2024-2-SP-A3-2020312914-백민호

작성자 : 백민호
학과 : 소프트웨어학과
학번 : 2020312914
작성일 : 2024-12-7
과목명 : 시스템 프로그램
학수번호 : SWE2001_42
프로젝트명 : Bigram Analyzer

Target Program 구현 및 테스트/검증

프로그램 개요

Bigram Analyzer는 대규모 텍스트 파일을 처리하여 연속된 단어 N개의 쌍의 빈도를 분석하는 프로그램입니다. 교안 SP05-3에서 소개한 "Bigram Analyzer"를 사용했고 input data 역시 William Shakespeare의 텍스트 파일을 사용했습니다. 해당 Program은 각 단어들을 문자열 처리와 함께 hash table을 활용한 데이터 저장 및 sorting을 진행했습니다.

이 프로그램은 알고리즘이 아닌 코드 성능 분석 및 최적화를 두고 설계되었습니다. 따라서 안정적으로 잘 동작하는 bubble sort를 사용하였으며, 프로파일링 및 최적화를 통해 성능 개선의 여지를 남겨두었습니다. 이를 통해 최적화 과정에서 학습과 개선점을 다양한 분야에 쓸 수 있기 위한 시야를 얻을 수 있을 것이라 기대했습니다.

주요 기능

1. Bigram Analyze
    - input text file에서 연속된 단어 쌍을 추출하여 빈도를 계산합니다.
    - hash table을 사용해 빠른 data insert과 검색을 구현합니다.
2. Sort
    - Bigram data를 빈도 순으로 정렬하여 가장 자주 나타나는 단어 쌍을 확인할 수 있습니다.
    - Sorting에는 Bubble Sort를 사용했습니다.
3. File input/output
    - 대규모 텍스트 파일을 읽어 데이터를 처리하며, 결과는 빈도 순으로 정렬된 Bigram data를 output.txt로 저장합니다.

테스트 및 검증

- 실행 환경

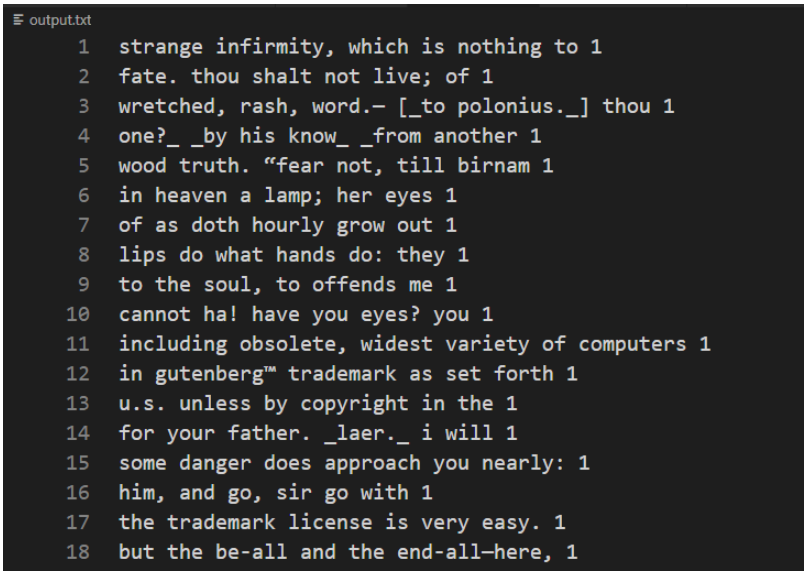
```
gcc --version
gcc (Ubuntu 11.4.0-1ubuntu1~22.04) 11.4.0

wsl --version
Wsman Shell commandLine, version 0.2.1

lsb_release -a
Description:    Ubuntu 22.04.4 LTS

gprof --version
GNU gprof (GNU Binutils for Ubuntu) 2.38

// HW 세팅 환경
11th Gen Intel(R) Core(TM) i7-11800H @ 2.30GHz
NVIDIA GeForce RTX 3060
16GB
```

- 프로그램 컴파일 및 실행 방법

```
gcc -Wall -Og -o 2024-2-SP-A3-Initial-2020312914-debug 2024-2-SP-A3-Initial-
2020312914.c

./2024-2-SP-A3-Initial-2020312914-debug <filename> <N(default : 2)>
./2024-2-SP-A3-Initial-2020312914-debug input.txt
./2024-2-SP-A3-Initial-2020312914-debug input.txt 3
```

- 입출력
  stdout : 각 주요 함수 별 소요 시간, sorting의 상위 5개 Bigram, file과 hash table count
  fileout : hash table의 각 Bigram count

output.txt sample



```
output.txt
1  strange infirmity, which is nothing to 1
2  fate. thou shalt not live; of 1
3  wretched, rash, word.– [_to polonius._] thou 1
4  one?_ _by his know_ _from another 1
5  wood truth. “fear not, till birnam 1
6  in heaven a lamp; her eyes 1
7  of as doth hourly grow out 1
8  lips do what hands do: they 1
9  to the soul, to offends me 1
10 cannot ha! have you eyes? you 1
11 including obsolete, widest variety of computers 1
12 in gutenberg™ trademark as set forth 1
13 u.s. unless by copyright in the 1
14 for your father. _laer._ i will 1
15 some danger does approach you nearly: 1
16 him, and go, sir go with 1
17 the trademark license is very easy. 1
18 but the be-all and the end-all–here, 1
```

stdout sample(./2024-2-SP-A3-Initial-2020312914-debug input.txt 6)

```
start
init_hash_table time: 0.020000 seconds
get_N_gram time: 0.800000 seconds
start sorting
project gutenberg literary archive foundation the 11
literary archive foundation the project gutenberg 11
archive foundation the project gutenberg literary 11
foundation the project gutenberg literary archive 11
not come again?_ _and will he 7
sort_N_gram time: 229.580000 seconds
==================================
file size: 711681
word count: 120329
N : 6
bucket count: 117356
write_to_file time: 0.930000 seconds
time: 231.420000 seconds
```

Profiling 및 최적화

1차 profiling 및 결과 분석

- Compiling and linkning for profiling

```
gcc -Og -pg 2024-2-SP-A3-Initial-2020312914.c -o init
./init input.txt 6
gprof init gmon.out > profile.txt
```

- Flat profile

```
  %   cumulative   self              self     total
 time   seconds   seconds    calls   s/call   s/call  name
 99.63    243.66   243.66        1   243.66   244.32  bubble_sort
  0.27    244.32     0.66 156864726     0.00     0.00  swap
  0.03    244.40     0.08        1     0.08     0.08  get_bucket_count
  0.02    244.46     0.06        1     0.06   244.38  sort_N_gram
  0.02    244.51     0.05   120324     0.00     0.00  insert_N_gram
  0.01    244.53     0.02        1     0.02     0.02  free_N_gram
  0.01    244.55     0.02        1     0.02     0.02  write_to_file
  0.00    244.56     0.01   120324     0.00     0.00  hash
  0.00    244.56     0.00   690482     0.00     0.00  get_string_length
  0.00    244.56     0.00   120329     0.00     0.00  to_lower
  0.00    244.56     0.00        9     0.00     0.00  get_time
  0.00    244.56     0.00        1     0.00     0.00  count_word
  0.00    244.56     0.00        1     0.00     0.06  get_N_gram
  0.00    244.56     0.00        1     0.00     0.00  get_file_size
  0.00    244.56     0.00        1     0.00     0.00  init_hash_table
  0.00    244.56     0.00        1     0.00     0.08  print_N_gram
```

- Call graph

```
index % time    self  children    called     name
                                                 <spontaneous>
[1]    100.0    0.00  244.56                 main [1]
                0.06  244.32       1/1           sort_N_gram [2]
                0.00    0.08       1/1           print_N_gram [6]
                0.00    0.06       1/1           get_N_gram [8]
                0.02    0.00       1/1           write_to_file [10]
                0.02    0.00       1/1           free_N_gram [9]
                0.00    0.00       9/9           get_time [14]
                0.00    0.00       1/1           get_file_size [16]
                0.00    0.00       1/1           init_hash_table [17]
                0.00    0.00       1/1           count_word [15]
```

```
-----------------------------------------------
                0.06  244.32       1/1           main [1]
[2]     99.9    0.06  244.32       1         sort_N_gram [2]
              243.66    0.66       1/1           bubble_sort [3]
-----------------------------------------------
              243.66    0.66       1/1           sort_N_gram [2]
[3]     99.9  243.66    0.66       1         bubble_sort [3]
                0.66    0.00 156864726/156864726     swap [4]
-----------------------------------------------
                0.66    0.00 156864726/156864726     bubble_sort [3]
[4]      0.3    0.66    0.00 156864726         swap [4]
-----------------------------------------------
                0.08    0.00       1/1           print_N_gram [6]
[5]      0.0    0.08    0.00       1         get_bucket_count [5]
-----------------------------------------------
                0.00    0.08       1/1           main [1]
[6]      0.0    0.00    0.08       1         print_N_gram [6]
                0.08    0.00       1/1           get_bucket_count [5]
-----------------------------------------------
                0.05    0.01  120324/120324      get_N_gram [8]
[7]      0.0    0.05    0.01  120324         insert_N_gram [7]
                0.01    0.00  120324/120324      hash [11]
-----------------------------------------------
                0.00    0.06       1/1           main [1]
[8]      0.0    0.00    0.06       1         get_N_gram [8]
                0.05    0.01  120324/120324      insert_N_gram [7]
                0.00    0.00  120329/120329      to_lower [13]
-----------------------------------------------
                0.02    0.00       1/1           main [1]
[9]      0.0    0.02    0.00       1         free_N_gram [9]
-----------------------------------------------
                0.02    0.00       1/1           main [1]
[10]     0.0    0.02    0.00       1         write_to_file [10]
-----------------------------------------------
                0.01    0.00  120324/120324      insert_N_gram [7]
[11]     0.0    0.01    0.00  120324         hash [11]
-----------------------------------------------
                0.00    0.00  690482/690482      to_lower [13]
[12]     0.0    0.00    0.00  690482         get_string_length [12]
-----------------------------------------------
```

```
                0.00    0.00  120329/120329      get_N_gram [8]
[13]     0.0    0.00    0.00  120329         to_lower [13]
                0.00    0.00  690482/690482      get_string_length [12]
-----------------------------------------------
                0.00    0.00       9/9           main [1]
[14]     0.0    0.00    0.00       9         get_time [14]
-----------------------------------------------
                0.00    0.00       1/1           main [1]
[15]     0.0    0.00    0.00       1         count_word [15]
-----------------------------------------------
                0.00    0.00       1/1           main [1]
[16]     0.0    0.00    0.00       1         get_file_size [16]
-----------------------------------------------
                0.00    0.00       1/1           main [1]
[17]     0.0    0.00    0.00       1         init_hash_table [17]
-----------------------------------------------
```

- 분석
  해당 결과를 보면 `bubble_sort`가 전체 실행시간의 99.63%를 차지하여 성능 저하의 주요 원인임을 알 수 있다. 그 중 `swap`의 호출 횟수가 `sort_N_gram`에서 `bubble_sort` 호출이 가장 많고, 나머지 함수들은 실행 시간의 1% 미만을 차지하고 있기 때문에 시간 소모에 크게 기여하지 않는다. 따라서 우선 호출 횟수가 가장 많은 `swap`을 최적화하기로 정했다.
- Inline Swap
  가장 자주 사용되는 `swap`함수를 inline 함수로 매크로화 시켜 사용하면 함수 호출에 시간을 더 단축시킬 수 있다는 사실을 알게 되었다. 이에 대한 이유로는 다음과 같다 :
  - 함수 호출 오버헤드 제거
    일반 함수의 경우 caller의 상태와 parameter를 stack에 쌓고 return address를 복구해야하는데 inline 함수는 이러한 과정이 생략되게 된다. 이는 컴파일러가 함수를 호출하는 대신 함수 본문 코드에 호출 위치에 삽입하도록 요청하는 함수로 약간 `a + b`를 함수 `add(a, b)`를 쓰지 않고 사용하던 함수에 `a + b`를 넣는 것과 비슷한 맥락이다.
  - cache locality 향상
    함수 코드가 호출 위치에 삽입되면서, 코드와 가까운 메모리 위치에 있게 되어 hit 될 확률이 높다고 한다.
  - 컴파일러 최적화 가능성 증가
    inline 된 코드는 호출 위치에 포함되서 컴파일러가 더 많은 최적화를 적용할 수 있다고 하는데 이 부분에 대해서는 좀 더 공부할 필요성이 있다.
- 결과
  기존의 243.66만큼 소모하던 `sort_N_gram`이 234.13초로 줄어들게 되었다. 또한 gprof의 flat profile과 call graph에서도 swap 항목이 사라진 모습을 볼 수 있었다. 보통 inline 함수의

경우 컴파일러가 inline 함수화 가능한 경우, 자동으로 inline해주는 기능들이 있는 일부 컴파일러가 있다고 들었는데 해당 컴파일은 그부분을 건드리지는 않는 것 같았다.

2차 profiling 및 결과 분석

- Compiling and linkning for profiling

```
gcc -Og -pg 2024-2-SP-A3-inline-2020312914.c -o inline
./inline input.txt 6
gprof inline gmon.out > profile.txt
```

- flat profile(일부분)

```
  %   cumulative   self              self     total
 time   seconds   seconds    calls   s/call   s/call  name
 99.94    234.13   234.13        1   234.13   234.13  bubble_sort
  0.03    234.19     0.06        1     0.06     0.06  get_bucket_count
  0.02    234.23     0.04        1     0.04   234.17  sort_N_gram
  0.00    234.26     0.01        1     0.01     0.01  free_N_gram
  0.00    234.27     0.01        1     0.01     0.01  write_to_file
  0.00    234.27     0.00        1     0.00     0.02  get_N_gram
```

- call graph(일부분)

```
index % time    self  children    called     name
                                                 <spontaneous>
[1]    100.0    0.00  234.27                 main [1]
                0.04  234.13       1/1           sort_N_gram [2]
                0.00    0.06       1/1           print_N_gram [5]
                0.00    0.02       1/1           get_N_gram [7]
                0.01    0.00       1/1           write_to_file [9]
                0.01    0.00       1/1           free_N_gram [8]
                0.00    0.00       9/9           get_time [13]
                0.00    0.00       1/1           init_hash_table [16]
                0.00    0.00       1/1           get_file_size [15]
                0.00    0.00       1/1           count_word [14]
-----------------------------------------------
                0.04  234.13       1/1           main [1]
[2]    100.0    0.04  234.13       1         sort_N_gram [2]
              234.13    0.00       1/1           bubble_sort [3]
```

```
-----------------------------------------------
               234.13    0.00       1/1           sort_N_gram [2]
[3]     99.9   234.13    0.00       1         bubble_sort [3]
-----------------------------------------------
                 0.02    0.00  120324/120324      get_N_gram [7]
[6]      0.0     0.02    0.00  120324         insert_N_gram [6]
                 0.00    0.00  120324/120324      hash [12]
-----------------------------------------------
                 0.00    0.02       1/1           main [1]
[7]      0.0     0.00    0.02       1         get_N_gram [7]
                 0.02    0.00  120324/120324      insert_N_gram [6]
                 0.00    0.00  120329/120329      to_lower [11]
-----------------------------------------------
                 0.00    0.00  690482/690482      to_lower [11]
[10]     0.0     0.00    0.00  690482         get_string_length [10]
-----------------------------------------------
                 0.00    0.00  120329/120329      get_N_gram [7]
[11]     0.0     0.00    0.00  120329         to_lower [11]
                 0.00    0.00  690482/690482      get_string_length [10]
-----------------------------------------------
                 0.00    0.00  120324/120324      insert_N_gram [6]
[12]     0.0     0.00    0.00  120324         hash [12]
-----------------------------------------------
```

- 분석
  이전과 동일하게 `bubble_sort` 함수를 주목했다. `bubble_sort` 에서는 특히나 반복문과 그 내부에서 조건문들이 있기 때문에 loop unrolling이나 branch prediction 같은 최적화 기법들을 쓸 수 있을 것이라고 생각했다.
- Loop Unroll 2
  2 by 2 unroll을 했을 때에는 그렇게 큰 효과를 보지 못했다. 교안과의 차이점을 생각해봤을 때 원인은 if문의 branch가 존재하고 병렬화한 코드인지 확실하지 않다는 점이 있었다.

```
void bubble_sort(Ngram** arr, int count) {
    for (int i = 0; i < count - 1; i++) {
        for (int j = 0; j < count - i - 1; j += 2) {
            if (arr[j]->count < arr[j + 1]->count) {
                SWAP(arr[j], arr[j + 1]);
            }
            if (j + 1 < count - i - 1 && arr[j + 1]->count < arr[j + 2]-
>count) {
```

```
                SWAP(arr[j + 1], arr[j + 2]);
            }
        }
    }
}
```

- flat profile
  for 문에 대한 코드 최적화를 진행했기 때문에 call graph의 변화는 없었다. unroll 마다 큰 차이는 없었기 때문에 unroll 2 또는 4를 선택했다.

```
  %   cumulative   self              self     total
 time   seconds   seconds    calls   s/call   s/call  name
 99.90    223.62   223.62        1   223.62   223.62  bubble_sort
```

- loop unroll 4

```
  %   cumulative   self              self     total
 time   seconds   seconds    calls   s/call   s/call  name
 99.94    219.47   219.47        1   219.47   219.47  bubble_sort
```

- loop unroll 6, 8, 10

```
 99.95    218.33   218.33        1   218.33   218.33  bubble_sort
 99.95    220.50   220.50        1   220.50   220.50  bubble_sort
 99.95    219.91   217.91        1   217.91   217.91  bubble_sort
```

- unroll을 제거한 버전에서 misprediction 방지

```
static inline void swap(Ngram** a, Ngram** b) {
    Ngram* temp = *a;
    *a = *b;
    *b = temp;
}
void bubble_sort(Ngram** arr, int count) {
    for (int i = 0; i < count - 1; i++) {
        for (int j = 0; j < count - i - 1; j++) {
            (arr[j]->count < arr[j + 1]->count) ?
            swap(&arr[j], &arr[j + 1]) : (void)0;
        }
```

```
    }
}
```

```
 time   seconds   seconds    calls   s/call   s/call  name
 99.90    223.78   223.78        1   223.78   223.78  bubble_sort
```

unroll 2에 적용해도 225초로 큰 변화를 보이지는 않았다.

3차 profiling 및 결과 분석

- call graph
  call graph를 보면 시간 점유율이 크진 않지만 과도하게 많이 함수가 실행되고 있다는 것을 확인했다. 따라서 해당 함수를 다시 확인해서 함수 호출 횟수를 줄이는 것을 목표로 정했다.

```
                0.01    0.02       1/1           main [1]
[6]      0.0    0.01    0.02       1         get_N_gram [6]
                0.01    0.01  120324/120324      insert_N_gram [7]
                0.00    0.00  120329/120329      to_lower [11]
-----------------------------------------------
                0.01    0.01  120324/120324      get_N_gram [6]
[7]      0.0    0.01    0.01  120324         insert_N_gram [7]
                0.01    0.00  120324/120324      hash [8]
-----------------------------------------------
                0.01    0.00  120324/120324      insert_N_gram [7]
[8]      0.0    0.01    0.00  120324         hash [8]
-----------------------------------------------
                0.00    0.00  690482/690482      to_lower [11]
[10]     0.0    0.00    0.00  690482         get_string_length [10]
-----------------------------------------------
                0.00    0.00  120329/120329      get_N_gram [6]
[11]     0.0    0.00    0.00  120329         to_lower [11]
                0.00    0.00  690482/690482      get_string_length [10]
```

- reducing strlen
  수업시간에 배웠던 것처럼 if문의 range를 계산하는 과정에서 함수를 호출하면 N번 함수를 호출하기 때문에 같은 값을 계속 쓰는 부분을 loop 바깥으로 빼주었다.

```
void to_lower(char* str){
    for(int i = 0; i < get_string_length(str); i++){
```

```
        if(str[i] >= 'A' && str[i] <= 'Z'){
            str[i] = str[i] - 'A' + 'a';
        }
    }
}

void to_lower2(char* str){
    uint len = get_string_length(str);
    for(int i = 0; i < len; i++){
        if(str[i] >= 'A' && str[i] <= 'Z'){
            str[i] = str[i] - 'A' + 'a';
        }
    }
}
```

실행 결과, 690482번 호출되던 `get_string_length` 함수가 다른 함수와 비슷한 120324번 호출됨을 확인할 수 있었다. 또한 해당 작업들에도 loop unrolling을 해주었지만 0.01초대로 확인이 불가능했다.

```
-----------------------------------------------
                0.00    0.07       1/1           main [1]
[6]      0.0    0.00    0.07       1         get_N_gram [6]
                0.04    0.02  120324/120324      insert_N_gram [7]
                0.01    0.00  120329/120329      to_lower [9]
-----------------------------------------------
                0.04    0.02  120324/120324      get_N_gram [6]
[7]      0.0    0.04    0.02  120324         insert_N_gram [7]
                0.02    0.00  120324/120324      hash [8]
-----------------------------------------------
                0.02    0.00  120324/120324      insert_N_gram [7]
[8]      0.0    0.02    0.00  120324         hash [8]
-----------------------------------------------
                0.00    0.00  120329/120329      to_lower [9]
[11]     0.0    0.00    0.00  120329         get_string_length [11]
-----------------------------------------------
                0.01    0.00  120329/120329      get_N_gram [6]
[9]      0.0    0.01    0.00  120329         to_lower [9]
                0.00    0.00  120329/120329      get_string_length [11]
```

- Enhancing Parallelism
  203ms의 좋은 시간 단축을 보여주었지만 output.txt파일을 확인한 결과 sort에

```
void bubble_sort(Ngram** arr, int count) {
    for (int i = 0; i < count - 1; i++) {
        for (int j = 0; j < count - i - 1; j += 2) {
            int acc0 = arr[j]->count;
            int acc1 = arr[j + 1]->count;
            acc0 = (acc0 < acc1) ? acc1 : acc0;
            arr[j]->count = acc0;
        }
    }
}
```

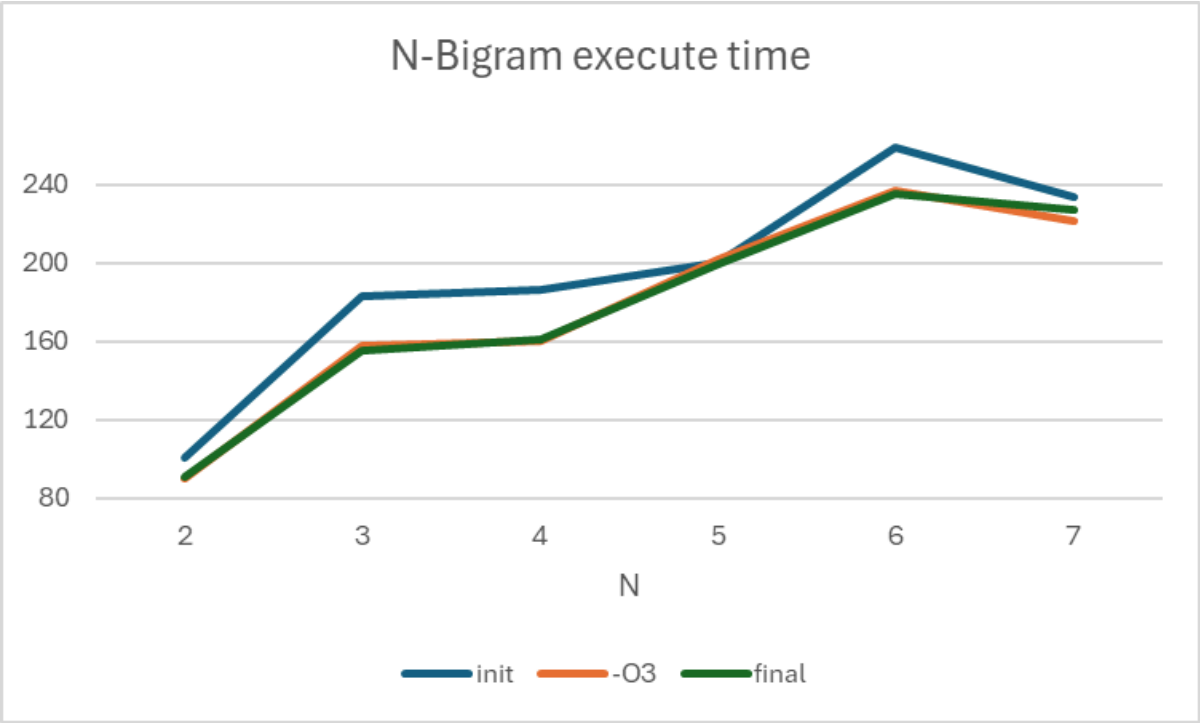
- 기타 최적화
  `to_lower`에서 unroll, `get_N_gram`에서 if문을 mis-prediction을 방지하기 위해 `(count >= N) ? insert_N_gram(table, swnd) : (void)0;`를 시행해주었다. 또한 새롭게 `Hash_Table`구조에 count를 추가해서 가장 처음에 3번째로 오래거렸던 `get_bucket_count` 함수를 0초에 가깝게 만들어주었다. 또한 hash 연산의 곱셈을 bit shift로 변경했다.

```
typedef struct {
        Ngram* bucket[HASH_TABLE_SIZE];
        uint count; // 전체 N-gram 개수 유지
} Hash_Table;

void insert_N_gram(Hash_Table* table, char swnd[MAX_NGRAM][MAX_WORD_LEN]){
    // previous code

    // 전체 N-gram 카운트 업데이트
    table->count++;
}

uint get_bucket_count(Hash_Table* table){
    return table->count;
}

hash = (hash << 5) - hash + *str++;
```

최종 결론

이번 프로젝트는 텍스트 데이터를 처리하는 Bigram Analyzer를 설계하고, 성능 분석 및 최적화를 통해 프로그램을 개선하는 데 중점을 두었다. **최적화 종료 시점은 Init 코드 기준 -O3로 컴파일 했을 때와 유사하여 어느정도 고급 최적화 수준과 비슷한 성능을 달성했다고 판단했다**. 주요 최적화 과정과 결과는 아래와 같다.

1. 초기 상태(2024-2-SP-A3-Initial-2020312914)
    - Bubble Sort를 활용한 정렬 작업이 대부분의 실행 시간(99.63%)을 차지하고, swap 함수 역시 호출 횟수가 많아 병목 현상의 원인이 되었다.
    - 기타 함수들(to_lower, get_bucket_count)에서도 비효율적인 함수 호출과 불필요한 반복문 구조라 발견되었다.
2. 주요 최적화
    1. swap 함수 inline화
        - swap 함수를 inline 함수 또는 매크로 함수화 시켜서 호출에 대한 오버헤드 제거
        - call graph 확인 결과, swap 호출 횟수를 제거, 정렬 속도 약 4% 향상
    2. Loop Unrolling
        - 2-way, 4-way, 6-way, 8-way Unroll 적용
        - 최대 10%의 단축이 있었지만 각 way 마다 큰 차이점은 없었음 따라서 2-way 또는 4-way Unroll을 적용
    3. Branch Prediction 최적화
        - bubble sort의 loop 내부의 if branch에서 mis-prediction 가능성 확인
        - 삼항 연산자 적용
    4. to_lower 반복 호출 최적화
        - 문자열 길이를 loop 외부에서 계산 후 적용
        - 실행 시간 0.01s 이하 수준의 개선
    5. get_bucket_count 및 구조체 일부 수정
        - bucket에 대한 count를 실시간으로 update하는 데이터 구조 수정
        - 0.06s의 점유율을 가지던 get_bucket_count 함수 시간 점유율 제거
    6. Hashing 개선
        - 곱셈 기반 해싱을 비트 연산으로 변경하여 해시 계산 속도 향상
3. 최종 성능 비교(N = 6 기준)
    - 초기 실행 시간 : 244.56초
    - 최적화 후 실행 시간 : 218.33초
    - N-Bigram 별 성능 평가(파랑 : init, 주황 : O3, 초록:final)



4. 결론

당연히 `quick_sort` 같은 알고리즘 측면에서의 효율적인 알고리즘을 통해 성능을 높일 수 있었지만 본 프로젝트에서는 시스템 적인 측면과 컴파일러에 대한 공부를 위함이었기 때문에 제외했습니다. 만약 Bigram Analyzer를 개선한다면 다음과 같은 개선점들을 떠올릴 수 있었습니다:

- Quick Sort나 Merge Sort 같은 효율적인 정렬알고리즘 사용
- 단순 loop의 경우 openMP나 CUDA, pthread 등을 통해 병렬화

이번 프로젝트에서 주요 병목 구간을 정확히 식별하고, 시스템 프로그램을 수강하면서 배운 다양한 최적화 기법을 통해 실행 시간을 성공적으로 단축하였습니다. 특히, `bubble_sort`에 대한 루프 최적화와 함수 호출 최적화는 실행 성능 개선에 좋은 결과를 보여주었습니다. 과제를 통해 성능 최적화와 프로파일링 도구의 사용 방법을 익히고, 수업 시간에 눈으로 보기만 했던 최적화 기법들을 적용하면서 O3 수준의 만족할만한 코드 개선을 수행할 수 있었습니다.