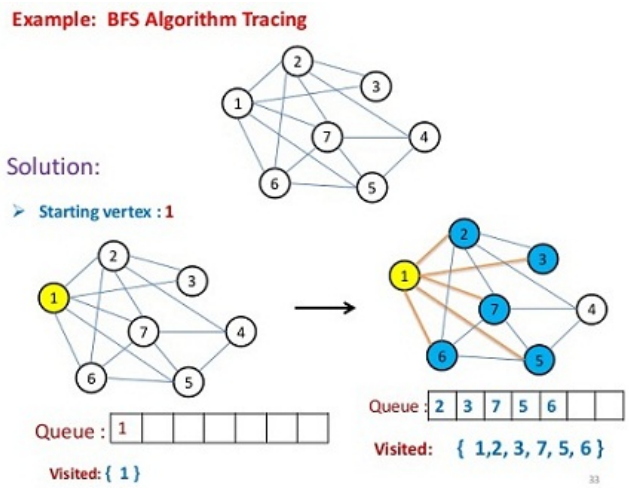
Breadth-First Search 原理



Breadth-First Search(廣度優先搜尋)：

1. 是用於遍歷或搜索樹或圖形數據結構的算法。 它從樹的根部（或圖的某個任意節點）開始，並探索當前深度的所有鄰居節點，然後再移至下一個節點，深度級別。

2. 是廣義的Level-Order Traversal，遇到的vertex就對該頂點所有鄰近的點進行Visiting」將使用情境從Tree推廣至Graph，類似於樹的廣度優先遍歷， 唯一的問題是，與樹不同，圖（Graph）可能包含循環，因此我們可能會再次來到同一節點。

3. BFS使用Queue（先進先出）而不是Stack，BFS在將頂點加入隊列之前檢查是否已發現頂點，而不是將此檢查延遲到頂點從隊列中出隊之前，Queue包含算法當前正在搜索的邊界。

4. 時間複雜度可以表示為O（| V | + | E |），因為在最壞的情況下將探索每個頂點和每個邊緣。
如果提前知道圖中的頂點數量，並且使用其他數據結構來確定哪些頂點已經添加到隊列中，則空間複雜度可以表示為| V |是一組頂點的基數。

Depth-First Search 原理

Depth-First Search（深度優先搜尋）：

1. 如同Pre-Order Traversal：「先遇到的vertex（頂點）就先Visiting」，並且以先遇到的vertex作為新的搜尋起點，直到所有「有edge相連的vertex」都被探索過。

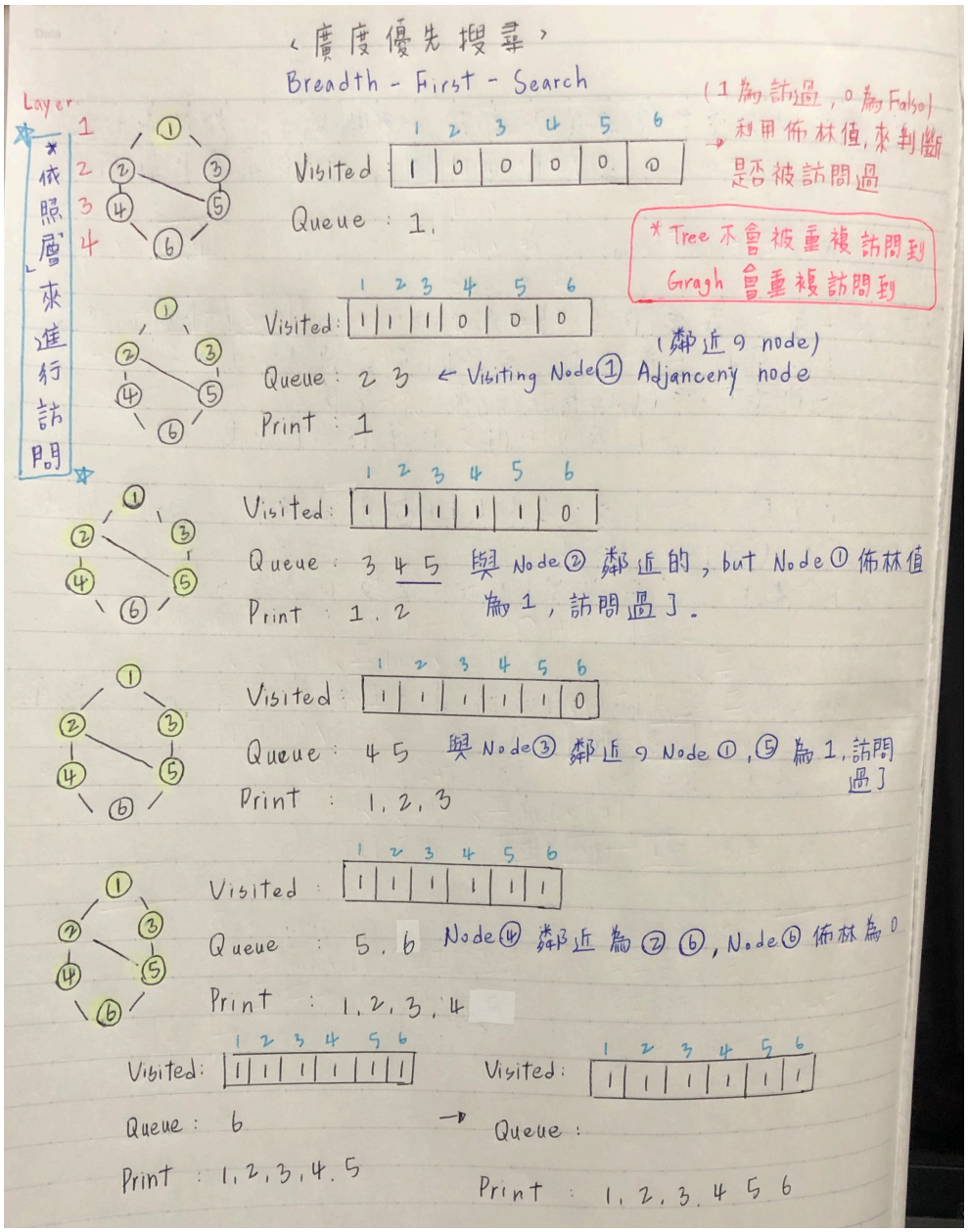
2. 以某一節點為出發點，不斷地前進拜訪未曾被拜訪過的節點， 直到無路可走或是所有相鄰的節點都已經拜訪過為止，然後再退回前一個節點，尋找 沒有拜訪過的節點，直到所有相鄰的節點都已被拜訪過。 因此，進行 depth-first search 時，需要使用 stack ，以便記錄所走過的路徑。

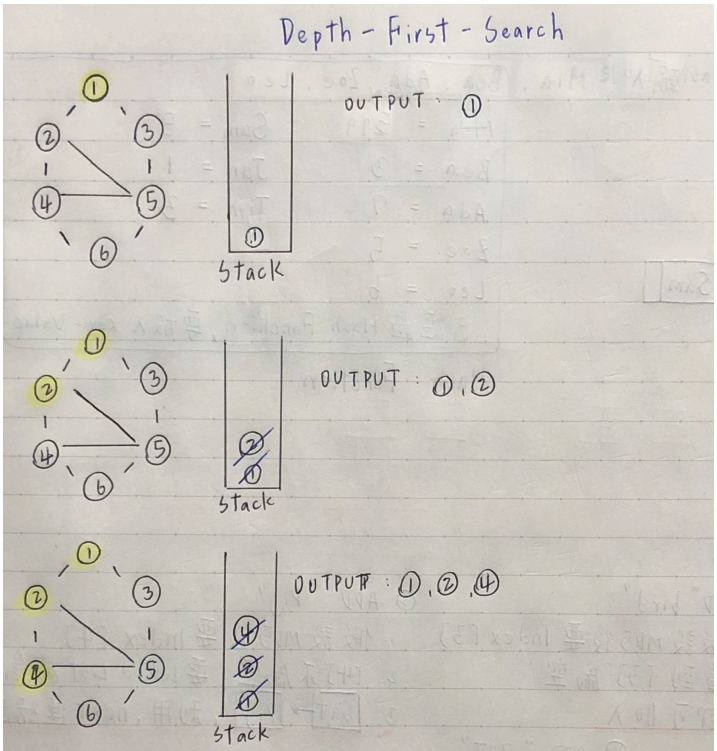
3. 在理論計算機科學中，DFS通常用於遍歷整個圖形，並花費時間O（| V | + | E |），在圖形大小上是線性的。在這些應用程序中，在最壞的情況下，它還會使用空間O（| V |）在當前搜索路徑上存儲頂點堆棧以及已訪問的頂點集。

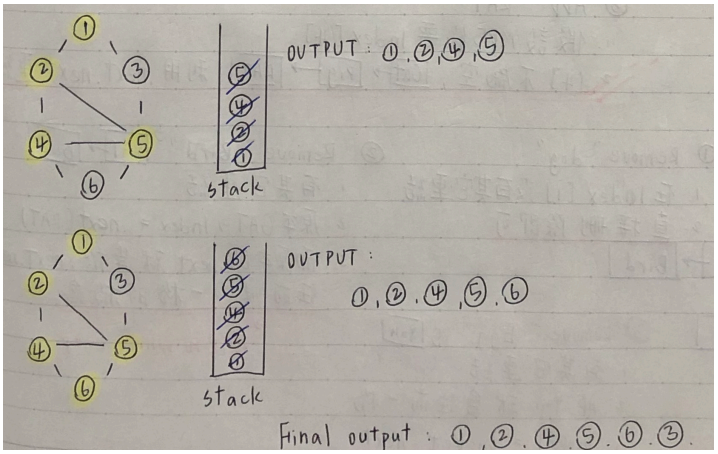
4. DFS變體的空間複雜度僅與深度限製成正比，因此，它比使用ＢＦＳ到相同深度所需的空間小得多。

5. 例如在人工智能或網絡爬網中尋找解決方案，要遍歷的圖通常太大或者無法完全訪問或無限訪問（DFS可能會遭受終止）。

BFS & DFS 流程圖


〈廣度優先搜尋〉
Breadth - First - Search
Layer
1
2
3
4
*依照層來進行訪問
Visited : 1 0 0 0 0 0
Queue : 1.
(1為訪過，0為False)
利用佈林值來判斷是否被訪問過
*Tree 不會被重複訪問到
Gragh 會重複訪問到
Visited : 1 1 1 0 0 0
Queue : 2 3 ← Visiting Node① Adjanceny node
(鄰近的 node)
Print : 1
Visited : 1 1 1 1 1 0
Queue : 3 4 5 與 Node② 鄰近的，but Node① 佈林值為1，訪問過了。
Print : 1 . 2
Visited : 1 1 1 1 1 0
Queue : 4 5 與 Node③ 鄰近的 Node①，⑤ 為1，訪問過了
Print : 1, 2, 3
Visited : 1 1 1 1 1 1
Queue : 5 . 6 Node④ 鄰近為 ② ⑥，Node⑥ 佈林為0
Print : 1, 2, 3, 4
Visited : 1 1 1 1 1 1
Queue : 6
Print : 1, 2, 3, 4, 5
Visited : 1 1 1 1 1 1
Queue :
Print : 1, 2, 3, 4 5 6


Depth - First - Search
OUTPUT : ①
Stack
OUTPUT : ①, ②
Stack
OUTPUT : ①, ②, ④
Stack


OUTPUT : ①, ②, ④, ⑤
Stack
OUTPUT :
①, ②, ④, ⑤, ⑥
Stack
Final output : ①, ②, ④, ⑤, ⑥, ③.

在廣度優先搜索（BFS）中，關鍵是它是基於級別或基於行的。 在每一級或每一行，從左到右或從右到左水平遍歷樹的節點。首先將Root加入其中從全局中取出第一個node，並檢驗它是否為目標。如果找到目標，則結束搜索並回傳結果，否則將其所有尚未檢驗過的直接子節點加入串聯中。
若類別為空，表示整張圖都檢查過了表示沒有欲搜索的目標。
在深度優先搜索（DFS）中，從頂部到底部或從底部到頂部垂直遍歷一棵樹， 過程一直進行到已發現從源可及性的所有例程為止。如果還存在發現的摘要，則選擇其中一個作為源例程並重複以上過程，整個過程重複進行直到直到所有例程都被訪問為止。深度優先搜索是圖論中的經典算法，利用深度優先搜索算法可以產生目標圖的相應拓撲排序表，利用拓撲排序表可以方便的解決很多相關的圖論問題，如最大路徑問題等等。

BFS	DFS
在BFS中，一次頂點被選中並被標記，然後相鄰的頂點被訪問並存儲在隊列中。 它比DFS慢。使用Queue數據結構來查找最短路徑。	執行兩個階段，首先將訪問的頂點推入堆棧，然後如果沒有頂點，則彈出訪問的頂點。使用Stack數據結構。
用於在未加權圖中查找單個源最短路徑，因為在BFS中，我們到達的頂點的邊距源頂點的邊數最少。	我們可能會遍歷更多邊以從源到達目標頂點。
更適合搜索更接近給定源的頂點。	如果有遠離源的解決方案，則DFS更適合。
BFS首先考慮所有鄰居，因此不適合用於遊戲或拼圖中的決策樹。	DFS更適用於遊戲或拼圖問題。
BFS的時間複雜度為O（V + E），其中V代表頂點，E代表邊。	DFS的時間複雜度也是O（V + E），其中V代表頂點，E代表邊。