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 artes non-oriente et peuvent parfois tre considrer comme deux arcs orientes. les rues sont dlimit par des places, les nuds du graphe.

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

1 Mapper en listes d'adjacence

```
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 agrgent selon les rgles suivante.

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

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

- On prend la rue elle mme comme valeur correspondant a la clef. Mais sous forme de liste pour permettre lagrgation.

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

- On dfinit la rgles pour les cas ou la clef apparat plusieurs fois, ici une concatnation des collections.

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

2 Calcul du degr de chaque sommets

En utilisant un Stream de Street on peut rduire 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 possdant toute les occurrences de sommet comme extrmit 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 apparat 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 lment dedans. en prenant soin d'enlever les nuds 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 numrote les rues grce 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(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

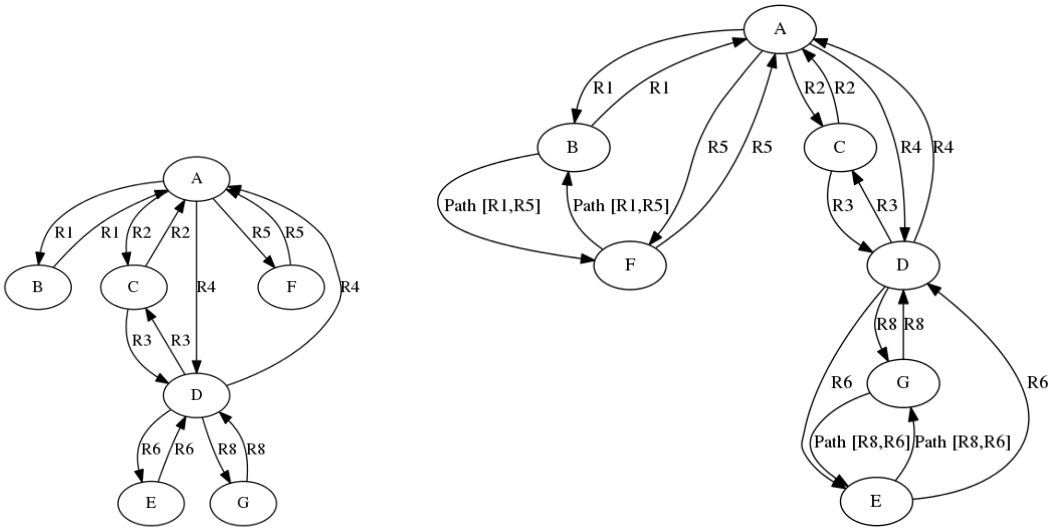
```
Path usedStreet = new Path();
for (int step=1; step<=city.getStreets().count()/2;step++) {
    Street next = adjM.get(current)
                        .stream()
                        .filter(s -> !usedStreet.contains(s.name))
                        .sorted((s1, s2) -> s1.pos.compareTo(s2.pos))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

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

GogolXL

Pour l'algo XL, le problme est qu'il n'est pas Eulerien et donc il est impossible de ne pas re-emprunter au moins une arete.

Nous chercherons donc a la rendre Eulerien en suivant la mthode du postier chinois. en connectant les sommets de degrs impair par des arcs virtuel reprsentant le chemin le plus cours entre ces nuds



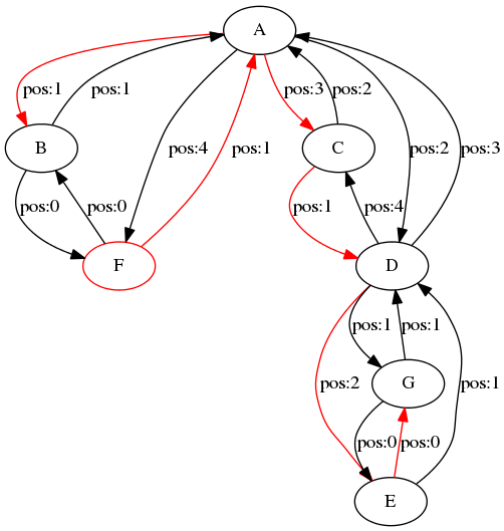
(a) GogolXL raw input graph

(b) GogolXL with virtual edges

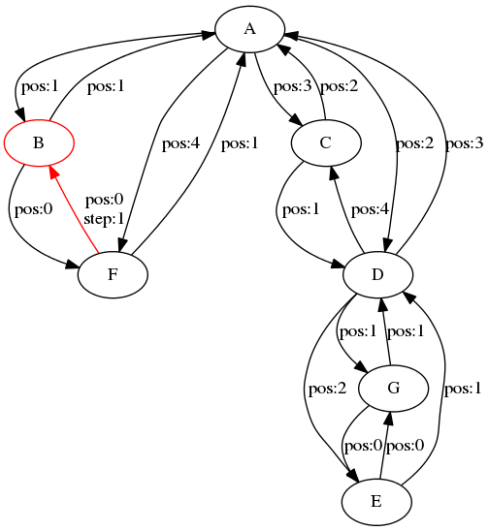
Figure 1: Etat initial

Conclusion

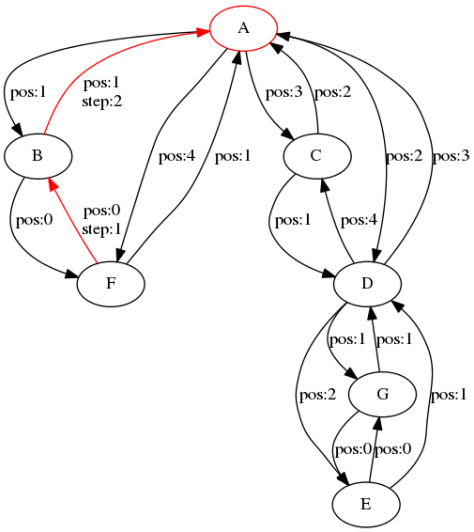
Stream API nous permet dcrire ce que nous voulons obtenir sans prciser comment on souhaite l'obtenir. ce qui permet a la VM de construire elle mme lexcution la plus adapte.



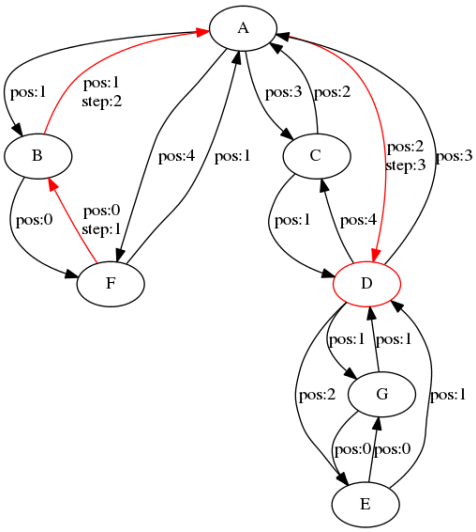
(a) Step 0



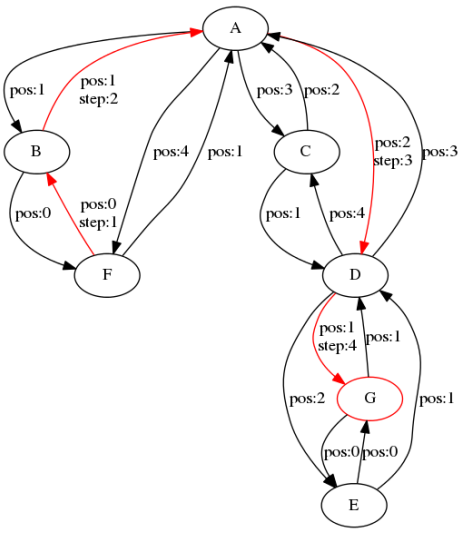
(b) Step 1



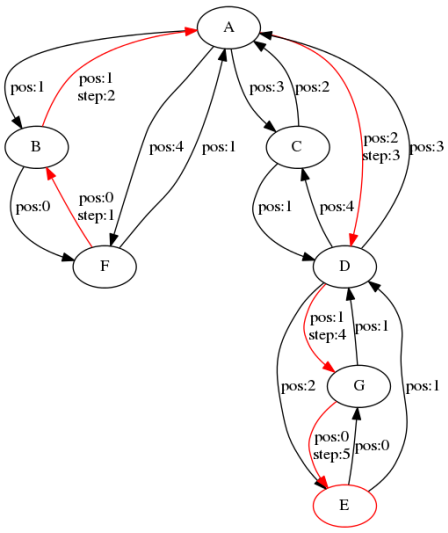
(c) Step 2



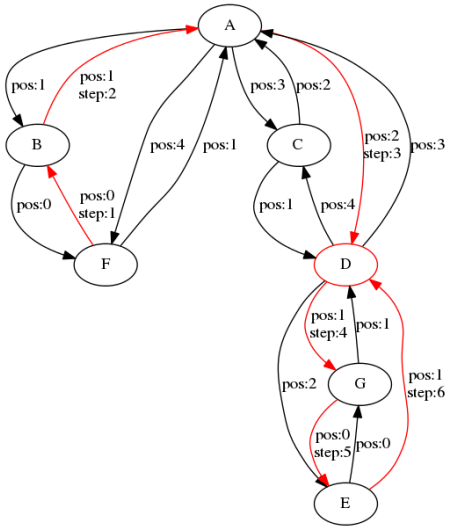
(d) Step 3



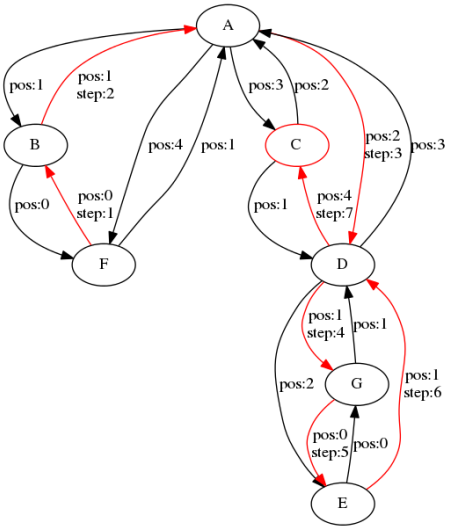
(a) Step 4



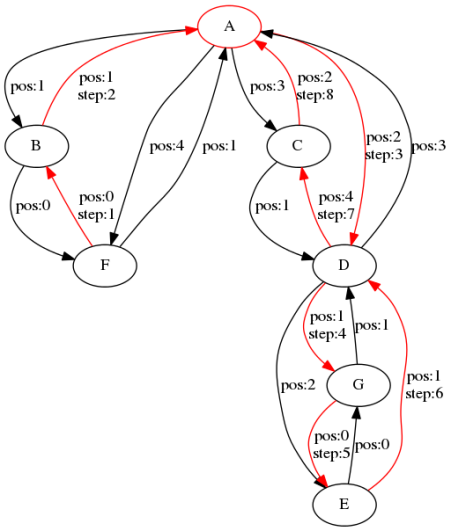
(b) Step 5



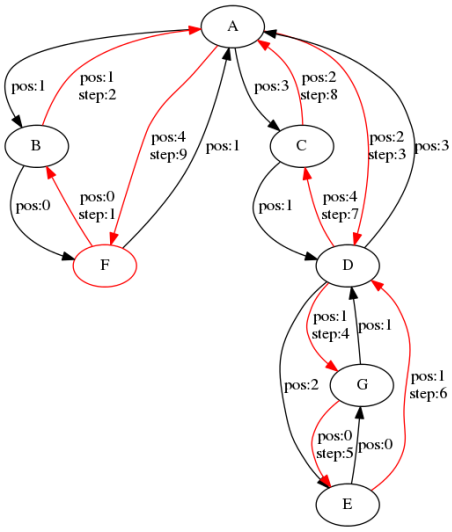
(c) Step 6



(d) Step 7



(e) Step 8



(f) Step 9

Figure 2: Execution