KRUSKAL’S ALGORITHM

Given a forest we’ve built so far,

1. look at all the edges that would join two fragments of the forest
2. pick the lowest-weight one and add it to the tree, thereby joining two fragments
3. Assert: the forest we have so far is part of some minimum spanning tree

Repeat until we have a spanning tree.
KRUSKAL’S ALGORITHM

Given a forest we’ve built so far,

1. look at all the edges that would join two fragments of the forest
2. pick the lowest-weight one and add it to the tree, thereby joining two fragments
3. Assert: the forest we have so far is part of some minimum spanning tree

Repeat until we have a spanning tree.

Don’t recompute these edges every iteration.

Def kruskal(g):
  tree_edges = []
  partition = DisjointSet()
  for v in g.vertices:
    partition.addsingleton(v)
  edges = sorted(g.edges, sortkey = λ(u,v,weight): weight)
  for (u,v,edgeweight) in g.edges:
    p = partition.getsetwith(u)
    q = partition.getsetwith(v)
    if p != q:
      tree_edges.append((u,v))
      partition.merge(p, q)

Just pre-sort the list of all edges, then ignore those that are within-fragment.
Total cost $O(V + E + E \log E)$

We're assuming a connected graph.

$E \geq V - 1 \implies V \leq E + 1$

$E \leq \frac{1}{2}V(V-1) \implies \log E \leq \log V$

Total cost $O(E \log V)$

The abstract data type **DisjointSet** stores a collection of disjoint sets, and supports:

- $O(1)$: `addsingleton(v)`
- $O(1)$: `p = getsetwith(v)`
- $O(1)$: `merge(p,q)`
SECTION 6.7

Topological sort
DEFINITION
Given a directed graph, a **total ordering** is an ordering of the vertices such that if there is an edge \( v \rightarrow u \) in the graph, then \( v < u \) in the ordering.

PROBLEM STATEMENT
Find a total ordering, if one exists.

This graph has a cycle, so no total order is possible.
def dfs_recurse(g, s):
    for v in g.vertices:
        v.visited = False
    visit(s)

def visit(v):
    v.visited = True
    for w in v.neighbours:
        if not w.visited:
            visit(w)

attempt 1: depth-first search

This might not even visit all vertices, so it might not produce a total order.
def dfs_recurse_all(g):
    for v in g.vertices:
        v.visited = False
    for v in g.vertices:
        if not v.visited:
            visit(v)

def visit(v):
    v.visited = True
    for w in v.neighbours:
        if not w.visited:
            visit(w)

attempt 2: comprehensive depth-first search
def dfs_recurse_all(g):
    for v in g.vertices:
        v.visited = False
    for v in g.vertices:
        if not v.visited:
            visit(v)

def visit(v):
    v.visited = True
    for w in v.neighbours:
        if not w.visited:
            visit(w)

attempt 2: comprehensive depth-first search

Some edges point backwards – not a total order.
attempt 2: comprehensive depth-first search

def dfs_recurse_all(g):
    for v in g.vertices:
        v.visited = False
    for v in g.vertices:
        if not v.visited:
            visit(v)

def visit(v):
    v.visited = True
    for w in v.neighbours:
        if not w.visited:
            visit(w)
```python
def toposort(g):
    totalorder = []
    for v in g.vertices:
        if not v.visited:
            visit(v, totalorder)
    return totalorder

def visit(v, totalorder):
    v.visited = True
    totalorder.append(v)
    for w in v.neighbours:
        if not w.visited:
            visit(w, totalorder)
    # v.colour = 'black'
```

```
```
def toposort(g):
    for v in g.vertices:
        v.visited = False
        # v.colour = 'white'
totalorder = []
    for v in g.vertices:
        if not v.visited:
            visit(v, totalorder)
    return totalorder

def visit(v, totalorder):
    v.visited = True
    # v.colour = 'grey'
    for w in v.neighbours:
        if not w.visited:
            visit(w, totalorder)
totalorder.append(v)
    # v.colour = 'black'

Correctness theorem.
Given a DAG $g$, this algorithm produces a totalorder such that for every edge $v_1 \rightarrow v_2$, $v_1$ appears to the right of $v_2$ in totalorder.

$O(V+E)$ running time, like DFS.
depth-first search
breadth-first search
Dijkstra's algorithm
Bellman-Ford algorithm
dynamic programming
Johnson's algorithm
Ford-Fulkerson algorithm
matchings
Prim's algorithm
Kruskal's algorithm
topological sort
translation strategy
QUESTION. How might we segment this image into “handsome stoat” and “background”? 
1. define a grid
2. measure dissimilarity along edges
3. run Kruskal’s algorithm, and stop when the forest it’s building has just a few trees.
2. measure dissimilarity along edges
3. ask the user to label some “stoat” points and some “background” points
4. set up a flow network
5. find a minimum-capacity cut
<table>
<thead>
<tr>
<th>Rank</th>
<th>First name</th>
<th>Last name</th>
<th>Submitted on</th>
<th>Submissions</th>
<th>Grade</th>
<th>Evaluator</th>
<th>Evaluated on</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Kevin</td>
<td>Xie</td>
<td>Sunday, 5 March 2023, 5:06 PM</td>
<td>22</td>
<td>76.76 / 100.00</td>
<td>Automatic grade</td>
<td>Sunday, 5 March 2023, 5:07 PM</td>
</tr>
<tr>
<td>2</td>
<td>Matej</td>
<td>Urban</td>
<td>Sunday, 5 March 2023, 5:33 PM</td>
<td>2</td>
<td>68.70 / 100.00</td>
<td>Automatic grade</td>
<td>Sunday, 5 March 2023, 5:33 PM</td>
</tr>
<tr>
<td>3</td>
<td>Milos</td>
<td>Puric</td>
<td>Wednesday, 1 March 2023, 1:03 AM</td>
<td>10</td>
<td>65.41 / 100.00</td>
<td>Automatic grade</td>
<td>Wednesday, 1 March 2023, 1:03 AM</td>
</tr>
<tr>
<td>4</td>
<td>Katy</td>
<td>Thackray</td>
<td>Friday, 3 March 2023, 2:24 PM</td>
<td>1</td>
<td>65.18 / 100.00</td>
<td>Automatic grade</td>
<td>Friday, 3 March 2023, 2:24 PM</td>
</tr>
<tr>
<td>5</td>
<td>Elizabeth</td>
<td>Ho</td>
<td>Sunday, 5 March 2023, 3:59 PM</td>
<td>2</td>
<td>64.98 / 100.00</td>
<td>Automatic grade</td>
<td>Sunday, 5 March 2023, 3:59 PM</td>
</tr>
<tr>
<td>6</td>
<td>Paul</td>
<td>DSouza</td>
<td>Sunday, 5 March 2023, 3:38 PM</td>
<td>5</td>
<td>0.00 / 100.00</td>
<td>Automatic grade</td>
<td>Sunday, 5 March 2023, 3:38 PM</td>
</tr>
<tr>
<td>7</td>
<td>George</td>
<td>Ogden</td>
<td>Saturday, 4 March 2023, 1:45 PM</td>
<td>5</td>
<td>0.00 / 100.00</td>
<td>Automatic grade</td>
<td>Sunday, 5 March 2023, 9:12 AM</td>
</tr>
</tbody>
</table>