Hashing with linear probing

Rasmus Pagh

IT University of Copenhagen

Teoripärlor, Lund, September 16, 2007
Hashing with linear probing
Hashing with linear probing
Hashing with linear probing
Hashing with linear probing

Diagram showing the concept of linear probing in a hash table.
Hashing with linear probing
Hashing with linear probing

It was settled in the 60s that this is inferior to e.g. double hashing. So why care?
389 km/h

20 km/h
Race car vs golf car

• Linear probing uses a sequential scan and is thus *cache-friendly*.

• On my laptop: **24x** speed difference between sequential and random access!

• Experimental studies have shown linear probing to be faster than other methods for load factor $\alpha$ in the range 30-70%.
Race car vs golf car

- Linear probing uses a sequential scan and is thus *cache-friendly*.
- On my laptop: 24x speed difference between sequential and random access!
- Experimental studies have shown linear probing to be faster than other methods for load factor $\alpha$ in the range 30-70%.
- **But**: No theory behind the hash functions used for linear probing in practice.
History of linear probing

- First described in 1954.
- Analyzed in 1962 by D. Knuth, aged 24.
Assumes hash function \( h \) is truly random.

---

NOTES ON "OPEN" ADDRESSING.

D. Knuth. 7/22/63

1. Introduction and Definitions. Open addressing is a widely-used technique for keeping "symbol tables." The method was first used in 1954 by Samuel, Amdahl, and Boehme in an assembly program for the IBM 701. An extensive discussion of the method was given by Peterson in 1957 [1], and frequent references have been made to it ever since (e.g. Schay and Spruth [2], Iverson [3]). However, the timing characteristics have apparently never been exactly established, and indeed the author has heard reports of several reputable mathematicians who failed to find the solution after some trial. Therefore it is the purpose of this note to indicate one way by which the solution can be obtained.

We will use the following abstract model to describe the method: \( N \) is a positive integer, and we have an array of \( N \) variables \( x_1, x_2, \ldots, x_N \). At the beginning, \( x_i = 0 \), for \( 1 \leq i \leq N \).

To "enter the \( k \)-th item in the table," we mean that an integer \( x_k \) is calculated, \( 1 \leq x_k \leq N \), depending only on the item, and the following process is carried out:
History of linear probing

- First described in 1954.
- Analyzed in 1962 by D. Knuth, aged 24. Assumes hash function h is truly random.
- Over 30 papers using this assumption.
- Siegel and Schmidt (1990) showed that it suffices that h is $O(\log n)$-wise independent.
History of linear probing

• First described in 1954.
• Analyzed in 1962 by D. Knuth, aged 24. Assumes hash function $h$ is truly random.
• Over 30 papers using this assumption.
• Siegel and Schmidt (1990) showed that it suffices that $h$ is $O(\log n)$-wise independent.

Our main result:
It suffices that $h$ is 5-wise independent.
log(n)-wise independence

- Siegel (1989) showed time-space trade-offs for evaluation of a function from a \( \log(n) \)-wise independent family:

<table>
<thead>
<tr>
<th></th>
<th>Time</th>
<th>Space</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lower bound</td>
<td>( \frac{\log(n)}{\log(s/\log n)} )</td>
<td>( s )</td>
</tr>
<tr>
<td>Upper bound 1*</td>
<td>( O(\log n) )</td>
<td>( O(\log n) )</td>
</tr>
<tr>
<td>Upper bound 2</td>
<td>( O(1) )</td>
<td>( n^\varepsilon )</td>
</tr>
</tbody>
</table>

- Upper bound 2 is theoretically appealing, but has a huge constant factor – and uses many random memory accesses!
log(n)-wise independence

- Siegel (1989) showed time-space trade-offs for evaluation of a function from a log(n)-wise independent family:

<table>
<thead>
<tr>
<th></th>
<th>Time</th>
<th>Space</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lower bound</td>
<td>(\log(n)) (\frac{\log(n)}{\log(s/\log n)})</td>
<td>(s)</td>
</tr>
<tr>
<td>Upper bound 1*</td>
<td>(O(\log n))</td>
<td>(O(\log n))</td>
</tr>
<tr>
<td>Upper bound 2</td>
<td>(O(1))</td>
<td>(n^\epsilon)</td>
</tr>
</tbody>
</table>

- Upper bound 2 is theoretically appealing, but has a huge constant factor – and uses many random memory accesses!
5-wise independence

- Polynomial hash function:
  \[ h(x) = \left( \sum_{i=0}^{4} a_i x^i \mod p \right) \mod r \]
  Carter and Wegman (FOCS '79)

- Tabulation-based hash function:
  \[ h(x_1, x_2) = T_1[x_1] \oplus T_2[x_2] \oplus T_3[x_1 + x_2] \]
  Thorup and Zhang (SODA '04)
Today

• Background and motivation
• Hash functions
  ▶ New analysis of linear probing
    Joint work with Anna Pagh and Milan Ružić
• Hash functions - details
Total cost of insertions

((analysis on blackboard))
Hash function details

\[ h(x_1, x_2) = T_1[x_1] \oplus T_2[x_2] \oplus T_3[x_1 + x_2] \]

((analysis on blackboard))
Single insertion upper bound
Single insertion upper bound
Single insertion upper bound

1. Choose max \( t \) so \( B \) balls hash to \( B-t \) slots, for some \( B \).
Single insertion upper bound

1. Choose max \( t \) so \( B \) balls hash to \( B-t \) slots, for some \( B \)

2. Choose max \( C \) such that \( C \) balls hash to \( C+t \) slots
Single insertion upper bound

1. Choose max $t$ so $B$ balls hash to $B-t$ slots, for some $B$

2. Choose max $C$ such that $C$ balls hash to $C+t$ slots
Single insertion upper bound

1. Choose max $t$ so $B$ balls hash to $B-t$ slots, for some $B$

2. Choose max $C$ such that $C$ balls hash to $C+t$ slots
Single insertion upper bound

1. Choose max $t$ so $B$ balls hash to $B-t$ slots, for some $B$

2. Choose max $C$ such that $C$ balls hash to $C+t$ slots

Lemma:

$\text{Cost( )} \leq 1 + C + t$
Proof idea

- **Lemma:** If operation on $x$ goes on for more than $k$ steps, then there are "unusually many" keys with hash values in either:

  1) Some interval with $h(x)$ as right endpoint, or
  2) The interval $[h(x), h(x) + k]$
Proof idea

• **Lemma**: If operation on $x$ goes on for more than $k$ steps, then there are “unusually many” keys with hash values in either:

1) Some interval with $h(x)$ as right endpoint, or
2) The interval $[h(x), h(x) + k]$

• To bound cost, upper bound probability of each event using tail bounds for sums of random variables with limited independence.
Our main result

**Theorem 2** Consider any sequence of insertions, deletions, and lookups in a linear probing hash table using a 5-wise independent hash function. Then the expected cost of any operation, performed at load factor $\alpha$, is

$$O(1 + (1 - \alpha)^{-3}) .$$

As a consequence, the expected average cost of successful lookups is $O(1 + (1 - \alpha)^{-2}).$
Our main result

**Theorem 2** Consider any sequence of insertions, deletions, and lookups in a linear probing hash table using a 5-wise independent hash function. Then the expected cost of any operation, performed at load factor $\alpha$, is

$$O(1 + (1 - \alpha)^{-3}) .$$

As a consequence, the expected average cost of successful lookups is $O(1 + (1 - \alpha)^{-2})$. 

Factor $(1 - \alpha)^{-1}$ from what can be proved using full independence.
End remarks

• Theory and practice of linear probing now (seem) much closer.

• We can generalize to variable key lengths.
End remarks

• Theory and practice of linear probing now (seem) much closer.

• We can generalize to *variable key lengths*.

• Open:
  ▶ Still many hashing schemes where theory does not provide satisfactory methods.
  ▶ Tighter analysis, lower independence?
Call for applications, PhD scholarships at ITU: http://www1.itu.dk/sw66047.asp

For project proposals in algorithms, talk to me before October 1.
Problems

• Show that the total number of insertion steps is independent of the insertion order.

• Show how to efficiently implement deletions in a linear probing hash table.

• Show that the following hash function is 5-wise independent (hint: recursion):

\[
h(x_1, x_2, x_3, x_4) = T_1[x_1] \oplus T_2[x_2] \oplus T_3[x_3] \oplus T_4[x_4] \\
\quad \oplus T_5[x_1 + x_2] \oplus T_6[x_3 + x_4] \\
\quad \oplus T_7[x_1 + x_2 + x_3 + x_4]
\]
Practical exercise

- Implement a dictionary for null-terminated strings using linear probing. Beware of the "bad" way(s) of implementing it!

- Use the dictionary to count the number of distinct words in "Love and War".
  http://www.gutenberg.org/etext/2600

- Compare the performance to the standard hash table in your programming environment.