자료구조 알고리즘
(Term Project)

21211758 박인호
21211779 윤경식

● 목차

1. 목표

이번 term project의 목표는 단어가 무수히 많은 textfile에서 원하는 단어를 찾고 몇 행, 몇 열에 있는지 판단해서 그 점수를 임의로 부여하여 여러 개의 단어 조합이 들어왔을 때 순위를 부여하는 것을 목표로 했다. 또한 Hash Table을 구성하여 무수히 많은 데이터에 대해 정렬을 압축적으로 하여 Linear Search를 피하고 PC의 효율성을 올릴 수 있도록 하는 알고리즘을 구현하는 것을 기초로 하였다.

2. 기능 구상

우리 조는 기능을 구상하기 위해서 여러 가지 자료 형과 다양한 헤더파일들에 대해 살펴보았으며 이번 자료구조 알고리즘 결과물에 사용된 헤더파일에 대해 아래에 나열했다.

헤더파일	사용 용도
iostream	기본적인 cmd출력 외 여러 가지 기본적인 사항들을 위함
fstream	file의 입출력
string	string 데이터 형을 이용하기 위함
map	map 구조체를 이용해서 hashtable의 bucket만들기
vector	bucket에 linked list를 만들어 주기 위해 사용
cmath	점수를 매길 때 이용하는 pow, sqrt 간단한 수식을 위함
time.h	시간을 측정하기 위해 사용하였다. clock(), clock_t

< 표 1. 헤더파일과 사용 용도 >

이 기능들을 구상하기 위해 순차적으로 생각을 한 방향은 다음과 같다.

1) document.txt에서 한글을 불러와서 어느 행과 열에 위치한지에 대한 정보를 hashtable에 저장. -> fstream의 기능과 string, vector, map 사용
2) 새로운 map.txt라는 형태로 만들어진 hashtable을 저장하고 언제든지 불러 올 수 있도록 만듦. -> fstream (ofstream, ifstream)
3) 만들어진 hashtable을 불러 올수도 있고 새로운 hashtable을 만들수도 있도록 구성. (map <string,vector<string>>word_map)
4) 간단하게 글자를 찾을 수 있도록 만듦. <stod를 이용>
5) 여러 글자들의 조합을 찾을 수 있도록 만듦.
6) 여러 글자들의 조합을 이용하여 점수를 측정할 수 있도록 만들고 시간을 측정함.

< 그림 1. Term Project 수행 순서도 >

3. 소스 설명

소스에 대해 순차적인 해석을 통해 이해를 돕도록 하겠다. 우선 데이터를 Sorting하기 위해 사용한 Sorting은 QuickSort이며 vector에 대한 소팅을 위해 만들었다. Quick Sort는 과제를 통해 충분히 내용을 알아볼 수 있었기에 자세한 설명은 생략하겠다.

```
template <typename Iterator> void quickSort(Iterator start, Iterator end) {
    int size = (end - start);
    if (size <= 0)
        return;

    int pivotIndex = randomNumber(0, size);
    typename std::iterator_traits<Iterator>::value_type pivot = *(start + pivotIndex);

    if (pivotIndex != 0)
        std::swap(*(start + pivotIndex), *start);

    int i = 1;
    for (int j = 1; j < size; j++) {
        if (*(start + j) < pivot) {
            std::swap(*(start + j), *(start + i));
            i++;
        }
    }

    std::swap(*start, *(start + i - 1));

    quickSort(start, start + i - 1);
    quickSort(start + i, end);
}
int randomNumber(int start, int end) {
    srand(time(NULL));

    return rand() % end + start;
}
```

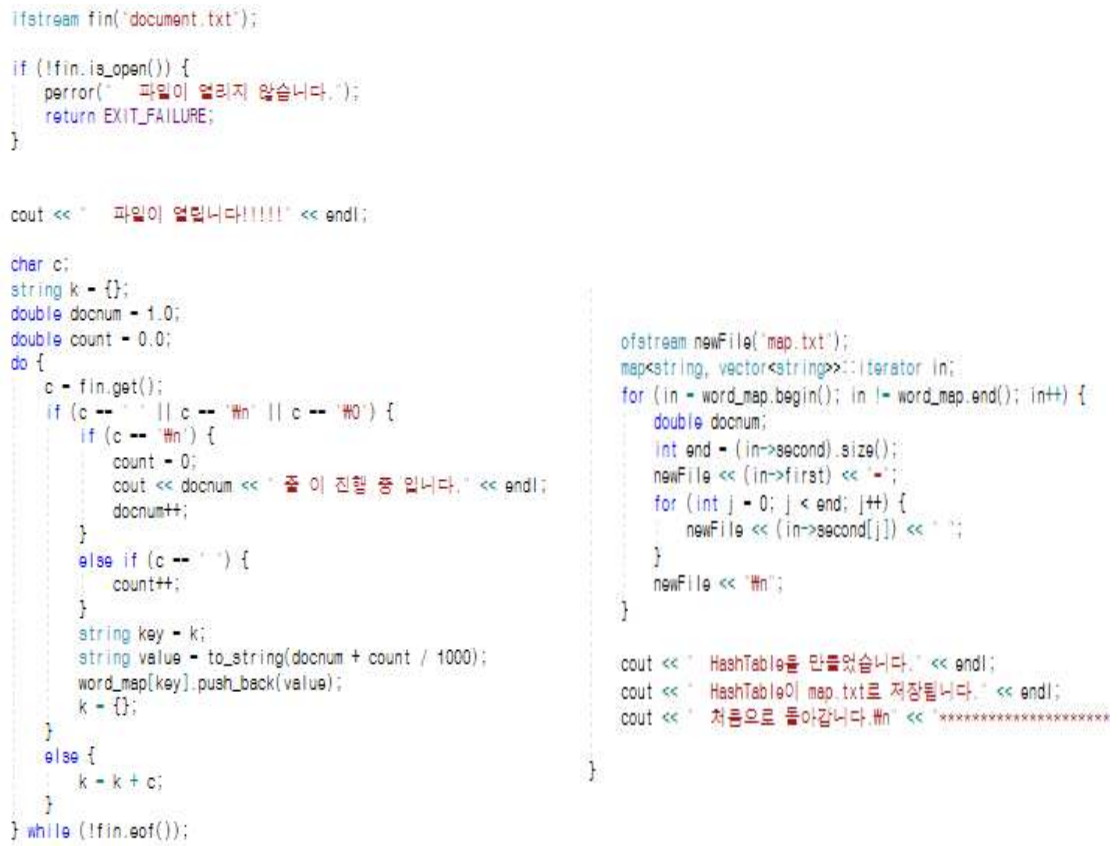
< 그림 2. Quick Sort 소스코드 >

이번 프로젝트에서 cmd를 깔끔하게 구성하기 위해서 다음과 같은 형태의 cout문들을 이용하여 화면을 구성하여 주었다. 또한 원하는 숫자를 입력받아 입력받은 숫자의 기능을 위해 do~while문을 이용해주어 원하는 입력이 들어왔을 때 다음과 같은 소스를 사용하였다.

```
int main() {
    // word_map 만들기
    map <string, vector<string>> word_map;
    int startNum;
    string findWord1;
    int findWord2;


    cout << "* 지금부터 텀프로젝트 창을 시작합니다.₩n₩n";
    cout << "* 조원 : 21211768박인호, 21211779 윤경식" << endl;
    cout << "₩n";
    cout << "* 이 프로그램은 document.txt를 통해 map으로 HashTable을 만듭니다.₩n";
    cout << "* 원하는 글자에 대해 몇 행에 있는지 찾습니다.₩n";
    cout << "* 여러 단어들의 조합에 대해 어떤 행에 있는지 결과를 알아 볼 수 있습니다." << endl;
    cout << "* 조합의 전부 혹은 몇 가지를 포함하는 행의 글자 사이 거리가 가까울 수록 행에 대한 낮은 점수로 매깁니다." << endl;
    cout << endl;
    cout << "*****************************************************************" << endl;
    cout << "1. document.txt를 이용하여 HashTable만들기고 map.txt로 저장하기" << endl;
    cout << "2. HashTable 불러오기" << endl;
    cout << "3. 원하는 한 글자 dataset.txt에서 찾기 ₩n";
    cout << "4. 원하는 갯수 글자 dataset.txt에서 찾기 ₩n";
    cout << "0. 화면을 종료합니다." << endl;
    cout << "*****************************************************************" << endl;

    do {
        cout << "₩n₩n ";
        cout << " 원하는 기능의 숫자를 입력하세요 : ";
        cin >> startNum;
        cout << endl;
        if (cin.fail()) {
            cout << "   숫자를 입력해야 합니다." << endl; // 에러 메시지 출력
            cin.clear(); // 오류스트림을 초기화
            cin.ignore(256, '₩n'); // 입력버퍼를 비움
        }
        if (startNum == 1) {
            word_map.clear();
            cout << "  HashTable을 만듭니다." << endl;
            // define the size of character array
```

< 그림 3. cmd 구성을 위한 소스코드 >

위의 cmd구성을 위한 소스코드를 살펴보았을 때 중간에 do{ }...의 형태를 볼 수 있다 이것은 끝에 while문을 선언해서 startNum을 입력받았을 때 어떤 식으로 작동을 할지에 대해 나타내 주는 문장이다. 이번 term project를 위해서 총 4가지 형태의 map을 이용해 주었으며 위의 word_map은 텍스트파일을 읽어왔을 때 그 행과 열의 정보를 key : 단어, value: xxxx.xxxx의 형태로 저장하는 map이다. value의 저장 정보는 정수부분은 몇 행인지, 소수부분은 해당 행의 몇 번째 열인지에 대한 정보를 담고 있다. 이러한 정보를 담는 소스코드에 대한 내용은 아래에 그림 4)를 통해서 확인해 볼 수 있다. document.txt의 정보를 읽고 그 결과에 대해 map으로 저장을 하며 이때 **가장 중요한 것은 document.txt가 UTF-8의 형태로 저장되면 안되고 반드시 ANSI의 형태로 저장되어야 한다는 것이다.** 우리가 작성한 소스코드는 character형의 데이터 형을 이용하여 file을 읽어드리는 선택을 했다. UTF-8은 한글을 읽을 수 있도록 만드는 코드임이 분명하지만, ANSI 형태로 된 메모장을 이용하여 character 형을 순서대로 읽어오는 소스코드를 작성했기 때문에 반드시 document.txt를 ANSI의 형태로 저장하여 읽어 주어야 한다.

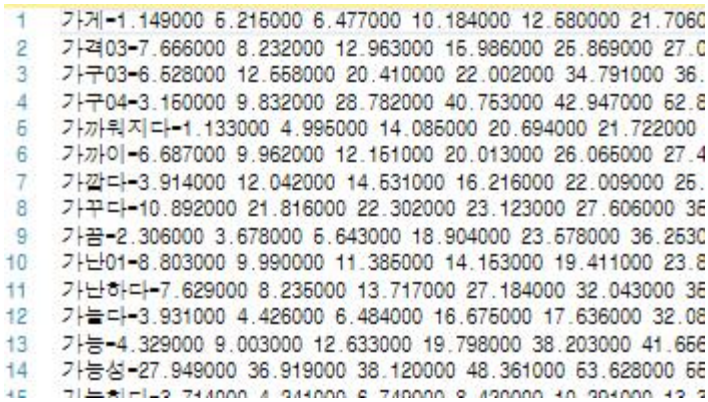


```
ifstream fin("document.txt");

if (!fin.is_open()) {
    perror("    파일이 열리지 않습니다.");
    return EXIT_FAILURE;
}

cout << "    파일이 열립니다!!!!!" << endl;

char c;
string k = {};
double docnum = 1.0;
double count = 0.0;
do {
    c = fin.get();
    if (c == ' ' || c == '\n' || c == '\0') {
        if (c == '\n') {
            count = 0;
            cout << docnum << "줄 이 진행 중 입니다." << endl;
            docnum++;
        }
        else if (c == ' ') {
            count++;
        }
        string key = k;
        string value = to_string(docnum + count / 1000);
        word_map[key].push_back(value);
        k = {};
    }
    else {
        k = k + c;
    }
} while (!fin.eof());
```

```
ofstream newFile("map.txt");
map<string, vector<string>>::iterator in;
for (in = word_map.begin(); in != word_map.end(); in++) {
    double docnum;
    int end = (in->second).size();
    newFile << (in->first) << "=";
    for (int j = 0; j < end; j++) {
        newFile << (in->second[j]) << " ";
    }
    newFile << "\n";
}

cout << "    HashTable을 만들었습니다." << endl;
cout << "    HashTable이 map.txt로 저장됩니다." << endl;
cout << "    처음으로 돌아갑니다.\n" << "**********************
}
```

< 그림 4. map구성과 map.txt저장을 위한 소스코드 >

왼쪽 소스코드는 document.txt를 읽어오고 파일을 map구조로 전환하기 위해 사용한 소스코드이다. 아래에 읽어보면 c=fin.get()이라는 구문이 있는데 여기서부터 if문을 이용하여 docnum, count 두 변수를 이용하여 docnum은 행의 숫자, count는 몇 번째 글자인지 판단하는 변수로 작성을 해주었다. docnum의 경우에는 c='\n'을 만났을 때 하나 증가시켜 주고, count의 경우에는 첫 번째 단어가 0으로 저장이 되고 c=' '(space)를 하나씩 만날 때 마다 더하기 1이 되도록 만들어 주었다. 이런 식으로 key에는 하나씩 받아오는 문자를 string으로 만들어 k로 저장했고 그 값을 key로 저장, value에는 docnum과 count값을 각각 정수, 소수 부분으로 저장하여 vector의 형태로 map에 삽입해 주었다. 오른쪽의 소스코드는 그렇게 만들어낸 map을 text로 저장하기 위한 소스코드이며 이 과정은 startNum이 1일 때 위와 같은 형태로 진행이 되도록 하였다. 저장되는 txt데이터는 map.txt로 저장되며 저장된 것에 대한 정보는 실행 project 폴더에서 확인할 수 있다. map.txt가 저장되는 정보는 아래의 사진에 첨부해 두었다.



```
1   가게-1.149000 5.215000 6.477000 10.184000 12.580000 21.706(
2   가격03-7.666000 8.232000 12.963000 15.986000 25.869000 27.(
3   가구03-6.528000 12.658000 20.410000 22.002000 34.791000 36.
4   가구04-3.160000 9.832000 28.782000 40.763000 42.947000 52.8
5   가까워지다-1.133000 4.995000 14.086000 20.694000 21.722000
6   가까이-6.687000 9.962000 12.161000 20.013000 26.065000 27.4
7   가깝다-3.914000 12.042000 14.631000 16.216000 22.009000 25.
8   가꾸다-10.892000 21.816000 22.302000 23.123000 27.606000 3(
9   가꿈-2.306000 3.678000 5.643000 18.904000 23.678000 36.253(
10  가난01-8.803000 9.990000 11.386000 14.163000 19.411000 23.8
11  가난하다-7.629000 8.235000 13.717000 27.184000 32.043000 3(
12  가늘다-3.931000 4.426000 6.484000 16.676000 17.636000 32.08
13  가능-4.329000 9.003000 12.633000 19.798000 38.203000 41.66(
14  가능성-27.949000 36.919000 38.120000 48.361000 63.628000 6(
```

< 그림 5. map.txt의 저장 정보 >

map.txt는 이렇게 key와 value의 형태를 txt로 저장하였으며 모든 단어들에 대해 몇 행에 몇 번째 정보인지 저장이 되어있다.

```
if (startNum == 2) {
    word_map.clear();
    cout << "  HashTable를 불러옵니다." << endl;

    pair< string, vector<string>> x;

    ifstream mapfile("map.txt");
    if (!mapfile.is_open()) {
        cout << "   저장된 HashTable이 없습니다.₩n";
    }
    else {
        word_map.clear();
        string mapword = {};
        char mapc;
        int num;
        vector<string> v_str;
        vector<int> v_int;
        do {
            mapc = mapfile.get();
            int row = 1;
            if (mapc == '=') {
                x.first = mapword;
                mapword = {};
            }
            else if (mapc == ' ') {
                v_str.push_back(mapword);
                mapword = {};
            }
            else if (mapc == '₩n') {

                mapword = {};
                x.second = v_str;
                word_map.insert(x);
                v_str = {};
                mapc = {};

            }
            else {
                mapword = mapword + mapc;

            }


        } while (!mapfile.eof());
    }
```

< 그림 6. map.txt 읽어 오기 >

왼쪽의 사진은 startNum의 입력이 2일 때 일어나는 형태로 이것은 이미 저장된 map.txt를 읽어 다시 Hashtable을 만들기 위한 소스코드이다. 기존의 word_map을 clear해주고 word_map에 위의 map.txt에서 보듯 '='의 왼쪽 부분과 오른쪽 부분을 갈라 따로 key, value 구분을 할 수 있도록 만들어 주었다.

mapword는 읽어오는 map의 단어이고 이것은 character형으로 읽어오기 때문에 하나씩 mapc를 더해주어 완전한 문자를 만들며 만약 mapc = '='를 만날 경우 그 mapword를 pair x의 first로 그리고 각각의 수치들을 second로 저장해서 map에 넣어주는 기능을 한다. map의 형태는 <string, vector< string>>의 형태이기 때문에 pair을 삽입해주면 추가적으로 vector에 저장되는 모습을 볼 수 있다.

```
if (startNum == 3) {
    cout << "  찾고자 하는 글자를 입력하세요 : ";
    cin >> findWord1;
    map<string, vector<string>> ::iterator iter;
    iter = word_map.find(findWord1);
    int end;
    if (word_map.empty()) {
        cout << "   저장된 word_map이 없습니다. 먼저 Hash Table를 만드세요" << endl;
    }
    else if (iter == word_map.end()) {

        cout << "   찾고자하는결과가 없습니다." << endl;

    }
    else {

        end = (iter->second).size();
        for (int j = 0; j < end; j++) {
            cout << floor(stod(word_map.find(findWord1)->second[j])) << "행 ";
            //   a = word_map.find(wantKey)->second[j];

        }

        cout << endl;

    }


    cout << endl;
    cout << "  처음으로 돌아갑니다.₩n" << "----------------------------------------------------------------------" << endl;
}
```

< 그림 7. startNum=3의 한 글자 찾기 >

위의 소스코드는 한 글자가 어디 행에 있는지 찾아볼 수 있는 코드이다. 이 코드를 이용하면 word_map에 글자가 있는지 없는지를 확인할 수 있다.

map<string,vector<string>>::iterator iter;
iter = word_map.find(findWord1)

위의 구문은 findWord1(찾고자 하는 단어)가 word_map에 있는지 없는지 판단할 수 있도록 만든 구문이다. 이후에 if 문을 순차대로 사용하여 만약 word_map이 empty라면 hash table이 없다고 뜨고, word_map의 끝까지 iter가 도착했으면 찾고자 하는 결과가 없다고 출력, 마지막으로 있을 경우에 대해서는 그 찾은 key 값에 대해 value vector들을 차례대로 출력할 수 있도록 만들 코드이다. 여기서 우리가 사용한 형태의 vector는 string형이 었기 때문에 stod를 이용하여 string형을 double형으로 변경해 주었다. 이렇게 변경한 형태는 xxxx.xxxx의 형태가 되며 소수점 아랫자리 수를 버리고 행의 정보만 얻기 위해서 floor함수를 이용해 주었다.

```
if (startNum == 4) {
    cout << "  찾고자 하는 글자가 몇개입니까? : ";
    cin >> findWord2;
    if (word_map.empty()) {
        cout << "   저장된 word_map이 없습니다. 먼저 Hash Table를 만드세요" << endl;
        cout << "  처음으로 돌아갑니다.₩n" << "*****************************************
        startNum = 100;
    }
    else {
        vector <double> vdouble = {};
        vector <int > vint = {};
        int end1;
        int wordnum=0;
        for (int i = 0; i < findWord2; i++) {
            cout << endl;
            cout << i+1 << " 번째 글자 를 입력하세요. : ";
            cin >> findWord1;
            cout << endl;
            map<string, vector<string>> ::iterator iter1 = word_map.find(findWord1);
            if (iter1 == word_map.end()) {
                cout << "찾고자 하는 단어의 결과가 없습니다." << endl;
                i = i - 1;
            }
            else {

                wordnum=wordnum+1;
                end1 = (iter1->second).size();
                for (int j = 0; j < end1; j++) {
                    double k = atod(word_map.find(findWord1)->second[j]);
                    vdouble.push_back(k);
                    int q = stoi(word_map.find(findWord1)->second[j]);
                    vint.push_back(q);
                    cout << floor(k) << "행 ";
                }
                cout << endl;

            }

        }
```

< 그림 7. startNum=4의 여러 글자 찾기 >

startNum=4일 때는 다음과 같은 기능을 하도록 구성하였다.

1) 단어 각기 입력 시 어떤 행에 있는지 확인.
2) 단어 조합에 대한 점수 출력.
3) 단어 조합에 대한 점수를 구하기까지 시간.

위의 그림 7)은 그 동작이 그림 6)과 같고 한 가지 다른 점은 vector vdouble, vint가 있다는 것인데 이것은 원하는 결과의 value를 찾게 되면 그 value들에 대해서 각각 int형과 double형으로 vector에 push하여 sorting을 위한 변수로 작동하게 된다. 이것을 sorting하는 이유는 한 단어를 입력할 때마다 그 단어가 어떤 행과 열에 있는지 나타나게 될 텐데 이것을 sorting을 하여 나타내게 된다면 …1.321, 1.425, 1.586, 1.669, 2.311, 2.435, 3.152… 와 같이 되게 되고 이렇게 sorting된 정보에 대해서 단어 조합이 포함된 행과 열에 대한 정보를 보다 쉽게 이해하고 읽을 수 있기 때문이다. 위의 것과 같은 경우에는 쉽게 1행에 321, 425, 586, 669의 자리에 단어가 위치하고 있다는 것을 알 수 있으며 이것은 0의 값부터 시작한 수치기 때문에 실질적으로는 322번째, 426번째, 587번째. 670번째 자리에 document.txt에 그 글자가 있다는 것을 알 수 있다.

```
clock_t tbegin, tend;

tbegin = clock();

quickSort(vdouble.begin(), vdouble.end());
quickSort(vint.begin(), vint.end());
cout << wordnum << "개의 글자에 대해서 점수를 찾습니다.₩n" << endl;

vector<double> compare;
map<int, int>rowcount;
pair<int, int> x;
int maxsize = 1;
for (int i = 0; i < vint.size(); i++) {
    map<int, int>::iterator it = rowcount.find(vint[i]); // v[*iter]값을 찾는데 찾있
    if (it != rowcount.end()) { //rowcount에서 v[*iter]에 해당하는 값을 찾았다는 뜻
        int b = rowcount[vint[i]]; // 기존의 v[*iter]값을 b로 받아와서
        rowcount[vint[i]] = b + 1; // +1를 해주겠다는 뜻이다.
        maxsize = maxsize < rowcount[vint[i]] ? rowcount[vint[i]] : maxsize;
    }
    else {
        x.first = vint[i];
        x.second = 1;
        rowcount.insert(x);
    }
}
map <int, vector<int>> maximumcount;
map <double, vector<int>> findscore;
pair<int, vector<int>> scorePair;
vector<int> scoreline;
for (map<int, int>::iterator rit = rowcount.begin(); rit != rowcount.end(); rit++)
{
    //cout << "    ";
    //std::cout << (rit)->first << "행이 " << (rit)->second << "개 있습니다.";
    //cout << endl;
    double k = (rit)->first;
    double n = (rit)->second;
    double score = 0;
    double standard = 0;
```

< 그림 8. startNum=4의 여러 글자 찾기(계속) >

위 사진의 소스코드를 살펴보면 시간을 측장하기 위한 clock_t,를 볼 수 있고 또한 quick sort를 이용하여 vector 2개를 sorting하는 것을 볼 수 있다. 이렇게 sorting된 vector들을 이용하여 rowcount란 map구조에 몇 행에 몇 개가 있는지에 대한 정보를 저장하게 되며 rowcount에는 key가 행, value가 단어의 조합 중, 행에 포함된 단어의 개수가 들어가게 된다. 아래 map 구조체가 2개 더 사용되게 되는데 maximumcount는 key에 단어 개수, value에 몇 행인지를 저장하는 구조체이며, findscore는 점수를 key에 그리고 행 정보를 value에 저장하며 value는 같은 점수를 가진 행이 존재할 수 있음으로 그 행들에 대해 vector값으로 저장하게끔 만들었다. 또한 점수는 오름차순으로 map에 저장됨으로 그 오름차순으로 저장된 점수들을 내림차순으로 1 ~ 10등까지만 출력하기 위해 새로운 vector인 vrow, vscore을 이용하여 내림차순으로 결과를 출력할 수 있도록 하였다. 위의 사진과 추가적으로 이용되는 소스코드에 대해서는 아래에 삽입하였다.

```
    for (int j = vdouble.size() - 1; j >= 0; j--)
    {
        if (floor(vdouble[j]) == k) {
            standard = pow((vdouble[j] - floor(vdouble[j])) * 1000, 2);
            score = standard + score;

        }
    }
    score = n * 1000 + 1000 - round(sqrt(score) / sqrt(n));
    findscore[score].push_back((rit)->first);
    maximumcount[(rit)->second].push_back((rit)->first);
}
vector<double> vscore;
vector<int> vrow;
for (map<double, vector<int>>::iterator fit = findscore.begin(); fit != findscore.end(); fit++)
{

    for (int j = 0; j < findscore[fit->first].size(); j++){
        //cout << "    ";
        //cout << (fit -> second)[j]<< "행이" << (fit->first) << "점입니다." << endl;
        vscore.push_back(fit->first);
        vrow.push_back((fit->second)[j]);
    }

}

cout << endl;
int k = 1;
for (int i = vrow.size()-1; i >= vrow.size() - 10; i--) {
    cout << "    " << k << "등은 " << vrow[i] << "행이 "<< vscore[i]<< "점입니다." << endl;
    k++;
}

cout << endl;
for (map<int, vector<int>>::iterator mit = maximumcount.begin(); mit != maximumcount.end(); mit++)
{
    cout << "    ";
    std::cout << (mit)->first << "개를 가진 행 : ";
    for (int i = 0; i < maximumcount[(mit)->first].size(); i++) {
        cout << ((mit)->second)[i] << "행 ";
    }
    cout << endl;
}

tend = clock();
cout << endl;
```

< 그림 9. startNum=4의 여러 글자 찾기(계속) >

```
        }
    }
} while (startNum != 0);
```

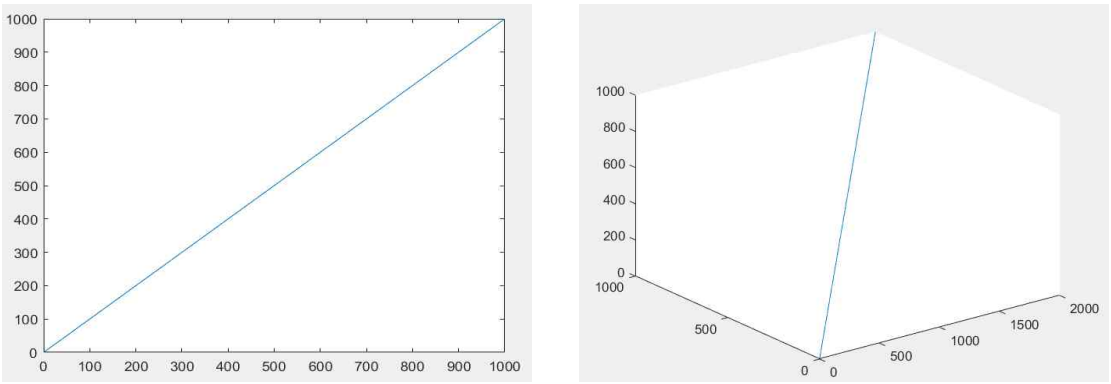
< 그림 10. do~while()문의 끝 >

do while문은 startNum = 0이 되면 cmd를 종료하게끔 만들었다. startNum이 글자가 들어오면 그 값들은 입력되지 않고 숫자를 입력해야 한다고 출력하게 만들었다. score 값은 직선의 기울기를 생각해 주었다. 0~999까지의 열의 값을 각각의 수치들이 가지게 되는데 다음 코드를 보면 사용된 이론을 알 수 있다.

```
standard = pow((vdouble[j] - floor(vdouble[j])) * 1000, 2);
score = standard + score;
score = n*1000 + 1000-round(sqrt(score)/sqrt(n))
```

standard 값은 0으로 초기화 되어 있고 열의 정보를 (vdouble[j] - floor(vdouble[j])) * 1000 수식을 통해 얻어 낼 수 있으며 이것을 제곱한 것이 standard가 되고 제곱한 것들 끼리 전부 다 더해내서 score값에 저장되게 된다.

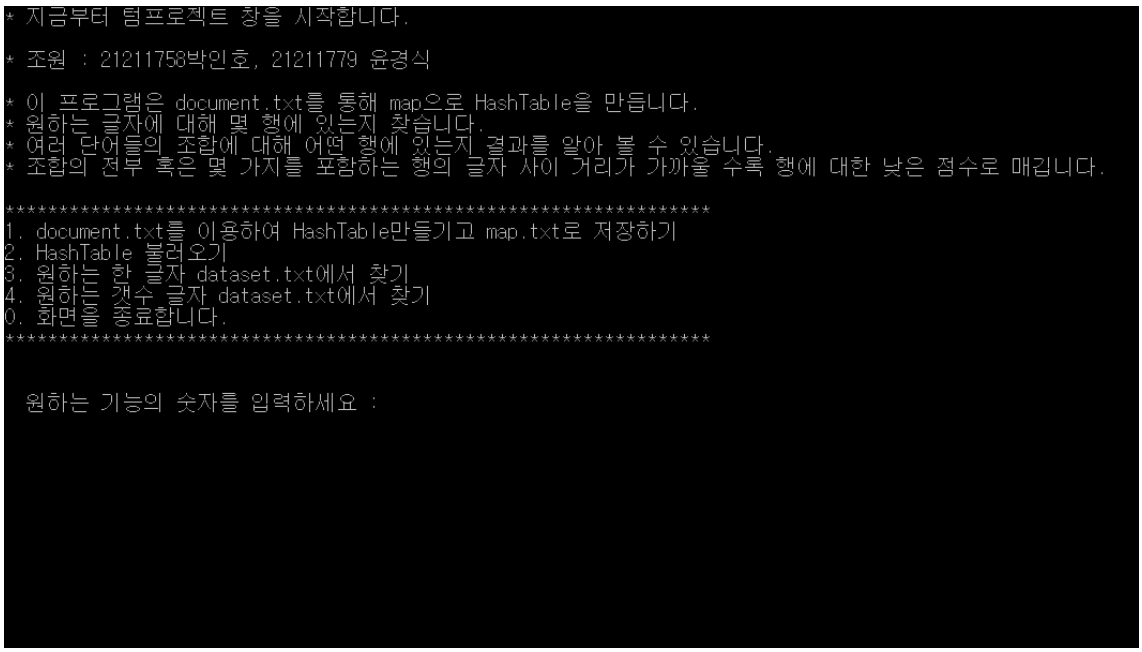
어떤 단어가 몇 열에 있는지에 대한 정보는 직선으로 표현을 할 수가 있다. 이 직선의 최대 길이는 $x_m = \sqrt{999^2 \times n}$ 이며 n은 찾은 단어의 개수가 된다. 가져오다, 기념일, 가다01은 67행에 3개의 단어가 다 포함되고 있으며 순서대로 795, 440, 975값을 가지며 이에 대한 길이는 $x_a = \sqrt{975^2 + 450^2 + 795^2}$ 이다. 이것을 sqrt(n)으로 나누어준 이유는 최대 값이 999가 되기 위함이며 1000의 자리 이상 침범하지 않게 만들어 주기 위함이다. 또 0, 0, 0기준으로 멀리 떨어져 있을수록 찾기가 오래 걸리기 때문에 길이가 길수록 점수가 낮아야 한다고 생각하여 이를 위해서 최대 길이 1000에서 x_a를 빼주었다. 또 천의 자리 수는 특정 행에 포함된 단어의 개수를 나타내었다. 예를 들어 어떤 단어조합의 특정 행의 점수가 4999이면 “특정 행에서 단어가 4개가 포함되며 열에 대한 점수를 매기자면 999이다”라고 해석해주면 되겠다.



< 그래프 1. 모든 값이 999일 때, 2차원 그래프(좌), 3차원 그래프(우) >

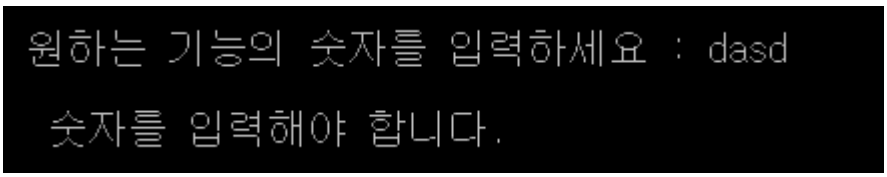
4. 화면 출력

지금부터는 화면 출력의 결과에 대해 살펴보고 결과가 옳은지 그른지를 확인하겠다. 우선 다음 화면은 시작화면이다. 각각의 기능들에 대한 정보와 이 프로그램이 무엇을 위한 프로그램인지 출력된다.



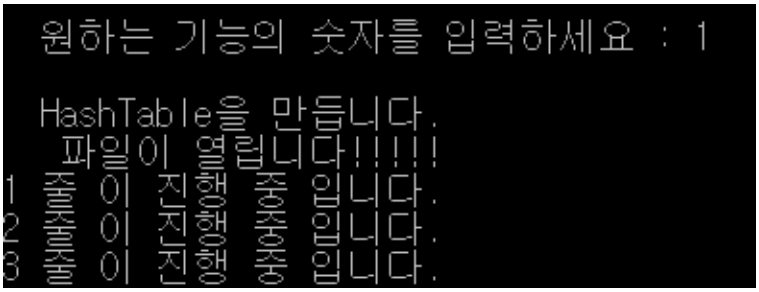
< 그림 11. 시작 화면 >

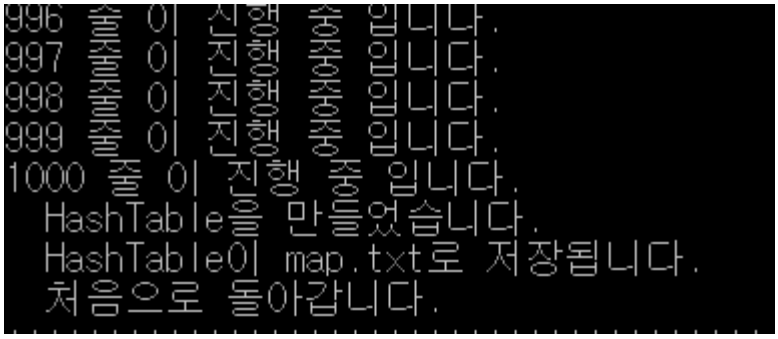
이 화면에서 숫자를 제외한 글자를 치게 되면 숫자를 입력해야 된다는 글자를 출력하게 만들었다.



< 그림 12. 글자 입력 화면 >

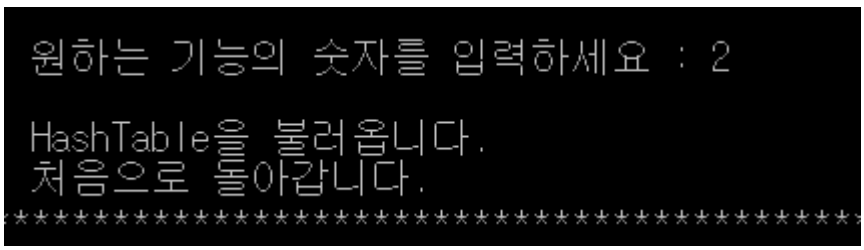
우선 숫자 1을 입력하여 Hash Table이 만들어 지는 화면을 살펴보자.



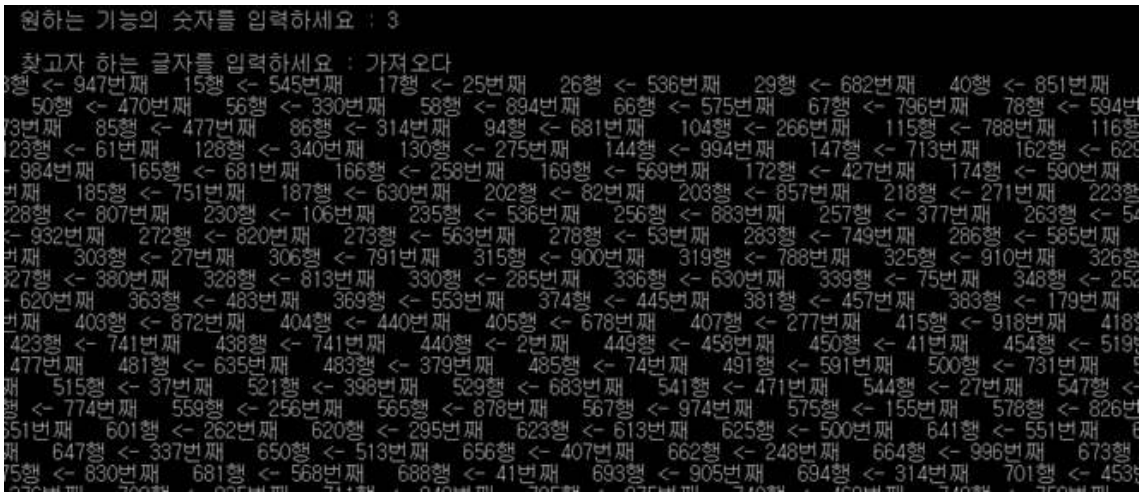


< 그림 13. Hash Table 생성 화면 >

이렇게 만들어진 Hash Table은 map.txt로 저장된다. 그 위치는 project파일 폴더에 저장이 되고 그 형태에 대해서는 그림 5) map.txt에 대한 정보를 살펴보면 알 수 있다. 다음은 기능 2를 이용하면 document.txt를 읽어 오지 않고도 map.txt를 이용하여 Hash Table을 불러 올 수 있다.

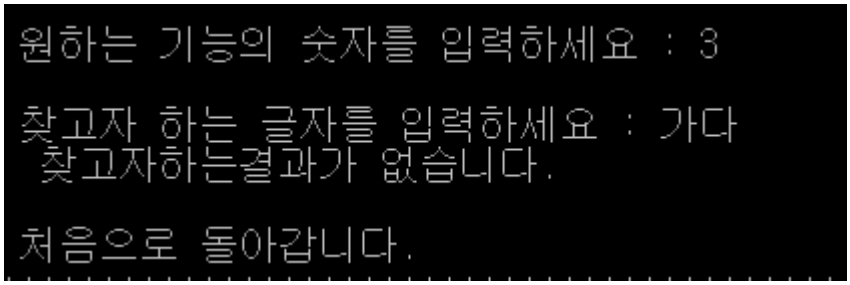


< 그림 14. startNum = 2일 때, Hash Table 생성 화면 >



< 그림 15. startNum = 3일 때, 단어 찾기 >

3을 입력하게 되면 찾고자 하는 글자를 입력할 수 있고 그 글자를 입력하면 어떤 행과 열에 포함되어 있는지 확인 할 수 있다.

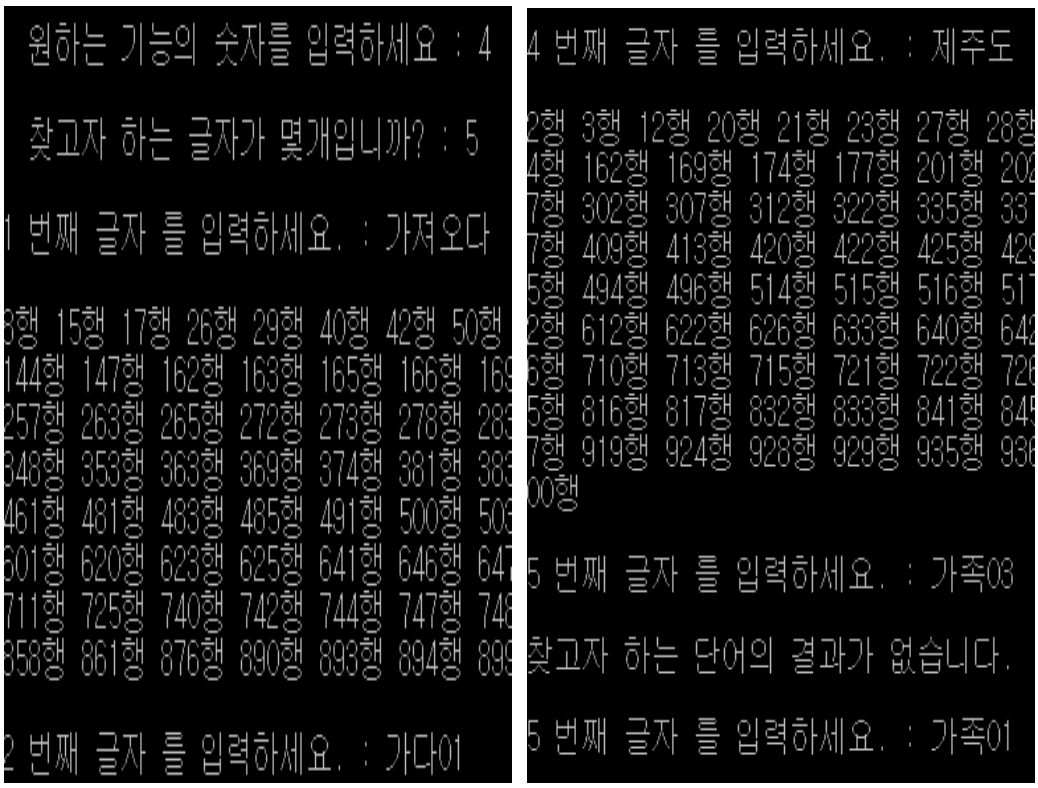


< 그림 15. startNum = 3일 때, 단어 찾기(계속) >

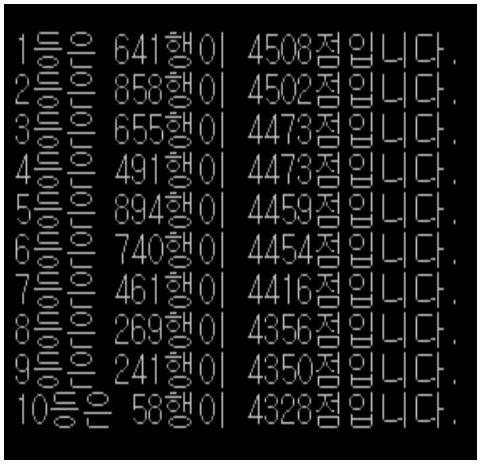
이렇게 찾는 단어가 없는 경우에는 위와 같은 출력 결과를 형성한다.

여러 단어 조합 찾기에 대해서는 5가지 단어를 찾아보도록 하자.

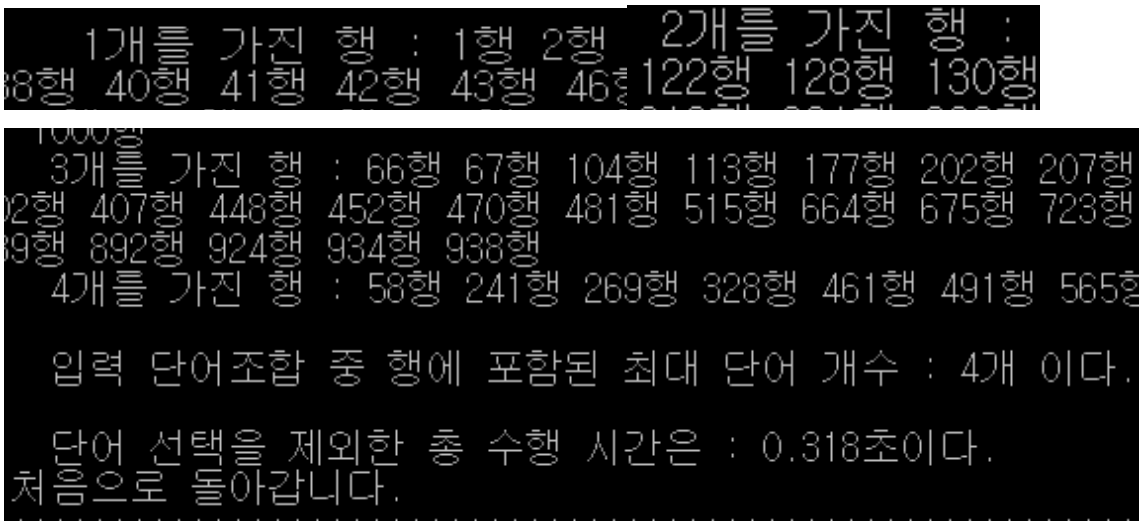
아래 사진처럼 가져오다, 제주도, 가다01, 가족01, 기념일에 대한 정보를 찾고 등수를 매기려고 했다. 이 과정에서 가족03의 경우 map에 찾고자 하는 단어가 없기 때문에 다시 (5 번째 글자를 입력하세요.)창이 뜨는 것을 볼 수 있다. 이렇게 원하는 단어들을 입력하게 되면 그림 17)에 볼 수 있듯이 단어들이 속한 행들과 그 행에 대한 점수가 1~10등까지 출력된다.



< 그림 16. startNum = 4일 때, 단어 조합 찾기(계속) >



< 그림 17. startNum = 4일 때, 점수 등수 >



< 그림 18. startNum = 4일 때, 출력화면 >

각 행에 대해 몇 개에 단어들을 포함하고 있는지 포함하는 개수에 따라 화면을 출력할 수 있도록 하였으며 이러한 과정까지 걸리는 총 시간을 출력할 수 있도록 하였다.

5. 결론 및 고찰

이번 텀 프로젝트를 통해서 map 구조와 vector를 이용하여 c++을 구현하고 sorting을 직접 프로그래밍에 사용함으로써 그 활용도를 깨달았다. 우리 조는 끊임없는 연구와 구글링을 통한 노력으로 완성적이고 안정적으로 이번 텀 프로젝트 과제를 마무리할 수 있었으며 결과에 대한 출력과 proposal을 모두 만족 할 수 있었다. 비록 어려운 과제였지만 어려운 과제를 해냄으로서 이론으로만 배웠던 c++의 코드 작성을 많이 배운 것 같다.

이번 과제에 가장 난해했던 점을 뽑자면 텍스트 파일을 읽어오는 과정이 아닐까란 생각이 든다. 함께 수업을 듣던 타 학생의 경우에는 string형으로 파일을 바로 읽어 와 한글을 출력하였다고 하는데 이 과정이 되지 않아서 매우 고민을 해보았지만 나중에 들어서야 그 원인이 UTF-8과 ANSI의 차이에서 나온 것임을 알 수 있었다. 우리 조는 원래 document의 자료가 방대하기 때문에 바로바로 확인을 할 수 있게 mytext.txt라는 텍스트 파일을 document자료로부터 100행까지만 복사한 것을 이용하여 주었었다. 그런데 복사를 하는 과정에서 UTF-8인코딩의 자료가 ANSI형태로 저장이 되었고 ANSI형태를 이용해서 문제를 해결하였으나 document자체의 인코딩이 UTF-8형이기 때문에 원인을 찾기 위해 많은 시간을 들였었다.

도움이 되었던 점은 알고리즘 수업과 직접적으로 관련이 있는 과제여서 수업을 이해하기 쉬웠다는 것이다. 이번 텀 프로젝트의 vector와 single-linked list, map과 hash table, 직접적인 sort 사용이 가장 수업을 이해하는데 도움을 주었다.

이러한 과정들을 거쳐서 자료구조 알고리즘의 텀 프로젝트를 마칠 수 있었으며 한학기간 고생한 우리 조에게 고생했다는 격려의 말과 교수님께도 감사드린다는 인사를 간접적으로 전하면서 텀 프로젝트를 마무리한다.

*Quick Sort Algorism 출처 : https://gist.github.com/lawliet89/4106929