TRed Version `v4` Documentation

1 Where can I find the Source Code?

1.1 GitHub Repository

- TRed Version `v4.0` (multi-threaded version) is at
  https://github.com/mary060196/CISC5001_Research_Project_Implementing_TRed_Efficiently
  tree/multi-threaded
- TRed Version `v4.1-alpha` (suffix arrays version) is at
  https://github.com/mary060196/CISC5001_Research_Project_Implementing_TRed_Efficiently
  tree/suffix-array

1.2 TRed Website

Releases of these programs can also be downloaded via the following website:
https://mary060196.github.io/CISC5001_Research_Project_Implementing_TRed_Efficiently/

2 Source Code Layout

The TRed Version `v4.0` program has the layout in Figure 1.

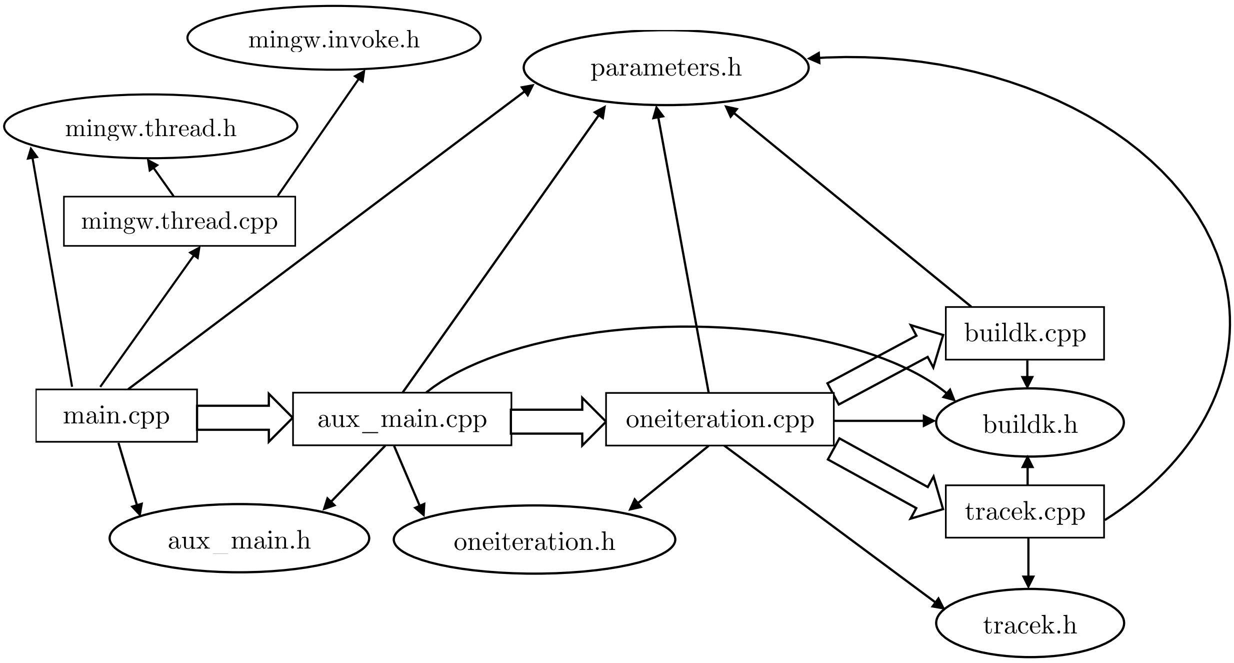

Figure 1: TRed Version `v4.0` Layout

The TRed Version `v4.1-alpha` program has the layout in Figure 2.

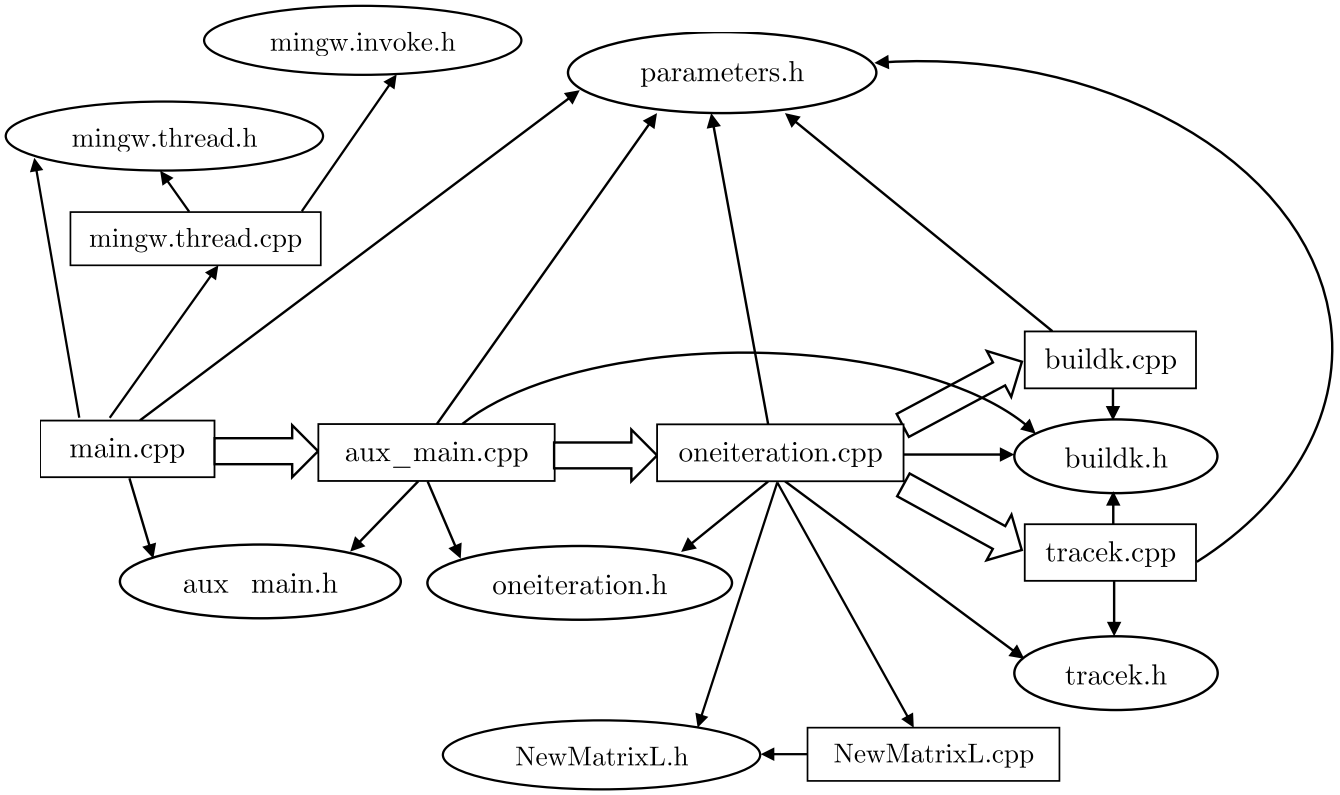

Figure 2: TRed Version `v4.1-alpha` Layout

3 File Content + Description

Below is a short description of what each `.cpp` file accomplishes, along with lists of defined functions, global variables, and constants:

- **main.cpp**:
    1. Retrieval of input sequence and intermediary filenames from either the command line argument list `argv` or, if not provided, by asking the user to enter them.
    2. Declaration of variables, including a structure of type `thread_info`, where thread-specific information will be given to each thread.
    3. Access to the input sequence file, scanning of the file, and storage of the entire sequence (without whitespaces or comments) into a buffer.
    4. Memory allocation for the `init_increments` and `numarray` integer pointers, and population of the allocated arrays with values.
    5. Thread creation, work allocation, and later thread joining.
    6. Appending of all temporary output files created by the threads (in the same directory as the binary) into the output file, and removal of the temporary files thereafter.
    7. Measurement and output to terminal /command line application of how much time the program executed (in milliseconds.)
    8. Freeing of allocated memory.

9. Functions:

   `int main (int argc, char *argv[])`

   Used external variables:

   - `int *init_increments;` – External variable declared in `buildk.h` and used in `buildk.cpp`
   - `int *numarray` – External variable declared in `aux_main.h` and used in `aux_main.cpp`
   - `unsigned long long wholestr_len = 0;` – External variable declared in `aux_main.h` and used in `aux_main.cpp`

- **mingw.thread.cpp**:
  1. A threading library called `mingw-std-threads` by Mega retrieved from: https://github.com/meganz/mingw-std-threads
  2. On Windows platforms, the Thread C++11 library is not fully supported, which prevents the program from using the `#include <thread>` preprocessor directive. This is where the `mingw.thread.cpp` code is used as a threading mechanism.
  3. This file, and the other `mingw.thread` files provided included in the TRed programs remain unmodified.
  4. The `main` function in `main.cpp` uses some of the `thread` object constructors and the `join` function. Since the `mingw-std-threads` library has an identical functioning as the Thread C++11 object, please refer to the description at

     http://www.cplusplus.com/reference/thread/thread/

     for documentation of the Thread C++11 library.

- **aux_main.cpp**:
  1. The `threads_func` and `threads_func_last` functions are those which threads call.
  2. Retrieval of information about what a particular thread should accomplish.
  3. Declaration of variables and memory allocation.
  4. Implementation of the Main-Lorentz algorithm upon the sequence buffer portion, a pointer to whom was passed in a structure `thread_info` to the function. The algorithm, implemented here in an iterative form, consists of calls to the `OneIteration` function defined in `oneiteration.cpp`.
  5. Freeing of allocated memory.
  6. The function `read_file`, which is called by `main`, performs the reading of the sequence input file into the buffer and measurement of how long the clear sequence (without comments or whitespaces) is.
  7. Functions:

     - `void threads_func (thread_info *my_info)`
     - `void threads_func_last (thread_info *my_info)`
     - `void read_file (char* s, unsigned long long &numRead, FILE* stream)`

     Global Variables (from `aux_main.h`):

     - `const long DOUBLE_PERIOD = 2*MAX_PERIOD;` – Twice the maximum period (see `parameters.h` for definition)

- `const long QUAD_PERIOD = 4*MAX_PERIOD;` – Four times the maximum period (see `parameters.h` for definition)
- `const long D_ERRORS_PLUS_1 = 2*MAX_ERRORS + 1;` – Twice the maximum allowed errors + 1
- `const unsigned short THREAD_NUM = thread::hardware_concurrency();` – Number of threads used in the program (besides the main thread)

External Variables (from `aux_main.h`):

- `extern int *numarray;` – Used for in Main-Lorentz algorithm
- `extern unsigned long long wholestr_len;` – Measures the length of the clear sequence in the buffer.

- **oneiteration.cpp**:
  1. Variable declaration and definition.
  2. Construction of the `KxK` matrix (via the either the conventional `KxK` construction algorithm in `buildk.cpp` or the suffix trees algorithm in `NewMatrixL.cpp` in version `v4.1-alpha` of the TRed program.)
  3. Analysis of whether the information in the `KxK` matrix constitutes a potential tandem repeat.
  4. If so, tracing of the exact tandem repeat pattern by calling `tracealign` and `traceforward` functions defined in `tracek.cpp`.
  5. Functions:

     In TRed Version `v4.0`:

     ```
     void OneIteration(char* str, long length, int ** matrix_forward,
     int ** matrix_reverse, Entry* down_errors, Entry* right_errors, struct
     LastReportedRepeat left, struct LastReportedRepeat right, long int offset,
     int*upper, int *lower, char *forward_pattern1, char *forward_pattern2,
     FILE* outfile)
     ```

     In TRed Version `v4.1-alpha`:

     ```
     void OneIteration(char* str, long length, int ** matrix_forward,
     int ** matrix_reverse, Entry* down_errors, Entry* right_errors, struct
     LastReportedRepeat left, struct LastReportedRepeat right, long int offset,
     int *upper, int *lower, char *forward_pattern1, char *forward_pattern2,
     FILE* outfile, char *joinedstr[2], suffix **suffixes[2], int **L, int *lcp[4],
     int *sizeToIndex[4], int *sizeToIndexFixed[4], int *smallest01[4][2], bool
     *isSuffixConsidered[4])
     ```

- **buildk.cpp**
  1. Construction of the `KxK` matrix by building portions of the `D` Edit Distance matrix.
  2. The file defines two functions, one of which is used for the "forward" string comparison case and the other for the "reverse" (backward) string comparison case, based on the Main-Lorentz algorithm.
  3. At each point, only 2 rows of the `D` Edit Distance matrix are created / stored.
  4. Functions:

- `void buildmatrixforward (int **m, int *upper, int *lower, const char *s1, const int len_s1, const char *s2, const int len_s2, const int limit)`
- `void buildmatrixbackward (int **m, int *upper, int *lower, const char *s1, const int len_s1, const char *s2, const int len_s2, const int limit)`

External Variables:

`extern int *init_increments;` – Used for initialization of the first 2 rows of the `D` Edit Distance matrix.

Macros:

`UNKNOWN 'N'` – Denotes the "wildcard" character in the provided sequence.

- **tracek.cpp**:
  1. The two functions defined in this file trace the `KxK` matrix to extract the positions where the tandem repeats occur, and print those positions into the output file.
  2. As mentioned, one function is used for the "forward" case, whereas the other is used for the "backward" case.
  3. Functions:
     - `void tracealign (int** m, char* s1, char* s2, int lowerRow, int s1pos, int s2pos, long &pos1, long &pos2, bool &state, FILE* outfile)`
     - `void traceforward (int** m, char* s1, char* s2, int upperRow, int s1pos, int s2pos, long pos1, long pos2, bool state, char forward_pattern1, char *forward_pattern2, FILE* outfile)`

  Used Macros:

  `UNKNOWN` – Defined in `buildk.h` (see above)

- **NewMatrixL.cpp** (only in TRed Version `v4.1-alpha`)
  1. Construction of a suffix array and other related data structures (such as the `lcp` – longest common prefix array).
  2. Construction of the `L` matrix described in the Landau-Vishkin algorithm [1], and from there, construction of the `KxK` matrix.
  3. Transition from one `L` matrix to another via the `stringIncrement` function, to be able to generate related `KxK` matrices.
  4. Functions:
     - `void prepareSuffixArrays (char *str, const long long str_len, const char* text, const int text_len, const char* pattern, const int pattern_len, const int curr_text_len, int *lcp, int *sizeToIndexFixed, int *sizeToIndex, suffix **suffixes, bool *isSuffixConsidered, int *smallest01[], bool reversed)`
     - `void stringIncrement (char *str, const long long str_len, const int curr_text_len, int *lcp, int *sizeToIndexFixed, int *sizeToIndex, suffix **suffixes, bool *isSuffixConsidered, int *smallest01[])`
     - `void buildMatrixL (int **matrix, int **L, const long long str_len, int *lcp, int *sizeToIndex, int *smallest01[], const int text_len, const int pattern_len, int upLim, int lowLim)`

References

[1] Landau, G. M., & Vishkin, U. (1989). Fast parallel and serial approximate string matching. *Journal of algorithms, 10(2), 157-169.* doi:10.1016/0196-6774(89)90010-2