계산복잡도 개론
검색 문제

알고리즘 해석
강의 슬라이드 #8
도경구

키를 비교함으로만 검색을 수행하는 경우의 하한

- **검색(Searching) 문제**:
  - n개의 키를 가진 배열 S와 어떤 키 x가 주어졌을 때, $x = S[i]$가 되는 첨자 i를 찾는다.
  - 만약 x가 배열 S에 없을 때는 오류로 처리한다.
- **이분 검색 알고리즘**:
  - 배열이 정렬이 되어 있는 경우 시간복잡도가 $W(n) = \lfloor \lg n \rfloor + 1$가 되어서, 매우 효율적인 알고리즘이라고 할 수 있다.
  - 이 보다 더 좋은(빠른) 알고리즘은 존재할까?
    답: 키를 비교함으로만 검색을 수행하는 경우에는 이진 검색알고리즘 보다 더 좋은 알고리즘은 존재할 수 없다.

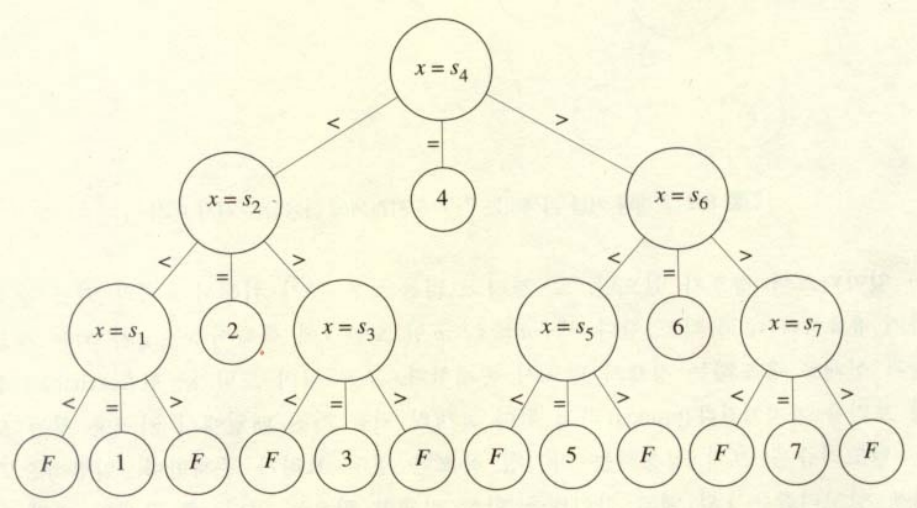
2001-03-19 알고리즘 강의 슬라이드 8 3

검색문제의 하한 찾기

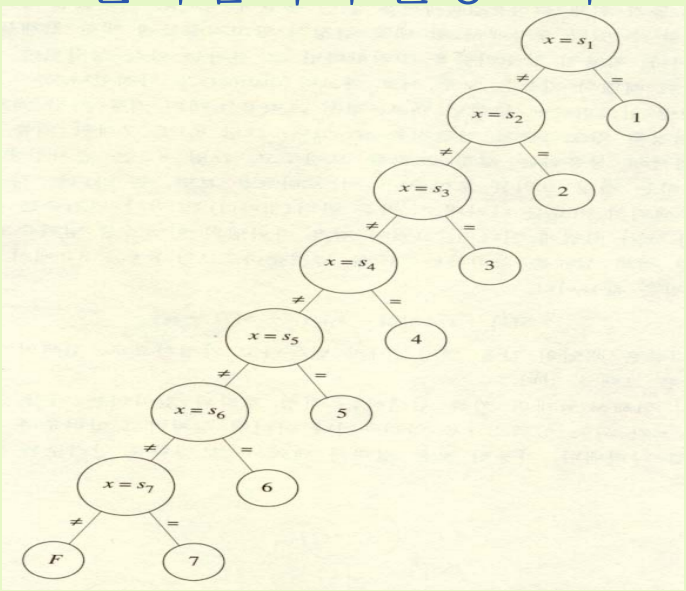
- 보기: 7개의 키를 검색하는 문제를 생각해 보자.
- 검색의 결정트리 (decision tree)
  - 큰 마디(내부마디) – 키와 각 아이템을 비교하는 마디
  - 작은 마디(잎마디) - 검색결과
  - 뿌리마디에서 잎마디까지의 경로는 검색하면서 비교하는 과정을 보여준다.
- 이진검색의 결정트리
  - 각 경로는 최대한(최악의 경우) 3개의 비교하는 마디(큰 마디)를 가진다.
- 순차검색의 결정트리
  - 각 경로는 최대한(최악의 경우) 7개의 비교하는 마디(큰 마디)를 가진다.

2001-03-19 알고리즘 강의 슬라이드 8 4

이진검색의 결정트리



순차검색의 결정트리



최악의 경우 하한 찾기

- 최악의 경우 비교하는 횟수는 결정트리의 뿌리마디에서 잎마디까지 가장 긴 경로상의 비교마디의 수 인데, 이는 트리의 깊이(depth) + 1이다.
- **보조정리 1**: n이 이진트리의 마디의 개수이고, d가 깊이 이면, $d \geq \lfloor \lg n \rfloor$

  증명:

$$n \leq 1 + 2 + 2^2 + \cdots + 2^d$$

$$n \leq 2^{d+1}$$

$$\lg n < d + 1$$

$$\lfloor \lg n \rfloor \leq d$$

- **보조정리 2**: n개의 다른 키 중에서 어떤 키 x를 찾는 가지친 유효 결정트리(pruned, valid decision tree)는 비교하는 마디가 최소한 n개 있다.
- **정리**: n개의 다른 키가 있는 배열에서 어떤 키 x를 찾는 알고리즘은 키를 비교하는 횟수를 기준으로 했을 때 최악의 경우 최소한 $\lfloor \lg n \rfloor + 1$만큼의 비교를 한다.

  증명: 그 알고리즘의 결정트리에서 비교하는 마디 만으로 이루어진 이진트리를 보면, 최악의 경우 비교횟수는 이진트리의 뿌리마디에서부터 잎마디까지의 모든 경로 가운데 가장 긴 경로의 마디 수가 된다. 그 수는 이진트리의 깊이 + 1이 된다. 그런데 보조정리 2에서 이 이진트리의 마디 수는 n이라 하였으므로, 그 이진트리의 깊이는 $\lfloor \lg n \rfloor$ 보다 크거나 같다.

2001-03-19 알고리즘 강의 슬라이드 8 7

평균의 경우 하한 찾기

- 최악의 경우와 비슷한 방법으로 구해보면 $\lfloor \lg n \rfloor - 1$이 된다.

2001-03-19 알고리즘 강의 슬라이드 8 8

보간 검색

- 검색의 하한을 개선할 수 있는가?
  - ✓ 비교하는 이외에, 다른 추가적인 정보를 이용하여 검색할 수 있다면 가능하다.
- 보기: 10개의 정수를 검색하는데, 여기서 첫번째 정수는 0-9중에 하나이고, 두 번째 정수는 10-19중에 하나이고, 세 번째 20-29중에 하나이고, …, 열번째 정수는 90-99중에 하나라고 하자. 이 경우 만약 검색 키 x가 0보다 작거나, 99보다 크면 바로 검색이 실패임을 알 수 있고, 그렇지 않으면 검색 키 x와 S[1+x div 10]과 비교해 보면 된다.
- 일반적으로 이 보기 만큼 자세한 정보를 항상 가질 수 있다고 보기에는 무리가 있다. 그러나 일반적으로 데이터가 가장 큰 값과 가장 작은 값이 균등하게 분포되어 있다고 가정해 보는 것은 타당성이 있어 보인다. 이 경우 반으로 무작정 나누는 대신에, 키가 있을 만한 곳을 바로 가서 검사해 보면 더욱 빨리 검색을 마칠 수가 있을 것이다. 이런 식의 검색알고리즘을 **보간검색(interpolation search)**이라고 한다.

2001-03-19 알고리즘 강의 슬라이드 8 9

선형보간법(Linear Interpolation)

$$mid = low + \left\lfloor \frac{x - S[low]}{S[high] - S[low]} \times (high - low) \right\rfloor$$

- **보기**: S[1] = 4이고 S[10] = 97일 때 검색 키가 25이면, $mid = 3$.
- **분석**:
  - ✓ 아이템이 균등하게 분포되어 있고, 검색 키가 각 슬롯에 있을 확률이 같다고 가정하면, 선형보간검색의 평균적인 시간복잡도는 $A(n) \approx \lg(\lg n)$이 된다. 예로서 n이 10억이라면, $\lg(\lg n)$은 약 5가 된다.
  - ✓ 단 보간검색의 단점은 최악의 경우의 시간복잡도가 나쁘다는 것이다. 예로서 10개의 아이템이 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 이고, 여기서 10을 찾으려고 한다면, *mid*값은 항상 *low*값이 되어서 모든 아이템과 비교를 해야 한다. 따라서 최악의 경우 시간복잡도는 순차검색과 같다.

2001-03-19 알고리즘 강의 슬라이드 8 10

보강된 보간검색법 (Robust Interpolation Search)

- *gap*이란 변수를 항상 *mid* - *low*와 *high* - *mid*보다 항상 작도록 유지하여 다음과 같이 *mid*를 구한다.
  1. gap = $\lfloor (high - low + 1)^{1/2} \rfloor$
  2. *mid*값을 위와 같이 선형보간법으로 구한다.
  3. 다음식으로 새로운 *mid*값을 구한다.

     $mid = minimum(high - gap, maximum(mid, low + gap))$
- **보기**: S[1] = 4이고 S[10] = 97이고, 검색 키가 25이면, $mid = 4$
- **분석**: 아이템이 균등하게 분포되어 있고, 검색 키가 각 슬롯에 있을 확률이 같다고 가정하면, 보강된 보간검색의 평균 시간복잡도는 $A(n) \approx \lg(\lg n)$이 되고, 최악의 경우는 $W(n) \approx (\lg n)^2$

2001-03-19 알고리즘 강의 슬라이드 8 11

트리 구조를 사용한 동적검색

- **정적검색(Static searching):** 데이터가 한꺼번에 저장되어 추후에 추가나 삭제가 이루어지지 않는 경우에 이루어 지는 검색, 즉 예를 들면 OS명령에 의한 검색.
- **동적검색(Dynamic searching):** 데이터가 수시로 추가 삭제되는 유동적인 경우, 예로서 비행기 예약 시스템.
- 배열 자료구조를 사용하면, 정적검색의 경우는 문제없이 이진검색이나 보간검색 알고리즘을 적용할 수 있지만, 동적검색의 경우는 이 알고리즘을 적용하기가 불가능하다. 따라서 트리 구조를 사용하여야 한다.

2001-03-19 알고리즘 강의 슬라이드 8 12

이진검색트리(Binary Search Tree)

- **정의**: 이진검색트리는 다음 조건을 만족하는 이진 트리이다.
  - ✓ 각 마디 마다 하나의 키가 할당되어 있다.
  - ✓ 어떤 마디 *n*의 왼쪽부분트리(left subtree)에 속한 모든 마디의 키는 그 마디 *n*의 키 보다 작거나 같다.
  - ✓ 어떤 마디 *n*의 오른쪽부분트리(right subtree)에 속한 모든 마디 *n*의 키 보다 크거나 같다.

2001-03-19 알고리즘 강의 슬라이드 8 13

이진검색트리를 사용하는 이유

- 부모중간횡단(inorder traversal)을 하면 정렬된 순서로 아이템을 추출할 수 있다.
- 평균 검색시간을 낮게 유지한다:
  - ✓ 동적으로 트리에 아이템이 추가되고 삭제되므로 트리가 항상 균형(balance)을 유지한다는 보장이 없다.
  - ✓ 특히 최악의 경우는 연결된 리스트(linked list)를 사용하는것(skewed tree)과 같은 효과를 준다.
  - ✓ 그러나 랜덤(random)하게 아이템이 트리에 추가되는 경우는 대체로 트리가 균형을 유지한다고 볼 수 있으므로 평균적으로 효율적인 검색시간을 기대할 수 있다.
- 이진검색트리를 이용하면 빨리 아이템을 추가하고 삭제한다.

2001-03-19 알고리즘 강의 슬라이드 8 14

이진검색트리 검색의 분석

- **정리**: 검색하는 아이템 x가 n개의 아이템 중의 하나가 될 확률이 동일하다고 가정하면, n개의 아이템을 가진 모든 입력을 검색하는 데 걸리는 평균시간은 이진검색트리를 사용하면 대략 $A(n) = 1.38 \lg n$이 된다.
- **증명**
  - ✓ k번째로 작은 아이템이 뿌리마디에 위치하고 있는 n개의 아이템을 가진 모든 이진검색트리들을 생각해 보자.
  - ✓ 이 트리들은 모두 왼쪽 부분트리에 k-1개의 마디가 있고, 오른쪽 부분트리에 n-k개의 마디가 있다.
  - ✓ $A(k\text{-}1)$을 왼쪽 부분트리를 검색하는 평균검색시간 이라고 하고, $A(n\text{-}k)$을 오른쪽 부분트리를 검색하는 평균검색시간 이라고 하자.
  - ✓ 그러면 x가 왼쪽 부분트리에 있을 확률은 $(k\text{-}1)/n$이 되고, 오른쪽 부분트리에 있을 확률은 $(n\text{-}k)/n$이 된다.
  - ✓ 이때 $A(n|k)$를 크기 n의 입력에 대하여 k번째로 작은 아이템이 뿌리마디에 위치하고 있는 이진검색트리에 대한 평균검색 시간 이라고 하면,

2001-03-19 알고리즘 강의 슬라이드 8 15

$$A(n \mid k) = A(k-1)\frac{k-1}{n} + A(n-k)\frac{n-k}{n} + 1$$

그러면,

$$A(n) = \frac{1}{n}\sum_{k=1}^{n}\left[\frac{k-1}{n}A(k-1) + \frac{n-k}{n}A(n-k) + 1\right]$$

여기서 $C(n) = nA(n)$으로 놓으면,

$$\frac{C(n)}{n} = \frac{1}{n}\sum_{k=1}^{n}\left[\frac{k-1}{n}\frac{C(k-1)}{k-1} + \frac{n-k}{n}\frac{C(n-k)}{n-k} + 1\right]$$

$$C(n) = \sum_{k=1}^{n}\left[\frac{C(k-1)}{k} + \frac{C(n-k)}{n} + 1\right] \qquad C(1) = 1A(1) = 1$$

$$= \sum_{k=1}^{n}\frac{1}{n}[C(k-1) + C(n-k)] + n$$

- ✓ 이 재현식(recurrence)은 빠른 정렬의 평균의 경우의 시간복잡도와 거의 같다. 따라서 $C(n) \approx 1.38(n+1)\lg n$. 결과적으로 $A(n) \approx 1.38 \lg n$

- 그러나 최악의 경우의 시간복잡도는 $\Theta(n)$이므로 이 방법이 항상 효율적이라고는 보장할 수 없다.

2001-03-19 알고리즘 강의 슬라이드 8 16

검색시간 향상을 위한 트리구조

- 디스크에 접근하는 시간(외부검색:external search)은 RAM에 접근하는 시간(내부검색:internal search)보다 훨씬 느리기 때문에 실제로는 검색시간이 선형으로 나타난다 하더라도 외부검색의 경우에는 좋다고 받아들여 질 수 없다. 따라서 이런 경우 이진트리가 항상 균형을 이루게 함으로서 검색시간을 줄인다.
- **AVL 트리**: 아이템의 추가시간, 삭제시간이 모두 $\Theta(\lg n)$이고, 검색시간도 마찬가지 이다.
- B-트리 / 2-3 트리: 잎마디들의 깊이(수준)를 항상 같게 유지.

2001-03-19 알고리즘 강의 슬라이드 8 17

해쉉(Hashing)

- 1에서 100사이의 정수로 된 키가 있고, 100개의 레코드가 있다고 하자. 그러면 크기가 100인 배열 S를 만들어서 저장하면 빠른 시간 안에 검색할 수 있다. 그런데 만약 키가 주민등록번호라면 너무 많은 저장장소가 필요하게 된다.
- **해법**: 0..99의 첨자를 가진 크기가 100인 배열을 만든 후에, 키를 0..99 사이의 값을 가지도록 해쉬(hash)한다. 여기서 해쉬함수는 키를 배열의 첨자 값으로 변환하는 함수이다. 해쉬함수의 예: h(key) = key % 100
- 여기서 2개 이상의 키가 같은 해쉬값을 갖는 경우 충돌(collision)이 생긴다.
- 충돌 방지법: 오픈 해쉉(open hashing)
  같은 해쉬값을 갖는 키들을 바구니에 모아 놓는다. 주로 바구니는 연결된 리스트(linked list)로 구현 한다. 나중에 바구니를 검색할 때는 순차검색으로 한다.

2001-03-19 알고리즘 강의 슬라이드 8 18

해쉉(Hashing) [또]

- 만일 모든 키가 같은 해쉬값을 갖는 경우, 즉, 같은 바구니에 모두 모여 있는 경우에는 순차검색과 동일하다. 그러나 다행히도 그럴 확률은 거의 없다.

$$100\times(\frac{1}{100})^{100}=10^{-198}$$

- 해쉉이 효과를 얻기 위해서는 키가 바구니에 균일하게 분포하면 된다. 즉, 예를 들면, n개의 키와 m개의 바구니가 있을 때, 각 바구니에 평균적으로 n/m개의 키를 갖게 하면 된다.
- **정리**: n개의 키가 m개의 바구니에 균일하게 분포되어 있다면, 검색에 실패한 경우 비교 횟수는 n/m이다.

2001-03-19 알고리즘 강의 슬라이드 8 19

해쉉(Hashing) [또또]

- **정리**: n개의 키가 m개의 바구니에 균일하게 분포되어 있고, 각 키가 검색하게 될 확률이 모두 같다면, 검색에 성공한 경우 비교 횟수는 $\frac{n}{2m}+\frac{1}{2}$이다.
- **증명**: 각 바구니의 평균 검색시간은 $\frac{n}{m}$ 개의 키를 순차검색하는 평균시간과 같다. x개의 키를 순차검색하는데 걸리는 평균 검색시간은

$$1\times\frac{1}{x}+2\times\frac{1}{x}+\cdots+x\times\frac{1}{x}=\frac{1}{x}\sum_{i=1}^{x}i$$

$$=\frac{1}{x}\times\frac{x(x+1)}{2}$$

$$=\frac{x+1}{2}$$

따라서 $\frac{\frac{n}{m}+1}{2}=\frac{n}{2m}+\frac{1}{2}$

2001-03-19 알고리즘 강의 슬라이드 8 20

해싱(Hashing) [또또또]

- 보기: 키가 균일하게 분포되어 있고 $n = 2m$일 때
  - ✓ 검색 실패시 걸리는 시간 = $\frac{2m}{m} = 2$
  - ✓ 검색 성공시 걸리는 시간 = $\frac{2m}{2m} + \frac{1}{2} = \frac{3}{2}$