



<http://algs4.cs.princeton.edu>

## 3.5 SYMBOL TABLE APPLICATIONS

---

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



## 3.5 SYMBOL TABLE APPLICATIONS

---

- ▶ *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
```

← list of exceptional words

```
% 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
```

## 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 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 words not in list

## Exception filter: Java implementation

---

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

```
public class BlackList
{
    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 words not in list



## 3.5 SYMBOL TABLE APPLICATIONS

---

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





## 3.5 SYMBOL TABLE APPLICATIONS

---

- ▶ *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.

domain name 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 domain name 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 LookupCSV 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.

```
% java LookupCSV classlist.csv 4 1
```

```
eberl
```

```
Ethan
```

```
nwebb
```

```
Natalie
```

```
% java LookupCSV classlist.csv 4 3
```

```
dpan
```

```
P01
```

first name  
login is key is value

section  
login is key is value

```
% more classlist.csv
```

```
13,Berl,Ethan Michael,P01,eberl
```

```
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
```

```
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]);
```

← process input file

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

← build symbol table

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

← process lookups  
with standard I/O

```
    }
}
```



## 3.5 SYMBOL TABLE APPLICATIONS

---

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



## 3.5 SYMBOL TABLE APPLICATIONS

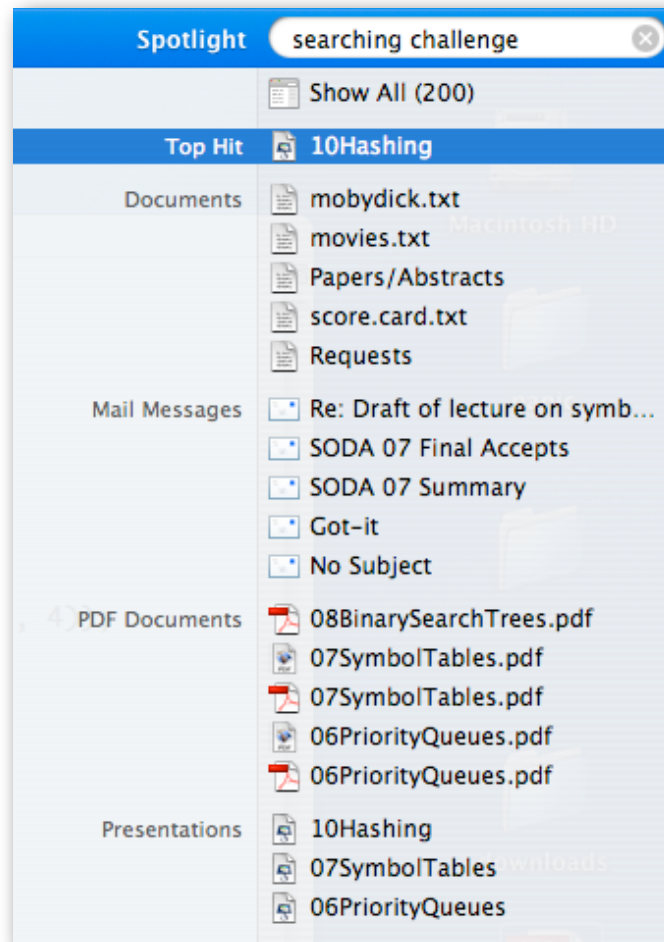
---

- ▶ *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, create an index so that you 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
BlackList.java Concordance.java
DeDup.java FileIndex.java ST.java
SET.java WhiteList.java

% java FileIndex *.java

import
FileIndex.java SET.java ST.java

Comparator
null
```

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

# File indexing

```
import java.io.File;
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 key = in.readString();
                if (!st.contains(key))
                    st.put(key, 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

- Abstract data type (ADT), 127-195
  - abstract classes, 163
  - classes, 129-136
  - collections of items, 137-139
  - creating, 157-164
  - defined, 128
  - duplicate items, 173-176
  - equivalence-relations, 159-162
  - FIFO queues, 165-171
  - first-class, 177-186
  - generic operations, 273
  - index items, 177
  - insert/remove* operations, 138-139
  - modular programming, 135
  - polynomial, 188-192
  - priority queues, 375-376
  - pushdown stack, 138-156
  - stubs, 135
  - symbol table, 497-506
- ADT interfaces
  - array (*myArray*), 274
  - complex number (*Complex*), 181
  - existence table (*ET*), 663
  - full priority queue (*PQfull*), 397
  - indirect priority queue (*PQi*), 403
  - item (*myItem*), 273, 498
  - key (*myKey*), 498
  - polynomial (*Poly*), 189
  - point (*Point*), 134
  - priority queue (*PQ*), 375
  - queue of *int* (*intQueue*), 166
  - stack of *int* (*intStack*), 140
  - symbol table (*ST*), 503
  - text index (*TI*), 525
  - union-find (*UF*), 159
- Abstract in-place merging, 351-353
- Abstract operation, 10
- Access control state, 131
- Actual data, 31
- Adapter class, 155-157
- Adaptive sort, 268
- Address, 84-85
- Adjacency list, 120-123
  - depth-first search, 251-256
- Adjacency matrix, 120-122
- Ajtai, M., 464
- Algorithm, 4-6, 27-64
  - abstract operations, 10, 31, 34-35
  - analysis of, 6
  - average-/worst-case performance, 35, 60-62
  - big-Oh notation, 44-47
  - binary search, 56-59
  - computational complexity, 62-64
  - efficiency, 6, 30, 32
  - empirical analysis, 30-32, 58
  - exponential-time, 219
  - implementation, 28-30
  - logarithm function, 40-43
  - mathematical analysis, 33-36, 58
  - primary parameter, 36
  - probabilistic, 331
  - recurrences, 49-52, 57
  - recursive, 198
  - running time, 34-40
  - search, 53-56, 498
  - steps in, 22-23
  - See also* Randomized algorithm
- Amortization approach, 557, 627
- Arithmetic operator, 177-179, 188, 191
- Array, 12, 83
  - binary search, 57
  - dynamic allocation, 87
  - and linked lists, 92, 94-95
  - merging, 349-350
  - multidimensional, 117-118
  - references, 86-87, 89
  - sorting, 265-267, 273-276
  - and strings, 119
  - two-dimensional, 117-118, 120-124
  - vectors, 87
  - visualizations, 295
  - See also* Index, array
- Array representation
  - binary tree, 381
  - FIFO queue, 168-169
  - linked lists, 110
  - polynomial ADT, 191-192
  - priority queue, 377-378, 403, 406
  - pushdown stack, 148-150
  - random queue, 170
  - symbol table, 508, 511-512, 521
- Asymptotic expression, 45-46
- Average deviation, 80-81
- Average-case performance, 35, 60-61
- AVL tree, 583
- B tree, 584, 692-704
  - external/internal pages, 695
  - 4-5-6-7-8 tree, 693-704
  - Markov chain, 701
  - remove*, 701-703
  - search/insert*, 697-701
  - select/sort*, 701
- Balanced tree, 238, 555-598
  - B tree, 584
  - bottom-up, 576, 584-585
  - height-balanced, 583
  - indexed sequential access, 690-692
  - performance, 575-576, 581-582, 595-598
  - randomized, 559-564
  - red-black, 577-585
  - skip lists, 587-594
  - splay, 566-571

# 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 = in.readAllStrings();
        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> set = st.get(s);
            set.add(i);
        }

        while (!StdIn.isEmpty())
        {
            String query = StdIn.readString();
            SET<Integer> set = st.get(query);
            for (int k : set)
                // print words[k-4] to words[k+4]
        }
    }
}
```

← read text and  
build index

← process queries  
and print  
concordances



## 3.5 SYMBOL TABLE APPLICATIONS

---

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



## 3.5 SYMBOL TABLE APPLICATIONS

---

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

# Matrix-vector multiplication (standard implementation)

---

$$\begin{array}{c} \text{a}[][] \\ \left[ \begin{array}{ccccc} 0 & .90 & 0 & 0 & 0 \\ 0 & 0 & .36 & .36 & .18 \\ 0 & 0 & 0 & .90 & 0 \\ .90 & 0 & 0 & 0 & 0 \\ .47 & 0 & .47 & 0 & 0 \end{array} \right] \end{array} \begin{array}{c} \text{x}[] \\ \left[ \begin{array}{c} .05 \\ .04 \\ .36 \\ .37 \\ .19 \end{array} \right] \end{array} = \begin{array}{c} \text{b}[] \\ \left[ \begin{array}{c} .036 \\ .297 \\ .333 \\ .045 \\ .1927 \end{array} \right] \end{array}$$

```
...
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)

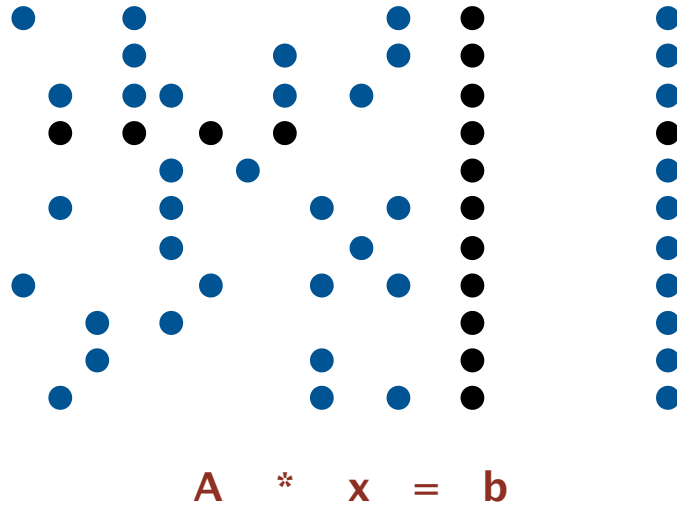


# Sparse matrix-vector multiplication

---

**Problem.** Sparse matrix-vector multiplication.

**Assumptions.** Matrix dimension is 10,000; average nonzeros per row  $\sim 10$ .



# Vector representations

---

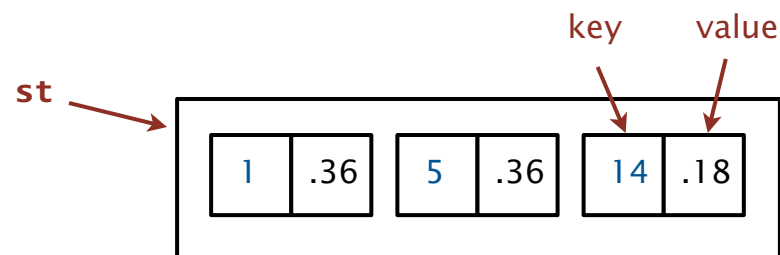
## 1d array (standard) representation.

- 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.



# Sparse vector data type

---

```
public class SparseVector  
{
```

```
    private HashST<Integer, Double> v;
```

← HashST because order not important

```
    public SparseVector()  
    {
```

```
        v = new HashST<Integer, Double>();  
    } }
```

← empty ST represents all 0s vector

```
    public void put(int i, double x)  
    {
```

```
        v.put(i, x);  
    }
```

← a[i] = value

```
    public double get(int i)  
    {
```

```
        if (!v.contains(i)) return 0.0;  
        else return v.get(i);  
    }
```

← return a[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;  
    }
```

← dot product is constant  
time for sparse vectors

```
}
```

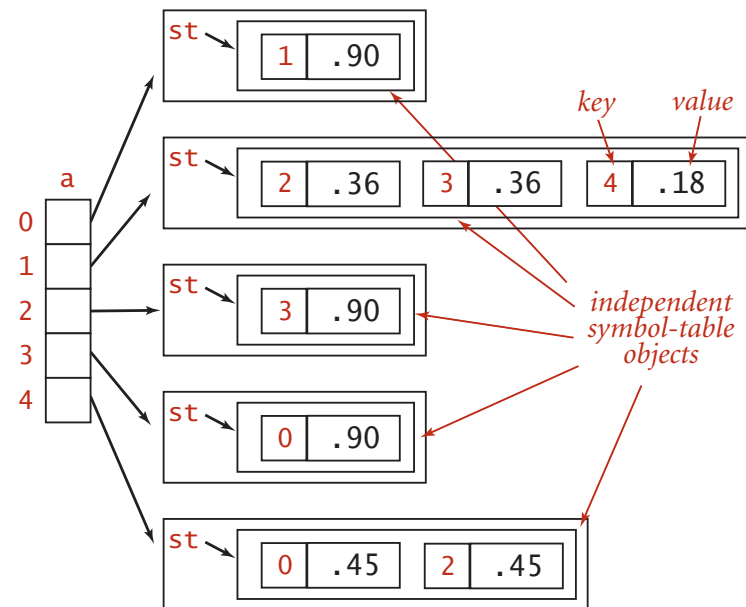
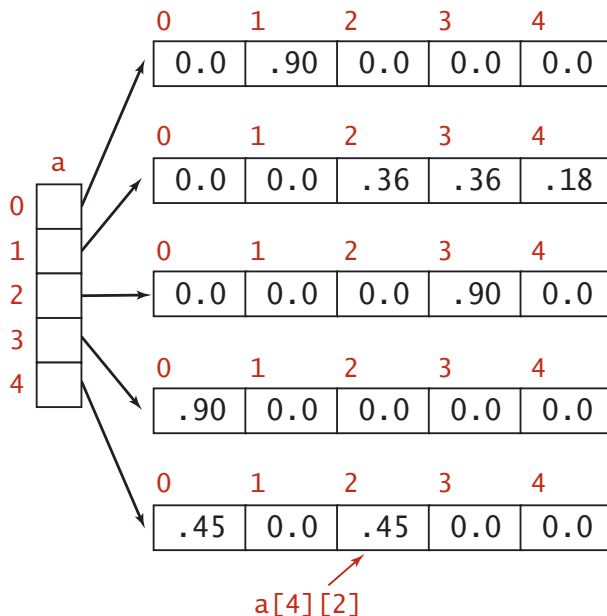
# Matrix representations

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

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

**Sparse matrix 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

---

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

```
..  
SparseVector[] 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);
```

← linear running time  
for sparse matrix



## 3.5 SYMBOL TABLE APPLICATIONS

---

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



<http://algs4.cs.princeton.edu>

## 3.5 SYMBOL TABLE APPLICATIONS

---

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