Data Structure Algorithm Improvements

Student : Koustav Chowdhury

Mentor : Jameson Nash

Implementation of Robin Hood Hashing scheme in Julia

The primary goal of this project would be to replace the present scheme of hashing in Dict, to this neat variation of open-address hash. This technique isn't very well-known, but it makes a huge practical difference because it both improves performance and space utilization compared to other "standard" hash tables (e.g. chaining), or the currently implemented one.

Details

Here are the benefits of Robin Hood Hashing scheme , summarized:

- **High load factors** can be used without seriously affecting performance. 0.9 is perfectly reasonable as a default (0.95 or higher works too, it mainly affects insertion cost a bit).
- No linked lists or other extra pointers. This reduces cache misses and storage overhead. Your underlying structure can be a simple flat array since it's just open addressing under the hood. [This is true for any open-addressed hash]
- Lookup and insertion logic is fast. Again, no linked lists to traverse or other complications, just a linear probe sequence with a few checks per element.
- Unlike other open addressing schemes, looking for non-existent elements is still fast.

As the load factor grows larger, the hash table becomes slower, and it may even fail to work (depending on the method used). The expected constant time property of a hash table assumes that the load factor is kept below some bound. For a *fixed* number of buckets, the time for a lookup grows with the number of entries and therefore the desired constant time is not achieved. As a real-world example, the default load factor for a HashMap in Java 10 is 0.75, which "offers a good tradeoff between time and space costs."

The current implementation of hashing scheme in Julia supports load factor upto 0.67, and then it is rehashed. See here

The idea is that a new key may displace a key already inserted, if its probe count is larger than that of the key at the current position. The net effect of this is that it reduces worst case search times in the table. This is similar to ordered hash tables except that the criterion for bumping a key does not depend on a direct relationship between the keys. Since both the worst case and the variation in the number of probes is reduced dramatically, an interesting variation is to probe the table starting at the expected successful probe value and then expand from that position in both directions. (source : Wikipedia).

What do you want to have completed by the end of the program?

A modified implementation of hashing scheme in Julia, port it to use Robin Hood hashing scheme.

Who's interested in the work, and how will it benefit them?

The projects targets a marked improvement in the performance of Dict in Julia, which can benefit each and every user of Julia and its packages.

What are the potential hurdles you might encounter, and how can you resolve them?

As mentioned in the milestones, listed below, implementing concurrent RH hash will be great challenge and learning opportunity for me, as I am not much experienced in that domain.

Milestones

1. Implementation of Robin Hood Hashing.
2. Transition of **rehash!** functions in Julia to use RH hashing scheme.
3. [*Extended milestone*] Implementation of concurrent Robin Hood hashing scheme.

Proposed Timeline of work

Week #1, #2

Implementation of Robin Hood Hashing scheme, as per the the original paper.

Week #3

Testing using the Chi-square test and Avalanche test, followed by benchmarking and documentation.

Week #4, #5

Optimise the performance by trying out heuristics.

There are several such heuristics mentioned in various papers and articles. I am listing them accordingly:

1. https://www.dmtcs.org/pdfpapers/dmAD0127.pdf
2. https://www.pvk.ca/Blog/numerical_experiments_in_hashing.html
3. https://www.pvk.ca/Blog/more_numerical_experiments_in_hashing.html
4. https://www.sebastiansylvan.com/post/robin-hood-hashing-should-be-your-default-hash-table-implementation/
5. https://www.sebastiansylvan.com/post/more-on-robin-hood-hashing-2/
6. http://codecapsule.com/2013/11/17/robin-hood-hashing-backward-shift-deletion/

In addition to this, I plan to try out prime-sized bucketing for the primary reason of improving collision safety.

Week #6

Buffer week, for reviewing and completing all the unfinished work.

Week #7, #8

Changing the implementation of functions in Julia, making them use RH hash, changes to be made in ***DataStructures.jl*** first, if nothing breaks, change in ***JuliaLang/julia***.

Week #9

Buffer week.

Roll out a beta-development model, for use. Smoothing out any rough edges in implementation.

Week #10, #11

Push for concurrent implementation of Robin Hood hashing scheme, as per this paper . This will be in development branch.

Week #12, 13

Buffer weeks.

For documenting all the work, and probably, see through the implementation of concurrent RH hashing scheme and its integration in *DataStructures.jl* .

I'll try to abide by this timeline as closely as possible. Apart from this, during the pre-GSoC period, I'll try contributing to different packages of Julia and working on several of my patches, with purpose of getting them merged.

During the community bonding period, I'll try to interact as much as I can, to the wonderful community and also, read up the codebase of *DataStructures.jl* and base repository of Julia, marking out places where potential change has to be made. Apart from these, I'll surely keep on disturbing my mentor asking for guidance and advices regarding this project.

About me

I am Koustav Chowdhury, a second-year undergraduate student from the Department of Computer Science and Engineering at Indian Institute of Technology (IIT), Kharagpur.

I like maths and programming, and you can interest me anytime with any puzzles. I like to explore new things, be it learning new programming languages or new data structures. I do competitive programming for fun, as I feel it's one of the best way to utilise your time outside the mundane academics. I got a rank of 202 in the recently organised round A of Google Kickstart, I hope to improve upon that in the upcoming rounds. Apart from this, I'm an avid reader and a Netflix addict. Also, I code while listening to music.

I wrote a music player in Ruby, while I was in the first-year of my college. Although there are bugs, I am still very much proud of that project.

I am current working on implementing a Music Recommender System, using the Spotify API. I plan to complete it by the first week of April. As of now, it is a **WIP.**

Contact Information

- **E-mail address : kc99.kol@gmail.com**
- **Github : https://github.com/eulerkochy**
- **Location : Indian Institute of Technology(IIT) Kharagpur, West Bengal, India**
- **Timezone: Indian Standard Time (UTC +05:30)**

My contributions in Julia

It all started with a implementation of NumberSuffix.jl. This was done upon seeing the query of someone(I can't recall, damn!) on the **#general** channel.

I now have the experience of working with Julia for almost three months, and day-by-day, I'm growing in confidence about writing code in this language.

In this context, I will list out all the PRs that I've worked upon, or I am working currently.

Merged PRs

1. https://github.com/JuliaLang/julia/pull/31125 - My first contribution and this taught me the proper way of using **git**. I am grateful to Stefan Karpinski and others, who guided me, in getting this merged.
2. https://github.com/JuliaLang/julia/pull/31237 - Second one, mostly getting familiar with the language
3. https://github.com/JuliaLang/julia/pull/31218 - Dumb, but learnt the syntax of docstrings in Julia
4. https://github.com/JuliaGraphs/LightGraphs.jl/pull/1160 - First contribution with ease of typing code in Julia, and vigorous testing.

Approved PRs

1. https://github.com/JuliaGraphs/LightGraphs.jl/pull/1181 - This solves the problem of having stack overflow on large graphs for finding out bridges.

2. https://github.com/JuliaLang/julia/pull/31309 - A simple hack for getting the job done.

WIP

1. https://github.com/JuliaLang/julia/pull/31321 - This is a minor bugfix, but this raises a slight confusion over a potential change in current implementation in Julia
2. https://github.com/JuliaLang/julia/pull/31313 - These features were absent for BlockDiagonal matrices, and adding this raised concerns of whether checking BlockDiagonals should be recursive, or not. Waiting for the final judgement regarding this.
3. https://github.com/JuliaCollections/DataStructures.jl/pull/485- Add the functionality of `Fenwick Tree` to *DataStructures.jl*, I'll be **bump**ing this now, it's activity has been stagnant for quite sometime.
4. https://github.com/JuliaStats/Distributions.jl/pull/840 - A dying PR was revived, and the codes were corrected. This taught me how to work on someone else's PR. A very subtle

amount of work is left in this one. This primarily focuses on implementing Chernoff distribution.

5. https://github.com/JuliaLang/julia/pull/31151 - This was in the earlier days of exploring Julia. I found a fix for the issue mentioned in the PR, but seems that the issue has deep roots for which this has been stagnant for quite a long period of time.
6. https://github.com/JuliaLang/julia/pull/31423 - A recent PR, which revealed the internal implementation in Julia.

Logistics

What other time commitments, such as summer courses, other jobs, planned vacations, etc., will you have over the summer?

I expect no major hurdles. I can easily allocate 40+ hours a week for working in this project and there can be no dearth in motivation in between. My college re-opens on July 17th after the summer break, but that shouldn't cause much of a headache. I'll have regular communication with my mentor, informing him of my inability to work in case of unforeseen circumstances.