

Data Structures

MST Part 3 and Single Source Shortest Path

CS 225

December 1, 2023

Brad Solomon & G Carl Evans



UNIVERSITY OF
ILLINOIS
URBANA - CHAMPAIGN

Department of Computer Science

Staff Hiring (CS 225 / CS 277)

Both have applications on <https://opportunities.cs.illinois.edu/>

Both have significant opportunities for dev work

If you regularly interact with course staff, ask them for a rec!

In your application, include things that make you stand out

Learning Objectives

Compare Kruskal and Prim MST Algorithms

Introduce Single-Source Shortest Path Problem

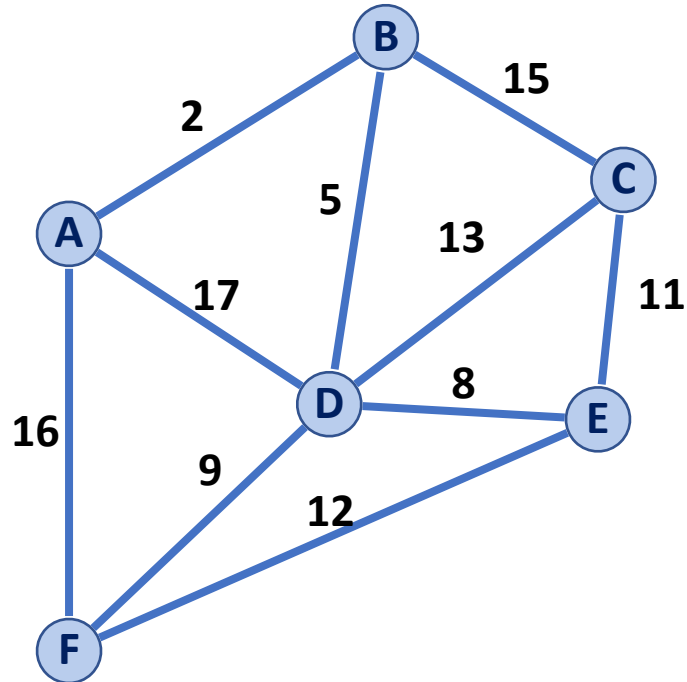
Discuss Dijkstra's Algorithm

Kruskal's Algorithm

Priority Queue:	Total Running Time
Heap	$O(n + m + m \log(n))$
Sorted Array	$O(n + m \log(n) + m)$

```
1 KruskalMST(G):
2   DisjointSets forest
3   foreach (Vertex v : G):
4     forest.makeSet(v)
5
6   PriorityQueue Q // min edge weight
7   Q.buildFromGraph(G)
8
9   Graph T = (V, {})
10
11  while |T.edges()| < n-1:
12    Vertex (u, v) = Q.removeMin()
13    if forest.find(u) != forest.find(v):
14      T.addEdge(u, v)
15      forest.union( forest.find(u),
16                  forest.find(v) )
17
18  return T
19
```

Prim's Algorithm



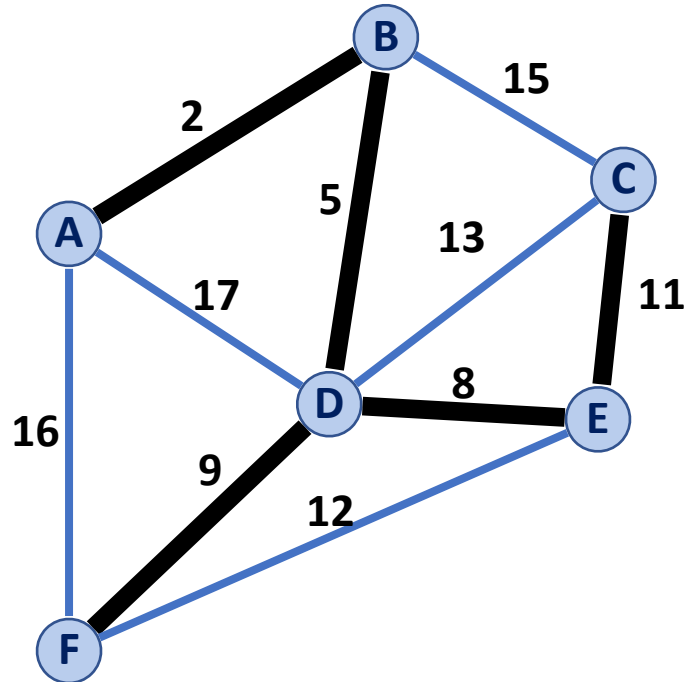
A	B	C	D	E	F
0, —	2, A	∞ , --	17, A	∞ , --	16, A

```

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.vertices()):
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



A	B	C	D	E	F
0, —	2, A	11, C	5, B	8, D	9, D

```
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.vertices()):
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
```

7-10: $O(n)$

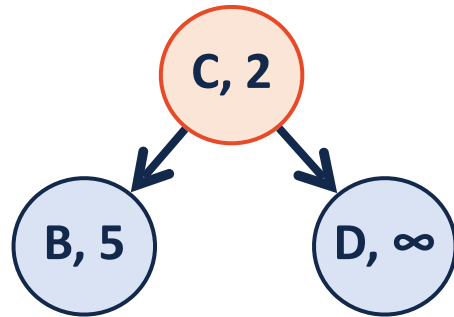
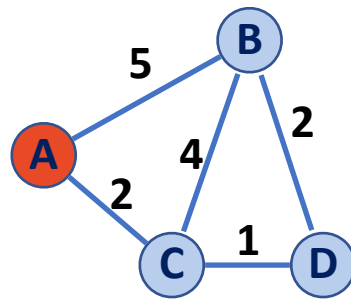
13: ???

17: ???

19-22: $O(m * ???)$

```
6 PrimMST(G, s):
7   foreach (Vertex v : G.vertices()):
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
23
```

A	B	C	D
0	5	2	∞



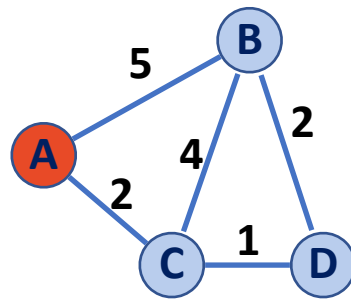
```

6 PrimMST(G, s):
7   foreach (Vertex v : G.vertices()):
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
23

```

	Adj. Matrix	Adj. List
Heap	$O(n) + \underline{\hspace{2cm}} + O(n^2) + \underline{\hspace{2cm}}$	$O(n) + \underline{\hspace{2cm}} + O(m) + \underline{\hspace{2cm}}$

(A, 0)
(D, ∞)
(C, 2)
(B, 5)



```

6 PrimMST(G, s):
7   foreach (Vertex v : G.vertices()):
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
23
  
```

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

Prim's Algorithm

Sparse Graph:

Dense Graph:

```
6 PrimMST(G, s):
7   foreach (Vertex v : G.vertices()):
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
23
```



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

MST Algorithm Runtime:

Kruskal's Algorithm:
 $O(n + m \log(n))$

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

Sparse Graph:

Dense Graph:

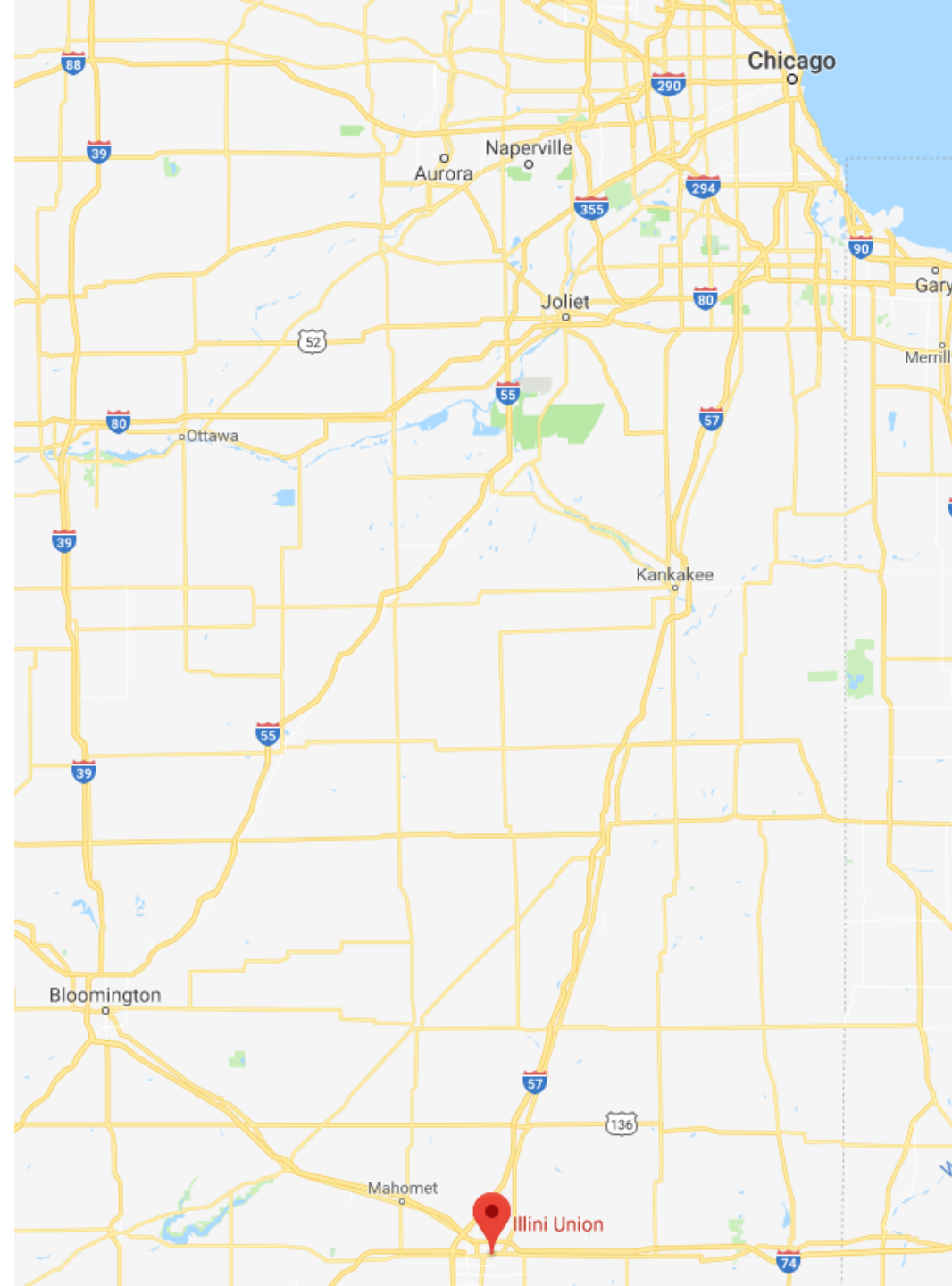
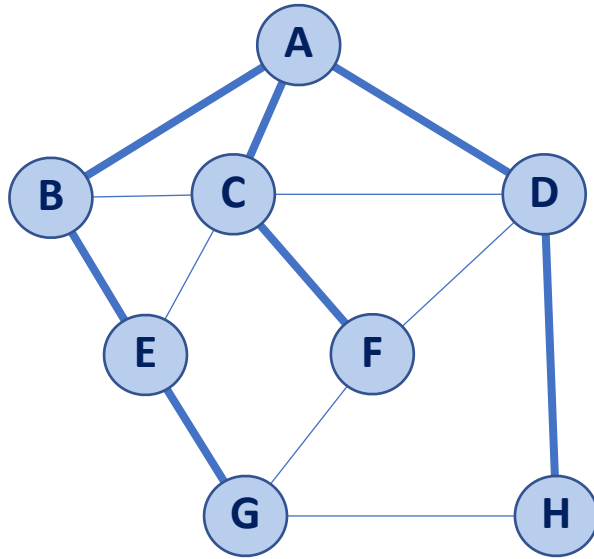
Suppose I have a new heap:

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

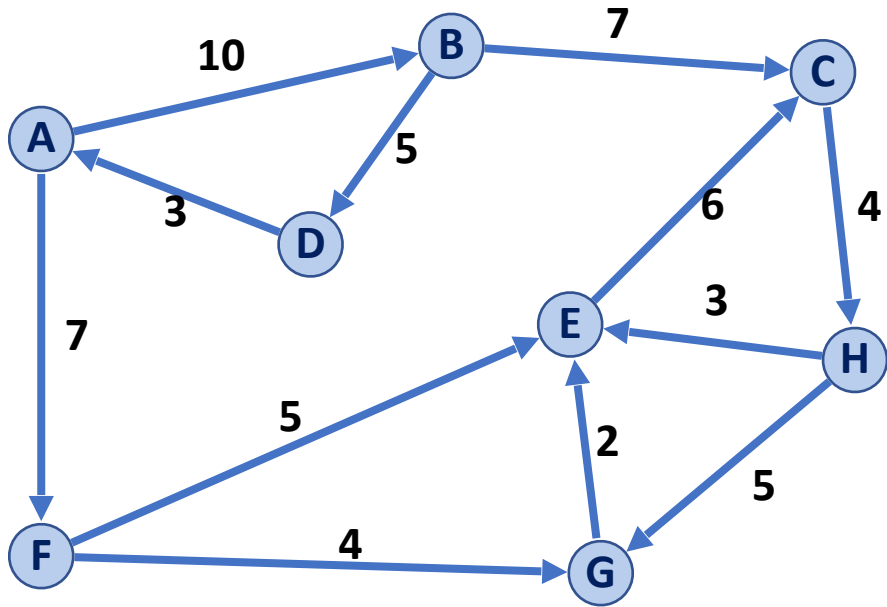
What's the updated running time?

```
PrimMST(G, s):
6   foreach (Vertex v : G.vertices()):
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
```

Shortest Path



Dijkstra's Algorithm (SSSP)



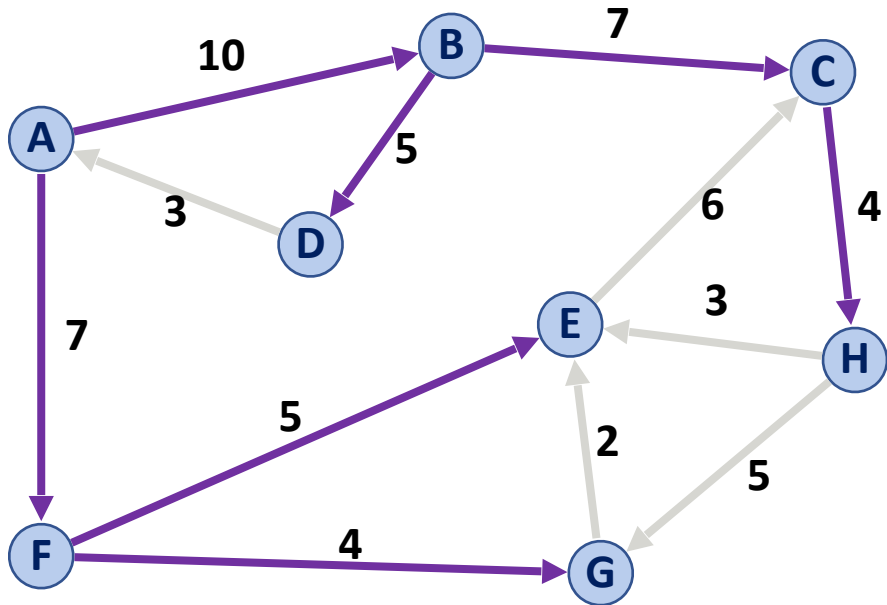
```

DijkstraSSSP(G, s):
6  foreach (Vertex v : G.vertices()):
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] = u
  
```

A	B	C	D	E	F	G	H
--							
0							



Dijkstra's Algorithm (SSSP)

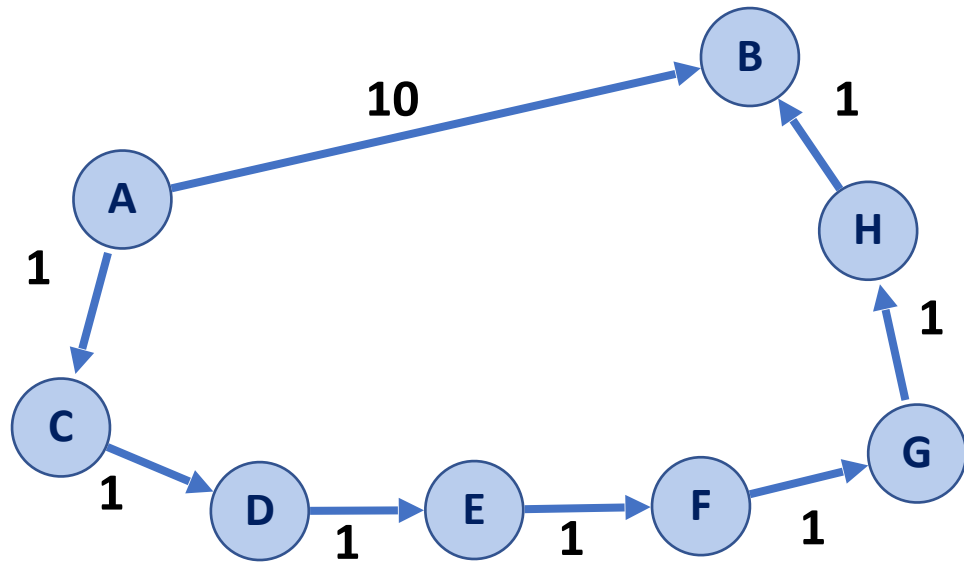


```
DijkstraSSSP(G, s):
6  foreach (Vertex v : G.vertices()):
7      d[v] = +inf
8      p[v] = NULL
9  d[s] = 0
10
11  PriorityQueue Q // min distance, defined by d[v]
12  Q.buildHeap(G.vertices())
13  Graph T          // "labeled set"
14
15  repeat n times:
16      Vertex u = Q.removeMin()
17      T.add(u)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

A	B	C	D	E	F	G	H
--	A	B	B	F	A	F	C
0	10	17	15	12	7	11	21

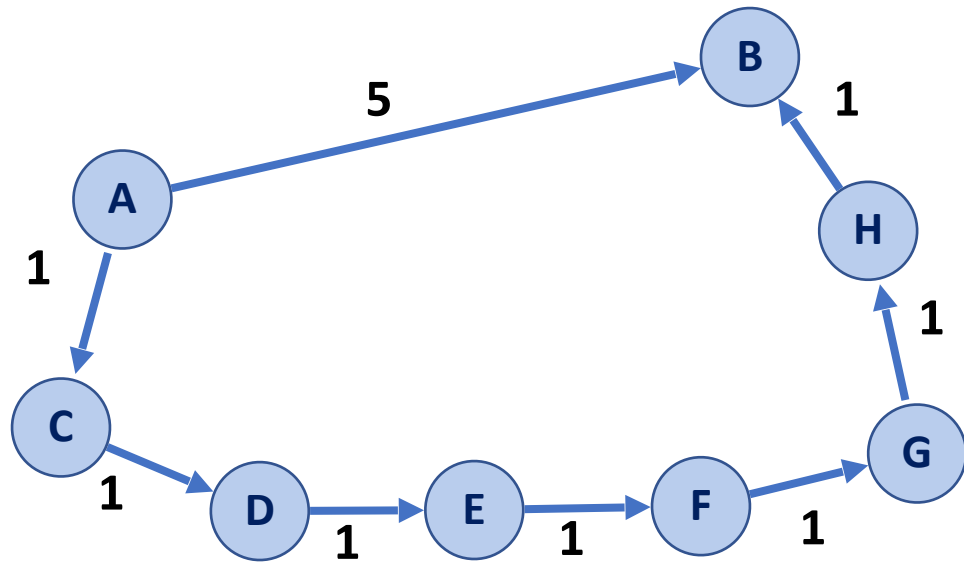
Dijkstra's Algorithm (SSSP)

When we will visit B in the following graph?



Dijkstra's Algorithm (SSSP)

When we will visit B in the following graph?



Dijkstra's Algorithm (SSSP)



What is the running time of Dijkstra's Algorithm?

```
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 cost(u, v) + d[u] < d[v]:
20              d[v] = cost(u, v) + d[u]
21              p[v] = m
22
23  return T
```