CSC 445 - Intro to Intelligent Robotics, Spring 2018

Graphs

Graphs

- **Definition:** A *graph* $G = (V, E)$ consists of a nonempty set V of *vertices* (or *nodes*) and a set E of *edges*. Each edge has either one or two vertices associated with it, called its *endpoints*. An edge is said to *connect* its endpoints.

Some Terminology

- In a *simple graph* each edge connects two different vertices and no two edges connect the same pair of vertices.
- A *multigraph* may have multiple edges connecting the same two vertices. When m different edges connect the vertices u and v, we say that $\{u, v\}$ is an edge with *multiplicity* m.
- An edge that connects a vertex to itself is called a *loop*.
- A *pseudograph* may include loops, as well as multiple edges connecting the same pair of vertices.

Directed Graphs

- **Definition:** An *directed graph* (or *digraph*) $G = (V, E)$ consists of a nonempty set of V *vertices* (or *nodes*) and a set of *directed edges* (or *arcs*). Each edge is associated with an ordered pair of vertices. The directed edge associated with the ordered pair (u, v) is said to *start at u* and *end at v*.

Some Terminology (continued)

- A *simple directed graph* has no loops and no multiple edges
- A *directed multigraph* may have multiple directed edges. When there are m directed edges from the vertex u to the vertex v, we say that (u, v) is an edge of *multiplicity* m.

Summary of Graph Terminology

Type	Edges	Multiple Edges	Loops
Simple graph	Undirected	No	No
Multigraph	Undirected	Yes	No
Pseudograph	Undirected	Yes	Yes
Simple directed graph	Directed	No	No
Directed multigraph	Directed	Yes	Yes
Mixed graph	Both	Yes	Yes

Representing Graphs: Adjacency Lists

- **Definition:** An *adjacency list* can be used to represent a graph with no multiple edges by specifying the vertices that are adjacent to each vertex in the graph

Representing Graphs: Adjacency Matrices

- **Definition:** Suppose that $G = (V, E)$ is a simple graph where $|V| = n$. Arbitrarily list the vertices of G as $v_1, v_2, \ldots, v_n$. The *adjacency matrix* A_G of G, with respect to the listing of vertices, is the $n \times n$ zero-one matrix with 1 as its $(i, j)th$ entry when v_i and v_j are adjacent, and 0 as its $(i, j)th$ entry when they are not adjacent.

Adjacency Matrices (continued)

- Adjacency matrices can also be used to represent graphs with loops and multiple edges.
- A loop at the vertex v_i is represented by a 1 at the $(i,j)th$ position of the matrix.
- When multiple edges connect the same pair of vertices v_i and v_j, the $(i,j)th$ entry equals the number of edges connecting the pair of vertices.

Adjacency Matrices (continued)

- Adjacency matrices can also be used to represent directed graphs. The matrix for a directed graph $G = (V, E)$ has a 1 in its $(i, j)th$ position if there is an edge from v_i to v_j where $v_1, v_2, \ldots, v_n$ is a list of vertices.
- The adjacency matrix for a directed graph does not have to be symmetric because there may not be an edge from v_i to v_j when there is an edge from v_j to v_i.
- To represent directed multigraphs, the value of a_{ij} is the number of edges connecting v_i to v_j.

Paths

- **Informal Definition:** A *path* is a sequence of edges that begins at a vertex of a graph and travels from vertex to vertex along edges of the graph. As the path travels along its edges, it visits the vertices along this path, that is, the endpoints of these.

Paths

- **Definition:** Let n be a nonnegative integer and G an undirected graph. A *path* of *length* n from u to v in G is a sequence of n edges $e_1, \ldots, e_n$ of G for which there exists a sequence $x_0 = u, x_1, \ldots, x_{n-1}, x_n = v$ of vertices such that e_i has, for $i = 1, \ldots, n$, the endpoints x_{i-1} and x_i.
  - When the graph is simple, we denote this path by its vertex sequence $x_0, x_1, \ldots, x_n$ (since listing the vertices uniquely determines the path).
  - The path is a *circuit* if it begins and ends at the same vertex ($u = v$) and has length greater than zero.
  - The path or circuit is said to *pass through* the vertices $x_1, x_2, \ldots, x_{n-1}$ and *traverses* the edges $e_1, \ldots, e_n$.
  - A path or circuit is *simple* if it does not contain the same edge more than once.

Connectedness in Undirected Graphs

- **Definition:** An undirected graph is called *connected* if there is a path between every pair of vertices. An undirected graph is that is not connected is called *disconnected*. We say that we *disconnect* a graph when we remove vertices or edges, or both, to produce a disconnected subgraph.
- **Definition:** A *connected component* of a graph G is a connected subgraph of G that is not a proper subgraph of another connected subgraph of G.

Connectedness in Directed Graphs

- **Definition:** An directed graph is called *strongly connected* if there is a path from a to b and a path from b to a whenever a and b are vertices in the graph.
- **Definition:** A directed graph is *weakly connected* if there is a path between every two vertices in the underlying undirected graph, which is the undirected graph obtained by ignoring the directions of the edges of the directed graph.

Trees

- **Definition:** A *tree* is a connected undirected graph with no simple circuits.
- **Definition:** A *forest* is a graph that has no simple circuit but is not connected. Each of the connected components in a forest is a tree.

Rooted Trees

- 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.
- An unrooted tree is converted into different rooted trees when different vertices are chosen as the root.

Rooted Tree Terminology

- If v is a vertex of a rooted tree other than the root, the *parent* of v is the unique vertex u such that there is a directed edge from u to v. When u is a parent of v, v is called the *child* of u. Vertices with the same parent are called *siblings*.
- The *ancestors* of a vertex are the vertices in the path from the root to this vertex, excluding the vertex itself and including the root.
- The *descendants* of a vertex v are those vertices that have v as an ancestor.
- A vertex of a rooted tree with no children is called a *leaf*. Vertices that have children are called *internal vertices*.
- If a is a vertex in a tree, the *subtree* with a as its root is a tree consisting of a and its descendants and all edges incident to these descendants.

Spanning Trees

- **Definition:** Let G be a simple graph. A *spanning tree* of G is a subgraph of G that is a tree containing every vertex of G.

Searching a Graph

```
function EXPLORE($G, v \in V$)
    visited($v$) = true
    previsit($v$)
    for each edge $(v, u) \in E$ do
        if not visited($u$) then
            EXPLORE($u$)
    postvisit($v$)
```

where

- visited is a predicate
- previsit is an optional procedure
- postvisit is an optional procedure

Depth First Search

```
function DFS($G$)
    for all $v \in V$ do
        visited($v$) $=$ false
    for all $v \in V$ do
        if not visited($v$) then
            EXPLORE($v$)
```

Breadth-First Search

```
function $\mathrm{BFS}(G, s \in V)$
    for all $u \in V$ do
        $\text{dist}(u) = \infty$
    $\text{dist}(s) = 0$
    $Q = [s]$ (queue containing only $s$)
    while $Q$ is not empty do
        $u = \text{eject}(Q)$
        for all edges $(u, v) \in E$ do
            if $\text{dist}(v) = \infty$ then
                inject(Q, $v$)
                $\text{dist}(v) = \text{dist}(u) + 1$
```

Shortest Paths in Graphs

- **Definition:** A *weighted graph* is a graph in which the edges have been assigned *weights* where the weights are numerical values.
- The *shortest path problem* is the problem of finding a path between two vertices in a weighted graph such that the sum of the weights of its constituent edges is minimized.
- Dijkstra's algorithm solves the problem of finding the shortest path from a source vertex $v \in V$ to all other vertices in the graph assuming that the weights are nonnegative.

Dijkstra's Algorithm

```
function $\textsc{Dijkstra}(G, w, s \in V)$
    for all $u \in V$ do
        $\text{dist}(u) = \infty$
        $\text{prev}(u) = \text{nil}$
    $\text{dist}(s) = 0$
    $H = \text{makequeue}(V)$ (priority queue using dist values)
    while $H$ is not empty do
        $u = \text{deletemin}(H)$
        for all edges $(u, v) \in E$ do
            if $\text{dist}(v) > \text{dist}(u) + w(u, v)$ then
                $\text{dist}(v) = \text{dist}(u) + w(u, v)$
                $\text{prev}(v) = u$
                $\text{decreasekey}(H, v)$
```

A* Search

- A* search selects a path that minimizes

  $$f(v) = g(v) + h(v)$$

  where $g(v)$ is the cost of the path from the start vertex to v and h is a heuristic that estimates the cost of the cheapest path from v to the goal.
- The heuristic function must be *admissible*, meaning that it never overestimates the actual cost to get to the goal vertex.
- If the heuristic is *monotone* if satisfies the condition

  $$h(x) \leq w(x, y) + h(y)$$

  for every edge (x, y) of the graph.
- For an admissible, monotone heuristic, A* is equivalent to Dijkstra's algorithm with the reduced cost function

  $$w'(x, y) = w(x, y) + h(y) - h(x)$$