DISTRIBUIRANI ALGORITMI ZA TRAŽENJE NAJKRAĆEG PUTA

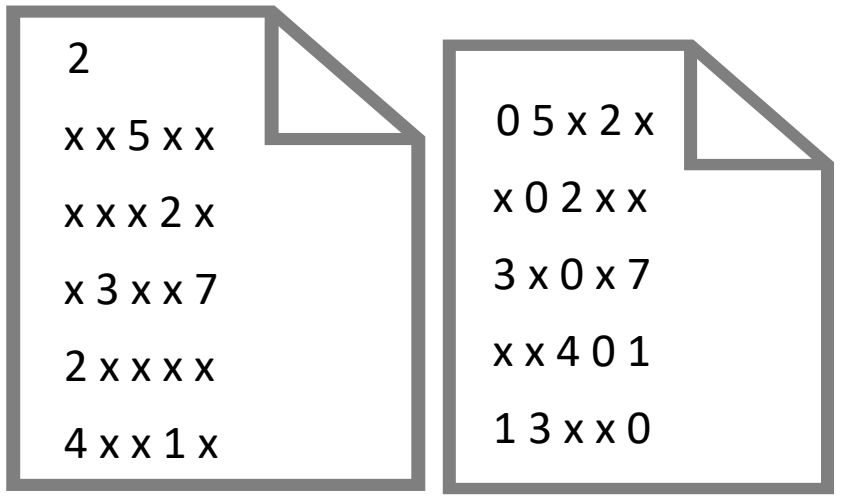
Dominik Mikulčić
Petra Škrabo

UVOD

- problem pronalaska najkraćeg puta u težinskom grafu
- distribuirane varijante algoritama
  - *Bellman-Ford* – sinkroni i asinkroni
  - *Floyd-Warshall* – asinkroni
- 1 vrh grafa = 1 proces
- Java programski jezik

REPREZENTACIJA GRAFOVA

graph.txt



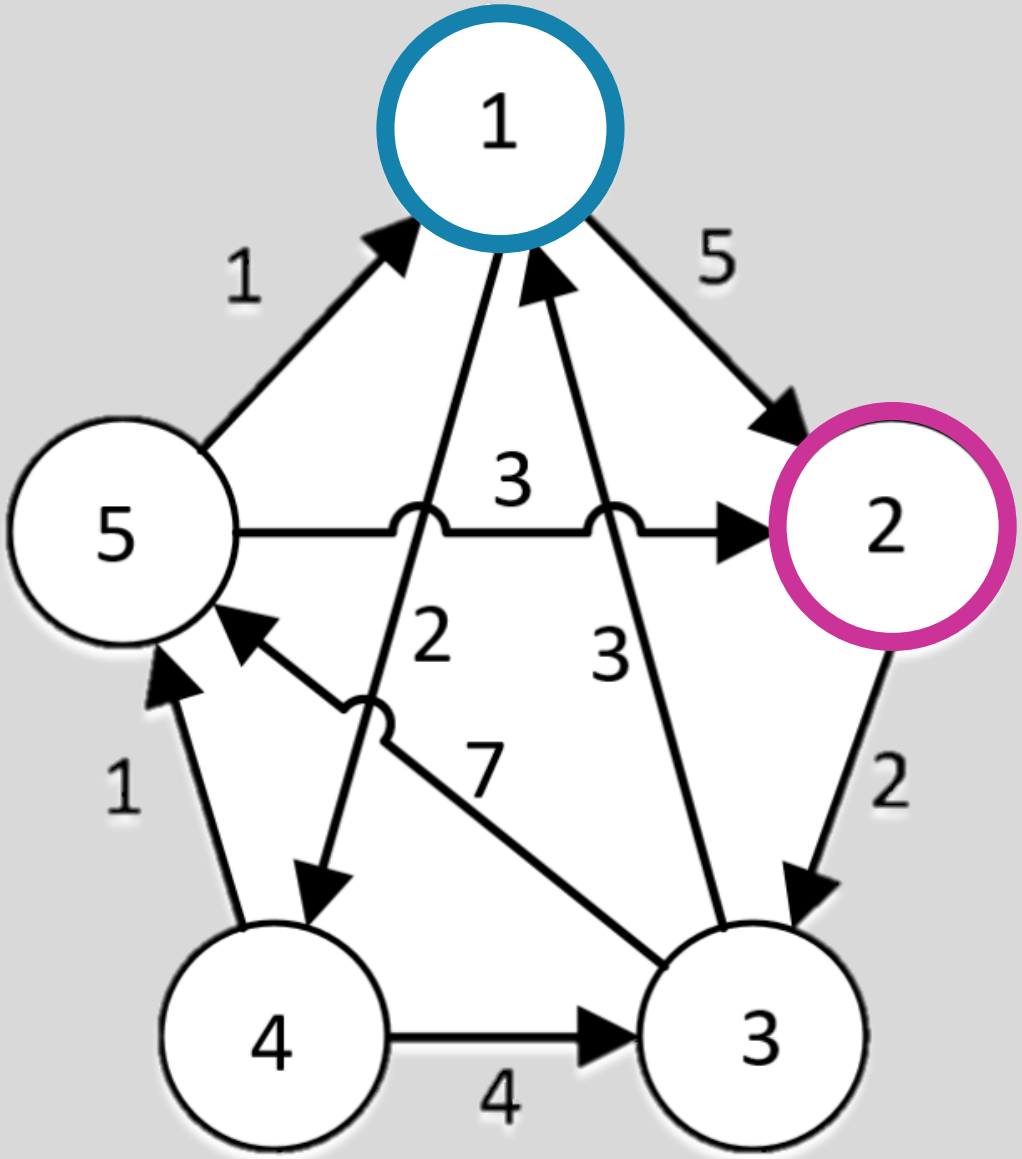
graphBellmanFord.txt graphFloydWarshall.txt

Graph.java

- `readFile()`
  - konstrukcija težinske matrice iz .txt datoteke
- `findNeighbours()`
- `findParents()`
- `setUpWeights()`

zadavanje grafa

pozitivne ili negativne duljine bridova
bez negativnih ciklusa



2
x 5 x 2 x
x x 2 x x
3 x x x 7
x x 4 x 1
1 3 x x x

graphBellmanFord.txt

0 5 x 2 x
x 0 2 x x
3 x 0 x 7
x x 4 0 1
1 3 x x 0

graphFloydWarshall.txt

BELLMAN-FORDOV ALGORITAM

Proces i

- **IZVOR**
  - čvor od kojeg se traže najkraći putevi do svih ostalih čvorova
- **N**
  - ukupan broj čvorova
- **DULJINE**
  - lista duljina najkraćih puteva od izvora do nekog čvora
- **RODITELJI**
  - lista roditelja čvorova

Algoritam 1 *Bellman-Fordov* algoritam

```
1: za svaki vrh radi
2:     duljine[vrh] := ∞
3:     roditelji[vrh] := null
4: kraj za
5: duljine[izvor] = 0
6: za i = 0 do N radi
7:     za svaki brid (u, v) s težinom w radi
8:         ako duljine[u] + w < duljine[v] onda
9:             duljine[v] = duljine[u] + w
10:            roditelji[v] = u
11:        kraj ako
12:    kraj za
13: kraj za
14: vrati
```

inicijalizacija duljina i roditelja

postavljanje duljine puta od izvora do izvora na 0

N puta se ponavlja: za svaki brid (u, v) se gleda je li trenutna duljina puta od izvora do čvora v veća od trenutne duljine puta od izvora do čvora u zbrojene s težinom promatranog brida – ukoliko jest, pridružuje se nova vrijednost

SINKRONI BELLMAN-FORDOV ALGORITAM

- Pravilni pulsevi
- Tipovi poruka:
  - `UPDATE`
  - `(PARENT)`

Algoritam 2 Sinkroni *Bellman-Fordov* algoritam za proces P_i

$duljina = \infty$
$roditelj = -1$

tip poruke: UPDATE

1: **ako** $i = i_0$ **onda**
2: $\quad duljina = 0$
3: **kraj ako**
4: **za** $puls = 1$ **do** $N - 1$ **radi**
5: $\quad$ **pošalji** UPDATE(i, $duljina$) svim susjedima
6: $\quad$ **čekaj** UPDATE(j, $duljina_j$) **za svaki** $j \in Susjedi$
7: $\quad$ **za svaki** $j \in Susjedi$ **radi**
8: $\quad\quad$ **ako** $duljina > duljina_j + težina_{i,j}$ **onda**
9: $\quad\quad\quad duljina = duljina_j + težina_{i,j}$
10: $\quad\quad\quad roditelj = j$
11: $\quad\quad$ **kraj ako**
12: $\quad$ **kraj za**
13: **kraj za**

inicijalizacija duljine i roditelja

postavljanje duljine puta od izvora do izvora na 0

N-1 puta se ponavlja: Proces i šalje svim susjedima poruku `UPDATE` i čeka poruku `UPDATE` od svakog susjeda – tada ažurira duljinu svog puta ovisno o sadržaju

od izvora pa nadalje se prosljeđuju poruke tipa `PARENT` i dohvaćaju informacije o putu

IMPLEMENTACIJA

- NameServer
- SynchBellmanFordTester
- Linker
- AlphaSynch
- SynchBellmanFord
- Process
- Msg

ASINKRONI BELLMAN-FORDOV ALGORITAM

- Nema pulseva
- Tipovi poruka:
  - `UPDATE`

Algoritam 3 Sinkroni *Bellman-Fordov* algoritam za proces P_i

$duljina = \infty$

tip poruke: UPDATE

1: **ako** $i = i_0$ **onda**
2: $\quad duljina = 0$
3: **kraj ako**

kada UPDATE$(i_0, duljina_j)$ stigne od čvora j:
4: **ako** $duljina > duljina_j + težina_{i,j}$ **onda**
5: $\quad duljina = duljina_j + težina_{i,j}$
6: $\quad$ **pošalji** UPDATE$(i_0, duljina)$ svakom susjedu
7: **kraj ako**

inicijalizacija duljine (i roditelja)

postavljanje duljine puta od izvora do izvora na 0

Čim god pristigne poruka tipa `UPDATE` od nekog drugog čvora, pristiže prije svega informacija o duljini puta ali i o trenutnom sadržaju puta – tada proces (po potrebi) ažurira trenutnu duljinu puta, sadržaj puta i šalje svim susjedima poruku tipa `UPDATE` s novim informacijama

IMPLEMENTACIJA

- `NameServer`
- `AsynchBellmanFordTester`
- `Linker`
- `AsynchBellmanFord`
- `Process`
- `Msg`

Proces i

- **`LENGTH[1..n]`**
  - duljina najkraćeg dosad poznatog puta od vrha `i` do preostalih vrhova grafa
- **`PARENT[1..n]`**
  - lista prethodnika na tom putu

FLOYD-WARSHALLOV ALGORITAM

ProcessFloydWarshall.java

KOMUNIKACIJA PROCESA ostvaruje se razmjenom poruka isključivo preko bridova grafa.

Identifikacija djece pivotnog čvora.

Ažuriranje najkraćeg dosad poznatog puta.

ProcessFloydWarshall.java

INICIJALIZACIJA

`Graph.java` `Linker.java`

Klasa **`Linker.java`** inicijalizira nizove **`LENGTH[]`** i **`PARENT[]`** te listu susjeda **`neighbours`** pomoću klase **`Graph.java`** koja sadrži sve informacije o grafu pročitane iz *.txt* datoteke.

Algoritam 4 Asinkroni distribuirani *Floyd–Warshallov* algoritam za proces P_i

```
1:  za pivot = 1 do n radi

2:    za svaki susjed nbh ∈ Neighbours radi
3:      ako PARENT[pivot] = nbh onda
4:        pošalji IN_TREE(pivot) susjedu nbh
        inače pošalji NOT_IN_TREE(pivot) susjedu nbh
5:      kraj ako
6:    kraj za svaki

7:    čekaj IN_TREE ili NOT_IN_TREE od svakog susjeda

8:    ako LEN[pivot] ≠ ∞ onda

9:      ako pivot ≠ i onda
10:       primi PIV_ROW(pivot, PIV_LEN[1...n], PIV_PAR[1...n]) od pivot
11:     kraj ako

12:     za svaki susjed nbh ∈ Neighbours radi
13:       ako je IN_TREE primljena od nbh onda
14:         ako pivot = i onda
15:           pošalji PIV_ROW(pivot, LEN[1...n], PARENT[1...n]) susjedu nbh
        inače pošalji PIV_ROW(pivot, PIV_LEN[1...n], PIV_PAR[1...n]) susjedu
    nbh
16:         kraj ako
17:       kraj ako
18:     kraj za svaki

19:     za t = 1 do n radi
20:       ako LEN[pivot] + PIVOT_LEN[t] < LEN[t] onda
21:         LEN[t] ← LEN[pivot] + PIV_LEN[t]
22:         PARENT[t] ← PIV_PAR[t]
23:       kraj ako
24:     kraj za

25:   kraj ako

26: kraj za
```

Svaki proces u `n` glavnih **iteracija** elementa `pivot` provjerava može li se do ostalih čvorova u grafu preko pivotnog elementa doći kraćim od dosad poznatog puta.

Identifikacija koji su susjedi ujedno i **djeca** `pivot` elementa.

Provjera je li poznat put **do** `pivot` čvora.

Primanje i **slanje** vektora duljina i roditelja čvora `pivot` sada poznatoj djeci.

Ažuriranje listi roditelja i duljina čvora `i` ukoliko je preko čvora pivot pronađen kraći od dosad poznatog puta.

ProcessFloydWarshall.java

IMPLEMENTACIJA .JAVA

NameServer

Linker

Msg

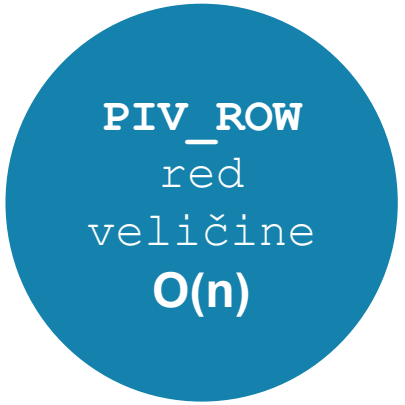
Process

ProcessFloydWarshall

FloydWarshallTester

U svakoj od n pivot iteracija imamo

- dvije **IN_TREE** ili **NOT_IN_TREE** poruke po svakom bridu
- najviše `n-1` **PIV_ROW** poruka



VREMENSKA SLOŽENOST izvršavanja Floyd-Warshallovog algoritma po čvoru grafa je $O(n^2)$.

Projekt je razvijan u **Java** programskom jeziku u **Eclipse** razvojnom okruženju.

Projekt je dostupan ovdje.