Analyse d’algorithmes et programmation

- Les structures de données -

Kahina BOUCHAMA

Université de Versailles, Saint-Quentin en Yvelines

Février 2023

Table des matières

Les structures de données

Une structure de données est une manière particulière de stocker et d'organiser des données dans un ordinateur de façon à pouvoir être utilisées efficacement.

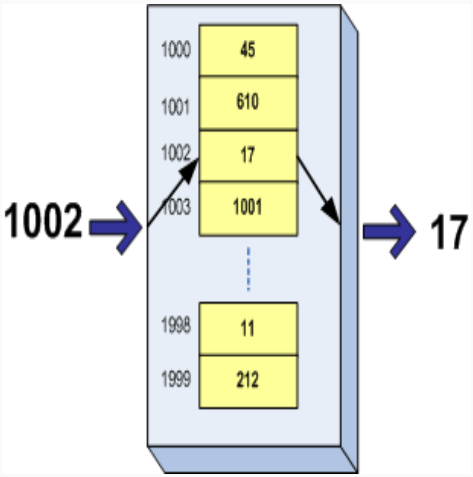
Chaque structure a ses propres avantages et limitations. Nous allons ici voir trois structures de données classiques : les listes chaînées, les arbres binaires et les tables de hachage.

Les opérations usuelles sur les structures de données

- Insertion d'un nouvel élément;
- Suppression d'un élément;
- Recherche d'un élément;
- Affichage d'un ensemble d'éléments;
- Concaténation de deux ensembles d'éléments
- . . .

Notion de pointeurs

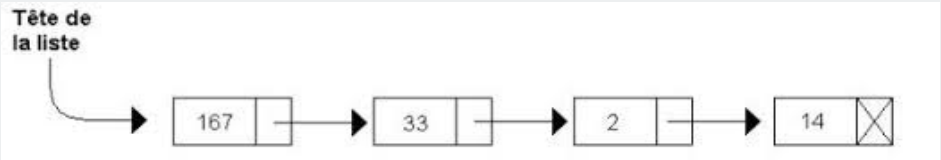
- Toute variable manipulée dans un programme est stockée quelque part en mémoire centrale. La mémoire peut être assimilée à un "tableau", dont chaque élément est identifié par une "adresse".
- Pour retrouver une variable, il suffit alors de connaître l'adresse de la cellule mémoire où elle est stockée. Le type de variables qui permettent de stocker des adresses mémoires s'appelle "pointeur".
- Un pointeur qui ne contient pas d'adresse mémoire est dit pointeur NULL.



Les listes linéaires chaînées(LLC):

Définition:

- Une liste est une collection totalement ordonnée de données. La relation d'ordre ne porte pas sur la valeur des données, mais sur leur position dans la liste.



- Une définition récursive consiste à dire qu'une liste est soit vide, soit composée d'un élément suivi d'une autre liste.

Implémentation d'une liste chaînées

- Une liste linéaire chaînée est représenté par sa tête.
- On appelle tête de liste, le pointeur qui pointe sur le premier éléments de la liste.
- Si une liste est vide, alors la tête de la liste prend la valeur *NULL*.

En algorithmique, un élément d'une liste est représenté par un enregistrement à 2 champs, décrit par:

```
Enregistrement cellule {
   val: entier;
   suivant: ↑ cellule;
}
```

Le champs "val" enregistre la valeur de l'élément et le champs "suivant" est un pointeur qui contient l'adresse mémoire du prochain élément de la liste.

Recherche d'un élément dans une liste chaînée

La procédure *RechercheListe(L, k)* renvoie l'adresse du premier élément trouvé de valeur *k* dans la liste *L*. Si aucun élément de valeur *k* n'existe dans la liste, elle renvoie None.

```
RechercheListe(L, k)
    x = L
    tant que x ≠ None et x↑.val ≠ k faire
       x = x↑.suivant
    renvoyer x
```

Sa complexité: $\Theta(n)$.

Création d'une liste chaînée de taille *n*

Il s'agit d'allouer au fur et à mesure des espaces mémoires pour stocker les éléments à mettre dans la liste. Après chaque allocation d'une case mémoire pour un élément donnée, on passe à l'insertion de l'élément au début, à la fin ou à une position donnée de la liste, initialement vide.

Insertion d'un élément au début d'une LLC

On utilisera la procédure suivante:

```
InsertionListe(L, x)
    nouveau(p) (pour allouer un espace mémoire)
    p↑.val = x
    p ↑.suivant = L
    L = p
    renvoyer L
```

Sa complexité: $\Theta(1)$.

Suppression d'un élément d'une LLC

Pour supprimer un élément *x* connu par sa valeur d'une liste chaînée, on utilise un pointeur qui va parcourir la liste jusqu'à ce que *x* soit trouvé. Il suffit alors de mettre à jour le champs *suivant* de l'élément précédant *x*, en lui affectant l'adresse de l'élément suivant de la cellule contenant *x*.

Dans le pire des cas, la complexité de la procédure de suppression est en $\Theta(n)$.

Suppression d'un élément repéré par sa valeur d'une LLC

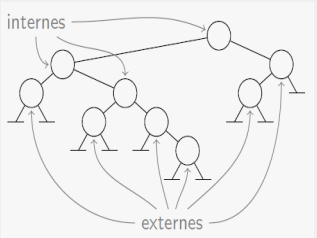
```
SuppressionElementListe(L, x)
    p = L
    Si (p↑.val = x) alors
     L = p↑.suivant
     Supprimer(p)
    Sinon
     Tant que (p↑.val ≠ x) et (p ≠ NULL) faire
      u = p
      p = p↑.suivant
     Si (p ≠ NULL) alors
      u↑.suivant = p↑.suivant
      p↑.suivant = NULL
      Supprimer(p)
```

Les arbres binaires: concepts et définitions

Un arbre est un ensemble de noeuds reliés entre eux par des arêtes. Il vérifie les propriétés suivantes:
1. Il existe un noeud particulier nommé racine.
Tout noeud *c* autre que la racine est relié par une arête à un noeud *p* appelé père de *c*.
2. Un arbre est connexe.
3. Un arbre est sans cycle.
Un arbre binaire est un arbre avec racine dans lequel tout noeud a au plus deux fils: un fils gauche et un fils droit.

Terminologies

- Racine: Un noeud qui n'a pas d'ascendant;
- Bord gauche de l'arbre: le chemin depuis la racine en ne suivant que des fils gauche.
- Bord droit de l'arbre: le chemin depuis la racine en ne suivant que des fils droits.
- Un noeud est dit interne s'il a deux fils non vide. Sinon il est dit externe.



Implémentation d'un arbre

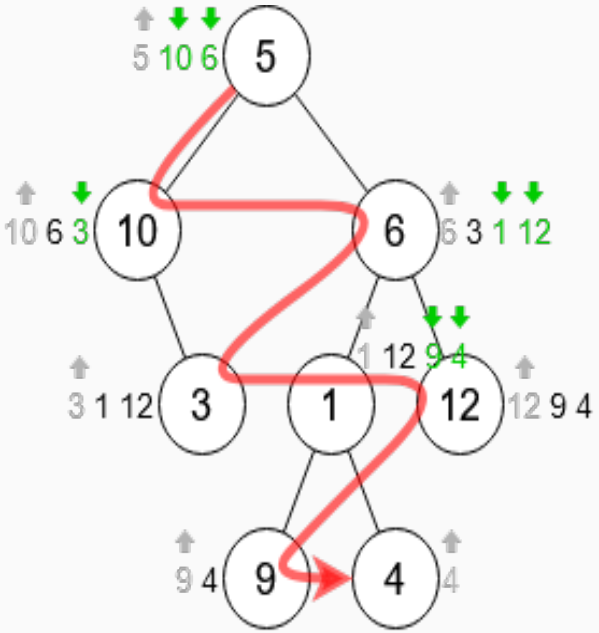
- En algorithmique, un arbre est représenté par sa racine.
- La racine d'un arbre est un noeud que l'on va représenter par l'enregistrement suivant:

```
Enregistrement Noeud {
   val: entier;
   filsgauche: ↑ Noeud;
   filsdroit: ↑ Noeud;
}
```

Parcours d'un arbre

Il existe deux types de parcours:

1. Le parcours largeur d'abord: On parcours les noeuds de la racine vers les feuilles. On traite tous les noeuds d'un même niveau de gauche vers la droite, puis on passe au niveau suivant.



2. Le parcours profondeur d'abord: Les principaux types de parcours en profondeurs sont

   - Le parcours préfixe: la racine d'abord, puis les noeuds les plus à gauche en allant en profondeur. Lorsqu'il n'est plus possible de descendre, on remonte d'un niveau. Sur l'arbre ci-dessus, on obtient la suite $(5, 10, 3, 6, 1, 9, 4, 12)$

   - Le parcours infixe: On traite en premier les noeuds qui n'ont pas de fils gauche. Sur l'exemple précédent, on obtient la suite: $(10, 3, 5, 9, 1, 4, 6, 12)$

   - Le parcours postfixe: On traite les feuilles les plus à gauche en premier, puis on remonte d'un niveau. Sur l'exemple, on obtient la suite $(3, 10, 9, 4, 1, 12, 6, 5)$

Quelques mesures sur les arbres

- **Taille** de l'arbre T, notée $taille(T)$ = nombre de nœuds.
- **Nombre de feuilles** noté $nf(T)$.
- **Longueur de cheminement** de l'arbre T, notée $LC(T)$ = somme des longueurs de tous les chemins issus de la racine.

$$LC(T) = \sum_{x \text{ nœud de } T} h(x).$$

- **Longueur de cheminement externe** de l'arbre T, notée $LCE(T)$ = somme des longueurs de tous les chemins aboutissant à une feuille issus de la racine.

$$LCE(T) = \sum_{x \text{ feuille de } T} h(x).$$

- La **hauteur d'un nœud** n, notée $h(n)$, est la longueur du chemin depuis la racine jusqu'à n.
- La **hauteur de l'arbre** T, notée $h(T)$:

$$h(T) = \max_{x \text{ nœud de l'arbre}} h(x)$$

Quelques mesures sur les arbres

- **Hauteur moyenne** de l'arbre T, notée $HM(T)$= moyenne des hauteurs de tous les nœuds.

$$HM(T) = \frac{LC(T)}{taille(T)}$$

- **Hauteur moyenne externe** de l'arbre T, notée $HME(T)$ = moyenne des longueurs de tous les chemins issus de la racine et se terminant par une feuille.

$$HME(T) = \frac{LCE(T)}{nf(T)}$$