**6 – Exemple de structures de données linéaire implémentées avec des tableaux ou des listes. Applications**

# Structures linéaires

## Les tableaux

* Définition :

Généralement, n case de même taille pour stocker un même type

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  |  | … |  |

* Taille = nombre d’éléments / de cases
* Accès aux éléments : rapide car taille de case fixé. L’accès se fait avec un saut de mémoire au partir du début du tableau jusqu’à l’indice i\*taille d’une case. Cela ce fait donc d’une manière arithmétique.
* Mémoire contigüe : les cases du tableau sont les unes à la suite des autres dans la mémoire.

## Les listes chainées

* Définition

|  |
| --- |
|  |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
|  |
|  |

**NONE**

* Mémoire avec indirection : on suit un chemin
* Longueur : calcul en parcourant toute la liste : O(n)
* Accès aux éléments : plus pénible, parcours de la liste
* Mémoire discontinue et sauts aléatoires

## Opérations sur les structures linéaires

|  |  |
| --- | --- |
| **Mémoire contigüe** | **Mémoire avec indirection** |
| **Concaténation** | |
| Arithmétique sur les adresses possible   |  |  |  |  | | --- | --- | --- | --- | |  |  | … |  |   t1 🡪  t2 🡪   |  |  |  |  | | --- | --- | --- | --- | |  |  | … |  |   En supposant qu’il y ait de la place en mémoire derrière t1, on recopie t2 à la suite sinon on recopie la concaténation à un autre endroit mémoire.   |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | |  |  | … |  |  |  | … |  |   *Au pire : O(len(t1)+len(t2))* | |  | | --- | |  | |  |   Pas d’arithmétique -> on stock les adresses  **NONE**   |  | | --- | |  | |  |   l1 🡪   |  | | --- | |  | |  |  |  | | --- | |  | |  |  |  | | --- | |  | |  |  |  | | --- | |  | |  |   O(len(l1))  **NONE** |
| **Ajout en tête** | |
| |  |  |  |  | | --- | --- | --- | --- | |  |  | … |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | |  |  |  | … |  |   t1 🡪  Complexité : O(len(t1)) | |  | | --- | |  | |  |   l1 🡪     |  | | --- | |  | |  |   Complexité : O(1)   |  | | --- | |  | |  |  |  | | --- | |  | |  |   **NONE** |
| **Ajout en fin** | |
| |  |  |  |  |  | | --- | --- | --- | --- | --- | |  |  | … |  |  |   t1 🡪  Complexité : O(1)  Si recopie : O(len(t1)) | |  | | --- | |  | |  |   l 🡪   |  | | --- | |  | |  |  |  | | --- | |  | |  |  |  | | --- | |  | |  |   **NONE**  **NONE**  Complexité : O(len(l)) |
| **Suppression en tête / milieu** | |
| |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | |  |  |  | … |  |  |   t1 🡪  t2 🡪   |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | |  |  | … |  |  |  |   Complexité au pire : O(len(t1)) | |  | | --- | |  | |  |   l1 🡪   |  | | --- | |  | |  |  |  | | --- | |  | |  |   **NONE**  Complexité en tête : O(1)  au milieu : O(i) |
| **Suppression en fin** | |
| |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | |  |  |  | … |  |  |     t1 🡪   |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | |  |  |  | … |  |  |   t1 🡪  Complexité : O(1) | |  | | --- | |  | |  |   l 🡪   |  | | --- | |  | |  |  |  | | --- | |  | |  |   **NONE**  **NONE**  Complexité : O(len(l)) |

## Complément : les listes Python

Les listes python sont des tableaux d’objets

[ 12, None, [5,2], (« bonjour », 3.14) ]

La liste python stocke les adresses mémoires de ces objets et non les objets en tant que tel.

|  |  |  |  |
| --- | --- | --- | --- |
| AdresseObjet1 | AdresseObjet2 | **…** | AdresseObjetn |

# Piles et Files

## a- Pile

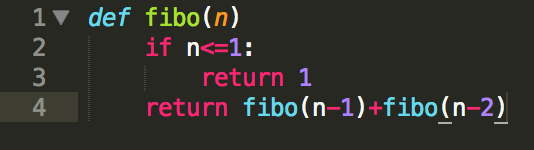
* Définition : Last in, first out
* Pile d’exécutions

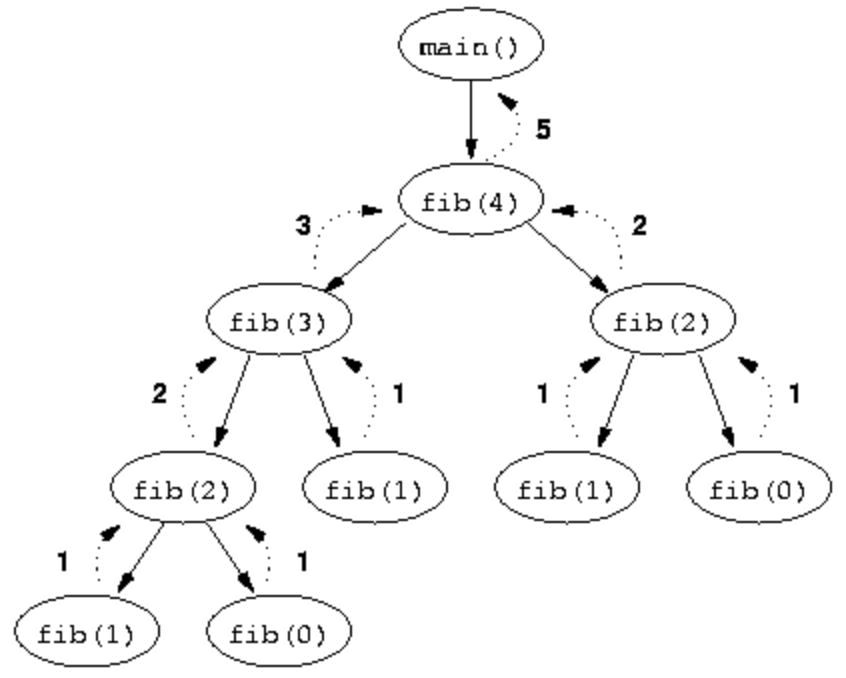
## b- File

* Définition : first in , first out
* Parcours largeurs d’un arbre

# Applications pour les piles et les files

* Double Files d’attente pouvant représenter une queue à un téléski, avec une file « école de ski » et une file « reste de monde ». On place dans le téléski une personne de chaque file, on gère les cas où l’une des deux files est vide.
* Pile d’exécution (ex : Fibonacci )





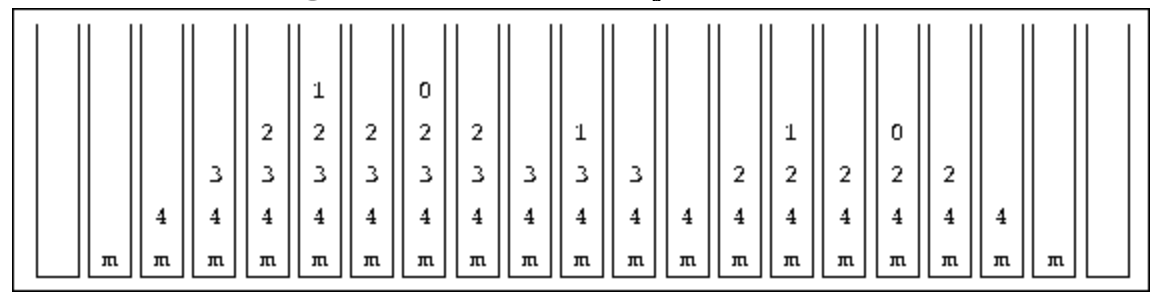


Figure : Pile d'exécution pour la suite de Fibonacci

Figure : Arbre d'invocation de la suite de Fibonacci

* Parcours en largeur d’un graphe avec une file

**x**

**u**

**t**

**v**

**f**

**g**

**h**

**d**

**i**

**b**

**e**

**a**

**Principe :**

On place dans une file l’élément le plus en haut de l’arbre (ici « a »), puis on l’enlève en ajouter dans la file ses enfants de gauche à droite.

|  |
| --- |
| **a**  **a**  **a e**  **a e b**  **a e b d**  **a e b d f**  **a e b d f g**  **a e b d f g h**  **a e b d f g h i**  **a e b d f g h i t u v x** |

|  |  |  |
| --- | --- | --- |
| **e** | **b** | **d** |

|  |  |  |  |
| --- | --- | --- | --- |
| **b** | **d** | **f** | **g** |

|  |  |  |  |
| --- | --- | --- | --- |
| **d** | **f** | **g** | **h** |

|  |  |  |  |
| --- | --- | --- | --- |
| **f** | **g** | **h** | **i** |

|  |  |  |
| --- | --- | --- |
| **g** | **h** | **i** |

|  |  |
| --- | --- |
| **h** | **i** |

|  |  |  |
| --- | --- | --- |
| **i** | **t** | **u** |

|  |  |  |  |
| --- | --- | --- | --- |
| **t** | **u** | **v** | **x** |

* Parcours en profondeur d’un graphe avec une pile

Principe :

On place dans un pile l’élément le plus en haut de l’arbre (ici « a »), puis on l’enlève en ajoutant, de droite à gauche, ses enfants dans la pile. On continue jusqu’à ce que la pile soit vide.

|  |
| --- |
| **f** |
| **g** |
| **b** |
| **d** |

|  |
| --- |
| **e** |
| **b** |
| **d** |

|  |
| --- |
| **t** |
| **u** |
| **d** |

|  |
| --- |
| **b** |
| **d** |

|  |
| --- |
| **h** |
| **d** |

|  |
| --- |
| **d** |

|  |
| --- |
| **i** |

|  |
| --- |
| **v** |
| **x** |

|  |
| --- |
| **a** |

**a**

**a e**

**a e f**

**a e f g**

**a e f g b**

**a e f g b h**

**a e f g b h t u**

**a e f g b h t u d**

**a e f g b h t u d i**

**a e f g b h t u d i v x**

* Calcul d’expression algébrique post-fixé avec un pile

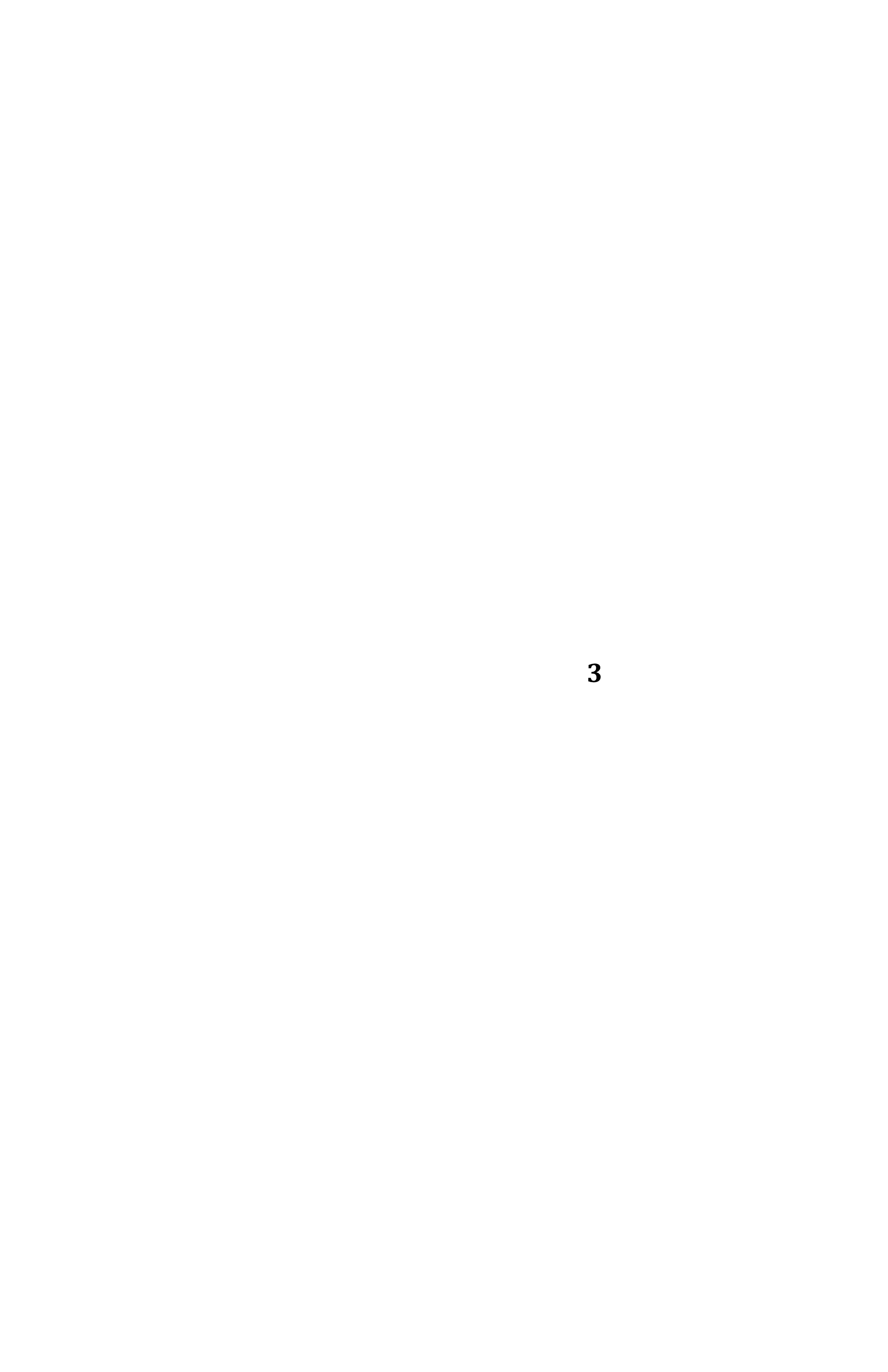
Création d’un graphe pour transformer l’écriture infixée en post fixé

**\***

**+**

**/**

**14**

**7**

**2**

**2**

**3**

**3**

**\_**

**\_**

|  |
| --- |
| **2** |
| **3** |
| **14** |
| **35** |

Ecriture post-fixée :

Principe :

Tant qu’on lit un chiffre, on l’ajoute dans la pile. Quand on lit un opérateur, on fait l’opération sur les 2 nombres en haut de la pile et on ajoute le résultat à la pile tout en retirant les 2 nombres.

|  |
| --- |
| **1** |
| **14** |
| **35** |

**3 - 2**

|  |
| --- |
| **3** |
| **2** |

|  |
| --- |
| **7** |
| **5** |

|  |
| --- |
| **14** |
| **35** |

**35-14**

**14 / 1**

**7 \* 5**

**3 + 2**

|  |
| --- |
| **21** |