Algorytmy optymalizacji dyskretnej 2025 / 26

LABORATORIUM 3

Porównanie implementacji algorytmu Dijkstry

Termin realizacji: **ostatnie zajęcia przed 18.12.2025 r.**

Waga listy: $w_3 = 0{,}2$

Warunek zaliczenia listy: *implementacja i przetestowanie co najmniej dwóch wariantów algorytmu Dijkstry (wraz ze sprawozdaniem).*

Zadanie 1. [5 pkt]
Zaimplementuj i przetestuj następujące warianty algorytmu DIJKSTRY dla problemu najkrótszych ścieżek z jednym źródłem w sieci $G = (N, A)$ o n wierzchołkach i m łukach z nieujemnymi kosztami (algorytmy wyznaczają najkrótsze ścieżki między zadanym źródłem $s \in N$ i wszystkimi wierzchołkami $i \in N \setminus \{s\}$):

- **[1,5 pkt]** wariant podstawowy (wykorzystujący np. kopiec binarny lub inną wydajną implementację kolejki priorytetowej),
- **[1,5 pkt]** algorytm DIALA (z $C + 1$ kubełkami),
- **[2 pkt]** algorytm RADIX HEAP.

Wymagania odnośnie rozwiązania zadania, w tym sposobu wywołania programów, zakresu przeprowadzonych testów oraz danych testowych opisane są poniżej.

Wymagania dotyczące kompilacji i uruchamiania programów

1. Dobierz technologię (język programowania, użyte biblioteki, itp.) tak, żeby wynikowy kod działał na tyle szybko i wydajnie, aby możliwe było przeprowadzenie testów dla wszystkich wskazanych danych testowych (ich rozmiar jest bardzo duży).
2. Programy powinny się kompilować, uruchamiać i działać poprawnie w systemie Ubuntu (ewentualnie w innej dystrybucji Linuxa).
3. Należy dołączyć plik `README` z danymi autora oraz opisem dostarczonych plików i listą bibliotek, które należy doinstalować, żeby programy się kompilowały i uruchamiały.
4. Programy powinny kompilować i linkować się automatycznie, np. za pomocą polecenia `make`. Do rozwiązania należy więc dołączyć odpowiedni plik `Makefile` (w przypadku języków interpretowanych opisz w pliku `README` sposób uruchomienia programu).
5. Programy powinny pozwalać na uruchomienie ich z odpowiednimi parametrami (patrz poniżej).
6. Pliki z danymi wejściowymi oraz pliki wynikowe powinny mieć określony format (patrz poniżej).

Przykładowo, dla algorytmu DIJKSTRY program powinien być uruchamiany za pomocą następujących poleceń (dwa sposoby wywołania).

- Badanie czasu wyznaczania najkrótszych ścieżek:

  `dijkstra -d plik_z_danymi.gr -ss zrodla.ss -oss wyniki.ss.res`,

  gdzie opcja `-d` oznacza dane, natomiast `plik_z_danymi.gr` jest plikiem, w którym zadana jest sieć z kosztami łuków. Postać pliku z danymi (sieć + koszty) jest następująca.

```
c Linia zaczynajaca sie od c jest komentarzem
c Przyklad sieci z kosztami - plik plik_z_danymi.gr
c
p sp 6 8
c p (problem) sp (shortest path)
c siec zawiera 6 wierzcholkow i 8 lukow
c wierzcholki ponumerowane sa od 1 do 6
c ...
c ...
a 1 2 17
c luk z wierzcholka 1 do wierzcholka 2 o koszcie 17
c
a 1 3 10
a 2 4 2
a 3 5 0
a 4 3 0
a 4 6 3
a 5 2 0
a 5 6 20
```

Opcja `-ss` oznacza, że należy wyznaczyć najkrótsze ścieżki między źródłem $s \in N$ i wszystkimi wierzchołkami $i \in N \setminus \{s\}$ dla wszystkich źródeł s, które zadane są w pliku `zrodla.ss`. Jego postać jest następująca.

```
c Przyklad pliku zrodla.ss, w ktorym zadane sa zrodla.
c
p aux sp ss 3
c   tu podane sa 3 zrodla w nastepujacych po sobie liniach
s 1
s 3
s 5
c
c algorytm powinien wyznaczyc nakrotsze sciezki miedzy
c zrodlem 1 i wszystkimi wierzcholkami sieci zadanej w pliku
c plik_z_danymi.gr, a nastepnie miedzy zrodlem 3 i wszystkimi
c wierzcholkami sieci itd.
```

Opcja `-oss wyniki.ss.res` określa plik wynikowy dla problemu najkrótszych ścieżek z jednym źródłem, którego postać jest następująca.

```
c Przyklad pliku wynikowego wyniki.ss.res dla problemu
c najkrotszych sciezek z jednym źródłem.
c
p res sp ss dikstra
c ------------------------------------------------------------
c
c wyniki testu dla sieci zadanej w pliku plik_z_danymi.gr
c i zrodel zrodla.ss:
f plik_z_danymi.gr zrodla.ss
c
c siec sklada sie z 1024 wierzcholkow, 4096 lukow,
c koszty naleza do przedzialu [0,1024]:
g 1024 4096 0 1024
c
```

```
c sredni czas wyznaczenia najkrotszych sciezek miedzy zrodlem
c a wszystkimi wierzcholkami wynosi 12.71 msec:
t 12.71
```

- Długości najkrótszych ścieżek między podanymi parami wierzchołków:

  `dijkstra -d plik_z_danymi.gr -p2p pary.p2p -op2p wyniki.p2p.res`,

  gdzie opcja `-d` jest jak poprzednio, natomiast opcja `-p2p` oznacza, że należy policzyć najkrótszą ścieżkę między każdą z par wierzchołków $s \in N$ oraz $t \in N$ podanych w pliku `pary.p2p`, którego postać jest następująca.

```
c Przyklad pliku pary.p2p (point-to-point), w ktorym zadane sa
c pary wierzchołkow, miedzy ktorymi nalezy wyznaczyc
c najkrotsze sciezki.
c
p aux sp p2p 3
c   tu np. podane sa 3 pary (1, 5), (5, 1) i (1, 2)
c
q 1 5
q 5 1
q 1 2
```

  Opcja `-op2p wyniki.p2p.res` określa plik wynikowy dla problemu najkrótszej ścieżki między parą wierzchołków, którego postać jest następująca.

```
c Przyklad pliku wynikowego wyniki.p2p.res dla problemu
c najkrotszej sciezki miedzy para wierzcholkow.
c
c wyniki testu dla sieci zadanej w pliku plik_z_danymi.gr
c i par zrodlo-ujscie podanych w pliku pary.p2p:
f plik_z_danymi.gr pary.p2p
c
c siec sklada sie z 2048 wierzcholkow, 8192 lukow,
c koszty naleza do przedzialu [0,1024]:
g 2048 8192 0 1024
c
c dlugosci najkrotszych sciezek
c   np. miedzy para (1,5) dlugosc sciezki wynosi 4351:
d 1 5 4351
d 5 1 7541
d 1 2 231
```

Podobnie powinno wyglądać wywołanie programu dla algorytmu DIALA:

- `dial -d plik_z_danymi.gr -ss zrodla.ss -oss wyniki.ss.res`
- `dial -d plik_z_danymi.gr -p2p pary.p2p -op2p wyniki.p2p.res`

oraz dla implementacji RADIX HEAP:

- `radixheap -d plik_z_danymi.gr -ss zrodla.ss -oss wyniki.ss.res`
- `radixheap -d plik_z_danymi.gr -p2p pary.p2p -op2p wyniki.p2p.res`

Dane testowe i opracowanie wyników

Dane testowe dostępne są na stronie 9th DIMACS Implementation Challenge – Shortest Paths. W celu przeprowadzenia testów pobierz źródła programów do pobierania i generowania danych testowych – Challenge 9 benchmarks. Następnie je rozpakuj, skompiluj za pomocą polecenia `make`[1] i wygeneruj dane testowe za pomocą polecenia `make gen`[2] (szczegóły opisane są w pliku README). Dane testowe zostaną wygenerowane do folderu `./inputs`. Formaty plików z danymi opisane są powyżej (szczegóły podane są też na stronie File formats).

Testy przeprowadź dla poniższych rodzin grafów opisanych w pliku `families.pdf` (znajdującym się w paczce z testami w folderze `docs/families/`) i wskazanego tam zakresu parametrów:

- **Random4-n** oraz **Random4-C** (podpunkt 3.1),
- **Long-n** oraz **Square-n** (podpunkt 3.2),
- **Long-C** oraz **Square-C** (podpunkt 3.2; dopuszczamy sytuację, w której dla kilku ostatnich wartości i bliskich 15, testy dla $C = 4^i$ mogą się nie skończyć; ewentualnie można wówczas spróbować wziąć mniejsze n, np. $n = 2^{15}$),
- jedna z rodzin **USA-road-t** lub **USA-road-d** (podpunkt 3.3).

W każdym przypadku zbadaj czas wyznaczania najkrótszych ścieżek do wszystkich wierzchołków:

- od jednego źródła będącego wierzchołkiem o najmniejszym indeksie,
- od pięciu wierzchołków wybranych jednostajnie losowo (te same dla wszystkich algorytmów); bierzemy średnią z czasów dla poszczególnych źródeł.

Wyniki przedstaw w formie wykresów.

Dodatkowo dla największej instancji w każdej z testowanych rodzin grafów (największe wartości n i / lub C; największy graf dla sieci drogowej) wyznacz długość najkrótszej ścieżki pomiędzy wierzchołkami o najmniejszym i największym indeksie oraz dla 4 innych losowo wybranych par wierzchołków (te same dla wszystkich algorytmów). Wyniki przedstaw w formie tabel.

Wyniki przeprowadzonych eksperymentów przedstaw w **zwięzłym** sprawozdaniu (plik pdf). Sprawozdanie powinno zawierać

1. zwięzły opis własnych implementacji algorytmów oraz ich złożoności (w przypadku korzystania z implementacji struktur danych dostępnych w bibliotekach języka należy sprawdzić, jaka jest złożoność wykonywanych operacji),
2. wyniki eksperymentów porównujących zaimplementowane algorytmy dla danych testowych (tabele, wykresy),[3]
3. interpretację uzyskanych wyników oraz wnioski.

Do sprawozdania należy dołączyć pliki z kodem źródłowym oraz pliki `Makefile` i `README` (spakowane programem `zip`, a archiwum nazwane numerem indeksu). Archiwum nie powinno zawierać żadnych zbędnych plików.

[1] **Uwaga!** Mogą wystąpić błędy kompilacji. Np. w przypadku błędu `undefined reference to 'pow|log|sqrt|...'` spróbuj przenieść flagę `-lm` (zmienna `LOADLIBS`) na koniec polecenia kompilującego w plikach `Makefile` w folderach `gens/grid`, `gens/rand` i `gens/tor`.

[2] **Uwaga!** Mogą wystąpić błędy. Np. w przypadku błędu pobierania danych dla sieci drogowej USA sprawdź, czy masz zainstalowanego klienta ftp (polecenie `ftp` w terminalu). Można też np. pobrać dane „ręcznie" ze strony 9th DIMACS Implementation Challenge – Shortest Paths i umieścić je w odpowiednim podkatalogu `inputs` (patrz skrypty `scripts/genUSA-road-[d|t].gr.pl`).

[3] Patrz Zadanie 0. z Listy 1. na laboratorium.

Użyteczne linki

- 9th DIMACS Implementation Challenge - Shortest Paths
- Shortest paths: label setting algorithms
- The radix heap algorithm