

대용량 그래프 데이터 처리 기술

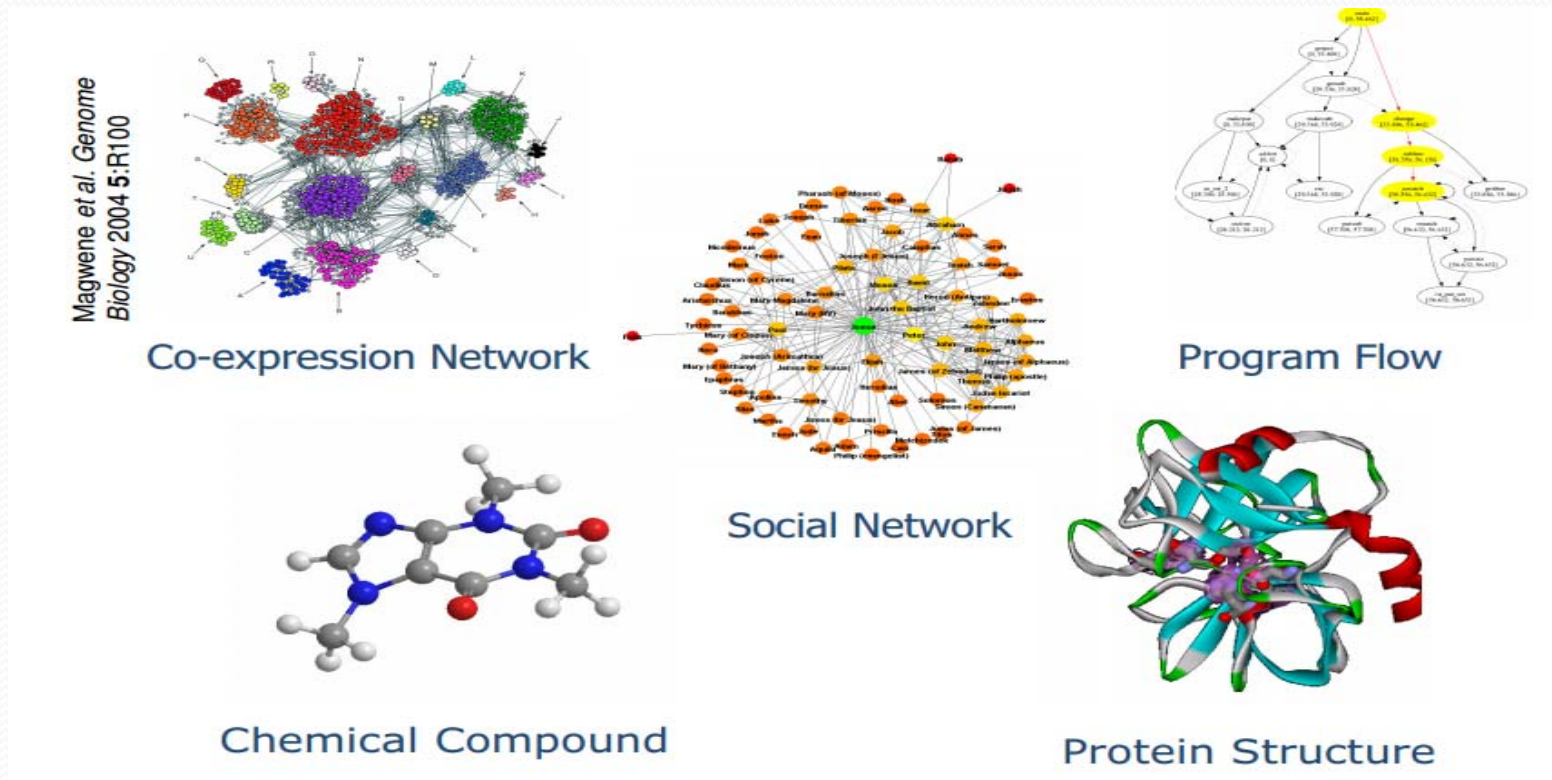
한 욱 신 (경북대학교)

Outline

- Overview
- Graph Database Management Systems
- Social and Information Network DBMS (TurboDB)
- An In-depth Comparison of Subgraph Isomorphism Algorithms in Graph Databases

Overview

- 그래프는 많은 영역에서 유용한 데이터 표현 방법으로 활용되고 있음



Xifeng, "Graph Mining and Graph Kernel," ACM SIGKDD 2008.

Overview

- 최근 소셜 네트워크 서비스의 활용도가 증가하고 있으며, 서비스 데이터의 규모가 폭발적으로 증가
 - 페이스북의 경우, 이용자 수가 9억 6천만 명(현재), 20분마다 약 100만개의 링크 공유, 271만개의 사진 upload, 약 1000만개의 댓글 작성
 - 트위터의 경우, 이용자 수가 5억 명(2011년 2월)을 넘어섰으며, 1년 6개월 후에는 10억 명을 돌파할 것으로 예상
 - 우리나라의 경우, 총 인구의 10%이상인 약 540만 명이 소셜 네트워크 서비스를 이용(2011년 11월)중이며, 그 수가 2010년에 비해 8.6배 증가
- 방대한 규모의 그래프 데이터를 효과적으로 저장/검색/관리하는 그래프 데이터베이스 관리 시스템(DBMS)의 필요성이 증가

Graph Database Management Systems

- 정의
 - 노드, 에지, 및 프로퍼티를 가진 그래프 구조를 저장하는 스토리지 시스템
- 유용한 연산자
 - Search (Bread First Search, Depth First Search)
 - Graph 구조 및 영향력 분석 (Cascade, Centrality, Finding and Detecting Community)
 - Graph 구성원 및 신뢰도 측정, 관리, 예측 (PageRank)
 - 동적 graph의 mining (Netevolve)
 - Subgraph Isomorphism
 - ...

Graph Database Management Systems

- 시스템 연구 방향
 - 순수 그래프를 처리하는 시스템
 - Neo4J, sones GraphDB, ...
 - 대용량 데이터를 처리하는 시스템
 - MapReduce, Hadoop, ...
 - 특정 그래프 연산 처리에 특화된 시스템
 - Specialized systems for pagerank, ...

순수 그래프를 처리하는 시스템

- 개요
 - 그래프로 모델링 되는 데이터를 저장/검색하고 관리하는 시스템
 - 일반적인 그래프를 다루는 시스템과 그래프로 표현된 RDF 데이터를 다루는 시스템으로 구분
 - 대용량 데이터의 처리하는데 효율적이지 못함
- 대표적인 시스템
 - 일반적인 그래프 데이터를 다루는 시스템: Neo4J(neo technology), sones GraphDB(sones), ...
 - 그래프로 표현된 RDF 데이터를 다루는 시스템: Allegrograph(Franz), Jena(Apache), ...

대용량 데이터를 처리하는 시스템

- 개요
 - 분산 컴퓨팅을 지원하는 소프트웨어 프레임워크 (MapReduce 등) 를 기반으로 대규모의 데이터를 분산 저장/검색하고 관리하는 시스템
 - 높은 안정성 및 확장성을 제공
 - 질의 형태가 단순하여 다양한 종류의 연산자 지원이 필요한 그래프 데이터 마이닝에 적합하지 못함
- 대표적인 시스템
 - Hadoop(Yahoo), ...
 - 분산 프레임워크 기반의 그래프 데이터 처리 시스템: Pregel(Google), GBase(CMU), Trinity(Microsoft), ...

Existing System들간의 비교

	Native Graph	Online Query	Index	Transaction Support	Parallel Graph Processing	Topology in memory	Distributed
Neo4J	Y	Y	Lucene	Y	N	N	N
Hyper GraphDB	N	Y	Built-in	Y	N	N	N
Infinite DB	N	Y	Built-in + Lucene	N	N	N	Y
Pregel	N	N	N	N	Y	N	Y
Trinity	N	Y	Built-in	Y	Y	Y	Y
...							

Haixun Wang, "Managing and Mining Billion Node Graphs," APWeb 2012.

Neo4J

- 개요
 - Java로 구현된 open-source 그래프 DBMS
- 특징
 - Graph model이 entity-relationship model과 match됨
 - Graph는 Node (vertex), Relationship (edge), 및 Property (Node와 Relation의 attribute)로 구성
 - 하부 시스템으로 Disk-based, native graph storage manager를 가짐
 - Apache Lucene Index를 사용하여 full text indexing 지원
 - Transaction 지원
 - Single-machine에서 billion nodes/relations/properties를 지원

Data Representation in Neo4J

- 개요
 - Node와 relationship 각각을 데이터 구조로 표현
 - Property는 <key, value>의 pair로 표현
- 데이터 구조

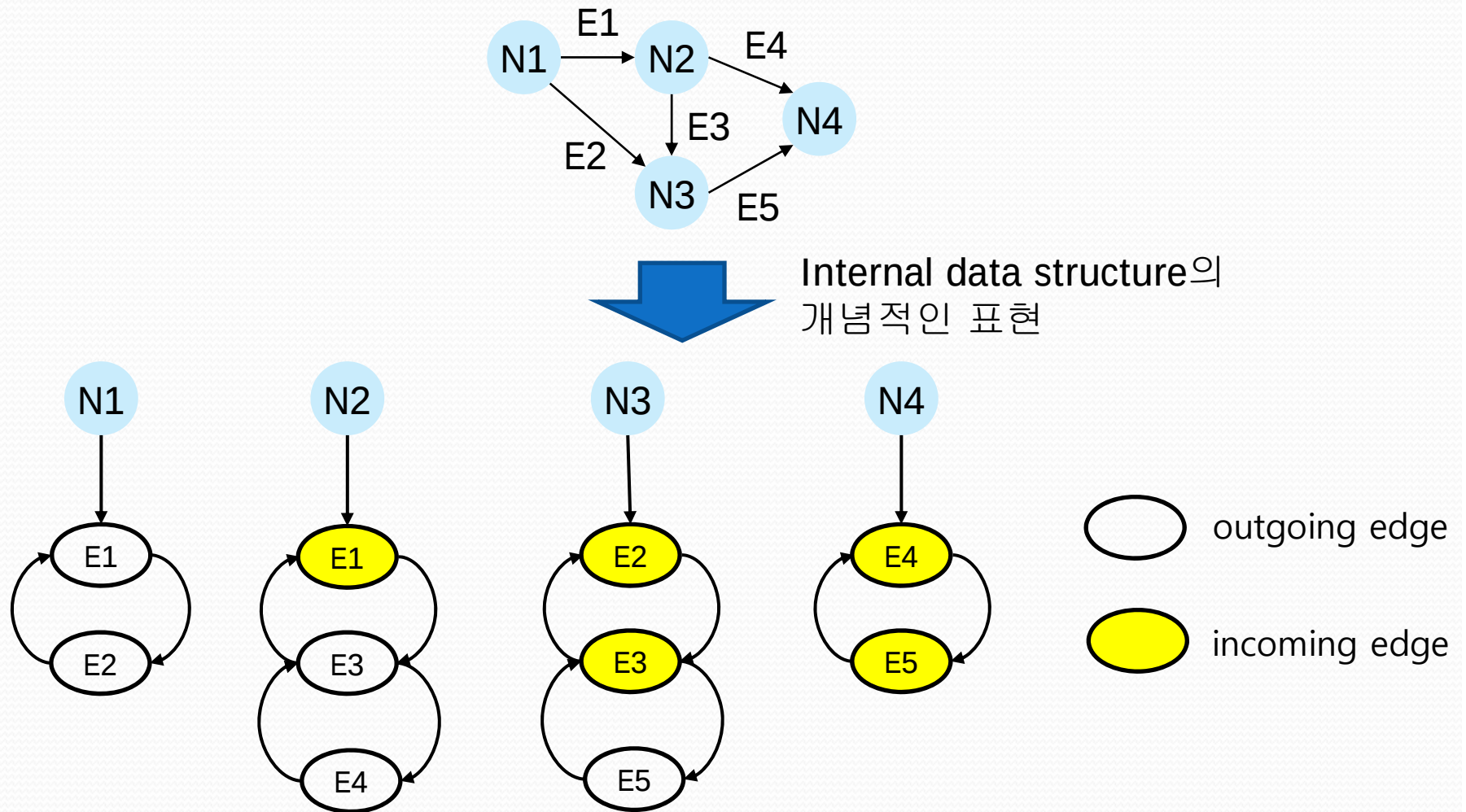
• Node

ID	식별자
inUse	존재함(1), 지워짐(0)
nextRelID	노드에 연결된 첫번째 relationship의 식별자

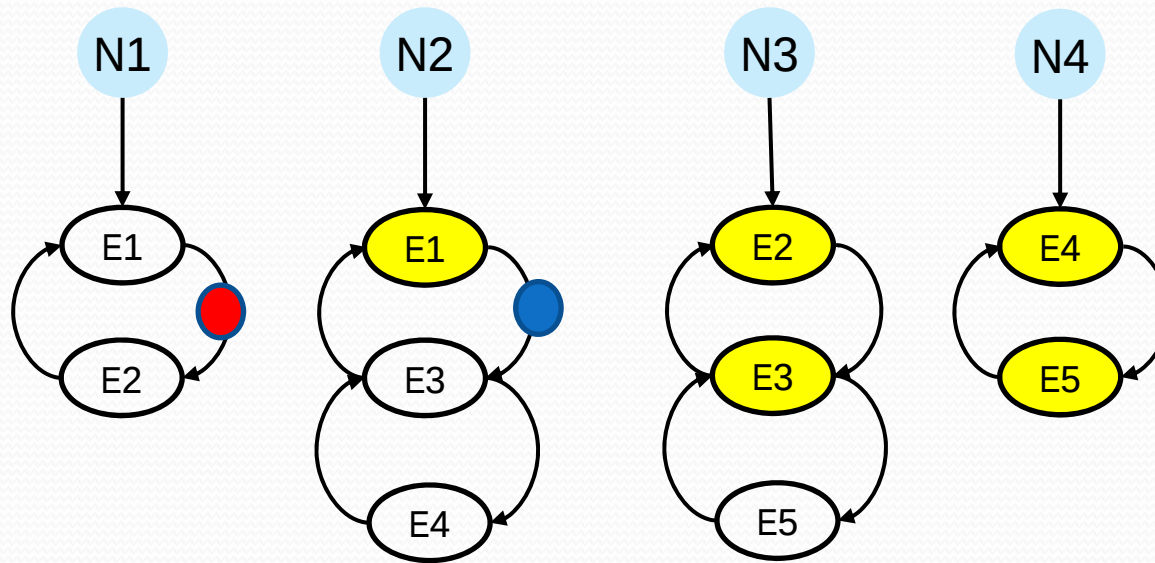
• Relationship

ID	식별자
inUse	존재함(1), 지워짐(0)
firstNode	에지의 source node 식별자
secondNode	에지의 destination node 식별자
firstPrevRelID	source node의 이전 relationship 식별자
firstNextRelID	source node의 다음 relationship 식별자
secondPrevRelID	destination node의 이전 relationship 식별자
secondNextRelID	destination node의 다음 relationship 식별자

Data Representation in Neo4J



Data Representation in Neo4J



Node

ID	inUse	nextRelId
N1	1	E1
N2	1	E1
N3	1	E2
N4	1	E4

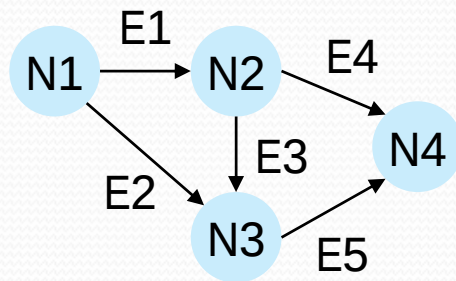
Relationship

ID	inUse	firstNode	secondNode	firstPrev RelId	firstNext RelId	secondPrev RelId	secondNext RelId
E1	1	N1	N2	Nil	E2	Nil	E3
E2	1	N1	N3	E1	Nil	Nil	E3
E3	1	N2	N3	E1	E4	E2	E5
E4	1	N2	N4	E3	Nil	Nil	E5
E5	1	N3	N4	E3	Nil	E4	Nil

Example: Search in Neo4J

- ID가 N2인 node로부터 N4로의 edge가 있는지 search
 - 1) Node로부터 N2(혹은 N4)의 첫 번째 edge ID를 검색
 - 2) 검색된 edge ID를 이용해서 relationship을 검색
 - 1) Relationship에서 N2(혹은 N4)가 firstNode이면 firstNextRelId를 secondNode이면 secondNextRelId를 따라감
 - 2) N4(혹은 N2)가 secondNode에서 발견될 때까지 위 과정을 반복 수행

Example: Search in Neo4J



1) search N2

Node

ID	inUse	nextRelId
N1	1	E1
N2	1	E1
N3	1	E2
N4	1	E4

2) traverse

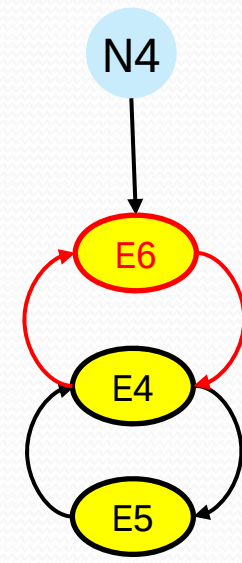
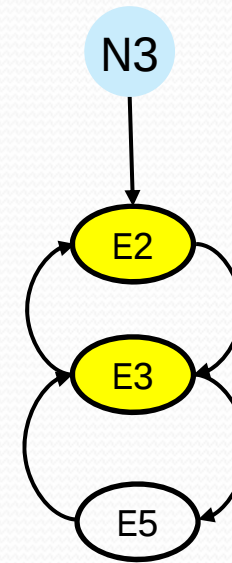
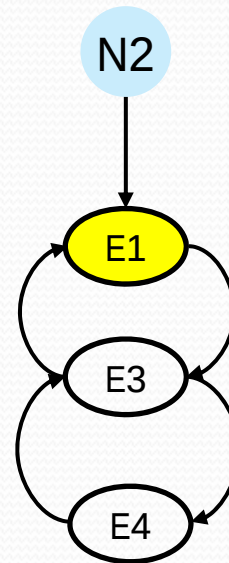
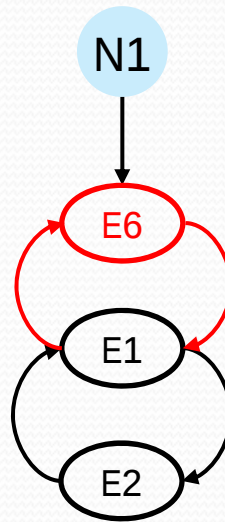
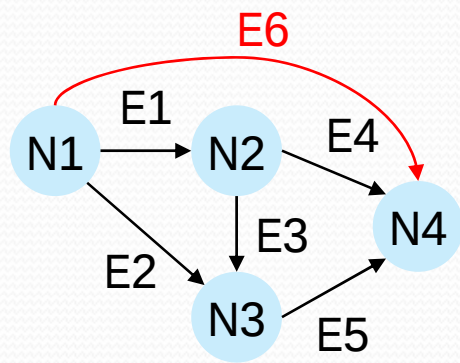
Relationship

ID	inUse	firstNode	secondNode	firstPrev RelId	firstNext RelId	secondPrev RelId	secondNext RelId
E1	1	N1	N2	Nil	E2	Nil	E3
E2	1	N1	N3	E1	Nil	Nil	E3
E3	1	N2	N3	E1	E4	E2	E5
E4	1	N2	N4	E3	Nil	Nil	E5
E5	1	N3	N4	E3	Nil	E4	Nil

발견

Example: Edge Insertion in Neo4J

- ID가 N1인 node로부터 N4로의 edge E6를 삽입



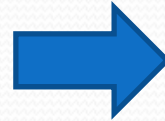
○ outgoing edge

● incoming edge

- ID가 N1인 node로부터 N4로의 edge E6를 삽입
 - Node 변경

• Node

ID	inUse	nextRelId
N1	1	E1
N2	1	E1
N3	1	E2
N4	1	E4



• Node

ID	inUse	nextRelId
N1	1	E6
N2	1	E1
N3	1	E2
N4	1	E6

- ID가 N1인 node로부터 N4로의 edge E6를 삽입
 - Relationship 변경

Relationship

ID	inUse	firstNode	secondNode	firstPrev RelId	firstNext RelId	secondPrev RelId	secondNext RelId
E1	1	N1	N2	Nil	E2	Nil	E3
E2	1	N1	N3	E1	Nil	Nil	E3
E3	1	N2	N3	E1	E4	E2	E5
E4	1	N2	N4	E3	Nil	Nil	E5
E5	1	N3	N4	E3	Nil	E4	Nil



Relationship

ID	inUse	firstNode	secondNode	firstPrev RelId	firstNext RelId	secondPrev RelId	secondNext RelId
E1	1	N1	N2	E6	E2	Nil	E3
E2	1	N1	N3	E1	Nil	Nil	E3
E3	1	N2	N3	E1	E4	E2	E5
E4	1	N2	N4	E3	Nil	E6	E5
E5	1	N3	N4	E3	Nil	E4	Nil
E6	1	N1	N4	Nil	E1	Nil	E4

GBase

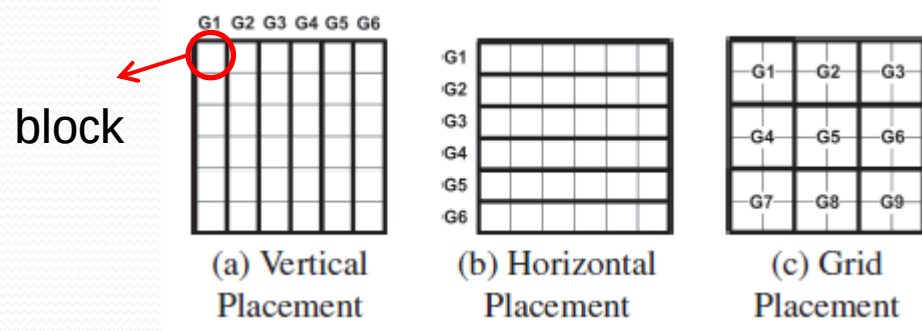
- 개요
 - MapReduce와 Adjacency Matrix 기반의 그래프 관리 시스템
- 특징
 - Block compression
Adjacency matrix를 block으로 partition하고, compress하여 저장
 - Grid placement
다양한 종류의 질의를 고려하여 평균 disk access 수를 줄일 수 있도록, block들의 집합인 grid단위로 disk에 저장하는 grid placement 정책을 사용
 - Query processing
전체 graph를 access하는 global query 5종류와 부분 graph를 access하는 targeted query 7종류를 처리

GBase: Block Compression

- Block Formulation
 - Adjacency matrix의 row와 column을 reordering하여 sparse한 matrix를 생성
 - Reordering algorithm으로 slashburn 알고리즘을 제안
 - Graph로부터 degree가 가장 큰 node부터 순서대로 relabeling
 - Graph로부터 relabeling된 node와 연결된 edge를 제거한 뒤, 생성되는 각 subgraph의 크기(=node 개수)가 threshold 미만일 때까지 반복
 - Threshold 미만 크기의 subgraph는 BFS로 labeling
- Block Compression
 - Block formulation이 완료된 adjacency matrix의 각 block을 zip encoding함

GBase: Grid Placement

- Compress된 block을 file system에 저장하는 방법
- 방법
 - 여러 개의 block을 1개 file로 저장
 - Vertical placement와 horizontal placement는 각각 out-neighbor와 in-neighbor query 중 1개 type의 query에 대해서만 잘 동작



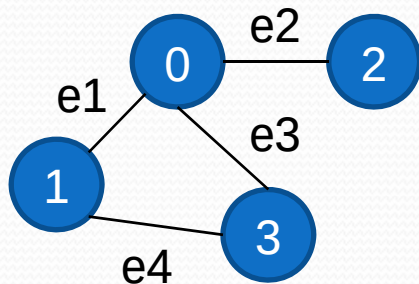
- Grid placement의 경우, file 개수가 K개 일 때, $O(\sqrt{K})$ 의 file access만으로 in/out-neighbor를 찾을 수 있음

GBase: Query Processing

- Global query와 targeted query의 처리를 지원
 - Global query: degree distribution, PageRank, PWR(Random Walk with Restart), radius estimation, connected component
 - Targeted query: 1 or K-step neighborhood, Induced graph, 1 or K-step egonet, K-core, Cross-edge
- HADOOP 기반의 질의 처리 수행

GBase: Query Processing

- Example: Induced subgraph query
 - 주어진 node 집합 V_q 에 대해, undirected graph G 로부터 V_q 를 node 집합으로 하고 V_q 에 속한 node들간의 edge만을 포함하는 subgraph를 찾음
 - V_q 의 induced subgraph $S(V_q) = B \times e_{V_q}$
 - B : $m \times n$ incident matrix (m : # of edges, n : # of nodes)
 - e_{V_q} = V_q 에 속한 element만 값이 1인 vector



$$V_q = \{0,1\}$$

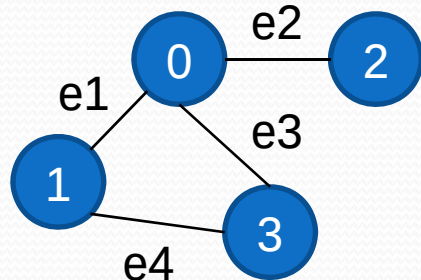
$$S(V_q) = \begin{bmatrix} 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} 1 \\ 1 \\ 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix}$$

값이 2인 것만 induced graph에서의 edge

GBase: Query Processing

- Induced subgraph를 위한 HADOOP 기반 알고리즘
 - Stage1: Map Stage
 - Adjacency matrix의 element로부터 대응하는 incident matrix의 2개 element를 생성하고 출력
 - $\langle \text{src}, [\text{src}, \text{dst}] \rangle, \langle \text{dst}, [\text{src}, \text{dst}] \rangle$
 - Query point는 $\langle \text{query}(=\text{node}) \text{의 id}, '1' \rangle$ 로 출력
 - Stage1: Reduce Stage
 - 입력인 value list에 '1'이 포함된 경우에만 value list에 포함된 $[\text{src}, \text{dst}]$ 로 $\langle [\text{src}, \text{dst}], 1 \rangle$ 를 생성하여 출력
 - Stage2: Map Stage는 bypass
 - Stage2: Reduce Stage
 - 입력인 value list의 element를 count해서 값이 2인 $\text{key}(=\text{edge} \langle [\text{src}, \text{dst}] \rangle)$ 를 출력

GBase: Query Processing



{0,1}의 induced subgraph를 찾아라.

Undirected Graph를 가정 (i.e., 각 e_i 가 1번만 입력됨)

Adjacency Matrix = $\begin{bmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 \end{bmatrix}$ 꺼

Stage 1

(0, [0,1])
(1, [0,1])
(0, '1')

(0, [0,2])
(2, [0,2])
(1, '1')

(0, [0,3])
(3, [0,3])

(1, [1,3])
(3, [1,3])

InducedSubgraph-**Map1**

(0, { [0,1], [0,2], [0,3], '1' })

(1, { [0,1], '1', [1,3] })

(2, { [0,2] })

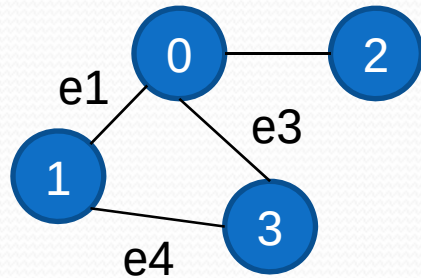
(3, { [0,3], [1,3] })

([0,1], 1)
([0,2], 1)
([0,3], 1)

([0,1], 1)
([1,3], 1)

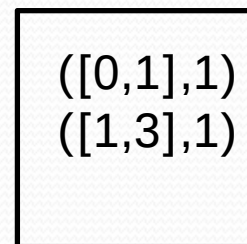
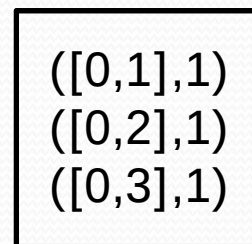
InducedSubgraph-**Reduce1**

GBase: Query Processing



{0,1}의 induced subgraph를 찾아라.

- Stage 2



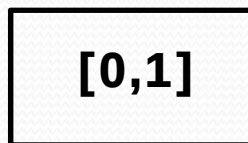
InducedSubgraph-**Map2**

$([0,1], \mathbf{\{1,1\}})$

$([0,2], \{1\})$

$([0,3], \{1\})$

$([1,3], \{1\})$



InducedSubgraph-**Reduce2**

**[0,1]이 2(=1+1)
번 나타남**

Pregel

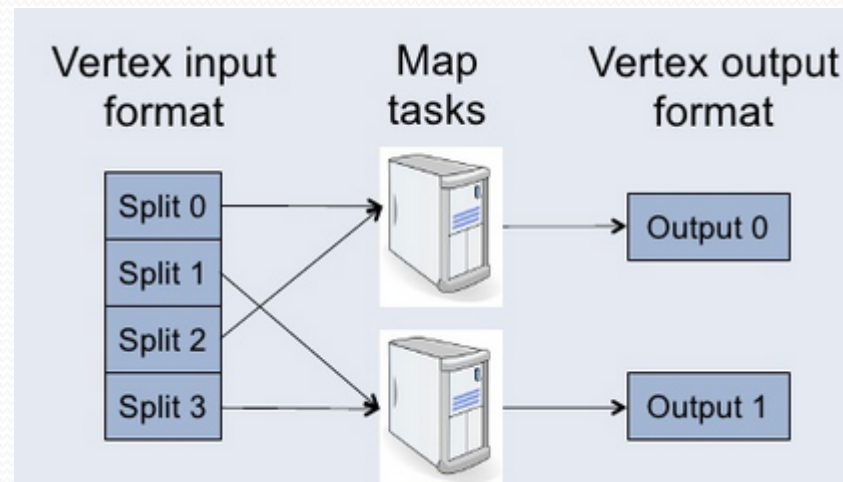
- 개요
 - Google의 internal graph processing platform
 - MapReduce와는 다른 computation model인 vertex-centric approach로 graph processing을 수행
- Vertex-centric approach
 - Program은 local computation, communication 및 synchronization으로 구성되는 superstep의 iteration으로 표현
 - 각 iteration에서 vertex는 다음과 같은 computation을 수행
 - 직전 iteration에서 전송된 message를 수신
 - Local computation을 통해, 자신이 state와 outgoing edge들을 수정
 - Outgoing edge들로 message를 전송

Differences from MapReduce

- Graph algorithm이 chained MapReduce invocation의 series로 표현될 수는 있으나, Pregel은 usability와 performance를 이유로 다른 model을 선택
- MapReduce
 - Passes the entire state of the graph from one stage to the next — requiring much more communication and associated serialization overhead
 - Needs to coordinate the steps of a chained MapReduce
- Pregel
 - Keeps vertices & edges on the machine that performs computation
 - Uses network transfers only for messages

Giraph: Pregel Implementation

- Giraph is a single Map-only job in Hadoop
- Hadoop is purely a resource manager for Giraph, all communication is done through Netty-based IPC (Hadoop RPC)
- Architecture



Single Source Shortes Path (SSSP)

- Problem
 - Find shortest path from a source node to all target nodes
- Solution
 - Single processor machine: Dijkstra's algorithm
 - MapReduce/Pregel: parallel bread-first search (BFS)

Example: SSSP – Parallel BFS in MapReduce

- Adjacency matrix

	A	B	C	D	E
A		10		5	
B			1	2	
C					4
D		3	9		2
E	7		6		

- Adjacency List

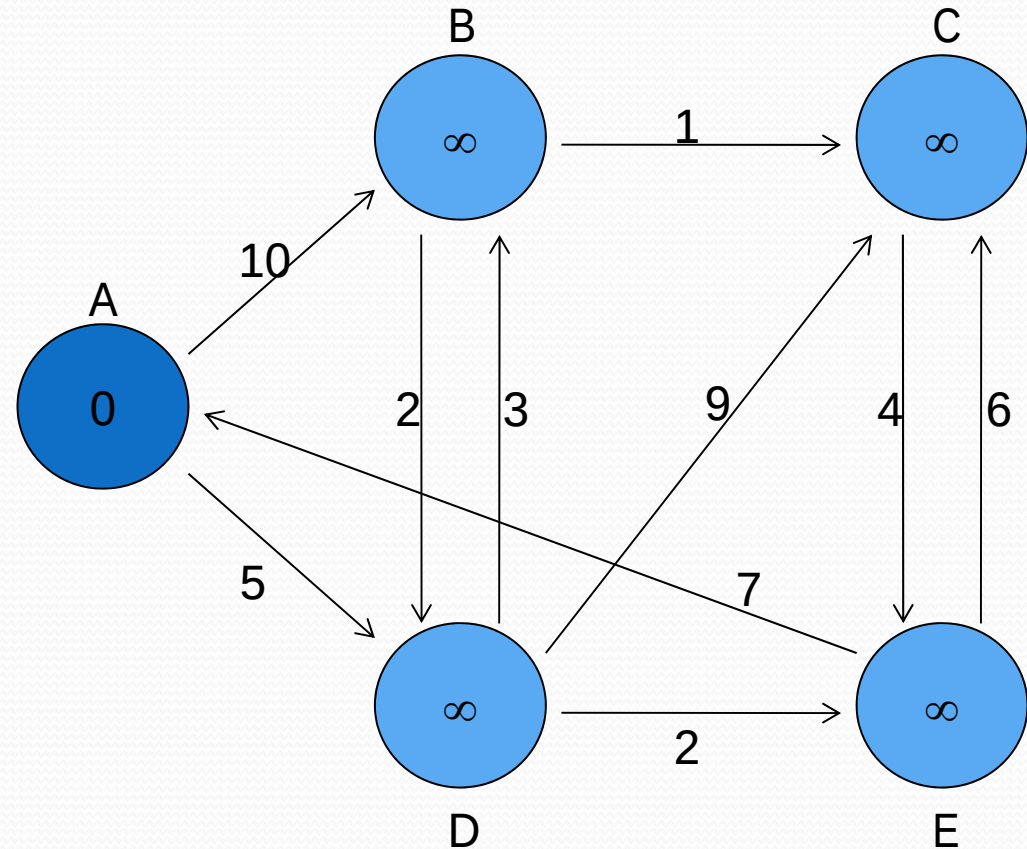
A: (B, 10), (D, 5)

B: (C, 1), (D, 2)

C: (E, 4)

D: (B, 3), (C, 9), (E, 2)

E: (A, 7), (C, 6)



Example: SSSP – Parallel BFS in MapReduce

- Map input: $\langle \text{node ID}, \langle \text{dist}, \text{adj list} \rangle \rangle$

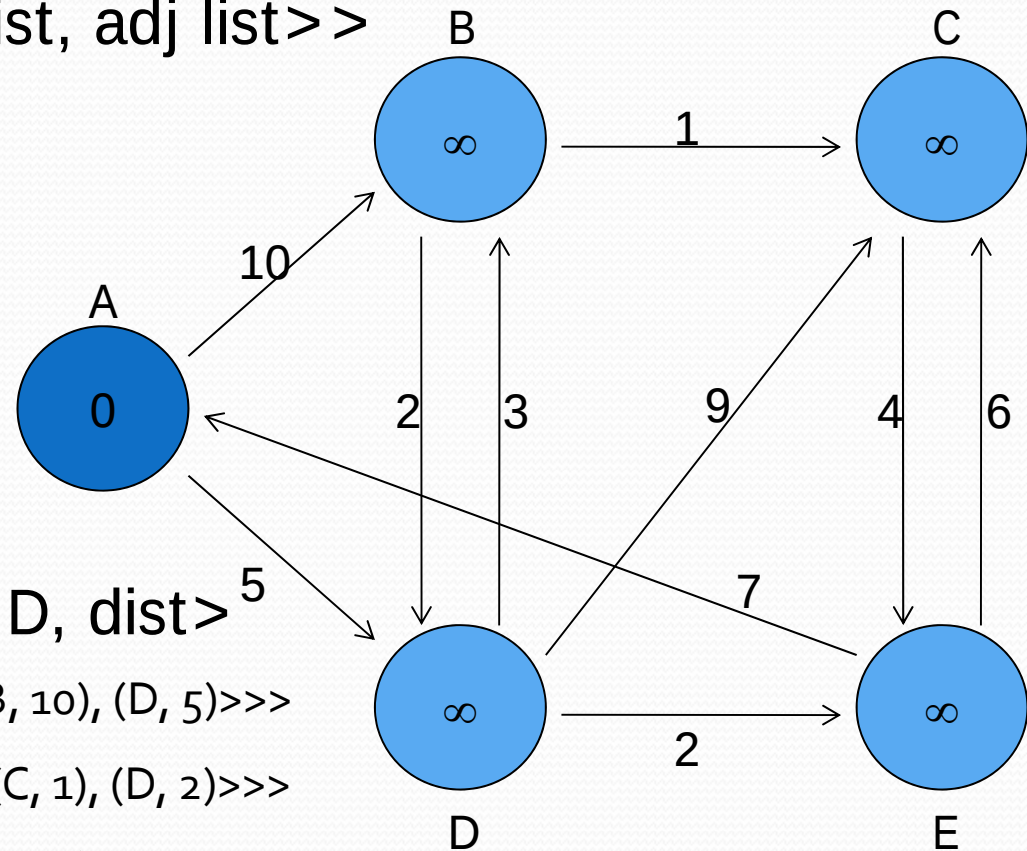
$\langle A, \langle 0, \langle (B, 10), (D, 5) \rangle \rangle \rangle$

$\langle B, \langle \text{inf}, \langle (C, 1), (D, 2) \rangle \rangle \rangle$

$\langle C, \langle \text{inf}, \langle (E, 4) \rangle \rangle \rangle$

$\langle D, \langle \text{inf}, \langle (B, 3), (C, 9), (E, 2) \rangle \rangle \rangle$

$\langle E, \langle \text{inf}, \langle (A, 7), (C, 6) \rangle \rangle \rangle$



- Map output: $\langle \text{dest node ID}, \text{dist} \rangle$

$\langle B, 10 \rangle \quad \langle D, 5 \rangle$

$\langle C, \text{inf} \rangle \quad \langle D, \text{inf} \rangle$

$\langle E, \text{inf} \rangle$

$\langle B, \text{inf} \rangle \quad \langle C, \text{inf} \rangle \quad \langle E, \text{inf} \rangle \quad \langle D, \text{inf}, \langle (B, 3), (C, 9), (E, 2) \rangle \rangle$

$\langle A, \text{inf} \rangle \quad \langle C, \text{inf} \rangle$

$\langle A, \langle 0, \langle (B, 10), (D, 5) \rangle \rangle \rangle$

$\langle B, \langle \text{inf}, \langle (C, 1), (D, 2) \rangle \rangle \rangle$

$\langle C, \langle \text{inf}, \langle (E, 4) \rangle \rangle \rangle$

$\langle E, \langle \text{inf}, \langle (A, 7), (C, 6) \rangle \rangle \rangle$

Flushed to local disk!!

Example: SSSP – Parallel BFS in MapReduce

- Reduce input: <node ID, dist>

<A, <0, <(B, 10), (D, 5)>>>

<A, inf>

<B, <inf, <(C, 1), (D, 2)>>>

<B, 10> <B, inf>

<C, <inf, <(E, 4)>>>

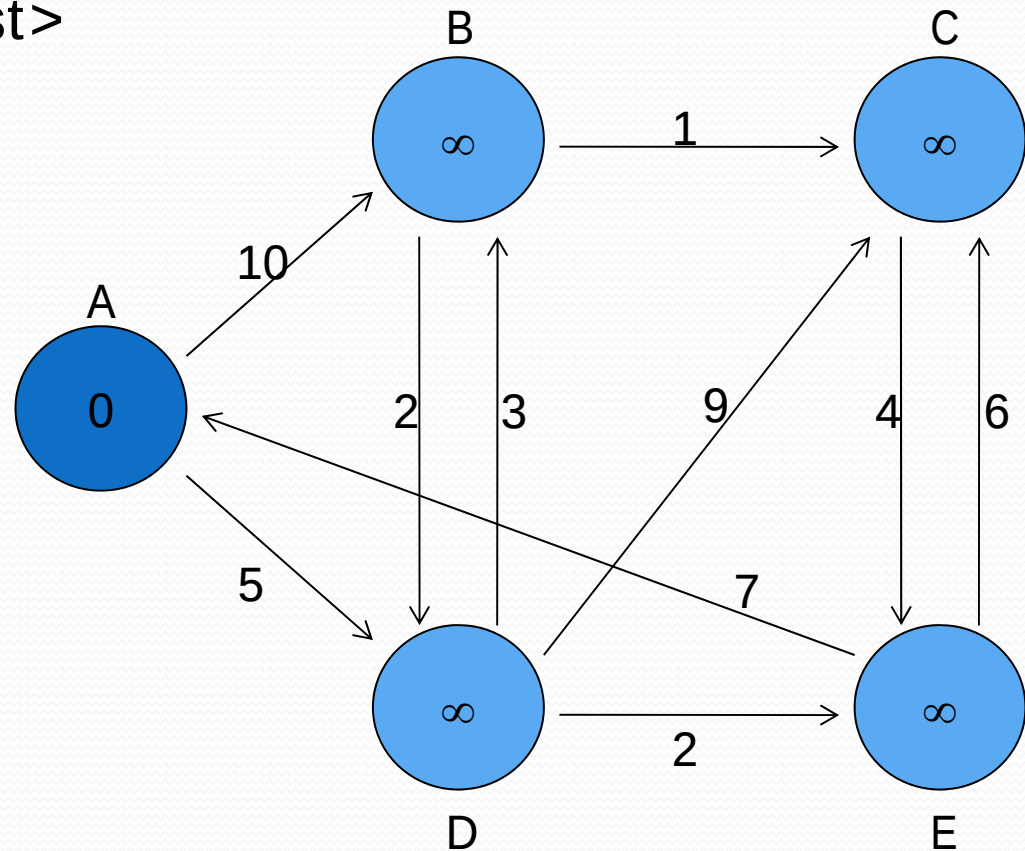
<C, inf> <C, inf> <C, inf>

<D, <inf, <(B, 3), (C, 9), (E, 2)>>>

<D, 5> <D, inf>

<E, <inf, <(A, 7), (C, 6)>>>

<E, inf> <E, inf>



Example: SSSP – Parallel BFS in MapReduce

- Reduce input: <node ID, dist>

<A, <0, <(B, 10), (D, 5)>>>

~~<A, inf>~~

<B, <inf, <(C, 1), (D, 2)>>>

<B, 10> ~~<B, inf>~~

<C, <inf, <(E, 4)>>>

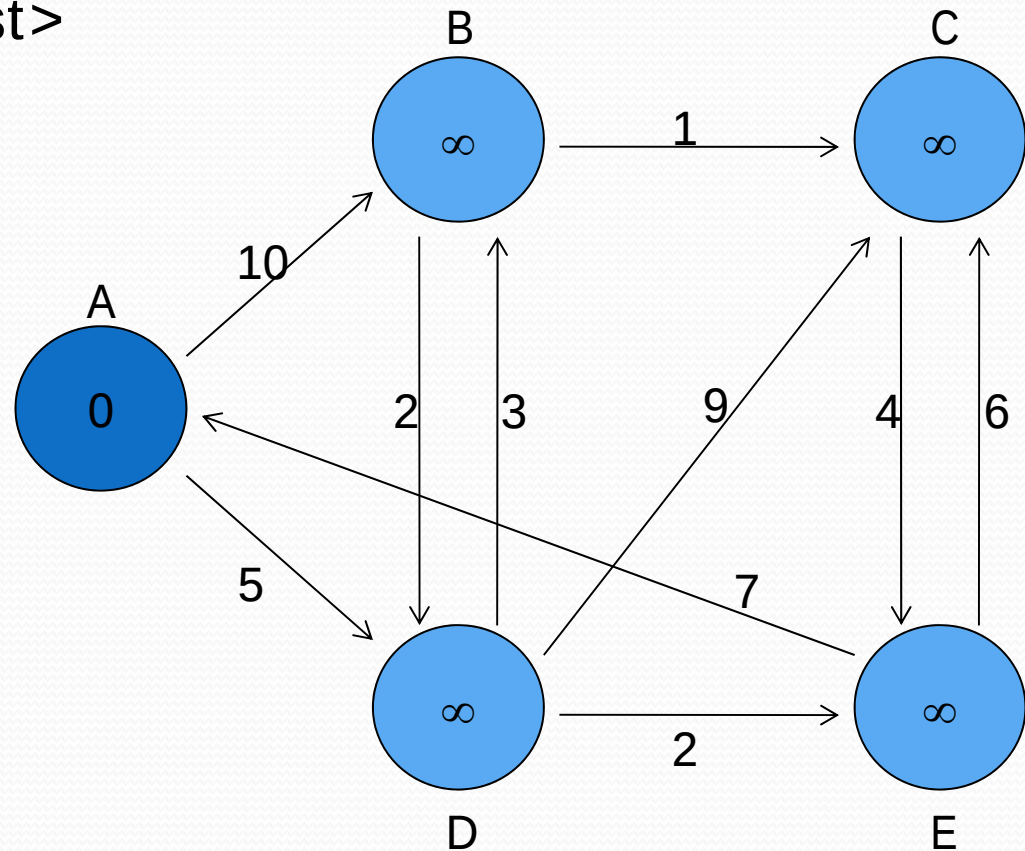
~~<C, inf>~~ ~~<C, inf>~~ ~~<C, inf>~~

<D, <inf, <(B, 3), (C, 9), (E, 2)>>>

<D, 5> ~~<D, inf>~~

<E, <inf, <(A, 7), (C, 6)>>>

~~<E, inf>~~ ~~<E, inf>~~



Example: SSSP – Parallel BFS in MapReduce

- Reduce output: $\langle \text{node ID}, \langle \text{dist}, \text{adj list} \rangle \rangle$
= Map input for next iteration

$\langle A, \langle 0, \langle (B, 10), (D, 5) \rangle \rangle \rangle$ **Flushed to DFS!!**

$\langle B, \langle 10, \langle (C, 1), (D, 2) \rangle \rangle \rangle$

$\langle C, \langle \text{inf}, \langle (E, 4) \rangle \rangle \rangle$

$\langle D, \langle 5, \langle (B, 3), (C, 9), (E, 2) \rangle \rangle \rangle$

$\langle E, \langle \text{inf}, \langle (A, 7), (C, 6) \rangle \rangle \rangle$

- Map output: $\langle \text{dest node ID}, \text{dist} \rangle$

$\langle B, 10 \rangle \quad \langle D, 5 \rangle$

$\langle C, 11 \rangle \quad \langle D, 12 \rangle$

$\langle E, \text{inf} \rangle$

$\langle B, 8 \rangle \quad \langle C, 14 \rangle \quad \langle E, 7 \rangle$

$\langle A, \text{inf} \rangle \quad \langle C, \text{inf} \rangle$

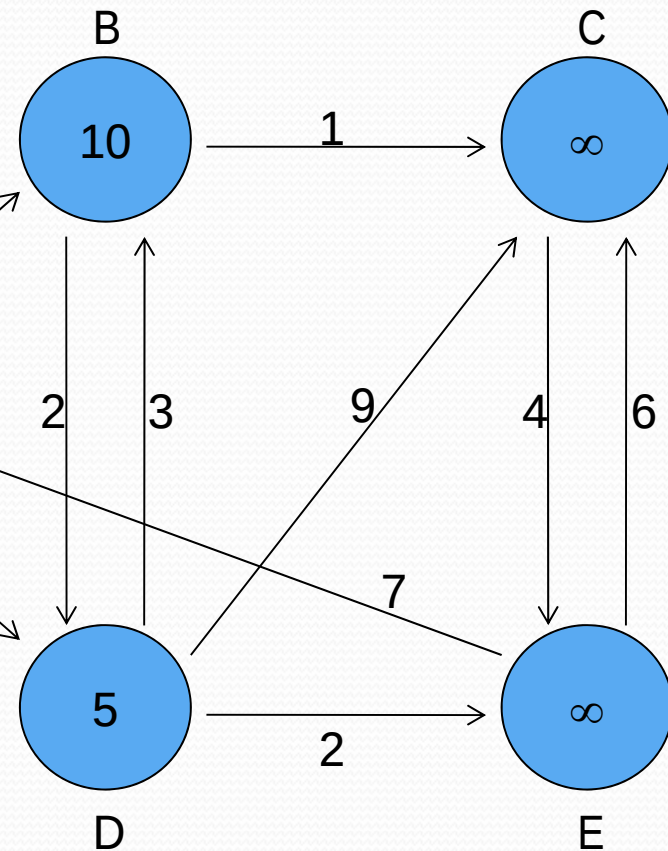
$\langle A, \langle 0, \langle (B, 10), (D, 5) \rangle \rangle \rangle$

$\langle B, \langle 10, \langle (C, 1), (D, 2) \rangle \rangle \rangle$

$\langle C, \langle \text{inf}, \langle (E, 4) \rangle \rangle \rangle$

$\langle D, \langle 5, \langle (B, 3), (C, 9), (E, 2) \rangle \rangle \rangle$

$\langle E, \langle \text{inf}, \langle (A, 7), (C, 6) \rangle \rangle \rangle$



Flushed to local disk!!

Example: SSSP – Parallel BFS in MapReduce

- Reduce input: <node ID, dist>

<A, <0, <(B, 10), (D, 5)>>>

<A, inf>

<B, <10, <(C, 1), (D, 2)>>>

<B, 10> <B, 8>

<C, <inf, <(E, 4)>>>

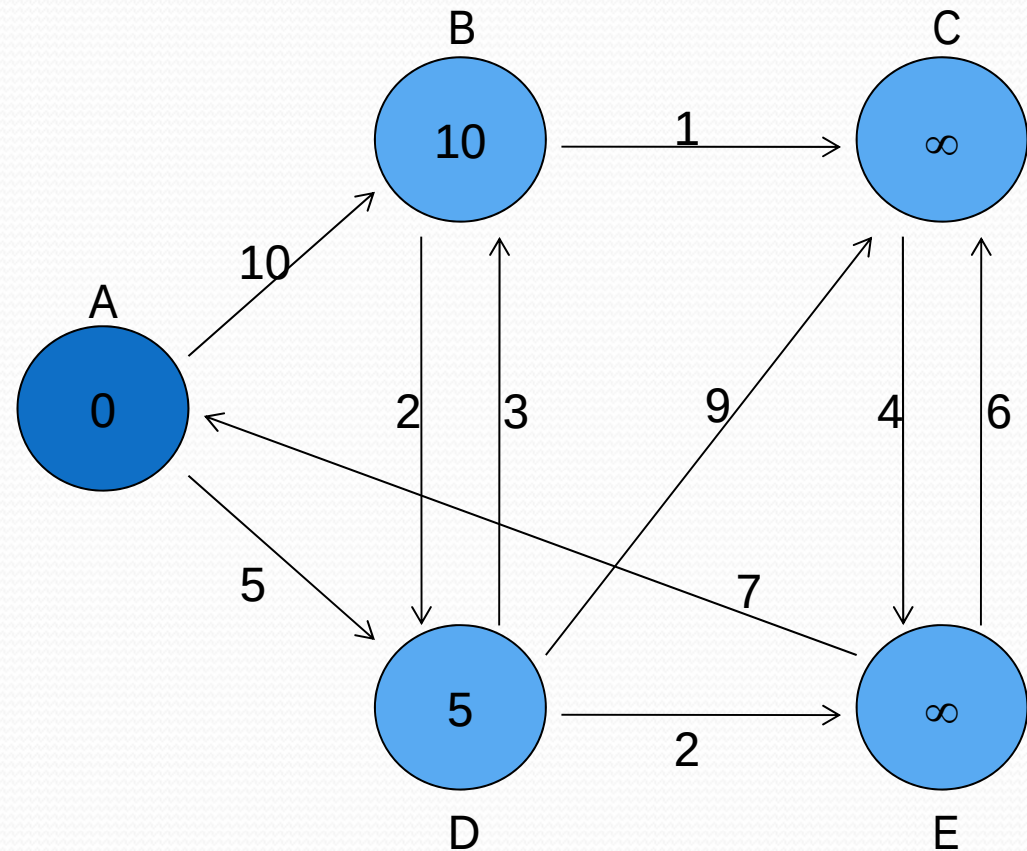
<C, 11> <C, 14> <C, inf>

<D, <5, <(B, 3), (C, 9), (E, 2)>>>

<D, 5> <D, 12>

<E, <inf, <(A, 7), (C, 6)>>>

<E, inf> <E, 7>



Example: SSSP – Parallel BFS in MapReduce

- Reduce input: <node ID, dist>

<A, <0, <(B, 10), (D, 5)>>>

~~<A, inf>~~

<B, <~~10~~, <(C, 1), (D, 2)>>>

~~<B, 10>~~ <B, 8>

<C, <inf, <(E, 4)>>>

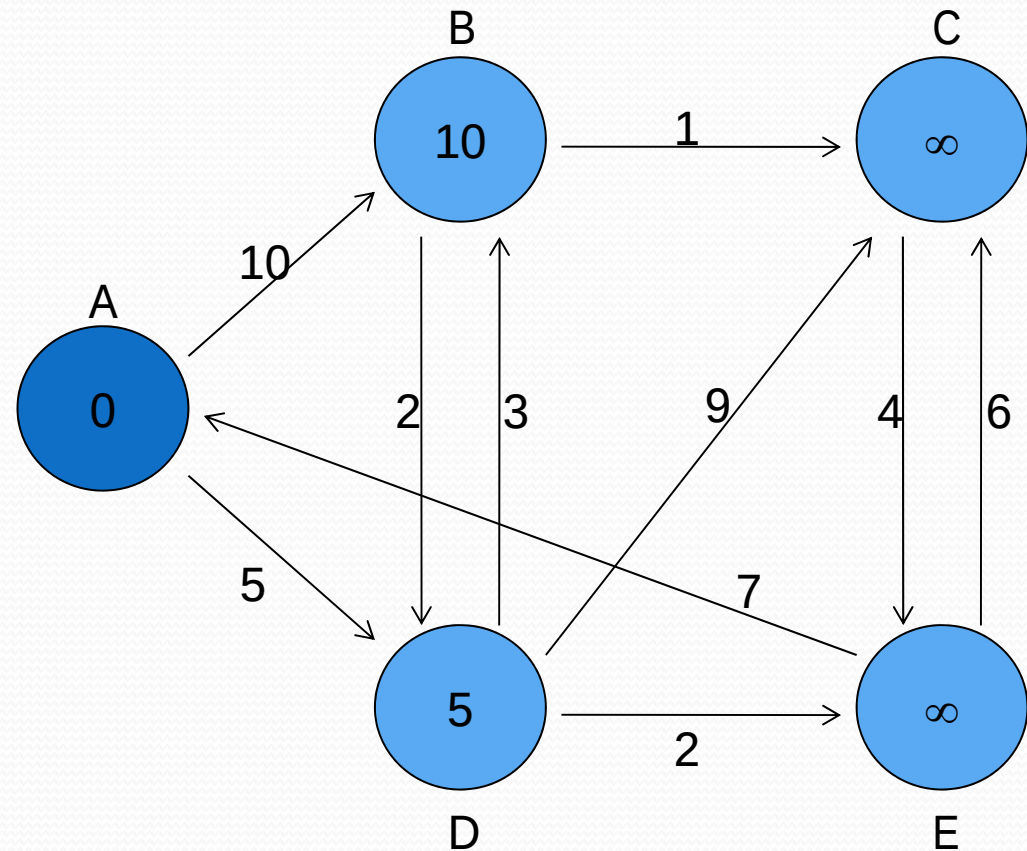
<C, 11> ~~<C, 14>~~ ~~<C, inf>~~

<D, <5, <(B, 3), (C, 9), (E, 2)>>>

~~<D, 5>~~ ~~<D, 12>~~

<E, <inf, <(A, 7), (C, 6)>>>

~~<E, inf>~~ <E, 7>



Example: SSSP – Parallel BFS in MapReduce

- Reduce output: $\langle \text{node ID}, \langle \text{dist}, \text{adj list} \rangle \rangle$
= Map input for next iteration

$\langle A, \langle 0, \langle (B, 10), (D, 5) \rangle \rangle \rangle$ **Flushed to DFS!!**

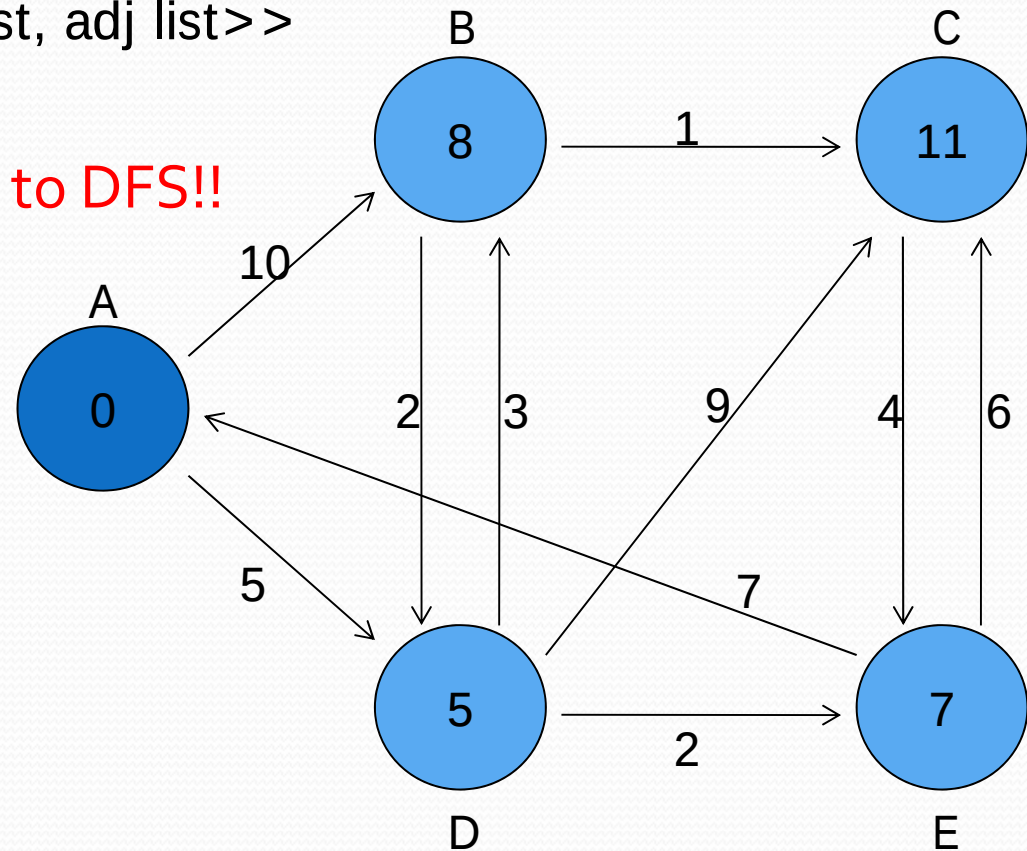
$\langle B, \langle 8, \langle (C, 1), (D, 2) \rangle \rangle \rangle$

$\langle C, \langle 11, \langle (E, 4) \rangle \rangle \rangle$

$\langle D, \langle 5, \langle (B, 3), (C, 9), (E, 2) \rangle \rangle \rangle$

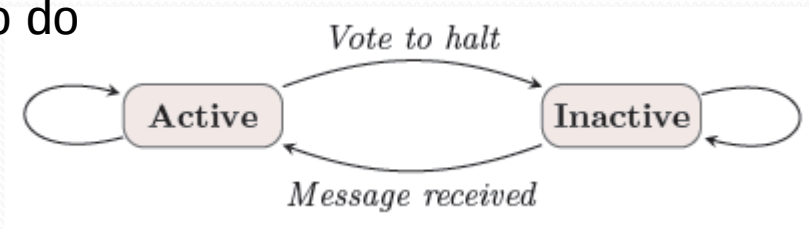
$\langle E, \langle 7, \langle (A, 7), (C, 6) \rangle \rangle \rangle$

... the rest omitted ...

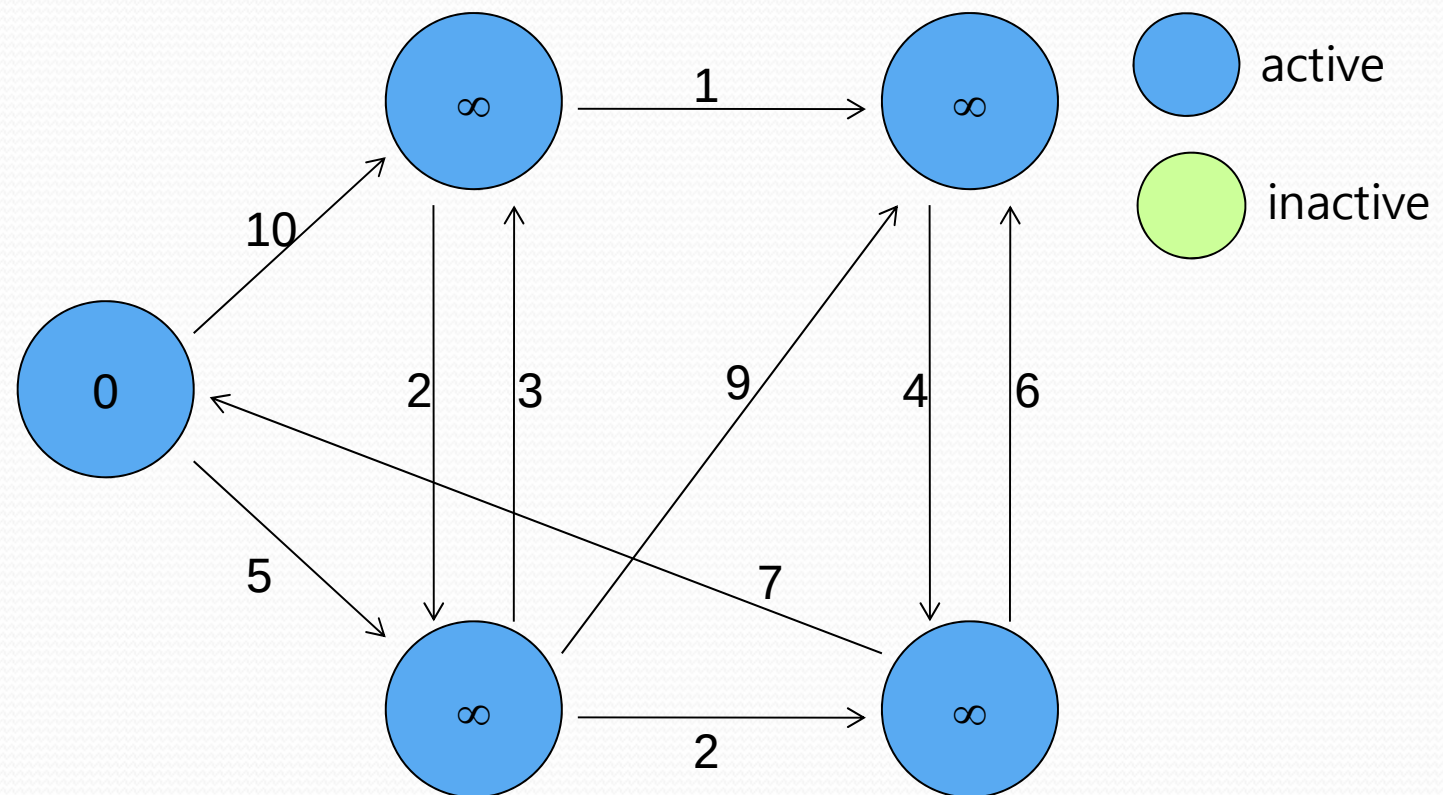


Pregel

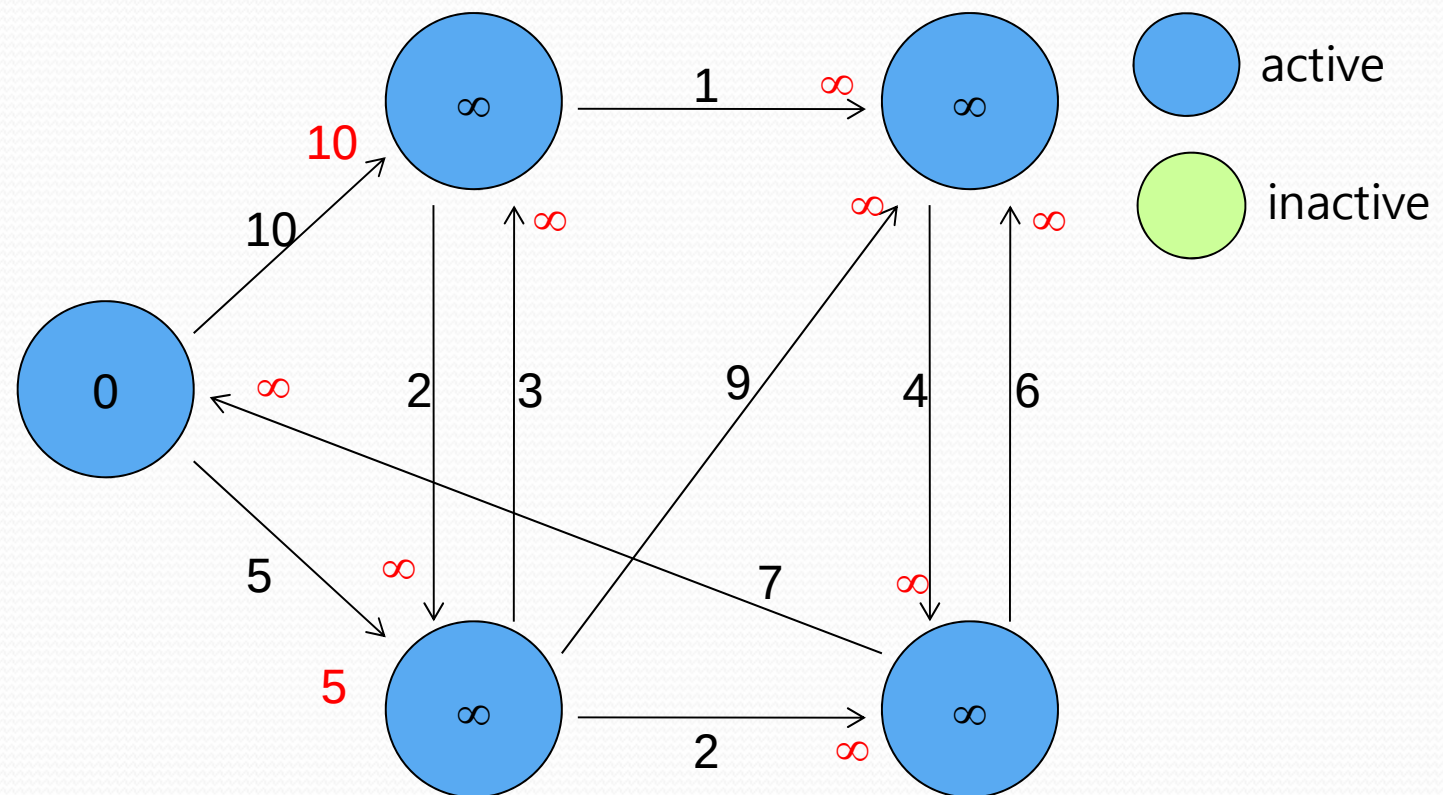
- Superstep: the vertices compute in parallel
 - Each vertex
 - Receives messages sent in the previous superstep
 - Executes the same user-defined function
 - Modifies its value or that of its outgoing edges
 - Sends messages to other vertices (to be received in the next superstep)
 - Mutates the topology of the graph
 - Votes to halt if it has no further work to do
 - Termination condition
 - All vertices are simultaneously inactive
 - There are no messages in transit



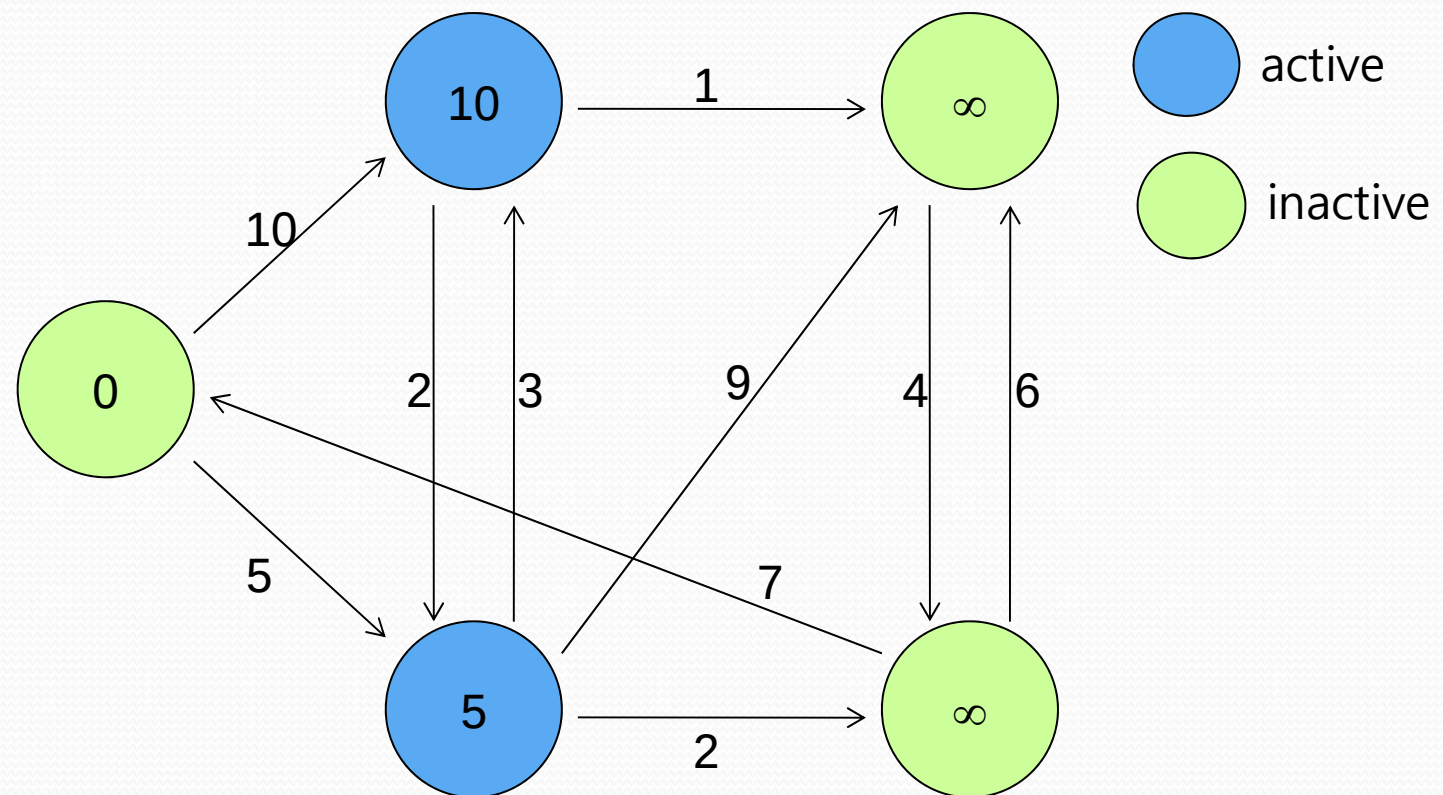
Example: SSSP – Parallel BFS in Pregel



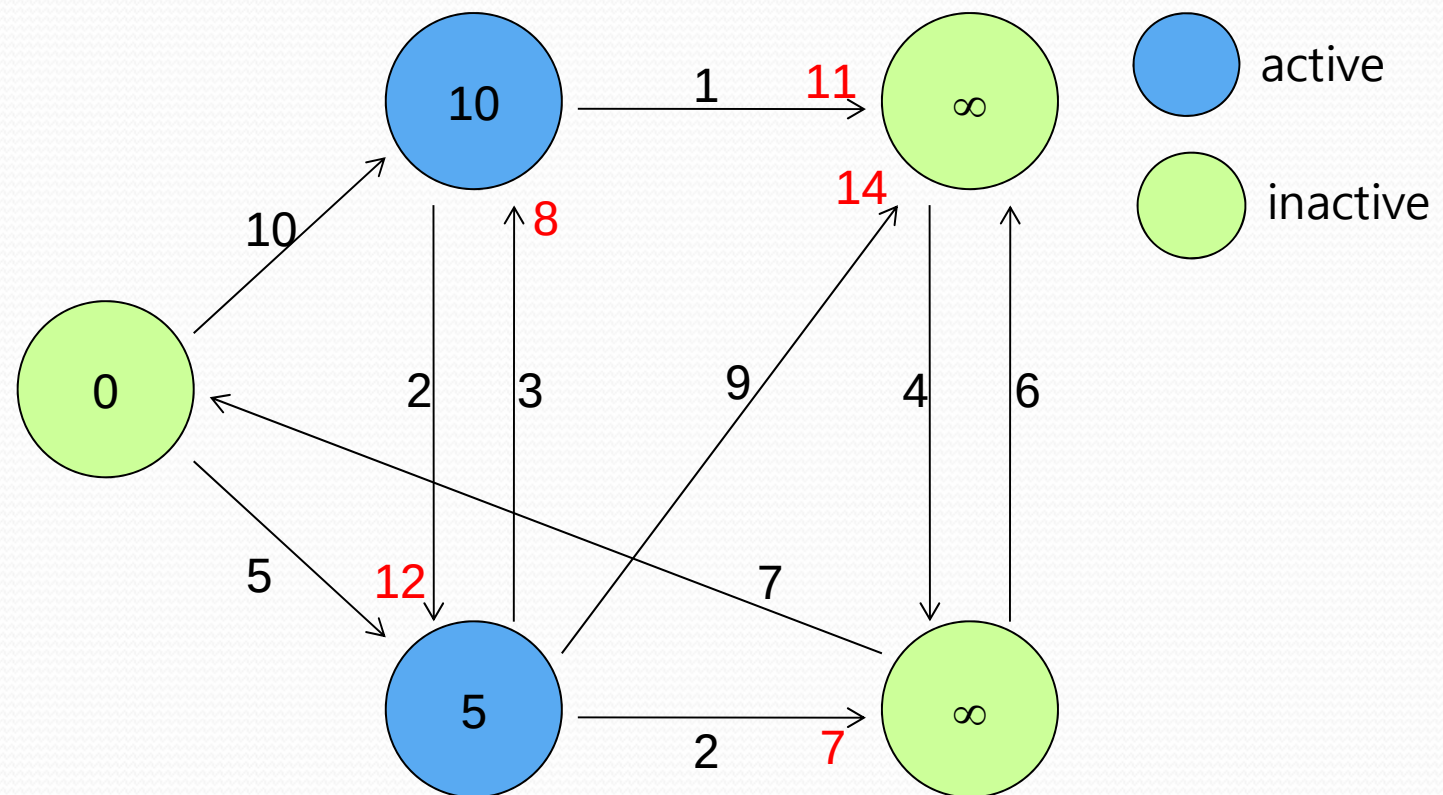
Example: SSSP – Parallel BFS in Pregel



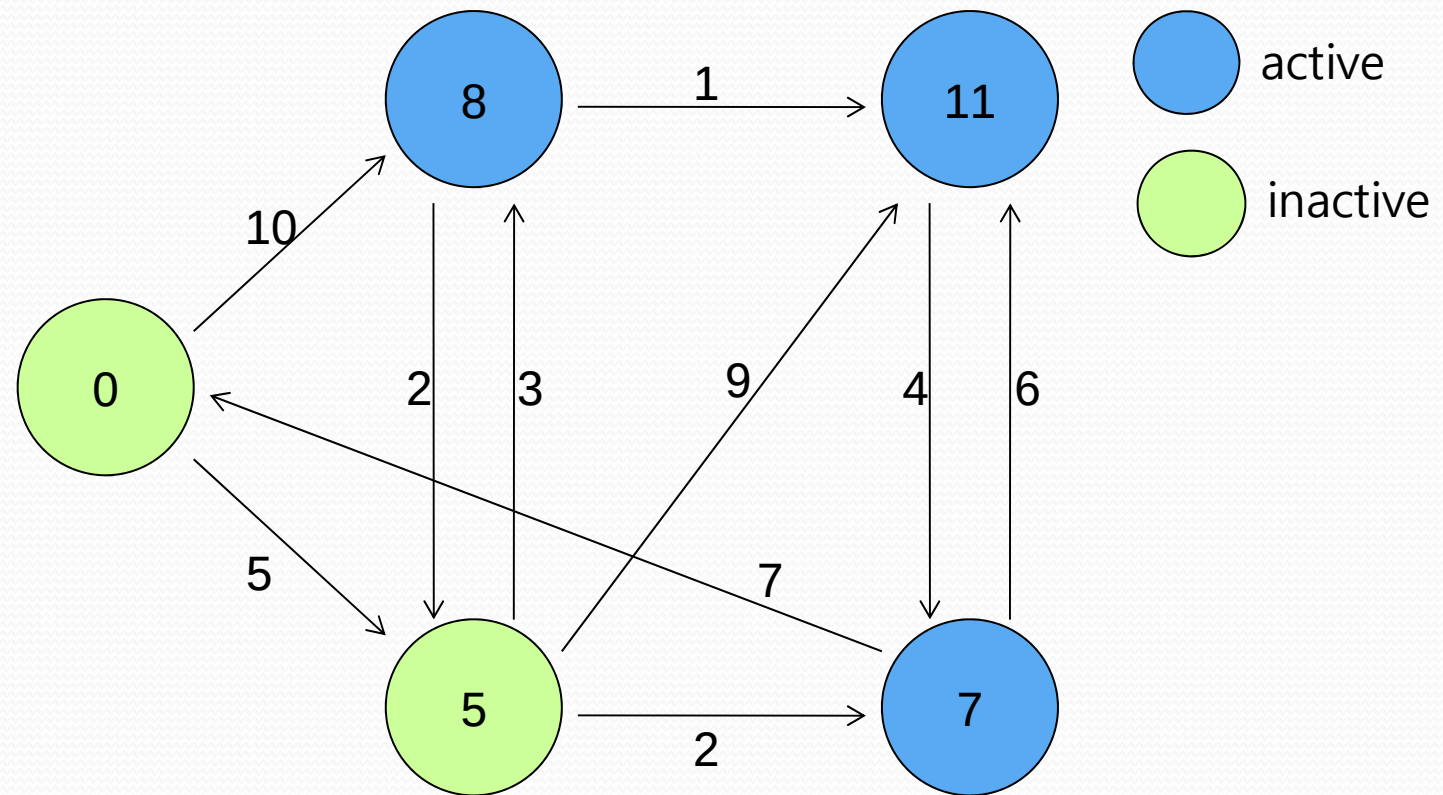
Example: SSSP – Parallel BFS in Pregel



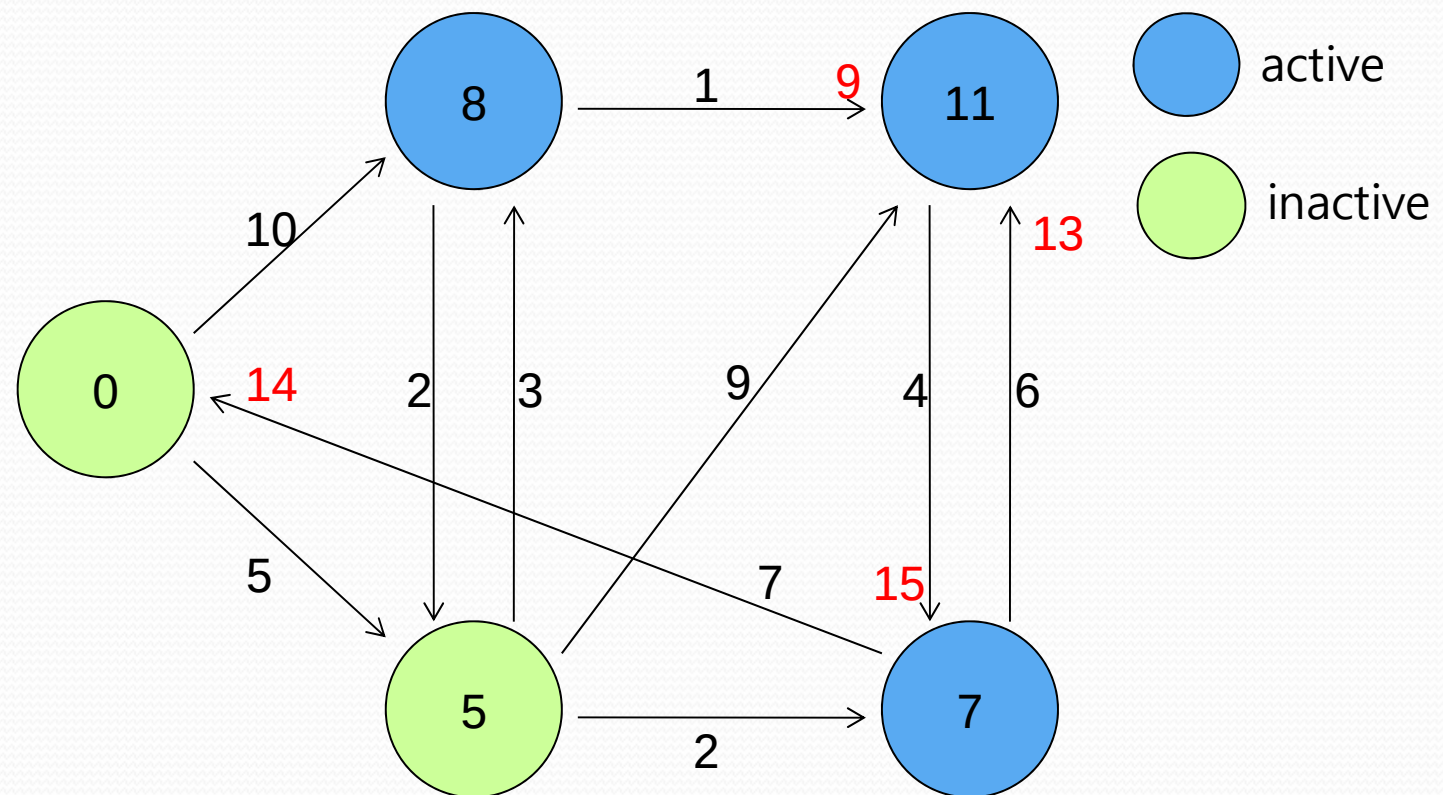
Example: SSSP – Parallel BFS in Pregel



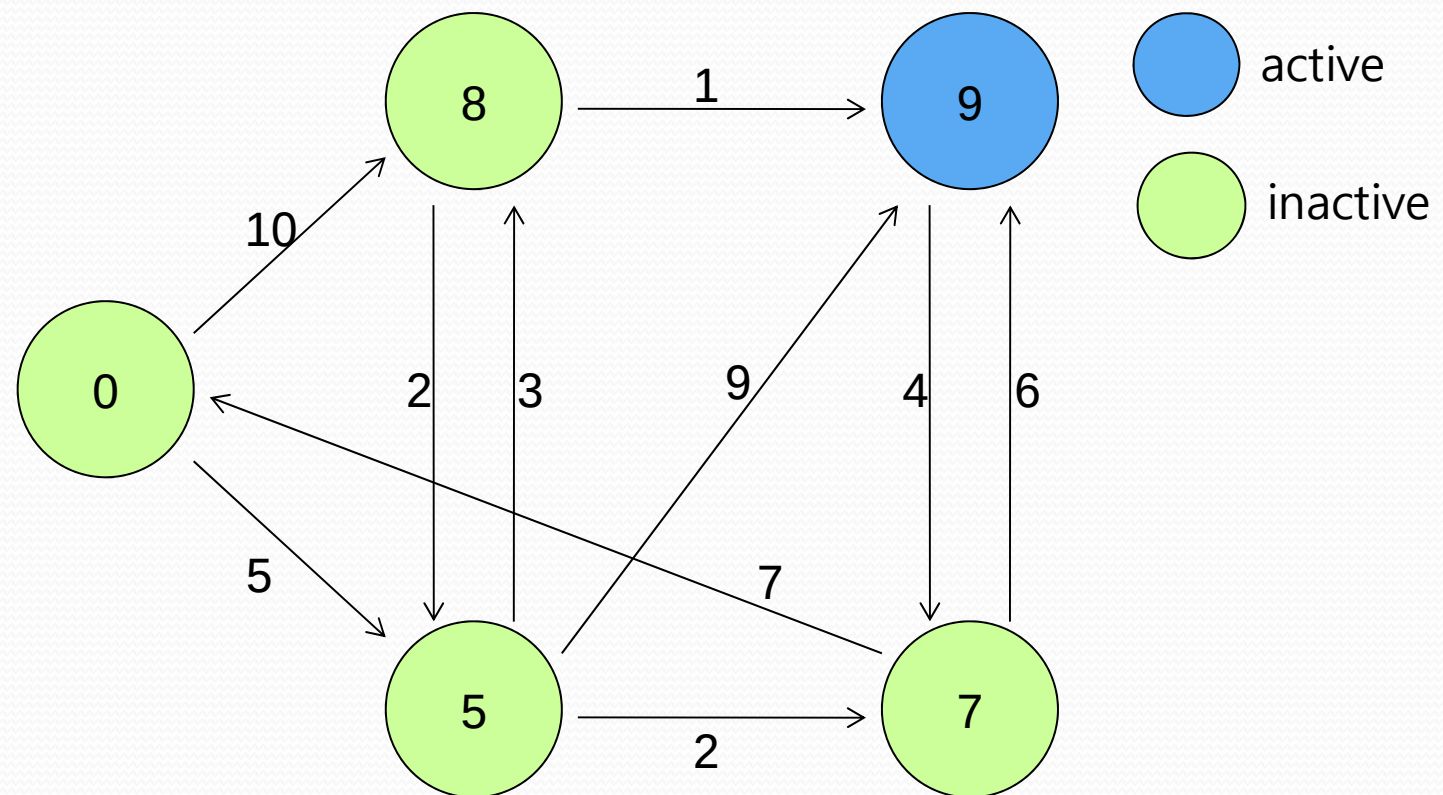
Example: SSSP – Parallel BFS in Pregel



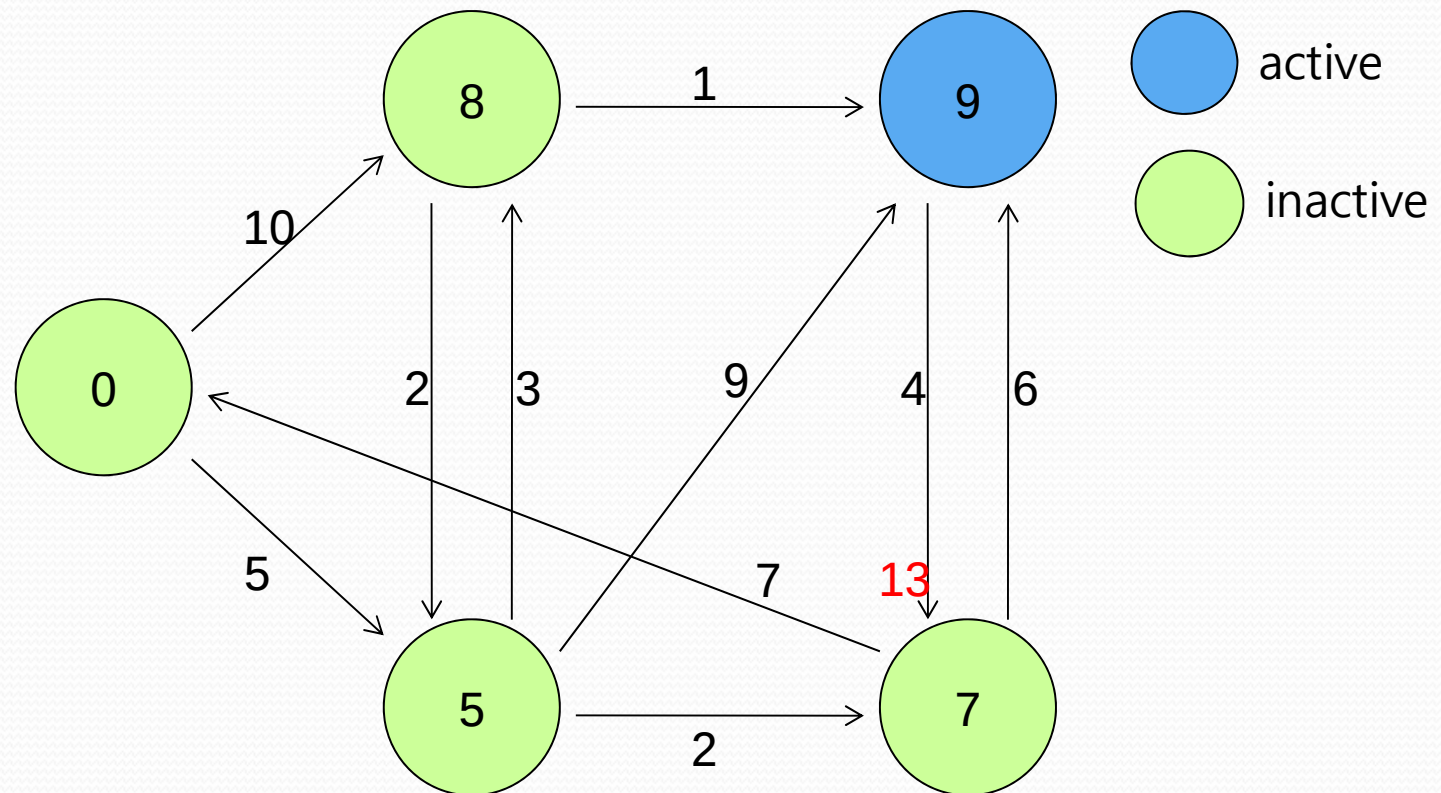
Example: SSSP – Parallel BFS in Pregel



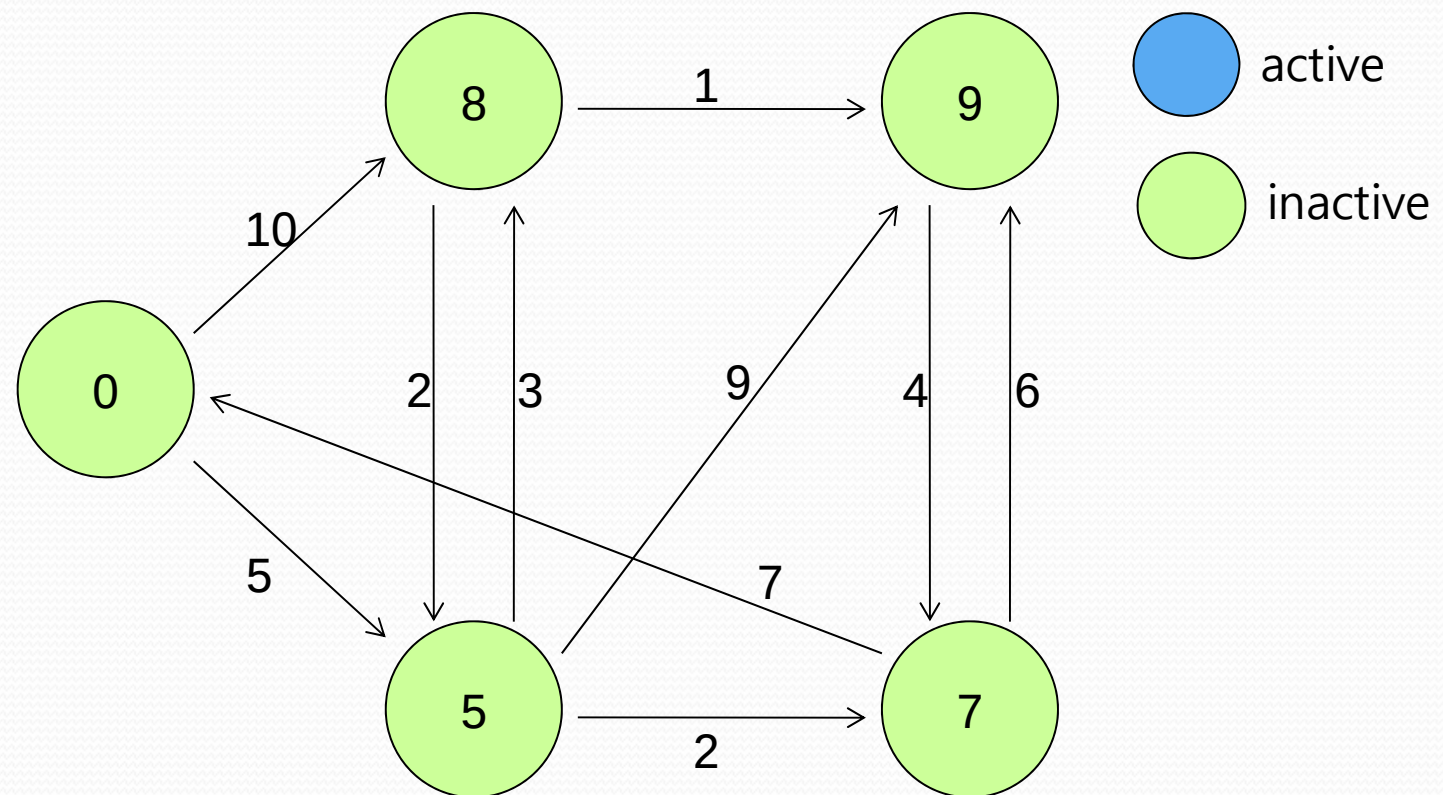
Example: SSSP – Parallel BFS in Pregel



Example: SSSP – Parallel BFS in Pregel



Example: SSSP – Parallel BFS in Pregel



모든 vertex가 inactive이므로 End

Social and Information Network DBMS (TurboDB)

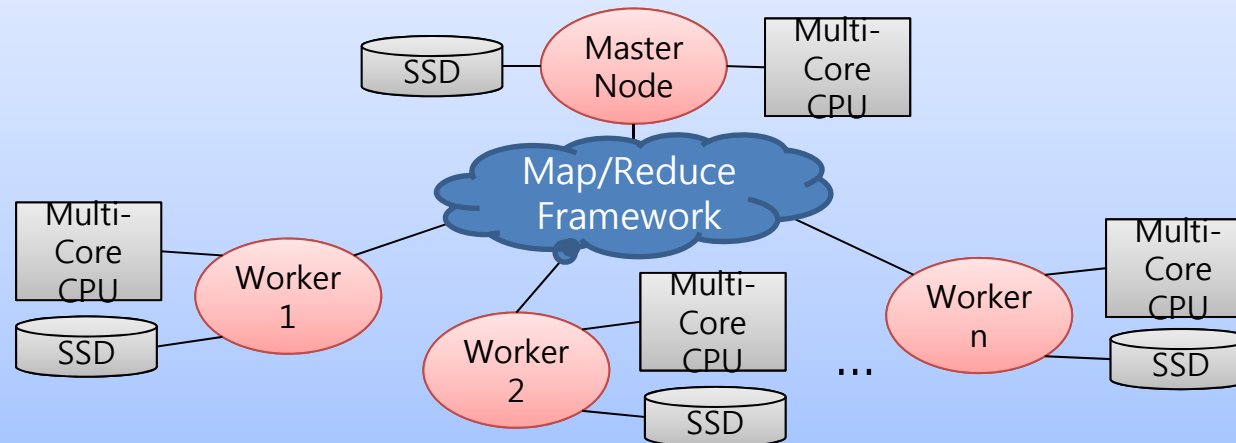
- 개요
 - 본 연구실에서 개발 중인 빅 데이터를 위한 병렬 그래프 DBMS
- 연구/개발 방향
 - MapReduce 기반의 병렬 처리 지원
 - BFS, Subgraph Isomorphism 등 그래프 DBMS가 지원해야 할 중요한 연산자들을 시스템 레벨에서 지원
 - SSD 및 multi-core CPU 등의 현대 hardware 특성을 효과적으로 활용하도록 시스템을 구성

TurboDB의 시스템 아키텍처

Parallel Social & Information Network
Big Data Mining Algorithm

Query / Results

Parallel Social & Information Network DBMS (TurboDB)



An In-depth Comparison of Subgraph Isomorphism Algorithms in Graph Databases

- 주요 공헌 요약
 - Subgraph isomorphism 문제는 주어진 질의 그래프를 포함하는 모든 데이터 서브 그래프를 찾는 문제임
 - 이 문제는 NP-complete에 속하는 아주 어려운 문제임
 - 기존 알고리즘들의 성능을 정형적이고 공정하게 비교하기 위한 프레임워크 제안
 - Generic algorithm을 통해 기존 알고리즘들이 몇 가지 함수만을 바꿈으로써 구현될 수 있음을 보임
 - 정형적이고 정성적인 방법을 통해 기존 알고리즘들의 문제점을 처음으로 밝혀 냄
 - 향후 기존 알고리즘들을 모두 개선할 수 있는 새 알고리즘의 근간을 제안함
- **DB 분야 Top 학술 대회인 VLDB 2013에 게재 승인됨**

-
- 감사합니다.