International Journal of Advance Research in Science and Engineering
Vol. No.6, Issue No. 04, April 2017
www.ijarse.com
IJARSE
ISSN (O) 2319 - 8354
ISSN (P) 2319 - 8346

COLLISION RESOLUTION USING SEARCH TREE IN HASHING

Rajeev Ranjan Kumar Tripathi[1],Shubham Rai[2],

Shivam Kumar Dwivedi[3]

[1]Assistant Professor, Computer Science and Engineering,

Buddha Institute of Technology, GIDA, Badgahan, Gorakhpur , Uttar Pradesh, (India)

[2,3] Students, Computer Science and Engineering,

Buddha Institute of Technology, GIDA, Badgahan, Gorakhpur, Uttar Pradesh, (India)

ABSTRACT

In hashing, records are always placed at computed address. Record is tightly bound with the address. If collision occurs, open addressing and chaining is used. Open addressing and chaining offer loose binding in between record and the computed hash code. When deletion operation is performed additional information is maintained to avoid any ambiguity. In proposed scheme search trees are being used. Search trees are more efficient than conventional lists. If table offers sufficient space for allocation of trees, they are maintained in the table else they are kept outside of table. On requirement additional tables are also used to store record but the scheme eliminates such requirements. Analysis shows that proposed scheme is more efficient than the conventional approaches.

Keywords: Hashing,Collision,Open Addressing,Chaining,Search Tree

I. INTRODUCTION

Searching a record from a collection of records is a tedious job when records are stored randomly. Generally linear search and binary search operations are used when searching is required. In linear searching, every record is compared against the key and it is stopped when one of the two is achieved, either record is found and search is declared successful or we come at end of the table. Linear search is directly proportional to the size of table. As insertion operation is performed, size of table increases and linear search becomes more costly [4-7]. In binary search records are placed in an order they may be in ascending order or in descending order. At every unsuccessful comparison search space area is halved and thus Binary search is more efficient than linear search. Hashing is a refinement of searching where records are stored in a table at an address which is a function of a key.

International Journal of Advance Research in Science and Engineering
Vol. No.6, Issue No. 04, April 2017
www.ijarse.com
IJARSE
ISSN (O) 2319 - 8354
ISSN (P) 2319 - 8346

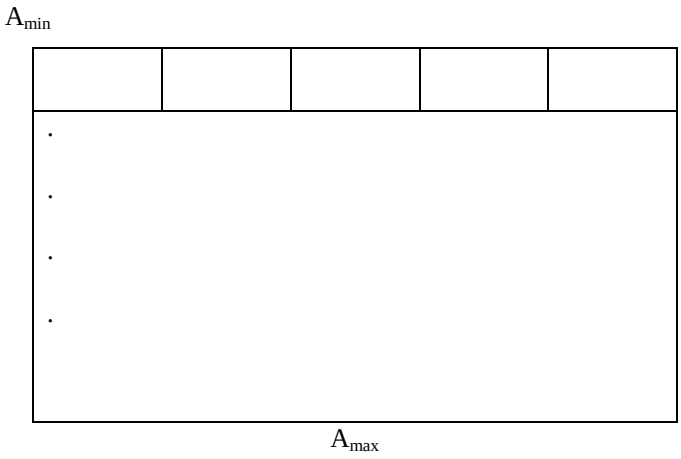

Fig. 1. Table T with its starting address A_{min} and ending address A_{max}.

Let we have a table T with A_{min} as starting address and A_{max} as an ending address. Let we want to store a record R with a key K[1-3]. This record may come at following addresses:

- At A_{min}
- At A_{max}
- In between A_{min} and A_{max}.

All addresses before A_{min} and all addresses after A_{max} are not valid for our purpose because table is starting from A_{min} and ending at A_{max}. To perform hashing a hash function H is applied on key K as:

$$A=H(K) \qquad (1)$$

Where, A is the address where record R has to store in the table. Now computed address A is said to be valid address if $A_{min}<=A<=A_{max}$. This property ensures that record will be always stored in the table. Let we have two records R_1 and R_2 with the keys K_1 and K_2. After applying hash functions on these two we get two addresses A1 and A2 as:

$$A_1=H(K_1) \qquad (2)$$

$$A_2=H(K_2) \qquad (3)$$

If $K_1 \neq K_2$ then $A_1 \neq A_2$. If this property is violated, collision is declared. To resolve collision several techniques were proposed and are used. Next section is shedding light on collision resolution techniques [6-7].

II. COLLISION RESOLUTION TECHNIQUES

Performance of collision resolution technique is measured in term of probes (comparisons). Generally probes depend on load factor, λ and it is defined as:

$$\lambda = n/m \qquad (4)$$

Where, n=Total number of records., m=total number of available locations in the table. If $\lambda \rightarrow 1$, records in the table appear as clusters. On the basis of λ, we define $S(\lambda)$ and $U(\lambda)$. The $S(\lambda)$ denotes average number of probes for a successful search and $U(\lambda)$ denotes average number of probes for an unsuccessful search. If collision does not take place, records are always placed at the address A which is computed by the hash function H. In other words record is tightly bound with A. If collision takes place record is not placed on computed address A and it is placed on some other address which is found available and this entire process is dependent on the schemes

IJARSE
ISSN (O) 2319 - 8354
ISSN (P) 2319 - 8346

being used. One thing is clear that in case of collision record R is loosely bound with the computed address and that's why this is often known as open addressing. Next sub section is shedding light on various types and modifications of open addressing scheme.

2.1 Linear Probing

Let we have a table T with m slots for storage. A new record R comes with the key K for storage. H (K) is computed for this record R but the computed address H (K) is already occupied by another record. To accommodate R, we start linear search for an empty slot in T as H (K) +1, H (K) +2 and so on. On availability of first empty slot, record R is placed there. Performance of this technique is given as:

$$S(\lambda) = \frac{1}{2}\left(1 + \frac{1}{1-\lambda}\right) \quad (5)$$

$$U(\lambda) = \frac{1}{2}\left(1 + \frac{1}{(1-\lambda)^2}\right) \quad (6)$$

Where λ is load factor. If λ is greater than 50% records in table form clusters. Further this clustering increases the average searching times. To minimize clustering and to means minimize the searching cost two modifications were proposed: *Quadratic Probing* and *Double Hashing*.

- In quadratic probing instead of searching empty slots linearly, we search as:
  $H(K), H(K)+1^2, H(K)+2^2, H(K)+3^2, H(K)+4^2, \ldots H(K)+i^2, \ldots$
  If m, number of slot in table T is a prime number then the above sequence accesses half of the locations in T.
- In double hashing we adopt one more hash function H^1 such that it gives value always less than m (total number of available slots). Here we search empty location to accommodate R as:
  $H(K), H(K)+H^1(K), H(K)+2H^1(K), H(K)+3H^1(K), \ldots$
  If number of slots available in T is a prime number then this scheme accesses all the locations of T.
  Open addressing scheme incurs extra cost when a record is deleted.

2.2 Chaining

In chaining we maintain two lists: one for address (calculated hash code) and second one for the values (records). To store values we have two options: first one is to maintain a list based on contiguous memory allocation and the second one to maintain linked list, based on non contiguous memory allocation. If collision occurs for a record, instead of searching a vacant slot as we do in linear probing, we insert this one into the second list. This second list always resides into the table. In this scheme hash code is kept same. Performance of chaining in terms of average number of probes is given as:

$$S(\lambda) \approx 1 + \frac{\lambda}{2} \quad (7)$$

$$U(\lambda) \approx \lambda + e^{-\lambda} \quad (8)$$

Where, S (λ) is number of probes in successful search and U (λ) is number of probes consumed in unsuccessful search and λ denotes load factor. Let us consider a case where we have a table that can contain 10 addresses starting from 01 to 10.Following sequence of keys are given as: 5, 6,15,42,34 and 23. We have to apply here division method. Division method is applied as:

IJARSE
ISSN (O) 2319 - 8354
ISSN (P) 2319 - 8346

$$H(K) = K \bmod m + 1$$

Where *k* is the key and *m* is prime number which is kept closer to the number of available addresses. In our case I have kept it 7. Then H (k) will be as:

TABLE I. RECORDS WITH HASH ADDRESSES

Records	5	6	15	42	34	23
H(k)	6	7	3	7	7	3

Collision takes places for 6, 42, 34 and 15, 23.

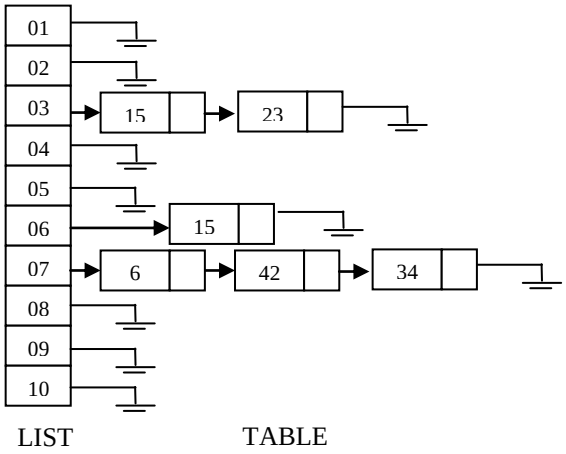

Fig.2. Conventional chaining

The linked list appearing in the above figure is being maintained in the table T where records have to be placed. The LIST is kept at some other location in memory. Let someone wants to search 34. The LIST is consulted and the attached list residing in the table T is linearly traversed now searching will take more time if length of list is increased (when collisions becomes very often)[4-5].

III. MOTIVATION FOR USING SEARCH TREE

Reducing the cost of searching is the main motivational factor behind the proposed scheme.

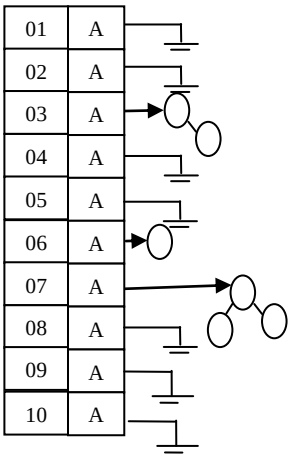

Fig.3. Proposed scheme

IJARSE
ISSN (O) 2319 - 8354
ISSN (P) 2319 - 8346

This paper suggests to use Binary search tree, AVL tree or B-tree as per requirement However AVL tree is more efficient in comparison to Binary search tree. If records are very large in volume and collision rate is very high then this paper suggests to use B-tree instead of using AVL tree. In proposed scheme, multiple searching trees are maintained in the memory at different memory locations.Root addresses of the trees are to be maintained with the LIST only. As we are maintaining this list in primary memory, switching from one location to another location is less expensive. Now we can virtually see the augmented LIST along with the addresses of trees as a table and records are actually stored at some other places in the main memory if table T does not offer enough space to accommodate the tree else trees are maintained in the table. When this scheme is applied with the chaining logically it appears as in figure.Nodes appearing in the circles are representing searching tree. The 'A' is representing the root address of the searching tree. If a record is deleted then its reflection will take place only in the tree. There is no need to maintain any additional information about the deleted node. Performance analysis of the proposed scheme is examined by the searching cost of linked list that we maintain in the conventional chaining and among the searching tree. In search tree, searching is more efficient than linear search.

IV. CONCLUSION

Deletion of a record is more tedious task in the conventional chaining process and in open addressing scheme. In case of collisions linear search is to perform for both: insertion and deletion of records. If table offers sufficient space to accommodate trees, they are maintained in the table else allocated somewhere else outside the table. As linear search is deprecated in the proposed scheme hence instead of consuming cost which is proportional to n, this scheme consumes cost proportional to log n, where n is number of keys causes collision. Additional information is not maintained when deletion is carried out in the proposed scheme. On the basis of above facts we can conclude that proposed scheme is more efficient over the conventional chaining.

REFERENCES

[1] A.C. Yao, "Uniform Hashing Is Optimal," Journal of ACM,32(1985),687-693.

[2] A.C. Yao, "A Note on The Analysis of Extensible Hashing," Communication of the ACM,25(1982),911-926.

[3] R.Morris, "Scatter Storage Techniques,"Communication of ACM,11(1968),38-44.

[4] B.J.McKenzie,R.Harries and T.Bell, "Selecting a Hashing Algorithm," Software-Practice and Experience,20(1990),209-224.

[5] J.L. Carter and M.N.Wegman, "Universal Classes of Hash Functions," Jouranl of Computer and System Sciences,18(1979),143-154.

[6] Rajeev Ranjan Kumar Tripathi, Deepak Singh Dikhit, Rajesh Kumar Singh , "Collision Free Cichelli Perfect Hash Function" at ACEIT2016 at Integral University Lucknow on 12 March-2016.

[7] Mark Allen Weiss, Data Structures and Algorithm Analysis in C,Second Edittion,Pearson1996.