Abstracting Positional Information in Data Structures: Locators and Positions in JDSL

M.T. Goodrich, M. Handy,
B. Hudson, R. Tamassia

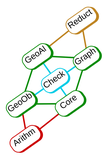
Locator

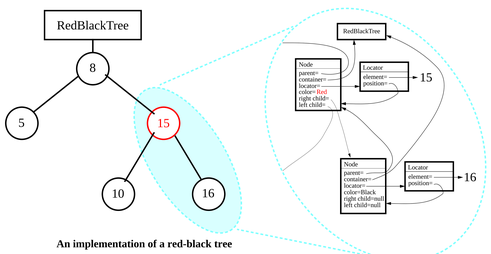
Useful in dictionaries, priority queues

- elements and positions rearranged to maintain an order; but may need to find element
- locator allows constant-time access to element
- only one search needed

Adds a level of indirection

- points back to in-memory structure of container
- allows user to ignore topology of container
- follows its element around; container forbidden to change the locator-to-element binding





An implementation of a red-black tree

The fields of each object in the highlighted region are indicated. The locators, to which the user has access, have pointers into the nodes of the data structure; these nodes then have pointers to their children and parent, and store the color of the node.

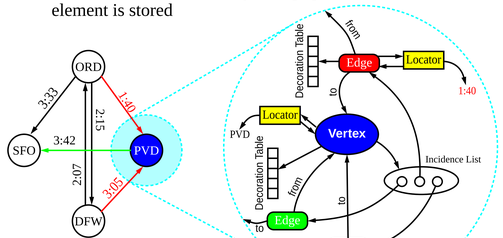
Position

Useful in sequences, trees, graphs

- connections between nodes are of primary interest
- no implied ordering or constraint on the elements
- mainly used in iterating over a data structure

Abstracts the concept of node or index

- alternative to LEDA's *items* or STL's *iterators*
- avoids indirection implied by Locators
- either user or container may change position at which an element is stored



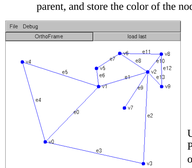
An implementation of a graph

This figure describes a possible implementation of the Graph interface. The Vertex has an incidence list (a Sequence of some kind), which holds a list of edges. It also has a table of decorations associated with the vertex, and a locator. The edge is similar, but stores exactly two vertices instead of a list of edges.

Decorable

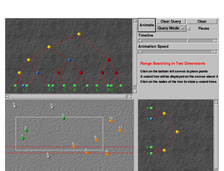
Allow attaching **attributes** to positions

- row-based access
- useful in graph algorithms, visualization
  - weight in Dijkstra, Prim; capacity in flow networks
  - graphical representation in GUIs



Graph Editor

Used to generate Embedded-PlanarGraphs. Decorations on the edges and vertices store visual elements.



Range Searching

This applet, written using JDSL, illustrates the data structures used in solving the 2-d range searching problem. The timeline allows stepping through the run of the algorithm at the user's pace, or to run through at a set speed.

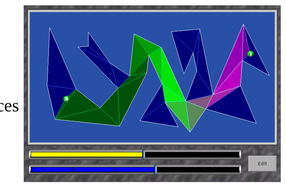
JDSL: Library of Data Structures in Java

Goals

- allowing rapid prototyping
- implementing complex algorithms in computational geometry
- teaching basic data structures and algorithms

Power and Flexibility

- rich interface hierarchy
  - data structures
  - algorithms
  - geometric primitives
- full-featured container interfaces
- universal locators
  - locator can be moved from any container to any other (even of different type)
  - feature of containers supplied with JDSL
  - can implement containers without this feature



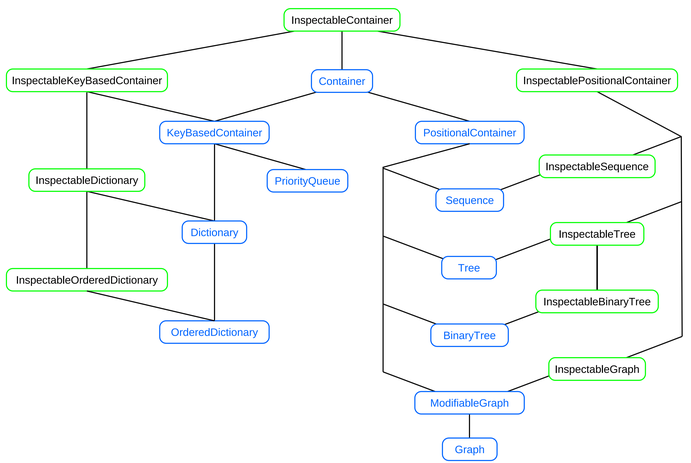
Shortest Path Inside a Polygon

Safety

- takes full advantage of the safety of Java
- behavior always defined
- exceptions thrown on invalid input

Optimizations

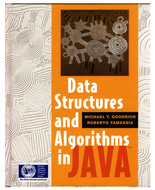
- Goal: Pay only for what you use
- On-Demand Locators
  - allocate Locators only when needed
- Enumeration Caching
  - compute the snapshot once; *cache* the result
  - successive calls return very quickly (constant time)
- Undecorated Positions
  - in KeyBasedContainers, user has no access to internal Positions
  - avoid overhead of Decorable

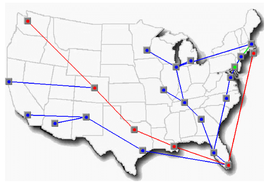


Teaching with JDSL

JDSL-Teach

- simplified version with focus on design principles
- used at Brown and Hopkins by over 300 students in CS2 courses (data structures & algorithms)
- facilitates implementation of advanced algorithms





Dijkstra's Algorithm

For the final project in CS 16 (the CS2-level class at Brown), the students implemented a significant subset of the JDSL-Teach Graph interface, and used it in Dijkstra's shortest path algorithm.

Dijkstra's algorithm written in JDSL

```
public void dijkstra() {
    while ( ! pq.isEmpty() ) {
        Vertex v = (Vertex) pq.removeMin();
        if (v==n) return;

        int vdist = distance(v);
        Enumeration outedges = graph.outincidentEdges (v);
        while (outedges.hasMoreElements()) {
            Edge e = (Edge) outedges.nextElement();
            Vertex dest = graph.destination (e);

            int destdist = vdist + weight(e);
            if ( destdist < distance(dest) ) { // relax
                pq.replaceKey ( (Locator) dest.get(locator), new Integer (destdist) );
                dest.set (incoming, e);
            }
        } // while there are outgoing edges
    } // while there are vertices in pq
} // dijkstra()

private int weight(Edge e) {
    return ((Integer)e.get(weight)).intValue();
} // distance()
```

Priority queue stores vertices of the graph. (Vertex is a subinterface of Position.)

replaceKey(.) takes a locator for a vertex already in the pq. That locator is stored as a label of the vertex.

Using the Decorable attribute mechanism to get and set labels on vertices.

References

[1] R. Baker, M. Boilen, M. T. Goodrich, R. Tamassia, and B. A. Stibel. Testers and visualizers for teaching data structures. Manuscript, 1998.

[2] N. Gelfand, M. T. Goodrich, and R. Tamassia. Teaching data structure design patterns. In *Proc. SIGCSE*, 1997.

[3] N. Gelfand and R. Tamassia. Algorithmic patterns for graph drawing. In *Proc. Graph Drawing '98*. Springer-Verlag, to appear.

[4] M. T. Goodrich, M. Handy, B. Hudson, and R. Tamassia. Accessing the internal organization of data structures in the JDSL library. In *Proc. Workshop on Algorithm Engineering and Experimentation (ALENEX'99)*. Springer-Verlag, to appear.

[5] M. T. Goodrich and J. G. Kloss II. Tiered vector: An efficient dynamic array for JDSL. In *DIMACS Technical Notes*, 1998.

[6] M. T. Goodrich and R. Tamassia. *Data Structures and Algorithms in Java*. Wiley, New York, NY, 1998.

[7] R. Tamassia, L. Vismara, and J. E. Baker. A case study in algorithm engineering for geometric computing. *Proc. Workshop on Algorithm Engineering*, pages 136–145, 1997.

Applications

[1] B. Hudson. *Graph Editor*

[2] N. Gelfand. *Range searching in two dimensions*

[3] J. Beall. *Shortest path between two points in a polygon*

[4] CS 16 Teaching Staff. *Flight*