

Algorithmics 3

Data structures, Parametric Polymorphism, Collections

November 22th, 2006

It's the final countdown

We have 5 lectures left, including this one, and 4 exercise sessions.

Collections Going beyond arrays.

Run-Time Type Information Checking the type of a class during the execution.

Abstract classes Bastard child of a class and an interface.

It's the final countdown

We have 5 lectures left, including this one, and 4 exercise sessions.

Collections Going beyond arrays.

Run-Time Type Information Checking the type of a class during the execution.

Abstract classes Bastard child of a class and an interface.

Various features ? Internal classes, anonymous classes, closures, final methods.

It's the final countdown

We have 5 lectures left, including this one, and 4 exercise sessions.

Collections Going beyond arrays.

Run-Time Type Information Checking the type of a class during the execution.

Abstract classes Bastard child of a class and an interface.

Various features ? Internal classes, anonymous classes, closures, final methods.

Something you want detailed ? Anything you didn't understand ?

Multimedia ? Sounds and videos.

3D ? Probably a bit complex but we can try.

Printing ? Printing stuff from an application.

Applets ? Putting a program in a web page.

Network ? Communicating through a network.

Drawing ? Drawing curves, altering images.

UI Design ? More Swing.

Something else ?

Collections

1. Complexity
2. Going beyond arrays
3. Iterating

Introduction

Notions of complexity

Counting operations

Arrays and beyond

Arrays

Lists

Data structures

Maps

Case studies

Collections

About collections

More about collections

Practice makes perfect

Done

Looking for information

```
T myArray[];  
T someValue;  
// ...  
for(int i = 0; i < myArray.length; ++i)  
{  
    if (myArray[i] == someValue)  
        return true;  
}  
return false;
```

Looking for information

```
T myArray[];  
T someValue;  
// ...  
for(int i = 0; i < myArray.length; ++i)  
{  
    if (myArray[i] == someValue)  
        return true;  
}  
return false;
```

Q How many operations ?

Looking for information

```
T myArray[];  
T someValue;  
// ...  
for(int i = 0; i < myArray.length; ++i)  
{  
    if (myArray[i] == someValue)  
        return true;  
}  
return false;
```

Q How many operations ?

Q $\mathcal{O}(\text{myArray.length})$

Looking for information

```
T myArray [];  
T someValue;  
// ...  
for(int i = 0; i < myArray.length; ++i)  
{  
    if (myArray[i] == someValue)  
        return true;  
}  
return false;
```

Q How many operations ?

Q $\mathcal{O}(\text{myArray.length})$ – that is, if n is the length,
number of operations
 n is bounded.

Looking for information

```
T myArray [];  
T someValue;  
// ...  
for(int i = 0; i < myArray.length; ++i)  
{  
    if (myArray[i] == someValue)  
        return true;  
}  
return false;
```

Q How many operations?

Q $\mathcal{O}(\text{myArray.length})$ – that is, if n is the length,
number of operations is bounded.

This is “linear in the worst case”, “linear in the average case”, “constant time in the best case”.

Looking inside sorted information

Q How can we improve this, by assuming that `myArray` is sorted ?

Looking inside sorted information

Q How can we improve this, by assuming that `myArray` is sorted ?

A The same way you look for information inside a dictionary or a phonebook.

Looking inside sorted information

Q How can we improve this, by assuming that `myArray` is sorted ?

A The same way you look for information inside a dictionary or a phonebook.

A Dichotomy !

Looking inside sorted information

Q How can we improve this, by assuming that `myArray` is sorted ?

A The same way you look for information inside a dictionary or a phonebook.

A Dichotomy !

See Dr. Java.

Looking inside sorted information

Q How can we improve this, by assuming that `myArray` is sorted ?

A The same way you look for information inside a dictionary or a phonebook.

A Dichotomy !

See Dr. Java.

Q $\mathcal{O}(\ln(\text{myArray.length}))$

Looking inside sorted information

Q How can we improve this, by assuming that `myArray` is sorted ?

A The same way you look for information inside a dictionary or a phonebook.

A Dichotomy !

See Dr. Java.

Q $\mathcal{O}(\ln(\text{myArray.length}))$ – that is, if n is the length,
number of operations
 $\ln(n)$ is bounded.

Looking inside sorted information

Q How can we improve this, by assuming that `myArray` is sorted ?

A The same way you look for information inside a dictionary or a phonebook.

A Dichotomy !

See Dr. Java.

Q $\mathcal{O}(\ln(\text{myArray.length}))$ – that is, if n is the length, $\frac{\text{number of operations}}{\ln(n)}$ is bounded.

This is “logarithmic in the worst case”, “logarithmic in the average case”, “constant time in the best case”.

So ?

Q Which one is fastest ?

So ?

Q Which one is fastest ?

Actually, it depends on the size of the sample.

So ?

Q Which one is fastest ?

Actually, it depends on the size of the sample.

For small samples, it doesn't always matter. For large samples, it does.

Introduction

Notions of complexity

Counting operations

Arrays and beyond

Arrays

Lists

Data structures

Maps

Case studies

Collections

About collections

More about collections

Practice makes perfect

Arrays

Q What's an array ?

Arrays

Q What's an array ?

A An array of length l of elements of type t is a *data structure* with the following operations:

- ▶ if i is an integer of $[0, l[$, retrieving element i in constant time
- ▶ if i is an integer of $[0, l[$, replacing element i in constant time

Limitation of an array

Q What can't you do with an array ?

Limitation of an array

Q What can't you do with an array ?

A Essentially: resize it.

Example

Q How do you implement a spell-checking dictionary with arrays ?

Example

Q How do you implement a spell-checking dictionary with arrays ?

A Each time you meet a word, look in the whole dictionary if that word exists.

Example

Q How do you implement a spell-checking dictionary with arrays ?

A Each time you meet a word, look in the whole dictionary if that word exists.

Q How long is it to check that a word appears in the dictionary ?

Example

Q How do you implement a spell-checking dictionary with arrays ?

A Each time you meet a word, look in the whole dictionary if that word exists.

Q How long is it to check that a word appears in the dictionary ?

A $\mathcal{O}(n \times \text{String comparison})$

Example

Q How do you implement a spell-checking dictionary with arrays ?

A Each time you meet a word, look in the whole dictionary if that word exists.

Q How long is it to check that a word appears in the dictionary ?

A $\mathcal{O}(n \times \text{String comparison})$

Q How do you add new words in the dictionary ?

Example

Q How do you implement a spell-checking dictionary with arrays ?

A Each time you meet a word, look in the whole dictionary if that word exists.

Q How long is it to check that a word appears in the dictionary ?

A $\mathcal{O}(n \times \text{String comparison})$

Q How do you add new words in the dictionary ?

A Replace the array when it's full !

Example

Q How do you implement a spell-checking dictionary with arrays ?

A Each time you meet a word, look in the whole dictionary if that word exists.

Q How long is it to check that a word appears in the dictionary ?

A $\mathcal{O}(n \times \text{String comparison})$

Q How do you add new words in the dictionary ?

A Replace the array when it's full !

Q How long is it to add new words to that dictionary ?

Example

Q How do you implement a spell-checking dictionary with arrays ?

A Each time you meet a word, look in the whole dictionary if that word exists.

Q How long is it to check that a word appears in the dictionary ?

A $\mathcal{O}(n \times \text{String comparison})$

Q How do you add new words in the dictionary ?

A Replace the array when it's full !

Q How long is it to add new words to that dictionary ?

A $\mathcal{O}(n)$ operations each time

Example

Q How do you implement a spell-checking dictionary with arrays ?

A Each time you meet a word, look in the whole dictionary if that word exists.

Q How long is it to check that a word appears in the dictionary ?

A $\mathcal{O}(n \times \text{String comparison})$

Q How do you add new words in the dictionary ?

A Replace the array when it's full !

Q How long is it to add new words to that dictionary ?

A $\mathcal{O}(n)$ operations each time ($\mathcal{O}(\ln(n))$ or *logarithmic time* if you're doing it more smartly)

Alternative

Q Can you think of an alternative where

- ▶ looking is done in linear time ;
- ▶ adding is done in constant time ?

Alternative

Q Can you think of an alternative where

- ▶ looking is done in linear time ;
- ▶ adding is done in constant time ?

Really, go ahead.

Alternative

Q Can you think of an alternative where

- ▶ looking is done in linear time ;
- ▶ adding is done in constant time ?

Really, go ahead.

Let's try lists.

Lists

A list is

- ▶ either the empty list ;
- ▶ or some element, with a value, and the rest of the list – itself a list.

Lists

A list is

- ▶ either the empty list ;
- ▶ or some element, with a value, and the rest of the list – itself a list.

See Dr. Java.

Lists

A list is

- ▶ either the empty list ;
- ▶ or some element, with a value, and the rest of the list – itself a list.

See Dr. Java.

Note the `<T>`. This class is a *generic* – it's a case of *parametric polymorphism*.

Lists

A list is

- ▶ either the empty list ;
- ▶ or some element, with a value, and the rest of the list – itself a list.

See Dr. Java.

Note the `<T>`. This class is a *generic* – it's a case of *parametric polymorphism*.

More about that in OCaml.

Data structures

A data structure is a description of how to store data so that it can be used efficiently by an algorithm.

Data structures

A data structure is a description of how to store data so that it can be used efficiently by an algorithm.

We're going to study this further in OCaml.

Data structures

A data structure is a description of how to store data so that it can be used efficiently by an algorithm.

We're going to study this further in OCaml.

For now, remember that

- Variables** are very fast (access/modification in constant time), they can be shared but their size is always 1.
- Arrays** are fast (access/modification in constant time) but they can't be shared and their size can't be changed.
- Records** (= recording information within one object without methods) are fast (access/modification in constant time) but their size or structure can't be changed.
- Lists** are slower (access/modification in linear time) but they can be shared, plus you can add or remove elements (linear time) – note that if you're only interested in the first element, you have constant time.

Introducing maps

A “map” (or “dictionary”, or “associative array”) is a data structure with the following operations:

- ▶ associate a key to an element (e.g. “name” \mapsto “Teller”) ;
- ▶ using a key, remove the associated element ;
- ▶ using a key, replace the associated element ;

Introducing maps

A “map” (or “dictionary”, or “associative array”) is a data structure with the following operations:

- ▶ associate a key to an element (e.g. “name” \mapsto “Teller”) ;
- ▶ using a key, remove the associated element ;
- ▶ using a key, replace the associated element ;
- ▶ without limits on the number of elements !

Introducing maps

A “map” (or “dictionary”, or “associative array”) is a data structure with the following operations:

- ▶ associate a key to an element (e.g. “name” \mapsto “Teller”) ;
- ▶ using a key, remove the associated element ;
- ▶ using a key, replace the associated element ;
- ▶ without limits on the number of elements !

Typically, if you’re attempting to write anything remotely related to a dictionary, that’s the kind of operations you’re looking for.

Hash Tables

Hash tables are the usual implementation of maps:

A hash table with keys of type \mathcal{K} and values of type \mathcal{V} is a data structure with the following operations:

- ▶ associate a key to an element, in logarithmic time ;
- ▶ using a key, remove the associated element, in logarithmic time ;
- ▶ using a key, replace the associated element, in logarithmic time ;

Hash Tables

Hash tables are the usual implementation of maps:

A hash table with keys of type \mathcal{K} and values of type \mathcal{V} is a data structure with the following operations:

- ▶ associate a key to an element, in logarithmic time ;
- ▶ using a key, remove the associated element, in logarithmic time ;
- ▶ using a key, replace the associated element, in logarithmic time ;

The only thing you need is for \mathcal{K} to be an ordered set.

Hash Tables

Hash tables are the usual implementation of maps:

A hash table with keys of type \mathcal{K} and values of type \mathcal{V} is a data structure with the following operations:

- ▶ associate a key to an element, in logarithmic time ;
- ▶ using a key, remove the associated element, in logarithmic time ;
- ▶ using a key, replace the associated element, in logarithmic time ;

The only thing you need is for \mathcal{K} to be an ordered set.

Hash tables live in `java.util.Hashtable`.

Hash Tables

Hash tables are the usual implementation of maps:

A hash table with keys of type \mathcal{K} and values of type \mathcal{V} is a data structure with the following operations:

- ▶ associate a key to an element, in logarithmic time ;
- ▶ using a key, remove the associated element, in logarithmic time ;
- ▶ using a key, replace the associated element, in logarithmic time ;

The only thing you need is for \mathcal{K} to be an ordered set.

Hash tables live in `java.util.Hashtable`.

Internally, hash tables involve dichotomy.

Case studies

What would you use to

- ▶ ... store a text file in memory ?
- ▶ ... count the number of times each symbol appears in a file ?
- ▶ ... mark where you've walked in a maze ?
- ▶ ... represent a landmap ?
- ▶ ... count the number of unique words in a document ?
- ▶ ... keep the high scores of your minesweeper ?
- ▶ ... store correct spellings for your spell-checker ?
- ▶ ... files and directories on your hard drive ?

Case studies

What would you use to

- ▶ ... store a text file in memory ?
- ▶ ... count the number of times each symbol appears in a file ?
- ▶ ... mark where you've walked in a maze ?
- ▶ ... represent a landmap ?
- ▶ ... count the number of unique words in a document ?
- ▶ ... keep the high scores of your minesweeper ?
- ▶ ... store correct spellings for your spell-checker ?
- ▶ ... files and directories on your hard drive ?

Some of these things are definitely not easy !

Big issues

Often, the main problems one has to deal with in algorithmics are

- ▶ defining exactly what kind of result one wants
- ▶ defining exactly how data is represented
- ▶ defining the transformations on data.

Big issues

Often, the main problems one has to deal with in algorithmics are

- ▶ defining exactly what kind of result one wants
- ▶ defining exactly how data is represented
- ▶ defining the transformations on data.

Again, we'll see more of that in OCaml – some of these things are just too annoying to do in Java.

Introduction

Notions of complexity

Counting operations

Arrays and beyond

Arrays

Lists

Data structures

Maps

Case studies

Collections

About collections

More about collections

Practice makes perfect

What are collections ?

Collections are Java standard library's standard data structures, plus a few nice tools.

What are collections ?

Collections are Java standard library's standard data structures, plus a few nice tools.

Of course, as often with Java, it's just about extremely complicated for what it is.

Ok, what are collections ?

A few interfaces and classes for data structures.

Set
(Extendable) List
Map

Ok, what are collections ?

A few interfaces and classes for data structures.

Hash tables

Set

HashSet

(Extendable) List

Map

HashMap

And a few utility interfaces (essentially Iterable).

Ok, what are collections ?

A few interfaces and classes for data structures.

	Hash tables	Arrays
Set	HashSet	
(Extendable) List		ArrayList
Map	HashMap	

And a few utility interfaces (essentially Iterable).

Ok, what are collections ?

A few interfaces and classes for data structures.

	Hash tables	Arrays	Linked Lists
Set	HashSet		
(Extendable) List		ArrayList	LinkedList
Map	HashMap		

And a few utility interfaces (essentially Iterable).

Ok, what are collections ?

A few interfaces and classes for data structures.

	Hash tables	Arrays	Linked Lists	...
Set	HashSet			...
(Extendable) List		ArrayList	LinkedList	...
Map	HashMap			...

And a few utility interfaces (essentially Iterable).

What's the point ?

Q Why these classes ?

What's the point ?

Q Why these classes ?

A Because Java developers are expected not to know how to write these things. Plus it's actually quite a time-saver to already have these kinds of utilities handy.

What's the point ?

Q Why these classes ?

A Because Java developers are expected not to know how to write these things. Plus it's actually quite a time-saver to already have these kinds of utilities handy.

Q Why interfaces ?

What's the point ?

Q Why these classes ?

A Because Java developers are expected not to know how to write these things. Plus it's actually quite a time-saver to already have these kinds of utilities handy.

Q Why interfaces ?

A So that you can replace the implementation, in more complex scenarios, without changing anything else.

What's the point ?

Q Why these classes ?

A Because Java developers are expected not to know how to write these things. Plus it's actually quite a time-saver to already have these kinds of utilities handy.

Q Why interfaces ?

A So that you can replace the implementation, in more complex scenarios, without changing anything else.

Q Any idea of such a scenario ?

What's the point ?

Q Why these classes ?

A Because Java developers are expected not to know how to write these things. Plus it's actually quite a time-saver to already have these kinds of utilities handy.

Q Why interfaces ?

A So that you can replace the implementation, in more complex scenarios, without changing anything else.

Q Any idea of such a scenario ?

A Say,

- ▶ if your dictionary is kept on a distant computer
- ▶ if you want persistence for your data structure (with a database or a file)
- ▶ if your list is infinite
- ▶ if your list is read from a stream/written to a stream
- ▶ if you want your existing data structure to interact with collections
- ▶ ...

About these interfaces

- Maps
 - ▶ `put(key, value)` Add/replace an element
 - ▶ `get(key)` Recover an element from its key.
- List
 - ▶ `add(index, value)` Add an element.
 - ▶ `get(index)` Recover an element from its index.
- Set
 - ▶ `add(value)` Add an element.
 - ▶ `contains(value)` Determine if an element is in the set.

Collection comprehension

There's a nice thing you can do with collections.

Collection comprehension

There's a nice thing you can do with collections.
Look at Dr. Java

Collection comprehension

There's a nice thing you can do with collections.

Look at Dr. Java

This for system is an example of (limited) *data structure comprehension*.

Iterability

Good news: you can add this feature to your own data structures.

Iterability

Good news: you can add this feature to your own data structures.
Again, look at Dr. Java.

Iterability

Good news: you can add this feature to your own data structures.
Again, look at Dr. Java.
Note that it's not a feature, just a nice, easy-to-read, notation.

Iterability

Good news: you can add this feature to your own data structures.

Again, look at Dr. Java.

Note that it's not a feature, just a nice, easy-to-read, notation.

It's also a good idea whenever you create a new data structure to create an iterator for that structure, as it saves people from the trouble of understanding how the data structure works before being able to use it.

Let's build a tree

Let's define an iterator for trees.

Let's build a tree

Let's define an iterator for trees.

Q When is depth-first search better ? When is breadth-first search better ?

Let's build a tree

Let's define an iterator for trees.

Q When is depth-first search better ? When is breadth-first search better ?

A Essentially, depth-first search takes less memory – but won't work on infinite structures.

Let's build a tree

Let's define an iterator for trees.

Q When is depth-first search better ? When is breadth-first search better ?

A Essentially, depth-first search takes less memory – but won't work on infinite structures.

Both can be extended to graphs.

Run-time length encoding

Typically, on a computer, all characters are represented by a sequence of 8 0s and 1s.

Run-time length encoding

Typically, on a computer, all characters are represented by a sequence of 8 0s and 1s.

However, “e”, “m” and “n” are the most common letters in English, while “;”, “\” or “&” are much less common.

Run-time length encoding

Typically, on a computer, all characters are represented by a sequence of 8 0s and 1s.

However, “e”, “m” and “n” are the most common letters in English, while “;”, “\” or “&” are much less common.

Perhaps we could use less bits for most common letters and more bits for less common letters ?

Run-time length encoding

Typically, on a computer, all characters are represented by a sequence of 8 0s and 1s.

However, “e”, “m” and “n” are the most common letters in English, while “;”, “\” or “&” are much less common.

Perhaps we could use less bits for most common letters and more bits for less common letters ?

If e , m and n are used 100 times each, while $;$, $\&$ and \backslash are only used 10 times each, the original file uses 880 bits. If we can use 2 bits for e , m and n and 16 bits for $;$, $\&$ and \backslash , the compressed file will only use 360 bits.

Run-time length encoding

Typically, on a computer, all characters are represented by a sequence of 8 0s and 1s.

However, “e”, “m” and “n” are the most common letters in English, while “;”, “\” or “&” are much less common.

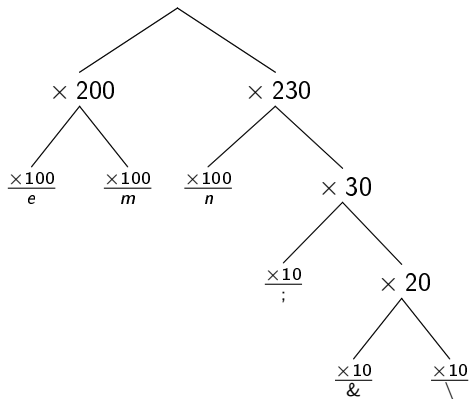
Perhaps we could use less bits for most common letters and more bits for less common letters ?

If e , m and n are used 100 times each, while $;$, $\&$ and \backslash are only used 10 times each, the original file uses 880 bits. If we can use 2 bits for e , m and n and 16 bits for $;$, $\&$ and \backslash , the compressed file will only use 360 bits.

Huffman compression is all about finding out a good representation for each character, so as to save space.

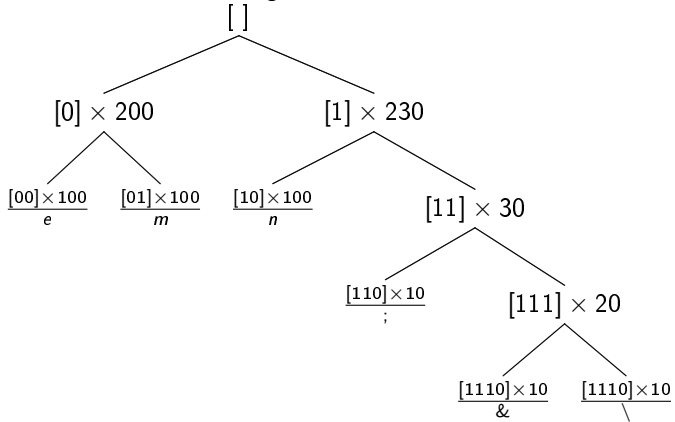
Huffman

The idea is the following



Huffman

The idea is the following



In Java

Q So, how would you implement that ?

In Java

Q So, how would you implement that ?

A Let's look at some source code.

Introduction

Notions of complexity

Counting operations

Arrays and beyond

Arrays

Lists

Data structures

Maps

Case studies

Collections

About collections

More about collections

Practice makes perfect

Enough for today

- ▶ Think about the final countdown.
- ▶ Revision exercises are available for anyone who wants them.
- ▶ You can return some of these exercises to get bonus points on your notes.