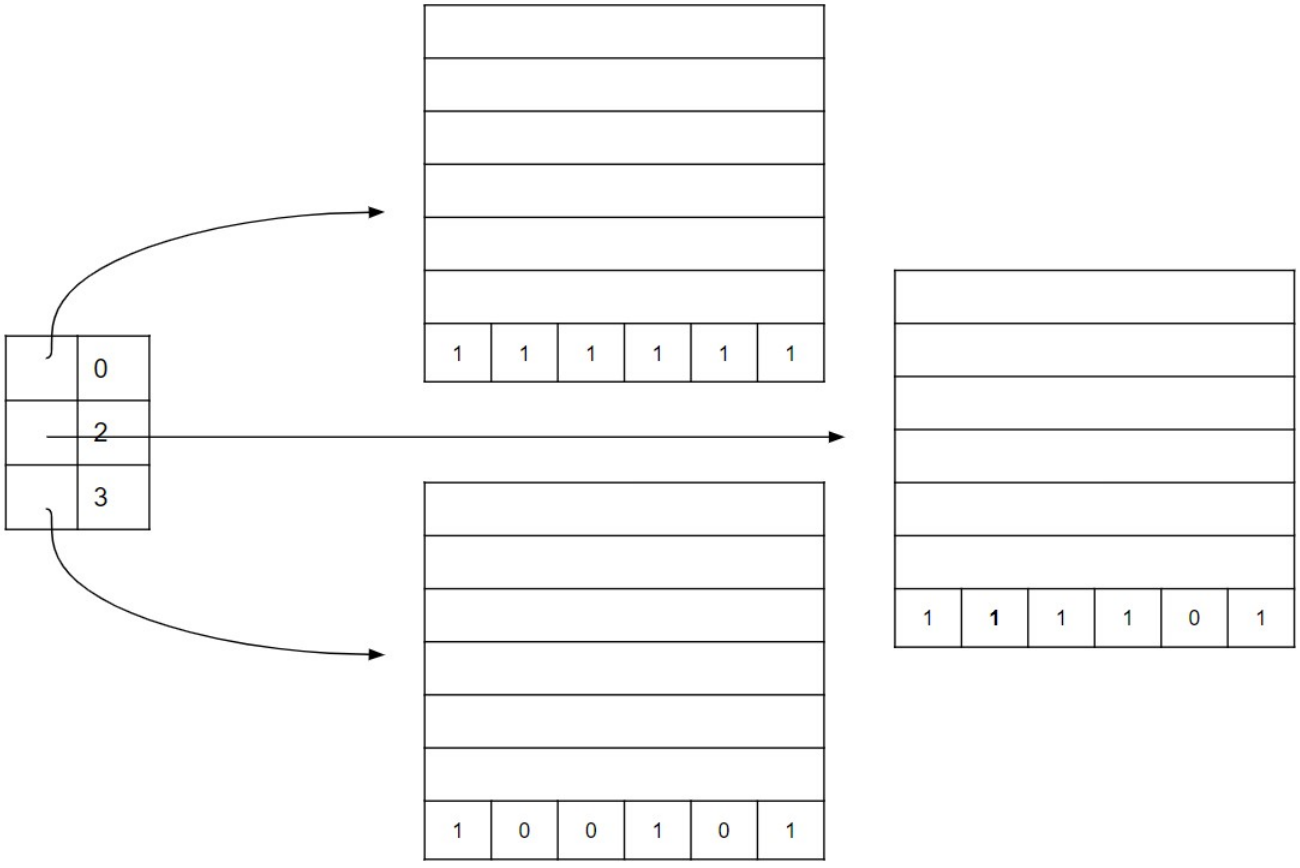
ComS 363
Exam 2

Sean Gordon

April 6, 2020

---

1) (a) The records are fixed unpacked.
(b) Page 1 and 3 would be unchanged, page 2's bitmap would be changed from [1, 0, 1, 1, 0, 1] to [1, **1**, 1, 1, 0, 1] with the record added in the second slot.



(c) Because the records are unpacked, we need only reset the fourth bitmap slot in page 2 to 0: [1, 0, 1, **0**, 0, 1].

---

2)

Assuming 4 I/Os for traversal from root to leaves
Question asks for # read from disk, so output IO will not be considered
Page space = Full page size - reserved space
Tuples per page = (Page space) / (Record size)
Entries per page = (Page space) / (Data size)
Assuming Page space = 4k B, Record size = 40 B, Data size = 20 B.

(a) As the salaries are not indexed, each page must be searched to check for a match. Thus, all 1000 pages must be read.
(b) # of matching pages for a sparse tree = 400 / (Tuples per page)
# of pages of matching data entries = (# of matching pages) / (# Entries per page)
This results in (Traversal) + (# Matching Pages) + (# Pages of matching entries) = 4 + 400 + 2 = 406 pages. (c) # of matching entries for a dense tree = 400 / (Entries per page)
Total pages = (Traversal) + (# of matching pages) + (# Matching Entries) = 4 + 400 + 2 = 406 pages.

---

3) # of blocks of R = $\frac{M}{B-2}$

Cost of join = M + ($\frac{M}{B-2}$) * N.

# of output pages = # of blocks of R = $\frac{M}{B-2}$

Thus total cost = M + ($\frac{M}{B-2}$) * N + ($\frac{M}{B-2}$)

---

4)

Partition: In this phase, we partition S using a hash function to put its tuples into output buffers, writing them to disk as needed. Afterward, we partition R using a hash function to put its tuples into output buffers, writing them to disk as needed. We have now divided all keys into different partitions.

Join: Read a page of R from partition x and construct a hash table of it., then read a page of S from partition x and probe the hash table for matches, concatenating them and putting them into the output buffer, flushing output to disk as necessary.

---

5) Out of the total memory, we need one page for an input buffer and one page for an output buffer, leaving us with one page to hold blocks of R.

Algorithm 1 Grace hash join

```
1: for all Tuple r in R do
2:     bucketNum = H1(r) //Find which bucket r belongs to by using the hash function
3:     Buckets[bucketNum].add(r)
4: end for
5:
6: for all Tuple s in S do
7:     bucketNum = H1(s) //Find which bucket s belongs to by using the hash function
8:     Buckets[bucketNum].add(s)
9: end for
10:
11: for all Bucket b in Table do
12:     Use hash function H2 to build a new hash table
13:
14:     for all Page of r in b do
15:         for all Tuple s in b do
16:             Probe for any matches for s in page r
17:             Store any matches in the output buffer, flush to disk if full
18:         end for
19:     end for
20:
21: end for
```

---

6)

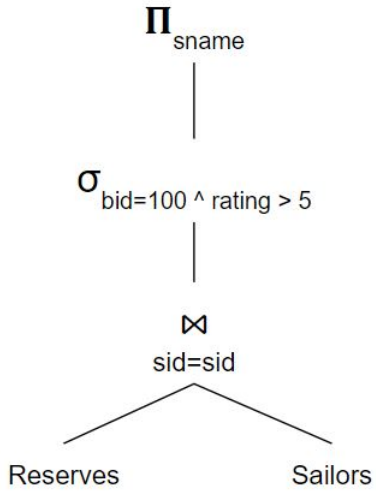
(a) $\pi_{sname}(\sigma_{cid==\text{"}COMS363\text{"}}(\text{Register} \bowtie \text{Students}))$

(b) $\pi_{sname}(\sigma_{semester==\text{"}Spring2020\text{"}}(\text{Register} \bowtie \text{Students}))$

(b) $\pi_{sname}(\text{Students})$ - $\pi_{sname}(\text{Register} \bowtie \text{Students})$

---

7)



---

8)

(a) Using left deep trees allows us to generate fully pipelined plans, meaning we can start from the bottom of the tree and work straight upward, without storing results temporarily. Everything is neatly funneled into a single 'pipe'.

(b) Because A ⋈ B has the fewest I/Os, it will be performed, leaving the result to be merged with C and D in arbitrary order.