DS4info

8 Mars 2025

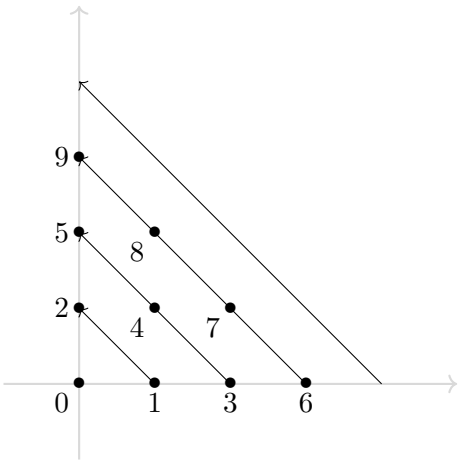
La calculatrice est interdite. L'usage de tout document est interdit. La rigueur, le soin, la présentation seront fortement pris en compte dans la notation. Les résultats de chaque question seront soulignés ou encadrés. Veuillez à bien indenter avec deux grands carreaux (ou quatre petits) par indentation.

On ne comprend pas la récursivité tant qu'on a pas compris la récursivité

1. Expliquer ce que fait la fonction `mystère` :

```
def Mystere(L:list,x):
    if len(L) == 0:
        return 0
    e = L.pop()
    if x == e:
        return Mystere(L,x) + 1
    else:
        return Mystere(L,x)
```

2. Écrire une fonction **récursive** `TriFusion(L)` qui trie la liste de nombres `L` en la coupant en deux parties. On supposera que l'on a disposition une fonction `Fusion(G,D)` (que l'on ne demande pas d'écrire) qui, à partir de deux listes triées `G` et `D`, renvoie la fusion des deux listes triées.
3. Soit $(x, y) \in \mathbb{N}^2$. On souhaite attribuer un numéro à (x, y) suivant le procédé suivant :



Ainsi, le numéro du point $(2, 1)$ est 7. Écrire une fonction **récursive** `numero(x,y)` qui à un couple d'entiers naturels renvoie son numéro.

Ça sert à rien les répétitions, vive l'impro !

1. Écrire une fonction `Presence(L,x)` qui renvoie `True` si `x` est un élément de la liste `L` et `False` sinon.
2. Déterminer sa complexité comme un $\mathcal{O}$ d'une quantité dépendant de n la longueur de la liste.
3. Si `D` est un dictionnaire et `x` un élément, donner la complexité permettant de vérifier si `x` est une clé de `D`.
4. On souhaite déterminer si une liste `L` contient des répétitions, c'est-à-dire qu'elle contient au moins deux fois un même élément. Écrire une fonction `PresenceRépétitions(L)` qui renvoie `True` si `L` contient au moins une répétition et `False` sinon. Pour cela, on comparera `L[i]` et `L[j]` pour tout entiers $0 \leqslant i < j < \text{len}(L)$.
5. Déterminer la complexité de la fonction `PresenceRépétitions`.
6. Écrire la fonction `DictionnaireOccurrence(L:list)` qui renvoie un dictionnaire dont chaque clé est un élément de `L` et dont la valeur associée à une clé est le nombre de fois où cet élément apparaît dans la liste `L`. Cette fonction devra avoir une complexité linaire en la longueur de la liste.
7. Justifier que la complexité est bien linéaire.
8. En utilisant la fonction `DictionnaireOccurrence` écrire une fonction `PresenceRépétitionsBis` qui a une complexité linéaire. On justifiera que l'on a une complexité linéaire.
9. Écrire une fonction `SupprimerDoublon` qui prend en entrée une liste et renvoie une liste en ayant supprimé tous les éléments redondants, sauf le premier élément parmi les répétitions sera conservé. Par exemple, si `[1,5,2,1,1,4,3,2]` la fonction renverra `[1,5,2,4,3]`. La complexité devra être linéaire en la longueur de la liste.