Pregel: A system for large scale graph processing

Grzegorz Malewicz, Matthew H. Austern, Aart J. C. Bik,
James C. Dehnert, Ilan Horn, Naty Leiser, and Grzegorz Czajkwoski
Google, Inc.

Chathura Kankanamge
21st September 2016

Outline

Google's Graphs

- Web-scale graphs are becoming increasingly common
  - Tera or Peta byte data

powered by
TouchGraph

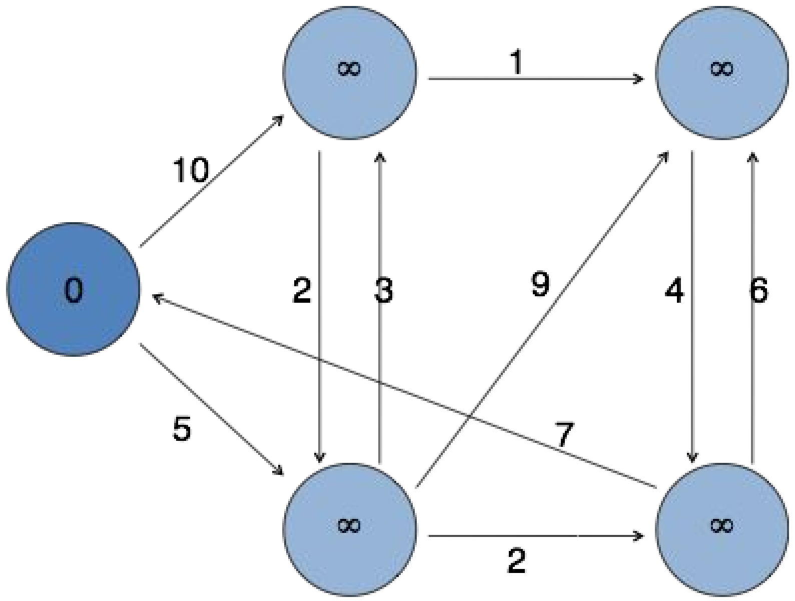
- ❏ Google Knowledge Graph
- ❏ Social Networks/ Web 2.0
- ❏ Maps

Processing large graphs - Issues

- Little locality of memory access
    - Neighbourhood traversing even worse in distributed systems
- Little work per vertex
    - High data access to computation ratio
- Degree of parallelism changes
    - Hard to partition computation

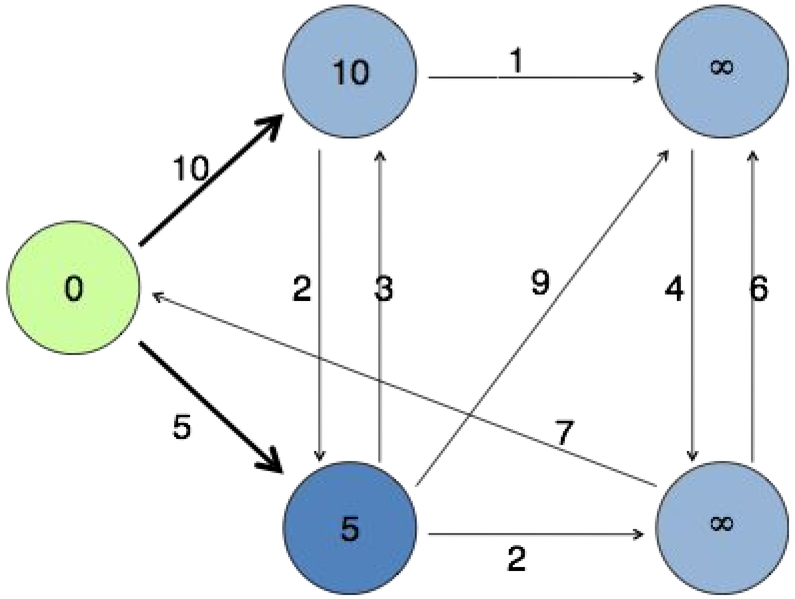
SSSP - Dijkstra’s Algorithm

- Find shortest distance from ‘source’ vertex to all other vertices



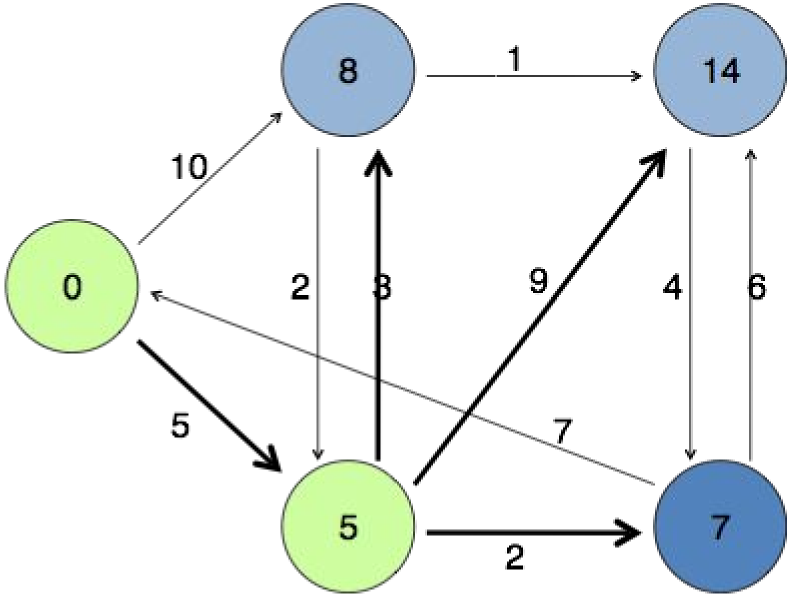
SSSP - Dijkstra's Algorithm

- Find shortest distance from 'source' vertex to all other vertices



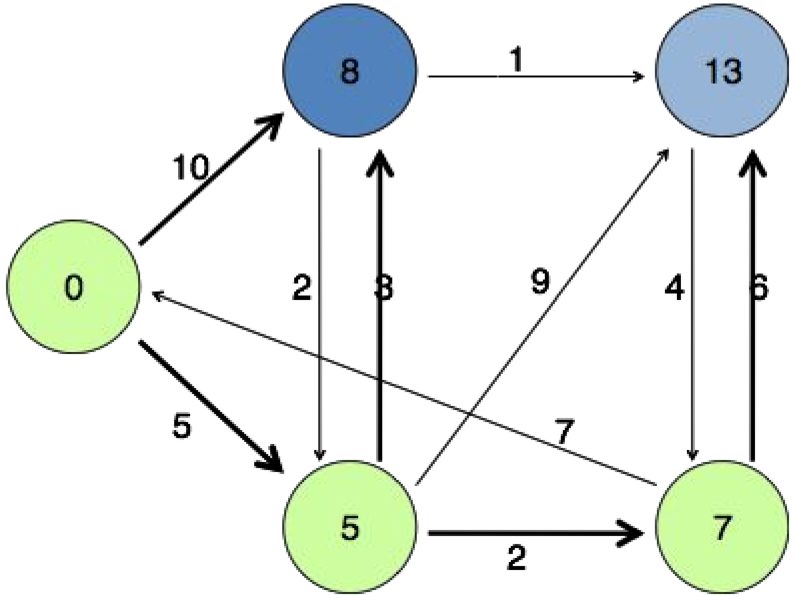
SSSP - Dijkstra's Algorithm

- Greedily find shortest distance from 'source' vertex to all other vertices



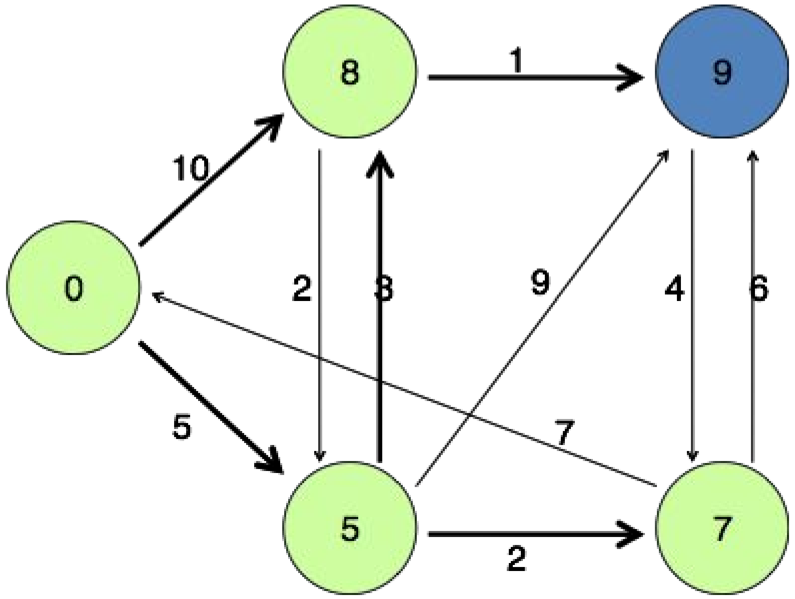
SSSP - Dijkstra’s Algorithm

- Find shortest distance from ‘source’ vertex to all other vertices



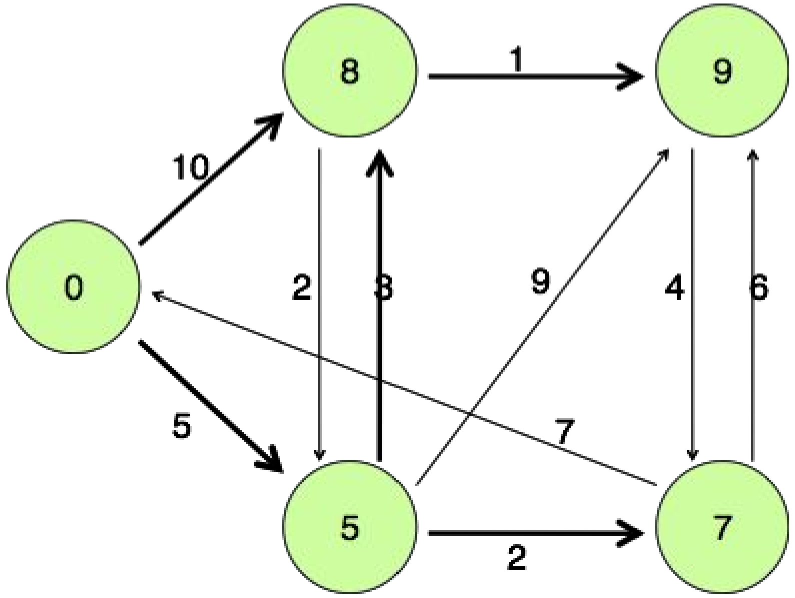
SSSP - Dijkstra’s Algorithm

- Find shortest distance from ‘source’ vertex to all other vertices

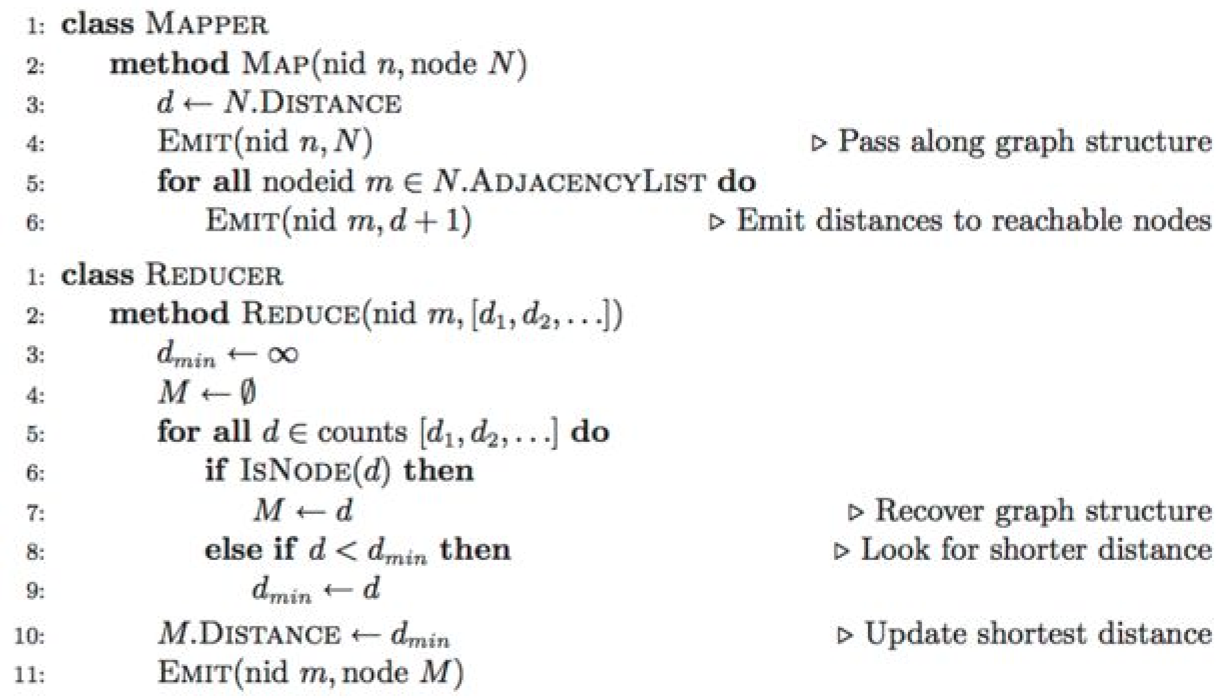


SSSP - Dijkstra’s Algorithm

- Find shortest distance from ‘source’ vertex to all other vertices



Processing SSSP in Hadoop



```
class MAPPER
    method MAP(nid $n$, node $N$)
        $d \leftarrow N$.DISTANCE
        EMIT(nid $n, N$)                                  ▷ Pass along graph structure
        for all nodeid $m \in N$.ADJACENCYLIST do
            EMIT(nid $m, d+1$)                   ▷ Emit distances to reachable nodes

class REDUCER
    method REDUCE(nid $m, [d_1, d_2, \ldots]$)
        $d_{min} \leftarrow \infty$
        $M \leftarrow \emptyset$
        for all $d \in$ counts $[d_1, d_2, \ldots]$ do
            if ISNODE($d$) then
                $M \leftarrow d$                          ▷ Recover graph structure
            else if $d < d_{min}$ then                   ▷ Look for shorter distance
                $d_{min} \leftarrow d$
        $M$.DISTANCE $\leftarrow d_{min}$                 ▷ Update shortest distance
        EMIT(nid $m$, node $M$)
```

From Lin & Dyer. (2010) Data-Intensive Text Processing with MapReduce

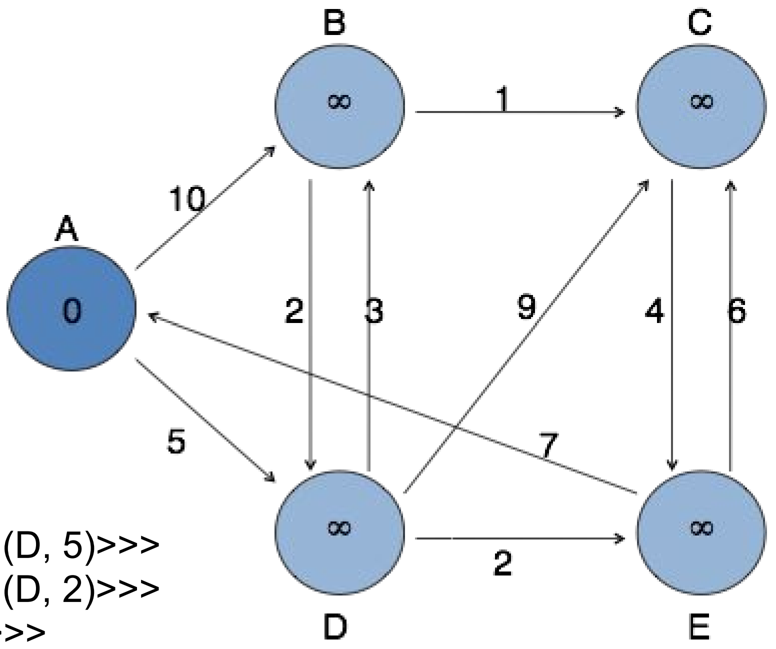
- Parallelized BFS in Map-Reduce
- Lot of intermediate disk writes

SSSP in Hadoop - Iteration 1

Map input: <node ID, <dist, adj list>>
<A, <0, <(B, 10), (D, 5)>>>
<B, <inf, <(C, 1), (D, 2)>>>
<C, <inf, <(E, 4)>>>
<D, <inf, <(B, 3), (C, 9), (E, 2)>>>
<E, <inf, <(A, 7), (C, 6)>>>

Map output 1: <dest node ID, dist>
<B, 10> <D, 5>
<C, inf> <D, inf>
<E, inf>
<B, inf> <C, inf> <E, inf>
<A, inf> <C, inf>

<A, <0, <(B, 10), (D, 5)>>>
<B, <inf, <(C, 1), (D, 2)>>>
<C, <inf, <(E, 4)>>>
<D, <inf, <(B, 3), (C, 9), (E, 2)>>>
<E, <inf, <(A, 7), (C, 6)>>>



Flushed to Local Disk

SSSP in Hadoop - Iteration 1

Reduce input: <node ID, dist>
<A, <0, <(B, 10), (D, 5)>>>
<A, ~~inf~~>

Get node from local disk

Min distances and update node

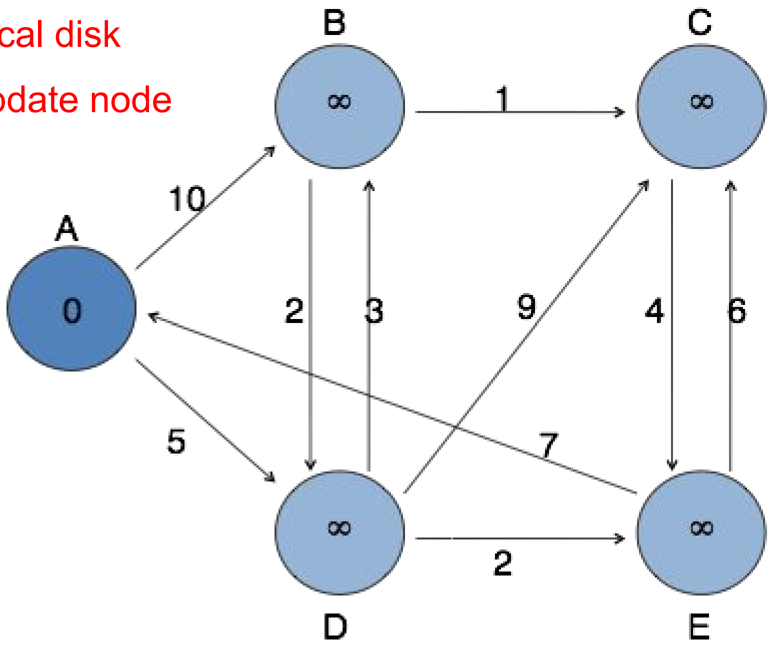
<B, <inf, <(C, 1), (D, 2)>>>
<B, 10> <B, ~~inf~~>

<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~~>

Output to HDFS

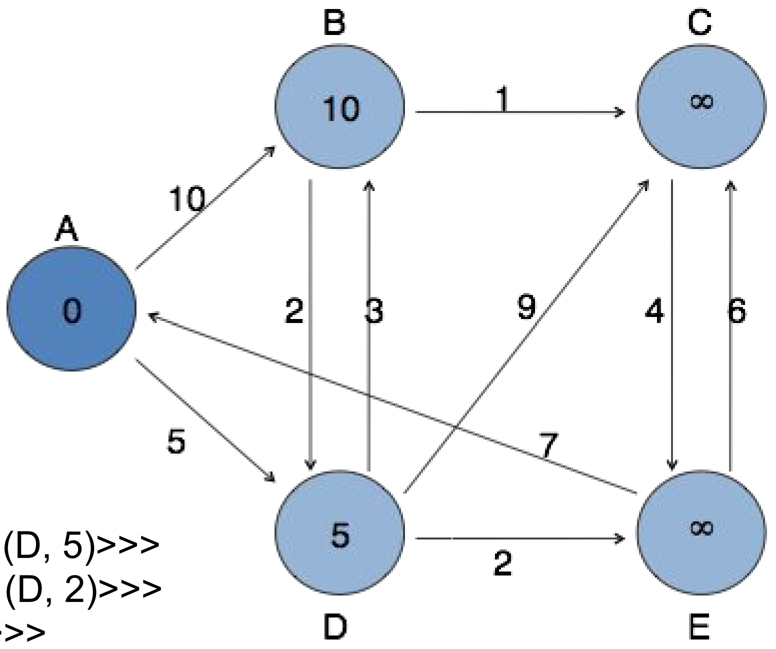


SSSP in Hadoop - Iteration 2

Map input: <node ID, <dist, adj list>>
<A, <0, <(B, 10), (D, 5)>>>
<B, <10, <(C, 1), (D, 2)>>>
<C, <inf, <(E, 4)>>>
<D, <5, <(B, 3), (C, 9), (E, 2)>>>
<E, <inf, <(A, 7), (C, 6)>>>

Map output 1: <dest node ID, dist>
<B, 10> <D, 5>
<C, 11> <D, 12>
<E, inf>
<B, 8> <C, 14> <E, 7>
<A, inf> <C, inf>

<A, <0, <(B, 10), (D, 5)>>>
<B, <10, <(C, 1), (D, 2)>>>
<C, <inf, <(E, 4)>>>
<D, <5, <(B, 3), (C, 9), (E, 2)>>>
<E, <inf, <(A, 7), (C, 6)>>>



Flushed to Local Disk

SSSP in Hadoop - Iteration 2

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~~>

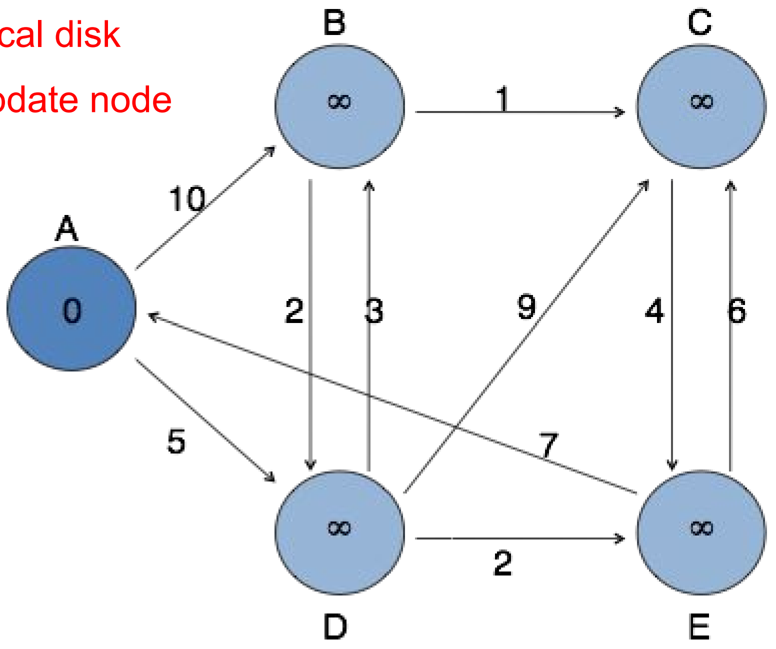
<D, <5, <(B, 3), (C, 9), (E, 2)>>>
<D, 5> <D, ~~12~~>

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

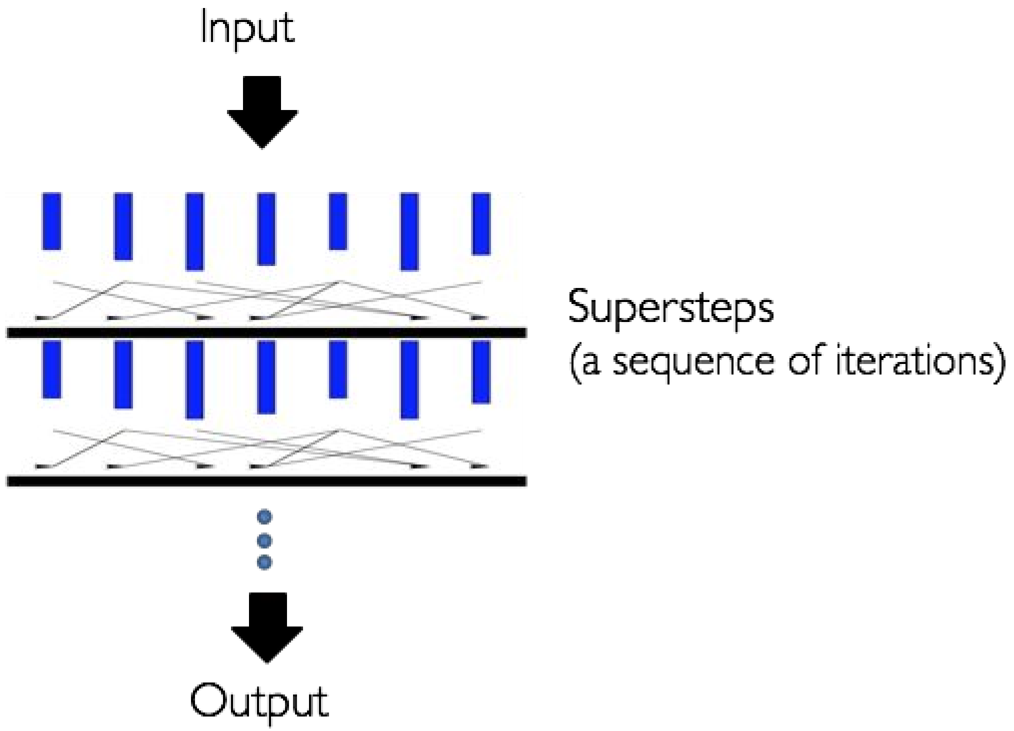
Get node from local disk

Min distances and update node

Output to HDFS

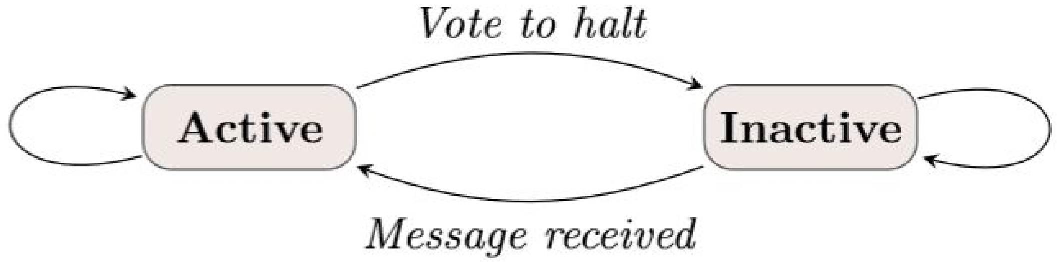


Pregel



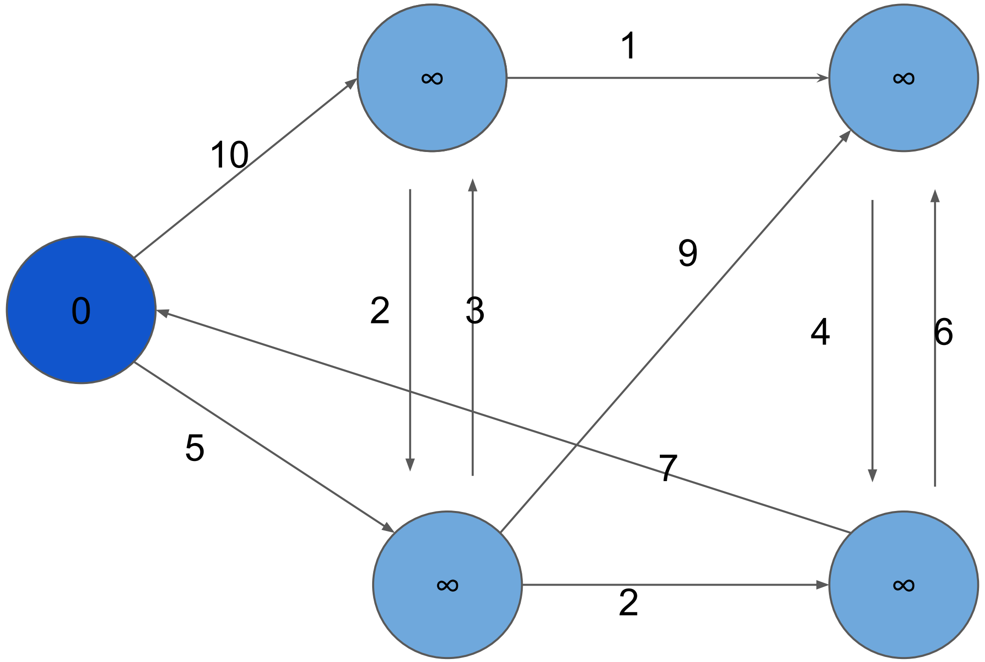
- BSP Model
  - Message Passing
  - Synchronization Barriers
- Think like a vertex!

Computation Model

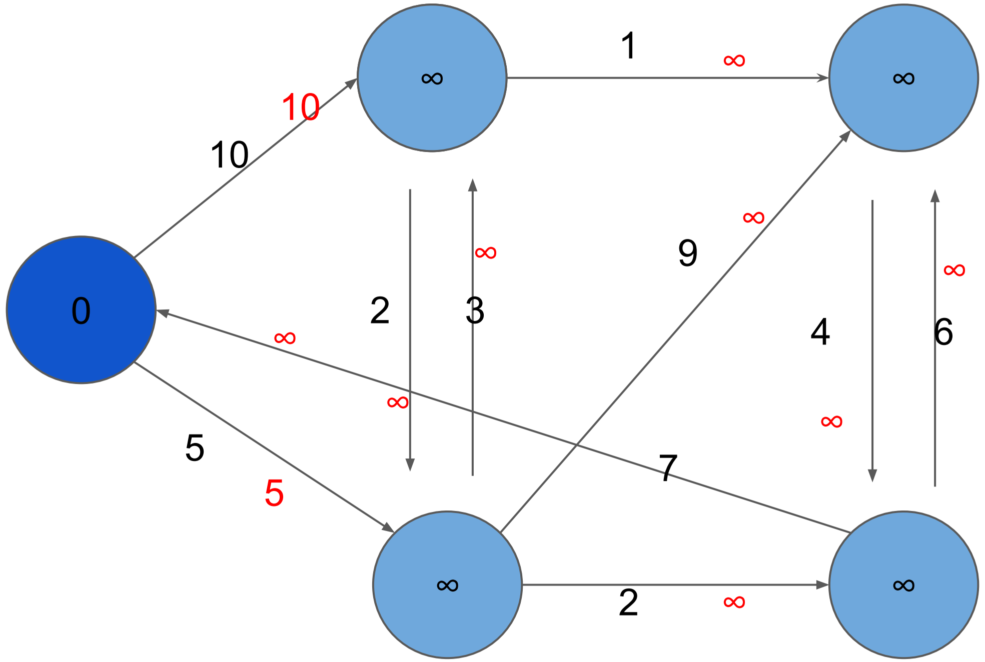


- Input - Directed Graph
  - vertexID + user defined value
  - Set of edges of source node
- In each Superstep: Vertices Compute in Parallel
  - Modify its state
  - Send and receive messages
  - Modify the topology of the graph
- Message Passing - No shared memory (expensive over network)
- Completion - All vertices have *voted to halt* when some condition is reached
- Output - Directed graph with new values

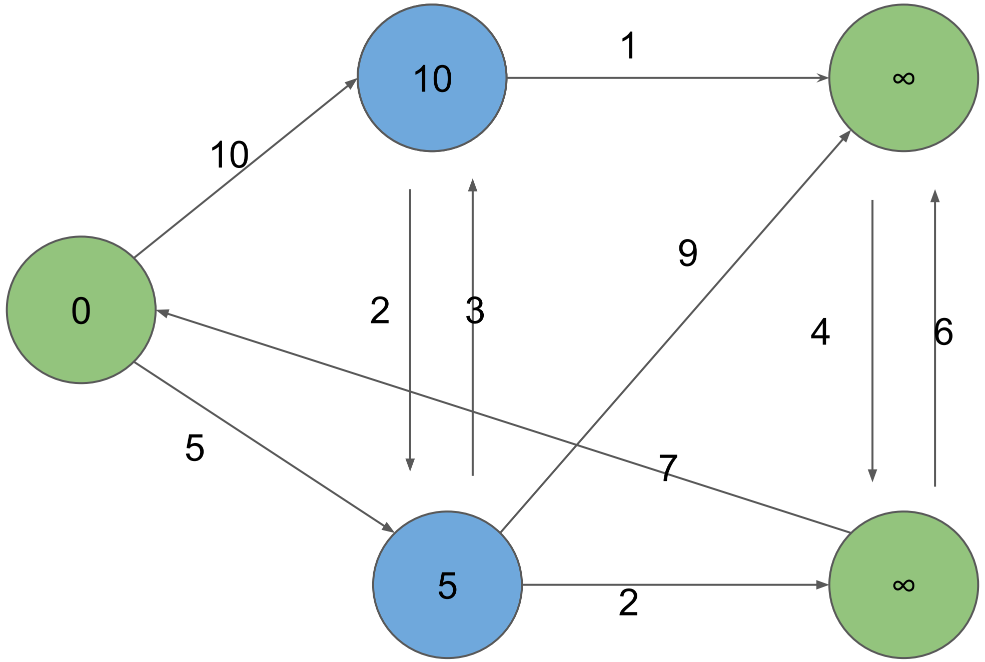
SSSP on Pregel -Super Step 1



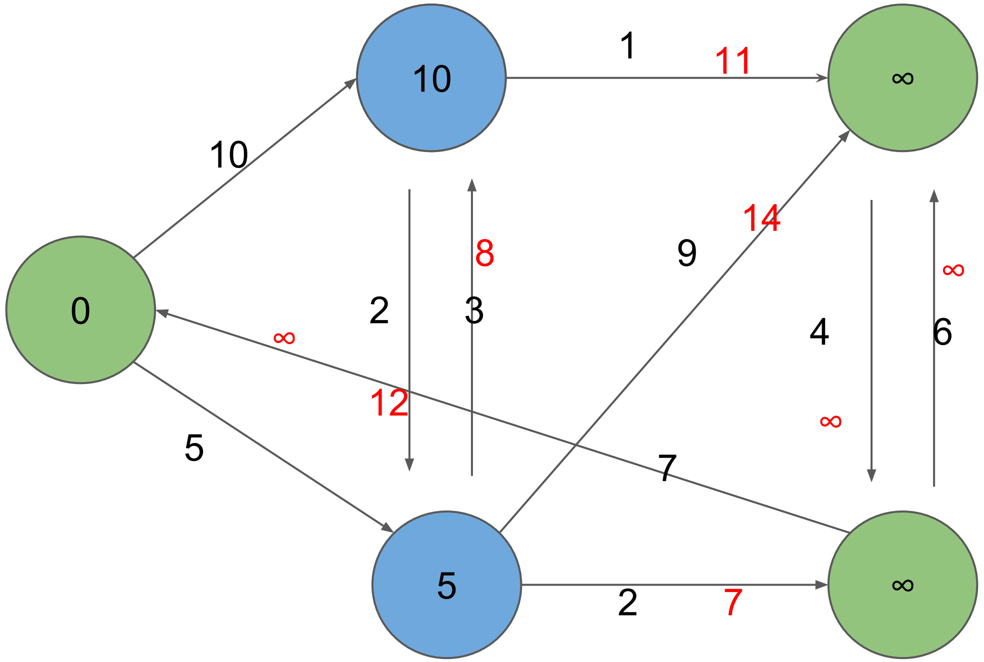
SSSP on Pregel - Super Step 1



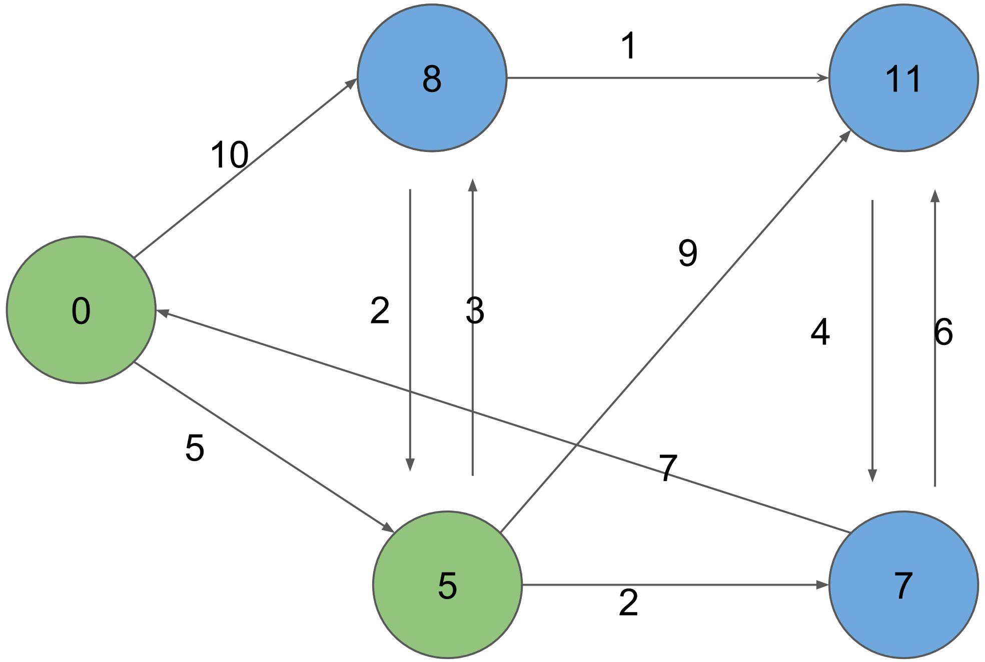
SSSP on Pregel - Super Step 2



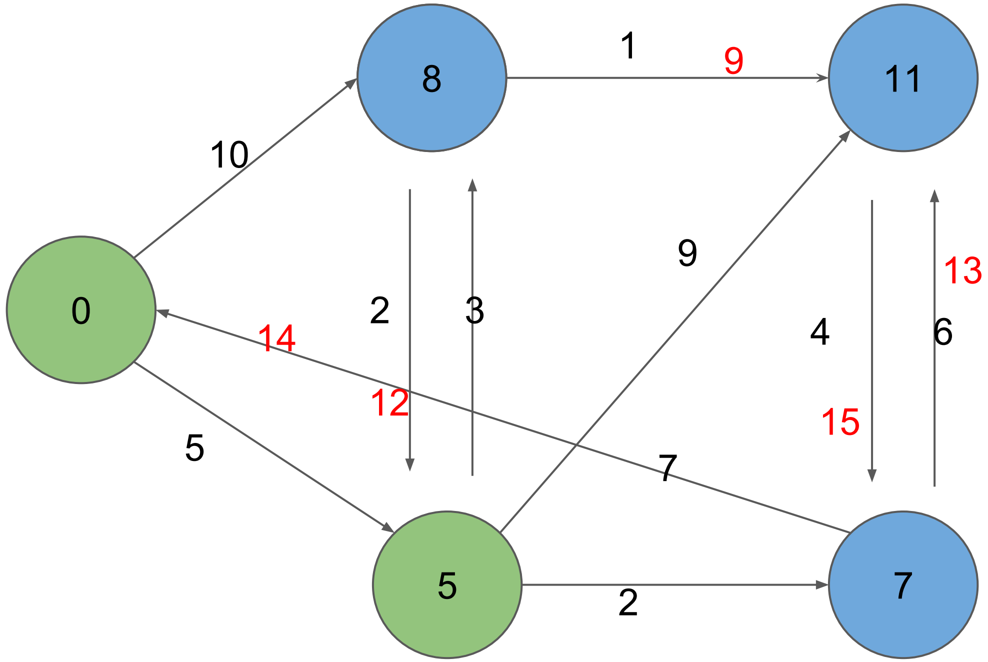
SSSP on Pregel - Super Step 2



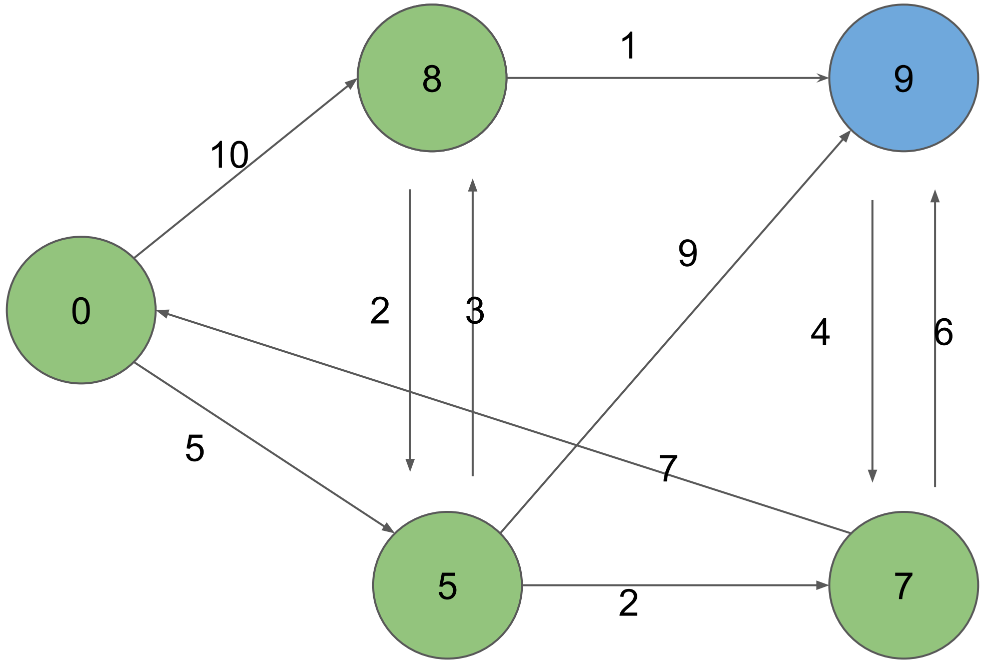
SSSP on Pregel - Super Step 3



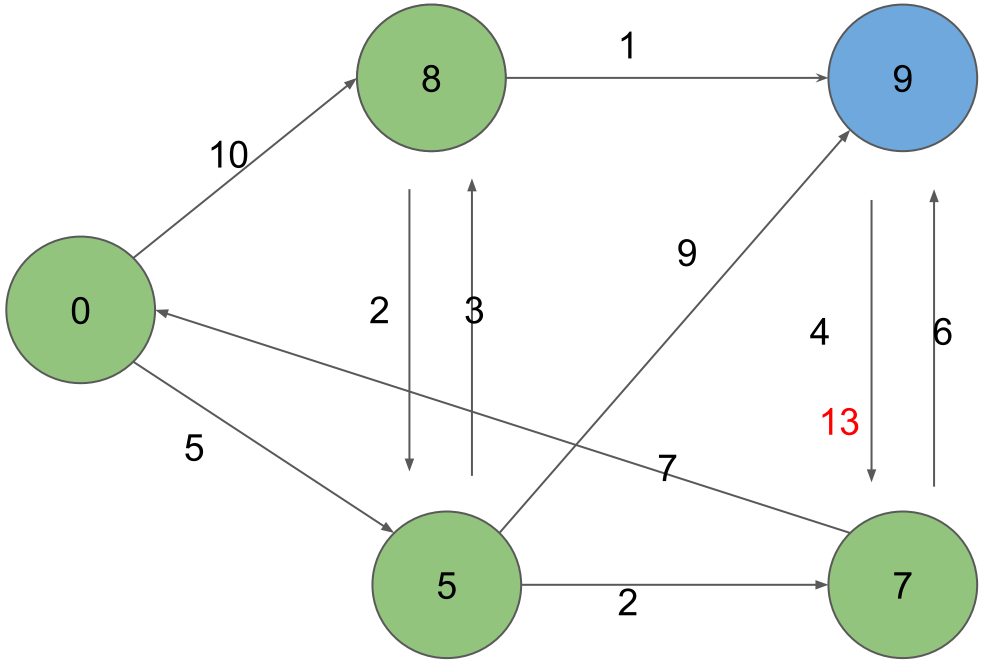
SSSP on Pregel - Super Step 3



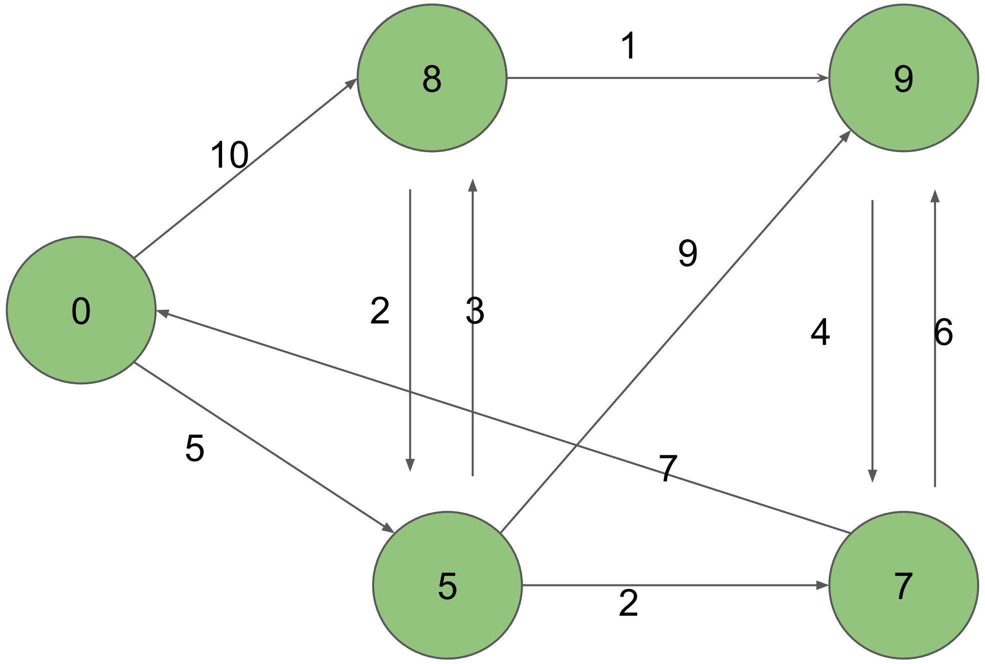
SSSP on Pregel - Super Step 4



SSSP on Pregel - Super Step 4



SSSP on Pregel - Super Step 5



Pregel API

```
class ShortestPathVertex
    : public Vertex<int, int, int> {

  void Compute(MessageIterator* msgs) {
    int mindist = IsSource(vertex_id()) ? 0 : INF;
    for (; !msgs->Done(); msgs->Next())
      mindist = min(mindist, msgs->Value());

    if (mindist < GetValue()) {
        *MutableValue() = mindist;
        OutEdgeIterator iter = GetOutEdgeIterator();
        for (; !iter.Done(); iter.Next())
          SendMessageTo(iter.Target(),
                        mindist + iter.GetValue());
    } else {
       VoteToHalt();
    }
  }
};
```

Messages Received

Compute function

Message Sending

Voting to halt

Pregel API Components

- Message Passing
  - Can send to any vertex (as long as vertexID is known)
  - Usually Iterate over edges and send messages
- Combiners
  - Reduce communication by combining messages
  - User defined - Override combiner class
- Aggregators
  - Global values for monitoring, communication and data
  - Vertices submit values to each global aggregator
  - Reduction operator (min, max etc) is applied to all the values for each aggregator
  - Resulting value available in S+1

Implementation

- Follows Master slave architecture
- Vertices are partitioned using *Hash(vertexID) mod N*
  - User defined partitioning also possible
- Master assigns partitions to workers
  - Usually partitions > workers
- Each worker gets a complete list of assignments of all vertices
  - So each worker knows where to send messages

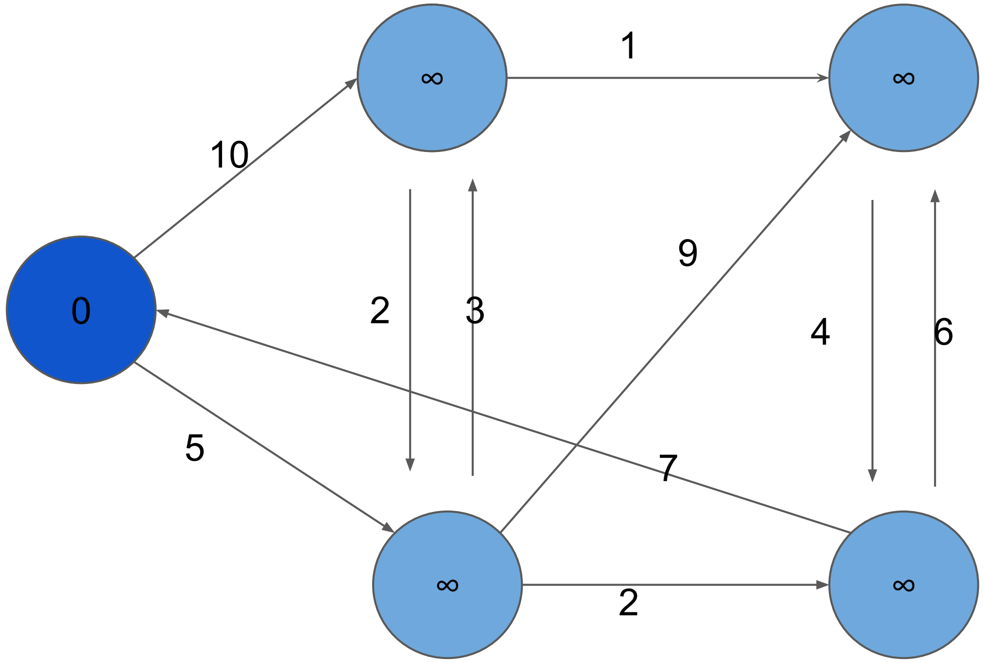
Master

- Responsible for coordinating workers
    - Input, output, computation, saving and recovering from checkpoints
- Keeps a list of live workers and their data
    - Worker UID
    - Worker’s addressing information
    - Graph partitions the worker has been assigned
- Barrier Synchronization
    - Send instruction to worker and wait for response
    - Enter recovery mode or proceed with super step
- Maintain statistics of graph (total size, distr. Of out-degrees, active vertices...)
    - Run HTTP server for user monitoring

Worker

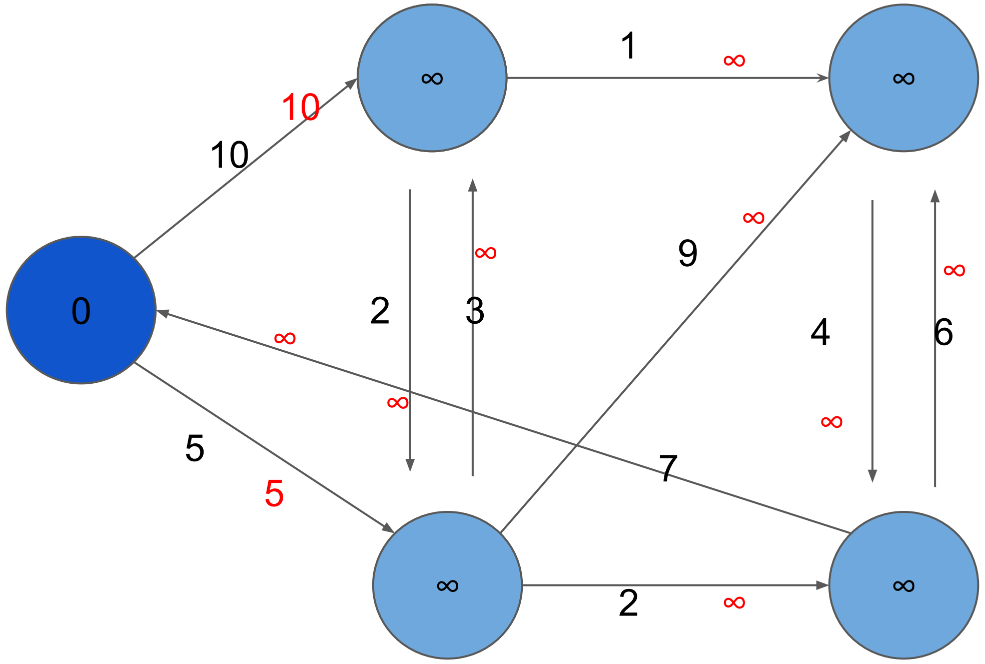
- Maintain graph in-memory
  - Vertices and outgoing edges
  - Incoming messages
  - Active flag
- Iterate through vertices and call compute()
  - Pass current value, incoming messages, outgoing edges
- Receive Messages for S+1 supersep
- Send Messages generated in compute()
  - Determine corresponding worker for vertex
  - Queue message for sending
- Apply combiners
  - Outgoing messages - save bandwidth
  - Incoming messages - save space

Fault Tolerance -Super Step 1

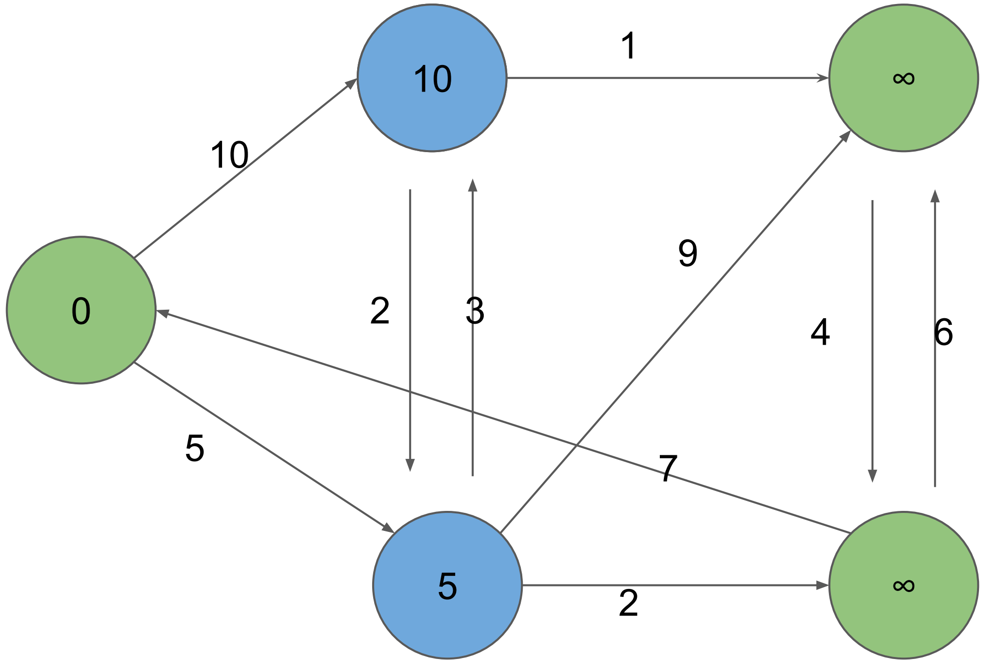


- Checkpointing!!
- Master pings workers
- Failure?
  - Recovery mode

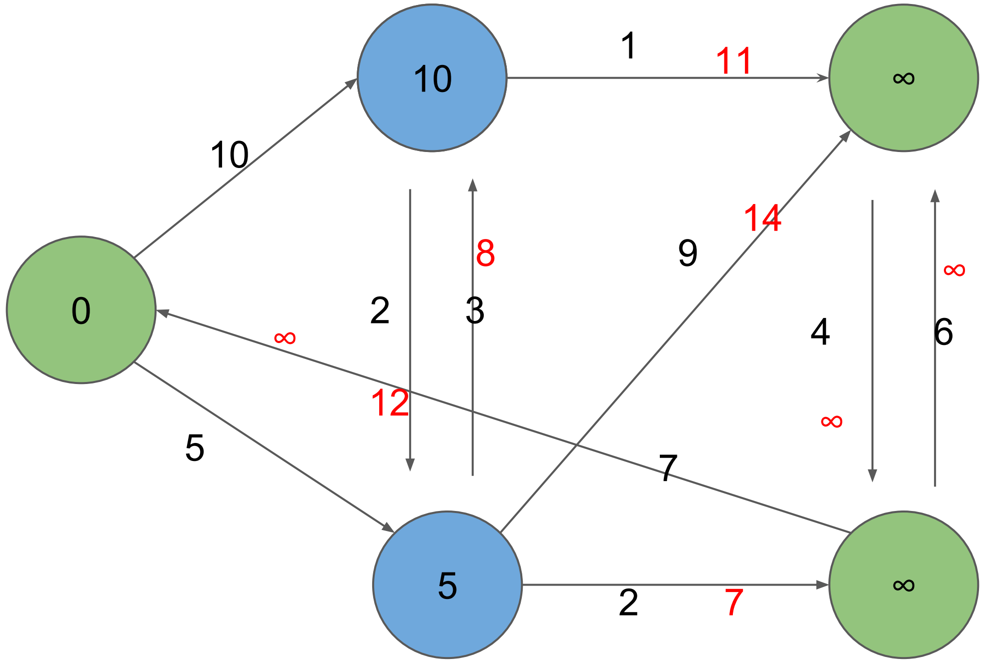
Fault Tolerance - Super Step 1



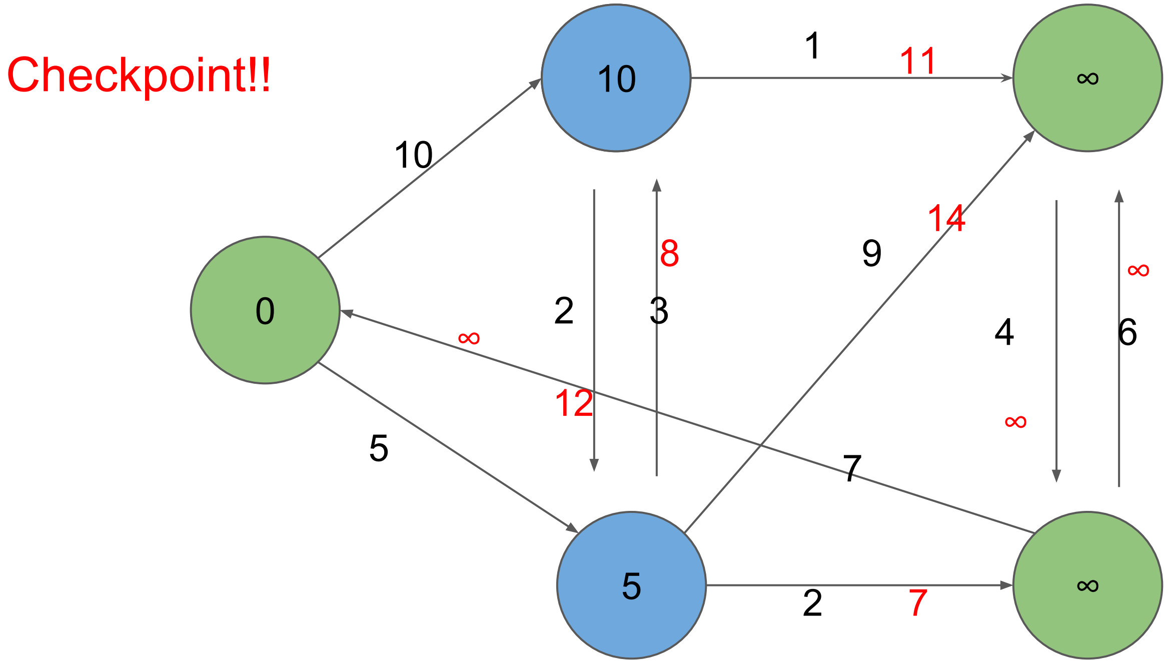
Fault Tolerance - Super Step 2



Fault Tolerance - Super Step 2

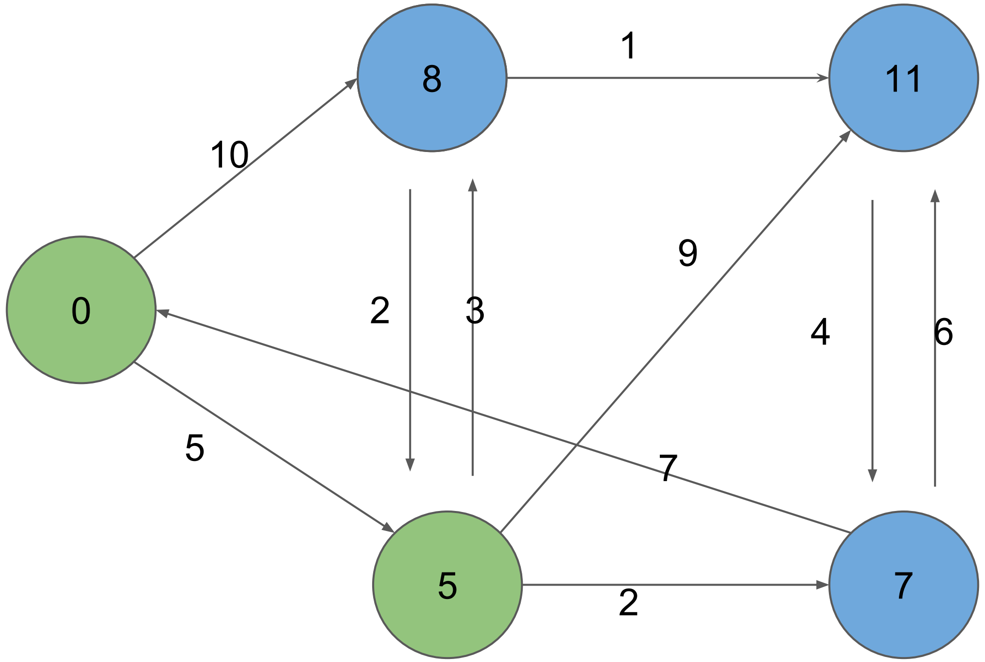


Fault Tolerance - Super Step 3

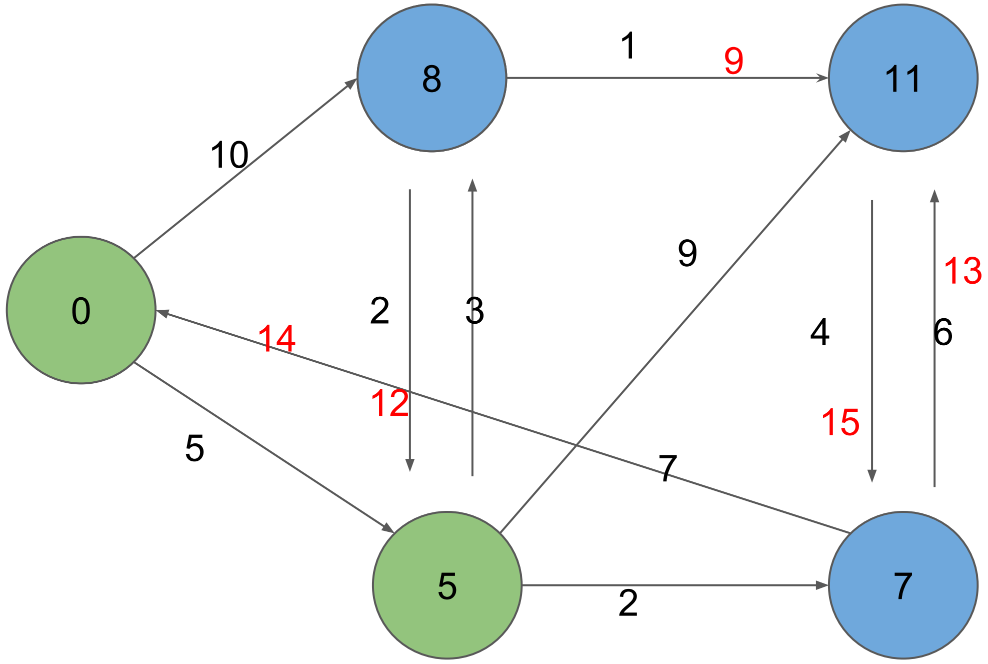


- Save to HDFS - VertexID,Out Edges, Values, Incoming messages

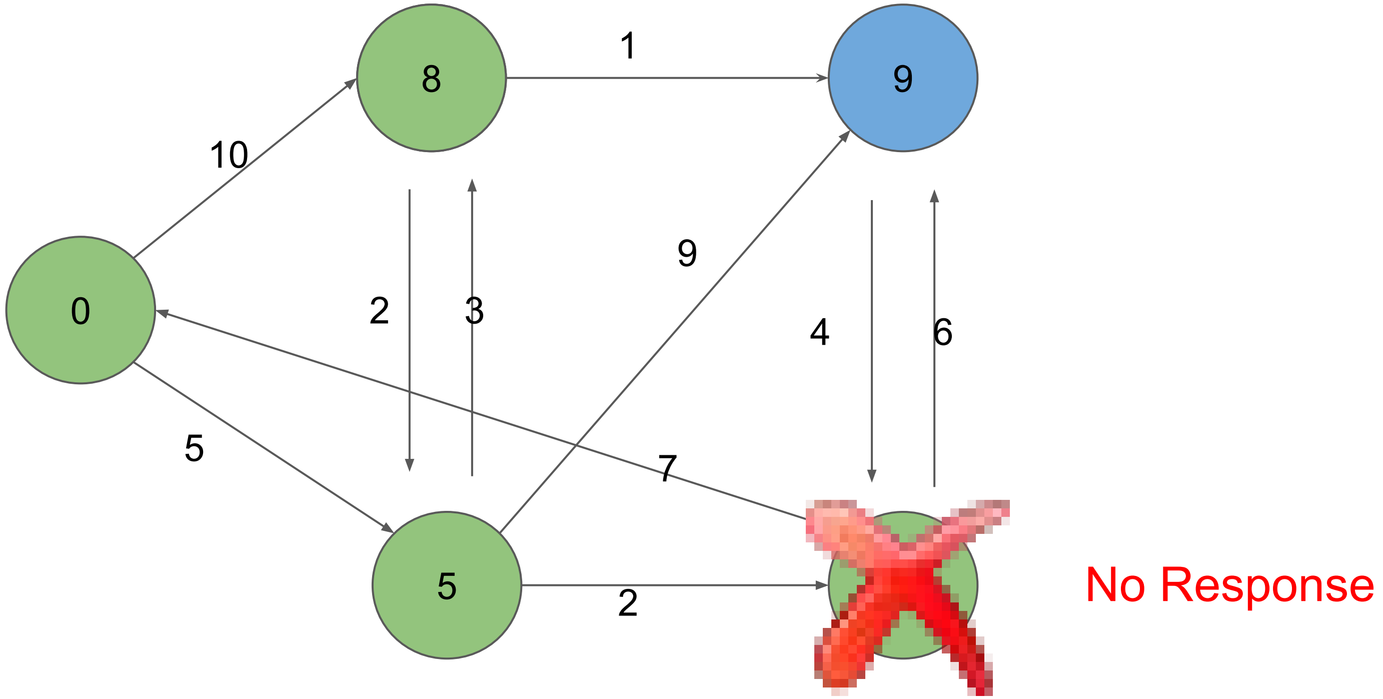
Fault Tolerance - Super Step 3



Fault Tolerance - Super Step 3

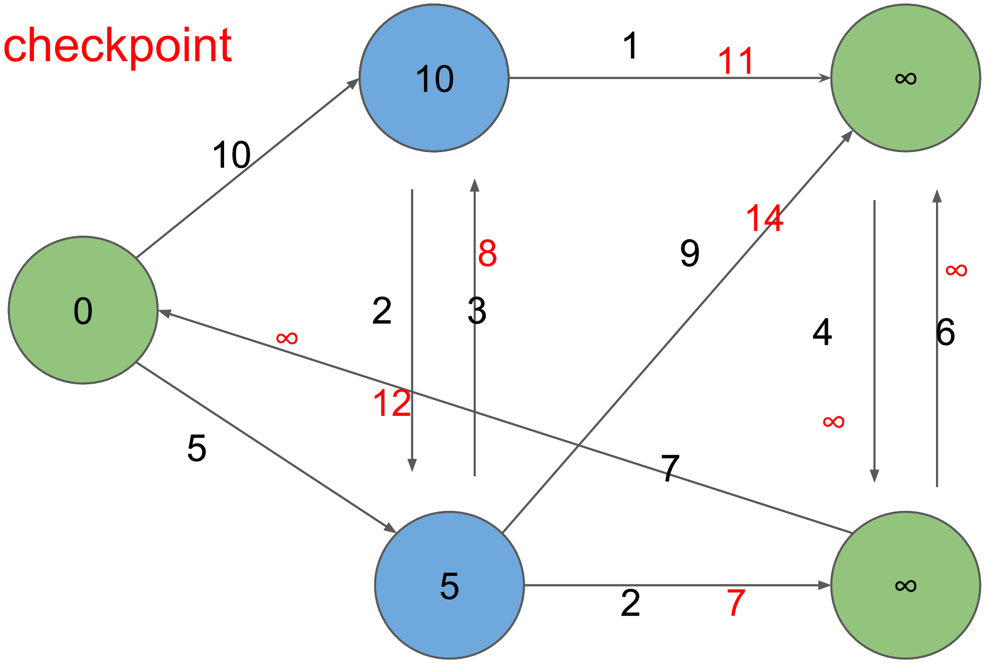


Fault Tolerance - Super Step 4

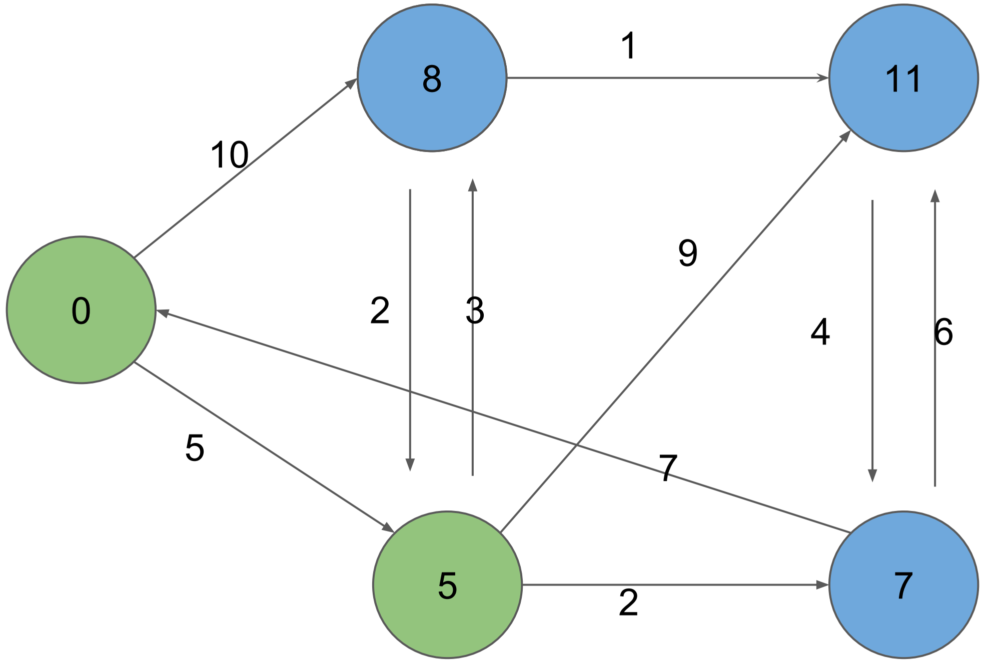


Fault Tolerance - Recovery Step 1

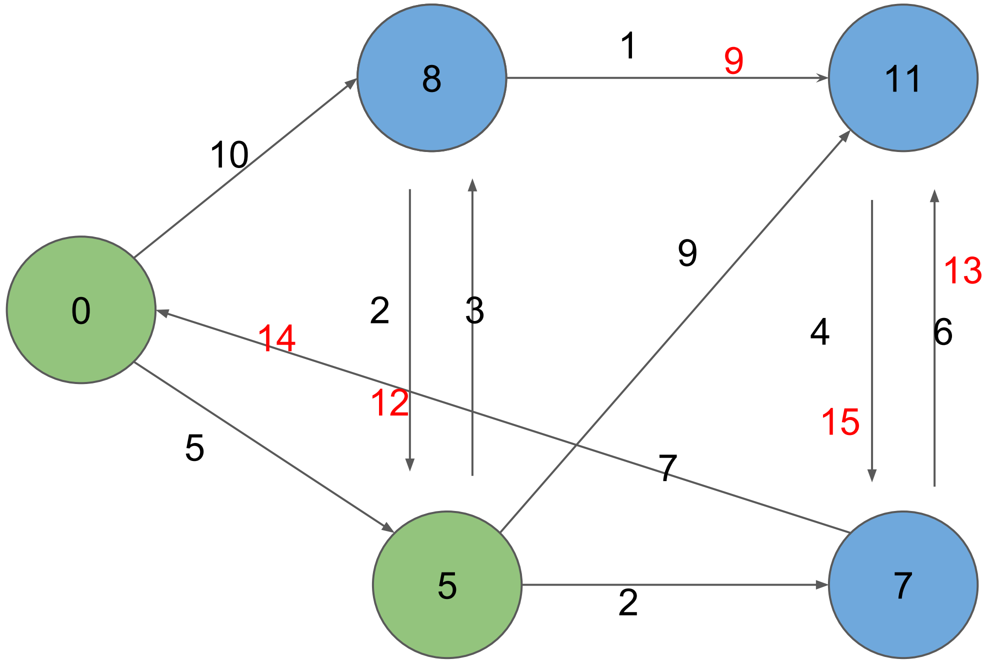
Reload from checkpoint



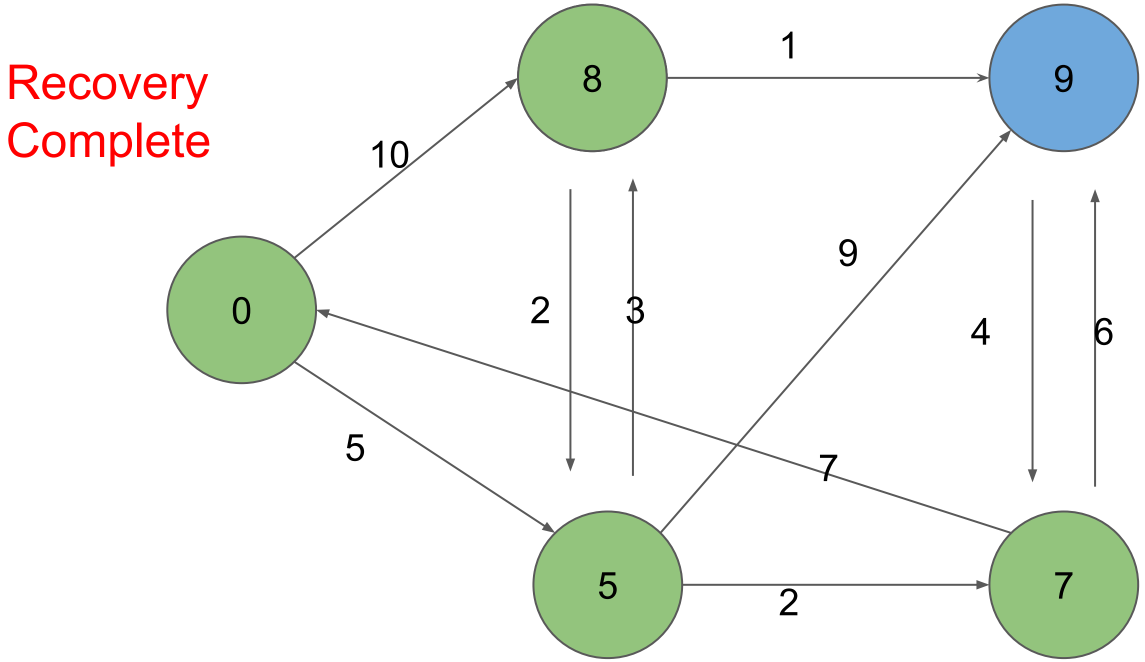
Fault Tolerance - Recovery Step 1



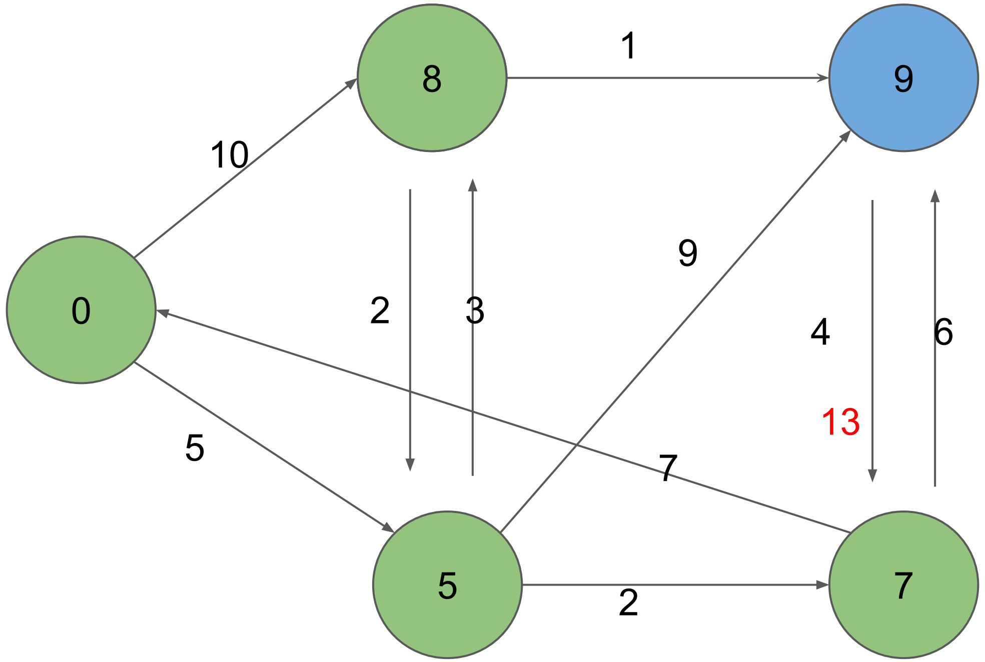
Fault Tolerance - Recovery Step 2



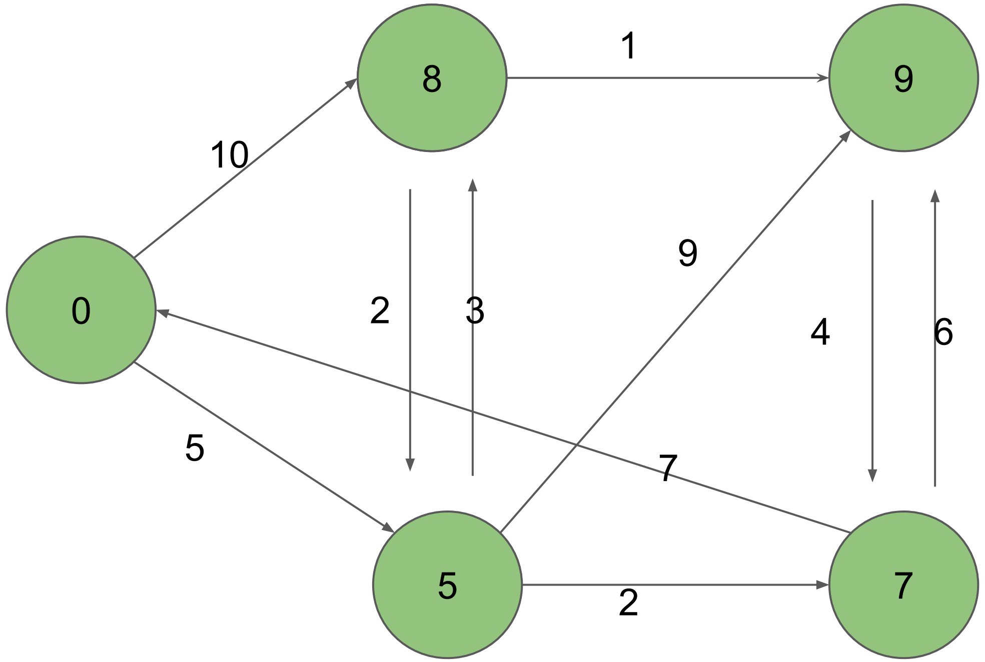
Fault Tolerance - Super Step 4



Fault Tolerance - Super Step 4



Fault Tolerance - Super Step 5



Research Problems

- How to address the problem of ‘skew’?
  - Partition edges among workers? (GPS)
- Optimizations done by subsequent systems like GPS
  - Dynamic partitioning
  - Master compute function
- Is the programming interface easy to maintain and reuse? Is it too low level?

References

1. Grzegorz Malewicz, Matthew H. Austern, Aart J.C Bik, James C. Dehnert, Ilan Horn, Naty Leiser, and Grzegorz Czajkowski. 2010. Pregel: a system for large-scale graph processing. In *Proceedings of the 2010 ACM SIGMOD International Conference on Management of data* (SIGMOD '10). ACM, New York, NY, USA, 135-146

2. Semih Salihoglu and Jennifer Widom. 2013. GPS: a graph processing system. In *Proceedings of the 25th International Conference on Scientific and Statistical Database Management* (SSDBM), Alex Szalay, Tamas Budavari, Magdalena Balazinska, Alexandra Meliou, and Ahmet Sacan (Eds.). ACM, New York, NY, USA, , Article 22 , 12 pages

3. Andrew Lumsdaine, Douglas Gregor, Bruce Hendrickson, and Jonathan W. Berry, Challenges in Parallel Graph Processing. Parallel Processing Letters 17, 2007, 5–20.

4. Minyang Han, Khuzaima Daudjee, Khaled Ammar, M. Tamer Özsu, Xingfang Wang, and Tianqi Jin. 2014. An experimental comparison of pregel-like graph processing systems. *Proc. VLDB Endow.* 7, 12 (August 2014), 1047-1058
5. Pregel:A System for Large-Scale Graph Processing PPT by Taewhi Lee, 7 July 2010