Chair of applied computer science III
Prof. Dr. Guido Moerkotte
Email: moerkotte@uni-mannheim.de

University of Mannheim

Query Optimization — Exercise sheet 7

Exercise 1

As we already know, the number of non-symmetric csg-cmp-pairs (#ccp) depends on the query graph:

$$\#ccp^{chain}(n) = 1/6 * (n^3 - n)$$
$$\#ccp^{cycle}(n) = (n^3 - 2n^2 + n)/2$$
$$\#ccp^{star}(n) = (n-1)2^{n-2}$$
$$\#ccp^{clique}(n) = (3^n - 2^{n+1} + 1)/2$$

Luckily, we don't have to store all of them in our DP-table. For all DP-based algorithm that don't consider cross products, we store only the cheapest plan (seen so far) for each connected sub graph.

$$\#csg^{chain}(n) = n(n+1)/2$$
$$\#csg^{cycle}(n) = n^2 - n + 1$$
$$\#csg^{star}(n) = 2^{n-1} + n - 1$$
$$\#csg^{clique}(n) = 2^n - 1$$

Exercise 1 a)

For a star query with with $n = 20$ relations, how many plans do you have to store in your DP-table? What about $n = 30$?

Solution

Use the formula and plug in the values for n.
$\#csg^{star}(20) = 524307$
$\#csg^{star}(30) = 536870941$

Exercise 1 b)

If each plan consumes 40 bytes of memory. Then how much memory consumes the DP-table for 30 relations?

Solution

$$\#csg^{star}(30) * 40B = 536870941 * 40B = 21474837640B > 21GB.$$

Exercise 1 c)

How to approach large problem sizes?

Solution

- Use heuristic (deterministic or probabilistic)
- Use algorithms that find the optimal solution that allow for early termination (but find optimal solution when run to the end)
- Use algorithms that allow for pruning of the search space (memoization), thereby reducing the number of connected sub graphs considered.
- Apply DP-algorithms to subproblems. Apply DP-algorithm on solution to subproblems where each solution to the subproblems is considered a single relation

Exercise 2

Exercise 2 a)

Recall the introductionary DP exercise:
`Walking up the stairs.` How many steps can you take at a time? Let's say up to three! Then how many ways are there to walk up a staircase with n steps?

... This time, use memoization to answer the question!

Solution

See code.

Exercise 2 b)

Implement `MemoizationJoinOrdering`. You may use the helper classes provided in the solution code.

Solution

See code.

Exercise 3

Modify `MemoizationJoinOrdering` such that cross products are excluded.

Solution

```
memoization(V)
  ▷ Input: A connected query graph with relations V = ∪_i{R_i}
  ▷ Output: An optimal join tree for V
for i ← 1 to n
    do BestTree({R_i}) ← R_i
return momoizationSub(R)
```

```
memoizationSub(S)
  ▷ Input: A connected (sub-)graph with relations S
  ▷ Output: An optimal join tree for S
if BestTree(S) ≠ null
  then return BestTree(S)
for all S_1 ⊂ S and S_1 ≠ ∅
    do if !isConnected(S_1)
          then continue
        S_2 ← S − S_1
        if !isConnected(S_2)
          then continue
        CurrTree ← createTree(memoizationSub(S_1), memoizationSub(S_2))
        if BestTree(S) = null or cost(BestTree(S)) > cost(CurrTree)
          then BestTree(S) ← CurrTree
return BestTree(S)
```

Exercise 3 a)

What do you observe with regard to the connection tests? Compare this to `DPsub`.

Solution

No need for check for connectivity of $S_1 \cup S_2$, is an invariant of the algorithms input.

Exercise 3 b)

Name Pros and Cons compared to DP/ bottom-up approaches.

Solution

Cons: recursiv (each recursive call comes at a significant constant cost). Not known algorithm similar to DPcsgCmp.

Pros: Allows for (Cost) pruning. (Or, as we can see above, Connectivity pruning.)