



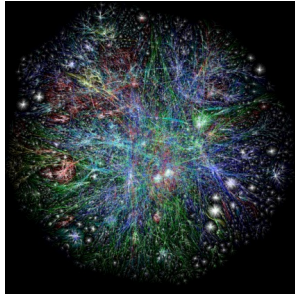
# CS 225

## Data Structures

*October 28 – Minimum Spanning Tree (Prim)*

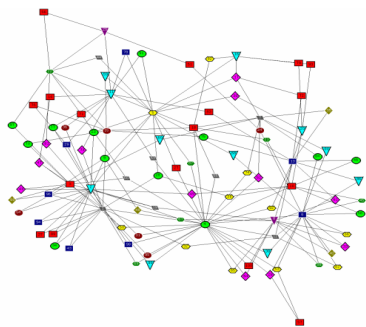
*G Carl Evans*

# Graphs



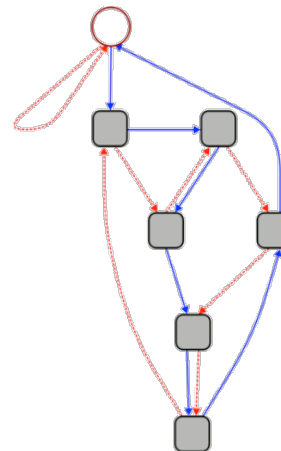
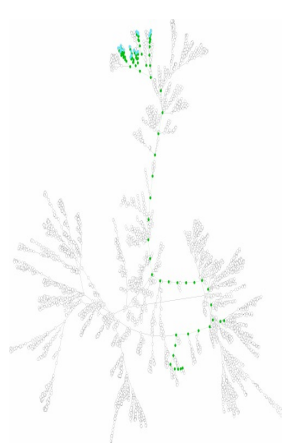
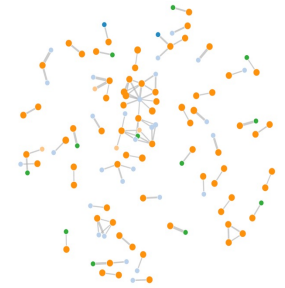
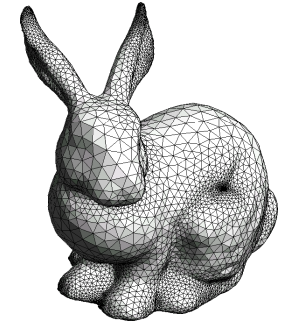
HAMLET

TROILUS AND CRESSIDA



To study all of these structures:

1. A common vocabulary
2. Graph implementations
3. Graph traversals
4. Graph algorithms



```
heapify(int*, unsigned int):  
  push rbp  
  mov rsi, rbp  
  sub rbp, 30  
  mov dword ptr [rbp + 8], rsi  
  mov dword ptr [rbp + 12], esi  
  mov dword ptr [rbp + 16], 1  
  jmp .LBB_4
```

```
heapify(int*, unsigned int):  
  mov rax, qword ptr [rbp + 8]  
  mov ecx, dword ptr [rbp + 12]  
  mov ebx, ecx  
  mov rax, qword ptr [rax + 4*ebx]  
  mov rax, qword ptr [rbp + 8]  
  mov esi, dword ptr [rbp + 12]  
  shr esi, 1  
  mov ebx, esi  
  mov ecx, qword ptr [rax + 4*ebx]  
  jmp .LBB_3
```

```
heapify(int*, unsigned int):  
  mov rax, qword ptr [rbp + 8]  
  mov rax, qword ptr [rbp + 8]  
  mov ecx, qword ptr [rbp + 12]  
  mov esi, dword ptr [rbp + 12]  
  shr esi, 1  
  mov ebx, esi  
  mov rax, qword ptr [rax + 4*ebx]  
  mov rax, qword ptr [rbp + 8]  
  mov esi, dword ptr [rbp + 12]  
  shr esi, 1  
  mov ebx, esi  
  mov rax, qword ptr [rax + 4*ebx]  
  mov rax, qword ptr [rbp + 8]  
  mov ecx, qword ptr [rbp + 12]  
  mov esi, ecx  
  mov rax, qword ptr [rbp + 12]  
  jmp .LBB_4
```

```
.LBB_3:  
  jmp .LBB_4
```

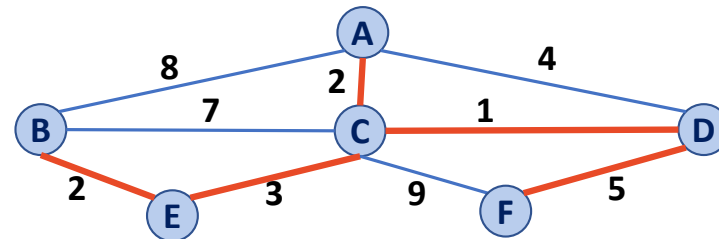
```
.LBB_4:  
  add rbp, 30  
  pop rbp  
  ret
```

# Minimum Spanning Tree Algorithms

**Input:** Connected, undirected graph  $G$  with edge weights (unconstrained, but must be additive)

**Output:** A graph  $G'$  with the following properties:

- $G'$  is a spanning graph of  $G$
- $G'$  is a tree (connected, acyclic)
- $G'$  has a minimal total weight among all spanning trees

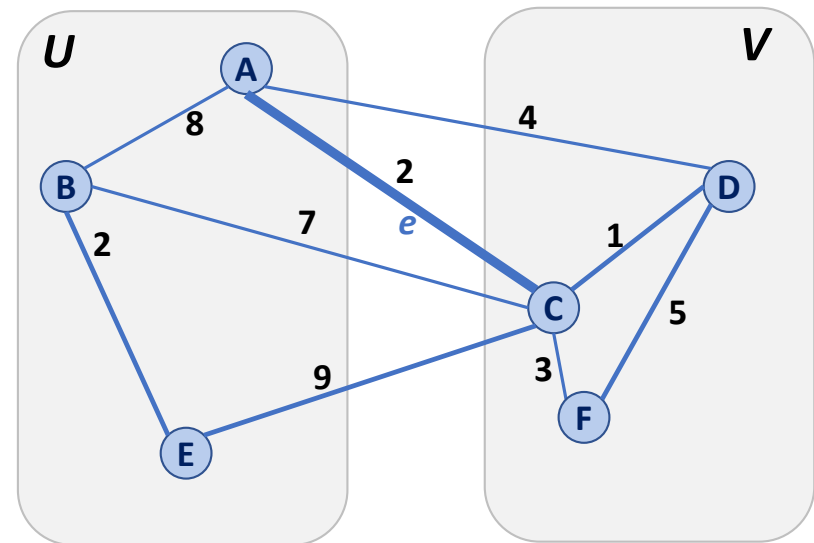


## Partition Property

Consider an arbitrary partition of the vertices on  $\mathbf{G}$  into two subsets  $\mathbf{U}$  and  $\mathbf{V}$ .

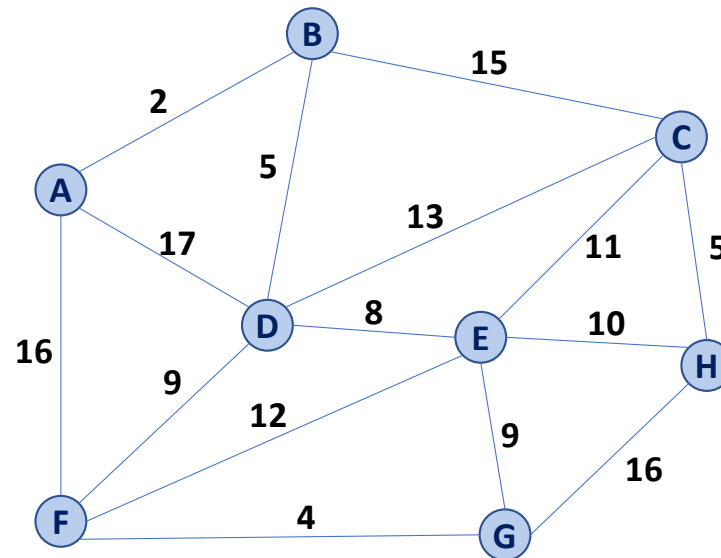
Let  $\mathbf{e}$  be an edge of minimum weight across the partition.

Then  $\mathbf{e}$  is part of some minimum spanning tree.

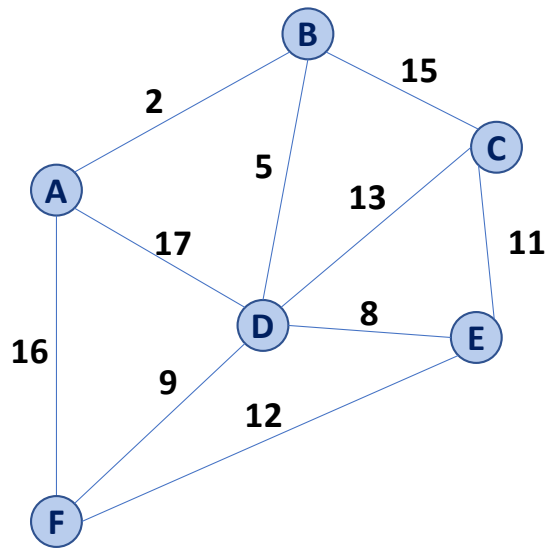


# Partition Property

The partition property suggests an algorithm:



# Prim's Algorithm



```
1 PrimMST(G, s):
2   Input: G, Graph;
3         s, vertex in G, starting vertex
4   Output: T, a minimum spanning tree (MST) of G
5
6   foreach (Vertex v : G):
7     d[v] = +inf
8     p[v] = NULL
9   d[s] = 0
10
11   PriorityQueue Q // min distance, defined by d[v]
12   Q.buildHeap(G.vertices())
13   Graph T // "labeled set"
14
15   repeat n times:
16     Vertex m = Q.removeMin()
17     T.add(m)
18     foreach (Vertex v : neighbors of m not in T):
19       if cost(v, m) < d[v]:
20         d[v] = cost(v, m)
21         p[v] = m
22
23   return T
```

```
1 PrimMST(G, s):
2   Input: G, Graph;
3         s, vertex in G, starting vertex
4   Output: T, a minimum spanning tree (MST) of G
5
6   foreach (Vertex v : G):
7     d[v] = +inf
8     p[v] = NULL
9   d[s] = 0
10
11  PriorityQueue Q // min distance, defined by d[v]
12  Q.buildHeap(G.vertices())
13  Graph T // "labeled set"
14
15  repeat n times:
16    Vertex m = Q.removeMin()
17    T.add(m)
18    foreach (Vertex v : neighbors of m not in T):
19      if cost(v, m) < d[v]:
20        d[v] = cost(v, m)
21        p[v] = m
22
23  return T
```

# Prim's Algorithm

```
6 PrimMST(G, s):
7   foreach (Vertex v : G):
8     d[v] = +inf
9     p[v] = NULL
10  d[s] = 0
11
12  PriorityQueue Q // min distance, defined by d[v]
13  Q.buildHeap(G.vertices())
14  Graph T          // "labeled set"
15
16  repeat n times:
17    Vertex m = Q.removeMin()
18    T.add(m)
19    foreach (Vertex v : neighbors of m not in T):
20      if cost(v, m) < d[v]:
21        d[v] = cost(v, m)
22        p[v] = m
```

	Adj. Matrix	Adj. List
Heap		
Unsorted Array		



# Prim's Algorithm

**Sparse Graph:**

**Dense Graph:**

```
6 PrimMST(G, s):
7   foreach (Vertex v : G):
8     d[v] = +inf
9     p[v] = NULL
10  d[s] = 0
11
12  PriorityQueue Q // min distance, defined by d[v]
13  Q.buildHeap(G.vertices())
14  Graph T          // "labeled set"
15
16  repeat n times:
17    Vertex m = Q.removeMin()
18    T.add(m)
19    foreach (Vertex v : neighbors of m not in T):
20      if cost(v, m) < d[v]:
21        d[v] = cost(v, m)
22        p[v] = m
```

	Adj. Matrix	Adj. List
Heap	$O(n \lg(n) + n^2 \lg(n))$	$O(n \lg(n) + m \lg(n))$
Unsorted Array	$O(n^2)$	$O(n^2)$



## MST Algorithm Runtime:

We know that MSTs are always run on a minimally connected graph:

$$n-1 \leq m \leq n(n-1) / 2$$

$$O(n) \leq O(m) \leq O(n^2)$$



## MST Algorithm Runtime:

- Kruskal's Algorithm:

$$O(n + m \lg(n))$$

Sparse Graph:

Dense Graph:

- Prim's Algorithm:

$$O(n \lg(n) + m \lg(n))$$

Sparse Graph:

Dense Graph:

# Suppose I have a new heap:

	Binary Heap	Fibonacci Heap
Remove Min	$O(\lg(n))$	$O(\lg(n))$
Decrease Key	$O(\lg(n))$	$O(1)^*$

## What's the updated running time?

```
PrimMST(G, s):
6   foreach (Vertex v : G):
7     d[v] = +inf
8     p[v] = NULL
9     d[s] = 0
10
11  PriorityQueue Q // min distance, defined by d[v]
12  Q.buildHeap(G.vertices())
13  Graph T          // "labeled set"
14
15  repeat n times:
16    Vertex m = Q.removeMin()
17    T.add(m)
18    foreach (Vertex v : neighbors of m not in T):
19      if cost(v, m) < d[v]:
20        d[v] = cost(v, m)
21        p[v] = m
```



## MST Algorithm Runtimes:

- Kruskal's Algorithm:  
 **$O(m \lg(n))$**

- Prim's Algorithm:  
 **$O(n \lg(n) + m \lg(n))$**

# Shortest Path

