Algorytmy i Struktury Danych II
Skarbonki

Vladyslav Slovinskyi

20 czerwca 2022

Zawartość:

Katalog KeyBox zawiera implementację zadania pt. "Skarbonki":
I. KeyBox:

1)	`Vertex.cpp`	- implementacja klasy, reprezentującej wierzchołek grafu;
2)	`Edge.cpp`	- implementacja klasy, reprezentującej pojedynczą krawędź;
3)	`Visitor.cpp`	- implementacja wizytatora;
4)	`GraphAsMatrix.cpp`	- implementacja grafu za pomocą macierzy sąsiedztwa;

Jak uruchomic programy:

Katalog zawiera program Makefile do kompilacji powyzszych programow, a takze czyszczenia katalogu.

- Aby skompilowac projekt należy wykonać: `$ make`;
- Aby skompilowac projekt oraz wykonać test należy wykonać:
  `$ make test`
  zawartość pliku `data.txt` zostanie wprowadzona do programu `main.x`;
- Aby wyczyścić zawartośc katalogu (usunąć zbędne pliki), należy wykonać:
  `$ make clean`;

Opis problemu i opis rozwiązania

Zadanie ma naturalną interpretację grafową. W celu rozwiązania problemu utwórzono graf skierowany mający n wierzchołków ponumerowanych liczbami od 0 do $n-1$. Wierzchołki o numerach a i b są połączone krawędzią, jesli klucz do skarbonki a znajduje się w skarbonce b. Zauważmy, że z każdego wierzchołka wychodzi dokładnie jedna krawędź; stąd wynika, że w grafie jest dokładnie n krawędzi. Graf, w którym z każdego wierzchołka wychodzi dokładnie jedna krawędź, ma zawsze podobną postac: każda składowa ma jeden cykl, do którego są dołączone drzewa. Oczywiscie rozbicie którejkolwiek skarbonki odpowiadającej

wierzchołkowi w cyklu pozwala otworzyc wszystkie skarbonki odpowiadające wierzchołkom składowej, w której ten cykl się znajduje. Oznacza to, że zadanie sprowadza się do policzenia składowych grafu.

Implementacja algorytmu

Istnieje wiele metod znajdowania liczby składowych grafu. Użyta metoda polega na przeszukiwaniu grafu (w głąb). Wierzchołki, do których można dotrzec z jednego wierzchołka poruszając się po krawędziach w dowolnym kierunku, tworzą jedną składową. Do realizacji zadania została użyta struktura danych pt. graf, zaimplementowana zgodnie z następującym interfejsem:

```
#ifndef GRAPHASMATRIX_H
#define GRAPHASMATRIX_H

#include "Edge.h"
#include "Iterator.hpp"
#include "Visitor.h"

#include <vector>

class GraphAsMatrix
{
public:
    class AllVerticesIter : Iterator<Vertex> {
        GraphAsMatrix& graph;
        int current;

    public:
        AllVerticesIter(GraphAsMatrix& owner);
        ~AllVerticesIter() = default;
        bool is_done();
        Vertex& operator*();
        void operator++();
    };

public:
    class EmanEdgesIter : Iterator<Edge> {
        GraphAsMatrix& owner;
        int row;
        int col;

    public:
        EmanEdgesIter(GraphAsMatrix& owner, int v);
        ~EmanEdgesIter() = default;
        void next();
        bool is_done();
        Edge& operator*();
```

```
37          void operator++();
38      };
39
40  protected:
41      std::vector<Vertex*> vertices;
42      std::vector<std::vector<Edge*>> adjacencyMatrix;
43      bool _isDirected;
44      int _numberOfVertices;
45      int _numberOfEdges;
46
47  public:
48      GraphAsMatrix(int n, bool b);
49      ~GraphAsMatrix();
50      int number_of_vertices() const;
51      bool is_directed();
52      int number_of_edges() const;
53      void add_edge(int u, int v);
54      Edge* select_edge(int u, int v);
55      Vertex* select_vertex(int v);
56      void DFS(Vertex* v);
57      void DFS(Vertex* v, std::vector<bool>& visited);
58      int number_of_money_box(Visitor* visitor);
59      void underlying_graph();
60
61      AllVerticesIter& vertices_iter() {
62          return *(new AllVerticesIter(*this));
63      }
64
65      EmanEdgesIter& eman_edges_iter(int v) {
66          return *(new EmanEdgesIter(*this, v));
67      }
68  };
69
70  #endif
```

Do przeglądania grafu został wykorzystany algorytm DFS. W celu wyznaczenia minimalnej ilości skarbonek, którą należy rozbić wykorzystano metodę `int number_of_money_box(Visitor* visitor)`. Jako argument metoda przyjmuje wizytatora, zliczającego minimalną ilość skarbonek. Za pomocą iteratora przeglądamy wszystkie wierzchołki, wołając algorytm DFS. Ostatecznie metoda zwraca minimalną ilość skarbonek, które należy rozbić.

```
1  #include "GraphAsMatrix.h"
2
3  void GraphAsMatrix::DFS(Vertex* v, std::vector<bool>& visited) {
4      visited[v->get_number()] = true;
5      for (GraphAsMatrix::EmanEdgesIter& it = this->eman_edges_iter(v->
       get_number()));
6          !it.is_done(); ++it) {
```

```
7                 Vertex* vertex = (*it).mate(v);
8             if (!visited[vertex->get_number()]) {
9                 DFS(vertex, visited);
10            }
11        }
12    }
13
14    int GraphAsMatrix::number_of_money_box(Visitor* visitor) {
15        if (_isDirected) underlying_graph();
16        std::vector<bool> visited(this->number_of_vertices(), false);
17        for (GraphAsMatrix::AllVerticesIter& it = this->vertices_iter();
18             !it.is_done(); ++it) {
19            if (!visited[(*it).get_number()]) {
20                DFS(&(*it), visited);
21                visitor->visit();
22            }
23        }
24        return visitor->count();
25    }
```

Złożoność obliczeniowa

Złożoności są następujące:

- inicjalizacja: $O(n^2)$;
- add_edge(v, u): $O(1)$;
- is_edge(v, u) $O(1)$;
- DFS(v): $O(n^2)$;