KAUNO TECHNOLOGIJOS UNIVERSITETAS

INFORMATIKOS FAKULTETAS

TAIKOMOSIOS INFORMATIKOS KATEDRA

DISKREČIOSIOS STRUKTŪROS (P170B008)

KURSINIS DARBAS

Užduoties nr. B36

Atliko:

IFD-4 gr. studentas

<Lukas Dargevičius>

Priėmė:

Dėst. Vardenis Pavardenis

KAUNAS

2025

Turinys

1. Užduotis (B36)

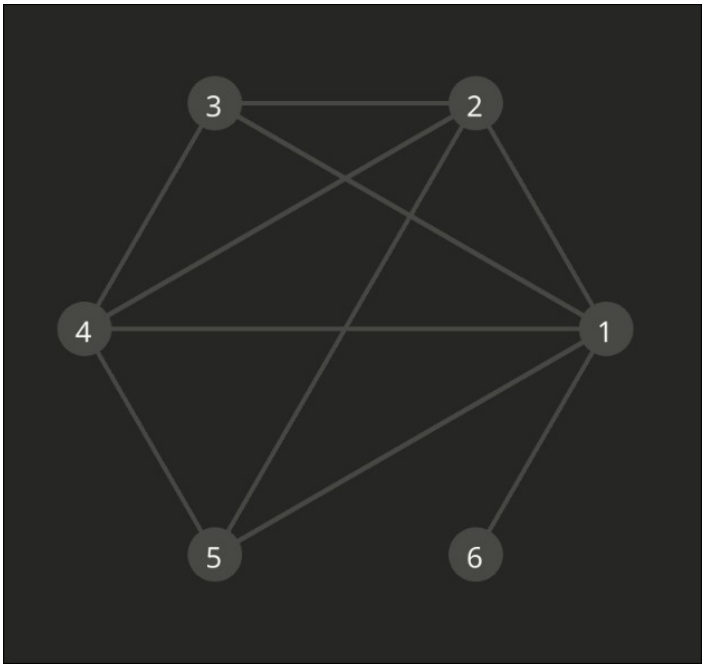
Nustatyti, ar duotasis grafas turi paprastąjį ciklą einantį per nurodytą viršūnių skaičių.

2. Užduoties analizė

Paprastasis ciklas – tai grandinę kurios pirmoji ir paskutinė viršūnės sutampa, bei visos ją sudarančios briaunos yra skirtingos.

Uždavinys. Duotas grafas $G = (V,U)$, kur n – grafo viršūnių skaičius ir m – grafo briaunų skaičius, V – viršūnių, o U – briaunų aibės, taipogi duota t - viršūnių seka, kurioje reikia patikrinti ar yra paprastasis ciklas.

Metodo idėja. Pradžioje tarkime, kad turime grafą su briaunomis ir viršūnių seką t. Einame per kiekvieną šios sekos viršūnę. Naudodami algoritmą nustatome, kiek iš tos viršūnės galima sudaryti paprastųjų ciklų. Vėliau šiuos ciklus galime parodyti naudotojui grafinėje sąsajoje.



pav 1. pradinis grafas

Pavyzdžiui, 1 pav. pavaizduotam grafui skaičiuojame ciklus pagal t viršūnių seka.

t viršūnių seka: 1,3,2,5,4

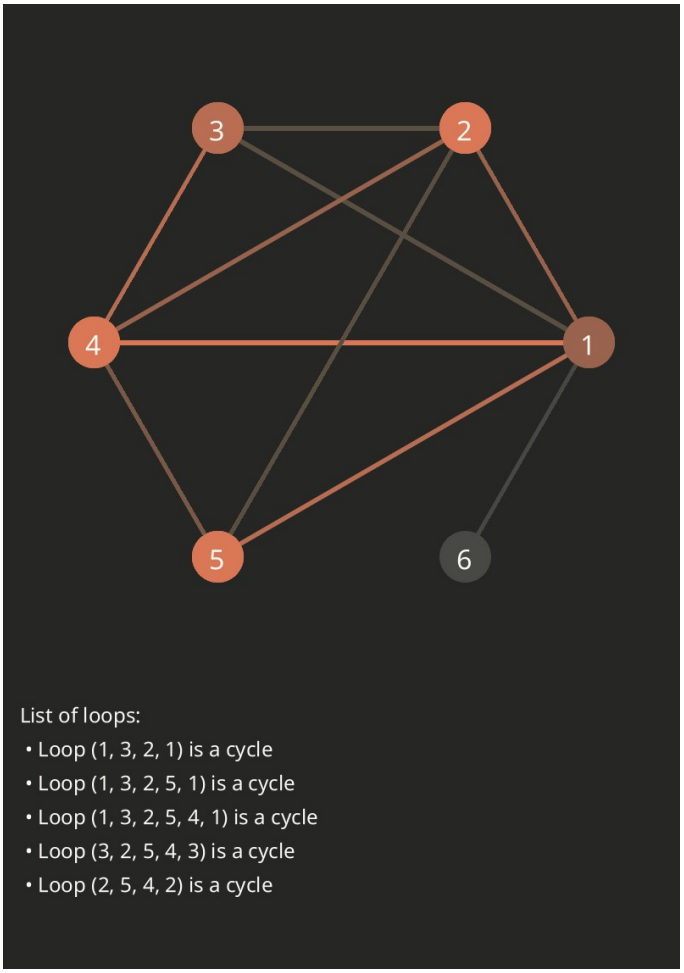
algoritmas suskaiciuoja:

List of loops:

- Loop (1, 3, 2, 1) is a cycle
- Loop (1, 3, 2, 5, 1) is a cycle
- Loop (1, 3, 2, 5, 4, 1) is a cycle

• Loop (3, 2, 5, 4, 3) is a cycle

• Loop (2, 5, 4, 2) is a cycle



pav 2. Galutinis nupiestas rezultatas

Įvedus visas briaunas gautas rezultatas rodo, kad šis grafas, su *t* viršūnių seka, turi 5 ciklus.

3. Programos algoritmo aprašymas

Programa iš tekstinio failo Data.txt nuskaito grafą, sudarytą iš aprašytų briaunų ir viršūnių, taip pat iš to paties failo gauna ir *t* viršūnių seką. Visi šie duomenys sudedami į Node struktūrų masyvą.

Iš pradžių ciklų skaičius yra 0, nes ciklų masyvas ListOfViableLoops yra tuščias. Patikriname, ar *t* viršūnių seka turi daugiau nei dvi viršūnes, nes kitu atveju ciklas negalimas. Tuomet einame per kiekvieną viršūnę *i, t* sekoje, kur *i* indeksas yra mažesnis nei t.len() - 2.

Kiekviename žingsnyje sudarome naują viršūnių seką: nuo *i* iki paskutinės viršūnės *t* sekoje ir tikriname, ar ši nauja seka yra paprastasis ciklas. Tikrinimas atliekamas einant per visą grandinę: žiūrima, ar galima pereiti iš vienos grandinės viršūnės į kitą, taip pat ar nebuvome jau aplankę tos pačios viršūnės ar briaunos. Jeigu viršūnių seka atitinka paprastojo ciklo sąlygas, ją įtraukiame į bendrą ciklų masyvą (ListOfViableLoops).

Radę visus galimus ciklus, juos galima atvaizduoti grafinėje sąsajoje.

Algoritmas ras teisingą atsakymą, nes mums tinka tik tie ciklai, kurie yra *t* viršūnių sekoje, o mūsų algoritmas optimizuotai tikrina tik tuos galimus ciklus, kurie yra pasiekiami iš kiekvienos viršūnės *t* sekoje.

1 lentelė. Grafo aprašymo būdų palyginimas

Kriterijus	**Gretimumo matrica**	**Incidencijų matrica**	**Briaunų matrica**	**Gretimumo struktūra**	**Gretimumo struktūra, užrašyta masyvais L ir lst**
Gavimo iš įvedimo duomenų paprastumas	paprasta	paprasta	paprasta	duomenys taip pateikiami	paprasta
Viršūnių, gretimų duotajai, radimo paprastumas	vidutiniškai sunku	sunku	vidutiniškai sunku	paprasta	paprasta

Grafui aprašyti buvo pasirinkta gretimumo struktūra, nes darbui atlikti naudota Rust programavimo kalba. Ši kalba yra sparčiai populiarėjanti pasaulyje dėl savo greičio ir atminties saugumo. Rust naudoja nauja programavimo principą „borrow checking“, kuris užtikrina visišką atminties saugumą. Be to, Rust neturi šiukšlių rinkėjo (garbage collector), todėl yra neįtikėtinai greita - savo našumu prilygsta C ir C++ programavimo kalboms. Naudota Rust struct duomenų struktūra, kuri leido lengvai įgyvendinti gretimumo struktūrą. Taip pat buvo pasitelktos šios bibliotekos: std – įvesties, išvesties ir failų operacijoms, plotters - paprasta piešimo biblioteka, leidžianti kurti vaizdus ir grafikus, rand - atsitiktiniams skaičiams generuoti.

4. Programos tekstas

main.rs

```
#![allow(nonstandard_style)]

extern crate plotters;
use plotters::prelude::*;
use plotters::style::FontStyle;
use std::collections::HashMap;
use std::{fs, path, vec};
use rand::Rng;


struct Node {
id: u64,
neighbors: Vec<u64>,
}

// Extracting data
fn readFileString(file_path : String) -> String{
let content = fs::read_to_string(file_path);
match content {
Ok(data) => {
String::from(data)
},
Err(_) => String::from("not ok"),
}
}

fn extractNodeFromData(dataBlob : String) -> Node{
let Split: Vec<&str> = dataBlob.split(":").collect();
let id: u64 = Split[0].parse::<u64>().unwrap();

let mut neigbors: Vec<u64> = Vec::new();
let neigborsStrings : Vec<&str> = Split[1].split(",").collect();

for neib in neigborsStrings{
neigbors.push(neib.parse::<u64>().unwrap());
}

let Node : Node = Node { id: id, neighbors: neigbors};

return Node;
}

fn extractPathFromData(dataBlob : String) -> Vec<u64>{
let mut neigbors: Vec<u64> = Vec::new();
let neigborsStrings : Vec<&str> = dataBlob.split(",").collect();

for neib in neigborsStrings{
neigbors.push(neib.parse::<u64>().unwrap());
}

return neigbors;
```

```
}

fn extractListData(file_path : String) -> (Vec<Node>, Vec<u64>) {
let StringData = readFileString(file_path);

let lines: Vec<&str> = StringData.split('\n').collect();

let mut NodesList: Vec<Node> = Vec::new();
let mut PathList: Vec<u64> = Vec::new();

for line in lines {
if line.contains(":") {
NodesList.push(extractNodeFromData(String::from(line)))
} else if line.contains(",") {
PathList = extractPathFromData(String::from(line));
}
}

return (NodesList, PathList);
}

// Utilities
fn find_node_by_id(node_list: &Vec<Node>, id: u64) -> Option<&Node> {
node_list.iter().find(|&node| node.id == id)
}

fn isNotInList(tryingToFind : u64, listOfNeighb : &Vec<u64>) -> bool {
if listOfNeighb.contains(&tryingToFind) {
return false;
}
return true;
}

// Main algorithm for finding if its true
fn checkIfLoopIsCorrect(nodeList : &Vec<Node>, pathList : &Vec<u64>) -> bool {
if pathList.last() != Some(&pathList[0]) { return false; }

let mut alreadyUsedNodeList : Vec<u64> = Vec::new();

for i in 0..(pathList.len()-1) {
let nextNodeId = pathList[i+1];
let currentNode = match find_node_by_id(&nodeList,pathList[i]) {
Some(node_ref) => node_ref,
None => {
println!("There is no node with id that is found in path\n");
return false;
}
};

if alreadyUsedNodeList.contains(&currentNode.id) {
println!("Point was used more then 1 time\n");
return false;
}

if isNotInList(nextNodeId, &currentNode.neighbors) {
println!("path list is not a complete / viable line to make a cicle\n");
return false;
}
```

```
alreadyUsedNodeList.push(currentNode.id);
}

return true;
}

fn _TestingPrint(path_list: &Vec<u64>) {
let mut print_string = String::from("Path: ");

for value in path_list {
print_string.push_str(&value.to_string());
print_string.push(' ');
}

println!("{}", print_string);
}



fn getAllValidLoops(nodeList : &Vec<Node>, mainGrandine : &Vec<u64>) -> Vec<Vec<u64>>{
let mut results: Vec<Vec<u64>> = Vec::new();
if mainGrandine.len() < 3 {
return results;
}


for i in 0..(mainGrandine.len() - 2) {
let current_node_id = mainGrandine[i];

let currentNodeFullGrandine = mainGrandine[i..].to_vec();

for u in 2..currentNodeFullGrandine.len() {
let mut loopToCheck = currentNodeFullGrandine[..=u].to_vec(); // Tipo teoriskai galeciau
sita padaryt ir be currentNodeFullGrandine
loopToCheck.push(current_node_id);

print!("Testing ");
_TestingPrint(&loopToCheck);

let isSubLoopViable = checkIfLoopIsCorrect(nodeList, &loopToCheck);
if isSubLoopViable {
println!("Is A Good Loop \n");
results.push(loopToCheck);
}
}
}


return results;
}

// DRAWING OUTPUT

fn get_color(max_index: u32, current_index: u32) -> RGBColor {
let max_color = RGBColor(217, 119, 87);
let min_color = RGBColor(89, 79, 66);

if max_index == 0 {
return min_color;
}
```

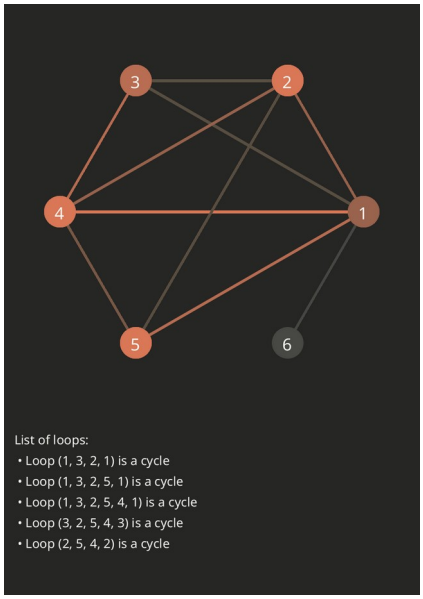
```
let t = current_index as f64 / max_index as f64;

let r = min_color.0 as f64 + t * (max_color.0 as f64 - min_color.0 as f64);
let g = min_color.1 as f64 + t * (max_color.1 as f64 - min_color.1 as f64);
let b = min_color.2 as f64 + t * (max_color.2 as f64 - min_color.2 as f64);

RGBColor(r.round() as u8, g.round() as u8, b.round() as u8)
}

// fn get_color(max_index: u32, current_index: u32) -> RGBColor {
// let mut rng = rand::rng();

// let r = rng.random_range(0..=255);
// let g = rng.random_range(0..=255);
// let b = rng.random_range(0..=255);

// RGBColor(r, g, b)
// }

fn draw_image(node_list: Vec<Node>, ListOfPaths : Vec<Vec<u64>>, text_to_output: String) ->
Result<(), Box<dyn std::error::Error>> {
let node_count = node_list.len();
let mut node_positions: HashMap<u64, (f64, f64)> = HashMap::new();

let radius = 300.0;
let center_x = 400.0;
let center_y = 400.0;
for (i, node) in node_list.iter().enumerate() {
let angle = 2.0 * std::f64::consts::PI * (i as f64) / (node_count as f64);
let x = center_x + radius * angle<span style="col
```

5. Testavimo pavyzdžiai

Buvo panaudoti trys testavimo pavyzdžiai:

Pirmas testas

Pradiniai duomenys (Data.txt):	1:2,3,4,5,6 2:1,3,4,5 3:1,2,4 4:1,2,3,5 5:1,2,4 6:1 1,3,2,5,4



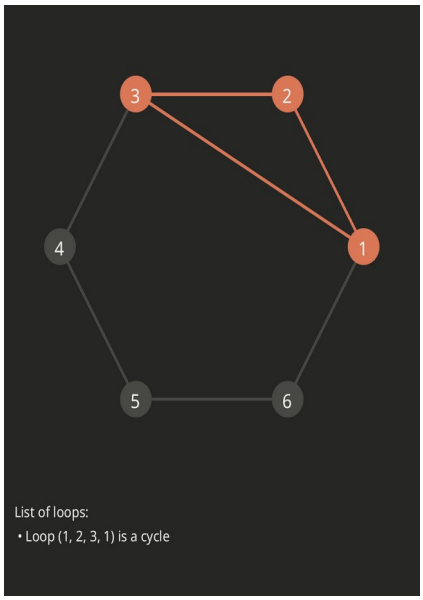
pav 3. Pirmo testo rezultatas

Programos atsakymas:	List of loops: • Loop (1, 3, 2, 1) is a cycle • Loop (1, 3, 2, 5, 1) is a cycle • Loop (1, 3, 2, 5, 4, 1) is a cycle

	• Loop (3, 2, 5, 4, 3) is a cycle • Loop (2, 5, 4, 2) is a cycle
Algoritmas truko laiko:	25.918µs.

Antras testas

Pradiniai duomenys (Data.txt):	1:2,3,6 2:1,3 3:1,2,4 4:3,5 5:4,6 6:1,5 1,2,3,4,5

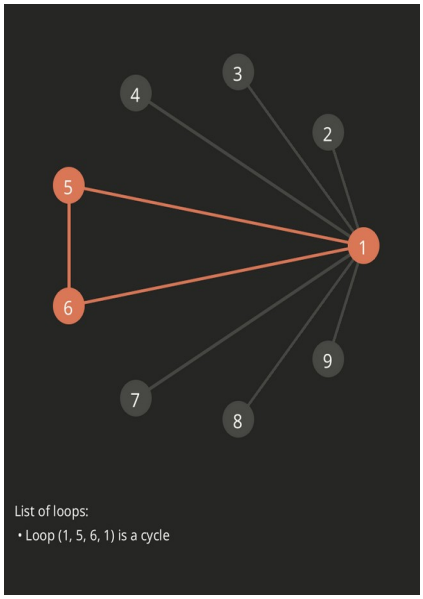


pav 4. Antro testo rezultatas

Programos atsakymas:	List of loops: • Loop (1, 2, 3, 1) is a cycle
Algoritmas truko laiko:	27.687µs.

Trecias testas

Pradiniai duomenys (Data.txt):	1:2,3,4,5,6,7,8,9 2:1 3:1 4:1 5:1,6 6:1,5 7:1 8:1 9:1 1,5,6,7



pav 5. Trecio testo rezultatas

Programos atsakymas:	List of loops: • Loop (1, 5, 6, 1) is a cycle
Algoritmas truko laiko:	13.563μs.

6. Išvados

Programa veikia teisingai ir pakankamai greitai. Nors Rust programavimo kalba yra greita, mano nuomone, ji ne visada tinkama tokio tipo užduotims. Kadangi tai kompiliuojama kalba, norint programą perkelti į kitą kompiuterio architektūrą, ją reikia iš naujo sukompiliuoti. Be to, Rust yra ganėtinai sudėtinga, palyginus su Python, R ar MATLAB. Vis dėlto, esant poreikiui apdoroti milžiniškus kiekius duomenų, ši programavimo kalba gali būti geras pasirinkimas.

7. Literatūros sąrašas

1. Rust dokumentacija https://doc.rust-lang.org/stable/ (žiūrėta 2025-12-02)
2. Rust plotters bibliotekos dokumentacija https://docs.rs/plotters/latest/plotters (žiūrėta 2025-12-02)
3. Rust std bibliotekos dokumentacija https://doc.rust-lang.org/std/ (žiūrėta 2025-12-02)
4. Rust rand bibliotekos dokumentacija https://docs.rs/rand/latest/rand/ (žiūrėta 2025-12-02)
5. „Diskrečiųjų struktūrų“ modulis „Moodle“ aplinkoje https://moodle.ktu.edu/course/view.php?id=39 (žiūrėta 2025-12-02)