EXTENDIBLE HASHING II

Contents of today's lecture:

- Insertions: a closer look at bucket splitting.
- Deletions in extendible hashing.
- Extendible hashing performance.

Reference: Folk, Zoellick and Riccardi, File Structures, 1998. Sections 12.4,12.5.

Insertions: bucket splitting re-visited

In some cases, several splits are necessary in order to accommodate a new key to be inserted. Let us consider one such case.
Consider the following example:

```
    +------+
000 |      |---+
    +------+   +-->+--------+
001 |      |------>|  2,6   |  bucket A
    +------+   +-->+--------+
010 |      |---|
    +------+   |
011 |      |---+   +--------+
    +------+    +->| 1, 9   |
100 |      |----+  +--------+
    +------+       +--------+
101 |      |------>|   5    |
    +------+       +--------+
110 |      |------>+--------+
    +------+   +-->|  3,7   |
111 |      |---+   +--------+
    +------+
```

Insert key 10
Bucket A must split.
Note that A has depth 1, since one digit (namely 0) determines whether a key should be placed in A.

The first split will create a new bucket A1, so that A and A1 have now depth 2.
With depth 2, two digits will be examined to determine whether a key goes to A (00*) or A1 (01*).
After redistribution, the structure becomes:

```
    +-----+       +-------+
000 |     |------>|(empty)|  bucket A
    +-----+  +--->+-------+
001 |     |--+
    +-----+       +-------+
010 |     |------>|  2,6  | bucket A1
    +-----+   +-->+-------+
011 |     |---+   +-------+
    +-----+    +->| 1, 9  |
100 |     |----+  +-------+
    +-----+       +-------+
101 |     |------>|   5   |
    +-----+       +-------+
110 |     |------>+-------+
    +-----+   +-->|  3,7  |
111 |     |---+   +-------+
    +-----+
```

A becomes empty and A1 get both keys.
After this split, the bucket pointed by 010, namely A1, is still full.
So, we must split A1.
Bucket A1 has depth 2. It will split into buckets A1 and A2, both with depth 3.

After redistribution, the structure becomes:

```
    +-----+       +-------+
000 |     |------>|(empty)|
    +-----+  +--->+-------+
001 |     |--+    +-------+
    +-----+   +-->|   2   | bucket A1
    |     |---+   +-------+
010 |     |       +-------+
    +-----+   +-->|   6   | bucket A2
011 |     |---+   +-------+
    +-----+       +-------+
100 |     |------>| 1, 9  |
    +-----+       +-------+
101 |     |---+   +-------+
    +-----+   +-->|   5   |
110 |     |---+   +-------+
    +-----+   +-->+-------+
111 |     |------>|  3,7  |
    +-----+       +-------+
```

Now the bucket pointed by 010 is not full, so that we can insert key 10 there.

After inserting key 10, the structure becomes:

```
    +-----+       +-------+
000 |     |------>|(empty)|
    +-----+  +--->+-------+
001 |     |--+    +-------+
    +-----+   +-->|  2,10 | bucket A1
    |     |---+   +-------+
010 |     |       +-------+
    +-----+   +-->|   6   | bucket A2
011 |     |---+   +-------+
    +-----+       +-------+
100 |     |------>| 1, 9  |
    +-----+       +-------+
101 |     |---+   +-------+
    +-----+   +-->|   5   |
110 |     |---+   +-------+
    +-----+   +-->+-------+
111 |     |------>|  3,7  |
    +-----+       +-------+
```

Insertion Algorithm

```
Insert (key) {

 indexKey = index in the directory where key
            should be placed
 Let A = bucket pointed by indexKey

 if (bucket A is not full) then
     Insert key into bucket A
 else  { // bucket is full so must split
   if (bucket depth is equal to directory depth) then
       { double directory size,
         re-adjusting bucket pointers;
       }
   split bucket A;
   Insert (key);  // recursive call to itself
   }
}
```

Deletions in Extendible Hashing

When we delete a key we may be able to **combine buckets**.

After the bucket combination, the directory may or may not be **collapsed**.

Combination and collapsing is a recursive process.

Bucket combination

A bucket may have a **buddy bucket**, that is a bucket that may be combined to it.
In the following example, which of the following buckets is such that when deleting a key from it, the bucket can be combined to another bucket?

```
                        bucket size = 2 records
    +-----+
000 |     | -> bucket 1 (1 record)
    +-----+
001 |     | -> bucket 2 (2 records)
    +-----+
010 |     |--+
    +-----+  +--> bucket 3 (1 record)
011 |     |--+
    +-----+
100 |     | -> bucket 4  (1 record)
    +-----+
101 |     | -> bucket 5  (2 record)
    +-----+
110 |     | -> bucket 6  (2 record)
    +-----+
111 |     | -> bucket 7  (2 record)
    +-----+
```

Answer: buckets 1&2, buckets 4&5
Why not buckets 3, 6, 7 ?

Buddy buckets

One bucket can only be combined with its **buddy bucket**.

A bucket and its buddy bucket must be distinct buckets pointed to by sibling nodes in the directory trie.

The index of the buddy bucket is obtained by switching the last bit of the bucket's index.

How to test if the bucket pointed out by index i has a buddy bucket?

```
i = 101
swap the last bit of i: iSwap = 100
if  (directory[i] != directory[iSwap]) then
     bucket pointed by direct[iSwap]
     is the buddy bucket
else there is no buddy bucket.
```

Directory collapsing

We can collapse a directory if every pair of "sibling" indexes point to the same bucket.

Check if we can collapse a directory as follows:

```
canColapse = false;
if (directory size is larger than 1) then {
    canColapse = true;
    for i=0 to (size/2 -1) do
      if (directory[2*i] != directory[2*i+1]
      then { canColapse = false;
             exit for;
           }
}
```

If `canColapse` is true then we can collapse the directory in the following way:

- Create a new directory with half the size
- Copy the pointers from positions 0 to 0, 2 to 1, . . . , (size-2) to (size/2 -1).

Deletion Algorithm

`Delete (key):`

1. Search for `key`.
2. If `key` not found, stop.
3. If `key` was found then remove `key` from its bucket `b` , in the following way:
   - physically remove `key` from bucket `b`
   - run `tryCombine(b)` in order to try to combine buckets and shrink directory.

Description of `tryCombine(b)`:

```
if (b has a buddy bucket c)  then
   if ( b.numkeys + c.numkeys) <= maxkeys {
       combine buckets b and c into bucket b.
       try to collapse the directory;
       if (directory was collapsed) then
          tryCombine(b);
       }
```

Deletion Example

Delete 5 from the following structure:

```
    +------+
000 |      |---+
    +------+   +-->+--------+
001 |      |------>|  2, 6  |
    +------+   +-->+--------+
010 |      |---|
    +------+   |
011 |      |---+   +--------+
    +------+    +->| 1, 9   |
100 |      |----+  +--------+
    +------+       +--------+
101 |      |------>|   5    |
    +------+       +--------+
110 |      |------>+--------+
    +------+   +-->|(empty)|
111 |      |---+   +--------+
    +------+
```

Search for 5 and physically remove it from its bucket `b`:

```
    +-----+
000 |     |---+
    +-----+   +-->+-------+
001 |     |------>|  2, 6 |
    +-----+   +-->+-------+
010 |     |---|
    +-----+   |
011 |     |---+   +-------+
    +-----+    +->| 1, 9  | <<<<< buddy bucket c
100 |     |----+  +-------+
    +-----+       +-------+
101 |     |------>|(empty)|  <<<< bucket b
    +-----+       +-------+
110 |     |------>+-------+
    +-----+   +-->|(empty)|
111 |     |---+   +-------+
    +-----+
```

Run `tryCombine(b)`.

Since `b` contains 0 keys and buddy bucket `c` contains 2 keys, buckets `b` and `c` can be combined...

After combining `c` and `b` we get:

```
    +-----+
000 |     |---+
    +-----+   +-->+-------+
001 |     |------>|  2, 6 |
    +-----+   +-->+-------+
010 |     |---|
    +-----+   |
011 |     |---+
    +-----+
100 |     |------>+-------+
    +-----+       |  1, 9 |  b = buckets b,c combined
101 |     |------>+-------+
    +-----+
110 |     |------>+-------+
    +-----+   +-->|(empty)|
111 |     |---+   +-------+
    +-----+
```

The directory can be collapsed...

The collapsed directory:

```
   +-----+       +-------+
00 |     |---+-->|  2, 6 |
   +-----+   |   +-------+
01 |     |---+
   +-----+       +-------+
10 |     |------>|  1, 9 | bucket b
   +-----+       +-------+
11 |     |------>+-------+
   +-----+       |(empty)| buddy bucket c
                 +-------+
```

The recursive call to `tryCombine(b)` now combines `b` and its buddy bucket `c`:

```
   +-----+       +-------+
00 |     |---+-->|  2, 6 |
   +-----+   |   +-------+
01 |     |---+
   +-----+       +-------+
10 |     |---+-->|  1, 9 | b = buckets b, c combined
   +-----+   |   +-------+
11 |     |---+
   +-----+
```

The directory can be collapsed again...

The collapsed directory:

```
   +-----+      +-------+
 0 |     |---+-->|  2, 6 | buddy bucket c
   +-----+      +-------+
 1 |     |--+   +-------+
   +-----+  +--->|  1, 9 | bucket b
                +-------+
```

The next recursive call to `tryCombine(b)` would detect that `b` cannot be combined with its buddy bucket `c`.

The deletion operation has been completed.

Extendible Hashing Performance

Time Performance (Worst Case):

operation	directory kept in main memory	directory kept in disk
search	1 disk access	2 disk accesses
insertion $d=$ dir size after insertion	$O(\log d)$ disk accesses	$O(d)$ disk accesses
deletion $d=$ dir size before deletion	$O(\log d)$ disk accesses	$O(d)$ disk accesses

The great advantage of extended hashing is that its search time is truly $O(1)$, independently from the file size.

In ordinary hashing, this complexity depends on the packing density, which could change after many insertions.

Space Performance:

1. **Space Utilization for Buckets**

   Here we describe results found by Fagin, Nievergelt, Pippinger and Strong (1979).

   Experiments and algorithm analysis by these authors have shown that the packing density for extendible hashing (space utilization for buckets) fluctuates between 0.53 and 0.93.

   They have also shown that, if:

   r = number of records
   b = block size
   N = average number of buckets

   Then,
   $$N \approx \frac{r}{b \cdot ln2}.$$
   Therefore,

   packing density $= \frac{r}{b \cdot N} \approx ln2 \approx 0.69.$

   So, we expect the average bucket utilization to be 69%.

2. Space Utilization for the Directory

Given r keys spread over some buckets, what is the expected size of the directory, assuming random keys are inserted into the table?

Flajolet (1983) addressed this problem by doing a careful analysis, in order to estimate the directory size.

The following table shows his findings:

b	5	10	20	...	200
r					
10^3	1.50 K	0.30 K	0.10 K		0.00 K
10^4	25.60 K	4.80 K	1.70 K		0.00 K
10^5	424.10 K	68.20 K	16.80 K		1.00 K
10^6	6.90 M	1.02 M	0.26 M		8.10 K
10^7	112.11 M	12.64 M	2.25 M		0.13 M

1 K = 10^3, 1 m = 10^6. (From Flajolet 1983)