Date: 11.09.2024

RFC 1.0: ChadGrok (proposed name, open to suggestions)

Objective

Make finding and understanding code easier in large C/C++ codebases.

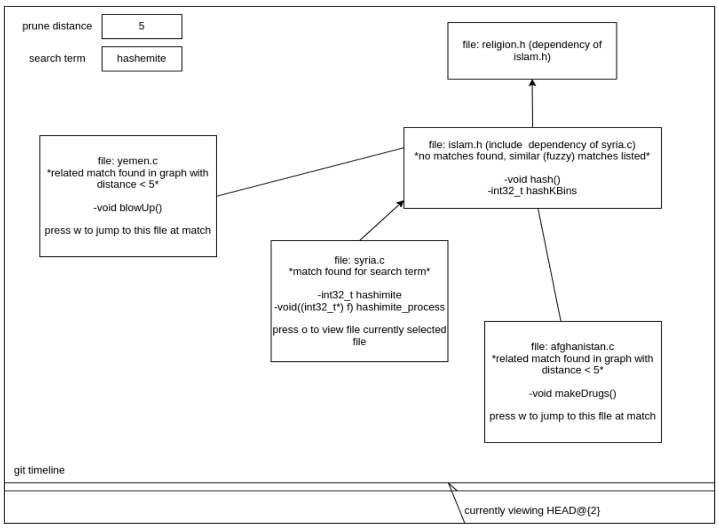
Motivation

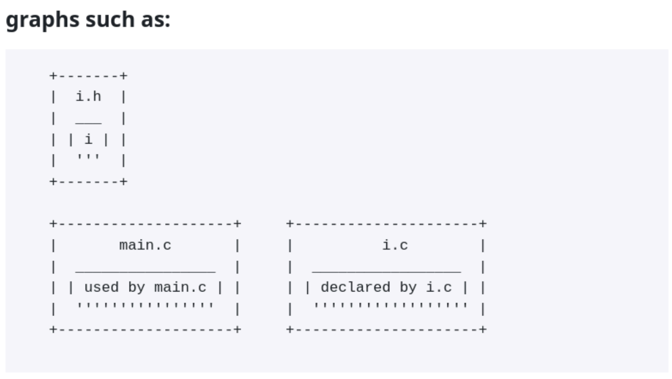
The current go-to tool for finding source files in a codebase is “grep”, or some similar code search functionality in modern IDEs (used by JS devs who do not know what grep is). These are useful tools for finding text, but code is not the same thing as text. Critically, code inherently has a graph-like structure. Each source file has parent dependencies and symbols, which in turn have their own dependencies, and so on.

The problem with using a tool like “grep” to find code in a large codebase is that grep is a linear-search tool, it doesn’t respect the syntax and semantics of code. As a result, instead of actually writing code, programmers spend precious time crawling through each file returned by grep, trying to find the ideal entry-point for their change.

Proposed Solution and User Benefit

The proposed solution is to build a graph-based code search tool, perhaps built on top of Anon’s current work with “depgrah”. This tool will have a few key features that chads find helpful, but primarily, it will serve as a way to “visually” grep a codebase. Instead of showing files as text, it will parse the syntactic trees of source code, collecting symbols and other information along with the actual code. Instead of simply displaying code matches, when a user searches for a particular symbol or piece of text, the tool will display a graph of the code matches. The tool will be highly configurable, allowing for recursive, fuzzy editing. Recursive fuzzy editing means that it will allow users to chain greps in an easy and natural way, allowing them to quickly narrow and restrict the “graph” of source code to find the correct file that they are looking to edit. Here are some examples of what it might look like (rough ideas):


prune distance
5
search term
hashemite
file: religion.h (dependency of islam.h)
file: yemen.c
related match found in graph with distance < 5
-void blowUp()
press w to jump to this file at match
file: islam.h (include dependency of syria.c)
no matches found, similar (fuzzy) matches listed
-void hash()
-int32_t hashKBins
file: syria.c
match found for search term
-int32_t hashimite
-void((int32_t*) f) hashimite_process
press o to view file currently selected file
file: afghanistan.c
related match found in graph with distance < 5
-void makeDrugs()
press w to jump to this file at match
git timeline
currently viewing HEAD@{2}


graphs such as:

```
+-------+
|  i.h  |
|  ___  |
| | i | |
|  '''  |
+-------+

+--------------------+     +---------------------+
|       main.c       |     |         i.c         |
|  ________________  |     |  _________________  |
| | used by main.c | |     | | declared by i.c | |
|  ''''''''''''''''  |     |  '''''''''''''''''  |
+--------------------+     +---------------------+
```

Date: 11.09.2024

RFC 1.0: ChadGrok (proposed name, open to suggestions)

High-level Architecture Proposal

We should discuss the specific details together over Mumble or IRC, but I think it would be wise and easier to split this program into 2 parts:

Anons NOTE: agreed

1. Front-end/client
    a. This can be implemented in various ways, e.g. as a script that uses graphviz and some Python crap with tkinter, it could be a C script, it could even be some Perl cgi crap for extra added nausea. The point is that the implementation of the front-end should be separate, so that it is easy to get this integrated with any client, text editor, or ecosystem as desired (including emacs through some elisp script / vim through some vim script).

        Anons NOTE: my vote is on C/Tcl/Tk

    b. We should decide on some consensus implementation for the test front end, maybe some TUI or GUI.

        Anons NOTE: TUI would not cut down on the development time at all.
        For a test front end i propose a simple cli and output images.

    c. The way we show this graph is also interesting. Should it be interactive? Should the user be able to click on the graph nodes and open the files/matches using their mouse cursor? Alternatively, should we just make it keyboard-only and do keyboard-jumping.

        Anons NOTE: clickable would be great, however i dont see how we could possibly accomplish it without taking months of dev time in out backs. I used graphviz, right? The way lib cgraph works is that you build a graph data structure, then it writes to a plaintext file format which the graphviz cli can process. The obvious problem is that theres no was to tell at which pixel did the "parts" end up, so we are totally alone with the rendering, where as i see it the most painful would be the layout. Therefor, unless someone has a really bright idea, i strongly advice against this feature.

2. Backend/daemon: This is the engine of the program, which does the fuzzy searching of files, parses the syntactic tree, and allows for configuration of the editor. There is more than meets the eye to this part, particularly:

    Anons NOTE: i get the appeal of deamonizing, but this should not be a primary concern. We are not retarded, we can write something that has reasonable response times on cold starts

    a. Handling fuzzy + recursive search: we want the search to be fuzzy (so that users can still find matches to text, comments, etc), but not too fuzzy that the search does not pick up any symbols. Thus some sweet spot of fuzzy + recursive search (where the user can type one thing, and then recursively narrow down/filter the results) seems optimal. We would have to play around and test various ways of this that feel good. Having this perfected would be a HUGE advantage over grep in and of itself, because grep chains are not very good.

i. Some implementation considerations with this can make a big difference. For example, how does recursive grep work? If I want to search for the word “hashimite” (1) and then afterwards want to narrow it down to (2) “hashimite” AND “salt”, how do we filter results? By appearances of BOTH the words hashimite and salt in the same function? Or by appearances of hashimite and salt in +/- 20 lines from the first match of “hashimite”? Or just multiple matches in the same file? So many ways of doing this, all seem potentially valid.

    Anons NOTE: dont even try using levensthein distance as googling may suggest it. Its way too slow for bigdata

b. Parsing code or integrating with existing compiler (e.g., clang, gcc, if there is a way to do this): A few options exist for getting the symbols, functions, and all the “processed code” out. Examples include CTAGS, GNU Global, using compile_commands database generated by the compiler, or some other clever nice way we can think of. What’s critical here is the performance and ease of processing the symbol data into a graph.

Anons NOTE:
> ctags
i love ctags, i use it in my vim plugin to get dynamic symbol highlighting. However exactly because of this i know its quite limited. It cannot inform you about symbol usage at all. This means that if you want a list of definitions and occurrences, you would have to parse the file twice anyways. Even tho one is someone complex while the other could be done with the equivalent of a grep, it still sounds iffy.
> gnu globals
No. no gnuware. Total GNU Death.
–
I recommend tree-sitter. Its basically ideal, but has 2 big problems:
> different languages produce wildly different syntax trees and theres no abstraction to interact with them even remotely homogeniously (for example getting the name of a function requires different code for each language)
> its annoyingly verbose

> *The first one however should not be a concern as long as we keep to C/C++ while the second is bearable and i have experimented with this: https://github.com/agvxov/tbsp*
> *Again, proof of concept, the code is horrible, but i wonder if you think it could be valid.*

c. Git timeline: The user should be able to view source code over time and go back/forth through a git history. Emil has some ideas on how to implement this, I believe. I am not sure where to start with this.
   > *Anons NOTE: good idea, but again, should not be a primary concern.*

d. Graph pruning and returning similar results: The way that we prune graph nodes is an important point. By pruning the graph I mean the following. In a large or even moderately sized codebase, there can be multiple matches of text and each match can have several dependencies or symbols. The symbols that we choose to show to the user need to be filtered out. We should only show the most important and relevant functions and symbols. We should also not show too many graph nodes - we should only show the most similar, likely matches to what the user is looking for.
   i. Other people may have thought about this. I think I have heard of things like "code query languages" before. I found this as a hit research paper online, but this is not what I remember- I think there's a popular repo that addresses the problem of querying large codebases through a declarative language (anyone know what this is called? If not I will find it – TODO).
      > *Anons NOTE: link to the full article above: https://www.cs.kent.edu/~jmaletic/papers/SANER17-srcQL.pdf*
      > *This is also a good example what NOT to do: https://codeql.github.com/publications/ql-for-source-code-analysis.pdf*
      > *(the SQL like stuff would be cool, but it devolves into mongodb tier retardedness)*

   ii. Could do some regex hacking.https://ieeexplore.ieee.org/document/7884655
   iii. The way we filter and show nearby matches will make or break this project. It is a very, very interesting question of how to make a nice, configurable algorithm that does this.
      1. Article about how a very old system at Google that searches code works: https://swtch.com/~rsc/regexp/regexp4.html (this is an

RFC 1.0: ChadGrok (proposed name, open to suggestions)

open source tool as well, but I don't know how it compares to ripgrep)

a. Note that there is a hack they mention very interesting here.

> There is too much source code in the world for Code Search to have kept it all in memory and run a regexp search over it all for every query, no matter how fast the regexp engine. Instead, Code Search used an inverted index to identify candidate documents to be searched, scored, ranked, and eventually shown as results.

Regular expression matches do not always line up nicely on word boundaries, so the inverted index cannot be based on words like in the previous example. Instead, we can use an old information retrieval trick and build an index of *n*-grams, substrings of length *n*. This sounds more general than it is. In practice, there are too few distinct 2-grams and too many distinct 4-grams, so 3-grams (trigrams) it is.

Tech Stack

We should discuss this on Mumble/IRC and come to a consensus.

Performance Considerations

We should benchmark our backend code on popular open source projects to ensure that it is performant. Open to suggestions, but something large like the Linux kernel, or some big Apache/GNU C project would be possible options. A specific metric for performance we can use is line count and time to parse per line. More than likely we will use an existing grep program and build on top of it, such as GNU grep, ag (the Silver Searcher), or ripgrep.

The rendering part of things is also non-trivial. We should benchmark this also. Depending on how we implement things, the number of nodes on the graph may be dynamic. We could benchmark the time it takes to draw 1000 nodes, or something like that. We could also benchmark the time it takes to draw one node per large file of code (perhaps some high LOC file in the kernel). It doesn't need to be insanely fast crap, as long as it isn't Javasir-level slow.

Anons NOTE:
>It doesn't need to be insanely fast crap, as long as it isn't Javasir-level slow.
Thats a very good way to put it

Contribution

More than anything this is meant to be a fun project for the Chad community. I am excited to work on this and the more chads that work on it, the more fun it will be.

Anons NOTE: that sounds like some from a reddit post

Date: 11.09.2024

RFC 1.0: ChadGrok (proposed name, open to suggestions)

Telegram: https://t.me/HelpAndDiscussion
IRC: irc.rizon.net #/g/chad

Anons NOTE: the name is terrible, i vote for whatever Emil comes up with. He has yet to fail me with naming
This all reminds me of what cscope's purpose is supposed to be and why i forked it. Too bad the code base is shit. Anyways, I very much support flexable design as i would like to merge them together in the future one way or another.

STOP USING ONLINE SLOP WORD DOCUMENTS

(as for a reason, im most certain i fucked up something by accident without even reallizing. For example, the font seems to be fucked in one paragraph. Hopefully i did not cut anything out)