Graph Algorithms

Nirdosh Bhatnagar

1. Introduction

Some useful graph algorithms are discussed. The algorithms we discuss are:

- Shortest path computation from a single node to all other nodes of the network. The popular and efficient algorithm is due to E. W. Dijkstra.
- Shortest paths computation between all pairs of nodes. An efficient algorithm for this computation is due to Floyd, Roy, and Warshall.
- Minimum spanning tree algorithms.
- Connectivity of a graph.

Notation used in describing the algorithms

The algorithms are given in a pseudo-Pascal language. Comments inside the algorithm are italicized. The assignment and logical-and operators are $\leftarrow$ and $\wedge$ respectively. The language elements are **all, begin, do, end, find, for, for all, go to label, if** ... **then, label, stop, such that, while.**

2. Dijkstra's Algorithm

Let $H = (V, E)$ be a directed graph, and $|V| = n$. This graph is assumed to be connected and simple (loop-free). A **cost** (or **weight** or **length**) is defined for each arc $(v_i, v_j) \in E$. Denote it by c_{ij}. It is assumed that $0 \leq c_{ij}$. If $(v_i, v_j) \notin E$, then $c_{ij} = \infty$. The length of a directed path P is given by $\sum_{(ij)\in P} c_{ij}$. The length of path from vertex v_s to vertex v_d is the **shortest path** if the length of path between these vertices is the shortest.

Given a source node v_s, Dijkstra's algorithm computes the shortest path between the node v_s and the nodes in the set $V - \{v_s\}$. The output of this algorithm is specified in terms of a shortest path tree T_s.

The tree T_s is rooted at the node v_s. In this tree, a path from node v_s to node v_t is the shortest path between these two nodes. Further, the arcs in this tree are directed away from the node v_s. The tree can be represented in a **predecessor vector** $\mathcal{T} = (\tau_1, \tau_2, \ldots, \tau_n)$ as follows. The vector element τ_i is the predecessor of node v_i in the tree.

Let the path P_t be equal to$\{v_s, v_1, v_2, \ldots, v_t\}$. In this example, the predecessor of node $v_1 = v_s$. Therefore denote the predecessor of v_1 by τ_1. That is $\tau_1 = v_s$. Similarly, the predecessor of node v_2 is equal to $\tau_2 = v_1$ and so on. Predecessor of the source node v_s is denoted by $\tau_s = \varnothing$ (null-indicator). This information is specified by the predecessor vector.

Dijkstra's algorithm is a **greedy** algorithm. Choosing a path **locally** under this criteria turns out to be the optimal result **globally** for all set of vertices. Consequently, if $P_t = \{v_s, v_1, v_2, \ldots, v_t\}$ is a shortest path from v_s to v_t then $P_k = \{v_s, v_1, v_2, \ldots, v_k\}$ is a shortest path from v_s to v_k for $1 \leq k \leq t$.

Associated with each node $v_j \in V$ in this tree is a scalar quantity d_j. It is the length of the directed path from node v_s to node v_j. Note that the distance d_s of the node v_s from itself is equal to 0. For the scalar d_j to be the shortest path, we have $d_j \leq (d_i + c_{ij})$ for all $(v_i, v_j) \in E$.

Dijkstra's algorithm is an iterative process. The number of iterations is equal to n. In this process a list of vertices L is maintained. This is a list of vertices, which have not yet been chosen to generate the final shortest path tree. In each iteration, two operations are performed. These are node selection and distance update. In the node selection process, a node $v_i \in L$ is chosen such that d_i is the smallest. Recall that A_{v_i} is the adjacency set of vertex v_i. It is the set of all arcs emanating from vertex v_i. The distance update operation performs the updating of the distances d_j where $(v_i, v_j) \in A_{v_i}$ and $v_j \in L$. The distances are output in a vector $\mathcal{D} = (d_1, d_2, \ldots, d_n)$

Dijkstra's Algorithm:

Input: Digraph $H = (V, E)$, $0 \leq c_{ij}$ for all arcs $(v_i, v_j) \in E$, and the source node v_s

Output: The rooted shortest path tree T_s specified by the predecessor vector $\mathcal{T}$, and the distance vector $\mathcal{D}$.

```
begin
    (Initialization)
    $d_s \leftarrow 0$
    $d_j \leftarrow \infty$, for all nodes $v_j \in V - \{v_s\}$
    $\tau_s \leftarrow \varnothing$
    $L \leftarrow V$
    while $L \neq \varnothing$ do
    begin
        (Select vertex)
        find $v_i \in L$ such that $d_i = \min\{d_j \mid v_j \in L\}$
        $L \leftarrow L - \{v_i\}$
        if $L = \varnothing$, stop ( the computation is complete )
        (Update distance)
        for all $((v_i, v_j) \in A_{v_i}) \wedge (v_j \in L)$ do
        begin
            if $(d_i + c_{ij}) < d_j$ then
            begin
                $d_j \leftarrow (d_i + c_{ij})$
                $\tau_j \leftarrow v_i$
            end (end if statement)
        end (end for loop)
    end (end while loop)
end (end of Dijkstra's algorithm)
```

The computational complexity of this algorithm is $O\left(n^2\right)$.

Example: We will determine the shortest path tree, and the shortest distance vector for the digraph $H = (V, E)$. The graph is specified as follows. $V = \{1, 2, 3, 4, 5, 6\}$, $E = \{(1,2), (1,3), (2,3), (2,4), (3,4), (3,5), (4,6), (5,3), (5,4), (5,6), (6,4)\}$. The lengths of these arcs are $c_{12} = 3, c_{13} = 5, c_{23} = 1, c_{24} = 2, c_{34} = 6, c_{35} = 2, c_{46} = 5, c_{53} = 7, c_{54} = 3, c_{56} = 1$, and $c_{64} = 4$. We will assume that the source node is node 1. For convenience we list the adjacency sets of the vertices. These are: $A_1 = \{(1,2), (1,3)\}, A_2 = \{(2,3), (2,4)\}, A_3 = \{(3,4), (3,5)\}, A_4 = \{(4,6)\}, A_5 = \{(5,3), (5,4), (5,6)\}$, and $A_6 = \{(6,4)\}$. The distance and predecessor vectors to be determined are $\mathcal{D} = (d_1, d_2, d_3, d_4, d_5, d_6)$, and $\mathcal{T} = (\tau_1, \tau_2, \tau_3, \tau_4, \tau_5, \tau_6)$ respectively. As per Dijkstra's algorithm, these are obtained via following steps.

Initialization:

$\mathcal{D} \leftarrow (0, \infty, \infty, \infty, \infty, \infty)$, $\tau_1 \leftarrow \varnothing$, $L \leftarrow \{1,2,3,4,5,6\}$

Iteration 1 :

Select vertex: $\min(d_1, d_2, d_3, d_4, d_5, d_6) = d_1 = 0$. Select vertex 1. $L \leftarrow \{2,3,4,5,6\}$.
Update distance: Since $A_1 = \{(1,2), (1,3)\}$ update d_2 and d_3.
$d_2 \leftarrow \min(\infty, 0+3)$, that is $d_2 \leftarrow 3$. Also $\tau_2 \leftarrow 1$.
$d_3 \leftarrow \min(\infty, 0+5)$, that is $d_3 \leftarrow 5$. Also $\tau_3 \leftarrow 1$.
$\mathcal{D} = (0, 3, 5, \infty, \infty, \infty)$
The arc set$\{(1,2), (1,3)\}$ belongs to the arc set of the generated tree.

Iteration 2 :

Select vertex: $\min(d_2, d_3, d_4, d_5, d_6) = d_2 = 3$. Select vertex 2. $L \leftarrow \{3,4,5,6\}$.
Update distance: Since $A_2 = \{(2,3), (2,4)\}$ update d_3 and d_4.
$d_3 \leftarrow \min(5, 3+1)$, that is $d_3 \leftarrow 4$. Also $\tau_3 \leftarrow 2$.
$d_4 \leftarrow \min(\infty, 3+2)$, that is $d_4 \leftarrow 5$. Also $\tau_4 \leftarrow 2$.
$\mathcal{D} = (0, 3, 4, 5, \infty, \infty)$
The arc set$\{(1,2), (2,3), (2,4)\}$ belongs to the arc set of the generated tree.

Iteration 3 :

Select vertex: $\min(d_3, d_4, d_5, d_6) = d_3 = 4$. Select vertex 3. $L \leftarrow \{4,5,6\}$.
Update distance: Since $A_3 = \{(3,4), (3,5)\}$ update d_4 and d_5.
$d_4 \leftarrow \min(5, 4+6)$, d_4 and τ_4 remain unchanged.
$d_5 \leftarrow \min(\infty, 4+2)$, that is $d_5 \leftarrow 6$. Also $\tau_5 \leftarrow 3$.
$\mathcal{D} = (0, 3, 4, 5, 6, \infty)$
The arc set$\{(1,2), (2,3), (2,4), (3,5)\}$ belongs to the arc set of the generated tree.

Iteration 4 :

Select vertex: $\min(d_4, d_5, d_6) = d_4 = 5$. Select vertex 4. $L \leftarrow \{5,6\}$.
Update distance: Since $A_4 = \{(4,6)\}$ update d_6.
$d_6 \leftarrow \min(\infty, 5+5)$, that is $d_6 \leftarrow 10$. Also $\tau_6 \leftarrow 4$.
$\mathcal{D} = (0, 3, 4, 5, 6, 10)$
The arc set$\{(1,2), (2,3), (2,4), (3,5), (4,6)\}$ belongs to the arc set of the generated tree.

Iteration 5 :

Select vertex: $\min(d_5, d_6) = d_5 = 6$. Select vertex 5. $L \leftarrow \{6\}$.
Update distance: Since $A_5 = \{(5,3), (5,4), (5,6)\}$ update d_6.
$d_6 \leftarrow \min(10, 6+1)$, that is $d_6 \leftarrow 7$. Also $\tau_6 \leftarrow 5$.
$\mathcal{D} = (0, 3, 4, 5, 6, 7)$
The arc set$\{(1,2), (2,3), (2,4), (3,5), (5,6)\}$ belongs to the arc set of the generated tree.

Iteration 6 :

Select vertex: $\min(d_6) = d_6 = 7$. Select vertex 6. $L \leftarrow \varnothing$.
Stop since $L = \varnothing$.

End of Dijkstra's algorithm.

The final distance and predecessor vectors are $\mathcal{D} = (0, 3, 4, 5, 6, 7)$ and $\mathcal{T} = (\varnothing, 1, 2, 2, 3, 5)$. The arc set of the corresponding shortest path tree is $\{(1,2), (2,3), (2,4), (3,5), (5,6)\}$.
□

3. All-pairs Shortest Path Algorithms

We will develop two algorithms for finding shortest paths between all pairs of nodes. Let $H = (V, E)$ be a directed graph, and $|V| = n$. For simplicity, the nodes are labeled from 1 through n. Costs of the edges (i, j) are specified in the $n \times n$ cost matrix $C = [c_{ij}]$. The cost $c_{ii} = 0, 1 \leq i \leq n; c_{ij}$ if $(i, j) \in E$; and the cost is ∞ if $(i, j) \notin E$. For simplicity, it is assumed that c_{ij} is positive for, $1 \leq i, j \leq n$. From each of the n nodes, shortest paths to other $(n-1)$ nodes have to be computed. Let d_{ij} be the length of the shortest path from node i to node j. Evidently $d_{ii} = 0$. Let D be a $n \times n$ matrix, where $D = [d_{ij}]$

Also define a $n \times n$ matrix $D^{(m)} = \left[d_{ij}^{(m)}\right]$, where $d_{ij}^{(m)}$ is the length of the shortest path from node i node j with no more than m arcs between the two nodes. Observe that $d_{ij}^{(1)} = c_{ij}$ and $d_{ij}^{(n-1)} = d_{ij}$ for $0 \leq i, j \leq n$. Then d_{ij}'s can be computed from the following recursive relationship

$$\begin{aligned} d_{ij}^{(1)} &= c_{ij} \\ d_{ij}^{(m)} &= \min_{1 \leq k \leq n} \left\{ d_{ik}^{(m-1)} + c_{kj} \right\} \text{ for } 2 \leq m \leq (n-1) \end{aligned}$$

Observe that the minimization operation in the above recursion is similar to matrix multiplication. If in matrix multiplication, the scalar multiplication is replaced by scalar addition, and scalar addition operation is replaced by minimization, then the above recursion can be stated in terms of a modified type of matrix multiplication. Denote this new matrix multiplication by $\otimes$. If $R = [r_{ij}], S = [s_{ij}]$, and $T = [t_{ij}]$, then

$$\begin{aligned} R &= S \otimes T \\ r_{ij} &= \min_{1 \leq k \leq n} \{s_{ik} + t_{kj}\} \text{ for } 1 \leq i, j \leq n \end{aligned}$$

Therefore the shortest path computation can now be stated in terms of this modified matrix operation.

$$\begin{aligned} D^{(1)} &= C \\ D^{(m)} &= D^{(m-1)} \otimes C \text{ for } 2 \leq m \leq (n-1) \\ D^{(n-1)} &= D \end{aligned}$$

It can be observed that $D^{(2)} = C \otimes C$, $D^{(3)} = C \otimes (C \otimes C)$, and so on. The complexity of these matrix operations can be computed as follows. Each modified matrix multiplication can be performed in $O(n^3)$ operations and $(n-1)$ times matrix exponentiation can be performed in $O(\log n)$ steps. Therefore the computational complexity of determining the matrix D is $O(n^3 \log n)$.

```
Modified matrix multiplication algorithm:
Input: Digraph $H = (V, E)$, and the cost matrix $C$.
Output: The shortest path distance matrix $D = D^{(n-1)}$.
begin
    $D^{(1)} = C$
    $D^{(m)} = D^{(m-1)} \otimes C$ for $2 \leq m \leq (n-1)$
end (end of modified matrix multiplication algorithm)
```

Example: We will determine the shortest path lengths between all pairs of a digraph $H = (V, E)$, where $V = \{1, 2, 3, 4\}$, $E = \{(1, 2), (1, 4), (2, 3), (2, 4), (3, 2), (3, 4), (4, 1), (4, 3)\}$.

The cost matrix is given by

$$C = \begin{bmatrix} 0 & 3 & \infty & 2 \\ \infty & 0 & 1 & 4 \\ \infty & 5 & 0 & 7 \\ 9 & \infty & 6 & 0 \end{bmatrix}$$

Then

$$\begin{aligned} D^{(1)} &= C \\ D^{(2)} &= C \otimes C = \begin{bmatrix} 0 & 3 & 4 & 2 \\ 13 & 0 & 1 & 4 \\ 16 & 5 & 0 & 7 \\ 9 & 11 & 6 & 0 \end{bmatrix} \\ D &= D^{(2)} \otimes D^{(2)} = D^{(2)} \end{aligned}$$

□

An all-pairs shortest path algorithm due to Floyd, Roy, and Warshall is more efficient than the above algorithm. It has a complexity of $O\left(n^3\right)$ steps. The basic idea behind this algorithm is as follows. Assume that the elements of the cost-matrix are all greater than or equal to zero. Let i and j be any two vertices in a digraph. Perform the following operations successively:

$$\begin{aligned} d_{ij}^{(1)} &= c_{ij} \\ d_{ij}^{(m+1)} &= \min\left\{ d_{ij}^{(m)}, d_{im}^{(m)} + d_{mj}^{(m)} \right\} \text{ for } 1 \le m \le n \end{aligned}$$

Then

$$d_{ij} = d_{ij}^{(n+1)} \qquad i, j \in V$$

The shortest paths can be kept track of via a matrix P of size $n \times n$. Let $P = [p_{ij}]$, where the element p_{ij} is generated as follows. Initialize p_{ij}'s as

$$p_{ij} = \begin{cases} j & \text{if } c_{ij} \neq \infty \\ 0 & \text{if } c_{ij} = \infty \end{cases}$$

While updating d_{ij}'s let

$$p_{ij} = \begin{cases} p_{im} & \text{if } d_{ij} > (d_{im} + d_{mj}) \\ p_{ij} & \text{if } d_{ij} < (d_{im} + d_{mj}) \end{cases}$$

The shortest path can then be extracted from p_{ij}'s. Observe that p_{ij} is the second vertex along the shortest path form vertex i to vertex j. If $p_{ij} = 0$ at the end of the algorithm, then a path does not exist from a node i to node j.

Floyd-Roy-Warshall Algorithm:
Input: Digraph $H = (V, E)$, and the cost matrix C.
Output: The shortest path matrix D and the path matrix P.

```
begin
    (Initialize )
    $d_{ij} \leftarrow c_{ij}$, for all $i, j \in V$
    $p_{ij} = \begin{cases} j & \text{if } c_{ij} \neq \infty \\ 0 & \text{if } c_{ij} = \infty \end{cases}$ , **for all** $i, j \in V$
    **for** $m = 1$ **to** $n$ **do**
        **for** $i = 1$ **to** $n$ **do**
            **for** $j = 1$ **to** $n$ **do**
                **if** $d_{ij} > (d_{im} + d_{mj})$
                **begin**
                    $d_{ij} \leftarrow (d_{im} + d_{mj})$
                    $p_{ij} \leftarrow p_{im}$
                **end** (*end if statement*)
            **end** (*end jfor-loop*)
        **end** (*end ifor-loop*)
    **end** (*end mfor-loop*)
**end** (*end Floyd-Roy-Warshall algorithm* )
```

At the end of this algorithm, the shortest path between the vertices i and j is recovered from the matrix P. Let this shortest path be given by the sequence of vertices $i, i_1, i_2, \ldots, i_q, j$, Then

$$i_1 = p_{ij}, i_2 = p_{i_1 j}, i_3 = p_{i_2 j}, \ldots, j = p_{i_q j}$$

Therefore the shortest path from vertex i to vertex j is obtained by examining elements of the jth column of the matrix P. The complexity of Floyd, Roy, and Warshall's algorithm is $O\left(n^3\right)$.

4. Minimum spanning trees

Let $G = (V, E)$ be an undirected loop-free connected graph, where V is the set of vertices, and E is the set of edges. Let the cardinality of these sets be $|V| = n$ and $|E| = m$. Also each edge of the graph is associated with a weight (cost or length). If the edge is non-existent, then its cost is assumed to be infinite. Let the cost of the edge (i, j) be $c_{ij} \geq 0$. The costs are specified in a cost matrix C, where $c_{ij} \geq 0$ for all edges $(v_i, v_j) \in E$, and $c_{ij} = \infty$ for $(v_i, v_j) \notin E$. Assume that this graph is connected. A subgraph of graph G is a spanning tree of the graph if the number of edges in this subgraph is equal to $(n-1)$. Denote this subgraph by T. In a tree a pair of vertices have a unique path.

A spanning tree of graph G is a **minimum spanning tree** (MST) if the sum of the weights of the edges is minimum.

Algorithms to generate MST's are greedy. We discuss two algorithms to generate MST.

- Prim's algorithm.
- Kruskal's algorithm.

These algorithms are so named after the mathematicians Prim and Kruskal, though they were not the first ones to invent them. Also note that the MST's are not unique.

4.1. Prim's Algorithm.

Prim's algorithm is given below.

Prim's Algorithm:
Input: Undirected loop-free connected graph $G = (V, E)$, the cost matrix $C = [c_{ij}]$.
Output: The minimum spanning tree T. The edge set T_E of the MST.
begin

```
    (Initialization)
    Select any arbitrary vertex $v_1 \in V$.
    Let $Q \leftarrow \{v_1\}, R \leftarrow (V - Q)$, and $T_E \leftarrow \varnothing$.
    for $i = 1$ to $(n - 1)$ do
    begin
        select an edge $(v_j, v_k)$ such that $c_{jk} = \min\{c_{ab} \mid v_a \in Q, v_b \in R\}$
        (if there is a tie in minimization pick arbitrarily)
        $Q \leftarrow Q \cup \{v_k\}$
        $R \leftarrow (V - Q)$
        $T_E \leftarrow T_E \cup (v_j, v_k)$
    end (end for loop)
end (end Prim's algorithm)
```

Observe that in each iteration of this algorithm, a single tree is generated. This algorithm was initially proposed by V. Jarnik (1930). It was later independently discovered by R. C. Prim (1957). The complexity of this algorithm is $O\left(n^2\right)$.

Example: We will determine the MST of the graph $G = (V, E)$. The graph is specified as follows. $V = \{1, 2, 3, 4, 5, 6\}$, $E = \{(1, 2), (1, 3), (2, 3), (2, 4), (3, 4), (3, 5), (4, 6), (5, 4), (5, 6)\}$. The lengths of these arcs are $c_{12} = 3, c_{13} = 5, c_{23} = 1, c_{24} = 5, c_{34} = 6, c_{35} = 2, c_{46} = 4, c_{54} = 3$, and $c_{56} = 1$. Since $|V| = 6$, the number of edges in the MST is equal to 5.

Initialization:
$Q \leftarrow \{1\}, R \leftarrow \{2, 3, 4, 5, 6\}$, and $T_E \leftarrow \varnothing$.

Iteration 1:
Selected least cost edge $= (1, 2), c_{12} = 3$
$Q = \{1, 2\}, R = \{3, 4, 5, 6\}$, and $T_E = \{(1, 2)\}$

Iteration 2:
Selected least cost edge $= (2, 3), c_{23} = 1$
$Q = \{1, 2, 3\}, R = \{4, 5, 6\}$, and $T_E = \{(1, 2), (2, 3)\}$

Iteration 3:
Selected least cost edge $= (3, 5), c_{35} = 2$
$Q = \{1, 2, 3, 5\}, R = \{4, 6\}$, and $T_E = \{(1, 2), (2, 3), (3, 5)\}$

Iteration 4:
Selected least cost edge $= (5, 6), c_{56} = 1$
$Q = \{1, 2, 3, 5, 6\}, R = \{4\}$, and $T_E = \{(1, 2), (2, 3), (3, 5), (5, 6)\}$

Iteration 5:
Selected least cost edge $= (4, 5), c_{45} = 3$
$Q = \{1, 2, 3, 4, 5, 6\}, R = \varnothing$, and $T_E = \{(1, 2), (2, 3), (3, 5), (5, 6), (4, 5)\}$

End of Prim's MST

The total weight of this MST is equal to 10.

4.2. Kruskal's Algorithm.

Kruskal's algorithm generates a MST by initially sorting edges in increasing order of costs. Edges are added to a forest of trees if the candidate edge does not create a cycle. The algorithm is terminated once a tree with $(n - 1)$ edges is created. The formal algorithm for finding the MST via Kruskal's algorithm is given below.

Kruskal's Algorithm:
Input: Undirected loop-free connected graph $G = (V, E)$, the cost matrix $C = [c_{ij}]$.
Output: The minimum spanning tree T. The edge set T_E of the MST.

```
begin
```

```
    (*Initialization*)
    Sort the $m$ edges in increasing order of cost.
    Edges with the identical value are ranked arbitrarily.
    Let $T_E \leftarrow \varnothing$
    **while** $|T_E| \leq (n-1)$ **do**
    **begin**
        Examine the first unexamined edge in the ascending-ordered list of edges.
        If this edge $e$ does not form a cycle with edges in $T_E$, then $T_E \leftarrow T_E \cup e$
    **end** (*end while loop*)
**end** (*end Kruskal's algorithm*)
```

Observe that some iterations of this algorithm may generate a forest (a group of trees). This algorithm was independently discovered by Kruskal (1956) and by H. Loberman and A. Weinberger (1957). The complexity of this algorithm generally depends upon the data structure of the variables. Prim's algorithm generally outperforms Kruskal's algorithm. We illustrate Kruskal's algorithm via an example.

Example: We will obtain the MST of the graph of the previous example.

Initialization:

We first sort the edge list in order of increasing costs. The list is

$$\{(2,3),(5,6),(3,5),(1,2),(4,5),(1,3),(2,4),(4,6),(3,4)\}$$

Let $T_E \leftarrow \varnothing$.

Iteration 1:

Examine the edge $(2,3)$. Evidently it does not form a circuit with T_E.

$T_E = \{(2,3)\}$

Iteration 2:

Examine the edge $(5,6)$. Evidently it does not form a circuit with T_E.

$T_E = \{(2,3),(5,6)\}$

Iteration 3:

Examine the edge $(3,5)$. Evidently it does not form a circuit with T_E.

$T_E = \{(2,3),(5,6),(3,5)\}$

Iteration 4:

Examine the edge $(1,2)$. Evidently it does not form a circuit with T_E.

$T_E = \{(2,3),(5,6),(3,5),(1,2)\}$

Iteration 5:

Examine the edge $(4,5)$. Evidently it does not form a circuit with T_E.

$T_E = \{(2,3),(5,6),(3,5),(1,2),(4,5)\}$.

$|T_E| = 5$.

T_E is the MST.

End of Kruskal's MST

The total weight of this MST is equal to 10.

5. Connectivity of a Graph

An undirected graph G is connected if and only if there is a path between every pair of vertices, otherwise it is disconnected. The connectivity of a graph can be described by its **connectedness matrix**. If $|V| = n$, then this matrix is a square matrix of size n. The (i,j)th element of this matrix is equal to 1, if vertex v_i is connected to vertex v_j, irrespective of the number of arcs that are required in getting there.

Let the number of disconnected components of a graph $G = (V, E)$ be equal to d. Then $G = \bigcup_{i=1}^{d} G_i$, where $G_i = (V_i, E_i)$, $V = \bigcup_{i=1}^{d} V_i$, and $E = \bigcup_{i=1}^{d} E_i$. Further $V_i \cap V_j = \varnothing$ and $E_i \cap E_j = \varnothing$ for all $i \neq j, 1 \leq i, j \leq d$. Evidently, the graph is connected if d is equal to 1.

The basic description of the algorithm to find the connectedness of a graph is as follows. The algorithm starts with the adjacency matrix of the graph. The basis of the algorithm is the *fusion* of adjacent vertices. It initially starts with any vertex in the graph. The algorithm then fuses all the vertices which are adjacent to it into a single vertex. The fused vertex is again fused with its adjacent vertices. This process is repeated till it cannot fuse anymore vertices. That is a connected component of graph is fused into a single fused-vertex. At the end of this fusion operation, the algorithm checks to see if all the vertices of the graph G are fused. The algorithm terminates, if all the vertices have been fused, otherwise it starts a new vertex fusion process. If the number of fused vertices d is equal to 1, then the graph G is connected, otherwise it is disconnected.

The fusion of the vertex v_j to vertex v_i is achieved by logical addition of the jth row to the ith row and the jth column to the ith column. Denote the logical addition operator by $\oplus$, where $0 \oplus 0 = 0, 0 \oplus 1 = 1, 1 \oplus 0 = 1$, and $1 \oplus 1 = 1$. After this fusion operation the jth column and jth row are discarded from the matrix. Generally, the discarded rows and columns in the original matrix are marked as such. This tagging ensures that the corresponding elements of the matrix are not considered in the successive fusion operations.

Connectedness Algorithm:
Input: An undirected graph $G = (V, E)$.
Output: The vertex sets $V_i, 1 \leq i \leq d$, of the disconnected components.

```
begin
    (Initialize )
    $Y \leftarrow A$ (Initialize $Y$ by the adjacency matrix $A$. $Y = [y_{ij}]$)
    $d \leftarrow 0$
    $i \leftarrow 1$
    label A
    $d \leftarrow d + 1$
    $V_d \leftarrow \{v_i\}$
    while $((y_{ij} = 1) \wedge (i < j))$
    begin
        Logically add row $j$ to row $i$, and column $j$ to column $i$ of matrix $Y$.
        Discard row $j$ and column $j$ of matrix $Y$.
        The phrase disc is entered in the elements of these columns and rows
        of the matrix $Y$.
        $V_d \leftarrow V_d \cup \{v_j\}$
    end (end of while loop)
    Find a row $k > i$ which has not been discarded.
    if such a row has been found $i \leftarrow k$, go to label A.
    else stop
end (end of Connectedness algorithm )
```

If $d = 1$ then the graph G is connected. The complexity of this connectedness algorithm is $O(n^2)$. The steps in this algorithm are clarified by the following example.

Example: Determine the connectivity of the graph G specified by the adjacency matrix A.

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

Initialization:
$Y \leftarrow A$, $d \leftarrow 0$, $i \leftarrow 1$
Step 1:
$d \leftarrow 1$, $V_1 \leftarrow \{v_1\}$.
$y_{12} = 1$. Therefore logically add row 2 to row 1, and column 2 to column 1 of matrix Y. Discard row 2 and column 2 of matrix Y. Also$V_1 = \{v_1, v_2\}$. At the end of this step, the Y matrix is:

$$Y = \begin{bmatrix} 1 & disc & 0 & 1 & 0 \\ disc & disc & disc & disc & disc \\ 0 & disc & 0 & 0 & 1 \\ 1 & disc & 0 & 0 & 0 \\ 0 & disc & 1 & 0 & 0 \end{bmatrix}$$

Step 2:
$y_{14} = 1$. Therefore logically add row 4 to row 1, and column 4 to column 1 of matrix Y. Discard row 4 and column 4 of matrix Y. Also $V_1 = \{v_1, v_2, v_4\}$. At the end of this step, the Y matrix is:

$$Y = \begin{bmatrix} 1 & disc & 0 & disc & 0 \\ disc & disc & disc & disc & disc \\ 0 & disc & 0 & disc & 1 \\ disc & disc & disc & disc & disc \\ 0 & disc & 1 & disc & 0 \end{bmatrix}$$

The first component of the graph G has been determined.
Step 3:
The row 3 has not yet been discarded. $i \leftarrow 3$.
$d \leftarrow 2$, $V_2 \leftarrow \{v_3\}$.
$y_{35} = 1$. Therefore logically add row 5 to row 3, and column 5 to column 3 of matrix Y. Discard row 5 and column 5 of matrix Y. Also $V_2 = \{v_3, v_5\}$. At the end of this step, the Y matrix is:

$$Y = \begin{bmatrix} 1 & disc & 0 & disc & disc \\ disc & disc & disc & disc & disc \\ 0 & disc & 1 & disc & disc \\ disc & disc & disc & disc & disc \\ disc & disc & disc & disc & disc \end{bmatrix}$$

The algorithm terminates, since all the rows have been exhausted.
End of Connectedness algorithm.
At the end of this algorithm, the set of vertices of the connected components are $V_1 = \{v_1, v_2, v_4\}$ and $V_2 = \{v_3, v_5\}$. Therefore the graph G is disconnected, and has two components.