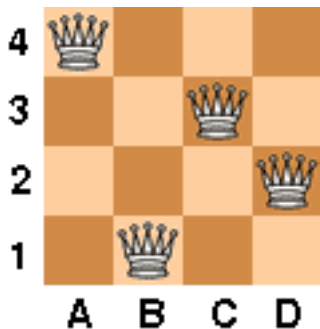


Intuition heuristiques

Soit le problème des dames et cette solution de départ :



On utilise cette fonction objectif : le nombre de conflits !

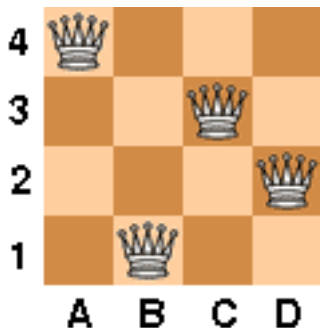
Une transformation : une dame peut aller une case en haut ou une case en bas

On a combien de possibilités ?

On choisit $C3 \rightarrow C4$, la transformation qui minimise l'objectif.

Intuition heuristiques

Soit le problème des dames et cette solution de départ :



Objectif actuel=... ?...

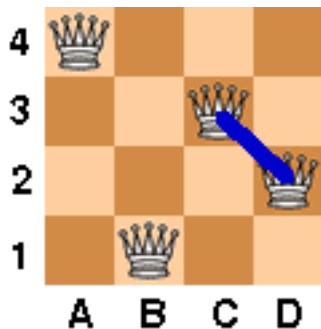
On utilise cette fonction objectif : le nombre de conflits !

Une transformation : une dame peut aller une case en haut ou une case en bas

On a combien de possibilités ?

On choisit $C3 \rightarrow C4$, la transformation qui minimise l'objectif.

Intuition heuristiques



Objectif actuel=1

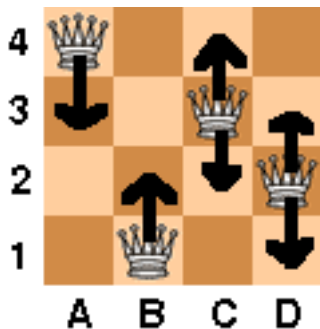
On utilise cette fonction objectif : le nombre de conflits !

Une transformation : une dame peut aller une case en haut ou une case en bas

On a combien de possibilités ?

On choisit $C3 \rightarrow C4$, la transformation qui minimise l'objectif.

Intuition heuristiques



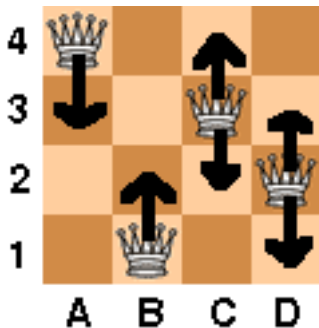
On utilise cette fonction objectif : le nombre de conflits !

Une transformation : une dame peut aller une case en haut ou une case en bas

On a combien de possibilités ?

On choisit $C3 \rightarrow C4$, la transformation qui minimise l'objectif.

Intuition heuristiques



Évaluons le nombre de conflits pour chacune des six transformations.

A4 \rightarrow A3 :

B1 \rightarrow B2 :

C3 \rightarrow C4 :

C3 \rightarrow C2 :

D2 \rightarrow D3 :

D2 \rightarrow D1 :

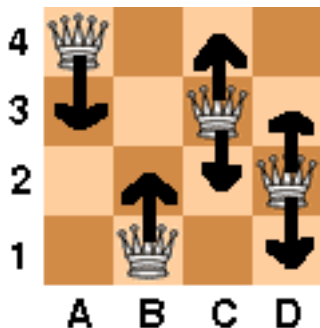
On utilise cette fonction objectif : le nombre de conflits !

Une transformation : une dame peut aller une case en haut ou une case en bas

On a combien de possibilités ?

On choisit $C3 \rightarrow C4$, la transformation qui minimise l'objectif.

Intuition heuristiques



Évaluons le nombre de conflits pour chacune des six transformations.

$A4 \rightarrow A3$: 2

$B1 \rightarrow B2$: 3

$C3 \rightarrow C4$: 1

$C3 \rightarrow C2$: 3

$D2 \rightarrow D3$: 2

$D2 \rightarrow D1$: 2

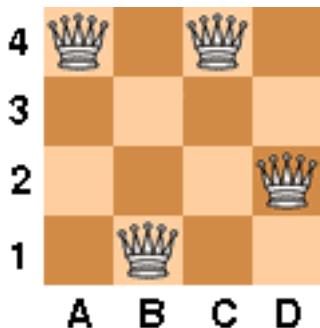
On utilise cette fonction objectif : le nombre de conflits !

Une transformation : une dame peut aller une case en haut ou une case en bas

On a combien de possibilités ?

On choisit $C3 \rightarrow C4$, la transformation qui minimise l'objectif.

Intuition heuristiques



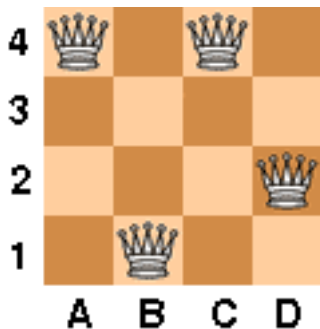
On utilise cette fonction objectif : le nombre de conflits !

Une transformation : une dame peut aller une case en haut ou une case en bas

On a combien de possibilités ?

On choisit $C3 \rightarrow C4$, la transformation qui minimise l'objectif.

Intuition heuristiques



Quelle est la meilleure transformation maintenant ?

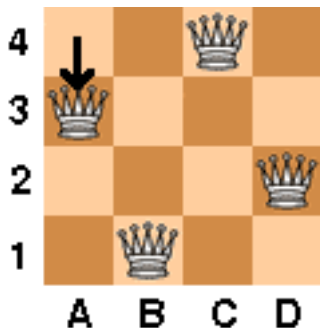
On utilise cette fonction objectif : le nombre de conflits !

Une transformation : une dame peut aller une case en haut ou une case en bas

On a combien de possibilités ?

On choisit $C3 \rightarrow C4$, la transformation qui minimise l'objectif.

Intuition heuristiques



Quelle est la meilleure transformation maintenant ? $A4 \rightarrow A3$ réduit le nombre de conflits à 0, le minimum.

On utilise cette fonction objectif : le nombre de conflits !

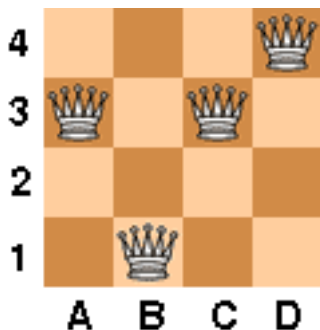
Une transformation : une dame peut aller une case en haut ou une case en bas

On a combien de possibilités ?

On choisit $C3 \rightarrow C4$, la transformation qui minimise l'objectif.

Pas toujours facile de trouver l'optimum

Soit le même problème mais avec cette solution de départ :



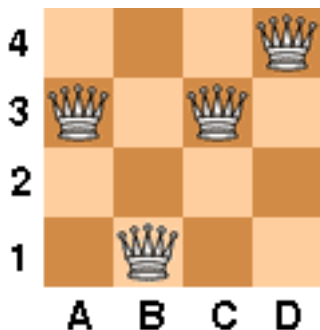
On utilise cette fonction objectif : le nombre de conflits !

Une transformation : une dame peut aller une case en haut ou une case en bas

On a combien de possibilités ?

Pas toujours facile de trouver l'optimum

Soit le même problème mais avec cette solution de départ :



Objectif actuel=... ?...

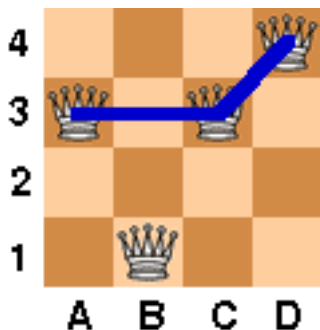
On utilise cette fonction objectif : le nombre de conflits !

Une transformation : une dame peut aller une case en haut ou une case en bas

On a combien de possibilités ?

Pas toujours facile de trouver l'optimum

Soit le même problème mais avec cette solution de départ :



Objectif actuel=2

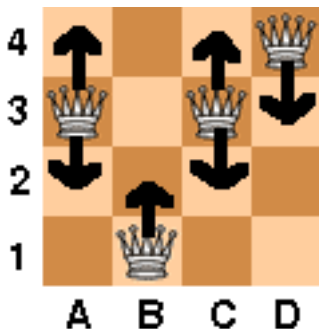
On utilise cette fonction objectif : le nombre de conflits !

Une transformation : une dame peut aller une case en haut ou une case en bas

On a combien de possibilités ?

Pas toujours facile de trouver l'optimum

Soit le même problème mais avec cette solution de départ :



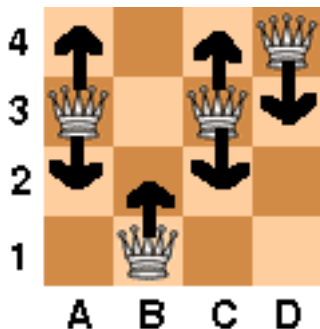
On utilise cette fonction objectif : le nombre de conflits !

Une transformation : une dame peut aller une case en haut ou une case en bas

On a combien de possibilités ?

Pas toujours facile de trouver l'optimum

Soit le même problème mais avec cette solution de départ :



Évaluons le nombre de conflits pour chacune des six transformations.

A3→A4 :

A3→A2 :

B1→B2 :

C3→C4 :

C3→C2 :

D4→D3 :

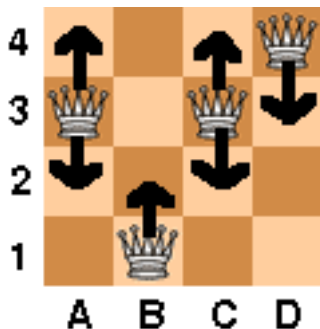
On utilise cette fonction objectif : le nombre de conflits !

Une transformation : une dame peut aller une case en haut ou une case en bas

On a combien de possibilités ?

Pas toujours facile de trouver l'optimum

Soit le même problème mais avec cette solution de départ :



Évaluons le nombre de conflits pour chacune des six transformations.

A3→A4 : 2

A3→A2 : 2

B1→B2 : 5

C3→C4 : 1

C3→C2 : 1

D4→D3 : 4

On utilise cette fonction objectif : le nombre de conflits !

Une transformation : une dame peut aller une case en haut ou une case en bas

On a combien de possibilités ?

Quelle transformation choisir parmi les six ?

La plupart des heuristiques prennent une des deux transformations qui minimisent le nombre de conflits :

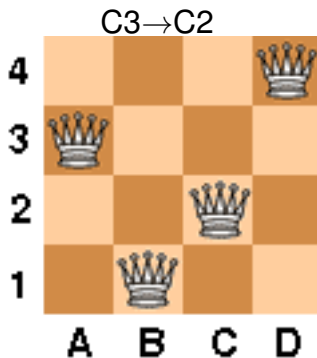
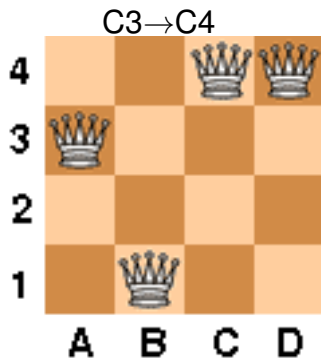
$C3 \rightarrow C4$

$C3 \rightarrow C2$

Une recherche locale fait un choix aléatoire entre les deux. On va voir quel choix mène plus vite à la solution optimale ; à la fin du cours, on va coder tout cela.

Quelle transformation choisir parmi les six ?

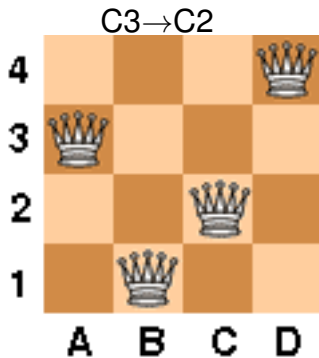
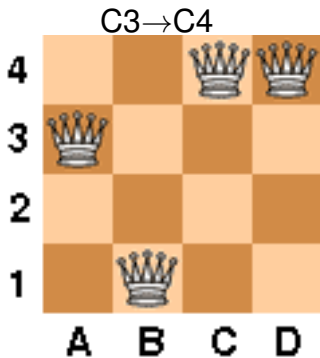
La plupart des heuristiques prennent une des deux transformations qui minimisent le nombre de conflits :



Une recherche locale fait un choix aléatoire entre les deux. On va voir quel choix mène plus vite à la solution optimale ; à la fin du cours, on va coder tout cela.

Quelle transformation choisir parmi les six ?

La plupart des heuristiques prennent une des deux transformations qui minimisent le nombre de conflits :

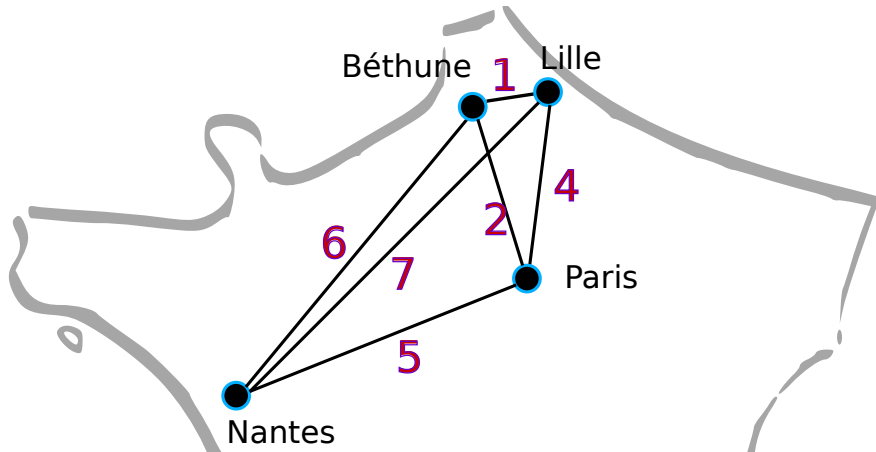


Une recherche locale fait un choix aléatoire entre les deux. On va voir quel choix mène plus vite à la solution optimale ; à la fin du cours, on va coder tout cela.

Heuristiques et méta-heuristiques

Daniel Porumbel

- 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



Trouver un circuit de longueur minimale

- il faut visiter chaque ville **une seule fois**
- Il faut partir de Paris et revenir à Paris
- chaque chiffre = une distance/temps

L'importance du nombre de villes

Après quelques calculs, on pourrait trouver la solution

- Paris → Nantes → Lille → Béthune → Paris
- longueur minimale = $5 + 7 + 1 + 2 = 15$

Deux autres solutions candidates de longueur différente :

- [Paris, Nantes, Béthune, Lille, Paris]
- [Paris, Lille, Nantes, Béthune, Paris]

$n = 4$ villes \Rightarrow 3 trajets (ou circuits ou solutions candidates)

$n = 5$ villes \Rightarrow il y a 4 places pour insérer la 4ème ville,
le nombre total de trajets est

$n = 6$ villes \Rightarrow à suivre

L'importance du nombre de villes

Après quelques calculs, on pourrait trouver la solution

- Paris → Nantes → Lille → Béthune → Paris
- longueur minimale = $5 + 7 + 1 + 2 = 15$

Deux autres solutions candidates de longueur différente :

- [Paris, Nantes, Béthune, Lille, Paris]
- [Paris, Lille, Nantes, Béthune, Paris]

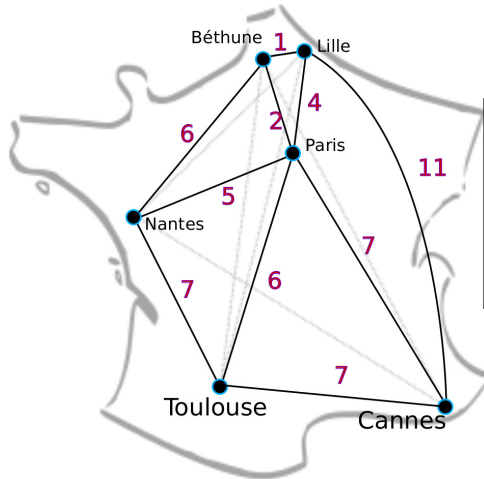
$n = 4$ villes \Rightarrow 3 trajets (ou circuits ou solutions candidates)

$n = 5$ villes \Rightarrow il y a 4 places pour insérer la 4ème ville,

le nombre total de trajets est : $3 \times 4 = 12$

$n = 6$ villes \Rightarrow à suivre

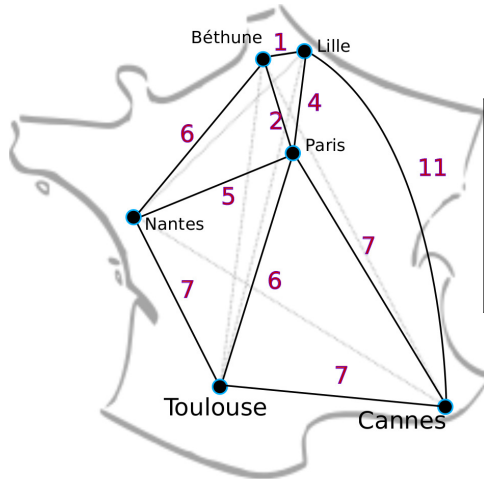
Un exemple avec $n = 6$



- Les arêtes en gris (ex., Lille-Nantes) : une valeur de 1000
- **Nombre total** d'arêtes pour n villes : ...

- Le meilleur trajet de Paris peut commencer par : Lille, Paris, Béthune, Toulouse, Cannes
- Quel est le nombre total de trajets pour n villes ?

Un exemple avec $n = 6$



- Les arêtes en gris (ex., Lille-Nantes) : une valeur de 1000

- **Nombre total** d'arêtes pour n villes : $\frac{n \cdot (n-1)}{2}$

- Le meilleur trajet de Paris peut commencer par : Lille, Paris, Béthune, Toulouse, Cannes
- Quel est le nombre total de trajets pour n villes ?

L'implémentation d'une solution algorithmique

Trois étapes :

- ➊ coder le problème **en langage formel (informatique)**
 - définir les variables, les tableaux, les matrices
- ➋ décrire le **pseudo-code** (l'algorithme théorique “sur papier”)
 - on peut simuler l'exécution étape par étape sur papier
- ➌ programmer le pseudo-code (en Julia)

Codage en langage informatique

- Villes codés par des chiffres : 1 = Lille, 2 = Paris, 3 = Nantes, 4 = Béthune
- L'ensemble de villes $V = \{1, 2, \dots, n\}$
- 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 $V = \{1, 2, \dots, n\}$
- 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

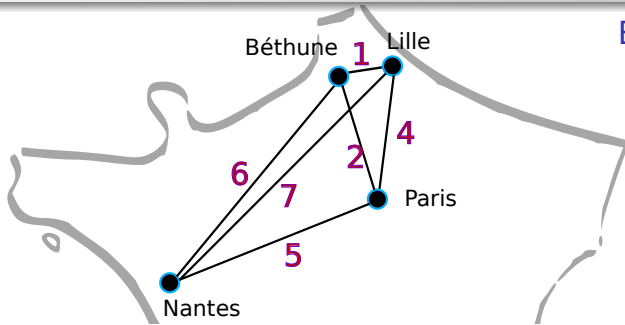
Codage en langage informatique

- Villes codés par des chiffres : 1 = Lille, 2 = Paris, 3 = Nantes, 4 = Béthune
- L'ensemble de villes $V = \{1, 2, \dots, n\}$
- 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 à Julia

- 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. $d_{12} + d_{23} + d_{34} + d_{41} = \dots$
- 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

Simulation du pseudo-code étape par étape

- 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
```

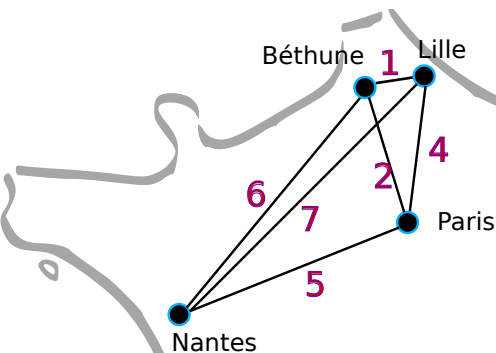
```
end
```

```
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
```

Trouver une valeur dans un tableau

```
function test()  
  s=[2,4,6,9,11,14,25,32,99]  
  mA=0  
  mB=0  
  for i in 1:9  
    if (s[i]>mA)  
      mB = mA  
      mA = s[i]  
    end  
  end  
  println(mB)  
end  
test()
```

La valeur affichée est : __?__

- Peut-on remplacer 11 avec 50 et obtenir la même valeur ?

- Écrire une méthode `trouve(val, tab)` qui renvoie l'indice de `val` dans un tableau `tab`

- `trouve(9, s)` doit renvoyer 3, car `s[3] = 9`

bonus Écrire une méthode `trouveVite(val, tab)` qui trouve avec moins de 4 comparaisons toute valeur `val` dans le tableau `s` ci-dessus !

Trouver une valeur dans un tableau

```
function test()  
  s=[2,4,6,9,11,14,25,32,99]  
  mA=0  
  mB=0  
  for i in 1:9  
    if (s[i]>mA)  
      mB = mA  
      mA = s[i]  
    end  
  end  
  println(mB)  
end  
test()
```

La valeur affichée est :32

- Peut-on remplacer 11 avec 50 et obtenir la même valeur ?

- Écrire une méthode `trouve(val, tab)` qui renvoie l'indice de `val` dans un tableau `tab`

- `trouve(9, s)` doit renvoyer 3, car `s[3] = 9`

bonus Écrire une méthode `trouveVite(val, tab)` qui trouve avec moins de 4 comparaisons toute valeur `val` dans le tableau `s` ci-dessus !

Trouver une valeur dans un tableau

```
function test()  
  s=[2,4,6,9,11,14,25,32,99]  
  mA=0  
  mB=0  
  for i in 1:9  
    if (s[i]>mA)  
      mB = mA  
      mA = s[i]  
    end  
  end  
  println(mB)  
end  
test()
```

La valeur affichée est :32

- Peut-on remplacer 11 avec 50 et obtenir la même valeur ?

- Écrire une méthode `trouve(val, tab)` qui renvoie l'indice de `val` dans un tableau `tab`

- `trouve(9, s)` doit renvoyer 3, car `s[3] = 9`

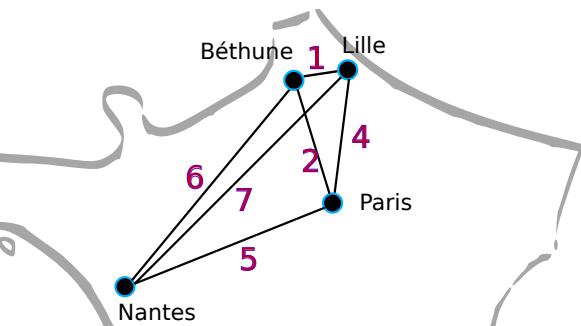
bonus Écrire une méthode `trouveVite(val, tab)` qui trouve avec moins de 4 comparaisons toute valeur `val` dans le tableau `s` ci-dessus !

Des matrices

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

- Afficher la matrice sous cette forme :
0, 1, 2, 6
1, 0, 4, 7
2, 4, 0, 5
6, 7, 5, 0
- Ecrire une fonction `septExiste(tab)` qui renvoie `true` si le tableau `tab` contient une valeur de 7
- Afficher que les lignes qui ne contiennent pas de valeur 7

Calculer la longueur d'un trajet



$n = 4$

$d = \begin{bmatrix} 0 & 1 & 2 & 6; \\ 1 & 0 & 4 & 7; \\ 2 & 4 & 0 & 5; \\ 6 & 7 & 5 & 0 \end{bmatrix}$

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

$d[s[1], s[2]] +$

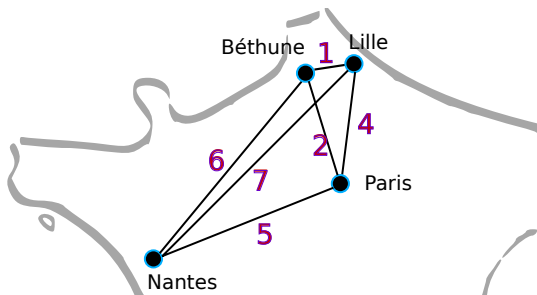
$d[s[2], s[3]] +$

$d[s[3], s[4]] +$

$d[s[4], s[5]]$

- 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 $s = [1, 2, \dots, n, 1]$

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 $s = [1, 2, \dots, n, 1]$

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 $s2$

$s2 = \text{une copie de } s \text{ avec les villes } i \text{ et } j \text{ inversées}$

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()

```