DS Assignment 2

201716472 컴퓨터공학과

최유진

* 1. 셸 정렬

1.1.1 셸 정렬(shell sort) 구현(c++) 및 테스트

코드 설계(클래스 계층도) :

shellsort.h

test\_shellsort.cpp

shellsort.cpp

#include ”shellsort.h”

shellsort.h에 void insertionsort(int \*A, int size, int start, int h);, void shellsort(int \*A, int size, int K);

를 선언하였고 그것을 #include “shellsort.h”를 사용하여 test\_shellsort.cpp에서 shellsort.cpp의 함수들을 사용할 수 있게 하였다.

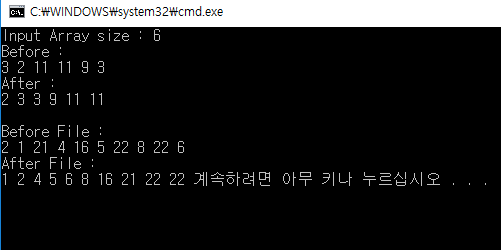
shellsort.cpp에서는 shellsort.h에서 선언한 void insertionsort 함수와 void shellsort 함수를 작성하였다. insertionsort 함수는 배열 A 안에서 start 부터 h 만큼의 간격의 수들을 비교하여 집어 넣는 역할을 하는 함수이다. for문과 temp를 사용하였고, A[j]와 temp의 크기를 비교하여 배열을 크기 순으로 정렬하였다.

shellsort 함수는 insertionsort 함수를 사용하여 k의 값에 따라 insertion의 h값을 조정하는 역할을 하였다.

test\_shellsort.cpp 에는 main() 함수를 만들고 그 안에 배열의 size와 배열의 값을 rand()를 사용하여 랜덤으로 받아, shellosrt함수를 실행하고 출력하였다.

문제에서 제공한 input.txt 또한 ifstream으로 읽어 shellsort 함수를 사용하여 정렬하였다.

실행 결과 :



1.1.2 셸 정렬(sehll sort) 실행시간 비교

코드 설계(클래스 계층도)

shellsort.h

shellsort.cpp

#include ”shellsort.h”

test\_shellsort\_comp.cpp

shellsort.h와 shellsort.cpp의 코드는 1.1.1의 코드와 동일하다.

test\_shellsort\_comp.cpp 에는 문제의 test\_dtime.cpp 예제 코드를 참고하여 main()을 작성하였다.

for 문을 사용하여 K의 값과 N의 값을 순서대로 커지게 만들고 그 안에 shellsort 함수를 넣어 k와 N의 값이 커지는 대로 랜덤 배열을 받아 shellsort 방식으로 정렬되게 작성하였다.

실행 결과 :

#include <sys/time.h>

#include <unistd.h> 이 둘이 visual studio에서 지원되지 않아 실행화면을 올리지 못했다.

1.1.3 셸 정렬(shell sort) : Sedgewick Upper Bound

코드 설계(클래스 계층도)

#include ”shellsort.h”

shellsort.h

shellsort.cpp

test\_shellsort\_sedgewick\_comp.cpp

#include ”shellsort\_sedgewick.h”

shellsort\_sedgewick.cpp

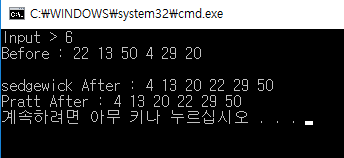
shellsort\_sedgewic.h

shellsort.h와 shellsort.cpp는 1.1.1과 동일하다. shellsort\_sedgewic.h에 void insertionsort2(int \*A, int size, int start, int h);와 void shellsort\_sedgewick(int \*A, int size);을 선언하였고 그것을 #include “shellsort\_sedgewic.h”를 사용하여 test\_shellsort\_sedgewick\_comp.cpp 에서 shellsort\_sedgewick.cpp의 함수들을 사용할 수 있게 하였다.

shellsort\_sedgewick.cpp 에서는 shellsort\_sedgewic.h 에서 선언한 void insertionsort2 함수와 void shellsort\_sedgewick 함수를 작성하였다. insertionsort2 함수는 sehllsort.cpp함수 안에 있는 insertionsort 함수와 동일하다. shellsort\_sedgewick 함수는 문제에서 제공한 h의 식 4^i + 3 \* 2^(i-1) +1 을 사용해야 하므로 for문의 범위를 h(i+1) < size 로 하여 i를 구하였다. h의 식은 #include<cmath>를 사용하여 pow를 넣어 만들었다. 그 후 i가 0이 될 때까지 while문을 사용하여 그 i의 값을 h에 넣고 그것을 insertsort2 함수에 넣어 배열을 정렬 하였다.

test\_shellsort\_sedgewick\_comp.cpp 에는 main() 함수를 만들고 그 안에 배열의 size와 배열의 값을 rand()를 사용하여 랜덤으로 받아 shellsort 함수와 shellsort\_sedgewick 함수를 실행하고 출력하였다. 실행 시간을 비교하는 것은 #include <sys/time.h> #include <unistd.h> 를 지원하지 않아 하지 않았다.

실행 결과 :



1.1.4 셸 정렬(shell sort): 계산복잡도

셸 정렬은 최상의 경우 O(n)의 복잡도를 가지지만 보통의 경우의 시간 복잡도는 간격을 어떻게 조정하느냐에 따라 달라진다.

최악의 경우 Ο(n²)의 시간 복잡도를 가지지만 평균적으로는 Ο(nlog²n) 의 시간 복잡도를 가진다.

1.2 정렬 알고리즘 구현 및 비교 : 삽입 정렬, 셸 정렬, 병합 정렬, 퀵 정렬 비교(c++)

1.2.1 병합 정렬, 퀵 정렬 구현 및 테스트

코드 설계(클래스 계층도) :

test.cpp

quicksort.cpp

mergesort.cpp

#include ”mergesort.cpp”

”

#include “quicksort.cpp”

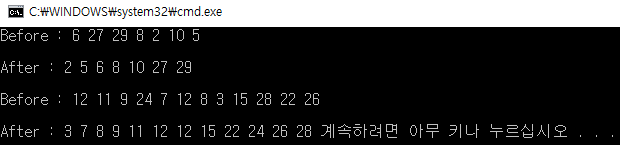
”

quicksort.cpp에는 inline void Swap(in \*A, int first, int last) 함수와 inline int partition(int \*A, int first, int last) 함수, inline void quicksort(int \*A, int first, int last) 함수를 작성하였다. Swap 함수는 A[first] 값과 A[last] 값을 바꾸는 역할을 한다. partition 함수는 p를 배열 A[first] 값으로 지정하고 while(low <= high)문으로 A[low] 값과 A[high] 값을 p와 비교하고 low high의 값을 바꾼다. 그리고 Swap(A, first, high)를 사용하여 나온 high 값을 return 한다. quicksort 함수는 first 값이 last 값보다 작거나 같을 때 partition 함수를 실행, return 받은 값을 quicksort 에 넣는 역할을 한다.

mergesort.cpp에는 inline void merge(int \*A, int F, int Mid, int L) 함수와 mergesort(int \*A, int first, int last) 함수를 작성하였다. merge 함수는 두 그룹으로 배열을 나누고 각 그룹 인덱스가 밖으로 나가지 않을 때까지 for문을 실행한다. 두 그룹의 first 값을 비교하여 작은 쪽을 Temp 배열에 넣고 작은 쪽의 first를 1씩 올린다. 남은 값이 두 그룹에 있으면 temp 배열에 저장하고 다시 배열 A에 배열 temp의 복사한다. mergesort 함수는 middle을 구하고 그 값을 중심으로 mergesort 함수에 first와 last, middle 값을 넣는다. 그 후 merge(A, first, middle, last);를 실행하는 역할을 한다.

test.cpp 에는 main() 함수를 작성한다. main 함수에는 rand()를 사용하여 배열 두 개의 size와 값을 랜덤으로 입력받곡 mergesort와 quicksort 함수를 사용하여 정렬을 출력한다.

실행 결과 :



1.2.2 삽입정렬, 병합정렬, 퀵 정렬 비교(c++)

코드 설계(클래스 계층도) :

shellsort.h

#include ”shellsort.h”

shellsort.cpp

#include ”mergesort.cpp”

”

mergesort.cpp

#include “quicksort.cpp”

”

quicksort.cpp

test\_shellsort\_sedgewick\_comp.cpp

#include ”shellsort\_sedgewick.h”

shellsort\_sedgewick.cpp

shellsort\_sedgewic.h

shellsort.h, shellsort.cpp 는 1.1.1와 같고 shellsort\_sedgewic.h와 shellsort\_sedgewic.cpp는 1.1.3과 같다. mergesort.cpp와 quicksort.cpp는 1.2.1과 동일하다. test\_sort\_comp.cpp 에는 int main() 함수를 작성하였다. for 문을 사용하여N의 값을 1000부터 늘렸고 실행시간을 구하기 위해 1.1.2와 같이 test\_dtime.cpp 예제 코드를 참고하였다. for문과 rand() 를 사용하여 배열 N의 크기와 값을 정하였고 그 값을 shellsort, shellsort\_sedgewick, mergesort, quicksort를 사용하여 정렬한 뒤 실행시간을 출력하였다.

실행 결과 :

#include <sys/time.h>

#include <unistd.h> 이 둘이 visual studio에서 지원되지 않아 실행화면을 올리지 못했다.

1.3 기수 정렬(radix sort)

1.3.1 셈 정렬(count sorting) 구현(c++)

코드 설계(클래스 계층도) :

test\_countingsort.cpp

countingsort.cpp

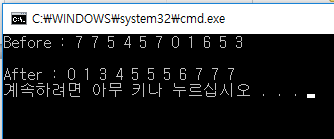
#include ”countingsort.cpp”

countingsort.cpp 에는 inline void countingsort(int \*A, int size, int k) 함수를 작성하였다.

countingsort 함수는 count[MAX], result[MAX]를 선언하여, 배열 A의 값들을 i<size를 조건을 for문을 실행하여, A의 값들이 중복으로 나오는 횟수를 count 배열에 저장한다. 그 후 for 문을 사용하여 count 배열을 누적배열로 바꾼 뒤 for 문으로 result[count[A[i]] -1]을 실행하여 result 값이 각 중복이 있는 숫자들의 정렬을 정리하게 한다. 그 후 배열 A 에 result 값을 복사하는 역할을 한다.

test\_countingsort.cpp 에는 mian() 함수를 작성한다. main 함수는 rand()를 사용하여 배열의 크기와 값들을 정하고 그 배열을 countingsort 함수에 넣어 정렬 한 것을 출력하였다.

실행 결과 :



1.3.2 랜덤 token 생성기 구현

코드 설계(클래스 계층도) :

void file(int \*A)

Assignment1\_3\_2.cpp

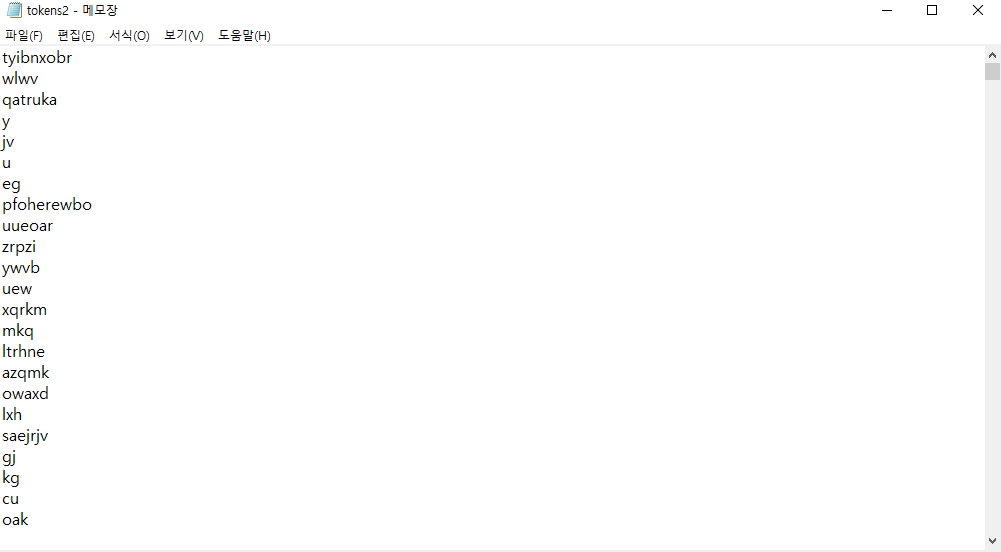
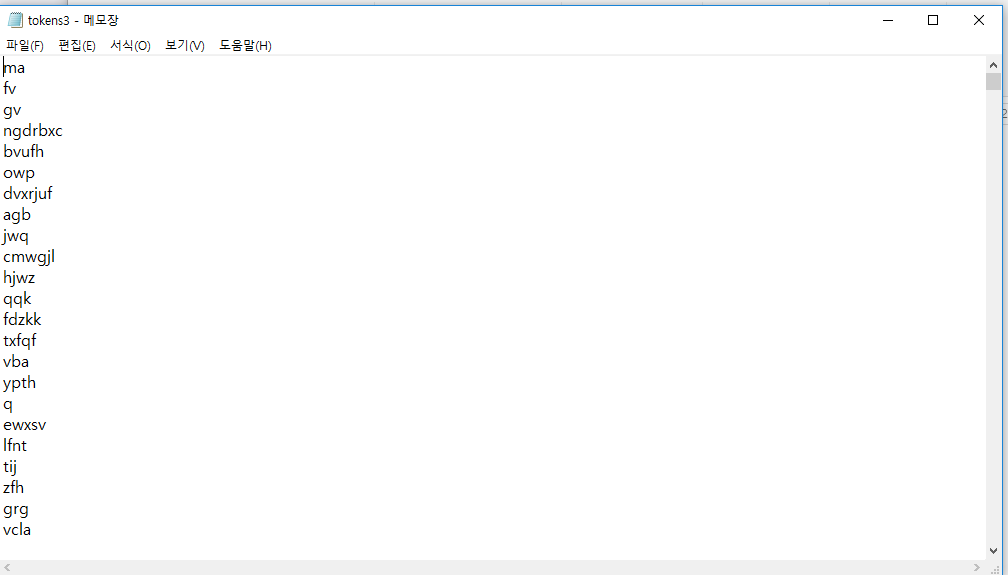
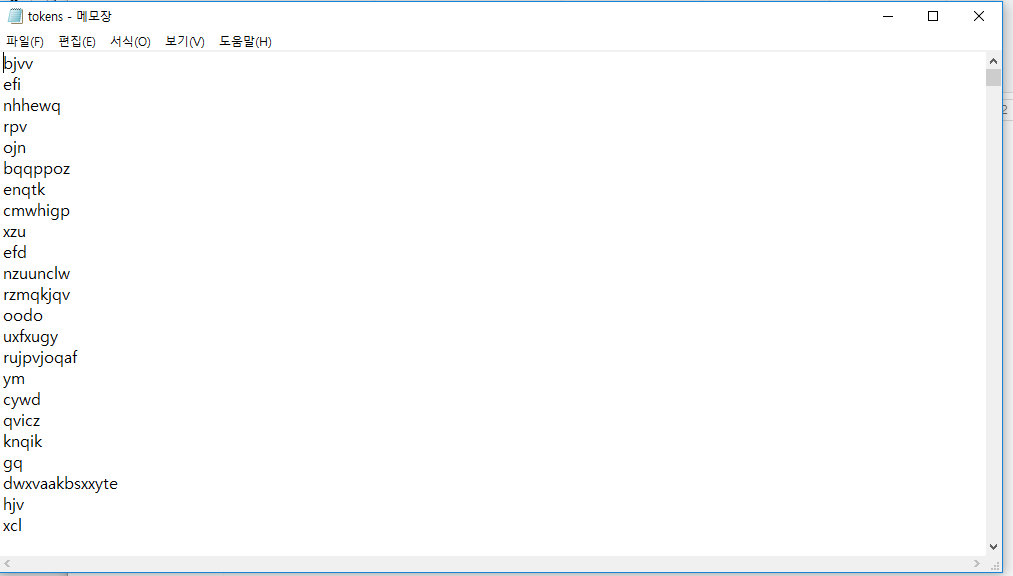
int main()

void file(int \*A) 함수는 word\_length\_stat.txt 파일을 ifstream을 이용하여 읽어 들이고, txt의 각 숫자의 길이가 나올 확률의 값을 받아 num 배열에 정수의 값으로 저장하였다. 그리고 그 값을 for문을 사용하여 누적 값으로 저장하였다. 그렇게 나오는 값의 수만큼 i의 값을 중복으로 배열 A에 저장하는 역할을 하였다.

main() 함수는 랜덤으로 x의 값을 받아 A[x]의 값을 랜덤 token의 길이로 설정하였다. 그 후 n의 값만큼 while 문을 실행하였다. 또한 그 안에 for문을 length 만큼 실행하여 그 수만큼 랜덤 알파벳을 받아 한 줄에 출력하였다. 그 출력 값들을 ofstream을 사용하여 tokens.txt에 출력하였다.

또 다른 값 test : n의 값을 10000, 10000000으로 한 tokens2.txt, tokens3.txt 를 while 문을 사용하여 출력하였다.

실행 결과 :



1.3.3 MSD 기수 정렬 구현(c++)

코드 설계(클래스 계층도) :

void radix\_sort(string \*A, int n, int p)

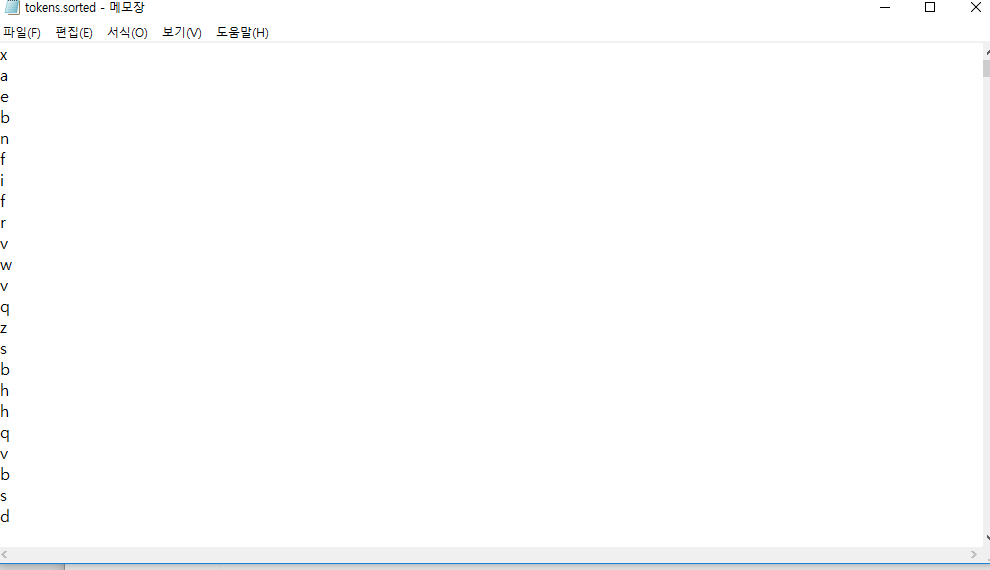
Assignment1\_3\_3.cpp

int main()

void radix\_sort(string \*A, int n, int p) 함수에서는 string 배열과 int 배열을 동적할당으로 받아 while 문을 배열이 나올 수 있는 최대의 길이 p를 받아 조건이 (--p >= 0) 이 될 때까지 실행하였다. cnt 배열에 length 에 따른 배열 A를 countingsort와 비슷한 방식으로 받아 길이가 짧은 순서대로 배열 A의 값이 나열될 수 있도록 한다. 그 후 값을 배열 tmp로 옮겨 ofstream을 사용하여 tokens.sorted.txt에 출력한다.

main() 함수에서는 ifstream을 이용하여 1.3.2에서 실행한 txt를 불러온 후 그것을 배열 A에 저장해 radix\_sort(A, MAX, 26);을 실행한다.

실행 결과



길이 정렬에는 성공하였다.

1.3.4 MSD 기수 교환 정렬 구현(c++)

코드 설계(클래스 계층도) :

typedef struct element

int compare(element e1, element e2)

int partition(element list[], int left, int right)

Assignment1\_3\_3.cpp

void insert(element e)

int main()

#define SWAP(x, y, t) 를 사용하였다

typedef struct 즉 구조체를 사용하여 char word를 list 배열에 받을 수 있게 하였다.

int compare(element e1, element e2) 함수는 strcmp 를 사용하여 e1의 word와 e2의 word가 같은 단어인지를 return 한다. 동일할 경우 1을 아닐 경우 0을 return 한다.

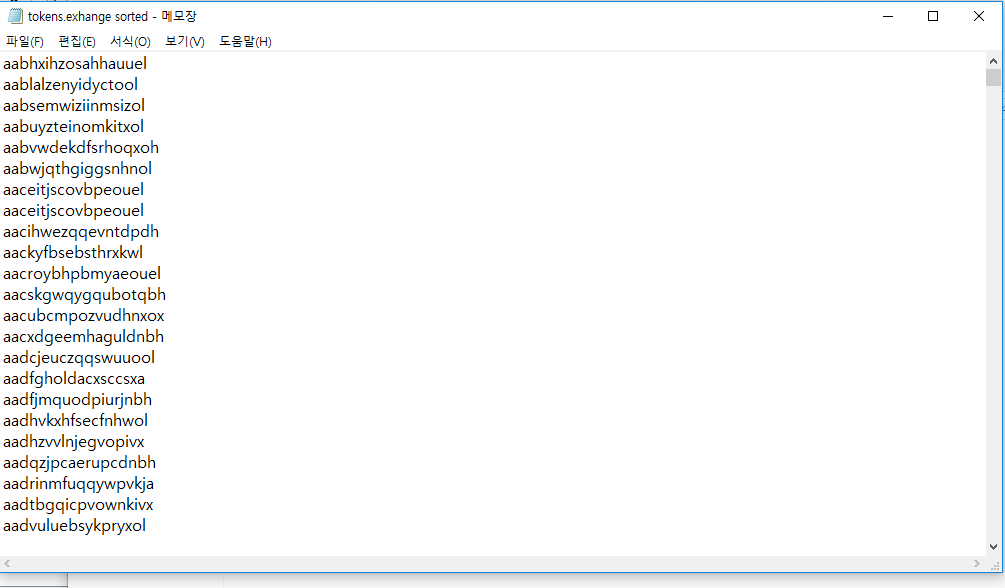
int partition(element list[], int left, int right) 함수는 while(low < high) 일 경우에 실행되며, 그 안의 while 문들 또한 compare(list[low or high], pivot)의 값이 0보다 크거나 같음에 따라 low와 high의 값을 정한다. 그렇게 정한 값의 list 배열을 SWAP하고 high 값을 return 하는 역할을 한다.

void quick\_sort(element list[], int left, int right) 함수는 1.2.1과 비슷한 방식으로 quick\_sort를 실행하여 구조체의 list 배열을 정리하는 역할을 한다.

void insert(element e) 함수는 list 배열에 element e 값을 추가하여 넣는 역할을 한다.

main() 함수에서는 1.3.2의 txt를 ifsteam으로 읽어서 구조체의 word 배열에 저장한다. 그리고 quick\_sort 를 사용하여 배열을 정렬하고 그 값들을 ofstream 으로 출력하였다.

실행 결과 :



1.3.3 과의 출력속도 비교는 실행이 되지 않아 하지 못하였다.

1.4 이진 탐색 트리(binary search tree)

1.4.1 이진 탐색 트리 : 탐색, 삽입, 삭제, 갱신, 구현(c++)

코드 설계(클래스 계층도) :

#include ”BST.cpp”

BST.cpp

test\_BST.cpp

BST.cpp 에는 문제에서 제공한 구조체와 insert update delete search함수들 그리고 inline void SuccessorCopy(Nptr& T, datatype& DelNodeData)함수, inline void Display(Nptr T)함수들을 작성하였다. insert 함수는 T가 null 일 경우 새 노드를 만들어 key 를 받고 LChild와 RChild 를 null로 하였다. 그리고 data의 key 와 크기를 비교하여 LChild나 RChild로 이동한 값을 insert 함수에 넣었다. key 가 같은 경우, 그 data 의 count를 +1 하며, T값을 return 하는 역할을 하였다.

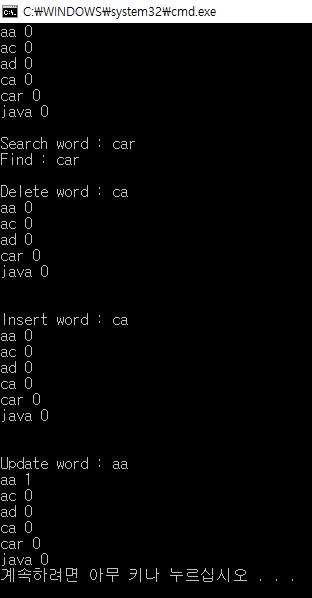
SucccessorCopydml 함수는 Nptr T를 받아 다른 노드에 똑같이 복사하는 역할을 하는 함수 이다.

Delete 함수는 insert 와 비슷한 방식으로 크거나 작거을 경우 Delete 함수를 실행하고 key와 같을 경우 T를 null 로 비우고 삭제하는 역할을 하였다.

Search 함수는 key의 값과 T의 data를 비교하며 T의 data key에 key가 존재할 경우 T를 return 시키는 역할을 하였다.Display 함수는 T의 트리를 출력하는 역할을 하였다.

test\_BST.cpp는 Nptr s를 선언하고 insert함수와 Search함수 Delete, Update 함수 들을 사용하였다. 함수들에 넣을 값들을 cin으로 받아 함수에 넣고 Display 함수를 사용하여 그것들을 출력하였다.

실행 결과 :



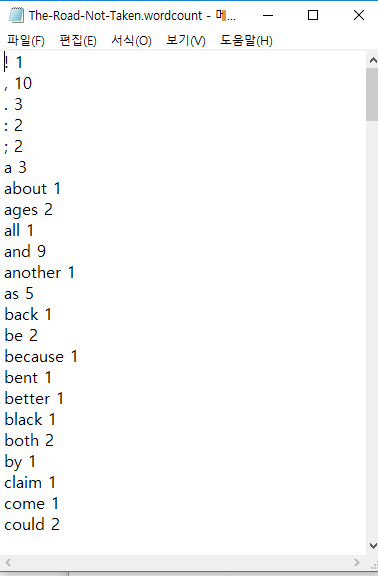
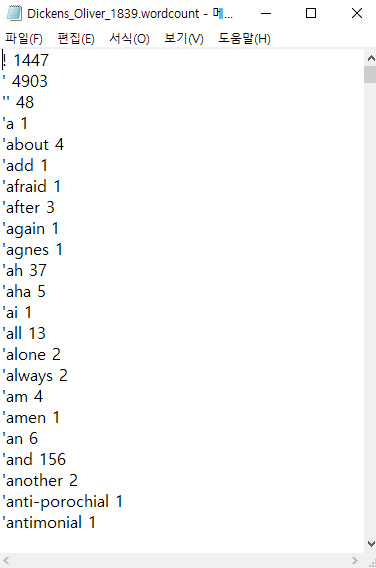
1.4.2 이진 탐색 트리를 이용한 Word Count 계산

코드 설계(클래스 계층도) :

1.4.1에서 사용한 구조체와 Update Insert Display 함수를 사용하였다. update 와 insert 함수는 1.4..1과 동일하며, display 함수는 ofstream 을 이용하여 filename 을 입력받아 그것을 txt로 출력하는 역할이 추가되었다.

mian()함수는 ifstream으로 The-Road-Not-Taken.tockens.txt 와 Dickens\_Oliver\_1839.tokens.txt를 받아 그것들을 각 벡터에 넣고 string 배열에 복사한 후 벡터의 크기만큼 실행 될 for문 안에서 insert 함수를 사용하여 Nptr 에 값들을 넣고 update 함수를 실행하였다. 그 후 Display 함수를 사용하여 The-Road-Not-Taken.wordcount 와 Dickens\_Oliver\_1839.wordcount 를 출력하였다.

실행 결과 :

1.4.3 Word Count 결과로부터 이진탐색(binary search)

코드 설계(클래스 계층도) :

#include ”BST.cpp”

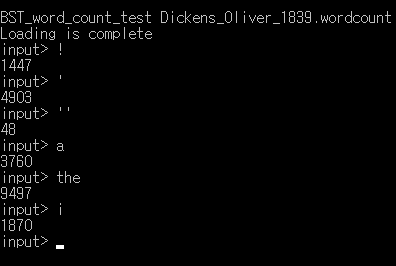
BST.cpp

test\_BST.cpp

BST.cpp 는 1.4.1의 BST에서 구조체와 Search, insert, Display 함수를 사용한 것이다.

test\_BST.cpp 에서는 main() 함수를 사용하였다. main() 함수에서는 1.4.2에서 출력하였던 The-Road-Not-Taken.wordcount와 Dickens\_Oliver\_1839.wordcount를 ifstream을 사용하여 읽어왔다. 그 txt들의 값을 1.4.2처럼 벡터로 받아 배열로 옮긴 후 for문 안에서 Insert 함수를 사용하면서 search 함수를 사용하여 그 data 의 count 값을 txt에서 받은 숫자로 바꾸었다. 그리고 그것을 확인하기 위하여 cin 으로 원하는 단어를 입력 받아 seach 함수를 사용하여 그 단어의 count 값을 받고 그 값을 출력하였다. The-Road-Not-Taken.wordcount 은 10번을 입력 받을 수 있게 하였고, Dickens\_Oliver\_1839.wordcount는 무한히 입력 받을 수 있게 하였다.

실행 결과 :

1.4.4 Word Count 결과로부터 이진탐색(binary search) : 대용량 데이터로 확장 방안

1.5 해시(hash)

1.5.1 해시 클래스 구현(c++)

코드 설계(클래스 계층도) :

hash.h

hash.cpp

#include ”hash.cpp”

hash\_test.cpp

hash.h 에는 문제에서 제공한 구조체와 Create, Insert, search, Update, Delete, Save, Open 함수들 그리고 Hash(), Display() 함수들을 선언하였다.

hash.cpp 에서는 선언한 함수들을 작성하였다. Hash 함수는 table 과 size를 설정하는 역할을 하였다. Create 함수는 size와 table 값을 새로 받고 그것을 저장하는 역할을 하였다.

strhashfunc 함수는 문제에서 제공한 함수를 사용하였고 문자열에 대한 해시 함수이다.

insert 함수는 새로운 노드를 받아 그 노드에 key 을 넣고 count 를 1 증가시키고 원래 있던 노드들을 다음 노드로 이동하는 역할을 하였다.

search 함수는 while 문을 사용하여 node의 data에서 일치하는 key가 있는지 확인하는 역할을 하였다.

update 함수는 같은 key 값이 있는 node 일 경우 count 를 증가시키고 비어있을 경우 data를 추가하는 역할을 하였다.

Delete 함수는 while 문을 이용하여 일치하는 key의 값을 찾아 존재하면 그 값을 없애는 역할을 하였다.

save 함수는 ofstream 을 사용하여 file에 값을 받아 txt 로 출력하는 역할을 하였다.

Open 함수는 ifstream을 사용하여 txt의 값을 읽어와 그 값들을 insert 와 update 함수를 사용하여 hash 에 넣는 역할을 하였다.

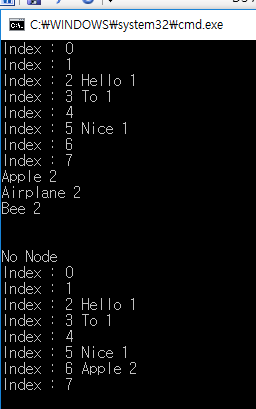
Rehash 함수는 hash를 복제하는 역할을 하였다.

마지막으로 display 함수는 받은 값들을 출력하게 하는 역할을 맡았다.

hash\_test.cpp 에서는 main() 함수를 작성하였다.

hash A를 만들어 insert, Delete, Open, Display, Update 함수들을 사용하여 hash를 나타내었다.

실행 결과 :



1.5.2 해시를 이용한 Word count 계산 및 사용자 테스트

코드 설계(클래스 계층도) :

Hash.h

Hash.cpp

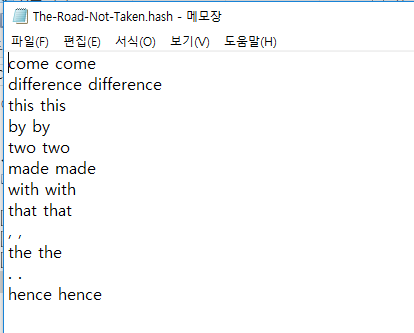
#include ”hash.cpp”

Hash\_word\_count\_test.cpp

Hash.h 는 display 함수가 없어진 것 외에는 1.5.1과 같다. Hash.cpp 또한 동일하다. 단 open 함수에는 create(20000); 가 추가되었다.

Hash\_word\_count\_test.cpp 에서는 main 함수가 작성되었다. 파일을 읽어 들이기 위해 const char\*로 파일들을 받은 후 hash 를 선언하고 Open, Save 함수들을 사용하였다.

실행 결과 :



1.5.3 해시와 이진탐색트리 효율 비교

코드 설계(클래스 계층도) :

Hash.h

Hash.cpp

#include ”hash.cpp”

BST.cpp

test.cpp

hash.cpp 와 hash.h의 코드는 1.5.2와 같다. BST.cpp 또한 1.4.1의 코드와 동일하다.

test.cpp 에서는 1.2 문제에 있던 gettime() 함수와 char random() 함수, main() 함수를 작성하였다.

random 함수는 const char 로 받은 배열 alpha 를 random 으로 return 하는 역할을 한다.

그리고 main 함수에서 그것을 받아 hash를 받아 for 문을 실행할 때 랜덤으로 Search 함수를 실행하기 위해 사용한다. 그 뒤 1.2.2 와 비슷한 방식으로 실행 시간을 구하여 출력한다.

BST 또한 배열과 벡터를 받고 파일을 ifstream으로 읽어 배열에 집어 넣고 for문을 사용하여 random search 함수를 실행 시킨다. 그리고 실행 시간을 구하여 출력한다.

실행 결과 :

#include <sys/time.h>

#include <unistd.h> 이 둘이 visual studio에서 지원되지 않아 실행화면을 올리지 못했다.

1.6 힙 정렬(Heap sort)

1.6.1 하향식 힙 구성 + 1.6.2 힙 정렬

코드 설계(클래스 계층도) :

heap\_sort.cpp

#include ” Assignment 1\_6.cpp

”

Assignment 1\_6.cpp

Assignment 1\_6.cpp 는 1.6.1을 위한 것이다. Assignment 1\_6.cpp 에는 inline void swap(int \*a, int \*b) 함수와 inline void build\_heap(int \* A, int size) 함수를 작성하였다.

Swap은 a와 b의 값을 교환하는 역할을 하는 함수이다.

build\_heap 함수는 for문을 사용하여 (i=2; i<=size; i++) 안에 (j=I; j>1; j = j/2)인 for문을 사용하였다. 그 안에 배열 A[j]의 값이 A[j/2]의 값보다 크면 Swap 함수를 사용하는 역할을 하는 함수이다. 이것은 자식 트리와 부모 트리를 비교하였을 때 자식이 부모 보다 크면 서로 교체하기 위해 이렇게 코딩한 것이다.

1.6.2를 위한 heap\_sort.cpp 에서는 int remove\_heap(int \*A, int size) 함수와 mian() 함수를 작성하였다. remove 함수는 root에 배열 A[1] 즉, 트리의 루트 값을 입력하고 A[1]에는 제일 작은 값인 A[size]의 값을 넣은 뒤 size를 1 줄인다. 그 후 다시 build\_heap함수를 사용하여 A[1]에 들어온 값이 알맞은 자리에 찾아갈 수 있게 만든다. 그 후 root 의 값을 return 하여 트리의 가장 큰 값을 차례로 return 하게 해주는 역할을 한다.

mian() 함수에는 배열의 크기와 값들을 rand()를 사용하여 랜덤으로 입력 받은 후 build\_heap 함수를 사용하여 1.6.1의 문제에서 원하는 동작을 출력한 후 remove\_heap 또한 for 을 사용하여 출력하여 1.6.2에서 원하는 값 즉, 큰 값부터 차례로 출력되게 하였다.

실행 결과 :

