lecture 7
single source shortest paths
Dijkstra's algorithm

[graph can be directed or undirected]

Resources for this lecture
• Sedgewick Algorithms 2
  https://class.coursera.org/algs4partII-002/lecture/20
• Roughgarden's Algorithms 1
  https://class.coursera.org/algo-004/lecture/57

Warmup Problem
Given a vertex $s$ ("source/start") in a graph (directed or not), find the "shortest" path (fewest edges) to all reachable vertices.

Solution: use breadth first search

Finds vertices reachable by paths of length $2^i = 0, 1, 2, \ldots$

BFS (review COMP 250)

$S = \{\}$ // set of vertices for which we know shortest path
for each $u$ in $V$, $d[u] = \infty$ // length of shortest path
parent[$u$] = null // previous node in shortest path
visited[$u$] = false
initialize empty queue
queue.add(startingVertex)

dist[startingVertex] = 0
visited[starting Vertex] = true

while queue is not empty
  $u = queue.remove()$
  add $u$ to $S$
  for each edge $(u,v)$ // for each $v$ in $u$'s adjList
    if ![visited[v]]
      queue.add(v)
      visited[v] = true // shortest known path to $v$ is via $u$
      parent[v] = u // shortest known path to $v$ is via $u$
      dist[v] = dist[u] + 1

BFS defines a rooted tree which is typically different than the DFS tree. The edges of the tree are defined by the 'parent'.

Given a weighted graph and a starting vertex $s$, what is the shortest cost path (sum of weights) from $s$ to each vertex $v$.

Assume:
- edge weights (cost) are $\geq 0$.
- all vertices are reachable from $s$.

Shortest paths in weighted graphs?

Applications: Google Maps, TomTom.
Dijkstra's Algorithm: Given a graph $G = (V,E)$ and vertex $s$ in $V$.

1. For all $u$ in $V$, initialize $\text{dist}[u] = \infty$.
2. $S = \{s\}$
3. $\text{dist}[s] = 0$
4. While $|S| < |V|$
   a. Choose "best crossing edge" $(u,v)$ as defined earlier
   b. Add $v$ to $S$
   c. $\text{dist}[v] = \text{dist}[u] + \text{cost}(u, v)$

Main idea: at any stage of the algorithm, the vertices are partitioned into two sets (similar to BFS).

Vertices for which we know the shortest cost path from $s$. (includes vertex $s$)

The chosen edge $(u,v)$ might not be the shortest crossing edge from $S$ to $V \setminus S$.

E.g.

Dijkstra would choose the 3 edge next instead of the 2 edge.

Notation / Definitions

- $(u,v)$ is an edge with weight/cost $\text{cost}(u,v) \geq 0$
- "path length" = sum of costs of edges on a path from $s$ to $v$
- $\text{dist}[v]$ is the smallest "path length" (length of the shortest path) from $s$ to $v$.

Dijkstra’s Algorithm: Given a graph $G = (V,E)$ and vertex $s$ in $V$.

For all $u$ in $V$, initialize $\text{dist}[u] = \infty$

$S = \{s\}$

$\text{dist}[s] = 0$

While $|S| < |V|$

- Choose "best crossing edge" $(u,v)$ as defined earlier
- Add $v$ to $S$
- $\text{dist}[v] = \text{dist}[u] + \text{cost}(u, v)$

\[ S \quad \text{and} \quad V \setminus S \]
Like BFS, Dijkstra grows a rooted tree.

Unlike BFS, which increments path lengths by 1, Dijkstra increments path lengths by an edge cost (possibly different than 1).
Claim: during the execution of Dijkstra's algorithm, for each \( u \) in \( S \), the algorithm has found the shortest path from \( s \) to \( u \).

Proof (by induction on the size of \( S \)):

Base case: if \( |S| = 1 \), then \( \text{dist}(s) = 0 \). (trivial)

Induction hypothesis: the claim is true when \( |S| = k \).

Induction step: if claim is true for \( |S| = k \), then claim is true for \( |S| = k+1 \).

Let \( v \) be the \( k+1 \)-st node added to \( S \) by Dijkstra.

Let \( (u,v) \) be the crossing edge chosen by Dijkstra, where \( u \) in \( S \).

Let \( P \) be the path from \( s \) to \( v \) found by Dijkstra, namely \((s, \ldots, u, v)\).

Dijkstra's algorithm seems simple enough but there are a few subtleties:

1) proving it is correct

2) using a priority queue to efficiently find the best crossing edge

Let \( v \) be the \( k+1 \)-st node added to \( S \) by Dijkstra.

Let \( (u,v) \) be the crossing edge chosen by Dijkstra, where \( u \) in \( S \).

Let \( P \) be the path from \( s \) to \( v \) found by Dijkstra, namely \((s, \ldots, u, v)\).

By induction hypothesis, \((s, \ldots, u)\) the shortest path from \( s \) to \( u \).

Could there be a shorter path from \( s \) to \( v \) than the one found by Dijkstra?
Take any other path \( P^* \) from \( s \) to \( v \). Let \( x \) be the last vertex in \( S \) along this path \( P^* \), and let \( y \) be the vertex that follows \( x \). By definition, \( y \) is in \( V \setminus S \).

\[
\text{cost}(P^*) \geq \text{dist}[x] + \text{cost}(x,y) \quad \text{since edge costs are non-negative and } y \text{ might be different than } v.
\]

\[
\geq \text{dist}[u] + \text{cost}(u, v) \quad \text{since Dijkstra chose } v \text{ rather than } y \text{ at step } k+1
\]

\[
= \text{cost}(P)
\]

[Proof still works if \( x \) is \( u \) and/or \( y \) is \( v \).]

Q: How to decide which is the next vertex to be added to \( S \)?

A: Use a priority queue (pq), whose priorities are distances of the shortest known path from \( s \) to \( v \), where \( v \) in \( V \setminus S \).

Dijkstra's algorithm (find shortest path from a given vertex \( s \))

// initialization

\( S = \{ \} \)

initialize empty priority queue pq

for each \( u \) in \( V \)

\( \text{parent}[u] = \text{null} \quad // \ u.\text{parent} = \text{null} \)

pq.add( \( u \), infinity) \quad // add all vertices to pq

pq.changePriority( s, 0) \quad // see next slide for the main loop

while pq is not empty {

\( u = \text{pq.getMinVertex()} \quad // \text{slightly different from} \)

\( \text{dist}[u] = \text{pq.removeMinDistance()} \quad // \text{IndexedHeap} \)

\( S.\text{add}(u) \quad // \text{in assignments} \)

for each \( v \) in \( u.\text{adjList} \)

\( \text{if } v \text{ in } V \setminus S \) {

\( \text{tmpDistToV} = \text{dist}[u] + \text{cost}(u,v) \)

\( \text{if } \text{tmpDistToV} < \text{pq.getPriority}(v) \)

\( \text{pq.setPriority}(v, \text{tmpDistV}) \)

\( \text{parent}[v] = u \)

}\n
}
How much space and time does Dijkstra’s algorithm take?

Recall we assumed there is a path from $s$ to all nodes.

**Space:** Each vertex $u$ in $V$ enters the priority queue once. So the size of the priority queue is at most $|V|$.

**Time:** There is at most one `pq.changePriority()` per edge in the graph. If the priority queue is implemented as a heap, then each update takes $O(\log |V|)$ time.

So Dijkstra’s algorithm takes $O(|E| \log |V|)$ time.

Assignment 2

Implement two versions of Dijkstra

1) $O(|E| \log |V|)$ version that I just described.

2) $O(|E| \log |E|)$ version

Store edges rather than vertices in the priority queue.