Huffman Codes and Data Compression

Data Compression

- Given a text that uses 32 symbols (26 different letters, space, and some punctuation characters), how can we encode this text in bits?
  - » We can encode 2^5 different symbols using a fixed length of 5 bits per symbol. This is called fixed length encoding.
- Some symbols (e, t, a, o, i, n) are used far more often than others. How can we use this to reduce our encoding?
  - » Encode these characters with fewer bits, and the others with more bits.

Data Compression

- How do we know when the next symbol begins?
  - » Use a separation symbol (like the pause in Morse), or make sure that there is no ambiguity by ensuring that no code is a prefix of another one.
- Ex. c(a) = 01

  c(b) = 010

  c(e) = 1

  What is 0101?

Prefix Codes

- **Definition.** A prefix code for a set S is a function c that maps each $x \in S$ to 1s and 0s in such a way that for $x, y \in S$, $x \neq y$, $c(x)$ is not a prefix of $c(y)$.
- Ex. $c(a) = 11$

  $c(e) = 01$

  $c(k) = 001$

  $c(l) = 10$

  $c(u) = 000$
- Q. What is the meaning of 1001000001 ?

Prefix Codes

- Q. What is the meaning of 1001000001 ?
- A. “leuk”
- Suppose frequencies are known in a text of 1k:
  $f_a = 0.4$, $f_e = 0.2$, $f_k = 0.2$, $f_l = 0.1$, $f_u = 0.1$
- Q. What is the size of the encoded text?
- A. $2{*}f_a + 2{*}f_e + 3{*}f_k + 2{*}f_l + 3{*}f_u = 2.3\text{k}$

- Length of the encoding $= \sum_{x \in S} n\, f_x |c(x)|$
  $= \text{n} \sum_{x \in S} f_x |c(x)|$

Optimal Prefix Codes

- **Definition.** The average bits per letter of a prefix code c is the sum over all symbols of its frequency times the number of bits of its encoding:

$$ABL(c) = \sum_{x \in S} f_x |c(x)|$$

- We would like to find a prefix code that has the lowest possible value of average bits per letter.
- Suppose we model a code in a binary tree…

Representing Prefix Codes using Binary Trees

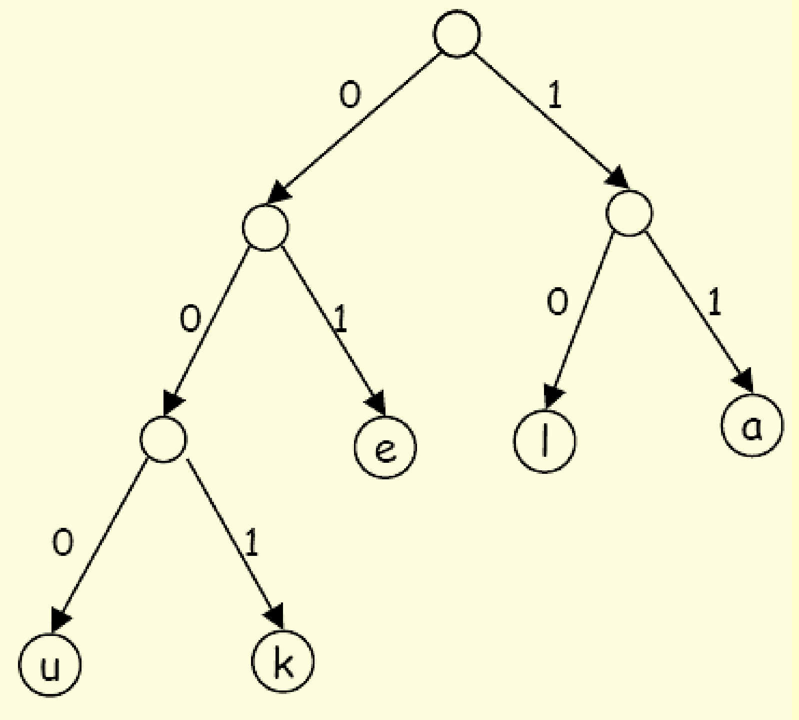
- Ex. $c(a) = 11$

  $c(e) = 01$

  $c(k) = 001$

  $c(l) = 10$

  $c(u) = 000$

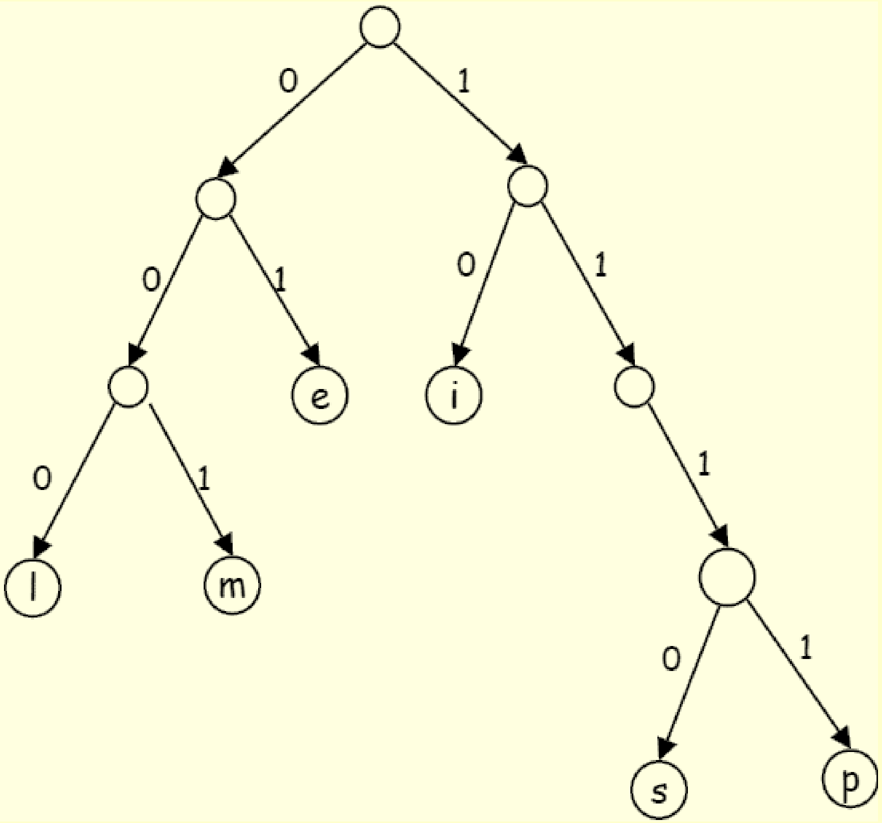


- Only the leaves are labelled with a symbol/letter.
- $c(x)$ = bit string along the path from root to x, $\forall x \in S$.
- Edges to the left child are labelled with 0 and to the right child are marked with 1.
- An encoding of x is a prefix of an encoding of y if and only if the path of x is a prefix of the path of y.

Representing Prefix Codes using Binary Trees

- What is the meaning of 111010001111101000 ?

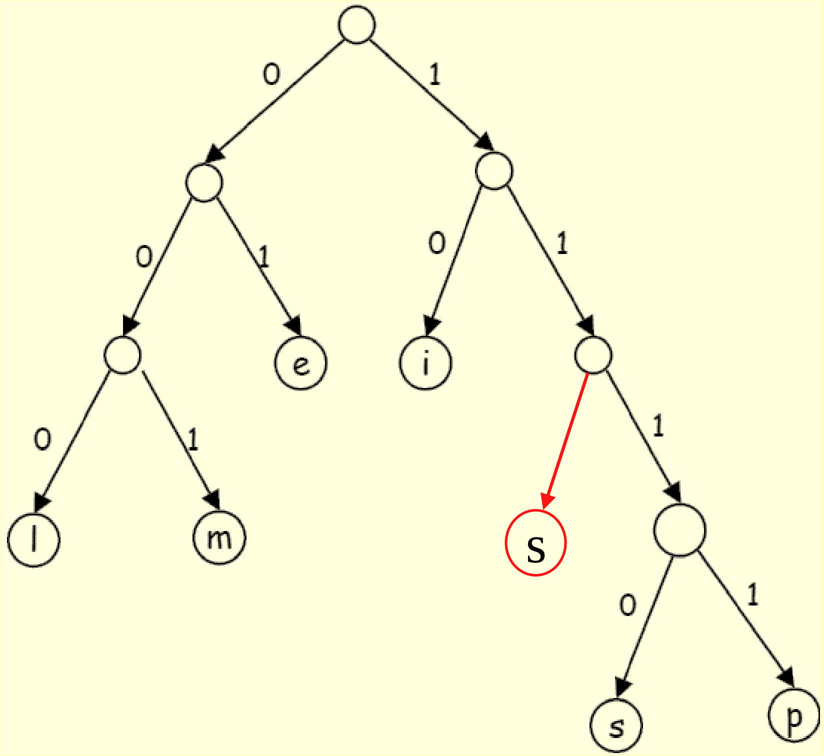
$$ABL(T) = \sum_{x \in S} f_x \cdot \text{depth}_T(x)$$



Representing Prefix Codes using Binary Trees

- What is the meaning of 111010001111101000 ?
  - » "simpel"

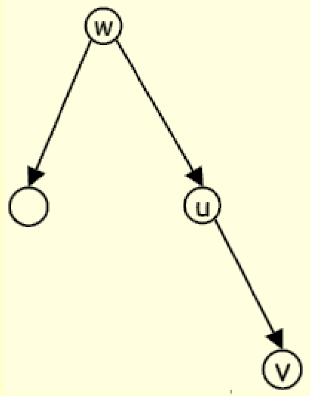
$$ABL(T) = \sum_{x \in S} f_x \cdot \text{depth}_T(x)$$



- Q. How can this prefix code be made more efficient?
  - » Change encoding of p and s to a shorter one.
  - » This tree is now full.
- Definition. A tree is full if every node that is not a leaf has two children.

Representing Prefix Codes using Binary Trees

- Claim. The binary tree corresponding to the optimal prefix code is full.
- **Pf.** (by contradiction)
- Suppose T is binary tree of optimal prefix code and is not full.
- This means there is a node u with only one child v.
- **Case 1:** u is the root; delete u and use v as the root
- **Case 2:** u is not the root
  - » let w be the parent of u
  - » delete u and make v be a child of w in place of u

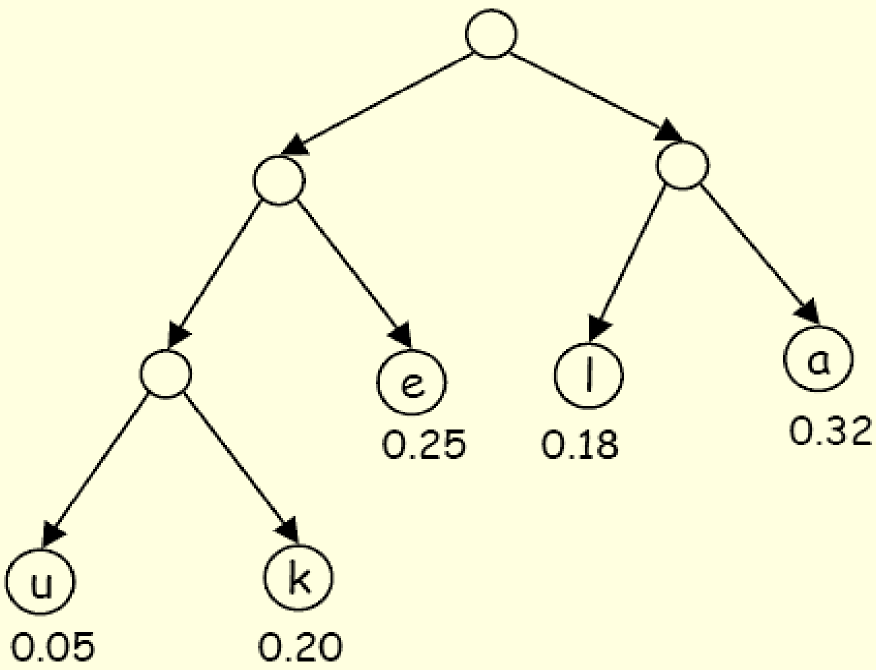


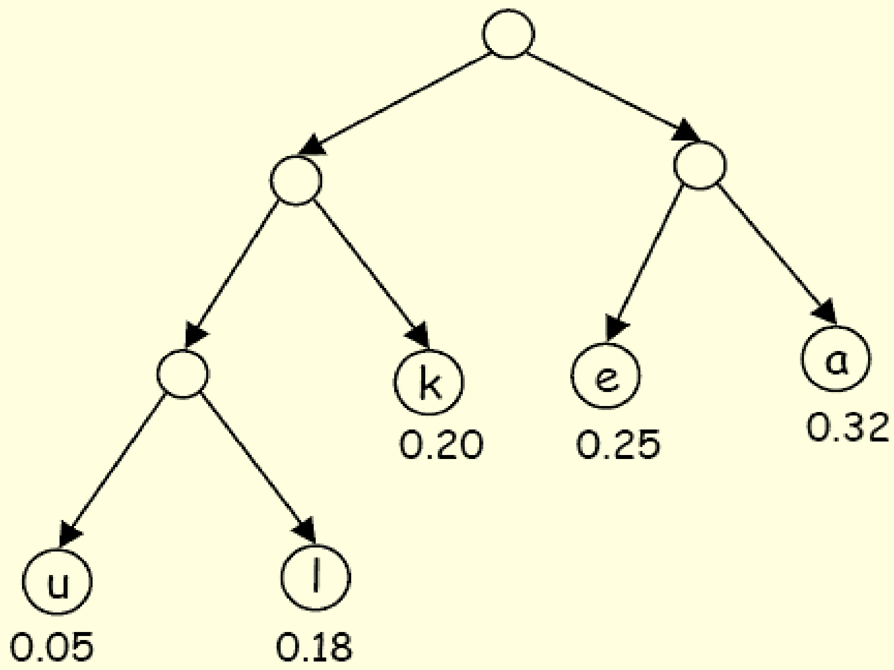
- In both cases the number of bits needed to encode any leaf in the subtree of v is decreased. The rest of the tree is not affected.
- Clearly this new tree T' has a smaller ABL than T - A contradiction.

Optimal Prefix Codes: False Start

Q. Where in the tree of an optimal prefix code should letters be placed with a high frequency?
A. Near the top.

Greedy template. Create tree top-down, split S into two sets S_1 and S_2 with (almost) equal frequencies. Recursively build tree for S_1 and S_2.
[Shannon-Fano, 1949] f_a=0.32, f_e=0.25, f_k=0.20, f_l=0.18, f_u=0.05





Optimal Prefix Codes: Huffman Encoding

- Observation. Lowest frequency items should be at the lowest level in tree of optimal prefix code.
- Observation. For $n > 1$, the lowest level always contains at least two leaves.
- Observation. The order in which items appear in a level does not matter.
- Claim. There is an optimal prefix code with tree T* where the two lowest-frequency letters are assigned to leaves that are siblings in T*.

Optimal Prefix Codes: Huffman Encoding

- Greedy template. [Huffman, 1952] Create tree bottom-up.
- Make two leaves for two lowest-frequency letters y and z. Recursively build tree for the rest using a meta-letter for yz.

```
Huffman(S) {
   if |S|=2 {
      return tree with root and 2 leaves
   } else {
      let y and z be lowest-frequency letters in S
      S′ = S
      remove y and z from S′
      insert new letter ω in S′ with fω=fy+fz
      T′ = Huffman(S′)
      T = add two children y and z to leaf ω from T′
      return T
   }
}
```

Optimal Prefix Codes: Huffman Encoding

```
Huffman(S) {
    if |S|=2 {
        return tree with root and 2 leaves
    } else {
        let y and z be lowest-frequency letters in S
        S′ = S
        remove y and z from S′
        insert new letter ω in S′ with f_ω=f_y+f_z
        T′ = Huffman(S′)
        T = add two children y and z to leaf ω from T′
        return T
    }
}
```

- Time complexity: $T(n) = T(n\text{-}1) + O(n)$
- So, $O(n^2)$
- To implement finding lowest-frequency letters efficiently, we can use priority queue for S:

  $T(n) = T(n\text{-}1) + O(log\ n)$ so $O(n\ log\ n)$

Huffman Encoding: Greedy Analysis

- Claim. Huffman code for S achieves the minimum ABL of any prefix code.
- Claim. $ABL(T') = ABL(T) - f\omega$
- Pf.

$$\begin{aligned}
\mathrm{ABL}(T) &= \sum_{x \in S} f_x \cdot \mathrm{depth}_T(x) \\
&= f_y \cdot \mathrm{depth}_T(y) + f_z \cdot \mathrm{depth}_T(z) + \sum_{x \in S, x \neq y,z} f_x \cdot \mathrm{depth}_T(x) \\
&= \left(f_y + f_z\right) \cdot \left(1 + \mathrm{depth}_T(\omega)\right) + \sum_{x \in S, x \neq y,z} f_x \cdot \mathrm{depth}_T(x) \\
&= f_\omega \cdot \left(1 + \mathrm{depth}_T(\omega)\right) + \sum_{x \in S, x \neq y,z} f_x \cdot \mathrm{depth}_T(x) \\
&= f_\omega + \sum_{x \in S'} f_x \cdot \mathrm{depth}_{T'}(x) \\
&= f_\omega + \mathrm{ABL}(T')
\end{aligned}$$

Huffman Encoding: Greedy Analysis

- Claim. Huffman code for *S* achieves the minimum *ABL* of any prefix code.
- **Pf.** (by induction)
- **Base:** For $n = 2$ there is no shorter code than root and two leaves.
- **Hypothesis:** Suppose Huffman tree *T'* for *S'* of size *n-1* with ω instead of *y* and *z* is optimal. (IH)
- **Step:** (by contradiction)
- Idea of proof:
  - » Suppose other tree *Z* of size n is better.
  - » Delete lowest frequency items *y* and *z* from *Z* creating *Z'*
  - » *Z'* cannot be better than *T'* by IH.

Huffman Encoding: Greedy Analysis

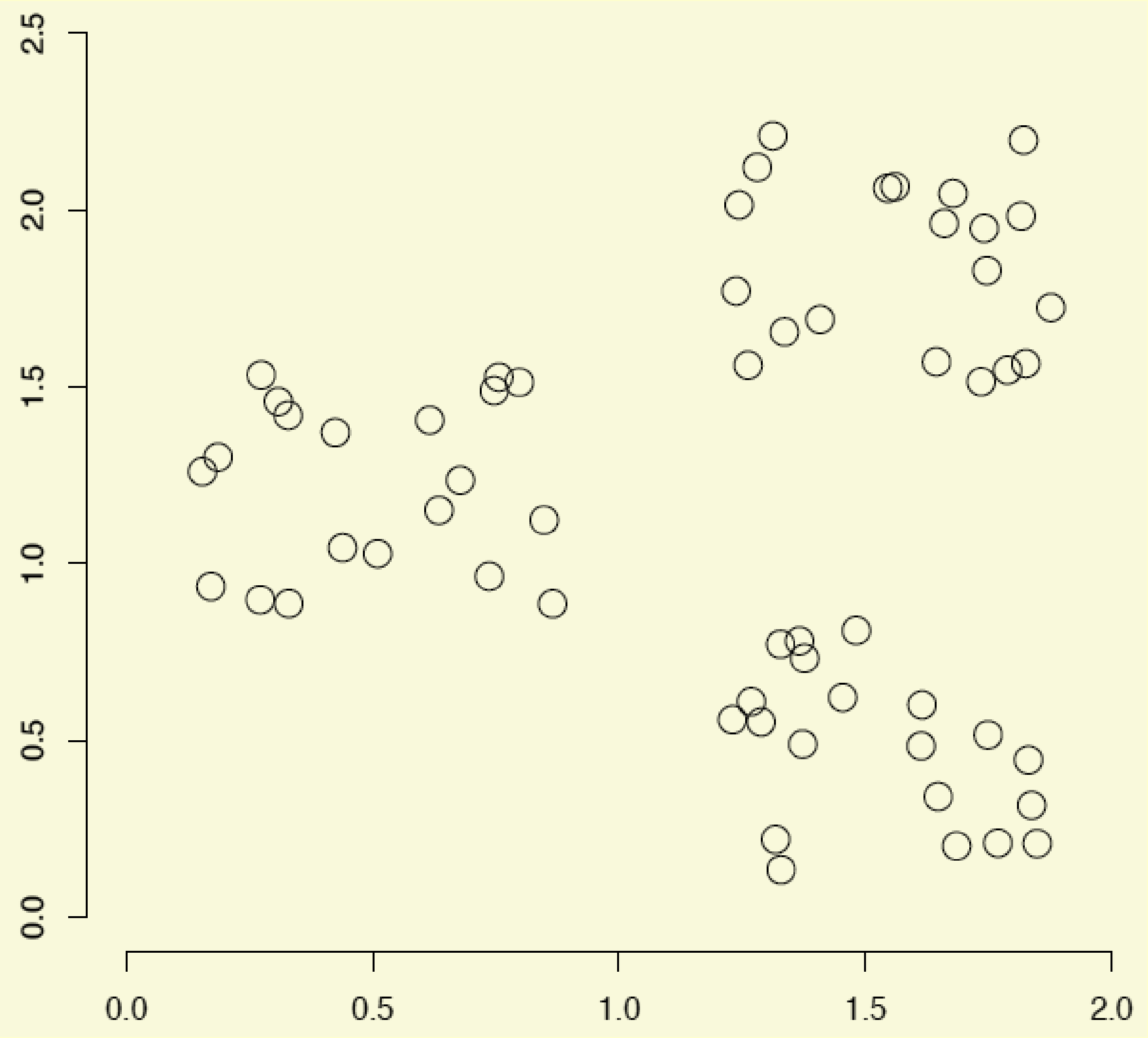
- **Inductive Step:** (by contradiction)
- Suppose Huffman tree T for S is not optimal.
- So there is some tree Z such that $ABL(Z) < ABL(T)$.
- Then there is also a tree Z for which leaves y and z exist that are siblings and have the lowest frequency (see observation).
- Let Z' be Z with y and z deleted, and their former parent labeled ω.
- Similar T' is derived from S' in our algorithm.
- We know that $ABL(Z') = ABL(Z)\text{-}f\omega$, as well as $ABL(T') = ABL(T) - f\omega$.
- But also $ABL(Z) < ABL(T)$, so $ABL(Z') < ABL(T')$. - Contradiction with IH.

Clustering

What is clustering?

- Clustering: the process of grouping a set of objects into classes of similar objects
  - » Objects within a cluster should be similar.
  - » Objects from different clusters should be dissimilar.
- One common approach: define a *distance function* on the objects, with the interpretation that objects at a larger distance from one another are less similar to each other.
- Given a distance function on the objects, the clustering problem seeks to divide them into groups so that, intuitively, objects within the same group are close and objects in different groups are far apart

A data set with clear cluster structure



- How would you design an algorithm for finding the three clusters in this case?

Clustering

Clustering. Given a set U of n objects labeled $p_1, \ldots, p_n$, classify into coherent groups.

photos, documents. micro-organisms

Distance function. Numeric value specifying "closeness" of two objects.

number of corresponding pixels whose intensities differ by some threshold

Fundamental problem. Divide into clusters so that points in different clusters are far apart.

- Routing in mobile ad hoc networks.
- Identify patterns in gene expression.
- Document categorization for web search.
- Similarity searching in medical image databases
- Skycat: cluster 10^9 sky objects into stars, quasars, galaxies.

Clustering of Maximum Spacing

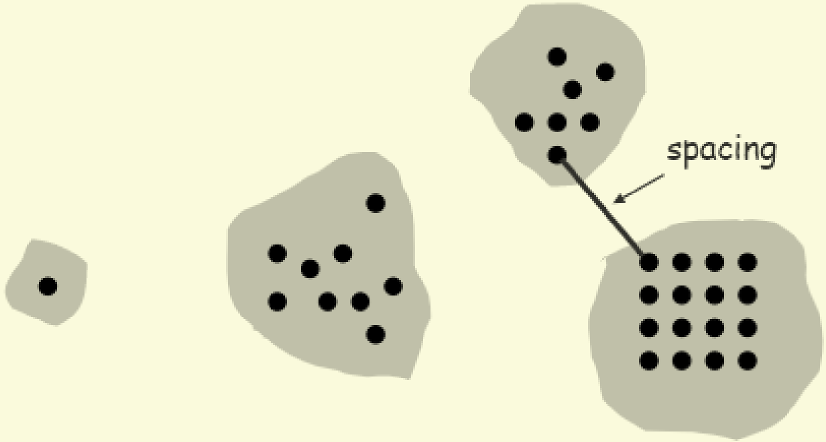
k-clustering. Divide objects into k non-empty groups.

Distance function. Assume it satisfies several natural properties.

- $d(p_i, p_j) = 0$ iff $p_i = p_j$ (identity of indiscernibles)
- $d(p_i, p_j) \geq 0$ (nonnegativity)
- $d(p_i, p_j) = d(p_j, p_i)$ (symmetry)

Spacing. Min distance between any pair of points in different clusters.

Clustering of maximum spacing. Given an integer k, find a k-clustering of maximum spacing.



k = 4

Greedy Clustering Algorithm

Single-link k-clustering algorithm.

- Form a graph on the vertex set U one edge at a time
- Initially we have 0 edge, thus we have n clusters
- Find the closest pair of nodes such that each node is in a different cluster, and add an edge between them → these two clusters merge into one cluster
- Repeat n-k times until there are exactly k clusters.

Key observation. This procedure is precisely Kruskal's algorithm (except we stop when there are k connected components).

Remark. Equivalent to finding an MST and deleting the k-1 most expensive edges.

Greedy Clustering Algorithm: Analysis

Theorem. The k clustering $C^*_1, \ldots, C^*_k$ (denoted by C^*)formed by deleting the k-1 most expensive edges of a MST is a k-clustering of max spacing.

Pf. Let C denote some other clustering $C_1, \ldots, C_k$.

- The spacing of C^* is the length d* of the (k-1)-th most expensive edge (the edge that will be added by Kruskal next).
- Let p_i, p_j be in the same cluster in C^*, say C^*_r, but different clusters in C, say C_s and C_t.
- Some edge (p, q) on p_i-p_j path in C^*_r spans two different clusters in C.
- All edges on p_i-p_j path have length $\leq$ d* since Kruskal chose them.
- Spacing of C is $\leq$ d* since p and q are in different clusters in C, thus C's max spacing is smaller than C^*. ▪