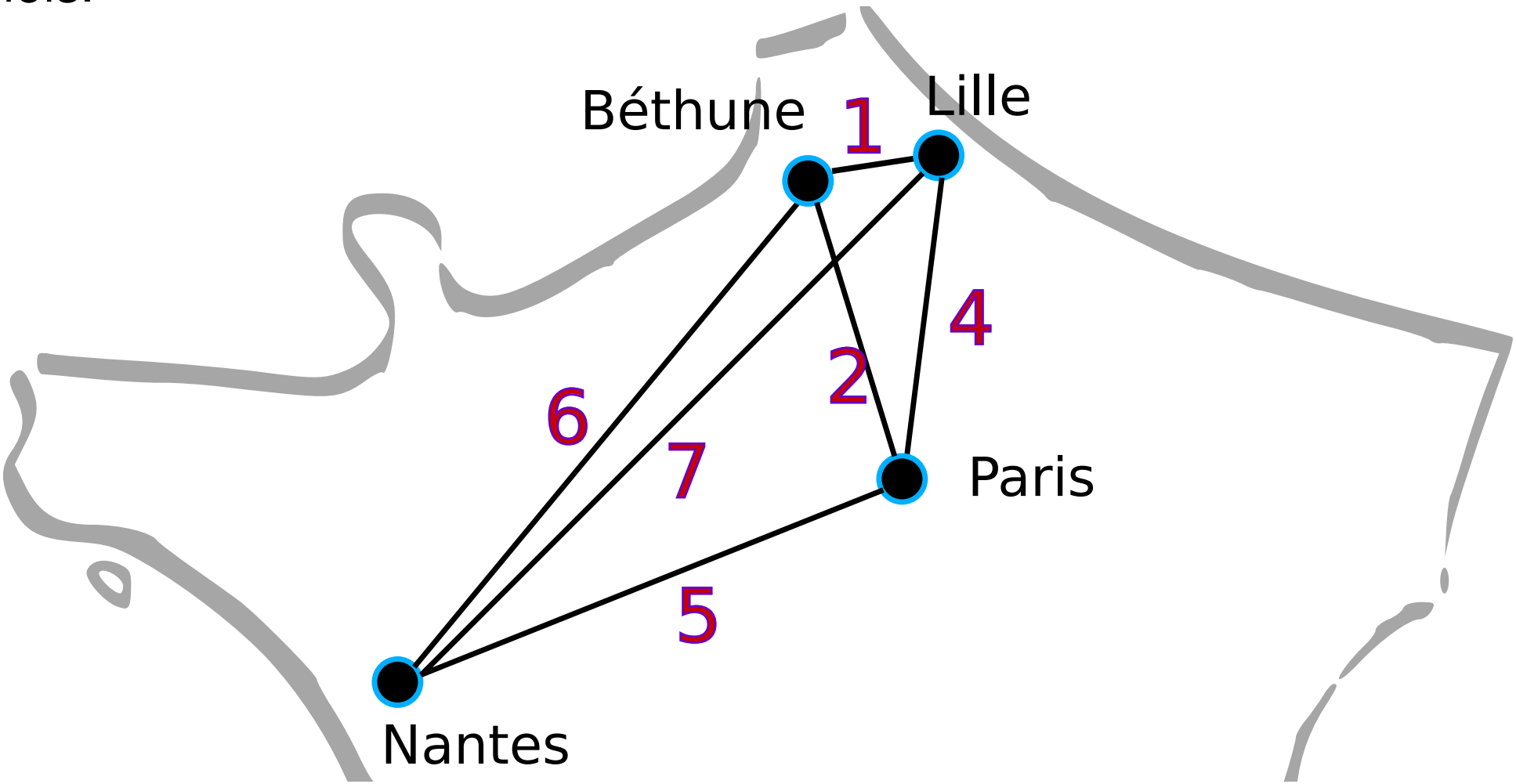
Rappel petite instance voyageur de commerce

Trouver le plus court circuit passant par chaque ville une seule fois.



L'implémentation d'une solution algorithmique

Trois étapes :

1. coder le problème en langage formel (informatique)
   - définir les variables, les tableaux, les matrices
2. décrire le pseudo-code (l'algorithme théorique "sur papier")
   - on peut simuler l'exécution étape par étape sur papier
3. programmer le pseudo-code (en Julia ou Python)

Codage en langage informatique

- Villes codés par des chiffres : 1 = Lille, 2 = Paris, 3 = Nantes, 4 = Béthune
- L'ensemble de villes $V = \{1, 2, \ldots, n\}$
- Solution candidate : un trajet codé par un vecteur
  - Exemple : $[1, 3, 2, 0, 1]$ représente : $1 \rightarrow 3 \rightarrow 2 \rightarrow 0 \rightarrow 1$
- L'ensemble d'arêtes E comporte toutes les paires de villes
- Une matrice D (d'adjacence) indique la distance entre chaque deux villes
  - Ex., D_{12} : la distance Ville 1 $\rightarrow$ Ville 2

Codage en langage informatique

- Villes codés par des chiffres : 1 = Lille, 2 = Paris, 3 = Nantes, 4 = Béthune
- L'ensemble de villes $V = \{1, 2, \ldots, n\}$
- Solution candidate : un trajet codé par un vecteur
  - Exemple : $[1, 3, 2, 0, 1]$ représente : $1 \rightarrow 3 \rightarrow 2 \rightarrow 0 \rightarrow 1$
- L'ensemble d'arêtes E comporte toutes les paires de villes
- Une matrice D (d'adjacence) indique la distance entre chaque deux villes
  - Ex., D_{12} : la distance Ville 1 $\rightarrow$ Ville 2

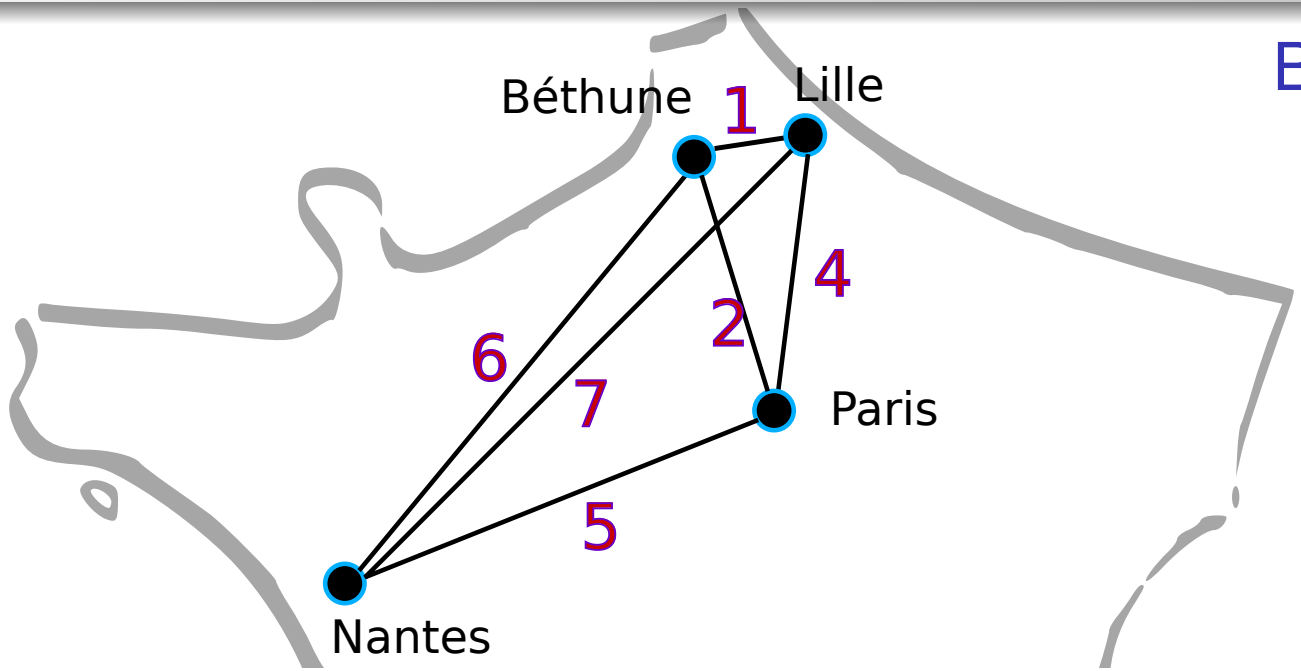
Codage en langage informatique

- Villes codés par des chiffres : 1 = Lille, 2 = Paris, 3 = Nantes, 4 = Béthune
- L'ensemble de villes $V = \{1, 2, \ldots, n\}$
- Solution candidate : un trajet codé par un vecteur
  - Exemple : $[1, 3, 2, 0, 1]$ représente : $1 \rightarrow 3 \rightarrow 2 \rightarrow 0 \rightarrow 1$
- L'ensemble d'arêtes E comporte toutes les paires de villes
- Une matrice D (d'adjacence) indique la distance entre chaque deux villes
  - Ex., D_{12} : la distance Ville 1 $\rightarrow$ Ville 2

Pseudo-code : du langage naturel au code

1. Définir Solution candidate initiale s codée $[1, 2, 3, 4, 1]$
2. s est évaluée avec la fonction d'évaluation
   - la somme des distances, i.e. $d_{12} + d_{23} + d_{34} + d_{41} = \ldots$
3. Chercher deux villes à inverser pour améliorer la somme des distances
4. Amélioration possible $\Rightarrow$ revenir à l'étape 2

Simulation exécution



Béthune `ville 1`

Lille `ville 2`

Paris `ville 3`

Nantes `ville 4`

Simulons le pseudo-code étape par étape sur ces distances

```
d = [0 1 2 6;
     1 0 4 7;
     2 4 0 5;
     6 7 5 0]
```

1. Objectif : la résolution pratique d'un exemple de problème
   - Le problème du voyageur de commerce : pseudo-code
   - Codage informatique : **tableaux** et **boucles**
   - Le problème du voyageur de commerce : code Julia

Modélisation du problème

- `n=7`
- Une solution candidate `s` est un tableau
  - La taille du tableau est ....
  - ```
    s=zeros(n+1)
    for i in 1:n
        s[i]=i
    end
    s[n+1] = 1
    ```
- Matrice de distances, avec des valeurs aléatoires de 1 à 9 :
  - `d=rand(1:9, 4,4)`
  - ou `[random.choices(range(1,10), k=4) for _ in range(4)]` en Python après `import random`
    ```

Modélisation du problème

- `n=7`
- Une solution candidate `s` est un tableau
  - La taille du tableau est $n+1$
  - `s=zeros(n+1)`

```
for i in 1:n
    s[i]=i
end
s[n+1] = 1
```

- Matrice de distances, avec des valeurs aléatoires de 1 à 9 :
  - `d=rand(1:9, 4,4)`
  - ou `[random.choices(range(1,10), k=4) for _ in range(4)]` en Python après `import random`

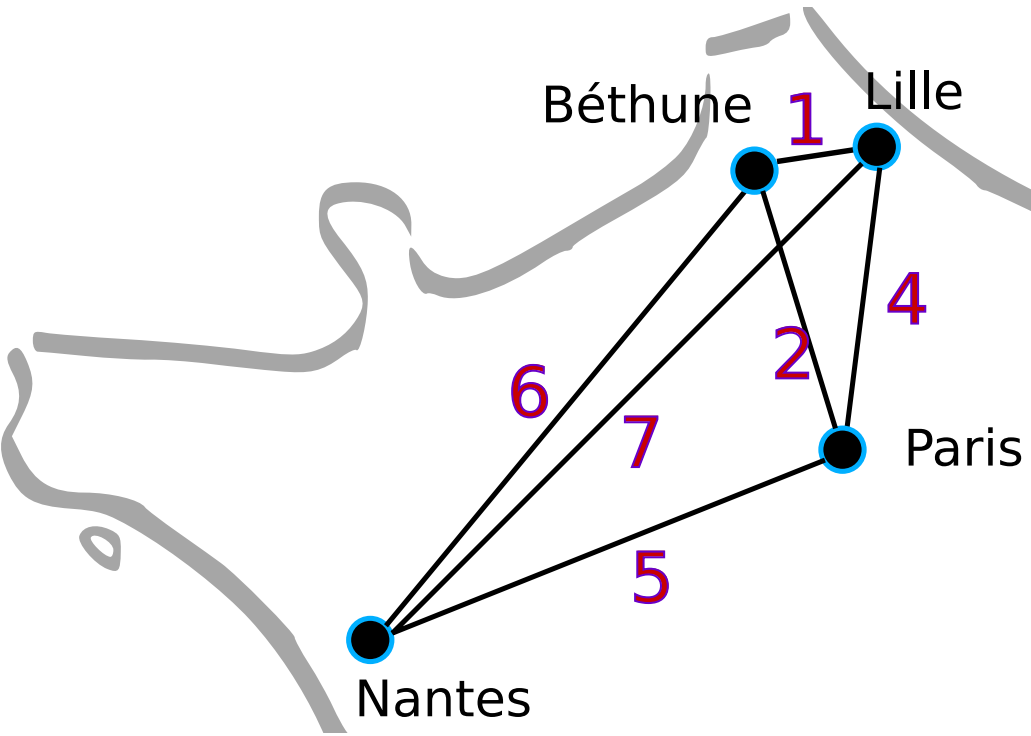
Modélisation du problème

- `n=7`
- Une solution candidate `s` est un tableau
  - La taille du tableau est
  - `s=zeros(n+1)`

```
for i in 1:n
    s[i]=i
end
s[n+1] = 1
```

- Matrice de distances, avec des valeurs aléatoires de 1 à 9 :
  - `d=rand(1:9, 4,4)`
  - ou `[random.choices(range(1,10), k=4) for _ in range(4)]` en Python après `import random`

Revenons à notre problème

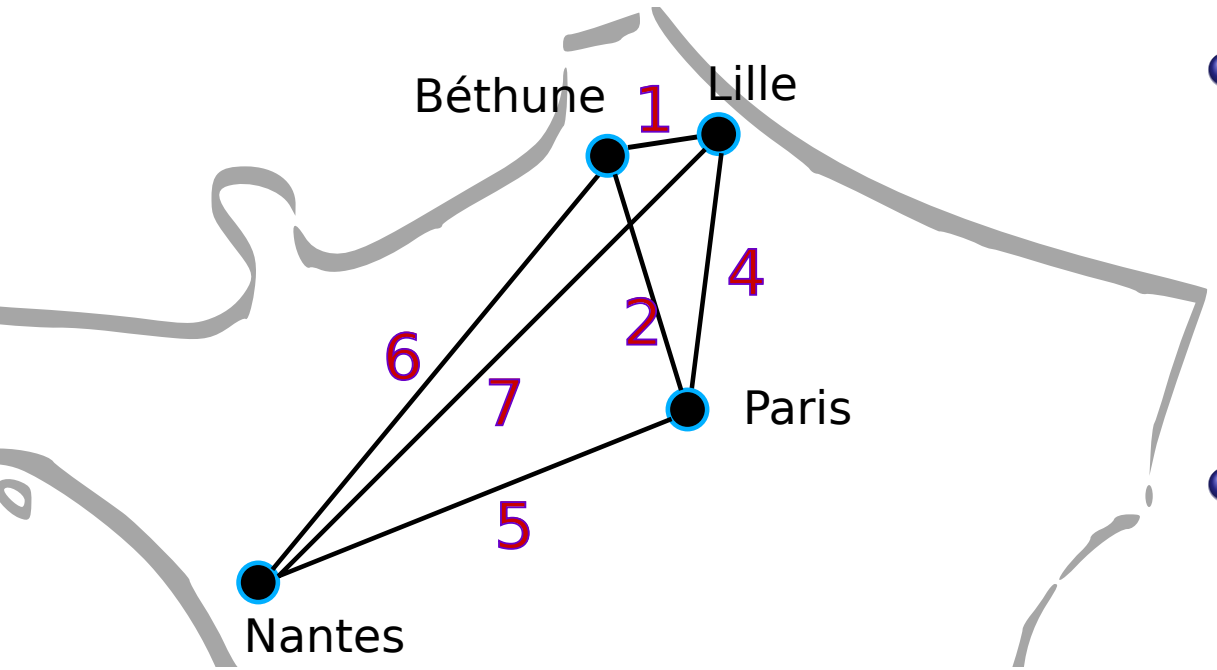


Objectif : Initialiser $s = [1, 2, 3, 4, 1]$

Béthune `ville 1`
Lille `ville 2`
Paris `ville 3`
Nantes `ville 4`

```
n = 4
d = [0 1 2 6;
     1 0 4 7;
     2 4 0 5;
     6 7 5 0]
s = zeros(Int, n+1)
for i in 1:n
    s[i] = i
end
s[n+1] = 1
```

Calculer la longueur d'un trajet



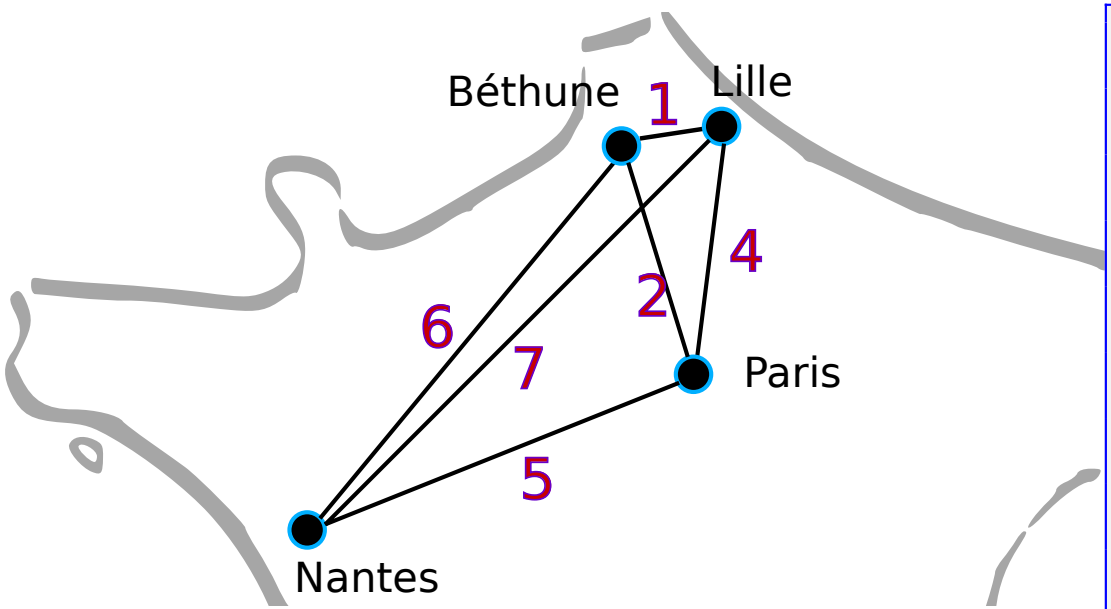

```
n = 4
d = [0 1 2 6;
     1 0 4 7;
     2 4 0 5;
     6 7 5 0]
```

- Initialiser s à un trajet valide, ex. $s = [1, 2, 3, 4, 1]$ représente le tour $1 \to 2 \to 3 \to 4 \to 1$
- Calculer la longueur du trajet (d indique les distances entre les 4 villes)

$$d[s[1], s[2]] +$$
$$d[s[2], s[3]] +$$
$$d[s[3], s[4]] +$$
$$d[s[4], s[5]]$$

1. Objectif : la résolution pratique d'un exemple de problème
   - Le problème du voyageur de commerce : pseudo-code
   - Codage informatique : **tableaux** et **boucles**
   - Le problème du voyageur de commerce : code Julia

Les données du problème




```
n = 4
d = [0 1 2 6;
     1 0 4 7;
     2 4 0 5;
     6 7 5 0]
s = zeros(Int,n+1)
for i in 1:n
    s[i] = i
end
s[n+1] = 1
```

```
function evalSol(sol) #sol comporte n+1 valeurs
    totDist=0
    for i in 1:n
        totDist = totDist + d[sol[i],sol[i+1]]
    end
    return totDist
end
```

Objectif : implémenter les étapes ci-dessous

1. $s = [1, 2, \ldots n, 1]$
2. `do`
   - $distInit = evalSol(s)$
   - $nouvDist = distInit$
   - Pour chaque paire de villes (i, j)
     A. initialiser une solution $s2$
     $s2$=une copie de s avec les villes i et j inversées
     B. si $evalSol(s2) < nouvDist$, alors
        - $s = s2$
        - $nouvDist = evalSol(s2)$

`while` ($nouvDist < distInit$) # *une itération*

Comment tourne cet algorithme sur notre exemple ?

Objectif : implémenter les étapes ci-dessous

1. $s = [1, 2, \ldots n, 1]$
2. `do`
   - $distInit = evalSol(s)$
   - $nouvDist = distInit$
   - Pour chaque paire de villes (i, j)
     A. initialiser une solution $s2$
     $s2$=une copie de s avec les villes i et j inversées
     B. si $evalSol(s2) < nouvDist$, alors
        - $s = s2$
        - $nouvDist = evalSol(s2)$

`while` ($nouvDist < distInit$) # *une itération*

Comment tourne cet algorithme sur notre exemple ?

Code Julia une itération de la boucle For

$nouvDist = distInit$

Pour chaque paire de villes (i, j)

A. initialiser une solution $s2$

   $s2$=une copie de s avec les villes i et j inversées

B. si $evalSol(s2) < nouvDist$, alors

   - $s = s2$
   - $nouvDist = evalSol(s2)$

Code Julia une itération de la boucle For

$nouvDist = distInit$

Pour chaque paire de villes (i, j)

A. initialiser une solution $s2$

   $s2$=une copie de s avec les villes i et j inversées

B. si $evalSol(s2) < nouvDist$, alors

   - $s = s2$
   - $nouvDist = evalSol(s2)$

Pour quoi besoin d'une copie ?

```
> x = [1, 2, 3]
> z = x
> z[1] = 10
> print(x)
[10, 2, 3]
```

```
> x = [1, 2, 3]
> z = copy(x)
> z[1] = 10
> print(x)
[1, 2, 3]
```

```
function tsp()
    global s
    while true
        distStart = evalSol(s)
        nDist = distStart
        for i in 2:n
            for j in i+1:n
                s2    = copy(s)
                s2[i] = s[j]
                s2[j] = s[i]
                if evalSol(s2)<nDist
                    nDist = evalSol(s2)
                    s     = s2
                    println(nDist," ",s)
                end
            end
        end
        if nDist == distStart
            break;
        end
    end
end
tsp()
```