2 Fundamentals of data structures

2.6 Hash tables

Learning objectives:

- *Be familiar with the concept of a hash table and its uses.*
- *Be able to apply simple hashing algorithms.*
- *Know what is meant by a collision and how collisions are handled using rehashing.*

2.6.1 Hash tables

Tables

Using a table to store records

A table in computer science is a data structure of rows and columns, an example is shown in *Table 2.6.1.1*. This table consists of 4 rows of data in three columns, labelled `ULN, Forename, Surname`. Each row stores a single record of three fields containing data for an individual student as follows:

- student's unique learner number `(ULN)` consisting of eight digits, e.g. 34567890
- Forename
- Surname

An individual record is uniquely identified by its key field, `ULN`.

The rows of this table are indexed with the first row that contains a student record being labelled with index 0, the second with index 1, and so on.

	ULN	Forename	Surname
0	34567890	Fred	Bloggs
1	90002789	Mary	Smith
2	74432167	Ahmed	Khan
3	24567813	Sarah	White

Table 2.6.1.1 Student records stored in a table

This table will occupy a part of the computer's RAM (main memory). It can also be stored permanently in backing store or secondary storage, e.g. magnetic disk. However, to be searched or manipulated, it must first be copied from secondary store to RAM.

Searching the table for a record

The table could be searched for a particular record by starting at the row labelled with index 0 and scanning the entries in turn until the record is found if it is present, or the end of the table is reached. This is known as linear search which is one of several ways that an existing record can be 'looked up'.

Inserting a new record into the table

Table 2.6.1.2 shows a table similar to *Table 2.6.1.1* but this table has three empty rows following the four rows of data. A new record could be inserted in the first empty row, a second new record in the next row and so on until the table is full.

	ULN	Forename	Surname
0	34567890	Fred	Bloggs
1	90002789	Mary	Smith
2	74432167	Ahmed	Khan
3	24567813	Sarah	White
4			
5			
6			

Table 2.6.1.2 Student records stored in a table with room for new records

Deleting a record in the table

The row containing the student record to be deleted is located by searching from row 0. Once found, the data in the row is deleted. To avoid gaps appearing in the table, the occupied rows following this row are moved up to remove the gap.

Key concept

Hash table:
The table gets its name from the method used to determine the row to use.
The hash value generated by applying a hash function to a key is the table index where the record with this key should be stored if the row is free.

Limitations of this type of table and table access

A problem surfaces with the operations of searching, inserting and deleting described above when the table contains a large number of records, e.g. 10,000. It just takes too much of the computer's time to perform these operations. One solution is to use a hash table based on a well-chosen hash function.

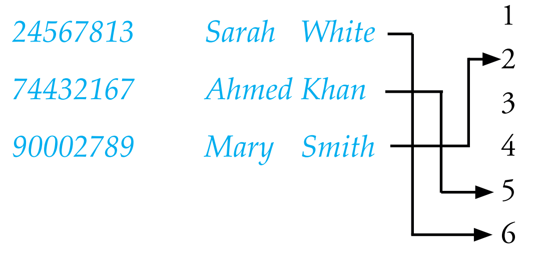
Hash table

A hash table resembles an ordinary table as described above but differs in the method used to access the rows of the table.

A row of a hash table is accessed directly when looking up, inserting and deleting a record, i.e. it does not start from row 0 every time but instead goes directly to the required row. Movement of records when deleting a record is also eliminated.

Table 2.6.1.3 shows a hash table that has gone from being empty to containing 3 records located in three different rows with indices, 2, 5, and 6, respectively.

as input data.



	ULN	Forename	Surname
0			
1			
2	90002789	Mary	Smith
3			
4			
5	74432167	Ahmed	Khan
6	24567815	Sarah	White

Table 2.6.1.3 Hash table storing three student records

The table gets the name **hash** because of the method used to generate the address or row number. A randomising function called a **hash function** is applied to the record's key, in this case the 8-digit unique learner number or `ULN`, to map the possible 8-digit `ULN` values into a much smaller range of values, the possible row numbers. This process is known as hashing.

If the `ULN` values were used directly as specifiers of row addresses we would have to accommodate addresses covering all possible values of an 8-digit number, 10^8 addresses in total, even though only a small subset of `ULN`s might be required, e.g. those used in a particular school.

For ease of understanding, the number of rows for the table in *Table 2.6.1.3* has been made small intentionally at seven, and labelled 0, 1, 2, ..., 5, 6.

Hash function

The hash function takes as input the record's key (hash key) and outputs the row address of the row for this record. The output is called the hash value or hash.

In our example, the hash value ranges from 0 to 6 for the seven rows of the given table. A hash function, `H`, that will map 8-digit `ULN`s to the set {0, 1, 2, ..., 5, 6} is shown below

```
H(ULN) = ULN Mod 7
```

Mod is the modulo arithmetic operator which calculates the remainder after integer division (see *Chapter 1.1.3*).

Table 2.6.1.4 shows three possible values of `ULN` being mapped to 2, 5 and 6 respectively e.g. 90002789 when divided by 7 gives 12857541 with a remainder of 2.

ULN	H(ULN)
90002789	2
74432167	5
24567815	6

Table 2.6.1.4 Some hash values produced by hash function H applied to ULN keys.

Questions

1 Calculate `H(ULN)` for the following `ULN`s
(a) 31258745 (b) 62517493 (c) 49981627
Hint: The scientific mode of Microsoft Windows calculator has a `Mod` operator.

Key concept

Hash function:
Is a function H, applied to a key k, which generates a hash value H(k) of range smaller than the domain of values of k,
e.g.
H : {00000000..99999999}
$\rightarrow$ {0..6}

Key concept

Hash key:
Is the key that the hash function is applied to.

Key concept

Hashing:
The process of applying a hash function to a key to generate a hash value.

Information

The term "hash" originates by analogy with its non-technical meaning, to "chop and mix". Hash functions often "chop" the input domain into many sub-domains that get "mixed" e.g. add the first three digits of the key, add the last three digits, concatenate the two resulting digit strings then map into the output range by applying modulo N.

Simple hashing functions

Hashing and hash tables are a way that memory locations for records can be assigned so that records can be retrieved quickly.

A hashing function must be relatively quick to compute whilst at the same time generating an even spread of values for the given inputs, the keys.

Another way that the latter can be expressed is that each hash value generated by the hashing function should be equally probable.

Achieving this depends on both the particular key values being hashed, and the particular hash function employed.

The value of N in modulo N is chosen to be prime because this can contribute to producing an even spread of hash values.

One simple hash function that attempts to achieve these objectives, sums the squares of the ASCII codes of each character of `Key`, as shown in *Figure 2.6.1.1* in pseudo-code.

The `Ord` function returns the ASCII code of a given character,

e.g. `Ord('A') = 65`.

The individual characters of `Key` are accessed using array indexing starting at 0, e.g. `Key[0]` accesses the first character in the string.

The algorithm generates hash values in the range 0 ... 522 because `Sum` is Modded with 523, a prime number.

Suppose that `Key` stores a string, then the steps to convert `Key` into a storage-address returned in Hash is as follows:

```
Sum ← 0
For i ← 1 To Length(Key)
  Sum ← Sum + Ord(Key[i]) * Ord(Key[i])
Endfor
Hash ← Sum Mod 523
```

Figure 2.6.1.1 Hashing algorithm that calculates a storage address in range 0 to 522

Looking up a record in a hash table

A record with a given key can be looked up by just calculating the hash of its key and checking the associated storage location.

English-French dictionary example

In this example, English words and their French equivalents are stored in records in a hash table, `HashTable`, using a hashing function, `H`, based on the hashing algorithm shown in *Figure 2.6.1.1*. Each record must have a key field which uniquely identifies the record. In this case, the key is the English word.

The hashing function, `H`, assigns hash table memory location `H(k)` to the record with key, `k`.

In our English-French dictionary example, `H(k)` could be `H('BEACH')` where `k = 'BEACH'` for the record containing the English word `'BEACH'` and the equivalent French word `'PLAGE'`.

The storage structure, `HashTable`, that will be used with this address has the following data structure:

```
THashTableArray = Array[0..522] Of TRecord
```

Where the data structure `TRecord` is defined as follows

```
TRecord = Record
            EnglishWord : String
            FrenchWord : String
          End
```

Questions

2 Calculate `H(k)` for the following keys, `k`

(a) PEN (b) CAT (c) NOW (d) WON

(ASCII codes for the characters `'A'...'Z'` map to the range 65 ... 90 - see Unit 2 Chapter 5.5)

Programming tasks

1 Write a program to store English words and their French equivalents in a hash table which is an array or its equivalent with addresses in range 0 to 522. The English word and its French equivalent should be stored together in a record or equivalent data structure at an address which is calculated by the hashing function, `H`, described above. The table should be initialised so that every key field stores the string `'-1'` to indicate that this field's record has yet to be used to store an English-French word pair. Use your program to temporarily store the English words, PEN, CAT, NOW and their French equivalents.
(English word with its French equivalent:
PEN – PLUME, CAT – CHAT, NOW – MAINTENANT)

2 Extend your program so that after storing the English-French word pairs for PEN, CAT and NOW, the program uses the hashing function, `H`, to retrieve the French equivalent when the user enters PEN, CAT or NOW. Use a loop to enable the user to continue to look up the French equivalent until the user decides otherwise.

Key concept

Collision:
A collision occurs when two or more different keys hash to the same hash value. For the hash table this means a hash value of a location in the hash table which is already occupied.

Collisions

The hash values calculated in Questions 2(c) and 2(d) are identical because the English words contain the same letters, but arranged in a different order (NOW and WON). So both words hash to the same address. This situation is known as a **collision**. Clearly, there is only space at this address for one English-French word pair.

Collisions can be resolved in two ways:

1. Store the record in the "next available" empty location in the table, or
2. Store a pointer in each table location that points to a list of records that have all collided at this table location, otherwise set the pointer value to null.

Key concept

Closed hashing or open addressing:
Method in which a collision is resolved by storing the record in the "next available" location.

Method 1 – closed hashing or open addressing

The first way of resolving a collision is to *rehash* which means to generate a new table row address at which to store the English-French word pair.
One rehash method, called **linear rehash**, calculates a new address by adding one to the original address before testing that the location with this new address is empty, e.g. indicated by `'-1'` in the `EnglishWord` field.

The rehash step may have to be repeated until an empty slot is found.

To avoid going off the end of the table, the new address is made to wrap around to the beginning of the hash table, if necessary and assuming there is an empty slot, by using modular arithmetic as follows:

```
Repeat
  Address ← (Address + 1) Mod 523
Until HashTable[Address].EnglishWord = '-1'
```

This method is an example of **closed hashing** or **open addressing** because other row addresses of the hash table are open to being used but access to addresses outside the hash table are closed off.

The table, `HashTable`, is an array whose addresses run from 0 to 522.

The table is initialised with 523 empty English-French word pair records in which every EnglishWord field has the string `'–1'` stored in it to indicate that this field is unoccupied and the whole record is empty.

Figure 2.6.1.2 shows an algorithm expressed in pseudo-code for inserting an English-French word pair into an initialised `HashTable`. The English word to insert is supplied in `WordInE` and its French equivalent in `WordInF`. Each row of the hash table has space for a record with two fields, `EnglishWord` and `FrenchWord`.

```
Address ← Hash(WordInE)
If HashTable[Address].Key = '-1'
                 {-1 indicates field is empty}
  Then
    Begin
      HashTable[Address].EnglishWord ← WordInE
      HashTable[Address].FrenchWord ← WordInF
    End
  Else
    If Not(HashTable[Address].EnglishWord = WordInE)
                                {not already stored}
      Then
        Begin
          {find empty slot)
          Repeat
            Address ← (Address + 1) Mod 523
          Until (HashTable[Address].EnglishWord = '-1')
            Or (HashTable[Address].EnglishWord = WordInE)
                                         {already stored}
          If (HashTable[Address].EnglishWord = '-1')
              Then
                Begin
                  HashTable[Address].EnglishWord ← WordInE
                  HashTable[Address].FrenchWord ← WordInF
                End
        End
```

Figure 2.6.1.2 Hashing algorithm incorporating a linear rehash that inserts an English-French word pair into a hash table

Clearly for this algorithm to work the hash table must have at least one empty row.

Searching for a specific record in a hash table accommodating collisions

Figure 2.6.1.3 shows an algorithm expressed in pseudo-code that can be used to search for an English-French word pair in a hash table, `HashTable`, given an English word stored in the variable `WordInE`.

The English word may or may not be present in the hash table.

If it is, then its French equivalent is returned in variable, `Retrieve` otherwise message `'Not found'` is returned in `Retrieve`.

Clearly for this algorithm to work the hash table must have at least one empty row.

```
Address ←  Hash(WordInE)
Found ←  False
Repeat
  If HashTable[Address].EnglishWord = WordInE
    Then Found ←  True
    Else Address ←  (Address + 1) Mod 523
Until Found Or (HashTable[Address].EnglishWord = '-1')
If Found
  Then Retrieve ←  HashTable[Address].FrenchWord
  Else Retrieve ←  'Not found'
```

Figure 2.6.1.3 Hashing algorithm incorporating a linear rehash method that is used to search a hash table for the French equivalent of a given English word

Setting up a hash table that uses closed hashing

Method 1 (closed hashing or open addressing) requires that the number of rows in the table exceeds by about a third the maximum number of records that will ever be stored in the table. When every record has been stored in the table the table should still contain empty rows (i.e. table should never be more than roughly two thirds full). If this isn't the case then search times will be extended as will the time to insert new records.

Although this might seem a waste of storage space, there is a very good reason for working in this way. Studies have shown that the number of collisions depends on

- the hash keys
- the hash function
- the ratio of total number of records to total number of possible locations available to these records in the hash table.

A perfect hash function hashes all the hash keys to hash values without the occurrence of a single collision. That is why it is called perfect.

However, finding a perfect hash function is extremely difficult.

The effectiveness of a hash function is very sensitive to the hash key values. These are not always fully known in advance.

Using a ratio of roughly two thirds for total number of records to total number of hash table locations seems to minimise collisions for hash functions that are close to perfect. The hash table shown in *Table 2.6.1.5* has six student records and seven rows. One improvement could be to change the number of rows to 9 or even better, 11, a prime number. **Using a prime number for modulo arithmetic helps to minimise collisions**.

However, the hash function could be further improved as well as it is far from being perfect.

The aim is to make each hash value generated by the hash function equally likely when the function is applied to any of the possible hash keys, i.e. no one particular hash value should be more favoured than any other.

	ULN	Forename	Surname
0	34567876	Fred	Bloggs
1			
2	90002789	Mary	Smith
3	64156906	Alex	Black
4	24567805	Visha	Baal
5	74432167	Ben	Brown
6	90002985	Shena	Patel

Table 2.6.1.5 Hash table with not enough rows to minimise collisions

Questions

3 Copy and complete *Table 2.6.1.6*.

ULN	ULN Mod 7	ULN Mod 11
24567805		
34567876		
64156906		
74432167		
90002789		
90002985		

Table 2.6.1.6

Questions

4 Insert the ULNs from *Table 2.6.1.6* into a copy of the hash table shown in *Table 2.6.1.7* using the hashing function,

```
H(ULN) = ULN Mod 7
```

The student `Forename` and `Surname` fields do not need to be completed.
You should deal with any collision by performing a linear rehash until an empty slot is found.

	ULN	Forename	Surname
0			
1			
2			
3			
4			
5			
6			

Table 2.6.1.7 Hash table

5 Insert the ULNs from *Table 2.6.1.6* into a copy of the hash table shown in *Table 2.6.1.8* using the hashing function,

```
H(ULN) = ULN Mod 11
```

The student `Forename` and `Surname` fields may be omitted for convenience.

	ULN	Forename	Surname
0			
1			
2			
3			
4			
5			
6			
7			
8			
9			
10			

Table 2.6.1.8 Hash table

Questions

6 Explain how the hash table in *Table 2.6.1.8* when populated with student records would be used to look up the forename and surname of student with ULN = 24567805.

7 Explain how the hash table in *Table 2.6.1.7* when populated with student records would be used to look up the forename and surname of student with ULN = 24567805.

8 Why is it necessary to store the key field in a hash table even when an application using this hash table must already know the value of the key field?

Investigation

1 Devise an experiment to investigate collisions on a hash table that is to store 6000 student records. Use a random number generator to generate unique student ID numbers. Try different ratios of total number of records to total number of table rows in the hash table.

Investigation

2 The hash function H that we have used so far is far from perfect for many data sets that we wish to store in a hash table. Investigate other hashing functions.

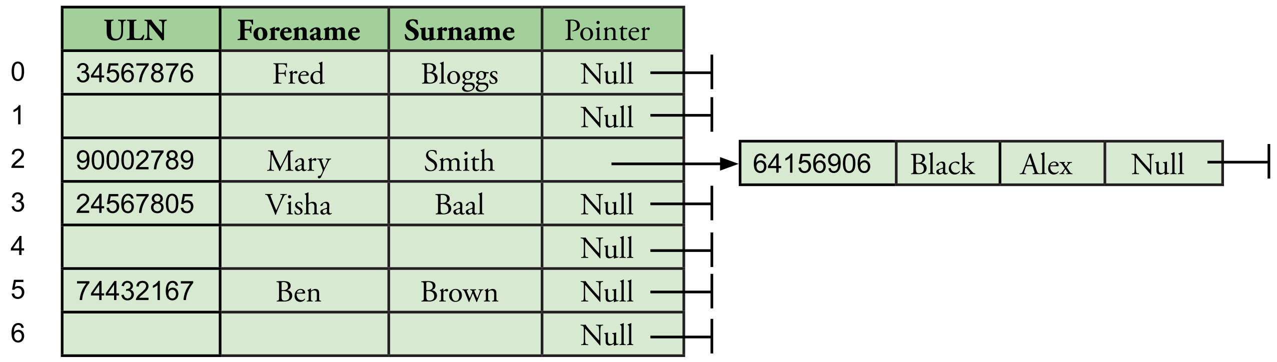
Method 2 - open hashing or closed addressing

In this alternative method of dealing with collisions, the hash table is extended to include a pointer field. The pointer field for each row is initialised to the null pointer value when the table is set up (⊣).

When a collision occurs the colliding record is linked to the corresponding table row by changing the pointer field of this row to point to the colliding record as shown in *Figure 2.6.1.4*.

Key concept

Open hashing or closed addressing:
In a collision, the other rows of the hash table are closed to the colliding record which must, instead, be attached to the addressed table row in a chain or linked list of other colliding records. The table row uses a pointer field to point to the linked list.



	ULN	Forename	Surname	Pointer
0	34567876	Fred	Bloggs	Null
1				Null
2	90002789	Mary	Smith	
3	24567805	Visha	Baal	Null
4				Null
5	74432167	Ben	Brown	Null
6				Null

64156906	Black	Alex	Null

Figure 2.6.1.4 Hash table that uses open hashing

Information

The definitions assigned to the terms closed hashing and open hashing have been interchanged over the years so care needs to be exercised when interpreting them. The key is to focus on concept/method not name and to make sure that you understand the former.

Another record colliding with row 2 will be linked or chained to the record of `'Mary Smith'` by changing the pointer field of Mary's record to point to this record and so on, thus forming a chain of linked records or linked list. Method 2 is called **open hashing** or **closed addressing** because locations outside the table are open for use by the hashing algorithm, i.e. the linked list locations, whilst other row addresses are closed off.

Deleting a record

Care must be exercised when an entry in a hash table is deleted.

Closed hashing

In closed hashing, collisions are resolved by rehashing and storing the colliding record in another row whose table index is the rehash value.

However, if the entry at the original hash value table index or any of the rehash value table entries are deleted and the deleted entry remains empty, searching can be stopped prematurely before all potential matching entries have been examined.

Therefore, a deleted entry must be distinguishable from an entry that has never been used. This requires a special marker to be present in the key field part of the hash table entry when the entry is not in use. The special marker will use one value to indicate that this entry has never been used and a different value to indicate that it has been used but the entry has been deleted.

The special marker values should not use any value that potentially could occur in the key fields of the data set to be stored in the hash table.

Open hashing

In open hashing, collisions are resolved by chaining the colliding record to the table entry slot whose index is the hash. Care must be taken when deleting the record in the table row when the row has a nonempty chain.

A special marker can be left in the key field to signal that there is at least one record in a chain (linked list) attached to the row so that a search does not fail to look at the chain when seeking a match.

There are at least two alternatives that do not rely on a special marker.

In alternative one, the search examines the pointer field of an empty slot to see if a chain is attached.

In alternative two, the first record in the chain is moved into the table slot whilst preserving its link to the rest of the chain.

Questions

9 An empty hash table is set up for open hashing. The following hashing function is to be used to store variable names beginning with an uppercase letter in range A...Z, as well as other information.

`H(VariableName)` = (code for first letter of `VariableName x 11) Mod M`

Where `M` is the number of rows in the hash table.

Using `M = 5` and coding letters of the alphabet as follows, A=1, B=2, ..., Z=26 show the contents of the hash table after inserting the following variable names:

CHECK, OVERTIME, MAIN, P, URL, TAXRATE, INDEX, N, GENDER

You may ignore in your answer the other information associated with each variable name.

10 (a) Using the hashing algorithm expressed in pseudo-code below, calculate the hash value for the hash key 'PEN' stored in string variable `Key`. You will need access to an ASCII code table to map characters to their equivalent ASCII codes. This is performed in the pseudo-code by the function `Ord`. The `Length` function returns the number of characters in the string. The symbol `'*'` means multiply.

```
Sum ← 0
For i ← 0 To Length(Key)- 1
    Sum ← Sum + Ord(Key[i]) * Ord(Key[i])
EndFor
HashValue ← Sum Mod 523
```

(b) Now repeat the exercise with the made-up word `'NEP'`.

(c) Can you see that there is a problem? What is the problem?

(d) Describe **two** ways that could be used to overcome this problem.

11 Explain why care must be exercised when deleting an entry in a hash table that uses closed hashing and on which searching occurs after deletion.

12 A person owns n distinct pairs of socks, which are kept in an unmatched pile in a drawer.
Individual socks are pulled from the drawer blindly, then identified and placed in a separate pile according to identity.

(a) How many individual socks must the person pull from the drawer to ensure that two are pulled that match?

(b) In what respect does this process resemble a hash table and open hashing?

Questions

13 In an application, student records are identified by their key field, the student's unique learner number (`ULN`) consisting of eight digits, e.g. 34567890. The application has to process a `ULN` allocated in the range 1000000 to 99999999 but it will never have to deal with more than 500 ULNs.

(a) Explain why when storing student records in a table in memory it would not be sensible to use the `ULN` as the row address for the record, e.g. 34128496.

(b) Explain why the use of a hash table would be a better option for this application.

14 (a) State **two** advantages of using hashing and the hash table approach over the alternative approach which just stores records in an ordinary table starting from the first row.

(b) It is noticed that after inserting many records into a hash table that uses closed hashing, searches are taking much longer than they did.

(i) Explain why this may be the case

(ii) Suggest a solution that could potentially restore searching times to what they were.

15 Explain why it is necessary to store the hash key in a hash table.

In this chapter you have covered:

- The concept of a hash table and its uses.
- Applying simple hashing algorithms.
- What is meant by a collision and how collisions are handled using rehashing.