

Data Structures

MST3 & Single Source Shortest Path

CS 225

April 10, 2026

Brad Solomon

Enjoy EOH!



UNIVERSITY OF
ILLINOIS
URBANA - CHAMPAIGN



Department of Computer Science

Engineering Open House today / tomorrow!

Explore the many stations or events going on all day!

Learning Objectives

Finish discussion about Prim's runtime

A red underline is drawn under the text. A red arrow starts from the right side of the underline and points downwards towards the text below.

Compare Kruskal and Prim MST Algorithms

A red wavy underline is drawn under the text.

~~Introduce Single-Source Shortest Path Problem~~

A red underline is drawn under the text, which is also crossed out with a red strike-through line.

Discuss Dijkstra's Algorithm

Extend to All-Paths Shortest Path (if time)

Kruskal's Algorithm

$$|V| = n, |E| = m$$

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

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

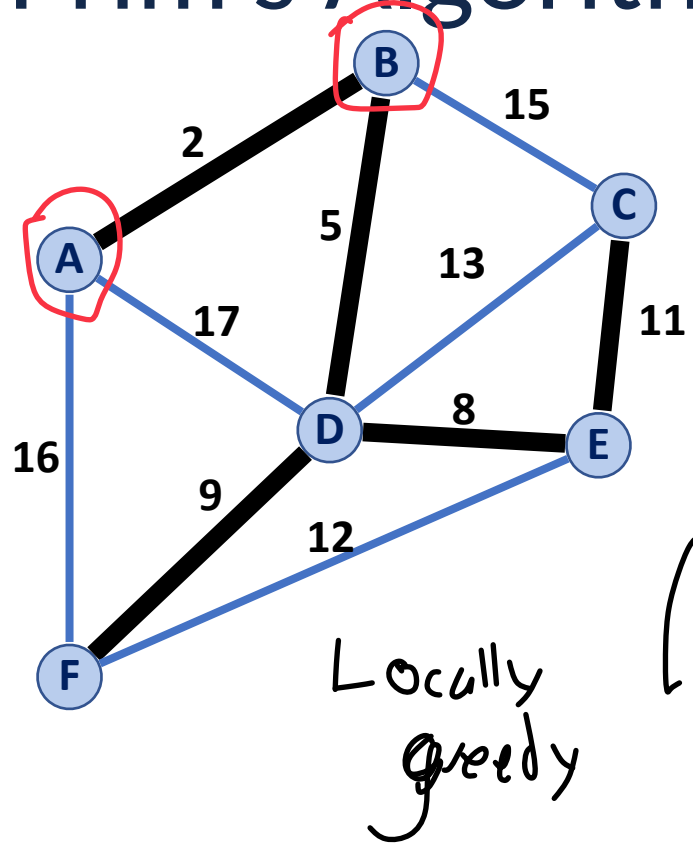
Globally Greedy,

```
1 KruskalMST(G):
2   DisjointSets forest
3   foreach (Vertex v : G.vertices()):
4     forest.makeSet(v)
5
6   PriorityQueue Q // min edge weight
7   Q.buildFromGraph(G.edges())
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
```

← all edges



Prim's Algorithm



A	B	C	D	E	F
0, —	2, A	11, E	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

```

depth/dist
previous

update all old values

Prim's Big O

$$|V| = n, |E| = m$$

7 — 9: $O(n)$

Init

12—14: Construction of
MinHeap: $O(n)$ priority queue

Unsorted Array: $O(1)$

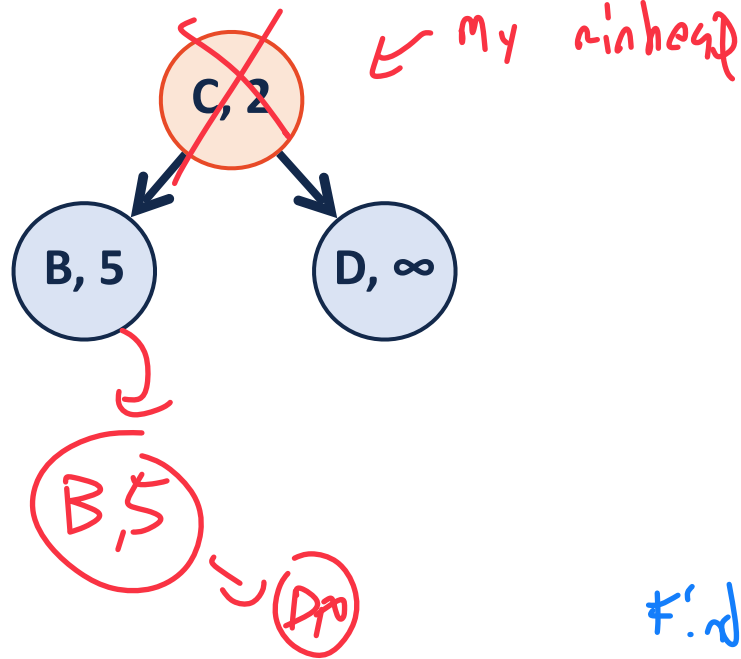
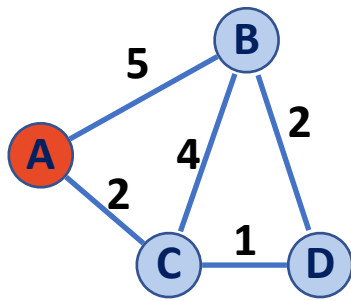
16—22: Complicated!

```
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
```

Depends on choice of **PriorityQueue** (MinHeap vs Unsorted Array)

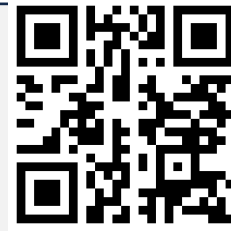
Depends on choice of **Graph** (Adjacency Matrix vs Adjacency List)

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
  
```



find neighbors is $O(n) \times n$

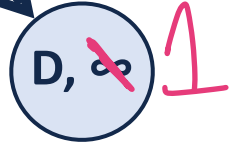
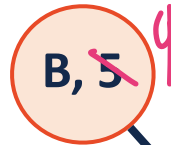
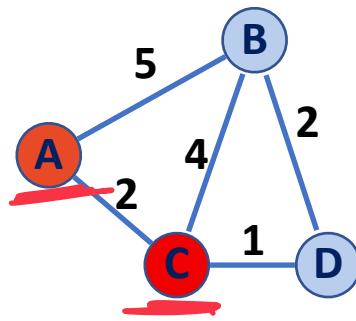
$$\sum \deg(u) = 2|E| = 2m$$

	Adj. Matrix	Adj. List
<u>Heap</u>	$O(n) + n \log(n) + O(n^2) +$	$O(n) + n \log(n) + O(m) +$

A	B	C	D
0	5	2, A	∞

4, B

1, B



```

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

```

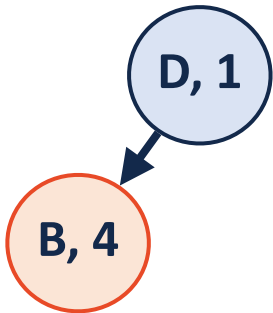
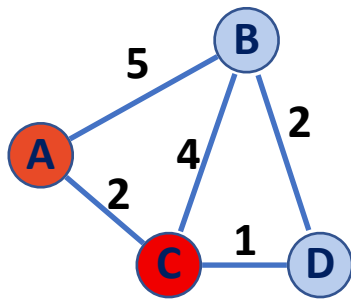
changing value is $O(1)$
 But... we are in heap!
 After each change heapify!

$O(\log n)$

$$\sum \deg(u) = 2|E| = O(n)$$

	Adj. Matrix	Adj. List
Heap	$O(n) + O(n \log n) + O(n^2) + O(n \log n)$	$O(n) + O(n \log n) + O(m) + O(n \log n)$

A	B	C	D
0	4	2, A	1



```

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)  1) Change minheap value
22        p[v] = m           2) HeapifyUp()
23

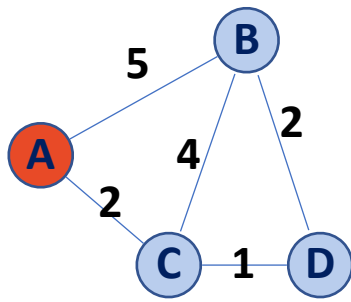
```



	Adj. Matrix	Adj. List
Heap	$O(n) + O(n \log n) + O(n^2) + O(m \log n)$	$O(n) + O(n \log n) + O(m) + O(m \log n)$

(A, 0)
(D, ∞)
(C, 2)
(B, 5)

↑
 $O(n)$
 ↓



↳ This array stores best dist to each node


What is the size of array?

```

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
  
```

Handwritten notes on code:

- Line 11: ← is now unsorted array
- Line 13: ~~Array~~ $O(1)$
- Line 16: $n \times$
- Line 17: To find min $O(n)$ (remove min $O(n)$)
- Line 21: ← ~~$O(1)$~~ ← $n-1$



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

Prim's Algorithm

Sparse Graph: $m \sim n$

Adj List Heap best

Dense Graph: $m \sim n^2$

Unsorted Array best

```

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

```

$O(n^2 + n^2 \log n)$

$O(n^2 \log n)$



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

← If dense matrix better!

MST Algorithm Runtime:

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

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

Sparse Graph: $m \sim n$

$$n + n \log(n)$$

$$n \log(n) + n \log(n)$$

Dense Graph: $m \sim n^2$

$$n + n^2 \log(n)$$

$$n^2 \log(n)$$

Suppose I have a new heap:

↳ 1980s wrote new heap

	Binary Heap	Fibonacci Heap
Remove Min	$O(\lg(n))$	$O(\lg(n))$
Decrease Key	$O(\lg(n))$	$O(1)^*$ ↖ Inverse Ackermann Pseudo-constant

What's Prim's updated running time?

w/ Fib heap

15-16

$O(n \lg n + m)$

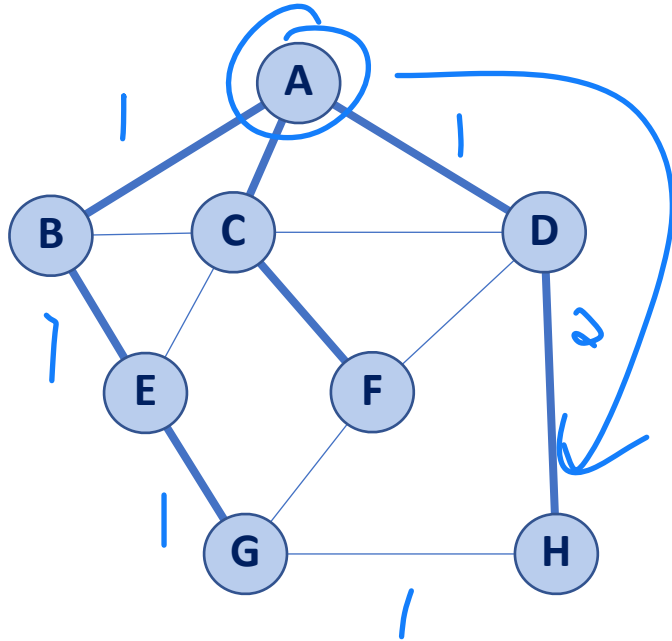
$O(n)$

$O(n)$

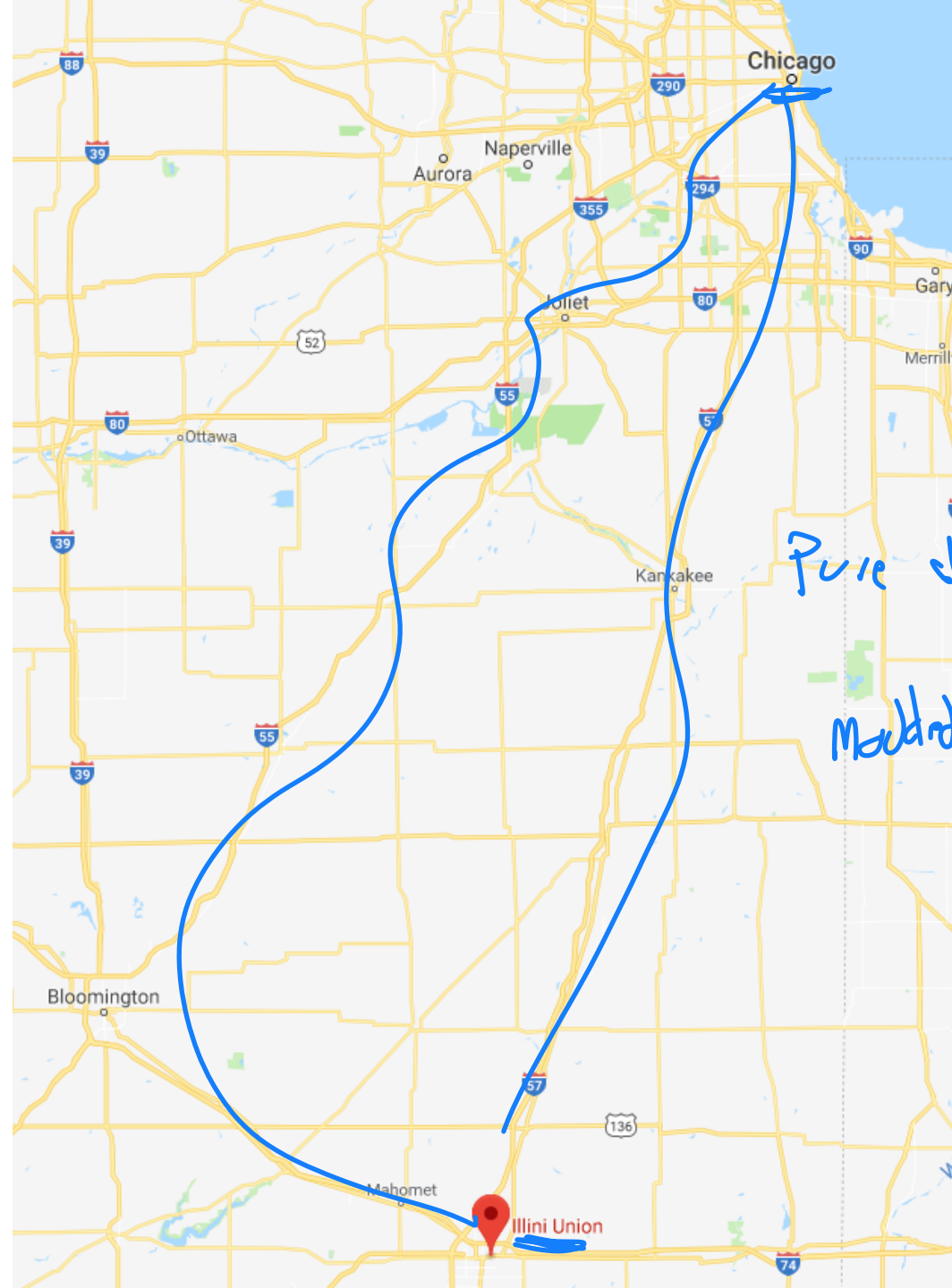
```

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: ← n ← lg n
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]: ← O(lg n) time binary heap
20              d[v] = cost(v, m) ← O(1)* time
21              p[v] = m
    
```

Shortest Path



Give me start
me end



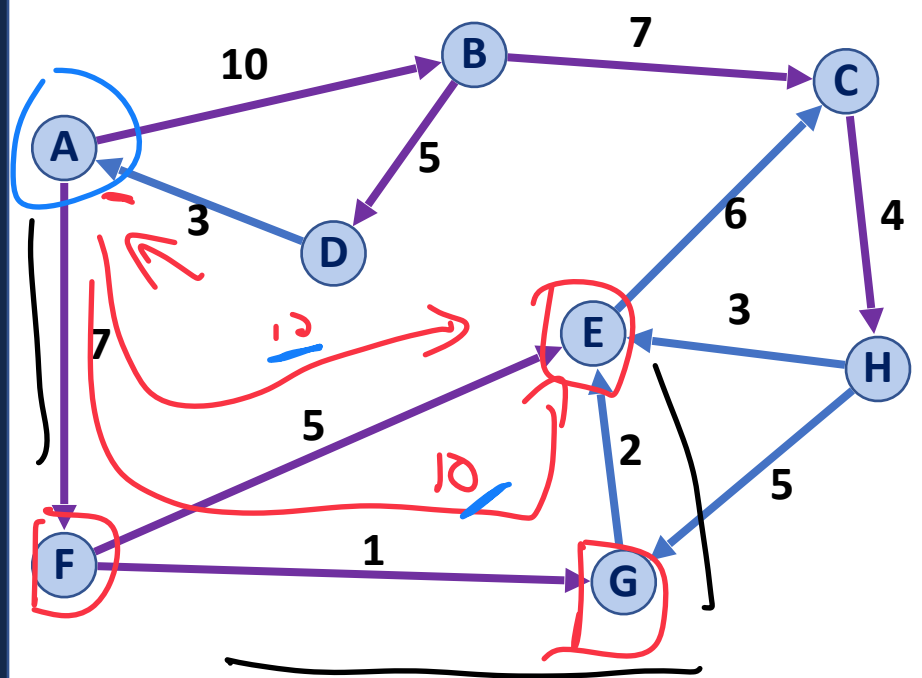
Weight on
each edge

Pure distance?

Modified by traffic

This is Prim's but...

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
    
```

Initialize

locally greedy

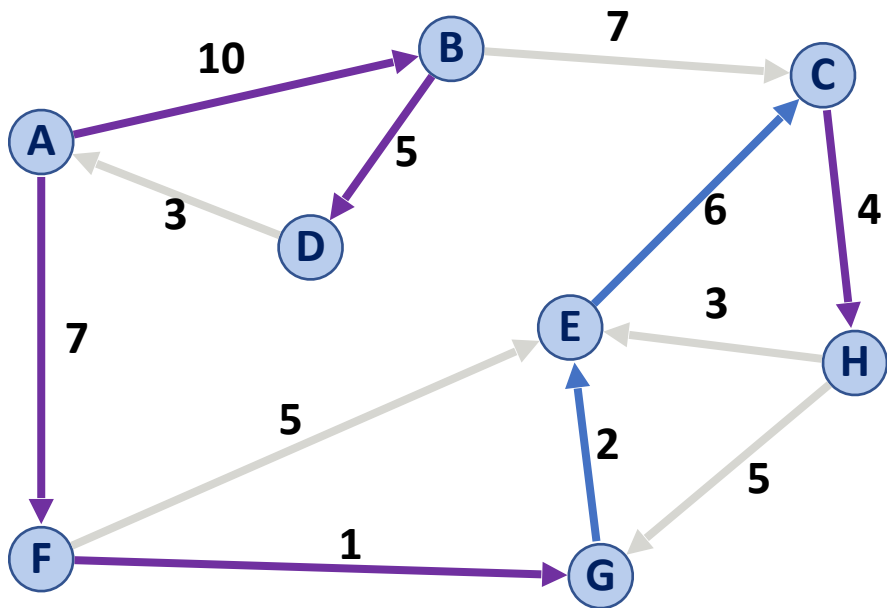
we have dist to u
is it better to
depart through u
to get to v?

A	B	C	D	E	F	G	H
--	A			G	A	F	
0	10	∞	∞	10	7	8	∞

Handwritten annotations on the table: red '10' next to E, red '7' next to F, red '8' next to G, and a red '10' below E.



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	E	B	G	A	F	C
0	10	16	15	10	7	8	20

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
```

Dijkstra's Algorithm (SSSP)

$O(m + n \log n)$

What is the running time of Dijkstra's Algorithm?

The same as Prim's!

(Times here are minheap)

6-9: $O(n)$

11-12: $O(n)$

15: repeat below n x

16-22: $O(\log n)$

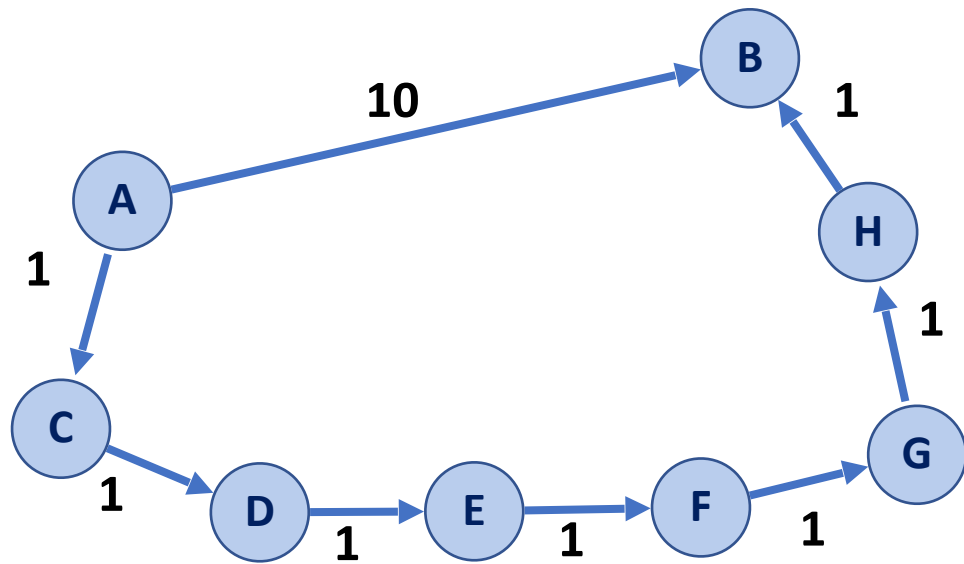
[w/ Fib Heap $O(1)$ updates]

```
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] = u
22
23  return T
```

Dijkstra's Algorithm (SSSP)

Claim: Dijkstras will always visit a node through its optimal shortest path.

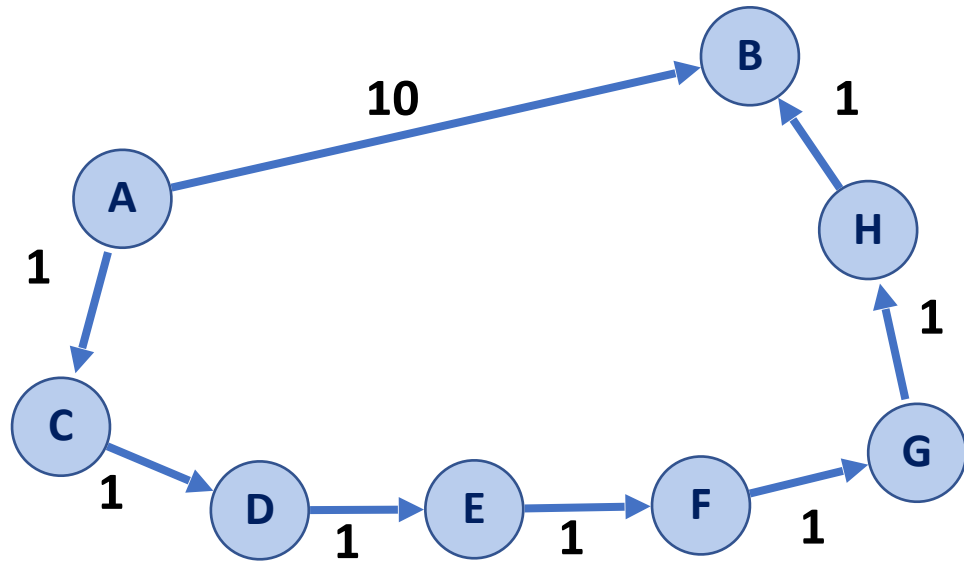
When we will visit B in the following graph?



Dijkstra's Algorithm (SSSP)

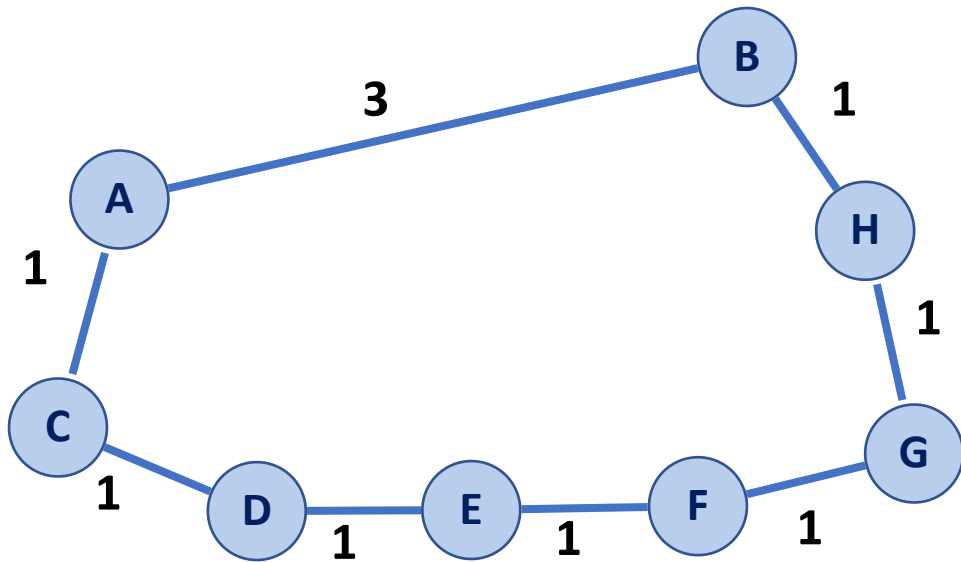
Claim: Dijkstras will always visit a node through its optimal shortest path.

When we will visit H in the following graph?



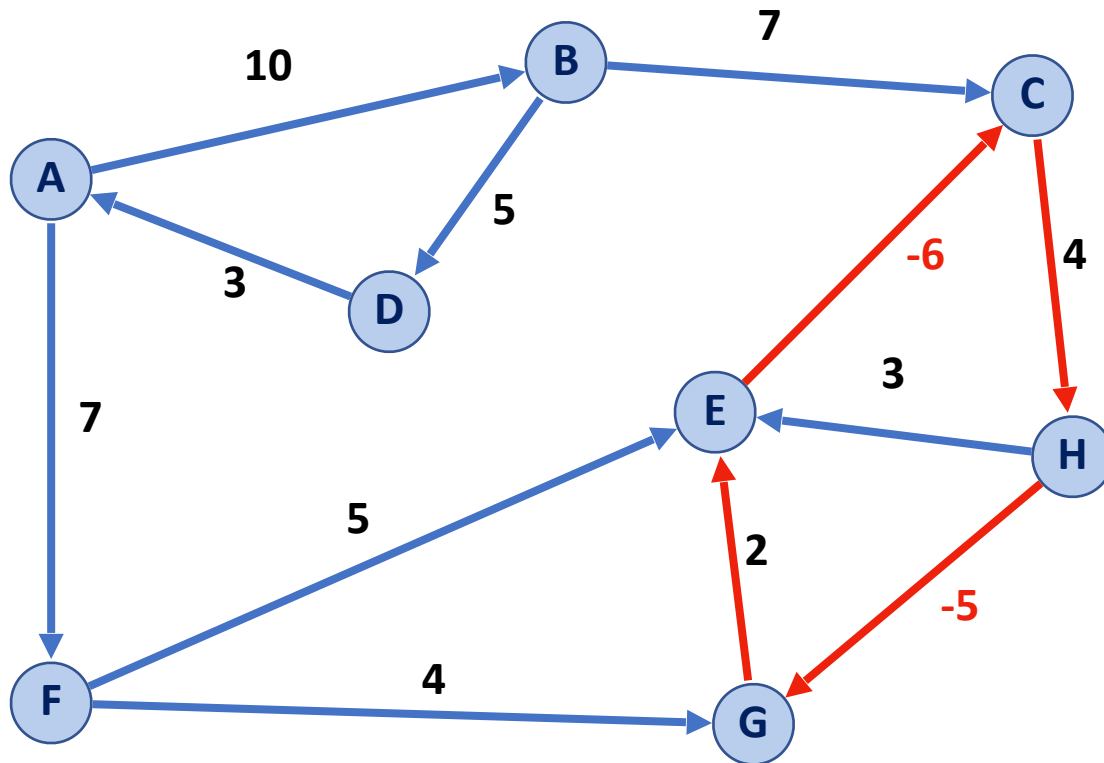
Dijkstra's Algorithm (SSSP)

How does Dijkstra's algorithm handle undirected graphs?



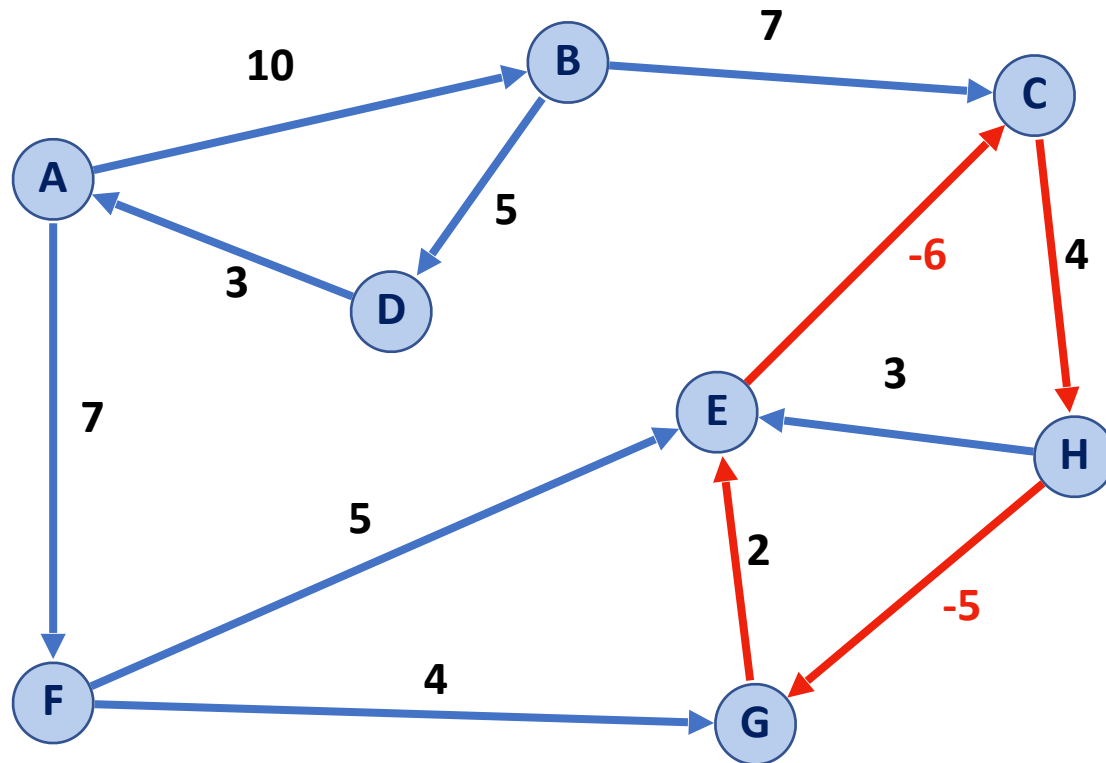
Dijkstra's Algorithm (SSSP)

How does Dijkstra's handle a negative weight cycle?



Dijkstra's Algorithm (SSSP)

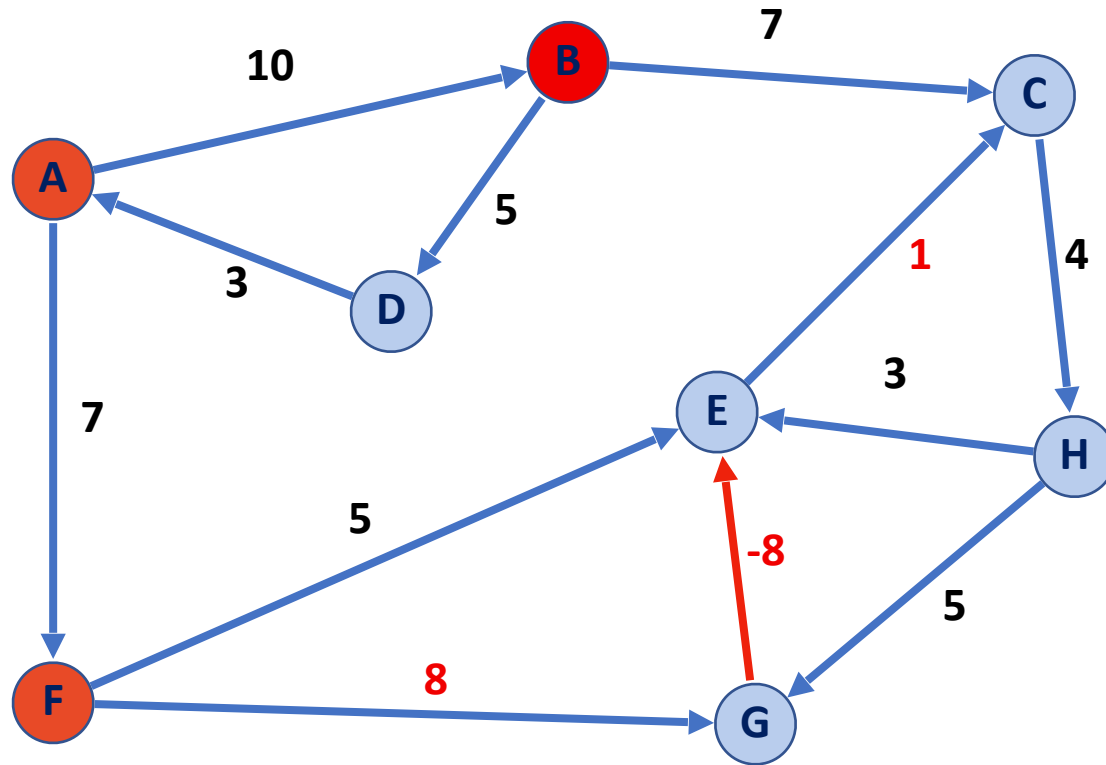
How does Dijkstra's handle a negative weight cycle?



Shortest Path (A → E): A → F → E → (C → H → G → E)*
Length: 12 Length: -5 (repeatable)

Dijkstra's Algorithm (SSSP)

How does Dijkstra's handle a negative weight edge without a cycle?



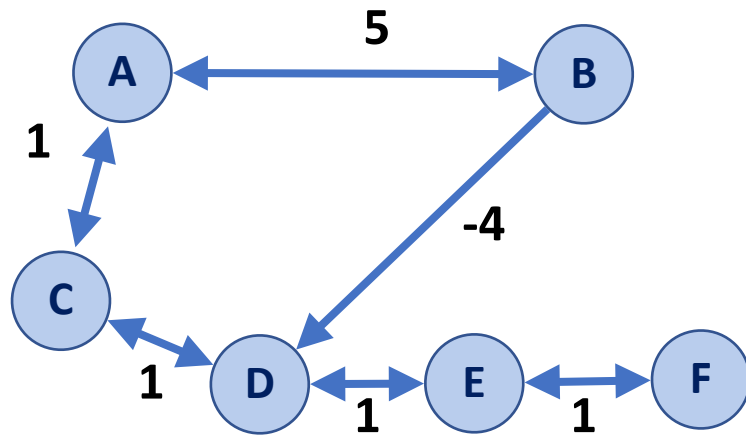
A	B	C	D	E	F	G	H
--	A	B	B	F	A	F	--
0	10	17	15	12	7	15	∞

Dijkstra's Algorithm (SSSP)

We assume that item pulled out of priority queue is **the next smallest item**

Negative weights break this assumption!

A	B	C	D	E	F
--					
0					



Dijkstra's Algorithm (SSSP)

Recalculating all distances is possible, but algorithm runtime is very bad!

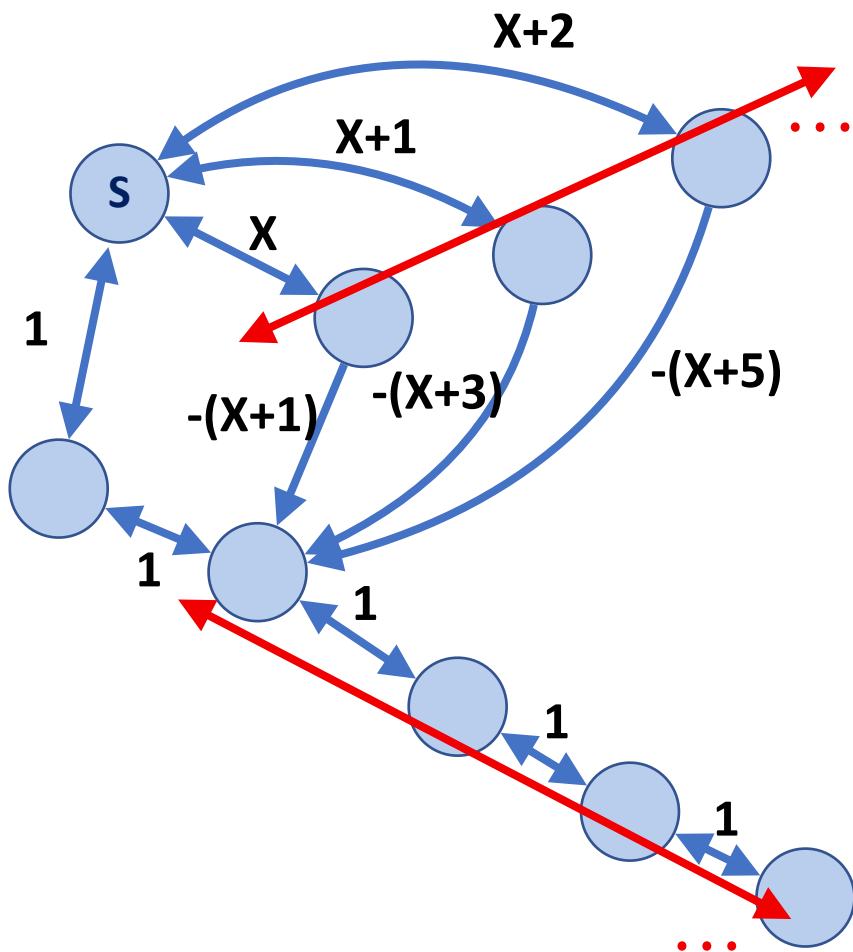
```

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 until Q.empty():
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
22              if v not in Q:
23                  Q.push(v)
24  return T
```

Dijkstra's Algorithm (SSSP)

Recalculating all distances is possible, but algorithm runtime is very bad!

Worst case: $\sim n/2$ nodes each updating $\sim n/2$ nodes distances



```
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 until Q.empty():
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        if v not in Q:
23          Q.push(v)
24  return T
```



Dijkstra's Algorithm (SSSP)

Dijkstra's Algorithm works only on non-negative weights

Optimal implementation:

Fibonacci Heap

If dense, unsorted list ties

Optimal runtime:

Sparse: $O(m + n \log n)$

Dense: $O(n^2)$

```
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] = u
22
23  return T
```

Landmark Path Problem

What if I wanted to get the shortest path from A to G but stopping at L along the way?

