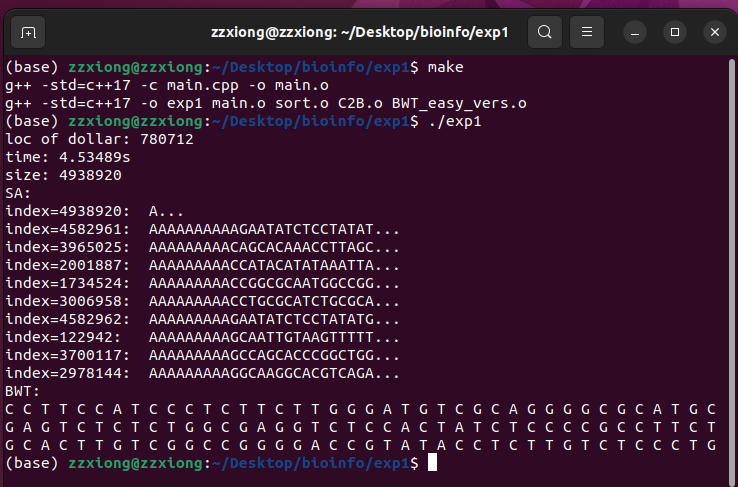


图 3 实验结果

可以看到，所有后缀基本按照碱基顺序排列。

1. **经验体会**

在本次实验中，我们着重学习了生物序列自索引的构建方法。这种数据结构能够有效地处理大规模的生物序列数据，提供了高效的查询和存储方式。通过实验的过程，我不仅熟悉了生物序列自索引的构建过程，还学会了如何将理论知识应用到实际编程中，解决真实世界的问题。这次实验不仅加深了我对课程所学知识的理解，也提升了我的编程能力和问题解决能力。

1. **附录：源代码（带注释）**

**main.cpp**

|  |
| --- |
| #include <iostream>  #include <fstream>  #include <string>  #include <vector>  #include <bitset>  #include <cstdlib>  #include "sort.h"  #include "C2B.h"  #include "BWT\_easy\_vers.h"  int main() {  std::vector<uint8\_t> sequences;  //open a file  std::string file = "./NC\_008253.fna";  std::ifstream myfile;  myfile.open(file);  std::string cur\_row;  uint64\_t size = 0;  std::getline(myfile, cur\_row);  while(std::getline(myfile, cur\_row)){  size += cur\_row.size();  }  myfile.close();  myfile.open(file);  uint64\_t num\_of\_u64 = (size \* 2) / 64;  int\* SA = (int\*) malloc(sizeof(int) \* size + 1);  for(int i = 0 ; i <= size + 1 ; i ++){  SA[i] = i;  }  uint64\_t\* sequence = (uint64\_t \*)malloc(sizeof(uint64\_t) \* num\_of\_u64 + 1);  C2B(sequence, myfile);  //record the time  clock\_t start, finish;  double totaltime;  start = clock();  my\_quicksort(SA, 0, size, sequence);  uint64\_t\* BWT = (uint64\_t \*)malloc(sizeof(uint64\_t) \* num\_of\_u64 + 1);  uint64\_t loc = C2B\_BWT(sequence, BWT, SA, size);  std::cout << "loc of dollar: " << loc << std::endl;  finish = clock();  totaltime = (double)(finish - start) / CLOCKS\_PER\_SEC;  std::cout<<"time: "<<totaltime<<"s"<<std::endl;  std::cout<<"size: "<<size<<std::endl;  //打印前10个SA数组的值  std::cout << "SA:" << std::endl;  for(int i = 0 ; i < 10 ; i ++){  std::cout << "index=" << SA[i] << ":\t";  int j = 0;  while(SA[i] + j <= size && j < 25){  std::cout << extract\_char(sequence, SA[i] + j);  j++;  }  std::cout << "..." << std::endl;  }  std::cout << "BWT:" << std::endl;  print\_BWT(SA, sequence, 120);  return 0;  } |

**C2B.cpp**

|  |
| --- |
| #include <iostream>  #include <fstream>  #include <string>  #include <vector>  #include <bitset>  #include <cstdlib>  #include "sort.h"  //convert the char into the 00,01,10,11  uint64\_t encode(char c) {  switch (c) {  case ('A') :  return 0;//00  case ('C') :  return 1;//01  case ('G') :  return 2;//10  case ('T') :  return 3;//11  default:  return -1;  }  }  //this function is used to convert the char into the binary code  void C2B(uint64\_t\* addr, std::ifstream &infile){  if(!infile.is\_open()){  std::cout << "ERROR: No such file or directory." <<std::endl;  return;  }  short bits = 0;  uint64\_t cur = 0;  long i = -1;  std::string cur\_row;  std::getline(infile, cur\_row);  while(std::getline(infile, cur\_row)){  for(char c : cur\_row){  uint64\_t cur\_char = encode(c);  if(cur\_char == -1){  std::cout << "ERROR: A character is not one of \'ACGT\' " <<std::endl;  return;  }  if(bits == 0){  if (i == -1){  i++;  }else{  addr[i++] = cur;  }  cur = cur\_char;  bits = 62;  }else{  cur = (cur << 2) | cur\_char;  bits -= 2;  }  }  }  if(bits){  cur = cur << bits;  addr[i] = cur;  }  }  uint32\_t C2B\_BWT(uint64\_t\* seq, uint64\_t \*BWT, int\* SA, int size){  short bits = 0;  uint64\_t cur = 0;  long i = -1;  uint64\_t cur\_char = 0;  uint64\_t loc\_of\_dollar = 0;  for(int j = 0 ; j < size ; j ++){  if(SA[j] == 0) {  encode('A');  loc\_of\_dollar = j;  }  else uint64\_t cur\_char = encode(extract\_char(seq, SA[j] - 1));  if(bits == 0){  if (i == -1){  i++;  }else{  BWT[i++] = cur;  }  cur = cur\_char;  bits = 62;  }else{  cur = (cur << 2) | cur\_char;  bits -= 2;  }  }  if(bits){  cur = cur << bits;  BWT[i] = cur;  }  return loc\_of\_dollar;  } |

**Sort.cpp**

|  |
| --- |
| #include <iostream>  #include <fstream>  #include <string>  #include <vector>  #include <bitset>  #include <cstdlib>  uint64\_t extract\_val(uint64\_t\* addr, int idx){  int i = idx >> 5 ;  int j = idx % 32;  if(j == 0) return addr[i];  uint64\_t code1 = addr[i];  code1 = (code1 << j \* 2);  uint64\_t code2 = addr[i + 1];  code2 = code2 >> (64 - j \* 2);  uint64\_t res = code1 | code2;  return res;  }  int compare(uint64\_t\* addr, int i, int j){  uint64\_t val1, val2;  val1 = extract\_val(addr, i);  val2 = extract\_val(addr, j);  if(val1 == val2) return (compare(addr, i + 1, j + 1));  else return extract\_val(addr, i) < extract\_val(addr, j);  }  void my\_quicksort(int SA[], int left, int right, uint64\_t\* addr) {  if (left < right) {  int pivot = SA[right];  int i = left - 1;  for (int j = left; j <= right - 1; j++) {  if (compare(addr,SA[j], pivot)) {  i++;  std::swap(SA[i], SA[j]);  }  }  std::swap(SA[i + 1], SA[right]);  int partition = i + 1;  my\_quicksort(SA, left, partition - 1, addr);  my\_quicksort(SA, partition + 1, right, addr);  }  } |

**BWT\_easy\_vers.cpp**

|  |
| --- |
| #include <iostream>  #include <fstream>  #include <string>  #include <vector>  #include <bitset>  #include <cstdlib>  #include "BWT\_easy\_vers.h"  char extract\_char(uint64\_t\* addr, int idx){  int i = idx / 32;  int j = idx % 32;  uint64\_t code = addr[i];  code = (code << j \* 2) >> 62;//get the bits we want: shift left j\*2 bits, then shift right 62 bits  switch (code) {  case(0) : return 'A';  case(1) : return 'C';  case(2) : return 'G';  case(3) : return 'T';  default : return 'N';  }  }  void print\_BWT(int\* SA, uint64\_t\* addr, int len){  for(int i = 0; i < len; i++){  if(SA[i] == 0)  std::cout << '$';  else  std::cout << extract\_char(addr, SA[i] - 1) << ' ';  }  std::cout << std::endl;  } |