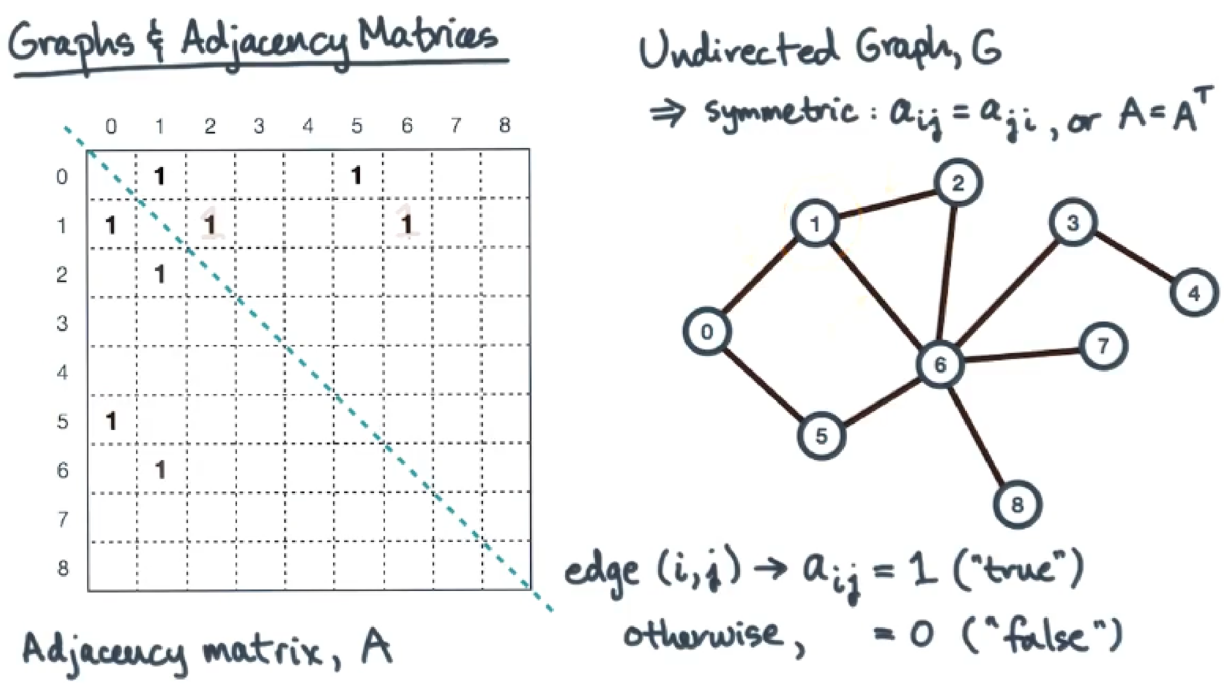
Distributed Breadth-First Search

Introduction

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

Graphs and Adjacency Matrices

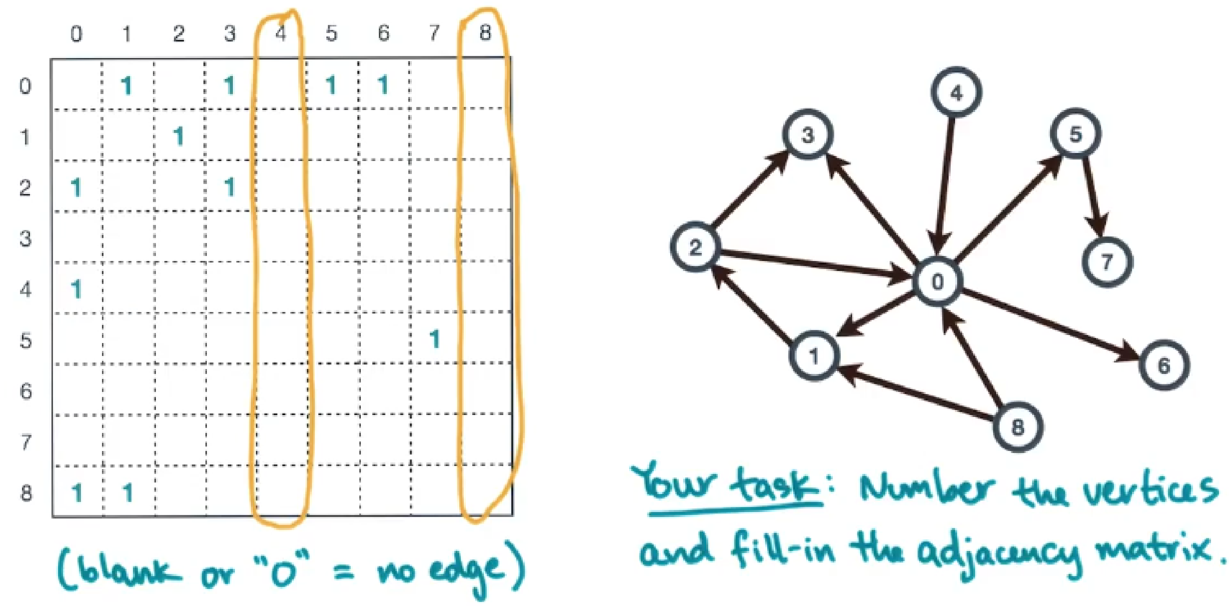
1. Adjacency matrix: Number the nodes, use NxN matrix to represent edges
   - 0 = not connected, 1 = connected
2. For an undirected graph G with n vertices and m edges, its adjacency matrix is NxN
   - G = (V,E)
   - |V| = n
   - |E| = m
   - Number of non-zero entries = 2 * m



Adjacency Matrix

The Adjacency Matrix of a Directed Graph

1. Number the verices and fill in the adjacency matrix



Adjacency Matrix Quiz

Losing Your Direction

1. Given the directed graph, how do you compute the boolean adjacency of the undirected graph for rmatrix B?
   - or(B, trans(B))

Breadth-First Search Review

1. Level-synchronous BFS
   - G = (V,E)
   - Source vertex S
   - Distance vector d[:]
2. 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
3. Algorithmic complexity
   - Running time: O(m + n)

Matrix-Based BFS

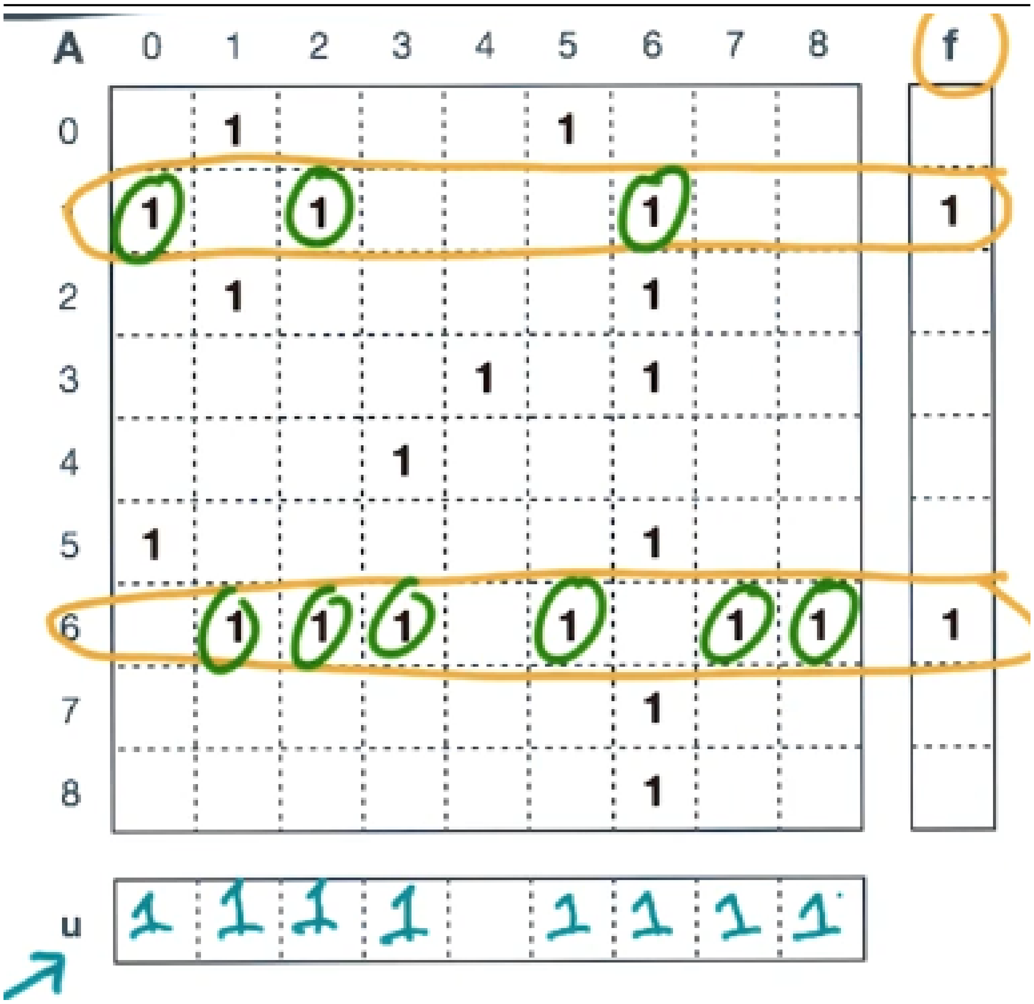
1. Adjacency matrix A contains true where edges exist and false otherwise
2. Frontier f is a vector with true in the nodes that are in the current frontier
3. Update u is a vector containing the nodes for any vertex j that is in the frontier and adjacency matrix
   - u[i] <- OR(AND(f[j], A[j][i])) for all j
   - u <- transpose(A) * f
   - Matrix-vector multiply
4. 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.



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 <- tranpose(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 ~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 betweeness centrality