대규모의 정보 검색을 위한 효율적인 최소 완전 해시함수의 생성

김 수 희† · 박 세 영††

요 약

대량의 정보를 빠르게 검색하기 위해 성능좋은 인덱스를 개발하는 것은 매우 중요하다. 본 연구에서는 $5 \cdot m$개의 키들을 m개의 버켓에 충돌없게 해시하는 최소 완전 해시함수를 다시 고려하게 되었다. 대량의 정보를 대상으로 최적의 인덱스를 성공적으로 구축하기 위해 Heath가 개발한 MOS 알고리즘을 개선하고, 이를 토대로 최소 완전 해시함수들을 생성하는 시스템을 개발하였다. 이를 실험하기 위해 대량의 데이터들에 적용한 결과 Heath의 알고리즘보다 효율적으로 각각의 최소 완전 해시함수를 계산하였다. 본 연구에서 개발한 시스템은 자주 변하지 않는 대량의 정보나 탐색 속도가 매우 느린 저장 매체에 저장할 데이타를 대상으로 인덱스를 구축하는 데 이용할 수 있다.

Effective Generation of Minimal Perfect Hash Functions for Information Retrieval from Large Sets of Data

Su-Hee Kim† · Se-Young Park††

ABSTRACT

The development of a high performance index system is crucial for the retrieval of information from large sets of data. In this study, a minimal perfect hash function (MPHF), which hashes m keys to m buckets with no collisions, is revisited. The MOS algorithm developed by Heath is modified to be successful for computing MPHFs of large sets of keys. Also, a system for generating MPHFs for large sets of keys is developed. This system computed MPHFs for several large sets of data more efficiently than Heath's. The application areas for this system include those for generating MPHFs for the indexing of large and infrequently changing sets of data as well as information stored in a medium whose seek time is very slow.

1. 서 론

대용량의 정보를 저장할 수 있는 매체들이 비교적 저렴하게 생산됨에 따라, 이러한 매체들을 이용하여 방대한 양의 데이타를 어렵지 않게 저장할 수 있게 되었다. 검색 성능을 보다 향상하기 위한 노력은 이러한 대량의 정보를 효율적으로 검색하기 위해서 반드시 필요하다.

검색대상이 되는 문헌들이 많지 않을 때에는 비트맵 기법을 응용한 정보 검색이 비교적 효율적이다. 그러나 방대한 양의 문헌들을 저장하는 경우, 문서들을 대표하는 키워드들의 수가 이에 비례하여 많아지므로, 비트맵 기법이나 이를 응용한 기법을 정보 검색 시스

† 정 회 원 : 호서대학교 컴퓨터학부 교수
†† 정 회 원 : 한국전자통신연구원 자연어처리연구부 부장
논문접수 : 1998년 2월 11일, 심사완료 : 1998년 7월 7일

템에 활용하기에는 실용적이지 못하다[1].

최근 CD_ROM (Compact Disc Read-Only Memory)과 DVD_ROM (Digital Video Disk Read-Only Memory)은 많은 양의 데이타를 값싸게 저장할 수 있고 영구적이기 때문에 상업적으로 많은 각광을 받고 있다. CD_ROM과 DVD_ROM은 많은 사람들에게 정보를 배포하는 좋은 출판 수단이다. 즉 이러한 저장 매체에 저장할 파일들을 일단 한번 생성하고 나면, 우리는 더 이상 정보를 갱신하거나 추가할 수 없고 수 천번 혹은 수 백만번 접근하여 읽기만 할 수 있는 것이다. 이 매체들의 가장 큰 약점은 데이타를 접근하기 위한 디스크 탐색 시간(seek time)이 매우 오래 걸린다는 것이다. CD_ROM과 DVD_ROM의 평균 접근 시간 (seek time + rotational delay)은 각각 약 270 msec와 약 350 msec 이다. CD_ROM의 평균 접근 시간은 자기 디스크의 평균 접근 시간의 약 20배 정도이다[2]. 어떤 정보를 CD_ROM이나 DVD_ROM에 저장할 때는 검색에 사용될 모든 키들을 이미 가지고 있으며 데이타의 추가나 갱신을 고려할 필요가 없으므로, 탐색 속도가 매우 느린 단점을 보완하기 위하여 인덱스의 구조를 최적화할 가치가 있다. 일단 한번 파일들을 생성하면 그 다음은 계속 읽기만 하므로, 최적의 파일 구조를 구축하기 위해서 비교적 값비싸고 시간이 걸리는 작업일지라도 해 볼 가치가 있다.

본 연구에서는 정보의 양은 방대하지만 자주 변하지 않는 정보나 CD_ROM이나 DVD_ROM과 같은 탐색 속도가 매우 느린 저장 매체에 저장할 정보를 대상으로 최적의 인덱스를 구축하기 위해 최소 완전 해시함수를 다시 고려하게 되었다. 최소 완전 해시함수 알고리즘은 m개의 키들을 m개의 버켓에 충돌이 일어나지 않게 해시하는 함수를 계산하는 알고리즘이다[3]. 대용량의 정보를 대상으로 효율적으로 최소 완전 해시함수를 계산하기 위해 Heath가 개발한 MOS(Mapping, Ordering, Searching) 알고리즘을 수정하여 좀 더 강력하게 개선하였으며, 이를 PC상에서 C 언어로 구현하고 성능 평가를 수행하였다.

본 논문의 구성은 다음과 같다. 제 2장은 최소 완전 해시함수의 정의와 이를 생성하기 위해 개발된 기존의 MOS 알고리즘을 소개한다. 제 3장에서는 Heath의 MOS 알고리즘을 해시할 키들의 수가 매우 많을 경우에도 효율적이고, 실용적으로 사용할 수 있도록 개선할 점들을 제안한다. 제 4장에서는 개선한 MOS 알고리즘의 구현, 실험, 평가, 검색방법에 대하여 논의하고 제 5장에서 결론을 맺는다.

2. 관련 연구

2.1 해싱

해싱은 어떤 해싱 함수에 데이타의 키값을 입력하여, 그와 관련된 버켓 주소를 산출하는 과정이다. 해시를 이용한 정보 검색 시스템에서는 이 주소를 바탕으로 데이타를 저장하며, 이미 저장된 데이타를 검색, 삭제, 수정 등을 할 수 있다. 저장할 데이타에서 두 개 이상의 키들이 같은 버켓으로 사상될 때 충돌이 일어난다. 충돌이 일어나면 주어진 키의 데이타가 저장될 유일한 버켓을 결정하기 위해 더 많은 계산이 필요하게 되며, 결과적으로 시스템의 성능이 저하된다. 해시할 키들의 수가 n이고 저장할 수 있는 버켓의 수를 m이라고 가정하자. 만약 $n > m$이면 항상 충돌이 일어나며, $n <= m$인 경우에도 일반적으로 충돌이 일어난다. 그러므로 충돌의 횟수를 줄임으로써 시스템의 성능이 향상될 수 있다.

2.2 해시함수

해시함수는 입력되는 키를 하나의 버켓으로 사상한다. 해시에 이용하는 버켓의 수가 m일 때, 해시함수의 범위는 0과 m-1내의 정수이어야 한다. 그러므로 해시함수는 다음과 같은 형태를 취한다.

$$h(k) = f(k) \bmod m$$

해시함수의 선택은 시스템의 성능면에서 매우 중요하다. 완전 해시함수는 충돌이 전혀 일어나지 않도록, 모든 키 값들을 서로 다른 유일한 버켓으로 사상하는 함수이다. 완전(perfect) 해시함수를 발견하는 것은 매우 힘들며, 일반적으로 해시할 키들을 미리 알고 있을 때에만 가능하다. 완전 해시함수는 특정 키에 대응하는 버켓을 찾는 데 걸리는 시간이 해시함수를 계산하는 데 필요한 시간과 같다는 점에서 이상적인 함수이다.

2.3 최소 완전 해시함수 (Minimal Perfect Hash Function: MPHF)

최소 완전 해시함수는 m개의 키들을 m개의 버켓을

이용하여 해시하는 완전 해시함수이다. 즉 주어진 키에 대응하는 버켓을 찾는 데 최소의 시간이 소요되며, 해시 테이블에 사용된 메모리의 소비가 전혀 없다는 점에서 매우 이상적이다. 완전 해시함수는 시간적인 성능면에서 최적인 반면, 최소 완전 해시함수는 시간적, 기억 공간적으로 모두 최적인 함수라 할 수 있다.

2.3.1 최소 완전 해시함수는 존재하는가?

Jaeschke는 주어진 키들의 최소 완전 해시함수는 항상 존재한다는 것을 증명했다[4]. N을 초과하지 않는 m개의 양수들로 이루어진 키들의 집합을 m개의 버켓을 가진 해시 테이블로 충돌없게 사상하는 문제를 생각해보자. 다음과 같은 알고리즘이 하나의 최소 완전 해시함수를 정의할 수 있다[4].

Store the keys in an array k of length m
Allocate an array A of length N and initialize all values to ERROR
for (j=0; j < m; j++)
$A[k[j]] = j$;

위의 배열 A가 하나의 최소 완전 해시함수를 정의한다. 키들의 집합에 소속된 키들은 버켓 주소 {0, ..., m-1} 범위의 값으로 사상되며, 그렇지 않은 키들은 ERROR로 사상된다. 대부분의 경우, $m \ll N$ 이므로 해시함수 A가 너무 많은 기억 공간을 필요로하므로 실용적으로 이용되기는 힘들다. 하나의 최소 완전 해시함수가 실용적으로 사용되려면, 그 최소 완전 해시함수를 정의하기 위해 소요되는 공간도 작아야 한다.

2.3.2 최소 완전 해시함수를 계산하는 MOS 알고리즘

최소 완전 해시함수를 계산하는 여러 가지 알고리즘이 개발되었다. 대부분의 알고리즘들이 키들의 수가 작을 때 적용할 수 있다[5,6,7]. Cichelli를 비롯한 연구자들이 최소 완전 해시함수를 계산하기 위해 일련의 파라메타값들을 저장하는 테이블과 mapping, ordering, searching (사상, 순서화, 탐색: 이하 MOS) 방법을 이용하였다[8,9,11]. 다음은 이들의 알고리즘을 간략하게 소개한다.

1) Cichelli의 MOS 알고리즘

Cichelli는 해싱하고자하는 키들을 각 키들의 문자열의 길이, 첫 문자와 마지막 문자의 세 요소로 사상한다. 순서화 단계에서는 먼저 키들의 리스트에서 각 키의 첫 문자와 마지막 문자들의 빈도수를 계산한다. 이 두 빈도수의 합의 내림차순으로 키들을 정렬한다. 이렇게 정렬된 리스트는 어떤 키가 그 이전의 키들에 의해 첫 문자와 마지막 문자들의 해시값이 결정될 수 있으면, 그러한 키가 그들의 다음 순서로 재조정된다. 탐색 단계는 순서화 단계에서 결정된 키 리스트의 순서대로 모든 키들을 유일하게 해시하는 과정이다. Cichelli는 다음과 같이 각 키에 대하여 해시값을 탐색하였다[8].

Hash value ← key length + associated value of the key's first character + associated value of the key's last character

그러므로 해시할 모든 키의 첫 문자와 마지막 문자에 대해, 0부터 시작하여 적당한 값을 탐색하여 충돌이 없는 해시가 되도록 하는 것이다[8].

Cichelli의 해시함수는 매우 간단하며 계산하기 쉽다는 점이 있지만, 만약 어떤 두 키의 길이가 같고 첫 문자와 마지막 문자들이 같다면 충돌을 피할 수 없다. 예를 들어 "ODD"와 "ORD"는 해시 값이 같게 된다. 즉 이 경우에는 최소 완전 해시함수를 생성하지 못하게 되는 것이다. Cichelli의 방법은 30~40개 정도의 키들에 대해 매우 제한된 범위에서 최소 완전 해시함수를 생성하지만, 그의 MOS 연구법은 이 분야의 연구에 지대한 공헌을 하였다.

2) Sager의 MOS 알고리즘

Sager[9]는 Cichelli의 방법을 확장하여 더 많은 키들을 대상으로 최소 완전 해시함수를 생성하는 알고리즘을 개발하였다.

- 사상(Mapping) 단계

다음과 같은 세 보조함수를 이용하여 각각의 키 k를 세 정수 ($h_0(k)$, $h_1(k)$, $h_2(k)$)로 이루어지는 트리플로 변환한다.

$h_0 : W \rightarrow I$, $h_0(k)$ = (length(k) + (ord($k[i]$), i:=1 to length(k) by 3)) mod m

$h_1 : W \rightarrow R_1$, $h_1(k)$ = (ord($k[i]$),

i := 1 to length(k) by 2) mod r

h_2 : $W \rightarrow R_2$, $h_2(k)$ = (ord(k[i]),

i := 2 to length(k) by 2) mod r+r

여기서 W, I, R_1, R_2, m, r 은 각각

W : 모든 키들의 집합,

I : 모든 정수들의 집합,

m : 집합 W 에 있는 키들의 수, $|W|$

r : m에 비례하는 값으로 $m/2$를 상한으로 하는 정수

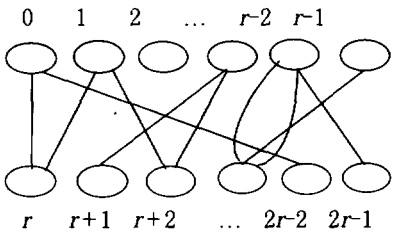
R_1 = {0, ... , r - 1}, R_2 = {r, ... , 2r - 1},

ord 함수 ord(k[i]) : 키의 i 번째 문자의 EBCDIC 코드를 나타낸다.

매개변수 r은 완전 해시함수를 저장하는 기억 공간($|h|$ = 2r)을 결정한다. 사용자는 이 공간을 적절히 선택할 수 있다. Sager는 키들의 관계를 그래프를 이용하여 제한하기 위해 $h_1(k)$와 $h_2(k)$값들을 이용하여 그림 1과 같이 양분 그래프(bipartite graph)인 종속 그래프로 변환한다. 양쪽 꼭지점들의 집합은 R_1과 R_2이다.

그림 1에서 보는 바와 같이 각각의 키 k에 대하여 k로 레이블된 $h_1(k)$와 $h_2(k)$를 연결하는 간선이 있다. 어떤 꼭지점들은 부속되는 키들이 전혀 없을 수 있으며, 반면에 여러 개의 키들이 같은 꼭지점 쌍들을 가질 수도 있다. 성공적으로 최소 완전 해시함수를 생성하려면 사상 단계에서 서로 다른 어떠한 두개의 키도 같은 트리플로 사상되어서는 안 된다.

Keys → Distinct Triples → a Bipartite Graph, the Dependency Graph



(그림 1) 종속 그래프
(Fig. 1) Dependency Graph

● 순서화(Ordering) 단계

순서화 단계에서 Sager는 그래프 내에서 최소 사이클을 발견하는 휴리스틱을 도입하여, 해시 값들을 할당하기 위한 키들의 일련의 순서를 정한다. 즉 키들은 사상 단계에서 양분 그래프의 간선들로 변환되는데, 이 간선들 중에서 최소 길이이며 최대 개수의 사이클에 포함되는 간선을 먼저 선택한다. 선택된 간선은 그래프에서 제외되며, 남은 간선들 중 최소 길이이며 최대 개수의 사이클에 포함되는 간선을 다음으로 선택한다. 최소 사이클을 이용한 방법은 $O(m^4)$의 시간이 걸리며 $O(m^3)$의 기억 공간이 필요하다[9].

● 탐색(Searching) 단계

순서화 단계에서 해시값들을 할당하기 위한 키들의 일련의 순서가 정해진다. 이 순서는 몇 개의 부순서들로 이루어지며, 이 부순서들의 레벨에 따라 해시값을 할당한다. 최소 완전 해시함수를 찾기 위한 함수들의 클래스는

$h(k)$ = ($h_0(k)$+go $h_1(k)$)+go $h_2(k)$) (mod m)

이며, g는

g : R {0, ..., m-1}, R = R_1 R_2

로 위의 클래스에서 완전 해시함수를 탐색하는 함수이다. 각 레벨에서 g값을 이미 해시를 수행한 이전 레벨에 있는 키들의 해시값과 충돌이 일어나지 않도록 가능한 범위에 있는 모든 값들을 탐색한다.

이 방법은 키들의 수가 수백 개일 때는 성공적으로 하나의 최소 완전 해시함수를 계산한다. 이 알고리즘의 시간 복잡도는 $O(m^4)$인데 Macintosh II를 이용하여 120개의 키들의 최소 완전 해시함수를 생성하는데 약 0.29시간이 걸렸다고 보고하고 있으며, 데이터의 크기가 240인 경우에는 4시간 이상 소요된다[10].

3) Heath의 MOS 알고리즘

이러한 연구에 이어서 Heath는 다음과 같은 현상들을 고려하여 $O(m \log m)$의 시간이 걸리는 알고리즘으로 발전시켰다[11]:

1) 사상 단계에서 서로 다른 키들이 같은 트리플로 변환되는 가능성을 줄일 수 있도록 난수를 생성하여 이용한다.
2) 양분 그래프에서 꼭지점들의 차수가 대부분 작다는 사실을 이용하여, 순서화 단계를 $O(m \log m)$의 시간이 걸리는 알고리즘으로 개선하였다.

3) 탐색 단계에서 난수를 생성하여 각 꼭지점의 g값을 정한다. 상대적으로 큰 연결 요소(component)에 속해있는 키들에 대해 우선적으로 해시하여, 충돌이 있을 때의 어려움을 가능하면 줄인다.

● 사상 단계

보조함수 h_0, h_1, h_2 를 계산하기 위하여 난수들을 생성하여 세 테이블에 저장한다. 각각의 키 k는 다음과 같은 식을 이용하여 트리플 $(h_0(k), h_1(k), h_2(k))$로 사상된다.

$$h_0(k) = (\Sigma\ \text{table}[0][\text{ord}(k[i])][i],\ i := 1\ \text{to}\ \text{length}(k))\ \text{mod}\ m$$

$$h_1(k) = (\Sigma\ \text{table}[1][\text{ord}(k[i])][i],\ i := 1\ \text{to}\ \text{length}(k))\ \text{mod}\ r$$

$$h_2(k) = (\Sigma\ \text{table}[2][\text{ord}(k[i])][i],\ i := 1\ \text{to}\ \text{length}(k))\ \text{mod}\ r+r$$

● 순서화 단계

대부분의 꼭지점들의 차수가 작다는 사실을 이용하여 Heath는 순서화 단계를 실제 기대되는 시간은 $O(m)$이고, 최악의 경우 $O(m \log m)$인 알고리즘으로 개선하였다[11]. 다음의 과정을 거쳐 각 꼭지점이 순서화된다.

Step 1) 모든 꼭지점들을 차수에 따라 내림차순으로 정렬한다.

Step 2) 가장 높은 차수를 가진 꼭지점(여러 개의 경우 임의의 하나를 선택)을 선택하여 해시할 순서 리스트에 저장하고, 이 꼭지점과 인접하면서 아직 순서 리스트 없는 꼭지점들을 선택하여 힙(heap)과 스택(stack)들을 이용하여 임시로 저장한다. 이때 차수가 낮은 꼭지점들은 각 차수에 따른 스택에 저장하고, 차수가 높은 꼭지점들은 힙에 저장한다.

Step 3) 나머지 꼭지점들 중 가장 차수가 높은 꼭지점을 제거하여 순서 리스트에 추가하고, 이와 인접하면서 순서 리스트에 없으며, 힙이나 스택에 없는 꼭지점들을 그들의 차수에 따라 힙이나 스택에 추가한다.

Step 4) 3)의 과정을 반복하여 하나의 연결 요소(component)에 속하는 모든 꼭지점들의 순서 리스트를 작성한다.

Step 5) 남은 연결 요소들 중에서 차수가 높은 꼭지점을 선택하여 3)의 과정을 반복하여 순서 리스트에 추가한다.

종속 그래프는 여러 개의 연결 요소들로 구성될 수 있기 때문에 각 연결 요소에 속한 모든 꼭지점들을 해시할 순서 리스트에 올릴 때에 순서화단계가 완성된다. 힙에서 가장 차수가 높은 꼭지점을 선택하는 데는 $O(\log m)$의 시간이 걸리지만, 스택에서는 어떤 상수 시간이 소요된다. 실험적으로 대부분의 꼭지점들의 차수가 작기 때문에 해시하는 순서를 정하기 위해 대기하는 꼭지점들이 거의 해당하는 스택에 저장이 된다. 그러므로 해시할 순서 리스트를 작성하는데 걸리는 기대 시간은 $O(m)$이며 최악의 경우 $O(m \log m)$의 시간이 소요된다.

● 탐색 단계

순서화 단계에서 구축된 꼭지점들의 순서가 v_1, v_2, ..., v_t 일때, 키들의 레벨 $K(v_i)$, $1 \langle = i \langle = t$, 는 v_i와 순서 리스트에서 v_i 보다 먼저 나열된 꼭지점에 부속된 키들의 집합이다.

만약 $0 \langle = v_i \langle = r-1$ 이면

$K(v_i) = \{ k_j \mid h_1(k_j) = v_i,\ h_2(k_j) = v_s,\ s \langle i \}$ 이고

만약 $r \langle = v_i \langle = 2r-1$ 이면

$K(v_i) = \{ k_j \mid h_2(k_j) = v_i,\ h_1(k_j) = v_s,\ s \langle i \}$ 이다.

이와 같이 각 꼭지점에 대한 레벨이 정해지면, 레벨별로 탐색 단계에서 해시를 시작한다. 탐색 단계에서 난수를 생성하여 각 꼭지점의 g값을 정한다. 어떤 키 k 에 대한 해시값은

$$h(k) = (h_0(k) + g_0 h_1(k)) + g_0 h_2(k)) \pmod m$$

이므로 어떤 꼭지점 v_i의 레벨에 속하는 키들의 해시 함수값들은 v_i의 g값에 의해 일부 결정된다.

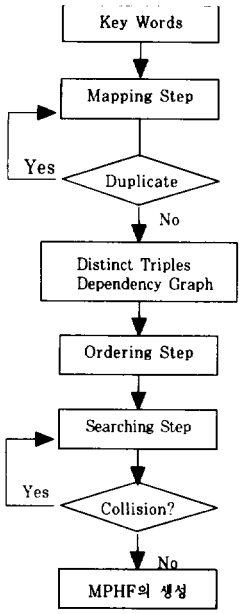
충돌시 새로운 g 값 선정: $K(v_i)$에 속하는 어떤 키의 해시 값이 충돌하였을 때, 꼭지점 v_i 에 대한 g값을

$$g = g + s \pmod m$$

만큼 이동하여 이 레벨에 속한 모든 키에 대하여 다시 해시를 시도한다. 여기서 s는 임의의 어떤 소수(pri-

me)이다. 만약 계속 충돌이 일어난다면, g 값은 s만큼 계속 이동하게 된다. 즉 g 값은 g, $g + s \pmod m$, $g + 2s \pmod m$ ... 과 같은 패턴으로 새로운 g 값을 선택하여 다시 해시를 시도한다[11].

Heath의 MOS 알고리즘은 사상 단계에서 $O(m)$, 순서화 단계에서 $O(m \log m)$ 그리고 탐색 단계에서 $O(m)$의 계산 시간을 요하므로 전체적으로 $O(m \log m)$ 알고리즘이다. 사상 단계에서 두 개 이상의 키들이 동일한 트리플로 변환되거나, 탐색 단계에서 충돌이 일어나면 최소 완전 해시함수를 계산할 수 없다. 그러므로 사상 단계와 탐색 단계에서 이들을 확인하는 과정이 필요하다. 그러므로 Heath의 MOS 알고리즘을 적용하여 실질적으로 최소 완전 해시함수를 구축하기 위한 전체 과정은 $O(m^2)$의 시간이 소요된다. 그림 2는 이러한 과정을 나타낸다.



(그림 2) MOS 알고리즘의 순서도
(Fig. 2) Flowchart of MOS Algorithm

3. Heath의 MOS 알고리즘의 개선

앞 절에서 소개한 Heath의 MOS 알고리즘이 사용한 매개변수 r은 m 에 비례하는 값으로 최소 완전 해시함수의 값들을 계산하기 위해 필요한 g 값을 저장하는 기억 공간($|h|=2r$)을 결정한다. 즉 양분 그래프에서 각 꼭지점에 대응하는 g값을 저장한다. 이 값들은 검색 과정에서 키들의 해시값을 재생성하기 위해 역시 필요하다. 그러므로 $2r$ 개의 g 값은 검색을 위해서 저장하여야 하므로 r의 값은 적으면 적을수록 바람직하다. 특히 해시할 키들의 수가 매우 많을 때는 r의 값을 최소로 하는 노력이 필요하다.

그리고 탐색 단계에서 충돌이 있을 때에 g 값을 어떤 임의의 소수 값 s 만큼 이동하여 다시 해시를 시도하는 데, 어떤 수가 소수인지 아닌지 결정하는 데 시간이 많이 걸리므로 사용하는 소수들의 수도 가급적 최소로 하는 것이 바람직하다. Heath의 알고리즘에서 r의 값을 최상한($r = m/2$)으로 취하더라도 사용하는 소수들의 수를 20 이하로 제한하면 탐색 과정에서 많은 실패를 경험하게 된다.

본 연구에서는 키들의 어떤 특징이나 분포에 의존하지 않고 각 키들을 가급적이면 균일하게 사상되도록 노력함으로써, 탐색 과정에서 r의 값과 소수들의 수가 적더라도 성공적으로 최소 완전 해시함수를 발견할 수 있도록 사상 단계와 탐색 단계를 부분적으로 개선하였다.

3.1 사상 과정의 개선

사상 단계에서 난수들을 생성하여, 키들과 대응하는 트리플들을 계산한다. 이때 서로 다른 두 개의 키 k_1과 k_2의 트리플 $(h_0(k_1), h_1(k_1), h_2(k_1))$과 $(h_0(k_2), h_1(k_2), h_2(k_2))$가 서로 같게 되면 탐색 단계에서 해시 값이 같게 되므로, 사상 단계에서 새로운 난수들을 생성하여 다시 각각의 트리플을 계산하여야 한다.

어떤 키 k를 $(h_0(k), h_1(k), h_2(k))$의 트리플로 변환할 때, 세 함수 h_0, h_1, h_2 는 일반적으로 다음과 같은 형태이다.

$$h_0(k) = f_0(k) \bmod m \quad (1)$$

$$h_1(k) = f_1(k) \bmod r \quad (2)$$

$$h_2(k) = f_2(k) \bmod r + r \quad (3)$$

Heath가 사용한 $f_0(k)$, $f_1(k)$, $f_2(k)$는 키값 k와 관련된 난수들이다. 난수 $f_0(k)$를 해시할 키들의 수 m으로 나눈 나머지가 $h_0(k)$ 이다. $h_1(k)$, $h_2(k)$의 값들도 이와 유사하게 계산된다.

비록 $f_0(k)$, $f_1(k)$, $f_2(k)$들이 키 k와 관련된 난수라 하더라도, m과 r의 정수들이 갖는 소인수들에 따라 $h_0(k)$, $h_1(k)$, $h_2(k)$값들은 주어진 범위 내에서

균일하게 분포하지 못하고 어떤 편중된 분포를 가질 수 있다. m이 짝수라고 가정하자. 만약 어떤 키 k와 관련된 난수 $f_0(k)$가 짝수라면 $h_0(k)$는 0~m 1 범위의 어떤 짝수가 되며, 반대로 $f_0(k)$가 홀수라면 $h_0(k)$는 0~m - 1 범위의 어떤 홀수가 된다. 예를들어 m = 6이고 난수들이 7, 15, 21, 65이면 이들 각각을 m 으로 나눈 나머지는 1, 3, 5 중의 어떤 값이 되며, 반면에 난수들이 8, 16, 22, 66이면 각각의 나머지는 2, 4, 0 중의 어떤 값이 된다.

임의의 양의 정수 m은 일반적으로 다음과 같이 표현될 수 있다[12].

$$m = p_1^{m_1} p_2^{m_2} \cdots p_s^{m_s}$$

여기서 $p_1, p_2, \ldots, p_s$는 서로 다른 소수이며, $m_1, m_2, \ldots, m_s$는 양의 정수들이다. 어떤 키 k와 관련된 난수 $f_0(k)$는 m의 소인수들을 이용하여 다음과 같이 나타낼 수 있다.

$$f_0(k) = p_1^{m_1'} p_2^{m_2'} \cdots p_s^{m_s'} f$$

각 소수 p_i의 지수 m_i'은 $0 \langle = m_i' \langle = m_i$ 의 정수이며, 이때 i는 $1 \langle = i \langle = s$ 이다. 그리고 정수 f와 m의 최대 공약수는 1이다. 즉 $f_0(k)$와 m의 최대 공약수는

$$p_1^{m_1'} p_2^{m_2'} \cdots p_s^{m_s'}$$

이다. 그러므로 난수 $f_0(k)$가 사상될 수 있는 $h_0(k)$의 값은 {0, ..., m-1} 범위 내에서

$$p_1^{m_1'} p_2^{m_2'} \cdots p_s^{m_s'}$$

의 배수가 되는 어떤 값들 중의 하나로 제한된다. 이 사실을 다음과 증명할 수 있다.

어떤 양의 정수 x와 y의 최대 공약수 GCD(x, y) = c 라고 가정하자.

그러면 $x = c * x_1$, $y = c * y_1$으로 나타낼 수 있고, 이때 x_1과 y_1은 양의 정수이며 GCD(x_1, y_1) = 1이다. x를 y로 나눈 몫과 나머지를 q와 r이라고 할 때, 우리는 x와 y를 다음과 같이 표현할 수 있다.

$x = c * x_1 = q * y + r = q * c * y_1 + r$

위의 식에서 $c * x_1 = q * c * y_1 + r$ 이며 $r = c * x_1 - q * c * y_1 = c(x_1 - q * y_1)$ 이므로 r은 항상 c의 배수가 된다.

그러므로 m이 많은 소인수들을 가지면 임의의 어떤 난수와 공통되는 소인수들을 가질 가능성이 높아지며, 결과적으로 h_0값들이 편중되어 분포할 가능성이 높아진다. 매개변수 r이 많은 소인수들을 가지면, 역시 h_1과 h_2의 값들도 편중되어 분포할 가능성이 높아진다. 사상 단계는 키들과 대응하는 트리플들이 모두 다를 때에 성공적으로 수행되는 것이다. 본 연구에서는 h_0, h_1, h_2의 값들을 고르게 분포하게 하여 트리플들이 중복될 가능성을 배제하며 더 나아가서 최소 완전 해시 함수를 성공적으로 생성할 수 있도록, m보다 크면서 m에 가장 가까운 소수 p를 계산하고 r보다 크면서 r에 가장 가까운 소수 q를 계산하여 이용한다. W 에 있는 키들이 사상되는 과정을 p와 q를 적용하여 계산하기 위하여, 다음과 같이 변형한다.

h_0: W {0, ..., p-1} {0, ..., m-1}
h_1: W {0, ..., q-1} {0, ..., r-1}
h_2: W {0, ..., q-1} {0, ..., r-1}

위와 같이 W 에 있는 키들을 사상하기 위해 (1), (2), (3) 식을

$h_0(k)=(f_0(k) \bmod p) * \text{int}((m-1)/(p-1)+0.5)$ (1)'
$h_1(k)=(f_1(k) \bmod q) * \text{int}((r-1)/(q-1)+0.5)$ (2)'
$h_2(k)=(f_2(k) \bmod q) * \text{int}((r-1)/(q-1)+0.5)+r$ (3)'

으로 계산한다.

3.2 탐색 과정의 개선

사상 단계에서 각각의 키 k는 ($h_0(k)$, $h_1(k)$, $h_2(k)$)의 트리플로 사상되며, h_1과 h_2의 함수의 값들로 양분 그래프를 형성한다. 모든 키들의 트리플들이 중복되지 않으면 순서화 단계를 거쳐 각 레벨에 속하는 키들의 순서대로 해시를 시도한다. 사용하는 해시함수식은

$h(k)=(h_0(k)+go\ h_1(k))+go\ h_2(k))$ (mod m) 이다.

여기서 g는 $g : R \to \{0, \ldots, m-1\}$이며, R은 $\{0, \ldots, 2r-1\}$로서 양분 그래프의 꼭지점들의 집합이다. 우리의 목표는 m개의 키들에 대한 최소 완전 해시함수를 생성하기 위해 R에 속하는 양분 그래프의 각 꼭지점 v에 대한 g값을 탐색하는 것이다. Heath는 양분 그래프의 각 꼭지점에 대한 g의 초기값으로 난수를 이용한다.

Heath는 탐색 단계에서 어떤 레벨 $K(v_i)$에 속하는 어떤 키의 해시 값이 충돌하였을 때, 꼭지점 v_i 에 대한 g값을 $g + s$ (mod m) 만큼 이동하여 이 레벨에 속한 모든 키에 대하여 다시 해시를 시도한다. 여기서 s는 임의의 어떤 소수이다. 만약 계속 충돌이 일어난다면, g값은 s만큼 계속 이동하게 된다. 즉 g값은 g, $g + s$ (mod m), $g + 2s$ (mod m) 과 같은 패턴으로 변한다.

만약 k가 $1 <= k <= m-1$인 어떤 정수이고 $g \equiv g + k * s$ (mod m)이면,

즉 g (mod m) $= g + k * s$ (mod m)이면,

$g + i * s \equiv g + (k + i) * s$ (mod m)이다.

그러므로 어떤 v_i 의 레벨에 속한 키들의 해시값들이 연속하여 k번 충돌이 일어나고 $g \equiv g + k * s$ (mod m)이면, 그 이후로 g를 s만큼 더 이동하여 다시 해시를 시도하여도 계속 충돌이 일어난다. 만약 s가 m의 어떤 소인수라면 k 는 m/s의 어떤 배수이다. 최소의 k 는 m/s 이다. 충돌 시 선택된 임의의 소수 s가 m의 소인수가 아니면 g $g + k * s$ (mod m)을 만족하는 k, $1 <= k <= m-1$, 는 존재하지 않지만 s가 m의 어떤 소인수라면 $k = m/s$이다. 예를 들어 $m = 10$, $s = 5$이면 최소의 k 는 2 이다. 즉 어떤 g, $g + s$에서 충돌이 일어나면, 그 이후로 g 값을 s만큼 이동하여도 계속 충돌이 일어나 결국 최소 완전 해시함수를 계산하는 데 실패한다.

이러한 가능성을 배제하기 위하여 매번 충돌이 일어날 때 g의 이동하는 값을 임의의 어떤 소수로 선택하는 대신에, m의 소인수가 아닌 어떤 소수 s를 선택한다. 즉 s와 m의 최대 공약수 GCD(s, m) = 1인 s를 선택한다. 매번 충돌이 일어날 때마다 g 값은 $g + s$ (mod m) 만큼 이동하여 다시 해시를 시도한다. 제안하는 g 의 패턴은 g, $g + s$ (mod m), $g + 2s$ (mod m), ·, $g + (m - 1)s$ (mod m)인데, 이들의 값은 모두 다르다[12]. 그러면 왜 이들의 값이 모두 다른가를 살펴보자.

이 사실을 부정하여 $g + i * s$ $g + j * s$ (mod m) 라고 가정하자. 여기서 GCD(s, m) = 1이며, i와 j는 $0 \le i, j \le m-1$, $i < j$인 정수이다.

그러면 $g + i * s$ (mod m) $= g + j * s$ (mod m) 이다.

그러므로 만약 $g + i * s = m * q_1 + r$이면 $g + j * s = m * q_2 + r$이다.

즉 $g + j * s - (g + i * s) = (j - i) * s = m * (q_2 - q_1)$ 이다.

그러므로 $j - i = m * (q_2 - q_1) / s$ 이다.

그런데 GCD(s, m) = 1이므로 $(q_2 - q_1)$은 s의 배수가 되며, 따라서 $j-i$ 는 m의 배수가 된다. 즉 $j-i$의 값은 0, m, $2m$, $3m$, … 형태이다. 이를 만족하는 j 의 값은 i, $i + m$, $i + 2m$, … 형태이다. 그런데 j 는 $0 \le j \le m-1$, $i < j$ 인 정수이므로, 이러한 조건을 만족하는 j 값은 존재하지 않는다. 그러므로 g, $g + s$ (mod m), $g + 2s$ (mod m), … , $g + (m - 1)s$ (mod m) 들의 값은 모두 다르다.

충돌이 일어날 때 임의의 어떤 소수 s를 선택하는 것이 아니라 m의 소인수가 아닌 어떤 소수 s를 선택함으로써, $0 \sim m - 1$ 의 범위에 있는 새로운 g값을 선택하여 가능한 해시 값을 테스트하는 것을 제안한다.

4. 개선된 MOS 알고리즘의 구현 및 성능 평가

본 연구에서 제안하는 MOS 알고리즘은 Heath의 MOS 알고리즘과 전체적인 골격은 같고 다만 사상 단계와 탐색 단계에서 좀 더 효율적으로 최소 완전 해시함수를 발견할 수 있도록 개선한 것이다. 이를 바탕으로 시스템을 구축하였고, 몇 개의 샘플 데이터를 이용하여 기존의 Heath 알고리즘과 비교 평가하여 보았다.

4.1 시스템의 구축

Heath의 MOS 알고리즘을 방대한 정보의 검색에 실용적으로 이용하기 위하여, [3]에 있는 샘플 프로그램을 기초로 하여 사상 단계와 탐색 단계를 3장에서 제안한 방법으로 개선하여 C 언어로 PC상에서 구현

하였다.

[3]에 있는 샘플 프로그램에서 g값을 탐색하기 위한 소수들을 첫 20개의 소수들로 제한한 것을 사용자가 원하는 대로 소수들의 수를 지정할 수있고, 모든 도메인의 키들을 처리할 수 있도록 프로그램의 일부를 변경하였다. 이것은 Heath의 알고리즘을 그대로 구현한 시스템이다.

4.2 실험 데이터

두 시스템을 실험하고 비교 평가하기 위하여 다음의 데이타를 이용하였다[13]. 1) 1996년 CD로 출판된 계몽사 학생 백과 사전의 역 파일의 약 18만개의 색인어 2) 전자통신 연구원의 자연어 처리 연구실에서 만든 약 30만개의 키워드, 그리고 3) 1)의 데이타를 구성하는 단어들 마지막에 어떤 문자들을 추가하여 인공적으로 만든 약 54만 단어들이다. 이 데이타들은 한국 문자뿐만 아니라 영어 알파벳, 한문자, 다른 기호들로 이루어져 있다.

4.3 실험 및 성능 평가

4.3.1 실험

표 2와 표 3은 크기가 10인 샘플 데이터를 이용하여 본 연구에서 구현한 시스템과 Heath의 시스템을 이용하여 최소 완전 해시함수를 생성할 때의 사상, 순서화, 탐색 과정의 결과를 나타낸다. 사용한 샘플 데이터 W는 W = {복숭아, 사과, 딸기, 포도, 수박, 참외, 배, 토마토, 키위, 멜론} 이며, 그 크기 m은 10이다. 선택한 비율 r / m은 0.4 로 $r = 4$, $2r = 8$이다. 난수를 생성하는 초기치는 101이다. 키들이 먼저 트리플로 사상되고, 이를 기반으로 형성된 양분 그래프에서 꼭지점들의 차수대로 모든 꼭지점들을 순서화하며, 이들을 연결 요소와 차수에 입각하여 재 조정하여 각 키에 대하여 해시할 순서를 정한다. 최종적으로 최소 완전 해시함수를 생성하기 위하여 각 꼭지점에 해당하는 g값을 탐색한다.

표 2와 표 3에서 보는 바와 같이, 사상 과정에서 꼭지점들에 대한 차수의 분포가 본 연구에서 개선한 알고리즘에서는 1 ~ 4의 범위에 걸쳐 있고 Heath의 알고리즘에서 0 ~ 6의 범위에 걸쳐있다. 이 하나의 예로 단정적으로 말할 수 없지만 개선한 알고리즘에서는 사상 단계에서 키들이 편중되게 사상되지 않도록 함으로써 트리플들이 중복되지 않게 하고, 탐색 단계에서 좀 더 수월하게 g값을 결정할 수 있도록 한다는 것을 알 수 있다.

두 시스템을 비교하고 평가하기 위하여 앞 절에서 소개한 데이터들을 이용하여 팬티엄 32.0 MB RAM의 노트북 PC를 이용하여 실험하였다. r과 m의 비율과 소수들의 수를 다양하게 선택하여 그 결과를 비교해 보았다.

앞서 언급한 대로 $2r$개의 꼭지점에 할당한 g값들은 검색 과정에서 키의 해시값을 계산하기 위해 필요하므로, 이러한 기억 공간은 적으면 적을수록 실용하기가 용이하다. 이 값을 최소에 가깝도록 하기위해 r과 m의 비율을 소수점 이하 두 자리에서 단계적으로 낮추어 가며 두 시스템을 실험하였다. 사실 얼마나 열심히 최적의 값을 찾고자 하는 열의에 따라, 이 값은 더 세분화될 수 있다. 표 4는 최소 완전 해시함수를 성공적으로 생성하는 $2r/m$의 최소의 값을 선택하여 두 시스템을 실행하였을 때의 각종 수치를 나타낸다. 물론 전체 알고리즘이 난수를 이용하므로 난수 생성을 위한 초기값을 선택하는 데 따라 이 비율이 다소 차이가 날 가능성이 있다. 그리고 두 시스템에서 사상 과정과 탐

〈표 1〉 실험 데이터
〈Table 1〉 Experimental Data

데이터	파일 이름	크기
계몽사 학생 백과 사전 역화일의 색인어들	KeMomg	약 18만 단어들
전자통신연구원 자연어처리 연구실에서 만든 키워드들	OkSeo	약 30만 단어들
인위적으로 만든 키워드들	Art	약 54만 단어들

〈표 2〉 개선한 알고리즘을 이용한 예
〈Table 2〉 Example of Application of the Improved Algorithm

키	트리플			h	꼭지점	차수	연결요소에 의한 꼭지점들의 재정렬	$K(v)$	g
	h_0	h_1	h_2						
복숭아	4	2	6	8					
사과	4	1	7	1					
딸기	0	0	4	3	4	4	4		8
포도	1	0	4	4	1	4	1	{배},	2
수박	7	1	6	5	0	4	0	{딸기, 포도, 키위}	5
참외	4	1	6	2	6	3	6	{참외,수박}	6
배	7	1	4	7	5	2	5	{토마토}	6
토마토	5	0	5	6	7	1	3	{멜론}	2
키위	6	0	4	9	3	1	2	{복숭아}	8
멜론	2	3	5	0	2	1	7	{사과}	5

〈표 3〉 Heath의 알고리즘을 이용한 예〉
〈Table 3〉 Example of Application of Heaths Algorithm

키	트리플			h	꼭지점	차수	연결요소에 의한 꼭지점들의 재정렬	$K(v)$	g
	h_0	h_1	h_2						
복숭아	6		7	4					
사과	2	2	6	0					
딸기	7	2	5	7	5	6	5		6
포도	5	2	5	5	2	4	2	{딸기, 포도,토마토}	4
수박	2	0	5	3	7	3	0	{수박}	5
참외	2	3	5	6	0	3	7	{멜론,복숭아}	3
배	6	1	5	9	1	2	1	{배,키위}	7
토마토	1	2	5	1	6	1	6	{사과}	4
키위	2	1	7	2	3	1	3	{참외}	8
멜론	0	0	7	8	4	0	4		8

색 과정에서 실패하였을 때 각각의 과정에서 다시 시도할 수 있는 횟수의 한계는 10으로 선택하였다. 사용자는 이 값을 임의로 선택할 수 있다. 한 행내의 첫 행은 Heath의 알고리즘을 적용하였고 두번째 행은 본 연구에서 개선한 알고리즘을 적용한 것이다. 표 4에서 사용한 기호들의 의미는 다음과 같다.

File: 사용한 데이터 파일

m : 파일 내에서 파일의 처음부터 해시에 사용한 키들의 수

R : 키들의 수 m 과 양분 그래프에 사용한 꼭지점들의 수 $2r$의 비율, $2r/m$

NP : 가장 작은 소수 2부터 시작하여 해시에 사용한 소수들의 수

Seed : 난수를 생성하기 위한 초기값

MT : 단위는 sec로 사상 과정에 소요된 시간

OT : 단위는 sec로 순서화 과정에 소요된 시간
ST : 단위는 sec로 탐색 과정에 소요된 시간
TT : 단위는 sec로 전체 해시 과정에 소요된 시간
nMap : 사상 과정을 수행한 횟수. 1의 의미는 중복되는 트리플이 없어 1회에 성공적으로 사상 과정을 완료했다는 뜻이다.
nSear : 탐색 과정을 수행한 횟수. 1의 의미는 충돌 없이 1회에 성공적으로 탐색 과정을 완료했다는 뜻이다.
mDeg : 양분 그래프에서 가장 높은 꼭지점의 차수, 즉 그래프의 차수

4.3.2 성능 평가

● 사상 과정

표 4에 나타나 있지 않지만 사상 과정은 데이터의 크기가 적을 때는 다소 개선되었음을 볼 수 있으나, 데이터의 크기가 클 때는 이 과정 자체로는 크게 향상된 점을 볼 수 없다. 두 알고리즘에서 매우 적은 R의 값을 취하면 사상 과정에서도 충첩이 일어나게 되나, 주어진 R값에서 이 과정보다는 탐색 과정에서 더 많이 실패를 하게 된다.

표 4에 있는 데이터를 이용하여 실험해 본 결과, 개선한 사상 과정과 탐색 과정을 이용하는 것이 Heath의 사상 과정과 개선한 탐색 과정을 이용하는 것보다

〈표 4〉 개선한 알고리즘과 Heath의 알고리즘을 적용한 실험 결과
〈Table 4〉 Results of Application of the Improved Algorithm and Heaths Algorithm

File	*M*	*R*	*NP*	*Seed*	*MT*	*OT*	*ST*	*TT*	*mDeg*	*nMap*	*nSear*
KeMong	8,931	0.40	1000	101	1	0	2	3	16	1	10
		0.40	100	79	1	0	2	3	16	1	3
KeMong	9,089	0.50	500	37	1	0	1	3	13	1	3
		0.42	500	23	1	0	1	2	15	1	1
KeMong	53,157	0.38	500	1159641323	5	0	19	27	17	1	3
		0.36	100	2339	6	1	23	33	18	1	1
KeMong	54,793	0.52	800	101	5	1	8	16	13	1	5
		0.36	20	101	5	1	21	30	20	1	5
KeMong	150,150	0.36	800	101	13	1	143	162	18	1	3
		0.34	20	101	28	2	104	152	18	1	4
KeMong	179,401	0.46	750	101	17	1	48	74	15	1	8
		0.34	20	101	21	1	113	145	18	1	4
OkSeo	235,620	0.38	500	101	67	1	78	165	19	1	2
		0.34	200	79	25	2	170	209	19	1	1
OkSeo	260,043	0.40	1000	101	28	2	192	239	16	1	9
		0.32	50	3911	30	2	340	386	21	1	8
OkSeo	294,527	0.48	3000	101	31	3	113	170	16	1	8
		0.32	50	23	33	2	732	789	21	1	4
Art	397,761	0.40	1000	101	67	3	101	210	19	1	1
		0.32	50	23	87	3	492	622	20	1	1
Art	410,447	0.52	100	101	122	4	74	237	15	1	3
		0.32	50	101	123	2	548	711	21	1	5
Art	538,171	0.54	3000	101	178	4	52	290	15	1	1
		0.32	100	23	149	4	699	904	21	1	1

좀더 성공적이었다. 이점은 개선한 사상 과정은 이 단계에서의 중첩 가능성도 다소 줄이겠지만, 키들의 트리플들이 편중되게 분포하는 가능성을 줄임으로써 다음 단계에 오는 탐색 과정을 좀더 성공적으로 수행하는 데 기여한다는 것을 시사한다.

● 탐색 과정

어떤 수가 소수인지 아닌지 판별하는 작업은 비교적 값 비싼 연산이라 볼 수 있고, 많은 양의 소수들을 저장하여 탐색 단계에 사용하게 되면, 이에 따른 작업 메모리가 많이 필요하다.

표 4에서 보는 바와 같이 개선한 알고리즘에서 사용한 소수들의 수는 20 ~ 100 사이에 주로 분포하며, 데이터의 크기가 증가할수록 증가하는 경향이 있다. Heath의 알고리즘에서는 이들의 수가 500 ~ 3000 사이에 주로 분포하며 데이터의 크기가 증가할수록 역시 증가하는 경향이 있다. 개선한 알고리즘에서는 데이터의 크기가 매우 크더라도 적은 소수들의 수로 성공적으로 최소 완전 해시함수를 생성한다. 그러나 Heath의 알고리즘에서는 R의 값이 크고 사용하는 소수들의 수가 매우 많을 경우에도, 여러 번의 탐색을 시도하게 된다. 더구나 Heath의 알고리즘에서는 데이터의 크기가 크고 사용하는 소수들의 수가 20인 경우에, 선택하는 R의 값에 관계없이 거의 실패를 경험하게 된다.

● 수행 속도

본 연구에서 제안하는 사상 과정과 탐색 과정은 Heath의 두 과정보다 몇 개의 계산 작업들이 더 필요하다. 사상 과정에서 m보다 크면서 m에 가장 가까운 소수 p를 계산하고 r보다 크면서 r에 가장 가까운 소수 q를 계산하여 이용한다. 이 값들을 이용하여 각 키의 트리플을 계산할 때 식 (1), (2), (3)형태의 식으로 계산하므로 (1), (2), (3) 형태의 식 보다는 더 많은 계산이 더 필요하다. 탐색 과정에서는 소수 s가 m의 인수가 아닌 것을 선택하여야 하므로 이에 대한 추가 작업이 필요하다. 그러므로 좀더 적은 r의 값에서 최소 완전 해시함수를 발견하기 위해 전체 수행시간이 좀더 길어진다.

두 알고리즘에서 공통적으로 수행 속도에 영향을 줄 수 있는 점들을 살펴보자.

표 4에서, 두 알고리즘의 사상 과정을 수행하는 시간은 R의 값이 크면 양분 그래프의 꼭지점들이 많게 되므로 키들의 트리플들이 중첩이 있는 가를 조사하는 작업에서 많은 시간이 요구된다. 그리고 트리플들의 중첩이 있는 경우는 사상 과정을 다시 시도해야 하므로 재 시도 횟수가 이 과정의 처리 시간을 결정한다.

순서화 단계를 수행하는 시간은 실제 기대하는 시간은 $O(m)$이지만 최악의 경우는 $O(m \log m)$이라고 앞서 설명하였다. R의 값이 크면 순서화 단계에서도 역시 많은 시간이 요구된다. 이 과정은 표 4에서 보는 바와 같이 세 과정 중 가장 시간이 적게 걸리는 단계이다.

탐색 단계를 수행하는 데 걸리는 시간은 R의 값이 적으면 각 꼭지점에 대한 차수가 평균적으로 높아지므로, 키들의 해시 값이 충돌할 가능성이 높고 몇 번의 실패를 거듭하는 가 하는 그 횟수가 처리 시간을 결정한다.

● 비율 $2r/m$

표 4에서 보는 바와 같이 개선한 알고리즘에서 Heath의 알고리즘에서 보다 $2r/m$ 이 더 적은 값에서 성공적으로 최소 완전 해시함수를 생성한다. 개선한 알고리즘에서는 이 비율의 분포가 사용한 데이터에 민감하지 않으며, 데이터의 크기가 증가할수록 조금씩 감소하는 경향이 있다. 그러나 Heath의 알고리즘에서는 이 비율이 매우 불규칙하고 데이터의 크기가 증가할수록 조금씩 증가하는 경향이 있다.

두 알고리즘에서 성공하는 $2r/m$의 차이로 인한 g 값을 저장하기 위한 메모리의 절감 효과를 계산해보자. 각 꼭지점이 취할 수 있는 g 값은 0 ~ m-1중의 어떤 정수이므로, 이를 표현하기 위해 $\log_2 m$ 비트가 소요된다고 가정하자. 표 5는 표 4에 있는 데이터를 이용하여 계산한 기억 공간의 절감 효과를 나타낸다. 파일 Art에서 538,171개의 키들을 해시하는 경우 Heath의 알고리즘에서 필요한 기억공간의 41%인 약 290K 바이트를 절약하게 된다. 표 5에서 사용한 기호들의 의미는 다음과 같다.

R_h : Heath의 시스템에서 성공적으로 최소 완전 해시함수를 생성한 최소의 R
즉 표 4에서 한 행내의 첫 행에 있는 R의 값

R_k : 개선한 시스템에서 성공적으로 최소 완전 해시함수를 생성한 최소의 R
즉 표 4에서 한 행내의 두 번째 행에 있는 R의 값

〈표 5〉 개선한 알고리즘에서 g값을 위한 기억 공간의 절약 (단위: 비트)
〈Table 5〉 Reduction of Memory for Storing g values in the Improved System

File	m	R_h	R_k	$R_h - R_k$	$(R_h - R_k)/R_h$	NB	$\lceil M^*(R_h - R_k) * NB \rceil$
KeMong	8,931	0.40	0.40	0	0	14	0
KeMong	9,089	0.50	0.42	0.08	0.16	14	10,180
KeMong	53,157	0.38	0.36	0.02	0.05	16	17,011
KeMong	54,793	0.52	0.36	0.16	0.31	16	140,270
KeMong	150,150	0.36	0.34	0.02	0.06	18	54,054
KeMong	179,401	0.46	0.34	0.12	0.26	18	387,506
OkSeo	235,620	0.38	0.34	0.04	0.11	18	169,646
OkSeo	260,043	0.40	0.32	0.08	0.20	18	37,4461
OkSeo	294,527	0.48	0.32	0.16	0.33	19	895,362
Art	397,761	0.40	0.32	0.08	0.20	19	604,597
Art	410,447	0.52	0.32	0.20	0.38	19	1,559,699
Art	538,171	0.54	0.32	0.22	0.41	20	2,367,953

NB : $0 \sim m$ 1을 표현하기 위한 비트 수, $\log_2 m$

$M^*(R_h - R_k)^*NB$: Heath의 알고리즘대신 본 연구에서 개선한 알고리즘을 적용할 때 g값을 위해 절약할 수 있는 기억 공간, 단위는 비트

개선한 알고리즘에서는 검색 과정에서 필요한 해시 함수를 재생성하는 데 필요한 정보를 저장하기 위한 기억 공간이 매우 많이 절약되며, 이 효과는 사상 과정과 탐색 과정의 개선한 점들에서 모두 연유하였겠으나 탐색 과정의 영향이 더 큰 것으로 분석된다.

4.4 검색 방법

사용자들이 어떤 키워드를 이용해 원하는 정보를 검색하기 위해서, 해시에 적용한 MPHF를 다시 재구성하여야 한다. 재구성하는데 필요한 정보는 다음과 같다.

1) 난수들을 다시 생성하기 위한 초기값 : *Seed*
   이 값은 해시에 사용한 난수들을 다시 생성하기 위해 필요하고, 표 6에서 *Seed* 값을 말한다.
2) 해시에 사용된 키들의 수 : m
   이 값은 검색에 사용하는 키 k의 $h_0(k)$와 해시값 $h(k)$를 계산하는 데 필요하다.
   표 4에서 *Keys* 값이다.
3) 양분 그래프를 형성하기 위한 꼭지점들의 수와 키들의 수에 대한 비율: $2r/m$
   이 값은 양분 그래프에 사용된 꼭지점들의 수를 계산하고, 검색에 사용하는 키 k의 양쪽 꼭지점 $h_1(k)$와 $h_2(k)$, 그리고 해시 값 $h(k)$를 계산하는 데 필요하다.
   표 4에서 R값이다.
4) 각 꼭지점에 할당한 g값
   이 값은 해시값 $h(k)$를 계산하는 데 필요하다. 검색에 사용하는 키 k의 양쪽 꼭지점 $h_1(k)$와 $h_2(k)$에 할당된 g 값을 이용하여 해시값 $h(k) = (h_0(k) + g\ h_1(k)) + g\ h_2(k)) \pmod m$을 계산한다.

위의 정보를 이용하여 해시에 사용한 최소 완전 해시함수를 재구성한 후, 입력되는 키워드와 관련한 정보가 저장된 장소를 접근할 수 있다. 참고로 검색을 위한 프로그램도 역시 C 언어를 이용하여 PC에서 구현하였다.

5. 결 론

본 연구는 Heath에 의하여 개발된 MOS 알고리즘의 사상 과정과 탐색 과정의 일부를 개선하여, 대량의 데이터를 대상으로 최소 완전 해시함수를 효율적이고 성공적으로 계산함으로써 실용할 수 있도록 하였다. 이를 C 언어로 PC상에서 구현하였으며 다양한 크기의 데이터를 이용하여 기존의 Heath 알고리즘과 개선한 알고리즘을 비교 분석하였다.

사상 과정은 키들이 트리플로 사상되는 과정을 개선하 것으로, 데이터의 크기가 적을 때는 다소 향상되었다고 할 수 있으나 데이터의 크기가 커짐에 따라 이 과정만으로는 크게 향상된 점을 볼 수 없다. 실험적으로 이 과정에서 키들의 트리플들이 편중되게 분포하는 가능성을 줄임으로써, 탐색 단계에서 효율적으로 최소 완전 해시함수를 생성하는 데 기여한다는 점을 알 수 있다. 탐색 과정은 해시 함수를 탐색하기 위한 방법을 개선한 것으로 기존의 Heath 방법보다 훨씬 더 많이 향상되었음을 볼 수 있다. 개선한 알고리즘에서는 검색 과정에서 필요한 해시함수를 재생성하는 데 필요한 정보를 저장하기 위한 기억 공간이 매우 많이 절약되며, 이 효과는 사상 과정과 탐색 과정의 개선한 점들에서 모두 연유하였겠으나 탐색 과정의 영향이 더 큰 것으로 분석된다.

본 연구에서 개발한 시스템은 어떠한 도메인의 키워드들이라도 모두 처리하며 방대한 양의 키들의 최소 완전 해시함수를 빠른 속도로 계산한다. 이 시스템은 자료는 방대하지만 자주 변하지 않는 정보나 탐색 속도가 매우 느린 저장 매체에 저장할 데이타를 대상으로 인덱스를 구축하는 데 이용할 수 있다.

참 고 문 헌

[1] Salton, G., Automatic Text Processing, Addison Wesley, 1989.

[2] Folk, M. J., Zoellick, Bill, File Structures 2nd., Addison Wesley, 1992.

[3] Wartik, S., E., Fox, L., Heath, Q., Chen, "Hashing Algorithms," in Information Retrieval Data Structures & Algorithms (293 - 362), ed. Frakes, W. B., R., Baeza-Yates, Prentice-Hall, 1992.

[4] Jaeschke, G., "Reciprocal Hashing --- a Method for Generating Minimal Perfect Hash Functions," Communication of the ACM, 24, 829-833, 1981.

[5] Cormack, G.V., Horspool R.N.S., and Kaiserwerth M., "Practical Perfect Hashing," The Computer Journal 28, 54-58, 1985.

[6] Ramakrishna, M. V., Larson, P., "File Organization Using Composite Perfect Hashing," ACM Transactions on Database Systems, 14, 231-263, 1989.

[7] Sprugnoli, R., "Perfect Hashing Functions: a Single Probe Retrieving Method for Static Sets," Communication of the ACM, 20, 841-850, 1978.

[8] Cichelli, R. J., "Minimal Perfect Hash Functions Made Simple," Communication of the ACM, 23, 17-19, 1980.

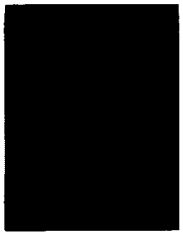
[9] Sager, T. J., "A Polynomial Time Generator for Minimal Perfect Hashing Functions," Communication of the ACM, 28, 523-532, 1985.

[10] Fox, E. A., Q. Chen, and L. Heath, S. Datta, "A More Cost Effective Algorithm for Finding Minimal Perfect Hash Functions." Proceeding of the Seventeenth Annual ACM Computer Science Conference, Feb. 21-23, 1989, Louisville, KY, 114-122.

[11] Fox, E. A., Q. Chen, and L. Heath, "An O(n log n) Algorithm for Finding Minimal Perfect Hash Functions," Blackburg, Va.: TR 89-10, Department of Computer Science, Virginia Polytechnic Institute and State University, 1989.

[12] Rosen, K., Elementary Number Theory and Its Applications, Addison Wesley, 1984.

[13] 김만수, 최동시, 박세영, "멀티미디어 백과 사전 CD-Rom Title (옥서) 설계 및 구현," 소프트웨어 연구부 연구 논문집, 한국 전자통신연구원, 397-400, 1995.



김 수 회

1979년 부산대학교 과학교육학과 졸업(이학사)

1986년 University of Georgia Dept. of Computer Science (MAMS)

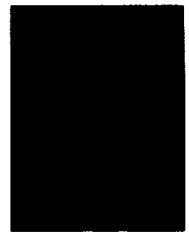
1988년 University of Georgia Dept. of Mathematics (MA)

1993년 University of South Carolina Dept. of Computer Science (Ph.D)

1993년 Benedict College Assistant Professor

1994년~현재 호서대학교 컴퓨터학부 부교수

관심분야 : 정보검색, 데이터베이스, 하이퍼미디어



박 세 영

1980년 경북대학교 전자공학과 졸업(학사)

1981년 KAIST 전산학과 졸업(석사)

1989년 France Paris 7 Universite 졸업(박사)

1996년 미국 University of Missouri 객원연구원

1982년~현재 한국전자통신연구원 책임연구원 자연어처리연구부 부장

관심분야 : 자연어처리, 한국어처리, 정보검색, 사용자 인터페이스, 인공지능