პროგრამირების აბსტრაქციები
სემინარის ამოცანები #11

1 ჰამილტონის ციკლის პოვნა

დაწერეთ ფუნქცია რომელსაც გადაეცემა გრაფი (boolean-ების ცხრილის სახით, რომელშიც i-ური სტრიქონის და j-ური სვეტის გადაკვეთაზე წერია i-ური და j-ური ინდექსის წვეროები შეერთებულია თუ არა), რომლისთვისაც ამ ფუნქციამ უნდა გამოთვალოს არსებობს თუ არა ე.წ. ჰამილტონის ციკლი (ანუ ციკლი რომელიც გრაფის ყველა წვეროს გაივლის ზუსტად ერთხელ). თუ ჰამილტონის ციკლი მოიძებნა, მაშინ ეს ციკლი დამატებით არგუმენტად გადმოცემულ ცვლადში ჩაწერეთ.

```
#include "grid.h"
#include "vector.h"
#include "set.h"
using namespace std;

bool isHamiltonCycle(Grid<bool>& graph, Vector<int> sequence) {
    for (int i = 0; i < sequence.size(); i++) {
        int v1 = sequence[i];
        int v2 = sequence[(i + 1) % sequence.size()];
        if (!graph[v1][v2])
            return false;
    }
    return true;
}

bool hasHamiltonCycleStartingFrom(Grid<bool>& graph,
                                  Vector<int>& chosen,
                                  Set<int>& chosenSet) {
    if (chosen.size() == graph.numCols()) {
        return isHamiltonCycle(graph, chosen);
    }

    for (int newVertex = 0; newVertex < graph.numCols(); newVertex++) {
        if (!chosenSet.contains(newVertex)) {
            chosen.add(newVertex);
            chosenSet.add(newVertex);
            if (hasHamiltonCycleStartingFrom(graph, chosen, chosenSet))
                return true;
            chosen.remove(chosen.size() - 1);
            chosenSet.remove(newVertex);
        }
    }
    return false;
}

bool hasHamiltonCycle(Grid<bool>& graph, Vector<int>& resultCycle) {
    Set<int> emptySet;
    resultCycle.clear();
    return hasHamiltonCycleStartingFrom(graph, resultCycle, emptySet);
}
```

2 გრაფის შეფერადება

დაწერეთ ფუნქცია, რომელსაც გადაეცემა გრაფი და ფერების რაოდენობა და ეს ფუნქცია დაადგენს ამ გრაფის წვეროების ამდენი ფერით გაფერადება თუ არის შესაძლებელი ისე, რომ მეზობელ წვეროებს ერთიდაიგივე ფერი არ ჰქონდეთ. თუ გაფერადება შესაძლებელია, მაშინ გაფერადების ერთ-ერთი ვარიანტი მისამართით გადაცემულ ცვლადში დააბრუნეთ.

```
bool isColoringValid(Grid<bool>& graph,
                     Vector<int>& coloringResult,
                     int newColor) {
    int newVertex = coloringResult.size();

    for (int vertex = 0; vertex < newVertex; vertex++) {
        if (graph[vertex][newVertex] && coloringResult[vertex] == newColor)
            return false;
    }
    return true;
}

bool canBeColored(Grid<bool>& graph,
                  int colorCount,
                  Vector<int>& coloringResult) {
    if (coloringResult.size() == graph.numCols())
        return true;

    for (int newColor = 0; newColor < colorCount; newColor++) {
        if (isColoringValid(graph, coloringResult, newColor)) {
            coloringResult.add(newColor);
            if (canBeColored(graph, colorCount, coloringResult))
                return true;
            coloringResult.remove(coloringResult.size() - 1);
        }
    }
    return false;
}
```