**Graph Algorithms**

Chapter 9 Overview

- Definitions
- Representation
- Topological Sort
- Graph Matching (for Program 3)
- Shortest Path Algorithms
- Network Flow Problems
- Minimum Spanning Trees
- Depth-First and Breadth-First Search
- NP-Completeness

**Graphs and Digraphs**

A graph is a pair \( G = (V,E) \) where

- \( V \) is a set of vertices (or nodes)
- \( E \) is a set of edges (or arcs)

Example:

\[ V = \{ a, b, c, d \} \]
\[ E = \{ (a,b), (b,a), (a,c) \} \]

This graph represents a binary relation that is not symmetric.

**More Examples**

**Undirected Graph**

Lynnwood

Woodinville

Seattle

Tacoma

Bellevue

I5

I405

I405

I5

M126

142

143

321

322

326

373

415

EE562

**Directed Graph**

This graph represents a binary relation that is not symmetric.

**Paths through Graphs**

A path in a graph is a sequence of vertices \( v_1, v_2, \ldots, v_N \) such that \( (v_i,v_{i+1}) \in E \) for \( 1 \leq i < N \).

The length of the path is \( N-1 \), the number of edges on the path.

A path from a node to itself with no repeated edges is a cycle.

What are the cycles of this graph?
What are all the paths from \( a \) to \( f \)?
Paths through Digraphs

A path in a digraph is a sequence of vertices \( w_1, w_2, \ldots, w_N \) such that \((w_i, w_{i+1}) \) is in \( E \) for \( 1 \leq i < N \).

The only difference is moving along directed edges in the proper direction.

What are the cycles of this digraph?
What are all the paths from \( a \) to \( f \)?

This is an acyclic digraph.

Graph Representations

- \( N \times N \) Adjacency Matrix for \( N \) node graph

  Digraphs:
  \[
  A_{i,j} =
  \begin{cases}
  1 & \text{if there is an arc from node } i \text{ to node } j \\
  0 & \text{otherwise}
  \end{cases}
  \]

Note: since \( A_{i,j} = 1 \) if \( A_{j,i} = 1 \), we only need to store half the matrix.

2. Linked Representation: Adjacency Lists

- \( N \) element array of lists

  \( V[i] \) points to a list of nodes that are adjacent to node \( i \).

  For digraphs, this is usually the list of nodes reachable by following one arc out of node \( i \).

  But, we can have another set of lists for nodes whose arcs go into node \( i \).

What does this structure tell us for a graph?
For a digraph?

Topological Sort

Given an acyclic digraph \( G \), where

\( (N_i, N_j) \in E \) means that \( N_i \) precedes \( N_j \)

Find an ordering of the nodes \( N_1, N_2, N_3, \ldots, N_n \)
so that \( N_1 \prec N_2 \prec N_3 \prec \ldots \prec N_n \).

Find an ordering in which all these courses can be taken, satisfying their prerequisites.
Complexity of Topological Sort

Assuming the adjacency list representation,
- The indegree of each vertex is computed in an initialization step. \( |V| \)
- Each node will go into the queue and come out exactly once. \( |V| \)
- Each edge will be examined once (in the for loop when its from-node is processed). \( |E| \)

So the complexity is \( O(|V| + |E|) \).

Graph Matching

Input: 2 digraphs \( G_1 = (V_1,E_1), G_2 = (V_2,E_2) \)

Questions to ask:
- Are \( G_1 \) and \( G_2 \) isomorphic?
- Is \( G_1 \) isomorphic to a subgraph of \( G_2 \)?
- How similar is \( G_1 \) to \( G_2 \)?
- How similar is \( G_1 \) to the most similar subgraph of \( G_2 \)?

Isomorphism for Digraphs

\( G_1 \) is isomorphic to \( G_2 \) if there is a 1-1, onto mapping \( h: V_1 \to V_2 \) such that

\[
(v_i,v_j) \in E_1 \iff (h(v_i),h(v_j)) \in E_2
\]

Find an isomorphism \( h: \{1,2,3,4,5\} \to \{a,b,c,d,e\} \).
Check that the condition holds for every edge.

Isomorphism and subgraph isomorphism are defined similarly for undirected graphs.

In this case, when \( (v_i,v_j) \in E_1 \), either \( (v_i,v_j) \text{ or } (v_j,v_i) \) can be listed in \( E_2 \), since they are equivalent and both mean \( (v_i,v_j) \).

Similar Digraphs

Sometimes two graphs are close to isomorphic, but have a few “errors.”

Let \( h(1)=b, h(2)=e, h(3)=c, h(4)=a, h(5) = d \).

\[
\begin{array}{c|c}
(1,2) & (b,e) \\hline
(2,1) & (e,b) \\hline
(3,2) & (c,b) \\hline
(4,5) & (a,d) \\hline
(2,5) & (e,d) \\hline
(3,4) & (c,a)
\end{array}
\]

\( (1,2) \in E_1 \), but \( (b,e) \notin E_2 \)
\( (2,1) \in E_1 \), but \( (e,b) \notin E_2 \)
\( (3,2) \in E_1 \), but \( (c,b) \notin E_2 \)
\( (4,5) \in E_1 \), but \( (a,d) \notin E_2 \)
\( (2,5) \in E_1 \), but \( (e,d) \notin E_2 \)
\( (3,4) \in E_1 \), but \( (c,a) \notin E_2 \)

The mapping \( h \) has 2 errors.

Error of a Mapping

Intuitively, the error of mapping \( h \) tells us
- how many edges of \( G_1 \) have no corresponding edge in \( G_2 \) and
- how many edges of \( G_2 \) have no corresponding edge in \( G_1 \).

Let \( G_1 = (V_1,E_1) \) and \( G_2 = (V_2,E_2) \), and let \( h: V_1 \to V_2 \) be a 1-1, onto mapping.

**Forward error**

\[
E_F(h) = |\{(v_i,v_j) \in E_1 | (h(v_i),h(v_j)) \notin E_2\}|
\]

**Backward error**

\[
E_B(h) = |\{(v_i,v_j) \in E_2 | (h(v_i),h(v_j)) \notin E_1\}|
\]

**Total error**

\[
\text{Error}(h) = E_F(h) + E_B(h)
\]

**Relational distance**

\[
GD(G_1,G_2) = \min_{h \in H} \text{Error}(h)
\]

for all \( h \in H \), where \( H \) is the set of all 1-1, onto mappings \( h: V_1 \to V_2 \).
Variations of Relational Distance

- normalized relational distance: Divide by the sum of the number of edges in $E_1$ and those in $E_2$.
- undirected graphs: Just modify the definitions of $EF$ and $EB$ to accommodate.
- one way mappings: $h$ is 1-1, but need not be onto. Only the forward error $EF$ is used.
- labeled graphs: When nodes and edges can have labels, each node should be mapped to a node with the same label, and each edge should be mapped to an edge with the same label.

Graph Matching Algorithms

- graph isomorphism
- subgraph isomorphism
- relational distance
- attributed relational distance (uses labels)

Subgraph Isomorphism

Given model graph $M = ( VM, EM )$

data graph $D = ( VD, ED )$

Find 1-1 mapping $h: VM \rightarrow VD$
satisfying $(vi,vj) \in EM \Rightarrow (h(vi),h(vj)) \in ED$

Method: Backtracking Tree Search

![Diagram of tree search for subgraph isomorphism in digraphs]

```
procedure Treesearch(VM, VD, EM, ED, h)
{ v = first(VM);
  for each w ∈ VD
  { h' = h ∪ {(v,w)};
    OK = true;
    for each edge (vi,vj) in EM satisfying that
      either 1. vi = v and vj ∈ domain(h')
            or 2. vj = v and vi ∈ domain(h')
      if ( (h'(vi),h'(vj)) ∉ ED )
        {OK = false; break;};
    if OK
      { VM' = VM – v;
        VD' = VD – w;
        if (isempty(VM')) output(h');
        else Treesearch(VM', VD', EM, ED, h')
      };
  }
}
```

Branch-and-Bound Tree Search

Keep track of the least-error mapping.

![Diagram of branch-and-bound tree search for subgraph isomorphism in digraphs]