МОСКОВСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ имени М.В.ЛОМОНОСОВА  
ФАКУЛЬТЕТ ВЫЧИСЛИТЕЛЬНОЙ МАТЕМАТИКИ И КИБЕРНЕТИКИ  
КАФЕДРА СИСТЕМНОГО ПРОГРАММИРОВАНИЯ

Отчёт по второму заданию

Вариант 3Г.Расскраска вершин графа

Чечин Иван Алексеевич

428 группа

Москва 2018

# Содержание

[1. Постановка задачи 3](#__RefHeading___Toc710_286103203)

[2. Генерация тестов 4](#__RefHeading___Toc1212_366253523)

[3. Генетический алгоритм 5](#__RefHeading___Toc1214_366253523)

[4. Результаты 7](#__RefHeading___Toc1222_366253523)

[Заключение 9](#__RefHeading___Toc743_1226708816)

[Приложение. Код программы 10](#__RefHeading___Toc1228_366253523)

# 1. Постановка задачи

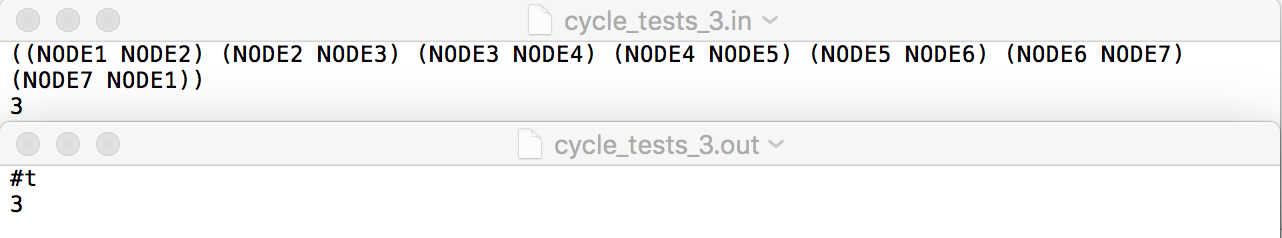
На вход программы подаётся список рёбер неориентированного графа G = (V, E). Вершины vi∈V обозначаются атомами, рёбра представляются списками вида (NODE1 NODE2). Затем на вход программы подаётся целое число K (K ≥ 1). Требуется написать программу на языке Scheme, которая решает задачу поиска правильной раскраски графа в минимально возможное количество цветов, методом генетического программирования. Раскраска является правильной, если нет смежных вершин одинакового цвета.

В качестве ответа нужно вывести #f, если раскраска не существует, иначе #t, минимально возможное число цветов, и раскраску с таким количеством цветов в виде списка, каждый элемент которого – список из ребра и его цвета.

# 2. Генерация тестов

Генератор тестов написан на Scheme и находится в файле tests.rkt. Тестируются случаи: полный граф, путь, цикл, звезда. Для всех рассмотрех случае ответ заранее известен и может быть указан как единственно верный. Входные данные и правильный ответ помещаются в директорию /Users/ivan/Desktop/fp/ в виде двух файлов, поименованных в соответствие с видом графа

Пример:



Правильную раскраска в ответе не указывается, т.к. вариантов правильной раскраски может быть несколько

Файл gen.py проверяет наличие директорий /Users/ivan/Desktop/fp/ и /Users/ivan/Desktop/res/ , если они уже существуют(т. е. в них есть тесты), то директории удаляются и создаются заново(уже пустые). Запускается генерация тестов и вход, сгенерированный ими, подается на вход генетике gen\_in\_progress.rkt. Выход генетики записывается в файлы в директорию /res, где названия файлов \*.out совпадают с названиями аналогичных файлов из директории /fp.

Затем в директории /res заводится файл aaa\_res в который записывается результат сравнения файлов из /res и /fp (то есть выход генетики и правильный ответ полученный при генерации тестов)

Существует 3 реакции на данный генетикой ответ :

1. Тест пройден
2. Неправильно определенное количество цветов
3. Неверный определена возможность расскраски

Пример

test star\_tests\_3.out wrong paint

test path\_tests\_3.out wrong answer

test path\_tests\_3.out passed

При тестировании реализованной генетики иногда выявлялись неправильные ответы (примерно 1-2 из 50-70 тестов). Ошибки эти в основном проявлялись при раскраске длинного графа вида путь или цикл в 2 цвета.

# 3. Генетический алгоритм

Генетические алгоритмы — это адаптивные методы поиска, которые в последнее время используются для решения задач оптимизации. В них используются как аналог механизма генетического наследования, так и аналог естественного отбора. При этом сохраняется биологическая терминология в упрощенном виде и основные понятия линейной алгебры.

Образцом популяции в данном случае является вектор номеров раскрасок вершин, он соответствует вектору вершин, построенному из входных данных.

Начальная популяция создаётся случайно, цвет каждой вершины представителя популяции выбирается случайно и независимо.

Целевой функцией будет являться функция, считающая количество конфликтов, т.е. смежных вершин одного цвета, в данном варианте раскраски. Подсчет происходит проходом всех ребер нашего графа, и сравнения цветов концов этих ребер.

Количество цветов для раскраски есть параметр генетики

Задача генетического алгоритма – минимизировать целевую функцию и в итоге привести ее значение к 0 на одной из хромосом. Данная хромосома и будет являться верной раскраской. Если за определенное количество итераций не удается улучшить функцию, то считается, что для заданного количества цветов построить раскраску невозможно, если же функция для данного поколения меньше предыдущего минимума, то счетчик итераций сбрасывается.

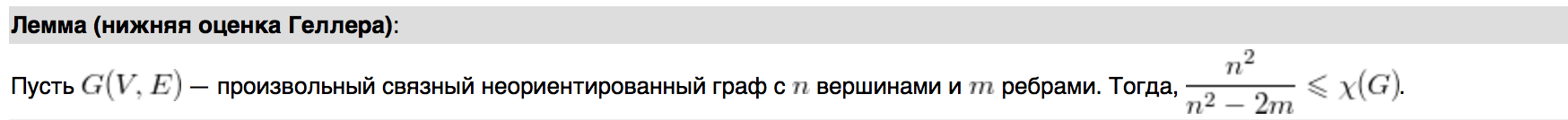
Хромосомы в каждом поколении сортируются по возрастанию оценочной функции. Для возможности перехода в следующее поколение в неизмененном виде или с помощью скрещивания, образец должен входить в квоту лучших представителей своего поколения

Каждый образец нового поколения является либо неизмененным представителем предыдущего (из квоты лучших, как уже было указано) или результатом скрещивания (для скрещивания также берутся лучшие представители). При скрещивании двух особей произвольным образом выбирается точка в векторе, результат скрещивания – левая часть от выбранной точки первого предка + правая часть второго. При этом 1 лучший образец точно попадает в следующую популяцию.

Каждая особь нового поколения может подвергнуться мутации с некоторой вероятностью, мутация подразумевает изменения цвета случайной вершины.

Изначально генетика запускается с данным количеством цветов, если только он не превышает количество вершин в данном графе, во втором случае параметром генетики становится количество вершин. При положительном ответе алгоритм генетики запускается заново с уменьшенным на 1 количеством цветов, и так до момента, когда генетика вернет #f. В таком случае ответом будет являться ответ предыдущей итерации генетики (если она была, конечно)

Изначально поиск по цветам был линейным, т.е. количество цветов уменьшалось на единицу с каждым новым запуском генетики. Затем был испробован бинарный алгоритм поиска. Нижним ограничителем диапазона для данного алгоритма была взята нижняя оценка Геллера хроматического числа



Прирост в скорости составил примерно 25% на небольших тестах(под небольшими в данном случае подразумеваются тесты, где входное количество цветов недалеко от истинного значения, и линейному алгоритму нужно пройти 2-50 шагов) и более чем в два раза на крупных (пример крупного – путь с изначально заданным 1000 цветом, для ответа линейному алгоритму придется запустить генетику 998 раз). Очевидно, что бинарный поиск куда более лучше справляется с данной задачей, в итоговой реализации остается он.

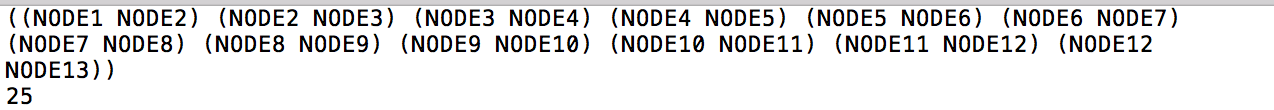
# 5. Результаты

В результате на сгенерированных тестах доля ошибки примерно 0-3%. В основном ошибки возникают на двухцветных раскрасках больших путей или циклов. Долю ошибок можно уменьшить, запустив алгоритм с большим дозволенным числом итераций и большей популяцией, но тогда увеличится время выполнения, и наоборот, понизив число итераций мы ускорим работу алгоритма. Большие тесты значительно замедляют работу алгоритма, однако он все еще лучше переборного.

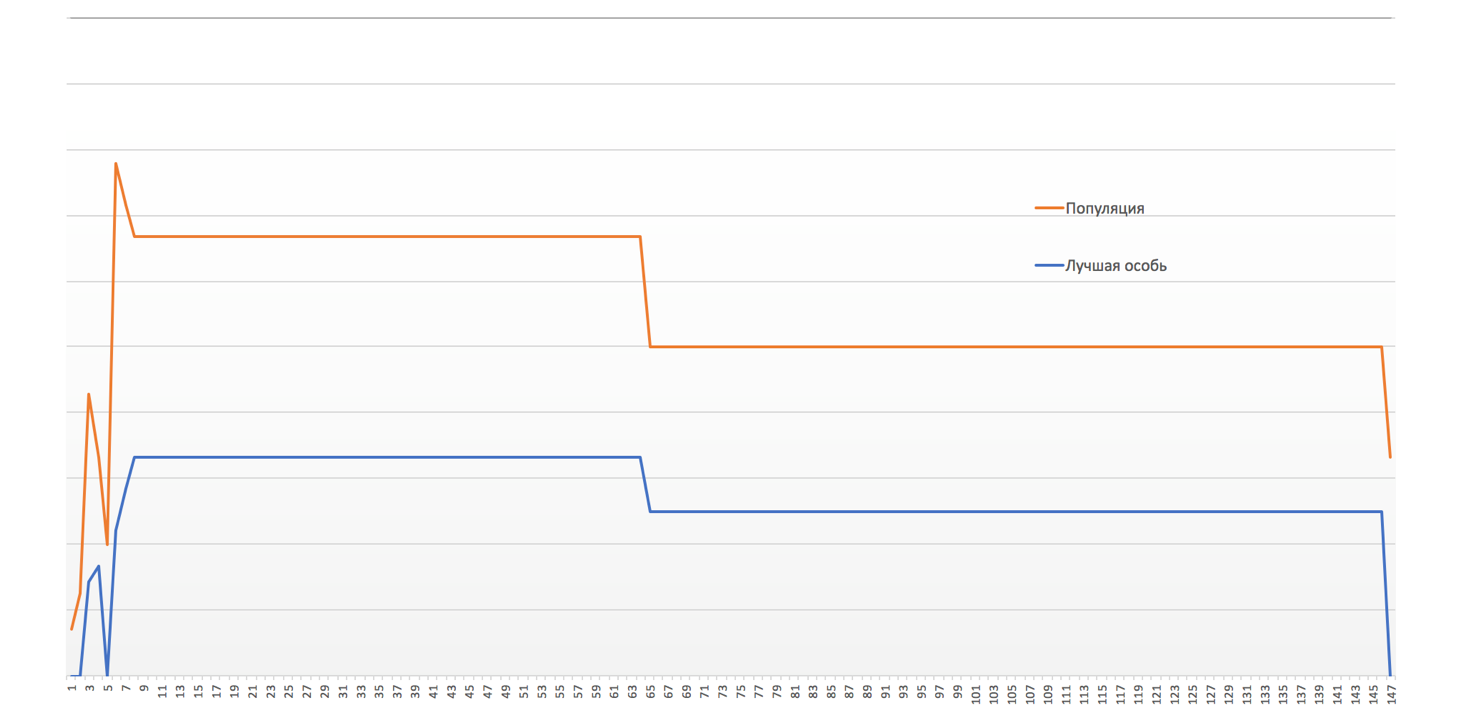
При решении удалось значительно ускорить работу путем замены линейного поиска минимальной раскраски на бинарный.

Изначально задается заведомо большая популяция, делается это для того, чтобы как можно прийти к минимум целевой функции. Зачастую, уже на первых этапах пападаются хромосомы с количеством конфликтов 1-3, и дальнейшая, самая долгая и часть алгоритма заключается в том, чтобы из большого количества хромосом с 1 конфликтом создать новую истинную.

Для примера разберем тест путь графа



Будут проработаны случаи с 13,7,4,2 цветами



Как мы видим, правильные раскраски для 13 и 7 цветов находятся сразу же в нашей популяции, для раскраски в 4 цвета требуется три поколения. Затем начинается самая тяжеловесная часть алгоритма, раскраска графа в 2 цвета, количество конфликтов сначала долгое время не может стать лучше 2, затем 1, затем алгоритм получает нужную хромосому с 0 конфликтов.  
Так как в следующее поколение попадает только определенная часть лучших особей, то графики средней приспособленности популяции и лучшей хромосомы быстро становятся идентичными( при раскраске в малое количество цветов почти все хромосомы будут иметь на поздних этапах по 1 конфликту )

# Заключение

Раскраска графа при помощи генетического алгоритма возможна. На больших графах генетический алгоритм может решает задачу раскраски гораздо быстрее простого переборного. На особо сложных случаях может возникать ошибка, ее вероятность может быть уменьшена с помощью увеличения итераций алгоритма, но в этом случае сильно повысится время выполнения. 50 тестов разного рода этот алгоритм решал за 12 секунд, что кажется в целом приемлемым, значительного прогресса удалось добиться с помощью перехода с пошагового поиска минимально цвета на бинарный. Как итог, генетический алгоритм пригоден для решения задачи о раскраске графа.

# Приложение. Код программы

Код генетики

#lang scheme/base

(require scheme/list)

(require scheme/math)

(require compiler/find-exe)

;(println #f)

(define (cmd)

;(call-with-input-file (car (vector->list (current-command-line-arguments)));чтение входа из командной строки

(call-with-input-file "/Users/ivan/Desktop/fp/path\_tests\_6.in"

(lambda (in)

(list (read in) (read in)))))

(define edges (list->vector (caar (list (cmd))))); граф

(define max-colors (cadar (list (cmd)))); количество цветов

;(println edges)

;(println max-colors)

;(define edges (list->vector (read)))

;(define max-colors (read))

(define len (vector-length edges))

(define (gather-points i ht)

(cond ((< i 0) ht)

(else (let ((edge (vector-ref edges i)))

(gather-points (- i 1) (hash-set\* ht (car edge) '() (cadr edge) '()))))))

(define points (list->vector (sort (hash-keys (gather-points (sub1 len) (make-immutable-hash)))

string<? #:key symbol->string)))

(define n\_points (vector-length points));количество вершин

(define max\_generations 300)

(define population\_size 100)

(define mutation\_probability 0.6)

(define mutation\_severity 0.05)

(define cross\_probability 0.4)

(define crossing\_quota (exact-round (\* 0.4 population\_size)))

(define selection\_quota (exact-round (\* 0.4 population\_size)))

(define keep\_quota (max 2 (exact-round (\* 0.1 population\_size))))

(define (random-specimen colors)

(build-vector n\_points (lambda (i) (+ (random colors) 1))))

(define (random-population colors)

(build-vector population\_size (lambda (i) (random-specimen colors))))

(define (get-score specimen);функция оценки - количетсво конфликтов для входной раскраски

(define (get\_s spec i table ans)

(cond ((< i 0) ans)

(else (let\* ((first (hash-ref table (car (vector-ref edges i)) 0))

(second (hash-ref table (cadr (vector-ref edges i)) 0)))

(cond ((equal? first second) (get\_s spec (sub1 i) table (add1 ans)))

(else (get\_s spec (sub1 i) table ans)))))))

(get\_s specimen (sub1 len) (make-immutable-hash (map (lambda (x y) (cons x y)) (vector->list points) (vector->list specimen))) 0))

(define (build-scores population)

(build-vector (vector-length population)

(lambda (i) (get-score (vector-ref population i)))))

(define (cross-specimen a b)

(let ((cross\_point (random n\_points)))

(build-vector n\_points (lambda (i)

(if (< i cross\_point)

(vector-ref b i)

(vector-ref a i))))))

(define (mutate-specimen colors specimen)

(build-vector n\_points (lambda (i)

(cond ((< (random) mutation\_severity)

(+ 1 (random colors)))

(else (vector-ref specimen i))))))

(define (find-best scores)

(define (find-best\_ scores i j)

(cond ((<= i 0) j)

(else (find-best\_ scores (- i 1)

(if (< (vector-ref scores i) (vector-ref scores j)) i j)))))

(find-best\_ scores (- (vector-length scores) 1) 0))

(define (next-generation col population old\_best remaining\_generations)

(let\*((scores (build-scores population))

(best (find-best scores))

(bestscore (vector-ref scores best)))

;(print bestscore) (print " " )

;(print (quotient (foldl + 0 (vector->list scores)) (vector-length scores))) (print " ")

;(println col)

(cond((<= remaining\_generations 0)

(if (= 0 bestscore)

(vector-ref population best)

'()))

((> bestscore 0)

(let\* ((sorted

(list->vector (sort (range (vector-length population)) <

#:key (lambda (i) (vector-ref scores i)))));сортировка по оценочной функции по возрастанию(т.е. лучшие особи на первых местах)

(next (build-vector population\_size (lambda (i); новая популяция

(let((chromosome

(cond ((< i keep\_quota)

(vector-ref population (vector-ref sorted i)))

((< (random) cross\_probability);скрещивание

(cross-specimen

(vector-ref population (vector-ref sorted (random crossing\_quota)))

(vector-ref population (vector-ref sorted (random crossing\_quota)))))

(else

(vector-ref population (vector-ref sorted (random selection\_quota)))))))

(if (and (>= i keep\_quota) (< (random) mutation\_probability));мутация

(mutate-specimen col chromosome)

chromosome))))))

;(write bestscore) (write '()) (write (quotient (foldl + 0 (vector->list scores)) (vector-length scores))) (write'()) (write col)(newline) ;(write scores)(newline)

(if (< bestscore old\_best)

(next-generation col next bestscore max\_generations) ;сброс счетчика

(next-generation col next old\_best (- remaining\_generations 1)))))

(else (vector-ref population best)))));)

(define (genetic colors)

(next-generation colors (random-population colors) +inf.0 max\_generations))

(define (gen-step colors prev\_answer)

(let\*((answer (genetic colors)))

(if (null? answer)

(cons #t prev\_answer)

(gen-step (- colors 1) (cons colors answer)))))

(define geller ( quotient (\* n\_points n\_points) ( - (\* n\_points n\_points) (\* 2 len))))

(define (main)

(if (>= max-colors n\_points)

(bin-step geller n\_points (cons n\_points (genetic n\_points)))

;(gen-step n\_points (cons n\_points (genetic n\_points)))

(let ((answer (genetic max-colors)))

(if (null? answer)

'(#f)

(bin-step geller max-colors (cons max-colors answer))))))

;(gen-step (- max-colors 1) (cons max-colors answer))))))

(define (bin-step min\_colors max-colors upper\_answer)

(if (>= min\_colors max-colors)

(cons #t upper\_answer)

(let\* ((mid-colors (quotient (+ min\_colors max-colors) 2))

(answer (genetic mid-colors)))

;(write mid-colors)

(if (null? answer)

(bin-step (+ 1 mid-colors) max-colors upper\_answer)

(bin-step min\_colors mid-colors (cons mid-colors answer))))))

(cond ((equal? edges #() )

(write "Пустой граф")(newline))

((<= max-colors 0)

(write "Ошибка цветов")(newline))

(else

(let ((answer (main)))

(write (car answer))(newline)

(cond ((car answer)

(write (cadr answer))(newline)

(write (build-list n\_points (lambda (i) (list (vector-ref points i) (vector-ref (cddr answer) i)))))(newline))))))

**Генерация тестов**

#lang scheme/base

(require racket/list)

(define (NODE n)

(string->symbol (string-append "NODE" (number->string n)))); вершина NODE с номером n

(define (cycle n);3 еcли n четно, 2 если нет

(define (loop n result)

(if (= n 1)

result

(loop (- n 1) (cons (list (NODE (- n 1)) (NODE n)) result))))

(if (>= n 2)

(loop n (list (list (NODE n) (NODE 1))))

'() ))

(define (full n);n

(define (loop n result)

(if (= n 1) result

(loop (- n 1) (foldl cons result (map (lambda (x) (list (NODE (+ 1 x)) (NODE n) )) (build-list (- n 1) values))))))

(cond ((= n 1) '())

((> n 0) (loop n '()))))

(define (star n);n

(define (loop n result)

(if (= n 1) result

(loop (- n 1) (cons (list (NODE 1) (NODE n)) result))))

(if (= n 1) '()

(loop n '())))

(define (path n);2

(define (loop n result)

(if (= n 1)

result

(loop (- n 1) (cons (list (NODE (- n 1)) (NODE n)) result))))

(if (>= n 2)

(loop n '())

'() ))

(define (demo\_gen lst num)

'(#f))

(define (cycle\_tests number)

(cond (( = number 0)

(println 'over))

(else (let\* ((v (random 3 20))

(p (random 1 14))

(res (cycle v)))

(cond ((and (> p 2) (= 0 (modulo v 2)))

(with-output-to-file (string-append "/Users/ivan/Desktop/fp/cycle\_tests\_" (number->string number) ".in") #:exists 'append (lambda() (println res) (println p )))

(with-output-to-file (string-append "/Users/ivan/Desktop/fp/cycle\_tests\_" (number->string number) ".out") #:exists 'append (lambda() (println #t) (println 2))))

((and (> p 2) (= 1 (modulo v 2)))

(with-output-to-file (string-append "/Users/ivan/Desktop/fp/cycle\_tests\_" (number->string number) ".in") #:exists 'append (lambda() (println res) (println p ) ))

(with-output-to-file (string-append "/Users/ivan/Desktop/fp/cycle\_tests\_" (number->string number) ".out") #:exists 'append (lambda() (println #t) (println 3)))) ; (if (equal? (car res) #t)

((and (= p 2) (= 0 (modulo v 2)))

(with-output-to-file (string-append "/Users/ivan/Desktop/fp/cycle\_tests\_" (number->string number) ".in") #:exists 'append (lambda() (println res) (println p ) ))

(with-output-to-file (string-append "/Users/ivan/Desktop/fp/cycle\_tests\_" (number->string number) ".out") #:exists 'append (lambda() (println #t) (println 2))))

((and (= p 2) (= 1 (modulo v 2)))

(with-output-to-file (string-append "/Users/ivan/Desktop/fp/cycle\_tests\_" (number->string number) ".in") #:exists 'append (lambda() (println res) (println p ) ))

(with-output-to-file (string-append "/Users/ivan/Desktop/fp/cycle\_tests\_" (number->string number) ".out") #:exists 'append (lambda() (println #f) (println 0))))

((= p 1)

(with-output-to-file (string-append "/Users/ivan/Desktop/fp/cycle\_tests\_" (number->string number) ".in") #:exists 'append (lambda() (println res) (println p ) ))

(with-output-to-file (string-append "/Users/ivan/Desktop/fp/cycle\_tests\_" (number->string number) ".out") #:exists 'append (lambda() (println #f) (println 0))))))

(cycle\_tests (- number 1)))))

;(cycle\_tests 5)

(define (star\_tests number)

(cond (( = number 0)

(println 'over))

(else (let\* ((v (random 2 10))

(p (random 2 15))

(res (star v) ))

(cond ((> v p)

(with-output-to-file (string-append "/Users/ivan/Desktop/fp/star\_tests\_" (number->string number) ".in") #:exists 'append (lambda() (println res) (println p ) ))

(with-output-to-file (string-append "/Users/ivan/Desktop/fp/star\_tests\_" (number->string number) ".out") #:exists 'append (lambda() (println #t) (println 2))))

((<= v p)

(with-output-to-file (string-append "/Users/ivan/Desktop/fp/star\_tests\_" (number->string number) ".in") #:exists 'append (lambda() (println res) (println 2 ) ))

(with-output-to-file (string-append "/Users/ivan/Desktop/fp/star\_tests\_" (number->string number) ".out") #:exists 'append (lambda() (println #t) (println 2))))))

(star\_tests (- number 1)))))

(define (full\_tests number)

(cond (( = number 0)

(println 'over))

(else (let\* ((v (random 2 18))

(p (random 1 100))

(res (full v) ))

(cond ((> v p)

(with-output-to-file (string-append "/Users/ivan/Desktop/fp/full\_tests\_" (number->string number) ".in") #:exists 'append (lambda() (println res) (println p ) ))

(with-output-to-file (string-append "/Users/ivan/Desktop/fp/full\_tests\_" (number->string number) ".out") #:exists 'append (lambda() (println #f) (println 0))))

((<= v p)

(with-output-to-file (string-append "/Users/ivan/Desktop/fp/full\_tests\_" (number->string number) ".in") #:exists 'append (lambda() (println res) (println p ) ))

(with-output-to-file (string-append "/Users/ivan/Desktop/fp/full\_tests\_" (number->string number) ".out") #:exists 'append (lambda() (println #t) (println v))))))

(full\_tests (- number 1)))))

(define (path\_tests number)

(cond ((= number 0 )

(println 'over))

(else (let\* ((v (random 2 18))

(p (random 1 4))

(res (path v) ))

(cond ((> p 1)

(with-output-to-file (string-append "/Users/ivan/Desktop/fp/path\_tests\_" (number->string number) ".in") #:exists 'append (lambda() (println res) (println p ) ))

(with-output-to-file (string-append "/Users/ivan/Desktop/fp/path\_tests\_" (number->string number) ".out") #:exists 'append (lambda() (println #t) (println 2))))

((= p 1)

(with-output-to-file (string-append "/Users/ivan/Desktop/fp/path\_tests\_" (number->string number) ".in") #:exists 'append (lambda() (println res) (println p ) ))

(with-output-to-file (string-append "/Users/ivan/Desktop/fp/path\_tests\_" (number->string number) ".out") #:exists 'append (lambda() (println #f) (println 0))))))

(path\_tests (- number 1)))))

(define (test)

(println 'cycle\_tests)

(cycle\_tests (random 10 30))

(println 'path\_tests)

(path\_tests (random 10 30))

(println 'full\_tests)

(full\_tests (random 10 30))

(println 'star\_tests)

(star\_tests (random 5 10)))

(test)

**Запуск и проверка тестов**

import shutil  
import os  
import subprocess  
import glob  
import re  
import time  
  
start\_time = time.time()  
  
if 'fp' in os.listdir('/Users/ivan/Desktop/'):  
 shutil.rmtree('/Users/ivan/Desktop/fp')  
os.mkdir('/Users/ivan/Desktop/fp')  
  
if 'res' in os.listdir('/Users/ivan/Desktop/'):  
 shutil.rmtree('/Users/ivan/Desktop/res')  
os.mkdir('/Users/ivan/Desktop/res')  
  
testsDir = '/Users/ivan/Desktop/fp/'  
racketDir = '/Applications/Racket v6.11/bin/racket'  
resultsDir = '/Users/ivan/Desktop/res/'  
  
#print(os.listdir('/Users/ivan/Desktop/'))  
  
  
  
subprocess.call([racketDir, '/Users/ivan/Desktop/tests.rkt'])  
  
tests = glob.glob(testsDir + '\*.in')  
  
for test in tests:  
 print(test)  
 out = subprocess.Popen([racketDir, '/Users/ivan/Desktop/gen\_in\_progress.rkt', test],  
 stdin=subprocess.PIPE,  
 stdout=subprocess.PIPE,  
 stderr=subprocess.PIPE)  
 output, err = out.communicate()  
 print(err)  
 print(output)  
 #output = out.stdout  
 #print(out)  
 output = output.decode("utf-8").split("\n")  
 #print(output)  
 #for line in output:  
 # print('wot' + line)  
 with open(resultsDir + test.split('/')[-1].replace('in', 'out'), 'w') as fil:  
 for line in output:  
 print(line, file=fil)  
  
  
  
  
  
results = glob.glob(resultsDir + '\*.out')  
final = open('/Users/ivan/Desktop/res/aaa\_res','w')  
for result in results:  
 with open(testsDir + result.split('/')[-1],'r') as fil, open(result,'r') as res:  
 print(fil)  
 print(res)  
 #print(fil.readline() == res.readline())  
 a = res.readline()  
 b = fil.readline()  
 if a == b:  
 print(a)  
 print(b)  
 print('test ' + result.split('/')[-1])  
 print()  
 if a == '#f\n' or b == '#f\n':  
 final.write('test ' + result.split('/')[-1] + ' passed' + '\n')  
 else:  
 if res.readline() == fil.readline():  
 final.write('test ' + result.split('/')[-1] + ' passed' + '\n')  
 else:  
 final.write('test ' + result.split('/')[-1] + ' wrong paint'+'\n')  
 else:  
 final.write('test ' + result.split('/')[-1] + ' wrong answer'+'\n')  
  
  
print("--- %s seconds ---" % (time.time() - start\_time))  
  
  
  
#f = open('/Users/ivan/Desktop/hello.out')  
#r = open('/Users/ivan/Desktop/bye.in')  
  
def check(f,r):  
 f.readline()  
 tmp = int(f.readline())  
 graph = r.readline()  
 str = f.readline()  
 a = re.findall("\w+",str)  
 c = {a[i] : a[i+1] for i in range(0,len(a)-1,2)}# словарь с раскрасками (Node2: 1, где 1 это номер цвета)  
 b = re.sub('\("\w+"\)', '', graph)# отсекаем несвязные выршины графа  
 b = re.findall("\w+",b)# список, в котором две соседние вершины это ребро графа  
 print(b)  
 print(c)  
 if tmp != len(set(c.values())):  
 return False  
 for i in range(0,len(b) - 1,2):  
 #print(c.get(b[i]), ' ' , c.get(b[i+1]))  
 if c.get(b[i]) == c.get(b[i+1]):  
 return False  
 return True