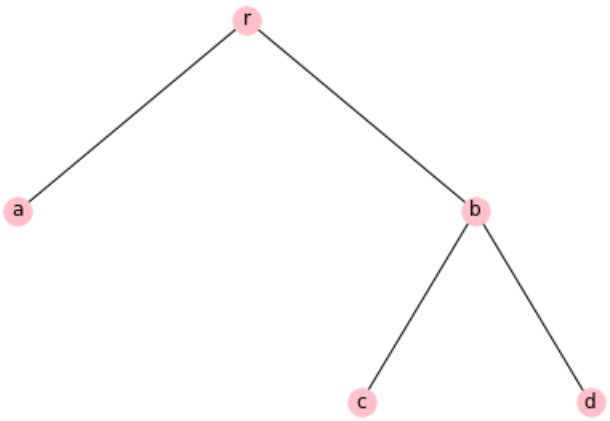
TD: implémentation des arbres binaires:

1. **Implémentation d'un arbre binaire à l'aide d'un dictionnaire:**
   On s'interesse à l'arbre binaire qui suit:



   On l'implémente de la façon suivante:

```python
def noeud(nom, fg = None, fd = None) :
    return {'racine': nom, 'fg' : fg, 'fd': fd}

#partie droite:
c = noeud('c')
d = noeud('d')
b = noeud('b', c, d)

#partie gauche:
a = noeud('a')

#arbre en entier:
A = noeud('r', a, b)
```

arbres_binaires.py

   1.1 Quel est le type de A? Quel est le type de b?

   1.2 Que renvoient les commande b['racine'], b['fg'] et c['fd']

2. **Implémentation d'un arbre binaire à l'aide d'un tableau:**
   On souhaite implémenter cet arbre binaire sous la forme d'un tableau du type:

   ['r', ['a', [ ], [ ]], ['b', ['c', [ ], [ ]], ['d', [ ], [ ]]]]

   On utilise alors la fonction suivant:

```python
def construit(arbre):
    if arbre == None:
        return []
    else:
        return [arbre['racine'], construit(arbre['fg']), construit(arbre['fd']) ]
```

arbres_binaires.py

   2.1 Quelle est la particularité dans l'écriture de cette fonction?

   2.2 Si C = construit(A)

   - Que valent C[0], C[1] et C[2]?
   - Commment à partir de C accède t-on à a?
   - Comment à partir de C accède t-on à d?

3. **Implémentation 'plus visuelle' d'un arbre binaire:**
   On souhaite une représentation encore plus visuelle. On utilise alors la fonction suivante:

```python
def represente(arbre, p = 0) :
    if arbre == []:
        print('*')
    else:
        print(arbre[0])
        p += 1
        print('—' * p, end ='')
        represente(arbre[1], p)

        print('—' * p, end ='')
        represente(arbre[2], p)
```

arbres_binaires.py

   3.1 Quelle est la particularité dans l'écriture de cette fonction?

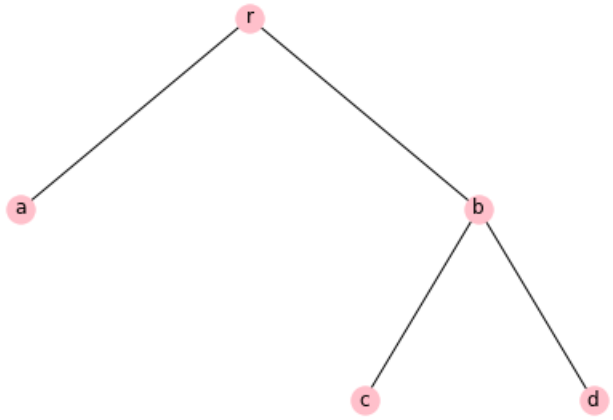
   3.2 Que va afficher represente(C)?

4. **Implémentation d'un arbre binaire avec la bibliothèque networkx:**
   networkx est une bibliothèque python permettant l'étude et la manipulation de graphes. Elle peut être utilisée pour implémenter des arbres.

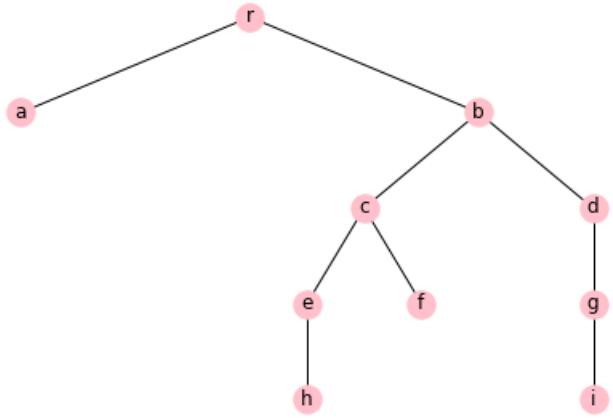
```python
import networkx as nx
import random
import matplotlib.pyplot as plt

def hierarchy_pos(G, root=None, width=1., vert_gap = 0.2, vert_loc = 0, xcenter = 0.5):
    if not nx.is_tree(G):
        raise TypeError('cannot use hierarchy_pos on a graph that is not a tree')

    if root is None:
        if isinstance(G, nx.DiGraph):
            root = next(iter(nx.topological_sort(G)))  #allows back compatibility with nx version 1.11
        else:
            root = random.choice(list(G.nodes))

    def _hierarchy_pos(G, root, width=1., vert_gap = 0.2, vert_loc = 0, xcenter = 0.5, pos = None, parent = None):
        if pos is None:
            pos = {root:(xcenter,vert_loc)}
        else:
            pos[root] = (xcenter, vert_loc)
        children = list(G.neighbors(root))
        if not isinstance(G, nx.DiGraph) and parent is not None:
            children.remove(parent)
        if len(children)!=0:
            dx = width/len(children)
            nextx = xcenter - width/2 - dx/2
            for child in children:
                nextx += dx
                pos = _hierarchy_pos(G,child, width = dx, vert_gap = vert_gap,
                                     vert_loc = vert_loc-vert_gap, xcenter=nextx,
                                     pos=pos, parent = root)
        return pos

    return _hierarchy_pos(G, root, width, vert_gap, vert_loc, xcenter)


G=nx.Graph()
G.add_edges_from([('r','a'), ('r','b'), ('b','c'), ('b','d')])
pos = hierarchy_pos(G,'r')


nx.draw(G, node_color = 'pink', pos=pos, with_labels=True)
plt.savefig('hierarchy.png')

print(nx.is_tree(G))
```

networkx.py

Le programme précédent permet alors d'obtenir la représentation suivante:



5. **Application:**
   On s'interesse à l'arbre binaire qui suit:



5.1 Implémenter cet arbre en utilisant la fonction noeud.

5.2 Implémenter cet arbre à l'aide de la fonction construit.

5.3 Représenter cet arbre à l'aide de la fonction représente.

5.4 Implémenter puis représenter cet arbre à l'aide de la bibliothèque networkx.

6. **Un premier algorithme:**
On souhaite maintenant écrire une fonction permettant de calculer la hauteur de l'arbre.

6.1 Définir la hauteur d'un arbre.

On utilise la fonction suivante:

```
0 def hauteur(arbre):
      if arbre == []:
2         return -1
      else:
4         h1 = 1 + hauteur( arbre[1] )
          h2 = 1 + hauteur( arbre[2] )
6         return max(h1, h2)
```

arbres_binaires.py

6.2 A quelle implémentation cette fonction s'applique t-elle?

6.3 Que va renvoyer hauteur(C)?

6.4 Expliquer avec précision ce que fait la fonction hauteur().

6.5 Tester cette fonction sur les deux arbres de ce TD.

6.6 Écrire une fonction hauteur_bis permettant de renvoyer la hauteur d'un arbre implémenté sous la forme d'un dictionnaire.