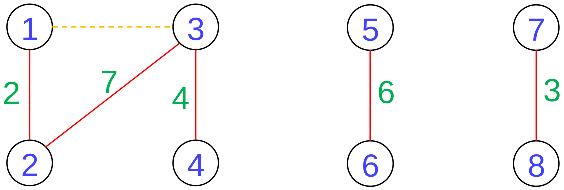
Data Structure for Disjoint Sets

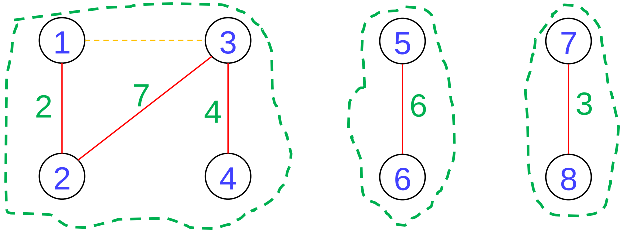
Dr. Shrutilipi Bhattacharjee, Assistant Professor, Dept. of IT, NIT Karnataka, India

Data Structures for Kruskal's Algorithm

- Does the addition of an edge (***u***, ***v***) to ***T*** result in a cycle?



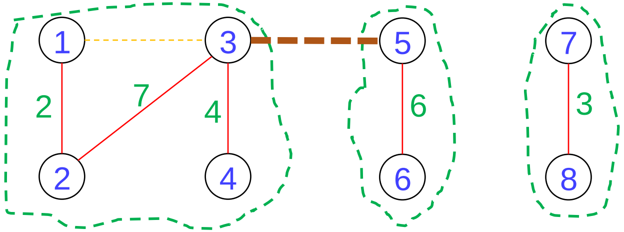
- Each connected component of ***T*** is a tree
- When ***u*** and ***v*** are in the same component, the addition of the edge (***u***, ***v***) creates a cycle
- When ***u*** and ***v*** are in the different components, the addition of the edge (***u***, ***v***) does not create a cycle



- Each component of ***T*** is defined by the vertices in the component
- Represent each component as a set of vertices

{1, 2, 3, 4}, {5, 6}, {7, 8}

- Two vertices are in the same component iff they are in the same set of vertices



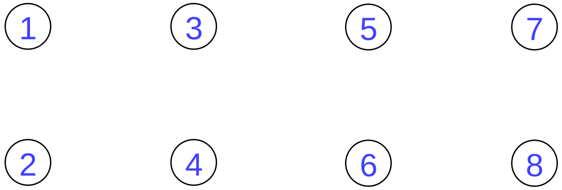
- When an edge (***u***, ***v***) is added to ***T***, the two components that have vertices ***u*** and ***v*** combine to become a single component

Data Structures for Kruskal's Algorithm

- In our set representation of components, the set that has vertex ***u*** and the set that has vertex ***v*** are united

{1, 2, 3, 4} + {5, 6} → {1, 2, 3, 4, 5, 6}

- Initially, ***T*** is empty and we had ***V*** connected components



- Initial sets are:

{1} {2} {3} {4} {5} {6} {7} {8}

- Does the addition of an edge (***u***, ***v***) to ***T*** result in a cycle? If not, add edge to ***T***

s1 = find(u); s2 = find(v);

if (s1 != s2)   union(s1, s2);

- Eventually we should have one tree ***T***, i.e., one connected components

Disjoint Set ADT

- The universe consists of ***N*** elements, named as $s_1, s_2, \ldots, s_N$
- The ADT is a collection of sets of elements
- Initially, each element is in exactly one set
  - Sets are disjoint
  - $\{s_1\}, \{s_2\}, \ldots, \{s_N\}$
  - To start, each set contains one element

s_i = **find**(u); s_j = **find**(v);

if (s_i != s_j) **union**(s_i, s_j);

- Each set has a name, which is the name of one of its elements (any one will do)
- For our Kruskal's algorithm:
  - Initially each of these elements (s_i) corresponds to one vertex
  - What are the operations required? **union(**s_i, s_j**)**
    - Example: **union**($\{s_1, s_2, s_3\}, \{s_4, s_5, s_6\}$) = $\{s_1, s_2, s_3, s_4, s_5, s_6\}$
  - Given an edge, I have to look at the two end points of the edge and determine if they belong to the same connected component or not: **find**(u)

Modified Kruskal's Algorithm

- How many **union**() operations required?
  - $V - 1$
- How many **find**() operations required?
  - $\leq 2E$
- What is the total running time of the modified algorithm?
  - Lets say each **union**() procedure takes *U* time and each **find**() operation takes *F* time
  - O($E \log E + U.\ V + F\ .\ E$)
- So now, we have to find out a good data structure, which will have small *U* and small *F*

Sort edges in increasing order of cost of edges

$e_1, e_2, e_3, e_4, \ldots e_E$ such that $c(e_i) \leq c(e_{i+1})$

T = Φ

for i = 1 to E **do**

  let $\{e_i\} = (u, v)$

  if find(u) != **find**(v)

    T = $\{e_i\} \cup$ T

    union(**find**(u), **find**(v));

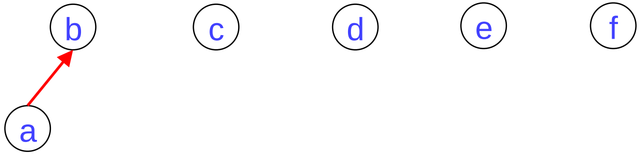
return T

Disjoint Sets

- How will we maintain this collection of disjoint sets?
  - If we need to have the overall time complexity as O(*E log E* + *U*. *V* + *F* . *E*), somehow we need to bound both *U* and *F* by (*E log E*) time, i.e., ≤ *E log E*

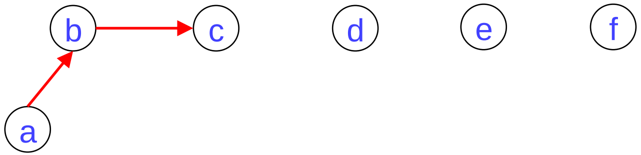
- A new data structure is needed? Let us have the example first
  - Suppose, my universe *U* = {a, b, c, d, e, f} with 6 elements
  - Initially what are the sets in my collection?
  - The singletons sets: {a}, {b}, {c}, {d}, {e}, {f}
  - One vertex for each of these six sets

- Now suppose we want to perform **union**(**find**(*a*), **find**(*b*))
  - We will make either **{a}** or **{b}** point to the other



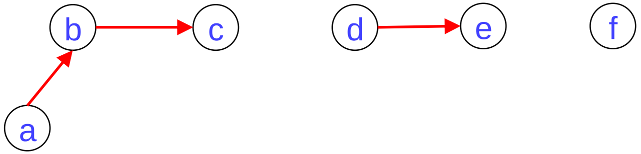
Disjoint Sets

- Now suppose we want to perform **union**(**find**(*a*), **find**(*c*))
  - When we will perform **find**(*a*), we will start from *a* and keep going up the tree till the root
  - So every connected component is a set now
  - How do we represent a set?
    - Each set is represented by one of the elements in a set (leader)
    - In this representation, it will be the root of the set
  - So **find**(*a*) and **find**(*c*) will return the roots of the tree they belong to
    - Therefore, **find**(*a*) will return ***b*** and **find**(*c*) will return ***c***
  - What union does?
    - It takes the root of these two trees and links them up and makes one point to the other
  - **union**(**find**(*a*), **find**(*c*))

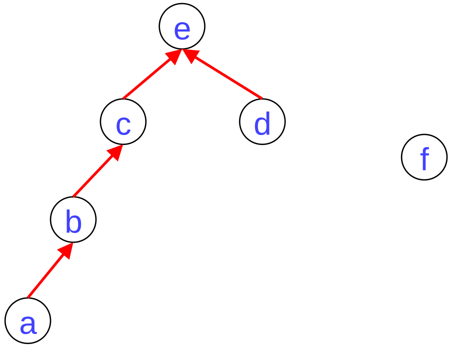


Disjoint Sets

- Now suppose we want to perform **union**(**find**(*d*), **find**(*e*))



- **union**(**find**(*b*), **find**(*d*))
  - ***a*** returns a pointer to ***c*** and ***d*** returns a pointer to ***e***
  - We need to link up these two vertices



- So, what is the problem with this implementation?
  - How much time does **union**() take?
    - O(1)
  - How much time does **find**() take?
    - O(V)

- Can we do any better?

Next Lecture

Union by Rank and Path Compression Heuristics

Thank you for your attention...

Any question?

Contact:
Department of Information Technology, NITK Surathkal, India
6th Floor, Room: 13
Phone: +91-9477678768
E-mail: shrutilipi@nitk.edu.in, shrutilipi.bhattacharjee@tum.de