პროგრამირების აბსტრაქციები
სემინარის ამოცანები #23

გრაფის კლასი და “dfs”-ით გზის პოვნა

დაწერეთ N წვეროიანი, მიმართული გრაფის შესაბამისი კლასი, რომლის წვეროებიც უბრალოდ გადანომრილი იქნება 0-დან $N-1$-მდე. კლიენტი კონსტრუქტორში გადასცემს გრაფის წვეროების რაოდენობა N-ს და ეს რაოდენობა შემდეგ აღარ იცვლება. კლიენტს უნდა შეეძლოს გრაფში მოცემულ ორ წვეროს შორის წიბოს ჩამატება და ამოღება. ასევე, კლიენტს უნდა შეეძლოს შეამოწმოს მოცემულ ორ წვეროს შორის არის თუ არა გზა. ეს შემოწმება “dfs”-ით უნდა მუშაობდეს.

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);
};

#endif
```

implementation (გრაფის შედგენა/შეცვლა)

```
#include "DGraph.h"

DGraph::DGraph(int size) {
    numVertices = size;
    numEdges = 0;
    vertices = new Vertex[numVertices];
}

DGraph::~DGraph() {
    delete[] vertices;
}

int DGraph::size() {
    return numVertices;
}

int DGraph::edgeCount() {
    return numEdges;
}

void DGraph::addEdge(int i, int j) {
    if (!hasEdge(i, j)) {
        vertices[i].neighbours.insert(vertices + j);
        numEdges++;
    }
}

void DGraph::removeEdge(int i, int j) {
    if (hasEdge(i, j)) {
        vertices[i].neighbours.remove(vertices + j);
        numEdges--;
    }
}
```

implementation (გზის პოვნა)

```
bool DGraph::hasEdge(int i, int j) {
    return vertices[i].neighbours.contains(vertices + j);
}

bool DGraph::hasPath(Vertex* start,
                     Vertex* finish,
                     Set<Vertex*> visited) {
    if (start == finish)
        return true;

    visited.insert(start);
    foreach (Vertex* next in start -> neighbours) {
        if (!visited.contains(next)) {
            if (hasPath(next, finish, visited))
                return true;
        }
    }

    return false;
}

bool DGraph::hasPath(int i, int j) {
    Set<Vertex*> visited;
    return hasPath(vertices + i, vertices + j, visited);
}
```