



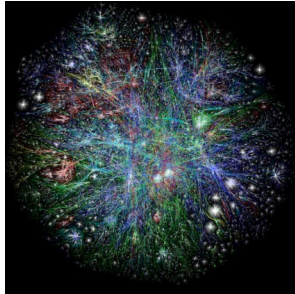
CS 225

Data Structures

April 25 – Dijkstra's Algorithm

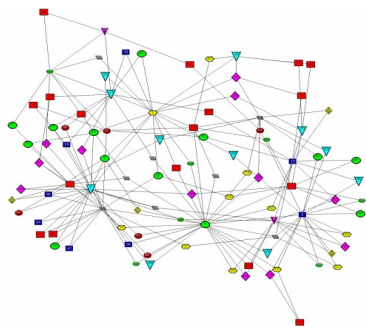
G Carl Evans

Graphs

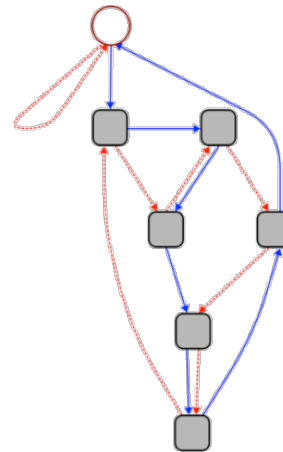
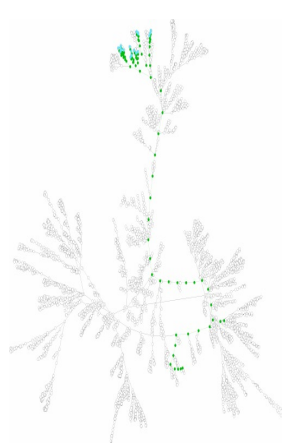
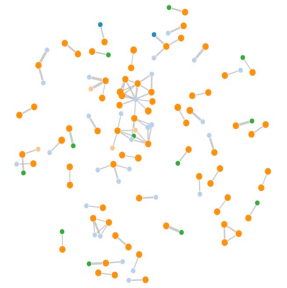
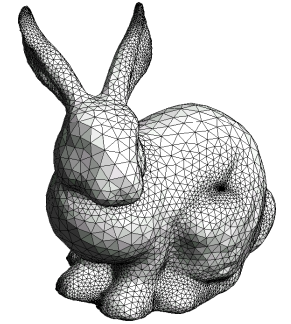


HAMLET

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, 20  
  mov dword ptr [rbp + 8], rdi  
  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 ecx, ecx  
  mov rax, qword ptr [rax + 4*rdi]  
  mov rax, qword ptr [rbp + 8]  
  mov esi, dword ptr [rbp + 12]  
  shr esi, 1  
  mov esi, esi  
  mov ecx, esi  
  mov ecx, dword ptr [rax + 4*rdi]  
  jmp .LBB_3
```

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

```
.LBB_3:  
  jmp .LBB_1
```

```
.LBB_4:  
  add rbp, 20  
  pop rbp  
  ret
```



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



Final Big-O MST Algorithm Runtimes:

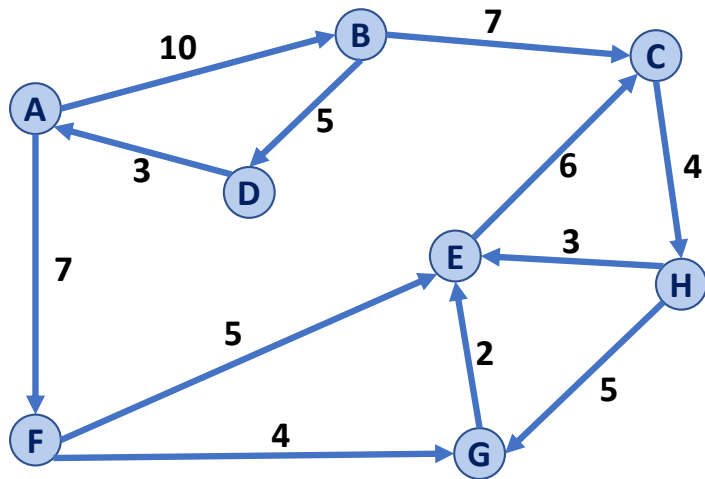
- Kruskal's Algorithm:
 $O(m \lg(n))$

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

Shortest Path



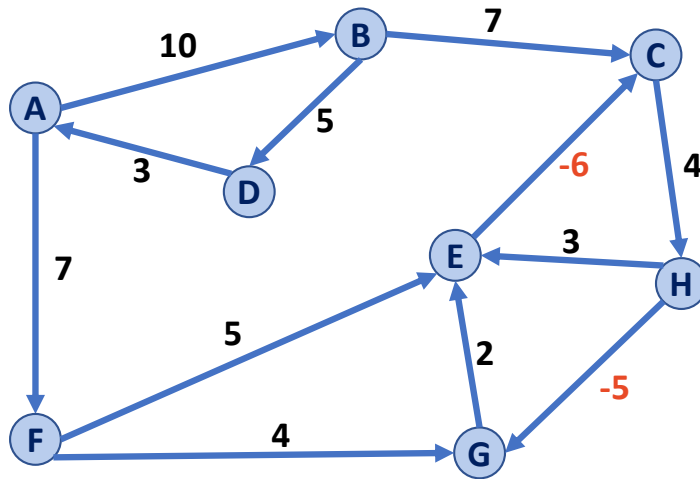
Dijkstra's Algorithm (SSSP)



```
DijkstraSSSP(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 u = Q.removeMin()
17    T.add(u)
18    foreach (Vertex v : neighbors of u not in T):
19      if _____ < d[v]:
20        d[v] = _____
21        p[v] = m
```

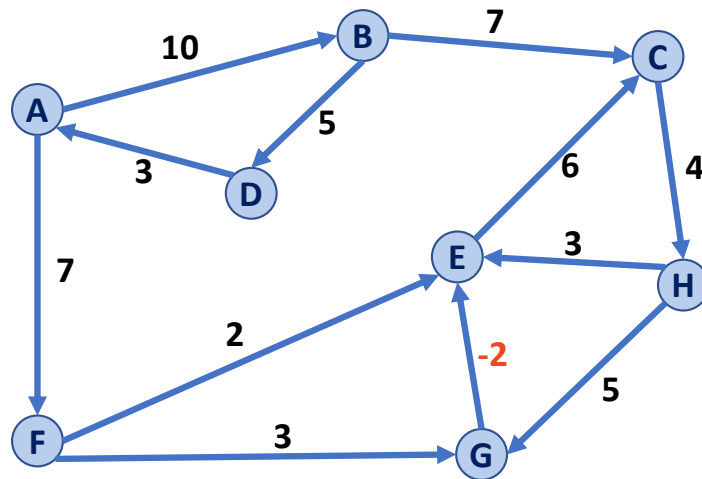
Dijkstra's Algorithm (SSSP)

What about negative weight cycles?



Dijkstra's Algorithm (SSSP)

What about negative weight edges, without negative weight cycles?



Dijkstra's Algorithm (SSSP)

What is the running time?

```
DijkstraSSSP(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 u = Q.removeMin()
17      T.add(u)
18      foreach (Vertex v : neighbors of u not in T):
19          if _____ < d[v]:
20              d[v] = _____
21              p[v] = m
```