Interactive KD-Tree Implementation

Gautam Kumar
Vitaly Sergeyev
May 6, 2014

1 ABSTRACT

A KD-Tree is a popular binary-tree datastructure used in many applications such as range-search and nearest-neighbor search. The algorithm to build a kd-tree partitions k dimension input space based on the median points in the spacial frame. This type of data-structure makes searching for points in space an aver- age O(logn) algorithm. While a query with an axis-parallel rectangle in a KD-Tree storing n points can be performed in O($(n)^{1-\frac{1}{d}}$ + k) time, where k is the number of reported points and d is dimension.

2 PROJECT OVERVIEW

We have implemented an interactive 2D KD-Tree application. The user can click a set of points in a 2D input space or program can read the point set from the input file, after which the application visualize the kd-tree partitions and the tree itself. The user can also draw a bounding box on the screen to capture points in the plane or program can randomly draw the bounding box. Performance time is a metric of input size as well time. We have also implemented methods to generate a large, uniformly distributed set of points for input to the KD-Tree implementation. On top of that, we have analyzed and build a plot for analyzing the performance of the KD-Tree range search vs the brute force approach. We have the report on the N values that show a significant improvement in performance over the brute-force approach.

3 Implementation Details

We have mainly used following classes.

1. **Point:** It represents a two dimensional point in the 2-D space. It has x and y co-ordinates as attributes. It also consist the methods to do basic operations on point set. In our implementation a point object requires 16 bytes of memory. It has been implemented in **Point2D.java**.

2. **RectHV:** It represents a rectangle in 2-D space. A rectangle has been represented by lower leftmost point and upper rightmost point. It also has method to check whether two rectangles intersect each other or not. In our implementation a RectHV object requires 32 bytes of memory. It has been implemented in **RectHV.java**.

3. **KdTree:** It represents the KD-Tree in 2-D space. Node structure of KD-Tree consists of point associated with that node, rectangle represented by that node, and left, right children of that node. It also provides methods for various operations (insert, search, draw) on KD-Tree. In our implementation a Node object requires 64 bytes of memory. It has been implemented in **KdTree.java**.

4. **KdTreeVisualizer:** It provides the graphical view of KD-Tree as well as execution of building of KD-Tree. It has been implemented in **KdTreeVisualizer.java**.

5. **RangeSearchVisualizer:** It provides the graphical view of range search execution on point set. It uses KD-Tree search as well brute force search method. It has been implemented in **RangeSearchVisualizer.java**.

We have implemented following algorithms.

- BUILDKDTREE(P, depth)

  Data: A set of points P and the current depth *depth*
  Result: The root of a kd-tree storing P
  initialization;
  if *P contains only one point* **then**
  return a leaf storing this point;
  else
  if *depth is even* **then**
  Split P into two subsets with a vertical line L through the median x-coordinate of the points in P. Let P1 be the set of points to the left of L or on , and let P2 be the set of points to the right of L.;
  else
  Split P into two subsets with a horizontal line L through the median y-coordinate of the points in P. Let P1 be the set of points below or on L, and let P2 be the set of points above L.;
  end
  end
  $v_{left} \leftarrow$ BUILDKDTREE(P1, depth+1);
  $v_{right} \leftarrow$ BUILDKDTREE(P2, depth+1);
  Create a node v storing L, make v_{left} the left child of v, and make v_{right} the right child of v;
  return v;

  Analysis: A kd-tree for a set of n points uses O(n) storage and can be constructed in O(n log n) time.

- SEARCHKDTREE(v, R)

  Data: The root of (a subtree of) a kd-tree, and a range R.
  Result: All points at leaves below $\hat{I}_i$ that lie in the range.
  initialization;
  if *if v is leaf* **then**
  | Report the point stored at v if it lies in R.;
  else
  | **if** *region($lc(v)$) is fully contained in R* **then**
  | | REPORTSUBTREE(lc($\hat{I}_i$));
  | **else**
  | | **if** *region($lc(v)$) intersects R* **then**
  | | | SEARCHKDTREE($lc(v)$, R);
  | | **else**
  | | **end**
  | **end**
  | **if** *region($rc(v)$) is fully contained in R* **then**
  | | REPORTSUBTREE(rc($\hat{I}_i$));
  | **else**
  | | **if** *region($rc(v)$) intersects R* **then**
  | | | SEARCHKDTREE($rc(v)$, R);
  | | **else**
  | | **end**
  | **end**
  end

Analysis: A query with an axis-parallel rectangle in a kd-tree storing n points can be performed in $O(\sqrt[2]{n}+k)$ time, where k is the number of reported points.

4 How to use

Goto the directory which has the source files.
Type make

- Build KD-Tree: Type java KdTreeVisualizer It will popup a window, you can click the point and hit enter. It will show you KD-Tree, now you can draw a rectangle on it. and it will show you the point inside the rectangle in blue color.
- Build Range-Search: Type java RangeSearchVisualizer <filename> (Your file name should conatin point in 2-D space between 0 to 1). It will popup a window with points plotted on it. you can draw rectangle over it and color those points which are inside the rectangle. The blue color represents the points discovered by KD-Tree search while red color represents the points discovered by brute search. In the background terminal you can also see the time take by both methods.

5 Experiment

We compared the KD tree implementation method with brute force method for various values of point set. We uniformly randomly generated the point sets between 0-1 and query rectangle of size 0-0.1. We ploted the graph (figure 4.1) of time taken by both methods for axis-aligned query.

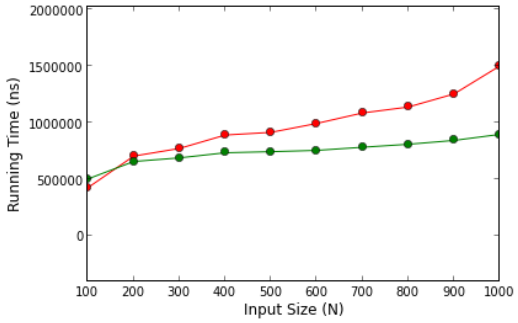

Figure 5.1: Red curve for brute force, green curve for KD-Tree

References

[1] Computational Geometry: Algorithms and Applications.

[2] https://www.cs.princeton.edu/courses/archive/fall12/cos226/assignments/kdtree.html