Algorithms at Scale
(Week 12)

k-Machine Models
<table>
<thead>
<tr>
<th>Summary</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Today: k-Machine</strong></td>
</tr>
<tr>
<td>k-Machine Model</td>
</tr>
<tr>
<td>• Cluster computing</td>
</tr>
<tr>
<td>Some simple examples</td>
</tr>
<tr>
<td>• Luby’s</td>
</tr>
<tr>
<td>• Bellman-Ford</td>
</tr>
<tr>
<td>Minimum Spanning Tree</td>
</tr>
<tr>
<td>• Basic algorithm</td>
</tr>
<tr>
<td>• Fully distributed algorithm</td>
</tr>
<tr>
<td>• Lower bound</td>
</tr>
<tr>
<td>Map-Reduce Model</td>
</tr>
<tr>
<td>• Cluster computing</td>
</tr>
<tr>
<td>Some simple examples</td>
</tr>
<tr>
<td>• Word count</td>
</tr>
<tr>
<td>• Join</td>
</tr>
<tr>
<td>Algorithms</td>
</tr>
<tr>
<td>• Bellman-Ford</td>
</tr>
<tr>
<td>• PageRank</td>
</tr>
</tbody>
</table>
Announcements / Reminders

Today:

MiniProject presentation due today.

Next week:

Six groups (TBA) to present in class

Nov. 17:

Final report due
A few comments...

On writing a report:

1. Begin with an overview / introduction.

   What is this report about?

   What will I learn if I read it?

   What are the “results” or conclusions?

   (Maybe: why is this topic important?)
A few comments...

On writing a report:

2. Explain so that everyone can understand.

Anyone in this class should understand the algorithm.

Goal: more clear than a Wikipedia page!
A few comments...

On writing a report:


From your description, can I implement the algorithm?

Did you include enough detail that I know how every step works?
On writing a report:


   From your description, do I understand WHY the algorithm works?

   Which steps are important?

   Which steps are just optimization?

   Why do we do it this way?
On writing a report:

5. Draw pictures. Use examples

Illustrate how the algorithm works.

Draw a picture of the data structure.

Go through a step-by-step example.
A few comments...

On writing a report:

6. Cite properly

Did you invent the algorithm? If not, cite.

Did you invent this proof? If not, cite.

Do not simply copy proofs directly from existing sources. (Do cite sources you used.) Your goal is to give a better proof.

Don’t plagiarize.
A few comments...

On dimensionality reduction:

1. Think about the trade-offs.

   Cost of doing the dimensionality reduction vs. benefit of lower dimensions.

2. For non-linear methods especially, think about cost.

   Is the method reusable (with a high one-time cost) or is each use expensive?

3. The final dimension is an important parameter.

   Many techniques do better then the theory would predict on real-world data.
On discrete elements with windows:

1. It is interesting to adapt FM and HLL to generic windowed techniques.
   
   For example, using smoothed histogram techniques.

2. If you look more closely, there is a simpler direct technique.
   
   You don’t need histograms.

3. Interesting variants?
   
   Queries on different window lengths? Other types of sketches?
A few comments...

On write-optimized data structure:

1. LSM is used a lot in practice. COLA is not.

Why? Is that a correct evaluation?

2. Are there hybrid LSM/COLA algorithms that might be good?

Imagine using the COLA for $x$ levels and the LSM for levels $> x$.

3. Can you speed up the COLA with LSM-optimizations?

For example, a LSM often uses a Bloom filter to speed up queries. A COLA?
Summary

Today: k-Machine

k-Machine Model
- Cluster computing

Some simple examples
- Luby’s
- Bellman-Ford

Minimum Spanning Tree
- Basic algorithm
- Fully distributed algorithm
- Lower bound

Last Week: Map-Reduce

Map-Reduce Model
- Cluster computing

Some simple examples
- Word count
- Join

Algorithms
- Bellman-Ford
- PageRank
Fork-Join algorithms

Assumptions:
- Tightly synchronized
- Shared memory

Advantages:
- Simple algorithm design
- Focus on parallelism (computational)
- Easy analysis: work and span is enough!
- Minimizes race conditions, deadlocks, etc.
High Performance Clusters

Assumptions:
- Loosely synchronized
- No shared memory
- Data exchanged over fast interconnect

Issues:
- Communication cost?
- Coordination among cores?
- Fine-grained parallelism?

Fork/Join is not a good model for clusters.
Map-Reduce Model

Basic round:

1. **Map**: process each (key, value) pair
2. **Shuffle**: group items by key
3. **Reduce**: process items with same key together

Key goals:

**Target**: high-performance clusters.

**Focus**: data (not computation)
Map-Reduce

Advantages:

– Based on real working systems (e.g., Hadoop)
– Focus on data processing
– Simple programming model: Map and Reduce
– Scales well in practice

Disadvantages:

– Bandwidth issues are invisible
– Expensive sorting operation is hidden
– Hard to coordinate data movement
– Stateless model is tricky
Map-Reduce

Advantages:
- Based on real working systems (e.g., Hadoop)
- Focus on data processing
- Simple programming model: Map and Reduce
- Scales well in practice

Disadvantages:
- Bandwidth issues are invisible
- Expensive sorting operation is hidden
- Hard to coordinate data movement
- Stateless model is tricky

Today’s Goal:
- A more abstract model.
- Stateful.
- Easier to design algorithms.
- Easier to get a realistic sense of algorithm performance.
Basic assumptions:

- \( k \) servers: system is a collection of cores/CPUs/etc.
- all-to-all communication: communicate via messages
- bandwidth limit \( B \): limited data transfer
k-Machine Model

Basic assumptions:

- **k servers**: system is a collection of cores/CPUs/etc.
- **all-to-all communication**: communicate via messages
- **bandwidth limit B**: limited data transfer

![Diagram]

- Machine can send kB bits total
- B bits/round
k-Machine Model

Basic assumptions:

- **k servers**: system is a collection of cores/CPUs/etc.
- **all-to-all communication**: communicate via messages
- **bandwidth limit B**: limited data transfer

![Diagram showing machine can receive kB bits total and communicate with each other at B bits/round.]
Basic assumptions:

- **k servers**: system is a collection of cores/CPUs/etc.
- **all-to-all communication**: communicate via messages
- **bandwidth limit B**: limited data transfer

Total “switch” bandwidth:
\[ B_k(k-1) \approx k^2B \]
k-Machine Model

Basic assumptions:

– **k servers**: system is a collection of cores/CPUs/etc.
– **all-to-all communication**: communicate via messages
– **bandwidth limit B**: limited data transfer

Total "switch" bandwidth: \( B_k(k-1) \approx k^2B \)

Example numbers:

- \( k = 5000, \) 10 Gbps switch
  \( \Rightarrow \) \( B = 400 \text{ bits/sec} \)
k-Machine Model

Basic assumptions:

- **k servers**: system is a collection of cores/CPUs/etc.
- **all-to-all communication**: communicate via messages
- **bandwidth limit B**: limited data transfer

**Total “switch” bandwidth:**

\[ B_k(k-1) \approx k^2B \]

**Example numbers:**

- \( k = 5000 \)
- 100 Gbps switch
- \( B = 4000 \) bits/sec

\[ k = 4 \]
k-Machine Model

Space restriction:

- Problem size: assume size $n$
- Per server: approximately $O(n/k)$

$k = 4$

Example numbers:

$k = 5000$, 1 TB data
$\Rightarrow$ space/core = 200MB
k-Machine Model

Space restriction:

- Problem size: assume size $n$
- Per server: approximately $O(n/k)$

Difference from Map-Reduce:

All the data always needs to be stored somewhere.

Size $O(n/k)$ is optimal.

Example numbers:

$k = 5000$, 1 TB data

$\Rightarrow$ space/core = 200MB
k-Machine Model

Implement Map-Reduce:

- **Map**:
  1. Each server locally runs map function on every key-value pair, saving the new key-value pairs.

- **Reduce**:
  1. Use hash function $h$ to map each key to a machine.
  2. Send $(k, v)$ to machine $h(k)$.
  3. Each machine execute reduce function locally.

- **Repeat**
Implement Map-Reduce:

- **Map:**
  1. Each server locally runs map function on every key-value pair, saving the new key-value pairs.

- **Reduce:**
  1. Use hash function $h$ to map each key to a machine.
  2. Send $(k, v)$ to machine $h(k)$.
  3. Each machine execute reduce function locally.
- **Repeat**

Works correctly if bandwidth/space are sufficient to send/store key-values pairs during the reduce phase.
Implement Map-Reduce:

- **Map:**
  1. Each server locally runs map function on every key-value pair, saving the new key-value pairs.

- **Reduce:**
  1. Use hash function \( h \) to map each key to a machine.
  2. Send \((k, v)\) to machine \( h(k) \).
  3. Each machine execute reduce function locally.

- **Repeat**

**k-Machine Model**

 Works correctly if bandwidth/space are sufficient to send/store key-values pairs during the reduce phase.

**ToDo:** Implement associated reduce functions.
k-Machine Model

Conclusion:

If you can solve the problem in $T$ rounds of Map-Reduce, then you can solve it in the $k$-Machine model in $T$ rounds.
Where is the data?

Random Partition Model:
Initially, data is randomly divided among the machines.
Example: sorting $n$ integers.

Each integer is assigned to a random machine.
k-Machine Model

Example: sorting $n$ integers.

Each integer is assigned to a random machine.

$E[\text{ints per machine}] = \frac{n}{k}$
Detour: balls-in-bins

Random process:
- Take $n$ balls and $k < n$ bins.
- Put each ball in a random bin.

Theorem:
Each bin has $O\left(\frac{n}{k} + \log n\right)$ balls with high probability.

\[ \geq \left(1 - \frac{1}{n^c}\right) \]
Detour: balls-in-bins

Random process:

– Take $n$ balls and $k$ bins.
– Put each ball in a random bin.

Proof:

Pick one bin.
Random process:

- Take \( n \) balls and \( k \) bins.
- Put each ball in a random bin.

Proof:

Define \( x_i = 1 \) if ball \( i \) is in the bin.
Define \( x_i = 0 \) if ball \( i \) is NOT in the bin.
Detour: balls-in-bins

Random process:

- Take $n$ balls and $k$ bins.
- Put each ball in a random bin.

Proof:

Define $x_i = 1$ if ball $i$ is in the bin.
Define $x_i = 0$ if ball $i$ is NOT in the bin.

$$E[x_i] = \Pr(x_i = 1) = \frac{1}{k}$$
Detour: balls-in-bins

Random process:
- Take $n$ balls and $k$ bins.
- Put each ball in a random bin.

Proof:
Define $x_i = 1$ if ball $i$ is in the bin.
Define $x_i = 0$ if ball $i$ is NOT in the bin.

number of balls in bin = $X = \sum_{i=1}^{n} x_i$
Detour: balls-in-bins

Random process:
- Take \( n \) balls and \( k \) bins.
- Put each ball in a random bin.

Proof:

number of balls in bin = \( X = \sum_{i=1}^{n} x_i \)

\[
E[X] = \sum_{i=1}^{n} E[x_i] = \frac{n}{k}
\]
Random process:

- Take \( n \) balls and \( k \) bins.
- Put each ball in a random bin.

Proof:

**Chernoff Bound:** \( \delta > 1 \)

\[
\Pr \left( X \geq (1 + \delta) \frac{n}{k} \right) \leq e^{-\frac{n \delta}{3}}
\]
Detour: balls-in-bins

Random process:

– Take \( n \) balls and \( k \) bins.
– Put each ball in a random bin.

Proof:

Case 1: \( (n/k) > \log(n) \)

\[
\Pr \left( X \geq (1 + 5) \frac{n}{k} \right) \leq e^{- \frac{n}{k} \frac{\delta}{3}} \\
\leq e^{-2 \log n} \\
\leq \frac{1}{n^2}
\]
Detour: balls-in-bins

Random process:
- Take $n$ balls and $k$ bins.
- Put each ball in a random bin.

Proof:

Case 2: $(n/k) < \log(n)$

$$\Pr \left( X \geq \left( 1 + 6 \log(n) \frac{k}{n} \right) \frac{n}{k} \right) \leq e^{-\frac{n \delta}{k \frac{1}{3}}}$$

$$\leq e^{-6 \log n \frac{k}{n} \frac{n}{k} \frac{1}{3}}$$

$$\leq e^{-2 \log n}$$

$$\leq 1/n^2$$
Random process:

- Take $n$ balls and $k$ bins.
- Put each ball in a random bin.

Proof:

Conclusion: w.p. $> (1 - 1/n^2)$

$$X \leq 6 \frac{n}{k}$$

or

$$X \leq \left(1 + 6 \log n \frac{k}{n}\right) \frac{n}{k} \leq 7 \log n$$

$n/k < \log(n)$
Detour: balls-in-bins

Random process:
- Take \(n\) balls and \(k\) bins.
- Put each ball in a random bin.

Proof:

Conclusion: w.p. \(> (1 - 1/n^2)\)

\[ X \leq O \left( \frac{n}{k} + \log n \right) \]

\[ \delta = 6 \log n \frac{k}{n} \]
Detour: balls-in-bins

Random process:

- Take $n$ balls and $k$ bins.
- Put each ball in a random bin.

Proof:

Conclusion: w.p. $> (1 - 1/n^2)$

$$X \leq O \left( \frac{n}{k} + \log n \right)$$

Union bound over all $k < n$ bins…
Random process:
- Take $n$ balls and $k < n$ bins.
- Put each ball in a random bin.

Theorem:
Each bin has $O\left(\frac{n}{k} + \log n\right)$ balls with high probability.

$\geq \left(1 - \frac{1}{n^c}\right)$
Example: sorting $n$ integers.

Each integer is assigned to a random machine.

With high probability, per machine:

$$O\left(\frac{n}{k} + \log n\right)$$
Let $G = (V, E)$ be a graph with $n$ nodes and $m$ edges.
Graph Algorithms

Let $G = (V, E)$ be a graph with $n$ nodes and $m$ edges.

Randomly assign nodes to machines.

Assume $k < n^{\frac{1}{2}}$
Let $G = (V, E)$ be a graph with $n$ nodes and $m$ edges.

Randomly assign nodes to machines.

Assume $k < n^{1/2}$
Graph Algorithms

Let $G = (V, E)$ be a graph with $n$ nodes and $m$ edges.

Randomly assign nodes to machines.

With high probability, node per machine:

$$O \left( \frac{n}{k} + \log n \right) \leq O \left( \frac{n}{k} \right)$$
How many edges stored on each machine?

\[ O \left( \frac{m}{k} \right) \]
Graph Algorithms

How many edges stored on each machine?

$O\left(\frac{m}{k}\right)$

one node $\Rightarrow$ 7 edges!

$m = 11$
How many edges stored on each machine?

$$O\left(\frac{m}{k}\right)$$

Why doesn’t Chernoff Bound work?

Assume $k < n^{1/2}$
How many edges stored on each machine?

\[ O \left( \frac{m}{k} \right) \]

Why doesn’t Chernoff Bound work?

Edges are not independent!
Graph Algorithms

Theorem:
With high probability, each machine has

\[
O \left( \frac{m}{k} + \Delta \log n \right)
\]

edges, where \( \Delta = \text{maximum degree of } G \).
Proof:

Let $n_j =$ number of nodes with degree $[2^i, 2^{i+1})$

- $N_1 =$ nodes with degree $\{1\}$
- $N_2 =$ nodes with degree $\{2, 3\}$
- $N_3 =$ nodes with degree $\{4, 5, 6, 7\}$
- ...
Graph Algorithms

Proof:
Let $n_i = \text{number of nodes with degree } [2^i, 2^{i+1})$

Balls and bins:
Each machine has at most $O(n_i/k + \log n)$ nodes with degree $[2^i, 2^{i+1})$, w.h.p.

Assume $k < n^{1/2}$
Proof:

\[ \text{edges} \leq \sum_{i=1}^{\log \Delta} \left( \frac{n_i}{k} \cdot 2^{i+1} + 2^{i+1} \log n \right) \]

Assume \( k < n^{1/2} \)
Proof:

\[
\text{edges} \leq \sum_{i=1}^{\log \Delta} \left( \frac{n_i}{k} 2^{i+1} + 2^{i+1} \log n \right)
\]

\[
= \frac{1}{k} \sum_{i=1}^{\log \Delta} (n_i 2^{i+1}) + \log n \sum_{i=1}^{\log \Delta} 2^{i+1}
\]

Assume \( k < n^{\frac{1}{2}} \)
Graph Algorithms

Proof:

\[
\text{edges} \leq \sum_{i=1}^{\log \Delta} \left( \frac{n_i}{k} 2^i + 2^{i+1} \log n \right)
\]

\[
= \frac{1}{k} \sum_{i=1}^{\log \Delta} (n_i 2^{i+1}) + \log n \sum_{i=1}^{\log \Delta} 2^{i+1}
\]

\[
= \frac{1}{k} \sum_{i=1}^{\log \Delta} (n_i 2^{i+1}) + 4\Delta \log n
\]

sum:

\[2\Delta + \Delta + \Delta/2 + \Delta/4 + \Delta/8 + \ldots\]
Proof:

\[
\text{edges} \leq \sum_{i=1}^{\log \Delta} \left( \frac{n_i}{k} 2^{i+1} + 2^{i+1} \log n \right)
\]

\[
= \frac{1}{k} \sum_{i=1}^{\log \Delta} (n_i 2^{i+1}) + \log n \sum_{i=1}^{\log \Delta} 2^{i+1}
\]

\[
= \frac{1}{k} \sum_{i=1}^{\log \Delta} (n_i 2^{i+1}) + 4\Delta \log n
\]

\[
= \frac{1}{k} (4m) + 4\Delta \log n
\]

sum:

each edge in the graph, twice

Assume \( k < n^{\frac{1}{2}} \)
Theorem:

With high probability, each machine has

\[ O \left( \frac{m}{k} + \Delta \log n \right) \]

edges, where \( \Delta = \) maximum degree of \( G \).
Theorem:
With high probability, each pair of machines has edges connecting them.

\[ O \left( \frac{m}{k^2} + \frac{\Delta}{k} \log n \right) \]
edges connecting them.
Theorem:
With high probability, each pair of machines has edges connecting them.

\[ O \left( \frac{m}{k^2} + \frac{\Delta}{k} \log n \right) \]

edges connecting them.

Assume \( k < n^{\frac{1}{2}} \)

Balls-and-bins?

Chernoff Bound?
Graph Algorithms

Theorem:

With high probability, each pair of machines has

\[ O \left( \frac{m}{k^2} + \frac{\Delta}{k} \log n \right) \]

edges connecting them.

Assume \( k < n^{\frac{1}{2}} \)

Balls-and-bins?
Chernoff Bound? YES
Graph Algorithms

Theorem:
With high probability, each pair of machines has
\[ O \left( \frac{m}{k^2} + \frac{\Delta}{k} \log n \right) \]
edges connecting them.

Proof:
Fix a machine.
The other endpoint of
each edge is independent.
Graph Algorithms

Theorem:
With high probability, each pair of machines has
\[ O \left( \frac{m}{k^2} + \frac{\Delta}{k} \log n \right) \]
edges connecting them.

Proof:
W.h.p., machine has
\[ O \left( \frac{m}{k} + \Delta \log n \right) \] edges.

Assume \( k < n^{1/2} \)
Theorem:
With high probability, each pair of machines has edges connecting them.

Proof:
W.h.p., machine has edges.

\[ O\left(\frac{m}{k^2} + \frac{\Delta}{k} \log n\right) \]
edges connecting them.

Assume \( k < n^{\frac{1}{2}} / \log n \).

So w.h.p. \( 1/(k-1) \) got to each other machine.
Key Theorems

\[ O \left( \frac{n}{k} \right) \] nodes per machine, w.h.p.

\[ O \left( \frac{m}{k} + \Delta \log n \right) \] edges per machine, w.h.p.

\[ O \left( \frac{m}{k^2} + \frac{\Delta}{k} \log n \right) \] edges between two machines, w.h.p.
Example 1: Send information

Each node in the graph sends 1 bit to each of its neighbors.
Example 1: Send information

Each node in the graph sends 1 bit to each of its neighbors.

Time:

\[ O \left( \frac{1}{B} \left[ \frac{m}{k^2} + \frac{\Delta}{k} \log n \right] \right) \]
Example 2: Luby’s Algorithm

Repeat $\log(n)$ times:

1. Mark and send to neighbors.
2. Unmark and send to neighbors.
3. Delete and send to neighbors.

Time:

$O \left( \frac{1}{B} \left[ \frac{m}{k^2} + \frac{\Delta}{k} \log n \right] \right)$
Example 2: Luby’s Algorithm

Better analysis:

• Each node sends same message to all neighbors.
• Only need to send \((n/k)\) messages per link.

one message, not two!
Example 2: Luby’s Algorithm

Repeat $\log(n)$ times:

1. Mark and send to neighbors.
2. Unmark and send to neighbors.
3. Delete and send to neighbors.

Time:

$O \left( \frac{1}{B \frac{n}{k}} \log n \right)$
Sparse graph:
- $k = 5000$
- $n = 100,000$
- $m = 1,000,000$
- $B = 400$ (10GBps switch)

Some possible numbers:

$$\frac{1}{B} \left( \frac{m}{k} \right) \approx 50s$$

$$\frac{1}{B} \left( \frac{n}{k} \right) \approx 50ms$$

$$\frac{1}{B} \left( \frac{m}{k^2} \right) \approx 10ms$$
Some possible numbers:

Dense graph:
- $k = 5000$
- $n = 100,000$
- $m = 3,000,000,000$
- $B = 400$ (10GBps switch)

$$\frac{1}{B} \left( \frac{m}{k} \right) \approx 25\text{min}$$ very slow

$$\frac{1}{B} \left( \frac{n}{k} \right) \approx 50\text{ms}$$ fastest

$$\frac{1}{B} \left( \frac{m}{k^2} \right) \approx 300\text{ms}$$
Example 2: Luby’s Algorithm

Repeat $\log(n)$ times:

1. Mark and send to neighbors.
2. Unmark and send to neighbors.
3. Delete and send to neighbors.

Time:

$O \left( \frac{1}{B} \frac{n}{k} \log n \right)$

$\Rightarrow < 20$ seconds?
Example 3: Bellman-Ford

Repeat $D$ times:

1. Send your estimate to all your neighbors.
2. After receiving all neighbors estimates, relax all neighboring edges.
Example 3: Bellman-Ford

Repeat $D$ times:

1. Send your estimate to all your neighbors.
2. After receiving all neighbors estimates, relax all neighboring edges.

Time:

$O \left( \frac{D}{B} \frac{n}{k} \right)$
Summary

Today: k-Machine

k-Machine Model
- Cluster computing

Some simple examples
- Luby’s
- Bellman-Ford

Minimum Spanning Tree
- Basic algorithm
- Fully distributed algorithm
- Lower bound

Last Week: Map-Reduce

Map-Reduce Model
- Cluster computing

Some simple examples
- Word count
- Join

Algorithms
- Bellman-Ford
- PageRank
Minimum Spanning Tree

Assumptions:

Graph $G = (V,E)$
- Undirected
- Weighted
- Connected
- $n$ nodes
- $m$ edges

Output:
Each machine knows which edges adjacent to its nodes are in the MST.

Example: output 16
Key idea:

For every cut in the graph, the minimum weight edge across the cut is in the MST.
Minimum Spanning Tree

Boruvka’s Algorithm

Key idea:

For every cut in the graph, the minimum weight edge across the cut is in the MST.
Minimum Spanning Tree

Boruvka’s Algorithm

Key idea:
For every cut in the graph, the minimum weight edge across the cut is in the MST.
Minimum Spanning Tree

Boruvka’s Algorithm

Key idea:

For every cut in the graph, the minimum weight edge across the cut is in the MST.
Minimum Spanning Tree

Boruvka’s Algorithm

Key idea:

For every cut in the graph, the minimum weight edge across the cut is in the MST.
Minimum Spanning Tree

Key idea:

For every cut in the graph, the minimum weight edge across the cut is in the MST.

Proof (sketch):
• Add the edge $e$, creating a cycle.
• Delete $e'$, heaviest edge on cycle.
• Since $e$ is smallest across cut, there is some heavier edge on cycle, i.e., $e' \neq e$. 

Boruvka’s Algorithm
Minimum Spanning Tree

Boruvka’s Algorithm

Initially: Every node is in its own component.
Minimum Spanning Tree

Boruvka’s Algorithm

Every component finds its min weight outgoing edge.
Minimum Spanning Tree

Boruvka’s Algorithm

Add min weight outgoing edges to MST.
Minimum Spanning Tree

Boruvka’s Algorithm

Merge components connected by MWOE.
Minimum Spanning Tree

Boruvka’s Algorithm

Repeat
Minimum Spanning Tree

Boruvka’s Algorithm

Find and add MWOE
Minimum Spanning Tree

Boruvka’s Algorithm

Merge components connected by MWOE.
Minimum Spanning Tree

Boruvka’s Algorithm

Merge components connected by MWOE.
Claim: in each step, the number of components at least divides by 2.
Minimum Spanning Tree

Boruvka’s Algorithm

Claim: Terminates in $O(\log n)$ iterations.
Boruvka’s Algorithm

Repeat $\log n$ times:

1. Find minimum weight outgoing edge (MWOE) for each component.
2. Merge components connected by MWOEs.
Tag each node with its component identifier.

Initially, each node is in its own component.

Component id = node id
Every node broadcasts *to everyone* its component id.

Each node now knows the component id of each neighbor in the graph.
Every node computes its MWOE.

Each node now knows the component id of each neighbor in the graph.

So it considers only edges that go to other components.
**k-Machine Boruvka’s Algorithm**

**Boruvka’s Algorithm**

Every node broadcasts its MWOE to everyone.

Each node can compute MWOE for its component because it knows MWOE for every node in its component.
Every node broadcasts its MWOE to everyone.

Each node can compute MWOE for all components!

Can find all components that you will merge with.
k-Machine Boruvka’s Algorithm

Boruvka’s Algorithm

Compute new component id.

Find minimum component id of any component that you merge with.
Boruvka’s Algorithm

Repeat \( \log n \) times:

1. Broadcast component id to all.
2. Broadcast MWOE to all.
3. Compute new component id.
Boruvka’s Algorithm

Repeat \( \log n \) times:
1. Broadcast component id to all.
2. Broadcast MWOE to all.
3. Compute new component id.

What is the cost of broadcasting a message to “all” nodes in the graph?
Boruvka’s Algorithm

Repeat \( \log n \) times:

1. Broadcast component id to all.
2. Broadcast MWOE to all.
3. Compute new component id.

What is the cost of broadcasting a message to “all” nodes in the graph?

\[
O \left( \frac{1}{B} \frac{n}{k} \right)
\]

Each machine needs to send \( \frac{n}{k} \) identifiers to all \( k \) other machines.
Boruvka’s Algorithm

Repeat $\log n$ times:
1. Broadcast component id to all.
2. Broadcast MWOE to all.
3. Compute new component id.

Total running time:

$$O \left( \frac{1}{B} \frac{n}{k} \log n \right)$$
Assume each node in the graph is its own machine.

Almost like $k=n$?
Assume each node in the graph is its own machine.

Each edge in the graph is a real communication edge.

Cannot send message to everyone like in k-machine model.
Fully Distributed Model

CONGEST Model

Assume each node in the graph is its own machine.

Each edge in the graph is a real communication edge.

Each edge carries 1 message per round.

Sort of like $B = \log n$
Key challenge:

Find minimum weight outgoing edge for a component.
Step 1:

Each node sends a message to all its neighbors with its component id.

$O(1)$ rounds
Step 2:

Each node computes its minimum weight outgoing edge to a different component.

0 rounds
Step 3:

Send MWOE on the MST tree edges in your component.

Fully Distributed Model

Boruvka’s Algorithm
Fully Distributed Model

Boruvka’s Algorithm

Detail:

Maintain MST fragment in component as a rooted tree.
Each node broadcasts MWOE up the tree to the root.
Full Distributed Model

Boruvka’s Algorithm

Detail:

Root chooses smallest weight MWOE.
Boruvka’s Algorithm

Detail:

Root broadcasts MWOE to everyone.
Fully Distributed Model

Boruvka’s Algorithm

Detail:

Merge edge is found.
Fully Distributed Model

Boruvka’s Algorithm

Detail:

Flood minimum component id through all merged components.
If root is in another component, reorient tree.
Step 3:

Send MWOE on the MST tree edges in your component.

And merge.

Any problem?
Fully Distributed Model

Boruvka’s Algorithm

Detail:

Each node broadcasts MWOE up the tree to the root.

Time: $\Omega(n)$
Step 3 (revised):

If component size is \(< n^{\frac{1}{2}}\), then send MWOE on the MST tree edges in your component, and merge.

Time: \(O(n^{\frac{1}{2}})\)
Repeat until all components are size >$n^{\frac{1}{2}}$ :

1. Find MWOE for each node.
2. Collect MWOE for each component at the root of the component, using the MST fragment edges.
3. Merge components.

Time: $O(n^{\frac{1}{2}} \log n)$
Fully Distributed Model

\textbf{Boruvka’s Algorithm}

Idea 2: Use a BFS tree.

1. Find a BFS tree for the entire graph.
2. Collect MWOE for each component at the root of the BFS tree, using the BFS tree edges.
3. Merge components.
Idea 2: Use a BFS tree.
Idea 2: Use a BFS tree.

Easy to find.
Just have a root start broadcasting a message to all its neighbors.
Idea 2: Use a BFS tree.

Easy to find.

When receive BFS message, then rebroadcast to your neighbors.
Idea 2: Use a BFS tree.

Easy to find.

Parent in BFS tree is first node that you received a message from.
Idea 2: Use a BFS tree.

Max depth: $O(D)$

$D = \text{diameter of graph.}$
How to send MWOE up tree?

Wait until you have received all MWOE from all your children.
How to send MWOE up tree?

Compute one min weight edge for each component.
How to send MWOE up tree?

Send all to your parent.
How to send MWOE up tree?

Send all to your parent.

At most $n^{\frac{1}{2}}$ MWOE to send to parent.
How to send MWOE up tree?

Send all to your parent.

At most $n^{\frac{1}{2}}$ MWOE to send to parent.

Takes at most $Dn^{\frac{1}{2}}$ time for all messages to reach root.
How to send MWOE up tree?

Send all to your parent.

At most $n^{\frac{1}{2}}$ MWOE to send to parent.

Takes at most $Dn^{\frac{1}{2}}$ time for all messages to reach root.

Key reason why we first had to build components of size $n^{\frac{1}{2}}$!
Improvement: first aggregate in $n^{\frac{1}{2}}$ sized base fragments.

Never more than $n^{\frac{1}{2}}$ MWOE to send to root total.

Key reason why we first had to build components of size $n^{\frac{1}{2}}$!
Fully Distributed Model

Boruvka’s Algorithm

Improvement: pipeline.

Send on MWOE as soon as you receive it.

Never delayed by another MWOE more than once.
Conclusion.

\[ O(D + n^{\frac{1}{2}}) \] time to aggregate MWOE and perform merge.
Repeat:

1. Find MWOE for each node.
2. If component is $< n^{\frac{1}{2}}$ then aggregate MWOE in component. Otherwise aggregate on BFS tree.
3. Merge components.

Time: $O((D + n^{\frac{1}{2}})\log n)$
Can we do better than $n^{\frac{1}{2}}$?
Can we do better than $n^{\frac{1}{2}}$?

NO!
Lower Bound

Minimum Spanning Tree

\(n^{\frac{3}{2}}\) by \(n^{\frac{3}{2}}\) grid
Two special nodes: A and B
Lower Bound

Minimum Spanning Tree

Thick green edges: light weight (should go in MST).
Dashed red edges: heavy weight (should NOT go in MST)
Lower Bound

Minimum Spanning Tree

How do A and B decide which edges to include?
Must communicate with each other!
How do A and B decide which edges to include?
Must communicate with each other!
Shortest path connecting A and B is $n^{\frac{1}{2}}$ so algorithm must take $n^{\frac{1}{2}}$ time.
Shortest path connecting A and B is $n^{\frac{1}{2}}$ so algorithm must take $n^{\frac{1}{2}}$ time.
Lower Bound

Minimum Spanning Tree

Diameter is $n^{\frac{1}{2}}$ so algorithm runs in $O(D)$ time.
Build a tree!

\[ D = O(\log n) \]
A and B still have to decide whether their adjacent edges are in the MST.
But:
What if A and B exchange information via the root?
Theorem
A and B have to exchange at least $n^{\frac{1}{2}}$ bits of information.

Uses 2-party communication complexity.
Theorem

A and B have to exchange at least $n^{\frac{1}{2}}$ bits of information.

To send $n^{\frac{1}{2}}$ bits of information through the root takes $\Omega(n^{\frac{1}{2}} / \log n)$ time.
Theorem
A and B have to exchange at least $n^{\frac{1}{2}}$ bits of information.

To send $n^{\frac{1}{2}}$ bits of information through the root takes $\Omega(n^{\frac{1}{2}} / \log n)$ time.

Either A and B exchange information through the grid (which has long paths) or they exchange information via the tree/root which has congestion.
Theorem:

A and B have to exchange at least $n^{\frac{1}{2}}$ bits of information.

To send $n^{\frac{1}{2}}$ bits of information through the root takes $\Omega(n^{\frac{1}{2}} / \log n)$ time.

Theorem:

Finding an MST takes $\Omega(D + n^{\frac{1}{2}} / \log n)$ time (even in a graph with $D = O(\log n)$).

Either A and B exchange information through the grid (which has long paths) or they exchange information via the tree/root which has congestion.
Summary

Today: k-Machine

k-Machine Model
- Cluster computing

Some simple examples
- Luby’s
- Bellman-Ford

Minimum Spanning Tree
- Basic algorithm
- Fully distributed algorithm
- Lower bound

Last Week: Map-Reduce

Map-Reduce Model
- Cluster computing

Some simple examples
- Word count
- Join

Algorithms
- Bellman-Ford
- PageRank
Design Some Algorithms

Design k-Machine algorithms:

- Sorting
- Finding a median
- Prefix-Sum

Maximal matching

A little more:

What about PageRank?
PageRank (Last Week)

**PageRank(G)**

Choose a random node \( v \) (uniformly) from \( G \)

Repeat many times:

1. With probability \( \frac{1}{2} \): stay at node \( v \).
2. With probability \( \frac{1}{2} \): choose a neighbor of \( v \) uniformly at random and go to that neighbor.

Assign to each node \( u \) the probability that you are at node \( u \) when the process terminates.
PageRank (Today)

PageRank(G) (Version 1)

Choose a random node \( v \) (uniformly) from \( G \)

Repeat many times:

1. With probability \( \varepsilon \): restart at a new node chosen uniformly at random.

2. With probability \( (1 - \varepsilon) \): choose a neighbor of \( v \) uniformly at random and go to that neighbor.

Assign to each node \( u \) the probability that you are at node \( u \) when the process terminates.
PageRank

Equivalent: (Version 2)

- Start a random walk at a random node \( v \).

- At every step:
  1. With probability \( \varepsilon \) stop and return \( v \).
  2. With probability \((1-\varepsilon)\) choose a neighbor uniformly at random and go there.

\[
\text{PageRank}(v) = \text{probability that process stops at } v.
\]
PageRank

1) Explain why the two versions are equivalent.
Imagine running the process above $n \log n$ times.

If $x$ random walks visit a node, then

$$(\varepsilon x / n \log n)$$

is a good estimate of the PageRank.

(Prove it? Essentially, just Chernoff Bounds.)
1) Explain why the two versions are equivalent.

2) Give an algorithm for the k–machine model that runs the process $n \log n$ times in parallel and computes the PageRank. How long does it take?
Design Some Algorithms

Design k-Machine algorithms:

- Sorting
- Finding a median
- Prefix-Sum
- Maximal matching

A little more:

What about PageRank?

1) Explain why the two versions are equivalent.

2) Give an algorithm for the k-machine model that runs the process $n \log n$ times in parallel and computes the PageRank. How long does it take?