CSC6021 & AIR6001 Tutorial

- Brief Introduction of Graph Representation

Xudong Wang

SDS, CUHK(SZ)

September 12, 2023

香港中文大學(深圳)
The Chinese University of Hong Kong, Shenzhen

Graph (ADT)

Definition of a Graph (ADT)

A graph G is an **ordered pair** $G = (V, E)$ comprising a set V of **vertices** or nodes together with a set E of **edges** or branches. Each edge has either one or two vertices associated with it, called its "**endpoints**".

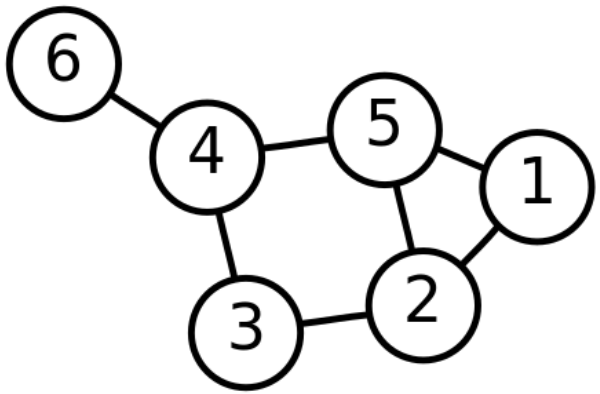


Figure 1: A graph with six vertices and seven edges

In discrete mathematics, and more specifically in graph theory, a graph is a structure amounting to a set of objects in which some pairs of the objects are in some sense "related".

Graph (ADT) Type

Single Graph vs. Multigraph

- **Single Graph**: A graph in which each edge connects two different vertices and where **no two edges connect the same pair of vertices**.
- **Multigraph Graph**: Multigraphs are graphs that **allow multiple edges between any pair of vertices**, either directed or undirected. Multigraphs can be simple (no self-loops) or may allow self-loops.

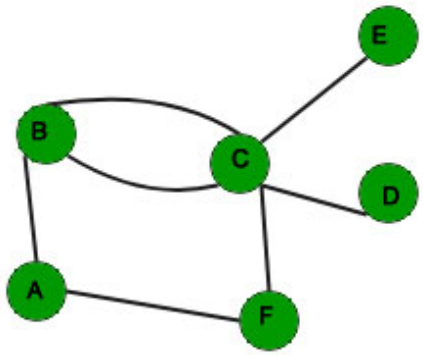


Figure 2: A graph with six vertices and seven edges.The above graph is a multigraph since there are multiple edges between Node B and C. The multiplicity of the edge $\{B, C\}$ is 2.

Except for special declarations and simplifications, subsequent **Graphs** in this tutorial refer to **Single Graph**.

Graph (ADT) Type

Directed vs. Undirected Graphs

- **Directed Graph**: Edges have a direction, represented as (u, v) means the edge from node $u \rightarrow v$..
- **Undirected Graph**: Edges have no direction, usually, represented as $\{u, v\}$.

Weighted vs. Unweighted Graphs

- **Unweighted Graph**: E contains no additional information.
- **Weighted Graph**: Edges in E are associated with weights, often represented as $E = \{(u, v, w)\}$ where w is the weight.

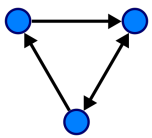


Figure 3: A directed graph

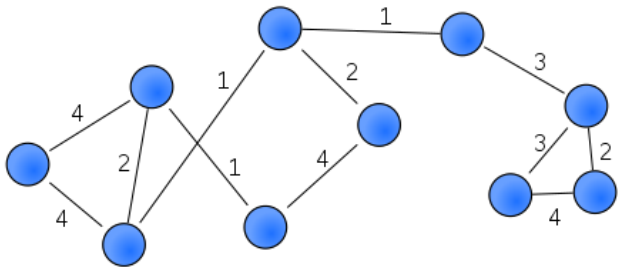


Figure 4: A weighted graph with ten vertices and twelve edges

Basic Graph Terminology

Adjacency

Adjacency: In a graph G two vertices u and v are said to be adjacent if they are the **endpoints** of an edge.
The edge $e : \{u, v\}$ is said to be **incident** with the vertices.
In case the edge is directed, u is said to be adjacent to v and v is said to be adjacent from u . Here, u is the **initial vertex** and v is the **terminal vertex**.

Degree

Degree: The degree of a vertex is **the number of edges incident with it**. (except the self-loop, which contributes twice to the vertex degree.)
Degree of a vertex u is denoted as $deg(u)$.
In case of **directed graphs**, the degree is further classified as **in-degree and out-degree**.

- The **in-degree** $deg^-(u)$ of a vertex is the number of edges with the given vertex as the terminal vertex.
- The **out-degree** $deg^+(u)$ of a vertex is the number of edges with the given vertex as the initial vertex.

Adjacency List

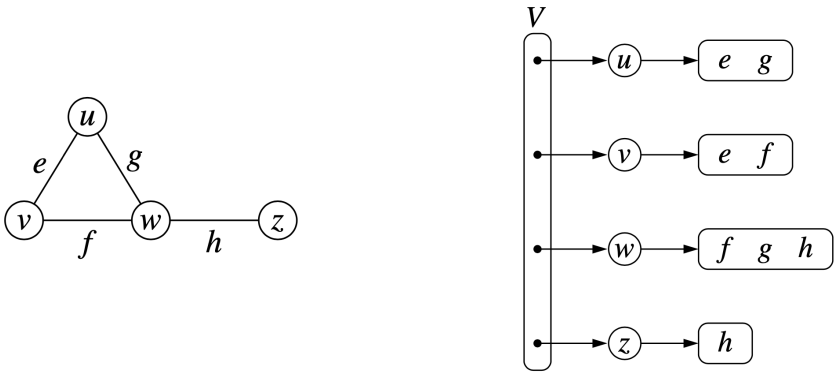


Figure 5: **Left:** An undirected graph G; **Right:** a schematic representation of the adjacency list structure for G. Collection V is the primary list of vertices, and each vertex has an associated list of incident edges. Although not diagrammed as such, we presume that each edge of the graph is represented with a unique Edge instance that maintains references to its endpoint vertices.

- **Adjacency List:** Array of lists, one list per vertex, where the list at index i contains vertex i's adjacent vertices. e.g: $0 \to [1], 1 \to [0, 2], 2 \to [1]$
- **Directed:** Single entry for each edge
  Undirected: Double entry for each edge
- Space Complexity: $O(|V| + |E|)$
- Time Complexity for edge insertion/deletion: $O(1)$,
  Query edge (u, v) existence $O(min(deg(u), deg(v)))$

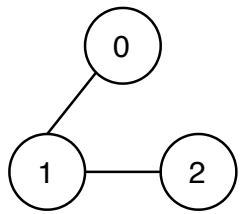
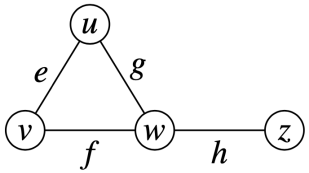


Figure 6: Example

Adjacency Matrix



	0	1	2	3
u → 0		e	g	
v → 1	e		f	
w → 2	g	f		h
z → 3			h	

Figure 7: **Left:** An undirected graph G; **Right:** a schematic representation of the auxiliary adjacency matrix structure for G, in which n vertices are mapped to indices 0 to $n-1$.

- **Adjacency Matrix:** Square matrix (2D array) A where A_{ij} is 1 if there's an edge from i to j, otherwise 0.
- **Directed:** Asymmetrical matrix
  Undirected: Symmetrical matrix.
- Space Complexity: $O(|V|^2)$
- Time Complexity for edge insertion/deletion: $O(1)$
  Query edge existence: $O(1)$

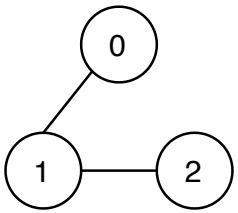


Figure 8: Example

$$\begin{pmatrix} 0 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 0 \end{pmatrix}$$

Weight Matrix

- **Definition:** Similar to adjacency matrix, but contains weights instead of Booleans.
- **Directed:** Asymmetrical matrix
  Undirected: Symmetrical matrix.
- Time Complexity: Same as adjacency matrix.

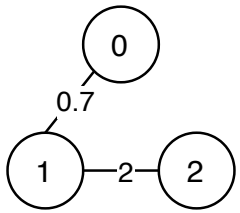


Figure 9: Example of a weight graph

$$\begin{pmatrix} 0 & 0.7 & 0 \\ 0.7 & 0 & 2 \\ 0 & 2 & 0 \end{pmatrix}$$

Degree Matrix and Handshaking Theorem

Degree Matrix

Diagonal matrix D where $D(i, i)$ contains the degree $deg(i)$ of vertex i.
For Directed Graphs, In-Degree and Out-Degree matrices. Calculation: $O(|V|)$ for adjacency list, $O(|V|^2)$ for adjacency matrix.

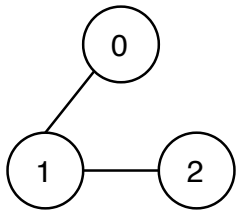


Figure 10: Example

$$\begin{pmatrix} 1 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

Handshaking Theorem

For a graph, the sum of all vertex degrees is equal to twice the number of edges.

$$\text{Undirected: } \sum_{v \in V} \deg(v) = 2|E|$$

$$\text{Directed: } \sum_{v \in V} (\deg^+(v) + \deg^-(v)) = 2|E|$$

Incidence Matrix

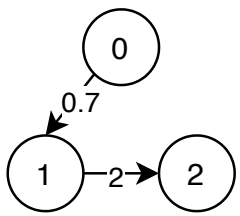
Definition

A $V \times E$ matrix B where $B_{ij} = 1$ if vertex i and edge j are incident, otherwise 0.
For directed graphs:

$$[\mathbf{B}_{ij}]_k = \begin{cases} 1, & k = i \\ -1, & k = j \\ 0, & \text{otherwise} \end{cases}.$$

- - Space: $O(|V| \times |E|)$
  - Query an edge (u, v) existence: $O(|E|)$, Edge Operations: $O(|V|)$
- Consumes more space, but useful in specific scenarios, e.g., multigraphs, calculate Laplacian matrix for weighted directed graphs: $L = BW_DB^T$, where the W_D in $E \times E$ is the diagonal weight matrix.

$$B = \begin{pmatrix} 1 & 0 \\ -1 & 1 \\ 0 & -1 \end{pmatrix}, W_D = \begin{pmatrix} 0.7 & 0 \\ 0 & 2 \end{pmatrix}, L = \begin{pmatrix} 0.7 & -0.7 & 0 \\ -0.7 & 2.7 & -2 \\ 0 & -2 & 2 \end{pmatrix}$$



Summary and Comparison of 4 Common Graph Representation

Operation	Edge List	Adj. List	Adj. Map	Adj. Matrix
vertex_count( )	$O(1)$	$O(1)$	$O(1)$	$O(1)$
edge_count( )	$O(1)$	$O(1)$	$O(1)$	$O(1)$
vertices( )	$O(n)$	$O(n)$	$O(n)$	$O(n)$
edges( )	$O(m)$	$O(m)$	$O(m)$	$O(m)$
get_edge(u,v)	$O(m)$	$O(\min(d_u, d_v))$	$O(1)$ exp.	$O(1)$
degree(v)	$O(m)$	$O(1)$	$O(1)$	$O(n)$
incident_edges(v)	$O(m)$	$O(d_v)$	$O(d_v)$	$O(n)$
insert_vertex(x)	$O(1)$	$O(1)$	$O(1)$	$O(n^2)$
remove_vertex(v)	$O(m)$	$O(d_v)$	$O(d_v)$	$O(n^2)$
insert_edge(u,v,x)	$O(1)$	$O(1)$	$O(1)$ exp.	$O(1)$
remove_edge(e)	$O(1)$	$O(1)$	$O(1)$ exp.	$O(1)$

Figure 11: A summary of the running times for the methods of the graph ADT. Here m denote the number of vertices, n the number of edges, and dv the degree of vertex v. Note that the adjacency matrix uses $O(m^2)$ space, while all other structures use $O(m + n)$ space.

Adjacency List: Space-efficient for sparse graphs, Slower lookups
Adjacency Matrix: Fast lookups, Space inefficient, Good for small (V<1000), dense graphs.

Other Graph Representations

Edge List

An edge list is one of the simplest graph representations, listing all the edges of a graph as vertex pairs (u, v) (and possibly the weight w, if it's a weighted graph).

Complexity

- Space: $O(|E|)$
- Add/Remove/Check Edge: $O(1)$ for adding, $O(|E|)$ for removing or checking

Applications

- Useful for graphs that need to be serialized/deserialized.

Adjacency Map

An adjacency map is similar to an adjacency list but uses a hash map to store adjacent vertices, allowing for quick edge weight lookups.

Complexity

- Space: $O(|V| + |E|)$
- Add/Remove/Check Edge: $O(1)$

Applications

- Useful for graphs with complex edge attributes.

Other Graph Representations

Compressed Sparse Row (CSR)

Compressed Sparse Row is mainly used for representing sparse matrices but can also be used for sparse graphs. It compresses the adjacency list to save space.

Complexity

- Space: $O(|V| + |E|)$

Applications

- Useful in machine learning, operations research, and scientific computing.

Hypergraph

A hypergraph generalizes a graph by allowing edges to connect any number of vertices, not just two. In a hypergraph, each hyperedge can connect a set of vertices, not just a pair.

Complexity

- Depends on the specific representation (could be edge list or incidence matrix).

Applications

- Circuit design, finite element methods, and social network analysis.

Other Graph Representations

Bipartite Graph Representation

In a bipartite graph, **vertices can be divided into two disjoint sets** such that all edges go between the sets and not within a set. This property can be exploited for a more efficient representation.

Complexity

- Space: $O(|V| + |E|)$

Applications

- Matching problems, flow networks.

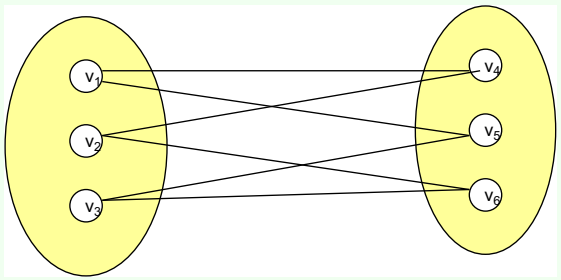


Figure 12: Bipartite graphs

Dynamic Graph Data Structures

These are specialized data structures like dynamic trees, Euler tour trees, or link/cut trees, useful for handling graph updates efficiently.

Applications

- Dynamic connectivity, minimum spanning tree maintenance(MST).

Thank you for listening!

香港中文大學(深圳)
The Chinese University of Hong Kong, Shenzhen

Xudong Wang
xudongwang@link.cuhk.edu.cn
https://xd-w.github.io

September 12, 2023