```
//Program      : Biodiff
//Explaination : To compare the difference between two Bioinformatics files.
//Author       : YuanEnming
//Usage        : Biodiff [options] from-file to-file
//Example      : Biodiff -co -a 3,4 -b 3,4 fileA fileB
//Example      : Biodiff -no -a 0 -b 8 fileA fileB
//Example      : Biodiff -ce -a 3,4 -b 3,4 fileA fileB
//Example      : Biodiff -ne -a 0 -b 8 fileA fileB
//Date         : 2017/06/01

#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <time.h>

#define FILE_BUFFER 1024
#define LINE_BUFFER 512
#define COLUMN_SIZE 256
#define BRANCH_SIZE 128
#define NOTEXIST 0
#define EXIST 1
#define SEPARATORS '\t'

/*****************************************************************************/

struct TrieNode /* the defination of structure TrieNode. */
{
    int exist;
    struct TrieNode *next[BRANCH_SIZE];
};

typedef struct TrieNode TrieNode;



/* an information function */
void Info(int option);
/* c_equal: coordinated-based equivalent differences */
void c_equal(char *col_A,char *col_B, FILE *fileA, FILE *fileB,FILE *fileAB_A,FILE *fileAB_B, FILE *A_B,
FILE *B_A);
/* n_diff: name-based equivalent & overlap differences */
void n_diff(int col_A, int col_B, FILE *fileA, FILE *fileB,FILE *fileAB_A, FILE *AB_B, FILE *A_B, FILE
*B_A, int mode);
/* c_overlap: coordinated-based overlap differences*/
void c_overlap(char *col_A, char *col_B, char *file_name_a,char *file_name_b, FILE *fileA,FILE *fileB, FILE
*fileAB_A, FILE *fileAB_B,FILE *fileA_B, FILE *fileB_A);
/* create a tire tree root */
TrieNode *Create_tire(void);
/* insert a node to the trie tree */
void Insert_trie(TrieNode *root, char *word);
/* search for a string according to a trie tree based on total equal*/
int Search_trie1(TrieNode *root, char *word);
/* search for a string according to a trie tree based on prefix equal*/
int Search_trie2(TrieNode *root, char *word);
/* Get_col: get a specific column from a line with separators according to c */
char *Get_col(char *line, char *col, char separator, int c);
/* get_row: to get the number of rows from a specific file. */
int Get_row(char *file);
/* index_: creat index from 1 to row. */
int *index_(int row);
/* advector: create an adjoint vector for the whole file to mark whether a row is overlap.*/
int *advector(int row);
/* store_col: get and store the specific column from a file.*/
char** store_col(int row, char *file_name, int column);

/* define globle variables for cmp_A & cmp_B */
char ***Columns_A;
char ***Columns_B;
/* cmp_A & cmp_B : the comparison function for qsort. */
int cmp_A(const void *a, const void *b);
int cmp_B(const void *a, const void *b);
/*****************************************************************************/
int main(int argc, char *argv[])
{
    FILE *fileA, *fileB, *fileAB_A, *fileAB_B, *fileA_B, *fileB_A;


    if(argc == 1)
        Info(0);     /* print the usage information */
    else if(argc != 8)
        Info(1);     /* usage error */
    if ((fileA = fopen(argv[6],"r")) == NULL || (fileB = fopen(argv[7],"r")) == NULL)
        Info(2);     /* open the input file . fileA and fileB should be openable*/

    /* set buffer for the input streams*/
    setvbuf(fileA, NULL, _IOFBF, FILE_BUFFER);
    setvbuf(fileB, NULL, _IOFBF, FILE_BUFFER);

    /* create target files */
```

```
    if (!(fileAB_A = fopen("A&B_A", "w")) ||
        !(fileAB_B = fopen("A&B_B", "w")) ||
        !(fileA_B  = fopen("A-B", "w")) ||
        !(fileB_A  = fopen("B-A", "w")))
        Info(3);     /* create false */

    /* set buffer for the output streams */
    setvbuf(fileAB_A, NULL, _IOFBF, FILE_BUFFER);
    setvbuf(fileAB_B, NULL, _IOFBF, FILE_BUFFER);
    setvbuf(fileA_B, NULL, _IOFBF, FILE_BUFFER);
    setvbuf(fileB_A, NULL, _IOFBF, FILE_BUFFER);

    /* to record the time */
    clock_t start = clock();
    if (!strcmp(argv[1], "-ce"))  /* use [-ce] mode */
        c_equal(argv[3],argv[5],fileA,fileB,fileAB_A,fileAB_B,fileA_B,fileB_A);
    else if (!strcmp(argv[1], "-ne")) /* use [-ne] mode */
        n_diff(atoi(argv[3]),atoi(argv[5]),fileA,fileB,fileAB_A,fileAB_B,fileA_B,fileB_A, 1);
    else if (!strcmp(argv[1], "-no")) /* use [-no] mode */
        n_diff(atoi(argv[3]),atoi(argv[5]),fileA,fileB,fileAB_A,fileAB_B,fileA_B,fileB_A, 2);
    else if (!strcmp(argv[1], "-co")) /* use [-co] mode */
        c_overlap(argv[3], argv[5], argv[6], argv[7], fileA,fileB,fileAB_A,fileAB_B,fileA_B,fileB_A);
    else          /* usage error */
        Info(4);
    clock_t end = clock();
    printf("%lu min %lu s %lu ms\n", (end - start) / (60 * CLOCKS_PER_SEC), (end - start) % (60 *
CLOCKS_PER_SEC) / CLOCKS_PER_SEC, (end - start) % (60 * CLOCKS_PER_SEC) % CLOCKS_PER_SEC * 1000 /
CLOCKS_PER_SEC);

    /* close the opend files */
    fclose(fileA);
    fclose(fileB);
    fclose(fileAB_A);
    fclose(fileAB_B);
    fclose(fileA_B);
    fclose(fileB_A);

    printf("Complete!\n");
    return 0;
}
/*****************************************************************************/
/* create a tire tree root */
TrieNode *Create_tire(void)
{
    TrieNode *temp = (TrieNode *) malloc(sizeof(TrieNode)); /* apply for space */
    temp -> exist = NOTEXIST;                               /* initialization */
    for(int i = 0; i < BRANCH_SIZE; i++)                    /* initialization */
        temp -> next[i] = NULL;
    return temp;
}
/*****************************************************************************/
/* insert a node to the trie tree */
void Insert_trie(TrieNode * root, char *col)
{
    TrieNode *temp = root;
    for(int i; *col; col++)
    {
        i = *col;
        if (temp -> next[i])           /* node existed already */
            ;
        else
            temp -> next[i] = Create_tire(); /* create a new node */
        temp = temp -> next[i];           /* point to next node */
    }
    temp -> exist = EXIST;               /* complete an insertion and record it */
}

/*****************************************************************************/
/* search for a string according to a trie tree based on total equal.*/
int Search_trie1(TrieNode *root, char *str)
{
    int i;
    TrieNode *temp = root;
    if (!temp)  /* tire tree must not be empty */
        return 0;
    for(int i; *str; str++)
    {
        i = *str;
        if(temp -> next[i])    /* if a specific character exist, point to the next one*/
            temp = temp -> next[i];
        else           /* not match */
            return 0;
    }
    if (temp -> exist)  /* match */
        return EXIST;
    else                /* include but not equal */
        return NOTEXIST;
}
```

```
/*****************************************************************************/
/* search for a string according to a trie tree based on prefix*/
int Search_trie2(TrieNode *root, char *str)
{
    int i;
    TrieNode *temp = root;
    if (!temp)  /* tire tree must not be empty */
    return 0;
    for(int i; *str; str++)
    {
        i = *str;
        if(temp -> next[i])    /* if a specific character exist, point to the next one*/
            temp = temp -> next[i];
        else           /* not match */
            return NOTEXIST;
    }
    return EXIST;      /* include */
}

/*****************************************************************************/
/* an information function to help users*/
void Info(int option)
{
    switch (option)
    {
        case 0:
            printf("#########################################################################\n");
            printf("#                                Biodiff                                #\n");
            printf("#                        Author : Yuan Enming                           #\n");
            printf("#             A progrem to compare two input files                      #\n");
            printf("#########################################################################\n");
            printf("#  Usage: Biodiff [options] from-file to-file                           #\n");
            printf("#  Example: Biodiff -ce -a 3,4 -b 3,4 fileA fileB                       #\n");
            printf("#  Example: Biodiff -ne -a 0 -b 8 fileA fileB                           #\n");
            printf("#  Example: Biodiff -co -a 3,4 -b 3,4 fileA fileB                       #\n");
            printf("#  Example: Biodiff -no -a 0 -b 8 fileA fileB                           #\n");
            printf("#########################################################################\n");
            printf("#  > * [-ce] : coordinate-based equivalent comparation;                 #\n");
            printf("#  > * [-ne] : name-based equivalent comparation;                       #\n");
            printf("#  > * [-co] : coordinate-based overlap comparation;                    #\n");
            printf("#  > * [-no] : name-based overlap comparation;                          #\n");
            printf("#  > * In [-ne]or[-no] mode you only need to select one column;        #\n");
            printf("#  > * In [-ce]or[-co] mode 2 columns separated by ',' are required #\n");
            printf("#  > * Here 'name-based overlap' means that the name prefix overlap #\n");
            printf("#########################################################################\n");
            exit(1);
            break;
        case 1:
            printf("Usage: Biodiff [-ce -ne -co -no] -a col_a -b col_b fileA fileB.\n");
            exit(1);
        case 2:
            printf("Error: Can not open the input files.\n");
            exit(1);
        case 3:
            printf("Error: Can not create the output files.\n");
            exit(1);
        case 4:
            printf("Usage: Biodiff [-ce -ne -co -no] -a col_a -b col_b fileA fileB.\n");
            printf("       You should choose one mode.\n");
            exit(1);
    }
}
/*****************************************************************************/
/* c_equal: coordinated-based equivalent differences */
void c_equal(char *col_A, char *col_B, FILE *fileA,
            FILE *fileB, FILE *fileAB_A, FILE *fileAB_B,
            FILE *fileA_B, FILE *fileB_A)
{
    char colA1s[4], colA2s[4], colB1s[4], colB2s[4];
    Get_col(col_A, colA1s, ',', 1);
    Get_col(col_A, colA2s, ',', 2);
    Get_col(col_A, colB1s, ',', 1);
    Get_col(col_B, colB2s, ',', 2);
    int col_A1 = atoi(colA1s);
    int col_A2 = atoi(colA2s);
    int col_B1 = atoi(colB1s);
    int col_B2 = atoi(colB2s);

    char line_A[LINE_BUFFER];       /* A buffer to store a line from fileA*/
    char line_B[LINE_BUFFER];       /* A buffer to store a line from fileB*/
    char column_A1[COLUMN_SIZE];    /* A buffer to store a column from fileA's line*/
    char column_A2[COLUMN_SIZE];    /* A buffer to store a column from fileA's line*/
    char column_B1[COLUMN_SIZE];    /* A buffer to store a column from fileB's line*/
    char column_B2[COLUMN_SIZE];    /* A buffer to store a column from fileB's line*/
    TrieNode *root_A = Create_tire();   /* create a root node*/
    TrieNode *root_B = Create_tire();

    while (fgets(line_A, LINE_BUFFER, fileA) != NULL)/* build a tire tree according to fileA */
    {
```

```
        if (*line_A == '\n')
            ;    /* skip the empty lines */
        else
        {
            Get_col(line_A, column_A1, SEPARATORS, col_A1);
            Get_col(line_A, column_A2, SEPARATORS, col_A2);
            strcat(column_A1, column_A2);
            Insert_trie(root_A, column_A1);  /* insert a string to the tire tree*/
        }
    }

    while (fgets(line_B, LINE_BUFFER, fileB) != NULL)/* search and write to files A&B_B and B-A */
    {
        if (*line_B == '\n')
            ;   /* skip the empty lines*/
        else
        {
            Get_col(line_B, column_B1,SEPARATORS, col_B1);
            Get_col(line_B, column_B2,SEPARATORS, col_B2);
            strcat(column_B1, column_B2);
            if (Search_trie1(root_A, column_B1)) /* write to file A&B_B*/
                fprintf(fileAB_B, "%s", line_B);
            else
                fprintf(fileB_A, "%s", line_B); /* write to file B-A */
            Insert_trie(root_B, column_B1);     /* build a tire tree according to fileB*/
        }
    }
    free(root_A);   /* release the storage of root_A*/
    fseek(fileA, 0, SEEK_SET);  /* move the pointer to the start of fileA*/

    while (fgets(line_A, LINE_BUFFER, fileA) != NULL)/* search and write to the files A&B_A and A-B */
    {
        if (*line_A == '\n')
            ;  /* skip the empty lines*/
        else
        {
            Get_col(line_A, column_A1, SEPARATORS, col_A1);
            Get_col(line_A, column_A2, SEPARATORS, col_A2);
            strcat(column_A1, column_A2);
            if (Search_trie1(root_B, column_A1))  /* write to file A&B_A */
                fprintf(fileAB_A, "%s", line_A);
            else
                fprintf(fileA_B, "%s", line_A);  /* write to file A-B */
        }
    }
    free(root_B);   /* release the storage of root_B*/
}

/*****************************************************************************/
/* c_overlap: coordinated-based overlap differences*/
void c_overlap(char *col_A, char *col_B,char *file_A, char *file_B,
               FILE *fileA, FILE *fileB, FILE *fileAB_A, FILE *fileAB_B,
               FILE *fileA_B, FILE *fileB_A)
{
    int row_A = Get_row(file_A);         /* get the number of rows of fileA*/
    int row_B = Get_row(file_B);         /* get the number of rows of fileB*/
    char colA1s[4], colA2s[4], colB1s[4], colB2s[4];
    Get_col(col_A, colA1s, ',', 1);      /* get the column number from command line arguements */
    Get_col(col_A, colA2s, ',', 2);
    Get_col(col_A, colB1s, ',', 1);
    Get_col(col_B, colB2s, ',', 2);
    int col_A1 = atoi(colA1s);           /* transfer string to int */
    int col_A2 = atoi(colA2s);
    int col_B1 = atoi(colB1s);
    int col_B2 = atoi(colB2s);
    int *index_A = index_(row_A);        /* create index */
    int *index_B = index_(row_B);
    int *advector_A = advector(row_A);   /* create adjoint vector*/
    int *advector_B = advector(row_B);
    Columns_A = (char***)malloc(sizeof(char**) * 3);  /* allocal space */
    Columns_B = (char***)malloc(sizeof(char**) * 3);
    char **lines_A;
    char **lines_B;
    Columns_A[1] = store_col(row_A, file_A, col_A1);  /* get and store the specific columnn from a file */
    Columns_A[2] = store_col(row_A, file_A, col_A2);
    Columns_B[1] = store_col(row_B, file_B, col_B1);
    Columns_B[2] = store_col(row_B, file_B, col_B2);

    qsort(index_A + 1, row_A, sizeof(index_A[1]), cmp_A);/* qsort the index according to the left end
point*/
    qsort(index_B + 1, row_B, sizeof(index_B[1]), cmp_B);

    int i, j, temp;
    /* judge whether the coordinate is overlap and mark in the adjoint vector. */
    for(i=1, j=1; i <= row_A; ++i) /* mark on advector when B's left end point is between A's left & right
end point.*/
    {
        for(; j <= row_B; ++j)
        {
```

```
            /* skip extra B when B's left end point is smaller than A' left end point */
            if (atoi(Columns_A[1][index_A[i]]) > atoi(Columns_B[1][index_B[j]])) continue;
            else
                for(temp = j; temp<row_B; ++temp) /* search for target B and mark both A and B */
                {
                    /* break when B's left end point is biger than A's right point */
                    if(atoi(Columns_A[2][index_A[i]]) < atoi(Columns_B[1][index_B[temp]]))
                        break;
                    else if(advector_B[index_B[temp]]==1) continue; /* skip the B which has already been
marked */
                    else
                        advector_A[index_A[i]] = advector_B[index_B[temp]] = EXIST; /* mark on both A&B's
advector*/
                }
            break;
        }
    }
    for(i=1, j=1; j <= row_B; ++j)/* mark on advector when A's left end point is between B's left & right
end point.*/
    {
        for(; i <= row_A; ++i)
        {
            /* skip extra A when A's left end point is smaller than B' left end point */
            if (atoi(Columns_B[1][index_B[j]]) > atoi(Columns_A[1][index_A[i]])) continue;
            else
                for(temp = i; temp<row_A; ++temp) /* search for target B and mark both A and B */
                {
                    /* break when A's left end point is biger than B's right point */
                    if(atoi(Columns_B[2][index_B[j]]) < atoi(Columns_A[1][index_A[temp]]))
                        break;
                    else if(advector_A[index_A[temp]]==1) continue;  /* skip the B which has already been
marked */
                    else
                        advector_A[index_A[temp]] = advector_B[index_B[j]] = EXIST; /* mark on both A&B's
advector*/
                }
            break;
        }
    }
    /* print every line to target files according to the index and adjoint vector.*/
    char line[LINE_BUFFER];
    for (i = 1; i <= row_A; ++i)
    {
        fgets(line, LINE_BUFFER, fileA);
        if (advector_A[i])
            fprintf(fileAB_A, "%s", line);
        else
            fprintf(fileA_B,  "%s", line);
    }
    for (i = 1; i <=row_B; ++i)
    {
        fgets(line, LINE_BUFFER, fileB);
        if (advector_B[i])
            fprintf(fileAB_B, "%s", line);
        else
            fprintf(fileB_A,  "%s", line);
    }

    /* close files and free the space */
    fclose(fileA);
    fclose(fileB);
    fclose(fileA_B);
    fclose(fileB_A);
    fclose(fileAB_A);
    fclose(fileAB_B);

    /* free the space of dynamic variables */
    for (i = 1; i < 3; ++i) {
        for (j = 1; j < row_A + 1; ++j) {
            free(Columns_A[i][j]);
        }
        free(Columns_A[i]);
    }
    free(Columns_A);

    for (i = 1; i < 3; ++i) {
        for (j = 1; j < row_B + 1; ++j) {
            free(Columns_B[i][j]);
        }
        free(Columns_B[i]);
    }
    free(Columns_B);

}


/*****************************************************************************/
/* n_diff: name-based equivalent & overlap differences */
void n_diff(int col_A, int col_B,
```

```
           FILE *fileA, FILE *fileB, FILE *fileAB_A,
           FILE *fileAB_B, FILE *fileA_B, FILE *fileB_A, int mode)
{
    char line_A[LINE_BUFFER];
    char line_B[LINE_BUFFER];
    char columnA[COLUMN_SIZE];
    char columnB[COLUMN_SIZE];
    TrieNode *root_A = Create_tire();
    TrieNode *root_B = Create_tire();
    /* bulid a tire tree according to fileA */
    while (fgets(line_A, LINE_BUFFER, fileA) != NULL)
    {
        if (*line_A == '\n')
            ;   /* skip the empty lines*/
        else
        {
            Get_col(line_A, columnA, SEPARATORS, col_A);
            Insert_trie(root_A, columnA); /* insert a string into the tire tree */
        }
    }
    /* search and write to the files A&B_A and A-B */
    while (fgets(line_B, LINE_BUFFER, fileB) != NULL)
    {
        if(*line_B == '\n')
            ;   /* skip the empty lines */
        else
        {
            Get_col(line_B, columnB, SEPARATORS, col_B);
            if(mode==1?Search_trie1(root_A, columnB):Search_trie2(root_A,columnB))  /* write to file
A&B_B*/
                fprintf(fileAB_B,"%s", line_B);
            else
                fprintf(fileB_A, "%s", line_B);     /* write to file B-A */
            Insert_trie(root_B, columnB);        /* build a tire tree according to fileB*/
        }
    }
    free(root_A);     /* release the storage of root_A*/
    fseek(fileA, 0, SEEK_SET);      /* move the pointer to the start of fileA*/

    /* search and write to the files A&B_A and A-B */
    while (fgets(line_A, LINE_BUFFER, fileA) != NULL)
    {
        if (*line_A == '\n')
            ;    /* skip the empty lines*/

        else
        {
            Get_col(line_A, columnA, SEPARATORS, col_A);
            if ( mode==1?Search_trie1(root_B, columnA):Search_trie2(root_B,columnA))    /* write to file
A&B_A */
                fprintf(fileAB_A, "%s", line_A);
            else
                fprintf(fileA_B, "%s", line_A);   /* write to file A-B */
        }
    }
    free(root_B);   /* release the storage of root_B*/
}


/*****************************************************************************/
/* Get_col: get a specific column from a line with separators according to c */
char *Get_col(char *line, char *col, char separator, int c)
{
    int count = 1; /* To count the occurrences of separators.*/
    if (c < 1)
    return NULL;
    while (*line != '\0' && *line == separator )
    line++;    /* To skip the separators at the beginning of the line.*/
    while (*line != '\0' && count < c)
    {
        if (*line == separator)
        {
            while (*line != '\0' && *line == separator)
            line++;        /* many separators are put together.*/
            count++;
        }
        else
        line++;
    }
    while (*line != '\0' && *line != '\n' && *line != separator) /* get specific column */
       *col++ = *line++;
    *col = 0;
    return col;

}


/*****************************************************************************/
/*Get_row: to get the number of rows from a specific file. */
```

```
int Get_row(char *file_name)
{
    FILE *file = fopen(file_name, "r");
    char line[LINE_BUFFER];
    int i;
    for (i = 0; fgets(line, LINE_BUFFER, file); ++i)
    {
        if (!strcmp(line, "\n")) --i; /* skip empty lines */
    }
    fclose(file);
    return i;
}

/*****************************************************************************/
/* index_: creat index from 1 to row. */
int *index_(int row)
{
    int *temp = (int *)malloc(sizeof(int)*(row + 1));
    for(int i = 1; i <= row; ++i) temp[i] = i;
    return temp;
}

/*****************************************************************************/
/*advector: create an adjoint vector for the whole file to mark whether a row is overlap.*/
int *advector(int row)
{
    int *temp = (int *)malloc(sizeof(int)*(row + 1));
    for(int i = 1; i <= row; ++i) temp[i] = NOTEXIST;
    return temp;
}

/*****************************************************************************/
/*store_col: get and store the specific column from a file.*/
char** store_col(int row, char *file_name, int column)
{
    FILE *file = fopen(file_name, "r");
    char line[LINE_BUFFER];
    char col[COLUMN_SIZE];
    int i, j, k, l;
    char **columns = (char**)malloc(sizeof(char*) * (row + 1));  /* allocal space for pointer */
    for (l = 1; fgets(line, LINE_BUFFER, file); ++l)
    {
        if (!strcmp(line, "\n"))    /* skip the empty lines*/
        {
            --l;
            continue;
        }
        Get_col(line, col, SEPARATORS, column);  /* get a specific column */
        columns[l] = (char*)malloc(sizeof(char) * (strlen(col)+1)); /* allocal space for column */
        for(int i = 0; i <= strlen(col); i++)
            columns[l][i] = col[i];
    }
    fclose(file);
    return columns;
}

/*****************************************************************************/
/* cmp_A & cmp_B : the comparison function for qsort. */
/* first according to left end point then right end point */
char ***Columns_A; /* define globle variables for cmp_A & cmp_B */
char ***Columns_B;
int cmp_A(const void *a, const void *b)
{
    if (strcmp(Columns_A[1][*(int*)a], Columns_A[1][*(int*)b]) < 0)
        return -1;
    else if (strcmp(Columns_A[1][*(int*)a], Columns_A[1][*(int*)b]) > 0)
        return 1;
    else if (strcmp(Columns_A[2][*(int*)a], Columns_A[2][*(int*)b]) < 0)
        return -1;
    else if (strcmp(Columns_A[2][*(int*)a], Columns_A[2][*(int*)b]) > 0)
        return 1;
    else
        return 0;
}

int cmp_B(const void *a, const void *b)
{
    if (strcmp(Columns_B[1][*(int*)a], Columns_B[1][*(int*)b]) < 0)
        return -1;
    else if (strcmp(Columns_B[1][*(int*)a], Columns_B[1][*(int*)b]) > 0)
        return 1;
    else if (strcmp(Columns_B[2][*(int*)a], Columns_B[2][*(int*)b]) < 0)
        return -1;
    else if (strcmp(Columns_B[2][*(int*)a], Columns_B[2][*(int*)b]) > 0)
        return 1;
    else
        return 0;
}
/*****************************************************************************/
```