3.5 Symbol Tables Applications

- sets
- dictionary clients
- indexing clients
- sparse vectors

- sets
- dictionary clients
- indexing clients
- sparse vectors

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, not in } the list.

```
% more list.txt
was it the of

% java WhiteList list.txt < tinyTale.txt
it was the of it was the of
it was the of it was the of
it was the of it was the of
it was the of it was the of
it was the of it was the of

% java BlackList list.txt < tinyTale.txt
best times worst times
age wisdom age foolishness
epoch belief epoch incredulity
season light season darkness
spring hope winter despair
```

list of exceptional words

Exception filter applications

- Read in a list of words from one file.
- Print out all words from standard input that are { in, not in } the list.

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

Exception filter: Java implementation

- Read in a list of words from one file.
- Print out all words from standard input that are { in, not in } the list.

```
public class WhiteList
{
   public static void main(String[] args)
   {
      SET<String> set = new SET<String>();        // create empty set of strings

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

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


Exception filter: Java implementation

- Read in a list of words from one file.
- Print out all words from standard input that are { in, not in } the list.

```
public class BlackList
{
   public static void main(String[] args)
   {
      SET<String> set = new SET<String>();        // create empty set of strings

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

      while (!StdIn.isEmpty())
      {
         String word = StdIn.readString();
         if (!set.contains(word))                 // print words not in list
            StdOut.println(word);
      }
   }
}
```


- sets
- **dictionary clients**
- indexing clients
- sparse vectors

Dictionary lookup

Command-line arguments.

- A comma-separated value (CSV) file.
- Key field.
- Value field.

Ex 1. DNS lookup.

URL is key · IP is value

```
% java LookupCSV ip.csv 0 1
adobe.com
192.150.18.60
www.princeton.edu
128.112.128.15
ebay.edu
Not found
```

IP is key · URL is value

```
% java LookupCSV ip.csv 1 0
128.112.128.15
www.princeton.edu
999.999.999.99
Not found
```

```
% more ip.csv
www.princeton.edu,128.112.128.15
www.cs.princeton.edu,128.112.136.35
www.math.princeton.edu,128.112.18.11
www.cs.harvard.edu,140.247.50.127
www.harvard.edu,128.103.60.24
www.yale.edu,130.132.51.8
www.econ.yale.edu,128.36.236.74
www.cs.yale.edu,128.36.229.30
espn.com,199.181.135.201
yahoo.com,66.94.234.13
msn.com,207.68.172.246
google.com,64.233.167.99
baidu.com,202.108.22.33
yahoo.co.jp,202.93.91.141
sina.com.cn,202.108.33.32
ebay.com,66.135.192.87
adobe.com,192.150.18.60
163.com,220.181.29.154
passport.net,65.54.179.226
tom.com,61.135.158.237
nate.com,203.226.253.11
cnn.com,64.236.16.20
daum.net,211.115.77.211
blogger.com,66.102.15.100
fastclick.com,205.180.86.4
wikipedia.org,66.230.200.100
rakuten.co.jp,202.72.51.22
...
```


Dictionary lookup

Command-line arguments.

- A comma-separated value (CSV) file.
- Key field.
- Value field.

Ex 2. Amino acids.

codon is key · name is value

```
% java Lookup amino.csv 0 3
ACT
Threonine
TAG
Stop
CAT
Histidine
```

```
% more amino.csv
TTT,Phe,F,Phenylalanine
TTC,Phe,F,Phenylalanine
TTA,Leu,L,Leucine
TTG,Leu,L,Leucine
TCT,Ser,S,Serine
TCC,Ser,S,Serine
TCA,Ser,S,Serine
TCG,Ser,S,Serine
TAT,Tyr,Y,Tyrosine
TAC,Tyr,Y,Tyrosine
TAA,Stop,Stop,Stop
TAG,Stop,Stop,Stop
TGT,Cys,C,Cysteine
TGC,Cys,C,Cysteine
TGA,Stop,Stop,Stop
TGG,Trp,W,Tryptophan
CTT,Leu,L,Leucine
CTC,Leu,L,Leucine
CTA,Leu,L,Leucine
CTG,Leu,L,Leucine
CCT,Pro,P,Proline
CCC,Pro,P,Proline
CCA,Pro,P,Proline
CCG,Pro,P,Proline
CAT,His,H,Histidine
CAC,His,H,Histidine
CAA,Gln,Q,Glutamine
CAG,Gln,Q,Glutamine
CGT,Arg,R,Arginine
CGC,Arg,R,Arginine
...
```


Dictionary lookup

Command-line arguments.

- A comma-separated value (CSV) file.
- Key field.
- Value field.

Ex 3. Class list.

login is key · first name is value

```
% java Lookup classlist.csv 4 1
eberl
Ethan
nwebb
Natalie
```

login is key · precept is value

```
% java Lookup classlist.csv 4 3
dpan
P01
```

```
% more classlist.csv
13,Berl,Ethan Michael,P01,eberl
11,Bourque,Alexander Joseph,P01,abourque
12,Cao,Phillips Minghua,P01,pcao
11,Chehoud,Christel,P01,cchehoud
10,Douglas,Malia Morioka,P01,malia
12,Haddock,Sara Lynn,P01,shaddock
12,Hantman,Nicole Samantha,P01,nhantman
11,Hesterberg,Adam Classen,P01,ahesterb
13,Hwang,Roland Lee,P01,rhwang
13,Hyde,Gregory Thomas,P01,ghyde
13,Kim,Hyunmoon,P01,hktwo
11,Kleinfeld,Ivan Maximillian,P01,ikleinfe
12,Korac,Damjan,P01,dkorac
11,MacDonald,Graham David,P01,gmacdona
10,Michal,Brian Thomas,P01,bmichal
12,Nam,Seung Hyeon,P01,seungnam
11,Nastasescu,Maria Monica,P01,mnastase
11,Pan,Di,P01,dpan
12,Partridge,Brenton Alan,P01,bpartrid
13,Rilee,Alexander,P01,arilee
13,Roopakalu,Ajay,P01,aroopaka
11,Sheng,Ben C,P01,bsheng
12,Webb,Natalie Sue,P01,nwebb
...
```


Dictionary lookup: Java implementation

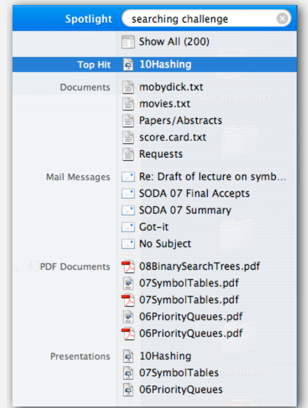
```
public class LookupCSV
{
   public static void main(String[] args)
   {
      In in = new In(args[0]);
      int keyField = Integer.parseInt(args[1]);
      int valField = Integer.parseInt(args[2]);

      ST<String, String> st = new ST<String, String>();
      while (!in.isEmpty())
      {
         String line = in.readLine();
         String[] tokens = database[i].split(",");
         String key = tokens[keyField];
         String val = tokens[valField];
         st.put(key, val);
      }

      while (!StdIn.isEmpty())
      {
         String s = StdIn.readString();
         if (!st.contains(s)) StdOut.println("Not found");
         else                 StdOut.println(st.get(s));
      }
   }
}
```

- process input file
- build symbol table
- process lookups with standard I/O

- sets
- dictionary clients
- **indexing clients**
- sparse vectors

File indexing

Goal. Index a PC (or the web).



File indexing

Goal. Given a list of files specified as command-line arguments, create an index so that can efficiently find all files containing a given query string.

```
% ls *.txt
aesop.txt magna.txt moby.txt
sawyer.txt tale.txt

% java FileIndex *.txt
freedom
magna.txt moby.txt tale.txt

whale
moby.txt

lamb
sawyer.txt aesop.txt
```

```
% ls *.java

% java FileIndex *.java
BlackList.java Concordance.java
DeDup.java FileIndex.java ST.java
SET.java WhiteList.java

import
FileIndex.java SET.java ST.java

Comparator
null
```

Solution. Key = query string; value = set of files containing that string.

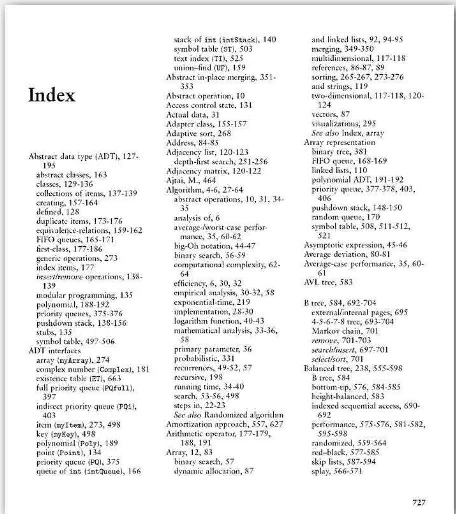
File indexing

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

      for (String filename : args) {
         File file = new File(filename);
         In in = new In(file);
         while !(in.isEmpty())
         {
            String word = in.readString();
            if (!st.contains(word))
               st.put(s, new SET<File>());
            SET<File> set = st.get(key);
            set.add(file);
         }
      }

      while (!StdIn.isEmpty())
      {
         String query = StdIn.readString();
         StdOut.println(st.get(query));
      }
   }
}
```

- symbol table
- list of file names from command line
- for each word in file, add file to corresponding set
- process queries

Book index

Goal. Index for an e-book.



Index

727

Concordance

Goal. Preprocess a text corpus to support concordance queries: given a word, find all occurrences with their immediate contexts.

```
% java Concordance tale.txt
cities
tongues of the two *cities* that were blended in

majesty
their turnkeys and the *majesty* of the law fired
me treason against the *majesty* of the people in
of his most gracious *majesty* king george the third

princeton
no matches
```


Concordance

```
public class Concordance
{
   public static void main(String[] args)
   {
      In in = new In(args[0]);
      String[] words = StdIn.readAll().split("\\s+");
      ST<String, SET<Integer>> st = new ST<String, SET<Integer>>();
      for (int i = 0; i < words.length; i++)
      {
         String s = words[i];
         if (!st.contains(s))
            st.put(s, new SET<Integer>());
         SET<Integer> pages = st.get(s);
         set.put(i);
      }

      while (!StdIn.isEmpty())
      {
         String query = StdIn.readString();
         SET<Integer> set = st.get(query);
         for (int k : set)
            // print words[k-5] to words[k+5]
      }
   }
}
```

read text and build index

process queries and print concordances

- sets
- dictionary clients
- indexing clients
- **sparse vectors**

Matrix-vector multiplication (standard implementation)

$$\underbrace{\begin{bmatrix} 0 & .90 & 0 & 0 & 0 \\ 0 & 0 & .36 & .36 & .18 \\ 0 & 0 & 0 & .90 & 0 \\ .90 & 0 & 0 & 0 & 0 \\ .47 & 0 & .47 & 0 & 0 \end{bmatrix}}_{a[][]} \underbrace{\begin{bmatrix} .05 \\ .04 \\ .36 \\ .37 \\ .19 \end{bmatrix}}_{x[]} = \underbrace{\begin{bmatrix} .036 \\ .297 \\ .333 \\ .045 \\ .1927 \end{bmatrix}}_{b[]}$$

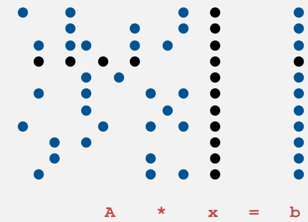
```
...
double[][] a = new double[N][N];
double[] x = new double[N];
double[] b = new double[N];
...
// initialize a[][] and x[]
...
for (int i = 0; i < N; i++)
{
   sum = 0.0;
   for (int j = 0; j < N; j++)
      sum += a[i][j]*x[j];
   b[i] = sum;
}
```

nested loops
N^2 running time

Sparse matrix-vector multiplication

Problem. Sparse matrix-vector multiplication.

Assumptions. Matrix dimension is 10,000; average nonzeros per row ~ 10.



Vector representations

1D array (standard) representation.

- Constant time access to elements.
- Space proportional to N.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
0	.36	0	0	0	.36	0	0	0	0	0	0	0	0	.18	0	0	0	0	0

Symbol table representation.

- key = index, value = entry
- Efficient iterator.
- Space proportional to number of nonzeros.

st → (key, value)

key	value
1	.36
5	.36
14	.18

Sparse vector data type

```
public class SparseVector
{
   private HashST<Integer, Double> v;

   public SparseVector()
   {  v = new HashST<Integer, Double>();  }

   public void put(int i, double x)
   {  v.put(i, x);  }

   public double get(int i)
   {
      if (!v.contains(i)) return 0.0;
      else return v.get(i);
   }

   public Iterable<Integer> indices()
   {  return v.keys();  }

   public double dot(double[] that)
   {
       double sum = 0.0;
       for (int i : indices())
           sum += that[i]*this.get(i);
       return sum;
   }
}
```

- `HashST` because order not important
- empty ST represents all 0s vector
- `a[i] = value`
- `return a[i]`
- dot product is constant time for sparse vectors

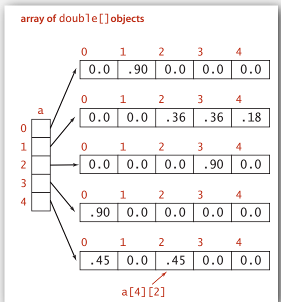
Matrix representations

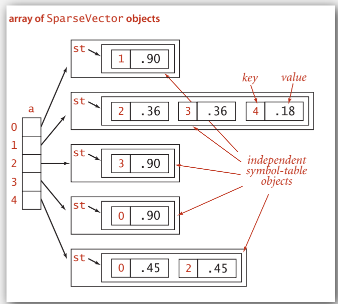
2D array (standard) representation: Each row of matrix is an array.

- Constant time access to elements.
- Space proportional to N^2.

Sparse representation: Each row of matrix is a sparse vector.

- Efficient access to elements.
- Space proportional to number of nonzeros (plus N).





Sparse matrix-vector multiplication

a[][] x[] b[]

$$\begin{bmatrix} 0 & .90 & 0 & 0 & 0 \\ 0 & 0 & .36 & .36 & .18 \\ 0 & 0 & 0 & .90 & 0 \\ .90 & 0 & 0 & 0 & 0 \\ .47 & 0 & .47 & 0 & 0 \end{bmatrix} \begin{bmatrix} .05 \\ .04 \\ .36 \\ .37 \\ .19 \end{bmatrix} = \begin{bmatrix} .036 \\ .297 \\ .333 \\ .045 \\ .1927 \end{bmatrix}$$

```
..
SparseVector[] a;
a = new SparseVector[N];
double[] x = new double[N];
double[] b = new double[N];
...
// Initialize a[] and x[]
...
for (int i = 0; i < N; i++)
   b[i] = a[i].dot(x);
```

one loop
linear running time
for sparse matrix

- sets
- dictionary clients
- indexing clients
- sparse vectors
- **challenges**

Searching challenge 2A:

Problem. IP lookups in a web monitoring device.

Assumption A. Billions of lookups, millions of distinct addresses.

Which searching method to use?

1) Sequential search in a linked list.
2) Binary search in an ordered array.
3) Need better method, all too slow.
4) Doesn't matter much, all fast enough.

Searching challenge 2A

Problem. IP lookups in a web monitoring device.

Assumption A. Billions of lookups, millions of distinct addresses.

Which searching method to use?

1) Sequential search in a linked list.
2) Binary search in an ordered array.
✓ 3) Need better method, all too slow.
4) Doesn't matter much, all fast enough.

total cost of insertions is $c*1000000^2$ = $c*1{,}000{,}000{,}000{,}000$ (way too much)

Searching challenge 2B

Problem. IP lookups in a web monitoring device.

Assumption B. Billions of lookups, thousands of distinct addresses.

Which searching method to use?

1) Sequential search in a linked list.
2) Binary search in an ordered array.
3) Need better method, all too slow.
4) Doesn't matter much, all fast enough.

Searching challenge 2B

Problem. IP lookups in a web monitoring device.

Assumption B. Billions of lookups, thousands of distinct addresses.

Which searching method to use?

1) Sequential search in a linked list.
✓ 2) Binary search in an ordered array.
3) Need better method, all too slow.
4) Doesn't matter much, all fast enough.

total cost of insertions is $c_1*1000^2 = c_1*1000000$ and dominated by $c_2*1000000000$ cost of lookups

Searching challenge 4

Problem. Spell checking for a book.

Assumptions. Dictionary has 25,000 words; book has 100,000+ words.

Which searching method to use?

1) Sequential search in a linked list.
2) Binary search in an ordered array.
3) Need better method, all too slow.
4) Doesn't matter much, all fast enough.

Searching challenge 4

Problem. Spell checking for a book.
Assumptions. Dictionary has 25,000 words; book has 100,000+ words.

Which searching method to use?
1) Sequential search in a linked list.
✓ 2) Binary search in an ordered array. ← easy to presort dictionary total cost of lookups is optimal c_2*1,500,000
3) Need better method, all too slow.
4) Doesn't matter much, all fast enough.

Searching challenge 1A

Problem. Maintain symbol table of song names for an iPod.
Assumption A. Hundreds of songs.

Which searching method to use?
1) Sequential search in a linked list.
2) Binary search in an ordered array.
3) Need better method, all too slow.
4) Doesn't matter much, all fast enough.

Searching challenge 1A

Problem. Maintain symbol table of song names for an iPod.
Assumption A. Hundreds of songs.

Which searching method to use?
1) Sequential search in a linked list.
2) Binary search in an ordered array.
3) Need better method, all too slow.
✓ 4) Doesn't matter much, all fast enough. ← $100^2 = 10{,}000$

Searching challenge 1B

Problem. Maintain symbol table of song names for an iPod.
Assumption B. Thousands of songs.

Which searching method to use?
1) Sequential search in a linked list.
2) Binary search in an ordered array.
3) Need better method, all too slow.
4) Doesn't matter much, all fast enough.

Searching challenge 1B

Problem. Maintain symbol table of song names for an iPod.
Assumption B. Thousands of songs.

Which searching method to use?
1) Sequential search in a linked list.
2) Binary search in an ordered array. ← maybe, but 1000^2 = 1,000,000 so user might wait for complete rebuild of index
✓ 3) Need better method, all too slow.
4) Doesn't matter much, all fast enough.

Searching challenge 3

Problem. Frequency counts in "Tale of Two Cities."
Assumptions. Book has 135,000+ words; about 10,000 distinct words.

Which searching method to use?
1) Sequential search in a linked list.
2) Binary search in an ordered array.
3) Need better method, all too slow.
4) Doesn't matter much, all fast enough.

Searching challenge 3

Problem. Frequency counts in "Tale of Two Cities."
Assumptions. Book has 135,000+ words; about 10,000 distinct words.

Which searching method to use?
1) Sequential search in a linked list. ← total cost of searches: c_2*1,350,000,000
2) Binary search in an ordered array. ← maybe, but total cost of insertions is c_1*100,000,000
✓ 3) Need better method, all too slow.
4) Doesn't matter much, all fast enough.

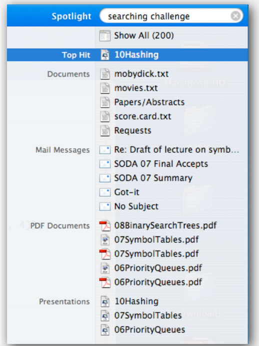
Searching challenge 3 (revisited):

Problem. Frequency counts in "Tale of Two Cities"
Assumptions. Book has 135,000+ words; about 10,000 distinct words.

Which searching method to use?
1) Sequential search in a linked list.
2) Binary search in an ordered array.
3) Need better method, all too slow.
4) Doesn't matter much, all fast enough.
✓ 5) BSTs. ←
insertion cost < 10000 * 1.38 * lg 10000 < .2 million
lookup cost < 135000 * 1.38 * lg 10000 < 2.5 million

Searching challenge 5

Problem. Index for a PC or the web.
Assumptions. 1 billion++ words to index.

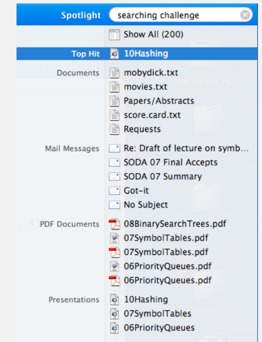


Which searching method to use?

- Hashing
- Red-black-trees
- Doesn't matter much.

Searching challenge 5

Problem. Index for a PC or the web.
Assumptions. 1 billion++ words to index.



Which searching method to use?

- ✓ Hashing
- Red-black-trees ← too much space
- Doesn't matter much.

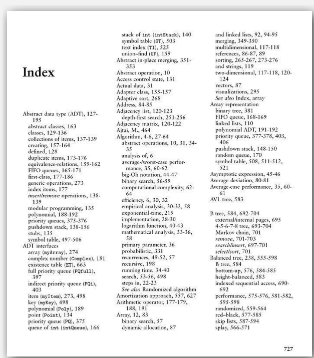
Solution. Symbol table with:

- Key = query string.
- Value = set of pointers to files.

sort the (relatively few) search hits

Searching challenge 6

Problem. Index for an e-book.
Assumptions. Book has 100,000+ words.

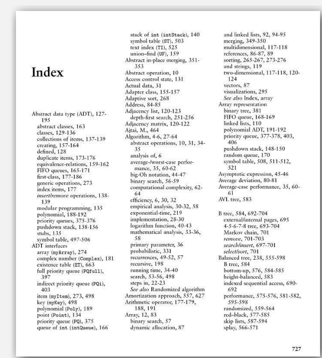


Which searching method to use?

1. Hashing
2. Red-black-tree
3. Doesn't matter much.

Searching challenge 6

Problem. Index for an e-book.
Assumptions. Book has 100,000+ words.



Which searching method to use?

1. Hashing
2. ✓ Red-black-tree ← need ordered iteration
3. Doesn't matter much.

Solution. Symbol table with:

- Key = index term.
- Value = ordered set of pages on which term appears.