Parallel Algorithms Single-Source Shortest Path

Computação de Alto Desempenho 2024/25

Hervé Paulino

Slides adapted from Ananth Grama, Anshul Gupta, George Karypis, and Vipin Kumar, "Introduction to Parallel Computing", Addison Wesley, 2003

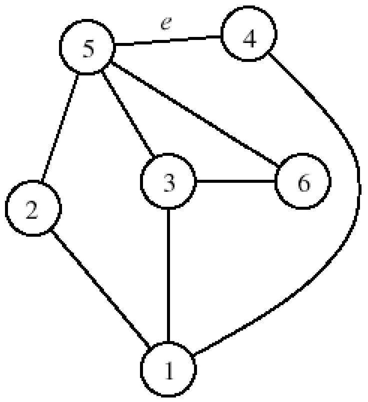
Bibliography

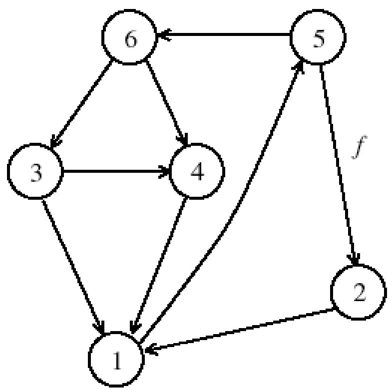
- Chapter 10 (10.1 to 10.3 and 10.7.2) of Introduction to Parallel Computing (2nd Edition) 2nd Edition, Ananth Grama, George Karypis, Vipin Kumar and Anshul Gupta. Pearson, 2003

Graph Algorithms

- Graph theory important in computer science
- Many complex problems are graph problems

- G = (V, E)
  - V finite set of points called vertices
  - E finite set of edges
  - $e \in E$ is an pair (u,v), where $u,v \in V$
  - Unordered and ordered graphs



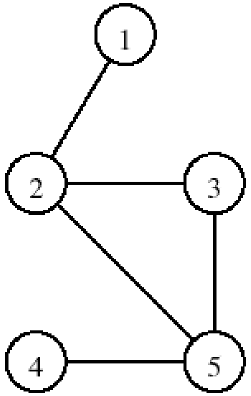


Graph Terminology

- Vertex adjacency if (u,v) is an edge
- Path from u to v if there is an edge sequence starting at u and ending at v
- If there exists a path, v is reachable from u
- A graph is connected if all pairs of vertices are connected by a path
- A weighted graph associates weights with each edge
- Adjacency matrix is an n x n array A such that
  - $A_{i,j} = 1$ if $(v_i,v_j) \in E$; 0 otherwise
  - Can be modified for weighted graphs (∞ is no edge)
  - Can be represented as adjacency lists

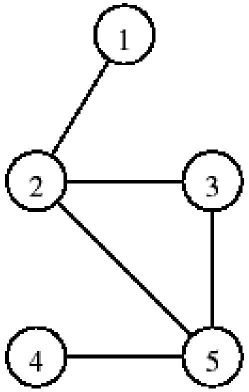
Graph Representations

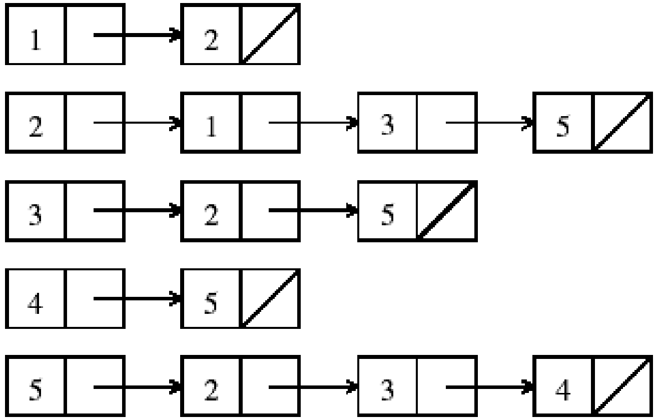
- Adjacency matrix



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

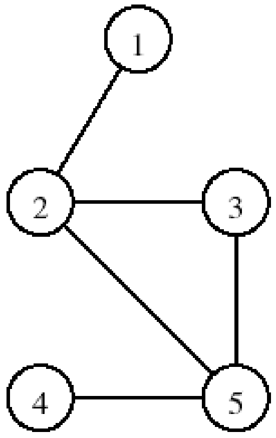
- Adjacency list





Graph Representations
Compressed Sparse Row (CSR) format

- Represents a matrix with 3 one-dimensional arrays
  - Data: non zero entries of the matrix
  - Columns: the column of each value
  - Rows: the index where each row starts (may contain final element with the number of non-zero elements)
- Used in many GPU Graph processing frameworks
  - Limitations:
    - expensive to access the neighbors of a node
    - Hard to add and delete nodes
  - Some frameworks use more complex methods



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

Data	1	1	1	1	1	1	1	1	1	1
columns	1	0	2	4	1	4	4	1	2	3
rows	0	1	4	6	7	10				

Graph Partitioning: Challenges

- Localizing portions of the computation
  - How to partition the workload so that nearby nodes in the graph are mapped to the same processor?
  - How to partition the workload so that edges that represent significant communication are co-located on the same processor?
- Balancing the load
  - How to give each processor a comparable amount of work?
  - How much knowledge of the graph do we need to do this since complete traversal may not be realistic?
- All of these issues must be addressed by a graph partitioning algorithm that maps individual subgraphs to individual processors.

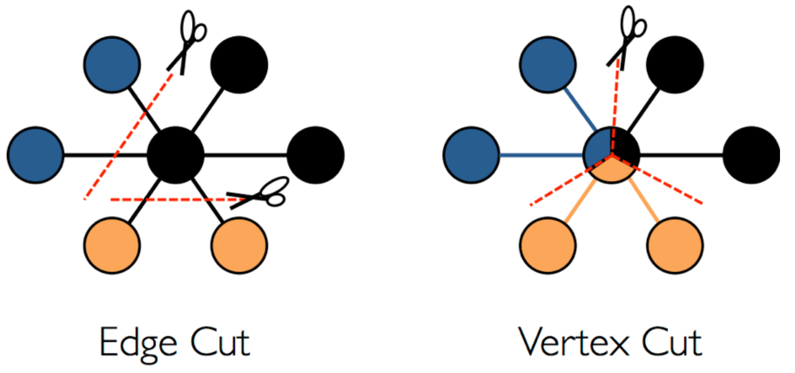


Image taken from https://spark.apache.org/docs/latest/graphx-programming-guide.html#pregel-api

Graph Processing Models

- Vertex centric
  - The kernel is applied over a set of vertexes
- Edge centric
  - The kernel is applied over a set of edges
- There is no clean better solution, it depends on the kernel to apply
  - Frameworks tend to favor vertex-centric models

Big Data Graph Processing Models

- Pregel
  - Developed by Google
  - Based on the BSP model
  - Vertex centric
  - A graph is divided into partitions: set of vertices plus outgoing edges
  - An hash function on the vertexes id defines to which partition is it assigned
- GraphX
  - Uses the Pregel API
  - Vertex and Edge RDDs
  - Optimized representation
    - Split along vertexes

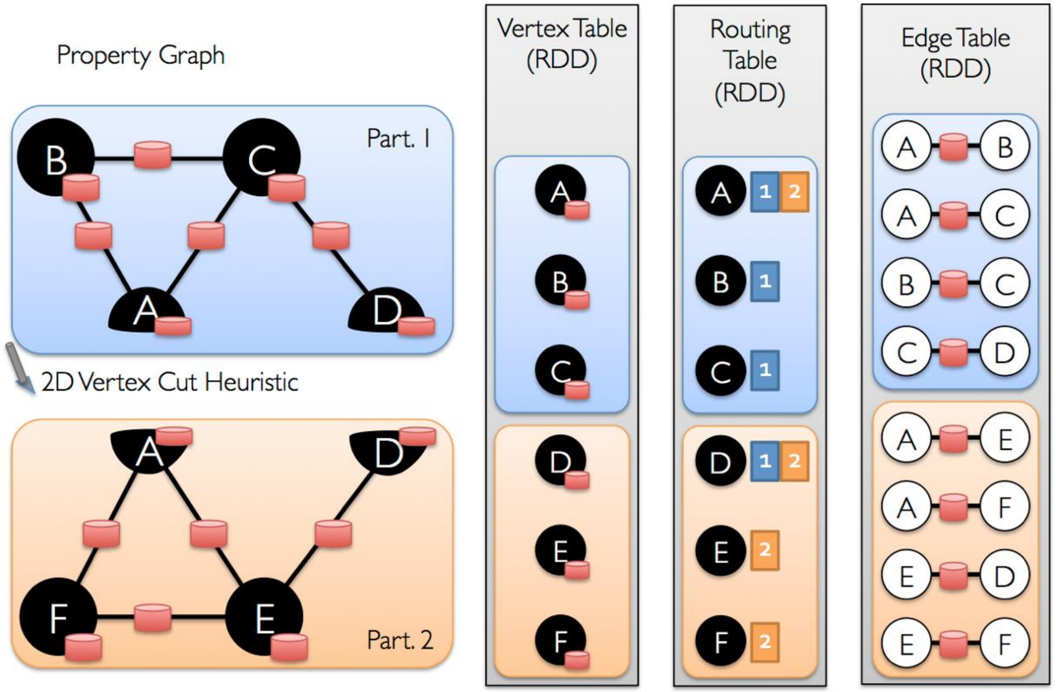


Image taken from https://spark.apache.org/docs/latest/graphx-programming-guide.html#pregel-api

Widely Used Graph Algorithms

- Minimum Spanning Tree
  - A spanning tree of an undirected graph G is a subgraph of G that is a tree containing all the vertices of G
  - The minimum spanning tree (MST) for a weighted undirected graph is a spanning tree with minimum weight
- Single Source Shortest Path (SSSP)
  - Shortest paths from a node to all the others
- All Sources Shortest Path
  - Shortest paths between all nodes
- Page Rank
  - Node ranking base on their in-degree and the rank of the nodes that point to it
- Maximal Independent Set
  - A set of vertices $I \subset V$ is called independent if no pair of vertices in I is connected via an edge in G.

Single-Source Shortest Path

- Find shortest path from a vertex v to all other vertices
- The shortest path in a weighted graph is the edge with the minimum weight
- Weights may represent time, cost, loss, or any other quantity that accumulates additively along a path

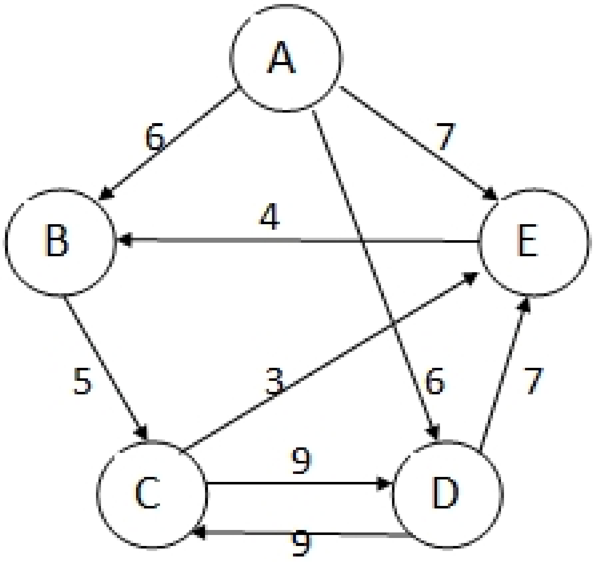
- Dijkstra’s algorithm finds shortest paths from vertex s
  - Incrementally finds shortest paths in greedy manner
  - Keep track of minimum cost to reach a vertex from s
  - O(n2)

Dijkstra's Single-Source Shortest Path

```
DIJKSTRA SINGLE SOURCE SP(V, E,w, s) {
    VT := {s};
    for all v ∈ (V - VT )
       if (s, v) exists set l[v] := w(s, v);
       else set l[v] :=∞;
    while VT ≠ V
       find a vertex u such that l[u] := min{l[v] | v ∈ (V - VT )};
       VT := VT ∪ {u};
       for all v ∈ (V - VT )
           l[v] := min{l[v], l[u] + w(u, v)};
}
```

SSSP for Vertex A

Graph

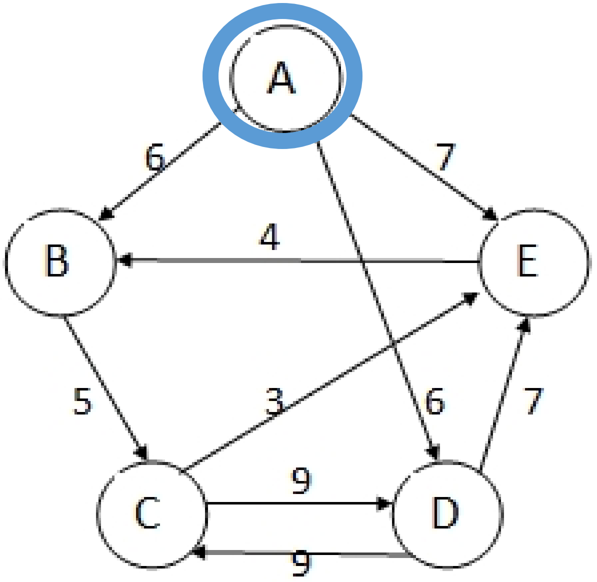


	A 1	B 2	C 3	D 4	E 5
L					

	A 1	B 2	C 3	D 4	E 5
A 1	0	6	∞	6	7
B 2	∞	0	5	∞	∞
C 3	∞	∞	0	9	3
D 4	∞	∞	9	0	7
E 5	∞	4	∞	∞	0

SSSP for Vertex A

Graph

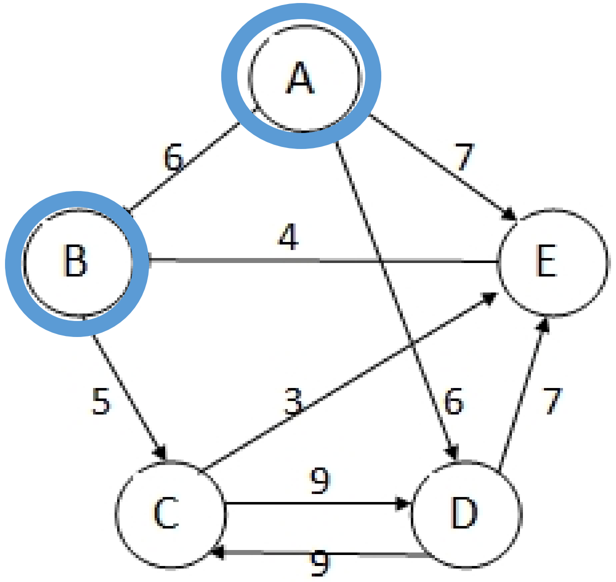


	A 1	B 2	C 3	D 4	E 5
L	0	6	∞	6	7

	A 1	B 2	C 3	D 4	E 5
A 1	0	6	∞	6	7
B 2	∞	0	5	∞	∞
C 3	∞	∞	0	9	3
D 4	∞	∞	9	0	7
E 5	∞	4	∞	∞	0

SSSP for Vertex A

Graph

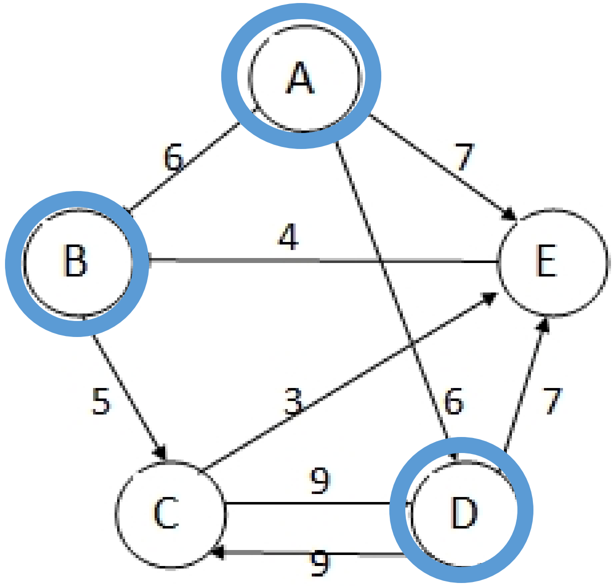


	A 1	B 2	C 3	D 4	E 5
L	0	6	11	6	7

	A 1	B 2	C 3	D 4	E 5
A 1	0	6	∞	6	7
B 2	∞	0	5	∞	∞
C 3	∞	∞	0	9	3
D 4	∞	∞	9	0	7
E 5	∞	4	∞	∞	0

SSSP for Vertex A

Graph

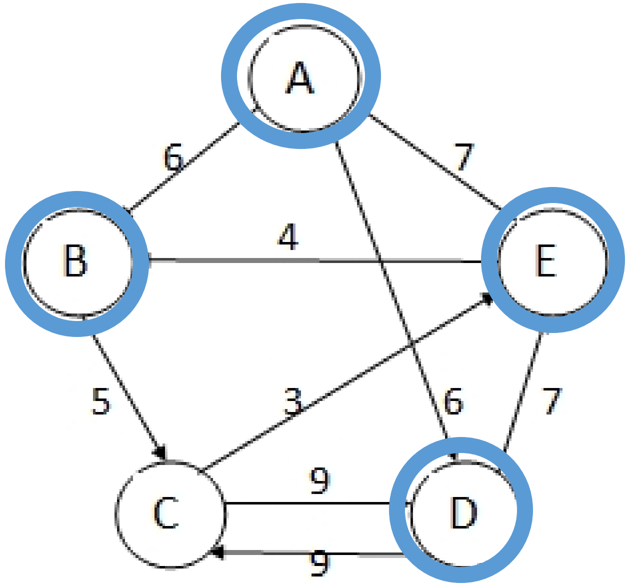


	A 1	B 2	C 3	D 4	E 5
L	0	6	11	6	7

	A 1	B 2	C 3	D 4	E 5
A 1	0	6	∞	6	7
B 2	∞	0	5	∞	∞
C 3	∞	∞	0	9	3
D 4	∞	∞	9	0	7
E 5	∞	4	∞	∞	0

SSSP for Vertex A

Graph

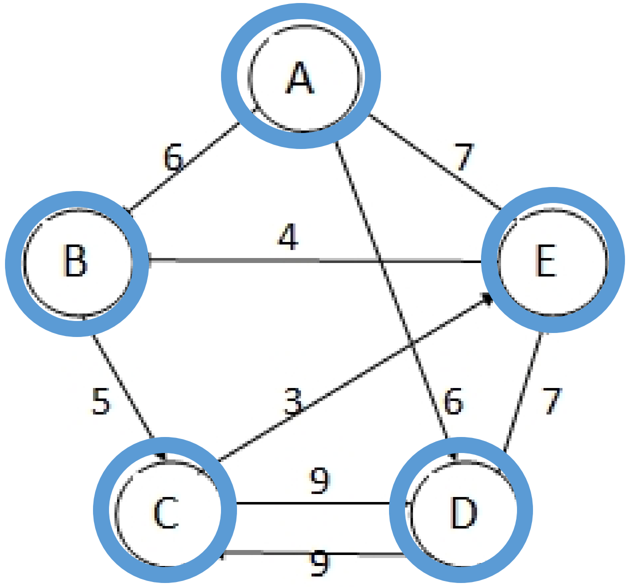


	A 1	B 2	C 3	D 4	E 5
L	0	6	11	6	7

	A 1	B 2	C 3	D 4	E 5
A 1	0	6	∞	6	7
B 2	∞	0	5	∞	∞
C 3	∞	∞	0	9	3
D 4	∞	∞	9	0	7
E 5	∞	4	∞	∞	0

SSSP for Vertex A

Graph



	A 1	B 2	C 3	D 4	E 5
L	0	6	11	6	7

	A 1	B 2	C 3	D 4	E 5
A 1	0	6	∞	6	7
B 2	∞	0	5	∞	∞
C 3	∞	∞	0	9	3
D 4	∞	∞	9	0	7
E 5	∞	4	∞	∞	0

Parallel Formulation of Dijkstra's Algorithm

- Difficult to perform different iterations of the while loop in parallel because l[v] may change each time
- Classic parallel algorithm paralyzes each iteration:
  - Partition vertices into p subsets Vi, i=0,…,p-1
  - Each process Pi computes li[u]=min{li[v] | $v \in (V-VT) \cap Vi$}
  - Global minimum is obtained using all-to-one reduction
  - New vertex is added to VT and broadcast to all processes
  - New values of l[v] are computed for local vertex
- Is this algorithm implementable in GPUs or Spark?

No. Because it is not possible to implement grid-wide **all-one-reduction** reduction, nor **broadcasts.** **Adaptations must be made**

Single-Source Shortest Path

- Dijkstra's algorithm, modified to handle sparse graphs is called Johnson's algorithm.
- The modification accounts for the fact that the minimization step in Dijkstra's algorithm needs to be performed only for those nodes adjacent to the previously selected nodes (the frontier).
- Johnson's algorithm uses a priority queue Q to store the value l[v] for each vertex v ∈ (V – VT).
  - Smaller values (shortest paths) have higher priority

Single-Source Shortest Path: Parallel Johnson's Algorithm

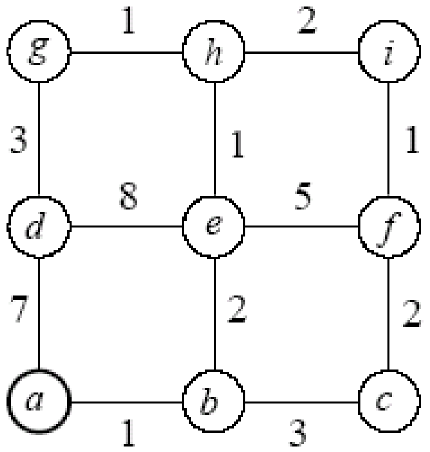
- Maintaining strict order of Johnson's algorithm generally leads to a very restrictive class of parallel algorithms.

- We need to allow exploration of multiple nodes concurrently. This is done by simultaneously extracting p nodes from the priority queue, updating the neighbors' cost, and augmenting the shortest path.

- If an error is made, it can be discovered (as a shorter path) and the node can be reinserted with this shorter path.

Single-Source Shortest Paths: Parallel Johnson's Algorithm

- An example of the modified Johnson's algorithm for processing unsafe vertices concurrently.



Shortest path from vertex a

Priority Queue

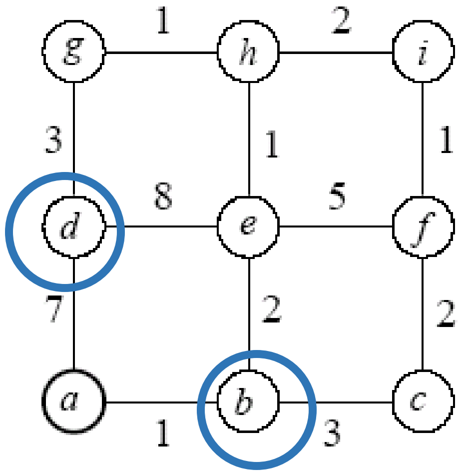
(1) *b:1, d:7, c:inf, e:inf, f:inf, g:inf, h:inf, i:inf*

Array l[]

a	*b*	*c*	*d*	*e*	*f*	*g*	*h*	*i*
0	1	∞	7	∞	∞	∞	∞	∞

Single-Source Shortest Paths: Parallel Johnson's Algorithm

- An example of the modified Johnson's algorithm for processing unsafe vertices concurrently.



Shortest path from vertex a

Priority Queue

(1) *b:1, d:7, c:inf, e:inf, f:inf, g:inf, h:inf, i:inf*

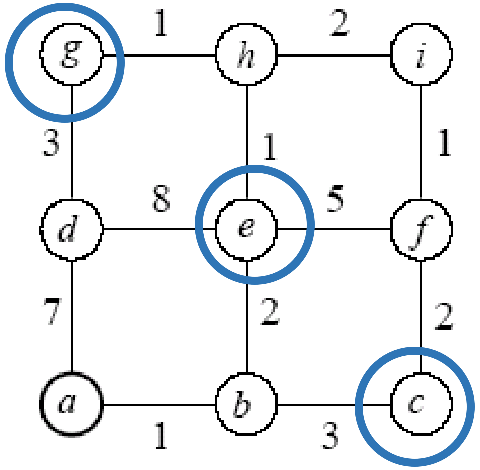
(2) *e:3, c:4, g:10, f:inf, h:inf, i:inf*

Array l[]

	a	*b*	*c*	*d*	*e*	*f*	*g*	*h*	*i*
(1)	0	1	∞	7	∞	∞	∞	∞	∞
(2)	0	1	4	7	3	∞	10	∞	∞

Single-Source Shortest Paths: Parallel Johnson's Algorithm

- An example of the modified Johnson's algorithm for processing unsafe vertices concurrently.



Shortest path from vertex a

Priority Queue

(1) *b:1, d:7, c:inf, e:inf, f:inf, g:inf, h:inf, i:inf*

(2) $\boxed{e{:}3,\ c{:}4,\ g{:}10}$ *f:inf, h:inf, i:inf*

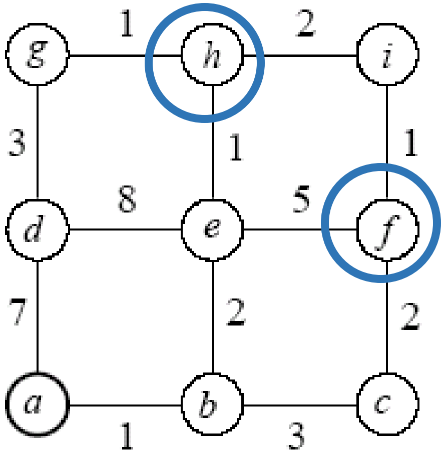
(3) *h:4, f:6, i:inf*

Array l[]

	a	*b*	*c*	*d*	*e*	*f*	*g*	*h*	*i*
(1)	0	1	∞	7	∞	∞	∞	∞	∞
(2)	0	1	4	7	3	∞	10	∞	∞
(3)	0	1	4	7	3	$\boxed{6}$	10	$\boxed{4}$	∞

Single-Source Shortest Paths: Parallel Johnson's Algorithm

- An example of the modified Johnson's algorithm for processing unsafe vertices concurrently.



	Priority Queue	a	b	c	d	e	f	g	h	i
(1)	*b:1, d:7, c:inf, e:inf, f:inf, g:inf, h:inf, i:inf*	0	1	∞	7	∞	∞	∞	∞	∞
(2)	*e:3, c:4, g:10, f:inf, h:inf, i:inf*	0	1	4	7	3	∞	10	∞	∞
(3)	*h:4, f:6, i:inf*	0	1	4	7	3	6	10	4	∞
(4)	*g:5, i:6*	0	1	4	7	3	6	5	4	6

Array l[]

Shortest path from vertex a

Single-Source Shortest Paths: Parallel Johnson's Algorithm

- Even if we can extract and process multiple nodes from the queue, the queue itself is a major bottleneck.
- For this reason, we use multiple queues, one for each processor. Each processor builds its priority queue only using its own vertices.
- When process Pi extracts the vertex $u \in Vi$, it sends a message to processes that store vertices adjacent to u.
- Process Pj, upon receiving this message, sets the value of l[v] stored in its priority queue to min{l[v],l[u] + w(u,v)}.

Once again, in GPUs and Spark this communication requires communication between workers

Single-Source Shortest Paths: Parallel Johnson's Algorithm

- If a shorter path has been discovered to node v, it is reinserted back into the local priority queue.
- The algorithm terminates only when all the queues become empty.
- A number of node partitioning schemes can be used to exploit graph structure for performance.