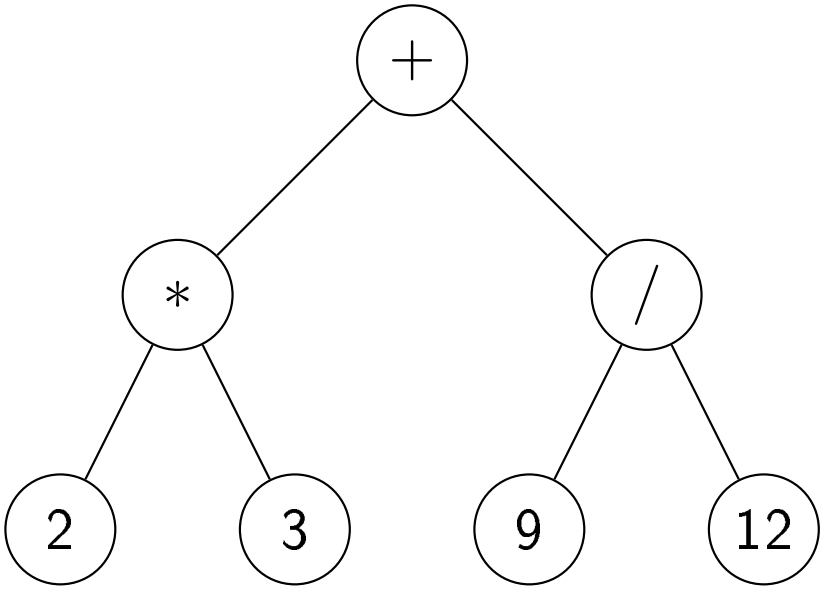
Vocabulaires des arbres binaires (Charlotte) :

- noeuds
- feuilles
- degré
- taille
- hauteur (= profondeur)
- chemin
- parent / père / fils
- clef/etiquette

- arbre ordonné : tout les chemin forme une suite décroissante
- arbre d'expression arithmétiques



Méthodes et complexités sur les Arbres Binaires de Recherches (ABR ou BST pour la suite) (Charlotte + Thiziri)

parcours en largeur :

```
1  def parcours(self):
2      f = [] #file
3      f.append(self)
4          while(len(f)!=0):
5              racine = f.pop()
6              #traitement
7                  f.append(racine.gauche)
8                  f.append(racine.droit)
```

Complexité : $\Theta(n)$: on passe exactement une fois sur chaque nœud

parcours en profondeur

- parcours préfixe
- parcours suffixe / postfixe
- parcours infixe

```
1  def parcours_prefixe(arbre):
2      if arbre vide:
3          return []
4      else:
5          return [racine(arbre) + parcours_prefixe(sous-
       arbre gauche) + parcours_prefixe(sous arbre droit)]
```

complexité linéaire en la taille également

recherche : on compare la valeur cherchée à la clef du noeud courant , 3 cas possible :

- valeur cherchée = clef => on renvoie True
- valeur cherchée < clef => appel récursif sur n.gauche
- noeud cherché > clef => appel récursif sur n.droit

Complexité : $\Omega(1)$: meilleur cas : le noeud cherché est à la racine, et $\mathcal{O}(h)$ pire cas : le noeud cherché est dans la feuille la plus profonde.

Version sans rotation : Même fonctionnement que la recherche, une fois qu'on a atteint une feuille on créé le noeud correspondant à la valeur à ajouté. La complexité est aussi en $\mathcal{O}(h)$: si la valeur doit être insérée dans la feuille la plus profonde. Note : dans cette version on insère forcément dans une feuille, on ne se soucie pas d'avoir une structure la plus équilibrée possible.

suppression

explication au tableau

On a donc des méthodes linaires en la hauteur au pire cas.
Plusieurs questions en découlent :

- Complexité en la taille ? dépend de la structure d'arbre. Notions d'arbre parfait / complet : h = $\Theta(log(n))$ (Preuve de Kevin)
- Que dire de la hauteur "moyenne" d'un ABR ? Espérance, variance ?
- questions de combinatoire : combien d'arbre binaire possible d'une taille donnée ? combien de BST possible pour une même liste de nombre ?
- position moyenne dans l'arbre d'un noeud spécifique ?

pistes de réponses :

- position moyenne d'un noeud = h-1 = log(n) (preuve par Kevin)
- hauteur moyenne : équivalente à $2\sqrt{\pi n}$ en ∞ (d'après un article de Philippe Flajolet).
- $\frac{\mathbb{E}[H_n]}{log(n)} \rightarrow c = 4.31107...$ d'après un article de Luc Devroye On peut peut être trouver une inégalité moins optimale qui donne la complexité moyenne en log(n) avec une preuve accessible.

mesure expérimentale

même principe que la dernière fois. Le protocole est de 100 mesures par taille d'arbre. On incrémente la taille plus finement au début quand ça ne coûte pas trop de faire tourner l'algorithme. Cf courbes pour les résultats. La principale différence avec la dernière fois : on teste sur des arbres parfait ou quasi-parfait. Pour garantir que l'arbre s'approche d'un arbre parfait constituée des valeurs de 1 à n on construit une liste avec des multiplications par 1/2 ou 3/2 successives

implémentation quadtree

Première ébauche d'une implémentation de Quadtree par Thiziri. Toujours en Python en paradigme objet. C'est une implémentation de Quadtree "Régions" et pas de Quadtree "Points"

pour la suite

- affiner les résultats des mesures expérimentales
- implémenter les rotations d'arbres et l'insertion avec rotation + comparer les hauteurs moyennes obtenues
- se documenter sur la théorie des quadtree region
- faire une implémentation de quadtree points
- synthétiser les résultats des articles sur les questions d'hauteur moyenne / combinatoire etc.