BARBOTAUD Nominoe LECHAT Pierre ROLLAND Stanislas 1C2

RAPPORT SAÉ 2.02

Introduction :

Le problème du cavalier est un problème classique dans le domaine des échecs et des algorithmes de recherche. Ce problème consiste à trouver un chemin pour un cavalier d'échecs qui visite chaque case exactement une fois sur un plateau d'échecs sans se retrouver bloqué. Le cavalier d'échecs se déplace selon un mouvement spécial en forme de "L", ce qui signifie qu'il peut sauter de deux cases dans une direction et une case dans une direction perpendiculaire, ce qui nous donne donc 8 mouvements possibles. Cette particularité de mouvement rend le problème du cavalier unique et stimulant à résoudre. Le défi réside dans la recherche d'un algorithme efficace capable de trouver une solution valide pour le parcours du cavalier sur un plateau d'échecs de taille donnée. En raison du grand nombre de possibilités de mouvement et de la complexité de l'espace de recherche, il peut être difficile de trouver une solution optimale pour des tailles de plateau importantes. Si nous avons choisis ce problème par rapport aux autres qui nous étaient proposés, c’est parce qu’il nous à parut être le plus intéressant à implémenter et à résoudre.

Description :

Nous avons fait 2 programmes pour résoudre le problème du tour du cavalier ; tous deux utilisant un algorithme de parcours en profondeur (Depth-First Search, DFS) avec un backtracking. Cet algorithme de recherche permet d’avancer autant que possible le long d'une branche d'arbre avant de reculer. On commence par choisir un nœud de départ, puis explore autant que possible le long de chaque branche avant de revenir en arrière. Dans le programme « parcoursCavalierDFS_V1 » que nous avons réalisé, nous avons implémenté 5 fonctions :

La fonction « coupValide » prenant comme paramètres :

- « sommet », une liste de liste représentant l’échiquier (chaque élément de la liste externe représente une rangée de l'échiquier, et chaque élément de la liste interne représente une case de cette rangée).

- « coup », un tuple représentant les coordonnées du coup à vérifier.

- « taille », la taille de l’échiquier.

Elle permet de vérifier si un coup est valide sur l’échiquier en vérifiant si les coordonnées du coup sont valides et si la case n’a pas été parcourue.

La fonction « coupsValides » prenant comme paramètres :

- « sommet », une liste de liste représentant l’échiquier (chaque élément de la liste externe représente une rangée de l'échiquier, et chaque élément de la liste interne représente une case de cette rangée).

- « coup », un tuple représentant les coordonnées du coup actuel.

- « taille », la taille de l’échiquier.

Elle permet de déterminer tous les coups possibles pour le prochain déplacement du cavalier à partir de la position actuelle du cavalier et vérifie si ces coups sont valides à l’aide de la fonction « coupValide ».

La fonction « parcoursCavalierDFS » prenant comme paramètres :

- « sommet », une liste de liste représentant l’échiquier (chaque élément de la liste externe représente une rangée de l'échiquier, et chaque élément de la liste interne représente une case de cette rangée).

- « coup », un tuple représentant les coordonnées du coup actuel.

- « position », la position actuelle du cavalier sur l’échiquier.

- « taille », la taille de l’échiquier.

Elle permet d’effectuer un parcours du cavalier en profondeur sur l’échiquier en plaçant le cavalier sur la case actuelle, puis pour chaque coup valide suivant, elle effectue récursivement un nouveau parcours en profondeur jusqu'à ce que toutes les cases soient visitées ou qu'aucun coup valide ne soit possible.

La fonction « parcoursCavalier » prenant comme paramètres :

- « taille », la taille de l’échiquier.

- « depart », un tuble représentant les coordonnées de départ du cavalier choisies par l’utilisateur.

Elle permet d’initialiser l’échiquier puis d’appeler la fonction « parcoursCavalierDFS » pour trouver un chemin hamiltonien (cycle) pour le cavalier. Si un chemin est trouvé elle renvoie l’échiquier avec le chemin hamiltonien trouvé sinon elle ne renvoie rien.

La fonction « programmePrincipal » :

- ne prends aucun paramètre.

Elle permet de demander à l’utilisateur de fournir la taille de l’échiquier et les coordonnées de départ du cavalier, puis, d’appeler la fonction « parcoursCavalier » et affiche si un chemin est trouvé ou non ainsi que le temps d’éxécution du programme.

Dans le programme « parcoursCavalierDFS_V2 » que nous avons réalisé, nous avons implémenté 4 fonctions :

La fonction « trouverParcoursHamiltonien » prenant comme paramètres :

- « taille », la taille de l’échiquier.
- « caseDepart », un tuple qui représente les coordonnées de la case initiale à partir de laquelle le parcours du cavalier commence son parcours.

Elle permet de rechercher un parcours hamiltonien pour le cavalier sur l’échiquier et de renvoyer le parcours hamiltonien trouvé sous forme de liste de coordonnées de cases ou rien si elle ne trouve aucun parcours.

La fonction « obtenirCasesAccessibles » prenant comme paramètres :

- « case », un tuple qui représente les coordonnées d’une case.

- « taille », la taille de l’échiquier.

Elle permet de déterminer les cases accessibles à partir de la case donnée en paramètre et elle retourne une liste de coordonnées accessibles à partir de la case donnée.

La fonction « demanderCoordonnees » :

- ne prends aucun paramètre.

Elle permet de demander à l'utilisateur quelle taille d'échiquier il désire utiliser ainsi que les coordonnées de la case de départ du cavalier sur l'échiquier. Elle renvoie un tuple contenant la taille de l'échiquier et les coordonnées de départ fournies par l'utilisateur.

La fonction « trouverParcours » :

- ne prends aucun paramètre.

Elle permet d'appeler la fonction « trouverParcoursHamiltonien » et affiche si un chemin est trouvé ou non ainsi que le temps d'éxécution du programme.

Comparaison :

Pendant la conception des deux programmes, nous avons décidé d'utiliser le même algorithme de résolution (DFS), cependant nous utilisons des approches différentes ; dans le programme « parcoursCavalierDFS_V1 », nous avons divisé le programme en plusieurs fonctions distinctes ayant chacune une tâche précise à réaliser telles que la validation des coups et la recherche du parcours pour résoudre le problème du cavalier. Le fait d'avoir plusieurs fonctions permet également d'avoir une meilleure lisibilité du code tandis que le programme « parcoursCavalierDFS_V2 » utilise moins de fonctions ce qui le rend moins long mais moins intuitif et moins compréhensible mais cette seconde approche est simple et directe en explorant chaque possibilité jusqu'à ce qu'elle trouve un parcours complet. En terme d'efficacité, le programme « parcoursCavalierDFS_V2 » est nettement plus rapide que le deuxième, malgré le fait que pour les deux programmes, la complexité et le coût en données sont proportionnels à la taille de l'échiquier. En résumé, le programme « parcoursCavalierDFS_V2 » peut être plus simple à implémenter en raison de sa structure plus linéaire, mais il peut être moins flexible en termes de modification et d'optimisation tandis que le programme « parcoursCavalierDFS_V1 » favorise une meilleure modularité et une séparation des préoccupations, ce qui peut faciliter la lisibilité, la maintenance et l'optimisation du code.

Conclusion :

Durant cette saé sur les graphes, nous avons pu nous pencher davantage sur le parcours en profondeur (DFS) pour la résolution de problème (ici le

parcours du cavalier) en essayant différentes approches algorithmiques. Pour améliorer nos programmes, nous aurions pu ajouter une interface graphique, afin que les résultats soient plus lisibles pour les utilisateurs. Nous aurions pu également essayer de résoudre le problème du cavalier en essayant d'autres approches algorithmique comme un parcours en largeur (BFS - Breadth-First Search) ou un parcours en Dijkstra par exemple. Autre amélioration possible, nous aurions pu essayer de coder dans d'autres langages pour voir si de grandes différences en termes de performances existaient (comme implémenter nos programmes en langage C par exemple).