CSc 3102: Introdction to Graphs

Supplementary Notes

- Understand and use basic graph terminology and concepts
- Define and discuss basic graph structures
- Graph ADT Primitive Operations
- Graph Adjacency Linked-List and Matrix Representations

1 Basic Concepts

Definition 1.

1. A **graph** is a pair $G = (V, E)$ of sets satisfying $E \subseteq [V]^2$. V is the set of *vertices* and E is the set of *edges*, pairs of vertices.

2. A **directed graph** or **digraph** is a graph in which the edges are ordered. An *undirected graph* is one in which the edges are unordered.

3. A graph with vertex set V is said to be a graph **on** V. The vertex set of a graph G is referred to as $V(G)$, its edge set as $E(G)$. The number of vertices of a graph G is its **order**, written as $|G|$. $\|G\|$ denotes the number of edges.

4. A vertex v is **incident** with an edge e, and vice verse, if $v \in e$. The two vertices incident with an edge are its **endvertices**, **ends** or **endpoints**.

5. Two vertices x, y of G are **adjacent**, or **neighbors**, if xy is an edge of G. Two edges $e \neq f$ are **adjacent** if they have an endpoint in common.

6. A **complete graph** is a graph with every pair of its vertices connected by an edge. A complete graph of order n is usually denoted K^n or K_n.

7. A **weighted graph** (or **weighted digraph**) is a graph (or digraph) with numbers (weights or costs) assigned to its edges.

8. A graph with relatively few possible edges missing is called a **dense** graph; a graph with few edges relative to the number of its vertices is called a **sparse** graph.

9. A **path** is a sequence of vertices. A **cycle** is a path that consists of at least three vertices that starts and ends with the same vertex. A graph with no cycles is said to be **acyclic**. A **loop** is an edge which begins and ends with the same vertex. A graph without a loop is a **simple**graph.

10. Two vertices are **connected** if there is an edge between them. The **degree** of a vertex is the number of edges incident to it. The **outdegree** in a digraph is the number of edges leaving the vertex and the **indegree** is the number of edges entering the vertex.

11. A graph is **connected** if there is a path between every pair of vertices. If a graph is not connected then it consists of two or more connected pieces called **connected components**.

2 Graph Operations

The six primitive graph operations required to maintain a graph are

1. Insert Vertex
2. Delete Vertex
3. Add Edge
4. Delete Edge
5. Find Vertex
6. Traverse Graph

3 Graph Representations

Some common graph representations are adjacency linked list, adjacency matrix and incidence matrix.

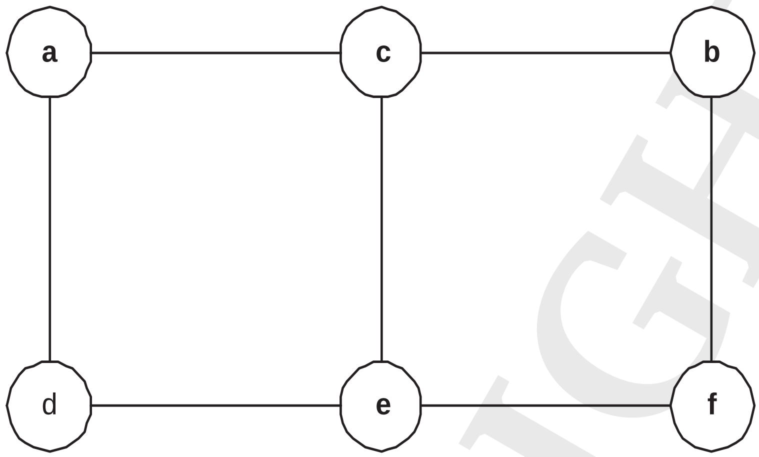

Figure 1: An undirected graph

$$
\begin{array}{cc}
 & \begin{array}{cccccc} a & b & c & d & e & f \end{array} \\
\begin{array}{c} a \\ b \\ c \\ d \\ e \\ f \end{array} &
\begin{bmatrix}
0 & 0 & 1 & 1 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 1 \\
1 & 1 & 0 & 0 & 1 & 0 \\
1 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 1 & 1 & 0 & 1 \\
0 & 1 & 0 & 0 & 1 & 0
\end{bmatrix}
\end{array}
$$

Figure 2: Adjacency matrix of graph in fig. 1

$\boxed{\text{a}} \rightarrow \text{c} \rightarrow \text{d}$

$\boxed{\text{b}} \rightarrow \text{c} \rightarrow \text{f}$

$\boxed{\text{c}} \rightarrow \text{a} \rightarrow \text{b} \rightarrow \text{e}$

$\boxed{\text{d}} \rightarrow \text{a} \rightarrow \text{e}$

$\boxed{\text{e}} \rightarrow \text{c} \rightarrow \text{d} \rightarrow \text{f}$

$\boxed{\text{f}} \rightarrow \text{b} \rightarrow \text{e}$

Figure 3: Adjacency linked list of graph in fig. 1

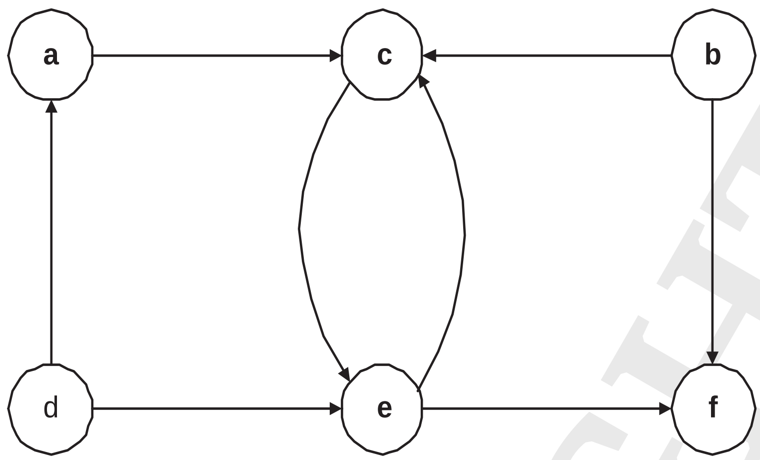

Figure 4: A directed graph

$$
\begin{array}{c c}
 & \begin{array}{cccccc} a & b & c & d & e & f \end{array} \\
\begin{array}{c} a \\ b \\ c \\ d \\ e \\ f \end{array} &
\begin{bmatrix}
0 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 1 & 0 \\
1 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 1 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 0
\end{bmatrix}
\end{array}
$$

Figure 5: Adjacency matrix of graph in fig. 4

$\boxed{\text{a}} \to \text{c}$

$\boxed{\text{b}} \to \text{c} \to \text{f}$

$\boxed{\text{c}} \to \text{e}$

$\boxed{\text{d}} \to \text{a} \to \text{e}$

$\boxed{\text{e}} \to \text{c} \to \text{f}$

$\boxed{\text{f}}$

Figure 6: Adjacency linked list of graph in fig. 4

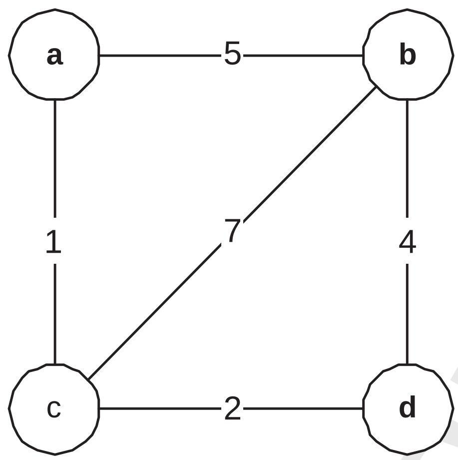

Figure 7: A weighted undirected graph

$$
\begin{array}{c}
\\
a\\
b\\
c\\
d
\end{array}
\begin{array}{c}
\begin{array}{cccc} a & b & c & d \end{array}\\
\begin{bmatrix}
0 & 5 & 1 & \infty\\
5 & 0 & 7 & 4\\
1 & 7 & 0 & 2\\
\infty & 4 & 2 & 0
\end{bmatrix}
\end{array}
$$

Figure 8: Adjacency matrix of graph in fig. 7

$\boxed{\text{a}} \rightarrow$ b,5 $\rightarrow$ c,1
$\boxed{\text{b}} \rightarrow$ a,5 $\rightarrow$ c,7 $\rightarrow$ d,4
$\boxed{\text{c}} \rightarrow$ a,1 $\rightarrow$ b,7 $\rightarrow$ d,2
$\boxed{\text{d}} \rightarrow$ b,4 $\rightarrow$ c,2

Figure 9: Adjacency linked list of graph in fig. 7

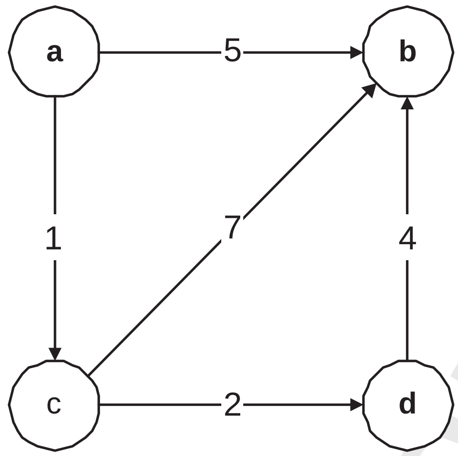

Figure 10: A weighted directed graph

$$
\begin{array}{c}
\begin{array}{cccc} a & b & c & d \end{array} \\
\begin{array}{c} a \\ b \\ c \\ d \end{array}
\begin{bmatrix}
0 & 5 & 1 & \infty \\
\infty & 0 & \infty & \infty \\
\infty & 7 & 0 & 2 \\
\infty & 4 & \infty & 0
\end{bmatrix}
\end{array}
$$

Figure 11: Adjacency matrix of graph in fig. 10

$\boxed{\text{a}} \rightarrow$ b,5 $\rightarrow$ c,1
$\boxed{\text{b}}$
$\boxed{\text{c}} \rightarrow$ b,7 $\rightarrow$ d,2
$\boxed{\text{d}} \rightarrow$ b,4

Figure 12: Adjacency linked list of graph in fig. 10

A graph may also be represented by its incidence matrix. The incidence matrix is a $n \times m$ matrix, where n is the number of vertices and m is the number of edges. For undirected graphs, entry e_{ij} of the incidence matrix is 1 if vertex i is incident with edge j and 0, otherwise. For directed graphs, entry e_{ij} is 1 if the edge is from vertex i to j, -1 if the edge is from j to i and 0 if there is no edge. In some publications, the -1 and 1 are reversed but in this class we will use this convention. For weighted graphs there is no standard. In some publications, the indicator values, 1s and -1s, are replaced with the weights on the edges for weighted graphs and digraphs. Note, that the columns of the incidence matrix are conventionally arranged in lexicographical order by the incident vertices of the edges. We now give incidence matrices for the graphs in fig 1 and fig 4.

$$
\begin{array}{c}
 \\ a \\ b \\ c \\ d \\ e \\ f
\end{array}
\begin{array}{c}
\begin{array}{ccccccc} e1 & e2 & e3 & e4 & e5 & e6 & e7 \end{array} \\
\begin{bmatrix}
1 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 1 & 0 & 0 & 0 \\
1 & 0 & 1 & 0 & 1 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 1 & 1 & 1 \\
0 & 0 & 0 & 1 & 0 & 0 & 1
\end{bmatrix}
\end{array}
$$

Figure 13: Incidence Matrix of graph in fig. 1

$$
\begin{array}{c}
 \\ a \\ b \\ c \\ d \\ e \\ f
\end{array}
\begin{array}{c}
\begin{array}{cccccccc} e1 & e2 & e3 & e4 & e5 & e6 & e7 & e8 \end{array} \\
\begin{bmatrix}
1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\
-1 & 0 & -1 & 0 & 1 & -1 & 0 & 0 \\
0 & 1` & 0 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & -1 & 1 & -1 & 1 \\
0 & 0 & 0 & -1 & 0 & 0 & 0 & -1
\end{bmatrix}
\end{array}
$$

Figure 14: Incidence Matrix of graph in fig. 4

The convention that we will follow for incidence matrices for digraphs and weighted digraphs are given by the piece-wise functions below:

$$
M_{i,e} = \begin{cases} 1 & if\, e_{ij}\, for\ vertex\ j, \\ -1 & if\, e_{ji}\, for\ vertex\ j, \\ 0 & otherwise. \end{cases}
$$

$$
for\ digraphs,\ and\ for\ weighted\ digraphs,
$$

$$
M_{i,e} = \begin{cases} w_{ij} & if\, e_{ij}\, for\ vertex\ j, \\ -w_{ji} & if\, e_{ji}\, for\ vertex\ j, \\ 0 & otherwise. \end{cases}
$$

Problem 1. Give the incidence matrix of the graph in Figure 10 using the conventions described in this handout.

4 Implementation

4.1 Abstract Representation of A Weighted Digraph

```
type Graph
-----------------------------------------------
Integer order
Vertex Ref first {reference to the vertex list}
-----------------------------------------------

type Vertex
------------------------------------------------------
Vertex Ref pNextVertex {reference to the next vertex}
GraphItemType data {data must be comparable}
Integer inDegree
Integer outDegree
Edge Ref pEdge {reference to the edge list}
Integer Processed
------------------------------------------------------

type Edge
-------------------------------------------------------------
Vertex Ref destination {reference to the destination vertex}
Real weight
Edge Ref pNextEdge
-------------------------------------------------------------
```

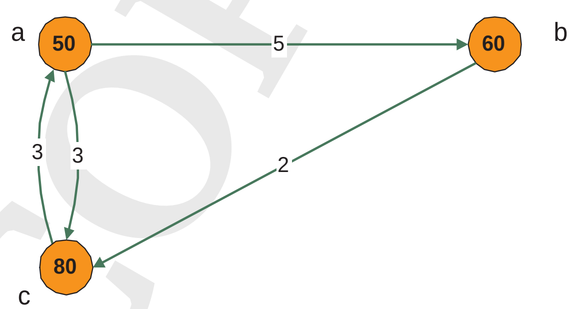

Figure 15: A Weighted Digraph

4.2 Visual Depiction of a Weighted Digraph

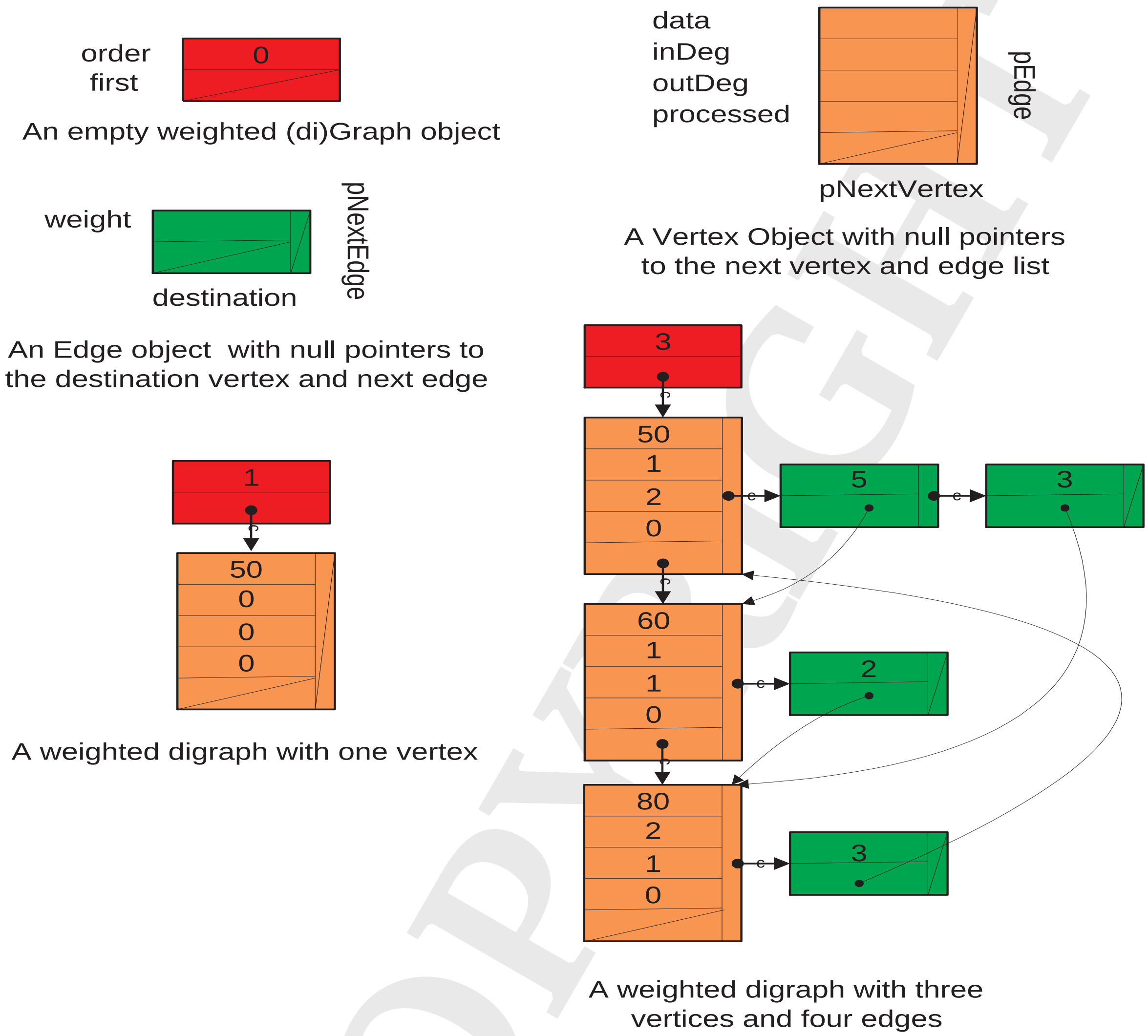

Figure 16: A Visual Depiction of the Weighted Digraph in Figure 15

4.3 Insert Vertex

Listing 1: Vertex Insertion Algorithm

```
ALGORITHM: insertVertex(graph, data)
allocate memory for new vertex
store data in new vertex
initialize metadata elements in new node
increment graph count
if (graph empty)
    set graph first to new node
else
    search for insertion point
    if (inserting before first vertex)
        set graph first to new vertex
    else
        insert new vertex in sequence
    end if
end insertVertex
```

4.4 Delete Vertex

Listing 2: Vertex Deletion Algorithm

```
ALGORITHM: deleteVertex(graph, key)
if (empty graph)
    return false
end if
search for vertex to be deleted
if (not found)
   return false
end if
if (vertex indegree > 0 or outdegree > 0)
   return false
end if
delete vertex
decrement graph count
return true
end deleteVertex
```

Listing 3: Edge Insertion Algorithm

```
ALGORITHM: insertEdge(graph, fromkey, tokey)
allocate memory for new edge
search and set fromvertex
if (from vertex not found)
   return false
end if
search and set tovertex
if (to vertex not found)
   return false
end if
increment fromvertex outdegree
increment tovertex indegree
set edge destination to tovertex
if (fromvertex edge list empty)
   set from vertex first edge to new edge
   set new edge nextedge to null
   return true
end if
find insertion point in edge list
if (insert at beginning of edge list)
    set from vertex first edge to new edge
else
    insert in edge list
end if
return true
end insertEdge
```

Listing 4: Edge Deletion Algorithm

```
ALGORITHM: deleteEdge(graph, fromkey, tokey)
if (empty graph)
   return false
end if
search and set fromvertex to vertex with key equal to from key
if (from vertex not found)
   return false
end if
search and set tovertex
if (fromvertex edge list is null)
   return false
end if
search and find edge with key equal to tokey
if (tokey not found)
    return false
end if
set tovertex to edge destination
delete edge
decrement fromvertex outdegree
decrement tovertex indegree
return true
end deleteEdge
```

The algorithms above assume directed graph. For simple graphs we may use only the outdegree field in both vertices. We may also choose some ordering to determine in which vertex the edge structure is placed to avoid redundancy; that is, to avoid saving the edge in the edge list of both vertices.