Glossary:

Trees:

AVL Tree-A self adjusting binary tree structure that can guarantee good access time for data in memory. Fast retrieval, slow insert/removal.

Splay: self adjusting tree, recently accessed are easiest to access. Can get way out of balance. Use when there is current structure.

BTree: every page has ceil(m/2). Tree of sorted lists. Larger nodes, less wasted space. Logm copies in tree.

B+Tree: A Btree that provides sequential access to the data as well as fast indexed access.

2-3 tree: Always balanced, time to maintain is not significant.

Red-Black Trees: the path from the root to the farthest leaf is no more than twice as long as the path from the root to the nearest leaf

UNIX Commands:

Pipe: Carries data from one process to another.

Read(Source, Dest, Size)

Tail – read last ten lines of the file

Fseek(file, byteoffset, origin)

FileStructures: The organization of data on secondary storage devices.

Buffering: When input or output is saved up rather than sent off to its destination immediately, we say that it is buffered.

Block: unit of data, corresponds to amount of data transferred in a single access. **Block Factor**: Records per block **Count sublock:** block that preceeds each data block & contains info about data block like byte count and address **Interblock gap:** Interval of blank space that separates sectors, blocks, or subblocks on disk.

**Extent:** Adjacent clusters allocated to a file. More extents, more dispersed a file is.

**Extensibility-**the possibility of file orgs that makes is possible to extend the types of objects that the format can accommodate. **Portability-**Char that describes how amenable they are to access on a variety of machines. **Metadata**- Data that describes the data in a file. **Coalescence-** two deleted avail rec slots are combined. **DoubleBuffering**- Swapping the roles of the 2 buffers after each i/o operation.

QUIZZES:

1: Class members consist of data members and methods.

FIRST development that was added to sequential files to facilitate searching was the index.

3: FAT maps clusters locations on the disk.

4: od -> dump file’s contents, dd -> facilitates data conversion, or byte swapping.

6: A **Stack** is recommended as simplest for keeping track of available INTERCHANGEABLE rec slots.

7: Simple index allows easy work for **entry sequenced** disk file. For read-only, use binding during the preparation of the file. Tight binding takes place at the preparation of the file. For dynamic files bind during prog execution. Data struct where 2ndary keys leads to a set of one or more pri keys -> **INVERTED LIST**.

Exam 1:

**In unix rmhardLi**, IF(file is deleted) ”delete file” ELSE “decrement link count.

**RAID facilitates**: striping, thoroghput, parity, redundancy, mirroring, error correction.

**Sector** based track partitioning doesn’t allow for varying sizes

**Inhieritance** is better epressed by “is-a” relationships

2 ops for when blks are not integral number of records. **1**: Blocking without spanning sectors +each record access is 1 seek –more internal fragmentation **2** Blocking with spanning sectors + less external fragmentation – records can span sectors, can have 2 seeks per record.

5methods to organizing recs: 1. Indexing 2. Record Delimited 3.fixed length 4. Variable length 5. Fixed # of fields.

**BoyerMoore-** Worst case (m+n), becomes MORE efficient as keys become long. **Radix**- use when few chars per word. **Counting Sort**- use when many duplicated words of varying frequencies.

**Insertion**- Online sorting algorithm

**Record insertion**: Best fit, worst fit, first fit.

**Avg #** tracks crossed for each seek -> 1/3

DISK EQUATIONS:

Track capacity = # sectors/ track \* bytes/ sector

Cylinder Cap= # tracks/cylinder \* track capacity

Drive Capacity = # cylinders \* cylinder capacity

Transfer Time = #bytes transferred / #bytes on track \* rotation time.

MERGE EQUATIONS: k-way merge

(1/k)xsizeofmem = (1/k)xsizeofeachrun

K runs = k times through the entire file

Each K runs is K accesses

HW based solutions:

Increase memory-fewer runs, fewer seeks/run

# drives – reduces the movement of disk between runs **Increase IO ch –** give each drive it’s own io ch

SW solutions: Multi step merge.

INDEXING:

**Operations Required to Maintain the Indexed Files: Create** the original empty index and data files

**Load** the index file into memory before using the index file **Rewrite** the index file from memory after using the index file **Add** data records to the data file **Delete** records from the data file **Update records** in the data file, and **Update the index** to reflect changes in the data file.

**Record Updating**: 2Possibilities 1. The update changes the value of the key field 2. The update does not affect the key field

HASHING:

Strats for reducing collisions: **1.** Spread out records **2.** Use extra memory **3.** Put more than one rec in a single address (buckets).

Simple hash algorithm: 1. Rep the key in numerical form 2. Fold and add (divide string on ASCII) 3. Divide by the size of the address space.

Make It Better:

Examine for patterns, fold parts of the key, divide the key by a number, square the key and take the middle, radix transformation.

**Poisson Distribution** can be used to analyze a random distribution.

A – the key does not hast to the given address, so P(A)| B – the key hashes to the given address, so P(B) is prob that addr is chosen.

P(B) = 1/N, P(A) = 1 – 1/N

Better choice:

N = available addresses, R = # recs to be stored, x = the number of recs assigned to a given address.

P(x) =

N×[1×p(2)+2×p(3)+3×p(4)+4×p(5)+5×p(6)]=3026 (#overflow recs expected.)

For buckets, Add to n, where p(n), by bucket size.

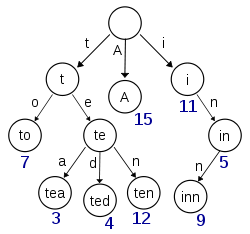
**Searching a hash** remember cases: 1. find the record: return found information 2. come to the end of the file: start looking at the beginning again 3. come to an empty record: return record not present 4. come to the original address: return record not present

\*If there are a lot of deletions and insertions, the hashed file will reach equilibrium, but search WILL deteriorate.

**OtherWays to handle overflow**: 1. Chained progressive overflow ->use pointers to link overflow recs together. 2. Chaining with separate overflow area - > Move all overflow recs to separate area (Must do this will grow beyone address space size)

Extendible hashing: re-hashing is an incremental operation (done one bucket at a time, as needed)

**Trie**: Or Prefix Tree. EXwKeys: "A", "to", "tea", "ted", "ten", "i", "in", and "inn":



ISAM(**Indexed Sequential Access Method**) ISAM files have ahierarchy of indexes reflecting the hierarchy of structures within a disk. The lowest level is the track index; above that is the cylinder index; finally, there is the master index. The number of levels is fixed.

**VSAM**: VSAM uses abstract data structures [control areas and control intervals] rather than direct mapping onto hardware elements. When the file is initially created. space is left for future insertions.

**Normalization:** A normal form (NF) is a set of criteria that must be met by a set of record types.

[1NF), all fields must be atomic (i.e., cannot be decomposed into simpler elements. [2NF], the data must be in INF and, in addition, all non-key fields must be dependent on the whole of the primary key. [3NF), the data must be in 2NF and, in addition, there must be no dependencies between nonkey fields. [5NF](catch all) the data cannot be decomposable into simpler record types without loss of information.

**Choosing a solution:**

|  |  |  |
| --- | --- | --- |
| Organization | Single Record by primary key | All Records in Pri Key Order |
| None | Slow | Very Slow |
| Sequential | Slow | Fast |
| Hashed | Fast | Very Slow |
| Indexed Seq | Moderate | Fast |

|  |  |
| --- | --- |
| Organization | Overhead |
| B-tree | Pointers to nodes. Node utilization ~ 60% |
| Grid File | Unused space in Bucks |
| Dynamic Hashing | Indexed Trees util~69% |
| Table assisted hash | Table. Unused bucks.. |
| Fixed hash file | Unused Bucks,Pointers. |

**Compute a Methodology:**

1. Consider application. 2. Identify candidate file structures, give score 3. Compute weighted score