236358
Distributed Graph Algorithms

Spring 2017
Class by 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
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 $\in$ {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 $m+1 < dist$ 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 asynch. 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 $v$:

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

1 if $v=r$ then
2 send dist to neighbors
3 $\text{state} \leftarrow \text{done}$

4 for message $m \in \{\text{ack, continue}\}$ received from $w$ do
5 if $\text{state} = \text{deactivated}$ then
6 $\text{state} \leftarrow \text{activated}$
7 $\text{parent} \leftarrow w$
8 $\text{dist} \leftarrow m + 1$
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 \(\infty\), except for v = r, initially 0
- **next**, initially false

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

Variables for node v:
\(\text{state} \in \{\text{activated, deactivated, done}\},\)
\(\text{initially deactivated, except for } v=r, \text{ initially activated}\)
\(\text{parent, initially } v\)
\(\text{dist, initially } \infty, \text{ except for } v = r, \text{ initially } 0\)
\(\text{next, initially false}\)

\begin{verbatim}
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
\end{verbatim}
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 asynch. 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 $O(\sum_{i=1}^{D} i) = O(D^2)$. 
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>
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 $O(\log n)$ 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 $W = \{0, 1, \ldots, \text{poly}(n)\}$.

• **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. $E_v \leftarrow E_v \cup \{e\}$
7. for each cycle $C$ in $E_v$
8. $e = \arg\max_{e' \in C} \{w(e')\}$
9. $E_v \leftarrow E_v \setminus \{e\}$
10. $r$ downcasts $E_r$ over $T$
11. return $\{\{v,u\}\subseteq E_r\}$
Correctness

• **Claim 1**: If the algorithm runs for enough iterations then \( Er \) 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 $T_v$

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

- $F_v = \text{lightest maximal forest of } E_{Tv}$
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:

$E_U$, initially $\{v, u\} \in E$

$S_u$, initially empty

1. compute an unweighted BFS tree $T$ from $r$
2. for $i = 1, \ldots, n+D-2$ rounds
3. \hspace{1em} $e = \arg\min_{e' \in E \setminus S_u} \{w(e')\}$
4. \hspace{1em} send $(e, w(e))$ to parent in $T$
5. \hspace{2em} $S_u \leftarrow S_u \cup \{e\}$
6. \hspace{2em} for each received $(e, w(e))$
7. \hspace{3em} $E_u \leftarrow E_u \cup \{e\}$
8. \hspace{2em} for each cycle $C$ in $E_u$
9. \hspace{3em} $e = \arg\max_{e' \in C} \{w(e')\}$
10. \hspace{3em} $E_u \leftarrow E_u \setminus \{e\}$
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