

```

1  /*
2  Program:
3      This program can find differences between two files containing bioinformatics
        data.
4
5      If provided two files A (from-file) and B (to-file), program will generate all
        lines
6      in A-B, A&B_A, A&B_B and B-A in terms of the criteria given. The file format of
7      file A and B can be different.
8
9      There will be two styles for comparison: one is coordinate based (option -c ) and
10     the other is name based (option -n). The default style is option -c.
11
12     The two styles were described as follows.
13
14     1) Coordinate-based diff. Two or more columns from file A and B will be selected
        and
15     compared to check if the two regions overlap. If two regions from the two files
        overlap,
16     then these two regions will be put into to A&B_A and A&B_B; those regions in A
        but not
17     in A&B will be put into A-B; and those in B but not in A&B will be put into B-A.
18
19     2) Name-based diff. Two columns from file A and B will be selected and compared
        in terms
20     of string comparison. Users need to specify the column numbers in two files to
        be compared.
21     If their names "overlap", it should generate 4 result files corresponding to
        A&B_A, A&B_B,
22     A-B, and B-A, where A&B_A contains those lines from file A and overlapping with
        some entries
23     in file B; A&B_B contains lines from file B and overlapping with entries in file
        A; A-B
24     contains those lines from file A and with no overlapping entries in B; and B-A
        stands for
25     those lines from file B but with no overlapping entries in A.
26
27 Usage:
28     This program can be used by following code when execute:
29
30     ./Biodiff [options] from-file to-file
31
32     In which,
33         option          --- the option you choose. See also OPTION in Reference
34         from-file, to-file --- the absolute or relative path to the two files
35 Reference:
36     OPTION:
37         -c output according to coordinate-based diff
38         -n output according to name-based diff
39         -a specify the criteria of from-file(following by the criteria)
40         -b specify the criteria of to-file(following by the criteria)
41         -h get the help of this program
42 Author:
43     Zhang 517111910078
44 */
45
46
47 #include<stdio.h>
48 #include<unistd.h>
49 #include<stdlib.h>
50 #include<stdbool.h>
51 #include<string.h>
52 #include<ctype.h>
53 #include <sys/stat.h>
54 #include <time.h>
55
56
57 #define NLINE_MAX 200
58 #define HELP_MESSAGE "\
59 #-----#\n\
60 #               Biodiff               #\n\

```

```

61 # Author: Zhang Zhizhuo #\n\
62 # A bioinformatics program to compare two data files and output #\n\
63 #-----#\n\
64 # Program function: #\n\
65 # This program can find differences between two files containing #\n\
66 #bioinformatics data. #\n\
67 # #\n\
68 # If provided two files A (from-file) and B (to-file), program will #\n\
69 #generate all lines in A-B, A&B_A, A&B_B and B-A in terms of the range #\n\
70 #given.The file format of file A and B can be different. #\n\
71 # #\n\
72 # There will be two styles for comparison: one is coordinate based #\n\
73 #(option -c ) and the other is name based (option -n).The default style#\n\
74 #is option -c. #\n\
75 # #\n\
76 # The two styles were described as follows. #\n\
77 # #\n\
78 # 1)Coordinate-based diff. Two or more columns from file A and B will#\n\
79 #be selected and compared to check if the two regions overlap. If two #\n\
80 #regions from the two files overlap, then these two regions will be put#\n\
81 #into to A&B_A and A&B_B; those regions in A but not in A&B will be put#\n\
82 #into A-B; and those in B but not in A&B will be put into B-A. #\n\
83 # #\n\
84 # 2)Name-based diff. Two columns from file A and B will be selected #\n\
85 #and compared in terms of string comparison. Users need to specify the #\n\
86 #column numbers in two files to be compared. If their names "overlap", #\n\
87 #it should generate 4 result files corresponding to A&B_A, A&B_B, A-B, #\n\
88 #and B-A, where A&B_A contains those lines from file A and overlapping #\n\
89 #with some entries in file B;A&B_B contains lines from file B and over-#\n\
90 #lapping with entries in file A; A-B contains those lines from file A #\n\
91 #and with no overlapping entries in B; and B-A stands for those lines #\n\
92 #from file B but with no overlapping entries in A. #\n\
93 #-----#\n\
94 # Usage: #\n\
95 # This program can be used by following code when execute: #\n\
96 # #\n\
97 # ./Biodiff [options] from-file to-file #\n\
98 # #\n\
99 # In which, #\n\
100 # option --- the option you choose. #\n\
101 # See also OPTION in Reference #\n\
102 # from-file, to-file --- the absolute or relative path to the two#\n\
103 # files #\n\
104 # #\n\
105 # Reference: #\n\
106 # OPTION: #\n\
107 # -c output according to coordinate-based diff #\n\
108 # -n output according to name-based diff #\n\
109 # -a specify the criteria of from-file(following by the criteria)#\n\
110 # -b specify the criteria of to-file(following by the criteria) #\n\
111 # -h get the help of this program #\n\
112 #-----#\n\
113 # Example: #\n\
114 # ./Biodiff -a 3,4 -b 3,4 geneA.gtf geneB.gtf #\n\
115 # ./Biodiff -c -a 3,4 -b 3,4 geneA.gtf geneB.gtf #\n\
116 # ./Biodiff -n -a 0 -b 8 geneA.gtf geneB.gtf #\n\
117 #-----#\n"

```

```

118
119
120 //数据存储结构体
121 typedef struct DataNode
122 {
123     char data[NLINE_MAX]; //存储整行数据
124     int start, end; //存储起始和终止位点
125     char name[30]; //存储名字
126     bool isoverlap; //判断是否存在重叠
127 }DataNode;
128
129
130 //字典树节点结构体
131 typedef struct TreeNode

```

```

132 {
133     struct TreeNode *children[NLINE_MAX]; //字典树子节点
134     bool flag; //判断是否为单词结束
135     char c; //该节点的字符
136 }TreeNode;
137
138
139 //创建一个输出结果文件夹存储结果，如果文件夹已存在则不新建文件夹
140 void result_mkdir();
141 /*-----对原始数据进行初步处理-----*/
142
143 //获得选项以及两个输入文件的区间，同时对异常输入进行提示
144 char get_option(int argc, char **argv, char file_column[2][NLINE_MAX], char
file_path[2][NLINE_MAX]);
145 //计算字符串中逗号的个数，用来判断输入的区域是否满足要求
146 int coma_count(char *p);
147 //获得输入的区域并转换为数字，方便后续处理
148 bool get_region(char file_column[2][NLINE_MAX], int file_col[2][2], char option);
149 //获得文件原始行数
150 int get_line_num(FILE *fp);
151 //读取文件数据，并排除空行，返回实际读取数据个数
152 int read_data(FILE *fp, DataNode *file_data, int file_col[2]);
153 //打开文件并返回是否存在
154 FILE *open_file(char *file_path);
155 //处理原始数据
156 DataNode *compose_data(char *file_path, int *file_col, int *read_num);
157
158 /*-----快排和判断处理-c的区域重叠-----*/
159
160 //qsort所需排序函数，按照区间起始值从小到大排序；如果起始值相等则按终止值排序
161 int end_cmp(const void *a, const void *b);
162 //判断是否重叠并标记（通过判断一组数据的start是否位于另一组数据之中来判断）
163 DataNode *isoverlap_c(DataNode *data_A, DataNode *data_B, int num_A, int num_B);
164 //判断两组区间是否重合并输出至相应的文件
165 void region_overlap(DataNode *data_A, DataNode *data_B, int num_A, int num_B);
166 //对标记后的数据依据是否重叠进行输出
167 void cprint(DataNode *data, int read_num, FILE *AB, FILE *A_B);
168 //对数据依据端点进行排序
169 DataNode *sort_data(DataNode *data, int read_num, char *file_path);
170
171 /*-----字典树处理-n的名字重叠-----*/
172
173 //创建字典树节点并初始化
174 TreeNode *create_node(char c, int flag);
175 //如果不存在，扩展字典树节点
176 bool append_node(TreeNode *temp, char c);
177 //向字典树中增加单词
178 bool add_word(TreeNode *root, char *name);
179 //在字典树中查找单词是否存在（相等）
180 bool search_word(TreeNode *root, char *name);
181 //创建数据集的字典树
182 TreeNode *create_tree(DataNode *data, int read_num);
183 //使用一组数据集在另一组数据集构建的字典树中全部比对并输出
184 void name_oversearch(TreeNode *root, DataNode *data, int read_num, FILE *AB_A, FILE
*A_B);
185 //名字比较的主体函数
186 void name_overlap(DataNode *data_A, DataNode *data_B, TreeNode *root_A, TreeNode
*root_B, int read_num_A, int read_num_B);
187
188
189 int main(int argc, char **argv)
190 {
191     clock_t begin, finish; //程序计时
192     begin = clock();
193     char file_path[2][NLINE_MAX], file_column[2][NLINE_MAX], option,
working_path[NLINE_MAX];
194     int file_col[2][2]; //用来存储输入的范围
195     double time_cost; //用来记录程序运行时间
196     option = get_option(argc, argv, file_column, file_path); //获得输入的参数
197     if (option != '0')
198     {

```

```

199     if(!get_region(file_column, file_col,
200         option))//如果区域获取返回false说明发生错误退出程序
201         return 0;
202     }
203     else return 0;//如果option为'0'说明发生错误退出程序
204     result_mkdir();//如果输出文件夹不存在，则创建输出文件夹
205     getcwd(working_path, sizeof(working_path));//读取当前工作路径
206
207     DataNode *data_A = NULL, *data_B = NULL;//存储两个数据集的结构体数组
208     int read_num_A, read_num_B;
209     //分别读取两个文件的数据并进行与选项相对应的提取处理
210     data_A = compose_data(file_path[0], file_col[0], &read_num_A);
211     data_B = compose_data(file_path[1], file_col[1], &read_num_B);
212     if (option == 'c')
213     {
214         //对两个数据集进行排序
215         data_A = sort_data(data_A, read_num_A, file_path[0]);
216         data_B = sort_data(data_B, read_num_B, file_path[1]);
217         //判断重叠数据并输出
218         region_overlap(data_A, data_B, read_num_A, read_num_B);
219     }
220     else if (option == 'n')
221     {
222         //对两个数据集分别建立字典树
223         TreeNode *root_A, *root_B;
224         root_A = create_tree(data_A, read_num_A);
225         root_B = create_tree(data_B, read_num_B);
226         //判断重叠数据并输出
227         name_overlap(data_A, data_B, root_A, root_B, read_num_A, read_num_B);
228         free(root_A);
229         free(root_B);
230     }
231     printf("-----Write over-----\n");
232     printf("The results are in %s/result\n", working_path);
233     //释放内存
234     free(data_A);
235     free(data_B);
236     finish = clock();
237     time_cost = ((double) (finish-begin)/CLOCKS_PER_SEC);
238     printf("Program time: %f s.\n", time_cost);
239     return 0;
240 }
241
242 //创建一个输出结果文件夹存储结果，如果文件夹已存在则不新建文件夹
243 void result_mkdir()
244 {
245     int is_exist;
246     is_exist = access("./result", 0);//判断文件夹是否已经存在
247     if (is_exist == -1)//如果不存在则新建文件夹
248     {
249         mkdir("./result", S_IRWXU | S_IRWXG | S_IROTH | S_IXOTH);// if run in
250         Windows, using <mkdir("./result");> instead of this line
251     }
252     else return;
253 }
254
255 /*-----对原始数据进行初步处理-----*/
256
257
258 //获得选项以及两个输入文件的区间，同时对异常输入进行提示
259 char get_option(int argc, char **argv, char file_column[2][NLINE_MAX], char
260 file_path[2][NLINE_MAX])
261 {
262     char opt, option = 'c';//默认选项为-c
263     bool by_region = false, by_name = false;//判断模式是否同时选中
264     extern char *optarg;
265     extern int optind, opterr, optopt;
266     opterr = 0;//静默错误选项提示

```

```

266     if (argc < 7 && argc != 2)
267     {
268         printf("Warning: the arguments you provide are not enough.\n"
269             "Options -a, -b must be chosen and two file paths must be provided. \n"
270             "If you want to get help, enter the following code:\n\n\t./Biodiff -h\n\n");
271         return '0';
272     }
273     else if (argc > 8)
274     {
275         printf("Warning: you provide too many arguments. Please check your command
276             line.\n"
277             "If you want to get help, enter the following code:\n\n\t./Biodiff -h\n\n");
278         return '0';
279     }
280     while ((opt = getopt(argc, argv, "a:b:cnh")) != -1)
281     {
282         switch (opt)
283         {
284             case 'a':
285                 strcpy(file_column[0], optarg);
286                 break;
287             case 'b':
288                 strcpy(file_column[1], optarg);
289                 break;
290             case 'c':
291                 by_region = true;
292                 option = opt;
293                 break;
294             case 'n':
295                 by_name = true;
296                 option = opt;
297                 break;
298             case 'h':
299                 printf(HELP_MESSAGE);
300                 return '0';
301                 break;
302             case '?':
303                 printf("Warning: you have entered undefined option(s). Only -abcnh are
304                     acceptable.\n"
305                     "If you want to get help, enter the following code:\n\n\t./Biodiff
306                     -h\n\n");
307                 return '0';
308                 break;
309         }
310     }
311     if (strlen(file_column[0]) == 0 || strlen(file_column[1]) == 0)
312     {
313         printf("Warning: -a or -b is not used to provide criteria.\n"
314             "If you want to get help, enter the following code:\n\n\t./Biodiff -h\n\n");
315         return '0';
316     }
317     //排除-c和-n同时选择的情况
318     if (by_name && by_region)
319     {
320         printf("Warning: -c and -n can't be chosen at the same time!\n"
321             "If you want to get help, enter the following code:\n\n\t./Biodiff -h\n\n");
322         return '0';
323     }
324     //检验是否输入了两个文件的路径
325     if ((optind+1) < argc)
326     {
327         strcpy(file_path[0], argv[optind]);
328         strcpy(file_path[1], argv[optind+1]);
329     }
330     else
331     {
332         printf("You need to provide two file paths for this program.\n"
333             "If you want to get help, enter the following code:\n\n\t./Biodiff -h\n\n");
334         return '0';
335     }

```

```

334     return option;
335 }
336
337
338 //计算字符串中逗号的个数，用来判断输入的区域是否满足要求
339 int coma_count(char *p)
340 {
341     int count = 0,i = 0;
342     while(p[i])
343     {
344         if(p[i] == ',') ++count;
345         ++i;
346     }
347     return count;
348 }
349
350 //获得输入的区域并转换为数字，方便后续处理
351 bool get_region(char file_column[2][NLINE_MAX], int file_col[2][2], char option)
352 {
353     //以-2作为是否满足区域格式要求的判断标准
354     file_col[0][0] = -2;
355     file_col[0][1] = -2;
356     if (option == 'c')
357     {
358         //只有只存在一个逗号的情况下才可以继续
359         if (coma_count(file_column[0]) == 1 && coma_count(file_column[1]) == 1)
360         {
361             sscanf(file_column[0], "%d,%d", &file_col[0][0], &file_col[0][1]);
362             sscanf(file_column[1], "%d,%d", &file_col[1][0], &file_col[1][1]);
363         }
364     }
365     else if (option == 'n')
366     {
367         //只有不存在逗号的情况下才可以继续
368         if (coma_count(file_column[0]) == 0 && coma_count(file_column[1]) == 0)
369         {
370             //以第二个数字为-1作为-n的标志
371             file_col[0][1] = -1;
372             file_col[1][1] = -1;
373             file_col[0][0] = atoi(file_column[0]);
374             file_col[1][0] = atoi(file_column[1]);
375         }
376     }
377     //判断是否可以继续处理
378     if (file_col[0][0] == -2 || file_col[0][1] == -2)
379     {
380         printf("The criteria you enter isn't in the right pattern with option -%c\n",
381             "If you want to get help, enter the following code:\n\n\t./Biodiff -h\n\n",
382             option);
383         return false;
384     }
385     else return true;
386 }
387
388 //获得文件原始行数
389 int get_line_num(FILE *fp)
390 {
391     int line_num = 0;//记录行数
392     char temp[NLINE_MAX];//用来存放临时数据的字符串
393     while (feof(fp) == 0)
394     {
395         ++line_num;
396         fgets(temp, NLINE_MAX, fp);//用来将位置指针指向下一行
397     }
398     if (line_num == 1)
399     {
400         printf("WARNING: You have provided a file with no data in it!\n");
401         exit(1);
402     }
403 }

```

```

404     rewind(fp); //将位置指针重新设到开头
405     return line_num-1;
406 }
407
408 //读取文件数据，并排除空行，返回实际读取数据个数
409 int read_data(FILE *fp, DataNode *file_data, int file_col[2])
410 {
411     int index = 0; //读取时的索引记录
412     char temp_data[NLINE_MAX]; //作为临时的数据存储变量
413     while (feof(fp) == 0)
414     {
415         strcpy(temp_data, "");
416         fgets(temp_data, NLINE_MAX, fp); //先将数据存在临时变量中，用来排除空行
417         if (strlen(temp_data) <= 2) continue; //排除空行
418         strcpy((file_data+index)->data, temp_data); //排除空行后将数据写入结构体数组
419         (file_data+index)->isoverlap = false; //初始化判断是否重叠的标志
420         char *p; //保存分割后的字符串
421         int i = 0; //用来判断当前分割后字符串位置
422         p = strtok(temp_data, "\t\n;\""); //以换行符、制表符、分号和引号为分隔符
423         if ((file_col[0]) == 0) //如果是第一列，由于循环中无法处理所以单独处理
424         {
425             if ((file_col[1]) != -1) //通过是否为-1判断-c还是-n模式
426             {
427                 (file_data+index)->start = atoi(p);
428             }
429             else strcpy((file_data+index)->name, p); //当读到输入列数时，存为name
430             ++index;
431             continue; //因为是第一列，不需要进入后续分割
432         }
433         else
434         {
435             for (i = 1; (p = strtok(NULL, "\t\n;\"")) != NULL; ++i)
436             {
437                 if (file_col[1] != -1) //通过是否为-1判断-c还是-n模式
438                 {
439                     if (i == file_col[0]) //当读到输入列数的第一个时，存为start
440                     {
441                         (file_data+index)->start = atoi(p);
442                     }
443                     else if (i ==
444                         file_col[1]) //当读到输入列数的第二个时，存为end并退出循环
445                     {
446                         (file_data+index)->end = atoi(p);
447                         break;
448                     }
449                 }
450                 else
451                 {
452                     if (i ==
453                         (file_col[0]+1)) //当读到输入列数时，存为name;由于数据文件格式原因
454                         , 为提取名字，实际为列数+1
455                     {
456                         strcpy((file_data+index)->name, p);
457                         break;
458                     }
459                 }
460             }
461             ++index;
462         }
463     }
464     fclose(fp);
465     return index; //返回实际读取的非空行数
466 }
467
468 //打开文件并返回是否存在
469 FILE *open_file(char *file_path)
470 {
471     FILE *fp;
472     if((fp = fopen(file_path, "r")) == NULL)

```



```

472     {
473         printf("file dosen't exist!\n");
474         return NULL;
475     }
476     else return fp;
477 }
478
479
480 //处理原始数据
481 DataNode *compose_data(char *file_path, int *file_col, int *read_num)
482 {
483     DataNode *data = NULL;
484     int line_num;
485     FILE *fp;
486     fp = open_file(file_path);
487     line_num = get_line_num(fp); //获得原始行数
488     data = (DataNode*)malloc(line_num*sizeof(DataNode));
489     printf("Reading %s\n", file_path);
490     *read_num = read_data(fp, data,
491         file_col); //按照要求读取数据，并返回实际读取行数（排除空行）
492     printf("-----Read over-----\n");
493     return data;
494 }
495
496 /*-----快排和判断处理-c的区域重叠-----*/
497
498 //判断是否重叠并标记（通过判断一组数据的start是否位于另一组数据之中来判断）
499 DataNode *isoverlap_c(DataNode *data_A, DataNode *data_B, int num_A, int num_B)
500 {
501     int index_A = 0, index_B = 0, count;
502     for (index_A = 0, index_B = 0; index_A < num_A; ++index_A)
503     {
504         for (; index_B < num_B; ++index_B)
505         {
506             //如果B数据集的start小于A数据集的start，则跳过
507             if ((data_B+index_B)->start < (data_A+index_A)->start) continue;
508             else
509             {
510                 for (count = index_B; count < num_B; ++count)
511                 {
512                     //如果B数据集的start大于A数据集的end，说明后面都不会重叠，则跳出循环
513                     if ((data_B+count)->start > (data_A+index_A)->end) break;
514                     else
515                     {
516                         //此时B数据集的start位于A数据集的两端点之间，对两个数据都进行标记
517                         (data_B+count)->isoverlap = true;
518                         (data_A+index_A)->isoverlap = true;
519                     }
520                 }
521             }
522         }
523         break;
524     }
525 }
526 }
527
528 //判断两组区间是否重合并输出至相应的文件
529 void region_overlap(DataNode *data_A, DataNode *data_B, int num_A, int num_B)
530 {
531     FILE *AB_A, *AB_B, *A_B, *B_A;
532     AB_A = fopen("../result/A&B_A.gtf", "w");
533     AB_B = fopen("../result/A&B_B.gtf", "w");
534     A_B = fopen("../result/A-B.gtf", "w");
535     B_A = fopen("../result/B-A.gtf", "w");

```



```

537     //进行两次标记保证每一个重叠的数据都被标记
538     isoverlap_c(data_A, data_B, num_A, num_B);
539     isoverlap_c(data_B, data_A, num_B, num_A);
540     //分别按照标记输出
541     cprint(data_A, num_A, AB_A, A_B);
542     cprint(data_B, num_B, AB_B, B_A);
543
544     fclose(AB_A);
545     fclose(AB_B);
546     fclose(A_B);
547     fclose(B_A);
548 }
549
550
551 //对标记后的数据依据是否重叠进行输出
552 void cprint(DataNode *data, int read_num, FILE *AB, FILE *A_B)
553 {
554     int index;
555     for (index = 0; index < read_num; ++index)
556     {
557         if ((data+index)->isoverlap) fputs((data+index)->data, AB);
558         else fputs((data+index)->data, A_B);
559     }
560 }
561
562
563 //qsort所需排序函数，按照区间起始值从小到大排序;如果起始值相等则按终止值排序
564 int end_cmp(const void *a, const void *b)
565 {
566     DataNode *p = (DataNode *)a;
567     DataNode *q = (DataNode *)b;
568     if (p->start != q->start) return p->start - q->start;
569     else return p->end - q->end;
570 }
571
572
573 //对数据依据端点进行排序
574 DataNode *sort_data(DataNode *data, int read_num, char *file_path)
575 {
576     printf("Sorting %s\n", file_path);
577     //使用内建函数qsort进行快排，将数据按start从小到大排列
578     qsort(data, read_num, sizeof(DataNode), end_cmp);
579     printf("-----Sort over-----\n");
580     return data;
581 }
582
583
584 /*-----字典树处理-n的名字重叠-----*/
585
586
587 //创建字典树节点并初始化
588 TreeNode *create_node(char c, int flag)
589 {
590     TreeNode *temp = (TreeNode*)malloc(sizeof(TreeNode));
591     temp->c = c;
592     temp->flag = flag;
593     int i = 0;
594     while (i < NLINE_MAX) temp->children[i++] = NULL;
595     return temp;
596 }
597
598
599 //如果不存在，扩展字典树节点
600 bool append_node(TreeNode *temp, char c)
601 {
602     TreeNode *ptr = temp->children[c - ' '];
603     if (ptr) return false;
604     else
605     {
606         temp->children[c - ' '] = create_node(c, false);

```

```

607         return true;
608     }
609 }
610
611 //向字典树中增加单词
612 bool add_word(TreeNode *root, char *name)
613 {
614     char c = *name;
615     TreeNode *ptr = root;
616     bool flag = true; //判断是否增加了单词
617     while (c != '\0')
618     {
619         if (!append_node(ptr, c))
620         {
621             flag = false;
622         }
623         ptr = ptr->children[c - ' '];
624         c = *(++name);
625     }
626     //在单词结尾将flag标志变为true
627     if (!ptr->flag)
628     {
629         flag = false;
630         ptr->flag = true;
631     }
632     return !flag;
633 }
634
635 //创建数据集的字典树
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652 //在字典树中查找名字是否存在(包含关系)
653 bool search_word(TreeNode *root, char *name)
654 {
655     TreeNode *ptr = root;
656     char *p = name;
657     int i = 0;
658     while (*p != '\0')
659     {
660         if (ptr->children[*p - ' '] != NULL) //判断是否可以继续查找
661         {
662             ptr = ptr->children[*p - ' '];
663             ++p;
664         }
665         else break; //当名字查找到字典树无法继续则退出循环
666     }
667
668     //符合包含关系的第一种情况为名字全部查找完毕，即名字属于字典树前缀；第二种情况是字典树查找完毕，即字典树内包含名字的前缀
669     if (*p == '\0' || ptr->flag) return true;
670     else return false;
671 }
672
673 //使用一组数据集在另一组数据集构建的字典树中全部比对并输出
674 void name_oversearch(TreeNode *root, DataNode *data, int read_num, FILE *AB_A, FILE *A_B)

```

```

675 {
676     int index = 0;
677     bool isoverlap = false; //判断是否重叠
678     while (index < read_num)
679     {
680         //依次判断每个单词是否在字典树中出现，并根据结果分至两个文件中
681         isoverlap = search_word(root, (data+index)->name);
682         if (isoverlap) fputs((data+index)->data, AB_A);
683         else fputs((data+index)->data, A_B);
684         ++index;
685     }
686 }
687
688 //名字比较的主体函数
689 void name_overlap(DataNode *data_A, DataNode *data_B, TreeNode *root_A, TreeNode
690 *root_B, int read_num_A, int read_num_B)
691 {
692     FILE *AB_A, *AB_B, *A_B, *B_A;
693     AB_A = fopen("./result/A&B_A.gtf", "w");
694     AB_B = fopen("./result/A&B_B.gtf", "w");
695     A_B = fopen("./result/A-B.gtf", "w");
696     B_A = fopen("./result/B-A.gtf", "w");
697     printf("Searching\n");
698     //对两个数据集分别进行字典树全查找
699     name_oversearch(root_A, data_B, read_num_B, AB_B, B_A);
700     name_oversearch(root_B, data_A, read_num_A, AB_A, A_B);
701     printf("-----Search over-----\n");
702 }
703
704

```