

Lecture 25: Graph Algorithms

Agenda:

- Graph notions (recall)
- Graph representations (recall)
- Graph traversals
 - Breadth-first search (BFS)

Reading:

- Textbook pages 525 – 539

Lecture 24: Graph Algorithms

Review of some notions:

- (simple, undirected) graph $G = (V, E)$
 - vertex set V
 - edge set E
 - * an edge is an unordered pair of two distinct vertices
- Notions:
 - adjacent (vertex – vertex, edge – edge)
 - incident (vertex – edge)
 - neighborhood of a vertex
 - degree of a vertex — size of its neighborhood
 - path (vertex – vertex)
 - simple path
 - connected (every pair of vertices is connected via a path)
 - subgraph $G' = (V', E')$ of $G = (V, E)$
 - * it is a graph, $V' \subseteq V$, and $E' \subseteq E$
 - connected component (maximal connected subgraph)
 - vertex-disjoint simple paths
 - biconnected graph
 - * connected, and every pair of vertices are connected via two vertex-disjoint (simple) paths
 - biconnected component — maximal biconnected subgraph

Lecture 24: Graph Algorithms

More notions:

- Notions on simple, undirected graphs:
 - cycle — a path with two ending vertices collapsed
 - simple cycle
 - acyclic graph — a graph containing no cycles — also called *forest*
 - tree — connected forest
 - complete graph ($|E| = \frac{|V| \times (|V| - 1)}{2}$)
every pair of vertices are adjacent
 - induced subgraph on a subset of vertices, say $U \subset V$
 $(U, E[U])$, where $E[U] = \{(v_1, v_2) : (v_1, v_2) \in E \&\& v_1, v_2 \in U\}$
 - clique (subset of vertices) — the induced subgraph is complete
 - independent set (of vertices) — the induced subgraph contains no edge
 - vertex coloring — adjacent vertices get distinct colored
- Graph variants:
 - multigraph (remove “simple”)
 - digraph (remove “undirected”)
 - multi-digraph (remove “simple” and “undirected”)
 - edge-weighted graph (every edge has a weight)

Lecture 24: Graph Algorithms

Additional notions on graph variants:

- Rooted tree:
 - directed tree
 - one vertex as the root
 - can define the child-parent relationship
 - we have seen this (forest of rooted trees, in DSUF)
- When the base graph is directed, path/cycle is also directed
- Directed acyclic graph — DAG

Lecture 24: Graph Algorithms

Two representations:

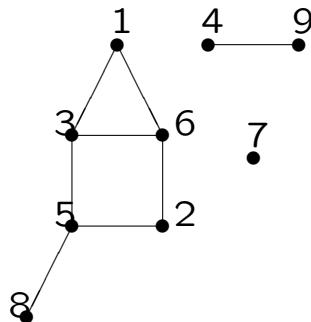
- Adjacency lists: for example,

```
1:  3  6
2:  5  6
3:  1  5  6
4:  9
5:  2  3  8
7:  1  2  3
7:
8:  5
9:  4
```

- Adjacency matrix: for example,

	1	2	3	4	5	6	7	8	9
1			*			*			
2					*	*			
3	*				*	*			
4									*
5		*	*					*	
6	*	*	*						
7									
8					*				
9				*					

They both describe the following graph (graphical view):



Graph traversal:

- The most elementary graph algorithm:
 - goal: visit all vertices, by following all edges in some order
 - e.g., maze traversal
 - the most common graph traversal with a list storing “waiting” vertices
 1. FIFO list (queue) — breadth first search
 2. LIFO list (stack) — depth first search
 3. recursive — depth first search
- Applications:
 - finding connected components
 - determining if the graph is 2-colorable
 - finding an odd cycle, if exists
 - computing pairwise (unweighted) distance (BFS)
 - finding biconnected components (DFS)
 - finding strongly connected components (DFS)

Breadth First Search (BFS):

- Input: simple undirected graph $G = (V, E)$ and start vertex s
- Output: distance (smallest number of edges) from s to each reachable vertex
(in a same connected component, if G is not connected)
- Pseudocode:

```

procedure BFS( $G, s$ )           ** $G = (V, E)$ ,  $s \in V$  start vertex

  for each  $v \in V - s$  do
     $c[v] \leftarrow \text{WHITE}$        **unknown yet
     $d[v] \leftarrow \infty$        **distance from  $s$ 
     $p[v] \leftarrow \text{NIL}$        **predecessor
   $Q \leftarrow \emptyset$          **waiting vertex queue
  enqueue( $Q, s$ )
   $c[s] \leftarrow \text{GRAY}$         **in queue  $Q$ 
   $d[s] \leftarrow 0$ 
  while  $Q \neq \emptyset$  do
     $u \leftarrow \text{dequeue}(Q)$ 
    for each  $v \in \text{Adj}[u]$  do
      if  $c[v] = \text{WHITE}$  then
         $c[v] \leftarrow \text{GRAY}$ 
         $d[v] \leftarrow d[u] + 1$ 
         $p[v] \leftarrow u$ 
        enqueue( $Q, v$ )
     $c[u] \leftarrow \text{BLACK}$       **visited

```

- An example:

$V = \{1, 2, 3, 4, 5, 6\}$

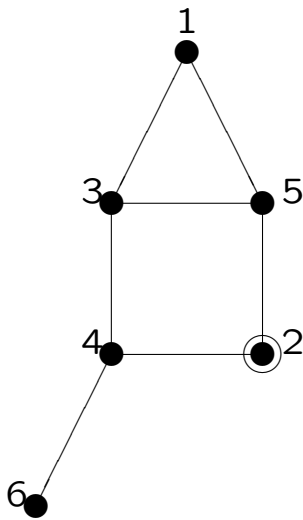
$E = \{\{1, 3\}, \{1, 5\}, \{2, 4\}, \{2, 5\}, \{3, 4\}, \{3, 5\}, \{4, 6\}\}$

$s = 2$

Lecture 24: Graph Algorithms

BFS example:

- $V = \{1, 2, 3, 4, 5, 6\}$
 $E = \{\{1, 3\}, \{1, 5\}, \{2, 4\}, \{2, 5\}, \{3, 4\}, \{3, 5\}, \{4, 6\}\}$
 $s = 2$



Adjacency lists:

1:	3	5	
2:	4	5	
3:	1	4	5
4:	2	3	6
5:	1	2	3
6:	4		

Lecture 24: Graph Algorithms

BFS example:

	1	2	3	4	5	6	Q
color	W	G	W	W	W	W	{2}
distance	∞	0	∞	∞	∞	∞	
parent	NIL	NIL	NIL	NIL	NIL	NIL	
color	W	B	W	G	G	W	{4, 5}
distance	∞	0	∞	1	1	∞	
parent	NIL	NIL	NIL	2	2	NIL	
color	W	B	G	B	G	G	{5, 3, 6}
distance	∞	0	2	1	1	2	
parent	NIL	NIL	4	2	2	4	
color	G	B	G	B	B	G	{3, 6, 1}
distance	2	0	2	1	1	2	
parent	5	NIL	4	2	2	4	
color	G	B	B	B	B	G	{6, 1}
distance	2	0	2	1	1	2	
parent	5	NIL	4	2	2	4	
color	G	B	B	B	B	B	{1}
distance	2	0	2	1	1	2	
parent	5	NIL	4	2	2	4	
color	B	B	B	B	B	B	$\emptyset$
distance	2	0	2	1	1	2	
parent	5	NIL	4	2	2	4	

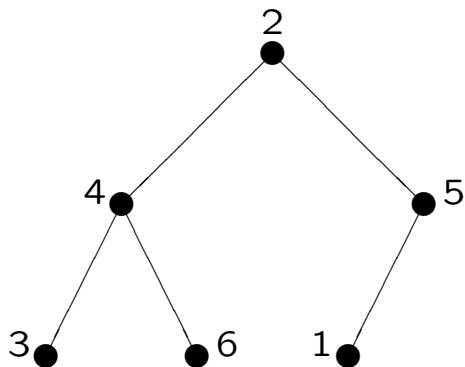
Lecture 24: Graph Algorithms

BFS example:

- Adjacency lists:

```
1:  3  5
2:  4  5
3:  1  4  5
4:  2  3  6
5:  1  2  3
6:  4
```

- BFS tree:



Notes:

- root is the start vertex s
- parent of x is predecessor $p[x]$
- left-to-right child order depends on neighbor ordering (in $Adj[u]$)

BFS analysis:

- $n = |V|$, $m = |E|$
 - Handshaking Lemma: $\sum_{v \in V} \text{degree}(v) = 2m$
 - Analysis:
 - each vertex enqueued exactly once: WHITE $\rightarrow$ GRAY
 - each vertex dequeued exactly once: GRAY $\rightarrow$ BLACK
 - running time:
 1. adjacency list representation:
 $\Theta(n + \sum_{v \in V} \text{degree}(v)) = n + 2m = \Theta(n + m)$
 2. adjacency matrix representation:
 $\Theta(n + \sum_{v \in V} n = n + n^2) = \Theta(n^2)$
 - space complexity:
 1. adjacency list representation:
 $\Theta(n + \sum_{v \in V} \text{degree}(v)) = n + 2m = \Theta(n + m)$
 2. adjacency matrix representation:
 $\Theta(\sum_{v \in V} n = n^2) = \Theta(n^2)$
 - BFS product:
 1. every s -to- v shortest path (tracing the parents)
 2. putting these paths together forms the BFS tree
 - **Warning:** vertices in other connected components wouldn't be discovered !!!
- EXERCISE:** modify the pseudocode to discover ALL vertices

Lecture 24: Graph Algorithms

Have you understood the lecture contents?

well	ok	not-at-all	topic
☐	☐	☐	graph representation recall
☐	☐	☐	graph notions
☐	☐	☐	graph variants?
☐	☐	☐	breadth first search execution
☐	☐	☐	breadth first search analysis