

Library of **E**fficient **D**ata Types and **A**lgorithms

Vorstellung der Bibliothek und Vorführung eines
Beispiels

I. Was ist LEDA?

- Motivation**
- Die 4 Designziele**
- Anwendung/Vertrieb**

II. Ein Einblick

- Zahlendatentypen**
- Itemtypen**
- Initialisieren/Kopieren**
- Speicherverwaltung**

III. Anwendungsbeispiel

Was ist LEDA?

C ++ - Library für kombinatorische/geometrische Algorithmen

Leitmotiv:

LEDA soll die größten Errungenschaften aus der Algorithmenwelt enthalten und auch Laien zugänglich machen

Motivation

- Erhalt bereits implementierter Algorithmen
- Vermeidung von redundantem Code
- Spezifikationen oft nicht komplett/abstrakt genug

Die vier Designziele von LEDA

- Einfachheit
- Erweiterbarkeit
- Validität
- Effizienz

Vertrieb

Free

Research:

→ Team: 1,200 €

ObjectCode / Standort : 4,000

SourceCode / Standort : 6,000

Professional:

→ Single User: 2,500 €

ObjectCode / Standort : 7,000

SourceCode / Standort : 16,000

Für Windows, Linux und Solaris

LEDA

Ein Einblick

Die Bibliothek ist unterteilt in:

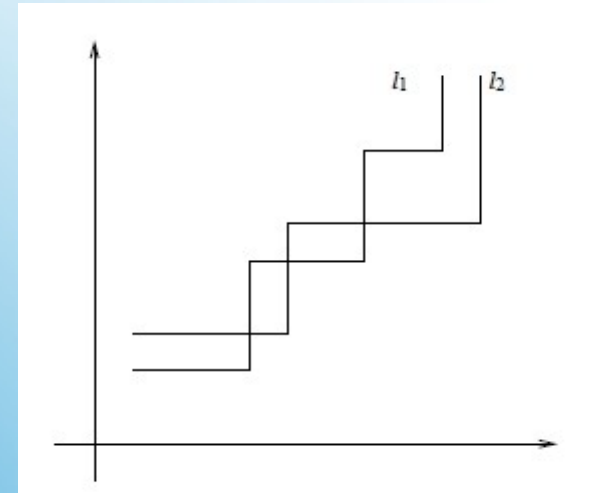
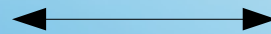
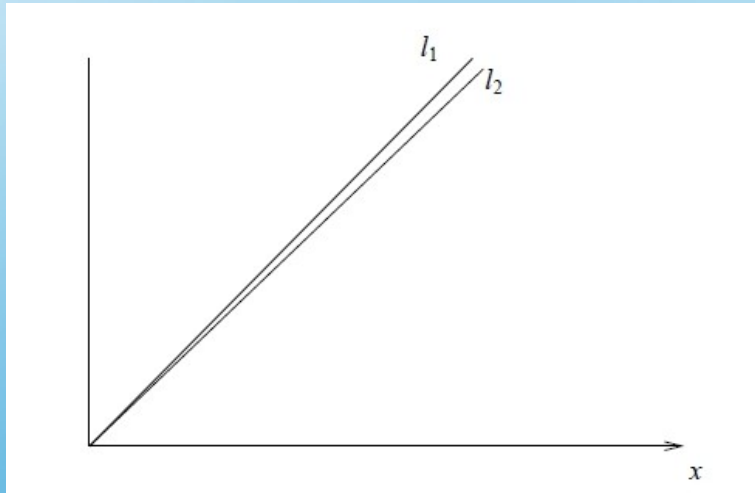
- Basistypen (Strings, Listen, Stacks, Felder, Partitionen, Bäume)
- Zahlen, Vektoren und Matrizen
- Wörterbücher und Queues
- Graphen
- Geometrie
- Grafik

Algorithmus + LEDA = Programm

Zahlendatentypen

- integer
- rational
- bigfloat
- real

Der real Datentyp



Itemtypen

- Abhängige Itemtypen
- Unabhängige Itemtypen

Itemtypen

Unabhängig	Abhängig
Punkte, Linien	DictionaryItems, QueueItems
point.xcoord()	D.inf(item)
Klassen mit Pointerattribut	Pointer
Nach der Erstellung unveränderlich	Können im Nachhinein noch geändert werden

Warum Itemtypen?

- Effizienz
- Sicherheit

Initialisieren/Kopieren

- Die Datentypen in LEDA haben eine Defaultinitialisierung
- Es sind auch Initialisierungen „by copy“ möglich

Initialisieren/Kopieren

- Primitiver Typ → Wert selbst
- Nicht-Item-basierter Typ → komponentenweises Kopieren
- Item-basierter Typ → Neue Items mit alten Werten

Objektlebensdauer

- Ein Objekt existiert solange sein Block durchlaufen wird
- Ausser es ist statisch oder anonym

Speicherverwaltung

- LEDA verwendet viele kleine Einheiten
- Gebrauch für edges, nodes, Itemtypes
- Auch vom Programmierer einsetzbar
- Laufzeiteffizienz
- Garbagecollection