Association of Urals State University Alumni

USU Mathematics and Mechanics Department

Student's Scientific Computer Society

presents

international computer programming competition

"Karpinsk Tram – 96"

dedicated to second anniversary since abolishment of tram route in the town of Karpinsk.

Graph algorithms lovers will have a chance to show their worth, have fun, and possibly get a PRIZE by taking part in the programming contest. The problem is not hard to understand, so anyone can try solving it – from school students to professors.

The organizers sincerely appreciate help provided by Association of Urals State University Alumni for becoming general sponsor of the contest, and Ural Relcom company for providing communication resources and informational support.

One track tram route.

For a long time there was a unique system of tram routes in Karpinsk. What made it unique was the only pair of rails so that trams were following it in both directions. To prevent accidents, at several points of the route double track section was established, so that two trams going in opposite directions could safely pass each other. If one of the trams would come to the passing track first, he would have to wait there until the one going to the opposite direction arrives.

The authors found it interesting to create an algorithm for building schedules for such transportation schemes. The generalized problem, formulated in terms of graph theory, is offered to reader's attention.

The problem.

Consider a non-oriented connected graph without loops and multiple edges. For convenience let us introduce some terms. A vertex of degree one we will call *terminal*. A vertex of degree two will be either *stage* or *passing track*. The difference is that on a passing track two trams going in the opposite directions can pass each other; on a station they cannot. A vertex of degree three or more is called *junction*. A vertex that is either junction, stage or passing track is called *station*.

Later we will suppose that the given graph contains not lass than two terminals.

Route is a sequence of vertexes satisfying the following *transport conditions*:

- any two consequent route vertexes are connected by an edge of the graph;
- the first vertex of the route is a terminal and is the same with the last one; exactly one of the rest vertexes of the route is also a terminal, and these two terminals are not the same[*].
- if $V_1 \ldots V_n$ is a route, then $\forall i \in \{2,\ldots,n-1\}$ ($\deg V_i \neq 1 \Rightarrow V_{i-1} \neq V_{i+1}$); The meaning of this restriction is that trams can turn around only on terminals. (Here notation deg V is used to denote degree of vertex V).

Route Schedule is a function S(V), that maps any junction and passing track into a non-negative integer. Note, that if some vertex is used several times in the route, it can every time map to a different integer. The value of the function is equal to the number of trams going to the opposite direction, that our tram has to pass before leaving the particular vertex. A tram is considered to be going to the opposite direction if it comes from the edge, where the yielding tram is required go. For our convenience let us suppose S(V) function to be equal zero on all stages and terminals.

Each tram has its own route. At the beginning all trams are at the first vertexes of the corresponding routes. If at some moment a tram is at vertex V, then the edge by which it came to V (if V is a stage, then both edges are incident to V) is (are) *blocked* for all other trams at this moment.

Let us describe movement of a fixed tram T. Let the tram came to vertex V and U is the next vertex in the route. Then

1. The tram is waiting until S(V) other trams pass edge UV. This includes the tram (if there was one) which followed edge UV and was in V vertex when tram T came there.
2. Let S(V) trams have passed (or S(V)=0). If edge VU is blocked, then tram T is waiting until it is free.
3. If the tram is not waiting, it follows to vertex U.

Time is measured in steps. On each step all trams that are not waiting make a simultaneous move.

Transportation schedule is a set of route schedules, that allows to get from any station to any other station (maybe changing trams on some stations). Any two starting points of different routes in transportation schedule must not be the same. At the beginning (step zero) every tram is at the staring vertex of the corresponding route.

Working time for a transportation schedule is a minimum number of steps required so that each participating tram finishes not less than three full routes[*] . If some transportation schedule does not let all trams to complete three full routes, working time for this schedule is considered to be infinity.

---

[*] The meaning of this is that route is a path from the first terminal to the second one and back, and the paths there and back are not necessarily the same.

[*] The first route of a tram may differ from the rest ones because of different positions of other trams at the starting moment. That's why second and third routes may take longer (or shorter) than the first one.

All vertexes of degree two in original graph are stages. Operation of *adding a passing route* is the procedure of attaching to the graph a new vertex W and substituting an existing edge UV with a pair of edges UW and WV. After this operation the new vertex W is a passing route. Let us call a graph *valid* if it either is the same with the original graph, or can be derived from it by adding a finite number of passing routes.

The problem is to build a transportation schedule that has minimal working time among all valid graphs.

It has been proven (see appendix) that for any original graph with M terminals it is possible to build a valid graph, which has a transportation schedule with finite working time and which can be derived from the original graph by adding not more than M-1 passing routes. Thus, the problem stated above always has a solution.

Contest conditions.

The contest is open to everyone. To participate one should write a program that for a given graph will built some valid graph and transportation schedule for it. The program must be submitted to contest organization committee before the due date stated below. Every participant is allowed to submit no more than two programs for the competition.

Several test graph will be offered to participating programs. On each of them working time of the transportation schedule built by programs will be measured. If a program is unable to generate transportation schedule for a given graph, then the corresponding working time is assumed to be equal to doubled maximum working time for this graph among all participating programs. If for some graph no program is able to generate transportation schedule, than for all programs the working time for this graph is considered to be zero. The number of vertexes of the test graph will not be more than twenty.

A program, which will have minimal sum of working times for all test graphs, will be declared the winner. The author of winning program will receive the prize. For programs ranked first, second and third, the authors will receive contest diplomas.

For those who wants to compete with computers there is an additional contest. On this contest the same test graph will be offered to human beings. Participants will need to submit transportation schedules to the organization committee, one schedule for each given graph. The solutions will be scored in the same way as the program-generated ones. The winners of this contest will be awarded with diplomas.

To participate in the additional contest one need to submit to organization committee an application in the following form:

name	
city and country	
school or organization	
electronic mail address	

regular address (for diploma)	

Besides of the above, it is a good idea to provide any extra information about yourself that you consider appropriate. The test graphs will be sent to participants by email ONLY. In addition they will be published on WWW at "*http://www.mplik.ru/~sg/tramk*" and on the bulletin board near USU Mathematics and Mechanics department dean's office.

Program requirements

1. The program must be submitted in the form of C or C++ source code all in a single file.
2. The program must be portable, i.e. it must not depend on such parameters as length in bits of int or char, CPU command set, etc.
3. The file must start with a comment, containing the contest application in the form as shown above for additional contest.
4. The program will be compiled with gnu gcc/g++ under UNIX. Program that will not compile or link will not take part in the contest.
5. The program must take input from standard input stream (stdin) and write the results to standard output stream (stdout). Using of any other files, including temporary ones, is prohibited.
6. The time that program will work is limited. The maximum allowed time of calculations will be supplied to program along with input data. The program may check the time spent using standard library routines (e.g. gmtime()). After the maximum allowed time has passed, the program will be forcibly stopped. In this case the program will not be run again.
7. The program that does not provide upon completion a transportation schedule in the format described below, or finishes with run-time error is considered to be unable to generate transportation schedule for the given graph.
8. The number of passing tracks added by the program must not be greater that the doubled number of vertexes in the given graph.
9. If working time of the transportation schedule built by the program is greater than $10*N^2$, where N is a number of vertexes of the given graph, then the program is considered to be unable to generate transportation schedule for the given graph.

Input data format

maximum allowed calculation time (in minutes)

the number of vertexes in the graph

Graph adjacency list

For each vertex of the graph in adjacency list there is a line:

```
vertex_name: vertex_name vertex_name … vertex_name
```

where the left one is the current vertex, and after the colon there is the list of vertexes, adjacent with it. There are no any other lines in adjacency list .

The name of the vertex does not contain spaces, tabulation, punctuation marks and starts with a letter.

Example. For a graph, consisting of two terminals, connected with an edge, the input file fill be the following:

```
15
2
V1: V2
V2: V1
```

Output data format

The number of passing tracks added

The list of passing tracks added

The list of routes

The list of passing tracks added consists of pairs:

```
vertex_name vertex_name
```

where the named vertexes define the edge where the new passing track should be placed. The newly added passing tracks are automatically given the names R1, R2, … . If two passing tracks must be added to the same edge, the corresponding part of the list must look like this:

```
vertex_name vertex_name
vertex_name passing_track_name
```

i.e. the second passing track is placed onto the new edge created after adding the first one.

The list of routes looks like this:

```
<empty line>
terminal_name: 0
vertex_name: number
…
vertex_name: number
terminal_name: 0
<empty line>
terminal_name: 0
… and so on.
```

The strings between empty lines define a route, the numbers after colons define the schedule function for the route. For stages and terminals the numbers must be zeros. The list ends with two empty lines.

Example. For the input file above the resulting output may be the following:

```
2
V1 V2
V2 R1

V1: 0
R1: 0
R2: 0
V2: 0
R2: 0
R1: 0
V1: 0
```

Due dates and coordinates

Programs must be received by the organization committee not later than May 19, 1996. Application for participating in the additional contest must also be received before this date. Test graph will be published on May 20th. Works (transportation schedules) for additional contest must be submitted till May 26th. The contests results will be published as soon as they are ready.

The following way of delivering works and applications to the organization committee exist:

- send by email on address *games@www.mplik.ru* with subject line "*Karpinsk Tram-96*"
- send by regular mail on a 3.5" 1.44M diskette in IBM PC format on address: *"Karpinsk Tram-96", Mathematics and Mechanics department, Dean's office, Urals University, 51 Lenina Ave, Ekaterinburg, 620083, Russia.* Instead of sending by mail, the envelope may be brought to the dean's office in person. In any case, the diskette will not be returned to the participant.
- deliver in person to the organization committee (math and mechanics USU dept., group MGMT-5). In this case it will be possible to get the diskettes back by requesting them in person during a week after the contest has ended.

All possible questions and comments should be sent to the above addresses. Answers to frequently asked questions and other up-to-date information will be available at "*http://www.mplik.ru/~sg/tramk*".

Contest organization committee

- Sergey Gershtein, *sg@mplik.ru*. Urals University master's program student.

- Eugeny Shtykov, *eug@kontur.e-burg.su*. Urals University master's program student.

Appendix

Theorem. For any given graph with M>1 terminals it is possible to built a valid graph for which has a transportation schedule with finite working time and which can be derived from the original graph by adding not more than M-1 passing tracks.

Let us preface the proof with a auxiliary definition and lemma.

Definition. Let us denote route beginning with terminal V and going through terminal U with M(U,V). We will use term *cyclic stations covering* of graph G with set of terminals K = $\{V_1, \ldots, V_n\}$ for collection of routes $M(V_1,V_2)$, $M(V_2,V_3)$, … , $M(V_n,V_1)$, which covers all the stations.

Lemma. Cyclic stations covering exists for any given graph.

Proof of the lemma. Let us make our proof by induction on the number of stages in the graph. The base of induction is obvious: for zero stages in the graph any collection of routes $M(V_1,V_2)$, $M(V_2,V_3)$, … , $M(V_n,V_1)$ will be cyclic stations covering. Existence of such collection follows from graph connectivity.

Consider a graph with P stages. According to induction hypothesis we can build cyclic stations covering for arbitrary chosen P-1 stages. Denote the remaining stage with W, and the incident edges with UW and VW. If W belongs to some route, the covering built is the sought one. Suppose it does not.

Let in the graph there is a cycle $W_0=W$, W_1, … , $W_t=W$. We can find a path U_1, … ,U_n, that connects some vertex of the cycle with a junction of some route. Let k is the greatest number, such that vertex $U_k=W_s$ belongs to our cycle, and m is the lowest number, such that $m \geq k$ and vertex U_m belongs to an existing route. Let this route is V_i, … , U_m, … , V_i. We will change it into the following: V_i, … , U_m, U_{m-1}, … , $U_k=W_s$, W_{s+1}, … , W_{t-1}, W, W_1, … , $W_s=U_k$, U_{k+1}, … , U_{m-1}, U_m, … , Vi. The resulting collection of routes is the sought covering.

Let W is a cutpoint of the graph. It is easy to see that in this case one of the two connected components created when removing vertex W, contains all the terminals of the original graph. Obviously, if it were not true, there would be no way to get from one component to the second one. Degree of each vertex in the component without terminals is not less than two, and hence there is a cycle there. Using the cycle, any route can be modified as shown above, and the modified route will necessarily pass vertex W. The lemma is proven.

Proof of the theorem. Let V_1, … , V_n are all terminals of the original graph. According to the above lemma there exists a cyclic stations covering $M(V_1,V_2)$, $M(V_2,V_3)$, … , $M(V_n,V_1)$. We will add one passing track to edges that are incident to vertexes V_2, … , V_n, and include these passing tracks into corresponding points of the covering routes. The function of route schedule will be equal to zero everywhere besides the newly added passing tracks, on which it will be equal to one.

The transportation schedule obtained solves our problem because according to it all trams take turns moving along their routes and do not interfere with each other. The theorem is proven.