# Data Structures

November 9

# Graphs

# Objectives

Discuss the following topics:

- Minimum Spanning Trees (Kruskal's and Prim's algorithms)
- Union-find data structure
- Priority queues for implementation of Prim's algorithm
- **Clustering**
- **Data Compression and Huffman Codes**
- Shortest Paths

# Wrap Up of Kruskal and Union Find

# Wrap Up of Prim and Priority Queue

# Eliminating the assumption of mutually unequal weights in the MST algo's

- Suppose we have a graph in which it happens that some edges have the *same* weight
- What we can do add extremely small numbers to edges of equal weight so that they will have different weight
- ((Example: suppose you have three edges with weight 1 and all other edges have mutually different weights. Let m be the minimum absolute difference between unequal weights (suppose 1,1,1, 3, 4.5, 7, 10; then minimum: is 1.5 among pairs of unequal weight). 1,

$1+2^{-L}1.5$, $2^{-(L+1)}1.5$, 3, 4.5, 7, and 10 are the new weights for extremely large number L. ))

Denote the old weight function by weight(..) and the new by

$weight_P(..)$. We now find MST T with respect to the perturbed weight function $weight_P(..)$. Now suppose T' is a spanning tree and suppose with respect to the old weight function weight(T) > weight(T') but then it **cannot** be $weight_P(T) \leq weight_P(T')$ for a small enough perturbation. In other words T is not an MST wrt to the $weight_P(..)$ function, contradiction. Thus T is MST also with respect to old weight function.

# Clustering: an application of Kruskal's MST algo

- Clustering: organizing a collection of objects into coherent groups
- Examples of collections you want to organize: photographs, artwork, documents, microorganisms etc
- First task: find / construct a measure of how similar or dissimilar each pair of objects is
- One common approach: define ***distance function*** (***dissimilarity function*** would be a better name which is also used by the way)

# Clustering

- Objects at larger distance from one another are less similar to each other

# Clustering

- Distance often a more abstract meaning than physical distance
  - Distance between species number of years since they diverged in the course of evolution
  - Distance between images in a video stream: the number of corresponding pixels at which their intensity differ by at least some threshold.

# Clustering

- Clustering Problem: *given* a distance function on the objects, divide the objects into groups so that, intuitively, objects within the same group are “close”, and objects in different groups are “far apart.”
- Vague set of goals;  has given rise to a vast number of different  approaches

# Clustering

- **Clusterings of Maximum Spacing**
- Let U be a set of n objects (labeled as $p_1$, …, $p_n$).
- For each pair $(p_i,p_j)$ a numerical distance is given $d(p_i,p_j)$ -- sometimes also called distance matrix (dissimilarity matrix)
- Requirements:
  - $d(p_i,p_i) = 0$
  - $p_i \neq p_j \Rightarrow d(p_i,p_j) > 0$
  - symmetry: $d(p_i,p_j) = d(p_j,p_i)$

# Clustering

- Recall that a ***partition*** of a nonempty set D is a set of subsets $S_i \subseteq D$, $i \in J$ such that for
  - all $i,j \in J$, $i \neq j$, $[S_i \cap S_j = \emptyset]$ and
  - $\cup_{i \in J} S_i = D$, moreover
  - for all $i \in J$ $[S_i \neq \emptyset]$
- Problem divide the objects in U into k groups (k is a given parameter). (The division into groups should have *maximum spacing* (see below).)
- Definition: a ***k-clustering*** is any partition of U into k nonempty sets $C_1, \ldots, C_k$.
- $\forall \exists \ni \& \% \simeq <> \infty \leftrightarrow \leftarrow \uparrow \rightarrow \downarrow \aleph \Im \Re \wp \otimes \oplus \emptyset \cap \cup \sqsupset \neg$

# Clustering

- Definition: a ***k-clustering*** is any partition of U into k nonempty sets $C_1$, …, $C_k$. The $C_i$'s are also referred to as clusters.
- Definition: ***spacing of a k-clustering*** is the minimum distance between any pair of points lying in different clusters.
- Alternatively: one can define the distance between two subsets $S_1$ and $S_2$ of U as follows: $d(S_1,S_2) := \min_{(e,f) \in S_1 x S_2} d(e,f)$. The spacing of a k-clustering can then be defined as **$\text{spacing}(C_1, \ldots, C_k) := \min_{(i,j) \in J x J \text{ and } i \neq j} d(C_i, C_j)$**, where J={1,…, k}

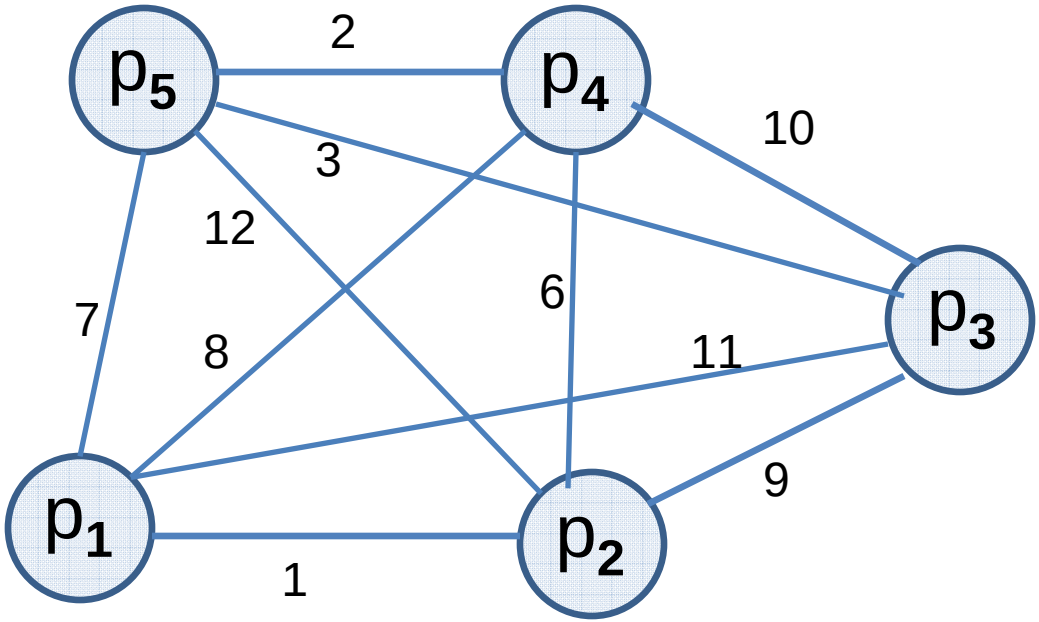
# Hierarchical agglomerative clustering: single-link clustering: the algo

- Bring close by points into a common cluster as rapidly as possible: “this way different clusters will not be close”;
- Draw an edge between closest pair of points, then draw an edge between the next pair of closest points, keep doing this in order of increasing distance $d(p_j,p_i)$.
- We are actually growing a graph H on U edge by edge

# Hierarchical agglomerative clustering: single-link clustering: the algo

- Note: need only connected components, not the full set of edges: *if we are about to add edge ($p_j$,$p_i$) and find that $p_j$ and $p_i$ already belong to the same cluster, we will refrain from adding this edge, it does not change the set of connected components.* Note that it does not hurt though to add edges ($p_j$,$p_i$) with $p_j$ and $p_i$ in the same cluster: refraining opens up the way to fall back on a modified version of Kruskal's MST algorithm!
- Hence, we never create a cycle; H will be a forest ( a union of trees)

# Single-link clustering algo

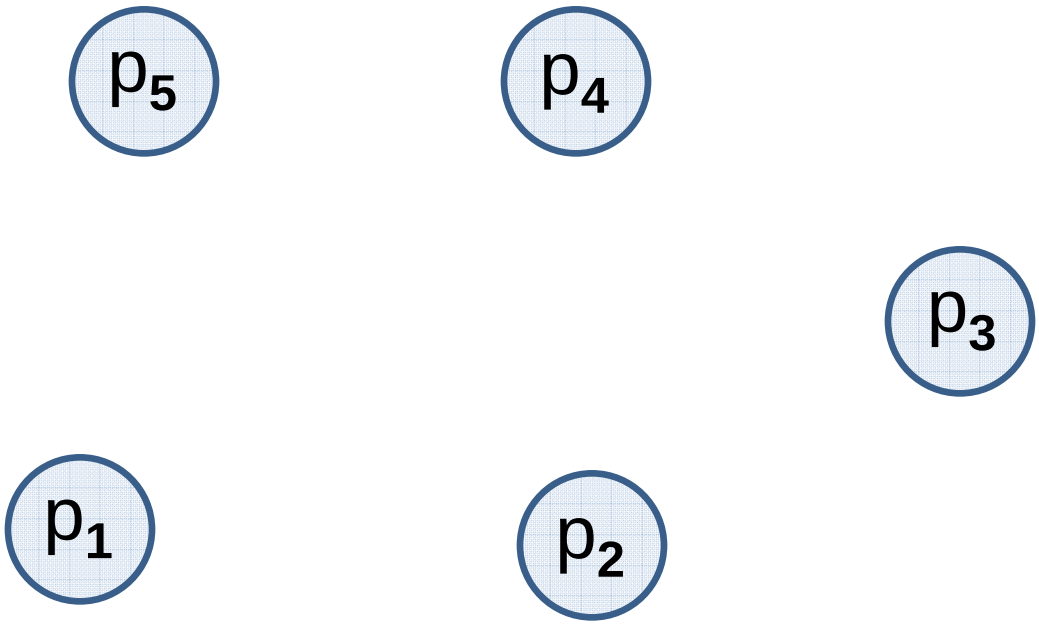



Graph of Distances

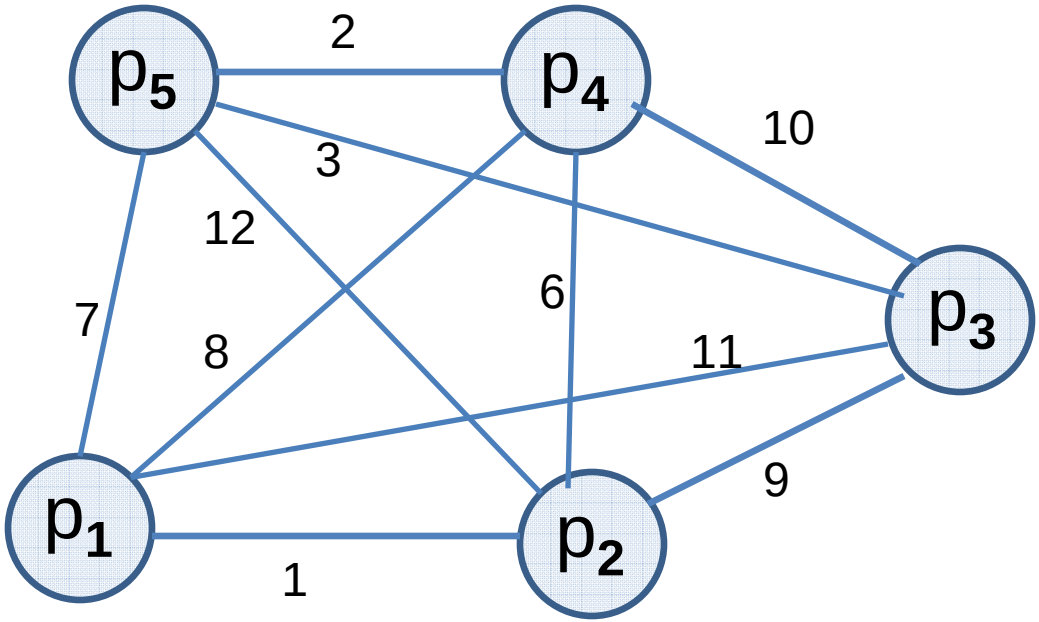
We run the algorithm with
**k = 3**

Initialization:
5 disconnected
Components
Keep track of number
Connected component

**numOfConnectedComp ← 5**



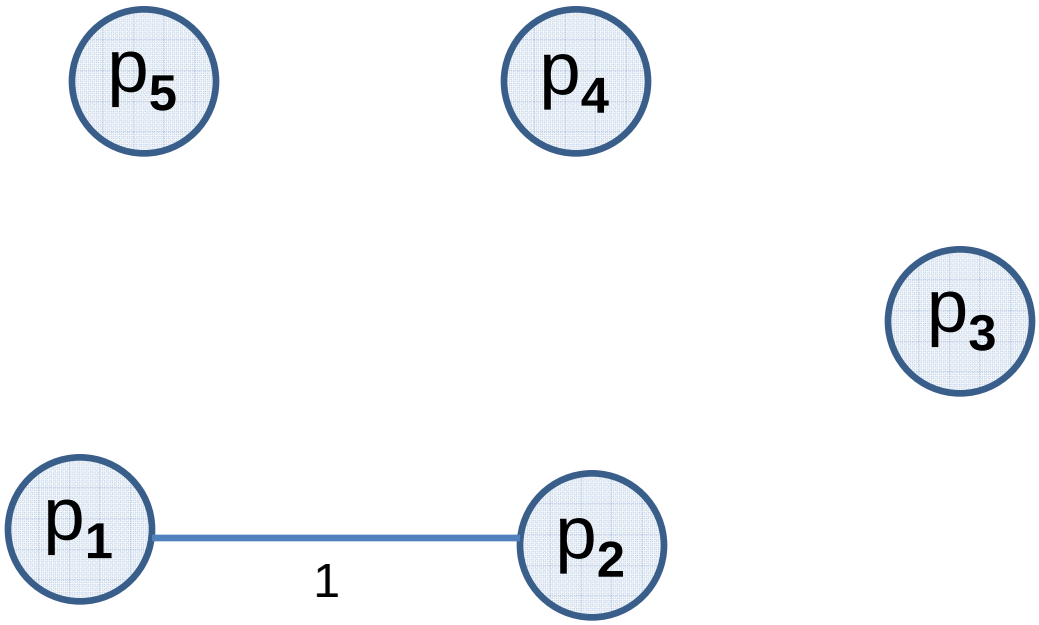
# Single-link clustering algo



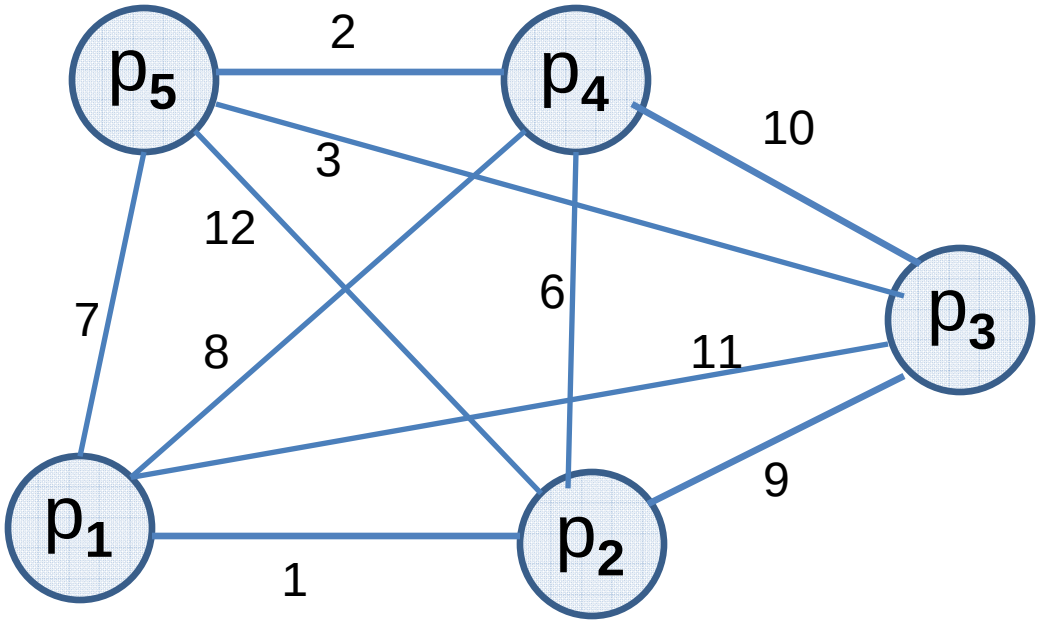

Graph of Distances

**Result of step 1**

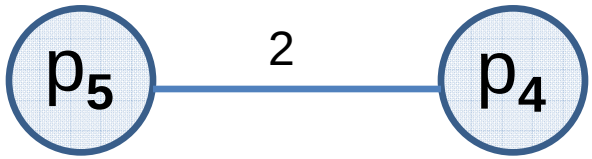
**numOfConnectedComp ← 4**

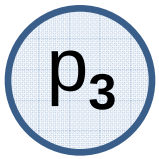


# Single-link clustering algo




Graph of Distances





**Result of steps 1 and 2**

**numOfConnectedComp ← 3**

**Can stop number of Connected components is 3**

# Hierarchical agglomerative clustering: single-link clustering: the algo

- The graph growing procedure we are using is nothing else than Kruskal's MST algo for a graph G on U in which there is a weight edge $d(p_j,p_i)$ between each pair of nodes $(p_j,p_i)$.
- The only difference: we seek k-clustering, so we can stop, once we obtain k connected components.

# Hierarchical agglomerative clustering: single-link clustering: the algo

- *In other words:* we are running Kruskal's algorithm but stopping it just before it adds its last k-1 edges
- So we have another equivalent way of getting the k-cluster: take the full MST T (as Kruskal would have produced it), delete the k-1 most expensive edges (these are the ones the single-link clustering algo never adds in the first place). Define the resulting connected components $C_1$, …, $C_k$ as the k-cluster.

# Hierarchical agglomerative clustering: single-link clustering: the algo

- In summary:
- Iteratively merging clusters is equivalent to computing an MST (=minimum spanning tree) and deleting the most expensive edges.

# single-link clustering: correctness of the algo

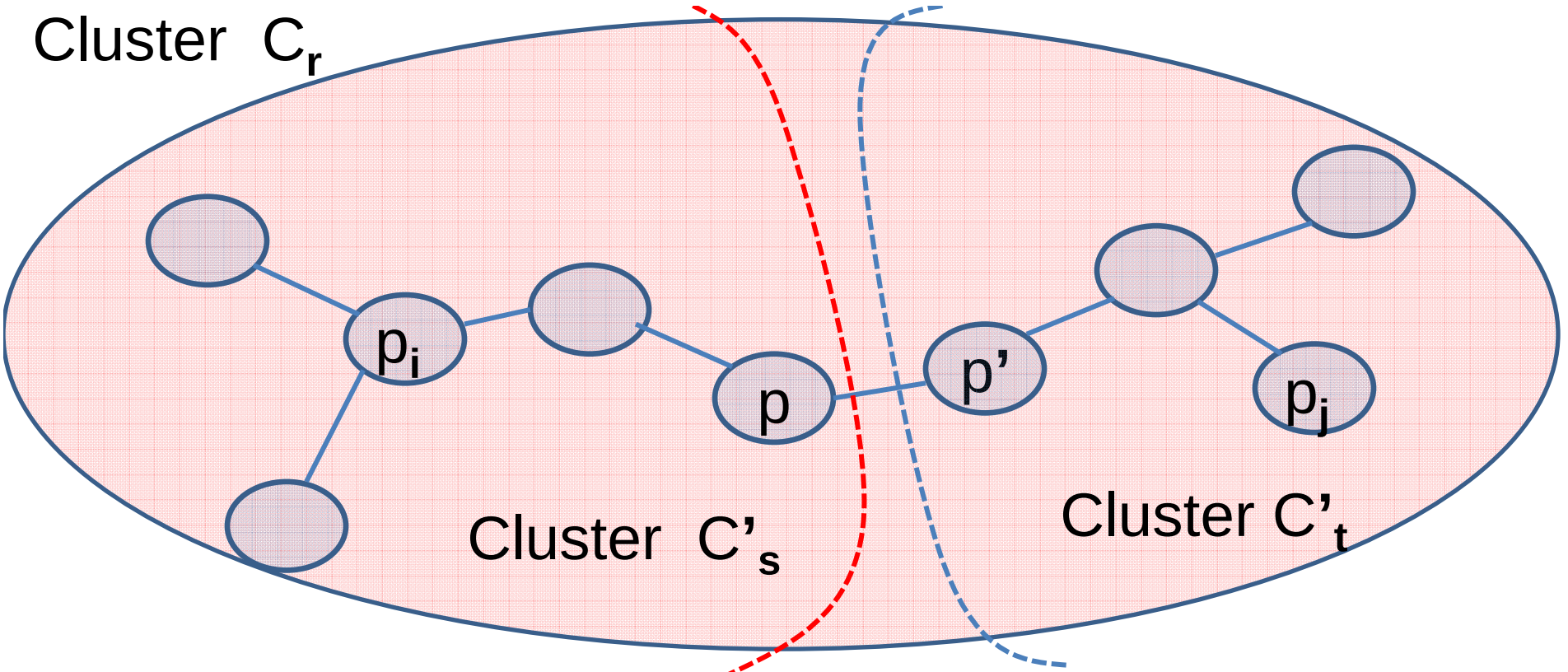
The components $C_1$, …, $C_k$ formed by deleting the k-1 most expensive edges of the minimum spanning tree T constitute a k-clustering of maximum spacing.

Proof. Let **C** denote the clustering $C_1$, …, $C_k$ produced by our clustering algorithm. The spacing of **C** is precisely the length d* of the (k-1)-st most expensive edge that Kruskal's algo would have added next, at the moment we stopped it.

# single-link clustering: correctness of the algo

Let **C'** be another k-clustering, which partitions U into nonemtpy sets $C'_1, \ldots, C'_k$. Must show: spacing of **C'** is at most d*.

The two clusterings **C** and **C'** are different. So there is a cluster $C_r$ in **C** which is not a subset of any of the k sets $C'_s$ in **C'**. (Remember **C** and **C'** are partitions both of which contain precisely k sets.) Hence, there are points $p_i$ and $p_j$ in $C_r$ that belong to different clusters in **C'** -- say $p_i \in C'_s$ and $p_j \in C'_t$ with $C'_s \neq C'_t$.




$p_i$ and $p_j$ in same component $C_r$; Kruskal added all the edges of a $p_i$ – $p_j$ path P in $C_r$ before it stopped. Thus each edge on this path has length at most d*. As said before $p_i \in C'_s$ and $p_j \in C'_t$ with $C'_s \neq C'_t$. Hence $p_i \in C'_s$ and $p_j \notin C'_s$; so let p' be node on P that does not belong to $C'_s$ and let p on P be the node that just comes before p'. We know d(p,p') ≤ d*, since this edge (p,p') was added by Kruskal. But p and p' belong to different sets of **C'**. Hence, spacing of **C'** is at most d(p,p') (**≤ d***).

# single-link clustering, algo

- Implementation discussion
- see also: http://nlp.stanford.edu/IR-book/html/htmledition/hierarchical-clustering-1.html also for other clustering approaches (not just hierarchical)