Hash table – performance analysis

Hash function assumption: computed in $\Phi(1)$

Load factor: $\alpha = n / m$

- n = number of entries used
- m = hash table size

Hash table and **α**

$\alpha = 1$: table is full ➔ increase the table

rehash !!

$\alpha < 1$

α from 0 to 0.75 **(up to 0.7, about 2/3 full)**

- With a good hash function, the average lookup cost is nearly constant
  Beyond that point, the prob. of collisions and the cost of handling them increases.
  a low load factor
- wasted memory
- not necessarily any reduction in search cost

Performance : collision resolution by chaining

under simple uniform hashing

average complexity for operations: $\Phi(1)$

add - $\Phi(1)$

search:

Theorem:

under simple uniform hashing an unsuccessful search takes $\Phi(1 + \alpha)$ time on the average.

Theorem:

under simple uniform hashing a successful search takes $\Phi(1 + \alpha)$ time on the average.

delete: search (the list) and remove

Performance : collision resolution by open addressing

under the assumption of simple uniform hashing and constant α

average complexity for operations: $\Phi(1)$

Theorem:

under the assumption of uniform hashing

with load factor $\alpha = n/m < 1$

the expected number of probes

- in an unsuccessful search is at most $1/(1-\alpha)$,
- for operation add is at most $1/(1-\alpha)$
- in a successful search is at most $1/\alpha * \log(1/(1-\alpha))$

Collision resolution by chaining. Variations.

we can use: list

other data structure

- sorted list
- doubly linked list
- self-balancing tree
  - worst-case time O(log n)
  - extra memory and extra design
    - if long delays must be avoided at all costs

      e.g. in a real-time application

.

Collision resolution. Variations

two-level hashing

- first level

  use a hashing function chosen from a family of universal hash functions.

- second level

  Use (small) secondary table S_j with an associated hash function hj

…