**Graphs**

At it’s most basic a graph is both a set of vertices and a set of edges.

* Structure – We can reflect back on tress
  + Trees are a type of graph.
    - Differences:
      * Only 2 connections outgoing
      * No cycles
  + Any vertex can have n-1 edges although this is not necessary or even common.
  + We describe the edges in terms of their vertices ie. (A,B) is an edge connecting A and B.
  + Graphs can be:
    - Directed:
      * Edges may have direction A -> B
        + (A,B) != (B,A)
    - Undirected:
      * Edges work in both directions A-B
        + (A,B) == (B,A)
    - Symmetric or Asymmetric:
      * Generally follows Directed
    - Weighted;
      * Edges can have weight assigned
      * A weight can be any value.
      * Adjacency – we have various edges and paths in a group
      * To vertices are adjacent if distinctly.
      * An edge is incident to a vertex if it is ‘moving into’ that vertex.
        + The the edge is incident to B.
      * In-degree: The # of edges moving into a vert.
      * Out-degree: The # of vertices moving out a sam
* Terminology
  + Lists, Trees, etc... -> nodes, connections
  + Graphs -> Vertex, edge.
  + Simple Graph: no loops or multiple edges between vertices.
  + Path: a series of one or more edges between two vertices.
  + Cycle: A path where start and end are the same.
  + Connected: undirected, a path from one to any other vertex.
  + Strongly Connected: directed, path from one to any other.
  + Sparse, Dense, Complete: how many edges, relatively... except Complete.
  + Spanning Tree: A subgraph which contains all the vertices but not necessarily all edges
    - No Cycles
    - Lower “cost”
* Algorithms
  + Graph: { Vertices }

{ ( , ) ( , ) }

{ ( , ) ( , ) }

* We will use: Adjacency Matrix, adjacency list.
  + Matrix 2D table where each side is labelled as a vertex and edges are the cells.
    - Unweighted -> cell = 1
    - Weighted -> cell = weight.
      * int[ ][ ] adjMatrix = new int [ 5 ][ 5 ];

V = {0,1,2,3,4}

* E= {0,1), 1(,0), (1,2), (1,3), (2,1), (3,1), (3,4), (4,3)

0 1 2 3 4

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | 1 3 |  |  |  |
| 1 3 |  | 1 3 | 1 3 |  |
|  | 1 3 |  |  |  |
|  | 1 3 |  |  | 1 4 |
|  |  |  | 1 4 |  |

* + List: for each Vertex we list its adjacencies
    - 0 1
    - 1 0, 2, 3
    - 2 1
    - 3 1, 4
    - 4 3
* Searching (traversal):
  + Vs. Traversal are we going to every vertex or looking for something?
  + 2 kinds:
    - Depth first
    - Breadth first
* Spanning Tree:
  + Produces a subgraph
    - All vertices
    - Minimal cost
    - No Cycles
  + Important application – network switches
  + 2 Algorithms:
    - Prim’s
    - Kruskal’s
* Shortest Path:
  + Determine the shortest path through a graph from vertex \_\_ to the other vertices in the graph.
  + The algorithm we’ll look at is called Dijkstra’s algorithm.
  + Can use number of hops but usually uses weights.

|  |  |
| --- | --- |
| Graph | Vertex |
| List Vertices  AdjMatrix  Stack  Queue  NumVerts  Const.  addVertex()  addEdge()  showGraph() | Label  wasVisited |