Data Structure

(Java programming)

Chapter 10.

Contents

- Dictionary
- Binary Search
- Homework3

- ***Dictionary***

• Dictionary

Dictionary

- Like a map, a ***dictionary*** models a searchable collection of (*key*, *value*) entries
- Unlike a map, multiple entries with the same key are allowed
  - Cf., **`put`** of map vs. **`insert`** of dictionary
- The main operations of a dictionary are also searching, inserting and deleting entries
- Applications
  - English dictionary (same words)
  - Phone book (same names)
  - Etc.

• Dictionary

Dictionary Queue ADT

- `entry` **`find`**`(`***k***`)`
  - If the dictionary has an entry with key ***k***, returns it, else, returns null
- `list` **`findAll`**`(`***k***`)`
  - Returns a collection (list) of all entries with key ***k***
- `entry` **`insert`**`(`***k***`,` ***v***`)`
  - Inserts and returns the entry (***k***, ***v***)
  - Just inserts, does **NOT** replace the old entry
- `entry` **`remove`**`(`***e***`)`
  - Removes and returns the entry ***e*** from the dictionary, if fails, returns null
  - Cf., `entry` **`remove`**`(`***k***`)`: find an entry with key ***k*** and remove it
- `list` **`entries`**`()`
  - Returns a list of all the entries in the dictionary
- Common operations: **`size`**`()`, **`isEmpty`**`()`

• Dictionary

Dictionary Example

Operation	*Output*	*Dictionary*
insert(5,*A*)	(5,*A*)	(5,*A*)
insert(7,*B*)	(7,*B*)	(5,*A*), (7,*B*)
insert(2,*C*)	(2,*C*)	(5,*A*), (7,*B*), (2,*C*)
insert(8,*D*)	(8,*D*)	(5,*A*), (7,*B*), (2,*C*), (8,*D*)
insert(2,*E*)	(2,*E*)	(5,*A*), (7,*B*), (2,*C*), (8,*D*), (2,*E*)
find(7)	(7,*B*)	(5,*A*), (**7**,*B*), (2,*C*), (8,*D*), (2,*E*)
find(4)	**null**	(5,*A*), (7,*B*), (2,*C*), (8,*D*), (2,*E*)
find(2)	(2,*C*)	(5,*A*), (7,*B*), (**2**,*C*), (8,*D*), (2,*E*)
findAll(2)	{(2,*C*), (2,*E*)}	(5,*A*), (7,*B*), (**2**,*C*), (8,*D*), (**2**,*E*)
size()	5	(5,*A*), (7,*B*), (2,*C*), (8,*D*), (2,*E*)
remove(find(5))	(5,*A*)	(7,*B*), (2,*C*), (8,*D*), (2,*E*)
find(5)	**null**	(7,*B*), (2,*C*), (8,*D*), (2,*E*)

• Dictionary

Dictionary Operations

Algorithm findAll(k):
Input: A key k
Output: An iterable collection of entries with key equal to k
Create an initially-empty list L
for each entry e in D.entries() **do**
if e.getKey() $= k$ **then**
L.addLast(e)
return L {the elements in L are the selected entries}

• Dictionary

Dictionary Operations

Algorithm insert(k, v):
  Input: A key k and value v
  Output: The entry (k, v) added to D
  Create a new entry $e = (k, v)$
  S.addLast(e) {S is unordered}
  return e

Algorithm remove(e):
  Input: An entry e
  Output: The removed entry e
  {We don't assume here that e stores its location in S}
  for each node p in S.nodes() **do**
    if p.element() $= e$ **then**
      S.remove(p)
      return e
  return null {there is no entry e in D}

- ***Binary Search***

• Binary Search

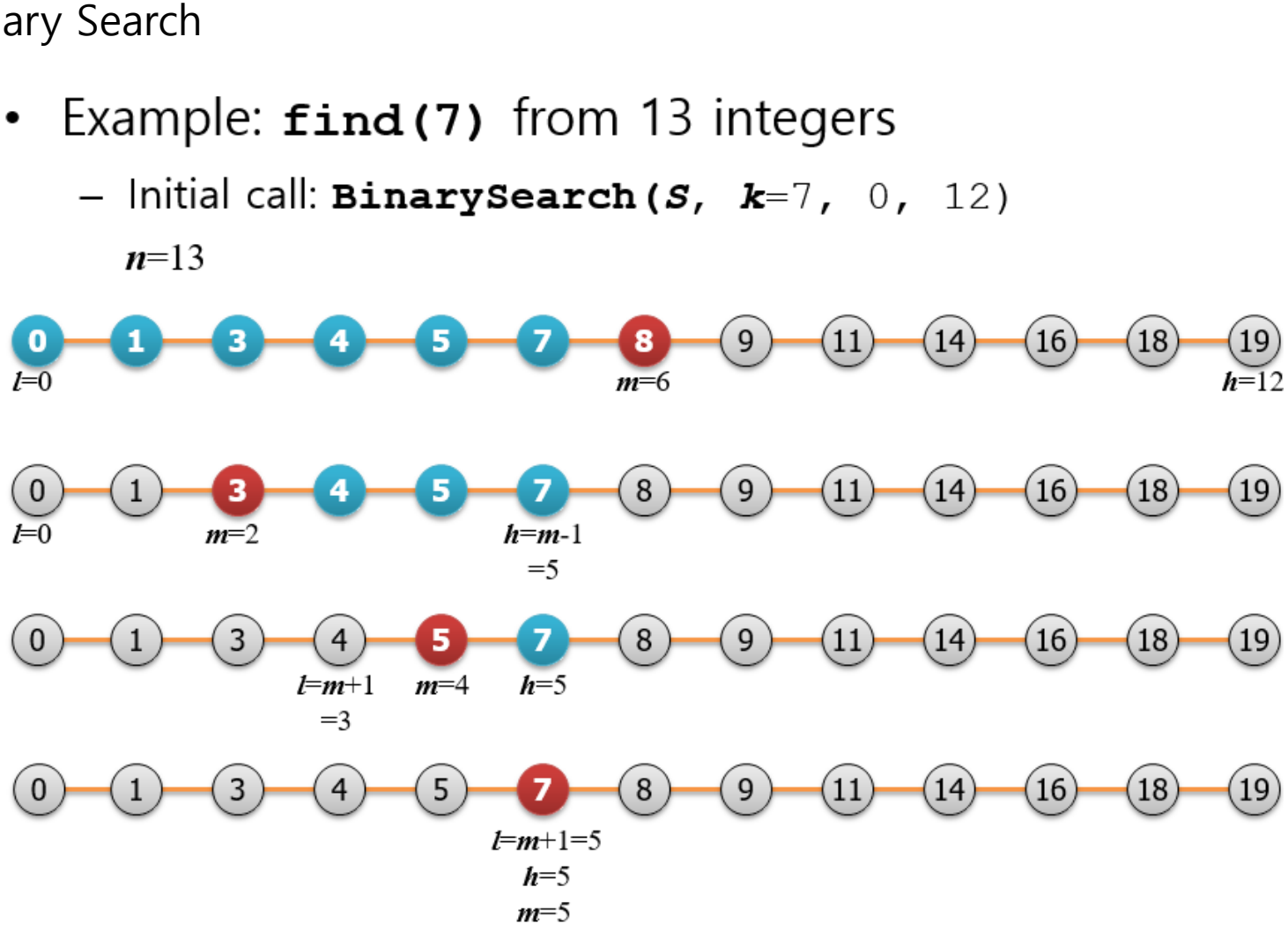
Ordered Search Table

- An ***ordered search table*** (***search table*** for short) is a dictionary implemented with a sorted array list
  - Stores the entries of the dictionary in an array list sorted by the keys
  - We can use an external comparator for arbitrary object keys
- Performance of ordered search tables
  - `insert` and `remove` take $O(n)$ time
    - With linear search, we need to probe n entries to find the location of the entry to be inserted or removed
    - In addition, we have to shift n entries at maximum to make room for the new entry or compact the array list after the removal
  - Whereas, we can perform `find` in $O(\log n)$ time using ***binary search***
- A search table is effective for
  - Dictionaries on which searches are the most common operations, while insertions and removals are rarely performed
  - E.g., Credit card authorizations
    - Once a person registers the credit card information, it is retrieved every time he uses the credit card

• Binary Search

Binary Search

- Example: **`find(7)`** from 13 integers
  - Initial call: **`BinarySearch(S, k=7, 0, 12)`**



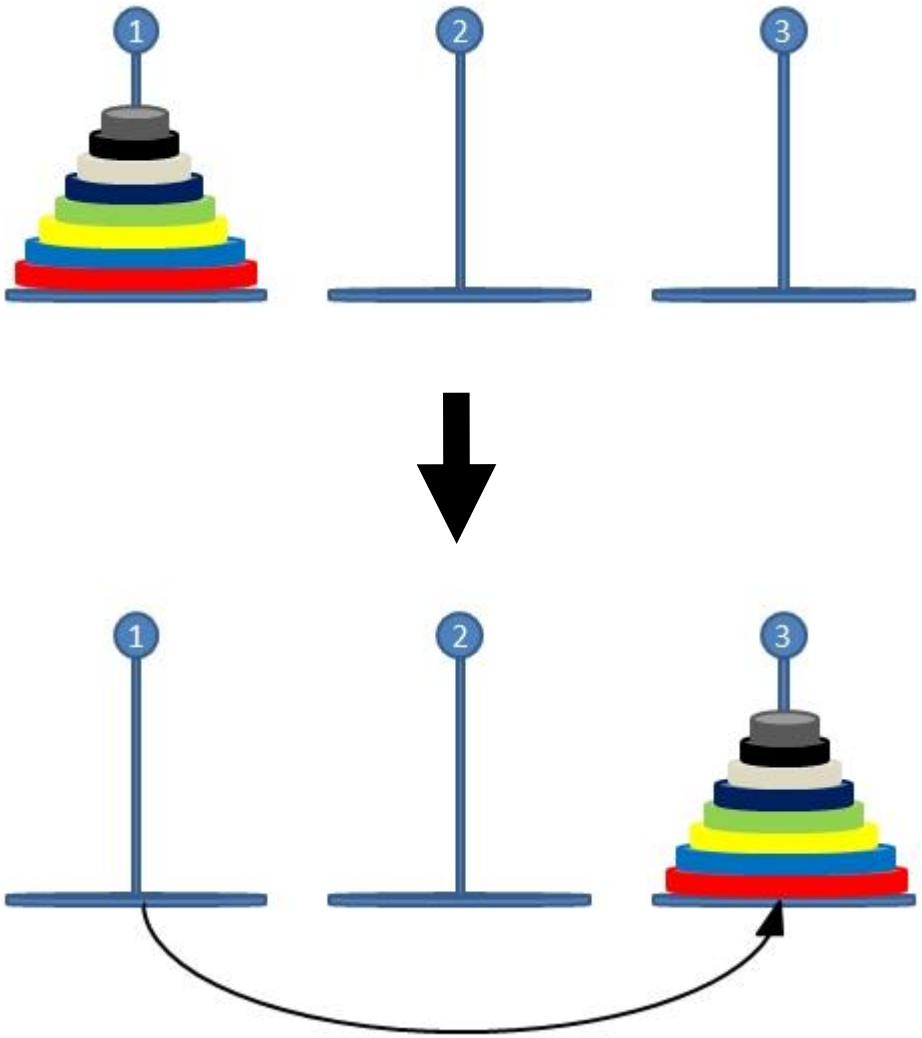
• Binary Search

Binary Search Algorithm

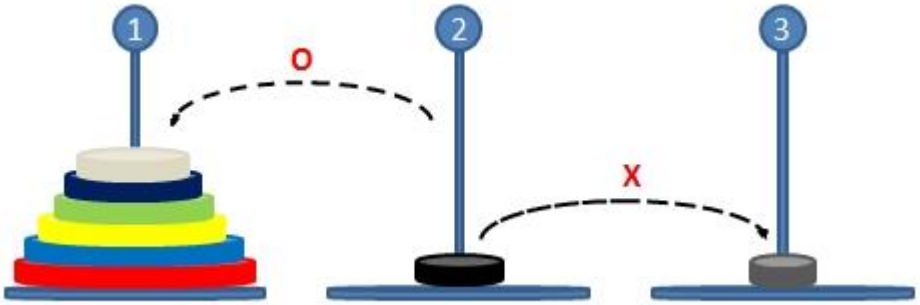
```
Algorithm BinarySearch(S, k, low, high):
    Input: An ordered array list S storing n entries and integers low and high
    Output: An entry of S with key equal to k and index between low and high, if
      such an entry exists, and otherwise null
  if low > high then
    return null
  else
    mid ← ⌊(low + high)/2⌋
    e ← S.get(mid)
    if k = e.getKey() then
      return e
    else if k < e.getKey() then
      return BinarySearch(S, k, low, mid − 1)
    else
      return BinarySearch(S, k, mid + 1, high)
```

- ***Homework3 – Tower of Hanoi with Stack***

- # Tower of Hanoi with Stack



- # Tower of Hanoi with Stack

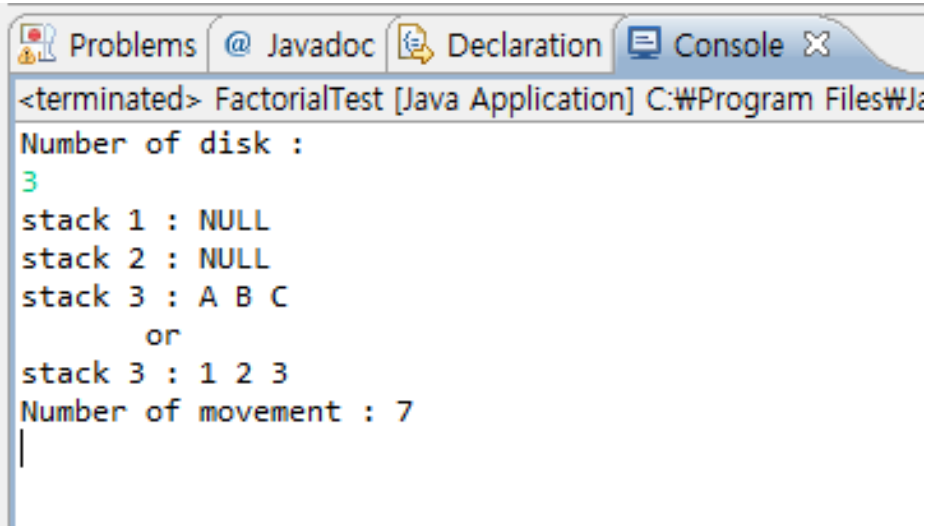


Tower of Hanoi with Stack

- Stack을 이용하여 하노이 탑을 구현하시오
- 3개의 기둥과 각각 다른 크기의 여러 개의 원반으로 구성되어있으며, 원반은 인접해있는 기둥 사이에서만 옮길 수 있다.
- 한 기둥 안에서는 가장 큰 원반이 최하단에, 가장 작은 원반이 최상단에 위치해야 하며, 크기 순서로 쌓여야 한다.
- 처음 시작은 모든 원반이 가장 왼쪽 기둥에 쌓여있으며, 이를 모두 위의 규칙을 지키면서 가장 오른쪽 기둥으로 옮겨야 한다.
- 한 번에 하나의 원반만 옮길 수 있으며, 가장 위에 있는 원반만 꺼낼 수 있다.

• Tower of Hanoi with Stack

- Input : number of disk.
- Output : condition of three stacks, and number of movement



제출기한 : 2017. 05. 31 자정(24:00)

• Submission

e-mail : 2017kudatastructure@gmail.com

Attach methods :

1. Create zip file (java project folder)
2. Modify the file names :
   ID_name_datastructure[Classcode].dat
   ex) 201173378_이명재_datastructure_homework3[2403-1].dat
3. e-mail title :
   datastructure_name_chapter
   ex) datastructure_최수용_homework3[2404-1]

Classcode

2403-1 **[A-1반]**	2403-2 **[A-2반]**	
2404-1 **[B-1반]**	2404-2 **[B-2반]**	2405 **[C반]**