Lecture 9
Graph Traversal

Euiseong Seo
(euiseong@skku.edu)
Need for Graphs

- One of unifying themes of computer science
- Closely related to many daily life problems
  - Navigation
  - Circuit generation
  - Social network services
  - Games
  - Computer networks
- How can a problem be represented as a graph?
- How to solve a graph problem?
Graph Notation

- A graph $G = (V, E)$
  - $V$ is a set of vertices (nodes)
  - $E$ is a set of edges
    - $E = (x, y)$ where $x, y \in V$
    - Ordered or unordered pairs of vertices from $V$

- Modeling of problems
  - What are vertices and edges in the followings?
    - Road networks
    - Human interactions
    - Program analysis
## Flavors of Graphs

- **Undirected or directed**
  - A graph is undirected if edge $(x, y) \in E$ implies that $(y, x) \in E$, too.
  - Otherwise, the graph is directed.
Flavors of Graphs

- Weighted or unweighted
  - If each edge of a graph is assigned a numerical value, or weight, the graph is a weighted graph.
  - Otherwise, it is a unweighted graph.

![Graph Example](image-url)
Flavors of Graphs

- Cyclic or acyclic
  - An acyclic graph does not contain any cycles
  - Trees are connected acyclic undirected graphs
  - Directed acyclic graphs are called DAGs
Data Structures for Graphs

- Assume that a graph \( G = (V, E) \) contains \( n \) vertices and \( m \) edges

- Adjacency matrix
  - Use a \( n \times n \) matrix \( M \)
  - \( M[i,j] = 1, \) if \((i,j) \in E\)
  - \( M[i,j] = 0, \) if \((i,j) \notin E\)
  - **Pros**
    - Easy to add or remove edges
    - Easy to find a specific edge, \((i,j)\), if exists
  - **Cons**
    - Waste of memory space for sparse graphs
Data Structures for Graphs

- Adjacency lists in lists

1 - 2 - 5
2 - 1 - 5 - 3 - 4 - 1
3 - 2 - 4 - 5
4 - 2 - 5 - 3
5 - 4 - 1 - 2

1 - 2 - 5
2 - 5
3 - 3 - 4
4 - 4
5 - 4
Data Structures for Graphs

- Adjacency lists in matrices
  - Use arrays instead of linked lists
  - Looks like it combines the worst properties of both, but.

```
1  2
2  1  5  3  4
3  1  2
4  2  5  3
5  1  2  4

1  2
2  1  5  4  3
3  2  4
4  2  5  3
5  1  2  4

1  2
2  5
3  1  2
4  3
5  1  2  4
```
List in Array Representation

- An undirected edge (x,y) appears twice, once as y in x’s list and once as x in y’s list
Tree Traversal

- BFS (Breadth First Search)
- DFS (Depth First Search)
Shortest path from $s$

Queue: $s$ 2

Undiscovered
Discovered
Top of queue
Finished
Queue: s 2 3
Queue: s 2 3 5
Queue: 2 3 5 4

5 already discovered: don't enqueue
Queue: 2 3 5 4
Queue: 3 5 4
Queue: 3 5 4 6
Undiscovered
Discovered
Top of queue
Finished

Queue: 5 4 6
### Queue: 4 6
Undiscovered
Discovered
Top of queue
Finished

Queue: 4 6 8
Queue:

Undiscovered
Discovered
Top of queue
Finished

Queue: 6 8 7
Queue: 6 8 7 9
Queue: 8 7 9
Queue: 7 9
Queue: 7 9
Queue: 9
Queue:

⇒ Since Queue is empty, STOP!
DFS

- **Similar to Backtracking**
  - Go as deep as you can
  - Next one is your siblings
- **Stack is an ideal candidate**

DFS(G, v)
for all edges e incident on v
    do if edge e is unexplored then
        w ← opposite(v, e) // return the end point of e distant to v
        if vertex w is unexplored then
            mark e as a discovered edge
            recursively call DFS(G, w)
        else
            mark e as a back edge
Finding Paths

- BFS Tree from x is unique
- Parent[i] is the node that discovered node i during the BFS originated from x
- Finding the shortest path from x to y in a undirected graph
  - By following the chain of ancestors backward from y to the root
Connected Components

- Many seemingly complicated problems reduce to finding connected components
  - 15-Puzzle
- Connected components can be found by using repetitive application of DFS or BFS
Topological Sorting

- One of the fundamental operations on DAGs
- Construct an ordering of the vertices such that all directed edges go from left to right
  - Cannot exist over cyclic graphs
- This gives us a way to process each vertex before any of its successors
  - Suppose we seek the shortest (or longest) path from $x$ to $y$
  - No vertex appearing after $y$ in the topological order can contribute to any such path
Definition

A topological sort of a DAG $G$ is a linear ordering of all its vertices such that if $G$ contains a link $(u,v)$, then node $u$ appears before node $v$ in the ordering.

![Diagram](image)
Topological Sorting Algorithm

- find source nodes (indegree = 0)
  - if there is no such node, the graph is NOT DAG

![Graph Diagram]

Sorted: -
Topological Sorting Algorithm

- span c; decrement in_deg of a, b, e
  - store a in Queue since in_deg becomes 0

```
Sorted: c
```
Topological Sorting Algorithm

- span a; decrement in_deg of b, f
  - store b, f in Queue since ...

Sorted: c a
Topological Sorting Algorithm

- span b; store d in Queue

Sorted: c a b
Topological Sorting Algorithm

- span f; decrement in_deg of e
  - no node with in_deg = 0 is found

Queue Sorted: c a b f
Topological Sorting Algorithm

- span d; store e in Queue.

Sorted: c a b f d
- span e; Queue is empty

**Sorted:** c a b f d e