CSE 373

MAY 8TH – DIJKSTRAS
GRAPH REVIEW

- What is some of the terminology for graphs and what do those terms mean?
  - Vertices and Edges
  - Directed v. Undirected
  - In-degree and out-degree
  - Connected
  - Weighted v. unweighted
  - Cyclic v. acyclic
  - DAG: Directed Acyclic Graph
TRAVERSALS

• For an arbitrary graph and starting node $v$, find all nodes reachable from $v$.
  • There exists a path from $v$
  • Doing something or “processing” each node
  • Determines if an undirected graph is connected? If a traversal goes through all vertices, then it is connected

• Basic idea
  • Traverse through the nodes like a tree
  • Mark the nodes as visited to prevent cycles and from processing the same node twice
void traverseGraph(Node start) {
    Set pending = emptySet()
    pending.add(start)
    mark start as visited
    while (pending is not empty) {
        next = pending.remove()
        for each node u adjacent to next
            if (u is not marked visited) {
                mark u
                pending.add(u)
            }
    }
}
RUNTIME AND OPTIONS

• Assuming we can add and remove from our “pending” DS in O(1) time, the entire traversal is O(|E|)

• Our traversal order depends on what we use for our pending DS.
  • Stack : DFS
  • Queue: BFS

• These are the main traversal techniques in CS, but there are others!
EXAMPLE: TREES
A tree is a graph and make DFS and BFS are easier to “see”

DFS(Node start) {
    mark and process start
    for each node u adjacent to start
        if u is not marked
            DFS(u)
}

- A, B, D, E, C, F, G, H
- Exactly what we called a “pre-order traversal” for trees
  - The marking is because we support arbitrary graphs and we want to process each node exactly once
DFS2(Node start) {
    initialize stack s to hold start
    mark start as visited
    while(s is not empty) {
        next = s.pop() // and “process”
        for each node u adjacent to next
            if(u is not marked)
                mark u and push onto s
    }
}

- A, C, F, H, G, B, E, D
- A different but perfectly fine depth traversal
COMPARISON

Breadth-first always finds shortest length paths, i.e., “optimal solutions”

• Better for “what is the shortest path from \( x \) to \( y \)”

But depth-first can use less space in finding a path

• If \( \text{longest path} \) in the graph is \( p \) and highest out-degree is \( d \) then DFS stack never has more than \( d*p \) elements
• But a queue for BFS may hold \( O(|V|) \) nodes

A third approach (useful in Artificial Intelligence)

• \( \text{Iterative deepening (IDFS)} \):
  • Try DFS but disallow recursion more than \( \kappa \) levels deep
  • If that fails, increment \( \kappa \) and start the entire search over
• Like BFS, finds shortest paths. Like DFS, less space.
TOPOLOGICAL SORT

Problem: Given a DAG $G=(V, E)$, output all vertices in an order such that no vertex appears before another vertex that has an edge to it.

Example input:

One example output:

126, 142, 143, 374, 373, 417, 410, 413, XYZ, 415
Why do we perform topological sorts only on DAGs?
• Because a cycle means there is no correct answer

Is there always a unique answer?
• No, there can be 1 or more answers; depends on the graph
• Graph with 5 topological orders:

Do some DAGs have exactly 1 answer?
• Yes, including all lists

Terminology: A DAG represents a partial order and a topological sort produces a total order that is consistent with it
USES OF TOPOLOGICAL SORT

Figuring out how to graduate

Computing an order in which to recompute cells in a spreadsheet

Determining an order to compile files using a Makefile

In general, taking a dependency graph and finding an order of execution

...
TOPOLOGICAL SORT

1. Label (“mark”) each vertex with its in-degree
   • Think “write in a field in the vertex”
   • Could also do this via a data structure (e.g., array) on the side

2. While there are vertices not yet output:
   a) Choose a vertex $v$ with labeled with in-degree of 0
   b) Output $v$ and conceptually remove it from the graph
   c) For each vertex $u$ adjacent to $v$ (i.e. $u$ such that $(v,u)$ in $E$),
      decrement the in-degree of $u$
Example

Output

Node: 126 142 143 374 373 410 413 415 417 XYZ

Removed?

In-degree: 0 0 2 1 1 1 1 1 1 3
Node:  126 142 143 374 373 410 413 415 417 XYZ
Removed?  x
In-degree:  0  0 2 1 1 1 1 1 1 1 3
  1
Output:  
126
Example

Output:

126

142

Node: 126 142 143 374 373 410 413 415 417 XYZ

Removed? x x

In-degree: 0 0 2 1 1 1 1 1 1 3

1

0
Example

Output:

126
142
143

Node: 126 142 143 374 373 410 413 415 417 XYZ

Removed?: x x x

In-degree: 0 0 2 1 1/ 1 1 1 1 1 1 3

CSE 142 → CSE 143 → CSE 373 → CSE 374 → CSE 410 → XYZ
CSE 413 → CSE 415 → CSE 417
MATH 126

CSE373: Data Structures & Algorithms
Example

Output:

126
142
143
374

Node: 126 142 143 374 373 410 413 415 417 XYZ

Removed?: x x x x

In-degree: 0 0 2 1 1 1 1 1 1 3

1 0 0
0 0

CSE 142 → CSE 143 → CSE 373 → CSE 374 → XYZ
CSE 413 → CSE 410 → CSE 415 → CSE 417
Example

CSE 142 → CSE 143 → CSE 373 → CSE 374
MATH 126 → CSE 373

CSE 410 → CSE 413 → CSE 415 → CSE 417 → XYZ

Node: 126 142 143 374 373 410 413 415 417 XYZ
Removed? x x x x x x
In-degree: 0 0 2 1 1 1 1 1 1 3
1 0 0 0 0 0 0 0 0 2
0

Output:
126
142
143
374
373
Example

Output:

126
142
143
374
373
410
413
415
417
XYZ

Node: 126 142 143 374 373 410 413 415 417 XYZ

Removed? x x x x x x x

In-degree: 0 0 2 1 1 1 1 1 1 3

1 0 0 0 0 0 0 0 2

0

CSE373: Data Structures & Algorithms
Example

Output:
126
142
143
374
373
417
410

Node: 126 142 143 374 373 410 413 415 417 XYZ
Removed? x x x x x x x x x
In-degree: 0 0 2 1 1 1 1 1 1 3

MATH 126
CSE 142
CSE 143
CSE 373
CSE 374
CSE 410
CSE 413
CSE 415
CSE 417
XYZ
Example

CSE 142

CSE 143

CSE 373

CSE 374

CSE 410

CSE 413

CSE 415

CSE 417

XYZ

Node:

126 142 143 374 373 410 413 415 417 XYZ

Removed?

x x x x x x x x x

In-degree:

0 0 2 1 1 1 1 1 1 1 3

Output:

126
142
143
374
373
417
410
413
Example

Output:

126
142
143
374
373
410
413
417
XYZ

Node:

Removed?

In-degree:

CSE 142 → CSE 143 → CSE 373 → CSE 374 → CSE 410 → CSE 413 → CSE 415 → CSE 417 → XYZ
Example

Output:
126
142
143
374
373
417
410
413
XYZ

Node: 126 142 143 374 373 410 413 415 417 XYZ

Removed? x x x x x x x x x x

In-degree: 0 0 2 1 1 1 1 1 1 3

1 0 0 0 0 0 0 0 2

0 1 0

CSE373: Data Structures & Algorithms
Needed a vertex with in-degree 0 to start

• Will always have at least 1 because no cycles

Ties among vertices with in-degrees of 0 can be broken arbitrarily

• Can be more than one correct answer, by definition, depending on the graph
IMPLEMENTATION

The trick is to avoid searching for a zero-degree node every time!

- Keep the “pending” zero-degree nodes in a list, stack, queue, bag, table, or something
- Order we process them affects output but not correctness or efficiency provided add/remove are both $O(1)$

Using a queue:

1. Label each vertex with its in-degree, enqueue 0-degree nodes
2. While queue is not empty
   a) $v = \text{dequeue}$
   b) Output $v$ and remove it from the graph
   c) For each vertex $u$ adjacent to $v$ (i.e. $u$ such that $(v,u)$ in $E$), decrement the in-degree of $u$, if new degree is 0, enqueue it
SINGLE SOURCE SHORTEST PATHS

Done: BFS to find the minimum path length from v to u in $O(|E|+|V|)$

Actually, can find the minimum path length from v to every node
  • Still $O(|E|+|V|)$
  • No faster way for a “distinguished” destination in the worst-case

Now: Weighted graphs

Given a weighted graph and node v,
find the minimum-cost path from v to every node

As before, asymptotically no harder than for one destination
Unlike before, BFS will not work -> only looks at path length.
SHORTEST PATH: APPLICATIONS

Driving directions

Cheap flight itineraries

Network routing

Critical paths in project management
Why BFS won’t work: Shortest path may not have the fewest edges
  • Annoying when this happens with costs of flights

We will assume there are no negative weights
  • *Problem is* ill-defined *if there are negative-cost* cycles
  • *Today’s algorithm is* wrong *if edges can be negative*
    – There are other, slower (but not terrible) algorithms
DIJKSTRA’S ALGORITHM

The idea: reminiscent of BFS, but adapted to handle weights

- Grow the set of nodes whose shortest distance has been computed
- Nodes not in the set will have a “best distance so far”
- A priority queue will turn out to be useful for efficiency
**DIJKSTRA’S ALGORITHM**

Initially, start node has cost 0 and all other nodes have cost $\infty$

At each step:

- Pick closest unknown vertex $v$
- Add it to the “cloud” of known vertices
- Update distances for nodes with edges from $v$

That’s it! (But we need to prove it produces correct answers)
THE ALGORITHM

1. For each node \( v \), set \( v\.cost = \infty \) and \( v\.known = \) false
2. Set \( \text{source}.cost = 0 \)
3. While there are unknown nodes in the graph
   a) Select the unknown node \( v \) with lowest cost
   b) Mark \( v \) as known
   c) For each edge \((v,u)\) with weight \( w \),
      \[
      c1 = v\.cost + w \quad \text{// cost of best path through } v \text{ to } u \\
      c2 = u\.cost \quad \text{// cost of best path to } u \text{ previously known} \\
      \]
      if \((c1 < c2)\){ \quad \text{// if the path through } v \text{ is better} \\
      \begin{align*}
      u\.cost &= c1 \\
      u\.path &= v \quad \text{// for computing actual paths} \\
      \end{align*}
      }
IMPORTANT FEATURES

When a vertex is marked known, the cost of the shortest path to that node is known

- The path is also known by following back-pointers

While a vertex is still not known, another shorter path to it *might* still be found
Order Added to Known Set:

<table>
<thead>
<tr>
<th>vertex</th>
<th>known?</th>
<th>cost</th>
<th>path</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td></td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>B</td>
<td></td>
<td>??</td>
<td></td>
</tr>
<tr>
<td>C</td>
<td></td>
<td>??</td>
<td></td>
</tr>
<tr>
<td>D</td>
<td></td>
<td>??</td>
<td></td>
</tr>
<tr>
<td>E</td>
<td></td>
<td>??</td>
<td></td>
</tr>
<tr>
<td>F</td>
<td></td>
<td>??</td>
<td></td>
</tr>
<tr>
<td>G</td>
<td></td>
<td>??</td>
<td></td>
</tr>
<tr>
<td>H</td>
<td></td>
<td>??</td>
<td></td>
</tr>
</tbody>
</table>
Order Added to Known Set:

A
Order Added to Known Set:
A, C
Order Added to Known Set:

A, C, B
Order Added to Known Set:

A, C, B, D

<table>
<thead>
<tr>
<th>vertex</th>
<th>known?</th>
<th>cost</th>
<th>path</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>Y</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>B</td>
<td>Y</td>
<td>2</td>
<td>A</td>
</tr>
<tr>
<td>C</td>
<td>Y</td>
<td>1</td>
<td>A</td>
</tr>
<tr>
<td>D</td>
<td>Y</td>
<td>4</td>
<td>A</td>
</tr>
<tr>
<td>E</td>
<td></td>
<td>≤ 12</td>
<td>C</td>
</tr>
<tr>
<td>F</td>
<td></td>
<td>≤ 4</td>
<td>B</td>
</tr>
<tr>
<td>G</td>
<td></td>
<td>??</td>
<td></td>
</tr>
<tr>
<td>H</td>
<td></td>
<td>??</td>
<td></td>
</tr>
</tbody>
</table>
Order Added to Known Set:
A, C, B, D, F

<table>
<thead>
<tr>
<th>vertex</th>
<th>known?</th>
<th>cost</th>
<th>path</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>Y</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>B</td>
<td>Y</td>
<td>2</td>
<td>A</td>
</tr>
<tr>
<td>C</td>
<td>Y</td>
<td>1</td>
<td>A</td>
</tr>
<tr>
<td>D</td>
<td>Y</td>
<td>4</td>
<td>A</td>
</tr>
<tr>
<td>E</td>
<td></td>
<td>≤ 12</td>
<td>C</td>
</tr>
<tr>
<td>F</td>
<td>Y</td>
<td>4</td>
<td>B</td>
</tr>
<tr>
<td>G</td>
<td></td>
<td>??</td>
<td></td>
</tr>
<tr>
<td>H</td>
<td></td>
<td>≤ 7</td>
<td>F</td>
</tr>
</tbody>
</table>
Order Added to Known Set:
A, C, B, D, F, H

<table>
<thead>
<tr>
<th>vertex</th>
<th>known?</th>
<th>cost</th>
<th>path</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>Y</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>B</td>
<td>Y</td>
<td>2</td>
<td>A</td>
</tr>
<tr>
<td>C</td>
<td>Y</td>
<td>1</td>
<td>A</td>
</tr>
<tr>
<td>D</td>
<td>Y</td>
<td>4</td>
<td>A</td>
</tr>
<tr>
<td>E</td>
<td></td>
<td>≤ 12</td>
<td>C</td>
</tr>
<tr>
<td>F</td>
<td>Y</td>
<td>4</td>
<td>B</td>
</tr>
<tr>
<td>G</td>
<td></td>
<td>≤ 8</td>
<td>H</td>
</tr>
<tr>
<td>H</td>
<td>Y</td>
<td>7</td>
<td>F</td>
</tr>
</tbody>
</table>
Order Added to Known Set:
A, C, B, D, F, H, G

<table>
<thead>
<tr>
<th>vertex</th>
<th>known?</th>
<th>cost</th>
<th>path</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>Y</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>B</td>
<td>Y</td>
<td>2</td>
<td>A</td>
</tr>
<tr>
<td>C</td>
<td>Y</td>
<td>1</td>
<td>A</td>
</tr>
<tr>
<td>D</td>
<td>Y</td>
<td>4</td>
<td>A</td>
</tr>
<tr>
<td>E</td>
<td></td>
<td>≤ 11</td>
<td>G</td>
</tr>
<tr>
<td>F</td>
<td>Y</td>
<td>4</td>
<td>B</td>
</tr>
<tr>
<td>G</td>
<td>Y</td>
<td>8</td>
<td>H</td>
</tr>
<tr>
<td>H</td>
<td>Y</td>
<td>7</td>
<td>F</td>
</tr>
</tbody>
</table>
Order Added to Known Set:
A, C, B, D, F, H, G, E
FEATURES

When a vertex is marked known, the cost of the shortest path to that node is known

- The path is also known by following back-pointers

While a vertex is still not known, another shorter path to it might still be found

Note: The “Order Added to Known Set” is not important

- A detail about how the algorithm works (client doesn’t care)
- Not used by the algorithm (implementation doesn’t care)
- It is sorted by path-cost, resolving ties in some way
  - Helps give intuition of why the algorithm works
INTERPRETING THE RESULTS

Now that we’re done, how do we get the path from, say, A to E?

Order Added to Known Set:
A, C, B, D, F, H, G, E

<table>
<thead>
<tr>
<th>vertex</th>
<th>known?</th>
<th>cost</th>
<th>path</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>Y</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>B</td>
<td>Y</td>
<td>2</td>
<td>A</td>
</tr>
<tr>
<td>C</td>
<td>Y</td>
<td>1</td>
<td>A</td>
</tr>
<tr>
<td>D</td>
<td>Y</td>
<td>4</td>
<td>A</td>
</tr>
<tr>
<td>E</td>
<td>Y</td>
<td>11</td>
<td>G</td>
</tr>
<tr>
<td>F</td>
<td>Y</td>
<td>4</td>
<td>B</td>
</tr>
<tr>
<td>G</td>
<td>Y</td>
<td>8</td>
<td>H</td>
</tr>
<tr>
<td>H</td>
<td>Y</td>
<td>7</td>
<td>F</td>
</tr>
</tbody>
</table>
How would this have worked differently if we were only interested in:

- The path from A to G?
- The path from A to E?

Order Added to Known Set:

A, C, B, D, F, H, G, E
Order Added to Known Set:

<table>
<thead>
<tr>
<th>vertex</th>
<th>known?</th>
<th>cost</th>
<th>path</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td></td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>B</td>
<td>??</td>
<td></td>
<td></td>
</tr>
<tr>
<td>C</td>
<td>??</td>
<td></td>
<td></td>
</tr>
<tr>
<td>D</td>
<td>??</td>
<td></td>
<td></td>
</tr>
<tr>
<td>E</td>
<td>??</td>
<td></td>
<td></td>
</tr>
<tr>
<td>F</td>
<td>??</td>
<td></td>
<td></td>
</tr>
<tr>
<td>G</td>
<td>??</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Order Added to Known Set:
A

<table>
<thead>
<tr>
<th>vertex</th>
<th>known?</th>
<th>cost</th>
<th>path</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>Y</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>B</td>
<td>??</td>
<td></td>
<td></td>
</tr>
<tr>
<td>C</td>
<td>≤ 2</td>
<td>A</td>
<td></td>
</tr>
<tr>
<td>D</td>
<td>≤ 1</td>
<td>A</td>
<td></td>
</tr>
<tr>
<td>E</td>
<td>??</td>
<td></td>
<td></td>
</tr>
<tr>
<td>F</td>
<td>??</td>
<td></td>
<td></td>
</tr>
<tr>
<td>G</td>
<td>??</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Order Added to Known Set:

A, D
Order Added to Known Set:
A, D, C
Order Added to Known Set:
A, D, C, E

<table>
<thead>
<tr>
<th>vertex</th>
<th>known?</th>
<th>cost</th>
<th>path</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>Y</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>B</td>
<td></td>
<td>≤ 3</td>
<td>E</td>
</tr>
<tr>
<td>C</td>
<td>Y</td>
<td>2</td>
<td>A</td>
</tr>
<tr>
<td>D</td>
<td>Y</td>
<td>1</td>
<td>A</td>
</tr>
<tr>
<td>E</td>
<td>Y</td>
<td>2</td>
<td>D</td>
</tr>
<tr>
<td>F</td>
<td></td>
<td>≤ 4</td>
<td>C</td>
</tr>
<tr>
<td>G</td>
<td></td>
<td>≤ 6</td>
<td>D</td>
</tr>
</tbody>
</table>
Order Added to Known Set:
A, D, C, E, B
Order Added to Known Set:

A, D, C, E, B, F

<table>
<thead>
<tr>
<th>vertex</th>
<th>known?</th>
<th>cost</th>
<th>path</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>Y</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>B</td>
<td>Y</td>
<td>3</td>
<td>E</td>
</tr>
<tr>
<td>C</td>
<td>Y</td>
<td>2</td>
<td>A</td>
</tr>
<tr>
<td>D</td>
<td>Y</td>
<td>1</td>
<td>A</td>
</tr>
<tr>
<td>E</td>
<td>Y</td>
<td>2</td>
<td>D</td>
</tr>
<tr>
<td>F</td>
<td>Y</td>
<td>4</td>
<td>C</td>
</tr>
<tr>
<td>G</td>
<td></td>
<td>≤6</td>
<td>D</td>
</tr>
</tbody>
</table>
Order Added to Known Set:

A, D, C, E, B, F, G
NEXT WEEK

• Another topological sort problem
• Weights and pathfinding
• Start Dijkstra’s algorithm