Tasks in Graphs

Kr. Manev
Sofia University, BULGARIA

"In the beginning…"[1]

- In **May 1989** the first IOI take place in Pravetz, Bulgaria with a single

  Task. A sequence of $2N$ boxes is given, $N-1$ of them filled with letter **A**, $N-1$ with letter **B** and 2 consecutive boxes are empty. Moving two consecutive letters (saving their order) in the empty boxes is permitted. Find the minimal number of moves that are necessary to arrange all letters **A** leftmost than all letters **B** (does not mater where the empty boxes are).

[1]The Bible, Genesis 1:1

"In the beginning…"[1]

- In **May 1987**, an open competition in programming for school students was organized just before and in connection with the International Conference of IFIP "Children in the Information Age"
- The contestants was separated in three age groups: under 14 years, under 16 years and older than 16 years (seniors).

“In the beginning…”

- **Tasks (seniors).** Bus stops in a city are labeled with 1, 2,…, N. All bus routes of the city are: $M_1 = (i_{1,1}, i_{1,2},\ldots, i_{1,m1})$, $M_2 = (i_{2,1}, i_{2,2},\ldots, i_{2,m2}),\ldots, M_r = (i_{r,1}, i_{r,2},\ldots, i_{r,mr})$, $1 \le i_{j,k} \le N$, $i_{j,k} \ne i_{j,l}$ when $k \ne l$. Each bus start from one end of its route, visit all stops of the route in the given order and, reaching the other end of the route, go back visiting all stops in reverse order. Write a program that, for given stops i and j, finds fastest possible way of getting from stop i to stop j by bus (if possible). Times for travel from stop to stop are equal and 3 times less than the time to change busses.

Graph structures

A ***graph structure*** $G(V,E)$ is composed of:

- finite set $V=\{v_1, v_2,\ldots,v_n\}$ of ***vertices;***
- ***collection*** $E = \{e_1, e_2,\ldots,e_m\}$ of ***links***. Each link e is connecting 2 vertices v and w: $e = (v,w)$.

Links could be ***directed*** (from v to w) – ***arcs*** or ***undirected – edges*** (notation is **the same** (v,w)).

Collection E could be set (then G is a ***graph***) or multi-set (then G is ***multi-graph***).

Graph structures

Combining directed/undirected with set/multi-set we obtain 4 categories of graph structures:

- Directed multi-graphs;
- Directed graphs or ***digraphs;***
- Undirected multi-graphs or ***multi-graphs;***
- Undirected graphs or ***graphs.***

Rem. Links (*i*,*i*) are called ***loops***. We will consider graph structures without loops.

Traversals in graph structures

- The sequence $v_0,v_1,\ldots,v_l$ of vertices of a digraph is called ***course of length*** l ***from*** v_0 ***to*** v_l, if there is an arc (v_i,v_{i+1}) for each $i = 0, 1, \ldots, l-1$. When $v_0 = v_l$ then the course is called ***circuit***.
- The sequence $v_0,v_1,\ldots,v_l$ of vertices of a graph is called ***path of length*** l ***from*** v_0 ***to*** v_l, if there is an arc (v_i,v_{i+1}) for each $i = 0, 1, \ldots, l-1$ and $v_{i-1} \neq v_{i+1}$. When $v_0 = v_l$ then the path is called ***cycle***.

Traversals in graph structures

- Graph structure in which there is a course, respectively path, from each vertex to each other vertex is called ***connected***.
- Directed graph structure in which for each two vertices there is a course in at least one of the two possible directions is called ***weakly connected***.

Traversals in graph structures

- Moving in a graph structure that pass trough each link once is called ***Euler moving*** (Euler course, Euler circuit, Euler path or Euler cycle).
- Moving in graph structure that pass trough each vertex once is called ***Hamilton moving*** (Hamilton course, Hamilton circuit, Hamilton path or Hamilton cycle).

Cost functions

- On each graph structure it is possible to define ***cost function*** – on vertices c_V: $V \rightarrow C$, on links c_E:$E \rightarrow C$, or both, where C is usually some numerical set of possible values.
- Values of the cost functions, beside ***cost***, are called also ***length***, ***weight***, ***reliability***, etc. depending of the situation.
- If a graph structure has no cost function defined then we will presume that the cost of each vertex and link is 1.

Cost functions

- Cost function is usually extended in some natural way on sub-graphs and other sub-structures defined in the graph structure. For example the ***cost of a path*** in a graph is usually defined as a sum of costs of its edges, of its vertices or both of vertices and edges (if applicable).
- The notion ***path*** (***course***) ***of a minimal cost***, called also ***shortest path*** (***course***) is fundamental for algorithmics in graphs.

Shortest path/course task

- ***Task 1*** (IOI'1989) **is a shortest path task:** Let V be the set of strings of length $2N$ composed of N–1 letters 'A', N–1 letters 'B', and 2 consecutive letters 'O'. Two vertices of V are linked by an edge if one of the strings could be obtained from the other by swapping letters 'O' and two other consecutive letters, conserving their order. The strings in which all letters 'A' are leftmost of all letters 'B' (does not mater where the letters 'O' are) are called ***final***. Write a program that for given string S finds one path of minimal length (trivial cost of each edge is 1) from S to some final string. If no paths between S and a final string, the program has to print corresponding message.

Shortest path/course tasks

- ***Task 2.*** (International Contest 1987) **is also a shortest path task**: A graph $G(V=\{1,2,\ldots,n\},E)$ is given. The set E of edges is defined by r of its paths of length m_1, m_2,…, m_r, respectively, in such a way that each edge of G is included in at least one of the given paths. The cost of each vertex is 3 and the cost of each edge is 1. Write a program that, for given two vertices v and w, to find the shortest path between v and w (if such path exists).

Shortest path is a distance

- Let $G(V,E)$ be a graph with cost function $c_E: E \rightarrow C$, where C is a numeric set with **non negative values**. Then the function $d: V \times V \rightarrow C$, where $d(v,w)$ is the cost of the shortest path from v to w is a ***distance*** in classic mathematical sense of the word because:
  - (i) $\forall\ v, w \in V,\ d(v,w) \geq 0$ and $d(v,w) = 0$ iff $v = w$;
  - (ii) $\forall\ v, w \in V,\ d(v,w) = d(w,v)$;
  - (iii) $\forall\ v, w, u \in V,\ d(v,w) \leq d(v,u) + d(u,w)$.

Shortest path is a distance

- Introducing of distance function gives us the possibility to consider the graph structure $G(V,E)$ as a **(geo)metric object** and to define, for example:
- ***Center*** of G – each vertex v, which minimize $D(v)=\max\{d(v,w)|w\in V\}$. If v is a center then $D(v)$ is called ***radius*** of G.
- The value $D(G)=\max\{d(v,w)|v,w\in V\}$ is called ***diameter*** of the graph G, etc.

Graph structures and relations

- A,B – sets. $R \subseteq A \times B$ is called ***relation***. Examples: "$x<y$", "$x \leq y$", "$x=y$", etc.; "the point p lye on the line l", "the line l pass trough the point p", "lines l and m are parallel" etc.; "$A \subseteq B$", "A and B intersects", etc.
- Relations found ***outside mathematics*** – "x is a son of y", "x likes y", "x and y are in a same class", etc; "the village x is linked with the village y by a road" (similar relations could be established among city crossroads linked by streets, railway stations linked by railway roads, etc.

Graph structures and relations

- Many tasks arise, in a natural way, in connection with a specific finite relation – abstract (mathematical) or from the real world.
- That is why it is important to know the properties of relations over Cartesian squares $A \times A$ – ***reflexivity***, ***symmetry***, ***anti-symmetry*** and ***transitivity***, as well as the most popular classes of relations – ***equivalences*** and ***orders***.

Graph structures and relations

- ***Finite relation*** is the same as ***digraph***. Indeed, each digraph $G(V,E)$ could be considered as a relation $E \subseteq V \times V$ and vice versa.
- Finite relation $E \subseteq V \times V$ which is ***symmetric*** **(*and optionally reflexive*)** is really a ***graph***.
- That is why, each task connected with some finite relation could be considered as a task in digraph or graph. For the following examples we will fix V to $\{1, 2, \ldots, n\}$.

Graph structures and relations

- ***Task 3***: Let $E \subseteq V \times V$ be an equivalence. Find the number of classes of equivalence of E. Is this number equal to 1? If the number of classes is great than 1, then find explicitly the classes of equivalence of E.
- **Graph formulation:** How many connected components has the graph $G(V, E)$? Is the graph connected? If not, then find the vertices of each connected component of G.

Graph structures and relations

- ***Task 4***: Let $E \subseteq V \times V$ be a total order (we will denote $(x,y) \in E$ with $x \leq y$) and $|V|=M$. Find a chain of all elements of V, *i.e.* such sequence $a_1, a_2, \ldots, a_M$ that $a_1 \leq a_2 \leq \ldots \leq a_M$. (sorting!!!)
- ***Task 5***: Let $E \subseteq V \times V$ be a partial order which is not total (we will denote $(x, y) \in E$ with $x \angle y$. Find a chain of elements of V with maximal length such that if $i<j$, $a_i \angle a_j$ or a_i and a_j are not comparable.

Graph structures and relations

- **Both tasks could be covered by the following graph formulation:** Given a digraph $G(V,E)$ without circuits. Find a course with a maximal length in G.
- Digraphs without circuits are very popular and have specific name – ***dag*** (**d**irected **a**cyclic **g**raphs).

Graph structures and relations

- ***Task 6***: Let $R_1 \subseteq A \times B$ and $R_2 \subseteq B \times A$ are such that $(a,b) \in R_1$ iff $(b,a) \in R_2$ (***mutually reversed***). Find a subset $\{(a_1,b_1), (a_2,b_2), \ldots, (a_M,b_M)\}$ of R_1 with maximal number of elements such that $a_i \neq a_j$ and $b_i \neq b_j$, $1 \leq i < j \leq M$.
- An example of mutually reversed relations from the real life could be the couple "the person *p* could do the work *w*" and "the work *w* could be done by the person *p*". For each couple of mutually reversed relations we can build a ***bipartite graph*** $G(V=A \cup B, R1)$ considering elements of $R1$ as not ordered.

Graph structures and relations

- Searched in task 6 subset M of edges such that each vertex is end of at most one edge in M is called ***maximal matching***.
- In a graph formulation the task will be: "Given a bipartite graph $G(V=A \cup B, R1)$. Find one maximal matching of G."
- Task for finding a ***maximal matching*** of the vertices in an **arbitrary graph** is much more difficult!

Trees and rooted trees

- Discussion of tasks in graph structures is impossible without introducing the notion ***tree***.
- By the **classic definition,** the graph $T(V,E)$ is a ***tree*** if it is connected and has no cycles.
- For the purposes of algorithmics more helpful is the notion ***rooted tree***. Two equivalent inductive definitions of rooted tree are given below.

Trees and rooted trees

Definition 1.

- (i) The graph $T(\{r\},\varnothing)$ is a ***rooted tree***. r is a ***root*** and a ***leaf*** of T;
- (ii) Let $T(V,E)$ be a rooted tree with root r and leaves $L = \{v_1, v_2,\ldots, v_k\}$. Let $v \in V$ and $w \notin V$;
- (iii) Then $T'(V' = V \cup \{w\}, E' = E \cup \{(v,w)\})$ is also a rooted tree. r is a ***root*** of T' and ***leaves*** of T' are $(L - \{v\}) \cup \{w\}$.

Trees and rooted trees

Definition 2.

- The graph $T(\{r\},\varnothing)$ is a rooted tree. r is a ***root*** and a ***leaf*** of T;
- (ii) Let $T_1(V_1,E_1),T_2(V_2,E_2),\ldots, T_k(V_k,E_k)$, be rooted trees with roots $r_1, r_2,\ldots, r_k$, and leaves $L_1, L_2,\ldots, L_k$, respectively. Let $r \notin L_1 \cup L_1 \cup \ldots \cup L_k$;
- (iii) Then $T'(V' = V_1 \cup V_2 \cup \ldots \cup V_k \cup \{r\}, E' = E_1 \cup E_2 \cup \ldots \cup E_k \cup \{(r,r_1), (r,r_2),\ldots,(r,r_k)\})$ is also a rooted tree. r is a ***root*** of T' and ***leaves*** of T' are $L_1 \cup L_2 \cup \ldots \cup L_k$. Rooted trees $T_1,T_2,\ldots,T_k$ are called ***sub-trees*** of T'.

Trees and rooted trees

- Rooted trees are undirected graphs. Anyway, Definition 1 is introducing an ***implicit direction*** on the edges of the rooted tree. That is way we could say that *v* is a ***parent*** of *w* and that *w* is a ***child*** of *v* (***Defintion 1***).
- Obviously, each rooted tree is a tree and each tree could be rebuild as rooted when we choose one of the vertices for root.

Trees and rooted trees

- If $G(V,E)$ is a graph and $T(V,E')$ is a (rooted) tree such that $E' \subseteq E$ than T is called ***spanning*** tree of G. The most natural way to check whether the graph G is connected is to try to build a spanning tree of G.
- If $c: E \to C$ we could define $c(T(V,E')) = \Sigma_{e \notin E'} c(e)$. Each spanning tree T of G with minimum (maximum) $c(T)$ is called ***minimal*** (***maximal***) ***spanning tree*** of G.

Presentation of graphs and trees

- Well known presentations of graphs are:
  - List of edges (or arcs)
  - The adjacency matrix
  - Lists of neighbors (or children)
  - Matrix of incidence, etc.
- Specific presentations of trees are:
  - List of parents
  - “Left child – right neighbor”, etc.

Presentation of graphs and trees

- Presentation of graph is crucial for the effectiveness of the algorithms
  - `for e ∈ E do{...}` – list of links
  - `for (v∈V'⊆ V)` – adjacency matrix

    `{ for w∈V''⊆ V`

    `{... if(v,w)∈E {...}}}`
  - `for v∈V`

    `{for w such that (v,w)∈E do{...}}`

    – lists of neighbors

Classification of tasks in graphs

- Two approaches for classification of tasks in graphs could be considered – this of the profiled textbooks for algorithms in graphs and this of the general textbooks in algorithms:
  - The profiled textbooks for algorithms in graphs use as a criteria some notion or property. The approach is based on the ***inner***, ***graph-theoretical logic***;
  - Examples: Christophides, N. (1975). *Graph Theory. An Algorithmic Approach*, Academic Pres.

    Gondran, M. and M. Minoux (1984). *Graphs and Algorithms*, John Wiley & Sons.

Classification of tasks in graphs

- The general textbooks in algorithms classify the tasks by the ***class of algorithms*** (or the ***algorithmic scheme***) that can solve a set of tasks:
  - "Divide and Conquer";
  - Dynamic programming;
  - Exhaustive search;
  - Greedy;
  - Algorithms in graphs, etc.;
- So, the algorithms in graphs are specific category in general classification of algorithmic approaches.
- Example: Cormen, T. H., Ch. E. Leiserson and R. L. Rivest (1990). *Introduction to Algorithms*, Second Edition, The MIT Press.

Classification of tasks in graphs

- Such classification does not exclude the possibility an algorithm on graphs to be classified as:
- ❖ Dynamic programming – for example, the shortest path task;
- ❖ Greedy – for example, min/max spanning tree;
- ❖ Exhaustive search.

Classification of tasks in graphs

- Using the second approach in depth we propose the following categories of the algorithms in graphs (and so – of the tasks solved by these algorithms) :
  - "Bread-first" search;
  - "Depth-first" search;
  - Euler traversals;
  - Min/Max Spanning trees;
  - Relaxation approach (Dijkstra);
  - Exhaustive search in graphs;
  - Matching/Flows in Networks;
  - Games in graphs.

"Bread-first" search

- Algorithmic scheme "***Bread-first***" is the easiest for understanding and applying, but needs knowledge of the ADT ***queue***.
- Searching the graph "**in bread**" we can:
  - To check the ***accessibility*** of a vertex from another vertex in a graph;
  - To check the ***connectivity*** of a graph;
  - To find the ***connected components*** of a graph;
  - To find the ***shortest path*** from each vertex of a graph to each other vertex – for **graphs without cost function** on edges, etc.

"Depth-first" search

- Algorithmic scheme "***Depth-first***" is more difficult for understanding and applying. But the knowledge of the ADT ***stack*** is escaped by using recursion. With DF we can solve the mentioned above tasks ***accessibility, connectivity, connected components***.
- DF can't solve ***shortest path*** but DF is a first step for ***topological sorting, articulation vertices*** and ***links***, ***strongly connected components***, etc.

Euler traversal

- The algorithm for finding an ***Euler cycle*** seems to be a specific algorithm, but it could be considered as an algorithmic scheme because small modification of it can solve other tasks as:
  - ***Euler path*** in a multi-graph;
  - ***Covering of edges*** of a multi-graph with paths that not intersect in edges, etc.

Min/Max Spanning Tree

- Building ***min/max spanning tree*** is specific, easy to understand, optimization tasks in graphs with cost function on the edges.
- The algorithms of Prim and Kruskal solve this task
- Some authors classify algorithms for building MST as "***greedy***" and this is reasonable. MST tasks have the mathematical structure – ***matroid*** – that is necessary and sufficient condition to be solved with the "greedy" scheme.

Relaxation approach

- ***Finding of the shortest path*** ("1 to 1", "1 to all" or "all to all) is the most important optimization tasks in graphs with cost function (on edges, on vertices or both).
- ***Relaxation approach*** of Dijkstra is a base of practically all existing algorithms. Many related optimization tasks (largest path, most reliable path, etc.) could be solved by the same approach. It is possible to classify it as "***greedy***" as well as "***dynamic programming***"

Exhaustive search in graphs

- Typical task in graphs that can be solved by ***exhaustive search*** is finding of the ***Hamilton traversal***.
- Speaking for "exhaustive search" in graphs, people usually have in mind the algorithmic scheme "***backtracking***".
- But exhaustive search in graphs are also the algorithms that generate ***(all) permutations, combinations, variations, partitions,*** and so on, of the vertices, of the edges or both.

Matching, flows and games

- These are relatively ***difficult categories*** of tasks. Understanding of algorithms that can solve them suppose **deep mathematical background**.
- Beside mentioned categories of tasks we would like to mention that **the graphs and especially the trees are used as a technological instrument** in algorithms from other domains.

Graphs in Bulgarian Olympiads

Category of tasks	Number of tasks	Age group		
		C	B	A
Bred-first search*	15	6	3	6
Depth first search	15	2	4	9
Euler traversals	3	1		2
Minimum spanning tree	2			2
Shortest path	25	2	7	16
Matching and flows in networks	6		2	4
Games in graph (Nim)	2		1	1
Exhaustive search	15	1	2	12
Difficult to classify	2			2

Unclassified tasks

- ***Task 7*** A set V of vertices and the length of the shortest path $d(v,w)$ for each $v,w \in V$ are given. Find a graph $G(V,E)$ with minimal number of edges and cost function on the edges, in which the length of the shortest path for each couple of vertices $v,w \in V$ is equal to the given $d(v,w)$.
- Even if the terminology of the shortest path task is involved this tasks is not solvable by the relaxation approach.

Unclassified tasks

- ***Task 8*** A rooted tree $T(V,E)$ is given. For each vertex v, the vertices that belong to the unique path from v to the root of T are called ***ancestors*** of v. For given vertices u and v find their lowest common ancestor (LCA), i.e. such vertex w that is ancestor of u and v, and there is no common ancestor w' of u and v, such that w is ancestor of w'.
- LCA task needs an approach that is not among the mentioned above.

Conclusions

- Graph structures are **an important origin of tasks** for olympiads in informatics.
- Most of the notions and concepts are understandable by relatively young students. In Bulgaria, students aged 14-15 years solve regularly such task in national contests.
- So teaching of graph concepts and algorithms really could starts **at the age of 12-13 years**.

Conclusions

- Proposed classification of tasks in graphs is **one of many possible**. Classification based on other principals are also possible.
- But some **classification of tasks in graphs is necessary** for each team of teachers that is coaching contestants in programming.
- Classification of tasks could help to **organize better** both the training process and the contests.

Conclusions

- Proposed classification of tasks in graphs is **one of many possible**. Classification based on other principals are also possible.
- But some **classification of tasks in graphs is necessary** for each team of teachers that is coaching contestants in programming.
- Classification of tasks could help to **organize better** both the training process and the contests.

Conclusions

- BFS and DFS are very good possibility to introduce the contestant of age 14-15 years in the topic.
- At the age of 16-17 years shortest path tasks could be included.
- At the age of 18-19, beside algorithmically hard tasks, solvable by different kind of exhaustive search, some specific topics, like Matching in bipartite graphs, Flows in networks and Games of type Nim, also could appear in the contests.

Thanks for your attention!