22. Graph Algorithm

22.1 Graph Representation

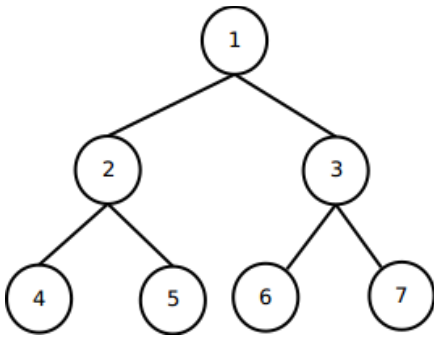
For given **Graph** (**G**) consisted by two fundamental elements: **Vertex** (**V**) and **Edge** (**E**), so we call a graph equivalent to **G=(V, E)**. Vertex is how many destination while edge is how many possible (or allowed) path to travel into all vertex. There are two type of graph: **undirected graph** and **directed graph**. There are two known data structure to represent graph: **Matrix Adjacency** using array has size $\Theta(V^2)$, but has O(1) to do searching and **List adjacency** using linked list that has size $\Theta(E)$, but has O(V) to do searching.

In the practice, there is no one best way to implementing vertex and index because the implementation is depend on programming language which used to built program. For example, some programming may used class to represent vertex in object oriented programming using java.

22.1-1 Given an adjacency-list represent of a directed graph, how long does it take to compute the out-degree of every vertex? How long does it take to compute the in-degrees?

Out-degree is number of outward edges from given vertex. Compute out-degree take O(E). In-degree is number of inward edges into given vertex. Compute out-degree take O(E*V).

22.1-2 Give an adjacency-list representation for a complete binary tree on 7 vertices. Give an equivalent adjacency-matrix representation. Assume that vertices are numbered from 1 to 7 as in binary heap.

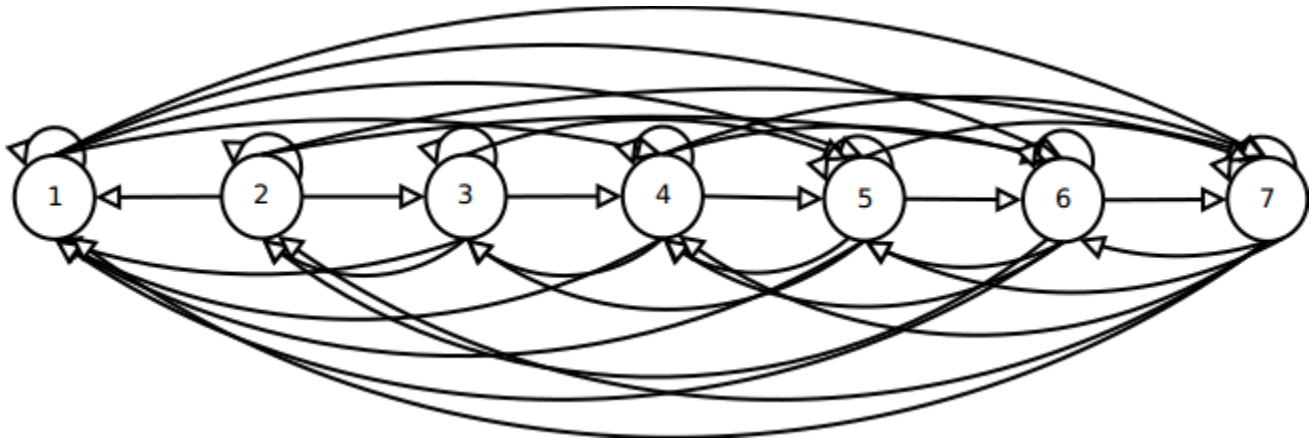


$1 \rightarrow 2 \rightarrow 3$
$2 \rightarrow 4 \rightarrow 5$
$3 \rightarrow 6 \rightarrow 7$

	1	2	3	4	5	6	7
1	0	1	1	0	0	0	0
2	0	0	0	1	1	0	0
3	0	0	0	0	0	1	1
4	0	1	0	0	0	0	0
5	0	1	0	0	0	0	0
6	0	0	1	0	0	0	0
7	0	0	1	0	0	0	0

22.1-3 The **transpose** of directed graph G = (V, E) is graph $G^T = (V, E^T)$, where $E^T = [(v,u) \in V \times V : (u,v) \in E]$. Thus G^T is G with all its edges reversed. Describe efficient algorithms for computing G^T from G, for both the adjacency-list and adjacency-matrix representations of G. Analyze the running times of your algorithms.

Below, the transpose of 7 vertices from problem 22.1-2:



1 → 1 → 4 → 5 → 6 → 7 2 → 1 → 2 → 3 → 6 → 7 3 → 1 → 2 → 3 → 4 → 5 4 → 1 → 3 → 4 → 5 → 6 → 7 5 → 1 → 3 → 4 → 5 → 6 → 7 6 → 1 → 2 → 4 → 5 → 6 → 7 7 → 1 → 2 → 4 → 5 → 6 → 7	

	1	2	3	4	5	6	7
1	1	0	0	1	1	1	1
2	1	1	1	0	0	1	1
3	1	1	1	1	1	0	0
4	1	0	1	1	1	1	1
5	1	0	1	1	1	1	1
6	1	1	0	1	1	1	1
7	1	1	0	1	1	1	1

Algorithm to transpose adjacency matrix:

```
for i = 0 to V-1:
    for j = 0 to V-1:
        if AdjMatrix[i][j] == 0:
            AdjMatrix[i][j] = 1
        else:
            AdjMatrix[i][j] = 0
```

Running Time: $O(V^2)$

Algorithm to transpose adjacency list using priority queue:

```
for i = 0 to (V - 1):
    j = 0

    do:
        v[j] = dequeue(node[i])
        j++
    while(v[j-1] != 0)

    l = 0
    for k = 0 to (V - 1):
        if (k+1 is not v[l]) or (tmp[l] is 0):
            enqueue(node[i], k+1)
```

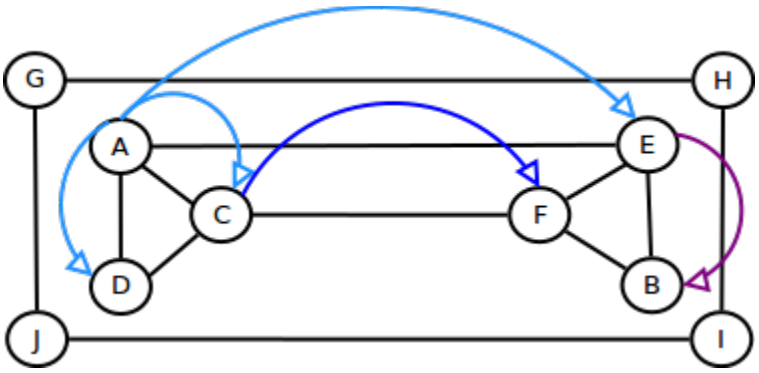
```
        else:
            if l < j:
                l++
```

Running Time: $O(2V^2)$

22.2 Breath First Search (BFS)

BFS is categorized as blind search algorithm. BFS read all neighbor of current vertex until reach vertex that has no neighborhood. In tree representation, BFS will traversing from root into each level of tree until reach leaf. Since this algorithm need to traversing into each level of tree, it need to store every vertex in current level into memory. To avoid restore or visit already visited vertex, we need to marked every vertex that just visited. Thus running time of this algorithm is O(V+E).

Take a look for a graph below:

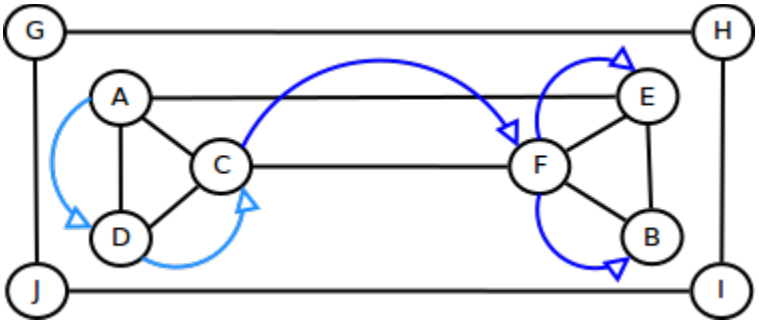


If we want to traversing a path which started from vertex A, BFS will show you A-D-C-E-F-B. Notice, we need to scan all child of current vertex. Queue commonly used to store all child of current vertex, after all child stored and marked, then we dequeue each stored child of last vertex as current vertex and iterate over its child. That's pretty easy.

22.3 Depth First Search (DFS)

DFS is categorized as blind search algorithm. DFS read all vertex that connected directly with current vertex until all vertex scanned. In tree representation, DFS will walking from root into each first found connected vertex in each level until reach a leaf. This algorithm can be implemented easily using recursive strategy. Because we need to scan all vertex and edge in a path, thus the running time is O(V+E).

Take a look for a graph below:



If we want to traversing a path which started from vertex A, DFS will show you A-D-C-F-E-B.