Graphs and Graph Algorithms

Formal Definition: Graph

Def'n: A graph \( G = (V, E) \) is a set of vertices \( V = \{v_1, v_2, \ldots\} \) together with a set of edges \( E = \{e_1, e_2, \ldots\} \) that connect pairs of vertices.

Edges can be thought of as tuples of vertices. That is \( e_m = (v_s, v_t) \)

Graph: More Detail

- In general
  - Parallel edges are allowed
  - Self loops are allowed
- However, graphs without parallel edges and without self-loops are called simple graphs
- In general, assume graph is simple unless otherwise specified
Graphs: Complexity

- Complexity of graph algorithms is typically defined in terms of:
  - Number of edges $|E|$, or
  - Number of vertices $|V|$, or
  - Both

Graphs: Data Structures

- Sparse Graph
  - few edges ($|E| \ll |V|^2$) or ($|E| \approx |V|$)
  - represent as adjacency list

- Dense Graph
  - many edges ($|E| \approx |V|^2$)
  - represent as adjacency matrix

Graphs: Directed vs Undirected

- Directed Graph (aka digraph)
  - edges have direction
  - nodes on edges form ordered pairs
    - order of vertices in edge is important
    - $e_u = (u, v)$ means there is an edge from $u$ to $v$

- Undirected Graph
  - nodes on edges form unordered pairs
    - order of vertices in edge is not important
    - $e_u = (u, v)$ means there is an edge between $u$ and $v$
Graphs: Weighted Graphs

- Edges may be 'weighted'
  - often, algorithms search a graph for a path (unweighted), or least cost path (weighted)

Graphs: Definitions

- Simple Path: sequence of edges leading from one vertex to another with no vertex appearing twice
- Connected Graph: a simple path exists between any pair of vertices
- Cycle: simple path, except that first and final nodes are the same

Graphs: Data Structures

Adjacency Matrix Implementation
- $|V| \times |V|$ matrix representing graph
- directed vs undirected
  - directed adjmat has to/from
  - undirected adjmat only needs $\frac{|V|^2}{2}$ space
- unweighted vs weighted
  - unweighted: 0:no edge::1:edge
  - weighted: $\infty$:no edge::val:edge
Graphs: Data Structures

Adjacency List Implementation
- complexity determined as follows:
  - edges are distributed on vertices \((E/V)\)
  - costs 1 to access a vertex list
  - cost for individual vertex is \(O(1 + E/V)\)
  - cost for all vertices is \(O(V) \times O(1 + E/V) = O(V + E)\)
- directed vs undirected
  - directed adjlist contains each edge once in edge set
  - undirected adjlist contains each edge twice in edge set
- unweighted vs weighted
  - unweighted: null:no edge :: list item:edge
  - weighted: null:no edge :: val in list item:edge

Depth-First Search

Given a graph \(G = (V,E)\), systematically explore the edges of \(G\) to discover if a path exists from the source \(s\) to the goal \(g\)
- that is, create stack (think project 1)
- algorithm works on both graphs and digraphs
- discovers a path from source \(s\) to goal \(g\) if it exists

Depth-First Search

Algorithm
1. Form a one element stack of the source \(s\)
2. Until the stack is empty or the goal \(g\) is reached, determine if the first element in the stack is \(g\)
   2a. If the first element is \(g\), do nothing
   2b. If the first element is not \(g\), remove the first element from the stack and add the first element’s unvisited children (if any) to the front of the stack
3. If \(g\) is found, announce ‘success’, Else announce ‘failure’
DFS Algorithm

The algorithm uses a mechanism for setting and getting "labels" of vertices and edges.

**Algorithm DFS(G)**

**Input** graph G

**Output** labeling of the edges of G as discovery edges and back edges

for all \( u \in G.\text{vertices}() \)

setLabel(\( u \), \text{VISITED}())

for all \( e \in G.\text{edges}() \)

setLabel(\( e \), \text{UNEXPLORED}())

for all \( v \in G.\text{vertices}() \)

if getLabel(\( v \)) = \text{UNEXPLORED}()

\( w \leftarrow \text{opposite}(v, e) \)

if getLabel(\( w \)) = \text{UNEXPLORED}()

setLabel(\( e \), \text{DISCOVERY}())

DFS(G, \( w \))

else

setLabel(\( e \), \text{BACK}())

**Algorithm DFS(G, v)**

**Input** G and a start vertex v of G

**Output** labeling of the edges of G in the connected component of v as discovery edges and back edges

setLabel(v, \text{VISITED}())

for all \( e \in G.\text{incidentEdges}(v) \)

if getLabel(\( e \)) = \text{UNEXPLORED}()

w \leftarrow \text{opposite}(v, e)

if getLabel(\( w \)) = \text{UNEXPLORED}()

setLabel(\( e \), \text{DISCOVERY}())

DFS(G, \( w \))

else

setLabel(\( e \), \text{BACK}())

Example

- unexplored vertex
- visited vertex
- unexplored edge
- discovery edge
- back edge

Example (cont.)

- unexplored vertex
- visited vertex
DFS: Analysis of Adjacency List

- DFS:
  - Called for each vertex exactly once
    - $O(V)$
  - Adjlist for each vertex is visited at most once and set of edges is distributed over set of vertices
    - $O(1 + E/V)$
  - $O(V + E)$: linear with number of vertices and edges

DFS: Analysis of Adjacency Matrix

- DFS:
  - Called for each vertex exactly once
    - $O(V)$
  - Adjmat row for each vertex is visited at most once
    - $O(V)$
  - $O(V^2)$: quadratic with number of vertices

Breadth-First Search

Given a graph $G = (V, E)$, systematically explore the edges of $G$ to discover the shortest path from the source $s$ to the goal $g$

- that is, create queue (think project 1)
- algorithm works on both graphs and digraphs
- discovers shortest path from source $s$ to any goal $g$
Breadth-First Search

Algorithm
1. Form a one element queue of the source \( s \)
2. Until the queue is empty or the goal \( g \) is reached, determine if the first element in the queue is \( g \)
   2a. If the first element is \( g \), do nothing
   2b. If the first element is not \( g \), remove the first element from the queue and add the first element’s unvisited children (if any) to the end of the queue
3. If \( g \) is found, announce ‘success’, Else announce ‘failure’

BFS Algorithm

The algorithm uses a mechanism for setting and getting "labels" of vertices and edges

Algorithm BFS(G, v)
Input graph \( G \)
Output labeling of the edges and partition of the vertices of \( G \)
for all \( u \in G\).vertices() setLabel(u, UNEXPLORED)
for all \( e \in G\).edges() setLabel(e, UNEXPLORED)
for all \( v \in G\).vertices() if getLabel(v) = UNEXPLORED BFS(G, v)

Example

\[ \text{unexplored vertex} \quad \text{visited vertex} \quad \text{unexplored edge} \quad \text{discovery edge} \quad \text{cross edge} \]
Example (cont.)

BFS: Analysis of Adjacency List

- BFS:
  - Called for each vertex exactly once
    - $O(V)$
  - Adjlist for each vertex is visited at most once and set of edges is distributed over set of vertices
    - $O(1 + E/V)$

- $O(V + E)$: linear with number of vertices and edges
BFS: Analysis of Adjacency Matrix

- BFS:
  - Called for each vertex exactly once
    - $O(V)$
  - Adjmat row for each vertex is visited at most once
    - $O(V)$

- $O(V^2)$: quadratic with number of vertices

Graph Summary

- Background and Definitions
- Implementation
  - as adjacency matrix
  - as adjacency list
- Depth-First Search
  - implementation of stack
- Breadth-First Search
  - implementation of queue