Warshall's Algorithm

Warshall's algorithm is an efficient method for computing the transitive closure of a relation. Warshall's algorithm takes as input the matrix M_R representing the relation R, and outputs the matrix M_{R^*} of the relation R^*, the transitive closure of R. Below are two version of the algorithm. The first is taken from Rosen [1], and the second is a slight variation of the algorithm.

Warshall Algorithm 1

Warshall(M_R: $n \times n$ 0-1 matrix)
$W := M_R$ ($W = [w_{ij}]$)
for(k=1 to n) {
  for(i=1 to n) {
    for(j=1 to n) {
      $w_{ij} = w_{ij} \vee (w_{ik} \wedge w_{kj})$
      }
    }
  }
return W

Warshall Algorithm 2

Warshall(M_R: $n \times n$ 0-1 matrix)
$W := M_R$ ($W = [w_{ij}]$)
for(k=1 to n){
  for(i=1 to n) {
    if(w_{ik}=1) {
      for(j=1 to n) {
        $w_{ij} = w_{ij} \vee w_{kj})$
        }
      }
    }
  }
return W

Let's examine the first algorithm closely. When the inner *for* loop is being executed, the only value which is changing is j. Notice that the value of w_{ik} does not depend on j. Thus, during each iteration of the inner loop, w_{ik} is constant. If $w_{ik} = 0$, then

$$w_{ij} = w_{ij} \vee (w_{ik} \wedge w_{kj}) = w_{ij} \vee (0 \wedge w_{kj}) w_{ij} \vee (0) = w_{ij},$$

and if $w_{ik} = 1$, then

$$w_{ij} = w_{ij} \vee (w_{ik} \wedge w_{kj}) = w_{ij} \vee (1 \wedge w_{kj}) = w_{ij} \vee w_{kj},$$

for each value of j. Thus, the values of w_{ij} remain unchanged if $w_{ik} = 0$, and become $w_{ij} \vee w_{kj}$ if $w_{ik} = 1$. It is this observation which leads to the second algorithm. Notice that when $w_{ik} = 1$, we are simply replacing the ith row of the matrix with OR of the ith and kth rows of the matrix.

In words, the second version of the algorithm says the following: To compute the ith row of W_k, look at the kth column of the ith row of W_{k-1}. If this entry is a "0", the ith row of W_k is the ith row of W_{k-1}. If this entry is a "1", the ith row of W_k is the OR of the ith and kth rows of W_{k-1}.

A comparison of the running times of the algorithms is easy. The first algorithm requires $2n^3$ bit operations, and the second no more than $n^2 + n^3$ (if we count comparison as a bit operation). Thus, the second algorithm is in general slightly faster, and if implemented in a very special way, potentially much faster.

Example: The matrix M_R below is the matrix representation for a relation R. Find the matrix representation M_{R^*} of R^*, the transitive closure of R.

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

Solution We know that $W_0 = M_R$. To compute W_1, we notice that in the first column of W_0, there are "1"s in rows 1 and 4. Thus, we replace rows 1 and 4 with the OR of themselves and row 1. We obtain

$$W_1 = \begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 0 \end{bmatrix}.$$

To compute W_2, we notice that in the second column of W_1, there is a "1" in row 3. Thus, we replace row 3 with the OR of rows 3 and 2, obtaining

$$W_2 = \begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 0 \end{bmatrix}.$$

To compute W_3, we notice that in the third column of W_2, there is a "1" in every row. Thus, we replace each row with the OR of itself and row 3, obtaining

$$W_3 = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{bmatrix}.$$

To compute W_4, we notice that in the fourth column of W_3, there is a "1" in every row. Thus, we replace each row with the OR of itself and row 4, obtaining

$$W_4 = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{bmatrix} = M_{R^*} \quad \blacksquare$$

For more details on relations and transitive closures, consult chapter 6 of [1].

References

[1] Rosen, Kenneth H., *Discrete Mathematics and its Applications*, Third Edition, McGraw-Hill, Inc, 1994.