Algorithms

Review of MST Algorithms

Disjoint-Set Union

Amortized Analysis

Review: MST Algorithms

- *In a connected, weighted, undirected graph, will the edge with the lowest weight be in the MST? Why or why not?*
- Yes:
  - If T is MST of G, and A $\subseteq$ T is a subtree of T, and (u,v) is the min-weight edge connecting A to V-A, then $(u,v) \in$ T
  - The lowest-weight edge must be in the tree (A=$\varnothing$)

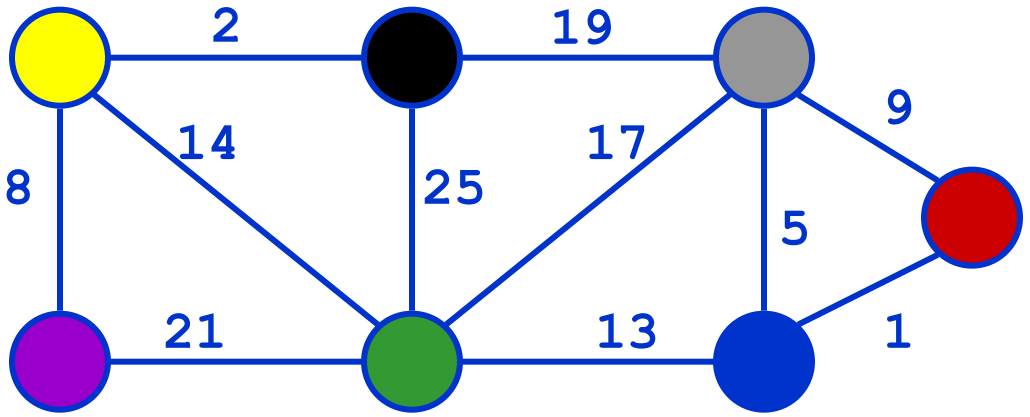
Review: MST Algorithms

- *What do the disjoint sets in Kruskal's algorithm represent?*
- A: Parts of the graph we have connected up together so far

Kruskal's Algorithm

```
Kruskal()
{
    T = ∅;
    for each v ∈ V
        MakeSet(v);
    sort E by increasing edge weight w
    for each (u,v) ∈ E (in sorted order)
        if FindSet(u) ≠ FindSet(v)
            T = T U {{u,v}};
            Union(FindSet(u), FindSet(v));
}
```

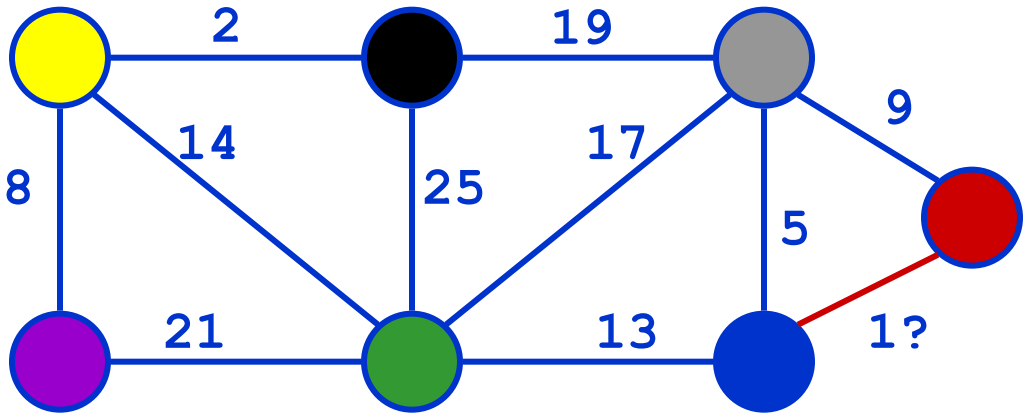
Run the algorithm:



Kruskal's Algorithm

```
Kruskal()
{
    T = ∅;
    for each v ∈ V
        MakeSet(v);
    sort E by increasing edge weight w
    for each (u,v) ∈ E (in sorted order)
        if FindSet(u) ≠ FindSet(v)
            T = T U {{u,v}};
            Union(FindSet(u), FindSet(v));
}
```

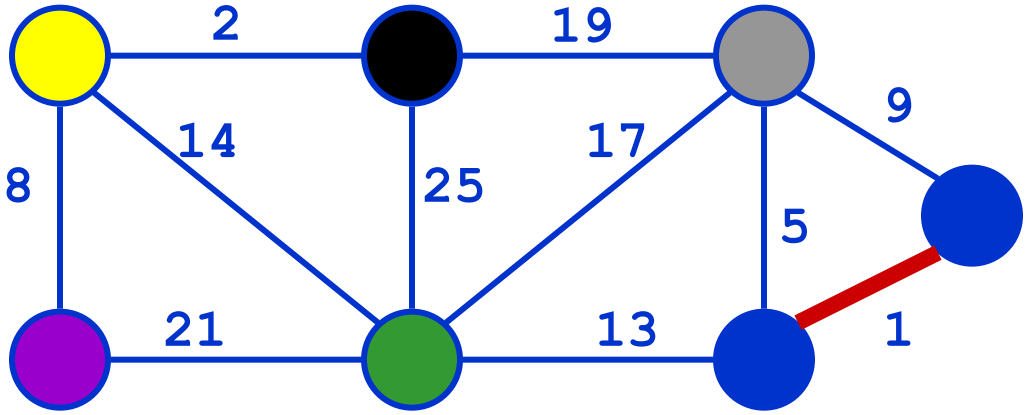
Run the algorithm:



Kruskal's Algorithm

```
Kruskal()
{
    T = ∅;
    for each v ∈ V
        MakeSet(v);
    sort E by increasing edge weight w
    for each (u,v) ∈ E (in sorted order)
        if FindSet(u) ≠ FindSet(v)
            T = T U {{u,v}};
            Union(FindSet(u), FindSet(v));
}
```

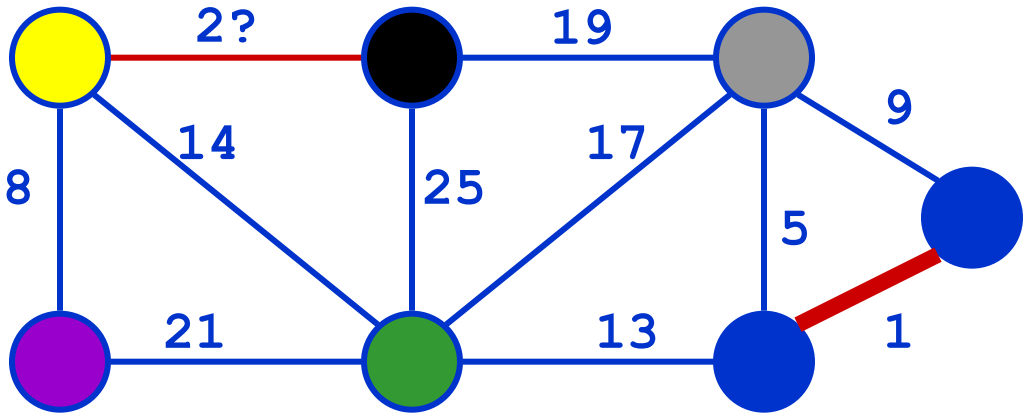
Run the algorithm:



Kruskal's Algorithm

```
Kruskal()
{
    T = ∅;
    for each v ∈ V
        MakeSet(v);
    sort E by increasing edge weight w
    for each (u,v) ∈ E (in sorted order)
        if FindSet(u) ≠ FindSet(v)
            T = T U {{u,v}};
            Union(FindSet(u), FindSet(v));
}
```

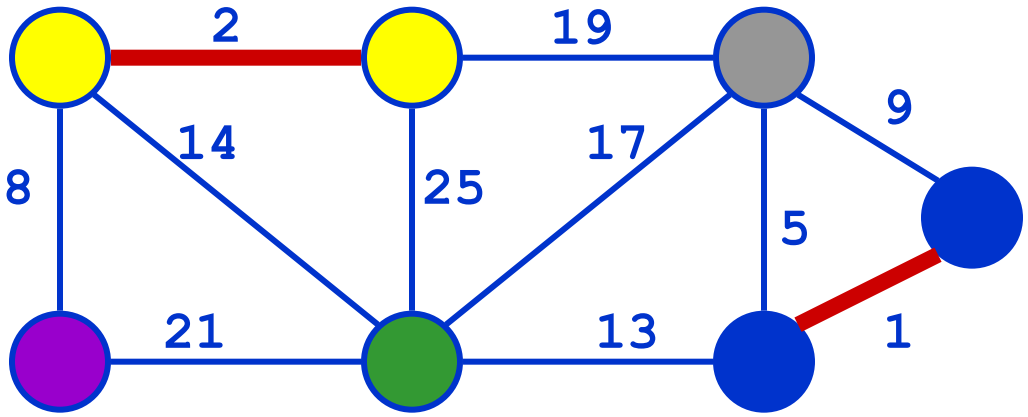
Run the algorithm:



Kruskal's Algorithm

```
Kruskal()
{
    T = ∅;
    for each v ∈ V
        MakeSet(v);
    sort E by increasing edge weight w
    for each (u,v) ∈ E (in sorted order)
        if FindSet(u) ≠ FindSet(v)
            T = T U {{u,v}};
            Union(FindSet(u), FindSet(v));
}
```

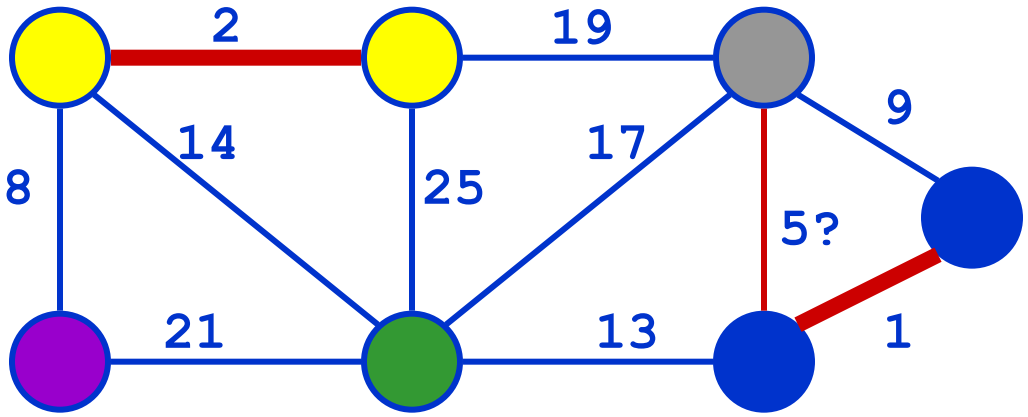
Run the algorithm:



Kruskal's Algorithm

```
Kruskal()
{
    T = ∅;
    for each v ∈ V
        MakeSet(v);
    sort E by increasing edge weight w
    for each (u,v) ∈ E (in sorted order)
        if FindSet(u) ≠ FindSet(v)
            T = T U {{u,v}};
            Union(FindSet(u), FindSet(v));
}
```

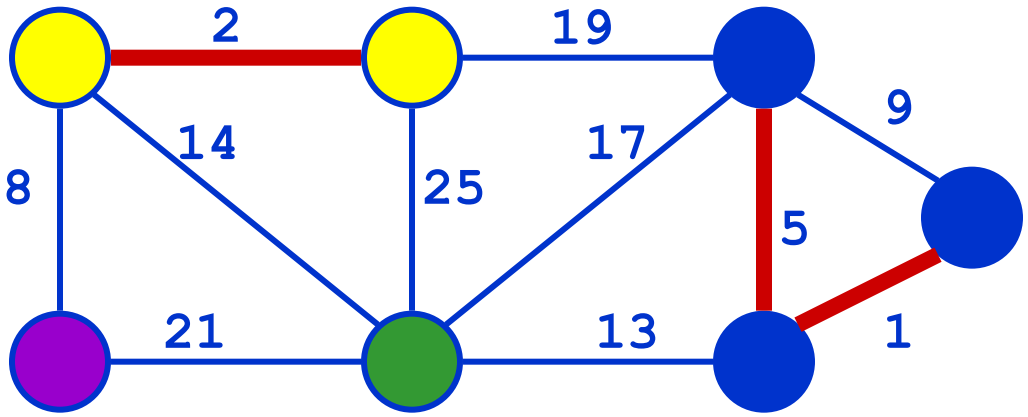
Run the algorithm:



Kruskal's Algorithm

```
Kruskal()
{
    T = ∅;
    for each v ∈ V
        MakeSet(v);
    sort E by increasing edge weight w
    for each (u,v) ∈ E (in sorted order)
        if FindSet(u) ≠ FindSet(v)
            T = T U {{u,v}};
            Union(FindSet(u), FindSet(v));
}
```

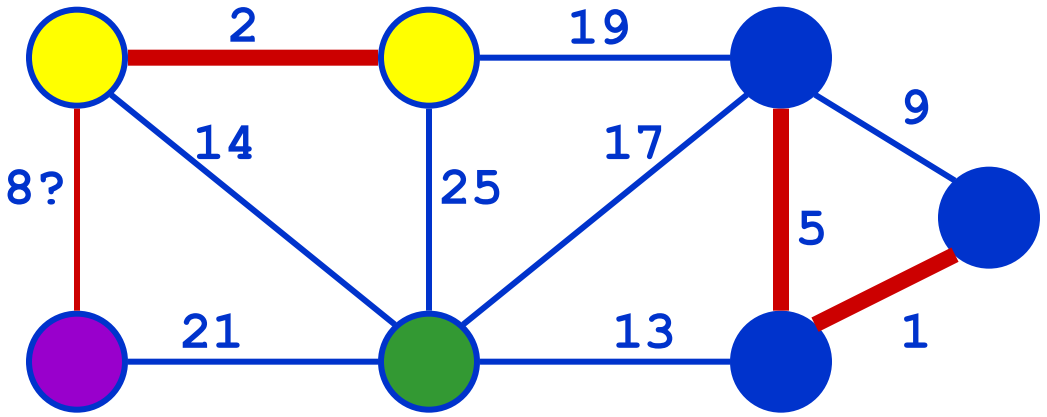
Run the algorithm:



Kruskal's Algorithm

```
Kruskal()
{
    T = ∅;
    for each v ∈ V
        MakeSet(v);
    sort E by increasing edge weight w
    for each (u,v) ∈ E (in sorted order)
        if FindSet(u) ≠ FindSet(v)
            T = T U {{u,v}};
            Union(FindSet(u), FindSet(v));
}
```

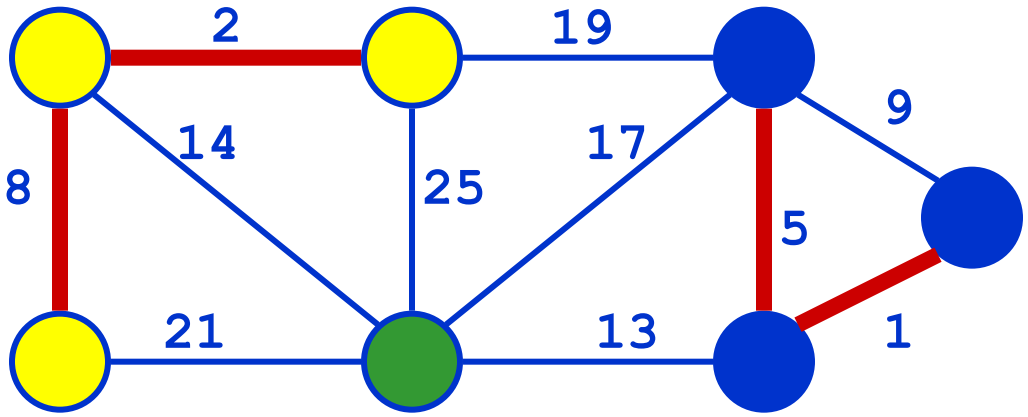
Run the algorithm:



Kruskal's Algorithm

```
Kruskal()
{
    T = ∅;
    for each v ∈ V
        MakeSet(v);
    sort E by increasing edge weight w
    for each (u,v) ∈ E (in sorted order)
        if FindSet(u) ≠ FindSet(v)
            T = T U {{u,v}};
            Union(FindSet(u), FindSet(v));
}
```

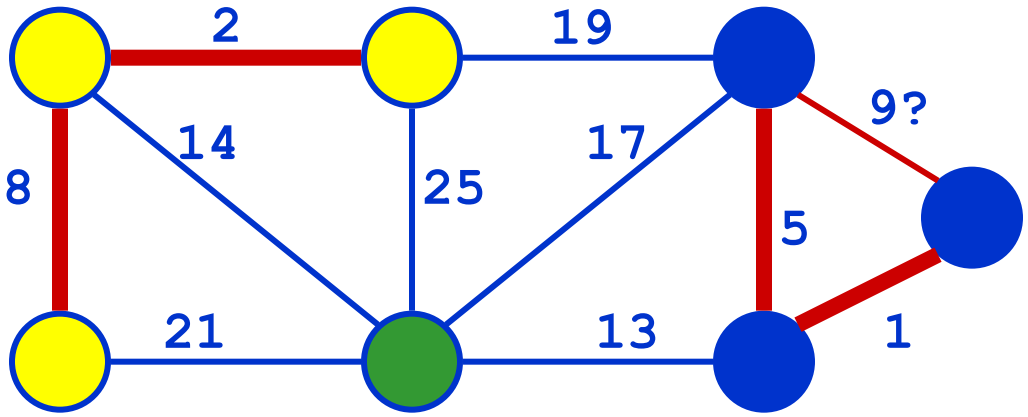
Run the algorithm:



Kruskal's Algorithm

```
Kruskal()
{
    T = ∅;
    for each v ∈ V
        MakeSet(v);
    sort E by increasing edge weight w
    for each (u,v) ∈ E (in sorted order)
        if FindSet(u) ≠ FindSet(v)
            T = T U {{u,v}};
            Union(FindSet(u), FindSet(v));
}
```

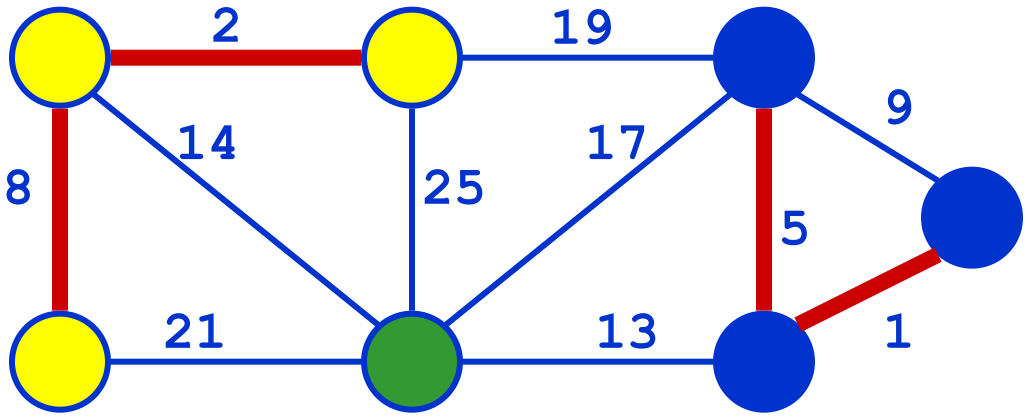
Run the algorithm:



Kruskal's Algorithm

```
Kruskal()
{
    T = ∅;
    for each v ∈ V
        MakeSet(v);
    sort E by increasing edge weight w
    for each (u,v) ∈ E (in sorted order)
        if FindSet(u) ≠ FindSet(v)
            T = T U {{u,v}};
            Union(FindSet(u), FindSet(v));
}
```

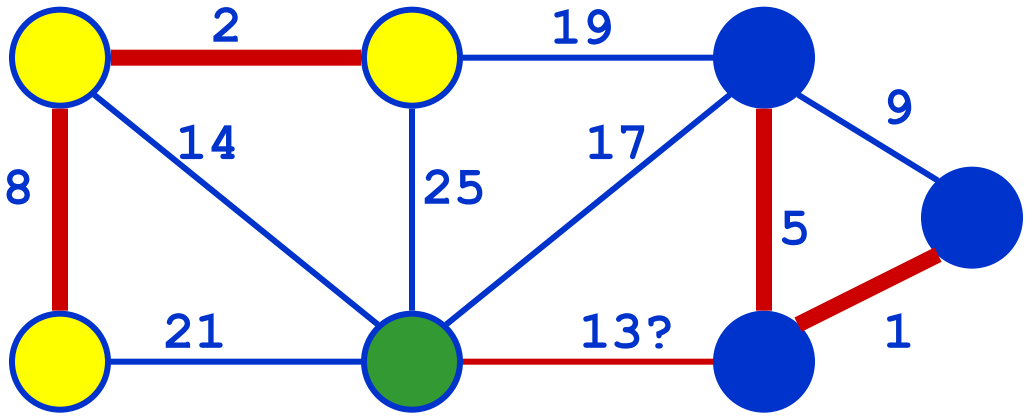
Run the algorithm:



Kruskal's Algorithm

```
Kruskal()
{
    T = ∅;
    for each v ∈ V
        MakeSet(v);
    sort E by increasing edge weight w
    for each (u,v) ∈ E (in sorted order)
        if FindSet(u) ≠ FindSet(v)
            T = T U {{u,v}};
            Union(FindSet(u), FindSet(v));
}
```

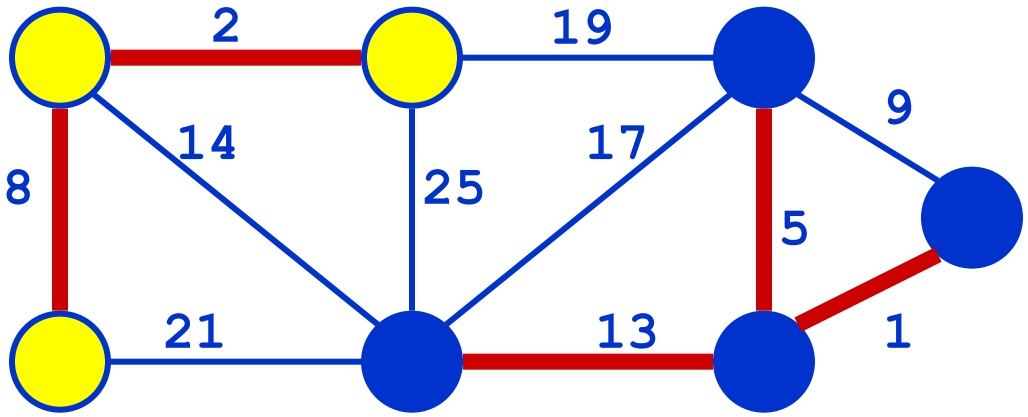
Run the algorithm:



Kruskal's Algorithm

```
Kruskal()
{
    T = ∅;
    for each v ∈ V
        MakeSet(v);
    sort E by increasing edge weight w
    for each (u,v) ∈ E (in sorted order)
        if FindSet(u) ≠ FindSet(v)
            T = T U {{u,v}};
            Union(FindSet(u), FindSet(v));
}
```

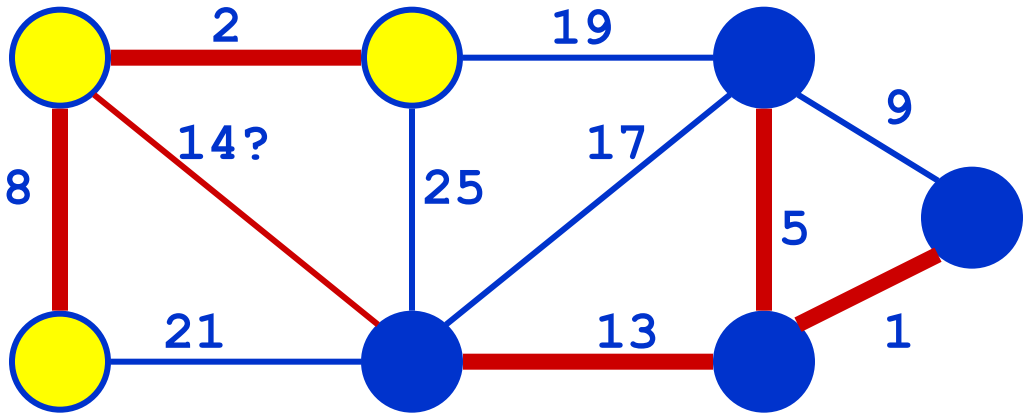
Run the algorithm:



Kruskal's Algorithm

```
Kruskal()
{
    T = ∅;
    for each v ∈ V
        MakeSet(v);
    sort E by increasing edge weight w
    for each (u,v) ∈ E (in sorted order)
        if FindSet(u) ≠ FindSet(v)
            T = T U {{u,v}};
            Union(FindSet(u), FindSet(v));
}
```

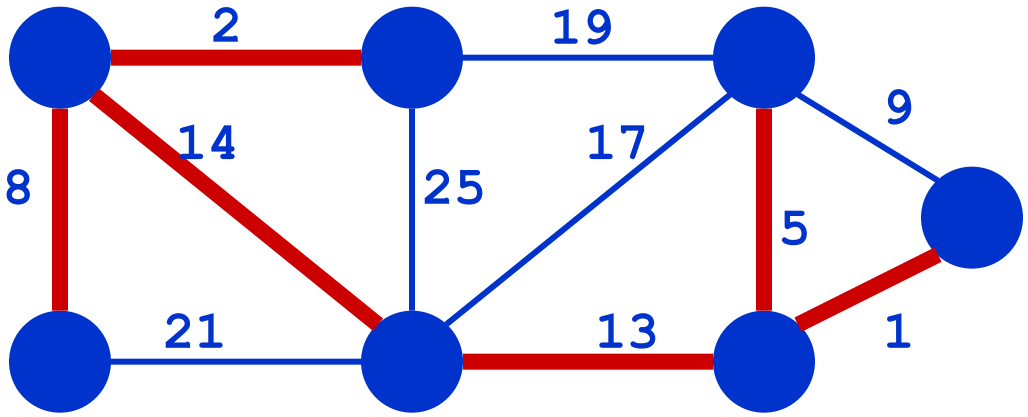
Run the algorithm:



Kruskal's Algorithm

```
Kruskal()
{
    T = ∅;
    for each v ∈ V
        MakeSet(v);
    sort E by increasing edge weight w
    for each (u,v) ∈ E (in sorted order)
        if FindSet(u) ≠ FindSet(v)
            T = T U {{u,v}};
            Union(FindSet(u), FindSet(v));
}
```

Run the algorithm:



Review: Shortest-Path Algorithms

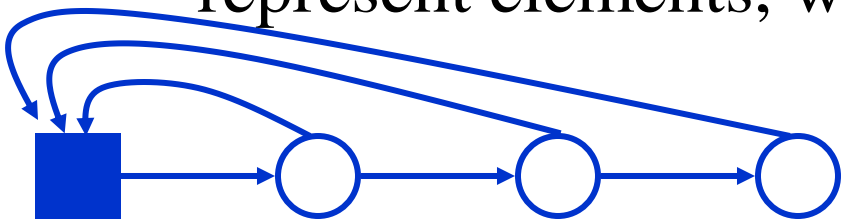
- *How does the Bellman-Ford algorithm work?*
- *How can we do better for DAGs?*
- *Under what conditions can we use Dijkstra's algorithm?*

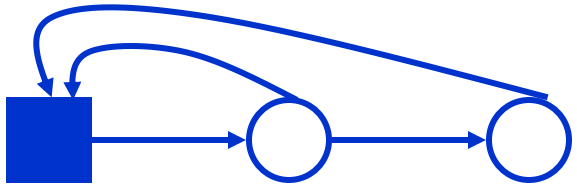
Review: Running Time of Kruskal's Algorithm

- Expensive operations:
  - Sort edges: O(E lg E)
  - O(V) MakeSet()'s
  - O(E) FindSet()'s
  - O(V) Union()'s
- Upshot:
  - Comes down to efficiency of disjoint-set operations, particularly Union()

Disjoint Set Union

- So how do we represent disjoint sets?
  - Naïve implementation: use a linked list to represent elements, with pointers back to set:





    - MakeSet(): O(1)
    - FindSet(): O(1)
    - Union(A,B): "Copy" elements of A into set B by adjusting elements of A to point to B: O(A)
  - *How long could n Union()s take?*

Disjoint Set Union: Analysis

- Worst-case analysis: $O(n^2)$ time for n Union's

Union(S_1, S_2)	"copy"	1 element
Union(S_2, S_3)	"copy"	2 elements
…		
Union(S_{n-1}, S_n)	"copy"	n-1 elements
		$O(n^2)$

- Improvement: always copy smaller into larger
  - *How long would above sequence of Union's take?*
  - Worst case: n Union's take O(n lg n) time
  - Proof uses amortized analysis

Amortized Analysis of Disjoint Sets

- If elements are copied from the smaller set into the larger set, an element can be copied at most $lg\ n$ times
  - Worst case: Each time copied, element in smaller set

1st time	resulting set size	≥ 2
2nd time		≥ 4
…		
(lg n)th time		$\geq n$

Amortized Analysis of Disjoint Sets

- Since we have n elements each copied at most lg n times, n Union()'s takes O(n lg n) time
- Therefore we say the *amortized cost* of a Union() operation is O(lg n)
- This is the *aggregate method* of amortized analysis:
  - n operations take time T(n)
  - Average cost of an operation = T(n)/n

Amortized Analysis: Accounting Method

- *Accounting method*
  - Charge each operation an amortized cost
  - Amount not used stored in "bank"
  - Later operations can used stored money
  - Balance must not go negative
- Book also discusses *potential method*
  - But we won't worry about it here