**tests.rkt**

**Вспомогательные функции:**

1. (gen-vertex-name n) – генерирует имя вершины вида “vn”
2. (weight) – генерирует вес для ребер
3. (inc x) – прибавляет единицу к x
4. (push lst elem) – добавляет элемент elem в список lst, если он в него не входит
5. (all-vertex graph) – итеративный поиск всех вершин в графе, возвращает список из вершин
6. (vertex-neighbors vertex graph) – возвращает список вершин-соседей для вершины vertex в графе graph с их весами вида ((v1 . weight1) … (vN . weight)). Итеративна.
7. (gen-test-edges num\_of\_vertex) – итеративно генерирует список ребер для тестов. Количество сгенерированных ребер определяется значением   
   (random (max (+ (quotient num\_of\_vertex 2) 1) 1)) для того, чтобы тесты имели решение.
8. (solver graph control\_weight edge\_list) – находит гамильтонов цикл с весом не больше control\_weight и включающий ребра из списка edge\_list. Если таких циклов несколько, то выбирает цикл с наименьшим весом.
   * (find-hamiltonian-cycle start visited neighbors weight control\_edges) –

переборный алгоритм нахождения гамильтонова цикла в графе из вершины start.

**Рассмотренные тестовые случаи:**

1. **Граф-цикл**

(gen-cycle n) – итеративно генерирует граф-цикл с более чем двумя вершинами вида ((v1 v2 weight12) (v2 v3 weight23) …   
(vN-1 vN weightN-1N) (vN v1 weightN1))

(gen-cycle-tests count) – генерирует count тестовых наборов с входом:

* граф-цикл с количеством вершин   
  (define num\_of\_vertex (+ count 2))
* весом (w (+ (\* num\_of\_vertex 5) (random (\* num\_of\_vertex 10)))) (для того, чтобы не было слишком мало тестов с ответом #f)
* пустым список проверочных ребер

Пустой список ребер связан с тем, что граф-цикл сам по себе является гамильтоновым циклом, и добавление проверочных ребер, которые генерируется случайным образом, с большой вероятностью приведут к отрицательному ответу.  
Единственным критерием существования ответа является сверка с весом w веса всего графа.

1. **Граф-звезда**

(gen-star n) – итеративно генерирует граф-звезду с более чем тремя вершинами вида ((v1 v2 weight12) (v1 v3 weight13) … (v1 vN weight1N))

(gen-star-tests count) – генерирует count тестовых наборов с входом:

* граф-звезда с количеством вершин  
  (define num\_of\_vertex (+ count 3))
* весом (random (\* num\_of\_vertex 10))
* ребрами (gen-test-edges num\_of\_vertex)

Вес и ребра никак не влияют на ответ, т.к. он заведомо отрицательный из-за структуры данного вида графов.

1. **Полный граф**(gen-full n) – итеративно генерирует полный граф с одной и больше вершинами вида

((v1 v2 weight12) … (v1 vN weight1N) (v2 vN weight2N) … (vN-1 vN weightN-1N))

(gen-full-tests count) – генерирует 5 \* count тестовых наборов с входом:

* полный граф с количеством вершин count
* весом (weight (+ (\* count 5) (random (\* count 10))))
* ребрами (edges (gen-test-edges count))

Далее с помощью функции (solver graph control\_weight edge\_list) ищется ответ и записывается в выходной файл. В полном графе заведомо есть гамильтонов цикл, влияние на ответ оказывают лишь параметры control\_weight и edge\_list.

1. **Граф со случайными ребрами**

(gen-random v\_count) – итеративно генерирует случайный граф с v\_count вершинами. Вызывается функция (helper e\_c v\_c result), которая генерирует максимум e\_c ребер для v\_c вершин. Изначально e\_c равняется (/ (\* (- v\_count 1) v\_count) 2), т.е. количеству ребер в полном графе с v\_count вершинами. Но т.к. ребра генерируются случайно, и в список result не добавляются ребра с одинаковыми вершинами и те, которые уже есть в списке, с помощью функции (push elem list) (аналог другого push), то итоговый граф будет не полным с очень большой вероятностью.  
(gen-random-tests count) – полная аналогия функции (gen-full-tests count) за исключением вида графа.

Все тесты помещаются в папку tests в текущей директории с именами, заканчивающимися на .in для входа и .out для выхода теста.

Функция (gen-tests) вызывает поочередно генерацию тестов для каждого вида графов и записывает время генерации в файл testsLog.txt в текущей директории.

Не стал соединять графы из вышерассмотренных между собой без пересечения вершин для получения графов больших размеров, т.к. для несвязных графов ответ заведомо отрицательный, а такой случай уже рассмотрен в виде графов-звезд.

**gentests.py**

Скрипт запуска тестов

1. Удаляет папку tests в текущей директории, если она существует.
2. Создает папку tests в текущей директории.
3. Запускает генерацию тестов.

**genetics.rkt**

(solver graph control\_weight edge\_list) – solver из tests.rkt, вместо генетического алгоритма.

(cmd-parser) – парсер командной строки. На вход подается название файла, из которого надо считать входные параметры для функции solver.

Ответ выводится в стандартный поток вывода.

**check.py**

Скрипт проверки работы генетического алгоритма

1. Удаляет папку results в текущей директории, если она существует.
2. Создает папку results в текущей директории.
3. Для каждого файла \*.in из папки tests запускает genetics.rkt, подавая на вход этот файл и записывает результат в файл с таким же названием, но с расширением .out. Файлы с результатами находятся в папке results.
4. Для каждого файла \*.out из папки results производится сверка результата генетического алгоритма с выходом теста, т.е. сравниваются два файла:

/results/file.out и /tests/file.out. Проверяется соответствие только двух первых строк. В случае отрицательного ответа строка #f и пустая строка, иначе строка #t и строка с весом. Путь не проверяется, т.к. с одним и тем же весом могут быть несколько гамильтоновых циклов (например, в графах-циклах). Результаты и статистика записываются в файл log.html в текущей директории.