Le type abstrait Arbre

Arbres et Graphes

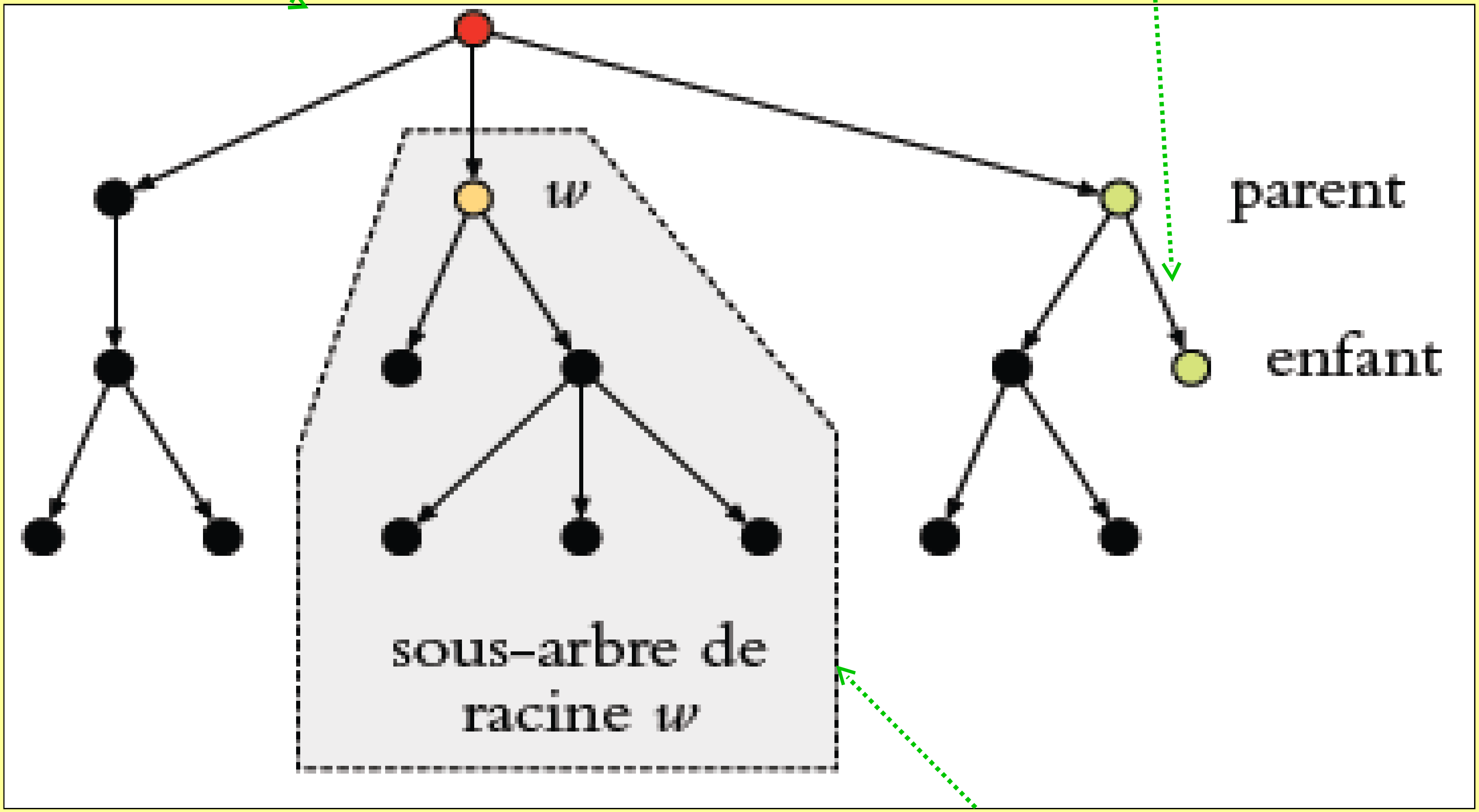
- Arbre libre : C'est un graphe non orienté, connexe et sans cycle.
- Arbre avec racine ou arborescence : C'est un graphe orienté et connexe avec un sommet spécial, la racine, dans lequel il y a 1 chemin simple de la racine à chaque sommet.
- Dans la suite :
  - on étudiera que des arborescences mais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

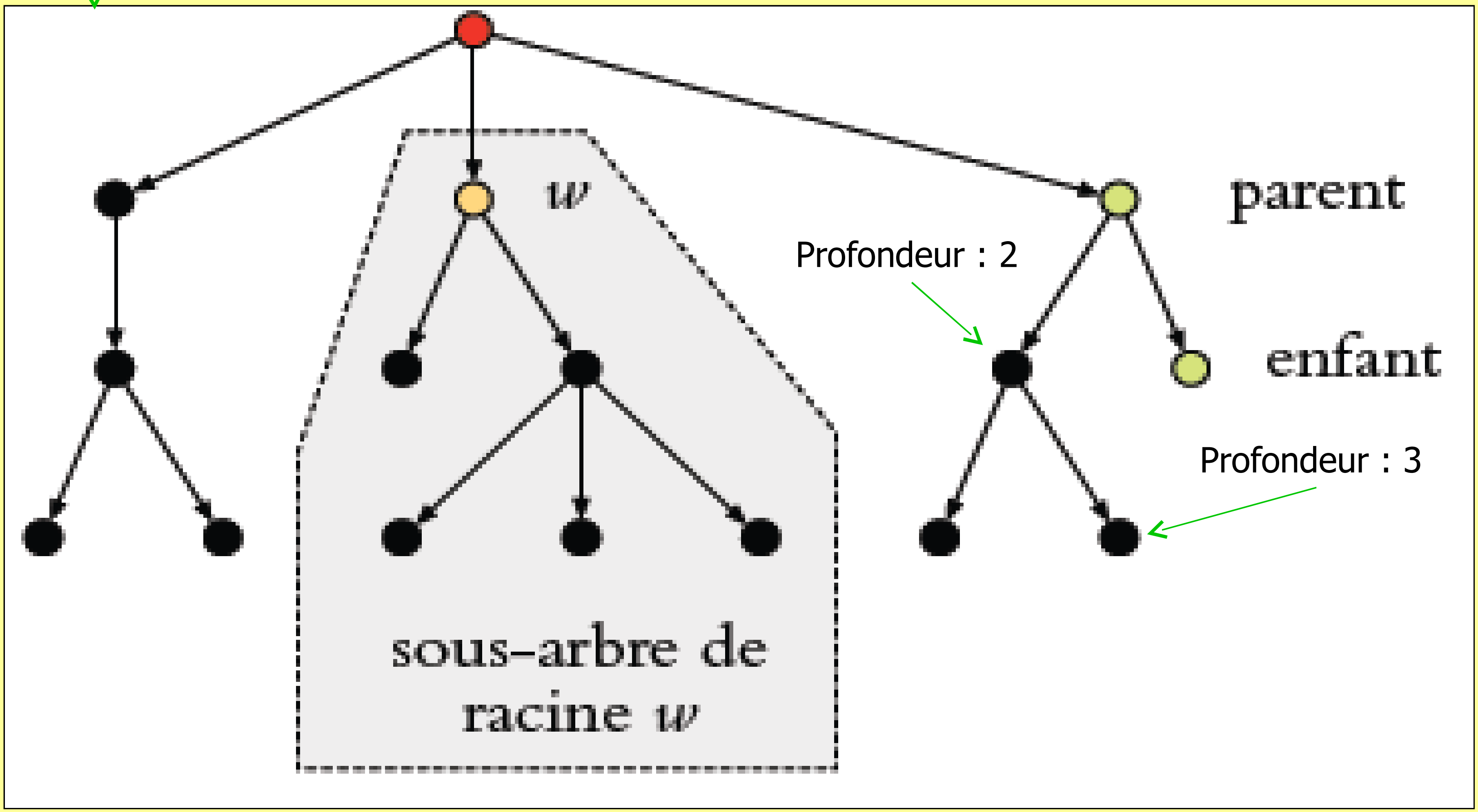
Terminologie

- 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
  - Arité k : c'est un arbre dans lequel il n'existe pas d'enfants dont l'étiquette est supérieure 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

Arbre
Taille : 7
Hauteur : 3



Parcours des arbres

- Un parcours est un algorithme qui permet de visiter tous les sommets d'un arbre.
- Un parcours est préfixé si tous le nœuds internes sont visités avant leurs enfants.
- Un parcours est postfixé si tous le nœuds internes sont visités après leurs enfants.
- Dans un arbre binaire, un parcours est infixé quand un nœud est visité après son fils gauche et avant son fils droit.

Algorithmes de parcours

- 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
- 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

Type abstrait ArbreKAire

TYPE ABSTRAIT ArbreKAire

Ce type modélise les arbres k-aire

Opérations de base

Constructeur ArbreKAire : arbreVide() : ArbreKAire

Constructeur ArbreKAire : creerArbre(Element, Entier) : ArbreKAire

Transformateur ArbreKAire : modifierRacine (Element) : ArbreKAire

Transformateur ArbreKAire : affEnfant (Entier, ArbreKAire): ArbreKAire

Transformateur ArbreKAire : supprimerFeuille(Entier) :ArbreKAire

Observateur ArbreKAire : estVide () : Booleen

Observateur ArbreKAire : recRacine() : Element

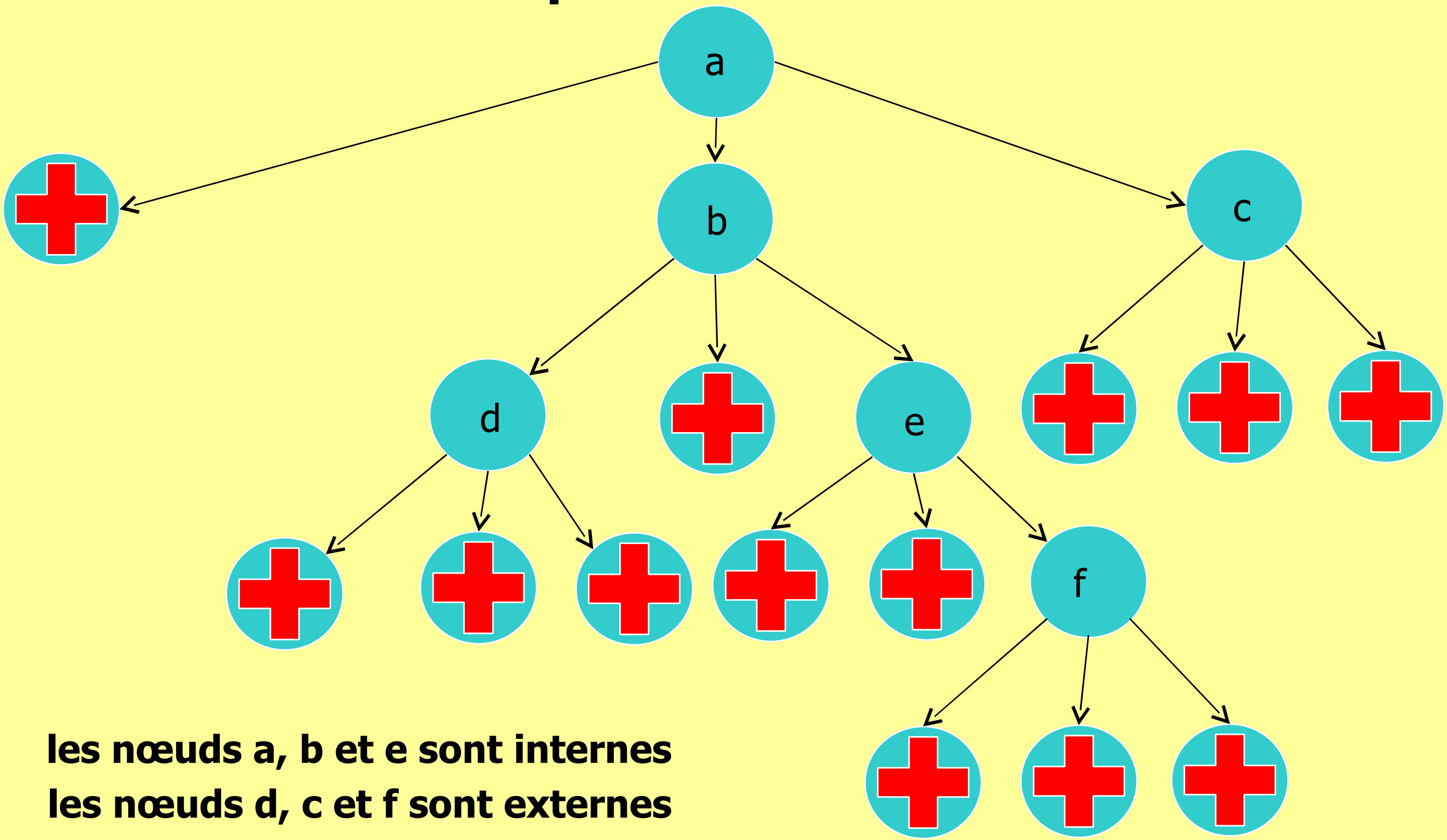
Observateur ArbreKaire : existeParent() : Booleen

Observateur ArbreKAire : recEnfant(Entier) : ArbreKAire

Observateur ArbreKAire : recParent() : ArbreKAire

Observateur ArbreKAire : recArite() : Entier

Exemple d'un d'arbre 3-aire



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 abstrait ArbreKAire

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

Axiomes du Type abstrait ArbreKAire

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