პროგრამირების აბსტრაქციები
სემინარის ამოცანები #26

მანძილის და კომპონენტების რაოდენობის პოვნა "bfs"-ით

მიმართული გრაფის კლასში დაამატეთ ორი ფუნქცია: "shortestPath" და "numComponents". პირველი მათგანი უნდა პოულობდეს ორ წვეროს შორის უმოკლესი გზის სიგრძეს. თუ ამ ორ წვეროს შორის გზა არ არის, მაშინ ფუნქციამ (-1) უნდა დააბრუნოს. მეორე ფუნქცია კი უნდა პოულობდეს რამდენი ბმული კომპონენტია მოცემულ გრაფში. მეორე ფუნქციის გაკეთებისას ჩათვალეთ რომ გრაფი სიმეტრიულია, ანუ ყოველთვის როდესაც გვაქვს წიბო i-დან j-მდე, ასევე გვაქვს წიბო j-დან i-მდეც.

interface

```
#ifndef DGRAPH_H_
#define DGRAPH_H_

#include "set.h"

class DGraph {
private:
    struct Vertex {
        Set<Vertex*> neighbours;
    };
    int numVertices;
    int numEdges;
    Vertex* vertices;
    bool hasPath(Vertex* start,
                 Vertex* finish,
                 Set<Vertex*> visited);
public:
    DGraph(int size);
    ~DGraph();
    int size();
    int edgeCount();
    void addEdge(int i, int j);
    void removeEdge(int i, int j);
    bool hasEdge(int i, int j);
    bool hasPath(int i, int j);
    int shortestPath(int i, int j);
    int numComponents();
};

#endif
```

implementation (მხოლოდ გზის სიგრძის პოვნა)

```
#include "DGraph.h"
#include "queue.h"

int DGraph::shortestPath(int i, int j) {
    Queue<Vertex*> yellow;
    Set<Vertex*> visited; // yellow and green combined
    yellow.enqueue(vertices + i);
    visited.insert(vertices + i);
    int distance = 0;
    int nodesLeftToConsider = 1; // at current distance

    while (!yellow.isEmpty()) {
        Vertex* curNode = yellow.dequeue();

        if (curNode == vertices + j) {
            return distance;
        }

        foreach (Vertex* neighbour in curNode -> neighbours) {
            if (!visited.contains(neighbour)) {
                yellow.enqueue(neighbour);
                visited.insert(neighbour);
            }
        }

        nodesLeftToConsider--;
        if (nodesLeftToConsider == 0) {
            distance++;
            nodesLeftToConsider = yellow.size();
        }
    }

    return -1;
}
```

implementation (მხოლოდ კომპონენტების დათვლა)

```
int DGraph::numComponents() {
    int components = 0;
    Set<Vertex*> visited;
    Queue<Vertex*> q;
    for (int i = 0; i < numVertices; i++) {
        if (!visited.contains(vertices + i)) {
            components++;
            q.enqueue(vertices + i);
            visited.insert(vertices + i);
            while (!q.isEmpty()) {
                Vertex* curNode = q.dequeue();
                foreach(Vertex* neighbour in curNode -> neighbours) {
                    if (!visited.contains(neighbour)) {
                        q.enqueue(neighbour);
                        visited.insert(neighbour);
                    }
                }
            }
        }
    }
    return components;
}
```