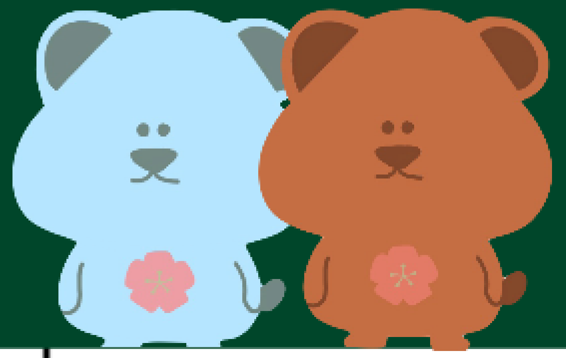
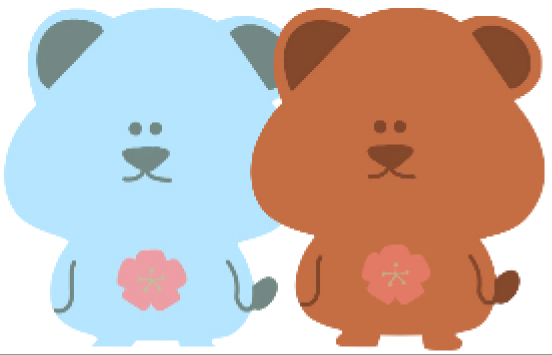


4주차 세션

DL팀 이다현, 최하경

목차

00. 목표



0.0 Graph as Matrix

이번 세션에서 배울 내용

- **Matrix** 관점에서 graph analysis와 learning을 해석해 다음을 공부함.

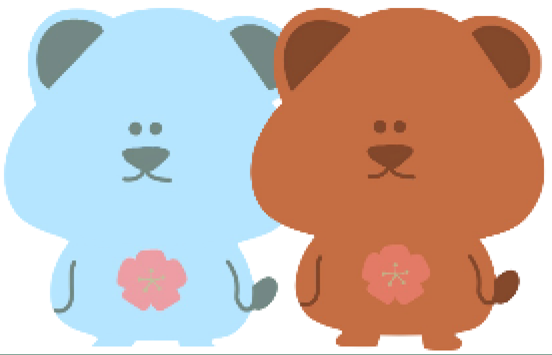
  ① random walk (PageRank)를 통해 node importance를 결정

  ② matrix factorization (MF)를 통해 node embedding을 얻음

  ③ Node2Vec 등 node embedding 방법들을 MF을 이용해 해석

- Random walk, Matrix factorization, Node embedding 사이의 관계

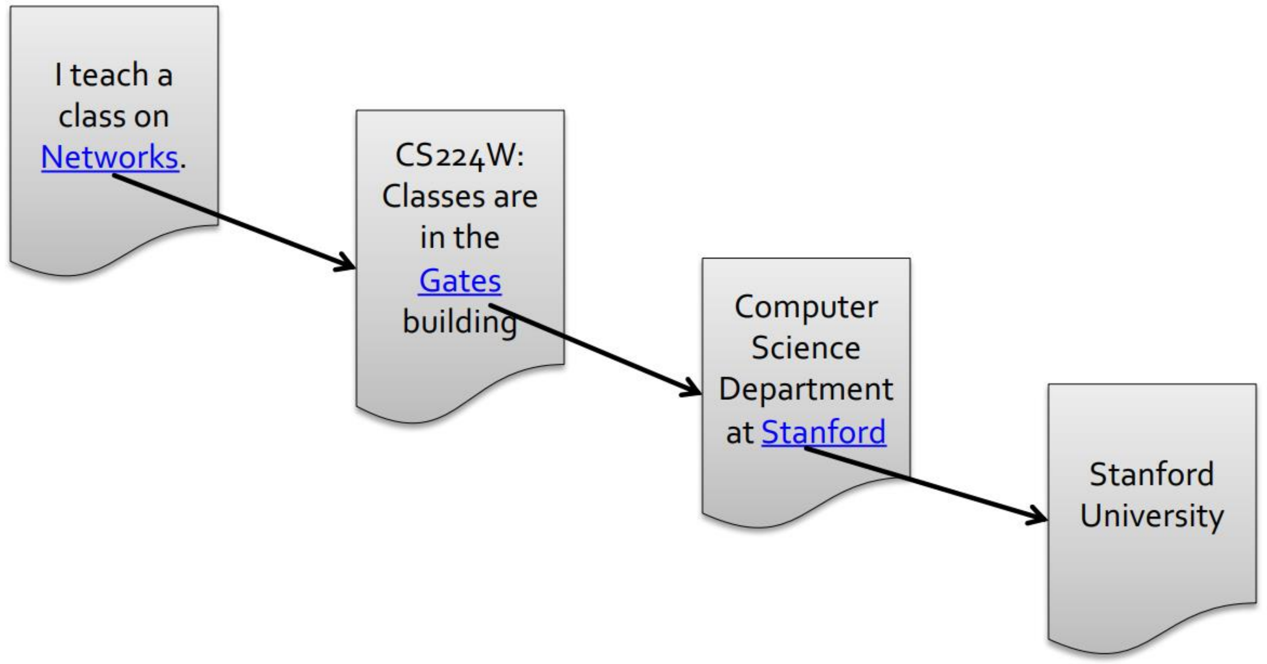
#01. PageRank



#1.1 The Web as a Graph

Web as a graph

- Nodes : web pages
- Edges : hyperlinks



인터넷 기술 발전에 따라 고려해야할 점

- Web pages
  - 동적인 웹페이지 존재
  - Database 상에서 접근 불가능한 page 존재

  ⇒ web을 이 page들이 만들어지기 이전의 상태로 취급

- Hyperlinks (web links)
  - 과거 : 페이지와 페이지 간 링크로만 존재 (navigational)
  - 현재 : 좋아요, 포스트, 댓글 등을 통한 연결 (transactional)

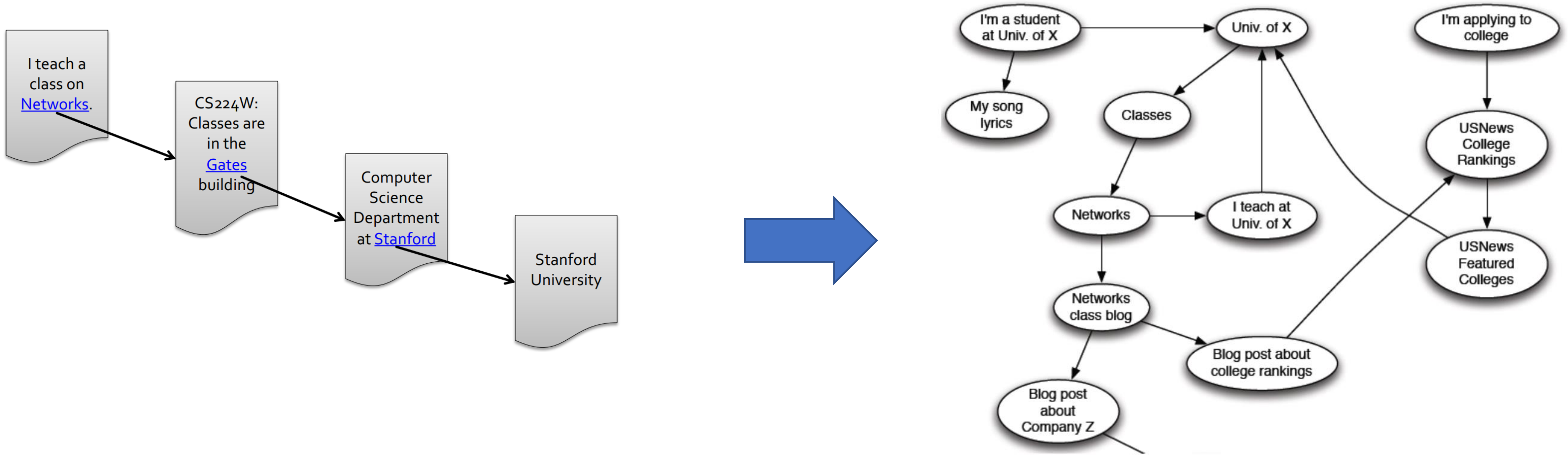
  ⇒ Navigational link만 취급

⇒ PageRank에서는 static web page들과 navigational set of link만을 고려함

#1.1 The Web as a Graph

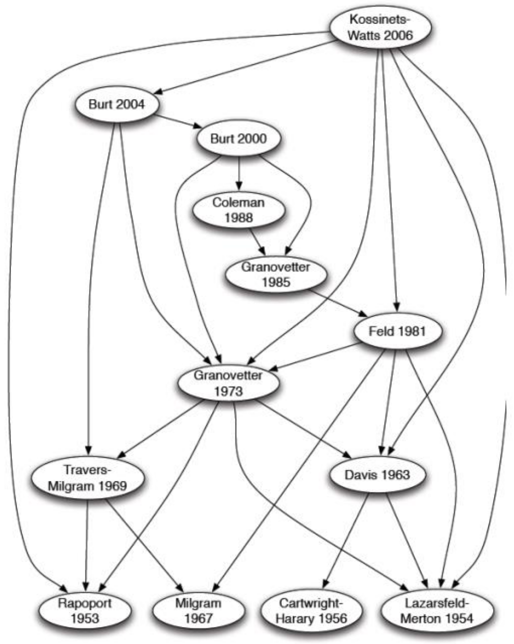
Web as a directed graph

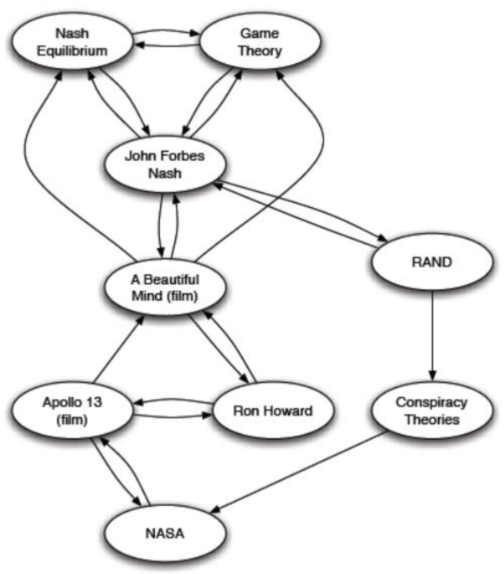
- Web에 static page들과 navigational hyperlink만 존재한다면, **directed graph**로 표현할 수 있음.



- web뿐만 아니라 다른 종류의 information도 graph로 표현 가능

  (ex) citation, references in an encyclopedia



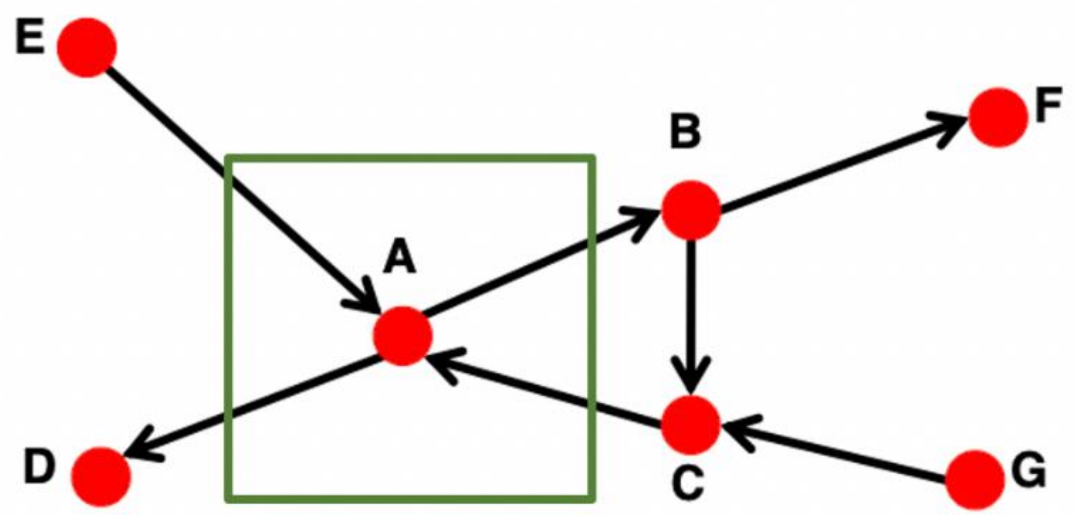


#1.1 The Web as a Graph

Web as a directed graph

- Directed graph로 표현된 web은 다음과 같다.

  - Edge : 해당 node에 대한 **in-link / out-link** 로 구성



A의 in-link: [C,E]
A의 out-link: [B,D]

- 새로운 질문
  - Web이 어떻게 생겼는지
  - 어떻게 web을 directed graph로 분석하고 해석하는지

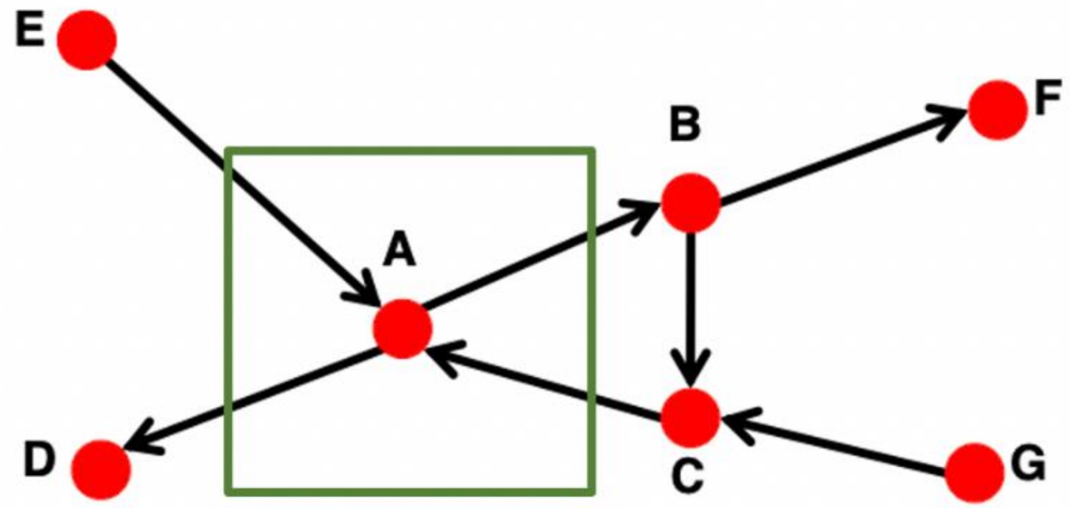
  ⇒ 이 질문들에 대한 대답들을 set of nodes based on vertices로 표현

  ⇒ in-link/out-link를 이용해 페이지 A와 연결된 페이지 확인

#1.1 The Web as a Graph

Web as a directed graph

- Directed graph로 표현된 web은 다음과 같다.
  - Edge : 해당 node에 대한 **in-link / out-link** 로 구성



- 새로운 질문
  - Web이 어떻게 생겼는지
  - 어떻게 web을 directed graph로 분석하고 해석하는지

  ⇒ 이 질문들에 대한 대답들을 set of nodes based on vertices로 표현

  ⇒ in-link/out-link를 이용해 페이지 A와 연결된 페이지 확인

But! 모든 web page (node)들이 동일한 importance를 가진 것이 아님!

⇒ Web graph link 구조를 이용해 **page들을 rank**한다. (여기서 PageRank 나옴)

#1.2. Ranking Nodes on the Graph

Ranking nodes on the graph

- 모든 web page (node)들이 **동일한 importance를 가진 것이 아님.**

  - unknown domain name이 우선순위가 더 낮고, 덜 신뢰적임 => page들에 따라 importance 달라짐
- **Web-graph node connectivity가 다양**해 어떻게 page에 순위를 매겨야 하는지 의문.

⇒ Web graph link 구조를 이용해 **page들을 rank**한다.

(웹 그래프 구조를 노드 간 우선순위에 대한 signal로 받아들임)

Link analysis algorithm

- Graph에서 node importance를 계산하는 방법

  - 아래 세 알고리즘은 base는 같고, 아래로 갈수록 extension version임.

  ① PageRank

  ② Personalized PageRank (PPR)

  ③ Random Walk with Restarts

#1.2. Ranking Nodes on the Graph

Main Idea : Links as Votes

- Link가 많을수록 해당 page는 더 중요하다.

  - in-link : 다른 page들이 해당 page를 link해야 하니까 fake하기 어려움.

  - out-link : 해당 page가 다른 page로 link하는데, 이때 page 유무와 관련없이 link를 "생성"만 하면 되기 때문에 fake하기 쉬움.

  ⇒ in-link를 node importance의 지표, 즉 vote으로 사용

- 이때, importance가 높은 page로부터 들어온 in-link일수록 더 많이 count해야 한다. (All in-links are not equal.)

  - 같은 수의 in-link더라도 더 중요한 page로부터 들어온 link의 개수가 많을수록 중요하다고 판단.

  - node importance가 해당 node의 in-link importance에 의존.

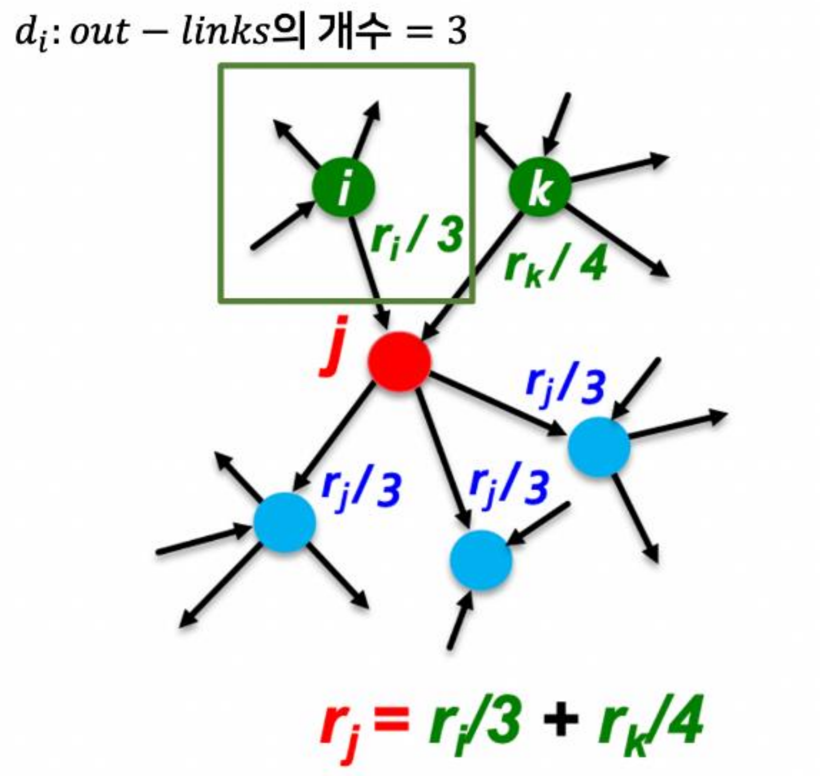
    ⇒ 모든 node는 해당 node로 link된 node들의 importance를 필요로 함

    ⇒ Recursive problem

#1.3. PageRank : The “Flow” Model

PageRank

- 계산 방법
  - 각 page의 link는 해당 page의 importance에 비례
  - page i가 importance *ri* 와 *di* out-links를 가질 때
    ⇒ 각 out-link = *ri* / *di* (vote)
  - page j의 importance *rj* = in-links의 vote의 합

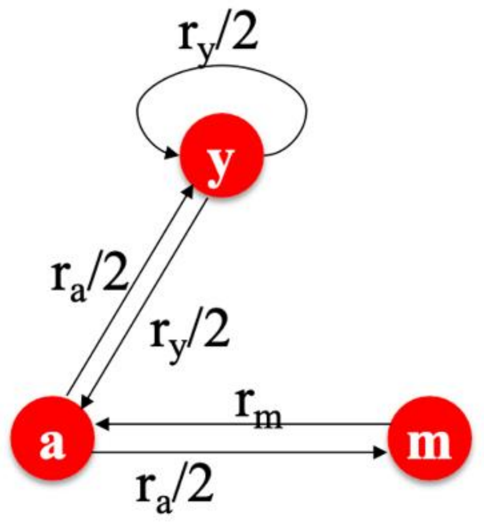


- Node j에 대한 rank *rj*

$$r_j = \sum_{i \to j} \frac{r_i}{d_i}$$

d_i ... **out-degree of node** i

- Flow equation



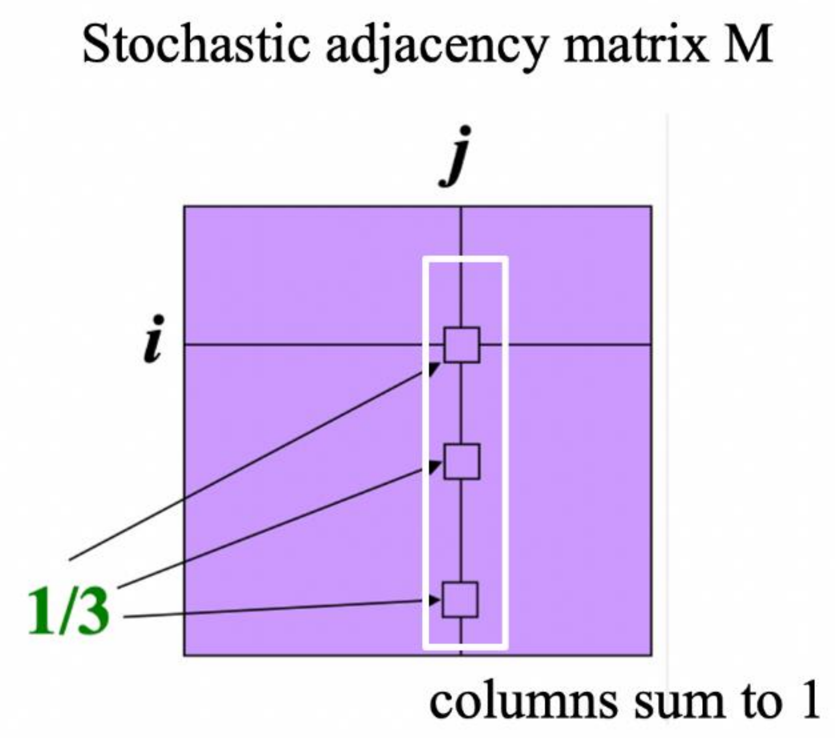
$$r_y = r_y/2 + r_a/2$$
$$r_a = r_y/2 + r_m$$
$$r_m = r_a/2$$

⇒ Gaussian elimination을 통해 풀 수 있지만,
그래프가 커질수록 실제로 활용하기엔 어려움.
⇒ Matrix Formulation 이용함.

#1.3. PageRank : The “Flow” Model

PageRank : Matrix Formulation

- Graph를 node와 edge의 조합이 아닌, **stochastic adjacency matrix로 표현**
  - page j가 dj개의 out-link가 있다면, Mij = 1/dj (page j에서 page i로 향하는 vote)
  - 각 column의 합은 1 (M ; column stochastic matrix)
  - 각 column은 page j의 neighboring node들에 대한 probability distribution



- Rank vector (r)
  - 각 page에 대한 importance (entry)
  - Page i에 대한 importance를 rj라고 할 때, rj의 합은 1

- Flow equation

① Matrix form

$$\boldsymbol{r} = \boldsymbol{M} \cdot \boldsymbol{r}$$

② equation

$$r_j = \sum_{i \to j} \frac{r_i}{d_i}$$

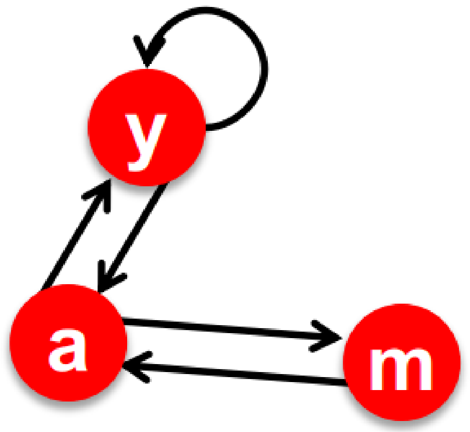
⇒ Matrix 형태와 equation 모두 동일한 의미

#1.3. PageRank : The "Flow" Model

PageRank : Matrix Formulation

- Example

① Equation



$$
\begin{aligned}
r_y &= r_y/2 + r_a/2 \\
r_a &= r_y/2 + r_m \\
r_m &= r_a/2
\end{aligned}
$$

② Matrix

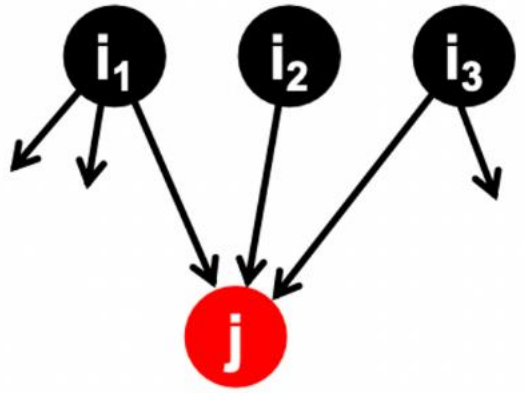
$$
\begin{array}{c|c|c|c|}
 & r_y & r_a & r_m \\
\hline
r_y & \frac{1}{2} & \frac{1}{2} & 0 \\
\hline
r_a & \frac{1}{2} & 0 & 1 \\
\hline
r_m & 0 & \frac{1}{2} & 0 \\
\hline
\end{array}
$$

$$
\underbrace{\begin{bmatrix} r_y \\ r_a \\ r_m \end{bmatrix}}_{r} = \underbrace{\begin{bmatrix} \frac{1}{2} & \frac{1}{2} & 0 \\ \frac{1}{2} & 0 & 1 \\ 0 & \frac{1}{2} & 0 \end{bmatrix}}_{M} \underbrace{\begin{bmatrix} r_y \\ r_a \\ r_m \end{bmatrix}}_{r}
$$

#1.4. Connection to Random Walk

Random web surfer

- Web page를 randomly navigate하는 사람 (intuition을 위해 random walk와 연결)
  - (t 시점) surfer이 page i에 위치
  - (t+1 시점) surfer은 page i의 out-link에 대해 랜덤하게 uniform한 확률로 이동
  - page j에 도달하는 과정을 반복

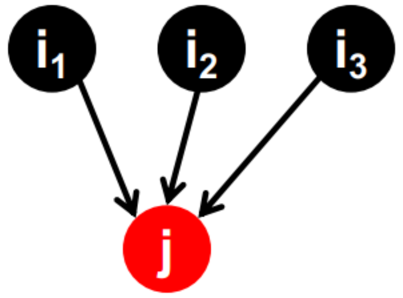


$$r_j = \sum_{i \to j} \frac{r_i}{\mathrm{d_{out}(i)}}$$

p(t)

- 전체 page 개수만큼의 벡터
- i번째 원소는 surfer이 t 시점에서 page i에 있을 확률
- 즉, page들에 대한 probability distribution을 의미함 (t 시점에서 surfer이 해당 page에 있을 확률)
- (t+1) 시점에서 surfer의 위치

$$\boldsymbol{p(t+1) = M \cdot p(t)}$$



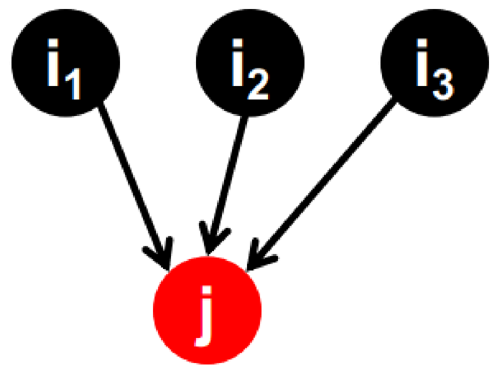
$$p(t+1) = \mathrm{M} \cdot p(t)$$

#1.4. Connection to Random Walk

Eigenvector centrality

- (t+1) 시점에서 surfer의 위치

이후 시점 $\boxed{\boldsymbol{p(t+1)}} = \boldsymbol{M} \cdot \boxed{\boldsymbol{p(t)}}$ 이전 시점



$$p(t+1) = \mathrm{M} \cdot p(t)$$

- Random walk가 steady state에 도달했을 때

$$\boldsymbol{p(t+1)} = \boldsymbol{M} \cdot \boldsymbol{p(t)} = \boldsymbol{p(t)}$$

- p(t)는 random walk에서 stationary distribution (확률이 정확하다면 시점이나 surfer의 위치에 따라 변화하지 않기 때문)

- Rank vector도 위와 비슷한 형태

$$\boxed{\boldsymbol{r}} = \boldsymbol{M} \cdot \boxed{\boldsymbol{r}}$$

이후 시점 이전 시점

⇒ rank vector r도 stationary distribution임

#1.5. Recall Eigenvector of a Matrix

Eigenvector centrality

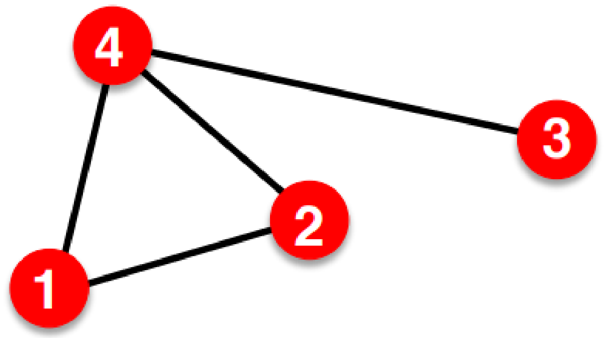
- A : adjacent matrix of undirected graph

$$A \in \{0, 1\}^{n \times n}$$

- Eigenvector of adjacency matrix

$$\boxed{\lambda} c = A \boxed{c}$$

eigenvalue eigenvector



$$A = \begin{pmatrix} 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{pmatrix}$$

Eigenvector Formulation

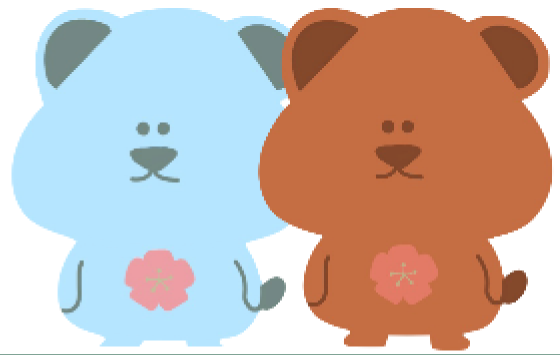
- Flow equation

$$\boxed{1} \cdot \boldsymbol{r} = \boldsymbol{M} \cdot \boxed{\boldsymbol{r}}$$

eigenvalue eigenvector

- Rank vector r : principal eigenvector (because eigenvalue=1)
  - 즉, importance가 eigenvector가 된다.
- 아무 벡터 u에서 시작했을 때
  - M(M(⋯(Mu))) 는 long-term distribution
  - Katz centrality
- Power iteration
  - Rank vector을 구하기 위한 방법

#02. How to solve?



#2.1 PageRank : How to solve?

👀 노드의 중요도 벡터 r 을 어떻게 구할까 → power iteration 으로!

$$\underset{\boldsymbol{r}}{\begin{bmatrix} r_y \\ r_a \\ r_m \end{bmatrix}} = \underset{\boldsymbol{M}}{\begin{bmatrix} ½ & ½ & 0 \\ ½ & 0 & 1 \\ 0 & ½ & 0 \end{bmatrix}} \underset{\boldsymbol{r}}{\begin{bmatrix} r_y \\ r_a \\ r_m \end{bmatrix}}$$

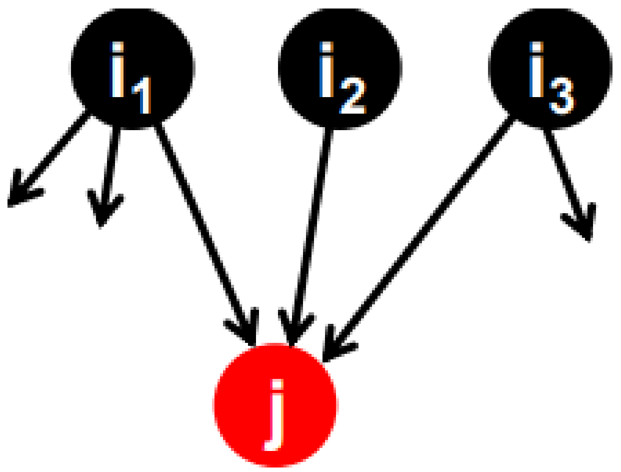
↪ 반복적인 실행을 통해 r 벡터의 값이 더 이상 변화가 없을 때까지
즉, 값이 수렴할 때까지 계산을 반복하자

$$\left(\sum_i \left|\boxed{r_i^{t+1} - r_i^t}\right| < \epsilon\right)$$

변화하는 값의 폭이 점차 작아질 때까지 = 수렴할 때까지

$$r_j^{(t+1)} = \sum_{i \to j} \frac{r_i^{(t)}}{d_i}$$

Out link



👉 (t+1) 시점에서의 노드 j 의 중요도 = SUM(t 시점에서 노드 i의 중요도/ 노드 i의 outlink 개수)

#2.2 Power Iteration Method

👀 N 개의 노드를 가진 web graph 에 대한 Power iteration 과정 예시

Power iteration: a simple iterative scheme

- 초기화 / 같은 중요도를 가지게 초기화 : 1/(노드의 개수)
  Initialize: $\boldsymbol{r}^{(0)} = [1/N, \ldots., 1/N]^T$
- 반복 / 동일한 수식
  Iterate: $\boldsymbol{r}^{(t+1)} = \boldsymbol{M} \cdot \boldsymbol{r}^{(t)}$ ⟷ $r_j^{(t+1)} = \sum_{i \to j} \frac{r_i^{(t)}}{d_i}$
- Stop when $|\boldsymbol{r}^{(t+1)} - \boldsymbol{r}^{(t)}|_1 < \varepsilon$

d_i …. out-degree of node i

L1 norm 이나 L2 norm 을 사용할 수 있다.

$|x|_1 = \sum_1^N |x_i|$ is the L₁ norm sum of absolute differences
Can use any other vector norm, e.g., Euclidean

↪ 50번 정도 반복하면 수렴한다.

#2.2 Power Iteration Method

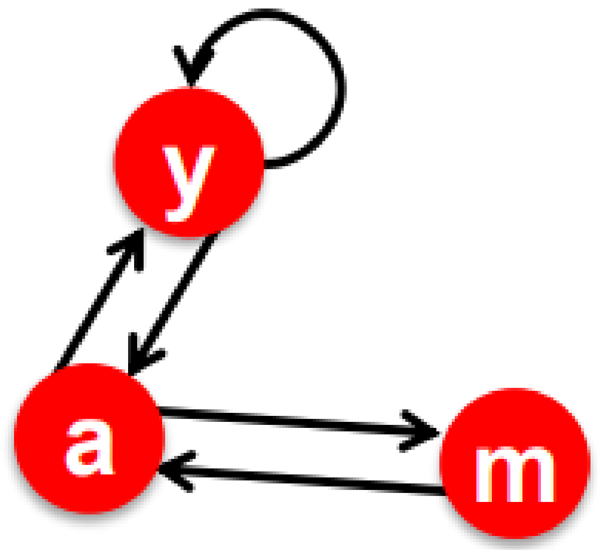
👀 iteration First : 초기값 설정 : 1/N

- **Power Iteration:**
  - Set $r_j \leftarrow \boxed{1/N}$
  - **1:** $r'_j \leftarrow \boxed{\sum_{i \to j} \frac{r_i}{d_i}}$
  - **2:** If $|r - r'| > \varepsilon$:
    - $r \leftarrow r'$
  - **3:** go to **1**
- **Example:**

$$\begin{bmatrix} r_y \\ r_a \\ r_m \end{bmatrix} = \begin{bmatrix} 1/3 \\ 1/3 \\ 1/3 \end{bmatrix}$$

초기값

Iteration 0, 1, 2,



	y	a	m
y	½	½	0
a	½	0	1
m	0	½	0

* d(y) = 2 : node y, node a
* d(a) = 2 : node y, node m
* d(m) = 1 : node a

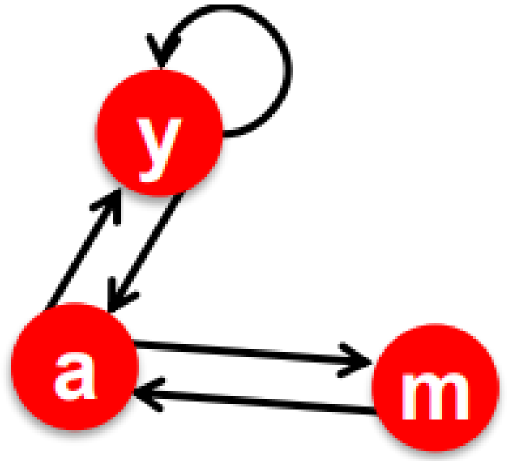
$$\begin{aligned} r_y &= r_y/2 + r_a/2 \\ r_a &= r_y/2 + r_m \\ r_m &= r_a/2 \end{aligned}$$

↪ node y = j 일때 노드 y 로 들어오는 link 의 노드는 y 자신과 노드 a 에 해당한다. i = {y,a}

#2.2 Power Iteration Method

👀 iteration Second

- **Power Iteration:**
  - Set $r_j \leftarrow 1/N$
  - **1:** $r'_j \leftarrow \sum_{i \to j} \frac{r_i}{d_i}$
  - **2:** If $|r - r'| > \varepsilon$:
    - $r \leftarrow r'$
  - **3:** go to **1**
- **Example:**



	y	a	m
y	½	½	0
a	½	0	1
m	0	½	0

* d(y) = 2 : node y, node a
* d(a) = 2 : node y, node m
* d(m) = 1 : node a

$$\begin{cases} r_y = r_y/2 + r_a/2 \\ r_a = r_y/2 + r_m \\ r_m = r_a/2 \end{cases}$$

* ry = 1/3/2 + 1/3/2 = 2/6 = 1/3
* ra = 1/3/2 + 1/3 = 3/6
* rm = 1/3/2 = 1/6

$$\begin{pmatrix} r_y \\ r_a \\ r_m \end{pmatrix} = \begin{pmatrix} 1/3 \\ 1/3 \\ 1/3 \end{pmatrix} \quad \begin{pmatrix} 1/3 \\ 3/6 \\ 1/6 \end{pmatrix}$$

Iteration 0, 1, 2, …

👀 수렴 결과

$$\begin{pmatrix} r_y \\ r_a \\ r_m \end{pmatrix} = \begin{pmatrix} 6/15 \\ 6/15 \\ 3/15 \end{pmatrix}$$

↪ 노드 중요도 순서
: y = a > m

#2.3 Three Questions

노드의 중요도 r 벡터를 구하는 방법

$$r_j^{(t+1)} = \sum_{i \to j} \frac{r_i^{(t)}}{d_i}$$

or
equivalently

$$r = Mr$$

두 식을 동일하게 볼 수 있을까

🙄 Questions

(1) 수렴을 보장하는가

(2) 수렴하는 지점이 정말 우리가 원하는 최적의 지점인가

(3) 노드 중요도의 순위 결과가 타당한가

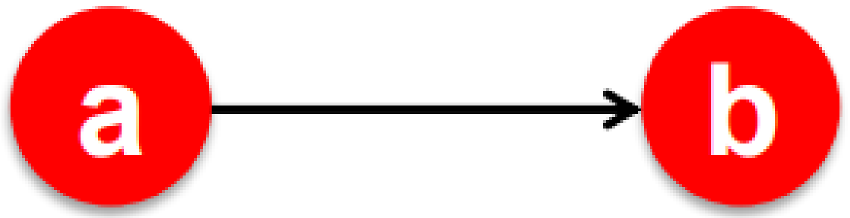
#2.4 Problems and solutions

😵 May be … 좀 문제가 있다

① 첫번째 문제 : Dead ends

밖으로 나가는 out link 가 없는 노드가
존재하여 중요도가 사라지는 문제

node b has NO out link



$$r_j^{(t+1)} = \sum_{i \to j} \frac{r_i^{(t)}}{d_i}$$

lost the importance

- **Example:**

Iteration: 0, 1, 2, 3…

$$\begin{array}{c} r_a \\ r_b \end{array} = \begin{array}{c|c|c|c} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{array}$$

→ converge all to ZERO

b 가 dead end 에 속하는 노드로 중요도가 더이상 흘러가지
않아 사라져 버린다. (r 이 더이상 확률분포가 되지 못함)

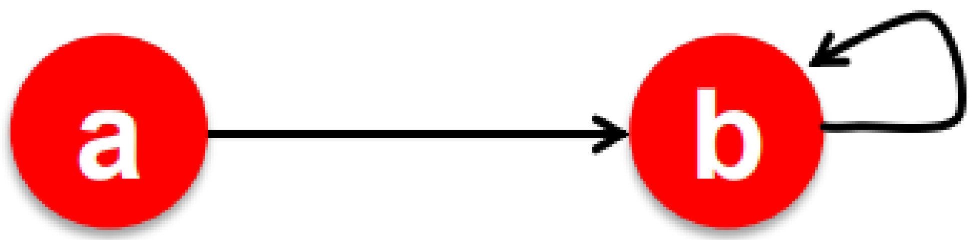
#2.4 Problems and solutions

😵 May be … 좀 문제가 있다

② 두번째 문제 : Spider trap

Out link 가 자기 자신으로 되돌아 오는 노드가 존재하여 중요도가 흐르지 못하고 특정 지점에 고이게 되는 문제

stuck in b forever



$$r_j^{(t+1)} = \sum_{i \to j} \frac{r_i^{(t)}}{d_i}$$

* a 의 중요도를 계속 b 로 보내어, a 의 중요도는 결국 0이 되버리고 b 는 중요도를 계속 받게 되어 1이 되는 결과를 가진다.

- **Example:**

Iteration: 0, 1, 2, 3…

		0	1	2	3
r_a	=	1	0	0	0
r_b		0	1	1	1

—〉 모든 중요도를 b 가 흡수

#2.4 Problems and solutions

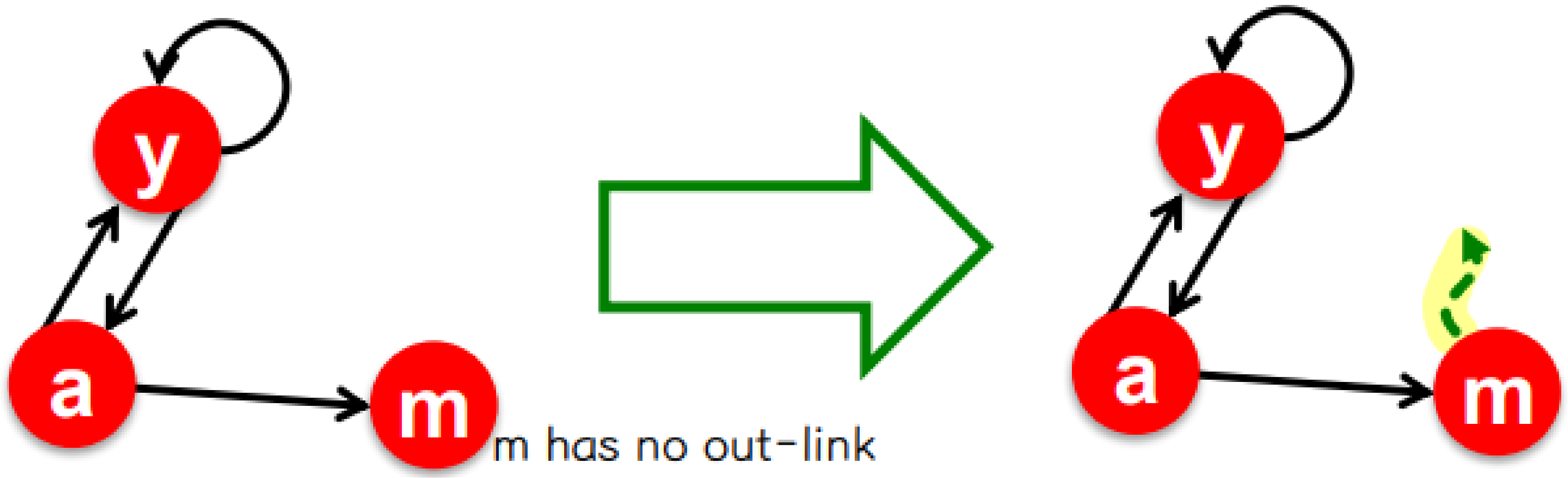
① Dead ends 문제 해결책

Teleports 랜덤하게 다른 페이지로 이동하게 하여 해결

순간이동

- Adjust matrix accordingly

dead end 에 도달하면 uniform distribution 에 따라 다른 노드로 중요도를 전달하게 된다. (무조건 다른 페이지로 이동하게 만든다)



m has no out-link

	y	a	m
y	½	½	0
a	½	0	0
m	0	½	0

randomly teleport wherever you want

	y	a	m
y	½	½	⅓
a	½	0	⅓
m	0	½	⅓

#2.4 Problems and solutions

② Spider traps 문제 해결책

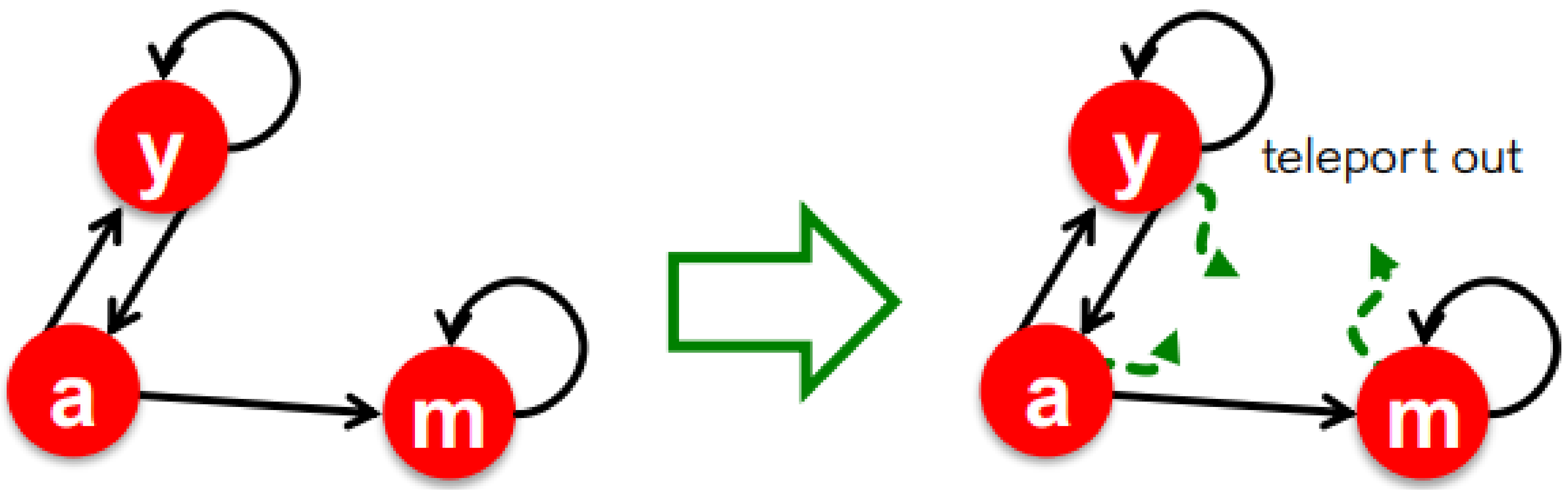
Random Surf

베타의 확률로 다른 링크를 랜덤하게 따라가거나
- With prob. β, follow a link at random

(1-베타) 의 확률로 랜덤한 페이지로 이동하게 하는 것
- With prob. **1-β**, jump to a random page (teleport)

※ 보통 β 는 0.8~0.9 사이의 값으로 지정한다.

중요도가 고이지 않고 노드 집합의 외부로 흐르게 만듦



#2.4 Problems and solutions

👀 정말 Teleport 와 같은 이동 방식이 해결책이 되는가?

① Spider traps

- Spider trap 은 중요도 r 벡터가 값이 leak out 사라지지는 않아 중요도가 그래프 어딘가로는 계속 흐르는 구조로, 확률분포를 유지하기 때문에 수학적으로 문제가 되진 않는다. 그러나 local 한 부분만 중요도가 1에 가까워지고 외부 노드는 중요도가 거의 0이 되기 때문에 현실적인 구조를 띄지는 못한다. (not what we want)

👉 teleporting 을 통해 값이 고이지 않도록 함으로써 해결

② Dead ends

- Dead ends 문제는 r 벡터가 0이 되는, 즉 더 이상 확률 분포를 띄지 않게 되는 구조를 가지므로 초기의 가정을 위반해 수학적으로 문제를 발생한다 (M is not column stochastic anymore)

👉 teleporting 을 통해 행렬의 열을 확률분포 꼴로 만들어 줌으로써 해결

#2.5 Random Teleports

- With probability β, follow a link at random (outlink at random)
- With probability $1-\beta$, jump to some random page

B 를 반영한 PageRank 공식

$$r_j = \underbrace{\sum_{i \to j} \beta \frac{r_i}{d_i}}_{\text{follow outlink}} + \underbrace{(1-\beta)\frac{1}{N}}_{\text{by teleporting}}$$

노드 j 의 중요도

Outlink node를 고려한 부분

Spider trap, dead end 문제를 해결하기 위해 추가한 부분 (teleport)

#2.5 Random Teleports

초록색 화살표 = teleport 로 생긴 확률

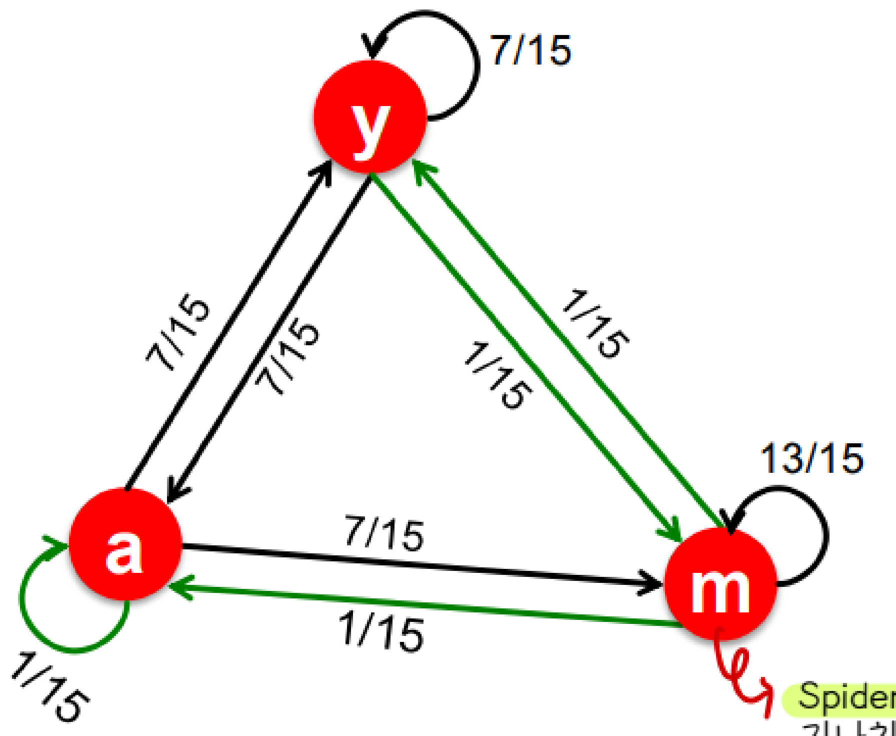
- **PageRank equation** [Brin-Page, '98]

$$r_j = \sum_{i \to j} \beta \frac{r_i}{d_i} + (1-\beta)\frac{1}{N}$$

- **The Google Matrix G:**

[1/N
whe

행렬식으로 변형 $$G = \beta\, M + (1-\beta)\left[\frac{1}{N}\right]_{N\times N}$$



$$
\begin{array}{cc}
 & \textbf{M} \\
\beta\ 0.8 & \begin{bmatrix} 1/2 & 1/2 & 0 \\ 1/2 & 0 & 0 \\ 0 & 1/2 & 1 \end{bmatrix}
\end{array}
+ 0.2
\begin{array}{c}
\textbf{[1/N]}_{NxN} \\
\begin{bmatrix} 1/3 & 1/3 & 1/3 \\ 1/3 & 1/3 & 1/3 \\ 1/3 & 1/3 & 1/3 \end{bmatrix}
\end{array}
$$

$$
\begin{array}{cc}
y & \\
a & \\
m &
\end{array}
\begin{bmatrix} 7/15 & 7/15 & 1/15 \\ 7/15 & 1/15 & 1/15 \\ 1/15 & 7/15 & 13/15 \end{bmatrix}
$$

G

Spider trap 으로 인해 노드 m 은 지나치게 중요도가 높게 평가됨

$$
\begin{array}{ccccccccc}
y & & 1/3 & 0.33 & 0.24 & 0.26 & & 7/33 \\
a & = & 1/3 & 0.20 & 0.20 & 0.18 & \ldots & 5/33 \\
m & & 1/3 & 0.46 & 0.52 & 0.56 & & 21/33
\end{array}
$$

=) 베타값을 높여서 텔레포트가 더 자주 일어나게 해 m 노드에서 중요도가 더 자주 빠져나갈 수 있게 할 수도 있다.

Converge

노드 m 이 중요노드

#2.6 PageRank Example

* 모든 노드가 중요도 값을 가진다

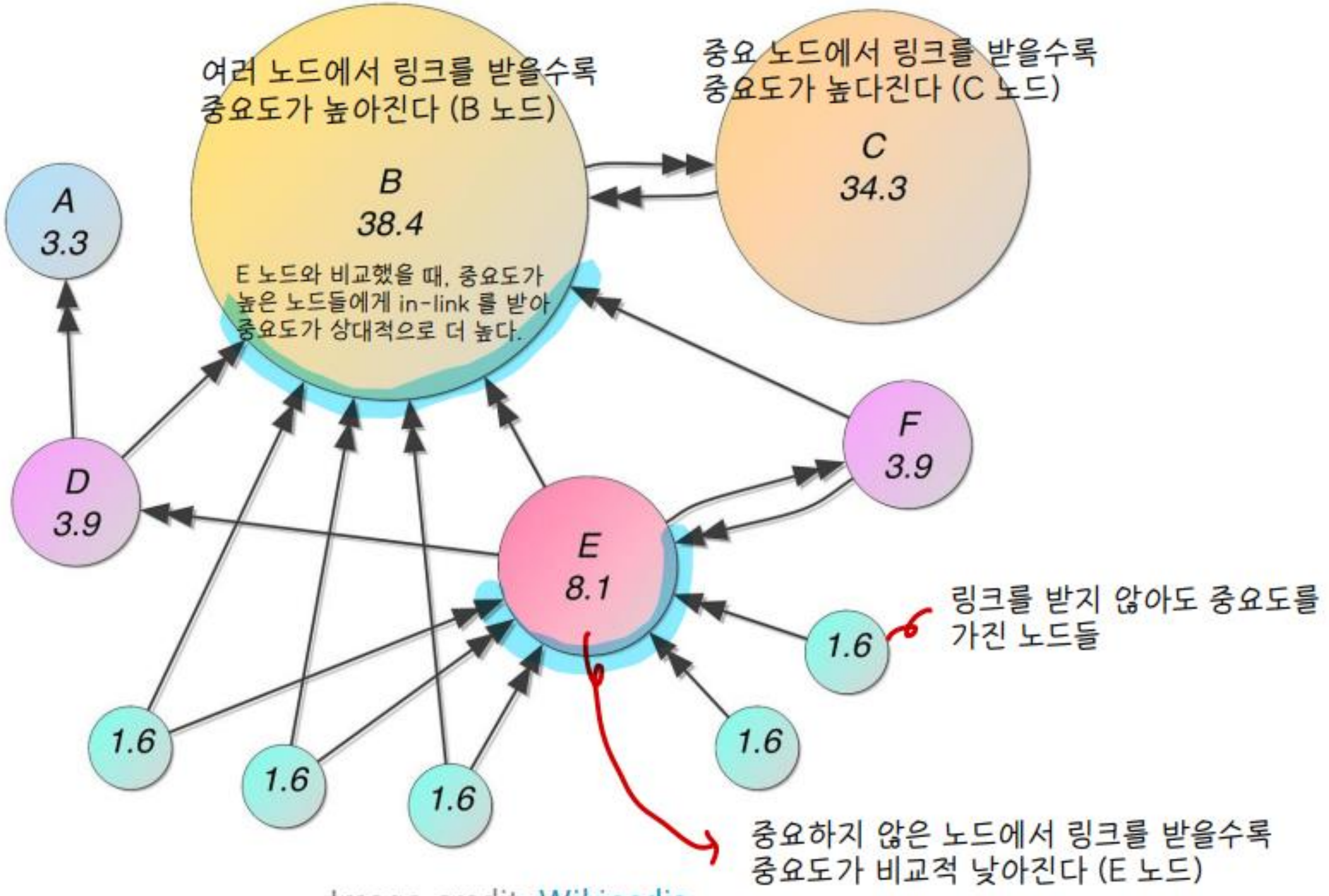


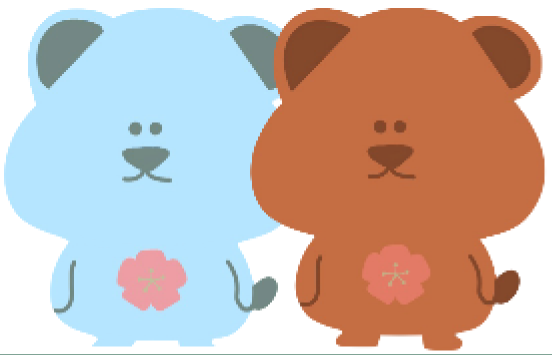
Image credit: Wikipedia

👀 정리

↪ PageRank 는 r=Gr 을 **power iteration** 계산 방법에 적용해 효율적으로 풀 수 있다.

↪ random uniform **teleportation** 을 추가함으로써 문제점들을 해결할 수 있다.

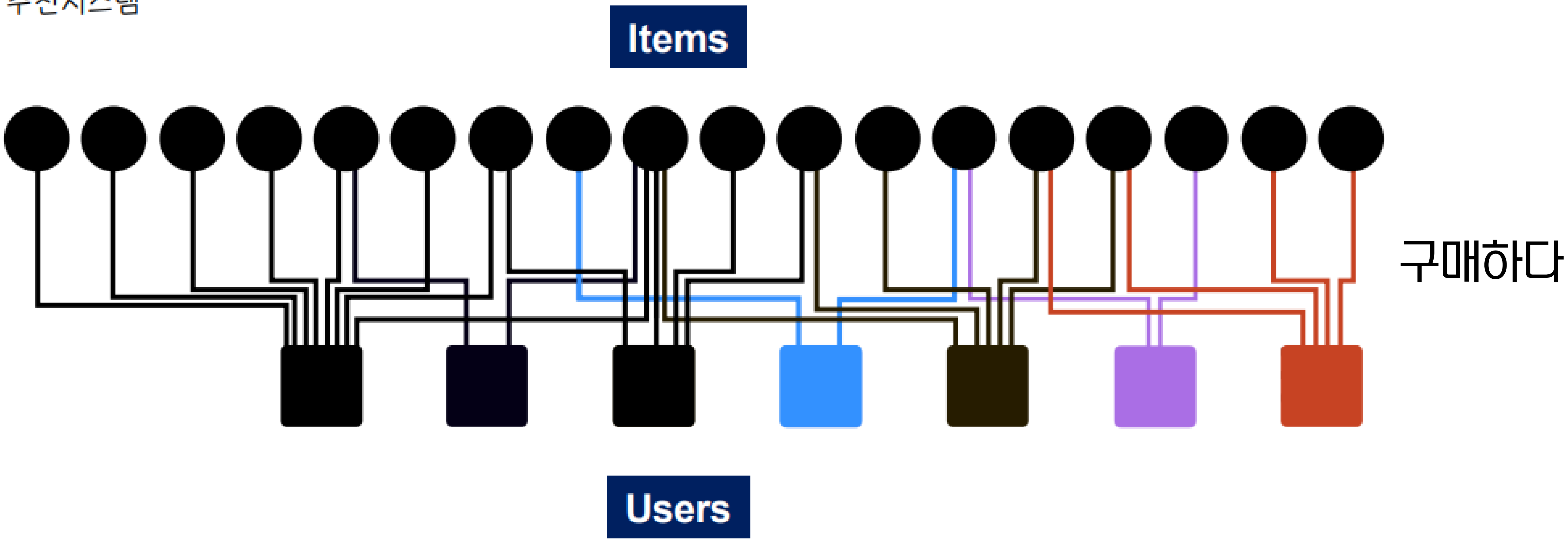
#03. PageRank 변형 알고리즘



#3.1 Recommendation

📌 Personalized PageRank ： 추천시스템을 예로 들어보자

추천시스템



↪ bipartite graph (1강 참고) 구조 = Item 들 끼리 (혹은 user 들 끼리) 의 연결성은 없음

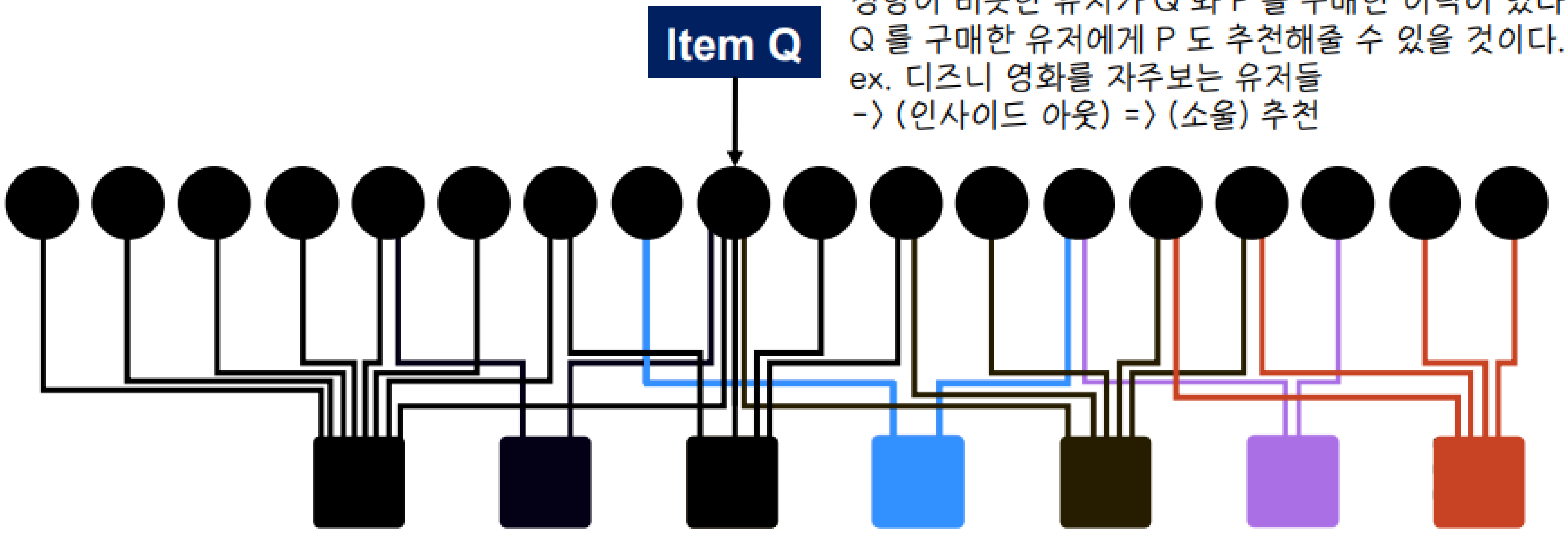
#3.1 Recommendation

Personalized PageRank : 추천시스템을 예로 들어보자

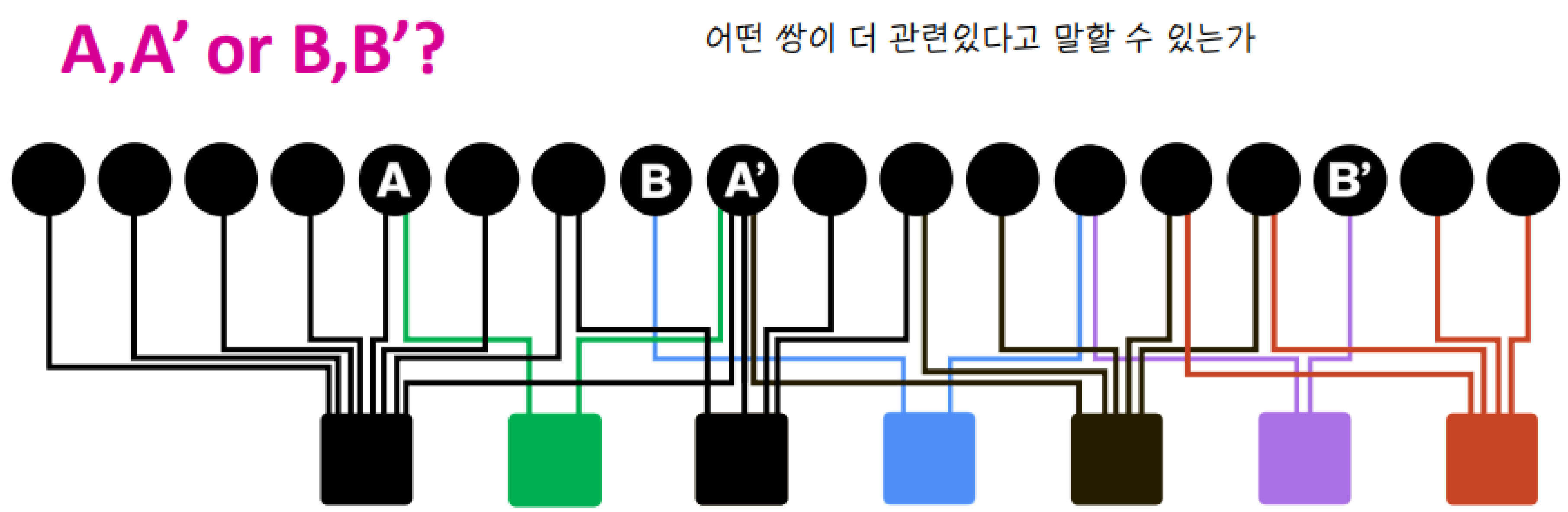
목표 : 그래프에서 노드 간 근접도 구하기 **Proximity**

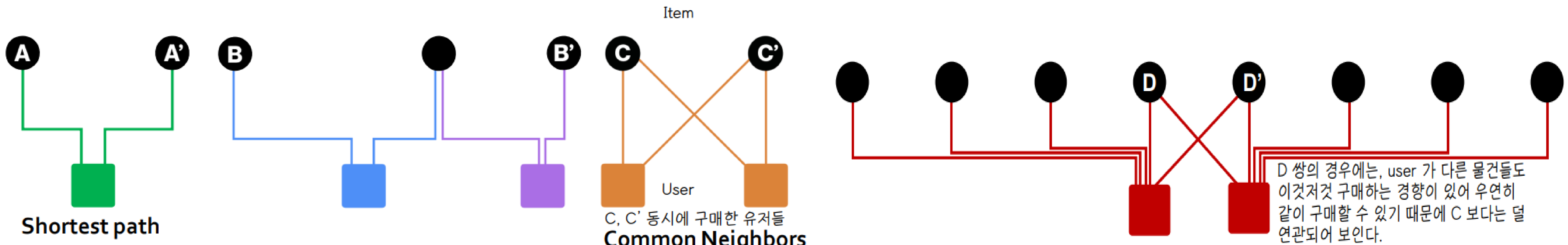
특정 아이템 Q 를 구매한 사용자에 대해 어떤 아이템을
추천해줄지 페이지 랭크 알고리즘으로 알려줌

성향이 비슷한 유저가 Q 와 P 를 구매한 이력이 있다면
Q 를 구매한 유저에게 P 도 추천해줄 수 있을 것이다.
ex. 디즈니 영화를 자주보는 유저들
-> (인사이드 아웃) => (소울) 추천



#3.2 Node Proximity measurements





Shortest path

Common Neighbors

Personalized Page Rank/Random Walk with Restarts

B 쌍이 가장 멀어보이고, A 가 중간, C 쌍이 가장 가까운 관계처럼 보인다.

#3.2 Node Proximity on Graphs

PageRank	Personalized PageRank	Random Walks with Restarts
• 중요도를 기반으로 노드의 순위를 매긴다. • Uniform 한 확률을 가지고 랜덤하게 Teleport 한다. • S = {all of the nodes of the network}	• 근접도를 기반으로 노드의 순위를 매긴다. • S = {interesting nodes} • 기존 pageRank 와 다른 것은 teleport 할 때 특정 노드 집단 S 로만 건너갈 수 있는 것	• teleport back to the starting node S • S = {Q} • 시작 노드로 돌아가는 것

#3.3 Node Proximity – Random Walk

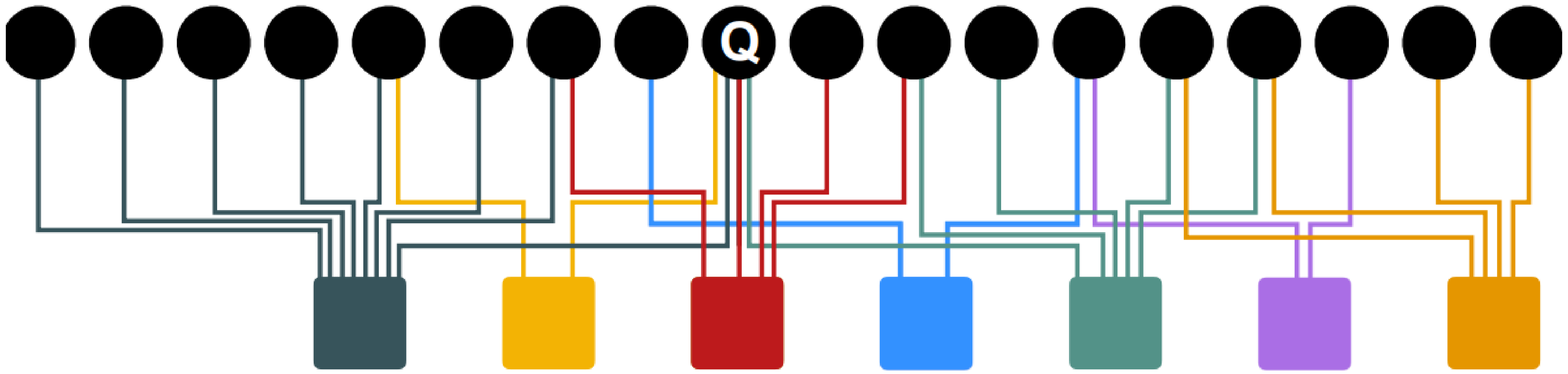
Personalized PageRank

↪ query node 집합이 주어졌을 때 (S)

- 임의의 이웃노드를 방문하여 count 를 기록한다
- 특정 확률을 가지고 집합 S 에 속한 노드 중 한 곳에서 다시 random walk 를 시작한다
- 가장 많이 방문한 노드는 가장 높은 근접도를 가진다고 볼 수 있다.

- Given a set of **QUERY NODES Q**, simulate a random walk:

Q 에서 시작해서, 임의의 노드를 방문하며 visit count 를 계산한다.



#3.3 Node Proximity – Random Walk

Personalized PageRank

- Proximity to query node(s) ***Q***: 수도코드

```
ALPHA = 0.5
QUERY_NODES = { Q }
```

```
item = QUERY_NODES.sample_by_weight( )
for i in range( N_STEPS ):
    user = item.get_random_neighbor( )      pick random user
    item = user.get_random_neighbor( )      pick random item
    item.visit_count += 1
    if random( ) < ALPHA:
        item = QUERY_NODES.sample.by_weight ( )
```

- 랜덤워크

Q 에서 다시 uniform 확률로 다른 노드로 jump

특정 확률값에 따라 restart 를 결정해 query node 로 back 하는 것을 결정

Random walk with Restarts

#3.3 Node Proximity – Random Walk

```
ALPHA = 0.5
QUERY_NODES = { Q }
```

랜덤워크 시점마다 초기값으로 돌아갈 확률

```
item = QUERY_NODES.sample_by_weight( )
for i in range( N_STEPS ):
    user = item.get_random_neighbor( )
    item = user.get_random_neighbor( )
    item.visit_count += 1
    if random( ) < ALPHA:
        item = QUERY_NODES.sample.by_weight ( )
```

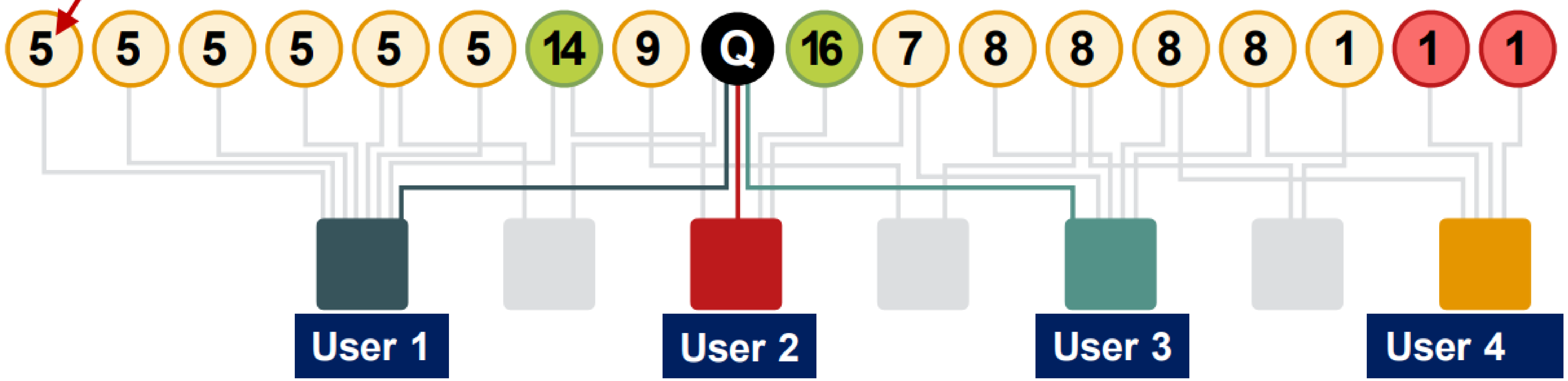
visit count

Number of visits by random walks starting at Q

random walker will visit more often
= more common neighbor
= more paths —> proximity

Query Item Q

더 가까운 관계의 노드라면 높은 방문 숫자를 가진다.



(1) Multiple connections
(2) Mutiple paths
(3) Direct and indirect connections
(4) Degree of the node

☞ 시작 노드와 특정 노드 간에 엣지가 많거나, 경로가 많거나, 직접 연결되어 있거나, 차수가 낮은 user 로 연결되어 있다면 랜덤워크에서 많이 방문하게 되고 유사도가 높아질 수밖에 없다.

#3.4 Summary

① PageRank

- Teleports to any node
- Nodes can have the same probability of the surfer landing:

10 개 노드 예시

$$S = [0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1]$$

② Personalized PageRank

- Teleports to a specific set of nodes 정해진 노드 집합에만 가중치가 반영된 분포로 이동
- Nodes can have different probabilities of the surfer landing there:

Bipartite graph 를 갖는
추천시스템 그래프 구조와 같이

50% 는 해당 노드로 이동할 가능성이 높음

$$S = [0.1, 0, 0, 0.2, 0, 0, 0.5, 0, 0, 0.2]$$

이동할 확률이 0인 노드도 존재 (personalized)

③ Random Walk with Restarts

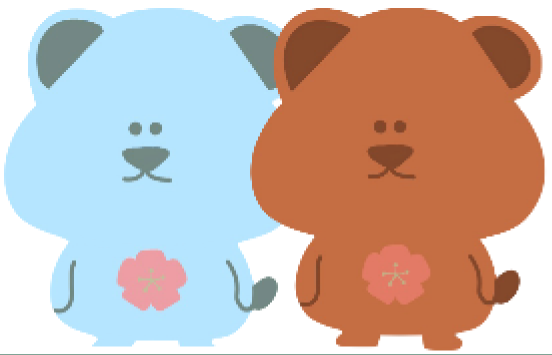
Topic-Specific PageRank where teleport is always to the same node:

restart 에 해당하는 열벡터의 확률이 1

$$S = [0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0]$$

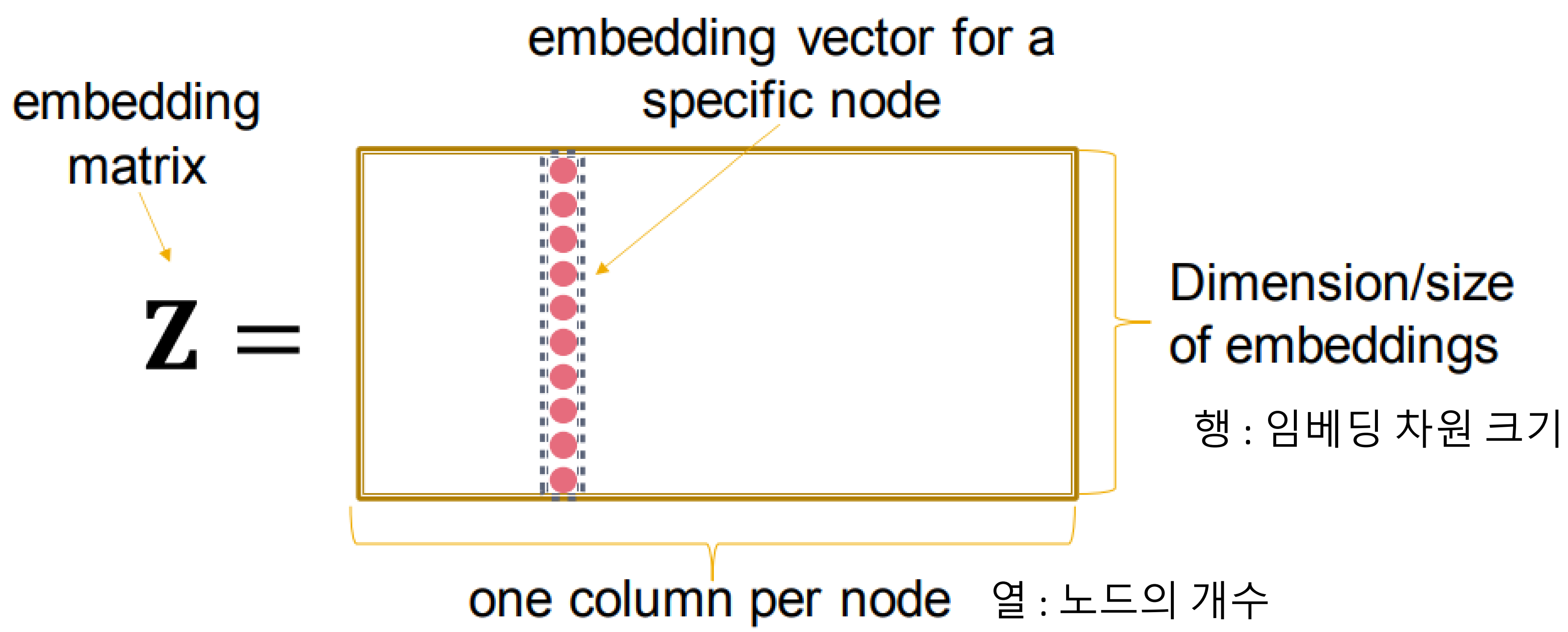
single node

#04. 행렬분해와 노드 임베딩



EWHA EURON

#4.1 Embeddings & matrix factorization



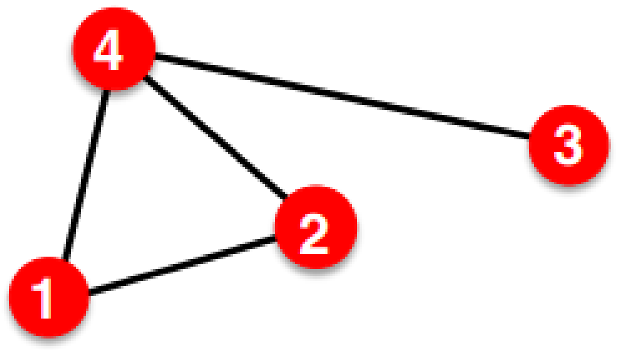
Objective: maximize $\mathbf{z}_v^{\mathrm{T}} \mathbf{z}_u$

↪ 비슷한 노드라면 내적 값이 최대가 되도록 목적함수를 구성한다.

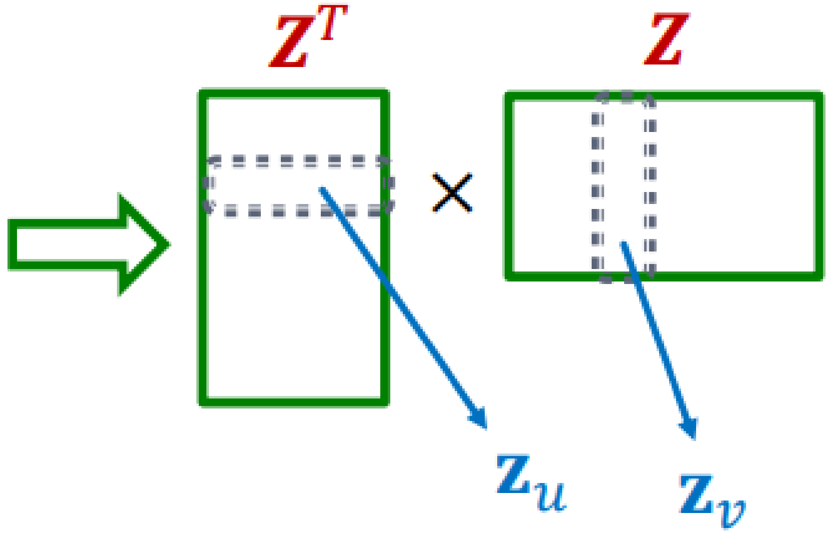
💡 비슷한 노드를 단순하게 정의한다면, 두 노드가 엣지로 연결되어 있을 때, 비슷한 노드라고 할 수 있다

→ 인접행렬에서 element 값이 1일때

- Therefore, $\underbrace{Z^T}_{1\times4} \underbrace{Z}_{4\times1} = \underbrace{A}_{1\times1}$ 인접행렬 A 가 임베딩 행렬 Z 로 분해됨

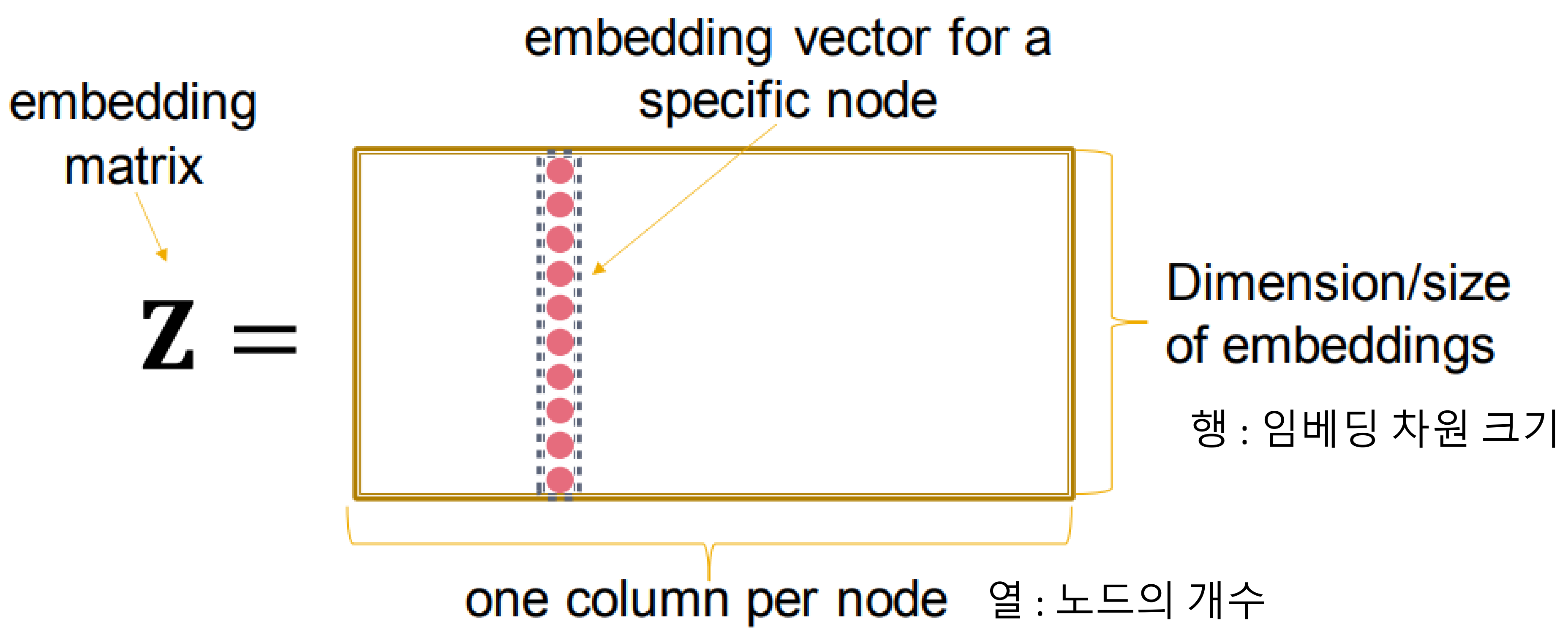


$$A = \begin{pmatrix} 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & \boxed{1} \\ 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{pmatrix}$$



📌 노드 임베딩도 행렬 분해 방식으로 접근할 수 있게 된다.

#4.1 Embeddings & matrix factorization



☞ 임베딩 벡터의 차원 d 가 그래프 정보를 담기에는 부족할 수밖에 없기에 (d < N) 정확한 행렬 분해는 불가능하다. 따라서 Z 를 근사시키어 행렬을 분해한다.

Objective: $\min_{Z} \| A - Z^T Z \|_2$ 근사시켜서 찾음

Objective: maximize $\mathbf{z}_v^{\mathrm{T}} \mathbf{z}_u$ $\qquad A = Z^T Z$

↪ Edge connectivity 로 정의할 수 있는 노드 벡터 간의 내적 연산은 인접행렬 A 의 행렬분해 결과와 같다.

#4.2 Random Walk based Similarity

랜덤워크 기반의 노드 임베딩 기법인 DeepWalk 와 node2vec 도 행렬 분해로 접근할 수 있다.

Volume of graph

$$vol(G) = \sum_i \sum_j A_{i,j}$$

transformation of matrix form

(node degree)

Diagonal matrix D

$$D_{u,u} = \deg(u)$$

$$\log\left(vol(G)\left(\frac{1}{T}\sum_{r=1}^{T}(D^{-1}A)^r\right)D^{-1}\right) - \log b$$

length of the random walks

context window size

See Lec 3 slide 30:

$T = |N_R(u)|$

Power of normalized adjacency matrix

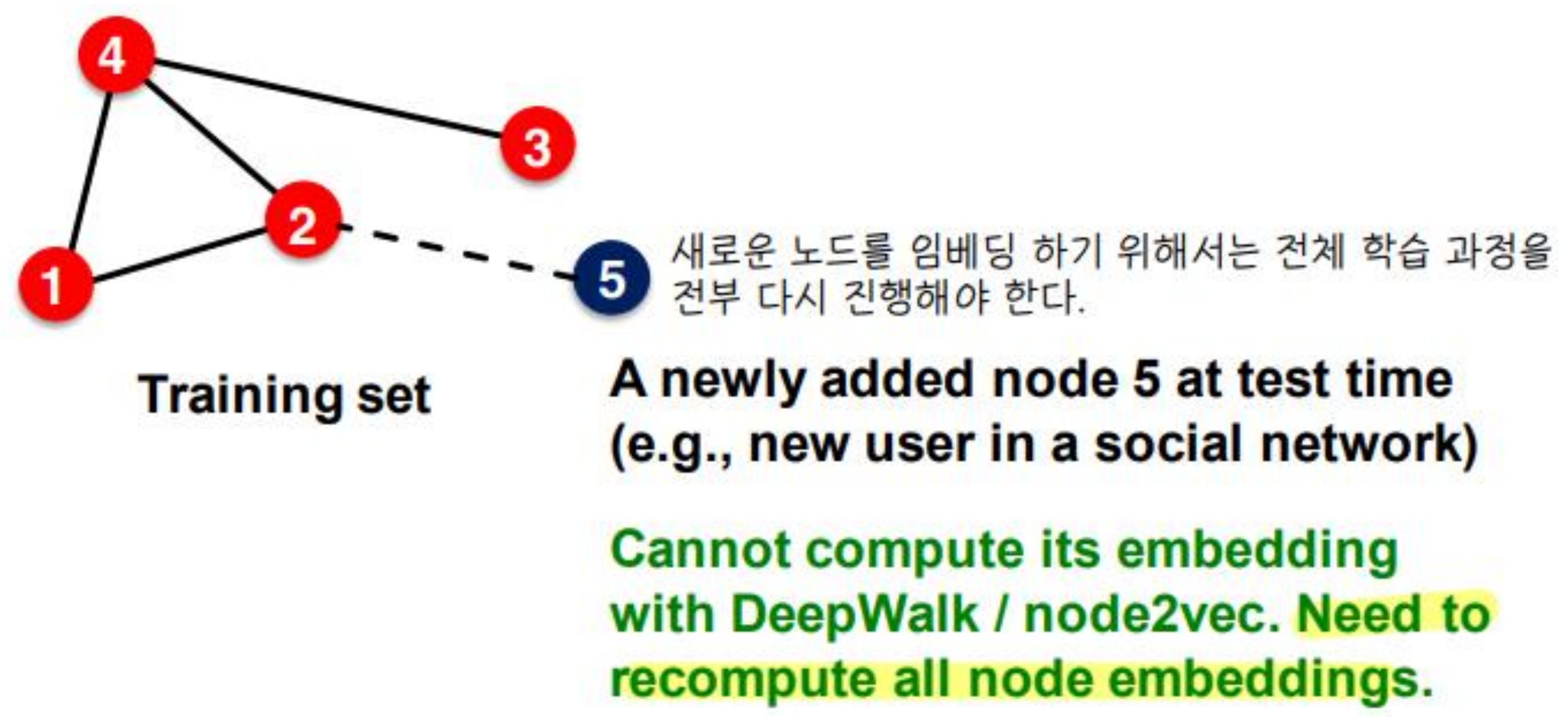
Number of negative samples

랜덤워크에 기반한 방식도 행렬분해로 풀 수 있다.

#4.3 Limitations

💡 랜덤워크 방식에 기반한 노드, 엣지, 그래프 임베딩 방법론들의 한계점

① 훈련시 학습되지 않은 노드의 임베딩 벡터를 만들 수 없다. 랜덤워크 기반의 방법들은 BoW 의 개념을 차용하고 있어 동일한 한계점을 가진다.



#4.3 Limitations

💡 랜덤워크 방식에 기반한 노드, 엣지, 그래프 임베딩 방법론들의 한계점

② 구조적 유사성을 잘 잡아내지 못한다.

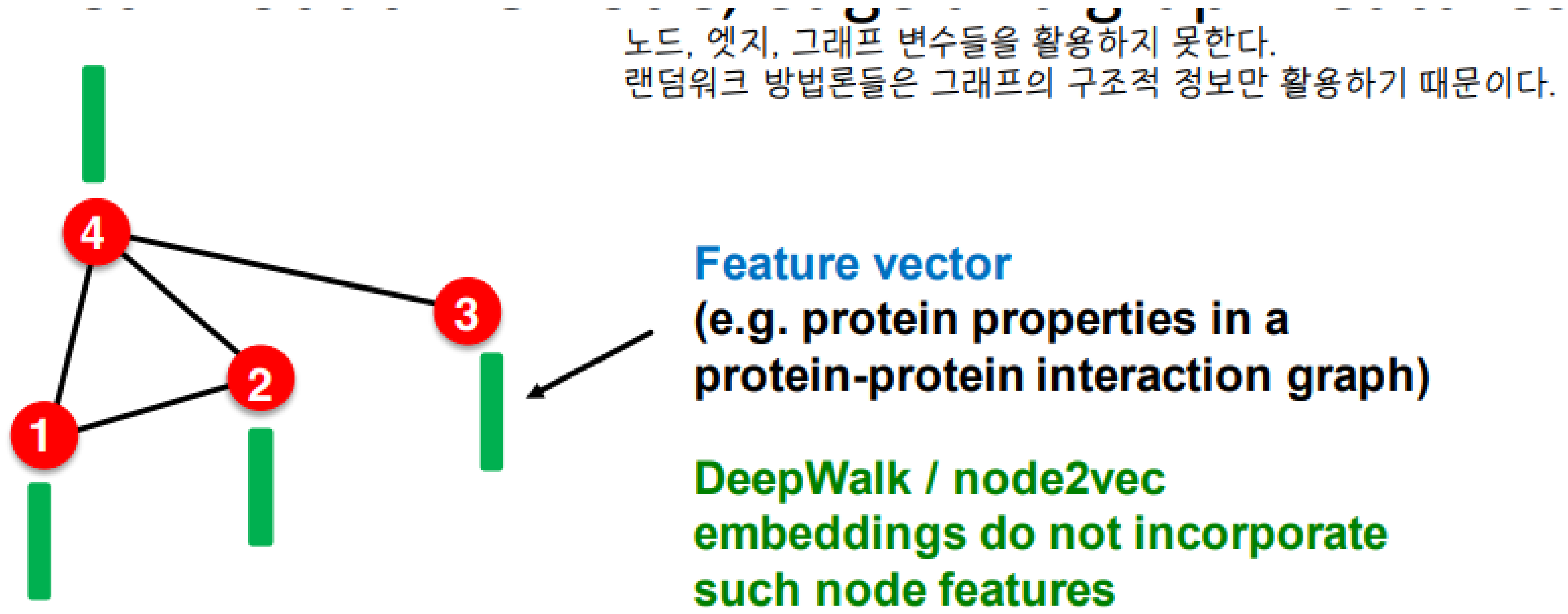
1번 노드와 11번 노드는 구조적으로 유사하다

- Node 1 and 11 are **structurally similar** – part of one triangle, degree 2, … 그러나 매우 다른 임베딩 값을 가진다.
- However, they have very **different** embeddings.
  - It's unlikely that a random walk will reach node 11 from node 1. (deepwalk 나 node2vec 은 인접 노드의 임베딩 벡터를 이용해 최적화 하기 때문)

#4.3 Limitations

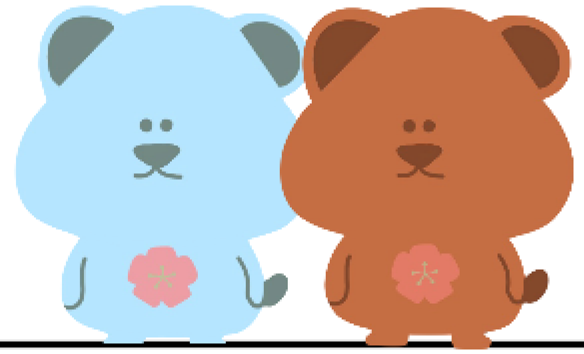
💡 랜덤워크 방식에 기반한 노드, 엣지, 그래프 임베딩 방법론들의 한계점

③ feature 들을 활용하지 못한다. 랜덤워크 방법론들은 그래프의 구조적인 정보만 활용하기 때문



이러한 한계점들의 해결 방법 : Deep representation learning & GNN (다음시간에 배움…)

THANK YOU



EWHA EURON