Idea: in each iteration set taxonomic ID of an acession to its direct ancestor ID. If this ID is existent, the accession list associated with taxID will be merged with the one of its ancestor, else the taxID is replaced by its ancestor in the next iteration.

Algorithm 1 Sequential distribution of accessions into N equally filled bins.

procedure TAXONOMICBINNING(accList, acc2tax, π, B)
    bins $\leftarrow \emptyset$ ▷ Dictionary with keys representing clades and values accessions
    H $\leftarrow 4 \cdot \lceil \frac{|\text{accList}|}{B} \rceil$ ▷ Size threshold of a bin
    for acc in accList **do** ▷ Initial compression ensuring uniqueness of keys
        $\text{tax}_{acc} \leftarrow$ acc2tax[acc]
        if $\text{tax}_{acc} \notin$ bins.keys() **then**
            bins$[tax_{acc}] \leftarrow (1, [\text{acc}])$
        else
            bins$[tax_{acc}] \leftarrow$ bins$[tax_{acc}] + [(1, \text{acc})]$
    while $|\text{bins}| > B$ **do** ▷ Number of clades still larger than target
        $keys \leftarrow$ bins.keys().sort() ▷ Handle lower-level keys first
        climbers $\leftarrow \emptyset$ ▷ Key-value pairs to be merged with their parent
        blockedSet $\leftarrow \emptyset$ ▷ Set of nodes that are parent of another key
        for key $\in$keys **do**
            if key $\in$ blockedSet **then** ▷ Node has descendants to be merged into
                continue
            if π[key] $\in$keys **then** ▷ Prevent parent from being replaced
                blockedSet $\leftarrow \pi$[key]
            climbers $\leftarrow (\pi[key], key)$
        for parent_key, key $\in climbers$ **do**
            if key $\in$ blockedList **then**
                continue
            tax2accs$[parent_key] \leftarrow$ [tax2accs.pop(key)]
    return tax2accs