Hashing

Jonathan Windle

University of East Anglia

J.Windle@uea.ac.uk

June 4, 2017

Overview I

Intro

- Technique for performing insertions,deletions and finds in a dictionary in constant average time.
- **Hash table:**
  - An array, T of some fixed size is used to store the keys.
  - `size` referres to the size of T.
  - $S = \{0, 1, ..., size - 1\}$
- **Hashing function:**
  - $h : K \to S$.
  - Suppose K is the set of 6 digit non-negative integers, then a possible (but poor) choce for h is:
    $$h(k) = k(mod 1000)$$
- **Collisions:**
  - A collision occurs when two keys hash to the same location in the hash table:
    $h(k) = h(k')$.
  - Want to choose the hash function to minimise the chance of collisions.
  - Need to decide how to handle collisions when they do occur.

Choosing a Hash Function

- A good hash function maps keys uniformly and randomly into the full range of possible locations.
- A good hash function should depend on all of the charactersf the characters in a key, but this is not a sufficient condition for a good hash function.
- Must not just depend on all of the characters in a key but must also distribute keys evenly over the table.
- The built in Java function `hashCode` returns an integer based on the objects reference unless the object is a string then it is based on the string itself.
- The Java class `HashTable` can be used with keys of any user-defined data type provided an instance method `hashCode` is defined.

Resolving Collisions

- Use some other location that is open in the table:
  - Open addressing
- Change the structure of the hash table so that each location can correspond to more than one value:
  - Chaining.
  - Buckets.

Chaining/Buckets

- Chaining:
  - For each location T, keep a list of allthe keys hashed to that location.
  - Each entry in T is thus a reference to a linked list of keys.
  - To form a search, just hash to find the list and then perform the appropriate operation.
- Buckets:
  - Each location in the hash table is a bucket.
  - A fixed number, b of locations to store the keys.
  - Total space available is thus:
    $size \times b$

Open Addressing

- If a collision occurs, alternative cells in T are tried until an empty cell is found.
- Locating an open loaction in te hash table is called probing
- May be necessary to try more than one alternqative location.
- The locations examined when a new key is inserted is called a probe sequence.
- Let $\langle S_j^k \rangle$ denote the probe sequence then:
  $s_0^k = h(k)$
  $s_j^k = (s_{j-1}^k + p(j,k))\%size, \quad j \geq 1.$
- Where $p(j,k)$ is called a probe increment.
- In the simplist scheme the probe increment is independant of both j and k. i.e. it is a constant p in particular linear probing, $p = 1$.

Linear Probing

- If $T[h(k)]$ is occupied, try successive locations in T with wrap-around.
- Retrieval must be able to follow the same path used on insertion.
- Care must be taken over deletion:
  - When searching for the key, if an empty location is encountered before the key is found, this implies that the key is not present in T.
  - Only flag a location for deletion in a delete operation, otherwise a subsequent search might reach an incorrect conclusion.
- When inserting should first check that the given key is not already present in the table:
  - If not, probe sequence ends and is added into the empty location.
  - May encounter flagged locations before hitting the empty location.
  - If we record the position of the first flagged location encountered we can store the entry in that location.
- Been shown that provided load factor is 0.5 (half full table), only 2.5 probes are required onaverage for insertion and only 1.5 on average for a successful search.

Clustering

- Resolving collisions with linear probing leads to clusters of occupied locations.
- Suppose when inserting we hash to a location k in the table and we then have to go through a probe sequence of length r.
- The next time we hash to location k we need to go through a probe sequence of length $r + 1$ at least.
- Each addition to the table, increases the size of some cluster by 1 (at least - may also get merging of clusters).
- Known as primary clustering.

Quadratic Probing I

- Some schenes, the probe increment depends on the value of the index, j, in the probe sequence, but is still independant of k.
- In quadratic probing, the probe sequence is given by:

$$s_j^k = (h(k) + j^2)\%size$$

i.e.

$$\begin{aligned} s_0^k &= h(k) \\ s_j^k &= (s_{j-1}^k + 2j - 1)\%size, \quad j \geq 1, \end{aligned}$$

since

$$j^2 = (j-1)^2 + 2j - 1$$

- The increment function for quadratic probing is $inc(j, k) = 2j - 1$

Quadratic Probing II

```
int i = h(k);
int j = 0;
while (T[i].key != emptyKey) {
  j++;
  i += 2*j - 1%T.size;
}
T[i].key = k;
T[i].data = d;
```

Must be prime

- If size is not prime then there are only p locations that can be generated as alternative loctions when a clash occurs.
- However, if a size is a prime, and T is at least half empty, a key can always be inserted.

Example

- $h(k) = k\%13$
- Insert 8,42,29,51,47,13,26 into table of size 13:

j	inc(j,k)=2j-1
1	1
2	3
3	5
4	7
5	9

k	h(k)=k%13
8	8
42	3
29	3
51	12
47	8
13	0
34	8

i	T[i]	Probe Sequence
0	13	0
1		
2		
3	42	3
4	29	3,4
5		
6		
7		
8	8	8
9	47	8, 9
10		
11	34	8,9,12,4,11
12	51	12

- 8 - straight in position 8.
- 42 - stright in position 3.
- 29:
  - $29\%13 = 3$ - 3 is full,
  - $(3+1)\%13 = 4$ - inserted into 4.
- 51 - straight into 12.
- 47:
  - $47\%13 = 8$ - 8 is full,
  - $(8+1)\%13 = 9$ - inserted into 9.
- 13 - straight into 0
- 34:
  - $34\%13 = 8$ - 8 is full,
  - $(8+1)\%13 = 9$ - 9 is full,
  - $(9+3)\%13 = 12$ - 12 is full,
  - $(12+5)\%13 = 4$ - 4 is full,
  - $(4+7)\%13 = 11$ - inserted into 11

Double Hashing

- Use another hash functionas the increment function.
- E.g. $h_2(k) = r - (k\%r)$
- Same increment is added each stage, but the increment is different for different keys.

```
int i = h(k);
while (T[i].key != emptyKey) {
  i += r-(k%r)%size;
}
T[i].key = k;
T[i].data = d;
```

Example

- $h(k) = k\%13$
- $h_2(k) = 5 - (k\%5)$
- Insert 8,42,29,50,47,13,26

k	h(k)=(k%13)	h2(k)=5-(k%5)
8	8	**2**
42	3	**3**
29	3	**1**
51	12	**4**
47	8	**3**
13	0	**2**
34	8	**1**

i	T[i]	Probe Sequence
0	13	0
1		
2		
3	42	3
4	29	3,4
5		
6		
7		
8	8	8
9	34	8, 9
10		
11	47	8,11
12	51	12

- 8 - straight into 8
- 42 - straight into 3
- 29:
  - $29\%13 = 3$ - 3 is full,
  - $(3 + 1)\%13 = 4$ - inserted into 4
- 51 - straight into 12
- 47:
  - $47\%13 = 8$ - 8 is full,
  - $(8 + 3)\%13 = 11$ - inserted into 11
- 13 - inserted into 0
- 34:
  - $34\%13 = 8$ - 8 is full,
  - $(8 + 1) = 9$ - insert into 9.

Rehashing

- If T gets too full, the running time for operations increases and for hashing with quadratic probing, inserts may fail.
- A solution is to build another hash table, $T2$, approximately the size of T e.g. double size and then increase to the next prime number.
- T is then scanned and each (non-deleted) key from T is inserted into $T2$

The End