Travail pratique #5

Travail présenté à
Ettore Merlo

De la part de
Miah,Mohammad Jamil - 2184862
Rouleau, Thomas - 2221053

Dans le cadre du cours de
INF2010 - Structures de données et algorithmes

26 novembre 2023
Polytechnique Montréal

Résultats

Pour alléger le rapport nous ne présentons que les résultats détaillés des algorithmes BFS et DFS pour le labyrinthe 3 seulement, mais les tendances remarquées sont les mêmes pour tous les labyrinthes.

La figure 1 présente, pour le labyrinthe 3, la taille de la pile (*stack size*) de l'algorithme BFS en fonction de l'itération. Ces données ont été obtenues en affichant à la console la taille de la pile à chaque début d'itération de l'algorithme BFS (BFSMaze.java) lors de l'exécution des tests (BFSMazeTest.java).

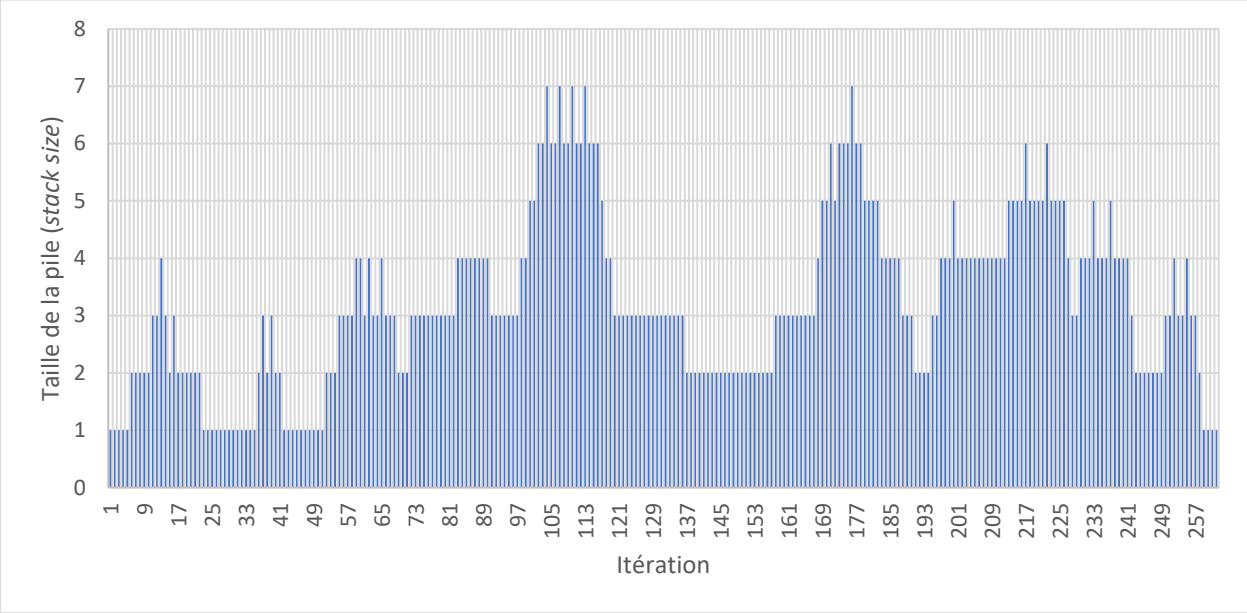

Figure 1 : Taille de la pile (*stack size*) de l'algorithme BFS en fonction de l'itération pour le labyrinthe 3

La figure 2 présente, pour le labyrinthe 3, la taille de la pile (*stack size*) de l'algorithme DFS en fonction de l'itération. Ces données ont été obtenues en affichant à la console la taille de la pile à chaque début d'itération de l'algorithme DFS (DFSMaze.java) lors de l'exécution des tests (DFSMazeTest.java).

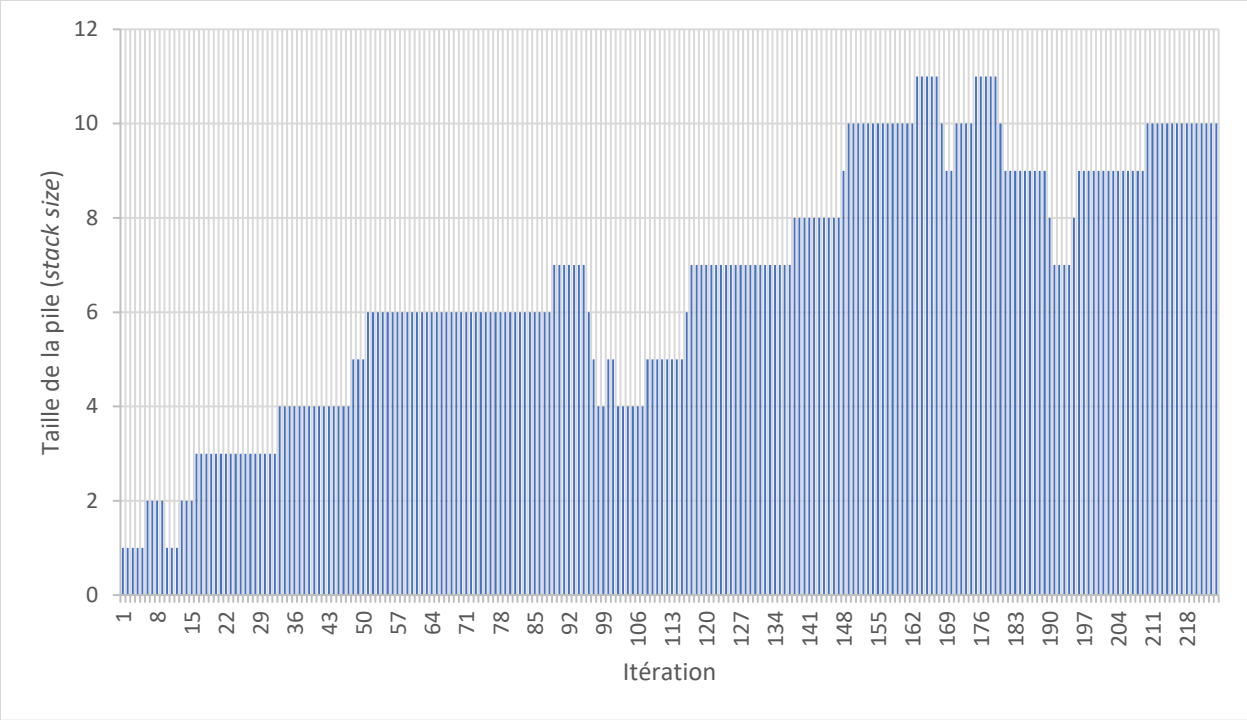

Figure 2 : Taille de la pile (*stack size*) de l'algorithme DFS en fonction de l'itération pour le labyrinthe 3

Le tableau 1 montre la taille maximale de la pile et le nombre de nœuds traversés par l'algorithme BFS en fonction du labyrinthe à résoudre. La taille maximale est obtenue à l'aide de la fonction suivante : *Math.max(counter.maxStackSize, counter.stackedNodes)*. Le nombre de nœuds traversés est obtenue en incrémentant de 1 un compteur à chaque itération de l'algorithme.

Tableau 1 : Taille maximale de la pile et nombre de nœuds/tuiles traversés par l'algorithme BFS en fonction du labyrinthe

Labyrinthe	Taille max de la pile (*max stack size*)	Nombre de nœuds/tuiles traversés
0	9	211
1	7	179
2	8	453
3	7	262
4	10	418
5	9	480
6	16	689
7	10	972
8	17	3154
9	30	10998

Le tableau 2 montre la taille maximale de la pile et le nombre de nœuds traversés par l'algorithme DFS en fonction du labyrinthe à résoudre. La taille maximale est obtenue à l'aide de la fonction suivante : *Math.max(counter.maxStackSize, counter.stackedNodes)*. Le nombre de nœuds traversés est obtenue en incrémentant de 1 un compteur à chaque itération de l'algorithme.

Tableau 2 : Taille maximale de la pile et nombre de nœuds/tuiles traversés par l'algorithme DFS en fonction du labyrinthe

Labyrinthe	Taille max de la pile (*max stack size*)	Nombre de nœuds/tuiles traversés
0	11	122
1	11	199
2	14	192
3	11	224
4	25	478
5	8	451
6	14	576
7	17	616
8	73	4024
9	95	9524

La figure 3 présente la taille maximale de la pile en fonction du labyrinthe parcouru pour l'algorithme BFS et DFS. Celui-ci est obtenue à l'aide des données des tableaux 1 et 2.

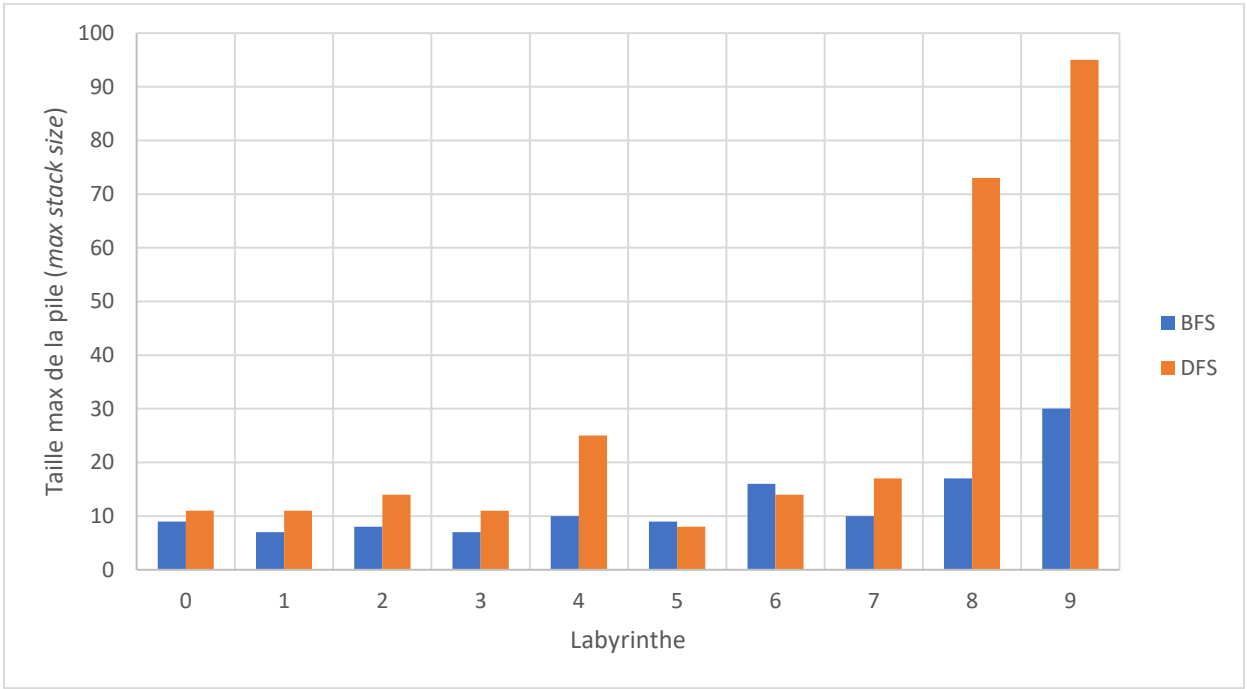

Figure 3 : Taille maximale de la pile en fonction du labyrinthe pour l'algorithme BFS et DFS

La figure 4 présente la taille maximale de la pile en fonction du labyrinthe parcouru pour l'algorithme BFS et DFS. Celui-ci est obtenue à l'aide des données des tableaux 1 et 2.

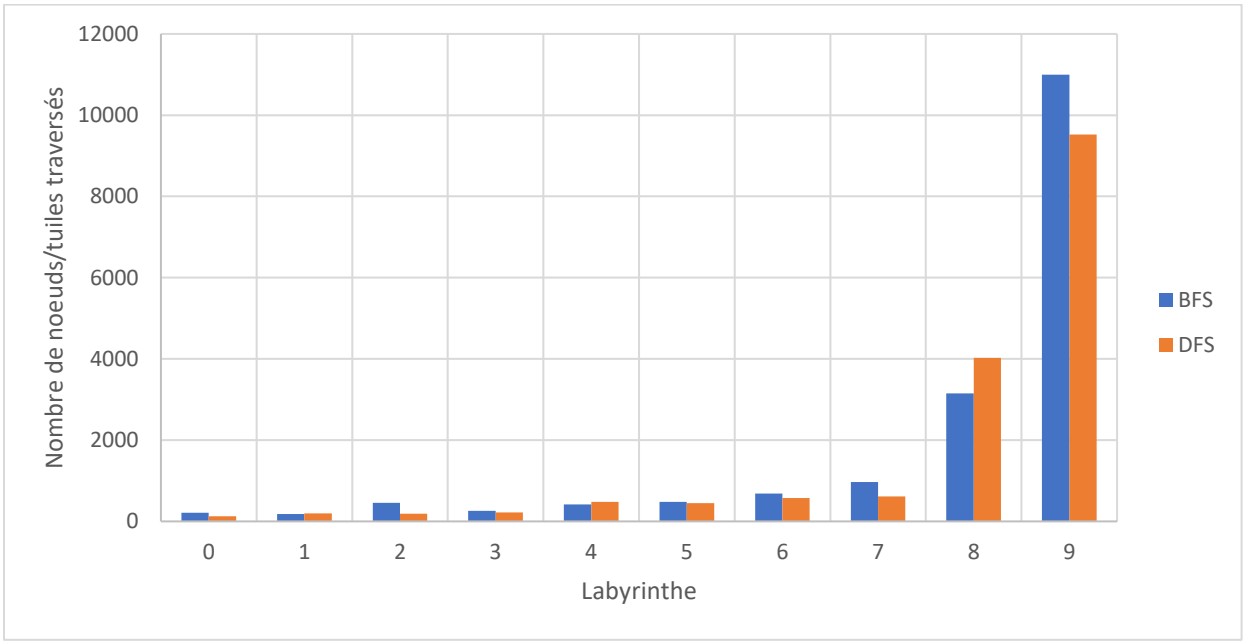

Figure 4 : Nombre de nœuds/tuiles traversés en fonction du labyrinthe pour l'algorithme BFS et DFS

Analyse des résultats

Les figures 1 et 2 semble montrer que la taille de la pile à une plus grande variance dans l'algorithme BFS et que dans l'algorithme DFS. La figure 3 semble montrer que la taille maximale de la pile est beaucoup plus élevée pour l'algorithme DFS par rapport à l'algorithme BFS. La figure 4, quant à elle, semble montrer que l'algorithme DFS traverse un nombre moins élevé de tuile que l'algorithme BFS. De plus, pour chacun des labyrinthe, l'algorithme DFS effectue moins d'itérations avant d'obtenir la réponse. Cette observation est démontrée dans les figures 1 et 2.

La moyenne de la taille maximale de la pile de l'algorithme BFS obtenus avec les données du tableau 1 est de 12 nœuds et celle de l'algorithme DFS obtenus avec les données du tableau 2 est de 28 nœuds. L'algorithme DFS est donc celui qui alloue le plus de mémoire en moyenne et l'algorithme BFS est celui qui alloue le moins de mémoires en moyennes.

La moyenne du nombre de tuiles traversés de l'algorithme BFS obtenus avec les données du tableau 1 est de 1782 tuiles et celle de l'algorithme DFS obtenus avec les données du tableau 2 est de 1641 tuiles. L'algorithme BFS est donc celui qui visite le plus de tuile en moyenne et l'algorithme DFS est celui qui visite le moins de tuile en moyenne.

Discussion

Le résultat obtenu montre que l'algorithme DFS est celui qui, en moyenne, utilise le plus de mémoire, mais qui visite le moins de tuiles alors que l'algorithme BFS utilise en moyenne le moins de mémoire, mais visite le plus de tuiles. Ces résultats sont expliqués par le fait que l'algorithme DFS à comme objectif de trouver un chemin et non le chemin le plus courts comme c'est le cas pour l'algorithme BFS. En explorant les tuiles/nœuds en profondeurs, l'algorithme DFS permet d'obtenir un chemin non optimal en un nombre moins grand d'itérations et de visite de tuiles. Cependant, cela vient avec un coût en mémoire plus élevés comme il revient moins souvent sur son chemin si la solution n'est pas optimale. Il est possible d'observer se phénomène de peu de variance de la taille de la pile dans la figure 2. L'algorithme BFS permet d'obtenir le chemin le plus court. Cette particularité est également la raison pour laquelle la taille moyenne de sa pile est plus faible, mais que son nombre de visite de tuile est plus élevés. La recherche du chemin optimale est partiellement observable à travers la plus grande variation de la taille de sa pile visible dans la figure 1.