236610
Distributed Graph Algorithms

Spring 2018
Keren Censor-Hillel
A BFS tree

• Breadth-First Search Tree
  – Input: a root r
  – Output: a tree of shortest paths to r

• Distributed setting:
  – Input to node v: indication whether v is r
  – Output: \( d(v,r) \) and parent in the BFS tree
    • No knowledge of the entire tree is required
A BFS algorithm

Variables for node v:
state ∈ \{activated, deactivated\}, initially deactivated, except for v = r
parent, initially v
dist, initially ∞, except for v = r, initially 0

1 for i=1, . . . do
  2       if state = activated then
  3           send i to all neighbors
  4       state ← deactivated
  5       else
  6           if receive message for first time (from node w) then
  7               parent ← w
  8               dist ← i
  9               state ← activated
Lower bound

• **Claim 3**: Every synchronous BFS algorithm requires $\Omega(D)$ rounds.

• Intuitively, this is because a node within distance $D$ from the root $r$ can find it out only after $D$ rounds.
The Local Claim

- **The Local Claim**: After i rounds, a node cannot know anything about nodes that are within distance $j > i$ from it.

- **Proof**: We prove this by induction.

- **The base case**: For $i = 0$, indeed before the algorithm starts every node knows only about itself.

- **Induction hypothesis**: In $i - 1$ rounds every node does not know anything about nodes that are farther than $i - 1$ hops away.

- **Induction step**: Therefore, in the i-th round, a node can only hear from its neighbors, which only know about their $i - 1$ neighborhood, and therefore the node only know about its $i$ neighborhood.
Lower bound

• **Claim 3**: Every synchronous BFS algorithm requires $\Omega(D)$ rounds.

• **Proof**: We use an *indistinguishability* argument:

  • Let $v$ be a node within distance $D$ from $r$. Consider a graph $G'$, which has one other node $r'$ that is connected to $r$ and is the root of a BFS we want to build in $G'$. The distance of every node $u$ from $r'$ in $G'$ is larger by one compared with its distance to $r$ in $G$.

  • By **The Local Claim**, in less than $D$ rounds, $v$ cannot distinguish between $G$ and $G'$, and so must output the same distance. But for one of these graphs this is incorrect.
Lower bound

• **Illustration**: Every synchronous BFS algorithm requires $\Omega(D)$ rounds.

• **Note**: It is also possible to find $G$ and $G'$ with the same number of nodes.
Asynchronous model

• No timing guarantees on delivery of messages

• **Complexity measures:**
  – Number of messages
  – Time: worst case number of time units assuming each message takes at most single unit
BFS algorithm

• What happens if we run the previous one?

• **Illustration:** It is possible that the first message that a node \( v \) receives is from a neighbor \( u \) whose distance from \( r \) is larger
Sequential algorithms

• **Bellman-Ford:**
  – Initially all distances are set to $\infty$
  – For $n-1$ iterations, we travel all $m$ edges and update the distances
  – **Time:** $O(mn)$

• $m$ is the number of edges in the graph
Sequential algorithms

• **Dijkstra**:  
  – Initially all distances are set to $\infty$  
  – For $n$ iterations, we choose a node that we didn’t handle yet which has smallest distance, and update all distances of its neighbors  
  – **Time**: $O(m+n\log n)$  
    (or $O(n^2)$ with a naïve implementation)  
  – **Note**: does not handle negative weights
Asynchronous BFS

• Both of the above algorithms can be implemented in an asynchronous distributed setting

• With different trade-offs between time complexity and message complexity
Update-based asynch. BFS

Variables for node $v$:

- **state** $\in \{\text{activated, deactivated}\}$, initially deactivated, except for $v = r$
- **parent**, initially $v$
- **dist**, initially $\infty$, except for $v = r$, initially 0

1. **if** $v = r$ **then**
2. send **dist** to neighbors
3. **state** $\leftarrow$ deactivated
4. **for** distance $m$ received from $w$ **do**
5. **if** $m + 1 < \text{dist}$ **then**
6. **state** $\leftarrow$ activated
7. **parent** $\leftarrow w$
8. **dist** $\leftarrow m + 1$
9. send **dist** to neighbors
10. **state** $\leftarrow$ deactivated
Update-based async. BFS

• Correctness (informal): It may be that a node becomes activated more than once. To see this, consider a graph where $v$ is connected to $r$ by disjoint paths of length 1, 2, 3 etc.

• The claim is that eventually, no node is activated and no message is in transit, and at this time, for every node $v$ the variable $\text{dist}(v)$ holds the distance from $r$, and $\text{parent}(v)$ points to a parent in the BFS tree.
Update-based asynch. BFS

• **Message Complexity (informal):**
  It could be that every message received causes the node to send an updated distance message on every edge.
  This can happen at most $n$ times, resulting in $O(nm)$ messages.
  – As in the analysis of Bellman-Ford
Update-based asynch. BFS

• **Time Complexity (informal):**

  The time is $O(D)$, where $D$ is the diameter of the graph. This is because after this number of time units, all shortest paths have sent messages.
Root-controlled asynch. BFS

Variables for node v:

- **state** ∈ \{activated, deactivated, done\}, initially deactivated, except for v=r, initially activated
- **parent**, initially v
- **dist**, initially ∞, except for v = r, initially 0
- **next**, initially false

1. **if** v=r **then**
   2. send dist to neighbors
   3. **state** ← done

4. **for** message m /∈ \{ack, continue\} received from w **do**
   5. **if** state = deactivated **then**
      6. **state** ← activated
      7. **parent** ← w
      8. **dist** ← m + 1
      9. send ack to w
Root-controlled asynch. BFS

Variables for node v:

- **state** ∈ \{activated, deactivated, done\}, initially deactivated, except for v = r, initially activated
- **parent**, initially v
- **dist**, initially ∞, except for v = r, initially 0
- **next**, initially false

10 for message m = ack received from w do
11 If received ack from all neighbors after last time next set to false then
12 next ← true
13 if v ≠ r then
14 send ack to parent
15 else (v=r)
16 send continue to all neighbors
17 next ← false
Root-controlled asynch. BFS

Variables for node v:

- **state** ∈ \{activated, deactivated, done\},
  - initially deactivated, except for v=r, initially activated
- **parent**, initially v
- **dist**, initially \(\infty\), except for v = r, initially 0
- **next**, initially false

18 **for** message m = continue received from w **do**
19    **if** state = done **then**
20    send continue to neighbors
21    **if** state = activated **then**
22    send dist to neighbors
23    **state** ← done
Root-controlled asynch. BFS

• Correctness (informal):
A node becomes activated exactly once. The root \( r \) coordinates when to send the messages to the next level in the tree, so when a node receives a message containing distance information for the first time, this is its true parent and distance in the BFS tree.
Root-controlled asynch. BFS

• **Message Complexity (informal):**
A distance message is sent only once over each edge. An `ack` or a `continue` message is forwarded at most $D$ times by each node. An `ack` is sent only to the parent.

We can modify the variables of the algorithm to know which neighbors are children in the tree, and send continue messages only on tree edges, so the total number of messages is $O(m + nD)$. 
Root-controlled async. BFS

- **Time Complexity (informal):**
The time is $O(D^2)$, because every new level $i$ requires an additional $O(i)$ time, so the total is $O(\Sigma_{i=1}^{D} i) = O(D^2)$. 

Keren Censor-Hillel, Spring 2018
## Asynchronous BFS

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Message complexity</th>
<th>Time complexity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Update-based</td>
<td>$O(nm)$</td>
<td>$O(D)$</td>
</tr>
<tr>
<td>Root-controlled</td>
<td>$O(m+nD)$</td>
<td>$O(D^2)$</td>
</tr>
</tbody>
</table>

Keren Censor-Hillel, Spring 2018
An MST

• Minimum Spanning Tree
  – Input: a weight function $w:E \rightarrow \mathbb{R}$
  – Output: a spanning tree $T$ of $G$, such that for every spanning tree $S$ of $G$, $w(T) \leq w(S)$
    • $w(S) = \sum_{e \in S} w(e)$

• Distributed setting:
  – Input to node $v$: the weight $w(e)$ of each $e$ touching $v$
  – Output: which edges touching $v$ are in $T$
    • No knowledge of the entire tree is required
Model

• **Synchronous** system. **Complexity**: number of rounds
• In each round:

  - **LOCAL**

  node v can send messages to every node in N(v)

  - **CONGEST**

  node v can send messages of **O(logn) bits** to every node in N(v)

    – Motivation: sending an ID in a single message
MST in CONGEST

• In general, we can replace $O(\log n)$ for the number of bits by any parameter $B$.

• We need to make sure that edge weights also fit in a message, so we assume $w:E \rightarrow W$, where $W=\{0,1,...,\text{poly}(n)\}$.

• **Assumption**: $w$ is 1:1 (there is a unique MST)
Sequential MST

- **Kruskal**: removing heavy edges from cycles by going over edges in increasing order of weights, adding an edge to T if it does not create a cycle *(red rule)*

- **Prim**: maintain a connected component by adding the lightest edges leaving it *(blue rule)*

- **Burovka**: initially each node is a connected component. Go over connected components in arbitrary order and added lightest edge leaving the component *(blue rule)*
MST in CONGEST

• We can simulate each of the above algorithms
  – In a naïve manner: going over all $m$ edges in each iteration. **Time: $O(nm)$** rounds

• In $O(m)$ rounds we can learn the graph
BFS-based MST

• **High-level description:**
  Simulate the **Kruskal** algorithm by one node r.
  – Say, minimal ID

• The nodes send edges to r over a BFS tree.

• Instead of learning about all the edges, the nodes send edges of increasing weights, and only those that do not create a cycle.
BFS-based MST

Variables for node $v$:

- $E_v$, initially $\{v,u\} \subseteq E$
- $S_v$, initially empty

1. compute an unweighted BFS tree $T$ from $r$
2. for $i=1,\ldots,n+D-2$ rounds
3. $e = \arg\min_{e' \in E \setminus S_v} \{w(e')\}$
4. send $(e, w(e))$ to parent in $T$
5. $S_v \leftarrow S_v \cup \{e\}$
6. for each received $(e, w(e))$
7. $E_v \leftarrow E_v \cup \{e\}$
8. for each cycle $C$ in $E_v$
9. $e = \arg\max_{e' \in C} \{w(e')\}$
10. $E_v \leftarrow E_v \setminus \{e\}$
11. $r$ downcasts $E_r$ over $T$
12. return $\{v,u\} \subseteq E_r$
Correctness

• **Claim 1**: If the algorithm runs for enough iterations then $E_r$ is the MST.

• **Proof**: Edges that do not reach $r$ are heaviest in a cycle and therefore correctness follows from correctness of Kruskal’s algorithm.
Complexity

• $T_v$: the subtree of $v$ in $T$
• $d_v$: the depth of $T_v$

• $E_{T_v} = \bigcup_{w \in T_v} \{ e \in E_w \}$

• $F_v =$ lightest maximal forest of $E_{T_v}$
Complexity

• **Claim 2**: For every \( v \), for every \( k=0,1,...,|F_v| \), after \( dv+k-1 \) iterations of line 2, the \( k \) lightest edges of \( F_v \) are in \( E_v \), and are sent to the parent of \( v \) in \( T \) by the end of iteration \( dv+k \).

• **Claim 3**: The algorithm returns an MST in \( O(n) \) rounds.

• **Proof**: Correctness follows from **Claim 1**. By **Claim 2**, after \( dr+(n-1)-1=O(n) \) rounds, \( E_r \) is the MST. An additional number of \( O(D) \) rounds are needed for downcasting \( E_r \).
Complexity

• **Proof of Claim 2**: by a double-induction over $k$ and $dv$.

• **Base case**: immediate for $k=0$ and for $dv=0$.

• **Induction hypothesis**: Assume this holds for all $dw<dv$ and for $dv$ up to some $k$. 
Complexity

• **Induction step**: We prove for $dv$ and $k+1$. Assume, towards a contradiction that the $k+1$ lightest edge $e$ in $Fv$ is not known to $v$ by iteration $dv+(k+1)$.

• By the induction hypothesis, $e$ is known to some child $w$ of $v$, and is sent to $v$ by iteration $dw+(k+1) \leq dv+k = dv+(k+1)-1$. 
Complexity

• We need to show that $e$ is sent to the parent $u$ of $v$ by round $dv+(k+1)$. Assume otherwise, then a different edge $e'$ is sent by $v$ to $u$, which is not one of the $k$ lightest edges in $F_v$.

• Since edges are sent by increasing weights, then $e'$ is not a part of $F_v$. Hence, there is a cycle in which all other edges are in $F_v$ and $e'$ is the heaviest. But this contradicts sending $e'$. 
BFS-based MST

Variables for node $v$:
$E_v$, initially $\{\{v,u\}\in E\}$
$S_v$, initially empty

1. compute an unweighted BFS tree $T$ from $r$
2. for $i=1,\ldots,n+D-2$ rounds
3. $e = \arg\min_{e' \in E \setminus S_v} \{w(e')\}$
4. send $(e, w(e))$ to parent in $T$
5. $S_v \leftarrow S_v \cup \{e\}$
6. for each received $(e, w(e))$
7. $E_v \leftarrow E_v \cup \{e\}$
8. for each cycle $C$ in $E_v$
9. $e = \arg\max_{e' \in C} \{w(e')\}$
10. $E_v \leftarrow E_v \setminus \{e\}$
11. $r$ downcasts $E_r$ over $T$
12. return $\{\{v,u\}\in E_r\}$
BFS-based MST

• **Summary:**
  - Simulates *Kruskal’s* algorithm
  - $O(n)$ rounds
The GHS Algorithm for MST

• **Gallager-Humblet-Spira**

• **Combinatorial claim**: For every subset $S$ of $V$, the lightest edge from $S$ to $V\setminus S$ belongs to the MST.

• **High-level description**: Simulate the **Burovka/Prim** algorithms by maintaining a connected component that grows by adding the lightest edge leaving it.
The GHS Template

Variables:
\( T \), initially empty

1 repeat
2 \( F \leftarrow \) set of connected components of \( T \)
3 \textbf{For} each \( C \in F \)
4 add to \( T \) the lightest edge leaving \( C \)
5 until \( T \) is a spanning subgraph
   (no outgoing edges, single component)
6 return \( T \)
Correctness

• Still without implementation details

• A phase: one iteration of the loop

• Claim 1: The returned set of edges $T$ is an MST.

• Proof: An edge $e$ is added to $T$ only if it is the lightest leaving $C$. By the combinatorial claim, $e$ belongs to the MST. We return only when we have a spanning subgraph.
Complexity

• **Claim 2**: The algorithm completes after $O(\log n)$ phases.

• **Proof**: We prove by induction, that at the end of phase $i$, the size of each connected component is at least $2^i$.

• **Base case**: For $i=0$, at the end of the initialization, each singleton is a connected component of size 1.
Complexity

• **Induction hypothesis**: At the end of phase $i-1$, each connected component is of size at least $2^{i-1}$.

• **Induction step**: In phase $i$, each connected component adds an edge to another connected component. By the induction hypothesis, the size of each new component is at least $2^{i-1} + 2^{i-1} = 2^i$.

• Hence, after $O(\log n)$ iterations there is a single connected component.
Implementation

• What does it mean for a connected component to choose an edge?

• For each component $C$, we assign a root node $r_C$, which is the node with the smallest ID.
• The ID of $r_C$ is the ID of the component $C$. 
Implementation

• In each phase, every node $v$ sends to all of its neighbors a triplet $(u, w(e), C')$
  
  – $e = \{v, u\}$ is the lightest edge that leaves $v$ to a different component
  
  – $w(e)$ is the weight of $e$
  
  – $C'$ is the ID of the component of $u$
Implementation

• Each node forwards the triplet of the lightest edge it received so far, towards $r_c$ using the edges of $T$.

• The root $r_c$ picks the triplet of the lightest edge it received and sends it back to all nodes of the component.
Implementation

• The chosen node $v$ sends a message to its neighbor $u$, which forwards it through $C'$. The node with the minimal ID among all roots of the merged components becomes the new root.

— There are missing details in the above description
Theorem: The GHS algorithm computes an MST within \( O(n \log n) \) rounds.

Proof:

• Correctness follows from Claim 1.
  — with additional details

• Claim 2 gives that there are \( O(\log n) \) phases.

• The implementation completes a phase in \( O(n) \) rounds, since this is the maximum diameter of each component.
GHS – notes

• This was not a formal pseudocode
• The complexity can be reduced to $O(n)$
• The GHS algorithm can be implemented in an asynchronous setting within $O(m \log n)$ messages
  – This can be improved to $O(m + n \log n)$