

Сведения о документе.....	1
Упражнение 1. Строки.....	1
Упражнение 2. Считывание графов из файла, вывод на экран и в файл, нахождение Эйлера пути .....	5
Упражнение 3. Проверка изоморфности двух графов; нахождение в графе всех подграфов, изоморфных данному образцу .....	9

## Сведения о документе

Настоящий документ содержит несколько примеров использования функций библиотеки CBioInfCpp.h.

Настоящий документ распространяется на условиях лицензии Creative Commons Attribution 4.0 International Public License (сокращенно – CC BY, гиперссылка на текст лицензии: <https://creativecommons.org/licenses/by/4.0/legalcode.ru>).

Автор – Черноухов Сергей ([chernouhov@rambler.ru](mailto:chernouhov@rambler.ru))

Код по всем упражнениям также имеется в файле StudyingCBioInfCpp.cpp

## Упражнение 1. Строки

Скачиваем строки из файла, находим наименьшую надстроку "жадным" алгоритмом. Затем находим для нее reverse complement, вычисляем GC-состав, вычисляем Hamming distance между 2мя строками; выравниваем между собой две строки с вычислением редакционного расстояния, а также находим в заданной строке все подстроки, редакционное расстояние до которых от данной не более 1.

Начнем с подключения библиотеки. Разместите библиотеку в том каталоге, в котором лежат остальные подключаемые библиотеки (в этом случае к ней не надо будет прописывать полный путь), либо, при произвольном размещении, пропишите до нее полный путь (имени файла после директивы `#include` будет уже недостаточно).

Наш код в этом случае будет иметь следующий вид:

```
#include <CBioInfCpp.h>
// Including CBioInfCpp
// Подключаем библиотеку CBioInfCpp

using namespace std;

int main()
{
    return 0;
}
```

Все готово. Теперь давайте скачаем набор строк в файл. Для этого скачайте и разместите себе в рабочий каталог проекта файл со строками `In1(strings).txt` (либо, опять же, при произвольном размещении, укажите полный путь к файлу для `ifstream`).

Считывать строки мы будем в вектор `DataS`. Результат по данному упражнению будем выводить на экран и в файл `Out1(strings).txt`. Сделаем это (для проверки выведем результат считывания на экран и в файл с помощью команд `VectorCout` и `VectorFout`):

```
#include <CBioInfCpp.h>
// Including CBioInfCpp
// Подключаем библиотеку CBioInfCpp
```

```

using namespace std;

int main()
{
    vector<string> DataS;
    // Container for strings to be read from file In1(strings).txt
    DataS.clear();
    ifstream fin1 ("In1(strings).txt");
    // If you place file not to "standart place" please write the entire
    path
    ofstream fout1 ("Out1(strings).txt"); // File to write results for the
    Exercise 1.

    StringsRead(fin1, DataS); //Reading strings to vector DataS
    VectorCout(DataS); // Out DataS to screen
    VectorFout(DataS, fout1); // Out dataS to file

    fin1.close();
    fout1.close();

    return 0;
}

```

Получилось? Прибавим же к нашему массиву (вектору) строк еще пару пустых (для  
верности) и узнаем, какую наименьшую надстроку нам найдет «жадный» алгоритм  
(функция **ShortestSuperstring**):

```

int main()
{
    vector<string> DataS;
    // Container for strings to be read from file In1(strings).txt
    DataS.clear();
    ifstream fin1 ("In1(strings).txt");
    // If you place file not to "standard place" please write the entire
    path
    ofstream fout1 ("Out1(strings).txt"); // File to write results for the
    Exercise 1.

    StringsRead(fin1, DataS); //Reading strings to vector DataS
    VectorCout(DataS); // Out DataS to screen
    VectorFout(DataS, fout1); // Out DataS to file

    DataS.push_back(""); //adding empty string
    DataS.push_back(""); //another one
    string ShortestSuperstring = ShortSuperstringGr(DataS);
    // Generating shortest supersring using greedy algorithm
    cout<<"ShortestSuperstring: "<<ShortestSuperstring<<endl; // out it to
    screen
    fout1<<"ShortestSuperstring: "<<ShortestSuperstring<<endl; // out it
    too file

    fin1.close();
    fout1.close();

    return 0;
}

```

Видите? А теперь задачки для любителей биоинформатики.  
Начнем с того, что найдем для строки ShortestSuperstring GC-состав, reverse  
complement, а для последней – и соответствующую ей РНК:

```

int main()
{

```

```

vector <string> DataS;
// Container for strings to be read from file In1(strings).txt
DataS.clear();
ifstream fin1 ("In1(strings).txt");
// If you place file not to "standard place" please write the entire
path
ofstream fout1 ("Out1(strings).txt"); // File to write results for the
Exercise 1.

StringsRead(fin1, DataS); //Reading strings to vector DataS
VectorCout(DataS); // Out DataS to screen
VectorFout(DataS, fout1); // Out DataS to file

DataS.push_back(""); //adding empty string
DataS.push_back(""); //another one
string ShortestSuperstring = ShortSuperstringGr(DataS);
// Generating shortest superstring using greedy algorithm
cout<<"ShortestSuperstring: "<<ShortestSuperstring<<endl; // out it to
screen
fout1<<"ShortestSuperstring: "<<ShortestSuperstring<<endl; // out it
too file

cout<<"Reverse complement of ShortestSuperstring:"
<<rp(ShortestSuperstring)<<endl;
// generating and printing to screen reverse complement of
ShortestSuperstring
fout1<<"Reverse complement of ShortestSuperstring: "
<<rp(ShortestSuperstring)<<endl;
// generating and printing to file reverse complement of
ShortestSuperstring

cout<<"GC-content of Reverse complement of ShortestSuperstring: "
<<gcDRNA (rp(ShortestSuperstring))<<endl;
// calculating GC-content of this reverse complement (to screen)
fout1<<"GC-content of Reverse complement of ShortestSuperstring: "
<<gcDRNA(rp(ShortestSuperstring))<<endl; // calculating GC-content of
this reverse complement (to file)

string RNAofShortestSuperstring;
RNAfromDNA (ShortestSuperstring, RNAofShortestSuperstring); // now let's
generate RNA string upon ShortestSuperstring and write it to
RNAofShortestSuperstring
cout<<"RNAofShortestSuperstring: "<<RNAofShortestSuperstring<<endl;
fout1<<"RNAofShortestSuperstring: "<<RNAofShortestSuperstring<<endl;

fin1.close();
fout1.close();

return 0;
}

```

Ну а теперь давайте посчитаем Hamming distance между ShortestSuperstring и RNAofShortestSuperstring, затем «укоротим» RNAofShortestSuperstring на 2 символа и выровняем RNAofShortestSuperstring и RNAofShortestSuperstring, а наконец – найдем в строке ShortestSuperstring все ее подстроки, редакционной расстояние от которых до строки ACA" будет<=1 (функция [FindMutatedRepeatsED](#)).

```

int main()
{
    vector <string> DataS;
    // Container for strings to be read from file In1(strings).txt
    DataS.clear();
    ifstream fin1 ("In1(strings).txt");

```

```

// If you place file not to "standard place" please write the entire
path
ofstream fout1 ("Out1(strings).txt"); // File to write results for the
Exercise 1.

StringsRead(fin1, DataS); //Reading strings to vector DataS
VectorCout(DataS); // Out DataS to screen
VectorFout(DataS, fout1); // Out DataS to file

DataS.push_back(""); //adding empty string
DataS.push_back(""); //another one
string ShortestSuperstring = ShortSuperstringGr(DataS);
// Generating shortest superstring using greedy algorithm
cout<<"ShortestSuperstring: "<<ShortestSuperstring<<endl; // out it to
screen
fout1<<"ShortestSuperstring: "<<ShortestSuperstring<<endl; // out it
too file

cout<<"Reverse complement of ShortestSuperstring:"
<<rp(ShortestSuperstring)<<endl;
// generating and printing to screen reverse complement of
ShortestSuperstring
fout1<<"Reverse complement of ShortestSuperstring: "
<<rp(ShortestSuperstring)<<endl;
// generating and printing to file reverse complement of
ShortestSuperstring

cout<<"GC-content of Reverse complement of ShortestSuperstring: "
<<gcDNA (rp(ShortestSuperstring))<<endl;
// calculating GC-content of this reverse complement (to screen)
fout1<<"GC-content of Reverse complement of ShortestSuperstring: "
<<gcDNA(rp(ShortestSuperstring))<<endl; // calculating GC-content of
this reverse complement (to file)

string RNAofShortestSuperstring;
RNAfromDNA (ShortestSuperstring, RNAofShortestSuperstring); // now let's
generate RNA string upon ShortestSuperstring and write it to
RNAofShortestSuperstring
cout<<"RNAofShortestSuperstring: "<<RNAofShortestSuperstring<<endl;
fout1<<"RNAofShortestSuperstring: "<<RNAofShortestSuperstring<<endl;

let's calculate Hamming distance between ShortestSuperstring and
RNAofShortestSuperstring
cout<<"Hamming Distance between ShortestSuperstring and
RNAofShortestSuperstring: "<<HmDist(ShortestSuperstring,
RNAofShortestSuperstring)<<endl;
fout1<<"Hamming Distance between ShortestSuperstring and
RNAofShortestSuperstring: "<<HmDist(ShortestSuperstring,
RNAofShortestSuperstring)<<endl;

RNAofShortestSuperstring.pop_back();
// Deleting the last symbol from RNAofShortestSuperstring
RNAofShortestSuperstring.pop_back();
// Another time

string srl, sr2;
// These strings will contain results of Alignment (minimizing Edit
Distance)
cout<<"Edit Distance between ShortestSuperstring and
RNAofShortestSuperstring: "<<EditDistA(ShortestSuperstring,
RNAofShortestSuperstring, srl, sr2)<<endl;
fout1<<"Edit Distance between ShortestSuperstring and
RNAofShortestSuperstring: "<<EditDistA(ShortestSuperstring,
RNAofShortestSuperstring, srl, sr2)<<endl;

```

```

cout<<"Edit Distance Alignment: "<<endl<<sr1<<endl<<sr2<<endl;
fout1<<"Edit Distance Alignment: "<<endl<<sr1<<endl<<sr2<<endl;

// Now let's find in the string ShortestSuperstring all its substrings
that have edit distance to "ACA" <=1.
std::set <std::pair <int, int>> Result;
// The function FindMutatedRepeatsED returns results as set of pairs
there the first one is a starting position of substring in the string,
and the second is its lenght

FindMutatedRepeatsED ("ACA", ShortestSuperstring, 1, Result); //done
cout<<"Substrings of ShortestSuperstring that have EditDistance to
string ACA <=1 : "<<endl;
fout1<<"Substrings of ShortestSuperstring that have EditDistance to
string ACA <=1 : "<<endl;

for (auto it=Result.begin(); it!=Result.end(); it++)
{
    cout<<ShortestSuperstring.substr((*it).first,
    (*it).second)<<endl; //outing substrings to screen
    fout1<<ShortestSuperstring.substr((*it).first,
    (*it).second)<<endl; //outing substrings to file
}

fin1.close();
fout1.close();

return 0;
}

```

Ну вот и все. Результат должен быть и в файле Out1(strings).txt.

## Упражнение 2. Считывание графов из файла, вывод на экран и в файл, нахождение Эйлераового пути

Если мы начинаем с нуля (а не продолжаем предыдущее упражнение), опять же начнем с подключения библиотеки. Подробнее об этом – см. Упражнение 1.

Код:

```

#include <CBioInfCpp.h>
// Including CBioInfCpp
// Подключаем библиотеку CBioInfCpp

using namespace std;

int main()
{

    return 0;
}

```

Считываем из файлов невзвешенный граф, взвешенный граф с целочисленными ребрами и взвешенный граф с нецелочисленными ребрами. Считывать будем в вектора A1 и A2 первые два, а третий – в такую структуру `pair < vector<int>, vector<double>>` A3.

Отметим, что вершины графов здесь могут обозначаться и отрицательными числами.

Для этого нам сначала надо скачать файлы с графами и разместить их в папке проекта (или в произвольном месте, но с указанием полного пути к ним для ifstream). Файлы следующие:

```
InUnWeightedGraph.txt; // An unweighted graph is here (edge list format)
InWeightedGraphIntegers.txt // A weighted graph is here (edge list format,
weights are integers)
InWeightedGraphDoubles.txt // A weighted graph is here (edge list format,
weights are double)
```

Считывание невзвешенного графа производится с помощью функции **UWGraphRead**, взвешенного – **WGraphRead**. Вывод на экран графа производится с помощью функции **GraphCout**, а в файл – **GraphFout**.

Код же будет иметь вид:

```
#include <CBioInfCpp.h>
// Including CBioInfCpp
// Подключаем библиотеку CBioInfCpp

using namespace std;

int main()
{
    ifstream fin2_1 ("InUnWeightedGraph.txt"); // An unweighted graph is
    here (edge list format)
    ifstream fin2_2 ("InWeightedGraphIntegers.txt"); // A weighted graph
    is here (edge list format, weights are integers)
    ifstream fin2_3 ("InWeightedGraphDoubles.txt"); // A weighted graph is
    here (edge list format, weights are double)
    ofstream fout2 ("Out2(Graphs1).txt"); // File to write results for the
    Exercise 2.

    vector<int> A1; //Container for unweighted graph A1
    UWGraphRead(fin2_1, A1);
    cout<<"Here is unweighted graph A1: "<<endl;
    GraphCout(A1, false); //Out graph to screen, as it is unweighted the
    second parameter should be "false"
    fout2<<"Here is unweighted graph A1: "<<endl;
    GraphFout(A1, false, fout2); //Out graph to file, as it is unweighted
    the second parameter should be "false"

    vector<int> A2; //Container for weighted graph A2, weights are
    integers
    WGraphRead(fin2_2, A2);
    cout<<"Here is weighted graph A2: "<<endl;
    GraphCout(A2, true); //Out graph to screen, as it is weighted the
    second parameter should be "true"
    fout2<<"Here is weighted graph A2: "<<endl;
    GraphFout(A2, true, fout2); //Out graph to file, as it is weighted the
    second parameter should be "true"

    pair< vector<int>, vector<double>> A3; //Container for unweighted
    graph A3, weights are double
    WGraphRead(fin2_3, A3);
    cout<<"Here is weighted graph A3: "<<endl;
    GraphCout(A3); //Out graph to screen, as it is set by pair A3 it may be
    weighted only
    fout2<<"Here is weighted graph A3: "<<endl;
    GraphFout(A3, fout2); //Out graph to file, as it is set by pair A3 it
    may be weighted only

    fout2.close();
    fin2_1.close();
}
```

```

        fin2_2.close();
        fin2_3.close();

    return 0;
}

```

Мы научились считывать и выводить графы в файл и на экран. Обратим внимание, что графы A1, A2, A3 отличаются лишь весами ребер, т.е. решение задачи нахождения Эйлера пути/ цикла должно быть для них одинаковым. Убедимся в этом с помощью функции **EPathDGraph** (и не забываем, что она трактует подаваемые ей графы как ориентированные). Также не забудем, что она заполнит нам два вектора – R, где и будет – при наличии – искомым Эйлера путь (в этом случае функция вернет 2)/ цикл (в этом случае функция вернет 1) и I, который будет содержать перечень изолированных вершин в промежутке от минимальной до максимальной по номеру. Если путь/цикл найти не удалось, или же исходные данные некорректны, функция вернет -1, а R и I будут пустыми. Отметим еще раз, что вершины графов могут обозначаться и отрицательными числами.

```

#include <CBioInfCpp.h>
// Including CBioInfCpp
// Подключаем библиотеку CBioInfCpp

using namespace std;

int main()
{
    ifstream fin2_1 ("InUnWeightedGraph.txt"); // An unweighted graph is
    here (edge list format)
    ifstream fin2_2 ("InWeightedGraphIntegers.txt"); // A weighted graph
    is here (edge list format, weights are integers)
    ifstream fin2_3 ("InWeightedGraphDoubles.txt"); // A weighted graph is
    here (edge list format, weights are double)
    ofstream fout2 ("Out2(Graphs1).txt"); // File to write results for the
    Exercise 2.

    vector<int> A1; //Container for unweighted graph A1
    UWGraphRead(fin2_1, A1);
    cout<<"Here is unweighted graph A1: "<<endl;
    GraphCout(A1, false); //Out graph to screen, as it is unweighted the
    second parameter should be "false"
    fout2<<"Here is unweighted graph A1: "<<endl;
    GraphFout(A1, false, fout2); //Out graph to file, as it is unweighted
    the second parameter should be "false"

    vector<int> A2; //Container for weighted graph A2, weights are
    integers
    WGraphRead(fin2_2, A2);
    cout<<"Here is weighted graph A2: "<<endl;
    GraphCout(A2, true); //Out graph to screen, as it is weighted the
    second parameter should be "true"
    fout2<<"Here is weighted graph A2: "<<endl;
    GraphFout(A2, true, fout2); //Out graph to file, as it is weighted the
    second parameter should be "true"

    pair< vector<int>, vector<double>> A3; //Container for unweighted
    graph A3, weights are double
    WGraphRead(fin2_3, A3);
    cout<<"Here is weighted graph A3: "<<endl;
    GraphCout(A3); //Out graph to screen, as it is set by pair A3 it may be
    weighted only
    fout2<<"Here is weighted graph A3: "<<endl;
}

```

```
GraphFout(A3, fout2); //Out graph to file, as it is set by pair A3 it
may be weighted only
```

```
vector<int> R; //Container for Eulerian cycle/path to be found
vector<int> I; //Container for isolated vertices to be found
cout<<EPathDGraph (A1, R, false, I)<<endl; //as A1 is unweighted we
should set 3d parameter as "false"
fout2<<EPathDGraph (A1, R, false, I)<<endl;
// EPathDGraph returns value "1" if Eulerian cycle has been found or
value "2" if Eulerian path has been found
// or "-1" together with empty R and Isolates if no cycle/ path found.
//Note that EPathDGraph considers graps as directed ones.
cout<< "Eulerian cycle of graph A1: "<<endl;
VectorCout(R); // vector R - to screen
cout<< "Isolated vertices of graph A1: "<<endl;
VectorCout(I); // vector I - to screen
fout2<< "Eulerian cycle of graph A1: "<<endl;
VectorFout(R, fout2); // vector R - to file
fout2<< "Isolated vertices of graph A1: "<<endl;
VectorFout(I, fout2); // vector I - to file
```

```
cout<<EPathDGraph (A2, R, true, I)<<endl; //as A1 is unweighted we
should set 3d parameter as "true"
fout2<<EPathDGraph (A2, R, true, I)<<endl;
// EPathDGraph returns value "1" if Eulerian cycle has been found or
value "2" if Eulerian path has been found
// or "-1" together with empty R and Isolates if no cycle/ path found.
cout<< "Eulerian cycle of graph A2: "<<endl;
VectorCout(R); // vector R - to screen
cout<< "Isolated vertices of graph A2: "<<endl;
VectorCout(I); // vector I - to screen
fout2<< "Eulerian cycle of graph A2: "<<endl;
VectorFout(R, fout2); // vector R - to file
fout2<< "Isolated vertices of graph A2: "<<endl;
VectorFout(I, fout2); // vector I - to file
```

```
cout<<EPathDGraph (A3, R, I)<<endl; //as A3 is set by pair it may be
only weighted
fout2<<EPathDGraph (A3, R, I)<<endl;
// EPathDGraph returns value "1" if Eulerian cycle has been found or
value "2" if Eulerian path has been found
// or "-1" together with empty R and Isolates if no cycle/ path found.
cout<< "Eulerian cycle of graph A3: "<<endl;
VectorCout(R); // vector R - to screen
cout<< "Isolated vertices of graph A3: "<<endl;
VectorCout(I); // vector I - to screen
fout2<< "Eulerian cycle of graph A3: "<<endl;
VectorFout(R, fout2); // vector R - to file
fout2<< "Isolated vertices of graph A3: "<<endl;
VectorFout(I, fout2); // vector I - to file
```

```
fout2.close();
fin2_1.close();
fin2_2.close();
fin2_3.close();
```

```
return 0;
}
```

Результат см. также и в файле Out2(Graphs1).txt.



### Упражнение 3. Проверка изоморфности двух графов; нахождение в графе всех подграфов, изоморфных данному образцу

А теперь попробуем замахнуться на такие сложные задачи как проверка изоморфности двух графов и поиск всех подграфов, изоморфных данному образцу, в заданном графе. Для этого нам понадобится функция **SubGraphsInscribed**.

Подробнее о ее достижениях и эволюции – см. здесь:

<https://github.com/chernouhov/CBioInfCpp-0-/tree/master/TestsGraphIsomorphismInfo>

<https://github.com/chernouhov/CBioInfCpp-0-/tree/master/TestsIsomorphicSubGraphsFinding>

<https://doi.org/10.24108/preprints-3111977>

Опять же – если мы начинаем с нуля (а не продолжаем предыдущее упражнение), начинаем с подключения библиотеки. Подробнее об этом – см. Упражнение 1.

Код:

```
#include <CBioInfCpp.h>
// Including CBioInfCpp
// Подключаем библиотеку CBioInfCpp

using namespace std;

int main()
{

    return 0;
}
```

Считываем из файлов невзвешенные графы: A (тот, что побольше – файл **InA.txt**) и B1 (шаблон для поиска – файл **InB1.txt**). А граф B2 сделаем из B1 прибавив к номеру каждой вершины 2, и потом убедимся, что графы B1 и B2 изоморфны. Переименование вершин в графе «сдвигом» на определенное число производится с помощью функции **RenumVGraph**.

Для этого нам сначала надо скачать файлы с графами и разместить их в папке проекта (или в произвольном месте, но с указанием полного пути к ним для **ifstream**).

Считывание невзвешенного графа, как мы помним, производится с помощью функции **UWGraphRead**. Вывод на экран графа производится с помощью функции **GraphCout**, а в файл – **GraphFout**.

```
#include <CBioInfCpp.h>
// Including CBioInfCpp
// Подключаем библиотеку CBioInfCpp

using namespace std;

int main()
{
    vector<int> A; // Container for unweighted graph A ("more large one")
                  // is here (edge list format)
    vector<int> B1; // Container for unweighted graph B1 is here (edge
                  // list format)
    vector<int> B2; // Container for unweighted graph B2 is here (edge
                  // list format)
```

```

    ifstream fin3_A ("InA.txt"); // Unweighted graph A is here (edge list
    format)
    ifstream fin3_B ("InB1.txt"); // Unweighted graph B1 is here (edge list
    format)
    ofstream fout3 ("Out3(Graphs2).txt"); // File to write results for the
    Exercise 3.
    UWGraphRead(fin3_A, A); // Reading unweighted graph A from a file
    UWGraphRead(fin3_B, B1); // Reading unweighted graph B1 from a file
    B2=B1; // Let's graph B2=B1
    RenumVGraph(B2,2, false); // Rename all vertices of B2: let's add 2 to
    their numbers; the 3d parameter is set as false as graph is unweighted.

    cout<<"Here is unweighted graph B1: "<<endl;
    GraphCout(B1, 0); //Out graph to screen, as it is unweighted the second
    parameter should be "false" i.e. 0
    cout<<"Here is unweighted graph B2: "<<endl;
    GraphCout(B2, 0); //Out graph to screen, as it is unweighted the second
    parameter should be "false" i.e. 0

    fout3<<"Here is unweighted graph B1: "<<endl;
    GraphFout(B1, 0, fout3); //Out graph to file, as it is unweighted the
    second parameter should be "false" i.e. 0
    fout3<<"Here is unweighted graph B2: "<<endl;
    GraphFout(B2, 0, fout3); //Out graph to file, as it is unweighted the
    second parameter should be "false" i.e. 0

    return 0;
}

```

Теперь – самое время проверить изоморфность B1 и B2. Проверим для двух случаев – считая указанные графы ориентированными (`bool directed = true`) и нет (`directed = false`); `directed` будет параметром для вызова функции **SubGraphsInscribed**.

Запись же найденных изоморфных образцу подграфов (а при проверке изоморфизма он будет один – это сам граф) будем производить в `set<vector<int>>` `SubGraphs`. Также будем помнить, что **SubGraphsInscribed** возвращает количество найденных изоморфных шаблону подграфов (для ускорения можно задать предельное число найденных подграфов, после чего поиск прекращается), а в случае некорректных исходных данных вернет -1 и пустой `set<vector<int>>` `SubGraphs`.

Итак, код:

```

#include <CBioInfCpp.h>
// Including CBioInfCpp
// Подключаем библиотеку CBioInfCpp

using namespace std;

int main()
{
    vector<int> A; // Container for unweighted graph A ("more large one")
    is here (edge list format)
    vector<int> B1; // Container for unweighted graph B1 is here (edge
    list format)
    vector<int> B2; // Container for unweighted graph B2 is here (edge
    list format)
    ifstream fin3_A ("InA.txt"); // Unweighted graph A is here (edge list
    format)
    ifstream fin3_B ("InB1.txt"); // Unweighted graph B1 is here (edge list
    format)
    ofstream fout3 ("Out3(Graphs2).txt"); // File to write results for the
    Exercise 3.
    UWGraphRead(fin3_A, A); // Reading unweighted graph A from a file
    UWGraphRead(fin3_B, B1); // Reading unweighted graph B1 from a file

```

```

B2=B1; // Let's graph B2=B1
RenumVGraph(B2,2, false); // Rename all vertices of B2: let's add 2 to
their numbers; the 3d parameter is set as false as graph is unweighted.

cout<<"Here is unweighted graph B1: "<<endl;
GraphCout(B1, 0); //Out graph to screen, as it is unweighted the second
parameter should be "false" i.e. 0
cout<<"Here is unweighted graph B2: "<<endl;
GraphCout(B2, 0); //Out graph to screen, as it is unweighted the second
parameter should be "false" i.e. 0

fout3<<"Here is unweighted graph B1: "<<endl;
GraphFout(B1, 0, fout3); //Out graph to file, as it is unweighted the
second parameter should be "false" i.e. 0
fout3<<"Here is unweighted graph B2: "<<endl;
GraphFout(B2, 0, fout3); //Out graph to file, as it is unweighted the
second parameter should be "false" i.e. 0

set<vector<int>> SubGraphs; // Here will be subgraphs found

bool directed = true; //lets consider both graphs as directed ones
cout<<SubGraphsInscribed (B1, B2, SubGraphs, directed, 0, 1, 0)<<endl;
//trying to find in B1 all subgraphs that are isomorphic to B2
fout3<<SubGraphsInscribed (B1, B2, SubGraphs, directed, 0, 1, 0)<<endl;
// it should be 1 as they are isomorphic themselves

for (auto it = SubGraphs.begin(); it!= SubGraphs.end(); it++) // now
let's out all found (sub)graphs:
{
    cout<<"isomorphic (sub)graph found (directed graph case):"<<"
    "<<(*it).size()/2<<" edges"<<endl; // to screen
    GraphCout(*it,false);

    fout3<<"isomorphic (sub)graph found: (directed graph case)"<<"
    "<<(*it).size()/2<<" edges"<<endl; // to file
    GraphFout(*it,false, fout3);
}

directed = 0; //lets consider both graphs as undirected ones
cout<<SubGraphsInscribed (B1, B2, SubGraphs, directed, 0, 1, 0)<<endl;
fout3<<SubGraphsInscribed (B1, B2, SubGraphs, directed, 0, 1, 0)<<endl;

for (auto it = SubGraphs.begin(); it!= SubGraphs.end(); it++)
{
    cout<<"isomorphic (sub)graph found (undirected graph case):"<<"
    "<<(*it).size()/2<<" edges"<<endl;
    GraphCout(*it,false);

    fout3<<"isomorphic (sub)graph found (directed graph case):"<<"
    "<<(*it).size()/2<<" edges"<<endl;
    GraphFout(*it,false, fout3);
}

return 0;
}

```

Ну и осталось немного. Найдем же в графе A все подграфы, изоморфные графу B1 – опять же для случаев как ориентированных, так и неориентированных графов. Подсказка: количество найденных подграфов будет разное, а именно: 8 и 24.

Код по всему Упражнению 3:

```
#include <CBioInfCpp.h>
```

```

// Including CBioInfCpp
// Подключаем библиотеку CBioInfCpp

using namespace std;

int main()
{
    vector<int> A; // Container for unweighted graph A ("more large one")
    is here (edge list format)
    vector<int> B1; // Container for unweighted graph B1 is here (edge
    list format)
    vector<int> B2; // Container for unweighted graph B2 is here (edge
    list format)
    ifstream fin3_A ("InA.txt"); // Unweighted graph A is here (edge list
    format)
    ifstream fin3_B ("InB1.txt"); // Unweighted graph B1 is here (edge list
    format)
    ofstream fout3 ("Out3(Graphs2).txt"); // File to write results for the
    Exercise 3.
    UWGraphRead(fin3_A, A); // Reading unweighted graph A from a file
    UWGraphRead(fin3_B, B1); // Reading unweighted graph B1 from a file
    B2=B1; // Let's graph B2=B1
    RenumVGraph(B2,2, false); // Rename all vertices of B2: let's add 2 to
    their numbers; the 3d parameter is set as false as graph is unweighted.

    cout<<"Here is unweighted graph B1: "<<endl;
    GraphCout(B1, 0); //Out graph to screen, as it is unweighted the second
    parameter should be "false" i.e. 0
    cout<<"Here is unweighted graph B2: "<<endl;
    GraphCout(B2, 0); //Out graph to screen, as it is unweighted the second
    parameter should be "false" i.e. 0

    fout3<<"Here is unweighted graph B1: "<<endl;
    GraphFout(B1, 0, fout3); //Out graph to file, as it is unweighted the
    second parameter should be "false" i.e. 0
    fout3<<"Here is unweighted graph B2: "<<endl;
    GraphFout(B2, 0, fout3); //Out graph to file, as it is unweighted the
    second parameter should be "false" i.e. 0

    set<vector<int>> SubGraphs; // Here will be subgraphs found

    bool directed = true; //lets consider both graphs as directed ones
    cout<<SubGraphsInscribed (B1, B2, SubGraphs, directed, 0, 1, 0)<<endl;
    //trying to find in B1 all subgraphs that are isomorphic to B2
    fout3<<SubGraphsInscribed (B1, B2, SubGraphs, directed, 0, 1, 0)<<endl;
    // it should be 1 as they are isomorphic themselves

    for (auto it = SubGraphs.begin(); it!= SubGraphs.end(); it++) // now
    let's out all found (sub)graphs:
    {
        cout<<"isomorphic (sub)graph found (directed graph case):"<<"
        "<<(*it).size()/2<<" edges"<<endl; // to screen
        GraphCout(*it, false);

        fout3<<"isomorphic (sub)graph found: (directed graph case)"<<"
        "<<(*it).size()/2<<" edges"<<endl; // to file
        GraphFout(*it, false, fout3);
    }

    directed = 0; //lets consider both graphs as undirected ones
    cout<<SubGraphsInscribed (B1, B2, SubGraphs, directed, 0, 1, 0)<<endl;
    fout3<<SubGraphsInscribed (B1, B2, SubGraphs, directed, 0, 1, 0)<<endl;

    for (auto it = SubGraphs.begin(); it!= SubGraphs.end(); it++)

```

```

{
    cout<<"isomorphic (sub)graph found (undirected graph case):"<<"
    "<<(*it).size()/2<<" edges"<<endl;
    GraphCout(*it,false);

    fout3<<"isomorphic (sub)graph found (directed graph case):"<<"
    "<<(*it).size()/2<<" edges"<<endl;
    GraphFout(*it,false, fout3);
}

//now let's find in A all subgraphs that are isomorphic to graph B1

//first of all let's out them
cout<<"Here is unweighted graph A: "<<endl;
GraphCout(A, 0); //Out graph to screen, as it is unweighted the second
parameter should be "false" i.e. 0
cout<<"Here is unweighted graph B1: "<<endl;
GraphCout(B1, 0); //Out graph to screen, as it is unweighted the second
parameter should be "false" i.e. 0

fout3<<"Here is unweighted graph A: "<<endl;
GraphFout(A, 0, fout3); //Out graph to file, as it is unweighted the
second parameter should be "false" i.e. 0
fout3<<"Here is unweighted graph B1: "<<endl;
GraphFout(B1, 0, fout3); //Out graph to file, as it is unweighted the
second parameter should be "false" i.e. 0

// directed graph case

directed = true; //lets consider both graphs (A and B1) as directed
ones
cout<<SubGraphsInscribed (A, B1, SubGraphs, directed, 0, 1, 0)<<"
subraphs of A that are isomorphic to B1 is found:"<<endl; //trying to
find in A all subgraphs that are isomorphic to B1
fout3<<SubGraphsInscribed (A, B1, SubGraphs, directed, 0, 1, 0)<<"
subraphs of A that are isomorphic to B1 is found:"<<endl;

for (auto it = SubGraphs.begin(); it!= SubGraphs.end(); it++) // now
let's out all found (sub)graphs:
{
    cout<<"isomorphic (sub)graph found (directed graph case):"<<"
    "<<(*it).size()/2<<" edges"<<endl; // to screen
    GraphCout(*it,false); //Out graph to screen, as it is unweighted
the second parameter should be "false"

    fout3<<"isomorphic (sub)graph found: (directed graph case)"<<"
    "<<(*it).size()/2<<" edges"<<endl; // to file
    GraphFout(*it,false, fout3); //Out graph to file, as it is
unweighted the second parameter should be "false"
}

cout<<SubGraphs.size()<<" subraphs of A that are isomorphic to B1 is
found"<<endl;
fout3<<SubGraphs.size()<<"subraphs of A that are isomorphic to B1 is
found"<<endl;

// undirected graph case
directed = 0; //lets consider both graphs (A and B1) as undirected ones
cout<<SubGraphsInscribed (A, B1, SubGraphs, directed, 0, 1, 0)<<"
subraphs of A that are isomorphic to B1 is found:"<<endl; //trying to
find in A all subgraphs that are isomorphic to B1
fout3<<SubGraphsInscribed (A, B1, SubGraphs, directed, 0, 1, 0)<<"
subraphs of A that are isomorphic to B1 is found:"<<endl;

```

```

for (auto it = SubGraphs.begin(); it!= SubGraphs.end(); it++)
{
    cout<<"isomorphic (sub)graph found (undirected graph case):"<<"
    "<<(*it).size()/2<<" edges"<<endl;
    GraphCout(*it,false); //Out graph to screen, as it is unweighted
    the second parameter should be "false"

    fout3<<"isomorphic (sub)graph found (directed graph case):"<<"
    "<<(*it).size()/2<<" edges"<<endl;
    GraphFout(*it,false, fout3); //Out graph to file, as it is
    unweighted the second parameter should be "false"
}
cout<<SubGraphs.size()<<" subraphs of A that are isomorphic to B1 is
found"<<endl;
fout3<<SubGraphs.size()<<"subraphs of A that are isomorphic to B1 is
found"<<endl;

return 0;
}

```

Результат представлен и в файле Out3(Graphs2).txt.