**Homework 6**

|  |
| --- |
| 그림입니다. 원본 그림의 이름: YU_UI_RGB-10.png 원본 그림의 크기: 가로 2256pixel, 세로 3047pixel 프로그램 이름 : Adobe ImageReady |

|  |  |
| --- | --- |
| 과목명 | 프로그래밍 언어 |
| 교수님 | 김영탁 교수님 |
| 이 름 | 김주환 |
| 학 번 | 21812158 |
| 일 자 | 2021.04.12.월 |

1. **단어 (문자열) 배열의 선택 정렬 및 퀵 정렬 구현, 실행시간 측정**

1) rand() 함수를 사용하여 5 ~ 15자의 문자로 구성된 단어(문자열)을 생성하여 지정된 주소에 저장하는 함수 (void genWord(char word[], int min\_word\_len, int max\_word\_len))를 작성하라. 단어의 최소길이 및 최대 길이는 min\_word\_len과 max\_word\_len으로 지정된다. 문자열의 첫번째 문자는 대문자로 구성되어야 하며, 나머지 문자는 소문자로 구성된다. 문자의 마지막에는 반드시 ‘\0’이 포함되도록 할 것. 단어(문자열) 관련 함수들은 WordArray.cpp 파일에 구현하며, 이 함수들의 함수원형은 WordArray.h 헤더파일에 포함시킬 것.

void genRandWordArray(char words[][MAX\_WORD\_LEN], int num\_words) {

char temp[MAX\_WORD\_LEN] = ""; // 버퍼역할

int len = 0; // 생성할 문자열의 길이

int j;

for (int i = 0; i < num\_words; i++) {

len = rand() % 10 + 5; // 문자열 길이

temp[0] = rand() % 26 + 65; // 대문자

for (j = 1; j < len; j++) { // 소문자

temp[j] = rand() % 26 + 97;

}

temp[j] = NULL; // 센티널

strcpy(words[i], temp); // 문자열 복사

}

}

2) 최대 MAX\_WORD\_LEN (15)자 길이의 문자열을 최대 MAX\_NUM\_WORDS (500,000)개 저장할 수 있는 2차원 문자 배열을 “WordArrayData.cpp” 소스파일에 선언하고, 위 1)에서 작성한 genWord() 함수를 사용하여 문자열을 생성하여 2차원 문자 배열의 값으로 초기화하라.

#include "WordArray.h"

extern char words[MAX\_NUM\_WORDS][MAX\_WORD\_LEN] = {

"double", "char", "auto", "default", "const", "continue", "break", "case", "do",

"union", "unsigned", "void", "volatile", "while",

"return", "short", "signed", "sizeof", "static", "struct", "switch", "typeof",

"else", "enum", "extern", "float", "for", "goto", "if", "int","long", "register"

};

3) 큰 규모의 단어 배열의 첫부분과 마지막 부분을 파일로 출력하는 함수 void fprintBigWordArray(FILE \*fout, char wordlist[][MAX\_WORD\_LEN], int size, int words\_per\_line = 10, int sample\_lines = 5)를 작성하라. 이 함수는 한 줄에 words\_per\_line 개수의 단어를 출력하며, 첫 부분의 sample\_lines, 마지막 부분의 sample\_lines 수 만큼의 줄을 출력한다.

void fprintBigWordArray(FILE\* fout, char wordList[][MAX\_WORD\_LEN], int size, int words\_per\_line = 10, int sample\_lines = 5) {

int cnt = 0; // 어디까지 출력했는지 확인

for (int i = 0; i < sample\_lines; i++) { // 앞부분 50단어

for (int j = 0; j < words\_per\_line; j++) {

fprintf(fout, "%-25s", wordList[cnt++]);

}

fprintf(fout, "\n");

}

fprintf(fout, ". . . . .\n"); // 중간 다리

cnt = size - 50;

for (int i = 0; i < sample\_lines; i++) { // 뒷부분 50단어

for (int j = 0; j < words\_per\_line; j++) {

fprintf(fout, "%-25s", wordList[cnt++]);

}

fprintf(fout, "\n");

}

}

텍스트, 모니터, 화면, 스크린샷이(가) 표시된 사진

자동 생성된 설명

4) selection sorting 구조를 기반으로 NUM\_WORDS 개의 문자열을 정렬하는 함수 void selectionSortWordArray(char word[][MAX\_WORD\_LEN], int num\_words)를 작성하라. 정렬된 결과의 첫번째 10개 단어 x 5줄과 맨 마지막 10개 단어 x 5줄을 fprintBiigWordArray() 함수를 사용하여 파일로 출력하라.

void selectionSortWordArray(char word[][MAX\_WORD\_LEN], int num\_words) {

char temp[MAX\_WORD\_LEN]; // 버퍼역할

int min\_pos;

for (int pos = 0; pos < num\_words; pos++) { // 기준

min\_pos = pos;

for (int j = pos + 1; j < num\_words; j++) { // 비교대상

if (strcmp(word[min\_pos], word[j]) > 0) min\_pos = j; // 작은지 판단

}

if (min\_pos != pos) { // 작을 때 교환

strcpy(temp, word[pos]);

strcpy(word[pos], word[min\_pos]);

strcpy(word[min\_pos], temp);

}

}

}

텍스트이(가) 표시된 사진

자동 생성된 설명

5) quick sorting 구조를 기반으로 NUM\_WORDS개의 문자열을 정렬하는 함수 void quickSortWordArray(char word[][MAX\_WORD\_LEN], int num\_words)를 작성하라. 정렬된 결과의 첫 번째 10개 단어 x 5줄과 맨 마지막 10개 단어 x 5줄을 fprintBigWordArray() 함수를 사용하여 파일로 출력하라.

void quickSortWordArray(char word[][MAX\_WORD\_LEN], int num\_words, int left, int right) {

int pi, n\_pi;

char pi\_v[MAX\_WORD\_LEN] = "";

char temp[MAX\_WORD\_LEN] = "";

int i;

if (left >= right)return;

else if (left < right)pi = (left + right) / 2; // 피벗 설정

strcpy(pi\_v, word[pi]); // 피벗 값

strcpy(temp, word[pi]); // 마지막 값과 교환

strcpy(word[pi], word[right]);

strcpy(word[right], temp);

n\_pi = left; // 처음부터 시작

for (i = left; i <= (right - 1); i++) { // 하나씩 옮겨가며 판단

if (strcmp(word[i], pi\_v) <= 0) {

strcpy(temp, word[i]);

strcpy(word[i], word[n\_pi]);

strcpy(word[n\_pi], temp);

n\_pi++;

}

}

strcpy(temp, word[n\_pi]);

strcpy(word[n\_pi], word[right]);

strcpy(word[right], temp);

if (left < (n\_pi - 1))quickSortWordArray(word, num\_words, left, n\_pi - 1);

if ((n\_pi + 1) < right)quickSortWordArray(word, num\_words, n\_pi + 1, right);

}

텍스트, 모니터, 스크린샷, 화면이(가) 표시된 사진

자동 생성된 설명

6) Windows 운영체제에서 제공하는 Performance Counter를 사용하여 selectionSortWordArray() 함수와 quickSortWordArray() 함수가 10000 ~ 100000 개의 단어를 정렬할 때 걸린 시간을 millisecond 단위로 측정하여 출력하라. 10000개 단어씩 증가시키며 측정할 것.

텍스트이(가) 표시된 사진

자동 생성된 설명

7) 단어 배열의 정렬에서 selectioSortWordArray() 함수와 quickSortWordArray() 함수의 실행시간에 차이가 나는 이유를 설명하라. 만약 NUM\_WORDS가 500,000 및 1,000,000이 되었을 때 어떤 차이가 나는지 예상하여 설명하라.

선택정렬은 반복문으로 퀵정렬은 재귀함수로 동작한다. 특정 임계값 이전에는 선택정렬이 더 빠르고 이후에는 퀵정렬이 더 빠르다.

퀵정렬의 재귀함수 호출에서 함수 호출의 오버헤드가 발생하며, 따라서 배열의 원소 개수가 작을 경우 선택정렬보다 늦은 성능을 가질 수 있다.

선택정렬은 반복문의 크기만큼 그대로 실행하지만 퀵정렬의 경우에는 크기를 나눠가며 재귀하기 때문에 크기가 커질수록 선택정렬에 비해 훨씬 빨라지는 것을 확인할 수 있다.

시간 복잡도 측면에서 보면 선택정렬은 O(N^2), 퀵정렬은 NlogN이다. 이는 배열의 크기가 커질수록 속도에 많은 차이가 발생함을 알 수 있다. 예를들어, 크기가 500,000 혹은 1,000,000일 때 선택정렬은 그 크기에 전부 반복문을 실행하지만 퀵정렬 같은 경우에는 재귀함수를 통해 크기를 나눠가며 동작하기 때문에 크기가 커질수록 퀵정렬이 더 효율적임을 알 수 있다.

8) main() 함수

void main() {

extern char words[][MAX\_WORD\_LEN];

LARGE\_INTEGER freq, t1, t2;

LONGLONG t\_diff;

double elapsed\_t;

FILE\* fout;

fout = fopen("output.txt", "w"); // 파일 쓰기모드로 열기

if (fout == NULL) {

printf("Error in creation of output.txt file !!\n");

exit(-1);

}

srand(time(NULL)); // 시드값 설정

QueryPerformanceFrequency(&freq); // 주파수 값

printf("Performance Analysis of Sorting Algorithms\n");

for (int num\_words = 10000; num\_words <= 100000; num\_words += 10000) {

printf("Word array(size : %7d) : ", num\_words);

fprintf(fout, "\n=================================================================================\n");

genRandWordArray(words, num\_words); // 무작위 문자열 생성

fprintf(fout, "Randomly generated Word Array (size : %d)\n", num\_words);

fprintBigWordArray(fout, words, num\_words, WORDS\_PER\_LINE, SAMPLE\_LINES);

fprintf(fout, "\nQuick sorting for Word Array (size : %d) .....\n", num\_words);

QueryPerformanceCounter(&t1); // 이전시간

quickSortWordArray(words, num\_words, 0, num\_words - 1); // 퀵정렬

QueryPerformanceCounter(&t2); // 이후시간

t\_diff = t2.QuadPart - t1.QuadPart;

elapsed\_t = ((double)t\_diff / freq.QuadPart);

fprintf(fout, "QuickSort\_WordArray (size : %d) took %10.3lf[millisecond], ", num\_words, elapsed\_t \* 1000.0); // 밀리초 단위로 출력

printf("QuickSort (%10.3lf ms), ", elapsed\_t \* 1000.0);

fprintf(fout, "\nAfter sorting word array (size : %d)\n", num\_words);

fprintBigWordArray(fout, words, num\_words, WORDS\_PER\_LINE, SAMPLE\_LINES);

fprintf(fout, "\n");

shuffleWordArray(words, num\_words, num\_words); // 문자열 섞기

fprintf(fout, "Word Array (size : %d) after Shuffling\n", num\_words);

fprintBigWordArray(fout, words, num\_words, WORDS\_PER\_LINE, SAMPLE\_LINES);

fprintf(fout, "\nSelection sorting for Word Array (size : %d) .....\n", num\_words);

QueryPerformanceCounter(&t1); // 이전시간

selectionSortWordArray(words, num\_words); // 선택정렬

QueryPerformanceCounter(&t2); // 이후시간

t\_diff = t2.QuadPart - t1.QuadPart;

elapsed\_t = ((double)t\_diff / freq.QuadPart);

fprintf(fout, "SelectionSort\_WordArray (size : %d) took %10.3lf [millisecond]\n", num\_words, elapsed\_t \* 1000.0); // 밀리초 단위로 출력

printf("Select\_Sort (%10.3lf ms)\n", elapsed\_t \* 1000.0);

fprintf(fout, "\nAfter sorting word array (size : %d)\n", num\_words);

fprintBigWordArray(fout, words, num\_words, WORDS\_PER\_LINE, SAMPLE\_LINES);

fprintf(fout, "\n");

}

fclose(fout); // 파일 닫기

}

8) main() 함수

void main() {

extern char words[][MAX\_WORD\_LEN];

LARGE\_INTEGER freq, t1, t2;

LONGLONG t\_diff;

double elapsed\_t;

FILE\* fout;

fout = fopen("output.txt", "w"); // 파일 쓰기모드로 열기

if (fout == NULL) {

printf("Error in creation of output.txt file !!\n");

exit(-1);

}

srand(time(NULL)); // 시드값 설정

QueryPerformanceFrequency(&freq); // 주파수 값

printf("Performance Analysis of Sorting Algorithms\n");

for (int num\_words = 10000; num\_words <= 100000; num\_words += 10000) {

printf("Word array(size : %7d) : ", num\_words);

fprintf(fout, "\n=================================================================================\n");

genRandWordArray(words, num\_words); // 무작위 문자열 생성

fprintf(fout, "Randomly generated Word Array (size : %d)\n", num\_words);

fprintBigWordArray(fout, words, num\_words, WORDS\_PER\_LINE, SAMPLE\_LINES);

fprintf(fout, "\nQuick sorting for Word Array (size : %d) .....\n", num\_words);

QueryPerformanceCounter(&t1); // 이전시간

quickSortWordArray(words, num\_words, 0, num\_words - 1); // 퀵정렬

QueryPerformanceCounter(&t2); // 이후시간

t\_diff = t2.QuadPart - t1.QuadPart;

elapsed\_t = ((double)t\_diff / freq.QuadPart);

fprintf(fout, "QuickSort\_WordArray (size : %d) took %10.3lf[millisecond], ", num\_words, elapsed\_t \* 1000.0); // 밀리초 단위로 출력

printf("QuickSort (%10.3lf ms), ", elapsed\_t \* 1000.0);

fprintf(fout, "\nAfter sorting word array (size : %d)\n", num\_words);

fprintBigWordArray(fout, words, num\_words, WORDS\_PER\_LINE, SAMPLE\_LINES);

fprintf(fout, "\n");

shuffleWordArray(words, num\_words, num\_words); // 문자열 섞기

fprintf(fout, "Word Array (size : %d) after Shuffling\n", num\_words);

fprintBigWordArray(fout, words, num\_words, WORDS\_PER\_LINE, SAMPLE\_LINES);

fprintf(fout, "\nSelection sorting for Word Array (size : %d) .....\n", num\_words);

QueryPerformanceCounter(&t1); // 이전시간

selectionSortWordArray(words, num\_words); // 선택정렬

QueryPerformanceCounter(&t2); // 이후시간

t\_diff = t2.QuadPart - t1.QuadPart;

elapsed\_t = ((double)t\_diff / freq.QuadPart);

fprintf(fout, "SelectionSort\_WordArray (size : %d) took %10.3lf [millisecond]\n", num\_words, elapsed\_t \* 1000.0); // 밀리초 단위로 출력

printf("Select\_Sort (%10.3lf ms)\n", elapsed\_t \* 1000.0);

fprintf(fout, "\nAfter sorting word array (size : %d)\n", num\_words);

fprintBigWordArray(fout, words, num\_words, WORDS\_PER\_LINE, SAMPLE\_LINES);

fprintf(fout, "\n");

}

fclose(fout); // 파일 닫기

}

fprintf(fout, "\nAfter sorting word array (size : %d)\n", num\_words);

fprintBigWordArray(fout, words, num\_words, WORDS\_PER\_LINE, SAMPLE\_LINES);

fprintf(fout, "\n");

shuffleWordArray(words, num\_words, num\_words); // 문자열 섞기

fprintf(fout, "Word Array (size : %d) after Shuffling\n", num\_words);

fprintBigWordArray(fout, words, num\_words, WORDS\_PER\_LINE, SAMPLE\_LINES);

fprintf(fout, "\nSelection sorting for Word Array (size : %d) .....\n", num\_words);

QueryPerformanceCounter(&t1); // 이전시간

selectionSortWordArray(words, num\_words); // 선택정렬

QueryPerformanceCounter(&t2); // 이후시간

t\_diff = t2.QuadPart - t1.QuadPart;

elapsed\_t = ((double)t\_diff / freq.QuadPart);

fprintf(fout, "SelectionSort\_WordArray (size : %d) took %10.3lf [millisecond]\n", num\_words, elapsed\_t \* 1000.0); // 밀리초 단위로 출력

printf("Select\_Sort (%10.3lf ms)\n", elapsed\_t \* 1000.0);

fprintf(fout, "\nAfter sorting word array (size : %d)\n", num\_words);

fprintBigWordArray(fout, words, num\_words, WORDS\_PER\_LINE, SAMPLE\_LINES);

fprintf(fout, "\n");

}

fclose(fout); // 파일 닫기

}

/\*

\* 파일명 : main.cpp

\* 목적 및 기본 기능

\* 무작위 문자열을 생성하고 선택정렬과 퀵정렬을 각각 실행하여 속도를 비교하고 파일에 출력하라

\* 작성자 : 김주환(21812158)

\* 작성일 : 2021년 4월 12일

\*/

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <stdio.h>

#include <stdlib.h>

#include <Windows.h>

#include <time.h>

#include <string.h>

#include "WordArray.h"

void main() {

extern char words[][MAX\_WORD\_LEN];

LARGE\_INTEGER freq, t1, t2;

LONGLONG t\_diff;

double elapsed\_t;

FILE\* fout;

fout = fopen("output.txt", "w"); // 파일 쓰기모드로 열기

if (fout == NULL) {

printf("Error in creation of output.txt file !!\n");

exit(-1);

}

srand(time(NULL)); // 시드값 설정

QueryPerformanceFrequency(&freq); // 주파수 값

printf("Performance Analysis of Sorting Algorithms\n");

for (int num\_words = 10000; num\_words <= 100000; num\_words += 10000) {

printf("Word array(size : %7d) : ", num\_words);

fprintf(fout, "\n=================================================================================\n");

genRandWordArray(words, num\_words); // 무작위 문자열 생성

fprintf(fout, "Randomly generated Word Array (size : %d)\n", num\_words);

fprintBigWordArray(fout, words, num\_words, WORDS\_PER\_LINE, SAMPLE\_LINES);

fprintf(fout, "\nQuick sorting for Word Array (size : %d) .....\n", num\_words);

QueryPerformanceCounter(&t1); // 이전시간

quickSortWordArray(words, num\_words, 0, num\_words - 1); // 퀵정렬

QueryPerformanceCounter(&t2); // 이후시간

t\_diff = t2.QuadPart - t1.QuadPart;

elapsed\_t = ((double)t\_diff / freq.QuadPart);

fprintf(fout, "QuickSort\_WordArray (size : %d) took %10.3lf[millisecond], ", num\_words, elapsed\_t \* 1000.0); // 밀리초 단위로 출력

printf("QuickSort (%10.3lf ms), ", elapsed\_t \* 1000.0);

// WordArrayData.cpp

#include "WordArray.h"

extern char words[MAX\_NUM\_WORDS][MAX\_WORD\_LEN] = {

"double", "char", "auto", "default", "const", "continue", "break", "case", "do",

"union", "unsigned", "void", "volatile", "while",

"return", "short", "signed", "sizeof", "static", "struct", "switch", "typeof",

"else", "enum", "extern", "float", "for", "goto", "if", "int","long", "register"

};

// WordArray.h

#ifndef WA // 헤더 중복 방지

#define WA

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <stdio.h>

#include <stdlib.h>

#include <Windows.h>

#include <time.h>

#include <string.h>

#define MAX\_WORD\_LEN 16

#define MAX\_NUM\_WORDS 100000

#define WORDS\_PER\_LINE 10

#define SAMPLE\_LINES 5

void genRandWordArray(char words[][MAX\_WORD\_LEN], int num\_words);

void fprintBigWordArray(FILE\* fout, char wordList[][MAX\_WORD\_LEN], int size, int words\_per\_line = 10, int sample\_lines = 5);

void selectionSortWordArray(char word[][MAX\_WORD\_LEN], int num\_words);

void quickSortWordArray(char word[][MAX\_WORD\_LEN], int num\_words, int left, int right);

void shuffleWordArray(char word[][MAX\_WORD\_LEN], int num\_words, int cnt);

#endif

// WordArray.cpp

#ifndef WA // 헤더 중복 방지

#define WA

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <stdio.h>

#include <stdlib.h>

#include <Windows.h>

#include <time.h>

#include <string.h>

#define MAX\_WORD\_LEN 16

#define MAX\_NUM\_WORDS 100000

#define WORDS\_PER\_LINE 10

#define SAMPLE\_LINES 5

void genRandWordArray(char words[][MAX\_WORD\_LEN], int num\_words) {

char temp[MAX\_WORD\_LEN] = ""; // 버퍼역할

int len = 0; // 생성할 문자열의 길이

int j;

for (int i = 0; i < num\_words; i++) {

len = rand() % 10 + 5; // 문자열 길이

temp[0] = rand() % 26 + 65; // 대문자

for (j = 1; j < len; j++) { // 소문자

temp[j] = rand() % 26 + 97;

}

temp[j] = NULL; // 센티널

strcpy(words[i], temp); // 문자열 복사

}

}

void fprintBigWordArray(FILE\* fout, char wordList[][MAX\_WORD\_LEN], int size, int words\_per\_line = 10, int sample\_lines = 5) {

int cnt = 0; // 어디까지 출력했는지 확인

for (int i = 0; i < sample\_lines; i++) { // 앞부분 50단어

for (int j = 0; j < words\_per\_line; j++) {

fprintf(fout, "%-25s", wordList[cnt++]);

}

fprintf(fout, "\n");

}

fprintf(fout, ". . . . .\n"); // 중간 다리

cnt = size - 50;

for (int i = 0; i < sample\_lines; i++) { // 뒷부분 50단어

for (int j = 0; j < words\_per\_line; j++) {

fprintf(fout, "%-25s", wordList[cnt++]);

}

fprintf(fout, "\n");

}

}

void selectionSortWordArray(char word[][MAX\_WORD\_LEN], int num\_words) {

char temp[MAX\_WORD\_LEN]; // 버퍼역할

int min\_pos;

for (int pos = 0; pos < num\_words; pos++) { // 기준

min\_pos = pos;

for (int j = pos + 1; j < num\_words; j++) { // 비교대상

if (strcmp(word[min\_pos], word[j]) > 0) min\_pos = j; // 작은지 판단

}

if (min\_pos != pos) { // 작을 때 교환

strcpy(temp, word[pos]);

strcpy(word[pos], word[min\_pos]);

strcpy(word[min\_pos], temp);

}

}

}

void quickSortWordArray(char word[][MAX\_WORD\_LEN], int num\_words, int left, int right) {

int pi, n\_pi;

char pi\_v[MAX\_WORD\_LEN] = "";

char temp[MAX\_WORD\_LEN] = "";

int i;

if (left >= right)return;

else if (left < right)pi = (left + right) / 2; // 피벗 설정

strcpy(pi\_v, word[pi]); // 피벗 값

strcpy(temp, word[pi]); // 마지막 값과 교환

strcpy(word[pi], word[right]);

strcpy(word[right], temp);

n\_pi = left; // 처음부터 시작

for (i = left; i <= (right - 1); i++) { // 하나씩 옮겨가며 판단

if (strcmp(word[i], pi\_v) <= 0) {

strcpy(temp, word[i]);

strcpy(word[i], word[n\_pi]);

strcpy(word[n\_pi], temp);

n\_pi++;

}

}

strcpy(temp, word[n\_pi]);

strcpy(word[n\_pi], word[right]);

strcpy(word[right], temp);

if (left < (n\_pi - 1))quickSortWordArray(word, num\_words, left, n\_pi - 1);

if ((n\_pi + 1) < right)quickSortWordArray(word, num\_words, n\_pi + 1, right);

}

void shuffleWordArray(char word[][MAX\_WORD\_LEN], int num\_words, int cnt) {

int t1, t2;

char t3[MAX\_WORD\_LEN] = "";

for (int i = 0; i < num\_words/2; i++) {

t1 = rand() % cnt;

t2 = rand() % cnt;

strcpy(t3, word[t1]);

strcpy(word[t1], word[t2]);

strcpy(word[t2], t3);

}

}

#endif