https://algs4.cs.princeton.edu

3.1 SYMBOL TABLES

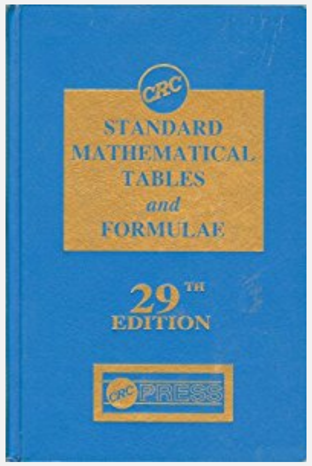
- *API*
- *elementary implementations*
- *ordered operations*

https://algs4.cs.princeton.edu

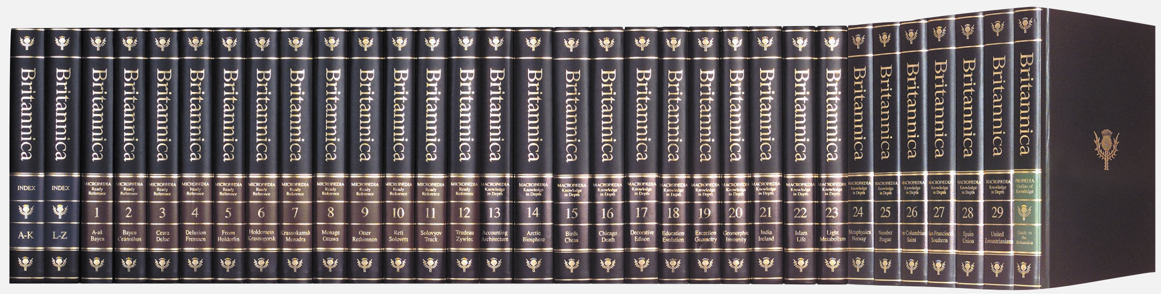
3.1 Symbol Tables

- *API*
- *elementary implementations*
- *ordered operations*

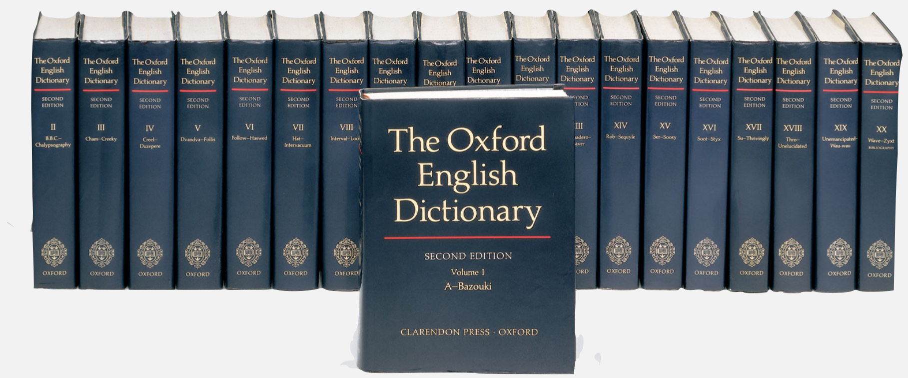
Printed symbol tables



key = math function and input
value = function output

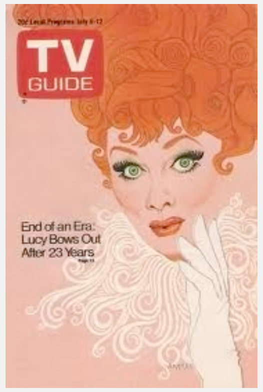


key = term
value = article



key = word
value = definition

key = name
value = phone number



key = time and channel
value = TV show

Unsupported operations.

- Add a new name and associated number.
- Remove a given name and associated number.
- Change the number associated with a given name.

Symbol tables

Key–value pair abstraction.

- Insert a value with specified key.
- Given a key, search for the corresponding value.

Ex. DNS lookup.

- Insert domain name with specified IP address.
- Given domain name, find corresponding IP address.

domain name	IP address
www.cs.princeton.edu	128.112.136.11
www.princeton.edu	128.112.128.15
www.yale.edu	130.132.143.21
www.harvard.edu	128.103.060.55

↑ key ↑ value

Symbol table applications

application	purpose of search	key	value
dictionary	find definition	word	definition
book index	find relevant pages	term	list of page numbers
file share	find song to download	name of song	computer ID
financial account	process transactions	account number	transaction details
web search	find relevant web pages	keyword	list of page names
compiler	find properties of variables	variable name	type and value
routing table	route Internet packets	destination	best route
DNS	find IP address	domain name	IP address
reverse DNS	find domain name	IP address	domain name
genomics	find markers	DNA string	known positions
file system	find file on disk	filename	location on disk

Symbol tables: context

Also known as: maps, dictionaries, associative arrays.

Generalizes arrays. Keys need not be integers between 0 and $n - 1$.

Language support.

- External libraries: C, VisualBasic, Standard ML, bash, ...
- Built-in libraries: Java, C#, C++, Scala, ...
- Built-in to language: Awk, Perl, PHP, Tcl, JavaScript, Python, Ruby, Lua.

every array is an associative array (PHP)

every object is an associative array (JavaScript)

table is the only "primitive" data structure (Lua)

```
has_nice_syntax_for_associative_arrays["Python"] = True
has_nice_syntax_for_associative_arrays["Java"]   = False
```

legal Python code

Basic symbol table API

Associative array abstraction. Associate key–value pairs.

public class ST<Key extends Comparable<Key>, Value>			
	ST()	*create an empty symbol table*	
void	put(Key key, Value val)	*insert key–value pair*	← **a[key] = val;**
Value	get(Key key)	*value paired with key*	← **a[key]**
boolean	contains(Key key)	*is there a value paired with key?*	
Iterable<Key>	keys()	*all the keys in the symbol table*	
void	delete(Key key)	*remove key (and associated value)*	
boolean	isEmpty()	*is the symbol table empty?*	
int	size()	*number of key–value pairs*	

Conventions

- Method `get()` returns `null` if key not present.
- Method `put()` overwrites old value with new value.
- Values are not `null`. ← `java.util.Map` allows `null` values

Key and value types

Value type. Any generic type.

Key type: different assumptions.

- This lecture: keys are `Comparable`, use `compareTo()`.
- Hashing lecture: keys are any generic type, use `equals()` to test equality and use `hashCode()` to scramble key.

Best practices. Use immutable types for symbol-table keys.

- Immutable in Java: `String`, `Integer`, `Double`, `Color`, ...
- Mutable in Java: `StringBuilder`, `Stack`, `URL`, arrays, ...

https://algs4.cs.princeton.edu

3.1 Symbol Tables

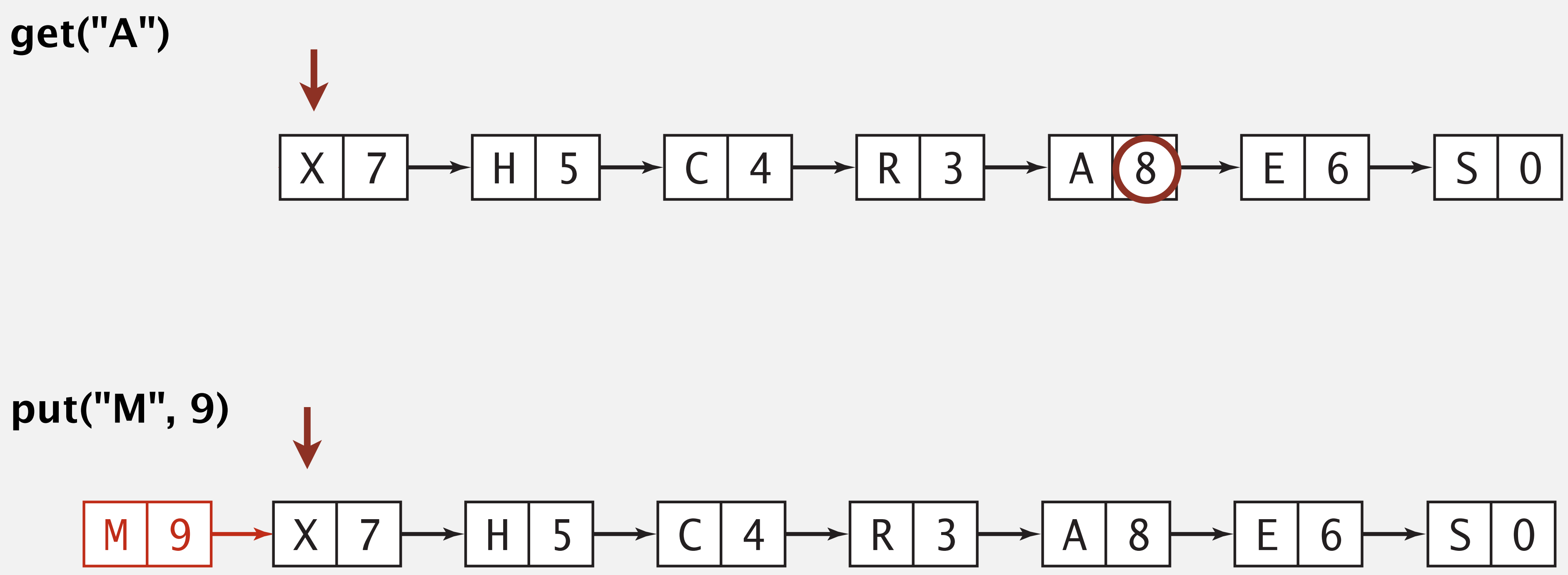
- API
- *elementary implementations*
- ordered operations

Sequential search in a linked list

Data structure. Maintain an (unordered) linked list of key–value pairs.

Search. Scan through all keys until find a match.

Insert. Scan through all keys until find a match; if no match add to front.



Elementary ST implementations: summary

implementation	guarantee (worst case)		average case		operations on keys
	search	insert	search hit	insert	
sequential search (unordered list)	n 🙁	n 🙁	n 🙁	n 🙁	`equals()`

Challenge. Efficient implementations of both search and insert.

Binary search in an ordered array

Data structure. Maintain parallel arrays for keys and values, sorted by keys.

Search. Use binary search to find key.

Proposition. At most $\sim \lg n$ compares to search a sorted array of length n.

keys[]

0	1	2	3	4	5	6	7	8	9
A	C	E	H	L	M	P	R	S	X

vals[]

0	1	2	3	4	5	6	7	8	9
8	4	2	5	11	9	10	3	0	7

get("P")

lo	hi	m										
0	9	4	A	C	E	H	L	M	P	R	S	X
5	9	7	A	C	E	H	L	M	P	R	S	X
5	6	5	A	C	E	H	L	M	P	R	S	X
6	6	6	A	C	E	H	L	M	P	R	S	X

entries in black are `a[lo..hi]`

entry in red is `a[m]`

`return vals[6]`

Binary search: insert

Data structure. Maintain an ordered array of key–value pairs.

Insert. Use binary search to find place to insert; shift all larger keys over.

Proposition. Takes linear time in the worst case.

put("P", 10)

keys[]

0	1	2	3	4	5	6	7	8	9
A	C	E	H	M	R	S	X	-	-

vals[]

0	1	2	3	4	5	6	7	8	9
8	4	6	5	9	3	0	7	-	-

Elementary ST implementations: summary

implementation	guarantee		average case		operations on keys
	search	insert	search hit	insert	
sequential search (unordered list)	n	n	n	n	`equals()`
binary search (ordered array)	$\log n$	n † 🙁	$\log n$	n † 🙁	`compareTo()`

† can do with $\log n$ compares, but requires n array accesses

Challenge. Efficient implementations of both search and insert.

https://algs4.cs.princeton.edu

3.1 Symbol Tables

- *API*
- *elementary implementations*
- *ordered operations*

Examples of ordered symbol table API

	keys	*values*
min() →	09:00:00	Chicago
	09:00:03	Phoenix
get(09:00:13) →	09:00:13	Houston
	09:00:59	Chicago
	09:01:10	Houston
floor(09:05:00) →	09:03:13	Chicago
	09:10:11	Seattle
select(7) →	09:10:25	Seattle
	09:14:25	Phoenix
keys(09:15:00, 09:25:00) →	09:19:32	Chicago
	09:19:46	Chicago
	09:21:05	Chicago
	09:22:43	Seattle
	09:22:54	Seattle
	09:25:52	Chicago
ceiling(09:30:00) →	09:35:21	Chicago
	09:36:14	Seattle
max() →	09:37:44	Phoenix

size(09:15:00, 09:25:00) *is* 5

rank(09:10:25) *is* 7

Ordered symbol table API

public class ST<Key extends Comparable<Key>, Value>	
⋮	
Key min()	*smallest key*
Key max()	*largest key*
Key floor(Key key)	*largest key less than or equal to key*
Key ceiling(Key key)	*smallest key greater than or equal to key*
int rank(Key key)	*number of keys less than key*
Key select(int k)	*key of rank k*
⋮	

Rank in a Sorted Array

Problem. Given a sorted array of n distinct keys, find the number of keys strictly less than a given query key.

Rank in a Sorted Array

Problem. Given a sorted array of n distinct keys, find the number of keys strictly less than a given query key.

easy modification to binary search

```
public Value get(Key key)  public int rank(Key key)
{
   int lo = 0, hi = n-1;
   while (lo <= hi)
   {
      int mid = lo + (hi - lo) / 2;
      int cmp = key.compareTo(keys[mid]);
      if      (cmp  < 0) hi = mid - 1;
      else if (cmp  > 0) lo = mid + 1;
      else if (cmp == 0) return vals[mid];  mid
   }
   return null;  lo
}
```

Binary search: ordered symbol table operations summary

	sequential search	binary search
search	n	$\log n$
insert	n	n
min / max	n	1
floor / ceiling	n	$\log n$
rank	n	$\log n$
select	n	1

order of growth of the running time for ordered symbol table operations