세그먼트 트리를 이용한 IP 주소 검색
(IP Address Lookup using Segment Trees)

이 인 복 † 박 근 수 †† 최 양 희 †† 정 성 권 †††
(Inbok Lee) (Kunsoo Park) (Yanghee Choi) (SungKwon Chung)

요 약 IP 주소 검색 문제는 주어진 IP 주소에 대해서 매칭되는 길이가 가장 긴 IP 접두사를 라우팅 테이블로부터 찾는 것으로서, 인터넷의 고속화에 장애가 되어 왔다. 본 논문에서는 세그먼트 트리라는 자료 구조를 이용하여 이 문제에 대한 새로운 알고리즘을 제시한다. 본 논문에서 제시하는 알고리즘은 n개의 접두사가 주어졌을 때 $O(\log n)$ 시간에 IP 접두사를 검색할 수 있다. 또 기존의 알고리즘에서 문제가 되었던 IP 접두사의 삽입과 삭제 연산을 자료 구조 전체를 다시 만들지 않고 효율적으로 처리한다.

Abstract The IP address lookup problem is to find the longest matching IP prefix for a given IP address from the routing table and has been a central bottleneck in speeding up the Internet. In this paper, we propose a new algorithm for this problem based on the segment tree data structure. Given n IP prefixes, our algorithm can do IP address lookup in $O(\log n)$ time. It also handles the insertion and deletion of IP prefixes efficiently without rebuilding the total data structure.

1. 서 론

인터넷에서 라우터는 입력받은 패킷의 목표 주소를 읽고, 다음과 같이 정의되는 라우팅 테이블을 이용하여 이에 해당하는 경로를 선택한다.

정의 1. m 비트 길이 IP 주소에 대한 라우팅 테이블 T는 (p, hop) 쌍의 집합이다. 여기서 p는 $|p| \le m$인 0과 1로 이루어진 스트링이며 IP 주소의 접두사이다. hop는 $1 \le hop \le H$ 인 정수인데, H는 이 라우터에 직접 연결된 다음 경로(next hop)들의 갯수이다. n은 라우팅 테이블 T에 저장된 IP 접두사의 수이다.

본 논문에서는 라우팅 테이블에서의 IP 주소 검색, 삽입, 삭제 문제를 고려하고자 한다.라우팅 테이블 T에서 임의의 IP 주소 x를 검색하는 문제는 다음 조건을 만족하는 $(p_x, hop_x) \in T$를 찾는 것이다.

1. p_x는 x의 접두사이다.
2. p가 x의 접두사인 다른 $(p, hop) \in T$에 대해서, $|p_x| > |p|$이어야 한다.

라우팅 테이블 T에 새로운 $(p', hop') \notin T$를 삽입하는 문제는 새로운 라우팅 테이블 $T' = T \cup \{(p', hop')\}$을 만드는 것이다. 또, T에서 $(p', hop') \in T$를 삭제하는 문제는 새로운 라우팅 테이블 $T' = T - \{(p', hop')\}$을 만드는 것이다.

모든 IP 주소와 해당하는 다음 경로를 라우팅 테이블에 저장할 경우, 단순한 테이블 조회만으로 IP 주소 검색을 할 수 있다. 그러나 m비트 길이 IP 주소에 대해서 2^m 크기의 라우팅 테이블이 필요하므로 현실적으로 쓰이기 어렵다. 따라서 정의 1과 같이 IP 주소의 접두사만을 저장하여 라우팅 테이블의 크기를 줄였는데[1], 대신 앞에서 설명한 IP 주소 검색 문제가 일어나게 되어 인터넷의 고속화에 장애가 되고 있다.

라우팅 테이블의 내용은 여러 가지 원인으로 변경될 수 있다. 네트워크가 추가되거나 삭제되는 경우 이외에도, 라우터의 설정 이상, 소프트웨어의 오류, 순간적인 물리적 연결 이상 등의 원인으로 접근 불가능한 경로가 생기면 인터넷의 위상(topology)이 바뀌게 되고[2,3], 이는 IP 접두사의 삽입이나 삭제로 이어진다.

현재의 표준인 IPv4에서는 IP 주소의 길이는 $m=32$

· 이 논문은 2000년도 두뇌 한국 21 사업에 의하여 지원되었음
† 학생회원 : 서울대학교 컴퓨터공학부
iblee@theory.snu.ac.kr
†† 종신회원 : 서울대학교 컴퓨터공학부 교수
kpark@theory.snu.ac.kr
ychoi@smart.snu.ac.kr
††† 비 회 원 : 유비쿼스(주) 대표이사
sung@ubiquix.com
논문접수 : 2000년 12월 11일
심사완료 : 2001년 8월 28일

이나, IP 주소 부족 문제를 풀기 위해 제안된 IPv6[4]에서는 $m=128$로 늘어났다. IPv6에서는 사용할 수 있는 IP 주소의 수가 늘어난 반면, 라우팅 테이블의 크기가 커지게 된다. 따라서 IP 주소의 검색, 삽입, 삭제 문제에 대한 효율적인 알고리즘이 필요하게 된다.

IP 주소 검색 문제에 대한 전통적인 접근 방법은 패트리시아 트라이(Patricia trie)[5]로, m비트 길이의 IP 주소에 대해서 $O(m)$ 시간에 검색을 수행한다. 패트리시아 트라이는 검색 시간이 느리고 메모리를 많이 차지하는 단점을 가져서 이를 개선하기 위한 많은 연구가 진행되어 왔다. 이들은 크게 별도의 하드웨어를 이용하는 방법과 소프트웨어에 기반한 방법으로 나눌 수 있다.

IP 주소 검색에 별도의 하드웨어를 이용하는 방법은 여러 가지가 있다. McAuley등[6]은 내용으로 검색 가능한 메모리(content-addressable memory, CAM)를 이용한 방법을 제안하였다. Pei등[7]은 라우팅 테이블을 RAM 대신 전용 VLSI에 옮겨 IP 주소 검색에 이용할 수 있음을 보였다. 하드웨어를 이용한 방법은 빠르게 IP 주소 검색을 할 수 있으나, 설계가 어렵고 추가적인 가격 부담이 있으며 인터넷 환경의 변화를 반영하기 어렵다. 이러한 문제를 고려해 볼 때, 소프트웨어에 기반한 방법이 보다 바람직하다.

Waldvogel, Varghese, Turner와 Plattner[8]는 해싱(hashing)을 이용한 알고리즘을 제안하였다. 이들의 알고리즘은 메모리를 많이 사용하지만 평균적으로 $O(\log m)$ 시간에 검색을 수행한다. IP 접두사는 다양한 길이를 갖고 있어서 검색이 어렵다는 점에 착안하여 Srinivasan과 Varghese는 IP 접두사의 길이를 제한하는 controlled prefix expansion[9]이라는 방법을 제안하였다. 이들의 방법은 제한된 IP 접두사 길이의 가짓수를 k라 할 때, IP 주소 검색에 최악의 경우 $O(k)$ 시간이 걸린다. Srinivasan, Lampson과 Varghese[10] 등은 IP 접두사를 정수의 구간으로 나타낸 후 이진 탐색을 하는 알고리즘을 제안하였다. 이들의 알고리즘은 다음 장에서 보다 상세히 설명된다.

CPU 연산에 비해 메모리의 참조는 시간이 많이 걸리므로, 라우팅 테이블을 압축하여 메모리 참조 횟수를 줄이고 빠른 캐시 메모리를 이용하게 할 수 있다. Brodnik, Carlsson과 Degermark[11]는 비트맵을 이용하여 라우팅 테이블을 나타내는 트라이를 작게 압축하는 방식을 제안하였다. 트라이를 압축하는 다른 알고리즘으로는 LC 트라이[12]가 있다. Crescenzi, Dardini와 Grossi[13]의 알고리즘은 직사각형 모양으로 나타낸 라우팅 테이블을 run length encoding으로 압축하여 3번의 메모리 참조만으로 IP 주소 검색을 할 수 있다.

위와 같이 IP 주소 검색 문제를 효율적으로 풀기 위한 많은 연구가 진행되었으나, 두 가지 문제점을 가지고 있다. 첫 번째 문제는 IP 접두사의 삽입과 삭제에서 생긴다. 기존의 알고리즘들은 IP 주소 검색의 속도를 높이기 위해 압축 등의 사전 연산을 이용하는데, 라우팅 테이블이 변경되었을 때에는 사전 연산을 다시 해야 하므로 IP 접두사의 삽입과 삭제에 문제가 생긴다. 두 번째 문제는 확장성에 있다. IPv4를 대상으로 최적화된 일부 알고리즘은 빠른 검색 속도를 얻기 위해서 주 메모리를 많이 사용하거나 캐시 메모리를 이용한다. 이 알고리즘들을 IPv6에 적용할 경우 크기가 커진 라우팅 테이블을 저장하기 위해서 거대한 주 메모리나 캐시 메모리가 필요하게 되는데, 이에 따른 가격 부담으로 알고리즘은 비현실적이 된다.

본 논문에서는 세그먼트 트리라는 자료 구조를 이용하여, IP 주소 검색 문제를 $O(\log n)$ 시간에 풀 수 있음을 보였다. 본 논문에서 제시하는 알고리즘은 기존의 알고리즘에서 문제가 되었던 IP 접두사의 삽입과 삭제 연산을 자료 구조 전체를 다시 만들지 않고 효율적으로 처리하며 메모리를 과도하게 이용하지 않으므로, 현재 32 bit 주소 (IPv4)에서 사용되는 알고리즘에 비해서 128 bit 주소 (IPv6)에서도 적용될 수 있는 확장성을 가진다.

본 논문은 다음과 같이 구성된다. 2장에서는 본 논문에서 사용하는 자료 구조인 세그먼트 트리에 대해 설명하고, 어떻게 IP 주소 검색 문제를 푸는데 쓰이는지를 설명한다. 3장에서는 논문의 결과를 요약하고 앞으로의 연구 과제를 언급한다.

2. 알고리즘

2.1 IP 접두사의 속성

k비트 길이의 IP 접두사 P의 2진수 표현이 $p_1p_2\cdots p_k$일 때, 이 뒤에 0과 1을 각각 $m-k$개 붙여서 얻은 정수를 $P_L=p_1p_2\cdots p_k0\cdots0_{(2)}$, $P_H=p_1p_2\cdots p_k1\cdots1_{(2)}$이라고 하자. 이 값들은 각각의 IP 접두사마다 유일하게 결정되므로 P를 구간 $[P_L, P_H]$로 표현할 수 있다. k 비트 길이의 IP 접두사 P를 나타내는 구간의 폭은 $l(P)=P_H-P_L+1=2^{m-k}$로 정의되고, m은 상수이므로 접두사의 길이가 길어질 수록 구간의 폭은 좁아진다. 또 P가 입력받은 IP 주소 x의 접두사일 때, x는 정규식(regular expression) $p_1p_2\cdots p_k(0-1)^{m-k}$로 표현되는 2진수이므로 $x\in[P_L, P_H]$가 된다.

주어진 IP 주소 x와 매칭되는 접두사는 둘 이상이 있을 수 있으므로, x가 포함되는 구간도 둘 이상이 있을 수 있다. IP 주소 검색 문제는 입력된 IP 주소 x와 매칭되는 길이가 가장 긴 IP 접두사 P를 찾는 것이므로, 이를 x가 속하면서 가장 폭이 좁은 구간 $[P_L, P_H]$를 찾는 문제로 바꾸어 생각할 수 있다.

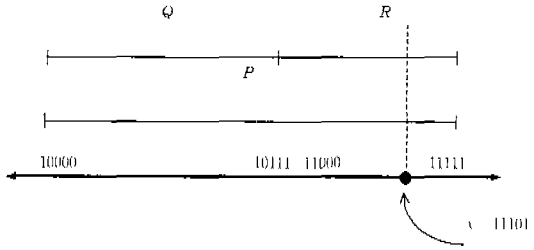

그림 1 IP 주소 검색의 예

예제 1. $m=5$이고, 라우팅 테이블에 세 IP 접두사 $P=1$, $Q=10$, $R=11$이 있다고 하자. (그림 1) 이들은 세 구간 $[P_L, P_H]=[10000_{(2)}, 11111_{(2)}]$, $[Q_L, Q_H]=[10000_{(2)}, 10111_{(2)}]$, $[R_L, R_H]=[11000_{(2)}, 11111_{(2)}]$로 표현된다. IP 주소 $x=11101_{(2)}$가 입력되었을 때, $x\in[P_L, P_H]$, $x\in[R_L, R_H]$이고 $x\notin[Q_L, Q_H]$이다. 이는 P, R이 x의 접두사인 반면 Q는 x의 접두사가 아님을 나타낸다. $l(P)=16$이고 $l(R)=8$이므로, x가 포함되면서 가장 길이가 짧은 구간은 $[R_L, R_H]$이고 R이 x와 매칭하는 길이가 가장 긴 접두사가 된다.

Varghesc 등의 알고리즘[10]은 위의 성질을 이용한다. 라우팅 테이블의 모든 접두사 P에 대해서 P_L, P_H를 계산한 후 이들을 크기 순으로 정렬한다. 다음 정렬된 값들에 대해서 매칭하는 길이가 가장 긴 IP 접두사가 무엇인지를 계산한다. 이 사전 연산은 $O(n)$ 시간에 이루어진다. IP 주소 검색은 정렬된 P_L, P_H들에서 x를 이진 탐색하고, 탐색이 종료한 위치에서 사전 연산을 통해 계산된 IP 접두사를 읽는 것으로 $O(\log n)$ 시간에 이루어진다.

IP 주소 검색 문제를 1차원 공간으로 옮겨서 생각해 보자. IP 주소 x는 좌표가 x인 정수점으로 표현된다. IP 접두사 P는 선분 $\overline{P_L P_H}$로 표현되며, 정수점들의 집합 $\{P_L, P_L+1, \cdots, P_H\}$를 나타낸다. x가 포함되는 가장 좁은 구간 $[P_L, P_H]$를 찾는 것은 점 x를 지나는 가장 짧은 선분 $\overline{P_L P_H}$를 찾는 것과 같다.

두 선분이 서로 겹치는 부분을 가지면서 한 선분이 다른 선분에 포함되지 않는 관계를 가지고 있을 때, 이들을 서로 교차한다고 하자. IP 접두사를 나타내는 선분은 다음과 같은 성질을 갖고 있다.

보조정리 1. IP 접두사를 선분으로 나타낼 때 이 중 어떤 두 선분도 서로 교차하지 않는다.

증명. 두 IP 접두사 P, Q와 이들을 나타내는 선분 $\overline{P_L P_H}$, $\overline{Q_L Q_H}$가 있다고 하자. $\overline{P_L P_H}$와 $\overline{Q_L Q_H}$가 서로 교차한다면 일반성을 잃지 않고 $P_L < Q_L < P_H < Q_H$의 관계가 성립한다고 가정할 수 있다. P의 길이가 a일 때 P_L, P_H는 각각 $P_L = p_1 p_2 \cdots p_a 0 \cdots 0_{(2)}$, $P_H = p_1 p_2 \cdots p_a 1 \cdots 1_{(2)}$이다. $P_L < Q_L < P_H < Q_H$이고 Q_L은 IP 접두사를 나타낸 선분의 왼쪽 끝이므로, Q_L의 2진수 표현은 $p_1 p_2 \cdots p_a$로 시작하고 $m-k$개 미만의 연속된 0으로 끝나게 된다. 끝의 연속된 0을 모두 1로 바꾸어 Q_H를 얻으면 $Q_H < P_H$가 되어 모순이 된다. 따라서 IP 접두사를 나타내는 선분은 서로 교차하지 않는다. □

2.2 세그먼트 트리(segment tree)

세그먼트 트리[14]는 계산 기하학에서 선분을 저장하기 위해 사용하는 자료 구조이다. IP 주소 검색 문제에서는 좌표가 정수인 점들만을 다루므로 세그먼트 트리는 [14]의 정의와는 약간 다르게 다음과 같이 정의할 수 a있다. 수직선 상에 있는 각각의 선분 $\overline{P_L P_H}$마다 네 값 P_L-1, P_L, P_H, P_H+1을 계산한다. 이 값들에서 가장 작은 값과 가장 큰 값을 뺀 수들의 집합을 $\{u_1, u_2, \cdots, u_N\}$이라고 하자. 세그먼트 트리는 루트를 가진 균형잡힌 트리(balanced rooted tree)로서, 트리의 단말 노드는 각각 기본 구간 $[u_1, u_2]$, $[u_3, u_4]$, $\cdots$, $[u_{N-1}, u_N]$을 나타낸다. 부모 노드가 나타내는 구간은 두 자식 노드가 나타내는 구간을 연결한 것이다. 즉 자식 노드가 나타내는 구간이 각각 $[u_i, u_j]$, $[u_{j+1}, u_k]$일 때 부모 노드가 나타내는 구간은 $[u_i, u_k]$이다. 이렇게 정의된 세그먼트 트리의 높이는 $O(\log n)$이다. 이후로 세그먼트 트리의 높이를 h라고 하자.

세그먼트 트리의 각 노드는 자신이 나타내는 구간 외에, 별도의 자료 구조와 이를 가리키는 포인터를 두어 선분을 저장한다. 선분을 저장하는 자료 구조는 여러 가지가 가능하나, 선분의 삽입과 삭제에 $O(1)$시간이 걸리도록 단순한 양방향 링크드 리스트를 쓰기로 한다.

예제 2. 전체 구간이 $[0,9]$이고, $[0,9]$, $[0,6]$, $[3,4]$의 세 선분이 있다고 하자. (그림 2) 이 때 기본 구간은 $[0,2]$, $[3,4]$, $[5,6]$, $[7,9]$이고, 해당하는 세그먼트 트리는 그림 3과 같다.

세그먼트 트리에서 선분을 삽입하는 과정은 다음과

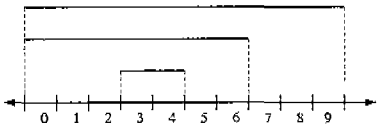

그림 2 구간 [0, 9], [0. 6], [3, 4]

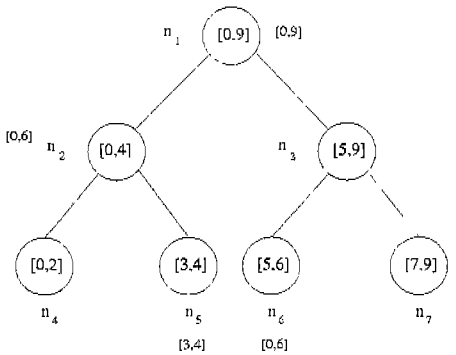

그림 3 그림 2에 대한 세그먼트 트리

같다. 루트 노드에서부터 노드가 나타내는 구간이 삽입될 선분에 포함되는지를 조사한다. 만약 포함된다면, 노드가 가리키는 자료 구조에 선분을 삽입한다. 그렇지 않고 삽입될 선분과 자식 노드가 나타내는 구간이 겹치는 부분이 있다면 같은 작업을 해당하는 자식 노드에 대해서 재귀적으로 수행한다. 이 작업은 $O(h)$ 시간에 이루어진다. (증명은 [15]를 참조)

예제 2의 경우 선분 [0,6]이 삽입되는 과정을 살펴보자. 이 선분은 루트 노드 n_1의 구간 [0,9]을 포함하지 않지만, 자식 노드 n_2와 n_3의 구간 [0,4]와 [5,9]에 모두 겹친다. 따라서 삽입 연산은 노드 n_2와 n_3에서 계속된다. 노드 n_2에서는 이 노드의 구간이 선분에 포함되므로 선분을 저장한다. 노드 n_3의 구간은 선분에 포함되지 않지만, 자식 노드 n_6의 구간 [5,6]이 선분과 겹치므로 삽입 연산은 노드 n_6에서 계속된다. 노드 n_6에서는 이 노드의 구간이 선분에 포함되므로 선분을 저장한다. 따라서 선분 [0,6]은 노드 n_2와 n_6에 나뉘어 저장된다. 비슷한 과정을 거쳐서 선분 [3,4]는 노드 n_5에, 선분 [0,9]는 노드 n_1에 저장된다.

저장된 선분을 세그먼트 트리에서 삭제하기 위해서는 선분들의 정보가 기록된 별도의 테이블을 이용한다 [15]. 이 테이블은 각각의 선분마다 이들이 나뉘어 저장되어 있는 위치를 리스트 형태로 기록한다. 테이블에는 n개의 선분이 저장되어 있으므로 삭제될 선분은 $O(\log n)$ 시간에 찾을 수 있다. 다음 리스트를 따라가면서 이 선분이 저장된 위치에 삭제 연산을 수행한다. 선분은 최대 $O(h)$ 부분으로 나뉘어 저장되므로, 이 과정은 $O(h)$ 시간이 걸린다. 따라서 선분의 삭제 연산에 대한 전체 시간 복잡도는 $O(h)$이다.

이렇게 만들어진 세그먼트 트리는 다음과 같이 이용될 수 있다.

예제 3. 세그먼트 트리를 이용하여 임의의 점 x를 지나는 선분을 모두 나열할 수 있다. 이는 루트 노드에서부터 x가 포함되는 구간을 따라 내려가면서, 이 과정에서 방문하는 각 노드가 갖고 있는 선분들을 나열하면 된다. 그림 3의 세그먼트 트리에서 $x=3$이 주어진다면, n_1, n_2, n_5의 차례로 노드들을 방문하게 되고, 선분 [0,9], [0,6], [3,4]를 지남을 알 수 있다. 위 과정은 $O(h+k)$ 시간이 걸리며, k는 x를 지나는 선분의 개수이다.

2.3 IP 주소 검색 알고리즘

본 논문에서 제안하는 알고리즘은 앞 절에서 설명한 세그먼트 트리를 이용한다. 라우팅 테이블의 모든 IP 접두사를 선분으로 나타내고, 이 선분들을 세그먼트 트리에 저장한다. 예제 3에서 볼 수 있듯이 세그먼트 트리를 이용하여 한 점을 지나는 모든 선분을 나열하는 것이 가능하므로, 남은 문제는 이 중에서 가장 짧은 선분을 찾는 것이다.

보조정리 2. 세그먼트 트리에서 한 노드에 저장되는 선분들은 서로 다른 길이를 갖는다.

증명. 만약 한 노드에 같은 길이의 서로 다른 선분 둘이 저장되어 있다고 생각하자. 이 선분을 $\overline{P_LP_H}$, $\overline{Q_LQ_H}$라고 하면, $\overline{P_LP_H}$와 $\overline{Q_LQ_H}$는 서로 다른 선분이므로 이들은 서로 교차해야 하는데, 이는 보조정리 1에 어긋난다. □

보조정리 2를 통해서, 세그먼트 트리의 각 노드에 저장된 선분 중에서 가장 짧은 선분이 유일하게 존재함을 알 수 있다. 이제 이 선분을 가리키는 포인터를 각 노드에 추가한다. 선분을 새로 노드에 삽입할 때에는 기존의 가장 짧은 선분과 비교한 후 더 짧은 쪽을 가리키도록 포인터를 $O(1)$ 시간에 갱신할 수 있다. 노드에 저장된 선분들이 삭제될 확률이 모두 같다고 가정하면, 선분을 노드에서 삭제할 때는 평균 $O(1)$ 시간에 포인터를 갱신할 수 있다. 수정된 세그먼트 트리에서 선분의 삽입 연산은 $O(h)$ 시간, 삭제 연산은 평균 $O(h)$ 시간이 걸린다.

이제 트리를 내려가는 과정에서 방문한 노드가 가진

모든 선분 대신, 이 중 가장 짧은 선분들만을 나열하여 $O(h)$ 길이의 리스트를 얻을 수 있다. x를 지나는 가장 짧은 선분은 $O(h)$ 번의 비교를 통해 찾을 수도 있으나, 아래의 보조정리 3을 통해서 비교 없이 찾을 수 있다.

보조정리 3. 세그먼트 트리에서 임의의 IP 주소 x를 검색할 때, 노드 i를 먼저 방문한 후 노드 j를 방문하고, 노드 i, j가 가진 가장 짧은 선분을 각각 $\overline{P_LP_H}$, $\overline{Q_LQ_H}$라 하자. 그러면 $\overline{P_LP_H}$는 $\overline{Q_LQ_H}$를 포함한다.

증명. 보조정리 1에 의해서 선분 $\overline{P_LP_H}$와 $\overline{Q_LQ_H}$는 서로 교차하지 않고, 두 선분은 모두 좌표 x를 지나므로 서로 포함 관계에 있다. 선분 $\overline{Q_LQ_H}$가 노드 i가 나타내는 구간을 포함한다면 $\overline{Q_LQ_H}$는 노드 i에 저장되었어야 하므로, 노드 i가 나타내는 구간 중 $\overline{Q_LQ_H}$에 속하지 않는 부분이 존재한다. 반면 선분 $\overline{P_LP_H}$는 노드 i가 나타내는 구간을 포함하므로 $\overline{P_LP_H}$는 $\overline{Q_LQ_H}$를 포함하게 된다.(그림 4) □

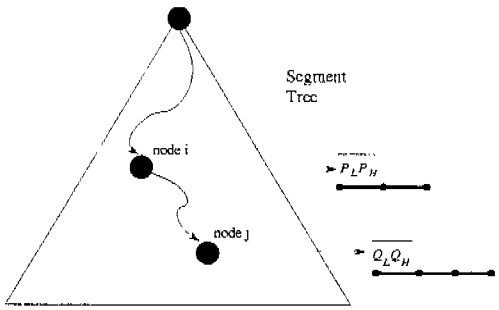

그림 4 최소 길이 IP 접두사 검색의 과정

보조정리 3을 통해서 검색 과정에서 마지막에 만난 선분이 x를 지나는 가장 짧은 선분임을 알 수 있다. 이 선분이 나타내는 IP 접두사가 x와 매칭하는 길이가 가장 길다.

정리 1. 세그먼트 트리를 이용하여 IP 주소 검색 문제를 $O(h)$ 시간에 풀 수 있다.

증명. IP 주소 x를 세그먼트 트리에서 검색할 때 노드에 저장된 가장 짧은 선분은 포인터를 통해서 $O(1)$ 시간에 찾을 수 있고, 세그먼트 트리의 높이가 h이므로 검색은 총 $O(h)$ 시간이 걸린다. 검색 과정에서 별도의 시간 부담 없이 x를 지나는 가장 짧은 선분을 찾을 수 있으므로, 전체 시간 복잡도는 $O(h)$이다. □

2.4 선분의 삽입과 삭제

세그먼트 트리에 새로운 선분을 삽입할 때, 양 끝점이 이전에 저장된 선분들의 양 끝점으로 쓰였을 때에는 트리의 모양을 바꾸지 않고 새로운 선분을 트리에 삽입할 수 있다.

이전에 쓰이지 않았던 끝점을 갖는 선분 $[c,d]$를 삽입하는 경우를 생각해보자. 세그먼트 트리에서 c, d가 각각 포함되는 구간을 따라 내려가서 $a<c<b$, $e<d<f$ 인 $[a,b]$, $[e,f]$ 구간을 나타내는 두 단말 노드를 찾아내면, 그림 5와 같이 새로운 단말 노드를 만들 수 있다. 이 작업은 $O(1)$ 시간에 이루어질 수 있으므로 기존의 선분 삽입과 시간 복잡도가 같다.

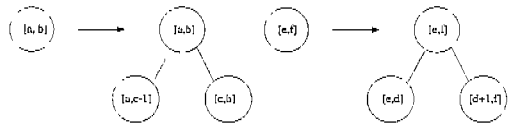

그림 5 새로운 끝점을 갖는 선분의 삽입

세그먼트 트리에서 선분의 삭제는 2.2절에서 설명한 연산을 수행하고, 마지막 단계에서 다음의 두 조건이 만족되는지를 검사한다.

1. 선분 삭제 연산에 의해서 저장된 선분이 없는 단말 노드가 생겼다.
2. 이 단말 노드의 형제 노드에도 저장된 선분이 없다.

이 경우에는 그림 6과 같이 노드를 삭제한다. 이 연산도 $O(1)$ 시간에 이루어지므로, 선분의 삭제가 노드의 삭제로 이어지더라도 시간 복잡도는 같다. 그 외의 경우에는 선분만이 삭제되고, 트리의 모양은 바뀌지 않는다.

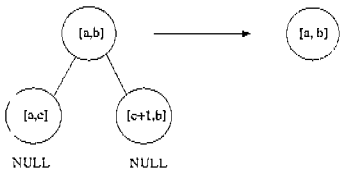

그림 6 세그먼트 트리에서 노드의 삭제

노드의 삽입과 삭제 연산을 세그먼트 트리에 추가할 경우, 트리의 높이가 바뀌게 된다. 노드의 삽입만이 무작위로 이루어질 경우 트리의 높이 h는 $O(\log n)$이 보장되지만[16], 삽입과 삭제가 혼합되면 트리의 높이가 $O(\log n)$임을 보장할 수 없다. [17, 18]에서 제시된 알고리즘은 별도의 추가 정보 없이 트리의 재구성을 통하여 트리의 높이를 $O(\log n)$로 보장한다. 트리의 단말 노드의 수를 $|T|$, $d(T)$를 마지막으로 트리를 재구성한 이후 삭제 연산을 한 횟수로 정의하면, 다음의 두 경우 트

리를 재구성한다. 여기에서 c와 b는 $c>1$과 $b>0$을 만족하는 상수이다. (예, $c=2$, $b=1$)

- 새로 삽입된 노드에서 루트 노드까지의 간선의 수가 $\lceil c\log(|T|+d(T))\rceil$를 넘었을 때
- 삭제 연산을 할 때마다 $d(T)$를 하나씩 증가시켜서, $d(T)\geq(2^{b/c}-1)|T|$가 되었을 때

트리의 높이가 $O(\log n)$이므로 정리 1에 의하여 IP 주소 검색의 시간 복잡도가 $O(\log n)$이 된다. 트리의 재구성 과정은 정확한 IP 주소 검색을 위해서가 아니라 검색의 속도 보장을 위해서이므로, 기존의 알고리즘에서의 사전 연산과는 목적이 다르다.

2.5 동적 세그먼트 트리

Van Kreveld와 Overmars는 동적 세그먼트 트리(dynamic segment tree)[19]를 제안하였다. 동적 세그먼트 트리는 전통적인 세그먼트 트리의 제약조건을 완화시키고 유니온-카피 구조(union-copy structure)를 사용하여 새로운 양 끝점을 갖는 선분도 트리에 삽입될 수 있게 하였다.

n개의 선분이 저장된 동적 세그먼트 트리에서 연산의 시간 복잡도는 다음과 같다.

- 동적 세그먼트 트리의 생성은 $O(n\log n)$ 시간과 공간을 차지한다.
- 선분의 검색은 $O(\log n)$ 시간에 이루어진다.
- 새로운 선분의 삽입은 새로운 끝점의 삽입까지 포함하여 $O(\log n)$ 시간이 걸린다.
- 선분을 동적 세그먼트 트리에서 삭제하는데는 $O(\log n\cdot\alpha(i,n))$ 시간이 걸리는데, i는 알고리즘 내부에서 사용하는 상수이고 $\alpha(i,n)$ 함수는 애커먼 함수(Ackerman function)의 역이다.

동적 세그먼트 트리는 매우 복잡한 구조를 가지고 있어서 구현이 어려우므로, 빈번한 삽입과 삭제가 벌어지는 최악의 경우에서 좋은 시간 복잡도를 얻고자 할 때 적합하다.

이 외에도, Mehlhorn[20]은 무게 균형 트리(weighted balanced tree)를 이용하여 새로운 양 끝점을 갖는 선분의 삽입을 허용하는 세그먼트 트리를 구현할 수 있음을 보였다.

위의 두 자료구조 모두 2.3 절에서 설명한 방법으로 IP 주소 검색에 쓰여질 수 있다. [19]의 경우에서는 보조정리 3이 성립하지 않지만 $O(\log n)$ 번의 비교를 통해서 가장 짧은 선분을 구할 수 있으므로 IP 주소 검색의 시간 복잡도는 변하지 않는다.

3. 결 론

본 논문에서는 IP 주소 검색 문제를 계산 기하학의 문제로 바꾸어 생각할 수 있고, 세그먼트 트리라는 자료구조를 통해 풀 수 있음을 보였다. 본 논문에서 제안된 알고리즘은 IP 주소를 $O(\log n)$ 시간에 검색할 수 있고 (n은 라우팅 테이블에 저장된 IP 접두사의 수), 기존의 알고리즘에 비하여 유연하게 IP 접두사의 삽입과 삭제를 할 수 있다.

이론적인 차원에서 앞으로의 연구 과제는, IP 접두사의 속성을 이용하여 보다 빠르고 간단한 자료 구조를 만드는 것이다. 동적 세그먼트 트리와 같은 복잡한 내부 구조를 갖지 않으면서 IP 접두사의 삽입과 삭제를 효율적으로 처리할 수 있는 자료구조를 찾을 수 있을 것으로 예상된다.

이론적인 복잡도 외에도, IP 주소 검색 문제는 실제 인터넷에서 생기는 문제이므로 실제 구현에서의 효율도 중요하다. 실행 시간의 대부분은 메모리 참조가 차지하므로, [9]와 비슷하게 세그먼트 트리를 한 노드의 자식의 개수가 t인 t-ary 트리로 구현하여 메모리 참조 횟수를 줄일 수 있다.

참 고 문 헌

[1] V. Fuller, T. Li, J. Yu and K. Vardhan. Classless Inter-domain Routing(CIDR): An Address Assignment and Aggregation Strategy, *RFC 1519*, 1993

[2] Internet Performance Measurement and Analysis Project(IPMA). *http://www.merit.edu/ipma*

[3] C. Labovitz, G. Malan and F. Jahanian. Internet Routing Instability. *ACM SIGCOMM 97*, 1997

[4] S. Deering and R. Hinden. Internet Protocol, Version 6(IPv6), *RFC 1883*, 1996

[5] R. Sedgewick. *Algorithms in C++*. Addison-Wesley, 1992

[6] A. McAuley and P. Francis. Fast Routing Table Lookup Using CAMs. *IEEE INFOCOM 93*, 1993

[7] T. Pei and C. Zukowski. Putting Routing Tables in Silicon. *IEEE Network magazine*, January 1992

[8] M. Waldvogel, G. Varghese, J. Turner and B. Plattner. Scalable High Speed IP Routing Lookups. *ACM SIGCOMM 97*, 1997

[9] V. Srinivasan and G. Varghese. Faster IP Lookups Using Controlled Prefix Expansion. *ACM SIGMETRICS 98*, 1998

[10] B. Lampson, V. Srinivasan and G. Varghese. IP Lookups Using Multiway and Multicolumn Search. *IEEE INFOCOM 98*, 1998

[11] A. Brodnik, S. Carlsson and M. Degermark. Small Forwarding Tables for Fast Routing Lookups. *ACM SIGCOMM 97*, 1997
[12] S. Nilsson and G. Karlsson. Fast Addresss Lookup for Internet Routers. *IEEE Broadband Communications*, 1998
[13] P. Crescenzi, L. Dardini and R. Grossi. IP Address Lookup Made Fast and Simple. *ESA 99*, 1999
[14] F. P. Preparata and M. I. Shamos. *Computational Geometry-An Introduction*. Springer-Verlag, 1995
[15] J. L. Bentley and D. Wood, An Optimal Worst Case Algorithm for Reporting Intersections of Rectangles. *IEEE Transactions on Computers*, 29(7):571-576, July 1980
[16] T. H. Cormen, C. E. Leiserson and R. L. Rivest, *Introduction to Algorithms. MIT Press*, 1990
[17] A. Anderson. General Balanced Trees. *Journal of Algorithms*, 30:1-18, 1999
[18] I. Galperin and R. L. Rivest. Scapegoat Trees. Proceedings of the Fourth Annual ACM-SIAM Symposium on Discrete Algorithms, p. 165-174, 1993
[19] M. J. van Kreveld and M. H. Overmars. Union-Copy Structures and Dynamic Segment Trees. *Journal of ACM*, 40(3):635-652, July 1993
[20] K. Melhorn. *Data Structures and Efficient Algorithms Vol. III: Multi-dimensional Searcning and Computational Geometry*. Springer-Verlag, 1984

이 인 복
1993년 3월 ~ 1997년 2월 서울대학교 컴퓨터공학과 학사. 1997년 3월 ~ 1999년 2월 서울대학교 컴퓨터공학과 석사. 1999년 3월 ~ 현재 서울대학교 전기컴퓨터공학부 박사과정. 관심분야는 알고리즘 설계 및 분석, 스트링 알고리즘

박 근 수
정보과학회논문지 : 시스템 및 이론
제 28 권 제 7 호 참조

최 양 희
1975년 서울대학교 전자공학 학사. 1977년 한국과학기술원 전자공학 석사. 1977년 ~ 1979년 한국통신기술연구소. 1980년 ~ 1984년 프랑스 ENST대학교 전산학과 박사. 1984년 ~ 1991년 한국전자통신연구소. 1988년 ~ 1989년 미국 IBM 왓슨연구소. 1991년 ~ 현재 서울대학교 컴퓨터공학과 교수. 관심분야는 멀티미디어 시스템 및 초고속 망

정 성 권
1982년 서울대학교 컴퓨터공학과 학사. 1984년 서울대학교 컴퓨터공학과 석사. 1990년 University of Washington, Department of Computer Science & Engineering, 박사. 1998년 ~ 2000년 서울대학교 컴퓨터공학과 초빙교수. 1996년 ~ 현재 유비쿼스 주식회사 대표이사. 관심분야는 이동 컴퓨팅, 분산 시스템 등임