Directed Graphs

Depth first search

Transitive closure

Topological sort

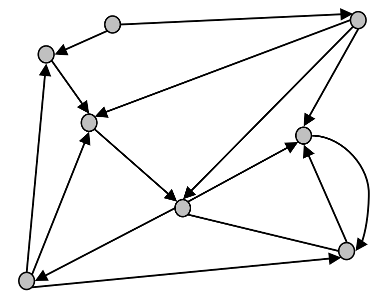
PERT/CPM

Reference: Chapter 19, Algorithms in Java, 3rd Edition, Robert Sedgewick.

Directed Graphs

Digraph. Directed graph.

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



Graph Applications

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

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.

```
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());
}
```

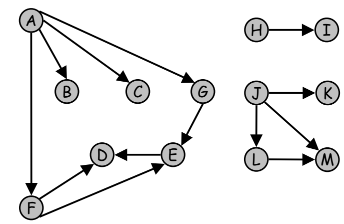
calculate number of strongly connected components

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.



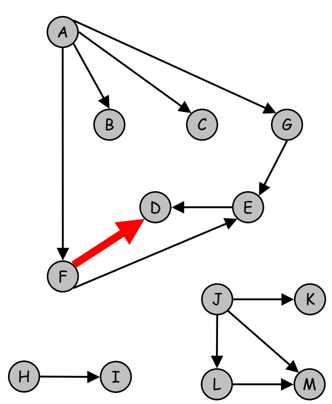
Set of edges representation.

- A-B A-G A-C L-M J-M J-L J-K E-D F-D H-I F-E A-F G-E

Adjacency Matrix Representation

Adjacency matrix representation.

- Two-dimensional `V × V` boolean array.
- Edge `v-w` in graph: `adj[v][w] = true`.



		A	B	C	D	E	F	G	H	I	J	K	L	M
0	A	0	1	1	0	0	1	1	0	0	0	0	0	0
1	B	0	0	0	0	0	0	0	0	0	0	0	0	0
2	C	0	0	0	0	0	0	0	0	0	0	0	0	0
3	D	0	0	0	0	0	0	0	0	0	0	0	0	0
4	E	0	0	0	1	0	0	0	0	0	0	0	0	0
5	F	0	0	0	1	1	0	0	0	0	0	0	0	0
6	G	1	1	0	0	1	0	0	0	0	0	0	0	0
7	H	0	0	0	0	0	0	0	0	1	0	0	0	0
8	I	0	0	0	0	0	0	0	0	0	0	0	0	0
9	J	0	0	0	0	0	0	0	0	0	0	1	1	1
10	K	0	0	0	0	0	0	0	0	0	0	0	0	0
11	L	0	0	0	0	0	0	0	0	0	0	0	0	1
12	M	0	0	0	0	0	0	0	0	0	1	0	1	0

adjacency matrix

Adjacency Matrix: Java Implementation

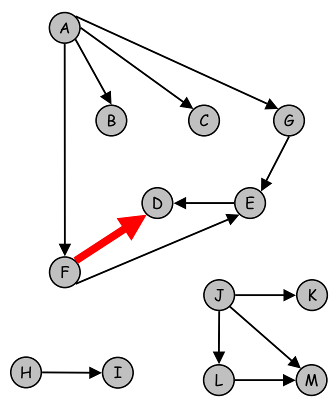
Same as for undirected graphs, but only insert one copy of each edge.

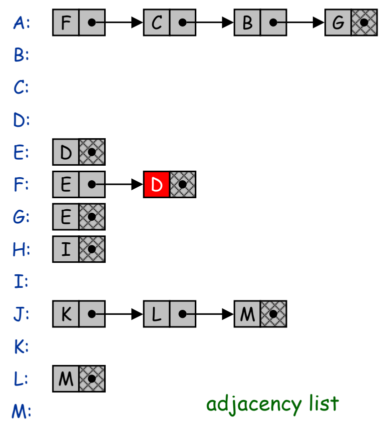
```
public class Digraph {
   private int V;             // number of vertices
   private int E;             // number of edges
   private boolean[][] adj;   // adjacency matrix

   // 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 representations of each directed edge.





adjacency list

Adjacency List: Java Implementation

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

```
public class Digraph {
   private int V;          // # vertices
   private int E;          // # edges
   private AdjList[] adj;  // adjacency lists

   public Digraph(int V) {
      this.V = V;
      this.E = 0;
      adj = new AdjList[V];
   }

   // insert edge v-w, parallel edges allowed
   public void insert(int v, int w) {
      adj[v] = new AdjList(w, adj[v]);
      E++;
   }
   ...
}
```


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

```
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]; }
```

run dfs from every vertex

reachability from s, made it to v

is w reachable from v?

Transitive Closure: Cost Summary

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

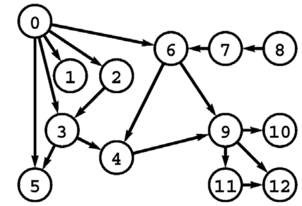
	Method	Preprocess	Query	Space
⇨	DFS (preprocess)	E V	1	V^2
	DFS (online)	1	E + V	E

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.

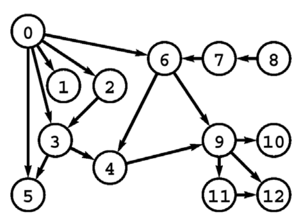


Graph model.

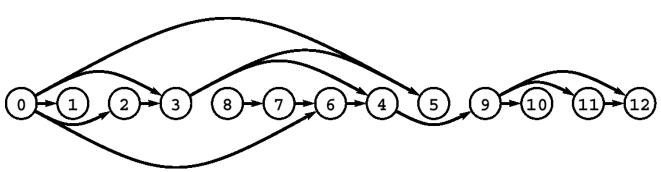
- Create a vertex `v` for each task.
- Create an edge `v-w` if task `v` must precede task `w`.

Directed Acyclic Graph

DAG: directed acyclic graph.



Topological sort: all edges point left to right.



	0	1	2	3	4	5	6	7	8	9	10	11	12
ts	0	1	2	3	8	7	6	4	5	9	10	11	12

Topological Sort with DFS: Java Implementation

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

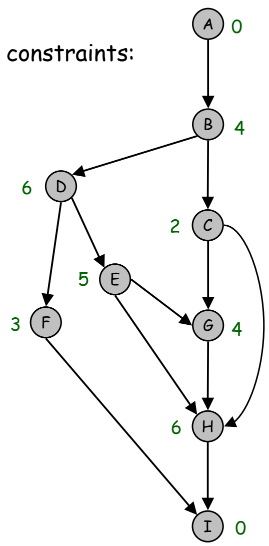
```
public class TopologicalSorter {
   ...
   public TopologicalSorter(Digraph G) {
      ...
      this.cnt = G.V();
      for (int v = 0; v < G.V(); v++)
         if (!visited[v]) dfs(v);
   }

   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;   ⇦ assign numbers in
                         reverse DFS postorder
   }
```

Application: PERT/CPM

Program Evaluation and Review Technique / Critical Path Method.

- Task `v` requires `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?

Index	Task	Duration	Prereq
A	Begin	0	-
B	Framing	4	A
C	Roofing	2	B
D	Siding	6	B
E	Windows	5	D
F	Plumbing	3	D
G	Electricity	4	C, E
H	Paint	6	C, E
I	Finish	0	F, H

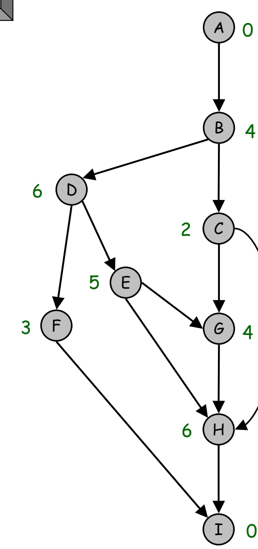


Longest Path in DAG

Longest path algorithm in DAG.

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

In general graphs, longest path problem is 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 `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

```
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();
   System.out.println(v);
   In in = new In(v);                     read in raw html
   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);
            q.enqueue(w);                 if unvisited, mark as visited
                                          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.

Application: Google's PageRank Algorithm

Solution 1: Simulate random surfer for a long time.

Solution 2: Compute ranks directly until they converge.

```
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];
      }
   }
}
```

Solution 3: Compute eigenvalues of adjacency matrix!

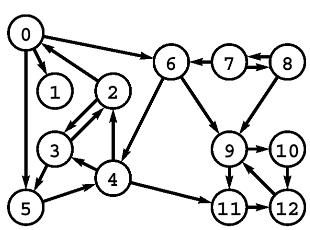
PageRank Caveats

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

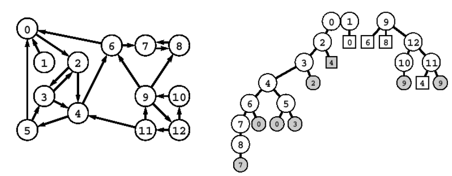


	0	1	2	3	4	5	6	7	8	9	10	11	12
sc	2	1	2	2	2	2	2	3	3	0	0	0	0

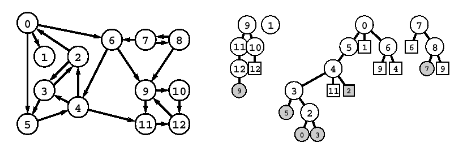
Strongly Connected Components

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.



	0	1	2	3	4	5	6	7	8	9	10	11	12
post	8	7	6	5	4	3	2	0	1	11	10	12	9



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