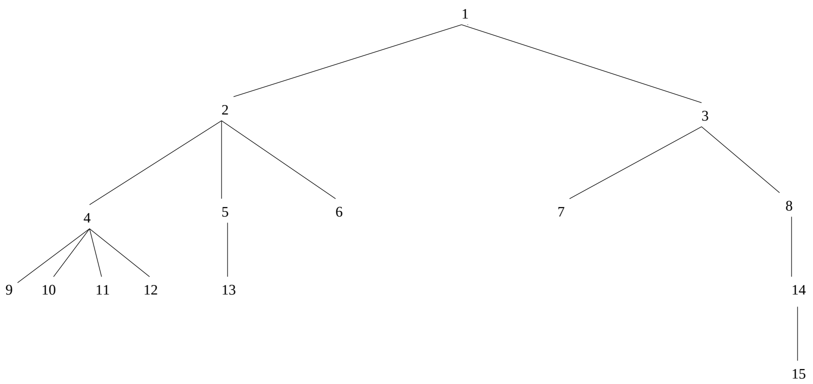
# TP4

## 1 Parcours d'un arbre n-aire en largeur d'abord

Un arbre n-aire est représenté :

- par un atome si A compte un seul noeud
- par une liste dont le 1er élément est la racine de l'arbre et le cdr est la forêt (la liste) des fils de la racine

L'arbre

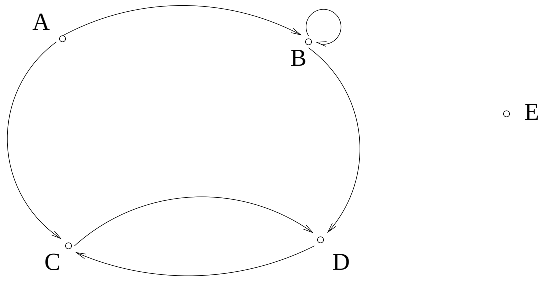



est donc représenté par la liste
(1 (2 (4 9 10 11 12) (5 13) 6) (3 7 (8 (14 15))))

Le parcours en largeur de cet arbre conduit à (1 2 3 4 5 6 7 8 9 10 11 12 13 14 15)
Ecrire le programme LISP permettant le parcours en largeur d'abord d'un arbre.

## 2 Graphes

Un graphe orienté peut être représenté par une liste de sous-listes. Chaque sous-liste contient le nom d'un sommet puis la liste de ses successeurs.



Le graphe ci-dessus est représenté par la liste:

((A B C) (B B D) (C D) (D C) (E))

2.1. Ecrire la fonction **liste_sommets** qui construit le liste des sommets d'un graphe.

2.2. Ecrire la fonction **liste_succ** qui construit le liste des successeurs d'un sommet donné.

2.3. Ecrire la fonction **liste_arcs** qui construit le liste des arcs d'un graphe.

2.4. Ecrire la fonction **liste_pred** qui construit le liste des prédecesseurs d'un sommet donné.

2.5. Ecrire la fonction **rep_pred** qui donne la représentation du graphe sous la forme liste de sous-listes où chaque sous-liste contient un sommet et la liste de ses prédécesseurs.

2.6. Ecrire une fonction qui teste si un graphe est un arbre.
Indications: on utilisera la réprésentation avec liste de prédécesseurs. On écrira une fonction **sans_pred(G)** qui renvoie la liste des sommets sans prédécesseurs. Dans un arbre, il y a un seul sommet sans prédécesseur et il n'y a pas de cycle. La recherche des cycles peut se faire par l'algorithme suivant:

- étape 1. Calculer les sommets sans prédécesseur
- étape 2. Eliminer ces sommets du graphe

Répéter 1 et 2 soit jusqu'à obtention du graphe vide (alors le graphe donné est un arbre), soit jusqu'à ce qu'il n'y ait plus de sommets sans prédécesseur (on n'a pas un arbre).