CMRIT
CMR INSTITUTE OF TECHNOLOGY, BENGALURU.
ACCREDITED WITH A+ GRADE BY NAAC
No.132, AECS Layout
I.T. Park Road, Bangalore 560 037
T: +91 80 28524466/77, F : +91 80 28524630
E: info@cmr.ac.in | www.cmr.ac.in

DEPARTMENT OF INFORMATION SCIENCE & ENGINEERING

FS MODULE 2

Prepared by: Prof. Prasad B S

Organization of files for performance and Indexing

1. **What is redundancy reduction? Explain how run- length- encoding helps in redundancy reduction with an example**
2. **What is Data compression? Explain any two data compression algorithm with example.**
3. **Write an algorithm for searching a record from a file using**
   a. **binary search**
   b. **sequential search**

   Explain the limitations of binary searching and internal sorting.
4. **Explain how spaces can be reclaimed in files.**
   Or
5. **How spaces can be reclaimed from deletion of records from fixed length record file and variable length record file?**
6. **Explain the advantages and disadvantages of 3 types of placement strategies.**
7. **What is an index? Explain a simple index for entry-sequenced file.**
8. **Write a C++ program to simple index on primary key for a file of student objects. Implement add(), search() functions using the index.**
9. **Explain the operations required to maintain the index files.**
10. **Explain the key sorting algorithm, with an example. What are its limitations?**
11. **Discuss the advantages and disadvantages of indices that are too large to hold in memory.**
12. **What is secondary indexing? What are the limitations of secondary indexing? Explain the solution by using 'linking the reference' techniques.**
13. **How do you improve secondary index structure using inverted lists.**
14. **Write a note on selective indexing**

Answers:

1. **What is redundancy reduction? Explain how run- length- encoding helps in redundancy reduction with an example**
2. **What is Data compression? Explain any two data compression algorithm with example.**

Ans: (For both 1 and 2)

Data compression: The encoding of data in such a way as to reduce its size on the disk.

Redundancy reduction: Any form of compression which removes only redundant information in the file making the file smaller. Data compression is possible because most data contains redundant (repeated) or unnecessary information. Reversible compression removes only redundant information, making it possible to restore the data to its original form. Irreversible compression goes further, removing information which is not actually necessary, making it impossible to recover the original form of the data.

Compression methods:

1. Compact Notation

The replacement of field values with an ordinal number which index an enumeration of possible field values. Compact notation can be used for fields which have an effectively fixed range of values. Compact notation can be used for fields which have an effectively fixed range of values.

The State field of the Person record, is an example of such a field. There are 676 (26 x 26) possible two letter abbreviations, but there are only 36 states (including 7 union territories). By assigning an ordinal number to each state, and storing the code as a one byte binary number, the field size is reduced by more than 50 percent.

Ex: Karnataka -> KA -> 01001

Advantage:

- No information has been lost in the process.
- The compression can be completely reversed, replacing the numeric code with the two letter abbreviation when the file is read.
- Compact notation is an example of redundancy reduction.

Dis-Advantage:

- On the other hand, programs which access the compressed data will need additional code to compress and expand the data.
- An array can used as a translation table to convert between the numeric codes and the letter abbreviations. The translation table can be coded within the program, using literal constants, or stored in a file which is read into the array by the program.
- Since a file using compact notation contains binary data, it cannot be viewed with a text editor, or typed to the screen. The use of delimited records is prohibitively expensive, since the delimiter will occur in the compacted field.

2. Run Length Encoding

An encoding scheme which replaces runs of a single symbol with the symbol and a repetition factor.

- Run-length encoding is useful only when the text contains long runs of a single value.
- Run-length encoding is useful for images which contain solid color areas.
- Run-length encoding may be useful for text which contains strings of blanks.

Example:

Uncompressed text (hexadecimal format):

40 40 40 40 40 40 43 43 41 41 41 41 41 42

Compressed text (hexadecimal format):

FE 06 40 43 43 FE 05 41 42

where FE is the compression escape code, followed by a length byte, and the byte to be repeated.

3. Variable Length Codes

An encoding scheme in which the codes for differenct symbols may be of different length. **Huffman code:** A variable length code, in which each code is determined by the occurence

frequency of the corresponding symbol. Huffman codes are based on the frequency of occurrence of characters in the text being encoded. The characters with the highest occurence frequency are assigned the shortest codes, minimizing the average encoded length. Huffman encoding takes two passes through the source text: one to build the code, and a second to encode the text by applying the code. The code must be stored with the compressed text.

4. Irreversible Compression Techniques

Irreversible compression goes beyond redundancy reduction, removing information which is not actually necessary, making it impossible to recover the original form of the data. Irreversible compression is useful for reducing the size of graphic images. Irreversible compression is used to reduce the bandwidth of audio for digital recording and telecommunications. JPEG image files use an irreversible compression based on cosine transforms. The amount of information removed by JPEG compression is controllable. • The more information removed, the smaller the file. For photographic images, a significant amount of information can be removed without noticably affecting the image.

3. Write an algorithm for searching a record from a file using

1. **binary search**
2. **sequential search**

Explain the limitations of binary searching and internal sorting.

Ans:

Binary Search Algorithm:

```
int BinarySearch
   (FixedRecordFile & file, RecType & obj, KeyType & key)
// binary search for key
// if key found, obj contains corresponding record, 1 returned
// if key not found, 0 returned
{
   int low = 0; int high = file.NumRecs()-1;
   while (low <= high)
   {
      int guess = (high - low) / 2;
      file.ReadByRRN (obj, guess);
      if (obj.Key() == key) return 1; // record found
      if (obj.Key() < key) high = guess - 1;// search before guess
      else low = guess + 1;// search after guess
   }
   return 0; // loop ended without finding key
}
```

Figure 6.13 A binary search algorithm.

Sequential Search:

```
Int SequentialSearch
{
int low=0, high= file.NumRecs()-1;
for ( int i=low;i<high;i++)
        {
        file.ReadByRRN(obj,i);
        if (obj.Key==key) return 1;
        }
return -1;
```

}

- A binary search of n items requires $\log_2 n + 1$ comparisons at most.
- A binary search of n items requires $\log_2 n + 1/2$ comparisons on average.
- Binary searching is $O(\log_2 n)$.
- Binary searching is only possible on ordered lists.
- Sequential search of n items requires n comparisons at most.
- A successful sequential search of n items requires n / 2 comparisons on average.
- An unsuccessful sequential search of n items requires n comparisons.
- Sequential searching is O(n).

4. **Explain how spaces can be reclaimed in files.**
   Or
5. **How spaces can be reclaimed from deletion of records from fixed length record file and (or) variable length record file?**

Ans:

As files are maintained, records are added, updated, and deleted. As records are deleted from a file, they are replaced by unused spaces within the file. The updating of variable length records can also produce fragmentation.

Issues on reclaiming space quickly:

– How to know immediately if there are empty slots in the file?

– How to jump to one of those slots, if they exist?

It is done by creating by linking all deleted records together using a linked list known as Avail List.

Fixed-Length Records

Deleted records are marked by special character or sequence of characters at the beginning of the deleted record position.

Record 1	Record 2	Record 3	Record 4	Record 5

Record 3 deleted results in

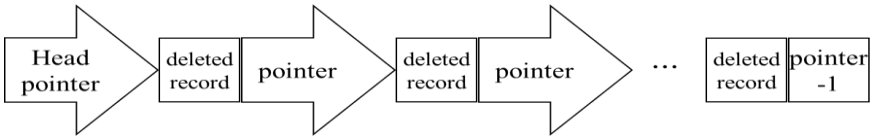
Record 1	Record 2	*	Record 4	Record 5

To reuse the empty space, there must be a mechanism for finding it quickly.

- One way of managing the empty space within a file is to organize as a linked list, known as the avail list.
- The location of the first space on the avail list, the head pointer of the linked list, is placed in the header record of the file.
- Each empty space contains the location of the next space on the avail list, except for the last space on the list.
- The last space contains a number which is not valid as a file location, such as -1.

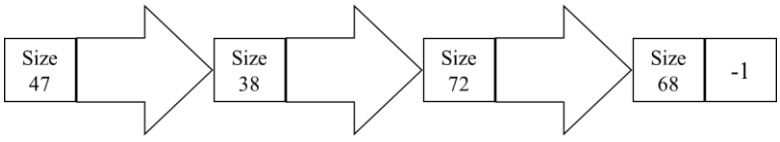
Header	Slot 1	Slot 2	Slot 3	Slot 4	Slot 5	Slot 6
3	* -1	Record 2	* 1	Record 4	Record 5	Record 6

- If the file uses fixed length records, the spaces are interchangeable; any unused space can be used for any new record. The simplest way of managing the avail list is as a stack. As each record is deleted, the old list head pointer is moved from the header record to the deleted record space, and the location of the deleted record space is placed in the header record as the new avail list head pointer, pushing the new space onto the stack.



Variable-Length Records

If the file uses variable length records, the spaces not are interchangeable; a new record may not fit just in any unused space. With variable length records, the byte offset of each record can be used as location pointers in the avail list. The size of each deleted record space should also be placed in the space.



A policy for determining the location of a new record in a file.

First fit: A placement strategy which selects the first space on the free list which is large enough.

Best fit: A placement strategy which selects the smallest space from the free list which is large enough.

Worst fit: A placement strategy which selects the largest space from the free list (if it is large enough.)

6. Explain the advantages and disadvantages of 3 types of placement strategies.

Ans:

A policy for determining the location of a new record in a file with variable length records and deleted slots available in avail list.

First fit: A placement strategy which selects the first space on the free list which is large enough. The simplest placement strategy is first fit. With first fit, the spaces on the avail list are scanned in their logical order on the avail list. The first space on the list which is big enough

for a new record to be added is the one used. The used space is delinked from the avail list, or, if the new record leaves unused space, the new (smaller) space replaces the old space.

Best fit: A placement strategy which selects the smallest space from the free list which is large enough. The best fit strategy leaves the smallest space left over when the new record is added. There are two possible algorithms:

1. Manage deletions by adding the new record space to the head of the list, and scan the entire list for record additions.
2. Manage the avail list as a sorted list; the first fit on the list will then be the best fit.

Worst fit: A placement strategy which selects the largest space from the free list (if it is large enough. The worst fit strategy leaves the largest space left over when the new record is added. The rational is that the leftover space is most likely to be usable for another new record addition. There are two possible algorithms:

1. Manage deletions by adding the new record space to the head of the list, and scan the entire list for record additions.
2. Manage the avail list as a reverse sorted list; the first fit on the list, which will be the first entry, will then be the worst fit.

7. What is an index? Explain a simple index for entry-sequenced file.

Ans:

Indexing is a structure containing a set of entries, each consisting of a key field and a reference field, which is used to locate records in a data file. It helps in faster access of records in a file if the size of the index file is small. Since index files are sorted on key field, binary search can be applied to find the presence of the key and use the reference field for performing a direct access to locate the record in single seek.

Simple index for entry sequenced file

A simple index for entry sequenced file consists of two files

1. An index file consist of key-reference pairs which are sorted by key.
2. A data file which is entry sequenced and hence not sorted.

A class for implementing index is as below :

```
class TextIndex
{public:
   TextIndex (int maxKeys = 100, int unique = 1);
   int Insert (const char * key, int recAddr); // add to index
   int Remove (const char * key); // remove key from index
   int Search (const char * key) const;
      // search for key, return recaddr
   void Print (ostream &) const;
protected:
   int MaxKeys; // maximum number of entries
   int NumKeys; // actual number of entries
   char * * Keys; // array of key values
   int * RecAddrs; // array of record references
   int Find (const char * key) const;
   int Init (int maxKeys, int unique);
   int Unique; // if true, each key must be unique in the index
};
```

Ex:

Index			Recording file
Key	*Reference field*	*Address of record*	*Actual data record*
ANG3795	152	17	LON | 2312 | Romeo and Juliet | Prokofiev | ...
COL31809	338	62	RCA | 2626| Quartet in C Sharp Minor | Beethoven | ...
COL38358	196	117	WAR | 23699 | Touchstone | Corea | ...
DG139201	382	152	ANG | 3795 | Symphony No. 9 | Beethoven | ...
DG18807	241	196	COL | 38358 | Nebraska | Springsteen | ...
FF245	427	241	DG | 18807 | Symphony No. 9 | Beethoven | ...
LON2312	17	285	MER | 75016 | Coq d'Or Suite | Rimsky-Korsakov | ...
MER75016	285	338	COL | 31809 | Symphony No. 9 | Dvorak | ...
RCA2626	62	382	DG | 139201 | Violin Concerto | Beethoven | ...
WAR23699	117	427	FF | 245 | Good News |Sweet Honey in the Rock | ...

8. Write a C++ program to simple index on primary key for a file of student objects. Implement add(), search() functions using the index.

Lab Program no 5 –simple indexing (write pack and search function) and explain

9. Explain the operations required to maintain the index files.

Ans:

Operations required to maintain an indexed file:

1) Creating the data and index files.
2) Loading the index file to memory
3) Rewriting the index file from memory
4) Record addition
5) Record deletion
6) Record updating

1. Creating the Data and Index file: Two file must be created: Data file to hold the data objects and Index file to hold the primary key index. Both the files are created as empty files. The index file is of fixed size records.
2. Loading the Index file to memory: Before any manipulation of the file, index file is loaded on to the memory to ensure that changes due to manipulation of data file is reflected in the index file also.
3. Rewriting the Index file from memory: Once the changes in the data file is done and the same is reflected on the index file which is on memory. Index file must be written back to the disk else index file on disk will not be in sync with the data file,

4. Record addition: This consists of appending the data file with a new record and inserting a new entry in index file. The rearrangement of index consists of sliding down the records with the key larger than the inserted key and then placing the new record in the opened space. This is done in main memory.
5. Record deletion : This uses the technique of reclaiming space in files, Index entry of the deleted record is removed from the index file and all records key with larger than the deleted records are shift-up to its original position.
6. Record Updating : This consists of two cases:
    i. Update changes the value of the key : In this case it is treated as a deletion operation followed by insertion.
    ii. Update does not affect the key: If record size is unchanged, just modify data record and index file is unchanged but if the record size is changed then it is also treated as deletion/insertion operation.

10. Explain the key sorting algorithm, with an example. What are its limitations?

Ans:

Keysorting: A sort performed by first sorting keys and then moving records.

- Read each record sequentially into memory, one by one.
- Save the key of the record and the location of the record, in an array (KEYNODES)
- After all records have been read, internally sort the KEYNODES array of record keys and locations
- Using the KEYNODES array, read each record back into memory a second time using direct access.
- Write each record sequentially into a sorted file.

```
int KeySort (FixedRecordFile & inFile, char * outFileName)
{
  RecType obj;
  KeyRRN * KEYNODES = new KeyRRN [inFile . NumRecs()];
  // read file and load Keys
  for (int i = 0; i < inFile . NumRecs(); i++)
  {
    inFile . ReadByRRN (obj, i);// read record i
    KEYNODES[i] = KeyRRN(obj.Key(),i);//put key and RRN into Keys
  }
  Sort (KEYNODES, inFile . NumRecs());// sort Keys
  FixedRecordFile outFile;// file to hold records in key order
  outFile . Create (outFileName);// create a new file
  // write new file in key order
  for (int j = 0; j < inFile . NumRecs(); j++)
  {
    inFile . ReadByRRN (obj, KEYNODES[j].RRN);//read in key order
    outFile . Append (obj);// write in key order
  }
  return 1;
}
```

Limitations:

- Only possible when the KEYNODES array is small enough to be held in memory.
- Each record must be read twice: Once sequentially and once directly.
- Each direct access requires a seek.

- Key sorting is a way to sort medium sized files.

11. Discuss the advantages and disadvantages of indices that are too large to hold in memory.

Ans:

If the index file is too large to fit in main memory , it can lead to following problems:

1. Binary searching the index file requires more than one seek call
2. Index rearrangement on record insertion, deletion or updating requires index shifting on disk

This limitation requires to provide solution such as tree structured index file and hashing.

12. What is secondary indexing? What are the limitations of secondary indexing? Explain the solution by using 'linking the reference' techniques.

13. How do you improve secondary index structure using inverted lists.

Ans:

Secondary Indexing: In secondary indexing the secondary key is related to a primary key which then will point to the actual byte offset.

- When a secondary index is used, adding a record involves updating the data file, the primary index and the secondary index.
- The secondary index update is similar to the primary index update. Secondary keys are entered in canonical form (all capitals).
- The upper- and lower- case form must be obtained from the data file. As well, because of the length restriction on keys, secondary keys may sometimes be truncated.
- The secondary index may contain duplicate (the primary index couldn't).

Ex : Secondary index file of all students with name as secondary key

- Secondary indexes lead to two difficulties:
  - The index file has to be rearranged every time a new record is added to the file.
  - If there are duplicate secondary keys, the secondary key field is repeated for each entry ==> Space is wasted.
- Solution 1: Change the secondary index structure so it associates an array of reference with each secondary key.

Revised composer index

Secondary key	*Set of primary key references*			
BEETHOVEN	ANG3795	DG139201	DG18807	RCA2626
COREA	WAR23699			
DVORAK	COL31809			
PROKOFIEV	LON2312			
RIMSKY-KORSAKOV	MER75016			
SPRINGSTEEN	COL38358			
SWEET HONEY IN THE R	FF245			

Figure 7.11 Secondary key index containing space for multiple references for each secondary key.

- Advantage: helps avoid the need to rearrange the secondary index file too often.
- Disadvantages:
  - » It may restrict the number of references that can be associated with each secondary key.
  - » It may cause internal fragmentation, i.e., waste of space.

- Solution 2: each secondary key points to a different list of primary key references. Each of these lists could grow to be as long as it needs to be and no space would be lost to internal fragmentation.

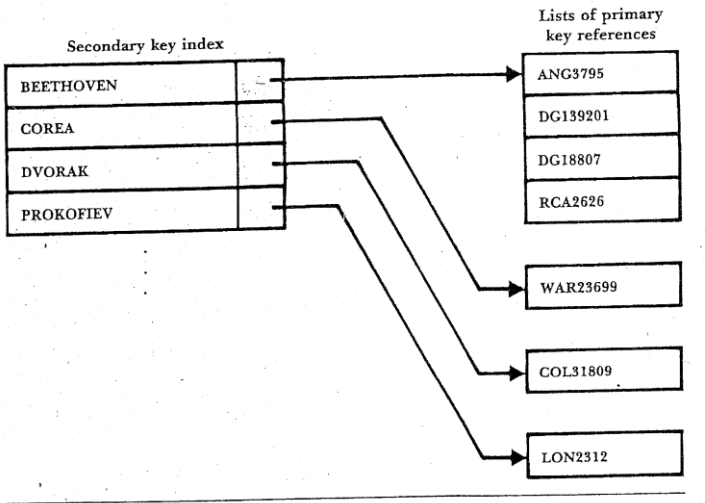


Figure 7.12 Conceptual view of the primary key reference fields as a series of lists.

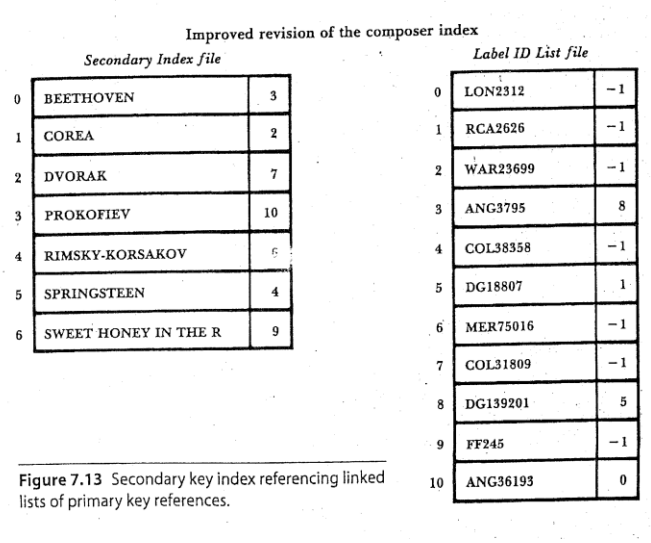
Improved version of inverted list

14. Write a note on selective indexing

Ans:

Using selective indexing , on can divide the file into parts and provide a selective view.

- For example, you can build a selective index that contains only titles to classical recordings or recordings released prior to 1970, and since 1970. A possible query could then be: “List all the recordings of Beethoven’s Symphony no. 9 released since 1970.”



Improved revision of the composer index

Secondary Index file

0	BEETHOVEN	3
1	COREA	2
2	DVORAK	7
3	PROKOFIEV	10
4	RIMSKY-KORSAKOV	6
5	SPRINGSTEEN	4
6	SWEET HONEY IN THE R	9

Label ID List file

0	LON2312	−1
1	RCA2626	−1
2	WAR23699	−1
3	ANG3795	8
4	COL38358	−1
5	DG18807	1
6	MER75016	−1
7	COL31809	−1
8	DG139201	5
9	FF245	−1
10	ANG36193	0

Figure 7.13 Secondary key index referencing linked lists of primary key references.

  - Advantages:
    - » The secondary index file needs to be rearranged only when a record with a new secondary key is added. It is not that costly keeping the secondary index on disk.
    - » The rearranging is faster since there are fewer records in the secondary index.
    - » The primary index is entry sequences -- never needs to be sorted.
    - » Space from deleted primary index records can easily be reused.
  - Disadvantage:
    - » Locality (in the secondary index) has been lost ==> More seeking may be necessary.