Directed Graph Algorithms

CSE 373
Data Structures
Lecture 14
Readings

• Reading
  › Sections 9.2, 9.3 and 10.3.4
Topological Sort

Problem: Find an order in which all these courses can be taken.

Example: 142 → 143 → 378
→ 370 → 321 → 341 → 322
→ 326 → 421 → 401

In order to take a course, you must take all of its prerequisites first.
Topological Sort

Given a digraph $G = (V, E)$, find a linear ordering of its vertices such that:

for any edge $(v, w)$ in $E$, $v$ precedes $w$ in the ordering
Topo sort - good example

Any linear ordering in which all the arrows go to the right is a valid solution.

Note that F can go anywhere in this list because it is not connected. Also the solution is not unique.
Topo sort - bad example

Any linear ordering in which an arrow goes to the left is not a valid solution

NO!
Paths and Cycles

- Given a digraph \( G = (V,E) \), a path is a sequence of vertices \( v_1, v_2, \ldots, v_k \) such that:
  - \((v_i, v_{i+1})\) in \( E \) for \( 1 \leq i < k \)
  - path length = number of edges in the path
  - path cost = sum of costs of each edge

- A path is a cycle if:
  - \( k > 1; v_1 = v_k \)

- \( G \) is acyclic if it has no cycles.
Only acyclic graphs can be topo. sorted

- A directed graph with a cycle cannot be topologically sorted.
Step 1: Identify vertices that have no incoming edges
  • The “in-degree” of these vertices is zero
Step 1: Identify vertices that have no incoming edges
- If no such vertices, graph has only cycle(s) (cyclic graph)
- Topological sort not possible – Halt.

Example of a cyclic graph
Topo sort algorithm - 1b

Step 1: Identify vertices that have no incoming edges
  • Select one such vertex

Select
Step 2: Delete this vertex of in-degree 0 and all its outgoing edges from the graph. Place it in the output.
Continue until done

Repeat Step 1 and Step 2 until graph is empty
Select B. Copy to sorted list. Delete B and its edges.
Select C. Copy to sorted list. Delete C and its edges.
Select D. Copy to sorted list. Delete D and its edges.
E, F

Select E. Copy to sorted list. Delete E and its edges.
Select F. Copy to sorted list. Delete F and its edges.
Done
Implementation

Assume adjacency list representation

Translation array

value next
Calculate In-degrees

In-Degree array; or add a field to array A
Calculate In-degrees

for i = 1 to n do D[i] := 0; endfor
for i = 1 to n do
    x := A[i];
    while x ≠ null do
        D[x.value] := D[x.value] + 1;
        x := x.next;
    endwhile
endfor
**Maintaining Degree 0 Vertices**

**Key idea:** Initialize and maintain a *queue (or stack)* of vertices with In-Degree 0

Queue: 1 6
Topo Sort using a Queue
(breadth-first)

After each vertex is output, when updating In-Degree array, enqueue any vertex whose In-Degree becomes zero
Topological Sort Algorithm

1. Store each vertex’s In-Degree in an array D
2. Initialize queue with all “in-degree=0” vertices
3. While there are vertices remaining in the queue:
   (a) Dequeue and output a vertex
   (b) Reduce In-Degree of all vertices adjacent to it by 1
   (c) Enqueue any of these vertices whose In-Degree became zero
4. If all vertices are output then success, otherwise there is a cycle.
Main Loop
while notEmpty(Q) do
  x := Dequeue(Q)
  Output(x)
  y := A[x];
  while y ≠ null do
    D[y.value] := D[y.value] – 1;
    if D[y.value] = 0 then Enqueue(Q,y.value);
    y := y.next;
  endwhile
endwhile
Topological Sort Analysis

- Initialize In-Degree array: $O(|V| + |E|)$
- Initialize Queue with In-Degree 0 vertices: $O(|V|)$
- Dequeue and output vertex:
  - $|V|$ vertices, each takes only $O(1)$ to dequeue and output: $O(|V|)$
- Reduce In-Degree of all vertices adjacent to a vertex and Enqueue any In-Degree 0 vertices:
  - $O(|E|)$
- For input graph $G=(V,E)$ run time = $O(|V| + |E|)$
  - Linear time!
Topo Sort using a Stack (depth-first)

After each vertex is output, when updating In-Degree array, *push any vertex whose In-Degree becomes zero*

Stack

Output

**Stack**

pop

Output

push

**Digraph**

1

2

3

4

5

6

**Array**

D

A

1

2

3

4

5

6

0

1

1

2

0

2

3

4

5

4

5

5

6
Recall Path cost, Path length

- Path cost: the sum of the costs of each edge
- Path length: the number of edges in the path
  - Path length is the unweighted path cost

\[ \text{length}(p) = 5 \]
\[ \text{cost}(p) = 11 \]
Shortest Path Problems

- Given a graph $G = (V, E)$ and a “source” vertex $s$ in $V$, find the minimum cost paths from $s$ to every vertex in $V$
- Many variations:
  - unweighted vs. weighted
  - cyclic vs. acyclic
  - pos. weights only vs. pos. and neg. weights
  - etc
Why study shortest path problems?

- Traveling on a budget: What is the cheapest airline schedule from Seattle to city X?
- Optimizing routing of packets on the internet:
  - Vertices are routers and edges are network links with different delays. What is the routing path with smallest total delay?
- Shipping: Find which highways and roads to take to minimize total delay due to traffic
- etc.
Unweighted Shortest Path

Problem: Given a “source” vertex $s$ in an unweighted directed graph $G = (V,E)$, find the shortest path from $s$ to all vertices in $G$

Only interested in path lengths
Breadth-First Search Solution

- Basic Idea: Starting at node s, find vertices that can be reached using 0, 1, 2, 3, ..., N-1 edges (works even for cyclic graphs!)
Breadth-First Search Alg.

- Uses a queue to track vertices that are “nearby”
- source vertex is $s$

\[
\text{Distance}[s] := 0 \\
\text{Enqueue}(Q,s); \quad \text{Mark}(s) // After a vertex is marked once \quad // it won’t be enqueued again
\]

while queue is not empty do

\[
X := \text{Dequeue}(Q); \\
\text{for each vertex } Y \text{ adjacent to } X \text{ do} \\
\text{if } Y \text{ is unmarked then} \\
\quad \text{Distance}[Y] := \text{Distance}[X] + 1; \\
\quad \text{Previous}[Y] := X; // if we want to record paths \\
\quad \text{Enqueue}(Q,Y); \quad \text{Mark}(Y);
\]

- **Running time** $= O(|V| + |E|)$
Example: Shortest Path length

Queue Q = C
Example (ct’d)

Queue $Q = \{A, D, E\}$

Indicates the vertex is marked

Previous pointer
Example (ct’d)

Q = D E B
Example (ct’d)

\[ Q = B \ G \]
Example (ct’d)

\[ Q = F \]
Example (ct’d)

Q = H
What if edges have weights?

- Breadth First Search does not work anymore
  - minimum cost path may have more edges than minimum length path

Shortest path (length) from C to A:
C → A (cost = 9)

Minimum Cost Path = C → E → D → A (cost = 8)
Dijkstra’s Algorithm for Weighted Shortest Path

- Classic algorithm for solving shortest path in weighted graphs (without negative weights)
- A greedy algorithm (irrevocably makes decisions without considering future consequences)
- Each vertex has a cost for path from initial vertex
Basic Idea of Dijkstra’s Algorithm

- Find the vertex with smallest cost that has not been “marked” yet.
- Mark it and compute the cost of its neighbors.
- Do this until all vertices are marked.
- Note that each step of the algorithm we are marking one vertex and we won’t change our decision: hence the term “greedy” algorithm