b-skojarzenia b-adoratorów

Spis treści

Nowy termin zadania: 31/12/2017 21:00

Termin zadania: 21/12/2017 21:00

Modyfikacje treści:

- 2018/01/17 notes on results
- 2018/01/02 Results
- 2017/12/22 FAQ
- 2017/12/20 Due date moved to 31/12/2017; caution: I'm on vacations 23/12-07/01. I won't read my mail on vacations.
- 2017/12/18 FAQ; also, please verify that your student login is on the list https://www.mimuw.edu.pl/~krzadca/studenci-pw1718.txt ; if it's missing, write me. After the deadline, we will put here a link to a document listing ids of students whose programs passed `verify.py` tests.
- 2017/12/11 FAQ
- 2017/12/07 small errors in b-adorators algorithm corrected (thanks to Wojciech Przybyszewski); FAQ
- 2017/12/05 submission instructions modified
- 2017/12/04 plik: `ab123456.zip` zamień ab123456 na login ze `students.mimuw.edu.pl`; nie korzystaj z zewnętrznych bibliotek takich jak OpenMP.
- 2017/11/30 zadanie opublikowane.

1 Comments on the results

We scored correctness (5 points), performance (3 points) and the report (2 points). Our test data: https://www.mimuw.edu.pl/~krzadca/adorators-tests.zip .

We verified correctness of your programs doing over 1500 tests. We used 13 graphs; 1,2,8 and 100 threads; and 4 different ways to generate b for a node (one of which was repeated 27 times). We run tests on each (graph, thread number) combination with a time limit of 5 seconds. We then counted the number of incorrect outputs. We used the following score: 5 points for no errors; 4 points for less than 50 errors; 3 – 100; 2 – 200; 1 – 400.

We measured performance of your programs by executing them on a (slightly oldish) server with Intel E5-2620 v3, 2.40GHz. We executed your programs on all cores with hyperthreading (24 threads). Initially, we used 5 big graphs, but some of your programs (even the ones that passed the correctness tests) failed to produce correct results on some of the graphs, so finally we took two graphs that were the least difficult ones. We used

- `erdos-10000-1000`, an Erdos-Renyi graph with 10.000 nodes (and a probability of an edge of 0.2) and weights between 1 and 10;
- `dblp-w`, the DBLP graph (from SNAP) with randomly-assigned weights (between 1 and 50).

The programs had a time limit of 200 seconds.

We set blimit to 100. This value allowed us to amortize the I/O time needed to read the graph. Comparing to the blimit of 50, the run-time increased slightly less than 2 times, meaning that the I/O time becomes negligible.

We repeated each run 3 times, but found only minor differences between runs (less than 2%).

We then verified whether the result is correct: we did not award points for incorrect results (if the result is incorrect, your algorithm might did much less work than the correct one, so the comparison would be unfair).

We used the following score for each graph (if the result is correct and if the program had at least 3 points for correctness): 1.5 points if the program completed in 30 seconds or less; 1 point for 100 seconds; 0.5 points for completing in the time limit.

We tested your programs from 01/01/2018 and from 05/01/2018; we based our grades on versions from 05/01/2018.

The 2 points for the report score takes into account: speed-up computation (-0.5); quality of the description of your implementation (-0.5 or -1 points); and usage of mutexes in your code without checking the spin-locks (-0.5). We added a bonus of 0.5 points if the description or the tests were detailed.

We subtracted 1 point if your program from 01/01 did not pass the basic tests (the ones performed by the python verification script below). We emphasized that you should follow the specification; not only we published a python verificator, but also the way we will compile and run your programs and example input and output. If your program did not compile only because of missing `blimit.hpp`, we will not subtract this point (we will update the score in the next few days).

If you have any further questions, please contact us during our office hours (Tuesday, 12:00-14:00, room 4280).

2 Results

Results of the basic `verify.py`:

- copied at 2017/12/31 - https://www.mimuw.edu.pl/~krzadca/pw-res-2017-31-12.txt
- copied at 2018/01/01 - https://www.mimuw.edu.pl/~krzadca/pw-res-2018-01-01.txt
- copied at 2018/01/02 - https://www.mimuw.edu.pl/~krzadca/pw-res-2018-01-02.txt
- copied at 2018/01/03 - https://www.mimuw.edu.pl/~krzadca/pw-res-2018-01-03.txt
- copied at 2018/01/05 - https://www.mimuw.edu.pl/~krzadca/pw-res-2018-01-05.txt

Programs not having 'OK' as a result of the basic verification will get 0 points, unless corrected very soon. We will copy your programs automatically once or twice again, so correct your program and prepare the code as it is specified in Section 6 .

If you are not on this list, you were not on the list https://www.mimuw.edu.pl/~krzadca/studenci-pw1718.txt . Write me to be added to that list.

If your status code is 'missing', our script did not find directory `pw-zadanie2-cpp` under your home dir at `students.mimuw.edu.pl`.

If your status code is 'cmake' or 'make', your program does not compile on `students.mimuw.edu.pl` (for instance, `blimit.hpp` is missing). Make sure your program compiles with the method specified in Section 5.1 .

If your status code is 'notfound' your executable does not have the correct name (`adorate`).

If your status code is 'out' your output is incorrect (e.g. you write debugging info to stdout). Make sure your program passes `verify.py` script.

3 Wstęp

Problem b-skojarzeń (b-matching) uogólnia problem skojarzeń dopuszczając skojarzenie każdego wierzchołka \(v\) w grafie \(G=(V,E)\) z co najwyżej \(b(v)\) wierzchołkami (czyli standardowy problem skojarzeń to problem b-skojarzeń z `b(v)=1` dla każdego \(v\)). Zajmiemy się problemem b-skojarzeń w nieskierowanym grafie ważonym \(w(E)\). Celem jest znalezienie skojarzenia o maksymalnej sumarycznej wadze krawędzi. Algorytmy dokładne dla tego problemu istnieją, ale mają dużą złożoność lub są trudne w implementacji.

4 Algorytm b-adoratorów

Algorytm b-adoratorów (KHAN, Arif, et al, 2016) jest szybkim i prostym w implementacji rozwiązaniem problemu b-skojarzeń. Algorytm ten jest 2-aproksymacją: w najgorszym przypadku zwróci rozwiązanie co najwyżej dwa razy gorsze od optymalnego.

Prosimy o zaimplementowanie równoległej wersji algorytmu b-adoratorów w C++ 14.

Pseudokod sekwencyjnego algorytmu b-adoratorów przedstawiamy poniżej; dalsze wyjaśnienia w (KHAN, Arif, et al, 2016).

Algorytm używa następujących zmiennych: `Q, R`: kolejki węzłow; `S[v]`: adoratorzy węzła v (maksymalnie `b(v)`); `T[v]` węzły, które `v` adoruje; `N[v]`: sąsiedzi węzła v; `W(u,v)` waga krawędzi `(u,v)`; `S[v].last`: adorator węzła `v` z najmniejszą wagą gdy `S[v].size()==b[v]`, w przeciwnym wypadku `null` (zakładamy, że `W(v, null)==0`).

Żeby uzyskać deterministyczne rozwiązanie, algorytm wymaga ostrego porządku `:<:` wśród krawędzi wychodzących z każdego wierzchołka. Porządek ten definiujemy następująco: `W(v,w) :<: W(v,x) <=> (W(v,w) < W(v,x)) || (W(v,w)==W(v,x) && w < x)`.

```
Q = V

while (!Q.empty()) {

  for each u : Q {

    while (T[u].size() < b(u)) {

       x = arg max(v) {W(u,v) : (v in N[u] - T[u]) && (W(u,v) :>: W(v,
S[v].last))};

       if (x == null)

          break;

       else { // u will adorate x

          y = S[x].last;

          S[x].insert(u);

          T[u].insert(x);

          if (y != null) { // see also the FAQ

             T[y].remove(x);

             R.add(y);

          }

       }

    }

  }

  Q=R; R={};

}
```

5 Wejście i wyjście

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

Publikujemy skrypt `verify.py` weryfikujący poprawność w podstawowym stopniu (nasze skrypty będą zawierały zdecydowanie więcej testów). Programy nie przechodzące weryfikacji przez `verify.py` dostaną 0 punktów. (link i instrukcje poniżej)

5.1 Sposób uruchomienia programu

```
unzip xx123456.zip; cd xx123456; cp $TESTROOT$/blimit.cpp .; mkdir build; cd build; cmake -DCMAKE_BUILD_TYPE=Release ..; make
```

```
./adorate liczba-wątków plik-grafu limit-b
```

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.

5.2 Wejście

Plik grafu opisuje krawędzie w grafie w formacie `<id wierzchołek1> <id wierzchołek2> <waga>`. Graf jest nieskierowany. Krawędzie nie są posortowane. Wierzchołki nie muszą być numerowane sekwencyjnie (tzn. poprawny jest graf bez np. wierzchołka o numerze 0). Graf nie musi być spójny. Plik może zaczynać się od kilku linii komentarza: każda z takich linii zaczyna się symbolem #.

Możesz przyjąć, że `id wierzchołka` \(\in [0, 10^9]\); `waga` \(\in [1, 10^6]\); a waga najlepszego b-dopasowania (rozwiązanie) nie przekroczy \(10^9\).

Możesz przyjąć, że cały graf zmieści się w pamięci operacyjnej.

Możesz przyjąć, że plik grafu jest poprawnie sformatowany.

Maksymalną liczbę dopasowanych wierzchołków \(b(v)\) zwraca funkcja `bvalue` o następującej sygnaturze (`blimit.hpp`):

```
unsigned int bvalue(unsigned int method, unsigned long node_id);
```

Gdzie `method` to metoda generowania tej liczby. Przykładową implementację tej funkcji znajdziesz w `blimit.cpp`. Możesz założyć, że koszt obliczeniowy `bvalue()` jest niski (kilka instrukcji arytmetycznych).

Uwaga: w czasie testów będziemy podmieniać implementację `blimit.cpp` na nasze wersje używając `cp nasz_katalog_testowy/nasz_plik_blimit.cpp twój_katalog/blimit.cpp`. Po takim kopiowaniu twój program musi poprawnie kompilować się i następnie używać tej podmienionej wersji.

Twój program powinien wczytać graf wejściowy. Następnie, kolejno dla każdego `metoda_b` \(\in\) `[0, limit_b]`: (1) uruchomić algorytm adoratorów z \(b(v)\) wygenerowanymi funkcją `bvalue(metoda_b, ...)`; i (2) wypisać na standardowym wyjściu sumę wag dopasowanych krawędzi.

Przykładowe wejście: plik grafu

```
# this graph is adopted from Khan et al, 2016
0 2 8
0 3 6
0 4 4
0 5 2
1 2 3
1 3 7
1 4 1
1 5 9
```

Funkcja bvalue (`blimit.cpp`):

```
unsigned int bvalue(unsigned int method, unsigned long node_id) {

    switch (method) {

    case 0: return 1;

    default: switch (node_id) {

        case 0: return 2;

        case 1: return 2;

        default: return 1;

        }

    }

}
```

Program uruchomiony przez: `./adorate liczba-watkow graf.txt 1` powinien zwrócić:

```
17
28
```

6 Forma oddania zadania

Prosimy o stworzenie katalogu `pw-zadanie2-cpp` jako podkatalogu twojego katalogu domowego na students. W katalogu powinny znajdować się następujące pliki:

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

Po terminie oddania zadania, nasze skrypty przekopiują zawartości tych katalogów.

W raporcie opisz twoją implementację i zastosowane optymalizacje. Dla wybranego grafu z biblioteki SNAP (bibliografia) zmierz speed-up, czyli zmierz czas działania twojego programu \(p_m\) dla \(m=1, 2, ..., 8\) wątków, a następnie oblicz przyspieszenie: \(S_m=p_1 / p_m\). (zadbaj o taki sposób przeprowadzenia pomiaru, by wczytanie grafu z pliku nie zdominowało mierzonego czasu działania).

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

Program może korzystać tylko ze standardowej biblioteki cpp (w szczególności - nie może korzystać z boost'a ani z OpenMP).

7 Kryteria oceny

Będziemy oceniać poprawność, wydajność oraz raport.

Poprawność będziemy oceniać porównując wyniki implementacji z naszym rozwiązaniem wzorcowym. 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 dużych grafach. Będziemy testować programy na komputerach kilku-procesorowych (każdy procesor kilkunasto-rdzeniowy). Będziemy ustawiać liczbę wątków równą łącznej liczbie rdzeni. Pojedyncze wykonanie programu będziemy ograniczać do kilku (max. 5) minut.

8 Nasze materiały

W pliku adorators-files.zip znajdziesz sprawdzaczkę, przykładowe rozwiązanie i przykładowe dane. Instrukcja obsługi:

```
unzip adorators-files.zip
cd ab123456/ # sample solution
./packme.sh # moves the solution to ../sols where verify will look for it
cd ../verify
python verify.py # on students, use python3 verify.py
```

Oczekiwany rezultat:

```
INFO:verifyprograms:run number 1
INFO:verifyprograms:solution: ab123456
DEBUG:verifyprograms:file: ../../../ref-results/res-paper-1.txt
INFO:verifyprograms:dir: ab123456 params: 1_paper_1
INFO:verifyprograms:subprocess finished
DEBUG:verifyprograms:file: stdout_1_paper_1.txt
INFO:verifyprograms:error in the result: expected 17 seen 42
INFO:verifyprograms:error in the result: expected 28 seen 42
DEBUG:verifyprograms:file: ../../../ref-results/res-paper-1.txt
INFO:verifyprograms:dir: ab123456 params: 2_paper_1
INFO:verifyprograms:subprocess finished
DEBUG:verifyprograms:file: stdout_2_paper_1.txt
INFO:verifyprograms:error in the result: expected 17 seen 42
INFO:verifyprograms:error in the result: expected 28 seen 42
INFO:verifyprograms:solution ab123456 errors 4
```

9 Dodatkowe materiały

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

- KHAN, Arif, et al. Efficient approximation algorithms for weighted b-matching. SIAM Journal on Scientific Computing, 2016, 38.5: S593-S619, https://www.cs.purdue.edu/homes/apothen/Papers/bMatching-SISC-2016.pdf
- Dane do testów wydajnościowych można wziąć z http://snap.stanford.edu/data/.

10 FAQ

- Should we process different methods (different b values) in parallel?

You can, but this is a matter of efficiency vs code complexity. In our performance tests, we will run your code on graphs having large number of vertices. In constrast, `b-limit` might be lower than the available number of threads, `liczba-wątków`.

- Do we have to maintain exactly `liczba-wątków` threads during the whole runtime?

No. Consider `liczba-wątków` rather as the upper limit on the total number of threads your program can use. You might use less threads, but this could impact the efficiency. (note however, that this is a parralel programming course - single-threaded programs are not admissible).

- Shouldn't the code of the algorithm remove node `y` from S[x] when `y != null`?

`S[x]` is a priority queue limited to maximally `b[x]` nodes. So, if `y != null`, `S[x].insert(u)` implicitly removes `y`, a worse adorator, from `S[x]`. In other words, if `y != null`, after adding `u`, `y` is no longer in `S[x]`.

- Can we use `liczba-wątków + 1` threads (with the last thread just coordinating the work of the workers)?

No. Please follow the specification - `liczba-wątków` is the upper limit on the number of threads your program can use (including the main thread).

- Can we use C++17?

You can use all CPP features that are in the compiler installed on `students.mimuw.edu.pl`.

- Can we assume that `b[v]>0`?

Yes, you can.

- Is \(G(V,E)\) a simple graph?

Yes, it is. There is at most one edge between each pair of nodes. Also, the input is correctly formatted: if there is a line `x y w`, there won't be a line `y x w1`.

- Can the graph have loop edges (i.e., can the input have a line `v v w`)?

No, it can't.

- Can we use active waiting?

Yes, you can.

Data: 2018/01/17

Autor: Krzysztof Rządca

Created: 2018-01-18 Thu 07:40

Emacs 25.1.1 (Org mode 8.2.10)

Validate