**ЛАБОРАТОРНА РОБОТА #2**

З дисципліни «Комп’ютерні системи штучного інтелекту»

На тему «Мурашиний алгоритм»

|  |  |
| --- | --- |
| **Виконав**  Студент групи КВ-21  Комарницький О.Б. | **Перевірила**  Доцент кафедри СП і СКС  Зорін Ю.М. |

|  |  |  |
| --- | --- | --- |
| **Оцінка** | **Дата** | **Підпис** |
|  |  |  |

**Варіант #8**

Задание на работу.

1. Написать программу реализации приведенного алгоритма для задачи о k-раскраске графа.
2. Найти бесконфликтную раскраску для хроматических чисел следующих групп графов:

а) легкие: yuzGCP130.13.col, yuzGCP660.33.col и yuzGCP990.33.col;

b) средние: yuzGCPrnd83.10.col, yuzGCPrnd127.14.col и yuzGCPrnd157.10.col;

с) трудные: yuzGCPrnd83.9.col, yuzGCPrnd127.13.col и yuzGCPrnd157.9.col.

**Файл Graph.h**

#pragma once

#include <iostream>

#include <vector>

#include <string>

#define COLORS\_SIZE 26

using namespace std;

struct Vertice

{

int index;

int color;

};

class Graph

{

private:

vector<Vertice> points; // pair<index, color>

vector<pair<int, int>> edges;

int \* colorsConflicts();

int countMaxConf(); // кол-во конфликтов в наихудшей вершине данного решения.

public:

unsigned POINTS\_NUMBER;

unsigned EDGES\_NUMBER;

Graph(string);

~Graph();

void read(string name);

void show();

double pn(int iter);

Vertice \* maxConfVertice(int except\_index);

int minConfColor();

int countConfsoverall(); // общее кол-во конфликтов в данном решении (муравье)

vector<Vertice> & getPoints();

vector<pair<int, int>> & getEdges();

vector<Vertice \*> getNeighbours(int index);

};

**Файл Graph.cpp**

#include "Graph.h"

#include <fstream>

Graph::Graph(string filename)

{

read(filename);

for (unsigned i = 0; i < POINTS\_NUMBER; i++) {

Vertice v;

v.color = 0;

v.index = i;

points.push\_back(v);

}

}

Graph::~Graph() {}

void Graph::read(string name)

{

ifstream f(name);

string line = "";

int counter = 1, p1, p2;

while (getline(f, line) && counter)

{

if (line == "") break;

string tmp;

switch (line[0])

{

case 'c':

continue;

break;

case 'p':

tmp = line.substr(line.find("edge ") + 5);

POINTS\_NUMBER = stoi(tmp.substr(0, tmp.find(" ")));

counter = EDGES\_NUMBER = stoi(tmp.substr(tmp.find(" ") + 1));

break;

case 'e':

tmp = line.substr(line.find("e ") + 2);

p1 = stoi(tmp.substr(0, tmp.find(" "))) - 1;

p2 = stoi(tmp.substr(tmp.find(" ") + 1)) - 1;

edges.push\_back(pair<int, int>(p1, p2));

counter--;

break;

default:

break;

}

}

}

void Graph::show()

{

//for (unsigned i = 0; i < points.size(); i++)

// cout << "#" << points[i].index << " color=" << points[i].color << endl;

cout << "All confs: " << countConfsoverall() << endl;

}

int \* Graph::colorsConflicts() // index - colorID, value - conflicts number

{

int \* cols = new int[COLORS\_SIZE];

for (unsigned i = 0; i < COLORS\_SIZE; i++)

cols[i] = 0;

for (unsigned i = 0; i < POINTS\_NUMBER; i++)

cols[points[i].color]++;

return cols;

}

int Graph::countMaxConf()

{

int max\_conf = 0;

for (unsigned i = 0; i < POINTS\_NUMBER; i++) {

vector<Vertice \*> neighbours = getNeighbours(points[i].index);

int count = 0;

for (unsigned j = 0; j < neighbours.size(); j++)

if (neighbours[j]->color == points[i].color)

count++;

if (count > max\_conf)

max\_conf = count;

}

return max\_conf;

}

int Graph::countConfsoverall()

{

int sum = 0;

for (unsigned i = 0; i < POINTS\_NUMBER; i++)

for (unsigned j = 0; j < edges.size(); j++) {

// Find edges from current vertice

if (edges[j].first == points[i].index)

if (points[edges[j].second].color == points[i].color)

sum++;

if (edges[j].second == points[i].index)

if (points[edges[j].first].color == points[i].color)

sum++;

}

return sum / 2;

}

double Graph::pn(int iter)

{

double avgX = 1. \* countMaxConf();

double avgY = 4.8\*countConfsoverall() / (POINTS\_NUMBER \* POINTS\_NUMBER);

return exp(-3.2\*((5 \* iter + 1)\*avgY / (avgX)));

}

Vertice \* Graph::maxConfVertice(int except\_index)

{

int max\_conflicts = 0;

Vertice \* v = &points[0];

for (unsigned i = 0; i < POINTS\_NUMBER; i++) {

if (i == except\_index) continue;

vector<Vertice \*> neighbours = getNeighbours(points[i].index);

int count = 0;

for (unsigned j = 0; j < neighbours.size(); j++)

if (points[i].color == neighbours[j]->color)

count++;

if (count > max\_conflicts) {

max\_conflicts = count;

v = &points[i];

}

}

return v;

}

int Graph::minConfColor()

{

int \* cols = colorsConflicts();

int min\_conf\_color = 0;

int min\_confs = cols[0];

for (unsigned i = 0; i < COLORS\_SIZE; i++)

if (cols[i] < min\_confs) {

min\_confs = cols[i];

min\_conf\_color = i;

}

delete cols;

return min\_conf\_color;

}

vector<Vertice>& Graph::getPoints()

{

return points;

}

vector<pair<int, int>>& Graph::getEdges()

{

return edges;

}

vector<Vertice \*> Graph::getNeighbours(int index)

{

vector<Vertice \*> v = vector<Vertice \*>();

for (unsigned i = 0; i < EDGES\_NUMBER; i++) {

if (edges[i].first == index)

v.push\_back(&points[edges[i].second]);

if (edges[i].second == index)

v.push\_back(&points[edges[i].first]);

}

return v;

}

**Файл Graph.cpp**

#include "Graph.h"

#include <iostream>

#include <ctime>

#define ANTS\_SIZE 20

void init();

int main() {

init();

system("pause");

return 0;

}

#include <stdio.h>

void init()

{

Graph g("yuzGCP990.33.col");

vector<Vertice \*> ants(ANTS\_SIZE, &g.getPoints()[0]); //= &g.getPoints()[0];

srand((unsigned)time(0));

int iter = 0;

while (ants.size()) {

for (unsigned i = 0; i < ants.size(); i++) {

if (rand() % 100 < 100 \* g.pn(iter)) {

// Select max conflict

ants[i] = g.maxConfVertice(ants[i]->index);

ants[i]->color = g.minConfColor();

}

else {

// Select neighbour vertices

vector<Vertice \*> tmp = g.getNeighbours(ants[i]->index);

ants[i] = tmp[rand() % tmp.size()];

ants[i]->color = g.minConfColor();

}

if (ants[i]->index == g.POINTS\_NUMBER - 1) {

ants.erase(ants.begin() + i);

i--;

}

}

g.show();

//printf("%20.18f\n", g.pn(iter));

iter++;

}

cout << "Graph painted successfully!!!" << endl;

}