Data Structures and Algorithms

Lecture 8

For a long time it puzzled me how something so expensive, so leading edge, could be so useless, and then it occurred to me that a computer is a stupid machine with the ability to do incredibly smart things, while computer programmers are smart people with the ability to do incredibly stupid things. They are, in short, a perfect match. -- Bill Bryson

Hash Functions

- A **hash function** is any algorithm that maps data of arbitrary length to data of a fixed length. The values returned by a hash function are called **hash values**, **hash codes**, **hash sums**, checksums or simply **hashes**
- In order for an algorithm to be a good hashing function, it needs to have certain properties:
  - Like any function, it must return the same output if applied to the same input value
  - Perfect hash functions are injective, if applied to different inputs, the outputs will be different as well
  - However, usual hash functions are NOT perfect – if two different inputs have the same output value, this is called a hash collision
- Hash functions are extremely useful and used in several application domains. Most importantly:
  - Searching
  - Cryptography (MD5, SHA)

Hash Functions (cont'd)

- Hashing a string can be done by adding all ASCII codes together modulo 255 (1 byte) – only 256 possible hash values means A LOT of hash collisions;
- Another option would be to consider the hash value on 32 bits, then aligning the characters four by four, then adding them together:

The brown dog jumps over the lazy fox.

```
54 68 65 20 62 72 6F 77 6E 20 66 6F 78 20
6A 75 6D 70 73 20 6F 76 65 72 20 74 68 65
20 6C 61 7A 79 20 64 6F 67 2E
```

```
54 68 65 20
62 72 6F 77
6E 20 66 6F
78 20 6A 75
6D 70 73 20
6F 76 65 72
20 74 68 65
20 6C 61 7A
79 20 64 6F
67 2E
```

```
98 2E A9 CA
```

Hash Functions (cont'd)

- A very simple hash function for a struct construct holding data can be the actual memory address of that struct;

- Question: what's the size of the hash code in this case?
- Question: if two structures hold the same data, but are stored at different addresses, will they have the same hash code?

Sets

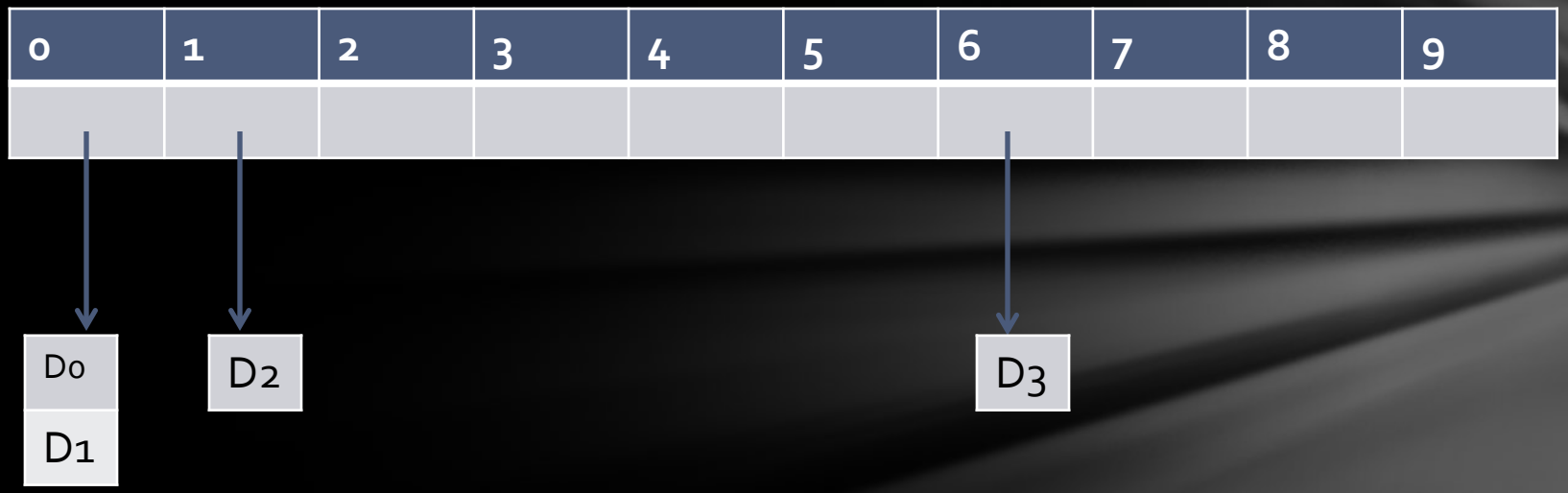
- Sets are data structures that can hold elements, with the condition that no two elements can be equal.

- Sets can be implemented in several different ways:
  - Using lists and comparing all elements each time a new node is added
  - Using arrays and comparing all elements each time a new cell is added
  - Using arrays, hash functions and lists, comparing elements only on hash collisions
  - Using binary trees

Hash Sets

- A set implemented with hash functions uses an array to store lists.
- When an element is added to the hash set, its hash code is computed and that value is the exact position in the array where the element needs to be stored.
- If the list at that position already has an element, that means either that the element is already in the set, or that there has been a hash collision;
- In this case, the range of hash codes needs to be large enough to handle the application demands, but small enough not to need a lot of memory for the array.

Hash Sets (cont'd)



- D0, D1, D2 and D3 are data stored in this set

Maps

- Maps are data structures that associate a KEY with a VALUE, where a key can only appear once in a map.
- Technically, a map is actually a set of keys that also carry a value (or payload)

- Maps can be implemented in several different ways:
  - Using lists and comparing all keys each time a new value is added
  - Using arrays and comparing all keys each time a new value is added
  - Using arrays, hash functions and lists, comparing keys only on hash collisions
  - Using trees

Exercises

- Create the header file and implementation file for a hash set that will store generic data structures identified by a pointer; What can we use for the hash function?

- Create the header file and implementation file for a hash set that will store strings; What can we use as a hash function?

- What is the difference between the equal function for the two implementations above?

- What is the time complexity for adding a new value and searching for a value in the set in the best case scenario (no collisions)?

Thank you!