Directed Graphs

Depth first search
Transitive closure
Topological sort
PERT/CPM


Digraph. Directed graph.
- Edge from v to w.
- One-way street.
- Hyperlink from Yahoo to Princeton.

Graph Applications

<table>
<thead>
<tr>
<th>Graph</th>
<th>Vertices</th>
<th>Edges</th>
</tr>
</thead>
<tbody>
<tr>
<td>communication</td>
<td>telephones, computers</td>
<td>fiber optic cables</td>
</tr>
<tr>
<td>circuits</td>
<td>gates, registers, processors</td>
<td>wires</td>
</tr>
<tr>
<td>mechanical</td>
<td>joints</td>
<td>rods, beams, springs</td>
</tr>
<tr>
<td>hydraulic</td>
<td>reservoirs, pumping stations</td>
<td>pipelines</td>
</tr>
<tr>
<td>financial</td>
<td>stocks, currency</td>
<td>transactions</td>
</tr>
<tr>
<td>transportation</td>
<td>street intersections, airports</td>
<td>highways, airway routes</td>
</tr>
<tr>
<td>scheduling</td>
<td>tasks</td>
<td>precedence constraints</td>
</tr>
<tr>
<td>software systems</td>
<td>functions</td>
<td>function calls</td>
</tr>
<tr>
<td>internet</td>
<td>web pages</td>
<td>hyperlinks</td>
</tr>
<tr>
<td>games</td>
<td>board positions</td>
<td>legal moves</td>
</tr>
<tr>
<td>social relationship</td>
<td>people, actors</td>
<td>friendships, movie casts</td>
</tr>
<tr>
<td>neural networks</td>
<td>neurons</td>
<td>synapses</td>
</tr>
<tr>
<td>protein networks</td>
<td>proteins</td>
<td>protein-protein interactions</td>
</tr>
<tr>
<td>chemical compounds</td>
<td>molecules</td>
<td>bonds</td>
</tr>
</tbody>
</table>

A Few Directed Graph Problems

Transitive closure. Is there a directed path from v to w?

Topological sort. Can you draw the graph so that all of the edges point from left to right?

PERT/CPM. Given a set of tasks with precedence constraints, what is the earliest we can complete each task?

Pagerank. What is the importance of a web page?

Strong connectivity. Are all vertices mutually reachable?

Shortest path. Given a weighted graph, find best route from v to w?
Digraph ADT in Java

Typical client program.
- Create a Digraph.
- Pass the Digraph to a graph processing routine, e.g., DFSearcher.
- Query the graph processing routine for information.

```java
public static void main(String args[]) {
    int V = Integer.parseInt(args[0]);
    int E = Integer.parseInt(args[1]);
    Digraph G = new Digraph(V, E);
    System.out.println(G);
    DFSearcher dfs = new DFSearcher(G);
    System.out.println("Components = " + dfs.components());
}
```

calculate number of strongly connected components

Set of edges representation.

Directed Graph Representation

Vertex names. A B C D E F G H I J K L M
- This lecture: use integers between 0 and V-1.
- Real world: convert between names and integers with symbol table.

Orientation of edge matters.

Adjacency Matrix Representation

Adjacency matrix representation.
- Two-dimensional V x V boolean array.
- Edge v-w in graph: adj[v][w] = true.

```
<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
<th>G</th>
<th>H</th>
<th>I</th>
<th>J</th>
<th>K</th>
<th>L</th>
<th>M</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>B</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>C</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>D</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>E</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>F</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>G</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>H</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>I</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>J</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>K</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>L</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>M</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>
```

Adjacency Matrix: Java Implementation

Same as for undirected graphs, but only insert one copy of each edge.

```java
public class Digraph {
    private int V; // number of vertices
    private int E; // number of edges
    private boolean[][] adj; // adjacency matrix

    // empty graph with V vertices
    public Digraph(int V) {
        this.V = V;
        this.E = 0;
        this.adj = new boolean[V][V];
    }

    // insert edge v-w if it doesn't already exist
    public void insert(int v, int w) {
        if (!adj[v][w]) E++;
        adj[v][w] = true;
    }
}
```
Adjacency List Representation

Vertex indexed array of lists.
- Space proportional to number of edges.
- One representation of each directed edge.

A: F
B: C
C: G
D: E
E: D
F: E
G: D
H: I
I: J
J: K
K: L
L: M
M: adjacency list

Depth First Search

Transitive closure. Is there a directed path from \(v\) to \(w\)?

Use DFS to calculate all nodes reachable from \(v\).

To visit a node \(v\):
- mark it as visited
- recursively visit all unmarked nodes \(w\) adjacent to \(v\)

Enables direct solution of simple graph problems.
- Transitive closure.
- Directed cycles.
- Topological sort.

Basis for solving difficult graph problems.
- Strong connected components.
- Directed Euler path.

Transitive Closure: Java Implementation

```java
public class TransitiveClosure {
    private Digraph G;
    private boolean[][] tc;

    public TransitiveClosure(Digraph G) {
        this.G = G;
        this.tc = new boolean[G.V()][G.V()];
        for (int v = 0; v < G.V(); v++)
            dfs(v, v);
    }

    private void dfs(int s, int v) {
        tc[s][v]= true;
        IntIterator i = G.neighbors(v);
        while (i.hasNext()) {
            int w = i.next();
            if (!tc[s][w]) dfs(s, w);
        }
    }

    public boolean reachable(int v, int w) { return tc[v][w]; }
}
```

Transitive Closure: Cost Summary

Transitive closure. Is there a directed path from \( v \) to \( w \)?

<table>
<thead>
<tr>
<th>Method</th>
<th>Preprocess</th>
<th>Query</th>
<th>Space</th>
</tr>
</thead>
<tbody>
<tr>
<td>DFS (preprocess)</td>
<td>( E )</td>
<td>( V^2 )</td>
<td>( E )</td>
</tr>
<tr>
<td>DFS (online)</td>
<td>( 1 )</td>
<td>( E + V )</td>
<td>( E )</td>
</tr>
</tbody>
</table>

Open research problem. \( O(1) \) query, \( O(V^2) \) preprocessing time.

Application: Scheduling

Given a set of tasks to be completed with precedence constraints, in what order should we schedule the tasks?

- Task 0: read programming assignment.
- Task 1: download files.
- Task 2: write code.
- ....
- Task 12: sleep.

Graph model.
- Create a vertex \( v \) for each task.
- Create an edge \( v \rightarrow w \) if task \( v \) must precede task \( w \).

Directed Acyclic Graph

DAG: directed acyclic graph.

Topological sort: all edges point left to right.

Topological Sort with DFS: Java Implementation

Topologically sort a DAG. What if input graph is not a DAG?

```java
public class TopologicalSorter {
    ...
    public TopologicalSorter(Digraph G) {
        ...
        this.cnt = G.V();
        for (int v = 0; v < G.V(); v++)
            if (!visited[v]) dfs(v);
    }
    private void dfs(int v) {
        visited[v] = true;
        IntIterator i = G.neighbors(v);
        while (i.hasNext()) {
            int w = i.next();
            if (!visited[w]) dfs(w);
        }
        ts[--cnt] = v; \( \Phi \) assign numbers in reverse DFS postorder
    }
}
```
**Application: PERT/CPM**

Program Evaluation and Review Technique / Critical Path Method.

- Task \(v\) requires \(\text{time}[v]\) units of processing time.
- Can work on jobs in parallel subject to precedence constraints:
  - must finish task \(v\) before beginning \(w\)
- What’s the earliest we can complete each task?

<table>
<thead>
<tr>
<th>Index</th>
<th>Task</th>
<th>Duration</th>
<th>Prereq</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>Begin</td>
<td>0</td>
<td>-</td>
</tr>
<tr>
<td>B</td>
<td>Framing</td>
<td>4</td>
<td>A</td>
</tr>
<tr>
<td>C</td>
<td>Roofing</td>
<td>2</td>
<td>B</td>
</tr>
<tr>
<td>D</td>
<td>Siding</td>
<td>6</td>
<td>B</td>
</tr>
<tr>
<td>E</td>
<td>Windows</td>
<td>5</td>
<td>D</td>
</tr>
<tr>
<td>F</td>
<td>Plumbing</td>
<td>3</td>
<td>D</td>
</tr>
<tr>
<td>G</td>
<td>Electricity</td>
<td>4</td>
<td>C, E</td>
</tr>
<tr>
<td>H</td>
<td>Paint</td>
<td>6</td>
<td>C, E</td>
</tr>
<tr>
<td>I</td>
<td>Finish</td>
<td>0</td>
<td>F, H</td>
</tr>
</tbody>
</table>

**Longest Path in DAG**

Longest path algorithm in DAG.

- Compute topological order of vertices.
- Initialize \(\text{fin}[v] = 0\) for all vertices \(v\).
- Consider vertices \(v\) in topological order:
  - for each edge \(v\)-\(w\), set
    \[
    \text{fin}[w] = \max(\text{fin}[w], \text{fin}[v] + \text{time}[w])
    \]

In general graphs, longest path problem is \(\mathsf{NP}\)-hard.

**Application: Web Crawler**

Goal. Crawl Internet and visit every page.

Solution. BFS with implicit graph.

Vertices are websites instead of integers.

- Use string to represent vertex.
- Use symbol table \(\text{visited}\) to mark website already visited.

Directed edges from website \(v\) are URLs that appear in page \(v\).

- Use regular expression to find patterns like \(http://xxx.yyy.zzz\).
- Add newly discovered webpages to Queue of strings.

**Web Crawler: Java Implementation**

```java
Queue q = new Queue(); // queue of sites to crawl
HashSet visited = new HashSet(); // ST of visited websites
q.enqueue(s); // start crawl from site s
visited.add(s);
while (!q.isEmpty()) {
    String v = (String) q.dequeue(); // read in raw html
    System.out.println(v);
    In in = new In(v);
    String input = in.readAll(); // http://xxx.yyy.zzz
    String regexp = "http://(\w+\.)*(\w+)";
    Pattern pattern = Pattern.compile(regexp);
    Matcher matcher = pattern.matcher(input);
    while (matcher.find()) {
        String w = matcher.group(); // search using regular expression
        if (!visited.contains(w)) {
            visited.add(w); // if unvisited, mark as visited
            q.enqueue(w); // and put on queue
        }
    }
}
```
Application: Google’s PageRank Algorithm

Goal. Determine which web pages on Internet are important.
Solution. Ignore keywords and content, focus on hyperlink structure.

Random surfer model.
- Start at random page.
- With probability 0.85, randomly select a hyperlink to visit next;
  with probability 0.15, randomly select any page.
- Never hit "Back" button.
- PageRank = proportion of time random surfer spends on each page.

Intuition.
- Each page evenly distributes its rank to all pages that it points to.
- Each page receives rank from all pages that point to it.
- "Hard" to cheat.

Solution 1: Simulate random surfer for a long time.
Solution 2: Compute ranks directly until they converge.
Solution 3: Compute eigenvalues of adjacency matrix!

```
for (i = 0; i < PHASES; i++) {
    for (int v = 0; v < G.V(); v++) oldrank[v] = rank[v];
    for (int v = 0; v < G.V(); v++) rank[v] = 0;
    for (int v = 0; v < G.V(); v++) {
        IntIterator i = G.neighbors(v);
        while (i.hasNext()) {
            int w = i.next();
            rank[w] += 1.0 * oldrank[v] / outdegree[v];
        }
    }
}
```

PageRank Caveats

Dead end: page with no outgoing links.
- All importance will leak out of web.
- Easy to detect and ignore.

Spider trap: group of pages with no links leaving the group.
- Group will accumulate all importance of Web.
- Compute strongly connected components.
  - use transitive closure - \(O(E V)\) time
  - ingenious algorithms using DFS - \(O(E + V)\) time

Strongly Connected Components

Kosaraju’s algorithm.
- Run DFS on reverse digraph and compute postorder.
- Run DFS on original digraph. In search loop that calls dfs, consider vertices in reverse postorder.

Theorem. Trees in second DFS are strong components. (!)