CSE 101, Winter 2018

Design and Analysis of Algorithms

Lecture 2: Graphs, DFS (Undirected, Directed), DAGs

Class URL: http://vlsicad.ucsd.edu/courses/cse101-w18/
Graphs

Internet topology

Country Code: from mask
- DE
- IT
- JP
- Other
- SE
- UK
- US
Graphs

Gene-gene interactions affecting phenotype in yeast
Graphs

Airline routes

System dependence graph
Graph Representation

- Graph G = (V,E)
  - Vertex set V = V(G)
  - Edge set E = E(G) (connecting pairs of vertices)

How is this stored in a computer?
Graph Representation

- Graph $G = (V,E)$
  - Vertex set $V = V(G)$
  - Edge set $E = E(G)$ (connecting pairs of vertices)

- Adjacency Matrix

- Adjacency List

- Adjacency Matrix

$O(|V| + |E|)$ storage needed

$O(|V|^2)$ storage needed
Depth-First Search

- **Depth-first search** is a method for exploring a graph
  - Explore “deeper” in the graph whenever possible
  - Edges are explored out of the most recently discovered vertex \( v \) that still has unexplored edges
  - When all of \( v \)’s edges have been explored, backtrack to the vertex from which \( v \) was discovered
- Analogy: Exploring a maze
  - Mark with chalk: has the vertex been visited already
  - Unwind / rewind ball of string: stack push / pop
  - Pseudocode in book: stack is implicit from recursion
Depth-First Search: Pseudocode

Figure 3.3 Finding all nodes reachable from a particular node.

procedure explore(G,v)
Input: G = (V,E); v ∈ V
Output: visited(u) set to true for all nodes u reachable from v

visited(v) = true
previsit(v)  // pre[v] = clock; clock++;
for each edge (v,u) in E:
    if not visited(u):  explore(u)
postvisit(v)   // post[v] = clock; clock++;
Depth-First Search: Pseudocode

Figure 3.3 Finding all nodes reachable from a particular node.

procedure explore(G,v)
Input: G = (V,E); v ∈ V
Output: visited(u) set to true for all nodes u reachable from v
visited(v) = true
previsit(v) // pre[v] = clock; clock++; ccnum[v] = cc
for each edge (v,u) in E:
    if not visited(u): explore(u)
postvisit(v) // post[v] = clock; clock++;

Figure 3.5 Depth-first search.

procedure dfs (G) // clock = 1; cc = 0;
for all v ∈ V: visited(v) = false
for all v ∈ V: if not visited(v): cc++; explore(v)
**Depth-First Search: Example (Fig. 3.6)**

Tree edge: when encountering a new vertex

Back edge: when encountering a previously-seen vertex (i.e., from descendant to ancestor)
Depth-First Search: Notes

- pre[v] is the step when a vertex is first touched, and post[v] is the step when a vertex is last touched
  - Correspond to preorder and postorder listings of nodes
  - Preorder of CC1: A-B-E-I-J; Postorder of CC1: B-J-I-E-A

- A connected component is a maximal subgraph that is connected but has no edges to vertices outside the subgraph
  - cc counts connected components.

- DFS runtime is $O(|V| + |E|)$
  - $O(|V|)$ for marking each node as visited, pre/post-visit operations
  - $O(|E|)$ total work for traversing edges: each edge $(x,y) \in E$ is examined exactly twice, once during explore($x$) and once during explore($y$)

- QUESTION: Does DFS correctly reach all reachable v’s?
Depth-First Search: Example (Fig. 3.6)

CONNECTED COMPONENTS
- Linear-time CC finding
- Traversed edges in a CC form a tree
- Trees together constitute a forest

Tree edge: when encountering a new vertex
Back edge: when encountering a previously-seen vertex (i.e., from descendant to ancestor)
Depth-First Search: Example (Fig. 3.6)

PRE- and POST-ORDERING

- For all u,v the intervals \([\text{pre}(u), \text{post}(u)]\) and \([\text{pre}(v), \text{post}(v)]\) are either disjoint or nested.
- \([\text{pre}(u), \text{post}(u)]\) is essentially the time interval during which node u is on the stack (and, stack has LIFO behavior).

Tree edge: when encountering a new vertex

Back edge: when encountering a previously-seen vertex (i.e., from descendant to ancestor)
DFS in **DIRECTED** Graphs

**TERMINOLOGY**

- A is *root* of tree; all other nodes are A’s *descendants*
- E has *descendants* F, G, H  (E is an *ancestor* of G)
- C is the *parent* of D
- H is a *child* of E
More Edge Types

**Figure 3.7 DFS on a directed graph.**

- **Tree** edges = part of the DFS forest
- **Forward** edges = from a node to a non-child descendant
- **Back** edges = to an ancestor in the tree
- **Cross** edges = to neither descendant nor ancestor

**EDGE TYPES**
Ancestry and Pre/Post Numbers

- Node $u$ is an ancestor of node $v$ iff $\text{pre}(u) < \text{pre}(v) < \text{post}(u)$
- *Because* $u$ is an ancestor of $v$ *iff* $u$ is discovered first, and then $v$ is encountered during the exploration of $u$
- [Def. Node $v$ is a *descendant* of $u$ *iff* node $u$ is an *ancestor* of $v*$]
# Edge Types by Pre/Post Numbers

<table>
<thead>
<tr>
<th>Edge Type</th>
<th>Pre/Post Properties of (u,v)</th>
<th>In Search Tree?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tree</td>
<td>pre(u) &lt; pre(v) &lt; post(v) &lt; post(u)</td>
<td>Yes</td>
</tr>
<tr>
<td>Forward</td>
<td>pre(u) &lt; pre(v) &lt; post(v) &lt; post(u)</td>
<td>No</td>
</tr>
<tr>
<td>Back</td>
<td>pre(v) &lt; pre(u) &lt; post(u) &lt; post(v)</td>
<td>No</td>
</tr>
<tr>
<td>Cross</td>
<td>pre(v) &lt; post(v) &lt; pre(u) &lt; post(u)</td>
<td>No</td>
</tr>
</tbody>
</table>

**Every edge must fall into one of these groups**
Cycles in Directed Graphs

- Definition: A cycle in a directed graph is a circular path 
  \( v_0 \rightarrow v_1 \rightarrow v_2 \rightarrow \ldots \rightarrow v_k \rightarrow v_0 \)
- If a graph has no cycles, then it is an acyclic graph

- **A directed graph has a cycle iff DFS reveals a back edge**
  - \((\leftarrow)\) If \((u,v)\) is a back edge, then it along with the \(v \rightarrow u\) path in the search tree will form a cycle.
  - \((\rightarrow)\) If the graph has a cycle 
    \( v_0 \rightarrow v_1 \rightarrow v_2 \rightarrow \ldots \rightarrow v_k \rightarrow v_0 \) 
    then consider the node with smallest \(pre\) number, call it \(v_i\). 
    All other \(v_j\) on the cycle are reachable from \(v_i\) and will therefore be descendants of \(v_i\) in the search tree. Thus, the edge \(v_{i-1} \rightarrow v_i\) is a back edge.

- So, we can determine whether \(G\) is acyclic in linear time
Topological Ordering and DAGs

- Prelude to shortest paths
- Generic scheduling problem
- Input:
  - Set of tasks \( \{T_1, T_2, T_3, \ldots, T_n\} \)
    - Example: getting dressed in the morning: put on shoes, socks, shirt, pants, belt, …
  - Set of dependencies \( \{T_1 \rightarrow T_2, T_3 \rightarrow T_4, T_5 \rightarrow T_1, \ldots\} \)
    - Example: must put on socks before shoes, pants before belt, …
- Want:
  - ordering of tasks which is consistent with dependencies
- Problem representation: Directed Acyclic Graph
  - Vertices = tasks; Directed Edges = dependencies
  - Acyclic: if \( \exists \) cycle of dependencies, no solution possible
  - General model for causality, dependency
Topological Ordering

- **TOP_ORDER PROBLEM:** Given a DAG $G = (V,E)$ with $|V|=n$, assign labels $1,...,n$ to $v_i \in V$ s.t. if $v$ has label $k$, all vertices reachable from $v$ have labels $> k$

Every edge is from a lower label to a higher label
Topological Ordering

• **TOP_ORDER PROBLEM**: Given a DAG $G = (V,E)$ with $|V| = n$, assign labels $1,...,n$ to $v_i \in V$ s.t. if $v$ has label $k$, all vertices reachable from $v$ have labels $> k$

*Every edge is from a lower label to a higher label*

• “Induction idea”:
  – Know how to label DAG’s with $< n$ vertices

• Claim: A DAG $G$ always has some vertex with indegree $= 0$
  – Take an arbitrary vertex $v$. If $v$ doesn’t have indegree $= 0$, traverse any incoming edge to reach a predecessor of $v$. If this vertex doesn’t have indegree $= 0$, traverse any incoming edge to reach a predecessor, etc.
  – Eventually, this process will either identify a vertex with indegree $= 0$, or else reach a vertex that has been reached previously (a contradiction, given that $G$ is acyclic).

• “Inductive approach”:
  – Find $v$ with indegree$(v) = 0$, give it lowest available label, then delete $v$ (and incident edges), update degrees of remaining vertices, and repeat
Topological Ordering

• TOP_ORDER PROBLEM: Given a DAG $G = (V,E)$ with $|V|=n$, assign labels $1,...,n$ to $v_i \in V$ s.t. if $v$ has label $k$, all vertices reachable from $v$ have labels $> k$

Every edge is from a lower label to a higher label

• Solution: Run DFS, then perform tasks in decreasing order of post numbers

• Fact: In a DAG, every edge leads to a vertex with lower post number.

• Why? Any edge $(u,v)$ for which $\text{post}(v) > \text{post}(u)$ is a back edge. But a DAG, being acyclic, has no back edges.
Sources and Sinks

• In a DAG, the node with smallest post number has no outgoing edges
  – Such a node is called a sink

• In a DAG, the node with largest post number has no incoming edges
  – Such a node is called a source
Administrative Notes, January 11

• You are strongly urged to attend at least one discussion section per week.
  – On exams, you are responsible for material that is (i) presented by TAs and (ii) posted on the class website as Discussion Notes

• Reminder
  – HW #1 is due next Friday, January 19, on Gradescope
    • Deadline =11:59pm Pacific Time
    • HW #0, PA #0 pipecleaners are due by tomorrow, 11:59pm PT
    • HW review ression tomorrow, Friday 5pm in WLH 2001
  – PA #1 (on graph traversal) due on Friday, January 26 at 11:59pm PT. Next Friday, January 19, 5pm: PA review session
  – Podcasting for Friday review sessions has been requested from ACMS
EXTRA MATERIAL 1: Triangle-Finding by Matrix Multiplication?

Definition of “EXTRA MATERIAL”:

“Of current or future relevance and interest, but unless included in a lecture you are not responsible for it on exams”
Finding Triangles

• Given a graph $G = (V,E)$, count the number of triangles in $G$

• Vertices $v_i, v_j, v_k \in V$ form a triangle iff edges $(v_i,v_j), (v_j,v_k), (v_i,v_k) \in E$

• QUESTION: What is an $O(n^3)$ algorithm?

• QUESTION: Would the algorithm complexity be different if I asked you to “… list all triangles in $G$”? Or, if I asked you to “… determine whether there is a triangle in $G$”?

Worst-case complexity can depend on size of output
A Property of the Adjacency Matrix

- Let $A$ = the adjacency matrix of $G$
- What is the meaning of $A_{ij}^2$?

$A_{ij}^2 = \#$ of length-2 paths from $v_i$ to $v_j$ in $G$

$A_{ii}^2 = \text{deg}(v_i)$

\[
\begin{array}{cc}
A & A^2 \\
\begin{array}{cccccc}
1 & 2 & 3 & 4 & 5 \\
1 & 0 & 0 & 1 & 1 & 0 \\
2 & 0 & 0 & 1 & 0 & 1 \\
3 & 1 & 1 & 0 & 0 & 1 \\
4 & 1 & 0 & 0 & 0 & 1 \\
5 & 0 & 1 & 1 & 1 & 0 \\
\end{array} & \begin{array}{cccccc}
1 & 2 & 3 & 4 & 5 \\
1 & 0 & 0 & 1 & 1 & 0 \\
2 & 0 & 0 & 1 & 0 & 1 \\
3 & 1 & 1 & 0 & 0 & 1 \\
4 & 1 & 0 & 0 & 0 & 1 \\
5 & 0 & 1 & 1 & 1 & 0 \\
\end{array} & \begin{array}{cccccc}
1 & 2 & 3 & 4 & 5 \\
1 & 2 & 1 & 0 & 0 & 2 \\
2 & 1 & 2 & 1 & 1 & 1 \\
3 & 0 & 1 & 3 & 2 & 1 \\
4 & 0 & 1 & 2 & 2 & 0 \\
5 & 2 & 1 & 1 & 0 & 3 \\
\end{array}
\end{array}
\]

\[\textbf{X} = \textbf{A}^2\]
Finding Triangles

- Let $A$ = the adjacency matrix of $G$
- **We can use $A$, $A^2$ to find triangles in $G$!**
  (How much time does it take to compute $A^2$?)

```
A =
1  2  3  4  5
1  0  0  1  1  0
2  0  0  1  0  1
3  1  1  0  0  1
4  1  0  0  0  1
5  0  1  1  1  0

A^2 =
1  2  3  4  5
1  0  0  1  1  0
2  0  0  1  0  1
3  1  1  0  0  1
4  1  0  0  0  1
5  0  1  1  1  0
```

(X = A × A)

- A × A = $A^2$
What’s Different Here?

• Graph $G = (V,E)$
  – Vertex set $V = V(G)$
  – Edge set $E = E(G)$ (connecting pairs of vertices)

• Adjacency List

• Adjacency Matrix

Directed Graph

1 2 3 4 5
1 0 0 1 1 0
2 0 0 1 0 0
3 0 0 0 0 1
4 0 0 0 0 1
5 0 1 0 0 0
EXTRA MATERIAL 2: Illustration of the Celebrity Problem (see Lecture 1, Slide 32)
The Celebrity Problem

- Finding a Celebrity. Given a set S of N people, assume that for any pair I, J exactly one of the following is true: I “knows” J, or J “knows” I. Further, define a “celebrity” as someone who knows no one (and who is therefore known by everyone else). Given the “knows” relation over S, determine whether S contains a celebrity.

- Celebrity Problem: directed graph whose adjacency matrix has for each i,j pair either (A_{ij} = 0, A_{ji} = 1 (=J knows I)) or (A_{ij} = 1, A_{ji} = 0 (= I knows J))

Adjacency Matrix of a Directed Graph

\[
\begin{array}{ccccc}
1 & 2 & 3 & 4 & 5 \\
1 & - & 0 & 1 & 1 & 0 \\
2 & 1 & - & 1 & 1 & 0 \\
3 & 0 & 0 & - & 1 & 1 \\
4 & 0 & 0 & 0 & - & 0 \\
5 & 1 & 1 & 0 & 1 & - \\
\end{array}
\]

(⇒ Person #4 is a Celebrity)

Question: What is a practical application for this problem?

Can you determine existence of a celebrity in \(O(n^2)\) time? \(O(n)\)?
EXTRA MATERIAL 3: DFS in a Minimum Spanning Tree == A Heuristic for the Geometric Traveling Salesperson Problem
Side Note: Concept of “Approximation” Algorithm

- **Approximation algorithm**: An algorithm that returns near-optimal solutions (i.e. is "provably good") is called an *approximation algorithm*.

- **Performance Ratio (Ratio Bound)**: We say that an approximation algorithm for the problem has a *ratio bound* of $\rho(n)$ if for any instance $I$ of size $n$, the cost $C$ of the solution produced by the approximation algorithm is within a factor of $\rho(n)$ of the cost $C^*$ of an optimal solution:

  $$\max_{|I|=n} \frac{C}{C^*} \leq \rho(n)$$
Approximation Algorithm : Euclidean TSP

• Euclidean Traveling Salesperson Problem: Let $C_1, C_2, \ldots, C_n$ be a set of points in the plane corresponding to the location of $n$ cities. Find a minimum-distance Hamiltonian cycle (traveling salesman tour) among them.

• TSP is a hard problem, but there is a minimum spanning tree (MST) based approximation with performance ratio $= 2$
  – Fact: $\text{cost(MST)} < \text{cost(Tour}_\text{opt})$
  – MST is the minimum-cost graph that connects all vertices, and has only $n-1$ edges
  – Any TSP tour must also connect all vertices, and will have $n$ edges

  • Notice that a tour can be viewed as a spanning tree (that happens to be a chain) plus another edge
Approximation Algorithm: Euclidean TSP

– Idea: Consider the circuit that consists of a DFS traversal of MST (starting from any city), and includes an edge in the opposite direction whenever the search backtracks. And then we can take shortcut on the tour we get. Next slide is an example: DFS traversal starting from city a produces a circuit a-b-c-b-a-d… We can then use a shortcut c-d to replace original path c-b-a-d.

– Note: Being in a metric space (Euclidean is just one possibility) means that the triangle inequality holds, which means that the shortcuts reduce tour cost.
Approximation Algorithm: Euclidean TSP

DFS traversal of MST

Taking shortcut from DFS tour. (e.g. replacing a-b-c-b-a-d, by a-b-c-d)

\[ \text{Tour}_{\text{Heur}} \leq 2 \times \text{MST} \leq 2 \times \text{Tour}_{\text{opt}} \]