# Tema 1

## Cel mai scurt drum între două puncte

Responsabili: Irina Toma

### Introducere

Scopul acestei teme este de a vă familiariza cu programarea orientată pe obiecte și cu limbajul Java. În același timp, vă puteți aprofunda cunoștințele cu privire la o structuri de date folosite în practică, cum ar fi grafurile sau cozile de prioritate.

### Descrierea problemei

O companie care produce sisteme de navigație pentru mașini vă cere să implementați un program pentru găsirea celui mai scurt drum între două puncte și timpul în care acesta e parcurs. Pentru implementare vă hotărâți să folosiți algoritmul **Dijkstra**.

Compania vă pune la dispoziție harta străzilor, informații despre trafic, restricții de circulație și autovehiculul condus de client.

Vehiculele conduse de clienți pot fi biciclete (b), motociclete (m), autoturisme (a), camioane (c) și au următoarele proprietăți:

| **Nume proprietate** | **Bicicletă** | **Motocicletă** | **Autoturism** | **Camion** |
|---|---|---|---|---|
| Tip | Moped | Moped | Autovehicul | Autoutilitar |
| Gabarit | 1 | 1 | 2 | 3 |
| Cost | 1 | 2 | 4 | 6 |

Harta orașului este reprezentată de o serie de puncte și străzi ce se pot transpune sub forma unui graf în care punctele sunt nodurile, iar străzile sunt muchiile. Fiecare nod are asociat un nume. Fiecare stradă are asociate nume, cost și o serie de restricții de circulație.

Restricțiile de circulație sunt:

- Restricții de viteză – proprietăți: cost suplimentar
- Restricții de gabarit – proprietăți: limită gabarit
- Ambuteiaje – proprietăți: tip ambuteiaj (accident, trafic, blocaj), cost asociat

Atenție:

- Restricțiile de viteză și de gabarit sunt disponibile la crearea hărții, dar ambuteiajele pot apărea pe parcurs
- Ambuteiajul este asociat unei singure străzi, dar pot fi mai multe ambuteiaje pe aceeași stradă
- Străzile au sens unic. P1 P2 reprezintă o strada cu sens unic de la P1 la P2
- O mașină poate circula pe stradă dacă are gabaritul mai mic sau egal cu limita de gabarit de pe stradă
- Costul unei mașini de a circula pe o stradă este egal cu cost_stradă*cost_mașină + cost_ambuteiaj1 + ... + cost_ambuteiajN

## Testare

Tema va fi testată pe **VMChecker** (va apărea în curând), astfel încât va trebui să aveți și un *makefile* cu o regulă *run* care să ruleze o clasă care conține metoda „main”, pentru a ușura testarea automată.

Fișierul de intrare se va numi *map.in*, iar cel de ieșire *map.out*.
Fișierul map.out trebuie să conțină rezultatul comenzilor *drive*, anume cel mai scurt drum intre două puncte, respectiv timpul necesar pentru a ajunge la destinație.

Exemplu:

În map.in veți primi o serie de comenzi structurate în felul următor: număr străzi, detalii străzi, listă comenzi.

| **Valori** | **Descriere** |
|---|---|
| 6 5 | Număr_total_străzi număr_total_noduri |
| P0 P1 1 3 | Punct_start punct_destinație cost limita_gabarit |
| P0 P2 2 1 | |
| P1 P2 2 3 | |
| P1 P3 1 2 | |
| P3 P4 3 2 | |
| P4 P3 1 3 | |
| accident P0 P1 10 | Tip_ambuteiaj punct_start punct_end cost |
| drive a P0 P4 | Afișează drumul pentru un autoturism din punctul P0 spre P4 |
| drive b P0 P2 | Afișează drumul pentru o bicicleta din punctul P0 spre P2 |
| trafic P1 P3 5 | Tip_ambuteiaj punct_start punct_end timp_întârziere |
| blocaj P0 P1 10 | Tip_ambuteiaj punct_start punct_end timp_întârziere |
| drive c P0 P4 | Afișează drumul pentru un camion din punctul P0 spre P4 |

În urma comenzilor *drive* se va în fișierul map.out câte o linie ce conține: punctele prin care trece drumul si costul total al drumului.

map.out conține:
P0 P1 P3 P4 30
P0 P2 2
P0 P4 null

Explicații:

- drive a P0 P4
  - cost(P0, P1) + cost (P1, P3) + cost (P3, P4) = 1 * 4 + 10 + 1 * 4 + 3 * 4 = 14 + 4 + 12 = 30
- drive b P0 P2
  - cost(P0, P2) = 2 * 1 = 2
- drive c P0 P4
  - cost(P0, P4) = null; Camionul nu poate trece pe strada P1 – P3 pentru ca are gabaritul prea mare

## Cerințe

1. Să se modeleze conform principiilor programării orientate pe obiecte entitățile din aplicație (de exemplu, Stradă, Bicicletă, Autoturism etc.).
    - Trebuie să folosiți conceptele **moștenire, polimorfism** și **abstractizare**
    - Explicați în README alegerile pe care le-ați făcut
    - Pentru o modelare corecta, folosiți ca exemple informațiile prezentate în [1]
2. Să se implementeze o structură de date (graf) pentru stocarea hărții și o modalitate de a găsi drumul cu cel mai mic cost între două puncte. Graful trebuie să implementeze următoarele metode:
    - addStreet(start, end, cost, size)
    - addRestriction(type, start, end, cost)
    - drive(vehicle, start, end)
3. Să se ruleze linie cu linie comenzile primite în fișierul map.in
4. Să se afișeze outputul cerut.

## Punctaj

- 4.5p teste publice
- 4.5p teste private
- 1p Javadoc
- 2p bonus

## Depunctări

- 2p – crearea mai multor grafuri pentru tipurile de vehicule existente
- 1p – code styling
- 1p – lipsa metodelor addStreet, addRestriction și drive
- 1p – lipsa explicațiilor din README
- Până la 3p pentru modelarea incorectă a entităților din aplicație

## Bonus

- 1p – implementarea algoritmului Dijkstra folosind PriorityQueue din Java [2]
- 2p – implementarea algoritmului Dijkstra folosind o coadă de prioritate scrisă de voi [3]

## Referințe

[1] https://docs.oracle.com/javase/tutorial/java/IandI/index.html
[2] https://docs.oracle.com/en/java/javase/11/docs/api/java.base/java/util/PriorityQueue.html
[3] https://www.hackerearth.com/practice/notes/heaps-and-priority-queues/