INSA INSTITUT NATIONAL DES SCIENCES APPLIQUÉES CENTRE VAL DE LOIRE

Listes chaînées

Hugo RAGUET

STPI 2A 2020–21

Algorithmique et programmation 3 & 4

Structures de données

Une structure de données est *une façon de ranger des objets*

Structures de données

Une structure de données est *une façon de ranger des objets*

La taille d'une structure de données est *le nombre d'objets*

Structures de données

Une structure de données est *une façon de ranger des objets*

La taille d'une structure de données est *le nombre d'objets*

Une structure de données est caractérisée par la façon dont

- on accède à un objet particulier
- on ajoute ou supprime un objet

Structures de données

Une structure de données est *une façon de ranger des objets*

La taille d'une structure de données est *le nombre d'objets*

Une structure de données est caractérisée par la façon dont

- on accède à un objet particulier
- on ajoute ou supprime un objet

et par les complexités de ces opérations

Structures de données

Une structure de données est *une façon de ranger des objets*

La taille d'une structure de données est *le nombre d'objets*

Une structure de données est caractérisée par la façon dont

- on accède à un objet particulier
- on ajoute ou supprime un objet

et par les complexités de ces opérations généralement en fonction de la taille de la structure

Tableaux et listes chaînées

Structures de données linéaires

Structure de donnée linéaire : séquence d'objets ordonnés

Tableaux et listes chaînées

Structures de données linéaires

Tableaux

Tableaux et listes chaînées

Structures de données linéaires

Tableaux

Accès aux éléments :

Tableaux et listes chaînées

Structures de données linéaires

Tableaux

Accès aux éléments : avec des indices
`t(i)` donne accès en lecture et en écriture au i^e élément

Tableaux et listes chaînées

Structures de données linéaires

Tableaux

Accès aux éléments : avec des indices en temps constant
`t(i)` donne accès en lecture et en écriture au i^e élément

Comment ?

Tableaux et listes chaînées

Structures de données linéaires

Tableaux

Accès aux éléments : avec des indices en temps constant
`t(i)` donne accès en lecture et en écriture au i^e élément

Comment ? en assurant la contigüité en mémoire

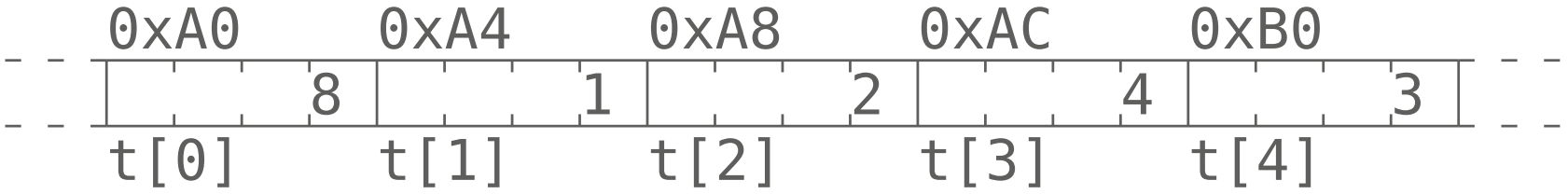
Tableaux et listes chaînées

Structures de données linéaires

Tableaux

Accès aux éléments : avec des indices en temps constant
`t(i)` donne accès en lecture et en écriture au i^e élément

Comment ? en assurant la contigüité en mémoire



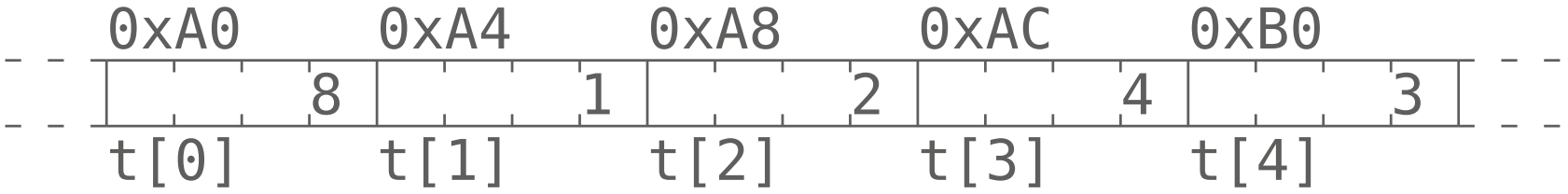
Tableaux et listes chaînées

Structures de données linéaires

Tableaux

Accès aux éléments : avec des indices en temps constant
`t(i)` donne accès en lecture et en écriture au i^e élément

Comment ? en assurant la contigüité en mémoire



adresse i^e élément =
adresse début + $(i-1)$ × espace mémoire d'un élément

Tableaux et listes chaînées

Structures de données linéaires

Tableaux

Accès aux éléments : avec des indices en temps constant

Ajout et suppression d'éléments :

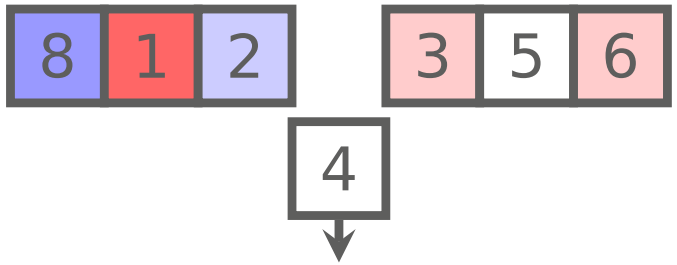
Tableaux et listes chaînées

Structures de données linéaires

Tableaux

Accès aux éléments : avec des indices en temps constant

Ajout et suppression d'éléments :



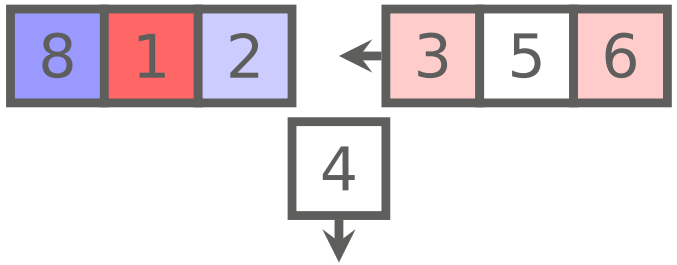
Tableaux et listes chaînées

Structures de données linéaires

Tableaux

Accès aux éléments : avec des indices en temps constant

Ajout et suppression d'éléments :



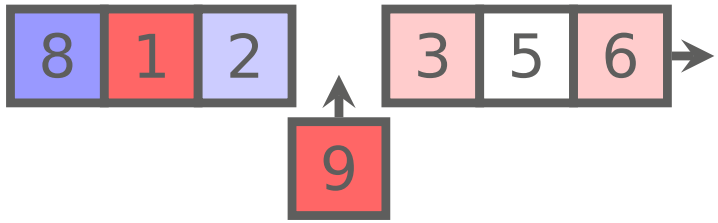
Tableaux et listes chaînées

Structures de données linéaires

Tableaux

Accès aux éléments : avec des indices en temps constant

Ajout et suppression d'éléments :



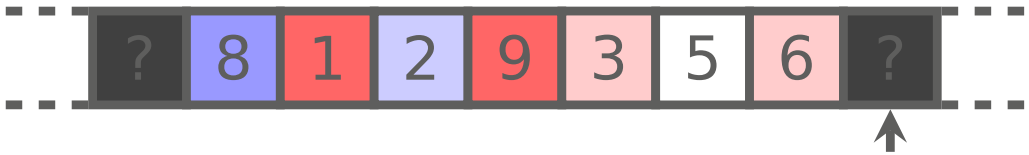
Tableaux et listes chaînées

Structures de données linéaires

Tableaux

Accès aux éléments : avec des indices en temps constant

Ajout et suppression d'éléments :



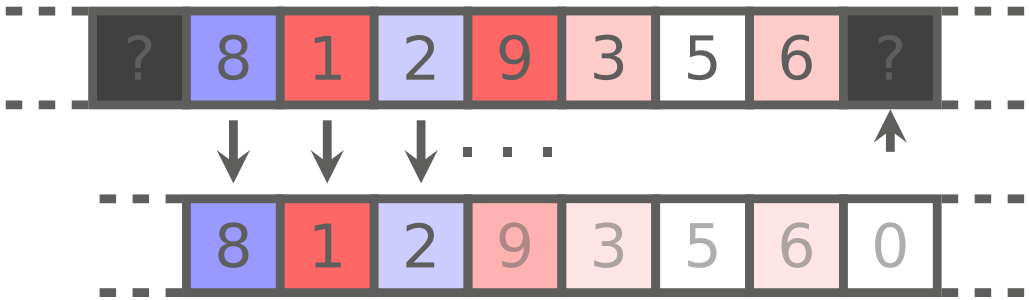
Tableaux et listes chaînées

Structures de données linéaires

Tableaux

Accès aux éléments : avec des indices en temps constant

Ajout et suppression d'éléments :



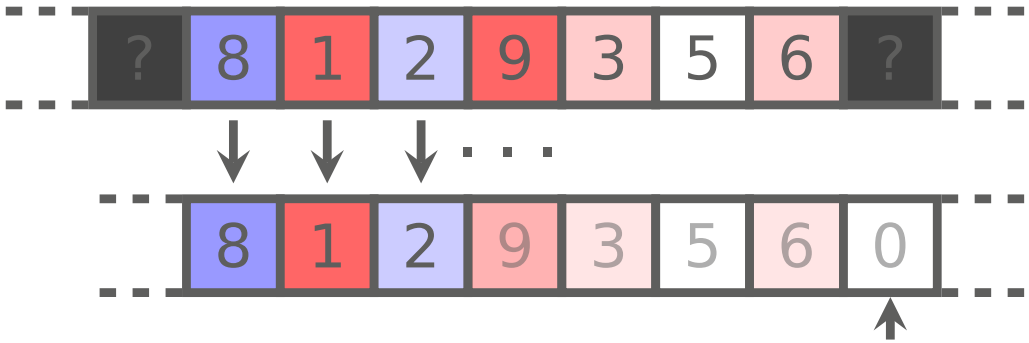
Tableaux et listes chaînées

Structures de données linéaires

Tableaux

Accès aux éléments : avec des indices en temps constant

Ajout et suppression d'éléments :



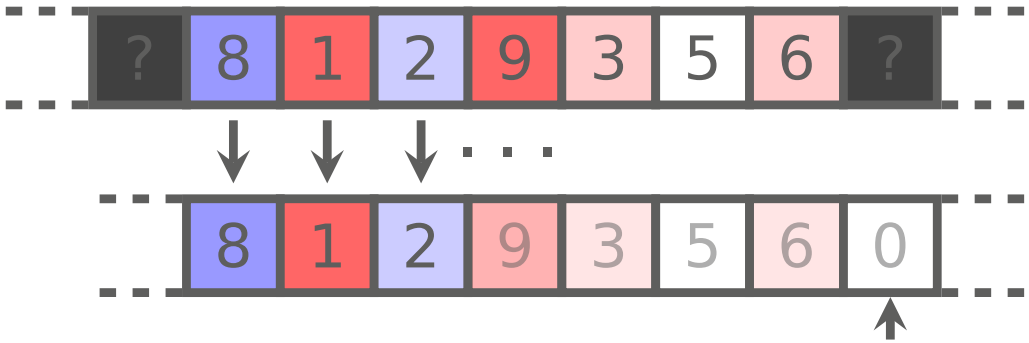
Tableaux et listes chaînées

Structures de données linéaires

Tableaux

Accès aux éléments : avec des indices en temps constant

Ajout et suppression d'éléments : complexité linéaire



Tableaux et listes chaînées

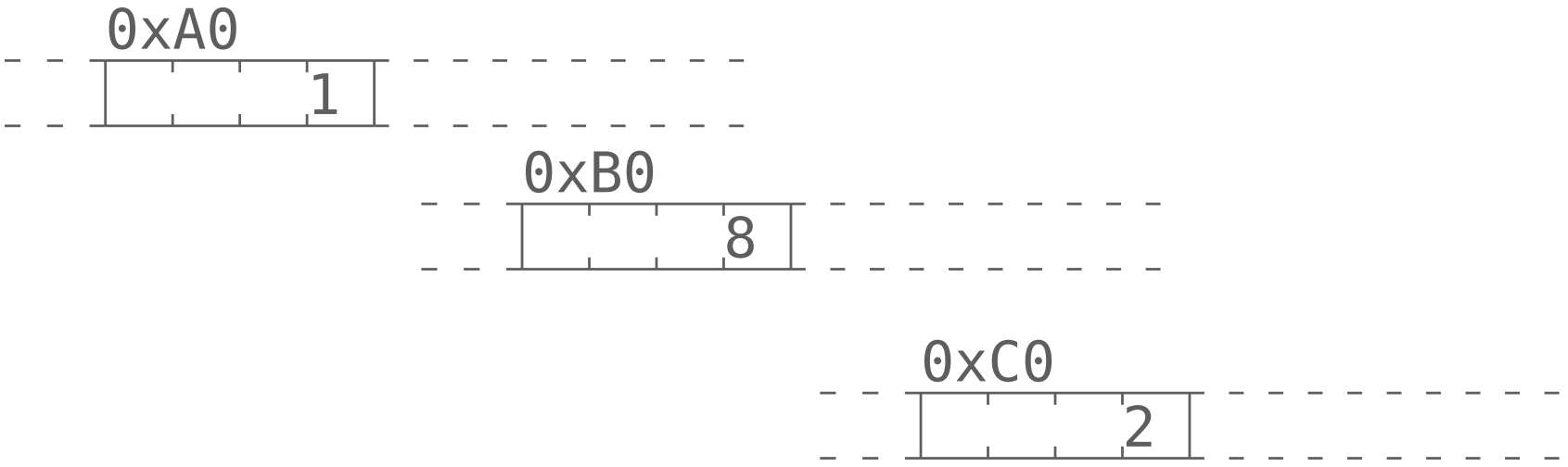
Structures de données linéaires

Tableaux

Accès aux éléments : avec des indices en temps constant

Ajout et suppression d'éléments : complexité linéaire

Listes chaînées



Tableaux et listes chaînées

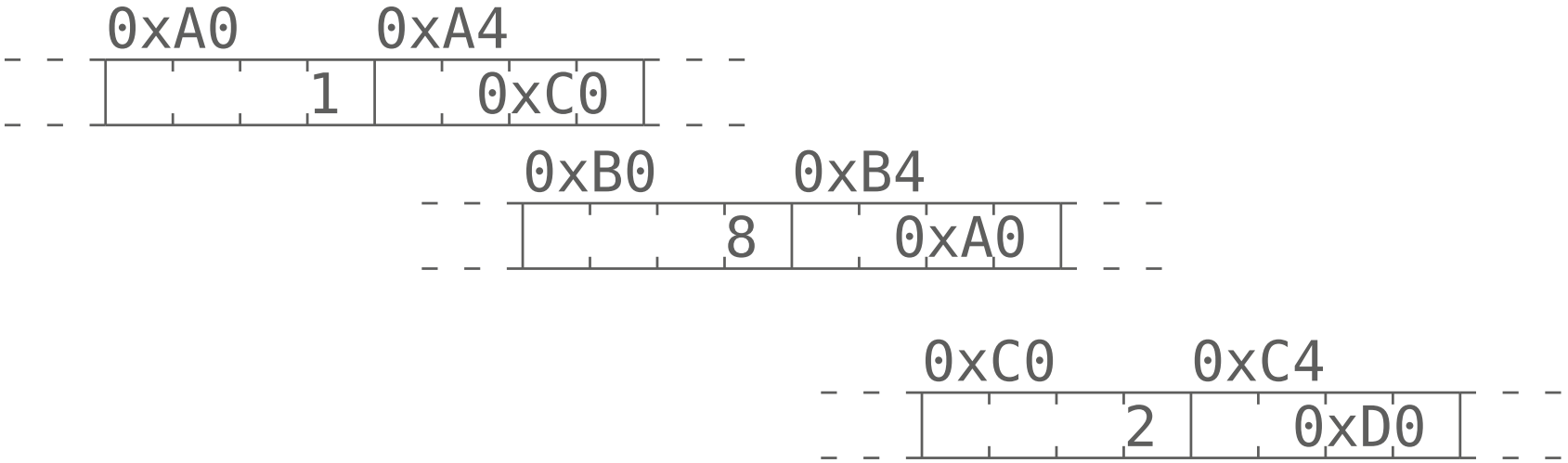
Structures de données linéaires

Tableaux

Accès aux éléments : avec des indices en temps constant

Ajout et suppression d'éléments : complexité linéaire

Listes chaînées



Tableaux et listes chaînées

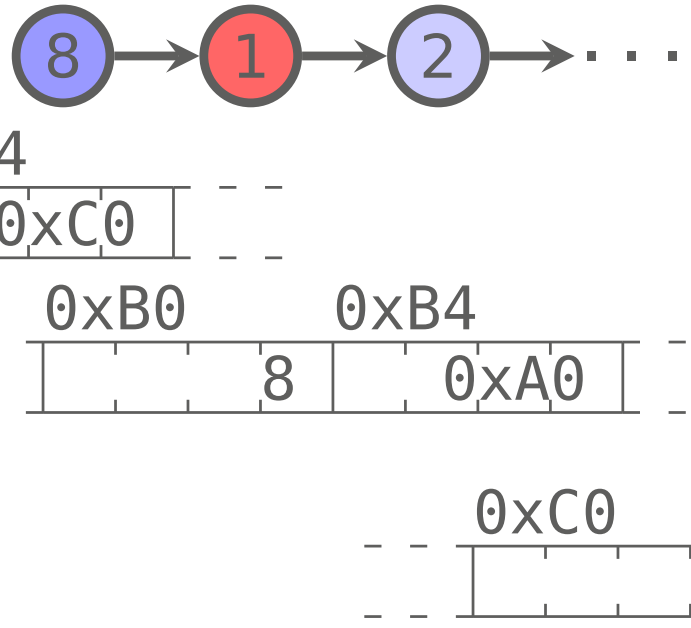
Structures de données linéaires

Tableaux

Accès aux éléments : avec des indices en temps constant

Ajout et suppression d'éléments : complexité linéaire

Listes chaînées



Tableaux et listes chaînées

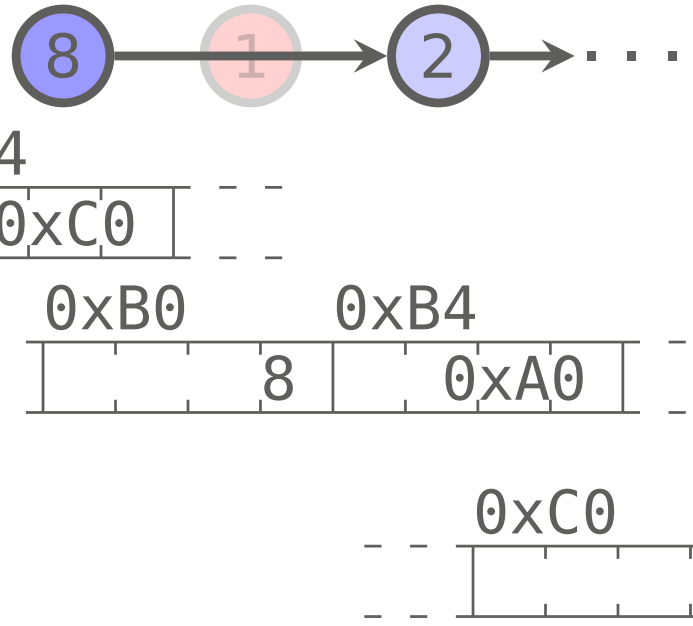
Structures de données linéaires

Tableaux

Accès aux éléments : avec des indices en temps constant

Ajout et suppression d'éléments : complexité linéaire

Listes chaînées



Tableaux et listes chaînées

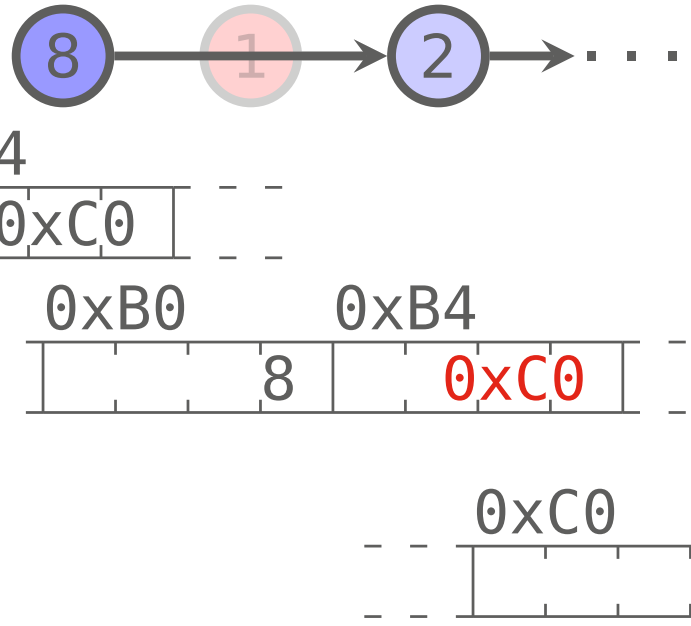
Structures de données linéaires

Tableaux

Accès aux éléments : avec des indices en temps constant

Ajout et suppression d'éléments : complexité linéaire

Listes chaînées



Tableaux et listes chaînées

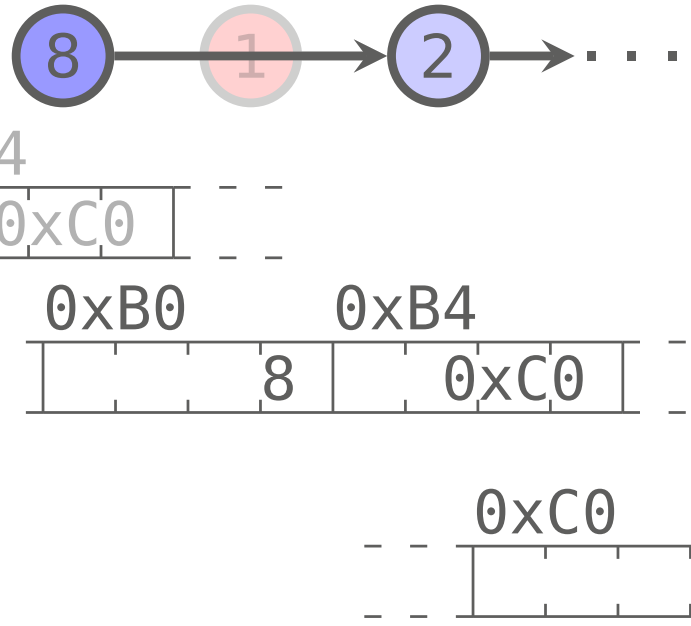
Structures de données linéaires

Tableaux

Accès aux éléments : avec des indices en temps constant

Ajout et suppression d'éléments : complexité linéaire

Listes chaînées



Tableaux et listes chaînées

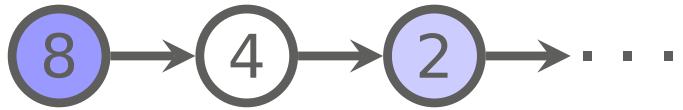
Structures de données linéaires

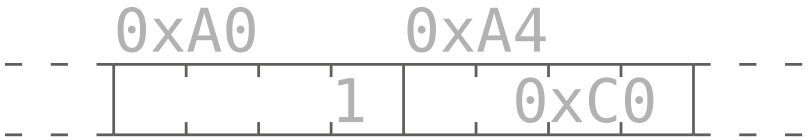
Tableaux

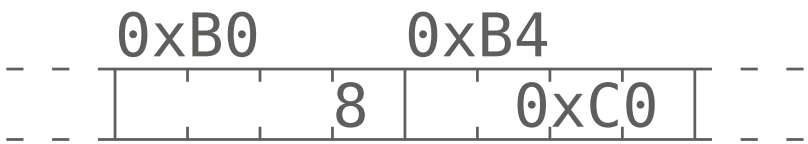
Accès aux éléments : avec des indices en temps constant

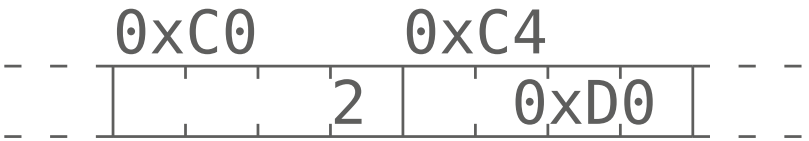
Ajout et suppression d'éléments : complexité linéaire

Listes chaînées









Tableaux et listes chaînées

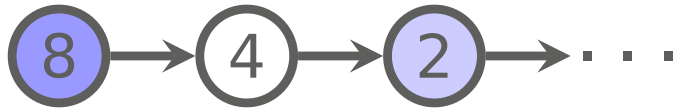
Structures de données linéaires

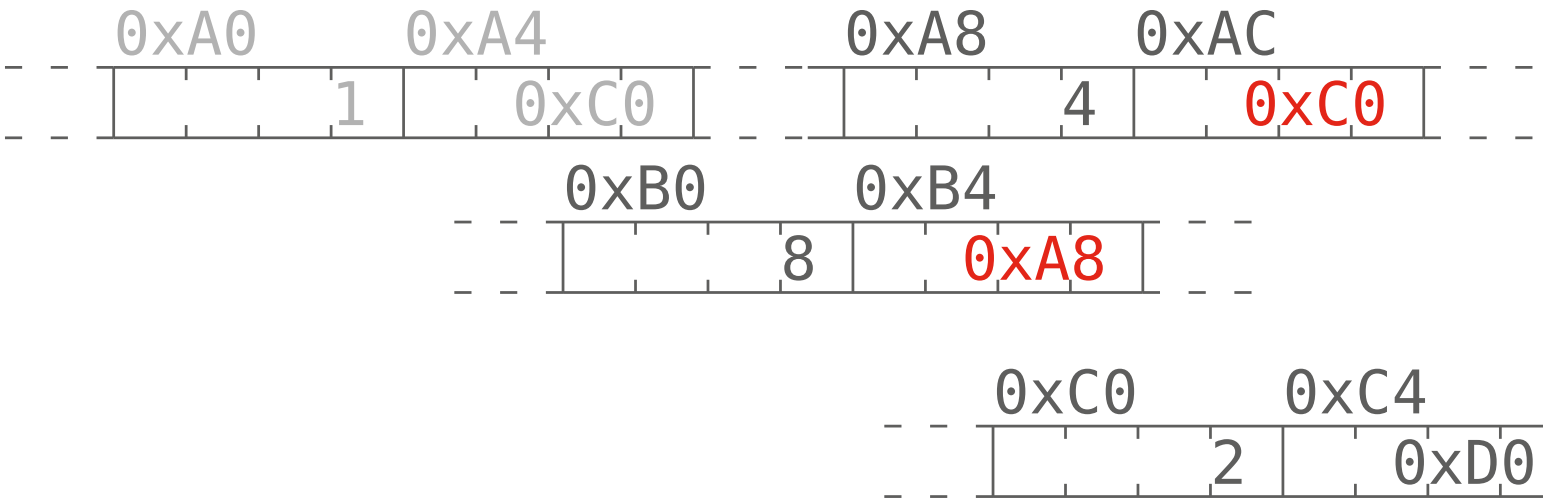
Tableaux

Accès aux éléments : avec des indices en temps constant

Ajout et suppression d'éléments : complexité linéaire

Listes chaînées





Tableaux et listes chaînées

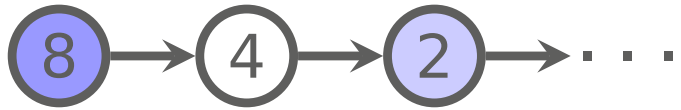
Structures de données linéaires

Tableaux

Accès aux éléments : avec des indices en temps constant

Ajout et suppression d'éléments : complexité linéaire

Listes chaînées



Accès aux éléments : en parcourant

Ajout et suppression d'éléments :

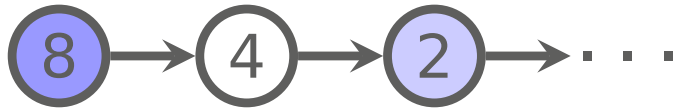
Tableaux et listes chaînées

Structures de données linéaires

Tableaux 

Accès aux éléments : avec des indices en temps constant

Ajout et suppression d'éléments : complexité linéaire

Listes chaînées 

Accès aux éléments : en parcourant complexité linéaire

Ajout et suppression d'éléments :

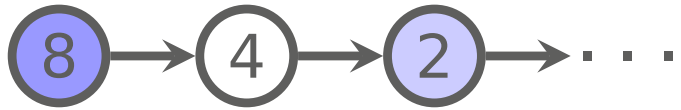
Tableaux et listes chaînées

Structures de données linéaires

Tableaux 

Accès aux éléments : avec des indices en temps constant

Ajout et suppression d'éléments : complexité linéaire

Listes chaînées 

Accès aux éléments : en parcourant complexité linéaire

Ajout et suppression d'éléments : complexité constante

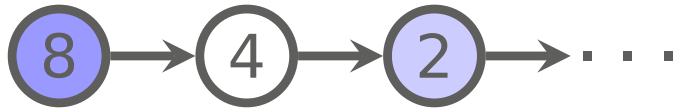
Tableaux et listes chaînées

Structures de données linéaires

Tableaux 

Accès aux éléments : avec des indices en temps constant

Ajout et suppression d'éléments : complexité linéaire

Listes chaînées 

Accès aux éléments : en parcourant complexité linéaire

Ajout et suppression d'éléments : complexité constante

Les tableaux sont dits statiques

Les listes chaînées sont dites dynamiques

Représentation et manipulation

Dans notre pseudo-langage

Information immédiatement disponible : la tête

Représentation et manipulation

Dans notre pseudo-langage

Information immédiatement disponible : la tête

Le reste est aussi une liste chaînée : la queue

Représentation et manipulation

Dans notre pseudo-langage

Information immédiatement disponible : la tête

Le reste est aussi une liste chaînée : la queue

Construction composite (*tête*, *queue*)

Représentation et manipulation

Dans notre pseudo-langage

Information immédiatement disponible : la tête

Le reste est aussi une liste chaînée : la queue

Construction composite (*tête, queue*) *récursive*

Représentation et manipulation

Dans notre pseudo-langage

Information immédiatement disponible : la tête

Le reste est aussi une liste chaînée : la queue

Construction composite (*tête*, *queue*) *récursive*

On dira aussi une cellule

Représentation et manipulation

Dans notre pseudo-langage

Information immédiatement disponible : la tête

Le reste est aussi une liste chaînée : la queue

Construction composite (*tête*, *queue*) *récursive*

On dira aussi une cellule

La dernière cellule n'a pas de queue : liste vide ∅

Représentation et manipulation

Dans notre pseudo-langage

Création manuelle

`l ← (8, (1, (2, ∅)))`

Représentation et manipulation

Dans notre pseudo-langage

Création manuelle

`l ← (8, (1, (2, ∅)))`

`l ← (2, ∅), l ← (1, l), l ← (8, l)`

Représentation et manipulation

Dans notre pseudo-langage

Création manuelle

`l ← (8, (1, (2, ∅)))`

`l ← (2, ∅), l ← (1, l), l ← (8, l)`

Conversion d'un tableau en liste chaînée

Représentation et manipulation

Dans notre pseudo-langage

Création manuelle

```
l ← (8, (1, (2, ∅)))
```

```
l ← (2, ∅), l ← (1, l), l ← (8, l)
```

Conversion d'un tableau en liste chaînée

```
Algorithme tableau_vers_liste : t → l selon
    l ← ∅, i ← ncomp(t) # on commence par la fin
    Tant que i ⩾ 1 répéter l ← (t(i), l), i ← i − 1 .
.
```

Représentation et manipulation

Dans notre pseudo-langage

Décomposition en éléments constitutifs

```
l ← (8, (1, (2, ∅)))

(t, q) ← l          # t vaut 8, q vaut (1, (2, ∅))
```

Représentation et manipulation

Dans notre pseudo-langage

Décomposition en éléments constitutifs

```
l ← (8, (1, (2, ∅)))
(t, q) ← l          # t vaut 8, q vaut (1, (2, ∅))
(t, q) ← réf l      # t et q sont des références
```

Représentation et manipulation

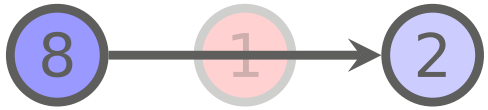
Dans notre pseudo-langage

Décomposition en éléments constitutifs

```
l ← (8, (1, (2, ∅)))
(t, q) ← l          # t vaut 8, q vaut (1, (2, ∅))
(t, q) ← réf l      # t et q sont des références
q ← (2, ∅)          # modifie l
```

Représentation et manipulation

Dans notre pseudo-langage



Décomposition en éléments constitutifs

```
l ← (8, (1, (2, Ø)))
(t, q) ← l          # t vaut 8, q vaut (1, (2, Ø))
(t, q) ← réf l      # t et q sont des références
q ← (2, Ø)          # modifie l
```

Représentation et manipulation

Dans notre pseudo-langage

Accès aux éléments avec des routines

Représentation et manipulation

Dans notre pseudo-langage

Accès aux éléments avec des routines

```
Algorithme tête : réf l → réf t selon
    Si l = ∅ alors exception(‘‘liste vide’’) .
    (t, q) ← réf l
.
```

Représentation et manipulation

Dans notre pseudo-langage

Accès aux éléments avec des routines

```
Algorithme tête : réf l → réf t selon
    Si l = ∅ alors exception(“liste vide”) .
    (t, q) ← réf l
.

Algorithme queue : réf l → réf q selon
    Si l = ∅ alors q ← ∅
    sinon (t, q) ← réf l .
.
```

Représentation et manipulation

Dans notre pseudo-langage

Accès aux éléments avec des routines

```
Algorithme tête : réf l → réf t selon
    Si l = ∅ alors exception(“liste vide”) .
    (t, q) ← réf l
.

Algorithme queue : réf l → réf q selon
    Si l = ∅ alors q ← ∅
    sinon (t, q) ← réf l .
.
```

Accès en *écriture* : on précise **réf** lors de l’appel.

Accès en *lecture seule* : on ne précise rien.

Représentation et manipulation

Dans notre pseudo-langage

Accès aux éléments avec des routines

```
l ← (8, (1, (2, ∅)))
q ← queue(l)          # q vaut (1, (2, ∅))
```

Représentation et manipulation

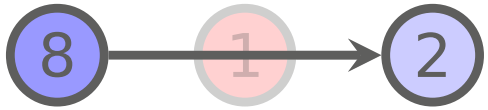
Dans notre pseudo-langage

Accès aux éléments avec des routines

```
l ← (8, (1, (2, Ø)))
q ← queue(l)          # q vaut (1, (2, Ø))
q ← réf queue(l)      # q est une référence
```

Représentation et manipulation

Dans notre pseudo-langage

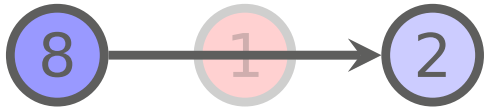


Accès aux éléments avec des routines

```
l ← (8, (1, (2, Ø)))
q ← queue(l)          # q vaut (1, (2, Ø))
q ← réf queue(l)      # q est une référence
q ← (2, Ø)            # modifie l
```

Représentation et manipulation

Dans notre pseudo-langage

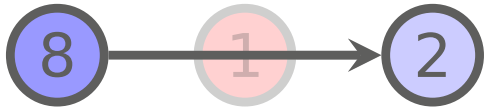


Accès aux éléments avec des routines

```
l ← (8, (1, (2, Ø)))
q ← queue(l)          # q vaut (1, (2, Ø))
q ← réf queue(l)      # q est une référence
q ← (2, Ø)            # modifie l
q ← queue(q)          # formulation alternative
```

Représentation et manipulation

Dans notre pseudo-langage

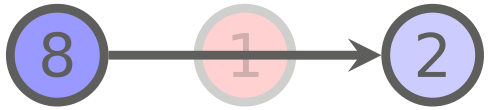


Accès aux éléments avec des routines

```
l ← (8, (1, (2, Ø)))
q ← queue(l)          # q vaut (1, (2, Ø))
q ← réf queue(l)      # q est une référence
q ← (2, Ø)            # modifie l
q ← queue(q)          # formulation alternative
queue(l) ← queue(queue(l)) # sans utiliser q
```

Représentation et manipulation

Dans notre pseudo-langage



Accès aux éléments avec des routines

```
l ← (8, (1, (2, Ø)))
q ← queue(l)          # q vaut (1, (2, Ø))
q ← réf queue(l)      # q est une référence
q ← (2, Ø)            # modifie l
q ← queue(q)          # formulation alternative
queue(l) ← queue(queue(l)) # sans utiliser q
```

Affectation par référence en sortie : **réf** inutile

Représentation et manipulation

Dans notre pseudo-langage

Supprimer ou insérer un élément en tête

```
(8, (1, (2, Ø)))
```

Représentation et manipulation

Dans notre pseudo-langage

Supprimer ou insérer un élément en tête

```
Algorithme supprimer_tête : réf l → () selon
    l ← queue(l)
.
```

```
(8, (1, (2, Ø)))
```

Représentation et manipulation

Dans notre pseudo-langage

Supprimer ou insérer un élément en tête

```
Algorithme supprimer_tête : réf l → () selon
    l ← queue(l)
.
```

```
(8, (1, (2, Ø)))
```

Représentation et manipulation

Dans notre pseudo-langage

Supprimer ou insérer un élément en tête

```
Algorithme supprimer_tête : réf l → () selon
    l ← queue(l)
.

Algorithme ajouter_tête : (réf l, x) → () selon
    l ← (x, l)
.


(8, (1, (2, Ø)))
```

Représentation et manipulation

Dans notre pseudo-langage

Supprimer ou insérer un élément en tête

```
Algorithme supprimer_tête : réf l → () selon
    l ← queue(l)
.
Algorithme ajouter_tête : (réf l, x) → () selon
    l ← (x, l)
.


(8, (1, (2, Ø)))
```

Représentation et manipulation

Dans notre pseudo-langage

Recherche d’un élément avec accès en écriture

Représentation et manipulation

Dans notre pseudo-langage

Recherche d'un élément avec accès en écriture

```
Algorithme rechercher_cellule :
                                (réf l, x) → réf c selon
    c ← réf l
    Tant que c ≠ Ø et tête(c) ≠ x répéter
        c ← réf queue(c)
    .
.
```

Représentation et manipulation

Dans notre pseudo-langage

Recherche d'un élément avec accès en écriture

```
Algorithme rechercher_cellule :
                                (réf l, x) → réf c selon
    c ← réf l
    Tant que c ≠ Ø et tête(c) ≠ x répéter
        c ← réf queue(c)
    .
.
(8, (1, (2, Ø)))
```

Représentation et manipulation

Dans notre pseudo-langage

Recherche d'un élément avec accès en écriture

```
Algorithme rechercher_cellule :
                              (réf l, x) → réf c selon
    c ← réf l
    Tant que c ≠ Ø et tête(c) ≠ x répéter
        c ← réf queue(c)
    .
.
(8, (1, (2, Ø)))
```

Représentation et manipulation

Dans notre pseudo-langage

Recherche d'un élément avec accès en écriture

```
Algorithme rechercher_cellule :
                              (réf l, x) → réf c selon
    c ← réf l
    Tant que c ≠ Ø et tête(c) ≠ x répéter
        c ← réf queue(c)
    .
.
(8, (1, (2, Ø)))
```

Représentation et manipulation

Dans notre pseudo-langage

Recherche d'un élément avec accès en écriture

```
Algorithme rechercher_cellule :
                              (réf l, x) → réf c selon
    c ← réf l
    Tant que c ≠ Ø et tête(c) ≠ x répéter
        c ← réf queue(c)
    .
.
(8, (1, (2, Ø)))
```