# 4.5 Symbol Table Applications

Robert Sedgewick and Kevin Wayne · Copyright © 2005 · http://www.Princeton.EDU/~cos226

### Set Client: Remove Duplicates

Remove duplicates. [e.g., from commercial mailing list]

- Read in a key.
- If key is not in set, insert and print it out.

#### Set ADT

### Set. Unordered collection of distinct keys.

#### API for SET.

add (key) insert the key into the set
 contains (key) is the given key in the set?
 remove (key) remove the key from the set
 iterator() return iterator over all keys

### Q. How to implement?

Java library. java.util.HashSet.

## More Set Applications

Application	Purpose	Key
Spell checker	Identify misspelled words	Word
Browser	Highlight previously visited pages	URL
Chess	Detect repetition draw	Board position
Spam blacklist	Prevent spam	IP address
Security whitelist	Allow trusted traffic	IP address
Credit card fraud	Identify stolen credit cards	Credit card number

4

# Inverted Index

Robert Sedgewick and Kevin Wayne · Copyright © 2005 · http://www.Princeton.EDU/~cos226

### Inverted Index Implementation

#### Inverted Index

Inverted index. Given a list of page, preprocess so that you can quickly find all pages containing a given query.

Ex 1. Book index.

Ex 2. Web search engine index.

Ex 3. File index, e.g., Spotlight.

Key. Query word. Value. Set of pages.

	stack of sat (iastStank), 140 symbol table (ST), 593 text index (TI), 525 taxon-find (SF), 159 Abstract in place merging, 351	and linked lize, 92, 94-95 merging, 349-350 multidimensional, 117-118 reference, 84-87, 89 seering, 263-267, 273-276
Index	333 Abstract operation, 10 Access control state, 131 Actual data, 31 Adapter class, 155-157 Adaptic seet, 248 Address, 84-85	and strings, 119 two-dimensional, 117-118, 120- 124 tection, 87 translitations, 295 See also bides, acray Assay representation
Abstract data type (ADT), 127- 195	Adjacency list, 120-123 depth-first search, 251-256 Adjacency matrix, 120-122	binary tree, 581 HFO queue, 168-169 linked lies, 110
abstract classes, 163 classes, 129-136 collections of items, 137-139 corating, 157-164 defined, 128	Ajrai, M., 464 Algorithm, 4-6, 27-64 abstract operations, 10, 31, 34- 35	polynomial ADT, 191-192 priority queue, 377-378, 403, 406 pushdown stack, 148-190
duplicate items, 173-176 equivalence-eclations, 159-162 FIFO queues, 165-171	analysis of, 6 average-fevera-case perfor- mance, 35, 60-62	random queue, 170 symbol table, 506, 511-512, 521
first-class, 177-186 generic operations, 273	big-Oh notation, 44-47 binary wards, 56-59 communicated complexity, 62-	Asymptotic expression, 45-46 Asymptoticon, 80-81 Asymptoticon, 80-81 Asymptoticon, 35, 60-
index items, 177 insertiveness operations, 135- 139	64 efficiency, 6, 30, 32	61 AVL tree, 183
modular programming, 135 polynomial, 188-192 priority appace, 375-376	empirical analysis, 50-32, 58 exponential time, 219 implementation, 28-30	B tree, 584, 692-704 corema/internal pages, 695
peshdown stack, 158-156 stabe, 135 symbol table, 497-506	logarithm function, 40-43 mathematical analysis, 33-36,	4.5-6-7-8 tree, 693-704 Markov chain, 701 remon, 700-703
ADT interfaces array (askgraw), 274	primary parameter, 36 poobabilistic, 331	search/bosest, 697-701 select/sort, 701
complex number (Complex), 181 existence table (III), 663	recurrences, 49-52, 57 recurring firm, 34-40	Balanced tree, 238, 555-598 B tree, 584
fall priority queue (PQfall), 397 indirect priority queue (PQI),	search, 53-56, 498 steps in, 22-23	heremore, 576, 584-585 height-halanced, 583 indexed regrential access, 650-
403 itom inglites), 273, 498	See also Randomized algorithm Amortization approach, 557, 627	692 performance, 575-576, 581-582,
key (ngKey), 498 polynomial (Poly), 189 point (Point), 134	Arithmetic operator, 177-179, 188, 191 Array, 12, 83	595-598 randomized, 559-564 rad-black, 577-585
priority queue (PQL 375 queue of int (intQueue), 166	binary search, 57 dynamic allocation, 87	skip lim, 587-594 selec, 566-571

### Inverted Index

#### Extensions.

- Ignore stopwords: the, on, of, etc.
- Multi-word queries:
  - set intersection (AND)
  - set union (OR).
- Record position and number of occurrences of word in document.

#### Vectors and Matrices

# Sparse Vectors and Matrices

Vector. Ordered sequence of N real numbers. Matrix. N-by-N table of real numbers.

$$a = \begin{bmatrix} 0 & 3 & 15 \end{bmatrix}, b = \begin{bmatrix} -1 & 2 & 2 \end{bmatrix}$$
  
 $a + b = \begin{bmatrix} -1 & 5 & 17 \end{bmatrix}$   
 $a \circ b = (0 \cdot -1) + (3 \cdot 2) + (15 \cdot 2) = 36$ 

$$A = \begin{bmatrix} 0 & 1 & 1 \\ 2 & 4 & -2 \\ 0 & 3 & 15 \end{bmatrix}, \quad B = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 3 \end{bmatrix}, \quad A + B = \begin{bmatrix} 1 & 1 & 1 \\ 2 & 6 & -2 \\ 0 & 3 & 18 \end{bmatrix}$$

$$\begin{bmatrix} 0 & 1 & 1 \\ 2 & 4 & -2 \\ 0 & 3 & 15 \end{bmatrix} \times \begin{bmatrix} -1 \\ 2 \\ 2 \end{bmatrix} = \begin{bmatrix} 4 \\ 2 \\ 36 \end{bmatrix}$$

Robert Sedgewick and Kevin Wayne  $\cdot$  Copyright © 2005  $\cdot$  http://www.Princeton.EDU/~cos226

#### Sparsity

Def. An N-by-N matrix is sparse if it has O(N) nonzeros entries.

Empirical fact. Large matrices that arise in practice are usually sparse.

#### Matrix representations.

- 2D array: space proportional to N<sup>2</sup>.
- Goal: space proportional to number of nonzeros, without sacrificing fast access to individual elements.

Ex. Google performs matrix-vector product with N = 4 billion!

Sparse Vector Implementation

11

```
// return a · b
public double dot(SparseVector b) {
    SparseVector a = this;
    double sum = 0.0;
    for (int i : a.st)
        if (b.st.contains(i)) sum += a.get(i) * b.get(i);
    return sum;
}

// return c = a + b
public SparseVector plus(SparseVector b) {
    SparseVector a = this;
    SparseVector c = new SparseVector(N);
    for (int i : a.st) c.put(i, a.get(i));
    for (int i : b.st) c.put(i, b.get(i) + c.get(i));
    return c;
}
```

```
public class SparseMatrix {
   private final int N;
   private SparseVector[] rows;
   public SparseMatrix(int N) {
                                                           N-by-N matrix
      this.N = N;
                                                           of all zeros
      rows = new SparseVector[N];
     for (int i = 0; i < N; i++)
         rows[i] = new SparseVector(N);
                                                           A[i][j] = value
   public void put(int i, int j, double value) {
      rows[i].put(j, value);
   public double get(int i, int j) {
                                                           A[i][i]
     return rows[i].get(j);
```

# Sparse Matrix Implementation (cont)

```
// return b = A*x
public SparseVector times(SparseVector x) {
    SparseMatrix A = this;
    SparseVector b = new SparseVector(N);
    for (int i = 0; i < N; i++)
        b.put(i, rows[i].dot(x));
    return b;
}

// return C = A + B
public SparseMatrix plus(SparseMatrix B) {
    SparseMatrix A = this;
    SparseMatrix C = new SparseMatrix(N);
    for (int i = 0; i < N; i++)
        C.rows[i] = A.rows[i].plus(B.rows[i]);
    return C;
}</pre>
```

# Similarity Search

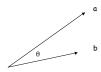
### Similarity Measure

Dot product. Given two vectors  $a = (a_1, ..., a_n)$  and  $b = (b_1, ..., b_n)$ , their dot product  $a \cdot b = a_1b_1 + ... + a_nb_n$ .

Magnitude. The magnitude of vector a is given by  $|a| = \sqrt{a \cdot a}$ .

Similarity. Given two vectors a and b,  $\cos \theta = \frac{a \cdot b}{|a| |b|}$ 

- Similar if  $\cos \theta$  close to 1.
- Not similar if  $\cos \theta$  close to 0.



Applications. Literature, code, music, video, artwork.

# A Plan for Spam

#### Bayesian spam filter.

- Filter based on analysis of previous messages.
- User trains the filter by classifying messages as spam or ham.
- Parse messages into tokens (alphanumeric, dashes, ', \$).

#### Build data structures.

- Symbol table A of tokens and frequencies for spam.
- Symbol table B of tokens and frequencies for ham.
- Symbol table C of tokens with prob p that they appear in spam.

```
double h = 2.0 * ham.freq(word);
                                                   bias probabilities to
double s = 1.0 * spam.freq(word);
                                                   avoid false positives
double p = (s/spams) / (h/hams + s/spams);
```

Reference: http://www.paulgraham.com/spam.html

# A Plan for Spam

Robert Sedgewick and Kevin Wayne · Copyright © 2005 · http://www.Princeton.EDU/~cos226

### A Plan for Spam

#### Identify incoming email as spam or ham.

- Find 15 most interesting tokens (difference from 0.5).
- Combine probabilities using Bayes law.

$$\frac{p_1 \times p_2 \times \dots \times p_{15}}{(p_1 \times p_2 \times \dots \times p_{15}) + ((l-p_1) \times (l-p_2) \times \dots \times (l-p_{15}))}$$

• Declare as spam if threshold > 0.9.

#### Details.

- Words you've never seen.
- Words that appear in ham corpus but not spam corpus, vice versa.
- Words that appear less than 5 times in spam and ham corpuses.
- Update data structures.