O. C. BÂSCA
Bucharest University

On Algorithms of Minimum Paths in Graphs

The aim of this article is to present several variants of the classical modified Warshall and Dantzig algorithms of finding the minimum paths in a diagraph and indications about the programming of these algorithms on a computation system. Part of these algorithms were elaborated within a scientific research theme on a contract basis for "Optimizing Public Transport Lines in Bucharest Municipality"

The modified Warshall algorithm permits to find the length of the minimum paths between any two nodes and the nodes crossed by one of the shortest paths for each pair of nodes (if it exists).

One of the classical variants of the modified Warshall algorithm is the following [3]:

Algorithm 1. Let X be the matrix of the weights of a digraph with n nodes, a square matrix of order n and having as elements x_{ij} the weight of arc from node i to node j if there is an arc from node i to node j and ∞ in case there is no arc from node i to node j. The algorithm builds two square matrices X^* and M of order n which have as elements x^*_{ij} — the length of the shortest path from i to j (if there is no path from i to j $x^*_{ij} = \infty$) and respectively m_{ij} — the first intermediate node on one of the minimum path from i to j ($m_{ij} = 0$ if there is no path from i to j).

1. Initialize matrix M by putting $m_{ij} = j$ if $x_{ij} \neq \infty$ and $m_{ij} = 0$ if $x_{ij} = \infty$ for $i, j = 1, 2, ..., n$; set $x^*_{ij} = x_{ij}$ for $i, j = 1, 2, ..., n$;
2. Set $j = 1$;
3. Set $i = 1$;
4. If $x^*_{ij} = \infty$ go to step 8;
5. Set $k = 1$;
6. If $x^*_{ij} + x^*_{ij} < x^*_{ij}$ set $x^*_{ij} = x^*_{ij} + x^*_{ij}$ and $m_{ik} = m_{ij}$;
7. Set $k = k + 1$. If $k \leqslant n$, go to step 6;
8. Set $i = i + 1$. If $i \leqslant n$, go to step 4;

9. Set $j = j + 1$. If $j \leqslant n$, go to step 3, else STOP.

The effective finding of one the minimum paths from node i to node j is made by applying the following algorithm:

Algorithm 2. Let M be the square matrix of order n built in algorithm 1.

1. If $m_{ij} = 0$, there is no path from i to j and STOP;
2. Set $k = i$;
3. Pass k into the output vector;
4. If $k = j$, then STOP;
5. Set $k = m_{kj}$ and go to step 3.

Upon the completion of the algorithm through step 4, the indices of the nodes appearing in one of the minimum paths from i to j in the order from i to j are found in the output vector.

Another variant of the algorithm 1 is the following:

Algorithm 3. In algorithm 1 steps 1 and 6 respectively are replaced by:

1. Initialize matrix M by putting $m_{ij} = i$ if $x_{ij} \neq \infty$ and $m_{ij} = 0$ if $x_{ij} = \infty$ for $i, j = 1, 2, ..., n$, set $x^*_{ij} = x_{ij}$ for $i, j = 1, 2, ..., n$;

.....

6. If $x^*_{ij} + x^*_{ij} < x^*_{ij}$, set $x^*_{ij} = x^*_{ij} + x^*_{ij}$ and $m_{ik} = m_{jk}$;

.....

(the rest of steps remain unchanged).

Upon finishing the algorithm, matrix M will contain as elements m_{ij} the index of the last intermediate node on one of the minimum paths from i to j ($m_{ij} = 0$ if there is no path from i to j).

The effective finding of one of the minimum paths from node i to node j is made by applying the following algorithm:

Algorithm 4. Let M be the square matrix of order n built in algorithm 3.

1. If $m_{ij} = 0$, there is no path from i to j and STOP;
2. Set $k = j$;
3. Pass k into the output vector;
4. If $k = i$ then STOP;
5. Set $k = m_{ij}$ and go to step 3.

Upon finishing the algorithm through step 4, the indices of the nodje appearing in one of the minimum paths from node i to j in the order from j to i are found in the output vector.

By applying algorithms 1 and 3 to a particular case, the number of operations made (comparisons, additions, attributions, etc.) is the same. Similarly, if algorithms 2 and 4 are applied to the matrices resulting from algorithms 1 and 3 respectively for a particular case, the number of operations made is the same. Differences appear in case these algorithms are programmed on a computation system and the matrices one works with do no go in the internal storage (the number of nodes of the digraph is too big). In this case, a file of direct access to the disc is defined for each matrix. The recordings of the files correspond respectively to the elements of a line in the given matrices. In this way, in algorithms 1 and 3 for a given i and j for making steps 5, 6 and 7

we use only lines i and j of the matrix X^* and line i of matrix M in algorithm 1, respectively lines i and j of matrix M in algorithm 3.

To make an ever smaller number of transfers of recordings from the disc into the internal storage and vice versa, the transfer of data is made as follows:

a) for algorithm 1:

— at step 1 read a line of matrix X, calculate the corresponding line of matrix M and transfer on the disc the line of M and the line of X^* (identical with that of X); totally, there are n transfers from the disc into the internal storage and $2n$ transfers in an opposite direction;

— at step 3, before making $i = 1$ read line j of matrix X^* in vector $L1$; in total, there are n readings;

— at step 4 read first line i of matrix X^* in vector $L2$ and if $L2(j) = \infty$ go to step 8, otherwise read in vector $L3$ line i of matrix M; the total number of readings in this step is at the most $2n^2$;

— step 6 is written now in the following form:

If $L2(j) + L1(k) < L2(k)$, set $L2(k) = L2(j) + L1(k)$ and $L3(k) = = L3(j)$

— at step 7, before going to step 8, write in the files corresponding to matrices X^* and M in line i vectors $L2$ and respectively $L3$; there are at the most $2n^2$ writing operations.

In total, make at the most $2n^2 + 2n$ readings and as many writings on the disc.

b) for algorithm 3:

— at step 1, proceed similarly as for algorithm 1;

— at step 3, proceed similarly as for algorithm 1, but, also read in vector $L4$ line j of matrix M; n additional readings;

— at step 4, proceed similarly as for algorithm 1;

— step 6 is re-written in the form:

If $L2(j) + L1(k) < L2(k)$ set $L2(k) = L2(j) + L1(k)$ and $L3(k) = = L4(k)$;

— at step 7 proceed similarly as for algorithm 1.

In total, do at the most $2n^2 + 3n$ readings from the disc and $2n^2 + 2n$ writings on the disc.

With the matrices obtained through algorithms 1 and 3, let us see how a minimum path can be effectively obtained between the given nodes i and j by using algorithms 2 and 4 respectively.

At algorithm 2, read first line i of matrix M in vector L and if $L(j) = 0$ the algorithm is finished. Otherwise the algorithm is continued and read line k of matrix M in vector L at step 5, after $k = L(j)$ is put (the correspondent of $k = m_{kj}$). If the path contains p nodes, p readings are totally made on the disc.

At algorithm 4, at the beginning read in vector L line i of matrix M and instead of $k = m_{ik}$ at step 5 put $k = L(k)$. In this case, a single reading from the disc is made regardless of the number of nodes contained by the path.

Moreover, if the minimum paths from a node i to several nodes are necessary, they can be found only by using line i of matrix M (thus after one reading) in algorithm 4, whereas in algorithm 2, the number of readings depends

on the total number of nodes of the paths sought and on the number of paths.

Thus, by using algorithms 3 and 4 instead of 1 and 2 in case minimum paths are sought in a digraph with a big number of nodes, less transfer operations are made between the internal and the external storage and thus, the calculation time is smaller.

For a digraph with approximately 100 nodes, the program corresponding to algorithm 3 lasted for about 40 minutes and the program corresponding to algorithm 4 which finds the minimum paths for all pairs of nodes lasted for about 3 minutes (without their printing).

By applying the previous algorithms, a single minimum path is preserved for each pair of nodes. But, for some pairs of nodes it may happen that there should be several paths of the same length for which the minimum is achieved. In order to find all minimum paths, algorithms similar to those presented before may be used.

A first algorithm is similar to algorithms 1 and 3. In this algorithm, matrix M has as elements m_{ij} vectors containing node indices. In the programming of this algorithm, M will be a tridimensional matrix, the first two indices having the same significance as before and the third one representing the index of an element of the corresponding vector. To know the number of elements in each vector at a given moment, a quadratic matrix M' of order n is used. The algorithm is the following:

Algorithm 5. Let X and X^* be the matrix of weights of a digraph with n nodes and respectively the matrix of the minimum paths defined as in algorithm 1 and M a matrix of vectors defined as above.

1. Initialize matrices M and X^*. Put in vector m_{ij} index i if $x_{ij} \neq \infty$, and put nothing if $x_{ij} = \infty$ and set $x^*_{ij} = x_{ij}$ for $i, j = 1, 2, .., n$;
2. Set $j = 1$;
3. Set $i = 1$;
4. If $x^*_{ij} = \infty$ go to step 10;
5. Set $k = 1$;
6. If $x^*_{ij} + x^*_{jk} > x^*_{ik}$ go to step 9;
7. If $x^*_{ij} + x^*_{jk} = x^*_{ik}$ put in vector m_{ik} the reunion of the elements of vectors m_{ik} and m_{jk} and go to step 9;
8. Set $x^*_{ik} = x^*_{ij} + x^*_{jk}$ and replace vector m_{ik} by m_{jk};
9. Set $k = k + 1$. If $k \leqslant n$, go to step 6;
10. Set $i = i + 1$. If $i \leqslant n$, go to step 4;
11. Set $j = j + 1$. If $j \leqslant n$, go to step 3; else STOP.

At the end of the algorithm, just as for algorithms 1 and 3, X^* will have as elements x^*_{ij} the length of the minimum paths if there is a path from i to j or ∞ otherwise. For finding the minimum paths from node i to node j, the following algorithm is applied:

Algorithm 6. Let M be the matrix established in the previous algorithm. Similarly, a steck L is used.

1. If vector m_{ij} contains no node, there is no path from node i to node j and STOP;
2. Set $k = j$;
3. Set $p = 1$;
4. Pair (k, p) is passed into steck L;

5. If $k = i$ go to step 7;

6. Set k equal to the p-th element of vector m_{ik} and go to step 3;

7. Print the elements on the first place of pairs in steck L in the order from top to bottom of the steck (without taking the elements out of the steck), obtaining one of the minimum paths from i to j;

8. Take out pair (i, p) of the steck;

9. If the steck is empty, STOP;

10. Take out pair (k, p) of steck L;

11. If p is equal to the number of elements in vector m_{ik} go to step 9, otherwise, set $p = p + 1$, the pair (k, p) is passed into steck L and go to step 6.

In the programming of these algorithms on a computation system, for a third dimension of matrix M a number p is considered, which represents the maximum number of elements which may appear at a given moment in a vector. If no other better approximation for p can be found the maximum number of arces contained in a node of the digraph can be considered as a third dimension and if this number cannot be approximated either from the beginning, consider number $n - 1$ as the third dimension of matrix M. Besides matrix M, matrices X, X^*, the square matrix M' of the order n are also used and, in algorithm 6 a steck L which is a matrix with two lines and the most n columns, each column representing a pair defined as above.

In the case of a big number of nodes, the defined matrices do no go into the internal storage and therefore three files in direct access on the disc are organized, the first two corresponding to matrices X and X^*, as described above and in the third file, each recording contains vectors in a line of matrix M and the corresponding line of matrix M'.

The operations made in this case are the following:

a) for algorithm 5:

— at step 1 for each node i read line i of matrix X, write in recording of the file corresponding to X^* and for each j put $m_{j1} = i$ and $m'_j = 1$ if $x_{ij} \neq \infty$ and $m'_j = 0$ otherwise. Write in the recording i of the third file the rectangular matrix M (with n lines and p columns, where p is defined as above) and vector M' with n elements;

— at step 3, at the beginning, read the recording j of file X^* in vector $L1$ and recording j of the third file in matrices M and $M1$;

— at step 4 read first the recording i of the file X^* in vector $L2$ and, if $L2(j) \neq \infty$, read the recording i of the third file in matrices $M2$ and $M3$, else go to step 10;

— at step 6, if $L2(j) + L1(k) > L2(k)$ go to step 9;

— at step 7, if $L2(j) + L1(k) = L2(k)$ add to line k of matrix $M2$ the elements of line k of matrix M which are not already in it and change correspondingly the value of $M3(k)$. Go to step 9;

— at step 8, set $L2(k) = L2(j) + L1(k)$ and replace line k of matrix $M2$ by line k of matrix M and element $M3(k)$ by $M1(k)$.

b) for algorithm 6:

— at step 1, read the recording i of the third file in matrices M and $M1$ and if $M1(j) = 0$ the algorithm is ended;

— at step 6, set k equal to $M(k, p)$ and go to step 3;

— at step 11, if p is equal to $M1(k)$ go to step 9, else set $p = p + 1$, pass pair (k, p) into steck L and go to step 6.

If the minimum paths from node i to several nodes are necessary, the application of algorithm 6 may be repeated without reading matrices M and $M1$. In order to find all the minimum paths of the digraph n readings of matrices are necessary in all.

We shall further present several variants of the Dantzig algorithm for finding the minimum paths from a node of a digraph to the other nodes of the digraph and variants of the modified Dantzig algorithm for finding the minimum path between two given nodes of a digraph. The variants proposed may by easily programmed on a computation system and enable the effective finding of the minimum paths.

In the classical variant, the Dantzig algorithm is the following [3]:

Algorithm 7. Considering a node a_1 in a digraph $< A, R>$, let A_p be the set of all nodes attained from a_1 (for which there is at least a path from a_1 of that node). For any $a_j \in A_p$ the algorithm finds the minimum path $(a_1, ..., a_j)$ through iterations. Let $c(a_j)$ be the length of this minimum path. We shall denote by A_i the set of all nodes a_k for which $c(a_k)$ is known at the beginning of iteration i. Start with $A_1 = \{a_1\}$ and find gradually $A_2, ..., A_p$. We shall have

$$A_1 \subset A_2 \subset ... \subset A_p.$$

1. Set $A_1 = \{a_1\}$, $c(a_1) = 0$, $i = 1$. Build a digraph T having a single node a_1;
2. For any arc $< a_k, a_t >$ so that $a_k \in A_i$ and $a_t \notin A_i$ calculate $d_{kt} = c(a_k) + w_{kt}$, where w_{kt} is the (positive) weight of arc $< a_k, a_t >$;
3. set $A_{i+1} = A_i$ and establish $d = min\, d_{kt}$. For any arc $< a_k, a_t >$ for which $d_{kt} = d$ (there can be several such arces):
   a. Set $c(a_t) = d$;
   b. Set $A_{i+1} = A_{i+1} \cup \{a_t\}$;
   c. In T add node a (if there is not already) and arc $< a_k, a_t >$;
4. If from A_{i+1} no arc comes out to the remaining nodes STOP; otherwise set $i = i + 1$ and go to step 2.

The digraph T obtained in the previous algorithm is acyclic. If all minimum paths are unique, it is an oriented tree. Any path in T corresponds to a minimum path in digraph $< A, R >$.

In case that only the minimum path between two given points of the digraph is sought, the following algorithm can be applied, known as the modified Dantzig algorithm [3]:

Algorithm 8. Considering the nodes a_1 and a_n, the following algorithm finds the minimum path $(a_1, ..., a_n)$:

The Dantzig algorithm is applied in the direction from a_1 to the other nodes and simultaneously a similar method is applied in an opposite direction from a_n by calculating the lenghts of the minimum paths $c'(a_j)$ from node a_j to node a_n. When $c(a_m)$ and $c'(a_m)$ were found for the same node a_m, we obtained a set A and one A'. The minimum path is the path $(a_1, ..., a_i, a_j, ..., a_n)$, where $a_i \in A$ and $a_j \in A'$ (eventually $a_i = a_j = a_m$) for which $c(a_i) + c'(a_j) + w_{ij}$ is minimum.

In order to find the nodes crossed by the minimum paths established in algorithm 7, a variant of this algorithm is used:

Algorithm 9. Instead of digraph T in algorithm 7 a vector M with n components is used (where n is the number of nodes of the digraph). At step 1 initialize this vector with 0 for all components and step 3 is replaced by:

3'. Set $A_{i+1} = A_i$ and establish $d = min\, d_{kt}$. For any arc $< a_k, a_t >$ for which $d_{kt} = d$ and $a_t \notin A_{i+1}$:

a. Set $c(a_t) = d$;

b. Set $A_{i+1} = A_{i+1} \cup \{a_t\}$

c. Set $m_t = k$

the rest of the steps of the algorithm 7 remain unchanged.

At the end of the algorithm, for finding the nodes through which the minimum path passes from a_1 to a_j with $j \neq 1$, the following algorithm is applied:

Algorithm 10. Let L be a steck and M the vector established for the previous algorithm.

1. If $m_j = 0$ there is no path from a_1 to a_j and STOP;
2. Set $k = j$;
3. Put a_k in steck L;
4. If $k = 1$, STOP. (The minimum path from a_1 to a_j contains the nodes found in steck L in the order from the top of the steck to its bottom);
5. Set $k = m_k$ and go to step 3.

Algorithms 9 and 10 give one of the minimum paths (if there is at least one) from node a_1 to any other node a_j of the digraph. Some changes can be made so as to obtain all minimum paths from a_1 to the other nodes of the digraph. Thus, corresponding to algorithm 9 is:

Algorithm 11. A vector M is used which has as elements m_i a vector which will contain node symbols. The algorithm operates in the same way as algorithm 7 except for the fact that digraph T is no longer used, at step 1 initialize M by indicating for all its elements that fact the they contain no element, and step 3 is replaced by:

3''. Set $A_{i+1} = A_i$ and establish $d = min\, d_{kt}$. For any arc $< a_k, a_t >$ for which $d_{kt} = d$:

a. Set $c(a_t) = d$;

b. Set $A_{i+1} = A_{i+1} \cup \{a_t\}$;

c. Add to vector m_t element k;

The finding of all minimum paths from node a_1 to any other node a_j of the digraph is made by applying:

Algorithm 12. Consider M the vector (with vector elements) established in the previous algorithm and let L be a steck.

1. If vector m_j contains no element, there is no path from node a_1 node a_j and STOP;
2. Set $k = j$;
3. Set $p = 1$;
4. Put in steck L the pair (k, p);
5. If $k \neq 1$ set k equal to the p — th element of vector m_k and go to step 3;

6. Print the nodes which have as indices the first elements of the pairs of steck Lin the order from the top to the bottom of the (without emptying the steck), thus obtaining a minimum path from a_1 to a_j; pair (k, p) is taken out of steck L;

7. If steck L is empty, STOP;

8. Take out pair (k, p) of the steck;

9. If p is equal to the number of elements in vector m_k go to step 7;

10. Set $p = p + 1$ and go to step 4.

In the case of programming on a computation system, use for M a rectangular matrix with a number of lines equal to the number of the nodes of the digraph and a number of columns equal to the maximum number of elements appearing at a given moment in a vector (if no better estimation can be found, a number of columns equal to the biggest of the interior grades of the nodes of the digraph *) can be considered or, if this number cannot be estimated, $n - 1$ columns are considered, where n is the number of nodes of the digraph). Steck L will be a matrix with 2 lines and at the most n columns.

The modified Dantzig algorithm too can be put in certain forms, so that the path or minimum paths from a_1 to a_n may be found. In order to find a minimum path (if there is one) the following algorithm is used:

Algorithm 13. Vectors c and c' are used as in algorithm 8, a vector M whose elements will be the indices of several nodes, files L, $L1$, $L2$ and matrix W of weights.

1. Initialize vector M with 0, vectors c and c' with -1 and $c_1 = 0$, $c'_n = 0$;

2. Set $d = \infty$ and initialize steck L as an empty steck;

3. For all i and j for which $c_i \neq -1$, $c_j = -1$ and $w_{ij} \neq \infty$ if $c_i + w_{ij} = d$ put pair (i, j) in steck L and if $c_i + w_{ij} < d$ empty steck L, then put pair (i, j) in steck L and set $d = c_i + w_{ij}$;

4. If steck L is empty, there is no path from a_1 to a_n and STOP;

5. For all pairs (i, j) for example for which $c'_j = -1$ and $c_j = -1$ set $m_j = i$ and $c_j = d$; if there is in the steck at least a pair (i, j), so that $c'_j \neq -1$ go to step 10;

6. Set $d = \infty$ and initialize steck L as an empty steck;

7. For all i and j for which $c'_j = -1$, $c'_j \neq -1$ and $w_{ij} \neq \infty$, if $c'_j + w_{ij} = d$ put pair (i, j) in steck L and if $c'_j + w_{ij} < d$ empty steck L, then put in it pair (i, j) and set $d = c'_j + w_{ij}$;

8. If steck L is empty, there is no path from a_1 to a_n and STOP;

9. For all pairs (i, j) in the steck for which $c'_j = -1$ and $c_i = -1$ set $m_i = j$ and $c'_j = d$; if for all pairs (i, j) in the steck we had $c_i = -1$ go to step 2;

10. Establish a pair (i, j) for which $c_i \neq -1$, $c'_j \neq -1$ and $c_i + c'_j + w_{ij}$ has the lowest value (this value represents the length of the minimum path);

11. Put i in steck $L1$;

12. If $i \neq 1$ set $i = m_i$ and go to step 11;

We call the interior grade of a node, the number of arces entering that node and they come from nodes other than the given node.

13. Put j in stecks $L2$;

14. If $j \neq n$ set $j = m_j$ and go to step 13;

15. Print the symbols of nodes which have as indices the elements of steck $L1$ and $L2$ in the order from the top to the bottom of steck $L1$ and then from the bottom to the top of steck $L2$, obtaining the nodes which belong to the minimum path from a_1 to a_n and STOP.

In order to find all minimum paths from a_1 to a_n the following algorithm id applied:

Algorithm 14. Use vectors c and c', stecks L, $L1$ and $L2$ and matrix W with the significance as in algorithm 13 and vector M has the significance as in algorithm 11 and 12.

1. Initialize vector M by putting all components equal to the empty set, vectors c and c' by putting all their components equal to -1 and $c_1 = 0$, $c'_n = 0$;

2. Set $d = \infty$ and initialize steck L as an empty steck;

3. For all i and for which $c_i \neq -1$ and $w_{ij} \neq \infty$ if $c_i + w_{ij} = d$ put pair (i, j) in steck L, and if $c_i + w_{ij} < d$ empty steck L and then put pair (i, j) in steck L and set $d = c_i + w_{ij}$;

4. If steck L is empty, there is no path from a_1 to a_n and STOP;

5. For all pairs (i, j) in the steck for which $c'_j = -1$ and $c_i = -1$ set $m_j = m_j \cup \{i\}$ and $c_j = d$; if there was in the steck at least a pair (i, j) so that $c'_j \neq -1$ go to step 10;

6. Set $d = \infty$ and initialize steck L as an empty steck;

7. For all i and j for which $c'_i = -1$, $c'_j \neq -1$ and $w_{ij} \neq \infty$ if $c'_i + w_{ij} = d$ put pair (i, j) into steck L, and if $c'_j + w_{ij} < d$ empty steck L, then put pair (i, j) into steck L and set $d = c'_j + w_{ij}$;

8. If steck L is empty, there is no path from a_1 to a_n and STOP;

9. For all pairs (i, j) in the steck for which $c_i = -1$ and $c'_i = -1$ set $m_i = m_i - \{j\}$ and $c'_i = d$; if there was not in the steck at least a pair (i, j) so that $c_i \neq -1$ go to step 2;

10. Establish pairs (i, j) with $i \neq j$ for which $c_i \neq -1$, $c'_j \neq -1$ and $c_i + c'_j + w_{ij}$ has the minimum value and put in steck L;

11. If steck L is empty, STOP;

12. Take out pair (i, j) of steck L;

13. Set $p = 1$;

14. Put pair (i, p) in steck $L1$;

15. If $i \neq 1$ set i equal to the p-th element of vector m_i and go to step 13;

16. Set $p = 1$;

17. Put pair (j, p) in steck $L2$;

18. If $j \neq n$ set j equal to the p-th element of vector m_j and go to step 16;

19. Print the symbols of nodes whose indices are the first elements of the pairs found in steck $L1$ and $L2$ in the order from top to the bottom of steck $L1$ and then from the bottom to top of steck $L2$, obtaining the nodes which belong to a minimum path from a_1 to a_n; take pair (j, p) out of steck $L2$;

20. If steck $L2$ is empty go to step 22;

21. Take out pair (j, p) of the steck and if p is equal to the number of

elements in vector m_j go to step 20, otherwise set $p = p + 1$ and go to step 17;

22. Take out pair (i, p) of steck $L1$;

23. If steck $L1$ is empty, go to step 11;

24. Take out pair (i, p) of steck $L1$ and if p is equal to the number of elements in vector m_i go to step 23, else set $p = p + 1$ and go to step 14.

In the programming on a computation system of algorithms 7—14 for digraphs with a big number of nodes, we use techniques similar to those described previously for algorithms 1—6. However, we shall add that the number of calculi in algorithms, 7, 8, 9, 11, 13 and 14 becomes smaller if we use a list of connections [3, 4] in which each cell has four elements representing respectively the origin of an arc, the extremity of the arc, the calculated value of d for the extremity (origin) of the arc taking into account the value of d for the origin (extremity) of the arc and its share and the fourth element is a connection with another cell (contains 0 if the cell is the last in the list). Appearing in the list are all arces coming out of set A_i (entering set A'_i) at a given moment, ordered in keeping with the third element of the cell. The cells containing in the second (first) place the index of a node which is added to sets A_i (A'_i) are taken out, and the arces setting out from that node to the nodes which are not in A_i (respectively $A)'_i$ are added. The free cells are organized as an empty cell [3]. The algorithms end when the linear list defined above no longer contains any arc in algorithms 7, 9 and 11 for algorithms 8, 13 and 14, when one of the two lists is empty or at least one element is found which should be put in set A_i or A'_i and it already belongs to the other set.

REFERENCES

1. BÂSCĂ O.: :*Sisteme de prelucrare şi teleprelucrarea datelor* (Operating Systems and Data Teleprocessing). Tipografia Universităţii din Bucureşti, 1976.
2. BENTLEY D., COOKE K.: : *Convergence of Successive Approximations in the Shortest Route Problem.* J. Math. Anal. Appl., 10, 1965, 269—274.
3. BERZTISS A. T.: *Data Structures; Theory and Practice.* Academic Press, 1971.
4. BILDE O., KRARUP J.: *A Modified Cascade Algorithm for Shortest Paths.* Metra, 8, 1969, 231—241.
5. DANTZIG G.: *On the Shortest Route through a Network.* Management Sci., 6, 1960, 187.
6. DANTZIG G.: *All Shortest Routes in a Graph;* Theory of Graphs. (International Symposium). International Computer Centre, Rome. Gordon and Breach, New York and Dunod, Paris, 1967, 91—92.
7. FARBEY B., LEND A., MURCHLAND J.: *The Cascade Algorithm for Finding all Shortest Distances in a Directed Graph.* Management Sci., 14, 1967, 19—28.
8. FLOYD R.: *Algorithm 97; Shortest Path.* Comm. ACM, 5, 1962, 345.
9. KNUTH D. E.: *Tratat de programarea calculatoarelor.* (Treatise for Programming the Computers) Editura Tehnică, 1974.
10. NOLIN L.: *Traitement des données groupées.* Publication de l'Institut Blaise-Pascal, Paris, 1964.
11. POLLACK M., WIEBENSON W.: *Solutions of the Shortest-Route Problem. A review.* Operations res., 1960, 224—230.

12. ROBERT P., FERLAND J.: *Généralisation de l'algorithme de Warshall.* Revue Française Informat. Recherche Opérationelle, 7, 1968, 71—85.
13. TOMESCU I.: *Un algorithme pour la détermination des plus petites distances entre les sommets d'un réseau.* Revue Française Informat. Recherche Opérationnelle, 5, 1967, 133—139.
14. TOMESCU I.: *Grafuri şi programare lineară* (Graphs and Linear Programming). Editura didactică şi pedagogică, 1975.
TOMESCU I.: *Introduction to Combinatorics.* Collet's (Publishers), LTD. London and Wellinborough, 1975.

DE CERTAINS ALGORITHMES DE CHEMINS MINIMA DANS LES GRAPHES

Le but de cet article est celui de présenter quelques variantes des algorithmes classiques, Warshall modifié et Dantzig modifié, pour trouver les chemins minima dans un graphe, ainsi que des indications concernant la programmation de ces algorithmes en système de calcul. Une partie de ces algorithmes ont été élaborés à l'occasion d'une recherche scientifique à base de contrat en vue de „l'optimisation des trajets du transport en commun" à Bucarest.

О НЕКОТОРЫХ АЛГОРИТМАХ МИНИМАЛЬНОЙ СВЯЗИ В ГРАФАХ

Целью этой статья является анализ некоторых вариантов улучшенных классических алгоритмов Варшаля и Данцига, определяемых минимальные связи в ориентировочном графе (дирафе), а также указания по програмированию этих алгоритмов на вычислительной системе. Часть этих алгоритмов была разработана в рамках научно-исследовательской темы на основе договора «Оптимизации пассажирского транспорта в муниципим Бухарест».

14.I.1977