Olympiads in Informatics, 2011, Vol. 5, 82–91

Reconstruction of Trees Using Metric Properties *

Krassimir MANEV[1], Nikolai NIKOLOV[2], Minko MARKOV[1]
[1]*Department of Mathematics and Informatics, Sofia University*
J. Bourchier blvd. 5, 1164 Sofia, Bulgaria
[2]*Institute of Mathematics and Informatics, Bulgarian Academy of Sciences*
G. Bontchev str. 8, 1113 Sofia, Bulgaria
e-mail: manev@fmi.uni-sofia.bg, nik@math.bas.bg, minkom@fmi.uni-sofia.bg

Abstract. To reconstruct a graph from some of its elements or characteristics is a hard problem. Reconstructions require very good mathematical background and programming skills. That is why some not so difficult graph reconstruction problems may be appropriate for competitions in programming both for university and school students. In this paper two such problems are considered – reconstruction of a tree from the distances between its outer vertices and from the distances between all its vertices. It is proved that any tree can be reconstructed in either case. The corresponding algorithms are presented and their time complexities are estimated.

Key words: graphs, trees, shortest paths, reconstruction of tree.

1. Introduction

In the preface of a book dedicated to the 60th anniversary of Tutte the great Paul Erdös writes: "There are three diseases in graph theorists. The first is four-color-disease, the second is reconstruction-disease, and the third – Hamiltonian-disease." Regarding *reconstruction* Paul Erdös without any doubt had in mind a hypothesis conjectured by Kelly (1942) and recited by Ulam (1960). The hypothesis became popular as the *graph reconstruction conjecture* from the paper of Harary (1964), where it was reformulated in a weaker form: every graph $G = (V, E)$, $V = \{v_1, v_2, \ldots, v_n\}$, $n \geqslant 3$, can be reconstructed from the set of its vertex-deleted subgraphs $\{G_1, G_2, \ldots, G_n\}$, where G_i is obtained from G by deleting v_i and its incident edges.

That conjecture is still neither proved nor disproved for arbitrary graphs and is considered one of the hardest graph problems. An easier problem of graph reconstruction, to reconstruct a graph from the set of subgraphs, each one obtained by a single edge removal from the original graph, was formulated and solved in Harary and Palmer (1966).

Graph reconstruction problems can be formulated in different ways and using different elements or characteristics of the graph. These problems are usually not trivial and good mathematical and algorithmic background is necessary for solving them. It is possible, however, to formulate reconstruction problems for restricted classes of graphs that are not too hard and thus to propose interesting tasks for competitions in programming.

*This work was partly supported by Scientific Research Fund of Sofia University through Contract 242/2010.

In this paper we consider two problems of reconstruction of trees from given metric characteristics, *viz.* the distances between some pairs of vertices. We prove such reconstructions are possible and the reasoning is not very involved. That is why we included these problems in two programming contests, one for university students and one for high school students, and they were successfully solved by some students.

In Section 2 we give some basic definitions and prove some properties that are used further on. In Section 3 we formulate the first problem and propose an algorithm that solves it. In Section 4 we formulate the second problem and present a version of the algorithm from Section 3 that solves it. In Section 5 we estimate the time complexity of both algorithms. We outline and discuss an important subproblem that can arise in that estimation. Section 6 contains some conclusions and ideas for future research.

2. Basic Notions and Properties

Some necessary notions are defined below. The other definitions we use can be found in each textbook on graph algorithms. We consider *finite undirected graphs* without loops or multiple edges. The graph G with vertex set V and edge set E is denoted by $G = (V, E)$. By $V(G)$ we denote the vertex set of G in case that vertex set is not specified. Without loss of generality we assume $V = \{1, 2, \ldots, n\}$. For each edge e with endpoints, say, u and v we denote e by either (u, v) or (v, u). For any vertex v, the number of edges incident with v is denoted by $\deg(v)$ and called the *degree of* v. To each edge (u, v), a positive real number $c(u, v)$ is assigned called the *length of* (u, v). If the edge lengths are not specified explicitly each one of them is assumed to be 1 and such graphs are called *non-weighted*.

The alternating sequence $\pi = v_0, e_0, v_1, e_1, \ldots, v_{k-1}, e_{k-1}, v_k$ of distinct vertices and edges where $v_i \in V$ for $i = 0, 1, \ldots, k$ and $e_i = (v_i, v_{i+1}) \in E$ for $i = 0, 1, \ldots, k-1$, is called a *path between* v_0 and v_k. The *length of* π is equal to $\sum_{i=0}^{k-1} c(e_i)$ and is denoted by $c(\pi)$. Every $v \in V$ is a *trivial* path between v and v of length 0. The edge names are omitted when we describe a path. The path $\pi_0 = v_0, v_1, \ldots, v_k$ is called a *shortest path* between v_0 and v_k if $c(\pi_0) \leqslant c(\pi)$ for each other path π from v_0 to v_k. A graph G is *connected* if there is a path between every two of its vertices.

The *distance between u and v* is the length of any shortest path between them and is denoted by $d(u, v)$. We emphasize that the function $d(u, v)$ defined in any connected graph has the three properties of any other function in mathematics called distance:

1. $d(u, v) \geqslant 0 \forall u, v \in V$, and $d(u, v) = 0$ iff $u = v$;
2. $d(u, v) = d(v, u) \forall u, v \in V$;
3. $d(u, v) + d(v, w) \geqslant d(u, w) \forall u, v, w \in V$ (triangle inequality).

Therefore, when solving distance problems on graphs it is helpful to use analogies with the more familiar Euclidean space. Of course, these analogies have to be used carefully. For example, in Euclidean space the triangle inequality will turn to equality iff the point v lies on the linear segment between u and w, while in the metric space of the graph the triangle inequality will turn to equality iff vertex v lies on any shortest path between u and w.

The commonplace definition of a *tree* is a connected graph with no cycles, and of a *rooted tree*, a tree in which one vertex is chosen to be the *root* and the *leaves* are defined via the maximum length paths having the root as one endpoint. We find the following inductive definition of *rooted tree* more useful:

(i) The graph $T = (\{r\}, \varnothing)$ is a rooted tree, r is both the root and the sole leaf of T.

(ii) Let $T = (V, E)$ be a tree with root r and leaves $L = \{v_1, v_2, \ldots, v_k\}$, let $u \in V$, and $w \notin V$. Then $T' = (V \cup \{w\},\ E \cup \{(u, w)\})$ is also a rooted tree. Its root is r and its leaves are $(L - \{u\}) \cup \{w\}$.

Obviously, every rooted tree is a tree and every tree can be turned into a rooted one. Many properties are valid both for trees and rooted trees. For example, for each (rooted) tree $T = (V, E)$ the equality $|V| = |E| + 1$ holds, there is a unique path between each two vertices u and v of a (rooted) tree, etc.

Suppose T is a non rooted tree. For every $u \in V(T)$, *the subtrees of T induced by u* are the connected components obtained after the deletion of u and its incident edges. The vertices of degree one are the *outer vertices* and the other vertices are the *inner vertices*. A tree with one vertex only is called *trivial*. If all inner vertices have degree $\geqslant 3$, T is called *homeomorphically irreducible*, shortly *irreducible*. We find the following equivalent (except for the trivial tree) inductive definition of irreducible trees useful:

(i) $T = (\{u, v\}, \{(u, v)\})$ is an irreducible tree with outer vertices u and v.

(ii) Let $T = (V, E)$ be an irreducible tree, $|V| \geqslant 2$, and the outer vertices be $W = \{v_1, v_2, \ldots, v_t\}$. Let $w \in W$, and let $x_1, \ldots, x_k$ for some $k \geqslant 2$ be vertices not in V. Then $T' = (V \cup \{x_1, \ldots, x_k\}, E \cup \{(w, x_1), \ldots, (w, x_k)\})$ is also an irreducible tree. Its outer vertices are $(W - \{w\}) \cup \{x_1, \ldots, x_k\}$.

3. Reconstruction of a Tree from the Distances Between Its Outer Vertices

Consider the following reconstruction problem.

Problem 1. Let $T = (V, E)$ be non-weighted tree with outer vertices $L = \{1, 2, \ldots, k\}$. Let the function d': $L \times L \to N$ be the restriction of the distance function d: $V \times V \to N$ of T on $L \times L$, i.e., $d'(i, j) = d(i, j),\ 1 \leqslant i \leqslant k,\ 1 \leqslant j \leqslant k$. Given L and d', reconstruct T. In the discussions below we will use only $d(i, j)$, having in mind that $d(i, j) = d(i, j)$‘ when i and j are outer vertices. Otherwise $d(i, j)$ is calculated by our algorithm.

The main question regarding every graph reconstruction is, is it possible to reconstruct the graph from the given elements or characteristics. That is why we proceed immediately with the following theorem.

Theorem 1. *Each non-weighted tree $T = (V, E)$ is uniquely determined by the distances between all couples of its outer vertices.*

Lemma 1. *Let $T = (V, E)$ be non-weighted tree and u, v, and w be any three pairwise distinct vertices from V. Let p, q_1, and q_2 be the (unique) paths between u and v, between*

u and w, and between v and w, respectively. Let q' be the maximum common subpath of q_1 and q_2 that has w as one endpoint. Let x be the other endpoint of q', if $c(q') \geqslant 1$, or $x = w$, else. Call x, the connecting point of w and p. Then x is uniquely defined by $d(u, v), d(u, w)$, and $d(v, w)$.

Proof of Lemma 1. It is obvious that x is a vertex in p, therefore:

$$d(u, x) + d(x, v) = d(u, v).$$

Furthermore,

$$d(u, x) + d(x, w) = d(u, w), \quad \text{and}$$
$$d(v, x) + d(x, w) = d(v, w).$$

The system of these three equations in three unknowns has a unique solution:

$$d(u, x) = (d(u, v) + d(u, w) - d(v, w))/2,$$
$$d(v, x) = d(x, v) = (d(u, v) + d(v, w) - d(u, w))/2,$$
$$d(w, x) = d(x, w) = (d(u, w) + d(v, w) - d(u, v))/2.$$

Because of the triangle inequalities for u, v and w, these three values are always nonnegative integers. So, x is a uniquely defined vertex in T. See Fig. 1 for illustration.

Proof of the Theorem. Let L be the set of the outer vertices of T. Choose arbitrarily two distinct $u, v \in L$. Let $d = d(u, v)$ and p be the path between u and v(see Fig. 1). Call p, the *backbone of* T. Since we know the number $d - 1$ of the inner vertices of p it is trivial to reconstruct p. As suggested by Lemma 1, for each outer vertex $w \notin \{u, v\}$ we compute x, the connecting point of w and p, and the distance $d(w, x)$. Let $T_0, \ldots, T_d$ be the rooted subtrees of T relative to the backbone (see Fig 1). Let $L_i = L \cap V(T_i)$, for $0 \leqslant i \leqslant d$. Each nontrivial T_i can be reconstructed using L_i as follows. Choose an arbitrary $w \in L_i$. Let x be the root of T_i. Apply the procedure described so far for the subtree T_i with $L_i \cup \{x\}$ as set of its outer vertices and w and x as the endpoints of its backbone. Note that x must be treated as an outer vertex although $\deg(x)$ in T_i can be $\geqslant 1$, the

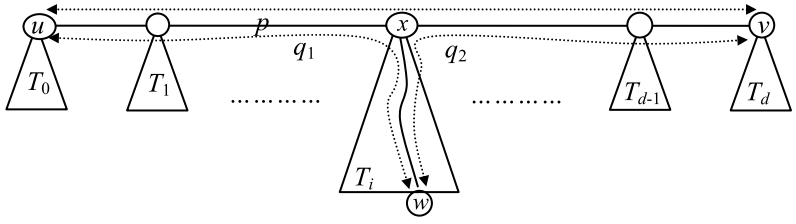

Fig. 1. Reconstruction by outer vertices. x is the connecting point of w and p.

reason being that we know (by Lemma 1) and need the distances between x and the other vertices from L_i.

Proceed recursively within the rooted subtrees of T_i relative to its backbone in order to reconstruct T_i. Applying the same procedure to each nontrivial subtree relative to the backbone of T, we reconstruct the whole tree T.

Suppose that the distances between outer vertices of T are kept in the two dimensional matrix `D`, `D[i][j]` $= d'(i,j)$ and `vert` and `dist` are defined as `int`. The calculation of Lemma 1 is performed by the function

`vert find(vert u, vert v, vert x, dist * d)`,

which returns a not used vertex w, such that the vertex x is the connection point of w and the path from u to v. Further, find stores in d the distance between x and w. If there is no such w (i.e., the subtree rooted in x is trivial) the function return 0.

The proof of Theorem 1 allows us to verify the following algorithm which by a set L of outer vertices of a tree $T, \mid L\mid = K$, and the distances between each couple of vertices of L, reconstructs T(as a list of edges):

Algorithm 1

```
vert new; dist D[MAXK][MAXK];
int used[MAXK];

reconstruct(vert from, vert to, dist d) {
  vert w,old; dist di;
  used[to]=1; old=from;
  if(w=find(from,to,from,&di))
    reconstruct(from,w,di);
  for(int i=1;i<=d-1;i++){
    output(old, new);
    if(w=find(from,to,new,&di);
      reconstruct(new,w,di);
    old=new++;
  }
  output(old,to);
  if(w=find(from,to,to,&di);
  reconstruct(to,w,di);
}
int main() {
  input(K,D[][]);
  used[1]=1;new=K+1;
  reconstruct(1,2,D[1][2]);
}
```

Note that if the tree is weighted the reconstruction by the distances between outer vertices is impossible in general. Indeed, if x is a vertex of degree 2 and y and z are its neighbors there are infinitely many possibilities for $d(x, y)$ and $d(x, z)$ such that $d(x, y)+ d(x, z) = d(y, z)$. However, a small modification of Algorithm 1 solves the problem for irreducible weighed trees. In this case every vertex of the backbone is a root of some nontrivial tree relative to that backbone. We can identify the inner vertices, call them $x_1, x_2, \ldots, x_q$, of the backbone using Lemma 1. To reconstruct the backbone it suffices to sort the distances $d(u, x_1), d(u, x_2), \ldots, d(u, x_q)$, where u is one of the endpoints of the backbone. So we proved the following theorem.

Theorem 2. *Suppose T is a weighted tree. T can be reconstructed from the distances between its outer vertices iff T is irreducible.*

4. Reconstruction of a Tree from the Distances Between All Vertices

Problem 2. Let $T = (V, E)$ be weighted tree and d: $V \times V \to N$ is the distance function of T, i.e., $d(v_i,\ v_j)$ is the weighted distance between v_i and v_j, $1 \leqslant v_i \leqslant n$, $1 \leqslant v_j \leqslant n$. Given V and d, reconstruct T.

If all edges have the same weight C then the problem is trivial: the smallest $N - 1$ of the distances will be equal to C and they will identify the edges uniquely. So we consider trees with arbitrary lengths of the edges and we prove the following theorem.

Theorem 3. *Each weighted tree $T = (V, E)$ can be reconstructed from the distances between all its vertices.*

Proof. Proceeding as in Theorem 1 we choose arbitrarily two distinct vertices u and v and call the path p between them the *backbone.* Any vertex x belongs to p iff $d(u, x) + d(x, v) = d(u, v)$. Let the inner vertices in p be $x_1, x_2, \ldots, x_q$. As mentioned above, p can easily be reconstructed by sorting the distances $d(u, x_1), d(u, x_2), \ldots, d(u, x_q)$. Using Lemma 1, for each vertex w not in p we can compute the corresponding connecting point x_i by searching for the value $(d(u, v) + d(u, w) - d(v, w))/2$ in the sorted list of distances.

Consider, as in Theorem 1, a non trivial subtree T_i relative to the backbone, T_i being rooted at x_i. Let w be any other vertex in T_i. Let the path between x_i and w be the backbone of T_i. Reconstruct that backbone as above and proceed recursively with the rooted subtrees of T_i relative to its backbone. Thus we reconstruct T_i. Applying that procedure to each non trivial subtree T_j we reconstruct the whole tree T.

Some elementary modifications of Algorithm 1 are necessary in order to obtain Algorithm 2 that solves Problem 2. First, `dist` could be defined now as `double`. The function `int build_path(vert u, vert v, int **p int)` reconstructs the path p from u to v by all vertices x such that $d(u, x) + d(x, v) = d(u, v)$, sorted

by $d(u, x)$ and mark all these vertices as used. The function int find(vert x) returns an unused vertex w such that its connecting point to previously found path p is x or 0 if no such w exists.

Algorithm 2

```
dist D[MAXN][MAXN]; int used[MAXN]

reconstruct(vert from, vert to) {
  vertex w, path[MAXN]; dist di; int psize;
  psize=build_path(from, to, path);
  for(int i=0;i<psize;i++) {
    output(path[i], path[i+1]);
    if(w=find(path[i])) reconstruct(path[i],w);
  }
  if(w=find(to)) reconstruct(to,w);
}
int main() {
  input(N,D[][]); reconstruct(1,2);
}
```

5. Time Complexity of the Algorithms

Let us first estimate the time complexity of Algorithm 2. Because the size of the input is at worst $\Theta(n^2)$ where $n = |V|$ we estimate that complexity as a function, say $t_2(m)$, of $m = n\ (n-1)/2$. The reading of the input data necessitates $\Theta(m)$ steps. The algorithm breaks the vertex set of the tree into subsets, each one of which induces a path, those paths being pairwise edge-disjoint (but not necessarily vertex-disjoint), and each recursive call works on one of those paths. The number r of those paths cannot be greater than the number $n-1$ of edges so r is $O(n)$. Let the paths have $n_1, n_2, \ldots, n_r$ vertices respectively. Then $n_1 + n_2 + \ldots + n_r < n + r < 2n$ because each path but the first one contains exactly one vertex that we have already considered.

During the ith recursive call we need $O(n)$ steps to find the n_i vertices that belong to the corresponding path, $O(n_i \lg n_i)$ steps to sort them and to reconstruct the path and $O(n)$ steps to choose one vertex from each of the subtrees rooted at a vertex from that path. So all recursive calls need $2r.O(n) + \Sigma_i O(n_i \lg n_i)$. But $n_1 + n_2 + \ldots + n_r < 2n$, therefore $\Sigma_i O(n_i \lg n_i) = O(n \lg n)$. So the complexity of Algorithm 2 is linear:

$$\begin{aligned} t_2(m) &= O(m) + 2r.O(n) + \Sigma_i O(n_i \lg n_i) \\ &= O(m) + O(n).O(n) + O(n \lg n) = O(m). \end{aligned}$$

The estimation of the time complexity of Algorithm 1 presents us with the following challenge. Assume the output is T represented by, say, adjacency lists. Then the number

of steps performed by Algorithm 1 is at least as big as the number of the inner vertices. That number can vary dramatically for different trees with the same number of outer vertices. As one extreme example let T be a path. The input in this case is a single number $n-1$: the length of the path. If we make the standard simplifying assumption that any integer necessitates only a constant amount of space to be represented and can be operated upon in constant time, it will turn out that the worst case time complexity is *infinite* since n can be arbitrarily large, therefore there is no *a priori* upper bound on the number of vertices of T. Even if we take into consideration the length of n and express the time complexity of Algorithm 1 as a function of that length, in the worst case the time complexity will be exponential for obvious reasons.

However, if instead of T we output its *equivalent irreducible tree* (defined below) it turns out the time complexity is linear in the length of the input, under the standard assumption that integers require constant space and can be operated upon in constant time. Notice that the problem with the traditional representation is the possibility of long paths in the tree of vertices of degree two. It is because of such paths that the inner vertices can be exponentially more than the outer ones. If there are no vertices of degree two then the number of inner vertices cannot even exceed the number of outer ones, as shown by the following:

Lemma 2. *Let T be an irreducible tree, n_{out} be the number of its outer vertices, and n_{in}, the number of its inner vertices. Then $n_{in} \leqslant n_{out} - 2$.*

Proof. By induction on the number of steps that the inductive definition has been carried out. The basis is a single edge: obviously, $0 \leqslant 2 - 2$. Assume the claim holds for some irreducible tree. Consider any irreducible tree obtained from it by adding k new outer vertices to some outer vertex w where $k \geqslant 2$. Since w is not an outer vertex any more, the number of outer vertices goes up by $k-1 \geqslant 1$, and the number of inner vertices goes up by one. The inequality is preserved.

The equivalent irreducible tree of any nontrivial, non-weighted tree T is the weighted irreducible tree T' that is obtained from T after substituting every maximal path $p = u, x_1, x_2, \ldots, x_{q-1,}v$, such that $x_1, x_2, \ldots, x_{q-1}$ are vertices of degree two by the single edge $e = (u, v)$ with weight $c(e)$ in T' equal to length of p in T, i.e., q. Clearly, the degree two vertices disappear so T' is indeed irreducible. Notice that the other vertices are not affected by this operation and it is in that sense that T' is equivalent to T. It is trivial to restore T from T'.

We estimate the time complexity of Algorithm 1 as a function, call it $t_1(m)$, of $m = n_{out}(n_{out} - 1)/2$. Let n_{in} is the number of inner vertices of degree at least 3 and $n = n_{in} + n_{out}$. From Lemma 2, $n_{in} = O(n_{out})$ and so $n = O(n_{out})$ too. The reading of input data will need $O(n_{out}^2)$ steps and printing of the output in irreducible form – $O(n) = O(n_{out})$ steps. Let us denote again with t the number of paths without common edges passed to the recursive calls and with $n_1, n_2, \ldots, n_t$ – the number of inner vertices of these paths, respectively. As it was mentioned above t is $O(n_{out})$ and

$n_1 + n_2 + \cdots + n_t < n_{in} + t = O(n_{out})$, because each path use at most one inner vertex of degree at least 3, used by some other path also.

Now the estimation of t_1 is almost the same as of t_2. During ith recursive call of Algorithm 1 we first find for each not used outer vertex w the vertex x from the backbone where the subtree of w is rooted – $O(n_{out})$ steps – and identify in such way n_i inner vertices. For reconstruction of the backbone in irreducible form it will be enough to sort the found inner vertices – $O(n_i \lg n_i)$ steps. So all recursive calls will need $t.O(n_{out}) + \Sigma_i O(n_i \lg n_i) = O(n_{out}^2) + O(n_{in} \lg n_{in}) = O(n_{out}^2) + O(n_{out} \lg n_{out})$ and the complexity of Algorithm 1 is $t_1(m) = O(n_{out}^2) + O(n_{out}) + O(n_{out}^2) + O(n_{out} \lg n_{out}) = O(n_{out}^2) = O(m)$.

We suppose that it is not a problem for the reader to make the necessary changes in Algorithm 1 and to obtain a new one with linear time complexity.

6. Comments and Conclusions

The first problem was proposed to the South-East European Regional Round of ACM ICPC in October 2010 (ACM ICPC, 2010 → Problems → Problem G) where 52 teams of university students from 7 countries took part, among them 33 teams from traditionally very strong Ukraine, Romania and Bulgaria. Only 11 teams succeeded in solving the problem (one problem was not solved at all and one was solved by 5 teams). This gave us some reason to classify the Problem 1 as harder than average.

The second problem, which we consider easier, was proposed to the traditional Bulgarian Autumn Tournament in Informatics for school students (INFOS, 2010). The problem was included in the problem set for the second age group (till 16 years) with 53 participants, including students from Croatia, Greece, FYR of Macedonia, Serbia and Romania. We supposed that it would be the hardest problem in the set. Unexpectedly 43 students submitted solutions for this problem (more than for any of the other two problems), 15 of the solutions got at least 70% of the grading marks and 10 solutions got 100%.

In our opinion the second version was more attractive for the contestants than the first because of the existence in the first problem of the mentioned above dependence of the time complexity from a hidden (searched) parameter of the graph. Each such parameter could involve additional difficulties. It seems that some efforts have to be spent during the training of contestants in order to be able to manage tasks with hidden parameters like graph reconstruction problems. Anyway, graph reconstruction problems seem to be a good way to enlarge the scope of the tasks given at programming contests.

With respect to the reconstruction of trees by some metric properties, it will be interesting to try to reconstruct the tree from the distances between all vertices or between the outer vertices when it is not given which distance is between which two vertices. For solutions of the discussed two problems this knowledge is crucial.

References

ACM ICPC (2010). *ACM ICPC South-East European Regional Contest*. `http://acm.ro`.

Harary, F. (1964). On the reconstruction of a graph from a collection of subgraphs. In: Fiedler, M. (Ed), *Proc. of the Symposium Theory of Graphs and its Applications*, Czechoslovak Academy of Sciences, Prague, 47–52.

Harari, F., Palme, E.M. (1966). The reconstruction of a tree from its maximal proper subtrees, *Can. J. Math.*, 18, 803–810.

Kelly, P.J. (1942). *On Isometric Transformations*, Doctoral Thesis, University of Wisconsin.

INFOS (2010), *Autumn Tournament in Informatics "John Atanasov"*. `http://math.bas.bg/infos/`.

Ulam, S.M. (1960). *A Collection of Mathematical Problems*, Wiley (Interscience), New York.

K. Manev is an associated professor of discrete mathematics and algorithms in Sofia University, Bulgaria, PhD in computer science. He was a member of Bulgarian National Committee for Olympiads in Informatics since 1982 and president of NC from 1998 to 2002. He was member of the organizing team of IOI'1989, IOI'1990, vice president of IOI'2009 and leader of Bulgarian team for IOI in 1989, 1998, 1999, 2000 and 2005. From 2000 to 2003 he was elected member of IC of IOI, since 2005 to 2010 represented in IC the host country of IOI'2009, and during IOI'2010 was elected as a member of IC of IOI again for the period 2010–2013.

N. Nikolov is an associated professor (2002) in section "Complex Analysis", Institute of Mathematic and Informatics, Bulgarian Academy of Sciences; Dr.Sc. (2010), Ph.D. (2000). His main scientific interests are in analysis of functions of several complex variables. Leader from 2009 and deputy leader (2004–2008) of the Bulgarian team for the International Mathematical Olympiad as well as leader (2004–2008) and deputy leader (1999–2003) of the team for the Balkan Mathematical Olympiad.

M. Markov is a lecturer and lab instructor of discrete mathematics and algorithms at the Faculty of Mathematics and Informatics, Sofia University, Bulgaria. He is a PhD in computer science from the Institute of Mathematics and Informatics at the Bulgarian Academy of Sciences.