Chapitre 3 : parcours d’arbres et de graphes

Plan

- 1/ Parcours d’arbre
  - 1/ Parcours en largeur
  - 2/ Parcours en profondeur : préfixe, infixe, suffixe
- 2/ Parcours de graphe
  - 1/ Parcours en largeur (BFS)
  - 2/ Parcours en profondeur (DFS)

1/ Parcours d’arbre

- Le parcours d’arbre permet de parcourir une et une seule fois tous les nœuds de l’arbre

1.1/ Parcours en largeur

- On parcourt les nœuds par niveau, en commençant par la racine et en s'éloignant de plus en plus jusqu'aux feuilles
- Algorithme :
  - *file* <— { }
  - enfiler racine(*T*) dans *file*
  - tantque *file* non vide faire
    - *n* <— défiler *file*
    - traiter *n*
    - pour tout fils *f* de *n* dans *T* faire
      - enfiler *f* dans *file*

1.2/ Parcours en profondeur : préfixe, infixe, suffixe

- Les parcours en profondeur se programment de manière récursive
- Algorithme :
  - traiter *n*
  - pour tout fils *f* du noeud actuel *n* faire
    - parcours(*f*)
- Préfixe / infixe / suffixe : pour les arbres binaires uniquement, dépend à quel moment on traite la racine par rapport aux fils

2/ Parcours de graphe

- Le parcours de graphe permet de parcourir une et une seule fois tous les nœuds du graphe
  - Mais les boucles, possibles dans un graphe, rendent un peu plus compliqué le parcours que pour un arbre

2.1/ Parcours en largeur

- Parcours en largeur = *Breadth-First Search* (BFS)
- Idem que pour le parcours en largeur d'un arbre, sauf qu'il ne faut pas parcourir plusieurs fois les mêmes nœuds
  - On utilise un tableau pour savoir si un nœud a déjà été parcouru ou non

2.2/ Parcours en profondeur

- Parcours en profondeur = *Depth-First Search* (DFS)
- Idem que pour le parcours en profondeur d'un arbre, sauf qu'il ne faut pas parcourir plusieurs fois les mêmes nœuds
  - On utilise un tableau pour savoir si un nœud a déjà été parcouru ou non
- Ce parcours permet d'implémenter une recherche de chemin dans un labyrinthe (et faisable en se déplaçant case par case, contrairement au parcours en largeur)