COMPSCI 311 Introduction to Algorithms
Lecture 4: Graphs
Marius Minea
University of Massachusetts Amherst
slides credit: Dan Sheldon, Akshay Krishnamurthy, Andrew McGregor

Last time: Generic Graph Traversal
Let \( A \) = data structure of discovered nodes
\[
\text{Traverse}(s) \\
\text{Put } s \text{ in } A \\
\text{while } A \text{ is not empty do} \\
\text{Take a node } v \text{ from } A \\
\text{if } v \text{ is not marked "explored" then} \\
\text{Mark } v \text{ as "explored"} \\
\text{for each edge } (v, w) \text{ incident to } v \text{ do} \\
\text{Put } w \text{ in } A \quad \triangleright \ w \text{ is discovered} \\
\text{end for} \\
\text{end if} \\
\text{end while}
\]
\[
\text{BFS: } A \text{ is a queue (FIFO)} \\
\text{DFS: } A \text{ is a stack (LIFO)}
\]

Clicker Question 1
Put \( s \) in \( A \)
\[
\text{while } A \text{ is not empty do} \\
\text{Take a node } v \text{ from } A \\
\text{if } v \text{ is not marked "explored" then} \\
\text{Mark } v \text{ as "explored"} \\
\text{for each edge } (v, w) \text{ incident to } v \text{ do} \\
\text{Put } w \text{ in } A \quad \triangleright \ w \text{ is discovered} \\
\text{end for} \\
\text{end if} \\
\text{end while}
\]
What is the maximum number of times a node \( w \) can be put in \( A \)?
- A: once
- B: degree(\( w \)) times
- C: 2 \times \text{degree}(\( w \)) times
- D: \( |V| \) times

BFS Running Time
How many times does each line execute?
\[
\text{Traverse}(s) \\
\text{Put } s \text{ in } A \quad 1 \\
\text{while } A \text{ is not empty do} \quad 2m \\
\text{Take a node } v \text{ from } A \quad 2m \\
\text{if } v \text{ is not marked "explored" then} \quad 2m \\
\text{Mark } v \text{ as "explored"} \quad n \\
\text{for each edge } (v, w) \text{ incident to } v \text{ do} \quad 2m \\
\text{Put } w \text{ in } A \quad 2m \\
\text{end for} \\
\text{end if} \\
\text{end while}
\]
Running time \( O(m + n) \)

DFS Implementation
Let \( A \) = empty Stack structure of discovered nodes
\[
\text{Traverse}(s) \\
\text{Put } s \text{ in } A \\
\text{while } A \text{ is not empty do} \\
\text{Take a node } v \text{ from } A \\
\text{if } v \text{ is not marked "explored" then} \\
\text{Mark } v \text{ as "explored"} \\
\text{for each edge } (v, w) \text{ incident to } v \text{ do} \\
\text{Put } w \text{ in } A \quad \triangleright \ w \text{ is discovered} \\
\text{end for} \\
\text{end if} \\
\text{end while}
\]
Is this actually DFS? Yes (reverse order for node neighbors)
Running time? \( O(m + n) \)
Clicker Question 2

DFS(u)
Mark u as "explored"
for each edge (u, v) do
if v is not "explored" then
Call DFS(v) recursively
end if
end for

Put s in A
while A is not empty do
Take a node v from A
if v is not "explored" then
Mark v as "explored"
for each edge (v, w) do
Put w in A
end if
end for
end while

Suppose we have a tree with n nodes, height h and degree d.

Compare the memory used by recursive and non-recursive DFS (clarification: for the stack)
- A: recursive: Θ(hd), non-recursive: Θ(h)
- B: recursive: Θ(h), non-recursive: Θ(hd)
- C: recursive: Θ(n), non-recursive: Θ(hd)
- D: recursive: Θ(h), non-recursive: Θ(d)

Back to Connected Components

while There is some unexplored node s do
BFS(s)
Extract connected component containing s
end while

Running time?

Naive: O(m + n) for each component
⇒ O(c(m + n)) if c components.

Better: BFS on component C only works on nodes/edges in C
- Time for component C: Θ(#edges in C + #nodes in C)
- Total time: O(m + n)

Bipartite Testing

Definition Graph G = (V, E) is bipartite if V can be partitioned into sets X, Y such that every edge has one end in X and one in Y.

Can color nodes red/blue s.t. no edges between nodes of same color.

Examples
- Bipartite: student-college graph in stable matching
- Bipartite: client-server connections
- Not bipartite: "odd cycle" (cycle with odd # of nodes)
- Not bipartite: any graph containing odd cycle

Claim (easy): If G contains an odd cycle, it is not bipartite.

Review and Outlook

- Graph traversal by BFS/DFS
  - Different versions of general exploration strategy
  - O(m + n) time
  - Produce trees with useful properties (for other problems)
  - Basic algorithmic primitive — used in many other algorithms
    - path from s to t, connected components

- Bipartite testing
- Directed graphs
  - Traversal
  - Strong connectivity
  - Topological sorting

Bipartite Graphs

Definition Graph G = (V, E) is bipartite if V can be partitioned into sets X, Y such that every edge has one end in X and one in Y.

Can color nodes red/blue s.t. no edges between nodes of same color.

Examples
- Bipartite: student-college graph in stable matching
- Bipartite: client-server connections
- Not bipartite: "odd cycle" (cycle with odd # of nodes)
- Not bipartite: any graph containing odd cycle

Claim (easy): If G contains an odd cycle, it is not bipartite.

Bipartite Testing

Question Given G = (V, E), is G bipartite?

Algorithm? Idea: run BFS from any node s
- L0 = red
- L1 = blue
- L2 = red
- …
- Even layers red, odd layers blue
What could go wrong? Edge between two nodes at same layer.

Algorithm

Run BFS from any node s
if there is an edge between two nodes in same layer then
Output "not bipartite"
else
X = even layers
Y = odd layers
end if

Correctness? Recall: all edges between same or adjacent layers.
1. If there are no edges between nodes in the same layer, then G is bipartite.
2. If there is an edge between two nodes in the same layer, G has an odd cycle and is not bipartite. Proof?
Proof

- Let $T$ be BFS tree of $G$ and suppose $(x, y)$ is an edge between two nodes in the layer $j$
- Let $z \in L_j$ be the least common ancestor of $x$ and $y$ (Useful in proofs: take least/greatest item with some property)
  - $P_{zx} =$ path from $z$ to $x$ in $T$
  - $P_{zy} =$ path from $z$ to $y$ in $T$
  - Path that follows $P_{zx}$ then edge $(x, y)$ then $P_{yz}$ is a cycle of length $2(j - i) + 1$, which is odd
- Therefore $G$ is not bipartite.

Directed Graphs

$G = (V, E)$

- $(u, v) \in E$ is a directed edge
- $u$ points to $v$

Examples

- Facebook: undirected
- Twitter: directed
- Web: directed
- Road network: directed (if one-way roads)

Directed Graph Traversal

Reachability. Find all nodes reachable from some node $s$.

$s$-$t$ shortest path.

What is the length of the shortest directed path from $s$ to $t$?

Algorithm? BFS naturally extends to directed graphs.

Add $v$ to $L_{i+1}$ if there is a directed edge from $L_i$ and $v$ is not already discovered.

Some problems require us to consider the graph $G^{rev}$ with edges reversed.

Useful to keep adjacency lists for both outgoing and incoming edges.

Topological Sorting

Definition A topological ordering of a directed graph is an ordering of the nodes such that all edges go “forward” in the ordering

- Label nodes $v_1, v_2, \ldots, v_n$ such that
- For all edges $(v_i, v_j)$ we have $i < j$
- A way to order the classes so all prerequisites are satisfied

Q: Is a topological ordering possible for any graph?

Directed Acyclic Graphs

Definition

A directed acyclic graph (DAG) is a directed graph with no cycles.

Models dependencies, e.g. course prerequisites:

Math: (strict) partial order (irreflexive, antisymmetric, transitive)

Clicker Question 3

The maximum number of edges in a DAG with $n$ nodes is

- A) $2(n - 1)$
- B) $2n - 1$
- C) $n(n - 1)/2$
- D) $n(n - 1)$
Exercise
1. Find a topological ordering.
2. Devise an algorithm to find a topological ordering.

Claim
If $G$ has a topological ordering, then $G$ is a DAG.

Problem
Given DAG $G$, compute a topological ordering for $G$.

topo-sort($G$)
    while there are nodes remaining do
        Find a node $v$ with no incoming edges
        Place $v$ next in the order
        Delete $v$ and all of its outgoing edges from $G$
    end while

Running time? $O(n^2 + m)$ easy. $O(m + n)$ more clever

Topological Sorting Analysis

▶ In a DAG, there is always a node $v$ with no incoming edges.
Try to prove. (contradiction, pigeonhole principle)

▶ Removing a node $v$ from a DAG, produces a new DAG.

▶ Any node with no incoming edges can be first in topological ordering.

Theorem: $G$ is a DAG if and only if $G$ has a topological ordering.

Directed Graph Connectivity

Strongly connected graph.
Directed path between any two nodes.

Strongly connected component (SCC).
Maximal subset of nodes with directed path between any two.

SCCs can be found in time $O(m + n)$. (Tarjan, 1972)

Graph of SCCs (one node for each) is a DAG.