4.4 Shortest Paths in a Graph Revisited

shortest path from computer science department to Einstein’s house
Shortest path network.
- Directed graph $G = (V, E)$.
- Source $s$, destination $t$.
- Length $\ell_e = \text{length of edge } e$ ($\ell_e \geq 0$).

**Shortest path problem**: find shortest directed path from $s$ to $t$.

Cost of path $s$-2-3-5-$t$ = $9 + 23 + 6 + 6 = 44$. 
Dijkstra's Algorithm

Dijkstra's algorithm.

- Maintain a set of explored nodes $S$ for which we have determined the shortest path distance $d(u)$ from $s$ to $u$.
- Initialize $S = \{ s \}$, $d(s) = 0$.
- Repeatedly choose unexplored node $v$ which minimizes

$$
\pi(v) = \min_{e = (u,v): u \in S} d(u) + \ell_e,
$$

add $v$ to $S$, and set $d(v) = \pi(v)$.

![Diagram of Dijkstra's algorithm](image)

- Shortest path to some $u$ in explored part, followed by a single edge $(u, v)$.
Dijkstra's Algorithm

Dijkstra's algorithm.

- \( M = \{\} \)
- \( \forall v \in V \)
  - \( d(v) = \pi(v) = \infty \)
  - \( \pi(s) = 0 \)
- loop
  - LI\(_1\): \( \forall v \in M, d(v) \) is the minimal distance from \( s \) to \( v \) and uses edges only in \( M \)
  - exit when \( V=M \)
  - Choose \( v \) from \( V-M \) with minimal \( \pi(v) \)
  - Add \( v \) to \( M \)
  - \( d(v) = \pi(v) \)
- end loop
- Postcondition: \( \forall v \in V, d(v) \) is the minimal distance from \( s \) to \( v \)
Dijkstra’s Algorithm—Proof of Correctness

Loop invariant initialization

Given:
M = {}
forall v ∈ V
   d(v) = π(v) = ∞
π(s) = 0

Show:
LI1: forall v ∈ M, d(v) is the minimal distance from s to v
   Since M is empty, LI1 is trivially true
Dijkstra's Algorithm—Proof of Correctness

Loop invariant maintenance

Given:

LI₁': forall v ∈ M', d'(v) is the minimal distance from s to v
NOT Exit condition: V≠M
code

Show:

LI₁'': forall v ∈ M'', d''(v) is the minimal distance from s to v
Dijkstra's Algorithm—Proof of Correctness

Loop invariant maintenance
Given:
\( \text{LI}_1', V \neq M, \text{LI}_2', \text{code} \)
Show:
\( \text{LI}_1'': \forall v \in M'', d''(v) \) is the minimal distance from \( s \) to \( v \) and uses edges only between vertices in \( M \)

\( M'' \) is \( M' \) plus one new node, \( v \). The values of \( d \) for elements of \( M' \) are unchanged.
Choose \( v \) from \( V-M \) with minimal \( \pi(v) \)
Add \( v \) to \( M \)
\( d(v) = \pi(v) \)

For that one new node in \( M'' \), \( v \), \( d(v) \) is set to \( \pi(v) \), which is the minimal distance from \( s \) to \( v \), using nodes only in \( M' \) plus one extra edge. Any other path to \( v \) would need to go through some other node in \( V-M \), \( y \), which would have higher \( \pi \). Since edge lengths are non-negative \( l(s-y-v) \geq l(s-y) \geq l(s-v) \). Therefore, \( \pi(v) \) is the minimal distance from \( s \) to \( v \).
The code sets \( d(v) \) to that minimal distance. In addition, \( d(u) \) uses edges only between vertices in \( M'' \) and \( v \) is in \( M'' \).
Dijkstra’s Algorithm—Proof of Correctness

Loop termination

Given:

loop
  exit when \( V = M \)
  Choose \( v \) from \( V - M \) with minimal \( \pi(v) \)
  Add \( v \) to \( M \)
  \( d(v) = \pi(v) \)
end loop

Show:

exit condition \((M=V)\) is eventually satisfied

Every time through the loop \(|M|\) increases by one
Dijkstra’s Algorithm—Proof of Correctness

Postcondition correctness

Given:

LI₁: forall v ∈ M, d(v) is the minimal distance from s to v
Exit condition: V=M

Show:

Postcondition: forall v ∈ V, d(v) is the minimal distance from s to v

Since the exit condition shows that V=M, we can rewrite replace M with V in LI₁, yielding the postcondition
Dijkstra's Algorithm: Implementation

For each unexplored node, explicitly maintain $\pi(v) = \min_{e=(u,v): u \in S} d(u) + \ell_e$.

- Next node to explore = node with minimum $\pi(v)$.
- When exploring $v$, for each incident edge $e = (v, w)$, update
  $$\pi(w) = \min \{ \pi(w), \pi(v) + \ell_e \}.$$ 

Efficient implementation. Maintain a priority queue of unexplored nodes, prioritized by $\pi(v)$.

<table>
<thead>
<tr>
<th>PQ Operation</th>
<th>Dijkstra</th>
<th>Binary heap</th>
</tr>
</thead>
<tbody>
<tr>
<td>Insert</td>
<td>$n$</td>
<td>$\log n$</td>
</tr>
<tr>
<td>ExtractMin</td>
<td>$n$</td>
<td>$\log n$</td>
</tr>
<tr>
<td>ChangeKey</td>
<td>$m$</td>
<td>$\log n$</td>
</tr>
<tr>
<td>IsEmpty</td>
<td>$n$</td>
<td>$1$</td>
</tr>
<tr>
<td>Total</td>
<td></td>
<td>$m \log n$</td>
</tr>
</tbody>
</table>
Dijkstra’s Algorithm—Runtime analysis

Executes the for-loop n times
Each time through the loop, must:
- remove from set—$O(1)$
- find minimum $\pi(v)$
- update $\pi(v)$ for all edges adjoining $v$.

If we keep $\pi(v)$ in a priority queue, we can do the following
- Before loop Insert all vertices in priority queue—$O(n \log n)$
- During loop (executed n times)
  - Check Priority Queue IsEmpty—$O(1) \times n = O(n)$
  - ExtractMinimum $\pi(v)$—$O(\log n) \times n = O(n \log n)$
  - For each edge adjoining $v$, ChangeKey—$O(\log n)$
  
  How many times do we execute?
  - We will use each directed edge only once in the algorithm
  - $O(m \log n)$ (total, not per loop execution)

$T(n) = \max(O(n \log n), O(n), O(n \log n), O(m \log n)) = O(m \log n)$
Given minimal distances, how to find minimal path?

Work backwards

Find adjacent vertex such that \( d(u) + l(u, v) = d(v) \)

\[
\begin{align*}
d(t) &= 44 = d(4) + 6 \\
d(4) &= 38 = d(3) + 6 \\
d(3) &= 32 = d(2) + 23 \\
d(2) &= 9 = d(s) + 9
\end{align*}
\]

done!

For each vertex, find adjoining edges

\( T(n, m) = O(n + m) = O(m) \)
Steps for presenting an algorithm

Provide algorithm pseudo-code

Prove correctness

- Loop invariant
  - Initialization
    - Follows from precondition and pre-loop code
  - Maintenance
    - Follows from loop invariant and loop code and not(exit condition)
  - Loop Termination
    - Define some measure of progress
    - Verify progress made in loop

- Postcondition correctness
  - Follows from Loop invariant, exit condition, and post-loop code

Runtime Analysis

- Provide $T(n) = O(\ldots)$
- Make bound as tight as possible