

Міністерство освіти і науки України
Національний технічний університет України «Київський політехнічний
інститут імені Ігоря Сікорського»
Факультет інформатики та обчислювальної техніки

Кафедра ІІІ

Звіт

з лабораторної роботи №4 з дисципліни
«Алгоритми та структури даних 2. Структури даних»

„ Прикладні задачі теорії графів”

Виконав ІІ-45 Янов Богдан Євгенійович

Перевірів Соколовський Владислав Володимирович

Київ 2025

Лабораторна робота №4

Прикладні задачі теорії графів

Мета: вивчити основні прикладні алгоритми на графах та способи їх імплементації.

Граф - це математична структура, яка складається з вершин (вузлів) та ребер (зв'язків між вершинами). Графи бувають орієнтовані (ребра мають напрям) та неорієнтовані (напрямку немає), а також зважені (ребра мають вагу) і незважені.

Пошук Таррі (Tarry Traversal) - це алгоритм обходу графа, який гарантує, що кожне ребро буде пройдено не більше одного разу. Це обхід у глибину (DFS) з додатковою умовою: перед переходом по ребру алгоритм перевіряє, чи воно вже використовувалось. Такий підхід виключає зациклення і забезпечує проходження всіх ребер у графі, що корисно, наприклад, для побудови маршрутів у мережах.

Алгоритм Таррі не шукає найкоротший шлях - його ціль саме обхід усіх ребер без повторів.

Псевдокод алгоритмів

```
dfs(u, end, adj, edge_mark, path) {
    push(path, u)

    if u == end {
        return true
    }

    n = len(adj)
    for v from 0 to n {
        if adj[u][v] == 1 and !edge_mark[u][v] {
            edge_mark[u][v] = true
            edge_mark[v][u] = true

            if dfs(v, end, adj, edge_mark, path)
                return true
        }
    }

    pop(path)
    return false
}

tarry_traversal(adj_matrix, start, end) {
    n = len(adj_matrix)
    if start >= n or end >= n {
        return
    }

    edge_mark = nxn matrix
```

```

    path = []

    if dfs(start, end, adj_matrix, edge_mark, path) {
        return path
    }

    return
}

```

Вихідний код

```

use std::env;
use std::fs::File;
use std::io::{self, BufRead};
use std::path::Path;

fn tarry_traversal(adj_matrix: &[Vec<u8>], start: usize, end: usize)
-> Option<Vec<usize>> {
    let n = adj_matrix.len();
    if start >= n || end >= n {
        return None;
    }

    let mut edge_mark = vec![vec![false; n]; n];
    let mut path = Vec::new();

    fn dfs(
        u: usize,
        end: usize,
        adj: &[Vec<u8>],
        edge_mark: &mut Vec<Vec<bool>>,
        path: &mut Vec<usize>,
    ) -> bool {
        path.push(u);
        if u == end {
            return true;
        }
    }
}

```

```

let n = adj.len();
for v in 0..n {
    if adj[u][v] == 1 && !edge_mark[u][v] {
        edge_mark[u][v] = true;
        edge_mark[v][u] = true;

        if dfs(v, end, adj, edge_mark, path) {
            return true;
        }
    }
}

path.pop();
false
}

if dfs(start, end, adj_matrix, &mut edge_mark, &mut path) {
    Some(path)
} else {
    None
}
}

fn main() {
    let args: Vec<String> = env::args().collect();
    let (start, end) = if args.len() >= 3 {
        (
            args[1].parse::<usize>().unwrap_or(1),
            args[2].parse::<usize>().unwrap_or(1),
        )
    } else {
        println!("Usage: {} <start> <end>", args[0]);
        return;
    };
};

let path = "data.txt";

```

```

let mut adj_matrix = Vec::new();
if let Ok(lines) = read_lines(path) {
    for line in lines.flatten() {
        let row: Vec<u8> = line
            .split_whitespace()
            .filter_map(|s| s.parse:::<u8>().ok())
            .collect();
        adj_matrix.push(row);
    }
} else {
    println!("Failed to open file: {}", path);
    return;
}

if let Some(path) = tarry_traversal(&adj_matrix, start - 1, end -
1) {
    println!(
        "Path found: {:?}",
        path.iter().map(|x| x + 1).collect:::<Vec<_>>()
    );
} else {
    println!("No path found.");
}
}

fn read_lines<P>(filename: P) ->
io::Result<io::Lines<io::BufReader<File>>>
where
    P: AsRef<Path>,
{
    let file = File::open(filename)?;
    Ok(io::BufReader::new(file).lines())
}

```

Приклад роботи



- `λ Partur lab4 → λ git master* → cat data.txt`
0 1 0 0 0 0 1 0 0
1 0 1 1 0 0 0 0 1
0 1 0 0 0 1 0 0 0
0 1 0 0 1 0 0 1 0
0 0 0 1 0 0 0 1 0
0 0 1 0 0 0 0 0 0
1 0 0 0 0 0 0 0 0
0 0 0 1 1 0 0 0 0
0 1 0 0 0 0 0 0 0
- `λ Partur lab4 → λ git master* → cargo run -- 6 8`
Finished `dev` profile [unoptimized + debuginfo] target(s) in 0.04s
Running `target/debug/lab4 6 8`
Path found: [6, 3, 2, 4, 5, 8]
- ✧ `λ Partur lab4 → λ git master* → |`

Висновок

У ході виконання лабораторної роботи було реалізовано алгоритм пошуку Таррі для обходу графа, представленого у вигляді матриці суміжності. Було розглянуто принцип роботи алгоритму, який забезпечує обхід усіх вершин і ребер без повторного використання ребер, що запобігає зацикленню. Такий підхід є корисним для побудови маршрутів у мережах та аналізу структури графа. Отримані результати підтверджують правильність реалізації та ефективність методу для задач обходу графів.