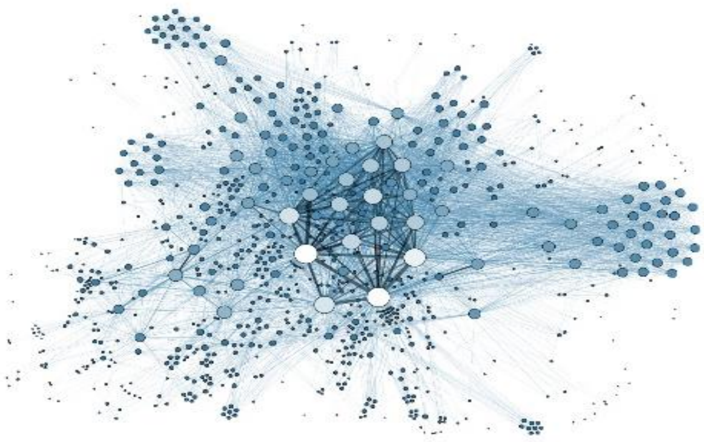
8th Semester Final Year Project Presentation 2021



Source: Google Images

Custom Accelerator For *Graph Analytics*

GROUP 26

Avani S (1PE17EC027)
Annette Antony (1PE17EC018)
Bhimala Subbarayudu (1PE17EC031)
Jeevan R (1PE17EC055)

Under the guidance of
Dr. Madhura Purnaprajna

INTRODUCTION

Problem Statement:

To design a custom FPGA accelerator for **four** different graph algorithms, namely,

- Single Source Shortest Path
- PageRank
- Breadth-First-Search
- Depth-First-Search.

- Since we can abstract most of the real-world data as graphs, the study of such graphs provides answers to many arrangement, networking, matching and operational problems.

- With the ever-increasing sizes of the graph datasets, rises the need of specialized hardware to compute them efficiently.

- FPGAs could be explored as an efficient solution to provide a specialized hardware for graph processing.

Source: Wikipedia

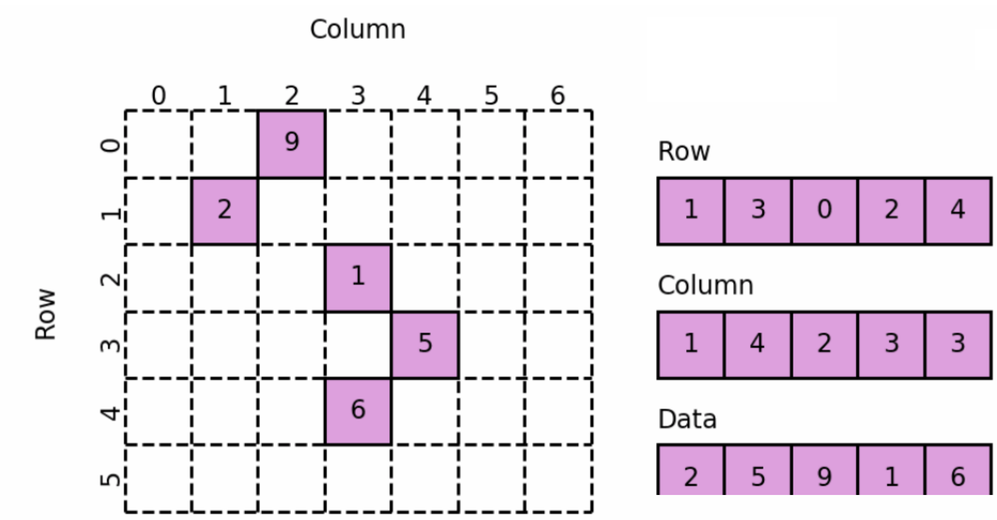
INTRODUCTION

Solution:

- Implement the respective graph algorithms in Vivado HLS.
- Synthesize to generate RTL files.
- Analyse the results by examining the latency, II, throughput, and resource utilization.
- Optimise the algorithms with the relevant directives available in the Vivado HLS tool
- Implement Graph Partitioning
- Verify the results using C/RTL Cosim and export the IP.
- Execute the graph on GAP Benchmark
- Compare the performance of the kernel.

SPARSE MATRIX REPRESENTATIONS

COO Format

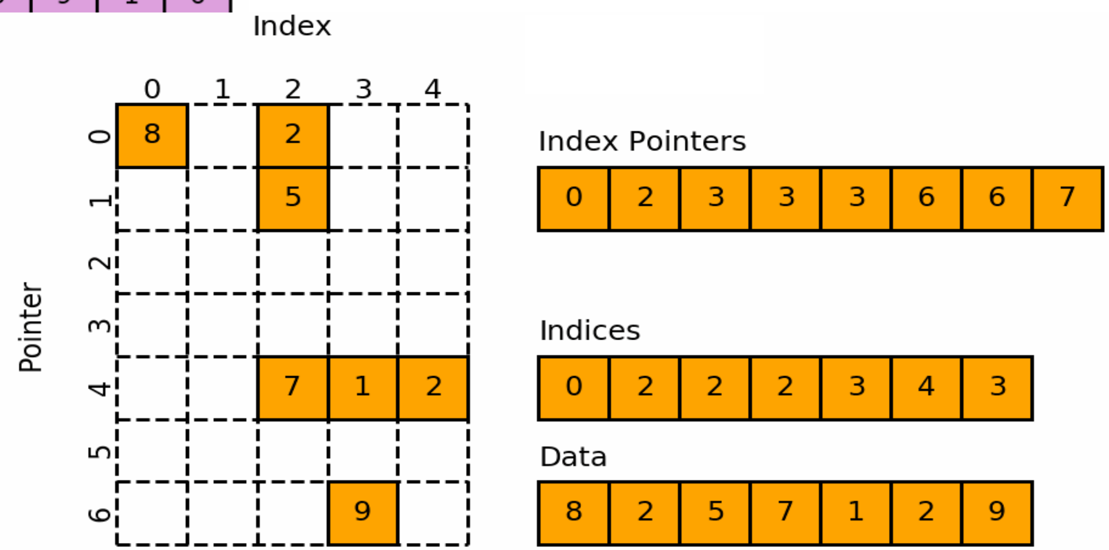


Source: Google Images

- Real world graph data sets have 96%-99% sparsity
- Sparse matrix representations comprises only of non-zero values
- Data, Row and Col are the three 1D arrays used in this representation.

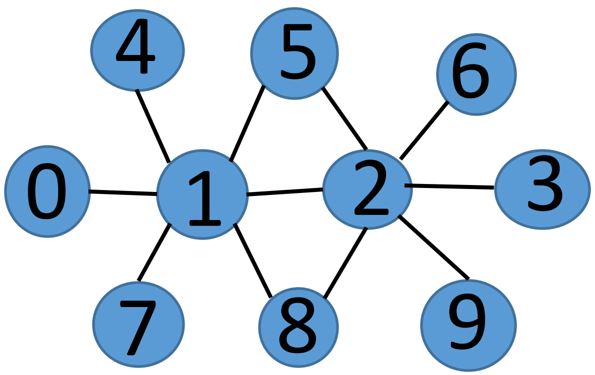
- Data: consists of the non-zero values
- Row: consists of row indices of the elements in Data
- Col: consists of column indices of the elements in Data
- Index Pointers: represents the number of elements in each row.

CSR Format



Source: Google Images

Single Source Shortest Path

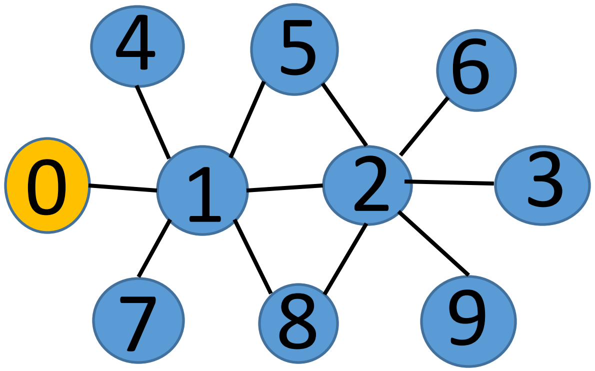


0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
1	0	2	4	5	7	8	1	3	5	6	8	9	2	1	1	2	2	1	1	2	2

0	1	7	13	14	15	17	18	19	21	22

	0	1	2	3	4	5	6	7	8	9
q										
vis	0	0	0	0	0	0	0	0	0	0
dist	inf	inf	inf	inf	inf	inf	inf	inf	inf	inf

Single Source Shortest Path



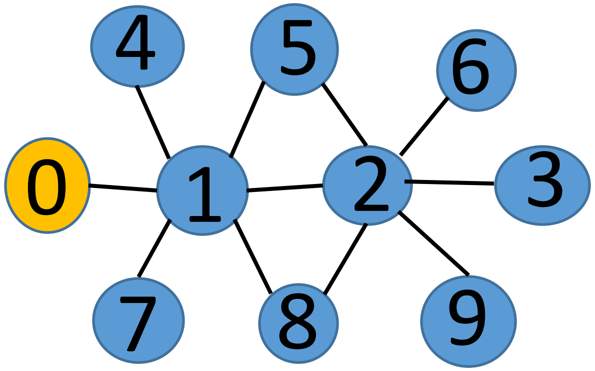

```
REPEAT 'V' TIMES:
  u = 0
  for all neighbors 'v' of 'u' do
    if(!vis[v] && dist[v] > dist[u] + edge_weight) do
            dist[v] = dist[u] + edge_weight
            vis[v] = 1
            add 'v' to q
  vis[u] = 1
```

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
1	0	2	4	5	7	8	1	3	5	6	8	9	2	1	1	2	2	1	1	2	2

0	1	7	13	14	15	17	18	19	21	22

	0	1	2	3	4	5	6	7	8	9
q	**0**									
vis	0	0	0	0	0	0	0	0	0	0
dist	0	inf	inf	inf	inf	inf	inf	inf	inf	inf

Single Source Shortest Path



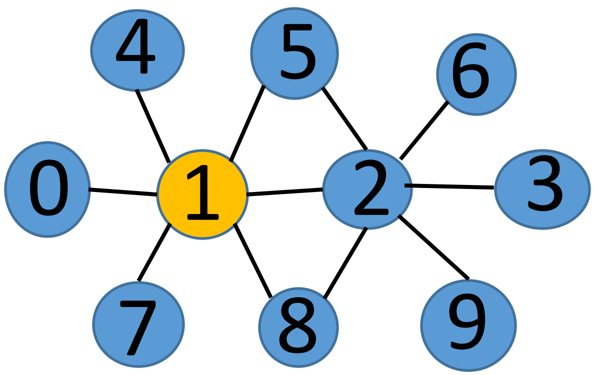

```
REPEAT 'V' TIMES:
  u = 0
  for all neighbors 'v' of 'u' do
    if(!vis[v] && dist[v] > dist[u] + edge_weight) do
            dist[v] = dist[u] + edge_weight
            vis[v] = 1
            add 'v' to q
  vis[u] = 1
```

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
1	0	2	4	5	7	8	1	3	5	6	8	9	2	1	1	2	2	1	1	2	2

0	1	7	13	14	15	17	18	19	21	22

	0	1	2	3	4	5	6	7	8	9
q	**0**	**1**								
vis	1	1	0	0	0	0	0	0	0	0
dist	0	1	inf	inf	inf	inf	inf	inf	inf	inf

Single Source Shortest Path



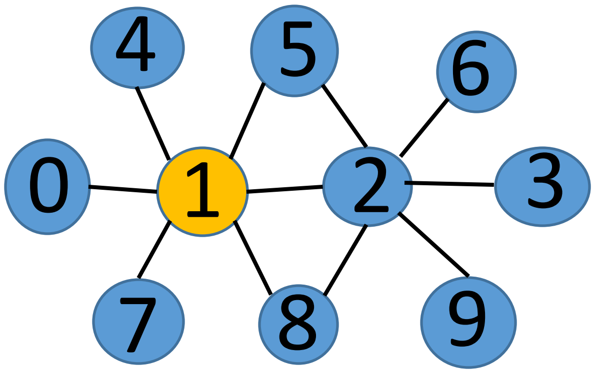

```
REPEAT 'V' TIMES:
  u = 1
  for all neighbors 'v' of 'u' do
    if(!vis[v] && dist[v] > dist[u] + edge_weight) do
              dist[v] = dist[u] + edge_weight
              vis[v] = 1
              add 'v' to q
  vis[u] = 1
```

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
1	0	2	4	5	7	8	1	3	5	6	8	9	2	1	1	2	2	1	1	2	2

0	1	7	13	14	15	17	18	19	21	22

	0	1	2	3	4	5	6	7	8	9
q	**0**	**1**								
vis	1	1	0	0	0	0	0	0	0	0
dist	0	1	inf	inf	inf	inf	inf	inf	inf	inf

Single Source Shortest Path



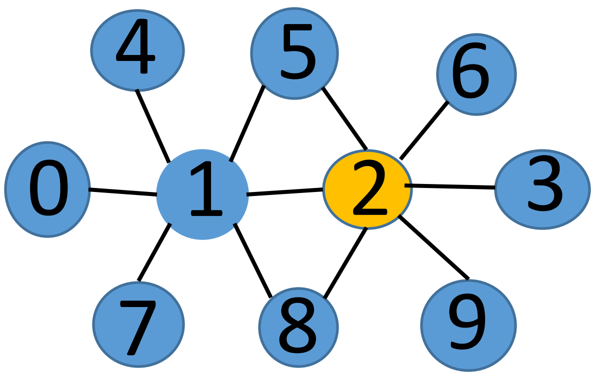
```
REPEAT 'V' TIMES:
  u = 1
  for all neighbors 'v' of 'u' do
    if(!vis[v] && dist[v] > dist[u] + edge_weight) do
            dist[v] = dist[u] + edge_weight
            vis[v] = 1
            add 'v' to q
  vis[u] = 1
```

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
1	0	2	4	5	7	8	1	3	5	6	8	9	2	1	1	2	2	1	1	2	2

0	1	7	13	14	15	17	18	19	21	22

	0	1	2	3	4	5	6	7	8	9
q	**0**	**1**	**2**	**4**	**5**	**7**	**8**			
vis	1	1	1	0	1	1	0	1	1	0
dist	0	1	2	inf	2	2	inf	2	2	inf

Single Source Shortest Path



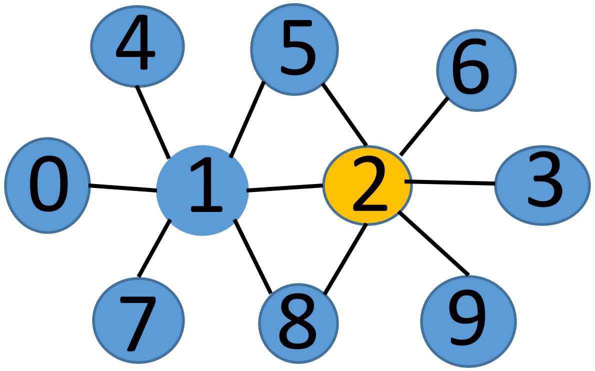

```
REPEAT 'V' TIMES:
  u = 2
  for all neighbors 'v' of 'u' do
    if(!vis[v] && dist[v] > dist[u] + edge_weight) do
            dist[v] = dist[u] + edge_weight
            vis[v] = 1
            add 'v' to q
  vis[u] = 1
```

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
1	0	2	4	5	7	8	1	3	5	6	8	9	2	1	1	2	2	1	1	2	2

0	1	7	13	14	15	17	18	19	21	22

	0	1	2	3	4	5	6	7	8	9
q	**0**	**1**	**2**	**4**	**5**	**7**	**8**			
vis	1	1	1	0	1	1	0	1	1	0
dist	0	1	2	inf	2	2	inf	2	2	inf

Single Source Shortest Path



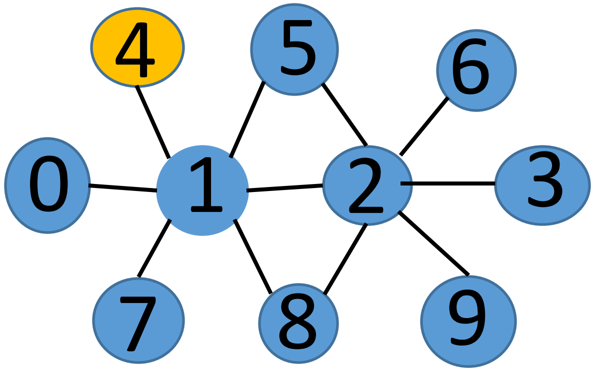

```
REPEAT 'V' TIMES:
  u = 2
  for all neighbors 'v' of 'u' do
    if(!vis[v] && dist[v] > dist[u] + edge_weight) do
            dist[v] = dist[u] + edge_weight
            vis[v] = 1
            add 'v' to q
  vis[u] = 1
```

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
1	0	2	4	5	7	8	1	3	5	6	8	9	2	1	1	2	2	1	1	2	2

0	1	7	13	14	15	17	18	19	21	22

	0	1	2	3	4	5	6	7	8	9
q	**0**	**1**	**2**	**4**	**5**	**7**	**8**	**3**	**6**	**9**
vis	1	1	1	1	1	1	1	1	1	1
dist	0	1	2	3	2	2	3	2	2	3

Single Source Shortest Path



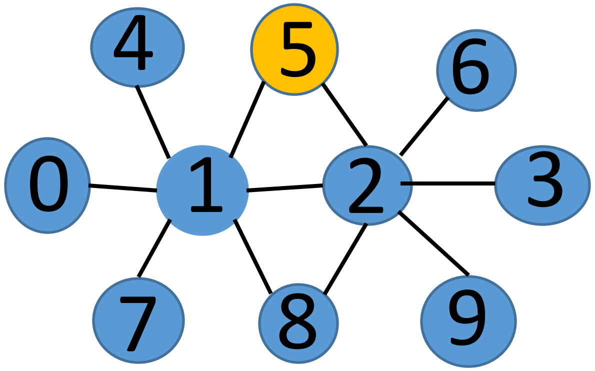

```
REPEAT 'V' TIMES:
  u = 4
  for all neighbors 'v' of 'u' do
    if(!vis[v] && dist[v] > dist[u] + edge_weight) do
            dist[v] = dist[u] + edge_weight
            vis[v] = 1
            add 'v' to q
  vis[u] = 1
```

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
1	0	2	4	5	7	8	1	3	5	6	8	9	2	1	1	2	2	1	1	2	2

0	1	7	13	14	15	17	18	19	21	22

	0	1	2	3	4	5	6	7	8	9
q	0	1	2	4	5	7	8	3	6	9
vis	1	1	1	1	1	1	1	1	1	1
dist	0	1	2	3	2	2	3	2	2	3

Single Source Shortest Path



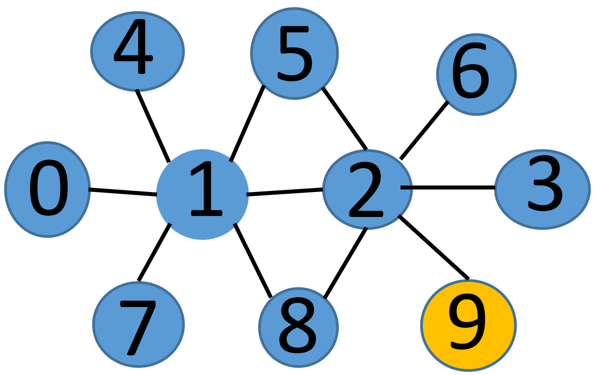
```
REPEAT 'V' TIMES:
  u = 5
  for all neighbors 'v' of 'u' do
    if(!vis[v]  && dist[v] > dist[u] + edge_weight) do
            dist[v] = dist[u] + edge_weight
            vis[v] = 1
            add 'v' to q
  vis[u] = 1
```

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
1	0	2	4	5	7	8	1	3	5	6	8	9	2	1	1	2	2	1	1	2	2

0	1	7	13	14	15	17	18	19	21	22

	0	1	2	3	4	5	6	7	8	9
q	**0**	**1**	**2**	**4**	**5**	**7**	**8**	**3**	**6**	**9**
vis	1	1	1	1	1	1	1	1	1	1
dist	0	1	2	3	2	2	3	2	2	3

Single Source Shortest Path




```
REPEAT 'V' TIMES:
  u = 9
  for all neighbors 'v' of 'u' do  #pragma HLS PIPELINE
    if(!vis[v]  && dist[v] > dist[u] + edge_weight) do
              dist[v] = dist[u] + edge_weight
              vis[v] = 1
              add 'v' to q
  vis[u] = 1
```

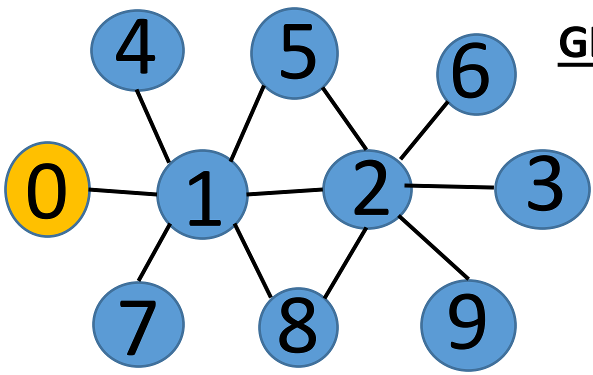
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
1	0	2	4	5	7	8	1	3	5	6	8	9	2	1	1	2	2	1	1	2	2

0	1	7	13	14	15	17	18	19	21	22

	0	1	2	3	4	5	6	7	8	9
q	**0**	**1**	**2**	**4**	**5**	**7**	**8**	**3**	**6**	**9**
vis	1	1	1	1	1	1	1	1	1	1
dist	0	1	2	3	2	2	3	2	2	3

FINAL DISTANCES

Single Source Shortest Path – PARTITIONING METHODOLOGY



GLOBAL

	0	1	2	3	4	5	6	7	8	9
q	**0**									
vis	0	0	0	0	0	0	0	0	0	0
dist	0	inf	inf	inf	inf	inf	inf	inf	inf	inf

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
1	0	2	4	5	7	8	1	3	5	6	8	9	2	1	1	2	2	1	1	2	2

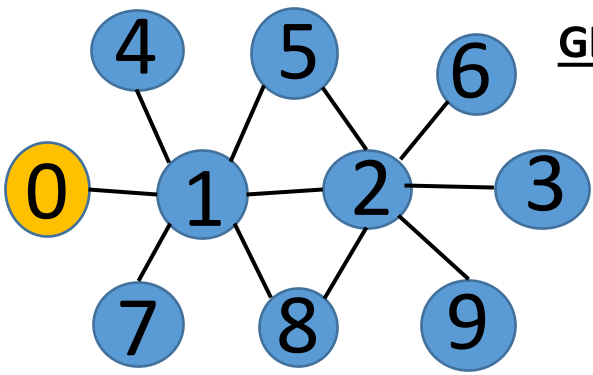
0	1	7	13	14	15	17	18	19	21	22
1	6	6	1	1	2	1	1	2	1	

LOCAL

queue						
visited	0					
distance	inf					
neighbors	1					

```
REPEAT 'V' TIMES:
  u = 0
  partition(u, dist[u], degree[u], queue, visited,
                distance, neighbors)
  vis[u] = 1
```

Single Source Shortest Path – PARTITIONING METHODOLOGY



GLOBAL

	0	1	2	3	4	5	6	7	8	9
q	**0**									
vis	0	0	0	0	0	0	0	0	0	0
dist	0	inf	inf	inf	inf	inf	inf	inf	inf	inf

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
1	0	2	4	5	7	8	1	3	5	6	8	9	2	1	1	2	2	1	1	2	2

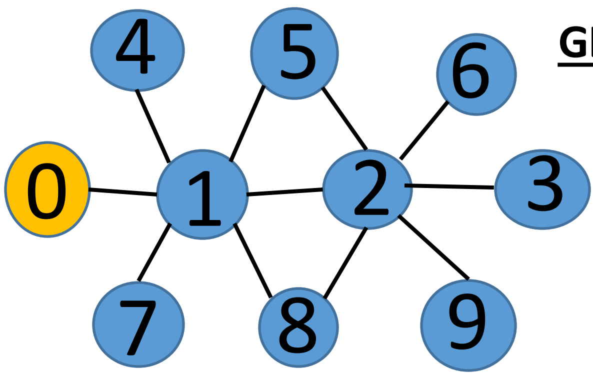
0	1	7	13	14	15	17	18	19	21	22
1	6	6	1	1	2	1	1	2	1	

LOCAL

queue	**1**					
visited	0					
distance	1					
neighbors	1					

```
REPEAT 'V' TIMES:
  u = 0
  partition(u, dist[u], degree[u], queue, visited,
                 distance, neighbors)
  vis[u] = 1
```

Single Source Shortest Path – PARTITIONING METHODOLOGY



GLOBAL

	0	1	2	3	4	5	6	7	8	9
q	**0**	**1**								
vis	1	1	0	0	0	0	0	0	0	0
dist	0	1	inf	inf	inf	inf	inf	inf	inf	inf

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
1	0	2	4	5	7	8	1	3	5	6	8	9	2	1	1	2	2	1	1	2	2

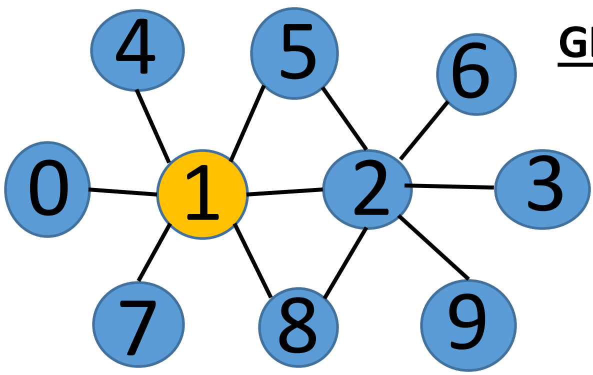
0	1	7	13	14	15	17	18	19	21	22
1	6	6	1	1	2	1	1	2	1	

LOCAL

queue	**1**					
visited	1					
distance	1					
neighbors	1					

```
REPEAT 'V' TIMES:
  u = 0
  partition(u, dist[u], degree[u], queue, visited,
                distance, neighbors)
  vis[u] = 1
```

Single Source Shortest Path – PARTITIONING METHODOLOGY



GLOBAL

	0	1	2	3	4	5	6	7	8	9
q	0	1								
vis	1	1	0	0	0	0	0	0	0	0
dist	0	1	inf	inf	inf	inf	inf	inf	inf	inf

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
1	0	2	4	5	7	8	1	3	5	6	8	9	2	1	1	2	2	1	1	2	2

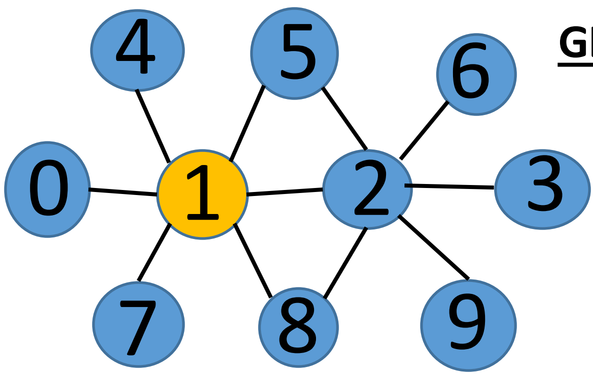
0	1	7	13	14	15	17	18	19	21	22
1	6	6	1	1	2	1	1	2	1	

LOCAL

queue						
visited	1	0	0	0	0	0
distance	0	inf	inf	inf	inf	inf
neighbors	0	2	4	5	7	8

REPEAT ‘V’ TIMES:
u = 1
partition(u, dist[u], degree[u], queue, visited,
distance, neighbors)
vis[u] = 1

Single Source Shortest Path – PARTITIONING METHODOLOGY



GLOBAL

	0	1	2	3	4	5	6	7	8	9
q	**0**	**1**								
vis	1	1	0	0	0	0	0	0	0	0
dist	0	1	inf	inf	inf	inf	inf	inf	inf	inf

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
1	0	2	4	5	7	8	1	3	5	6	8	9	2	1	1	2	2	1	1	2	2

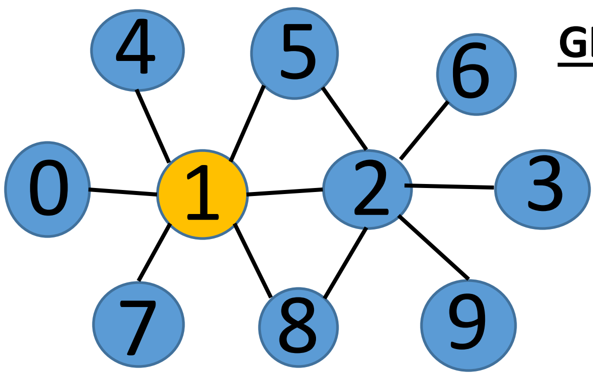
0	1	7	13	14	15	17	18	19	21	22
1	6	6	1	1	2	1	1	2	1	

LOCAL

queue	**2**	**4**	**5**	**7**	**8**	
visited	1	1	1	1	1	1
distance	0	2	2	2	2	2
neighbors	0	2	4	5	7	8

```
REPEAT 'V' TIMES:
  u = 1
  partition(u, dist[u], degree[u], queue, visited,
                 distance, neighbors)
  vis[u] = 1
```

Single Source Shortest Path – PARTITIONING METHODOLOGY



GLOBAL

	0	1	2	3	4	5	6	7	8	9
q	**0**	**1**	**2**	**4**	**5**	**7**	**8**			
vis	1	1	1	0	1	1	0	1	1	0
dist	0	1	2	inf	2	2	inf	2	2	inf

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
1	0	2	4	5	7	8	1	3	5	6	8	9	2	1	1	2	2	1	1	2	2

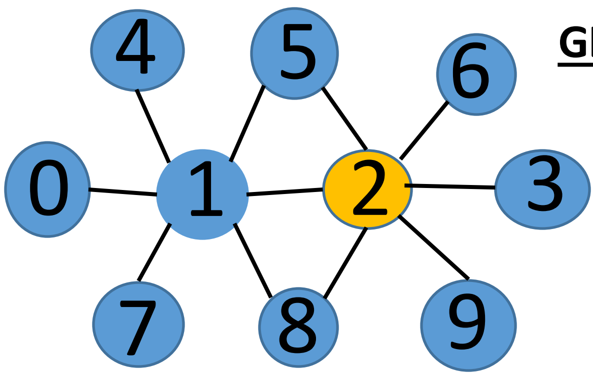
0	1	7	13	14	15	17	18	19	21	22
1	6	6	1	1	2	1	1	2	1	

LOCAL

queue	**2**	**4**	**5**	**7**	**8**	
visited	1	1	1	1	1	1
distance	0	2	2	2	2	2
neighbors	0	2	4	5	7	8

```
REPEAT 'V' TIMES:
  u = 1
  partition(u, dist[u], degree[u], queue, visited,
                 distance, neighbors)
  vis[u] = 1
```

Single Source Shortest Path – PARTITIONING METHODOLOGY



GLOBAL

	0	1	2	3	4	5	6	7	8	9
q	**0**	**1**	**2**	**4**	**5**	**7**	**8**			
vis	1	1	1	0	1	1	0	1	1	0
dist	0	1	2	inf	2	2	inf	2	2	inf

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
1	0	2	4	5	7	8	1	3	5	6	8	9	2	1	1	2	2	1	1	2	2

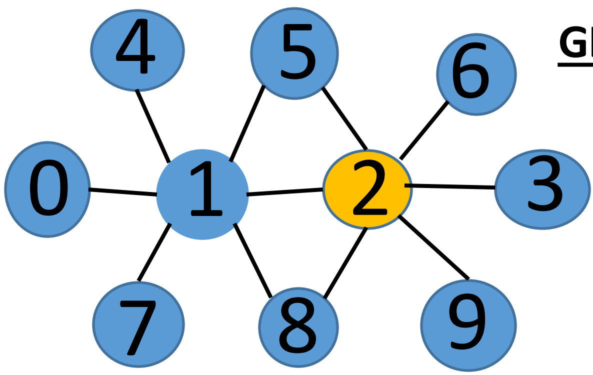
0	1	7	13	14	15	17	18	19	21	22
1	6	6	1	1	2	1	1	2	1	

LOCAL

queue						
visited	1	0	1	0	1	0
distance	1	inf	1	inf	2	inf
neighbors	1	3	5	6	8	9

REPEAT ‘V’ TIMES:
u = 2
partition(u, dist[u], degree[u], queue, visited,
distance, neighbors)
vis[u] = 1

Single Source Shortest Path – PARTITIONING METHODOLOGY



GLOBAL

	0	1	2	3	4	5	6	7	8	9
q	**0**	**1**	**2**	**4**	**5**	**7**	**8**			
vis	1	1	1	0	1	1	0	1	1	0
dist	0	1	2	inf	2	2	inf	2	2	inf

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
1	0	2	4	5	7	8	1	3	5	6	8	9	2	1	1	2	2	1	1	2	2

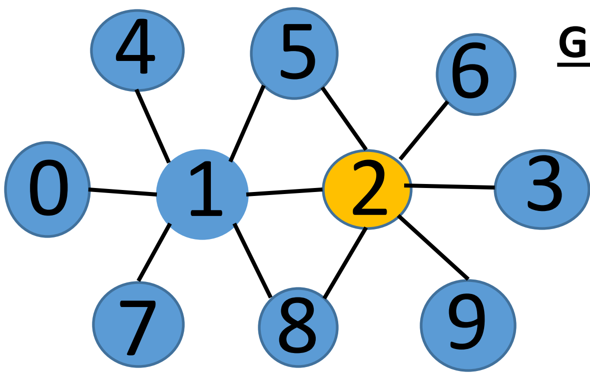
0	1	7	13	14	15	17	18	19	21	22
1	6	6	1	1	2	1	1	2	1	

LOCAL

queue	**3**	**6**	**9**			
visited	1	1	1	1	1	1
distance	1	3	1	3	2	3
neighbors	1	3	5	6	8	9

```
REPEAT 'V' TIMES:
  u = 2
  partition(u, dist[u], degree[u], queue, visited,
                distance, neighbors)
  vis[u] = 1
```

Single Source Shortest Path – PARTITIONING METHODOLOGY



GLOBAL

	0	1	2	3	4	5	6	7	8	9
q	**0**	**1**	**2**	**4**	**5**	**7**	**8**	**3**	**6**	**9**
vis	1	1	1	1	1	1	1	1	1	1
dist	0	1	2	3	2	2	3	2	2	3

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
1	0	2	4	5	7	8	1	3	5	6	8	9	2	1	1	2	2	1	1	2	2

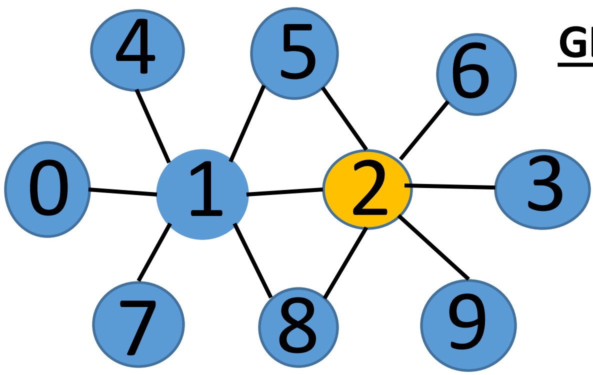
0	1	7	13	14	15	17	18	19	21	22
1	6	6	1	1	2	1	1	2	1	

LOCAL

queue	**3**	**6**	**9**			
visited	1	1	1	1	1	1
distance	1	3	1	3	2	3
neighbors	1	3	5	6	8	9

REPEAT ‘V’ TIMES:
u = 2
partition(u, dist[u], degree[u], queue, visited, distance, neighbors)
vis[u] = 1

Single Source Shortest Path – PARTITIONING METHODOLOGY



GLOBAL

	0	1	2	3	4	5	6	7	8	9
q	**0**	**1**	**2**	**4**	**5**	**7**	**8**	**3**	**6**	**9**
vis	1	1	1	1	1	1	1	1	1	1
dist	0	1	2	3	2	2	3	2	2	3

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
1	0	2	4	5	7	8	1	3	5	6	8	9	2	1	1	2	2	1	1	2	2

0	1	7	13	14	15	17	18	19	21	22
1	6	6	1	1	2	1	1	2	1	

LOCAL

queue						
visited	1					
distance	2					
neighbors	2					

REPEAT 'V' TIMES:
u = 9
partition(u, dist[u], degree[u], queue, visited,
distance, neighbors)
vis[u] = 1

PageRank

PageRank is a **link analysis algorithm** which assigns a numerical weighting to each element of a hyperlinked set of nodes with the purpose of measuring its relative importance within the set.

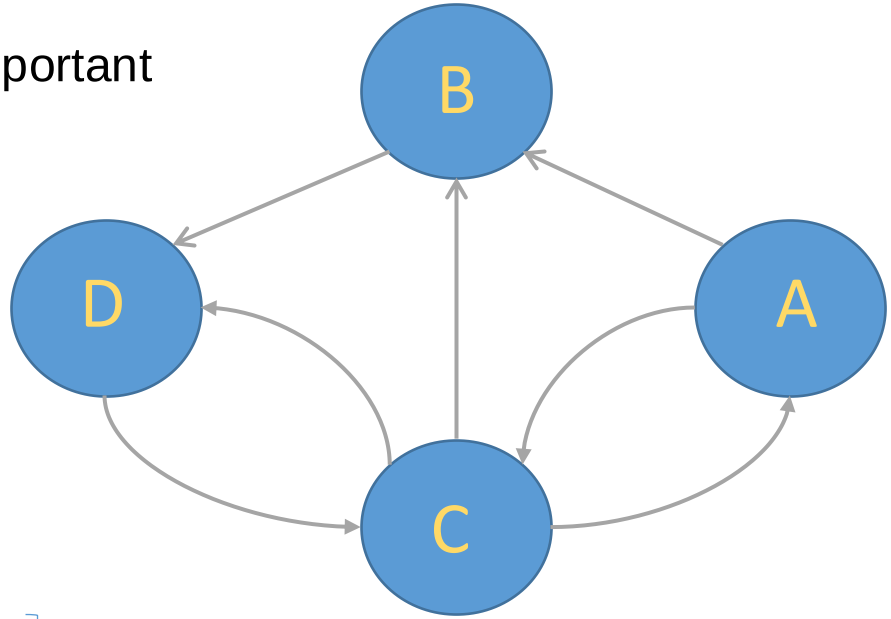
More links to a page ⇒ Page is more important

Adjacency Matrix:

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

Outlink vector:

$$\begin{bmatrix} 2 \\ 1 \\ 3 \\ 1 \end{bmatrix}$$



Normalized Matrix:

$$\begin{bmatrix} 0 & 0.5 & 0.5 & 0 \\ 0 & 0 & 0 & 1 \\ 0.33 & 0.33 & 0 & 0.33 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

Transpose

$$\begin{bmatrix} 0 & 0 & 0.33 & 0 \\ 0.5 & 0 & 0.33 & 0 \\ 0.5 & 0 & 0 & 1 \\ 0 & 1 & 0.33 & 0 \end{bmatrix}$$

GRAPH CONSIDERED:
4 NODES
7 EDGES

PageRank

COO Format:

$$\text{VALUE:} \begin{bmatrix} 0.33 & 0.5 & 0.33 & 0.5 & 1 & 1 & 0.33 \end{bmatrix}$$

$$\text{ROW:} \begin{bmatrix} 0 & 1 & 1 & 2 & 2 & 3 & 3 \end{bmatrix}$$

$$\text{COL:} \begin{bmatrix} 2 & 0 & 2 & 0 & 4 & 1 & 2 \end{bmatrix}$$

PAGERANK Vector:

$$\begin{bmatrix} 0.25 \\ 0.25 \\ 0.25 \\ 0.25 \end{bmatrix}$$

PageRank Formula:

$$PR(p_i) = \frac{1-d}{N} + d\left(\sum_{p_j \text{ links to } p_i} \frac{PR(p_j)}{L(p_j)} + \sum_{p_j \text{ has no out-links}} \frac{PR(p_j)}{N}\right)$$

PageRank

MATRIX MULTIPLICATION: (ADJACENCY MATRIX)

```
p_new[0] = p_new[0] + a[0][0]*p[0]
         = p_new[0] + a[0][1]*p[1]
         = p_new[0] + a[0][2]*p[2]
         = p_new[0] + a[0][3]*p[3]

p_new[1] = p_new[1] + a[1][0]*p[0]
         = p_new[1] + a[1][1]*p[1]
         = p_new[1] + a[1][2]*p[2]
         = p_new[1] + a[1][3]*p[3]

p_new[2] = p_new[2] + a[2][0]*p[0]
         = p_new[2] + a[2][1]*p[1]
         = p_new[2] + a[2][2]*p[2]
         = p_new[2] + a[2][3]*p[3]

p_new[3] = p_new[3] + a[3][0]*p[0]
         = p_new[3] + a[3][1]*p[1]
         = p_new[3] + a[3][2]*p[2]
         = p_new[3] + a[3][3]*p[3]
```

PageRank

MATRIX MULTIPLICATION: (ADJACENCY MATRIX)

```
p_new[0] = p_new[0] + a[0][0]*p[0]
         = p_new[0] + a[0][1]*p[1]
         = p_new[0] + a[0][2]*p[2]
         = p_new[0] + a[0][3]*p[3]

p_new[1] = p_new[1] + a[1][0]*p[0]
         = p_new[1] + a[1][1]*p[1]
         = p_new[1] + a[1][2]*p[2]
         = p_new[1] + a[1][3]*p[3]

p_new[2] = p_new[2] + a[2][0]*p[0]
         = p_new[2] + a[2][1]*p[1]
         = p_new[2] + a[2][2]*p[2]
         = p_new[2] + a[2][3]*p[3]

p_new[3] = p_new[3] + a[3][0]*p[0]
         = p_new[3] + a[3][1]*p[1]
         = p_new[3] + a[3][2]*p[2]
         = p_new[3] + a[3][3]*p[3]
```

MATRIX MULTIPLICATION: (COO Format)

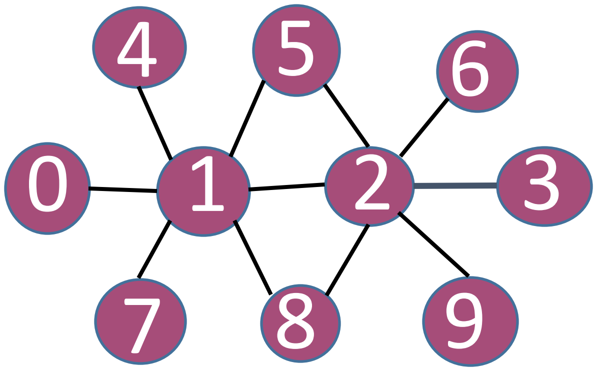
```
p_new[0] = p_new[0] + val[0]*p[2]

p_new[1] = p_new[1] + val[1]*p[0]
         = p_new[1] + val[2]*p[2]

p_new[2] = p_new[2] + val[3]*p[0]
         = p_new[2] + val[4]*p[4]

p_new[3] = p_new[3] + val[5]*p[1]
         = p_new[3] + val[6]*p[2]
```

Breadth First Search

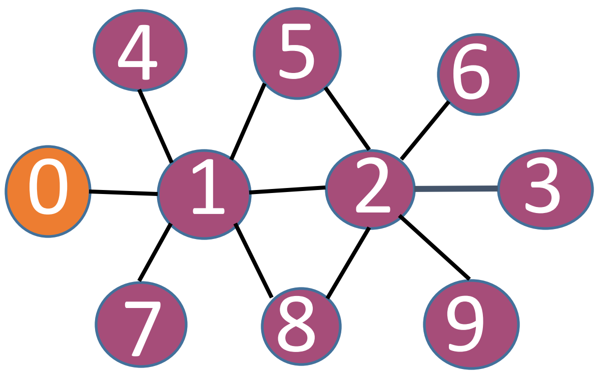


0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
0	1	1	1	1	1	1	2	2	2	2	2	2	3	4	5	5	6	7	8	8	9
1	0	2	4	5	7	8	1	3	5	6	8	9	2	1	1	2	2	1	1	2	2

0 1 7 13 14 15 17 18 19 21 22

	0	1	2	3	4	5	6	7	8	9
q										
vis	0	0	0	0	0	0	0	0	0	0

Breadth First Search



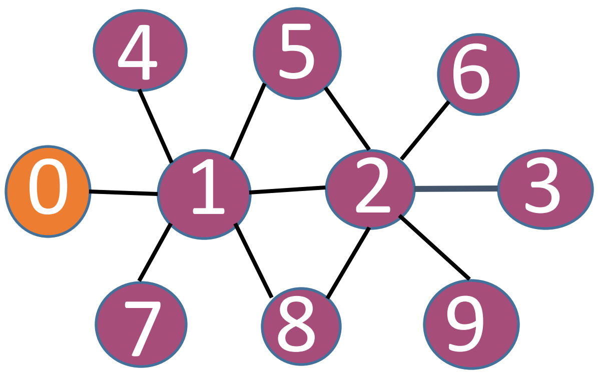

```
REPEAT 'V' TIMES:
  u = 0
  for all neighbors 'v' of 'u' do
    if(!vis[v]) do
              vis[v] = 1
              add 'v' to q
  vis[u] = 1
```

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
1	0	2	4	5	7	8	1	3	5	6	8	9	2	1	1	2	2	1	1	2	2

0	1	7	13	14	15	17	18	19	21	22

	0	1	2	3	4	5	6	7	8	9
q	0									
vis	0	0	0	0	0	0	0	0	0	0

Breadth First Search



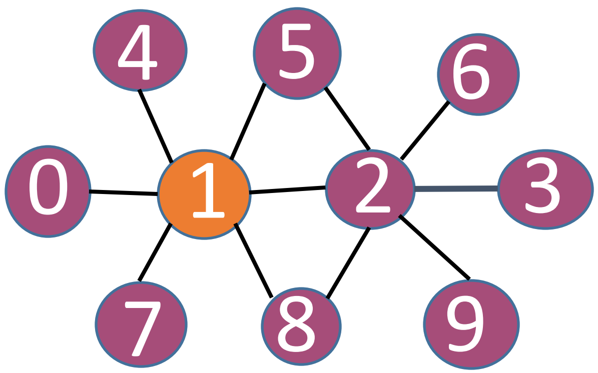

```
REPEAT 'V' TIMES:
  u = 0
  for all neighbors 'v' of 'u' do
    if(!vis[v]) do
            vis[v] = 1
            add 'v' to q
  vis[u] = 1
```

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
1	0	2	4	5	7	8	1	3	5	6	8	9	2	1	1	2	2	1	1	2	2

0	1	7	13	14	15	17	18	19	21	22

	0	1	2	3	4	5	6	7	8	9
q	0	1								
vis	1	1	0	0	0	0	0	0	0	0

Breadth First Search



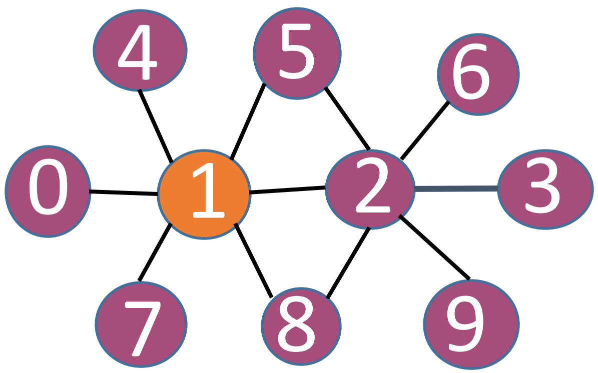

```
REPEAT 'V' TIMES:
  u = 1
  for all neighbors 'v' of 'u' do
    if(!vis[v]) do
            vis[v] = 1
            add 'v' to q
  vis[u] = 1
```

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
1	0	2	4	5	7	8	1	3	5	6	8	9	2	1	1	2	2	1	1	2	2

0	1	7	13	14	15	17	18	19	21	22

	0	1	2	3	4	5	6	7	8	9
q	0	1								
vis	1	1	0	0	0	0	0	0	0	0

Breadth First Search



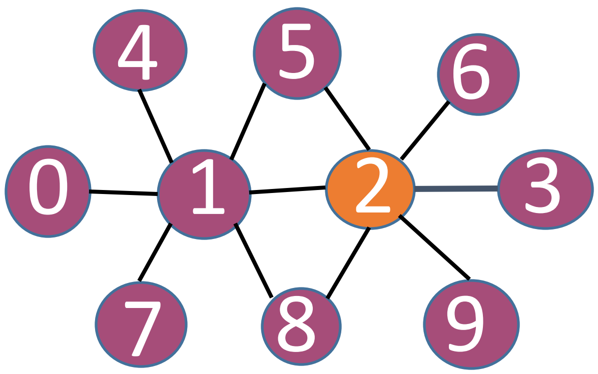

```
REPEAT 'V' TIMES:
  u = 1
  for all neighbors 'v' of 'u' do
    if(!vis[v]) do
          vis[v] = 1
          add 'v' to q
  vis[u] = 1
```

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
1	0	2	4	5	7	8	1	3	5	6	8	9	2	1	1	2	2	1	1	2	2

0	1	7	13	14	15	17	18	19	21	22

	0	1	2	3	4	5	6	7	8	9
q	0	1	2	4	5	7	8			
vis	1	1	1	0	1	1	0	1	1	0

Breadth First Search



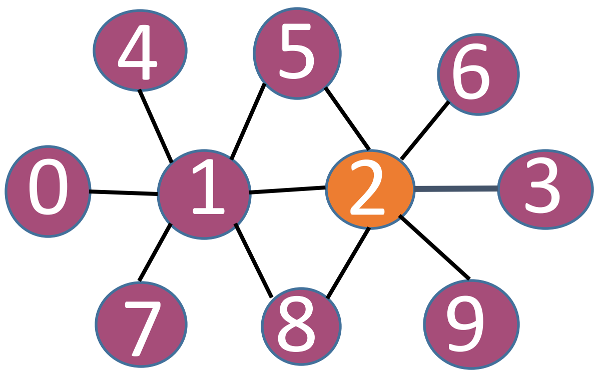
```
REPEAT 'V' TIMES:
  u = 2
  for all neighbors 'v' of 'u' do
    if(!vis[v]) do
        vis[v] = 1
        add 'v' to q
  vis[u] = 1
```

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
1	0	2	4	5	7	8	1	3	5	6	8	9	2	1	1	2	2	1	1	2	2

0	1	7	13	14	15	17	18	19	21	22

	0	1	2	3	4	5	6	7	8	9
q	0	1	2	4	5	7	8			
vis	1	1	1	0	1	1	0	1	1	0

Breadth First Search



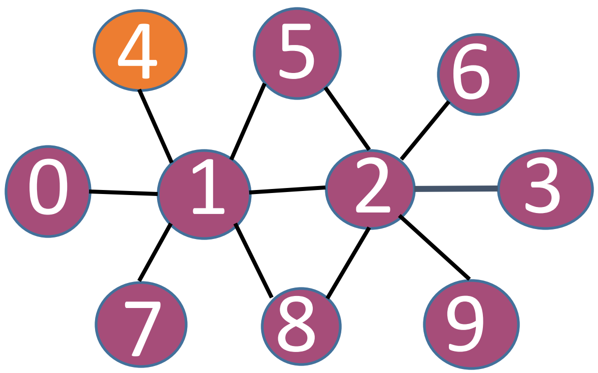

```
REPEAT 'V' TIMES:
  u = 2
  for all neighbors 'v' of 'u' do
    if(!vis[v]) do
              vis[v] = 1
              add 'v' to q
  vis[u] = 1
```

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
1	0	2	4	5	7	8	1	3	5	6	8	9	2	1	1	2	2	1	1	2	2

0	1	7	13	14	15	17	18	19	21	22

0	1	2	3	4	5	6	7	8	9

q	0	1	2	4	5	7	8	3	6	9
vis	1	1	1	1	1	1	1	1	1	1



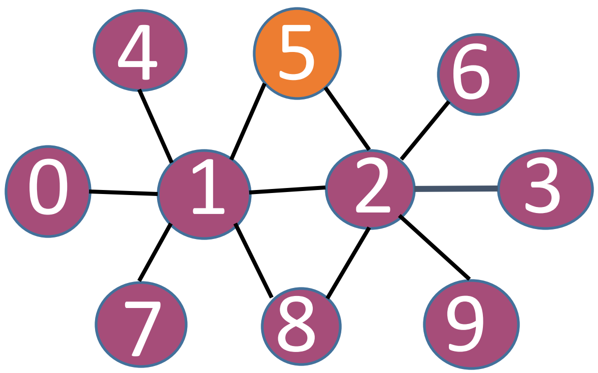

```
REPEAT 'V' TIMES:
  u = 4
  for all neighbors 'v' of 'u' do
    if(!vis[v]) do
          vis[v] = 1
          add 'v' to q
  vis[u] = 1
```

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
1	0	2	4	5	7	8	1	3	5	6	8	9	2	1	1	2	2	1	1	2	2

0	1	2	3	4	5	6	7	8	9
0	1	7	13	14	15	17	18	19	21

	0	1	2	3	4	5	6	7	8	9
q	0	1	2	4	5	7	8	3	6	9
vis	1	1	1	1	1	1	1	1	1	1

Breadth First Search



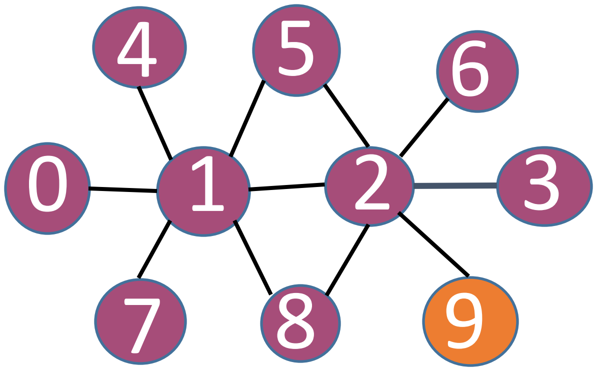

```
REPEAT 'V' TIMES:
  u = 5
  for all neighbors 'v' of 'u' do
    if(!vis[v]) do
          vis[v] = 1
          add 'v' to q
  vis[u] = 1
```

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
1	0	2	4	5	7	8	1	3	5	6	8	9	2	1	1	2	2	1	1	2	2

0	1	7	13	14	15	17	18	19	21	22

	0	1	2	3	4	5	6	7	8	9
q	0	1	2	4	5	7	8	3	6	9
vis	1	1	1	1	1	1	1	1	1	1

Breadth First Search




```
REPEAT 'V' TIMES:
  u = 9
  for all neighbors 'v' of 'u' do
    if(!vis[v]) do
          vis[v] = 1
          add 'v' to q
  vis[u] = 1
```

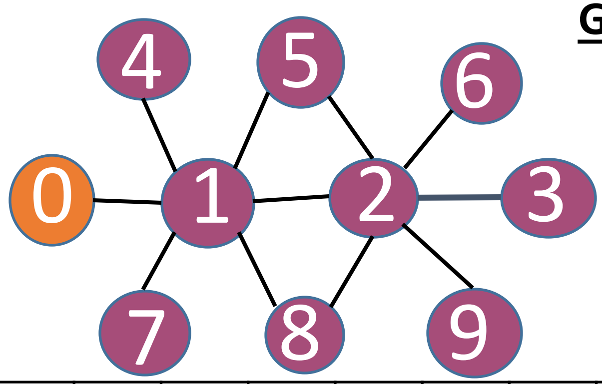
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
1	0	2	4	5	7	8	1	3	5	6	8	9	2	1	1	2	2	1	1	2	2

0	1	7	13	14	15	17	18	19	21	22

	0	1	2	3	4	5	6	7	8	9
q	0	1	2	4	5	7	8	3	6	9
vis	1	1	1	1	1	1	1	1	1	1

Final Traversal Order

Breadth First Search – PARTITIONING METHODOLOGY



GLOBAL

	0	1	2	3	4	5	6	7	8	9
q	0									
vis	0	0	0	0	0	0	0	0	0	0

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
1	0	2	4	5	7	8	1	3	5	6	8	9	2	1	1	2	2	1	1	2	2

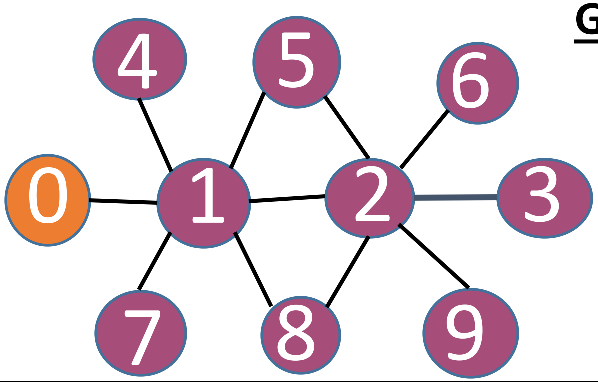
0	1	7	13	14	15	17	18	19	21	22
1	6	6	1	1	2	1	1	2	1	

LOCAL

queue						
visited	0					
neighbors	1					

REPEAT 'V' TIMES:
u = 0
partition(u, degree[u], queue, visited,neighbors)
vis[u] = 1

Breadth First Search– PARTITIONING METHODOLOGY



GLOBAL

	0	1	2	3	4	5	6	7	8	9
q	0									
vis	0	0	0	0	0	0	0	0	0	0

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
1	0	2	4	5	7	8	1	3	5	6	8	9	2	1	1	2	2	1	1	2	2

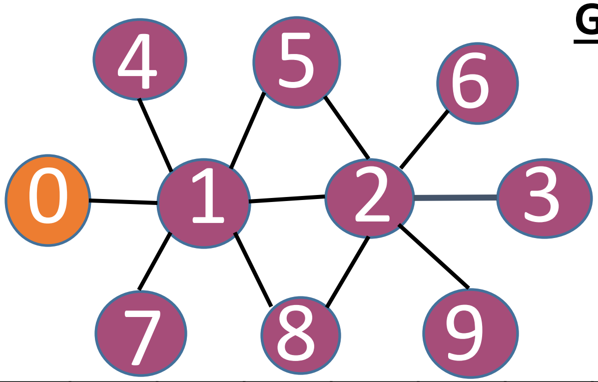
0	1	7	13	14	15	17	18	19	21	22
1	6	6	1	1	2	1	1	2	1	

LOCAL

queue	1					
visited	0					
neighbors	1					

```
REPEAT 'V' TIMES:
  u = 0
  partition(u, degree[u], queue, visited,neighbors)
  vis[u] = 1
```

Breadth First Search – PARTITIONING METHODOLOGY



GLOBAL

	0	1	2	3	4	5	6	7	8	9
q	0	1								
vis	1	1	0	0	0	0	0	0	0	0

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
1	0	2	4	5	7	8	1	3	5	6	8	9	2	1	1	2	2	1	1	2	2

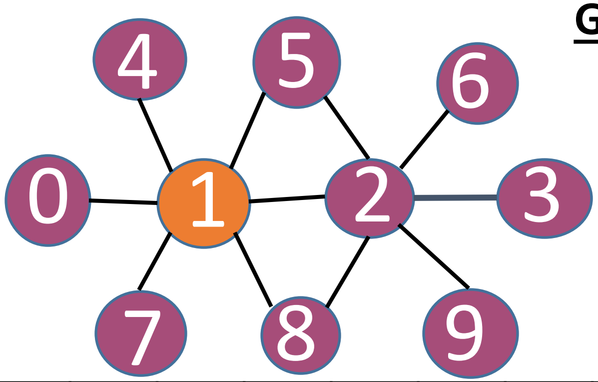
0	1	7	13	14	15	17	18	19	21	22
1	6	6	1	1	2	1	1	2	1	

LOCAL

queue	1					
visited	1					
neighbors	1					

```
REPEAT 'V' TIMES:
  u = 0
  partition(u, degree[u], queue, visited,neighbors)
  vis[u] = 1
```

Breadth First Search – PARTITIONING METHODOLOGY



GLOBAL

	0	1	2	3	4	5	6	7	8	9
q	0	1								
vis	1	1	0	0	0	0	0	0	0	0

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
1	0	2	4	5	7	8	1	3	5	6	8	9	2	1	1	2	2	1	1	2	2

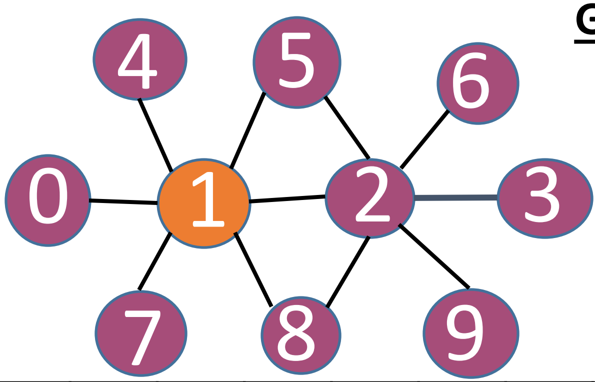
0	1	7	13	14	15	17	18	19	21	22
1	6	6	1	1	2	1	1	2	1	

LOCAL

queue						
visited	1	0	0	0	0	0
neighbors	0	2	4	5	7	8

```
REPEAT 'V' TIMES:
  u = 1
  partition(u, degree[u], queue, visited,neighbors)
  vis[u] = 1
```

Breadth First Search– PARTITIONING METHODOLOGY



GLOBAL

	0	1	2	3	4	5	6	7	8	9
q	0	1								
vis	1	1	0	0	0	0	0	0	0	0

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
1	0	2	4	5	7	8	1	3	5	6	8	9	2	1	1	2	2	1	1	2	2

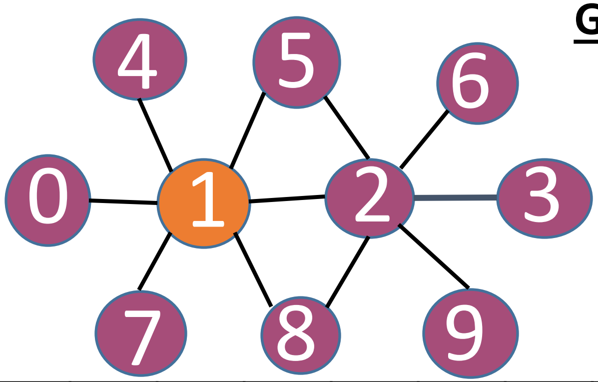
0	1	7	13	14	15	17	18	19	21	22
1	6	6	1	1	2	1	1	2	1	

LOCAL

queue	2	4	5	7	8	
visited	1	1	1	1	1	1
neighbor s	0	2	4	5	7	8

REPEAT 'V' TIMES:
u = 1
partition(u, degree[u], queue, visited,neighbors)
vis[u] = 1

Breadth First Search – PARTITIONING METHODOLOGY



GLOBAL

	0	1	2	3	4	5	6	7	8	9
q	0	1	2	4	5	7	8			
vis	1	1	1	0	1	1	0	1	1	0

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
1	0	2	4	5	7	8	1	3	5	6	8	9	2	1	1	2	2	1	1	2	2

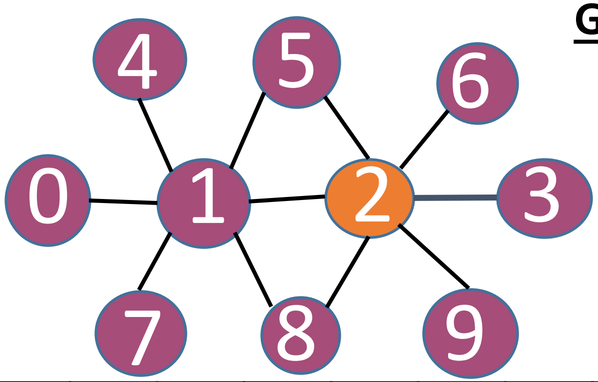
0	1	7	13	14	15	17	18	19	21	22
1	6	6	1	1	2	1	1	2	1	

LOCAL

queue	2	4	5	7	8	
visited	1	1	1	1	1	1
neighbors	0	2	4	5	7	8

```
REPEAT 'V' TIMES:
  u = 1
  partition(u, degree[u], queue, visited,neighbors)
  vis[u] = 1
```

Breadth First Search – PARTITIONING METHODOLOGY



GLOBAL

	0	1	2	3	4	5	6	7	8	9
q	0	1	2	4	5	7	8			
vis	1	1	1	0	1	1	0	1	1	0

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
1	0	2	4	5	7	8	1	3	5	6	8	9	2	1	1	2	2	1	1	2	2

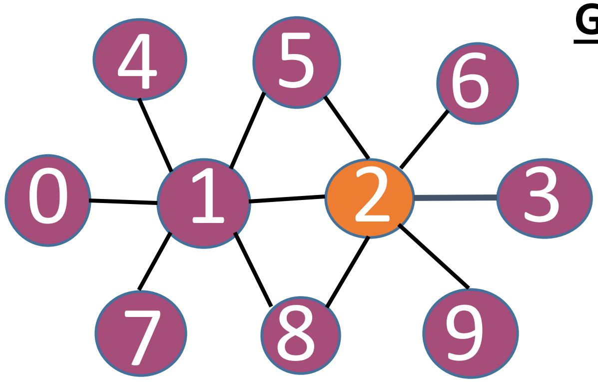
0	1	7	13	14	15	17	18	19	21	22
1	6	6	1	1	2	1	1	2	1	

LOCAL

queue						
visited	1	0	1	0	1	0
neighbors	1	3	5	6	8	9

REPEAT 'V' TIMES:
u = 2
partition(u, degree[u], queue, visited,neighbors)
vis[u] = 1

Breadth First Search– PARTITIONING METHODOLOGY



GLOBAL

	0	1	2	3	4	5	6	7	8	9
q	0	1	2	4	5	7	8			
vis	1	1	1	0	1	1	0	1	1	0

1	0	2	4	5	7	8	1	3	5	6	8	9	2	1	1	2	2	1	1	2	2

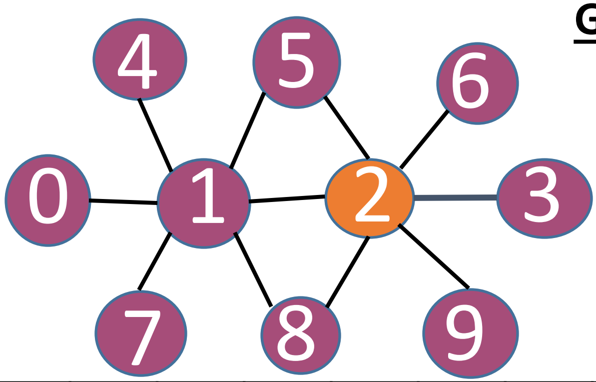
0	1	7	13	14	15	17	18	19	21	22
1	6	6	1	1	2	1	1	2	1	

LOCAL

queue	3	6	9			
visited	1	1	1	1	1	1
neighbors	1	3	5	6	8	9

```
REPEAT 'V' TIMES:
  u = 2
  partition(u, degree[u], queue, visited,neighbors)
  vis[u] = 1
```

Breadth First Search – PARTITIONING METHODOLOGY



GLOBAL

	0	1	2	3	4	5	6	7	8	9
q	0	1	2	4	5	7	8	3	6	9
vis	1	1	1	1	1	1	1	1	1	1

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
1	0	2	4	5	7	8	1	3	5	6	8	9	2	1	1	2	2	1	1	2	2

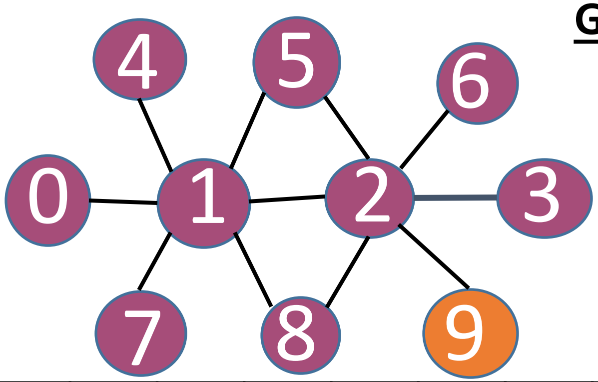
0	1	7	13	14	15	17	18	19	21	22
1	6	6	1	1	2	1	1	2	1	

LOCAL

queue	3	6	9			
visited	1	1	1	1	1	1
neighbors	1	3	5	6	8	9

REPEAT ‘V’ TIMES:
u = 2
partition(u, degree[u], queue, visited,neighbors)
vis[u] = 1

Breadth First Search – PARTITIONING METHODOLOGY



GLOBAL

	0	1	2	3	4	5	6	7	8	9
q	0	1	2	4	5	7	8	3	6	9
vis	1	1	1	1	1	1	1	1	1	1

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
1	0	2	4	5	7	8	1	3	5	6	8	9	2	1	1	2	2	1	1	2	2

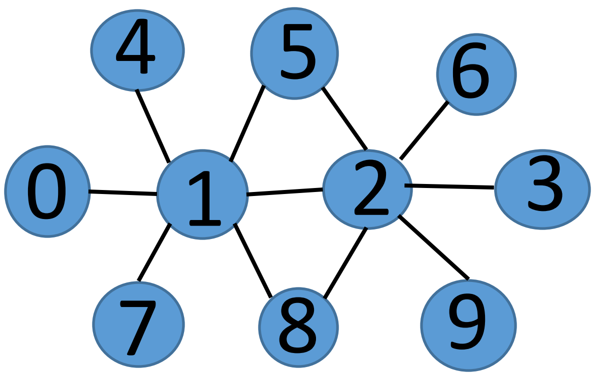
0	1	7	13	14	15	17	18	19	21	22
1	6	6	1	1	2	1	1	2	1	

LOCAL

queue						
visited	1					
neighbors	2					

REPEAT 'V' TIMES:
  u = 9
  partition(u, degree[u], queue, visited,neighbors)
  vis[u] = 1

DEPTH FIRST SEARCH (DFS)

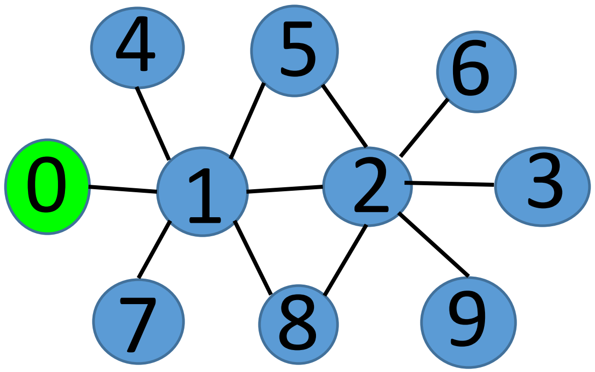


	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
U	0	1	1	1	1	1	1	2	2	2	2	2	2	3	4	5	5	6	7	8	8	9
V	1	0	2	4	5	7	8	1	3	5	6	8	9	2	1	1	2	2	1	1	2	2

ST	0	0	0	0	0	0	0	0	0	0

	0	1	2	3	4	5	6	7	8	9
vis	0	0	0	0	0	0	0	0	0	0
d	0	0	0	0	0	0	0	0	0	0

DEPTH FIRST SEARCH (DFS)



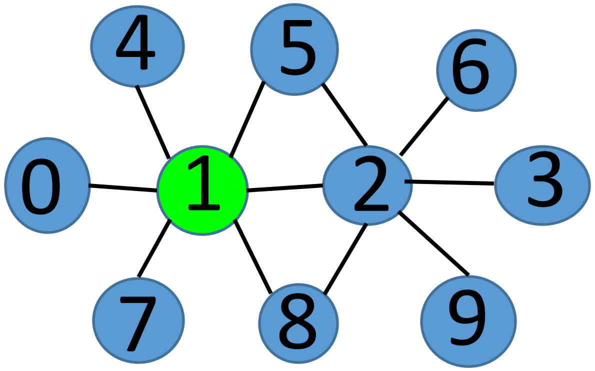

```
REPEAT 'V' TIMES:
  s = 0
              add s to ST
                 remove s from ST
                 if(vis[s]==0) do
                 vis[s] = 1;
                       add s to d
  for all neighbors 'v' of 's' do
     if((u[i] == s) && (vis[v[i]] == 0))do
              ST[++top] = v[i]
```

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
U	0	1	1	1	1	1	1	2	2	2	2	2	2	3	4	5	5	6	7	8	8	9
V	1	0	2	4	5	7	8	1	3	5	6	8	9	2	1	1	2	2	1	1	2	2

ST	0									

	0	1	2	3	4	5	6	7	8	9
vis	1	0	0	0	0	0	0	0	0	0
d	0									

DEPTH FIRST SEARCH(DFS)



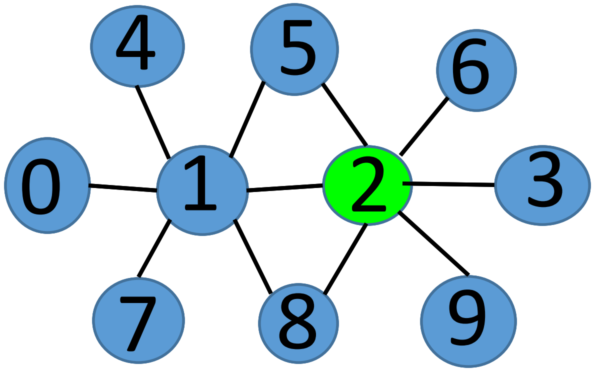

```
REPEAT 'V' TIMES:
  s = 0
            add s to ST
              remove s from ST
              if(vis[s]==0) do
              vis[s] = 1;
                  add s to d
  for all neighbors 'v' of 's' do
    if((u[i] == s) && (vis[v[i]] == 0))do
            ST[++top] = v[i]
```

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
U	0	1	1	1	1	1	1	2	2	2	2	2	2	3	4	5	5	6	7	8	8	9
V	1	0	2	4	5	7	8	1	3	5	6	8	9	2	1	1	2	2	1	1	2	2

ST	0	1								

	0	1	2	3	4	5	6	7	8	9
vis	1	1	0	0	0	0	0	0	0	0
d	0	1								

DEPTH FIRST SEARCH(DFS)



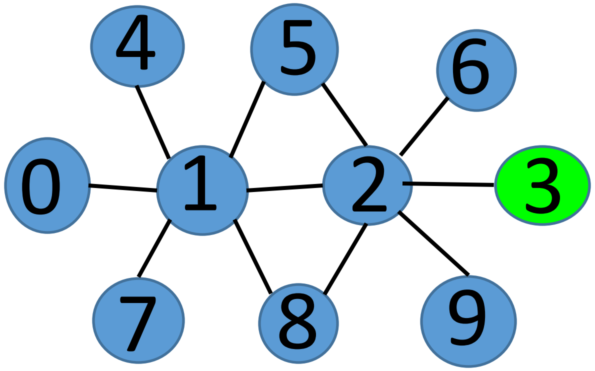
```
REPEAT 'V' TIMES:
  s = 0
        add s to ST
          remove s from ST
          if(vis[s]==0) do
          vis[s] = 1;
              add s to d
  for all neighbors 'v' of 's' do
    if((u[i] == s) && (vis[v[i]] == 0))do
        ST[++top] = v[i]
```

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
U	0	1	1	1	1	1	1	2	2	2	2	2	2	3	4	5	5	6	7	8	8	9
V	1	0	2	4	5	7	8	1	3	5	6	8	9	2	1	1	2	2	1	1	2	2

ST	0	1	2							

	0	1	2	3	4	5	6	7	8	9
vis	1	1	1	0	0	0	0	0	0	0
d	0	1	2							

DEPTH FIRST SEARCH(DFS)



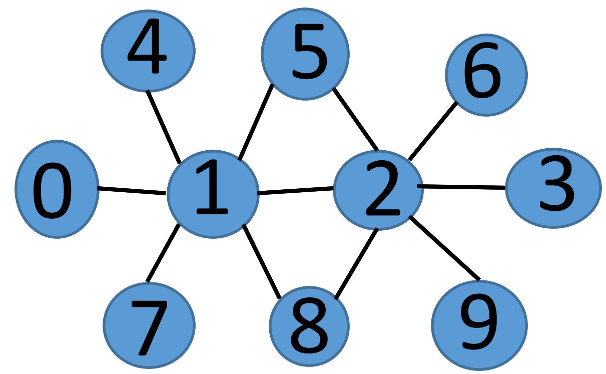

```
REPEAT 'V' TIMES:
  s = 0
            add s to ST
              remove s from ST
              if(vis[s]==0) do
              vis[s] = 1;
                  add s to d
  for all neighbors 'v' of 's' do
    if((u[i] == s) && (vis[v[i]] == 0))do
            ST[++top] = v[i]
```

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
U	0	1	1	1	1	1	1	2	2	2	2	2	2	3	4	5	5	6	7	8	8	9
V	1	0	2	4	5	7	8	1	3	5	6	8	9	2	1	1	2	2	1	1	2	2

ST	0	1	2	3						

	0	1	2	3	4	5	6	7	8	9
vis	1	1	1	1	0	0	0	0	0	0
d	0	1	2	3						

DFS ST OPERATION



0	1	2	3						

0	1	2	5						

0	1	2	6						

0	1	2	8						

0	1	2	9						

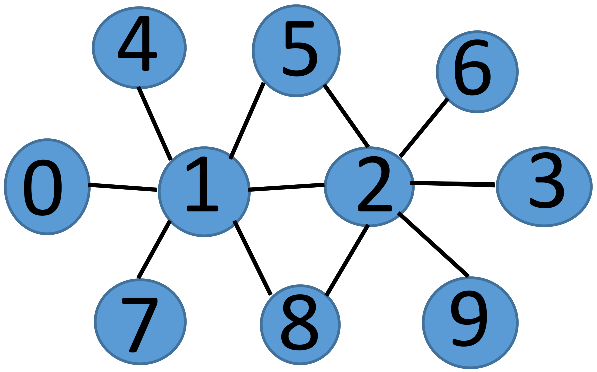
0	1	4							

0	1	7							

0	1								

0									

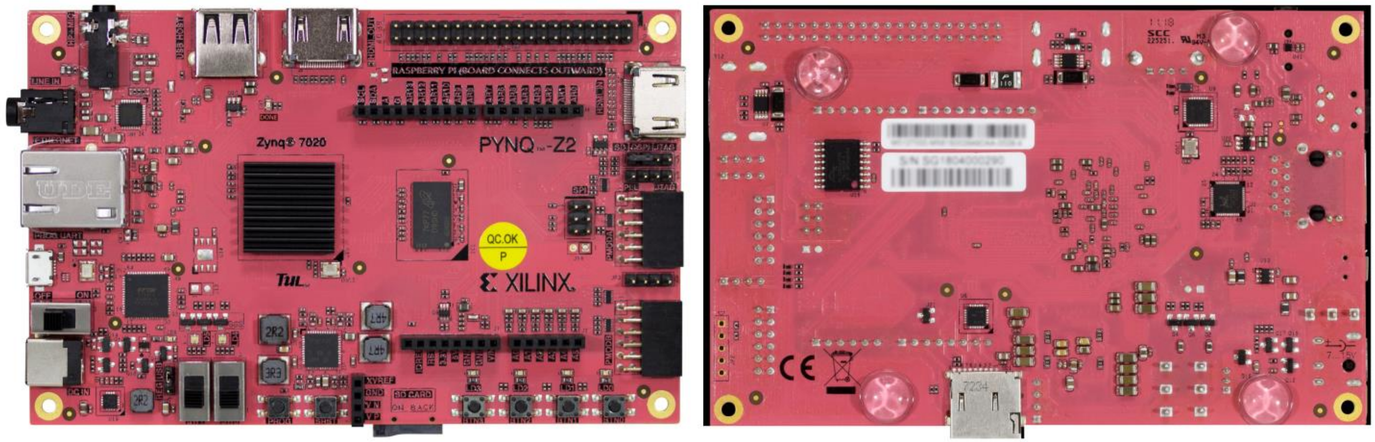
DEPTH FIRST SEARCH(DFS)



	0	1	2	3	4	5	6	7	8	9
d	**0**	**1**	**2**	**3**	**5**	**6**	**8**	**9**	**4**	**7**
vis	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**
ST										

DFS TRAVERSAL

PYNQ Z2 Board



Source: www.tul.com.tw/pynqz2.html

•PYNQ is an open-source project from Xilinx.

•By Employing Python language and libraries, designers can exploit the advantages of programmable logic and microprocessors.

•PYNQ can be used with Zynq, Zynq Ultra Scale+ accelerator boards.

Features of PYNQ-Z2:

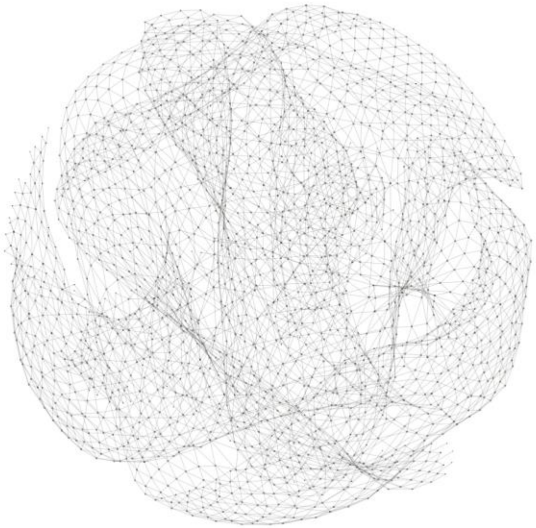
- 650MHz ARM Cortex-A9 dual-core processor
- 13,300 logic slices, each with four 6-input LUTs and 8 flipflops
- 630 KB block RAM
- 220 DSP slices
- One On-chip Xilinx analog-to-digital converter (XADC)

RESULTS

ALGORITHM	LATENCY	CLOCK (ns)	EXECUTION TIME (ms)	GAP BENCHMARK TIMINGS (ms)	Speed Up
BFS	311520	6.319	1.968	9.808	5x
SSSP	457840	7.282	3.33	67.215	20x
PageRank	2648400	8.644	22.89	1.256	0.06x
DFS	335354529	8.373	2808	-	-

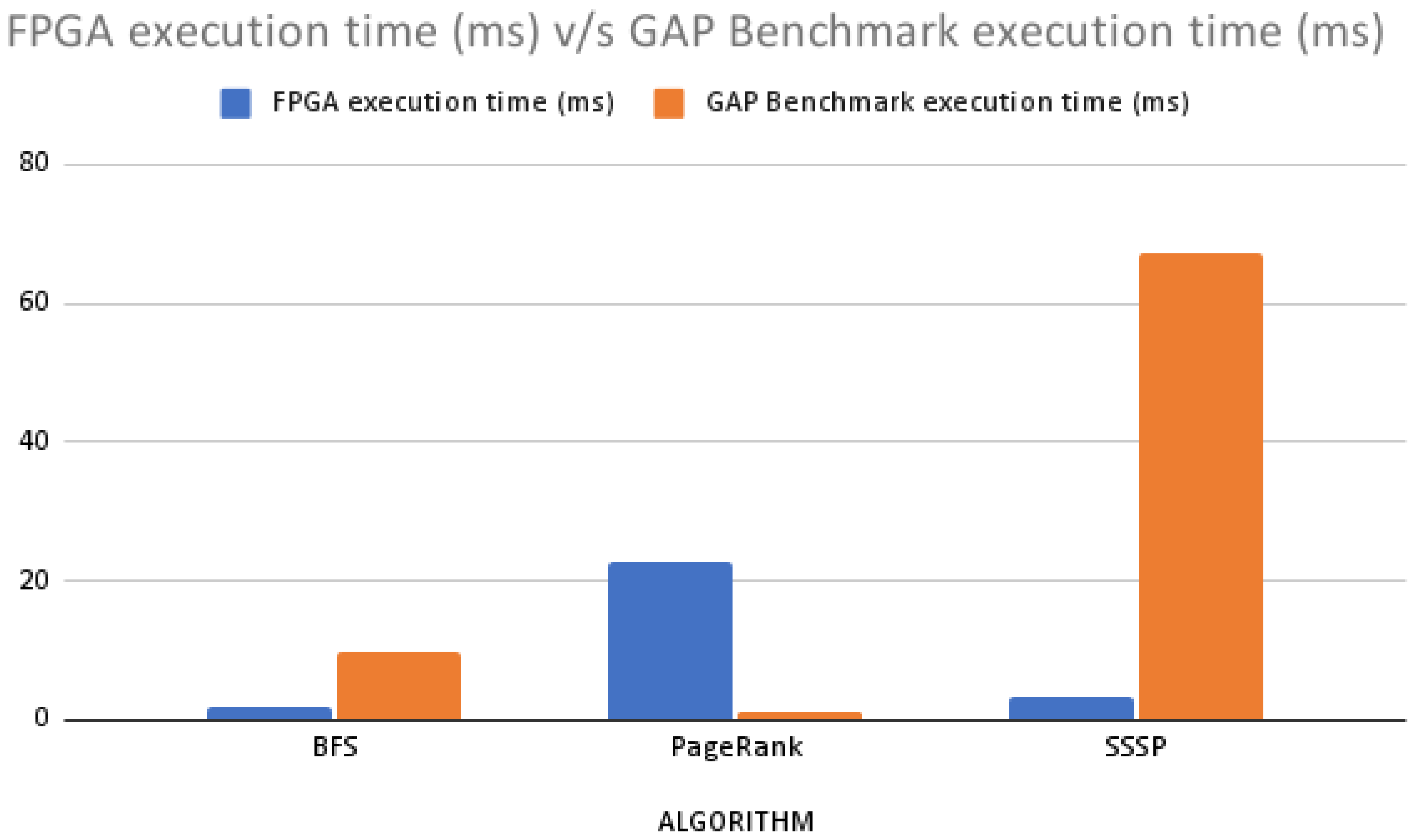
ALGORITHM	Energy consumed on FPGA (J)	Energy consumed on CPU (J)
BFS	0.003338	0.0193
PageRank	0.03847	0.002267
SSSP	0.00565	0.132

Graph Size:
V=4720
E=27444



System Specifications of Benchmark execution:
CPU: AMD A9-9420 @3GHz| OS: KDE neon| RAM: 8GB

RESULTS


FPGA execution time (ms) v/s GAP Benchmark execution time (ms)
FPGA execution time (ms)
GAP Benchmark execution time (ms)
80
60
40
20
0
BFS
PageRank
SSSP
ALGORITHM

WORK DIVISION

The four of us divided the graph algorithms among ourselves such that:

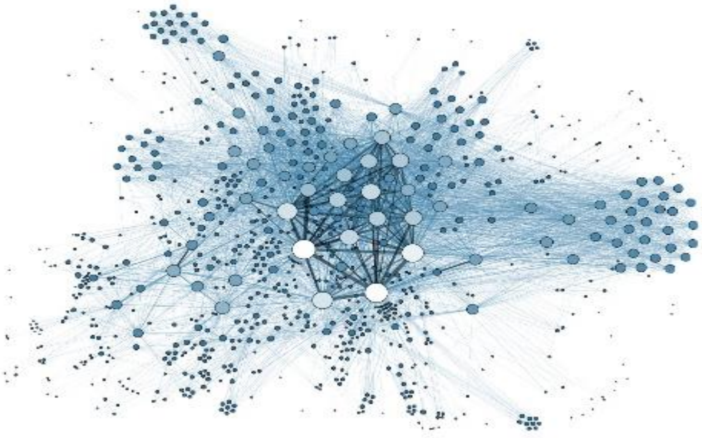
- Avani S was assigned Single Source Shortest Path Algorithm.
- Annette Antony was assigned the PageRank Algorithm.
- Bhimala Subbarayudu was assigned the Breadth-First-Search Algorithm.
- Jeevan R was assigned the Depth-First-Search Algorithm.

Each of us were personally responsible for our individual algorithms in terms of:

- improving performance
- trying different partitioning techniques and
- verifying the functionality of the code.

REFERENCES

- *Graph Processing on FPGAs: Taxonomy, Survey, Challenges by Maciej Besta, Dimitri Standojevic et al.*
- *Evaluation of Graph Analytics Frameworks Using the GAP Benchmark Suite by Ariful Azad, Mohsen Mahmoudi Aznavehy, Scott Beamer et al.*
- *A Study of Partitioning Policies for Graph Analytics on Largescale Distributed Platforms by Gurbinder Gill, Roshan Dathathri, Loc Hoang and Keshav Pingali.*
- *Vitis High-Level Synthesis User Guide by Xilinx*
- *Vivado Design Suite User Guide by Xilinx*
- *Parallel Programming for FPGAs by Ryan Kastner, Janarbek Matai, and Stephen Neuendorffer.*
- *https://github.com/purtroppo/PageRank*
- *http://networkrepository.com/3elt.php#panel-body*



Source: Google Images

Thank you!