Modeling Bellman-Ford

Grace Jiang, Karen Xiao

Motivation

- Shortest path algorithms are useful
- Graph traversal modeling in alloy
- Ad hoc networks using Bellman-Ford
- (Also CS343 :))

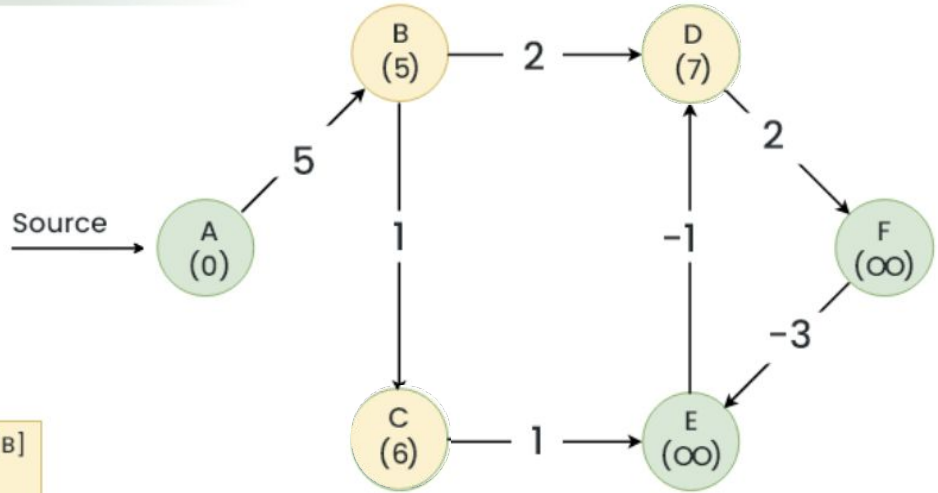
Normal BF Pseudocode

```
BellmanFord(list vertices, list edges, vertex source):
    for each vertex v in vertices:
        distance[v] := inf

    distance[source] := 0

    repeat |V|−1 times:
       for each edge (u, v) with weight w in edges:
           if distance[u] + w < distance[v] then
               distance[v] := distance[u] + w
```

1st Relaxation Of Edges



Dist [A] + 5 <Dist[B]
0+5<(∞)
Dist[B] = 5

Distance Array

A	B	C	D	E	F
0	5	∞	∞	∞	∞

ALLOY!

How we modeled the basic version

- sig Node
  - neighbors (set Node)
  - weights Node->Finite
- one sig Source extends Node
  - distances (Node->Distance)
- pred relax
- abstract sig Distance (either Finite or Infinite)
  - value (Int)
- fun compareDistances

DEMO!

normal!

Trial & Error in the ad hoc model

- Main differences
  - No longer *single* source
  - Every node broadcasts its own distance table whenever updated
  - Start the algorithm by allowing either sending tables or just doing nothing
- Inadvertently modeled “deadlock”
- So chose to simulate the distributed algorithm “iteratively” - tradeoff

DEMO!

distributed!

Workflow & Trade-offs

- Normal
  - No negative cycle detection
- Distributed
  - Modeled iteratively to *simulate* a basic distributed version
  - Assumed that all edges have weight 1

Possible future work

- Mutex/distributed, similar to ring election, raise flags, etc.
- Dynamic topology

Thank you!

Questions?