Informatique fondamentale

Structures de données

R.Gosswiller

1. Introduction
2. Types majeurs
3. Dictionnaires
4. Autres structures
5. Etude de complexité
6. Tris

Introduction

Structures

Définition

Une structure de données est un type de variable dont le rôle est de stocker un ensemble d'éléments

Utilisation

Manipuler et traiter des ensembles regroupés sous la même variable

Structures

Critères de construction d'une structure

- Taille
- Indexation
- Doublons
- Accès concurrent
- Méthodes de parcours
- Méthodes de tri
- Organisation en mémoire

Taille

Principe

Chaque structure peut avoir une taille fixe ou dynamique en mémoire

Taille fixe

Déterminée à l'initialisation

Taille dynamique

Adaptée aux ajouts
Taille non connue à l'avance, modifiée à l'exécution

Indexation

Principe

Pointeur sur chaque élément
Ordre (ou non-ordre) de classement en mémoire

Indexation numérique

Chaque élément est associé à une clé numérique unique

Clés/valeurs

Chaque clé peut être d'un type quelconque
Type fixe non modifiable (Str, int, ...)

Accès concurrent

Principe

Dans le cas d'un accès parallèle, comment sont gérées les priorités d'accès ?
Ordres de priorité
Cadencement
Sémaphores et mécanismes d'attente
Déterminisme

Types majeurs

Listes

Type list

Ensemble typé d'éléments

```
1 lst = []
2
3 lst = [1,42,1337]
4
5 lst= ["hello", "world", "Python"]
```

Listes

Eléments indexés

Chaque élément possède une position dans la liste, indiquée par un entier
Cet entier est appelé l'index de l'élément

Exemple

```
1 lst = ["a","b","c","d","e","f"]
2 print lst[0] // prints "a"
3 print lst[4] // prints "e"
4 print lst[1:3] // prints "bcd"
```

Ajout d'un élément

```
1 lst = ["a","b","c"]
```

En tête

```
1 lst = ["new"] + lst
```

En queue

```
1 lst.append("new") // ["a","b","c","new"]
2 lst.extend("new") // ["a","b","c",["new"]]
```

Indexé

```
1 lst.insert("new", 2) // ["a","b","new","c"]
```

Suppression

Fonction in

Contrôle de la présence d'un élément dans la liste

```
1 lst.in("c") // Returns True
2 lst.in("t") // Returns False
3 lst.index("b") // Return 1
```

Suppression

```
1 lst.pop(2) // ["a","b"]
2 lst.remove("b") // ["a","c"]
```

Parcours d'une liste

Récursif

```
1 def parcours(n, lst):
2         if(n == len(lst)):
3                 return n;
4         print(lst[n])
5         return parcours(n+1, lst)
```

Range

Définition

Range permet de définir un intervalle de valeurs
Renvoie une liste
Boucles et parcours

Syntaxe

```
1 range (limit) // From 0 to limit
2           [0,1,2,...,limit-1]
3
4 range(1, 10, 2) // 1, 3, 5, 7, 9
```

Range

Prototype

```
1 range ([start,]stop[,step])
```

Listes/Range

Fonctions

in : vérifier la présence d'un élément
append : ajout d'élément
insert : ajout à une position donnée
extend : combinaison de listes
len : longueur

Autres

count, reverse, sort, del, ...

Listes et String

Attention !

Les String sont des types particuliers de listes !

Exemple

```
1 myStr = "idolovePythonverymuch"
2
3 print myStr[0:5] // "idolo"
```

Dictionnaires

Dictionnaires

Définition

Ensemble de paires clé/valeur
Chaque association est unique
Tableaux associatifs

Construction

```
1 dico = {"toto":1, "titi":2, "tata":3}
2
3 print dico["toto"] // Prints 1
4
5 type(dico) // Prints dict
```

Dictionnaires

```
1  ages={"bob":15,"alice":20, "dave:48","miranda":67"}
2
3  "bob" in ages // Renvoie True
4
5  "bob" in ages // Returns True
6  "bill" in ages // Returns False
7
8  list(ages.keys()) // ["bob","alice","dave","miranda"]
```

Construire un dictionnaire

La fonction dict

La fonction dict permet de convertir un ensemble en dictionnaire
Conversion de listes, de tuples

Syntaxe

```
1 dico = dict([("bob",15),("alice",23)])
2
3 dico = dict({"bob":15,"alice":23})
```

Dictionnaires et fonctions

Ajout

```
1 dico["bill"] = 15
```

Suppression

```
1 del dico["bill"]
2 del dico[3] // Indexation
3 del dico
4 dico.clear()
```

Pointage

```
1 dico.get("bill") // Renvoie None par defaut
2 dico.pop("bill") // Renvoie la valeur (15)
3                  // et retire la ligne du dico
4 dico.values() // Donne les valeurs
```

Autres structures

Piles/Files

Principe

Les piles et files sont des cas particuliers de listes
Accès FIFO (First In First Out) ou LIFO (Last In First Out)
Premier et/ou dernier élément uniquement accessible
Ajout en tête/en queue

Piles

```
1 stack = ["a","b","c"]
2
3 stack.append("d") // ["a","b","c","d"]
4 stack.pop() // "d" -> ["a","b","c"]
```

Files

```
1 queue = ["a","b","c"]
2
3 queue.append("d") // ["a","b","c","d"]
4 queue.popleft() // "a" -> ["b","c","d"]
```

Tuples

Définition

Liste à éléments non modifiables
Ensemble de groupes
Parenthèses optionnelles

Exemple

```
1 tupA = ('a','b','c')
2 tupB = tupA + d // ('a','b','c','d')
```

Combinaisons

Principe

La programmation en Python permet la combinaison d'ensembles de types variés !
Exemple : Liste de tuples, dictionnaire de listes, dictionnaire de dictionnaire, etc...

Etude de complexité

Complexité

Principe

Etudier le temps de calcul pire cas
Calcul du nombre d'opérations élémentaires

Détails

Calculer les ressources nécessaires à l'exécution d'un algorithme
Ressources en temps, ressources en mémoire

Complexité

Attention

La complexité est évaluée en fonction d'un paramètre d'entrée
Plusieurs paramètres peuvent impliquer plusieurs évaluations de la complexité
On regarde le paramètre le plus influent

Complexité

Classes de complexité

- Sublinéaire : $o(log(n))$
- Linéaire : $o(n)$
- Sub-quadratique $o(log(n) * n^2)$
- Quadratique/Cubique : $o(n^2), o(n^3)$
- Polynomiaux $o(n^k)$ (classe P)
- Exponentiels $> o(n^k)$

Complexité polynomiale

Ensemble des problèmes calculables en un temps "raisonnable"

Evaluer la complexité

On compare le nombre d'instructions d'une itération sur l'autre

- $c(n+1) = c(n) \rightarrow$ Complexité constante $o(1)$
- $c(n+1) = c(n) + x \rightarrow x$ constante, Complexité linéaire $o(n)$
- $c(n+1) = c(n) + \epsilon c(k*n) = c(n) + k \rightarrow \epsilon$ valeur variable indépendante de n, Complexité sublinéaire $o(log(n)))$
- $c(n+1) = c(n) + n \rightarrow$ Complexité quadratique $o(n^2)$
- $c(n+1) = c(n) + 2*n \rightarrow$ Complexité cubique $o(n^3)$
- $c(n+1) = c(n) + k*n \rightarrow$ Complexité polynomiale $o(n^{k-1})$
- $c(n+1) = c(n)*2 \rightarrow$ Complexité exponentielle $o(2^n)$

Exemple

Objectif : lire un livre, page par page

Complexité constante

Lire la première page

```
1 def readBook(n):
2   readPage(1)
```

$c(n+1) = c(n) \rightarrow o(1)$

Complexité logarithmique

Lire quelques pages

```
1 def readBook(n):
2   readPage(1)
3   readPage(10)
4   ...
5   readpage(10^n)
```

$c(n+1) = c(n) + \epsilon \rightarrow o(log(n))$

Exemple

Objectif : lire un livre, page par page

Complexité linéaire

Lire chaque page

```
1  def readBook(n):
2    readpage(n)
3    if(n>1):
4      readBook(n-1)
```

$c(n+1) = c(n) + 1 \rightarrow o(n)$

Complexité quadratique

Recommencer à chaque page

```
1  def readPage(i):
2          print("Reading page "+str(i))
3
4  def readBook(page):
5          if(page == 0):
6                  return 0
7          if(page >=1):
8                  readBook(page-1)
9                  readPage(page)
10
11 def readBookN(pages):
12         if(pages==0):
13                 return 0
14         readBookN(pages-1)
15         readBook(pages)
```

$$c(n+1) = c(n) + n$$
$$i = \frac{n(n+1)}{2} = o(n^2)$$

Tris

Les tris de structures

Principe

L'un des intérêts de travailler sur des ensembles est de pouvoir les ordonner
Tri automatisé/manuel

Le tri à bulles

Principe

Permuter successivement les éléments pour faire remonter les valeurs les plus hautes

Le tri à bulles

Algorithme

```
1  triBulles(tab):
2      POUR i de 1 a n:
3          POUR j de 1 a i-1:
4              SI tab[j] > tab[j+1]
5                  permuter(tab[j], tab[j+1])
```

Complexité

$o(n^2)$

Le tri par insertion

Principe

Insérer dans le tableau les éléments 1 par 1, de manière ordonnée

Détails

Lent et coûteux sur des entrées de grande taille
Simple d'implémentation
Intuitif

Les algorithmes de Tri

Intitulé	Temps	Espace
Tri à bulles	$o(n^2)$	1
Quicksort	$o(n^2)$	$o(log(n))$
Tri par insertion	$o(n^2)$	1
Tri par tas	$o(n * (log(n)))$	1
Tru cocktail	$o(n^2)$	1
Tri fusion	$o(n * (log(n)))$	1
Timsort	$o(n^2)$	$o(n)$

La fonction sorted

Principe

La fonction sorted permet de trier automatiquement des ensembles selon leur valeur

Syntaxe

```
1 a[1,5,3,2,4]
2 sorted(a) // [1,2,3,4,5]
3 a.sort() // Auto assignation
```

La fonction sorted

Tris d'ensembles

```
1  listPeople = [("Bill",48, 198),
2                ("Alice",23,171),
3                ("Bob",32,164)]
4
5  sorted(listPeople, key=lambda people:people[1])
6  // [("Alice",23,171),("Bob",32,164),("Bill",48,198)]
7
8  def donneElementZero(table)
9          return table[0]
10
11 sorted(listPeople, key=donneElementZero)
12 // [("Alice",23,171),("Bill",48,198),("Bob",32,164)]
```

Conclusion

- Ensembles de données
- Complexité algorithmiques
- Algorithmes de tri