Министерство образования и науки Российской Федерации Федеральное агентство по образованию

Федеральное государственное бюджетное образовательное учреждение высшего образования

«Вятский государственный университет»

Факультет автоматики и вычислительной техники

Кафедра электронных вычислительных машин

**Исследование и последовательная реализация вычислительно сложного алгоритма**

Лабораторная работа №1

по курсу «Параллельное программирование»

Вариант 9

Выполнил студент группы ИВТб-31 /Категов А. Д./ Проверил преподаватель /Мельцов В. Ю./

Киров 2024

Задание

1. Изучить алгоритм, полученный в соответствии с выданным преподавателем вариантом и провести доказательную оценку алгоритма по временной сложности и затратам памяти
2. Реализовать алгоритм с помощью языка С++
3. Построить набор тестовых примеров (не менее 10) и провести оценку эффективности реализованного алгоритма
4. Изучение алгоритма

Жадный алгоритм упорядочивает вершины 𝑣1… 𝑣n в соответствии с некоторой перестановкой и последовательно присваивает вершине 𝑣i наименьший доступный цвет, не использовавшийся для окраски соседей 𝑣i среди 𝑣1 … 𝑣n-1, либо добавляет новый. Качество полученной раскраски зависит от выбранного порядка. Для улучшения результата можно использовать несколько различных перестановок вершин.

Асимптотическая сложность данного алгоритма: O(kn2). Сложность по памяти O(n). Для каждой перестановки вершин (всего 𝑘 перестановок), сгенерированной случайно, внешний цикл перебирает все вершины 𝑣i и для каждой определяет цвета ее соседей, что в худшем случае будет выполнено за 𝑛 шагов.

1. Программная реализация

Листинг программной реализации алгоритма приведен в приложении А.

1. Тестирование

В ходе тестирования выполнялась раскраска графов, сгенерированных случайным образом. Количество вершин и ребер каждого графа и результаты тестирования приведены в таблице 1.

Таблица 1 – Результаты тестирования

|  |  |
| --- | --- |
| Граф (кол-во вершин, кол-во ребер) | Время, мс |
| 5, 7 | 8 |
| 75, 1000 | 102 |
| 75, 2500 | 142 |
| 90, 300 | 93 |
| 95, 2500 | 210 |
| 150, 5000 | 369 |
| 150, 10000 | 508 |
| 200, 5000 | 400 |
| 200, 10000 | 625 |
| 170, 10000 | 455 |
| 250, 14000 | 643 |
| 250, 17000 | 666 |
| 300, 34000 | 1197 |
| 400, 40000 | 1822 |

1. Вывод

В ходе лабораторной работы был реализован «жадный» алгоритм раскраски графа. Для улучшения качества раскраски перебор вершин выполнялся несколько раз, порядок вершин определялся случайной перестановкой. В ходе тестирования было замечено, что время работы алгоритма увеличивается как при увеличении числа вершин, так и при увеличении числа ребер. Второй критерий в большей степени влияет на время выполнения алгоритма.

Приложение А

(обязательное)

Листинг программы

#include <vector>

#include <iostream>

#include <cstdint>

#include <set>

#include <unordered\_set>

#include <algorithm>

#include <fstream>

#include <chrono>

#include <iomanip>

using namespace std;

using namespace chrono;

using graph\_t = vector<vector<size\_t>>;

istream& operator>>(istream& is, graph\_t& graph) {

size\_t n; is >> n; // vertexes

size\_t m; is >> m; // edges

graph.clear();

graph.resize(n);

for (size\_t i = 0; i < m; ++i) {

size\_t a, b;

is >> a >> b;

graph[a].push\_back(b);

graph[b].push\_back(a);

}

return is;

}

size\_t \_mex(const vector<size\_t>& set) {

return static\_cast<size\_t>(find(set.begin(), set.end(), 0) - set.begin());

}

size\_t colorize(const graph\_t& graph) {

size\_t size = graph.size();

vector<size\_t> order(size);

size\_t orders\_count = 500;

vector<vector<size\_t>> orders(orders\_count);

for (size\_t i = 0; i < size; ++i) {

order[i] = i;

}

for (auto& ord : orders) {

random\_shuffle(order.begin(), order.end());

ord = order;

}

size\_t min = graph.size();

for (const auto& ord : orders) {

vector<size\_t> colored(size, 0);

vector<size\_t> colors(size, 0);

vector<size\_t> used\_colors(size, 0);

for (size\_t v : ord) {

for (auto to : graph[v]) {

if (colored[to]) {

used\_colors[colors[to]] = 1; //этот цвет занят

}

}

auto c = \_mex(used\_colors); //ищем незанятый цвет

colored[v] = 1; //помечаем вершину как раскрашеную

colors[v] = c; //раскрашиваем вершину

used\_colors.assign(size, 0); //сброс всех цветов

}

size\_t colors\_count = 1 + \*max\_element(colors.begin(), colors.end());

min = std::min(min, colors\_count);

}

return min;

}

int main() {

ifstream input("input.txt");

ofstream output("output.txt");

graph\_t graph;

size\_t cnt;

input >> cnt;

output << cnt << endl;

for (size\_t i = 0; i < cnt; ++i) {

input >> graph;

auto start = chrono::system\_clock::now();

auto res = colorize(graph);

auto stop = chrono::system\_clock::now();

auto time = duration\_cast<milliseconds>(stop - start).count();

output << res << endl;

cout << time << endl;

}

return 0;

}