PatternFinder is a tool that finds non-overlapping or overlapping patterns in any input sequence.

Pattern Finder Input Parameters:

USAGE:

PatternDetective.exe [

-help

/?

-f [filename]

-min [minimum pattern length]

-max [maximum pattern length]

-c

-threads [number of threads]

-mem [memory limit in MB]

-ram

-hd

-i [minimum times a pattern has to occur in order to keep track of it]

-v [verbosity level]

-n

-o

-his [hd files]

-lr [low range pattern search]

-hr [high range pattern search]

-pnoname

-plevel [level to show detailed output]

-ptop [n most common patterns indicated by -plevel]

Options:

-help Displays this help page

/? Displays this help page

-f [string] Sets file name to be processed

-min [unsigned long] Sets the minimum pattern length to be searched

-max [unsigned long] Sets the maximum pattern length to be searched

-c Finds the best threading scheme for computer

-threads [unsigned int] Sets thread count to be used

-mem [unsigned long] Sets the maximum RAM memory that can be used in MB

-ram Forces program to use only RAM

-hd Forces program to use Hard Disk based on -mem

-i [unsigned long] Minimum occurrences to consider a pattern (Default occurences will be 2)

-v [unsigned long] Verbosity level, turn logging and pattern generation on or off with 1 or 0

-n Only look for patterns that do not overlap each other

-o Only look for patterns that overlap each other

-his HD processing file history keeps or removes files level by level with a 1 or a 0

-lr Search for patterns that begin with the value lr to 255 if hr isn't set, otherwise lr to hr range

-hr Search for patterns that begin with the value hr to 0 if lr isn't set, otherwise lr to hr range

-pnoname Do not print pattern string data

-plevel Sets the level the user wants to see detailed output for

-ptop Display the top N most common patterns in detail for the level indicated by -plevel

How to build PatternFinder:

PREREQS:

cmake version 2.5 or higher

c++11 compatible compiler

python 2.7 to run parallel serial jobs

Visual Studio 2012 or 2015 for building with Windows

download repo using https address https://github.com/octopusprime314/PatternDetective.git

or use git and ssh using address git@github.com:octopusprime314/PatternDetective.git

BUILD INSTRUCTIONS:

!!!!!!!!!!!ALWAYS BUILD IN RELEASE UNLESS DEBUGGING CODE!!!!!!!!!!

Linux:

create a build folder at root directory

cd into build

cmake -D CMAKE_BUILD_TYPE=Release -G "Unix Makefiles" ..

cmake --build .

Windows:

create a build folder at root directory

cd into build

cmake -G "Visual Studio 11 2012 Win64" .. OR cmake -G "visual Studio 14 2015 Win64" ..

cmake --build . --config Release

How to run PatternFinder as a standalone executable:

LOCATION OF FILES TO BE PROCESSED:

Place your file to be processed in the Database/Data folder

EXAMPLE USES OF PATTERNFINDER:

1) ./PatternFinder -f Database -v 1 -threads 4 -ram

Pattern searches all files recursively in directory using DRAM with 4 threads

2) ./PatternFinder -f TaleOfTwoCities.txt -v 1 -c -ram

Finds the most optimal thread usage for processing a file

3) ./PatternFinder -f TaleOfTwoCities.txt -v 1

Processes file using memory prediction per level for HD or DRAM processing

4) ./PatternFinder -f TaleOfTwoCities.txt -v 1 -mem 1000

Processes file using memory prediction per level for HD or DRAM processing with a memory constraint of 1 GB

5) ./PatternFinder -f TaleOfTwoCities.txt -min 5 -max 100

Finds patterns of length 5 to 100 and then terminates processing

6) ./PatternFinder -f TaleOfTwoCities.txt -n

Processes file using non overlapping processing

7) ./PatternFinder -f TaleOfTwoCities.txt -v 1 -hd

Processes file using the hard disk only.

8) ./PatternFinder -f TaleOfTwoCities.txt -i 10

Processes patterns that occur at least 10 times or more. Default is 2.

How to run PatternFinder Python Scripts:

PYTHON RUN EXAMPLES:

1) python splitFileForProcessing.py [file path] [number of chunks]

Use splitFileForProcessing.py Python script to split files into chunks and run multiple instances of PatternFinder on those chunks

Ex. python splitFileForProcessing.py ~/Github/PatternDetective/Database/Data/TaleOfTwoCities.txt 4

equally splits up TaleOfTwoCities.txt into 4 files and 4 instances of PatternFinder get dispatched each processing one of the split up files.

2) python segmentRootProcessing.py [file path] [number of jobs] [threads per job]

Use segmentRootProcessing.py Python script splits up PatternFinder jobs to search for patterns starting with a certain value

Ex. python segmentedRootProcessing.py ../Database/Data/Boosh.avi 4 4

Dispatches 4 processes equipped with 4 threads each. Each PatternFinder will only look for patterns starting with the byte

representation of 0-63, 64-127, 128-191, 192-255.

PatternFinder Input Files:

PatternFinder accepts any type of input file because it processes at the byte level.

PatternFinder Output Files:

Nine outputs are available. One is a general logger using ascii text format, another is the Output file which generates patterns based on -pnoname, -plevel, ptop and the remaining seven are Comma Separated Variable files used for post processing in MATLAB.

1) Logger file: records general information including the most common patterns, number of time a pattern occurs and the pattern's coverage at every level until the last pattern is found. Simple text file.
2) Output file: generates patterns based on -pnoname, -plevel and -ptop
3) Collective Pattern Data file: records each level's most common pattern and number of times the pattern occurs in CSV format.
4) File Processing Time: records each file's processing time in CSV format. Used for processing large data sets with many files.
5) File Coverage: records the most common pattern's coverage of the file in CSV format.
6) File Size Processing Time file: records each file's processing time and corresponding size in CSV format. Used primarily to isolate files in a large dataset that contain large patterns.
7) Thread Throughput: records the processing throughput improvement while incrementing the number of processing threads in CSV format. Typically used with -c option which tests threads in multiples of 2 starting at 1 until the number of cores on the machine has been met.
8) Thread Speed: records the processing time taken while incrementing the number of processing threads in CSV format. Typically used with -c option which tests threads in multiples of 2 starting at 1 until the number of cores on the machine has been met.

Output file contents is pattern string, number of instances, occurrence, average distance and location:

```
./PatternFinder.exe -f TaleOfTwoCities.txt -plevel 2 -ptop 4 -threads 1 -ram
```

Level 1

unique patterns = 83, average occurrence frequency = 9490.39, frequency of top pattern: 129157

Level 2

unique patterns = 1401, average occurrence frequency = 562.024, frequency of top pattern: 21032

1. pattern = e , instances = 21032, coverage = 5.34006%, average pattern distance = 37.4536, first occurrence index = 4

2. pattern = t, instances = 18017, coverage = 4.57454%, average pattern distance = 43.7134, first occurrence index = 75

3. pattern = he, instances = 16814, coverage = 4.2691%, average pattern distance = 46.8478, first occurrence index = 3

4. pattern = th, instances = 16713, coverage = 4.24346%, average pattern distance = 47.1234, first occurrence index = 91

Level 3

unique patterns = 7934, average occurrence frequency = 98.9629, frequency of top pattern: 12204

Level 4

unique patterns = 26273, average occurrence frequency = 29.5211, frequency of top pattern: 8907

Level 5

unique patterns = 57380, average occurrence frequency = 13.0067, frequency of top pattern: 6427

Level 6

unique patterns = 86748, average occurrence frequency = 7.87542, frequency of top pattern: 2153

Level 7

unique patterns = 103739, average occurrence frequency = 5.72812, frequency of top pattern: 974

Level 8

unique patterns = 108726, average occurrence frequency = 4.53085, frequency of top pattern: 780

Level 9

unique patterns = 102229, average occurrence frequency = 3.80874, frequency of top pattern: 339

Level 10

unique patterns = 88663, average occurrence frequency = 3.34268, frequency of top pattern: 276

PatternFinder post processing scripts using the seven available CSV outputs with MATLAB:

1) DRAM versus HD Processing Speeds->DRAMtoHDProcessingLiminationSpeeds.m
2) DRAM versus HD Performance->DRAMVsHardDiskPerformance.m
3) Most Common Pattern versus Coverage->MostCommonPatternLengthVsCoveragePercentage.m
4) Overlapping versus Non Overlapping Comparison->Overlapping_NonOverlappingComparison.m
5) Overlapping versus Non Overlapping File Speeds->OverlappingVsNonOverlappingFileSpeeds.m
6) Process Time versus File Size->ProcessTimeVsFileSize.m