GRAPHS: SHORTEST PATHS

David Kauchak
CS 62 – Fall 2026

1

Admin

Project

- Plan to spend 5-10 hours this week
- Proposal due tomorrow (11/20) at noon
- Status report 1 due next Wednesday (11/25) at noon

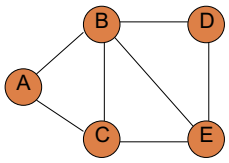
Next week:

- No class or lab
- No mentor hours or office hours
  - We will have regular mentor hours through Sunday (11/21)
- Reach out via slack/email if you have questions

2

Shortest paths

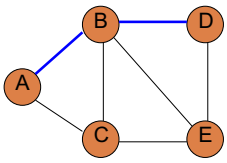
What is the shortest path from a to d?



3

Shortest paths

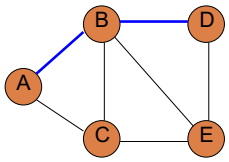
How can we find this?



4

Shortest paths

BFS visits vertices in increasing distance!



5

BFS with distances

Look at ShortestPaths.bfsDistances in GraphExamples

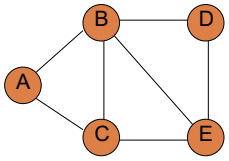
https://github.com/pomonacs622021sp/LectureCode/tree/master/GraphExamples

6

BFS for shortest paths

```
BFS( start )
   q.add(start)
   dist[start] = 0
   q.add(start)

   while !q.empty()
      v = q.remove()
      for adj in v.getAdjacent()
         visited[v] = true
         dist[adj] = dist[v] + 1
         edgeTo[adj] = v
         toVisit.add(adj)
```

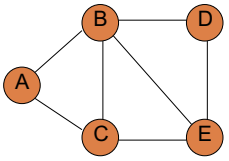


7

BFS for shortest paths

```
BFS( start )
   q.add(start)
   dist[start] = 0
   q.add(start)

   while !q.empty()
      v = q.remove()
      for adj in v.getAdjacent()
         visited[v] = true
         dist[adj] = dist[v] + 1
         edgeTo[adj] = v
         toVisit.add(adj)
```



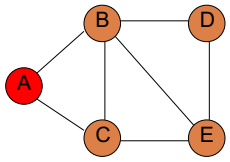
Q: A

8

BFS for shortest paths

```
BFS( start )
   q.add(start)
   dist[start] = 0
   q.add(start)

   while !q.empty()
      v = q.remove()
      for adj in v.getAdjacent()
         visited[v] = true
         dist[adj] = dist[v] + 1
         edgeTo[adj] = v
         toVisit.add(adj)
```



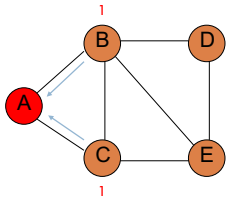
Q:

9

BFS for shortest paths

```
BFS( start )
   q.add(start)
   dist[start] = 0
   q.add(start)

   while !q.empty()
      v = q.remove()
      for adj in v.getAdjacent()
         visited[v] = true
         dist[adj] = dist[v] + 1
         edgeTo[adj] = v
         toVisit.add(adj)
```



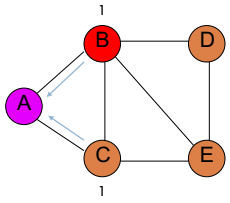
Q: B C

10

BFS for shortest paths

```
BFS( start )
   q.add(start)
   dist[start] = 0
   q.add(start)

   while !q.empty()
      v = q.remove()
      for adj in v.getAdjacent()
         visited[v] = true
         dist[adj] = dist[v] + 1
         edgeTo[adj] = v
         toVisit.add(adj)
```



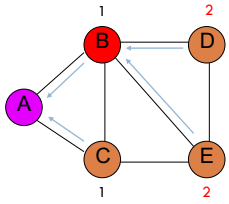
Q: C

11

BFS for shortest paths

```
BFS( start )
   q.add(start)
   dist[start] = 0
   q.add(start)

   while !q.empty()
      v = q.remove()
      for adj in v.getAdjacent()
         visited[v] = true
         dist[adj] = dist[v] + 1
         edgeTo[adj] = v
         toVisit.add(adj)
```



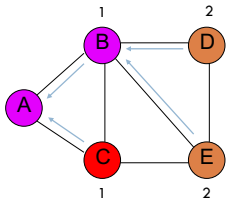
Q: C D E

12

BFS for shortest paths

```
BFS( start )
   q.add(start)
   dist[start] = 0
   q.add(start)

   while !q.empty()
      v = q.remove()
      for adj in v.getAdjacent()
         visited[v] = true
         dist[adj] = dist[v] + 1
         edgeTo[adj] = v
         toVisit.add(adj)
```



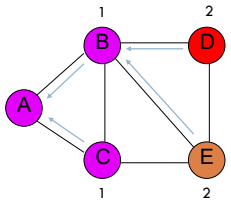
Q: D E

13

BFS for shortest paths

```
BFS( start )
   q.add(start)
   dist[start] = 0
   q.add(start)

   while !q.empty()
      v = q.remove()
      for adj in v.getAdjacent()
         visited[v] = true
         dist[adj] = dist[v] + 1
         edgeTo[adj] = v
         toVisit.add(adj)
```



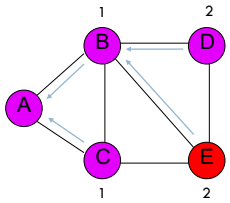
Q: E

14

BFS for shortest paths

```
BFS( start )
   q.add(start)
   dist[start] = 0
   q.add(start)

   while !q.empty()
      v = q.remove()
      for adj in v.getAdjacent()
         visited[v] = true
         dist[adj] = dist[v] + 1
         edgeTo[adj] = v
         toVisit.add(adj)
```



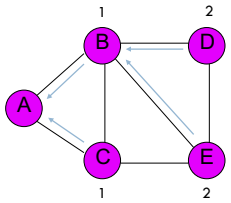
Q:

15

BFS for shortest paths

```
BFS( start )
   q.add(start)
   dist[start] = 0
   q.add(start)

   while !q.empty()
      v = q.remove()
      for adj in v.getAdjacent()
         visited[v] = true
         dist[adj] = dist[v] + 1
         edgeTo[adj] = v
         toVisit.add(adj)
```



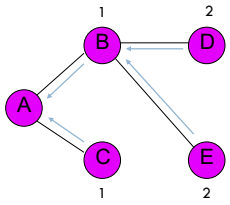
Q:

16

BFS for shortest paths

```
BFS( start )
   q.add(start)
   dist[start] = 0
   q.add(start)

   while !q.empty()
      v = q.remove()
      for adj in v.getAdjacent()
         visited[v] = true
         dist[adj] = dist[v] + 1
         edgeTo[adj] = v
         toVisit.add(adj)
```



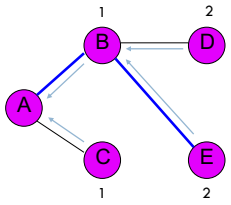
Q: What is the shortest path from A to E?

17

BFS for shortest paths

```
BFS( start )
   q.add(start)
   dist[start] = 0
   q.add(start)

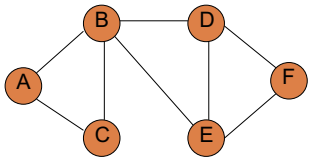
   while !q.empty()
      v = q.remove()
      for adj in v.getAdjacent()
         visited[v] = true
         dist[adj] = dist[v] + 1
         edgeTo[adj] = v
         toVisit.add(adj)
```



Q: What is the shortest path from A to E?

18

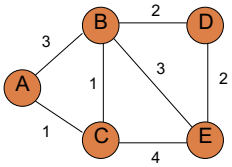
Worksheet



19

Shortest paths

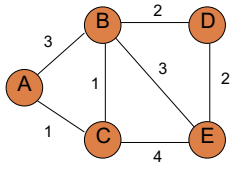
What is the shortest path from a to d?



20

Shortest paths

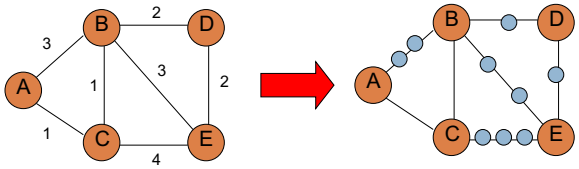
We can still use BFS



21

Shortest paths

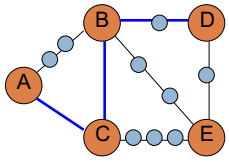
We can still use BFS



22

Shortest paths

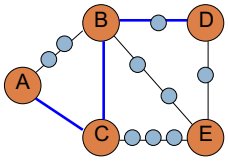
We can still use BFS



23

Shortest paths

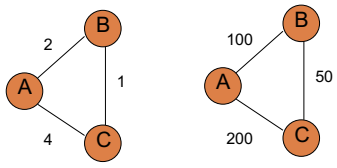
What is the problem?



24

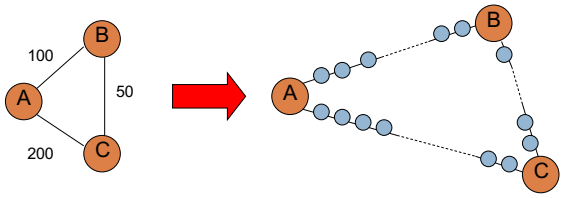
Shortest paths

Running time is dependent on the weights!



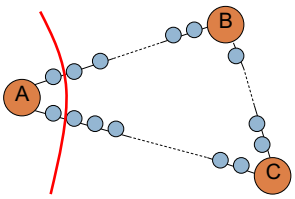
25

Shortest paths



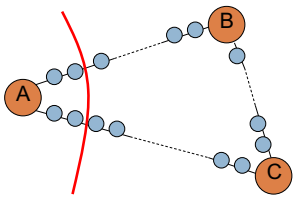
26

Shortest paths



27

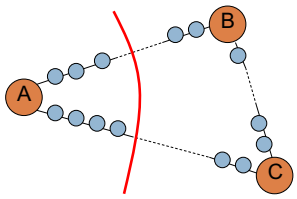
Shortest paths



28

Shortest paths

Nothing will change as we expand the frontier until we've gone out 100 levels



29

Key idea

Explore the vertices in order of increasing distance from the starting vertex

Keep track of the distances to each vertex

If we find a better path, update that distance

30

Dijkstra's high-level

Explore the vertices in order of increasing distance from the starting vertex

Use a priority queue to keep track of the shortest path found so far to a vertex

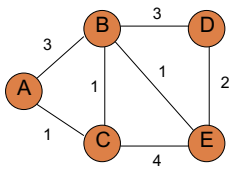
```
dijkstras( start )
    dist[start] = 0, all other v: dist[v] = ∞

    while !pq.empty()
        v = pq.delMin()
        for adj in v.getAdjacent()
            if dist[v] + adj.edgeWeight < dist[adj]
                dist[adj] = dist[v] + adj.edgeWeight
                edgeTo[adj] = v
                pq.decreaseKey(adj, dist[adj])
```

31

```
dijkstras( start )
    dist[start] = 0, all other v: dist[v] = ∞

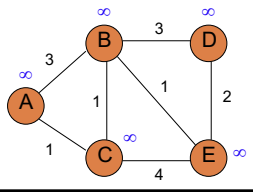
    while !pq.empty()
        v = pq.delMin()
        for adj in v.getAdjacent()
            if dist[v] + adj.edgeWeight < dist[adj]
                dist[adj] = dist[v] + adj.edgeWeight
                edgeTo[adj] = v
                pq.decreaseKey(adj, dist[adj])
```



32

```
dijkstras( start )
   dist[start] = 0, all other v: dist[v] = ∞

   while !pq.empty()
      v = pq.delMin()
      for adj in v.getAdjacent()
            if dist[v] + adj.edgeWeight < dist[adj]
               dist[adj] = dist[v] + adj.edgeWeight
               edgeTo[adj] = v
               pq.decreaseKey(adj, dist[adj])
```



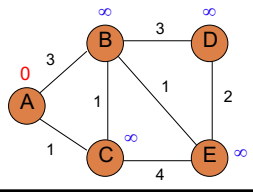
33

```
dijkstras( start )
   dist[start] = 0, all other v: dist[v] = ∞

   while !pq.empty()
      v = pq.delMin()
      for adj in v.getAdjacent()
            if dist[v] + adj.edgeWeight < dist[adj]
               dist[adj] = dist[v] + adj.edgeWeight
               edgeTo[adj] = v
               pq.decreaseKey(adj, dist[adj])
```

PQ

A	0
B	∞
C	∞
D	∞
E	∞



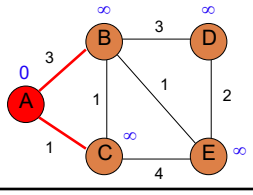
34

```
dijkstras( start )
   dist[start] = 0, all other v: dist[v] = ∞

   while !pq.empty()
      v = pq.delMin()
      for adj in v.getAdjacent()
            if dist[v] + adj.edgeWeight < dist[adj]
               dist[adj] = dist[v] + adj.edgeWeight
               edgeTo[adj] = v
               pq.decreaseKey(adj, dist[adj])
```

PQ

B	∞
C	∞
D	∞
E	∞



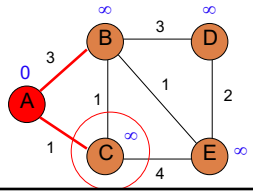
35

```
dijkstras( start )
   dist[start] = 0, all other v: dist[v] = ∞

   while !pq.empty()
      v = pq.delMin()
      for adj in v.getAdjacent()
            if dist[v] + adj.edgeWeight < dist[adj]
               dist[adj] = dist[v] + adj.edgeWeight
               edgeTo[adj] = v
               pq.decreaseKey(adj, dist[adj])
```

PQ

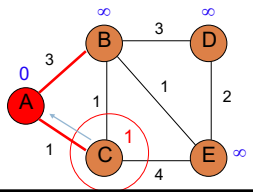
B	∞
C	∞
D	∞
E	∞



36

```
dijkstras( start )
   dist[start] = 0, all other v: dist[v] = ∞

   while !pq.empty()
      v = pq.delMin()
      for adj in v.getAdjacent()
            if dist[v] + adj.edgeWeight < dist[adj]
               dist[adj] = dist[v] + adj.edgeWeight
               edgeTo[adj] = v
               pq.decreaseKey(adj, dist[adj])
```

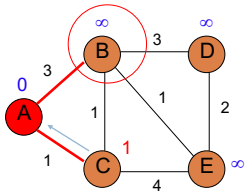


PQ
C 1
B ∞
D ∞
E ∞

37

```
dijkstras( start )
   dist[start] = 0, all other v: dist[v] = ∞

   while !pq.empty()
      v = pq.delMin()
      for adj in v.getAdjacent()
            if dist[v] + adj.edgeWeight < dist[adj]
               dist[adj] = dist[v] + adj.edgeWeight
               edgeTo[adj] = v
               pq.decreaseKey(adj, dist[adj])
```

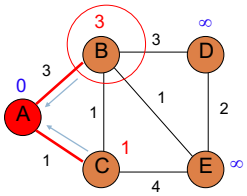


PQ
C 1
B ∞
D ∞
E ∞

38

```
dijkstras( start )
   dist[start] = 0, all other v: dist[v] = ∞

   while !pq.empty()
      v = pq.delMin()
      for adj in v.getAdjacent()
            if dist[v] + adj.edgeWeight < dist[adj]
               dist[adj] = dist[v] + adj.edgeWeight
               edgeTo[adj] = v
               pq.decreaseKey(adj, dist[adj])
```

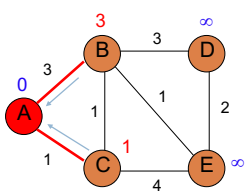


PQ
C 1
B 3
D ∞
E ∞

39

```
dijkstras( start )
   dist[start] = 0, all other v: dist[v] = ∞

   while !pq.empty()
      v = pq.delMin()
      for adj in v.getAdjacent()
            if dist[v] + adj.edgeWeight < dist[adj]
               dist[adj] = dist[v] + adj.edgeWeight
               edgeTo[adj] = v
               pq.decreaseKey(adj, dist[adj])
```



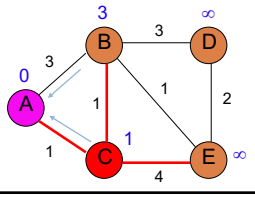
PQ
C 1
B 3
D ∞
E ∞

40

```
dijkstras( start )
   dist[start] = 0, all other v: dist[v] = ∞

   while !pq.empty()
      v = pq.delMin()
      for adj in v.getAdjacent()
            if dist[v] + adj.edgeWeight < dist[adj]
               dist[adj] = dist[v] + adj.edgeWeight
               edgeTo[adj] = v
               pq.decreaseKey(adj, dist[adj])
```

PQ
B 3
D ∞
E ∞

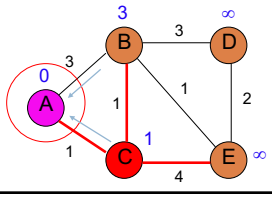


41

```
dijkstras( start )
   dist[start] = 0, all other v: dist[v] = ∞

   while !pq.empty()
      v = pq.delMin()
      for adj in v.getAdjacent()
            if dist[v] + adj.edgeWeight < dist[adj]
               dist[adj] = dist[v] + adj.edgeWeight
               edgeTo[adj] = v
               pq.decreaseKey(adj, dist[adj])
```

PQ
B 3
D ∞
E ∞

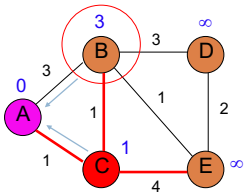


42

```
dijkstras( start )
   dist[start] = 0, all other v: dist[v] = ∞

   while !pq.empty()
      v = pq.delMin()
      for adj in v.getAdjacent()
            if dist[v] + adj.edgeWeight < dist[adj]
               dist[adj] = dist[v] + adj.edgeWeight
               edgeTo[adj] = v
               pq.decreaseKey(adj, dist[adj])
```

PQ
B 3
D ∞
E ∞

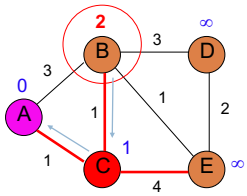


43

```
dijkstras( start )
   dist[start] = 0, all other v: dist[v] = ∞

   while !pq.empty()
      v = pq.delMin()
      for adj in v.getAdjacent()
            if dist[v] + adj.edgeWeight < dist[adj]
               dist[adj] = dist[v] + adj.edgeWeight
               edgeTo[adj] = v
               pq.decreaseKey(adj, dist[adj])
```

PQ
B 2
D ∞
E ∞



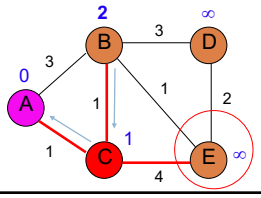
44

```
dijkstras( start )
   dist[start] = 0, all other v: dist[v] = ∞

   while !pq.empty()
      v = pq.delMin()
      for adj in v.getAdjacent()
            if dist[v] + adj.edgeWeight < dist[adj]
               dist[adj] = dist[v] + adj.edgeWeight
               edgeTo[adj] = v
               pq.decreaseKey(adj, dist[adj])
```

PQ

B 2
D ∞
E ∞



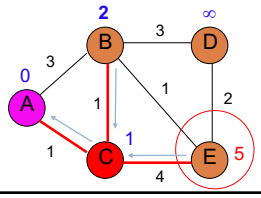
45

```
dijkstras( start )
   dist[start] = 0, all other v: dist[v] = ∞

   while !pq.empty()
      v = pq.delMin()
      for adj in v.getAdjacent()
            if dist[v] + adj.edgeWeight < dist[adj]
               dist[adj] = dist[v] + adj.edgeWeight
               edgeTo[adj] = v
               pq.decreaseKey(adj, dist[adj])
```

PQ

B 2
E 5
D ∞



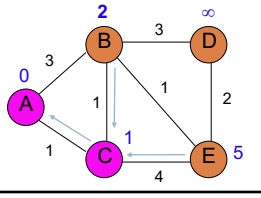
46

```
dijkstras( start )
   dist[start] = 0, all other v: dist[v] = ∞

   while !pq.empty()
      v = pq.delMin()
      for adj in v.getAdjacent()
            if dist[v] + adj.edgeWeight < dist[adj]
               dist[adj] = dist[v] + adj.edgeWeight
               edgeTo[adj] = v
               pq.decreaseKey(adj, dist[adj])
```

PQ

B 2
E 5
D ∞



Frontier?

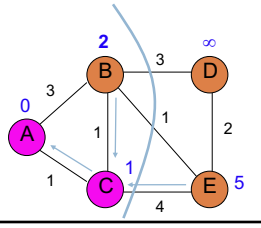
47

```
dijkstras( start )
   dist[start] = 0, all other v: dist[v] = ∞

   while !pq.empty()
      v = pq.delMin()
      for adj in v.getAdjacent()
            if dist[v] + adj.edgeWeight < dist[adj]
               dist[adj] = dist[v] + adj.edgeWeight
               edgeTo[adj] = v
               pq.decreaseKey(adj, dist[adj])
```

PQ

B 2
E 5
D ∞



All nodes reachable from starting node within a given distance

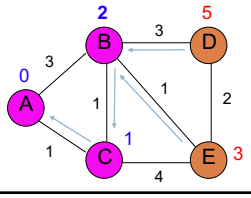
48

```
dijkstras( start )
   dist[start] = 0, all other v: dist[v] = ∞

   while !pq.empty()
      v = pq.delMin()
      for adj in v.getAdjacent()
            if dist[v] + adj.edgeWeight < dist[adj]
               dist[adj] = dist[v] + adj.edgeWeight
               edgeTo[adj] = v
               pq.decreaseKey(adj, dist[adj])
```

PQ

E 3
D 5



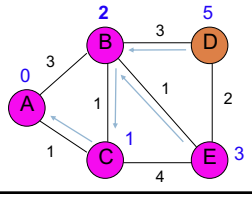
49

```
dijkstras( start )
   dist[start] = 0, all other v: dist[v] = ∞

   while !pq.empty()
      v = pq.delMin()
      for adj in v.getAdjacent()
            if dist[v] + adj.edgeWeight < dist[adj]
               dist[adj] = dist[v] + adj.edgeWeight
               edgeTo[adj] = v
               pq.decreaseKey(adj, dist[adj])
```

PQ

D 5

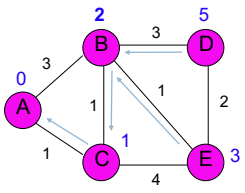


50

```
dijkstras( start )
   dist[start] = 0, all other v: dist[v] = ∞

   while !pq.empty()
      v = pq.delMin()
      for adj in v.getAdjacent()
            if dist[v] + adj.edgeWeight < dist[adj]
               dist[adj] = dist[v] + adj.edgeWeight
               edgeTo[adj] = v
               pq.decreaseKey(adj, dist[adj])
```

PQ

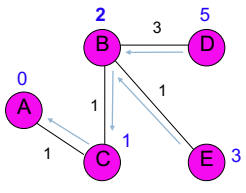


51

```
dijkstras( start )
   dist[start] = 0, all other v: dist[v] = ∞

   while !pq.empty()
      v = pq.delMin()
      for adj in v.getAdjacent()
            if dist[v] + adj.edgeWeight < dist[adj]
               dist[adj] = dist[v] + adj.edgeWeight
               edgeTo[adj] = v
               pq.decreaseKey(adj, dist[adj])
```

PQ

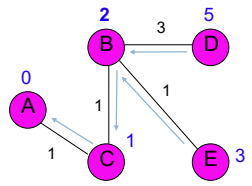


52

```
dijkstras( start )
   dist[start] = 0, all other v: dist[v] = ∞

   while !pq.empty()
      v = pq.delMin()
      for adj in v.getAdjacent()
            if dist[v] + adj.edgeWeight < dist[adj]
               dist[adj] = dist[v] + adj.edgeWeight
               edgeTo[adj] = v
               pq.decreaseKey(adj, dist[adj])
```

PQ



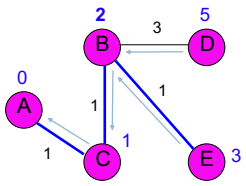
What is the shortest path from A to E?

53

```
dijkstras( start )
   dist[start] = 0, all other v: dist[v] = ∞

   while !pq.empty()
      v = pq.delMin()
      for adj in v.getAdjacent()
            if dist[v] + adj.edgeWeight < dist[adj]
               dist[adj] = dist[v] + adj.edgeWeight
               edgeTo[adj] = v
               pq.decreaseKey(adj, dist[adj])
```

PQ



What is the shortest path from A to E?

54

Dijkstra's algorithm

```
public static void dijkstra(WeightedGraph g, int start) {
    IndexMinPQ<Double> pq = new IndexMinPQ<Double>(g.numberOfVertices());
    int[] edgeTo = new int[g.numberOfVertices()];
    double[] distTo = new double[g.numberOfVertices()];

    for( int v = 0; v < g.numberOfVertices(); v++ ) {
        distTo[v] = Double.POSITIVE_INFINITY;
        pq.insert(v, Double.POSITIVE_INFINITY);
    }

    distTo[start] = 0.0;
    pq.decreaseKey(start, 0.0);

    while( !pq.isEmpty() ) {
        int v = pq.delMin();

        for (WeightedEdge e : g.adj(v)) {
            int adj = e.to();

            if( distTo[v] + e.weight() < distTo[adj] ) {
                distTo[adj] = distTo[v] + e.weight();
                edgeTo[adj] = v;
                pq.decreaseKey(adj, distTo[adj]);
            }
        }
    }
```

55

Dijkstra's algorithm

Dijkstra's

```
distTo[start] = 0.0;
pq.decreaseKey(start, 0.0);

while( !pq.isEmpty() ) {
    int v = pq.delMin();

    for (WeightedEdge e : g.adj(v)) {
        int adj = e.to();

        if( distTo[v] + e.weight() < distTo[adj] ) {
            distTo[adj] = distTo[v] + e.weight();
            edgeTo[adj] = v;
            pq.decreaseKey(adj, distTo[adj]);
        }
    }
}
```

BFS

```
q.addLast(start);
visited[start] = true;
distTo[start] = 0;

while( !q.isEmpty() ) {
    int v = q.removeFirst();

    for( int adj: g.adj(v) ) {
        if( !visited[adj] ) {
            visited[adj] = true;
            edgeTo[adj] = v;
            distTo[adj] = distTo[v] + 1;
            q.addLast(adj);
        }
    }
}
```

56

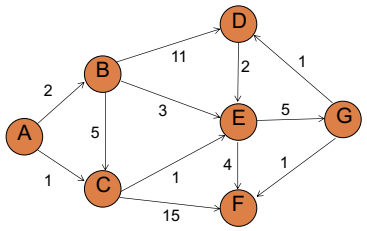
Dijkstra example

Look at ShortestPaths.dijkstra in GraphExamples

https://github.com/pomonacs622021sp/LectureCode/tree/master/GraphExamples

57

Worksheet



58

Why does it work?

When a vertex is removed from the priority queue, distTo[v] is the actual shortest distance from s to v

- The only time a vertex gets removed is when the distance from s to that vertex is smaller than the distance to any remaining vertex
- Therefore, there cannot be any other path that hasn't been visited already that would result in a shorter path

59

Why does it work?

When a vertex is removed from the priority queue, distTo[v] is the actual shortest distance from s to v

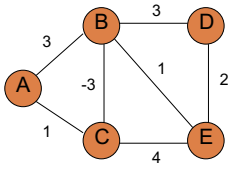
- The only time a vertex gets removed is when the distance from s to that vertex is smaller than the distance to any remaining vertex
- Therefore, there cannot be any other path that hasn't been visited already that would result in a shorter path

Does this make any assumptions?

60

What about this graph?

What's the shortest path from A to C?
What would Dijkstra's do?




```
distTo[start] = 0.0;
pq.decreaseKey(start, 0.0);

while( !pq.isEmpty() ) {
    int v = pq.delMin();

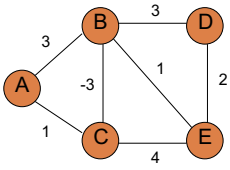
    for (WeightedEdge e : g.adj(v)) {
        int adj = e.to();

        if( distTo[v] + e.weight() < distTo[adj] ) {
            distTo[adj] = distTo[v] + e.weight();
            edgeTo[adj] = v;
            pq.decreaseKey(adj, distTo[adj]);
        }
    }
}
```

61

What about this graph?

Dijkstra's only works on graphs with positive edge weights




```
distTo[start] = 0.0;
pq.decreaseKey(start, 0.0);

while( !pq.isEmpty() ) {
    int v = pq.delMin();

    for (WeightedEdge e : g.adj(v)) {
        int adj = e.to();

        if( distTo[v] + e.weight() < distTo[adj] ) {
            distTo[adj] = distTo[v] + e.weight();
            edgeTo[adj] = v;
            pq.decreaseKey(adj, distTo[adj]);
        }
    }
}
```

C wouldn't be in the pq

62

Why does it work?

When a vertex is removed from the priority queue, distTo[v] is the actual shortest distance from s to v

- The only time a vertex gets removed is when the distance from s to that vertex is smaller than the distance to any remaining vertex
- Therefore, there cannot be any other path that hasn't been visited already that would result in a shorter path

Assuming no negative edge weights!

63

Relaxing an edge

This update is called "relaxing" an edge

```
if( distTo[v] + e.weight() < distTo[adj] ) {
    distTo[adj] = distTo[v] + e.weight();
    edgeTo[adj] = v;
    pq.decreaseKey(adj, distTo[adj]);
}
```

We can apply this to an edge as many times as we want

This idea is used in other shortest paths algorithms (e.g., Bellman-Ford)

64

```
public static void fasterDijkstra(WeightedGraph g, int start) {
    IndexMinPQ<Double> pq = new IndexMinPQ<Double>(g.numberOfVertices());
    int[] edgeTo = new int[g.numberOfVertices()];
    double[] distTo = new double[g.numberOfVertices()];

    for( int v = 0; v < g.numberOfVertices(); v++ ) {
        distTo[v] = Double.POSITIVE_INFINITY;
    }

    distTo[start] = 0.0;
    pq.insert(start, 0.0);

    while( !pq.isEmpty() ) {
        int v = pq.delMin();

        for (WeightedEdge e : g.adj(v)) {
            int adj = e.to();

            if( distTo[v] + e.weight() < distTo[adj] ) {
                distTo[adj] = distTo[v] + e.weight();
                edgeTo[adj] = v;

                if( pq.contains(adj) ) {
                    pq.decreaseKey(adj, distTo[adj]);
                } else {
                    pq.insert(adj, distTo[adj]);
                }
            }
        }
```

don't insert everything into pq

only insert starting vertex

insert when we discover a vertex

65

Run-time

```
public static void bfsDistances(Graph g, int start) {
    Deque<Integer> q = new ArrayDeque<Integer>();
    boolean[] visited = new boolean[g.numberOfVertices()];
    int[] edgeTo = new int[g.numberOfVertices()];
    int[] distTo = new int[g.numberOfVertices()];

    q.addLast(start);
    visited[start] = true;
    distTo[start] = 0;

    while( !q.isEmpty() ) {
        int v = q.removeFirst();

        for( int adj: g.adj(v) ) {
            if( !visited[adj] ) {
                visited[adj] = true;
                edgeTo[adj] = v;
                distTo[adj] = distTo[v] + 1;
                q.addLast(adj);
            }
        }
    }
```

V calls

E calls

66

Running time?

Depends on the heap implementation

	V * delMin	E * decreaseKey	Total
Array			
Bin heap			

Heap as an unordered array, where each index corresponds to a vertex and we store the weight at that index. What is the big-O run-time of delMin and decreaseKey?

67

Running time?

Depends on the heap implementation

	V * delMin	E * decreaseKey	Total
Array	$O(\|V\|^2)$	$O(\|E\|)$	$O(\|V\|^2)$
Bin heap	$O(\|V\| \log \|V\|)$	$O(\|E\| \log \|V\|)$	$O((\|V\|+\|E\|) \log \|V\|)$ $O(\|E\| \log \|V\|)$

68

Running time?

Depends on the heap implementation

	V * delMin	E * decreaseKey	Total
Array	$O(\|V\|^2)$	$O(\|E\|)$	$O(\|V\|^2)$
Bin heap	$O(\|V\| \log \|V\|)$	$O(\|E\| \log \|V\|)$	$O((\|V\|+\|E\|) \log \|V\|)$ $O(\|E\| \log \|V\|)$
Fib heap	$O(\|V\| \log \|V\|)$	$O(\|E\|)$	$O(\|V\| \log \|V\| + \|E\|)$

69

Shortest paths

Dijkstra's: single source shortest paths for positive edge weight graphs

What is single source?

70

Shortest paths

Dijkstra's: single source shortest paths for positive edge weight graphs

Many other variants:

- graphs with negative edges
- all pairs shortest paths
- …

71