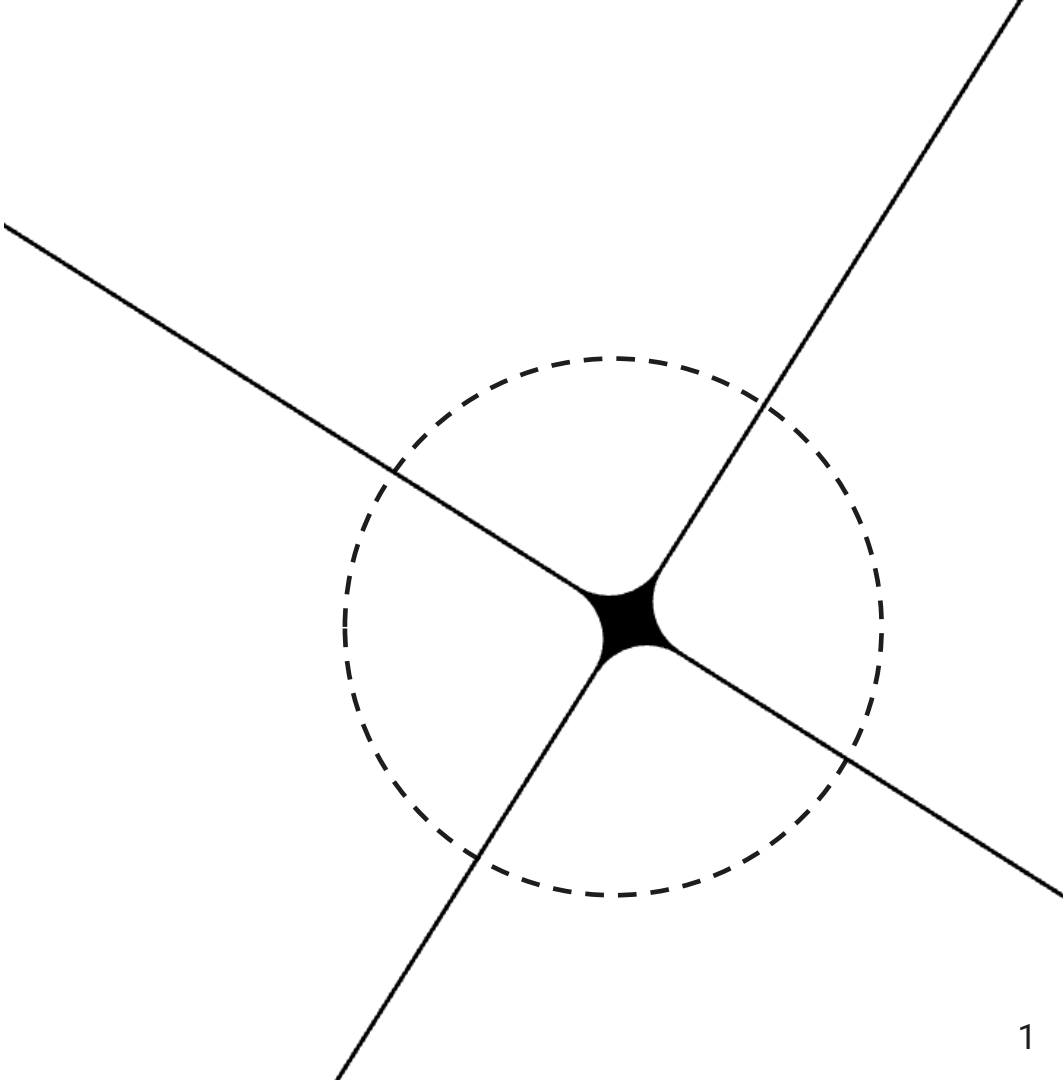
INSTITUT POLYTECHNIQUE DE PARIS

TELECOM SudParis

Introduction TP 6

Julien Romero



Dans ce TP…

Nous allons implémenter une suite de structures algorithmiques très utilisées en pratique:

- La liste doublement chaînée (objectif principal)
- Table d'association (objectif principal)
- Ensemble/Set (entraînement)
- File (entraînement)
- Pile (entraînement)

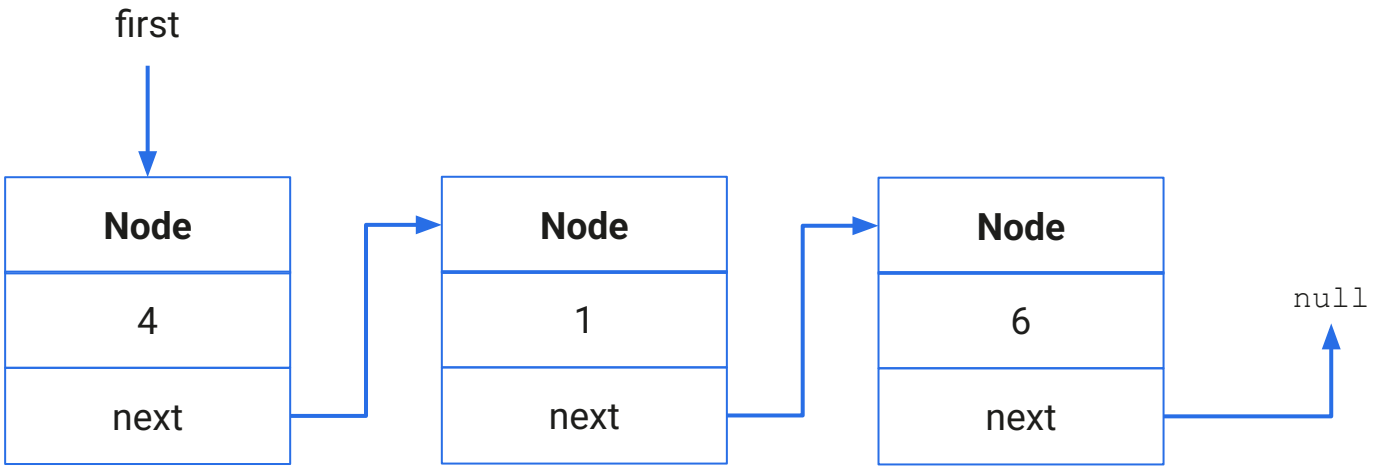
Tout cela avec des classes génériques !

La liste doublement chaînée - Rappels

Rappel: Au TP2, nous avons vu la liste chaînée
Limitations:

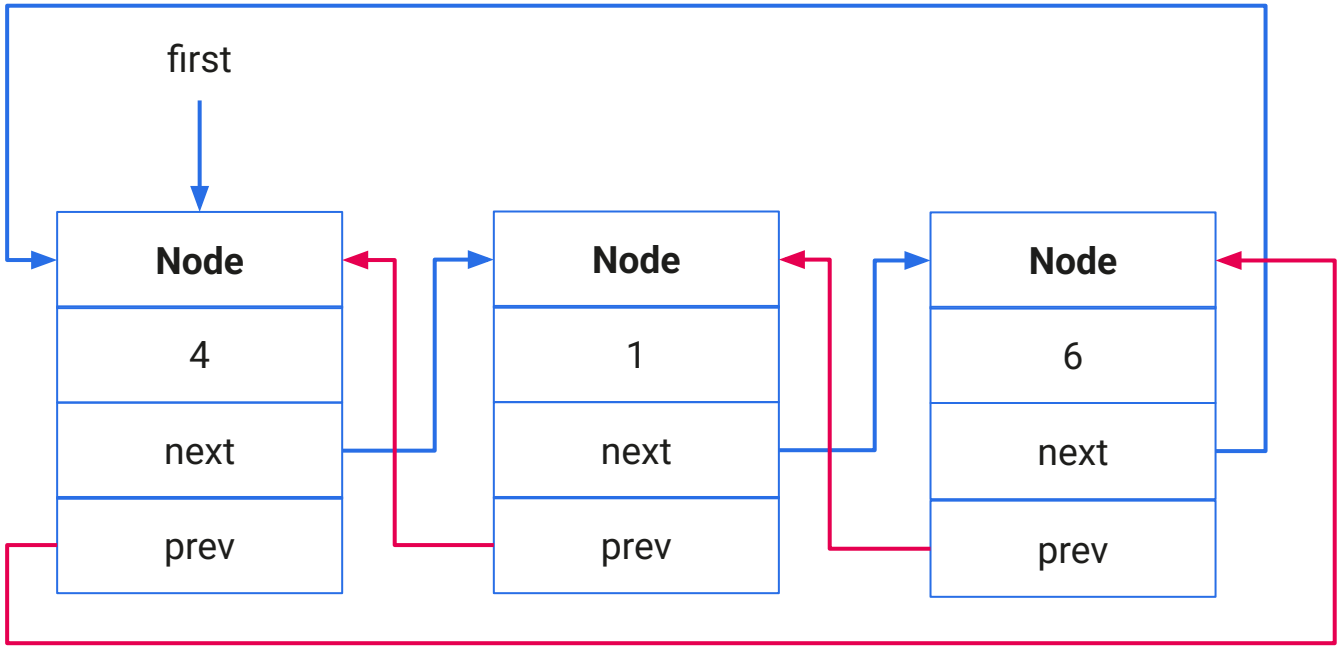
- On ne peut parcourir la liste que dans une seule direction
- Supprimer un nœud donné est cher (on doit parcourir la liste jusqu'à ce nœud)
- Ajouter un nœud avant un nœud donné est cher (on doit parcourir la liste jusqu'à ce nœud)

Solution: conserver une référence vers le nœud précédent grâce à la liste doublement chaînée



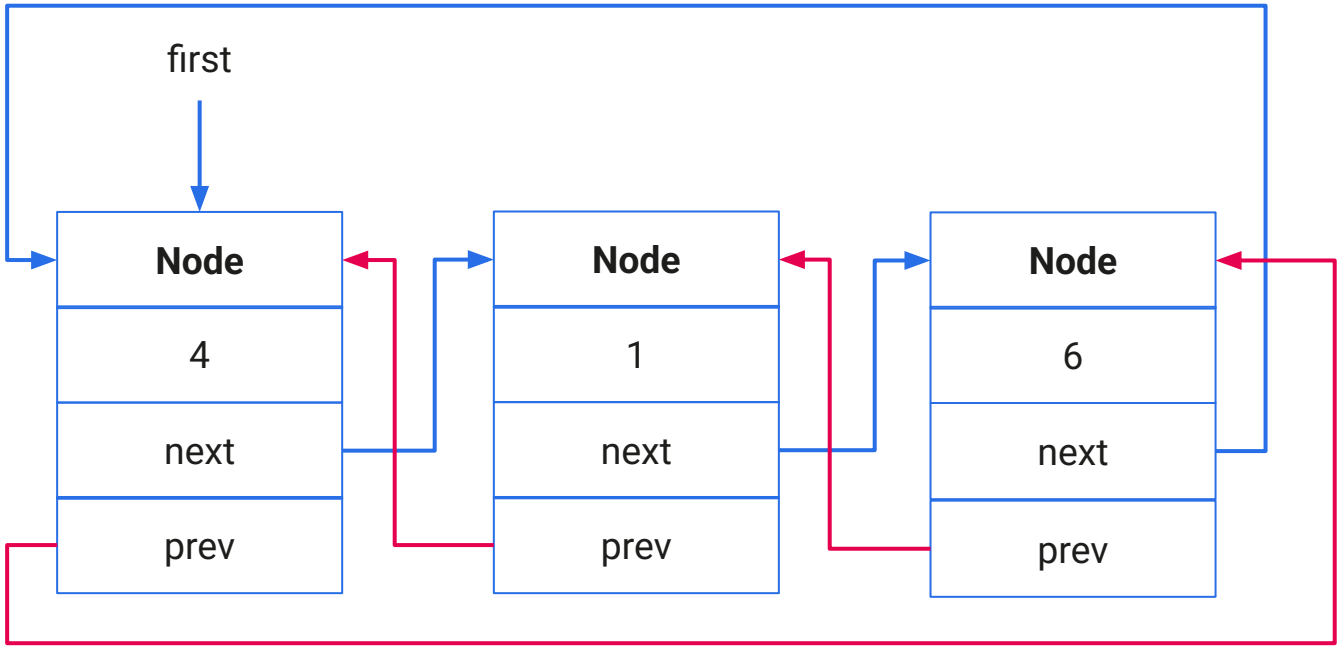
La liste doublement chaînée

Idée: Dans une liste doublement chaînée, on ajoute une référence vers le nœud précédent, et (potentiellement) le premier et dernier nœuds sont connectés.



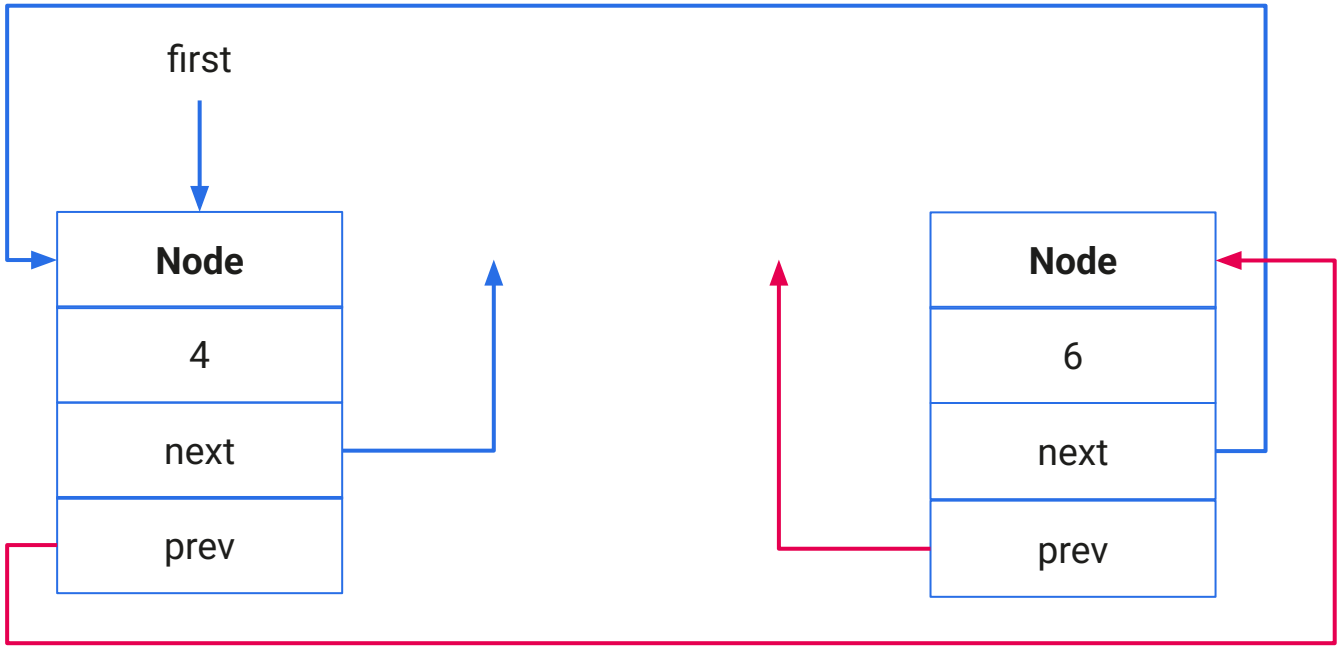
La liste doublement chaînée - Suppression

Pour supprimer un nœud donné, il suffit de connecter le successeur de son prédécesseur au nœud suivant, et, de même de connecter le prédécesseur de son successeur à son prédécesseur.



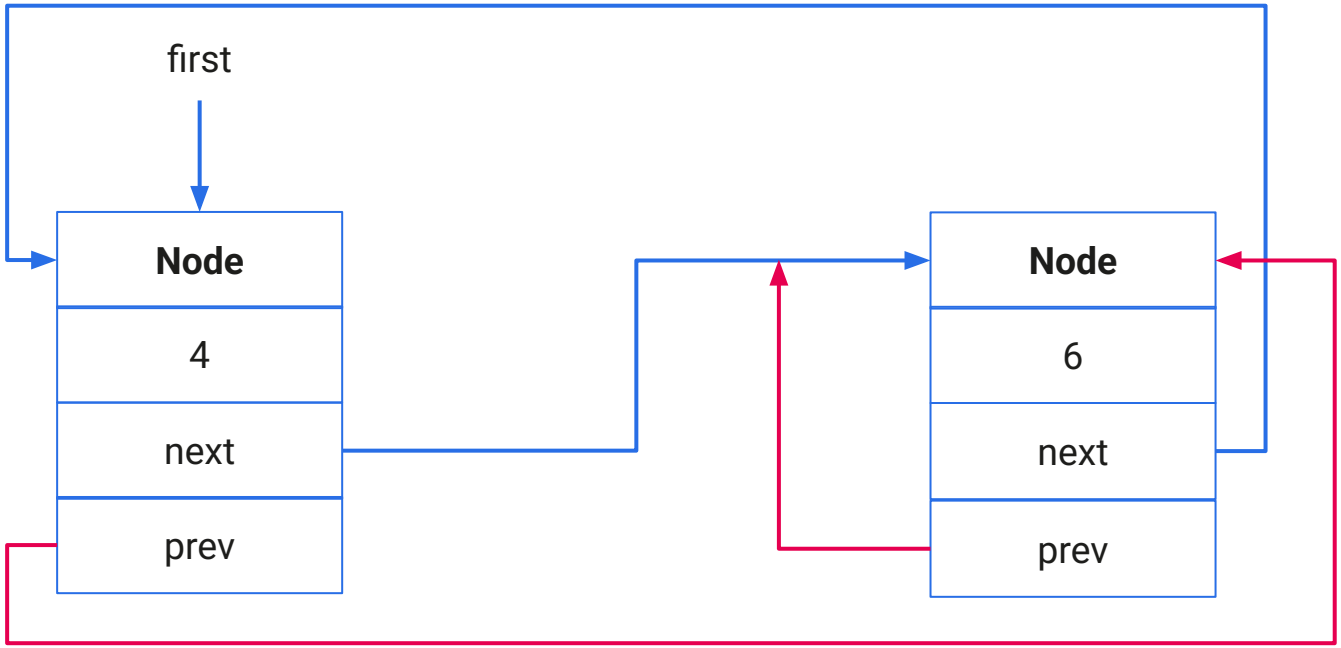
La liste doublement chaînée - Suppression

Pour **supprimer un nœud donné**, il suffit de connecter le successeur de son prédécesseur au nœud suivant, et, de même de connecter le prédécesseur de son successeur à son prédécesseur.



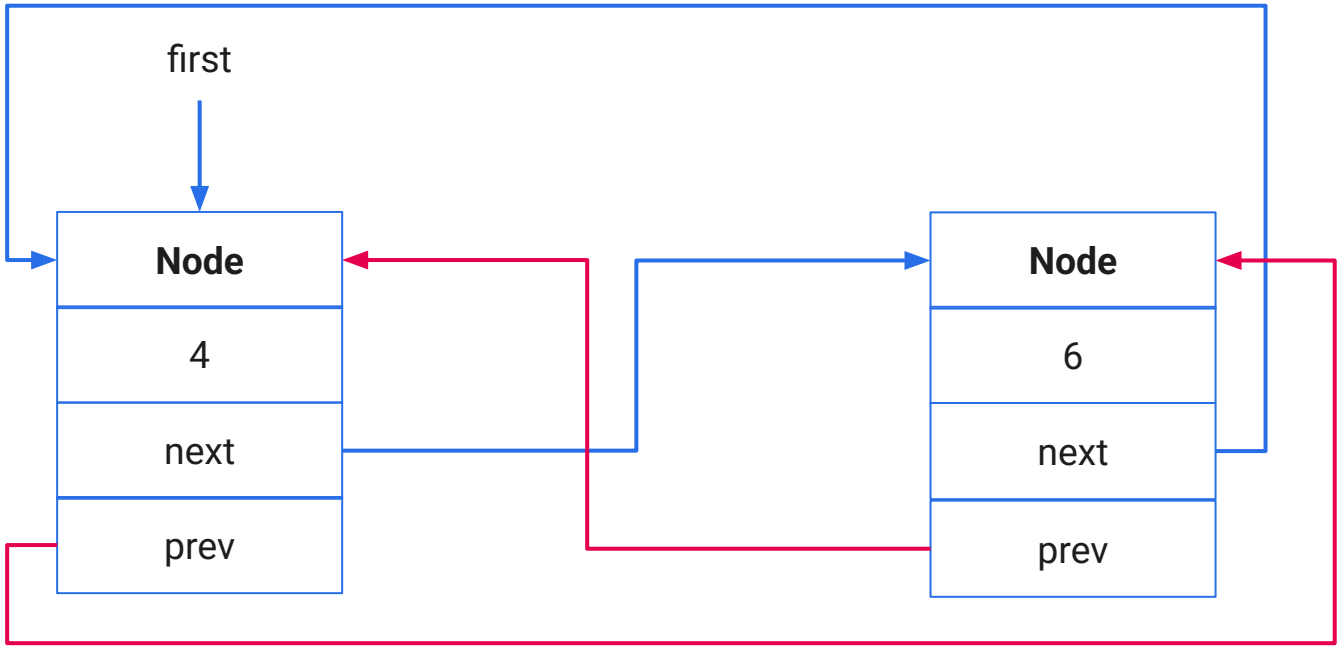
La liste doublement chaînée - Suppression

Pour supprimer un nœud donné, il suffit de **connecter le successeur de son prédécesseur au nœud suivant**, et, de même de connecter le prédécesseur de son successeur à son prédécesseur.



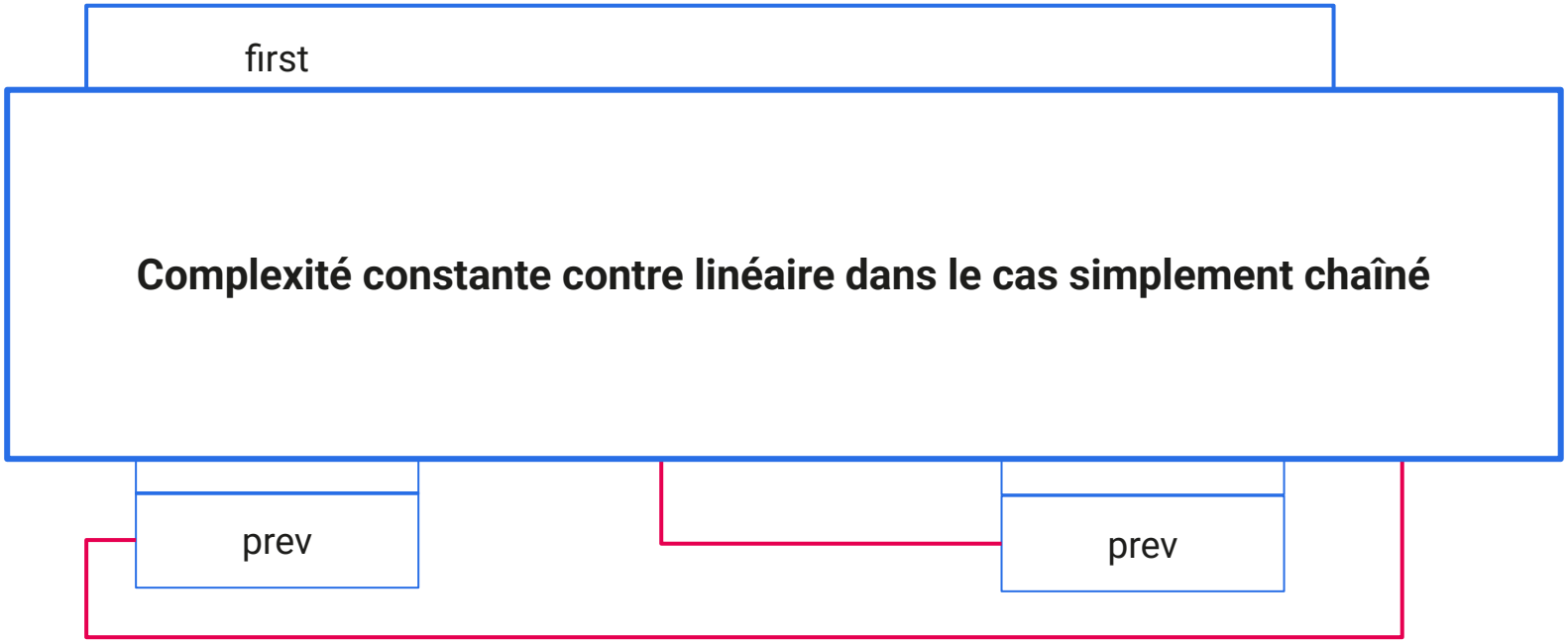
La liste doublement chaînée - Suppression

Pour supprimer un nœud donné, il suffit de connecter le successeur de son prédécesseur au nœud suivant, et, de même de **connecter le prédécesseur de son successeur à son prédécesseur**.



La liste doublement chaînée - Suppression

Pour supprimer un nœud donné, il suffit de connecter le successeur de son prédécesseur au nœud suivant, et, de même de **connecter le prédécesseur de son successeur à son prédécesseur**.



La table d'association

- Une table d'association est une structure de donnée associant des clefs à des valeurs.
  - Similaire au dictionnaire en Python
  - Map en anglais
- En général, nous avons les méthodes suivantes:
  - **boolean put(K key, V value)** : crée une association et renvoie vrai si la clé n'avait aucune association auparavant,
  - **V get(K key)** : renvoie la valeur associée à key si elle existe et null sinon,
  - **boolean remove(K key)** : supprime l'association de key et renvoie vrai si elle existait,
- On pourrait l'implémenter avec un tableau ou une liste (doublement chaînée) où chaque élément représente un couple clef/valeur
  - Cependant, les complexités ne sont pas bonnes (linéaires)
- On peut obtenir des complexités constantes (en moyenne) avec les tables de hachages

Les fonctions de hachage

Une **fonction de hachage** est une fonction qui prend une entrée de taille quelconque et la transforme en une sortie de taille fixe, appellée hash value, digest, ou hash code.

Les fonctions de hachage sont souvent utilisées en cryptographie, et possèdent souvent les propriétés suivantes:

- **Déterministique**: Une entrée donnée donne toujours la même sortie
- **Résistance à la préimage**: Il est pratiquement impossible de retrouver l'entrée menant à une sortie donnée
- **Résistance aux collisions**: Il est pratiquement impossible de trouver deux entrées avec la même sortie
- **Effet avalanche:** Une petite perturbation de l'entrée donne de grandes perturbations sur la sorte (sauf cas particuliers, voir MinHash)
- **Taille fixe:** La sortie a toujours la même taille
- **Efficacité:** Le calcul est rapide

Dans ce TP, nous sommes surtout intéressés par la **résistance aux collisions**.
En Java, la fonction de hachage se calcule pour une classe en redéfinissant `public int hashCode()` (par défaut, il s'agit d'une fonction basée sur l'adresse de l'objet)

Culture: La gestion des mots de passe

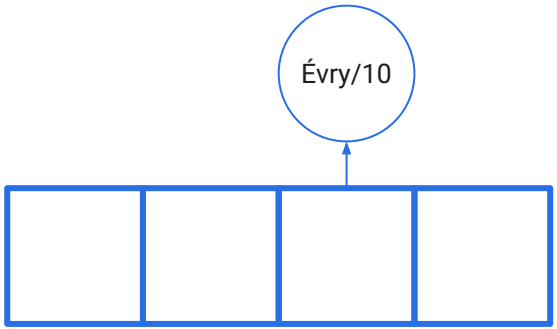
- Quand une application ou un site web doit enregistrer le mot de passe d'un utilisateur, il ne doit **jamais** enregistrer le mot de passe en clair
  - Trop risqué en cas de fuite de données (voir https://haveibeenpwned.com/)
- On enregistre toujours le **mot de passe chiffré avec une fonction de hachage**
  - La résistance à la préimage est alors très importante
  - En général, Argon2id, bcrypt, scrypt, PBKDF2
- Pour vérifier un mot de passe, on calcule son hash et on le compare au hash stocké
  - La résistance aux collisions est très importante ici

La table de hachage

- **Idée**: Nous allons stocker les paires de clef/valeur dans un tableau où chaque éléments est une table d’association utilisant une liste doublement chaînée. La case de la paire clef/valeur est déterminée par une fonction de hachage modulo la taille du tableau, appliquée sur la clef.
- Une bonne fonction de hachage devrait répartir uniformément les éléments
- Comme pour le tableau extensible, on redimensionne le tableau quand il y a trop d'éléments (et donc de collisions)

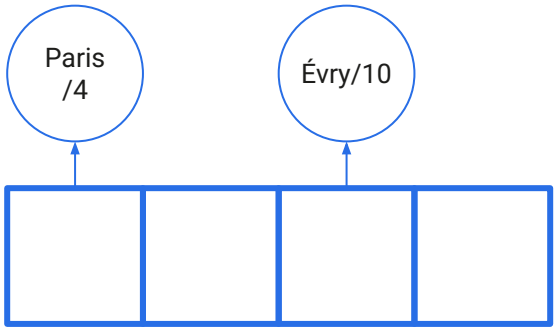
La table de hachage - Insertion

Évry -> 10: hash(“Évry”) % 4 = 2



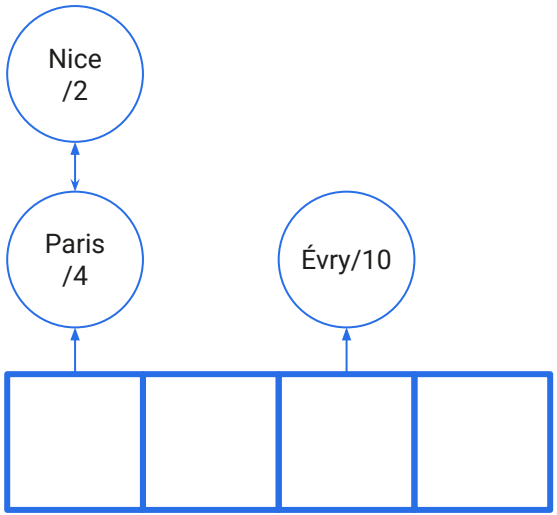
La table de hachage - Insertion

Paris -> 4: hash(“Paris”) % 4 = 0



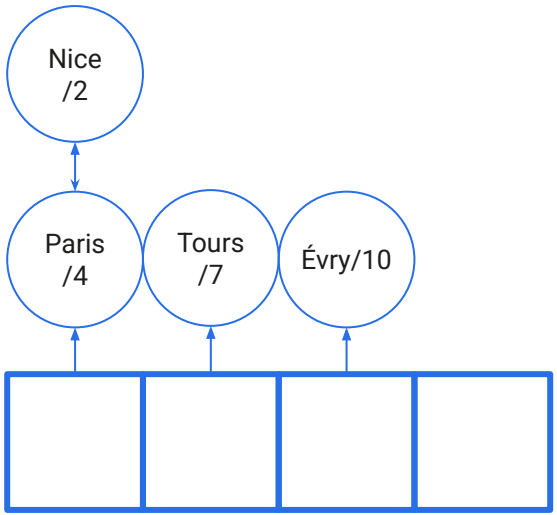
La table de hachage - Insertion

Nice -> 2: hash(“Nice”) % 4 = 0 **Collision**



La table de hachage - Insertion

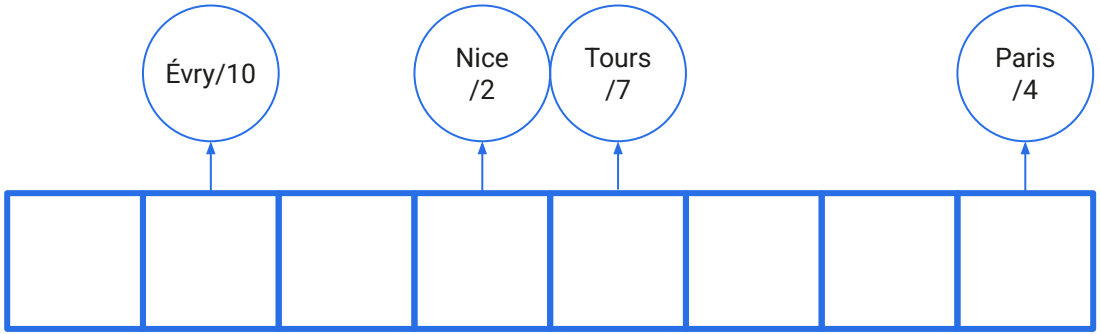
Tours -> 7: hash(“Tours”) % 4 = 1



La table de hachage - Redimensionnement

- Quand on atteint un certain nombre d'éléments, on double la taille du tableau
  - Il faut réinsérer les éléments, potentiellement à de nouveaux indices

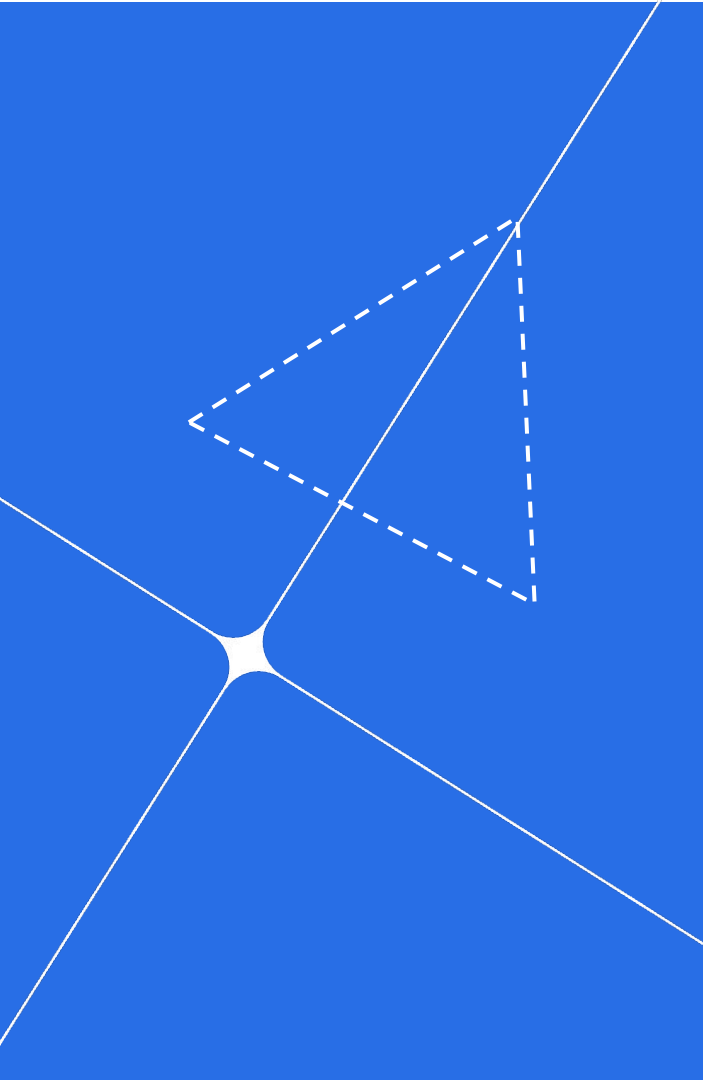
Évry -> 10: hash("Évry") **% 8** = 1
Paris -> 4: hash("Paris") % 8 = 7
Nice -> 2: hash("Nice") % 8 = 3
Tours -> 7: hash("Tours") % 8 = 4



Autres structures de données

Les plus avancés implémenteront :

- **L'ensemble/Set** : structure de donnée où l'opération d'ajout et de vérification d'appartenance est rapide
  - Implémenté avec une table de hachage sans valeur
- **La file/Queue** : structure de donnée permettant d'ajouter des éléments et de les récupérer dans l'ordre d'ajout
- **La Pile/Stack** : structure de donnée permettant d'ajouter des éléments et de les récupérer dans l'ordre **inverse** d'ajout



En route pour le TP