

Distributed Breadth-First Search

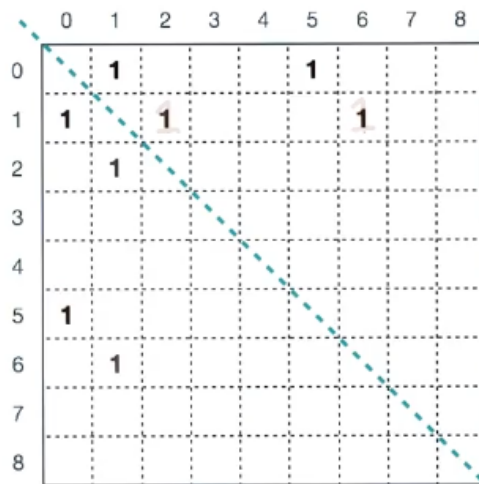
Introduction

1. Approaching the problem of BFS on a distributed memory system from the perspective of linear algebra
 - Represent graph of a matrix
 - Use ideas from distributed matrix multiply

Graphs and Adjacency Matrices

1. Adjacency matrix: Number the nodes, use $N \times N$ matrix to represent edges
 - 0 = not connected, 1 = connected
2. For an undirected graph G with n vertices and m edges, its adjacency matrix is $N \times N$
 - $G = (V, E)$
 - $|V| = n$
 - $|E| = m$
 - Number of non-zero entries = $2 * m$

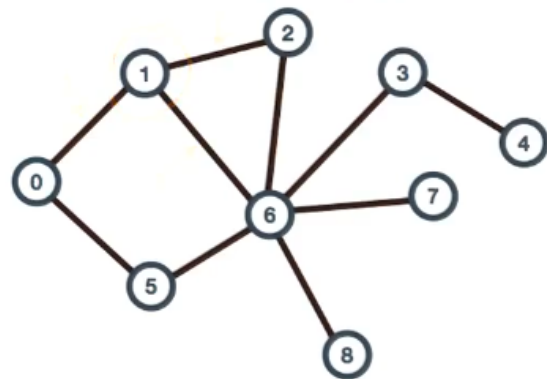
Graphs & Adjacency Matrices



Adjacency matrix, A

Undirected Graph, G

\Rightarrow symmetric: $a_{ij} = a_{ji}$, or $A = A^T$



edge $(i,j) \rightarrow a_{ij} = 1$ ("true")
otherwise, $= 0$ ("false")

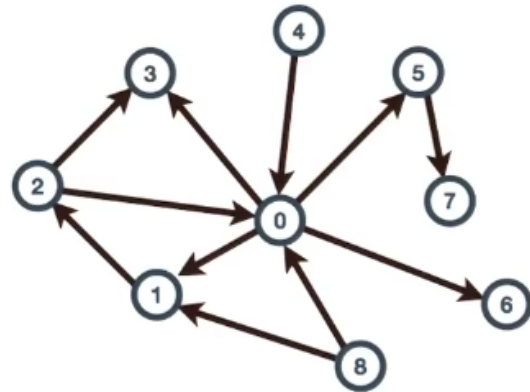
Adjacency Matrix

The Adjacency Matrix of a Directed Graph

1. Number the vertices and fill in the adjacency matrix

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

(blank or "0" = no edge)



Your task: Number the vertices and fill-in the adjacency matrix.

Adjacency Matrix Quiz

Losing Your Direction

- Given the directed graph, how do you compute the boolean adjacency of the undirected graph for matrix B?
 - $\text{or}(B, \text{trans}(B))$

Breadth-First Search Review

- Level-synchronous BFS
 - $G = (V, E)$
 - Source vertex S
 - Distance vector $d[:]$
- At each level l , gather the vertices in the frontier of l
 - Frontier: All adjacent nodes that haven't been visited
 - Mark their distance as the level + 1
 - Repeat
- Algorithmic complexity
 - Running time: $O(m + n)$

Matrix-Based BFS

- Adjacency matrix A contains true where edges exist and false otherwise
- Frontier f is a vector with true in the nodes that are in the current frontier
- Update u is a vector containing the nodes for any vertex j that is in the frontier and adjacency matrix
 - $u[i] \leftarrow \text{OR}(\text{AND}(f[j], A[j][i]))$ for all j
 - $u \leftarrow \text{transpose}(A) * f$
 - Matrix-vector multiply
- Because the adjacency matrix and frontier vector are sparse, we can implement this in a work-optimal way by only looping over vertices that exist

```
// going from update vector to distance
for-all ui = 1 and di = inf do
```

```
di <- l + 1
fnext <- 1
```

Matrix-Based BFS Quiz

1. Mark the entries of u that may need updates, given f .

A	0	1	2	3	4	5	6	7	8	f
0		1				1				
1	1		1				1			1
2		1					1			
3					1		1			
4				1						
5	1						1			
6		1	1	1		1		1	1	1
7							1			
8							1			

u	1	1	1	1		1	1	1	1
---	---	---	---	---	--	---	---	---	---

Update Vector Quiz

1D Distributed BFS

1. Partition the columns across processes (corresponds to a partitioning of the vertices)
 - Also implies a partitioning of u
 - Must replicate the frontier vector across processes
 - Creating the next frontier requires an all-to-all communication
2. Algorithm description
 - Partition columns of A and entries of u
 - Compute $u \leftarrow \text{transpose}(A) * f$
 - Locally update distances
 - Identify local vertices of the next frontier
 - All-to-all, to exchange frontier
3. Algorithmic complexity

- Closed-form solution depends on the graph structure, but due to the all-to-all communication, we expect that the communication cost scales linearly according to number of processors

2D Distributed BFS Quiz

1. How might the $O(P)$ scaling of the 1D algorithm change if we switch to a 2D scheme?
 - If we split the grid across both rows and columns, we might be able to achieve \sqrt{P} scaling as each node will have $\sim\sqrt{P}$ of the matrix
 - We would only need to merge across columns or rows, not both

Conclusion

1. Key idea: Recast BFS in terms of a matrix to make distribution easier
 - Allows us to reuse basic ideas from matrix computations
 - Might be able to frame other graph computations in this way
2. Other graph algorithms:
 - Depth first search
 - All pairs shortest path
 - Triangle counting
 - Computing betweenness centrality