



PlayList of Artwork

Data structure design – Personal Project

20164204

컴퓨터공학과

이상민



- 1. Goal of Program
- 2. Dynamic Graph – What & How
- 3. Data Collection
- 4. Implement
- 5. about Code
- 6. Play



Goal of Program

- Playlist : Display 하나로도 만들 수 있는 작은 전시회
- Dynamic Graph 기반의 예술작품 PlayList 구현
 - Dynamic Graph 그래프가 갖는 성질을 활용
 - 노드의 생성과 소멸에 따라 Edge 가 Dynamic 하게
 - 변형되는 그래프
 - 실용성과 효율성에 중점을 둔 프로그램 구현
 - 쉽게 접할 수 있는 PlayList 를 예술작품에 접목
 - 플레이리스트로써 필요한 기능 구현



Dynamic Graph in PlayList

- 재생 목록 = Dynamic Graph
 - 재생목록의 군집
 - (구현부) 동일한 Category (Information) 를 갖는 작품들이 연결된 그래프
 - (추가 , 삭제) User Add // User Delete 에 따라 Dynamic 한 그래프

Node :

Artwork

Edge :

Artwork Information

(작가, 연도, 장르, 동/서양)

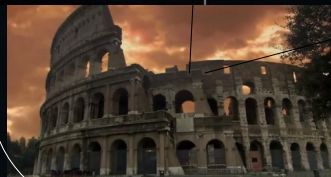


군집화 ? - 구현부

Edge : 고희



Edge : 서양



Edge : 동양



Edge : 건축물

○ . 작품 정보중 한 항목이 동일 -> 해당 Edge 로 연결 // 마치 군집



군집화 ? - 추가 , 삭제

- 추가

- Artwork – Information
- Information
 - 1. 작가 2. 연도 3. 장르 4. 동양 / 서 양
- Information 을 기준으로
- 기존의 그래프에서 동일한 Information 을 갖는
- 작품들과 Edge 연결

- 삭제

- Artwork
- 해당 Artwork 삭제
- 해당 Artwork 와 다른 Artwork 의
- Edge 모두 연결 해제



Programming - python

- Dynamic Graph

- File function

- 영구적 Play

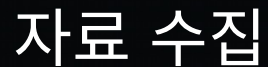
- Dictionary

- Python data structure
 - Key() : Artwork
 - Values() : Information

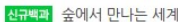
- Programming

- PyQt5 GUI tool

- 입력 , 출력 , 삭제 선택
 - 입력
 - 출력 : What Category
 - 삭제



- 네이버
- 미술 백과
- 조회수에 따라
- 다양한 작품에 대한
- 정확한 정보 제공
- 결점 : 그림 복사 X



정보 제공처

국립고궁박물관

국립중앙박물관

프랑스국립박물관연합(RMN)

한국데이터베이스진흥원 공공저작권 신탁관리시스템

한국사전연구사 한국미술오천년

전체펼치기 ▾





현대미술이 궁금하다면

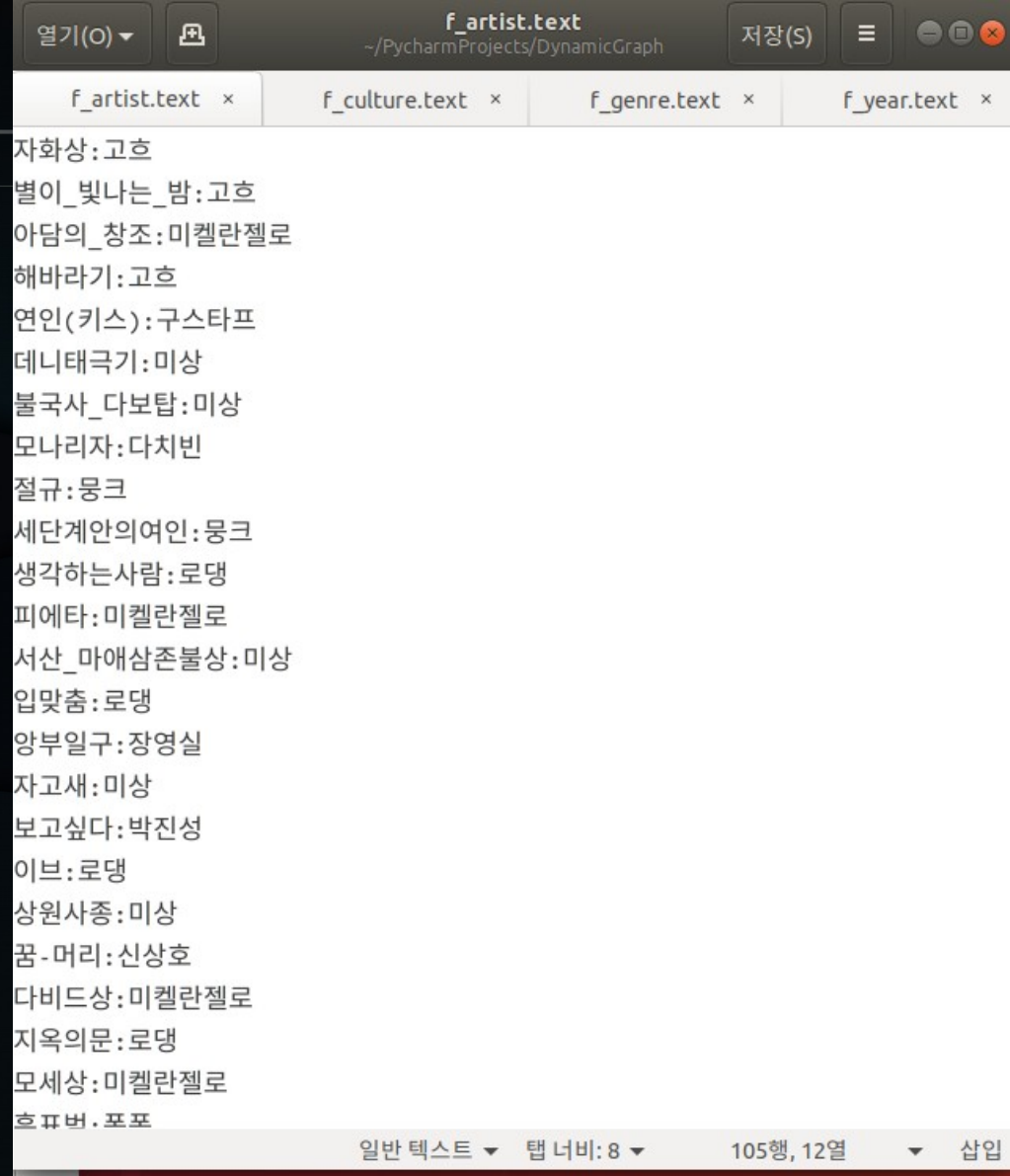
hello!ARTIST

우리 시대 작가들과의 만남



자료 수집 - 파일 연동

-    
f_artist.text f_culture.text f_genre.text f_year.text
- 총 100 개의 node 수집
- Node 는 사용자의 입력에 따라 끊임없이 추가 - 파일에 저장
- 장르별 조회수가 높은 작품 위주로
- 일차적 자료수집





Design - Outline

Input.py x Delete.py x Output_WhatKind.py x Playlist_DynamicGraph.py x

프로그램 : smart playlist

초기화면 : 입력할지 출력할지 결정 (QCheckBox)

1) 입력 : 정보 입력 -> node(딕셔너리.키)로 저장, edge(딕셔너리.벨류)로 연결

2) 출력 :

a) 어떤 특징을 갖는 작품 list를 출력할 것인지 선택

b) 해당 특징(벨류)를 갖는 작품(키) 출력 | GUI로 구현

3) 삭제 : 딕셔너리에서 삭제

Dynamic Graph

입력 : 작품과 정보 -> 기존에 입력된 정보값이 동일한 노드(작품)들과 연결

삭제 : 작품 -> 딕셔너리의 keys()값을 조사하며 있으면 삭제

- Consist of 6parts

- 1. 초기화면 - 추가, 삭제, 재생 선택
- 2. 추가 module
- 3. 삭제 module
- 4. 재생 intro : What Category module
- 5. 재생목록 module



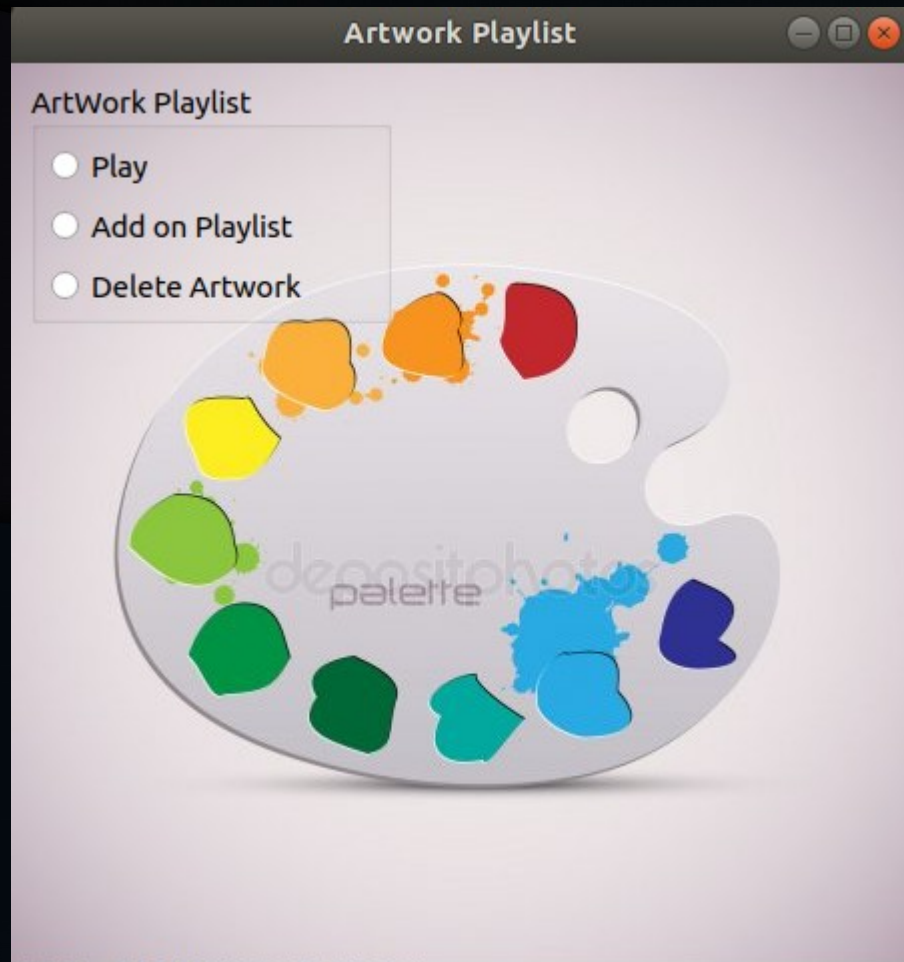
프로그램 실행

- 프로그램 실행에 관한 부분은
- **Dynamic Playlist** 실행
메뉴얼 을 통해
- 설명해놓았습니다 .
- 참고 부탁드립니다 .

- 구현영상
 - Git 업로드 용량 제한으로
메일로 보내드렸습니다
 - 후반부 Artwork 파일연동
중 Artist 부분이 미처
연동되지 않았던 점 인지하고
수정했습니다 .



구현 - 1. 초기화면



- 프로그램 메인 화면
- 노드 입력 / 삭제 / 그래프 출력 결정
- 선택에 따라 해당 모듈 실행



핵심 코드 - 1. 초기화면

```
#클릭에 따라 프로그램 동작
def radioButtonClicked(self):
    if self.radio1.isChecked():
        fname = Output_WhatKind()
    elif self.radio2.isChecked():
        fname = Input()
    elif self.radio3.isChecked():
        fname = Delete()
    self.statusBar.showMessage("고고링")

#메인
if __name__ == '__main__':
    app = QApplication(sys.argv)
    ex = Select()
    sys.exit(app.exec_())
```

- 각 모듈을 끌어와 동작하는
- 프로그램의 중심틀

```
#기능 선택
class Select(QWidget):
    def __init__(self):
        super().__init__()
        self.title = 'Artwork Playlist'
        self.initUI()

    def initUI(self):
        self.setWindowTitle(self.title)
        self.setGeometry(0, 0, 300, 300)

        # 배경 이미지
        label = QLabel(self)
        pixmap = QPixmap('palette')
        label.setPixmap(pixmap)
        self.resize(pixmap.width(), pixmap.height())

        groupBox = QGroupBox("ArtWork Playlist", self)
        groupBox.move(10, 10)
        groupBox.resize(180, 120)

        #선택 함꼐
        self.radio1 = QRadioButton("Play", self)
        self.radio1.move(20, 40)
        self.radio1.clicked.connect(self.radioButtonClicked)

        self.radio2 = QRadioButton("Add on Playlist", self)
        self.radio2.move(20, 70)
        self.radio2.clicked.connect(self.radioButtonClicked)

        self.radio3 = QRadioButton("Delete Artwork", self)
        self.radio3.move(20, 100)
        self.radio3.clicked.connect(self.radioButtonClicked)

        self.show()
```




핵심 코드 - 2. Artwork 추가

The screenshot shows a window titled "Select.py" with standard OS window controls (minimize, maximize, close). Inside the window, there are five text input fields labeled "Artwork", "Artist", "Year", "Genre", and "Culture". To the right of the "Artwork" field is a button labeled "Add on Playlist". To the right of the "Culture" field is a button labeled "Not anymore".

- 프로그램

- 각 값들을 입력 → 기존에 입력받은 Artwork로 이루어진 dynamic-graph 구동 → 그래프에서 같은 information(ex: 고흐, 동양 ...)을 갖는 Artwork와 모두 연결

- Node 추가

- Information(Artist/Year/Genre/Culture) : edge
- 같은 Information 값 (ex: 고흐, 동양 ...)을 갖는 node끼리 연결
- Information은 총 4가지이기 때문에 4가지 종류의 edge를 가지며 해당 종류에 따라 다양한 weight 값 (ex: genre : 그림, 조각, 건축 ...)을 갖는다

- Not anymore : 프로그램 종료



핵심 코드 - 2. Artwork 추가

```
#Dynamic Graph 추가 - Node 연결
#Dynamic Graph 파일 쓰기
def inputNode(self):
    Artwork = self.lineEdit1.text()
    Artist = self.lineEdit2.text()
    preYear = self.lineEdit3.text()
    preYear = int(int(preYear)/100*100) #100년 단위로 분류
    Year = str(preYear)
    Genre = self.lineEdit4.text()
    Culture = self.lineEdit5.text()

    n_Artist[Artwork] = Artist
    n_Year[Artwork] = Year
    n_Genre[Artwork] = Genre
    n_Culture[Artwork] = Culture

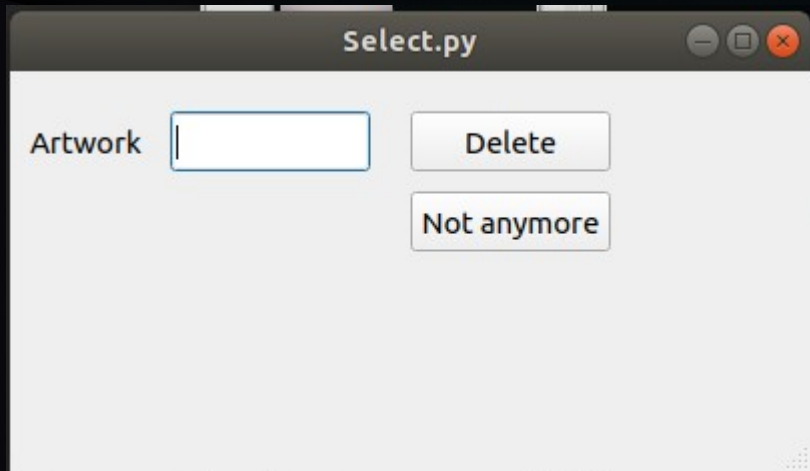
    f_artist = open("f_artist.text", 'at')
    f_culture = open("f_culture.text", 'at')
    f_genre = open("f_genre.text", 'at')
    f_year = open("f_year.text", 'at')

    f_artist.write(Artwork+':'+Artist+'\n')
    f_culture.write(Artwork+':'+Culture+'\n')
    f_genre.write(Artwork+':'+Genre+'\n')
    f_year.write(Artwork+':'+Year+'\n')
```

- 딕셔너리에 추가
 - 실행중의 다이나믹 그래프 구동
- 파일에 추가
 - 종료후에도 영구적인 다이나믹 그래프 설



핵심 코드 - 3. Artwork 삭제



- Node 삭제 → Node 는 Information 값 lose → edge 해제 → 재생목록 (Dynamic Graph) 에서 Node 삭제

- Node 삭제
 - 다른 node 와 모든 edge 해제
- Not anymore : 프로그램 종료



핵심 코드 - 3. Artwork 삭제

```
#Dynamic graph에서 삭제 -> 삭제에 따른 edge 연결해제 ~ 파일에 저장
def deleteArt(self):
    delnode = self.lineEdit.text()
    if delnode in n_Artist:
        n_Artist.pop(delnode)
    if delnode in n_Year:
        n_Year.pop(delnode)
    if delnode in n_Genre:
        n_Genre.pop(delnode)
    if delnode in n_Culture:
        n_Culture.pop(delnode)
    self.statusBar.showMessage(self.lineEdit.text())

    f_artist = open("f_artist.text", 'at')
    f_culture = open("f_culture.text", 'at')
    f_genre = open("f_genre.text", 'at')
    f_year = open("f_year.text", 'at')

    f_artist.write(delnode+": "+"delete"+"\\n")
    f_culture.write(delnode+": "+"delete"+"\\n")
    f_year.write(delnode+": "+"delete"+"\\n")
    f_genre.write(delnode+": "+"delete"+"\\n")

    f_artist.close
    f_culture.close
    f_genre.close
    f_year.close
```

- 딕셔너리에서 삭제
 - Dictionary 함수 pop를 통해 Dictionary 내에서 삭제
 - File 와 연동
- 파일에서 삭제
- Dictionary 와 연동되며 삭제 시, Artwork : delete 가 파일에 쓰여짐
- delete 가 쓰여지면 이를 삭제라고 인지하고 더이상 dynamic graph 에서 작품을 읽어 드리지 않음



핵심 코드 - 4. 재생 -1

- ☐ Artist
- ☐ Year
- ☐ Genre
- ☐ Culture

- ☒ Artist
- ☐ Year
- ☐ Genre
- ☐ Culture

What do you Want to Play?

미켈란젤로

Play

- Dynamic graph 출력 :
사용자가 보고싶은 작품특성 입력 !
 - 특정 edge 로 연결된 작품
node 선별적 출력 ; 입력값에
따라 다이나믹한 Dynamic
Graph 표현 가능



핵심 코드 - 5. 재생 -2

- ☒ Artist
- ☐ Year
- ☐ Genre
- ☐ Culture

What do you Want to Play? 미켈란젤로

Play

아담의_창조

피에타

다비드상

모세상

라우렌시오도서관

아담의창조

인생의꿈

최후의심판

- Display
- 특정 Edge 로 연결된 Node 출력
- 보고싶은 예술 작품들 선별적 감상 혹은 재생 가능



핵심 코드 - 1. 초기화면 -1

```
import sys
from PyQt5.QtWidgets import *
from PyQt5.QtGui import QIcon, QPixmap
from Playlist_DynamicGraph import *
from Input import *
from Delete import *
from Output_WhatKind import *
```

- Dynamic graph 출력 :
사용자가 보고싶은 작품특성 입력 !
 - 특정 edge 로 연결된 작품
node 선별적 출력 ; 입력값에
따라 다이내믹한 Dynamic
Graph 표현 가능

```
#파일 불러오기
f_artist = open("f_artist.text", 'rt')
f_culture = open("f_culture.text", 'rt')
f_genre = open("f_genre.text", 'rt')
f_year = open("f_year.text", 'rt')

#파일에 저장된 정보를 그래프에 업로드
#프로그램이 꺼졌을때도 기록된 정보들을 잃지 않기 위해
line1 = f_artist.readlines()
for str_A in line1:
    str_Akey = str_A.split(':')[0]
    str_Avalue = str_A.split(':')[1]
    n_Artist[str_Akey] = str_Avalue

line2 = f_year.readlines()
for str_Y in line2:
    str_Ykey = str_Y.split(':')[0]
    str_Yvalue = str_Y.split(':')[1]
    n_Year[str_Ykey] = str_Yvalue

line3 = f_genre.readlines()
for str_G in line3:
    str_Gkey = str_G.split(':')[0]
    str_Gvalue = str_G.split(':')[1]
    n_Genre[str_Gkey] = str_Gvalue

line4 = f_culture.readlines()
for str_C in line4:
    str_Ckey = str_C.split(':')[0]
    str_Cvalue = str_C.split(':')[1]
    n_Culture[str_Ckey] = str_Cvalue
```



핵심 코드 - 4. 재생 -1

```
#입력에 대한 설명, 가이드라인
tistmsg = ""
for x in list(n_Artist.values()):
    tistmsg = tistmsg + " / " + str(x);
yearmsg = ""
for x in list(n_Year.values()):
    yearmsg = yearmsg + " / " + str(x);
enremsg = "drawing / performance / sculpture / architecture / etc"
cultmsg = "동양/서양"

#fir kind = 0
se_kind = 0
kind2 = ""
a_num=0
```

```
f_artist = open("f_artist.text", 'rt')
f_culture = open("f_culture.text", 'rt')
f_genre = open("f_genre.text", 'rt')
f_year = open("f_year.text", 'rt')
```



핵심 코드 - 5. 재생 -2

```
#재생 intro - 재생값 입력 1
def play_intro(self):
    if self.checkBox1.isChecked() == True:
        msg = "Artist Category : " + tistmsg
        global fir_kind
        fir_kind = 1
        self.play_final()
    if self.checkBox2.isChecked() == True:
        msg = "Year Category : " + yearmsg
        fir_kind = 2
        self.play_final()
    if self.checkBox3.isChecked() == True:
        msg = "Genre Category : " + enremsg
        fir_kind = 3
        self.play_final()
    if self.checkBox4.isChecked() == True:
        msg = "Genre Category : " + cultmsg
        fir_kind = 4
        self.play_final()
```

```
#재생 final - 재생값 입력 2
def play_final(self):
    # Label
    label = QLabel("What do you Want to Play?", self)
    label.move(10, 100)
    label.resize(200, 30)

    # QLineEdit
    self.lineEdit = QLineEdit("", self)
    self.lineEdit.move(200, 100)

    self.pushButton1 = QPushButton("Play", self)
    self.pushButton1.move(310, 100)
    self.pushButton1.clicked.connect(self.PlayOn)

    # StatusBar
    self.statusBar = QStatusBar(self)
    self.setStatusBar(self.statusBar)

    label.show()
    self.lineEdit.show()
    self.pushButton1.show()
```




핵심 코드 - 5. 재생 -3

#재생 - 입력한 edge로 연결되어 있는 작품 수집 출력

```
def PlayOn(self):
    kind1 = self.lineEdit.text()
    d = 'delete'
    i=150
    j=0
    num=0
    global a_num
    print(fir_kind)
    for x in range(0,314):
        if num == 21:
            j += 120
            i = 150
            num = 0
        btn1 = QPushButton("", self)
        btn1.move(20+j, 20 + i)
        btn1.clicked.connect(QCoreApplication.instance().quit)
        btn1.show()
        i += 40
        num+=1
    if fir_kind == 1:
        i=150
        j=0
```

```
if fir_kind == 1: #Artist
    i=150
    j=0
    for x, p_Artist in n_Artist.items():
        if p_Artist == d or p_Artist == d + '\n':
            continue
        if kind1 == "":
            if num == 21:
                j += 120
                i = 150
                num = 0
            self.btn1 = QPushButton(x, self)
            self.btn1.move(20+j, 20 + i)
            self.btn1.clicked.connect(QCoreApplication.instance().quit)
            i += 40
            num+=1
            self.btn1.show()
        if p_Artist == kind1 or kind1 + "\n" == p_Artist:
            if num == 21:
                j += 120
                i = 150
                num = 0
            btn1 = QPushButton(x, self)
            btn1.move(20+j, 20 + i)
            btn1.clicked.connect(QCoreApplication.instance().quit)
            i += 40
            num += 1
            btn1.show()
```




핵심 코드 - 6. 재생목록 (graph)

```
n_Artist = {}  
n_Year = {}  
n_Genre = {}  
n_Culture = {}
```

#파일 불러오기

```
f_artist = open("f_artist.text", 'rt')  
f_culture = open("f_culture.text", 'rt')  
f_genre = open("f_genre.text", 'rt')  
f_year = open("f_year.text", 'rt')
```

- Dynamic_Graph

Dictionary와 File로 구성

- Dictionary

- Key : Node || Values : edge(한 node당 4개의edge를 갖는다)
- 프로그램 구동 중 Dynamic_Graph 사용

- File

node 값 : edge 값 형식으로 저장

- 파일을 불러와 해당 노드와 엣지값을 읽어드리는 형식으로 Dynamic Graph 불러옴
- 프로그램 구동 중, 노드의 추가와 삭제 그리고 대용량 노드 추가를 가능하게

#파일 불러오기

```
f_artist = open("f_artist.text", 'rt')  
f_culture = open("f_culture.text", 'rt')  
f_genre = open("f_genre.text", 'rt')  
f_year = open("f_year.text", 'rt')
```

#파일에 저장된 정보를 그래프에 업로드

#프로그램이 꺼졌을때도 기록된 정보들을 잃지 않기 위한

```
line1 = f_artist.readlines()
```

```
for str_A in line1:
```

```
    Alist = str_A.split(':')
```

```
    str_Akey = Alist[0]
```

```
    str_Avalue = Alist[1]
```

```
    n_Artist[str_Akey] = str_Avalue
```

```
line2 = f_year.readlines()
```

```
for str_Y in line2:
```

```
    Ylist = str_Y.split(':')
```

```
    str_Ykey = Ylist[0]
```

```
    str_Yvalue = Ylist[1]
```

```
    n_Year[str_Ykey] = str_Yvalue
```

```
line3 = f_genre.readlines()
```

```
for str_G in line3:
```

```
    Glist = str_G.split(':')
```

```
    str_Gkey = Glist[0]
```

```
    str_Gvalue = Glist[1]
```

```
    n_Genre[str_Gkey] = str_Gvalue
```

```
line4 = f_culture.readlines()
```

```
for str_C in line4:
```

```
    Clist = str_C.split(':')
```

```
    str_Ckey = Clist[0]
```

```
    str_Cvalue = Clist[1]
```

```
    n_Culture[str_Ckey] = str_Cvalue
```



- Git 업로드 용량 제한으로 메일로 보내드렸습니다
- 후반부 Artwork 파일연동 중 Artist 부분이 미처 연동되지 않았던 점 인지하고 수정했습니다 .



Q&A
