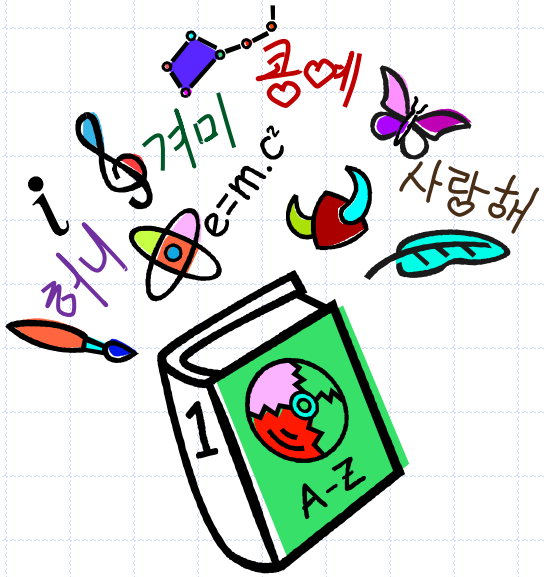
사전

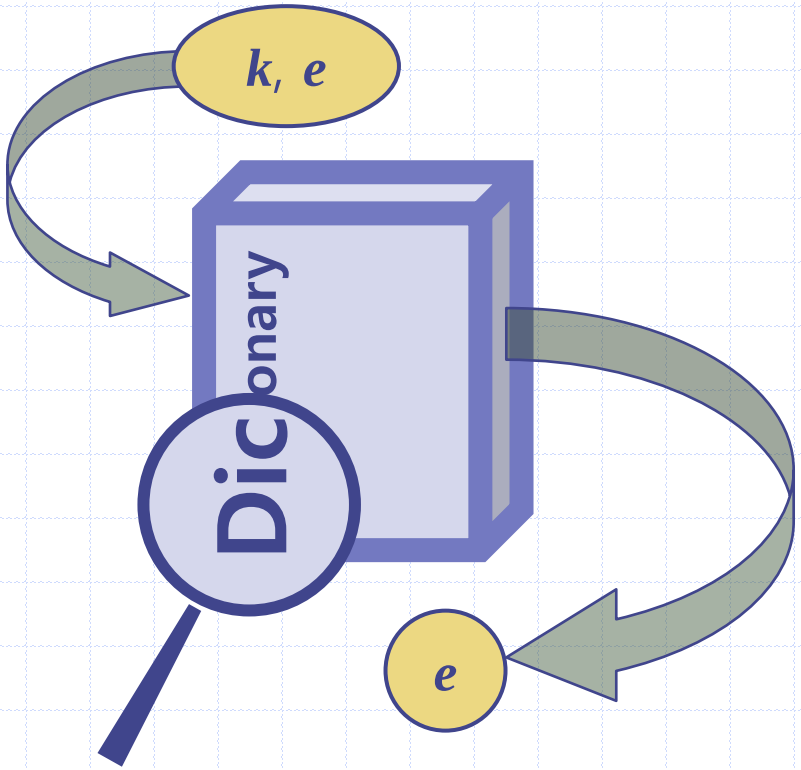


Outline

- 10.1 사전 ADT
- 10.2 사전 ADT 메쏘드
- 10.3 사전 ADT 구현
- 10.4 응용문제

사전 ADT

- 사전 ADT는 탐색 가능한 형태의 (**키**, **원소**) 쌍 항목들의 모음을 모델링
- 사전에 관한 주요 작업
  - **탐색**(searching)
  - **삽입**(inserting)
  - **삭제**(deleting)
- 두 종류의 사전
  - 무순사전 ADT
  - 순서사전 ADT

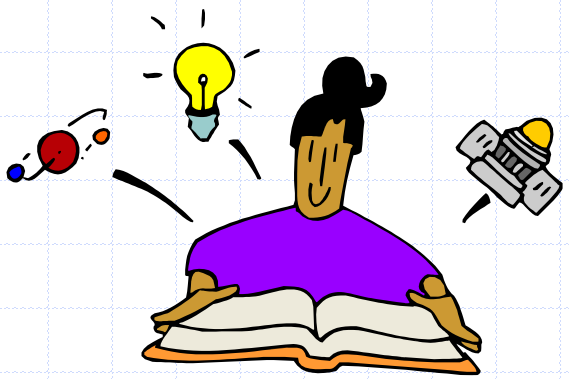


사전 ADT 메쏘드

- 일반 메쏘드
  - integer size(): 사전의 항목 수를 반환
  - boolean isEmpty(): 사전이 비어 있는지 여부를 반환
- 접근 메쏘드
  - element findElement(k): 사전에 키 k를 가진 항목이 존재하면 해당 원소를 반환, 그렇지 않으면 특별 원소 $NoSuchKey$를 반환
- 갱신 메쏘드
  - insertItem(k, e): 사전에 (k, e) 항목을 삽입
  - element removeElement(k): 사전에 키 k를 가진 항목이 존재하면 해당 항목을 삭제하고 원소를 반환, 그렇지 않으면 특별 원소 $NoSuchKey$를 반환

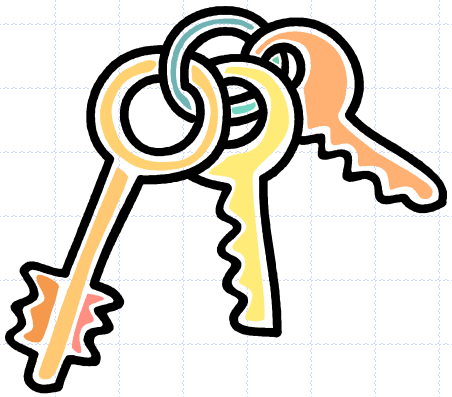
사전 응용

- 직접 응용
  - 연락처 목록
  - 신용카드 사용승인
  - 인터넷주소 매핑
    - 호스트명(**예:** www.sejong.ac.kr)을 인터넷 주소(**예:** 128.148.34.101)로 매핑
- 간접 응용
  - 알고리즘 수행을 위한 보조 데이터구조
  - 다른 데이터구조를 구성하는 요소



탐색

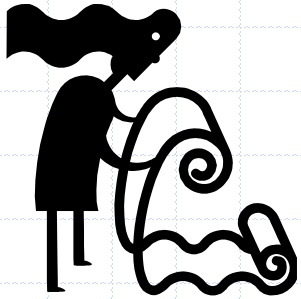
- **비공식적으로**, **탐색**(search)은 데이터 집단으로부터 특정한 정보를 추출함을 말한다
- **공식적으로**, **탐색**은 사전으로 구현된 데이터 집단으로부터 지정된 키(key)와 연관된 데이터 원소를 반환함을 말한다
  - 상이한 전제
    - **유일 키:** 한 개의 키에 대해 하나의 데이터 항목만 존재
      - **예:** 학번, 은행계좌, login ID
    - **중복 키:** 한 개의 키에 대해 여러 개의 데이터 항목 존재
      - **예:** 이름, 나이, 계좌개설일자



사전 구현에 따른 탐색기법

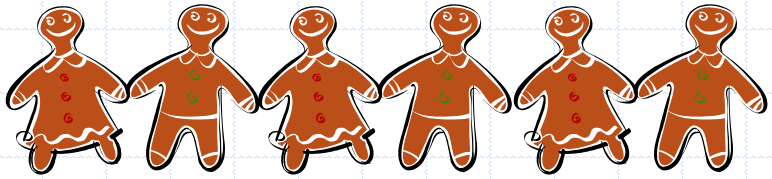
구현 형태	구현 종류	예	주요 탐색 기법
리스트	무순사전 ADT	기록파일	선형탐색
	순서사전 ADT	일람표	이진탐색
트리	탐색트리	이진탐색트리, AVL 트리, 스플레이 트리	트리탐색
해시테이블			해싱

무순사전 ADT: 기록파일



- **기록파일**(log file): **무순리스트**를 사용하여 구현된 사전
  - 사전 항목들을 임의의 순서로 리스트에 저장(이중연결리스트 또는 원형배열을 이용)
- 성능
  - insertItem: 새로운 항목을 기존 리스트의 맨 앞 또는 맨 뒤에 삽입하면 되므로 $\mathbf{O}(1)$ 시간 소요
  - findElement 및 removeElement: 최악의 경우(즉, 항목이 존재하지 않을 경우), 주어진 키를 갖는 항목을 찾기 위해 리스트 전체를 순회해야 하므로 $\mathbf{O}(n)$ 시간 소요
- 기록파일의 사용이 효과적인 경우
  - 소규모의 사전, 또는
  - 삽입은 빈번하지만 탐색이나 삭제는 드문 사전(**예:** 서버의 로그인 기록)

선형탐색



- findElement 작업은 사전에 대해 지정된 키 ***k***에 관한 **선형탐색**(linear search)을 수행하여 ***k***를 가진 원소를 반환

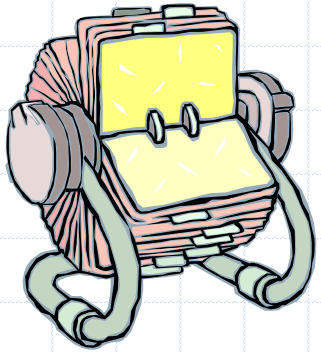
```
Alg findElement(k)                    {generic}
    input list L, key k
    output element with key k

1. L.initialize(i)
2. while (L.isValid(i))
       if (L.key(i) = k)
           return L.element(i)
       else
           L.advance(i)
3. return NoSuchKey
```

선형탐색 분석

- 시간
  - 입력크기(즉, 데이터항목의 수)를 n이라 하면 최악의 경우는 찾고자 하는 키가 맨 뒤에 있거나 아예 없는 경우다
  - 따라서 $O(n)$ 시간에 수행
- 공간
  - 입력 데이터구조에 대해 읽기 작업만 수행하므로 $O(1)$ 공간으로 수행

순서사전 ADT: 일람표



- **일람표**(lookup table): 순서리스트를 사용하여 구현된 사전
  - 사전 항목들을 배열에 기초한 리스트에 키로 정렬된 순서로 저장
- 성능
  - findElement: **이진탐색**을 사용하면 $O(\log n)$ 시간 소요
  - insertItem: 새로운 항목을 삽입하기 위한 공간 확보를 위해 최악의 경우 n개의 기존 항목들을 이동해야 하므로 $O(n)$ 시간 소요
  - removeElement: 항목이 삭제된 공간을 기존 항목들로 메꾸기 위해 최악의 경우 n개의 기존 항목들을 이동해야 하므로 $O(n)$ 시간 소요
- 일람표 사용이 효과적인 경우
  - 소규모 사전, 또는
  - 탐색은 빈번하지만 삽입이나 삭제는 드문 사전(**예:** 신용카드 사용승인, 전화번호부)

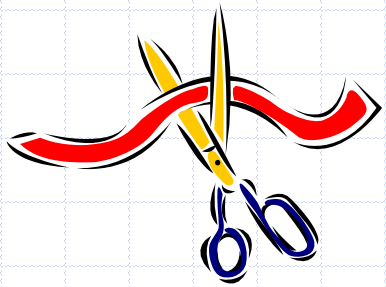
선형탐색

- 실패가 예정된 선형탐색의 경우, 평균적으로 입력크기의 절반 정도만 탐색하고 정지(elseif 절 참조)
- **O(*n*)** 시간 소요

```
Alg findElement(k)                    {generic}
    input list L, key k
    output element with key k

1. L.initialize(i)
2. while (L.isValid(i))
        if (L.key(i) = k)
            return L.element(i)
        elseif (L.key(i) > k)
            return NoSuchKey
        else
            L.advance(i)
3. return NoSuchKey
```

이진탐색



- **이진탐색**(binary search): 키로 정렬된 배열에 기초한 리스트로 구현된 사전에 대해 findElement 작업을 수행
- 재귀할 때마다 후보 항목들의 수가 반감
- 입력크기의 로그 수에 해당하는 수의 재귀를 수행한 후 정지
- **참고:** 스무고개

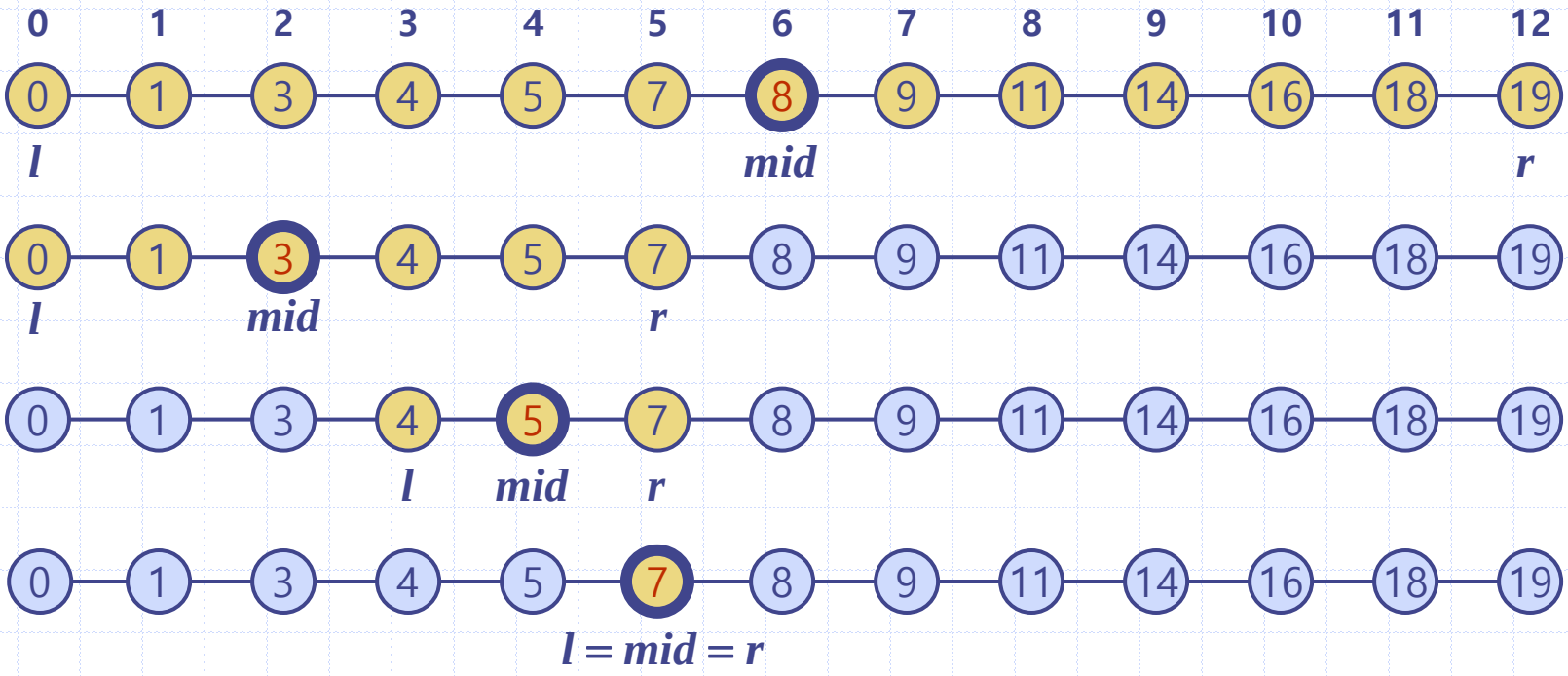
Alg ***findElement(k)*** {driver}
  input sorted array $A[0..n-1]$, key k
  output element with key k

1. **return** $rFE(k, 0, n-1)$

Alg ***rFE(k, l, r)*** {recursive}
1. **if** $(l > r)$
   return *NoSuchKey*
2. $mid \leftarrow (l + r)/2$
3. **if** $(k = key(A[mid]))$
   return $element(A[mid])$
  elseif $(k < key(A[mid]))$
   return $rFE(k, l, mid - 1)$
  else $\{k > key(A[mid])\}$
   return $rFE(k, mid + 1, r)$

이진탐색 수행 예

- findElement(7)



이진탐색 분석

- 입력 순서리스트가 **배열**로 구현된 경우
  - 총 비교회수는 최악의 경우 $\mathbf{O}(\log n)$
  - 따라서, $\mathbf{O}(\log n)$ 시간에 수행
- 입력 순서리스트가 **연결리스트**로 구현된 경우
  - 가운데 위치로 접근하는 데만 $\mathbf{O}(n)$ 시간 소요되므로 전체적으로 $\mathbf{O}(n)$ 시간에 수행
- **이진탐색의 힘**을 보여주는 예: **스무고개**
  - 2^{20}(약 100만)개의 후보 범위에서 출발하더라도 20회의 이등분이 가능한 질문을 통해 1개 후보(즉, 답)로 압축 가능
- **분할통치** vs. **이진탐색**
  - **분할통치:** 이등분된 두 개의 범위 **양쪽**을 모두 고려
  - **이진탐색:** 이등분된 두 개의 범위 중 **한쪽**만 고려