박정진

2015104027

2019.06.09

자료구조 프로젝트2 결과보고서

1. 목적

현재 우리 일상 속 컴퓨터의 운영체제 안에 구현 되어있는 폴더 관리 시스템을 비슷하게 구현하고 그 와 같은 구조로 후에 있을 음원 및 파일 관리도 할 수 있게 끔 하는 것을 계획한 프로그램을 구상한다. 폴더, 파일, 음원 등 이런 아이템들을 효율적으로 검색하고 다룰 수 있는 자료구조를 이용하여 나만의 스타일을 갖춘 폴더 관리 시스템을 만든다.

2. 기능

(macOs운영체제 이므로 macOs 기준에 맞춰서 구현한다)

폴더와 파일은 매우 유사한 구조 및 정보를 가지고 동작한다. 하지만 그 둘의 차이점이 몇가지가 있다.

1. 폴더만이 서브 아이템을 가진다.

2. 폴더 실행은 change directory이고 파일 실행은 확장자에 맞는 파일 어플리케이션을 실행하는 것이다.

기능 구현

1. Interface는 Python 라이브러리 Tkinter 로 리모콘 형식으로 구성

2. 폴더, 파일 생성 (중복 이름 자동 \_숫자 로 처리)

3. 폴더, 파일 삭제 및 이름 수정

4. 폴더, 파일 정보 관리 및 출력

5. 폴더 내 서브 폴더 및 파일 들 리스트 및 상세 정보 출력

6. Change Directory ( PathProcessor 함수를 통한 Path 접근 허용)

7. History를 이용한 앞으로가기, 뒤로가기

8. Recently 항목을 통한 최근 Access 한 폴더 및 파일 정리

9. Heap 구조를 통한 cache 멤버 변수 생성, Rectly Item을 이름순으로 정렬

9. 서브 폴더 및 파일 이름 순, 생성 날짜 순, Access time 순으로 내림차순, 오름차순 정렬

10. Cut, Copy, Duplicate, Paste 기능 구현

11. TextApplication, JPGApplication 기능을 통한 확장자에 따른 파일 실행

3. 기능에 대한 설명

1. 폴더 와 파일 은 아이템이고, 공통점이 많은 점을 이용해 공통 기능은 ItemType에 구현하고 FolderType과 FileType은 ItemType을 상속받아 고유의 속성 및 함수를 구현함. 이외 FileType을 상속받은 TextFIleType, JPGFileType, MP3FileType구현

2. 모든 동작이 현재 working directory 기준이다. Working directory 안에서 다양한 기능을 제공하는 BaseApplication에서 폴더를 관리(BaseApplication의 현재 폴더를 가리키는 FolderType\* 포인터로 관리) 하는 구조로 구현함

3. interface Python Tkinter로 구성,리모콘이 보내는 인자들을 String [Operation] [Option / Target] [Value] 형식으로 명령에 대한 처리

4. 폴더, 파일 의 정렬은 이름으로 정렬한 것을 기본으로 하고, 삽입, 삭제, 검색 연산 시 시간 복잡도 O(logn)을 보장하는 AVL Tree로 구성

5. AVL Tree 는 Binary Search Tree를 상속 받고 Binary Search Tree 는 Binary Tree를 상속 받은 형태로 구성

6. 다양한 타입을 담을 수 있는 Heterogeneous 형식으로 구성

7. Sort는 Comparer 클래스를 통해 오름차순, 내림차순 정렬, Name, CreatedTime, ModifiedTime, AccessTime 기준 별로 정렬이 가능하다.

8. Cut, paste, copy는 ItemType\* heap이라는 BaseApplication 의 멤버 변수로

지정을 하여 polymorphism으로 구현한다.(폴더, 파일 모두 사용해야 한다)

4. 사용한 자료 구조

1. AVL Tree<ItemType\*>(Heterogenous)

* AVL TREE는 메모리를 효율적으로 사용할 수 있는 Linked List의 장점과

정렬된 리스트 내 Index로 빠르게 binary search 의 장점을 결합한 구조

* Binary Search Tree는 검색 속도를 보장하지 못하지만 AVL Tree는 Balance Factor가 무너질 때 Re –balancing을 통해 O(1.44logN)을 보장할 수 있다.
* 폴더 관리 특성 상 유저가 얼마나 많은 정보를 생성할지 모르기 때문에 메모리 효율적 관리 중요
* 어떤 특정한 아이템을 생성 , 삭제, 검색 연산이 많은 폴더 관리 시스템에서

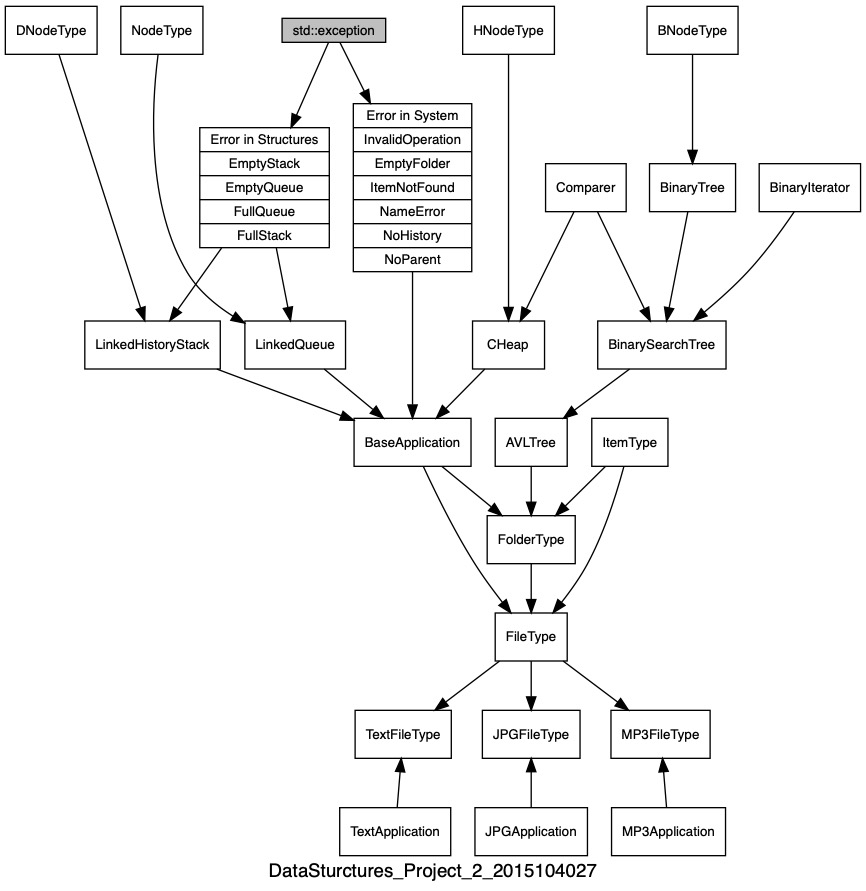
해당 연산들의 효율성이 좋아야 하는 것이 중요

* 이 두 가지 특성을 고려했을 때 AVL Tree가 가장 적합하다고 판단

2. Linked History Stack, Linked Queue

* History는 가장 최근 것이 먼저 나와야 한다는 점에서 Stack 구조가 적합
* Recent Item과 Path Processor는 가장 처음에 들어간 것을 가장 먼저 접근 해야한다는 점에서 적합
* Stack과 Queue 둘다 Linked 형태로 구현했는데 이 또한 List에서의 이유 처럼 유저가 얼마나 많은 연산을 할지 모르기 때문에 메모리를 효율적으로 사용할 수 있는 Linked 스타일의 구조가 적합하다고 판단

5. Class ADT



6. 한계 및 보완점

1. Python 언어가 C++ 과는 다른 개념으로 메모리를 할당하고 인자를 pass하므로

C++에 있는 기능을 완벽히 구성하지 못함

Ex) Python 은 operator = 연산자 오버로딩이 불가능

Python은 Copy constructor가 없고 대신 기본라이브러리 copy 모듈의 deepcopy 메서드 사용 권장

함수에 인자를 pass 할 때 call by value, call by reference를 정하지 못하고 , Call by Assignment라는 방식 사용

2. 실제 디렉토리 및 파일이 아니어서 실제와 연관되지 않아 매번 프로그램을 실행할 때 기록들이 사라지고, 불러오기가 안됨.

3. Binary Iterator를 사용해 모든 노드를 한번씩 방문하는데 Iterator 생성 시 노드 개수 만큼의 리스트를 생성해 접근하므로 메모리 낭비가 있음

4. Heap을 이용한 검색이 다양한 곳에 사용되지 못함

(전체 폴더에서 keyword 검색할 때만 사용)

5. Project\_1 에서 부족하고 추가로 구현해야 된다고 생각하는 기능들을 구현하지 못하고,

Project\_1 때 봤던 다른 좋은 기능, 코드들을 사용하지 못함