## Общие соображения

При разработке тестов нужно проверить следующее:

1. Зависания
2. Правильная реакция на неправильные входные данные
3. Правильность (соответствие требованиям) полученного результата

Важное (почти обязательное) условие: тестирующие алгоритмы должны быть заметно проще тестируемого.

Также в некоторых случаях нужно проверять боле сложные предикаты:

1. Контрольные счетчики (например: «обращение к объектам не более N раз» , или «из входного потока прочитаны все данные» или «число итераций (прохождения конкретного участка) не более М раз» … и пр.)
2. … или контрольные выражения (например, «работа с файлом «А» идет только тогда, когда объект «В» находится в несигнальном состоянии», «объектов типа «с» всегда в два раза меньше, нежели объектов типа «е»), и пр.)

Для данного задания проверки 4 и 5 не нужны (точнее сказать – проверкой типа «4» можно считать проверку того, ЧитательДанных вычитывает все данные из входного потока, но для данного случая эту проверку можно отнести к п.3 – поскольку проверка для ЧитателяДанных правильности полученного результата проверяет и факт вычитывания всех входных данных).

П.1 (зависания) проверяются «лично» - обычно самим фактом «висения» излишне продолжительное время.

## За все отвечает заказчик

Что касается п.2 – реакции на неправильные данные – то при разработке я придерживаюсь принципа «за все отвечает заказчик (вызывальщик)» - поэтому корректность входных данных на неправильные форматы и «все такое прочее» я не проверяю. Это, к примеру, означает, что я не буду проверять тот факт, что в передаваемом массиве строк все они одной длинны. Или что передаваемом вектор-векторе, в котором содержится граф, каждому ребру «а->b», есть соответствующее ребро «b->a».

Этот принцип не абсолютный – иногда я добавляют проверки корректности входных данных. Но это обычно я делаю для некоторого набора стандартных ситуаций – например, в случае отложенных вычислений (например, когда забираю данные из кэша), либо в особо сложных случаях, либо, когда «чувствую, что нужно подстелить соломку». Иногда – в достаточно сложных системах – делаю «firewall-level» - набор объектов, где проводятся тотальные проверки правильности данных. Но в любом случае такие пере-проверки – это исключение из правила и должны занимать небольшую часть кода.

Есть одно стандартное исключение – первый объект в цепочке должен полностью проверить входные данные.

## Проверки для данного случая

Для данного кода я сделаю следующие проверки:

1. Корректность чтения данных
2. Правильность построенного графа
3. Полученный путь – кратчайший
4. Комплексная проверка: сформировать набор строк, получить «путь» между исходной и целевой строками.

## Проверка пункта 1

(корректность чтения данных) содержит три проверки (первые две – проверка реакции на входные данные):

### Проверка 1.1

проверка неправильных входных данных:

* + 1. Файла целей нет
    2. Файл целей пустой
    3. Файл целей с одной строкой
    4. Файл целей имеет две строки разной длины

### Проверка 1.2

проверка опережающего получения ответа

Файл с произвольным числом строк, но такой, что 1ая и 2ая строки отличаются только на 1 символ – ситуация вылавливается по good\_exception.

### Проверка 1.3

проверка штатной работы на нормальном наборе данных (не подпадающем под 1.1 или 1.2).

Формируется (и запоминается) исходный набор строк одной длины, гарантированно различающихся от друга.

На основании исходного набора формируется новый набор, в котором:

* Добавляются строки другой длины (в том числе нулевой длины)
* Добавляются дубликаты строк, входящих в исходный набор.

Полученный набор строк подается на вход «ЧитателяДанных»

Проверяется факт совпадения набора строк – ИсходногоНабора и набора, полученного на выходе ЧитателяДанных

NB 1.3.а – в исходном наборе нужно проверить две первые строки – они должны быть одинаковой длины и отличаться более чем на одну букву.

NB 1.3.б – проверяемые наборы строк перед сравнением надо отсортировать – чтобы убрать в зависимость от способа построения – когда одна и та же строка может оказаться в разных позициях двух наборов.

## Проверка пункта №2

проверяется правильность построения Графа (точнее вектор-вектора, в котором он закодирован) по ВходномуНабору строк:

2.1) проверить, что в вектор-векторе (в котором закодировано представление графа) для любого ребра любого ребра a->b существует b-> a

2.2) проверить, что слова, связанные связкой a->b отличаются ровно на одну букву

2.3 проверить, что если слова из ВходногоНабора отличаются на одну букву, то они занесены в граф (соответствующая «запись» есть в вектор-векторе).

не отменяет, однако, «глубинных логических проверок» производимых в процессе обработки данных

## Проверка пункта №3

проверяется, что полученный путь – кратчайший

Вычисляется путь с помощью алгоритма.

Берется длина на единицу меньше.

Работаем по алгоритму прямого комбинаторного перебора всех возможных путей на предмет поиска пути искомой длины (на 1 меньше найденного алгоритмом).

## Генераторы входных данных для проверок №1, №2 и №3.

### Проверка 1:

Для 1.1. – набор формируется вручную (фиксированный набор файлов: 1.1\*.txt)

Для 1.2 – набор файлов формируется вручную (фиксированный набор файлов:1.2\*.txt )

Для 1.3 – генератор строк + небольшая программа

## Проверки 2 и 3:

Генератор данных + тест-программа