Data Structures and its Applications

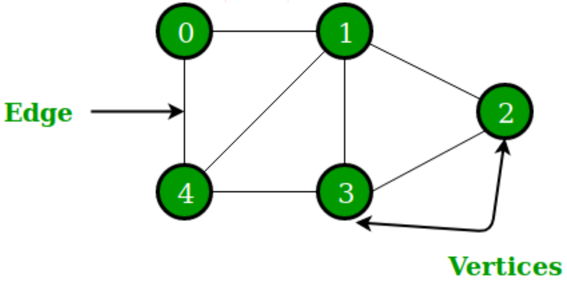
Dr. K. Sita Ramana
Associate Professor
Department of Mathematics,
Malla Reddy University, Hyderabad

I must extend my sincere thanks to Dr. P. Praveen Kumar and Dr. Sreenath Itikela for their assistance in gathering data, advice, and ideas during the creation of these slides.

Introduction

- Graphs are fundamental mathematical structures used to represent and analyze relationships between objects.
- In the context of computer science and mathematics, a graph is a collection of nodes (also known as vertices) connected by edges i.e, A graph G is a pair of sets (V,E), where V is a set of vertices or nodes and E is a set of edges.

Here is the example of Graph.



Types of Graphs

- **Directed Graph** : The digraph, which is a collection of sets of vertices and edges, is another name for the directed graph. Each edge in this case will be directed and connected by an order pair of vertices. In this graph, we can traverse only in the direction (uni direction) pointed by the edges.
- **Undiredcted Graph** : An undirected graph is a type of graph where the edges have no specified direction assigned to the them. It is also called as bidirectional graph. We can traverse an undirected graph in either direction.
- **Weighted Graph** : A graph that has a weight, i.e a certain numeric value associated with its edges is known as aweighted graph. Weighted graphs can be both directed as well as undirected. The weight associated with the edges may represents the cost to travel between the connected nodes or distance between the connected nodes and so on.
- **Multigraph** : If one allows more than one edge to join pair of vertices, then such graph is called Multigraph.

Graph - Properties

Several characteristics of Graph are as follows.

- **Node (Vertex)**: The points where edges meet in a graph are known as vertices or nodes. A vertex can represent a physical object, concept, or abstract entity, usually represented by a dot in a graph.
- **Edges**: The connections between vertices are known as edges. They can be undirected (bidirectional) or directed (unidirectional).
- **Loop** : It is an edge from a vertex to itself. A graph without loops is called simple graph
- **Weight**: A weight can be assigned to an edge, representing the cost or distance between two vertices.
- **Degree**: The degree of a vertex is the number of edges incident (connected to) on vertex. The degree of a vertex is determined by counting each loop incident on it twice and each other edge once.
- **In-degree and Out-degree**: In a directed graph, the in-degree of a vertex is the number of edges that point to it, and the out-degree is the number of edges that start from it.

Graph - Propeties

- **Order**: The number of vertices in the graph is called Order of the graph.
- **Size**: The number of edges present in the graph is called size of the graph.
- **Path**: A path is a sequence of vertices that are connected by edges. A simple path does not contain any repeated vertices or edges. If the end vertices of a path are same then it is called closed path othewise open path.
- **Cycle**: A cycle is a path that starts and ends at the same vertex. A simple cycle does not contain any repeated vertices or edges.
- **Adjacent Vertices**: If the two vertices are connected by an edge, then they are called adjacent vertices.
- **Connected Graph**:A graph is said to be connected iff each pair of its vertices is connected, otherwise called disconnected graph.
- **Bipartiteness**: A graph is said to be bipartite if its vertices can be divided into two disjoint sets such that no two vertices in the same set are connected by an edge.

Graph - Propeties

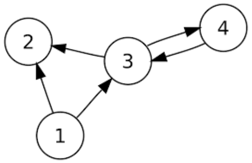
- **DAG (Directed Acyclic Graph)**: A directed acyclic graph is a directed graph that does not contain any cycles. DAGs are commonly used to represent dependencies, scheduling, and directed flow of information or tasks.

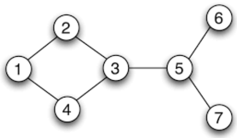
a component of an undirected graph is a connected subgraph that is not part of any larger connected subgraph.

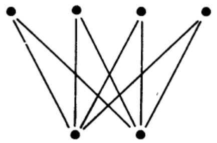
- **Components**: A component is a subgraph of an undirected graph where any two vertices are connected by a path. If there are disconnected subsets of vertices in a graph, it can have more than one component. Strong components in a directed graph are subsets of vertices where there is a directed path between any two vertices.
- **Planarity**: A graph is said to be planar if it can be drawn on a plane without any edges crossing each other. Planarity is an important concept in graph theory and has applications in areas such as circuit design and network topology.

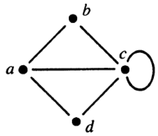
These characteristics define the fundamental properties and structure of graphs. Understanding these properties is crucial for analyzing and solving graph-related problems and designing efficient algorithms for graph traversal, connectivity, and optimization.

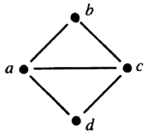
Examples

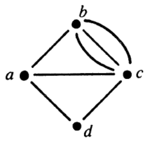


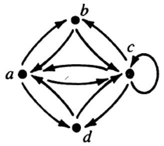








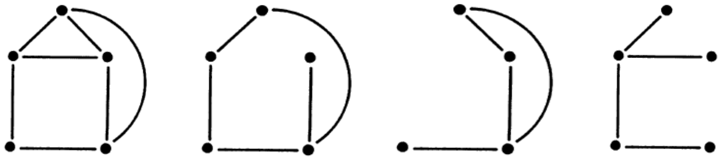




A directed graph, An undirected graph, bipartiate graph, a nonsimple graph, simple graph, a multigraph and a symmetric directed graph

Subgraph

- **Subgraph**:If G and H are graphs then H is a subgraph of G iff V(H) [set of vertices of H] is a subset of V(G) [set of vertices of G] and E(H) [set of edges of H] is a subset of E(G) [set of edges of G]. A subgraph H of G is called a spanning subgraph of G iff V(H) = V(G).

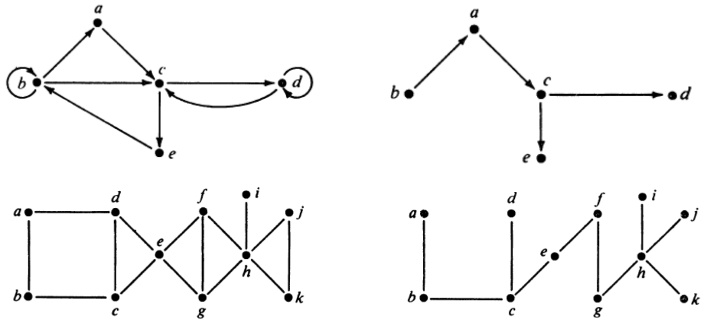


A Graph and some of its subgraphs

Spanning Tree

- **Spanning Tree**:A subgraph H of a graph G is called a spanning tree of G if
  - H is a tree, and
  - contains all the vertices of G.

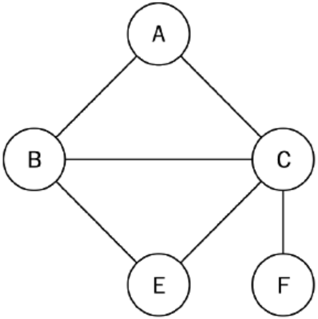
A spanning tree that is a directed is called a directed spanning tree.



A Graphs and its Spanning Trees

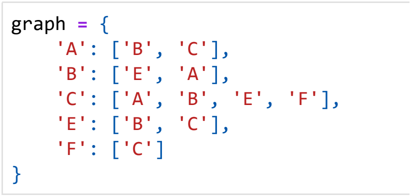
Representation of Graph

Graphs can be represented in many ways. Representing a Graph using a dictionary is more appropriate to implement using python. To illustrate we use the following graph.

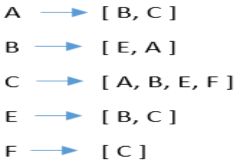


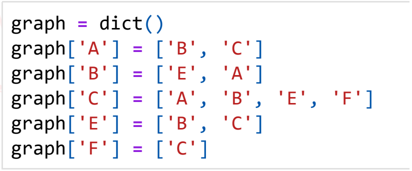
Representation of Graph

The above graph adjacency list and its dictionary representation in python is given below.



```python
graph = {
    'A': ['B', 'C'],
    'B': ['E', 'A'],
    'C': ['A', 'B', 'E', 'F'],
    'E': ['B', 'C'],
    'F': ['C']
}
```





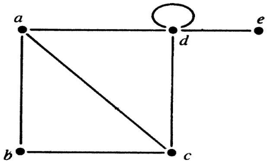
```python
graph = dict()
graph['A'] = ['B', 'C']
graph['B'] = ['E', 'A']
graph['C'] = ['A', 'B', 'E', 'F']
graph['E'] = ['B', 'C']
graph['F'] = ['C']
```

A Graph's adjacency list and its dictionary representation

Representation of Graph - Adjacency Matrix

Adjacency Matrix, which is based on the adjacency of vertices or nodes, is an alternate method of representing a graph..

- If $v_1, v_2, v_3, \ldots., v_n$ are the vertices of G, then the adjacency matrix for this ordering of the vertices of G is then $n \times n$ matrix A, where the ij^{th} element A_{ij} of A is 1 iff the edge $\{v_i, v_j\}$ is an edge of G; otherwise, $A_{ij} = 0$.
- A is a symmetric matrix each of whose entries either 0 or 1.
- Moreover, the digit one will appear on the i^{th} position of the diagonal of A iff there is a loop at v_i.
- If we change the ordering of the vertices of G, then the entries of A will be rearranged.



$$\begin{array}{c|ccccc} & a & b & c & d & e \\ \hline a & 0 & 1 & 1 & 1 & 0 \\ b & 1 & 0 & 1 & 0 & 0 \\ c & 1 & 1 & 0 & 1 & 0 \\ d & 1 & 0 & 1 & 1 & 1 \\ e & 0 & 0 & 0 & 1 & 0 \end{array}$$

Graph Traversals

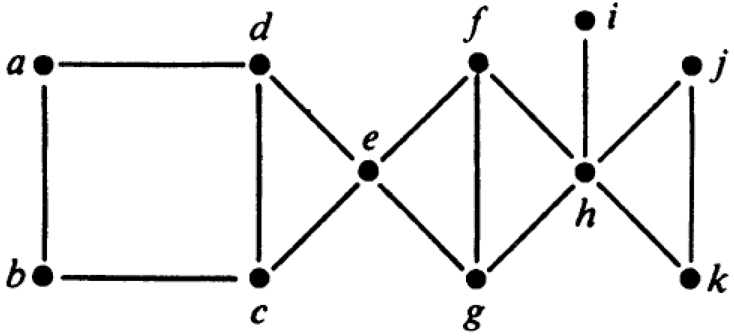
- Since ordered structure is not always present in graphs, traversing a graph is more difficult.
- Keeping track of which nodes or vertices have already been visited and which ones have not is a typical requirement of traversal.
- A common strategy is to follow a path until a dead end is reached, then walking back up until there is a point where there is an alternative path.
- We can also iteratively move from one node to another in order to traverse the full graph or part of it.
- Traversing a Graph will leads to finding a Spanning Tree of it.

Finding Spanning Tree of a Graph

- There are two efficient algorithms to find a spanning tree of a connected graph, namely Breadth First Search and Depth First Search.
- The idea of breadth-first search is to visit all vertices sequentially on a given level before going onto the next level.
- Depth-first search, on the other hand, proceeds successively to higher levels at the first opportunity.
- Traversals of the Inorder, Preorder, and Postorder will fall under the Depth First Search technique.
- Under the Breadth First Search approach, Levelorder will be included.

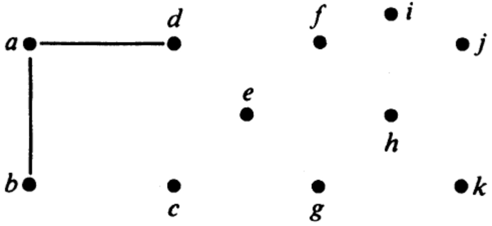
Breadth First Search

Letus illustrate Breadth First Search (BFS) on the following graph.



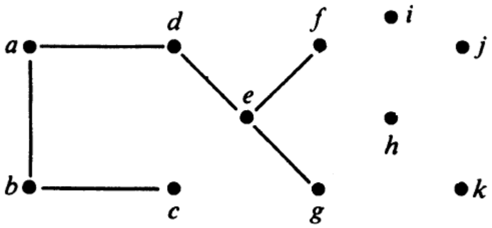
Breadth First Search

- We select the ordering of the vertices *abcdefghijk*
- Then select vertex a as the first vertex and designate it as root.
- Add all edges {a,x}, as x runs in order from b to k, that do not produce a cycle. Thus, we add {a,b}and {a,d}.
- Now repeat the process for all vertices on level one from the root by examining each vertex in the designated order.Thus, since b and d are at level one,we first examine b.



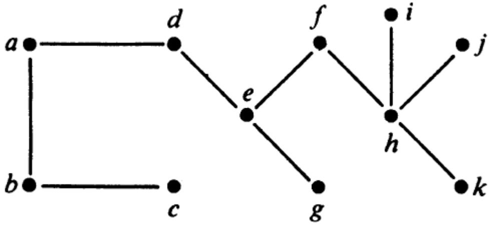
Breadth First Search

- For b, we include the edge {b,c} as an edge.Then for d, we reject the edge {d,c} since its inclusion would produce a cycle. But we include {d,e}.
- Next,we consider the vertices at level two. Reject the edge{c,e}; include {e,f} and {e,g}.



Breadth First Search

- Then repeat the procedure again for vertices on level three. Reject {f,g}, but include{f,h}. At g, reject {f,g} and {g,h}.
- On level four, include {h,i}, {h,j}, and {h,k}.
- Next, we attempt to apply the procedure on level five at i,j and k, but no edge can be added at these vertices so the procedure ends.
- The follwing graph is the final one and is said to be a spanning tree of the given graph.



Breadth First Search - Algorithm

1. Choose a starting node: Select a node from the graph to serve as the starting point for the BFS traversal.
2. Initialize data structures: Create a queue (FIFO) data structure to store the nodes that will be visited during the traversal. Also, create a set or array to keep track of the visited nodes to avoid revisiting them.
3. Enqueue the starting node: Add the starting node to the queue and mark it as visited.
4. While the queue is not empty, repeat steps 5-7
5. Dequeue a node: Remove a node from the front of the queue.
6. Visit the node: Process the node in any desired way. For example, you can print or store its value, or apply any other necessary operation.
7. Enqueue adjacent unvisited nodes: For each neighboring node of the current node that has not been visited, add it to the queue and mark it as visited.
8. Repeat until the queue is empty: Continue this process until the queue becomes empty, indicating that all reachable nodes have been visited.

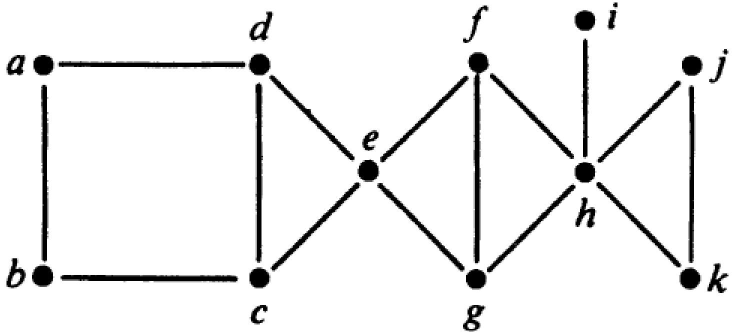
Breadth First Search - Code

```python
from collections import deque

def bfs(graph, start):
    visited = set()
    queue = deque([start])
    visited.add(start)

    while queue:
        vertex = queue.popleft()
        # Process the current node (you can modify this according to your needs)
        print(vertex, end=" ")

        for neighbor in graph[vertex]:
            if neighbor not in visited:
                queue.append(neighbor)
                visited.add(neighbor)
# Example usage
graph = {
    'A': ['B', 'C'],
    'B': ['A', 'D', 'E'],
    'C': ['A', 'F'],
    'D': ['B'],
    'E': ['B', 'F'],
    'F': ['C', 'E']
}

start_vertex = 'A'
print("BFS Traversal:")
bfs(graph, start_vertex)
```

```
BFS Traversal:
A B C D E F
```

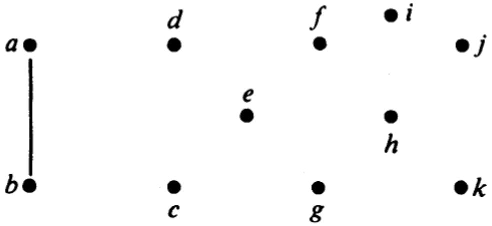
Depth First Search

Now let us illustrate Depth First Search (DFS) on the following graph.



Depth First Search

- We select the ordering of the vertices *abcdefghijk*
- Then select vertex a as the first vertex and designate it as root. The vertex a is said to be visited.
- Add all edges {a,x}, as x runs in order from b to k, that do not produce a cycle. Thus, we add {a,b}. The edge {a,b} is now said to be examined. Now a is the parent of b and b is the child of a.



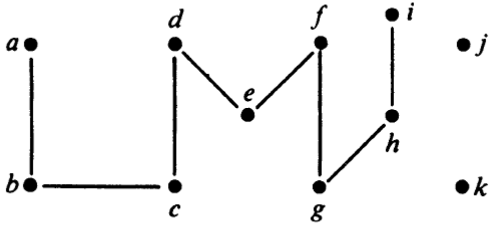
In general, while we are at some vertex x, two situations arise.

Depth First Search

- **Situation 1**: If there are some unexamined edges incident on x, then we consider the edge $\{x, y\}$, where y is the first vertex in the designated ordering on the vertices for which $\{x, y\}$ is unexamined. We have the following cases here.
  - **Case 1**: If y has not been previously visited, visit y, select $\{x, y\}$ as a tree edge, and continue the search from y. In this case, x is the parent of y.
  - **Case 2**:If y has been visited previously, then reject the edge $\{x, y\}$, consider it examined, and proceed to select another unexamined edge $\{x, z\}$ incident on x where z is the first vertex for which $\{x, z\}$ is an unexamined edge. Each such rejected edge in the context of depth-first search is called a back edge.
- **Situation 2**: If all the edges incident on x have already been examined, then we return to the parent of x and continue the search from the parent of x. The vertex x is now said to be completely scanned. Moreover, the process of returning to the parent of x is called backtracking.

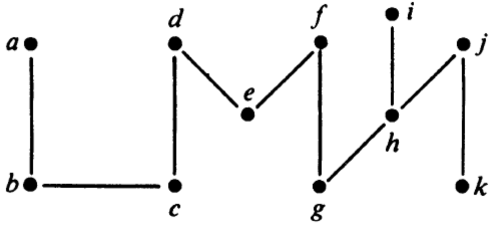
Depth First Search

- Now, in the present example, we would select the edge $\{b, c\}$, continue the search at c, and select $\{c, d\}$.
- Then we would continue the search at d and first reject the edge $\{d, a\}$ and then select the edge $\{d, e\}$.
- At e, we reject the edge $\{e, c\}$, select $\{e, f\}$.
- Now at f, we select $\{f, g\}$.
- Now at g, reject $\{g, e\}$ and select $\{g, h\}$.
- Now at h, reject $\{h, f\}$ and select $\{h, i\}$.



Depth First Search

- Now, there are no unexamined edges at i, we must backtrack to h and continue the search. [Situation 2]
- Now, at h, we select $\{h, j\}$ and $\{j, k\}$ and, finally, reject $\{k, h\}$.
- The following graph represents a spanning tree using Depth First Search Method for the given graph.



Depth First Search - Algorithm

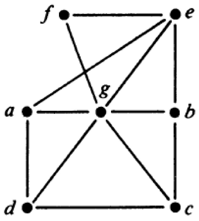
1. Choose a starting node: Select a node from the graph to serve as the starting point for the DFS traversal.
2. Initialize data structures: Create a stack (LIFO) data structure to store the nodes that will be visited during the traversal. Also, create a set or array to keep track of the visited nodes to avoid revisiting them.
3. Push the starting node onto the stack: Add the starting node to the stack and mark it as visited.
4. While the stack is not empty, repeat steps 5-7
5. Pop a node from the top of the stack: Remove a node from the top of the stack.
6. Visit the node: Process the node in any desired way. For example, you can print or store its value, perform calculations, or apply any other necessary operation.
7. Push adjacent unvisited nodes onto the stack: For each neighboring node of the current node that has not been visited, add it to the stack and mark it as visited.
8. Repeat until the stack is empty: Continue this process until the stack

Depth First Search - Code

```python
def dfs(graph, start):
    visited = set()
    stack = [start]

    while stack:
        vertex = stack.pop()
        if vertex not in visited:
            print(vertex, end=" ")
            visited.add(vertex)
            stack.extend(reversed(graph[vertex]))

# Example usage
graph = {
    'A': ['B', 'C'],
    'B': ['A', 'D', 'E'],
    'C': ['A', 'F'],
    'D': ['B'],
    'E': ['B', 'F'],
    'F': ['C', 'E']
}

start_vertex = 'A'
print("DFS Traversal:")
dfs(graph, start_vertex)
```

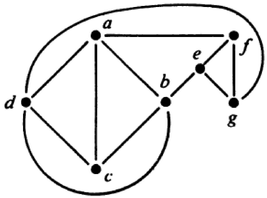
```
DFS Traversal:
A B D E F C
```

Examples



BFS : a - d - g - e - c - b - f

DFS : a - d - c - b - e - f - g



BFS : a - b - c - d - f - e - g

DFS : a - b - c - d - g - e - f

Applications of Graphs

- **Computer Science** : Here graphs are used to represent the flow of computation.
- **Google Maps** : Google maps uses graphs for building transportation systems, where intersection of two(or more) roads are considered to be a vertex and the road connecting two vertices is considered to be an edge, thus their navigation system is based on the algorithm to calculate the shortest path between two vertices.
- **Facebook** : In Facebook, users are considered to be the vertices and if they are friends then there is an edge running between them. Facebooks Friend suggestion algorithm uses graph theory. Facebook is an example of undirected graph. We can find similar things in social networks like LinkdIn, Twitter etc.
- **World Wide Web** : In World Wide Web, web pages are considered to be the vertices. There is an edge from a page u to other page v if there is a link of page v on page u. This is an example of Directed graph. It was the basic idea behind Google Page Ranking Algorithm.

Applications of Graphs

- **Transportation Networks**: Graphs are used to model transportation networks such as road networks, railway systems, and flight routes. Nodes represent cities or locations, and edges represent the connections between them. Graph algorithms help optimize routes, calculate shortest paths, and analyze traffic flow.
- **Biological Networks**: Graphs are used in biological research to model molecular interactions, protein-protein interaction networks, gene regulatory networks, and metabolic pathways. Graph algorithms help understand complex biological processes, identify key components, and predict functional relationships.
- **Knowledge Graphs**: Graphs are used to represent structured knowledge in knowledge graphs like Wikipedia and semantic networks. Concepts or entities are represented as nodes, and relationships between them are represented as edges. Graph algorithms help extract meaningful information, perform semantic search, and support question-answering systems.

Applications of Graphs

- **Computer Networks**: Graphs are used to model computer networks, communication networks, and data flow. Nodes represent devices or network components, and edges represent connections or communication links. Graph algorithms are applied to optimize network performance, analyze network traffic, and ensure efficient data transmission.
- **Recommendation Systems**: Graphs are used in recommendation systems to provide personalized recommendations to users. Users and items are represented as nodes, and the interactions or preferences between users and items are represented as edges. Graph-based algorithms are used to find similar users, identify related items, and make recommendations based on graph connections.

These are just a few examples of the wide range of applications of graphs. The flexibility and versatility of graph structures and algorithms make them applicable to various domains and problem-solving scenarios.

thank you