Zadanie zaliczeniowe CPP - algorytm Brandesa

Spis treści

Termin zadania: 9 stycznia 2017

Modyfikacje treści:

- 2016/12/26 FAQ
- 2016/12/23 FAQ
- 2016/12/13 FAQ
- 2016/12/09 ogłoszenie zadania

1 Wprowadzenie

Analiza sieci polega m.in. na identyfikacji ważnych wierzchołków. Naiwnym sposobem wyrażania ważności wierzchołka jest np. zliczenie sąsiadów—im więcej sąsiadów tym ważniejszy jest wierzchołek dla sieci. Liczenie sąsiadów nie identyfikuje jednak ważnych z punktu widzenia spójności sieci wierzchołków-pośredników (np. wierzchołka łączącego dwie grupy niepołączonych ze sobą wierzchołków). Zaproponowano wiele alternatywnych

miar, m.in. closeness centrality czy eigenvector centrality; tematem zadania jest obliczenie pośrednictwa (betweenness).

Pośrednictwo węzła v określa jaka część najkrótszych ścieżek w sieci przechodzi przez ten węzeł: $BC(v) = \sum_{s \neq v \neq t} \frac{\sigma_{st}(v)}{\sigma_{st}}$, gdzie σ_{st} jest liczbą najkrótszych ścieżek między węzłami s i t; a $\sigma_{st}(v)$ jest liczbą najkrótszych ścieżek między węzłami s i t przechodzącymi przez v.

W sieciach opisanych grafami ważonymi obliczenie pośrednictwa wymaga obliczenia najkrótszych ścieżek między wszystkimi parami węzłów (np. algorytmem Floyda-Warshalla), z kosztem $O(|V|^3)$. Na szczęście, wiele ciekawych sieci nie jest ważonych—do takich sieci można stosować algorytm Brandesa ($O(|V||E|)$).

Prosimy o zaimplementowanie algorytmu Brandesa w C++ 14.

2 Algorytm Brandesa

Pseudokod współbieżnego algorytmu Brandesa przedstawiamy poniżej; dalsze wyjaśnienia w artykułach z bibliografii. Algorytm używa następujących zmiennych: `BC`: pośrednictwo; `V`: wierzchołki; `d[w]` odległość do wierzchołka w; `sigma[w]` liczba najkrótszych ścieżek do wierzchołka w; `P[w]` poprzednicy wierzchołka `w` na wszystkich najkrótszych ścieżkach; `delta[v]` wartość pośrednictwa dla `v` w ścieżkach startujących z `s`.

```
for each v : V
   BC[v] = 0;
for each s : V { // in parallel
   S = stack();
   for all w in V {
      P[w] = list();
```

```
   sigma[w] = 0;
   d[w] = -1;
   delta[v] = 0;
}
sigma[s] = 1;
d[s] = 0;
Q = queue(); // FIFO
Q.push_back(s);
while (!Q.empty()) {
   v = Q.pop_front();
   S.push(v);
   for each neighbor w of v {
      if d[w] < 0 {
         Q.push_back(w);
         d[w] = d[v] + 1;
      }
      if (d[w] == d[v] + 1) {
         sigma[w] += sigma[v];
         P[w].append(v);
```

```
        }
      }
    }
    while (!S.empty()) {
      w = S.pop();
      for each v in P[w]
         delta[v] += (sigma[v] / sigma[w])(1 + delta[w]);
      if (w != s)
         BC[w] += delta[w];
    }
}
```

3 Wejście i wyjście

Programy będą testowane automatycznie. Prosimy o ścisłe przestrzeganie podanych poniżej: formatowania nazw plików oraz wejścia i wyjścia programów.

3.1 Sposób uruchomienia programu

```
unzip xx123456; cd xx123456; mkdir build; cd build; cmake ..; make
```

```
./brandes liczba-wątków plik-wejsciowy plik-wyjsciowy
```

3.2 Wejście

Plik wejściowy opisuje krawędzie w grafie w formacie `<wierzchołek1> <wierzchołek2>`. Graf jest skierowany. Krawędzie są posortowane po pierwszym wierzchołku i po drugim wierzchołku. Wierzchołki nie muszą być numerowane sekwencyjnie (tzn. poprawny jest graf bez np. wierzchołka o numerze 0). Graf nie musi być spójny.

Program powinien przeprowadzać obliczenia używając `liczba-wątków` wątków. W testach wydajnościowych będzie zdecydowanie więcej wierzchołków niż wątków, ale musisz zapewnić poprawne działanie programu dla `liczba-wątków < 100` niezależnie od liczby wierzchołków.

Przykładowe wejście:

```
0 2
2 0
2 3
2 4
3 2
3 5
3 6
```

3.3 Wyjście

Plik wyjściowy składa się z dokładnie tylu linii, ile jest wierzchołków z co najmniej jedną krawędzią wychodzącą. *i*-ta linia zawiera dwie liczby: liczbę całkowitą — identyfikator wierzchołka, oraz liczbę rzeczywistą — pośrednictwo tego wierzchołka. Wyjście musi być posortowane po pierwszej kolumnie.

Standardowe wyjście programu będzie ignorowane.

Plik wyjściowy odpowiadający przykładowemu plikowi wejściowemu:

```
0 0
2 6
3 4
```

4 Forma oddania zadania

Prosimy o oddanie pojedynczego pliku `.zip` zawierającego pojedynczy katalog odpowiadający loginowi (`ab123456`), a w nim następujące pliki:

- kod źródłowy;
- CMakeLists.txt;
- raport w pliku raport.pdf

W raporcie opisz twoją implementację i zastosowane optymalizacje. Dla grafu `wiki-vote-sort.txt` zmierz speed-up, czyli zmierz czas działania twojego programu p_m dla $m = 1, 2, \ldots, 8$ wątków, a następnie oblicz przyspieszenie: $S_m = p_1/p_m$.

CMakeLists powinien kompilować program do jak najszybszej wersji: pamiętaj o włączeniu optymalizacji kompilatora i wyłączeniu debugowania.

5 Kryteria oceny

Będziemy oceniać poprawność, wydajność oraz raport.

Poprawność będziemy oceniać porównując wyniki implementacji z naszym rozwiązaniem wzorcowym; nasze testy biorą pod uwagę błędy zaokrągleń. Udostępniamy dwa przykładowe testy, ale nie wyczerpują one wszystkich przypadków brzegowych — pamiętaj o dodaniu własnych testów. Nie będziemy oceniać wydajności programów

niepoprawnych.

Szybkość działania będziemy oceniać porównując czas działania waszych implementacji na sieciach "społecznych", tzn. nie Erdős'a–Rényi'ego. Pojedyncze wykonanie programu będziemy ograniczać do kilku (max. 5) minut.

6 Dodatkowe materiały

Prosimy o niekorzystanie z kodów źródłowych - gotowych implementacji algorytmu Brandesa. Zalecamy natomiast przeczytanie następujących prac:

- A faster algorithm for betweenness centrality, U Brandes, Journal of Mathematical Sociology, 2001 Taylor & Francis, https://kops.uni-konstanz.de/bitstream/handle/123456789/5739/algorithm.pdf?sequence=1
- Parallel algorithms for evaluating centrality indices in real-world networks DA Bader, K Madduri, https://smartech.gatech.edu/bitstream/handle/1853/14428/GT-CSE-06-13.pdf
- Dane do testów wydajnościowych można wziąć z http://snap.stanford.edu/data/.

7 FAQ

- Dokładność obliczeń

Proszę używać liczb zmiennoprzecinkowych o podwójnej precyzji.

- Czy możemy założyc, że wejście jest poprawne?

Tak. Czyli program może zachowywać się nieprzewidywalnie dla wejść niespełniających specyfikacji.

- Czy możemy stworzyć dodatkowe, pomocnicze wątki?

Liczba wątków "liczących", tzn. wykonujących algorytm Bradesa, musi być równa `liczba-watkow`, pierwszemu parametrowi wywołania programu. Można stworzyć dodatkowe, pomocnicze wątki, ale co najwyżej O(`liczba-`

`watkow`). Dodatkowe wątki mogą wykonać co najwyżej O(1) operacji.

- Czy można korzystać z dodatkowych bibliotek?

Nie. W rozwiązaniu można korzystać tylko ze standardowej biblioteki CPP14. W szczególności nie można korzystać z Boosta.

- Co można założyć o identyfikatorach węzłów grafu?

Zmieszczą się do zwykłego `int`.

Data: 2016/12/26
Autor: Krzysztof Rządca
Created: 2016-12-26 Mon 21:06
Emacs 25.1.1 (Org mode 8.2.10)
Validate