

Grafy i Sieci. Sprawozdanie 3.

SK11 Kolorowanie grafu za pomocą przeszukiwania z tabu.

Michał Aniserowicz, Jakub Turek

Temat projektu

SK11 Kolorowanie grafu za pomocą przeszukiwania z tabu.

Dokumentacja kodu źródłowego

Kod źródłowy projektu został stworzony w języku Python. Program jest kompatybilny z wersją 2.7.x interpretera. Aplikacja testowana była w Pythonie w wersji 2.7.5, pod kontrolą systemu OS X 10.9 (Mavericks). Do uruchomienia testów jednostkowych wymagane jest zainstalowanie biblioteki Mock¹ w wersji 1.0.1.

Ogólna struktura kodu źródłowego została przedstawiona na poniższym diagramie.

```
/
├── aspiration_criteria
│   └── aspiration_criteria.py
├── evaluation
│   └── cost_evaluator.py
├── graph
│   ├── graph_cloner.py
│   ├── node.py
│   └── node_iterator.py
├── input
│   ├── dimacs_input_reader.py
│   ├── input_reader.py
│   └── input_reader_factory.py
├── memory
│   └── memory.py
├── permutation
│   ├── color_permutator.py
│   └── fast_color_permutator.py
├── progress
│   └── progress_writer.py
├── search
│   └── search_performer.py
├── stop_criteria
│   └── stop_criteria.py
├── test
│   └── ...
├── validation
│   ├── coloring_validator.py
│   └── connection_validator.py
└── main.py
```

¹Biblioteka została wcielona do specyfikacji języka począwszy od wersji 3.3.

Reprezentacja grafu

Graf reprezentowany jest z wykorzystaniem klasy `Node` reprezentującej wierzchołek. Ponieważ, z założenia, aplikacja operuje wyłącznie na grafach spójnych nie ma znaczenia, od którego wierzchołka rozpoczynamy analizę struktury.

```
class Node:
    Id = 0

    def __init__(self, color=None,
                  node_id=None, previous_color=None):

        self.edges = []
        self.color = color

        if node_id is not None:
            self.node_id = node_id
        else:
            self.node_id = Node.Id
            Node.Id += 1

        self.previous_color = self.color

        if previous_color is not None:
            self.previous_color = previous_color

    def add_edges(self, nodes):
        for node in nodes:
            if node not in self.edges:
                self.edges.append(node)

            if self not in node.edges:
                node.edges.append(self)

    def iterator(self):
        return NodeIterator(self)

    def get_node_of_id(self, node_id):
        for node in self.iterator():
            if node.node_id == node_id:
                return node

    def node_count(self):
        return sum(1 for _ in self.iterator())

    def get_colors_count(self):
        colors = set()

        for node in self.iterator():
            colors.add(node.color)

        return len(colors)
```

Metoda `init` służy do konstrukcji węzła. Węzeł posiada następujące składowe:

- `edges` lista wierzchołków połączonych z danym węzłem,
- `color` kolor wierzchołka,
- `node_id` identyfikator wierzchołka,
- `previous_color` poprzedni kolor wierzchołka używany do wyznaczania permutacji.

Identyfikator, jak również kolor wierzchołka, mogą być dowolnego typu (liczba, ciąg znaków...). Identyfikatory mogą, ale nie muszą być nadawane automatycznie - są wtedy typu liczbowego. Kolejne identyfikatory pobierane są ze zmiennej „statycznej” `Id`.

Metoda `add_edges` pozwala na łączenie wierzchołka z innymi wierzchołkami. Implementacja została przygotowana dla grafów nieskierowanych, a więc podczas dodawania krawędzi tworzone jest od razu wiązanie dwustronne.

Do poruszania się po grafie wykorzystywany jest iterator, który korzysta z algorytmu DFS.

Metoda `get_node_of_id` pozwala na dojście do dowolnego wierzchołka po identyfikatorze.

Metoda `node_count` zlicza liczbę wierzchołków w grafie.

Metoda `get_colors_count` zwraca liczbę kolorów, którymi w chwili obecnej pokolorowany jest graf.

Klasa `NodeIterator` dostarcza interfejs iteratora dla wierzchołka grafu. Udostępnia ona metodę `next`, która dla danego wierzchołka zwraca kolejny w porządku przeszukiwania w głąb. Przeszukiwanie w głąb

oznacza, że w pierwszej kolejności przechodzimy do pierwszego dziecka danego wierzchołka, a dopiero po powrocie algorytmu do tego samego wierzchołka przeglądamy jego kolejne dziecko. Wykorzystanie wzorca iteratora pozwala na przeglądanie grafu w wygodny sposób - używając do tego pętli `for`.

Oprócz narzędzia do przeglądania grafu zaimplementowana została też metoda do kopiowania całego grafu. Jest ona zawarta w metodzie `clone` klasy `GraphCloner`. Klonowanie grafu jest przydatne podczas wyznaczania możliwych permutacji kolorów. Wystarczy powielić cały graf i zmienić barwę analizowanego wierzchołka.