8.30

기준 데이터프레임 생성

In [369]:

```
df1 = df[['Publication Number','Application Year','Title','Serial Number']]
len_d = len(df1) # 데이터프레임 행의 길이
indomain = ['1']*len_d
df1['indomain'] = indomain
df1 = df1.rename(columns={'Publication Number':'Cites Patents'})
#df1.loc[:,'Publication Number'] = df['Publication Number']
df1.head()
```

```
<ipython-input-369-5a3bf14a4d10>:4: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  df1['indomain'] = indomain
```

	Cites Patents	Application Year	Title	Serial Number	indomain
0	US3679949	1970	SEMICONDUCTOR HAVING TIN OXIDE LAYER AND SUBST...	0	1
1	US3658596	1970	FLEXIBLE SOLAR CELL MODULAR ASSEMBLY	1	1
2	US4024397	1970	Shock resistant encapsulated infrared detector	2	1
3	US3780424	1970	METHOD OF MAKING SILICON SOLAR CELL ARRAY	3	1
4	US3690953	1970	VERTICAL JUNCTION HARDENED SOLAR CELL	4	1

스플릿

In [382]:

```
def chainer(s):
    return list(chain.from_iterable(s.str.split(' | ')))
lens = df['Cites Patents'].str.split(' | ').map(len) #앞으로 공백 있는애, 뒤로 공백있는애 다 있음
res = pd.DataFrame({'Serial Number': np.repeat(df['Serial Number'],lens),
                    'Publication Number': np.repeat(df['Publication Number'],lens),
                    'Cites Patents' : chainer(df['Cites Patents'])
                   })
res.head()
```

	Serial Number	Publication Number	Cites Patents
0	0	US3679949	US3104188
0	0	US3679949	
0	0	US3679949	US3443170
0	0	US3679949	
0	0	US3679949	US3267317

구분자 제거

In [293]:
```
res = res[res['Cites Patents'] != '|']
```

In [294]:
```
res
```

	Serial Number	Publication Number	Cites Patents
0	0	US3679949	US3104188
0	0	US3679949	US3443170
0	0	US3679949	US3267317
0	0	US3679949	US3560812
0	0	US3679949	US3053926
...	...	...	...
8085	8085	US9035015	US20080087324A1
8085	8085	US9035015	US7605225
8085	8085	US9035015	US20070131270A1
8085	8085	US9035015	US20070154704A1
8085	8085	US9035015	US6297351

179598 rows × 3 columns

num of indomain edge = 48053

In [295]:

```
indomain_p = pd.merge(res,df1,on='Cites Patents',how='left',sort=False)
#indomain_p = indomain_p[indomain_p['indomain']=='1']
indomain_p = indomain_p[['Serial Number_x','Publication Number','Cites Patents','indomain']]
indomain_p = indomain_p[indomain_p['indomain']=='1']
indomain_p.rename(columns={'Serial Number_x':'Serial Number'},inplace=True)
indomain_p
```

	Serial Number	Publication Number	Cites Patents	indomain
117	22	US3735943	US3620846	1
123	23	US3735942	US3620846	1
148	26	US3948468	US3620846	1
274	48	US3760257	US3615853	1
320	58	US3969746	US3772770	1
...	...	...	...	...
179551	8082	US8846431	US6180869	1
179552	8082	US8846431	US4330680	1
179554	8082	US8846431	US4153476	1
179562	8083	US9029188	US8110431	1
179572	8083	US9029188	US8071418	1

48053 rows × 4 columns

edgedata

In [387]:
```
edge_data=indomain_p[['Serial Number','Publication Number','Cites Patents']]
edge_data.head()
```

	Serial Number	Publication Number	Cites Patents
117	22	US3735943	US3620846
123	23	US3735942	US3620846
148	26	US3948468	US3620846
274	48	US3760257	US3615853
320	58	US3969746	US3772770

edgedata cites patents serial number 만들기

In [402]:

```
s1 = df[['Publication Number','Serial Number']]
s2 = edge_data[['Cites Patents']]
s3 = pd.merge(s2,s1,left_on ='Cites Patents',right_on='Publication Number',how='left',sort=False)
#s3 = s3.dropna()
#s3 = s3.astype({'Serial Number':'int'})
s3.rename(columns={'Serial Number':'Serial_c'},inplace=True)
s3 = s3[['Cites Patents','Serial_c']]
s3
```

	Cites Patents	Serial_c
0	US3620846	7
1	US3620846	7
2	US3620846	7
3	US3615853	6
4	US3772770	36
...	...	...
48048	US6180869	2973
48049	US4330680	773
48050	US4153476	419
48051	US8110431	5997
48052	US8071418	6003

48053 rows × 2 columns

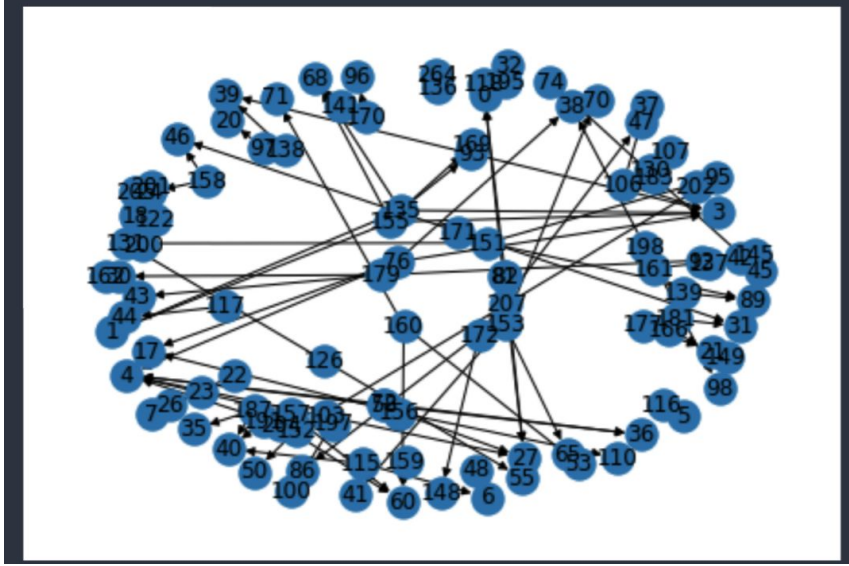
serial number로 표현가능

```
edge_data = pd.concat([edge_data,s3],axis=1)
edge_data
```

	index	Serial Number	Publication Number	Cites Patents	Cites Patents	Serial_c
0	117	22	US3735943	US3620846	US3620846	7
1	123	23	US3735942	US3620846	US3620846	7
2	148	26	US3948468	US3620846	US3620846	7
3	274	48	US3760257	US3615853	US3615853	6
4	320	58	US3969746	US3772770	US3772770	36
...	...	...	...	...	...	...
48048	179551	8082	US8846431	US6180869	US6180869	2973
48049	179552	8082	US8846431	US4330680	US4330680	773
48050	179554	8082	US8846431	US4153476	US4153476	419
48051	179562	8083	US9029188	US8110431	US8110431	5997
48052	179572	8083	US9029188	US8071418	US8071418	6003

48053 rows × 6 columns

```
G = nx.from_pandas_edgelist(edge_data.head(100),'Serial Number','Serial_c', create_using=nx.DiGraph
nx.draw(G, with_labels=True)
```



유사도 추가학습

- 자카드 유사도(Jaccard similarity) : 공통으로 가지는 원소의 비율을 구함

$$J(A,B) = \frac{|A \cap B|}{|A \cup B|} = \frac{|A \cap B|}{|A| + |B| - |A \cap B|}$$

- 청구항이 텍스트니까 텍스트 사이의 유사도를 구하면 청구항의 내용상 유사도를 구할 수 있지 않을까라는 생각을 해서 데이터를 써보려 했는데, 엑셀 데이터가 너무 길어서 제목으로 대체.

```
df1 = df[['Title']]
df1
```

	Title
0	SEMICONDUCTOR HAVING TIN OXIDE LAYER AND SUBST...
1	FLEXIBLE SOLAR CELL MODULAR ASSEMBLY
2	Shock resistant encapsulated infrared detector
3	METHOD OF MAKING SILICON SOLAR CELL ARRAY
4	VERTICAL JUNCTION HARDENED SOLAR CELL
...	...
8081	Solar cell and method of manufacturing the same
8082	N-type silicon solar cell with contact/protect...
8083	Solar cell and method for manufacturing the same
8084	Solderable polymer thick film conductive elect...
8085	Photovoltaic cell containing novel photoactive...

8086 rows × 1 columns

```
df2 = df1[df1['Title'].str.contains('mounting')]
```

- [Solar PV's 3 sub-components]
- solar cell : generate electricity
- PV module(panel) : bundle of solar cell
- mounting system : install & control PV system
- -> 제일 적을 것 같은 mounting 선택

```
df1[df1['Title'].str.contains('mounting')].count()
```

```
Title    56
dtype: int64
```

단어의 빈도수를 알기 위해 각 단어로 split

```
title1 = df2.iloc[0].to_string() #iloc하면 시리즈
title2 = df2.iloc[1].to_string()

t_title1 = title1.split()
t_title2 = title2.split()

print(t_title1)
print(t_title2)
```

```
['Title', 'Method', 'of', 'planar', 'mounting', 'of', 'silicon', 'solar', 'cells']
['Title', 'Solar', 'cell', 'mounting', 'and', 'interconnecting', 'assembly']
```

두 변수의 합집합과 교집합 구하기

```
u_title = set(t_title1).union(set(t_title2))
print(u_title)
```

```
{'planar', 'of', 'and', 'Solar', 'interconnecting', 'assembly', 'Title', 'silicon', 'mounting', 'solar', 'cells', 'Method', 'ce
ll'}
```

```
i_title = set(t_title1).intersection(set(t_title2))
print(i_title)
```

```
{'Title', 'mounting'}
```

자카드 유사도 구하기

```
print(len(i_title)/len(u_title))
```

```
0.15384615384615385
```

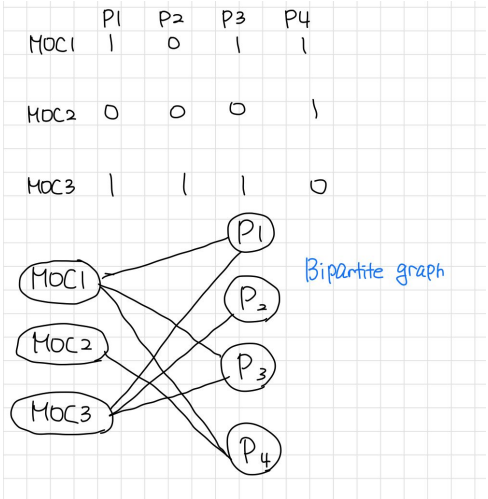
$$J(A,B) = \frac{|A \cap B|}{|A \cup B|} = \frac{|A \cap B|}{|A| + |B| - |A \cap B|}$$

그래프

- graph : (node,vertex)와 edge로 이루어져 있는 데이터를 표현하는 방식 G =(V,E)
- 노드(마디)의 종류가 하나로만 이루어져 있다면 : homogeneous graph
- 노드의 종류가 다르면 : heterogeneous
- 노드의 종류가 두개인데, 같은 종류끼리는 연결이 없고 다른 종류끼리만 연결된다면 : bipartite graph
- 그래프는 행렬형태로 나타낼 수 있다. X[1,5] = 6 -> 1번 node와 5번 node를 연결하는 edge의 값이 6
- 그래프는 벡터 공간보다 더 자유로운 표현이 가능하다. 벡터 공간에서는 p1, p2가 가깝고 p2,p3가 가까우면 p1과 p3 역시 가까울 가능성이 높지만, 그래프는 임의로 거리를 정할 수 있다.

그래프

- Metric space : 거리를 정의할 수 있는 공간 (points, distance)로 정의
- d(x,y) = 0 : 점 x와 y 사이의 거리가 0 -> 같은 점이다.
- d(x,y) = d(y,x) : x에서 y로의 거리와 y에서 x로의 거리가 같다.
- d(x,z) <= d(x,y) +d(y,z) : 삼각부등식이 성립
- 삼각부등식을 보면 두점끼리 가까우면 다른 점과도 가깝다는 것을 보여준다.
  -> 벡터는 그렇지만 그래프는 꼭 그렇지만은 않음.
- term frequency matrix를 bipartite graph로 표현가능



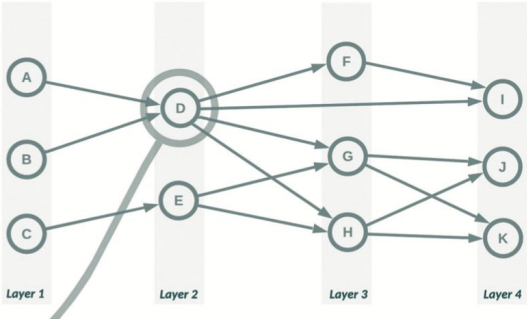
PageRank

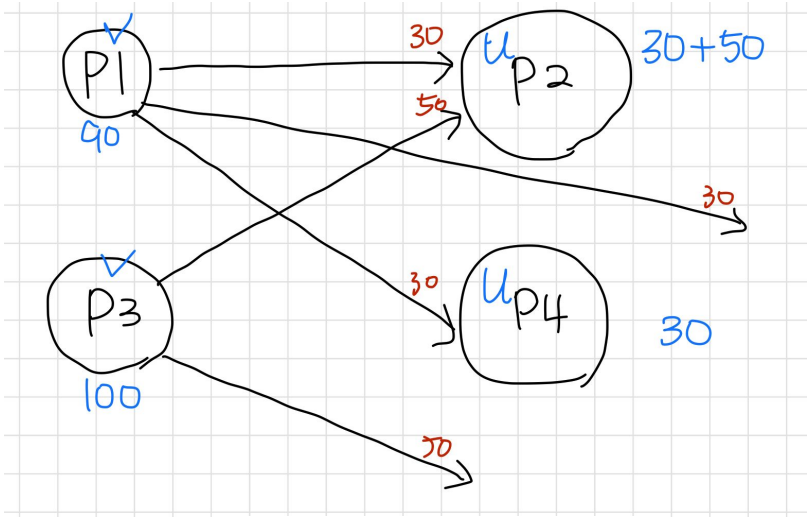
- 웹페이지 그래프에서 중요한 페이지를 찾아 검색결과를 re-ranking하는 과정에서 중요한 page의 ranking을 올리는데 사용.
- 많은 backlink를 지니는 page가 중요한 page.
- 각 page가 다른 page에 점수를 부여, 중요할수록 점수도 높고 많이 가지고 있다.
- backlinks가 적어도 중요한 page에 연결된 page는 중요
- Bibliometrics(계량서지학)의 citation

PageRank

$$PR(u) = c \times \sum_{v \in B_u} \frac{PR(v)}{N_v} + (1-c) \times \frac{1}{N}$$

- V = U노드의 backlink 노드
- Nv = V의 링크수
- c = 0,1사이 상수
- KP와는 접근 방향이 반대

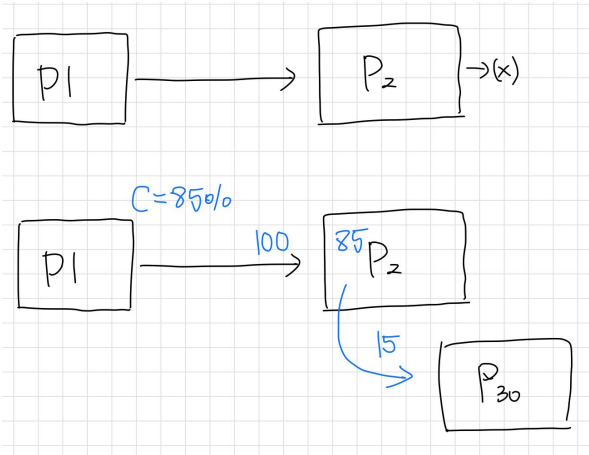




$$PR(u) = \sum_{v \in B_u} \frac{PR(v)}{N_v}$$

PageRank

- rank를 부여하는 과정을 여러번 수행하면서 ranking도 변해야한다.
- -> Markov model(마르코프 모델, 확률분야) 스텝마다 변하는 시스템
- backlink만 있는 경우 빠져나갈 수 없음. (dangling node)
- 따라서 그래프가 cyclic하게 만들어줘야함
- c라는 상수를 이용해서 c만큼은 분배한만큼 남아있고 1-c만큼은 임의로 분배되도록 한다.



PageRank

- Markov model에서 cycle을 반복하다보면 어느순간 숫자가 변하지 않는 steady state가 생김.
- 결국 random jump로 cyclic network를 형성.

Sparse matrix

- 0 값이 많은 행렬
- term frequency matrix -> sparse format
- 전체 종류에 비해 한 요소에 해당되는 경우가 현저히
- 적을때 -> 추천시스템도 비슷
- sparsity = 전체 벡터 공간에서 0인 값들의 비율
- -> ex)100개의 문서, 1000개의 단어
- 평균 10개의 단어가 한문서에 존재
- sparsity = (1000-10) * 100 / (100 * 1000) = 0.99

	patent 1	patent 2	patent 3	patent 4
MOC1	1	0	1	1
MOC2	0	0	1	1
MOC3	0	0	0	1

KP

$$KP_A = \sum_{i=1}^{n} \sum_{j=1}^{m_i} \prod_{k=1}^{l_j - 1} \frac{1}{BWDCit(P_{ijk})},$$

P_A를 연결한 endpoint들의 개수

Back-path에 해당하는 노드 수

P_A에서 endpoint(i)까지 가능한 backward path 수 (P_A이전 layer로부터 오는 건 제외)

↳ path 상의 노드로 들어온 back-cite 수

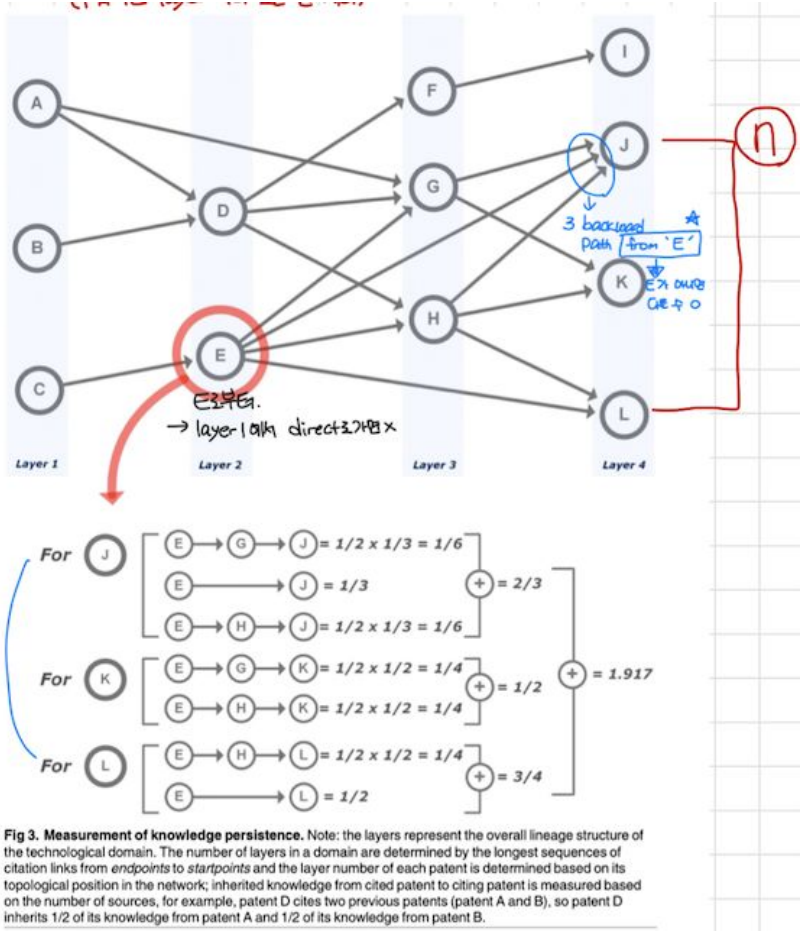


Fig 3. Measurement of knowledge persistence. Note: the layers represent the overall lineage structure of the technological domain. The number of layers in a domain are determined by the longest sequences of citation links from *endpoints* to *startpoints* and the layer number of each patent is determined based on its topological position in the network; inherited knowledge from cited patent to citing patent is measured based on the number of sources, for example, patent D cites two previous patents (patent A and B), so patent D inherits 1/2 of its knowledge from patent A and 1/2 of its knowledge from patent B.

KP

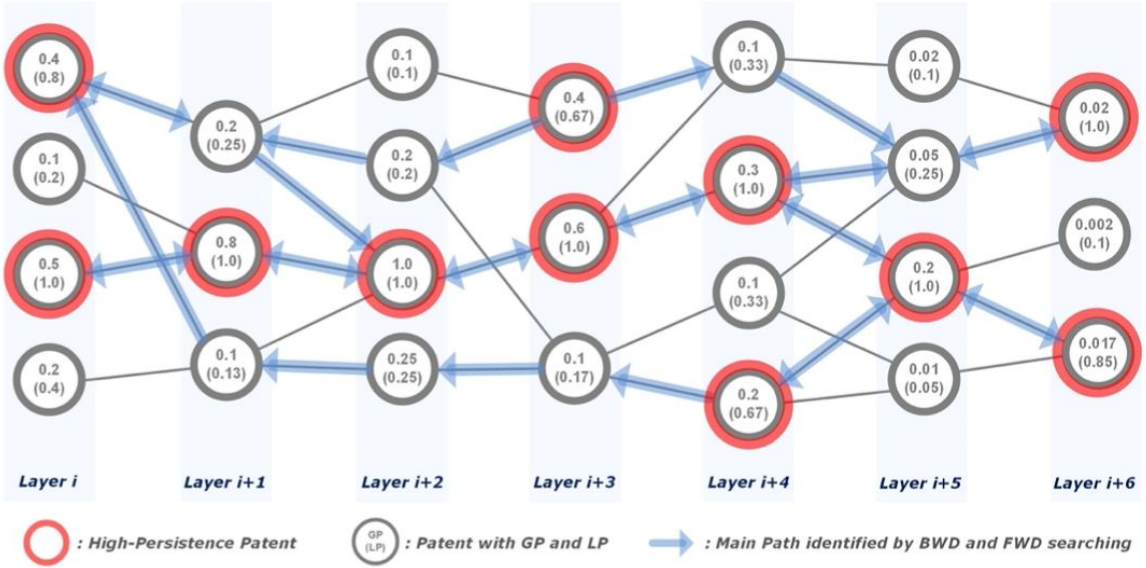


Fig 4. Searching backward and forward paths. Note: every HPP has both left and right arrows for backward and forward searches; HPP (0.2 GP and 1.0 LP) on layer *i+5* is directly connected with two HPPs on layer *i+4* and both links are chosen as main paths; if a HPP is not directly connected with other HPPs, e.g. the HPP (0.4 GP and 0.8 LP) on layer *i*, a patent which is not HPP but having the highest GP among the directly connected patents, e.g. the patent (0.2 GP and 0.25 LP) on layer *i+1*, is chosen and further searching is continued from that patent using the same algorithm.

1.HPP를 GP LP cutoff 0.3, 0.8 정도로 구한다.
2.HPP개수만큼 back-for search 수행
3.각 HPP가 directly 연결된 것 중 높은 p를 가진 것을 선택
4. 모든 HPP의 start, endpoint에 도달하면 멈춘다.