CS 4414/5416 Fall 2025 - Assignment 1 Infrastructure
Release Date: September 1, 2025

These instructions assume familiarity with the CS 3410 and their infrastructure (Fall 2024 and beyond); for more details on specific steps or command arguments, please refer to https://www.cs.cornell.edu/courses/cs3410/2024fa/course/infra.html.

Retrieve Starter Code

Download the starter code from the provided GitHub repository onto your local machine:

```
> git clone https://github.com/shouxulin/CS4414-HW1
```

Docker Image Setup

Make sure you have Docker installed and have the Docker desktop application ***actively running*** on your machine.*

To download the Docker container image we have set up:

```
> docker pull ghcr.io/shouxulin/cs4414hw1:latest
```

* Important Note:

Some students running on macOS may run into malware issues installing Docker via the Docker website:

Please note that it's not actually malware – Docker messed up with signing their application in compliance with certain Apple rules. To get around this issue, go to the terminal and run the following commands:

```
> brew uninstall --cask docker --force
```

```
> brew uninstall --formula docker --force
> brew install --cask docker
```

The above commands will uninstall the existing Docker installation and install Docker via homebrew. After installation finishes, Docker should pop up in the application folder and you should be able to access the Docker GUI.

Shortcut to Docker Command (`rv` alias)

We will create an alias to shorten the command that tells Docker to run a given command in the CS 4414 infrastructure container.

A. Temporary Alias (Current Session Only)

If you want the alias to only last for your current terminal session:

```
> alias rv='docker run -i --rm -v "$PWD":/root ghcr.io/shouxulin/cs4414hw1'
```

B. Persistent Alias (All Future Sessions)

To make it stick around when you open a new terminal window, type this command to find out which shell you're using:

```
> echo $SHELL
```

At the end of the following command, either change to `bashrc` or `zshrc` or other according to your shell.

```
> echo "alias rv='docker run -i --rm -v \"\$PWD\":/root ghcr.io/shouxulin/cs4414hw1'" >> ~/.zshrc
```

Example Usage:

A. Compile & Run

To compile, make sure you are in the `part1` directory

```
> rv g++ -std=c++20 -O3 -Wall main.cpp knn.cpp -o main
```

NOTE: In Assignment 1, a `Makefile` has been provided for each part to simplify the compilation for you. Instead of running the above, run the following:

```
> rv make all
```

To run the newly compiled program, here are example inputs:

```
> rv ./main ./data/1d-1.json ./data/1d-100.json 2
```

B. Debugging (gdb & valgrind)

>

```
(base) tinacheng@Tinas-Air 4414-hw1 % rv valgrind --leak-check=full --track-origins=yes ./knn 0 ./data/1d-1.json ./data/1d-10
0.json 2
==1== Memcheck, a memory error detector
==1== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==1== Using Valgrind-3.18.1 and LibVEX; rerun with -h for copyright info
==1== Command: ./knn 0 ./data/1d-1.json ./data/1d-100.json 2
==1==
query:
  text:    "0.44733878216547507"

Neighbor 1:
  id:      25, dist = 0.00543949
  text:    "0.4527782840515048"

Neighbor 2:
  id:      33, dist = 0.00682023
  text:    "0.4405185480468711"

#### Performance Metrics ####
Elapsed time: 541.175 ms
Processing time: 463.877 ms
KD-tree build time: 44.0561 ms
K-NN query time: 21.3734 ms
==1==
==1== HEAP SUMMARY:
==1==     in use at exit: 2,400 bytes in 100 blocks
==1==   total heap usage: 1,684 allocs, 1,584 frees, 191,561 bytes allocated
==1==
==1== 2,400 (24 direct, 2,376 indirect) bytes in 1 blocks are definitely lost in loss record 100 of 100
==1==    at 0x4849013: operator new(unsigned long) (in /usr/libexec/valgrind/vgpreload_memcheck-amd64-linux.so)
==1==    by 0x11701D: Node<float>* buildKD<float>(std::vector<std::pair<float, int>, std::allocator<std::pair<float, int> > >
&, int) (knn.cpp:101)
==1==    by 0x1129D0: int runMain<float>(int, char**) (knn.cpp:272)
==1==    by 0x10DDF1: main (knn.cpp:359)
==1==
==1== LEAK SUMMARY:
==1==    definitely lost: 24 bytes in 1 blocks
==1==    indirectly lost: 2,376 bytes in 99 blocks
==1==      possibly lost: 0 bytes in 0 blocks
==1==    still reachable: 0 bytes in 0 blocks
==1==         suppressed: 0 bytes in 0 blocks
==1==
==1== For lists of detected and suppressed errors, rerun with: -s
==1== ERROR SUMMARY: 1 errors from 1 contexts (suppressed: 0 from 0)
```

Example gdb usage for part1:
Note: we recommend the following command line compilation when using gdb instead of `rv make all`

```
(base) tinacheng@Tinas-Air part1 % rv g++ -std=c++20 -g -O0 -Wall knn.cpp main.cpp -o main
(base) tinacheng@Tinas-Air part1 % rv gdb ./main
GNU gdb (Ubuntu 12.1-0ubuntu1~22.04.2) 12.1
Copyright (C) 2022 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
Type "show copying" and "show warranty" for details.
This GDB was configured as "x86_64-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<https://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
    <http://www.gnu.org/software/gdb/documentation/>.

For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from ./main...
(gdb) break main
Breakpoint 1 at 0xc1aa: file main.cpp, line 130.
(gdb) run ./data/1d-1.json ./data/1d-100.json 2
Starting program: /root/main ./data/1d-1.json ./data/1d-100.json 2
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/x86_64-linux-gnu/libthread_db.so.1".

Breakpoint 1, main (argc=4, argv=0x7fffffffed68) at main.cpp:130
130     {
(gdb) next
131         if (argc != 4) {
(gdb) break runMain
Breakpoint 2 at 0x55555555f2b7: file main.cpp, line 9.
(gdb) next
137         new_argv[0] = argv[1];   // pass JSON-filename as argv[0]
(gdb) step
138         new_argv[1] = argv[2];   // pass JSON-filename as argv[1]
(gdb) print argc
$1 = 4
(gdb) continue
Continuing.

Breakpoint 2, runMain (argv=0x7fffffffec30) at main.cpp:9
9       {
(gdb) backtrace
#0  runMain (argv=0x7fffffffec30) at main.cpp:9
#1  0x0000555555560239 in main (argc=4, argv=0x7fffffffed68) at main.cpp:141
(gdb) break knn.cpp:42
Breakpoint 3 at 0x555555559df5: file knn.cpp, line 44.
```

Example gdb usage for part2 (you can create your own <query json> using any of the ones in `queries_emb.json`; instructions to download the file can be found in `part2/download.sh`)

Note: we recommend the following command line compilation when using gdb instead of `rv make all`

```
(base) tinacheng@Tinas-Air part2 % rv g++ -std=c++20 -g -O0 -Wall main.cpp -o main
(base) tinacheng@Tinas-Air part2 % rv gdb ./main
GNU gdb (Ubuntu 12.1-0ubuntu1~22.04.2) 12.1
Copyright (C) 2022 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
Type "show copying" and "show warranty" for details.
This GDB was configured as "x86_64-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<https://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
    <http://www.gnu.org/software/gdb/documentation/>.

For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from ./main...
(gdb) break main
Breakpoint 1 at 0x5cef: file main.cpp, line 158.
(gdb) run 384 ./data/peony.json ./data/passages1.json 2
Starting program: /root/main 384 ./data/peony.json ./data/passages1.json 2
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/x86_64-linux-gnu/libthread_db.so.1".

Breakpoint 1, main (argc=5, argv=0x7fffffffed58) at main.cpp:158
158     {
(gdb) next
159         if (argc != 5) {
(gdb) next
165         size_t dim = std::stoi(argv[1]);
```