МОСКОВСКИЙ АВИАЦИОННЫЙ ИНСТИТУТ
(НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ)

Кафедра теории вероятностей и компьютерного моделирования

Лабораторная работа № 5
По спецкурсу «Теория сложности алгоритмов»

Верификаторы

Выполнил: Покхарел П.К.
Группа: М8О-101М-22, Вариант 7
Преподаватель: Рассказова В.А.

Москва, 2023

Задание. Для заданного языка

3COLOR = {⟨G⟩: G – неориентированный граф, вершины которого могут быть окрашены в три цвета так, чтобы никакие две смежные вершины не были окрашены одинаково};

1. Построить описание верификатора с полиномиальной временной сложностью и соответствующего сертификата принадлежности;

2. Реализовать данный верификатор в виде программы;

3. Провести тестовые исследования, демонстрирующие совпадение фактической временной сложности с теоретической.

Описание Верификатора:

Алгоритм 1 (Сертификат)

Идея:

Пусть n - число вершин графа, t - список смежности графа, cert - словарь, в котором ключи (keys) - номера вершин графа от 1 до n, а соответствующие им значения (values) - цвета “r”, “g” или “b”, c_verts - список случайно сгенерированных цветов “r”, “g” или “b” длины n (цвета вершин графа)

1. Случайным образом генерируем цвета вершин
2. Случайным образом набрасываем ребра между вершинами, проверяя, что цвет смежных вершин не совпадает

Входные данные: n

Выход: n, t, cert

Алгоритм 1 подробнее:

Для каждого i из range(1, n+1):

Для каждого j из range(1, n+1):

```
// проверяем, что вершины разные
check_colors = c_verts[i-1] == c_verts[j-1]
```

// генерируем рандомное число 0 или 1, которое либо добавит смежность, либо выкинет ее
prob = random.randint(0, 1)

// проверяем, что ранее эту связку вершин не добавляли
exist = True если (j, i) in t иначе False

ЕСЛИ prob И (НЕ check_colors) И (НЕ (exist И i != j)):

new_adj = (i, j) ЕСЛИ i < j ИНАЧЕ (j, i)

// добавляем связку вершин в список смежности
t.append(new_adj)

Вернуть n, t, cert

Алгоритм 2 (Верификатор)

Для каждого ребра проверяем, что цвета смежных вершин различны.

Если хотя бы для одного ребра цвета совпадут:
Вернуть False
Иначе:
Вернуть True

Входные данные: graph = (n, t), cert

Выход: True если удалось окрасить все вершины

Алгоритм 2 подробнее:

для каждого adj из t:

vert1 = adj[0]
vert2 = adj[1]

// проверяем по словарю, что цвета данной связки вершин отличаются
ЕСЛИ cert[vert1] == cert[vert2]:

Вернуть False

Вернуть True

Программная реализация

```
# %pip install igraph

from igraph import *
import numpy as np
import random

# Напишем генератор тестов

# color_verts = [colors[random.randint(0, 2)] for _ in range(NumberOfVertices)]
def col_func_test(n):
    colors = ['r', 'g', 'b']
    color_verts = [colors[random.randint(0, 2)] for _ in range(n)]
    color_distr = dict(zip(list(range(1, n+1)), color_verts))
    adjs = []
    m = 1
    for i in range(1, n+1):
        for j in range(1, n+1):
#            print(m)
            m += 1
            check_colors = color_verts[i-1] == color_verts[j-1]
            prob = random.randint(0, 1)
            exist = True if (j, i) in adjs else False
            if prob and not check_colors and not exist and i!=j:
                new_adj = (i, j) if i < j else (j, i)
                adjs.append(new_adj)

#        for j in range(i+1, n+1):
#            print(i, j)
#            check_colors = color_verts[i-1] == color_verts[j-1]
#            prob = random.randint(0, 1)
#            if prob and not check_colors:
#                new_adj = (i, j)
#                adjs.append(new_adj)

    return n, adjs, color_distr

# Напишем верификатор
# На вход подаются граф и сертификат
# Граф в формате graph = (n, adj_list),
# где n - число вершин графа, а adj_list - список смежности вершин в виде
# [(v1,v2), (v1,v3), ..., (vn-1, vn)]
# Сертификат в виде словаря {вершина: цвет}

def verify(graph, cert):
    n = graph[0]
    adjs = graph[1]
    for adj in adjs:
        vert1 = adj[0]
        vert2 = adj[1]
        if cert[vert1] == cert[vert2]:
            return False
    return True

# Напишем функцию для тестов с сертификатами, полученными с помощью генератора тестов
def graph_col_test(n):
    j = 1
    for i in n:
        graph_test = col_func_test(i)
        print('Тест', j)
```

```
        print(verify((graph_test[0], graph_test[1]), graph_test[2]))
        print('-----')
        print()
        j += 1

# Тесты с сертификатами, полученными с помощью генератора тестов
# Каждый тест должен вернуть True
n = [10, 30, 50, 70, 90] # Задаем число вершин для каждого из 5 тесовых графов
graph_col_test(n)

# Зададим случайный граф, который заведомо нельзя раскрасить в 3 цвета
# Тест должен вернуть False
n1 = 5
adjs1 = [(1, 2), (1, 3), (1, 4), (4, 5)]
graph1 = (n1, adjs1)
cert1 = {1: 'r', 2: 'g', 3: 'b', 4: 'r', 5: 'g'}

print('Тест', 6)
print(verify(graph1, cert1))
print('-----')
```

Оценка сложности

1. Количество итераций внешнего цикла For -- O(n)
2. Количество итераций внутреннего цикла For -- O(n)
3. Количество итераций присвоения check_colors, exist() -- O(n^2)
4. Общее время работы 1-го цикла -- O(n*n*n^2)=O(n^4)
5. Количество итераций 2-го цикла -- O((n*(n-1))/2)=O(n^2)
6. Общее время работы алгоритма -- O(n^4+n^2)=O(n^4)

Тестирование

Тест 1

Дан граф размера 130

Теоретическая сложность O(285 610 000)

Практическая сложность 1 сек

Дан граф размера 170

Теоретическая сложность O(835 210 000)

Практическая сложность 2 сек

Вывод

Практическая сложность увеличивается ~ в 2 раз при увеличении размера входа в 1,3 раза, что не превышает теоретической оценки сложности (~ в 2,9 раз)

Тест 2

Дан граф размера 130
Теоретическая сложность O(285 610 000)
Практическая сложность 1 сек

Дан граф размера 180
Теоретическая сложность O(1 049 760 000)
Практическая сложность 3 сек

Вывод
Практическая сложность увеличивается ~ в 3 раз при увеличении размера входа в 1,4 раза, что не превышает теоретической оценки сложности (~ в 3,8 раз)

Тест 3
Дан граф размера 130
Теоретическая сложность O(285 610 000)
Практическая сложность 1 сек

Дан граф размера 210
Теоретическая сложность O(1 944 810 000)
Практическая сложность 6 сек

Практическая сложность увеличивается ~ в 6 раз при увеличении размера входа в 1,6 раза, что не превышает теоретической оценки сложности (~ в 6,6 раз)

Тест 4
Дан граф размера 130
Теоретическая сложность O(285 610 000)
Практическая сложность 1 сек

Дан граф размера 220
Теоретическая сложность O(2 342 560 000)
Практическая сложность 7 сек

Практическая сложность увеличивается ~ в 7 раз при увеличении размера входа в 1,7 раза, что не превышает теоретической оценки сложности (~ в 8,4 раз)

Тест 5
Дан граф размера 130
Теоретическая сложность O(285 610 000)
Практическая сложность 1 сек

Дан граф размера 260
Теоретическая сложность O(4 569 760 000)
Практическая сложность 14 сек

Практическая сложность увеличивается ~ в 14 раз при увеличении размера входа в 2 раза, что не превышает теоретической оценки сложности (~ в 16 раз)