

DSC40B: Theoretical Foundations of Data Science II

Lecture 17: *Kruskal's algorithm for
MST, and hierarchical clustering*

Instructor: Yusu Wang

Previously

- ▶ Given a weighted undirected graph
 - ▶ Prim's algorithm to compute the minimum spanning tree (MST) in $\Theta((V + E) \lg V)$ time
 - ▶ It is a greedy algorithm
- ▶ Today:
 - ▶ Yet another greedy algorithm to compute MST, called Kruskal's algorithm
 - ▶ Relation to hierarchical clustering



Kruskal's Algorithm for MST



Recall the general greedy idea:

- ▶ **Input:**
 - ▶ a weighted undirected graph $G = (V, E)$, with $\omega: E \rightarrow R$
- ▶ **Output:**
 - ▶ the set of edges in a MST T of G
- ▶ A MST T is $V - 1$ number of edges that connect all nodes, with no cycle.
- ▶ Intuitively, we will choose “**safe**” edges greedily to incrementally build T
 - ▶ such that any time, the edges we choose will form a part of some MST



- ▶ Last time: Prim's algorithm

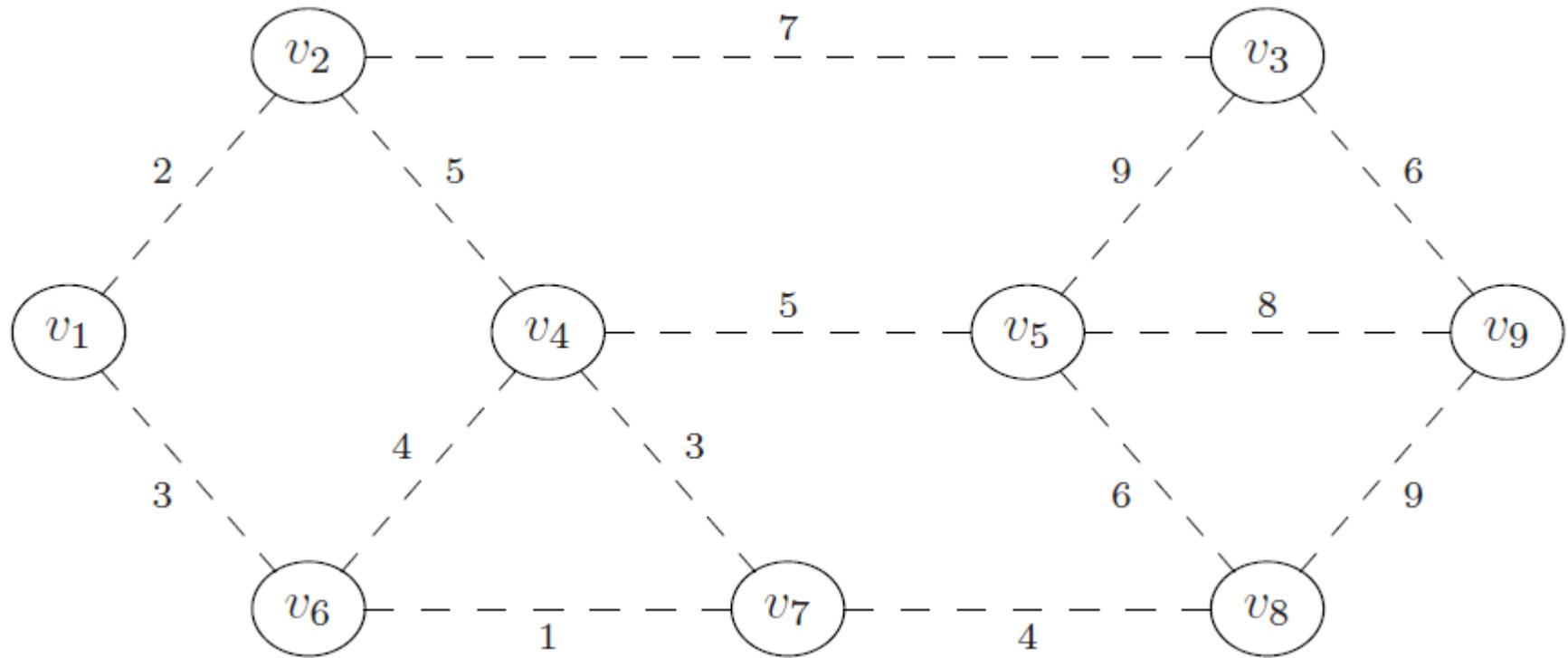
- ▶ Starting from any node, it starts to **grow a partial tree**, by repeatedly choosing **the minimum-weight edge** to reach an unvisited node, till it connects all graph nodes

- ▶ Today: Kruskal's algorithm

- ▶ We will choose the ``**safe**'' edges in a different order, and still grow the tree edge by edge
 - ▶ However, during the intermediate stages, what we have may not be a partial tree, could be disconnected.



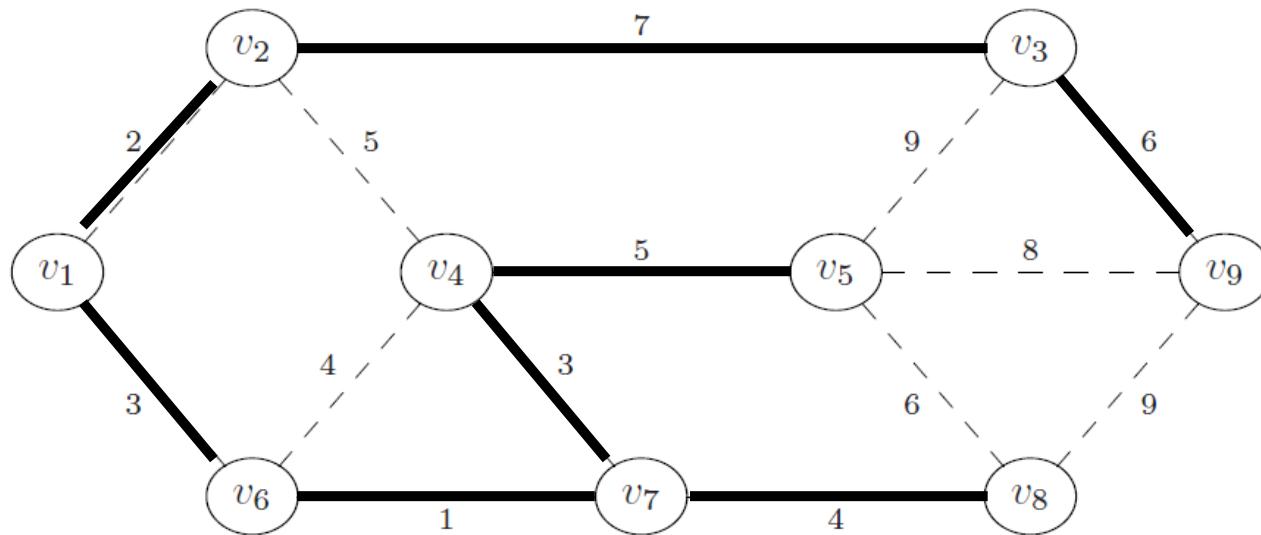
Example



- ▶ What is a “safe” edge to add first?
- ▶ What will be the next a “safe” edge to add?

Strategy of Kruskal

- ▶ We will add edges gradually in a greedy manner using smallest weights, while maintaining what we have so far does not have any cycle
 - ▶ add edges to tree in ascending order by weight
 - ▶ but if an edge creates a **cycle**, do not add it, and move on to next edge



Strategy

- ▶ We will add edges gradually in a greedy manner using smallest weights, while maintaining what we have so far does not have any cycle
 - ▶ add edges to tree in ascending order by weight
 - ▶ but if an edge creates a **cycle**, do not add it, and move on to next edge
- ▶ How do we check whether adding a new edge $e = (u, v)$ creates a **cycle or not?**
 - ▶ check whether nodes u, v are in the same component in the collection of edges we added so far



Kruskal's algorithm

```
def kruskal(graph, weights):
    mst = UndirectedGraph()

    # sort edges in ascending order by weight
    sorted_edges = sorted(graph.edges, key=weights)

    for (u, v) in sorted_edges:
        if not mst.in_same_component(u, v):
            mst.add_edge(u, v)

    return tree
```

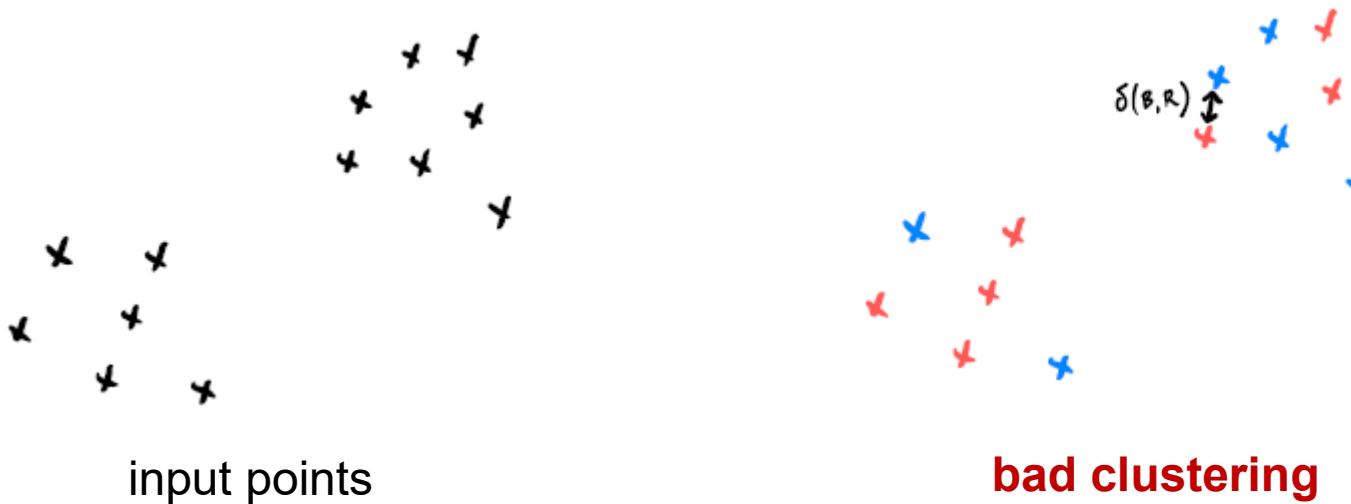
- ▶ We can maintain the components information of edges already collected, and perform `mst.in_same_component` using the **union-find** data structure
- ▶ The algorithm can be implemented in $\Theta((V + E) \lg V)$ time
 - ▶ which is $\Theta(E \lg V)$ time if input graph is connected

MSTs and (hierarchical) clustering



Clustering problem

- ▶ Identify the groups in data
- ▶ We frame clustering as a loss minimization problem
 - ▶ Input: n data points in R^d
 - ▶ Goal: assign each data point a color (red or blue, which is class labels) so that the distance between the closest pair of red and blue points is maximized



Clustering problem

- ▶ Identify the groups in data
- ▶ We frame clustering as a loss minimization problem
 - ▶ Input: n data points in R^d
 - ▶ Goal: assign each data point a color (red or blue, which is class labels) so that the distance between the closest pair of red and blue points is maximized



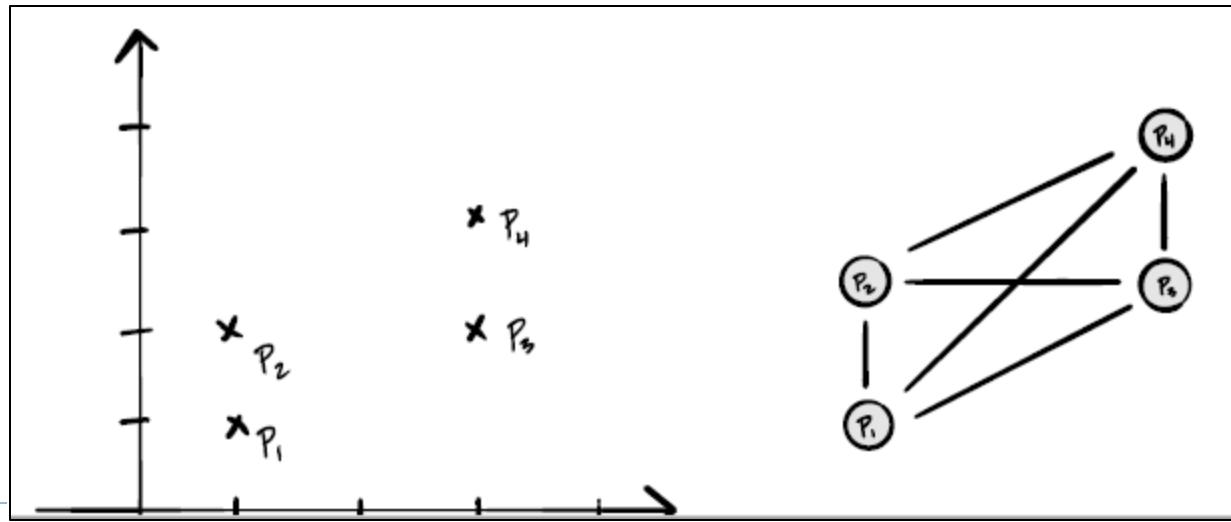
Recall the brute-force solution

- ▶ Try all possible assignments
 - ▶ there are 2^n possible assignments! (n is the number of input points)
 - ▶ highly not efficient!
- ▶ Instead, we will convert it to a graph problem



Distance graph

- ▶ Given n data points $V = \{p_1, p_2, \dots, p_n\}$
 - ▶ create a complete undirected graph $G = (V, E)$ such that for any $p_i \neq p_j$, there is an edge $(p_i, p_j) \in E$
 - ▶ the weight of an edge (p_i, p_j) is $\omega(p_i, p_j) = dist(p_i, p_j)$
- ▶ We call this resulting weighted graph the **distance graph** spanned by V



Single linkage clustering

- ▶ Given n data points $V = \{p_1, p_2, \dots, p_n\}$
- ▶ Create distance graph G
- ▶ Run Kruskal's Algorithm to compute MST of G, T :



- ▶ Delete largest edge in MST, and we obtain two components => clusters!
- ▶ In general, if we remove largest $k - 1$ edges in MST
 - ▶ then we obtain k clusters (components)

Single linkage clustering

- ▶ Alternatively, we can perform Kruskal's algorithm, adding edges in ascending order by weights without forming cycles, and stop till we have a target k number of components (clusters)
 - ▶ That is, we terminate early in the Kruskal's algorithm
- ▶ Time complexity
 - ▶ $\Theta(E \lg V) = \Theta(V^2 \lg V)$ as $E = \Theta(V^2)$ in this case
- ▶ This is called the **single-linkage clustering** algorithm
 - ▶ One of the most well-known clustering algorithm.
 - ▶ It is popular, although it suffers from the so-called **chaining-effect** in practice.
 - ▶ Variants of it, e.g., average-linkage clustering algorithm and complete-linkage clustering algorithm, are popular as a simple clustering algorithm.



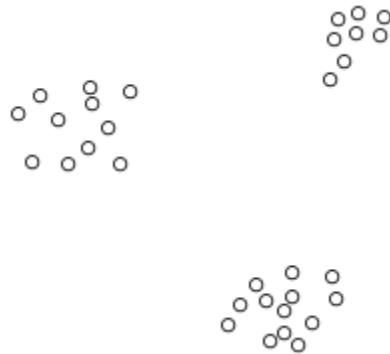
Example of chaining effect



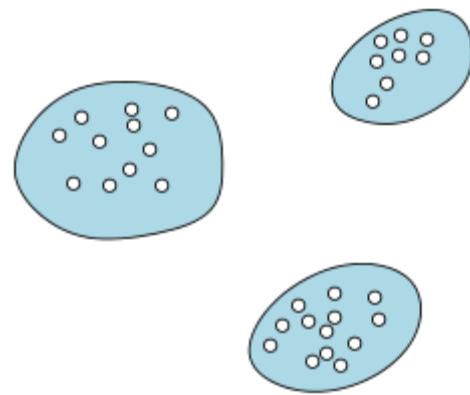
Hierarchical clustering



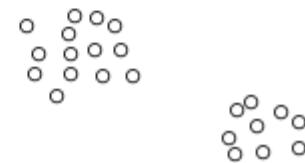
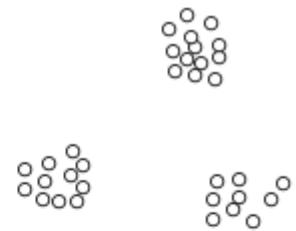
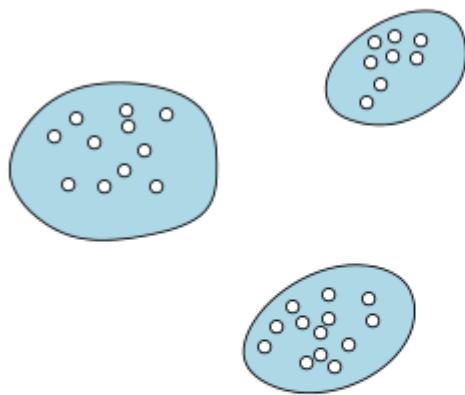
Clustering



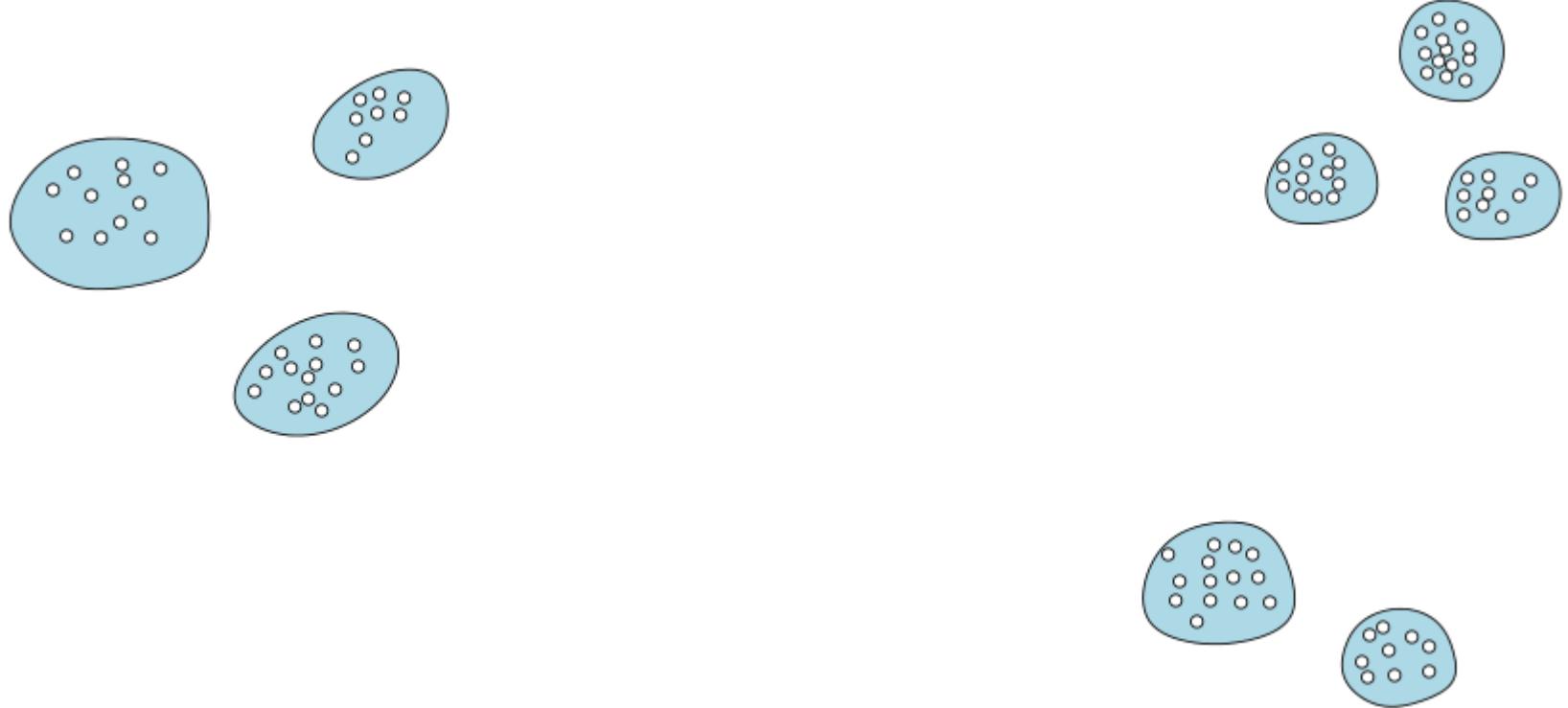
Clustering



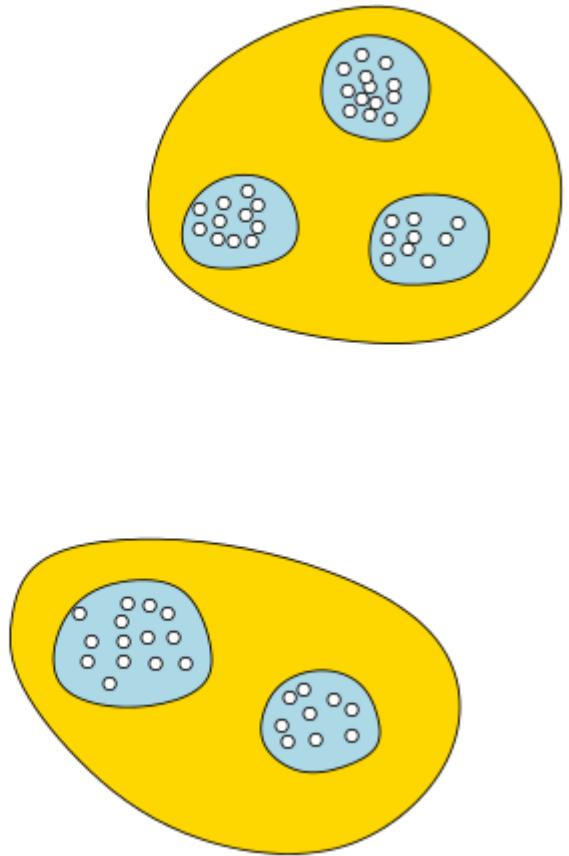
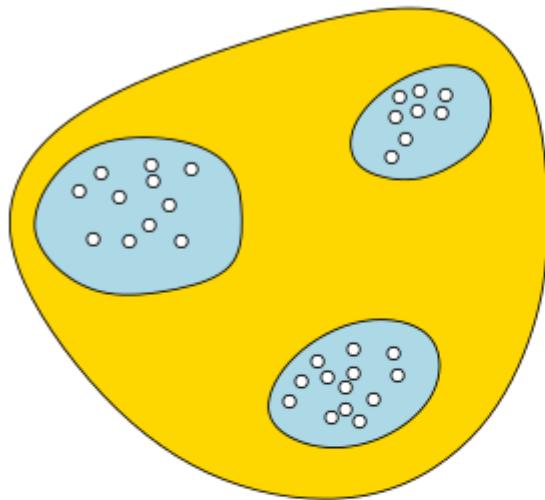
Hierarchical Clustering



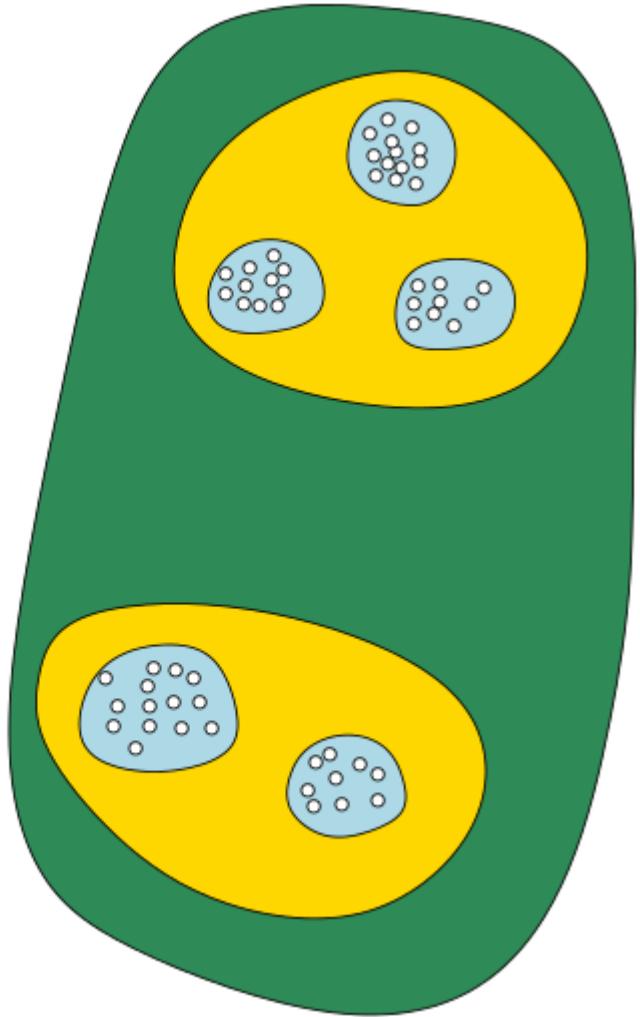
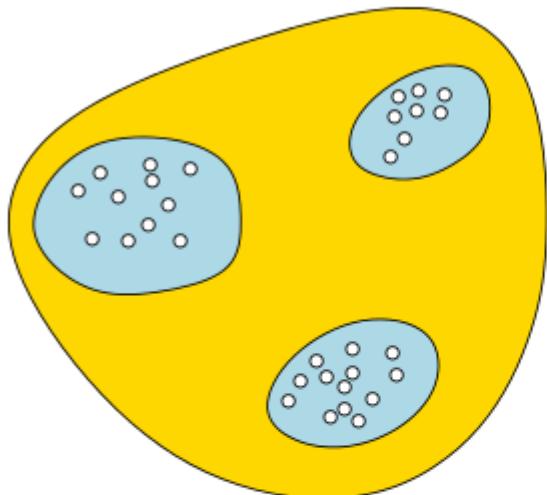
Hierarchical Clustering



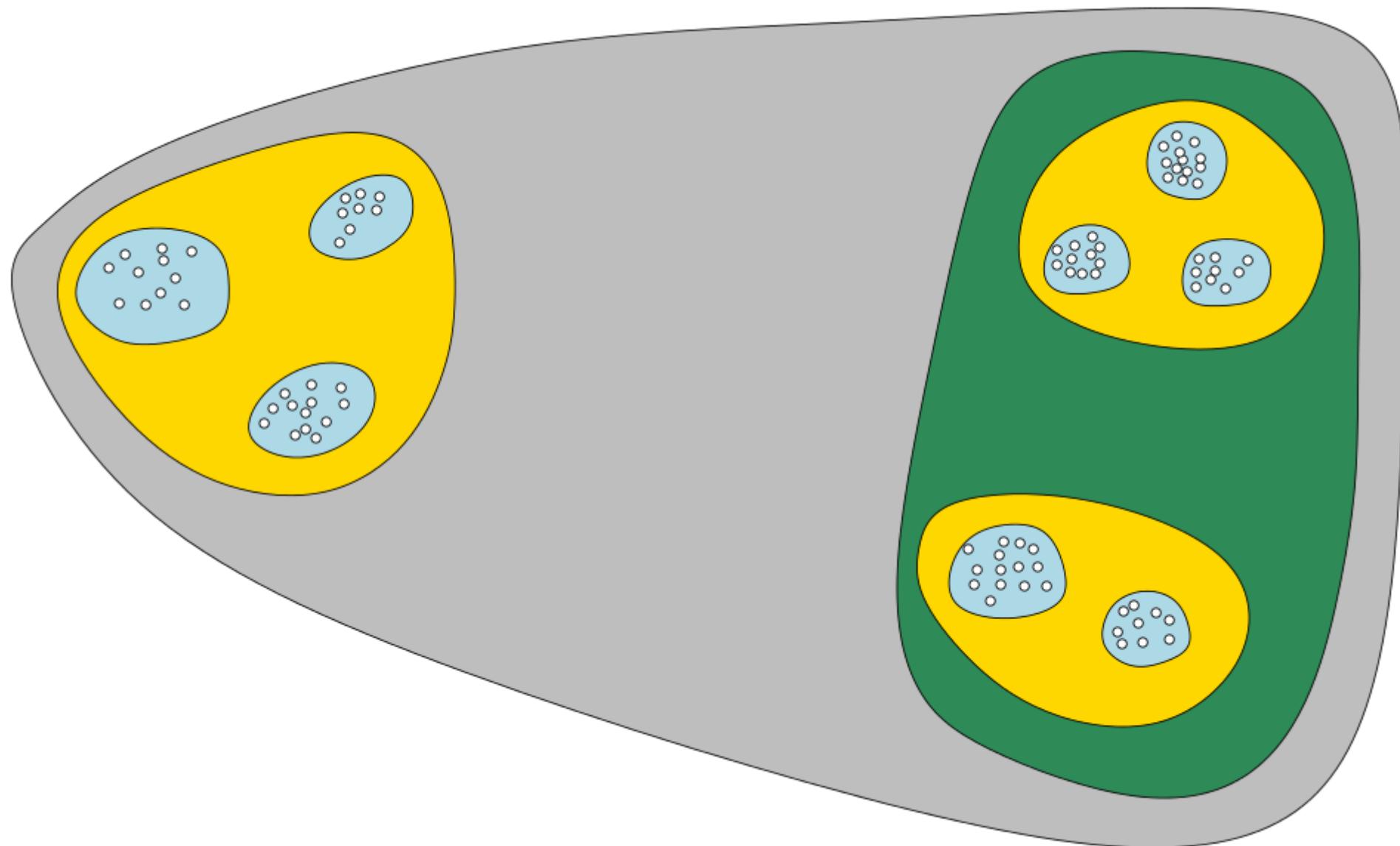
Hierarchical Clustering



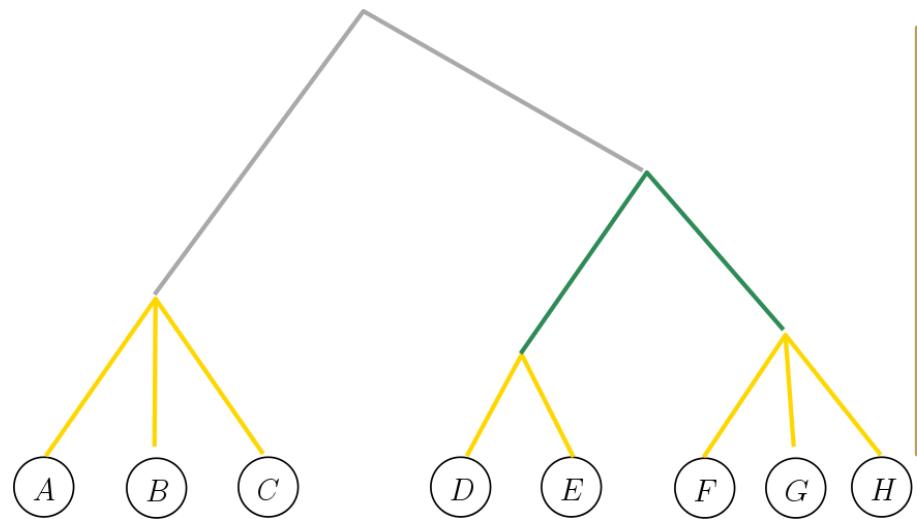
Hierarchical Clustering



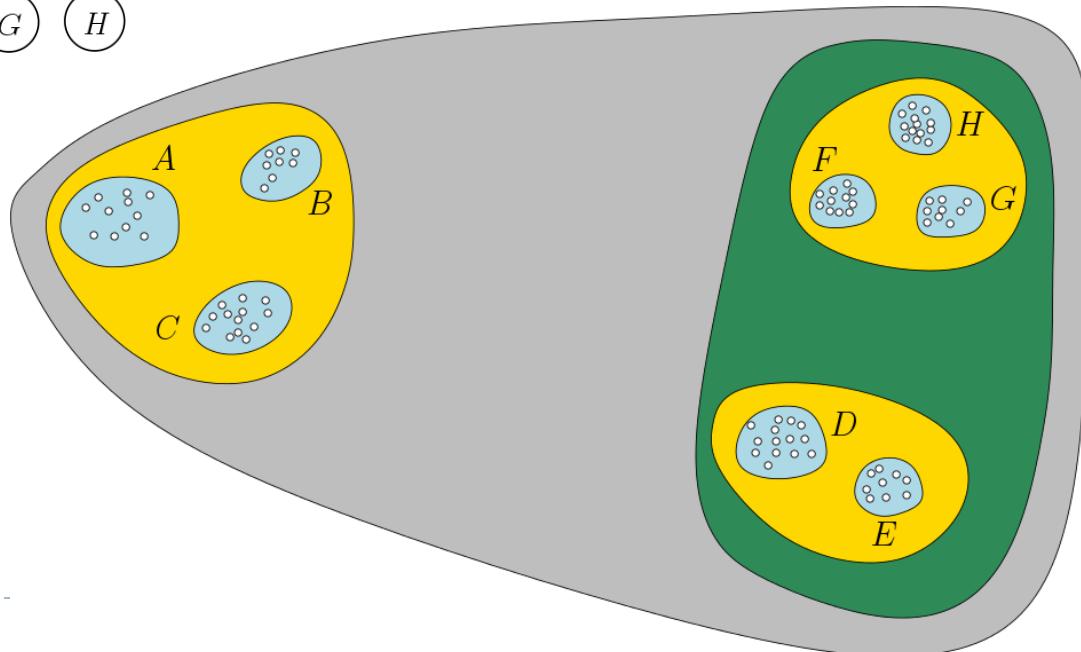
Hierarchical Clustering



Hierarchical Clustering Tree (HCT)



- Each internal tree node indicates a cluster, containing all leaves in subtree
- Ancestor/descendent indicates containment relation
- Height at each tree node corresponds to certain *cost* of the corresponding cluster (e.g, tightness of cluster)



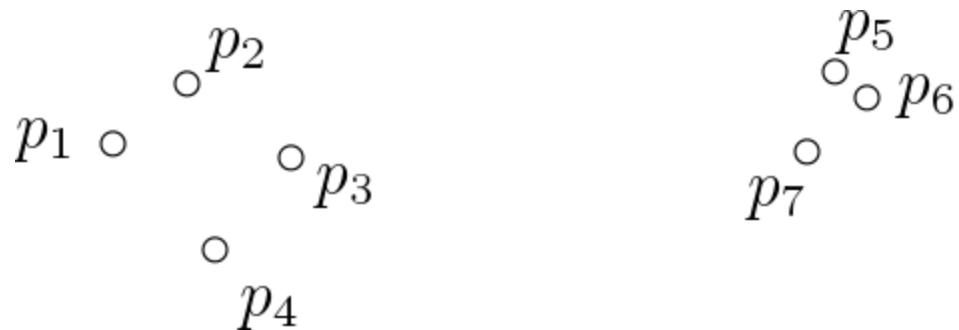
Single Linkage Clustering

- ▶ One of the family of **agglomerative clustering** methods
- ▶ Input: A discrete n-point metric (P, d_P)
- ▶ Output: A hierarchical clustering tree T , with points in P being leaves
- ▶ Starting with each data point as a single cluster
- ▶ Keep merging clusters based on nearest distance between points from their members



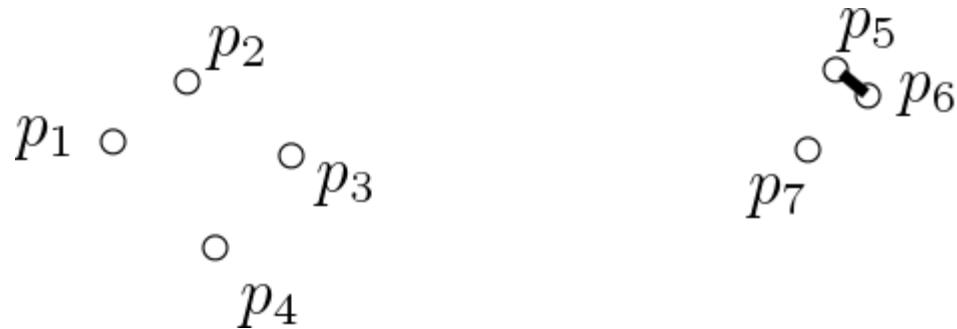
Single Linkage Clustering

- ▶ Starting with each data point as a single cluster
- ▶ Keep merging clusters based on nearest distance between points from their members



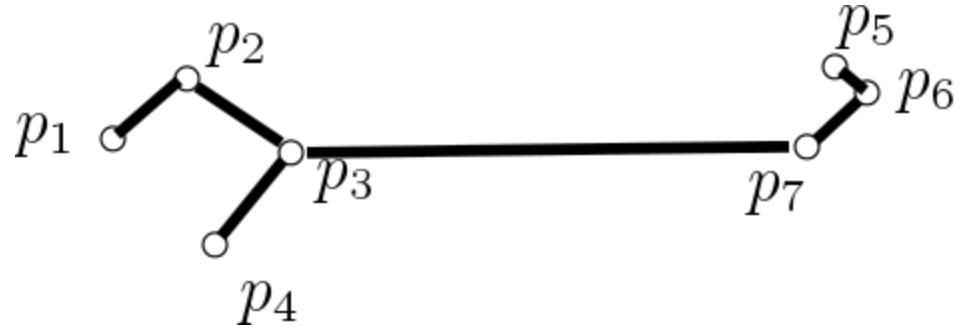
Single Linkage Clustering

- ▶ Starting with each data point as a single cluster
- ▶ Keep merging clusters based on nearest distance between points from their members



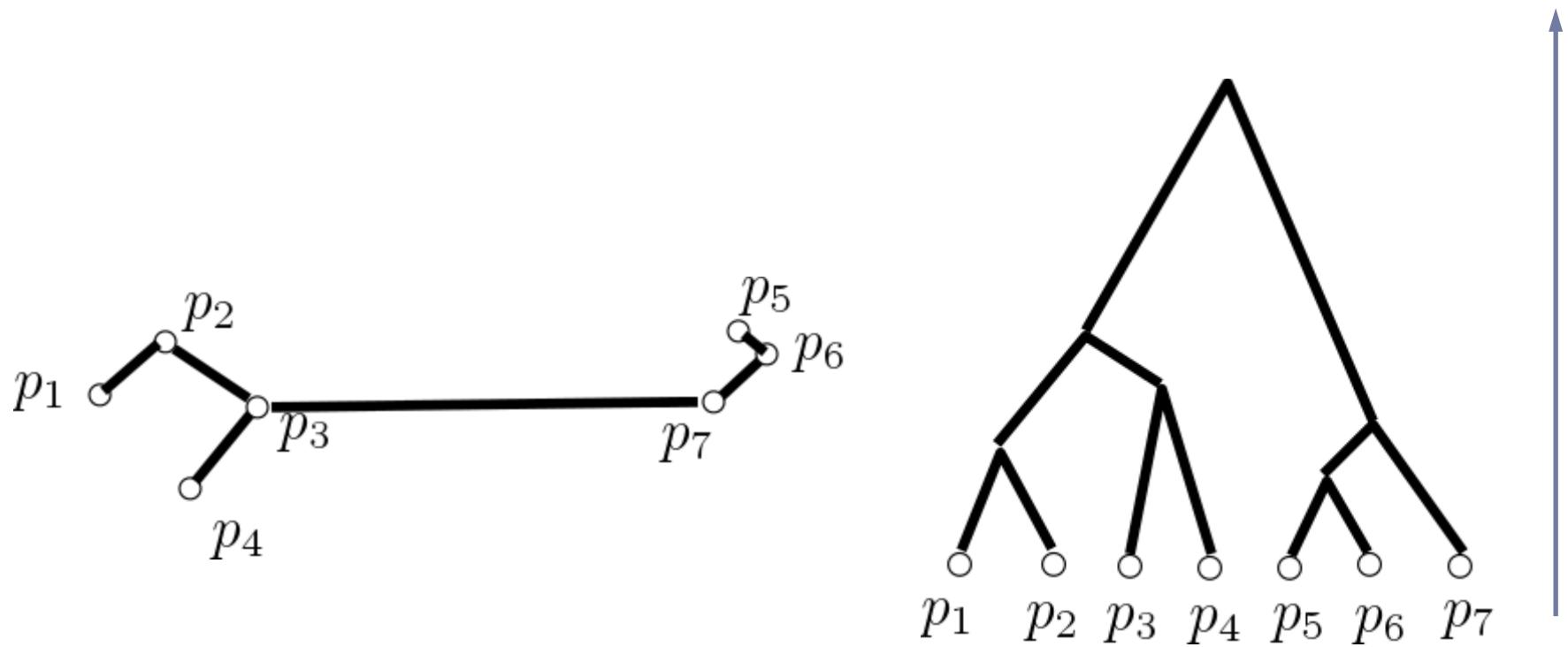
Single Linkage Clustering

- ▶ Starting with each data point as a single cluster
- ▶ Keep merging clusters based on nearest distance between points from their members



Single Linkage Clustering

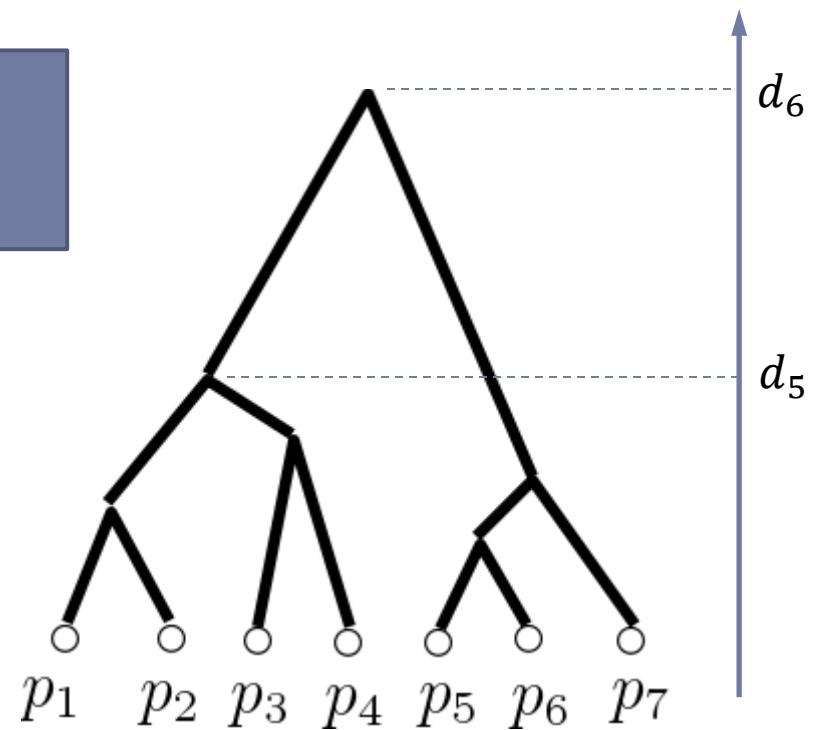
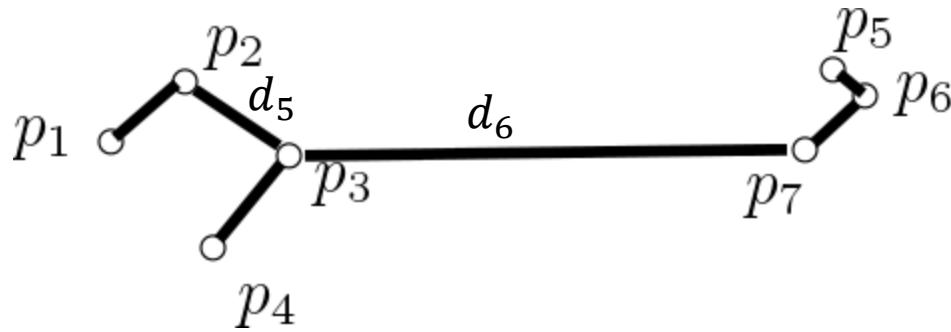
- ▶ Starting with each data point as a single cluster
- ▶ Keep merging clusters based on nearest distance between points from their members



One Example: Single Linkage Clustering

- ▶ Starting with each data point as a single cluster
- ▶ Keep merging clusters based on nearest distance between points from their members

This procedure is exactly Kruskal's algorithm!



Summary

- ▶ MST for weighted undirected graphs
 - ▶ Prim's algorithm
 - ▶ Kruskal's algorithm
 - ▶ Both $\Theta((V + E) \lg V)$ (which is $\Theta(E \lg V)$ for connected graphs)
- ▶ Kruskal's algorithm can be used to produce single linkage clustering



FIN

