Discrete Mathematics

Unit 4: Graphs and Trees

Syedur Rahman

syedur.rahman@wolfson.oxon.org

Discrete Mathematics Lecture Notes

Acknowledgements

- These lecture notes contain some material from the following sources:
  - [ICM]: *Introduction to Computer Mathematics* by C. Runciman, 2003
  - [Rosen]: *Discrete Mathematics and Its Applications* by K. Rosen, 5th Edition, Tata McGraw-Hill Edition.

Definition of graphs

Graphs are discrete structure consisting of vertices and edges between vertices.

A **simple graph** $G = (V, E)$ consists of V a nonempty set of vertices and E, a set of unordered pairs of distinct elements of V called edges.

A **multigraph** $G = (V, E)$ consists of a set of vertices V, a set of edges E and a function f from E to $\{ \{u, v\} \mid u, v \in V, u \neq v\}$. Edges e_1 and e_2 are called multiple or parallel edges if $f(e_1) = f(e_2)$

A **pseudograph** $G = (V, E)$ consists of a set of vertices V, a set of edges E and a function f from E to $\{ \{u, v\} \mid u, v \in V\}$. An edges e_1 is a loop if $f(e_1) = \{u, v\} = \{u\}$ for some $u \in V$.

Definition of graphs

A **directed graph** $G = (V, E)$ consists of V a nonempty set of vertices and E, a set of ordered pairs of elements of V called edges.

A **directed multigraph** $G = (V, E)$ consists of a set of vertices V, a set of edges E and a function f from E to $\{ (u, v) \mid u, v \in V\}$. Edges e_1 and e_2 are called multiple or parallel edges if $f(e_1) = f(e_2)$

Type of Graph	***Edges***	***Multiple Edges***	***Loops***
Simple graph	Undirected	Not allowed	Not allowed
Multigraph	Undirected	Allowed	Not allowed
Pseudograph	Undirected	Allowed	Allowed
Directed graph	Directed	Not allowed	Allowed
Directed multigraph	Directed	Allowed	Allowed

Graph terminology

Consider an undirected graph $G = (V, E)$

Two vertices v and u are ***adjacent*** if an edge $e_x = \{v, u\}$ and $e_x \in E$. e_x is said to be ***incident*** with u and v. e_x is also said to connect u and v. The vertices u and v are called the ***endpoints*** of the edge e_x.

The ***degree*** of a vertex v, denoted by deg(v), is the number of edges incident with it, except that a loop at a vertex contributes twice to the degree of that vertex. A vertex of degree zero is called **isolated**. A vertex is **pendant** if and only if it has degree one.

Handshaking theorem:

$$\sum_{v \in V} \deg(v) = 2 \times |E|$$

Graph terminology

Consider a directed graph $G = (V, E)$

When $e_x = (u, v)$ and $e_x \in E$, u is said to be ***adjacent to*** v and said to be ***adjacent from*** u. u is the ***initial vertex*** of e_x *and* v is the ***terminal*** or ***end vertex*** of e_x

The ***in-degree*** of a vertex v, denoted by $\deg^-(v)$, is the number of edges for which v is the terminal vertex. The ***out-degree*** of a vertex v, denoted by $\deg^+(v)$, is the number of edges for which v is the initial vertex.

A theorem:

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

Special Types of Graphs

The **complete graph** on *n* vertices, denoted by K_n, is the simply graph that contains an edge between each pair of distinct vertices.

The **cycle** C_n (given that $n \geq 3$) consists of n vertices v_1, v_2, …, v_n and edge $\{v_1, v_2\}$, $\{v_2, v_3\}$, …, $\{v_{n-1}, v_n\}$ and $\{v_n, v_1\}$.

We obtain the **wheel** W_n, when we add an additional vertex to the cycle C_n (given that $n \geq 3$) and connect this new vertex to each of the *n* vertices in C_n by new edges.

The *n*-dimensional cube or ***n*-cube** Q_n, is the graph that has vertices representing 2^n bit strings of length *n*. Two vertices are adjacent if and only if their bit strings differ in exactly one bit position.

A simple graph *G* is called a **bipartite graph** if its vertex set *V* can be partitioned into two disjoint sets V_1 and V_2 such that every edge in the graph connects a vertex in V_1 to a vertex in V_2 (so no edge connects two vertices in V_1 or two edges in V_2).

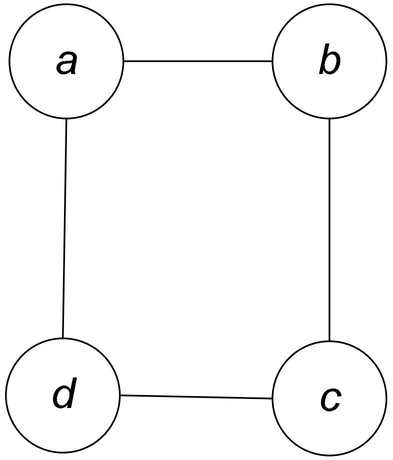
Representation of undirected graphs

Consider an undirected graph $G = (V, E)$

The **adjacency matrix** M for G is a matrix of dimensions $\#V$, where $M_{x,y} = 1$ if and only if $x, y \in V \wedge \{x,y\} \in E$ (or $\{y,x\} \in E$) .

A graph can also be represented using an **edge list**. For each vertex, this gives what vertices are adjacent to it.

An Undirected Graph



Its Adjacency Matrix

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

Its Edge List

Vertex	*Adjacent Vertices*
a	b, d
b	a, c
c	b, d
d	a, c

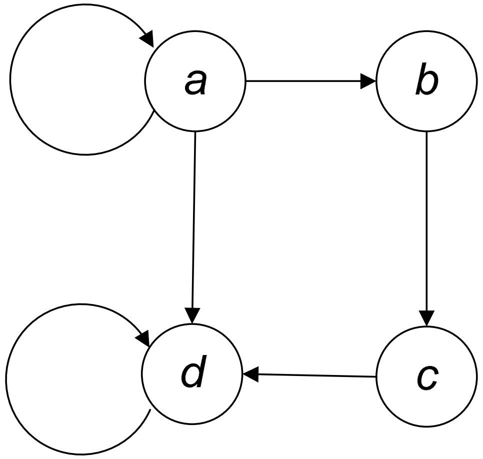
Representation of directed graphs

Consider a directed graph $G = (V, E)$

The **adjacency matrix** M for G is a matrix of dimensions $\#V$, where $M_{x,y} = 1$ if and only if $x, y \in V \wedge (x,y) \in E$.

A graph can also be represented using an **edge list**. For each vertex, this returns the vertices it is adjacent to.

A Directed Graph



Its Adjacency Matrix

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

Its Edge List

Initial Vertex	***Terminal Vertices***
a	a, b, d
b	c
c	d
d	d

Isomorphism of Graphs

The simple graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_1, E_2)$ are **isomorphic** if there is a one-to-one and onto function f from V_1 to V_2 with the property that a and b are adjacent in G_1 if and only if $f(a)$ and $f(b)$ are adjacent in G_2, for all a and b in V_1. Such a function f is called an isomorphism.

It is often difficult to whether two graphs are isomorphic exists since with n vertices there are $n!$ such correspondences. If two graphs G_1 and G_2 are isomorphic, they will have certain properties (or **invariant**)

- G_1 and G_2 have the same of vertices, i.e. $|V_1| = |V_2|$
- G_1 and G_2 have the same number of edges, i.e. $|E_1| = |E_2|$
- The degrees of vertices must be the same, i.e. vertex v of degree d in G_1 must correspond to a vertex $f(v)$ of degree d in G_2 since a vertex w in G_1 is adjacent to v if and only if $f(v)$ and $f(w)$ are adjacent in G_2.

Connectivity in Undirected Graphs

In an undirected graph, a **path** of length n from to u to v which are vertices in G, is a sequence of n edges, e_1, e_2, …, e_n where $f(e_1)=\{x_1, x_2\}$, $f(e_2)=\{x_2, x_3\}$, … $f(e_n)=\{x_{n-1}, x_n\}$ and $x_0 = u$ and $x_n = v$. When the graph is simple we denote this path by its vertex sequence i.e. $x_1, x_2, x_3, \ldots, x_{n-1}, x_n$.

The path is a **circuit** if it begins and ends at the same vertex (u = v).

An undirected graph is called **connected** if there is a path between every pair of distinct vertices on the graph.

A graph that is not connected is the union of two or more connected subgraph each pair of which has no vertex in common. These disjoint connected subgraphs are called the **connected components** of the original graph.

Connectivity in Directed Graphs

In an undirected graph, a **path** of length n from to u to v which are vertices in G, is a sequence of n edges, e_1, e_2, …, e_n where $f(e_1)=(x_1, x_2)$, $f(e_2)=(x_2, x_3)$, … $f(e_\text{n})=(x_{n\text{-}1}, x_n)$ and $x_0 = u$ and $x_n = v$. When the graph is simple we denote this path by its vertex sequence i.e. x_1, x_2, x_3, …, $x_{n\text{-}1}$, x_n.

The path of length greater than 0 is a **circuit** if it begins and ends at the same vertex (u = v).

A directed graph is called **strongly connected** if there is a path between every pair of distinct vertices on the graph. A graph that is not connected is the union of two or more connected subgraph each pair of which has no vertex in common.

A directed graph is called **weakly connected** if its underlying undirected graph is connected (i.e. there is a path each pair of vertices if the directions of edges are ignored)

The subgraphs of a directed graph G that are strongly connected are called the **strongly connected components** or **strong components** of G.

Hamilton Circuits and Weighted Graphs

In a graph $G = (V, E)$, a path $x_0, x_1, x_2, x_3, \ldots, x_{n-1}, x_n$ is called a **Hamilton path** if $V = \{x_0, x_1, x_2, x_3, \ldots, x_{n-1}, x_n\}$ and $x_i \neq x_j$ *for* $0 \leq i < j \leq n$. A circuit $x_0, x_1, x_2, x_3, \ldots, x_{n-1}, x_n, x_0$ (n>1) in a graph G is called a **Hamilton circuit** if $x_0, x_1, x_2, x_3, \ldots, x_{n-1}, x_n$ is a Hamilton path.

Graphs that have a number assigned to each edge are called **weighted graphs**. A weight graph G can be defined using a function w such that $w(u, v)$ gives the weight of the edge between vertice u and v, whereas $w(u, v) = \alpha$ if there is no path between them.

Weighted Graphs and Related Problems

Dijkstra's algorithm was proposed by Edsger Dijkstra in 1959, finds the shortest path between two vertices *u* and *v* in a weighted graph. The algorithm proceeds by finding the shortest path from *u* to a first vertex, the length of a shortest path from *u* to a second vertex and so on until it discovers the shortest path to *z*. Once, it finds the shortest path to a vertex it labels it with the path and the total weight (so that the path and weight can be replaced if a cheaper path is later found). Dijkstra's algorithm is $O(n^2)$.

The **travelling salesman's problem** is the problem of finding a path visiting each of the *n* nodes in a graph exactly once and return to the starting node.

Others define it as the problem of finding the circuit of minimum total weight in a weighted complete undirected graph that visits each vertex exactly once and return ot its starting point.

This is equivalent to asking for a Hamiltonian circuit with minimum total weight in the complete graph. With an initial vertex selected, there are (n-1)! Hamiltonian circuits in a complete graph.

E.g. with 25 vertices, there are 24!/2 (we divide by 2 since any such circuit can be travelled in reverse order) Hamilton circuits = 3.1×10^{23}. If checking each circuit takes 10^{-9} seconds, this exhaustive technique takes up about 10 million years.

Approximation algorithms are used instead the best of which are within 2% of the exact solution and only take a few minutes of computer time even with a 1000 vertices.

Introduction to Trees

A **tree** is a connected undirected graph with no simple circuits.

An undirected graph is a tree if and only if there is a unique simple path between any of its vertices.

Graphs that contain no simple circuits but are not connected are called **forests**. Connected components in a forest are trees by themselves.

A **rooted tree** is a tree in which one vertex has been designated as the root and every edge is directed away from the root.

A rooted tree in which each internal vertex corresponds to a decision with a subtreee at these vertices for each possible outcome of the decision is called a **decision tree**.

A tree where the vertices represent position of a game that a game can be in as it progresses (and edges represent legal moves between positions) are called **game trees**. Such trees are not only used in modelling games but also in complex planning and searching problems.

About Rooted Trees

Let us consider a rooted tree *T*:

If *v* is a vertex (other than the root) in *T*, the **parent** of *v* is the unique vertex *u* such that there is a directed edge from *u* to *v*. If *u* is the parent of *v* then *v* is a **child** of *u*. Vertices with the same parent are called **siblings**.

The **ancestors** of a vertex other than the root are the vertices in the path from the root to this vertex, excluding the vertex itself and including the root. The **descendents** of a vertex are those vertices that have it as an ancestor.

A vertex with no children is called a **leaf** while vertices with children are called **internal vertices**.

If *v* is a vertex of *T*, the **subtree** with *v* as its root is the subgraph of the *T* consisting of *v*, its descendents and all edges incident on them.

The **level** of a vertex *v* is the length of the unique path from the root to *v*. The **height** of *T* is the length of the longest path from any vertex to the root. A rooted tree of height *h* is **balanced** if all leaves are at levels *h* or *h*-1

M-ary and Binary Trees

A rooted tree is called an ***m*-ary tree** if every internal vertex has at most *m* children. All internal vertices in a **full *m*-ary tree** have exactly *m* children.

An *m*-ary tree with *m*=2 is called a **binary tree**.

An **ordered rooted tree** is a rooted tree where the children of each internal vertex are ordered. Such trees are drawn so that the children of each vertex are shown in order from left to right.

In an ordered binary tree, the **first child** of a vertex is called the **left child** and the other the **right child**. The tree rooted at the left child of a vertex is called its **left subtree** and that rooted at the right child is called its **right subtree**.

A **binary search tree** is a special ordered binary tree which is arranged according to the key identifying each vertex such that if *x* and *y* are left and right children of *v*, then $key_X \leq key_V \leq key_Y$

Some Properties of Trees

A tree with n vertices has n-1 edges

A full m-ary tree with i internal vertices has mi+1 vertices

There are at most m^h leaves in an m-ary tree of height h.

A full m-ary tree with

- n vertices has $i = (n-1)/m$ internal vertices and $l = [(m-1)n+1]/m$ leaves
- i internal vertices has $n = mi+1$ vertices and $i = (m-1)i+1$
- l leaves has $n = (ml-1)/(m-1)$ vertices and $i = (l-1)/(m-1)$ internal vertices.

Tree Traversal

Let T be an ordered rooted tree with root r. If T consists only of r, then r is the **preorder traversal** of T. Otherwise, suppose that T_1, T_2, …, T_n are the subtrees at r from left to right in T. Then **preorder traversal** begins by visiting r. It continues by traversing T_1 in preorder, then T_2 in preorder and so on until T_n is traversed in preorder.

Let T be an ordered rooted tree with root r. If T consists only of r, then r is the **inorder traversal** of T. Otherwise, suppose that T_1, T_2, T_3, …, T_n are the subtrees at r from left to right in T. Then **inorder traversal** begins by T_1 in inorder and then visits r. It continues by traversing T_2 in inorder, then T_3 in inorder and so on until T_n is traversed in inorder.

Let T be an ordered rooted tree with root r. If T consists only of r, then r is the **postorder traversal** of T. Otherwise, suppose that T_1, T_2, …, T_n are the subtrees at r from left to right in T. Then **postorder traversal** begins by visiting T_1 in postorder, then T_2 in postorder and so on until T_n is traversed in postorder, and finally it visits r.

Search Algorithms & Spanning Trees

Breadth first search and **depth first search** are two algorithms that can be used to find a vertex (goal node) in a graph given an initial node (which becomes the root of the search tree).

A **spanning tree** of a simple graph, is its subgraph that is a tree containing every one of its vertices.

A **minimum spanning tree** in a connected weighted graph is a spanning tree that has the smallest possible sum of weights of its edges.

Kruskal's and **Prim's** algorithms are used to find minimum spanning trees.