STUDIA UNIV. BABEŞ-BOLYAI, INFORMATICA, VOLUME XLIV, NUMBER 1, 1999

AN IMPROVEMENT OF THE ROY-FLOYD ALGORITHM

GABRIELA SERBAN

Rezumat. In acest articol este prezentat algoritmul Roy-Floyd pentru determinarea drumurilor de cost minim }I maxim in grafe. Se justifica extinderea aplicarii algoritmului si in cazul deplasarilor pe caroiaje, din punctul de vedere al eficientei implementarii (atat ca timp de executie, cat si ca memorie ocupata). Sunt prezentate cateva exemple semnificative.

1 Introduction

A great variety of problems required by the practice are in fact graph problems, or which can be reduced to graph problems. Many times it is necessary to find the shortest or longest path between two nodes in a graph.

The paper presents below a possibility to use the Roy-Floyd Algorithm (finding the minimum value path between two nodes of a graph) in certain problems and also the advantages of this approach.

2 The ROY-FLOYD Algorithm

Let us consider a directed graph with n nodes, given by the cost matrix. For each pair of nodes (i, j) we have to find the minimum value path from i to j, as well as the total cost of the path.

The cost matrix is a matrix A(n,n) where:

$$a(i,j) = \begin{cases} c & \text{if}\,(i,j)\text{ is an arc and its cost is } c \\ 0 & \text{if } i = j \\ +\infty & \textit{if}\,(i,j)\text{ is not an arc} \end{cases}$$

The ***dynamic programming***, is used as a solving method - results from the following optimality principle: "if the shortest path from i to j pass through the node k, then the paths from i to k and from k to j are the shortest".

In what follows we recall the Roy-Floyd algorithm (see aiso [3]).

Step 1 we compute the cost matrix A

Step 2 for each intermediary node $k \in [1,n]$ do

```
Step 2.1    for each nodes $i,j\in[1,n]$ do
                $a(i,j):=\min\{a(i,j),a(i,k)+a(k,j)\}$
Step 3  for each nodes $i,j\in[1,n]$, $a(i,j)$ will represent the minimal cost of
        the path from i to j (if $a(i,j)=+\infty$, there is no path between i and j)
```

Remark

For determining the shortest path between 2 nodes, the step 2.1 of the algorithm become step 2.1'; and on add step 4.

```
Step 2.1'   for each  nodes $i,j\in[1,n]$ do
            if $a(i,j)>a(i,k)+a(k,j)$ then
                $a(i,j):= a(i,k)+a(k,j)$
                $c(i,j):=k$
```

where $c(i,j)$ represents the intermediary node through which the shortest path from i to j passes.

```
Step 4  the shortest path determination is made by recursion:
            if $c(i,j)=0$ then i is a node from the path
            else
                determine the shortest path from i to $c(i,j)$
                determine the shortest path from $c(i,j)$ to j
```

The complexity degree of the algorithm is $O(n^3)$.

3 The necessity for applying the algorithm

Let us consider the following problem:

It is given a n x n square. Certain positions of the square are occupied. A moving object is located in the position (i0,j0). The goal is to find the shortest path of the moving object to reach position (i1,j1), knowing that:

- it can move in 4 directions: N,S,E,W

- it can not pass through occupied positions

- it can't pass twice through the same position.

The above-defined problem is essentially simple, but representative for a certain kind of problems, in which the request is finding the shortest paths not in graphs, but in squares.

There are known, at least, two methods for solving the problem, which certainly lead to a solution: backtracking and branch & bound, but which requires a long execution time; especially for large values for n, the time to perform grows exponentially.

The method we will present below use the Roy-Floyd Algorithm and reduces this way the execution time.

4 Variant of the Roy-Floyd Algorithm for shortest path in matrices

Step 1. We consider each element of the matrix like a node in a graph. We number the matrix elements (starting from one), beginning from the (1,1) corner, on rows, from left to right.

For example, the (i,j) element of the matrix becomes the (i - 1) x n + j node in the graph; where n is the matrix size.

Step 2. We compute the cost matrix of the graph formed at step 1 like this: in this graph we form an edge between the (i – 1) x n + j and (k – 1) x n + l nodes if the positions (i,j) and (k,l) from the square are free and besides are adjacent (on one of directions N, S, E, V,). The cost of the new edge will be 1 (the number of steps made for moving between the two positions).

Step 3. We apply the Roy-Floyd Algorithm to the graph given by the cost matrix A.

Step 4. a[(i0 – 1) x n + j0 , (i1 – 1) x n + j1] will contain the minimum number of steps made by the moving object for reaching to position (i1,j1) position (i0,j0).

The path finding will be made by using the same idea as the Roy-Floyd Algorithm, but taking into account that the node x from the graph correspond to the position (i,j) from the square, where:

$$i = \begin{cases} 1 & \text{if } x < n \\ x \operatorname{div} n & \text{otherwise} \end{cases}$$

$$j = \begin{cases} n & \text{if } x \bmod n = 0 \\ x \bmod n & \text{otherwise} \end{cases}$$

Remarks

A. The complexity degree of the algorithm is O(n^6).

B. We can solve the problem without transforming the matrix into a graph, only by modifying the Roy-Floyd Algorithm.

The cost matrix becomes the table A(n,n,n,n) where:

a(i,j,k,l) =

$$\begin{cases} 1 & \text{if positions } (i,j) \text{ and } (k,l) \text{ are adjacent on the square and } (k,l) \text{ is free} \\ 0 & \text{if } i = k \text{ and } j = l \\ +\infty & \text{otherwise} \end{cases}$$

In rest, we have to adapt the Roy-Floyd Algorithm.

The inconvenient of this method is that using a four-dimensional table, the memory space required is very large.

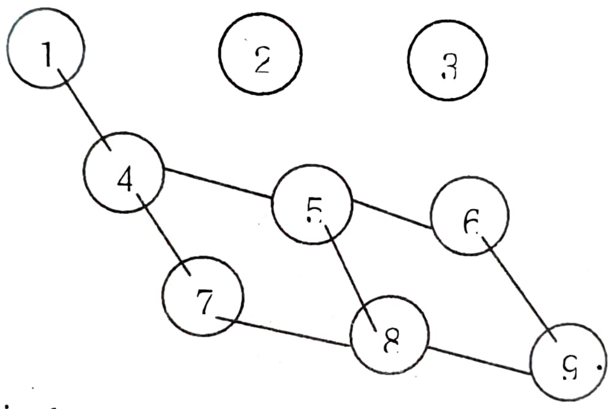
5 Example

Let us consider the square

$$\begin{array}{ccc} 0 & 1 & 1 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{array}$$

where 0 represents a free position and 1 an occupied position. The initial position of the moving object is (1,1), and the final position of the moving object (2,3). By applying our variant of the Roy-Floyd algorithm, we will obtain the minimum number of steps $\underline{3}$ and the shortest path: (1,1) – (2,1) – (2,2) – (2,3).

The graph obtained by the matrix straightness will be:



The solution is obvious: the shortest path between 1 and 6 is 1 – 4 – 5 – 6.

6 Remarks

There are two variants of Roy-Floyd Algorithm that can be used even in case of moving on squares.

A. We have to find the largest value path, not the shortest. In this case the cost matrix will be formed like:

$$a(i,j) = \begin{cases} c & \text{if}\,(i,j) \text{ is an arc and its cost is } c \\ 0 & \text{if } i = j \\ -\infty & \textit{if}\,(i,j) \text{ is not an arc} \end{cases}$$

and the step 2.1 from the algorithm becomes:

step 2.1' $a(i,j) := \max\{a(i,j), a(i,k)+a(k,j)\}$

The other steps are identical.

Application: *It is given a labyrinth with n x m rooms. In each room there is a certain sum of money and each room communicates with the adjacent room (N, S, E, W).*

A person enters the labyrinth on at the position (1,1) and has to go out at position (n,m). Knowing that passing through a room the person takes the sum of money from that room, we have to find out the largest value path (the path value is the total sum of money that the person will collect on the path).

B. It is not required to determine the shortest or largest path between two nodes, but to verify that node j can be reached from node i through a path. This variant is known as the Roy-Warshall Algorithm.

In this case, the Roy-Floyd Algorithm will be modified as follows: the cost matrix will be formed like:

$$a(i,j) = \begin{cases} 1 & \text{if } (i,j) \text{ is an arc} \\ 0 & \text{if } (i,j) \text{ is not an arc} \end{cases}$$

and the 2.1 step becomes

```
step 2.1'        if a(i,j) = 0 then
                       a(i,j) := a(i,k) x a(k,j)
```

Finally, if a(i,j) = 1, a path exists between i and j, otherwise no.

Application: *It is given a labyrinth with n x m rooms. Certain rooms of the labyrinth are occupied. In two specified positions are two lovers. It is requested to specify if the two lovers will come to find each other, knowing that they can pass only through free positions, the moving being made in 4 directions (N, S, E, W).*

7 Conclusions

Take some time to consider:

1. the time to provide the solution;
2. the memory space used by the problem data.

The classics methods (backtracking and branch & bound), increase **1**.

The above analyzed method (based on the Roy-Floyd Algorithm) decrease **1**, but increase **2**.

Although, if the program execution time decreases (the algorithm complexity is small), it requires a large memory space if working with large size matrices.

Anyway, between an algorithm that requires a large memory space and one that requires long time, is preferable the one that run faster (the problem of memory space can be solved by using the dynamic allocation of data).

As a conclusion, it results the efficiency of using the variant of Roy-Floyd Algorithm, in case of shortest paths in matrices.

REFERENCES

1. C. Croitoru, *Tehnici de baza in optimizarea combinatorie*, Editura Universitatii "Alexandru I. Cuza", Iasi, 1992

2. G.A. Dirac, *Some theorems on abstract graphs*, Proceedings of the London Mathematical Society, 2, 1952, 69-81.

3. T. Toadere, *Teoria grafelor*, Litografia Universitatii "Babes-Bolyai", Cluj, 1992

Babeş-Bolyai University, Faculty of Mathematics and Informatics, RO 3400 Cluj-Napoca, str. M. Kogalniceanu 1, România.

E-mail address: `gabis@cs.ubbcluj.ro`