Wyznaczanie w grafie najkrótszej ścieżki z ograniczoną liczbą krawędzi

Łukasz Korpal
Wiktor Ślęczka

7 czerwca 2016

Spis treści

1 Realizowane zadanie

Dany jest spójny graf nieskierowany z określonymi długościami krawędzi i dwoma wyróżnionymi wierzchołkami. Problem polega na znalezieniu najkrótszej ścieżki między tymi wierzchołkami, składającej się z co najwyżej k krawędzi. W ramach projektu należy opracować algorytm i program do wyznaczania takiej ścieżki dla zadanego parametru k.

2 Interfejs programu

Utworzony w ramach projektu program należy uruchomić poprzez wydanie następującego polecenia:

```
gis <graph_description_file> <start> <target>
```

Spowoduje to wczytanie podanego pliku, uruchomienie dla zapisanego w jego wnętrzu grafu algorytmu znajdowania ścieżki i wypisania znalezionego rozwiązania, lub komunikatu o napotkanym problemie. Dodatkowo muszą zostać podane wierzchołki startowy i końcowy oraz parametr określający maksymalną liczbę krawędzi, z których składać ma się ścieżka.

3 Założenia programu

Założenia dotyczą danych wejściowych. Są one następujące:

- Wierzchołek początkowy i końcowy muszą należeć do grafu
- Wierzchołek początkowy i końcowy każdej krawędzi są różne
- Minimalna liczba skoków jest całkowita dodatnia
- Podawany graf powinien być spójny i niekierowany
- Każda krawędź powinna mieć wagę
- Każdy wierzchołek powinien mieć unikalną nazwę
- Krawędź łączy zdefiniowane uprzednio wierzchołki
- Wagi krawędzi muszą mieć wartości dodatnie całkowite
- Dla metody BFS między dwoma dowolnymi wierzchołkami nie powinno być więcej niż jednej krawędzi
- Pliki wejściowe są stworzone zgodnie z podaną specyfikacją

4 Wejście

Wejściem programu jest pojedynczy plik tekstowy, zawierający opis grafu. Na jego początku znajduje się lista wierzchołków, oddzielonych znakiem nowej linii. W nazwie wierzchołka mogą znaleźć się dowolne znaki nie będące znakami białymi. Pierwsza linia pliku zawiera informacje o wielkości grafu:

```
<liczba_wierzchołków> <liczba_krawedzi>
```

Następnie podawana jest lista krawędzi, gdzie krawędź jest zdefiniowana jako

```
<nazwa_wierzchołka_1> <nazwa_wierzchołka_2> <waga_krawedzi>
```

definiująca po jednej krawędzi w każdej kolejnej linii. Na końcu pliku występuje pojedyncza pusta linia.

5 Wyjście

Wyjściem programu jest lista wierzchołków, na które składa się znaleziona ścieżka, wraz z jej obliczoną długością. Wierzchołki rozpoznawane są poprzez nazwy podane w pliku opisującym graf.

6 Technologie

6.1 Projekt

Projekt jest realizowany przy użyciu języka Scala w wersji 2.11.8. W chwili obecnej nie są wykorzystywane żadne dodatkowe biblioteki ani innego rodzaju zależności. Do kompilacji programu używane jest narzędzie SBT.

6.2 Testy

Grafy tworzone są na potrzeby testów przy użyciu skryptu języka Python, korzystającego z biblioteki GraphViz, służącej ich wizualizacji.

7 Struktury danych

W programie będą wykorzystywane dwie struktury danych: jedna odpowiedzialna za przechowywanie grafu wczytanego z pliku tekstowego oraz druga, przechowująca aktualny stan procesu przeszukiwania grafu. Klasa opisująca obiekt wczytanego grafu będzie składała się z dwóch list, reprezentujących wierzchołki i krawędzie. Instancja wierzchołka pozwala na sprawdzenie krawędzi, które z niego wychodzą, natomiast krawędzi sprawdza, czy wierzchołek jest do niej przyłączony. Konstruktor pozwala na utworzenie pustego grafu, który będzie zapełniany w miarę odczytywania informacji z pliku w trakcie inicjalizacji programu.

Aktualny stan procesu przeszukiwania grafu reprezentowany będzie przez mapę przypisującą do każdego wierzchołka odległość od punktu startowego oraz inne, niezbędne do przeszukiwania informacje, pozwalające odtworzyć ścieżkę prowadzącą do opisywanego wierzchołka. Mapa realizowana będzie poprzez wbudowany typ mapy.

8 Wykorzystane algorytmy

Algorytm wykorzystywany do odnajdywania najlżejszej ścieżki w grafie będzie algorytm bazujący na algorytmie DFS. Jego działanie opiera się na sprawdzaniu wierzchołków o najkrótszej długości drogi (do maksymalnej liczby skoków) do momentu dotarcia do wierzchołka docelowego. W przypadku sprawdzania wierzchołków oddalonych o więcej niż maksymalną liczbę skoków, algorytm pomija takie rozwiązanie i wyszukuje kolejnych możliwości poprawy ścieżki. W przypadku, gdy podczas tworzenia grafu wykryte zostanie, że wszystkie krawędzie są takiej samej długości, zastosowany zostanie inny algorytm, przyspieszający rozwiązanie zadania. Algorytmem tym będzie Breadth First Search.

8.1 Algorytm przedstawiony w pseudokodzie

```
DFS(from, to, maxjumps, distance=0):
  if maxjumps == 0:
    return routes
  for each neighbour from.neighbours:
    DFS(neighbour, to, maxjumps-1, distance+dist(from, neighbour))
  if from == to && (routes is empty || routes.distance > distance):
    routes.change(from,distance)
```

9 Testy

Testy programu zostały przeprowadzone zarówno dla grafów tworzonych losowo, jak i specjalnie przygotowanych przypadków szczególnych.

9.1 Przypadki szczególne

9.2 Grafy tworzone losowo

Grafy tworzone były za pomocą skryptu *creator_viz.py*

10 Wyniki

Testy przeprowadzano na komputerze z procesorem AMD Phenom(tm) II X6 1090T. Około 0.4 sekundy z podanych czasów jest pochłaniane na uruchamianie wirtualnej maszyny Java.
Dla podanych przykładów wyniki działania programu, poszukującego ścieżki - dla ograniczenia do odległości 5 krawędzi - były następujące:

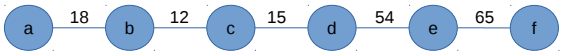
- 9.1 a

```
Distance: 99
a
b
c
d
e
```

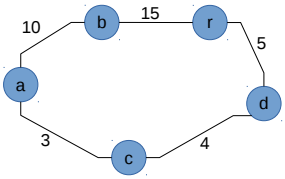
Czas wykonania: 0m0.399s

- 9.1 b

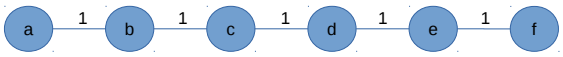
```
Distance: 12
```



(a) Przypadek tworzący linię prostą.

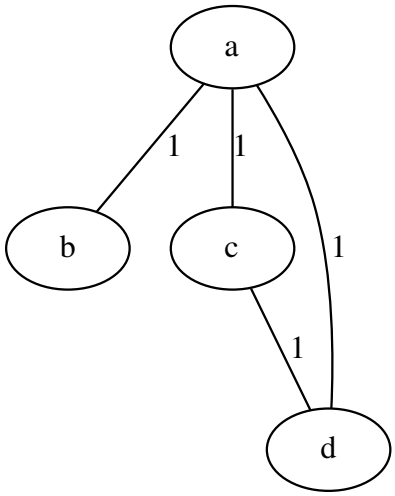


(b) Przypadek sprawdzający poprawne działanie wyboru najkrótszej drogi.

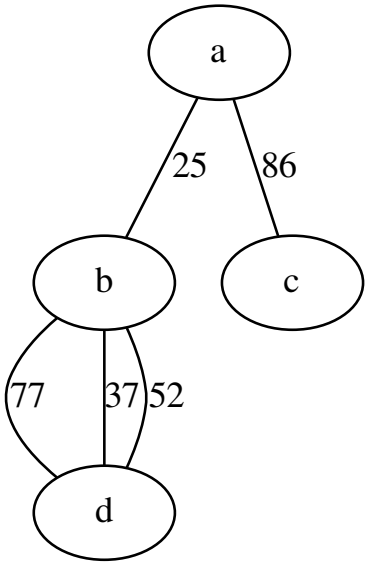


(c) [Przypadek tworzący linię prostą, uruchamiający przeszukiwanie BFS.

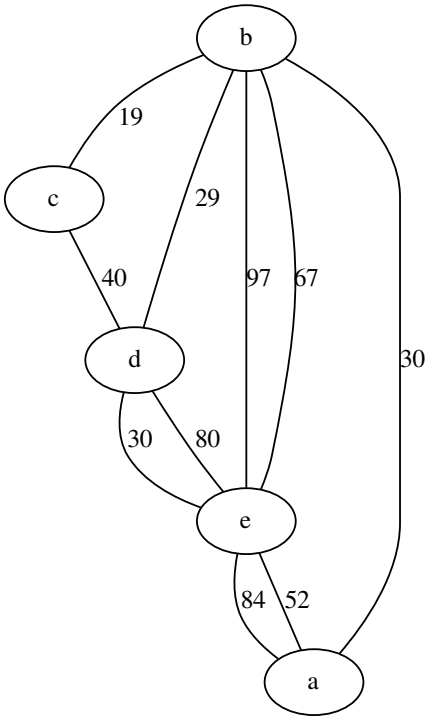
Rysunek 1: Przypadki szczególne



(a) Przypadek, w którym graf uruchamia metodę BFS.

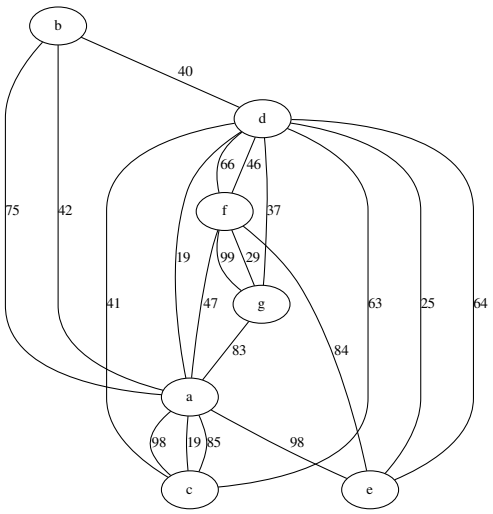


(b) Przypadek dla 4 wierzchołków i 5 krawędzi.

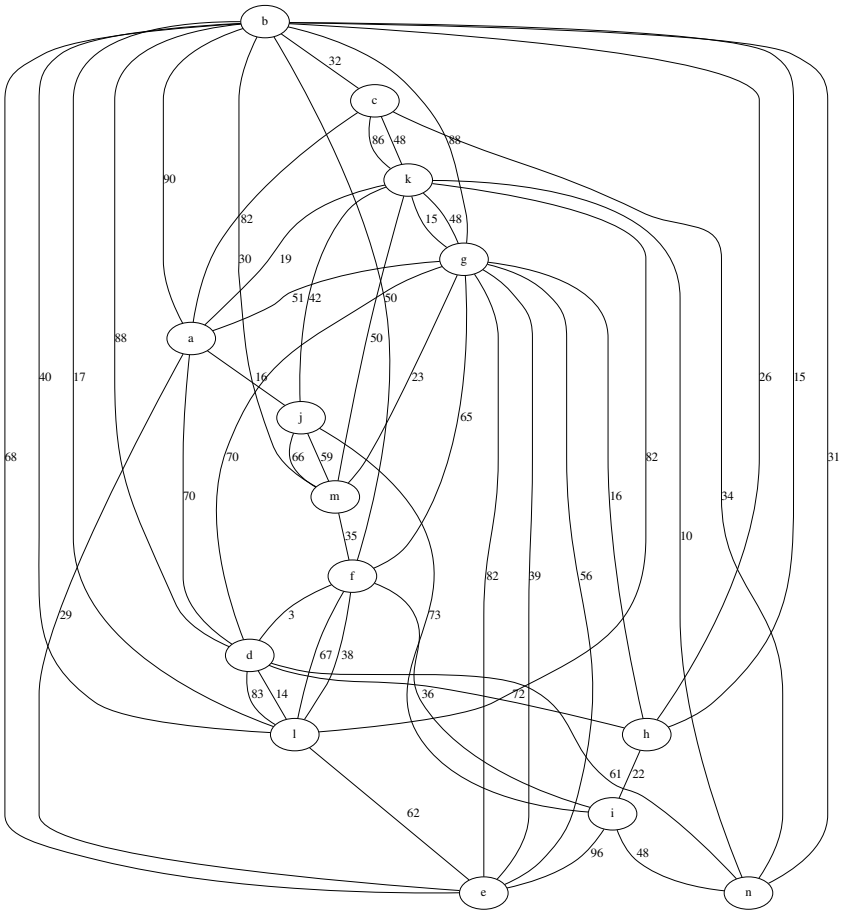


(c) Przypadek dla 5 wierzchołków i 10 krawędzi.

Rysunek 2: Grafy tworzone losowo - część 1



(a) Przypadek dla 7 wierzchołków i 20 krawędzi.



(b) Przypadek dla 14 wierzchołków i 50 krawędzi.

Rysunek 3: Grafy tworzone losowo - część 2

```
a
c
d
e
```

Czas wykonania: 0m0.383s

- 9.1 c

```
Distance: 5
a
b
c
d
e
f
```

Czas wykonania: 0m0.659s

- 9.2 a

```
Distance: 2
c
a
b
```

Czas wykonania: 0m0.544s

- 9.2 b

```
Distance: 86
a
c
```

Czas wykonania: 0m0.354s

- 9.2 c

```
Distance: 49
a
b
c
```

Czas wykonania: 0m0.375s

- 9.2 a

```
Distance: 56
a
d
g
```

Czas wykonania: 0m0.433s

- 9.2 b

```
Distance: 50
a
k
g
h
```

Czas wykonania: 0m0.580s

Dla największego testowanego grafu (12000 wierzchołków i 50000 krawędzi)

```
Distance: 265
ax
oxa
qku
coh
bun
abs
```

Czas wykonania dla maksymalnej liczby skoków:
6 0m32.022s
7 0m43.391s
8 1m22.649s

Dla innego wygenerowanego grafu zawierającego tyle samo krawędzi i wierzchołków, jednak rozpatrywanego algorytmem BFS (tworzone wagi krawędzi są równe 1) wynik jest następujący:

```
Distance: 41
ax
hyr
pxm
b
ihe
bum
ojk
dfz
puh
dwj
csn
mef
acz
jbv
pkb
itu
dqe
gsq
fft
erk
gdo
ivw
cqz
qfv
cey
dmo
apc
pyh
mg
ckz
nfq
jbg
jpf
frk
rx
avh
axr
cna
dxw
jko
uw
abs
```

Czas wykonania: 0m2.744s

11 Wnioski

Oryginalny algorytm DFS, zmodyfikowany o skracanie poszukiwań do maksymalnej odległości (wyrażonej w skokach) sprawdza się bardzo dobrze dla ilości wierzchołków poniżej rzędu wielkości $2 \cdot 10^4$ i ilości krawędzi poniżej 10^5. Dla wartości liczby wierzchołków zbliżających się do tej granicy czas oczekiwania na wynik działania programu trwa ponad 3 minuty (przy pełnym użyciu procesora Intel Core i5 o taktowaniu 2.4 GHz).
Porównanie działania algorytmów BFS i zmodyfikowanego algorytmu DFS wyraźnie pokazują, że dla dużych grafów algorytm BFS działa nieporównywalnie sprawniej.