1 Introduction to Graphs

A graph G is a pair (V, E), where V is a set of nodes, called vertices and E is a collection of pairs of vertices, called edges.

Edge Types
A graph can be directed or undirected. In a directed graph, the edges are ordered pairs of vertices (u, v), where u is the origin/tail and v is the destination/head. In an undirected graph, the edges are unordered pairs of vertices (u,v), whereby u and v can access each other either way.

Simple paths
A path is a sequence/set of vertices such that every pair of consecutive vertices is connected by an edge. A simple path is a path with no repeated vertices. As such, all vertices in that path are distinct.

Cycles in graphs
A cycle is a path with at least one edge, whose first and last vertices are the same. A simple cycle is a cycle. A simple cycle is one where all vertices are distinct, except for the first and last vertices. An acyclic graph has no cycle in it, which is impossible for undirected graphs. A directed graph can be acyclic.

Subgraphs
A subgraph S of a graph G is defined as S = (U, F) where U is a subset of vertices in V and F is a subset of edges in E. An induced subgraph of G is a subgraph that is formed by selecting a subset U of vertices and all the edges in G that have both endpoints in U, which is denoted as G[U]. Similarly, an induced subgraph can be formed by selecting a subset F of edges and the vertices that are endpoints of the edges in F, which is denoted as G[F].

Connectivity
A graph G = (V, E) is considered connected if there is a path between every pair of vertices in V. In other words, every vertex in the graph is reachable from every other vertex. A connected component of a graph G is a maximally connected subgraph of G, meaning that it is a subgraph that is connected, and there is no larger connected subgraph that contains it. A graph is considered disconnected if it has two or more connected components, which means that there are pairs of vertices that are not connected by any path in the graph.

Trees and Forests
An unrooted tree is a graph that is connected and has no cycles. A forest is a graph without cycles, and its connected components are trees. Every tree on n vertices has exactly n-1 edges, which is a well-known property of trees in graph theory. A rooted tree is a type of tree that is produced by a directed graph where the edges are directed away from the root.

1.1 Graph Properties

- $\sum_{v \in V}$ deg(v) = 2m, where m is the number of edges in the graph. This states that the sum of the degrees of all the vertices in the graph is equal to twice the number of edges in the graph.
- In a simple undirected graph m <= n (n - 1)/2.
- n represents the number of vertices, m represents the number of edges. And $\triangle$ represents the maximum degree.

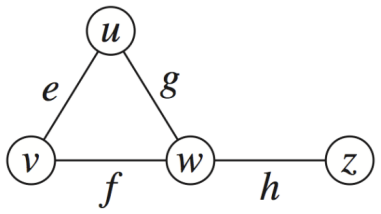
2 Graph ADT

We model the abstraction as a combination of three data types: Vertex, Edge, and Graph. A Vertex stores an associated object (e.g., an airport code) that is retrieved with a getElement() method. An Edge stores an associated object (e.g., a flight number, travel distance) that is retrieved with a getElement() method.

2.1 Edge List Structure

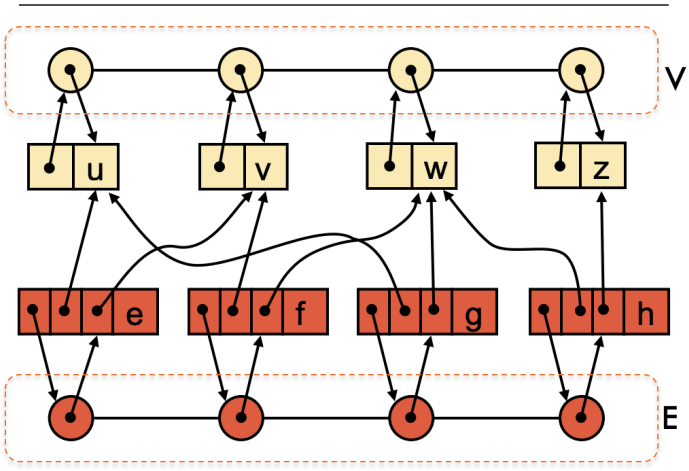
Vertex sequence holds

- sequence of vertices
- vertex object keeps track of its position in the sequence



Edge sequence

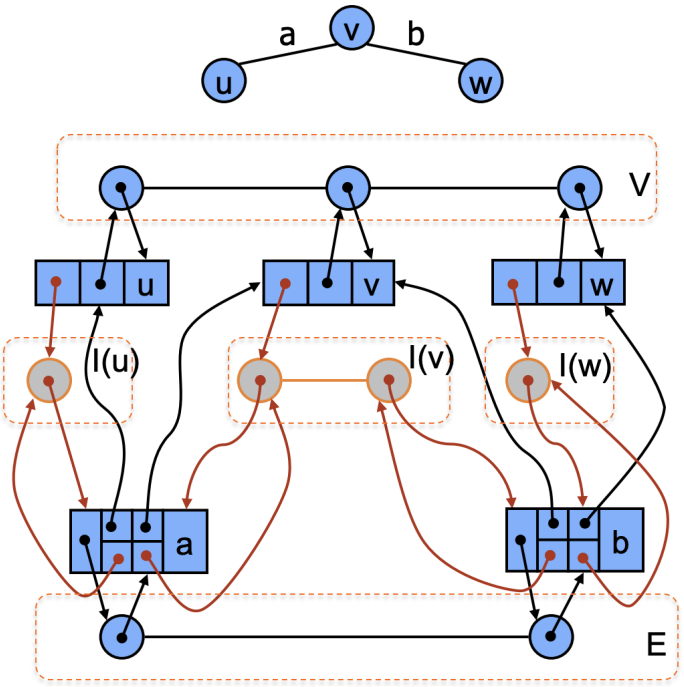
- sequence edges
- edge object keeps track of its position in the sequence
- Edge object points to the two vertices it connects



2.2 Adjacency List

Additionally each vertex keeps a sequence of edges incident on it

Edge objects keep reference to their position in the incidence sequence of its endpoints

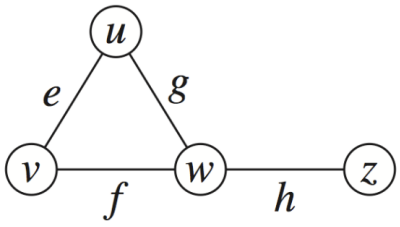


2.3 Adjacency Matrix Structure

Vertex array induces an index from 0 to n-1 for each vertex

2D-array adjacency matrix

- Reference to edge object for adjacent vertices
- Null for nonadjacent vertices



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

2.4 Asymptotic performance

No parallel edges/s-loops	Edge List	Adjacency List	Adjacency Matrix
Space	O(n + m)	O(n + m)	O(n^2)
incidentEdges(v)	O(m)	O(deg(v))	O(n)
getEdge(u, v)	O(m)	O(min(deg(u), deg(v)))	O(1)
insertVertex(x)	O(1)	O(1)	O(n^2)
insertEdge(u, v, x)	O(1)	O(1)	O(1)
removeVertex(v)	O(m)	O(deg(v))	O(n^2)
removeEdge(e)	O(1)	O(1)	O(1)

3 Depth-First Search (DFS)

This strategy tries to follow outgoing edges leading to yet unvisited vertices whenever possible, and backtrack if "stuck." If an edge is used to discover a new vertex, we call it a DFS edge, otherwise we call it a back edge. The DFS edges form a spanning tree of the connected component being explored.

```
1. def DFS(G):                                  # Total run time is O(n + m)
2.     for u in G.vertices() do                 # Set things up for DFS: O(n)
3.         visited[u] ← false
4.         parent[u] ← null
5.     for u in G.vertices() do                 # Visit all vertices: O(n) not counting DFS_Visit
6.         if not visited[u] then
7.             DFS_Visit(u)
8.     return parent
```

```
1. def DFS_Visit(u):                            # O(deg(u)). Total runtime: O(m)
2.     visited[u] ← true
3.     for v in G.incidentEdges(u) do           # Check out all neighbors of u
4.         if not visited[v] then
5.             parent[v] ← u
6.             DFS_Visit(v)
```

Let Cv be a connected component of v in our graph G. The DFS_Visit visits all vertices in Cv before returning to the outer loop. Edges {(u, parent[u]): u in Cv} form a spanning tree of C. Edges {(u, parent[u]): u in V} form a spanning forest of G.

DFS runs in O(n + m). It can also solve other graph problems in this time. For example, it can find a path between two given vertices, if any, find a cycle in the graph, test whether a graph is connected, compute connected components of a graph and compute a spanning forest of a graph.

3.1 Cut edges

In a connected graph G=(V, E), we say that an edge (u, v) in E is a cut edge if (V, E / (u, v)) is not connected. The cut edge problem is to identify all cut edges. We can solve this problem by running DFS on G. Trivial O(m2) time algorithm: For each edge (u,v) in E, remove (u,v) and check using DFS if G is still connected, put back (u,v). Better O(nm) time algorithm: Only test edges in a DFS tree of G. Way to do it in O(n + m) but we don't know how to do it yet.

4 Breadth-First Search (BFS)

This strategy tries to visit all vertices at distance k from a start vertex s before visiting vertices at distance k + 1:

- L_0 = s
- L_1 = vertices one hop away from s
- L_2 = vertices two hops away from s but no closer
- k = vertices k hops away from s but no closer

```
1. def BFS(G, s):                                          # Total run time is O(n + m)
2.     for u in G.vertices() do                            # Set things up for BFS: O(n)
3.         visited[u] ← false
4.         parent[u] ← null
5.     seen[s] ← true
6.     layers ← []
7.     current ← [s]
8.     next ← []
9.     while current is not empty do                       # O(m)
10.        layers.append(current)
11.        for u in current do                             # iterate over current layer
12.            for v in G.incidentEdges(u) do              # iterate over neighbors of u
13.                if not seen[v] then
14.                    seen[v] ← true
15.                    parent[v] ← u
16.                    next.append(v)
17.        current ← next                                  # update current and next layer
18.        next ← []
19.    return layers, parent
```

4.1 BFS Properties

- Let C_v be the connected component of v in our graph G.
- Fact: BFS(G, s) visits all vertices in C_s.
- Fact: Edges $\{(u, \text{parent}[u]) : u \in C_s\}$ form a spanning tree T_s of C_s.
- Fact: For each v in L_i, there is a path in T_s from s to v with i edges.
- Fact: For each v in L_i, any path in G from s to v has at least i edges.
- Fact: Assuming adjacency list representation we can perform a BFS traversal of a graph with n vertices and m edges in O(n+m) time
- Fact: Assuming adjacency matrix representation we can perform a BFS traversal of a graph with n vertices and m edges in O(n^2) time

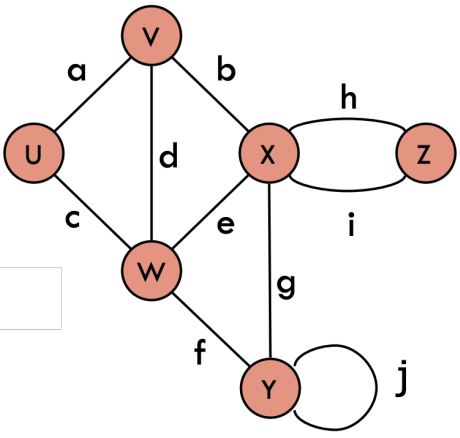
4.2 BFS Applications

BFS can be used to solve other graph problems in O(n + m) time. For example, it can find the shortest path between two given vertices, find a cycle in a graph, test whether a graph is connected or compute a spanning tree of a graph (if connected).

5 Terminology

5.1 Undirected graphs

- Edges are incident on endpoints (e.g., a, d, and b are incident on V)
- Adjacent vertices are connected (e.g., U and V are adjacent)
- Degree is the number of edges on a vertex (e.g., X has degree 5)
- Parallel edges share the same endpoints (e.g., h and i are parallel)
- Self-loop have only one endpoint (e.g., j is a self-loop)
- Simple graphs have no parallel or self-loops



5.2 Directed graphs

- Out-degree is the number of edges out of a vertex (e.g., W has out-degree 2)
- In-degree is the number of edges into a vertex (e.g., W has in-degree 1)
- Parallel edges share tail and head (e.g., no parallel edge on the right)
- Self-loop have the same head and tail (e.g., X has a self-loop)
- Simple directed graphs have no parallel or self-loops, but are allowed to have antiparallel loops like f and a