Network Science	
Lecture 3: July 23, 2009	Lecturer: Maleq Khan
Scribe: Fei Huang	

Graph Theory III

3.1 Motivation and Outline

Graph is a fundamental tool in the research of network science, which builds mathematical structures to model the relations among objects from a certain collection. In this lecture, we go through some basic definitions in graph theory, such as graph connectivity, component, tree, and distance, etc. Some useful algorithms in graph theory are also introduced, such as *Breadth First Search* (BFS) and *Depth First Search* (DFS).

3.1.1 Basic Definitions

Given a graph $G(V, E)$ with vertices V and edges E and a length function $l : E \to \Re^+$. We define the following concepts, respectively:

Reachability. Node u is reachable from node v if there is a path from v to u, where $u, v \in V$.

Connectivity. Graph G is connected if each node is reachable from every other node.

Component. A component is 1) a subgraph which is connected, and 2) not a subgraph of larger connected subgraph.

Giant Component. It is an informal term for a connected component covering significant portion of all the nodes.

Tree. Tree is a connected acyclic simple graph.

Binary Tree. Binary tree is a rooted tree with each node having at most two children.

Path Length. Path length is the number of edges in the path.

Shortest Path. Shortest path is the path that has the least length between two nodes.

Distance. Distance, denoted as $d(u, v)$, between nodes u and v is the length of the shortest path between u and v.

Diameter. Diameter is the maximum distance between any pair of nodes in the graph. Mathematically, it can be expressed as $D = \max_{u,v \in V} d(u, v)$. If we specify diameter as a *weighted diameter*, it refers to maximum weighted distance between any pair of nodes in the graph.

3.1.2 Small World Experiment

A well-known example using the idea of distance in social network is **small world experiment**, where Stanley Milgram et al. examines the average path length for social networks of people [1]. The result of this research reveals human society is a small world type network characterized by short path length of 6 in average. The experiments are often called "six degrees of separation", which can illustrated by Figure 3.1.

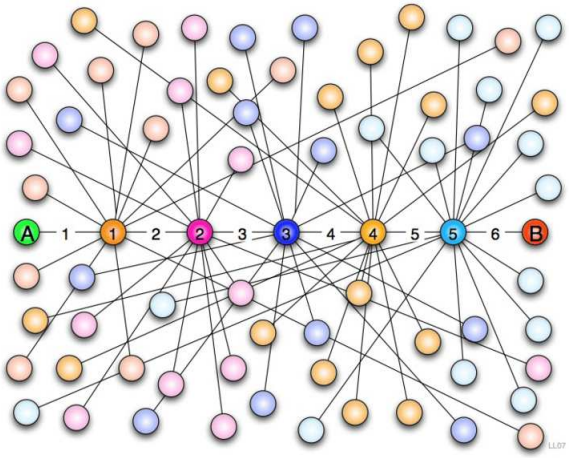


Figure 3.1: The "six degrees of separation" model (Figure comes from *http://en.wikipedia.org/wiki/File:Six_degrees_of_separation.png*)

3.1.3 Erdös Number

Another example in social network is Erdös number, which describes the "collaborative distance" between a person and prolific mathematician Paul Erdös, who is an influential and itinerant mathematician, creating a record in publications (at least 1400) compared with other mathematicians in history. Figure 3.2 describes the method of how to calculate the Erdös number.

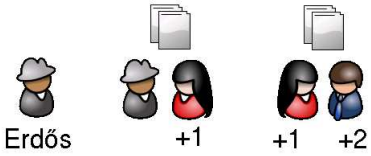


Figure 3.2: If Rose collaborates with Paul Erdös on one paper, and with Jack on another, but Jack never collaborates with Erdös himself, then Jack is given an Erdös number of 2, as he is two steps from Erdös. (Figure comes from *http://upload.wikimedia.org/wikipedia/commons/f/fe/Erdosnumber.png*)

Question:

Suppose $\max_{v \in V} d(u, v) = x$. (1) What is the lower and upper bound for $d(w_1, w_2), \forall w_1, w_2$? (2) And the same question for diameter D.

Answer: (1) $1 \leq d(w_1, w_2) \leq 2x, \forall w_1, w_2$. (2) $x \leq D \leq 2x$.

3.1.4 Bipartite Graph

A bipartite graph, called bigraph, is a graph whose vertices consist of two disjoint sets A and B with every edge connects a vertex in A to one in B, i.e., A and B are independent sets. We denote bipartite graph as $G = (A, B, E)$. Figure 3.3 shows an example.

A complete bipartite graph, called biclique, is a special kind of bipartite graph where every vertex of one set is connected to every vertex of another set. An example of biclique is illustrated in Figure 3.4.

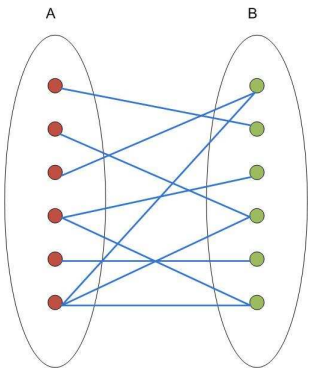


Figure 3.3: An example of bipartite graph $G = (A, B, E)$.

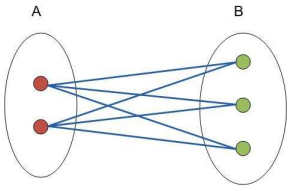


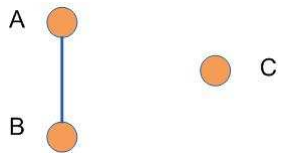
Figure 3.4: An example of complete bipartite graph $G = (A, B, E)$.

3.1.5 Graph-theoretic Data Structures

There are generally two types of methods to store graphs in a computer system, including *List Structures* and *Matrix Structures*.

One frequently used list structure is *adjacency list*, where each vertex has a list of vertices it is adjacent to. For the graph in Figure 3.5a, we can express it by an adjacency list as Figure 3.5b.

One frequently used matrix structure is *adjacency matrix*, where the corresponding element (i, j) in matrix will be marked as 1 if there is an edge from vertex i to vertex j; otherwise, (i, j) will be marked as 0. For the same graph in Figure 3.5a, we can express it by an adjacency matrix as Figure 3.5c.



(a) An example graph.

Vertex	Adjacent Vertices
A	B
B	A
C	

(b) Adjacency list

	A	B	C
A	0	1	0
B	1	0	0
C	0	0	0

(c) Adjacency matrix

Figure 3.5: An example graph.

Comparing those structures, the space complexity of adjacency list is $2|E|$ while adjacency matrix consumes a space complexity of $O(n^2)$ for graph $G = (V, E)$. On the other hand, the access time is constant for adjacency matrix while the access time of adjacency list depends on the number of the elements on the list: the larger the number, the longer the access time. If it is an unsorted adjacent list, the access time is $O(d)$; otherwise, the access time is $O(\log d)$.

From the perspective of computing, adjacency matrix makes it easy for modeling a graph computation, such as find subgraphs, or reverse a directed graph.

Denote $\mathbf{M}$ as the adjacency matrix. Element (i, j) in the result of matrix self-multiplication $\mathbf{M}^r$ tells how many paths from i to j have a length of r in the graph, where $\mathbf{M}^r = \underbrace{\mathbf{M} \times \mathbf{M} \ldots \times \mathbf{M}}_{r \text{ terms}}$ and $i, j \in V$.

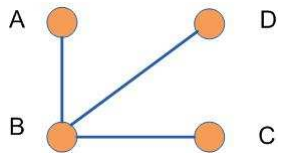


Figure 3.6: An example graph for computation based on adjacency matrix.

For example, for the graph in Figure 3.6, it can be expressed by adjacency matrix as

$$\begin{bmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix} \tag{3.1}$$

A matrix self-multiplication is carried out in Equation (3.1), where element (i, j) in the result tells how many paths from i to j have a length of 2 in the graph.

$$\begin{bmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix} \begin{bmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 1 & 1 \\ 0 & 3 & 0 & 0 \\ 1 & 0 & 1 & 1 \\ 1 & 0 & 1 & 1 \end{bmatrix}. \tag{3.2}$$

3.1.6 Algorithms for BFS and DFS

Breadth-first search (BFS) is a graph search algorithm that begins at the root node and explores all the neighboring nodes. Then for each of those nearest nodes, it explores their unexplored neighbor nodes, and so on, until it finds the goal [2]. The following Algorithm 1 describes the basic algorithm of BFS.

Algorithm 1 BFS(u)): Breadth-first search algorithm

```
1: Enqueue(Q, u)
2: while Q is not empty do
3:     v ← Dequeue(Q)
4:     Examine v & mark it as visited
5:     Enqueue all unmarked neighbors of v
6: end while
```

Depth-first search (DFS) is an algorithm for traversing or searching a tree, tree structure, or graph. One starts at the root (selecting some node as the root in the graph case) and explores as far as possible along each branch before backtracking [3]. Algorithm 2 shows a simple method implementing DFS.

Algorithm 2 DFS-1(u)): Depth-first search algorithm 1

```
1: Examine u & mark u as visited
2: for each unmarked neighbor v of u do
3:     DFS-1(v)
4: end for
```

Another method can be described by Algorithm 3. In general, Algorithm 2 consumes additional memory and time than Algorithm 3 due to the stack operation in its iterative process.

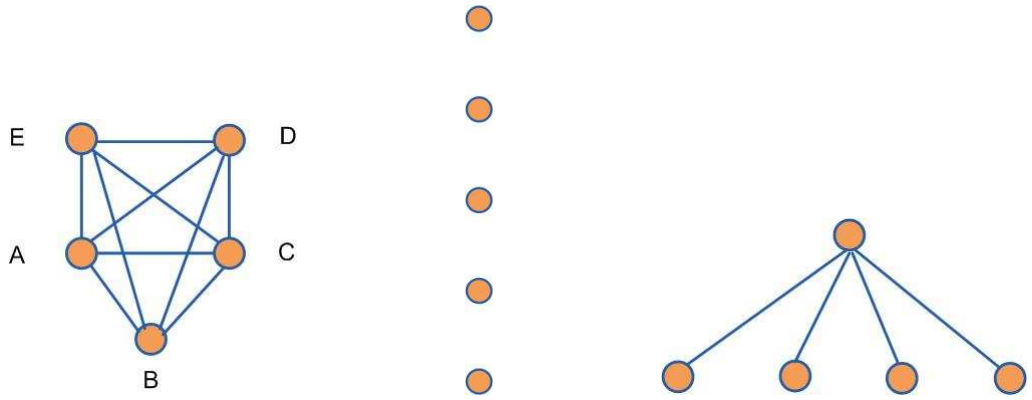
Comparing BFS with DFS, BFS will cost more memory since in BFS all of the nodes of a level must be saved until their child nodes in the next level have been generated. However, if we want to find the shortest path, BFS provides the way to implement it.

Algorithm 3 DFS-2(u)): Depth-first search algorithm 2

```
1: PUSH(S, u)
2: while S is not empty do
3:    v ← POP(S)
4:    if v is not visited then
5:       Examine v & mark it as visited
6:       Push all unmarked neighbors of v
7:    end if
8: end while
```

3.1.7 Exercise

For the graph in Figure 3.7a, do BFS. (1) Can you get a tree like Figure 3.7b or Figure 3.7c? (2) The same question if we do DFS on this graph.



(a) (b) (c)

References

[1] J. Travers and S. Milgram, "An experimental study of the small world problem," *Sociometry*, vol. 32, no. 4, pp. 425–443, 1969.

[2] *http://en.wikipedia.org/wiki/Breadth-first_search.*

[3] *http://en.wikipedia.org/wiki/Depth-first_search.*