ALGORITMI PARALELI SI DISTRIBUITI
Tema #2 Traffic simulator

Termen de predare: 16.12.2020

*** pus si pe moodle ***

Cerinta

Pornind de la notiunile invatate in cadrul laboratoarelor, in aceasta tema veti avea de implementat modele de sincronizare a unor situatii din trafic. Pentru oricare dintre situatii, sablonul fisierelor de intrare folosite pentru testare este, aproximativ, urmatorul:

nume_intersectie
n
id_masina_0 banda_start_masina_0 timp_asteptare_masina_0_milisecunde
…
id_masina_n banda_start_masina_n timp_asteptare_masina_n_milisecunde
(optional) detalii despre intersectie (detaliate in cadrul fiecarei cerinte)

unde:

- **nume_intersectie** este numele unuia dintre scenariile descrise mai jos. Vedeti exemplele de input aferente fiecarei cerinte pentru detalii
- **n** este numarul de masini care asteapta sa intre in intersectie

Astfel, pentru fiecare situatie, veti considera ca sunt **n** masini care incearca sa treaca de intersectia descrisa in cadrul respectivei situatii. Aceste masini vor trebui *sincronizate* pentru a respecta regulile descrise in situatiile de mai jos.

Cerinta 1: simple_semaphore

Acest scenariu presupune ca un numar oarecare de masini sa astepte la semafor. Fiecare masina asteapta un anumit numar de secunde, dupa care poate pleca.

Sarcina voastra este sa afisati, sub forma mesajelor de mai jos, ordinea de intrare si iesire din intersectie a fiecarei masini.

Exemplu de input:
simple_semaphore

3
0 1 3000
1 1 3500
2 2 2000

Exemplu de output:
Car 0 has reached the semaphore, now waiting...
Car 1 has reached the semaphore, now waiting...
Car 2 has reached the semaphore, now waiting...
Car 2 has waited enough, now driving...
Car 0 has waited enough, now driving...
Car 1 has waited enough, now driving...

Explicatie:
Fiecare masina asteapta timpul care este scris in fisierul de intrare in dreptul acesteia, dupa care trece de intersectie. Sensul din care vin masinile este irelevant pentru aceasta cerinta.

Nota

Aceasta cerinta **nu** necesita detalii despre intersectie.

Cerinta 2: simple_n_roundabout

La intersectie pot **ajunge** oricate masini, dar in sensul giratoriu pot **intra maximum n** masini la un anumit moment. Odata **intrata** in sensul giratoriu, dureaza **T** milisecunde ca o masina sa il **paraseasca**. Numerele **n** si **t** sunt scrise in aceasta ordine pe ultima linie a fisierelor de test.

Exemplu de input:
simple_n_roundabout
3
0 1 3000
1 1 3500
2 2 2000
2 2000

Exemplu de output:
Car 0 has reached the roundabout, now waiting...
Car 1 has reached the roundabout, now waiting...
Car 2 has reached the roundabout, now waiting…

Car 0 has entered the roundabout
Car 1 has entered the roundabout
Car 1 has exited the roundabout after 2 seconds
Car 2 has entered the roundabout
Car 0 has exited the roundabout after 2 seconds
Car 2 has exited the roundabout after 2 seconds

Explicatie:
In sensul giratoriu sunt permise **maximum** 2 masini in orice moment, iar timpul necesar pentru ca acestea sa paraseasca sensul giratoriul este de 2000 ms. Masinile cu ID-urile 0 si 1 intra in sensul giratoriu, asteapta 2 secunde, dupa care ies si intra masina 2. Observati ca desi masina 2 intra in sensul giratoriu inainte ca masina 0 sa iasa, cerinta problemei este respectata.

Cerinta 3: simple_strict_1_car_roundabout

Aceasta cerinta simuleaza un sens giratoriu in care poate intra **o singura masina din fiecare directie** la un moment de timp. Masinile nu pot intra in intersectie pana nu intra **exact** cate una din fiecare directie. Voi trebuie sa va structurati outputul in functie de mesajele urmatoare:

M1: Car ID has reached the roundabout
M2: Car ID has entered the roundabout from lane X
M3: Car ID has exited the roundabout after T seconds

Numarul **T** de secunde cat dureaza ca o masina sa iasa din sensul giratoriu este scris pe ultima linie a fisierului de input.

Exemplu de input:
simple_strict_1_car_roundabout
4
3 0
0 1
1 1
2 0
2 2000

Exemplu de output:
Car 3 has reached the roundabout
Car 0 has reached the roundabout
Car 2 has reached the roundabout

Car 3 has entered the roundabout from lane 0
Car 1 has reached the roundabout
Car 0 has entered the roundabout from lane 1
Car 3 has exited the roundabout after 2 seconds
Car 0 has exited the roundabout after 2 seconds
Car 1 has entered the roundabout from lane 1
Car 2 has entered the roundabout from lane 2
Car 1 has exited the roundabout after 2 seconds
Car 2 has exited the roundabout after 2 seconds

Explicatie:
In sensul giratoriu vor intra mai intai masinile cu id-urile 0 si 3 (care vin din directiile 1 si 2), urmate de cele cu id-urile 1 si 2 (care vin din aceleasi doua directii). Observati ca ordinea de afisare a masinilor care sunt in sensul giratoriu la un moment dat este irelevanta.

Nota
NU conteaza daca mai **ajung** masini la girator in timp ce altele sunt intrate sau ies din el.

Cerinta 4: simple_strict_x_car_roundabout

Aceeasi cerinta ca mai sus, cu modificarea ca acum sunt permise maximum **x** masini dintr-o directie in sensul giratoriu, in loc de una singura. Acest numar **x** va fi scris pe ultima linie a fisierului de intrare. Cerinta 2 este echivalenta cu aceasta cand **x = 1**.
Pe ultima linie a fisierelor de input sunt scrise numarul **x** si timpul cat dureaza ca o masina sa iasa din sensul giratoriu, **T**.

Tipurile de mesaje pe care va trebui sa le afisati sunt:
M1: Car ID has reached the roundabout
M2: Car ID has been selected to enter the roundabout from lane X
M3: Car ID entering the roundabout from lane X
M4: Car ID has exited the roundabout after T seconds

Exemplu de input:
simple_strict_x_car_roundabout
8
3 0
0 1
1 1

2 1
4 1
5 0
6 0
7 0
2 2000 2

Exemplu de output:
Car 3 has reached the roundabout
Car 0 has reached the roundabout
Car 2 has reached the roundabout
Car 1 has reached the roundabout
Car 4 has reached the roundabout
Car 5 has reached the roundabout
Car 3 was selected to enter the roundabout from lane 0
Car 1 was selected to enter the roundabout from lane 1
Car 0 was selected to enter the roundabout from lane 1
Car 5 was selected to enter the roundabout from lane 0
Car 0 has entered the roundabout from lane 1
Car 3 has entered the roundabout from lane 0
Car 2 has entered the roundabout from lane 1
Car 5 has entered the roundabout from lane 0
Car 3 has exited the roundabout after 2 seconds
Car 2 has exited the roundabout after 2 seconds
Car 5 has exited the roundabout after 2 seconds
Car 0 has exited the roundabout after 2 seconds
Car 2 was selected to enter the roundabout from lane 1
Car 4 was selected to enter the roundabout from lane 1
Car 6 was selected to enter the roundabout from lane 0
Car 7 was selected to enter the roundabout from lane 0
Car 4 has entered the roundabout from lane 1
Car 7 has entered the roundabout from lane 0
Car 2 has entered the roundabout from lane 1
Car 6 has entered the roundabout from lane 0
Car 4 has exited the roundabout after 2 seconds
Car 6 has exited the roundabout after 2 seconds
Car 7 has exited the roundabout after 2 seconds
Car 2 has exited the roundabout after 2 seconds

Nota

CONTEAZA ca toate masinile sa **ajunga** la sensul giratoriu **inainte** sa se treaca mai departe. Toate masinile trebuie sa paraseasca sensul giratoriul inainte ca o runda noua de masini sa porneasca.

Cerinta 5: simple_max_x_car_roundabout

Aceasta cerinta este inca o generalizare a ultimelor doua cerinte, in cadrul careia in sensul giratoriu este permis accesul a **cel mult x** masini care vin din **aceeasi** directie.

Important

Pastrati sleepul pus in schelet inainte ca o masina sa intre in sensul giratoriu!

Nota

NU conteaza daca mai **ajung** masini la girator in timp ce altele sunt intrate sau ies din el.

Mesaje utilizate in rezolvarea problemei:

- Car ID has reached the roundabout from lane START_LANE_ID
- Car ID has entered the roundabout from lane START_LANE_ID
- Car ID has exited the roundabout after T seconds

Cerinta 6: priority_intersection

Aceasta cerinta simuleaza o intersectie dintre un drum cu prioritate si unul fara. Masinile cu prioritate vor intra in intersectie la orice moment si fiecare are nevoie de **2 secunde** sa paraseasca intersectia. Masinile fara prioritate vor trece prin intersectie doar daca in aceasta **nu** exista deja masini cu prioritate. Mai exact, masinile fara prioritate trebuie sa astepte pana trec **toate** masinile cu prioritate **aflate deja in intersectie** si trec una cate una, **in ordinea in care au venit**. Masinile **fara prioritate** trec **instant** prin intersectie.

O masina cu prioritatea mica va avea prioritatea **1**, iar masinile cu prioritate mare au **orice alta prioritate mai mare decat 1**, fiind irelevante diferentele intre prioritatile mari in rezolvarea acestei cerinte.

Exemplu de input:
priority_intersection
5

0 0 1 1200 1
1 0 1 1320 2
2 1 1 2100 1
3 1 1 2300 2
4 1 1 1200 1
3 2

Exemplu de output:
Car 0 with low priority is trying to enter the intersection...
Car 1 with high priority has entered the intersection
Car 2 with low priority is trying to enter the intersection…
Car 1 with high priority has exited the intersection
Car 3 with high priority has entered the intersection
Car 4 with low priority is trying to enter the intersection…
Car 3 with high priority has exited the intersection
Car 0 with low priority has entered the intersection
Car 2 with low priority has entered the intersection
Car 4 with low priority has entered the intersection

Important

Pastrati sleepul pus in schelet inainte ca o masina sa intre in intersectie!

Cerinta 7: crosswalk

Aceasta cerinta modeleaza o trecere de pietoni semaforizata intr-un mod inteligent. Mai exact, masinile au voie sa treaca (iar pietonii nu) pana cand se strang un numar de pietoni. In acest moment, semaforul arata masinilor culoarea rosu, iar pietonilor verde, pana cand trec toti pietonii. Se considera ca masinile merg in cerc, iar dupa ce trec de trecerea de pietoni, se reintorc sa astepte inaintea acesteia.

Urmariti comportamentul pietonilor in fisierul **entities/Pedestrians.java**
Pe ultima linie din fisierele de input sunt scrise timpul cat cat inca vin pietoni, precum si numarul maxim al acestora de la care semaforul le permite sa treaca.

Mesaje utilizate in rezolvarea problemei:

- Car ID has now green light
- Car ID has now red light

Un thread va afisa un mesaj doar daca el difera de mesajul precedent sau daca mesajul anterior a fost null (adica la inceput). Aici trebuie sa tineti o evidenta a istoricului mesajelor.

Cerinta 8: simple_maintenance

In acest scenariu, drumul este in lucru, motiv pentru care ambele sensuri vor circula alternativ cate **X** masini dintr-un sens pe o singura banda. Primele care trec sunt cei din sensul **0**. Se garanteaza ca pe oricare sens de circulatie asteapta un numar de masini divizibil cu **X**.

Pe ultima linie a fisierelor de input este scris numarul **X**.

Exemplu de input:
simple_maintenance
4
0 0
1 0
2 1
3 1
1

Exemplu de output:
Car 1 from side number 0 has reached the bottleneck
Car 3 from side number 1 has reached the bottleneck
Car 0 from side number 0 has reached the bottleneck
Car 1 from side number 0 has passed the bottleneck
Car 3 from side number 1 has passed the bottleneck
Car 2 from side number 1 has reached the bottleneck
Car 0 from side number 0 has passed the bottleneck
Car 2 from side number 1 has passed the bottleneck

Cerinta 9: complex_maintenance

O generalizare a task-ului precendent in care exista un singur sens de mers, N benzi (pe care le vom numi ***benzi vechi***), dintre care doar M sunt circulabile de la un punct incolo (pe care le vom numa ***benzi noi***), M < N, M > 0.
Voi trebuie sa redirectati traficul din **benzile vechi** pe cele **noi** sub o forma **many-to-one**. Vom nota cu Li benzile vechi asignate unei benzi noi cu indicele i.
Dintre cele Li benzi asignate se alege, pe rand, cate una din care trebuie sa treaca maxim X masini odata. Dupa ce au trecut cele maxin X masini, daca banda veche este goala, atunci este scoasa din discutie. Daca mai sunt masini pe aceasta, atunci va fi permutata la coada listei de asteptare Li.
De pe o banda veche, masinile trebuie sa porneasca **in ordinea** in care au ajuns.
Repartitia pe noile benzi incepe in momentul in care **toate** masinile s-au incolonat.

Exemplu:

Initial drumul are A1 … A10 benzi. La un moment dat, unele intra in renovare, ramanand disponibile doar 3 benzi de circulat. Vom considera ca vor ramane libere benzile A1, A2 si A3.
Putem realiza cuplajul de benzi:

- L1: {A1, A2, A3}, unde L1 e coada de asteptare a benzii A1
- L2: {A4, A5, A6}, unde L2 e coada de asteptare a benzii A2
- L3: {A7, A8, A9, A10}, unde L3 e coada de asteptare a benzii A3

In prima iteratie vor circula cate maxim X masini de pe A1, A4, respectiv A7.
Noua ierarhie va fi acum:

- L1: {A2, A3, A1}, unde L1 e coada de asteptare a benzii A1
- L2: {A5, A6, A4}, unde L2 e coada de asteptare a benzii A2
- L3: {A8, A9, A10, A7}, unde L3 e coada de asteptare a benzii A3

La a doua iteratie vom presupune ca A8 ramane fara masini disponibile, astfel o scoatem din coada L3:

- L1: {A2, A3, A1}, unde L1 e coada de asteptare a benzii A1
- L2: {A5, A6, A4}, unde L2 e coada de asteptare a benzii A2
- L3: {A9, A10, A7}, unde L3 e coada de asteptare a benzii A3

Mesaje utilizate in rezolvarea problemei:

- Car ID has come from the lane number OLD_LANE_ID
- Car ID from the lane OLD_LANE_ID has entered lane number NEW_LANE_ID
- The initial lane OLD_LANE_ID has been emptied and removed from the new lane queue
- The initial lane OLD_LANE_ID has no permits and is moved to the back of the new lane queue

Cerinta 10: railroad

Scenariul acesta este cel al unei treceri la nivel cu calea ferata. Initial bariera este coborata, timp in care din fiecare directie (de pe fiecare banda) ajung in intersectie masini. Dupa ce toate masinile au ajuns in intersectie si asteapta, trenul trece, iar masinile incep sa se miste **in ordinea in care au ajuns pe fiecare sens**.

Exemplu de input:
railroad
5
0 1
1 0

2 0
3 1
4 0

Exemplu de output:
Car 1 from side number 0 has stopped by the railroad
Car 2 from side number 0 has stopped by the railroad
Car 0 from side number 1 has stopped by the railroad
Car 4 from side number 0 has stopped by the railroad
Car 3 from side number 1 has stopped by the railroad
The train has passed, cars can now proceed
Car 1 from side number 0 has started driving
Car 0 from side number 1 has started driving
Car 3 from side number 1 has started driving
Car 2 from side number 0 has started driving
Car 4 from side number 0 has started driving

Explicatie:
Toate masinile ajung la calea ferata, dupa care una singura va afisa "The train has passed, cars can now proceed". Apoi trecerea se face pe fiecare sens, in ordinea in care masinile au ajuns la calea ferata.

Bonus:

- Intersectie nemarcata in care intra 4 masini din si cu directii diferite fixate; se doreste trecerea lor in ordinea corecta conform codului rutier, fara a exista accidente, fara deadlock-uri
- Intrecere a N masini fiecare pe banda ei; un obstacol poate aparea random blocand masina de pe linia sa timp de T secunde (hint: use Listeners). Considerand un numar maxim de obstacole ce pot aparea, printati clasamentul la fiecare schimbare a acestuia (inceput + dupa fiecare obstacol). Hint: use Listeners

Observatii

- Masinile vor avea intotdeauna un id si un sens de mers (start - end). La majoritatea task-urilor veti primi doar start-ul. Numai la bonus e nevoie si de end.
- Vi se vor oferi in teste informatii exacte de initializare a sensurilor giratorii (timp T de parcurge, respectiv capacitate N unde este cazul)

- Vi se va specifica in teste numarul variabil / maxim de masini ce pot intra in intersectie unde este cazul
- Citirea se va face din fisiere
- Un test verifica o **singura** situatie enumerata anterior (un singur task).
- Pe langa testul in sine veti primi si
    - un test_sablon care va specifica semnificatia fiecarui element din fisierul de test din tipul respectiv
- Veti primi un script de testare automata. Testarea manuala se poate realiza prin parcurgerea test cu test si redirectarea output-ului intr-un fisier.
- Afisarea se va face la Standard Output.
- Puteti modifica dupa cum doriti structura codului / puteti sa nu folositi scheletul cat timp respectati, unde este cazul, timpii de asteptare, respectiv modul de generare a pietonilor.
- Se garanteaza ca la testare se va tine cont de variabilitatea id-urilor.
- Pentru a putea utiliza scheletul, aveti nevoie de java 14 sau 15. O alternativa ar fi sa modificati sectiunile de cod unde se face “return switch” in switch si apoi return.

Sugestii

- Incercati sa va scrieti codul cat mai lizibil, modular si usor de extins
- Utilizati variabile si clase cu nume sugestive
- Lasati cat mai multe comentarii explicative in cod
- Respectati un coding style
- Utilizati cat mai multe design pattern-uri care considerati ca v-ar usura viata
- Exemple de design pattern-uri: Singleton, Factory, Prototype, Listener, Strategy etc.
- Puteti utiliza atat structuri de date sincronizate, cat si bariere, semafoare, syncronized, wait - notify - notifyAll, sleep

Notare

Tema se va trimite si testa automat la aceasta adresa: https://apd-checker.dfilip.xyz. Se va trimite sub forma unei arhive Zip care, pe langa fisierele sursa, va trebui sa contina urmatoarele doua fisiere ın radacina arhivei:

- Makefile - cu directiva build care compileaza tema voastra si genereaza un executabil numit tema2 par aflat in radacina arhivei
- README - fisier text in care sa se descrie pe scurt implementarea temei, precum si alternative la implementare dupa cum considerati.

Punctajul este divizat dupa cum urmeaza:

- 50p - trecerea testelor
- 30p - corectitudinea implementarii
- 20p - claritatea codului si a explicatiilor din README
- 20p - Bonus

Prin corectitudinea implementarii se intelege regasirea elementelor de sincronizare potrivite fiecarui caz din cele mentionate anterior. **NU** se vor considera corecte solutiile care **forteaza** sincronizarea prin **sleep-uri** si **yield-uri**. Sleep poate fi folosit numai in situatiile mentionate in enunt, in care este impus un timp de asteptare.