IC2: Algorithmique Avancée

Réseaux & Télécommunications

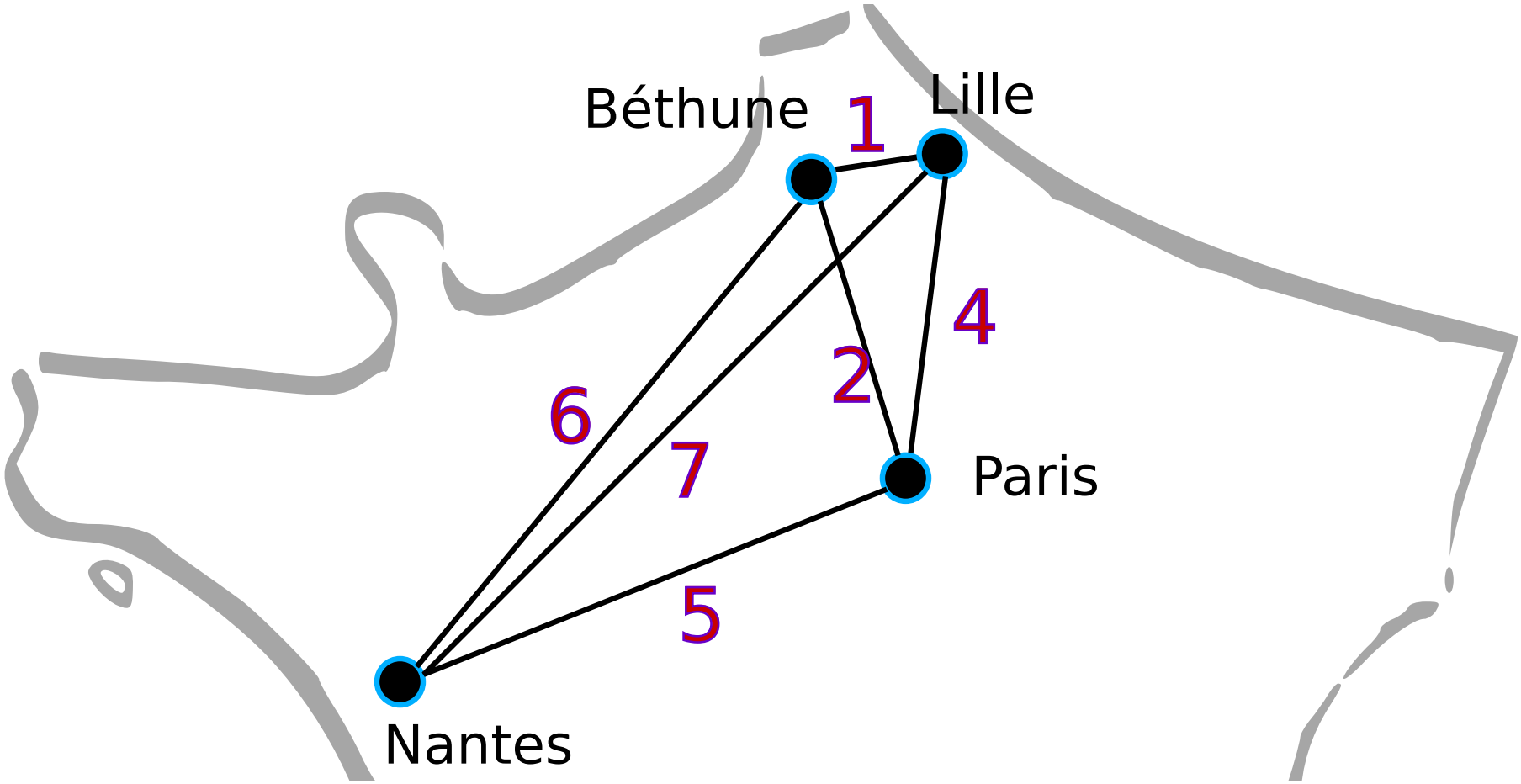
Daniel Porumbel

1. Exemple de problème algorithmique
   - Le voyageur de commerce : exemple d'approche
   - Manipulation rapides des tableaux et matrices
   - Le voyageur de commerce : programme complet

2. La Complexité des Algorithmes

3. Cryptographie : fondements mathématiques et attaques
   - Systèmes à clés privés/symétriques
   - Systèmes à clés publiques
   - Groupes : l'échange des clés Diffie-Hellman $\oplus$ idées RSA

Notation : 20% Colle, 30% DS, 50% TP



Trouver un trajet de longueur minimale

- il faut visiter chaque ville une seule foix
- Il faut partir de Béthune et revenir à Béthune
- les chiffres sont des durées approximatives de trajets

L'importance du nombre de villes

Après quelques calculs, on pourrait trouver la solution

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

Deux autres solutions candidates de longueur différente :

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

$n = 4$ villes $\Rightarrow$ 3 trajets (solutions candidates) à comparer
$n = 5$ 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

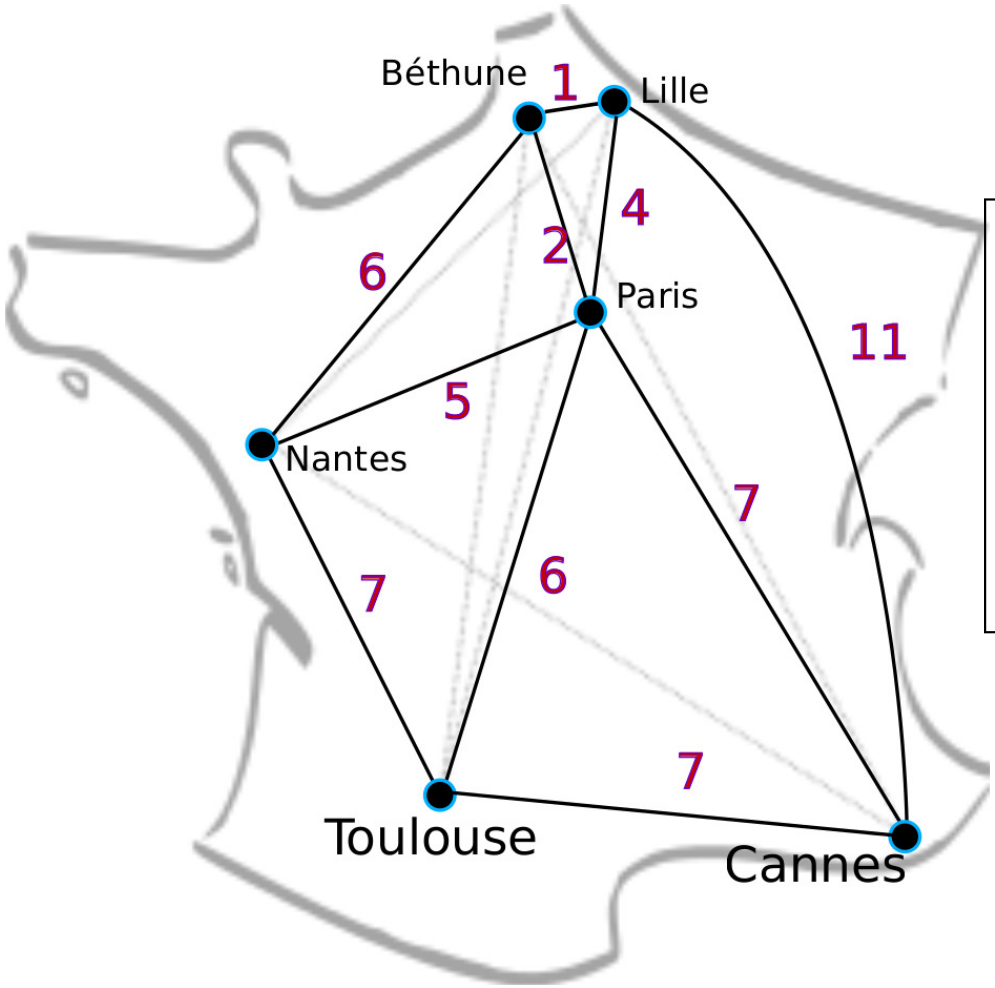
Après quelques calculs, on pourrait trouver la solution

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

Deux autres solutions candidates de longueur différente :

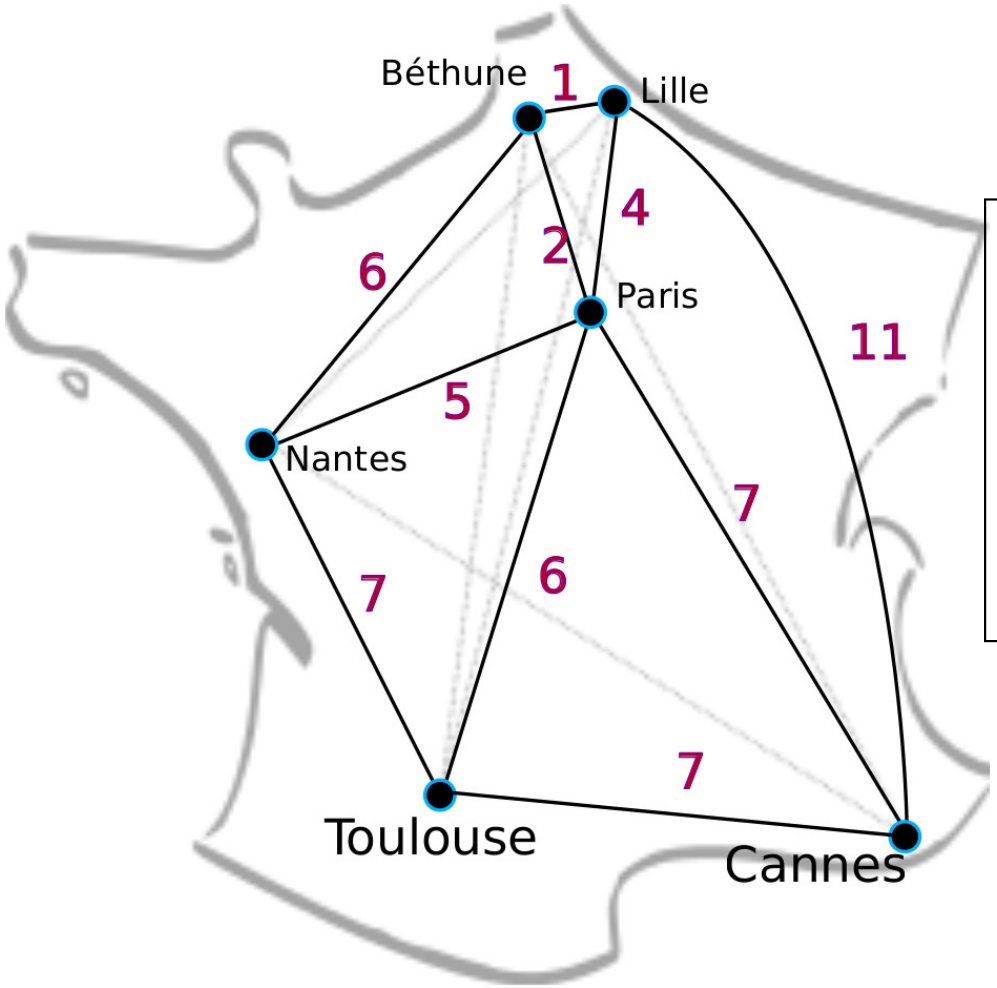
- [Béthune, Lille, Paris, Nantes, Béthune]
- [Béthune, Paris, Lilles, Nantes, Béthune]

$n = 4$ villes $\Rightarrow$ 3 trajets (solutions candidates) à comparer
$n = 5$ 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



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

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



- Les arêtes en gris (ex., Lille-Nantes) : distance de 1000
- Nombre total d'arêtes pour n villes : $\frac{n \cdot (n-1)}{2}$

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

Méthodologie Générale de Résolution

Trois étapes pour coder l'algorithme

1. énoncer le problème en langage informatique
   - définir les variables, les tableaux, les matrices
2. décrire l'algorithme en langage naturel
   - une personne qui ne connait pas le problème devrait pouvoir exécuter l'algorithme
3. programmer l'algorithme en Ruby

Codage en langage informatique

- Villes codés par des chiffres : 0 = Béthune, 1 = Lille, 2 = Paris, 3 = Nantes.
- L'ensemble de villes $V = \{0, 1, 2, \ldots, n-1\}$
- Solution candidate : un trajet codé par un vecteur
  - Exemple : $[0, 1, 3, 2, 0]$ représente : $0 \rightarrow 1 \rightarrow 3 \rightarrow 2 \rightarrow 0$
- L'ensemble d'arêtes E comporte toutes les paires de villes
  - Exemples : (0, 1), (0, 2), . . . , (0,n-1), (1,2), etc.
- Une matrice D indique la distance entre chaque deux villes
  - Ex., D_{01} : la distance entre Ville 0 (Béthune) et Ville 1 (Lille)

Codage en langage informatique

- Villes codés par des chiffres : 0 = Béthune, 1 = Lille, 2 = Paris, 3 = Nantes.
- L'ensemble de villes $V = \{0, 1, 2, \ldots, n-1\}$
- Solution candidate : un trajet codé par un vecteur
  - Exemple : $[0, 1, 3, 2, 0]$ représente : $0 \rightarrow 1 \rightarrow 3 \rightarrow 2 \rightarrow 0$
- L'ensemble d'arêtes E comporte toutes les paires de villes
  - Exemples : (0, 1), (0, 2), . . . , (0,n-1), (1,2), etc.
- Une matrice D indique la distance entre chaque deux villes
  - Ex., D_{01} : la distance entre Ville 0 (Béthune) et Ville 1 (Lille)

Codage en langage informatique

- Villes codés par des chiffres : 0 = Béthune, 1 = Lille, 2 = Paris, 3 = Nantes.
- L'ensemble de villes $V = \{0, 1, 2, \ldots, n-1\}$
- Solution candidate : un trajet codé par un vecteur
  - Exemple : $[0, 1, 3, 2, 0]$ représente : $0 \rightarrow 1 \rightarrow 3 \rightarrow 2 \rightarrow 0$
- L'ensemble d'arêtes E comporte toutes les paires de villes
  - Exemples : (0, 1), (0, 2), . . . , (0,n-1), (1,2), etc.
- Une matrice D indique la distance entre chaque deux villes
  - Ex., D_{01} : la distance entre Ville 0 (Béthune) et Ville 1 (Lille)

L'algorithme : du langage naturel à Ruby

1. Calculer la matrice D
2. Solution candidate initiale s codée $[0, 1, 2, 3, 0]$
3. s est évaluée avec la fonction d'évaluation
   - la somme des distances, i.e. $d_{01} + d_{12} + d_{23} + d_{30} = \ldots$
4. Chercher deux villes à inverser pour améliorer la somme des distances
5. Amélioration possible $\Rightarrow$ revenir à l'étape 4

Il faut savoir très bien manipuler :

- les boucles
- les tableaux
- les matrices

1. Exemple de problème algorithmique
   - Le voyageur de commerce : exemple d'approche
   - Manipulation rapides des tableaux et matrices
   - Le voyageur de commerce : programme complet

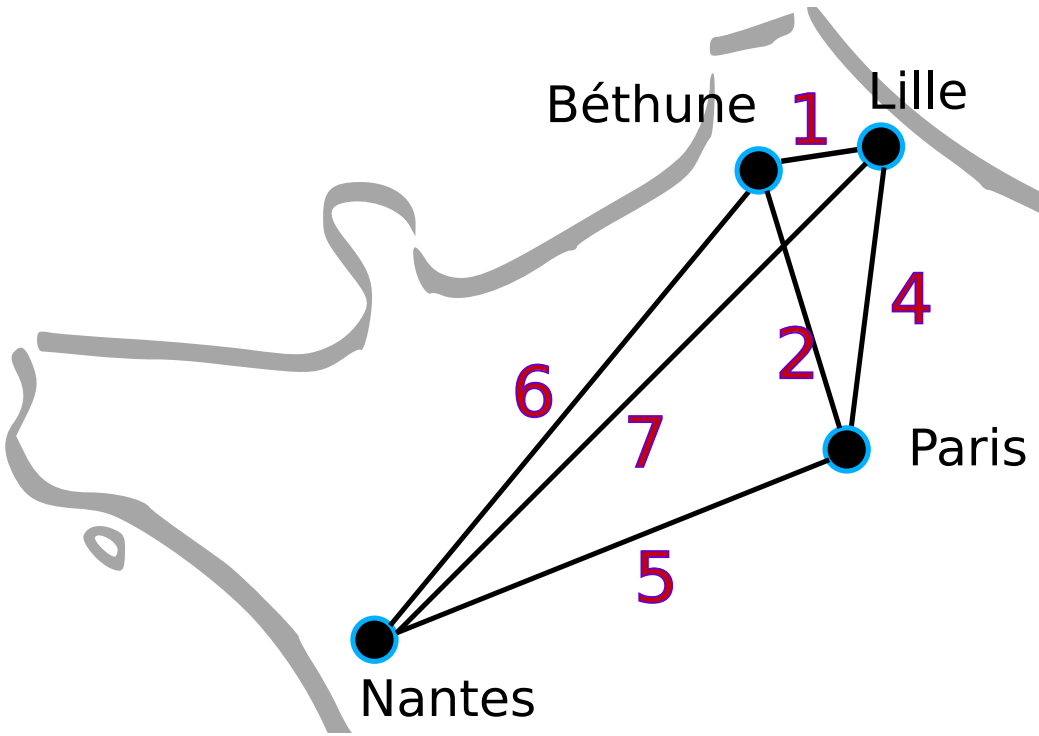
Modélisation du problème

- `n=7`
- Une solution candidate `s` est un tableau
  - `s=Array.new(...)`
  - La taille du tableau est ....
- Matrice de distances : un tableau de tableaux
  - `d=Array.new(n)`
  - `d[i]=Array.new(n)` pour chaque i de 0 à $n-1$
  - La valeur `d[i][j]` : la distance entre `villes i` et `j`
- Comment initialiser le tableau `s` ?
  - Sa valeur devrait être $[0, 1, 2 \ldots \ldots n-1, 0]$

Modélisation du problème

- `n=7`
- Une solution candidate `s` est un tableau
  - `s=Array.new(...)`
  - La taille du tableau est $n+1$
- Matrice de distances : un tableau de tableaux
  - `d=Array.new(n)`
  - `d[i]=Array.new(n)` pour chaque i de 0 à $n-1$
  - La valeur `d[i][j]` : la distance entre `villes i` et `j`
- Comment initialiser le tableau `s` ?
  - Sa valeur devrait être $[0, 1, 2 \ldots \ldots n-1, 0]$

Revenons à notre problème



Béthune `ville 0`

Lille `ville 1`

Paris `ville 2`

Nantes `ville 3`

Objectif : Ajouter 4 (ou 3) lignes pour initialiser $s = [0, 1, 2, 3, 0]$

```
1 n=4
2 s=Array.new(n+1)
3 d=Array.new(n)
4 d[0]=[0,1,2,6]
5 d[1]=[1,0,4,7]
6 d[2]=[2,4,0,5]
7 d[3]=[6,7,5,0]
```

Manipulations des tableaux

1. Comment initialiser $s = [n, n-1, n-2, \cdots -1]$
2. Comment initialiser $s = [1, 2^2, 3^2, \ldots n^2]$
3. Comment initialiser $s = [0, 1, 3, 6, 10, 15, 21 \ldots]$
   - La $n^{\text{ème}}$ valeur est $\sum_{i=1}^{n} i$?

Comment calculer x^2, x^{2^2}, x^{2^3}, x^{2^4}, ...

- initialiser $s = [7, 7^2, 7^{2^2}, 7^{2^3}, \ldots, 7^{2^n}]$
- Rappel : $x^{a+b} = x^a + x^b$

Manipulations des tableaux

1. Comment initialiser $s = [n, n-1, n-2, \cdots -1]$
2. Comment initialiser $s = [1, 2^2, 3^2, \ldots n^2]$
3. Comment initialiser $s = [0, 1, 3, 6, 10, 15, 21 \ldots]$
   - La $n^{\text{ème}}$ valeur est $\sum_{i=1}^{n} i$?

Comment calculer x^2, x^{2^2}, x^{2^3}, x^{2^4}, ...

- initialiser $s = [7, 7^2, 7^{2^2}, 7^{2^3}, \ldots, 7^{2^n}]$
- Rappel : $x^{a+b} = x^a + x^b$

Trouver une valeur dans un tableau

```
 1  s=[2,4,6,9,11,14,25,32,99]
 2  mA=0
 3  mB=0
 4  for i in 0..8 do
 5      if(s[i]>mA) then
 6          mB = mA
 7          mA = s[i]
 8      end
 9  end
10  puts mB
```

La valeur affichée est

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

- Écrire une méthode `pos(x,s)` qui renvoie la position de *x* dans le tableau *s*
  - `pos(9,s)` doit renvoyer 3, car $s[3] = 9$

bonus Écrire une méthode `posRapide(x,s)` qui trouve toute valeur *x* en faisant moins de 4 comparaisons !

Trouver une valeur dans un tableau

```
1  s=[2,4,6,9,11,14,25,32,99]
2  mA=0
3  mB=0
4  for i in 0..8 do
5      if(s[i]>mA) then
6          mB = mA
7          mA = s[i]
8      end
9  end
10 puts mB
```

La valeur affichée est 32

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

- Écrire une méthode `pos(x,s)` qui renvoie la position de *x* dans le tableau *s*
  - `pos(9,s)` doit renvoyer 3, car $s[3] = 9$

bonus Écrire une méthode `posRapide(x,s)` qui trouve toute valeur *x* en faisant moins de 4 comparaisons !

Plusieurs tableaux

Soit le tableau $s = [0, 1, 2, 3, -4, -5, 0]$

1. construire une copie de s
2. initialiser un tableau $t = [0, 1, 2, 3, 4, 5, 0]$
3. initialiser une copie de $x = [8, 9, 2, 3]$ mais qui inverse les valeurs aux positions 2 et 3
4. afficher $\sum s_i \cdot t_i$
5. afficher la valeur absolue maximale de s

Plusieurs tableaux

Soit le tableau $s = [0, 1, 2, 3, -4, -5, 0]$

1. construire une copie de s
2. initialiser un tableau $t = [0, 1, 2, 3, 4, 5, 0]$
3. initialiser une copie de $x = [8, 9, 2, 3]$ mais qui inverse les valeurs aux positions 2 et 3
4. afficher $\sum s_i \cdot t_i$
5. afficher la valeur absolue maximale de s

Plusieurs tableaux

Soit le tableau $s = [0, 1, 2, 3, -4, -5, 0]$

1. construire une copie de s
2. initialiser un tableau $t = [0, 1, 2, 3, 4, 5, 0]$
3. initialiser une copie de $x = [8, 9, 2, 3]$ mais qui inverse les valeurs aux positions 2 et 3
4. afficher $\sum s_i \cdot t_i$
5. afficher la valeur absolue maximale de s

Plusieurs tableaux

Soit le tableau $s = [0, 1, 2, 3, -4, -5, 0]$

1. construire une copie de s
2. initialiser un tableau $t = [0, 1, 2, 3, 4, 5, 0]$
3. initialiser une copie de $x = [8, 9, 2, 3]$ mais qui inverse les valeurs aux positions 2 et 3
4. afficher $\sum s_i \cdot t_i$
5. afficher la valeur absolue maximale de s

Plusieurs tableaux

Soit le tableau $s = [0, 1, 2, 3, -4, -5, 0]$

1. construire une copie de s
2. initialiser un tableau $t = [0, 1, 2, 3, 4, 5, 0]$
3. initialiser une copie de $x = [8, 9, 2, 3]$ mais qui inverse les valeurs aux positions 2 et 3
4. afficher $\sum s_i \cdot t_i$
5. afficher la valeur absolue maximale de s

Des matrices

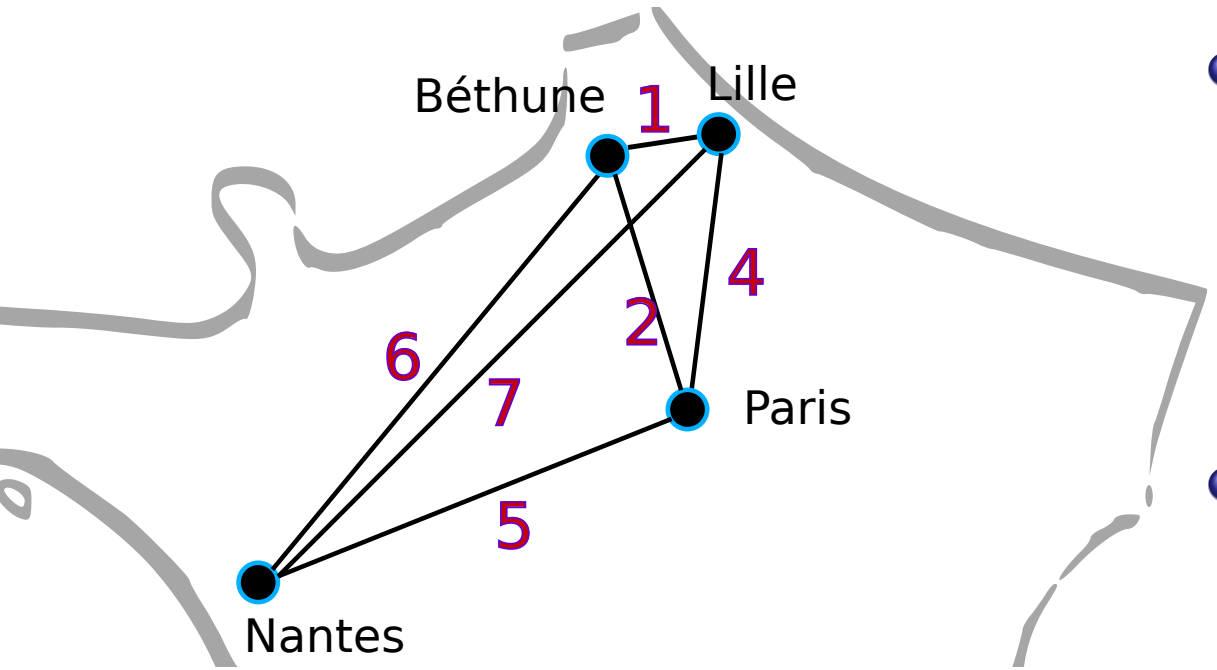
```
1 n=4
2 s=Array.new(n+1)
3 d=Array.new(n)
4 d[0]=[0,1,2,6]
5 d[1]=[1,0,4,7]
6 d[2]=[2,4,0,5]
7 d[3]=[6,7,5,0]
```

- Afficher la matrice sous cette forme :

```
0,1,2,6
1,0,4,7
2,4,0,5
6,7,5,0
```

- Afficher que les lignes qui ne contiennent pas la valeur 7
- Initialiser $s = [0, 0, 8, 42]$

Calculer la longueur d'un trajet




```
1 n=4
2 s=Array.new(n)
3 d=Array.new(n)
4 d[0]=[0,1,2,6]
5 d[1]=[1,0,4,7]
6 d[2]=[2,4,0,5]
7 d[3]=[6,7,5,0]
```

- Initialiser s à un trajet valide, ex. $s = [0, 2, 1, 3, 0]$ représente le tour $0 \to 2 \to 1 \to 3 \to 0$
- Calculer la longueur du trajet (d indique les distances entre les 4 villes)

$$d[s[0]][s[1]] +$$
$$d[s[1]][s[2]] +$$
$$d[s[2]][s[3]] +$$
$$d[s[3]][s[4]]$$

Générer les paires, trouver les duplicata

Soit le tableau $s = [2, 5, 3, 4]$

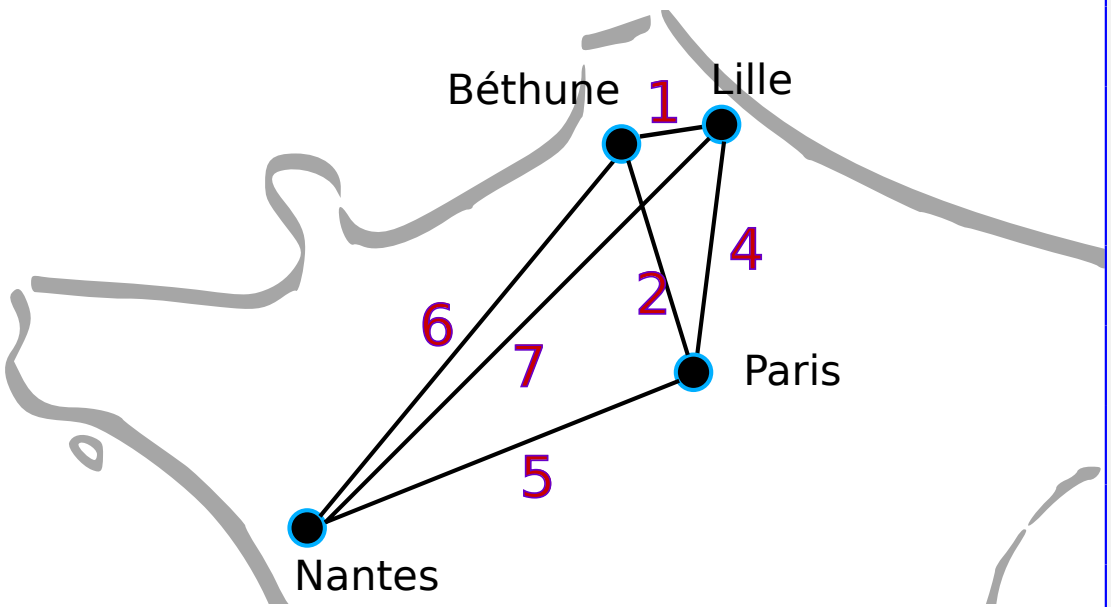
- Afficher toutes les paires :
  (2,5), (2,3), (2,4), (5,3), (5,4), (3,4)
- Afficher uniquement les paires (p_1, p_2) tel que $p_1 < p_2$

Duplicata

- Écrire une routine qui renvoie `true` uniquement s'il existe des valeurs dupliquées
  - Une idée : vérifier pour chaque paire si $p_1 \neq p_2$

1. Exemple de problème algorithmique
   - Le voyageur de commerce : exemple d'approche
   - Manipulation rapides des tableaux et matrices
   - Le voyageur de commerce : programme complet
2. La Complexité des Algorithmes
3. Cryptographie : fondements mathématiques et attaques

Les données du problème




```
1 n=4
2 d=Array.new(n,
3   Array.new(n))
4 d[0]=[0,1,2,6]
5 d[1]=[1,0,4,7]
6 d[2]=[2,4,0,5]
7 d[3]=[6,7,5,0]
8 s=[0,1,2,3,0]
```

```
1 def trajet(sol,d,n)
2     somme=0
3     for i in 0..n-1 do
4         somme = somme + d[sol[i]][sol[i+1]]
5     end
6     return somme
7 end
```

Algorithme en langage naturel

1. $s = [0, 1, 2, \ldots n, 0]$
2. 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 $trajet(s2) < trajet(s)$, alors
      - $s = s2$
      - $trajetpluspetit = true$
3. Tant que $trajetpluspetit = true$, répéter le point 2.

Comment tourne cet algorithme sur notre exemple ?

Étape 2 en Ruby

2. 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 $trajet(s2) < trajet(s)$, alors
      - $s = s2$
      - $trajetpluspetit = true$

```
1 for i in (1..n-2) do
2     for j in (i+1..n-1) do
3         s2=s.dup()          #ne pas mettre s2=s
4         s2[i]=s[j]; s2[j]=s[i]
5         if trajet(s2,d,n)<trajet(s,d,n) then
6             s=s2
7             trajetpluspetit = true
8         end
9     end
10 end
```

Étape 2 en Ruby

2. 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 $trajet(s2) < trajet(s)$, alors
      - $s = s2$
      - $trajetpluspetit = true$

```
1 for i in (1..n-2) do
2     for j in (i+1..n-1) do
3         s2=s.dup()          #ne pas mettre s2=s
4         s2[i]=s[j];  s2[j]=s[i]
5         if trajet(s2,d,n)<trajet(s,d,n) then
6             s=s2
7             trajetpluspetit = true
8         end
9     end
10 end
```

Étape 3 : la dernière

Étape 3

- Tant que l'étape 2 trouve un trajet plus petit :
  - éxécuter l'étape 2

```
trajetpluspetit = true
while (trajetpluspetit) do
    trajetpluspetit = false
    copier CODE_ETAPE2
end
```

Un exemple de complexité

Un autre algorithme pour le voyageur de commerce

- Générer tous les trajets possibles et impossibles :
  - **0**→ 1 → 1 → 1→**0** : impossible
  - **0**→ 1 → 1 → 2→**0** : impossible
  - **0**→ 1 → 1 → 3→**0** : impossible
  - **0**→ 1 → 2 → 1→**0** : impossible
  - **0**→ 1 → 2 → 2→**0** : impossible
  - **0**→ 1 → 2 → 3→**0** : possible,
    - la distance totale est calculée avec la méthode `trajet`
  - **0**→ 1 → 3 → 1→**0** : impossible,
  - **0**→ 1 → 3 → 2→**0** : possible, à comparer avec l'autre
  - . . .
  - . . .

Combien de trajets sont ainsi générés ?

Un exemple de complexité

Un autre algorithme pour le voyageur de commerce

- Générer tous les trajets possibles et impossibles :
  - **0**→ 1 → 1 → 1→**0** : impossible
  - **0**→ 1 → 1 → 2→**0** : impossible
  - **0**→ 1 → 1 → 3→**0** : impossible
  - **0**→ 1 → 2 → 1→**0** : impossible
  - **0**→ 1 → 2 → 2→**0** : impossible
  - **0**→ 1 → 2 → 3→**0** : possible,
    - la distance totale est calculée avec la méthode `trajet`
  - **0**→ 1 → 3 → 1→**0** : impossible,
  - **0**→ 1 → 3 → 2→**0** : possible, à comparer avec l'autre
  - ...
  - ...

Combien de trajets sont ainsi générés ?

Un exemple de complexité

Un autre algorithme pour le voyageur de commerce

- Générer tous les trajets possibles et impossibles :
  - **0**→ 1 → 1 → 1→**0** : impossible
  - **0**→ 1 → 1 → 2→**0** : impossible
  - **0**→ 1 → 1 → 3→**0** : impossible
  - **0**→ 1 → 2 → 1→**0** : impossible
  - **0**→ 1 → 2 → 2→**0** : impossible
  - **0**→ 1 → 2 → 3→**0** : possible,
    - la distance totale est calculée avec la méthode `trajet`
  - **0**→ 1 → 3 → 1→**0** : impossible,
  - **0**→ 1 → 3 → 2→**0** : possible, à comparer avec l'autre
  - ...
  - ...

Combien de trajets sont ainsi générés ?

La notion de complexité

- L'algorithme présenté a besoin de tester n^n trajets [1]
- $n = 5$ villes$\Rightarrow$ il faut générer 3125 trajets
- $n = 10$ villes$\Rightarrow$ il faut générer 10.000.000.000 trajets
- $n = 15$ villes$\Rightarrow$ 400.000.000.000.000.000 trajets
- $n = 100$ villes $\Rightarrow$ le nombre de trajets a 200 chiffres.

La complexité de l'algorithme est $O(n^n)$.

1. Ici, n représente le nombre de villes sans compter la ville 0 de départ

La notion de complexité

- L'algorithme présenté a besoin de tester n^n trajets [1]
- $n = 5$ villes$\Rightarrow$ il faut générer 3125 trajets
- $n = 10$ villes$\Rightarrow$ il faut générer 10.000.000.000 trajets
- $n = 15$ villes$\Rightarrow$ 400.000.000.000.000.000 trajets
- $n = 100$ villes $\Rightarrow$ le nombre de trajets a 200 chiffres.

La complexité de l'algorithme est $O(n^n)$.

1. Ici, n représente le nombre de villes sans compter la ville 0 de départ

Exemple complexité : Algorithme $\mathcal{A}$

Calculer 3^{2^n} modulo 10

1. $val = 1$
2. `for` i in $1..2^n$
   - $val = val \times 3 \,\%\, 10$

   end
3. `return` val

Complexité algorithme $\mathcal{A}$ par rapport à n : $O(2^n)$

- `Ruby` : une heure nécessaire pour $n = 30$
- Soit un langage/machine 8 fois plus rapide. Quelle valeur de n peut être traitée en une heure ?
  - .....
  - On a avancé de 10 sur l'échelle de n

Exemple complexité : Algorithme $\mathcal{A}$

Calculer 3^{2^n} modulo 10

1. $val = 1$
2. `for` i in $1..2^n$
   - $val = val \times 3 \,\%\, 10$

   end
3. `return` val

Complexité algorithme $\mathcal{A}$ par rapport à n : $O(2^n)$

- `Ruby` : une heure nécessaire pour $n = 30$
- Soit un langage/machine 8 fois plus rapide. Quelle valeur de n peut être traitée en une heure ?
  - n'=30+3=33
  - On a avancé de 10 sur l'échelle de n

Exemple complexité Algorithme $\mathcal{B}$

Calculer 3^{2^n} modulo 10

1. $val = 3$
2. `for` i in $1..n$
   - $val = val \times val \ \% \ 10$

   end
3. `return` val

Complexité algorithme $\mathcal{B}$:

- 1 seconde pour $n = 1.000.000$ sur une machine classique
- des siècles pour l'algorithme $\mathcal{A}$

Exemple complexité Algorithme $\mathcal{B}$

Calculer 3^{2^n} modulo 10

1. $val = 3$
2. `for` i in $1..n$
   - $val = val \times val \,\%\, 10$

   end
3. `return` val

Complexité algorithme $\mathcal{B}$: $O(n)$

- 1 seconde pour $n = 1.000.000$ sur une machine classique
- des siècles pour l'algorithme $\mathcal{A}$

Exemple complexité : Algorithme $\mathcal{C}$

Calculer 3^{2^n} modulo 10

1. `if n==1`
   - `return 9`

   `else`
   - `return 1`

Complexité Algorithme $\mathcal{C}$ par rapport à n : $O(1)$, une constante.

- $<$ 1 seconde (une comparaison) pour $n = 1.000.000.000$
- $\mathcal{B}$ aurait nécessité des heures, et $\mathcal{A}$ des siècles de siècles

Le calcul de la complexité

Nous avons vu trois algorithmes pour calculer 3^{2^n} *modulo 10* :

- $\mathcal{A}$ une boucle de 2^n itérations $\Rightarrow$ complexité $O(2^n)$
- $\mathcal{B}$ une boucle de n itérations $\Rightarrow$ complexité $O(n)$
- $\mathcal{C}$ pas de boucle $\Rightarrow$ complexité $O(1)$

- Déterminer la complexité pour les cas suivants :
  - 1 boucle de 2^n itérations suivie de 1 boucle de n^2 itérations $\Rightarrow$
  - 1 boucle de n^2 itérations suivie de 1 boucle de $\sqrt{n} \cdot n$ itérations $\Rightarrow$
  - 2 boucles de n itérations avec 3 multiplications par itération $\Rightarrow$
  - 3 boucles de n itérations avec 700 multiplications par itération $\Rightarrow$

Le calcul de la complexité

Nous avons vu trois algorithmes pour calculer 3^{2^n} *modulo 10* :

- $\mathcal{A}$ une boucle de 2^n itérations $\Rightarrow$ complexité $O(2^n)$
- $\mathcal{B}$ une boucle de n itérations $\Rightarrow$ complexité $O(n)$
- $\mathcal{C}$ pas de boucle $\Rightarrow$ complexité $O(1)$

- Déterminer la complexité pour les cas suivants :
  - 1 boucle de 2^n itérations suivie de 1 boucle de n^2 itérations $\Rightarrow O(2^n)$
  - 1 boucle de n^2 itérations suivie de 1 boucle de $\sqrt{n} \cdot n$ itérations $\Rightarrow O(n^2)$
  - 2 boucles de n itérations avec 3 multiplications par itération $\Rightarrow O(n)$
  - 3 boucles de n itérations avec 700 multiplications par itération $\Rightarrow O(n)$

Définition complexité

La complexité par rapport à n est $O(f(n))$ si le nombre $\mathcal{X}(n)$ d'opérations satisfait :

$$\lim_{n\to\infty} \frac{\mathcal{X}(n)}{f(n)} \leq c, \text{ où } c \text{ est une constante}$$

Exemples et équivalences

- $n^4 + n^3 + n^2 + n + 1 = O(n^4)$
- $n^2 + 3000 \cdot n = O(n^2)$
- $2^n + n^4 = O(2^n)$
  - explication : $2^n + n^4 \neq O(n^4)$, car $\lim_{n\to\infty} \frac{2^n+n^4}{n^4} = \infty$
  - si $n = 16$, alors $2^n = n^4$; si $n > 16$, $2^n > n^4$
- $log(n) + n = O(n)$

Les valeurs de n

Rappelons les algorithmes pour le calcul de 3^{2^n} (modulo 10)

- complexité $\mathcal{A}$: $O(2^n)$
- complexité $\mathcal{B}$: $O(n)$

Trouver le meilleur choix :

(1) machine classique $\oplus$ algorithme $\mathcal{B}$ rapide $O(n)$

(2) machine 10 fois plus rapide $\oplus$ algorithme $\mathcal{A}$ de $O(2^n)$

- $n \leq 5$: choix (2)
- $n \geq 5$: choix (1)

À retenir : l'intérêt de la complexité porte sur les valeurs élevées de n et non par sur les petites

Les valeurs de n

Rappelons les algorithmes pour le calcul de 3^{2^n} (modulo 10)

- complexité $\mathcal{A}$: $O(2^n)$
- complexité $\mathcal{B}$: $O(n)$

Trouver le meilleur choix :

(1) machine classique $\oplus$ algorithme $\mathcal{B}$ rapide $O(n)$

(2) machine 10 fois plus rapide $\oplus$ algorithme $\mathcal{A}$ de $O(2^n)$

- $n \leq 5$: choix (2)
- $n \geq 5$: choix (1)

À retenir : l'intérêt de la complexité porte sur les valeurs élevées de n et non par sur les petites

La complexité au pire des cas

Déterminer si un tableau contient un nombre pair ou pas

1. n=s.length()
2. for i in $0..n-1$ do
   1. if $s[i]\%2 == 0$ then
      return TRUE
      end
   end
3. return FALSE

La complexité est :

- $O(1)$ si un pair est trouvé au début
  - Souvent le cas si s est aléatoire
- $O(n)$ si le pair est trouvée à la fin

- Notre intérêt porte toujours sur la complexité au pire des cas

La complexité au pire des cas

Déterminer si un tableau contient un nombre pair ou pas

1. n=s.length()
2. for i in $0..n-1$ do
   1. if $s[i]\%2 == 0$ then
      return TRUE
      end
   end
3. return FALSE

La complexité est :

- $O(1)$ si un pair est trouvé au début
  - Souvent le cas si s est aléatoire
- $O(n)$ si le pair est trouvée à la fin

- Notre intérêt porte toujours sur la complexité au pire des cas

La cryptographie et la complexité

La difficulté de casser des codes réside dans la complexité de certains problèmes

- Factoriser un entier n en temps polynomial
  - il faut trouver deux facteurs $p \neq 1$ et $q \neq 1$ tel que $n = p \cdot q$
  - La puissance de l'algorithme RSA (utilisé par `ssh`) est basée sur le problème ci-dessous
- Existe-il un algorithme polynomial pour **le problème du logarithme discret** ?
  - La puissance de l'échange de clé Diffie-Hellman (utilisé par `ssh`) réside dans cette question

1. Exemple de problème algorithmique
2. La Complexité des Algorithmes
3. Cryptographie : fondements mathématiques et attaques
   - Systèmes à clés privés/symétriques
   - Systèmes à clés publiques
   - Groupes : l'échange des clés Diffie-Hellman $\oplus$ idées RSA

Principes de base : clés privés symétriques

- La même clé est utilisée pour coder et décoder

  $E(msg, S)$ le message *msg* codé par la clé S

  $E^{-1}(code, S)$ le code *code* décodé par la clé S

Formule de base du codage symétrique :

$$E^{-1}\left(E(msg, S), S\right) = msg$$

- Les fonctions E et E^{-1} peuvent indiquer plusieurs types d'opérations mathématiques :
  - Fonctions XOR
  - Arithmétique modulaire modulo p
  - Des algorithmes plus complexes : Mélange de permutations, fonctions XOR, et échanges de blocs de bits
    - Data Encryption Standard (DES)

Exemple de chiffrement symétriques : XOR

Chiffrement sur 4 bits : entiers entre 0 et 15

- Message à coder *msg* : 14, 12, 11, 9, 5
- Clé S : 12 (ou bien 1100 en base 2)
- Message codé $E(msg, S)$: 2 ($14 \oplus 12$), 0 ($14 \oplus 12$), 7, 5, 9

Exemple de chiffrement symétriques : XOR

Chiffrement sur 4 bits : entiers entre 0 et 15

- Message à coder *msg* : 14, 12, 11, 9, 5
- Clé S : 12 (ou bien 1100 en base 2)
- Message codé $E(msg, S)$: 2 ($14 \oplus 12$), 0 ($14 \oplus 12$), 7, 5, 9

```
   14     12     11     9      5
|1110|1100|1011|1001|0101|
              ⊕
|1100|1100|1100|1100|1100|
= = = = = = = = = = = = =
|0010|0000|0111|0101|1001|

   2      0      7      5      9
```

Exemple de chiffrement symétrique : XOR

Pour décoder un message codé via XOR, on utilise la formule

$$msg \oplus S \oplus S = msg$$

Cette formule est vérifiée par notre exemple : (*msg*=14, 12, 11, 9, 5 ; *S*=12, $E = E^{-1}$=XOR)

Exemple de chiffrement symétrique : XOR

Pour décoder un message codé via XOR, on utilise la formule

$$msg \oplus S \oplus S = msg$$

Cette formule est vérifiée par notre exemple : (*msg*=14, 12, 11, 9, 5 ; *S*=12, $E = E^{-1}$=XOR)

```
   14    12    11    9     5
|1110|1100|1011|1001|0101|    --- msg
                ⊕
|1100|1100|1100|1100|1100|
= = = = = = = = = = = = = =
|0010|0000|0111|0101|1001|    --- code
                ⊕
|1100|1100|1100|1100|1100|
= = = = = = = = = = = = = =
|1110|1100|1011|1001|0101|    --- msg
   14    12    11    9     5
```

Codage en Ruby 1

```
1 msg = [14, 12, 11, 9, 5]
2 s = 12
3 taille = msg.length
4 code = Array.new(taille)
5 final = Array.new(taille)
6 for i in (0..taille-1) do
7     code[i] = msg[i]^s
8 end
9 for i in (0..taille-1) do
10    final[i] = code[i]^s
11 end
12 puts code
13 puts final
```

Quel est le résultat de ce code ?

Chiffrement via l'arithmétique modulaire

L'idée de base de l'arithmétique modulaire : travailler non sur les nombres eux-mêmes, mais sur les restes de leur division par un nombre p.

- $14 = 2 \pmod{12}$; d'ailleurs, s'il est 14h, on peut dire 2h
- $7 + 4 = 1 \pmod{10}$
- $7 \times 4 = 2 \pmod{10}$
- $3 + 9 = 2 \pmod{10}$
- $9 = -1 \pmod{10}$
- $9 \times 9 = 1 \pmod{10}$
- $-1 \times -1 = 1 \pmod{10}$
- $-1 \times -1 = 1 \pmod{16}$
- Pour quelles valeurs de p l'égalité $-1 \times -1 = 1 \pmod{p}$ est valide ?

Chiffrement via l'arithmétique modulaire

L'idée de base de l'arithmétique modulaire : travailler non sur les nombres eux-mêmes, mais sur les restes de leur division par un nombre p.

- $14 = 2 \pmod{12}$; d'ailleurs, s'il est 14h, on peut dire 2h
- $7 + 4 = 1 \pmod{10}$
- $7 \times 4 = 2 \pmod{10}$
- $3 + 9 = 2 \pmod{10}$
- $9 = -1 \pmod{10}$
- $9 \times 9 = 1 \pmod{10}$
- $-1 \times -1 = 1 \pmod{10}$
- $-1 \times -1 = 1 \pmod{16}$
- Pour quelles valeurs de p l'égalité $\boxed{-1 \times -1 = 1 \pmod{p}}$ est valide ?

Clés privés via l'arithmétique modulaire

Soit le premier exemple présenté dans le codage XOR :

- msg=14, 12, 11, 9, 5 ;
- S=12
- La clé comporte 4 bits : des numéros entre 0 et 15, donc $p = 16$

Codage et décodage : description théorique

- Opération de codage : $E(msg, S) = msg + S \pmod{16}$
- Opération de décodage : $E^{-1}(code, S) = code - S \pmod{16}$

La formule de base du codage symétrique est vérifiée

$$E^{-1}\left(E(msg, S), S\right) = msg,$$

$$\text{car } msg + S - S = msg \pmod{\text{p}}\ \forall p \in \mathbb{N}$$

Clés privés via l'arithmétique modulaire

Soit le premier exemple présenté dans le codage XOR :

- msg=14, 12, 11, 9, 5 ;
- S=12
- La clé comporte 4 bits : des numéros entre 0 et 15, donc $p = 16$

Codage et décodage : exemple

- Codage : $E(14, 12) = 14 + 12 = 26 = 10 \pmod{16}$
- Décodage : $E^{-1}(10, 12) = 10 - 12 = -2 = 14 \pmod{16}$

msg :	14	12	11	9	5
	$+12 (mod 16)$				
$code$:	10	8	7	5	1
	$-12 (mod 16)$				
msg :	14	12	11	9	5

La longueur de la clé

- Pour les chaînes de caractères : utiliser une clé de 8 bits dans l'arithmétique modulo 256
- Pour coder "e" (code ASCII 101) avec la clé $S = 200$, on calcule $101 + 200 = 301 = 256 + 45 = 45 \pmod{256}$
- L'analyse de fréquences permet de trouver le code du caractère le plus fréquent (normalement "e"), et ensuite la clé

- Une clé plus longue (32 bits) rend l'analyse de fréquence plus difficile :
  - si on travaille sur des blocs de 4 lettres, un pirate serait obligé de chercher le bloc de 4 lettres le plus fréquent.
  - Codage XOR : une clé de 32 bits rend l'analyse de fréquence beaucoup plus difficile ? Réponse : Non, car une clé XOR peut être découpée et interprétée comme une série de plusieurs clés

La longueur de la clé

- Pour les chaînes de caractères : utiliser une clé de 8 bits dans l'arithmétique modulo 256
- Pour coder "e" (code ASCII 101) avec la clé $S = 200$, on calcule $101 + 200 = 301 = 256 + 45 = 45 \pmod{256}$
- L'analyse de fréquences permet de trouver le code du caractère le plus fréquent (normalement "e"), et ensuite la clé

- Une clé plus longue (32 bits) rend l'analyse de fréquence plus difficile :
  - si on travaille sur des blocs de 4 lettres, un pirate serait obligé de chercher le bloc de 4 lettres le plus fréquent.
  - Codage XOR : une clé de 32 bits rend l'analyse de fréquence beaucoup plus difficile ? Réponse : Non, car une clé XOR peut être découpée et interprétée comme une série de plusieurs clés

La longueur de la clé

- Pour les chaînes de caractères : utiliser une clé de 8 bits dans l'arithmétique modulo 256
- Pour coder "e" (code ASCII 101) avec la clé $S = 200$, on calcule $101 + 200 = 301 = 256 + 45 = 45 \pmod{256}$
- L'analyse de fréquences permet de trouver le code du caractère le plus fréquent (normalement "e"), et ensuite la clé

- Une clé plus longue (32 bits) rend l'analyse de fréquence plus difficile :
  - si on travaille sur des blocs de 4 lettres, un pirate serait obligé de chercher le bloc de 4 lettres le plus fréquent.
  - Codage XOR : une clé de 32 bits rend l'analyse de fréquence beaucoup plus difficile ? Réponse : Non, car une clé XOR peut être découpée et interprétée comme une série de plusieurs clés

La longueur de la clé

- Pour les chaînes de caractères : utiliser une clé de 8 bits dans l'arithmétique modulo 256
- Pour coder "e" (code ASCII 101) avec la clé $S = 200$, on calcule $101 + 200 = 301 = 256 + 45 = 45 \pmod{256}$
- L'analyse de fréquences permet de trouver le code du caractère le plus fréquent (normalement "e"), et ensuite la clé

- Une clé plus longue (32 bits) rend l'analyse de fréquence plus difficile :
  - si on travaille sur des blocs de 4 lettres, un pirate serait obligé de chercher le bloc de 4 lettres le plus fréquent.
  - Codage XOR : une clé de 32 bits rend l'analyse de fréquence beaucoup plus difficile ? Réponse : Non, car une clé XOR peut être découpée et interprétée comme une série de plusieurs clés

3. Cryptographie : fondements mathématiques et attaques
   - Systèmes à clés privés/symétriques
   - Systèmes à clés publiques
   - Groupes : l'échange des clés Diffie-Hellman $\oplus$ idées RSA

L'intérêt de la cryptographie asymétrique

Inconvénients de la cryptographie symétrique

- Avant de commencer la communication principale, les deux personnes doivent trouver un autre canal sécurisé pour transmettre la clé privé (clé USB, CD, papier)
- Si un serveur a des millions des clients (*google.fr*, *voyages-sncf.fr*), il doit trouver et stocker des millions des clés.
  - *voyages-sncf.fr* devrait échanger une clé avec chaque client

Cryptographie asymétrique : clé publique et privée

- Une clé publique unique par serveur pour le chiffrement
  - Tout client peut coder avec la clé publique *voyages-sncf.fr*
- Une clé privée pour le déchiffrement
  - Uniquement *voyages-sncf.fr* peut décoder les messages chiffrés avec sa clé publique
- Impossible de retrouver une clé à partir de l'autre

L'intérêt de la cryptographie asymétrique

Inconvénients de la cryptographie symétrique

- Avant de commencer la communication principale, les deux personnes doivent trouver un autre canal sécurisé pour transmettre la clé privé (clé USB, CD, papier)
- Si un serveur a des millions des clients (*google.fr*, *voyages-sncf.fr*), il doit trouver et stocker des millions des clés.
  - *voyages-sncf.fr* devrait échanger une clé avec chaque client

Cryptographie asymétrique : clé publique et privée

- Une clé publique unique par serveur pour le chiffrement
  - Tout client peut coder avec la clé publique *voyages-sncf.fr*
- Une clé privée pour le déchiffrement
  - Uniquement *voyages-sncf.fr* peut décoder les messages chiffrés avec sa clé publique
- Impossible de retrouver une clé à partir de l'autre

Clé publique via l'arithmétique modulaire

Soit le premier exemple présenté précédemment :

- *msg*=[14, 12, 11, 9, 5] – tableau de numéros entre 0 et 15
- L'objectif : coder ces numéros sur 4 bits

Rappel : une seule clé symétrique S

- Codage : calculer $E(msg, S) = msg + S \pmod{16}$
- Décodage : calculer $E^{-1}(code, S) = code - S \pmod{16}$
- $E^{-1}(E(msg, S), S) = msg + S - S = msg \pmod{16}$

Clé publique K, clé privée K'

- Codage : calculer $msg \times K \pmod{16}$
- Décodage : calculer $code \times K' \pmod{16}$
- Codage suivi par décodage : $msg \times K \times K' \pmod{16}$

Clé publique via l'arithmétique modulaire

Soit le premier exemple présenté précédemment :

- msg=[14, 12, 11, 9, 5] – tableau de numéros entre 0 et 15
- L'objectif : coder ces numéros sur 4 bits

Rappel : une seule clé symétrique S

- Codage : calculer $E(msg, S) = msg + S \pmod{16}$
- Décodage : calculer $E^{-1}(code, S) = code - S \pmod{16}$
- $E^{-1}\left(E(msg, S), S\right) = msg + S - S = msg \pmod{16}$

Clé publique K, clé privée K'

- Codage : calculer $msg \times K \pmod{16}$
- Décodage : calculer $code \times K' \pmod{16}$
- Codage suivi par décodage : $msg \times K \times K' \pmod{16}$

Clé publique via l'arithmétique modulaire

Soit le premier exemple présenté précédemment :

- *msg*=[14, 12, 11, 9, 5] – tableau de numéros entre 0 et 15
- L'objectif : coder ces numéros sur 4 bits

Formule de base

- Codage suivi par décodage = message initial

$$E^{-1}\left(E(msg, K), K'\right) = msg$$

- Notre cas : $E^{-1}\left(E(msg, K), K'\right) = msg \times K \times K' \pmod{16}$
- Question : $msg \times K \times K' = msg$?
- Oui, si la clé publique K et la clé privé K' vérifient $K \times K' = 1$
  - Exemple : $K = 3$ et $K' = 11$
  - Facile à vérifier $3 \times 11 = 1 \pmod{16}$
    - Plus difficile à trouver 11 à partir de 3

Clé publique via l'arithmétique modulaire

Soit le premier exemple présenté précédemment :

- *msg*=[14, 12, 11, 9, 5] – tableau de numéros entre 0 et 15
- L'objectif : coder ces numéros sur 4 bits

Formule de base

- Codage suivi par décodage = message initial

$$E^{-1}\left(E(msg, K), K'\right) = msg$$

- Notre cas : $E^{-1}\left(E(msg, K), K'\right) = msg \times K \times K' \pmod{16}$
- Question : $msg \times K \times K' = msg$?
- Oui, si la clé publique K et la clé privé K' vérifient $K \times K' = 1$
  - Exemple : $K = 3$ et $K' = 11$
  - Facile à vérifier $3 \times 11 = 1 \pmod{16}$
    - Plus difficile à trouver 11 à partir de 3

Exemple

Soit le premier exemple présenté précédemment :

- *msg*=[14, 12, 11, 9, 5] – tableau de numéros entre 0 et 15
- L'objectif : coder ces numéros sur 4 bits

Codage et décodage : exemple

- Codage avec $K = 3$ suivi par décodage avec $K' = 11$:
  - $E(14, 3) = 14 \times 3 = 10 \pmod{16}$
  - $E^{-1}(10, 11) = 10 \times 11 = 110 = 14 \pmod{16}$
- Exemple complet :

msg :	14	12	11	9	5
		$\times 3 (mod 16)$			
code :	10	4	1	11	15
		$\times 11 (mod 16)$			
msg :	14	12	11	9	5

- $E([14, 12, 11, 9, 5], 3) = [14 \times 3, 12 \times 3, 11 \times 3, 9 \times 3, 5 \times 3]$
  $= [10, 4, 1, 11, 15] \pmod{16}$

Fonctionnement sur Internet : https/ssh

A veut communiquer avec un serveur *B*.

1. *B* envoie sa clé publique K_B à *A*
2. *A* génère une unique clé de session, la code avec K_B et l'envoie à B
3. *B* décode la clé de session avec sa clé privée K'_B
4. Les messages sont ensuite codés avec la clé de session
   - Le pirate n'a pas accès à la clé de session, car il n'a pas K'_B
   - Cette clé est un secret partagé

Il faut pouvoir contourner un éventuel pirate *P* qui pourrait altérer toute communication $A \leftrightarrow B$

Vulnérabilité : *A* ne sais pas K_B est bien la clé publique de *B* ou une clé envoyée par le pirate *P*

Fonctionnement sur Internet : https/ssh

A veut communiquer avec un serveur *B*.

1. *B* envoie sa clé publique K_B à *A*
2. *A* génère une unique clé de session, la code avec K_B et l'envoie à B
3. *B* décode la clé de session avec sa clé privée K'_B
4. Les messages sont ensuite codés avec la clé de session
   - Le pirate n'a pas accès à la clé de session, car il n'a pas K'_B
   - Cette clé est un secret partagé

Il faut pouvoir contourner un éventuel pirate *P* qui pourrait altérer toute communication $A \leftrightarrow B$

Vulnérabilité : *A* ne sais pas K_B est bien la clé publique de *B* ou une clé envoyée par le pirate *P*

Fonctionnement sur Internet : https/ssh

A veut communiquer avec un serveur *B*.

---

1. *B* envoie sa clé publique K_B à *A*
2. *A* génère une unique clé de session, la code avec K_B et l'envoie à B
3. *B* décode la clé de session avec sa clé privée K'_B
4. Les messages sont ensuite codés avec la clé de session
   - Le pirate n'a pas accès à la clé de session, car il n'a pas K'_B
   - Cette clé est un secret partagé

Il faut pouvoir contourner un éventuel pirate *P* qui pourrait altérer toute communication $A \leftrightarrow B$

Vulnérabilité : *A* ne sais pas K_B est bien la clé publique de *B* ou une clé envoyée par le pirate *P*

Fonctionnement sur Internet : https/ssh

A veut communiquer avec un serveur B.

---

1. B envoie sa clé publique K_B à A
2. A génère une unique clé de session, la code avec K_B et l'envoie à B
3. B décode la clé de session avec sa clé privée K'_B
4. Les messages sont ensuite codés avec la clé de session
   - Le pirate n'a pas accès à la clé de session, car il n'a pas K'_B
   - Cette clé est un secret partagé

Il faut pouvoir contourner un éventuel pirate P qui pourrait altérer toute communication $A \leftrightarrow B$

Vulnérabilité : A ne sais pas K_B est bien la clé publique de B ou une clé envoyée par le pirate P

L'homme du milieu pour HTTP

Garantir l'Identité de B : authentifier la clé publique K_b

HTTP Au début, B n'envoie pas sa clé publique à A, mais un message contenant l'identité de B et K_B : ce message est codé avec la clé privée K'_{AC} d'une AC (autorité de certification)

- La clé publique K_{AC} est connue par tout navigateur Internet
- A est certain d'avoir la bonne clé K_B, et le pirate ne peut pas décoder les messages codés avec K_B

L'homme du milieu pour HTTP

Garantir l'Identité de B : authentifier la clé publique K_b

HTTP Au début, B n'envoie pas sa clé publique à A, mais un message contenant l'identité de B et K_B : ce message est codé avec la clé privée K'_{AC} d'une AC (autorité de certification)

- La clé publique K_{AC} est connue par tout navigateur Internet
- A est certain d'avoir la bonne clé K_B, et le pirate ne peut pas décoder les messages codés avec K_B

L'homme du milieu : SSH

Garantir l'Identité de B : authentifier la clé publique K_b

SSH Au début, A doit faire confiance à la clé K_B.

- l'échange de clés Diffie-Hellman : A et B peuvent se mettre d'accord sur un nombre sans qu'un pirate P puisse découvrir le nombre, même en ayant écouté tous leurs échanges

3. Cryptographie : fondements mathématiques et attaques
   - Systèmes à clés privés/symétriques
   - Systèmes à clés publiques
   - Groupes : l'échange des clés Diffie-Hellman $\oplus$ idées RSA

Les groupes : base de Diffie-Hellman et RSA

Un groupe $G = (S, \times)$ est un ensemble S muni d'une loi de composition $\times$ qui respecte plusieurs propriétés :

1. $\forall a, b \in S \Rightarrow a \times b \in S$
2. Associativité : $(a \times b) \times c = a \times (b \times c)$
3. Élément neutre : il existe un $e \in S$ tel que $\forall a \in S, a \times e = a$
4. Symétrie : $\forall a \in S$, il existe a^{-1} tel que $a^{-1} \times a = e$

Exemple : le groupe $Z_p = (S, \times)$, noté aussi Z/pZ

- $S = \{1, 2, \ldots p-1\}$
- Loi de composition $\times$: la multiplication modulo p
  - p premier $\Rightarrow$ groupe valide
  - $p = p_1 \cdot p_2$ pas premier $\Rightarrow$ groupe ? NON, car $p_1 \times p_2 \notin S$

Les groupes : base de Diffie-Hellman et RSA

Un groupe $G = (S, \times)$ est un ensemble S muni d'une loi de composition $\times$ qui respecte plusieurs propriétés :

1. $\forall a, b \in S \Rightarrow a \times b \in S$
2. Associativité : $(a \times b) \times c = a \times (b \times c)$
3. Élément neutre : il existe un $e \in S$ tel que $\forall a \in S, a \times e = a$
4. Symétrie : $\forall a \in S$, il existe a^{-1} tel que $a^{-1} \times a = e$

Exemple : le groupe $Z_p = (S, \times)$, noté aussi Z/pZ

- $S = \{1, 2, \ldots p-1\}$
- Loi de composition $\times$: la multiplication modulo p
  - p premier $\Rightarrow$ groupe valide
  - $p = p_1 \cdot p_2$ pas premier $\Rightarrow$ groupe ? NON, car $p_1 \times p_2 \notin S$

Les groupes : base de Diffie-Hellman et RSA

Un groupe $G = (S, \times)$ est un ensemble S muni d'une loi de composition $\times$ qui respecte plusieurs propriétés :

1. $\forall a, b \in S \Rightarrow a \times b \in S$
2. Associativité : $(a \times b) \times c = a \times (b \times c)$
3. Élément neutre : il existe un $e \in S$ tel que $\forall a \in S, a \times e = a$
4. Symétrie : $\forall a \in S$, il existe a^{-1} tel que $a^{-1} \times a = e$

Exemple : le groupe $Z_p = (S, \times)$, noté aussi Z/pZ

- $S = \{1, 2, \ldots p-1\}$
- Loi de composition $\times$: la multiplication modulo p
  - p premier $\Rightarrow$ groupe valide
  - $p = p_1 \cdot p_2$ pas premier $\Rightarrow$ groupe ? NON, car $p_1 \times p_2 \notin S$

Les groupes : base de Diffie-Hellman et RSA

Un groupe $G = (S, \times)$ est un ensemble S muni d'une loi de composition $\times$ qui respecte plusieurs propriétés :

1. $\forall a, b \in S \Rightarrow a \times b \in S$
2. Associativité : $(a \times b) \times c = a \times (b \times c)$
3. Élément neutre : il existe un $e \in S$ tel que $\forall a \in S, a \times e = a$
4. Symétrie : $\forall a \in S$, il existe a^{-1} tel que $a^{-1} \times a = e$

Exemple : le groupe $Z_p = (S, \times)$, noté aussi Z/pZ

- $S = \{1, 2, \ldots p-1\}$
- Loi de composition $\times$: la multiplication modulo p
  - p premier $\Rightarrow$ groupe valide
  - $p = p_1 \cdot p_2$ pas premier $\Rightarrow$ groupe ? NON, car $p_1 \times p_2 \notin S$

Les groupes : base de Diffie-Hellman et RSA

Un groupe $G = (S, \times)$ est un ensemble S muni d'une loi de composition $\times$ qui respecte plusieurs propriétés :

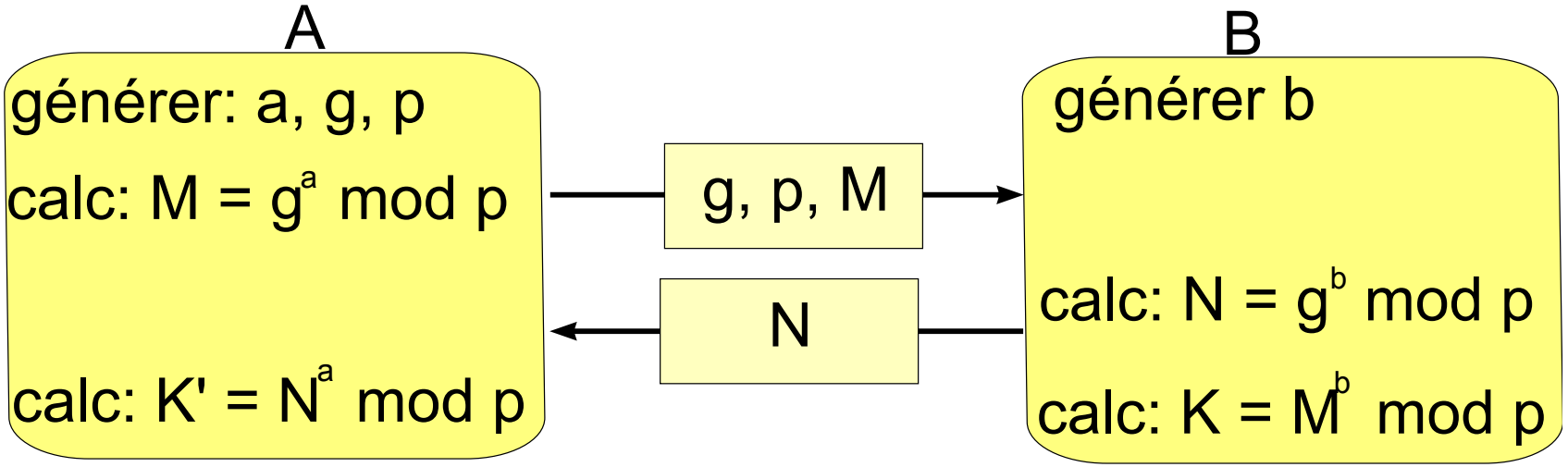
1. $\forall a, b \in S \Rightarrow a \times b \in S$
2. Associativité : $(a \times b) \times c = a \times (b \times c)$
3. Élément neutre : il existe un $e \in S$ tel que $\forall a \in S, a \times e = a$
4. Symétrie : $\forall a \in S$, il existe a^{-1} tel que $a^{-1} \times a = e$

Exemple : le groupe $Z_p = (S, \times)$, noté aussi Z/pZ

- $S = \{1, 2, \ldots p-1\}$
- Loi de composition $\times$: la multiplication modulo p
  - p premier $\Rightarrow$ groupe valide
  - $p = p_1 \cdot p_2$ pas premier $\Rightarrow$ groupe ? NON, car $p_1 \times p_2 \notin S$

Le groupe Z_p dans l'algorithme D-H

L'algorithme D-H est basé sur le groupe $Z_p = (\{1, 2 \ldots p-1\}, \times)$ avec un premier p de quelques centaines de chiffres

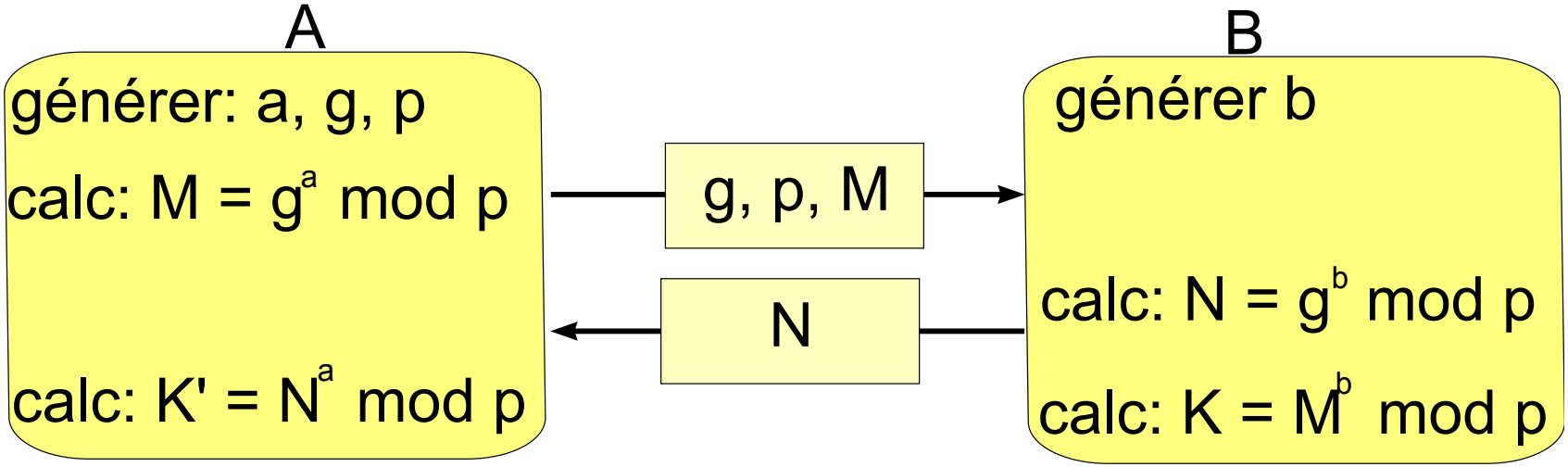


1. A génère : a, g, p et calcule $M = g^a \bmod p$
2. Transmission $A \rightarrow B : g, p, M$
3. B calcule $N = g^b \bmod p$
4. Transmission $B \rightarrow A : N$
5. A calcule $K' = N^a \bmod p$ et B calcule $K = M^b \bmod p$

Le groupe Z_p dans l'algorithme D-H

A détient la clé $K' = N^a = (g^b)^a = g^{ab} \bmod p$

B détient la clé $K = M^b = (g^a)^b = g^{ab} \bmod p$



Pourrait le pirate espérer calculer g^{ab} ?

- Pour calculer g^{ab}, il faut trouver a ou $b \rightarrow$ Le problème du logarithme discret : trouver $\boxed{a}$ à partir de $\boxed{g^a \bmod p}$
- Exemples de calculs pour : $p = 23$, $g = 5$ et $a = 16$.
  - A : $g^a = 5^{16} = (((5^2)^2)^2)^2 = (4^2)^2 = 3$
  - Le pirate : $5^1 = 3$? NO, $5^2 = 3$? NO, ... 5^{16}=3 ? $\boxed{\text{OUI}}$

Les idées de base d'algorithme RSA

RSA : le plus connu algorithme de cryptographie asymétrique :

- le plus utilisé algorithme dans le commerce électronique,
- breveté par MIT (Massachusetts Institute of Technology) en 1983
- comme DH, il est basé sur certaines propriétés du groupe Z_p (noté aussi Z/pZ)

Les groupes dans l'algorithme RSA

Soit $G(S, \times)$ un groupe et $a, b, q \in S$.

- Si $a \neq b$, alors $a \times q \neq b \times q$. Pourquoi ?
- $S = \{1, 2 \ldots p-1\} = \{1 \times q, 2 \times q \ldots (p-1) \times q\}$
- $prod = 1 \times 2 \times 3 \ldots \times (p-1) = 1 \times q \times 2 \times q \cdots \times (p-1) \times q$
- $prod = prod \times q \times q \times q \ldots \times q = prod \times q^{p-1}$
- Petit théorème de Fermat : $q^{p-1} = 1$
- Généralisation du théorème : $q^{\phi(p)} = 1$
  - $\phi(p) =$ le nombre de numéros $p' \in S$ premiers avec p
- Si $K \times K' = \phi(p) \cdot \ell + 1$, alors : K=clé privée et K'=clé publique
  - Codage : $b = E(a, K) = a^K \pmod p$
  - Décodage : $E(b, K') = b^{K'} = (a^K)^{K'} = a^{K \times K'} = a^{\phi(p) \cdot \ell + 1} = (a^{\phi(p)})^\ell \times a = a$

Les groupes dans l'algorithme RSA

Soit $G(S, \times)$ un groupe et $a, b, q \in S$.

- Si $a \neq b$, alors $a \times q \neq b \times q$. Pourquoi ?
- $S = \{1, 2 \ldots p-1\} = \{1 \times q, 2 \times q \ldots (p-1) \times q\}$
- $prod = 1 \times 2 \times 3 \ldots \times (p-1) = 1 \times q \times 2 \times q \cdots \times (p-1) \times q$
- $prod = prod \times q \times q \times q \ldots \times q = prod \times q^{p-1}$
- Petit théorème de Fermat : $q^{p-1} = 1$
- Généralisation du théorème : $q^{\phi(p)} = 1$
  - $\phi(p) =$ le nombre de numéros $p' \in S$ premiers avec p
- Si $K \times K' = \phi(p) \cdot \ell + 1$, alors : K=clé privée et K'=clé publique
  - Codage : $b = E(a, K) = a^K \pmod p$
  - Décodage : $E(b, K') = b^{K'} = (a^K)^{K'} = a^{K \times K'} = a^{\phi(p) \cdot \ell + 1} = (a^{\phi(p)})^\ell \times a = a$

Les groupes dans l'algorithme RSA

Soit $G(S, \times)$ un groupe et $a, b, q \in S$.

- Si $a \neq b$, alors $a \times q \neq b \times q$. Pourquoi ?
- $S = \{1, 2 \ldots p-1\} = \{1 \times q, 2 \times q \ldots (p-1) \times q\}$
- $prod = 1 \times 2 \times 3 \ldots \times (p-1) = 1 \times q \times 2 \times q \cdots \times (p-1) \times q$
- $prod = prod \times q \times q \times q \ldots \times q = prod \times q^{p-1}$
- Petit théorème de Fermat : $q^{p-1} = 1$
- Généralisation du théorème : $q^{\phi(p)} = 1$
  - $\phi(p) =$ le nombre de numéros $p' \in S$ premiers avec p
- Si $K \times K' = \phi(p) \cdot \ell + 1$, alors : K=clé privée et K'=clé publique
  - Codage : $b = E(a, K) = a^K \pmod p$
  - Décodage : $E(b, K') = b^{K'} = (a^K)^{K'} = a^{K \times K'} = a^{\phi(p) \cdot \ell + 1} = (a^{\phi(p)})^\ell \times a = a$

Les groupes dans l'algorithme RSA

Soit $G(S, \times)$ un groupe et $a, b, q \in S$.

- Si $a \neq b$, alors $a \times q \neq b \times q$. Pourquoi ?
- $S = \{1, 2 \ldots p-1\} = \{1 \times q, 2 \times q \ldots (p-1) \times q\}$
- $prod = 1 \times 2 \times 3 \ldots \times (p-1) = 1 \times q \times 2 \times q \cdots \times (p-1) \times q$
- $prod = prod \times q \times q \times q \ldots \times q = prod \times q^{p-1}$
- Petit théorème de Fermat : $q^{p-1} = 1$
- Généralisation du théorème : $q^{\phi(p)} = 1$
  - $\phi(p) =$ le nombre de numéros $p' \in S$ premiers avec p
- Si $K \times K' = \phi(p) \cdot \ell + 1$, alors : K=clé privée et K'=clé publique
  - Codage : $b = E(a, K) = a^K \pmod p$
  - Décodage : $E(b, K') = b^{K'} = (a^K)^{K'} = a^{K \times K'} = a^{\phi(p) \cdot \ell + 1} = (a^{\phi(p)})^{\ell} \times a = a$