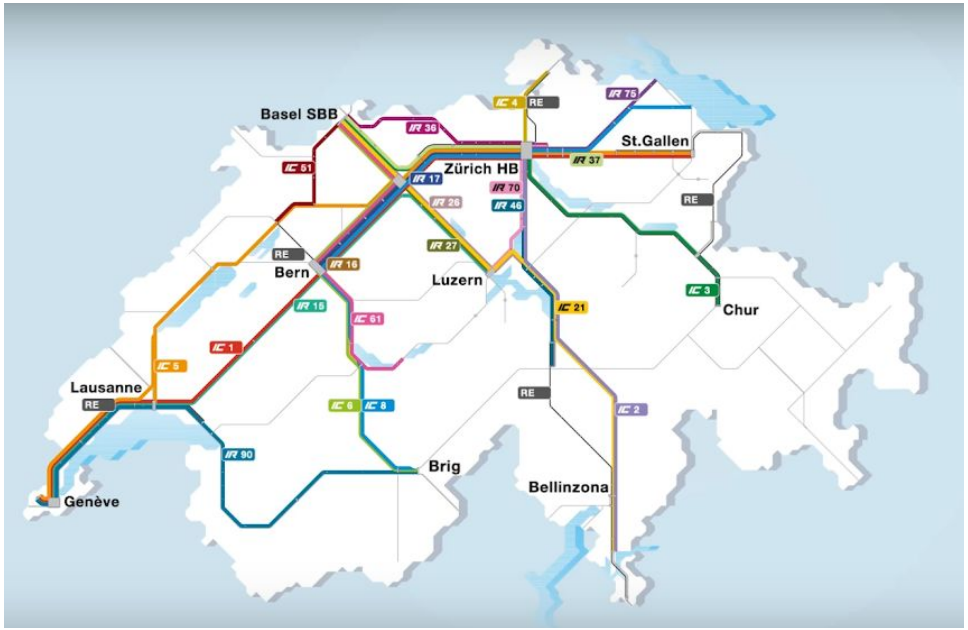
Travail Pratique : C-FF et plus courts chemins

Mardi 16 Juin 2020

Thomas Dagier

Généralités sur le projet : objectif général

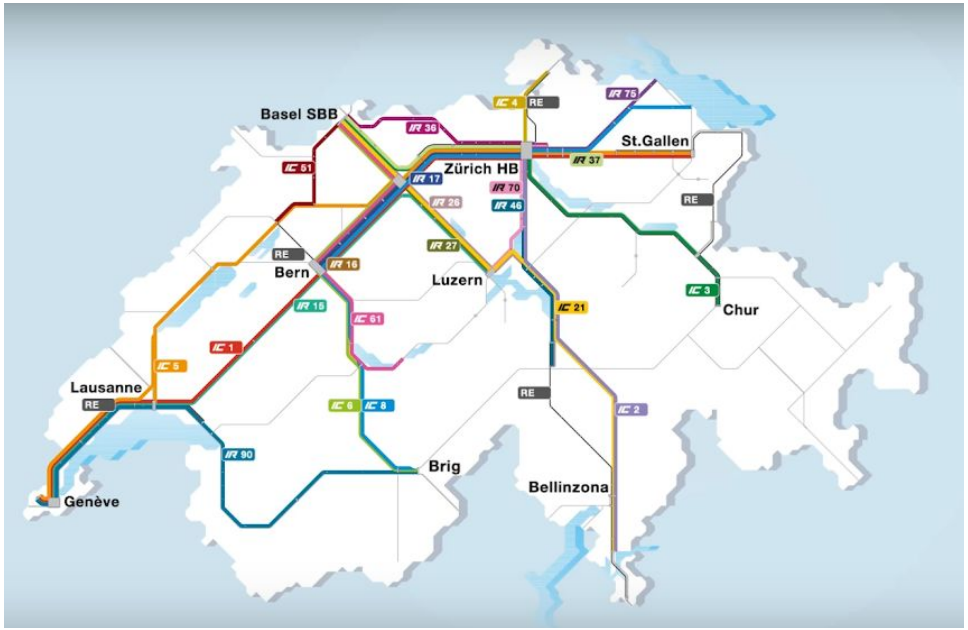
Trouver le plus court chemin entre deux villes ou un parcours à emprunter



Généralités sur le projet : objectif général

Trouver le plus court chemin entre deux villes ou un parcours à emprunter

Structures de données et approches algorithmiques qui reprennent le cours

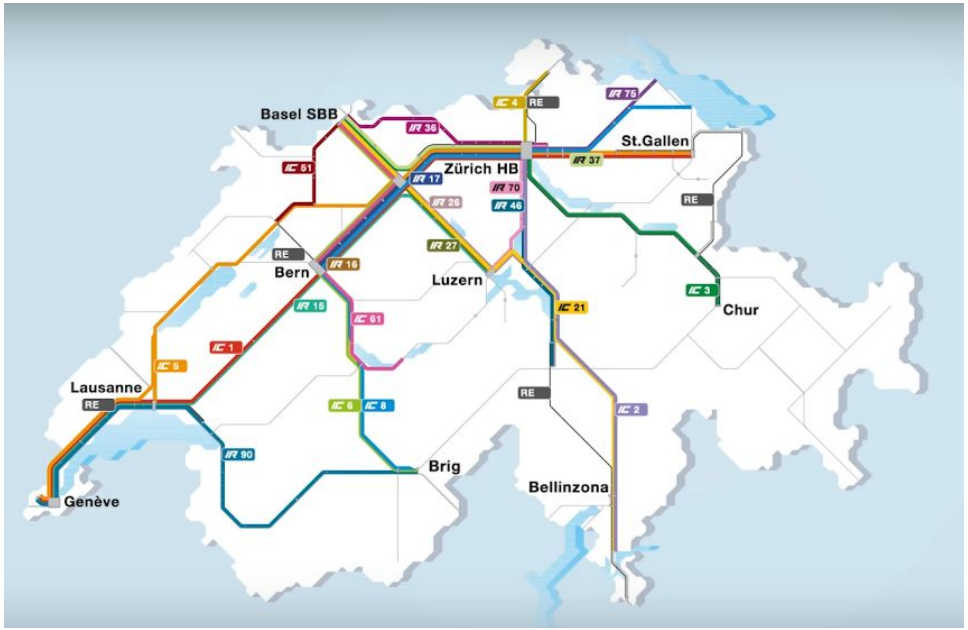


Généralités sur le projet : objectif général

Trouver le plus court chemin entre deux villes ou un parcours à emprunter

Structures de données et approches algorithmiques qui reprennent le cours

Consolider les connaissances en programmation et structures de données



I/ Présentation du code

- structures de données et algorithmes de plus courts chemins
- difficultés rencontrées
- solutions proposées

I/ Présentation du code

- structures de données et algorithmes de plus courts chemins
- difficultés rencontrées
- solutions proposées

II/ Démonstration

- exécution des tests et du projet terminé

I/ Présentation du code

- structures de données et algorithmes de plus courts chemins
- difficultés rencontrées
- solutions proposées

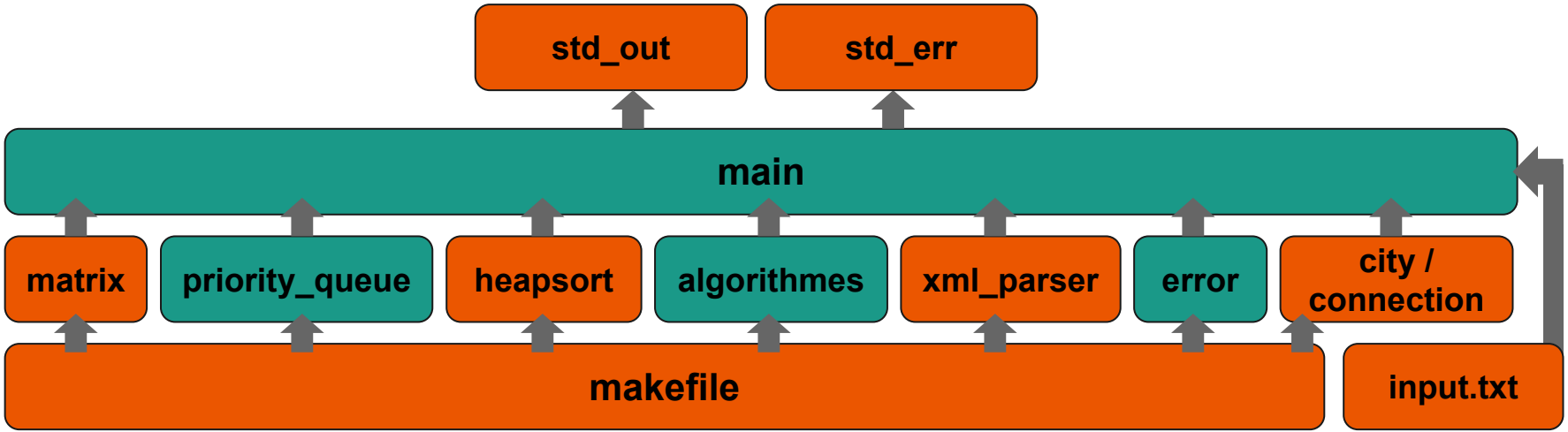
II/ Démonstration

- exécution des tests et du projet terminé

III/ Conclusion

- retour sur expérience, compétences acquises

Présentation du code : Vue globale du projet



Présentation du code : Les structures de données

```
struct element{
    void * data
    struct element * next
}

struct priority_queue {
    int length
    struct element * head
}
```

Présentation du code : Les structures de données

```
struct element{
    void * data
    struct element * next
}

struct priority_queue {
    int length
    struct element * head
}
```

```
struct matrix {
   int m,n
   int ** data
}
```

Présentation du code : Les structures de données

```
struct element{
    void * data
    struct element * next
}

struct priority_queue {
    int length
    struct element * head
}
```

```
struct matrix {
   int m,n
   int ** data
}

struct heap_queue{
   int* data
   int length
   int capacity
}
```

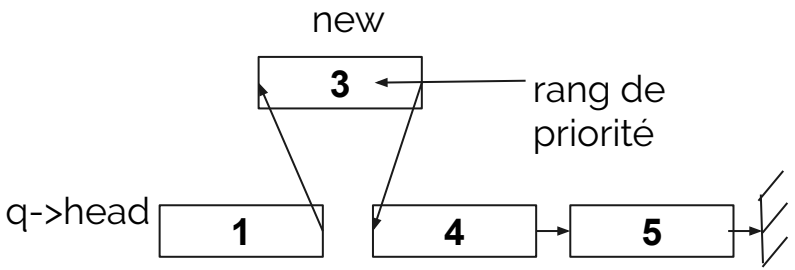
Présentation du code : Les structures de données

```
struct element{
    void * data
    struct element * next
}

struct priority_queue {
    int length
    struct element * head
}
```

```
struct matrix {
   int m,n
   int ** data
}

struct heap_queue{
   int* data
   int length
   int capacity
}
```



Présentation du code : Dijkstra, approche technique


```
struct city {
   char * name
   int longitude
   int latitude
   struct city * next
}
```

```
struct connexion {
   int neighbor
   int weight
   struct connexion* next
}

struct lst_connexion{
   int length
   struct connexion* head
}
```

Présentation du code : Dijkstra, difficultés rencontrées

comprendre - appliquer - faire fonctionner

Présentation du code : Dijkstra, difficultés rencontrées

comprendre - appliquer - faire fonctionner

utilisation correcte des pointeurs génériques

soucis de pseudo-code

Présentation du code : Dijkstra, solution proposée

parser le fichier xml - temps de parcours entre Genève et Berne

visiter src

- marquer tous les {(src,dst) = weight} dans Q
- dépiler la première connexion de Q
- modifier le poids de la connexion si on en trouve un plus faible
- ajouter la connexion dans dans T

recommencer avec dst tant que Q n'est pas vide

Présentation du code : Dijkstra, solution proposée

parser le fichier xml - temps de parcours entre Genève et Berne

visiter src

marquer tous les {(src,dst) = weight} dans Q

dépiler la première connexion de Q

modifier le poids de la connexion si on en trouve un plus faible

ajouter la connexion dans dans T

recommencer avec dst tant que Q n'est pas vi

Présentation du code : Dijkstra, solution proposée

parser le fichier xml - temps de parcours entre Genève et Berne

visiter src

marquer tous les {(src,dst) = weight} dans Q

dépiler la première connexion de Q

modifier le poids de la connexion si on en trouve un plus faible

ajouter la connexion dans dans T

recommencer avec dst tant que Q n'est pas vi

Présentation du code : Dijkstra, solution proposée

34 < ∞

parser le fichier xml - temps de parcours entre Genève et Berne

visiter src

marquer tous les {(src,dst) = weight} dans Q

dépiler la première connexion de Q

modifier le poids de la connexion si on en trouve un plus faible

ajouter la connexion dans dans T

recommencer avec dst tant que Q n'est pas vi

Présentation du code : Dijkstra, solution proposée

parser le fichier xml - temps de parcours entre Genève et Berne

visiter src

- marquer tous les {(src,dst) = weight} dans Q
- dépiler la première connexion de Q
- modifier le poids de la connexion si on en trouve un plus faible

ajouter la connexion dans dans T

recommencer avec dst tant que Q n'est pas vide

Présentation du code : Dijkstra, solution proposée

parser le fichier xml - temps de parcours entre Genève et Berne

visiter src

- marquer tous les {(src,dst) = weight} dans Q
- dépiler la première connexion de Q
- modifier le poids de la connexion si on en trouve un plus faible
- ajouter la connexion dans dans T

recommencer avec dst tant que Q n'est pas vide

Présentation du code : Dijkstra, solution proposée

parser le fichier xml - temps de parcours entre Genève et Berne

visiter src

- marquer tous les {(src,dst) = weight} dans Q
- dépiler la première connexion de Q
- modifier le poids de la connexion si on en trouve un plus faible
- ajouter la connexion dans dans T

recommencer avec dst tant que Q n'est pas vide

Présentation du code : Dijkstra, solution proposée

parser le fichier xml - temps de parcours entre Genève et Berne

visiter src

marquer tous les {(src,dst) = weight} dans Q

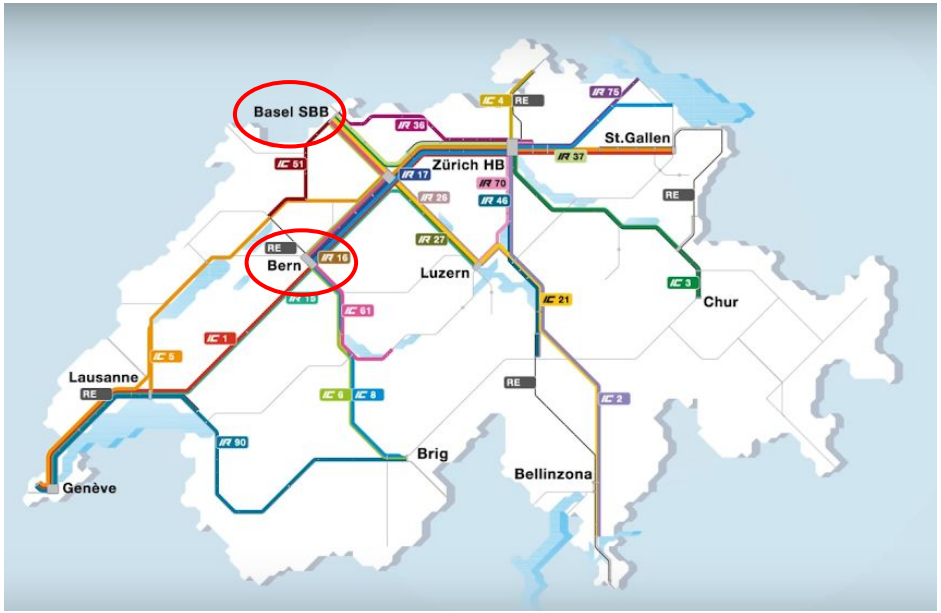
dépiler la première connexion de Q

modifier le poids de la connexion si on en trouve un plus faible

ajouter la connexion dans dans T

recommencer avec dst tant que Q n'est pas vide

Présentation du code : Dijkstra, solution proposée



parser le fichier xml - temps de parcours entre Genève et Berne

visiter src

marquer tous les {(src,dst) = weight} dans Q

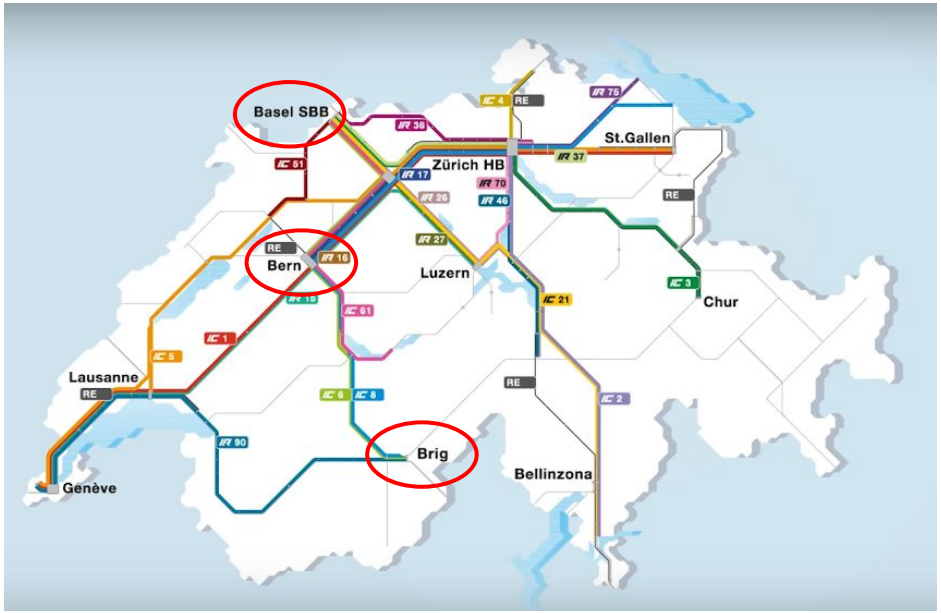
dépiler la première connexion de Q

modifier le poids de la connexion si on en trouve un plus faible

ajouter la connexion dans dans T

recommencer avec dst tant que Q n'est pas vide

Présentation du code : Dijkstra, solution proposée



parser le fichier xml - temps de parcours entre Genève et Berne

visiter src

marquer tous les {(src,dst) = weight} dans Q

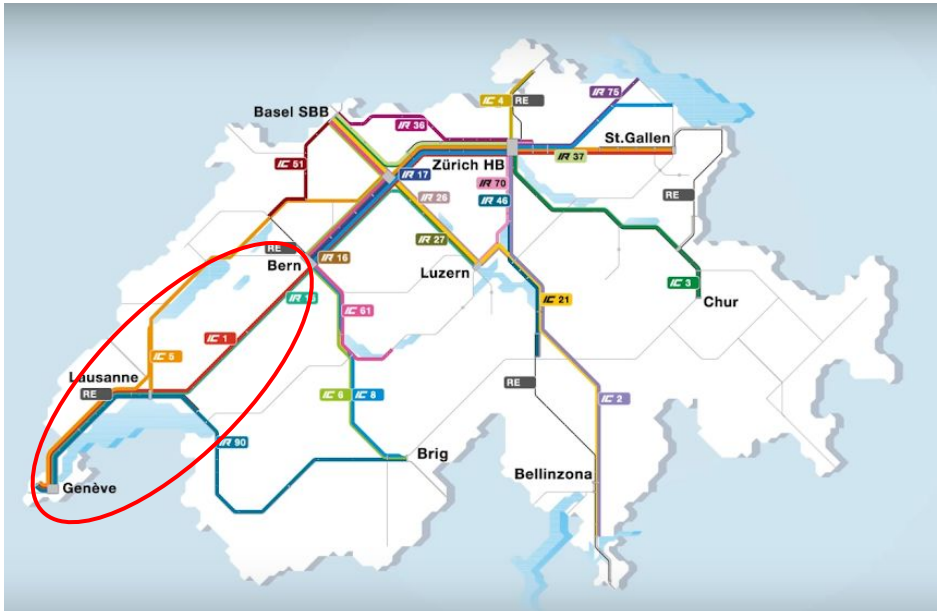
dépiler la première connexion de Q

modifier le poids de la connexion si on en trouve un plus faible

ajouter la connexion dans dans T

recommencer avec dst tant que Q n'est pas vide

Présentation du code : Dijkstra, solution proposée



parser le fichier xml - temps de parcours entre Genève et Berne

dst (Genève-Berne) = dst (Genève-Lausanne)

\+ dst (Lausanne-Berne)

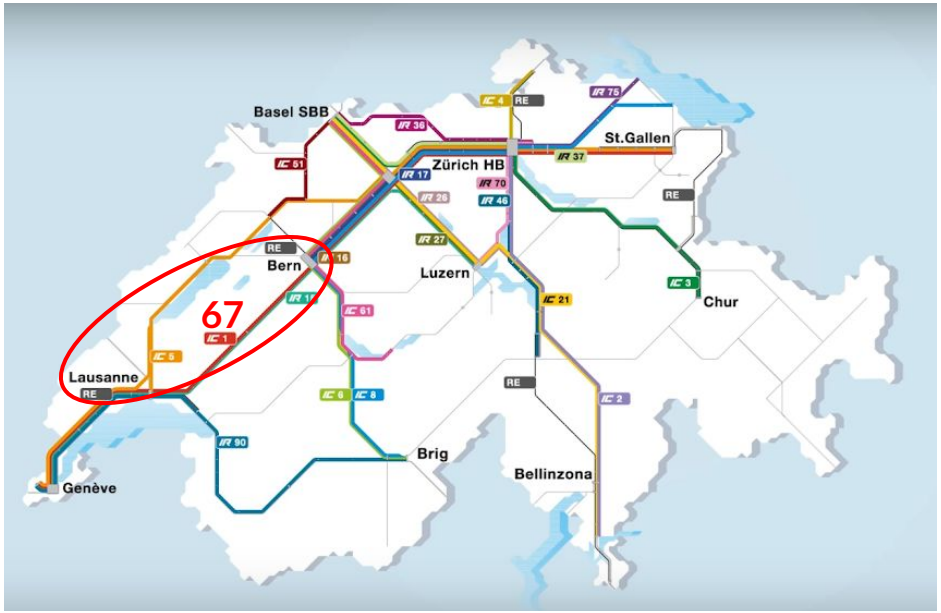
Présentation du code : Dijkstra, solution proposée

parser le fichier xml - temps de parcours entre Genève et Berne

dst (Genève-Berne) = **dst (Genève-Lausanne)**

\+ dst (Lausanne-Berne)

Présentation du code : Dijkstra, solution proposée

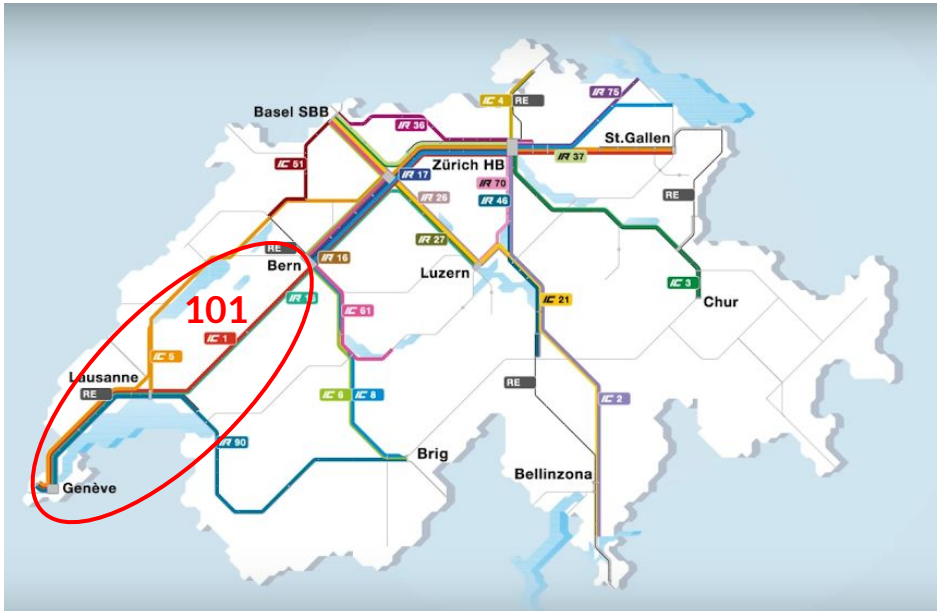


parser le fichier xml - temps de parcours entre Genève et Berne

dst (Genève-Berne) = dst (Genève-Lausanne)

+ dst (Lausanne-Berne)

Présentation du code : Dijkstra, solution proposée



parser le fichier xml - temps de parcours entre Genève et Berne

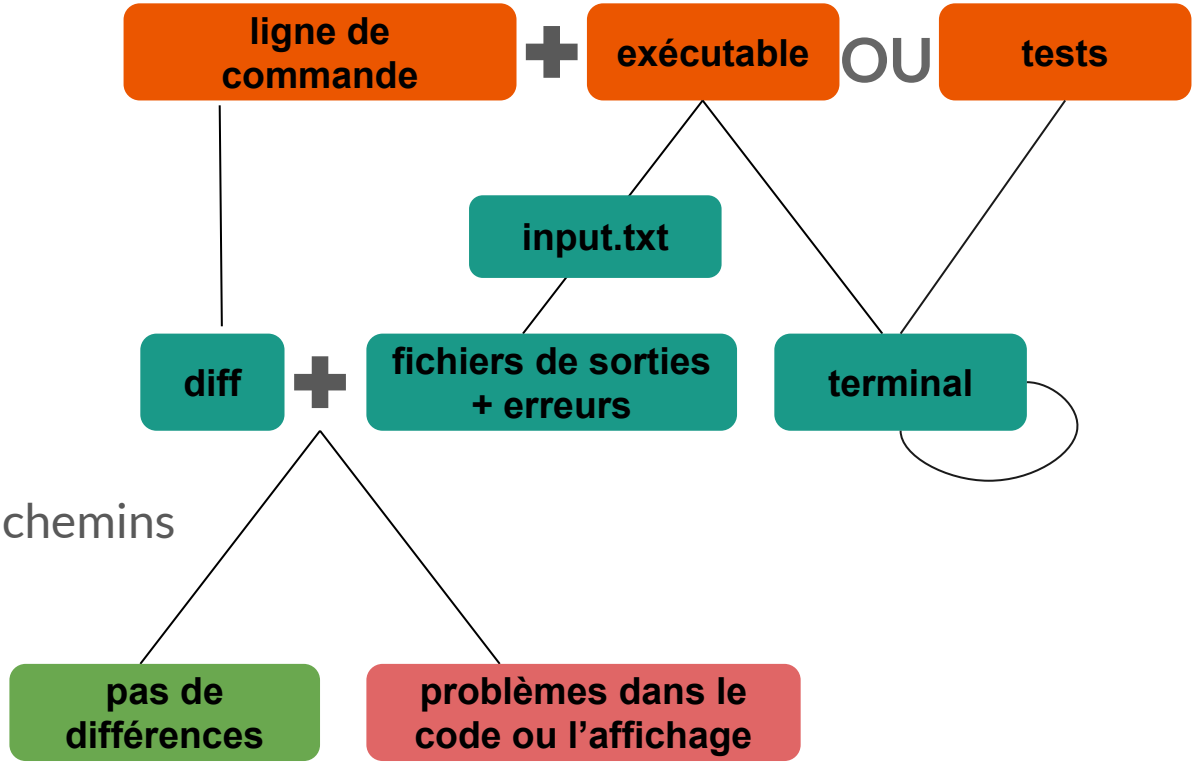
dst (Genève-Berne) = dst (Genève-Lausanne)

\+ dst (Lausanne-Berne)

Démonstration

exécutions des tests

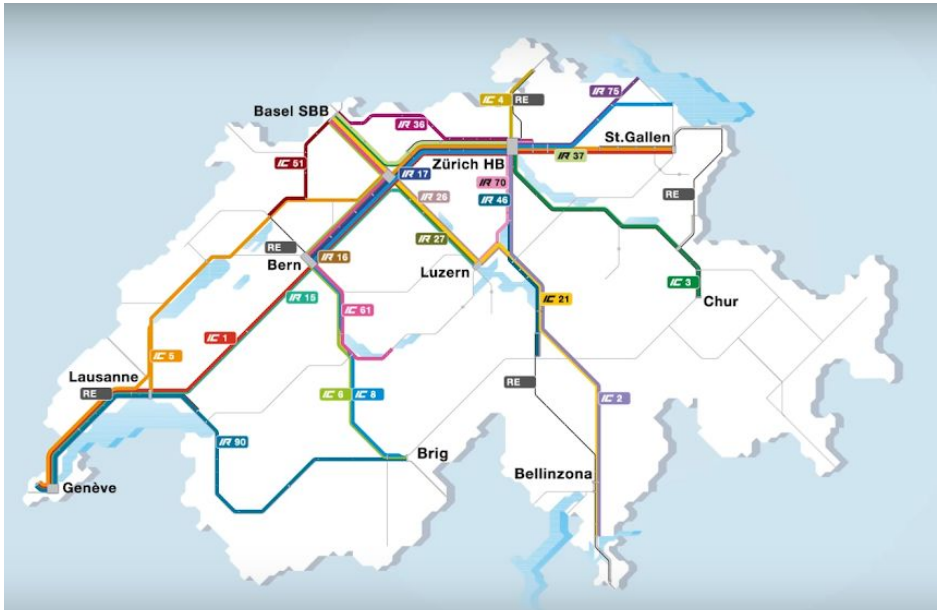
application de plus courts chemins



Conclusion

Retour sur expérience

Compétences acquises



Questions