Searching Applications

Outline

Sets

Sets

A mathematical set is a collection of distinct keys

Set	
`Set()`	construct an empty set
`void add(Key key)`	add *key* to the set
`boolean contains(Key key)`	is *key* in the set?
`boolean isEmpty()`	is the set empty?

Sets

Sets

An exception filter is a program that reads in a list of words from one file and prints out all words from standard input that are (are not) in the list

Sets

An exception filter is a program that reads in a list of words from one file and prints out all words from standard input that are (are not) in the list

Example

>_ ~/workspace/dsa/programs

```
$ more list.txt
was it the of
$ more tinyTale.txt
it was the best of times it was the worst of times
...
```

>_ ~/workspace/dsa/programs

```
$ java Filter + list.txt < tinyTale.txt
it
was
the
of
...
```

>_ ~/workspace/dsa/programs

```
$ java Filter - list.txt < tinyTale.txt
best
times
worst
times
...
```

Sets

Exception filter 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

Sets

Sets

Filter.java

```
import dsa.Set;
import stdlib.In;
import stdlib.StdIn;
import stdlib.StdOut;

public class Filter {
    public static void main(String[] args) {
        String filter = args[0];
        String filename = args[1];
        boolean isWhiteFilter;
        if (filter.equals("-")) {
            isWhiteFilter = false;
        } else if (filter.equals("+")) {
            isWhiteFilter = true;
        } else {
            throw new IllegalArgumentException("Illegal command-line argument");
        }
        Set<String> set = new Set<String>();
        In in = new In(filename);
        while (!in.isEmpty()) {
            String word = in.readString();
            set.add(word);
        }
        while (!StdIn.isEmpty()) {
            String word = StdIn.readString();
            if (isWhiteFilter && set.contains(word) || !isWhiteFilter && !set.contains(word)) {
                StdOut.println(word);
            }
        }
    }
}
```

Dictionary Clients

Dictionary Clients

Dictionary lookup: given a comma-separated value (CSV) file, a key field, and a value field as command-line arguments, lookup the keys read from standard input and print the corresponding values if they are found

Dictionary Clients

Dictionary lookup: given a comma-separated value (CSV) file, a key field, and a value field as command-line arguments, lookup the keys read from standard input and print the corresponding values if they are found

Example (DNS lookup)

```
>_ ~/workspace/dsa/programs

$ more ip.csv
www.princeton.edu,128.112.128.15
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
adobe.com,192.150.18.60
...
```

```
>_ ~/workspace/dsa/programs

$ java LookupCSV ip.csv 0 1
adobe.com
192.150.18.60
www.princeton.edu
128.112.128.15
ebay.edu
Not found
<ctrl-d>
```

```
>_ ~/workspace/dsa/programs

$ java LookupCSV ip.csv 1 0
128.112.128.15
www.princeton.edu
999.999.999.99
Not found
<ctrl-d>
```

Dictionary Clients

Dictionary Clients

Example (amino acid lookup)

>_ ~/workspace/dsa/programs

```
$ 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
...
```

>_ ~/workspace/dsa/programs

```
$ java LookupCSV amino.csv 0 3
TCT
Serine
TAG
Stop
TAC
Tyrosine
<ctrl-d>
```

Dictionary Clients

Dictionary Clients

LookupCSV.java

```
import dsa.SeparateChainingHashST;
import stdlib.In;
import stdlib.StdIn;
import stdlib.StdOut;

public class LookupCSV {
    public static void main(String[] args) {
        String filename = args[0];
        int keyField = Integer.parseInt(args[1]);
        int valField = Integer.parseInt(args[2]);
        SeparateChainingHashST<String, String> st = new SeparateChainingHashST<String, String>();
        In in = new In(filename);
        while (in.hasNextLine()) {
            String line = in.readLine();
            String[] tokens = line.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(st.get(s));
            } else {
                StdOut.println("Not found");
            }
        }
    }
}
```

Indexing Clients

Indexing Clients

File index: given a list of files, create an index so that you can efficiently find all files containing a given query string

Indexing Clients

File index: given a list of files, create an index so that you can efficiently find all files containing a given query string

Example

```
>_ ~/workspace/dsa/programs

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

```
>_ ~/workspace/dsa/programs

$ java FileIndex *.txt
Indexing files
  aesop.txt
  magna.txt
  moby.txt
  sawyer.txt
  tale.txt

freedom
  magna.txt
  moby.txt
  tale.txt
whale
  moby.txt
lamb
  aesop.txt
  sawyer.txt
<ctrl-d>
```

Indexing Clients

Indexing Clients

FileIndex.java

```
import dsa.SeparateChainingHashST;
import dsa.Set;
import stdlib.In;
import stdlib.StdIn;
import stdlib.StdOut;

public class FileIndex {
    public static void main(String[] args) {
        SeparateChainingHashST<String, Set<String>> st =
                new SeparateChainingHashST<String, Set<String>>();
        for (String filename : args) {
            In in = new In(filename);
            while (!in.isEmpty()) {
                String word = in.readString();
                if (!st.contains(word)) {
                    st.put(word, new Set<String>());
                }
                Set<String> set = st.get(word);
                set.add(filename);
            }
        }
        while (!StdIn.isEmpty()) {
            String query = StdIn.readString();
            if (st.contains(query)) {
                Set<String> set = st.get(query);
                for (String filename : set) {
                    StdOut.println("  " + filename);
                }
            } else {
                StdOut.println("Query not found");
            }
        }
    }
}
```

Sparse Vectors and Matrices

Sparse Vectors and Matrices

Sparse vector: standard representation

- Constant time access to elements
- Space proportional to n

Sparse Vectors and Matrices

Sparse vector: standard representation

- Constant time access to elements
- Space proportional to n

Sparse vector: symbol table representation

- Key is index, value is entry
- Efficient iterator
- Space proportional to number of nonzeros

Sparse Vectors and Matrices

Sparse vector: standard representation

- Constant time access to elements
- Space proportional to n

Sparse vector: symbol table representation

- Key is index, value is entry
- Efficient iterator
- Space proportional to number of nonzeros

Sparse matrix: standard representation

- Each row of matrix is an array
- Constant time access to elements
- Space proportional to n^2

Sparse Vectors and Matrices

Sparse vector: standard representation

- Constant time access to elements
- Space proportional to n

Sparse vector: symbol table representation

- Key is index, value is entry
- Efficient iterator
- Space proportional to number of nonzeros

Sparse matrix: standard representation

- Each row of matrix is an array
- Constant time access to elements
- Space proportional to n^2

Sparse matrix: symbol table representation

- Each row of matrix is a sparse vector
- Efficient access to elements
- Space proportional to number of nonzeros (plus n)

Sparse Vectors and Matrices

Sparse Vectors and Matrices

SparseVector.java

```
package dsa;

import stdlib.StdOut;

public class SparseVector {
    private int n;
    private SeparateChainingHashST<Integer, Double> st;

    public SparseVector(int n) {
        this.n = n;
        this.st = new SeparateChainingHashST<Integer, Double>();
    }

    public int dimension() {
        return n;
    }

    public int size() {
        return st.size();
    }

    public void put(int i, double value) {
        if (i < 0 || i >= n) {
            throw new IllegalArgumentException("Illegal index");
        }
        if (value == 0.0) {
            st.delete(i);
        } else {
            st.put(i, value);
        }
    }

    public double get(int i) {
        if (i < 0 || i >= n) {
            throw new IllegalArgumentException("Illegal index");
```

Sparse Vectors and Matrices

SparseVector.java

```
        }
        if (st.contains(i)) {
            return st.get(i);
        }
        return 0.0;
    }

    public SparseVector plus(SparseVector other) {
        if (this.n != other.n) {
            throw new IllegalArgumentException("Dimensions disagree");
        }
        SparseVector c = new SparseVector(n);
        for (int i : this.st.keys()) {
            c.put(i, this.get(i));
        }
        for (int i : other.st.keys()) {
            c.put(i, other.get(i) + c.get(i));
        }
        return c;
    }

    public SparseVector scale(double alpha) {
        SparseVector c = new SparseVector(n);
        for (int i : this.st.keys()) {
            c.put(i, alpha * this.get(i));
        }
        return c;
    }

    public double dot(SparseVector other) {
        if (this.n != other.n) {
            throw new IllegalArgumentException("Vector dimensions disagree");
        }
        double sum = 0.0;
        if (this.st.size() <= other.st.size()) {
```

Sparse Vectors and Matrices

SparseVector.java

```
            for (int i : this.st.keys()) {
                if (other.st.contains(i)) {
                    sum += this.get(i) * other.get(i);
                }
            }
        } else {
            for (int i : other.st.keys()) {
                if (this.st.contains(i)) {
                    sum += this.get(i) * other.get(i);
                }
            }
        }
        return sum;
    }

    public double magnitude() {
        return Math.sqrt(this.dot(this));
    }

    public String toString() {
        StringBuilder sb = new StringBuilder();
        sb.append("n = " + n + ", nnz = " + size() + ", entries = ");
        for (int i : st.keys()) {
            sb.append("(" + i + ", " + st.get(i) + ") ");
        }
        return sb.toString();
    }

    public static void main(String[] args) {
        SparseVector a = new SparseVector(10);
        SparseVector b = new SparseVector(10);
        a.put(3, 0.50);
        a.put(9, 0.75);
        a.put(6, 0.11);
        a.put(6, 0.00);
```

Sparse Vectors and Matrices

SparseVector.java

```
        b.put(3, 0.60);
        b.put(4, 0.90);
        StdOut.println("a            = " + a);
        StdOut.println("b            = " + b);
        StdOut.println("a.size()     = " + a.size());
        StdOut.println("a.plus(b)    = " + a.plus(b));
        StdOut.println("b.scale(3)   = " + b.scale(3));
        StdOut.println("a.dot(b)     = " + a.dot(b));
    }
}
```

Sparse Vectors and Matrices

Sparse Vectors and Matrices

SparseMatrix.java

```
package dsa;

import stdlib.StdOut;

public class SparseMatrix {
    private int m;
    private int n;
    private SparseVector[] rows;

    public SparseMatrix(int m, int n) {
        this.m = m;
        this.n = n;
        rows = new SparseVector[m];
        for (int i = 0; i < m; i++) {
            rows[i] = new SparseVector(n);
        }
    }

    public int nRows() {
        return m;
    }

    public int nCols() {
        return n;
    }

    public int size() {
        int nnz = 0;
        for (SparseVector row : rows) {
            nnz += row.size();
        }
        return nnz;
    }

    public void put(int i, int j, double value) {
```

Sparse Vectors and Matrices

SparseMatrix.java

```
        if (i < 0 || i >= m) {
            throw new IllegalArgumentException("Illegal row index");
        }
        if (j < 0 || j >= n) {
            throw new IllegalArgumentException("Illegal column index");
        }
        rows[i].put(j, value);
    }

    public double get(int i, int j) {
        if (i < 0 || i >= m) {
            throw new IllegalArgumentException("Illegal row index");
        }
        if (j < 0 || j >= n) {
            throw new IllegalArgumentException("Illegal column index");
        }
        return rows[i].get(j);
    }

    public SparseMatrix plus(SparseMatrix other) {
        if (this.m != other.m && this.n != other.n) {
            throw new RuntimeException("Dimensions disagree");
        }
        SparseMatrix result = new SparseMatrix(m, n);
        for (int i = 0; i < m; i++) {
            result.rows[i] = this.rows[i].plus(other.rows[i]);
        }
        return result;
    }

    public SparseVector times(SparseVector x) {
        if (n != x.dimension()) {
            throw new IllegalArgumentException("Dimensions disagree");
        }
        SparseVector b = new SparseVector(m);
```

Sparse Vectors and Matrices

SparseMatrix.java

```
        for (int i = 0; i < m; i++) {
            b.put(i, rows[i].dot(x));
        }
        return b;
    }

    public String toString() {
        StringBuilder sb = new StringBuilder();
        sb.append("m = " + m + ", n = " + n + ", nnz = " + size() + "\n");
        for (int i = 0; i < m - 1; i++) {
            sb.append("  " + i + ": " + rows[i].toString() + "\n");
        }
        sb.append("  " + (m - 1) + ": " + rows[m - 1].toString());
        return sb.toString();
    }

    public static void main(String[] args) {
        SparseMatrix a = new SparseMatrix(5, 5);
        SparseVector x = new SparseVector(5);
        a.put(0, 0, 1.0);
        a.put(1, 1, 1.0);
        a.put(2, 2, 1.0);
        a.put(3, 3, 1.0);
        a.put(4, 4, 1.0);
        a.put(2, 4, 0.3);
        x.put(0, 0.75);
        x.put(2, 0.11);
        StdOut.println("A: " + a);
        StdOut.println("x: " + x);
        StdOut.println("A.plus(A): " + a.plus(a));
        StdOut.println("A.times(x): " + a.times(x));
    }
}
```