4.5 Symbol Table Applications

Set ADT

Set ADT: unordered collection of distinct keys.
- **Insert** a key.
- **Check** if set contains a given key.
- **Delete** a key.

**SET interface.**
- `add(key)` insert the key
- `contains(key)` is given key present?
- `remove(key)` remove the key
- `iterator()` return iterator over all keys

**Q. How to implement?**

Java library: `java.util.HashSet`.

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.

```java
public class DeDup {
    public static void main(String[] args) {
        SET<String> set = new SET<String>();
        while (!StdIn.isEmpty()) {
            String key = StdIn.readString();
            if (!set.contains(key)) {
                set.add(key);
                System.out.println(key);
            }
        }
    }
}
```

More Set Applications

<table>
<thead>
<tr>
<th>Application</th>
<th>Purpose</th>
<th>Key</th>
</tr>
</thead>
<tbody>
<tr>
<td>Spell checker</td>
<td>Identify misspelled words</td>
<td>Word</td>
</tr>
<tr>
<td>Browser</td>
<td>Highlight previously visited pages</td>
<td>URL</td>
</tr>
<tr>
<td>Chess</td>
<td>Detect repetition draw</td>
<td>Board position</td>
</tr>
<tr>
<td>Spam blacklist</td>
<td>Prevent spam</td>
<td>IP addresses of spammers</td>
</tr>
</tbody>
</table>
Inverted Index Implementation

```java
public class InvertedIndex {
    public static void main(String[] args) {
        ST<String, SET<String>> st = new ST<String, SET<String>>();
        for (String filename : args) {
            In in = new In(filename);
            while (!in.isEmpty()) {
                String word = in.readString();
                if (!st.containsKey(word))
                    st.put(word, new SET<String>());
                st.get(word).add(filename);
            }
        }
    }
}
```

Inverted Index. Given a list of documents, preprocess so that you can quickly find all documents containing a given query.

Ex 1. Book index.
Ex 2. Web search engine index.

Key. Query word.
Value. Set of documents.
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.

Sparse Vectors and Matrices

Vectors and Matrices

**Vector.** Ordered sequence of \( N \) real numbers.

**Matrix.** \( N \)-by-\( N \) table of real numbers.

\[
\begin{bmatrix}
0 & 3 & 15 \\
2 & 4 & -2 \\
0 & 3 & 15
\end{bmatrix}
\]

\[
\begin{bmatrix}
1 & 0 & 0 \\
0 & 2 & 0 \\
0 & 0 & 3
\end{bmatrix}
\]

\[
\begin{bmatrix}
1 & 1 & 1 \\
2 & 6 & -2 \\
0 & 3 & 18
\end{bmatrix}
\]

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

**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 \( N^2 \).
- 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

```java
public class SparseVector {
    private final int N;
    private ST<Integer, Double> st = ...;

    public SparseVector(int N) { this.N = N; }
    public void put(int i, double value) {
        if (value == 0.0) st.remove(i);
        else st.put(i, value);
    }
    public double get(int i) {
        if (st.contains(i)) return st.get(i);
        else return 0.0;
    }
}
```

Sparse Matrix Implementation

```java
public class SparseMatrix {
    private final int N;
    private SparseVector[] rows;

    public SparseMatrix(int N) {
        this.N = N;
        rows = new SparseVector[N];
        for (int i = 0; i < N; i++)
            rows[i] = new SparseVector(N);
    }
    public void put(int i, int j, double value) {
        rows[i].put(j, value);
    }
    public double get(int i, int j) {
        return rows[i].get(j);
    }
}
```

Sparse Vector Implementation (cont)

```java
// return a \cdot 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;
}
```

Sparse Matrix Implementation (cont)

```java
// 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;
}
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.
- Hash table A of tokens and frequencies for spam.
- Hash table B of tokens and frequencies for ham.
- Hash table C of tokens with probability p that they appear in spam.

```c
double h = 2.0 * ham.freq(word);
double s = 1.0 * spam.freq(word);
double p = (s/spams) / (h/hams + s/spams);
```


Identify incoming email as spam or ham.
- Find 15 most interesting tokens (difference from 0.5).
- Combine probabilities using Bayes law.

\[
\frac{p_1 \times p_2 \times \cdots \times p_{15}}{(1-p_1) \times (1-p_2) \times \cdots \times (1-p_{15})}
\]

- 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.