DATA STRUCTURE

Chapter 7: Graphs

Study Notes for Engineering Students

7.1 Introduction to Graphs

A **graph** is a non-linear data structure used to represent relationships between objects. Unlike arrays, stacks and queues (linear) or trees (hierarchical), a graph allows any element to be connected to any other element. It consists of a set of **vertices** (also called nodes) and a set of **edges** (also called arcs) that join pairs of vertices.

Formally, a graph is written as **G = (V, E)**, where V is a finite non-empty set of vertices and E is a set of edges. Each edge is a pair of vertices (u, v). If the pair is unordered, the edge is undirected; if the pair is ordered, the edge is directed.

Why graphs?

- They model real-world networks such as roads, computer networks, social connections and web pages.
- A tree is a special graph that is connected and has no cycles, so every tree is a graph but not every graph is a tree.
- Many practical problems (routing, scheduling, dependency resolution) reduce to graph problems.

Main types of graphs

- **Undirected graph:** edges have no direction; edge (u, v) is the same as (v, u).
- **Directed graph (digraph):** every edge has a direction, written u → v.
- **Weighted graph:** each edge carries a numeric value (cost, distance, time).
- **Unweighted graph:** edges carry no value; all edges are treated equally.

7.2 Terms Associated with Graphs

The following terms are used constantly while studying graph algorithms. Learn them well, because every later topic (representation, traversal, spanning trees, shortest path) is described using them.

- **Self-loop:** an edge that starts and ends at the same vertex.
- **Parallel (multiple) edges:** two or more edges joining the same pair of vertices.
- **Simple graph:** a graph with no self-loops and no parallel edges.
- **Multigraph:** a graph that allows parallel edges.
- **Degree of a vertex:** number of edges attached to it (a self-loop counts twice).
- **Isolated vertex:** a vertex of degree 0. **Pendant vertex:** a vertex of degree 1.
- **Regular graph:** every vertex has the same degree.
- **Complete graph (Kn):** every pair of distinct vertices is joined by an edge. It has n(n − 1)/2 edges.

- **Subgraph:** a graph whose vertices and edges are subsets of another graph.
- **Connected graph:** there is a path between every pair of vertices. Otherwise it is disconnected.
- **Cycle:** a closed path that starts and ends at the same vertex without repeating any other vertex or edge.
- **Acyclic graph:** a graph with no cycles. A directed acyclic graph is called a **DAG**.
- **Bipartite graph:** vertices can be split into two sets such that every edge joins a vertex of one set to a vertex of the other.
- **Forest:** a collection of disjoint trees (an acyclic graph that need not be connected).

Handshaking property: in any undirected graph, the sum of the degrees of all vertices equals twice the number of edges, i.e. $\Sigma \deg(v) = 2|E|$.

7.3 Terminology

This section defines the core vocabulary of graph theory: graph, node (vertex), arc (edge), directed graph, in-degree, out-degree, adjacent, successor, predecessor, relation, weight, path and length.

- **Graph:** a pair G = (V, E) of a vertex set V and an edge set E.
- **Node (vertex):** a fundamental unit of the graph that stores data, drawn as a circle or point.
- **Arc (edge):** a connection between two nodes. In a directed graph an edge is called an arc and is drawn as an arrow from the tail (start) to the head (end).
- **Directed graph:** a graph in which every edge is an ordered pair (u, v), meaning the connection goes from u to v only.
- **In-degree of a vertex v:** the number of edges that end at (enter) v.
- **Out-degree of a vertex v:** the number of edges that start at (leave) v.
- **Adjacent vertices:** two vertices joined directly by an edge. In a digraph, v is adjacent to u if there is an edge $u \rightarrow v$.
- **Successor:** if there is an edge $u \rightarrow v$, then v is a successor of u.
- **Predecessor:** if there is an edge $u \rightarrow v$, then u is a predecessor of v.
- **Relation:** a graph represents a relation on the vertex set. The edge set E is a subset of $V \times V$, so an edge (u, v) means that u is related to v.
- **Weight:** a number attached to an edge that represents cost, distance, time or capacity.
- **Path:** a sequence of vertices v1, v2, …, vk such that each consecutive pair (vi, vi+1) is an edge.
- **Length of a path:** the number of edges in the path. In a weighted graph, it is the sum of the weights of the edges on the path.

Example of a directed graph

Consider a directed graph D with vertices A, B, C, D and arcs A→B, A→C, B→D, C→D.

Vertex	In-degree	Out-degree	Successors	Predecessors
A	0	2	B, C	—
B	1	1	D	A
C	1	1	D	A
D	2	0	—	B, C

- The path A → B → D has length 2 (two edges).
- The sum of all in-degrees = the sum of all out-degrees = number of arcs (here 4).
- The graph has no cycle, so it is a DAG.

7.4 Sequential Representation of Graphs

In sequential representation a graph is stored in an array. The most common method is the **adjacency matrix**, a two-dimensional array A of size n × n, where n is the number of vertices. The rows and columns correspond to vertices, and A[i][j] tells whether an edge exists from vertex i to vertex j.

Rules

- Unweighted graph: A[i][j] = 1 if an edge from i to j exists, otherwise 0.
- Weighted graph: A[i][j] = weight of the edge, and 0 or ∞ when no edge exists.
- Undirected graph: the matrix is **symmetric** (A[i][j] = A[j][i]).
- Directed graph: the matrix need not be symmetric; row i shows edges leaving i and column j shows edges entering j.
- Row sum = out-degree and column sum = in-degree (for a digraph). For an undirected graph, row sum = degree.

Example 1: Undirected graph G1

Let G1 have vertices A, B, C, D, E, F and edges A–B, A–C, B–D, C–D, C–E, D–F, E–F. This graph is used again in the traversal section.

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

Example 2: Directed graph D (from 7.3)

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

Advantages and disadvantages

- **Advantage:** checking whether an edge (i, j) exists takes constant time, O(1).
- **Advantage:** simple to implement and suits dense graphs.
- **Disadvantage:** needs $O(V^2)$ memory even if the graph has very few edges.
- **Disadvantage:** finding all neighbours of a vertex takes O(V) time, and adding or removing a vertex is costly.

Other sequential forms: an **incidence matrix** (vertices × edges) can also be used. For reachability, a **path matrix** can be computed from the adjacency matrix using Warshall's algorithm.

7.5 Linked Representation of Graphs

In linked representation each vertex has a linked list containing all the vertices adjacent to it. This structure is called an **adjacency list**. An array of head pointers is kept, one per vertex, and each list node stores the adjacent vertex (and the weight for a weighted graph) together with a pointer to the next node.

Node structure in C

```
struct node {
    int vertex;          /* adjacent vertex        */
    int weight;          /* used in weighted graph */
    struct node *next;   /* link to next neighbour */
};
struct node *adj[MAX];   /* adj[i] = head of list of vertex i */
```

Adjacency list of graph G1

Vertex	Adjacency list
A	B → C → NULL
B	A → D → NULL
C	A → D → E → NULL
D	B → C → F → NULL
E	C → F → NULL
F	D → E → NULL

Key points

- Space required is O(V + E), which is far less than a matrix for sparse graphs.

- In an undirected graph every edge appears twice (once in each end vertex's list); in a directed graph it appears once.
- Listing all neighbours of a vertex is fast, but checking whether a particular edge exists takes O(degree) time.
- An **adjacency multilist** is a variation in which each edge is stored once and shared between the two lists.

Adjacency matrix vs adjacency list

Point	Adjacency matrix	Adjacency list
Space	$O(V^2)$	O(V + E)
Check edge (u, v)	O(1)	O(degree of u)
List neighbours	O(V)	O(degree of u)
Best for	Dense graphs	Sparse graphs
Structure	Static 2-D array	Dynamic linked lists

7.6 Traversal of Graphs

Graph traversal means visiting every vertex of a graph exactly once in a systematic way. Because a graph may contain cycles, a **visited** array must be maintained so that no vertex is processed twice. The two standard techniques are Breadth First Search (BFS) and Depth First Search (DFS). All examples below use graph G1 (see 7.4) starting from vertex A, choosing neighbours in alphabetical order.

Breadth First Search (BFS)

BFS visits the start vertex, then all its neighbours, then all neighbours of those neighbours, level by level. It uses a **queue** (FIFO).

1. Mark the start vertex visited and insert it into the queue.
2. Remove a vertex v from the front of the queue and process it.
3. Insert every unvisited neighbour of v into the queue and mark it visited.
4. Repeat steps 2 and 3 until the queue becomes empty.

```
BFS(G, s):
    visited[s] = true; enqueue(Q, s)
    while Q is not empty:
        v = dequeue(Q); visit(v)
        for each w adjacent to v:
            if visited[w] == false:
                visited[w] = true; enqueue(Q, w)
```

BFS order for G1: A, B, C, D, E, F.

Depth First Search (DFS)

DFS goes as deep as possible along one path before backtracking. It uses a **stack** (explicitly, or implicitly through recursion).

1. Visit the start vertex and mark it visited.

2. Move to an unvisited neighbour of the current vertex and repeat.
3. If no unvisited neighbour remains, backtrack to the previous vertex.
4. Stop when the stack is empty (all reachable vertices are visited).

```
DFS(v):
    visited[v] = true; visit(v)
    for each w adjacent to v:
        if visited[w] == false:
            DFS(w)
```

DFS order for G1: A, B, D, C, E, F.

Comparison of BFS and DFS

Point	BFS	DFS
Data structure	Queue	Stack / recursion
Approach	Level by level	Deep first, then backtrack
Time (list)	O(V + E)	O(V + E)
Time (matrix)	$O(V^2)$	$O(V^2)$
Shortest path (unweighted)	Yes	No
Typical uses	Shortest hops, level order, peer discovery	Cycle detection, topological sort, connected components

7.7 Spanning Trees

A **spanning tree** of a connected graph G is a subgraph that includes all the vertices of G, is connected, and contains no cycles. If G has n vertices, every spanning tree has exactly **n − 1 edges**. A graph can have many spanning trees; a complete graph Kn has n^(n−2) of them (Cayley's formula).

Properties

- Adding any extra edge to a spanning tree creates exactly one cycle.
- Removing any edge from a spanning tree disconnects it.
- BFS and DFS traversals of a connected graph each produce a spanning tree (BFS tree and DFS tree).
- A disconnected graph has no spanning tree; it has a spanning forest.

Minimum Spanning Tree (MST)

In a weighted connected graph, the spanning tree whose total edge weight is the smallest is the **minimum spanning tree**. MSTs are used in laying cables, pipelines and road networks at minimum cost. Two classic algorithms are Kruskal's and Prim's.

Weighted graph W used in the examples: vertices A, B, C, D with edges A–B = 1, A–C = 4, B–C = 2, B–D = 7, C–D = 3.

Kruskal's algorithm (edge based, greedy)

1. Sort all edges in increasing order of weight.

2. Pick the smallest edge. If it does not form a cycle with the edges already chosen, include it; otherwise discard it.
3. Repeat until n − 1 edges have been selected.

On W: sorted edges AB(1), BC(2), CD(3), AC(4), BD(7). Select AB, BC, CD. AC would form a cycle, so it is rejected. **MST weight = 1 + 2 + 3 = 6.** Time complexity is O(E log E).

Prim's algorithm (vertex based, greedy)

1. Start with any vertex and add it to the tree.
2. Among all edges that connect a tree vertex to a non-tree vertex, choose the one with minimum weight.
3. Add that edge and its new vertex to the tree.
4. Repeat until all vertices are included.

On W (start at A): choose AB(1), then BC(2), then CD(3). The MST weight is again 6. Time complexity is $O(V^2)$ with a matrix, or O(E log V) with a heap.

7.8 Shortest Path

The **shortest path problem** asks for a path between two vertices whose total weight (length) is minimum. In an unweighted graph BFS already gives the shortest path in terms of number of edges. For weighted graphs the following algorithms are used.

- **Dijkstra's algorithm:** single-source shortest paths; works only when all weights are non-negative.
- **Bellman–Ford algorithm:** single-source shortest paths; also handles negative weights and can detect negative cycles. Time O(VE).
- **Floyd–Warshall algorithm:** shortest paths between all pairs of vertices; time $O(V^3)$.

Dijkstra's algorithm

1. Set the distance of the source to 0 and of every other vertex to ∞. Mark all vertices unvisited.
2. Select the unvisited vertex u with the smallest distance and mark it visited.
3. For every neighbour v of u, relax the edge: if dist[u] + w(u, v) < dist[v], update dist[v].
4. Repeat steps 2 and 3 until all vertices are visited (or the destination is reached).

Example on graph W with source A

Step	dist[A]	dist[B]	dist[C]	dist[D]
Initial	0	∞	∞	∞
Select A	0	1	4	∞
Select B	0	1	3	8
Select C	0	1	3	6
Select D	0	1	3	6

The shortest distances from A are B = 1, C = 3 (via B) and D = 6 (via B and C, path A → B → C → D). Dijkstra's time complexity is $O(V^2)$ with an array and $O((V + E) \log V)$ with a priority queue.

Remember: Dijkstra fails with negative edge weights because a vertex once finalised is never reconsidered.

7.9 Application of Graph

Graphs are among the most widely used data structures because so many real systems are naturally networks of connected entities. Important applications include:

- **Social networks:** people are vertices and friendships or follows are edges; used for friend suggestions and community detection.
- **Maps and navigation:** places are vertices and roads are weighted edges; shortest-path algorithms give routes in GPS systems.
- **Computer networks and routing:** routers are vertices and links are edges; protocols such as OSPF use Dijkstra's algorithm.
- **World Wide Web:** web pages are vertices and hyperlinks are directed edges; search engines rank pages using link graphs (PageRank).
- **Task scheduling and dependencies:** a DAG with topological sorting is used for build systems, course prerequisites and project planning (PERT/CPM).
- **Operating systems:** resource allocation graphs are used to detect deadlock.
- **Compilers:** control-flow graphs and dependency graphs are used in optimisation and register allocation.
- **Network design:** minimum spanning trees reduce the cost of cabling, pipelines and electrical grids.
- **Electrical circuits and VLSI:** components and connections are modelled as graphs.
- **Chemistry and biology:** molecules (atoms and bonds) and protein or gene interaction networks are stored as graphs.
- **Recommendation engines:** users and items form a bipartite graph for suggesting products.

Multiple Choice Questions (15)

Choose the correct option for each question. The answer key is given after the questions.

Q1. A graph is represented as G = (V, E). Here E denotes the:

(a) Set of vertices
(b) Set of edges
(c) Set of paths
(d) Set of cycles

Q2. In an undirected graph with e edges, the sum of the degrees of all vertices is:

(a) e
(b) e / 2
(c) 2e
(d) e^2

Q3. In a directed graph with 8 edges, the sum of all in-degrees is:

(a) 4
(b) 16
(c) 0
(d) 8

Q4. The maximum number of edges in a simple undirected graph with n vertices is:

(a) $n(n - 1)/2$
(b) $n(n + 1)/2$
(c) n^2
(d) $n - 1$

Q5. A closed path that starts and ends at the same vertex without repeating other vertices is a:

(a) Tree
(b) Cycle
(c) Spanning tree
(d) Forest

Q6. The adjacency matrix of an undirected graph is always:

(a) Diagonal
(b) Upper triangular
(c) Symmetric
(d) Identity

Q7. The space required by an adjacency matrix for a graph with V vertices is:

(a) O(V)
(b) O(V + E)
(c) O(E)
(d) $O(V^2)$

Q8. The space required by an adjacency list is:

(a) O(V + E)
(b) O(V^2)
(c) O(E^2)
(d) O(V log V)

Q9. Which data structure is used in Breadth First Search?

(a) Stack
(b) Queue
(c) Priority queue
(d) Array of pointers only

Q10. Which data structure is used (explicitly or by recursion) in Depth First Search?

(a) Queue
(b) Heap
(c) Stack
(d) Linked list only

Q11. A spanning tree of a connected graph with n vertices has exactly how many edges?

(a) n
(b) n + 1
(c) n / 2
(d) n − 1

Q12. Which algorithm builds an MST by repeatedly choosing the smallest edge that does not form a cycle?

(a) Kruskal's algorithm
(b) Dijkstra's algorithm
(c) Floyd–Warshall algorithm
(d) Bellman–Ford algorithm

Q13. Dijkstra's algorithm may fail if the graph contains:

(a) Cycles
(b) Parallel edges
(c) Negative edge weights
(d) Isolated vertices

Q14. The time complexity of the Floyd–Warshall all-pairs shortest path algorithm is:

(a) O(V^2)
(b) O(V^3)
(c) O(VE)
(d) O(E log V)

Q15. Topological sorting is possible only for a:

(a) Directed acyclic graph
(b) Undirected graph
(c) Complete graph
(d) Graph with a cycle

Answer Key

1 – (b)	2 – (c)	3 – (d)	4 – (a)	5 – (b)
6 – (c)	7 – (d)	8 – (a)	9 – (b)	10 – (c)
11 – (d)	12 – (a)	13 – (c)	14 – (b)	15 – (a)

Broad Questions (10)

Q1. Define a graph. Explain the different types of graphs (undirected, directed, weighted, complete, connected) with suitable examples.

Q2. Explain the following terms with a neat example of a directed graph: node, arc, in-degree, out-degree, adjacent vertices, successor, predecessor, path and length of a path.

Q3. Explain the sequential (adjacency matrix) representation of graphs. Draw the adjacency matrices of a directed graph and an undirected graph, and state the advantages and disadvantages.

Q4. Explain the linked representation (adjacency list) of a graph with the node structure. Compare it with the adjacency matrix representation.

Q5. Write the Breadth First Search algorithm. Trace it on a suitable graph and state its time complexity.

Q6. Write the Depth First Search algorithm. Trace it on a suitable graph and differentiate between BFS and DFS.

Q7. What is a spanning tree? Explain the minimum spanning tree and describe Kruskal's algorithm with an example.

Q8. Explain Prim's algorithm for finding a minimum spanning tree with an example. State its time complexity.

Q9. Explain Dijkstra's shortest path algorithm with a suitable weighted graph. Why does it fail for negative edge weights?

Q10. What are the applications of graphs? Explain any five applications in detail.