Algorithm Design and Analysis

CSE
565

LECTURE 8

Greedy Algorithms

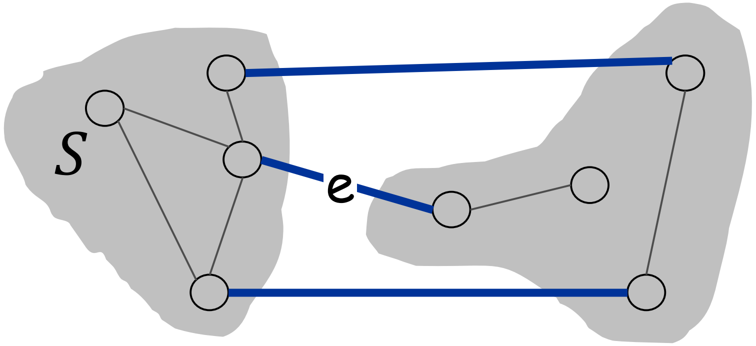
- Minimum Spanning Tree
- Clustering
- Huffman Codes

Sofya Raskhodnikova

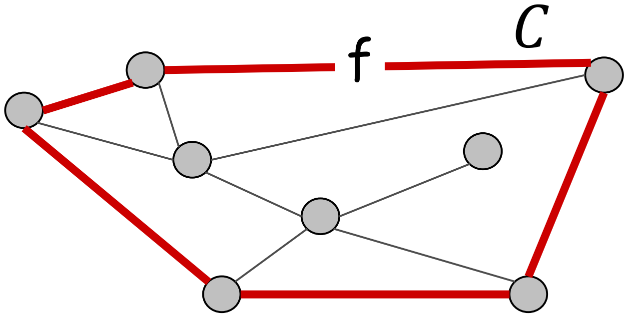
Minimum Spanning Tree

Cut and Cycle Properties

- **Cut property.** Let S be a subset of nodes. Let e be the min weight edge with exactly one endpoint in S. Then the MST contains e.
- **Cycle property.** Let C be a cycle, and let f be the max weight edge in C. Then the MST does not contain f.



e is in the MST



f is not in the MST

Review Questions

Let G be a connected undirected graph with distinct edge weights. Answer true or false:

- Let e be the cheapest edge in G. The MST of G contains e.
- Let e be the most expensive edge in G. The MST of G does not contains e.

Review Questions

Let G be a connected undirected graph with distinct edge weights. Answer true or false:

- Let e be the cheapest edge in G. The MST of G contains e.

(Answer: True, by the Cut Property)

- Let e be the most expensive edge in G. The MST of G does not contains e.

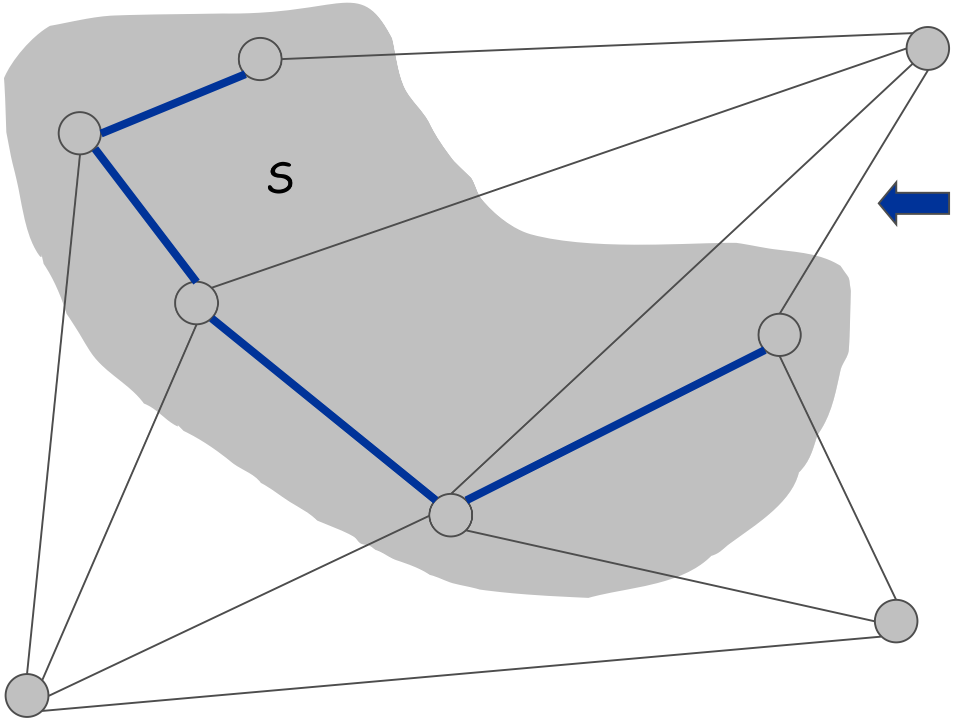
(Answer: False. Counterexample: if G is a tree, all its edges are in the MST.)

Greedy Algorithms for MST

- **Kruskal's:** Start with $T = \emptyset$. Consider edges in ascending order of weights. Insert edge e in T unless doing so would create a cycle.
- **Reverse-Delete:** Start with $T = E$. Consider edges in descending order of weights. Delete edge e from T unless doing so would disconnect T.
- **Prim's:** Start with some root node s. Grow a tree T from s outward. At each step, add to T the cheapest edge e with exactly one endpoint in S.
- **Borůvka's:** Start with $T = \emptyset$. At each round, add the cheapest edge leaving each connected component of T.

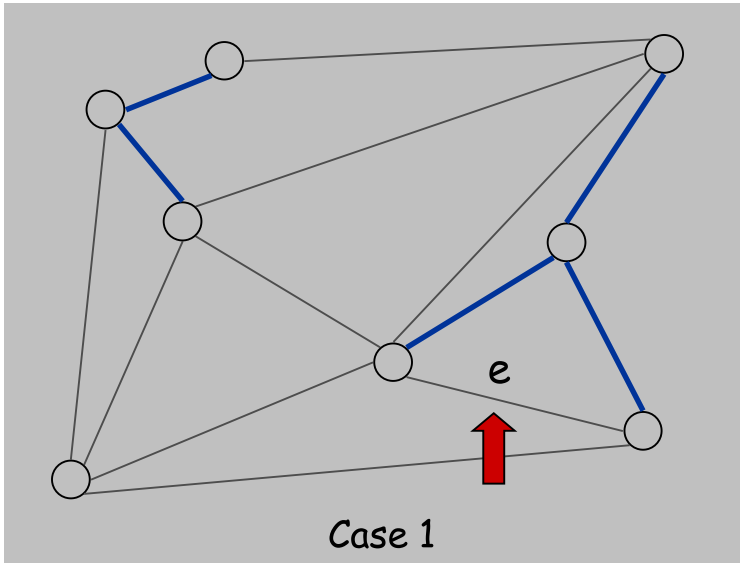
Prim's Algorithm: Correctness

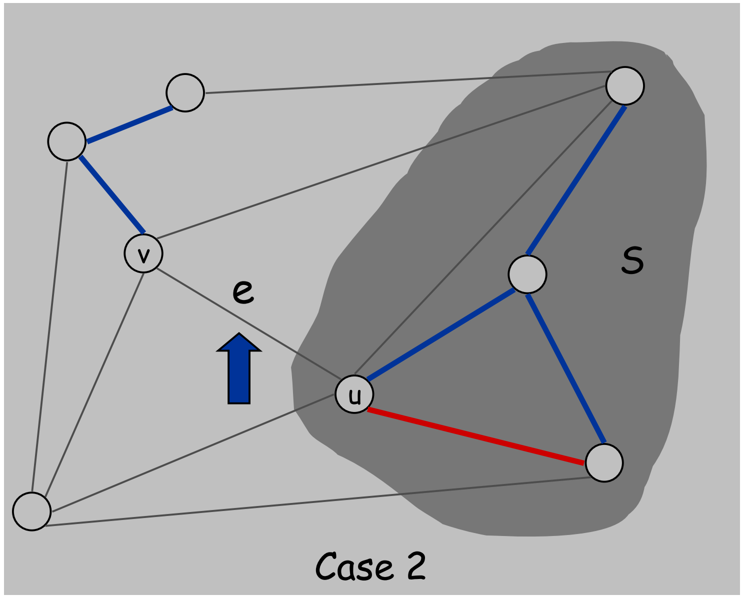
- Prim's algorithm. [Jarník 1930, Prim 1959]
  - Apply cut property to S.
  - When edge weights are distinct, every edge that is added must be in the MST
  - Thus, Prim's algorithm outputs the MST



Correctness of Kruskal

- [Kruskal, 1956]: Consider edges in ascending order of weight.
  - **Case 1:** If adding e to T creates a cycle, discard e according to cycle property.





  - **Case 2:** Otherwise, insert e = (u, v) into T according to cut property where S = set of nodes in u's connected component.

Non-distinct edges?

Lexicographic Tiebreaking

- To remove the assumption that all edge costs are distinct:

  perturb all edge costs by tiny amounts to break any ties.

- **Impact.** Kruskal and Prim only interact with costs via pairwise comparisons. If perturbations are sufficiently small, MST with perturbed costs is MST with original costs. ↑

  e.g., if all edge costs are integers, perturbing cost of edge e_i by i / n^2

- **Implementation.** Can handle arbitrarily small perturbations implicitly by breaking ties lexicographically, according to index.

```
boolean less(i, j) {
   if      (cost($e_i$) < cost($e_j$)) return true
   else if (cost($e_i$) > cost($e_j$)) return false
   else if (i < j)                      return true
   else                                 return false
}
```

Implementing MST algorithms

- Prim: similar to Dijkstra
- Kruskal:
  - Requires efficient data structure to keep track of "islands": Union-Find data structure
  - KT Chapter 4.6

Implementation of Prim(G,w)

IDEA: Maintain $V - S$ as a priority queue Q (as in Dijkstra). Key each vertex in Q with the weight of the least-weight edge connecting it to a vertex in S.

```
$Q \leftarrow V$
$key[v] \leftarrow \infty$ for all $v \in V$
$key[s] \leftarrow 0$ for some arbitrary $s \in V$
while $Q \neq \emptyset$
    do $u \leftarrow$ EXTRACT-MIN($Q$)
        for each $v \in Adjacency\text{-}list[u]$
            do if $v \in Q$ and $w(u, v) < key[v]$
                then $key[v] \leftarrow w(u, v)$    ⊳ DECREASE-KEY
                     $\pi[v] \leftarrow u$
```

At the end, $\{(v, \pi[v])\}$ forms the MST.

Analysis of Prim

$\Theta(n)$ total

```
$Q \leftarrow V$
$key[v] \leftarrow \infty$ for all $v \in V$
$key[s] \leftarrow 0$ for some arbitrary $s \in V$
```

n times

```
while $Q \neq \emptyset$
    do $u \leftarrow$ EXTRACT-MIN$(Q)$
```

$degree(u)$ times

```
        for each $v \in Adj[u]$
            do if $v \in Q$ and $w(u, v) < key[v]$
                then $key[v] \leftarrow w(u, v)$
                     $\pi[v] \leftarrow u$
```

$\Theta(m)$ implicit DECREASE-KEY's.

Time: as in Dijkstra

Implementation of Kruskal

- Use the **Union-Find** data structure.
  - Build set T of edges in the MST.
  - Maintain a set for each connected component.

```
Kruskal(G, w) {
    Sort edges weights so that $w_1 \le w_2 \le ... \le w_m$.
    $T \leftarrow \phi$
    foreach ($u \in V$) make a set containing singleton u

    foreach edge (u,v)
        //go through edges in sorted order
        if (u and v are in different sets) {
            $T \leftarrow T \cup \{e_i\}$
            merge the sets containing u and v
        }
    return T
}
```

are u and v in different connected components?

merge two components

Union-Find Data Structures

Operation\ Implementation	**Array + linked-lists and sizes**	**Balanced Trees**	**Trees with Path Compression**
Find (worst-case)	$\Theta(1)$	$\Theta(\log n)$	$\Theta(\log n)$
Union of sets A,B (worst-case)	$\Theta(\min(\|A\|,\|B\|))$ (could be as large as $\Theta(n)$	$\Theta(\log n)$	$\Theta(\log n)$
Amortized analysis: n unions and n finds, starting from singletons	$\Theta(n \log n)$	$\Theta(n \log n)$	$\Theta(n\, \alpha(n))$

- Here $\alpha(n)$ is the inverse Ackerman function, which grows much more slowly than log n.
- See KT Chapter 4.6

The Union-Find Data Structure

Operations:

- MAKE-UNION-FIND(S): creates the data structure; puts all elements in S into separate sets.

  $\boldsymbol{O}(\boldsymbol{n})$ time where $\boldsymbol{n} = |\boldsymbol{S}|$

- FIND(u): returns the representative of the set containing u.

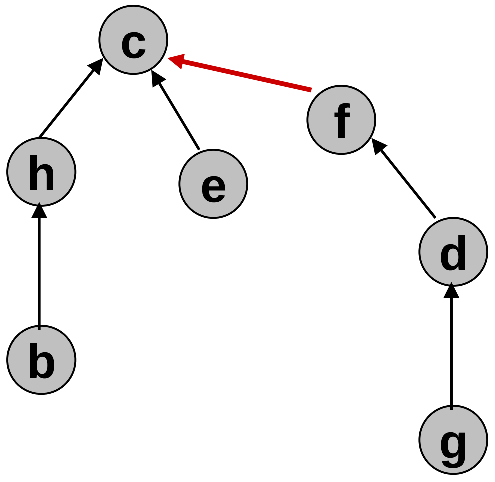
  $\boldsymbol{O}(\log \boldsymbol{n})$ time

- UNION(A,B): merge sets A,B into a single set.

  $\boldsymbol{O}(\mathbf{1})$ time

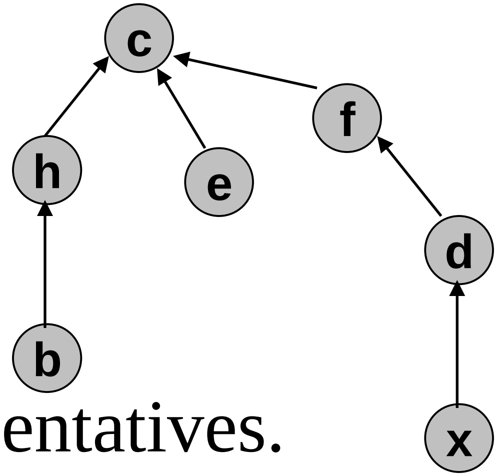
Forest Representation

- Each element is a node.
- Each tree represents one set (store its size).
- The root is the representative.
- MAKE-UNION-FIND: create roots
  - **O(1) time per element**
- UNION(A,B): point the root of the smaller tree to the root of the larger tree
  - **O(1) time**



FIND operation

- FIND(x): follow the links to the root.



Theorem. FIND takes **O(log n)** time.

Proof: Time to evaluate FIND(x)

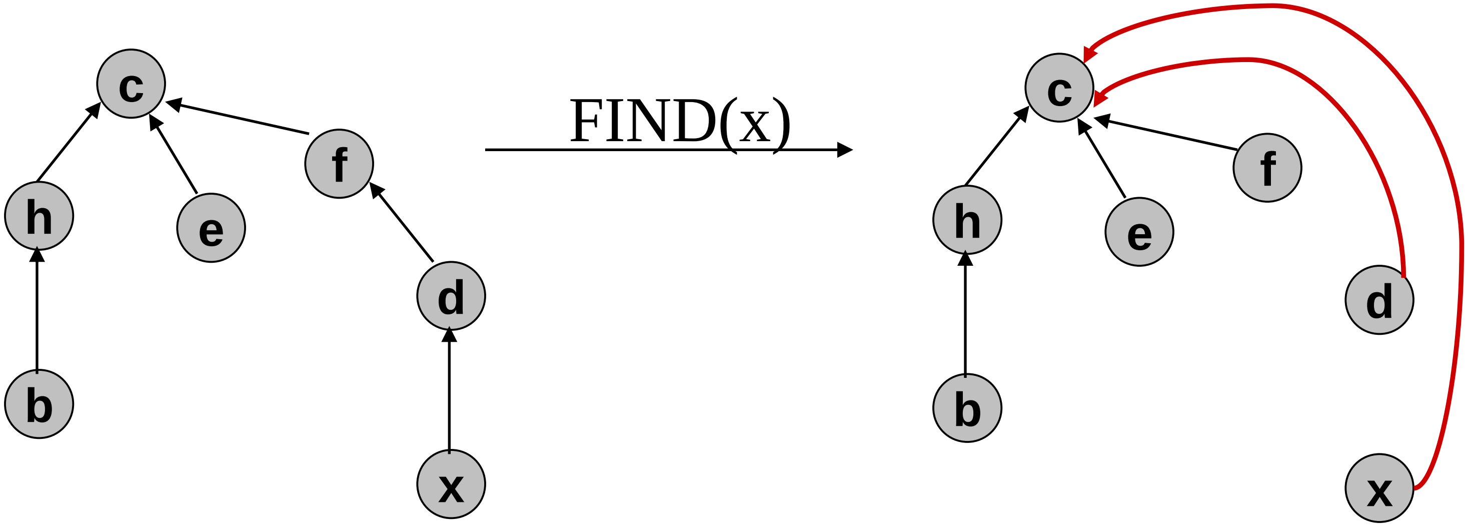
= number of predecessors of x

= number of times x changes representatives.

- Every time x changes representatives, the size of its set at least doubles. It can happen $\leq \log_2 n$ times. ▪

An Improvement to FIND

- **Path Compression:** update every pointer on the way to the root.



- **Theorem.** n FIND operations take **O(n α(n))** time, where **α(n)** is inverse Ackerman function.

S. Raskhodnikova; based on slides by E. Demaine, C. Leiserson, A. Smith, K. Wayne

Implementation of Kruskal

- Build set T of edges in the MST.
- Maintain a set for each connected component.

```
Kruskal(G, w) {
    Sort edges weights so that $w_1 \le w_2 \le \ldots \le w_m$.
    $T \leftarrow \phi$
    MAKE-UNION-FIND(V)
    foreach edge (u,v)
        //go through edges in sorted order
        if (FIND(u) $\neq$ FIND(v)) {
            $T \leftarrow T \cup \{e_i\}$
            UNION(FIND(u), FIND(v))
        }
    return T
}
```

are u and v in different connected components?

merge two components

- Sorting: $O(m \log m) = O(m \log (n^2)) = O(m \log n)$
- Union-Find operations: $O(m \log n)$

$\}$ $O(m \log n)$ time

MST Algorithms in 2016

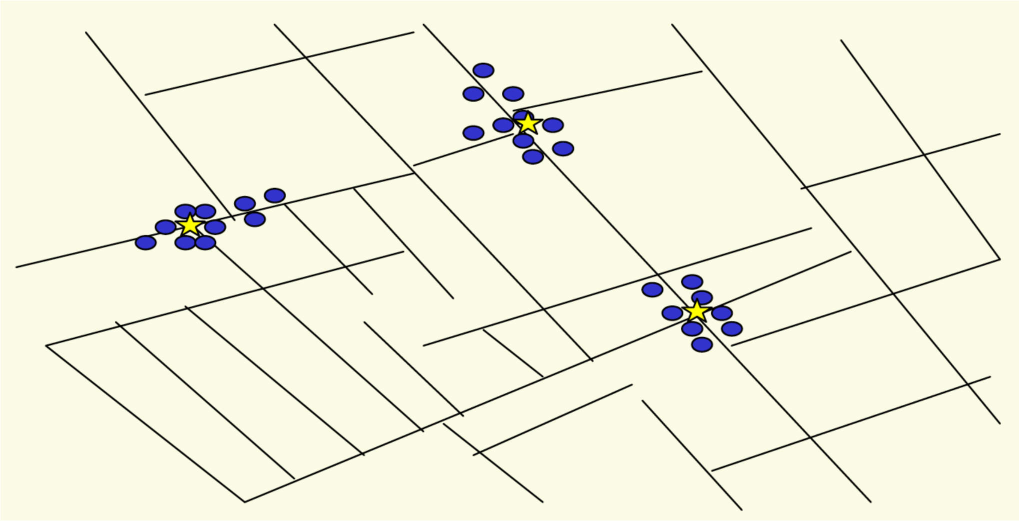
- **Deterministic** comparison-based algorithms.
  - O(m log n) [Jarník, Prim, Dijkstra, Kruskal, Boruvka]
  - O(m α (m, n)). [Chazelle 2000]
- Holy grail: O(m).

- Related.
  - O(m) **randomized**. [Karger-Klein-Tarjan 1995]
  - O(m) **verification**. [Dixon-Rauch-Tarjan 1992]

Max-Space Clustering

Clustering

Given a set of n items **(e.g., photos, documents, microorganizms)** labeled $p_1,\ldots,p_n$, classify them into coherent groups.



Outbreak of cholera deaths in London in 1850s.
Reference: Nina Mishra, University of Virginia

k-Clustering

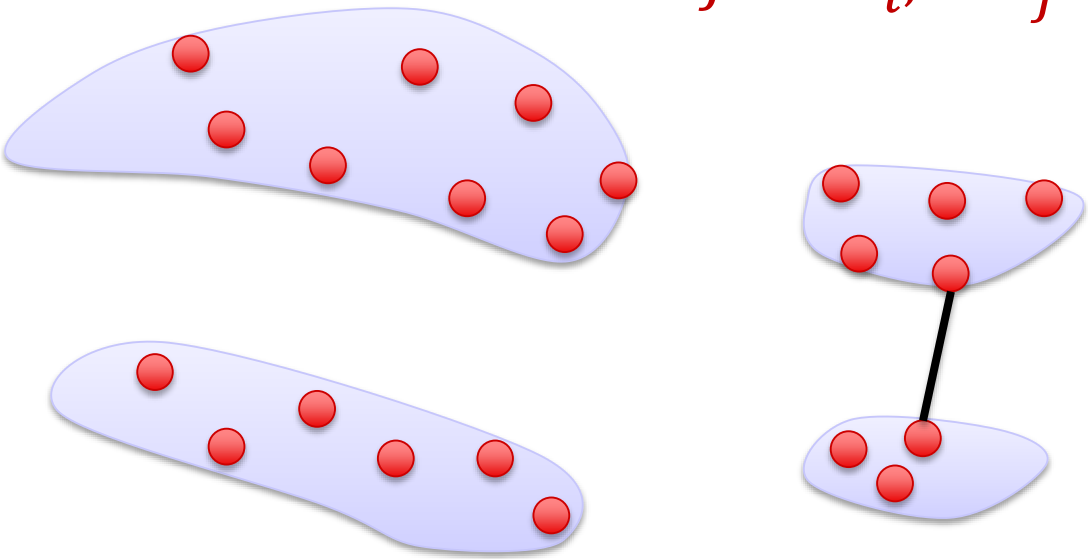
- Given: a set of n items
- Goal: partition items into k sets such that
  - “similar” items are together
  - “different” items are separate

Items	**Distance**
Newspaper articles	# words that appear in 1 but not both articles
Students at university X	Difference in course lists
Nodes in social network	Difference in “friends lists”

Max-spacing Clustering

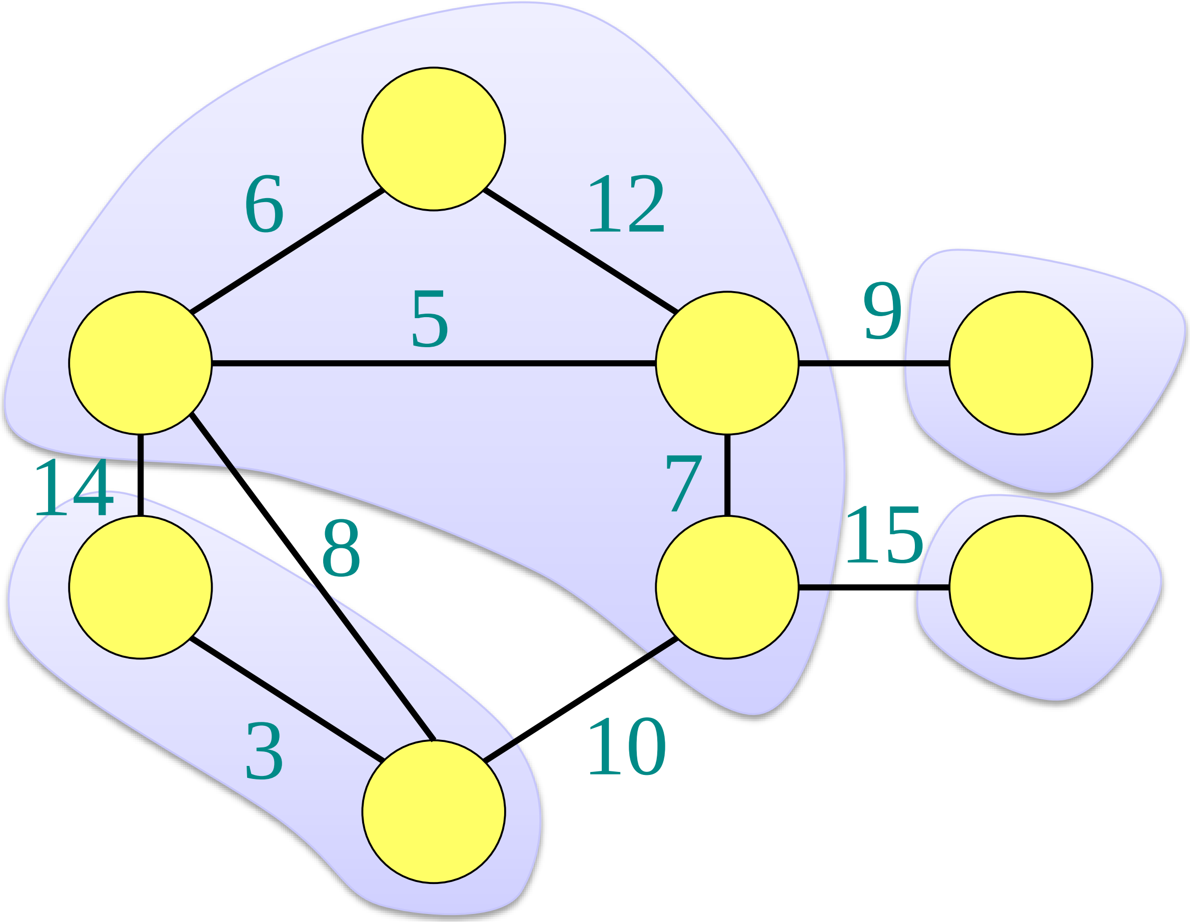
- Input: set V of n items and a distance function $d: V \times V \to \mathbb{R}^{\geq 0}$
- Goal: Find k disjoint nonempty sets $C_1, C_2, .. C_k \subseteq V$ that maximize

$$\text{Spacing}(C_1, \ldots, C_k) = \min_{i \neq j} \min_{u \in C_i, v \in C_j} d(u, v)$$



S. Raskhodnikova; based on slides by E. Demaine, C. Leiserson, A. Smith, K. Wayne

Example



9/19/2016

S. Raskhodnikova; based on slides by E. Demaine, C. Leiserson, A. Smith, K. Wayne

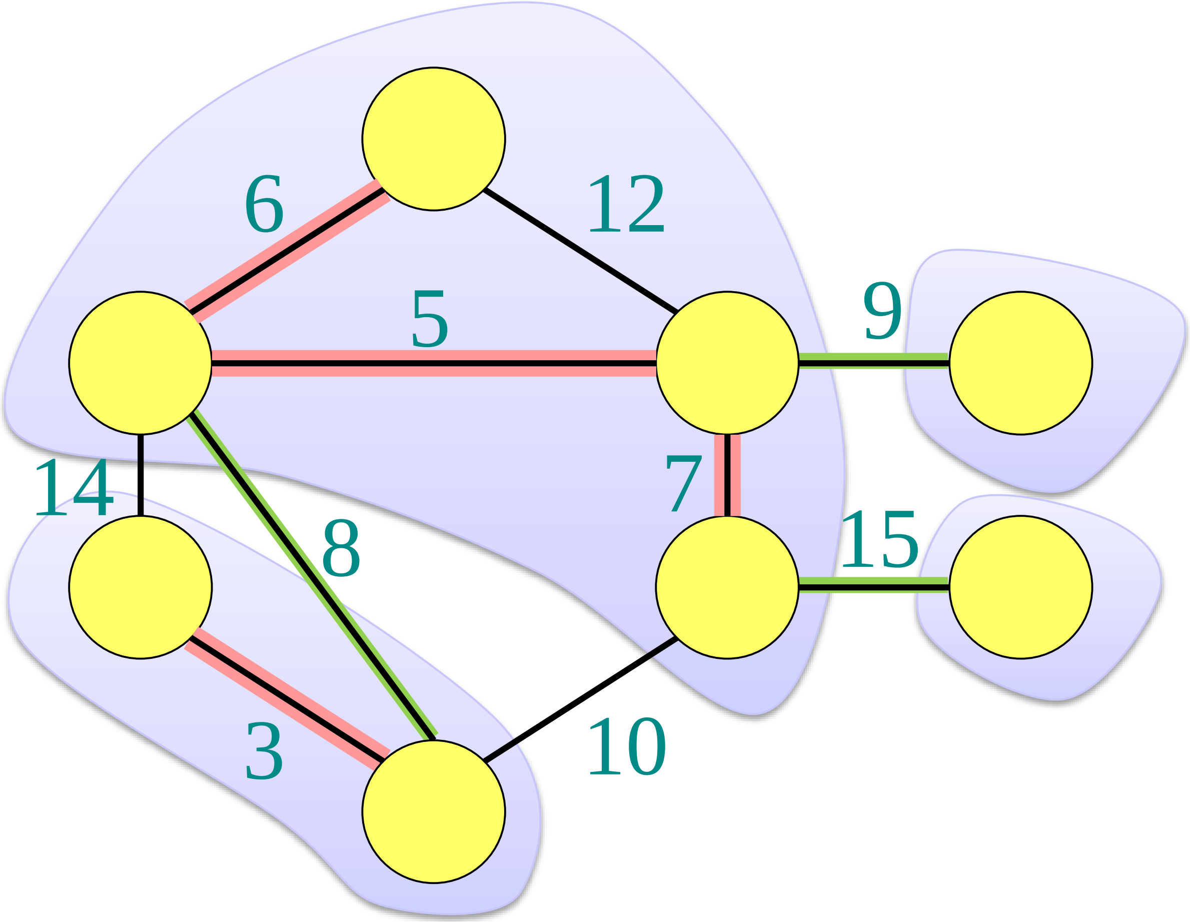
Single Linkage Clustering

1. Start with n clusters, one per node
2. While there are more than k clusters
   - Find a closest pair of clusters i, j, where
     $$dist\left(C_i, C_j\right) = \min_{u \in C_i, v \in C_j} d(u, v)$$
   - Merge C_i with C_j

What MST algorithm is this?

What is the running time?

Example with MST



S. Raskhodnikova; based on slides by E. Demaine, C. Leiserson, A. Smith, K. Wayne

Optimality of Single Linkage (SL)

Theorem. SL clustering has maximal spacing.

Proof: Pick any other clustering $C_1', \dots, C_k'$

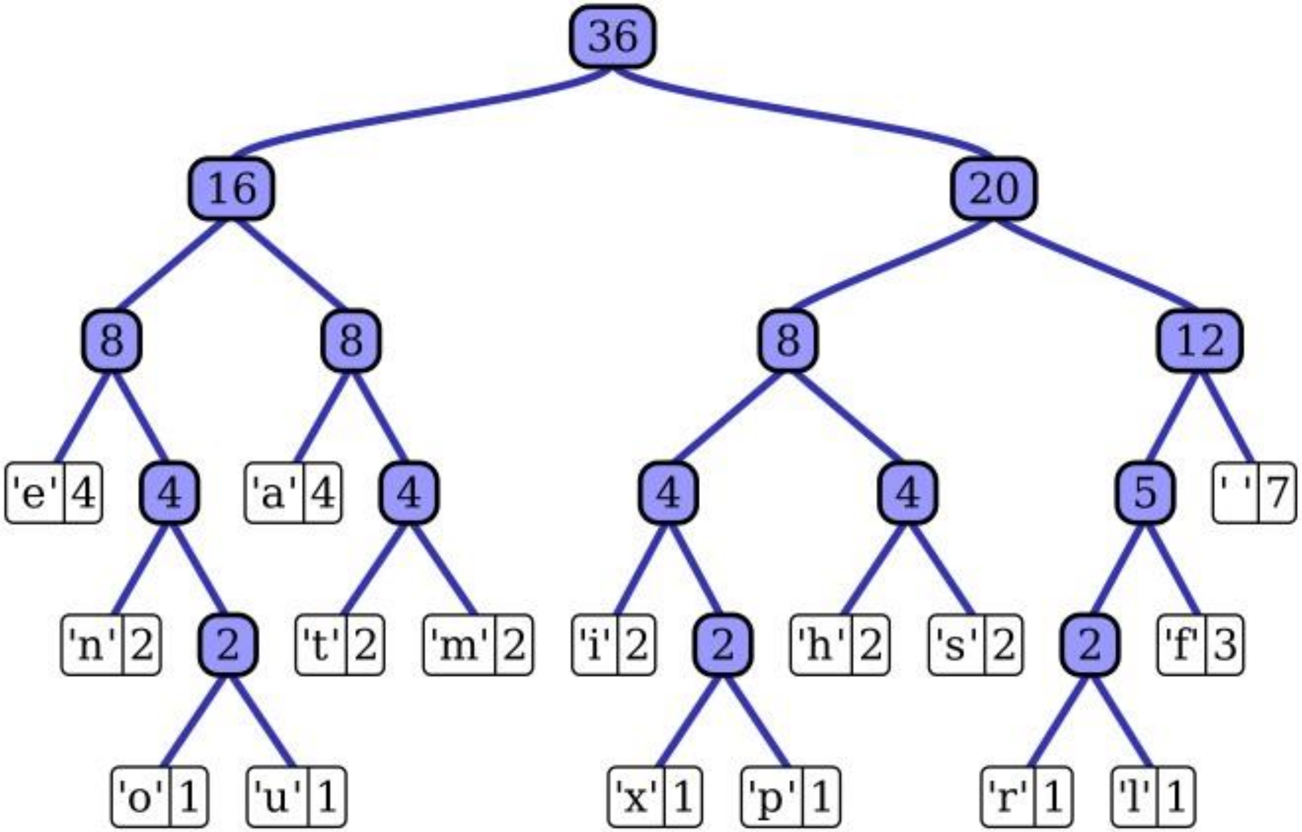
- There exists a SL cluster C_i that is "split" by the C_j's
  - $\exists\, x, y \in C_i$ such that $x \in C_j, y \in C_\ell$ and $j \neq \ell$.
- Look at the path P in MST from x to y.
  - All edges on P have weight less than $Spacing(C_1, \dots, C_k)$ since algorithm proceeds in ascending order of weight
  - Some edge e in P crosses from C_j' to C_ℓ'
- So $Spacing(C_1', \dots, C_k') \leq Spacing(C_1, \dots, C_k)$. QED

Huffman codes

Prefix-free codes

- **Binary code** maps characters in an alphabet (say {A,…,Z}) to binary strings
- **Prefix-free code**: no codeword is a prefix of any other
  - ASCII: prefix-free (all symbols have the same length)
  - Not prefix-free:
    - a → 0
    - b → 1
    - c → 00
    - d → 01
    - …
- Why is prefix-free good?

A prefix-free code for a few letters



A tree for "this is an example of a huffman tree“

- e.g. e → 00, p → 10011

Source: WIkipedia

How good is a prefix-free code?

- Given a text, let f[i] = # occurrences of letter *i*
- Total number of symbols needed

$$\sum_i f[i] \cdot depth(i)$$

- How do we pick the best prefix-free code?

S. Raskhodnikova; based on slides by E. Demaine, C. Leiserson, A. Smith, K. Wayne

Huffman's Algorithm (1952)

- Given individual letter frequencies f[1, .., n]:
  - Find the two least frequent letters i,j
  - Merge them into symbol with frequency f[i]+f[j]
  - Repeat
- e.g.
  - a: 6
  - b: 6
  - c: 4
  - d: 3
  - e: 2

Theorem: Huffman algorithm finds an optimal prefix-free code

Warming up

- **Lemma 0**: Every optimal prefix-free code corresponds to a **full** binary tree.
  - (Full = every node has 0 or 2 children)

- **Lemma 1**: Let x and y be two least frequent characters. There is an optimal code in which x and y are siblings.
  - Prove using an exchange argument.

Huffman codes are optimal

Proof by induction

- Base case: two symbols; only one full tree.
- Induction step:
  - Suppose f[1], f[2] are smallest in f[1,…,n]
  - T is an optimal code for {1,…,n}
  - Lemma 1 ==> can choose T where 1,2 are siblings.
  - New symbol numbered n+1, with f[n+1] = f[1]+f[2]
  - T’ = code obtained by merging 1,2 into n+1

Cost of T in terms of T’:

$$
\begin{aligned}
cost(T) &= \sum_{i=1}^{n} f[i] \cdot depth(i) \\
&= \sum_{i=3}^{n+1} f[i] \cdot depth(i) + f[1] \cdot depth(1) + f[2] \cdot depth(2) - f[n+1] \cdot depth(n+1) \\
&= cost(T') + f[1] \cdot depth(1) + f[2] \cdot depth(2) - f[n+1] \cdot depth(n+1) \\
&= cost(T') + (f[1] + f[2]) \cdot depth(T) - f[n+1] \cdot (depth(T) - 1) \\
&= cost(T') + f[1] + f[2]
\end{aligned}
$$

- Minimizing cost(T) is the same as minimizing cost(T’).
- By induction hypothesis T’ is optimal.
- So, T is optimal, too. ▪

Notes

- See Jeff Erickson's lecture notes on greedy algorithms:
  - http://theory.cs.uiuc.edu/~jeffe/teaching/algorithms/
  - efficient implementation using min-heap

Data Compression for real?

- Generally, we don’t use letter-by-letter encoding
- Instead, find frequently repeated substrings
  - Lempel-Ziv algorithm extremely common
  - also has deep connections to entropy