

Graph Theory

Computer Science 66-4260/65-4150; Spring, 2005
Computer Project: Graceful labeling of trees

The goal of the project is to experimentally investigate the problem of graceful labeling of trees; see pages 211- 213 of the textbook. You will develop three programs and conduct a series of experiments in order to establish that the conjecture about graceful labeling of trees holds true at least for the trees from your experiments.

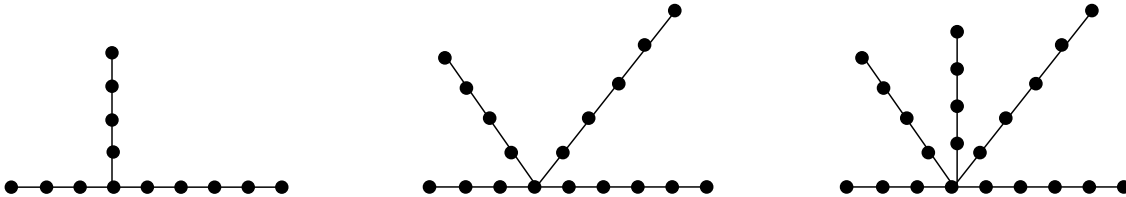
The project must be done in teams of four, or maybe five students. Let me know as soon as you form a team or talk to me immediately so that I can help you to find a team.

The project is due April 28, but before that, by March 14, every team must inform me about how you started the work: who are the members of the team, and how do you plan to distribute the work.

Contact me any time you are having questions.

The three programs to be created are:

1. GENERATOR-S: this program generates a series of “special” trees; what is special should be clarified with me, individually, individually, by every team in consultation with me. The figure below presents examples from some series.



2. GENERATOR-N: this program generates all trees with up to a given number n vertices (n is the input); and
3. LABEL: this program finds a graceful labeling for a given tree (*may or may not be produced by GENERATOR*).

One of the main factors that will be used to evaluate the project is the value of n for which programs GENERATOR-N and LABEL can be run till completion in a “reasonable” time (say at most a few hours on the CS-computers). Such a “reasonable” n is not expected to be very large, maybe close to 10, but it shouldn’t be too small either; $n \geq 6$ looks like a must.

The creation of the program is the first part of the project. The second part: experiments. You will run programs GENERATOR-N and LABEL, in a package, in order to answer several questions:

- How much execution time does it take, depending on n , to find a graceful labeling for all trees with n vertices ($n = 2, 3, 4, 5, \dots$)?

- Does finding a graceful labeling take approximately the same time for each tree of a given order n , or there are some “easy” and “difficult” trees?
- Maybe some others, *that you will ask*.

Combining your GENERATOR-S and LABEL, you will attempt to discover a *pattern* in the labeling of your series of special trees.

The third part of the project is your Report. It must be written using a type-setting system and submitted to me electronically. The Report must include your names; pseudocodes of your algorithms; programs; the results of the experiments; a discussions of those results.

The idea for GENERATOR-S.

The program must create all trees of a given order n . On the other hand, it should not create the same tree repeatedly: isomorphic copies of any tree should not be submitted to LABEL. The basic idea of generating the set \mathcal{S}_n of all trees of a given order n is to start with the unique tree of order $n = 2$, and for each values of n , use the set \mathcal{S}_n , to create \mathcal{S}_{n+1} . Your program may iterate over all trees in \mathcal{S} and for each of them, create new trees of order $n + 1$ by adding one vertex adjacent to a vertex of the tree. Clearly, even if \mathcal{S}_n does not contain isomorphic copies of the same tree, the process above may create some isomorphic trees. You can use the package NAUTY to check if the newly created tree is already in the set (and then not add it to the set). Using NAUTY is certainly a plus for your project.

The idea for LABEL.

Every labeling of the vertices of a tree is an assignment of integers $0, 1, 2, \dots, n - 1$ to the vertices of the tree. Thus, there are $n! = 1 \cdot 2 \cdot \dots \cdot n$ possible assignments. Only tiny portion of them are graceful. Your program may avoid trying all candidates and rejecting those that are not graceful. You may use *backtracking* for efficient search. For example, your program may start by selecting a vertex to be labeled with 0. For each choice x , the label $n - 1$ can only be used for vertices that are adjacent to x . This may substantially reduce the number of choices to consider: instead of mindless $n(n - 1)$ combinations to consider, (n choices for 0, and for each choice, $n - 1$ choices for $n - 1$), your program will consider just $2(n - 1)$ choices for placing 0 and $n - 1$ combined.

You can further reduce the number of combinations by employing nauty, which can identify “similar” vertices in a given tree.