Utilisation de dictionnaires

Fabrice Lembrez - PSI*

Lycée Pierre de Fermat

Plan

Dictionnaires - vus de l'utilisateur

Déf - Dictionnaire

Association clé $\rightarrow$ valeur avec plein de clés possibles.

Mathématiquement : application d'une partie $K' \subset K$ dans V.

Informatiquement : soit D[k] non défini, soit D[k] vaut v.

Dictionnaires - vus de l'utilisateur

Déf - Dictionnaire

Association clé $\rightarrow$ valeur avec plein de clés possibles.

Mathématiquement : application d'une partie $K' \subset K$ dans V.

Informatiquement : soit D[k] non défini, soit D[k] vaut v.

Affichage d'un dictionnaire sous la forme {clef:valeur,...}

$$\{1 : 18, \text{'a'} : 23, \text{'tor'} : \text{'tue'}\}$$

Dictionnaires - vus de l'utilisateur

Déf - Dictionnaire

Association clé $\rightarrow$ valeur avec plein de clés possibles.

Mathématiquement : application d'une partie $K' \subset K$ dans V.

Informatiquement : soit D[k] non défini, soit D[k] vaut v.

Affichage d'un dictionnaire sous la forme {clef:valeur,...}

$$\{1 : 18, \text{'a'} : 23, \text{'tor'} : \text{'tue'}\}$$

Cela sert aussi en création : `D=`$\{1 : 18, \text{'a'} : 23, \text{'tor'} : \text{'tue'}\}$

On peut aussi faire une création progressive :
- partir d'un dictionnaire vide `D=dict()` ou D={}
- puis créer des associations diverses : D[5]=3, D["toto"]=12 etc...

Plan

Les opérations "élémentaires"

Tester l'existence d'une clef : `if k in D` (porte sur la clef !)

Créer une association : `D[k]=v`

Modifier une association : `D[k]=v`

Supprimer une association :

vérifier qu'elle existe, puis `D.pop(k)` ou `del D[k]`

Les opérations "élémentaires"

Tester l'existence d'une clef : `if k in D` (porte sur la clef !)

Créer une association : `D[k]=v`

Modifier une association : `D[k]=v`

Supprimer une association :

vérifier qu'elle existe, puis `D.pop(k)` ou `del D[k]`

Ne pas faire pareil pour une liste Python

Pour une liste, si vous affectez L[k] alors que l'indice k n'est pas défini, vous avez une erreur.

Plan

Parcourir l'ensemble du dictionnaire

Parcours de l'ensemble du dictionnaire :

```
for clef in D
```

Il n'y a pas d'ordre "naturel" dans ce cas.

Mais il y a d'autres options (qui ne sont que des raccourcis syntaxiques, je les signale pour ne pas être surpris c'est tout)

- `for clef in D.values()` pour énumérer uniquement les valeurs
- `for clef in D.items()` pour énumérer les couples (clés,valeurs)

Plan

Dictionnaires - précautions

Différence/ressemblance avec la syntaxe des "listes Python"

Si la clef numéro k n'existe pas :

- pour une liste Python écrire `L[k]` renvoie de toute façon une erreur
- pour un dictionnaire, on peut créer l'association `D[k]=...` mais pas évaluer sa valeur, l'utiliser dans des calculs

Dictionnaires - précautions

Différence/ressemblance avec la syntaxe des "listes Python"
Si la clef numéro k n'existe pas :

- pour une liste Python écrire `L[k]` renvoie de toute façon une erreur
- pour un dictionnaire, on peut créer l'association `D[k]=...` mais pas évaluer sa valeur, l'utiliser dans des calculs

Précaution cruciale dans les tests

Toujours tester l'existence d'une clef AVANT de la lire. Utiliser le caractère "paresseux" du and pour cela. **Attention à l'ordre des termes.**

Dictionnaires - précautions

Différence/ressemblance avec la syntaxe des "listes Python"
Si la clef numéro k n'existe pas :

- pour une liste Python écrire `L[k]` renvoie de toute façon une erreur
- pour un dictionnaire, on peut créer l'association `D[k]=...` mais pas évaluer sa valeur, l'utiliser dans des calculs

Précaution cruciale dans les tests

Toujours tester l'existence d'une clef AVANT de la lire. Utiliser le caractère "paresseux" du and pour cela. **Attention à l'ordre des termes.**

Exemple pour une liste Python

```
if i<len(L) and L[i]>3
```

Exemple pour une entrée de dictionnaire

```
if i in D and D[i]>3
```

Plan

Dictionnaires - caractéristiques

Caractéristiques du dictionnaire

Il n'y a pas de liste prédéfinie de clés. Pourtant,
Rechercher / Insérer / Supprimer sont en O(1).

Dictionnaires - caractéristiques

Caractéristiques du dictionnaire

Il n'y a pas de liste prédéfinie de clés. Pourtant,
Rechercher / Insérer / Supprimer sont en O(1).

	Tableau	Dictionnaire
Clefs	Fixes (0 à n-1)	Libres
Lire/modifier à une valeur	direct par la clef	direct par la clef
Insérer, supprimer	impossible	OK en O(1)
Rechercher	Parcours complet O(n)	OK en O(1)

Mais alors pourquoi utiliser des tableaux ordinaires ?

Dictionnaires - caractéristiques

Caractéristiques du dictionnaire

Il n'y a pas de liste prédéfinie de clés. Pourtant,
Rechercher / Insérer / Supprimer sont en O(1).

	Tableau	Dictionnaire
Clefs	Fixes (0 à n-1)	Libres
Lire/modifier à une valeur	direct par la clef	direct par la clef
Insérer, supprimer	impossible	OK en O(1)
Rechercher	Parcours complet O(n)	OK en O(1)

Mais alors pourquoi utiliser des tableaux ordinaires ?

Le dictionnaire est une structure plus avancée, plus gourmande en espace mémoire. Il y a des coefficients dans le O(1). Utiliser un dictionnaire là où un tableau suffit n'est pas pertinent. On regardera tout cela de plus près.

Plan

Dictionnaires - intérêt

Intérêt du dictionnaire : mise en mémoire lors d'un parcours

- garder la mémoire des rencontres utiles,
- accès direct à l'info : as-tu vu ... ? combien de fois ?
- mémoriser des infos sur les objets rencontrés
- attribuer un état aux objets rencontrés
  (alternative à l'idée de "coloriage" des sommets).

Exemples :

- cahier d'appel : liste d'étudiants présents $\rightarrow$ untel est-il là ?
- histogramme : liste de valeurs $\rightarrow$ histogramme des valeurs
- arbre (généalogique) : parcours $\rightarrow$ dictionnaire des ancêtres
- graphe : dictionnaire des sommets visités $\rightarrow$ est-ce une boucle ?

Plan

Dictionnaires - code de base

Ici un "cahier d'appel" ou même un histogramme, où l'on note le nombre d'apparitions de chaque nom. Une fois le cahier construit, on peut demander si untel est présent, et combien de fois il l'est, en O(1), sans reparcourir.

```
def cahierAppel(listeNoms):
    cahier=dict()
    for nom in listeNoms:
        if nom in cahier:
            cahier[nom]+=1
        else:
            cahier[nom]=1
    return cahier #histogramme des présences
```

Plan

Problème : programmer une formule de récurrence

Exemple : calcul du coefficient binomial, donné par les relations

$$\binom{n}{p} = \binom{n-1}{p-1} + \binom{n-1}{p}, \qquad \binom{n}{0} = 1, \binom{n}{0} = 1 \text{ pour } n > 0$$

Problème : programmer une formule de récurrence

Exemple : calcul du coefficient binomial, donné par les relations

$$\binom{n}{p} = \binom{n-1}{p-1} + \binom{n-1}{p}, \qquad \binom{n}{0} = 1, \binom{n}{0} = 1 \text{ pour } n > 0$$

- Solution 1, impérative

Il est parfaitement raisonnable d'y voir le remplissage d'une "liste de listes" et de faire un calcul progressif des `B[n][p]`$=\binom{n}{p}$, ligne par ligne.
Aspect concret deux boucles imbriquées $0 \leq i \leq n$ et, à l'intérieur, $0 \leq j \leq p$. Complexité : $O(np)$.

Solution 2, récursive, version naïve

```
def binom(n,p): #version naïve
    if p==0:
        return 1
    if n==0: #et p>0
        return 0
    return binom(n-1,p-1)+binom(n-1,p)
```

Solution 2, récursive, version naïve

```
def binom(n,p): #version naïve
    if p==0:
        return 1
    if n==0: #et p>0
        return 0
    return binom(n-1,p-1)+binom(n-1,p)
```

La difficulté est liée au dédoublement des appels, de façon redondante : la complexité spatiale devient très élevée (de l'ordre de $2^{\min(n,p)}$).

Solution 3 : récursive avec mémoïsation

Déf - Mémoïsation

C'est le stockage des valeurs renvoyées par une fonction (à l'aide d'une structure adaptée), pour éviter des appels superflus.

Structure adaptée ?

Il faut pouvoir répondre rapidement à la question : cette valeur a-t-elle été appelée ? et si oui, avec quel retour ?
On emploiera donc un **dictionnaire.**

Solution 3 : récursive avec mémoïsation

```
def memoBinom(n,p,M):
    if p==0:
        M[(n,p)]=1
    elif n==0: #et p>0
        M[(n,p)]=0
    else:
        memoBinom(n-1,p-1,M)
        memoBinom(n-1,p,M)
        M[(n,p)]=M[(n-1,p-1)]+M[(n-1,p)]

def coeffBinomial(n,p):
    M=dict()
    memoBinom(n,p,M)
    return M[(n,p)]
```