Create account Log in

Article Talk Read Edit View history

Main page
Contents
Featured content
Current events
Random article
Donate to Wikipedia

Interaction
Help
About Wikipedia
Community portal
Recent changes
Contact Wikipedia

Toolbox

Print/export

Languages
Deutsch

Nederlands

Double hashing

From Wikipedia, the free encyclopedia

Double hashing is a computer programming technique used in hash tables to resolve hash collisions, cases when two different values to be searched for produce the same hash key. It is a popular collision-resolution technique in open-addressed hash tables. Double hashing is implemented in many popular computer libraries.

Contents [hide]

Classical applied data structure [edit]

Double hashing with open addressing is a classical data structure on a table T. Let n be the number of elements stored in T then T's load factor is $\alpha = \frac{n}{|T|}$.

Double hashing approximates uniform open address hashing. That is, start by randomly, uniformly and independently selecting two universal hash functions h_1 and h_2 to build a double hashing table T.

All elements are put in T by double hashing using h_1 and h_2. Given a key k, determining the $(i+1)$-st hash location is computed by:

$$h(i,k) = (h_1(k) + i\,h_2(k)) \mod |T|.$$

Let T have fixed load factor $\alpha : 1 > \alpha > 0$. Bradford and Katehakis [1] showed the expected number of probes for an unsuccessful search in T, still using these initially chosen hash functions, is $\frac{1}{1-\alpha}$ regardless of the distribution of the inputs.

Previous results include: Guibas and Szemerédi [2] showed $\frac{1}{1-\alpha}$ holds for unsuccessful search for load factors $\alpha < 0.319$. Also, Lueker and Molodowitch [3] showed this held assuming ideal randomized functions. Schmidt and Siegel [4] showed this with more realistic k-wise independent and uniform functions (for $k = c \log n$, and suitable constant c).

Like linear probing, it uses one hash value as a starting point and then repeatedly steps forward an interval until the desired value is located, an empty location is reached, or the entire table has been searched; but this interval is decided using a second, independent hash function (hence the name double hashing). Unlike linear probing and quadratic probing, the interval depends on the data, so that even values mapping to the same location have different bucket sequences; this minimizes repeated collisions and the effects of clustering. In other words, given independent hash functions h_1 and h_2, the *j*th location in the bucket sequence for value *k* in a hash table T is:

$$h(k,j) = (h_1(k) + j \cdot h_2(k)) \mod |T|$$

Disadvantages

[edit]

Linear probing and, to a lesser extent, quadratic probing are able to take advantage of the data cache by accessing locations that are close together. Double hashing has larger intervals and is not able to achieve this advantage. To avoid this situation, store your data with the second key as the row, and your first key as the column. Doing this allows you to iterate on the column, thus preventing cache problems. This also prevents the need to rehash the second key.

For instance:

```
pData[hk_2][hk_1]

int hv_1 = Hash(v)
int hv_2 = Hash2(v)

int original_hash = hv_1
while(pData[hv_2][hv_1]){
  hv_1 = hv_1 + 1
}
```

Like all other forms of open addressing, double hashing becomes linear as the hash table approaches maximum capacity. The only solution to this is to rehash to a larger size.

On top of that, it is possible for the secondary hash function to evaluate to zero. For example, if we choose k=5 with the following function:

$$h_2(k) = 5 - (k \mod 7)$$

The resulting sequence will always remain at the initial hash value. One possible solution is to change the secondary hash function to:

$$h_2(k) = (k \mod 7) + 1$$

This ensures that the secondary hash function will always be non zero.

Essentially, Double Hashing is hashing on an already hashed key.

See also

[edit]

- Hash collision
- Hash function
- Linear probing
- Cuckoo hashing

Notes

[edit]

1. ^ P. G. Bradford and M. Katehakis *A Probabilistic Study on Combinatorial Expanders and Hashing* , SIAM Journal on Computing 2007 (37:1), 83-111. http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.91.2647

2. ^ L. Guibas and E. Szemerédi: *The Analysis of Double Hashing*, Journal of Computer and System Sciences, 1978, 16, 226-274.
3. ^ G. S. Lueker and M. Molodowitch: *More Analysis of Double Hashing*, Combinatorica, 1993, 13(1), 83-96.
4. ^ J. P. Schmidt and A. Siegel: *Double Hashing is Computable and Randomizable with Universal Hash Functions*, manuscript.

External links [edit]

- How Caching Affects Hashing by Gregory L. Heileman and Wenbin Luo 2005.
- Hash Table Animation

This algorithms or data structures-related article is a stub. You can help Wikipedia by expanding it.

Categories: Search algorithms | Hashing | Algorithms and data structures stubs