Graphs & Networks GogolCar Report **Benjamin Bertrand**
M1 ATAL, ORO Due Date: 30/11/2016
F.S.T. de Nantes Prof. Irena Rusu

Problem Statement

3 algorithmes de parcours de graphes, pour simuler le parcours d'une voiture dans une ville. Les rues sont des aretes non-oriente et peuvent parfois etre considerer comme deux arcs orientes. les rues sont delimite par des places, les noeuds du graphe.

Pour ce projet, j'utilise les lambdas java 8, ainsi que l'API Stream qui permet de faire des operations ensembliste interessantes et simplifie grandement le code.

1 Mapper en listes dádjacence

```
Map<Square, List<Street>> adjacentStreet() {
    return getStreets().collect(
        toMap(
            st -> st.sq1,
            st -> Stream.of(st).collect(toList()),
            (list1, list2) -> Stream.concat(list1.stream(),list2.stream()).collect(toList())
        ));
}
```

On cherche par exemple a associer chaque sommet au rues qui lui sont adjacentes. Pour cela, on prends la liste des rues, on extrait les sommets et les agregent selon les regles suivante.

- On prend la clef de la map comme etant l'un des sommet de la rue (toujours le meme sq1).

  ```
  st -> st.sq1
  ```

- On prend la rue elle meme comme valeur correspondant a la clef. Mais sous forme de liste pour permetre l'agregation.

  ```
  st -> Stream.of(st).collect(toList())
  ```

- On definit la regles pour les cas ou la clef apparait plusieurs fois, ici une concatenation des collections.

  ```
  (list1, list2) -> Stream.concat(list1.stream(),list2.stream())
                          .collect(toList())
  ```

2 Calcul du degre de chaque sommets

En utilisant un Stream de Street on peut reduire le set entier en une valeur:

```
long nbDegreImpair() {
    return getStreets()
                .flatMap(street -> Stream.of( street.sq1, street.sq2 ))
                .collect(toMap(s->s, s -> 1, Integer::sum))
                .entrySet()
                .stream()
                .filter(ent -> (ent.getValue().longValue() % 2) == 1)
                .count();
    }
```

- Convertit une Street en un Stream de Square possedant toute les occurences de sommet comme extremite d'une rue.

  ```
  street -> Stream.of( street.sq1, street.sq2 )
  ```

- Map collector, on associe 1 a chaque sommet puis on les cumul par somme lorsque le sommet apparait plusieurs fois.

  ```
  toMap(s->s, s -> 1, Integer::sum)  // Map<Square,Integer>
  ```

- Pour chaque clef-valeur, On filtre les sommets impairs uniquement puis on en compte la somme.

  ```
  .entrySet()  // Set<Entry<Square,Integer>>
  .stream()    // Stream<Entry<Square,Integer>>
  .filter(ent -> (ent.getValue().longValue() % 2) == 1)
  .count();
  ```

GogolS

Pour l'algo S on a juste a prendre la map d'adjacence et avancer tant qu'il reste des element dedant. en prenant soin d'enlever les noeuds qui n'ont plus d'arcs sortant

```
int step=0;
do{
    List<Street> adjL = adjM.get(current);
    Street street = adjL.remove(0);
    path.add(street);

    if(adjL.isEmpty()){
        adjM.remove(current);
    }
    current=street.sq2;

    street.mark("step " + step++);
}while(!adjM.isEmpty());
```

GogolL

Pour le second algo:

- on calcul d'abords une arborescence quelconque (sous la forme d'une liste de rue).

```
public Path arborescence(Square current, Path pathTaken) {
    if (pathTaken.size() == city.getSquares().count())
        return pathTaken;

    List<Street> streetsOut = city.adjacentStreet().get(current);

    for (Street street : streetsOut) {
        if (!pathTaken.contains(street.sq2)) {
            pathTaken = arborescence(street.sq2, pathTaken.drive(street));
        }
    }
    return pathTaken;
}
```

- on numerote les rues grace a cette arborescence.

```
public void numerotation(Path arbo) {
    List<Street> antiArbo = city.oposingArcs(arbo);

    city.adjacentStreet().forEach((sq, list) -> {
        int degre = city.degreOfX().get(sq);
        list.sort((s1,s2)->{
            int res = 0;
            if(arbo.contains(s1)) res+=2;
            if(arbo.contains(s2)) res-=2;
            if(antiArbo.contains(s1)) res-=1;
            if(antiArbo.contains(s2)) res+=1;
            return res;
        });

        for(Street t : list){
            t.pos=degre--;
        }
    });
}
```

- On parcours en partant de current et en prenant le plus petit sommet.pos attribut

```
Path usedStreet = new Path();
for (int step=1; step<=city.getStreets().count()/2;step++) {
    Street next = adjM.get(current)
                        .stream()
                        .filter(s -> !usedStreet.contains(s.name))
                        .sorted((s1, s2) -> s1.pos.compareTo(s2.pos))
```

```
                            .findFirst()
                            .get();

        usedStreet.add(next);
        next.step = step;
        current = next.sq2;
    }
```

GogolXL

Conclusion

References