2 octobre 2018
40 minutes.
Tous documents interdits.
Une feuille A4 R/V manuscrite autorisée.

Nom : Général
Prénom : Bolle

Groupe de TD : A - 1

Exercice 1 (Permutations d'entiers)

Dans cet exercice, on s'intéresse à des permutations d'entiers. On veut stocker tous les entiers de 1 à n dans un ordre aléatoire dans une séquence S sous forme de **tableau avec longueur explicite**.
On propose l'algorithme bas-niveau ci-dessous, qui part d'une séquence vide et à chaque étape insère un nombre (l'entier courant i) à une position aléatoire p (par échange avec le dernier élément) dans la séquence. On suppose $n \leqslant$ `LMAX`.

$S \leftarrow$ nouvelle Séquence
pour chaque i de 1 à n **faire**
  $p \leftarrow$ aléatoire(0, S.longueur)
  S.longueur $\leftarrow$ A: S.longueur + 1
  S.tab[B: S.longueur - 1] $\leftarrow$ S.tab[p]
  S.tab[p] $\leftarrow$ C: i
retourner S

Consigne : compléter les trous de l'algorithme :
A : S.longueur + 1
B : S.longueur - 1
C : i

Exercice 2 (Analyse d'algorithme)

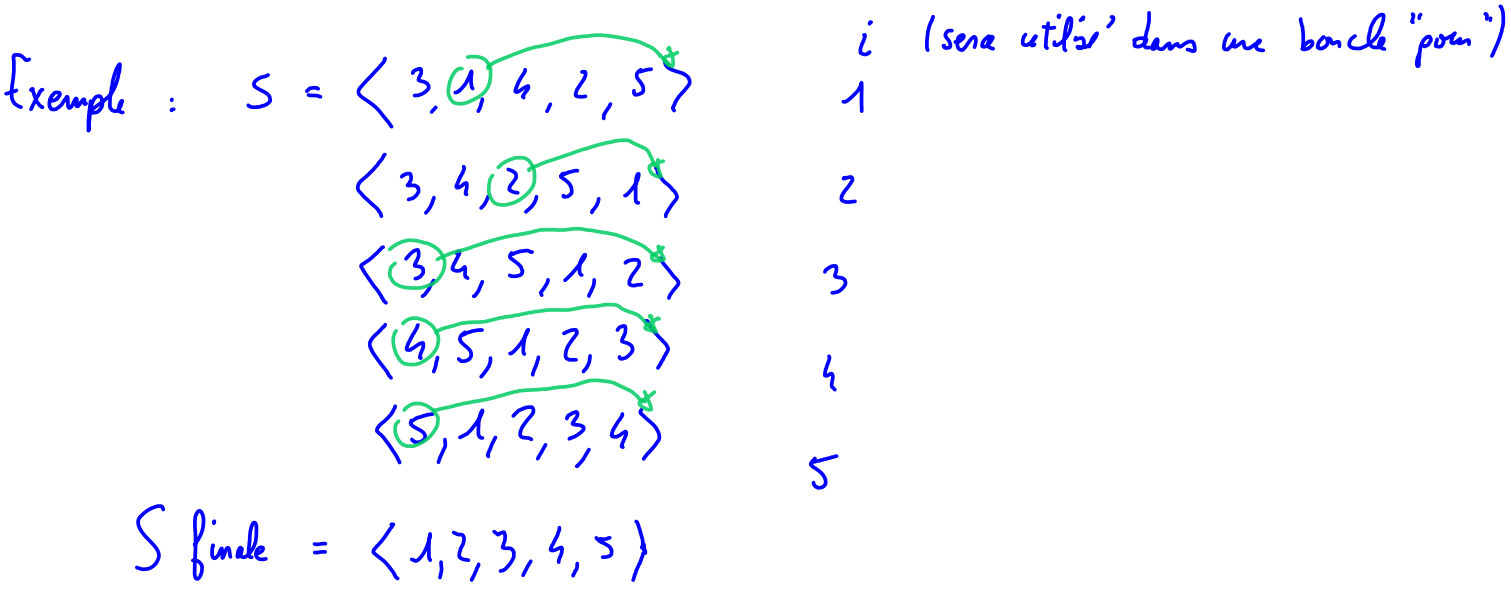
Dans cet exercice, on part de la séquence S aléatoire de l'exercice précédent. On suppose donc que tous les entiers de 1 à n sont stockés dans un ordre aléatoire dans S.

On considère l'algorithme décrit informellement ci-dessous :

1. on commence par chercher l'entier 1 dans S ;
2. on le déplace à la fin de S ;
3. on recommence avec l'entier 2, et ainsi de suite jusqu'à n.

Consignes

1. Commencez par montrer sur un exemple le comportement de l'algorithme en exhibant sa trace d'exécution (montrer l'évolution de la séquence, des variables, etc.).
2. *Question de cours :* donnez les définitions des structures de données qui seront nécessaires.
3. Donnez l'algorithme en pseudo-code. **Attention** : il est conseillé de donner l'algorithme en haut-niveau et de détailler les opérations bas-niveau dans des fonctions séparées.
4. Faites une analyse de complexité de votre algorithme.
5. *Bonus :* commentez rapidement l'effet de cet algorithme sur la séquence.



constante LMAX : entier

Structure de données nécessaire : type Séquence : { tab : tableau de LMAX entiers
longueur : entier }

Algo principal : (S définie par l'algorithme de l'exercice 1)

```
pour i de 1 à n :                                  × n
    p ← cherche_position(S, i)                     O(n)   } O(2n) = O(n)   } O(n²)
    déplace_pos_fin(S, p)                          O(n)   }                }
```

complexité : avec n la longueur de la séquence

voir ci-dessous

Fonctions bas-niveau :

```
cherche_position(S, x) : entier
    pour i de 0 à S.longueur - 1                   × n   } O(1)   } O(n)
        si S.tab[i] = x                                  }        }
            retourner i
    erreur "x non trouvé dans la séquence"
```

on s'arrête dès qu'on a trouvé

dans le pire cas (x dernier élément)

```
deplace_pos_fin(S, p)
    si p ≥ S.longueur ou p < 0                     erreur "position incorrecte"   } O(1)
    elem ← S.tab[p]                                                               }
    pour i de p à S.longueur - 2                   × n   } O(1)   } O(n)          }
        S.tab[i] ← S.tab[i+1]                              }        }             }
    S.tab[S.longueur - 1] ← elem                   O(1)                          }
```

on sauve la valeur

on décale tous les éléments vers la gauche pour conserver l'ordre

on met la valeur à la fin

Note : fonctionne même avec p en dernière position

$O(n) + O(1) = O(n)$ dans le pire cas ($p = 0$)

Commentaire : cet algorithme va trier la séquence car à l'étape i, les i derniers valeurs de la séquence seront 1 à i dans l'ordre croissant : d'abord uniquement le 1, puis on ajoute 2, puis 3 ...
A l'étape n, tous les entiers seront dans l'ordre croissant.

2 octobre 2018
40 minutes.
Tous documents interdits.
Une feuille A4 R/V manuscrite autorisée.

Nom : ..Général.....

Prénom : .....Balle.....

Groupe de TD
A - 1

Exercice 1 (Permutations d'entiers)

Dans cet exercice, on s'intéresse à des permutations d'entiers. On veut stocker tous les entiers de 1 à n dans un ordre aléatoire dans une séquence S sous forme de **tableau avec longueur explicite**.
On propose l'algorithme bas-niveau ci-dessous, qui part d'une séquence vide et à chaque étape insère un nombre (l'entier courant i) à une position aléatoire p (par échange avec le dernier élément) dans la séquence. On suppose $n \leqslant$ `LMAX`.

$S \leftarrow$ nouvelle Séquence
pour chaque i de 1 à n **faire**
  $p \leftarrow$ aléatoire(0, S.longueur)
  S.longueur $\leftarrow$ A:
  S.tab[B:] $\leftarrow$ S.tab[p]
  S.tab[p] $\leftarrow$ C:
retourner S

Consigne : compléter les trous de l'algorithme :

A : S.longueur + 1

B : S.longueur - 1

C : i

Exercice 2 (Analyse d'algorithme)

Dans cet exercice, on part de la séquence S aléatoire de l'exercice précédent. On suppose donc que tous les entiers de 1 à n sont stockés dans un ordre aléatoire dans S.

On considère l'algorithme décrit informellement ci-dessous :

1. on commence par chercher l'entier 1 dans S ;
2. on le décale de 1 position vers la droite dans S ;
3. on recommence avec l'entier 2 que l'on décale de 2 positions ;
4. on procède de même jusqu'à n ; si le décalage ferait « sortir » l'entier de la séquence, on le place en fin de la séquence.

Consignes

1. Commencez par montrer sur un exemple le comportement de l'algorithme en exhibant sa trace d'exécution (montrer l'évolution de la séquence, des variables, etc.).
2. *Question de cours :* donnez les définitions des structures de données qui seront nécessaires.
3. Donnez l'algorithme en pseudo-code. **Attention** : il est conseillé de donner l'algorithme en haut-niveau et de détailler les opérations bas-niveau dans des fonctions séparées.
4. Faites une analyse de complexité de votre algorithme.
5. *Bonus :* commentez rapidement l'effet de cet algorithme sur la séquence.

Exemple : $S = \langle 3, 1, 4, 2, 5 \rangle$ i (sera utilisé dans une boucle "pour")

$\langle 3, 1, 4, 2, 5 \rangle$ 1

$\langle 3, 4, 1, 2, 5 \rangle$ 2

$\langle 3, 4, 1, 5, 2 \rangle$ 3

$\langle 4, 1, 5, 3, 2 \rangle$ 4

$\langle 1, 3, 2, 5, 4 \rangle$ 5

$\langle 1, 3, 2, 4, 5 \rangle$

constante LMAX : entier

Structure de données nécessaire : type Séquence : { tab : tableau de LMAX entiers
longueur : entier }

Algo principal : (S définie par l'algorithme de l'exercice 1)

```
pour i de 1 à n
    p ← cherche_position (S)      O(n)   } O(2n) = O(n)      × n  } O(n²)
    deplace_pos (S,p)             O(n)   }
```

cf. ci dessous

fonctions bas niveau :

complexité, avec n la longueur de la séquence

cherche_poition : cf. correction page 2

```
deplace_pos(S, p)
    si p > S.longueur                          } O(1)   alors erreur "position incorrecte"
       ou p < 0                                }
    elem ← S.tab[p]                            O(1)
    limite = min (p+elem, S.longueur - 1)      O(1)
    pour i de p à limite - 1                   × n  } O(n)
        S.tab[i] ← S.tab[i+1]                  O(1) }
    S.tab[limite] ← elem                       O(1)
```

O(n) + O(1) = O(n)

sauvegarde de la valeur

calcul nouvelle position, sans dépasser

déplacement des valeurs entre ancienne et nouvelle position

valeur à la position voulue

Note : fonctionne même si p dernière position

Commentaire : cet algorithme trie "presque" la séquence S. On observe que les "grands" valeurs se retrouvent triés en fin de séquence, mais qu'il peut y avoir des inversions sur les petites valeurs (cf exemple).

Note pour information (ce n'était pas attendu de votre part) : on peut prouver que l'algorithme trie la séquence ssi à l'étape i, et soit p la position de i dans S, toutes les valeurs entre 1 et i-1 sont à une position ≤ p+i.
on peut alors facilement créer des exemples où ça ne fonctionne pas : < ... 3 ..... 1 ... >
au moins 3 éléments (sans le "2")
ou 4 éléments (avec le "2")