An Introduction to Hash Tables

Shaun Kaufmann
Farm Credit Canada

Farm Credit Canada
Advancing the business of agriculture

Canada

Agenda

This presentation will address four main questions:

- What are hashes and how do they work?
- What you can use them for?
- What kind of performance gains can hashes provide?
- How do I use hashes in SAS?

Before we start…

Before we introduce hashes (hash tables), there are three concepts that are helpful to facilitate understanding:

- What is a data structure?
- What is the Program Data Vector?
- What is an object in regard to object oriented programming?

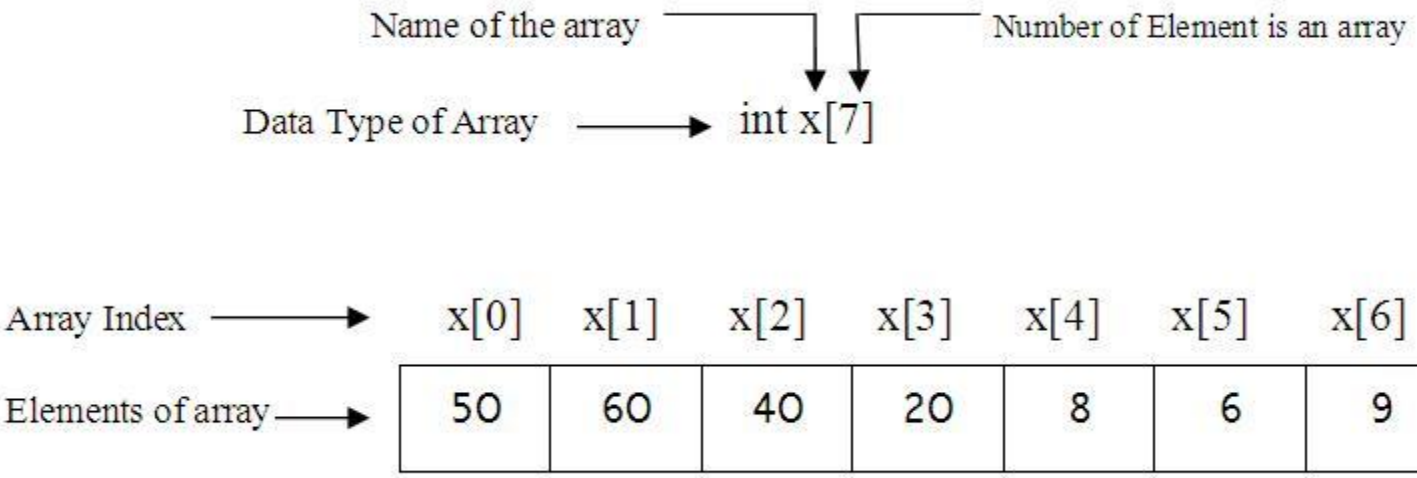
Data Structures

• a **data structure** is a particular way of storing and organizing data in a computer so that it can be used efficiently.

• A data structure can be thought of as being is constructed from **data types**. (numeric and character).

• The simplest data structure is the **one-dimensional (linear) array**, in which stored elements are numbered with consecutive integers and contents are accessed by these numbers.

One Dimensional Array



Program Data Vector

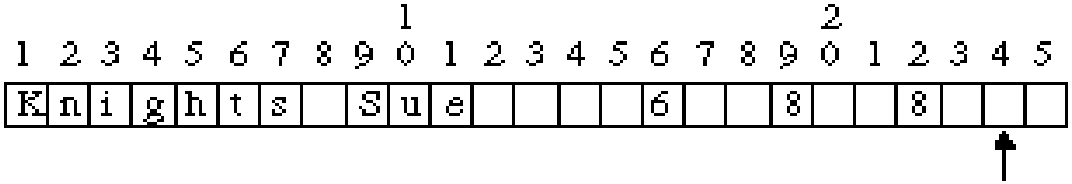
- The program data vector is a storage place in memory that contains all of the variables encountered by the data step.

- When the program runs, the program data vector contains the observation currently being processed.

- The PDV is important in the context of hashing as it is the mechanism through which values are stored in and retrieved from the hash table.

PDV Example

Input Buffer



Program Data Vector

TeamName	ParticipantName	Event1	Event2	Event3	TeamTotal	_N_	_ERROR_
Knights	Sue	6	8	8	0	1	0
Drop						Drop	Drop

Objects

• Software objects are conceptually similar to real-world objects. (a dog, a toaster, you).

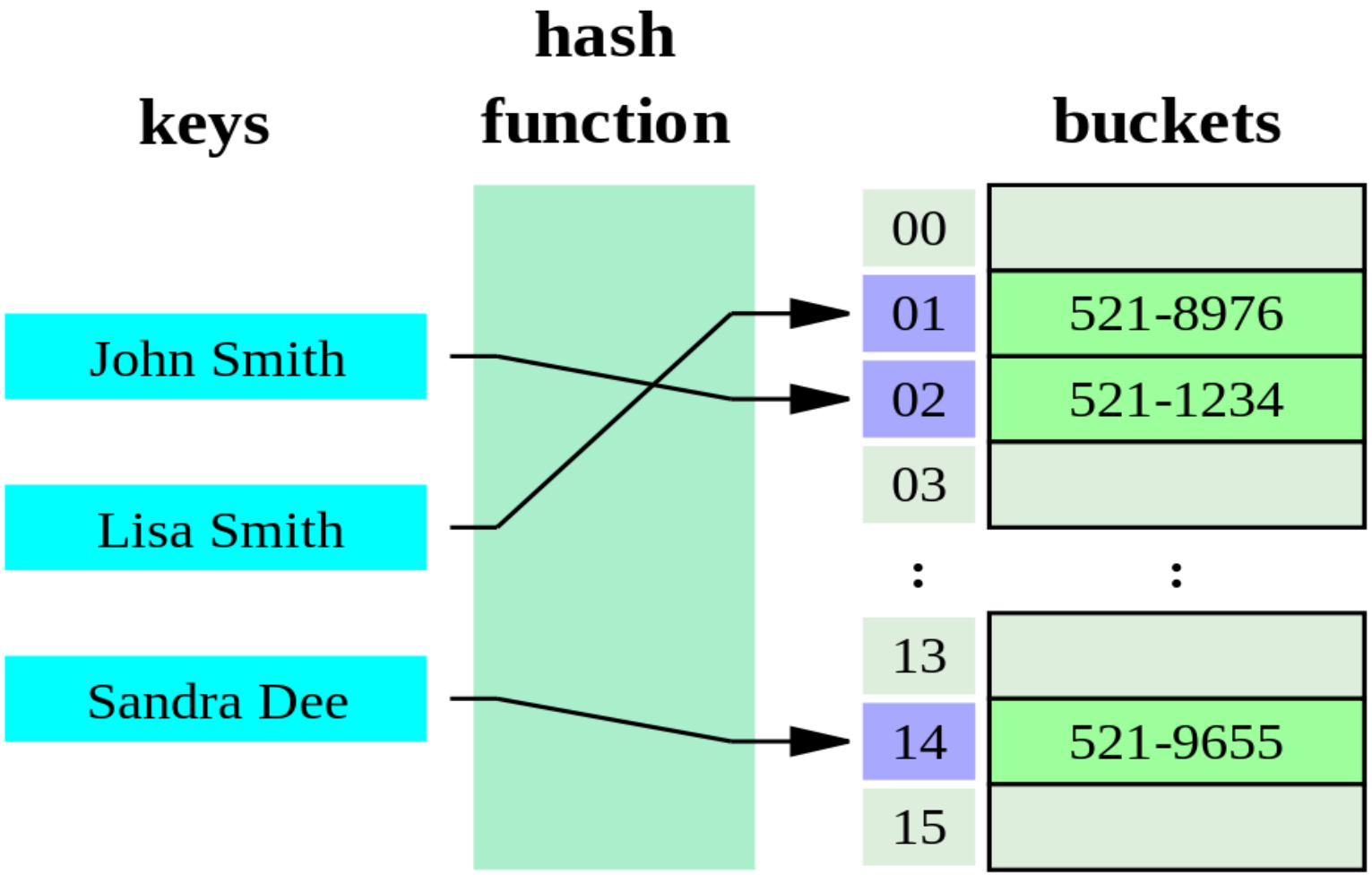
• Objects consist of things that they can do, called methods, and information about their current state, called properties or attributes.

• Methods and properties are referenced thought “dot” notation.

• dog.bark() or color = dog.fur_color;

What Are Hash Tables

- A hash table is a data structure.
- Resides in memory, not on disk like a table.
- Can be thought of as a type of array.
- More specifically, a hash table implements an associative array that maps keys to values.
- This mapping is accomplished through a hash function.
- For example, you could map names to phone numbers….


keys
hash function
buckets
John Smith
Lisa Smith
Sandra Dee
00
01
02
03
13
14
15
521-8976
521-1234
521-9655

Performance

• *For an appropriately implemented hash data structure*, the cost of lookups is independent of the number of elements stored in the table.

• Stated more simply, it doesn't take any longer to look up a value in a hash table with 10 million entries than it does to look up a value in a hash table with 1000 entries.

• Just to be completely clear, lookup time is constant and does not grow as the number of elements grow…***This is a big deal.*** This is why we use hash tables.

Uses for Hash Tables

- You can use hashes as a lookup table.

- You can use hashes for sorting.

- You can use it to merge datasets. Gregg Snell presented a paper at SUGI 31 comparing different merge methods and found hash tables to be the fastest.

- You can use hashes to access data outside of SAS. Because data stored in RDBMS can not be sorted and indexed, it is often not possible to use a data step with a merge statement.

    - John Blackwell's NESUG2010 paper sites an example where an extract, sort and merge resulted in a duration of 30 minutes. In comparison the an implementation using a hash table took about 3 minutes.

Uses for Hash Tables (cont.)

- For complex manipulations requiring continuous updating of intermediate results.
  - From personal experience, one such process resulted in a significant increase in performance:
    - 3 hours for 30,000 records (implemented in SAS without hashes)
    - 3 hours for 300,000 record (implemented in PL/SQL in Oracle on a much more powerful server – without hashes).
    - 30 seconds for 300,000 records (implemented in SAS using hashing)

- Note. The non-hash based implementation used database select and update statements.

SAS Hash Table Implementation

- Hashes are implemented in SAS as Objects
  - They have methods and properties and they use DOT notation
  - For example MyHash.find(); or This_Hash.definekey('empid');
- Hashes can only be used in a Data Step procedure or DS2 program.
- In SAS the user is shielded from most of the implementation details.
  - You don't need to know anything about the hash function used or collision resolution approaches.
  - You can trust that the hash function is appropriately implemented.

Setting up hashes in SAS

- **Hashes need to be declared.**
  - Declare hash *customer_hash*();
  - Declare hash *customer_hash*(dataset: 'mydata.customers');
- **The key(s) need to be defined:**
  - *customer_hash.*defineKey("Customer_Number");
- **The data items need to be defined:**
  - *customer_hash*.defineData("First_Name", "Last_Name");
- **The Definedone() method is called.**
  - *customer_hash*.defineDone();

Setting up hashes in SAS (cont.)

- The declaration and definition is done only once.
  - Can be accomplished using **If _n_ = 1 then do;** in a Data Step
  - Or in the init() method of a DS2 programs
- The Key and Data items defined on the previous slide **are not automatically added** to the PDV when defined as part of the hash definition. You must use a length statement to define and initialize the key and data variables in the PDV.
  - Length Customer_Number $ 10 First_Name $ 20 Last_Name $ 30;

SAS Hash Declaration Options

- **declare hash obj(dataset:** 'dataset_name', **duplicate**: 'replace' | 'error', **hashexp**: n, ordered: 'a' | 'd' | 'no', **suminc**: 'count_var'**)**;
    - **dataset**: loads the hash object from a data set.
    - **duplicate**: controls how duplicate keys are handled when loading from a data set.
    - **hashexp**: n declares 2 to the exponent n slots for the hash object.
    - **ordered**: specifies a key sort order when using a hash iterator or the output method.
    - **suminc**: count_var contains the increment value for a key summary that is retrieved by the sum method.
    - **multidata**: specifies whether multiple data items are allowed for each key.
- Note: multidata is a SAS 9.3 feature, as are the associated HAS_NEXT, FIND_NEXT, HAS_PREV, FIND_PREV methods.

SAS Hash Methods

- rc = **obj.defineKey**('key_var1', …, 'key_varN');
  - Defines a set of hash object keys given by key_var1…key_varN.
- rc = **obj.defineData**('data_var1', …, 'data_varN');
  - Defines data, given by data_var1…data_varN, to be stored in the hash object.
- rc = **obj.defineDone**();
  - Indicates that key and data definitions are complete.

SAS Hash Methods (cont.)

- rc = **obj.add**();
- rc = **obj.add(**key: key_val1,…, key: key_valN, data: data_val1,…, data: data_valN**)**;

  •Adds the specified data associated with the given key to the hash object.

- rc = **obj.find**();
- rc = **obj.find**(key: key_val1, …, key: key_valN);

  •Determines whether the given key has been stored in the hash object. If it has, the data variables are updated and the return code is set to zero. If the key is not found, the return code is non-zero.

SAS Hash Methods (cont.)

- rc = **obj.replace**();
- rc = **obj.replace(key**: key_val1,…, key: key_valN, data: data_val1, …, data: data_valN);

  - Replaces the data associated with the given key with new data as specified in data_val1…data_valN.

- rc = **obj.check**();
- rc = **obj.check(key**: key_val1, …, key: key_valN);

  - Checks whether the given key has been stored in the hash object. The data variables are not updated. Return codes are the same as for find.

SAS Hash Methods (cont.)

- rc = **obj.remove**();
- rc = **obj.remove**(key: key_val1, …, key: key_valN);
    - Removes the data associated with the given key.

- rc = **obj.clear**();
    - Removes all entries from a hash object without deleting the hash object.

SAS Hash Methods (cont.)

- rc = **obj.output**(dataset: 'dataset_name');
  - Creates dataset dataset_name which will contain the data in the hash object.

- rc = **obj.sum**(sum: sum_var);
- rc = **obj.sum**(key: key_val1, …, key: key_valN, sum: sum_var);
  - Gets the key summary for the given key and stores it in the DATA Step variable sum_var. Key summaries are incremented when a key is accessed.

SAS Hash Methods (cont.)

- rc = **obj.ref**();
- rc = **obj.ref**(key: key_val1, …, key: key_valN);

  - Performs a find operation for the current key. If the key is not in the hash object, it will be added.

- rc = **obj.equals**(hash: 'hash_obj', result: res_var);

  - Determines if two hash objects are equal. If they are equal, res_var is set to 1, otherwise it is set to zero.

- rc = **obj.delete**();

  - Deletes the hash object.

SAS Hash Attributes

- i = **obj.num_items**;
  - Retrieves the number of elements in the hash object.
- sz = **obj.item_size**;
  - Obtains the item size, in bytes, for an item in the hash object.

The Hash Iterator

• Gives you the ability to traverse your memory table from start to end, or vice versa.

•A hash iterator is associated with a specific hash object and operates only on that hash object.

•Before you declare your iterator you must declare your hash object.

Hash Iterator Methods

- declare **hiter iterobj**('hash_obj');
  - Creates a hash iterator to retrieve items from the hash object named hash_obj.
- rc = **iterobj.first**();
  - Copies the data for the first item in the hash object into the data variables for the hash object.
- rc = **iterobj.last**();
  - Copies the data for the last item in the hash object into the data variables for the hash object.

Hash Iterator Methods (cont.)

- rc = **iterobj.next**();
  - Copies the data for the next item in the hash object into the data variables for the hash object. A non-zero value is returned if the next item cannot be retrieved.
  - Use iteratively to traverse the hash object and return the data items in key order. If first has not been called, next begins with the first item.
- rc = **iterobj.prev**();
  - Copies the data for the previous item in the hash object into the data variables for the hash object. A non-zero value is returned if the next item cannot be retrieved.
  - Use iteratively to traverse the hash object and return the data items in reverse key order. If last has not been called, prev begins with the last item.

Summary

•Hash Tables are in memory data structures.

•Hash Tables can be used for lookups, sorting, merging and to facilitate complex data manipulations by removing the disk I/O associated with frequent query and update statements.

•Hash Tables can provide significant performance gains in certain circumstances.

•Hash tables are implemented in SAS as objects and provide a wide range of functionality through their methods and properties.

References

Think FAST! Use Memory Tables (Hashing) for Faster Merging. SUGI 31 Paper 244-31
Gregg P. Snell, Data Savant Consulting, Shawnee, KS

Find() the power of Hash - How, Why and When to use the SAS® Hash Object.
John Blackwell, NESUG 2010.

SAS Hash Object Programming Made Easy. Michele M. Burlew. SAS Press 2012.

Better Hashing in SAS 9.2. Robert Ray and Jason Secosky. SAS Global Forum 2008.

Table Look-Up by Direct Addressing: Key-Indexing–Bitmapping–Hashing.
Paul M. Dorfmann

References

LetITLearn.com

http://www.letitlearn.com/wp-content/uploads/2012/07/Array1.jpg

SAS Support Documentation.

http://support.sas.com/documentation/cdl/en/lrcon/62955/HTML/default/images/fig19-4.gif

SAS 9 – Hash Object Tip Sheet.

http://support.sas.com/rnd/base/datastep/dot/hash-tip-sheet.pdf

Farm Credit Canada
Advancing the business of agriculture

Canada