14.1 해싱이란?

해싱 O(1)의 시간 안에 탐색

키에 산술적인 연산을 적용해 항목이 저장되어 있는 테이블의 주소를 계산 🡪 항목 접근

사전 자료형에 주로 쓰임

14.2 추상 자료형 사전

사전의 개념 (키, 값)쌍의 집합

(키, 값)쌍을 삭제, 저장, 검색 가능

= 맵, 테이블

ex) 영어사전 키 영단어

값 단어 설명

항목들을 키에 의해 식별하고 관리함

사전의 연산 ADT14.1 객체: 일련의 (key, value) 쌍의 집합

연산:

add(key, value) ::= (key, value)를 사전에 추가한다

delete(key) ::= key에 해당하는 (key, value)를 찾아서 삭제, 관련된 value를 반환, 탐색이 실패하면 NULL반환

search(key) ::= key에 해당되는 value를 찾아서 반환, 탐색이 실패하면 NULL반환

사전 구조를 가장 효율적으로 구현할 수 있는 것 = 해싱

14.3 해싱의 구조

해싱 어떤 항목의 키만을 가지고 바로 항목이 들어있는 배열의 인덱스를 결정하는 기법

해시함수 키를 입력받아 해시주소를 생성하는 함수

해시주소는 해시테이블의 인덱스로 사용됨

ex) 영어사전 키 단어

인덱스 해시함수(단어)

해시테이블 인덱스 해시함수(키)

m개의 버킷 인덱스는 0~m-1까지

해시함수의 출력값은 0~m-1이어야 함 서로 다른 키가 같은 해시주소로 사상되는 경우가 있음

충돌 서로 다른 2개의 키 k1, k2에 대해 h(k1) = h(k2)인 경우

동의어 충돌이 발생한 경우의 서로 다른 키 k2, k2

s개의 슬롯 1개의 버킷에는 s개의 슬롯이 있음

해시주소의 충돌가능성 고려

오버플로우 충돌이 버킷에 할당된 슬롯 수보다 많이 발생해 버킷에 더 이상 항목을 저장할 수 없는 경우

ex) s=1이라면 충돌이 발생하면 오버플로우 발생

이상적인 해싱 ex) 학번을 키로 이용

알고리즘14.1 add(key, value) :

index = hash\_function(key)

ht[index] = value

search(key) :

index = hash\_function(key)

return ht[index]

실제의 해싱 해시테이블의 크기가 제한되어 있으므로 하나의 키당 해시테이블에서 하나의 공간을 할당할 수가 없음

보통의 경우, 키가 매우 많음 🡪 키를 해시테이블의 크기로 나눠 나머지를 취함, h(k) = k mod m 🡪 0~m-1까지의 숫자 생성 🡪 많은 충돌 발생

Quiz 1. 0 100 710(충돌3)

1

2 812

3 53 763(충돌1) 123(충돌4, 오버플로우1)

4 374

5 225 65(충돌2)

6

7 557

8 818

9 19

14.4 해시함수

좋은 해시함수의 조건 1. 충돌이 적어야 함

2. 해시함수 값이 해시테이블의 주소 영역 내에서 고르게 분포되어야 함

3. 계산이 빨라야 함

제산 함수 나머지 연산자를 이용해 테이블의 크기로 나눈 나머지를 해시 주소로 사용하는 방법

h(k) = k mod M k 키

M 테이블의 크기

대부분 소수 선택

구현 int hash\_function(int key) {

int hash\_index = key % M;

if (hash\_index < 0) hash\_index += M;

return hash\_index;

}

폴딩 함수 키가 해시테이블의 크기보다 더 큰 정수일 경우 사용

폴딩 키를 몇 개의 부분으로 나누어 더하거나 부울연산 실행

이동폴딩 키를 여러 부분으로 나눈 값을 더함 ex) 123 203 241 112 20 123 + 203 + 241 + 112 + 20 = 699

경계폴딩 키의 이웃한 부분을 거꾸로 더해 해시 주소를 얻음 ex) 123 203 241 112 20 123 + 302 + 241 + 211 + 20 = 897

폴딩함수 키를 여러 부분으로 나누어 모두 더한 값을 해시 주소로 사용

중간 제곱 함수 키를 제곱한 다음, 중간의 몇 비트를 취해서 해시 주소 생성

비트 추출 함수 M = 2k일 때, 키를 이진수로 간주해 임의의 위치의 k개의 비트를 해시주소로 사용하는 것

해시주소의 집중현상이 일어날 가능성이 높음

숫자 분석 방법 숫자로 구송된 키에서 각각의 위치에 있는 수의 특징을 미리 알고 있을 때 사용

편중되지 않는 수들을 해시 테이블의 크기에 적합한만큼 조합

탐색키가 문자열일 경우 주의할 점 문자열의 문자에 정수를 할당해 바꾸기 ex) 유니코드, 아스키코드

Quiz 1. 이동폴딩 1234 + 567 = 1801

경계폴딩 1234 + 765 = 1999

14.5 개방주소법

충돌과 오버플로우 충돌 서로 다른 키를 갖는 항목들이 같은 해시주소를 가지는 현상

오버플로우 충돌이 발생하고 해시주소에 더 이상 빈 버킷이 없으면 오버플로우 발생

해결법 개방주소법 충돌이 일어난 항목을 해시테이블의 다른 위치에 저장함

선형조사법, 이차조사법, 이중해싱법, 임의조사법

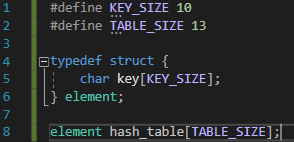
체이닝 해시테이블의 하나의 위치가 여러 개의 항목을 저장할 수 있도록 해시테이블의 구조 변경

선형조사법 개방주소법 특정버킷에서 충돌이 발생하면 비어있는 버킷을 찾는 방법

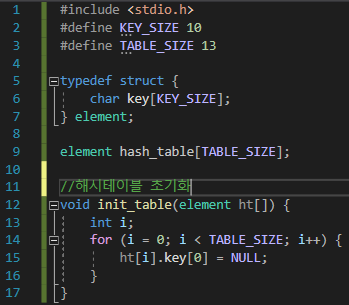
조사 비어있는 공간을 찾는 것

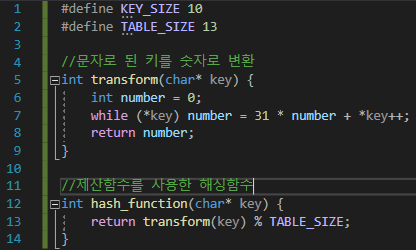
선형조사법 h(k)에서 오버프로우가 발생되었다면 h(h(k)+1)조사

항목의 저장을 위해 빈 버킷을 순차적으로 탐색해감

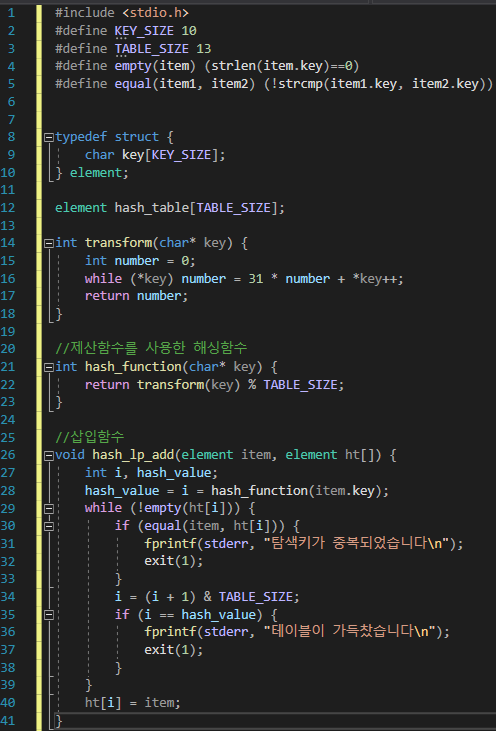
구현 프로그램14.1 

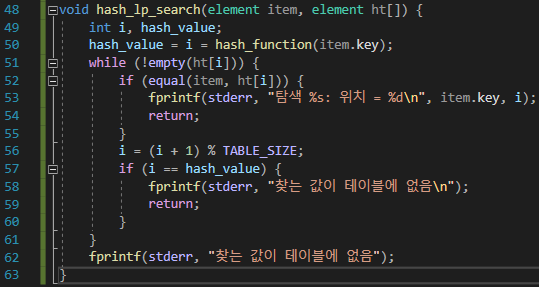
해시테이블은 1차원배열

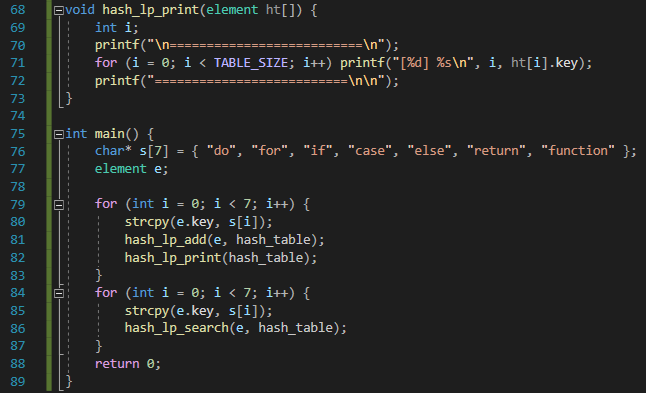
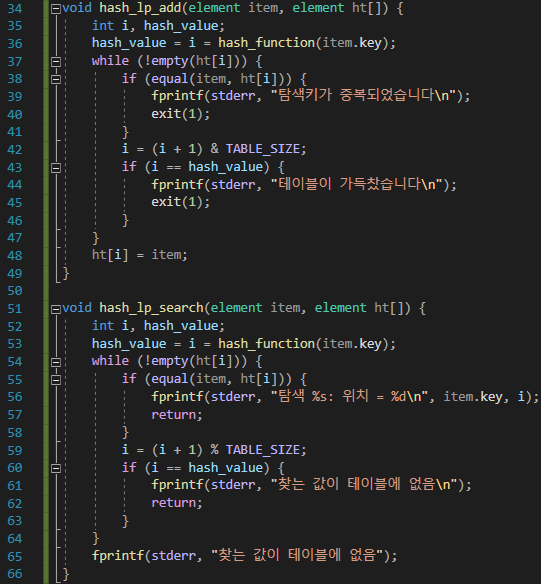
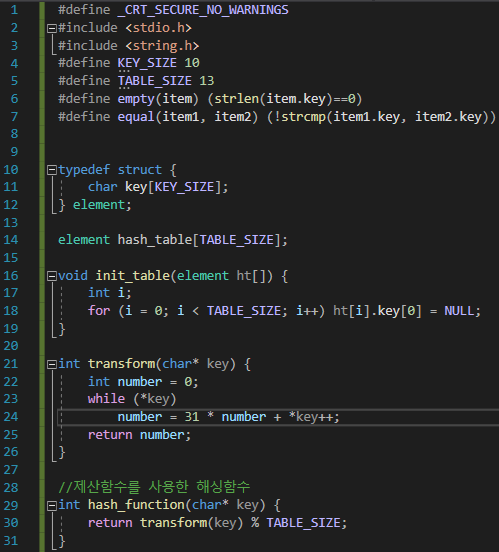
프로그램14.2 

프로그램14.3 

군집화 한번 충돌이 시작되면 그 위치에 항목들이 집중되는 현상

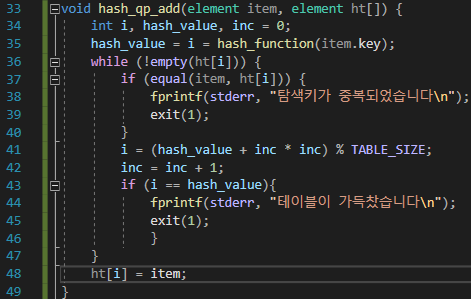
프로그램14.4 

프로그램14.5 

프로그램14.6  문제 탐색시간이 길어짐

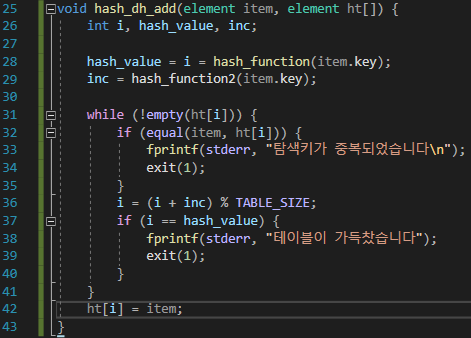
이차조사법 선형조사법과 유사

다음 조사할 위치 h(k) + inc\*inc mod M = 0,1,2,…,M-1

프로그램14.7 

이중해싱법 오버플로우가 발생함에 따라 항목을 저장할 다음 위치를 결정할 때, 별개의 다른 해시 함수를 이용하는 방법

해시테이블의 크기는 반드시 소수가 되어야 함

프로그램14-8 

Quiz 1. 0 2.30

1 8. 60

2 3.12

3 6. 33

4 1.74

5 5. 24

6 7. 52

7 9.10

8 4.28

9 10.20

14.6 체이닝

체이닝 오버플로우문제를 연결리스트로 해결

각 버킷에 삽입과 삭제가 용이한 연결리스트 할당

충돌이 발생하면 새로운 노드 생성 저장

항목 삽입 키들의 중복 허용 🡪 처음에다 삽입

키들의 중복 허용X 🡪 맨 끝에다 삽입

구조체 #define TABLE\_SIZE 7

typedef struct {

int key;

} element;

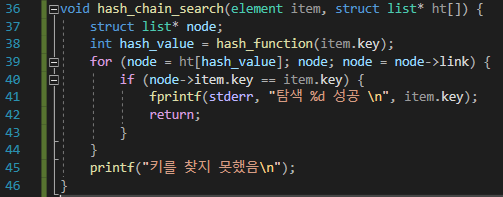
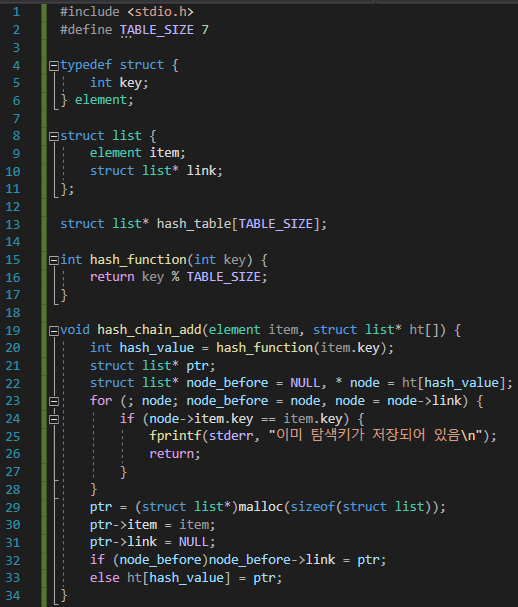
struct list {

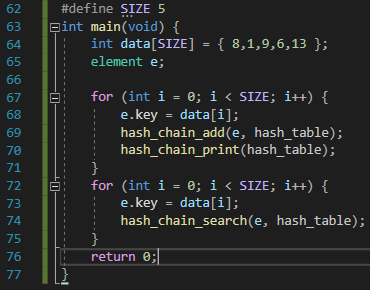
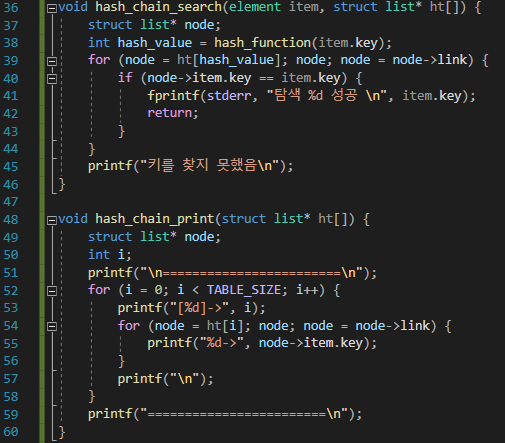
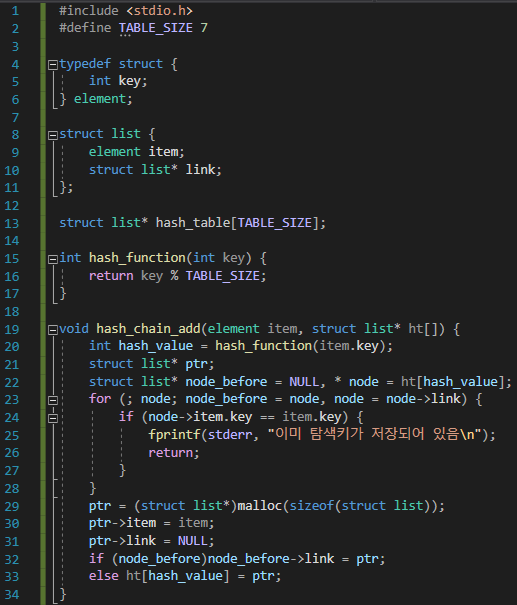
element item;

struct list\* link;

};

struct list\* hash\_table[TABLE\_SIZE];

프로그램14.9 

프로그램14.10 

Quiz 1. 0 2.30 8.60 9.10 10.20

1

2 3.12 7.52

3 6.33

4 1.74 5.24

5

6

7

8 4.28

9

14.7 해싱의 성능 분석

해싱에서 제일 중요한 연산 탐색

해싱 최선 탐색 O(1)

삽입 O(1)

삭제 O(1)

최악 탐색 O(n)

삽입 O(n)

삭제 O(n)

14.8 해싱의 응용 분야

1. 데이터베이스 인덱싱

2. 심볼테이블 구현

3. 인터넷 검색 엔진

연습문제

1. 2

2. c

3. (1) 0 2.44

1 1.12

2 3.12

3 4.88

4 5.23

5 7.11

6 6.94

7 8.39

8 10.16

9 9.20

10 11.5

3. (2) 0 2.44

1 1.12

2 3.13

3 10.16

4 4.88

5 5.23

6 6.94

7 8.39

8 11.5

9 7.11

10 9.20

3. (3) 0 2.44

1 1.12

2 3.13

3 4.88

4 8.39

5 9.20

6 5.23

7 11.5

8 10.16

9 7.11

10 6.94

3. (4) 0 2.44 🡪 4.88 🡪 7.11

1 1.12 🡪 5.23

2 3.13

3

4

5 10.16 🡪 11.5

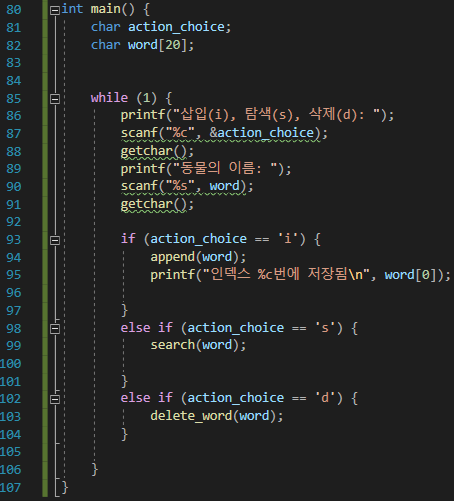
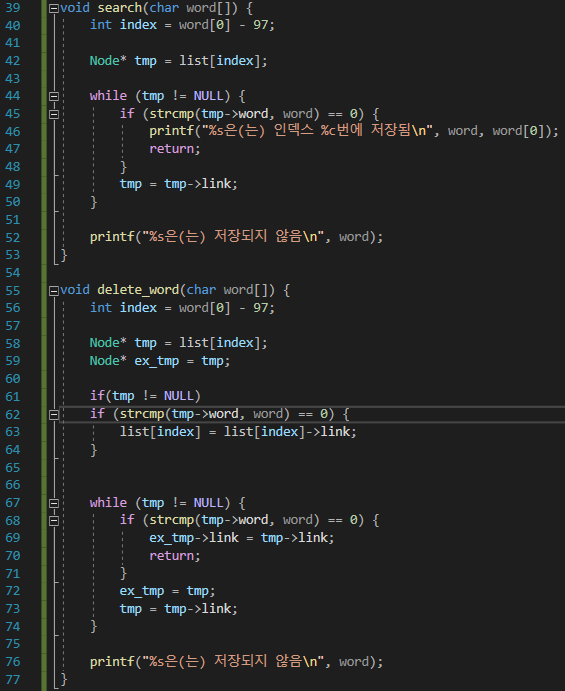
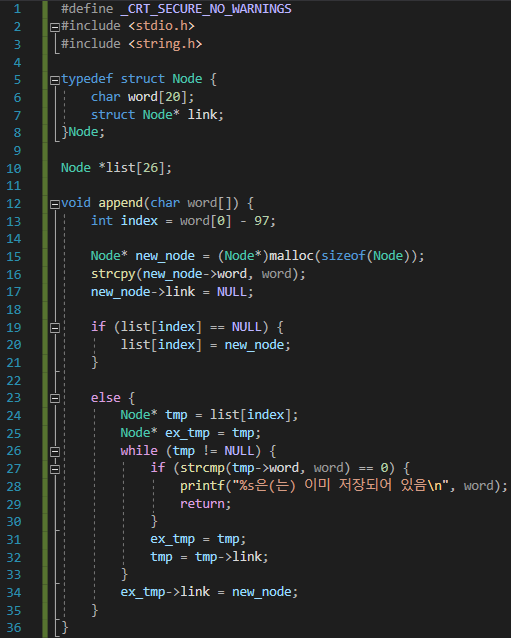
6 6.94 🡪 8.39

7

8

9 9.20

10

4. 

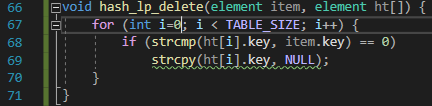
5. 3

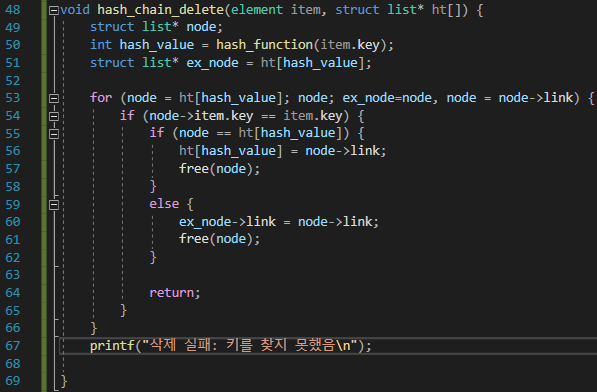
6. 모든 픽셀데이터를 더해 몇 비트만을 골라 해시함

7. (1) 3 4 7 12 2 11

7. (2) 더 효율적임

7. (3) 2i항은 i를 증가시킬 때마다 2씩 증가함 🡪 따라서 i를 2씩 증가시키고 i = (i +inc +1)로 해주어도 똑같은 결과를 얻을 수 있음

8. 

9. 

10. 이중조사법이 빠름