MapReduce II

MapReduce for Graph Queries

- Input: Query Q and graph G
- Output: answers Q(G) to Q in G
- Map (key: *node*, value: *(adjacency-list, others)*)
  - Computation
  - Emit (*mkey*, *mvalue*)

Match rkey, rvalue when multiple iterations of MapReduce are needed

- Reduce(key: *mkey*, value: list[*mvalue*])
  - Computation
  - Emit (*rkey*, *rvalue*)

Match mkey, mvalue

Dijkstra's Algorithm for Distance Query

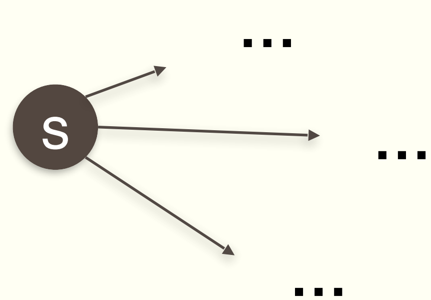
- Distance: single-source shortest-path problem
  - Input: A directed weighted graph G and a node s in G
  - Output: The length of shortest paths from s to all nodes in G
- Dijkstra (G, s, w)
  - For all nodes $v \in V$ do
    - $d[v] \leftarrow \infty$
  - $d[s] \leftarrow 0$; $Q \leftarrow V$
  - While Q is non-empty, do
    - $u \leftarrow ExtractMin(Q)$
    - For all nodes $v \in adj(u)$ do
      - $d[v] > d[u] + w(u, v)$ then $d[v] \leftarrow d[u] + w(u, v)$
- Complexity:
  - $O\left(|V|\log|V| + |E|\right)$

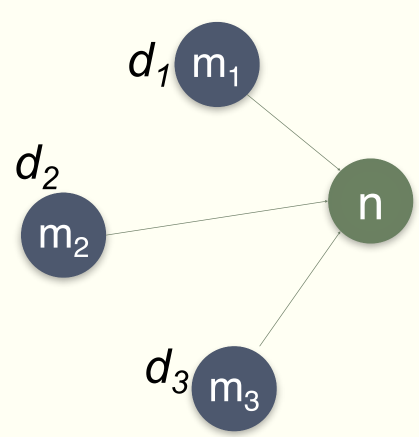
Use a priority queue Q;
$w(u, v)$: weight of edge (u, v);
$d(u)$: the distance from s to u

Extract one with the minimum $d(u)$

Finding the Shortest Path

- Consider simple case of equal edge weights: solution to the problem can be defined inductively
- Intuition:
  - Define: *b* is reachable from *a* if *b* is on adjacency list of *a*
  - $d[s] = 0$
  - For all nodes *p* reachable from *s*, $d[p] = 1$
  - For all nodes *n* reachable from some other set of nodes *M*, $d[n] = 1 + \min_{m \in M} d[m]$





Shortest Path: From Intuition to Algorithm

- Input: Graph *G*, represented by adjacency lists
- Key: node ID *n*
- Value: node value *N*
  - *N.distance*: from start node *s* to *n*
  - *N.adjList*: $\left[\left(m,\ w(n,m)\right)\right]$, node id and weight of edge (*n*, *m*)
- Initialization:
  - For all *n*, $N.distance = \infty$

Shortest Path: From Intuition to Algorithm

- Mapper
  - $\forall m \in N.AdjList$: Emit $\left(m,\ d + w(n,\ m)\right)$
- Sort/Shuffle
  - Groups distances by reachable nodes
- Reducer:
  - Selects minimum distance path for each reachable node
  - Additional book-keeping needed to keep track of actual path

Shortest Path: Mapper

- Map (nid *n*, nvalue *N*)
  - $d \leftarrow N.distance$
  - $emit(n,\ N)$
  - For each $(m,\ w) \in N.AdjList$
    - $emit\left(m,\ d + w(n,\ m)\right)$
- Parallel Processing
  - All nodes are processed in parallel, each by a mapper
  - $emit(n,\ N)$ preserve graph structure for iterative processing
  - For each node m adjacent to n, emit a revised distance via n

Shortest Path: Reducer

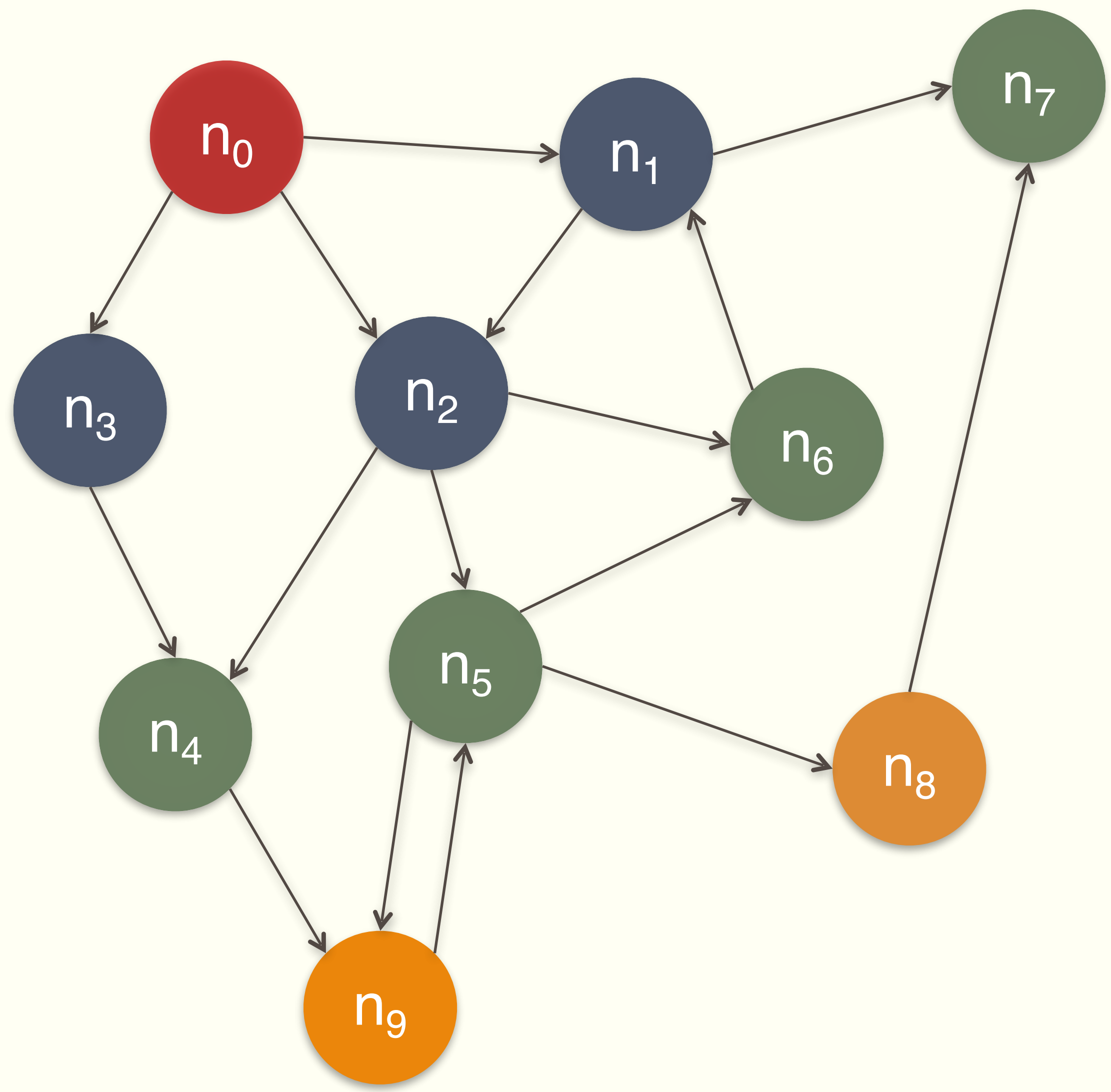
- Reduce (nid *m*, list[*d1*,*d2…*])
  - $d_{min} \leftarrow \infty$
  - $\forall d \in \left[d_1,\ d_2, \cdots\right]$
    - If d is Node
      - $M \leftarrow d$
    - Else
      - If $d < d_{min}$ then $d_{min} \leftarrow d$
  - $M.distance \leftarrow d_{min}$
  - $emit(m,\ M)$

- list for *m*:
  - distances from all predecessors so far
  - Node value *M*: must exist (from Mapper)

Iteration and Termination

- Each MapReduce iteration advances the “known frontier” by one hop
  - Subsequent iteration include more and more reachable nodes as frontier expands
  - Multiple iterations are needed to explore entire graph
- Termination: when the intermediate result no longer changes
  - Controlled by a non-MapReduce Driver
  - Use a flag – inspected by a non-MapReduce Driver

Visualizing Parallel BFS



Efficiency

- MapReduce explores all path in parallel
- Each MapReduce iteration advances the “unknown frontier” by one hop
  - Redundant work, since useful is only done at the “frontier”
- Dijkstra’s algorithm can be more efficient
  - At any step, it only pursues edges from the minimum-cost path inside the frontier

MapReduce: A Closer Look

- Data partitioned parallelism
  - Local computation at each node in mapper, in parallel:
    - Attributes of the node, adjacent edges and local link structure
  - Propagating Computations
    - Transversing the graph, this may involve iterative MapReduce
- Tips
  - Adjacency lists
  - Local computation in mapper
  - Pass along partial results via outlinks, keyed by destination node
  - Perform aggregation in reducer on inlinks to a node
  - Iterate until convergence: controlled by external “driver”
  - Pass graph structures between iterations
- Need a way to test for convergence!

MapReduce: PageRank

- The likelihood that a page v is visited by a random walk:

$$\alpha\left(\frac{1}{|V|}\right) + (1-\alpha)\sum_{u \in L(v)} \frac{P(u)}{C(u)}$$

- Recursive computation:
  - For each page $v \in G$
    - Compute P(v) using P(u) for all $u \in L(v)$
  - Until
    - Converge: no changes to any P(v)
    - After a fixed number of iteration

PageRank: MapReduce Algorithm

- Input: Graph G, represented by adjacency lists

- Key: Node ID *n*
- Value: node value *N*:
  - Rank: the page rank of a node
  - AdjList: Adjacency list
- Simplified Version:

$$(1-\alpha)\sum_{u\in L(v)}\frac{P(u)}{C(u)}$$

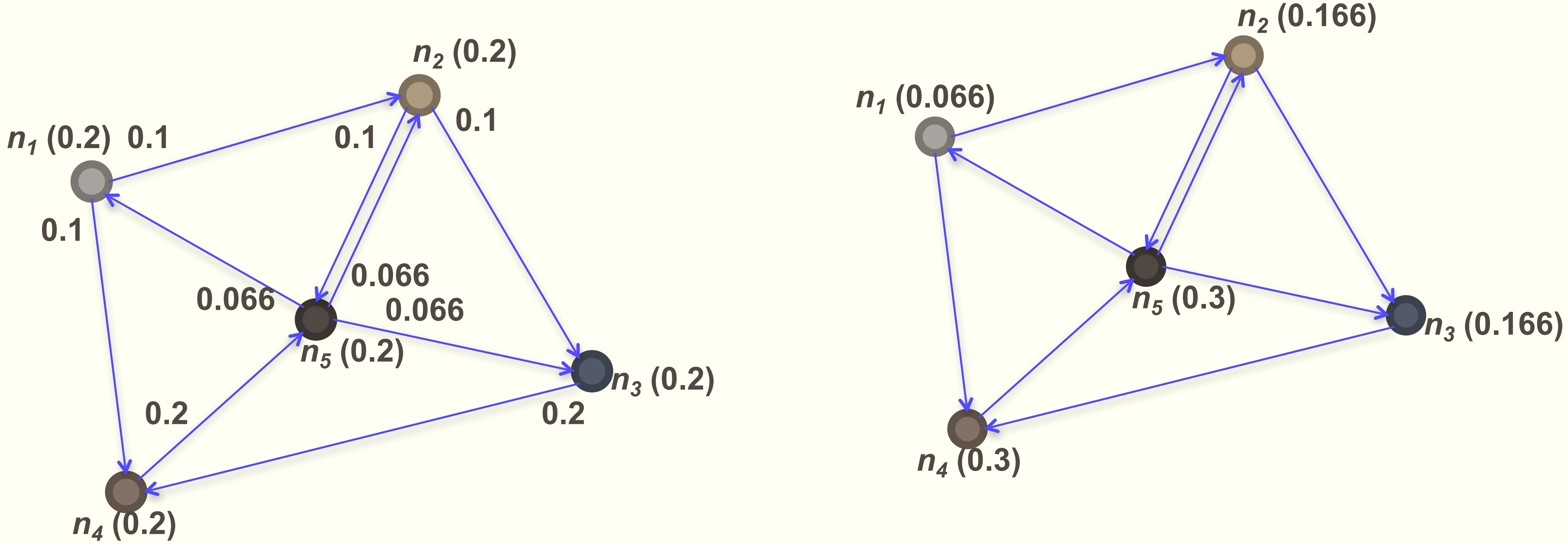
PageRank: Mapper

- Map (nid *n*, nvalue *N*)
  - $p \leftarrow N.rank / \left| N.AdjList \right|$
  - $emit(n,\ N)$
  - For each $(m,\ w) \in N.AdjList$
    - $emit\left(m,\ p\right)$
- Parallel Processing
  - All nodes are processed in parallel, each by a mapper
  - For each node m adjacent to n, emit PageRank contribution from n
  - $emit(n,\ N)$ preserve graph structure for iterative processing

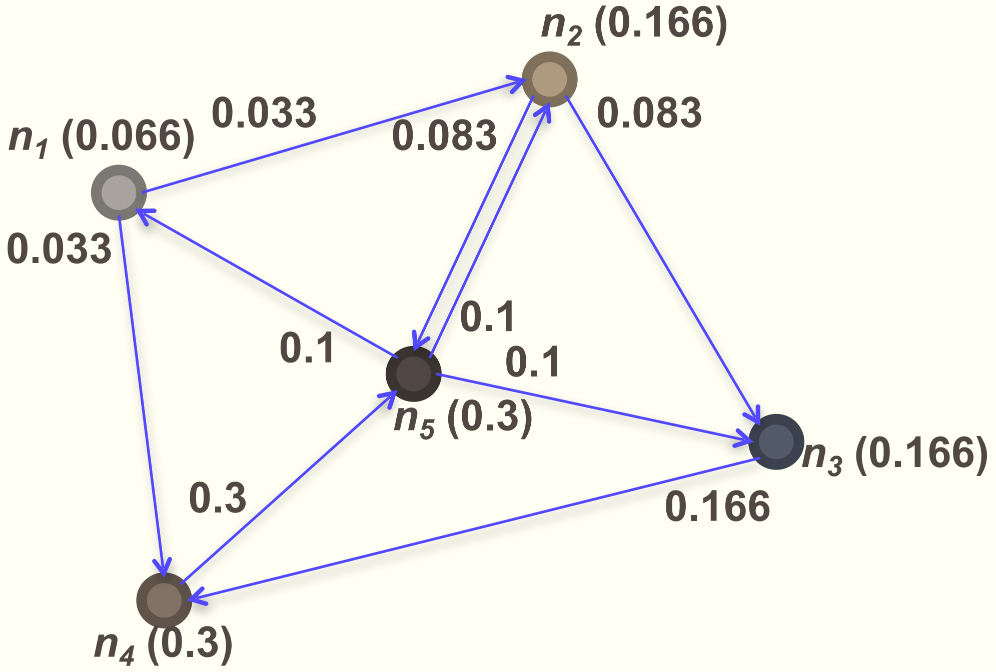
PageRank: Reducer

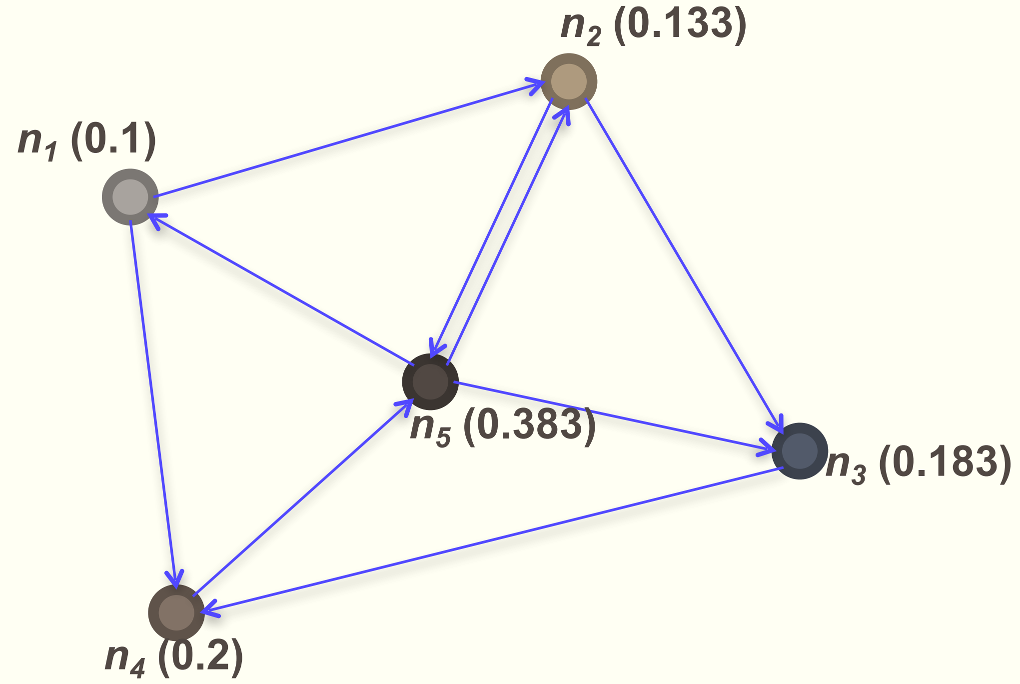
- Reduce (nid *m*, list[*p1*, *p2*,…])
  - $s \leftarrow 0$
  - $\forall p \in [p_1,\ p_2, \cdots]$
    - If p is Node
      - $M \leftarrow p$
    - Else
      - $s \leftarrow s + p$
  - $M.rank \leftarrow p$
  - $emit(m,\ M)$

Sample PageRank: Iteration 1

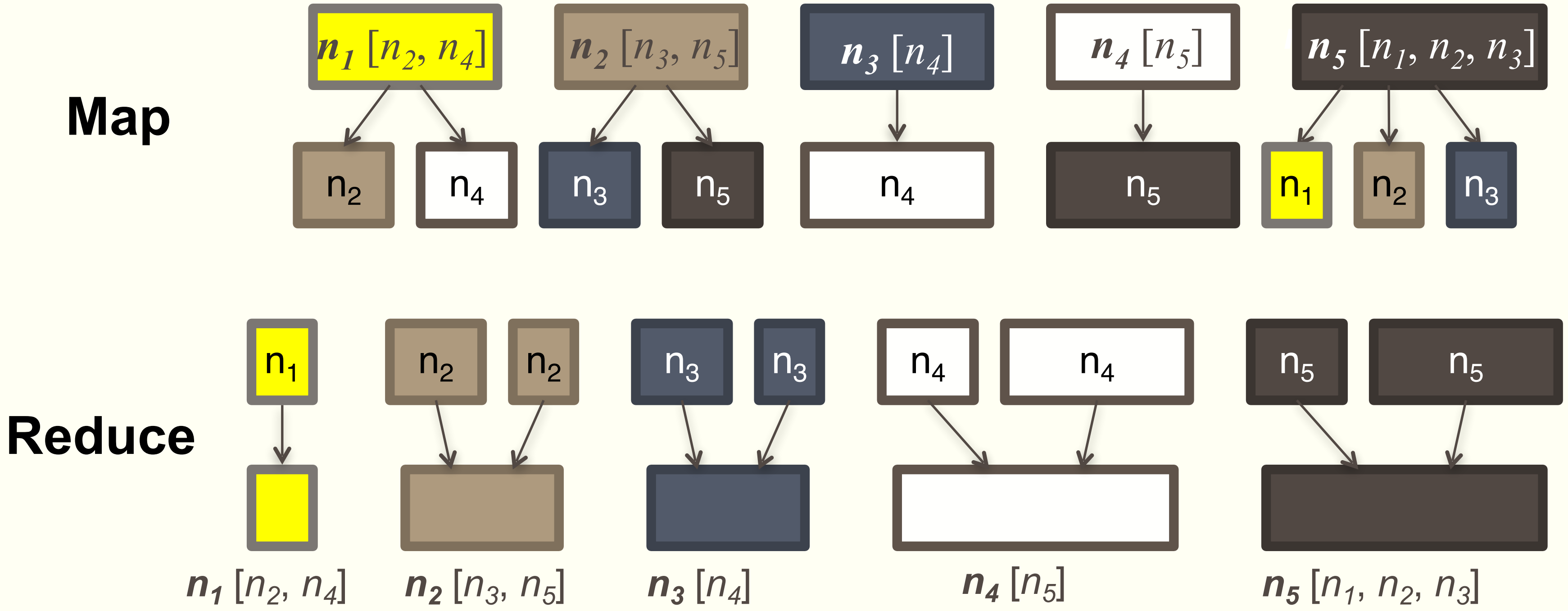


Sample PageRank: Iteration 2





PageRank in MapReduce



The need for parallel models beyond MapReduce

- Inefficiency:
  - Blocking
  - Intermediate result shipping (all to all)
  - Disk I/O in each step, even for invariant data in a loop
- Does not support iterative graph computation
  - Need for external driver
  - No mechanism to support global data structures that can be accessed and updated by all mappers and reducers
- Support for incremental Computation?
- Have to recast algorithm in MapReduce
- General model, not limited to graphs