Structures de Données Simples 2017 (CSV'17)

TD 2 : Tables associatives

Pour chaque exercice programmer une ou plusieurs fonctions et plusieurs tests unitaires.

Exercice 1 (compteurs). En considérant les tables associatives (`dict`) comme les fonctions avec domaine fini, implémenter les fonctions de l'union et de composition des fonctions. Par exemple,

```
f = {1 : 2, 2 : 4, 3 : 2}
g = {2 : 4, 4 : 7}
union(f, g) == {1 : 2, 2 : 4, 3 : 4, 4 : 7}
compose(f, g) == {1 : 4, 2 : 7, 3 : 4}
```

À noter que la fonction `union` doit produire une exception/erreur si les deux fonctions envoient la même clé aux valeurs différentes.

```
f = {1 : 2, 4 : 1}
g = {1 : 3, 4 : 6}
union(f, g) # erreur
compose(f, g) == {4 : 3}
```

Exercice 2 (multiensembles). Utiliser les tables associatives pour implémenter des compteurs (principalement, des multiensembles). Un compteur permet de compter le nombre d'occurrences des éléments. Par exemple, `d = {'a' : 1, 'b' : 3}` représente une occurrence de `a` et 3 occurrences de `b`. La procédure `add` ajoute une occurrence d'un élément à un compteur : `add(d,'a')` fait que `d == {'a' : 2, 'b': 3}` et `add(d,'c')` fait que `d == {'a' : 2, 'b': 3, c : '1'}`. La fonction `size` calcule le nombre total d'occurrences de tous les éléments : si `c = {'d' : 1, 'f' : 3}`, alors `size(c) = 4`. La fonction `merge` calcule la fusion de deux compteurs :

```
merge({'a' : 1, 'b' : 3}, {'b' : 2, 'c' : 4}) == {'a' : 1, 'b' : 5, 'c' : 4}
```

Les fonctions `min_key` et `max_key` trouvent un élément avec le moindre et le plus grand nombre d'occurrences : pour `d = {'a' : 2, 'b' : 4, 'c' : 3}` ça donne `min\_key(c) == 'a'` et `max\_key(c) == 'b'`. La fonction `cut` calcule la valeur minimale de tous les éléments en commun :

```
cut({'a' : 1, 'b' : 3, 'c' : 1}, {'b' : 2, 'c' : 4, 'd' : 5}) == {'b' : 2, 'c' : 1}
```

Exercice 3 (multimaps). Une relation binaire $R \subseteq A \times B$ peut être représentée en deux manières différentes : 1) avec un ensemble des paires ou 2) avec un multimap, un dictionnaire qui envoie un élément A à un ensemble d'éléments B. Par exemple, la relation $R = \{(1, a), (1, c), (2, b), (3, a)\}$ est représentée avec l'ensemble suivant

```
R = set([(1,'a'), (2,'b'), (1,'c'), (3,'a')])
```

et avec le multimap suivant

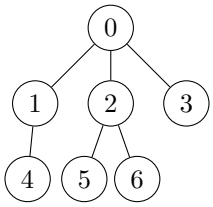
```
R = dict()
R[1] = set(['a'])
R[2] = set(['b', 'c'])
R[3] = set(['a'])
```

Pour les deux représentations implémenter des fonctions qui réalisent des opérations suivantes et caractériser leur complexité.

1. `add(R,a,b)` qui ajoute un lien à R : $R \cup \{(a,b)\}$,
2. `remove(R,a,b)` qui enlève un lien (s'il existe) : $R \setminus \{(a,b)\}$
3. `map_element(R,x)` qui renvoie un élément : $R(x) = \{y \mid (x,y) \in R\}$,
4. `inverse(R)` qui retourne l'inverse de la relation $R^{-1} = \{(b,a) \mid (a,b) \in R\}$,
5. `compose(R,P)` qui retourne la composition des relations : $R \circ P = \{(a,c) \mid \exists b.\ (a,b) \in R \wedge (b,c) \in P\}$.
6. `merge(R,P)` qui retourne l'union des deux relations : $R \cup P$.

Implémenter également des fonction de conversion entre les deux représentations.

Exercice 4 (arbres). Un arbre est une structure mathématique, en gros une relation, qui permet de représenter une hiérarchie (simple) dans un ensemble d'éléments. Par exemple, l'arbre suivant



est représentée avec la relation parent-enfant suivante

$$T = \{(0,1),(0,2),(0,3),(1,4),(2,5),(2,6)\}$$

qui souvent est représentée avec un multimap :

```
T[0] = set([1,2,3])
T[1] = set([4])
T[2] = set([5,6])
T[3] = set()
T[4] = set()
T[5] = set()
T[6] = set()
```

En utilisant la représentation d'un arbre avec un multimap, implémenter une fonction `traverse` qui parcourt l'arbre en largeur. Par exemple, `traverse(T)` donne la liste `[0, 1, 4, 2, 5, 6, 3]`. (Astuce : il faut implémenter une fonction qui identifié la racine, le seul noeud qui n'a pas de parent).

Exercice 5 (arbres récursives). Une représentation imbriquée des arbres est souvent utilisée. Un noeud est représente par une petite table avec des champs `label` qui stocke l'étiquette de noeud et `children` qui stocke la collection des enfants. Par exemple, l'arbre de l'exercice précédent est représenté de la manière suivante

```
T = {'label' : 0
     'children' : [
      { 'label' : 1,
        'children' : [
        { 'label' : 4, 'children' : []}]},
      { 'label' : 2,
        'children' : [
         { 'label' : 5, 'children' : []},
         { 'label' : 6, 'children' : []}]}
      { 'label' : 3, 'children' : []}]}
```

Implémenter la fonction `traverse` pour cette representation.

Exercice 6* (dictionnaires). Utiliser les arbres de préfixe pour implémenter une structure de données permettant de stocker un ensemble de chaînes de caractères et d'énumérer tous les chaînes de caractères ayant comme un préfixe la chaîne de caractères donnée.