Map-Reduce Applications: Counting, Graph Shortest Paths

Overview

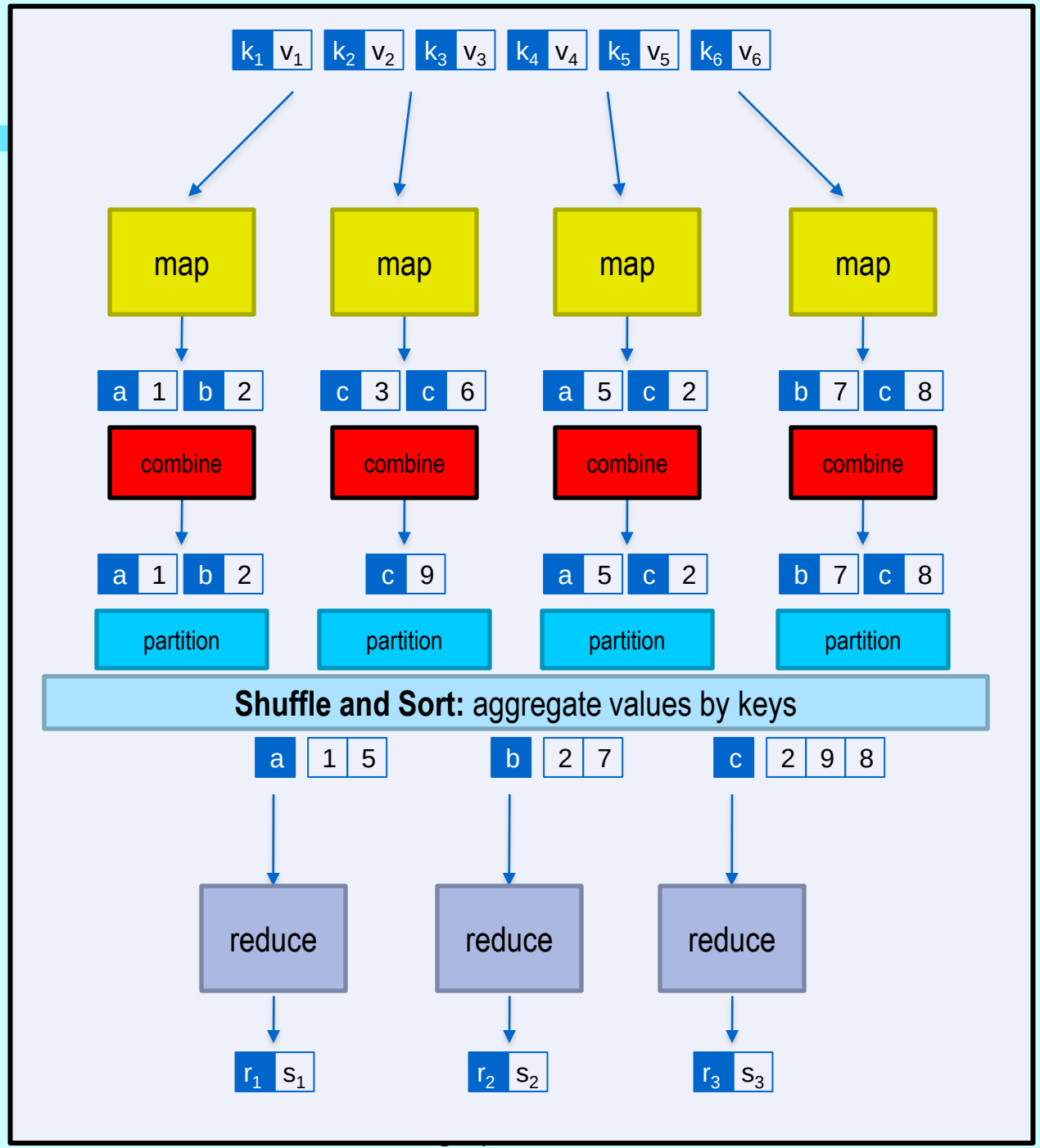
- Simple counting, averaging
- Graph problems and representations
- Parallel breadth-first search

MapReduce: Recap

- Programmers must specify:
  map (k, v) → <k', v'>*
  reduce (k', v') → <k', v'>*
  - All values with the same key are reduced together
- Optionally, also:
  partition (k', number of partitions) → partition for k'
  - Often a simple hash of the key, e.g., hash(k') mod n
  - Divides up key space for parallel reduce operations
  combine (k', v') → <k', v'>*
  - Mini-reducers that run in memory after the map phase
  - Used as an optimization to reduce network traffic
- The execution framework handles everything else…

"Everything Else"

- The execution framework handles everything else…
  - Scheduling: assigns workers to map and reduce tasks
  - "Data distribution": moves processes to data
  - Synchronization: gathers, sorts, and shuffles intermediate data
  - Errors and faults: detects worker failures and restarts
- Limited control over data and execution flow
  - All algorithms must expressed in m, r, c, p
- You don't know:
  - Where mappers and reducers run
  - When a mapper or reducer begins or finishes
  - Which input a particular mapper is processing
  - Which intermediate key a particular reducer is processing


k1 v1
k2 v2
k3 v3
k4 v4
k5 v5
k6 v6
map
map
map
map
a 1
b 2
c 3
c 6
a 5
c 2
b 7
c 8
combine
combine
combine
combine
a 1
b 2
c 9
a 5
c 2
b 7
c 8
partition
partition
partition
partition
Shuffle and Sort: aggregate values by keys
a 1 5
b 2 7
c 2 9 8
reduce
reduce
reduce
r1 s1
r2 s2
r3 s3

Word Count: Baseline

```
1: class Mapper
2:     method Map(docid a, doc d)
3:         for all term t ∈ doc d do
4:             Emit(term t, count 1)

1: class Reducer
2:     method Reduce(term t, counts [c1, c2, ...])
3:         sum ← 0
4:         for all count c ∈ counts [c1, c2, ...] do
5:             sum ← sum + c
6:         Emit(term t, count s)
```

What's the impact of combiners?

Word Count: Version 1

```
1: class Mapper
2:     method Map(docid a, doc d)
3:         H ← new AssociativeArray
4:         for all term t ∈ doc d do
5:             H{t} ← H{t} + 1                ▷ Tally counts for entire document
6:         for all term t ∈ H do
7:             Emit(term t, count H{t})
```

Word Count: Version 2

```
1: class Mapper
2:     method Initialize
3:         H ← new AssociativeArray
4:     method Map(docid a, doc d)
5:         for all term t ∈ doc d do
6:             H{t} ← H{t} + 1                ▷ Tally counts across documents
7:     method Close
8:         for all term t ∈ H do
9:             Emit(term t, count H{t})
```

Key: preserve state across input key-value pairs!

Design Pattern for Local Aggregation

- "In-mapper combining"
  - Fold the functionality of the combiner into the mapper by preserving state across multiple map calls
- Advantages
  - Speed
  - Faster than actual combiners
- Disadvantages
  - Explicit memory management required
  - Potential for order-dependent bugs

Combiner Design

- Combiners and reducers share same method signature
  - Sometimes, reducers can serve as combiners
  - Often, not…
- Remember: combiner are optional optimizations
  - Should not affect algorithm correctness
  - May be run 0, 1, or multiple times
- Example: find average of all integers associated with the same key

Computing the Mean: Version 1

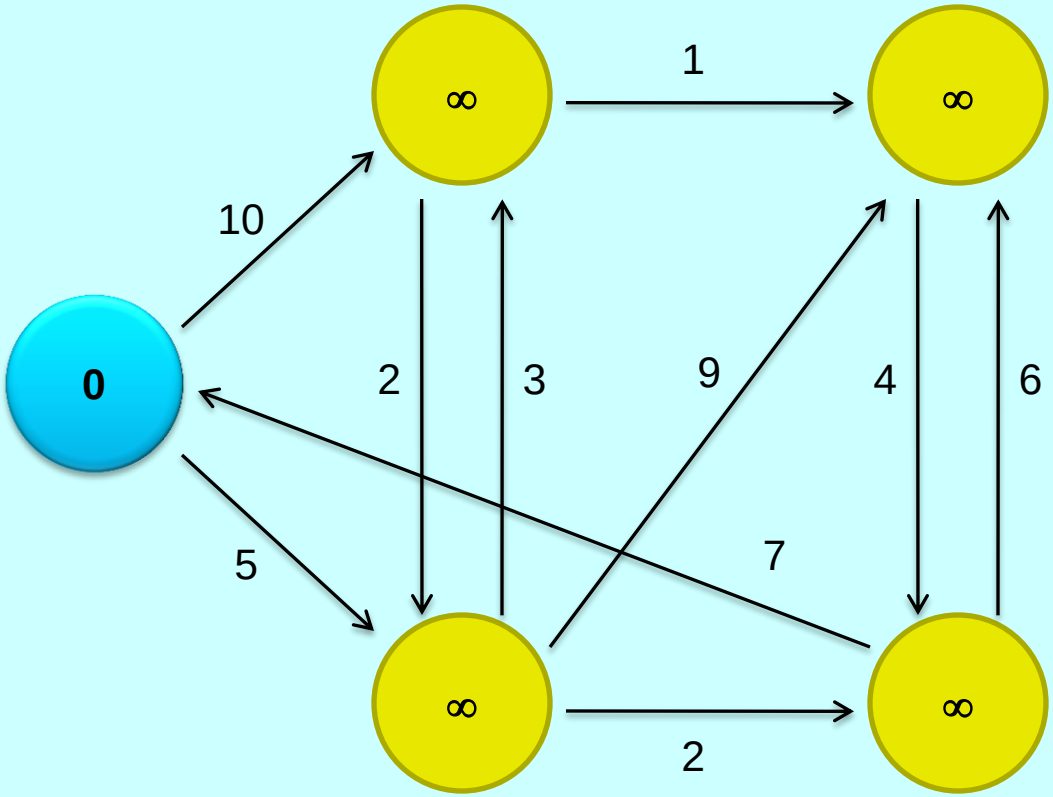
```
1: class Mapper
2:     method Map(string t, integer r)
3:         Emit(string t, integer r)

1: class Reducer
2:     method Reduce(string t, integers [r_1, r_2, ...])
3:         sum ← 0
4:         cnt ← 0
5:         for all integer r ∈ integers [r_1, r_2, ...] do
6:             sum ← sum + r
7:             cnt ← cnt + 1
8:         r_avg ← sum/cnt
9:         Emit(string t, integer r_avg)
```

Single Source Shortest Path

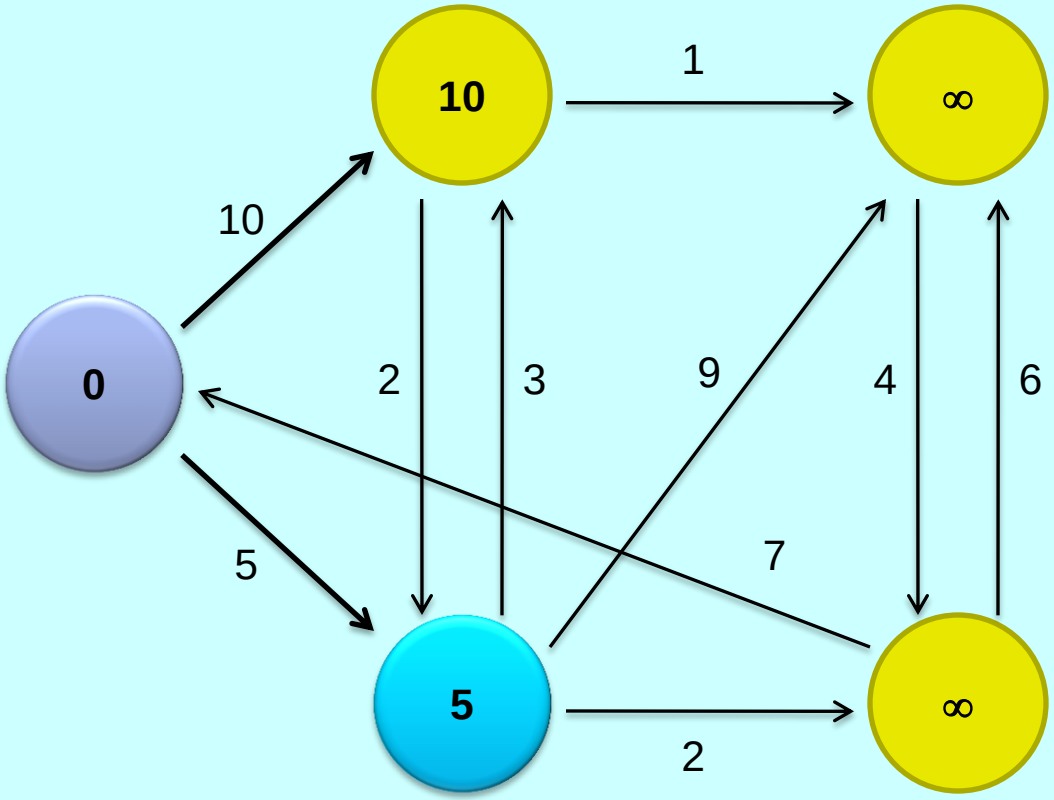
- Problem: find shortest path from a source node to one or more target nodes
  - Shortest might also mean lowest weight or cost
- First, a refresher: Dijkstra's Algorithm

Dijkstra's Algorithm Example



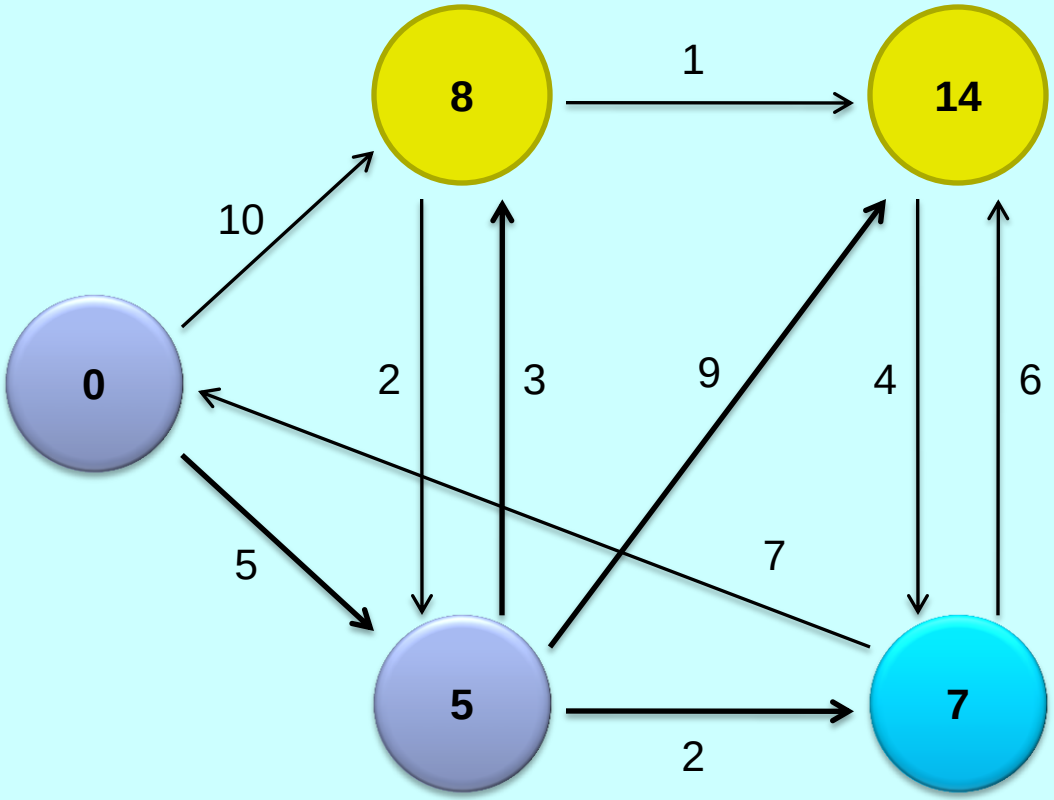
Example from CLR

Dijkstra's Algorithm Example



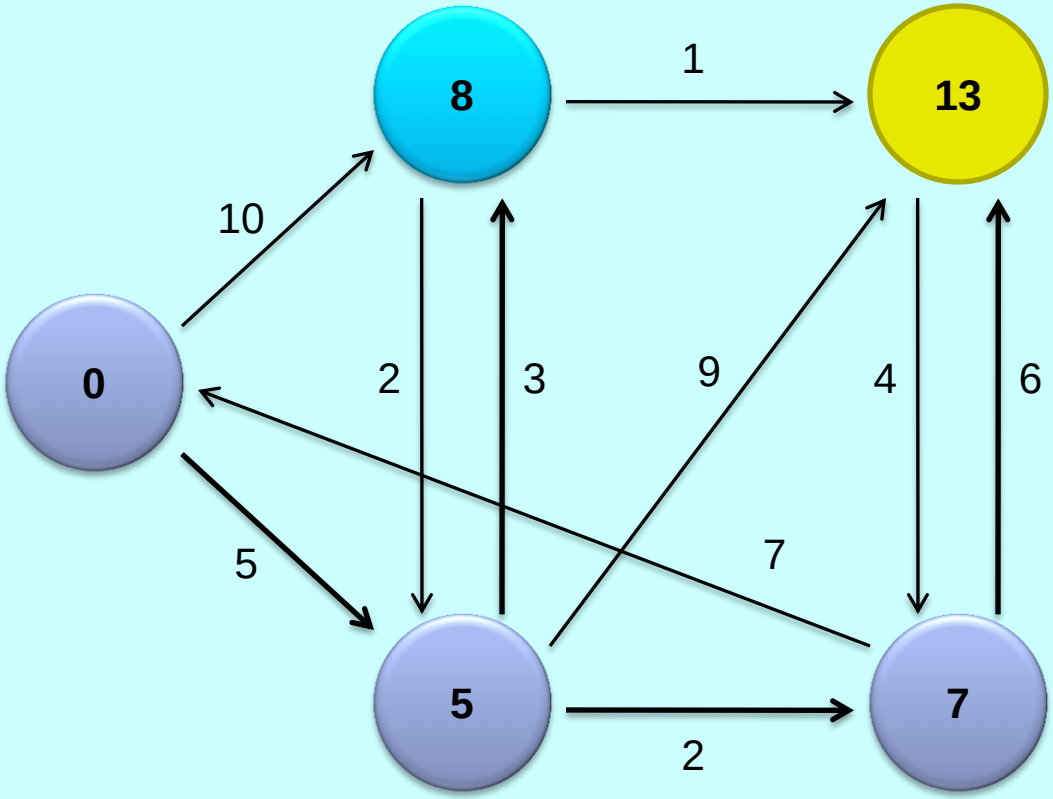
Example from CLR

Dijkstra's Algorithm Example



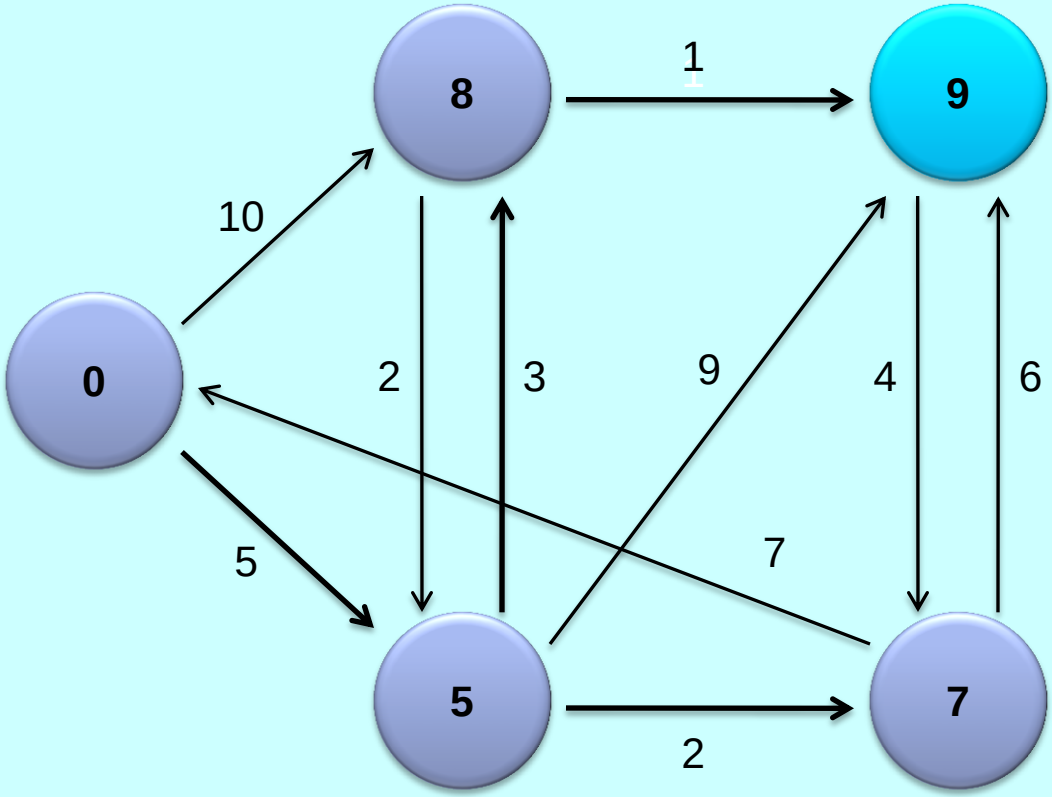
Example from CLR

Dijkstra's Algorithm Example



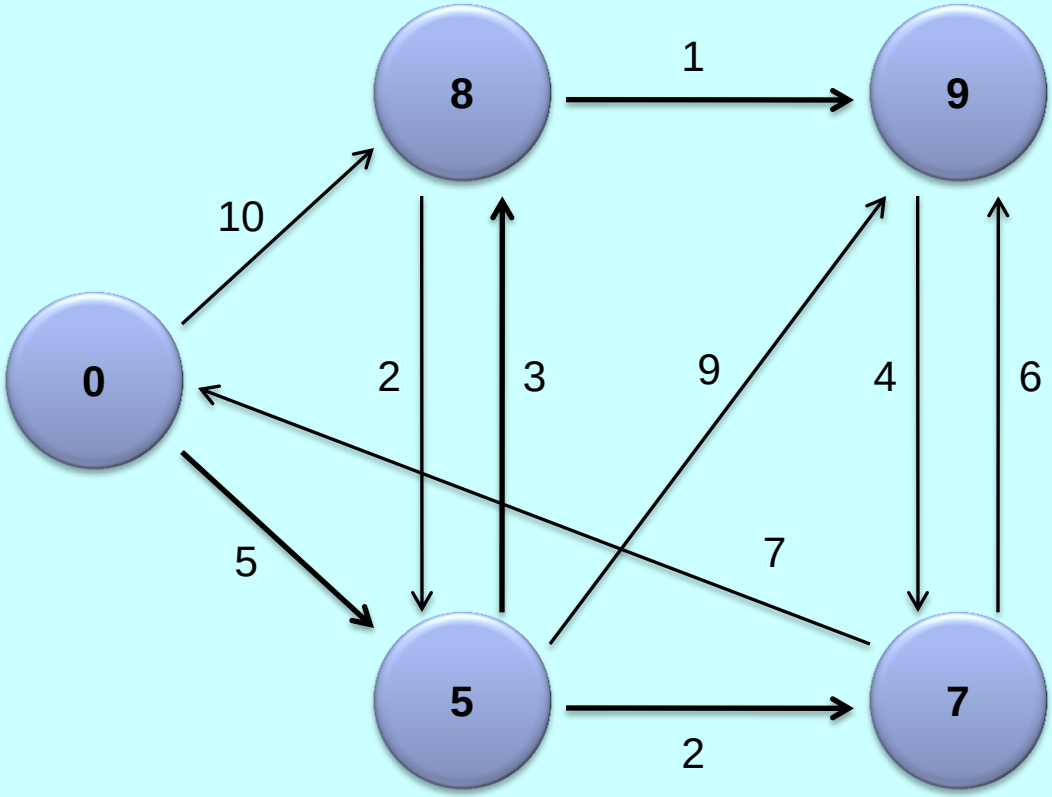
Example from CLR

Dijkstra's Algorithm Example



Example from CLR

Dijkstra's Algorithm Example



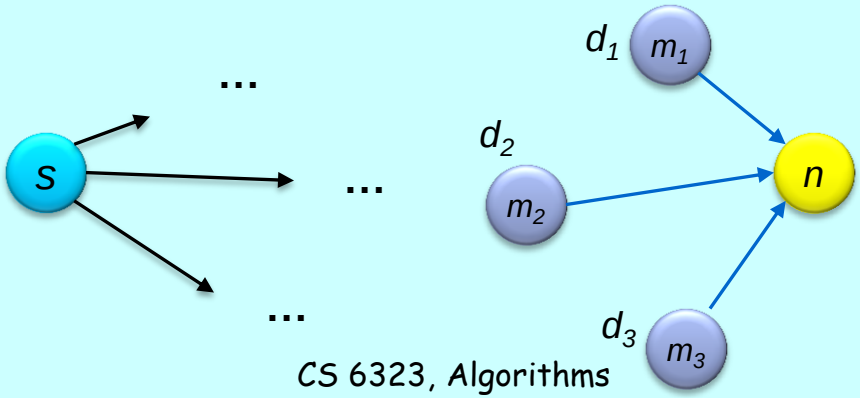
Example from CLR

Single Source Shortest Path

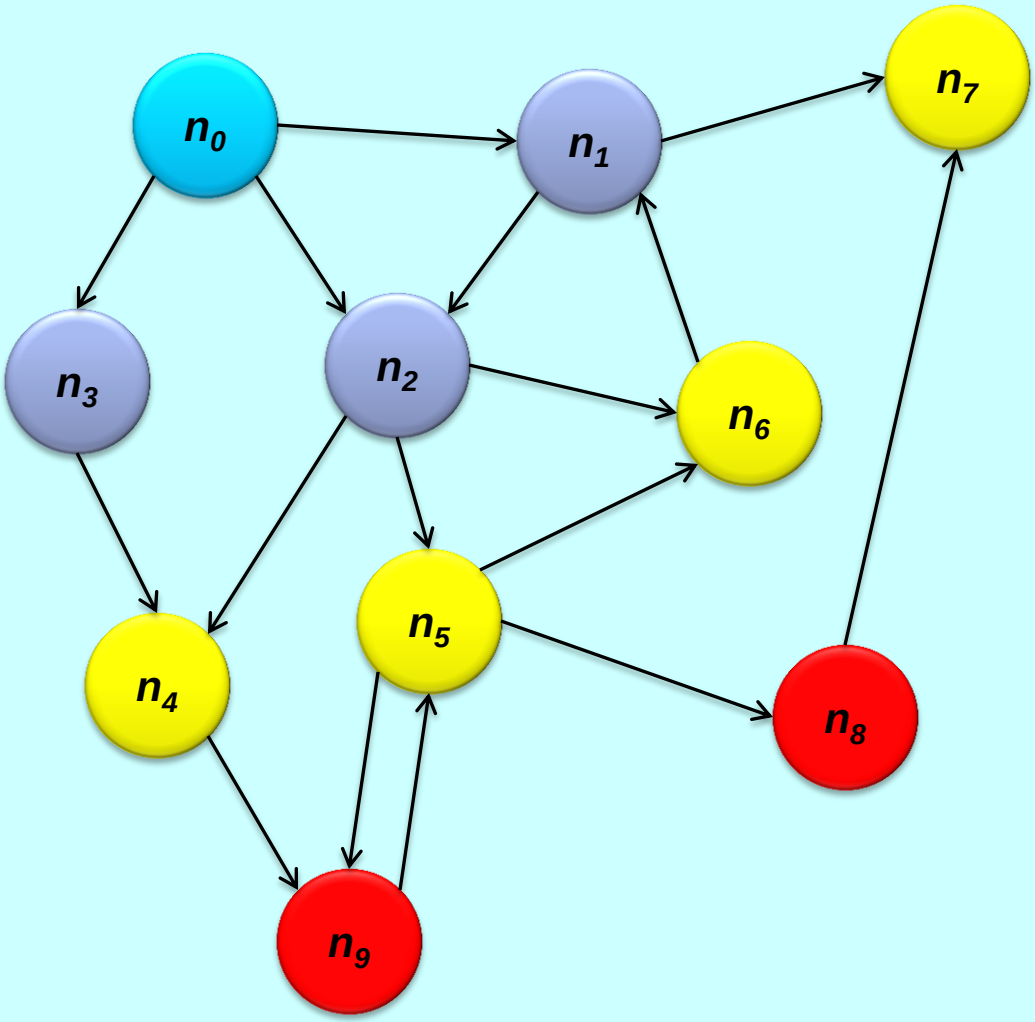
- Problem: find shortest path from a source node to one or more target nodes
  - Shortest might also mean lowest weight or cost
- Single processor machine: Dijkstra's Algorithm
- MapReduce: parallel Breadth-First Search (BFS)

Finding the Shortest Path

- Consider simple case of equal edge weights
- Solution to the problem can be defined inductively
- Here's the intuition:
  - Define: *b* is reachable from *a* if *b* is on adjacency list of *a*
  - DISTANCETO(*s*) = 0
  - For all nodes *p* reachable from *s*,
    DISTANCETO(*p*) = 1
  - For all nodes *n* reachable from some other set of nodes *M*,
    DISTANCETO(*n*) = 1 + min(DISTANCETO(*m*), $m \in M$)



Visualizing Parallel BFS



From Intuition to Algorithm

- **Data representation:**
  - Key: node *n*
  - Value: *d* (distance from start), adjacency list (list of nodes reachable from *n*)
  - Initialization: for all nodes except for start node, $d = \infty$
- **Mapper:**
  - $\forall m \in$ adjacency list: emit $(m, d + 1)$
- **Sort/Shuffle**
  - Groups distances by reachable nodes
- **Reducer:**
  - Selects minimum distance path for each reachable node
  - Additional bookkeeping needed to keep track of actual path

Multiple Iterations Needed

- Each MapReduce iteration advances the "known frontier" by one hop
  - Subsequent iterations include more and more reachable nodes as frontier expands
  - Multiple iterations are needed to explore entire graph
- Preserving graph structure:
  - Problem: Where did the adjacency list go?
  - Solution: mapper emits (*n*, adjacency list) as well

BFS Pseudo-Code

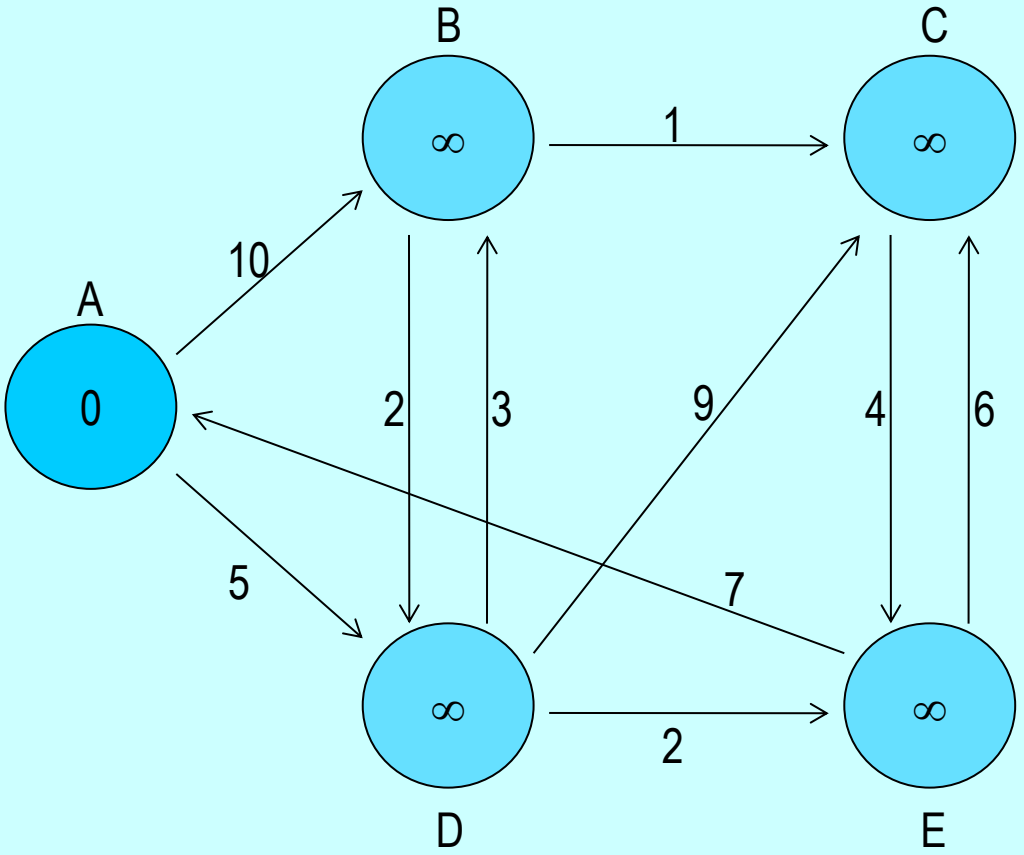
```
1: class Mapper
2:     method Map(nid n, node N)
3:         d ← N.Distance
4:         Emit(nid n, N)                          ▷ Pass along graph structure
5:         for all nodeid m ∈ N.AdjacencyList do
6:             Emit(nid m, d + 1)                  ▷ Emit distances to reachable nodes

1: class Reducer
2:     method Reduce(nid m, [d1, d2, ...])
3:         d_min ← ∞
4:         M ← ∅
5:         for all d ∈ counts [d1, d2, ...] do
6:             if IsNode(d) then
7:                 M ← d                           ▷ Recover graph structure
8:             else if d < d_min then              ▷ Look for shorter distance
9:                 d_min ← d
10:        M.Distance ← d_min                      ▷ Update shortest distance
11:        Emit(nid m, node M)
```

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: <node ID, <dist, adj list>>**

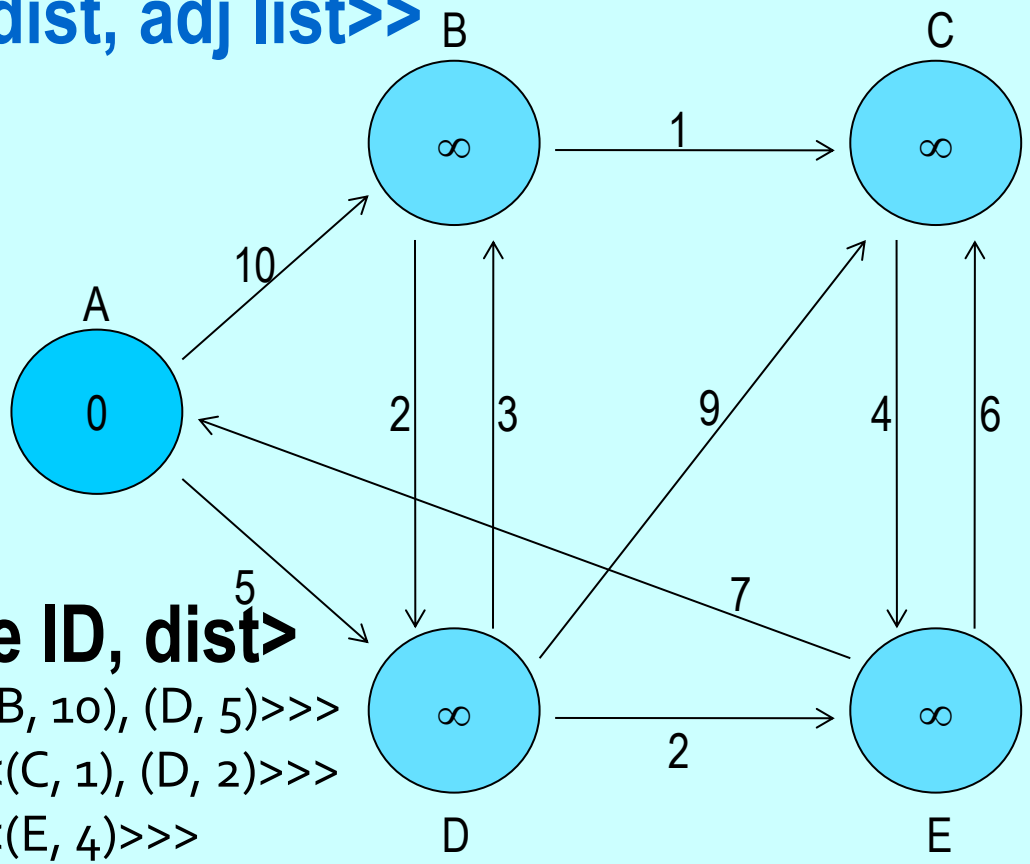
<A, <0, <(B, 10), (D, 5)>>>

<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: <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!!

Example: SSSP – Parallel BFS in MapReduce

- Reduce input: <node ID, dist>

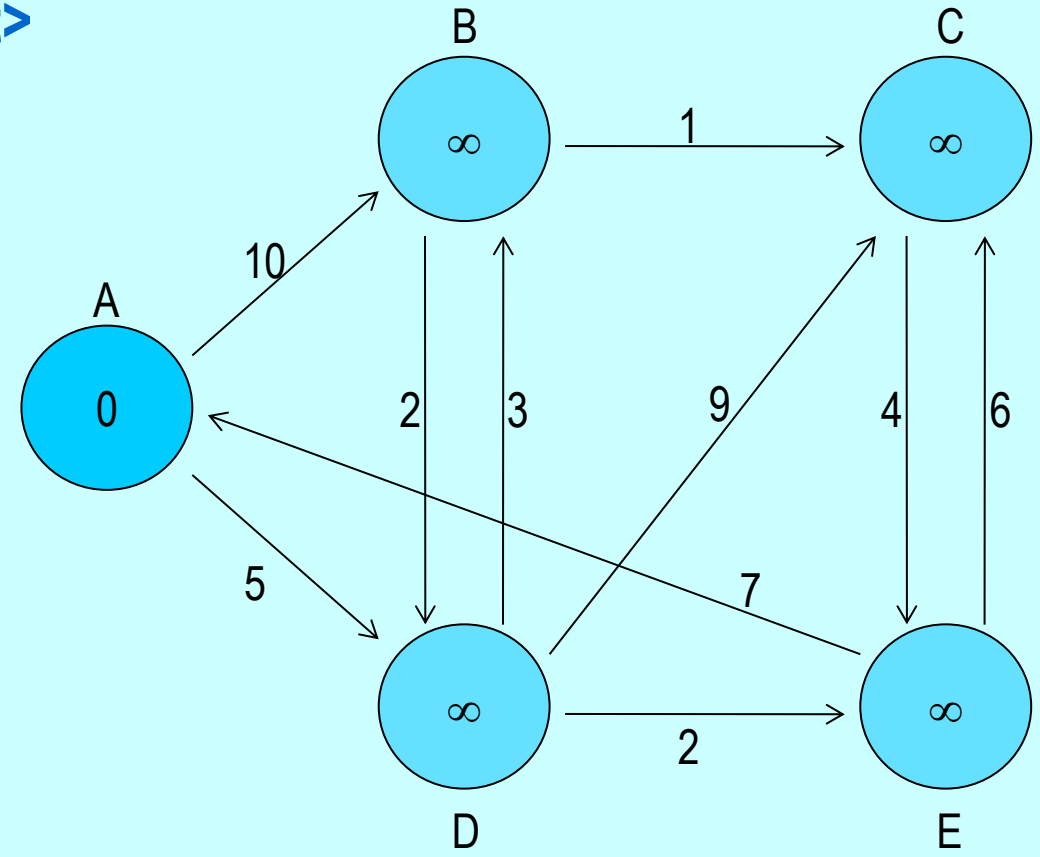
<A, <0, <(B, 10), (D, 5)>>>
<A, inf>

<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)>>>

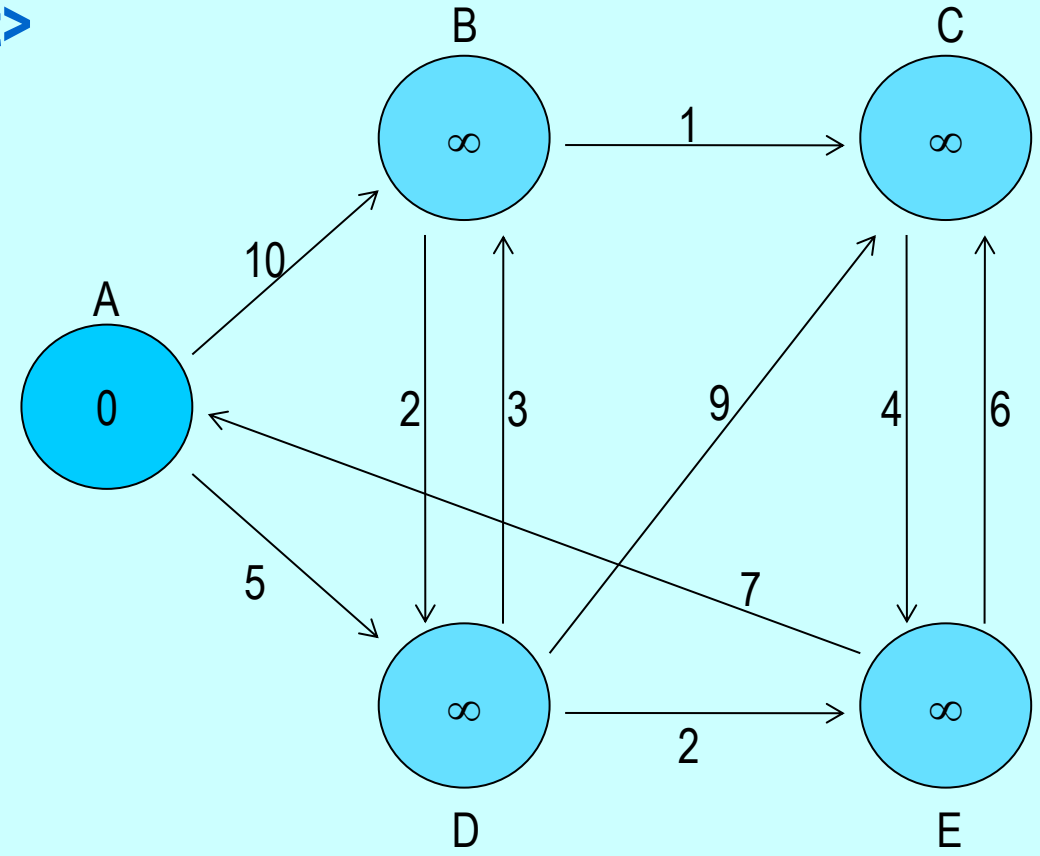
~~<C, inf> <C, inf> <C, inf>~~

<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: <node ID, <dist, adj list>>**
  = Map input for next iteration

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

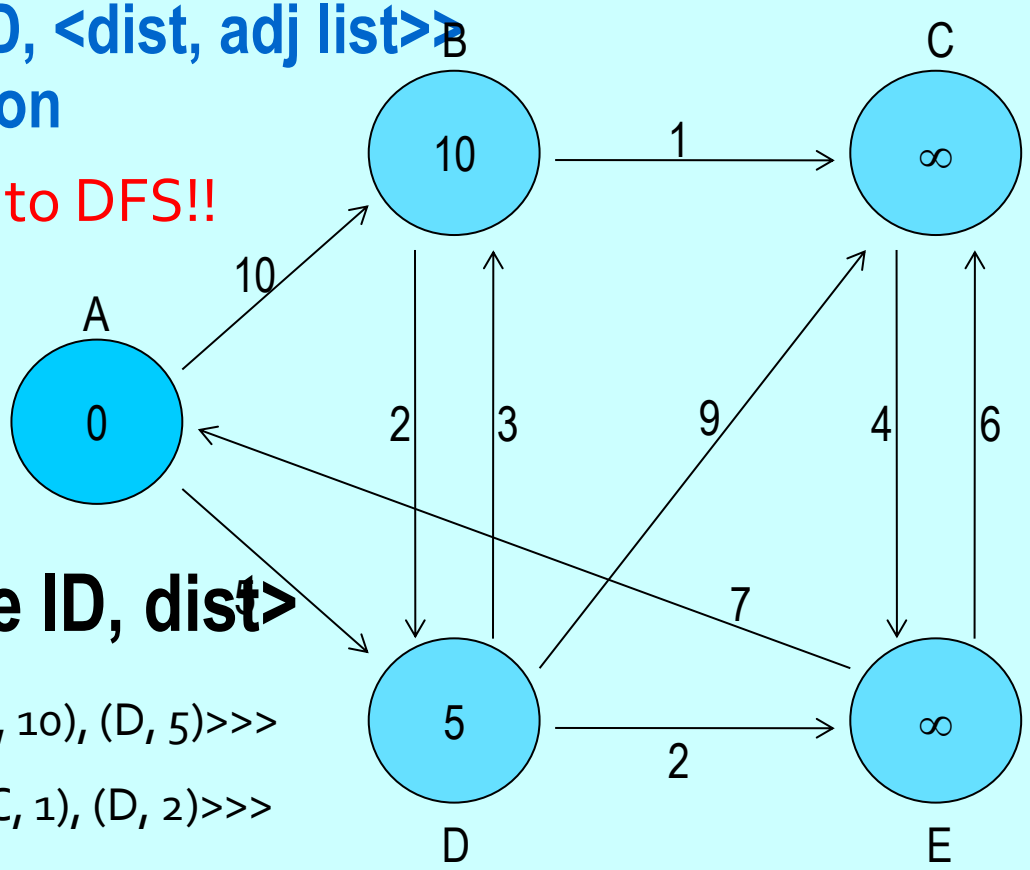
<B, <10, <(C, 1), (D, 2)>>>

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

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

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

Flushed to DFS!!



- **Map output: <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!!

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)>>>

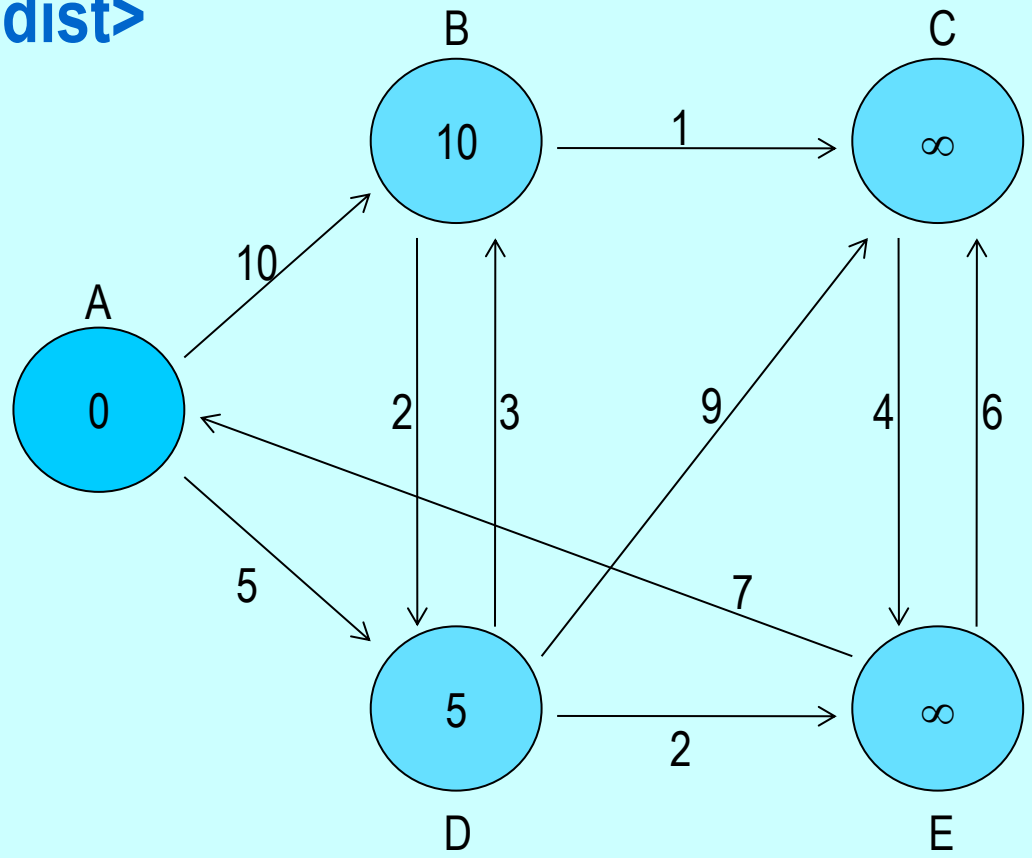
<C, 11> <C, 14> <C, inf>

<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)>>>

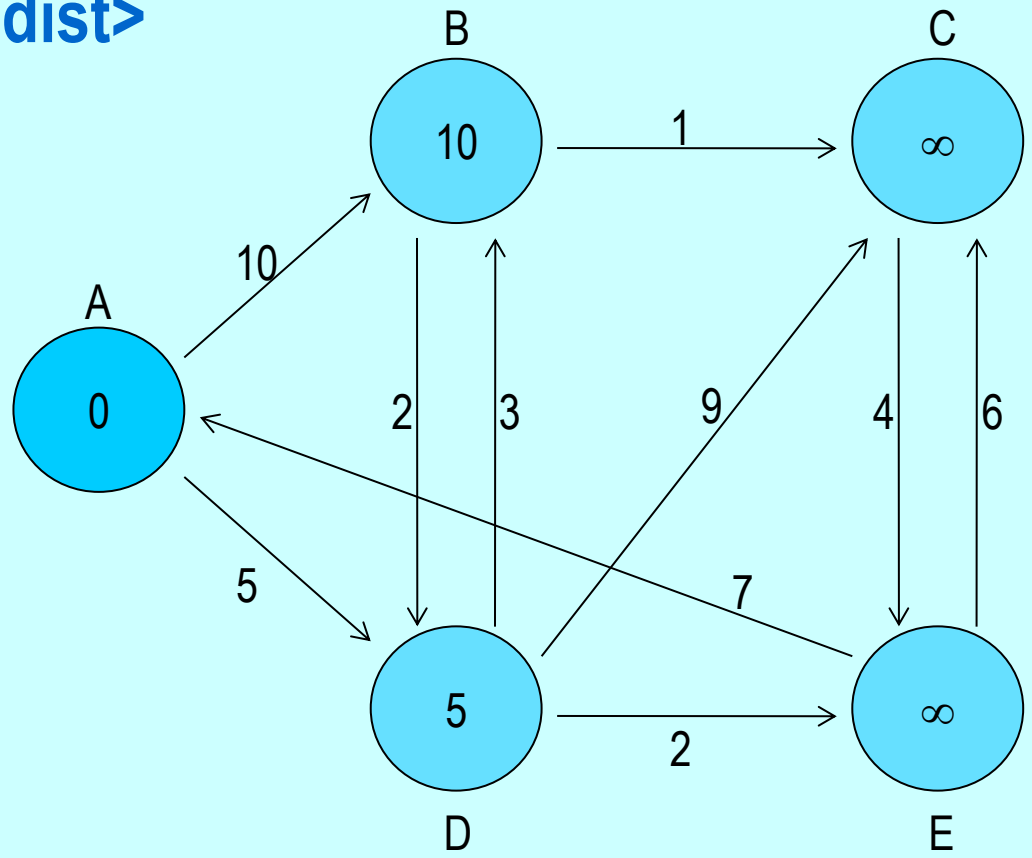
<C, 11> ~~<C, 14> <C, inf>~~

<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: <node ID, <dist, adj list>>**
  = Map input for next iteration

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

<B, <8, <(C, 1), (D, 2)>>>

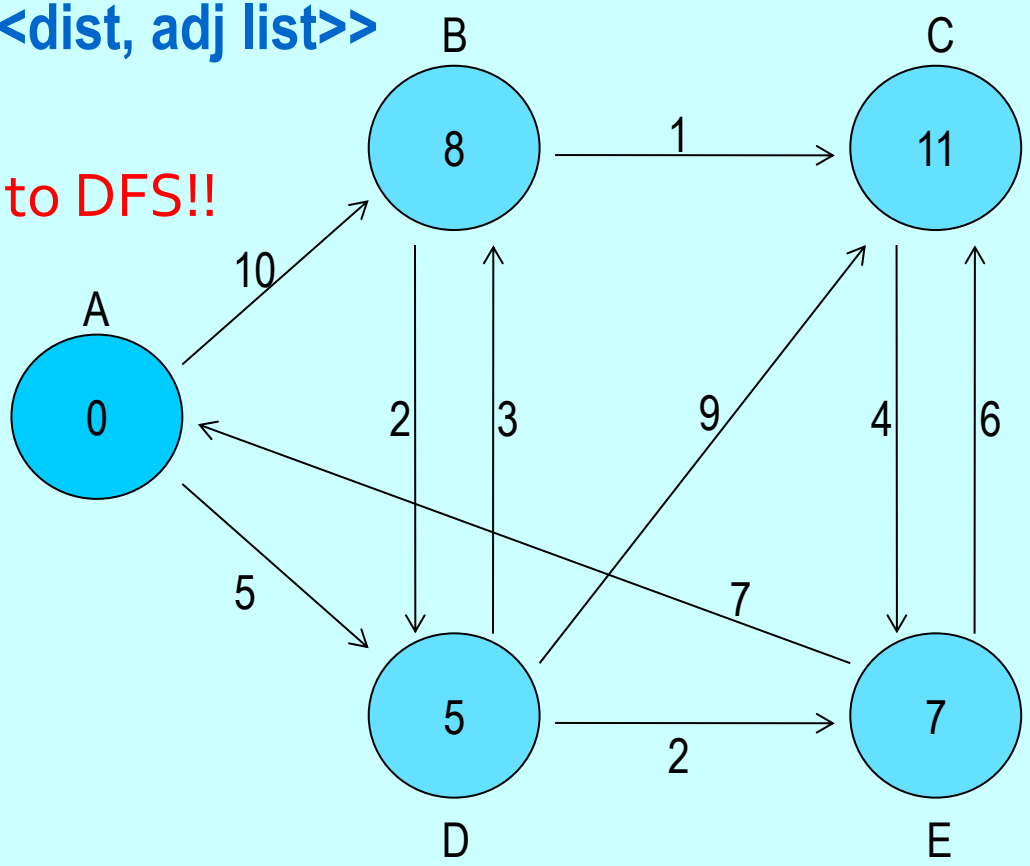
<C, <11, <(E, 4)>>>

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

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

Flushed to DFS!!

… the rest omitted …



Stopping Criterion

- How many iterations are needed in parallel BFS (equal edge weight case)?
- Convince yourself: when a node is first "discovered", we've found the shortest path
- Now answer the question...
  - Six degrees of separation?
- Practicalities of implementation in MapReduce

Comparison to Dijkstra

- Dijkstra's algorithm is more efficient
  - At any step it only pursues edges from the minimum-cost path inside the frontier
- MapReduce explores all paths in parallel
  - Lots of "waste"
  - Useful work is only done at the "frontier"
- Why can't we do better using MapReduce?

Weighted Edges

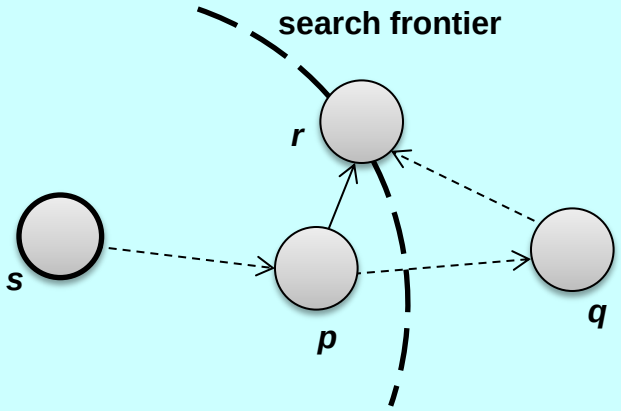
- Now add positive weights to the edges
  - Why can't edge weights be negative?
- Simple change: adjacency list now includes a weight *w* for each edge
  - In mapper, emit (m, $d + w_p$) instead of (m, $d + 1$) for each node m

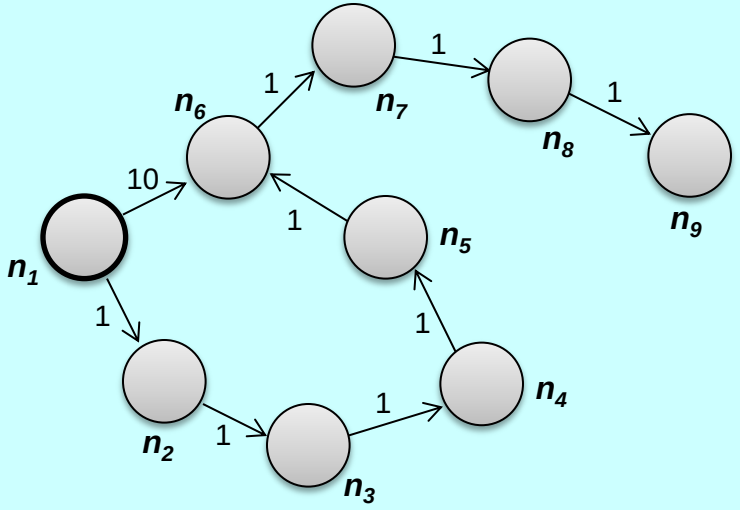
Stopping Criterion

- How many iterations are needed in parallel BFS (positive edge weight case)
  - Graph diameter D
- Convince yourself: when a node is first "discovered", we've found the shortest path

Not true!

Additional Complexities





Stopping Criterion

- How many iterations are needed in parallel BFS (positive edge weight case)?
- Practicalities of implementation in MapReduce

Graphs and MapReduce

- Graph algorithms typically involve:
  - Performing computations at each node: based on node features, edge features, and local link structure
  - Propagating computations: “traversing” the graph
- Generic recipe:
  - Represent graphs as adjacency lists
  - Perform local computations 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”
  - Don’t forget to pass the graph structure between iterations

http://famousphil.com/blog/2011/06/a-hadoop-mapreduce-solution-to-dijkstra%E2%80%99s-algorithm/

```
public class Dijkstra extends Configured implements Tool {
  public static String OUT = "outfile";
  public static String IN = "inputlarger";
  public static class TheMapper extends Mapper<LongWritable, Text, LongWritable, Text> {
      public void map(LongWritable key, Text value, Context context) throws IOException, InterruptedExceptio
Text word = new Text();
        String line = value.toString();//looks like 1 0 2:3:
        String[] sp = line.split(" ");//splits on space
        int distanceadd = Integer.parseInt(sp[1]) + 1;
        String[] PointsTo = sp[2].split(":");
        for(int i=0; i<PointsTo.length; i++){
          word.set("VALUE "+distanceadd);//tells me to look at distance value
          context.write(new LongWritable(Integer.parseInt(PointsTo[i])), word);
          word.clear(); }
        //pass in current node's distance (if it is the lowest distance)
        word.set("VALUE "+sp[1]);
        context.write( new LongWritable( Integer.parseInt( sp[0] ) ), word );
        word.clear();
        word.set("NODES "+sp[2]);//tells me to append on the final tally
        context.write( new LongWritable( Integer.parseInt( sp[0] ) ), word );
        word.clear();
```

```
public static class TheReducer extends Reducer<LongWritable, Text, LongWritable, Text> {
    public void reduce(LongWritable key, Iterable<Text> values, Context context) throws
IOException, InterruptedException {
String nodes = "UNMODED";
        Text word = new Text();
        int lowest = 10009;//start at infinity

        for (Text val : values) {//looks like NODES/VALUES 1 0 2:3:, we need to use the first
as a key
            String[] sp = val.toString().split(" ");//splits on space
            //look at first value
            if(sp[0].equalsIgnoreCase("NODES")){
                nodes = null;
                nodes = sp[1];
            }else if(sp[0].equalsIgnoreCase("VALUE")){
                int distance = Integer.parseInt(sp[1]);
                lowest = Math.min(distance, lowest);
            }
        }
        word.set(lowest+" "+nodes);
        context.write(key, word);
        word.clear();
```

```
public int run(String[] args) throws Exception {
    //http://code.google.com/p/joycrawler/source/browse/NetflixChallenge/src/org/niubility/learning/knn/KNNDriver.java?r=2
    getConf().set("mapred.textoutputformat.separator", " ");//make the key -> value space separated (for iterations)
....
    while(isdone == false){
        Job job = new Job(getConf());
        job.setJarByClass(Dijkstra.class);
        job.setJobName("Dijkstra");
        job.setOutputKeyClass(LongWritable.class);
        job.setOutputValueClass(Text.class);
        job.setMapperClass(TheMapper.class);
        job.setReducerClass(TheReducer.class);
        job.setInputFormatClass(TextInputFormat.class);
        job.setOutputFormatClass(TextOutputFormat.class);

        FileInputFormat.addInputPath(job, new Path(infile));
        FileOutputFormat.setOutputPath(job, new Path(outputfile));
        success = job.waitForCompletion(true);
        //remove the input file
        //http://eclipse.sys-con.com/node/1287801/mobile
        if(infile != IN){
            String indir = infile.replace("part-r-00000", "");
            Path ddir = new Path(indir);
            FileSystem dfs = FileSystem.get(getConf());
            dfs.delete(ddir, true);
```