EISTI
Ecole Internationale des Sciences du Traitement de l'Information

Algorithmique et Conteneurs

Le conteneur Arbre

Arbres et Graphes

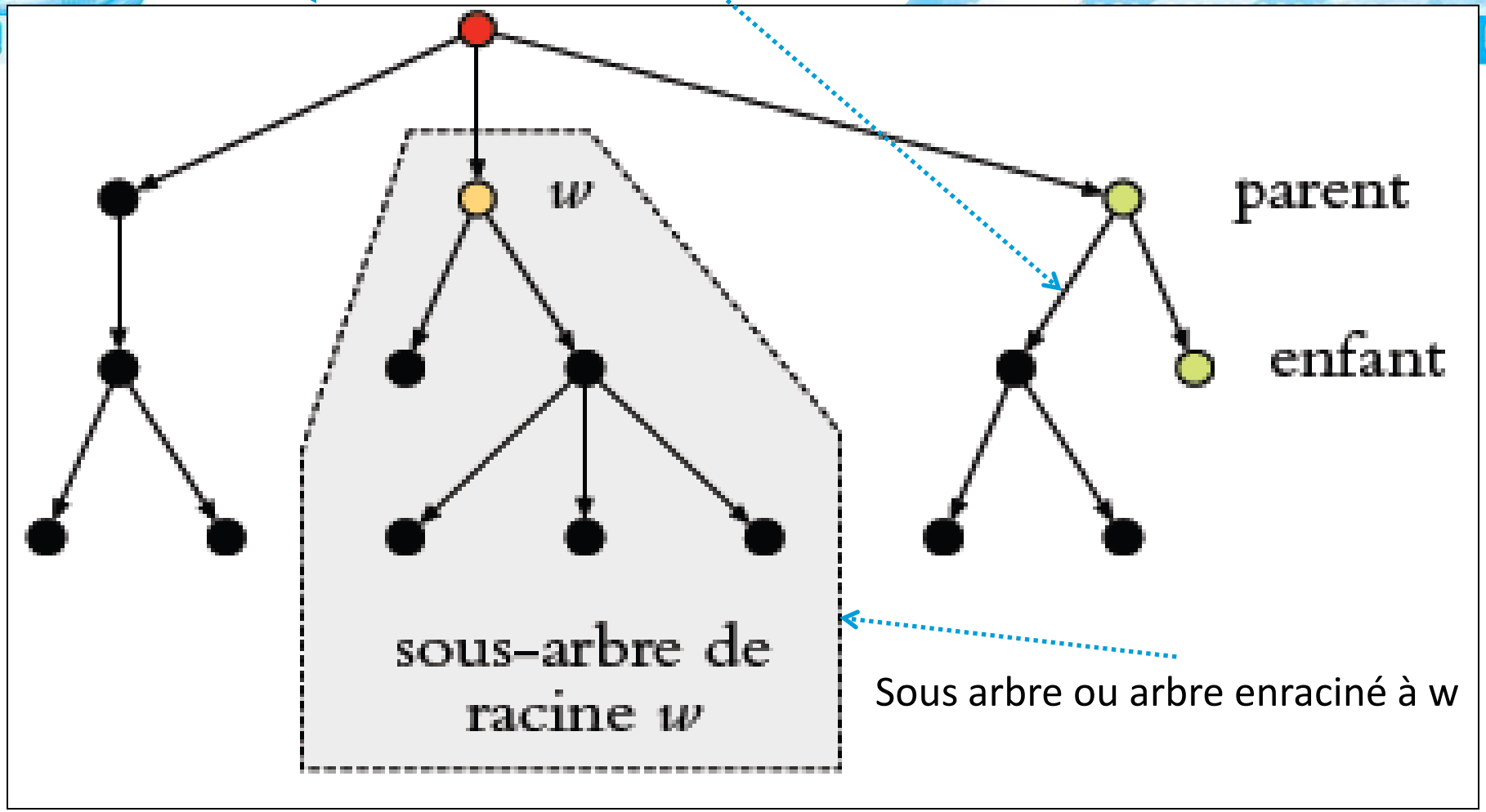
- Arbre libre : C'est un graphe non orienté, connexe et sans cycle.
- Arbre avec racine ou arborescence : C'est un graphe orienté et connexe avec un sommet spécial, la racine, dans lequel il y a 1 chemin simple de la racine à chaque sommet.
- Dans la suite :
  - on étudiera que des arborescencess mais on utilisera le terme arbre;
  - on utilisera le terme nœud à la place du terme sommet.

Utilité des arbres

- De nombreux systèmes sont naturellement des arbres : arbre généalogique, nomenclature, structure de répertoires, …
- Un des principes essentiels de l'algorithmique : Diviser pour mieux régner est une structure d'arbre.
- Evaluation d'expression : les noeuds internes sont des opérateurs et les noeuds externes des opérandes.
- Indexation de documents avec des arbres dits de recherche
- …

Arbre

un arc entre deux nœuds sera une relation dite parent-enfant



Sous arbre ou arbre enraciné à w

Terminologie

- noeud externe ou feuille : c'est un noeud qui n‘a pas d'arc sortant.
- noeud interne : c'est un nœud qui n'est pas externe.
- degré d'un nœud : c'est le nombre d'arc(s) sortants du nœud.
- nœuds frères ou sœurs : des nœuds qui ont le même parent.
- Soit u et w deux nœuds : u est un ancêtre de w si w est dans l'arbre enraciné à u.
- Soit u et w deux nœuds : w est un descendant de u si w est dans l'arbre enraciné à u.

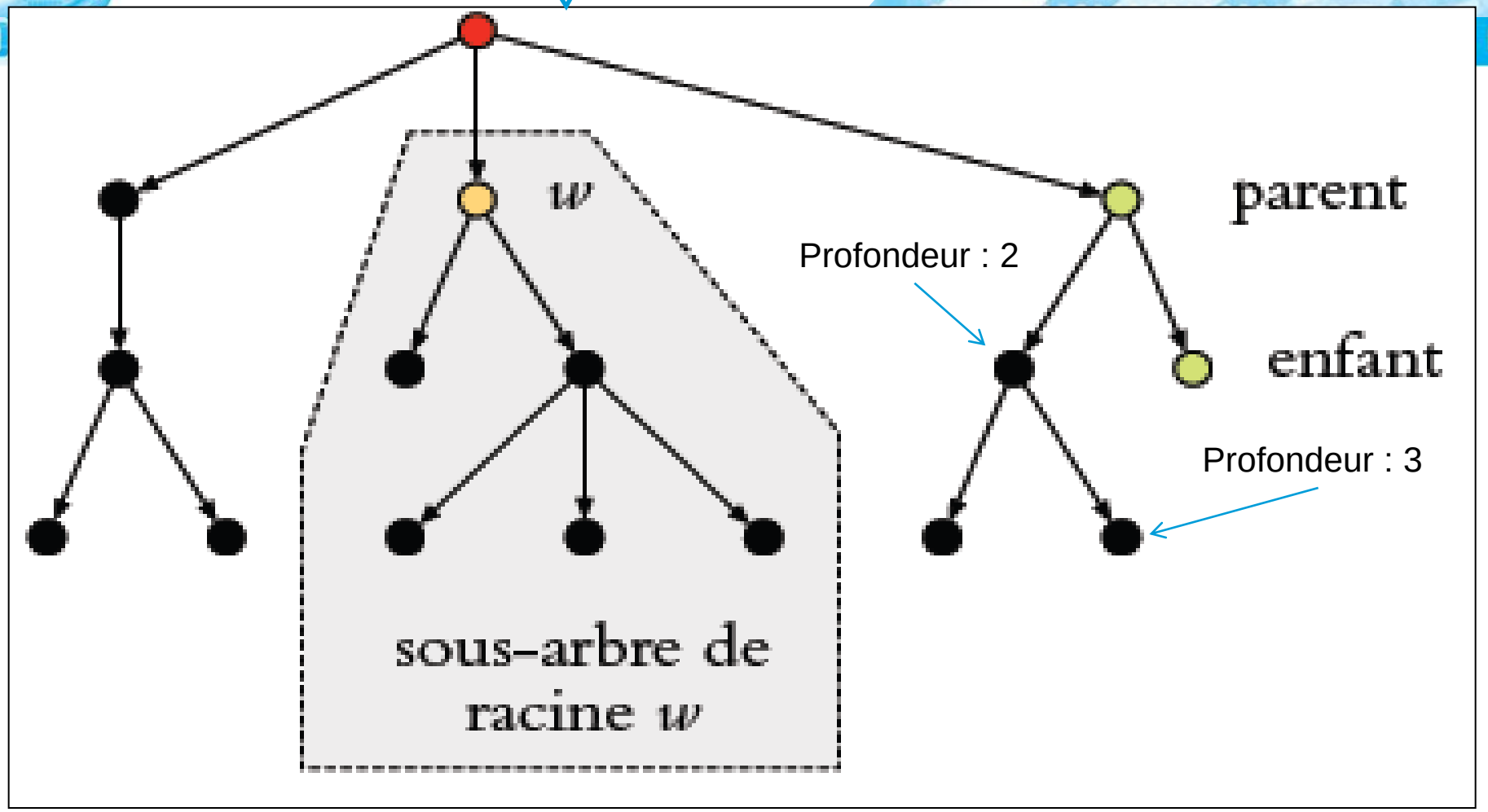
Terminologie

- Arbre ordonné : c'est un arbre dans lequel il existe un ordre entre les enfants des nœuds internes.
- Arbre numéroté :
  - C'est un arbre dont les enfants de chaque nœud sont étiquetés par des entiers positifs distincts.
  - Un arbre numéroté est donc ordonné.
  - i-ème enfant absent : si aucun enfant n'est étiqueté par i
  - Arbre kAire k : c'est un arbre dans lequel les enfants d'un nœud sont étiquetés de 1 à k.
  - Un arbre binaire est un arbre d'arité 2. L'enfant étiqueté 1 sera dit nœud gauche et l'enfant étiqueté 2 sera dit nœud droit

Terminologie

- Profondeur d'un nœud : c'est la longueur du chemin qui relie la racine au nœud.
- Hauteur d'un arbre : c'est la profondeur maximale de ses nœuds. La plupart des algorithmes sur les arbres ont une complexité qui dépend de la hauteur.
- Taille d'un arbre : c'est le nombre de nœuds internes de l'arbre.
- Théorème : on note h la hauteur d'un arbre et n la taille d'un arbre alors
  - $h = n$ dans le pire des cas
  - $h = \log_2(n)$ dans le meilleur des cas

Exemple


Arbre
Taille : 7
Hauteur : 3
w
parent
Profondeur : 2
enfant
Profondeur : 3
sous-arbre de
racine w

Parcours d’un arbre

- Un parcours est un algorithme qui permet de visiter tous les sommets d'un arbre.
- Un parcours en profondeur consiste à aller vers les fils avant d’aller vers les frères
- Un parcours en largeur consiste à parcourir les frères avant de parcourir les fils

Parcours en profondeur d'un arbre

- Un parcours en profondeur est préfixé si tous le nœuds internes sont visités avant leurs enfants.
- Un parcours en profondeur est postfixé si tous le nœuds internes sont visités après leurs enfants.
- Dans un arbre binaire, un parcours en profondeur est infixé quand un nœud est visité après son fils gauche et avant son fils droit.

Algorithmes de parcours en profondeur

- Algorithme d'un parcours préfixé d'un arbre

  parcours de l'arbre a
  - visiter la racine de a
  - pour chaque enfant e de la racine de a

    parcours du sous arbre enraciné en e
- Algorithme d'un parcours postfixé d'un arbre

  parcours de l'arbre a
  - pour chaque enfant e de la racine de a

    parcours du sous arbre enraciné en e
  - visiter la racine de a
- en ed

Algorithmes de parcours en profondeur

- Algorithme d'un parcours infixé d'un arbre binaire

  parcours de l'arbre a

  - soit eg l'enfant gauche de la racine de a
    parcours du sous arbre enraciné en eg
  - visiter la racine de a
  - soit ed l'enfant droit de la racine de a
    parcours du sous arbre enraciné en ed

Algorithme de parcours en largeur

Parcours en largeur de l'arbre a

- On crée et initialise une file f et on stocke dans cette file, la référence de a
- Tantque f est non vide
  - On récupère la tête ar de f
  - On défile f
  - On visite le sommet de ar
  - On enfile dans f tous les références des fils enracinés de ar
- Fin Tantque

Type ArbreKAire

Méthodes de construction et d'initialisation

- fonction arbreVide() : ArbreKAire
- Fonction creerArbre(Element, Entier) : ArbreKAire

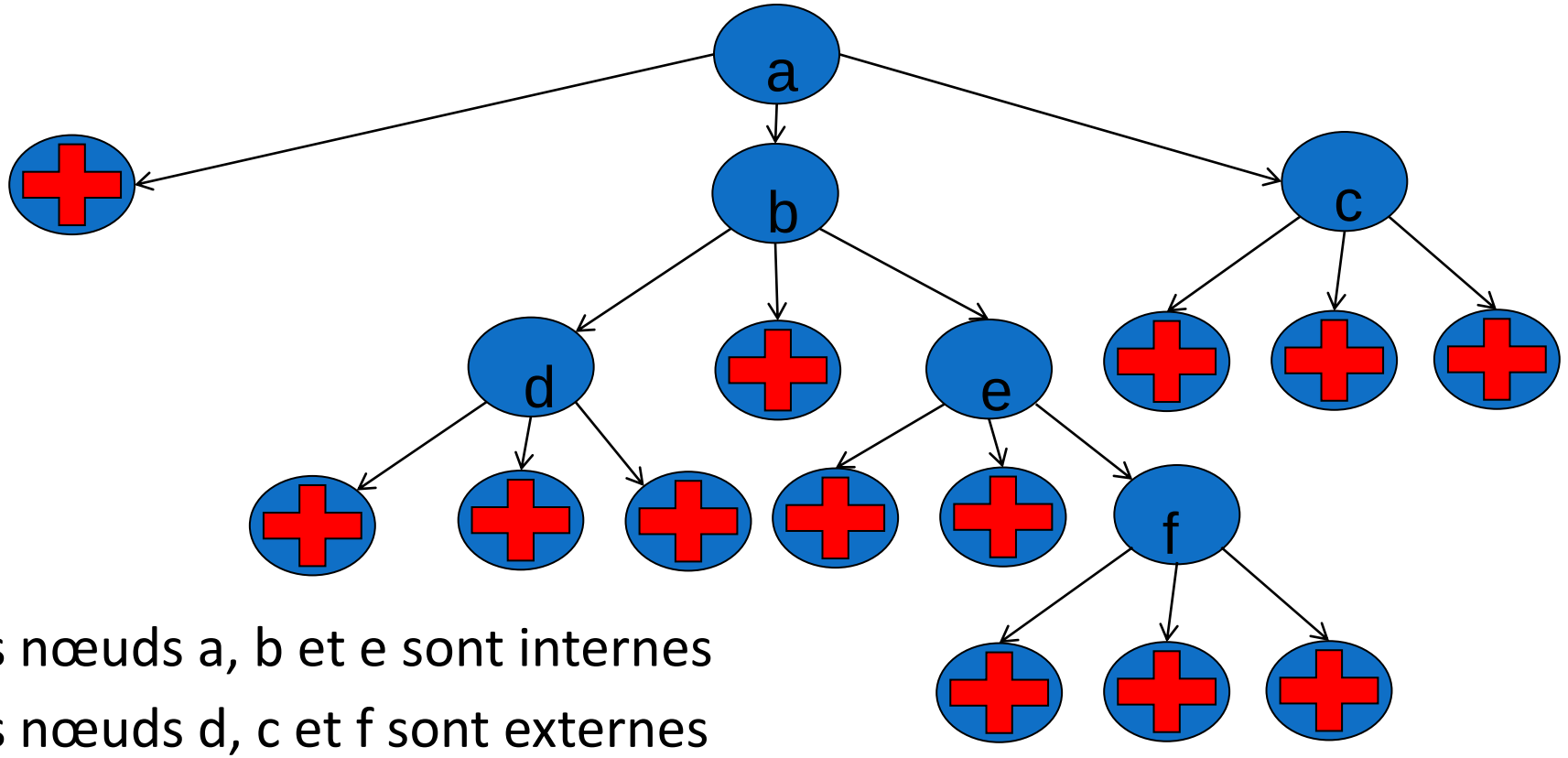
Méthodes de mise à jour

- fonction modifierRacine (ArbreKAire, Element) : ArbreKAire
- fonction affEnfant (ArbreKAire, Entier, ArbreKAire): ArbreKAire
- fonction supprimerFeuille(ArbreKAire, Entier) :ArbreKAire

Méthodes d'accès

- fonction estVide (ArbreKaire) : Booleen
- fonction recRacine(ArbreKaire) : Element
- fonction existeParent(ArbreKaire) : Booleen
- fonction recEnfant(ArbreKaire,Entier) : ArbreKAire
- fonction recParent(ArbreKaire) : ArbreKAire
- fonction recArite(ArbreKaire) : Entier

Exemple d'un arbre 3Aire



les nœuds a, b et e sont internes

les nœuds d, c et f sont externes

Ce symbole représente un arbre vide

Préconditions du Type ArbreKAire

- définie(creerArbre(e,k)) $\Leftrightarrow$ k >= 1
- définie(modifierRacine(ar,e)) $\Leftrightarrow$ non estVide(ar)
- définie(affEnfant(ar,i,enfant)) $\Leftrightarrow$ i >= 1 et i <= recArite(ar)
- définie(supprimerFeuille(ar,i)) $\Leftrightarrow$
  j >= 1 et j <= recArite(ar) $\Rightarrow$ estVide(recEnfant(recEnfant(ar,i), j))
- définie(recRacine(ar)) $\Leftrightarrow$ non estVide(ar)
- définie(recEnfant(ar,i)) $\Leftrightarrow$ non estVide(ar) et i >= 1 et i <= recArite(ar)

Axiomes du Type ArbreKAire

- recArite(creerArbre(e,k)) = k
- recRacine(creerArbre(e,k)) = e
- estVide(recParent(creerArbre(e,k)))
- estVide(recEnfant(creerArbre(e,k),i))
- non existeParent(creerArbre(e,k))
- non estVide(creerArbre(e,k))
- estVide(recEnfant(supprimerFeuille(ar,i),i))
- affEnfant(ar,i,enfant) $\Rightarrow$ recParent(enfant) = ar
- recEnfant(affEnfant(ar,i,enfant),i) = enfant
- recRacine(modifierRacine(ar,e)) = e
- existeParent(recEnfant(affEnfant(ar,i,enfant),i))