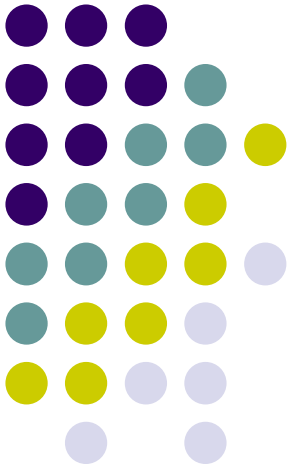
File Structures

Ch12. B. Extendible Hashing_Deletion



2020. Spring

Instructor: Joonho Kwon

jhkwon@pusan.ac.kr

Data Science Lab @ PNU

datalab

data science laboratory

Slides are based on the textbook.

Outline

- 12.4 Deletion
- 12.5 Extendible hashing performance
  - Skipped
- 12.6 Alternative approaches
  - skipped

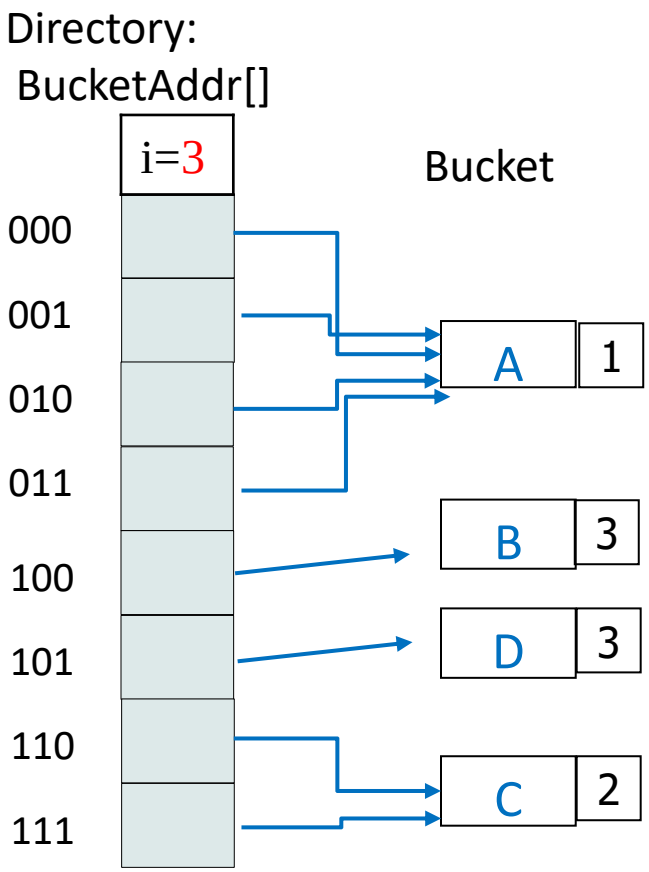
Overview of the deletion process (1/4)

- Extensible hashing: a dynamic structure
  - Grow and shrink files gracefully

- when do we combine buckets?
  - the definition of the triggering condition
  - which buckets can be combined?
    - buckets that are buddy buckets
    - In a B-tree case: sibling nodes

Overview of the deletion process (2/4)

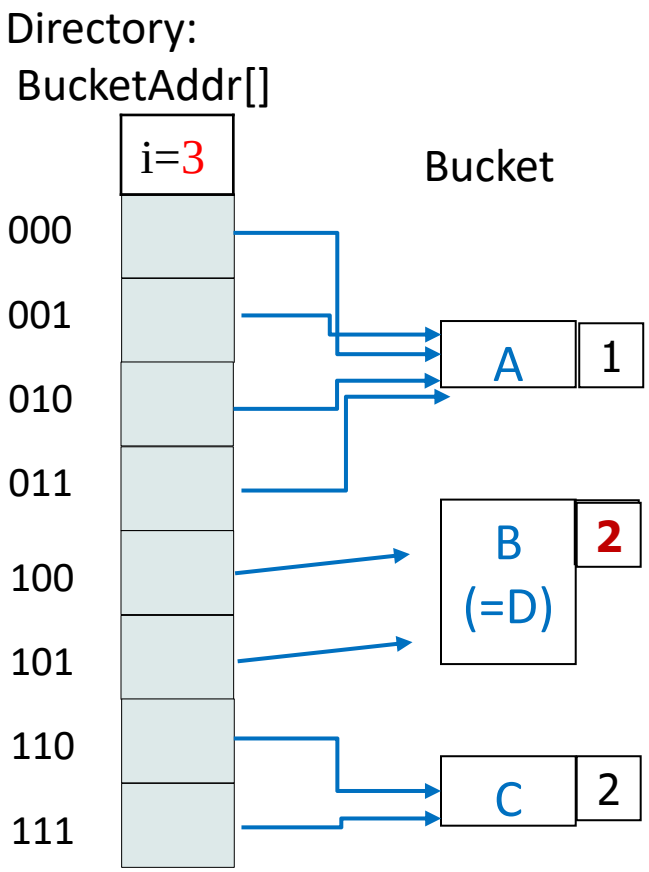
- Which are buddy buckets?
  - For a bucket A
    - None
  - For a bucket C
    - None
  - For buckets B and D
    - In the same configuration
      - ready to combine
    - → Buddy buckets



Overview of the deletion process (3/4)

- After combining B and D
  - Each of buckets has at least a pair of directory cells pointing to it
  - None of the buckets requires the dept of global depth

→ we can shrink the directory



Overview of the deletion process (4/4)

- Reducing the size of address spaces
  - Collapsing each adjacent pair of directory cells into a single cell
    - Easy, both cells in each pair point to the same bucket
  - Reversal of the directory splitting procedure

Buddy Bucket (1/2)

- Buddy buckets
  - immediate descendants of the same node in the trie
  - Pairwise siblings resulting from a split()
  - a bucket uses all the address bits in the directory = the bucket is at the lowest level of trie
  - When a bucket is at the outer edge of trie that it can have a single parent and a single buddy

Buddy Bucket (2/2)

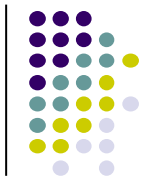
- Given a bucket with an address *uvwxy*,
  - where *u, v, w, x,* and *y* have values of either 0 or 1
  - **the buddy bucket**, if it exists, has the value *uvwxz*, such that

$$z = y\ XOR\ 1$$

  - the other bucket from split(), the buddy has the same address in all regards except for the last bit

- If enough keys are deleted, the contents of buddy buckets can be combined into a single bucket

Finding buddy buckets (1/2)



- Checking the directory depth and bucket depth
  - Dir.depth ==0 : only a single bucket --> no buddy
  - (local)depth < Dir.depth → no buddy
  - (local) Depth == Dir.depth → there exists a buddy

Finding buddy buckets (2/2)

- To get the buddy address
  - we flip the last bit with an exclusive OR (XOR)
  - return directory address of the buddy bucket

```
// find the directory address of the bucket that is paired with this
int Bucket::FindBuddy ()
{
    if (Dir.Depth == 0) return -1; // no buddy, empty directory
    // unless bucket depth == directory depth, there is no single
    // bucket to pair with
    if (Depth < Dir.Depth) return -1;
    int sharedAddress = MakeAddress(Keys[0], Depth); // address of any key
    return sharedAddress ^ 1; // exclusive or with low bit
}
```

Directory::Collapse() (1/2)

- Collapsing the Directory
  - Downsizing the directory
  - Benefits of deleting records

- Directory::Collapse
  - Make sure we are not the lower limit of directory
  - All directory size are evenly divisible by 2
  - Check whether each pair of directory cells point to different buckets or not
    - If yes, we cannot collapse

Directory::Collapse() (2/2)

```
// if collapse is possible, reduce size by half
int Directory::Collapse ()
{
    if (Depth == 0) return 0; // only 1 bucket
    // look for buddies that are different, if found return
    for (int i = 0; i < NumCells; i += 2)
        if (BucketAddr[i] != BucketAddr[i+1]) return 0;
    int newSize = NumCells / 2;
    int * newAddrs = new int [newSize];
    for (int j = 0; j < newSize; j++)
        newAddrs[j] = BucketAddr[j*2];
    delete BucketAddr;
    BucketAddr = newAddrs;
    Depth --;
    NumCells = newSize;
    return 1;
}
```

Implementing the deletion operations

- Operations for Deletion
  - Bucket::FindBuddy()
  - Directory::Collapse()
  - Directory::Remove()
  - Bucket::Remove()
  - Bucket::TryCombine()
  - Bucket::Combine()

Directory::Remove

- Directory::Remove()
  - Highest-level deletion operation
  - Find the key to be deleted
    - Not found: return failure
    - Found: Call Bucket::Remove()
      - Remove the key from the bucket

```
// remove the key and return its RecAddr
int Directory::Remove (char * key)
{
    int bucketAddr = Find(key);
    LoadBucket (CurrentBucket, bucketAddr);
    return CurrentBucket -> Remove (key);
}
```

Bucket::Remove

- Work in two steps
  - Step1 : removing the key from the bucket
    - Call TextIndex::Remove(key)
  - Step2: take places if a key is removed
    - Call Bucket::TryCombine() to see if deleting the key has decreased the size of the bucket

```
// remove the key, return its RecAddr
int Bucket::Remove (char * key)
{
    int result = TextIndex::Remove (key);
    if (!result) return 0; // key not in bucket
    TryCombine (); // attempt to combine with buddy
    // make the changes permanent
    Dir.StoreBucket(this);
    return 1;
}
```

Bucket::TryCombine (1/2)

- Checks to see if there is a buddy bucket
  - If not: done
  - If there is a buddy, and sum of keys in the bucket and its buddy <= size of a single bucket
    - Call Combine()

- after combining the buckets, call Directory::Collapse()
  - To see if the decrease in the number of buckets enables us to decrease the size of directory
  - Yes: Call TryCombine() recursively

Bucket::TryCombine (2/2)

```
// called after insert to combine buddies, if possible
int Bucket::TryCombine ()
{
    int result;
    int buddyIndex = FindBuddy ();
    if (buddyIndex == -1) return 0;// no combination possible
    // load buddy bucket into memory
    int buddyAddr = Dir.BucketAddr[buddyIndex];
    Bucket * buddyBucket = new Bucket (Dir, MaxKeys);
    Dir . LoadBucket (buddyBucket, buddyAddr);
    // if the sum of the sizes of the buckets is too big, return
    if (NumKeys + buddyBucket->NumKeys > MaxKeys) return 0;
    Combine (buddyBucket, buddyIndex);// collapse the 2 buckets
    result = Dir.Collapse ();
    if (result) TryCombine(); //if collapse, may be able to combine
    return 1;
}
```

Bucket::Combine

- Combining buckets means
  - use 1 less address bit to differentiate keys

```
// collapse this and buddy into a single bucket
int Bucket::Combine (Bucket * buddy, int buddyIndex)
{
    int result;
    // move keys from buddy to this
    for (int i = 0; i < buddy->NumKeys; i++)
    {// insert the key of the buddy into this
        result = Insert (buddy->Keys[i],buddy->RecAddrs[i]);
        if (!result) return 0;
    }
    Depth --;// reduce the depth of the bucket
    Dir . RemoveBucket (buddyIndex, Depth);
    return 1;
}
```

Q&A