14 novembre 2018
40 minutes.
Tous documents interdits.
Une feuille A4 R/V manuscrite autorisée.

Nom : .....

Prénom : .....

Groupe de TD

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 **listes chaînées**.
On a l'algorithme haut-niveau ci dessous à gauche, qui insère des cellules à des positions aléatoires. On s'intéresse à l'implantation bas-niveau ci-dessous à droite de la fonction `insere_position`.

```
S ← nouvelle Séquence
pour i de 1 à n faire
    p ← aléatoire(0, i − 1)
    insere_position(S, p, i)
retourner S
```

```
insere_position(S, p, val)
    ncel ← nouvelle Cellule
    ncel.valeur ← val
    A:
    cel ← S.tête
    i ← 0
    pour i de 1 à p − 1 faire
        B:
    ncel.suivant ← C:
    cel.suivant ← D:
```

Consigne : compléter les trous de l'algorithme :

A : (cas particulier) si p = 0 ncel.suivant ← S.tete ; S.tete ← ncel ; retourner

B : cel ← cel.suivant

C : cel.suivant

D : ncel

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.

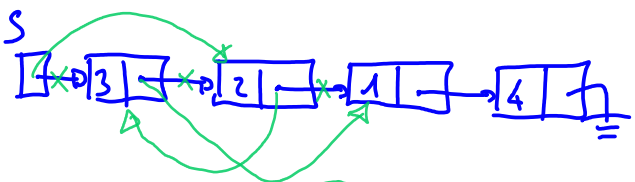
On considère l'algorithme décrit informellement ci-dessous :

1. on commence par chercher l'entier 1 dans S ;
2. on le déplace à la **fin** de S ;
3. on cherche l'entier 2, que l'on déplace au **début** de S ;
4. on recommence avec l'entier 3, et ainsi de suite jusqu'à n (les impairs à la fin, les pairs au début).

Consignes Vous devez procéder uniquement par **modification des liens de chaînage**.

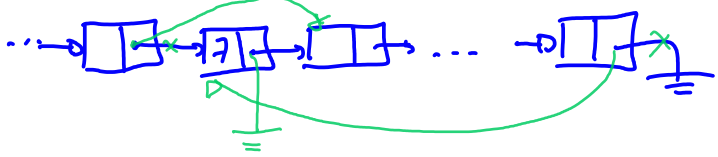
1. Commencez par montrer sur un ou deux exemples de déplacement le comportement de l'algorithme (dessinez la liste chaînée et les modifications effectuées).
2. Donnez l'algorithme en pseudo-code. **Attention** : il est demandé de donner l'algorithme en haut-niveau et de détailler les opérations bas-niveau dans des fonctions séparées.
3. Faites une analyse de complexité de votre algorithme.
4. *Bonus :* commentez rapidement l'effet de cet algorithme sur la séquence.

Exemple pour valeur 2 (paire)

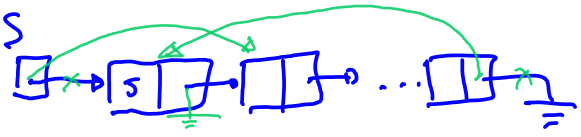


Séquence
⟨ 3 2 1 4 ⟩
⇓
⟨ 2 3 1 4 ⟩

Exemple valeur impaire (7)



Exemple impaire en tête (5)



fonction principale decale_tout (S, n)

```
n x | pour i de 1 à n                                  ⇒ O(n²)
    | O(n)   (prec, cel) ← cherche (S, i)
    | +      si i mod 2 = 0
    | O(1)   └ deplace_debut (S, prec, cel)
    | +      sinon deplace_fin (S, prec, cel)
    | O(n)
```

$O(n)$ ⇐ cherche (S, n)

```
O(1) | prec ← Nil
     | cel ← S.tete
n x  | tant que cel ≠ Nil
O(1) |   si cel.valeur = n
     |   └ retourner (prec, cel)
     |   prec ← cel
     |   cel ← cel.suivant
O(1) | retourner (Nil, Nil)
```

$O(1)$ ⇐ deplace_debut (S, prec, cel)

```
O(1) | si prec = Nil retourner   (déjà au début)
     | prec.suivant ← cel.suivant
     | cel.suivant ← S.tete
     | S.tete ← cel
```

(Note : pour deplace_fin, il serait plus intéressant de garder une référence vers la queue dans la structure Séquence pour avoir une complexité en $O(n)$)

$O(n)$ ⇐ deplace_fin (S, prec, cel)

```
O(1) | si cel.suivant = Nil retourner   (déjà à la fin)
     | queue ← cel.suivant
n x  | tant que queue.suivant ≠ Nil
O(1) | └ queue ← queue.suivant
     |
     | si prec = Nil
     |   S.tete ← cel.suivant
O(1) | sinon
     |   prec.suivant ← cel.suivant
     |
     | queue.suivant ← cel
     | cel.suivant ← Nil
```

Commentaire : après exécution de l'algorithme, S est composée de : d'abord les pairs entre 2 et n par ordre décroissant, puis les impairs par ordre croissant.

Algorithmique & Prog. Impérative **Quick test: Ensembles, Séquences et Listes chaînées**

14 novembre 2018
40 minutes.
Tous documents interdits.
Une feuille A4 R/V manuscrite autorisée.

Nom : .....

Prénom : .....

Groupe de TD

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 **listes chaînées**.
On a l'algorithme haut-niveau ci dessous à gauche, qui insère des cellules à des positions aléatoires. On s'intéresse à l'implantation bas-niveau ci-dessous à droite de la fonction `insere_position`.

```
S ← nouvelle Séquence
pour i de 1 à n faire
    p ← aléatoire(0, i − 1)
    insere_position(S, p, i)
retourner S
```

```
insere_position(S, p, val)
    ncel ← nouvelle Cellule
    ncel.valeur ← val
    A:
    cel ← S.tête
    i ← 0
    pour i de 1 à p − 1 faire
        B:
    ncel.suivant ← C:
    cel.suivant ← D:
```

Consigne : compléter les trous de l'algorithme :

A : (cas particulier)

B : Voir page 1

C :

D :

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.

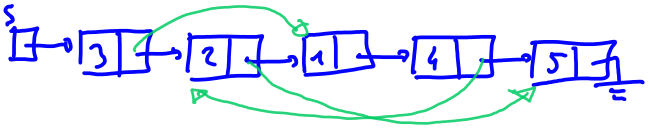
On considère l'algorithme décrit informellement ci-dessous :

1. on commence par chercher l'entier 1 dans S ;
2. on le décale de 1 position plus loin 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 fait « sortir » l'entier de la séquence, on le place en queue de la séquence.

Consignes Vous devez procéder uniquement par **modification des liens de chaînage**.

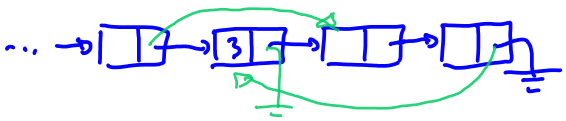
1. Commencez par montrer sur un ou deux exemples de déplacement le comportement de l'algorithme (dessinez la liste chaînée et les modifications effectuées).
2. Donnez l'algorithme en pseudo-code. **Attention** : il est demandé de donner l'algorithme en haut-niveau et de détailler les opérations bas-niveau dans des fonctions séparées.
3. Faites une analyse de complexité de votre algorithme.
4. *Bonus :* commentez rapidement l'effet de cet algorithme sur la séquence.

Exemple pour 2 :

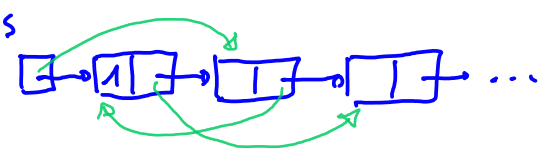


Séquence
⟨ 3 2 1 4 5 ⟩
⇓
⟨ 3 1 4 2 5 ⟩

Exemple 3 en queue :



Exemple modification tete :



fonction principale : décale_tout (S, n) $\Rightarrow O(n^2)$

```
n x   pour i de 1 à n
O(n)    (prec, cel) ← cherche (S, i)
O(n)    decale (S, prec, cel)
```

cherche : voir page 2

$O(n) \Leftarrow$ décale (S, prec, cel)

```
si cel.suivant = Nil alors retourner (déjà à la fin)
(on commence par enlever cel de la liste)
O(1) { si prec = Nil (valeur en tête)
         S.tete ← cel.suivant
       sinon
         prec.suivant ← cel.suivant
(on cherche l'endroit où insérer : après "pos")
O(1) pos ← cel.suivant
n x  pour i de 2 à cel.valeur
O(1)   si pos.suivant = Nil
         sortir de la boucle
       sinon
         pos ← pos.suivant
(on réinsère cel après pos)
O(1) { cel.suivant ← pos.suivant
       pos.suivant ← cel
```

Commentaire : cet algorithme va presque trier la séquence, mais des inversions peuvent avoir lieu, par exemple si quand on décale 3, il est plus de 3 positions avant 2 : $\langle 3\,4\,5\,1\,2 \rangle \Rightarrow \langle 4\,5\,1\,3\,2 \rangle$

Et quelle que soit la suite, 3 restera devant 2.