Optymalizacja wydajności wybranych algorytmów grafowych.

Łukasz Marcinkowski Jacek Sosnowski

9 kwietnia 2015

1 Opis zadania

Zadaniem projektu jest porównanie dwóch implementacji algorytmu poszukiwania cyklu Eulera. Przedstawione zostaną dwie realizacje, jedna w języku Java oraz druga w C++. Obie zostaną przetestowane na tych samych grafach. Ostatecznie obie implementacje będą ocenione pod względem wydajności.

2 Algorytm

Dla ustalenia uwagi cykl Eulera jest rozumiany zgodnie z definicją 2.1.

Definicja 2.1 ***Cyklem eulera*** *nazywa się drogę zamkniętą przechodzącą dokładnie jeden raz przez każdą krawędź grafu (zorientowanego bądź niezorientowanego).*

Graf posiadający cykl Eulera nazywany jest grafem Eulera. Definicja 2.2 jest kryterium, które jednoznacznie rozstrzyga, czy zadany graf niezorientowany posiada cykl Eulera, czy nie. Kryterium dla grafów zorientowanych jest definicja 2.3. Oba twierdzenia są udowodnione między innymi w pracy [1].

Definicja 2.2 *Spójny graf niezorientowany jest grafem Eulera wtedy i tylko wtedy, gdy wszystkie jego wierzchołki są parzystego stopnia.*

Definicja 2.3 *Spójny graf zorientowany jest grafem Eulera wtedy i tylko wtedy, gdy dla każdego wierzchołka liczba krawędzi wchodzących jest równa liczbie krawędzi wychodzących - zachodzi równanie:*

$$d^+(v) = d^-(v), v \in V$$

Algorytm poszukujący cyklu Eulera opiera się na twierdzeniach 2.2 i 2.3 w celu wykluczenia z analizy grafów, co do których jest pewność, że takiego cyklu zawierać nie mogą. Dodatkowo wymuszają one wprowadzenie założenia, że analizowane struktury będą spójne. Dla grafów niespójnych, każdą składową spójności można rozpatrywać oddzielnie.

Do znalezienia cyklu Eulera w zadanym grafie w obu implementacjach zostanie zastosowany algorytm Fleury'ego, który prezentuje listing 1. Jego działanie opiera się na wyborze wierzchołka startowego, a następnie budowie ścieżki przez wszystkie krawędzie w grafie, aż do jej zamknięcia w punkcie startowym. W każdym kroku budowy ścieżki, preferowane są te krawędzie, które nie są mostkami (unikanie rozspójnienia grafu). Realizacja algorytmu Fleury'ego będzie opierać się o wykorzystanie stosu, który łagodzi konieczność sprawdzania czy krawędź jest mostkiem, co jednocześnie zmniejsza złożoność algorytmu.

Algorytm 1 Algorytm Fleury'ego dla grafu G.

Require: Graf powinien spełniać kryterium z definicji 2.2 i 2.3.

```
STOS ← ∅
u ← dowolny wierzchołek grafu G.
EULER ← u

repeat
    if istnieje krawędź wychodząca z u then
        STOS ← u
        v ← dowolny wierzchołek połączony z u krawędzią (u, v)
        usuń z grafu krawędź (u, v)
        u ← v
    else
        u ← STOS
        EULER ← u
    end if
until STOS ≠ ∅

return EULER                ▷ Zawiera listę wierzchołków w cyklu Eulera
```

2.1 Dowód poprawności algorytmu

Algorytm 1 oparty jest na przechodzeniu pomiędzy wierzchołkami używając za każdym razem jednej krawędzi wchodzącej i jednej wychodzącej z danego

wierzchołka. Krawędź raz wybrana zostaje usunięta, dlatego ta procedura zapewnia, że żadna droga nie będzie użyta więcej niż raz.

Cała procedura oparta jest o pętlę, dlatego dla kontynuacji dowodu poprawności wybrany został następujący niezmiennik pętli:
Przed każdym obiegiem pętli, graf G jest podgrafem grafu wejściowego (w szczególności może nie być spójny), lista EULER zawiera fragment cyklu Eulera oraz STOS zawiera wierzchołki aktualnie rozpatrywanej pewnej drogi zamkniętej (nie musi to być cały poszukiwany cykl Eulera).

Na zakończenie powyższy niezmiennik przybiera postać:
Graf G jest podgrafem grafu wejściowego pozbawionym krawędzi, lista EULER zawiera cykl Eulera, STOS jest pusty.

Tuż przed pierwszą iteracją niezmiennik jest spełniony, bo graf G jest niewłaściwym podgrafem, lista EULER zawiera drogę otwartą rozpoczynającą się w wierzchołku startowym s, a STOS jest pusty.

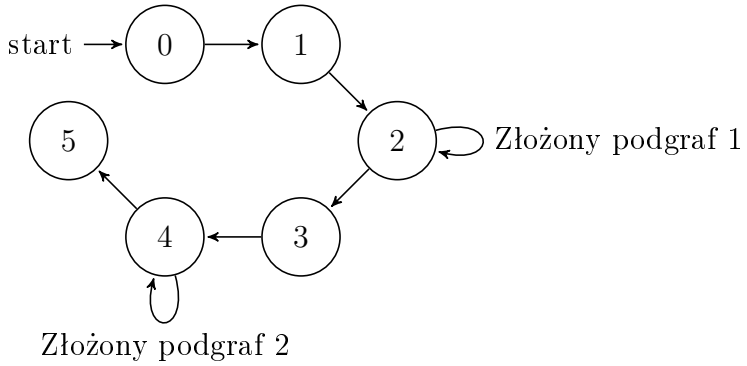
Każda kolejna iteracja poddaje analizie pewien wierzchołek u – jest on dodawany do stosu. Dzięki temu, stos zawiera "ślad" pewnej drogi w grafie. W celu wyznaczenia kolejnego wierzchołka do analizy wybierana jest dowolna krawędź wychodząca z u, która następnie zostaje usunięta. Ten krok zapewnia, że w każdej iteracji "znika" jedna krawędź, co **spełnia początkową część niezmiennika pętli mówiącą, że graf G jest podgrafem**. Dodatkowo zapewnia, że algorytm **zatrzyma się po przejrzeniu wszystkich krawędzi** - spełniona część niezmiennika końcowego.

W przypadku kiedy analizowany wierzchołek u nie posiada krawędzi wychodzących, nie jest dodawany ani do stosu ani do listy. Zgodnie z wnioskiem 2.4 jest on wierzchołkiem startowym s.

Wniosek 2.4 *W grafie spełniającym kryteria 2.2 i 2.3 usuwając kolejno odwiedzane krawędzie (na drodze rozpoczętej w ustalonym wierzchołku s) jedyna możliwa sytuacja kiedy nie można wybrać kolejnej krawędzi wyjściowej może wydarzyć się jedynie dla wierzchołka od którego rozpoczęła się dana droga.*

Ponowne osiągnięcie wierzchołka początkowego nie gwarantuje odkrycia cyklu Eulera, ale **utrzymuje dla każdej iteracji część niezmiennika mówiącą o tym, że STOS zawiera wierzchołki aktualnie rozpatrywanej drogi zamkniętej** (ale bez jej zamykania).

W tej sytuacji do następnej iteracji brany jest pierwszy wierzchołek ze stosu – sprawdzane jest, czy on posiada krawędzie wychodzące. Ale jest dodawany do cyklu Eulera. Oznacza to, że krawędź od wierzchołka startowego do wierzchołka zdjętego ze stosu będzie już elementem wynikowej drogi. Dzieje się tak, ponieważ pewne jest iż ta krawędź razem z krawędziami wyznaczonymi przez wierzchołki przechowywane na stosie tworzy zamkniętą drogę. Ta droga wzbogacona o cykle zaczynające i kończące się w poszczególnych



Rysunek 1: Ilustracja drogi zamkniętej reprezentowanej przez STOS oraz kilku podgrafów stanowiących drogi rozpoczynające się i kończące w tym samym wierzchołków – symbolizowane pętlami własnymi.

wierzchołkach ze stosu będzie wynikowym cyklem Eulera. To udowadnia, że **w każdej iteracji lista EULER opisuje fragment wynikowego cyklu Eulera**.

Przykładową sytuację pokazuje rysunek 2.1, który pokazuje aktualnie analizowaną drogę – ze stosem: { 0 1 2 3 4 5 }. Krawędź $(5, 0)$ również istnieje, ale nie zostaje uwzględniona na stosie lecz pozostaje od razu dodana do budowanego cyklu Eulera. Wynikowy cykl będzie zawierał wszystkie krawędzie widoczne na rysunku 2.1 oraz cykle odkryte w *Złożonym grafie 1* i *Złożonym grafie 2* (zachowując odpowiednią kolejność).

Ogólnie należy zauważyć, że wierzchołki są zdejmowane ze stosu i dodawane do reprezentacji cyklu Eulera, a za nimi cykl zaczynający i kończący się w każdym z nich. Dzięki temu spełnione jest warunek końcowy, który zakładał, że lista EULER będzie zawierać wierzchołki reprezentujące cykl Eulera, a STOS będzie pusty.

2.2 Złożoność algorytmu

Każda krawędź grafu jest rozpatrywana przez algorytm wyłącznie raz. Z kolei podczas każdego obiegu pętli rozpatrywana jest i usuwana jedna krawędź. Jeśli założy się, że operacje wykonywane podczas każdej iteracji mają koszt jednostkowy, to cały algorytm Fleury'ego ma złożoność O(q). Gdzie q jest liczbą krawędzi w grafie. Założenie kosztu jednostkowego pojedynczej iteracji jest słuszne z uwagi, że wykonywane operacje to dodanie/usunięcie ze stosu lub/i usunięcie krawędzi, które przy użytych strukturach danych mają koszty jednostkowe.

[directed|undirected] [liczba wierzchołków]

0 : 1 2 3

1 : 2 3 4

3 : 4

Rysunek 2: Format pliku przechowujący graf.

3 Implementacja

Zgodnie z założeniami projektu, algorytm Fleury'ego będzie dostarczony w postaci dwóch aplikacji stworzonych w językach Java i C++. Aplikacja testująca wydajność będzie odrębnym modułem.

Z różnych reprezentacji grafów dostępnych w książce [2] został wybrany format wierzchołkowy. Struktura grafu będzie więc reprezentowana za pomocą list wierzchołków powiązanych krawędziami, a nie listy krawędzi (jak w konkurencyjnej reprezentacji). To też wpływa na format danych wejściowych – zaprezentowany na listingu 3.1. Dane wejściowe mają postać plików.

3.1 Format danych wejściowych – grafów

Pierwsza linia ma wyłącznie charakter specyfikacji i stanowi informacje nadmiarowe. Zawiera pojedyncze słowo informujące czy graf przechowywany w pliku jest skierowany (ang. *directed*) lub nieskierowany (ang. *undirected*). Dalej pojawia się liczba wierzchołków w grafie.

Kolejne linie są już bezpośrednią reprezentacją grafu. Każda jest rozdzielona znakiem dwukropka na dwie części: pierwsza specyfikuje nazwę wierzchołka, druga jest listą wierzchołków z którymi jest połączony dany wierzchołek. W ramach tego projektu nazwy będą liczbami dziesiętnymi.

Obie informacje z linii specyfikacji (pierwszej linii) można wywnioskować na podstawie analizy reszty pliku. Liczba wierzchołków to po prostu liczba linii zawierających opis grafu. Natomiast typ grafu można wydedukować zauważając, że graf nieskierowany, który zawiera krawędź (u, v) musi również zawierać krawędź (v, u). Mimo to linia specyfikacji znacząco ułatwia etap analizy syntaktycznej pliku wejściowego.

4 Testowanie

Działanie aplikacji zostanie sprawdzone na podstawie dwóch zestawów danych. Pierwszy jest zestawem przeznaczonym dla testów akceptacyjnych. Zadaniem drugiego będzie sprawdzenie wydajności aplikacji.

4.1 Projekt testów akceptacyjnych

Zestaw tych testów składa się z kilkunastu niewielkich grafów w dwóch wariantach. Pierwszy wariant to grafy dla których analitycznie udowodniono istnienie cykli Eulera (spełniają warunek konieczny). Przedstawia je tablica 1. Natomiast drugi wariant składa się z grafów, które nie mogą takiego cyklu zawierać. Odpowiadają one pierwszemu wariantowi z losową modyfikacją jednej krawędzi grafu z zastrzeżeniem zachowania spójności. Dodanie krawędzi bądź usunięcie zaburza warunek konieczny istnienia cyklu Eulera.

4.2 Generacja testów wydajnościowych

Zestaw danych dla testów wydajnościowych będzie pochodzić z generatora grafów eulerowskich. Sam generator też jest elementem projektu. Jego działanie opierać się będzie na tworzeniu grafów o zadanej liczbie wierzchołków poprzez losowe rozmieszczenie krawędzi pomiędzy nimi. Liczbę krawędzi kontroluje parametr wypełnienia (gęstości), który jest równy proporcji liczby krawędzi w grafie do potencjalnej liczby wszystkich możliwych krawędzi. I tak dla grafu nieskierowanego jest to:

$$wypelnienie = \frac{liczbaKrawedzi}{\frac{N(N-1)}{2}}$$

dla grafu skierowanego:

$$wypelnienie = \frac{liczbaKrawedzi}{N(N-1)}$$

gdzie, N to liczba wierzchołków w grafie.

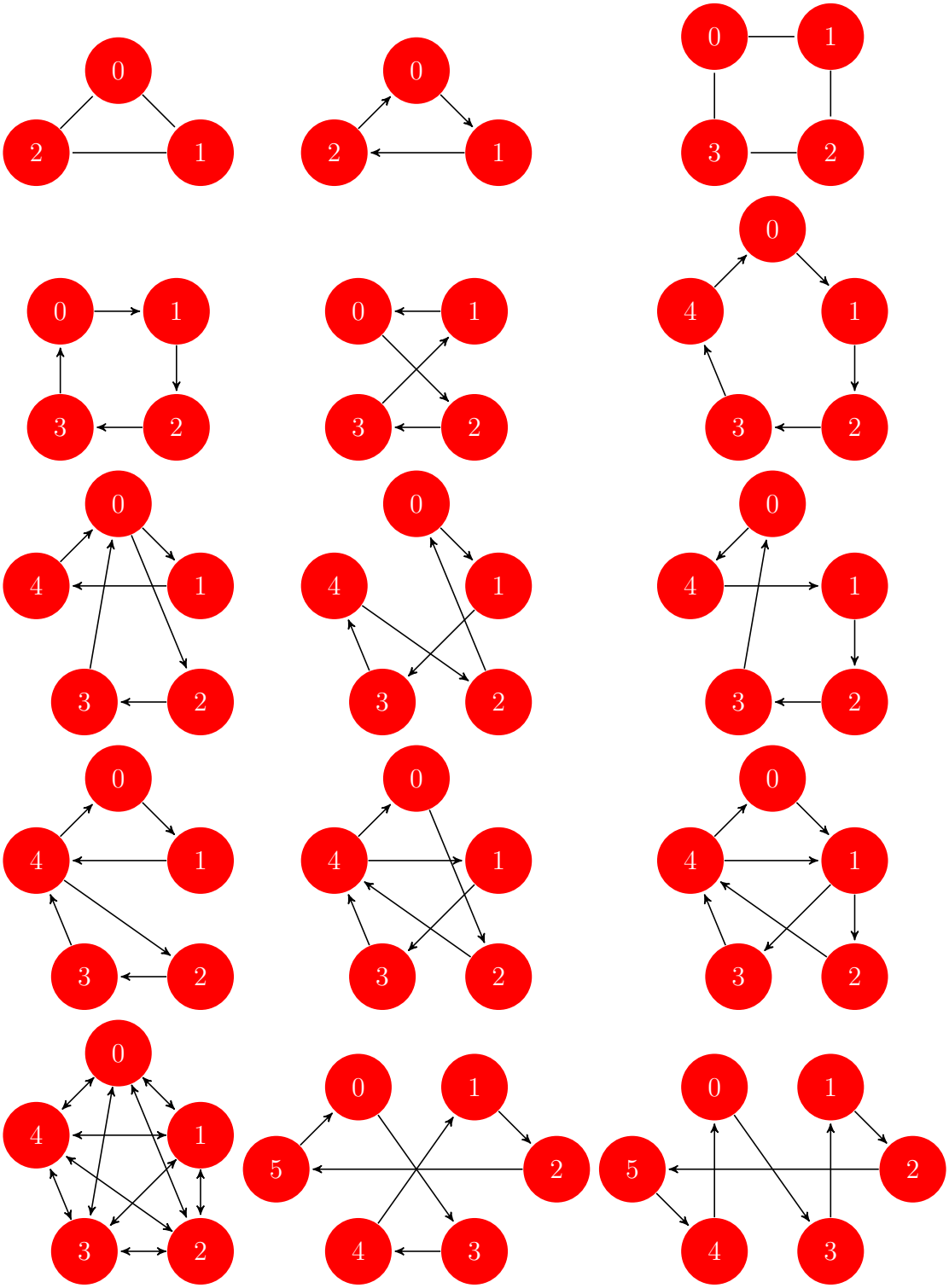
Dzięki zdefiniowaniu parametru wypełnienia, nie ma potrzeby podawania konkretnej liczby krawędzi podczas generowania nowego grafu. Przykładowo wystarczy informacja, że powinien on być wypełniony co najmniej w 50%.

Projekt testów zakłada utworzenie zestawu grafów o liczbie wierzchołków wahającej się od 5 do około 10000. Wypełnienie będzie zmienne. Ale raczej ze wskazaniem na struktury bardziej zbliżone liczbą krawędzi do grafów pełnych. Powodem tego jest fakt, iż grafy o niewielkim wypełnieniu nie stanowią

wyzwania dla omawianego algorytmu, którego złożoność jest wprost proporcjonalna do liczby krawędzi.

Każdy test będzie powtarzany ustaloną liczbę razy w celu wyznaczenia parametrów statystycznych czasu wykonania (między innymi: czas średni, mediana, odchylenie standardowe).

Tablica 1: Grafy zawierające cykl Eulera.



Literatura

[1] J. Wojciechowski, K. Pieńkosz, *Grafy i sieci*, Wydawnictwo Naukowe PWN, 2013

[2] T.H. Cormen, C.E. Leiserson, R.L. Rivest, *Wprowadzenie do algorytmów*, Wydawnictwo Naukowo-Techniczne, Warszawa 2001.