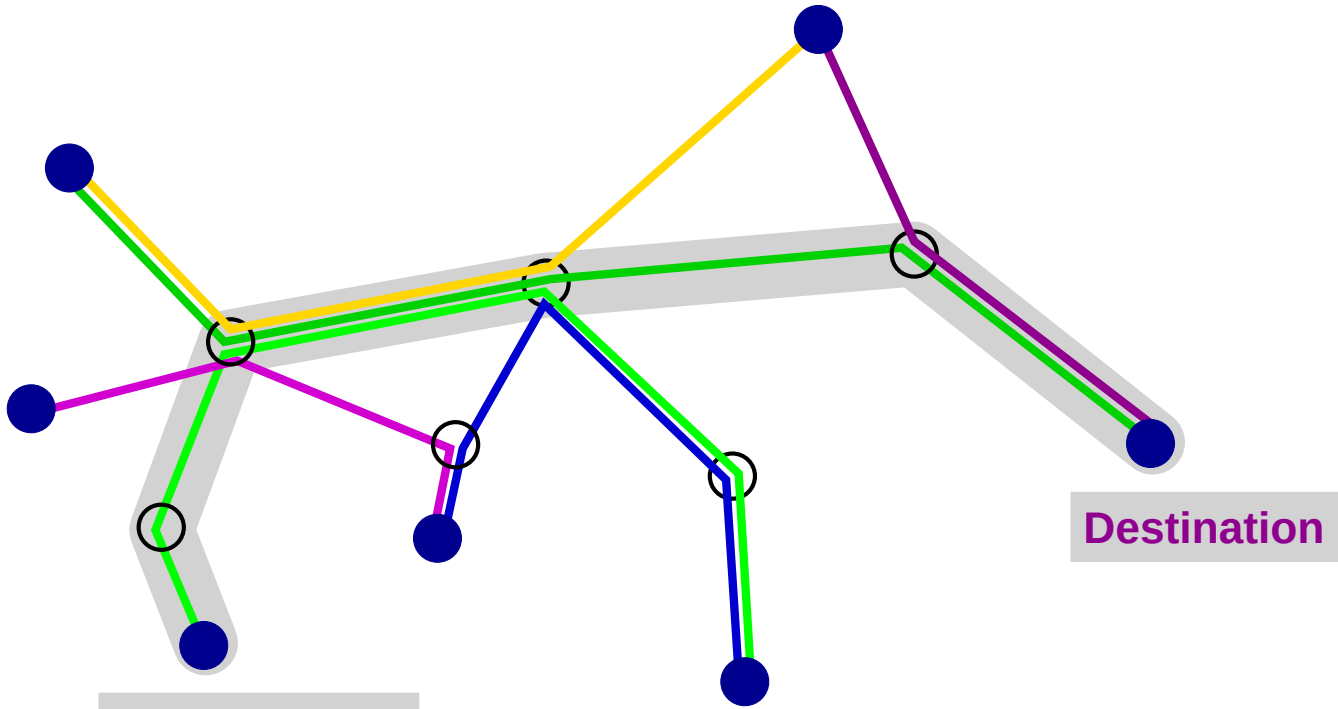
Calcul d’itinéraires ferroviaires.



Déroulement du projet

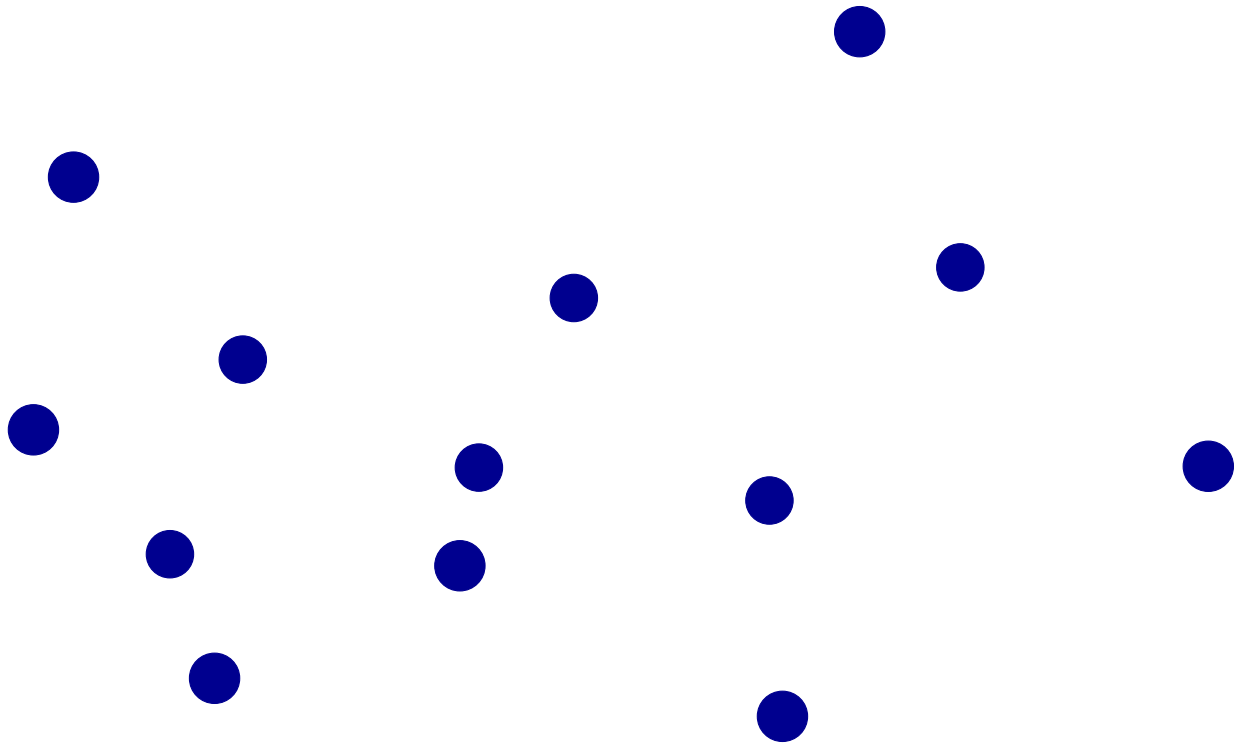
- **Etape 1 : calcul d'itinéraires ferroviaires.**
  - lecture des données
  - construction des structures de données
  - calcul d'itinéraires optimaux (algorithme de Dijsktra)
  - **16/17 février** : rapport intermédiaire
- **Etape 2 : tests, durées moyennes, clustering.**
  - génération aléatoire d'instances et tests
  - calcul des durées moyennes des trajets entre deux villes
  - problème de clustering
  - interface graphique
  - **30/31 mars** : soutenance, rapport final

Evaluation

- travail en groupe de une ou deux personnes,
- **Rapport intermédiaire** : 2 pages, 10 min. de présentation.
  Description des structures de données et des principales fonctions.
- **Rapport final** : 5 pages, 15 min. de présentation.
  Description de la méthode, des problèmes rencontrés, des solutions utilisées, résultats, observations,. . .

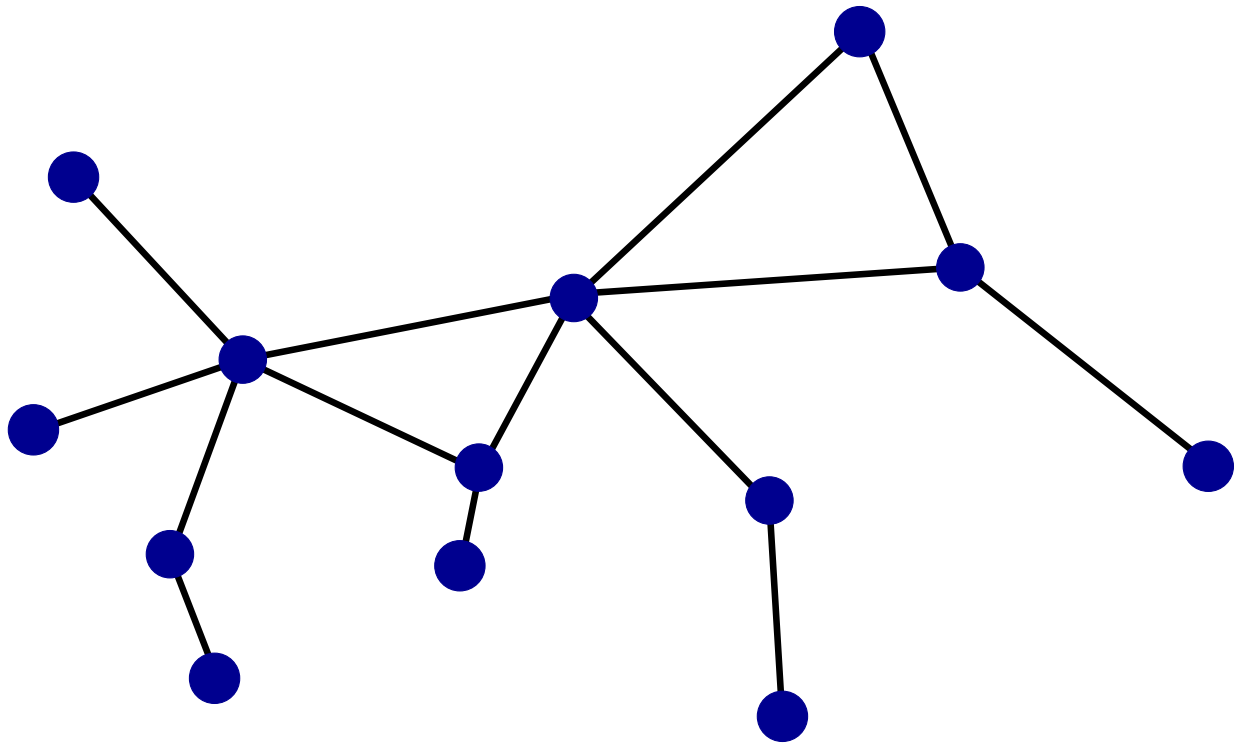
Le réseau ferroviaire.

1. Des villes



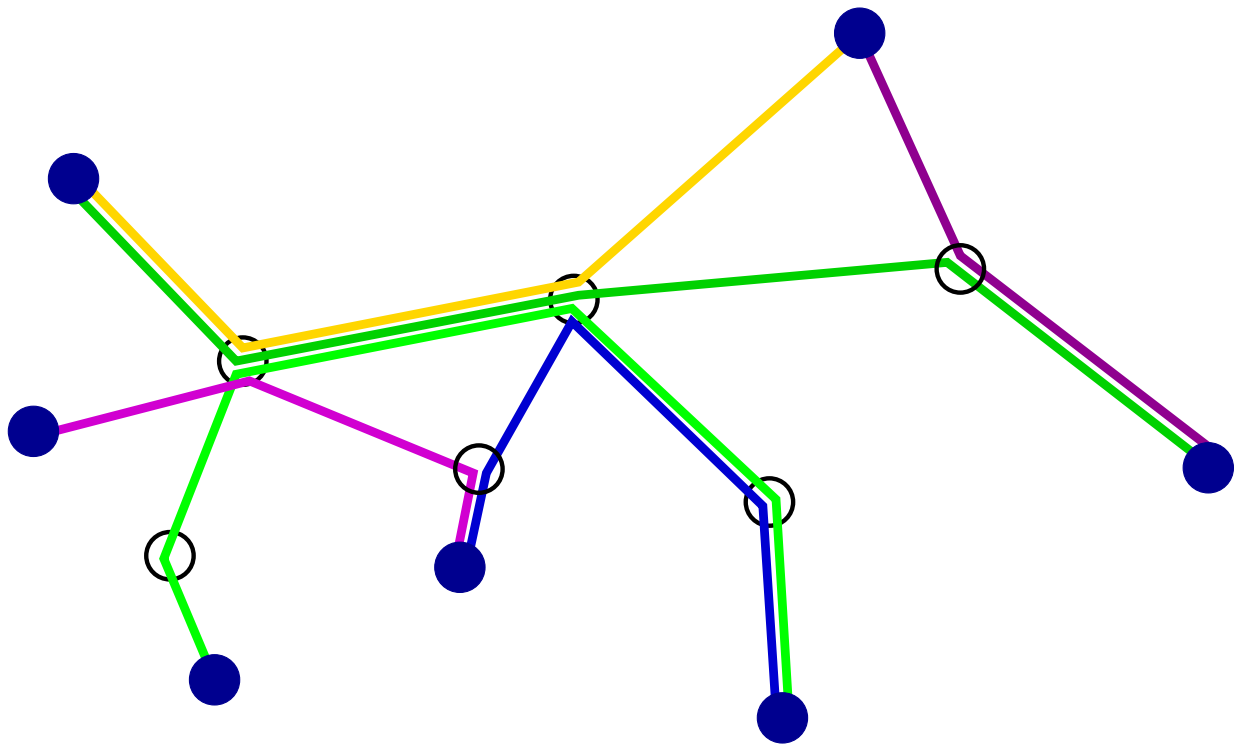
Le réseau ferroviaire.

2. Des voies



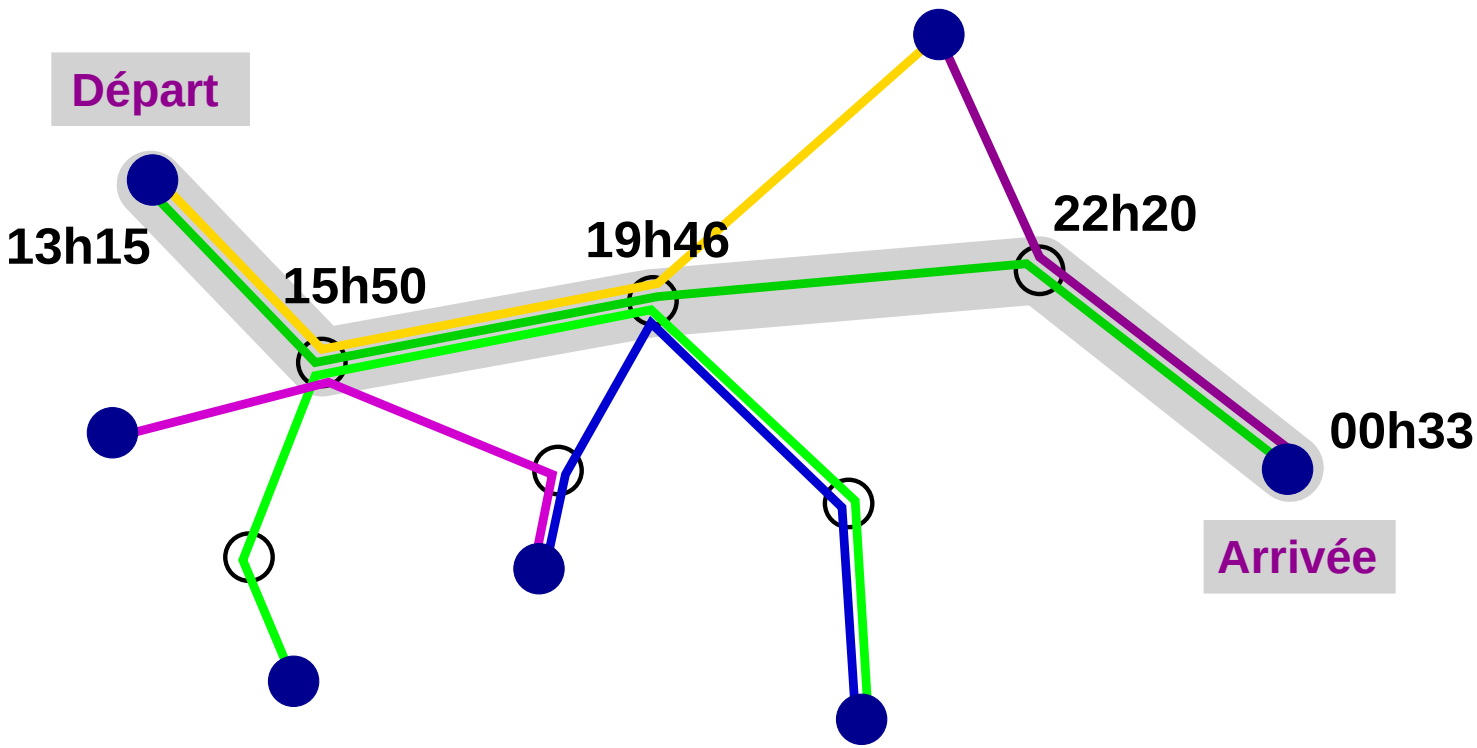
Le réseau ferroviaire.

3. Des lignes



Le réseau ferroviaire.

4. Des horaires



Données.

Syntaxe du fichier de données

```
<nombre de villes n>
<abscisse ville 1> <ordonnée ville 1>
<abscisse ville 2> <ordonnée ville 2>
...
<abscisse ville n> <ordonnée ville n>

<nombre de lignes m>
<definition de la ligne 1>
<definition de la ligne 2>
...
<definition de la ligne m>
```

Données.

Définition d’une ligne :

```
<nombre de villes k>
<ville 1> <ville 2> ... <ville k>

<nombre de passages journaliers p>
<P1 horaire 1> <P1 horaire 2> ... <P1 horaire k>
<P2 horaire 1> <P2 horaire 2> ... <P2 horaire k>
...
<Pp horaire 1> <Pp horaire 2> ... <Pp horaire k>
```

Données : exemple.

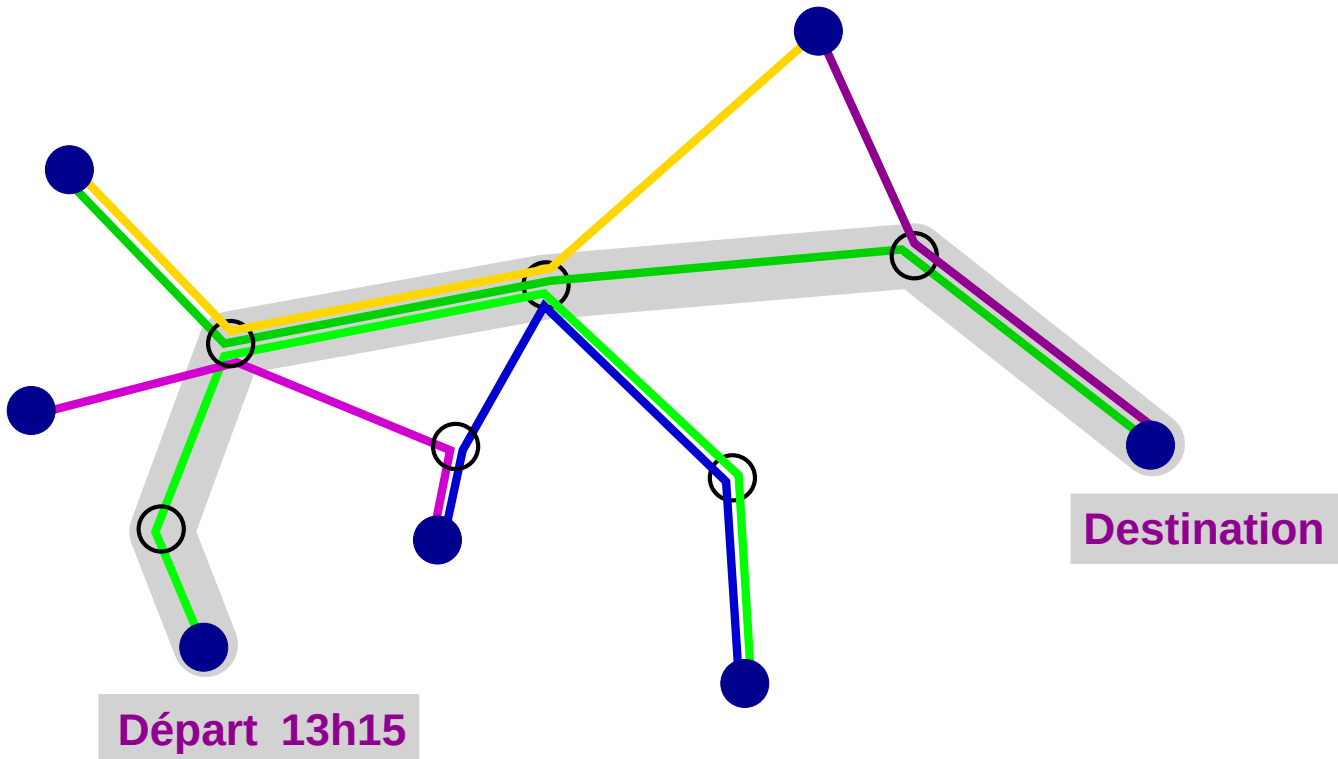
```
10
383 886 777 915 793 335 386 492 649 ... 763 926 540
426 172 736

2
3
1 0 9
2
14h44 18h41 21h16
03h30 07h27 10h02

3
3 8 6
3
20h33 22h13 02h10
05h02 06h42 10h39
12h50 14h30 18h27
```

Données : exemple.

```
10              nombre de villes
383 886 777 915 793 335 386 492 649 ... 763 926 540
426 172 736     coordonnées des villes

2               nombre de lignes

3               nombre de villes dans la première ligne
1 0 9           villes de la première ligne
2               nombre de passages journalier
14h44 18h41 21h16    horaires du 1er passage
03h30 07h27 10h02    horaires du 2è passage

3
3 8 6
3
20h33 22h13 02h10
05h02 06h42 10h39
12h50 14h30 18h27
```

Calcul d'itinéraires ferroviaires.

Problème : trouver l'itinéraire le plus rapide pour aller d'une ville s à une ville t en partant à l'heure h.



Algorithme de Dijkstra.

algorithme DIJKSTRA(G, s)
in : $G = (S, V[], l)$ un graphe pondéré, s un sommet de G,

pour chaque sommet $u \in S$ **faire**
    $d[u] := \infty$,
$d[s] := 0$,
construire un tas T *ordonné selon* d *contenant tous les sommets de* S,
tant que T n'est pas vide **faire**
    $u :=$EXTRAIRE_LE_MIN(T),
    pour chaque sommet $v \in V[u]$ **faire**
        si $d[v] > d[u] + l(u, v)$ **alors**
            $d[v] = d[u] + l(u, v)$,
            $pere[v] = u$,
            mettre à jour le tas T (puisque $d[v]$ a diminué),
    fin pour,
fin tant que,
renvoyer d et $pere$,

Application de Dijkstra au réseau ferroviaire.

Déf. La ville v' est une voisine de la ville v si un arrêt en v' intervient immédiatement après un arrêt en v sur au moins une ligne.

Structures de données.

- pour chaque ville v, calculer la liste de ses voisines.
- pour chaque voisine v' de v, calculer la liste des trajets de v à v' ordonnée dans l'ordre croissant des horaires d'arrivée en v' ;
- un trajet est caractérisé par un horaire de départ, un horaire d'arrivée et un numéro de ligne.

On suppose que le voyageur peut prendre un train qui part au moment où il (le voyageur) arrive.

Application de Dijkstra au réseau ferroviaire.

algorithme DIJKSTRA(R, s, h)
in : $R = (S, V[], t)$ un réseau ferroviaire, s et h la ville et l'heure de départ,

pour chaque ville $u \in S$ **faire**
    $d[u] := \infty$,
$d[s] := 0$,
construire un tas T *ordonné selon* d *contenant tous les sommets de* S,
tant que T n'est pas vide **faire**
    $u :=$EXTRAIRE_LE_MIN(T),
    pour chaque ville $v \in V[u]$ **faire**
        si $d[v] > d[u] + t_{min}(u, v, d[u], h)$ **alors**
            $d[v] := d[u] + t_{min}(u, v, d[u], h)$,
            $pred[v] := u$,
            mettre à jour le tas T *(puisque* $d[v]$ *a diminué)*,
    fin pour,
fin tant que,
renvoyer d, $pred$,

$t_{min}(u, v, d[u], h)$ **: durée du trajet de** u **à** v**, dépends de l'heure.**

Application de Dijkstra au réseau ferroviaire.

Distance d :

- $d[u]$: durée du voyage de s à u,
- on déduit de $d[u]$ et de l'heure de départ h de s l'heure d'arrivée en u.

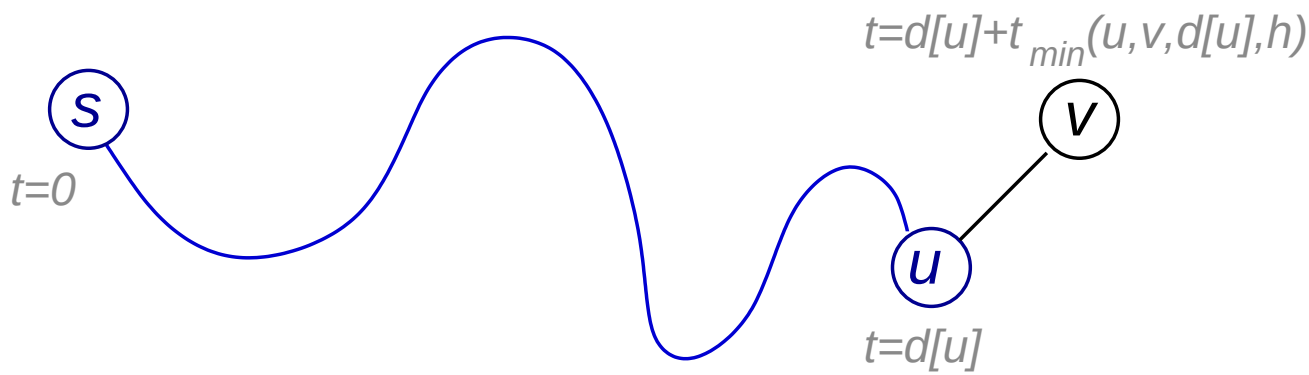
Application de Dijkstra au réseau ferroviaire.

Distance d :

- $d[u]$: durée du voyage de s à u,
- on déduit de $d[u]$ et de l'heure de départ h de s l'heure d'arrivée en u.

Temps de trajet $t_{min}(u, v, d[u], h)$: durée minimale pour aller de u à v

- dépends de l'heure d'arrivée en u,
- inclut le *temps d'attente* et la *durée effective du trajet.*



Application de Dijkstra au réseau ferroviaire.

Remarques :

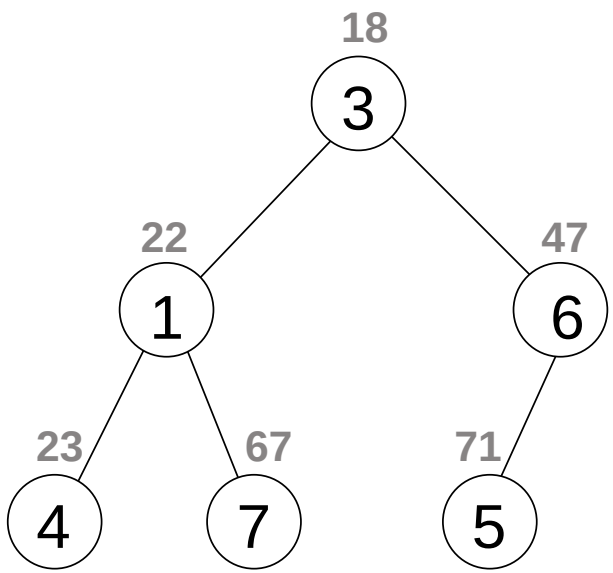
- les horaires et les durées sont en minutes,
- un train partant plus tard qu'un autre peut arriver plus tôt,
- un voyage peut durer plusieurs jours,
- sur une même ligne le temps de trajet entre deux villes successives est inférieur à 24 heures.

Dijkstra : gestion du tas.

EXTRAIRE_LE_MIN(T) : voir le cours sur les *tris* (tris par tas).

Mise à jour du tas : opération DIMINUER_LA_CLE

- *localiser* v dans le tas (nécessite un tableau des positions),
- *faire remonter* v dans le tas.



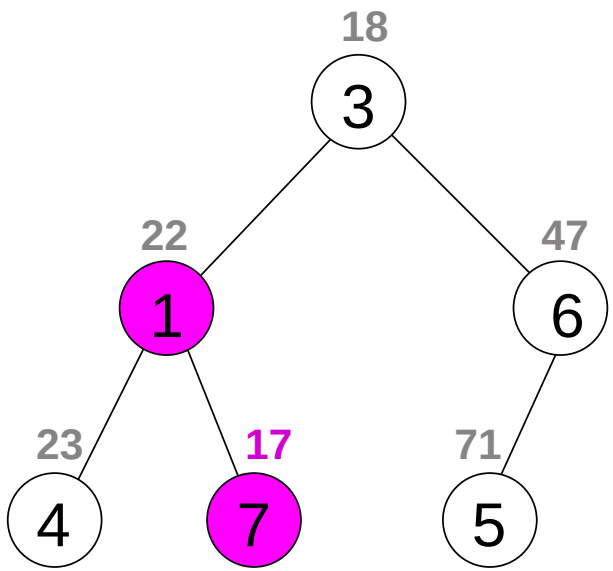
	0	1	2	3	4	5	6	7
d	12	22	0	18	23	71	47	67

	0	1	2	3	4	5	6	7
T	3	1	6	4	7	5	0	2

	0	1	2	3	4	5	6	7
pos	6	1	7	0	3	5	2	4

Dijkstra : gestion du tas.

Mise à jour du tas : opération DIMINUER_LA_CLE



	0	1	2	3	4	5	6	7
d	12	22	0	18	23	71	47	17

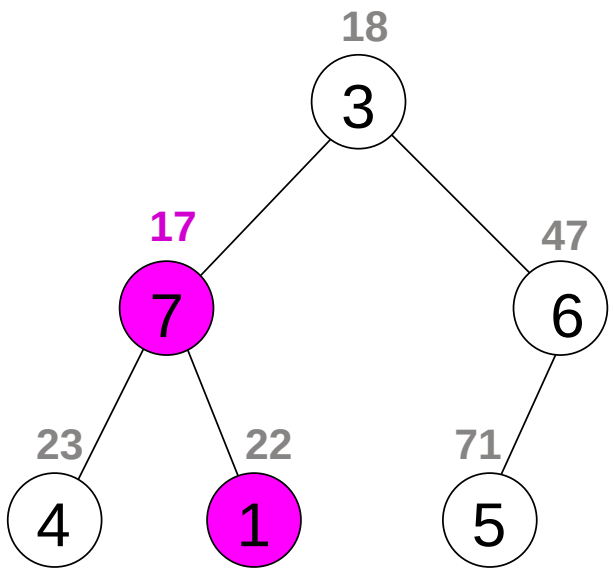
	0	1	2	3	4	5	6	7
T	3	1	6	4	7	5	0	2

	0	1	2	3	4	5	6	7
pos	6	1	7	0	3	5	2	4

Exemple : diminution de la distance $d[7]$ de 67 à 17

Dijkstra : gestion du tas.

Mise à jour du tas : opération DIMINUER_LA_CLE



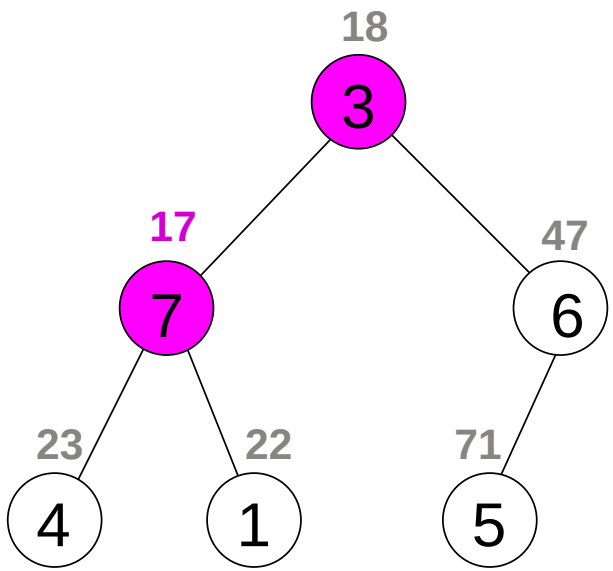
	0	1	2	3	4	5	6	7
d	12	22	0	18	23	71	47	17

	0	1	2	3	4	5	6	7
T	3	7	6	4	1	5	0	2

	0	1	2	3	4	5	6	7
pos	6	4	7	0	3	5	2	1

Dijkstra : gestion du tas.

Mise à jour du tas : opération DIMINUER_LA_CLE



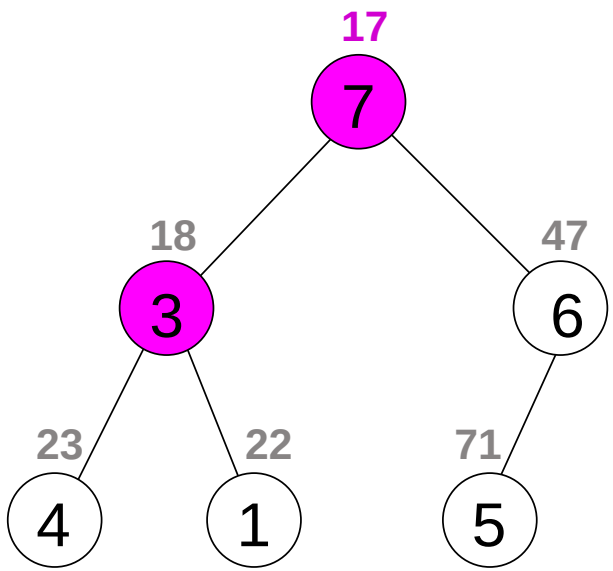
	0	1	2	3	4	5	6	7
d	12	22	0	18	23	71	47	17

	0	1	2	3	4	5	6	7
T	3	7	6	4	1	5	0	2

	0	1	2	3	4	5	6	7
pos	6	4	7	0	3	5	2	1

Dijkstra : gestion du tas.

Mise à jour du tas : opération DIMINUER_LA_CLE



	0	1	2	3	4	5	6	7
d	12	22	0	18	23	71	47	17

	0	1	2	3	4	5	6	7
T	7	3	6	4	1	5	0	2

	0	1	2	3	4	5	6	7
pos	6	4	7	1	3	5	2	0

Le sommet 7 ne peut pas remonter plus haut.
L'opération coûte $O(\log n)$.

Première étape

Ecrire un programme qui lit les données en suivant le format indiqué et qui construit les structures de données.

Problème : On se trouve dans une ville s à l'heure h, quel itinéraire suivre pour arriver au plus tôt dans la ville t ?

Algorithme : modifier la signification des marques dans l'algorithme de Dijkstra : $d[v]$ est la durée minimale du voyage pour atteindre la ville v, en partant de s à l'heure h.

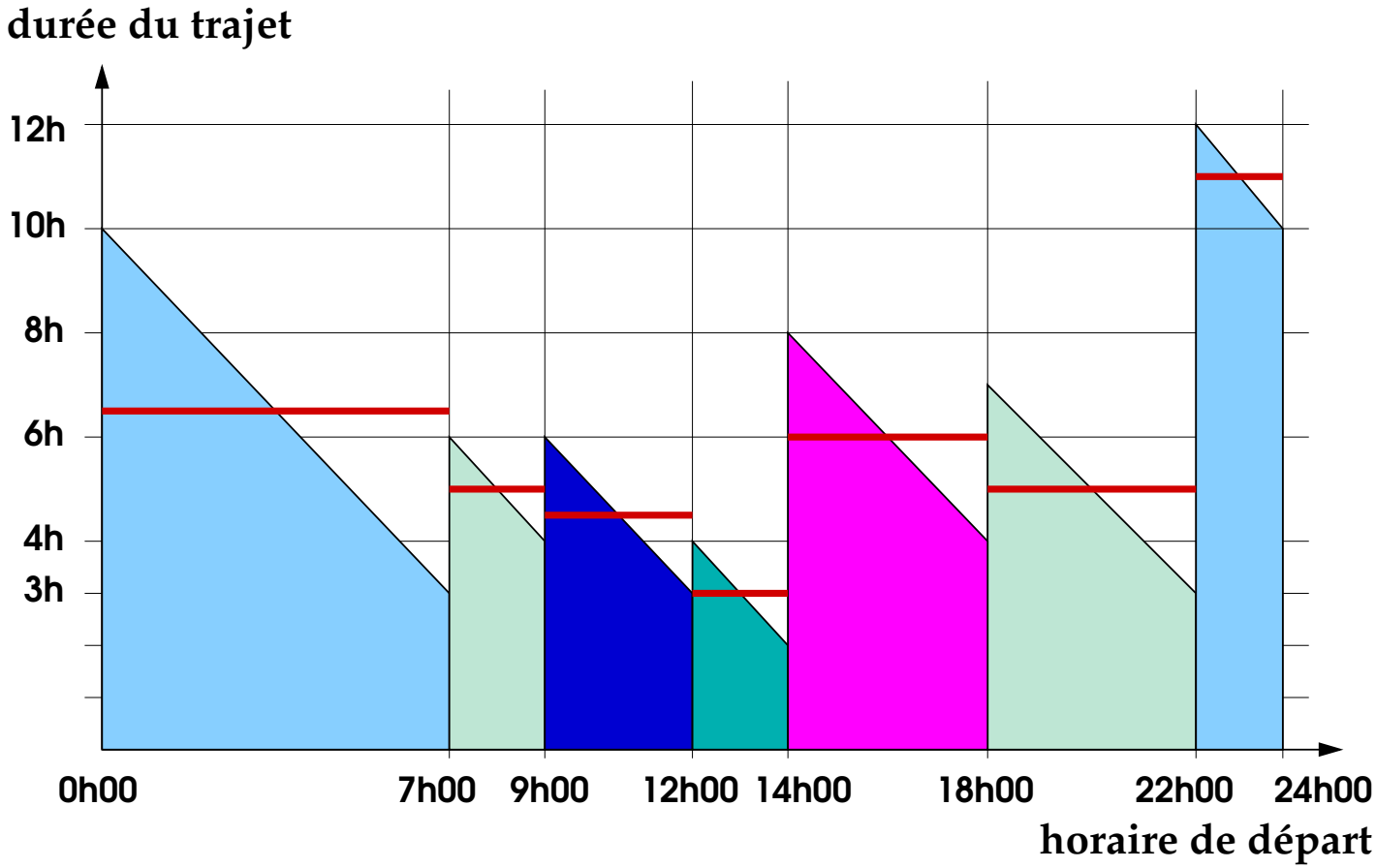
Le programme devra calculer la date d'arrivée au plus tôt à la destination t et afficher un itinéraire optimal (listes des villes traversées, lignes utilisées, horaires).

Deuxième étape

- Calcul des durées moyennes des trajets,
- Clustering (regroupement des villes en fonction de leur proximité),
- Interface graphique,
- Génération aléatoire des réseaux ferroviaires.

Calcul de la durée moyenne d'un trajet

On veut calculer la durée moyenne $m(u, v)$ du trajet pour se rendre de la ville u à la ville v. Cette durée inclut le temps d'attente dans la ville de départ ainsi que dans chacune des villes traversées.



Calcul de la durée moyenne d'un trajet

Soient $h_0 = 0$, $h_n = 24$ et soient $h_1, ..., h_{n-1}$ les horaires de départ des trains à partir de la ville de départ, dans l'ordre croissant.

Entre deux départs de trains consécutifs h_{i-1} et h_i, le temps de trajet diminue de façon linéaire, la durée moyenne sur cet intervalle est donc

$$m_i = \text{durée}(h_i) + \frac{h_i - h_{i-1}}{2}$$

La durée moyenne sur 24 heures est donnée par

$$m(u, v) = \frac{\sum_{i=1}^{n} m_i \times (h_i - h_{i-1})}{h_n - h_0}$$

Distance

On définit la distance $d(u, v)$ entre deux villes u et v de la façon suivante :

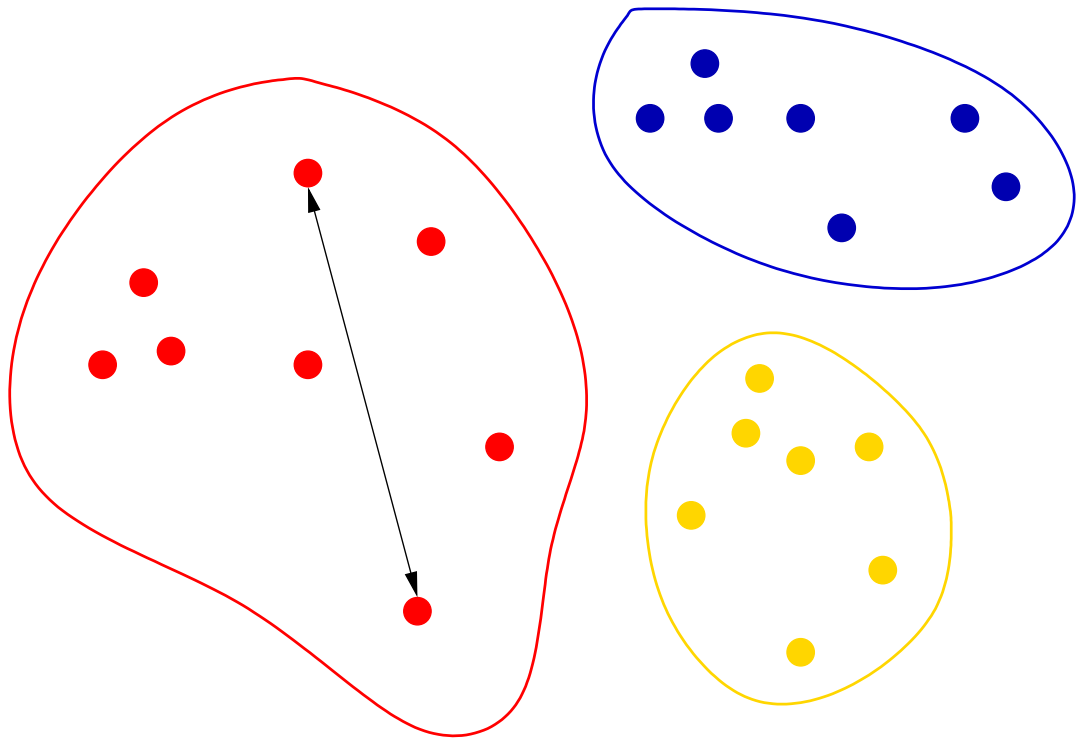
$$d(u, v) = \frac{m(u, v) + m(v, u)}{2}$$

Votre programme devra calculer la matrice des distances entre chaque paire de villes.

Remarque. Cette distance est définie de façon à être symétrique, i.e. $d(u, v) = d(v, u)$ ce qui n'est pas le cas pour $m(u, v)$.

Clustering

Pour un entier k donné, on veut partitionner les villes en k clusters $V_1, V_2, ..., V_k$ de telle sorte que la plus grande distance entre deux villes d'un même cluster soit la plus petite possible.



Clustering : solution initiale

Algorithme glouton (non optimal) :

- Soient v_1 et v_2 les deux villes les plus éloignées ($d(v_1, v_2)$ est max.).
- Placer v_1 dans V_1 et v_2 dans V_2.
- Pour $i = 3, ..., k$ faire
  - Trouver une ville v_i à *distance maximum* des villes $\{v_1, ..., v_{i-1}\}$.
  - Placer la ville v_i dans un nouveau groupe V_i.
- Pour chaque ville $v \notin \{v_1, ..., v_k\}$, placer v dans le groupe de la ville v_i la plus proche de lui.

Déf. La distance entre une ville v et un sous-ensemble S de villes est

$$d(v, S) = \min\{d(v, u) : u \in S\}.$$

Clustering : recherche locale

Méthode :

- construire une solution initiale (aléatoirement ou avec le premier algorithme).
- améliorer la solution en déplaçant une ville d'un groupe à un autre.
- recommencer

Stratégie :

- recherche du meilleur déplacement
- recherche (aléatoire) d'un déplacement améliorant

Interface graphique.

Voir la page de Edouard Thiel dédiée à ez-draw :

`http://pageperso.lif.univ-mrs.fr/~edouard.thiel/ez-draw/index.html`

Evaluation.

Répartition :

document 6 points

description du problème, de l'approche, améliorations apportées, problèmes rencontrés, solutions proposées, expérimentations,. . .

projet 10 points

structure, organisation, qualité du code, justesse, lisibilité, solutions employées,. . .

Validation, expérimentations 4 points

vérification de la justesse, efficacité, essais sur des problèmes particuliers,. . .

Génération du réseau.

fonction **genereReseau**(n, d, m, max)

génère m lignes de trains contenant au plus max gares
dans un réseau ferroviaire de n gares et de densité d.

début

générer aléatoirement les positions des n villes dans le plan,
calculer les distances euclidiennes l entre les villes,
$nb\_aretes := d \times n \times (n-1)/200$,
$A := \emptyset$,
$V = \{1, \ldots, n\}$,
pour $i := 1$ à nb_aretes **faire**
choisir au hasard $u \in V$ (non saturé),
soit $v \in V$ la plus proche de u telle que $(u, v) \notin A$*,
$A := A \cup \{(u, v)\}$,
fin faire
calculer les longueurs D des plus courts chemins dans (V, A, l),
pour $i := 1$ à m **faire**
$L[i] :=$ genererUneLigne (max, V, A, D),
fin faire
renvoyer L,

fin fonction

* *au sens de la distance euclidienne.*

Génération d'une ligne.

fonction **genererUneLigne**(max, V, A, D)

V un ensemble de n villes, A un réseau ferroviaire,

D les longueurs des plus courts chemins dans (V, A),

***Résultat :** une ligne de au plus max gares.*

begin

choisir deux villes a et b dans V au hasard,

$i := 0$,

$T[i] := a$,

while $i < max$ et $T[i] \neq b$ **do**

choisir au hasard une ville v voisine de $T[i]$ telle que $D(v, b) < D(T[i], b)$,

$i := i + 1$,

$T[i] := v$,

choisir h le nombre de passages par jour,

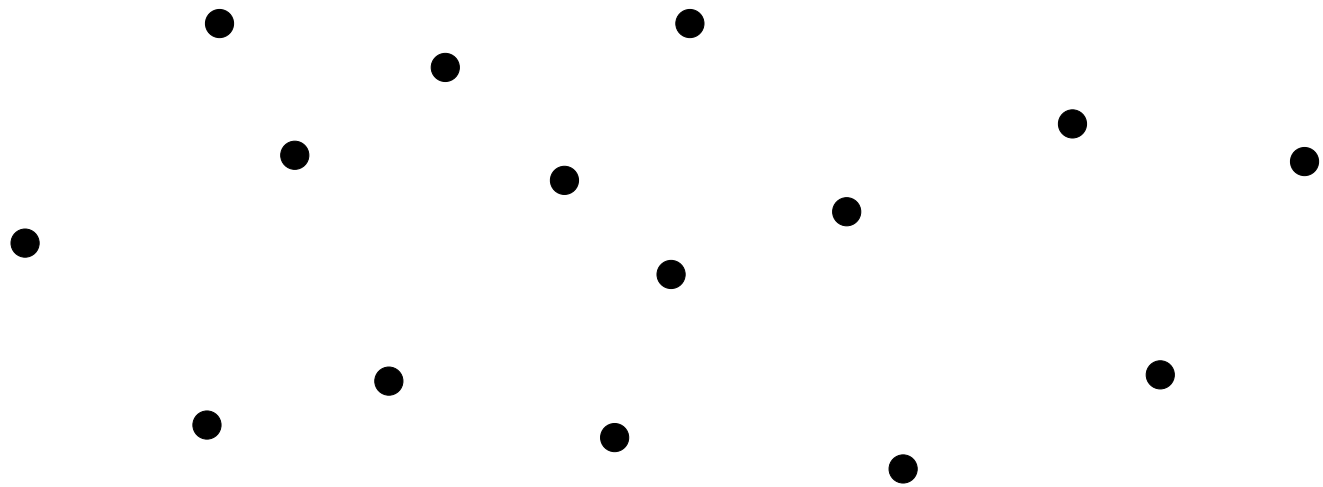
générer h horaires de passage H,

return (T, i, H, h),

end

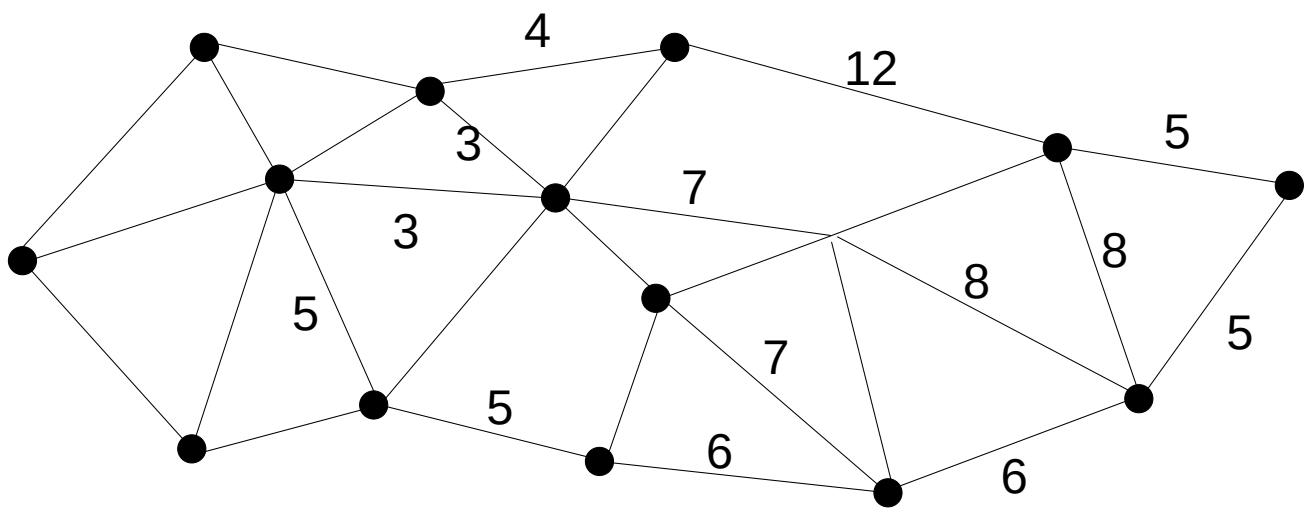
Génération aléatoire.

Positions des villes.



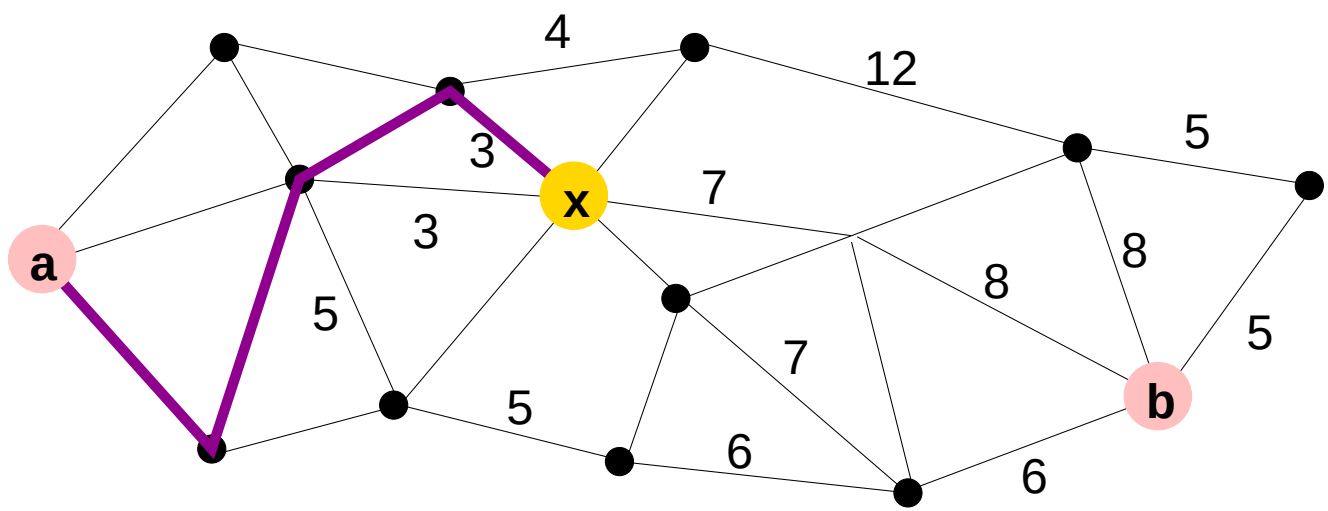
Génération aléatoire.

Voies ferrées.



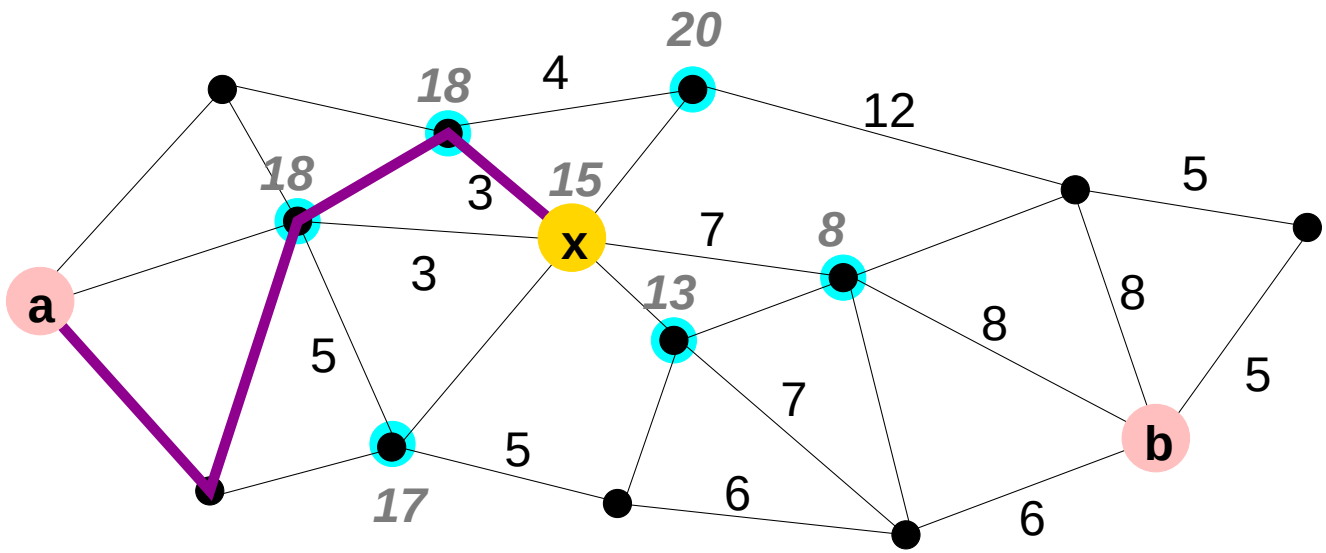
Génération aléatoire.

Construction des lignes.



Génération aléatoire.

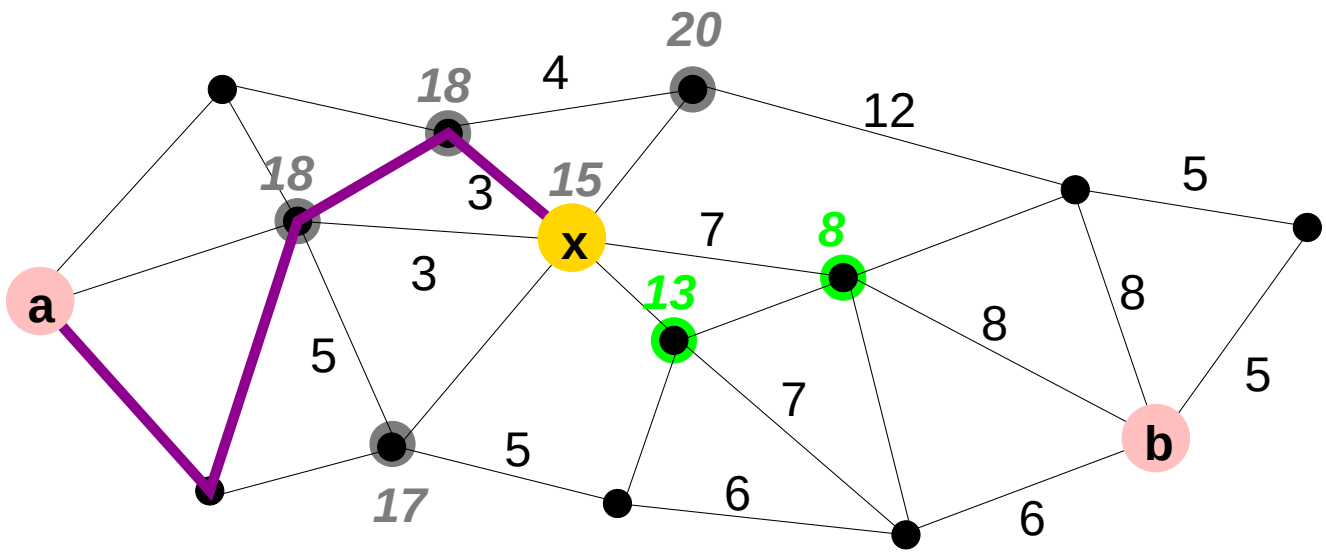
Construction des lignes.



Choix de l'étape suivante : parmi les gares reliées directement à x.

Génération aléatoire.

Construction des lignes.



Considérer uniquement les gares rapprochant *strictement* de la destination.