GRAPHS: SHORTEST PATHS

David Kauchak
CS 62 – Spring 2021

1

Admin

Last quiz!

Last assignment due next Friday (5/7)

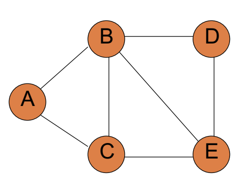
Next week:

- Tuesday: balanced trees
- Wednesday: course feedback forms, ethics discussion, work session
- Thursday: recap/review

2

Shortest paths

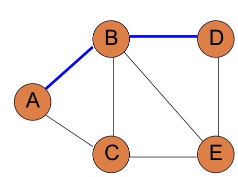
What is the shortest path from a to d?



3

Shortest paths

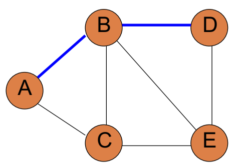
How can we find this?



4

Shortest paths

BFS visits vertices in increasing distance!



5

BFS with distances

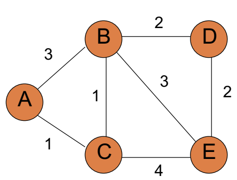
Look at ShortestPaths.bfsDistances in GraphExamples

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

6

Shortest paths

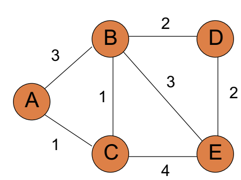
What is the shortest path from a to d?



7

Shortest paths

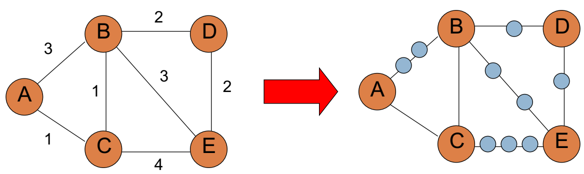
We can still use BFS



8

Shortest paths

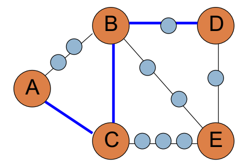
We can still use BFS



9

Shortest paths

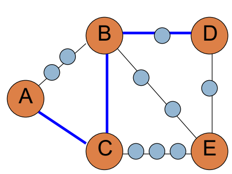
We can still use BFS



10

Shortest paths

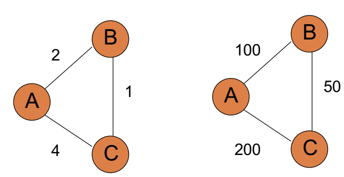
What is the problem?



11

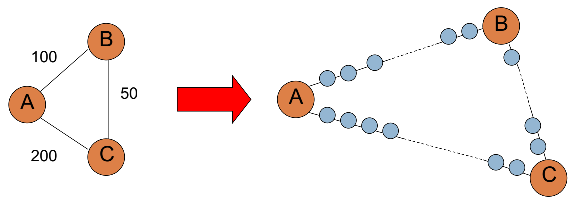
Shortest paths

Running time is dependent on the weights!



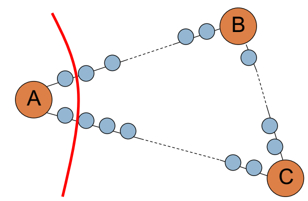
12

Shortest paths



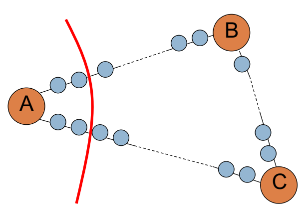
13

Shortest paths



14

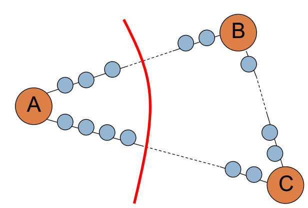
Shortest paths



15

Shortest paths

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



16

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

17

Dijkstra's high-level

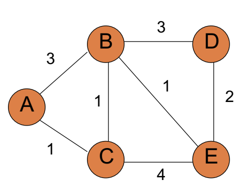
Explore the vertices in order of increasing distance from the starting vertex

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

```
Initialize: distance to start = 0 and all others infinity

repeat
    get vertex v with shortest distance

    for each vertex, adj, adjacent to v (edge exists v → adj)
        if path v → adj is shortest then best path for adj so far
            update the distance for adj
            update the priority queue
```

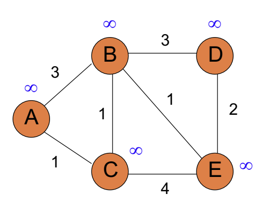
18

```
Initialize: distance to start = 0 and all others infinity

repeat
    get vertex v with shortest distance

    for each vertex, adj, adjacent to v (edge exists v → adj)
        if path v → adj is shortest then best path for adj so far
            update the distance for adj
            update the priority queue
```



19

```
Initialize: distance to start = 0 and all others infinity

repeat
    get vertex v with shortest distance

    for each vertex, adj, adjacent to v (edge exists v → adj)
        if path v → adj is shortest then best path for adj so far
            update the distance for adj
            update the priority queue
```



20

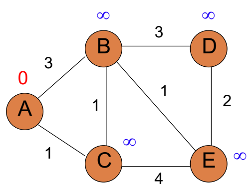
Initialize: distance to start = 0 and all others infinity

repeat
    get vertex v with shortest distance

    for each vertex, adj, adjacent to v (edge exists v ⟶ adj)
        if path v ⟶ adj is shortest then best path for adj so far
            update the distance for adj
            update the priority queue

PQ

A 0
B ∞
C ∞
D ∞
E ∞



21

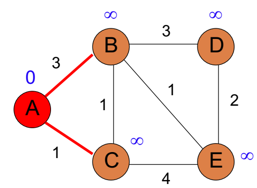
Initialize: distance to start = 0 and all others infinity

repeat
    get vertex v with shortest distance

    for each vertex, adj, adjacent to v (edge exists v ⟶ adj)
        if path v ⟶ adj is shortest then best path for adj so far
            update the distance for adj
            update the priority queue

PQ

B ∞
C ∞
D ∞
E ∞



22

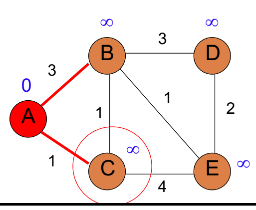
Initialize: distance to start = 0 and all others infinity

repeat
    get vertex v with shortest distance

    for each vertex, adj, adjacent to v (edge exists v ⟶ adj)
        if path v ⟶ adj is shortest then best path for adj so far
            update the distance for adj
            update the priority queue

PQ

B ∞
C ∞
D ∞
E ∞



23

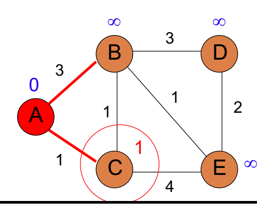
Initialize: distance to start = 0 and all others infinity

repeat
    get vertex v with shortest distance

    for each vertex, adj, adjacent to v (edge exists v ⟶ adj)
        if path v ⟶ adj is shortest then best path for adj so far
            update the distance for adj
            update the priority queue

PQ

C 1
B ∞
D ∞
E ∞



24

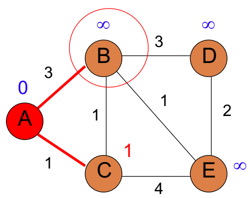
Initialize: distance to start = 0 and all others infinity

repeat
    get vertex v with shortest distance

    for each vertex, adj, adjacent to v (edge exists v ⟶ adj)
        if path v ⟶ adj is shortest then best path for adj so far
            update the distance for adj
            update the priority queue

PQ

C 1
B ∞
D ∞
E ∞



25

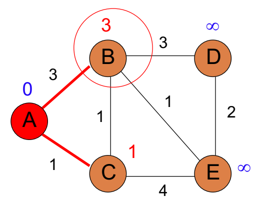
Initialize: distance to start = 0 and all others infinity

repeat
    get vertex v with shortest distance

    for each vertex, adj, adjacent to v (edge exists v ⟶ adj)
        if path v ⟶ adj is shortest then best path for adj so far
            update the distance for adj
            update the priority queue

PQ

C 1
B 3
D ∞
E ∞



26

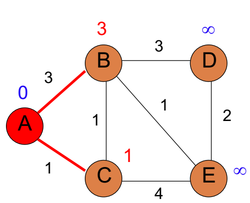
Initialize: distance to start = 0 and all others infinity

repeat
    get vertex v with shortest distance

    for each vertex, adj, adjacent to v (edge exists v ⟶ adj)
        if path v ⟶ adj is shortest then best path for adj so far
            update the distance for adj
            update the priority queue

PQ

C 1
B 3
D ∞
E ∞



27

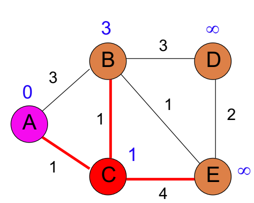
Initialize: distance to start = 0 and all others infinity

repeat
    get vertex v with shortest distance

    for each vertex, adj, adjacent to v (edge exists v ⟶ adj)
        if path v ⟶ adj is shortest then best path for adj so far
            update the distance for adj
            update the priority queue

PQ

B 3
D ∞
E ∞



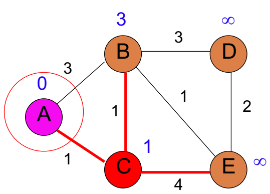
28

Initialize: distance to start = 0 and all others infinity

repeat
- get vertex v with shortest distance
- for each vertex, adj, adjacent to v (edge exists v ⟶ adj)
  - if path v ⟶ adj is shortest then best path for adj so far
    - update the distance for adj
    - update the priority queue

PQ

B 3
D ∞
E ∞



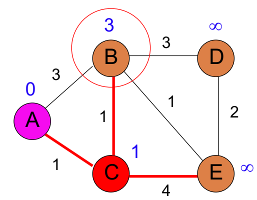
29

Initialize: distance to start = 0 and all others infinity

repeat
- get vertex v with shortest distance
- for each vertex, adj, adjacent to v (edge exists v ⟶ adj)
  - if path v ⟶ adj is shortest then best path for adj so far
    - update the distance for adj
    - update the priority queue

PQ

B 3
D ∞
E ∞



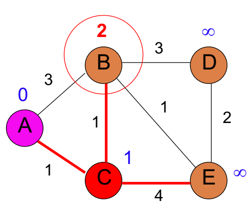
30

Initialize: distance to start = 0 and all others infinity

repeat
- get vertex v with shortest distance
- for each vertex, adj, adjacent to v (edge exists v ⟶ adj)
  - if path v ⟶ adj is shortest then best path for adj so far
    - update the distance for adj
    - update the priority queue

PQ

B 2
D ∞
E ∞



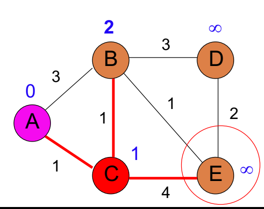
31

Initialize: distance to start = 0 and all others infinity

repeat
- get vertex v with shortest distance
- for each vertex, adj, adjacent to v (edge exists v ⟶ adj)
  - if path v ⟶ adj is shortest then best path for adj so far
    - update the distance for adj
    - update the priority queue

PQ

B 2
D ∞
E ∞



32

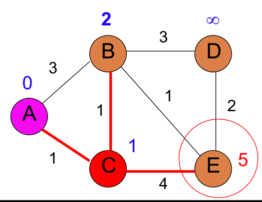
Initialize: distance to start = 0 and all others infinity

repeat
  get vertex v with shortest distance

  for each vertex, adj, adjacent to v (edge exists v ⟶ adj)
    if path v ⟶ adj is shortest then best path for adj so far
      update the distance for adj
      update the priority queue

PQ

B 2
E 5
D ∞



33

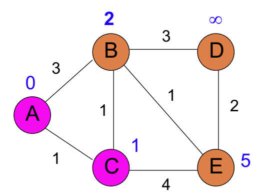
Initialize: distance to start = 0 and all others infinity

repeat
  get vertex v with shortest distance

  for each vertex, adj, adjacent to v (edge exists v ⟶ adj)
    if path v ⟶ adj is shortest then best path for adj so far
      update the distance for adj
      update the priority queue

PQ

B 2
E 5
D ∞



Frontier?

34

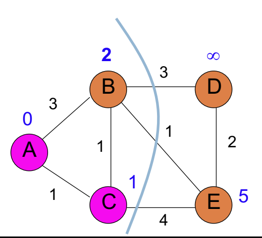
Initialize: distance to start = 0 and all others infinity

repeat
  get vertex v with shortest distance

  for each vertex, adj, adjacent to v (edge exists v ⟶ adj)
    if path v ⟶ adj is shortest then best path for adj so far
      update the distance for adj
      update the priority queue

PQ

B 2
E 5
D ∞



All nodes reachable from starting node within a given distance

35

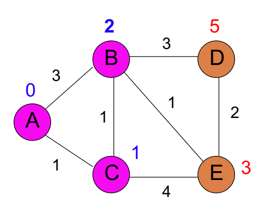
Initialize: distance to start = 0 and all others infinity

repeat
  get vertex v with shortest distance

  for each vertex, adj, adjacent to v (edge exists v ⟶ adj)
    if path v ⟶ adj is shortest then best path for adj so far
      update the distance for adj
      update the priority queue

PQ

E 3
D 5



36

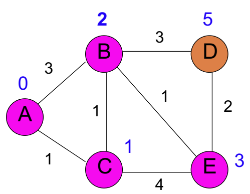
Initialize: distance to start = 0 and all others infinity

repeat
    get vertex v with shortest distance

    for each vertex, adj, adjacent to v (edge exists v ⟶ adj)
        if path v ⟶ adj is shortest then best path for adj so far
            update the distance for adj
            update the priority queue

PQ

D 5



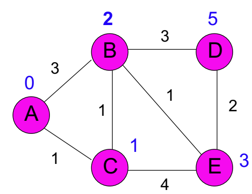
37

Initialize: distance to start = 0 and all others infinity

repeat
    get vertex v with shortest distance

    for each vertex, adj, adjacent to v (edge exists v ⟶ adj)
        if path v ⟶ adj is shortest then best path for adj so far
            update the distance for adj
            update the priority queue

PQ



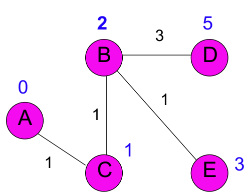
38

Initialize: distance to start = 0 and all others infinity

repeat
    get vertex v with shortest distance

    for each vertex, adj, adjacent to v (edge exists v ⟶ adj)
        if path v ⟶ adj is shortest then best path for adj so far
            update the distance for adj
            update the priority queue

PQ



39

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);

    // relax vertices in order of distance from s
    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]);
            }
        }
    }
}
```

40

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);
        }
    }
}
```

41

Dijkstra example

Look at ShortestPaths.dijkstra in GraphExamples

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

42

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

43

Why does it work?

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

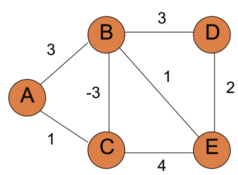
- The only time a vertex gets removed is when the distance from s to that vertex is smaller than the distance to any remaining vertex
- Therefore, there cannot be any other path that hasn't been visited already that would result in a shorter path

Does this make any assumptions?

44

What about this graph?

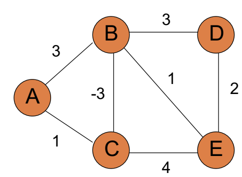
What's the shortest path from A to C?
What would Dijkstra's do?



45

What about this graph?

Dijkstra's only works on graphs with positive edge weights



46

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!

47

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)

48

```java
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;          don't insert everything into pq
    }

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

    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]);       insert when we discover a vertex
                }
            }
        }
    }
}
```

49

Run-time

```java
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);

    // relax vertices in order of distance from s
    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]);
            }
        }
    }
}
```

V calls

E calls

50

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|)

51

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|)

52

Shortest paths

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

What is single source?

53

Shortest paths

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

Many other variants:

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

54