Home Work - 5

Name: Sai Rohit Kalyan Gandham
Student ID: 1002070724
Email ID: Sxg0724@mavs.uta.edu

11.2.1) Let's assume Simple Hashing Say it is uniform.

Sol)

* The key order of the key's are $\{k_1, k_2, k_3 \ldots k_n\}$

* let x_i be the No. of 1's, s.t, $1 \le i \le n$ and $h(k_j) = h(k_i)$
So x_i is the Expected No of times the key k_i is collided by those keys hashed afterwards

i.e $\sum_{j>i} \text{Probability}(h(k_j) = h(k_i)) = \sum_{j>i} 1/m = (n-i)/m$.

* The No. of collision is the addition / Sum of the No. of collisions for each possible Smaller Element in the collisions

i.e Expected No. of collision is

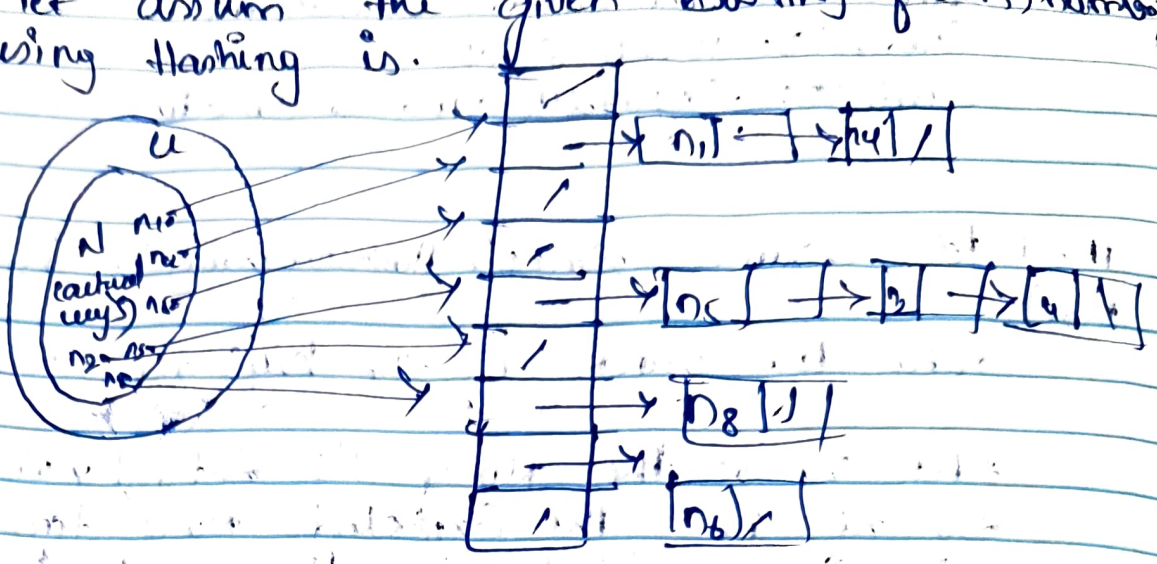
$$\sum_{i=1}^{n} \frac{n-i}{m} = \frac{n^2 - n(n+1)/2}{m}$$

$$= \frac{n^2 - n}{2m}$$

*

11.2-5)
Sol)

* let assum the given sorting of (n) numbers using Hashing is.



So each hash-table slot T(j) contains a linked list of cells with keys for that whose hash value is j

Eg $h(n_1) = h(n_4)$ and $h(n_5) = h(n_2) = h(n_3)$.

* So j contains a pointer to the head of the list of all stored Elements that hash to j; if there are no such Elements slots j contains NIL

* Chained_Hash_Insert (T, n):
insert n at the head of list $T[h(key(n))]$

* Chained_hash_Search (T, n)
Sears for an Element with key n in a list $T[h(key(n))]$
So Now We know that

① Worst running time for insertion is O(1), as insertion process is fast because we assume that Element (n) is inserted to

not present in table.

② worst running time for searching is proportional to the length of list.

③ deletion of an Element (or) can be done in O(1) as list are in linked already.

11.34 Explantion-1 : for each key1 is as long characters
Sol) = =

thus to compare key's at Every node we need to perform a string comparision operation which is very costly and time consuming. Instead we generate a hash value for the key ie., generate a numeric value for each string we are searching for and comparing hash values b(k) along the length of the list, which turns out to be numeric values and Comparison is faster.