Nom :_____

Numéro de votre place :_____

Code permanent :_____

Directives pédagogiques :

- Inscrivez votre nom, numéro de place et code permanent.
- Sortez votre carte étudiante et mettez la à vue.
- Lisez attentivement toutes les questions et **répondez directement sur le questionnaire**.
- Seule l'utilisation d'un crayon est permise, **aucune documentation, calculatrice, téléphone cellulaire, ordinateur, ou autre objet**.
- Cet examen contient 9 questions pour 165 points au total + 10 points en bonus !
- Le barème est établi à 1 point par minute environ.
- Cet examen contient 20 pages, incluant 3 pages détachables à la fin pour vos brouillons et 3 Appendices.
- Pour les questions à développement, **écrivez lisiblement et détaillez vos réponses**.
- Vous avez 165 minutes pour compléter cet examen.

BONNE CHANCE et JOYEUSES FÊTES !

| | |
|---|---|
| 1 | / 25 |
| 2 | / 20 |
| 3 | / 25 |
| 4 | / 10 |
| 5 | / 25 |
| 6 | / 30 |
| 7 | / 10 |
| 8 | / 10 |
| 9 | / 10 |
| Bonus | / 10 |
| Total | / 165 |

1. (25) Considérez la structure de données `Map` (Appendice A) et une implantation avec une liste qu'on garde non triée (Appendice B).

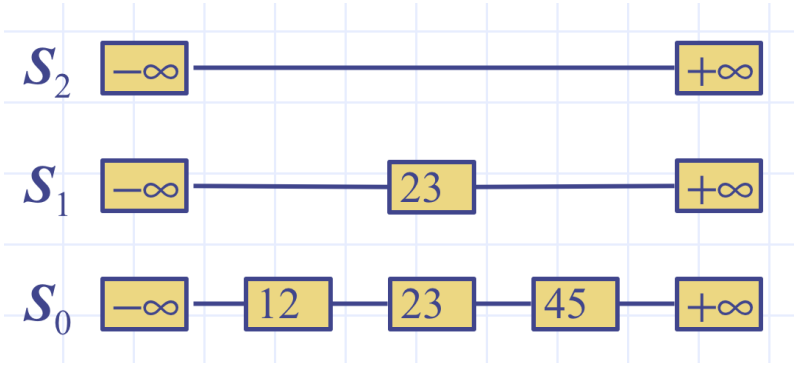
   a) (15) Donnez une implantation de la méthode `items()` directement dans la class `UnsortedListMap` qui roule en O(*n*), pour *n* clés dans la `Map`. Rappelez-vous que `items()` est une méthode implantant un itérateur et permettant de parcourir toutes les clés d'une `Map`.

```
def __items__( ):
```

   b) (5) Quelle est la complexité en temps en pire cas pour insérer *n* paires *clé-valeur* dans une `UnsortedListMap` initialement vide.

   c) (5) Quelle est la complexité en temps en pire cas pour retirer *n* paires *clé-valeur* d'une `UnsortedListMap` qui contient initialement *n* paires.

2. (20) Considérez la skip list, *S*, et les opérations suivantes :




a) (10) Dessinez *S* après chaque opération en prenant les valeurs de coin_flip() suivantes : `_FACE, _FACE, _FACE, _TAILS, _FACE, _TAILS, _FACE, _TAILS`. Consultez l'Appendice C pour le code de `SkipInsert`.

`del S[12]:`

`S[31] = 'x':`

```
S[47] = 'y'
```

```
del[31]
```

```
S[45] = 'z'
```

b) (10) Combien de comparaisons au total pour les 5 opérations seront effectuées par la méthode `SkipSearch` (Appendice C).

3. (25) Considérez les tables de hachage résultant de l'utilisation des fonction de hachage h(i) = (2i + 3) mod 11 (fonction primaire) et h'(k) = 5 - ( k mod 5 ) (fonction secondaire) pour insérer les clés 3, 13, 26, 23, 11, 36, 54, 12, 8, 65, 20 dans cet ordre

   a) (10) en assumant que les collisions sont prises en charge par chaînage linéaire (linear probing). Montrez les états de la table après chaque insertion.

| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | | | | | | |
| | | | | | | | | | | |
| | | | | | | | | | | |
| | | | | | | | | | | |
| | | | | | | | | | | |
| | | | | | | | | | | |
| | | | | | | | | | | |
| | | | | | | | | | | |
| | | | | | | | | | | |
| | | | | | | | | | | |
| | | | | | | | | | | |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |

b) (15) en assumant que les collisions sont prises en charge par hachage double. Montrez les états de la table après chaque insertion.

| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | | | | | | |
| | | | | | | | | | | |
| | | | | | | | | | | |
| | | | | | | | | | | |
| | | | | | | | | | | |
| | | | | | | | | | | |
| | | | | | | | | | | |
| | | | | | | | | | | |
| | | | | | | | | | | |
| | | | | | | | | | | |
| | | | | | | | | | | |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |

4. (10) Considérez les arbres binaires de recherche. Insérez les clés { 1, 61, 80, 33, 98, 87 } dans cet ordre dans un arbre binaire de recherche (ABR) initialement vide.

(10) De quel nombre célèbre les clés insérées forment les décimales ?

5. (25) Considérez les arbres de recherche AVL.

   a) (15) Insérez dans un arbre AVL initialement vide les clés suivantes { 10, 20, 30, 40, 50, 60, 70 }, dans cet ordre. Dessiner les arbres résultants après chaque insertion.

b) (10) Supprimez une à une toutes et dans l'ordre croissant les clés de l'arbre AVL que vous avez obtenu en (a). Dessiner l'arbre résultant après chaque suppression. Lorsqu'un noeud doit être remplacé, utilisez le prédécesseur. N.B. Si l'arbre AVL que vous avez obtenu en (a) n'est pas bon, vous obtiendrez la note 0 pour (b).

6. (30) Considérez les arbres 2-4 et rouge-noir.

   a) (15) Insérez dans un arbre 2-4 initialement vide les clés suivantes { 70, 60, 50, 40, 30, 20, 10 }, dans cet ordre. Dessiner les arbres résultants après chaque insertion.

b) (15) Insérez dans un arbre rouge-noir initialement vide les clés suivantes { 70, 60, 50, 40, 30, 20, 10 }, dans cet ordre. Dessiner les arbres résultants après chaque insertion.

7. (10) Soit la table de programmation dynamique suivante pour comparer deux chaînes de caractères :

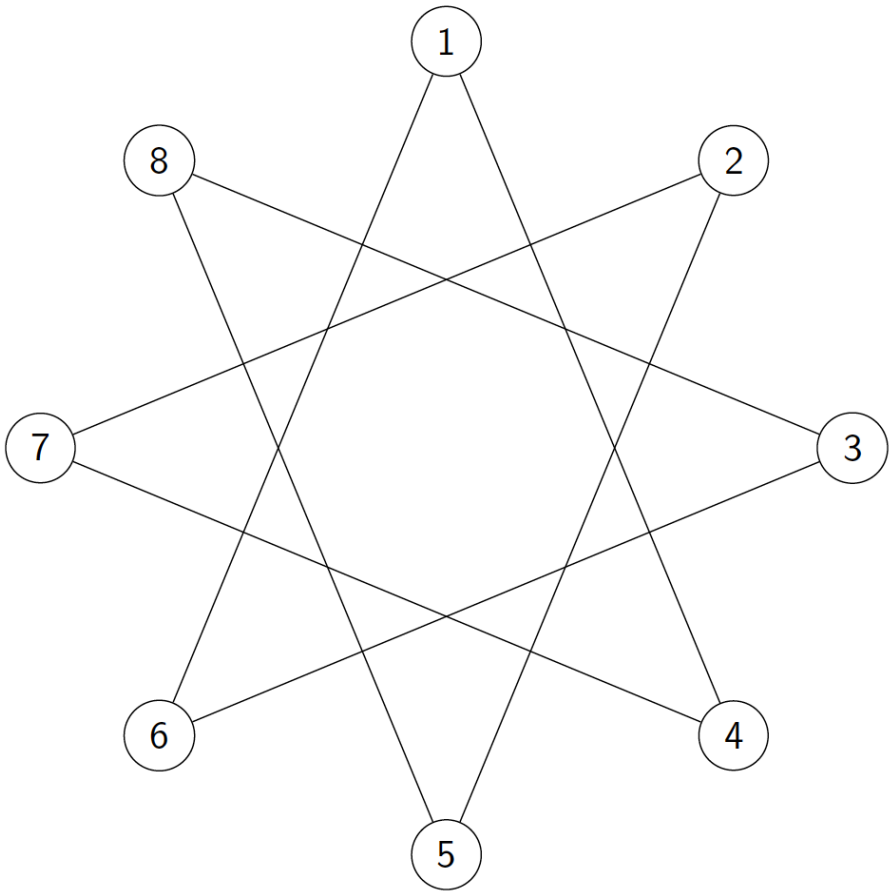
| | | m | y | m | m | e | c | a | c | e | c | o | i | t | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 |
| t | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| l | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| y | 2 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| m | 3 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| m | 4 | 0 | 1 | 1 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 |
| e | 5 | 0 | 1 | 1 | 2 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 |
| p | 6 | 0 | 1 | 1 | 2 | 3 | 4 | 4 | 4 | 4 | 4 | 4 | 4 | 4 | 4 |
| f | 7 | 0 | 1 | 1 | 2 | 3 | 4 | 4 | 4 | 4 | 4 | 4 | 4 | 4 | 4 |
| c | 8 | 0 | 1 | 1 | 2 | 3 | 4 | 4 | 4 | 4 | 4 | 4 | 4 | 4 | 4 |
| u | 9 | 0 | 1 | 1 | 2 | 3 | 4 | 5 | 5 | 5 | 5 | 5 | 5 | 5 | 5 |
| p | 10 | 0 | 1 | 1 | 2 | 3 | 4 | 5 | 5 | 5 | 5 | 5 | 5 | 5 | 5 |
| i | 11 | 0 | 1 | 1 | 2 | 3 | 4 | 5 | 5 | 5 | 5 | 5 | 5 | 5 | 5 |
| t | 12 | 0 | 1 | 1 | 2 | 3 | 4 | 5 | 5 | 5 | 5 | 5 | 5 | 6 | 6 |
| | 13 | 0 | 1 | 1 | 2 | 3 | 4 | 5 | 5 | 5 | 5 | 5 | 5 | 6 | 7 |

a) (5) Quelle est la plus longue sous-séquence commune ?

b) (5) Noircissez le chemin parcouru pour obtenir la plus longue sous-séquence, soit votre réponse à la question (a).

8. (10) Dessinez le `trie` standard contenant les chaînes de caractères suivantes (conservez l'ordre alphabétique des enfants) : { arbre, trie, arc, tree, arete, cycle, clique }

9. (10) Soit le graphe suivant. En respectant l'ordre croissant des noeuds adjacents, dans quel ordre seront visités les noeuds du graphe si on débute au noeud 1 :




a) (5) lors d'un parcours en profondeur ? (PS. une seule réponse est possible)

b) (5) lors d'un parcours en largeur ? (PS. une seule réponse est possible)

Brouillon :

Brouillon :

Brouillon :

## Appendice A : Map.py

```
import collections

class Map( collections.MutableMapping ):

    #nested _Item class
    class _Item:
        __slots__ = '_key', '_value'

        def __init__( self, k, v = None ):
            self._key = k
            self._value = v

        def __eq__( self, other ):
            return self._key == other._key

        def __ne__( self, other ):
            return not( self == other )

        def __lt__( self, other ):
            return self._key < other._key

        def __ge__( self, other ):
            return self._key >= other._key

        def __str__( self ):
            return "<" + str( self._key ) + "," + str( self._value ) + ">"

        def key( self ):
            return self._key

        def value( self ):
            return self._value

    def is_empty( self ):
        return len( self ) == 0

    def get( self, k, d = None ):
        if self[k]:
            return self[k]
        else:
            return d

    def setdefault( self, k, d = None ):
        if self[k]:
            return self[k]
        else:
            self[k] = d
            return d
```

## Appendice B : UnsortedListMap.py

```
from Map import Map

class UnsortedListMap( Map ):

    def __init__( self ):
        self._T = []

    def __getitem__( self, k ):
        for item in self._T:
            if k == item._key:
                return item._value
        return False

    def __setitem__( self, k, v ):
        for item in self._T:
            if k == item._key:
                item._value = v
                return
        #no match
        self._T.append( self._Item( k, v ) )

    def __delitem__( self, k ):
        for j in range( len( self._T ) ):
            if k == self._T[j]._key:
                self._T.pop( j )
                return
        return False

    def __len__( self ):
        return len( self._T )

    def __iter__( self ):
        for item in self._T:
            yield item._key

    def __contains__( self, k ):
        return self[k]
```

Appendice C : SkipSearch et SkipInsert

```
#search element
def SkipSearch( self, element ):
    p = self._start
    while not( p._belo is None ):
        p = p._belo
        while element >= p._next._elem:
            p = p._next
    return p


#insert element
def SkipInsert( self, element ):
    p = self.SkipSearch( element )
    if p._elem == element: #The keys are equal
        p._elem = element  #We must set the element to the new value
        return p
    #p points to the previous node
    #we're at the bottom level, so belo is None
    q = self.insertAfterAbove( p, None, element )

    #Insert at higher levels as determined by the coin_flip
    i = 0
    coin_flip = self._coin.flip()
    while coin_flip == _FACE:
        i += 1 #i indicates the current level of insertion
        if i >= self._height:
            self.increaseHeight()
        while p._abov is None:
            #we move to the previous Node
            p = p._prev
        #we move up one Node (to get to the desired level)
        p = p._abov
        #q is the previously inserted Node (belo the one to be inserted)
        q = self.insertAfterAbove( p, q, element )
        coin_flip = self._coin.flip()
    #before exiting, we increase the skip list count by one
    self._count += 1
    #we return the last inserted Node (the top level one)
    return q
```