MIT 6.830

Applying SimpleDB to d3.js

Research

by

Ermain, Nchinda, Teddy

Abstract

abstract stuff

Contents

1 Introduction

1.1 d3.js

Data Driven Documents, commonly known as d3.js is a popular graphics and visualization library for javascript. It is primarily developed by Mike Bostock (mbostock), a developer and former Ph.D student at Stanford University. It was during his studies there in the Stanford Vis Group that d3 was first conceived. MBostock and two other researchers published a paper describing a new paridigm for unifying data and web documents. D3 not only joins data to visual page elements, but it makes the underlying data representation transparent so it can be manipulated with changes reflected in the visual representation. The HTML Document Object Model is by itself a static data structure, d3 provides transforms, animations, and iteractivity. At the time of writing th D3 library has 70,451 stars on the code sharing repository GitHub, 17,903 more than Linus Torvald's Linux repository and placing it in the top 10 most popular GitHub repos.

D3 was designed to provide an alternative to existing data manipulation libraries which abstracted away the lower level representation of data. To that end D3 provides both operations on the HTML document elements that are bound to data elements as well as high level operations. The developer using the d3 api has access to common, tedious, and complex transforms as a single api call while maintaining the ability to explicitly override any of d3's functionality with custom functions. Developers can selectively bind html elements to their data expresing what should happen when data changes, new data is generated, and existing data destroyed. Operators are evaluated in real time and can be chained together, for example: d3 provides an api to create a scale from a range of data which then can be fed into an axis generator which then can be automatically displayed. Any updates to the underlying data propogates to the viewable result.

1.2 Motivation

In 6.830, MIT's Database Systems class students, including the authors, were challenged to use the concepts learned in class in a final project. We were inspired to look at D3.js by a comment from a person unaffiliated with the class that D3 used ”a number of database-like constructs internally, some of which are quite primitive. For example, they use nested loop joins and don't reorder selections.” We thought this to be an interesting challenge with multiple positive aspects.

1. We would have to apply concepts learned in class for database systems to a very different scope.
2. Our work if successful would have an impact on an already succesful and public project
3. Once in the d3 codebase our work would continue to be maintained in perpetuity

Unfortunately that remark alone is not enough to form a substantial project, it leaves out the important question of where among D3's submodules one can find the inefficiencies it speaks of. The Library Analysis section below details the results of our group's work to find potential inefficiencies and areas for improvement in the D3 library.

2 Library Analysis

We looked through all of the submodules of D3 for potential places it could be improved by techniques we learned in 6.830. We paid particular attention to places that used nested for loops as those were the motivation for the project. After reading through the d3 codebase the following are descriptions of d3 modules in which we found held potential for improvement via this project.

2.1 Selections

d3-selection is that first library that we analyzed. It fufils the original promise of d3, selecting HTML document elements and joining them with data. It provides setter and getter functions for html element attributes, text, and styling. We believe this library is the origin of the idea of this project, simply searching for the word ”for” in the module source code revealed mutiple places where similar looking code was used to do a nested loop over two elements.

-we looked at creating an index for ¡what module¿ -we looked at caching to remove the need to scan the document selector on each call to find elements as is the common paridgm -write about what this library is used for -it creates a new selection every time it's called, using the 'document' page object to find all the relevant objects -this could be memoized, using browser localstorage, don't need to reinsert elements that already have been inserted

2.2 Delimiter-Separated Values

d3-dsv parses tabular data into a format that can be manipulted and displayed. D3-dsv provides parsing the most common data formats as a single api call, but also provides a function to allow users to specify arbitrary operations on the input dat. In a github issue mbostock suggests a new feature for d3, streaming data [4] from files. Streaming data falls under the category of topics that we learned in 6.830. The idea is analogous to the iterator we each had to build to iterate over tuples in a simple database. Adding streaming capability to d3 could dramatically enhance it's efficiency in processing transforms over data, allowing it to pipeline data processing with data retrieval. An extension of this application would provide a more pleasant user experience, showing a visualization computed using a portion of the base data set while the rest loaded in the background.

2.3 Quadtrees

d3-quadtree is used by other d3 modules to represent datums stored in a 2D space. For example: d3-force uses d3-quadtree to efficiently find nodes that are colliding. Since this d3 module is so heavily used we decided to examine the implementation. -went through process of rewriting search, realized d3 already does it pretty fast - this is just the power of d3

2.4 Time Intervals

, d3-time - there are no nested loops here, but maybe for large intervals it would be better to get $O(logn)$ time queries instead of $O(n)$. This isn't our main focus though

2.5 Forces

, d3-force - collide looks like it's running slow based on experiments but it's about fastest possible using quadtrees, link to the test example in citations

2.6 Transitions

, d3-transition - https://github.com/d3/d3-transition/blob/master/src/transition/select.js we could do something here

2.7 Array

D3 has had an outstanding issue for a few months by mbostock suggesting the addition of database style joins to d3. The feature usecase on the web is analogous to one that might arise when using a traditional relational database such as SQL or PostgreSQL. While in a relational database table records are joined based on matching values in columns, in the proposed d3.join arrays of objects would be joined based on matching keys. In the analogy a table becomes an array of elements, a row becomes the element at a particular index, and the values in a column are a specific property of each element.

3 Improvements

-make a boolean flag to enable index creation -create an index based on user provided id for nodes and comparator, when new nodes are added make sure to add them to the index too

4 d3.join

Resolving the issue created by mbostock on the lack of a *d3.join* method seemed like an especially appropriate move for us to take. Not only was their demonstrable need from the side of d3's developer, but also ability from our end. Through taking 6.830 we learned tradeoffs between multiple types of database joins and how to implement them. We used the experience gained analysing the d3 libraries to determine what sort of functions would compose an extension to d3 for joins. D3 is a large codebase and it was important to us to maintain homogeneity in style of both implementation code as well as the user API with other d3 modules. Before describing the syntax in full the following is a code sample showing the type of code a user might write.

```
var joinOp = d3.join(dataset1, dataset2)
    .key(d => d.id)
var result = joinOp.apply()
console.log(result)
```

The user calls d3.join with two datasets (arrays) of values to be joined. They also specify a *key* function which takes a data element as input and returns the value on which the element should be joined. No computation is actually done until the user calls the *apply* function. Once the result is returned it can be stored, displayed, used in further processing, etc. As an example the last line of our sample code opts to display the result to the console. The following is a full description of the new *d3.join* API.

- *d3.join*()
- join.data(left, right)
- join.key(value) defaults to identity function
- join.leftKey(value)
- join.rightKey(value)
- join.join(value) defaults to nested loop join
- join.predicate(value)
- join.apply()
- d3.nestedLoopJoin
- d3.sortMergeJoin
- d3.hashJoin

5 Results

-how does our intersections example run with this new code

5.1 d3.join

We compared our implementation of common joins to the next two most popular join libraries for javascript and found our implementation to be many times faster.

6 Conclusion

7 Acknowledgements

References

[1] https://github.com/ermain/d3js_experiments

[2] https://github.com/d3/d3/blob/master/API.md

[3] http://vis.stanford.edu/files/2011-D3-InfoVis.pdf

[4] https://github.com/d3/d3-dsv/issues/20