One and Two Pass Algorithms

CS157B

Chris Pollett

Mar. 14, 2005.

Outline

- One Pass Algorithms
- Nested Loop Joins
- Two Pass Algorithms Based on Sorting

One Pass Algorithms -- Tuple at a Time Operations

- The basic format of these algorithms is: Read blocks of R one at a time into an input buffer, perform an operation on each tuple, and move result to the output buffer or next step in query process.
- Last day saw a one pass algorithm for union. It was also an example of a tuple at a time algorithm.
- Some other examples, are $\pi(R)$ and $\sigma(R)$.
- Cost will typically be B(R) if R is on disk.

One-Pass Algorithms for Unary, Full-Relation Operations

- Duplicate Elimination -- Read each block into memory. Use M-1 blocks to store tuples have seen so far. Before putting a tuple in the output block check if it is among those seen. To work this algorithm requires $B(\delta(R)) <= M-1$. To simplify future discussion, we will often approximate as $B(\delta(R)) <= M$.
- Grouping -- For MIN(a), MAX(a) can scan and keep copy of current min or max of group so far. For COUNT, add one for satisfying value. SUM and AVG are similarly handled. Need to have a bound like above on number of distinct groups.

One Pass Algorithms for Binary Operations

- For these operations need min(B(R), B(S)) <=M.
- Set Union -- Read S, do duplicate elimination and write to output. Then read R using same structure for duplicate elimination as used for S. Do duplicate elimination as write to output.
- Set Intersection -- Read S into M-1 buffers, build search structure. Read each block of R and for each tuple t in R check if t is in this structure from S before writing to output.
- Set Difference -- similar.
- Bag Intersection -- like set intersection, but need to count the number of occurrences of t in S and R and output the minimum of these two numbers.
- Bag Difference -- similar. R\S if number of occurrences of t in R > than the number of occurences of t in S make difference in count many copies into R\S.

More One Pass Algorithms for Binary Operations

- Product and Natural Join -- if one of R and S fits into main memory, just read it entirely into main memory. Then concatenate each of its tuples with the appropriate tuples from the other relation. Takes time B(R) + B(S).

Nested Loop Joins

- What if one of R or S doesn't fit into memory. How do we do a join or product then?
- Could do:

```
For each s in S do
    For each r in R  do
        if r and s join make a tuple t then output t;
```

- Notice this algorithm fits fairly well within the iterator framework.
- However, if one is not being careful this could take time T(R)T(S).
- We'll now discuss some improvements.

More Nested Loop Join

- Rather than being tuple oriented want to do our accesses on R and S in a block-wise manner.
- Want to store as much as we can of one of two relations in main memory. Assume B(S) <= B(R). Then can do:

```
For each M-1 blocks of S do begin
        read these blocks into main-memory buffers;
        organize their tuples into a structure whose search attributes are the common
            attributes of R and S;
        For each block b of R do begin
                for each tuple t of b do begin
                        find the tuples of S in main memory that join with t;
                        output the join of t with each of these tuples;
                end;
        end;
end;
```

- This is called nested block join . It takes time B(S) + B(S)B(R)/(M-1).

Two Pass Algorithms Based On Sorting

- We'll next consider some algorithms based on sorting. We'll assume sorting only takes two passes.
- Duplicate Elimination using sorting -- given R we sort R and output distinct values. Phase I creating sorted sublist of size M takes time B(R). Phase II can be done in one pass provided can merge all of these sublists together in one go. So B(R) must be less than M^2. In which case, total cost is 3B(R).
- Will give more algorithms next day.