>>

Week 4 Exercises

^ >>

❖ Things to Note

- "Disk quota exceeded"? Check your disk usage on /srvr/
  - use the **`du`** command to find "hot spots"
- This weekend is Census Weekend ...
  - if you want to drop a course and not pay for it, drop it now
  - if you want to drop COMP9315, drop it in MyUNSW and email me

<< ^ >>

❖ Assignment 1

- three data usages: readable, computable, storeable
- *could* use the same representation for all three
- or, more likely
  - readable → storeable in **`intSet_in()`**
  - storeable → readable in **`intSet_out()`**
  - storeable → computable in e.g. **`intSet_member()`**
  - *could* use same representation for storeable and computable
- reminder: complex in-memory structures (e.g. BSTs) are not storeable
  - need to serialize/flatten them to make a storeable byte-sequence

<< ^ >>

❖ Exercise: File Merging

Implement a merging algorithm

- for two sorted files, using 3 buffers, with b_1=5, b_2=3
- for one unsorted file, using 3 buffers, with b = 12
- for one unsorted file, using 5 buffers, with b = 27

Assume that we have functions

- **`get_page(rel, pid, buf)`** ... read specified page into buffer
- **`put_page(rel, pid, buf)`** ... write a page to disk, at position pid
- **`clear_page(rel, buf)`** ... make page have zero tuples
- **`sort_page(buf)`** ... in-memory sort of tuples in page
- **`nPages(rel)`**, **`nTuples(buf)`**, **`get_tuple(buf, tid)`**

<< ^ >>

❖ Exercise: Sort-based Projection

Consider a table *R(x,y,z)* with tuples:

```
Page 0:  (1,1,'a')   (11,2,'a')  (3,3,'c')
Page 1:  (13,5,'c')  (2,6,'b')   (9,4,'a')
Page 2:  (6,2,'a')   (17,7,'a')  (7,3,'b')
Page 3:  (14,6,'a')  (8,4,'c')   (5,2,'b')
Page 4:  (10,1,'b')  (15,5,'b')  (12,6,'b')
Page 5:  (4,2,'a')   (16,9,'c')  (18,8,'c')
```

SQL: **`create T as (select distinct y from R)`**

Assuming:

- 3 memory buffers, 2 for input, one for output
- pages/buffers hold 3 **`R`** tuples (i.e. $c_R=3$), 6 **`T`** tuples (i.e. $c_T=6$)

Show how sort-based projection would execute this statement.

<< ^ >>

❖ Exercise: Hash-based Projection

Consider a table *R(x,y,z)* with tuples:

```
Page 0:  (1,1,'a')   (11,2,'a')  (3,3,'c')
Page 1:  (13,5,'c')  (2,6,'b')   (9,4,'a')
Page 2:  (6,2,'a')   (17,7,'a')  (7,3,'b')
Page 3:  (14,6,'a')  (8,4,'c')   (5,2,'b')
Page 4:  (10,1,'b')  (15,5,'b')  (12,6,'b')
Page 5:  (4,2,'a')   (16,9,'c')  (18,8,'c')
-- and then the same tuples repeated for pages 6-11
```

SQL: **`create T as (select distinct y from R)`**

Assuming:

- 4 memory buffers, one for input, 3 for partitioning
- pages/buffers hold 3 **`R`** tuples (i.e. $c_R=3$), 4 **`T`** tuples (i.e. $c_T=4$)
- hash functions: h1(x) = x%3, h2(x) = (x%4)%3

Show how hash-based projection would execute this statement.

COMP9315 21T1 ♢ Week 4 Exercises ♢ [5/12]

<< ^ >>

❖ Exercise: Query Types

Using the relation:

```
create table Courses (
    id        integer primary key,
    code      char(8),   -- e.g. 'COMP9315'
    title     text,      -- e.g. 'Computing 1'
    year      integer,   -- e.g. 2000..2016
    convenor integer references Staff(id),
    constraint once_per_year unique (code,year)
);
```

give examples of each of the following query types:

1. a 1-d *one* query, an n-d *one* query
2. a 1-d *pmr* query, an n-d *pmr* query
3. a 1-d *range* query, an n-d *range* query

Suggest how many solutions each might produce ...

<< ^ >>

❖ Exercise: Cost of Deletion in Heaps

Consider the following queries ...

```
delete from Employees where id = 12345  -- one
delete from Employees where dept = 'Marketing'  -- pmr
delete from Employees where 40 <= age and age < 50  -- range
```

Show how each will be executed and estimate the cost, assuming:

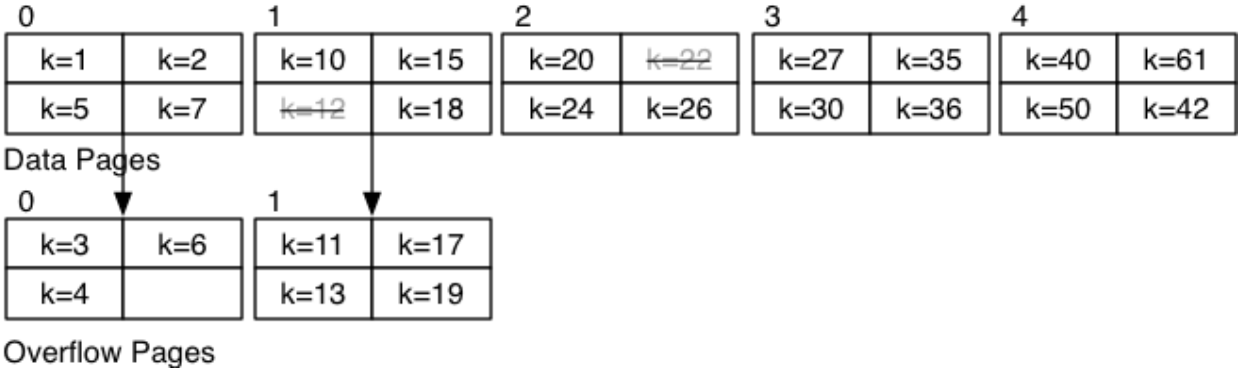
- $b = 100,\ b_{q2} = 3,\ b_{q3} = 20$

State any other assumptions.

<< ^ >>

❖ Exercise: Searching in Sorted File

Consider this sorted file with overflows (*b=5, c=4*):



Compute the cost for answering each of the following:

- `select * from R where k = 24`
- `select * from R where k = 3`
- `select * from R where k = 14`
- `select max(k) from R`

<< ^ >>

❖ Exercise: Optimising Sorted-file Search

The **`searchBucket(f,p,k,val)`** function requires:

- read the p^{th} page from data file
- scan it to find a match and min/max k values in page
- while no match, repeat the above for each overflow page
- if we find a match in any page, return it
- otherwise, remember min/max over all pages in bucket

Suggest an optimisation that would improve **`searchBucket()`** performance for most buckets.

<< ^ >>

❖ Exercise: Insertion into Static Hashed File

Consider a file with *b=4*, *c=3*, *d=2*, *h(x) = bits(d,hash(x))*

Insert tuples in alpha order with the following keys and hashes:

k	*hash(k)*		*k*	*hash(k)*		*k*	*hash(k)*		*k*	*hash(k)*
a	10001		g	00000		m	11001		s	01110
b	11010		h	00000		n	01000		t	10011
c	01111		i	10010		o	00110		u	00010
d	01111		j	10110		p	11101		v	11111
e	01100		k	00101		q	00010		w	10000
f	00010		l	00101		r	00000		x	00111

The hash values are the 5 lower-order bits from the full 32-bit hash.

<< ^ >>

❖ Exercise: Bit Manipulation

1. Write a function to display **`uint32`** values as **`01010110...`**

```
char *showBits(uint32 val, char *buf);
```

Analogous to **`gets()`** (assumes supplied buffer is large enough)

2. Write a function to extract the *d* bits of a **`uint32`**

```
uint32 bits(int d, uint32 val);
```

If *d* > 0, gives low-order bits; if *d* < 0, gives high-order bits

<< ^

❖ Exercise: Insertion into Linear Hashed File

Consider a file with *b=4*, *c=3*, *d=2*, *sp=0*, *hash(x)* as below

Insert tuples in alpha order with the following keys and hashes:

k	*hash(k)*		*k*	*hash(k)*		*k*	*hash(k)*		*k*	*hash(k)*
a	10001		g	00000		m	11001		s	01110
b	11010		h	00000		n	01000		t	10011
c	01111		i	10010		o	00110		u	00010
d	01111		j	10110		p	11101		v	11111
e	01100		k	00101		q	00010		w	10000
f	00010		l	00101		r	00000		x	00111

The hash values are the 5 lower-order bits from the full 32-bit hash.

Produced: 11 Mar 2021