

Indexing

We've been hinting at *indexing* since the second lecture. (In fact, we've hinted at it no fewer than nine times: 2:17, 2:34, 2:35, 5:83, 5:89, 7:108, 7:113, 8:148, 9:152).

Indexing

using a separate file to gain access to records in a data file

- the index is usually ordered (so the data file doesn't have to be)
- the index contains keys and references (RRN, offset) and possibly other stuff



Topic

Folk & Zoellick

- | | |
|--------------------------------------|-------------|
| • Index Basics | §§ 6.1, 6.2 |
| • Record Operations on Indexed Files | § 6.3 |
| • Multiple Keys | §§ 6.5, 6.6 |
| • Inverted Lists | § 6.7 |
| • External Indexes | § 6.4 |

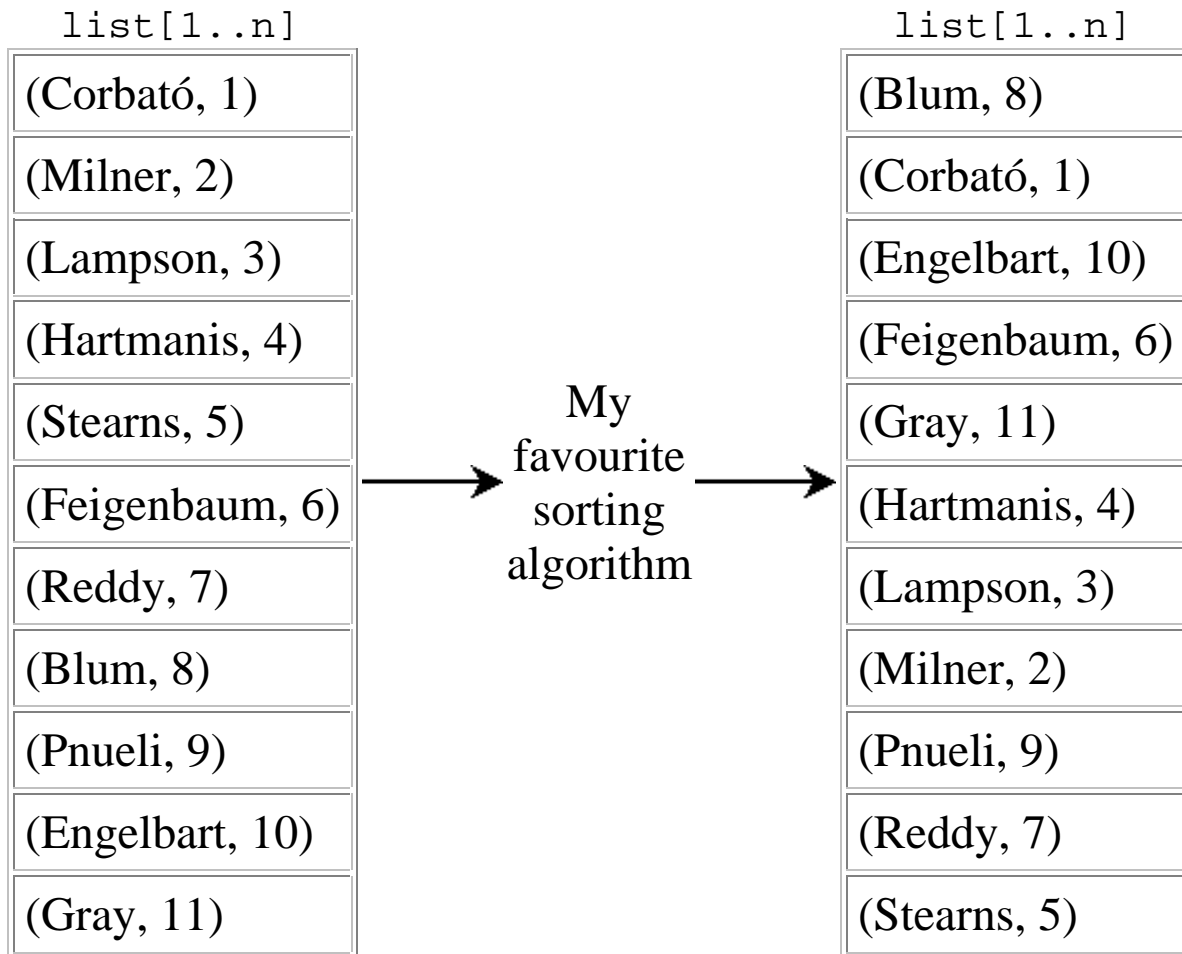
Keysort Revisited

Remember the example we saw for *keysort*?

Corbató	Fernando J.	90	EECS	MIT	...
Milner	Robin	91	TCL	CAMB	...
Lampson	Butler W.	92		MSFT	...
Hartmanis	Juris	93	CISE	NSF	...
Stearns	Richard E.	93	CS	ALBY	...
Feigenbaum	Edward	94	CS	STFD	...
Reddy	Raj	94	SCS	CMU	...
Blum	Manuel	95	CSD	UCB	...
Pnueli	Amir	96	MATS	WEIZ	...
Engelbart	Douglas	97		BOOT	...
Gray	James	98		CNVL	...

The file was too big to fit into RAM for sorting, so we loaded only the keys (and RRN) into RAM and sorted the keys.

Keysort Revisited Continued



At this point we could traverse the sorted list of keys, seeking to the records corresponding to the RRNs appearing in the sorted list and writing the records to a sorted data file. This is the *keysort*. It's also incredibly expensive.

Alternatively, we could just write the sorted list of keys to a file and use it to access the data file. This is an *index*.

Indexing

Index

A file consisting of (at least) a *key field* and a *reference field*, allowing direct access to records in a data file.

Primary Key

A key that is used to identify records in a file uniquely

Reference Field

A field indicating the position of a record in a data file; usually the RRN or a byte offset.



We want to use our sorted list of keys as an index, but we need to clean it up a little first:

- The primary key must be unique, so we must uniquify our key
- If the index file has fixed length records, we can do a simple binary search on it.

Index File Example

Blu142224	8
Cor351614	1
Eng466879	10
Fei134681	6
Gra823015	11
Har018618	4
Lam643080	3
Mil155900	2
Pnu789164	9
Red496173	7
Ste970555	5

Here is the new index file, with the appropriate changes:

- The key is unique, consisting of the first three letters of last name followed by a unique 6-digit number (say, an ACM membership number).
- The records are fixed length: 9 characters for the key and two bytes for the reference (in this case, RRN).

Using the index file to retrieve records from the data file is trivial:

1. Find the desired key in the index file
2. Get the RRN from the appropriate record in the index file
3. Convert the RRN to a byte offset
4. Seek to the byte offset in the data file to retrieve the desired record

A Day in the Life of an Index File

Anything we might want to *do* to the records in a data file, we will want to do to a data file that has been indexed. In particular, we might want to add records, delete records and modify records.

When dealing with an index file there are a few extra considerations. Here is a typical scenario:

1. Load the whole index file into RAM
2. Perform record operations to both the index file and the data file:
 - add records
 - delete records
 - modify records
3. Rewrite the index file from the modified version in RAM

□

Q: Why load the whole index file into RAM?

A:

Q: When should we rewrite the index file?

A:

Q: What if we're doing all this on a Novell system and it crashes before we can rewrite the index file (or while we're rewriting the index file (or both))?


Adding a Record

Hahn	Udo	99	CLL	FREI	...
------	-----	----	-----	------	-----

Since we no longer care about the order of records in the data file, adding a record is simple: just stick it on the end.

We must also add the record to the index file:

Blu142224	8		
Cor351614	1		
Eng466879	10		
Fei134681	6		
Gra823015	11		
Har018618	4		
Lam643080	3		
Mil155900	2		
Pnu789164	9		
Red496173	7		
Ste970555	5		



Q: How expensive is record addition?

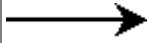
A:

Deleting a Record

Lampson	Butler W.	92		MSFT	...
---------	-----------	----	--	------	-----

(*sorry Butler*)

Blu142224	8		
Cor351614	1		
Eng466879	10		
Fei134681	6		
Gra823015	11		
Hah001159	12		
Har018618	4		
Lam643080	3		
Mil1155900	2		
Pnu789164	9		
Red496173	7		
Ste970555	5		



Q: How expensive is record deletion?

A:

Q: What about the data file?

A:

Secondary Keys

Usually we don't know the value of the primary key for the record we're searching for (especially if the primary key is dataless). We usually know the value of a secondary key.

Secondary Key

A key on which record searches are usually performed; often not unique.



Let's flash way back to the second lecture...

Secondary Keys (cont.)

Remember *this* file? I've added a primary key for the file composed of bits of other fields stuck together to make a unique key.

BOSCAR001	BOSE	JIT	CARLETONU	CS384	...
BOSCAR002	BOSE	JIT	CARLETONU	CS102	...
BARMAN001	BARKER	KEN	UMANITOBA	074-438	...
BARMAN002	BARKER	KEN	UMANITOBA	074-452	...
BAROTT001	BARKER	KEN	UOTTAWA	CSI2131	...
BAROTT002	BARKER	KEN	UOTTAWA	CSI4900	...
BOYOTT001	BOYD	SYLVIA	UOTTAWA	CSI5166	...
BOYOTT002	BOYD	SYLVIA	UOTTAWA	CSI4900	...
HOLOTT001	HOLTE	ROBERT	UOTTAWA	CSI1101	...
HOLOTT002	HOLTE	ROBERT	UOTTAWA	CSI4900	...
ROYOTT001	ROY	DAMIEN	UOTTAWA	MAT1741	...
ROYOTT002	ROY	DAMIEN	UOTTAWA	MAT3543	...
ROYOTT101	ROY	LANGIS	UOTTAWA	ELG4102	...
MORQTR001	MORIN	JOHANNE	UQTR	ASY1006	...
MORQTR002	MORIN	JOHANNE	UQTR	SIF1016	...

We're very unlikely to want to search on the primary key. On the other hand, we are likely to want to search on fields like the last name field.

So what's the solution?

Secondary Key Index

The solution is to create another index file!

Primary Index

BARMAN001	3
BARMAN002	4
BAROTT001	5
BAROTT002	6
BOSCAR001	1
BOSCAR002	2
BOYOTT001	7
BOYOTT002	8
HLOTT001	9
HLOTT002	10
MORQTR001	14
MORQTR002	15
ROYOTT001	11
ROYOTT002	12
ROYOTT101	13

Secondary Index

BARKER	BARMAN001
BARKER	BARMAN002
BARKER	BAROTT001
BARKER	BAROTT002
BOSE	BOSCAR001
BOSE	BOSCAR002
BOYD	BOYOTT001
BOYD	BOYOTT002
HOLTE	HLOTT001
HOLTE	HLOTT002
MORIN	MORQTR001
MORIN	MORQTR002
ROY	ROYOTT001
ROY	ROYOTT002
ROY	ROYOTT101

A Better Secondary Key Index

The secondary index in this example contains much duplication. Furthermore, if a new record is added to the data file, we must add a new record to the secondary index (possibly requiring a bunch of file manipulation). Here's a better secondary index:

BARKER	BARMAN001	BARMAN002	BAROTT001	BAROTT002	...
BOSE	BOSCAR001	BOSCAR002			...
BOYD	BOYOTT001	BOYOTT002			...
HOLTE	HOLOTT001	HOLOTT002			...
MORIN	MORQTR001	MORQTR002			...
ROY	ROYOTT001	ROYOTT002	ROYOTT101		...



This secondary index, containing a list of fields containing primary key values for each value of a secondary key, is called an *inverted list*.

Now instead of having to add a new record to the secondary index every time we add a record to the data file, we only have to add an entry to the next available primary key field.

The Usual Suspects

Q: What's wrong with using multiple fields for primary key values in an inverted list?

A:

Q: What's the alternative?

A:



Q: The usual procedure for using an index file is to load the whole thing into RAM to make searching and reorganizing maximally efficient.

What if the index file doesn't fit into RAM?**A:**

Q: What if there are still occasions where we want to read through the entire datafile in logical order?

A: