CS161 - Graph Algorithms

David Kauchak

- Types of graphs

  What is a graph? A graph is a set of vertices V and a set of edges $(u, v) \in E$ where $u, v \in V$

  - undirected
  - directed
  - weighted

  Some special cases

  - tree - connected, undirected graph without any cycles
  - DAG (directed acyclic graph) - directed graph without any cycles
  - complete graph - a graph where every vertex is connected
  - bipartite graph - a graph where every vertex can be partitioned into two sets X and Y such that all edges connect a vertex $u \in X$ and a vertex $v \in Y$

- Examples

  - Transportation networks (flights, roads, etc.)
  - Communication networks
  - Web
  - Bayesian networks
  - Social networks
  - Circuit design

- Representation

- Adjacency list
  Each vertex $u \in V$ contains an adjacency list of the set of vertices v such that there exists an edge $(u, v) \in E$

  Example of undirected and directed
- Adjacency matrix
  A graph is represented by a $|V|$ x $|V|$ matrix A such that

$$a_{ij} = \begin{cases} 1 & \text{if } (i,j) \in E \\ 0 & \text{otherwise} \end{cases}$$

  Example of undirected and directed
- Tradeoffs
  Dense vs. sparse graph

  Adjacency list: Space efficient, e.g. web

  Adjacency matrix: constant time lookup to see if an edge exists

  Best of both worlds?
- Weighted graphs
  Adjacency list: store the weight in the adjacency list

  Adjacency matrix:

$$a_{ij} = \begin{cases} weight & \text{if } (i,j) \in E \\ 0 & \text{otherwise} \end{cases}$$

- Graph algorithms
  - graph traversal (BFS, DFS)
  - Shortest path from a to b
    - unwieghted
    - weighted positive weights
    - negative/positive weights
  - All pair shortest paths
  - Are all nodes in the graph connected?

- Is the graph a bipartite graph?
- ...

- Breadth first search (BFS) on trees

```
TreeBFS(T)
1  Enqueue(Q, Root(T))
2  while !Empty(Q)
3          v ← Dequeue(Q)
4          Visit(v)
5          for all c ∈ Children(v)
6                  Enqueue(Q, c)
```

What order does the algorithm traverse the nodes?

Does it visit all of the nodes?

Frontier - The frontier of the algorithm is the set of vertices that have been visited

Runtime - Visits each node and edge exactly once - $O(|V| + |E|)$

- Depth first search (DFS) on trees

```
TreeDFS(T)
1  Push(S, Root(T))
2  while !Empty(S)
3          v ← Pop(S)
4          Visit(v)
5          for all c ∈ Children(v)
6                  Push(S, c)
```

What order does the algorithm traverse nodes?

Frontier?

Runtime

- BFS on graphs

  Visits all nodes reachable from s and records the shortest distance from s to each other node

  What happens when we generalize to graphs?

  Maze analogy

  BFS(G, s)

```
 1  for each v ∈ V
 2          dist[v] = ∞
 3  dist[s] = 0
 4  ENQUEUE(Q, s)
 5  while !EMPTY(Q)
 6          u ← DEQUEUE(Q)
 7          VISIT(u)
 8          for each edge (u, v) ∈ E
 9                  if dist[v] = ∞
10                          ENQUEUE(Q, v)
11                          dist[v] ← dist[u] + 1
```

  - Is it correct? Does BFS visit all nodes reachable from node s?

    Conside some u that the algorithm "misses". Choose any path from s to u and look at the last vertex on that path that the procedure actually visited. Call this node z, and let w be the node immediately after it on the same path. z would be visited and w was not, which is a contradiction given the pseudocode above.

  - Runtime

    The algorithm visits each node at most once and each edge is examined twice, once from u and once from v, $O(|V| + |E|)$

    What about directed graphs?

    What if the graph were stored as an adjacency matrix?

  BFS generates a tree from the start node s, called the BFS-Tree. What properties does this tree have?

- If a path exists from s to v (i.e. v is reachable from s), then v will be in the BFS-Tree
- The path in the BFS-Tree from any two nodes, u, v (including the root) is the shortest path between those two nodes.
  Think about how BFS expands the frontier

  Suppose it was not the shortest. Then there would exist some other node w, such that $distance(u, w)+distance(w, v) < distance(u, v)$, where the path from u to v does not include w. However, given that BFS expands the frontier one step at a time.

- DFS on graphs

  DFS-VISIT(u)
  ```
  1  visited[u] ← true
  2  PreVisit(u)
  3  for all edges (u, v) ∈ E
  4          if !visited[v]
  5                  DFS-Visit(v)
  6  PostVisit(u)
  ```

  DFS(G)
  ```
  1  for all v ∈ V
  2          visited[v] ← false
  3  for all v ∈ V
  4          if !visited[v]
  5                  DFS-Visit(v)
  ```

  The output from DFS is a *depth-first forest*

  Why do we try and visit all nodes using DFS and not BFS? We could do a similar procedure for BFS, however, BFS is generally used to find shortest paths, while DFS is generally used to look at connectedness.

  - Is it correct? Does it explore all nodes?
  - Runtime
    Similar to BFS each vertex is visted once and each edge visited twice, $O(|V| + |E|)$

- Topological sort of a DAG
  - A topological sort indicates precedence, e.g. classes.
  - Why not topological sort of any directed graph?
  - If G is a DAG, then it has a topological ordering. Why?

    TOPOLOGICAL-SORT1(G)
    1 Find a node v with no incoming edges
    2 Delete v from G
    3 Add v to linked list
    4 TOPOLOGICAL-SORT1(G)

    * Is it correct? Root of a DAG contains DAGs as children
    * Runtime
      1 - $O(|V| + |E|)$
      2 - $O(|E|)$ overall
      4 - Call $|V|$ times - $O(|V|^2 + |V||E|)$

    TOPOLOGICAL-SORT2(G)
    ```
    1  for all edges (u, v) ∈ E
    2          active[v] ← active[v] + 1
    3  for all v ∈ V
    4          if active[v] = 0
    5                  ENQUEUE(S, v)
    6  while !EMPTY(S)
    7          u ← DEQUEUE(S)
    8          add u to linked list
    9          for each edge (u, v) ∈ E
    10                 active[v] ← active[v] − 1
    11                 if active[v] = 0
    12                         ENQUEUE(S, v)
    ```

    * Is it correct?
    * Running time?

  Example

  Runtime

- Does a cycle exist in a directed graph, i.e. is it a DAG?

  What is a cycle? A cycle is a path in the tree $v_0, v_1, v_2, ..., v_n, v_0$ where the beginning and ending vertex are the same.

  Be careful, there are things that look like cycles, but are not.

  Call TOPOLOGICAL-SORT. If the number of nodes in linked list is the number of vertices, then yes, otherwise, no.

- Is an undirected graph connected? Discovering connected components in an undirected graph.

  Given a node $u \in V$ can we reach all other nodes in the graph?

  Run DFS or BFS and keep track of the visited nodes. If we visit all of the nodes, then yes, otherwise no.

- Is the graph strongly connected?

  A strongly connected graph is a *directed* graph where any node $u \in V$ is reachable from any other node v.

  STRONGLY-CONNECTED(G)

  ```
  1  Run DFS or BFS from some node u
  2  if not all nodes are visited
  3          return false
  4  Create graph G^R by reversing all edge dirctions
  5  Run DFS or BFS on G^R from node u
  6  if not all nodes are visited
  7          return false
  8  return true
  ```

- Is the graph a bipartite graph?

  What makes a graph bipartite?

  Is a triangle a bipartite graph? Why not?

Can we divide the graph into two colors? Pick a node and color it red. If the graph is bipartite, what must we know about all of it's neighbors? What about the neighbors of those nodes?

```
BIPARTITE(G)
 1  Run BFS
 2  for all v ∈ V
 3          if dist[v] mod 2 = 0
 4                  color[v] = red
 5          else
 6                  color[v] = blue
 7  for all (u, v) ∈ E
 8          if color[u] = color[v]
 9                  return false
10  return true
```

- Does a sink exist in a directed graph?
- Isomorphism

These notes are adapted from material found in chapter 22 of [1], chapter 3 of [2] and chapters 3,4 of [3].

References

[1] Thomas H. Cormen, Charles E. Leiserson Ronald L. Rivest and Clifford Stein. 2007. Introduction to Algorithms, 2nd ed. MIT Press.
[2] Jon Kleinberg and Eva Tardos. 2006. Algorithm Design. Pearson Education, Inc.
[3] Sanjoy Dasgupta, Christos Papadimitiou and Umesh Vazirani. 2008. Algorithms. McGraw-Hill Companies, Inc.