UPGMA with Priority Queues

CS181 Fall 2020
UPGMA Algorithm

1. Initialize: assign every sequence to its own cluster

2. Iterate: while multiple clusters remain:
   a. Find the two clusters with minimum distance
   b. Merge these clusters together
   c. Compute the distance between the new cluster and all other clusters
   d. Add a new node to the tree for the new cluster

3. Termination occurs when only 1 cluster remains
UPGMA Algorithm: Straightforward Runtime

1. Initialize: assign every sequence to its own cluster $\leftarrow O(n)$ sequences
2. Iterate: while multiple clusters remain: $\leftarrow O(n)$ iterations
   a. Find the two clusters with minimum distance $\leftarrow O(n^2)$ pairs of clusters to check
   b. Merge these clusters together $\leftarrow O(n)$ sequences to move to the new cluster
   c. Compute the distance between the new cluster and all other clusters $\leftarrow O(n)$ computations
   d. Add a new node to the tree for the new cluster $\leftarrow O(1)$ time to compute proper height
3. Termination occurs when only 1 cluster remains

Runtime is dominated by step 2a $\rightarrow O(n^3)$ time
Priority Queues

**Priority queue:** a data structure that stores (key, element) pairs—the key of each element determines its priority in the queue

Priority queues support the following operations:

- **insert(k,e):** insert element e with key k into the priority queue
- **min():** get the element with the smallest key from the priority queue
- **removeMin():** remove the element with the smallest key from the priority queue
Priority Queues as Heaps

Priority queues are often implemented as heaps

**Heap**: a complete binary tree where the key at every node is greater than or equal to the key of its parent

**Complete binary tree**: a binary tree in which every level is filled, except possibly the last level, which is filled from the left

Examples of heaps:
Priority Queues as Heaps: Insert

To insert an element into a heap, add it to the bottom of the heap and iteratively move the element upwards until its key is larger than its parent’s key.

Example:
Priority Queues as Heaps: Min and RemoveMin

The element with the smallest key is always at the root of a heap.

To remove this element, move the last element of the heap to the root, and iteratively move this element downwards by swapping it with its smaller child until its key is smaller than both of its children’s keys.

Example:
Priority Queues as Heaps: Operation Runtimes

A heap with \( n \) elements has height \( \log n \)

**insert**\((k,e)\) and **removeMin()** both require \( O(\log n) \) time because they need to move an element up or down the height of the tree

**min()** requires \( O(1) \) time because the minimum key is always at the root
Priority Queues as Heaps: Additional Functionality

If we store a locator $L$ for every node in a priority queue, then we can access any node in the priority queue in $O(1)$ time and remove any node from the priority queue in $O(\log n)$ time.

If we know all of the elements that will be inserted into the priority queue in advance, we can construct the priority queue bottom-up in $O(n)$ time.

This additional functionality is covered in more detail in CSCI 1570 (Design and Analysis of Algorithms).
UPGMA Algorithm with Priority Queues

The elements of the priority queue are pairs of clusters, and the keys of the priority queue are distances between those clusters

1. Initialize: assign every sequence to its own cluster and construct the initial priority queue
2. Iterate: while multiple clusters remain:
   a. Find the two clusters with minimum distance
   b. Merge these clusters together and remove all pairs containing the merged clusters from the priority queue
   c. Compute the distance between the new cluster and all other clusters and add all these pairs of clusters to the priority queue
   d. Add a new node to the tree for the new cluster
3. Termination occurs when only 1 cluster remains
UPGMA Algorithm: Improved Runtime

1. Initialize:
   a. Assign every sequence to its own cluster $\leftarrow O(n)$ sequences
   b. Construct the initial priority queue $\leftarrow O(n^2)$ time because we know all the elements

2. Iterate: while multiple clusters remain: $\leftarrow O(n)$ iterations
   a. Find the two clusters with minimum distance $\leftarrow O(1)$ time
   b. Merge these clusters together $\leftarrow O(n)$ sequences to move to the new cluster
   c. Remove all pairs containing the merged clusters from the priority queue $\leftarrow O(n \log n)$ time
      i. Note: each removal requires $O((\log n)^2) = O(2 \log n) = O(\log n)$ time
   d. Compute the distance between the new cluster and all other clusters $\leftarrow O(n)$ computations
   e. Add all these pairs of clusters to the priority queue $\leftarrow O(n \log n)$ time
   f. Add a new node to the tree for the new cluster $\leftarrow O(1)$ time to compute proper height

3. Termination occurs when only 1 cluster remains

Runtime is now dominated by 2c and 2e $\rightarrow O(n^2 \log n)$ time
Summary

The straightforward runtime of the UPGMA algorithm is $O(n^3)$

Priority queues are powerful data structures that can efficiently find the maximum or minimum of a large group of elements

Implementing the UPGMA algorithm with priority queues improves its runtime to $O(n^2 \log n)$