Name

CWID

Quiz
2

Due Nov 18th, 2020

CS525 - Advanced Database Organization

Solutions

Please leave this empty! 1 2 3 4 5 6 7

Sum

Instructions

- Multiple choice questions are graded in the following way: You get points for correct answers and points subtracted for wrong answers. The minimum points for each questions is **0**. For example, assume there is a multiple choice question with 6 answers - each may be correct or incorrect - and each answer gives 1 point. If you answer 3 questions correct and 3 incorrect you get 0 points. If you answer 4 questions correct and 2 incorrect you get 2 points. ...
- For your convenience the number of points for each part and questions are shown in parenthesis.
- There are 3 parts in this quiz
  1. Disk Organization and Buffering
  2. Index Structures
  3. Result Size Estimations
  4. I/O Cost Estimation
  5. Schedules
  6. ARIES (Optional)
  7. Physical Optimization (Optional)

Part 1 Disk Organization and Buffering (Total: 15 Points)

Question 1.1 Page Replacement Clock (15 Points)

Consider a buffer pool with 3 pages using the **Clock** page replacement strategy. Initially the buffer pool is in the state shown below. We use the following notation ${}^{flag}[page]_{fix}^{dirty}$ to denote the state of each buffer frame. *page* is the number of the page in the frame, fix is its fix count, *dirty* is indicating with an Asterix that the page is dirty, and $flag$ is the reference bit used by the Clock algorithm. E.g., ${}^{1}[5]_{2}^{*}$ denotes that the frame stores page 5 with a fix count 2, that the page is dirty, and that the reference bit is set to 1. Recall that Clock uses a pointer S that points to the current page frame (the one to be checked for replacement next). The page frame S is pointing to is indicated by by $\downarrow$. In your solution draw an arrow to the page frame that S is pointing to.

Current Buffer State

$${}^{1}[3]_{0}^{*} \quad {}^{1}[\overset{\downarrow}{10}]_{0} \quad {}^{0}[6]_{0} \quad {}^{1}[4]_{1}$$

Execute the following requests and write down state of the buffer pool after each request.

- p stands for pin
- u for unpin
- d for marking a page as dirty

`p(10),u(10),p(6),u(4),p(5),p(1),u(6)`

Solution

p(10)

$${}^{1}[3]_{0}^{*} \quad {}^{1}[\overset{\downarrow}{10}]_{1} \quad {}^{0}[6]_{0} \quad {}^{1}[4]_{1}$$

u(10)

$${}^{1}[3]_{0}^{*} \quad {}^{1}[\overset{\downarrow}{10}]_{0} \quad {}^{0}[6]_{0} \quad {}^{1}[4]_{1}$$

p(6)

$${}^{1}[3]_{0}^{*} \quad {}^{1}[\overset{\downarrow}{10}]_{0} \quad {}^{1}[6]_{1} \quad {}^{1}[4]_{1}$$

u(4)

$${}^{1}[3]_{0}^{*} \quad {}^{1}[\overset{\downarrow}{10}]_{0} \quad {}^{1}[6]_{1} \quad {}^{1}[4]_{0}$$

p(5)

$${}^{0}[3]_{0}^{*} \quad {}^{1}[5]_{1} \quad {}^{0}[\overset{\downarrow}{6}]_{1} \quad {}^{0}[4]_{0}$$

p(1)

$${}^{0}[\overset{\downarrow}{3}]_{0}^{*} \quad {}^{1}[5]_{1} \quad {}^{0}[6]_{1} \quad {}^{1}[1]_{1}$$

u(6)

$${}^{0}[\overset{\downarrow}{3}]_{0}^{*} \quad {}^{1}[5]_{1} \quad {}^{0}[6]_{0} \quad {}^{1}[1]_{1}$$

Part 2 Index Structures (Total: 25 Points)

Assume that you have the following table:

Student

name	gpa	credits
Will Wonton	4.0	1
Joe Joeton	3.5	15
Jill Johnson	2.4	3
John Johnson	3.1	4
Bo Zhao	3.1	5
Yu Wei	3.0	22
Heinz Bert	2.2	21

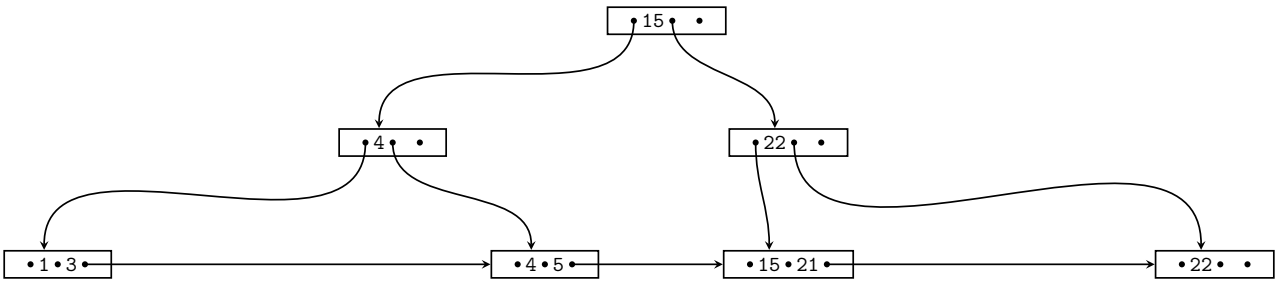
Question 2.1 Construction (9 Points)

Create a B+-tree for table *Student* over attribute *credits* with $n = 2$ (up to two keys per node). You should start with an empty B+-tree and insert the keys in the order shown in the table above. Write down the resulting B+-tree after each step.

When splitting or merging nodes follow these conventions:

- **Leaf Split**: In case a leaf node needs to be split during insertion and n is even, the left node should get the extra key. E.g, if $n = 2$ and we insert a key 4 into a node [1,5], then the resulting nodes should be [1,4] and [5]. For odd values of n we can always evenly split the keys between the two nodes. In both cases the value inserted into the parent is the smallest value of the right node.
- **Non-Leaf Split**: In case a non-leaf node needs to be split and n is odd, we cannot split the node evenly (one of the new nodes will have one more key). In this case the "middle" value inserted into the parent should be taken from the right node. E.g., if $n = 3$ and we have to split a non-leaf node [1,3,4,5], the resulting nodes would be [1,3] and [5]. The value inserted into the parent would be 4.
- **Node Underflow**: In case of a node underflow you should first try to redistribute values from a sibling and only if this fails merge the node with one of its siblings. Both approaches should prefer the left sibling. E.g., if we can borrow values from both the left and right sibling, you should borrow from the left one.

Solution

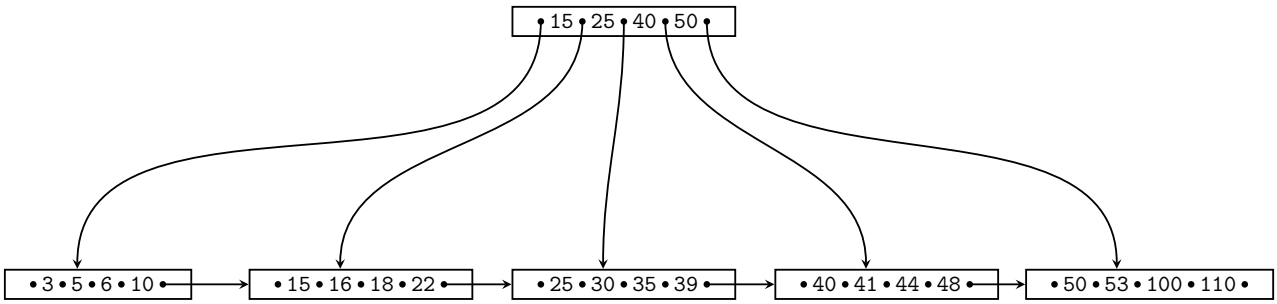


Question 2.2 Operations (9 Points)

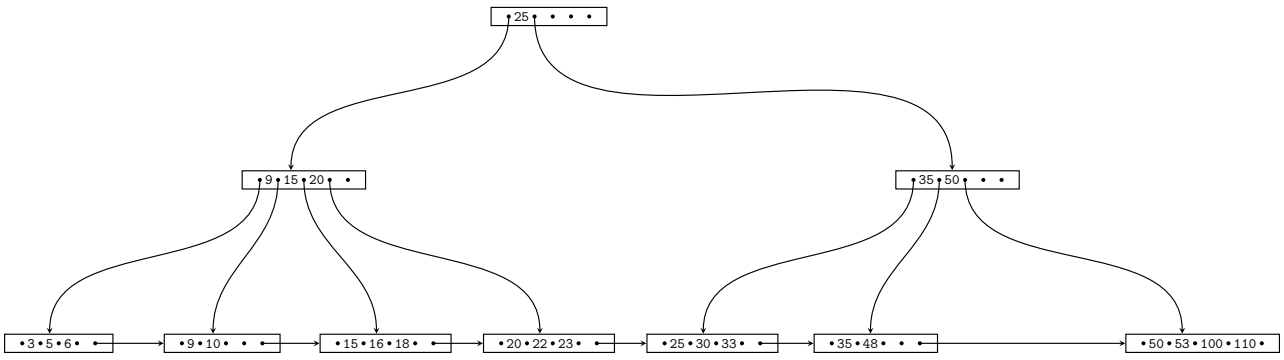
Given is the B+-tree shown below ($n = 4$). Execute the following operations and write down the resulting B+-tree after each operation:

delete(40), delete(41), delete(44), delete(50), delete(39), insert(9), insert(20), insert(23), insert(33), insert(50)

Use the conventions for splitting and merging introduced in the previous question.



Solution



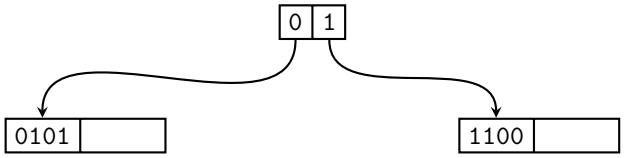
Question 2.3 Extensible Hashing (7 Points)

Consider the extensible Hash index shown below that is the result of inserting values 6 and 4. Each page holds two keys. Execute the following operations

`insert(7),insert(8),insert(2),insert(0),insert(1)`

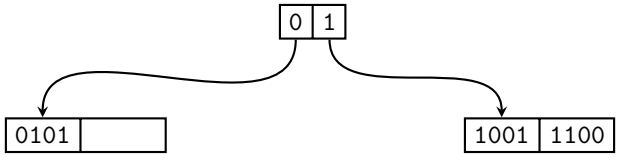
and write down the resulting index after each operation. Assume the hash function is defined as:

x	h(x)
0	1000
1	1101
2	0111
3	0000
4	1100
5	0100
6	0101
7	1001
8	1110

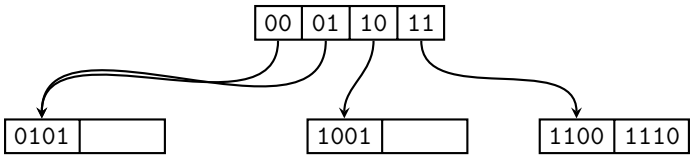


Solution

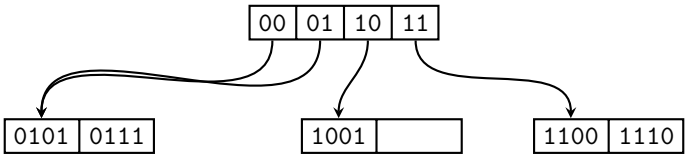
insert(7)



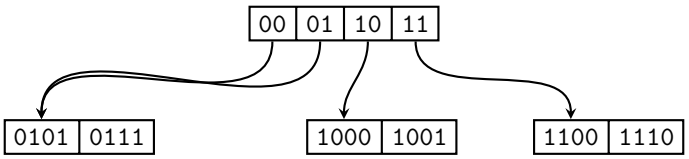
insert(8)



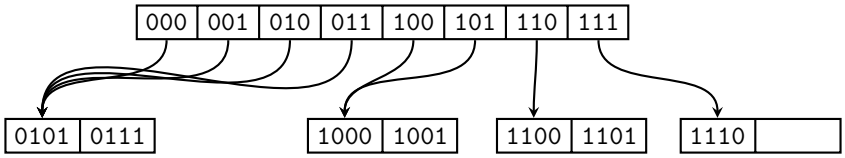
insert(2)



insert(0)



insert(1)



Part 3 Result Size Estimations (Total: 20 Points)

Consider a table book with attributes ISBN, title, author, edition (primary key is ISBN), a table library with loc, budget, public (primary key is loc), and a table catalog with attributes library and book. catalog.library is a foreign key to relation library. Attribute book of relation catalog is a foreign key to relation book. Given are the following statistics:

$$T(book) = 100,000 \qquad T(library) = 100 \qquad T(catalog) = 200,000$$
$$V(book, ISBN) = 100,000 \qquad V(library, loc) = 100 \qquad V(catalog, library) = 100$$
$$V(book, title) = 50,000 \qquad V(library, budget) = 40 \qquad V(catalog, book) = 90,000$$
$$V(book, author) = 30,000 \qquad V(library, public) = 2$$
$$V(book, edition) = 15$$

Question 3.1 Estimate Result Size (4 Points)

Estimate the number of result tuples for the query $q = \sigma_{author=Goethe \wedge edition=1}(book)$ using the first assumption presented in class (values used in queries are uniformly distributed within the active domain).

Solution

$$T(q) = \frac{T(book)}{V(book, author) \cdot V(book, edition)} = \frac{100,000}{30,000 \cdot 15} = \frac{2}{9} \approx 0.22$$

Question 3.2 Estimate Result Size (5 Points)

Estimate the number of result tuples for the query $q = \sigma_{loc=Chicago \vee public=true}(library)$ using the first assumption presented in class.

Solution

$$T(q) = (1 - [(1 - \frac{1}{V(library, loc)}) \cdot (1 - \frac{1}{V(library, public)})]) \cdot T(library)$$
$$= 1 - (1 - \frac{1}{100}) \cdot (1 - \frac{1}{2}) \cdot 100 = \frac{101}{2} = 50.5$$

Question 3.3 Estimate Result Size (5 Points)

Estimate the number of result tuples for the query $q = \sigma_{(ISBN=113-2132-5443 \vee (title=Databases \wedge author=Gert)}(book)$ using the first assumption presented in class.

Solution

$$T(q) = (1 - [(1 - \frac{1}{V(book, ISBN)}) \cdot (1 - (\frac{1}{V(book, title)} \cdot \frac{1}{V(book, author)}))]) \cdot T(book)$$
$$= (1 - [(1 - \frac{1}{100,000}) \cdot (1 - (\frac{1}{50,000} \cdot \frac{1}{30,000}))]) \cdot 100,000 = 1 : \frac{500033333}{500000000} \approx 1.000066666$$

Question 3.4 Estimate Result Size (6 Points)

Estimate the number of result tuples for the query

$q = book \bowtie_{title=book} catalog \bowtie_{library=loc} \sigma_{budget>300,000}(library)$

using the first assumption presented in class. Assume that the minimum value for `library.budget` is 0 and the maximum value is 10,000,000.

Solution

Let $q_1 = \sigma_{budget>300,000}(library)$.
To estimate the selection result size q_1:

$$T(q_1) = T(library) \cdot \frac{10,000,000 - 300,000 + 1}{10,000,000 - 0 + 1} = \frac{323326900}{3333267} \approx 97$$

Now for the full query we get

$$T(q) = \frac{T(book) \cdot T(catalog) \cdot T(q_1)}{max(V(book, title), V(catalog, book)) \cdot max(V(catalog, library), V(q_1, loc))}$$

$$= \frac{100,000 \cdot 200,000 \cdot 97}{max(50,000, 90,000) \cdot max(100, 97)} = \frac{1,940,000,000,000}{9,000,000} = \frac{1940000}{9} \approx 215555.56$$

Part 4 I/O Cost Estimation (Total: 20 Points)

Question 4.1 External Sorting (4 Points)

You have $M = 17$ memory pages available and should sort a relation R with $B(R) = 4,000,000,000$ blocks. Compute the number of I/Os necessary to sort R using the external merge sort algorithm introduced in class.

Solution

$$\begin{aligned} IO &= 2 \cdot B(R) \cdot (1 + \lceil log_{M-1}(\frac{B(R)}{M}) \rceil \\ &= 2 \cdot 4,000,000,000 \cdot (1 + 7) \\ &= 64,000,000,000 \end{aligned}$$

Question 4.2 External Sorting (4 Points)

You have $M = 4$ memory pages available and should sort a relation R with $B(R) = 120$ blocks. Compute the number of I/Os necessary to sort R using the external merge sort algorithm introduced in class.

Solution

$$\begin{aligned} IO &= 2 \cdot B(R) \cdot (1 + \lceil log_{M-1}(\frac{B(R)}{M}) \rceil \\ &= 2 \cdot 120 \cdot (1 + 4) \\ &= 1200 \end{aligned}$$

Question 4.3 I/O Cost Estimation (6 = 2+2+2 Points)

Consider two relations R and S with $B(R) = 100,000$ and $B(S) = 300,000$. You have $M = 201$ memory pages available. Estimate the minimum number of I/O operations needed to join these two relations using **block-nested-loop join**, **merge-join** (the inputs are not sorted), and **hash-join**. You can assume that the hash function evenly distributes keys across buckets. Justify you result by showing the I/O cost estimation for each join method.

Solution

- **BNL**: R is smaller, thus, keep chunks of R in memory
  $\lceil \frac{B(R)}{M-1} \rceil \cdot [B(S) + min(B(R), (M-1))] = 498 \cdot [300,000 + 200] = 150,100,000$ I/Os
- **MJ**: We can generate sorted runs of size 201. We need 2 merge pass for the sort for R and 2 merge passes for S. The last merge of R requires 3 pages and the last merge of S requires 8 pages, i.e., the number of sorted runs from R and S small enough to keep one page from each run of both R and S in memory. $5 \cdot B(R) + 5 \cdot B(S) = 5 \cdot 100,000 + 5 \cdot 300,000 = 2,000,000$ I/Os.
- **HJ**: We need 2 partitioning passes, because we can create 200 buckets and the bucket sizes of R will be 3 after two passes. The cost is $(4+1) \cdot (B(R) + B(S)) = 5 \cdot (100,000 + 300,000) = 2,000,000$ I/Os.

Question 4.4 I/O Cost Estimation (6 = 2+2+2 Points)

Consider two relations R and S with $B(R) = 60,000$ and $B(S) = 3,000$. You have $M = 81$ memory pages available. Compute the minimum number of I/O operations needed to join these two relations using **block-nested-loop join**, **merge-join** (the inputs are not sorted), and **hash-join**. You can assume that the hash function evenly distributes keys across buckets. Justify you result by showing the I/O cost estimation for each join method.

Solution

- **BNL**: S is smaller, thus, keep chunks of S in memory
  $\lceil \frac{B(S)}{M-1} \rceil \cdot [B(R) + min(B(S), (M-1))] = 38 \cdot [60,000 + 80] = 2,283,040$ I/Os
- **MJ**: We can Generate sorted runs of size 81 that means the number of sorted runs from R and S is low enough after 2 merge passes for R and 1 merge passes for S to keep one page from each run of both R and S in memory. $5 \cdot B(R) + 3 \cdot B(S) = 309,000$ I/Os.
- **HJ**: Relation S fits into memory after 1 partitioning phase. The cost is $3 \cdot (B(R) + B(S)) = 189,000$ I/Os.

Part 5 Schedules (Total: 20 Points)

Question 5.1 Schedule Classes (20 Points)

Indicate which of the following schedules belong to which class. Recall transaction operations are modelled as follows:

$w_1(A)$	transaction 1 wrote item A
$r_1(A)$	transaction 1 read item A
c_1	transaction 1 commits
a_1	transaction 1 aborts

$$S_1 = r_2(A), r_2(E), w_1(C), r_1(E), r_2(C), c_2, w_3(A), w_3(C), c_3, r_1(D), c_1$$
$$S_2 = w_2(C), r_4(D), r_3(B), w_2(A), c_2, r_1(B), w_4(C), w_4(D), c_4, r_3(E), w_3(A), w_3(E), c_3, r_1(E), c_1$$
$$S_3 = r_2(E), r_1(A), r_2(C), r_1(B), r_1(C), r_2(A), w_2(C), r_2(B), c_2, r_1(C), w_1(E), w_1(A), c_1$$
$$S_4 = r_3(D), w_3(D), r_3(C), w_4(C), r_2(D), w_2(D), c_2, r_4(D), c_4, r_3(C), c_3$$

- ❑ S_1 is recoverable
- ❑ S_1 is cascade-less
- ❑ S_1 is strict
- ■ S_1 is conflict-serializable
- ■ S_1 is 2PL

- ■ S_2 is recoverable
- ■ S_2 is cascade-less
- ■ S_2 is strict
- ■ S_2 is conflict-serializable
- ■ S_2 is 2PL

- ■ S_3 is recoverable
- ■ S_3 is cascade-less
- ■ S_3 is strict
- ❑ S_3 is conflict-serializable
- ❑ S_3 is 2PL

- ❑ S_4 is recoverable
- ❑ S_4 is cascade-less
- ❑ S_4 is strict
- ❑ S_4 is conflict-serializable
- ❑ S_4 is 2PL

Part 6 Optional: ARIES (Total: 10 Optional Points)

Question 6.1 Recovery (10 Points)

Consider the state of the log and pages on disk shown below. For simplicity we do not show the actual undo/redo actions for updates, but instead show only the affected page. Assume a crash occurred after the last log entry. Answer the following questions:

1. **Analysis**: Write down the result of the analysis phase (`RedoLSN`, Transaction Table, Dirty Page Table)
2. **Redo**: Which pages will be loaded from disk during redo? Which pages will be modified during redo?
3. **Undo**: Write down the additional log entries that will be written during undo.

Log

LSN	Type	TID	PrevLSN	UndoNxtLSN	Data
1	begin	1	-	-	-
2	update	1	1	-	Page 1
3	update	1	2	-	Page 1
4	begin	2	-	-	-
5	update	1	3	-	Page 3
6	update	2	4	-	Page 1
7	update	1	5	-	Page 1
8	begin	3	-	-	-
9	begin_cp	-	-	-	-
10	commit	1	7	-	-
11	begin	4	-	-	-
12	update	4	11	-	Page 1

Disk

PageID	PageLSN
1	6
2	0
3	6
4	0
5	0

Solution

(1):
RedoLSN: 2
Transaction Table: $< T_2, u, 6, - >, < T_3, u, 8, - >, < T_4, u, 12, - >$
Dirty Page Table: $< 1, 2 >, < 3, 5 >$

(2):
Pages 1 and 3 have to be loaded from disk.
Only page 1 will be modified based on redo info from log entries 2, 3, 6, 7, and 12.

(3):
Transactions T_2, T_3, and T_4 will be rolled back. The CLRs written during undoing these transactions' updates is shown below.

LSN	Type	TID	PrevLSN	UndoNxtLSN	Data
13	CLR	4	12	-	Page 1
14	CLR	2	6	-	Page 1

Part 7 Bonus: Physical Optimization (Total: 10 Bonus Points)

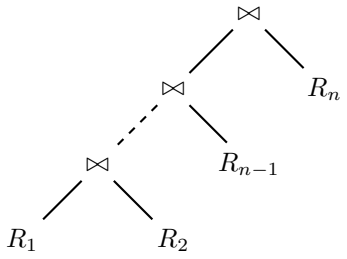
Consider the following relations $R(A, B)$, $S(C, D)$, $T(F, G)$ with $S = \frac{1}{10}$ (10 tuples fit on each page). The sizes and value distributions are:

$T(R) = 1,000$	$V(R, A) = 1,000$	$V(R, B) = 15$
$T(S) = 2,000$	$V(S, C) = 2,000$	$V(S, D) = 2,000$
$T(T) = 100,000$	$V(T, F) = 100,000$	$V(T, G) = 5,000$

Question 7.1 Greedy Join Enumeration (10 Points)

Use the greedy join enumeration algorithm to find the cheapest plan for the join $R \bowtie_{B=C} S \bowtie_{D=F \wedge G=A} T$. Assume that **nested-loop** (not the block based version) is the only available join implementation with the left input being the "outer" (for each tuple from the outer we have to scan the whole inner relation). Furthermore, there are no indicies defined on any of the relations (that is you have to use **sequential scan** for each of the relations). As a cost model consider the **total number of I/O operations**. For example, if you join two relations with $5,000$ and $10,000$ tuples with $S = \frac{1}{10}$, where the $5,000$ tuple relation is the outer, then the cost would be $5,000,000$ (scan the inner 5000 times) + 500 to scan the other once. The total cost is then $5,000,500$ I/Os. Assume that the system supports pipelining for the outer input of a join. That is if you join the result of a join with a relation where the join result is the outer, then there is no I/O cost for scaning the outer. Also under these assumptions you never have to store join results to disk. Hint: You will have to estimate the size of intermediate results. Use the estimation based on the number of values and not the one based on the size of the domain. Use the assumption that the number of values in a join attribute of a join result is the minimum of the number of values in the join attribute of each input.

Write down the state after each iteration of the algorithm using the following notation. Write $(((R_1, R_2), \ldots, R_{n-1}), R_n)^{C,S}$ to denote a plan as shown below with I/O cost C and result size S. Alternatively you are also allowed to draw join trees as shown below.



Solution

Calculate Result Sizes:

Using the formula from class the estimated result sizes are:

$$T(R \bowtie S) = \frac{T(R) \cdot T(S)}{max(V(R,B), V(S,C))} = \frac{1,000 \cdot 2000}{max(15, 2000)} = 1,000$$
$$T(S \bowtie T) = \frac{T(S) \cdot T(T)}{max(V(S,D), V(T,F))} = \frac{2,000 \cdot 100,000}{max(2,000, 100,000)} = 2,000$$
$$T(R \bowtie T) = \frac{T(R) \cdot T(T)}{max(V(R,A), V(T,G))} = \frac{1,000 \cdot 100,000}{max(1,000, 5,000)} = 20,000$$
$$R(R \bowtie S \bowtie T) = \frac{T(R) \cdot T(S) \cdot T(T)}{max(V(R,B), V(S,C)) \cdot max(V(S,D), V(T,F)) \cdot max(V(T,G), V(R,A))} = \frac{1,000 \cdot 2,000 \cdot 100,000}{2,000 \cdot 100,000 \cdot 5,000} = 0.2$$

Initialization:

$$(R)^{100;1,000}, (S)^{200;2,000}, (T)^{10,000;100,000}$$

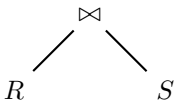
n = 1:

Here we have 6 different options how to join two of the plans from the initialization:

$(R \bowtie S)^{200,100;1,000}$ $(R \bowtie T)^{10,000,100;20,000}$ $(S \bowtie R)^{200,200;1,000}$
$(S \bowtie T)^{20,000,200;2,000}$ $(T \bowtie R)^{10,010,000;20,000}$ $(T \bowtie S)^{20,010,000;2,000}$

As an example take the join $(R \bowtie S)$. Here R is the outer and S is the inner. The cost is computed as: For each tuple from R ($T(R)$) we have to scan S once ($B(S)$ I/Os). Thus, the cost is $B(R) + T(R) \cdot B(S) = 100 + 1,000 \cdot 200 = 200,100$ I/Os. Greedy join enumeration chooses the plan with the lowest cost $(R \bowtie S)$.

$200, 100; 1, 000$



$, (T)^{10,000;100,000}$

n = 2:

Now we need to consider two join options.
For $((R \bowtie S) \bowtie T)$ we pipeline the result of $(S \bowtie R)$ so the cost is:

$$Cost(R \bowtie S) + T(R \bowtie S) \cdot B(T) = 200,100 + 1,000 \cdot 10,000 = 10,200,100$$

Recall that the assumption is that only the outer input of the join can be pipelined. For $(T \bowtie (R \bowtie S))$, the result of the join $(R \bowtie S)$ is the "inner", so we have to store the result of $R \bowtie S$ on disk resulting in $B(R \bowtie S)$ additional I/O. Since $(R \bowtie S)$ has 1,000 result tuples and $S(R \bowtie S) = S(R) + S(S) = 1/10 + 1/10 = 1/5$ it follows that $B(R \bowtie S) = 200$. Thus, the total cost is

$$Cost(R \bowtie S) + B(R \bowtie S) + B(T) + T(T) \cdot B(R \bowtie S) = 200,100 + 100 + 10,000 + 100,000 \cdot 100 = 10,210,200$$

$(R \bowtie S \bowtie T)^{10,200,100;0.2}$ $(T \bowtie R \bowtie S)^{10,210,200;0.2}$

$20, 211, 300; 0.1$