Minimum Spanning Trees
and Union-Find

CSE 101: Design and Analysis of Algorithms
Lecture 7
CSE 101: Design and analysis of algorithms

• Minimum spanning trees and union-find
  – Reading: Section 5.1
• Quiz 1 is today, last 40 minutes of class
• Homework 3 is due Oct 23, 11:59 PM
How to implement Kruskal’s algorithm

• Sort edges by weight, go through from smallest to largest, and add if it does not create cycle with previously added edges

• How do we tell if adding an edge will create a cycle?
  – Naive: depth-first search every time
    • Need to test for every edge, m times
  – Depth-first search on a forest: only edges added to minimum spanning tree
    • As such, each depth-first search is $O(n)$
    • Total time $O(nm)$

$n = |V|$
$m = |E|$
Disjoint sets data structure (DSDS)

- Main complication: want to check if $u$ is connected to $v$ efficiently
- Tree $T$ divides vertices into disjoint sets of connected components
- $u$ is connected to $v$ if they are in the same set
- Adding $e$ to $T$ merges the set containing $u$ with the set containing $v$
- So we need a data structure that
  - Represents a partition of a set $V$ into disjoint subsets
    - We will pick one element $L$ from each subset to be the “leader” of a subset, in order to give the subsets distinct names
  - Has an operation $\text{find}(u)$ that returns the leader of $u$’s set
  - Has an operation $\text{union}(u,v)$ that replaces the two sets containing $u$ and $v$ with their union
Kruskal’s algorithm using a DSDS

procedure kruskal(G,w)
   Input: undirected connected graph G with edge weights w
   Output: a set of edges X that defines a minimum spanning tree of G
   for all v in V
      makeset(v)
   X = { }
   Sort the edges in E in increasing order by weight
   For all edges (u,v) in E
      if find(u) ≠ find(v):
         Add edge (u,v) to X
         union(u,v)

Kruskal’s algorithm using a DSDS

procedure kruskal(G,w)
    Input: undirected connected graph G with edge weights w
    Output: a set of edges X that defines a minimum spanning tree of G
    for all v in V
        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)
Trees

• Definition: A tree is an undirected connected graph with no cycles

• An undirected connected graph is a tree if and only if removing any edge results in two disconnected graphs

• An undirected connected graph with n vertices is a tree if and only if it has n - 1 edges

• An undirected connected graph is a tree if and only if there is a unique path between nodes
Kruskal’s algorithm using a DSDS

procedure kruskal(G,w)
    Input: undirected connected graph G with edge weights w
    Output: a set of edges X that defines a minimum spanning tree of G
    for all v in V
        makeset(v)
    X = { }  
    Sort the edges in E in increasing order by weight
    For all edges (u,v) in E until |X| = |V| - 1
        if find(u) ≠ find(v):
            Add edge (u,v) to X
            union(u,v)
Kruskal’s algorithm using a DSDS

procedure kruskal(G,w)
   Input: undirected connected graph G with edge weights w
   Output: a set of edges X that defines a minimum spanning tree of G
   for all v in V
      makeset(v) |V| * makeset
   X = {} X = \{ \}
   Sort the edges in E in increasing order by weight sort(|E|)
   For all edges (u,v) in E until |X| = |V| - 1 2 * |E| * find
      if find(u) \neq find(v):
         Add edge (u,v) to X
      union(u,v) (|V| - 1) * union
Kruskal’s algorithm, DSDS subroutines

- **makeset(u)**
  - 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

- **Kruskal’s algorithm**
  \[ |V| \times \text{makeset} + 2 \times |E| \times \text{find} + (|V| - 1) \times \text{union} + \text{sort}(|E|) \]
DSDS, leader version

• Keep an array \texttt{leader(u)} indexed by element
• In each array position, keep the leader of its set
• \texttt{makeset(u)}: \texttt{leader(u)} = \texttt{u}
• \texttt{find(u)}: return \texttt{leader(u)}
• \texttt{union(u,v)}: set \texttt{leader(x)} = \texttt{leader(u)}
Example: DSDS, leader version

(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, leader version

(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, leader version

- Keep an array leader(u) indexed by element
- In each array position, keep the leader of its set
- makeset(v): leader(u) = u, O(1)
- find(u): return leader(u), O(1)
- union(u,v): For each array position, if it is currently leader(v), then change it to leader(u). O(|V|)

- Kruskal’s algorithm
  \[ |V| * \text{makeset} + 2 * |E| * \text{find} + (|V| - 1) * \text{union} + \text{sort}(|E|) \]
  \[ = |V| * O(1) + 2 * |E| * O(1) + (|V| - 1) * O(|V|) + \text{sort}(|E|) \]
  \[ = O(|V|^2) \]
A more efficient implementation

• We want to optimize DSDS for other uses as well
• And it’s fun (right?)
DSDS, directed trees with ranks version

• Each set is a rooted tree, with the vertices of the tree labeled with the elements of the set and the root the leader of the set

• To find, only need to go up to leader, so just need parent pointer

• To union, point one leader to other
DSDS, directed trees with ranks version

• 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 root is a convenient representation or name of the set containing it and all of its children
• In addition to the parent pointer of $x$, $\pi(x)$, each vertex also has a rank that tells you the height of the subtree hanging from that vertex
Directed trees with ranks
Directed trees with ranks

\[
\begin{align*}
\text{rank}(A) &= 1 \\
\text{rank}(B) &= 0 \\
\text{rank}(C) &= 0 \\
\text{rank}(D) &= 2 \\
\text{rank}(E) &= 0 \\
\text{rank}(F) &= 0 \\
\text{rank}(G) &= 1
\end{align*}
\]

\[
\begin{align*}
\pi(A) &= D \\
\pi(B) &= D \\
\pi(C) &= G \\
\pi(D) &= D \\
\pi(E) &= A \\
\pi(F) &= A \\
\pi(G) &= G
\end{align*}
\]
DSDS, directed trees with ranks version

procedure makeset(x)

\[ \pi(x) := x \]

rank(x):=0
DSDS, directed trees with ranks version

procedure find(x)
    while (x ≠ π(x))  \text{Goes up parent pointers until root is found}
        x := π(x)
    return x
DSDS, directed trees with ranks version

procedure union(x,y)
    rx:=find(x)
    ry:=find(y)
    if rx=ry then return
    if rank(rx)>rank(ry) then
        \( \pi(ry): = rx \)
    else
        \( \pi(rx): = ry \)
    if rank(rx)=rank(ry) then
        rank(ry):=rank(rx)+1
DSDS, directed trees with ranks version

procedure union(x,y)
    rx:=find(x)
    ry:=find(y)
    if rx=ry then return
    if rank(rx)>rank(ry) then
        π(ry): = rx
    else
        π(rx): = ry
    if rank(rx)=rank(ry) then
        rank(ry):=rank(rx)+1

• To save on runtime, we must keep the heights of the trees short
• As such, union of two ranks points the smaller rank to the bigger rank, that way, the tree will stay the same height
• If the ranks are equal, then it increments one rank and points the smaller to the bigger
  • This is the only way a rank can increase
DSDS, directed trees with ranks version

- union
DSDS, directed trees with ranks version

procedure makeset(x)
    \( \pi(x) := x \)
    rank(x):=0

procedure find(x)
    while \((x \neq \pi(x))\)
    \( x := \pi(x) \)
    return x

procedure union(x,y)
    rx:=find(x)
    ry:=find(y)
    if \( \text{rank}(rx) > \text{rank}(ry) \)
    \( \pi(ry) := rx \)
    else
    if \( \text{rank}(rx) = \text{rank}(ry) \)
    rank(ry):=rank(rx)+1

makeset \( \mathcal{O}(1) \)
find \( \mathcal{O}(\text{height of tree containing } x) \)
union \( \mathcal{O}(\text{find}) \)
Example: DSDS, directed trees with ranks version

(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, directed trees with ranks version
Height of tree

• Any root node of rank k has at least $2^k$ vertices in its tree

• Proof
  – Base Case: a root of rank 0 has 1 vertex
  – Suppose a root of rank k has at least $2^k$ vertices in its tree. Then, a root of rank k+1 can only be made by unioning 2 roots each of rank k. So, a root of rank k+1 must have at least $2^k + 2^k = 2^{k+1}$ vertices in its tree.
Next lecture

• Greedy algorithms
  – Reading: Kleinberg and Tardos, sections 4.1, 4.2, and 4.3