POLYTECHNIQUE
MONTRÉAL

INF2010

Structures de données et algorithmes

TP5 : Graphes

Automne 2023

Soumis par:

Massoud, IBRAHIM

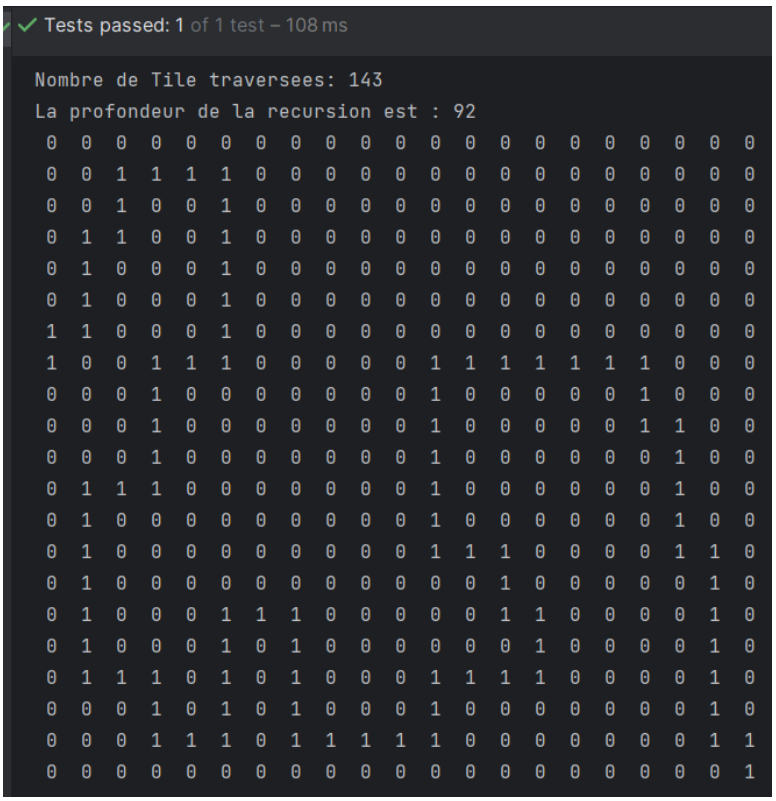

Fig1. Chemin trouvé avec l'algorithme DFS pour le labyrinthe input00.
(Les 1 correspondent aux nœud visités)

Fig2. Nœuds visités avec l'algorithme DFS pour le labyrinthe input00.
(Chemin trouvé : 0 *Backtracking* : 0)

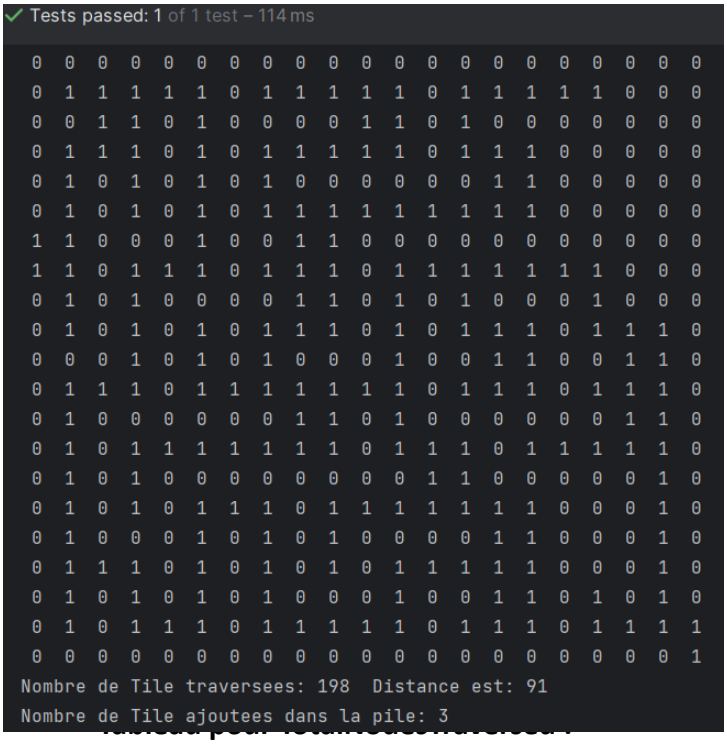

Fig3. Nœuds visités avec l'algorithme BFS pour le labyrinthe input00.
(Les 1 correspondent aux nœud visités)

Fig4. Nœuds visités avec l'algorithme BFS pour le labyrinthe input00

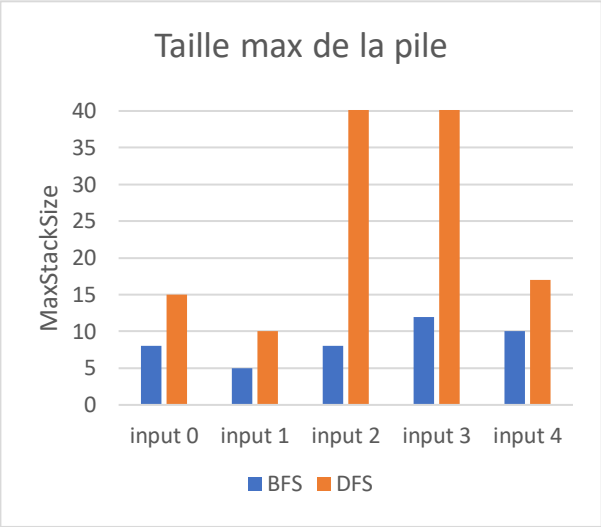

Figure 5 : Graphique du MaxStackSize pour les méthodes de BFS et DFS

Tableau pour MaxStackSize :

MaxStackSize	input 0	input 1	input 2	input 3	input 4
BFS	8	5	8	12	10
DFS	15	10	127	2070	17

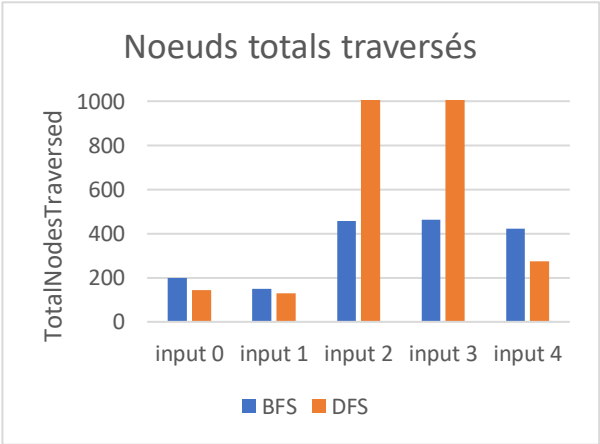

Figure 6 : Graphique du Total pour les méthodes de BFS et DFS

Tableau pour TotalNodesTraversed :

TotalNodes Traversed	input 0	input 1	input 2	input 3	input 4
BFS	198	151	457	462	421
DFS	143	129	1244	12746	273

Tableau des compteurs calculés pour l'algorithme de BFS pour les 10 labyrinthes

	Inputs des labyrinthes									
BFS	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
totalNodesTraversed	198	151	457	462	421	677	1013	1285	4641	8500
maxStackSize	8	5	8	12	10	13	15	12	19	30
Plus courte distance	91	56	96	107	134	142	172	254	646	684

Étude critique des résultats :

Par soucis d'équité, nous nous sommes limités aux résultats des tests d'input (0,1,2,3,4) car nous n'avons pas réussi à effectuer les tests DFS des autres inputs qui étaient trop longs. Nous montrons quand même les résultats obtenus pour BFS pour les 10 labyrinthes.

On remarque que pour tous les labyrinthes DFS allouent le plus de mémoires en ayant un plus grand MaxStackSize (figure5). Ceci est du au fait que l'algorithme explore en profondeur parfois le mauvais chemin et doit retourner sur ses pas ce qui remplit le MaxStackSize contrairement au BFS qui explore en largeur. Donc, selon nos observations sur les labyrinthes 0 à 4, l'algorithme de DFS alloue plus de mémoire en moyenne tandis que celui de BFS en alloue moins. Cela signifie donc que dans le cas de l'algorithme DFS, au cours des itérations, il y aura des moments où plus de nœuds seront conservés en mémoire puisqu'avec DFS il faut revenir sur ces pas (*backtracking*). Dans le cas BFS, le MaxStackSize correspond à tous les voisins de chaque nœud.

Ensuite, on remarque aussi que pour la majorité des labyrinthes DFS traverse le moins de nœuds (figure 6) car il trouve d'abord un chemin avant de traverser, ce qui n'est pas le cas de BFS. On peut voir la différence sur les résultats de input00 qui a été illustré par les figures 2 et 4 qui représentent les nœuds visités par la méthode BFS et DFS. Sur la figure 2 avec la méthode DFS, nous remarquons qu'il y a eu 147 nœuds traversés incluant les *backtrackings (en vert foncé)* pour trouver la sortie. Tandis que pour la méthode BFS, sur la figure 4, on remarque qu'il y a eu plus de nœuds visités. Sur certains labyrinthes, la méthode DFS trouve une sortie qui est plus longue que celle trouvée par BFS ce qui explique les résultats d'input (2 et 3) de la figure 6. Donc en moyenne, l'algorithme de BFS visite le plus de Tile en moyenne. Puis l'algorithme de DFS visite moins de Tile en moyenne.