Implementation of Relational Operations (Part 2)

R&G - Chapters 12 and 14
An Alternative to Sorting: Hashing!

- **Idea:**
  - Many of the things we use sort for don’t exploit the *order* of the sorted data
  - e.g.: removing duplicates in DISTINCT
  - e.g.: finding matches in JOIN

- **Often good enough to match all tuples with equal values**

- **Hashing does this!**
  - And may be cheaper than sorting! (Hmmm...!)
  - But how to do it for data sets bigger than memory??
General Idea

• **Two phases:**
  – **Partition**: use a hash function $h$ to split tuples into partitions on disk.
    • Key property: all matches live in the same partition.
  – **ReHash**: for each partition on disk, build a main-memory hash table using a hash function $h_2$
Two Phases

- **Partition:**

- **Rehash:**
Duplicate Elimination using Hashing

- read one bucket at a time
- for each group of identical tuples, output one
Cost of External Hashing

cost = 4*[R] IO’s
Memory Requirement

- **How big of a table can we hash in two passes?**
  - B-1 “partitions” result from Phase 0
  - Each should be no more than B pages in size
  - Answer: B(B-1).
    
    *Said differently:*
    
    We can hash a table of size N pages in about $\sqrt{N}$ space

  - *Note: assumes hash function distributes records evenly!*

- **Have a bigger table?**  *Recursive partitioning!*
How does this compare with external sorting?
Cost of External Hashing

\[ \text{cost} = 4 \times [R] \text{ IO's} \]
Cost of External Sorting

Cost = 4*[R] IO’s
Memory Requirement for External Sorting

- **How big of a table can we sort in two passes?**
  - Each “sorted run” after Phase 0 is of size B
  - Can merge up to B-1 sorted runs in Phase 1
  - Answer: B(B-1).

  *Said differently:*
  We can sort a table of size N pages in about $\sqrt{N}$ space

- **Have a bigger table?** Additional merge passes!
So which is better??

• Based on our simple analysis:
  – Same memory requirement for 2 passes
  – Same IO cost

• Digging deeper ...

• Sorting pros:
  – Great if input already sorted (or almost sorted)
  – Great if need output to be sorted anyway
  – Not sensitive to “data skew” or “bad” hash functions

• Hashing pros:
  – Highly parallelizable (will discuss later in semester)
    • So is sorting, with some work
  – Can exploit extra memory to reduce # IOs (stay tuned...)
before we optimize hashing further ...

**Q:** Can we use hashing for JOIN?
Hash Join

Original Relation

Disk

B main memory buffers

OUTPUT

1

2

\cdots

B-1

Partitions

Disk

INPUT

hash function

\( h \)

Partitions of R & S

Disk

Hash table for partition

R_i (B-2 pages)

Join Result

Disk

\( \text{hash function } h \)

\( h_2 \)

Input buffer for S_i

Output buffer
Cost of Hash Join

- **Partitioning phase**: read+write both relations
  ⇒ \(2([R]+[S])\) I/Os

- **Matching phase**: read both relations, write output
  ⇒ \([R]+[S]+[\text{output}]\) I/Os

- Total cost of 2-pass hash join = \(3([R]+[S])+[\text{output}]\)

**Q:** what is cost of 2-pass *sort join*?

**Q:** how much memory needed for 2-pass *sort join*?

**Q:** how much memory needed for 2-pass *hash join*?
An important optimization to hashing

- Have $B$ memory buffers
- Want to hash relation of size $N$

If $B < N < B^2$, will have **unused memory**...
Hybrid Hashing

- **Idea:** keep one of the hash buckets in memory!

Q: how do we choose the value of $k$?
Cost reduction due to hybrid hashing

• Now:

![Graph showing cost vs number of passes with points at (N, 1) and (3N, 2) connected by a line]
Summary: Hashing vs. Sorting

- **Sorting pros:**
  - Good if input already sorted, or need output sorted
  - Not sensitive to data skew or bad hash functions

- **Hashing pros:**
  - Often cheaper due to hybrid hashing
  - For join: # passes depends on size of *smaller* relation
  - Highly parallelizable