```
#!/usr/bin/env python3
import numpy as np
from matplotlib import pyplot as plt
from scipy.cluster.hierarchy import dendrogram

# Provides agglomeration schedule manipulation methods for use in hierarchical
clustering analysis
# Called in HCA.py
# Copyleft Ed Egan, 2022.

def get_linkage_matrix(model):
    counts = np.zeros(model.children_.shape[0])
    n_samples = len(model.labels_)
    for i, merge in enumerate(model.children_):
        current_count = 0
        for child_idx in merge:
            if child_idx < n_samples:
                current_count += 1  # leaf node
            else:
                current_count += counts[child_idx - n_samples]
        counts[i] = current_count
    linkage_matrix = np.column_stack([model.children_, model.distances_,
                                      counts]).astype(float)
    return linkage_matrix

def build_schedule(left,right,distances):
    """Build the full schedule in layer-cluster format
        Takes the left and right (i and merge) vectors from the linkage_matrix
        Notes:
            The linkage matrix only includes cluster merges
                It does not include the "zero" layer where all points are in
                there individual locations
                Accordingly, the length of the schedule is len(left) + 1
            The layers are initially coded in agglomerative (not divisive)
            order
            The distances vector is shorter than the agglomeration schedule by
            1
                This is corrected by adding the "first zero distance" (there
                may be others)
        Returns a dict of arrays of cluster assignments, keyed by layer number
        from 0 to the number of agglomerations in the schedule
    """
    distances_with_zero=np.concatenate(([0],distances))
    index=np.arange(len(left)+1)
    new_grp_no = index + len(index)
    full_schedule={}
    for i in index:
        if i == 0:
            current=np.copy(index)
        else:
            current=np.copy(full_schedule[i-1])
            for j in index:
                if current[j]==left[i-1] or current[j]==right[i-1]:
                   current[j] = new_grp_no[i-1]
        full_schedule[i]=np.copy(current)
    return full_schedule, distances_with_zero

def print_schedule(schedule,distances=np.empty(0)):
    """Print the agglomeration schedule in layer-cluster format"""
    for layer in dict.keys(schedule):
        str_layer=map(str,schedule[layer])
        if distances.size !=0:
            print(distances[layer],"\t",layer,"\t","\t".join(str_layer))
        else:
```

```
            print(layer,"\t","\t".join(str_layer))

def abridge_schedule(schedule,distances_with_zero):
    """Abridge a layer-cluster format schedule, starting with the last
    distance zero layers.
        Notes:
            There is always a distance zero layer by construction in
            build_schedule
        Returns a new schedule and a new distance array (of the right
        dimension).
    """
    abridge_schedule={}
    full_layers=dict.keys(schedule)
    first_zeros=np.where(distances_with_zero==0)[0]
    last_zero=first_zeros[len(first_zeros)-1]
    abridge distances=distances_with_zero[last_zero:len(full_layers)] #For
    return later
    a=0
    for f in range(last_zero,len(full_layers)):
        abridge_schedule[a]=schedule[f]
        a+=1
    return abridge_schedule, abridge_distances

def recode_schedule(schedule,reverse=False):
    """Recode the clusters so that they go from 0 to n
        Note:
            The original lat/long indices are maintained, so that reference
            data can be joined by index.
            order = 1 is agglomerative and order = -1 is divisive
        Returns a new schedule
    """
    if type(reverse) != bool:
        raise ValueError('reverse must be True or False' % (reverse))
    layers={}
    index=np.arange(len(schedule[0]))
    max_layer_no=max(dict.keys(schedule))
    for layer_no in dict.keys(schedule):
        sorted_grp_no_index=np.argsort(schedule[layer_no])
        last_val=schedule[layer_no][sorted_grp_no_index[0]]
        cluster_no=0
        current_clusters=np.zeros(len(schedule[0]))
        for k in index:
            if schedule[layer_no][sorted_grp_no_index[k]] != last_val:
                cluster_no+=1
            current_clusters[sorted_grp_no_index[k]]=cluster_no
            last_val=schedule[layer_no][sorted_grp_no_index[k]]
        new_layer_no=layer_no
        if reverse:
            new_layer_no=max_layer_no-layer_no
        layers[new_layer_no]=current_clusters
    return layers

def
get_schedule(model,abridge=True,recode=True,reverse=True,returndists=False):
    """Build, abridge and recode the schedule, given a model
    Note:
        Returns a schedule
    """
    linkage_matrix=get_linkage_matrix(model)
    left=linkage matrix[:,0]
    right=linkage_matrix[:,1]
    distances=linkage_matrix[:,2]
    result_schedule,result_distances=build_schedule(left,right,distances)
    if abridge:
```

```
        result_schedule,
        result_distances=abridge_schedule(result_schedule,result_distances)
    if recode:
        result_schedule=recode_schedule(result_schedule,reverse)
    if returndists:
        return result_schedule,result_distances
    return result_schedule
```