LEDA

A Library of Efficient Data Types and Algorithms

Kurt Mehlhorn and Stefan Näher

Fachbereich Informatik, Universität des Saarlandes
D-6600 Saarbrücken, Federal Republic of Germany

Abstract

LEDA is a library of efficient data types and algorithms. At present, its strength is graph algorithms and the data structures related to them. The computational geometry part is evolving. The main features of the library are
1) a clear separation of specification and implementation
2) parameterized data types
3) a comfortable data type graph, and
4) its ease of use.

I. Introduction

There is no standard library of the data structures and algorithms of combinatorial computing. This is in sharp contrast to many other areas of computing. There are e.g. packages in statistics (SPSS), numerical analysis (LINPACK, EISPACK), symbolic computation (MACSYMA, SAC-2) and linear programming (MPSX).

In fact the situation is worse, since even within small groups, say the algorithms group at our home institution, software is frequently not shared. Rather, each researcher starts from scratch and e.g. develops his own version of a balanced tree. Of course, this continuous “reimplementation of the wheel” slows down progress, within research and even more so outside. This is due to the fact that outside research the investment for implementing an efficient solution is frequently not made, because it is doubtful whether the implementation can be reused, and therefore methods which are known to be less efficient are used instead. Thus the scientific discoveries migrate only slowly into practice.

One of the major differences between combinatorial computing and other areas of computing such as statistics, numerical analysis and linear programming is the usage of complex data types. Whilst the built-in types, such as integers, reals, vectors, matrices
and functions, usually suffice in the other areas, combinatorial computing heavily relies on types like stacks, queues, dictionaries, sequences, sorted sequences, priority queues, graphs, points, planes, . . . .

One year ago, we started a project (called LEDA for Library of Efficient Data types and Algorithms) to build a small, but growing library of data types and algorithms in a form which allows them to be used by non-experts. We hope that the system will narrow the gap between algorithms research, teaching, and implementation. In this paper we report on our difficulties and achievements. The main features of the library are:

1) A clear separation between (abstract) data types and the data structures used to implement them, cf. section II. This distinction is frequently not made in the combinatorial algorithms literature, but it is crucial for a library. Note that we stated above that each researcher implemented its own version of a balanced tree, i.e. a data structure, and not its own version of a dictionary, i.e. a data type. The data types currently available are stack, queue, list, set, dictionary, ordered sequence, priority queue, directed graph and undirected graph. The most difficult decision, which we faced, was the treatment of positions and pointers. For the efficiency of many data structures it is crucial that some operations on the data structure take positions in (= pointers into) the data structure as arguments. We have chosen an item concept to cast the notion of position into an abstract form, cf. section II.

2) Type parameters, cf. section II. Most of our data types have type parameters. For example, a dictionary has a key type K and an information type I and a specific dictionary type is obtained by setting, say, K to int and I to real.

3) A comfortable data type graph. It offers the standard iterations such as “for all nodes v of a graph G do” (written forall_nodes(v, G)) or “for all neighbors w of v do” (written forall_adj_nodes(w, v)), it allows to add and delete vertices and edges and it offers arrays and matrices indexed by nodes and edges,..., cf. section III for details. The data type graph permits to write programs for graph problems in a form close to the way the algorithms are usually presented in text books. Section VII contains a list of the algorithms which are currently in LEDA.

4) Ease of use: LEDA is written in C++ (cfront version 1.2.1). All data types and algorithms are precompiled modules which can be linked with application programs. All examples given in this paper show executable code.

This paper is organized as follows. In section II we discuss data types and data structures, in section III the data type graph and in section IV we discuss the interaction of graphs and data types. In section V we briefly discuss the internal structure of LEDA and in section VI we report about our experiences with designing, implementing and using LEDA.

The design of LEDA is joint work by the two authors, the implementation was mostly done by the second author. LEDA is available from the authors for a handling charge of DM 100.