Graphs

Part II: SP and MST

Laura Toma
Algorithms (csci2200), Bowdoin College

Topics

Weighted graphs

- each edge (u, v) has a weight denoted $w(u, v)$ or w_{uv}
- stored in the adjacency list or adjacency matrix

The weight of a path $p = (v_1, v_2, v_3, ...v_k)$ is the sum of the weights of the edges on the path.

Problems:

- shortest paths (SP)
- minimum spanning tree (MST)

Shortest paths

Variants:

- P2P SP: given two vertices u, v: find SP from u to v
- SSSP: given a vertex u, find SP from u to all vertices in G
- APSP: find SP between any two vertices (u, v)

Shortest paths

Variants:

- P2P SP: given two vertices u, v: find SP from u to v
- SSSP: given a vertex u, find SP from u to all vertices in G
- APSP: find SP between any two vertices (u, v)

Notes:

- SPs not well-defined when graph has a negative cycle
  - might want shortest path that has no cycles $\Leftarrow$ NPC
- When all edge weights are equal, SP can be computed by BFS.
  - computing shortest paths in terms of number of edges on the path is a special case of the SP problem

Point-to-point SP

Problem: given two vertices u, v: find SP from u to v

No algorithm is known for computing SP(u,v) that's better, in the worst case, than running SSSP(u).

APSP

Problem: For any u, v: find SP from u to v

Can run SSSP(u) $|V|$ times, once for each vertex u.

Better algorithms exist.

SSSP

G is a weighted (directed or undirected) graph.
Problem: Given vertex s, find SP from s to all v in G.

If G has positive weights: Diskstra's algorithm
Otherwise: Bellman-Ford algorithm

SSSP: Dijkstra'a algorithm

SSSP(s)
Idea: for each vertex v, maintain $d[v]$ as the best known shortest path to v (from s)

Initially: $d[s] = 0$ and $d[v] = \infty$ for all $v \neq s$

Idea: Greedy: Visit first the vertex with smallest d.

Implementation: use a priority queue.

SSSP: Dijkstra'a algorithm

Idea: for each vertex v, maintain $d[v]$ as the best known shortest path to v (from s)

- Initialize: $d[s] = 0$ and $d[v] = \infty$ for all $v \neq s$. For every $v \in V$, insert $(v, d[v])$ in PQ.
- while PQ not empty
  - v = deleteMin(PQ)
  - for each outgoing edge (v, u): relax (v, u)

relax(v, u) tests whether we can improve the SP to u by going through v

- if $d[u] > d[v] + w_{vu}$ then
  - $d[u] = d[v] + w_{vu}$
  - decreaseKey of u in PQ to $d[u]$

SSSP: Dijkstra'a algorithm

Idea: for each vertex v, maintain $d[v]$ as the best known shortest path to v (from s)

- Initialize: $d[s] = 0$ and $d[v] = \infty$ for all $v \neq s$. For every $v \in V$, insert $(v, d[v])$ in PQ.
- while PQ not empty
  - $v =$ deleteMin(PQ)
  - for each outgoing edge (v, u): relax (v, u)

relax(v, u) tests whether we can improve the SP to u by going through v

- if $d[u] > d[v] + w_{vu}$ then
  - $d[u] = d[v] + w_{vu}$
  - decreaseKey of u in PQ to $d[u]$

$O(|V| + |E|) + |V| \cdot$ PQ-insert $+ |V| \cdot$ PQ-delete $+ |E| \cdot$ PQ-decreaseKey

SSSP: Dijkstra'a algorithm

Idea: for each vertex v, maintain $d[v]$ as the best known shortest path to v (from s)

- Initialize: $d[s] = 0$ and $d[v] = \infty$ for all $v \neq s$. For every $v \in V$, insert $(v, d[v])$ in PQ.
- while PQ not empty
  - $v =$ deleteMin(PQ)
  - for each outgoing edge (v, u): relax (v, u)

relax(v, u)

- if $d[u] > d[v] + w_{vu}$ then
  - $d[u] = d[v] + w_{vu}$
  - decreaseKey of u in PQ to $d[u]$

Analysis: With a heap, runs in $O(E \lg V)$

SSSP: Dijkstra'a algorithm

Let S denote the set of vertices that have been deleted from PQ.

Correctness: At every iteration of the `while` loop, the following invariants hold:

1. (I1) for any $v \in V - S$, $d[v]$ is the length of the shortest path from s to v among all paths that go only through vertices of S.
2. (I2) for any $v \in S$, $d[v]$ is the length of the shortest path from s to v.

SSSP: Dijkstra'a algorithm

Let S denote the set of vertices that have been deleted from PQ.
Correctness: At every iteration of the `while` loop, the following invariants hold:

1. (I1) for any $v \in V - S$, $d[v]$ is the length of the shortest path from s to v among all paths that go only through vertices of S.
2. (I2) for any $v \in S$, $d[v]$ is the length of the shortest path from s to v.

Prove by induction on the size of S.

SSSP: Dijkstra'a algorithm

At every iteration of the `while` loop, the following holds:
(I1) for any $v \in V - S$, $d[v]$ is the length of the shortest path from s to v among all paths that go only through vertices of S.

Basecase: (I1) is trivially true before the first iteration of the while loop, when S is empty.

Assume (I1) is true *before* an iteration of the while loop. We'll prove that it's true *after* this iteration.

After adding v to S, the only paths that can change are to those vertices that are adjacent to v. The algorithm checks them and releases them.

SSSP: Dijkstra'a algorithm

At every iteration of the `while` loop, the following holds:
(I2) for any $v \in S$, $d[v]$ is the length of the shortest path from s to v.

Basecase: (I2) is trivially true before the first iteration of the while loop, when S is empty.

Assume (I2) is true *before* an iteration of the while loop.
We'll prove that it's true *after* this iteration.

As we are adding v to S, assume by contradiction that the length of the shortest path to v is $|\delta(s,v)| < d[v]$. Let (x,y) be the first edge on $\delta(s,v)$ leaving S (x last vertex in S).

- $d[v] > \delta(s,v) = \delta(s,y) + \delta(y,v)$
- $d[y] = \delta(s,y)$ by (I1)
- $d[v] < d[y]$ because v comes out of PQ before y

$\Rightarrow \delta(y,v) < 0$ impossible

SSSP: Dijkstra'a algorithm

What happens if we run Dijkstra's algorithm on a graph with negative weights?

SSSP: Dijkstra’a algorithm

What happens if we run Dijkstra’s algorithm on a graph with negative weights?

Find an example of a graph where Dijkstra does not compute the SP correctly.

SSSP with negative weights

Note: If G is undirected and has negative weights, that immediately means a negative cycle.

SSSP with negative weights

G **directed** graph.

SSSP with negative weights

G **directed** graph.
If G has no negative cycles, then there exists a SP from s to v that is *simple* and hence has $|V| - 1$ edges.

SSSP with negative weights

G **directed** graph.
If G has no negative cycles, then there exists a SP from s to v that is *simple* and hence has $|V| - 1$ edges.

Let $\delta(u, v)$ denote the shortest path from u to v.
Start with $d[v] = \infty$ and progresively refine it, until $d[v] = |\delta(s, v)|$
Similar to Dijkstra: Dijkstra relaxes edges in greedy order of increasing $d[]$; that does not work for negative edges

SSSP with negative weights

G **directed** graph.

Bellman-Ford algorithm (s):

- Initialize: $d[s] = 0$ and $d[v] = \infty$ for all $v \neq s$.
- for $i = 1$ to $|V| - 1$ do:
  - for every edge (v, u) in G: relax(v, u)

relax(v, u)

- if $d[u] > d[v] + w_{vu}$ then
  - $d[u] = d[v] + w_{vu}$

Bellman-Ford

WHY does this work?
Intuition: look at the number of edges along a shortest path (SP) from s

Bellman-Ford

WHY does this work?
Intuition: look at the number of edges along a shortest path (SP) from s

- initially, only $d[s]$ is correct. Put differently, all SP that consist of 0 edges are correctly computed.

Bellman-Ford

WHY does this work?

Intuition: look at the number of edges along a shortest path (SP) from s

- initially, only $d[s]$ is correct. Put differently, all SP that consist of 0 edges are correctly computed.
- after round 1: all SP from s that consist precisely of 1 edge are correctly computed.

Bellman-Ford

WHY does this work?
Intuition: look at the number of edges along a shortest path (SP) from s

- initially, only $d[s]$ is correct. Put differently, all SP that consist of 0 edges are correctly computed.
- after round 1: all SP from s that consist precisely of 1 edge are correctly computed.
- after round 2: all SP from s that consist precisely of 2 edges are correctly computed.

Bellman-Ford

WHY does this work?

Intuition: look at the number of edges along a shortest path (SP) from s

- initially, only $d[s]$ is correct. Put differently, all SP that consist of 0 edges are correctly computed.
- after round 1: all SP from s that consist precisely of 1 edge are correctly computed.
- after round 2: all SP from s that consist precisely of 2 edges are correctly computed.
- ...
- after round i: all SP from s that consist precisely of i edges are correctly computed.

Bellman-Ford

WHY does this work?

Intuition: look at the number of edges along a shortest path (SP) from s

Let $\delta(u, v)$ denote the shortest path from u to v.

Let $OPT(v, i)$ denote the length of the shortest path from s to v among all paths containing $\leq i$ edges.

Bellman-Ford

WHY does this work?

Intuition: look at the number of edges along a shortest path (SP) from s

Let $\delta(u, v)$ denote the shortest path from u to v.

Let $OPT(v, i)$ denote the length of the shortest path from s to v among all paths containing $\leq i$ edges.

We have:

Bellman-Ford

WHY does this work?

Intuition: look at the number of edges along a shortest path (SP) from s

Let $\delta(u, v)$ denote the shortest path from u to v.

Let $OPT(v, i)$ denote the length of the shortest path from s to v among all paths containing $\leq i$ edges.

We have:

- $OPT(s, 0) = 0$

Bellman-Ford

WHY does this work?

Intuition: look at the number of edges along a shortest path (SP) from s

Let $\delta(u, v)$ denote the shortest path from u to v.

Let $OPT(v, i)$ denote the length of the shortest path from s to v among all paths containing $\leq i$ edges.

We have:

- $OPT(s, 0) = 0$
- $OPT(v, 0) = \infty$ for any $v \neq s$

WHY does this work?

Intuition: look at the number of edges along a shortest path (SP) from s

Let $\delta(u, v)$ denote the shortest path from u to v.

Let $OPT(v, i)$ denote the length of the shortest path from s to v among all paths containing $\leq i$ edges.

We have:

- $OPT(s, 0) = 0$
- $OPT(v, 0) = \infty$ for any $v \neq s$
- $OPT(v, |V| - 1) = |\delta(s, v)|$

Bellman-Ford

WHY does this work?

Intuition: look at the number of edges along a shortest path (SP) from s

Let $\delta(u, v)$ denote the shortest path from u to v.

Let $OPT(v, i)$ denote the length of the shortest path from s to v among all paths containing $\leq i$ edges.

We have:

- $OPT(s, 0) = 0$
- $OPT(v, 0) = \infty$ for any $v \neq s$
- $OPT(v, |V| - 1) = |\delta(s, v)|$
- $OPT(v, i) = min\{OPT(v, i-1), min_{u|(u,v)}\{OPT(u, i-1) + w_{uv}\}\}$

Bellman-Ford

WHY does this work?

Intuition: look at the number of edges along a shortest path (SP) from s

Let $\delta(u, v)$ denote the shortest path from u to v.

Let $OPT(v, i)$ denote the length of the shortest path from s to v among all paths containing $\leq i$ edges.

We have:

- $OPT(s, 0) = 0$
- $OPT(v, 0) = \infty$ for any $v \neq s$
- $OPT(v, |V| - 1) = |\delta(s, v)|$
- $OPT(v, i) = min\{OPT(v, i-1), min_{u|(u,v)}\{OPT(u, i-1) + w_{uv}\}\}$

Claim: After round i in Bellman-Ford we have $d[v] = OPT(v, i)$

Bellman-Ford

Running time: $O(V \cdot E)$

After $V - 1$ rounds, $d[v] = OPT(v, |V| - 1) = |\delta(s, v)|$

Bellman-Ford

Running time: $O(V \cdot E)$
After $V - 1$ rounds, $d[v] = OPT(v, |V| - 1) = |\delta(s, v)|$

What happens if we do more rounds? (beyond $|V| - 1$)

Bellman-Ford

Running time: $O(V \cdot E)$
After $V - 1$ rounds, $d[v] = OPT(v, |V| - 1) = |\delta(s, v)|$

What happens if we do more rounds? (beyond $|V| - 1$)

- the values $d[v]$ will not decrease any further.... unless.....there's a negative cycle

Bellman-Ford

Running time: $O(V \cdot E)$
After $V - 1$ rounds, $d[v] = OPT(v, |V| - 1) = |\delta(s, v)|$

What happens if we do more rounds? (beyond $|V| - 1$)

- the values $d[v]$ will not decrease any further.... unless.....there's a negative cycle

Negative cycles: What happens if G negative cycles?

Bellman-Ford

Running time: $O(V \cdot E)$
After $V - 1$ rounds, $d[v] = OPT(v, |V| - 1) = |\delta(s, v)|$

What happens if we do more rounds? (beyond $|V| - 1$)

- the values $d[v]$ will not decrease any further.... unless.....there's a negative cycle

Negative cycles: What happens if G negative cycles?

- $d[v]$ are not SP (there are no SP)
- some values $d[v]$ will keep decreasing

Bellman-Ford

Running time: $O(V \cdot E)$
After $V - 1$ rounds, $d[v] = OPT(v, |V| - 1) = |\delta(s, v)|$

What happens if we do more rounds? (beyond $|V| - 1$)

- the values $d[v]$ will not decrease any further.... unless.....there's a negative cycle

Negative cycles: What happens if G negative cycles?

- $d[v]$ are not SP (there are no SP)
- some values $d[v]$ will keep decreasing

$\longrightarrow$ Bellman-Ford can be used to test for the existence of negative cycles in the graph:

Bellman-Ford

G **directed** graph.
Bellman-Ford algorithm (s):

- Initialize: $d[s] = 0$ and $d[v] = \infty$ for all $v \neq s$.
- for $i = 1$ to $|V| - 1$ do:
  - for every edge (v, u) in G: relax(v, u)
- for each edge (v, u) in G: if $d[v] + w_{vu} < d[u] \Rightarrow$ NEG CYCLE

Bellman-Ford

G **directed** graph.
Bellman-Ford algorithm (s):

- Initialize: $d[s] = 0$ and $d[v] = \infty$ for all $v \neq s$.
- for $i = 1$ to $|V| - 1$ do:
  - for every edge (v, u) in G: relax(v, u)
- for each edge (v, u) in G: if $d[v] + w_{vu} < d[u] \Rightarrow$ NEG CYCLE

Note: detects negative cycle *reachable from* s. Can be extended to detect if G has any negative cycle.

SSSP

Summary of known algorithms:

- G unweighted
  - BFS in $O(V + E)$
- G DAG
  - dynamic programming in $O(V + E)$
- G directed, no negative weights
  - Dijkstra's algorithm in $O(E \lg V)$
- G directed, no negative cycles
  - Bellman-Ford algorithm in $O(V \cdot E)$