CSE 101

Algorithm Design and Analysis
Russell Impagliazzo
Miles Jones

mej016@eng.ucsd.edu
russell@eng.ucsd.edu

Russell’s Office 4248 CSE Building
Miles’ Office 4208 CSE Building
Kruskal’s algorithm for finding the minimum spanning tree

- Start with a graph with only the vertices.
- Repeatedly add the next lightest edge that does not form a cycle.
How to implement Kruskal’s

• Start with an empty graph $R$. (Only vertices, no edges.)

• Sort edges by weight from smallest to largest.
  • For each edge $e$ in sorted order:
    • If $e$ does not create a cycle in $R$ then
      • Add $e$ to $R$
    • otherwise
      • do not add $e$ to $R$

• How do we tell if adding an edge will create a cycle?
Let’s ask our standard DS questions

• What kind of object do we need to keep track on in Kruskal’s algorithm?

• What do we need to know in one step?

• How does the structure change in one step?
Let’s ask our standard DS questions

• What kind of object do we need to keep track on in Kruskal’s algorithm? We need to keep track of the way the edges added to the MST divide up the vertices into components.

• What do we need to know in one step? Are two vertices in the same component?

• How does the structure change in one step? If we add an edge, it merges the two components into one.
DSDS stands for Disjoint Sets Data Structure.

What can it do?

Given a set of objects, DSDS manage partitioning the set into disjoint subsets.

It does the following operations:

- **Makeset(S)**: puts each element of S into a set by itself.
- **Find(u)**: it returns the name of the subset containing u.
- **Union(u,v)**: it unions the set containing u with the set containing v.
procedure Kruskal(G,w)
  • Input: undirected connected graph G with edge weights w
  • Output a set of edges X that defines an MST of G

Makeset(V)

X = {}

Sort the edges in E in increasing order by weight.
For all edges (u,v) in E until X is a connected graph
  • if find(u) ≠ find(v):
    • Add edge (u,v) to X
    • Union(u,v)
Kruskal’s algorithm USING DSDS

• procedure Kruskal(G,w)
  • Input: undirected graph G with edge weights w
  • Output: A set of edges X that defines a minimum spanning tree

• for all \( v \in V \)
  • Makeset(v) \| V \| *(makeset)

• \( X = \{ \} \)

• sort the set of edges E in increasing order by weight sort(|E|)

• for all edges \((u, v) \in E\) until \(|X| = |V| - 1\)
  • if find(u) ≠ find(v):
    • add \((u, v)\) to \(X\)
    • union(u, v) \| V-1 \| *(union)
Subroutines of Kruskal’s

- `makeset (u)`: This creates a set with one element, `u`
- `find(u)`: finds the set to which `u` belongs
- `union(u,v)`: merges the sets containing `u` and `v`

Runtime of Kruskal’s:

\[ |V| \text{makeset} + 2|E| \text{find} + (|V| - 1) \text{union} + \text{sort}(|E|) \]
DSDS VERSION 1 (array)

- Keep an array Leader(u) indexed by element
- In each array position, keep the leader of its set
- Makeset(u):
- Find(u):
- union(u,v):
- Total time:
Example DSDS version 1 (array)

(A,D)=1
(E,G)=1
(A,B)=2
(A,C)=2
(B,C)=2
(B,E)=2
(D,G)=2
(D,E)=3
(E,F)=4
(F,G)=4
Example DSDS version 1 (array)

(A,D)=1  (E,G)=1  (A,B)=2  (A,C)=2  (B,C)=2  (B,E)=2  (D,G)=2  (D,E)=3  (E,F)=4  (F,G)=4
DSDS VERSION 1 (array)

- Keep an array Leader(u) indexed by element
- In each array position, keep the leader of its set
- Makeset(u):  $O(1)$
- Find(u):  $O(1)$
- union(u,v):  $O(|V|)$.
- Total time:  $O(|E|*1+|V|^2 + |E|\log|E|) = O (|V|^2 + |E|\log |E|)$.  

Each set is a rooted tree, with the vertices of the tree labelled with the elements of the set and the root the leader of the set.

Only need to go up to leader, so just need parent pointer.

Because we’re only going up, we don’t need to make it a binary tree or any other fixed fan-in.
Version 2a: DSDS operations

- Find: go up tree until we reach root
  - Find(v).
  - L = v.
  - Until p(L) == L, do: L = p(L)
- Assume union is only done for distinct roots. We just make one root the child of the other.
- Union(u, v)
  - p(v) = u
## Example DSDS version 2a (tree)

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
<th>G</th>
</tr>
</thead>
<tbody>
<tr>
<td>(E,G)=1</td>
<td>A</td>
<td>. B</td>
<td>. C</td>
<td>A</td>
<td>E</td>
<td>F</td>
<td>E</td>
</tr>
<tr>
<td>(A,B)=2</td>
<td>B</td>
<td>. B</td>
<td>. C</td>
<td>A</td>
<td>E</td>
<td>F</td>
<td>E</td>
</tr>
<tr>
<td>(A,C)=2</td>
<td>B</td>
<td>. C</td>
<td>. C</td>
<td>A</td>
<td>E</td>
<td>F</td>
<td>E</td>
</tr>
<tr>
<td>(B,C)=2</td>
<td>B</td>
<td>. C</td>
<td>. E</td>
<td>A</td>
<td>E</td>
<td>F</td>
<td>E</td>
</tr>
<tr>
<td>(B,E)=2</td>
<td>B</td>
<td>. C</td>
<td>E</td>
<td>A</td>
<td>E</td>
<td>F</td>
<td>E</td>
</tr>
<tr>
<td>(D,G)=2</td>
<td>B</td>
<td>. C</td>
<td>E</td>
<td>A</td>
<td>E</td>
<td>F</td>
<td>E</td>
</tr>
<tr>
<td>(D,E)=3</td>
<td>B</td>
<td>. C</td>
<td>E</td>
<td>A</td>
<td>E</td>
<td>E</td>
<td>E</td>
</tr>
<tr>
<td>(E,F)=4</td>
<td>B</td>
<td>. C</td>
<td>E</td>
<td>A</td>
<td>E</td>
<td>E</td>
<td>E</td>
</tr>
<tr>
<td>(F,G)=4</td>
<td>B</td>
<td>. C</td>
<td>E</td>
<td>A</td>
<td>E</td>
<td>E</td>
<td>E</td>
</tr>
</tbody>
</table>
Version 2a: DSDS operations

- **Find**: go up tree until we reach root:
  - Time = depth of tree, could be \(O(|V|)\)
- **Find**\((v)\).
- \(L = v\).
- Until \(p(L) == L\), do: \(L = p(L)\)
- We just make one root the child of the other. \(O(1)\)
- **Union**\((u, v)\)
- \(p(v) = u\)
DSDS VERSION 2a (tree)

- Keep an array parent(u) indexed by element
- In each array position, keep parent pointer
- Makeset(u): O(1)
- Find(u): O(|V|)
- union(u,v): O(|1|).
- Total time: O(|E|*|V|+|V|*1+ |E|log|E|) = O(|V||E|)
- Seems worse. But can we improve it?
DSDS VERSION 2a (tree)

- Find(u) : $O(|V|)\$
- union(u,v) : $O(|1|)\$
- Total time: $O(|E|*|V|+|V|*1+ |E|\log|E|) = O(|V||E|)$
- Seems worse. But can we improve it? Bottleneck: find when depth of tree gets large. Solution: To keep depth small, choose smaller depth to be child.
version 2b (union-by-rank)

- vertices of the trees are elements of a set and each vertex points to its parent that eventually points to the root.
- The root points to itself.
- The actual information in the data structure is stored in two arrays:
  - \( p(v) \): the parent pointer (roots point to themselves)
  - \( \text{rank}(v) \): the depth of the tree hanging from \( v \). (Note: in later versions, we’ll keep rank, but it will no longer be the exact depth, which will be more variable.) Initially, \( \text{rank}(v)=0 \)
Version 2b: DSDS operations

- Find(v).
- L=v.
- Until p(L) ==L, do: L=p(L)
- We make smaller depth root the child of the other.
- Union(u, v)
  - If rank(u) > rank(v) Then: p(v)= u;
  - If rank(u) < rank(v). Then: p(u)=v
  - If rank(u)=rank(v). Then p(v)=u, rank(u)++
Lemma:

- If we use union-by-rank, the depth of the trees is at most $\log(|V|)$.

- Proof: We show as loop invariant, that if for the leader of a set $u$, $\text{rank}(u)=r$, the tree rooted at $u$ has size at least $2^r$.
  - True at start, each $\text{rank}(u)=0$, tree size is $1 = 2^0$.
  - Only could change with union operation. If roots are different ranks, rank doesn’t change, set size increases.
  - If roots have same rank, rank increases by 1, set sizes add.
  - If we merge two sets of rank $r$, each had at least $2^r$ elements, so merged set has at least $2^{r+1}$ elements, and new rank is $r+1$. 
Lemma:

- If we use union-by-rank, the depth of the trees is at most \( \log(|V|) \).

Proof: Invariant: if for the leader of a set \( u \), \( \text{rank}(u)=r \), the tree rooted at \( u \) has size at least \( 2^r \). Therefore \( r \leq \log|V| \).
- Second invariant: \( \text{rank}(u) \) is the depth of the tree at \( u \).
- So depth is at most \( \log |V| \).
Version 2b: DSDS operations

- **Find(v).** Time = \( O(\text{depth } v) = O(\log |V|) \)
- \( L = v \)
- Until \( p(L) == L \), do: \( L = p(L) \)
- We make smaller depth root the child of the other.
- **Union(u, v).** Still \( O(1) \) time
  - If \( \text{rank}(u) > \text{rank}(v) \) Then: \( p(v) = u \)
  - If \( \text{rank}(u) < \text{rank}(v) \). Then: \( p(u) = v \)
  - If \( \text{rank}(u) = \text{rank}(v) \). Then \( p(v) = u \), \( \text{rank}(u)++ \)
DSDS VERSION 2b (tree)

- \text{Find}(u) : O(\log |V|)
- \text{union}(u,v) : O(|1|).
- Total time: \(O(|E|\log|V| + |V|*1 + |E|\log|E|) = O(|E| \log |V|)\)
- Rest of algorithm matches sort time!
SHOULD WE TRY FOR BETTER?

- Why continue? Can’t improve since bottleneck is sorting.
- Many times, sorting can be done in linear time, e.g., when values are small, can use counting or radix sort
- Many times, inputs come pre-sorted
- Because we want to optimize DSDS for other uses as well
- Because it’s fun (for me, at least)
Path Compression

• We can improve the runtime of find and union by making the height of the trees shorter.

• How do we do that?

• every time we call find, we do some housekeeping by moving up every vertex.
Path Compression

• new find function

function find(x)
  • if $x \neq p(x)$ then
    • $p(x) := find(p(x))$
  • return $p(x)$
Example DSDS version 2c (tree)

\[
\begin{array}{ccccccc}
A & . & B & . & C & . & D & . & E & . & F & . & G \\
\end{array}
\]

\( (A,D)=1 \)
\( (E,G)=1 \)
\( (A,B)=2 \)
\( (A,C)=2 \)
\( (B,C)=2 \)
\( (B,E)=2 \)
\( (D,G)=2 \)
\( (D,E)=3 \)
\( (E,F)=4 \)
\( (F,G)=4 \)

\[
\begin{array}{ccccccc}
\end{array}
\]

\( \text{Rank}(A)=1 \)
\( \text{Rank}(E)=1 \)
\( \text{Rank}(A)=2 \)

\( \text{Rank}(A)=2 \)

\( \text{Rank}(A)=3 \)
\( \text{Rank}(A)=4 \)
find (path compression)

- whenever you call find on a vertex v, it points v and all of its ancestors to the root.

- Seems like a good idea, but how much difference could it make?

- Since worst-case find could be first find, same worst-case time as before.
The ranks do not necessarily represent the height of the graph anymore and so will this cause problems?
Amortized analysis

• Amortized analysis: Bound total time of \( m \) operations, rather than worst-case time for single operation * \( m \)

• Intuition: Fast operations make things worse in the future, but slow operations make things better in the future. (Merging might build up the height of the tree, but finding a deep vertex shrinks the average heights.)