Non-Linear Data structures- Graphs

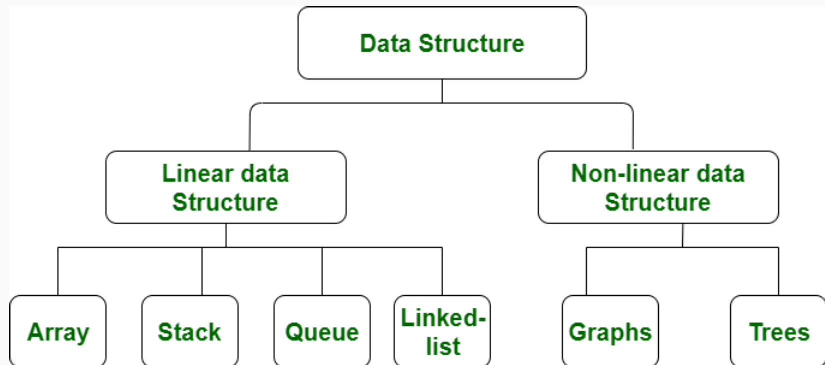
Dr. Sunil Kumar P. V.

Assistant Professor
SCOPE

Overview

- Graph Definition & Types
- Graph Representation
- Graph Traversals- DFS & BFS
- Minimum Spanning Trees- Prim's & Kruskal's
- Single Source Shortest Path Problems- Dijkstra's

Data structure Classification



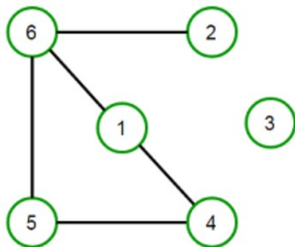
Credit:geeksforgeeks

Graph Definition & Types

What is a graph?

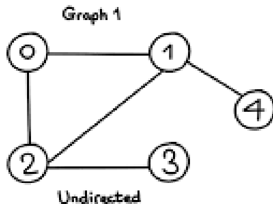
- A graph is an ordered pair $G = (V, E)$
- Set V – Collection of vertices/nodes
- Set E – Collection of pairs of vertices from V called edges

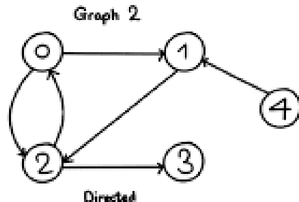
```
V = { 1, 2, 3, 4, 5, 6 }
E = { (1, 4), (1, 6), (2, 6), (4, 5), (5, 6) }
```



Types of graphs

- Undirected:-
  - Edges have no direction/orientation.
  - edge $(x, y) =$ edge (y, x)
- Directed (Digraph):-
  - Edges have direction/orientation.
  - edge (x, y) is not identical to edge (y, x)

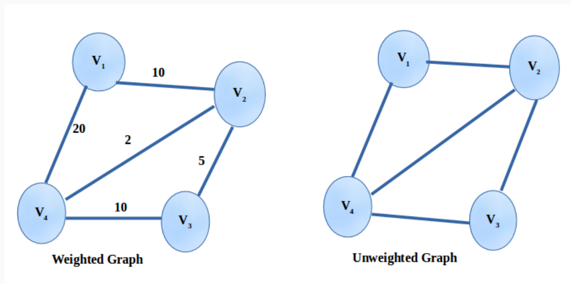




Credit:towardsdatascience.com

Types of graphs

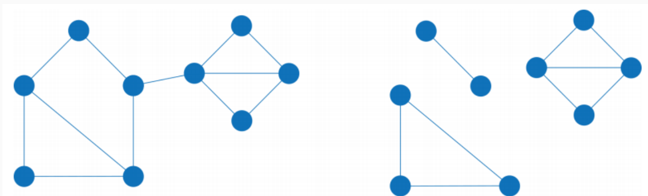
- Weighted:-
  - A weighted graph associates a value (weight) with every edge in the graph
- Unweighted:-
  - An unweighted graph doesn't have a value (weight) associated with any edge in the graph



Credit:towardsdatascience.com

Types of graphs

- A path is a sequence of alternating vertices and edges such that each edge connects each successive vertex, with no edge is repeated
- A cycle is a path that starts and ends at the same vertex
- Connected:-
    - A connected graph has a path between any pair of vertices
- Disconnected:-
    - A graph which is not connected



Graph Representation

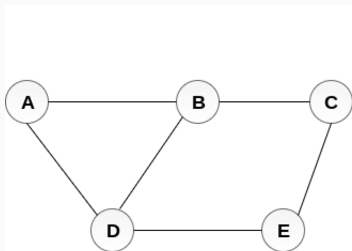
Representation of a Graph

1. Adjacency Matrix
2. Incidence Matrix
3. Adjacency List

Adjacency Matrix: Undirected graph

- Vertices along rows and columns of the $|V| \times |V|$ matrix

$$M[i][j] = \begin{cases} 1, \text{if } i \text{ and } j \text{ are adjacent} \\ 0, \text{otherwise} \end{cases}$$



Undirected Graph

	A	B	C	D	E
A	0	1	0	1	0
B	1	0	1	1	0
C	0	1	0	0	1
D	1	1	0	0	1
E	0	0	1	1	0

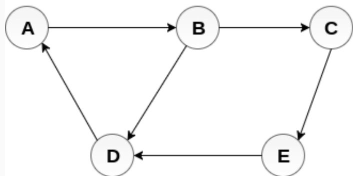
Adjacency Matrix

Credit: javatpoint.com

Adjacency Matrix: Directed graph

- Vertices along rows and columns of the $|V| \times |V|$ matrix

$$M[i][j] = \begin{cases} 1, \text{if } i \text{ and } j \text{ are adjacent} \\ 0, \text{otherwise} \end{cases}$$



Directed Graph

	A	B	C	D	E
A	0	1	0	0	0
B	0	0	1	1	0
C	0	0	0	0	1
D	1	0	0	0	0
E	0	0	0	1	0

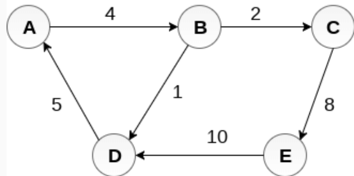
Adjacency Matrix

Credit: javatpoint.com

Adjacency Matrix: Weighted graphs

- Vertices along rows and columns of the $|V| \times |V|$ matrix

$$M[i][j] = \begin{cases} w_{ij}, \text{the weight of edge } (i, j) \\ 0, \textit{if } \; i \text{ and } j \text{ are not adjacent} \end{cases}$$



Weighted Directed Graph

	A	B	C	D	E
A	0	4	0	0	0
B	0	0	2	1	0
C	0	0	0	0	8
D	5	0	0	0	0
E	0	0	0	10	0

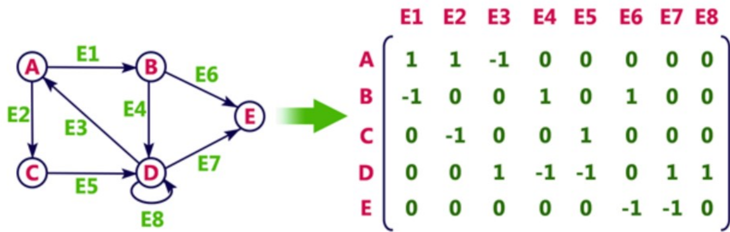
Adjacency Matrix

Credit: javatpoint.com

Incidence Matrix

- Vertices along columns and edges along rows of the $|V| \times |E|$ matrix

$$M[i][j] = \begin{cases} 1, & \text{if the edge } j \text{ is outgoing from vertex } i \\ -1, & \text{if the edge } j \text{ is incoming to vertex } i \\ 0, & \text{if the edge } j \text{ is not connected to vertex } i \end{cases}$$

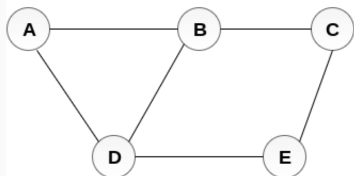


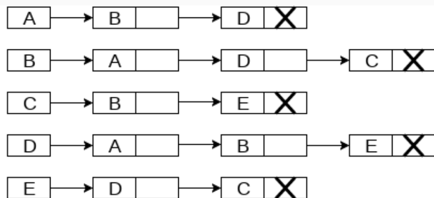
	E1	E2	E3	E4	E5	E6	E7	E8
A	1	1	-1	0	0	0	0	0
B	-1	0	0	1	0	1	0	0
C	0	-1	0	0	1	0	0	0
D	0	0	1	-1	-1	0	1	1
E	0	0	0	0	0	-1	-1	0

Adjacency List

- Adjacency list is a linked representation.
- In this representation, for each vertex in the graph, we maintain a linked list of its neighbours.

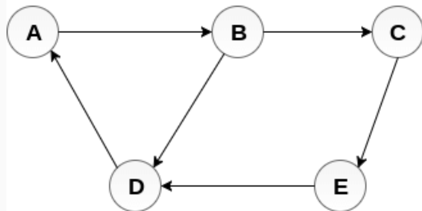
Adjacency List: Undirected graph

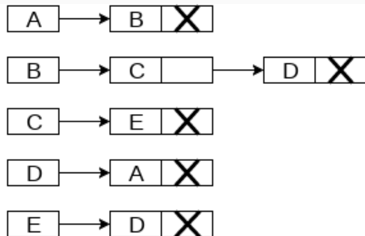




Credit:javatpoint.com

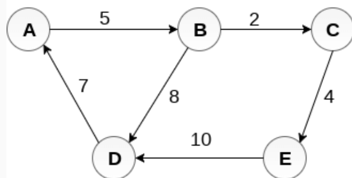
Adjacency List: Directed graph

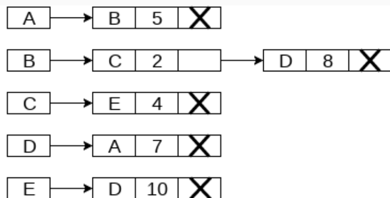




Credit:javatpoint.com

Adjacency List: Weighted graph





Credit:javatpoint.com

Graph Traversals- BFS & DFS

Graph Traversals

Algorithms

- Breadth First Search (BFS)
- Depth First Search (DFS)

Difference

- Visiting of a vertex
- Exploration of a vertex

Demonstration:

- On virtual board

BFS Algorithm

```
function BFS(G,s)                    ▷ G is graph, s is suorce vertex
    Q = Φ
    Q.enqueue(s)
    Result = {s}
    while Q ≠ Φ do
        v = Q.dequeue(s)
        for all neighbours w of v in graph G do
            if w is not visited then
                Q.enqueue(w)
                Result = Result ∪ {w}
            end if
        end for
    end while
    return Result
```

DFS Algorithm

```
function DFS(G,s)
    stk = Φ
    Result = {s}
    while stk ≠ Φ do
        v = stk.pop()
        for all neighbours w of v in Graph G do
            if w is not visited then
                stk.push(w)
                Result = Result ∪ {w}
            end if
        end for
    end while
    return Result
end function
```

Points to Note

- Answers to BFS and DFS are not unique. It depends on the order of visit
- Complexity of BFS and DFS are $O(V + E)$
- Why?
- What is the BFS and DFS of a binary tree??

Points to Note

- Answers to BFS and DFS are not unique. It depends on the order of visit
- Complexity of BFS and DFS are $O(V + E)$
- Why?
- What is the BFS and DFS of a binary tree??
- BFS: Level order traversal
- DFS: Preorder traversal

Minimum Spanning Trees

What are Spanning Trees?

- Graph $G(V, E)$
- $V =$
- $E =$
- Spanning tree is a subgraph
- All vertices ($|V|$)
- $|V| - 1$ edges
- If Spanning tree is $S = (V', E')$
- $S \subseteq G$ with $|V'| = |V|$ and $|E'| = |V| - 1$
- Several spanning trees possible

Minimum Cost Spanning Trees (MST)

- A minimum spanning tree (MST) or minimum weight/cost spanning tree is a subset of the edges of a connected, edge-weighted undirected graph that connects all the vertices together, without any cycles and with the minimum possible total edge weight
- Algorithms to find MST are
  - Prims Algorithm
  - Kruskal Algorithm

Spanning Tree Properties

- Removing one edge from ST will make it disconnected
- Adding one edge from ST will create a cycle
- If each edge has distinct weight, the minimum spanning tree will be unique
- Every connected graph has at least one ST
- Disconnected graph has no ST

Prims Algorithm: Demonstration

- On virtual board

Prims Algorithm

1. Remove all self loops and parallel edges, if any
2. Initialize the minimum spanning tree with a vertex chosen at random.
3. Find all the edges that connect the tree to new vertices, find the minimum and add it to the tree
4. Keep repeating step 2 until we get a minimum spanning tree

Analysis: $|V| - 1$ edges are to be selected from $|E|$ edges
$\Rightarrow O(|V||E|) = O(|V|^2)$

Kruskal Algorithm: Demonstration

- On virtual board

Kruskal Algorithm

1. Remove all self loops and parallel edges, if any
2. Sort all the edges from low weight to high
3. Take the edge with the lowest weight and add it to the spanning tree. If adding the edge created a cycle, then reject this edge.
4. Keep adding edges until we get a spanning tree.

Analysis: $|V| - 1$ edges are to be selected from $|E|$ edges
$\Rightarrow O(|V||E|) = O(|V|^2)$

Single Source Shortest Path Problem- Dijkstra's Algorithm

Single Source Shortest Path Problem

- What is single source shortest path problem?
- Dijkstra's algorithm demo: On virtual board

Dijkstras Algorithm

Initialisation Step:-

- *dist*, an array of distances from the source node s to each node in the graph, initialized the following way: $dist(s) = 0$; and for all other nodes v, $dist(v) = \infty$. This is done at the beginning because as the algorithm proceeds, the *dist* from the source to each node v in the graph will be recalculated and finalized when the shortest distance to v is found
- Q, a queue of all nodes in the graph. At the end of the algorithm's progress, Q will be empty.
- S, an empty set, to indicate which nodes the algorithm has visited. At the end of the algorithm's run, S will contain all the nodes of the graph.

Dijkstras Algorithm

Algorithm:-

1. While Q is not empty, pop the node v, that is not already in S, from Q with the smallest $dist(v)$. In the first run, source node s will be chosen because $dist(s)$ was initialized to 0. In the next run, the next node with the smallest *dist* value is chosen.
2. Add node v to S, to indicate that v has been visited
3. Relaxation step: Update *dist* values of adjacent nodes of the current node v as follows: for each new adjacent node u,
   - if $dist(v) + weight(u, v) < dist(u)$, there is a new minimal distance found for u, so update $dist(u)$ to the new minimal distance value;
   - otherwise, no updates are made to $dist(u)$.

Dijkstra's Algorithm

4. The algorithm has visited all nodes in the graph and found the smallest distance to each node. *dist* now contains the shortest path tree from source s.

Analysis:-

- For any vertex v, in worst case $|V|$ vertices to be relaxed. Hence, $O(|V|^2)$

Drawback of Dijkstra's Algorithm

- Failure cases

Thank You