Connected Components

Vertices v and w are connected if there is a path between them

Goal: Preprocess graph to answer queries of the form ‘is v connected to w?’ in constant time.

Public class CC

CC(Graph G) : find connected components in G  
Boolean connected(int v, int w)  
Int count() : number of connected components  
Int id(in v) component identifier for v

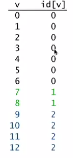
Not quite the same as Union-Find. Could not answer query always in constant time (UF).

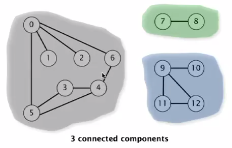
We will use a depth-first search, but keep different data this time than we would if finding paths.

The relation ‘is connected to’ is an equivalence relation’

* Reflexive: v is connect to v’
* Symmetric: if v is connect to w, then w is connected to v
* Transitive: if v is connected to w and w is connected to x, then v is connected to x

A connected component is a maximal set of connected vertices

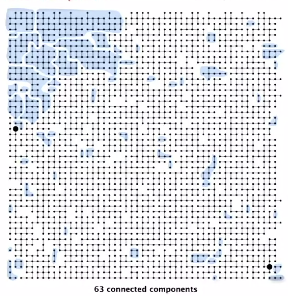




Assign identifier for each component (1, 2, 3 above), assign one of said identifiers to each vertex. Constant time connected lookup (check the id[v] == id[w])

Like a union-find tree, but trees are flat

Connected components visualized



Goal: Partition vertices into connected components

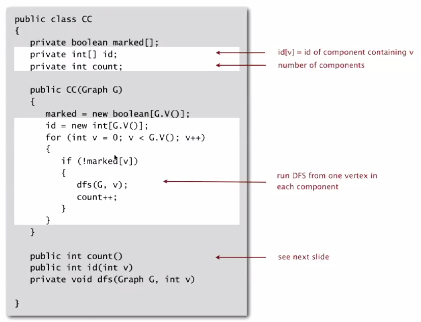
---

Connected components

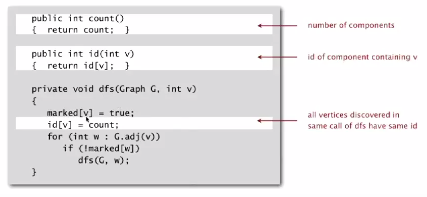
Initialize all vertices v as unmarked  
For each unmarked vertex v, run DFS to identify all vertices as part of the same component

---

Implementation of DFS with a goal of finding connected components

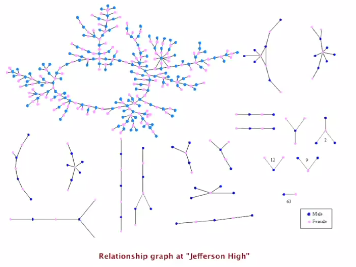


DFS implementation below:



Connected components application

Study spread of STDs



Particle detection

Given grayscale images, identify “blobs”

* Vertex pixel
* Edge: between two adjacent pixels with grayscale value >= 70
* Blob: connected components of 20-30 particles

