CS 4/56101

Design and Analysis of Algorithms

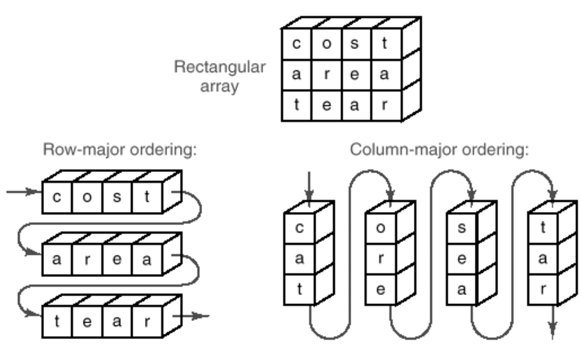
Kent State University

Dept. of Math & Computer Science

LECT-9

Breaking the Log n Barrier

- Can we search faster than log n speed?
- Table look up:

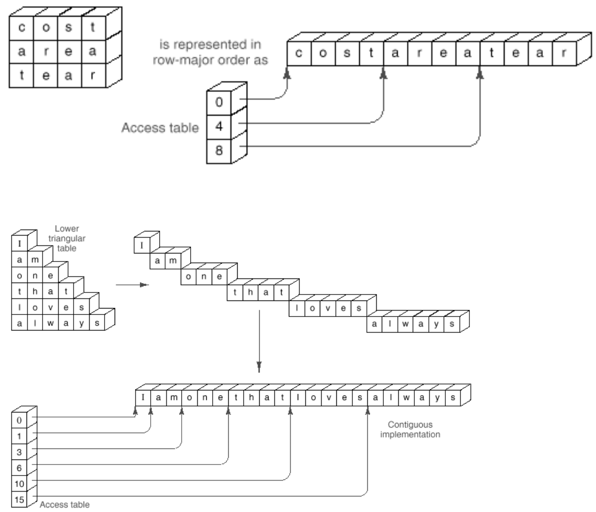


DESIGN & ALALYSIS OF ALGORITHM

ALG00S, javed@kent.edu

Tables of Various Shapes

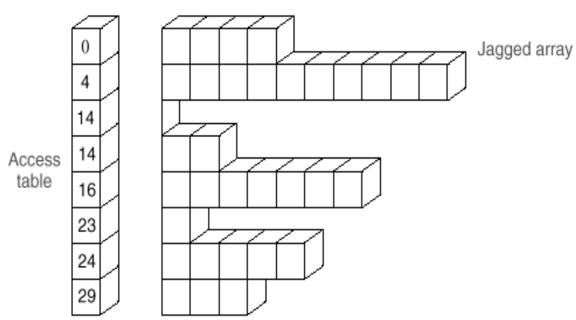
- Access Table



DESIGN & ALALYSIS OF ALGORITHM

LECT-09, S-4
ALG00S, javed@kent.edu
Javed I. Khan@1999

Irregular Table



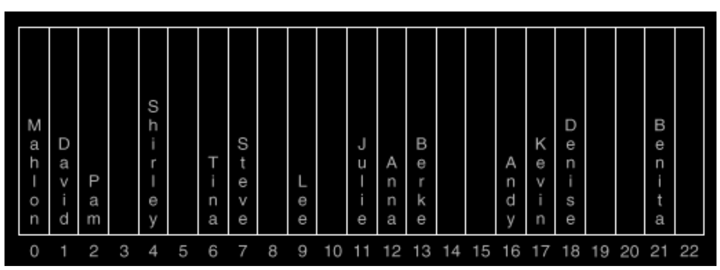
DESIGN & ALALYSIS OF ALGORITHM

LECT-09, S-5
ALG00S, javed@kent.edu
Javed I. Khan@1999

Hashing

6

Hashing



DESIGN & ALALYSIS OF ALGORITHM

- Start with an *array* that holds the hash table.
- Use a *hash function* to take a key and map it to some index in the array.
- If the desired record is in the location given by the index, then we are finished; otherwise we must use some method to resolve the *collision* that may have occurred between two records wanting to go to the same location.

Today's Math

$$U = 1 + 2x + 3x^2 ...... + mx^{m-1} = \sum_{k=1}^{m} k \cdot x^{k-1}$$

- We know:

$$\frac{x^{m+1} - 1}{x - 1} = 1 + x + x^2 ...... + x^m$$

$$\frac{(x-1)(m+1)x^m - (x^{m+1} - 1)}{(x-1)^2} = 1 + 2x + 3x^2 ...... + mx^{m-1}$$

$$if |x| < 1$$

$$\sum_{k=1}^{\infty} kx^{k-1} = \lim_{m \to \infty} \frac{(x-1)(m+1)x^m - (x^{m+1} - 1)}{(x-1)^2} = \frac{1}{(x-1)^2}$$

DESIGN & ALALYSIS OF ALGORITHM

LECT-09, S-8
ALG00S, javed@kent.edu
Javed I. Khan@1999

Choice of Hash Function

- Quick to Compute
- Randomization
- Truncation
  - pick first second and fifth numbers
  - example n=62538194, h=394
- Folding
  - use all
  - 62538194 maps to 625+381+94=1100
- Modular Arithmetic
  - n mod HASHSIZE. HASHSIZE is some prime
  - 47 mod 7 =5

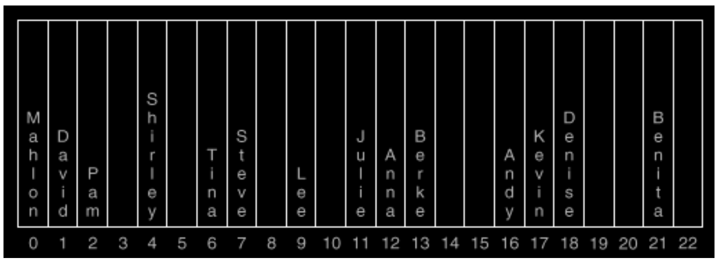
DESIGN & ALALYSIS OF ALGORITHM

LECT-09, S-9
ALG00S, javed@kent.edu
Javed I. Khan@1999

Collision Resolution by Open Addressing

DESIGN &
ALALYSIS OF
ALGORITHM

- Linear Probing
  - *Linear probing* **starts with the hash address and searches sequentially for the target key or an empty position. The array should be considered circular, so that when the last location is reached, the search proceeds to the first location of the array.**



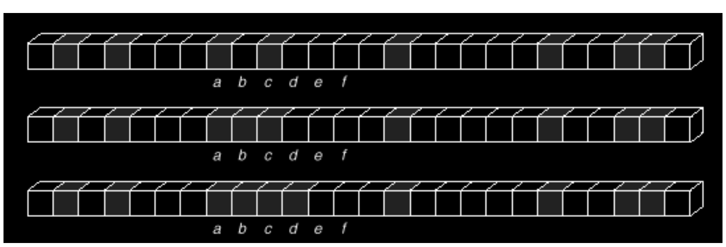
Collision Resolution by Open Addressing

DESIGN &
ALALYSIS OF
ALGORITHM

- Clustering
  - *Linear probing* **starts with the hash address and searches sequentially for the target key or an empty position. The array should be considered circular, so that when the last location is reached, the search proceeds to the first location of the array.**

What is the probability of hitting location f?

What is the probability of hitting location e?

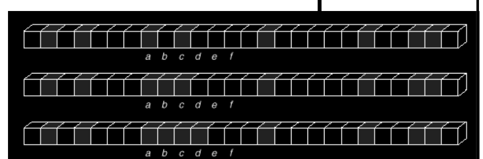


Collision Resolution by Open Addressing

- Quadratic Probing
  - If there is a collision at hash address *h*, *quadratic probing* goes to locations $h+1$, $h+4$, $h+9$, that is, at locations $h+i^2$ (mod hashsize) for $I = 1,2...$

Quadratic Probing only searches half of the locations.

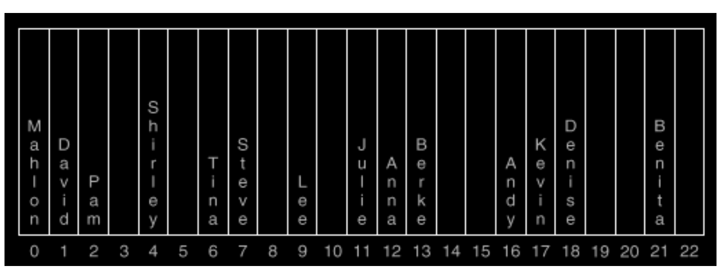
- Other Probing Methods
  - Key dependent probing
  - Random Probing



DESIGN &
ALALYSIS OF
ALGORITHM

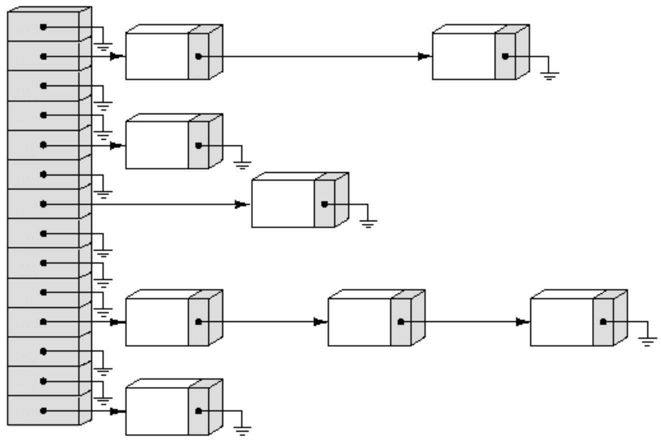
Key Deletion

- Simple Deletion from Hash Table:
  - 11=h(Julie); 12=h(Anna);11=h(Berke)
  - Now we delete Anne.
  - Can we find **Berke**?
- Solution?



DESIGN &
ALALYSIS OF
ALGORITHM

Collision Resolution by Chaining



Pros and Cons of Chaining

- Simple and efficient collision handling
- No Overflow
- Easy Deletion
- Space saving if records are large. The size of static hash table is still small. Only the chain grows.
- Cons: Extra space in links. The relative waste increases if records are small.

DESIGN &
ALALYSIS OF
ALGORITHM

Birthday Surprise

DESIGN & ALALYSIS OF ALGORITHM

- With randomly chosen people in a room, what is the probability that no two have the same birthday?
  - The probability that the second person has no birthday collision is 364/365
  - The probability that the second person has no birthday collision is 363/365
  - The probability that mth person has a different birthday is (365-m+1)/365
  - The probability that all m persons have separate birthday:

$$\frac{364}{365} \times \frac{363}{365} \times ..... \times \frac{365-m+1}{365}$$

  - This becomes less than .5 when m>23!

Analysis of Hashing

Definitions

- What is the cost factor?
  - A *probe* is one comparison of a key with the target.
- Load factor
  - The *load factor* of the table is $\lambda = n/t$, where *n* positions are occupied out of a total of *t* positions in the table.

DESIGN & ALALYSIS OF ALGORITHM

LECT-09, S-18
ALG00S, javed@kent.edu
Javed I. Khan@1999

Analysis of Chaining

- Unsuccessful retrieval
  - a chain have to be searched until end.
  - Average chain is $\lambda = n/t$
- Successful retrieval
  - (n-1) mismatched keys and 1 matching key.
  - Average mismatch keys per chain (n-1)/t
  - Average probe $(n-1)/2.t + 1 \approx 1 + \lambda/2$

DESIGN & ALALYSIS OF ALGORITHM

LECT-09, S-19
ALG00S, javed@kent.edu
Javed I. Khan@1999

Analysis of Open Addressing (random probe)

- Unsuccessful Probe:
  - An unsuccessful search terminates when it encounters an empty space.
  - Probability that the first probe hits a full cell= λ
  - Probability that the first probe hits an empty cell =(1- λ)
  - Probability for exact two probe and it termites is = λ.(1- λ)
  - Probability for exact k probe =$\lambda^{(k-1)}$ (1- λ)
  - The expected number of probe:

$$U(\lambda) = \sum_{k=1}^{\infty} k \cdot \lambda^{k-1}(1-\lambda) = ??$$

$$= \frac{1}{1-\lambda}$$

Analysis of Open Addressing (random probe)

- Successful Probe:
  - A successful probe will be equal to the number of unsuccessful search made before inserting the entry, plus one.
  - The table is initially empty with load=0, and it grows.
  - The average number of search in a successful search is:

$$S(\lambda) = \frac{1}{\lambda}\int_0^{\lambda} U(\mu)d\mu = \frac{1}{\lambda}\ln\frac{1}{1-\lambda}$$

Analysis of Open Addressing (linear probe)

- A little more complex analysis since, successive probes are dependant.

Retrieval from a hash table with open addressing, linear probing, and load factor λ requires approximately

$$\frac{1}{2}\left(1+\frac{1}{1-\lambda}\right) \quad \text{and} \quad \frac{1}{2}\left(1+\frac{1}{(1-\lambda)^2}\right)$$

probes in the successful case and in the unsuccessful case, respectively.

DESIGN & ALALYSIS OF ALGORITHM

ALG00S, javed@kent.edu
Javed I. Khan@1999

Theoretical Comparisons

Load factor	0.10	0.50	0.80	0.90	0.99	2.00
Successful search, expected number of probes:						
Chaining	1.05	1.25	1.40	1.45	1.50	2.00
Open, Random probes	1.05	1.4	2.0	2.6	4.6	—
Open, Linear probes	1.06	1.5	3.0	5.5	50.5	—
Unsuccessful search, expected number of probes:						
Chaining	0.10	0.50	0.80	0.90	0.99	2.00
Open, Random probes	1.1	2.0	5.0	10.0	100.	—
Open, Linear probes	1.12	2.5	13.	50.	5000.	—

Empirical Comparisons

Load factor	0.1	0.5	0.8	0.9	0.99	2.0
Successful search, average number of probes:						
Chaining	1.04	1.2	1.4	1.4	1.5	2.0
Open, Quadratic probes	1.04	1.5	2.1	2.7	5.2	—
Open, Linear probes	1.05	1.6	3.4	6.2	21.3	—
Unsuccessful search, average number of probes:						
Chaining	0.10	0.50	0.80	0.90	0.99	2.00
Open, Quadratic probes	1.13	2.2	5.2	11.9	126.	—
Open, Linear probes	1.13	2.7	15.4	59.8	430.	—

DESIGN & ALALYSIS OF ALGORITHM

ALG00S, javed@kent.edu
Javed I. Khan@1999

DESIGN & ALALYSIS OF ALGORITHM

Comments

- Chaining consistently requires fewer probing than open addressing.
- Which method to use when unsuccessful search is more common?
- If most cases are successful, and the table is not nearly full simpler method of linear probing is not significantly slower than other complex methods.

Load factor	0.10	0.50	0.80	0.90	0.99	2.00
Successful search, expected number of probes:						
Chaining	1.05	1.25	1.40	1.45	1.50	2.00
Open, Random probes	1.05	1.4	2.0	2.6	4.6	—
Open, Linear probes	1.06	1.5	3.0	5.5	50.5	—
Unsuccessful search, expected number of probes:						
Chaining	0.10	0.50	0.80	0.90	0.99	2.00
Open, Random probes	1.1	2.0	5.0	10.0	100.	—
Open, Linear probes	1.12	2.5	13.	50.	5000.	—

DESIGN & ALALYSIS OF ALGORITHM

Comments (contd..)

- In Hashing based IR, the retrieval time is dependant on load factor not on the number of elements in the list.
  - 20,000 keys in a hash table of 40,000 is same as 20 keys in a list of 40!
- The key to performance is the hash function
  - how quickly it can be evaluated
  - how well it spread the data.

Game of Life Revisited..

- In version 2 we solved the problem of sparse computation.
- How about space complexity?
  - Perhaps hashing can help!
- For each cell we need to keep:
  - status (live or dead)
  - neighbor count
  - x and y
- Open Addressing or Chaining?
  - Large hash table vs. 25% pointer overhead.
- 4 way linked list.
  - Each node must be a member of four lists maylive, maydie, newlive and newdie.