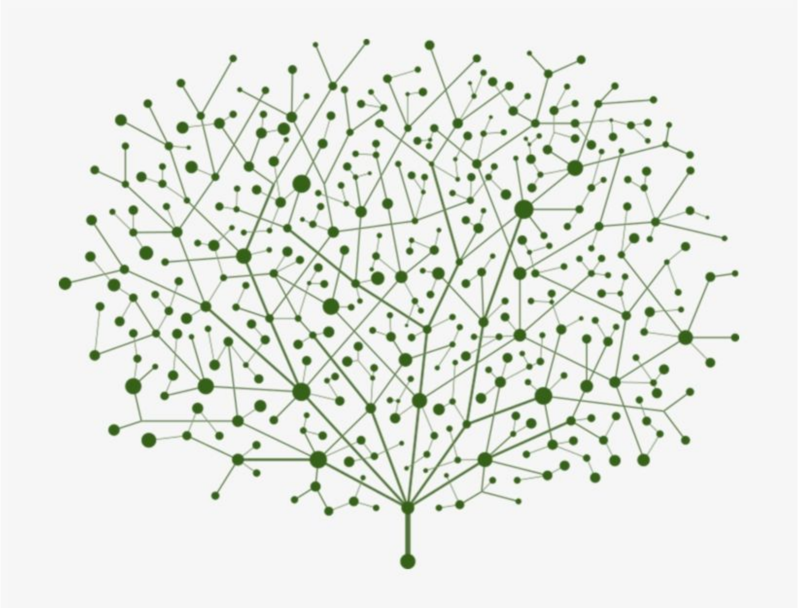
CSC 212 Final Project

K-D Tree



Team Members

Raymond Turrisi

Brennan Richards

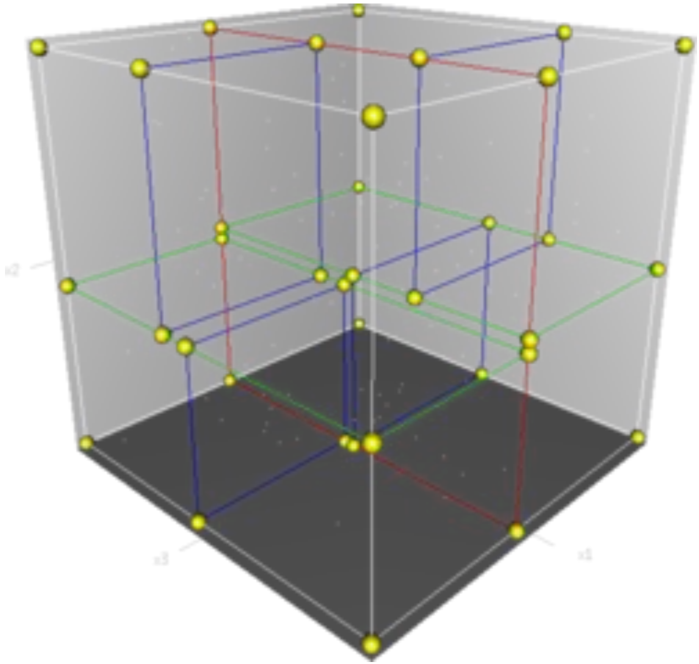
Alfred Timperley

https://github.com/raymondturrisi/CSC212-Final-Project

9 December 2020

Outline

1. Introduction to K-D Trees
2. Theory
3. Analysis
4. Implementation
5. Application
6. Conclusion



Intro (1.1)

- K-Dimensional Trees
  - Developed in the 1970’s, first notably discussed in Dr. Jon Louis Bentley’s paper on the investigation of multidimensional divide-and-conquer techniques, in “Divide and Conquer in Multidimensional Space” which was published in 1975 [1].

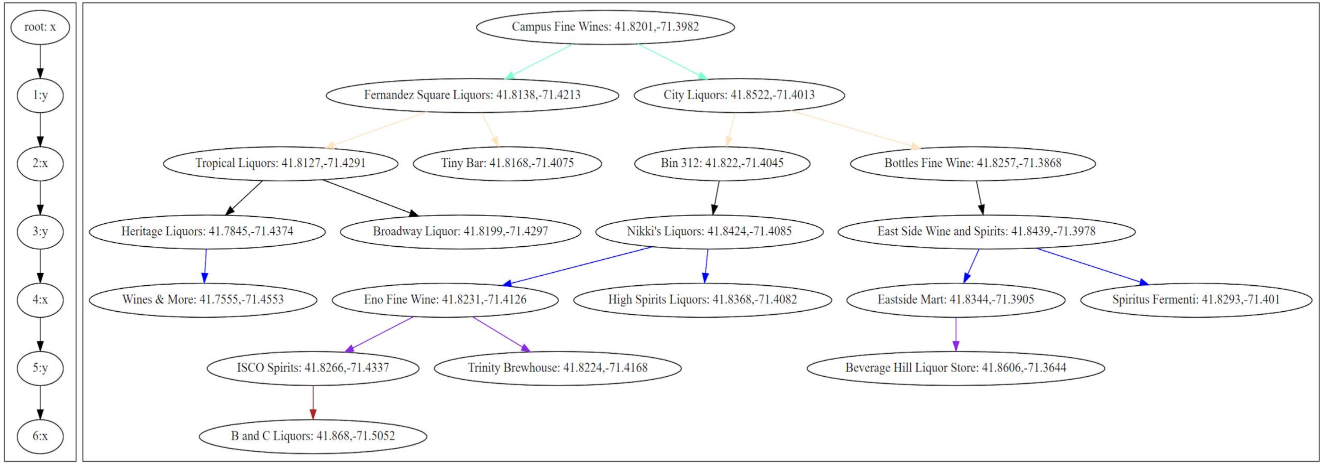


Fig. 1: 2-D KD Tree with local location data.

Intro (1.2)

- Building a K-D Tree
  - Like the BST, the K-D Tree traverses the tree to the left and right based on if the new member's current dimension is less than or greater than the observed nodes dimension.
  - Unlike the BST, it offers dimensionality, making branching decisions based not only on size relation, but also the observed dimension based on the depth from the root node
  - Keywords:
    - Depth, *d*
    - Dimensions, *k*
    - Discriminator, *i*

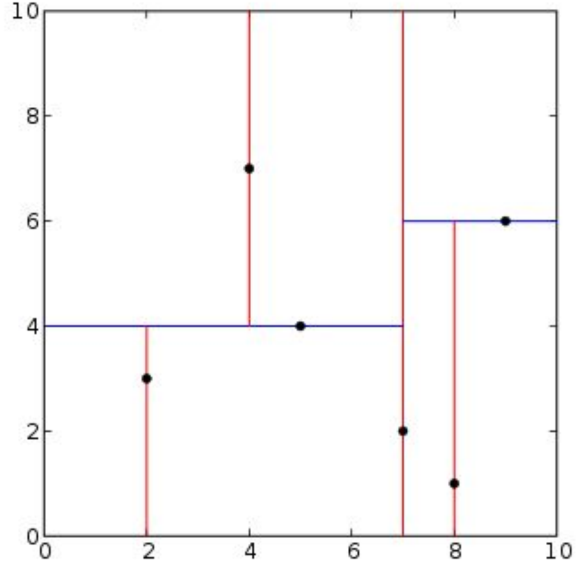


Fig. 2: X-Y plane partitioned by K-D Tree's members.

- Building a K-D Tree, cont. (Example)

$$\{(293,267),(271,98),(372,260),(337,156)\} \qquad i = d\%K, \quad i \in [0,K]$$

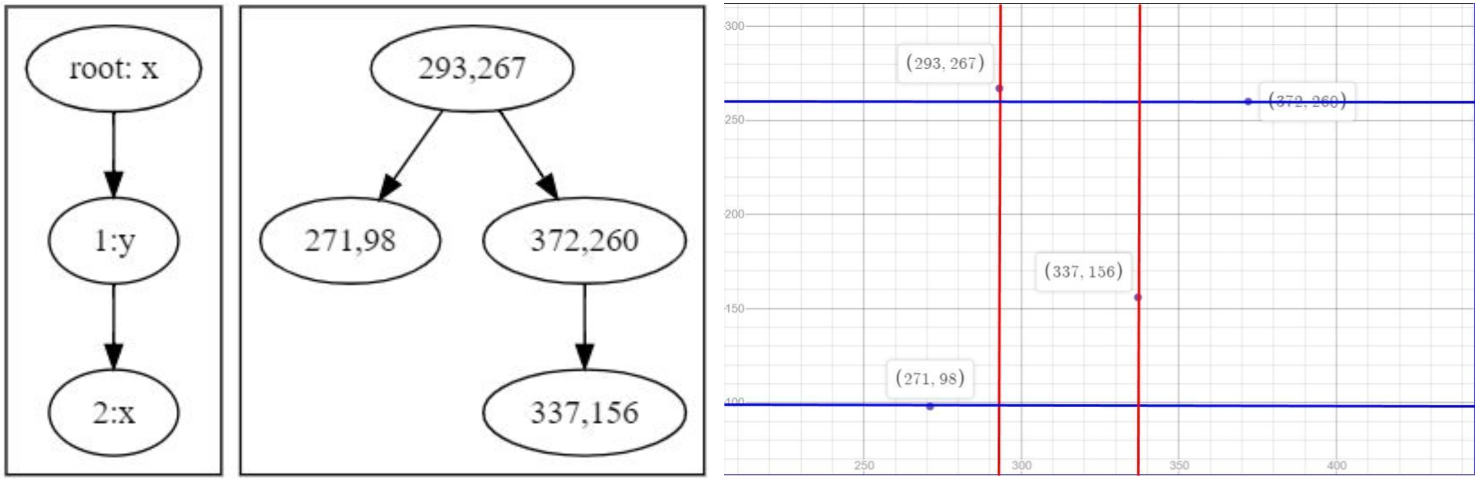


Fig. 3: Construction of a K-D Tree when K = 2, within an X-Y plane.

Theory (2.1)

- K-D Tree has the basic structure of a Binary Tree
- Basic Operations in have the same complexity

Table 1: Time and Space Complexity for Standard Functions

Algorithm	Average	Worst Case
Space	$\Theta(n)$	$\Theta(n)$
Search	$\Theta(log_2 n)$	$\Theta(n)$
Insert	$\Theta(log_2 n)$	$\Theta(n)$
Delete	$\Theta(log_2 n)$	$\Theta(n)$

Theory (2.2)

- Three methods for building a balanced K-D Tree with the following complexities:
  - Heap/Merge sort: Θ(n log^2(n))
  - Median of Medians: Θ(n log(n))
  - Pre-sorted by dimension: Θ(k n log(n))
- We constructed our balancing algorithm with the median of medians algorithm as it offered the best complexity Θ(n)
- Median of Medians Recurrence Relation:
  $T(n) = cn/5 + T(n/5) + cn + T(7n/10)$
  $\Theta(n) = cn$
  $\Theta(n) = n$

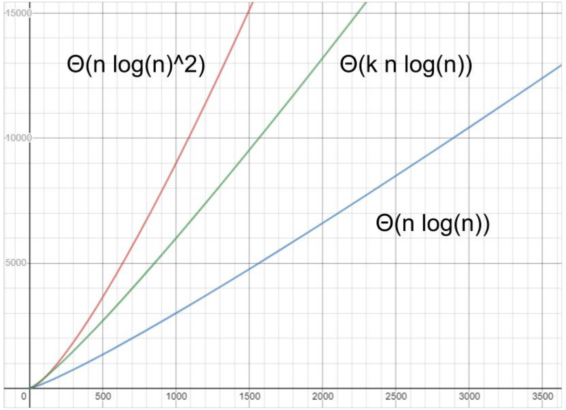


Fig. 4: Graph of asymptotic complexities as n approaches infinity.

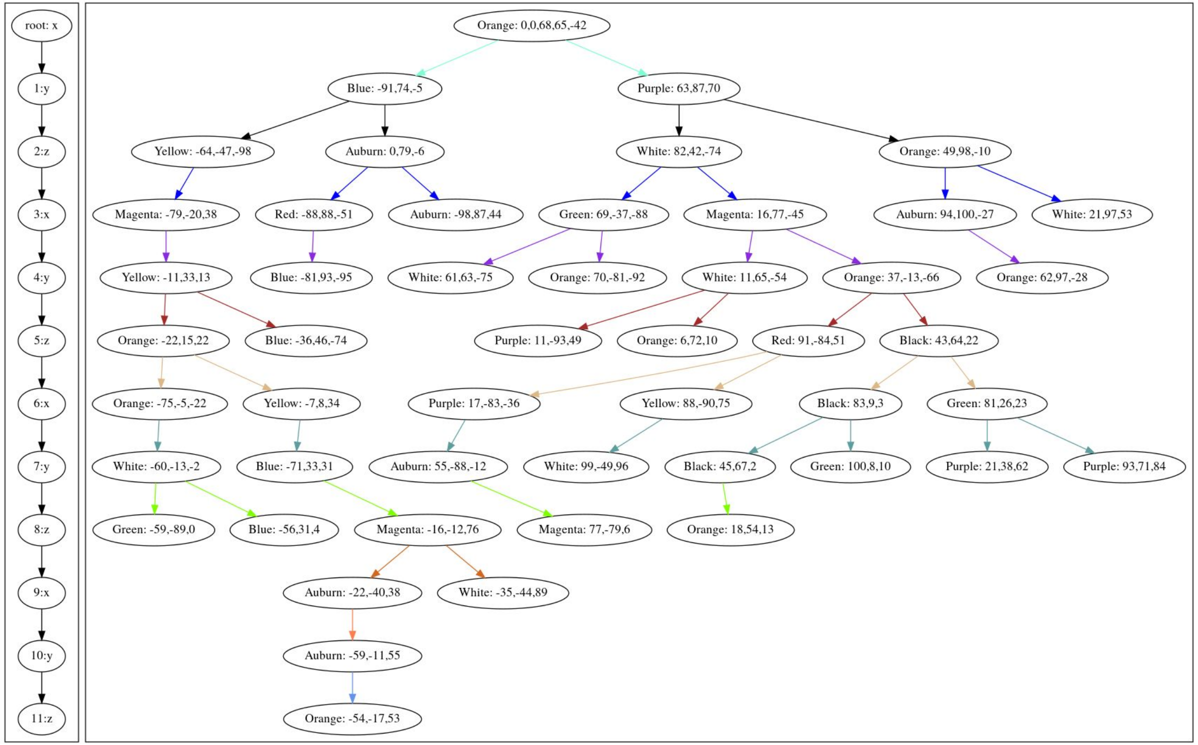


Fig. 5: Unbalanced 2D KD Tree with a height of 11.

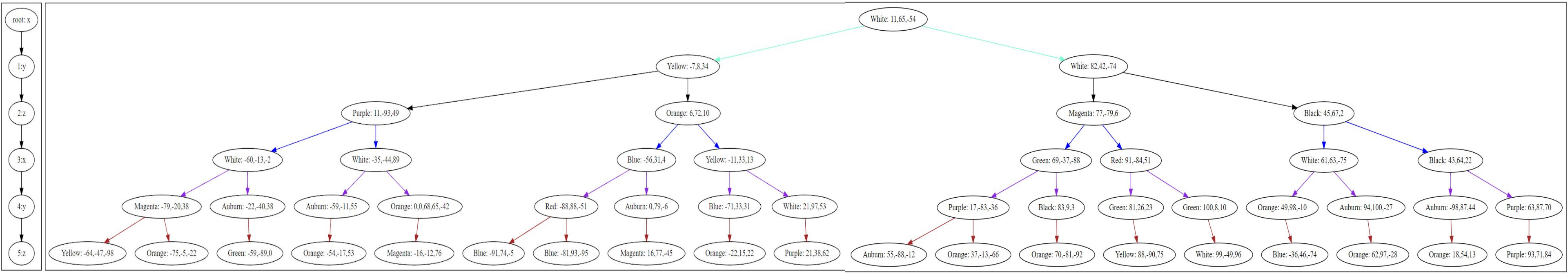


Fig. 6: Balanced 2D KD Tree with a height of 5.

- Balanced Tree Height: 5
  - Constructed in Θ(n log(n))
- Un-Balanced Tree Height: 11
  - Constructed in Θ(log(n))
- Big implications for efficiency of future operations
  - Search
  - Insert
  - Nearest Neighbors Approximation

Constructing a Balanced Tree

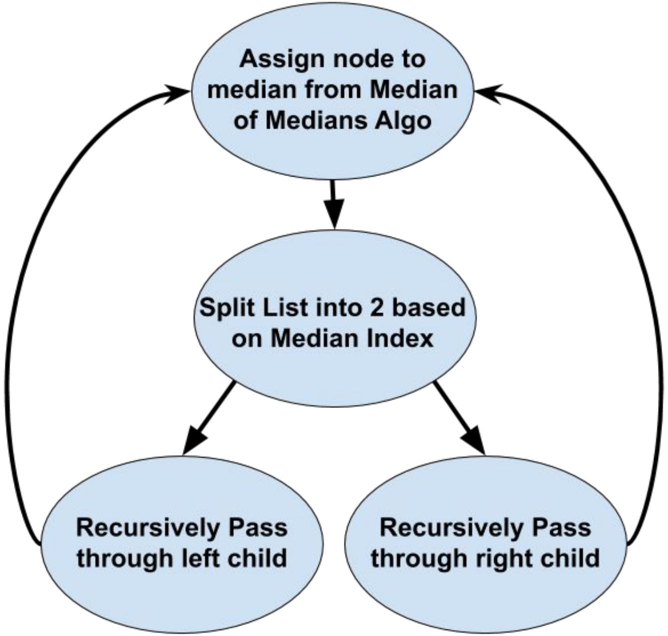


Fig. 7: Diagram of median of medians recurrence .

Algorithm performance (3.1)

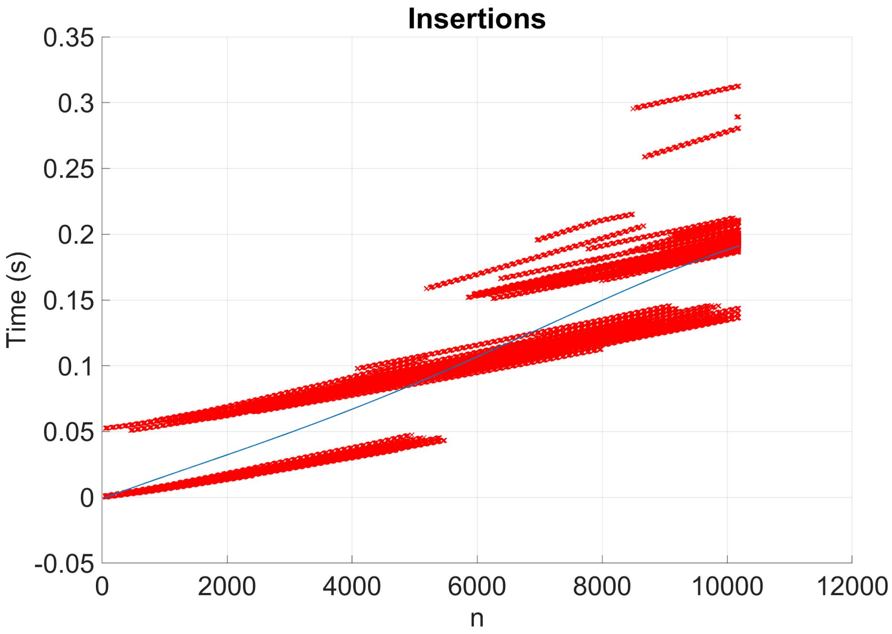


Fig. 8: Data on an unbalanced KD Trees performance in insertions.

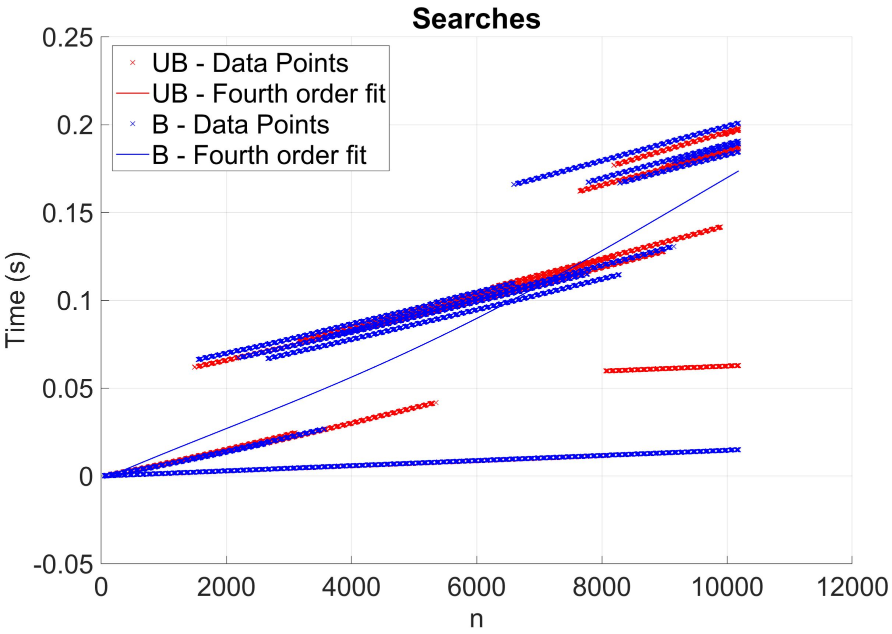


Fig. 9: Data on an unbalanced and balanced KD Trees performance in searching an existing tree.

AP - Nearest Neighbors (3.2)

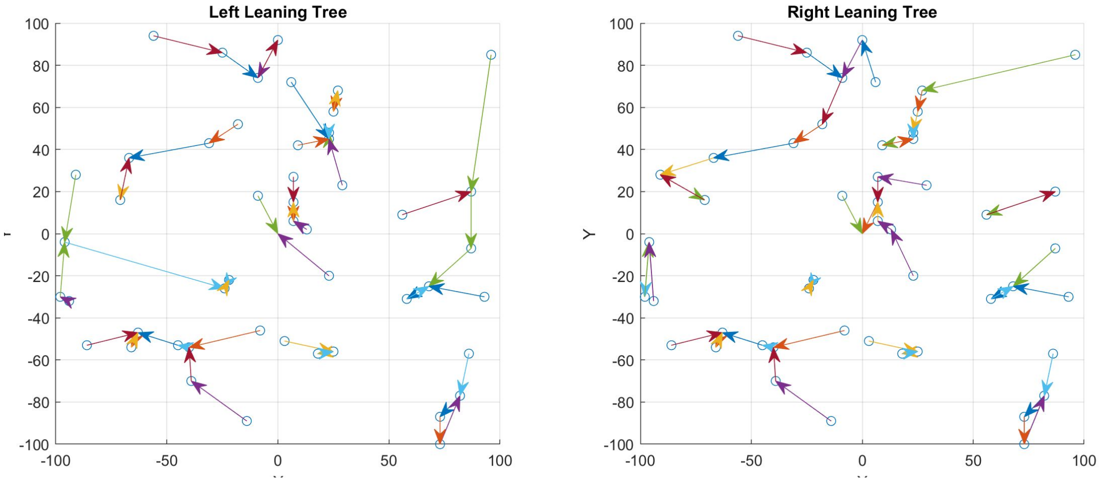


Fig. 10: Nearest Neighbors algorithm, approximated, on KD trees with left and right leaning biases.

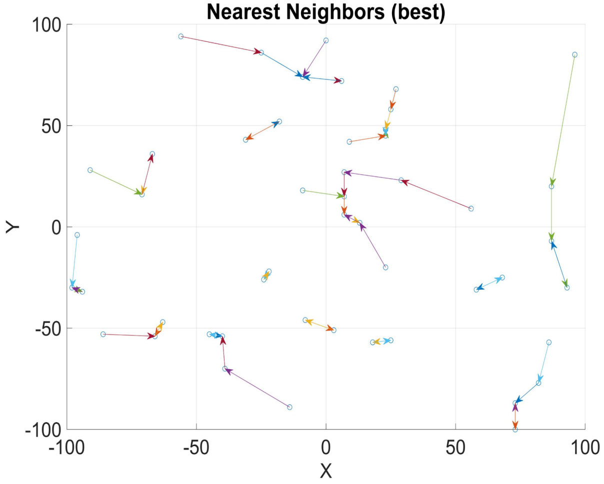


Fig. 11: Nearest Neighbors algorithm, best ()

Nearest Neighbors (3.3)

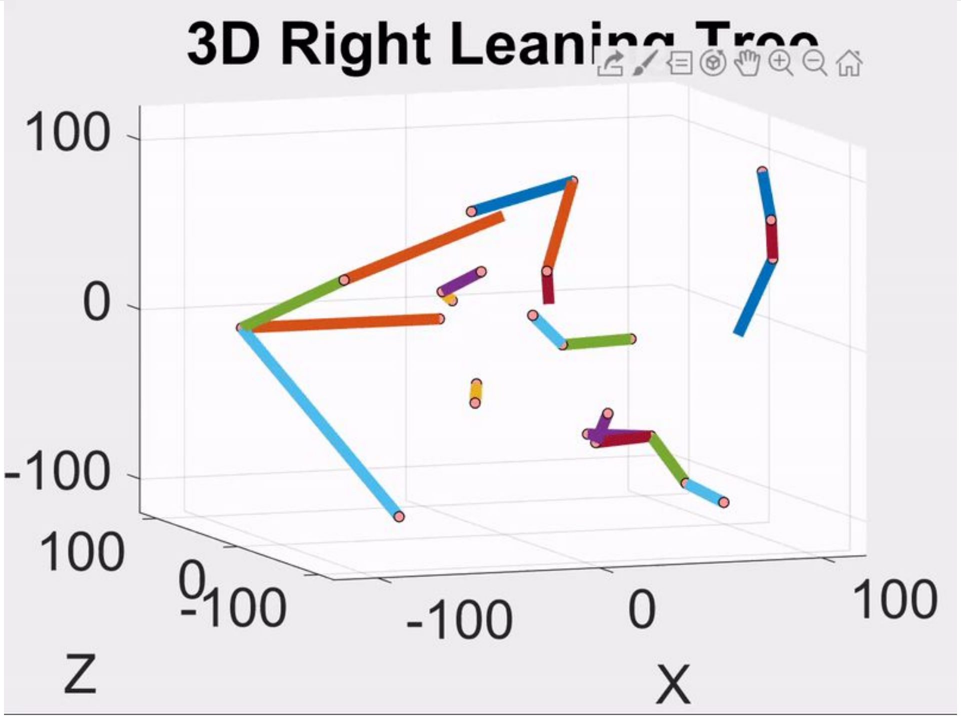


Fig. 12: Nearest Neighbors algorithm, fast, on a 3D right leaning K-D Tree

Nearest Neighbors (3.4)

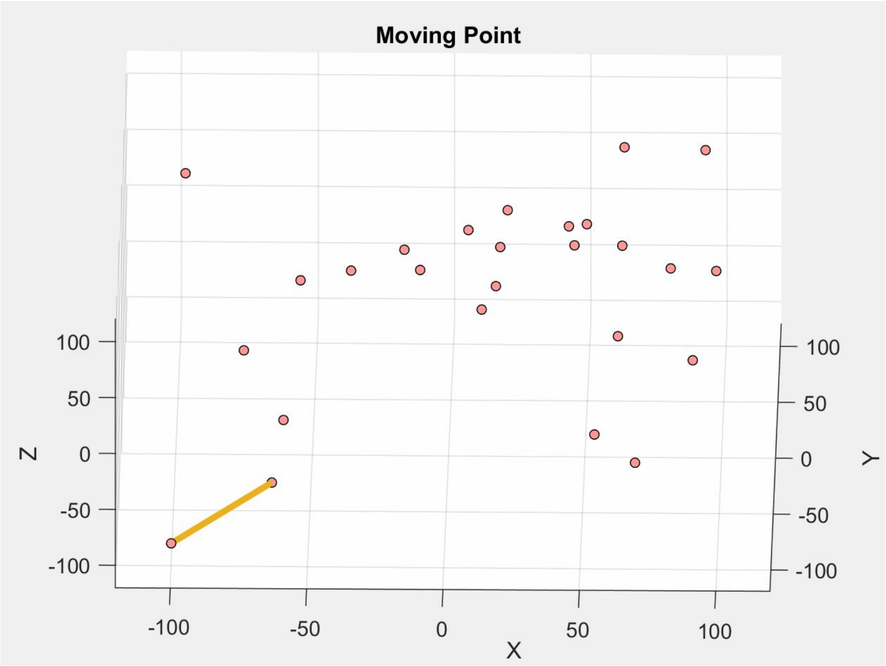


Fig. 13: Nearest Neighbors algorithm, approximated, on a point moving through 3D space

Implementation (4.1)

Implementation in C++

- Fully templated K-D Tree class for to easy deployment on existing classes, requiring small modifications
  - KDTree.hpp
    - Inserts, searches, nearest neighbors, and balancing constructor with Median of Medians
  - Nodes.hpp
  - Median.hpp
  - dataStructs.hpp
    - KDTree_Key class
    - DefaultLocation class

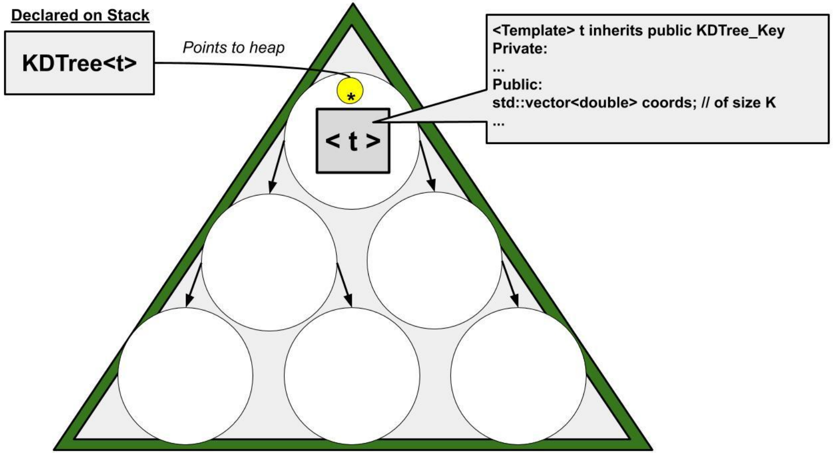


Fig. 14: K-D Tree C++ Class implementation diagram.

Application (5.1)

Motivation

- Find nearest location (coffee shop, theater, etc.)
- Instantiate a 2-dimensional tree
    - Dimension 1: Longitude
    - Dimension 2: Latitude

Application (5.2)

Retrieving location data:

- Python
- Used Yelp application programming interface (API), retrieve data from Yelp's servers using HTTP
- Results
    - 10,000 locations: RI, CT, MA
    - Comma separated value format (CSV)

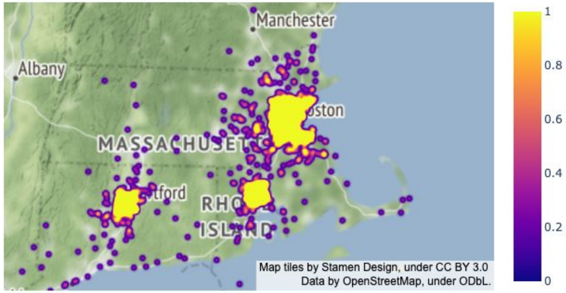


Fig. 15: Representation of our location data set. Shows geographical coverage and density of location data.

Application (5.4)

- User inputs longitude, latitude and an option for what type of location they want to search for

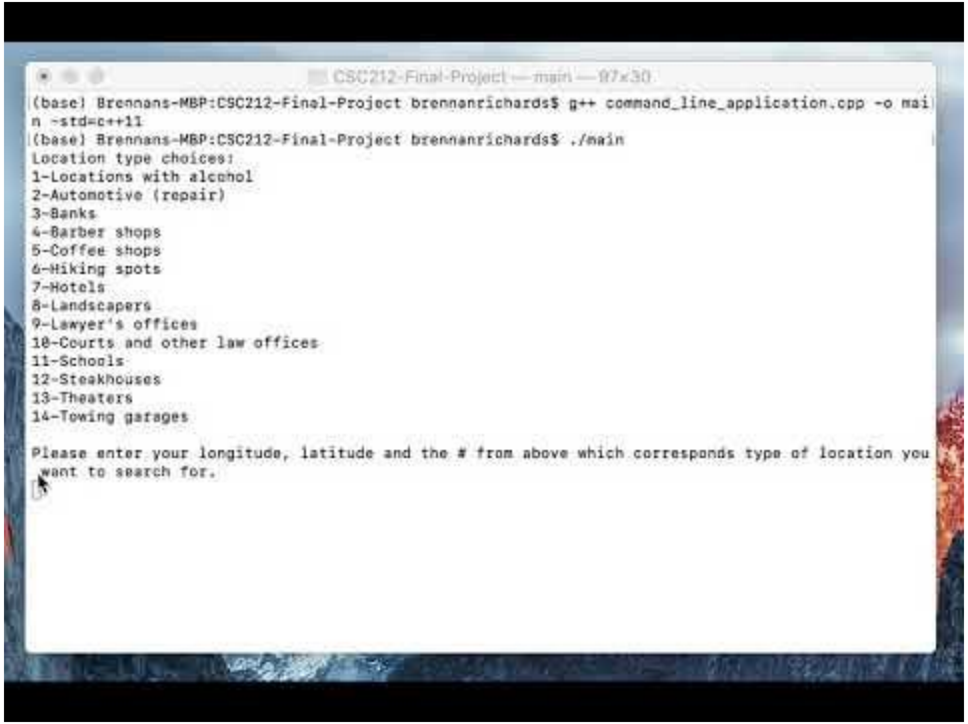


Fig. 16: Demonstration of our team’s location-finding application.

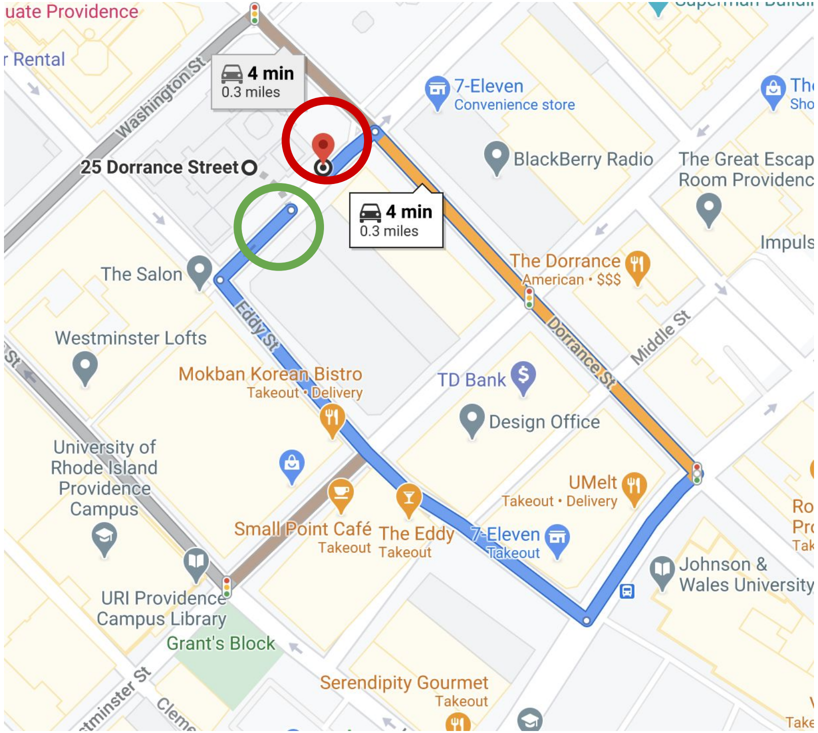
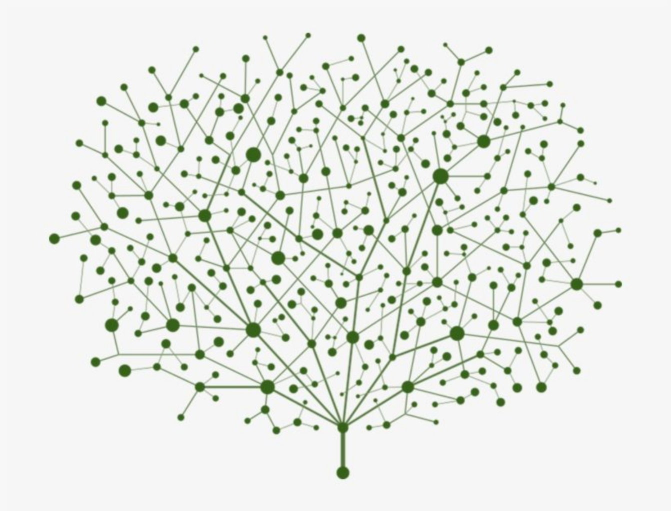


Fig. 17: The results of the demonstration in the previous slide. Circled in green is the longitude latitude passed as input, and circled in red is

Conclusions



Questions?

References

[1] J. L. Bentley and M. I. Shamos,Divide-And-Conquer In Multidimensional Space. Proceed-ings of the Eighth Annual ACM Symposium on Automata and Theory of Computing.