

DraFF_REVISION_1 - in search for exhaustive Dragonic Fuzzy Finding...

A review by sanmayce@sanmayce.com, 2024-Dec-31;

With ChatGPT's help, here comes a console tool reading 4+GB files (not necessarily textual) and dumping all hits matching the given Levenshtein Distance, using all the CPU threads.



DraFF -

Dragonic Fuzzy Finding

DraFF_REVISION_1 - in search for exhaustive Dragonic Fuzzy Finding...

A review by sanmayce@sanmayce.com, 2024-Dec-31;

With ChatGPT's help, here comes a console tool reading 4+GB files (not necessarily textual) and dumping all hits matching the given Levenshtein Distance, using all the CPU threads.



DraFF -

Dragonic Fuzzy Finding

The source code:

```
// gcc -O3 -mssse4.2 -o parallel_edit_distance_RAM parallel_edit_distance_RAM.c -fopenmp -D_FILE_OFFSET_BITS=64 -DMaxThreads
// ./parallel_edit_distance_RAM large_file.txt "target_string" 3
```

```
// [sanmayce@djudjeto7 ~]$ perf stat -d ./parallel_edit_distance_RAM masakari.wrd "and then" 2 | more
// Searching using 4 threads for pattern 'and then' ...
// Thread 0: Match found at offset 180928: 'antiithen' (Edit Distance: 2)
// Thread 1: Match found at offset 335539: 'and ben' (Edit Distance: 2)
// Thread 1: Match found at offset 1260482: 'and fen' (Edit Distance: 2)
// Thread 0: Match found at offset 1477673: 'and han' (Edit Distance: 2)
// Thread 1: Match found at offset 1478308: 'and han' (Edit Distance: 2)
// Thread 0: Match found at offset 1480203: 'and han' (Edit Distance: 2)
// Thread 1: Match found at offset 1493662: 'and hea' (Edit Distance: 2)
// Thread 0: Match found at offset 1494385: 'and hea' (Edit Distance: 2)
// Thread 3: Match found at offset 1495419: 'and hea' (Edit Distance: 2)
```

```
/*
```

```

DRAFT -
```

```

Drafting a Word Finding
```

```
*/
```

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <omp.h>
```

```
int computeEditDistance(const char* str1, const char* str2);
```

```
void makePrintable(char* str, size_t len) {
    for (size_t i = 0; i < len; i++) {
        if (str[i] == '\0' || str[i] == '\t' || str[i] == '\r' || str[i] == '\n') {
            str[i] = ' '; // Replace with SPC (ASCII 32)
        }
    }
}
```

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

```

    if (argc != 4) {
        fprintf(stderr, "Usage: %s <filename> <string2> <minimumEditDistance>\n", argv[0]);
        return EXIT_FAILURE;
    }

//omp_set_num_threads(1);
#ifdef MaxThreads
omp_set_num_threads(omp_get_max_threads());
#endif

    const char* filename = argv[1];
    const char* string2 = argv[2];
    int minimumEditDistance = atoi(argv[3]);
    int minimumEditDistanceEXTRA = 0;
    size_t len2 = strlen(string2);

    // Generate output filename
    char outputFilename[1024];
    snprintf(outputFilename, sizeof(outputFilename), "%s.hits", filename);

    // Open the output file for writing
    FILE* outputFile = fopen(outputFilename, "w");
    if (!outputFile) {
        perror("Error opening output file");
        return EXIT_FAILURE;
    }

    // Open the file
    FILE* file = fopen(filename, "rb");
    if (!file) {
        perror("Error opening file");
        return EXIT_FAILURE;
    }

    // Get the file size
    fseek(file, 0, SEEK_END);
    long file_size = ftell(file);
    rewind(file);

    if (file_size <= 0) {
        fprintf(stderr, "File is empty or inaccessible.\n");
        fclose(file);
        return EXIT_FAILURE;
    }

    // Allocate memory for the file content
    char* file_content = (char*)malloc(file_size + 1); // +1 for null-terminator
    if (!file_content) {

```

```

        fprintf(stderr, "Memory allocation failed.\n");
        fclose(file);
        return EXIT_FAILURE;
    }

    // Read the entire file into memory
    fread(file_content, 1, file_size, file);
    fclose(file);
    file_content[file_size] = '\0'; // Null-terminate for safety

printf("Searching using %d threads for pattern '%s' ...\n",
omp_get_max_threads(), string2);

    // Process the file content with OpenMP
    #pragma omp parallel for schedule(dynamic)
    for (long i = 0; i <= file_size - len2 - minimumEditDistanceEXTRA; i++) {
        char* substring = strdup(&file_content[i], len2 + minimumEditDistanceEXTRA); // ASCII 000 content possible
        //if (!substring) {
        //    fprintf(stderr, "Memory allocation failed in thread %d.\n", omp_get_thread_num());
        //    continue;
        //}
makePrintable(substring, len2 + minimumEditDistanceEXTRA);

        int editDistance = computeEditDistance(substring, string2);
        if (editDistance <= minimumEditDistance) {
            #pragma omp critical
            {
                printf("Thread %d: Match found at offset %ld: '%s' (Edit Distance: %d)\n",
                    omp_get_thread_num(), i, substring, editDistance);
                fprintf(outputFile, "%s\n", substring);
            }
        }

        free(substring);
    }

    // Clean up
    free(file_content);
    fclose(outputFile);

    return EXIT_SUCCESS;
}

// Function to calculate the Edit Distance between two strings
int computeEditDistance(const char* str1, const char* str2) {
    size_t len1 = strlen(str1);
    size_t len2 = strlen(str2);

```

```

// Allocate DP table
int dp[len1 + 1][len2 + 1];

// Initialize the DP table
for (size_t i = 0; i <= len1; i++) {
    dp[i][0] = i;
}
for (size_t j = 0; j <= len2; j++) {
    dp[0][j] = j;
}

// Fill the DP table
for (size_t i = 1; i <= len1; i++) {
    for (size_t j = 1; j <= len2; j++) {
        int cost = (str1[i - 1] == str2[j - 1]) ? 0 : 1;
        dp[i][j] = dp[i - 1][j - 1] + cost; // Substitution
        if (dp[i][j - 1] + 1 < dp[i][j]) dp[i][j] = dp[i][j - 1] + 1; // Insertion
        if (dp[i - 1][j] + 1 < dp[i][j]) dp[i][j] = dp[i - 1][j] + 1; // Deletion
    }
}

// Return the computed Edit Distance
return dp[len1][len2];
}

```