Hashing

- Hashing Introduction
- Hash Functions
- Hash Table
- Closed hashing(open addressing)
  - Linear Probing
  - Quadratic Probing
  - Double hashing
- Open hashing(separate chaining)

Hashing

- The search time of each algorithm discussed so far depends on number n depends on the number **n** of items in the collection S of data.
- A searching Technique, called Hashing or Hash addressing, which is independent of number n.
- We assume that
  1. there is a file F of n records with a set K of keys which unlikely determine the record in F.
  2. F is maintained in memory by a Table T of m memory locations and L is the set of memory addresses of locations in T.
  3. For notational convenience, the keys in K and Address L are Integers.

Example

- Suppose A company with 250 employees assign a 5-digit employee number to each employee which is used as primary key in company's employee file.
  - We can use employee number as a address of record in memory.
  - The search will require no comparisons at all.
  - Unfortunately, this technique will require space for 1,00,000 memory locations, where as fewer locations would actually used.
  - So, this trade off for time is not worth the expense.

Hashing

- The general idea of using the key to determine the address of record is an excellent idea, but it must be modified so that great deal of space is not wasted.
- This modification takes the form of a function H from the set K of keys in to set L of memory address.
  - $H: K \rightarrow L$, Is called a Hash Function or
  - Unfortunately, Such a function H may not yield distinct values: it is possible that two different keys k_1 and k_2 will yield the same hash address. This situation is called Collision, and some method must be used to resolve it.

Hash Functions

- the two principal criteria used in selecting a hash function H: K$\rightarrow$ L are as follows:

1. The function H should be very easy and quick to compute.
2. The function H should as far as possible, uniformly distribute the hash address through out the set L so that there are minimum number of collision.

Hash Functions

1. Division method: choose a number m larger than the number n of keys in K. (m is usually either a prime number or a number without small divisor) the hash function H is defined by

   H(k) = k (mod m) or **H(k) = k (mod m)** + 1.

   here k (mod m) denotes the reminder when k is divided by m. the second formula is used when we want a hash address to range from 1 to m rather than 0 to m-1.

2. Midsquare method: the key k is squared. Then the hash function H is defined by **H(k)** = ℓ. where ℓ is obtained by deleting digits from both end of k^2.

3. Folding Method: the key k is portioned into a number of parts, k1, k2, ……,kr, where each part is added togather, ignoring the last carry.

   H(k) = k1+k2+ ……………+Kr.

Sometimes, for extra “milling”, the even numbered parts, k2, k4, …. Are each reversed befor addition.

Example of Hash Functions

- consider a company with 68 employees assigns a 4-digit employee number to each employee. Suppose L consists of 100 two-digit address: 00, 01, 02 , ...99. we apply above hash functions to each of following employee numbers: 3205, 7148,2345.

1. Division Method:

choose a prime number m close to 99, m=97.

H(k)=k(mod m): H(3205)=4, H(7148)=67, H(2345)=17.

2. Midsquare Method:

k=	3205	7148	2345
k^2=	10272025	51093904	5499025
H(k)=	72	93	99

3. Folding Method: chopping the key k into two parts and adding yield the following hash address:

H(3205)=32+05=37, H(7148)=71+48=19, H(2345)=23+45=68

Or,

H(3205)=32+50=82, H(7148)=71+84=55, H(2345)=23+54=77

Collision Resolution

- Suppose we want to add a new record R with key K to our file F, but suppose the memory location address H(k) is already occupied. This situation is called Collision.
- There are two general ways to resolve collisions :
  - Open addressing,(array method)
  - Separate Chaining (linked list method)
- The particular procedure that one choose depends on many factors. One important factor is load factor (λ=n/m)i.e. ratio of number n of keys in K (number of records in F) to m of hash address in L.

e.g. suppose a student class has 24 students and table has space for 365 records.

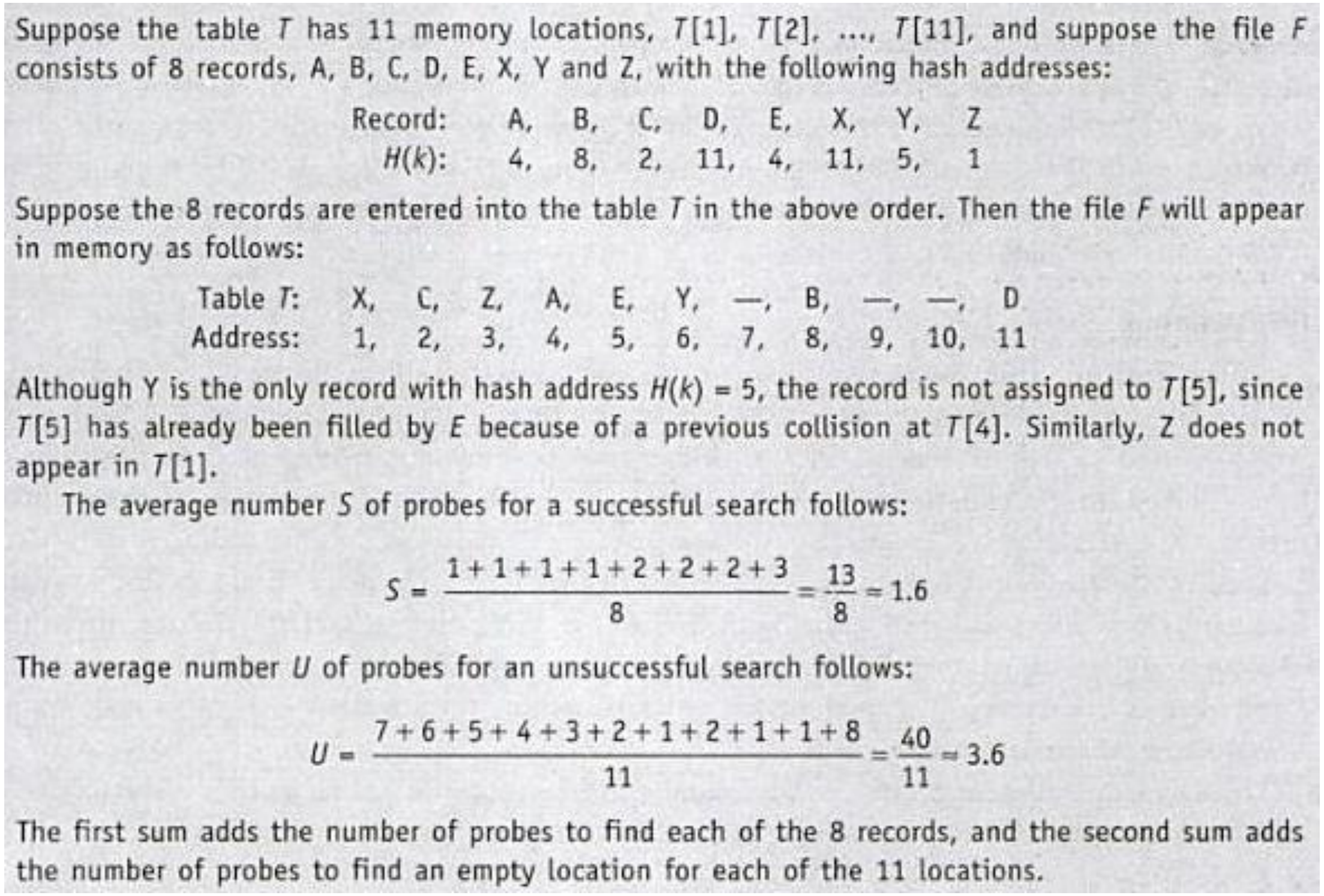
- The efficiency of hash function with a collision resolution procedure is measured by the average number of probes (key comparison) needed to find the location of record with a given k. The efficiency mainly depend on load factor.
- Specially we are interested in following two quantities:
  - $S(\lambda)$ = average number of probes for a successful search
  - $U(\lambda)$ = average number of probes for an unsuccessful search

Open Addressing: Liner Probing and Modifications

- Suppose that new record R with key k is added to memory table T, but that the memory location with hash address H(k)=h is already filled.
- One natural way to resolve the collision is to assign R to the first variable locating following T[h] (we assume that the table T with m location is circular i.e. T[1] comes after T[m]).
  - With such a collision procedure, we will search for record R in table T by linearly search the locations T[h], T[h+1], T[h+2], ………… Until finding R or meeting empty location, which indicates an unsuccessful search.
  - The above collision resolution is called Linear probing.
  - The average number of probes for load factor (λ =n/m) are:

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

EX: Linear Probing



Suppose the table *T* has 11 memory locations, *T*[1], *T*[2], ..., *T*[11], and suppose the file *F* consists of 8 records, A, B, C, D, E, X, Y and Z, with the following hash addresses:

Record:	A,	B,	C,	D,	E,	X,	Y,	Z
$H(k)$:	4,	8,	2,	11,	4,	11,	5,	1

Suppose the 8 records are entered into the table *T* in the above order. Then the file *F* will appear in memory as follows:

Table *T*:	X,	C,	Z,	A,	E,	Y,	—,	B,	—,	—,	D
Address:	1,	2,	3,	4,	5,	6,	7,	8,	9,	10,	11

Although Y is the only record with hash address $H(k) = 5$, the record is not assigned to *T*[5], since *T*[5] has already been filled by *E* because of a previous collision at *T*[4]. Similarly, Z does not appear in *T*[1].

The average number *S* of probes for a successful search follows:

$$S = \frac{1+1+1+1+2+2+2+3}{8} = \frac{13}{8} = 1.6$$

The average number *U* of probes for an unsuccessful search follows:

$$U = \frac{7+6+5+4+3+2+1+2+1+1+8}{11} = \frac{40}{11} = 3.6$$

The first sum adds the number of probes to find each of the 8 records, and the second sum adds the number of probes to find an empty location for each of the 11 locations.

Open Addressing

- One main disadvantage of linear probing is that records tend to cluster, that is, appear next to one another, when the load factor is greater then 50%.

The two technique that minimize the clustering are as:

1. Quadratic probing: Suppose the record R with key K has the hash address H(k)=h. Then instead of searching the locations with address h, h+1, h+2, …….. ., we search the location with address

   h,h+1,h+4,h+9, ………….,h+i^2,……

2. Double hashing: here the second hash function H' is used for resolving a collision, as follows. Suppose a record R with key k has a hash address H(k)=h and H'(k)=h'≠m. Then we linearly search the locations with address

   h, h+h', h+2h', h+3h', …………

Chaining

- Chaining involves maintaining two tables in memory.
- First of all, as before, there is a table T in memory which contains the records in F, except that T now has an additional field LINK which is used so that all record in T with same hash address h may be linked together to form a linked list. Second, there is a hash address table LIST which contain pointers to linked lists in T.
- Suppose a new record R with key k is added to the file F. we place R in the first available location in the table T and then add R to the linked list with pointer LIST[H(k)].
- The average number of probes for load factor (λ =n/m may be greater than 1) are:

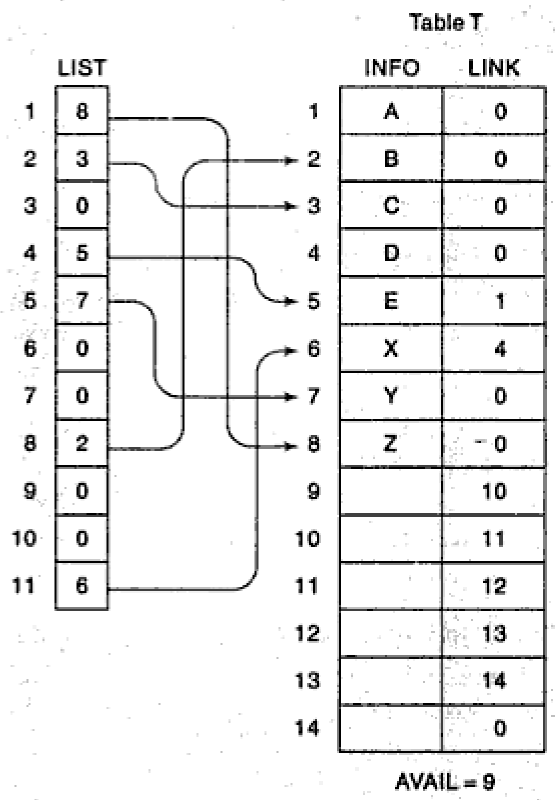
  $S(\lambda) \approx 1 + \lambda/2$ and $U(\lambda) \approx e^{(-\lambda)} + \lambda$.

Ex: Chaining

Suppose the table *T* has 11 memory locations, *T*[1], *T*[2], ..., *T*[11], and suppose the file *F* consists of 8 records, A, B, C, D, E, X, Y and Z, with the following hash addresses:

Record:	A,	B,	C,	D,	E,	X,	Y,	Z
H(*k*):	4,	8,	2,	11,	4,	11,	5,	1

Using chaining, the record will appear in memory as:



	LIST
1	8
2	3
3	0
4	5
5	7
6	0
7	0
8	2
9	0
10	0
11	6

	INFO	LINK
1	A	0
2	B	0
3	C	0
4	D	0
5	E	1
6	X	4
7	Y	0
8	Z	0
9		10
10		11
11		12
12		13
13		14
14		0

CODERINDEED

Thank You !!!