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

Shortest paths

What is the shortest path from a to d?

How can we find this?
Shortest paths

BFS visits vertices in increasing distance!

BFS with distances
Look at ShortestPaths.bfsDistances in GraphExamples

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

Shortest paths
What is the shortest path from a to d?

Shortest paths
We can still use BFS
Shortest paths

We can still use BFS

A

B

C

D

E

1

3

2

1

4

9

Shortest paths

We can still use BFS

A

B

C

D

E

10

Shortest paths

What is the problem?

A

B

C

D

E

11

Running time is dependent on the weights!

A

B

C

D

E

12
Shortest paths

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

---

**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, \( u \), adjacent to \( v \) (edge exists \( u \rightarrow v \))

if path \( u \rightarrow v \) is shortest then best path for \( v \) so far

update the distance for \( v \)

update the priority queue

---

**Sequence of steps**

1. Initialize: distance to start = 0 and all others infinity.
2. Repeat:
   - Get vertex \( v \) with shortest distance.
   - For each vertex, \( u \), adjacent to \( v \):
     - If path \( u \rightarrow v \) is shortest then best path for \( v \) so far:
       - Update the distance for \( v \).
       - Update the priority queue.

---

**Graph representation**

- Initial graph:
  - A to B: 3
  - A to C: 1
  - A to D: 4
  - B to C: 1
  - B to D: 1
  - C to E: 1
  - D to E: 2

- Updated graph:
  - A to B: 3
  - A to C: 1
  - A to D: 4
  - B to C: 1
  - B to D: 1
  - C to E: 1
  - D to E: 2
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
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
Initialize: distance to start = 0 and all others infinity

repeat
get vertex \( v \) with shortest distance

for each vertex, \( \text{adj} \), adjacent to \( v \) (edge exists \( v \rightarrow \text{adj} \))
if path \( v \rightarrow \text{adj} \) is shortest then best path for \( \text{adj} \) so far
update the distance for \( \text{adj} \)
update the priority queue
Initialize: distance to start = 0 and all others infinity
repeat
  get vertex \( v \) with shortest distance
  for each vertex, \( \text{adj} \), adjacent to \( v \) (edge exists \( v \rightarrow \text{adj} \))
    if path \( v \rightarrow \text{adj} \) is shortest then best path for \( \text{adj} \) so far
    update the distance for \( \text{adj} \)
    update the priority queue

Frontier?

All nodes reachable from starting node within a given distance
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 \( v \rightarrow adj \))
    if path \( v \rightarrow adj \) is shortest then best path for \( adj \) so far
      update the distance for \( adj \)
      update the priority queue

Dijkstra's algorithm

```java
public static void dijkstraWeightedGraph(G, let start) {
  int[] distances = new int[numberVertices];
  boolean[] isVisited = new boolean[numberVertices];
  PriorityQueue<int[]> PQ = new PriorityQueue<int[]>();

  PQ.offer(start);
  distances[start] = 0;
  for (int i = 0; i < numberVertices; ++i) {
    if (i != start && distances[i] == Integer.MAX_VALUE) {
      distances[i] = distances[start] + weight(start, i);
      PQ.offer(new int[]{i, distances[i]});
    }
  }

  while (!PQ.isEmpty()) {
    int[] edge = PQ.poll();
    int adj = edge[0];
    int dist = edge[1];
    for (int i = 0; i < numberVertices; ++i) {
      if (!isVisited[i] && dist + weight(adj, i) < distances[i]) {
        distances[i] = dist + weight(adj, i);
        PQ.offer(new int[]{i, distances[i]});
      }
    }
  }
}
```
Dijkstra's algorithm

Dijkstra's algorithm

\[ \text{distTo}(s) = 0; \]
\[ \text{visited}(s) = \text{true}; \]
\[ \text{distTo}(v) = \infty; \]
\[ \text{visited}(v) = \text{false}; \]
while (visited\((s)\) is false)

\[ \text{for all } (u, v) \in \text{adj}(s) \]
\[ \text{if } \text{distTo}(u) + \text{weight}(u, v) < \text{distTo}(v) \]
\[ \text{distTo}(v) = \text{distTo}(u) + \text{weight}(u, v); \]
\[ \text{visited}(v) = \text{true}; \]
\[ \text{decreaseKey}(v, \text{distTo}(v)); \]

BFS

\[ \text{q.addLast}(s); \]
\[ \text{visited}(s) = \text{true}; \]
\[ \text{distTo}(s) = 0; \]
while (q.isEmpty() is false)

\[ \text{for all } (u, v) \in \text{adj}(u) \]
\[ \text{if } \text{distTo}(u) + \text{weight}(u, v) < \text{distTo}(v) \]
\[ \text{distTo}(v) = \text{distTo}(u) + \text{weight}(u, v); \]
\[ \text{visited}(v) = \text{true}; \]
\[ \text{decreaseKey}(v, \text{distTo}(v)); \]

Why does it work?

When a vertex is removed from the priority queue, \(\text{distTo}[v]\)

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

Dijkstra example

Look at ShortestPaths.dijkstra in GraphExamples

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

Why does it work?

When a vertex is removed from the priority queue, \(\text{distTo}[v]\)

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?
What about this graph?

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

What about this graph?

Dijkstra's only works on graphs with positive edge weights

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!

Relaxing an edge

This update is called “relaxing” an edge

```java
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 path algorithms (e.g., Bellman-Ford)
Dijkstra in practice

don’t insert everything into pq
only insert starting vertex
insert when we discover a vertex

while (pq is not empty) {
    int v = pq.delMin();
    for (int adj : v.adj()) {
        if (distTo[v] + v.weight() < distTo[adj]) {
            distTo[adj] = distTo[v] + v.weight();
            edgeTo[adj] = v;
            if (pq.contains(adj)) {  // decreaseKey
                pq.decreaseKey(adj, distTo[adj]);
            } else {
                pq.insert(adj, distTo[adj]);  // insert when we discover a vertex
            }
        }
    }
}

Running time?

<table>
<thead>
<tr>
<th>Heap</th>
<th>V * delMin</th>
<th>E * decreaseKey</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>Array</td>
<td>O(</td>
<td>V</td>
<td>^2)</td>
</tr>
<tr>
<td>Bin heap</td>
<td>O(</td>
<td>V</td>
<td>log</td>
</tr>
</tbody>
</table>

Running time?

<table>
<thead>
<tr>
<th>Heap</th>
<th>V * delMin</th>
<th>E * decreaseKey</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>Array</td>
<td>O(</td>
<td>V</td>
<td>^2)</td>
</tr>
<tr>
<td>Bin heap</td>
<td>O(</td>
<td>V</td>
<td>log</td>
</tr>
<tr>
<td>Fib heap</td>
<td>O(</td>
<td>V</td>
<td>log</td>
</tr>
</tbody>
</table>
### Shortest paths

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

<table>
<thead>
<tr>
<th>What is single source?</th>
</tr>
</thead>
</table>

### Shortest paths

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

Many other variants:
- graphs with negative edges
- all pairs shortest paths
- ...