M1 MFA
Mémoire d'initiation à la recherche
Problème du mot dans les groupes de Coxeter

Lucas Pouillart
Encadré par Baptiste Rognerud

Janvier - Juin 2024

Table des matières

1 Groupes de Coxeter finis et mots réduits **1**
1.1 Introduction aux groupes de Coxeter 1
1.2 Le problème du mot dans les groupes de Coxeter 2

2 Groupe de réflexions réels **3**
2.1 Réflexions et racines 3
2.2 Mots et réduction des mots 9

3 Groupes de réflexions et systèmes de Coxeter **13**
3.1 Correspondance des groupes finis 13
3.2 Théorème de Matsumoto et retour au problème du mot 23

A Tables de hachage **28**

B Plus long élément de A_n et bijection d'Edelman-Greene **29**

C Complexité du programme **34**

1 Groupes de Coxeter finis et mots réduits

Dans cette partie, nous allons donner quelques définitions importantes sur les groupes de Coxeter et leur problème du mot, notamment afin de montrer par la suite que ce dernier est décidable.

1.1 Introduction aux groupes de Coxeter

Définition 1.1. On appelle système de Coxeter tout couple (W, S) tel que W est un groupe engendré par $S = \{s_1, ..., s_n\}$ admettant une présentation par relations de la forme

$$\langle s_1, ..., s_n | (s_i s_j)^{m_{ij}} = 1 \rangle$$

où $m_{ii} = 1$ et $m_{ij} = m_{ji} \geq 2 \in \mathbb{N}$ si $i \neq j$.

Si (W, S) est un système de Coxeter, alors on dit que W est un *groupe de Coxeter* et que S est son ensemble de *générateurs de Coxeter*.

Exemple 1.2. Pour tout $n \in \mathbb{N} - \{0\}$, (S_{n+1}, S) où $S = \{(i\ i+1) | i \in \{1, ..., n\}\}$ est un système de Coxeter, que l'on note usuellement A_n.

Définition 1.3. Soit S un ensemble de générateurs d'un groupe G. On appelle *mot* sur S dans G un produit fini d'éléments de $S \cup S^{-1}$, où S^{-1} est l'ensemble des inverses d'éléments de S. On dit que deux mots sont *équivalents* s'ils représentent le même élément de G.
Dans le cas des groupes de Coxeter, on a $S = S^{-1}$ donc on se contente de parler des mots sur S.

Exemple 1.4. Considérons le groupe de Coxeter A_2, on a $S = \{(1\ 2), (2\ 3)\} := \{s_1, s_2\}$.
s_1, s_2, s_1s_2, $s_2s_1s_2s_1$, $s_1s_1s_2$ sont des mots sur S.
Certains sont équivalents comme s_2 et $s_1s_1s_2$ et d'autres non comme s_2 et s_1.

1.2 Le problème du mot dans les groupes de Coxeter

Le problème du mot dans les groupes de type fini peut se définir de la manière suivante :
Soient G un groupe engendré par un ensemble fini S, et deux mots w_1 et w_2 sur S.
Le problème du mot consiste à savoir si dans le groupe G, les mots w_1 et w_2 sont équivalents, c'est-à-dire s'ils représentent le même élément de G.
Dans le cas général, ce problème est indécidable, il existe des groupes dont les présentations ne permettent pas de déterminer si deux mots sont équivalents, mais le sous-problème défini lorsqu'on se restreint aux groupes de Coxeter est décidable, on va le résoudre algorithmiquement.

Définition 1.5. Un mot $w = s_1...s_k$ sur S est dit *réduit* s'il n'existe aucun mot $w_2 = s'_1...s'_m$ sur S équivalent à w avec $m < k$.
On dit alors que k est la *longueur* du mot w.

Remarque. La fonction qui à un élément d'un groupe de Coxeter associe sa longueur est bien définie et est facilement calculable, en particulier dans le cas des groupes de Coxeter finis, ce que l'on montrera plus tard.

Une première idée pour décider si deux mots sont équivalents serait de trouver une seule et unique forme réduite, ce qui n'est quasiment jamais le cas dans des groupes finiment engendrés quelconques, et ne l'est pas non plus dans les groupes de Coxeter.
En revanche, si on peut trouver un algorithme qui permet soit de déterminer une unique forme réduite (qu'on appelle alors *forme normale* du mot) pour tout élément de G, c'est-à-dire que deux mots équivalents sur S renvoient le même mot par cet algorithme, soit de réduire algorithmiquement un mot et de déterminer l'ensemble des mots réduits équivalents, alors le problème sera résolu.
Bien que les deux approches soient possibles pour les groupes de Coxeter, nous nous intéresserons ici à la deuxième procédure, qui nous permettra d'étudier et d'illustrer des propriétés intéressantes des groupes de Coxeter. En général, la première approche possède une complexité beaucoup moins élevée, mais la deuxième approche nous donne beaucoup plus d'informations intéressantes à étudier.

Nous commencerons par étudier le problème de la réduction d'un mot grâce à la théorie des représentations, ce qui s'avèrera particulièrement intéressant pour étudier le cas des groupes de Coxeter finis, et ensuite nous étudierons les relations entre les différents mots réduits représentant un même élément d'un groupe de Coxeter.

2 Groupe de réflexions réels

Dans cette partie, soit E un espace euclidien de dimension n, muni du produit scalaire euclidien $(\cdot,\cdot)$, et pour $\alpha \in E$, on note H_α l'hyperplan défini par $H_\alpha = \{\beta \in E | (\alpha,\beta) = 0\}$. On pourra identifier E à $\mathbb{R}^n$.

2.1 Réflexions et racines

Définition 2.1. On appelle *réflexion* dans E un endomorphisme s de E tel que :
- $\exists \alpha \neq 0 \in E$ tel que $s(\alpha) = -\alpha$
- $\forall x \in H_\alpha$, $s(x) = x$

On note s_α la réflexion respectant ces conditions pour $\alpha \neq 0$.

Remarque. Cette notion de réflexion a du sens dans un espace vectoriel quelconque tant qu'il est muni d'une forme bilinéaire symétrique B et que l'on ne se préoccupe pas des questions d'existence (par exemple dans le cas de vecteurs B-isotropes) ou d'unicité.

On appellera *groupes de réflexions réels* les groupes engendrés par un ensemble fini de réflexions d'un espace euclicien E. Il nous arrivera par ailleurs d'appeler groupe de réflexions un groupe d'endomorphismes d'un espace vectoriel V respectant ces propriétés pour une forme bilinéaire symétrique quelconque plutôt qu'un produit scalaire.

Ces sous-groupes de $O(E)$ sont en particulier des groupes de Coxeter (ce que nous montrerons par la suite), et ils s'avèrent être particulièrement intéressants à étudier pour la compréhension de ces derniers, en particulier dans ce cas des groupes de réflexions finis. Nous nous concentrerons sur ces derniers par la suite, mais la plupart des résultats que nous énoncerons sont également valables dans le cas infini et dans les cas non-euclidiens.

Définition 2.2. Soit ϕ une famille finie de vecteurs de E, on dit que ϕ est un système de racines dans E si :
- $Vect(\phi) = E$ et $0 \notin \phi$
- Si $\alpha \in \phi$ les seuls multiples scalaires de α dans ϕ sont $-\alpha$ et lui-même
- Si $\alpha \in \phi$, $s_\alpha(\phi) = \phi$

Remarque. La définition d'un système de racines dans notre cas est plus large que la définition usuelle que l'on rencontre notamment dans la théorie des représentations des algèbres de Lie, dans ce cas on rajoute la condition suivante, appelée *condition d'intégralité* :
- Si $\alpha, \beta \in \phi$, $< \alpha, \beta >= \frac{2(\beta,\alpha)}{(\alpha,\alpha)} \in \mathbb{Z}$

De plus, pour la suite il ne sera pas très important que les vecteurs de ϕ engendrent E. On se permettra donc de parler de systèmes de racines même si $Vect(\phi) \neq E$

Les vecteurs d'un système de racines sont naturellement appelés les racines de ϕ, et tout groupe de réflexions fini W permet naturellement de construire un système de racines (qui n'est pas unique) en réalisant ϕ de la manière suivante :
- Considérons $s \in W$ une réflexion
- Par définition d'une réflexion, il existe un vecteur unitaire α de E tel que $s(\alpha) = -\alpha$, qui définit une droite vectorielle $D_\alpha := \mathbb{R}\alpha$
- On ajoute α et $-\alpha$ à ϕ

Il est facile de vérifier que si on omet effectivement la condition $Vect(\phi) = E$, on obtient un système de racines. Mais nous utiliserons le lemme suivant :

Lemme 2.1. *Si $t \in O(E)$ et $\alpha \neq 0 \in E$ alors $ts_\alpha t^{-1} = s_{t(\alpha)}$.*

Preuve. On doit montrer que :

— $ts_\alpha t^{-1}(t(\alpha)) = -t(\alpha)$:
On a $ts_\alpha t^{-1}(t(\alpha)) = ts_\alpha(\alpha) = t(-\alpha) = -t(\alpha)$

— $\forall v \in H_{t(\alpha)}\ ts_\alpha t^{-1}(t(\alpha)) = t(v)$:
Comme t est une isométrie, on a $(v, \alpha) = (t(v), t(\alpha))$ c'est-à-dire $t(v) \in H_{t(\alpha)} \iff v \in H_\alpha$.
Si $t(v) \in H_{t(\alpha)}$, on a $ts_\alpha t^{-1}(t(v)) = ts_\alpha(v) = t(v)$.

Ce résultat implique notamment que si $w \in W$, on a $s_{w(\alpha)} \in W$ (on peut l'écrire comme produit d'éléments de W), c'est-à-dire que W agit sur les droites vectorielles D_α définies précédemment en les permutant.

On peut maintenant vérifier que cet ensemble définit un système de racines :

— $0 \notin \phi$: en effet, 0 n'étant pas unitaire, on ne peut pas l'avoir dans ϕ.

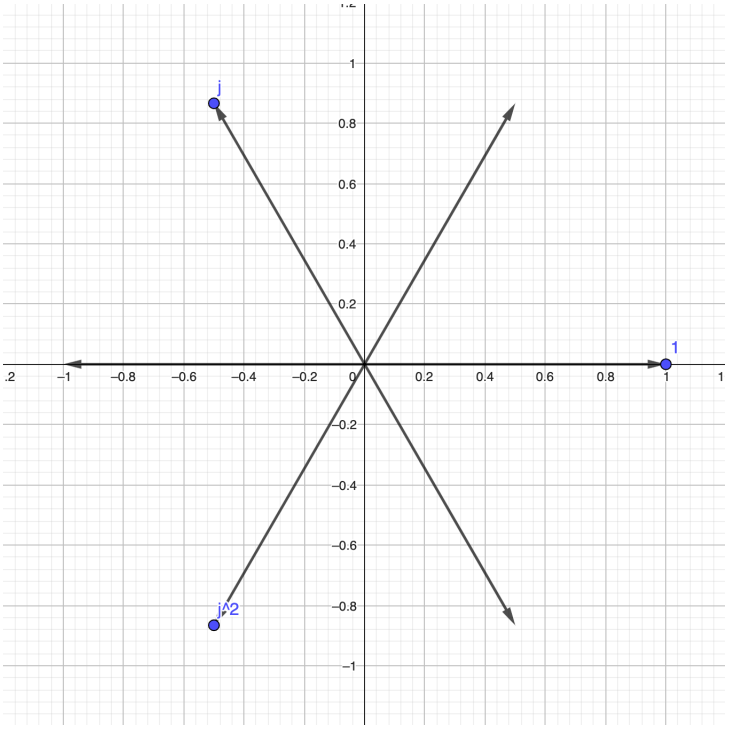
— Si $\alpha \in \phi$, par construction on a bien -$\alpha \in \phi$, et comme on rajoute uniquement des vecteurs unitaires, il n'existe pas d'autre multiple scalaire de α dans ϕ.

— On a vu que W agit en permutant les D_α, et comme chaque réflexion par rapport à une racine β est une isométrie, elle envoie un vecteur unitaire d'une de ces droites (donc une racine de ϕ) sur un autre vecteur unitaire d'une de ces droites (donc une autre racine de ϕ). Donc $s_\alpha(\phi) = \phi\ \forall \alpha \in \phi$.

Remarque. On n'a en réalité pas nécessairement besoin que toutes les racines soient de même longueur, mais c'est la manière la plus simple effectivement parlant de forcer la condition de stabilité des racines par l'action de W et de définir les réflexions par rapport aux racines.

Exemple 2.3. Cette construction peut avoir l'air compliquée en première approche, donc illustrons-la par des exemples :

1. Considérons $\mathbb{C}$ comme espace vectoriel de dimension 2 sur $\mathbb{R}$, et considérons le groupe W d'isométries engendré par les réflexions s_1 et s_j (avec $j = e^{\frac{2i\pi}{3}}$ qui est une racine primitive 3^e de l'unité).
Il est aisé de vérifier que ce groupe est exactement $W = \{Id_{\mathbb{C}}, r_{\frac{2\pi}{3}}, r_{-\frac{2\pi}{3}}, s_1, s_j, s_{\bar{j}}\}$ (dont on peut remarquer qu'il est isomorphe à A_2, c'est-à-dire S_3 en tant que système de Coxeter).
On a 3 réflexions, c'est-à-dire les 6 racines suivantes, qui sont les racines 6^e de l'unité dans $\mathbb{C}$:



Le système de racines associé à W

On peut remarquer que W est isomorphe au système de Coxeter A_2 tel que défini précédemment, donc que A_2 est réalisable comme groupe de réflexions réel.

2. Notons $(\varepsilon_i)_{1\leq i\leq m}$ la base canonique de $\mathbb{R}^m$ pour tout $m > 0$, et considérons $V = \{\sum_{i=1}^{n+1} a_i\varepsilon_i | \sum_{i=1}^{n+1} a_i = 0\} \subset \mathbb{R}^{n+1}$.
De manière générale, le système suivant est un système de racines dans V :

$$A_n := \{\varepsilon_i - \varepsilon_j | 1 \leq i < j \leq n+1\}$$

On peut remarquer que le groupe de réflexion qui lui est associé est le groupe de Coxeter A_n, soit le groupe S_{n+1} qui agit comme groupe de permutation en permutant les coefficients $a_1, ..., a_{n+1}$ des racines.

Remarque. De la même manière qu'on peut associer un système de racines (non unique) à un groupe de réflexions, on peut associer à tout système de racines ϕ le groupe engendré par les réflexions s_α avec $\alpha \in \phi$. En théorie de Lie et dans certains ouvrages de manière générale, ce groupe est appelé *groupe de Weyl* de ϕ, mais si ϕ n'est pas un système de racines au sens de la théorie de Lie, nous utiliserons le terme de groupe de réflexions associé à ϕ pour éviter toute confusion.

Pour la suite nous aurons besoin d'une relation d'ordre total (c'est-à-dire une relation binaire irréflexive et transitive) $<$ sur E qui soit compatible avec sa structure d'espace vectoriel. On impose donc de plus sur $<$ les conditions suivantes :

— $\forall u, v, w \in E$ tels que $u < v$, on a $u + w < v + w$

— $\forall u, v \in E$ tels que $u < v$ et $\lambda \in \mathbb{R}$: si $\lambda > 0$ alors $\lambda u < \lambda v$ et si $\lambda < 0$ alors $\lambda u > \lambda v$

Nous pouvons simplement considérer l'ordre lexicographique naturel sur $\mathbb{R}^n$.

Définition 2.4. Soit ϕ un système de racines, on appelle *racines positives* de ϕ (et on note généralement Π ou ϕ^+) l'ensemble des racines > 0. On appelle parfois *système positif* un tel ensemble.

Remarque. On peut définir de la même manière les racines négatives. Par ailleurs remarquons que si α est une racine positive, alors -α est une racine négative, on notera donc -Π ou ϕ^- l'ensemble des racines négatives.

Définition 2.5. Soit ϕ un système de racines, on dit que $\Delta \subset \phi$ est une *base* de ϕ si :

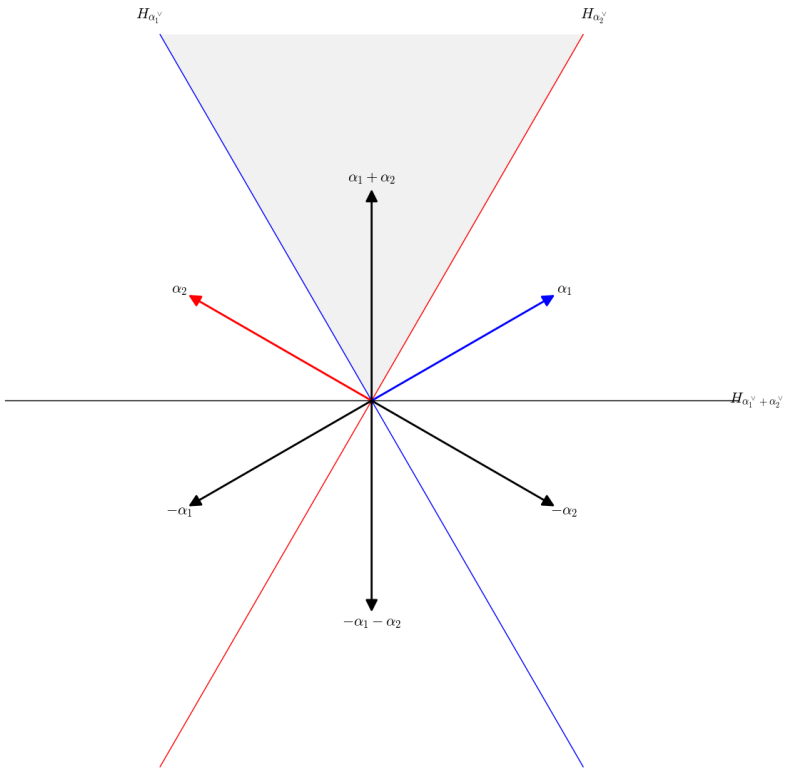
- Δ est une base de $Vect(\phi)$
- Toute racine $\beta \in \phi$ s'écrit sous la forme $\beta = \sum_{\alpha \in \Delta} k_\alpha \alpha$ avec tous les coefficients k_α de même signe

On appelle *racines simples* les racines appartenant à Δ, et on appelle parfois *système simple* une base de système de racines.

Remarque. Dans le cas des systèmes de racines de la théorie de Lie, on impose en plus le fait que les coefficients k_α soient entiers.

Exemple 2.6. Reprenons les exemples précédents :

1. En reprenant $W = A_2$, on peut choisir 1 et j comme base du système de racines associé à W.
   Pour prendre le même exemple de manière un peu plus abstraite, dans le plan euclidien classique $\mathbb{R}^2$, le système suivant appelé A_2 (ce nom fera exactement sens plus tard) est un système de racines de base $\Delta = \{\alpha_1, \alpha_2\}$ et de racines positives $\Pi = \{\alpha_1, \alpha_2, \alpha_1 + \alpha_2\}$.



Le système de racines A_2 dans $\mathbb{R}^2$

2. De manière générale, le système de racines A_n vu précédemment admet comme base $\Delta = \{\alpha_i | \alpha_i = \varepsilon_i - \varepsilon_{i+1}, 1 \leq i \leq n\}$.

Avant de montrer que tout système de racines admet une base, et de les caractériser, on montre le lemme suivant :

Lemme 2.2. *Si Δ est une base de système de racines, alors pour toutes racines $\alpha, \beta \in \Delta$ telles que $\alpha \neq \beta$ on a $(\alpha, \beta) \leq 0$.*

Preuve. Supposons que pour $\alpha, \beta \in \Delta$, on ait $(\alpha, \beta) > 0$.
On a alors $s_\alpha(\beta) = \beta - 2\frac{(\beta,\alpha)}{(\alpha,\alpha)}\alpha$. Par hypothèse on a donc $c := 2\frac{(\beta,\alpha)}{(\alpha,\alpha)} > 0$. Comme $s_\alpha(\beta) \in \phi$, alors $s_\alpha(\beta)$ est soit une racine positive, soit négative.

— Si $s_\alpha(\beta) > 0$, décomposons-la : $s_\alpha(\beta) = \sum_{\delta\in\Delta} k_\delta\delta$ avec tous les $k_\delta \geq 0$.
  — Si $k_\beta < 1$, on a $s_\alpha(\beta) = \beta - c\alpha = k_\beta\beta + \sum_{\delta\in\Delta-\{\beta\}} k_\delta\delta$.
  On obtient donc que $(1-k_\beta)\beta = (k_\alpha + c)\alpha + \sum_{\delta\in\Delta-\{\alpha,\beta\}} k_\delta\delta$, or $k_\alpha + c > 0$. On peut donc écrire $(1-k_\beta)\beta$ comme une combinaison linéaire positive d'éléments de $\Delta - \{\beta\}$, or comme $k_\beta < 1$, on a $(1-k_\beta) > 0$, donc β peut s'écrire comme combinaison linéaire positive d'éléments de $\Delta - \{\beta\}$, ce qui est absurde.
  — Si $k_\beta \geq 1$, on a toujours $\beta - c\alpha = k_\beta\beta + \sum_{\delta\in\Delta-\{\beta\}} k_\delta\delta$, ce qui implique $0 = (k_\beta - 1)\beta + (k_\alpha + c)\alpha + \sum_{\delta\in\Delta-\{\alpha,\beta\}} k_\delta\delta$, or 0 ne peut pas être combinaison linéaire positive avec certains termes > 0. C'est donc également absurde.
— Si $s_\alpha(\beta) < 0$, décomposons-la : $s_\alpha(\beta) = \sum_{\delta\in\Delta} k_\delta\delta$ avec tous les $k_\delta \leq 0$. Distinguons alors les deux cas suivants :

  — Si $k_\alpha + c \leq 0$, on a alors :

$$\beta - c\alpha = k_\alpha\alpha + k_\beta\beta + \sum_{\delta\in\Delta-\{\alpha,\beta\}} k_\delta\delta$$
$$(1-k_\beta)\beta = (k_\alpha + c)\alpha + \sum_{\delta\in\Delta-\{\alpha,\beta\}} k_\delta\delta$$

  On a bien $k_\beta \neq 1$ car k_β est supposé négatif, donc on peut diviser cette expression par k_β et obtenir que β est combinaison linéaire de $\Delta - \{\beta\}$ avec des coefficients de même signe. Ce qui contredit la minimalité de Δ et est donc absurde.
  — Si $k_\alpha + c > 0$, comme $c < 0$, on a $k_\alpha = k_\alpha + c - c > 0$, ce qui est absurde.

L'absurdité de tous les cas considérés nous montre bien que l'on doit avoir $(\alpha, \beta) \leq 0$.

Remarque. La preuve donnée ici n'est certainement pas celle qui utilise le mieux la définition de système simple, on peut la raccourcir facilement. Mais elle permet en réalité de voir que cette condition est vraie dans le cas où toute racine s'écrit comme combinaison linéaire de même signe d'éléments de notre sous-ensemble lorsqu'il est supposé minimal, ce qui est toujours le cas étant donné que l'on veut que $Vect(\Delta) = Vect(\phi)$ et que Δ soit une base de $Vect(\phi)$.

Théorème 2.3. *1. Toute base de système de racines de ϕ définit de manière unique des racines positives pour ϕ.*

2. Tout ensemble de racines positives de ϕ contient une unique base de système de racines.

Preuve. 1. Soit Δ une base de ϕ. Commençons par montrer l'unicité de cet ensemble de racines positives : supposons que $\Delta \subset \Pi$ avec Π un ensemble de racines positives de ϕ. Toutes les racines de ϕ s'écrivent donc $\beta = \sum_{\alpha\in\Delta} k_\alpha\alpha$ avec tous les coefficients k_α de

même signe.
Toutes les racines β pour lesquels les k_α sont positifs sont donc positives, et bien entendu $-\beta$ est négative.
Il y a donc correspondance entre les racines à coefficients positifs (resp négatifs) et les racines positives (resp négatives) de ϕ.
Il est facile de voir qu'un tel système positif existe, considérons la base Δ de $Vect(\phi) \subset E$, complétons-la en une base Δ' de E et considérons la relation d'ordre (telle que définie plus haut) engendrée par les $\alpha > 0$ pour tout vecteur de Δ'. On obtient donc un ensemble de racines positives Π tel que $\Delta \subset \Pi$.

2. Tout d'abord, montrons que si un ensemble Π de racines positives contient une base Δ, alors cette base est unique.
En raisonnant par l'absurde supposons qu'il existe une autre base $\Delta' \subset \Pi$. Considérons $\beta \in \Delta'$ tel que $\beta \notin \Delta$, on peut écrire $\beta = \sum_{\alpha\in\Delta} k_\alpha\alpha$ avec $k_\alpha \geq 0$. Fixons $\delta \in \Delta$ tel que $k_\delta \neq 0$, on obtient :
$$k_\delta\delta = (\textstyle\sum_{\alpha\in\Delta-\{\delta\}}(-k_\alpha)\alpha) + \beta$$
Or comme tous les $\alpha \in \Delta$ sont des racines positives, on peut écrire δ comme combinaison linéaire d'éléments de Δ' à coefficients non tous positifs en remplaçant dans l'expression précédente. Or Δ' étant en particulier une base d'espace vectoriel, l'écriture de δ comme combinaison linéaire d'éléments de Δ' est unique, et δ n'est pas une racine positive.
C'est absurde, donc tout ensemble de racines positives Π contient au plus une base.

Montrons maintenant que tout système positif contient une base :
Considérons un sous-ensemble minimal Δ de Π tel que tout élément de Π s'écrit comme combinaison positive d'éléments de Δ. D'après la preuve du lemme précédent, pour tout $\alpha, \beta \in \Delta$, on a $(\alpha, \beta) \leq 0$.
En raisonnant par l'absurde, supposons que Δ ne soit pas une base de ϕ : étant donné qu'il est évident que Δ est génératrice, supposons donc qu'elle ne soit pas libre :
Posons $\sum_{\alpha\in\Delta} k_\alpha\alpha = 0$ avec $k_\alpha \neq 0$ pour certains $\alpha \in \Delta$.
Séparons les racines pour lesquelles le coefficient est non nul et notons $\sigma = \sum k_\beta\beta = \sum -k_\gamma\gamma$ où les β sont les racines de Δ pour lesquelles $k_\beta > 0$ et les γ sont celles pour lesquelles $k_\gamma < 0$. On a $\sigma > 0$. D'après le lemme précédent, par bilinéarité du produit scalaire et par positivité de σ (au sens de l'ordre associé), on a :
$$(\sigma, \sigma) = (\sum k_\beta\beta, \sum k_\gamma\gamma) \leq 0$$
Or, on a $(\sigma, \sigma) \geq 0$, donc $(\sigma, \sigma) = 0$ et donc $\sigma = 0$. Ce qui contredit $\sigma > 0$.
Donc Δ est libre, et est donc une base de $Vect(\phi)$.

Tout ce travail sur les bases de systèmes de racines a pour but de montrer qu'en réalité, si W est engendré par les réflexions par rapport aux racines, on peut encore réduire l'ensemble des générateurs aux *réflexions simples*, c'est-à-dire les réflexions associées à un ensemble de racines simples de ϕ. Ces générateurs et leurs relations induisent de nombreuses propriétés très puissantes qui nous aideront dans la résolution du problème du mot dans les groupes de Coxeter en général.

Avant de montrer ce théorème, nous aurons besoin par la suite du lemme suivant :

Lemme 2.4. *Si $\alpha \in \Delta$, alors $s_\alpha(\Pi - \{\alpha\}) = \Pi - \{\alpha\}$*

Preuve. Soit $\beta \in \Pi$ tel que $\beta \neq \alpha$, et écrivons $\beta = \sum_{\gamma \in \Delta} c_\gamma \gamma$.
On a pour tout γ, $c_\gamma \geq 0$, et comme les seuls multiples scalaires de α dans ϕ sont α et $-\alpha$, alors il existe $\gamma \neq \alpha$ tel que $c_\gamma > 0$. En appliquant s_α des deux côtés de l'expression de β, on obtient $s_\alpha(\beta) = \beta - c\alpha$ avec $c \in \mathbb{R}$, $s_\alpha(\beta)$ est une combinaison linéaire de racines simples, avec les coefficients c_γ identiques à ceux de β. Or une expression de ce genre nous donne tous les coefficients de même signe, donc $s_\alpha(\beta)$ doit être une racine positive, et elle ne peut pas être α au risque d'obtenir la contradiction suivante : $s_\alpha s_\alpha(\beta) = s_\alpha(\alpha) = -\alpha$ qui est une racine négative.

Définition 2.7. Soit $\alpha \in \phi$, on définit la *hauteur* de la racine $\alpha = \sum_{\delta \in \Delta} k_\delta \delta$ par rapport à la base Δ, que l'on note $ht(\alpha)$, le nombre réel $\sum_{\delta \in \Delta} k_\delta$.

Théorème 2.5. *Si Δ est une base de ϕ, alors W est engendré par les réflexions s_α telles que $\alpha \in \Delta$.*

Preuve. Notons W' le sous-groupe de W engendré par les s_α définies ci-dessus, et montrons que $W' = W$:

1. Commençons par montrer que l'orbite par l'action de W' des racines simples contient Π : Soit $\beta \in \Pi$, considérons l'ensemble $W'(\beta) \cap \Pi$. Cet ensemble est non-vide (il contient au moins β). Notons γ la plus petite racine de cet ensemble au sens de la hauteur, et montrons que γ est une racine simple :
   Étant donné que $\gamma \in \Pi$, on peut écrire $\gamma = \sum_{\alpha \in \Delta} k_\alpha \alpha$ avec $k_\alpha \geq 0$. On a $0 \leq (\gamma, \gamma) = \sum_{\alpha \in \Delta} k_\alpha (\gamma, \alpha)$. Il existe donc forcément $\alpha \in \Delta$ tel que $(\gamma, \alpha) > 0$.
   — Si $\gamma = \alpha$, nous avons ce que nous cherchons.
   — Si $\gamma \neq \alpha$, considérons $s_\alpha(\gamma) = \gamma - c\alpha$ avec $c > 0$. D'après le lemme précédent, $s_\alpha(\gamma)$ est également positive, on obtient donc $ht(s_\alpha(\gamma)) < ht(\gamma)$, ce qui contredit la minimalité de γ.
   Donc le W'-orbite de β contient forcément une racine simple, donc le W'-orbite des racines simples contient toutes les racines positives.
2. Montrons maintenant que $\phi = W'(\Delta)$. On a déjà montré que $\Pi \subset W'(\Delta)$. Supposons donc maintenant que β est négative. Il existe d'après 1. $w \in W'$ tel que $-\beta = w(\alpha)$ pour un certain $\alpha \in \Delta$. On a $(ws_\alpha)(\alpha) = w(-\alpha) = -w(\alpha) = \beta$. Or comme $w, s_\alpha \in W'$, on a bien $ws_\alpha \in W'$ et donc $-\beta \in W'(\Delta)$.
3. Enfin, considérons n'importe quelle réflexion s_β avec $\beta \in \phi$. D'après 2., on peut toujours écrire $\beta = w(\alpha)$ avec $w \in W'$ et $\alpha \in \Delta$. D'après le lemme 2.1, on a $ws_\alpha w^{-1} = s_\beta$. Donc tout générateur s_β de W est un élément de W'.

Exemple 2.8. En reprenant l'exemple général de A_n, on obtient donc qu'il est engendré par les réflexions associées aux racines simples, déterminons-les :
On a besoin d'avoir pour tout $1 \leq i \leq n$:

$$s_i(\sum a_j \varepsilon_j) = \sum_{j \neq i, i+1} a_j \varepsilon_j + a_i \varepsilon_{i+1} + a_{i+1} \varepsilon_i$$

C'est-à-dire $s_i = (i(i+1))$.
Et on obtient bien que S_{n+1} est engendré par les transpositions $(i(i+1))$, ce qui est parfaitement cohérent.

2.2 Mots et réduction des mots

Commençons notre étude explicite du problème du mot en le considérant dans les groupes de réflexions réels. Dans toute cette partie, soit W un groupe de réflexions réel, considérons les

mots sur l'ensemble des réflexions simples. Nous voulions commencer par réduire les mots jusqu'à trouver un mot réduit.
Avant de donner un critère explicite de réduction, nous avons besoin d'introduire quelques définitions et lemmes :

Définition 2.9. Soit $w \in W$, on note l l'application $l : W \to \mathbb{N}$ telle que $l(w)$ est égal au plus petit entier r tel que w peut s'écrire $w = s_1 \cdots s_r$ avec s_i une réflexion simple pour tout i. On appelle cette fonction la fonction longueur du mot w.

Un critère de réductibilité très simple à utiliser serait de pouvoir déterminer à partir d'un élément quelconque de W sa longueur, afin de pouvoir commencer à traiter le cas où w est réduit et pouvoir donner une condition d'arrêt à la réduction.

Or, nous allons pouvoir donner une méthode générale simple de calcul : définissons $n : W \to \mathbb{N}$ telle que $n(w) := \text{Card}(\phi^+ \cap w^{-1}(\phi^-))$.
Géométriquement, on a $n(w) = \text{Card}(\{\alpha \in \phi^+ | w(\alpha) \in \phi^-\})$ pour tout $w \in W$.
Étant donné que nous cherchons à résoudre algorithmiquement un problème, nous illustrerons ce mémoire tout le long avec des morceaux d'un programme implémentant cet algorithme avec Sage (SageMath depuis peu). Commençons donc avec cette fameuse fonction n qui nous sera utile pour la suite :

```
def n(sigma, W) :
    pi = W.positive_roots()
    minus_pi = W.roots()[len(pi):]
    res = []
    for alpha in range(len(pi)) :
        beta = W.roots()[sigma.action_on_root_indices(alpha)]
        if beta in minus_pi :
            res += [alpha]
    return len(res)
```

Cette fonction prend en argument un élément `sigma` du groupe et renvoie son image par la fonction n décrite précédemment. Elle calcule cette image à l'aide de son interprétation géométrique en calculant l'intersection des racines positives du système et de l'image réciproque par `sigma` des racines envoyées sur des racines négatives.

Lemme 2.6. *Soit $\alpha \in \Delta, w \in W$:*

1. $w(\alpha) \in \phi^+ \implies n(ws_\alpha) = n(w) + 1$
2. $w(\alpha) \in \phi^- \implies n(ws_\alpha) = n(w) - 1$
3. $w^{-1}(\alpha) \in \phi^+ \implies n(s_\alpha w) = n(w) + 1$
4. $w^{-1}(\alpha) \in \phi^- \implies n(s_\alpha w) = n(w) - 1$

Preuve. On pose $\prod(w) = \phi^+ \cap w^{-1}(\phi^-)$. On a donc $n(w) = \text{Card}(\prod(w))$. Si $w(\alpha) \in \phi^+$, on remarque à l'aide du lemme 2.4 que $ws_\alpha(\phi^+ - \{\alpha\}) = w(\phi^+ - \{\alpha\})$ car s_α permute les racines simples autres que α et que $ws_\alpha(\alpha) = w(-\alpha) = -w(\alpha) \in \phi^-$. Donc $\prod(ws_\alpha)$ est l'union disjointe de $s_\alpha(\prod(w))$ et de $\{\alpha\}$.
De même, si $w(\alpha) \in \phi^-$, on remarque à l'aide du lemme 2.4 que $ws_\alpha(\phi^+ - \{\alpha\}) = w(\phi^+ - \{\alpha\})$ car s_α permute les racines simples autres que α et que $ws_\alpha(\alpha) = w(-\alpha) = -w(\alpha) \in \phi^+$. Donc on a bien $\prod(ws_\alpha) = s_\alpha(\prod(w)) - \{\alpha\}$.
Enfin, pour les deux dernières, on applique les propriétés précédentes à w^{-1}, on obtient alors que si $w^{-1}(\alpha) \in \phi^+$, alors $n(w^{-1}s_\alpha) = n(w^{-1}) + 1$ et que si $w^{-1}(\alpha) \in \phi^-$, alors $n(w^{-1}s_\alpha) =$

$n(w^{-1})-1$. Comme par une propriété immédiate de la fonction longueur, la longueur de tout mot dans W est égale à la longueur de son inverse, on en déduit respectivement que $n(s_\alpha w) = n(w)+1$ et $n(s_\alpha w) = n(w) - 1$.

Le théorème suivant est un théorème crucial dans la théorie des groupes de Coxeter, nous verrons plus tard que tout groupe engendré par des involutions satisfaisant l'énoncé 3., aussi appelé *condition de délétion* est un groupe de Coxeter.

Théorème de condition de délétion 2.7. *Soit $w = s_1 \cdots s_r$ un mot dans W. On note $s_i = s_{\alpha_i}$ en s'autorisant les répétitions, et on suppose que $n(w) < r$.*
Alors il existe $i, j \in \mathbb{N}$, $0 \leq i < j \leq r$ tels que :

1. $\alpha_i = (s_{i+1} \cdots s_{j-1})(\alpha_j)$
2. $s_{i+1}s_{i+2} \cdots s_j = s_i s_{i+1} \cdots s_{j-1}$
3. $w = s_1 \cdots \widehat{s_i} \cdots \widehat{s_j} \cdots s_r$ *(on considère que $\widehat{s_k}$ signifie l'omission de s_k)*

Preuve.
1. Comme $n(w) < r$, le lemme 2.6 nous montre que pour un certain $j \leq r$, on a $(s_1 \cdots s_{j-1})(\alpha_j) < 0$. Comme $\alpha_j > 0$, il existe $i \leq j-1$ tel que $s_i(s_{i+1} \cdots s_{j-1}(\alpha_j)) > 0$. En utilisant le lemme 2.4 avec s_i, on voit que la racine positive renvoyée sur une racine négative est forcément α_i. On obtient donc l'égalité cherchée.
2. Posons $\alpha = \alpha_j$ et $w' = s_{i+1} \cdots s_{j-1}$ tels que $w'(\alpha) = \alpha_i$, qui existe d'après (a).
Par le lemme 2.1, on a $ws_\alpha w^{-1} = s_{w'(\alpha)} = s_i$.
On obtient donc $w's_\alpha w^{-1}w = s_i w$, soit $s_{i+1} \cdots s_j = s_i \cdots s_{j-1}$.
3. Cette expression s'obtient facilement à partir de (b), il suffit de multiplier à droite de chaque côté par s_j, on obtient alors $s_{i+1} \cdots s_j s_j = s_i \cdots s_j$, et comme $s_j^2 = Id_E$, on obtient bien en remplaçant dans l'expression de w l'identité voulue.

Remarque. Dans notre preuve du théorème de condition de délétion, nous avons uniquement utilisé le fait que W soit un groupe finiment engendré par des réflexions dans un espace vectoriel réel de dimension finie muni d'une forme bilinéaire symétrique. Cette remarque peut sembler anodine, mais elle sera importante dans la suite afin de généraliser certains résultats à tous les groupes de Coxeter.

Exemple 2.10. Continuons avec l'exemple de A_2, on prend $S = \{s_1, s_2\}$ les réflexions simples de A_2 et on considère le mot $w = s_2s_1s_2s_1s_2s_1$. On a $w = 1$, ce que l'on peut retrouver en appliquant trois fois le théorème :

1. Commençons par l'appliquer une première fois, on trouve $i = 1$ et $j = 4$, on obtient donc $w = s_2s_1s_2s_1s_2s_1 = \widehat{s_2}s_1s_2\widehat{s_1}s_2s_1$.
2. Ensuite, en réappliquant le critère on trouve de nouveau $i = 1$ et $j = 4$, c'est-à-dire $w = s_1s_2s_2s_1 = \widehat{s_1}s_2s_2\widehat{s_1}$.
3. Enfin, en l'appliquant une dernière fois on obtient $i = 1$ et $j = 2$, donc $w = \widehat{s_2}\widehat{s_2} = 1$.

Il paraîtrait approprié d'illustrer les conséquences de ce théorème sur notre projet en montrant le morceau de code associé, ce que nous allons faire immédiatement après avoir introduit les fonctions auxiliaires suivantes qui sont utilisées dans le programme :

```
def associate_root(sigma,W):
    n = len(W.positive_roots())
    for alpha in range(n):
        k = sigma.action_on_root_indices(alpha)
        if W.roots()[k] == - W.positive_roots()[alpha] :
```

```
            return k - n
    return -1

def constructPartialSigma(sigma, i, j, W):
    w = W.one()
    for s in sigma[i+1:j]:
        w = w * s
    return w
```

La fonction `associate_root` prend en argument une réflexion `sigma` du groupe W et nous renvoie l'indice attribué par Sage à la racine positive associée à la réflexion que nous avons, en se rappelant que cette racine α est la seule racine positive renvoyée par `sigma` sur $-\alpha$; tandis que `constructPartialSigma` prend en argument une liste `sigma` de réflexions simples représentant un mot sur ces dernières et le groupe W, elle permet simplement de construire le sous-mot $s_{i+1} \cdots s_{j-1}$ comme élément du groupe afin d'observer son action sur les racines.
À partir de tout ce que l'on a, nous pouvons maintenant donner une fonction permettant d'appliquer le théorème de condition de délétion :

```
def deletionConditionTheorem(sigma,W):
    w = W.one()
    for s in sigma :
        w = w * s
    l = n(w,W)
    if l == len(sigma):
        return gen_to_indices(sigma,W)
    else :
        sigma2 = sigma.copy()
        for j in range(1,len(sigma2),1):
            alpha = associate_root(sigma2[j],W)
            for i in range(j):
                w = constructPartialSigma(sigma2, i, j, W)
                alphaI = w.action_on_root_indices(alpha)
                if associate_root(sigma2[i],W) == alphaI:
                    del(sigma2[j])
                    del(sigma2[i])
                    return gen_to_indices(sigma2,W)
        return -1
```

Cette fonction prend en argument le mot `sigma` représenté comme une liste de réflexions simples et le groupe W dans lequel nous travaillons, et applique simplement le critère du théorème : c'est-à-dire que si n(`sigma`) est inférieure aux nombres de lettres de `sigma`, elle recherche deux indices i et j tels que l'image par $s_{i+1} \cdots s_{j-1}$ de la racine α_j associée à s_j soit la racine α_i associée à s_i, et retire donc ces deux lettres de l'expression de σ.

De ce théorème, on peut très facilement déduire le résultat suivant, qui nous sera utile pour déterminer la longueur d'un mot de W par son action sur les racines.

Lemme 2.8. *$\forall w \in W$, $n(w) = l(w)$, où l est la fonction longueur du système de Coxeter.*

Preuve. Tout d'abord, on montre que $n(w) \leq l(w)$: en construisant l'expression réduite de tout mot $w \in W$, on observe qu'à chaque étape de la construction d'après le lemme 4.3, on a $n(s_1 \cdots s_{r+1}) = n(s_1 \cdots s_r) + 1$ ou $n(s_1 \cdots s_{r+1}) = n(s_1 \cdots s_r) - 1$ pour tout $r < l(w)$, donc que $n(s_1 \cdots s_{r+1}) \leq l(w)$.

Ensuite, le théorème précédent nous permet de réécrire $w = s_1 \cdots s_r$ comme un produit de $r-2$ réflexions si $n(w) < r$, ce qui contredit le fait que $l(w) = r$.

À partir de là, renommons la fonction n de notre programme `longueur` et donnons une fonction qui permet de réduire directement un mot w :

```
def reduction(sigma,W) :
    w = W.one()
    b = False
    sigma2 = sigma.copy()
    for s in sigma :
        w = w * s
    l = longueur(w,W)
    while len(sigma2) > l :
        j = 1
        while j < len(sigma2) and not(b):
            alpha = associate_root(sigma2[j],W)
            i = 0
            while i < j and not(b):
                w = constructPartialSigma(sigma2, i, j, W)
                alphaI = w.action_on_root_indices(alpha)
                if associate_root(sigma2[i],W) == alphaI:
                    del(sigma2[j])
                    del(sigma2[i])
                    b = True
                i += 1
            j += 1
        b = False
    return sigma2
```

Cette fonction fonctionne exactement comme la fonction `deletionConditionTheorem`, sauf qu'elle applique ce même critère en boucle tant que la taille de la liste `sigma2` est strictement supérieure à n(`sigma`).

3 Groupes de réflexions et systèmes de Coxeter

Dans cette partie, nous montrerons la correspondance entre groupes de réflexions réels finis et groupes de Coxeter finis et l'appliquerons au problème du mot dans les groupes de Coxeter, en particulier les groupes de Coxeter finis.

3.1 Correspondance des groupes finis

Avant toute chose, nous devons commencer par montrer que tout groupe de réflexion fini est un groupe de Coxeter. Ce n'est pas une chose aisée en général de montrer qu'un groupe admet une présentation par relations telle que cette présentation en fait un système de Coxeter, mais il se trouve qu'un critère bien utile nous permettra de caractériser entièrement les groupes de Coxeter par une propriété déjà vue des groupes de réflexions.

Théorème 3.1. *Soit W un groupe et S un ensemble d'involutions engendrant W. Si (W, S) respecte la condition de délétion, c'est-à-dire est tel que pour tout $w = s_1 \cdots s_r (s_k \in S)$ avec*

$r < l(w)$, il existe $i < j \in \mathbb{N}$ tels que $w = s_1 \cdots \widehat{s_i} \cdots \widehat{s_j} \cdots s_r$, il s'agit alors d'un système de Coxeter.

Preuve. Afin de montrer que (W, S) est un système de Coxeter, on montrera que toute relation de la forme $s_1 \cdots s_r = 1$ entre les éléments de S est conséquence de relations de tresse.
Remarquons tout d'abord que r doit être *pair* étant donné que $l(1) = 0$ et que la condition de suppression implique que l'on puisse toujours retirer un nombre *pair* de lettres.
Procédons par récurrence forte sur $q \geq 1$ tel que $r = 2q$:

— Pour le cas $r = 2$, on a la relation $s_1 s_2 = 1$, ce qui implique $s_2 = s_1^{-1} = s_1$, qui découle de la relation de tresse $s_1^2 = 1$.

— Supposons l'assertion vraie pour q, montrons qu'elle l'est également pour $q+1$: considérons une relation quelconque sur r lettres.

$$s_1 ... s_r = 1 \tag{R1}$$

On peut réécrire (R1) sous la forme équivalente suivante :

$$s_1 ... s_{q+1} = s_r ... s_{q+2} \tag{1}$$

Comme l'expression à droite de (1) est sur $q - 1$ lettres, l'expression à gauche n'est pas réduite et on peut lui appliquer la condition de délétion. On obtient donc deux indices i, j tels que $1 \leq i < j \leq q + 1$ tels que :

$$s_i ... s_{j-1} = s_{i+1} ... s_j \tag{2}$$

On réécrit (2) sous la forme suivante :

$$s_i s_{i+1} ... s_j s_{j-1} ... s_{i+1} = 1 \tag{3}$$

Si (3) implique strictement moins de r lettres, alors elle dérive des relations de tresses d'après l'hypothèse de récurrence, on remplace dans (R1) :

$$s_1 ... s_i (s_i ... s_{j-1}) s_{j+1} ... s_r = 1$$

$$s_1 ... \widehat{s_i} ... \widehat{s_j} ... s_r = 1$$

Ainsi, par l'hypothèse de récurrence, on voit que cette équation, et donc que (R1), est conséquence des relations de tresse.
En revanche, si (3) implique r lettres, c'est-à-dire si on obtient :

$$s_1 ... s_q = s_2 ... s_{q+1} \tag{4}$$

considérons la relation suivante, équivalente à (R1) :

$$s_2 ... s_r s_1 = 1 \tag{R2}$$

Essayons d'appliquer la même preuve à (R2), considérons la relation suivante :

$$s_2 ... s_{q+2} = s_1 ... s_r s_{q+3}$$

et appliquons-lui la condition de délétion comme précédemment, on voit que le procédé fonctionne toujours sauf si on obtient

$$s_2 ... s_{q+1} = s_3 ... s_{q+2} \tag{5}$$

Si on a (4), alors on réessaye avec

$$s_3...s_r s_1 s_2 = 1 \tag{R3}$$

et ainsi de suite. Si aucune de ces relations ne fonctionne, il faut changer de tactique. Supposons donc que ce soit le cas et reprenons notre relation (5), que l'on multiplie à gauche par $s_3 s_2$ puis à droite par $s_{q+1}...s_4$ pour obtenir

$$s_3 s_2 (s_3...s_{q+2}) s_{q+1}...s_4 = 1 \tag{5b}$$

De la même manière, réécrivons le produit de r lettres à gauche (5b) sous la forme suivante pour appliquer la condition de délétion à gauche :

$$s_3 s_2 s_3...s_{q+1} = s_4...s_{q+2}$$

De nouveau, la condition de délétion nous donne ce que nous voulons sauf si on obtient :

$$s_2...s_{q+1} = s_3 s_2 s_3...s_q \tag{6}$$

Par (4), on obtient donc :

$$s_1...s_q = s_3(s_2...s_q)$$

Cette dernière égalité nous donne donc $s_1 = s_3$. On peut également montrer que $s_2 = s_4$ en considérant la relation suivante :

$$s_4...s_{q+3} = s_3...s_{q+2}$$

qui est vraie par hypothèse, la relation (R3) n'ayant pas abouti à ce que l'on souhaitait plus haut. En multipliant à gauche par $s_4 s_3$ et à droite par $s_{q+2}...s_5$, on obtient :

$$s_4 s_3 (s_4...s_{q+3}) s_{q+2}...s_5 = 1$$

Appliquer la condition de délétion de la même manière fonctionne sauf si on obtient

$$s_4 s_3 (s_4...s_{q+2}) = s_3 s_4...s_{q+3}$$

Or par hypothèse, on a $s_3...s_{q+3} = s_2...s_{q+2}$, donc on a :

$$s_4 s_3 s_4...s_{q+2} = s_2...s_{q+2}$$

On obtient donc $s_2 = s_4$, en répétant ce procédé autant qu'il y a de lettres, si on ne peut jamais appliquer notre méthode du départ, on obtient alors $s_1 = s_3 = ... = s_{r-1}$ et $s_2 = s_4 = s_r$ (ce qui a du sens car r est pair). Donc (R1) est une relation de la forme :

$$(s_\alpha s_\beta)^{\frac{r}{2}} = 1$$

Et (R1) est donc une relation de tresse.

Corollaire 3.2. *Tout groupe de réflexions réel fini est un groupe de Coxeter.*

Preuve. Tout groupe de réflexions réel fini est engendré par des réflexions, donc par des involutions, et d'après le théorème de condition de délétion, ces groupes respectent la condition de délétion. Par le théorème précédent, ce sont des groupes de Coxeter.

Nous avons donc finalement montré que tout groupe de réflexions fini est un groupe de Coxeter, nous allons maintenant pouvoir montrer que tout groupe de Coxeter fini admet une représentation fidèle comme groupe de réflexions fini dans un espace euclidien, ce qui nous permettra de généraliser notre critère de réduction des mots à tous les groupes de Coxeter finis.

Afin de définir la représentation géométrique d'un groupe de Coxeter W, plaçons-nous dans un $\mathbb{R}$-espace vectoriel V muni d'une base $(\alpha_s)_{s\in S}$ et définissons la forme bilinéaire B_W suivante sur V :

— Si $m(s,s') < \infty : B_W(\alpha_s, \alpha_{s'}) = -\cos\frac{\pi}{m(s,s')}$

— Si $m(s,s') = \infty : B_W(\alpha_s, \alpha_{s'}) = -1$

À partir de cette forme bilinéaire (qui est symétrique car $m(s,s') = m(s',s)$), on peut définir pour tout $s \in S$ une réflexion σ_s de la manière suivante :

$$\sigma_s(\lambda) = \lambda - 2B_W(\alpha_s, \lambda)\alpha_s$$

Définition 3.1. Soit W un groupe de Coxeter, auquel est associé la forme bilinéaire B dans le $\mathbb{R}$-espace vectoriel V. On appelle *représentation géométrique* de W le morphisme $\rho : W \to GL(V)$ qui à s associe la réflexion σ_s.

Dans le cas général, cette représentation n'est pas un groupe de réflexion au sens exact de ce que nous avons précédemment étudié, en effet V n'est un espace euclidien que si W est fini, ce que nous montrerons plus tard. Elle reste tout de même très intéressante à étudier dans le cas générale, car cette représentation a l'avantage d'être fidèle, et c'est exactement ce que l'on va montrer maintenant.

Lemme 3.3. *Soit $V_{s,s'} := \mathbb{R}\alpha_s \oplus \mathbb{R}\alpha_{s'}$. Si $m(s,s') < \infty$, la restriction de B à $V_{s,s'}$ est un produit scalaire et $\langle s, s'\rangle$ est le groupe diédral $I_2(m(s,s'))$.*

Preuve. Le fait que $\langle s, s'\rangle$ soit le groupe diédral d'ordre $2m(s,s')$ découle directement de sa définition en tant que groupe de réflexions réel engendré par 2 réflexions dans le plan euclidien. Il reste à montrer que B est un produit scalaire.
Par hypothèse B est déjà une forme bilinéaire symétrique, il suffit de voir qu'elle est définie positive lorsque restreinte à $V_{s,s'}$:

— Prenons $\lambda = a\alpha_s + b\alpha_{s'}$ $(a, b \in \mathbb{R})$, posons $m = m(s,s')$ et calculons $B(\lambda,\lambda)$

$$\begin{aligned}
B(\lambda,\lambda) &= aB(\alpha_s,\lambda) + bB(\alpha_{s'},\lambda)\\
&= a^2B(\alpha_s,\alpha_s) + 2abB(\alpha_s,\alpha_{s'}) + b^2B(\alpha_{s'},\alpha_{s'})\\
&= a^2 - 2ab\cos(\frac{\pi}{m}) + b^2\\
&= a^2 - 2ab\cos(\frac{\pi}{m}) + b^2(\cos^2(\frac{\pi}{m}) + \sin^2(\frac{\pi}{m}))\\
&= a^2 - 2ab\cos(\frac{\pi}{m}) + b^2\cos^2(\frac{\pi}{m}) + b^2\sin^2(\frac{\pi}{m})\\
&= (a + b\cos(\frac{\pi}{m}))^2 + b^2\sin^2(\frac{\pi}{m})
\end{aligned}$$

Cette expression est très clairement positive, et non nulle pour tout vecteur non nul si et seulement si $\sin(\frac{\pi}{m}) \neq 0$: si $b \neq 0$, $b^2\sin^2(\frac{\pi}{m})$ est nul à cette condition, et si $b = 0$, on a $B(\lambda,\lambda) = a^2$, qui est donc nul si et seulement si $a = 0$, i.e $\lambda = 0$.
Or $\sin(\frac{\pi}{m}) = 0 \iff m = \infty$.
Donc la restriction de B à $V_{s,s'}$ est un produit scalaire si et seulement si $m(s,s') < \infty$.

À partir de maintenant, on se permet d'identifier $w \in W$ à son image par la représentation géométrique, et on définit le *système de racines* ϕ associé à W comme l'ensemble des vecteurs appartenant à l'orbite des α_s par l'action de W.

Théorème 3.4. *Soit $w \in W$ et $s \in S$. Si $l(ws) > l(w)$, alors $w(\alpha_s) > 0$, et si $l(ws) < l(w)$ alors $w(\alpha_s) < 0$.*

Preuve. Avant de démontrer la première partie du théorème, remarquons que la deuxième en découle naturellement : soit $w \in W$ et $s \in S$, si $l(ws) < l(w)$, on a $l((ws)s) > l(ws)$, i.e $ws(\alpha_s) > 0$ donc $w(-\alpha_s) > 0$, ce qui signifie $w(\alpha_s) < 0$.
Prouvons maintenant la première partie du théorème, on procède par récurrence sur $l(w)$:

— Commençons avec le cas $l(w) = 0$, on a alors $w = 1$.
$\forall s \in S$, on a $l(s) = 1 > l(1) = 0$, et $1(\alpha_s) = \alpha_s > 0$ car α_s est une racine simple.

— Ensuite, si $l(w) > 1$: soit $s \in S$ tel que $l(ws) > l(w)$. Comme $l(w) > 1$, il existe $s' \in S$ tel que $l(ws') < l(w)$, il suffit de considérer une expression réduite de w et de prendre la dernière réflexion simple de cette expression.

Considérons $I = \{s, s'\}$ et notons W_I le groupe diédral engendré par I, notons l_I sa fonction longueur. (On remarque immédiatement que partout où l_I est définie, on a $l_I \geq l$).

Considérons également $A = \{v \in W | v^{-1}w \in W_I \text{ et } l(v) + l_I(v^{-1}w) = l(w)\}$: on remarque d'abord que A est non vide puisque $w \in A$. On peut donc prendre $v \in A$ tel que $l(v)$ est minimal, et on note $v_I = v^{-1}w$.
On a donc $w = vv_I$ et $l(w) = l(v) + l_I(v_I)$ par définition de A.

Remarquons également que $ws' \in A$:
— $s'w^{-1}w = s' \in W_I$
— $l(ws') = l(w) - 1$
— $l_I(s') = 1$
et que $l(v) \leq l(ws')$ par minimalité de $l(v)$, donc que $l(v) < l(w)$.

Comparons $l(v)$ et $l(vs)$:
Supposons que $l(vs) < l(v)$, donc que $l(vs) = l(v) - 1$, on a

$$\begin{aligned} l(w) &\leq l(vs) + l(sv^{-1}w) \\ &\leq l(vs) + l_I(sv^{-1}w) \text{ car } l \leq l_I \\ &= l(v) - 1 + l_I(sv^{-1}w) \\ &\leq l(v) - 1 + l_I(v^{-1}w) + 1 \\ &= l(v) + l_I(v{-}1w) \\ &= l(w) \text{ car } v \in A \end{aligned}$$

Donc $l(w) = l(vs) + l(sv^{-1}w)$ et $vs \in A$, or $l(vs) < l(v)$ par hypothèse, c'est impossible par minimalité de $l(v)$.
Donc $l(vs) > l(v)$ et par l'hypothèse de récurrence, on a $v(\alpha_s) > 0$.
En supposant $l(vs') < l(v)$ et en appliquant le même raisonnement, on obtient également $l(vs') > l(v)$ i.e $v(\alpha_{s'})$ par l'hypothèse de récurrence.

Comme $w = vv_I$, on a gagné si on peut montrer que $v_I(\alpha_s)$ est une combinaison linéaire positive de α_s et $\alpha_{s'}$.
On a $l_I(v_I s) \geq l_I(v_I)$, sinon on obtiendrait :

$$\begin{aligned} l(ws) &= l(vv^{-1}ws) \\ &\leq l(v) + l(v^{-1}ws) \\ &\leq l(v) + l_I(v_I s) \text{ car } l \leq l_I \\ &< l(v) + l_I(v_I) \text{ par notre hypothèse} \\ &= l(w) \end{aligned}$$

on aurait donc $l(ws) < l(w)$, ce qui contredit notre choix pour s.
On en déduit donc assez facilement que tout expression réduite pour v_I est un produit alterné de s et s' qui finit en s' : si une expression réduite pour v_I finissait en s, on aurait $l_I(v_I s) < l_I(v_I)$, or nous venons de montrer le contraire.

Considérons les deux cas suivants :
— $m(s, s') = \infty$:
Montrons par récurrence sur $l_I(v_I)$ que si $l_I(v_I)$ est pair, alors $v_I(\alpha_s) = l_I(v_I)\alpha_{s'} + (l_I(v_I) + 1)\alpha_s$; et que si $l_I(v_I)$ est impair, alors $v_I(\alpha_s) = l_I(v_I)\alpha_s + (l_I(v_I) + 1)\alpha_{s'}$:
— Commençons avec les cas $l_I(v_I) = 0$ et $l_I(v_I) = 1$.
Si $l_I(v_I) = 0$, on a $v_I = 1$ et $1(\alpha_s) = \alpha_s$, ce qui est ce que l'on attendait.
Si $l_I(v_I) = 1$, on a $v_I = s'$ et $s'(\alpha_s) = \alpha_s - 2B(\alpha_s, \alpha_{s'})\alpha_{s'} = \alpha_s + 2\alpha_{s'}$, ce qui est également ce que l'on attendait.
— Maintenant supposons $l(v_I) > 1$:
Si $l_I(v_I)$ est pair, on a $v_I = (ss')^n$ pour un certain $n \in \mathbb{N}$ et considérons alors $v' = s'(ss')^{n-1}$, par l'hypothèse de récurrence, on a $v'(\alpha_s) = l_I(v')\alpha_s + (l_I(v') + 1)\alpha_{s'}$.
Étant donné que $v_I = sv'$, on a :

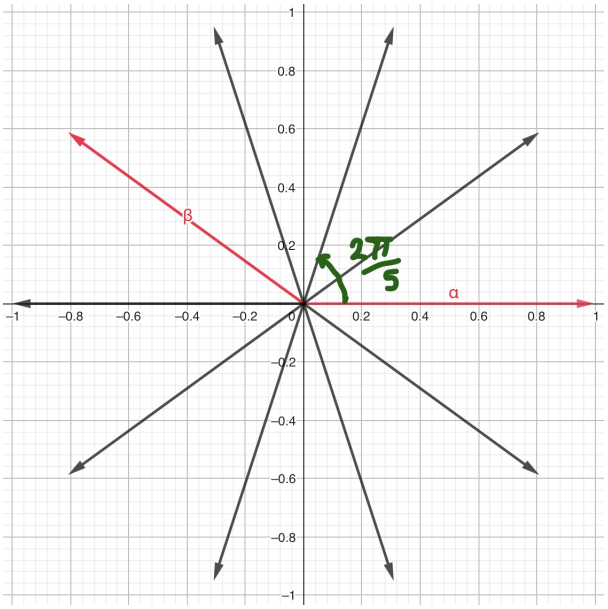
$$\begin{aligned} v_I(\alpha_s) &= s(v'(\alpha_s)) \\ &= s(l_I(v')\alpha_s + (l_I(v') + 1)\alpha_{s'}) \\ &= l_I(v')\alpha_s + (l_I(v') + 1)\alpha_{s'} - 2B(v'(\alpha_s), \alpha_s)\alpha_s \\ &= l_I(v')\alpha_s + (l_I(v') + 1)\alpha_{s'} + 2\alpha_s \\ &= (l_I(v') + 2)\alpha_s + (l_I(v') + 1)\alpha_{s'} \\ &= (l_I(v_I) + 1)\alpha_s + l_I(v_I)\alpha_{s'} \end{aligned}$$

Si $l_I(v_I)$ est impair, on a $v_I = s'(ss')^n$ pour un certain $n \in \mathbb{N}$ et considérons alors $v' = (ss')^n$, par l'hypothèse de récurrence, on a $v'(\alpha_s) = l_I(v')\alpha_{s'} + (l_I(v') + 1)\alpha_s$.
Étant donné que $v_I = s'v'$, on a :

$$\begin{aligned} v_I(\alpha_s) &= s'(v'(\alpha_s)) \\ &= s'(l_I(v')\alpha_{s'} + (l_I(v') + 1)\alpha_s) \\ &= l_I(v')\alpha_{s'} + (l_I(v') + 1)\alpha_s - 2B(v'(\alpha_s), \alpha_{s'}) \\ &= l_I(v')\alpha_s + (l_I(v') + 1)\alpha_{s'} + 2\alpha_{s'} \\ &= (l_I(v') + 2)\alpha_{s'} + (l_I(v') + 1)\alpha_s \\ &= (l_I(v_I) + 1)\alpha_{s'} + l_I(v_I)\alpha_s \end{aligned}$$

Donc l'image par v_I de α_s est bien une combinaison linéaire positive de α_s et $\alpha_{s'}$ dans ce cas.

— $m(s, s') < \infty$:
Remarquons tout d'abord que l'on a $l_I(v_I) < m(s, s') := m$: m est la plus grande valeur de longueur possible pour un tel pour sur S en conséquence de la relation de tresse $(ss')^m = 1$, et si on avait une expression réduite pour v_I de longueur m, il en existerait une se terminant avec s, ce qui est impossible.
En résumé, v_I peut s'écrire comme un produit de $< \frac{m}{2}$ facteurs ss' avec au plus un facteur s' en plus à gauche. D'après le lemme précédent, nous travaillons dans le plan euclidien avec les vecteurs α_s et $\alpha_{s'}$ unitaires, qui forment un angle de valeur $\pi - \frac{\pi}{m}$, et ss' est une rotation d'angle $\frac{\pi}{m}$ qui agit donc sur α_s par rotation vers $\alpha_{s'}$.



Le système de racines $I_2(5)$ dans $\mathbb{R}^2$

La figure précédente illustre bien la situation dans le cas $m = 5$ en identifiant $\alpha_s = \alpha$ et $\alpha_{s'} = \beta$.
La rotation $(ss')^n$ avec $n < \frac{m}{2}$ dans v_I agit donc par rotation sur α_s vers $\alpha_{s'}$ avec un angle θ tel que :

$$\theta < \frac{m}{2} \cdot \frac{2\pi}{m}$$
$$\theta < \pi$$

On est donc toujours dans le cône de positivité défini par α_s et $\alpha_{s'}$, et si ensuite v_I contient une réflexion s', on reste dans le cône puisque l'angle entre α_s et la droite stabilisée par s' est $\frac{\pi}{2} - \frac{\pi}{m}$.

Ce théorème important nous permet notamment d'énoncer un des résultats importants de notre démarche :

Corollaire 3.5. *La représentation géométrique $\sigma : W \to GL(V)$ est fidèle.*

Preuve. Soit $w \in \text{Ker } \sigma$, si $w \neq 1$, prenons une expression réduite $w = s_1...s_n$ de w et remarquons que $l(ws_n) < l(w)$. On a donc $w(\alpha_{s_n}) < 0$ par le théorème précédent. Or $w(\alpha_{s_n}) = \alpha_{s_n} > 0$ (c'est une racine simple). C'est absurde.
On a donc Ker $\sigma = \{1\}$ et σ est donc une représentation fidèle.

Remarque. Ce résultat implique que tout système de Coxeter satisfait la condition de délétion : d'après la remarque suivant notre preuve du théorème de condition de délétion, ce théorème s'applique à tout groupe engendré par un ensemble fini de réflexions dans un espace vectoriel réel muni d'une forme bilinéaire symétrique, et d'après le corollaire précédent, tout groupe de Coxeter se réalise de cette manière.

Pour terminer cette correspondance très pratique pour notre problème, il reste simplement à montrer que si W est fini, alors sa représentation géométrique est un groupe de réflexion fini. Pour cela, on doit simplement montrer que la forme B est définie positive afin d'en faire un produit scalaire.
Avant de se lancer dans cette tâche, nous devons définir un autre objet qui caractérise complètement un groupe de Coxeter :

Définition 3.2. On appelle *graphe de Coxeter* de W le graphe non-orienté dont les sommets sont les réflexions simples de W (l'ensemble de ses sommets est S) pour lequel $s, s' \in S$ sont reliés par une arête de multiplicité $m(s, s')$ si $m(s, s') > 3$ et 1 si $m(s, s') = 3$.
On dit que le système (W, S) est irréductible si son graphe de Coxeter est connexe.

Exemple 3.3. Jetons un œil à quelques exemples de graphes de Coxeter de groupes connus :

1. Le groupe diédral d'ordre $2m$, noté $I_2(m)$ comme groupe de Coxeter a pour graphe de Coxeter le graphe suivant :

2. Le groupe diédral infini, noté D_∞, a pour graphe de Coxeter le graphe suivant :

3. Le groupe symétrique sur $n+1$ éléments S_{n+1}, noté A_n comme groupe de Coxeter a pour graphe de Coxeter le graphe suivant :

Remarque. Ces graphes sont des objets particulièrement importants en théorie de Lie et en théorie des représentations, où les groupes de Coxeter apparaissent naturellement à de très nombreuses occasions, et sont en particulier des objets très utiles pour de nombreuses preuves de théorèmes de classification.

Définition 3.4. Si B est une forme bilinéaire symétrique sur V, on appelle *radical* de B l'ensemble des vecteurs orthogonaux à tous les autres, c'est-à-dire :

$$V^\perp = \{v \in V | B(u, v) = 0, \forall u \in V\}$$

Remarque. Dans le cas des groupes de Coxeter et de leur représentation géométrique, on a $V^\perp = \bigcap_{s \in S} H_{\alpha_s}$.
Il est clair que $V^\perp \subset \bigcap_{s \in S} H_{\alpha_s}$, pour l'autre inclusion, il suffit de se rappeler que $(\alpha_s)_{s \in S}$ est une base de V, et donc de remarquer que $B(v, \alpha_s) = 0 \ \forall s \in S$ force $B(v, u) = 0$ pour tout $u \in V$ pour cette raison.

Afin de montrer que si W est fini alors B est définie positive, nous avons besoin de quelques lemmes sur les représentations de groupe en général, le radical de la forme B_W et sur les sous-représentations de la représentation géométrique d'un groupe de Coxeter :

Lemme 3.6. *Soit (W, S) un système de Coxeter que l'on suppose irréductible :*

1. *Pour tout sous-espace strict non-nul V' de V stable par W, on a $V' \subset V^\perp$.*
2. *Si B est dégénérée, alors V n'est pas un W-module complètement réductible (n'est pas décomposable en somme directe d'une famille de représentations irréductibles).*
3. *Si B est non-dégénérée, alors V est un W-module complètement réductible.*
4. *Les seuls endomorphismes de V qui commutent avec l'action de W sont les actions des scalaires sur V.*

Preuve. On rappelle que (W, S) est supposé irréductible.

1. Soit $V' \subset V$ non-nul un W-sous module de V.
   — Supposons d'abord qu'aucune des racines n'appartient à V' :
   Pour chaque σ_s, on peut décomposer V' en deux sous-espaces stricts supplémentaires, qui sont les sous-espaces propres E_1 et E_{-1} associés aux valeurs propres 1 et -1 de σ_s. Le sous-espace propre associé à -1 est l'espace nul, car $E_{-1} = \mathbb{R}\alpha_s$ n'a pas d'autre intersection que le vecteur nul avec V', sinon on aurait $\alpha_s \in V'$. Donc pour toutes les réflexions σ_s la restriction de σ_s à V' est l'identité sur V', i.e $V' \subset \bigcup_{s\in S} H_{\alpha_s} = V^\perp$.
   — Ensuite, si $\alpha_s \in V'$:
   Considérons s' un voisin de s sur le graphe de Coxeter de W (qui existe toujours sauf dans le cas du graphe à un seul sommet puisque (W, S) est supposé irréductible), on a :
   $$\sigma_{s'}(\alpha_s) = \alpha_s - 2B(\alpha_s, \alpha_{s'})\alpha_{s'}$$
   Or $B(\alpha_s, \alpha_{s'}) < 0$ comme $m(s, s') > 2$. Comme V' est stable par W, on a que $\sigma_{s'}(\alpha_s) = \alpha_s + c\alpha_{s'}$ $(c > 0)$ est dans V', i.e $\alpha_{s'} \in V'$. Comme (W, S) est irréductible, son graphe de Coxeter est connexe et on peut appliquer successivement ce raisonnement à tous les $s \in S$, et donc obtenir que $V' = V$ (car $(\alpha_s)_{s\in S}$ est une base de V), ce qui contredit l'hypothèse de base.
2. Supposons que B est dégénérée : on a donc que $V^\perp$ est un W-sous module de V, par (1), comme tout W-sous module de V est inclus dans $V^\perp$ alors $V^\perp$ n'admet aucun supplémentaire dans V et V n'est donc pas complètement réductible.
3. Supposons que B est non-dégénérée : on a donc $V^\perp = \{0\}$ et par (1), V n'a pas de W-sous module strict non nul : V est irréductible.
4. Soit $z \in End(V)$ tel que $zw = wz\ \forall w \in W$ (en confondant w avec son action). Considérons $s \in S$: comme $z\sigma_s = \sigma_s z$, on a $\mathbb{R}\alpha_s$ stable par z, z agit avec une valeur propre c sur $\mathbb{R}\alpha_s$. Montrons que $z = c \cdot Id$:
   Considérons $\ker(z - c \cdot Id) := V'$
   — On a $\mathbb{R}\alpha_s \subset V'$, donc V' est non nul.
   — On a V' stable par w pour tout $w \in W$:
   $$\begin{aligned} x \in V' &\iff (z - c \cdot Id)(x) = 0 \\ &\iff z(x) = c \cdot x \\ &\iff w(z(x)) = w(c(x)) \\ &\iff z(w(x)) = c \cdot w(x) \\ &\iff w(x) \in V' \end{aligned}$$
   Or comme $\mathbb{R}\alpha_s \not\subset V^\perp$, par (1) comme V' est un W-sous module de V, on obtient que $V' = V$.
   On a donc $\forall u \in V, z(u) = c \cdot u$. Donc z est l'action par un scalaire.

Lemme 3.7. *Soit G un groupe et $\rho : G \to GL(E)$ une représentation de G dans E un espace vectoriel de dimension finie sur un corps de caractéristique 0.*

1. *Si G est fini, il existe alors un produit scalaire G-invariant sur E.*
2. *Si G est finie, alors ρ est complètement réductible.*
3. *Si les seuls endomorphismes de E commutant avec l'action de G sont les scalaires et que B, B' sont deux formes bilinéaires symétriques non-dégénérées sur E, stables par l'action de G, alors B est un multiple scalaire de B'.*

Preuve. 1. Considérons une forme bilinéaire symétrique définie positive quelconque B sur E, et notons $\forall u, v \in E$ $\overline{B}$ la forme bilinéaire suivante :

$$\overline{B}(u, v) := \frac{1}{|G|} \sum_{g \in G} B(g \cdot u, g \cdot v)$$

Cette forme bilinéaire est bien symétrique et définie positive. Vérifions qu'elle est G-invariante, soit $h \in G$:

$$\overline{B}(h \cdot u, h \cdot v) = \frac{1}{|G|} \sum_{g \in G} B((g \cdot h) \cdot u, (g \cdot h) \cdot u)$$

Posons $g' = gh$, tout élément de G pouvant s'écrire sous cette forme, on a :

$$\begin{aligned} \overline{B}(h \cdot u, h \cdot v) &= \frac{1}{|G|} \sum_{g \in G} B((g \cdot h) \cdot u, (g \cdot h) \cdot u) \\ &= \frac{1}{|G|} \sum_{g' \in G} B(g' \cdot u, (g' \cdot u) \\ &= \overline{B}(u, v) \end{aligned}$$

$\overline{B}$ est donc bien G-invariante.

2. Notons que comme $\overline{B}$ est non dégénérée, on a $E = V \oplus V^{\perp}$ pour tout sous-espace V. De plus, comme $\overline{B}$ est G-invariante, alors le supplémentaire orthogonal de tout G-sous module de E est lui-même un G-sous module de E car stable par l'action de G. En répétant cette étape successivement jusqu'à obtenir des G-modules irréductibles (ce qui arrive forcément car la dimension baisse à chaque étape), on en déduit la complète réductibilité (semisimplicité) de ρ.
3. Toute forme bilinéaire non dégénérée β sur un espace vectoriel de dimension finie E induit l'isomorphisme suivant entre E et E^* :

$$\begin{aligned} \varphi_v : & E \to E^* \\ & v \mapsto \beta(u, v) \end{aligned}$$

Cette application est bien injective, puisque $\beta(u, v) = 0$ $\forall u \in V$ signifie que $v \in V^{\perp}$, i.e que $v = 0$ car β est non-dégénérée. Et lorsque β est G-invariante, elle est donc un isomorphisme de G-modules. Considérons deux formes B et B' sur E respectant ces conditions, et considérons également $B'^{-1}B$ qui est donc un automorphisme de G-module de E, c'est-à-dire un endomorphisme de E commutant avec $\rho(g)$ pour tout $g \in G$.

Par hypothèse, $B'^{-1}B$ est simplement l'action par un scalaire c (que l'on note également c comme application).

$$\begin{aligned} B'^{-1}B(u) &= c \cdot u \\ B(u) &= B'(c \cdot u) \\ B(u) &= c \cdot B'(u) \end{aligned}$$

On a donc bien le résultat demandé.

Théorème 3.8. *Si W est un groupe de Coxeter fini, alors B_W est un produit scalaire.*

Preuve. Supposons sans perte de généralité que (W, S) est irréductible. (2) du lemme 3.7 nous dit que ρ est semisimple, et (2) du lemme 3.6 nous indique donc que B_W doit être nondégénérée. Comme B_W est nondégénérée, on a d'après le (3) du lemme 3.6 que ρ est irréductible, et (4) nous dit que les seuls endomorphismes de V commutant avec l'action de W sont les actions des scalaires, ce qui signifie par (3) du lemme 3.7 que B_W est la seule forme bilinéaire G-invariante nondégénérée à multiple par un scalaire près.
Or, par (1) du lemme 3.7, il existe un produit scalaire B' G-invariant. Donc $B' = cB_W$ pour un certain $c \in \mathbb{R}^*$. Or, comme $B_W(\alpha_s, \alpha_s) = 1$ on doit avoir $c > 0$ i.e B_W définie positive.
Donc B_W est une forme bilinéaire symétrique définie positive, c'est donc un produit scalaire sur V.

Corollaire 3.9. *Les groupes de Coxeter finis sont exactement les groupes de réflexions finis.*

Preuve. On a vu avec le corollaire 3.2 que tout groupe de réflexion fini est un groupe de Coxeter, puis on a vu avec le corollaire 3.5 que la représentation géométrique d'un groupe de Coxeter est fidèle (i.e que tout groupe de Coxeter est isomorphe à son image par la représentation géométrique). Par le théorème précédent, si W est fini, on peut en déduire que cette représentation se fait dans un espace euclidien, donc que les réflexions qui engendrent l'image par la représentation sont précisément des réflexions telles que définies pour un espace euclidien. Donc $\rho(W)$ est un groupe de réflexions fini.

3.2 Théorème de Matsumoto et retour au problème du mot

Maintenant que nous avons montré que les groupes de Coxeter finis sont exactement les groupes de réflexions fini, on peut en déduire qu'il est possible d'appliquer notre critère de réduction aux groupes de Coxeter finis. Par chance, Sage permet bien de manipuler les groupes de Coxeter avec leur représentation géométrique, donc rien n'est à changer dans nos fonctions.
Nous avons donc réussi la première étape de notre résolution, nous devons maintenant déterminer un moyen de voir que deux mots réduits sont équivalents, fort heureusement pour nous, le théorème suivant nous donne précisément un critère facile à appliquer :

Théorème de Matsumoto 3.10. *Soit M un monoïde, et $f : S \to M$. Pour $s, s' \in S$, posons :*

$$a(s, s') := \begin{cases} (f(s)f(s'))^l & si\ m(s, s') = 2l \\ (f(s)f(s'))^l f(s) & si\ m(s, s') = 2l + 1 \\ 1 & si\ m(s, s') = \infty \end{cases}$$

Si $a(s, s') = a(s', s)$ pour tous $s, s' \in S$, alors il existe $g : W \to M$ telle que

$$g(w) = f(s_1) \cdots f(s_r)$$

pour tout $w \in W$ et pour toute décomposition réduite $s_1 \cdots s_r$ de w.

Ce théorème nous donne de manière plus informelle que toutes les décompositions réduites d'un élément de W sont reliées par les mouvements de tresse (si l'on note $\alpha_{s,s'} = ss' \cdot \cdot$ le mot alternant s et s' de taille $m(s,s')$, un mouvement de tresse est l'action de remplacer $\alpha_{s,s'}$ par $\alpha_{s',s}$ dans un mot). Certains auteurs tels que [1] préfèrent appeler ce théorème la *propriété du mot* et l'énoncer directement avec les mouvements de tresse, mais nous suivrons ici en grande partie la preuve de [2].
Afin de montrer ce théorème, nous avons besoin de quelques lemmes :

Lemme 3.11. *Soit $w \in W$, pour toutes expressions réduites $s_1 \cdots s_r$ et $t_1 \cdots t_r$ de w, il existe $1 \leq j \leq r$ tel que $s_1 w = s_1 t_1 \cdots \widehat{t_j} \cdots t_r$.*

Preuve. Comme il existe une expression réduite de w de première lettre s_1, on a $l(s_1 w) < l(w)$, c'est-à-dire :

$$\begin{aligned} & s_1 \cdots s_r = t_1 \cdots t_r \\ \iff & s_2 \cdots s_r = s_1 t_1 \cdots t_r \end{aligned}$$

Comme l'expression à droite de l'égalité n'est pas réduite, on peut lui appliquer la condition de délétion : il existe donc deux lettres distinctes de ce mot que l'on peut supprimer.
Raisonnons par l'absurde pour montrer que la première lettre que l'on peut supprimer est toujours s_1. Si ce n'était pas le cas, on aurait :

$$\begin{aligned} & s_1 t_1 \cdots t_r = s_1 t_1 \cdots \widehat{t_i} \cdots \widehat{t_j} \cdots t_r \\ \iff & t_1 \cdots t_r = t_1 \cdots \widehat{t_i} \cdots \widehat{t_j} \cdots t_r \end{aligned}$$

L'expression à droite serait donc une expression de moins de r lettres pour w, ce qui contredit la minimalité de nos deux expressions.
On a donc l'égalité suivante :

$$sw = t_1 \cdots \widehat{t_j} \cdots t_r$$

qui est précisément celle que l'on cherchait.

Remarque. Ce lemme est en réalité une version faible de la condition d'échange qui s'énonce ainsi : si $s \in S$ et $l(sw) < l(w)$, soit $s_1 \cdots s_r$ une expression de w (pas forcément réduite), alors il existe $1 \leq j \leq r$ tel que $sw = ss_1 \cdots \widehat{s_j} \cdots s_r$.
Ce lemme se montre très bien en particulier dans le cas des expressions réduites à l'aide du théorème de Matsumoto : on peut alors déduire qu'il existe forcément une expression réduite de w commençant par s et appliquer notre lemme.

Lemme 3.12. *Soient $w \in W$ tel que $l(w) = q$, D l'ensemble des décompositions réduites de w, E un ensemble quelconque et $F : D \to E$.*
Supposons que si $s = s_1 \cdots s_q, s' = s'_1 \cdots s'_q \in D$ satisfont l'une des conditions suivantes, alors $F(s) = F(s')$:

1. *$s'_1 = s_1$ ou $s'_q = s_q$*
2. *Il existe $s, s' \in S$ tels que $s_j = s'_k = s$ et $s_k = s'_j = s'$ pour tous j impairs et k pairs.*

Alors F est constante.

Preuve. Montrons ce lemme en deux étapes :

— Soient $s, s' \in S$ et $t = s'_1 s_1 \cdots s_{q-1}$. Commençons par montrer que si $F(s) \neq F(s')$, on a $t \in D$ et $F(t) \neq F(s)$:
On a $w = s'_1 \cdots s'_q$, donc $s'_1 w = s'_2 \cdots s'_q$. Comme $l(s'_1 w) < l(w) = q$, par le lemme 3.11 il existe $1 \leq j \leq q$ tel que $u = s'_1 s_1 \cdots \widehat{s_j} \cdots s_q \in D$. Par 1, on a donc $F(u) = F(s')$.
On a $j = q$: si $j \neq q$, on a $F(u) = F(s)$ toujours d'après 1, ce qui est absurde.
Donc $j = q$, c'est-à-dire $t = u \in D$ et $F(t) = F(s') \neq F(s)$.

— Soient $s, s' \in D$, définissons la suite $(s_j)_{0\leq j\leq q}$ de la manière suivante :

$$\begin{cases} s_0 = s' \\ s_1 = s \\ s_{q+1-k} = s_1 s_1' \cdots s_1 s_1' s_1 s_2 \cdots s_k \text{ si } q-k \text{ pair et } 1 \leq k \leq q \\ s_{q+1-k} = s_1' s_1 \cdots s_1 s_1' s_1 s_2 \cdots s_k \text{ si } q-k \text{ impair et } 1 \leq k \leq q \end{cases}$$

Notons (H_j) l'assertion "$s_j \in D, s_{j+1} \in D, F(s_j) \neq F(s_{j+1})$". Selon la parité de q, on a $s_q = s_1 s_1' \cdots s_1 s_1' s_1$ et $s_{q+1} = s_1' s_1 \cdots s_1' s_1 s_1'$ ou $s_q = s_1' s_1 \cdots s_1' s_1$ et $s_{q+1} = s_1 s_1' \cdots s_1 s_1'$ (en se permettant de définir s_{q+1} de manière naturelle), or par 2., $F(s_q) \neq F(s_{q+1})$, donc (H_q) est fausse. Comme la première étape de la preuve montre que pour tout j, $(H_j) \implies (H_{j+1})$, on en déduit que (H_0) est fausse, donc que $F(s) = F(s')$ pour tous $s, s' \in D$,donc que F est constante.

Nous allons maintenant pouvoir montrer le théorème de Matsumoto :

Preuve du théorème de Matsumoto. Pour tout $w \in W$, notons D_w l'ensemble des décompositions réduites de w, et $F_w : D_w \to M$ telle que :

$$F_w(s_1 \cdots s_q) = f(s_1) \cdots f(s_q)$$

Montrons par récurrence sur $l(w)$ que F_w est constante.

— C'est évident pout le cas $l(w) = 0$, tout comme pour le cas $l(w) = 1$.

— Posons $l(w) = q > 1$ et supposons le théorème vrai pour les élément de W de longueur $< q$, utilisons le lemme 3.12 pour montrer que F_w est constante :

1. Posons $w' = s_1 \cdots s_{q-1}$ et $w'' = s_2 \cdots s_q$, on a :

$$F_w(s_1 \cdots s_q) = F_{w'}(s_1 \cdots s_{q-1}) f(s_q) = f(s_1) F_{w''}(s_2 \cdots s_q)$$

par l'hypothèse de récurrence, $F_{w'}$ et $F_{w''}$ sont constantes, donc pour tous $s, s' \in D_w$, si $s_1 = s_1'$, on a $F_w(s_1 \cdots s_q) = f(s_1) F_{w''}(s_2 \cdots s_q) = f(s_1') F_{w''}(s_2' \cdots s_q') = F_w(s_1' \cdots s_q')$ et si $s_q = s_q'$, on a $F_w(s_1 \cdots s_q) = F_{w'}(s_1 \cdots s_{q-1}) f(s_q) = F_{w'}(s_1' \cdots s_{q-1}') f(s_q') = F_w(s_1' \cdots s_q')$.

2. Supposons qu'il existe $s, s' \in S$ tels que $s_j = s_k' = s$ et $s_k = s_j' = s'$ pour tous j impairs et k pairs, si $s = s'$, on retourne au cas $l(w) = 0$ ou $l(w) = 1$, donc considérons le cas $s \neq s'$:
Soient σ, σ' deux décompositions réduites distinctes de w dans le sous-groupe diédral engendré par $\{s, s'\}$, si $m(s, s') = \infty$ alors il n'y a qu'une seule décomposition réduite possible pour w, donc considérons le cas $m := m(s, s') < \infty$. La première lettre de l'un doit être différente de celle de l'autre. Sans perte de généralité supposons que la première lettre de σ soit s et celle de σ' soit s' : On a alors $F_w(\sigma) = a(s, s')$ et $F_w(\sigma') = a(s', s)$, or par hypothèse $a(s, s') = a(s', s)$. Donc $F_w(\sigma) = F_w(\sigma')$.

Donc par le lemme 3.12, F_w est constante.

Avec le théorème de Matsumoto, on obtient une solution pour trouver toutes les décompositions réduites équivalentes : il suffit d'en trouver une, ce que l'on sait faire (en particulier dans le cas fini), et d'appliquer successivement des mouvements de tresse jusqu'à ce que l'on ne trouve plus de nouveau mot.
Avant d'obtenir une fonction Sage pour déterminer tous les mots réduits équivalents, nous devons construire quelques fonctions auxiliaires :

```
1 def indices_tresses(rel,sigma):
```

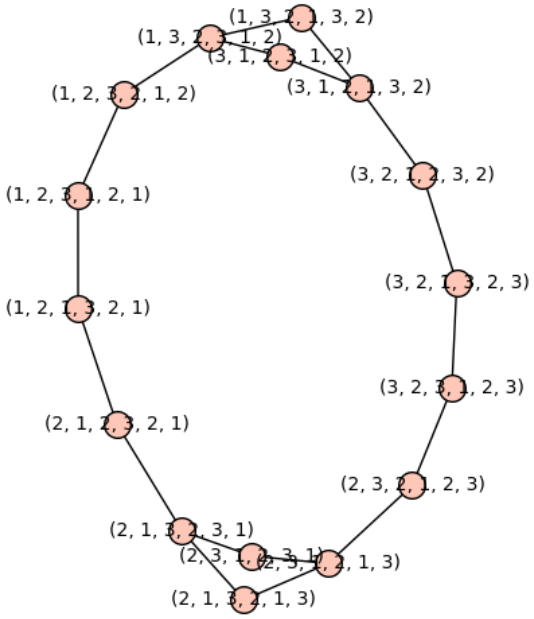
```
    res = []
    k = len(rel[0])
    for i in range(len(sigma)):
        if sigma[i:(i+k)] == rel[0] or sigma[i:(i+k)] == rel[1]:
            res += [i]
    return res

def appliquer_tresse(sigma,rel,i):
    if sigma[i] == rel[0][0]:
        res = sigma[:i] + rel[1] + sigma[i+len(rel[0]):]
    else :
        res = sigma[:i] + rel[0] + sigma[i+len(rel[0]):]
    return res
```

La fonction `indices_tresses` prend en argument notre mot `sigma` et une relation de tresse `rel` (représentée par une liste à deux éléments) et permet de rechercher les morceaux de notre mot (identifiés par la position de la première lettre) auquel on peut appliquer des mouvements de tresse, et la fonction `appliquer_tresse` prend en argument `sigma` et `rel` comme précédemment, en plus de l'indice `i` et permet d'appliquer `rel` à la position `i`.
À l'aide de ces fonctions, nous pouvons maintenant définir la fonction suivante :

```
def orbite_tresse(sigma, W):
    w = gen_to_indices(reduction(sigma,W),W)
    Tmp = {tuple(w)}
    Traite = set()
    rels = W.braid_relations()
    for i in range(len(rels)):
        rels[i][0] = tuple(rels[i][0])
        rels[i][1] = tuple(rels[i][1])
    while len(Tmp) != 0 :
        w = Tmp.pop()
        for rel in rels :
            i = indices_tresses(rel,w)
            for j in i :
                l = appliquer_tresse(w,rel,j)
                if not(l in Tmp) and not(l in Traite):
                        Tmp.add(l)
        Traite.add(w)
    return Traite
```

La fonction `orbite_tresse` ainsi définie prend en argument une liste `sigma` d'éléments de S représentant un mot, ainsi que le groupe de Coxeter W dans lequel on travaille, et renvoie un objet `Set` contenant la liste des mots réduits équivalents à `sigma` en réduisant le mot puis en appliquant successivement les relations de tresses sur les différents mots obtenus, jusqu'à ne plus obtenir de nouveau mot (ce qui doit arriver au bout d'un moment par le théorème de Matsumoto).
Le lecteur de ce document doit savoir que le but premier de ce projet était de construire le graphe ayant pour sommets l'ensemble des mots réduits équivalents à un mot, dans lequel deux sommets sont reliés par une arête si et seulement si on peut passer de l'un à l'autre par un unique mouvement de tresse : que l'on appelle sans grande inventivité le *graphe des mots réduits* d'un mot.

Exemple 3.5. Le graphe des mots réduits de $w := s_3s_2s_1s_3s_2s_3$ de A_3 est le graphe suivant :



Graphe des mots réduits de w

Remarque. Le théorème de Matsumoto et la propriété du mot peuvent également s'interpréter en terme des propriétés de ce graphe : ils sont équivalents à la connexité de ce graphe.

Quelques simples modifications de la fonction précédentes nous permettent de déterminer ce graphe :

```
def graphe_mots_reduits(sigma, W):
    w = gen_to_indices(reduction(sigma,W),W)
    G = graphs.EmptyGraph()
    Tmp = {tuple(w)}
    Traite = set()
    rels = W.braid_relations()
    for i in range(len(rels)):
        rels[i][0] = tuple(rels[i][0])
        rels[i][1] = tuple(rels[i][1])
    while len(Tmp) != 0 :
        w = Tmp.pop()
        for rel in rels :
            i = indices_tresses(rel,w)
            for j in i :
                l = appliquer_tresse(w,rel,j)
                G.add_edge(w,l,rel)
                if not(l in Tmp) and not(l in Traite):
                        Tmp.add(l)
        Traite.add(w)
    return G
```

Cette fonction `graphe_mots_reduits` prend exactement les mêmes arguments que `orbite_tresse` mais renvoie à la place le graphe qu'elle construit au fur et à mesure en reliant les différents mots par les relations de tresse qui permettent de passer de l'un à l'autre : à chaque fois que l'on utilise une relation de tresse, on rajoute le sommet correspondant au mot s'il n'était pas encore dans le graphe et une arête pour relier les deux mots par la relation de tresse permettant de passer de l'un à l'autre.

Remarque. Les deux fonctions `orbite_tresse` et `graphe_mots_reduits` utilisent le type python `Set`, qui est un type très efficace pour la recherche d'éléments possèdant le fonctionnement d'une *table de hachage*. Si le fonctionnement des tables de hachage vous intéresse, il est développé immédiatement après, dans l'annexe de ce mémoire.

Nous avons donc résolu le problème du mot dans les groupes de Coxeter. En particulier, nous avons un programme parfaitement fonctionnel pour le cas des groupes de Coxeter finis.

A Tables de hachage

Dans cet annexe, nous parlerons du problème de la représentation des ensembles en informatique avec les différentes structures de données à notre disposition, en particulier de la recherche d'éléments dans ces structures, et introduirons les concepts de base des tables de hachage.

Définition A.1. Soit E un ensemble fini, et $f : E \to \mathbb{N}$ d'image finie majorée par m. La table de hachage associée à ce couple est le tableau (dans le sens de l'informatique) T de taille m tel que $x \in E$ est rangé dans la i^e case de T si et seulement si $f(x) = i$.
On dit que E est *haché* et on appelle f la *fonction de hachage*.

Ces objets en apparence assez simples sont une vraie mine d'or lorsque bien codés pour améliorer la complexité moyenne de problèmes nécessitant de faire des recherches d'appartenance dans les gros ensembles. Mais comme dit précédemment, il faut avant tout qu'ils soient implémentés correctement.
Au moment de trouver une implémentation concrète de ces objets, nous avons besoin de remarquer certaines choses :

1. Une bonne fonction de hachage doit être calculable facilement : si la fonction de hachage est plus difficile à calculer que la recherche d'un élément dans une liste ou un tableau, ce n'est pas efficace.
2. Chaque case d'un tableau en informatique ne référence qu'un seul objet, et il est dans de nombreux cas trop demander d'exiger une fonction de hachage injective. Dans ce cas, comment gérer les collisions ? (quand deux éléments de E se retrouvent dans la même case)
3. Quelles structures de données sous-jacentes peut-on utiliser pour une implémentation efficace ?

Pour la complexité des fonctions de hachage, cela dépend essentiellement des objets utilisés. En Python, certaines classes d'objets sont dites *hachables* si elles possèdent une méthode `__hash__`.
Le type python `Set` est implémenté comme une table de hachage qui utilise comme fonction de hachage les méthodes `__hash__` des objets hachables.
En terme de structure de données, le plus courant est de prendre un tableau classique (comme en Java ou en C) qui a l'avantage de permettre un accès en $O(1)$ aux données référencées.
Nous allons donc nous concentrer en particulier sur la gestion des collisions :

— Une première méthode consiste à stocker une liste dans chaque case de notre tableau, et à rajouter chaque élément au début ou à la fin de la liste (ce qui est toujours possible en $O(1)$ selon l'implémentation choisie) au moment du remplissage. Cette solution est peu coûteuse lorsque les listes sont peu remplies, mais si la fonction nous donne une répartition très inégale, avec beaucoup de valeurs dans la même liste, on perd beaucoup en efficacité. Cette solution est donc efficace lorsque les classes sont assez petites.
— Une autre méthode efficace est la résolution des collisions par sondage : on définit un taux de remplissage maximal de chaque case du tableau, et au moment du remplissage si la case est pleine, on recherche une autre case non remplie. Il existe de nombreuses manières de chercher une case non remplie, notamment les sondages linéaires (si $f(x) = i$ on regarde si la case $i + n$ est vide, et on continue successivement), le sondage quadratique (la même chose mais on avance de manière quadratique), ou le sondage par double hachage (qui utilise une 2^{nde} fonction de hachage pour déterminer la prochaine case à visiter) qui ont pour avantage de permettre un remplissage homogène de la table. Le type python `Set` que nous avons utilisé utilise une méthode de sondage linéaire pour remplir la table de hachage.

Les tables de hachage en général ne sont pas les meilleures structures lorsqu'on cherche une très bonne complexité dans le pire cas (on est sur une complexité en $O(n)$ dans le pire cas pour les trois opérations classiques avec l'implémentation du type `Set`), mais une bonne implémentation nous donne un cas moyen en $O(1)$ pour ces trois opérations, rendant la gestion de grosses données beaucoup plus agréable qu'avec des listes ou des arbres.

B Plus long élément de A_n et bijection d'Edelman-Greene

Considérons le système de Coxeter A_n, dont on rappelle la présentation :

$$\langle (S_{n+1}, (\sigma_i)_{1\leq i\leq n}) | \sigma_i\sigma_i = 1, (\sigma_i\sigma_{i+1})^3 = 1, (\sigma_i\sigma_j)^2 = 1 \text{ si } |j-i| > 1, 1 \leq i < j \leq n \rangle$$

avec $\sigma_i = (i, i+1)$.
Tout groupe de Coxeter fini admet un élément dit le *plus long*, qui est défini assez naturellement de la manière suivante :

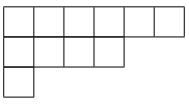
Définition B.1. Soit W un groupe de Coxeter fini, on dit que w est l'*élément le plus long* de W si $\forall w' \in W$, on a $l(w') \leq l(w)$.

Remarque. Si un groupe de Coxeter admet un élément le plus long (c'est-à-dire s'il est fini), alors il est unique. Ce résultat est conséquence du théorème de simple transitivité que l'on peut trouver dans [5], qui est lui-même une conséquence facile à voir du lemme 2.8.

Dans le cas du groupe de Coxeter A_n, l'élément le plus long est tout simplement la permutation qui à la i^e racine simple associe l'opposé de la $(n-i+1)^e$ racine simple.
En faisant différents tests sur le nombre de décompositions réduites de certains éléments de groupes de Coxeter, nous pouvons remarquer assez facilement que le nombre de décompositions réduites de l'élément le plus long de A_n est une suite familière à de nombreux mathématiciens.

Définition B.2. On appelle *diagramme de Ferrers* une collection finie de case organisée en lignes, telle que le nombre de cases pour chaque ligne est décroissant.

Exemple B.3. Chaque diagramme de Ferrers est défini par une partition d'un entier, on a par exemple ci-dessous le diagramme $(6, 4, 1)$:



Définition B.4. On appelle *tableau de Young* un diagramme de Ferrers dont on remplit les cellules avec des nombres entiers.
Un tel tableau est dit standard s'il possède n cellules tel qu'à chaque entier entre 1 et n on associe exactement une cellule, et si les valeurs dans les lignes et colonnes sont strictement croissantes.

Exemple B.5. Reprenons notre diagramme précédent, on peut par exemple considérer le tableau de Young suivant :

7	4	8	5	8	4
5	8	3	5		
1					

Ce tableau n'est pas standard, mais on peut considérer le tableau suivant, fait à partir du diagramme $(4, 3, 1)$ qui est standard.

1	3	5	6
2	7	8	
4			

Avec nos tests, nous pouvons voir que le nombre de décompositions réduites de l'élément le plus long de A_n est le nombre de tableaux de Young standards faits à partir du diagramme $(n, n-1, \cdots, 2, 1)$.
Dans [3], Paul Edelman et Curtis Greene ont exhibé une bijection entre ces deux ensembles, et je vous propose de vous la présenter dans les deux sens, puis de vous en donner une implémentation en Sage.

Soient T_n l'ensemble des tableaux de Young standards de forme $(n, n-1, \cdots, 2, 1)$ et C_n l'ensemble des décompositions réduites du plus long élément de A_n. Commençons par le sens $T_n \to C_n$:

Définition B.6. Soit T un tableau de Young standard et n un entier de T, on définit le *chemin d'évacuation* de n comme le chemin obtenu avec la procédure suivante :

1. Regarder les voisins immédiats à gauche et en haut de n
2. Tant qu'une des deux cases existe, échanger la position de n et du plus grand des deux voisins si on doit choisir
3. Lorsque n n'a plus de voisin à gauche ni en haut, on vide la case

Cette procédure peut se réaliser en Sage (avec l'objet `YoungTableau`) de la manière suivante :

```
def evacuation_path(n,T):
    coord = [-1,-1]
    for i in range(len(T)):
        if T[i][-1] == n:
            coord = [i,len(T[i])-1]
    c = coord.copy()
    res = [c]
    while c[0] != 0 and c[1] != 0:
        if T[c[0]][c[1]-1] > T[c[0]-1][c[1]] :
            c = [c[0],c[1]-1]
        else :
            c = [c[0]-1,c[1]]
        res.append(c)
    if c[0] == 0 :
```

```
15          while T[c[0]][c[1]] != 0 and c[1] != 0:
16              c = [c[0],c[1]-1]
17              res.append(c)
18      else :
19          while T[c[0]][c[1]] != 0 and c[0] != 0:
20              c = [c[0]-1,c[1]]
21              res.append(c)
22      return res
```

Cette fonction prend en argument un tableau de Young (un objet `YoungTableau` de Sage), et un élément `n` du tableau situé à la fin d'une des lignes, et calcule le chemin d'évacuation de `n`, qu'elle renvoie sous forme d'une liste.
Ces chemins d'évacuation permettent de définir une bijection de la manière suivante :

Définition B.7. Le premier sens de la *bijection d'Edelman-Greene*, est la fonction $f : T_n \to C_n$ définie par la procédure suivante pour tout tableau de Young standard T de forme $(n, n-1, \cdots, 2, 1)$:

1. Notons r, qui est pour l'instant le mot vide (l'identité dans A_n)
2. Tant que T n'est pas vide, on considère la valeur maximum m de T
3. On calcule le chemin d'évacuation de m, et on évacue ce dernier en suivant ce chemin
4. Si m se trouvait dans la i^e colonne, on ajoute σ_i au début du mot r
5. Lorsque T est vide, on retourne donc le mot r

Exemple B.8. Appliquons cette procédure au tableau suivant :

1	3	6	7
2	4	10	
5	9		
8			

Ici notre maximum est 10, le chemin d'évacuation est donc $(6, 3, 1)$, on obtient donc :

B	1	3	7
2	4	6	
5	9		
8			

et on ajoute 3 à gauche de notre mot.
En appliquant successivement cette procédure à 9,8,etc, on obtient :

B	1	3	7
B	4	6	
2	5		
8			

$\Longrightarrow$

B	1	3	7
B	4	6	
B	5		
2			

$\Longrightarrow$

B	B	1	3
B	4	6	
B	5		
2			

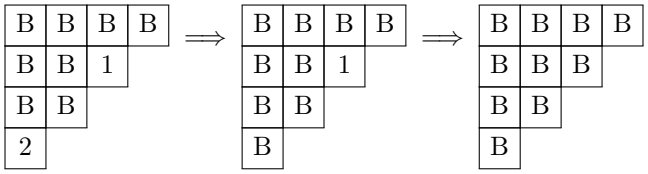
B	B	1	3
B	B	4	
B	5		
2			

$\Longrightarrow$

B	B	1	3
B	B	4	
B	B		
2			

$\Longrightarrow$

B	B	B	3
B	B	1	
B	B		
2			



Et on obtient donc le mot $(3, 1, 4, 3, 2, 3, 4, 1, 2, 3)$.

Comme pour chaque résultat qui s'y prête dans ce mémoire, donnons une fonction Sage qui permettrait de calculer cette fonction :

```
def max(t):
    max = 0
    for i in t:
        for j in i :
            if j > max :
                max = j
    return max

def bijection_tableaux_mots(t) :
    L = list()
    for i in t :
        L.append(list(i))
    maxi = max(L)
    res = []
    while maxi != 0 :
        path = evacuation_path(maxi,L)
        L[path[0][0]][path[0][1]] = 0
        for i in range(len(path)-1):
            L[path[i][0]][path[i][1]], L[path[i-1][0]][path[i-1][1]] = L[path[i-1
        maxi = max(L)
        res.append(path[0][1]+1)
    res.reverse()
    return tuple(res)
```

La fonction auxiliaire `max` permet de déterminer l'élément le plus grand du tableau `t` qu'elle prend en argument, et la fonction `bijection_tableau_mots` calcule l'image d'un tableau `t` de T_n par la bijection d'Edelman-Greene.
Allons maintenant explorer l'autre sens :

Définition B.9. On appelle *insertion de Coxeter-Knuth* la procédure suivante :
Soient $P_1, \cdots P_l$ les lignes de T un tableau de Young (pas forcément standard), et x la valeur que l'on cherche à insérer dans T :

- On commence avec $i = 1$
- Si $x \geq x_i$ pour tout $x_i \in P_i$, alors on insère x à la fin de la ligne P_i
- Sinon, considérons y le plus petit entier de P_i tel que $y > x$:
  - Si $y = x+1$ et que x est déjà présent dans P_i, on tente par la même procédure d'insérer $x+1$ dans la ligne P_{i+1} sans changer la ligne P_i
  - Sinon, on remplace y par x dans P_i et on tente par la même procédure d'insérer y dans la ligne P_{i+1}

Cette procédure est décrite dans la fonction Sage `coxeter_knuth` suivante, qui utilise la fonction `max_ligne` :

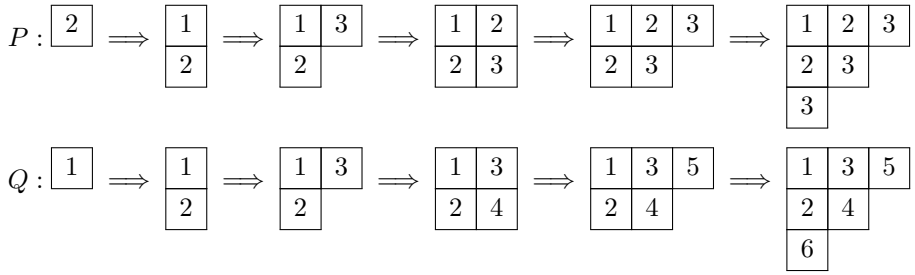
```
def max_ligne(l):
    max = l[0]
    for i in l :
        if i > max :
            max = i
    return max

def coxeter_knuth(n,T,S,c):
    if T == [] :
        T.append([n])
        S.append([c])
    else :
        i = 0
        while n != 0 :
            if i == len(T):
                T.append([n])
                S.append([c])
                n = 0
            elif n >= max_ligne(T[i]) :
                T[i].append(n)
                S[i].append(c)
                n = 0
            else :
                if n in T[i] and n+1 in T[i] :
                    n = n+1
                    i += 1
                else :
                    j = 0
                    while T[i][j] <= n :
                        j += 1
                    n, T[i][j] = T[i][j], n
                    i += 1
```

La fonction `max_ligne` prend en argument une ligne `l` d'un tableau de Young et renvoie l'élément le plus grand de cette ligne, et la fonction `coxeter_knuth` prend en argument deux tableaux de Young `T` et `S` et applique l'insertion de Coxeter-Knuth à `n` dans `T`, et insère `c` dans `S` de manière à avoir la même forme que `T` lorsqu'on applique l'insertion à `T` et `S` de même forme.

Définition B.10. L'autre sens de la *bijection d'Edelman-Greene*, est la fonction $f : C_n \to T_n$ définie par la procédure suivante pour tout mot réduit w du plus long élément de A_n :

1. Considérons deux tableaux P et Q vides, et $i = 1$
2. Pour chaque lettre s de w de gauche à droite, on applique l'insertion de Coxeter-Knuth à s et P, on ajoute la même case du diagramme de Ferrers sous-jacent à P à Q, et on remplit cette case par i, puis on incrémente i de 1
3. On retourne Q, qui est un tableau de Young standard de la bonne forme

Exemple B.11. Réalisons cette procédure pour le mot réduit $(2, 1, 3, 2, 3, 1)$ de l'élément le plus long de A_3 :



On obtient donc le tableau suivant :

1	3	5
2	4	
6		

Ce sens de la bijection peut se programmer de la manière suivante dans Sage :

```
def bijection_mots_tableaux(l):
    T = []
    S = []
    for i in range(len(l)):
        coxeter_knuth(l[i],T,S,i+1)
    return StandardTableau(S)
```

Cette fonction, qui établit l'autre sens de la bijection, prend en argument un mot réduit de C_n sous la forme d'une liste d'entiers `l` et applique successivement l'insertion de Coxeter-Knuth pour obtenir le tableau `T` composé des lettres du mot, et le tableau `S` obtenu en gardant en mémoire chaque insertion de case. Elle renvoie le tableau `S`.
Cette application nous donne une correspondance bijective exacte entre les mots réduits de l'élément le plus long de A_n et et les tableaux de Young standard de forme $(n, n-1, \cdots, 1)$ qui est étroitement liée à la correspondance dite de Robinson-Schensted-Knuth. Pour plus de détails sur le sujet (notamment les preuves associées à cette bijection), n'hésitez pas à jeter un œil à [3].

C Complexité du programme

Supposons que l'on travaille avec un système de Coxeter de rang n (c'est-à-dire sur un groupe à n générateurs de Coxeter), et que l'on travaille avec un mot de longueur l que l'on écrit à la base sur r lettres, dont on note I l'ensemble des décompositions réduites équivalents. On peut résumer les complexités des fonctions programmées tout au long de ce mémoire à l'aide du tableau suivant :

Fonction Sage	**Complexité dans le pire cas**	**Complexité moyenne (pour les fonctions utilisant le type `Set`)**
`longueur`	$O(\|\phi^+\|n^2)$	
`associate_root`	$O(\|\phi^+\|n^2)$	
`constructPartialSigma`	$O(\|j-i\|n^2)$	
`deletionConditionTheorem`	$O(r^3n^2)$	
`reduction`	$O(r^4n^2)$	
`indices_tresses`	$O(l\|\texttt{rel[0]}\|)$	
`appliquer_tresse`	$O(l)$	
`orbite_tresse`	$O(\|I\|n^2l^2+n^2\|I\|^2)$	$O(\|I\|n^2l^2)$
`graphe_mots_reduits`	$O(\|I\|n^2l^2+n^2\|I\|^2)$	$O(\|I\|n^2l^2)$
`evacuation_path`	$O(n)$	
`max`	$O(n^2)$	
`bijection_tableaux_mots`	$O(n^4)$	
`max_ligne`	$O(n)$	
`coxeter_knuth`	$O(n^2)$	
`bijection_mots_tableaux`	$O(n^4)$	

TABLE 1 – Complexité des différentes fonctions Sage programmées à travers ce mémoire

Références

[1] Anders Björner and Francesco Brenti. *Combinatorics of Coxeter Groups*, volume 231 of *Graduate Texts in Mathematics*. Springer, 2005.

[2] Nicolas Bourbaki. *Groupes et Algèbres de Lie, Chapitres 4 à 6*. Éléments de mathématique. Hermann, 1981.

[3] Paul H. Edelman and Curtis Greene. Balanced tableaux. *Advances in Mathematics*, 63(1) :42–99, 1987.

[4] James E. Humphreys. *Introduction to Lie Algebras and Representation Theory*, volume 9 of *Graduate Texts in Mathematics*. Springer, 1972.

[5] James E. Humphreys. *Reflection Groups and Coxeter Groups*, volume 29 of *Cambridge Studies in Advanced Mathematics*. Cambridge University Press, 1990.

Un grand merci à Tal GOTTESMAN de m'avoir encadré en stage l'été dernier et présenté ce sujet, et à Baptiste ROGNERUD d'avoir accepté d'encadrer ce projet ensuite.

Pour voir des exemples d'utilisation des différentes fonctions Sage présentées dans ce mémoire, n'hésitez pas à visiter le code ici.