Недобежкин Павел Владимирович

ПММ, 61 группа, 4 курс

Отчет по лабораторной работе №3

Задание 1.
Найти наилучшее локальное выравнивание между двумя строками нуклеотидных

или аминокислотных последовательностей. Файлы для примера брать в соответствии

с вариантом из Лабораторной работы 2 из базы данных GenBank.

Входные данные. Две строки V и W и матрица весов.

Выходные данные. Локальное выравнивание, определяемое подстроками строк

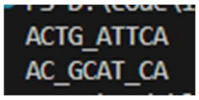
V и W, глобальное выравнивание которых определенное матрицей весов, является

наилучшим среди всех глобальных выравниваний всех подстрок V и W.
Код с примером

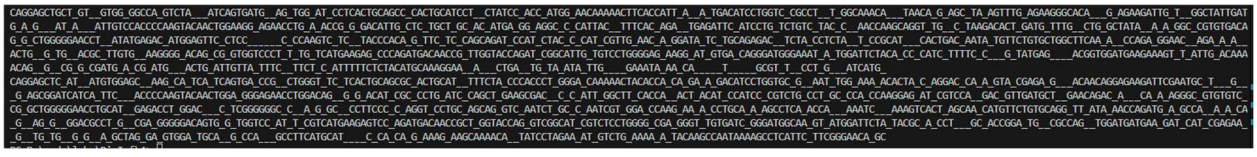
```
public class Main {
    public static void main(String[] args) {
        String s1 = "ACTGATTCA";
        String s2 = "ACGCATCA";
        int[][] matrix = { { 2, -3, -3, -3 }, { -3, 2, -3, -3 }, { -3, -3, 2, -3
}, { -3, -3, -3, 2 } };

        String[] res = NeedlemanWunschAlgorithm(s1, s2, matrix, -2);
        System.out.println(res[0]);
        System.out.println(res[1]);
    }

    static String nucleotidesToInd = "ATGC";

    public static String[] NeedlemanWunschAlgorithm(String s1, String s2, int[][]
matrixOfFine, int spaceFine) {
        int[][] matrix = new int[s1.length() + 1][s2.length() + 1];
        matrix[0][0] = 0;
        for (int i = 1; i <= s2.length(); i++) {
            matrix[0][i] = matrix[0][i - 1] + spaceFine;
        }
        for (int i = 1; i <= s1.length(); i++) {
            matrix[i][0] = matrix[i - 1][0] + spaceFine;
        }

        for (int i = 1; i <= s1.length(); i++) {
            for (int j = 1; j <= s2.length(); j++) {
                matrix[i][j] = Math.max(Math.max(matrix[i - 1][j] + spaceFine,
matrix[i][j - 1] + spaceFine),
                        matrix[i - 1][j - 1] +
matrixOfFine[nucleotidesToInd.indexOf(s1.charAt(i - 1))][nucleotidesToInd
                                .indexOf(s2.charAt(j - 1))]);
            }
```

```
        }

        int i = s1.length();
        int j = s2.length();
        String[] result = new String[2];
        result[0] = "";
        result[1] = "";

        while (i != 0 || j != 0) {
            if (matrix[i - 1][j - 1] > matrix[i - 1][j] && matrix[i - 1][j - 1] >
matrix[i][j - 1]) {
                result[0] = s1.charAt(i - 1) + result[0];
                result[1] = s2.charAt(j - 1) + result[1];
                i--;
                j--;
            } else if (matrix[i - 1][j] > matrix[i - 1][j - 1] && matrix[i -
1][j] > matrix[i][j - 1]) {
                result[0] = s1.charAt(i - 1) + result[0];
                result[1] = "_" + result[1];
                i--;
            } else {
                result[0] = "_" + result[0];
                result[1] = s2.charAt(j - 1) + result[1];
                j--;
            }
        }

        return result;
    }
}
```

Результат выполнения примера



Результат выполнения собственного варианта со своей матрицей замен



Найти наилучшее локальное выравнивание между двумя строками нуклеотидных или аминокислотных последовательностей. Файлы для примера брать в соответствии с вариантом из Лабораторной работы 2 из базы данных GenBank.

Входные данные. Две строки V и W и матрица весов.

Выходные данные. Локальное выравнивание, определяемое подстроками строк V и W, глобальное выравнивание которых определенное матрицей весов, является наилучшим среди всех глобальных выравниваний всех подстрок V и W.

Код с примером

```
public class Main {
    public static void main(String[] args) {
        String s1 = "ATGCATCCCATGAC";
        String s2 = "TCTATATCCGT";
        int[][] matrix = { { 2, -3, -3, -3 }, { -3, 2, -3, -3 }, { -3, -3, 2, -3
}, { -3, -3, -3, 2 } };

        String[] res = SmithWatermanAlgorithm(s1, s2, matrix, -2);
        System.out.println(res[0]);
        System.out.println(res[1]);
    }

    static String nucleotidesToInd = "ATGC";

    public static String[] SmithWatermanAlgorithm(String s1, String s2, int[][]
matrixOfFine, int spaceFine) {
        int[][] matrix = new int[s1.length() + 1][s2.length() + 1];
        for (int i = 0; i <= s2.length(); i++) {
            matrix[0][i] = 0;
        }
        for (int i = 1; i <= s1.length(); i++) {
            matrix[i][0] = 0;
        }

        int iMax = 0, jMax = 0;

        for (int i = 1; i <= s1.length(); i++) {
            for (int j = 1; j <= s2.length(); j++) {
                matrix[i][j] = Math.max(Math.max(matrix[i - 1][j] + spaceFine,
matrix[i][j - 1] + spaceFine), Math.max(
                        matrix[i - 1][j - 1] +
matrixOfFine[nucleotidesToInd.indexOf(s1.charAt(i - 1))][nucleotidesToInd
                                .indexOf(s2.charAt(j - 1))],
                        0));
                if (matrix[i][j] > matrix[iMax][jMax]) {
                    iMax = i;
                    jMax = j;
                }
            }
        }

        String[] result = new String[2];
        result[0] = "";
        result[1] = "";

        while (matrix[iMax][jMax] != 0) {
            if (matrix[iMax - 1][jMax - 1] >= matrix[iMax - 1][jMax]
                    && matrix[iMax - 1][jMax - 1] >= matrix[iMax][jMax - 1]) {
                result[0] = s1.charAt(iMax - 1) + result[0];
```

```
                result[1] = s2.charAt(jMax - 1) + result[1];
                iMax--;
                jMax--;
            } else if (matrix[iMax - 1][jMax] >= matrix[iMax - 1][jMax - 1]
                    && matrix[iMax - 1][jMax] >= matrix[iMax][jMax - 1]) {
                result[0] = s1.charAt(iMax - 1) + result[0];
                result[1] = "_" + result[1];
                iMax--;
            } else {
                result[0] = "_" + result[0];
                result[1] = s2.charAt(jMax - 1) + result[1];
                jMax--;
            }
        }
        return result;
    }

}
```

Результат выполнение примера

```
ATCC
ATCC
```

Результат выполнения задания