Homework2 Written

Wen Fan (wf85), Zhidong Liu (zl479), Siyu Zhu (sz432)

March 2016

1 Hashing Indices

1.1 Linear Hashing is bigger

Suppose data entries 0*,1*,2*,3*,5*,6*,7*,9*,10*,11*,12*,14*,15*,18*,19*,22*,27* have been inserted into a linear hash table:

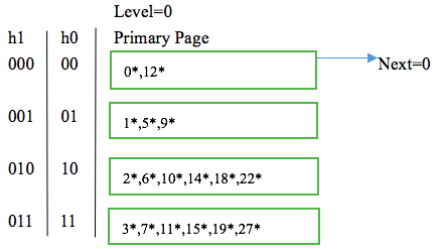

Figure 1: Initial Hash Table

Now we want to insert another 2 data entries: 26* and 31*. After the insertion, linear hash becomes:

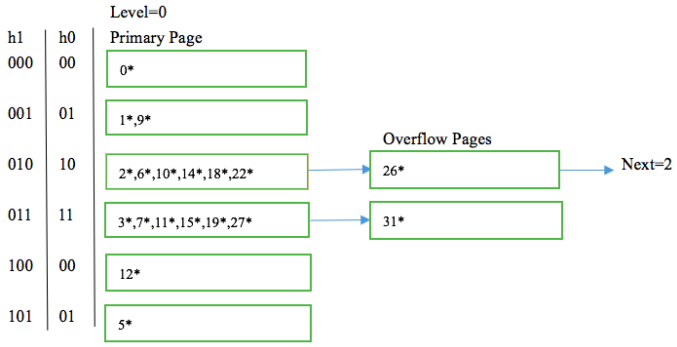

Figure 2: Insert entries

When 26* is inserted, a split is triggered because overflow happens. The first bucket splits into 2 buckets and the Next pointer becomes 1. For the same reason, the second bucket splits into 2 buckets when 31* is inserted and the value of Next becomes 2. So we have 8 pages in the end. For the Extendible Hash, after 31* is inserted, the hash table is shown below:

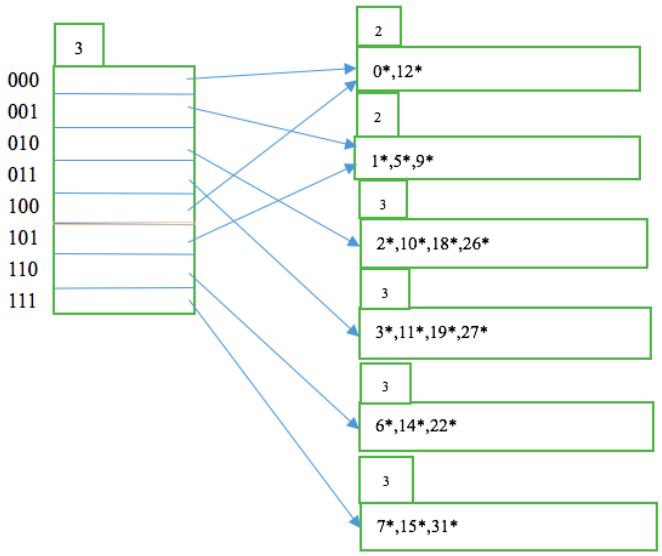

Figure 3: After entry 31* is inserted

Because the Hash indexes directory occupies a single page, the whole hash table in this situation occupies 7 pages, which means that Linear Hashing has more pages.

1.2 Extendible Hashing is bigger

Suppose data entries: 0*,1*,4*,5*,7*,8*,9*,10*,12*,13*,15*,16*,17*,19*,20*,21*,24* are inserted into hash table in order. For the Extendible Hashing, after insertion of entry 21*, the hash table is shown as below:

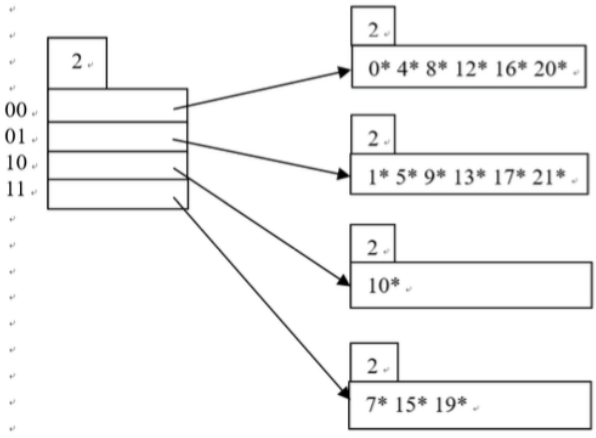

Figure 4: Initial extendible Hash Table

Then data entry 24* is inserted, which makes the first bucket split and the directory double:

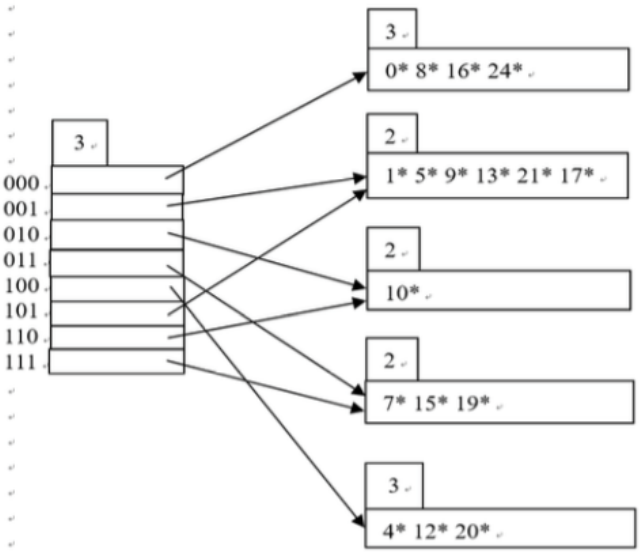

Figure 5: Data entry 24* inserted

Because an Extendible Hashing Index's directory occupies a single page, the whole hash table occupies 6 pages after the insertion of all the data entries.
For the Linear Hashing, after insertion of entry 21*, the hash table is shown as below:

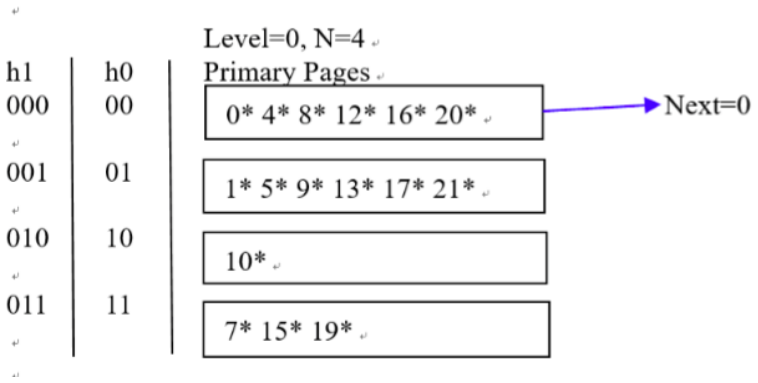

Figure 6: For Linear Hashing, after insertion of entry 21*

Then data entry 24* is inserted, which makes the first bucket overflow and the split is triggered:

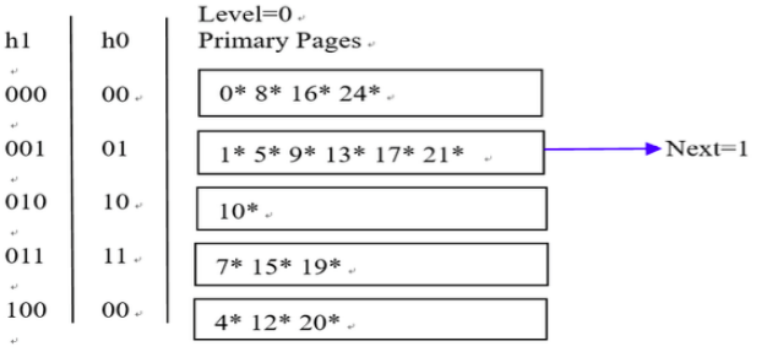

Figure 7: After insertion of entry 21*

In this situation, the whole hash table occupies 5 pages after the insertion of all the data entries. Which means that the Extendible Hashing has more pages.

2 The Cost of Joins

Suppose the number of tuples and pages of R are PR and M respectively. The number of tuples and pages of S are PS and N respectively. Let $Num(R)$ = total tuples of R, $Num(S)$ = total tuples of S.

- (a) Index Nested Loop Join (Clustered B+ Tree Index on S.A)

$$M + 7Num(R)$$

Because the B+ tree is clustered, the cost of retrieving matching S tuples is 1 I/O. The cost of joining the relation is $(M + M * PR * (6 + 1)) = M + 7Num(R)$ I/Os.

- (b) Index Nested Loop Join (Unclustered B+ Tree Index on R.A)

$$N + Num(S) * [Num(R)/Num(S) + 6]$$

Because the distribution on the foreign keys is uniform, every S has (M*PR)/(N*PS) R. Cost of retrieving matching tuples in R is (M*PR)/(N*PS) I/Os.
So the cost of joining the relation is $N+N*PS*((M*PR)/(N*PS)+6) = N + Num(S) * [Num(R)/Num(S) + 6]$ I/Os.

- (c) Index Nested Loop Join (Hash Index on S)

$$(M + Num(R) * 2.5)$$

If the Hash Indexes are clustered, cost of retrieving S tuples is 1 I/O. In this situation, the cost of joining the relation is $(M + M * PR * 2.5) = (M + Num(R) * 2.5)$ I/Os.
If the Hash Indexes are unclustered, cost of retrieving S tuples is (N*PS)/(M*PR) I/Os. In this situation, the cost of joining the relation is $M + Num(R) * [1.5 + Num(S)/Num(R)]$ I/Os.

- (d) Sort-Merge Join

$$5(M + N)$$

The cost to sort R and S are respectively as below:

$$2pass * 2M/pass = 4M$$

$$2pass * 2N/pass = 4N$$

Merge scan cost $M + N$ Therefore, the cost of Sort-Merge Join is $4M + 4N + M + N = 5(M + N)$

3 Least Expensive Evaluation

3.1 (a)

- (I). Using no index, Just scan the file. Cost = 56000 I/Os
Just scan 56000 pages. The corresponding cost is 56000 I/Os.
We can rule out the indexes method
The condition is age>10, there are two indexes involving age:
1. the unclustered B+ tree index on age
2. the clustered B+ tree index on $< location, age >$.
For 1, in the worst case, retrieving each tuple requires navigating the index root-to-leaf (2 I/Os). Because the value of RF is 0.1, in this situation, the cost is $56000 * 32 * 0.1 = 179200$ I/Os.
For 2, age is not a prefix of the search key. So this index is not viable.

- (II). Use the clustered B+ tree index on $< location, age >$, cost = 6536 I/Os
  Use the clustered B+ tree index on $< location, age >$. The corresponding cost includes:
  1. Traverse index to find the first leaf page: 2 I/Os.
  2. Scan leaf level pages to find data entries: $56000 * (20bytes/120bytes) * 0.1$
  3. Retrieve tuples: $56000 * 0.1$
  Total cost =

$$2 + 56000 * (20bytes/120bytes) * 0.1 + 56000 * 0.1 = 6536$$

  Rule out other methods:
  If using no index, cost = 56000
  If using unclustered, cost = $56000 * 32 * 0.1 = 179200$ I/Os
  They are both larger than 6535.

- (III). Sequential scan. Cost = 56000 I/Os.
  Reason: Since we do not have index on toypreference and the best scheme for the 1st condition is sequential scan, the best scheme for this condition is still sequential scan.

- (IV). Use the clustered B+ tree index on $< location, age >$, cost = 6536 I/Os
  Use the clustered B+ tree index on $< location, age >$. The corresponding cost includes:
  1. Traverse index to find the first leaf page: 2 I/Os.
  2. Scan leaf level pages to find data entries: $56000 * (20bytes/120bytes) * 0.1$
  3. Retrieve tuples: $56000 * 0.1$
  Total cost =

$$2 + 56000 * (20bytes/120bytes) * 0.1 + 56000 * 0.1 = 6536$$

  Rule out other possibilities:
  Sequential Scan, cost = 56000
  Unclustered B+ tree index on age, cost =

$$2 + 56000 * (20bytes/120bytes) * 0.1 + 56000 * 0.1 * 32 = 180136$$

  Unclustered hash index on location, cost = $56000 * 32 * 0.1 = 179200$
  Therefore, clustered B+ tree index on $< location, age >$ is the best method

3.2 b

- (I). The least expensive method is to use unclustered B+ tree. The cost is 5602 I/Os.
  Since that all we need to do is computing the average ages of qualifying tuples, there is no need to retrieve data records in the unclustered B+

tree. We can just traverse tree from root to leaf (2 I/Os) and then traverse the qualified data entries, the cost is $56000 * (20bytes/120bytes) * 0.1 = 934$. Threfore the total cost is 936 I/Os which is obviously smaller than sequential scanning cost 56000.

- (II). The least expensive method is to use clustered B+ tree index on $< location, age >$. The cost is 936 I/Os.
  The age attribute is on the index. So we do not need to retrieve data records. Just like (I), all we need to do is traversing tree from root to leaf (2 I/Os) and then traverse the qualified leaf pages. The total cost is still 936 I/Os.

- (III). The least expensive method is sequential scan, which takes 56000 I/Os.
  There is no index with the attribute toypreference. This condition is quite similar as 2.3(a)(III), where we need to scan the whole file to retrieve all qualified entries. So the total cost is still 56000 I/Os.

- (IV). The least expensive method is to use clustered B+ tree index on $< location, age >$. The cost is 6536 I/Os.
  Similar to (II). But here we need to retrieve records to specify attribute toypreference.
  Therefore, the cost is:

$$2 + 56000 * (20bytes/120bytes) * 0.1 + 56000 * 0.1 = 6536$$