CSCI3160 Design and Analysis of Algorithms (2025 Fall)

Review: Single Source Shortest Paths with Non-Negative Weights

Instructor: Xiao Liang[1]

Department of Computer Science and Engineering
The Chinese University of Hong Kong

[1]These slides are primarily based on materials prepared by Prof. Yufei Tao (please refer to Prof. Tao's version from 2024 Fall for the original content). Some modifications have been made to better align with this year's teaching progress, incorporating student feedback, in-class interactions, and my own teaching style and research perspective.

We will now commence our discussion on the **single source shortest path** (SSSP) problem. This lecture will start with **Dijkstra's algorithm**, which should have been covered in CSCI2100.

Notation and the SSSP Problem

Weighted Graphs

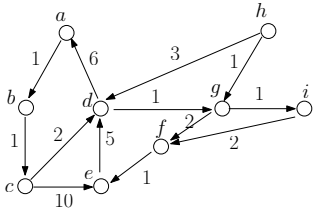
Let $G = (V, E)$ be a simple[2] directed graph.

Let w be a function that maps each edge $e \in E$ to a **non-negative** integer value $w(e)$, which we call the **weight** of e.

G and w together define a **weighted simple directed graph**.

[2] Here, "simple" means no self-loops — i.e., no edge from a vertex to itself.

Example



The integer on each edge indicates its weight. For example, $w(d, g) = 1$, $w(g, f) = 2$, and $w(c, e) = 10$.

Shortest Path

Consider a path in G: $(v_1, v_2), (v_2, v_3), ..., (v_\ell, v_{\ell+1})$, for some integer $\ell \geq 1$. We define the path's **length** as

$$\sum_{i=1}^{\ell} w(v_i, v_{i+1}).$$

A **shortest path** from u to v has the minimum length among all the paths from u to v. Denote by $spdist(u, v)$ the length of a shortest path from u to v.

If v is unreachable from u, $spdist(u, v) = \infty$.

Problem Statement

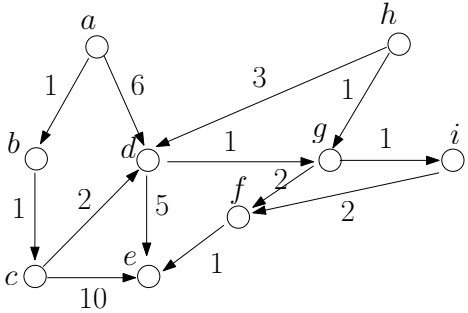
Single Source Shortest Path (SSSP) with Non-Negative Weights

Let $G = (V, E)$ be a simple directed graph, where function $w(\cdot)$ maps every edge of E to a non-negative value. Given a **source vertex** s in V, we want to find a shortest path from s to t for **every** vertex $t \in V$ reachable from s.

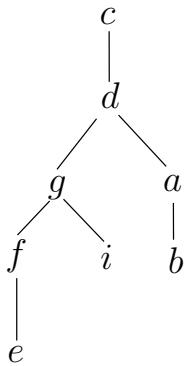
The output is a **shortest path tree** T:

- The vertex set of T contains all vertices reachable from s.
- The root of T is s.
- For each node $u \in V$, the root-to-u path of T is a shortest path from s to u in G.

Example



(a) Original graph G



(b) A shortest path tree for source vertex c

Think: why cannot we simply use DFS to solve this problem?

Dijkstra's algorithm

Algorithm Description (1/2)

Edge Relaxation

For every vertex $v \in V$, we will — at all times — maintain a value $dist(v)$ equal to the shortest path length from s to v **found so far**.

Relaxing an edge (u, v) means:

- If $dist(v) \leq dist(u) + w(u, v)$, do nothing;
- Otherwise, reduce $dist(v)$ to $dist(u) + w(u, v)$.

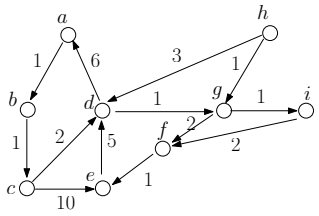
Algorithm Description (2/2)

Dijkstra's Algorithm

1. Set $parent(v) \leftarrow$ nil for all vertices $v \in V$
2. Set $dist(s) \leftarrow 0$ and $dist(v) \leftarrow \infty$ for each vertex $v \in V \setminus \{s\}$
3. Set $S \leftarrow V$ /* just don't want to messed up with the original vertex set */
4. Repeat the following until S is empty:
   - Remove from S the vertex u with the **smallest** $dist(u)$.
   - Relax every outgoing edge (u, v) of u.
     - If $dist(v)$ drops after the relaxation, set $parent(v) \leftarrow u$.

Example

Suppose that the source vertex is c.



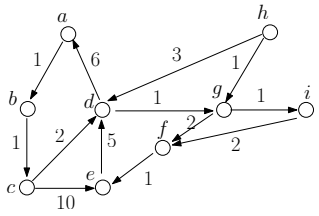
vertex v	$dist(v)$	$parent(v)$
a	∞	nil
b	∞	nil
c	0	nil
d	∞	nil
e	∞	nil
f	∞	nil
g	∞	nil
h	∞	nil
i	∞	nil

$S = \{a, b, c, d, e, f, g, h, i\}$.

/* S now contains all the vertices, and c is the vertex with the smallest $dist()$. */

Example

Relax the out-going edges of c.



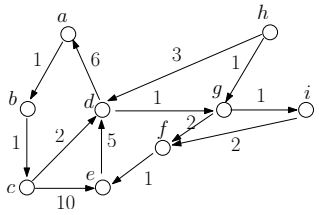
vertex v	$dist(v)$	$parent(v)$
a	∞	nil
b	∞	nil
c	0	nil
d	2	c
e	10	c
f	∞	nil
g	∞	nil
h	∞	nil
i	∞	nil

$S = \{a, b, d, e, f, g, h, i\}$.

/* c has been removed from S, leaving d as the vertex with the smallest $dist()$. */

Example

Relax the out-going edges of d.



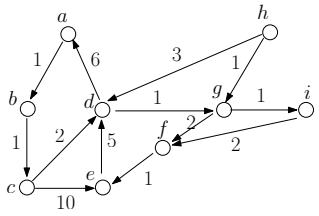
vertex v	$dist(v)$	$parent(v)$
a	8	d
b	∞	nil
c	0	nil
d	2	c
e	10	c
f	∞	nil
g	3	d
h	∞	nil
i	∞	nil

$S = \{a, b, e, f, g, h, i\}$.

/* d has been removed from S, leaving g as the vertex with the smallest $dist()$. */

Example

Relax the out-going edges of g.



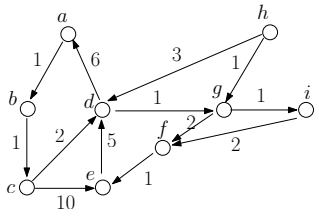
vertex v	$dist(v)$	$parent(v)$
a	8	d
b	∞	nil
c	0	nil
d	2	c
e	10	c
f	5	g
g	3	d
h	∞	nil
i	4	g

$S = \{a, b, e, f, h, i\}$.

/* g has been removed from S, leaving i as the vertex with the smallest $dist()$. */

Example

Relax the out-going edges of i.



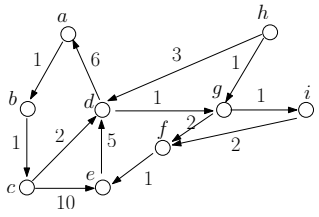
vertex v	$dist(v)$	$parent(v)$
a	8	d
b	∞	nil
c	0	nil
d	2	c
e	10	c
f	5	g
g	3	d
h	∞	nil
i	4	g

$S = \{a, b, e, f, h\}$.

/* i has been removed from S, leaving f as the vertex with the smallest $dist()$. */

Example

Relax the out-going edges of f.



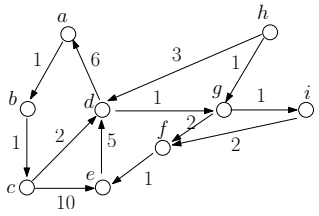
vertex v	$dist(v)$	$parent(v)$
a	8	d
b	∞	nil
c	0	nil
d	2	c
e	6	f
f	5	g
g	3	d
h	∞	nil
i	4	g

$S = \{a, b, e, h\}$.

/* f has been removed from S, leaving e as the vertex with the smallest $dist()$. */

Example

Relax the out-going edges of e.



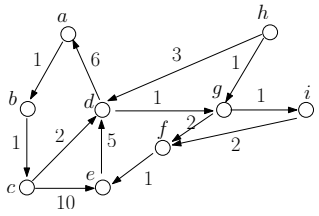
vertex v	$dist(v)$	$parent(v)$
a	8	d
b	∞	nil
c	0	nil
d	2	c
e	6	f
f	5	g
g	3	d
h	∞	nil
i	4	g

$S = \{a, b, h\}$.

/* e has been removed from S, leaving a as the vertex with the smallest $dist()$. */

Example

Relax the out-going edges of a.



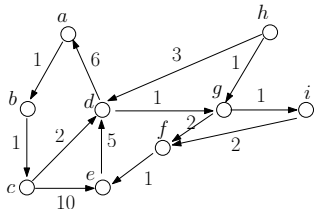
vertex v	$dist(v)$	$parent(v)$
a	8	d
b	9	a
c	0	nil
d	2	c
e	6	f
f	5	g
g	3	d
h	∞	nil
i	4	g

$S = \{b, h\}$.

/* a has been removed from S, leaving b as the vertex with the smallest $dist()$. */

Example

Relax the out-going edges of b.



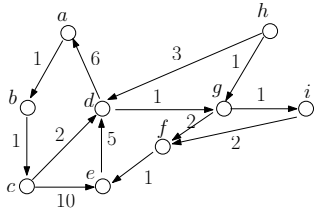
vertex v	$dist(v)$	$parent(v)$
a	8	d
b	9	a
c	0	nil
d	2	c
e	6	f
f	5	g
g	3	d
h	∞	nil
i	4	g

$S = \{h\}$.

/* b has been removed from S, leaving h as the vertex with the smallest $dist()$. */

Example

Relax the out-going edges of h.



vertex v	$dist(v)$	$parent(v)$
a	8	d
b	9	a
c	0	nil
d	2	c
e	6	f
f	5	g
g	3	d
h	∞	nil
i	4	g

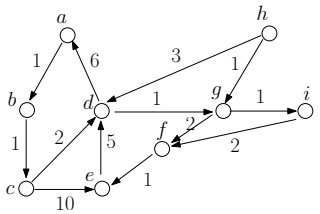
$S = \{\}$.

All the shortest path distances are now final.

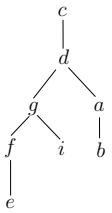
Notice how h is not included in the final Shortest Path Tree

Constructing the Shortest Path Tree

For every vertex v, if $u = parent(v)$ is not nil, then make v a child of u.



vertex v	$parent(v)$
a	d
b	a
c	nil
d	c
e	f
f	g
g	d
h	nil
i	g



Running Time and Correctness

Running time:

You should be able to implement Dijkstra's algorithm to make sure that it runs in $O((|V| + |E|) \cdot \log |V|)$ time.

- Using advanced (graduate-level) data structures, we can further reduce the time to

$$O(|V| \log |V| + |E|).$$

Proof of Correctness:

Since Dijkstra's algorithm was introduced in *CSCI2100 Data Structures*, we will not repeat the proof of its correctness here. If you would like to review it, please refer to **Section 22.3 of [CLRS]**.

Graphs with Negative Weights?

Graphs with Negative Weights?

Previously, we claimed that:

- Dijkstra's algorithm does **not** work if edges can take negative weights.

Let's now take a closer look at this statement.

Consider the following example graph with negative weights:

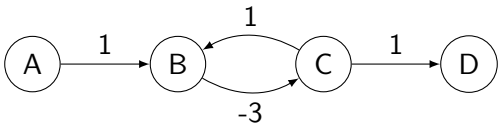


Figure: Graph with negative cycle

Question: what is the shortest path from A to D?
This question is ill-defined. Consider the following paths:

- Length -2: $A \to B \to C \to B \to C \to D$
- Length -4: $A \to B \to C \to B \to C \to B \to C \to D$
- Length -6: $A \to B \to C \to B \to C \to B \to C \to B \to C \to D$
- $\ldots\ldots$

Consider the following example graph with negative weights:

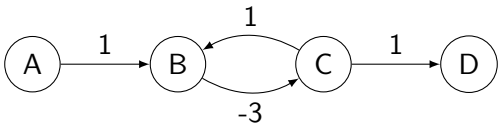


Figure: Graph with negative cycle

This problem is due to the **negative cycle** formed by B and C.

This has nothing to do with Dijkstra's algorithm — it is a definitional issue.
Can we fix it by simply asking for the shortest path that does not contain a cycle?

- Yes. This version of the problem is called the **Shortest Simple Path** problem.

Shortest Simple Path Problem

Input:

- A directed graph $G = (V, E)$, possibly with negative edge weights.
- Two vertices: a source $s \in V$ and a destination $t \in V$.

Goal: Find the shortest path from s to t that is **simple**, i.e., it does not visit any vertex more than once.

This is a perfectly valid problem.

However, this problem is known to be **NP-hard**, meaning that no polynomial-time algorithm is currently known for solving it — including Dijkstra's algorithm. (Refer to the supplementary material toward the end of the slides.)

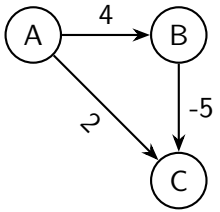
Graphs with Negative Weights?

Let's restrict our attention to graphs that contain no negative cycles.

Can Dijkstra's algorithm solve the shortest path problem on directed simple graphs that may contain negative edge weights **but are guaranteed to have no negative cycles**?

Unfortunately, the answer is still **No**!

Counterexample Graph



Goal: Find the shortest path from A to C.

Convince yourself: Dijkstra's algorithm, starting from node A, will find the shortest path to C to have a length of 2. That is, it will not discover the path $A \to B \to C$.

Summary

Shortest path problem:

1. Non-negative weights: Dijkstra's algorithm
2. Negative weights:
   1. negative cycles:
      - Shortest Paths are not well-defined.
      - Shortest Simple Paths are well defined. However, this is a NP-hard problem.
   2. no negative cycles:
      - Shortest Paths are well-defined.
      - But Dijkstra's algorithm does not work.

What should we do with graphs that contain no negative cycles?

- Bellman-Ford algorithm (next lecture).

(Supplementary Material) Reduce Hamiltonian Path to Shortest Simple Path

This supplementary material will not be included in quizzes or exams.

Problem Definitions

Hamiltonian Path (HP):

- Given a graph $G = (V, E)$ and two vertices s and t.
- Does there exist a simple path from s to t that visits **every vertex exactly once**?

Shortest Simple Path (SSP):

- Given a weighted graph $G = (V, E)$ with weights (possibly negative), find the **shortest simple path** from s to t.
- A simple path is a path that visits no vertex more than once.

Goal of the Reduction

- We reduce the known NP-hard problem **Hamiltonian Path** to **Shortest Simple Path**.
- This shows that solving SSP efficiently would allow us to solve HP efficiently.
- Therefore, SSP is at least as hard as HP $\Rightarrow$ SSP is **NP-hard**.

Reduction Strategy

Given: An instance of Hamiltonian Path:

- An unweighted graph $G = (V, E)$
- Two vertices s and t

Construct: A new graph G':

- Use the same graph structure as G
- Assign weight -1 to every edge

Question: What is the shortest simple path from s to t in G'?

Key Observation

- A Hamiltonian Path from s to t visits all $|V|$ vertices.
- Such a path has exactly $|V| - 1$ edges.
- Since each edge has weight -1, the total weight is:

$$\text{Total weight} = -(|V| - 1)$$

- Any other simple path uses fewer edges $\Rightarrow$ less negative total weight.

Conclusion of the Reduction

If the shortest simple path from s to t has weight $-(|V|-1)$:

- Then a Hamiltonian Path exists.

If not:

- Then no Hamiltonian Path exists.

Therefore: Solving the Shortest Simple Path problem allows us to solve Hamiltonian Path.

$$\Rightarrow \text{Shortest Simple Path is NP-hard}$$

Summary

- Hamiltonian Path is NP-hard.
- We reduced HP to SSP by assigning edge weights of -1.
- Deciding whether a path of weight $-(|V| - 1)$ exists solves HP.
- $\Rightarrow$ Shortest Simple Path is NP-hard.

Q.E.D.