Data Organization and Processing

File Organizations

David Hoksza, Petr Škoda
http://siret.cz/hoksza

Overview

- Terminology
  - records, files, …
- Querying files
- File organizations
  - types of file organizations
  - implementation of file operations

Data organization - terminology

- **Database** (*databáze*)
  - collection of related data (named files) in secondary memory
  - elements are related

- **File** (*soubor*)
  - named collection of records

- **Record** (*záznam*)
  - representation of an application object (person, car, flat, …)
  - **set** of **fields** (*pole*)
    - elementary unit of data
    - content provided by a user or a program
    - fixed or variable length
      - person – name, occupation, height, sex, …
      - car – license plate, type, color, …
  - **field F_i** together with its **domain dom(F_i),** i.e. pair `Fi:dom(Fi)` , is called **attribute** (*atribut*)
    - domains: INT, FLOAT, STRING, …

Data record

- **Logical**
  - attribute set

- **Physical**
  - **physical representation** of a logical record of size ***R*** (bytes) **on the medium**
  - contains **additional metadata** (such as record delimiters, etc.)
  - **Records** are stored **in blocks** of size ***B*** (bytes)
    - transferred between primary and secondary memory

Data record types

- **Fixed** length
  - **file header** contains **number of records** and **length of each field**
  - record can be accessed using the record number

- **Variable** length
  - variable length of the attributes (e.g. name, description, …)
  - when **similar objects relate to a nonuniform set of data** (e.g. for different types of employees different attributes are relevant)
  - **optional attributes** (e.g. product picture)
  - **attributes holding records** (e.g. an order with multiple items or employee with multiple phone numbers)

Record blocking (1)

- **Blocking factor** (*blokovací faktor*) ***b***
  - number of records in a block
  - $\left\lfloor \frac{B}{R} \right\rfloor$ **ratio**
    - **B** … block size
    - **R** … record size

- Basic division based on blocking
  - **non-blocked** records
    - 1 record fits 1 block
  - **blocked** records
    - N records fit 1 block
  - **overflown** records
    - 1 record fits N blocks

Record blocking (2)

Fixed blocking

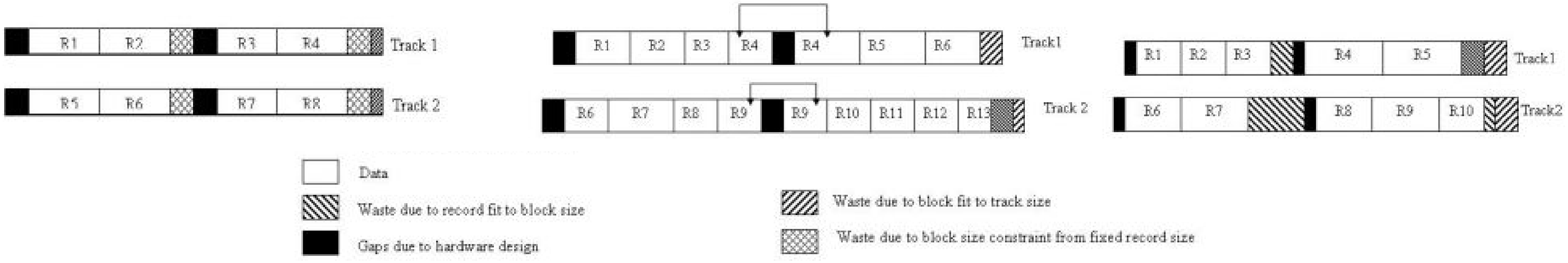
- fixed-length records
- possible internal fragmentation
  - unused space

Variable-length spanned blocking

- variable-length records
- no unused space
- continuation indicated by a pointer to the next block
- hard to implement
- need more time to read records in 2 blocks

Variable-length unspanned blocking

- variable-length records
- no spanning
- can show high internal fragmentation



Files

- Storage in secondary memory
  - **reading** from a file requires its contents to be read into main memory **in blocks**
  - when **modifying** a file its content needs to be **read** into main memory, **modified** and changes are then **written back** to the secondary memory
- **Homogenous**
  - stores fixed-length records of the same type
  - declared as $S(A_1: D_1, \ldots A_N: D_N)$
- **Non-homogenous**
  - stores either variable-length records or records of different types

Operations over files

- **Formation/Termination**
  - **CREATE, REMOVE**
    - builds or removes auxiliary structures
- **Modification**
  - **INSERT**
    - inserts a new record into a file
  - **UPDATE**
    - updates an existing record
  - **DELETE**
    - removes (or invalidates) an existing record
- **Querying**
  - **FIND**
    - finds a record in a file corresponding to the query key
  - **FETCH**
    - gets a record from a file (secondary memory) into main memory based on specified query conditions (query key)
- **Maintenance**
  - **REORGANIZE/REBUILD**
    - due to the efficiency reasons not all changes are directly projected into the underlying file organization

Querying files

- **One-dimensional queries**
  - querying according to only one attribute
    - fetch employees with age > 35
    - fetch cars with color = ‘red’

- **Multi-dimensional queries**
  - querying according to multiple attribute
  - **total match** (*na úplnou shodu*)
    - the values of all attributes entered
  - **partial match** (*na částečnou shodu*)
    - only values of some attributes entered
  - **total interval match** (*na úplnou intervalovou shodu*)
    - for each attribute the interval of values entered
  - **partial interval match** (*na částečnou intervalovou shodu*)
    - for selected attributes the interval of values entered

File organization

- File organization (*souborová organizace*)
  - how to organize a set of records in a file and how to access them
  - the description of the **logical memory structure** together with **algorithms** for handling that structure
- Files in the organization
  - FO can contain multiple files
- Optimal choice of organization depends on the usage
  - accessing one record, accessing multiple records, accessing whole file, …

File organization levels

- **Logical schema**
  - **algorithms**
    - defined to secure optimal manipulation with the data for given task
    - minimization of the number of operations while manipulating the file
  - **logical blocks (pages)**
    - logical blocks structure
    - logical blocks relations
    - logical blocks manipulation
    - fill factor
  - **logical files**
    - how are logical pages related to each other
    - **primary file**
      - data
    - **auxiliary files**
      - efficient access to the data (indexes, …)
- **Physical schema**
  - **mapping** between logical schema and physical pages
  - physical files
    - One logical file can span multiple physical files and the other way around
- **Implementation schema**
  - **implementation** of the physical files
  - **shielded** from the logical level **by OS**

Unlike in case of in-memory data structure, even the structure itself needs to be stored somewhere.

File organization types (1)

- **Heap file** (*halda*)
  - **variable-length** records
  - a record **placed** always at the **end of the file**

- **Sequential file** (*sekvenční soubor*)
  - **unsorted**
    - as heap file but contains **fixed-length** records
  - **sorted**
    - stores records in **sequential order**, based on the **value of the search key** of each record

File organization types

- **Indexed sequential file** (*index-sekvenční soubor*)
  - records stored based on the order of the search key on which index is built
- **Index file** (*indexový soubor*)
  - resembles indexed sequential file but multiple indexes can be present
- **Hashed file** (*hashovaný/hešovaný soubor*)
  - a hash function is computed on a chosen attribute of a record; the resulting structure specifies in which block of the primary file given record should be placed/found

Heap File (HF)

- **Data not sorted** in any way
- File is not homogeneous, i.e. contains records of various types/lengths
- Usually used only along with another supporting structure

- INSERT
  - fetch the last block in the file and append the new record
  - **O(1)**
  - realtime → **O(1) (s+r+btt)**
- FIND
  - whole file needs to be scanned if the search key attribute set is not unique
  - **O(N) → O(N/b)**
  - realtime
    - worst case: **O(N/b)(s+r+btt)**
    - best case: **s+r+O(N/b) btt**

We consider *b* as an average blocking factor (the records are variable-length)

HF – reasons for variable-length records

- When similar objects relate to a non-uniform set of data

- Examples
  - different types of employees within our application → a need to store different information for different roles
  - some records may have non atomic data types, e.g., an unspecified number of repeating groups of attribute values → variable number of items in orders

- Solution
  - variable-length records
  - setting maximum length for a given field

Unsorted sequential file (USF) organization

- Data not sorted in any way
- Suitable when data are collected without any relationship to other data
- INSERT
  - fetch the last block in the file and append the new record
  - **O(1)**
- FETCH
  - Single record
    - whole file needs to be scanned if the search key attribute set is not unique
    - **O(N) → O(N/b)**
    - realtime
      - worst case: **O(N/b)(s+r+btt)**
      - best case: **s+r+O(N/b) btt**

Block	Name	Department	…
0	Galvin Janice	Purchasing	
	Walters Rob	Marketing	
	Brown Kevin	Marketing	
1	Walters Rob	Developlment	
	Duffy Terri	Research	
	Brown Kevin	PR	
2	Duffy Terri	Developlment	
	Walters David	Production	
	Brown Kevin	Purchasing	
3	Matthew Gigi	Purchasing	
	Walters Rob	PR	
	…	…	

Sorted sequential file (SSF) organization

- **Records sorted** in the file on the primary search key
- File can be sorted only according to **one attribute** → the **most often searched** one (if querying is the prevalent operation)

Bl oc k	Name	Department	…
0	Brown Kevin	PR	
	Brown Kevin	Purchasing	
1	Brown Kevin	Marketing	
	Duffy Terri	Developlment	
2	Duffy Terri	Research	
	Galvin Janice	Purchasing	
3	Matthew Gigi	Purchasing	
	Walters David	Production	
4	Walters Rob	Marketing	
	Walters Rob	Developlment	
5	Walters Rob	PR	
	…	…	

SSF – FETCH

- **Sequential scan**
  - takes linear time - on average N/2 records (N/b/2 blocks) need to be checked
  - $\boldsymbol{O}(\boldsymbol{N}) \rightarrow \boldsymbol{O}\left(\frac{\boldsymbol{N}}{\boldsymbol{b}}\right)$
    - However, if fetching a range containing k records, only the first record needs to by found, the rest are fetched sequentially $\rightarrow \frac{N/b}{2} + \frac{k}{b} \rightarrow \frac{N/b}{2}$(s+r) +($\frac{N/b}{2} + \frac{k}{b}$)$btt$
- **Binary search** *(půlení intervalu)*
  - $\boldsymbol{O}(\log_2(\boldsymbol{N})) \rightarrow \boldsymbol{O}(\log_2(\boldsymbol{N}/\boldsymbol{b})) \rightarrow \boldsymbol{O}(\log_2(\boldsymbol{N}/\boldsymbol{b}))$**(s+r+btt)**
  - address of the i-th block can be obtained from the header information (the file is homogeneous)

SSF - modification

- **INSERT**
  - inserting new record into the structure would be costly since all the following records would have to be shifted
  - **auxiliary file/blocks** called **overflow file/bucket** (*stránka/oblast přetečení*) needs to be established where the new records are inserted
  - the file is **periodically reorganized**
- **UPDATE**
  - simple if the update does not include the primary search key
- **DELETE**
  - deleted records are not directly removed since reorganization would have to take place
  - **a bit designating deleted records is set**
  - deleted records are removed during periodical reorganization

SSF - size

- SSF storing employee records - ***Employee (Name, Department, Phone, SSN, …)***
- Record size $R = 500B$
- Block size B $= 8KiB = 8{,}192B$
- Records count $= 50{,}000$

- Block factor $b = \left\lfloor \frac{B}{R} \right\rfloor = 16$
  - **one page** can accommodate up to **16 employees**
- **File size**
  - $\frac{50{,}000}{16} = 3{,}125$ blocks
  - $3{,}125 * 8{,}192 = 25{,}6\ MB$

Indexed sequential file (ISF) organization

- **Primary file**/area (*primární soubor*)
  - data file sorted based on the search key
- **Index/secondary file**/area (*indexový soubor*)
  - primary index
- **Overflow file**/area (*oblast přetečení*)
- Data can be accessed either using the index or sequentially
  - favorable when high percentage of the records is to be fetched

Index

- An **index** is an **auxiliary structure** for a data file that consists of a specifically arranged structure containing **key-pointer pairs**
  - a key-pointer pair might be, e.g., a *Name* value and a disk-address pointer that points to the block that accommodates a record with the given *Name* value
  - a structure containing these pairs would form an index
- **Storage of the index**
  - **main memory**
  - **secondary memory**
    - if the index does not fit in the main memory (can't be cached) it has to be accessed in the same manner as the primary file
    - accessing index must also be taken into account when computing the find/fetch time

ISF – single-level index

- Constant-width records

- Index file associates a key (*Name*) with a pointer to the **block** containing that key

Index file

Bl oc k	Name	
6	Brown	0
	Clinard	1
	Duffy	2
	Leavy	3
	Peagler	4
7	Walters Rob	5
	…	

Primary file

Bl oc k	Name	Department	…
0	Brown Kevin	PR	
	Berkman Doloris	Purchasing	
1	Clinard Stephnie	Marketing	
	Coolidge Emily	Developlment	
2	Duffy Terri	Research	
	Galvin Janice	Purchasing	
3	Leavy Shirleen	Purchasing	
	Matthew Gigi	Production	
4	Peagler David	Marketing	
	Shackelford Elsie	Developlment	
5	Walters Rob	PR	
	…	…	

ISF – multiple-level index

- Index is typically **hierarchical** so that **less accesses** are needed
- Master level uses to be stored in the main memory
- In real use, the **blocking factor of the index** tends to be **much higher** then the one of the primary file

Index file
2. (top/master) level

Block	Name	
8	Brown	6
	Walters Rob	7

Index file
1. (base) level

Block	Name	
6	Brown	0
	Clinard	1
	Duffy	2
	Leavy	3
	Peagler	4
7	Walters Rob	5
	…	

Primary file

Block	Name	Department	..
0	Brown Kevin	PR	
	Berkman Doloris	Purchasing	
1	Clinard Stephnie	Marketing	
	Coolidge Emily	Development	
2	Duffy Terri	Research	
	Galvin Janice	Purchasing	
3	Leavy Shirleen	Purchasing	
	Matthew Gigi	Production	
4	Peagler David	Marketing	
	Shackelford Elsie	Development	
5	Walters Rob	PR	

ISF – FETCH

- Searching for a **specific value** (query key)
  - check the top level of the index and identify a ***key-value*** **pair with the highest value lower** than the query key
  - **fetch** the **block** referenced by the *value*
  - **repeat** the previous steps with lower index levels **until a primary file block is reached**
  - **search the primary file block** for the specified key

- Searching for a **range of values**
  - **search** for the **lower bound key of the interval**
  - **sequentially scan** the blocks of the primary file until record corresponding to the upper bound key is found

ISF – size & time (1)

- **Fetch time** depends on the index **tree height**
- **height =** $\left\lceil \boldsymbol{log_p N/b} \right\rceil$, $b = \left\lfloor \frac{B}{R} \right\rfloor$, B = block size, R = (average) record size, $\boldsymbol{p} = \left\lfloor \frac{\boldsymbol{B}}{\boldsymbol{K+P}} \right\rfloor$ block factor for index records (K = key size, P = pointer size)
  - **number of disk accesses** corresponds to the **number of index levels**
  - number of index **levels** can be **decreased** by
    - **decreasing K** (key size)
    - **increasing B** (block size)

ISF – size & time (2)

- ISF storing employee records - **Employee (Name, Department, Phone, SSN, …)**
  - Let the name of the employee (30B) be the key
- **R** = **500 B, B** = 8 KiB = **8,192 B, b** = **16,** records count = 1,000,000

- **Primary file size**
  - $\lceil 1{,}000{,}000/16 \rceil$ = **62,500 blocks**
  - 62,500 * 8,192 = **512 MB**
- **Index height**
  - p = 215
  - **height** = $\lceil \log_{215} 1000000/16 \rceil$ = **3**
- **Index size**
  - key size = 30 B, value (pointer) size = 8 B, **b** for index file = 215
  - $\lceil 62{,}500/215 \rceil$ = **290 blocks** → we will need **2 additional blocks in the next level** and **1 block in the top level**
  - 293 * 8,192 ≅ **2.4 MB**

ISF - update

- **Index structure** stays **static** when inserting data → **new records** need to be stored in **reserved areas** within the primary file
  - when an index is created the index nodes are fixed and do not change during modifications of the primary file
  - **pockets/buckets**
    - each record/block contains a pocket where the overflown records are stored

ISF – pockets/buckets

- **Overflown data** are inserted into a **new block** (created **dynamically**) pointed to by the overflown block
- **Buckets can be chained** and therefore theoretically the ISF does not need to be rebuilt
  - however **long pockets decrease efficiency**

- **Pointers** to the overflow area
  - **with each record**
    - takes more space
    - shorter sequences in the overflow area
  - **with each block**
    - longer sequences in the overflow area
    - it is possible to reserve some space in the block

ISF – pros and cons

Pros	**Cons**

• Fast access using primary search key
• Shares pros of the sequential file
	• Fast access **only** when using primary search key
	• Problems with primary file when updating
	  • pockets – slows down access to the data
	• Possible need of reorganization
	  • time consuming operation

Indexed file organization (IF)

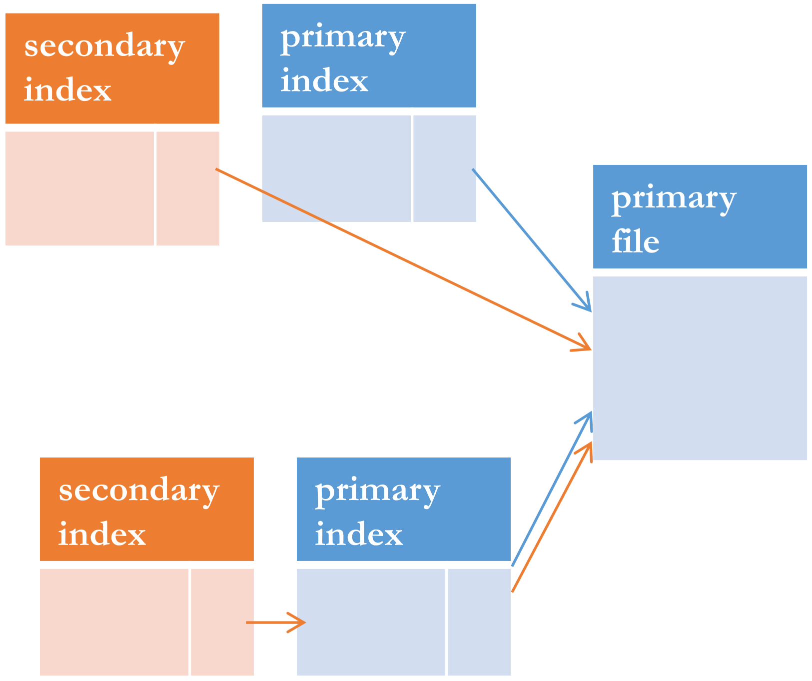
- **Motivation**
  - being able to search the file according to different attributes without the need to sequentially scan the whole file
    - in the ISF organization this is possible only for one attribute
- **Implementation**
  - the **primary file** stays **unsorted or** is **sorted** according to one key only (**primary index**)
  - **for each query key** an **index** file can be built → one primary data file, **multiple index files**
    - indexes can be of various types (even within one IF)
- IF basically **corresponds to** a standard DB **table** where we have **one table** and **multiple indexes** built over it (possibly of different types)

Primary vs secondary Index

- **Primary index** (*primární index*)
  - index over **the attribute** based on which the **records in the primary file are sorted**
  - if the value of the primary attribute is modified the file needs to be reorganized → should be relatively invariable
  - well-suited for range queries
  - there does not have to be a primary index in the IF
- **Secondary index** (*sekundární index*)
  - IF can have **multiple** secondary indexes
  - **range queries for long ranges can be very expensive** (an extreme example is a sequential scan based on a secondary index which can lead to an extremely deteriorated performance)

Direct vs. indirect indexing/addressing

- **Direct indexing** (*přímé indexování*)
  - index is **bound directly to the record**
  - primary file reorganization leads to modification of all the indexing structures
- **Indirect indexing** (*nepřímé indexování*)
  - **secondary indexes** contain **keys of the primary index** and not pointers to the primary file
  - accessing a record needs one more accesses to the primary index
  - if the primary file is reorganized, the secondary indexes stay intact



IF – example (1)

- Unsorted primary file
- Records in the index files are sorted based on given key

Block	Department	
33	Development	24
	Marketing	25
	Production	26
34	Purchasing	27

Block ID

Offset

Block	Department	
24	Development	1.1
	Development	4.1
	Marketing	1.0
25	Marketing	4.0
	PR	0.0
	PR	5.0
26	Production	3.1
	Purchasing	0.1
	Purchasing	2.1
27	Purchasing	3.0
	Research	2.0
	…	…

Primary file

Block	Name	Department	…
0	Brown Kevin	PR	
	Berkman Doloris	Purchasing	
1	Clinard Stephnie	Marketing	
	Coolidge Emily	Development	
2	Duffy Terri	Research	
	Galvin Janice	Purchasing	
3	Leavy Shirleen	Purchasing	
	Matthew Gigi	Production	
4	Peagler David	Marketing	
	Shackelford Elsie	Developlment	
5	Walters Rob	PR	
	…	…	

IF – example (2)

- Primary file sorted based on the primary key "Name"

Block	Name	
28	Brown	24
	Matthew	25
	Young	26
29	Zhang	27

Block	Name	
24	Brown	0
	Clinard	1
	Duffy	2
25	Leavy	3
	Peagler	4
	Walters Rob	5
26	Young	6
	…	
27	Zhang	

Primary file

Block	Name	Department	…
0	Brown Kevin	PR	
	Berkman Doloris	Purchasing	
1	Clinard Stephnie	Marketing	
	Coolidge Emily	Development	
2	Duffy Terri	Research	
	Galvin Janice	Purchasing	
3	Leavy Shirleen	Purchasing	
	Matthew Gigi	Production	
4	Peagler David	Marketing	
	Shackelford Elsie	Developlment	
5	Walters Rob	PR	
…	…	…	

IF – example (3)

- If the primary file is sorted on the the index key, addressing blocks in the primary file is sufficient
  - Less entries in the leaf level of the index than the actual records
- If the primary file is not sorted on the index key, leaf level needs to address every record in the primary file
  - E.g. department records are spread all over the primary file

IF – example (4)

- Primary index "Name" and secondary index "Department", primary file sorted over primary index

Secondary index

Block	Department	
8	Development	Coolidge
	Development	Shackelford
	Marketing	Clinard
9	Marketing	Peagler
	PR	Brown
	PR	Walters
10	Production	Matthew
	Purchasing	Galvin
	Purchasing	Leavy
11	Research	Duffy

Primary index

Block	Name	
6	Brown	0
	Clinard	1
	Duffy	2
7	Leavy	3
	Peagler	4
	Walters Rob	5
	…	

Primary file

Block	Name	Department	…
0	Brown Kevin	PR	
	Berkman Doloris	Purchasing	
1	Clinard Stephnie	Marketing	
	Coolidge Emily	Development	
2	Duffy Terri	Research	
	Galvin Janice	Purchasing	
3	Leavy Shirleen	Purchasing	
	Matthew Gigi	Production	
4	Peagler David	Marketing	
	Shackelford Elsie	Developlment	
5	Walters Rob	PR	

IF – comments

- The **indirect indexing** is in nowadays databases often the method of choice
- It is desirable for the **primary index** to stay **in main memory** and not to be swapped to disk
- **Keys** of a **primary index** are advised to be **small**, e.g., integers and not long strings

Hashed file organization

- **Direct access**
  - suitable when **accessing individual records** given **one unique key**
  - uses a **hash function** to **map keys to** the block/page **addresses**
  - **hashing field**
    - the attribute over which the quick access is provided
  - typically $\boldsymbol{O(1)}$ time
    - if the data can not fit into the page when inserting, an overflow strategy is employed

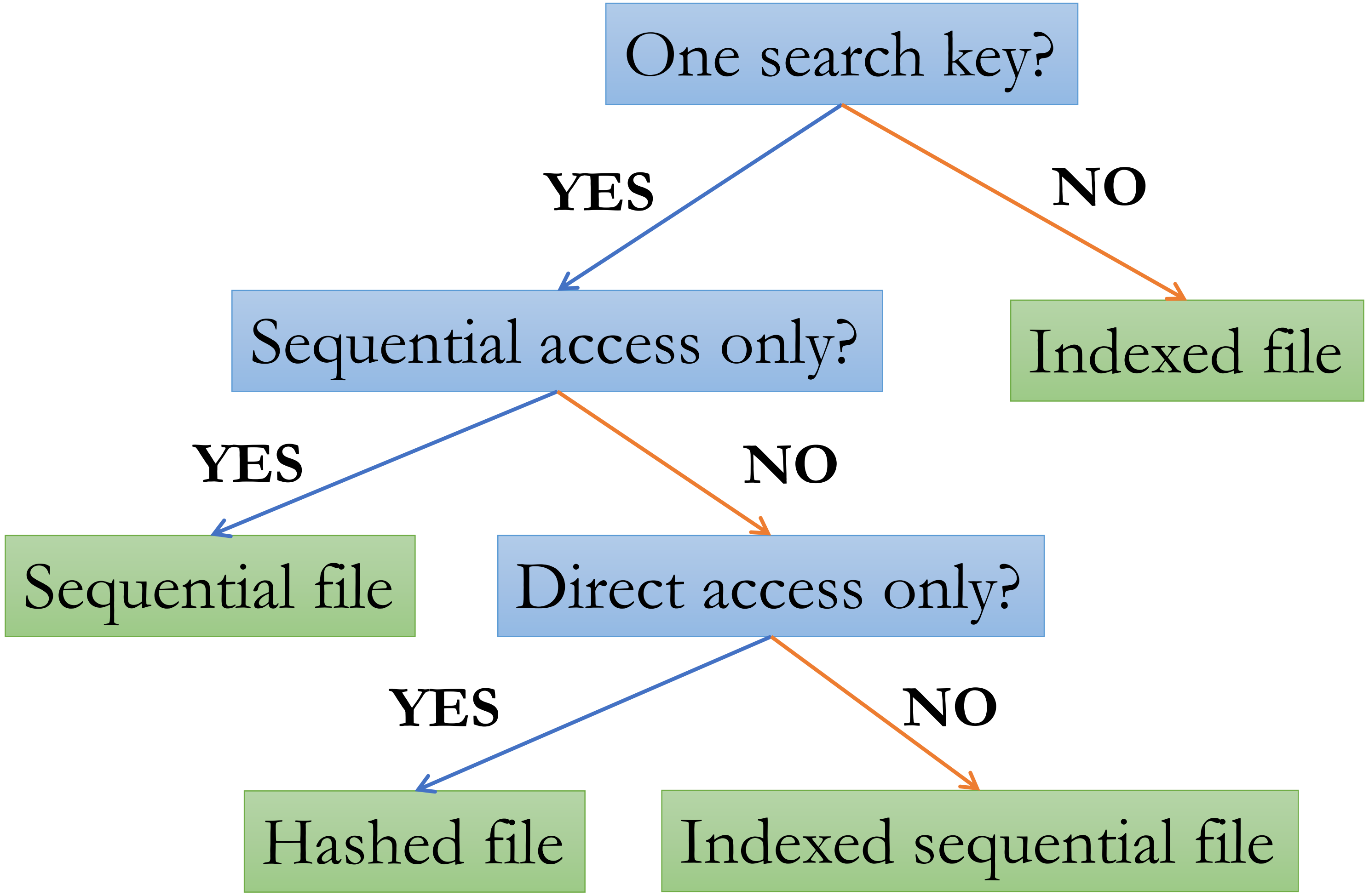
HF - example

- Hash function *h* uses three lower bits of the first integer field to address a page where the record is stored
  - `h({42, true, "foo"})` → 2 ($42 = 101010_2$)
  - `h({14, true, "bar"})` → 6 ($14 = 1110_2$)
  - `h({26, false, "false"})` → 2 ($26 = 11010_2$)

- Placement within the page is not specified

- When the file is being reorganized, the pages are filled to only ,e.g., 80%
  - avoiding an overflow immediately after the structure is built

Operating instructions



Demo

https://skodapetr.github.io