ІНФОРМАЦІЙНІ ПРОГРАМИ ТА КОМП'ЮТЕРНО-ІНТЕГРОВАНІ ТЕХНОЛОГІЇ

UDC 004.738:519.85
DOI: 10.31471/1993-9965-2023-2(55)-47-53

THE SHORTEST PATH PROBLEM IN A GRAPH FOR AN EXECUTOR WITH LIMITED RESOURCES

M. S. Lvov, O. I. Lemeshchuk *

Kherson State University; 27 *Universytets'ka St., Kherson, Ukraine, 73003,*
e-mail: office@ksu.ks.ua

У статті висвітлено алгоритм Дейстктри, можливості його модифікації та узагальнене бачення можливостей модифікації. Особливу увагу було сконцентровано на аналізі пошуку найкоротших шляхів з обмеженими ресурсами. Дослідження проводиться з метою розширення бачення можливостей застосування алгоритмів Флойда і Дейкстри з додатковими параметрами. Водночас різні узагальнення проблеми найкоротшого шляху розглядаються рідко. Мета даної роботи – привернути увагу вчених і викладачів вищих навчальних закладів до одного із найбільш закономірних узагальнень проблеми. Пов'язані проблеми з використанням алгоритму з кількома джерелами досліджував Шимін Ван під час дослідження шляхів дорожніх мереж для зменшення часу, витраченого між зупинками, а також Пітер В. Еклунд працював над модифікацією алгоритму, який включає статичні та динамічні евристичні компоненти, та кілька вихідних вузлів. Модифікований алгоритм застосовано в тривимірній просторовій інформаційній системі (SIS) для маршрутизації транспортних засобів екстреної служби. Наше узагальнення полягає в тому, що ми розглядаємо виконавця алгоритму (wayfarer) з обмеженими ресурсами, які витрачаються на те, як ці ресурси можуть бути поповнені в деяких вершинах графа. Потрібно знайти найкоротший шлях, за яким виконавець може досягти цільової вершини, правильно витрачаючи і поповнюючи свої ресурси. Поряд з цими алгоритмами розглядаються також деякі їх окремі випадки. Наприклад, це алгоритм транзитивного замикання орієнтованого графа. Крім того, формулювання проблеми найкоротшого шляху представлено як задачу множення матриць на замкнуті півкільця. Стаття розрахована на розширення бачення застосування алгоритмів Флойда та Дейкстри з додатковими параметрами.

Ключові слова: задача знаходження найкоротшого шляху у графі, алгоритм Флойда, алгоритм Дейкстри, ресурс виконавця, формальні методи, алгебраїчне програмування.

The article highlights Dijkstra's algorithm, the possibility of its modification and a generalized vision of the modifications' possibility. Attention was especially paid to the analysis of the problem of finding the shortest path with limited resources. The research is performed in order to expand the vision of the application possibilities of the Floyd's and Dijkstra's algorithms with additional parameters. At the same time, various generalizations of the shortest path problem are rarely considered. The purpose of this paper is to draw the attention of scientists and university professors to one of the natural generalizations of the problem. Related problems using a multi-source algorithm were explored by Shiming Wang when investigating the paths of road networks to reduce the time spent between stops , Peter W. Eklund's work on modifying the algorithm, which includes static and dynamic heuristic components and several source nodes. The modified algorithm is applied in a 3D Spatial Information System (SIS) for routing emergency service vehicles. Our generalization is that we consider an executor of the algorithm (wayfarer) with limited resources that are being spent on the way these resources can be replenished in some vertices of the graph. It is required to find the shortest path along which the executor can reach the target vertex,

correctly spending and replenishing his resources. Along with these algorithms, some of their special cases are also considered. For example, it is directed graph transitive closure algorithm. Furthermore, the formulation of the shortest path problem is presented as a problem of matrix multiplication over closed semirings. The paper considers generalizations of the Floyd's and Dijkstra's algorithms for the case of both a single limited resource and multiple resources.

Keywords: the shortest path problem in a graph, Floyd's algorithm, Dijkstra's algorithm, executor resource, formal methods, algebraic programming.

Introduction

E. Dijkstra's algorithm [1] is difficult to be applied for $1 \times n$ kind of problem and R. Floyd's algorithm [2] (Floyd–Warshall algorithm, Roy–Floyd algorithm) for $n \times n$ kind of problem. Such an interpretation is interesting, especially because efficient ("record-breaking" [4, 5]) algorithms for matrix multiplication can be applied to the theory of shortest path problems.

Here is the formulation and procedure of **Floyd's algorithm** for positive path costs:

> Given an oriented graph $G=<V, E>$, $|V| = n$, $V=<1, 2, ..., n>$, whose edges (i, j) are marked with positive numbers $C[i,j]$, called values. The values are represented by the $n \times n$ matrix C . Let us assume $M = n * \max_{i=1,j=1}^{n,n} C[i,j] + 1$. If there is no edge (i, j) in the graph G, we denote by $C[i,j] = M$. Also put $C[i, i] = 0$. The cost of path $(i_1, i_2, ..., i_k) = (i_1, l_2), (i_2, i_3), (i_{k-1}, i_k)$ in the graph is the sum of $C[i_1,i_2] + C[i_2,i_3] + ... + C[i_{k-1},i_k]$.

Since the number of simple path links does not exceed n, the cost of any simple path, including the path of least cost (the shortest path), is less than M . As two graph edges can be connected by different paths, among all such paths there is the least cost path. In $n \times n$ problem variant it is required to find the least cost paths for all possible pairs of vertices i, j. This problem can be solved by R. Floyd's algorithm given in the Floyd procedure. The costs of such paths are the elements of result of the procedure – $n \times n$ of the matrix A. Note that all information about the graph G is present in the matrix C, and information about the shortest paths is in the matrix A . In particular, if the vertices (i, j) of the graph G are connected by some path, $A[i, j] < M$. Or, if the path from vertex i to vertex j does not exist, so $A[i, j] = M$.

```
G = Graph()
G.add_vertex('A', {'B': 7, 'C': 8}, 0)
G.add_vertex('B', {'A': 7, 'F': 2}, 1)
G.add_vertex('C', {'A': 8, 'F': 6, 'G': 4}, 0)
G.add_vertex('D', {'F': 8}, 1)
G.add_vertex('E', {'H': 1}, 0)
G.add_vertex('F', {'B': 2, 'C': 6, 'D': 8, 'G': 9, 'H': 3}, 0)
G.add_vertex('G', {'C': 4, 'F': 9}, 0)
G.add_vertex('H', {'E': 1, 'F': 3}, 1)
```

Matrix D represents all the shortest paths in the following way: if the last edge of the shortest path from i to j exists (k, j), so $D[i, j] = k$. It can be easily seen that the shortest path from i to j can be extracted from D in linear time.

In [1], the classical Floyd's algorithm reduces to matrix-multiplication-like algorithm in a closed semiring with the operation of addition $a \oplus b \stackrel{def}{=} \min(a, b)$ and with the operation of multiplication $a \oplus b \stackrel{def}{=} a + b$. In [1] it is shown that $A = C^*$, where

$$C = C \oplus C^2 \oplus ... \oplus C^n. \qquad (1)$$

This formulation represents Floyd's algorithm as a single loop:

```
n = no of vertices
A = matrix of dimension n*n
for k = 1 to n
  for i = 1 to n
    for j = 1 to n
      Ak[i, j] = min (Ak-1[i, j], Ak-1[i, k]
      + Ak-1[k, j])
return A
```

It was also shown in [1] that the closure operation defined in [1] is equivalent in time complexity to the operation of multiplying two matrices $n \times n$.

Executor resources. Here is an example of one of the obvious applications of Floyd's algorithm. Let's presume, graph G represents the road network of some region, and the cost of the edges is the length of the roads that directly connect the settlements – vertices of G, expressed as kilometers. The executor of the algorithm R is a car. Its main resource is the fuel reserve r, expressed as distance in kilometers. So, the car R can travel r kilometers on its fuel budget. At the vertices of the subset $V_R \subset V$ (i.e. certain localities) there are fuel stations for R. At these vertices R can be fueled up completely. At the beginning of the travel the car R has full tank, regardless of which vertex it is at. The problem is to find the shortest paths in a graph G, provided

ISSN 1993–9965 print
ISSN 2415–3524 online
Науковий вісник ІФНТУНГ
2023. № 2(55)

that the distance between successive vertices of a subset V_r on this path does not exceed r.

1. The algorithm for finding the shortest paths with a limited resource.

If the classical formulation of *the shortest paths problem* variant $n \times n$ can be expressed with the tuple $n,\ V,E,C >$ and the algorithm with the Floyd function:

$$(A,D) = Floyd(n,C)$$

then *the shortest paths problem* with a limited resource of the executor with the tuple $< n,\ V,\ E,\ C,k,r,V_R \subset V >$ and the algorithm with the FloydRes function:

$$(A_R,D_R) = FloydRes(n,C,k,r,V_R).$$

The FloydRes algorithm below uses, at first, the Floyd procedure, and at second, the following operation on the matrices of costs:

$$A := \text{Reduce}(n,C,r),$$

which performs the transformation for all $(i,\ j)$

if C[i,j]>r:
C[i,j] = M

This means that if the cost of an edge $(i,\ j)$ exceeds the resource r, that edge is not included in any path with a constraint on that resource and, therefore, it can be excluded from the list of edges.

Here is an informal description of the FloydRes algorithm. The vertices from V_R set will be called as active ones and the vertices from $V - V_R$ set – passive ones.

Let's presume that $|V_R| = k$. Without the Limits of Generality, we will assume that the active vertices are renumbered from 1 to k: $Act = \{1,\ 2,\ ...,k\}$. Meanwhile, the passive ones are renumbered from $k+1$ to n. $Pas = \{k+1,\ ...,n\}$.

The FloydRes algorithm performs a transformation of the matrix of costs C at each step. The matrices of costs at each stage are represented as the following, where C_{act} - $k \times k$ is the submatrix of edge costs between pairs of active vertices, C_{pas} - $n-k \times n-k$ is the submatrix of edge costs between pairs of passive vertices, and C_{ap} - are the submatrices of edge costs between active and passive vertices. Thereby,

$$C = \begin{bmatrix} C_{act} & C_{ap} \\ C_{pa} & C_{pas} \end{bmatrix}.$$

1. Performing a conversion beforehand

$$A^{(1)} := \text{Reduce}(n,C,r).$$

Denote the transformed graph as $G^{(1)}$.

2. Performing the transformation

$$(A^{(2)},D) = Floyd(n,A^{(1)}).$$

The graph $\bar{G} = V,\{\bar{E}\}$, constructed from a matrix $A^{(2)}$, is essentially a transitive closure G: $G^{(2)} = G$ Performing the transformation

$$A^{(3)} := \text{Reduce}(n,A^{(2)},r).$$

Denote the transformed graph as $G^{(3)}$.

3. All shortest paths in the graph $G^{(3)}$ can be classified as following:

Pathways from passive vertex i to passive vertex j with costs $A^{(3)}_{pas}(i,\ j)$.

- If $A^{(3)}_{pas}[i,j] = M$, then the path from vertex i to vertex j does not exist.
- If $A^{(3)}_{pas}[i,j] \le r$, then the solution for this pair of vertices is found.
- The case of $A^{(3)}_{pas}[i,j] > r$ should have further calculations. Let's take a closer look at the shortest path of $A[i,j] > r$, cost, if it exits.

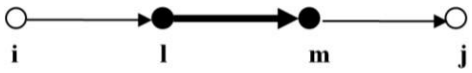

Figure 1 – The shortest path to *Pas*

The double arrow shows the path along the vertices of the subgraph $G^{(2)}_{act}$, the single arrows show the $G^{(2)}_{pa}$ and $G^{(2)}_{ap}$ edges. So, if the shortest path with a limited resource exists, then $A^{(3)}_{pas}[i,l] \le r,\ A^{(3)}_{pas}[m,j] \le r$.

Pathways from passive vertex i to active vertex m. So:

- If $A[i,m] = M$, then the pass from vertex i to vertex m does not exist;
- If $A[i,m] > r$, then the shortest path from i to m does not satisfy the constraint, and then $A[i,m] := M$;
- If $A[i,j] \le r$, then the shortest path from i to m is found and it has the following form:

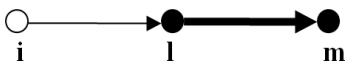

Figure 2 – The shortest path from *Pas* to *Act*

Pathways from active vertex l to passive vertex j. So:

- If $A[l,j] = M$, then the pass from vertex l to vertex j does not exist;
- If $A[l,j] > r$, then the shortest path from l to j does not satisfy the constraint, and then $A[l,j] := M$;
- If $A[l,j] \le r$, then the shortest path from l to j is found and it has the following form:

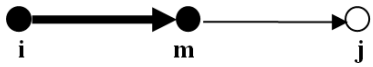

Figure 3 – The shortest path from *Act* to *Pas*

Pathways from active vertex l to active vertex m, consisting of several links.

- If $A[l,m] = M$, then the path from vertex l to vertex m does not exist;
- If $A[l,m] \le r$, then the shortest path from l to j is found and it has the following form:

Figure 4 – The shortest path to *Act*

- If $A[l,j] > r$, then the shortest path from l to j does not satisfy the constraint, and then

$$A[l,j] := M;$$

4. Let's presume that submatrix $A_{act}^{(3)}$ of the $k \times k$ size is formed with the first k rows and columns of matrix of costs $A^{(3)}$. Then, let's perform the transformation:

$$(A_{act}^{(4)}, D_{act}^{(4)}) = \text{Reduce}(k,\ Floyd(n, A_{act}^{(3)},\ r)$$

and take a look at matrix of costs

$$A^{(4)} = \left[A_{act}^{(4)}\ A_{ap}^{(3)}\right]$$

In graph $G^{(4)}$, based on matrix $A^{(4)}$, all shortest paths are represented by edges. Fig.1, for example, in this graph, the path has the following form

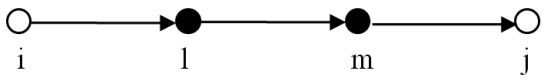

Figure 5 – New shortest path to *Act*

Similar changes are applied to Fig.2, Fig.3, Fig.4.

So,

$$A_{pas}^{(5)} = A_{pa}^{(3)} \otimes A_{act}^{(4)} \otimes A_{ap}^{(3)} \oplus A_{pas}^{(3)}$$

$$A_{pa}^{(5)} = A_{pa}^{(3)} \otimes A_{act}^{(4)} \oplus A_{pa}^{(3)}$$

$$A_{ap}^{(5)} = A_{act}^{(4)} \otimes A_{ap}^{(3)} \oplus A_{ap}^{(3)}$$

$$A^{(5)} = \begin{bmatrix} A_{act}^{(5)} & A_{ap}^{(5)} \\ A_{pa}^{(5)} & A_{pas}^{(4)} \end{bmatrix}.$$

5. Matrices of the shortest paths are computed with an extended algorithm that generates the shortest path of the executor with constraints for any pair of vertices $(i,j) \in E$ in linear time.

Analysis of the algorithm shows the complexity of the algorithm for finding the shortest path with one constraint $T(n) = O(n^3)$.

Since the most complex part of our algorithm is the classic Floyd's algorithm, its complexity can theoretically be estimated through the fastest matrix multiplication algorithm $T(n) = O(n^d)$, where $d < 3$. Moreover, the number d of efforts of many authors, starting with Strassen [2 Gaussian elimination is not optimal], the algorithm of matrix multiplication is constantly decreasing [3], [4].

In practice, however, the classical time complexity $O[n^3]$ algorithm is used.

$$A^{(1)} := \text{Reduce}(n, C, r);$$

$$(A^{(2)}, D) = Floyd(n, A^{(1)});$$

$$A^{(3)} := \text{Reduce}(n, A^{(2)}, r);$$

$$(A_{act}^{(4)}, D_{act}^{(4)}) = \text{Reduce}(k,\ Floyd(n, A_{act}^{(3)},\ r);$$

$$A^{(5)} = A^{(4)} \otimes A^{(3)} \oplus A^{(3)};$$

$$A_{pa}^{(5)} = A_{pa}^{(3)} \otimes A_{act}^{(4)} \oplus A_{pa}^{(3)}$$

$$A_{pas}^{(5)} = A_{pa}^{(3)} \otimes A_{act}^{(4)} \otimes A_{ap}^{(3)} \oplus A_{pas}^{(3)}$$

$$A^{(R)} = A^{(5)} = \begin{bmatrix} A_{act}^{(5)} & A_{ap}^{(5)} \\ A_{pa}^{(5)} & A_{pas}^{(4)} \end{bmatrix}.$$

The algorithm for calculating the matrix of costs $A^{(R)}$ must be supplemented by the algorithm for calculating matrix of the shortest paths $D^{(R)}$.

2. Problem of finding the shortest paths with two resources

Here is a description of the problem of finding the shortest paths in a directed graph when the executor has two finite resources. This can be, for example, the maximum distance travelled on a single tank of fuel and the maximum distance the driver can drive without taking a rest.

So, the following are given:

- Directed graph $G=<V, E>$, $|V| = n$, $V=<1, 2, \ldots, n>$, which edges (i, j) are marked with positive numbers $C[i,j]$, called weights or costs. The costs are presented with matrix C of $n \times n$ size. Suppose that $M = n \cdot \max\limits_{i,j \in \{1..n\}} (C[i,j]) + 1$. As in the previous formulation, if there is no edge (i, j) in graph G, then $C[i,j] \stackrel{df}{=} M$.
- The executor resource of type R_1 is specified by the maximum reserve value r_1 and the subset of vertices $V_1 \subset V$, $|V_1| = k_1$, where this resource can be replenished. The resource of type R_2 is specified by the maximum reserve value r_2

ISSN 1993–9965 print
ISSN 2415–3524 online
Науковий вісник ІФНТУНГ
2023. № 2(55)

and the subset of vertices $V_2 \subset V$, $|V_2| = k_2$ – generator of resource R_2.

The task is to find the shortest paths between all pairs of vertices in a graph along which the following constraints are satisfied: if $V_1, V_2 \in V_i$ $i = 1,2$ - are two vertices between which there are no other vertices from V_i, then the cost of the path is $(v_1, v_2) \leq r_i$.

The algorithm for finding the shortest paths with two resources

The algorithm for solving this problem is a direct generalization of the algorithm presented by the Floyd's function. FloydRes2 algorithm uses natural number n, matrices C, A, D of $n \times n$ size, matrix E of $2 \times n \times n$ size, sets R_1, $R_2 \subset V$, positive real numbers r_1, r_2. R_1, $R_2 \subset V$ - sets of active vertices of graph G. r_1, r_2 - are the constraints on resources, and matrix E stores the amounts of unused corresponding resources on a given path. The matrix E initialization is performed before the main loop

```
i = 1
for i in range(n):
    for i in range(n):
      E[1,i,j] = r1
      E[2,i,j] = r2
```

in the body of the main triple procedure loop instead of the conditional operator

```
if (A[i,k]+C[k,j])<A[i,j]:
 A[i,j] = A[i,k] + C[k,j]
 D[i,j] = k
```

here the following operator was used

```
If isCorrect(i,j,k,r1,r2,R1, R2)
  nextStep(r1,r2,R1,R2)
```

the semantics of the condition **isCorrect**(r1, r2, R1, R2) and the calculation of the next step **nextStep**(r1, r2, R1, R2) is shown on Fig. 6. It shows the *travelled path* (i, k) and *edge* (k, j) *attached to this path*

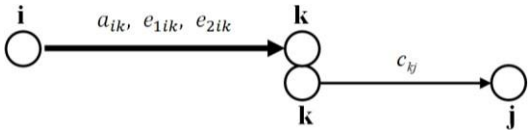

Figure 6 – Selecting an acceptable path continuation

The logic function **isCorrect** for selecting an acceptable path continuation considering subject to constraints is carried out by analyzing *the cost of the travelled path* $A[i, k]$ and *the cost of the attached edge* $C[k, j]$.

```
def isCorrect(C, A, i,j,k, r1, r2, E):
  if (E[1,i,k] >= C[k,j])&
   (E[2,i,k] >= C[k,j]) &
   (A[i,k]+C[k,j] < A[i, j]):
     return true
```

The matrix E of $2 \times n \times n$ size contains data on the remains of the corresponding resources after the travelled path (i, j).

The nextStep procedure changes the remaining resources at a vertex j taking into account passive and active vertices:

```
def nextStep()
    E[1,i,j] = E[1,i,k]–C[k,j];
    E[2,i,j] = E[2,i,k]–C[k,j];
if j in R1:
   E[1,i,j] = r1;
if j in R2:
   E[2,i,j] = r2;
A[i,j] = A[i,k]+C[k,j];
D[i,j] = k
```

As **isCorrect** and **nextStep** procedures are executed in $O(1)$, time, the time complexity of the FloydRes2 algorithm is estimated as $O(n^3)$.

Of course, the **FloydRes2** algorithm can be used in a task with a single resource. The benefits of the **FloydRes** algorithm are listed below:

0. **FloydRes** algorithm does not use any additional memory, while FloydRes2 algorithm uses matrix E of $n \times n$ size for this operation.

1. **FloydRes2** algorithm does not use calculations in a bounded semicircle so its complexity cannot be estimated $O(n^d)$, $d < 3$.

A clear advantage of the FloydRes2 algorithm is the possibility of using it in algorithms with other conditions **isCorrect()** and calculations **nextStep.**

Some algorithms for finding the shortest paths of FloydRes2 type.

1. Dijkstra's algorithm

The Dijkstra's algorithm [2, 3] solves the shortest path problem in the classical $1 \times n$ formulation, i.e. from a given vertex to all other vertices. Here is his text from [2] in the pseudolanguage and in our version

```
def Dijkstra(...):
    U: {2..n} # U – is the set of unprocessed vertices
    S: {1}  # S – set of processed vertices, U + S = V
    M = max(r1, r2)
    Inf = M + 1  # Inf – notation of unreachability
```

```
C: [1..n, 1..n] # Matrix of costs of edges No edge – C[I,j] = Inf
E1: [1..n] # vector of resource residuals R1
E2: [1..n] # vector of resource residuals R2
D: [1..n] # vector of path values reached

for i in range(n):
  D[i] = C[1,i]  # initialising path value arrays

for i in range(n-1):
  w = GetMin(U,D) finding the vertex with the min.distance to U:
    # moving w from U to S
    U = U – {w}
    S = S + {w}

# replenish D by the values of vertices, which are preceded by w
    for j in range(U):
      D[j] = min(D[j], D[w] + C[w,j])
```

An extension of the Dijkstra algorithm for the $1 \times m$ problem type will be called as **DijkstraRes2.**

Data structure specification:

U,S: array[1..n] of Boolean; //sets of vertices to be processed and unprocessed

D: array[1..n] of Real; //array of values of tops reached

E: array[1..2, 1..n] //array of residuals in attained nodes

The **DijkstraRes2** algorithm differs from the classical algorithm with an initialization loop of matrix E of $2 \times n$ size, pre-processing of the matrix of costs and subsequent implementation of the **GetMin(U,D)** function:

```
// Initialise resource residual matrix
 i = 1
  for i in range(n):
    E[1,i]=r1
    E[2,i]=r2
// Pre-processing the value matrix
  M=Max(r1,r2)
  Inf=M+1
  i = 1
  for i in range(n):
    j = 1
    for j in range(n):
      if C[i,j]>M:
        C[I,j]=Inf
```

```
// Search for a min. unprocessed edge which satisfies the constraints
  GetMinRes2 = GetMin

//Function for replenishing D with the values of the vertices of the indent w
  for j in range(U):

    if D[w]+C[w,j] < D[j]:
     if (E1[w] >= C[i,j])& E2[w] >= C[i,j]):
        E1[w] = E1[w] - C[i,j])
        If w in R1:
              E1[w] = r1
        E2[w] = E2[w] - C[i,j])
        If w in R2:
              E2[w] = r2
```

2. FloydResN algorithm with N resources

To solve a problem with N resources, we modify our previous algorithm by adding a set of n.

in the body of the main triple procedure loop instead of the conditional operator

```
if (A[i,k,n]+C[k,j,n])<A[i,j,n]:
 A[i,j,n] = A[i,k,n] + C[k,j,n]
 D[i,j,n] = k
```

here the following operator was used

```
If isCorrect(i,j,k,r1,r2,R1,R2,n)
  nextStep(r1,r2,R1,R2,n)
```

The **nextStep** procedure changes the remaining resources at a vertex j taking into account passive and active vertices:

```
def nextStep()
    E[1,i,j,n] = E[1,i,k,n]–C[k,j,n];
    E[2,i,j,n] = E[2,i,k,n]–C[k,j,n];
if j in R1:
   E[1,i,j,n] = r1;
if j in R2:
   E[2,i,j,n] = r2;
A[i,j,n] = A[i,k,n]+C[k,j,n];
D[i,j,n] = k
```

The benefits of the FloydRes algorithm are listed below:

A clear advantage of the FloydRes2 algorithm is the possibility of using it in algorithms with unlimited number of conditions

Conclusion

The problem considered in this paper, according to the scheme of its formulation, belongs to the class of *optimization problems*. Problems of this class such as the linear programming problem,

ISSN 1993–9965 print
ISSN 2415–3524 online
Науковий вісник ІФНТУНГ
2023. № 2(55)

or, in a more general formulation, the mathematical programming problem, are well known. Indeed, in this problem, a system of constraints is explicitly defined and a criterion for the optimality of the solution is formulated. This raises the question of whether this problem can be reduced to one of *the well-known optimization problems.*

Similar generalizations can be considered for some other problems on graphs. For example, traveling salesman problem. At the same time, the solution scheme given in this article is also applicable to the traveling salesman problem. Apparently, the classical Floyd's and Dijkstra's algorithms act as basic algorithms in various formulations of the shortest path problems.

From our point of view, the interesting question is whether it is possible to construct a solution algorithm using the **Reduse()** and **Floyd()** operations, as it is done in the algorithm for solving the problem with one constraint.

References

1. Dijkstra E. W. A note on two problems in connexion with graphs. *Numer. Math* / F. Brezzi — Springer Science + Business Media, 1959. Vol. 1, Iss. 1. P. 269–271. ISSN 0029-599X; 0945-3245. doi:10.1007/BF01386390

2. Robert W. Floyd. 1962. Algorithm 97: Shortest path. Commun. ACM 5, 6 (June 1962), 345. DOI: https://doi.org/10.1145/367766.368168

3. Anderson J. Discrete mathematics and combinatorics. Moscow: Williams Publishing House, 2003. URL: https://archive.org/details/discretemathemat0000ande

4. Evstigneev V.A. Chapter 3: Iterative algorithms for global graph analysis. Paths and coverages. *Application of graph theory in programming* / Edited by A. P. Ershov. Moscow: Nauka. Main editorial office of physical and mathematical literature, 1985. P. 138–150.

5. Alexeev V.E., Talanov V.A. Chapter 3.4. Finding of shortest paths in a graph. *Graphs. Calculation models. Data structures*. Nizhny Novgorod: Nizhny Novgorod State University Publisher, 2005. P. 236-237. ISBN 5-85747-810-8. Archived on December 13, 2013 at the Wayback Machine

6. Cherkassky B. V., Goldberg A. V., Radzik T. Shortest paths algorithms: Theory and experimental evaluation (англ.). *Math. Prog.* Springer Science + Business Media, 1996. Vol. 73, Iss. 2. P. 129–174. ISSN 0025-5610; 1436-4646. doi:10.1007/BF02592101

7. Wang S. et al. Double-Sources Dijkstra Algorithm within Typical Urban Road Networks / Xie A., Huang X. (eds) Advances in Computer Science and Education. *Advances in Intelligent and Soft Computing*. 2012. Vol. 140. Springer, Berlin, Heidelberg. https://doi.org/10.1007/978-3-642-27945-4_24

7. Eklund P. W., Kirkby S., Pollitt S. A dynamic multi-source Dijkstra's algorithm for vehicle routing. *1996 Australian New Zealand Conference on Intelligent Information Systems. Proceedings. ANZIIS 96*, 1996, pp. 329–333, doi: 10.1109/ANZIIS.1996.573976.