1. Les données
2. Les structures
3. Les structures de données

4. Objectifs de l'organisation des données

5. Les collections séquentielles

Une collection séquentielle permet de ranger des objets dans un ordre arbitraire. On parle de collection indexe quand on peut accéder a chaque élément de la collection par un numéro d'ordre ou indexe.

Le choix d'une implémentation particulière dépend d'un certain nombre de compromis, comme l'occupation mémoire, ou les performance requise pour divers opération de base : Itération, ajout d'un élément(au début, a la fin ou encore dans un emplacement quelquon que de la collection), indexation, suppression d'un éléments décompte du nombre d'éléments.

Il existe 02 grand type de collection séquentielle indexe :

- Les listes,
- Les tableau ou vecteurs

Un certain nombre de structure de donnée sont des restrictions de collection séquentielle qui n'autorise qu'un sous ensemble des opérations de bases :

- Les piles
- Les files

6. La collection de donnée

Une collection est un regroupement fini de donnée dont le nombre n'est pas fixé a priori.

Ainsi les collections que l'on utilise en informatique sont des objets dynamiques. Le nombre de leurs éléments varie au cours de l'exécution du programme puisqu'on peut y ajouter et supprimer des éléments en cours de traitement.

Plus précisément, les principales opérations que l'on autorise sur les collections sont les suivants

- Déterminer le nombre d'élément de la collection
- Tester l'appartenance d'un élément a la collection.
- Ajouter un élément a la collection
- Supprimer un element de la collection

Exemple : TAS de chaussures

Dans le TAS de chaussure, il est très facile d'ajouter une paire, il suffit de la jeter sans précaution sur les autres chaussures. Par contre pour supprimer une chaussure particulière du TAS, ça sera

beaucoup plus difficile car il faudra d'abord la retrouver cette chaussure dans cet amas de chaussure non-structure.

Collection Séquentielle ← Colllection de données

On distingue classiquement 03 grand type de collection :

- Les séquences
- Les arbres
- Les graphes

7. Les Séquences

Une séquence est une suite ordonnée d'élément éventuellement vide accessible par leurs rangs dans la séquence.

Dans une séquence chaque élément a un prédécesseur (sauf le premier élément qui n'a pas d prédécesseur) et un successeur (sauf le dernier élément qui n'as pas de successeur)

Image 1

Une liste de nom, une pile d'assiette ou une fil de spectateur sont des exemple de structure séquentielle de la vie courante.

En langage Python, on utilisera 03 types de séquence :

- Les chaines de caractère(Type str. Exemple : « bonjour », « ca va ? »)
- Les n-uplets(Type tuple, Exemple : exemple : (1 , (1,2,3), ('a', 2, (1,2,3)))
- Les listes (Type List, exemple : [], [a,b,c,d,e], [1, 'e', [1, 2, [x, y]]])

8. Les arbres

Un arbre est une collection d'éléments appeler nœud organiser de façon hiérarchique a partir d'un nœud particulier appeler la racine de l'arbre.

9. Les graphes

Un graphe est une collection d'élément appeler sommet et de relation entre ces sommets. Dans une graphe chaque élément peut avoir plusieurs prédécesseurs et plusieurs successeurs.

Un même élément peut être a la fois prédécesseur et successeur d'un autre sommet y compris de lui-même.

Les graphes permettent de manipuler plus facilement les objets et leurs relations.

10. Chaines de caractères

Une chaines de caractères est une séquence non modifiable de caractères.

D’un point de vu syntaxique, une chaine de caractères est une suite quelqu’onque de caractère délimiter soit par les apostrophe soit par les guillemets, soit par les griffes.

Exemple : ‘’ C’est ça ! ‘’

‘’ Nos Etudiants confondent les quotes et les guillemets‘’

11. Les listes

Une liste est une sequence modifiable d’element. D’un point de vue sybthaxique, une liste est une suite d’element separer par des virgule en encadrer par des crochets.

12. Les piles

Une pile est une séquence non-modifiable a laquelle on peut modifier ni supprimer car une seule extrémité : le sommet de la pile

13. Les files

Une file est une séquence dans laquelle on peut ajouter un élément car une seule extrémité et ne supprimer un élément qu’à l’autre extrémité : la tète de la file.

14. La complexité

On définit la complexité d’un algorithme A comme une fonction fA(n) de la taille n des données. Pour analyser cette complexité, on s’attache a déterminer l’ordre de grandeur asymptotique de fA(n).

On cherche une fonction connue qui a une rapidité de croissance voisine a celle de fA(n). On utilise pour cela la notation mathématique f = θ(g) qui se lit « f est en grand θ de g » et qui indique a quel rapidité une fonction augmente ou diminue.

*Les différents types de complexité sont :

Tableau 1

Notation	Type de Complexite
Θ(1)	Complexite constante
Θ(log(n))	Complexite logarithmique
Θ(n)	Complexite Lineaire
Θ(nlog(n))	Complexite quasi-Lineaier
Θ(n2)	Complexite quadratique
Θ(n3)	Complexite cubique
Θ(nP)	Complexite Polynomiale

Θ(n!)	Complexite factorielle
Θ(2expo(n))	Complexite exponentielle

15. Operation sur les structures

On a deux types d'operation sur les structures a savoir :

- L'operation de recherche
- L'operation de modification

*Recherche :

. SEARCH (S,k) : retourne une pointeur x vers un élément dans S tel que x.cle = k ou null si un tel élément n'appartient pas a S.

. MINIMUM (S) : retourne un pointeur vers l'element ayant la plus petite clé

. MAXIMUM (S) : retourne un pointeur vers un élément ayant la plus grande cle

. SUCCESSOR(S, x) et PREDECESSOR(S, x): retourne respectivement un pointeur vers l'element tout juste plus grand et tout juste plus petite que x dans S

*Modification :

.INSERT(S,k) et DELETE(S,k): insert l'élément dans S et en même retire l'element dans S

16. Les collections Java

Les interfaces list et Set implémente directement l'interface collection et que l'interface Map gravite autour de cette hiérarchie tout en faisant partie des collections Java.

Figure 2

- Les objets de type list

Ils servent à stocker des objets sans condition particulière sur la façon de les stockée. Il accepte tous les valeurs, même les valeurs null.

- Les objets de type set :

Ils sont un peu plus restrictive car il n'autorise pas deux la même valeur. Ce qui est pratique pour une liste d'élément unique par exemple

- Les map :

Elles sont particulières parce qu'elle fonctionne avec un système cle – valeur pour ranger et retrouver les objets quels qu'elle contienne.

- Les objets LinkedList

Une liste chainée (linkedlist) est une liste dans laquelle chaque élément est liée au élément adjacent par une référence a cette dernière.

Chaque élément contient une référence a l'élément précédent suivant excepter le premier dont l'élément précédant vaut null.et le dernier dont l'élément suivant vaut null.

```
import java.util.LinkedList;
import java.util.List;
import java.util.ListIterator;

public class Test {
    public static void main(String [] args) {
        List l = new LinkedList();
        l.add(12);
        l.add("toto!   !");
        l.add(12.20f);
        for(int i =0; i< l.size(); i++){
            System.out.println("Element d'index "+ i + " = "+ l.get(i));
        }
    }
```

- L'objet ArrayList :

ArrayList est un de ces objets qui non pas de taille limite et qui en plus accepte n'importe quel type de donnee y compris null. Nous pouvons mettre tout ce que nous voulons dans une arrayList. Voici un nouveau code qui le code.

```
import java.util.ArrayList;

public class Test {
    public static void main(String [] args) {
        ArrayList al = new ArrayList();
        al.add(12);
        al.add("Une chaine de charactere");
        al.add(12.20f);
        al.add('a');
        for(int i =0; i< al.size(); i++){
            System.out.println("Element d'index "+ i + " = "+ al.get(i));
        }
    }
}
```

Si vous excutez ce code, vous obtiendrai le resultat suivant :

Element d'index 0 = 12

Element d'index 1 = Une chaine de charactere

Element d'index 2 = 12.2

Element d'index 3 = a

Panel de Methode fourni avec cet objet ArrayList :

- add() : permet d'ajouter un element
- get(int index) : retourne a l'indice demander
- remove(int index) : efface l'entier a l'index demander
- IsEmpty() : Renvoie vrai si l'objet est vide
- RemoveAll() : Efface tout les contenu de l'objet

- Le objets Map :

Une collection qui fonctinne avec un type cle et valeur.

- Les objets Set :

Un set est une collection qui n'accepte pas les doublons. Par exemple, elle n'accepte qu'une seule fois null car 02 valeur null sont considere comme un doublons.

On trouve parmis les set les objets :

- HashSet
- treeSet
- LinkedHashSet

- Cas de HashSet

L'objet HashSet est sans nulle doute le plus utiliser des implementation des interface Set. On peut parcourir ce type d'iteration avec un objet Iterator pour extraire de ce objet un tableau d'objet.

Exemple de code :

QCM

1. Le type d'une variable

17. Structure de données en Pseudo-langage

On a besoin d'un langage formel minimum pour décrire un algorithme . Un langage de programmation (Java, c, kobol, Fortran, Pascal, Prolog, Perl, Python …) est trop contraignant.

Dans la littérature, les algorithmes sont décrit dans un pseudo-langage qui ressemble a un langage.

Les pseudo-langages recouvrent les memes concepts: Les variable, les affections, les structures de contrôle, les fonction et procedures et les structures de données.

Les variables sont indiquées par leurs types tel que booleen b, entier n, reel x, caractere c, …etc

Le signe de l'affection n'est pas egale : « = » ni « := » comme en pascal mais « <- » qui illustre bien les realite de l'affection (mettre dedans).

Les tableaux sont utiliser, si A est tableau A[i] est le i-eme élément du tableau.

Les structures sont utiliser si P est une structure modelisant un point x un champ de cette structure represante l'abscisse du point, P.x est l'abscisse de P.

Les instruction simple sont sequencer par « ; » et les blocs d'instruction sont entourer par des accolade « { }» ou par des mots DEBUT ….. FIN

- Le conditionnelle

```
Si (condition){
    instruction 1;
} SINON {
    instruction 2;
}
```

- Les iterations

```
Tant que (condition) { ... }
Faire { ... } Tant que (condition)
REPETER {...} JUSQU'A (condition)
POUR i de min a max FAIRE { ... }
```

- Les fonctions :

Une fonction a une liste de parametre typer et un type de retour.

Exemple : MaFonction(e int i, s int j, es int k)

  - En Entrée : la fonction lit la valeur du parametre ici i, les modifications quel fera avec i ne serons pas transmisse au programme appelant.
  - En Sortie : La fonction ne lit pas la valeur du parametre ici j, elle ecrit dans j et le programme appellant recupere cette valeur donc j peut etre modifier par la fonction
  - En Entrée-Sortie : la fonction lit la valeur du paramètre ici k. Elle passe au programme appelant les modifications faites pour k.

Le passage en Entrée-Sortie est souvent appeler passage par référence ou par variable.

Exemple : Code Appelant

```
i <- 3
j <- 5
k <- 8
MaFonction(i, j, k)
Afficher(i, j, k)
MaFonction(e int i, s int j, es int k) {
    i <- i + 1
    j <- 6
    k <- k + 2
}

Resultat de Afficher(i, j, k) = 3, 6, 10
```

CH2. STRUCTURES ARBORESCENTE

Introduction

Nous avons vu ou appris qu'il n'y plus de notion de disque sur Unix ; il en est de même pour les périphériques. De facon generale, tout est fichier. On ne voit donc au niveau utilisateur qu'une seule arborescence constitue de répertoire et de fichier décrivant les ressources du système.

Le nom d'un fichier sous Unix est une suite de caractere. Il n'existe pas de notion de type de fichier et de numero de version comme sur MS DOS.

Le caractere « . » est considerer comme etant un caractere dans le nom du fichier et non pas comme sur MS-DOS ou le caractere « . » de separation entre le nom d'un fichier et son type.

Il est donc possible d'avoir un fichier dont le nom comporte plusieurs points.

La philosophie du nom des fichier ressemblerai donc a celle l'environement suivant :

Figure 3

- Un arbre est un graphe sans cycle ou des nœud sont relier par des arretes. On distingue 03 sortes de nœud :
    - Les nœuds interne qui ont des fils
    - Les feuilles qui n'ont pas de fils
    - La racine de l'arbre qui est l'unique nœud ne possedant pas de père
- La profondeur d'un nœud est la distance c'est-à-dire le nombre d'arrete de la racine
- La hauteur d'un arbre est la plus grande profondeur
- La taille d'un arbre est son nombre de nœud en comptant les feuillees

1. Arbre binaire

En informatique, un arbre binaire est une structure de donnee qui peut se representer sous la forme d'une hierarchie dont chaque element est appeler nœud , le nœud initial etant appeler racine

Dans un arbre binaire chaque element possede deux element , habituellement appeler gauche et droit.

Du point de vue de ces element fils, l'element dont il sont issu au niveau superieur est appeler père.

Au niveau le plus elever, il y'a donc un nœud racine. Au niveau directement inferieur, il y a au plus deux nœud fils. En continuant a descendre au niveau inferieur, on peut en avoir 4 puis 8, 16 etc. C'est-à-dire la suite des puissances de 2. Un nœud n'ayant aucun fils est appeler feuille.

2. Les types d'arbre binaire
    - Un arbre binaire ou binaire unaire est un arbre avec racine dans lequel chaque nœud a au plus 02 fils.
    - Un arbre binaire entier est un arbre ou les fils possede 0 ou 2 fils

- Un arbre binaire parfait est un arbre binaire entier dans lequel toute les feuilles sont à la même distance de la racine

L'arbre binaire parfait est parfois nomme arbinaire complet

3. Arbre binaire de recherche

Un arbre binaire de recherche(abr) est arbre binaire dans lequel chaque nœud possede une etiquette, tel que chaque nœud du sous arche gauche est une etiquette inferieur ou egale a celle du nœud considere et que chaque nœud du sous arbre droite possede une etiquette superieur ou egale a celle-ci. Les nœuds que l'on ajoute devienne des feuilles de l'arbre.

Figure 4

- Recherche : La recherche dans un arbre binaire de recherche d'un nœud ayant une etiquette particuliere est un proceder recursive. On commence par examiner la racine, si l'etiquette de la racine est l'etiquette de la racine est l'etiquette rechercher, l'algorithme se termine et renvoie la racine. Si l'etiquette chercher est inferieur alors elle est dans le sous graphe gauche sur lequel on effectue recursivement la recherche.

De meme si l'etiquette rechercher est strictement superieur a celle de la racine, la recherche continue sur le sous-arbre droite. Si on atteind une feuille dont l'etiquette n'est pas celle rechercher alors sait que cette etiquette n'est pas dans l'arbre.

Cette operation requiert une complexite dans le pire des cas de $\theta(n)$.

- Insertion :

L'insertion d'un nœud commence par une recherche. On cherche l'etiquette du nœud a inséré. Lorsqu'on arrive à une feuille, on ajoute le nœud comme fils de la feuille en comparant son etiquette a celle de la feuille. Si elle est inferieur le nouveau nœud sera a gauche sinon elle sera a droite.

Exemple : Ajouter dans l'arbre si dessus les nœud 11 et 5.

Figure 5.

- Suppression

Plusieurs cas sont a considérer quand il s'agit de la suppression

  - Suppression d'une feuille : Il suffit de l'enlever de l'arbre étant donnée qu'elle n'a pas de fils

Ex : Sur l'arbre si dessus suprimer le nœud 7 on auras comme résultat l'arbre sans le nœud 7.

  - Suppression d'un nœud avec 1 seul fils : On l'enlève de l'arbre et on le remplace par son fils.

Ex : Sur l'arbre si dessus supprimer le nœud 10. On obtient :

Figure 6

- Suppression d'un arbre avec 2 fils : Supposons que le nœud a supprimer soit appeler N (Le nœud de valeur 7 dans le schéma ci-dessous), on le remplace alors par son successeur le plus proche donc le nœud le plus a gauche du sous-arbre droit(Ici le nœud de valeur 9) ou sont plus proche predecesseur donc le nœud le plus a droite du sous arbre gauche(ici le nœud de valeur 6) .

Figures 7

4. Methode d'iteration des arbres binaires

Souvent il est souhaitable de visiter chacun des nœuds dans un arbre et d'y examiner la valeur. Il existe plusieurs ordres dans lesquels les nœuds peuvent etre visites. Et chacun a des propriétés utiles qui sont exploiter par les algorithmes bases sur les arbres binaires. Nous avons le parcours préfixe, infixe et postfixe par quoi appeler preordre, inordre, postordre.

Soit une structure ordre dont la racine est A et une référence gauche et de droite de ces deux fils. Nous pouvons ecrire les fonctions suivantes :

PARCOURS PREFIXE

```
Visiter Prefixe(Arbre A) {
    Visiter(A)
    Si Non_vide(gauche(A))
        VisiterPrefixe(gauche(A))
    Si Non_vide(droite(A))
        VisiterPrefixe(droite(A))
}
```

PARCOURS POSTFIXE

```
Visiter Postfixe(Arbre A) {
    Si Non_vide(gauche(A))
        VisiterPostfixe(gauche(A))
    Si Non_vide(droite(A))
        VisiterPostfixe(droite(A))
    Visiter(A)
}
```

PARCOURS INFIXE

```
Visiter Infixe(Arbre A) {
    Si Non_vide(gauche(A))
        VisiterInfixe(gauche(A))
    Visiter(A)
    Si Non_vide(droite(A))
        VisiterInfixe(droite(A))
}
```

Exemple :

Figure 8

R= Racine G=Gauche D =Droite

- Parcours infixe (GRD): 4, 2, 7, 5, 8, 1, 3, 9, 6
- Parcours postfixe (GDR): 4, 7, 8, 5, 2, 9, 6, 3, 1
- Parcours prefixe (RGD): 1, 2, 4, 5,7, 8, 3, 6, 9

5. Les autres types d'arbres et méthodes
   a. Arbre n-aire

Un arbre est un graphe sans cycle ou les nœuds sont relier par des arrêtes.

On distingue 3 sortes de nœud : Les nœuds racine, les nœuds feuille et les nœud internes.

Un arbre dans lequel le nœud a trois fils est un arbre ternaire.

La définition des mots profondeur d'un nœud, hauteur d'un arbre, la taille d'un arbre et le dégrée d'un nœud est le même qu'en cas d'un arbre n-aire.

Le dégrée d'un nœud est égale au nombre de ces fils.

Le dégrée de l'arbre est égale au degré du nœud le plus grand.

Exemple :

Figure 9

b. Méthode pour stocker les arbres binaires

Les arbres binaires peuvent êtres construite de différente manière. Dans un langage avec structure et pointeur, les arbres binaires peuvent être conçu en ayant une structure a 3 nœuds qui contienne quelque donnée et des pointeurs vers son fils droit et son fils gauche. Parfois, il contient un pointeur vers son unique parent.

Si un nœud possède moins de 2 fils, l'un des deux pointeurs peut être affecter à la valeur spéciale null.

- Avantages :
  - Pas de gaspillage de mémoire
  - Concu pour contenir un nombre variable de nœud

- Inconveniants :
  - Il n'y pas d'accès directe a un nœud de l'arbre
  - L'implémentation est délicate a réaliser quand on est débutant

Figure 10

c. Méthode pour ranger les arbres binaires en tableau :

Les arbres binaires peuvent aussi etre ranger dans des tableaux et si l'arbre est un arbre binaire complet cette méthode ne gaspille pas de la place et la donnée structure résultante est appeler un TAS.

Dans cet arrangement compact, un nœud a un indice i et ces fils se trouve au indice 2i + 1 et 2i si l'indice commence par 1. Et 2i + 1 et 2i + 2 si l'indice commence par 0.

Figure 11

d. Méthode de rotation simple(MRS)

Une rotation est une modification local d'un arbre binaire. Elle consiste a échanger un nœud avec l'un de ces fils.

Dans la rotation à droite, un nœud droite devient le fils droit du nœud qui était son fils gauche.

Dans la rotation gauche un nœud devient le fils gauche du nœud qui était son fils droite.

Les rotation gauche et droite son inverse l'une de l'autre. Elle sont illustrer par la figure ci-dessous ou les triangle designe les sous arbre non vide.

Figure 12

Exercice de cours

Sur l'arbre ci-dessous effectuez

1. Une rotation droite sur 3
2. Une rotation gauche sur 10
3. Une rotation droite sur 8

Figure13

6. Arbre AVL

La dénomination arbre AVL provient des noms de ces inventaires Russe Georgy Adelson-Velsky et Eugeni Landis qui l'on publie en 1962. Les arbre AVL on été historiquement les premiers arbre binaire de recherche automatiquement equilibrer.

Dans un arbre AVL, les hauteurs des sous-arbres d'un meme nœud differs au plus de un.

La recherche, l'insertion et la suppresion sont toute en θ(log(n)).

L'insertion et la suppression neccessite d'effectuer des rotation :

Arbre AVL :=> |HSD - HSG| <= 1 (:=> = implique)

Exemple : figure 14

7. LE TAS

On dit qu'un arbre Binaire complet est ordonnee en TAS lorsque la propriete suivante est verifier pour tous les nœuds de l'arbre.

Etiquette(pere) >= Etiquette(fils)

Cette propriete implique la grande etiquette est situer a la racine du TAS. Ils ont ainsi tres utiliser pour implementer les fils a priorite car il permettent des insertion en temps logarithmique et un acces direct au plus grand element.

Exemple : Figure 15

8. Table Hachage

Une table de hashage est une structure de donnee permettant d'associer une valeur a une clee. Il s'agit d'un tableau ne comportant pas d'ordre. L'acces un element se fais en transformant la clee en une valeur de hachage. Le hachage est un nombre qui permet la localisation d'un element dans le

tableau. Typiquement le hachage est l'indice d'un element dans le tableau. Une case dans le tableau est appeler alveole. Differente operation peuvent etre effectuer sur une table de hachage :

- Creation d'une table de hachage
- Insertion d'un nouveau couple cle-valeur
- Suppression d'un element
- Recherche de la valeur associer a une clee(dans l'exemple de l'annuaire telephonique retrouver le numero de telephone d'une personne)
- Desruction d'une table de hachage pour liberer la mémoire occuper.

Tout comme les tableau en generale, les table de hachage permettent un acces a θ(1) en moyenne quelque soit le nombre d'element dans la table. Toutefois le temps d'acces dans le pire des cas peut etre θ(n).

La position des elements dans une table de hachage est aleatoire. Cette structure n'est donc pas adapter pour acceder a des donnee pyramide.

Exemple de table de hachage implememtant un annuaire telephonique

Figure 16

9. Les Arbres libres

Sot G =(S,A) un graphe orienter non vide. Les conditions suivantes sont equivalente :

- G est un arbre libre
- 2 nœuds quelconque de S sont connecter par un unique chemin simple.
- G est connexe mais ne l'est plus si on retire un arc qlqonq
- G est sans circuit mais ne l'est plus si on ajoute un arc qlqonq
- G est connexe et Card(A) =Card(S)
- G est sans circuit et Card(A) = Card(S) – 1

S est l'ensemble des nœuds ou sommets et A l'ensemble des arretes ou arcs.

Exemple :

Figure 17

10. Les arbres enracines

Un arbre enraciner est un arbre libre minu d'un nœud distinguer appeler la racine de l'arbre.

Si on change la racine d'un arbre enraciner, on obtient encore un arbre enraciner.

Exemple :

Figure18

11. Les arbres enracines en JAVA
    a. La suite des fils est representee par une liste

```
class Arbre {
        int contenu;
        ListeArbre suivant;
}

class ListeArbre {
    Arbre contenu;
    ListeArbre suivant;
}
```

b. La suite des fils est representee par un tableau

```
class Arbre {
    int contenu;
    Arbre [] fils;
}
```

12. Les arbres a lettres

Un arbre a lettres est une manière compact de representer un ensemble de mot(lexique). L'idee est de regrouper tout les mots en un arbre dont chaque arc est une lettre. Un mot est representer par un chemin de la racine a un nœud contenant la valeur « Fin de Mot ».

Les rond (o) sont des nœud de fin de mot.

Ex : Soit les mots suivant :

- SYNDRONE
- SYNCHRONE
- SYNTHAXE
- SOMMAIRE
- SOMMET
- SOT

L'arbre de ces mots est representer comme suit :

Figure19

13. La representation et la manipulation d'expression Arithmetiques

(Les nombres sont les feuilles dans l'expression et on utilise que les rangs pour designer)

- Le tableau de l'Arbre : 3*(4-6)

  Arbre => Tableau

  Tableau => Arbre

  Expression => Tableau => Arbre

N*	Contenu	G	D
1	*	2	3
2	3	0	0
3	-	4	5
4	4	0	0
5	6	0	o

On peut representer une expression arithmétique par un arbre dont la racine contient un operateur et le sous-arbre par un operant.

*Evaluation de l'expression ci-dessus consistera remplacer

L'evaluation de consiste a remplacer sous-arbre par la valeur de l'expression qu'il represente. Pour l'arbre ci-dessus on évalue d'abord le sous-arbre (-,4,6) qui donne (-2) puis (*,3,2) qui donne (-6).

Exercice du cours :

Soit l'expression arithmétique suivante :

((5+2)*(2-1))

Solution :

Prefixe : /,*, +, 5, 2, -, 2, 1, +, +, 2, 9, *, -, -, 7, 2, 1, 8.

Infixe : 5, +, 2 , *, 2, -, 1, /, 2, +, 9, +, 7, -, 2, -, 1, *, 8.

14. ABR : Adjonction aux feuilles et a la racine

On compare l'élément a la racine pour savoir si l'ajout ce fera dans le sous arbre gauche ou droit et on rappelle la procédure récursivement.

Le dernier appel récursif se fait sur un arbre vide. Et on a alors a cette place le nœud contenant l'élément a ajouter. On parle de l'Adjonction des feuilles.

Concernant l'adjonction a la racine, on peut ajouter un élément a n'importe quel niveau en particulier a la racine.

L'adjonction a la racine peut présenter un intérêt si on desire priviligier l'acces au dernier element.

On coupe l'arbre A en A1 et A2 ; tels que A1 contienne les element inferieurs a x et A2 les elements superieurs puis on construis l'arbre.

Remarque : On ne visite que les nœuds situer sur le chemin suivi lors de la recherche de x a partir de la racine

Exemple de code illustant la dermarche :

```
void coupure (item x, abr a, abr &G, abr &D){
    abr X, Y;
    if(A == NULL) then {
        coupure(X, d(A), X, Y);

  } else {
    coupure (x, g(A), X, Y);
    g(A) = Y; G = X; D = A;
  }
}
```

15. Type de donnée abstraite pour un arbre

- Principe

Les donnee sont associees au nœud d'un arbre. Les noeud sont accessibles les uns aux autres selon leurs position dans l'arbre.

- Interface

Pour un arbre T et un nœud n, nous devons retenir ce qui suit :

(C'est le rang on utilise)

- PARENT(T, n) : Renvoie le parent d'un nœud n(signal une erreur si n est a la racine)
- ISEMPTY(T) : Renvoie vrai si l'arbre est vide.
- CHILDREN(T, n) : Renvoie une structure de donnee contenant des fils du nœud n
- ISROOT : Renvoie vrai si n est la racine de l'arbre
- ISEXTERNAL(T, n) : Renvoie vrai si n est un nœud externe
- ISINTERNAL(T, n) : Renvoie vrai si n est un nœud interne
- GETDATA(T, n) : Renvoie les donnees associer au nœud petit n.
- ROOT(T) : renvoie le nœud racine de l'arbre.
- SIZE(T) : Renvoie le nombre de nœud de l'arbre
- LEFT(T, n) : Renvoie le fils gauche de n
- RIGHT(T, n) : Renvoie le fils droit de n

Exercice d'application

- PARENT(T, 6) : 3
- ISEMPTY(T) : Faux
- CHILDREN(T, 3) : 6, 7, 8, 9
- ISROOT(T, 2) : Faux
- ISEXTERNAL(T, 8): Vrai
- ISINTERNAL(T, 7): Faux
- GETDATA(T, 4) : 4

- ROOT(T) : 1
- SIZE(T) : 9
- LEFT(T, 6) : 8
- RIGHT(T, 6) : 9

16. Les arbres en Python

Python ne dispose pas un standard de type permettant de representer les arbres. Nous mettons a disposition le type arbre qui permet de realiser les operations ci-dessous :

- Un objet de type arbre n'est pas vide il contient au moins une racine
- a.racine est le label de la racine de l'arbre
- a = Arbre(v, fils = l) crée l'arbre ayant une racine v qui a pour fils les arbres de la liste l

() => un sous-arbre

Exemple : a = Arbre (5, 3, 4, (6, 7, 8))

Figure 21

- Un arbre est iterable et on peut don ecrire :

```
for f in a: print (f.racine)
```

- Len(a) donne le nombre de petit fils de a
- a[0] donne une reference vers le premier fils de a

Exercice d'application :

Soit un arbre de racine 5. La racine a 3 fils : 3, 4 et 6.

Le fils 6 a 02 fils 7 et 8

1/ Representer l'arbre en question en Python.

2/ Remplacer le sous droit par l'arbre 7, 8 et 9

Solution

1/ a = Arbre(v, fils=l)

=> a = Arbre(5, 3, 4, (6, 7, 8))

2./ a.remplace(5, 3, 4, (7, 8, 9))

CH-III/ STRUCTURE DE DONNEES LINEAIRES LISTES, PILES, FILES

1. Introduction

Le but de ce chapitre est de décrire les représentations de structures de donnée de base tel que les listes en générale et deux forme restrainte les piles et les files.

L'autre but rechercher de voir l'importance de ces structures a travers quelque exemple d'application.

2. Les listes

Les listes sont des structures de donnee informatique qui permettent au meme titre que les tableaux par exemple de garder en mémoire des donnees en respectant une certaine ordres.

On peut enlever, ajouter ou consulter un element en debut ou en fin de liste, vider une liste ou savoir si elle contient un ou plusieurs elements.

Figure1

- Quelque operations sur les listes
    - Tester si la liste est vide
    - Acceder au k-ieme element de la liste
    - Inserer un nouvelle element derriere le k-ieme
    - Fusionner 2 listes
    - Rechercher un element d'une valeur particuliere
    - Trier une liste

a. Implementation des listes

Il existe plusieurs methode pour implementer les listes, les plus courant sont l'utilisation de tableau et de pointeur.

- Utilisation de tableau

Implementer une liste a l'aide des tableaux n'est pas tres compliquer. Les elements de la liste sont simplement ranger dans le tableau a leur place respective. Cependant l'utilisation des tableaux possède quelque inconvenants :

    - La dimension d'un tableau doit être défini lors des déclarations et ne peut donc pas etre modifier dynamiquement lors de l'execution d'un programme.
    - Le tableau etant surdimensionner, il encombre en general la mémoire de l'ordinateur
    - Si la taille maximale venais a etre augmenter, il faudrais modifer le programme et recompiler.

- Utilisation de pointeur

Les pointeurs definissent une adresse dans la mémoire de l'ordinateur, adresse qui correspond a l'emplacement d'une autre varirable.

Il est possible a tout moment d'allouer ce espace dynamiquement lors de l'execution du programme.

```
int y = 5;
   int *p;
   p = &y;
   printf("%d", *p);
```

- Liste chainee, utilisation pointeur s

Figure 2

b. La représentation chaine d'une liste

Le principe est de representer chaque element de la liste a un endroit quelconque de la mémoire.

Figure 3.

c. Variante utiles d'une liste

*Liste circulaire

On a une liste L = <a1, a2, ….., an-1> dont le suivant du dernier du an-1 eestest premier a1

*Liste doublement chainee

Elle sont utile quand on veut acceder facilement au predecesseur d'un element de la liste.

d. Les fonctions

Ensemble dynamique d'objet ordonnee, accessible relativement les un aux autre **se base de leurs positions**. Les differente fonction utiliser sont :

- INSERT-BEFORE(L,p,x): insert x avant p3 dans la liste L
- INSERT-AFTER(L,p,x) : insert x apres p dans la lsite L
- REMOVE(L, p) : Retire l'element a la position p
- REPLACE(L,p,x) : Remplace par x l'objet situer a la position p

- FIRST(L) : Renvoie la premiere position dans la liste.
- LAST(L) : Renvoie la derniere positon dans la liste.
- PREV(L, p) : Renvoie la position precedant p dans la liste
- NEXT(L, p) : Renvoie la positon suivant p dans la liste

Exercice d'application :

Soit la liste L = <1, 8, 3, t, 14, 20, y>

La position commece par 0

- INSERT-BEFORE(L,p,x): Impossible
- INSERT-AFTER(L,p,x) :
- REMOVE(L, 3) : 1831020y
- REPLACE(L,3,Z) : 183Z1420y
- FIRST(L) : 1
- LAST(L) : y
- PREV(L, 3) : 3
- NEXT(L, 1) : 3

3. Pile

En informatique, une pile est une structure de donnee fonder sur le principe LIFO(Last in First out) ce qui veut dire que eles elements arrve ou ajouter a la pile serons les premier a etre reccuperer. Le fonctionnement est donc celui d'une pile d'assiette : On ajotue des assiette sur la pile et on les reccupere dans l'ordre inverse en commencalnt par le dernier.

a. Primitive :

Voici les primitives communement utiliser pour manier les piles :

- « Empiler » : Ajoute un element sur la pile en anglais PUSH
- « Depiler » : Enleve un element de la pile et le renvoie en anglais POP
- « Vide » : Renvoie vrai si la pile est vide faux sinon
- « Remplissage » : Renvoie lenombre d'element dans la pile

b. Application

La notion d epile est utiliser dans plusieur domaine.

Dans un navigateur web :

- Une pile sert a memoriser les pages web visiter, l'adresse de chaque nouvelle page visiter est empiler et l'utilisateur depiile l'adresse de la page precedente en cliquant sur le bouton afficherf la page precedente.
- L'evlution des expression mathematique en notation post-fixe utilise une pile
- La fonction « annuler la frappe » d'un traitement de texte comme word memorise la la modification apporter au texte dans une pile.
- Un algorithme de recherche en profondeur utilise une pile pour memoriser les nœud utliser

c. Representation d'une pile par un tableau

La representation d'une pile par un tableau a pour avantage :

- Facile car on ne modifie une pile que par un bout.
- Les operation sont facile mais l' inconveniant est que la hateur est borner et donc il faut une allocation statique de la mémoire.

d. Implementation par un tableau

S est un tableau qui contient les elements de la pile.

S.top est la position courant de l'element au somet de S. On a alors :

PUSH(S, x)	POP(S)

`if S.top == S.length`
`  error "overflow"`
`else`
`  S.top = S.top + 1`
`S.[S.top] = x`
	`if SATCK.EMPTY(x)`
	`  error "underflow"`
	`else`
	`  S.Top = S.Top + 1`
	`return S.[S.Top + 1]`

4. Les files

Une file est un structure de donnee basee sur le principe premier entree premeir sort ou enanglais FIFO(First In First Out).

Ce qui veut dire que les premier elemetn ajouter a la liste sont les premier a etre reccuperer.

Le fonctionement ressemble a une file d'attente ; les premeire personnes arrive seront les premieres a sortir.

a. Primitive

Voici les primitives communement utiliser pour manier les files.

- « ajoute »Ajoute un element dans la file en anglais « Enque »
- « enlever » : Renvoie le prochain element de la file en anglais « Deque »
- « vide » : Renvoie vrvai si la file est vide, faux sinon.
- « remplissage » : Renvoie le nombre d'element dans la file.
    - File = horizontal
    - Pile = vertical

Exercice d'application Telegram

b. Application

En generale, on utilise les files pour momoriser temporairement les transactionns qui doivent attendre pour etre traiter.

- Les serveurs d'impression qui doivent traiter les requettes en fonction qu'elle arrive et les insert dans une file d'attente
- Un algorithme de parcours en largeur utilise une file pour memoriser les nœuds visiters
- On utilise aussi les files pour créer une sorte de mémoire tempons

c. Representation d'une files par une liste chainees

Il est facile d'implemeter les 04 operations ci-dessus mais on perd la place due au pointeur.

d. Implémentation a l'aide d'un tableau

Θ est un tableau de taille fixe Θ.lengh. Mettre plus Θ.lengh element dans la file provoque une erreur de deplacement .

- Θ.head : Est la position a la tete de la file
- Θ.tail : est la premiere position vide a la fin de la file
- Initialement : Θ.ead = Θ.tail

5. Structure contigue

Un tableau est une liste contigue d'element. Chaque element est localliser efficacement par son indice ou adresse. Le temp d'acces est constant lorsque l'indice est connu.

La localiter permet d'exploiter les memoires caches rapide.

6. Structures chaines : Pointeur et liste chainees

Pointeur represente une adrrese d'un element

Notation P = pointeur

P↑ : acces a l'element a l'adresse P (&P)

Une liste chainee est une liste dont l'ordre des elements est determiner par le pointeur liant un element a son successeur.

TYPE Liste F = (el : elt, suivi : Liste).

TAF :

Rechercher un element chainee. Les algorithmes sont :

AlgoRechSeq

entree: p: liste P

e: elt

Sortie: In boolean

Voisin recursive

if(P== null) then

In <- false else

if(e == p|^.el) then

In <- true else

End <- RechSeqCh(P|^.suiv, l);

7. Structure

	Cas non trie			Cas trie		
	Tableau	Liste chainee	Liste 2x chainee	tableau	Lsite chaine	Liste 2x chainee
Rechi(E, k)	Θ(1)			Θ(1)		
-Reche(E, k)	Θ(n)	Θ(n)	Θ(n)	Θ(log(n))	Θ(n)	Θ(n)
Inser(E, e)	Θ(n)	Θ(1)	Θ(1)	Θ(n)	Θ(n)	Θ(n)
Supp(E, x)	Θ(1)	Θ(n)	Θ(1)	Θ(n)	Θ(1)	Θ(1)
Succ(E, x)	Θ(n)	Θ(n)	Θ(n)	Θ(1)	Θ(1)	Θ(1)
Pred(E, x)	Θ(n)	Θ(n)	Θ(n)	Θ(1)	Θ(n)	Θ(1)
-Min(E)	Θ(n)	Θ(n)	Θ(n)	Θ(1)	Θ(1)	Θ(1)
-Max(E)	Θ(n)	Θ(n)	Θ(n)	Θ(1)	Θ(n)	Θ(1)

Terminologie: Cle de recherché # element

Hypothese: Cle de recherché : elemen

k : indice tableau

e : element

8.

En python les liste sont des objets qui peuvent en contenir d'autre. Ce sont des seuences comme les chaines de caractere. Mais au lieu de contenir des caracteres elle peut contenir n'importe qu'elle objet. Comme d'habitude on va s'occuper des liste avant de voir tout ces inerets.

a. Creation de listse en Python

On a 2 moyen de cree des liste. La classe d'une liste et par la liste en declaration.

>>> = prompt

>>>ma_list = lsit() #on cree une lsite vide

>>> Type(ma_list)

<class 'list' >

>>> ma_list

[]

>>>

b. Creation d'une listenon vide

```
>>>ma_lis = [1,2,3,4,5]
>>> print(ma_list)
>>>[1 2 3 4 5]
```

c. Inserer les objets dans une liste

Exercie

Examen 2018

Exercice :

Soit la liste L = <0, 1, 2,3,4,5,6,7,8,9>

1. Utiliser deux autres structure avec lecture Ecriture pour trie les 10 chiffres par ordre croissant
2. Proposer un TAS de niveau 3
3. Proposer un ABR avec les 10 chiffres
4. Soit la liste A=<a, b, c> concatener L et A

d. Concatenation de liste

On peut egalement agrandire les liste en les avec d'autre

```
>>> ma_list1 = [3,4,5]
>>> ma_liste2 = [8,9,10]
>>> ma_list1.extend(ma_liste2)
>>> print(ma_list1)
[3, 4, 5, 8, 9, 10]
```

ou

```
>>> ma_list1 + ma_liste2
[3, 4, 5, 8, 9, 10, 8, 9, 10]
```

e. Suppression d'un element d'une liste

On peut utiliser del pour supprimer les element d'une sequence comme une liste.

```
>>> ma_list = {[-5,-2,1,4,7,10]
>>> del ma_list[0] #Supp 1ere cellule
>>> ma_list
```

[-2, 1, 4, 7, 10]

f. Le parcours de liste : Boucle TANT QUE

```
>>> ma_list = ['a', 'b', 'c','d', 'e', 'f', 'g', 'h']
>>> i=0
>>> while i <len(ma_list) :
```

CH IV- LES COLLECTIONS EN JAVA

1. Structure de donnees

C'est l'organisation efficace d'un ensemle de donnee sous la forme de tableau, de liste, de pile etc … Cette efficaciter reside dans la quantiter mémoire utiliser pour stocker les donnees et le temps neccessaire pour realiser des operations sur ces donnees.

2. Collection et JAVA

Une collection gere un groupe d'un ensemble d'objet d'un type donnee. Ou bien c'est un objet qui sert a stocker d'autre objet. Dans les premieres version de Java, les collections étaient representer par les « Array », « Vector », « Stack ». Puis avec java.2 et java2 est apparu le framework de collection qui tout en gardant les principes de bases, il a apporter des modification dans la manière avec laquelle ces colection on été realiser et hierachiser.

Tout en collabarant entre elle ces colection permettent de realiser dans des cactegorie des conceptions reutilisable.

3. Collection Framework JAVA
    1. Interfaces

Figure 1.

Les interfaces sont donc organiser en deux catégories : Collection et map.

- Collection : On groupe d'objet ou^ la duplication peut etre autoriser.

- Set : Ensemble ne contenant que des valeurs et ces valeurs ne sont pas dupliquer. Par exemple l'ensemble A={1,2,4,8} aucun element n'est dupliquer dedans. Set herite donc de Collection mais n'autorise pas la duplication. SortedSet est un Set trier

- List : herite aussi de collection mais autorise la duplication. Dans cette interface un système d'indexation a été introduit pour permettre l'acces rapide au element de al liste.

- Map : est un groupe de pair contenant une clee et une valeur associer a cette cle : Cette interface n'herite ni de Set ni de collection. La raison est que Collection traite des objets simple et Map des objets composer. SortedMap est un Map trier.

    2. Implementation : Framework

Le framework fournit les informations suivant des differentes interfaces :

	Classes d'implementation				
		Table de hachage	Table de taille variable	Arbre balancer	Liste chainee
Interface	Set	HashSet .		TreeSet.	
	List		ArrayList		LinkedList
	Map	HashMap.		TreeMap.	

Par contre il n'y pas d'implementation de l'interface collection.

Pour Set et Map, l'implementation est soit sous la forme d'une table de Hachage (HashSet et HashMap) ou bien sous la forme d'un arbre(TreeSet et TreeMap). Pour la liste, soit sous la forme de tableau ArrayList ou liste chainee LinkedList

4. Les algorithmes

Ils sont utiliser pour traiter les elements d'un ensemble de donnee. Il definisse une procedure informatique(le Tri, la recherche, etc).

5. Interateurs

Il fournissent au algorithme un moyen pour parcourir une collection du debut a la fin. Ce moyen permet de retirer a la demande des elements donnee de la collection.

6. Les operations mathematique des collections

Les collections sont vu comme des ensembles et realise les 3 operations de mathematiques sur ces ensembles.

- UNION : add et addAll
- INTERSECTION : return All
- DIFFERENCE : remove et removeAll.

7. Codification : Exemple de code

```
import java.util.*;

public class SetExemple{
  public static void main (String []args){
    Set set = new HashSet();
    set.add("Bernadine");
    set.add("Elizabeth");
    set.add("Gene");
    set.add("Clara");
```

```
    System.out.println(set);
    Set SetTree = new TreeSet(set);
    //Un set triee
    System.out.println(SortedSet);
  }
}
```

```
public interface Lsit extends Colection{
  //Positionnal
  Object get (int index);
  Object set (int index, Object element); // optional
  void add (int index, Object element); // optionnal
  Object remove(int index); //optinal
  boolean addAll(index, Collection c); //optinal
  //Search
  int indexOf(Object o);
  int lastIndexOf(Object o);
  //Iteration
  LsitIterator listIterator();
  ListIterator listIterator();
  //Range-view
  List subList (interator(int index))
  }
}
```

Devoir chapitre 1 et chapitre 4 pour les question de cours.

Execices sur les arbres.