Matrices and Graphs

P. Danziger

1 Matrices and Graphs

Definition 1 *Given a digraph G we can represent $G = (\{v_1, v_2, \ldots, v_n\}, E)$ by a matrix $A = (a_{ij})$ where a_{ij} = the number of edges joining v_i to v_j. A is called the inidence matrix of G. If the edges of G.*

Clearly if a digraph, $G = (V, E)$, satisfies $(v_i, v_j) \in E \Rightarrow (v_j, v_i) \in E$ ($A = A^t$) then G is equivalent to an undirected graph.

So G is a graph (as opposed to a digraph) if and only if its incidence matrix is symmetric. (i.e. the matrix is equal to its transpose, $A = A^T$).

Alternatively, we can create a digraph from an undirected graph by replacing each edge $\{u, v\}$ of the undirected graph by the pair of directed edges (u, v) and (v, u).

Definition 2 *A weighted graph is a graph in which each edge has an associated weight or cost.*

In a weighted graph we usually denote that weight of an edge e by $w(e)$, or if $e = u\,v$ we can write $w(u, v)$. If no explicit weight is given we assume that each edge has weight 1 and each non edge weight 0.

Definition 3 *Given a weighted graph G, the adjacency matrix is the matrix $A = (a_{ij})$, where $a_{ij} = w(v_i, v_j)$.*

For most purposes the adjacency matrix and incidence matrix are equivalent.
Note that if G is not connected then the connected components of G form blocks in the adjacency matrix, all other entries being zero.

Theorem 4 *Let G be a graph with connected components $G_1, \ldots, G_k$. Let n_i be the number of vertices in G_i, and let A_i be the adjacency matrix of G_i, then the adjacency matrix of G has the form*

$$\begin{pmatrix} A_1 & 0 & \ldots & 0 \\ 0 & A_2 & & 0 \\ & & \ddots & \vdots \\ & 0 & & A_k \end{pmatrix}$$

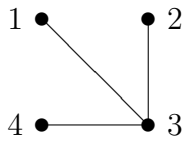
Theorem 5 *Given Two graphs, G and H, with adjacency matrices A and B respectively, $G \cong H$ if and only if there is a permutation of the row and columns of A which gives B.*

Isomorphism is just a relabeling of the rows and columns of the adjacency matrix.

2 Storing Graphs

We wish to be able to store graphs in computer memory. Obviously the incidence matrix or adjacency matrix provide a useful way of holding a graph in an array. One disadvantage to using an array is that it is wasteful, each edge information is stored twice, once as $a[i][j]$ and once as $a[j][i]$. Further just to specify the adjacency matrix requires $O(n^2)$ steps. There are two other (related) standard methods for storing graph in computer memory, adjacency lists and adjacency tables.
We use a list rather than an array, for each vertex we list those vertices adjacent to it. Note that in practice this can be done either as a matrix or a list. If it is done as a matrix then the matrix has size $n \times \Delta$ and is called an adjacency table.
In an adjacency list the vertices adjacent to a cvertex i are stored as a list, usually the end of the list is indicated by a non valid value. Thus for each i $L(i,0)$ gives the adjacent vertex number, $L(i,1)$, gives a pointer to the next list entry, or 0 for none.

Example 6



$$\begin{pmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix}$$

Adjacency Matrix

1	3		
2	3		
3	1	2	4
4	3		

$$\begin{pmatrix} 3 & -1 & -1 \\ 3 & -1 & -1 \\ 1 & 2 & 4 \\ 3 & -1 & -1 \end{pmatrix}$$

Adjacency Table

i	$L(i,0)$	$L(i,1)$
1	3	0
2	3	0
3	1	5
4	3	0
5	2	6
6	4	0

Adjacency List

The maximum number of edges in a simple graph is $O(n^2)$, a graph with relatively few edges, say $o(n^2)$, is called a sparse graph.

2.1 Matrices and Walks

Definition 7 *Given a walk $v_1 e_1 \dots e_{k-1} v_k$ in a graph G, the length of the walk is the number of edges it contains $(k-1)$.*

Problem given a positive integer k, a (directed) graph G and two vertices v_i and v_j in G, find the number of walks from v_i to v_j of length k.

Theorem 8 *If G is a graph with adjacency matrix A, and vertices $v_1, \dots, v_n$, then for each positive integer k the ij^{th} entry of A^k is the number of walks of length k from v_i to v_j.*

Proof Let G be a graph with adjacency matrix A, and vertices $v_1, \ldots, v_n$.
We proceed by induction on k to obtain the result.
Base Case Let $k = 1$. $A^1 = A$.
a_{ij} = the number of edges from v_i to v_j = the number of walks of length 1 from v_i to v_j.
Inductive Step Assume true for k.
Let b_{ij} be the ij^{th} entry of A^k, and let a_{ij} be the ij^{th} entry of A.
By the inductive hypothesis b_{ij} is the number of walks of length k from v_i to v_j.
Consider the ij^{th} entry of $A^{k+1} = AA^k = a_{i1}b_{1j} + a_{i2}b_{2j} + \ldots + a_{in}b_{2n} = \sum_{m=1}^{n} a_{im}b_{mj}$.
Consider $a_{i1}b_{1j}$
= number of walks of length k from v_1 to v_j times the number of walks of length 1 from v_i to v_1
= the number of walks of length $k+1$ from v_i to v_j, where v_1 is the second vertex.
This argument holds for each m, i.e. $a_{it}b_{tj}$ = number of walks from v_i to v_j in which v_m is the second vertex.
So the sum is the number of all possible walks from v_i to v_j. □

There is a related method for finding the shortest path between any specified pair of points. Suppose that the points have been ordered $V = \{v_1, v_2, \ldots, v_n\}$, for each pair i, j from 1 to n let

$$W_0(i,j) = \begin{cases} \text{The weight of the edge } v_i\,v_j & \text{if } v_i\,v_j \in E \\ 0 & \text{if } i = j \\ \infty & \text{if } v_i\,v_j \notin E \end{cases}$$

and for each k from 0 to $n-1$ define

$$W_{k+1}(i,j) = \min(W_k(i,j),\ \ W_k(i,k) + W_k(k,j))$$

Theorem 9 *$W_n(i,j)$ is the length of the shortest path from v_i to v_j.*

Proof: For a given value of k let $S_k = \{v_1, \ldots v_k\}$. We show that $W_k(i,j)$ is the length of the shortest path from v_i to v_j using only the vertices in the subset S_k by induction on k.
Base Case When $k = 0$, $W_0(i,j)$ is the weight of the edge $v_i\,v_j$, if it exists.
Inductive Step Now assume that $W_k(i,j)$ is the length of the shortest path from v_i to v_j using only the vertices in S_k.
Consider $W_{k+1}(i,j)$, if there is a shorter v_iv_j-path using the vertex v_{k+1} as well, it will have length equal to the shortest $v_iv_{k+1}-$path using only vertices from S_k plus the length of the shortest $v_{k+1}v_j-$path using only vertices from S_k, that is $W_k(i,k+1) + W_k(k+1,j)$. On the other hand, if there is no shorter path using v_{k+1}, the value $W_k(i,j)$ will remain unchanged.
Now, $S_n = V$, so the result follows. □
This suggests an algorithm for building the shortest path list.

Initialization:
    Initialise W
Iteration:
    for k = 1 to n
        for i = 1 to n
            for j = 1 to n
                $W_k(i,j) = \min(W_{k-1}(i,j),\ \ W_{k-1}(i,k-1) + W_{k-1}(k-1,j))$

This algorithm has running time $O(n^3)$.