Nom : .....

Tri par insertion dans une liste

On considère la liste

6	3	1	7	5	9	8	2	4

à trier dans l'ordre croissant. Pour cela on traite la liste comme une pile (c'est-à-dire qu'on n'utilise que la méthode `pop(self)` sur cette liste). On utilise l'algorithme suivant :

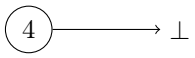
- Tant que la liste n'est pas vide, faire
  - dépiler le dernier élément de la liste à trier ;
  - insérer cet élément dans la liste triée, de telle manière que celle-ci reste triée.
- La liste obtenue alors est triée (c'est un invariant de l'algorithme).

.

L'objet de cet exercice est d'étudier la notion de liste triée et de ses diverses implémentations.

1 Exemple

La liste vide est notée $\perp$. Elle est considérée comme une liste triée, et la liste ne contenant que 4 est représentée ainsi :



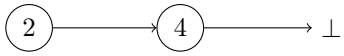
Allen Newell [1] (1927-1992) propose d'appeler `item` le graphe orienté ci-dessus, et de modéliser une liste par un item. Plus précisément

- $\perp$ est une liste (appelée `empty` ou vide).
- Toute autre liste est un couple (ou tuple de longueur 2) de la forme `(element,item)`

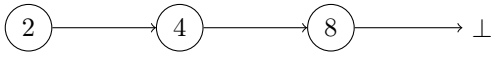
L'item ci-dessus est donc le couple `(4,`$\perp$`)`. C'est la fin de la liste puisque son deuxième élément est $\perp$.

Comme 2 est plus petit que 4, on insère 2 avant 4 (en tête de liste), en créant un nouvel item dont

- l'`element` est 2,
- l'`item` est la liste précédente `(4,`$\perp$`)`.

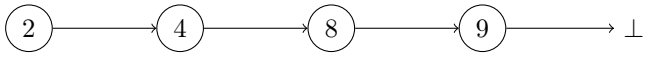


Comme 8 est plus grand que 4, on l'insère à la fin de la liste, ce qui revient à remplacer $\perp$ par l'item `(8,`$\perp$`)` :

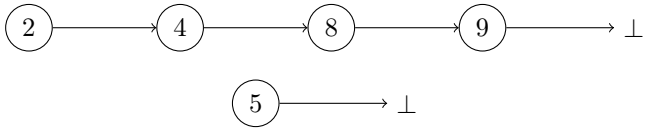


De même, 9 étant plus grand que 8, est placé à la fin, en remplaçant $\perp$ par `(9,`$\perp$`)` :

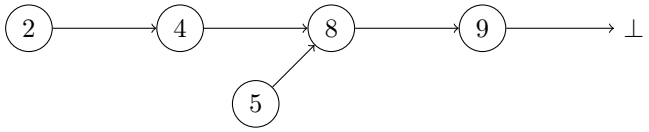
1. dans le rapport RAND titré `Programming the Logic Theory machine` et datant du 28 février 1957, coécrit avec Shaw. Newell a inventé les listes pour pouvoir programmer récursivement.



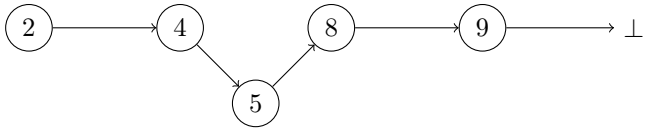
Mais 5 doit être placé entre 4 et 8, ce qui revient à effectuer une insertion dans la liste :



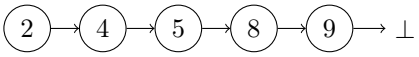
8 est après 5 donc on remplace l'item de 5 (⊥) par 8 :



5 est après 4 donc on remplace l'item de 4 (actuellement 8) par l'item 5 :

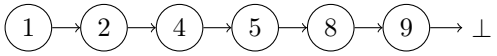


1. La liste est alors celle-ci :

   Elle se représente également par `(2,(4,(5,(8,(9,⊥)))))`. Écrire de la même manière l'état suivant de la liste, après l'insertion de 1 : .....

   On pourra s'aider du dessin que voici :

2. Dessiner la liste après insertion du 7 :

3. Dessiner la liste après l'insertion du 1 :

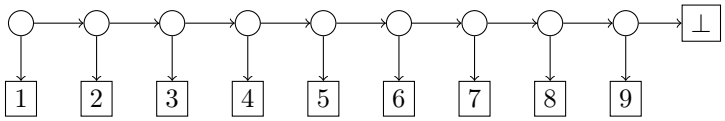
4. Combien de comparaisons ont-elles été nécessaires pour créer la liste triée à partir de la liste [6,3,1,7,5,9,8,2,4] ? .....

2 Architecture et systèmes d'exploitation

L'ordinateur sur lequel travaillait l'équipe de Newell s'appelait `Johnniac`[2].

1. Chaque instruction du Johnniac était un mot de 40 bits.
   Combien d'octets cela fait-il ? .....
2. La mémoire rapide (équivalente du cache du CPU dans un SoC) du Johnniac occupait 4096 mots de 40 bits. Donner la taille de cette mémoire en octets, à un kilooctet près : .....
3. La mémoire secondaire (équivalente au disque dur) du Johnniac occupait 9216 mots de 40 bits. Donner la taille de cette mémoire en octets, à un kilooctet près : .....
4. Le CPU du Johnniac effectuait 15000 opérations par seconde. Si on reprogrammait le solveur de Newell sur un ESP32 permettant 160 millions d'opérations par seconde, à quelle durée se ramèneraient les 20 heures machines du Johnniac utilisées par Newell pour démontrer un théorème ? .....
5. La taille du système d'exploitation du Johnniac est estimée à 3600 mots de 40 bits. Donner cette taille en octets, arrondie au kilooctet le plus proche : .....

Dans le Johnniac, Newell implémente la liste triée [1,2,3,4,5,6,7,8,9] par un arbre binaire dont les éléments sont les feuilles et les items sont des nœuds non étiquetés :

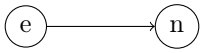


Cet arbre binaire est-il un arbre binaire de recherche ? Justifier :

2. en hommage à John Von Neumann (1903-1957)

3 Listes et tuples

Dans cette partie, on représente l'item



par le tuple `(e,n)`.

On représente l'item vide $\perp$ par un tuple vide `()`. Il s'agit d'un tuple de longueur 0. En dehors de ce cas, les items sont donc des tuples de longueur 2. On rappelle que pour accéder au premier attribut du tuple `T` on écrit `T[0]` et pour accéder au second attribut du tuple `T` on écrit `T[1]`.

La création de listes se fait donc avec ces fonctions Python :

```
def empty() :
    return ()
def item(e,L) :
    return (e,L)
```

1. Compléter les fonctions Python suivantes pour qu'elles implémentent le modèle de liste de Newell :

```
def is_empty(L) :
    return ........
def element(L) :
    return L[...]
def next_item(L) :
    return L[...]
```

2. Pour transformer une liste en liste triée, il faut une fonction d'insertion dans la liste. Pour insérer un élément `e` dans une liste `L` on fait ceci :
   - Si `L` est vide, on crée l'item `(e,L)`
   - Si `e` est plus petit que l'élément de `L`, on l'insère en début
   - Sinon on l'insère quelque part après le premier élément de `L`.

   Voici le codage Python de cette fonction qui, à un élément et une liste triée, associe la nouvelle liste triée où l'élément a été inséré :

```
def insert(e,L):
    if is_empty(L) or e<=element(L):
        return item(e,L)
    return item(element(L), insert(e,next_item(L)))
```

   (a) La fonction `insert` ci-dessus est
   - ☐ itérative
   - ☐ récursive

   (b) On voudrait montrer que la proposition « la liste est triée » est un invariant. On admet que la liste vide est triée.

Montrer que, si `e` est plus petit que le premier élément de la liste triée `L` alors la liste obtenue en insérant `e` en tout début de `L` est encore triée :

4 Tuples nommés

Pour ne pas avoir à se rappeler qui, parmi l'`element` et l'`item` vient en premier [3], on propose de les nommer. La syntaxe Python des « classes » le permet :

```
class Item():
        element = None
        next_item = None
```

Alors `element` et `next_item` sont des ..... de l'`Item()`.
Le test de vacuité s'écrit alors

```
def is_empty(L):
        return L.element is None
```

Pour insérer un élément dans la liste triée, on distingue trois cas :

- Si L n'est pas un `Item()` c'est que L est `None`, alors on crée un `Item()`, on l'affecte à L et on y met l'élément à insérer, comme `element`.
- Si L est en fin de liste ou si son `element` est plus grand que celui à insérer, alors
  - On crée une copie de L, nommée `wagons`,
  - on remplace l'`element` de L par celui à insérer,
  - on remplace le `next_item` de L par les `wagons`.
- Sinon, on insère l'élément dans le `next_item` de L.

La fonction `insert(e: integer, L: Item) -> None` est codée ainsi en Python :

3. D'autant que, chez Newell, c'est l'adresse mémoire de l'`item` qui venait en premier, suivi de l'`element` lui-même. Deux ans plus tard, McCarthy nommait `Car` l'`element` et `Cdr` l'`item` suivant.

```
def insert(e,L):
        if L is None:
                L = Item()
                L.element = e
        elif is_empty(L) or e <= L.element:
                wagons = Item()
                wagons.element = L.element
                wagons.next_item = L.next_item
                L.element = e
                L.next_item = wagons
        else:
                insert(e,L.next_item)
```

En SQL, on ajoute un troisième attribut `id` (entier) qui est une clé primaire, avec :

```
CREATE TABLE liste (id integer PRIMARY KEY, element integer, next_id integer)
```

Avec la liste [6,3,1,7,5,9,8,2,4] on obtient, après insertion du 4, cette table :

id	element	next_id
0	4	NULL

Puis après insertion du 2 :

id	element	next_id
0	4	NULL
1	2	0

Puis après insertion du 8 :

id	element	next_id
0	4	2
1	2	0
2	8	NULL

Puis après insertion du 9 :

id	element	next_id
0	4	2
1	2	0
2	8	3
3	9	NULL

Puis après insertion du 5 :

id	element	next_id
0	4	4
1	2	0
2	8	3
3	9	NULL
4	5	2

1. Compléter la table obtenue après insertion du 7 (colonne `next_id`) :

id	element	next_id
0	4	
1	2	
2	8	
3	9	
4	5	
5	7	

2. Compléter la table obtenue après insertion du 1 :

id	element	next_id
0	4	
1	2	
2	8	
3	9	
4	5	
5	7	
6	1	

3. Compléter la table obtenue après insertion du 3 :

id	element	next_id
0	4	
1	2	
2	8	
3	9	
4	5	
5	7	
6	1	
7	3	

4. La table (`liste`) obtenue à partir de la liste [6,3,1,7,5,9,8,2,4] est finalement

id	element	next_id
0	4	4
1	2	7
2	8	3
3	9	NULL
4	5	8
5	7	2
6	1	1
7	3	0
8	6	5

La requête SQL `SELECT element FROM liste WHERE next_id=NULL` renvoie
- ☐ l'élément placé en premier dans la liste
- ☐ l'élément placé en dernier dans la liste
- ☐ le plus petit élément de la liste
- ☐ le plus grand élément de la liste

5. La requête SQL `SELECT next_id FROM liste WHERE element=5` renvoie
   - ☐ l'élément d'identifiant 5 dans la liste
   - ☐ le cinquième élément de la liste
   - ☐ l'emplacement de l'élément qui précède 5 dans la liste
   - ☐ l'emplacement de l'élément qui suit 5 dans la liste

5 Objets

Dans le paradigme de la POO (programmation orientée objet), une liste triée est définie par la classe `Item` de Python mais avec une méthode `__init__` d'initialisation, qui se charge de créer l'objet et d'affecter ses attributs. L'insertion `insert` est maintenant une méthode de l'objet :

```
class Item():
        def __init__(self,element,next_item=None):
          self.element = element
          self.next_item = next_item
        def insert(self,elt):
          if elt<=self.element:
                self.next_item = Item(self.element,self.next_item)
                self.element = elt
          elif self.next_item is None:
                self.next_item = Item(elt)
          else:
                self.next_item.insert(elt)
```

1. La méthode `insert` est
   - ☐ un attribut
   - ☐ un accesseur
   - ☐ un mutateur
   - ☐ une classe
2. Pour insérer un entier `n` dans une liste triée `LT` qui est un objet de la classe `Item` décrite ci-dessus, on écrit
   - ☐ `insert(LT,n)`
   - ☐ `insert(n,LT)`
   - ☐ `LT.insert(n)`
   - ☐ `from LT import n`