CSCI 1102 Computer Science 2

Spring 2018

Lecture Notes

Week 14: Graphs

Topics:

1. Graphs
2. Graph Representations
3. Traversal
4. Deep and Shallow Copying of Data Structures

---

1. Graphs

Graph structures are ubiquitous in computing applications and there are literally thousands of interesting graph algorithms. The follow depicts a *weighted directed graph* with 4 *vertices* A, B, C and D and 6 weighted connections or *edges*.

```
A -------5------> B
| \             ^ |
|   \         /   |
3     6      5    8
|      \    /     |
v       v  /      |
C <--1-- D  <-----+
```

A *weighted directed graph* G is a pair (E, V) where V be a set of *vertices* and $E \subseteq V \times V \times \aleph$ is a set of *edges*. The graph G_0 shown above is

$$G_0 = (\{A, B, C, D\}, \{(A, B, 5), (A, C, 3), (A, D, 6), (D, C, 1), (D, B, 5), (B, D, 8)\})$$

Intuitively, an edge (A, B, w) has a source vertex, a destination vertex and a weight or cost w to get from the source to the destination. An *unweighted directed graph* is the special case of a weighted directed graph in which all weights are the same. A *weighted undirected graph* is the special case of a weighted directed graph in which for every edge $(A, B, w) \in E$, the edge $(B, A, w) \in E$. In a drawing for this latter case the arrows are usually either two-headed or they are simple lines.

A *path* in a graph is a sequence of edges. A path from source vertex A to destination vertex B is a path in which the first edge is (A, X, w) and the last edge is (Y, B, w') for some vertices X and Y and weights w and w'.

In G_0 there are paths from A to all of B, C and D, but not back to A. In fact there are infinitely many paths from A to each of B, C and D. In G_0 there are (infinitely many) paths from B and D to all of D, C and B but there are no paths to A. There are no paths starting at C.

Representations of Graphs

The two most common methods for representing a graph in code is to record edges in either a matrix or a list. The graph G_0 would have the following representations.

Adjacency Matrix

```
        A   B   C   D
      +---+---+---+---+
    A |   | 5 | 3 | 6 |
      +---+---+---+---+
    B |   |   |   | 8 |
      +---+---+---+---+
    C |   |   |   |   |
      +---+---+---+---+
    D |   | 5 | 1 |   |
      +---+---+---+---+
```

Adjacency List

```
  +---+     +---+---+---+     +---+---+---+     +---+---+---+
A | o-+--->| B | 5 | o-+--->| C | 3 | o-+--->| D | 6 | o-+-+
  |   |     +---+---+---+     +---+---+---+     +---+---+---+ =
  +---+     +---+---+---+
B | o-+--->| D | 8 | o-+-+
  |   |     +---+---+---+ =
  +---+
C | o-+--+
  |   |  =
  +---+     +---+---+---+     +---+---+---+
D | o-+--->| B | 5 | o-+--->| C | 1 | o-+--+
  |   |     +---+---+---+     +---+---+---+  =
  +---+
```

Note that for a graph with N vertices, the adjacency matrix representation requires N^2 space.

Depth First Traversal

Let $G = (V, E)$ with $v \in V$. Starting with source vertex v we can find all nodes in V reachable from v.

```
G = (V, E), v in V is a source vertex, S is a stack

1. mark(v)
2. push(v, S)
3. while !isEmpty(S):
4.   v = pop(S)
5.   visit(v)
6.   for every (v, w, _) in E
7.     if w is unmarked:
8.       mark(w)
9.       push(w, S)
```

Example

Let G_1 be

```
               +----- A -----+
               |      ^      |
               v      |      v
               B      |      C ---> D <---- F
               |      |      |
               +----> E <----+
```

Tracing traversal of G_1 with source vertex C.

```
  v       Stack        Marked
--------------------------------
  C         C          C
  C         -          C
  C         E, D       C, D, E
  E         D          C, D, E
  E         A, D       C, D, E, A
  A         D          C, D, E, A
  A         B, D       C, D, E, A, B
  B         D          C, D, E, A, B
  D         -          C, D, E, A, B
```

Breadth First Traversal

Let $G = (V, E)$ with $v \in V$. Starting with source vertex v we can find all nodes in V reachable from v.

```
G = (V, E), v in V is a source vertex, Q is a queue

mark(v)
enqueue(v, Q)
while !isEmpty(Q):
  v = dequeue(Q)
  visit(v)
  for every (v, w, _) in E
    if w is unmarked:
      mark(w)
      enqueue(w, Q)
```

Tracing traversal of G_1 with source vertex C.

```
  v        Queue        Marked
--------------------------------
  C          C           C
  C          -           C
  C          E, D        C, D, E
  E          D           C, D, E
  E          D, A        C, D, E, A
  D          A           C, D, E, A
  A          -           C, D, E, A
  A          B           C, D, E, A, B
  B          -           C, D, E, A, B
```

Dijkstra's Shortest Path Algorithm

Let $G = (V, E)$ with $v \in V$. Starting with source vertex v we can find all nodes in V reachable from v.

```
G = (V, E), v in V is a source vertex, PQ is a min priority queue

function Dijkstra(Graph, source):
    dist[source] = 0                                // Initialization

    for each vertex v in V:
        if v != source
            dist[v] = INFINITY                      // Unknown distance from source to v
            prev[v] = UNDEFINED                     // Predecessor of v
        enqueue(PQ, v, dist[v])
```

```
    while !isEmpty(PQ):                              // The main loop
        u = dequeue(PQ):                             // Remove and return best vertex
        for each (u, v, cost) in E:                  // only v that is still in Q
            newDistance = dist[u] + cost
            if newDistance < dist[v]:
                dist[v] = newDistance
                prev[v] = u
                decreasePriority(PQ, v, newDistance)

    return dist, prevq
```