Transformation from Incidence matrix to Adjacency matrix of undirected graph implement in C

Wich Moonsarn, Jeffrey Zhang
Special Mathematics Lecture (Graph Theory)
Nagoya University, Spring 2024

The Incidence matrix and Adjacency matrix are two different ways to store a graph as a matrix. Some matrices might be easier to analyze in some cases than others. In this report, we will propose an algorithm to transform an Incidence matrix into an Adjacency matrix for an undirected graph. C is one of the most popular languages used for implementing algorithms because of its ability to show the working process of the code more apparently.

Definition 1 (Incidence matrix of an undirected graph). For a finite undirected graph $G = (V, E)$ with $V = \{v_1, v_2, ..., v_N\}$ and $E = \{e_1, e_2, ..., e_M\}$. The *Incidence matrix* of G is $N \times M$ Matrix with elements

$$n_{jk} = \begin{cases} 0 \text{ if } x_j \text{ is not an endpoint of } e_k, \\ 1 \text{ if } x_j \text{ is an endpoint of } e_k \\ 2 \text{ if } e_k \text{ is a loop at } x_j \end{cases}$$

Definition 2 (Adjacency matrix). For a finite graph $G = (V, E)$ with $V = \{v_1, v_2, ..., v_N\}$. The *Adjacency matrix* of G is $N \times N$ Matrix with elements

$$a_{jk} = \#\{e \in E | i(e) = (v_j, v_k)\}$$

For an example, figure 1 show undirected graph G with Incidence matrix (figure 2) and Adjacency matrix (figure 11)

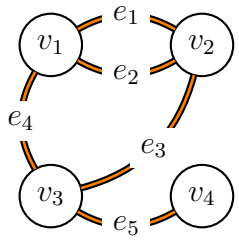

Figure 1: Graph G

Graph G has 4 vertices and 5 edges. So, the Incidence matrix has a size of 4×5 and the Adjacency matrix has a size of 4×4.

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

Figure 2: Incidence matrix of graph G

$$\begin{bmatrix} 0 & 2 & 1 & 0 \\ 2 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

Figure 3: Adjacency matrix of graph G

With the necessary background knowledge introduced, let's restate the question for this report more precisely.

1. **Question Statement**
   Given N vertices labeled from 1 to N and M undirected edges. Print out the Adjacency matrix of a graph that is inputted as an Incidence matrix.

2. **Input**
   The user will be asked to input the total vertices N and total undirected edges M of a graph G. Then the user will be asked to input the corresponding Incidence matrix of the same graph G.

3. **Output**
   Prints out N lines.
   For each line $0 < j \leq N$, the jth line contains N numbers that correspond to the jth row of the Adjacency matrix for graph G.

4. **Constraints**

$$1 \leq N \leq 10^3$$
$$1 \leq M \leq \cdot 10^3$$
Time Limit: 1.00s

5. **Sample Input:**

```
Enter number of vertices : 4
Enter number of edges : 5
Enter Incidence Matrix :
1 1 0 1 0
1 1 1 0 0
0 0 1 1 1
0 0 0 0 1

```

6. **Sample Output**

```
The Adjacency Matrix :
0 2 1 0
2 0 1 0
1 1 0 1
0 0 1 0

```

In the following, we will show an algorithm of $\mathcal{O}(\max(N \cdot M, N \cdot N))$ time complexity. We use the words "number" and "element" interchangeably.

Algorithm

1. **Initialize Inputs** Enter the number of vertices N and the number of edges M for a graph G. We then create two 2d lists A (size $N \times M$) and B (size $N \times N$), which will be used to store the graph G as Incidence matrix and Adjacency matrix correspondingly.

2. **Process the Incidence matrix** Enter $N \cdot M$ numbers separated by space (the numbers can be all entered in one line or multiple), where the jth number corresponds to the number at $((j-1)\%M+1)$th column and $\left\lfloor \frac{j}{N} + 1 \right\rfloor$th row of the Incidence matrix of graph G, 1-indexed. Then store those numbers in A.

3. **Initialize the Adjacency matrix** Set every element in the 2d list of B to 0.

4. **Transformation** For each column of the Incidence matrix A, if the column has two non-zero elements, consider the vertices a and b that are non-zero in this column. We increase the number at ath row, bth column, and the number at ath column, bth row of the Adjacency matrix B by 1. If the column only has one non-zero element, consider the vertex a, and increase the number at ath row, ath column of the Adjacency matrix B by 1. Also if the Incidence matrix has unexpected values for a certain column, then it will print out an error and exit the program.

5. **Output the Adjacency matrix** Print the Adjacency matrix one row at a time.

Let's walk through the algorithm step by step with another example.

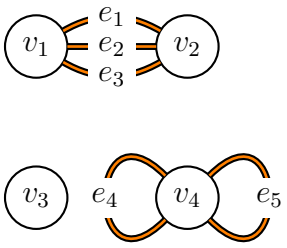

Figure 4: Another graph G visualization

$$\begin{bmatrix} 1 & 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 2 & 2 \end{bmatrix}$$

Figure 5: Incidence matrix of graph G

Initially, the Adjacency matrix is set as empty.

$$\begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

Figure 6: Adjacency matrix of graph G at step 0

We first process e_1 by looking at the first column of the Incidence matrix, and see that vertex v_1 and vertex v_2 are connected by e_1. We then increase the numbers in the Adjacency matrix at 1st row 2nd column and 1st column 2nd row by 1.

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

Figure 7: Adjacency matrix of graph G at step 1

For e_2 and e_3, we experience the same results.

$$\begin{bmatrix} 0 & 2 & 0 & 0 \\ 2 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

Figure 8: Adjacency matrix of graph G at step 2

$$\begin{bmatrix} 0 & 3 & 0 & 0 \\ 3 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

Figure 9: Adjacency matrix of graph G at step 3

For e_4, there is only one non-zero number on the 4th column of the Incidence matrix. This means the vertex v_4 must be a self-loop. So we just increase the number at 4th row and 4th column by 1.

$$\begin{bmatrix} 0 & 3 & 0 & 0 \\ 3 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Figure 10: Adjacency matrix of graph G at step 4

We see that e_5 also goes the same way so finally we have

$$\begin{bmatrix} 0 & 3 & 0 & 0 \\ 3 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 2 \end{bmatrix}$$

Figure 11: Adjacency matrix of graph G at step 5

This completes our algorithm's reconstruction of the graph as an Adjacency matrix.

Algorithm Implementation in C

```c
#include <stdio.h>
int main() {
    //Input of number of vertices and number of edges
    int n_v, n_e;
    printf("Enter number of vertices : ");
    scanf("%i", &n_v);
    printf("Enter number of edges : ");
    scanf("%i", &n_e);
    int N[n_v][n_e], A[n_v][n_v];
    //Input of the Incidence matrix
    printf("Enter Incidence Matrix : \n");
    for (int v = 0; v < n_v; v++){
        for (int e = 0; e < n_e; e++){
            scanf("%i", &N[v][e]);
        }
    }
    //Initialize element of Adjacency matrix
    for (int i = 0; i < n_v; i++){
        for (int j = 0; j < n_v; j++){
            A[i][j] = 0;
        }
    }
```

```
    //Transformation
    for (int e = 0; e < n_e; e++){
        int x1 = -1, x2 = -1;
        for (int v = 0; v < n_v; v++){
            if (N[v][e] == 0){
                continue;
            }
            else if (N[v][e] == 1 && x1 == -1){
                x1 = v;
            }
            else if (N[v][e] == 1 && x1 != -1 && x2 == -1){
                x2 = v;
            }
            else if (N[v][e] == 1 && x1 != -1 && x2 != -1){
                printf("Incidence matrix error");
                return 1;
            }
            else if (N[v][e] == 2 && x1 ==-1){
                x1 = v;
                x2 = v;
            }
            else {
                printf("Incidence matrix error");
                return 1;
            }
        }
        if (x1 != -1 && x2 != -1){
            A[x1][x2] += 1;
            A[x2][x1] += 1;
        }
        if (x1 != -1 && x2 == -1){
            printf("Incidence matrix error");
            return 1;
        }
    }
    //Output of the Adjacency matrix
    printf("The Adjacency Matrix : \n");
    for (int i = 0; i < n_v; i++){
        for (int j = 0; j < n_v; j++){
            printf("%i ", A[i][j]);
        }
        printf("\n");
    }
    printf("\n");
    return 0;
```

```
68 }
```

Note Since the algorithm takes the Incidence matrix as input, which by how we defined it, cannot differentiate directed edges, so this algorithm only works for undirected graphs.