Greedy Algorithms: Kruskal’s Algorithm

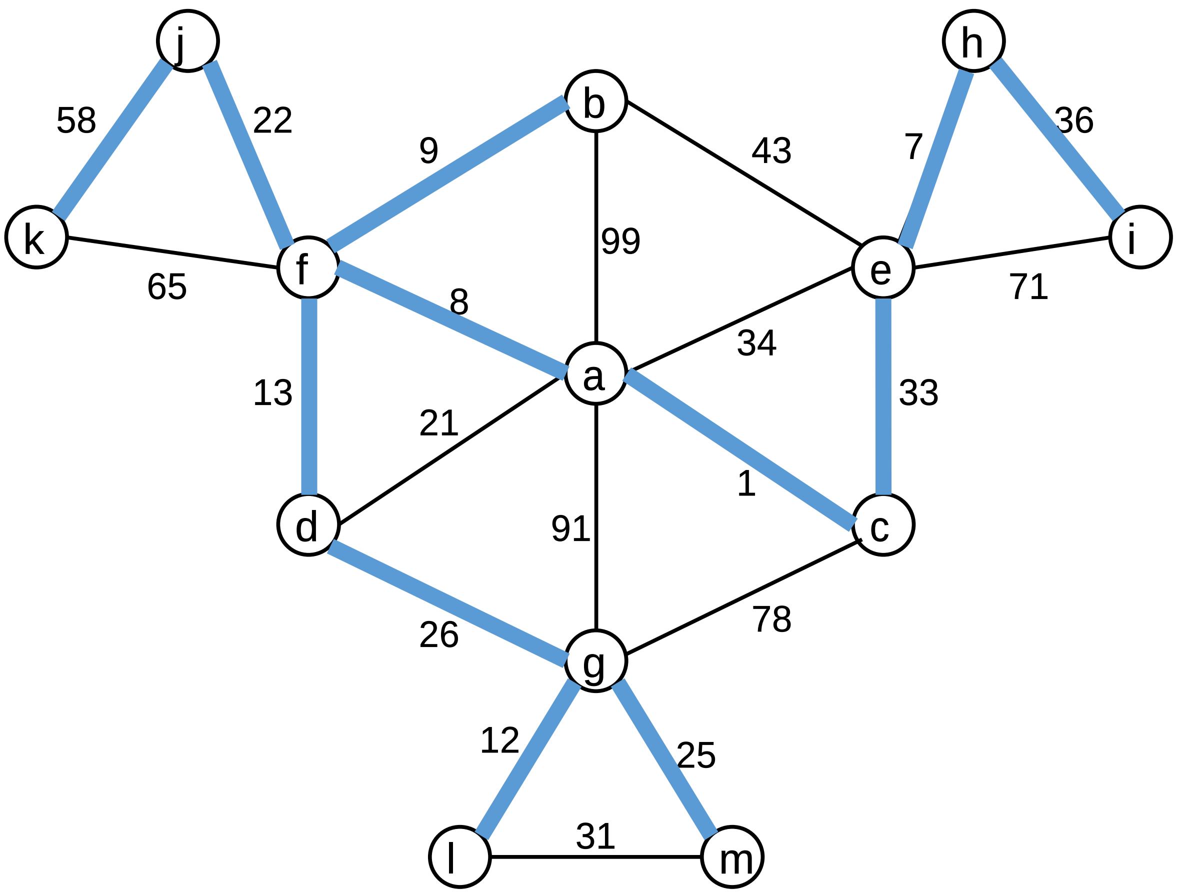
Textbook: Chapter 9.2

Context

- This is one of several “greedy algorithms” we will examine:
  - Minimum Spanning Tree of a graph
    - Prim’s algorithm
    - Kruskal’s algorithm
  - Shortest Paths from a Single Source in a graph
    - Dijkstra’s algorithm
  - Graph coloring

Kruskal's (overview)

- Repeatedly add a minimum-weight edge that does not introduce a cycle
- Example:



Kruskal’s algorithm (basic idea)

```
Kruskal(G)
    sort edges of E in ascending order by weight
    V_T ← V                              // T has all the vertices of G
    E_T ← ∅                              // start with no edges in T
    count ← 0
    k ← 0                                // index over edges of G
    while count < |V|-1 do               // done when T has this many edges
        k ← k + 1
        if E_T ∪ {e_k} is acyclic        // safe to add this edge to T?
            E_T ← E_T ∪ {e_k}            // ...then add it
            count ← count + 1
    return T = (V_T, E_T)
```

These two bits are “efficiency challenges”

Kruskal's algorithm: Implementation challenges

1. Sort the edges
   - We know several O(nlogn) methods
   - Which will serve us well?
2. Determine if adding an edge would create a cycle
   - Maybe use a DFS or BFS to test for a cycle?
     - These are $O(N^2)$ and we have to do it $O(N)$ times
     - Can we improve on $O(N^3)$?

     - The answer is Yes, with a clever data structure

Disjoint Subsets (aka "Union-Find")

- A collection of disjoint subsets – any element can only be in one subset at any time
- Operations on a DS:
  - Makeset(x) – creates a new subset with the element x
  - Find(x) – returns the subset that contains x
  - Union(x,y) – merges the subsets containing x and y together

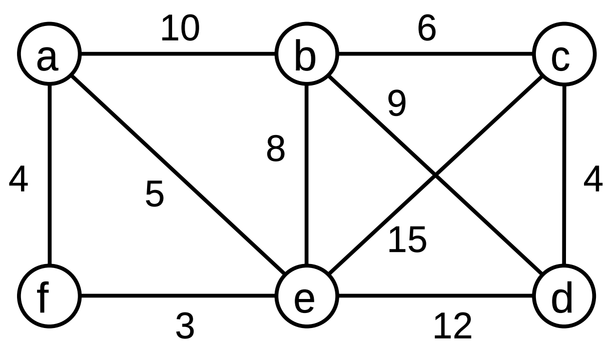
DS/Union-Find Example

```
for each element x in {1,2,3,4,5,6,7,8}
    makeset(x)
        →  DS is now {1} {2} {3} {4} {5} {6} {7} {8}
union(2,7)
        →  DS is now {1} {2,7} {3} {4} {5} {6} {8}
union(1,4)
        →  DS is now {1,4} {2,7} {3} {5} {6} {8}
y ← find(4)
        →  y is now {1,4}
union(y,3)
        →  DS is now {1,4,3} {2,7} {5} {6} {8}
x ← find(1)
        →  x is now {1,4,3}
y ← find(7)
        →  y is now {2,7}
union(x,y)
        →  DS is now {1,4,3,2,7} {5} {6} {8}
```

Kruskal’s with disjoint subsets

- Maintain DS of vertices in the spanning tree T
- Initially each vertex is a separate subset
- When an edge (u,v) is added to T:
  - DS.union(u,v)
- Each subset is a connected component
  - It’s also a tree – a subset of the eventual MST
- If u,v are in the same subset *do not add edge*
  - It would create a cycle
- At the end there will be only one subset in DS
  - T is a single connected component

Another Kruskal example (using disjoint subsets)



- After the initialization
- PQ contains sorted list of edges
- DS has one subset for each vertex

PQ

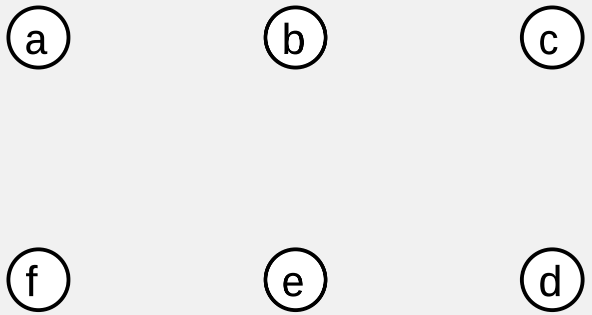
<u>key: value</u>

3:ef
4:af
4:cd
5:ae
6:bc
8:be
9:bd
10:ab
12:de
15:ce

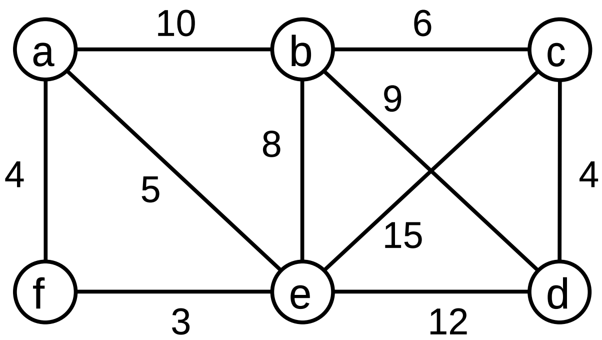
Subsets

{a} {b} {c} {d} {e} {f}

Solution



Another Kruskal example (using disjoint subsets)



- After iteration 1
- edge ef has been added
- e, f subsets merged

PQ

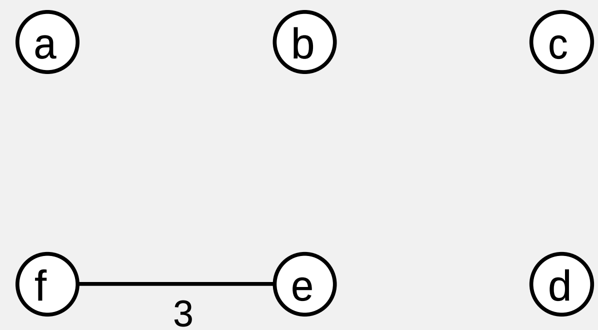
<u>key: value</u>

`3:ef`
`4:af`
`4:cd`
`5:ae`
`6:bc`
`8:be`
`9:bd`
`10:ab`
`12:de`
`15:ce`

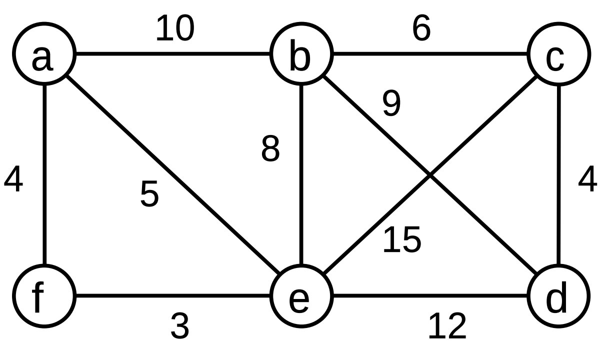
Subsets

`{a} {b} {c} {d} {e} {f}`
`{a} {b} {c} {d} {e,f}`

Solution



Another Kruskal example (using disjoint subsets)



- After iteration 2
- edge af has been added
- a, f subsets merged

PQ

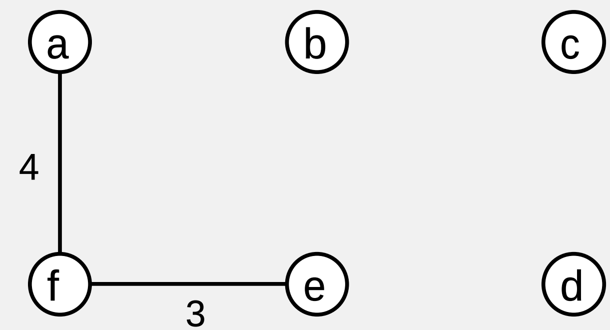
key: value

3:ef
4:af
4:cd
5:ae
6:bc
8:be
9:bd
10:ab
12:de
15:ce

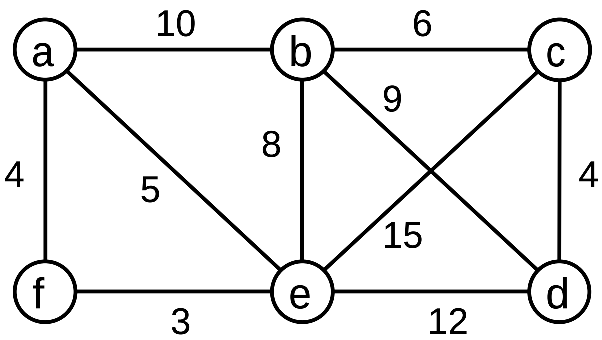
Subsets

{a} {b} {c} {d} {e} {f}
{a} {b} {c} {d} {e,f}
{a,e,f} {b} {c} {d}

Solution



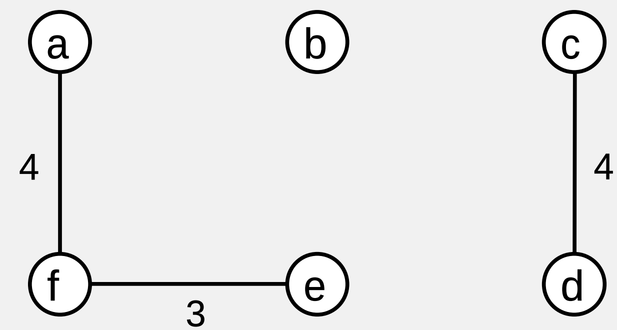
Another Kruskal example (using disjoint subsets)



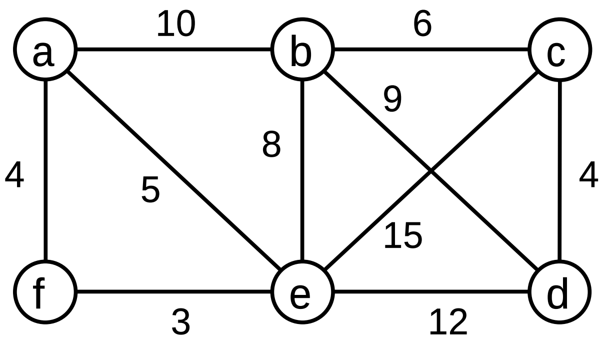
- After iteration 3
- edge cd has been added
- c, d subsets merged

PQ	Subsets
<u>key: value</u>	{a} {b} {c} {d} {e} {f}
3:ef	{a} {b} {c} {d} {e,f}
4:af	{a,e,f} {b} {c} {d}
4:cd	**{a,e,f} {b} {c,d}**
5:ae	
6:bc	
8:be	
9:bd	
10:ab	
12:de	
15:ce	

Solution



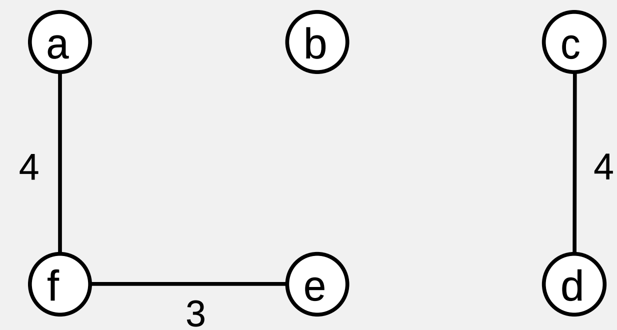
Another Kruskal example (using disjoint subsets)



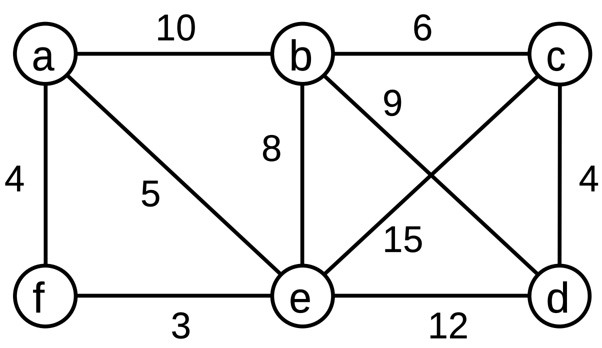
- *No change in iteration 4*
- *a and e are in the same subset*
- *edge ae is not added because it would cause a cycle*

PQ	Subsets
key: value	{a} {b} {c} {d} {e} {f}
3:ef	{a} {b} {c} {d} {e,f}
4:af	{a,e,f} {b} {c} {d}
4:cd	{a,e,f} {b} {c,d}
5:ae	**{a,e,f} {b} {c,d}**
6:bc	
8:be	
9:bd	
10:ab	
12:de	
15:ce	

Solution



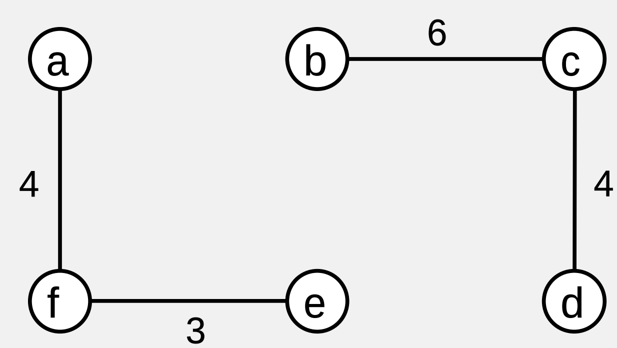
Another Kruskal example (using disjoint subsets)



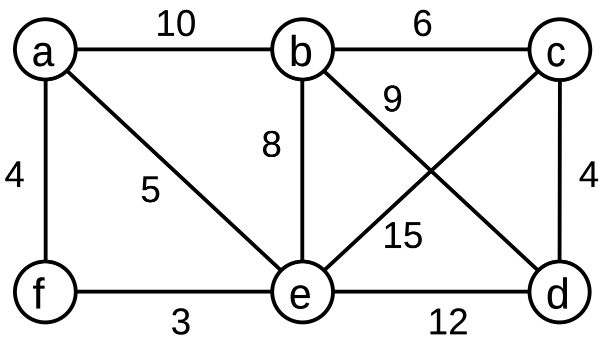
- After iteration 5
- edge bc has been added
- b, c subsets merged

PQ	Subsets
key: value	{a} {b} {c} {d} {e} {f}
3:ef	{a} {b} {c} {d} {e,f}
4:af	{a,e,f} {b} {c} {d}
4:cd	{a,e,f} {b} {c,d}
5:ae	{a,e,f} {b} {c,d}
6:bc	**{a,e,f} {b,c,d}**
8:be	
9:bd	
10:ab	
12:de	
15:ce	

Solution



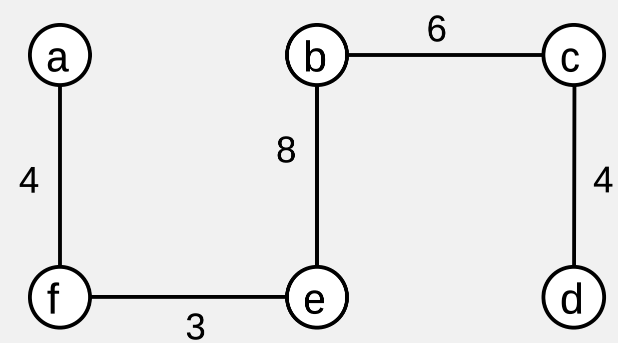
Another Kruskal example (using disjoint subsets)



- After iteration 6
- edge be has been added
- N-1 edges added, main loop ends
- algorithm returns solution

PQ	Subsets
key: value	{a} {b} {c} {d} {e} {f}
3:ef	{a} {b} {c} {d} {e,f}
4:af	{a,e,f} {b} {c} {d}
4:cd	{a,e,f} {b} {c,d}
5:ae	{a,e,f} {b} {c,d}
6:bc	{a,e,f} {b,c,d}
8:be	**{a,e,f,b,c,d}**
9:bd	
10:ab	
12:de	
15:ce	

Solution



Kruskal's algorithm with PQ + disjoint subsets

```
Algorithm Kruskal(G)
   Add all vertices in G to T     // add v's but don't add e's
   Create a priority queue PQ     // will hold candidate edges
   Create a collection DS         // disjoint subsets
   for each vertex v in G do
      DS.makeset(v)
   for each edge e in G do
      PQ.add(e.weight, e )        // PQ of edges by min weight
   while T has fewer than n-1 edges do
       (u,v) ← PQ.removeMin()    // get next smallest edge
       cu ← DS.find(u)
       cv ← DS.find(v)
       if cu ≠ cv then            // be sure u,v are not in
          T.addEdge(u,v)               // the same subset
          DS.union(cu, cv)
   return T
```

Efficiency of Kruskal's

- With an efficient union-find algorithm, the slowest thing is the initial sort on edge weights
  - $O(|E| \log|E|)$