CS143: Hash Index
What is a Hash Table?

• Hash Table
  – Hash function
    • \( h(k) \): key \( \rightarrow \) integer [0...n]
    • e.g., \( h(\text{Susan}) = 7 \)
  – Array for keys: \( T[0...n] \)
  – Given a key \( k \), store it in \( T[h(k)] \)

  \[
  \begin{array}{c|c}
    0 & \null \\
    1 & \text{Neil} \\
    2 & \null \\
    3 & \text{James} \\
    4 & \text{Susan} \\
    5 & \null \\
  \end{array}
  \]

  \( h(\text{Susan}) = 4 \)
  \( h(\text{James}) = 3 \)
  \( h(\text{Neil}) = 1 \)
Why Hash Table?

• Direct access
• Saved space
  – Do not reserve a space for every possible key

• How can we use this idea for a record search in a DBMS?
Hashing for DBMS (Static Hashing)

Disk blocks (buckets)

0
1
2
3
4

(search key □ h(key))
Issues?

- What hash function?
- Overflow?
- How to store records?
Record storage

1. Whole record

   h(key)

2. Key and pointer

   h(key)

   record

   key 1
Overflow and Chaining

- **Insert**
  - $h(a) = 1$
  - $h(b) = 2$
  - $h(c) = 1$
  - $h(d) = 0$
  - $h(e) = 1$

- **Delete**
  - $h(b) = 2$
  - $h(c) = 1$

Do we keep keys sorted inside a block?
Questions

• Q: Do we keep keys sorted inside a block?

• Q: How much space should we use for “good” performance?
What Hash Function?

• Example
  – Key = ‘x₁ x₂ ... xₙ’  n byte character string
  – Have b buckets
  – h: (x₁ + x₂ + ..... xₙ) mod b

• Desired property
  – Uniformity: same # of keys for every bucket

• Reference
  – “The Art of Computer Programming Vol. 3” by Knuth
Major Problem of Static Hashing

- How to cope with growth?
  - Data tends to grow in size
  - Overflow blocks unavoidable
Idea 1

- Periodic reorganization

- New hash function
- Complete reassignment of records
- Very expensive
Idea 2

- Why don’t we split a bucket when it overflows?

Insert: \( h(g) = 1 \)
Extendible Hashing (two ideas)

(a) Use \( i \) of \( b \) bits output by hash function

\[
h(K) \quad 00110101
\]

use \( i \) grows over time
Extendible Hashing (two ideas)

(b) Use directory that maintains pointers to hash buckets (indirection)
Example

- $h(k)$ is 4 bits; 2 keys/bucket

Insert 0111, 1010

\[
i = \begin{array}{c}
0 \\
1 \\
\end{array}
\]

\[
\begin{array}{c}
1 \\
0001 \\
1001 \\
1100 \\
\end{array}
\]
Example continued

Insert 0000

i = 2

```
00
01
10
11
1
2
0001
0111
2
1001
1010
2
1100
```
Example continued

\[ i = 2 \]

Insert 1011
Questions on Extendible Hashing

• Q: What will happen if we have a lot of duplicate search keys?
Duplicate keys

- Insert 0001
Questions on Extendible Hashing

• Can we provide minimum space guarantee?
Space Waste
Extendible Hashing: Deletion

- Two options
  - a) No merging of buckets
  - b) Merge buckets and shrink directory if possible
Bucket Merge Example

Can we merge \( a \) and \( b \)? \( b \) and \( c \)?

Can we shrink the directory?
Bucket Merge Condition

- Bucket merge condition
  - Bucket i’s are the same
  - First (i-1) bits of the hash key are the same

- Directory shrink condition
  - All bucket i’s are smaller than the directory i
Summary of Extendible Hashing

• Can handle growing files
  – No periodic reorganizations

• Indirection
  – Up to 2 disk accesses to access a key

• Directory doubles in size
  – Not too bad if the data is not too large
Hashing vs. Tree

• Can an extendible-hash index support?
  
  ```sql
  SELECT
  FROM R
  WHERE R.A > 5
  ```

• Which one is better, B+tree or Extendible hashing?
  
  ```sql
  SELECT
  FROM R
  WHERE R.A = 5
  ```