Shared Memory Parallel BFS

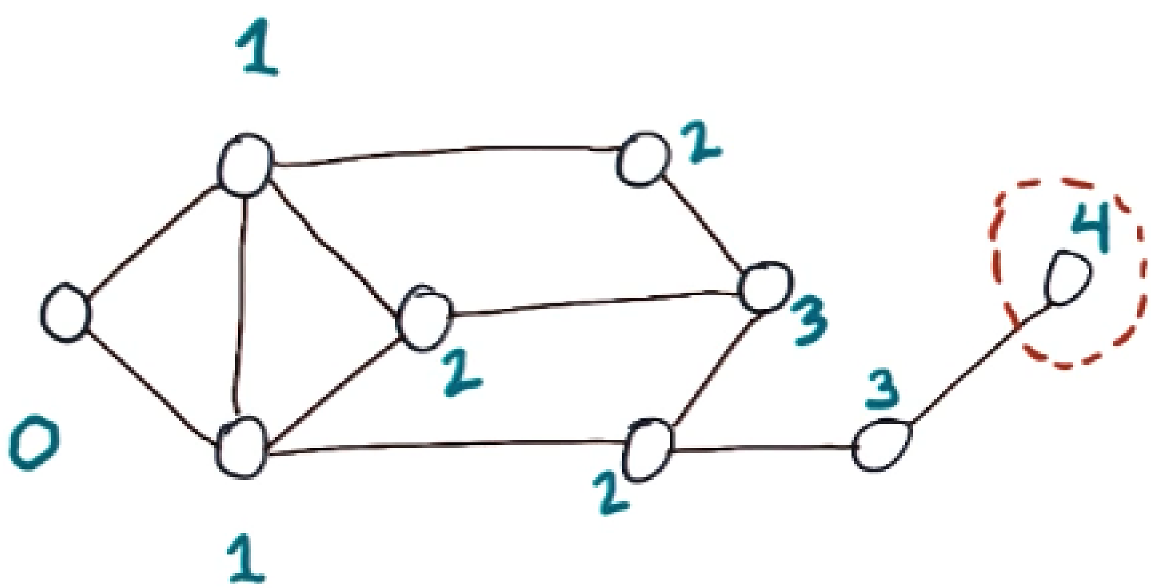
Introduction

1. Graphs are essential in modern computing
   - Social networks, roads, power grids
2. This lesson examines a parallel breadth-first search algorithm for the dynamic multithreading model

BFS 101

1. Breadth-first search: Given a graph and a starting vertex, what is the distance to all other nodes in the graph?
2. Work and span
   - W(n) = O(|V| + |E|)
   - We visit each vertex at most once and each edge at most once

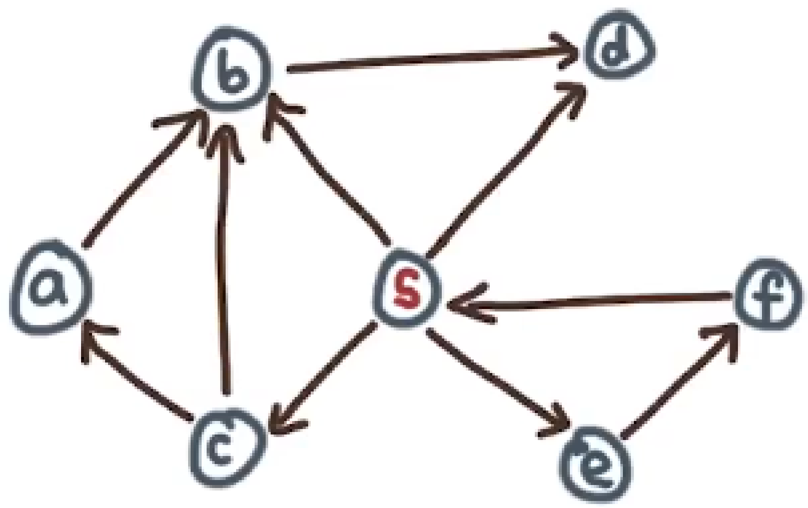
```
BFS(G=(V,E), s)
{
    D[V] <- inf
    D[s] <- 0
    F <- {s} // queue of unvisited vertices
    while F != 0 do
        v <- extract_one(F)
        for(v,w) <- E do
            if D[w] = inf then
                D[w] <- D[v] + 1
                F <- F U {w}
    return D // D[x] = dist(s -> x)
}
```



BFS Example

BFS Example

1. Consider the following directed graph:



BFS Quiz

2. What does F contain right after the third execution of the while loop? There are multiple correct answers due to the ambiguity in "extract_one"
   - c d e

Is BFS Inherently Sequential?

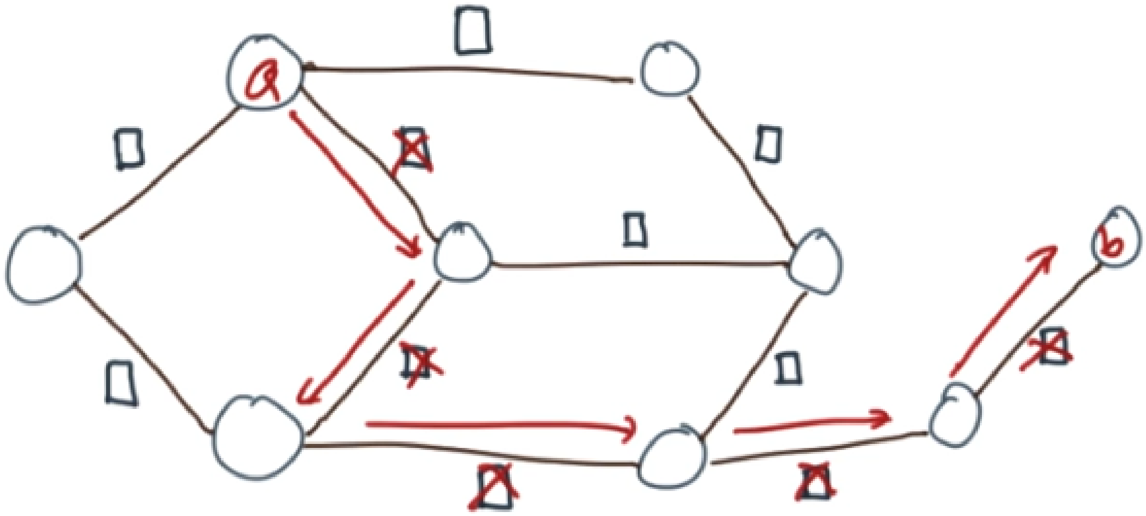
1. BFS Work and Span
   - W(n) = O(|V| + |E|)
   - D(n) = O(|V|)
   - W/D = O(1 + |E|/|V|)
   - In real life, graphs are sparse which means |E| = O(|V|)
     - This means the average available parallelism will be a constant, which is no good

Intuition - Why We Might Do Better

1. The upper bound on the span of BFS shouldn't be the number of vertices, it should be the number of waves
   - Waves are sets of nodes that are reachable from a different set of nodes
   - Diameter: Maximum shortest distance between any pair of vertices in a graph
2. Level-synchronous traversal: Visit nodes level by level
   - Doesn't matter which order we visit nodes within a level
   - This means we can look at nodes simultaneously -> parallelism

BFS Example 2

1. Find the path with 5 edges in the previous example



BFS Quiz

High Level Approach to Parallel BFS

1. Two key ideas:
   - Level synchronous
   - Process an entire level in parallel
2. processLevel takes the graph and current frontier and creates a new frontier, as well as updates the distances
   - Frontiers are level-specific waves of the graph
   - l is a level counter
   - Use a special data structure called a bag for the frontiers

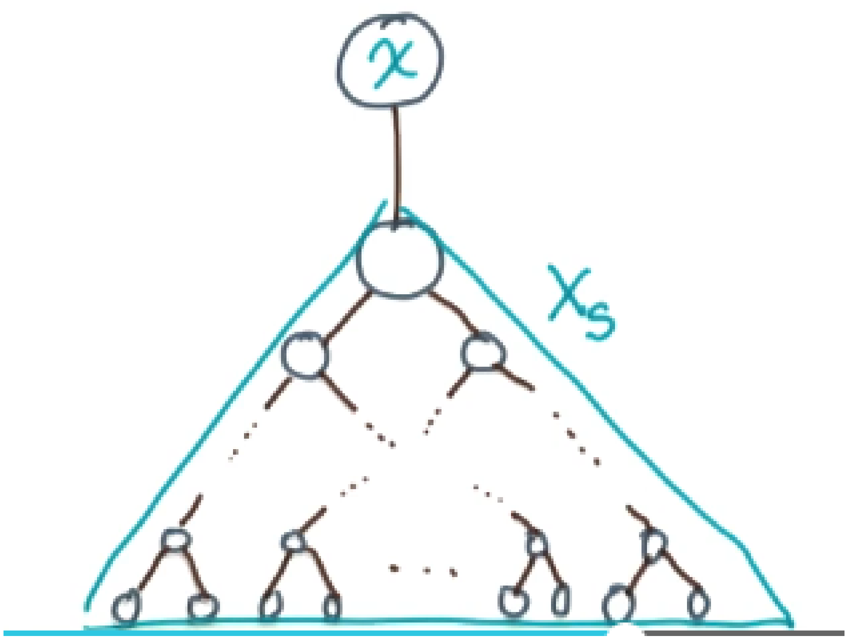
```
parallelBFS(G=(V,E), s)
{
    D[v] <- inf
    D[s] <- 0
    l <- 0
    F0 <- {s}
    while Fl != 0 do
        F(l+1) <- {}
        processLevel(G,Fl,F(l+1),D)
        l <- l + 1
    return D
}
```

Bags: Key Properties

1. Container having the following properties:
   - Unordered collection
   - With repetition
2. Operations
   - Fast traversal
   - Fast and logically associative (A U B == B U A)
     - "Union" and "split" should allow us to divide the data structure into roughly equal pieces and combine them back together

Pennants - Building Blocks for Bags

1. Pennant: Tree with 2ˆk nodes and a unary root have a child that is a complete binary tree
    - Root: x
    - Child: Xs
2. Consider two same-sized pennants x and y with children Xs and Ys respectively
    - To combine, make x the new root, y the child of x, and Xs and Ys the children of y
        - Gives a pennant of size 2ˆ(k+1)
    - We can undo this operation just as easily by repeating the same steps in reverse



Pennant

Pennants

1. What is the output of combining two pennants of different sizes?
    - This is invalid, pennants must be the same size to combine

Combining Pennants Into Bags

1. Represent 23 in binary = 10111
    - You can use this to split a bag into pennants by creating pennants of size 1, 2, 4, and 16 (in the case of 23)
    - Make an array of pointers to the start of each pennant (called a "spine")

Duality Between Bags and Binary Math

1. 23 + 1 = 10111 + 1 = 11000
2. If we want to add an element to a bag, we follow the same approach as base-2 addition
    - Carry pennants that are the wrong size into the next slot of the spine

What is the Cost of Insertion?

1. What is the cost of inserting one element into a bag?

- O(1)
- O(log(n)) (this one)
- O(n)
- O(n * log(n))

2. In the worst case, we must traverse the entire spine to insert
   - The spine requires ceil(log2(n)) elements to represent an integer n, so the complexity is O(log(n))
   - Each insertion requires constant time to perform

Combining Two Bags

1. What is the cost of bag union?
   - O(1)
   - O(log(n)) (this one)
   - O(n)
   - O(n * log(n))
   - O(n^2)
2. The same logic applies as above
   - This is cool because it takes the same amount of time to insert n elements into a bag as it does to insert one element
     - The amortized cost is a constant

Cost of Other Operations

1. What is the cost of bag splitting?
   - O(1)
   - O(log(n)) (this one)
   - O(n)
   - O(n * log(n))

Bag Splitting

1. Performing a right bit shift is roughly equivalent to dividing by two
   - 23 = 10111 » 1 = 1011 = 11
2. Cost = [O(log(n) splits] * [O(1) per split] = O(log(n))

Finishing the Parallel BFS with Bags

1. The following psuedocode implements the processLevel function for the parallel BFS algorithm stated above:

```
processLevel(G, Fl, Fl+1, D)
{
    if |Fl| > phi then
        (A,B) <- bagSplit(Fl)
        spawn processLevel(G, A, Fl+1, D)
        processLevel(G, B, Fl+1, D)
        sync
    else
        for v in Fl do
            parfor (v,w) in E do
                if D[w] = inf then
                    D[w] <- l + 1
                    bagInsert(Fl+1, w)
}
```

2. Even though there's a data race in the base case where multiple threads might write to the same D[w] location, because the algorithm is level-synchronous, each thread is guaranteed to write the same value
3. Work
   - W(n) = O(|V| + |E|)
     - Algorithm is work-optimal
4. Span
   - Number of levels
   - Span of processLevel (depth of recursion, cost of splitting, base case)
     - Depth of recursion is log(n)
     - Cost of splitting is log(n)
     - Base case is bounded by a constant because we have a constant cutoff
   - Diameter
   - D(n) = O(d * r * log(|V| + |E|)

Conclusion

1. The parallel BFS algorithm is based on binomial heaps