Chap 8. The Searching Problem

1. Lower Bounds for Searching by Comparisons of Keys
2. Interpolation Search
4. Hashing

Lower Bounds for Searching by Comparisons of Keys

- **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 시간복잡도 $W(n) = \lfloor \lg n \rfloor + 1$
  - 이보다 더 좋은(빠른) 알고리즘은 존재할까? 답은 "NO."
  - 정리: "키를 비교함으로만 검색을 수행하는 경우에는
    $\lfloor \lg n \rfloor + 1$ 이상의 비교가 필요하다." (다음 페이지에서 증명)
  - 따라서, 이분검색보다 더 빠른 알고리즘은 존재할 수 없다.

Lower Bounds for Searching by Comparisons of Keys

- 검색의 결정 트리 (Decision Tree)
  - 큰 마디(내부마디) – 키와 각 아이템을 비교하는 마디
  - 작은 마디(잎마디) - 검색결과
  - 뿌리마디에서 잎마디까지의 경로는 검색하면서 비교하는 과정을 보여준다.

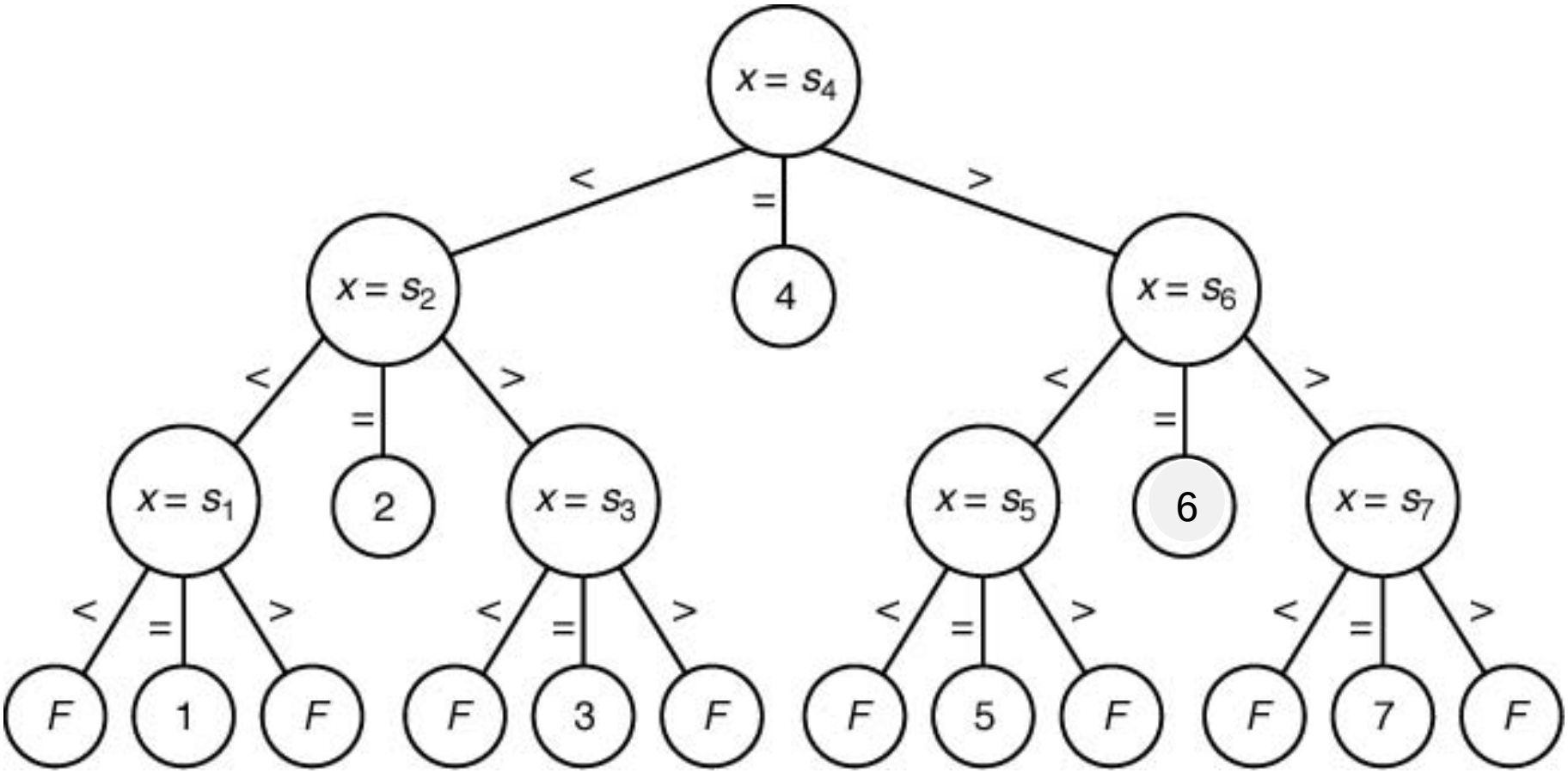
- 이진 검색의 결정 트리
  - 이진검색에서 최악의 경우 비교 횟수

  = (결정트리의 뿌리마디에서 잎마디까지 가장 긴 경로 상의 비교마디 수)

  = (노드의 개수가 n이 되는 이진트리의 깊이(depth) + 1)

  = $\lfloor \lg n \rfloor + 1$ (보조정리1)

Decision Tree of Binary Search



Lower Bounds for Worst-Case Behavior

- **보조정리 1**:
  n이 이진트리의 마디의 개수이고, d가 깊이라면, $d \geq \lfloor \lg n \rfloor$

  증명:

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

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

$$\lg n < d + 1$$

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

- **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 비교가 필요하다.

Interpolation Search

- 검색의 하한을 개선할 수 있는가?
  - 비교하는 이외에, 다른 추가적인 정보를 이용하여 검색할 수 있다면 가능

- Example
  - 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 실패
  - 그렇지 않으면 검색 키 x와 S[1+x div 10]과 비교

Interpolation Search

- 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.

  ➔ **interpolation search (보간검색)**

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

Interpolation Search

- Linear Interpolation (선형보간법)
  - 보기: S[1] = 4이고 S[10] = 97일 때 검색 키가 25이면, $mid = 3$.

$$mid = low + \left\lfloor \frac{x - S[low]}{S[high] - S[low]} \times (high - low) \right\rfloor = 1 + \left\lfloor \frac{25 - 4}{97 - 4} \times (10 - 1) \right\rfloor$$

  - 분석:
    - 아이템이 균등하게 분포되어 있고, 검색 키가 각 슬롯에 있을 확률이 같다고 가정
    - 선형보간검색의 평균적인 시간복잡도는 $A(n) \approx \lg(\lg n)$이 된다.
    - 예로서 n이 10억이라면, $\lg(\lg n)$은 약 5가 된다.

Interpolation Search

- Linear Interpolation (선형보간법)
  - 단점
    - 최악의 경우의 시간복잡도가 나쁘다.
    - 예로서 10개의 아이템이 1, 2, 3, 4, 5, 6, 7, 8, 9, 100 이고, 여기서 x=10을 찾으려고 한다면, *mid* 값은 항상 *low* 값이 되어서 모든 아이템과 비교를 해야 한다.
      따라서 최악의 경우 시간복잡도는 순차검색과 같다.

Interpolation Search

- Robust Interpolation Search (보강된 보간 검색법)
  - 항상 $mid - low$와 $high - mid$ 가 gap 값보다 크도록 다음과 같이 mid 값을 보정한다.

    1. 선형보간법에서 사용한 공식으로 mid값을 구한다.
    2. $gap = \lfloor (high - low + 1)^{1/2} \rfloor$
    3. 다음식으로 새로운 mid값을 구한다.

    $$mid = min(high - gap, max(mid, low + gap))$$

  - 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 $W(n) \approx (\lg n)^2$

B-Trees

- 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 마찬가지이다

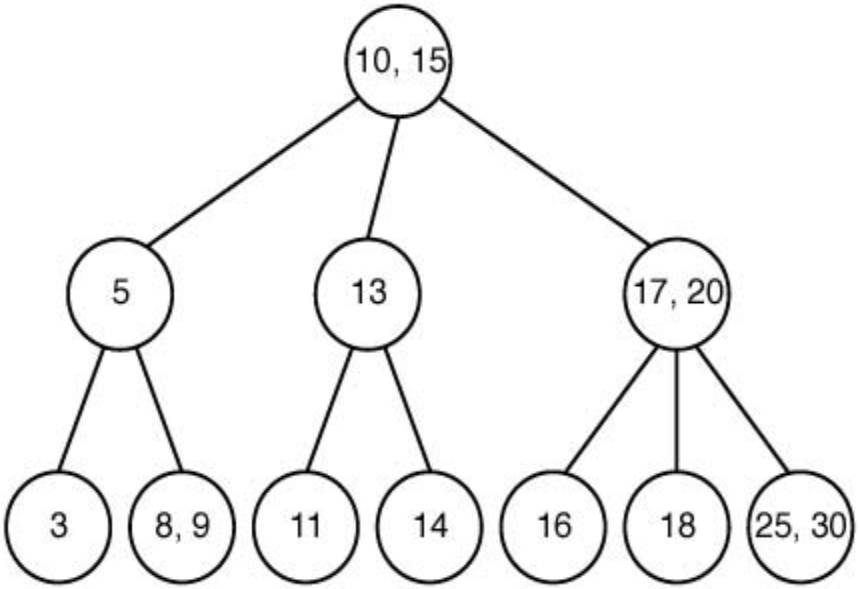
  - B-트리 / 3-2 트리
    - 잎마디들의 깊이(수준)를 항상 같게 유지

B-Trees

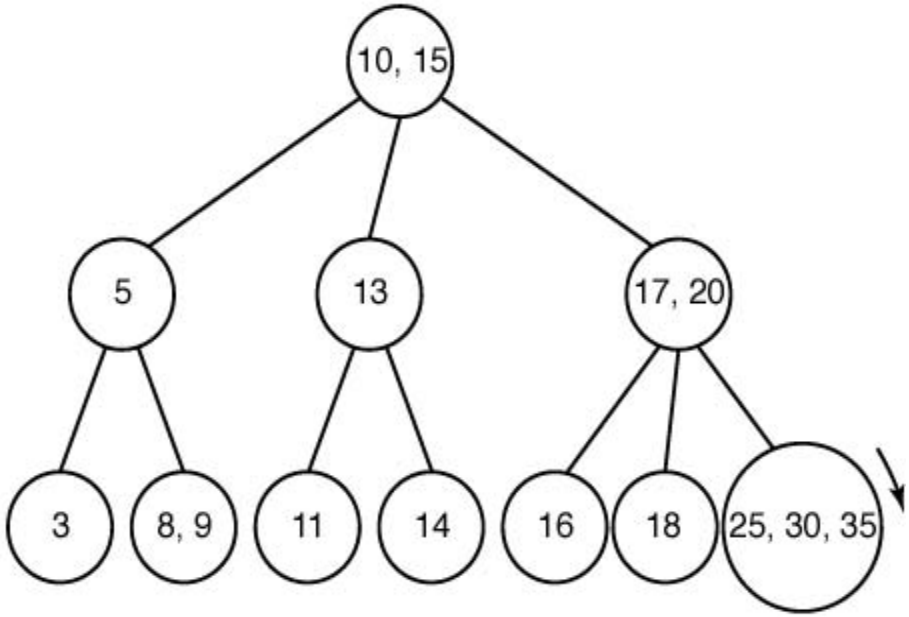
- 3-2 Tree
  - Properties
    - Each node contains 1 or 2 keys
    - If a nonleaf contains 1 key, it has 2 children, whereas if it contains 2 keys, it has 3 children
    - The keys in the left subtree of a given node are less than or equal to the key stored at that node
    - The keys in the right subtree of a given node are greater than or equal to the key stored at that node
    - If a node contains 2 keys, the keys in the middle subtree of the node are greater than or equal to the left key and less than or equal to the right key
    - All leaves are at the same level

B-Trees

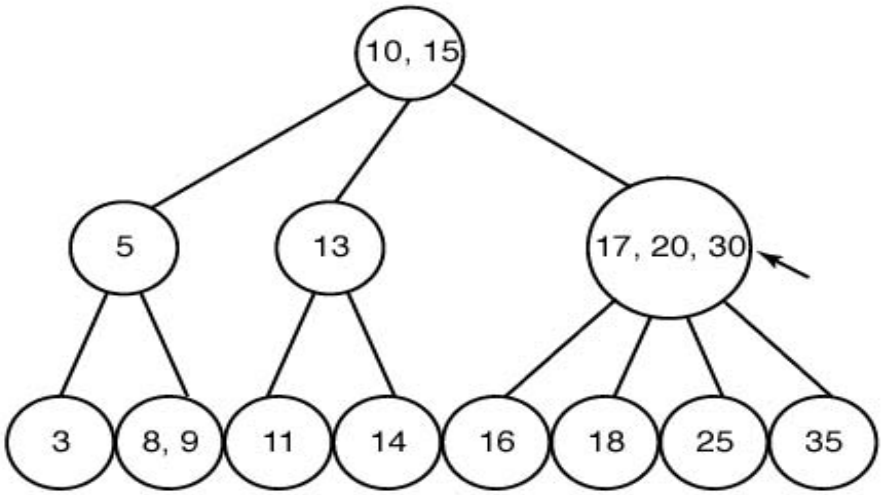
(a) A 3–2 tree



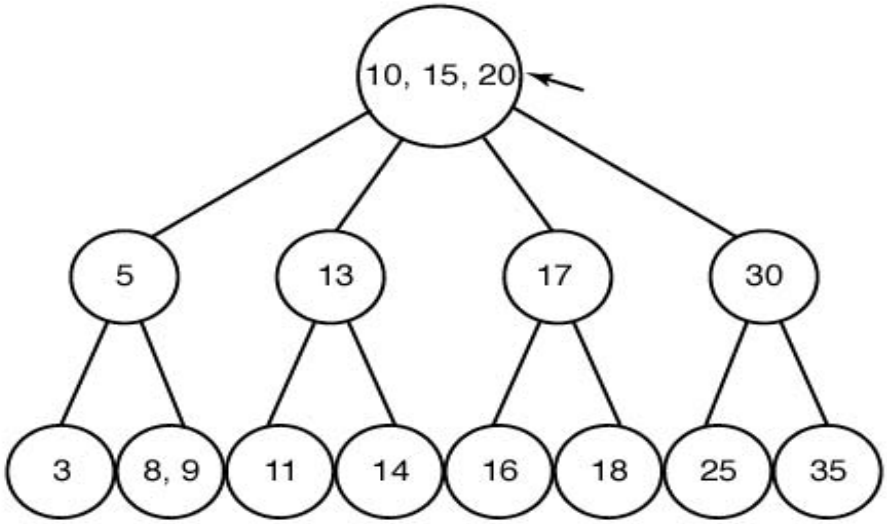
(b) 35 is added to the tree in sorted sequence in a leaf.



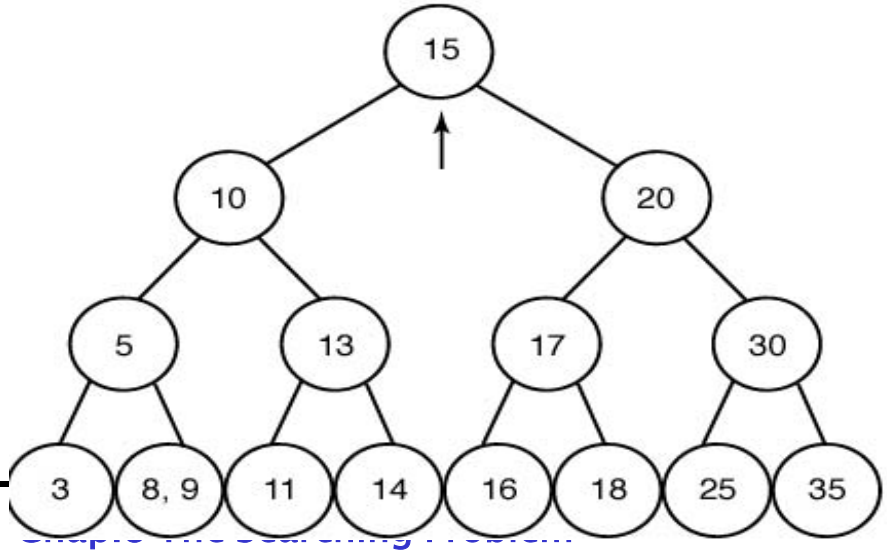
(c) If the leaf contains three keys, it breaks into two nodes and sends the middle key up to its parent.



(d) If the parent now contains three keys, the process of breaking open and sending the middle key up repeats.



(e) Finally, if the root contains three keys, it breaks open and sends the middle key up to a new root.



Hashing

- 키가 1에서 100사이의 정수이고, 100개의 레코드가 있다고 가정
  - 효율적인 방법은 100개의 아이템을 저장할 수 있는 배열 S를 만들어서 저장
    - 키가 *i*인 레코드는 *S[i]*에 저장
  - 만약 키가 13자리인 주민등록번호라면 너무 많은 저장장소가 필요하게 된다.
    - 100 billion items to store only 100 records

- 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 해싱(open hashing)
    - 같은 해쉬값을 갖는 키들을 바구니에 모아 놓는다.
    - 주로 바구니는 연결된 리스트(linked list)로 구현한다.
    - 나중에 바구니를 검색할 때는 순차 검색으로 한다.

Hashing

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

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

- 해쉬이 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.

Hashing

- **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 순차 검색하는 평균시간과 같다.
  - x개의 키를 순차 검색하는데 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 $\frac{\frac{n}{m}+1}{2}=\frac{n}{2m}+\frac{1}{2}$

Hashing

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