Symbol Tables

- API
- sequential search
- binary search
- applications

Algorithms in Java, 4th Edition · Robert Sedgewick and Kevin Wayne · Copyright © 2008 · October 7, 2008 7:01:34 AM

Symbol tables

Key-value pair abstraction.

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

Ex. DNS lookup.

- Insert URL with specified IP address.
- Given URL, find corresponding IP address.

URL	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`
`www.simpsons.com`	`209.052.165.60`

↑ key ↑ value

Symbol table applications

application	purpose of search	key	value
dictionary	look up word	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	value and type
routing table	route Internet packets	destination	best route
DNS	find IP address given URL	URL	IP address
reverse DNS	find URL given IP address	IP address	URL
genomics	find markers	DNA string	known positions
file system	find file on disk	filename	location on disk

Symbol table API

Associative array abstraction. Associate one value with each key.

```
      public class *ST<Key, Value>

                *ST()                             create a symbol table
          void  put(Key key, Value val)           put key-value pair into the table     ← a[key] = val;
         Value  get(Key key)                      return value paired with key          ← a[key]
       boolean  contains(Key key)                 is there a value paired with key?
          void  delete(Key key)                   delete key-value pair from table
 Iterator<Key>  iterator()                        iterator through keys in table
```

Conventions

- Values are not `null`.
- Method `get()` returns `null` if key not present.
- Method `put()` overwrites old value with new value.

Intended consequences.

- Easy to implement `contains()`.

```
public boolean contains(Key key)
{  return get(key) != null;  }
```

- Can implement lazy version of `delete()`.

```
public boolean delete(Key key)
{  put(key, null);           }
```

Keys and values

Value type. Any generic type.

Key type: several natural assumptions.

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

Best practices. Use immutable types for symbol table keys.

- Immutable in Java: `String`, `Integer`, `BigInteger`, ...
- Mutable in Java: `Date`, `GregorianCalendar`, `StringBuilder`, ...

ST test client

Build ST by associating value i with ith command-line argument.

```
public static void main(String[] args)
{
   ST<String, Integer> st = new ST<String, Integer>();
   for (int i = 0; i < args.length; i++)
      st.put(args[i], i);
   for (String s : st)
      StdOut.println(s + " " + st.get(s));
}
```

keys	S	E	A	R	C	H	E	X	A	M	P	L	E
values	0	1	2	3	4	5	6	7	8	9	10	11	12

ST output

```
A  8
C  4
E  12
H  5
L  9
M  11
P  10
R  3
S  0
X  7
```

▸ API

▸ **sequential search**

▸ binary search

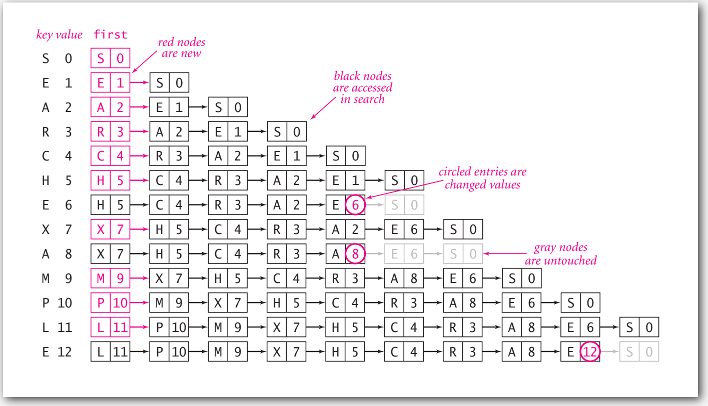
▸ applications

Sequential search

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.


key value
first
red nodes are new
black nodes are accessed in search
circled entries are changed values
gray nodes are untouched
S 0 — S 0
E 1 — E 1 → S 0
A 2 — A 2 → E 1 → S 0
R 3 — R 3 → A 2 → E 1 → S 0
C 4 — C 4 → R 3 → A 2 → E 1 → S 0
H 5 — H 5 → C 4 → R 3 → A 2 → E 1 → S 0
E 6 — H 5 → C 4 → R 3 → A 2 → E 6 → S 0
X 7 — X 7 → H 5 → C 4 → R 3 → A 2 → E 6 → S 0
A 8 — X 7 → H 5 → C 4 → R 3 → A 8 → E 6 → S 0
M 9 — M 9 → X 7 → H 5 → C 4 → R 3 → A 8 → E 6 → S 0
P 10 — P 10 → M 9 → X 7 → H 5 → C 4 → R 3 → A 8 → E 6 → S 0
L 11 — L 11 → P 10 → M 9 → X 7 → H 5 → C 4 → R 3 → A 8 → E 6 → S 0
E 12 — L 11 → P 10 → M 9 → X 7 → H 5 → C 4 → R 3 → A 8 → E 12 → S 0

Elementary ST implementations: summary

ST implementation	worst case		average case		ordered iteration?	operations on keys
	search	insert	search hit	insert		
sequential search (unordered list)	N	N	$N / 2$	N	no	`equals()`

Challenge. Efficient implementations of both search and insert.

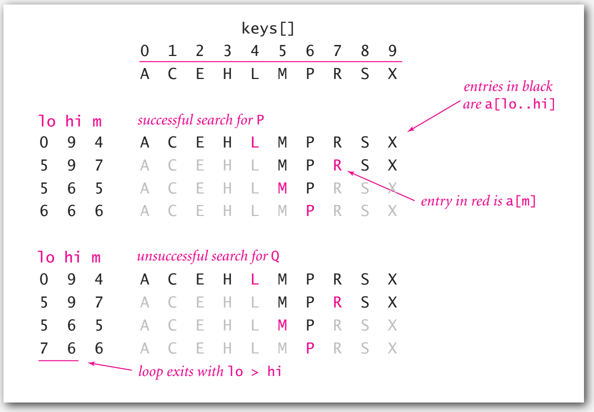
- API
- sequential search
- binary search
- applications

Binary search

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

Search. Binary search.

Insert. Binary search for key; if no match insert and shift larger keys.


keys[]
0 1 2 3 4 5 6 7 8 9
A C E H L M P R S X
entries in black are a[lo..hi]
entry in red is a[m]
lo hi m
successful search for P
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
lo hi m
unsuccessful search for Q
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
7 6 6 A C E H L M P R S X
loop exits with lo > hi

Binary search: Java implementation

```
public Value get(Key key)
{
   int i = bsearch(key);
   if (i == -1) return null;
   return vals[i];
}
```

symbol table method

```
private int bsearch(Key key)
{
   int lo = 0, hi = N-1;
   while (lo <= hi)
   {
       int m = lo + (hi - lo) / 2;
       int cmp = key.compareTo(keys[m]);
       if      (cmp  < 0) hi = m - 1;
       else if (cmp  > 0) lo = m + 1;
       else if (cmp == 0) return m;
  }
  return -1;
}
```

helper binary search method

not found

Binary search: mathematical analysis

Proposition. Binary search uses $\sim \lg N$ compares to search any array of size N.

Def. $\mathrm{T}(N) \equiv$ number of compares to binary search in a sorted array of size N.

$$\leq \mathrm{T}(N/2) + 1$$

left or right half

Binary search recurrence. $\mathrm{T}(N) \leq \mathrm{T}(N/2) + 1$ for $N > 1$, with $\mathrm{T}(1) = 1$.

- Not quite right for odd N.
- Same recurrence holds for many algorithms.

Solution. $\mathrm{T}(N) \sim \lg N$.

- For simplicity, we'll prove when N is a power of 2.
- True for all N. [see COS 340]

Binary search recurrence

Binary search recurrence. $\mathrm{T}(N) \leq \mathrm{T}(N/2) + 1$ for $N > 1$, with $\mathrm{T}(1) = 1$.

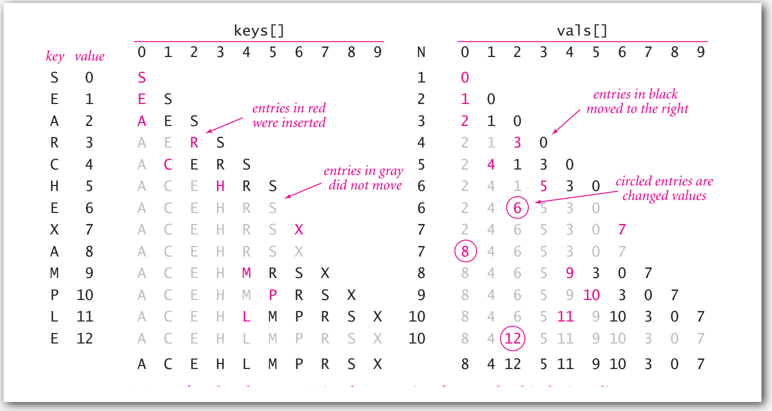
Proposition. If N is a power of 2, then $\mathrm{T}(N) \leq \lg N + 1$.

Pf.

$$\begin{aligned}
\mathrm{T}(N) &\leq \mathrm{T}(N/2) + 1 && \text{given} \\
&\leq \mathrm{T}(N/4) + 1 + 1 && \text{apply recurrence to first term} \\
&\leq \mathrm{T}(N/8) + 1 + 1 + 1 && \text{apply recurrence to first term} \\
&\;\;\cdots \\
&\leq \mathrm{T}(N/N) + 1 + 1 + \ldots + 1 && \text{stop applying, T(1) = 1} \\
&= \lg N + 1
\end{aligned}$$

Binary search: trace of standard indexing client

Problem. To insert, need to shift all greater keys over.



		keys[]											vals[]									
key	value	0	1	2	3	4	5	6	7	8	9	N	0	1	2	3	4	5	6	7	8	9
S	0	S										1	0									
E	1	E	S									2	1	0								
A	2	A	E	S								3	2	1	0							
R	3	A	E	R	S							4	2	1	3	0						
C	4	A	C	E	R	S						5	2	4	1	3	0					
H	5	A	C	E	H	R	S					6	2	4	1	5	3	0				
E	6	A	C	E	H	R	S					6	2	4	6	5	3	0				
X	7	A	C	E	H	R	S	X				7	2	4	6	5	3	0	7			
A	8	A	C	E	H	R	S	X				7	8	4	6	5	3	0	7			
M	9	A	C	E	H	M	R	S	X			8	8	4	6	5	9	3	0	7		
P	10	A	C	E	H	M	P	R	S	X		9	8	4	6	5	9	10	3	0	7	
L	11	A	C	E	H	L	M	P	R	S	X	10	8	4	6	5	11	9	10	3	0	7
E	12	A	C	E	H	L	M	P	R	S	X	10	8	4	12	5	11	9	10	3	0	7
		A	C	E	H	L	M	P	R	S	X		8	4	12	5	11	9	10	3	0	7

entries in red were inserted
entries in gray did not move
entries in black moved to the right
circled entries are changed values

Elementary ST implementations: summary

ST implementation	worst case		average case		ordered iteration?	operations on keys
	search	insert	search hit	insert		
sequential search (unordered list)	N	N	$N / 2$	N	no	`equals()`
binary search (ordered array)	$\log N$	N	$\log N$	$N / 2$	yes	`compareTo()`

Challenge. Efficient implementations of both search and insert.

- API
- sequential search
- binary search
- applications

Frequency counter

Goal. Read a sequence of strings from standard input and print out the number of times each string appears.

```
% more tiny.txt
it was the best of times
it was the worst of times
it was the age of wisdom
it was the age of foolishness


% java FrequencyCount < tiny.txt
2 age
1 best
1 foolishness
4 it
4 of
4 the
2 times
4 was
1 wisdom
1 worst
```

tiny example
24 words
10 distinct

```
% more tale.txt
it was the best of times
it was the worst of times
it was the age of wisdom
it was the age of foolishness
it was the epoch of belief
it was the epoch of incredulity
it was the season of light
it was the season of darkness
...

% java FrequencyCount < tale.txt
2941 a
1 aback
1 abandon
10 abandoned
1 abandoning
1 abandonment
1 abashed
1 abate
1 abated
...
```

real example
137177 words
9888 distinct

Frequency counter

```
public class FrequencyCount
{
   public static void main(String[] args)
   {
      ST<String, Integer> st = new ST<String, Integer>();

      while (!StdIn.isEmpty())
      {
         String key = StdIn.readString();
         if (!st.contains(key)) st.put(key, 1);
         else                   st.put(key, st.get(key) + 1);
      }

      for (String s: st)
         StdOut.println(st.get(s) + " " + s);
   }
}
```

create ST

read string and update frequency

print all strings

Set API

Mathematical set. A collection of distinct keys.

	public class SET<Key extends Comparable<Key>>	
	SET()	*create an empty set*
void	add(Key key)	*add the key to the set*
boolean	contains(Key key)	*is the key in the set?*
void	remove(Key key)	*remove the key from the set*
int	size()	*return the number of keys in the set*
Iterator<Key>	iterator()	*iterator through keys in the set*

Q. How to implement?

Exception filter

- Read in a list of words from one file.
- Print out all words from standard input that are in the list.

```
public class Whitelist
{
   public static void main(String[] args)
   {
      SET<String> set = new SET<String>();

      In in = new In(args[0]);
      while (!in.isEmpty())
         set.add(in.readString());

      while (!StdIn.isEmpty())
      {
         String word = StdIn.readString();
         if (set.contains(word))
            StdOut.println(word);
      }
   }
}
```

create empty set of strings

read in whitelist

print strings in list

Blacklist and whitelist applications

application	purpose	key	in list
spell checker	identify misspelled words	word	dictionary words
browser	mark visited pages	URL	visited pages
parental controls	block sites	URL	bad sites
chess	detect draw	board	positions
spam filter	eliminate spam	IP address	spam addresses
credit cards	check for stolen cards	number	stolen cards