

B561 Assignment 8

1 Object Relational Programming

1. For this problem you can not use arrays.

Consider the relational schema `Tree(parent int, child int)` representing the schema for storing a rooted tree.¹ A pair of nodes (m, n) is in `Tree` if m is the parent of n in `Tree`. Notice that a node m can be the parent of multiple children but a node n can have at most one parent node.

It should be clear that for each pair of different nodes m and n in `Tree`, there is a unique shortest path of nodes (n_1, \dots, n_k) in `Tree` from m to n provided we interpret the edges in `Tree` as undirected. A good way to think about this path from a node m to a node n is to first consider the *lowest common ancestor node* of m of n in `Tree`. Then the unique path from m to n is the path that is comprised of the path up the tree from m to this common ancestor and then, from this common ancestor, the path down the tree to the node n . (Note that in this path $n_1 = m$ and $n_k = n$.)

Define the *distance* from m to n to be $k - 1$ if (n_1, \dots, n_k) is the unique shortest path from m to n in `Tree`.

Write a PostgreSQL function `distance(m,n)` that computes the distance in `Tree` for any possible pair of different nodes m and n in `Tree`.

For example, if m is the parent of n in `Tree` then `distance(m,n) = 1` because the shortest path from m to n is (m, n) which has length 1. If m is the grandparent of n in `Tree` then `distance(m,n) = 2` since (m, p, n) is the path from m to n where p is the parent of m and p is a child of n . And if m and n have a common grandparent k then `distance(m,n) = distance(m,k) + distance(k,n) = 4`, etc.

2. For this problem you can use arrays.

Consider the relation schema `Graph(source int, target int)` representing the schema for storing a directed graph G of edges.

Now let G be a directed graph that is **acyclic**, a graph without cycles.²

A *topological sort* of an acyclic graph is a list of **all** of its nodes (n_1, n_2, \dots, n_k) such that for each edge (m, n) in G , node m occurs before node n in this list.

Write a PostgreSQL program `topologicalSort()` that returns a topological sort of G .

¹We assume that a tree is a connected graph with a finite number of nodes.

²A cycle is a path (v_0, \dots, v_k) where $v_0 = v_k$.

3. For this problem, you can not use arrays.

Consider the following relational schemas. (You can assume that the domain of each of the attributes in these relations is `int`.)

`partSubpart(pid,sid,quantity)`
`basicPart(pid,weight)`

A tuple (p, s, q) is in `partSubPart` if part s occurs q times as a **direct** subpart of part p . For example, think of a car c that has 4 wheels w and 1 radio r . Then $(c, w, 4)$ and $(c, r, 1)$ would be in `partSubPart`. Furthermore, then think of a wheel w that has 5 bolts b . Then $(w, b, 5)$ would be in `partSubpart`.

A tuple (p, w) is in `basicPart` if basic part p has weight w . A basic part is defined as a part that does not have subparts. In other words, the pid of a basic part does not occur in the pid column of `partSubPart`.

(In the above example, a bolt and a radio would be basic parts, but car and wheel would not be basic parts.)

We define the *aggregated weight* of a part inductively as follows:

- (a) If p is a basic part then its aggregated weight is its weight as given in the `basicPart` relation
- (b) If p is not a basic part, then its aggregated weight is the sum of the aggregated weights of its subparts, each multiplied by the quantity with which these subparts occur in the `partSubpart` relation.

Example tables: The following example is based on a desk lamp with pid 1. Suppose a desk lamp consists of 4 bulbs (with pid 2) and a frame (with pid 3), and a frame consists of a post (with pid 4) and 2 switches (with pid 5). Furthermore, we will assume that the weight of a bulb is 5, that of a post is 50, and that of a switch is 3.

Then the `partSubpart` and `basicPart` relation would be as follows:

partSubPart			basicPart	
pid	sid	quantity	pid	weight
1	2	4	2	5
1	3	1	4	50
3	4	1	5	3
3	5	2		

Then the aggregated weight of a lamp is $4 \times 5 + 1 \times (1 \times 50 + 2 \times 3) = 76$.

Write a PostgreSQL function `aggregatedWeight(p integer)` that returns the aggregated weight of a part p .

4. **For this problem you need to use arrays.**

Consider the relation schema `document(doc int, words text[])` representing a relation of pairs (d, W) where d is a unique id denoting a document and W denotes the set of words that occur in d .

Let \mathbf{W} denote the set of all words that occur in the documents and let t be a positive integer denoting a *threshold*.

Let $X \subseteq \mathbf{W}$. We say that X is t -frequent if

$$\text{count}(\{d \mid (d, W) \in \text{document and } X \subseteq W\}) \geq t$$

In other words, X is t -frequent if there are at least t documents that contain all the words in X .

Write a PostgreSQL program `frequentSets(t int)` that returns the set of all t -frequent set.

In a good solution for this problem, you should use the following rule: if X is not t -frequent then any set Y such that $X \subseteq Y$ is not t -frequent either. In the literature, this is called the *Apriori* rule of the frequent itemset mining problem. This rule can be used as a pruning rule. In other words, if you have determined that a set X is not t -frequent then you no longer have to consider any of X 's supersets.

To learn more about this problem you can visit the site https://en.wikipedia.org/wiki/Apriori_algorithm.

5. **For this problem you can use arrays.**

For this problem, first read about the k -means clustering problem in

<http://stanford.edu/~cpiech/cs221/handouts/kmeans.html>

Look at the k -means clustering algorithm described in this document. Your task is to implement this algorithm in PostgreSQL for a dataset that consists of a set of points in a 2-dimensional space.

Assume that the dataset is stored in a ternary relation with schema

`Points(p int, x float, y float)`

where p is an integer uniquely identifying a point (x, y) .

Write a PostgreSQL program `kMeans(k integer)` that returns a set of k points that denote the centroids for the points in `Points`. Note that k is an input parameter to the `kMeans` function.

You will need to reason about how to determine when the algorithm terminates. A possible termination condition is to set a number of iterations that denotes how many iterations are run to approximate the centroids. Another termination condition is to consider when the set of centroids no longer changes.

2 Physical Database Organization and Algorithms

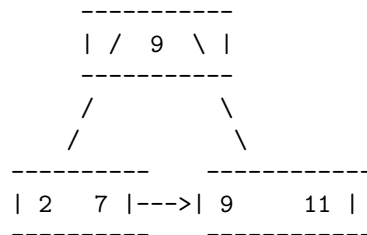
6. Consider the following parameters:

block size = 4096 bytes
 block-address size = 9 bytes
 block access time = 10 ms (micro seconds)
 record size = 200 bytes
 record key size = 12 bytes

Assume that there is a B⁺-tree, adhering to these parameters, that indexes 10⁸ million records on their primary key values.

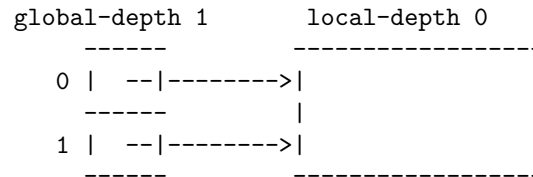
Show all the intermediate computations leading to your answers.

- (a) Specify (in ms) the minimum time to retrieve a record with key k in the B⁺-tree provided that there is a record with this key.
- (b) Specify (in ms) the maximum time to retrieve a record with key k in the B⁺-tree.
7. Consider the following B⁺-tree of order 2 that holds records with keys 2, 7, 9, and 11. (Observe that (a) an internal node of a B⁺-tree of order 2 can have either 1 or 2 keys values, and 2 or 3 sub-trees, and (b) a leaf node can have either 1 or 2 key values.)



- (a) Show the contents of your B⁺-tree after inserting records with keys 6, 10, 14, and 4, in that order.
- (b) Starting from your answer in question 7a, show the contents of your B⁺-tree after deleting records with keys 2, 14, 4, and 10, in that order.

8. Consider an extensible hashing data structure wherein (1) the initial global depth is set at 1 and (2) all directory pointers point to the same **empty** block which has local depth 0. So the hashing structure looks like this:



Assume that a block can hold at most two records.

- (a) Show the state of the hash data structure after each of the following insert sequences:³
- i. records with keys 2 and 6.
 - ii. records with keys 1 and 7.
 - iii. records with keys 4 and 8.
 - iv. records with keys 0 and 9.
- (b) Starting from the answer you obtained for Question 8a, show the state of the hash data structure after each of the following delete sequences:
- i. records with keys 1 and 2.
 - ii. records with keys 6 and 7.
 - iii. records with keys 0 and 9.
9. Let $R(A, B)$ and $S(B, C)$ be two relations and consider their natural join $R \bowtie S$.
- Assume that R has 1,500,000 records and that S has 5,000 records.
- Furthermore, assume that 30 records of R can fit in a block and that 10 records of S can fit in a block.
- Assume that you have a main-memory buffer with 101 blocks.
- (a) How many block IO's are necessary to perform $R \bowtie S$ using the block nested-loops join algorithm? Show your analysis.
 - (b) How many block IO's are necessary to perform $R \bowtie S$ using the sort-merge join algorithm? Show your analysis.

³You should interpret the key values as bit strings of length 4. So for example, key value 7 is represented as the bit string 0111 and key value 2 is represented as the bit string 0010.

- (c) Repeat question 9b under the following assumptions.
 Assume that there are p different B -values and that these are uniformly distributed in R and S .
 Observe that to solve this problem, depending on p , it may be necessary to perform a block nested-loop join per occurrence of a B -value.
- (d) How many block IO's are necessary to perform $R \bowtie S$ using the hash-join algorithm? Show your analysis.

3 Concurrency Control

10. State which of the following schedules S_1 , S_2 , and S_3 over transactions T_1 , T_2 , and T_3 are conflict-serializable, and for each of the schedules that is serializable, given a serial schedule with which that schedule is conflict-equivalent.
- (a) $S_1 = R_1(x)R_2(y)R_1(z)R_2(x)R_1(y)$.
 (b) $S_2 = R_1(x)W_2(y)R_1(z)R_3(z)W_2(x)R_1(y)$.
 (c) $S_3 = R_1(z)W_2(x)R_2(z)R_2(y)W_1(x)W_3(z)W_1(y)R_3(x)$.
11. Give 3 transactions T_1 , T_2 , T_3 and a schedule S on these transactions whose precedence graph (i.e. serialization graph) consists of the edges (T_1, T_2) , (T_2, T_1) , (T_1, T_3) , (T_3, T_2) .
12. Give 3 transactions T_1 , T_2 , and T_3 that each involve read and write operations and a schedule S that is conflict-equivalent with **all** serial schedules over T_1 , T_2 , and T_3 .
13. Consider the following transactions:

```
T1:  read(A);
      read(B);
      if A = 0 then B := B+1;
      write(B).
```

```
T2:  read(B);
      read(A);
      if B = 0 then A := A+1;
      write(A).
```

Let the consistency requirement be $A = 0 \vee B = 0$, and let $A = B = 0$ be the initial values.

- (a) Show that each serial schedule involving transaction T_1 and T_2 preserves the consistency requirement of the database.

- (b) Construct a schedule on T_1 and T_2 that produces a non-serializable schedule.
- (c) Is there a non-serial schedule on T_1 and T_2 that produces a serializable schedule. If so, give an example.