TP4 : Division et algorithme d'Euclide

1 La division euclidienne

Algorithme soustractif

Etant donnés deux entiers $a \geq b > 0$, on recherche deux entiers r et q tels que :

$$\begin{cases} a = bq + r \\ 0 \leq r < b \end{cases}$$

Pour ce faire, une méthode naïve consiste à retrancher b à a jusqu'à ce que le reste respecte la deuxième condition.

1. Ecrire, implémenter et tester votre algorithme `div_eucl1`.
2. Qu'est-ce qui vous garantit que son exécution s'arrête ?
3. Prouver que votre algorithme renvoit bien les (r, q) recherchés.
4. Trouver a_0 et b_0 tels que le temps de calcul soit supérieur à deux secondes.
5. Calculer la complexité de l'algorithme `div_eucl1`.

En décomposant selon une base

6. Poser, à la main, la division euclidienne de 1672 par 23. Avez-vous procédé comme l'algorithme que vous venez d'écrire ? Si non, en quoi était-ce "optimisé" ?
7. De la même façon, poser en binaire la division euclidienne de 10110 par 11.
8. ♠ Ecrire l'algorithme général de division euclidienne en travaillant directement sur les listes de bits. Quelle est sa complexité ?

On considère l'algorithme suivant :

```
def div_eucl2(a,b):
    if a >= b  and b > 0 :
        (q,r) = (0,a)
        # Soit n le plus petit entier
        # tel que a < 2^n b
        n = a.nbits() - b.nbits()+1
        s = 2^n * b
        for k in range(n):
            s = s/2
            q = q*2
            if r >= s:
                q = q+1
                r = r - s
        return (q,r)
```

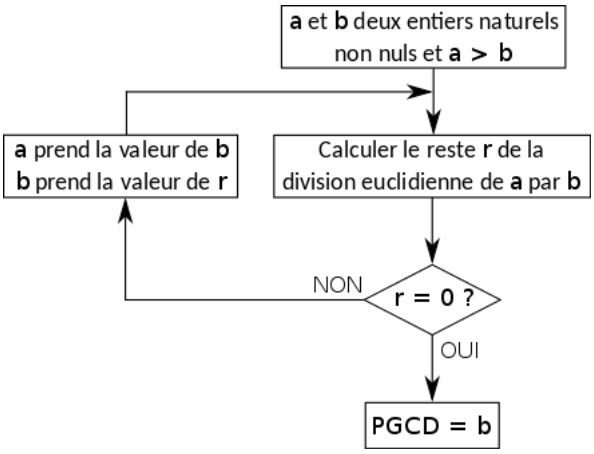
9. Implémenter et tester cet algorithme.
10. Vérifier qu'à chaque étape on a : $a = sq + r$, avec $0 \leq r < s$; vérifier qu'à l'issue de la dernière étape : $s = b$. Qu'en concluez-vous ?
11. Quel est, ce coup-ci, le temps de calcul pour l'entrée (a_0, b_0) trouvée à la question 4 ?

12. Calculer la complexité de l'algorithme.

    Nota : Vous pourrez, lors des TP suivants, obtenir le quotient et le reste de la division euclidienne de a par b grâce aux fonctions `a//b` et `a%b`.

2 Algorithme d'Euclide

Schéma de la procédure :



13. La procédure ci-dessus peut-elle boucler indéfiniment ?
14. Prouver que l'algorithme renvoit bien $pgcd(a, b)$ en justifiant qu'à chaque étape, le $pgcd$ est conservé.
15. Ecrire et implémenter l'algorithme, en utilisant votre fonction `div_eucl2`.
16. ♠ Quel est le plus petit couple d'entiers pour lequel l'exécution requiert n étapes ?
17. En déduire la complexité au pire de l'algorithme d'Euclide.
18. Pourquoi n'a-t-on pas directement listé les diviseurs premiers de a puis de b, et cherché leurs facteurs communs ?

Nota : Vous pourrez, lors des TP suivants, obtenir le PGCD de a et de b grâce à la fonction `gcd(a,b)`.

3 Algorithme d'Euclide étendu

L'algorithme d'Euclide étendu renvoit non seulement le PGCD de a et b, mais également un jeu de coefficients u et v satisfaisant l'idendité de Bézout : $pgcd(a, b) = au + bv$.

19. En déroulant à la main l'algorithme d'Euclide, calculer le PGCD de 106 et 24 et déduire des étapes intermédiaires un jeu de coefficients de Bézout.

    $\longrightarrow$ *Comment obtenir ces coefficients sans avoir besoin de "remonter" (et donc d'enregistrer) les calculs ?*
20. On désigne par $(r_n)_n$ la suite des restes successifs : $r_0 = a$, $r_1 = b$, et pour tout $n \geq 0$: r_{n+2} est le reste de la division euclidienne de r_n par r_{n+1}. Ecrire la relation matricielle liant le vecteur (r_{n+2}, r_{n+1}) au vecteur (r_{n+1}, r_n), puis au vecteur (a, b). Comment en déduit-on les coefficients de Bézout ?
21. Ecrire, implémenter et tester l'algorithme `euclide_etendu(a,b)`. Le comparer à `xgcd(a,b)`.
22. Utiliser l'algorithme pour calculer les inverses de 55 et 59 mod 105.

♠ Pour aller plus loin :

⟶ Tester vos algorithmes sur des polynômes à coefficients rationnels.

⟶ En appliquant l'algorithme d'Euclide non plus à des entiers, mais au couple $(x, 1)$, on peut construire le *développement en fractions continues* du nombre réel x, qui en est la description la plus efficace par une suite de rationnels.

⟶ *L'algorithme de Sturm* est un algorithme d'Euclide un peu modifié qui, appliqué à un polynôme P et à son polynôme dérivé, permet de calculer le nombre de racines distinctes de P dans un intervalle choisi (et donc, par dichotomie, de localiser ses racines...)