Question 1: Wavelets, Sketches (1 P.)

a) Calculate the wavelet transform of the following cumulative distribution over the numbers 0 to 7. Also calculate the Mean-Squared-Error (MSE) if you omit all coefficients ≤ 3.

$$[2, 6, 8, 12, 20, 25, 26, 29]$$

Solution

Resolution	Averages	Detail Coefficients
8	$[2, 6, 8, 12, 20, 25, 26, 29]$	
4	$[4, 10, 22.5, 27.5]$	$[4, 4, 5, 3]$
2	$[7, 25]$	$[6, 5]$
1	$[16]$	$[18]$

$$\hat{S} = [16, 18, 6, 5, 4, 4, 5, 3]$$

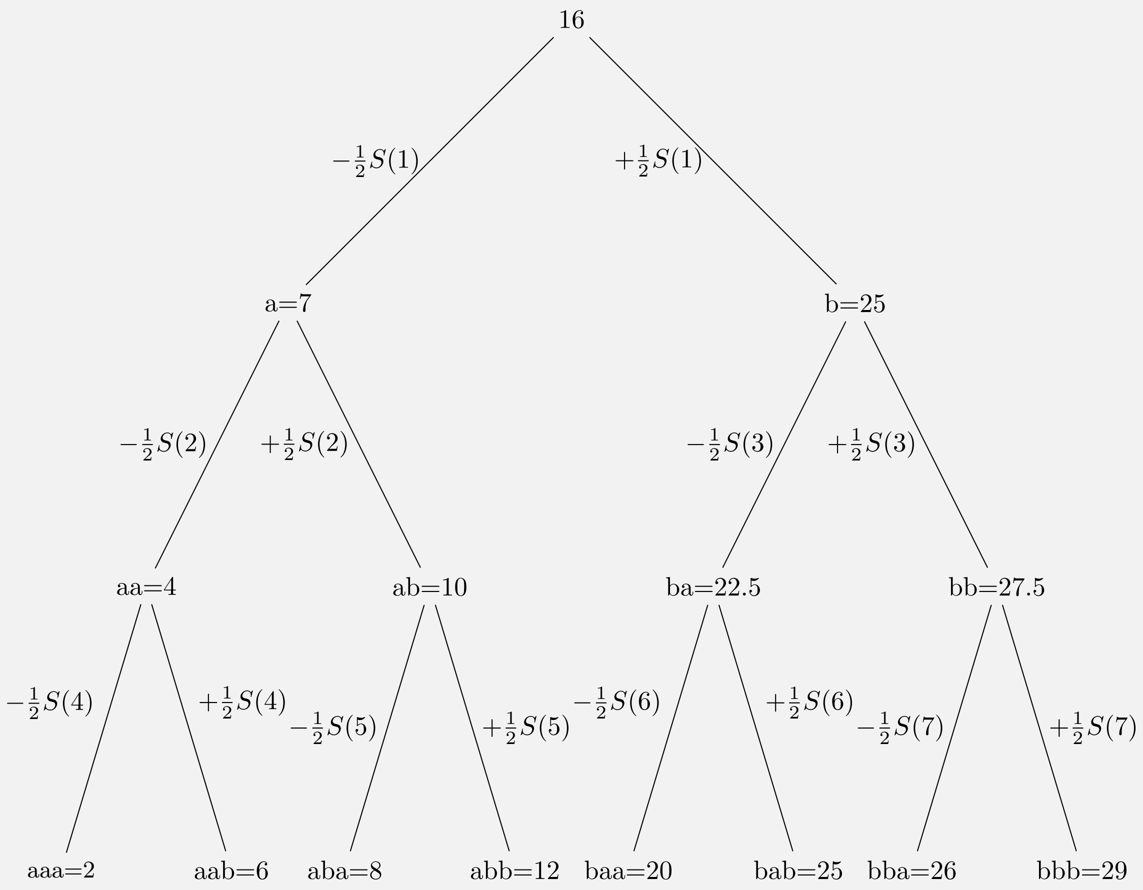
Reconstruction:

$$\begin{aligned}
transform &= [16, 18, 6, 5, 4, 4, 5, 3] \\
a &= \hat{S}(0) - \frac{1}{2} \cdot \hat{S}(1) && = 16 - \frac{1}{2} \cdot 18 && = 7 \\
b &= \hat{S}(0) + \frac{1}{2} \cdot \hat{S}(1) && = 16 + \frac{1}{2} \cdot 18 && = 25 \\
aa &= a - \frac{1}{2} \cdot \hat{S}(2) && = a - \frac{1}{2} \cdot 6 && = 4 \\
ab &= a + \frac{1}{2} \cdot \hat{S}(2) && = a + \frac{1}{2} \cdot 6 && = 10 \\
ba &= b - \frac{1}{2} \cdot \hat{S}(3) && = b - \frac{1}{2} \cdot 5 && = 22.5 \\
bb &= b + \frac{1}{2} \cdot \hat{S}(3) && = b + \frac{1}{2} \cdot 5 && = 27.5 \\
aaa &= aa - \frac{1}{2} \cdot \hat{S}(4) && = aa - \frac{1}{2} \cdot 4 && = 2 \\
aab &= aa + \frac{1}{2} \cdot \hat{S}(4) && = aa + \frac{1}{2} \cdot 4 && = 6 \\
aba &= ab - \frac{1}{2} \cdot \hat{S}(5) && = ab - \frac{1}{2} \cdot 4 && = 8 \\
abb &= ab + \frac{1}{2} \cdot \hat{S}(5) && = ab + \frac{1}{2} \cdot 4 && = 12 \\
baa &= ba - \frac{1}{2} \cdot \hat{S}(6) && = ba - \frac{1}{2} \cdot 5 && = 20 \\
bab &= ba + \frac{1}{2} \cdot \hat{S}(6) && = ba + \frac{1}{2} \cdot 5 && = 25 \\
bba &= bb - \frac{1}{2} \cdot \hat{S}(7) && = bb - \frac{1}{2} \cdot 3 && = 26 \\
bbb &= bb + \frac{1}{2} \cdot \hat{S}(7) && = bb + \frac{1}{2} \cdot 3 && = 29 \\
input &= [aaa, aab, aba, abb, baa, bab, bba, bbb]
\end{aligned}$$

Approximation just like above, but with coefficients ≤ 3 omitted:

$$[2, 6, 8, 12, 20, 25, 27.5, 27.5]$$

$$MSE = \frac{2 \cdot (1.5)^2}{8}$$



b) Given the following Table with elements $x \in X$ and the hash values for the hash function $h : X \to [0, 1]$.

x	$h(x)$	x	$h(x)$
Toyota	0.05	Mercedes	0.15
Ford	0.489	Dodge	0.565
Bughatti	0.678	Porsche	0.579
BMW	0.281	Nissan	0.395
Audi	0.81	Hyundai	0.91

Using the KMV Sketch algorithm for $k \in 1, 2, 3, 4, 5$, calculate the estimate as well as the absolute error. Will the error get smaller with a larger k? How does this generally relate to the KMV Sketch?

Solution

First, we sort the hash values in ascending order:

$$0.05 \quad 0.15 \quad 0.281 \quad 0.395 \quad 0.489 \quad 0.565 \quad 0.579 \quad 0.678 \quad 0.81 \quad 0.91$$

For the biased estimatior $n^{UB} = (k - 1)/U_{(k)}$ we have:

- $k = 1 :$ $0/0.05 = 0$, Error: 10
- $k = 2 :$ $1/0.15 = 6.67$, Error: 3.33
- $k = 3 :$ $2/0.281 = 7.12$, Error: 2.8
- $k = 4 :$ $3/0.395 = 7.5$, Error: 2.5
- $k = 5 :$ $4/0.489 = 8.1$, Error: 1.9

For the unbiased estimator $\hat{n}^{UB} = (k)/U_{(k)}$ we have:

- $k = 1 :$ $1/0.05 = 20$, Error: 10
- $k = 2 :$ $2/0.15 = 13.33$, Error: 3.33
- $k = 3 :$ $3/0.281 = 10.67$, Error 0.67
- $k = 4 :$ $4/0.395 = 10.12$, Error: 0.12
- $k = 5 :$ $5/0.489 = 10.22$, Error: 0.22

If the values are indeed uniformly distributed and you have a good hash function, the error gets smaller. However this is not the case for every hash function nor for every distribution of the inputs. For example, if the inputs are less uniformly mapped into the $[0, 1]$ interval, but all take very large hashes, the estimator also estimates too large values.

Question 2: Statistics in PostgreSQL (1 P.)

To improve the query optimization, PostgreSQL keeps track of various statistics about every column in every table. Consider the following statistics about the *lineitem* table from the TPC-H database:

```
SELECT * FROM pg_stats WHERE tablename = 'lineitem'
```

attname	null_frac	n_distinct	most_common_vals	most_common_freqs
l_linenumber	0	7	{1,2,3,4,5,6,7}	{0.25,0.21,0.17,0.14,0.10,0.07,0.03}
l_orderkey	0	419664	{1703939,3644579,507137, 703172,712326,770882, 971014}	{0.000133333,0.000133333,0.0001, 0.0001,0.0001,0.0001, 0.0001}
l_extendedprice	0	-0.11964	{13747.3,25165.2, 26677.8,32274.0, 33265.3,35938.8,50370.3 }	{0.0001,0.0001,0.0001,0.0001, 0.0001,0.0001,0.0001 }

a) What does each column of *pg_stats* represent?

Solution

See `https://www.postgresql.org/docs/10/static/view-pg-stats.html`:

null_frac The fraction of `NULL` tuples.

n_distinct ≥ 0: Estimated amount of distinct values in the column.
< 0: Estimated amount of distinct values is $-x \cdot |R|$, with $|R|$ the number of tuples in the column.

most_common_vals/freqs The most frequent values with their corresponding frequencies.

b) Which of the following queries benefit from these statistics? The total size of the table is 6 001 215.

☐ `SELECT * FROM lineitem`

Solution

The statistics do not help, as all 6 001 215 tuples will be returned.

☐ `SELECT * FROM lineitem WHERE l_orderkey IS NOT NULL`

Solution

null_frac can be used, as the value of 0 tells the system that all 6 001 215 tuples will be returned.

☐ `SELECT * FROM lineitem WHERE l_orderkey = 406631`

Solution

406 631 is not contained in *most_common_vals*. n_distinctcould still be used to make a rough estimation: $6\,001\,215/419\,664 \approx 14.3$
Additionally, as 406 631 is not contained in *most_common_vals*, we get a higher bound for its occurrence with $6\,001\,215 * 0.0001 \approx 600$.

☐ `SELECT * FROM lineitem WHERE l_extendedprice = 32274.0`

Solution

The value 32274.0 is contained in *most_common_vals*: $6\,001\,215 \cdot 0.0001 \approx 600$

☐ `SELECT * FROM lineitem WHERE l_extendedprice <= 9500.0`

Solution

The statistics can't be used in this example. We know that `l_extendedprice` has $-0.11964 \cdot 6\,001\,215 \approx 717\,985$ distinct values, but we do not know how they are distributed.

c) Use the statistics to estimate the result size of each query in part (b).

In the table below an additional column of the *pg_stats* table is shown. It shows "A list of values that divide the column's values into groups of approximately equal population. The values in *most_common_vals*, if present, are omitted from this histogram calculation." – `https://www.postgresql.org/docs/10/static/view-pg-stats.html`

attname	**histogram_bounds**
l_extendedprice	{906.00,1528.60,2062.06,3035.16,3639.44,4444.04,5265.52,5920.60, 6751.85,7518.63,8252.80,9032.32,9777.68,10540.68,11267.46, 11921.76,12652.00,13413.40,14078.28 (...) }

d) Decide for which queries in (b) this additional column is useful and estimate the result set size.

Solution

The histogram can be used to better estimate the result of `l_extendedprice <= 9500.0` :

If the histogram is split into m cells, then every cell contains approx. $6\,001\,215/m$ values (Minus the most common and NULL values). As 12 cells fulfill the predicate, at least $12 \cdot 6\,001\,215/m$ result tuples are to be expected. The 13. cell can be estimated to contain a factor of $(9500-9032)/(9777-9032) \approx 0.63$ of tuples fulfilling the predicate. In total the query will result in $\approx 12.63 \cdot 6\,001\,215/m$ tuples.

Question 3: Join ordering (1 P.)

Given the following query on the TPC-H database:

```sql
SELECT *
FROM orders o, customer c,  nation n,
WHERE o.o_custkey = c.c_custkey AND c.c_nationkey = n.n_nationkey
```

a) Draw the query graph.

Solution

$$\begin{tikzcd}
\mathrm{o} \arrow[r, dash, "\mathrm{o.o\_custkey = c.c\_custkey}"] & \mathrm{c} \arrow[r, dash, "\mathrm{c.c\_nationkey = n.n\_nationkey}"] & \mathrm{n}
\end{tikzcd}$$

b) Calculate the join selectivities.

Solution

$$f_{o,c} = \frac{|o \bowtie_{o.o\_custkey=c.c\_custkey} l|}{|o \times c|}$$

$$f_{c,n} = \frac{|c \bowtie_{c...} n|}{c \times n}$$

The following queries calculate the exact values:

```sql
WITH joinsize AS (
  SELECT COUNT(*) cj
  FROM orders o
  JOIN customer c
  ON(o.o_custkey = c.c_custkey)
), osize AS (
  SELECT COUNT(*) oc
  FROM orders o
), csize AS (
  SELECT COUNT(*) cc
  FROM customer c
)
SELECT CAST(cj AS real)/(oc * cc)
FROM joinsize, osize, csize
-- = 6.66666e-06
```

```sql
WITH joinsize AS (
    SELECT COUNT(*) cj
    FROM customer c
    JOIN nation n
    ON(n.n_nationkey = c.c_nationkey)
  ), nsize AS (
    SELECT COUNT(*) nc
    FROM nation o
  ), csize AS (
    SELECT COUNT(*) cc
    FROM customer c
  )
  SELECT CAST(cj AS real)/(nc * cc)
  FROM joinsize, nsize, csize
  -- = 0.04
```

$f_{o,c} = 0.00000666667$ and $f_{c,n} = 0.04$.

As no join predicate exists between `o` and `n`: $f_{o,n} = 1$.

c) Calculate the cost C_{out} for all possible join trees without cross products.

Solution

The possible join trees are $A := (n \bowtie c) \bowtie o$ and $B := n \bowtie (c \bowtie o)$. (And the swapped versions, but for these C_{out} is the same).

- $C_{out}(A)$:

$$C_{out}(n \bowtie c) = |n \bowtie c| + 0 + 0 = f_{n,c} \cdot 25 \cdot 150\,000 = 150\,000$$

$$|A| = f_{o,c} \cdot 1\,500\,000 \cdot 150\,000 = 1\,500\,000$$

$$C_{out}(A) = |A| + C_{out}(n \bowtie c) + C_{out}(o) = 1\,500\,000 + 150\,000 + 0 = 1\,650\,000$$

- $C_{out}(B)$:

$$C_{out}(c \bowtie o) = |c \bowtie o| + 0 + 0 = f_{c,o} \cdot 150\,000 \cdot 1\,500\,000 = 1\,500\,000$$

$$|B| = f_{n,c} \cdot 25 \cdot 1\,500\,000 = 1\,500\,000$$

$$C_{out}(B) = |B| + C_{out}(n) + C_{out}(c \bowtie o) = 1\,500\,000 + 0 + 1\,500\,000 = 3\,000\,000$$

Question 4: Join Ordering (1 P.)

a) **Left-deep trees vs. Bushy trees**
Given the following relations: R_1, R_2, R_3 and R_4, with $|R_1| = 100$, $|R_2| = 50$, $|R_3| = 200$, $|R_4| = 10$, and the following join selectivities: $j_{1,2} = 0.04$, $j_{2,3} = 0.10$, $j_{2,4} = 0.05$.

i) Draw the query graph.

Solution

$$\begin{tikzcd}
R_1 \arrow[r, dash] & R_2 \arrow[r, dash] \arrow[d, dash] & R_4 \\
& R_3 &
\end{tikzcd}$$

ii) List all left-deep trees without cross products.

Solution

The query graph in (i) shows, the query is star-shaped. As no cross products are allowed, each join has to start with the center relationship (R_2). After joining R_2, all other relations can be joined in arbitrary order.
Thereby, the allowed left-deep join orderings are:

$$((R_1 \bowtie R_2) \bowtie R_3) \bowtie R_4$$
$$((R_2 \bowtie R_1) \bowtie R_3) \bowtie R_4$$
$$((R_2 \bowtie R_3) \bowtie R_1) \bowtie R_4$$
$$((R_3 \bowtie R_2) \bowtie R_1) \bowtie R_4$$
$$((R_1 \bowtie R_2) \bowtie R_4) \bowtie R_3$$
$$((R_2 \bowtie R_1) \bowtie R_4) \bowtie R_3$$
$$((R_2 \bowtie R_3) \bowtie R_4) \bowtie R_1$$
$$((R_3 \bowtie R_2) \bowtie R_4) \bowtie R_1$$
$$((R_2 \bowtie R_4) \bowtie R_1) \bowtie R_3$$
$$((R_4 \bowtie R_2) \bowtie R_1) \bowtie R_3$$
$$((R_2 \bowtie R_4) \bowtie R_3) \bowtie R_1$$
$$((R_4 \bowtie R_2) \bowtie R_3) \bowtie R_1$$

iii) Calculate the C_{out} cost for the join orderings created in (ii) where R_3 is joined last (on the top of the tree).

Solution

As C_{out} is defined recursively, it is best to calculate it bottom-up:

Tree	C_{out}	**Cardinality**
$R_1 \bowtie R_2$	200	200
$R_2 \bowtie R_4$	25	25
$(R_1 \bowtie R_2) \bowtie R_4$	300	100
$(R_2 \bowtie R_4) \bowtie R_1$	125	100

The final orderings then have the following values:

Tree	C_{out}
$((R_1 \bowtie R_2) \bowtie R_4) \bowtie R_3$	2300
$((R_2 \bowtie R_1) \bowtie R_4) \bowtie R_3$	2300
$((R_2 \bowtie R_4) \bowtie R_1) \bowtie R_3$	2125
$((R_4 \bowtie R_2) \bowtie R_1) \bowtie R_3$	2125

We don't have to calculate all variations, as C_{out} is symmetric.

b) Is there a bushy tree (that is not a linear tree), without cross products, that has lower cost?

Solution

There are no bushy trees without cross product. The star-shaped query does not allow us to join relations without R_2.

Question 5: Join Ordering Algorithms (1 P.)

Write a program which compares the different join ordering algorithms, presented in the lecture. You may use any language you like. You have two write one method for each greedy algorithm plus two methods that find the best and worst possible join ordering. For the last two functions you have to find all possible orderings. Use the C_{out} cost as weight function and compare the algorithm results with these costs.

You may use the Java template provided in OLAT. The template provides all required boilerplate code and instructions on how to use it.

Use your program to generate plans for the following relations: $|R_1| = 200, |R_2| = 300, |R_3| = 20, |R_4| = 140$ with the selectivities: $j_{1,3} = 1/100$, $j_{2,3} = 1/50$, $j_{3,4} = 1/5$ Cross products are not allowed.

Submit the created plans with their C_{out} costs and the code that created them.

HINT: The best and worst plans range somewhere between $(8\,000, 11\,000)$.

Solution

The results should be:

Strategy	Plan	C_{out}
Greedy-1	$((R_3 \bowtie R_4) \bowtie R_1) \bowtie R_2$	8 400
Greedy-2	$((R_3 \bowtie R_1) \bowtie R_2) \bowtie R_4$	7 000
Greedy-3	$((R_3 \bowtie R_1) \bowtie R_2) \bowtie R_4$	7 000
Best	$((R_3 \bowtie R_1) \bowtie R_2) \bowtie R_4$	7 000
Worst	$R_1 \bowtie (R_2 \bowtie (R_3 \bowtie R_4))$	10 640

The greedy algorithms can be implemented just as the lecture showed. Greedy-1 uses the size of the relations as cost function, while Greedy-2/3 use the c_out of the joins.

To find the optimal(/worst) join order you have to create all orderings. You can do this by recursively creating all smaller join trees and extending them. It is no problem if your algorithm creates some trees multiple times, as we only need the optimal tree, as long as all trees are generated. After creating all join orderings you can compute their cost and choose the best(/worst) one.