Section 2.1

Distance

WEIGHTED EDGES

Many times in graphs modeling physical situations we label each edge with nonnegative number called a **weight**. Such weights might represent the physical distance between two vertices, the time it takes to travel between two vertices, etc.

If a graph has edges with no labels, we can consider all the weights to be one.

LENGTH AND DISTANCE

- In a graph with weighted edges, the **length of a path** is the sum of the lengths of the edges in the path.
- Let x and y be vertices of a graph. The **distance from x to y**, denoted $d(x, y)$, is the minimum length of an $x - y$ path in the graph.

METRIC FUNCTION

Let f be a function on a set of objects S. Let $x, y \in S$. The function f is a **metric function** (or simply a **metric**) if it satisfies the following properties.

1. $f(x, y) \geq 0$ and $f(x, y) = 0$ if and only if $x = y$.
2. $f(x, y) = f(y, x)$ [Symmetric Property]
3. $f(x, y) + f(y, z) \geq f(x, z)$ [Triangle inequality]

DISTANCE IS A METRIC

As defined previously, the distance between two vertices in a graph is a metric function.

DIAMETER AND RADIUS

- The **diameter**, denoted $diam(G)$, of a connected graph G equals

$$\max_{u \in V} \max_{v \in V} d(u, v)$$

In other words, let $S = \{\text{distance between } v \text{ and the vertex farthest from } v : v \in V(G)\}$, the diameter is the maximum of S.

- The **radius**, denoted $rad(G)$, of a connected graph G equals

$$\min_{u \in V} \max_{v \in V} d(u, v)$$

In other words, the radius in the minimum of S.

RELATIONSHIP BETWEEN RADIUS AND DIAMETER

Theorem 2.1.1: For any connected graph G,

$$rad(G) \leq diam(G) \leq 2\, rad(G)\,.$$

ISOMETRIC FROM

A connected graph H is isometric from a connected graph G if for each vertex x in G, there is a 1-1 and onto function $F_x\colon V(G) \to V(H)$ that preserves distances from x, that is $d_G(x, y) = d_H\big(F_x(x), F_x(y)\big)$.

THEOREM ON ISOMETRIC FROM

Theorem 2.1.2: The relation isometric from is not symmetric; that is, if G_2 is isometric from G_1, then G_1 need not be isometric from G_2.

BREADTH-FIRST SEARCH ALGORITHM FOR UNLABELED GRAPHS

Algorithm 2.1.1 Breadth-First Search (BFS).

Input: An unlabeled graph $G = (V, E)$ with distinguished vertex x.

Output: The distances from x to all vertices reachable from x.

Method: Use variable i to measure the distance from x, and label vertices with i as their distance is found.

BFS (CONCLUDED)

1. $i \leftarrow 0$.
2. Label x with "i."
3. Find all unlabeled vertices adjacent to at least one vertex with label i. If none is found, stop because we have reached all possible vertices.
4. Label all vertices found in step 3 with $i + 1$.
5. Let $i \leftarrow i + 1$, and go to step 3.

PROPERTIES OF THE BFS ALGORITHM

- The BFS algorithm produces a **search tree**, using some edge to reach each new vertex along a path from x.
- Using incidence lists for the data, the BFS algorithm has time complexity $O(|E|)$.
- To find distances between any two vertices in a graph, we perform the BFS algorithm starting at each vertex. Thus, to find all distances, the algorithm has time complexity $O(|V||E|)$.

THEOREM ON BFS

Theorem 2.1.3: When the BFS algorithm halts, each vertex reachable from x is labeled with its distance from x.

DISTANCES IN DIGRAPHS

- The arcs of the digraph are labeled with a weight $l(e)$.
- To determine the shortest path from v to u, we need information about the distances to intermediate vertices. We do this by labeling the intermediate vertices.
- This takes one of two forms:
  - The distance $d(u, w)$ between u and the intermediate vertex w.
  - The pair $d(u, w)$ and the predecessor of w on this path, $pred(w)$. The predecessor aids in backtracking to find the path.

TWO TYPES OF ALGORITHMS FOR DISTANCES IN DIGRAPHS

- In **label-setting** algorithms, during each pass through the algorithm, one vertex label is assigned a value which remains unchanged thereafter.
- In **label-correcting** algorithms, any label may be changed during the process.

DIJKSTRA'S DISTANCE ALGORITHM

Algorithm 2.1.2 Dijkstra's Distance Algorithm

Input: A labeled digraph $D = (V, E)$ with initial vertex v_1.

Output: The distance from v_1 to all other vertices.

Method: Label each vertex v with $(L(v), pred(v))$, which is the length of a shortest path from v_1 to v that has been found at that instant and the predecessor of v along the path.

1. For all $v \in V(D)$ and for all $v \neq v_1$ set $L(v) \leftarrow \infty$ and $C \leftarrow V$.
2. While $C \neq \emptyset$;
   Find $v \in C$ with minimum label $L(v)$.
   $C \leftarrow C - \{v\}$
   For every $e = v \rightarrow w$,
   if $w \in C$ and $L(w) > L(v) + l(e)$ then
   $L(w) \leftarrow L(v) + l(e)$ and $pred(w) = v$.

THEOREM ON DIJKSTRA'S ALGORITHM

Theorem 2.1.4: If $L(v)$ is finite when Algorithm 2.1.2 halts, then $d(x, v) = L(v)$.

PROPERTIES OF DIJKSTRA'S ALGORITHM

- Dijkstra's algorithm is label-setting.
- The algorithm has time complexity $O(|V|^2)$.
- To find distances between any two vertices in a graph, we perform the algorithm starting at each vertex. Thus, to find all distances, the algorithm has time complexity $O(|V|^3)$.
- Dijkstra's algorithm works on graphs with arcs replaced by edges.

FAILURE OF DIJKSTRA'S ALGORITHM

- Disjkstra's algorithm can fail if we allow negative edge weights.
- There are algorithms that will find distances in digraphs when the digraph contains no cycles whose total length is negative (called a **negative cycle**). These algorithms are those of Ford and Floyd.

FORD'S DISTANCE ALGORITHM

Algorithm 2.1.3 Ford's Distance Algorithm

Input: A digraph with (possibly) negative are weights $w(e)$, but no negative cycles.

Output: The distance from x to all vertices reachable from x.

Method: Label correcting.

1. $L(x) \leftarrow 0$ and for every $v \neq x$ set $L(v) \leftarrow \infty$.
2. While there is an arc $e = u \rightarrow v$ such that $L(v) > L(u) + w(e)$ set $L(v) \leftarrow L(u) + w(e)$ and $pred(v) = u$.

COMMENTS ON FORD'S ALGORITHM

- **Theorem 2.1.5**: For a digraph D with no negative cycles, when Algorithm 2.1.3 halts, $L(v) = d(x, v)$ for every vertex v.
- The time complexity of Ford's Algorithm is $O(|V|\,|E|)$.
- Ford's Algorithm can only be used on digraphs. In graphs, an edge $e = xy$ with a negative label causes an endless loop using this edge to continually decrease the labels on x and y.

A DEFINITION NEEDED FOR FLOYD'S ALGORITHM

For $i \neq j$, define

$$d^0(v_i, v_j) = \begin{cases} l(e) & \text{if } v_1 \to v_j \\ \infty & \text{otherwise} \end{cases}$$

Let $d^k(v_i, v_j)$ be the length of the shortest path from v_i to v_j among all paths from v_i to v_j that use only vertices from the set $\{v_1, v_2, \dots, v_k\}$.

FLOYD'S DISTANCE ALGORITHM

Algorithm 2.1.4 Floyd's Distance Algorithm

Input: A digraph $D = (V, E)$ without negative cycles.

Output: The distances from v_i to v_j.

Method: Constant refinement of the distances as the set of excluded vertices decreases.

1. $k \leftarrow 1$.
2. For every $1 \leq i, j \leq n$,
   $d^k(v_1, v_j) \leftarrow \min\{d^{k-1}(v_i, v_j), d^{k-1}(v_i, v_k) + d^{k-1}(v_k, v_j)\}$.
3. If $k = |V|$, then stop;
   else $k \leftarrow k + 1$ and go to step 2.

TIME COMPLEXITY OF FLOYD'S ALGORITHM

The time complexity of Floyd's Algorithm is $O(|V|^3)$.