CS C341 / IS C361
Data Structures & Algorithms

GRAPH ALGORITHMS

- Relations and Graphs
- Flows and Graphs
- Graph Properties
    - Symmetry: Directed vs. Undirected
    - Transitivity: Paths and Cycles
    - Weighted Graphs
    - Degrees and Connectedness
- Graph Representation
    - Adjacency Matrices
    - Adjacency Lists
    - Edge Lists

RELATIONS AND DATA STRUCTURES

- Sorted Lists
  - Used for capturing total order relations
- Trees
  - Used for capturing partial order relations
    - E.g. order of evaluating an expression
    - E.g. Priority order of processes
- Graphs
  - Used for capturing arbitrary binary relations

RELATIONS AND GRAPHS

- A binary relation R on a set S of elements is defined as a subset of S x S.
  - In general the pair (S,R), where R is a subset of S x S defines the relation R on elements S
- A relation is then modeled by a graph G defined as the pair (V,E) where
  - V is the set of vertices (or nodes)
    - V models S
  - E, a subset of V x V, is the set of edges (or links)
    - E models R
- Terminology
  - Often we'd say
    - G models R
  - to mean
    - G = (V,E) models (S,R)

RELATIONS AND GRAPHS - EXAMPLES

- A program is written as a set of files. (For compilation) a file may depend on another file. Capture the order of compilation (i.e. the dependencies) as a graph:
  - G = (V,E)
    - where V is the set of files and
    - E = { (f1, f2) | f1 and f2 are in V, f1 depends on f2 i.e. f2 must be compiled before f1 }
- A political map (of regions) captures adjacency (border) relations. This can be represented as a graph:
  - G=(V,E)
    - where V is the set of regions and
    - E = { (r1,r2) | r1 and r2 are in V, r1 is adjacent to (i.e. bordering) r2 }

RELATIONS AND GRAPHS - EXAMPLES

- Quick Exercises:
  - Capture the relation “is a classmate of” using a graph.
  - Capture the relation “is a friend of” using a graph.
  - Capture the relation “is connected by road” using a graph.
  - Capture the relation “can be seen from” on locations using a graph.
  - Capture the relation “has a pointer to” on data structures (often referred to as data objects or just objects)
  - Capture the relation “ belong to the same Facebook community“ on netizens
  - Capture the relation “ has a hyperlink to” on web pages

NETWORKS/FLOWS AND GRAPHS

- Networks/Flows also can be captured by graphs.
  - Usually flows happen on networks
    - i.e. typically a network is what gets captured in a graph along with (flow) capacities or costs
- Weighted Graph: G = (V,E,w) where
  - V and E are defined as earlier
  - w is a function on E
    - i.e. w : E --> Num and Num is typically N, Z, Q or R.
- Terminology:
  - We may (depending on the context) ignore w and talk about the projection (V,E) of the graph (V,E,w).

FLOWS AND GRAPHS

- Examples / Exercises:
  - Rivers with tributaries and distributaries
    - What are the vertices? What are the edges? What is the weight function?
  - Computer network
  - Rail (or other traffic) network
  - Electrical circuits
  - Program execution
    - Flow of control
    - Flow of data

GRAPHS - PROPERTIES

- A relation captured by a graph may be symmetric or asymmetric
  - Then the graph is referred to as undirected or directed respectively.
- Exercises:
  - For each of the following relations/networks decide whether you need a directed or an undirected graph:
    - Dependencies on files
    - Adjacencies of regions
    - Friends
    - Classmates
    - Connectivity by road
    - Visibility
    - Computer network
    - River network
    - Pointer-based data structures

GRAPHS - PROPERTIES

- A relation captured by a graph may be transitive or not
- A path in a graph G = (V,E) is defined as a sequence of edges (or vertices):
  - A path p from vertex v1 to vertex v2 is defined by a sequence of vertices (v_{j0}, v_{j1}, v_{j2}, … v_{jn-1}, v_{jn}) where
    - for each k from 0 to n-1 (v_{jk} , v_{jk+1}) is in E and v1= $v_{j0,}$ and v2= $v_{jn,}$
- A path captures "the transitivity" of the relation being modeled.
- A simple path from v1 and v2 is a path (v1=v_{j0}, v_{j1}, v_{j2}, … v_{jn-1}, v_{jn} =v2) such that
  - for each k=1 to n-1 each v_{jk} is unique.

GRAPHS - PROPERTIES

- Exercises:
  - What is the meaning of a path in the following examples?
    - Dependencies on files
    - Adjacencies of regions
    - Friends
    - Classmates
    - Connectivity by road
    - Visibility
    - Computer network
    - River network
    - Pointer-based data structures
    - Web pages and hyperlinks

GRAPHS - PROPERTIES

- A (simple) path from vertex v1 to itself is referred to as a cycle.
  - (Non-)Existence of cycles is an important property.
  - Graphs without cycles are referred to as Acyclic Graphs
    - In particular, directed graphs without cycles are referred to as Directed Acyclic Graphs (DAGs)
- In which of the following examples is “a cyclic path” interesting / meaningful / should be restricted?
    - Dependencies on files
    - Adjacencies of regions
    - Friends
    - Classmates
    - Connectivity by road
    - Visibility
    - Computer network
    - River network
    - Pointer-based data structures
    - Web Hyperlinks

SUBCLASSES OF GRAPHS

- What kind of a graph captures a total relation?
  - Degree of every node is at least 1
- What kind of a graph captures a function?
  - Assume f(a)=b is modeled as directed edge from a to b
    - Out-degree of every node is exactly 1
  - Alternatively, f(a)=b is modeled as directed edge from b to a
    - In-degree of every node is exactly 1
  - What about a 1-to-1 function?
    - In-degree and out-degree of every node are exactly 1 each
- What does a tree capture?
  - A (directed) tree captures a function:
    - If (u,v) is an edge then f(v)=u
    - Also, there are no cycles in a tree:
      - i.e. if f is defined on S, there is no subset T of S, such that f is a permutation on T.

GRAPHS - REPRESENTATION

- How do you represent a graph?
  - What operations are usually needed?
- Typical “high level” operations:
  - Traversing a graph / Uncovering a path
    - i.e. traversing a network
    - i.e. uncovering transitivity
- Typical “low level” operations:
  - Are two elements (directly) related?
    - Is there an edge between two vertices?
  - Find all elements related to a given element.
    - i.e. vertices adjacent to a given vertex.
  - How many elements are related to a given element?

GRAPHS – REPRESENTATION – ADJACENCY MATRIX

- Adjacency Matrix:
  - Given a directed graph G = (V,E) a boolean matrix M can be used to represent G:
    - |M| = |V| x |V|
    - M[j,k] = 1 if (j,k) is in E; 0 otherwise
  - Modify appropriately for undirected graph.
  - Given a directed graph G=(V,E,w) a matrix M can be used to represent G:
    - |M| = |V| x |V|
    - M[j,k] = w((j,k)) if (j,k) is in E; ?? Otherwise
      - Alternatively one may assume w is a total function, and define
      - M[j,k] = w((j,k))

GRAPHS – REPRESENTATION – ADJACENCY MATRIX

- Cost of typical "low level" operations:
  - Is there an edge between two vertices?
    - O(|V|)
  - Find all vertices adjacent to a given vertex.
    - O(|V|)
  - How many elements are related to a given element?
    - O(|V|)

GRAPHS – REPRESENTATION – ADJACENCY LISTS

- Adjacency Lists:
  - Given a directed graph G = (V,E) a table AL can be used to represent G:
    - |AL| = |V|
    - k is in AL[j] iff (j,k) is in E
  - Modify appropriately for undirected graph.
  - Given a directed graph G=(V,E,w) a matrix M(G) can be used to represent G:
    - |AL| = |V|
    - (k ,w((j,k))) is in AL[j] iff (j,k) is in E;
      - Alternatively one may assume w is a total function.
        - Why is this bad??

GRAPHS – REPRESENTATION – ADJACENCY LISTS

- Cost of typical "low level" operations:
  - Is there an edge between two vertices?
    - O(|V|) in the worst case
  - Find all vertices adjacent to a given vertex.
    - O(|V|) in the worst case
    - O(d(v)) for a given vertex v, where d is the "degree" of the vertex.
      - This is useful if vertices in the graph are "low degree"
  - How many elements are related to a given element?
    - O(|V|) unless a count is kept, in which case it is O(1)

GRAPHS – REPRESENTATION –EDGE LIST

- Edge List:
  - Given a graph G = (V,E) a list EL can be used to represent G:
    - |EL| = |E|
    - (j,k) is in EL iff (j,k) is in E
  - Given a graph G=(V,E,w) a matrix M(G) can be used to represent G:
    - |EL| = |E|
    - (j, k ,w((j,k))) is in EL iff (j,k) is in E;
      - Alternatively one may assume w is a total function.
        - Why is this bad??

GRAPHS – REPRESENTATION – EDGE LIST

- Cost of typical “low level” operations:
  - Is there an edge between two vertices?
    - O(|E|) in the worst case
  - Find all vertices adjacent to a given vertex.
    - O(|E|) in the worst case
  - How many elements are related to a given element?
    - O(|E|) in the worst case
- This representation is useful if E is sparse i.e. |E| << |V|*|V|
  - Why?
- Exercise:
  - Compare the space complexity of Edge List with the other two representations for various values of |E| from say, log|V|, |V|/k for some constant k, k*|V| for some constant k, |V|*log|V|, to |V|*|V|