Arrays

Johan Montelius

KTH

VT26

Data structures

- Primitive : integers, float, characters, boolean, identifiers, ... non-decomposable
- Compund : arrays, tuples, records, objects, graphs, ... composed of other data stuctures

Data structures

- Primitive : integers, float, characters, boolean, identifiers, ... non-decomposable
- Compund : arrays, tuples, records, objects, graphs, ... composed of other data stuctures

Data structures

- Primitive : integers, float, characters, boolean, identifiers, ... non-decomposable
- Compund : arrays, tuples, records, objects, graphs, ... composed of other data stuctures

Data structures

- Primitive : integers, float, characters, boolean, identifiers, ... non-decomposable
- Compund : arrays, tuples, records, objects, graphs, ... composed of other data stuctures

Abstract data structures

- An abstract data structure is defined by the operations we can perform on the structure.
- The opertions should be well defined and also implemented giving a predictable requirement of space and time.
- Exactly how the structure is implemented is up to the provider and should not be assumed by the user.
- Examples: array, set, vector, sequence, list, stack, tree, queue, priority queue, lock, map, ...

A *concrete data structure* is an description of how the structure is implemented in memory. We're talkning about memory layout, pointers, encoding of values etc.

Abstract data structures

- An abstract data structure is defined by the operations we can perform on the structure.
- The opertions should be well defined and also implemented giving a predictable requirement of space and time.
- Exactly how the structure is implemented is up to the provider and should not be assumed by the user.
- Examples: array, set, vector, sequence, list, stack, tree, queue, priority queue, lock, map, ...

A *concrete data structure* is an description of how the structure is implemented in memory. We're talkning about memory layout, pointers, encoding of values etc.

Abstract data structures

- An abstract data structure is defined by the operations we can perform on the structure.
- The opertions should be well defined and also implemented giving a predictable requirement of space and time.
- Exactly how the structure is implemented is up to the provider and should not be assumed by the user.
- Examples: array, set, vector, sequence, list, stack, tree, queue, priority queue, lock, map, ...

A *concrete data structure* is an description of how the structure is implemented in memory. We're talkning about memory layout, pointers, encoding of values etc.

Abstract data structures

- An abstract data structure is defined by the operations we can perform on the structure.
- The opertions should be well defined and also implemented giving a predictable requirement of space and time.
- Exactly how the structure is implemented is up to the provider and should not be assumed by the user.
- Examples: array, set, vector, sequence, list, stack, tree, queue, priority queue, lock, map, ...

A *concrete data structure* is an description of how the structure is implemented in memory. We're talkning about memory layout, pointers, encoding of values etc.

Abstract data structures

- An abstract data structure is defined by the operations we can perform on the structure.
- The opertions should be well defined and also implemented giving a predictable requirement of space and time.
- Exactly how the structure is implemented is up to the provider and should not be assumed by the user.
- Examples: array, set, vector, sequence, list, stack, tree, queue, priority queue, lock, map, ...

A *concrete data structure* is an description of how the structure is implemented in memory. We're talkning about memory layout, pointers, encoding of values etc.

Abstract data structures

- An abstract data structure is defined by the operations we can perform on the structure.
- The opertions should be well defined and also implemented giving a predictable requirement of space and time.
- Exactly how the structure is implemented is up to the provider and should not be assumed by the user.
- Examples: array, set, vector, sequence, list, stack, tree, queue, priority queue, lock, map, ...

A *concrete data structure* is an description of how the structure is implemented in memory. We're talkning about memory layout, pointers, encoding of values etc.

Abstract data structures

- An abstract data structure is defined by the operations we can perform on the structure.
- The opertions should be well defined and also implemented giving a predictable requirement of space and time.
- Exactly how the structure is implemented is up to the provider and should not be assumed by the user.
- Examples: array, set, vector, sequence, list, stack, tree, queue, priority queue, lock, map, ...

A *concrete data structure* is an description of how the structure is implemented in memory. We're talkning about memory layout, pointers, encoding of values etc.

An array

- Abstract: `get(index)` and `put(index, value)` operations in constant time, size proportional to number of items.
- Concrete: a consecutive segment of memory where each item can be indexed using the base adress.

An array

- Abstract: `get(index)` and `put(index, value)` operations in constant time, size proportional to number of items.
- Concrete: a consecutive segment of memory where each item can be indexed using the base adress.

An array

- Abstract: `get(index)` and `put(index, value)` operations in constant time, size proportional to number of items.
- Concrete: a consecutive segment of memory where each item can be indexed using the base adress.

An array

- Abstract: `get(index)` and `put(index, value)` operations in constant time, size proportional to number of items.
- Concrete: a consecutive segment of memory where each item can be indexed using the base adress.

in Java

```
public static void primes() {
  int[] p = new int[]{1,2,3,5,7,11,13,17,19,23};

  for (int i = 0; i < 10; i++) {
     System.out.println(" prime: " + p[i]);
  }

}
```

in C

```
void primes() {
  int p[] = {1,2,3,5,7,11,13,17,19,23};

  for (int i = 0; i < 10; i++) {
     printf(" prime: %d\n", p[i]);
  }

}
```

Accessing an array

How long time does it take ...

- .. to read or write an element in an array?
- .. to search for an element in an array?
- .. to find common elements in two arrays?

When the size of the array doubles?

Accessing an array

How long time does it take ...

- .. to read or write an element in an array?
- .. to search for an element in an array?
- .. to find common elements in two arrays?

When the size of the array doubles?

Accessing an array

How long time does it take ...

- .. to read or write an element in an array?
- .. to search for an element in an array?
- .. to find common elements in two arrays?

When the size of the array doubles?

Accessing an array

How long time does it take ...

- .. to read or write an element in an array?
- .. to search for an element in an array?
- .. to find common elements in two arrays?

When the size of the array doubles?

Accessing an array

How long time does it take ...

- .. to read or write an element in an array?
- .. to search for an element in an array?
- .. to find common elements in two arrays?

When the size of the array doubles?

Accessing an array

How long time does it take ...

- .. to read or write an element in an array?
- .. to search for an element in an array?
- .. to find common elements in two arrays?

When the size of the array doubles?

how hard can it be?