Reference Manual for Object-Oriented Interval Tree Data Structure

Vikas Kapoor [*] Alexander Wolff [*]

Revision: 1.1, Date: 1998/01/22

[*]Institut für Informatik, Fachbereich Mathematik und Informatik, Freie Universität Berlin, Takustraße 9, 14195 Berlin, Germany, {kapoor | awolff}@inf.fu-berlin.de

Introduction

The interval tree stores intervals in such a way that those intersecting a query interval can be reported efficiently. One of the applications of the interval tree is the rectangle intersection problem, i.e. given n axis-parallel rectangles in the plane, report for each rectangle all rectangles which intersect this rectangle. An $\mathcal{O}(k + n \log n)$ algorithm for this problem can be sketched as follows: initialize the interval tree with the intervals (on the y-axis) to which the left sides of the rectangles correspond. Then sweep horizontally across the plane. When the sweep line hits a rectangle, query with the corresponding interval and insert it into the interval tree. An interval is deleted when the sweep line leaves the corresponding rectangle.

Here we implement a static version which must be initialized with the endpoints of all intervals it is supposed to contain afterwards. The preprocessing, namely sorting the points and building up an empty balanced tree, takes $\mathcal{O}(n \log n)$ time, where n is the total number of intervals. Inserting intervals can then be done in $\mathcal{O}(\log n)$ time, deleting in constant time, while a query needs $\mathcal{O}(k + \log n)$ steps, where k is the number of intervals reported. Since every interval is stored once in the interval tree, only linear storage is needed.

Chapter 1

Interval Tree

Definition

Class *OOIntervalTree<CoordType, InfoType>* stores intervals in such a way that those intersecting a query interval can be reported efficiently.

#include <oo_interval_tree.h>

Template Parameters

The class *OOIntervalTree<CoordType, InfoType>* is templated with the type of the endpoints of the intervals and the information associated with each interval.

Types

struct Interval<T> { T left, right;};

Type of the intervals to be stored in the class *OOIntervalTree<CoordType, InfoType>*.

OOIntervalTree<CoordType, InfoType>:: CoordType value_type;

Type of the endpoints of the intervals stored in the class *OOIntervalTree<CoordType, InfoType>*.

OOIntervalTree<CoordType, InfoType>:: list<Interval<CoordType> > Interval_ls;

Type of the list used to store the input intervals.

OOIntervalTree<CoordType, InfoType>:: Interval_ls::iterator Interval_ls_it;

Type of the iterator used to traverse the input intervals.

Creation

OOIntervalTree<CoordType, InfoType> interval_tree(Interval_ls& intervals);

creates the object *interval_tree* of type *OOIntervalTree<CoordType, InfoType>*.

Modifiers

Interval_position

interval_tree.insert(Interval_ls_it interval)

inserts the given interval in the *interval_tree*.

bool *interval_tree.erase(Interval_position& interval)*

deletes the given interval from *interval_tree*.

Access Functions

void *interval_tree.report(Interval_ls_it interval, list<InfoType>& intersections)*

returns the information associated with the intervals intersecting the given interval in *intersections*.

Chapter 2

Example

The following example illustrates the use of interval tree.

```
#include <iostream.h>
#include <list.h>
#include <vector.h>
#include <oo_interval_tree.h>

/* input operator for an interval: */
istream &operator>> (istream &in, Interval<double>& interval)
{
    in >> interval.left >> interval.right;
    return in;
}

/* output operator for an interval: */
ostream &operator<< (ostream &out, Interval<double> const &interval)
{
    out << "(" << interval.left << ", " << interval.right << ")";
    return out;
}

main()
{
    cout << endl << "         Testing the interval tree" << endl << endl;

    /* type definitions */
    typedef OOIntervalTree<double, int> Interval_tree;
    typedef typename Interval_tree::Interval_ls Interval_ls;
    typedef typename Interval_tree::Interval_ls_it Interval_ls_it;
    typedef typename Interval_tree::Interval_position ID;

    /* read the input intervals */
    Interval_ls interval_ls;
    Interval<double> interval;
    cout << "Enter the endpoints of the intervals: " << endl <<
    "(enter an interval of length zero to mark the end of the input" <<
    " intervals)"<< endl;
```

```
    vector<Interval_ls_it> positions;
    while (cin >> interval && (interval.left != interval.right))
    positions.push_back(interval_ls.insert(interval_ls.end(), interval));

    /* output the read intervals */
    cout << "  Interval No.    " << "Interval" << "   Info" << endl;
    int i = 0;
    Interval_ls_it intervals_end = interval_ls.end();
    for (Interval_ls_it intervals_beg = interval_ls.begin();
     intervals_beg != intervals_end; intervals_beg++)
    cout << "      " << i++ << "           " << *intervals_beg << endl;
    cout << endl;

    /* declare an object of type OOIntervalTree */
    Interval_tree interval_tree(interval_ls);
    ID *del_info = new ID[interval_ls.size()];

    /* update on and report from the interval tree */
    cout << "i-insert, d-delete, r-report and number of interval:" << endl;
    char c;
    while (cin >> c)
    {
    cin >> i;
    switch (c)
    {
        case 'i':
        {
            int info;
            cout << "inserting" << endl;
            cin >> info;
            del_info[i] = interval_tree.insert(positions[i], info);
        }
        break;
        case 'd':
        {
            interval_tree.erase(del_info[i]);
        }
        break;
        case 'r':
        {
            list<int> intersections;
            interval_tree.report(positions[i], intersections);
            list<int>::iterator end = intersections.end();
            list<int>::iterator begin = intersections.begin();
            for(; !(begin == end); ++begin)
            cout << *begin << endl;
        }
        break;
    }
    }
    delete[] del_info;
}
```