

Set08: Dictionary Abstract Data Type: Introduction and Unordered ADTs

CS240: Data Structures and Data Management

Jérémy Barbay

Outline

Dictionary ADTs (introduction)

Dictionary ADT
Notes on Computation Model
Specific Dictionary ADTs

Unordered Dictionary ADT

Unordered List DS
MTF Heuristic
Transpose Heuristic
Final Thoughts

Dictionary ADT

- ▶ Container of key-element pairs
- ▶ Required operations:
 - ▶ `insert(k, e)`,
 - ▶ `remove(k)`,
 - ▶ `find(k)`,
 - ▶ `isEmpty()`
- ▶ May also support (when an **order** is provided):
 - ▶ `closestKeyBefore(k)`,
 - ▶ `closestElemAfter(k)`

Note: **No duplicate keys**

Set ADT: a simplified dictionary

- ▶ Container of **distinct** objects (keys)
- ▶ Required operations:
 - ▶ `insert(k)`, `remove(k)`,
 - ▶ `contains(k)`, `isEmpty()`
- ▶ Often support:
 - ▶ **union** $(X \cup Y)$, **intersection** $(X \cap Y)$,
 - ▶ **difference** $(X - Y)$, **subset** $(X \subseteq Y)$

Set ADT: Example

$X = \{1, 2, 3, 4\}$ and $Y = \{2, 4, 6\}$

- ▶ $X.\text{insert}(2) \Rightarrow$
- ▶ $X \cup Y =$
- ▶ $X \cap Y =$
- ▶ $X - Y =$
- ▶ $X \subseteq Y =$

Notes on Computation Model

- ▶ Dictionaries and Sets are implemented (almost) identically: Often we draw and discuss the Set scenario.
- ▶ Focus primarily on `find` operator
 - ▶ Usually the most common operator.
 - ▶ Insertion and removal usually start with a `find`.
 - ▶ Advanced implementations address the other operations.

Specific Dictionary ADTs and their DS

Outline

Dictionary ADTs (introduction)

Dictionary ADT

Notes on Computation Model

Specific Dictionary ADTs

Unordered Dictionary ADT

Unordered List DS

MTF Heuristic

Transpose Heuristic

Final Thoughts

Unordered Dictionary ADT

- ▶ Container of key-element pairs
which can be compared only for equality.
- ▶ Require general Dictionary operations:

Note: No duplicate keys, which can be costly!

Unordered List DS

- ▶ A chained list of n items $D = \{x_1, \dots, x_n\}$.
- ▶ **Insert**
 - ▶ Constant time insertion possible
- ▶ **Find**(x)

What about the complexity of **Find**(x) on average when $x \in D$?

Average-Case

- ▶ Assume uniform distribution of key requests
- ▶ Assume all searches successful
- ▶ Key K_i is requested with probability $p_i = \frac{1}{n}$
- ▶ Expected number of comparisons:

- ▶ Keys may not always be accessed uniformly

Optimal Order

- ▶ What is the best arrangement?
- ▶ C_{OPT} is the expected value of this perfect arrangement

Examples:

- ▶ if the distribution is uniform, $C_{OPT} =$
- ▶ if the distribution is uneven,
for instance $p_i = 2^{-i}$ and $p_n = 2^{-(n-1)}$
(Note that $\sum p_i = 1$ as $p_{n-1} = p_n = 2^{-(n-1)}$)

C_{OPT}

Self-Organizing Lists DS

- ▶ Why can we not generally use the optimal ordering?
 ⇒ approximate the ordering, using heuristics to get “good” results
- ▶ After every access we possibly rearrange a piece of the list, to possibly tend to a good average performance.

Ideas:

Move-To-Front Heuristic

- ▶ Access the key in position i
- ▶ **Heuristic:**
- ▶ **Example:**

Initial :	1	2	3	4	5	6	
Find(3)							
Find(4)							
Find(6)							
Find(4)							
							Cost

Cost of Move-To-Front

How does MTF compare to the optimal ordering?

Theorem

Assume that:

- ▶ the keys k_1, \dots, k_n have probabilities $p_1 \geq p_2 \geq \dots \geq p_n \geq 0$
- ▶ the list is used sufficiently to reach a steady state.

Then:

$$C_{MTF} < 2 \cdot C_{OPT}$$

Cost of Move-To-Front (Proof)

$$\begin{aligned}
 C_{OPT} &= \sum_{j=1}^n j p_j \\
 C_{MTF} &= \sum_{j=1}^n p_j (\text{cost of finding } k_j) \\
 &= \sum_{j=1}^n p_j (1 + \text{number of keys before } k_j)
 \end{aligned}$$

To compute the average number of keys before k_j :

$$\begin{aligned}
 \Pr[k_i \text{ before } k_j] &= \frac{p_i}{p_i + p_j} \\
 E(\text{number of keys before } k_j) &= \sum_{i \neq j} \frac{p_i}{p_i + p_j}
 \end{aligned}$$

Cost of Move-To-Front (Proof end)

Therefore,

$$\begin{aligned}
 C_{MTF} &= \sum_{j=1}^n p_j \left(1 + \sum_{i \neq j} \frac{p_i}{p_i + p_j} \right) && \text{(Joining both previous formulas.)} \\
 &= 1 + 2 \sum_{j=1}^n p_j \sum_{i < j} \frac{p_i}{p_i + p_j} && \text{(By reordering the terms.)} \\
 &\leq 1 + 2 \sum_{j=1}^n p_j \left(\sum_{i < j} 1 \right) && \text{(Because } \frac{p_i}{p_i + p_j} \leq 1. \text{)} \\
 &= 1 + 2 \sum_{j=1}^n p_j (j - 1) \\
 &= 1 + 2C_{OPT} + 2 \sum_{j=1}^n (-p_j) \\
 &= 2C_{OPT} - 1. && \text{(Because } \sum_{j=1}^n (p_j) = 1. \text{)}
 \end{aligned}$$

Transpose Heuristic

► Access the key in position i

► **Heuristic:**

► **Example:**

Initial :	1	2	3	4	5	6	
Find(3)							
Find(4)							
Find(6)							
Find(4)							
							Cost

Observations

► **Move-To-Front**

► **Transpose**

Final Thoughts

► How bad can each heuristic perform?

► Assume initial arrangement of:

A	B	C	...	Y	Z
---	---	---	-----	---	---

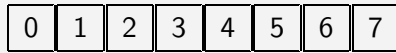
► What are the worse sequence of Find requests for:

► **Move-To-Front:**

► **Transpose:**

Exercises

- ▶ Assume initial arrangement of:



- ▶ Access in the order:

5 3 5 6 4 6 5 0 3 5 6 4

- ▶ Final arrangement for **Move-To-Front**:



- ▶ Total Comparisons:

- ▶ Final arrangement for **Transpose**:



- ▶ Total Comparisons:

Summary Unordered Dictionaries

- ▶ Without order, stuck to $O(n)$ in the worst case.
- ▶ Take advantage of **non-uniform distributions** to perform better.
- ▶ Some Heuristics perform close to optimal **without knowing the distribution**.

References:

- ▶ Goodrich and Tamassia: pp. 114-115, 28-30
- ▶ Cormen, Leiserson, Rivest, Stein: Not covered.