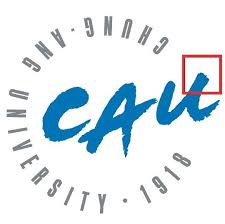
자연어 처리

자바로 자연어

Report

Subject: 영화 내 등장인물들 간의 관계 분석

Team leader: 민성재

|  |  |
| --- | --- |
| 학번 | 이름 |
| 20145223 | 민성재 |
| 20146703 | 박지호 |
| 20146518 | 손창우 |
| 20143583 | 김기환 |
| 20140786 | 조장연 |
| 20142127 | 이준호 |

1. 프로젝트 주제

프로젝트 주제는 ‘영화내 등장인물들의 관계 분석’이다. 영화의 스크립트를 가지고 영화 내의 등장인물들을 골라낸다. 영화의 등장인물들을 찾아서 등장인물들의 관계를 network형태로 표현하여 시각화 하는 것이 이번 프로젝트의 목표이다.

1. 프로젝트 목적

영화를 보다 보면 영화내에 수 많은 등장인물들이 나온다. 다양한 장르 별로 영화를 9개 선정하였다. 선정한 9개의 영화는 다음과 같다

|  |  |
| --- | --- |
| 번호 | 영화이름 |
| 1 | DeadPool (데드풀) |
| 2 | UP |
| 3 | 500 days of Summer(500일의 썸머) |
| 4 | 17 again |
| 5 | Kung Fu Panda (쿵푸팬더) |
| 6 | Beauty and the Beast(미녀와 야수) |
| 7 | Thor Ragnarok (토르 라그나로크) |
| 8 | Interstellar (인터스텔라) |
| 9 | Spider man (스파이더맨) |

9개의 영화의 평균 등장인물수가 약 34명 정도 나온다. 쿵푸팬더와 500일의 썸머는 등장인물이 적은 편이지만 미녀와 야수와 데드풀은 등장인물이 40명 정도로 많은 편이었다. 하지만 영화를 보고나서 생각나는 등장인물은 주인공 몇 명 밖에 생각나지 않는다. 캐릭터들이 어떠한 관계를 가지는지도 잘 생각나지 않는다. 누가 누구와 연인, 친구 관계이고 적 관계인지 말이다. 그리고 영화를 봐도 복잡해서 영화가 이해가 잘 안되는 영화들도 존재한다. 그래서 영화 등장인물 관계도를 검색하게 되면 블로그 등 사이트에서 볼 수도 있다. 하지만 그러한 글을 작성하려면, 영화를 완벽히 이해해야 하기 때문에 쉽지 않다. 그래서 등장인물들의 관계를 편하게 보고, 한 눈에 볼 수 있도록 하고자 함이 이번 프로젝트의 목표이다.

1. 프로젝트 코드 설명
2. Data Structure
   1. Scene class

scene에서 나오는 등장인물(actor)와 등장인물의 대사 횟수를 저장한다

등장인물들은 actor class type의 array list에 저장하였고 대사 횟수는 integer형태로 저장하였다. 이렇게 한 씬에 대해서 정보들을 저장한다.

* 1. Actor class

이 class는 등장인물(actor)에 대한 정보를 담는 class이다. 등장인물들의 이름(name), 등장인물의 대사 횟수, 각 노드들의 weight의 정보를 담는다. 세가지 정보에 대한 getter와 setter를 만들었다.

Weight에 대한 정보를 계산하기 위한 함수인 computeWeight 함수도 있다. 이 함수는 한 scene내에서 자신이 얼마나 비중이 큰지(대사를 얼마나 많이 했는지)를 나타내는 척도이다. 식은 아래와 같다

Weight = scene내에서 등장인물 A가 말한 횟수 / scene에서의 총 대사 횟수

* 1. MapData class

인물 관계도를 위한 node와 edge값을 가지고 있는 데이터 클래스.

* nodeData: TreeMap<String, Double>

Actor의 이름과 가중치. 이후 관계도에서 node의 이름과 크기로 표현된다.

* edgeData

Actor사이간의 관계의 크기. 이후 관계도에서 node간의 edge의 굵기로 표현된다.

* modeStr

output파일을 생성할 때 node와 edge생성기준을 명시해주기 위한 분석용 문자열

1. File parsing
   1. FileParser class

스크립트를 분석하기 위해서 제일 먼저 스크립트 파일을 읽는 함수들이 있는 class이다.

1. 우선 읽을 스크립트 텍스트 파일을 받아와서 FileReader와 BufferedReader를 통해 파일을 읽는다
2. 파일을 읽었을 때 등장인물들은 한 줄에 이름이 모두 대문자로 쓰여 있다. 하지만 스크립트에서 등장인물의 행동이나 상황을 지시하는 지시자들도 대문자로 쓰여 있다.

그러하기 때문에 이를 구분하기 위해서 indicator array list를 만들어서 이 array list에 담긴 지시자들은 걸러낸다. 이렇게 되면 등장인물을 구별해내기 위한 전 처리 작업이 끝나게 된다.

1. 그 다음 scene(씬)을 나누는 작업을 한다. Scene은 장면 표제(Scene Heading)을 기준으로 나누기로 정하였다. 장면 표제는 슬러그 라인(slug line)이라고 불리기도 하는 이것은, 위치를 설명함으로써 독자의 스테이지를 설정하고 모두 대문자로 표기된다. 실내인지 외부인지 나타내는 장면 표제에는 두 가지가 있다.

이렇게 기준을 선정한 이유는 scene을 나누는 것으로 보이는 번호가 있지만 스크립트마다 형식도 다르고 없는 것들도 있다. 그래서 장면 표제는 스크립트를 작성하는 업계의 표준 사항이기 때문에 모든 스크립트 내에 존재한다. 그래서 이를 이용해 scene을 나누기로 하였다.

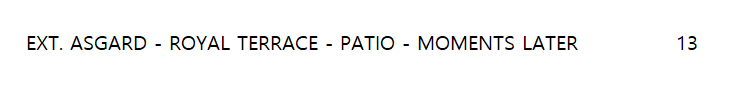
1. INT

실내 장면을 나타내는 지시자이다.

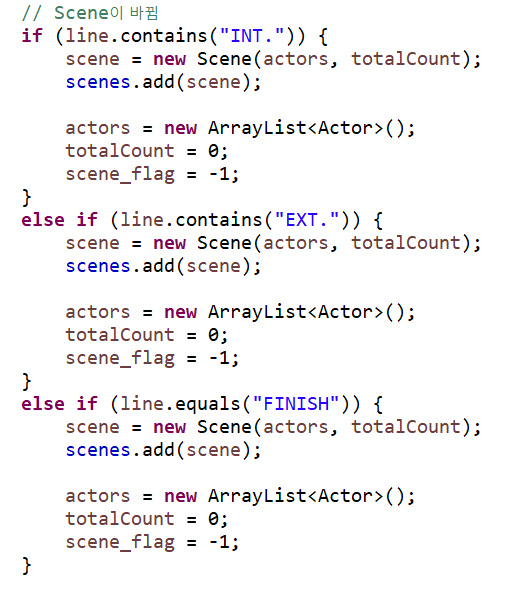
1. EXT

실외 장면을 나타내는 지시자이다.

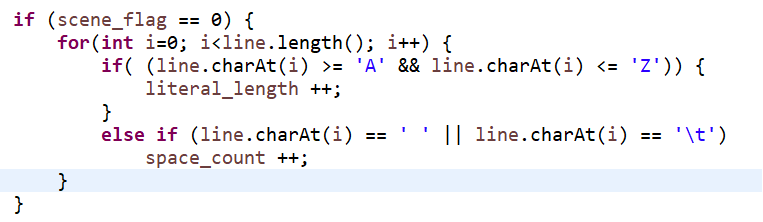
실제 아래와 같이 스크립트내에 쓰여있다



장면을 위와 같은 기준으로 나누고 마지막 scene을 인식하기 위해서 FINISH를 추가하여 인식하도록 하였다. 아래 코드에서 위의 과정을 진행하였다.



1. 장면을 나누고 장면마다 등장인물들을 찾는다. 등장인물(character)들은 스크립트 내에서 이름이 모두 대문자로 쓰여 있다. 그래서 한 줄에 글자가 모두 대문자로 쓰여 있는지 아래 코드에서 확인한다. 그래서 모두 대문자로 쓰여 있을 경우, 등장인물로 인식하고 등장인물들을 뽑아 내기 시작한다.



1. 등장인물들의 이름을 뽑아내서 scene마다 어떤 등장인물들이 나오고 몇 번 나오는지(대사를 몇번하는지)를 검사한다. 이때 앞에 나왔던 등장인물들이 뒤에도 계속해서 나오는 경우가 많다.

그대로 넣게 되면 똑 같은 인물들이 array list에 여러 번 담기기 때문에 isReduplication 함수를 통해 먼저 중복이 있는지 검사한다. 검사해서 등장인물이 이미 array list에 담겨 있으면 다시 넣지 않고 등장횟수에 1을 더해준다. 그렇지 않은 경우는 array list에 새로 추가한다.

1. 위와 같은 과정을 스크립트가 끝날 때까지(FINISH가 나올 때까지) 계속해서 반복하여 scene마다 등장인물들을 찾아내고 몇 번 나오는지 찾아내서 관계분석에 활용한다.
2. Script analyze
   1. ScriptAnalyzer class

Script 파일을 분석하기 위한 클래스이다. Main함수에서 이 클래스에 접근하여 대본을 분석하고 관계도를 그려주기 위한 Node와 Edge값을 얻어 낼 수 있다.

* analyzeScriptFile(String inputFilePath)

Script 파일을 매개변수로 받아와 해당파일을 FileParser클래스를 이용해 분석한다.

분석 결과를 scenes라는 멤버 변수에 저장한다

* generateActorMapData(String outputFilePath, int node\_mode, int edge\_mode)

scenes를 통하여 정해진 기준으로 관계도를 그려주기 위한 map data를 생성한다.

매개변수로 저장 위치와 node와 edge를 계산하는 기준을 조절할 수 있다.

* printData(MapData mapData , String fileName)

MapDataGenerator클래스를 통해 가져온 mapData를 파이썬 환경에서 이용할 수 있는 데이터 형식으로 출력해준다.

* 1. MapDataGenerator class

Scene을 기준으로 분리한 영화에 대해서 관계도를 생성을 위한 mapData를 만들어주는 클래스이다. Node와 Edge의 조건들이 상수로 지정하여 외부에서 mapData의 생성 조건을 정의해 줄 수 있다.

* getMapData(int NODE\_PARAM, int EDGE\_PARAM, ArrayList<Scene> scenes)

MapData를 생성하여 리턴한다. 파라미터로 node와 edge생성 조건대로 mapdata를 생성하며 ScriptAnalyzer에서 분석된 scene별로 나눠진 scenes를 받아온다.

Node와 edge의 값에 따라 아래의 함수들이 각기 수행되어 mapdata를 계산한다.

* fillEdgeDataByWeight1(ArrayList<Scene> scenes)

같은 scene에 함께 등장하는 빈도에 따라 더 높은 관계를 가진다고 가정하여 scene에 등장하는 actor들 간에 1의 가중치를 주어 edge값을 계산하였다. 많은 scene에 함께 나온 actor들은 더 큰 관계 값을 가지게 된다.

* fillEdgeDataByWeightMultiplex(ArrayList<Scene> scenes)

같은 scene에 함께 나온 actor라도 모두 같은 정도의 관계를 가지지는 않는다. 그 점을 보완하여 scene내 actor마다 가지고 있는 가중치의 값은 scene내에서 대사의 비중을 뜻하며 더 대화를 많이 한 actor들이 scene내에서도 높은 관계를 가진다고 가정하여 각 actor들의 가중치를 곱하여 edge값을 계산하였다. 즉 같은 scene에서 A가 5번, B가 4번, C가 1번 대사를 가지고 있다면 그 scene내에서 A와B의 관계가 가장 큰 관계 값을 가지게 된다.

* fillEdgeDataByWeightMultiplexAndCount(ArrayList<Scene> scenes)

위에서의 edge를 구하는 방식에는 하나의 문제점이 존재한다. 각각의 scene내에서는 관계 값이 적절할 수 있지만 대사가 100번 존재하는 scene과 대사가 10번 존재하는 scene에서의 actor간의 관계는 비율이 같더라도 다른 가중치를 가져야 하는 것이 정당하다. 이는 함수는 scene내에서의 값이 아닌 영화에서 scene마다의 globality를 고려한 edge추출 방식이다. 각 scene내에서 가중치를 곱한 후 scene에서의 대사의 횟수를 마지막에 곱하여 edge값을 계산해 준다. 이 과정을 거치면

* fillNodeDataByWeight1(ArrayList<Scene> scenes)

많은 scene에 등장하는 actor는 더 큰 비중을 가지는 배우라 가정한다. 이 경우 하나의 scene에 등장하는 actor들 마다 1의 가중치를 부여한다. 모든 scene에 대해 검사를 하게 되면 더 빈번히 나온 actor는 큰 가중치를 가지게 된다.

* fillNodeDataByWeightByCount(ArrayList<Scene> scenes)

같은 scene내에서도 많은 대사를 가지고 있는 actor가 더 높은 비중을 차지한다고 가정한다. 이 경우 scene내에서 대사 횟수만큼 actor의 가중치를 부여한다. 결국 총 영화에서 대사 횟수가 해당 node의 가중치가 된다.

1. NetworkX

이제 분석된 정보를 가지고 그래프(network)로 표현해야 한다. 이 부분은 그래프화 하기 좋은 파이썬(Python) 언어를 가지고 진행하였다. 파이썬에서 등장 인물들 간의 관계를 보여주기에 가장 적합한 그래프 형태가 network라고 생각했기 때문에, network 형태로 그래프를 그려줄 수 있는 network 라이브러리를 이용하기로 하였다. 그리고 처음 다뤄보는 라이브러리이기 때문에 공부를 먼저 진행하였다.

Networkx를 이용해 그래프를 그리기 위해서는 크게 node와 edge가 필요하기 때문에 이에 대해 먼저 공부하였다.

* 1. Node

등장인물들을 node 데이터에 넣어서 network를 나타내게 된다. 이때 등장인물들을 list에 모두 넣어서 표현 할 수 있다.

* 1. Edge

Edge는 등장인물을 이어주는 정보이다. 등장인물들의 관계에 따라서 edge의 width정보를 바꿔준다.

* 1. 그 외

그 외로 그래프를 예쁘게 그려주는 attribute들에 대해 조사하였다. Node의 크기를 크게해주는 node\_size, edge의 선의 굵기를 표현해주는 width, edge의 color를 표현해주는 color 속성 등을 추가하여 그래프를 그릴 수 있다.

* 1. Code implementation

3번에서 분석한 결과가 텍스트파일에 저장되어 있다. 이 텍스트 파일을 가지고 parsing하여 등장인물과 등장인물의 비중, 등장인물들 사이의 edge 크기 정보를 받아온다. 텍스트 파일은 아래와 같은 형식으로 저장되어 있다.



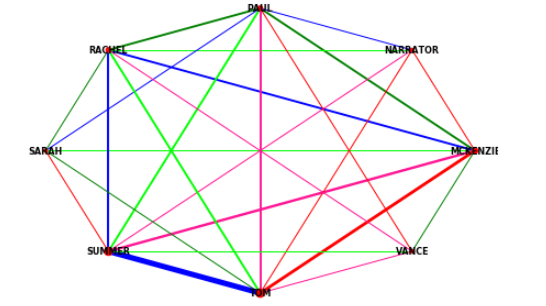
Parsing한 정보들을 가지고 등장인물들은 nodes list에 저장, 등장인물들의 비중을 나타 내주는 노드의 크기는 nodeSizes list에 저장, 인물들 간의 관계 정도를 나타내주는 weights list에 저장한다. 이 정보들을 가지고 그래프를 그려준다.

1. 결과 분석

Node 는 node weight -1 과 node weight count 로 두가지 방식의 가중치를 선정하고Edge는 weight -1 , weight multiply , weight multiply and count 세가지 방식으로 가중치를 선정한다. 따라서 map을 그리는 weight 알고리즘은 총 6개이다. 우리는 선정한 영화들 중에서 장르별로 6가지의 방식에 따라 어떤 차이점이 존재하는 지와 어떤 장점이 있는지를 파악하려고 한다.

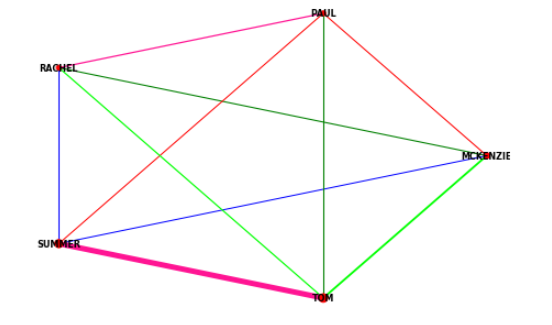
Romance: 500 days of summer

1. node weight -1 edge weight -1



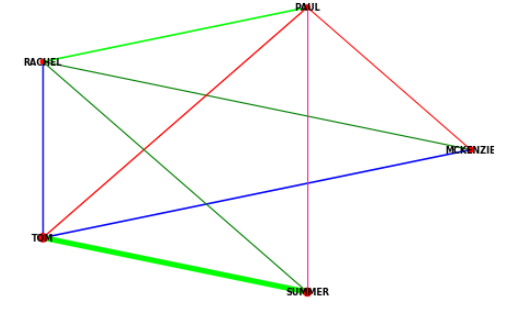
이 방식은 주인공들의 node 크기가 다른 등장인물과 비교했을 때 크지 않다 따라서 주인공이 누구인지 명확히 알 수 없다. 또한 TOM이 다니는 회사사장인 VANCE 와 사장 비서였지만 그만둔 SUMMER와의 edge가 TOM과 VANCE의 edge와 같은 관계를 나타내기 때문에 예상한 결과대로 나오지 않았다.

1. node weight -1 edge weight multiply



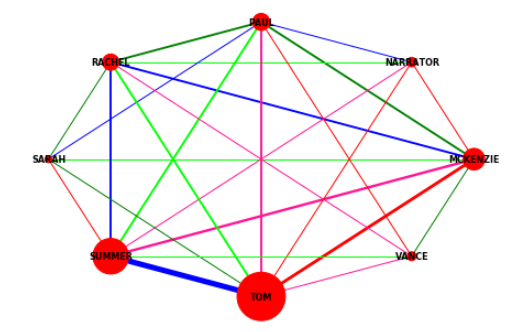
이 방식은 첫번째 방식과 마찬가지로 주인공들의 node 크기가 다른 등장인물과 비교했을 때 크지 않다 따라서 주인공이 누구인지 명확히 알 수 없다 또한 다른 사람들의 관계는 나타나지 않는다.

1. node weight -1 edge weight multiply and count



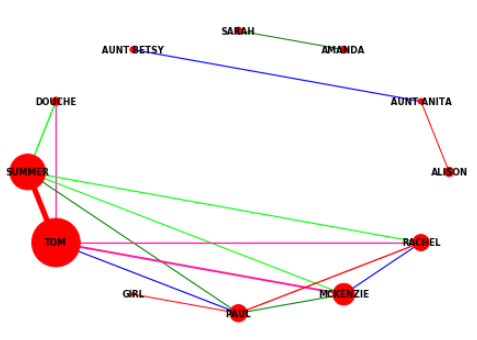
이 방식 또한 node의 크기가 명확하지 않다. Summer 와 MICKENZIE의 관계는 표현되지 않았다.

1. node weight count edge weight -1



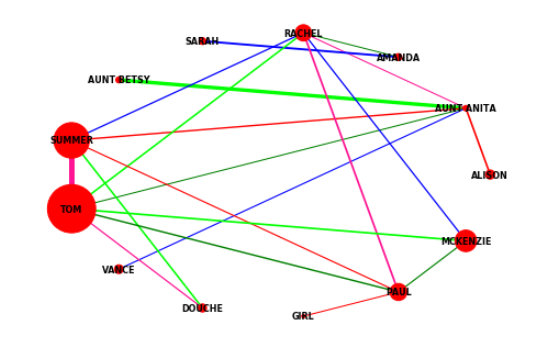
node크기를 통해 주인공을 판단할 수 있다. 그러나 주변인물의 node생성이 원활하지 않았다.

1. node weight count edge weight multiply



Node에 대해서는 문제없이 잘 나왔다. 그러나 edge를 표현할 때 주변인물과 주인공들의 관계는 표현되지 않았다.

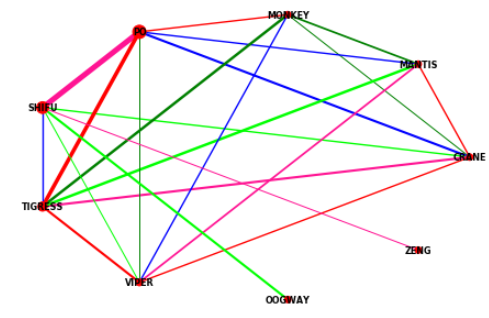
1. node weight count edge weight multiply and count



비중이 있는 등장인물들이 잘 표현되었고 그들 사이의 관계도 잘 표현되었다.

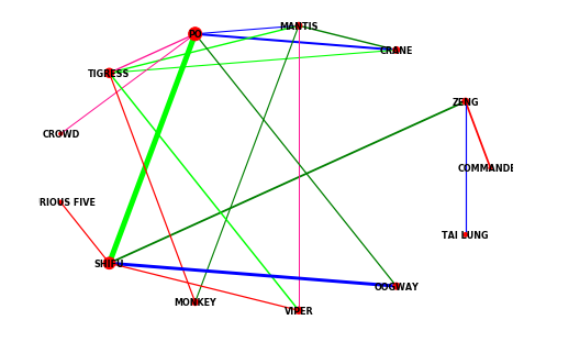
Animation: UP

1. node weight -1 edge weight -1



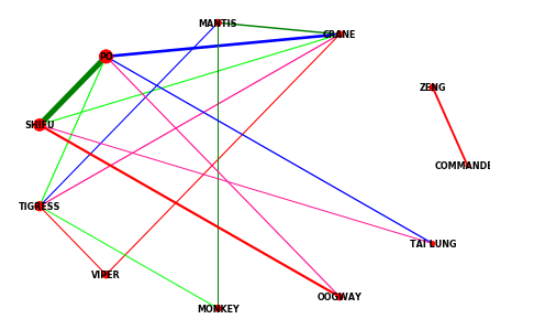
Node의 크기가 Romance와 같이 등장인물이 명확하지 않다. 악역인 TAI LUNG 이 나오지 않았다.

1. node weight -1 edge weight multiply



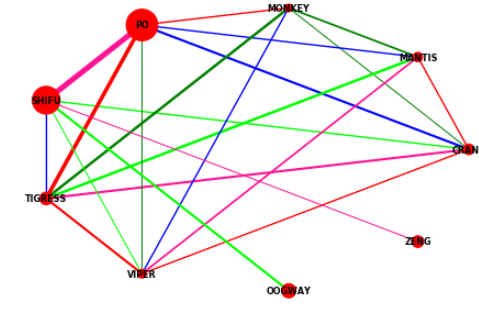
1과 마찬가지로 주인공을 명확하게 나타내지 못하며 악역과 주인공의 관계를 나타내지 못한다.

1. node weight -1 edge weight multiply and count



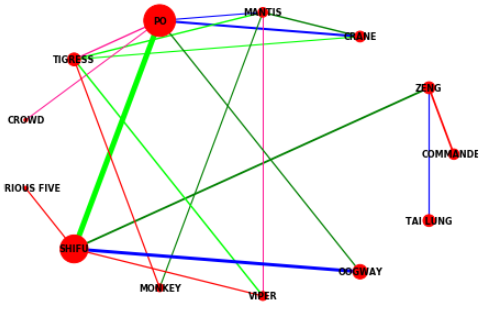
등장인물 사이의 관계는 잘 나왔지만 주인공 node 크기가 크지않아 누가 주인공인지 판단하기 어렵다.

1. node weight count edge weight -1



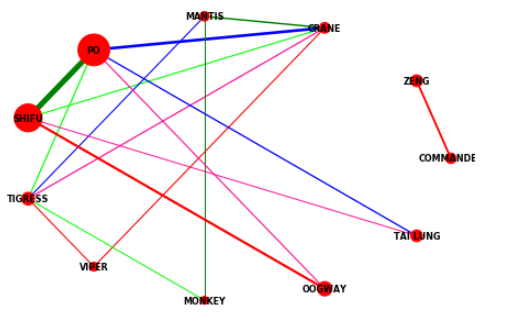
주인공이 누구인지 판단할 수 있지만 악역인 TAI LUNG이 나오지 않았다.

1. node weight count edge weight multiply



주인공이 누구인지 판단할 수 있지만 주인공과 악역 사이의 관계는 표현되어 있지 않다.

1. node weight count edge weight multiply and count

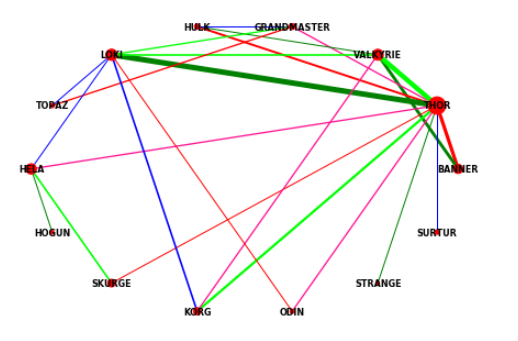


비중이 있는 등장인물들이 잘 표현되었고 그들 사이의 관계도 잘 표현되었다.

영화장르도 6번째 알고리즘을 통해 표현하는 것이 제일 좋다는 결론을 얻었다.

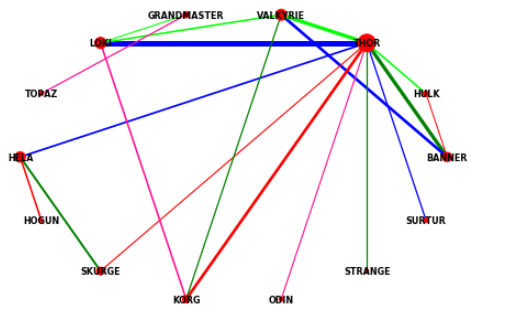
Hero: Thor Ragnarok

1. node weight -1 edge weight -1



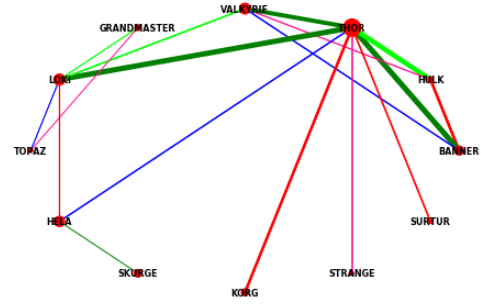
Node 크기로 주인공을 알기 힘들며 관계들도 알아보기 힘들다.

1. node weight -1 edge weight multiply



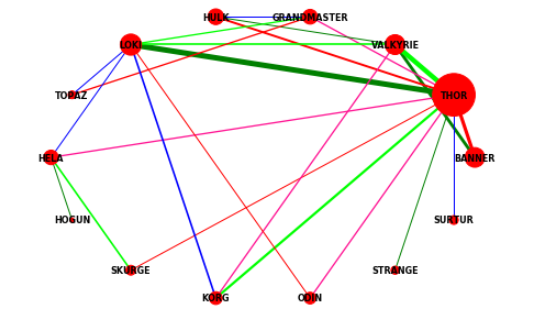
첫번째와 마찬가지로 주인공을 알아보기 힘들다. 또한 KORG 와 THOR사이의 관계가 VALKYRIE와 LOKI 보다 더 굵어야 하는데 굵지 않다.

1. node weight -1 edge weight multiply and count



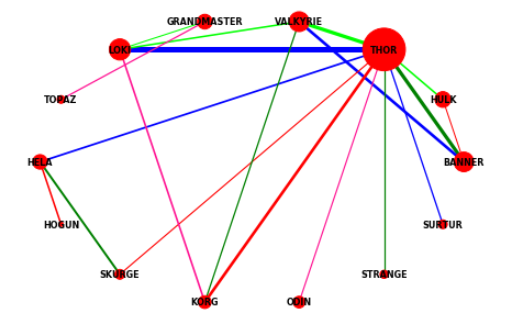
Node 크기가 역시 주인공을 명확하게 알기 힘들다.

1. node weight count edge weight -1



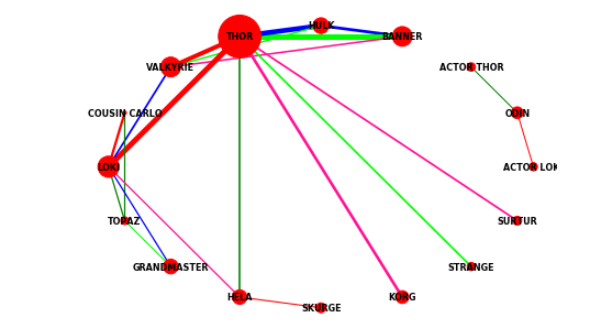
Node의 크기로 주인공을 알아볼 수 있다. Edge로 THOR와 HULK 사이의 관계가 더 굵게 나와야 한다.

1. node weight count edge weight multiply



Node의 크기로 주인공을 알아볼 수 있다. Edge로 THOR와 HULK 사이의 관계가 더 굵게 나와야 한다

1. node weight count edge weight multiply and count



Node를 통해 주인공도 판단할 수 있고 ACTORTHOR와 ACTORLOKI 까지 보여준다.

3가지 장르 테스트를 통해 6번째 알고리즘이 등장인물 관계도를 잘 나타냄을 알 수 있었다.

또한 장르마다 그림이 다른 특징을 가지는 걸 알 수 있었다.

로맨스는 남주인공과 여주인공의 비중이 제일 높고 관계 또한 굵게 나타냄을 알 수 있다. 애니메이션은 선한 역할을 가진 등장인물들이 많고 그들의 비중이 크다. 히어로 장르는 주인공의 비중이 제일 높고 그들을 도와주는 조력자들의 관계가 깊다. 또한 등장인물이 다양하게 나타남을 알 수 있었다.