TD3 - groupes et anneaux 2

RaphBi

12 mai 2024

Exercice 1. Exprimer le groupe

$$G = \left\{ \begin{pmatrix} a & 0 & d \\ 0 & b & 0 \\ 0 & 0 & c \end{pmatrix} \middle| a, b, c, d \in \mathbb{K}, abc = 1 \right\} < \mathrm{GL}_3(\mathbb{K})$$

comme un produit semi-direct $G = K \rtimes \mathrm{H}$ où $\mathrm{K} \cong \mathbb{K}$ et $H \cong \mathbb{K}^\times \times \mathbb{K}^\times$.

Exercice 2. Soit $\langle \_, \_ \rangle : \mathbb{R}^n \to \mathbb{R}$ le produit scalaire standat $\langle x, y \rangle = x^{\mathrm{T}} y$, et soit

$$\mathrm{O}_n = \{ A \in \mathrm{GL}_n(\mathbb{R}) \mid \langle Ax, Ay \rangle = \langle x, y \rangle \}$$

le groupe orthogonal.

(*i*) Montrer que $\det A = \pm 1$ pour tout $A \in \mathrm{O}_n$.

(*ii*) Exprimer le groupe O_n comme un produit semi-direct $\mathrm{O}_n = \mathrm{SO}_n \rtimes H$ où $\mathrm{SO}_n = \{ A \in \mathrm{O}_n \mid \det A = 1 \}$ et $H \cong \boldsymbol{\mu}_2$.

(*iii*) Montrer que, si n est impair, alors $\mathrm{O}_n \cong \mathrm{SO}_n \times \boldsymbol{\mu}_2$.

Exercice 3. Soit p un nombre premier et soit $0 < n < p$ un entier. Montrer que, si $G = K \rtimes H$ est un groupe qui s'écrit comme produit semi-direct de deux sous-groupes K et H de cardinal $|K| = n$ et $|H| = p$, alors $G \cong K \times \boldsymbol{\mu}_p$.

O(|V|+|E|*log(|V|))
$O(|V| + |E| \times \log(|V|))$

0.1 Autres algorithmes

Dijkstra et BFS Dans le projet, lorsqu'un joueur gagne au Hex, nous avons décidé de mettre en valeur son chemin gagnant. Dans un premier temps, nous devions détecter qu'un joueur ait remporté une partie. Pour cela nous avons implementé l'algorithme du parcours en largeur (BFS). Nous avons alors interprété le tableau dans lequel les coups des joueurs sont sauvegardés comme une graphe. Le parcous en largeur va générer un graphe couvrant depuis un bord. Ce graphe couvrant ne va passer que par les coups placés par un joueur. Si le graphe généré atteint le bord opposé, alors un chemin relie les deux bords, et donc le joueur a gagné. Nous appellons donc cet algorithme à chaque fois qu'un coup est placé. Sa complexité est $O(|V| + |E|)$. Notons ici que nous appelons l'algorithme seulement sur l'arbre correspondant aux coups joués par un joueur, donc en pratique, l'appel à cet algorithme est très rapide.
L'algorithme *BFS* nous assure qu'un joueur a gagné, mais il peut exister plusieurs chemins distincts reliant les deux bords. En effet, il peut y avoir plusieurs points de départ (points sur l'un des bords du gagnant) et points d'arrivée (points sur le bord opposé). Afin d'afficher le plus court chemin, nous avons décidé d'intégrer une version modifiée de l'algorithme de Dijkstra. Nous avons opté pour une solution naïve. En effet, nous appellons l'algorithme plusieurs fois (une pour chaque point de départ possible). Ensuite nous comparons la taille des chemins trouvés. Nous obtenons alors le plus court chemin, que l'on a choisi de mettre de jaune afin de faciliter sa visualisation.

À l'origine, dans le pire des cas, la complexité de l'algorithme de Dijkstra est $O(|V|+|E|\times\log(|V|))$. Cependant, notre implémentation est plus complexe, notre algorithme possède alors une complexité de $O(n \times (|V| + |E| \times \log(|V|)))$, avec n la taille de notre plateau. Notons que ce cas n'arrive en pratique jamais. En pratique le chemin le plus court est trouvé de façon instantanée.

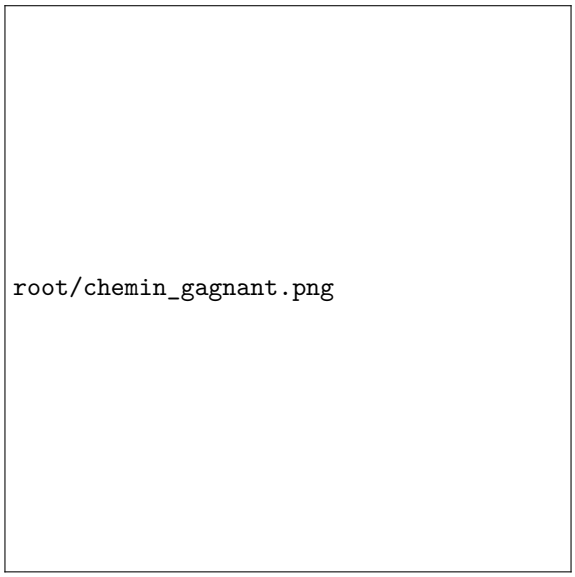

FIGURE 1 – Ici bleu a gagné, le chemin le plus court est trouvé