
자료구조 설계과제 보고서

[자료구조 학생 관리 프로그램]

010-8578-7628

12131489 김영훈

12131489@inha.edu

1. 개요

이 프로그램은 인하대 재학생들의 정보를 관리한다. 또한 해시테이블 자료구조의 이중해싱 기법을 이용하여 평균 수행속도를 빠르게 하고 효율적으로 작동한다. 이 프로그램을 설계하기 위해 해시테이블에 대한 이해와 이중해싱을 설계하는 방법에 대해 숙지하고 있어야 한다. 이 프로그램은 운영체제 Windows10에서 개발도구 Microsoft Visual Studio Community 2015를 이용하여 C++ 언어로 개발하였다.

2. 필요한 자료구조 및 기능

Student 구조체	Hash 클래스
<pre> struct Student { int studentNumber; string name; string department; int grade; bool switchBit; } </pre>	<pre> class Hash { // Hash의 기능이 담겨져 있는 클래스 private: int N, M, size; // N = 배열의 크기를 나타내는 소수, M = 2차 해시함수에 사용도 int probe; // 수행한 탐사 횟수 Student* arr; // 학생 정보가 담긴 배열 public: Hash() { // 생성자, 0으로 초기화 N = 0; M = 0; size = 0; } bool empty(); bool sizeFull(); bool arrSizeSet(int N, int M); int h1(int k); int h2(int k); void insert(int studentNumber, string name, string department, int grade); bool limitLength(int studentNumber, string name, string department, int grade); int insertIndexSet(int index, int studentNumber); void print(int studentNumber); void printIndexSet(int index, int studentNumber); void remove(int number); void removeIndexSet(int index, int studentNumber); }; </pre>
h1, h2 함수	
<pre> int Hash::h1(int k) { // 1차 해시함수 int index = k % N; return index; } int Hash::h2(int k) { // 2차 해시함수 int index = (M - k % M); return index; } </pre>	

Double Hash를 구현하기 위해 Hash 클래스를 만들었다. 클래스에는 배열의 크기를 나타내는 소수 N과 2차 해시함수에 사용되는 소수 M이 존재하고, 학생 정보를 저장하는 학생구조체형 배열이 있다. 또한 1차 해싱함수 h1(k), 2차 해싱함수 h2(k)와 삽입, 삭제, 출력의 기능을 수행하는 함수들과 포함 되어있다. h1(k) 함수는 배열의 크기 N을 이용해 처음 삽입될 인덱스를 정해주는 함수이다. h2(k) 함수는 1차 해싱함수가 실행된 이후 소수 M을 이용하여 인덱스 값을 재설정 해주는 함수이다. Hash의 기능은 크게 삽입, 삭제, 출력으로 3가지가 있다. 삽입 기능은 배열의 인덱스에 입력한 학생의 정보를 삽입해준다. 삭제 기능은 입력한 학번에 해당하는 학생의 정보를 삭제하는 기능이다. 출력 기능은 입력한 학번의 학생의 정보를 출력해주는 기능이다. 이 외에도 N, M과 배열의 크기를 설정하는 arrSizeSet 함수와 sizeFull, empty 등의 함수들이 있다.

3. 기능별 알고리즘 명세

1. 삽입

```
void Hash::insert(int studentNumber, string name, string department, int grade) {    // 삽입
    probe = 0;

    if (sizeFull() == 1 || limitLength(studentNumber, name, department, grade) == 1) { // 크기가 꽉차거나 학생 정보가 조건과 안맞을 경우 함수 종료
    } else {
        int index = h1(studentNumber);           // 삽입할 인덱스를 h1 함수를 통해 설정
        index = insertIndexSet(index, studentNumber); // 삽입할 인덱스에 이미 값이 있는지 확인하고, 있다면 인덱스 h2 함수를 통해 재설정
        if (index != -1) {                       // 인덱스가 -1이 아닐 경우 입력받은 학생의 정보를 해당 인덱스에 삽입한다.
            arr[index].studentNumber = studentNumber;
            arr[index].name = name;
            arr[index].department = department;
            arr[index].grade = grade;
            arr[index].switchBit = 0;             // 삭제된 자리가 아니기 때문에 0으로 초기화 시켜준다.
            size++;                             // 사이즈를 1 추가한다.
        }
    }
}

bool Hash::limitLength(int studentNumber, string name, string department, int grade) { // 입력받은 값이 초기조건과 안맞을 경우 1을 반환하는 함수
    if (studentNumber < 10000000 || studentNumber >= 100000000 || name.length() > 20 // 학번 8자리, 이름 20byte, 학과 20byte, 학년 1~4 정수가 아닐 경우
        || department.length() > 20 || grade < 1 || grade > 4) {
        outputFile << "추가할 수 없음 " << probe << endl; // 추가할 수 없음을 나타내고
        return 1; // 1 반환
    }
}

int Hash::insertIndexSet(int index, int studentNumber) { // h2 함수를 통해 인덱스를 재설정해주는 함수
    int temp = -1; // 반환할 인덱스의 값을 저장한다.
    bool stay = 0; // stay가 1이 되면 temp의 값을 더 이상 바꾸지 못하게 해준다.
    while (1) {
        probe++; // 반복될때마다 탐사횟수(probe)를 1 증가시킨다.
        if (arr[index].studentNumber == NULL) { // 인덱스가 비었고
            if (stay == 0) // stay가 0이면
                temp = index; // temp를 해당 index 값으로 설정한다.

            if (arr[index].switchBit == 1) { // 해당 인덱스가 값이 삭제된 자리면 다음 인덱스에 중복되는 값이 있을 수 있으므로 확인해 줘야한다.
                stay = 1; // stay를 1로 지정하여 다음 인덱스가 NULL이어도 인덱스 값이 변하지 못하게 한다.
                index = (index + h2(studentNumber)) % N; // 인덱스 값에 h2(k) 함수로 결정된 값을 더해 다음 인덱스로 설정 후 반복한다.
            }
        } else { // 해당 인덱스가 값이 삭제된 자리가 아니면
            if (txtType == 1) // txtType이 command.txt일때
                outputFile << probe << endl; // 탐사횟수를 출력한다.
            break; // 반복문을 빠져나간다.
        }
    }
    else if (arr[index].studentNumber == studentNumber) { // 입력받은 학번과 같은 학번이 인덱스에 존재할 때
        outputFile << "추가할 수 없음 " << probe << endl; // 추가할 수 없음을 나타내고
        temp = -1; // temp를 -1로 초기화하고
        break; // 반복문을 빠져나간다.
    }
    else { // 인덱스에 값이 있다면
        index = (index + h2(studentNumber)) % N; // 인덱스 값에 h2(k) 함수로 결정된 값을 더해 재설정
    }
}
return temp; //temp의 값을 반환한다.
}
```

insert 함수

- insert 함수를 통해 처음 probe(탐사 횟수)를 0으로 설정한다.
 - SzieFull, limtelLength 함수를 통해 배열이 가득 차거나, 입력 받은 학생의 정보가 주어진 초기조건과 맞지 않은 경우 1을 반환하여 값이 삽입되지 않도록 한다.
 - 그렇지 않을 경우 학번을 1차 해싱함수를 이용하여 해당되는 인덱스의 값을 지정한다. 그리고 **insertIndexSet** 함수를 통해 지정된 인덱스에 삽입할 수 있는지 여부를 판단한다. insertIndexSet으로 설정된 인덱스의 값이 -1이 아니면 해당 인덱스에 입력 받은 학생 정보를 삽입하고, switchBit를 0으로 설정하고 size를 1 증가시킨다. 이후 함수는 종료된다.

insertIndexSet 함수

- insertIndexSet 함수는 필요에 따라 h2 함수를 통해 인덱스를 재설정해주는 함수이다. int형 변수 temp를 통해 반환할 인덱스의 정수값을 저장한다. stay 변수는 1이 되면 temp의 값을 더 이상 바꾸지 못하게 한다.
 - 지정된 인덱스의 학번이 비었을 경우
 - * stay의 값이 0이면 temp에 index 값을 넣는다.
 - * 인덱스의 switchBit가 1이면 해당 빈자리는 원래 있던 값이 삭제된 자리이므로 h2 함수를 통해 다음 인덱스에 중복된 학번이 있는지 확인이 필요하다. 따라서 stay를 1로 바꿔 더 이상 temp의 값이 다음 호출된 함수의 index 값으로 변하지 못하게 하고, 인덱스의 값을 h2 함수를 통해 바꾼다. 그리고 while 문을 통해 이 작업을 반복한다.
 - * switchBit가 1이 아니면 txtType이 "command.txt"이면 1이므로 탐사횟수를 출력한다. 그리고 반복문을 빠져나가 함수가 temp의 값을 반환하고 종료된다.
 - 해당 인덱스의 학번과 인자로 받은 학번의 값이 일치한다면, 추가할 수 없음을 출력하고, temp의 값을 -1로 지정하고 반복문을 종료한다.
 - 해당 인덱스에 값이 존재하고, 같은 학번이 아니라면, h2함수를 통해 index의 값을 재설정하고 반복문을 수행한다.

시간 복잡도 : 삽입하고자 하는 인덱스에 계속 값이 있다면(충돌 발생) 함수는 NULL인 인덱스를 찾기 위해 계속 반복 될 것이다. 최악의 경우 배열의 모든 인덱스를 탐색하여 NULL인 자리를 찾으므로 배열의 크기 N 만큼 탐사한다. 따라서 최악의 경우 수행시간은 $O(N)$ 이다.

공간 복잡도 : 사용자에게 배열의 공간을 지정 받으면서 공간을 확보 받고 모두 NULL로 초기화된 상태이므로, 학생의 정보를 삽입해도 공간의 변화는 없다. 최악의 경우 공간 복잡도는 $O(1)$ 이다.

2. 삭제

```
void Hash::remove(int studentNumber) { // 삭제
    if (empty() == 1) {} // 배열이 비었을 경우 함수를 종료한다.
    else {
        probe = 1;

        int index = h1(studentNumber); // 삭제할 인덱스를 h1 함수를 통해 설정
        removeIndexSet(index, studentNumber);
    }
}

void Hash::removeIndexSet(int index, int studentNumber) { // 학번의 정보를 삭제하거나 h2 함수를 통해 인덱스를 재설정해주는 함수
    if (arr[index].studentNumber == studentNumber) { // 인덱스의 학번과 학번이 일치하면
        arr[index] = { NULL }; // 인덱스를 NULL값으로 초기화시킨다.
        arr[index].switchBit = 1; // 삭제된 자리임을 표시해준다.
        size--; // 배열의 원소 개수 -1
        outputFile << probe << endl; // 프로브 횟수 출력
    }
    else { // 그 외의 경우
        probe++; // 프로브 횟수 +1
        index = (index + h2(studentNumber)) % N; // 인덱스 값에 h2(k) 함수로 결정된 값을 더해 재설정
        if (N == probe || (arr[index].switchBit == 0 && arr[index].studentNumber == NULL)) { // 탐사횟수가 N번째이거나, 삭제된 자리가 아니면서 해당 인덱스의 학번이 비었을 경우
            outputFile << "삭제할 수 없음 "; // 찾는 학번이 없어 삭제할 수 없음을 나타내고 탐사 횟수를 출력한다.
            outputFile << probe << endl;
        }
        else {
            removeIndexSet(index, studentNumber); // 수정된 인덱스의 값을 파라미터로 받은 함수를 호출한다. (재귀함수)
        }
    }
}
```

remove 함수 (입력한 학번과 같은 학번의 인덱스의 학번을 찾아 인덱스의 정보를 비운다.)

- Empty 함수를 이용하여 배열의 원소가 없을 경우(size = 0) 1을 반환하여 함수를 종료한다.
- 탐사횟수는 1로 시작하고 배열의 인덱스를 h1 함수를 통해 설정 후 **removeIndexSet 함수**를 실행한다.

removeIndexSet 함수

- 이 함수에서는 인덱스의 학번과 입력 받은 학번이 같을 경우
 - 인덱스를 NULL로 초기화한다. 그리고 값이 삭제된 자리임을 표시하기 위해 switchBit를 1로 바꿔 표시한다. Size를 1 감소시키고 탐사횟수를 출력하고 함수를 종료한다.
- 인덱스의 학번이 입력 받은 학번이 아닐 경우 탐사횟수를 1 증가시키고, 다음 인덱스의 학번과 비교하기 위해 인덱스를 h2 함수를 통해 바꾼다.
 - 만약 탐사횟수가 N(배열의 크기)이 되거나, switchBit가 0이고 인덱스의 학번이 NULL이면 삭제할 수 없음과 탐사 횟수를 출력하고 함수를 종료한다. 그 이유는 모든 인덱스를 탐사했을 경우 탐사횟수가 N이 되기 된다. 또한 switchBit가 0이고 인덱스의 학번이 비었으면 찾는 학번이 존재하지 않는다. 따라서 이 조건문은 삭제하고자 하는 학번은 배열의 존재하지 않음을 뜻한다.
 - 위 두가지 경우가 아닐 경우에는 h2를 통해 수정된 인덱스를 가지고 이 함수를 다시 호출해 실행한다.

시간 복잡도 : 배열의 인덱스가 가득 찬 상태에서 값을 삭제하기 위하여 탐색한다고 할 때, 계속해서 삭제할 값을 찾지 못하면 모든 인덱스를 탐색 할 것이다. 탐색은 배열의 크기인 N번 하게 된다. 따라서 최악의 경우 수행시간은 $O(N)$ 이 걸린다.

공간 복잡도 : 삭제할 값을 계속하여 찾지 못할 경우 수행 횟수는 배열의 크기 N만큼 된다. 재귀 함수이므로 함수를 N번 호출하게 된다. 따라서 최악의 경우 공간 복잡도는 $O(N)$ 이다.

3. 출력

```
void Hash::print(int studentNumber) {    // 출력
    probe = 1;
    int index = h1(studentNumber);        // 출력할 인덱스를 h1 함수를 통해 설정
    printIndexSet(index, studentNumber);  // 인덱스의 값과 학번의 값을 넘겨 함수 실행
}

void Hash::printIndexSet(int index, int studentNumber) {    // 학번의 정보를 출력하거나 h2 함수를 통해 인덱스를 재설정해주는 함수
    if (arr[index].switchBit == 0 && arr[index].studentNumber == studentNumber) // 삭제된 자리가 아니고, 인덱스의 학번과 입력받은 학번이 같을 때
        outputFile << arr[index].studentNumber << " " << // 인덱스의 학생 정보와 탐사 횟수를 출력한다.
        arr[index].name << " " << arr[index].department << " " <<
        arr[index].grade << " " << probe << endl;
    else {    // 그 외의 경우
        probe++;    // 탐사횟수를 1 증가시키고
        index = (index + h2(studentNumber)) % N;    // 인덱스 값에 h2(k) 함수로 결정된 값을 더해 재설정
        if (N == probe || arr[index].switchBit == 0 && arr[index].studentNumber == NULL) { // 탐사횟수가 N번째이거나, 삭제된 자리가 아니면서 해당 인덱스의 학번이 비었을 경우
            outputFile << "없음 " << probe << endl;    // 찾는 학번이 없음을 나타내고 탐사 횟수를 출력한다.
        }
        else    // 그 외의 경우
            printIndexSet(index, studentNumber);    // 수정된 인덱스의 값을 파라미터로 받은 함수를 호출한다. (재귀함수)
    }
}
```

print 함수

- 함수를 시작하면서 탐사횟수를 1 증가시킨다. H1 함수를 이용하여 탐색할 인덱스를 설정한다. 그리고 printIndexSet 함수를 호출 시킨다.

printIndexSet 함수

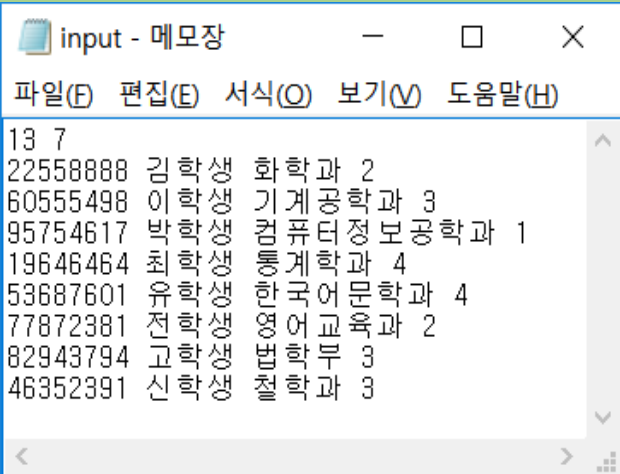
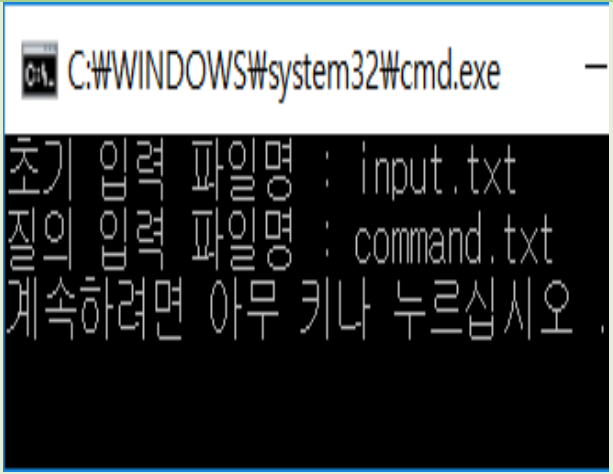
- switchBit가 0이면서 인덱스의 학번과 찾고있는 학번이 일치할 경우, 해당 인덱스의 정보를 출력한다.
- 이 외의 경우 탐사횟수를 증가시키고, index를 h2 함수를 통해 다음 인덱스로 설정한다.
 - switchBit가 0이고 인덱스의 학번이 NULL이면, 찾는 학번이 존재 하지 않으므로, 없음과 탐사횟수를 출력하고 함수를 종료한다.
 - 위 두가지 경우가 아닐 경우에는 h2를 통해 수정된 인덱스를 가지고 이 함수를 다시 호출하여 실행한다.

시간 복잡도 : 배열의 인덱스가 가득 찬 상태에서 값을 출력하기 위해 탐색한다고 할 때, 계속해서 출력해야 할 학번을 찾지 못하면 모든 인덱스를 탐색한다. 이 때 탐색은 배열의 크기인 N번 하게 되고 최악의 수행시간은 $O(N)$ 이 된다.

공간 복잡도 : 출력할 값을 계속하여 찾지 못할 경우 수행 횟수는 배열의 크기 N만큼 된다. 재귀 함수이므로 함수를 N번 호출하게 된다. 따라서 최악의 경우 공간 복잡도는 $O(N)$ 이다.

4. 인터페이스 및 사용법

초기 입력 파일 설정,

input.txt	cmd
	

	김학생	전학생	이학생		고학생	최학생	신학생			박학생	유학생	
0	1	2	3	4	5	6	7	8	9	10	11	12

K(학번)	이름	H1(k)	H2(k)	index						Probe	
22558888	김학생	1	5	1						1	
60555498	이학생	3	4	3						1	
95754617	박학생	10	7	10						1	
19646464	최학생	6	2	6						1	
53687601	유학생	6	5	6	11						2
77872381	전학생	2	1	2	1						
82943794	고학생	11	4	11	2	6	10	1	5	6	
46352391	신학생	7	6	7						1	

실행창에서 초기 입력 파일과 질의 입력 파일의 이름을 입력한다. 메인 함수에서 초기 입력 파일을 먼저 불러들여 각 학생정보들을 hash 배열의 인덱스에 저장한다. 이후 질의 입력 파일을 불러 각 단계를 수행한다.

Step 1. 학생 검색하기

command.txt	output.txt

	김학생	전학생	이학생		고학생	최학생	신학생			박학생	유학생	
0	1	2	3	4	5	6	7	8	9	10	11	12

K(학번)	이름	H1(k)	H2(k)	index						Probe	
22558888	김학생	1	5	1							1
60555498	이학생	3	4	3							1
95754617	박학생	10	7	10							1
19646464	최학생	6	2	6							1
53687601	유학생	6	5	6	11						2
77872381	전학생	2	1	2							1
82943794	고학생	11	4	11							2
46352391	신학생	7	6	7							1

S를 입력하면 main 함수에서 hash의 switch문을 통해 출력하는 case로 간다. 입력 받은 학번을 인자로 하는 print 함수를 실행시킨다.

- 53687601 : h1 함수를 통해 인덱스 6를 탐색한다. 인덱스 6은 찾고자 하는 학번(53687601)이 아니므로 h2 함수를 통해 다음 인덱스를 찾는다. H2 함수를 거치면 인덱스 11이 탐색하게 되는데 찾고자 하는 학번과 같은 학번이 저장되어 있으므로 해당 인덱스의 정보를 출력시킨다. 이 때의 탐사 횟수는 2번이 된다.

- 19646464 : h1 함수를 통해 인덱스 6을 탐색한다. 인덱스 6에 저장된 학번과 일치하므로 정보를 출력한다. 이 때 탐사 횟수는 1번이다.

- 82943794 : h1 함수를 통해 인덱스 11을 탐색한다. 인덱스 11의 학번과 일치 하지 않으므로 h2 함수를 실행시킨다. 인덱스 2를 탐색하게 되고 일치하지 않으므로, 다시 h2 함수로 인덱스의 값을 변경하여 탐색한다. 인덱스가 5가 되었을 때 찾고자 하는 학번과 일치하므로 학생의 정보를 출력한다. 이 때의 탐사 횟수는 6번이다.

- 94717719 : h1 함수를 통해 인덱스 5를 탐색한다. 학번이 일치하지 않으므로 h2 함수를 통해 재결정된 인덱스 7을 탐색한다. 일치하지 않아 다음 인덱스 9를 탐색한다. 인덱스 9는 삭제되었던 인덱스도 아니면서 학생의 정보를 가지고 있지 않기 때문에 입력한 학번의 학생이 없음과 탐색 횟수를 출력한다.

Step 2. 학생 추가하기

command.txt	output.txt

	김학생	전학생	이학생		고학생	최학생	신학생		정학생	박학생	유학생	
0	1	2	3	4	5	6	7	8	9	10	11	12

K(학번)	이름	H1(k)	H2(k)	Index						Probe	
22558888	김학생	1	5	1							1
60555498	이학생	3	4	3							1
95754617	박학생	10	7	10							1
19646464	최학생	6	2	6							1
53687601	유학생	6	5	6							11
77872381	전학생	2	1	2							1
82943794	고학생	11	4	11							2
46352391	신학생	7	6	7							1
37470006	정학생	2	7	2							9

77872381	허학생	2	1	2	인덱스 2 에 동일한 학번 발견					1(추가불가)
----------	-----	---	---	---	-------------------	--	--	--	--	---------

i를 입력하면 main 함수에서 hash의 switch문을 통해 삽입하는 case로 간다. 입력 받은 학생의 정보를 삽입하는 insert 함수를 실행시킨다.

- 정학생 : Insert 함수에서 h1 함수에 의해 결정된 인덱스 2에 간다. 인덱스 2에는 전학생이 있으므로, h2 함수가 실행되어 인덱스 9에 간다. 인덱스9에는 인덱스가 비었으므로 정학생의 정보가 추가된다.

- 허학생 : Insert 함수에서 h1 함수에 의해 결정된 인덱스 2에 간다. 인덱스 2에 허학생과 같은 학번인 전학생이 존재하므로, 동일한 학번은 추가시킬 수 없으므로 삽입할 수 없다. 따라서 추가할 수 없음을 나타내고 탐사 횟수를 출력하고 함수를 종료한다.

Step 3. 학생 삭제하기

command.txt	output.txt
<pre> s 53687601 s 19646464 s 82943794 s 94717719 i 37470006 정학생 정치외교학과 4 i 77872381 최학생 정보공학계열 1 d 19646464 d 17290644 </pre>	<pre> 53687601 유학생 한국어문학과 4 2 19646464 최학생 통계학과 4 1 82943794 고학생 법학부 3 6 없음 3 2 추가할 수 없음 1 1 삭제할 수 없음 3 </pre>

	김학생	전학생	이학생		고학생	최학생	신학생		정학생	박학생	유학생	
0	1	2	3	4	5	6	7	8	9	10	11	12

K(학번)	이름	H1(k)	H2(k)	Index						Probe		
22558888	김학생	1	5	1							1	
60555498	이학생	3	4	3							1	
95754617	박학생	10	7	10							1	
19646464	최학생	6	2	6							1	
53687601	유학생	6	5	6	11							2
77872381	전학생	2	1	2	1							
82943794	고학생	11	4	11	2	6	10	1	5	6		
46352391	신학생	7	6	7							1	
37470006	정학생	2	7	2							9	2

17290644		7	7	7	1	8				3
----------	--	---	---	---	---	---	--	--	--	---

d를 입력하면 main 함수에서 hash의 switch문을 통해 삭제하는 case로 간다. 입력 받은 학번을 인자로 하는 remove 함수를 실행시킨다.

- 19646464 : remove 함수에서 h1 함수에 의해 결정된 인덱스 6에 간다. 입력한 학번과 인덱스 6의 학번과 일치하므로 최학생의 데이터를 모두 NULL로 비워주고 switchBit를 1로 설정해주어 삭제된 자리임을 표시한다.

- 17290644 : remove 함수에서 h1 함수에 의해 결정된 인덱스 7로 간다. 학번이 일치하지 않으므로 h2 함수를 통해 인덱스를 다시 설정해 인덱스 1로 간다. 일치하지 않아 다시 인덱스를 h2로 다시 설정하여 인덱스 8로 간다. 인덱스 8은 학생의 정보가 없는 NULL 상태이고, switchBit가 0이므로 삭제된 자리가 아니므로 더 이상 탐색할 필요가 없다. 입력한 학번이 인덱스에 존재하지 않기 때문에 삭제할 수 없음과 탐색 횟수를 출력하고 종료한다.

Step 4. 학생 정보 삭제 후 다시 검색하기

command.txt

```
command - 메모장
파일(F) 편집(E) 서식(O) 보기(V) 도움말(H)
s 53687601
s 19646464
s 82943794
s 94717719
i 37470006 정학생 정치외교학과 4
i 77872381 허학생 정보공학계열 1
d 19646464
d 17290644
s 53687601
s 19646464
```

output.txt

```
output - 메모장
파일(F) 편집(E) 서식(O) 보기(V) 도움말(H)
53687601 유학생 한국어문학과 4 2
19646464 최학생 통계학과 4 1
82943794 고학생 법학부 3 6
없음 3
2
추가할 수 없음 1
1
삭제할 수 없음 3
53687601 유학생 한국어문학과 4 2
없음 2
```

	김학생	전학생	이학생		고학생	최학생	신학생		정학생	박학생	유학생	
0	1	2	3	4	5	6	7	8	9	10	11	12

K(학번)	이름	H1(k)	H2(k)	Index						Probe		
22558888	김학생	1	5	1							1	
60555498	이학생	3	4	3							1	
95754617	박학생	10	7	10							1	
19646464	최학생	6	2	6							1	
53687601	유학생	6	5	6	11							2
77872381	전학생	2	1	2	1							
82943794	고학생	11	4	11	2	6	10	1	5	6		
46352391	신학생	7	6	7							1	
37470006	정학생	2	7	2							9	2

19646464		6	2	6	8					3
----------	--	---	---	---	---	--	--	--	--	---

s를 입력하면 main 함수에서 hash의 switch문을 통해 출력하는 case로 간다. 입력 받은 학번을 인자로 하는 print 함수를 실행시킨다.

- 53687601 : print 함수에서 h1 함수에 의해 인덱스 6을 탐색한다. 인덱스 6은 학생의 정보가 없는 NULL 상태지만 SwitchBit가 1인 삭제된 자리인 상태이므로, 찾는 값이 다음 인덱스에 있을 수도 있다. 그러므로 h2에 의해 설정된 인덱스 11을 탐색한다. 인덱스 11의 학번과 일치하므로 학생의 정보를 출력한다.

- 19646464 : print 함수에서 h1 함수에 의해 인덱스 6을 탐색한다. 위와 같은 이유로 다음 탐색을 실행 해야 하므로 h2에 의해 결정된 인덱스 8을 탐색한다. 인덱스 8은 삭제된 자리가 아니고 NULL인 상태이므로 찾는 값은 배열에 저장 되어있지 않다. 따라서 없음과 탐사 횟수를 출력한다.

Step 5. 학생 정보 삭제 후 새로운 학생 추가하기

command.txt	output.txt
<pre> s 53687601 s 19646464 s 82943794 s 94717719 i 37470006 정학생 정치외교학과 4 i 77872381 허학생 정보공학계열 1 d 19646464 d 17290644 s 53687601 s 19646464 i 11600582 장학생 의예과 2 </pre>	<pre> 53687601 유학생 한국어문학과 4 2 19646464 최학생 통계학과 4 1 82943794 고학생 법학부 3 6 없음 3 2 추가할 수 없음 1 1 삭제할 수 없음 3 53687601 유학생 한국어문학과 4 2 없음 2 2 </pre>

	김학생	전학생	이학생		고학생	장학생	신학생		정학생	박학생	유학생	
0	1	2	3	4	5	6	7	8	9	10	11	12

K(학번)	이름	H1(k)	H2(k)	Index						Probe	
22558888	김학생	1	5	1						1	
60555498	이학생	3	4	3						1	
95754617	박학생	10	7	10						1	
19646464	최학생	6	2	6						1	
53687601	유학생	6	5	6	11						2
77872381	전학생	2	1	2	1						
82943794	고학생	11	4	11	2	6	10	1	5	6	
46352391	신학생	7	6	7						1	
37470006	정학생	2	7	2						9	2
11600582	장학생	6	7	6						0	2

i를 입력하면 main 함수에서 hash의 switch문을 통해 삽입하는 case로 간다. 입력 받은 학번을 인자로 하는 insert 함수를 실행시킨다.

- 장학생 : insert 함수에서 h1 함수에 의해 결정된 인덱스 6을 탐색한다. 인덱스 6은 학생의 정보가 없는 NULL 상태이지만, 삭제된 자리(switchBit =1)이므로 장학생과 같은 학번의 정보가 h2 함수에 결정된 다음 인덱스에 존재 할 수 있다. 따라서 다른 인덱스에 중복된 학번이 있는지 확인해줘야 한다. H2 함수로 인덱스 0이 되고, 인덱스 0은 NULL값이고 삭제된 자리가 아니므로 더 이상 다른 인덱스를 탐색 할 필요가 없게 된다. 중복된 학번이 있는지 확인했으므로 인덱스 6에 새로 받은 학생의 정보를 저장한다.

Step 6. 새로 추가한 학생 정보 검색하기

command.txt	output.txt
command - 메모장 파일(F) 편집(E) 서식(O) 보기(V) 도움말(H) <pre>s 53687601 s 19646464 s 82943794 s 94717719 i 37470006 정 학생 정치외 교학과 4 i 77872381 허 학생 정보공학계열 1 d 19646464 d 17290644 s 53687601 s 19646464 i 11600582 장 학생 의 예과 2 s 11600582</pre>	output - 메모장 파일(F) 편집(E) 서식(O) 보기(V) 도움말(H) <pre>53687601 유 학생 한국어문학과 4 2 19646464 최 학생 통계학과 4 1 82943794 고 학생 법학부 3 6 없음 3 2 추가할 수 없음 1 1 삭제할 수 없음 3 53687601 유 학생 한국어문학과 4 2 없음 2 2 11600582 장 학생 의 예과 2 1</pre>

	김학생	전학생	이학생		고학생	장학생	신학생		정학생	박학생	유학생	
0	1	2	3	4	5	6	7	8	9	10	11	12

K(학번)	이름	H1(k)	H2(k)	Index						Probe	
22558888	김학생	1	5	1						1	
60555498	이학생	3	4	3						1	
95754617	박학생	10	7	10						1	
53687601	유학생	6	5	6						11	2
77872381	전학생	2	1	2						1	
82943794	고학생	11	4	11	2	6	10	1	5	6	
46352391	신학생	7	6	7						1	
37470006	정학생	2	7	2						9	2
11600582	장학생	6	7	6						1	

s를 입력하면 main 함수에서 hash의 switch문을 통해 출력하는 case로 간다. 입력 받은 학번을 인자로 하는 print 함수를 실행시킨다.

- 11600582 : h1 함수를 통해 결정된 인덱스 6에 접근한다. 인덱스 6에 일치하는 학번이 있으므로 학번에 대한 학생의 정보를 출력한다.

5. 평가 및 개선 방향

1. 장점

이중해싱을 이용하여 탐색, 삽입, 삭제에 대한 평균수행시간이 $O(1)$ 만큼 빠르다. 또한 데이터가 배열에 골고루 분포될 경우 효율적으로 자원을 관리할 수 있다. 사용자의 용도에 따라 속도를 빠르게 하거나, 공간을 절약할 수 있다. 배열의 크기를 사용자가 직접 설정하기 때문에 필요한 공간을 예측할 수 있고, 그 만큼 낭비되는 공간을 줄일 수 있다. 반대로 공간이 더 많이 확보될수록 탐색횟수가 줄고 해당 되는 인덱스를 더 빨리 찾을 수 있으므로 속도가 빨라질 수 있다.

2. 단점

배열에 데이터들이 많아질수록 충돌이 잦아질 수 있다. 그에 따라 탐색횟수가 증가하고 프로그램 성능이 저하될 수 있다. 빠른 탐색을 위해서는 배열의 크기를 크게 설정해야 한다. 또한 배열의 각 원소들이 구조체로 되어있기 때문에 상대적으로 많은 양의 메모리 공간이 필요로 하게 된다. 그리고 확률은 낮지만 모든 키 값이 충돌을 일으킬 경우 $O(N)$ 시간이 걸릴 수 있다.

3. 향후 개선 방향

이 프로그램을 설계할 때 insert가 자주 일어나지 않는다면 배열의 타입을 구조체의 포인터형으로 하는 것이 효율적일 것이다. 구현한 프로그램에서는 배열의 각 인덱스들이 구조체로 되어 있다. 그래서 사용하지 않는 인덱스에 대해서도 메모리를 할당 받고 있다. 이와 같은 방법을 사용한다면 배열이 차지하는 공간을 충분히 줄일 수 있을 것이다.