Chapter 1:

Introduction to the Design and Specification of File Structure

Why is File Structure Design Necessary?

Tip: Data Access: data를 읽거나 쓸때를 일컫는다.

- Disks are very slow compared with RAM
- How slow is a disk?

목적: Cache는 레지스터와 메인메모리 사이의 속도차를 줄여주기 위해 사용
파일처리는 메인과 세컨더리사이의 속도차를 줄여줌

$$\text{RAM} : \text{Disk} = 120 \text{ nanosec} : 30 \text{ milisec}$$

$$= 1 \text{ sec} : \frac{30\times10^{-3}}{120\times10^{-9}}\text{sec} = 1\text{sec} : 25\times10^{4}\text{ sec}$$

$$= 1 \text{ sec} : 2 \text{ days and 22 hours}$$

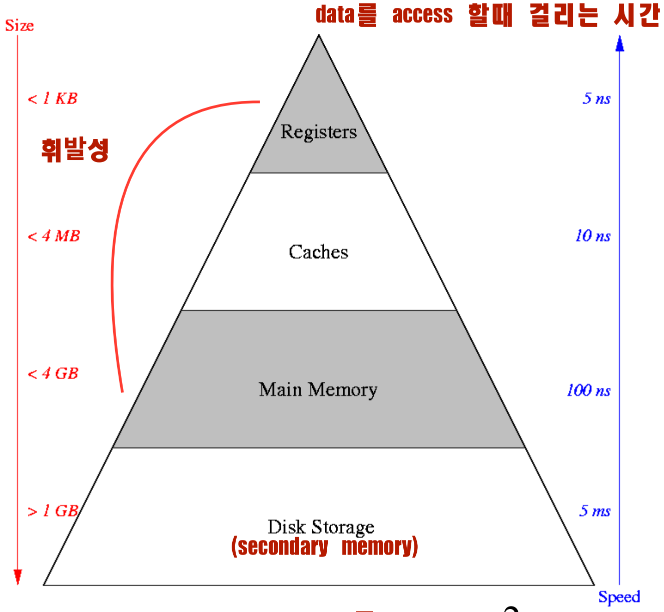
- However, disks provide enormous and

  nonvolatile capacity at much less

  cost than RAM

Performance를 측정할때
여러기준에서 볼 수 있어야함
1. Main memory사용량
2.Elapsed time & User response time
3. Network
4. Throughput(단위시간당 job처리량)
5. etc..
But 파일처리에서 가장 중요한 부분은
"2번"이다.

data를 access 할때 걸리는 시간



+ Tape, CD등
(tertiary memory)

What is a Good File Structure Design?

- What is a file?

세컨더리 메모리에 저장되는 같은종류의
레코드의 집합

  - A set of same kind of records which are stored into secondary memory like hard disk, solid state disk, CD, tape, etc.

- What is a file structure?

c에서 구조체,
cpp에서 클래스처럼
"표현"하는 것

insert,delete,seratch,read,write등

  - A combination of representations for data in files and of operations for accessing the data

- Goal of good file structure design
  - Allowing us to get the information with as small cost as possible
  - Main performance factor: number of disk accesses

디스크 엑세스의 비용이 크니까
최대한 접근하지 않으면서 정확한 구조가
좋은 파일구조임

A Short History of File Structure Design (1)

- Sequential access 순차적으로 읽는 것(=:linear search)
  - Accessing records in order, looking at the first, then the next, and so on
  - Most files were on tape in early work with files

Hard disk의 경우 앞과 다르게 하나의 data를 찾기위해
전체를 보지 않아도 됨
>일부를 읽으면서도 그사이 공간을 읽을 필요가 없음
>>random access

```
struct {
        char   name[20];
        char   telephone[20];
        char   address[60];
} phone_book;
```

Gildong Hong | +82-2-820-0914 | 1-1 Sangdo-dong, Dongjak-gu, Seoul 156-743, Korea

Paul B. Kantor | +1-732-932-1359 | New Brunswick, NJ 08901-1071, USA

Yasushi Ogawa | +81-45-477-1569 | 3-2-3, shin-yokohama, Kohoku-ku, Yokohama-shi 222, Japan

Hans-Peter Frei | +41-1-236-5714 | Ch-8021 Zurich, Bahnhofstrasse 45, Switzerland

⋮

A Short History of File Structure Design (2)

- Simple index
  - Storage devices like disk drives became available
  - Keeping a list of keys and pointers in a smaller file
  - Difficult to manage, especially for dynamic files

index는 각 record의 key, 위치를 저장
+ key값에 대해 sorting되어있음

Index File

Gildong Hong | 1
Hans-Peter Frei | 301
Paul B. Kantor | 101
Yasushi Ogawa | 201
⋮

Data File

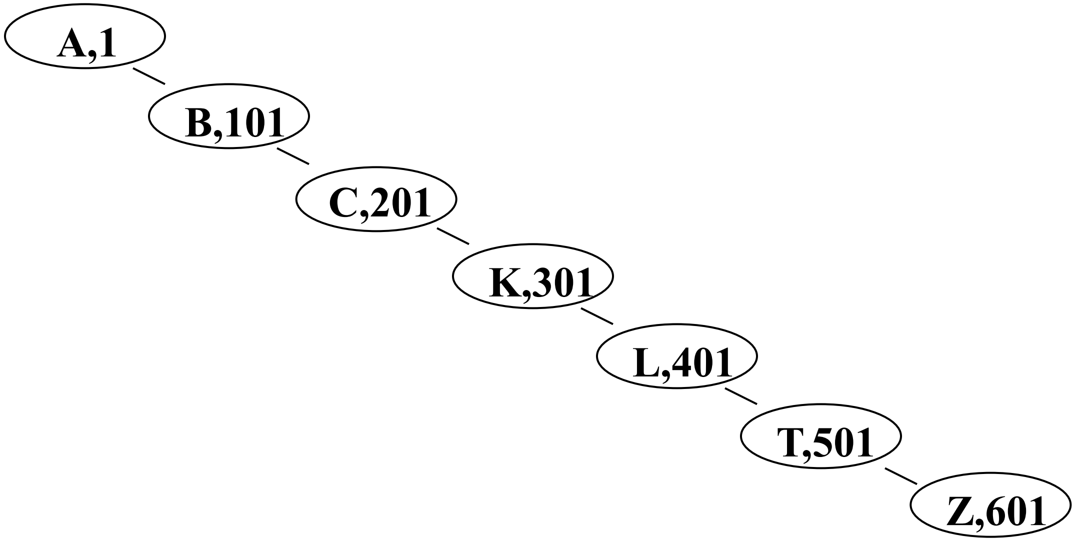
1 Gildong Hong | +82-2-820-0914 | 1-1 Sangdo- ……
101 Paul B. Kantor | +1-732-932-1359 | New Br ……
201 Yasushi Ogawa | +81-45-477-1569 | 3-2-3, s ……
301 Hans-Peter Frei | +41-1-236-5714 | Ch-8021 ……
⋮

A Short History of File Structure Design (3)

나보다 작은건 왼쪽서브트리
크거나 같은건 오른쪽서브트리

search

- Binary tree : early 1960s
  - binary trees can grow very unevenly as records are added and deleted
  - ex) insert (A,1), (B,101), (C,201), (K,301), (L,401), (T,501), (Z,601)

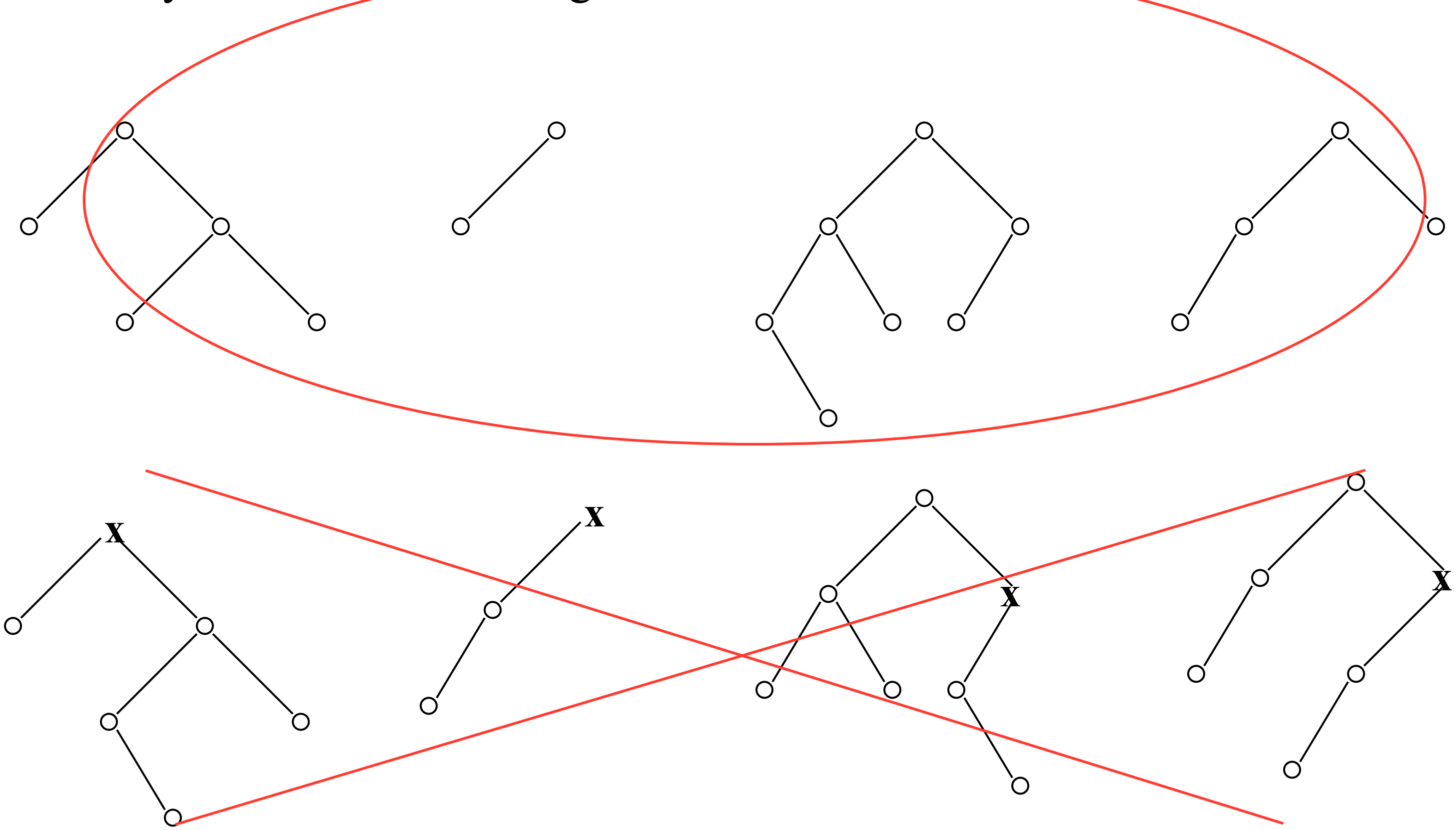


A Short History of File Structure Design (4)

- AVL tree

  왼쪽서브트리와 오른쪽서브트리의
  깊이차가 1이하라는 제한조건을 둠

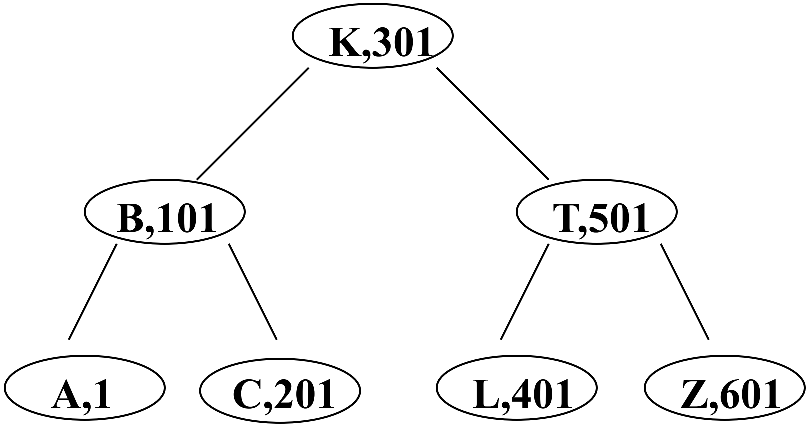
  – there is a limit on the amount of difference that is allowed between the heights of any two subtrees sharing a common root



A Short History of File Structure Design (5)

complete: 왼쪽부터 채워나감
full: 자식이 있으면 2개 여야함
balanced: 모든 leaf의 깊이차이가 1이하여야함

- Balanced binary tree
  - Given *N* keys, looks at $\lfloor \log_2 N \rfloor + 1$ levels of the tree
  - ex) insert (A,1), (B,101), (C,201), (K,301), (L,401), (T,501), (Z,601)



A Short History of File Structure Design (6)

Binary tree의
자식노드의 개수가 너무 적다고 생각하여
〉〉자식의 수를 증가시키면
자연스럽게 높이가 줄어들게 된다
〉〉이를 적용한 것이 b-tree

이미지 검색 조사 해 보기

- B-tree
  - A tree structure that provides fast access to data stored in files
  - Unlike binary trees, in which the branching factor from a node of the tree is two, the descendents from a node of a B-tree can be a much larger number
- B+ tree
  - A variation on the B-tree structure that provides sequential access to the data as well as fast-indexed access
- Hashing
  - An access mechanism that transforms the search key into a storage address, there by providing very fast access to stored data
- Extendible hashing
  - Hashing does not work well with dynamic files
  - Dynamic hashing that could retrieve information with one or, at most, two disk a ccesses no matter how big the file becomes

C++ Language: Using Objects(1)

```
class Person {
private:
        char name[20];
        int age;
        char address[50];
public:
        Person(char *pname, int page, char *paddress);
        ~Person();
        char* getName();
        int getAge();
        char* getAddress();
        void setName(char *personname);
        void setAge(int personage);
        void setAddress(char *personaddress);
};
```

C++ Language: Using Objects(2)

```
class Student: public Person {
private:
        int year;
        char major[30];
public:
        Student(int syear, char *smajor);
        ~Student();
        char* getMajor();
        int getYear();
        void setMajor(char *smajor);
        void setYear(int syear);
};
```