1 Introduction

Single-source shortest path problem:
- Input: A weighted directed graph $G$ with no negative weights, stored as adjacency lists, and a start vertex $s$ in $G$.
- Output: For each vertex $v$ in $G$, a path in $G$ from $s$ to $v$ that minimizes the total weight of edges crossed.

Key idea: Subpaths of shortest paths are also shortest paths.
- If $v_i \rightsquigarrow v_k \rightsquigarrow v_j$ is a shortest path from $v_i$ to $v_j$,
- then $v_i \rightsquigarrow v_k$ is a shortest path from $v_i$ to $v_k$,
- and $v_k \rightsquigarrow v_j$ is a shortest path from $v_k$ to $v_j$.

2 Dijkstra’s algorithm

For each vertex $v$, keep track of:
- $v.d$: the length of the shortest known path from $s$ to $v$
- $v.\pi$: a predecessor of vertex $v$ on that path

Use a priority queue $Q$ of vertices, keyed by their $d$ values.
- Start with all nodes in $Q$. Start with each $v.d = \infty$, except at the start node.
- For the node $v$ with the lowest $d$, consider each edge $v \rightarrow u$.
- If $v.d + w(v, u) < u.d$, update $u.d$ and $u.\pi$, then DECREASE_KEY on $u$.

3 Analysis of Dijkstra’s

With a simple array for the priority queue:
- Initialization: $O(V)$
- $V$ EXTRACT_MIN operations: $O(V^2)$
- $E$ DECREASE_KEY operations: $O(E)$
- Total: \( T(n) = O(V) + O(V^2) + O(E) = O(V^2) \)

With a binary heap:
- Initialization: (BUILD_MIN_HEAP): \( O(V) \)
- \( V \) EXTRACT_MIN operations: \( O(V \log V) \)
- \( E \) DECREASE_KEY operations: \( O(E \log V) \)
- Total: \( T(n) = O(V) + O(V \log V) + O(E \log V) = O(E \log V) \)

With a Fibonacci heap:
- Initialization (\( V \) INSERT operations): \( O(V) \)
- \( V \) EXTRACT_MIN operations: \( O(V \log V) \)
- \( E \) DECREASE_KEY operations: \( O(E) \)
- Total: \( T(n) = O(V) + O(V \log V) + O(E) = O(V \log V) + O(E) \)

4 **Shortest path trees**
The predecessor pointers form a **shortest path tree**.

5 **Be careful about negative-weight edges!**
Recall that we assumed that no edges have negative weights. What happens if this assumption is violated?
- Dijkstra’s algorithm may give incorrect results. When?
- The shortest path may not even be well-defined. When?

6 **Bellman-Ford algorithm**
For negative weights, use the Bellman-Ford algorithm instead.
BellmanFord($G, w, s$)

for $v \in G.V$ do
    $v.d = \infty$
    $v.\pi = \text{NIL}$
end for

for $i = 1, \ldots, |G.V| - 1$ do
    for each edge $(u, v)$ in $G.E$ do
        if $u.d + w(u, v) < v.d$ then
            $v.d = u.d + w(u, v)$
            $v.\pi = u$
        end if
    end for
end for

for each edge $(u, v)$ in $G.E$ do
    if $u.d + w(u, v) < v.d$ then
        return "Negative weight cycle found."
    end if
end for