```c
/*
Program:
    This program can find differences between two files containing bioinformatics data.

    If provided two files A (from-file) and B (to-file), program will generate all lines
    in A-B, A&B_A, A&B_B and B-A in terms of the criteria given. The file format of
    file A and B can be different.

    There will be two styles for comparison: one is coordinate based (option –c ) and
    the other is name based (option –n). The default style is option -c.

    The two styles were described as follows.

    1) Coordinate-based diff. Two or more columns from file A and B will be selected and
    compared to check if the two regions overlap. If two regions from the two files overlap,
    then these two regions will be put into to A&B_A and A&B_B; those regions in A but not
    in A&B will be put into A-B; and those in B but not in A&B will be put into B-A.

    2) Name-based diff. Two columns from file A and B will be selected and compared in terms
    of string comparison. Users need to specify the column numbers in two files to be compared.
    If their names “overlap”, it should generate 4 result files corresponding to A&B_A, A&B_B,
    A-B, and B-A, where A&B_A contains those lines from file A and overlapping with some entries
    in file B; A&B_B contains lines from file B and overlapping with entries in file A; A-B
    contains those lines from file A and with no overlapping entries in B; and B-A stands for
    those lines from file B but with no overlapping entries in A.

Usage:
    This program can be used by following code when execute:

        ./Biodiff [options] from-file to-file

    In which,
        option              --- the option you choose. See also OPTION in Reference
        from-file, to-file --- the absolute or relative path to the two files
Reference:
    OPTION:
        -c output according to coordinate-based diff
        -n output according to name-based diff
        -a specify the criteria of from-file(following by the criteria)
        -b specify the criteria of to-file(following by the criteria)
        -h get the help of this program
Author:
    Zhang 517111910078
*/


#include<stdio.h>
#include<unistd.h>
#include<stdlib.h>
#include<stdbool.h>
#include<string.h>
#include<ctype.h>
#include <sys/stat.h>
#include <time.h>


#define NLINE_MAX 200
#define HELP_MESSAGE "\
#--------------------------------------------------------------------#\n\
#                              Biodiff                               #\n\
#                       Author: Zhang Zhizhuo                             #\n\
#     A bioinformatics program to compare two data files and output     #\n\
#-----------------------------------------------------------------------#\n\
# Program function:                                                       #\n\
#   This program can find differences between two files containing     #\n\
#bioinformatics data.                                                     #\n\
#                                                                         #\n\
#   If provided two files A (from-file) and B (to-file), program will  #\n\
#generate all lines in A-B, A&B_A, A&B_B and B-A in terms of the range #\n\
#given.The file format of file A and B can be different.                 #\n\
#                                                                         #\n\
#   There will be two styles for comparison: one is coordinate based   #\n\
#(option -c ) and the other is name based (option -n).The default style#\n\
#is option -c.                                                            #\n\
#                                                                         #\n\
#   The two styles were described as follows.                            #\n\
#                                                                         #\n\
#   1)Coordinate-based diff. Two or more columns from file A and B will#\n\
#be selected and compared to check if the two regions overlap. If two  #\n\
#regions from the two files overlap, then these two regions will be put#\n\
#into to A&B_A and A&B_B; those regions in A but not in A&B will be put#\n\
#into A-B; and those in B but not in A&B will be put into B-A.          #\n\
#                                                                         #\n\
#   2)Name-based diff. Two columns from file A and B will be selected  #\n\
#and compared in terms of string comparison. Users need to specify the #\n\
#column numbers in two files to be compared. If their names “overlap”, #\n\
#it should generate 4 result files corresponding to A&B_A, A&B_B, A-B, #\n\
#and B-A, where A&B_A contains those lines from file A and overlapping #\n\
#with some entries in file B;A&B_B contains lines from file B and over-#\n\
#lapping with entries in file A; A-B contains those lines from file A  #\n\
#and with no overlapping entries in B; and B-A stands for those lines  #\n\
#from file B but with no overlapping entries in A.                      #\n\
#-----------------------------------------------------------------------#\n\
# Usage:                                                                  #\n\
#   This program can be used by following code when execute:            #\n\
#                                                                         #\n\
#       ./Biodiff [options] from-file to-file                            #\n\
#                                                                         #\n\
#   In which,                                                             #\n\
#       option              --- the option you choose.                   #\n\
#                               See also OPTION in Reference             #\n\
#       from-file, to-file --- the absolute or relative path to the two#\n\
#                               files                                     #\n\
#                                                                         #\n\
# Reference:                                                              #\n\
#    OPTION:                                                              #\n\
#       -c output according to coordinate-based diff                     #\n\
#       -n output according to name-based diff                           #\n\
#       -a specify the criteria of from-file(following by the criteria)#\n\
#       -b specify the criteria of to-file(following by the criteria)  #\n\
#       -h get the help of this program                                  #\n\
#-----------------------------------------------------------------------#\n\
# Example:                                                                #\n\
#   ./Biodiff -a 3,4 -b 3,4 geneA.gtf geneB.gtf                          #\n\
#   ./Biodiff -c -a 3,4 -b 3,4 geneA.gtf geneB.gtf                       #\n\
#   ./Biodiff -n -a 0 -b 8 geneA.gtf geneB.gtf                           #\n\
#-----------------------------------------------------------------------#\n"


//数据存储结构体
typedef struct DataNode
{
    char data[NLINE_MAX];//存储整行数据
    int start, end;//存储起始和终止位点
    char name[30];//存储名字
    bool isoverlap;//判断是否存在重叠
}DataNode;


//字典树节点结构体
typedef struct TreeNode
```

```c
{
    struct TreeNode *children[NLINE_MAX];//字典树子节点
    bool flag;//判断是否为单词结束
    char c;//该节点的字符
}TreeNode;


//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
void result_mkdir();
/*---------------------对原始数据进行初步处理---------------------*/

//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
char get_option(int argc, char **argv, char file_column[2][NLINE_MAX], char
file_path[2][NLINE_MAX]);
//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
int coma_count(char *p);
//获得输入的区域并转换为数字，方便后续处理
bool get_region(char file_column[2][NLINE_MAX], int file_col[2][2], char option);
//获得文件原始行数
int get_line_num(FILE *fp);
//读取文件数据，并排除空行，返回实际读取数据个数
int read_data(FILE *fp, DataNode *file_data, int file_col[2]);
//打开文件并返回是否存在
FILE *open_file(char *file_path);
//处理原始数据
DataNode *compose_data(char *file_path, int *file_col, int *read_num);

/*---------------------快排和判断处理-c的区域重叠---------------------*/

//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
int end_cmp(const void *a, const void *b);
//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）
DataNode *isoverlap_c(DataNode *data_A, DataNode *data_B, int num_A, int num_B);
//判断两组区间是否重合并输出至相应的文件
void region_overlap(DataNode *data_A, DataNode *data_B, int num_A, int num_B);
//对标记后的数据依据是否重叠进行输出
void cprint(DataNode *data, int read_num, FILE *AB, FILE *A_B);
//对数据依据端点进行排序
DataNode *sort_data(DataNode *data, int read_num, char *file_path);

/*---------------------字典树处理-n的名字重叠---------------------*/

//创建字典树节点并初始化
TreeNode *create_node(char c, int flag);
//如果不存在，扩展字典树节点
bool append_node(TreeNode *temp, char c);
//向字典树中增加单词
bool add_word(TreeNode *root, char *name);
//在字典树中查找单词是否存在（相等）
bool search_word(TreeNode *root, char *name);
//创建数据集的字典树
TreeNode *create_tree(DataNode *data, int read_num);
//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
void name_oversearch(TreeNode *root, DataNode *data, int read_num, FILE *AB_A, FILE
*A_B);
//名字比较的主体函数
void name_overlap(DataNode *data_A, DataNode *data_B, TreeNode *root_A, TreeNode
*root_B, int read_num_A, int read_num_B);


int main(int argc, char **argv)
{
    clock_t begin, finish;//程序计时
    begin = clock();
    char file_path[2][NLINE_MAX], file_column[2][NLINE_MAX], option,
    working_path[NLINE_MAX];
    int file_col[2][2];//用来存储输入的范围
    double time_cost;//用来记录程序运行时间
    option = get_option(argc, argv, file_column, file_path);//获得输入的参数
    if (option != '0')
    {
        if(!get_region(file_column, file_col, option))//如果区域获取返回false说明发生错误退出程序
            return 0;
    }
    else return 0;//如果option为‘0’说明发生错误退出程序
    result_mkdir();//如果输出文件夹不存在，则创建输出文件夹
    getcwd(working_path, sizeof(working_path));//读取当前工作路径

    DataNode *data_A = NULL, *data_B = NULL;//存储两个数据集的结构体数组
    int read_num_A, read_num_B;
    //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
    data_A = compose_data(file_path[0], file_col[0], &read_num_A);
    data_B = compose_data(file_path[1], file_col[1], &read_num_B);
    if (option == 'c')
    {
        //对两个数据集进行排序
        data_A = sort_data(data_A, read_num_A, file_path[0]);
        data_B = sort_data(data_B, read_num_B, file_path[1]);
        //判断重叠数据并输出
        region_overlap(data_A, data_B, read_num_A, read_num_B);
    }
    else if (option == 'n')
    {
        //对两个数据集分别建立字典树
        TreeNode *root_A, *root_B;
        root_A = create_tree(data_A, read_num_A);
        root_B = create_tree(data_B, read_num_B);
        //判断重叠数据并输出
        name_overlap(data_A, data_B, root_A, root_B, read_num_A, read_num_B);
        free(root_A);
        free(root_B);
    }
    printf("-----------------Write over-----------------\n");
    printf("The results are in %s/result\n", working_path);
    //释放内存
    free(data_A);
    free(data_B);
    finish = clock();
    time_cost = ((double)(finish-begin)/CLOCKS_PER_SEC);
    printf("Program time: %f s.\n", time_cost);
    return 0;
}


//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
void result_mkdir()
{
    int is_exist;
    is_exist = access("./result", 0);//判断文件夹是否已经存在
    if (is_exist == -1)//如果不存在则新建文件夹
    {
        mkdir("./result", S_IRWXU | S_IRWXG | S_IROTH | S_IXOTH);// if run in Windows, using <mkdir("./result");> instead of this line
    }
    else return;
}


/*-------------------------------------------------对原始数据进行初步处理--------------------------------------------------*/


//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
char get_option(int argc, char **argv, char file_column[2][NLINE_MAX], char file_path[2][NLINE_MAX])
{
    char opt, option = 'c';//默认选项为-c
    bool by_region = false, by_name = false;//判断模式是否同时选中
    extern char *optarg;
    extern int optind, opterr, optopt;
    opterr = 0;//静默错误选项提示
    if (argc < 7 && argc != 2)
    {
        printf("Warning: the arguments you provide are not enough.\n"
        "Options -a, -b must be chosen and two file paths must be provided. \n"
        "If you want to get help, enter the following code:\n\n\t./Biodiff -h\n\n");
        return '0';
    }
    else if (argc > 8)
    {
        printf("Warning: you provide too many arguments. Please check your command 
        line.\n"
        "If you want to get help, enter the following code:\n\n\t./Biodiff -h\n\n");
        return '0';
    }
    while ((opt = getopt(argc, argv, "a:b:cnh")) != -1)
    {
        switch (opt)
        {
        case 'a':
            strcpy(file_column[0], optarg);
            break;
        case 'b':
            strcpy(file_column[1], optarg);
            break;
        case 'c':
            by_region = true;
            option = opt;
            break;
        case 'n':
            by_name = true;
            option = opt;
            break;
        case 'h':
            printf(HELP_MESSAGE);
            return '0';
            break;
        case '?':
            printf("Warning:you have entered undefined option(s). Only -abcnh are 
            acceptable.\n"
            "If you want to get help, enter the following code:\n\n\t./Biodiff 
            -h\n\n");
            return '0';
            break;
        }
    }
    if (strlen(file_column[0]) == 0 || strlen(file_column[1]) == 0)
    {
        printf("Warning: -a or -b is not used to provide criteria.\n"
        "If you want to get help, enter the following code:\n\n\t./Biodiff -h\n\n");
        return '0';
    }

    //排除-c和-n同时选择的情况
    if (by_name && by_region)
    {
        printf("Warning: -c and -n can't be chosen at the same time!\n"
        "If you want to get help, enter the following code:\n\n\t./Biodiff -h\n\n");
        return '0';
    }
    //检验是否输入了两个文件的路径
    if ((optind+1) < argc)
    {
        strcpy(file_path[0], argv[optind]);
        strcpy(file_path[1], argv[optind+1]);
    }
    else
    {
        printf("You need to provide two file paths for this program.\n"
        "If you want to get help, enter the following code:\n\n\t./Biodiff -h\n\n");
        return '0';
    }
    return option;
}


//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
int coma_count(char *p)
{
    int count = 0,i = 0;
    while(p[i])
    {
        if(p[i] == ',') ++count;
        ++i;
    }
    return count;
}

//获得输入的区域并转换为数字，方便后续处理
bool get_region(char file_column[2][NLINE_MAX], int file_col[2][2], char option)
{
    //以-2作为是否满足区域格式要求的判断标准
    file_col[0][0] = -2;
    file_col[0][1] = -2;
    if (option == 'c')
    {
        //只有只存在一个逗号的情况下才可以继续
        if (coma_count(file_column[0]) == 1 && coma_count(file_column[1]) == 1)
        {
            sscanf(file_column[0], "%d,%d", &file_col[0][0], &file_col[0][1]);
            sscanf(file_column[1], "%d,%d", &file_col[1][0], &file_col[1][1]);
        }
    }
    else if (option == 'n')
    {
        //只有不存在逗号的情况下才可以继续
        if (coma_count(file_column[0]) == 0 && coma_count(file_column[1]) == 0)
        {
            //以第二个数字为-1作为-n的标志
            file_col[0][1] = -1;
            file_col[1][1] = -1;
            file_col[0][0] = atoi(file_column[0]);
            file_col[1][0] = atoi(file_column[1]);
        }
    }
    //判断是否可以继续处理
    if (file_col[0][0] == -2 || file_col[0][1] == -2)
    {
        printf("The criteria you enter isn't in the right pattern with option -%c\n"
        "If you want to get help, enter the following code:\n\n\t./Biodiff -h\n\n",
        option);
        return false;
    }
    else return true;
}


//获得文件原始行数
int get_line_num(FILE *fp)
{
    int line_num = 0;//记录行数
    char temp[NLINE_MAX];//用来存放临时数据的字符串
    while (feof(fp) == 0)
    {
        ++line_num;
        fgets(temp, NLINE_MAX, fp);//用来将位置指针指向下一行
    }
    if (line_num == 1)
    {
        printf("WARNING: You have provided a file with no data in it!\n");
        exit(1);
    }

    rewind(fp);//将位置指针重新设到开头
    return line_num-1;
}


//读取文件数据，并排除空行，返回实际读取数据个数
int read_data(FILE *fp, DataNode *file_data, int file_col[2])
{
    int index = 0;//读取时的索引记录
    char temp_data[NLINE_MAX];//作为临时的数据存储变量
    while (feof(fp) == 0)
    {
        strcpy(temp_data, "");
        fgets(temp_data, NLINE_MAX, fp);//先将数据存在临时变量中，用来排除空行
        if (strlen(temp_data) <= 2) continue;//排除空行
        strcpy((file_data+index)->data, temp_data);//排除空行后将数据写入结构体数组
        (file_data+index)->isoverlap = false;//初始化判断是否重叠的标志
        char *p;//保存分割后的字符串
        int i = 0;//用来判断当前分割后字符串位置
        p = strtok(temp_data, "\t\n;\"");//以换行符、制表符、分号和引号为分隔符
        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
        {
            if ((file_col[1]) != -1) //通过是否为-1判断-c还是-n模式
            {
                (file_data+index)->start = atoi(p);
            }
            else strcpy((file_data+index)->name, p);//当读到输入列数时，存为name
            ++index;
            continue;//因为是第一列，不需要进入后续分割
        }
        else
        {
            for (i = 1;(p = strtok(NULL, "\t\n;\"")) != NULL;++i)
            {
                if (file_col[1] != -1)//通过是否为-1判断-c还是-n模式
                {
                    if (i == file_col[0])//当读到输入列数的第一个时，存为start
                    {
                        (file_data+index)->start = atoi(p);
                    }
                    else if (i == file_col[1])//当读到输入列数的第二个时，存为end并退出循环
                    {
                        (file_data+index)->end = atoi(p);
                        break;
                    }
                }
                else
                {
                    if (i == (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，为提取名字，实际为列数+1
                    {
                        strcpy((file_data+index)->name, p);
                        break;
                    }
                }
            }
            ++index;
        }
    }
    fclose(fp);
    return index;//返回实际读取的非空行数
}


//打开文件并返回是否存在
FILE *open_file(char *file_path)
{
    FILE *fp;
    if((fp = fopen(file_path, "r")) == NULL)
    {
        printf("file dosen't exist!\n");
        return NULL;
    }
    else return fp;
}


//处理原始数据
DataNode *compose_data(char *file_path, int *file_col, int *read_num)
{
    DataNode *data = NULL;
    int line_num;
    FILE *fp;
    fp = open_file(file_path);
    line_num = get_line_num(fp);//获得原始行数
    data = (DataNode*)malloc(line_num*sizeof(DataNode));
    printf("Reading %s\n", file_path);
    *read_num = read_data(fp, data,
    file_col);//按照要求读取数据，并返回实际读取行数（排除空行）
    printf("------------------Read over-------------------\n");
    return data;
}


/*-------------------------------------------------快排和判断处理-c的区域重叠---------
----------------------------------------*/


//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）
DataNode *isoverlap_c(DataNode *data_A, DataNode *data_B, int num_A, int num_B)
{
    int index_A = 0, index_B = 0, count;
    for (index_A = 0, index_B = 0; index_A < num_A; ++index_A)
    {
        for (; index_B < num_B; ++index_B)
        {
            //如果B数据集的start小于A数据集的start，则跳过
            if ((data_B+index_B)->start < (data_A+index_A)->start) continue;
            else
            {
                for (count = index_B; count < num_B; ++count)
                {

                    //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
                    if ((data_B+count)->start > (data_A+index_A)->end) break;
                    else
                    {

                        //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
                        (data_B+count)->isoverlap = true;
                        (data_A+index_A)->isoverlap = true;
                    }
                }
            }
            break;
        }
    }
}


//判断两组区间是否重合并输出至相应的文件
void region_overlap(DataNode *data_A, DataNode *data_B, int num_A, int num_B)
{
    FILE *AB_A, *AB_B, *A_B, *B_A;
    AB_A = fopen("./result/A&B_A.gtf", "w");
    AB_B = fopen("./result/A&B_B.gtf", "w");
    A_B = fopen("./result/A-B.gtf", "w");
    B_A = fopen("./result/B-A.gtf", "w");
    //进行两次标记保证每一个重叠的数据都被标记
    isoverlap_c(data_A, data_B, num_A, num_B);
    isoverlap_c(data_B, data_A, num_B, num_A);
    //分别按照标记输出
    cprint(data_A, num_A, AB_A, A_B);
    cprint(data_B, num_B, AB_B, B_A);

    fclose(AB_A);
    fclose(AB_B);
    fclose(A_B);
    fclose(B_A);
}


//对标记后的数据依据是否重叠进行输出
void cprint(DataNode *data, int read_num, FILE *AB, FILE *A_B)
{
    int index;
    for (index = 0; index < read_num; ++index)
    {
        if ((data+index)->isoverlap) fputs((data+index)->data, AB);
        else fputs((data+index)->data, A_B);
    }
}


//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
int end_cmp(const void *a, const void *b)
{
    DataNode *p = (DataNode *)a;
    DataNode *q = (DataNode *)b;
    if (p->start != q->start) return p->start - q->start;
    else return p->end - q->end;
}


//对数据依据端点进行排序
DataNode *sort_data(DataNode *data, int read_num, char *file_path)
{
    printf("Sorting %s\n", file_path);
    //使用内建函数qsort进行快排, 将数据按start从小到大排列
    qsort(data, read_num, sizeof(DataNode), end_cmp);
    printf("-------------------Sort over--------------------\n");
    return data;
}


/*--------------------------------------------------字典树处理-n的名字重叠-------------
------------------------------------*/


//创建字典树节点并初始化
TreeNode *create_node(char c, int flag)
{
    TreeNode *temp = (TreeNode*)malloc(sizeof(TreeNode));
    temp->c = c;
    temp->flag = flag;
    int i = 0;
    while (i < NLINE_MAX) temp->children[i++] = NULL;
    return temp;
}


//如果不存在, 扩展字典树节点
bool append_node(TreeNode *temp, char c)
{
    TreeNode *ptr = temp->children[c - ' '];
    if (ptr) return false;
    else
    {
        temp->children[c - ' '] = create_node(c, false);
        return true;
    }
}


//向字典树中增加单词
bool add_word(TreeNode *root, char *name)
{
    char c = *name;
    TreeNode *ptr = root;
    bool flag = true;//判断是否增加了单词
    while (c != '\0')
    {
        if (!append_node(ptr, c))
        {
            flag = false;
        }
        ptr = ptr->children[c - ' '];
        c = *(++name);
    }
    //在单词结尾将flag标志变为true
    if (!ptr->flag)
    {
        flag = false;
        ptr->flag =true;
    }
    return !flag;
}


//创建数据集的字典树
TreeNode *create_tree(DataNode *data, int read_num)
{
    int index = 0;
    TreeNode *root = create_node('$', false);
    while (index < read_num)
    {
        //将每个单词添加进字典树中
        add_word(root, (data+index)->name);
        ++index;
    }
    return root;
}


//在字典树中查找名字是否存在(包含关系)
bool search_word(TreeNode *root, char *name)
{
    TreeNode *ptr = root;
    char *p = name;
    int i = 0;
    while (*p != '\0')
    {
        if (ptr->children[*p - ' '] != NULL)//判断是否可以继续查找
        {
            ptr = ptr->children[*p - ' '];
            ++p;
        }
        else break;//当名字查找到字典树无法继续则退出循环
    }
    //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
    if (*p == '\0' || ptr->flag) return true;
    else return false;
}


//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
void name_oversearch(TreeNode *root, DataNode *data, int read_num, FILE *AB_A, FILE
*A_B)
```

```
{
    int index = 0;
    bool isoverlap = false;//判断是否重叠
    while (index < read_num)
    {
        //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
        isoverlap = search_word(root, (data+index)->name);
        if (isoverlap) fputs((data+index)->data, AB_A);
        else fputs((data+index)->data, A_B);
        ++index;
    }
}


//名字比较的主体函数
void name_overlap(DataNode *data_A, DataNode *data_B, TreeNode *root_A, TreeNode
*root_B, int read_num_A, int read_num_B)
{
    FILE *AB_A, *AB_B, *A_B, *B_A;
    AB_A = fopen("./result/A&B_A.gtf", "w");
    AB_B = fopen("./result/A&B_B.gtf", "w");
    A_B = fopen("./result/A-B.gtf", "w");
    B_A = fopen("./result/B-A.gtf", "w");
    printf("Searching\n");
    //对两个数据集分别进行字典树全查找
    name_oversearch(root_A, data_B, read_num_B, AB_B, B_A);
    name_oversearch(root_B, data_A, read_num_A, AB_A, A_B);
    printf("------------------Search over-----------------\n");
}
```