Shortest Paths

Dijkstra  Bellman-Ford  Floyd  All-pairs paths

Lecturer: Georgy Gimel’farb

COMPSCI 220 Algorithms and Data Structures
1. Single-source shortest path

2. Dijkstra’s algorithm

3. Bellman-Ford algorithm

4. All-pairs shortest path problem

5. Floyd’s algorithm
**Paths and Distances Revisited**

**Cost** of a walk / path $v_0, v_1, \ldots, v_k$ in a digraph $G = (V, E)$ with edge weights $\{c(u, v) \mid (u, v) \in E\}$:

$$\text{cost}(v_0, v_1, \ldots, v_k) = \sum_{i=0}^{k-1} c(v_i, v_{i+1})$$

**Distance** $d(u, v)$ between two vertices $u$ and $v$ of $V(G)$: the minimum cost of a path between $u$ and $v$.

**Eccentricity** of a node $u \in V$: $\text{ec}[u] = \max_{v \in V} d(u, v)$.

**Radius** of $G$: the minimum eccentricity of $u \in V$: $\min_{u \in V} \text{ec}[u]$.

**Diameter** of $G$: the maximum eccentricity of $u \in V$: $\max_{u \in V} \text{ec}[u]$.

**Note**: there are analogous definitions for graphs.
Unweighted / Weighted Graphs: Shortest Paths

The shortest path from the vertex $A$ to the vertex $D$:

$$\min\{2_{A,C,D}, 3_{A,C,E,D}, 3_{A,B,F,D}\}$$

$$\min\{9_{A,C,D}, 6_{A,C,E,D}, 10_{A,B,F,D}\}$$
Single-source Shortest Path (SSSP) in $G = (V, E, c)$

Given a source node $v$, find the shortest (minimum weight) path to each other node.

- **Weight of a path**: the sum of weights (costs) on the arcs.
- **BFS works only if all weights $c(u, v); (u, v) \in E$, are equal.**
- **Dijkstra’s algorithm** – one of the known solutions.
  - A **greedy** algorithm: each locally best choice is globally best.
  - Works only if all weights are non-negative.
  - Initial paths: one-arc paths from $s$ to $v$ of weight $\text{cost}(s, v)$.
  - Each step compares the shortest paths with and without each new node.
Single-source Shortest Path (SSSP) in $G = (V, E, c)$

1. Build a list $S$ of visited nodes (say, using a priority queue).
2. Iterative propagation of the shortest paths:
   1. Choose the closest unvisited node $u$ being on a path with internal nodes in $S$.
   2. If adding the node $u$ has established shorter paths, update distances of remaining unvisited nodes $v$ from the source $s$.

Complexity depends on data structures used.

- For a priority queue, such as a binary heap, running time $O((m + n) \log n)$ is possible.
  - If every node is reachable from the source: $O(m \log n)$.
- More sophisticated Fibonacci heaps lead to the best complexity of $O(m + n \log n)$. 
Dijkstra’s Algorithm

algorithm Dijkstra( weighted digraph \((G, c), \text{ node } s \in V(G) \) )

array \(\text{colour}[n] = \{\text{WHITE}, \ldots, \text{WHITE}\}\)
array \(dist[n] = \{c[s, 0], \ldots, c[s, n - 1]\}\)

\(\text{colour}[s] \leftarrow \text{BLACK}\)

while there is a \text{WHITE} node do
  pick a \text{WHITE} node \(u\), such that \(dist[u]\) is minimum
  \(\text{colour}[u] \leftarrow \text{BLACK}\)
  for each \(x\) adjacent to \(u\) do
    if \(\text{colour}[x] = \text{WHITE}\) then
      \(dist[x] \leftarrow \min \{dist[x], dist[u] + c[u, x]\}\)
    end if
  end for
end while

return \(dist\)

end
Dijkstra’s Algorithm: Example 1

**BLACK**

<table>
<thead>
<tr>
<th>List $S$</th>
<th>$\text{dist}[x]$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$a$</td>
<td>0 3 8 $\infty$ $\infty$</td>
</tr>
<tr>
<td>$a b$</td>
<td>0 3 8 5 $\infty$</td>
</tr>
<tr>
<td>$a b d$</td>
<td>0 3 7 5 10</td>
</tr>
<tr>
<td>$a b c d$</td>
<td>0 3 7 5 9</td>
</tr>
<tr>
<td>$a b c d e$</td>
<td>0 3 7 5 9</td>
</tr>
</tbody>
</table>
Dijkstra’s Algorithm: Example 1

```
BLACK
List S

<table>
<thead>
<tr>
<th></th>
<th>dist[x]</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>0 3 8</td>
</tr>
<tr>
<td>a b</td>
<td>0 3 8 5</td>
</tr>
<tr>
<td>a b d</td>
<td>0 3 7 5 10</td>
</tr>
<tr>
<td>a b c d</td>
<td>0 3 7 5 9</td>
</tr>
<tr>
<td>a b c d e</td>
<td>0 3 7 5 9</td>
</tr>
</tbody>
</table>
```

Graph:
- Vertices: a, b, c, d, e
- Edges and weights:
  - a to b: 3
  - b to c: 2
  - b to d: 1
  - b to e: 2
  - c to d: 8
  - c to e: 7
  - d to e: 5

Distances:
- Start at a:
  - dist[a] = 0
  - dist[b] = 3
  - dist[c] = 8
  - dist[d] = ∞
  - dist[e] = ∞
Dijkstra’s Algorithm: Example 1

```
<table>
<thead>
<tr>
<th>List S</th>
<th>dist[x]</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>0</td>
</tr>
<tr>
<td>a b</td>
<td>0 3 5</td>
</tr>
<tr>
<td>a b d</td>
<td>0 3 7 10</td>
</tr>
<tr>
<td>a b c d</td>
<td>0 3 7 5 9</td>
</tr>
<tr>
<td>a b c d e</td>
<td>0 3 7 5 9</td>
</tr>
</tbody>
</table>
```

Diagram:
- Nodes: a, b, c, d, e
- Edges and weights:
  - a to b: 3
  - a to c: 8
  - b to d: 2
  - c to d: 7
  - d to e: 5
  - a to d: 2
  - b to c: 1
  - b to e: 2

The diagram illustrates the shortest path algorithm with a focus on the nodes and edges, showing the weights associated with each connection.
Dijkstra’s Algorithm: Example 1

**BLACK**

<table>
<thead>
<tr>
<th>List $S$</th>
<th>$dist[x]$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$a$</td>
<td>0 3 8 ∞ ∞</td>
</tr>
<tr>
<td>$a b$</td>
<td>0 3 8 5 ∞</td>
</tr>
<tr>
<td>$a b d$</td>
<td>0 3 7 5 10</td>
</tr>
<tr>
<td>$a b c d$</td>
<td>0 3 7 5 9</td>
</tr>
<tr>
<td>$a b c d e$</td>
<td>0 3 7 5 9</td>
</tr>
</tbody>
</table>
Dijkstra’s Algorithm: Example 1

![Graph diagram]

**BLACK**

<table>
<thead>
<tr>
<th>List S</th>
<th>dist[x]</th>
</tr>
</thead>
<tbody>
<tr>
<td>a b c d e</td>
<td>0 3 7 5 9</td>
</tr>
<tr>
<td>a b d</td>
<td>0 3 7 5 10</td>
</tr>
<tr>
<td>a b c</td>
<td>0 3 8 ∞ ∞</td>
</tr>
<tr>
<td>a</td>
<td>0 3 8 5 ∞</td>
</tr>
<tr>
<td>a</td>
<td>0 ∞ ∞ ∞ ∞</td>
</tr>
</tbody>
</table>

The graph shows the network with vertices labeled a, b, c, d, and e, and edges with weights indicating the distances between them. The table lists different lists S and the corresponding distances in the dist[x] array.
Why Does Dijkstra’s Algorithm Work?

Let an \textbf{S-path} be a path starting at node \(s\) and ending at node \(x\) with all the intermediate nodes coloured BLACK, i.e., from the list \(S\), except possibly \(x\).

\textbf{Theorem 6.8}: Suppose that all arc weights are nonnegative.

Then these two properties hold at the top of \texttt{while}-loop:

\textbf{P1}: If \(x \in V(G)\), then \(\text{dist}[x]\) is the minimum cost of an \(S\)-path from \(s\) to \(x\).

\textbf{P2}: If \(\text{colour}[w] = \text{BLACK}\) (i.e., \(w \in S\)), then \(\text{dist}[w]\) is the minimum cost of a path from \(s\) to \(w\).

Once a node \(u\) is added to \(S\) and \(\text{dist}[u]\) is updated, \(\text{dist}[u]\) never changes in subsequent steps. After \(S = V\), \(\text{dist}\) holds the goal shortest distances.
Proving Why Dijkstra’s Algorithm Works

The update rule: \( \text{dist}[x] \leftarrow \min \{ \text{dist}[x], \text{dist}[u] + c[u, x] \} \).

\(\text{dist}[x]\) is the length of some path from \(s\) to \(x\) at every step.

- If \(x \in S\), then it is an \(S\)-path.
- Updated \(\text{dist}[v]\) never increases.

To prove P1 and P2: induction on the number of times \(k\) of going through the while-loop \((S_k; S_0 = \{s\}; \text{dist}[s] = 0)\).

- \(k = 0\): P1 and P2 hold as \(\text{dist}[s] = 0\).
- Inductive hypothesis: P1 and P2 hold for \(k \geq 0\); \(S_{k+1} = S_k \cup \{u\}\).
- Inductive steps for P2 and P1:
  - Consider any \(s\)-to-\(w\) \(S_{k+1}\)-path \(\gamma = (s, \ldots, y, u)\) of the weight \(|\gamma|\).
  - If \(w \in S_k\), consider the hypothesis.
  - If \(w \notin S_k\), \(\gamma\) extends some \(s\)-to-\(y\) \(S_k\)-path \(\gamma_1 = (s, \ldots, y)\).
### Proving Why Dijkstra’s Algorithm Works

#### Inductive step for P2:

- For \( w \in S_{k+1} \) and \( w \neq u \), P2 holds by inductive hypothesis.
- For \( w = u \), P2 holds, too, because any \( S_{k+1} \)-path \( \gamma = (s, \ldots, y, u) \) of weight \( |\gamma| \) extends some \( S_k \)-path \( \gamma_1 = (s, \ldots, y) \) of weight \( |\gamma_1| \):
  - By the inductive hypothesis, \( \text{dist}[y] \leq |\gamma_1| \).
  - By the update rule, \( \text{dist}[u] \leq \text{dist}[y] + c(y, u) \).
  - Therefore, \( \text{dist}[u] \leq |\gamma| = |\gamma_1| + c(y, u) \).
Proving Why Dijkstra’s Algorithm Works

**Inductive step for P1:** \( x \in V(G); \gamma - \) any \( s\)-to-\( x \) \( S_{k+1} \)-path; \( S_{k+1} = S_k \cup \{u\} \):

- **\( u \notin \gamma \):** \( \gamma \) is an \( S_k \)-path and \( |\gamma| \leq dist[x] \) by the inductive hypothesis.

- **\( u \in \gamma = (s, \ldots, u, x) \):** by the update rule, \( |\gamma| = |\gamma_1| + c(u, x) \geq dist[x] \).

- **\( u \in \gamma = (s, \ldots, u, \ldots, y, x) \), returning to \( S_k \) after \( u \):** by the update rule,

\[
|\gamma| = |\gamma_1| + c(y, x) \geq |\beta| + c(y, x) \geq dist[y] + c(y, x) \geq dist[x]
\]

where \(|\beta|\) is the min weight of an \( s\)-to-\( y \) \( S_k \)-path.
Dijkstra’s Algorithm: Example 2

for $u \in V(G)$ \( \text{dist}[u] \leftarrow c[A, u] \)

<table>
<thead>
<tr>
<th>Node $u$</th>
<th>$A$</th>
<th>$B$</th>
<th>$C$</th>
<th>$D$</th>
<th>$E$</th>
<th>$F$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$A$</td>
<td>$0$</td>
<td>$7$</td>
<td>$9$</td>
<td>$\infty$</td>
<td>$\infty$</td>
<td>$14$</td>
</tr>
<tr>
<td>$A B$</td>
<td>$0$</td>
<td>$7$</td>
<td>$9$</td>
<td>$22$</td>
<td>$\infty$</td>
<td>$14$</td>
</tr>
<tr>
<td>$A B C$</td>
<td>$0$</td>
<td>$7$</td>
<td>$9$</td>
<td>$20$</td>
<td>$\infty$</td>
<td>$11$</td>
</tr>
<tr>
<td>$A B C F$</td>
<td>$0$</td>
<td>$7$</td>
<td>$9$</td>
<td>$20$</td>
<td>$20$</td>
<td>$11$</td>
</tr>
<tr>
<td>$A B C D F$</td>
<td>$0$</td>
<td>$7$</td>
<td>$9$</td>
<td>$20$</td>
<td>$20$</td>
<td>$11$</td>
</tr>
<tr>
<td>$A B C D E F$</td>
<td>$0$</td>
<td>$7$</td>
<td>$9$</td>
<td>$20$</td>
<td>$20$</td>
<td>$11$</td>
</tr>
</tbody>
</table>
Dijkstra’s Algorithm: Example 2

Node $u$ | $A$ | $B$ | $C$ | $D$ | $E$ | $F$
---|---|---|---|---|---|---
$A$ | 0 | 7 | 9 | $\infty$ | $\infty$ | 14
$A B$ | 0 | 7 | 9 | 22 | $\infty$ | 14
$A B C$ | 0 | 7 | 9 | 20 | $\infty$ | 11
$A B C F$ | 0 | 7 | 9 | 20 | 20 | 11
$A B C D F$ | 0 | 7 | 9 | 20 | 20 | 11
$A B C D E F$ | 0 | 7 | 9 | 20 | 20 | 11

$\text{colour}[A] \leftarrow \text{BLACK}; \text{dist}[A] \leftarrow 0$
Dijksra’s Algorithm: Example 2

while-loop:

WHITE $B, C, D, E, F$: min $\text{dist}[B]$

$\text{colour}[B] \leftarrow \text{BLACK}$

for $x \in V(G)$

$\text{dist}[x] \leftarrow$

min \{ $\text{dist}[x], \text{dist}[B] + c[B, x]$ \}
Dijkstras Algorithm: Example 2

While-loop:

WHITE C, D, E, F: $\min \text{dist}[C]$

colour[C] ← BLACK;

for $x \in V(G)$

$\text{dist}[x] ←$ 

$\min \{ \text{dist}[x], \text{dist}[C] + c[C, x] \}$
Dijkstra’s Algorithm: Example 2

while-loop:

WHITE $D, E, F$: $\min \text{dist}[F]$

$colour[F] \leftarrow \text{BLACK}$;

for $x \in V(G)$

$\text{dist}[x] \leftarrow \min \{\text{dist}[x], \text{dist}[F] + c[F, x]\}$
Dijkstra’s Algorithm: Example 2

while-loop:

WHITE $D, E$: $\min \text{dist}[D]$

$\text{colour}[D] \leftarrow \text{BLACK}$;

for $x \in V(G)$

$\text{dist}[x] \leftarrow \min \{\text{dist}[x], \text{dist}[D] + c[D, x]\}$

<table>
<thead>
<tr>
<th>Node $u$</th>
<th>$A$</th>
<th>$B$</th>
<th>$C$</th>
<th>$D$</th>
<th>$E$</th>
<th>$F$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$A$</td>
<td>0</td>
<td>7</td>
<td>9</td>
<td>$\infty$</td>
<td>$\infty$</td>
<td>14</td>
</tr>
<tr>
<td>$A B$</td>
<td>0</td>
<td>7</td>
<td>9</td>
<td>$\infty$</td>
<td>22</td>
<td>$\infty$</td>
</tr>
<tr>
<td>$A B C$</td>
<td>0</td>
<td>7</td>
<td>9</td>
<td>20</td>
<td>$\infty$</td>
<td>11</td>
</tr>
<tr>
<td>$A B C F$</td>
<td>0</td>
<td>7</td>
<td>9</td>
<td>20</td>
<td>20</td>
<td>11</td>
</tr>
<tr>
<td>$A B C D F$</td>
<td>0</td>
<td>7</td>
<td>9</td>
<td>20</td>
<td>20</td>
<td>11</td>
</tr>
<tr>
<td>$A B C D E F$</td>
<td>0</td>
<td>7</td>
<td>9</td>
<td>20</td>
<td>20</td>
<td>11</td>
</tr>
</tbody>
</table>
**Dijkstra’s Algorithm: Example 2**

**Diagram:**
- Nodes: A, B, C, D, E, F
- Edges and weights: A-B (7), A-F (9), B-C (11), B-E (6), C-D (14), C-F (15), D-E (22), E-F (11)

**Table:**

<table>
<thead>
<tr>
<th>Node u</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>0</td>
<td>7</td>
<td>9</td>
<td>20</td>
<td>20</td>
<td>11</td>
</tr>
<tr>
<td>A B</td>
<td>0</td>
<td>7</td>
<td>9</td>
<td>22</td>
<td>20</td>
<td>11</td>
</tr>
<tr>
<td>A B C</td>
<td>0</td>
<td>7</td>
<td>9</td>
<td>20</td>
<td>20</td>
<td>11</td>
</tr>
<tr>
<td>A B C F</td>
<td>0</td>
<td>7</td>
<td>9</td>
<td>20</td>
<td>20</td>
<td>11</td>
</tr>
<tr>
<td>A B C D F</td>
<td>0</td>
<td>7</td>
<td>9</td>
<td>20</td>
<td>20</td>
<td>11</td>
</tr>
</tbody>
</table>

**Algorithm:**

**while-loop:**

WHITE $E$: min $dist[E]$

$colour[E] \leftarrow$ BLACK;

for $x \in V(G)$

$dist[x] \leftarrow$

min \{ $dist[x]$, $dist[E] + c[E, x]$ \}
Dijkstra’s Algorithm: PFS Version

**Input:** weighted digraph \((G,c)\); source node \(s \in V(G)\); priority queue \(Q\); arrays \(dist[0..n-1]\); \(colour[0..n-1]\)

1. \(\text{for } u \in V(G) \text{ do:} \)
   - \(colour[u] \leftarrow \text{WHITE}\)
   - \(Q.\text{insert}(s, \text{key}_s = 0)\)
2. \(Q.\text{is}_\text{empty}()\)
   - yes: \(\text{return } dist\)
   - no:
     1. \(Q.\text{delete}()\)
     2. \(\text{for each } x \text{ adjacent to } u \text{ do:} \)
        1. \(t \leftarrow \tau + c(u,x)\)
        2. \(\text{if } colour[x] = \text{WHITE} \)
           - yes: \(\text{colour}[x] \leftarrow \text{GREY}\)
           - \(Q.\text{insert}(x, t)\)
           - no: \(Q.\text{decreaseKey}(x, t)\)
        3. \(\text{if } Q.\text{getKey}(x) > t \)
           - yes: \(\text{no}\)
           - no: \(\text{yes}\)
        4. \(\text{if } colour[x] = \text{GREY} \)
           - yes: \(\text{no}\)
           - no: \(\text{yes}\)
**Dijkstra’s Algorithm: PFS Version:**

**Initialisation:**

Priority queue $Q = \{a_{\text{key}=0}\}$

<table>
<thead>
<tr>
<th>$v \in V$</th>
<th>$a$</th>
<th>$b$</th>
<th>$c$</th>
<th>$d$</th>
<th>$e$</th>
<th>$f$</th>
<th>$g$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\text{key}_v$</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$\text{dist}[v]$</td>
<td>$-$</td>
<td>$-$</td>
<td>$-$</td>
<td>$-$</td>
<td>$-$</td>
<td>$-$</td>
<td>$-$</td>
</tr>
</tbody>
</table>
Dijkstra’s Algorithm: PFS Version: 

Steps 1 – 2

\[ u \leftarrow a; \quad t_1 \leftarrow \text{key}_a = 0; \quad x \in \{b, c, d\} \]

\[ x \leftarrow b; \quad t_2 = t_1 + \text{cost}(a, b) = 2; \quad Q = \{a_0, b_2\} \]

\[
\begin{array}{c|cccccccc}
\text{key}_v & a & b & c & d & e & f & g \\
\hline
\text{key}_v & 0 & 2 & & & & & \\
\end{array}
\]
Dijkstra’s Algorithm: PFS Version:

**Step 3**

1. **Initialization:**
   - **u = a; t₁ = keyₓ = 0; x ∈ {b, c, d}**

2. **Update:**
   - **x ← c:**
     - **t₂ = t₁ + cost(a, c) = 3; Q = {a₀, b₂, c₃}**

3. **Table:**
   - **v ∈ V | a | b | c | d | e | f | g**
   - **keyᵥ | 0 | 2 | 3**
   - **dist[v] | − | − | − | − | − | − | − | − | −**
Dijkstra’s Algorithm: PFS Version:

Step 4

\( u = a; \ t_1 = \text{key}_a = 0; \ x \in \{b, c, d\} \)

\( x \leftarrow d: \ t_2 = t_1 + \text{cost}(a, d) = 3; \ Q = \{a_0, b_2, c_3, d_3\} \)

<table>
<thead>
<tr>
<th>( v \in V )</th>
<th>a</th>
<th>b</th>
<th>c</th>
<th>d</th>
<th>e</th>
<th>f</th>
<th>g</th>
</tr>
</thead>
<tbody>
<tr>
<td>key(<em>v</em>)</td>
<td>0</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>dist[(v)]</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
</tr>
</tbody>
</table>
Completing the **while**-loop for \( u = a \)

\[
\text{dist}[a] \leftarrow t_1 = 0; \ Q = \{b_2, c_3, d_3\}
\]

<table>
<thead>
<tr>
<th>( v \in V )</th>
<th>a</th>
<th>b</th>
<th>c</th>
<th>d</th>
<th>e</th>
<th>f</th>
<th>g</th>
</tr>
</thead>
<tbody>
<tr>
<td>key(_v)</td>
<td>0</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>\text{dist}[v]</td>
<td>0</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>
Dijkstra’s Algorithm: PFS Version:

Steps 6 – 7

u ← b; \( t_1 \leftarrow \text{key}_b = 2 \); \( x \in \{c, e\} \)

\( x \leftarrow c: \ t_2 = t_1 + \text{cost}(b, c) = 2 + 4 = 6; \ \text{key}_c = 3 < t_2 = 6 \)

<table>
<thead>
<tr>
<th>( v \in V )</th>
<th>a</th>
<th>b</th>
<th>c</th>
<th>d</th>
<th>e</th>
<th>f</th>
<th>g</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \text{key}_v )</td>
<td>0</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>( \text{dist}[v] )</td>
<td>0</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>
Dijkstra’s Algorithm: PFS Version:

Step 8

\[ u = b; \; t_1 = \text{key}_b = 2; \; x \in \{c, e\} \]

\[ x \leftarrow e: \; t_2 = t_1 + \text{cost}(b, e) = 2 + 3 = 5; \; Q = \{b_2, c_3, d_3, e_5\} \]

<table>
<thead>
<tr>
<th>( v \in V )</th>
<th>( a )</th>
<th>( b )</th>
<th>( c )</th>
<th>( d )</th>
<th>( e )</th>
<th>( f )</th>
<th>( g )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \text{key}_v )</td>
<td>0</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>( \text{dist}[v] )</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Dijkstra’s Algorithm: PFS Version:

Completing the **while**-loop for \( u = b \)

\[
\begin{align*}
\text{dist}[b] & \leftarrow t_1 = 2; \ Q = \{c_3, d_3, e_5\} \\
\text{key}_v & \quad a \quad b \quad c \quad d \quad e \quad f \quad g \\
0 & \quad 2 \quad 3 \quad 3 \quad 5 \\
\text{dist}[v] & \quad 0 \quad 2 \quad - \quad - \quad - \quad - \quad - \\
\end{align*}
\]
Dijkstra’s Algorithm: PFS Version:  

Steps 10 – 11

**u ↷ c; \( t_1 ↷ \text{key}_c = 3 \); \( x \in \{d, e, f\} \)**

\( x ↷ d: \ t_2 = t_1 + \text{cost}(c, d) = 3 + 5 = 8; \ \text{key}_d = 3 < t_2 = 8 \)

\[ \begin{array}{c|cccccccc}
 v \in V & a & b & c & d & e & f & g \\
 \hline
 \text{key}_v & 0 & 2 & 3 & 3 & 5 \\
 \text{dist}[v] & 0 & 2 & - & - & - & - & - & - \\
\end{array} \]
Dijkstra’s Algorithm: PFS Version:

Step 12

\[ u = c; \quad t_1 = \text{key}_c = 3; \quad x \in \{d, e, f\} \]

\[ x \leftarrow e: \quad t_2 = t_1 + \text{cost}(c, d) = 3 + 1 = 4; \quad \text{key}_e = 5 < t_2 = 4; \quad \text{key}_e \leftarrow 4 \]

| \[ v \in V \] | \( a \) | \( b \) | \( c \) | \( d \) | \( e \) | \( f \) | \( g \) |
|---|---|---|---|---|---|---|
| \( \text{key}_v \) | 0 | 2 | 3 | 3 | 4 | 0 | 0 |
| \( \text{dist}[v] \) | 0 | 2 | - | - | - | - | - |
Dijkstra’s Algorithm: PFS Version:

**Step 13**

\[ u = c; \ t_1 = \text{key}_c = 3; \ x \in \{d, e, f\} \]

\[ x \leftarrow f: \ t_2 = t_1 + \text{cost}(c, f) = 3 + 6 = 9; \ Q = \{c_3, d_3, e_4, f_9\} \]

\[
\begin{array}{c|cccccccc}
 v \in V & a & b & c & d & e & f & g \\
\hline
\text{key}_v & 0 & 2 & 3 & 3 & 4 & 9 & \\
\text{dist}[v] & 0 & 2 & - & - & - & - & - \\
\end{array}
\]
Dijkstra’s Algorithm: PFS Version:

**Step 14**

Completing the **while**-loop for $u = c$

$dist[c] \leftarrow t_1 = 3; \ Q = \{d_3, e_4, f_9\}$

$\begin{array}{c|cccccccc} 
    v \in V & a & b & c & d & e & f & g \\
    \text{key}_v & 0 & 2 & 3 & 3 & 4 & 9 & \\
    \text{dist}[v] & 0 & 2 & 3 & - & - & - & - \\
\end{array}$
Dijkstra’s Algorithm: PFS Version:

Steps 15 – 16

\[
\begin{align*}
\text{u} & \leftarrow \text{d}; \ t_1 \leftarrow \text{key}_d = 3; \ x \in \{f\} \\
\text{x} & \leftarrow \text{f}: \ t_2 = t_1 + \text{cost}(d, f) = 3 + 7 = 10; \ \text{key}_f = 9 < t_2 = 10
\end{align*}
\]

<table>
<thead>
<tr>
<th>(v \in V)</th>
<th>a</th>
<th>b</th>
<th>c</th>
<th>d</th>
<th>e</th>
<th>f</th>
<th>g</th>
</tr>
</thead>
<tbody>
<tr>
<td>key(_v)</td>
<td>0</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>4</td>
<td>9</td>
<td></td>
</tr>
<tr>
<td>dist(_v)</td>
<td>0</td>
<td>2</td>
<td>3</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>
Completing the `while`-loop for $u = d$

\[
dist[d] \leftarrow t_1 = 3; ~ Q = \{e_4, f_9\}
\]

<table>
<thead>
<tr>
<th>$v \in V$</th>
<th>$a$</th>
<th>$b$</th>
<th>$c$</th>
<th>$d$</th>
<th>$e$</th>
<th>$f$</th>
<th>$g$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$key_v$</td>
<td>0</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>4</td>
<td>9</td>
<td></td>
</tr>
<tr>
<td>$dist[v]$</td>
<td>0</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>–</td>
<td>–</td>
<td>–</td>
</tr>
</tbody>
</table>

Dijkstra’s Algorithm: PFS Version: 
Step 17
Dijkstra’s Algorithm: PFS Version:

Steps 18 – 19

\[ u \leftarrow e; \ t_1 \leftarrow \text{key}_e = 4; \ x \in \{f\} \]

\[ x \leftarrow f: \ t_2 = t_1 + \text{cost}(e, f) = 4 + 8 = 12; \ \text{key}_f = 9 < t_2 = 12 \]

<table>
<thead>
<tr>
<th>( v \in V )</th>
<th>( a )</th>
<th>( b )</th>
<th>( c )</th>
<th>( d )</th>
<th>( e )</th>
<th>( f )</th>
<th>( g )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \text{key}_v )</td>
<td>0</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>4</td>
<td>9</td>
<td></td>
</tr>
<tr>
<td>( \text{dist}[v] )</td>
<td>0</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>–</td>
<td>–</td>
<td>–</td>
</tr>
</tbody>
</table>
Completing the **while**-loop for **$u = e$**

\[
\text{dist}[e] \leftarrow t_1 = 4; \ Q = \{f, g\}
\]

<table>
<thead>
<tr>
<th>$v \in V$</th>
<th>$a$</th>
<th>$b$</th>
<th>$c$</th>
<th>$d$</th>
<th>$e$</th>
<th>$f$</th>
<th>$g$</th>
</tr>
</thead>
<tbody>
<tr>
<td>key$_v$</td>
<td>0</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>4</td>
<td>9</td>
<td></td>
</tr>
<tr>
<td>dist$[v]$</td>
<td>0</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>4</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Dijkstra’s Algorithm: PFS Version:

Steps 21 – 22

\[
\begin{align*}
\text{key}_v & \quad a & b & c & d & e & f & g \\
0 & 2 & 3 & 3 & 4 & 9 & 18 \\
\text{dist}[v] & 0 & 2 & 3 & 3 & 4 & - & - \\
\end{align*}
\]

\[u \leftarrow f; \quad t_1 \leftarrow \text{key}_f = 9; \quad x \in \{g\}\]

\[x \leftarrow g: \quad t_2 = t_1 + \text{cost}(f, g) = 9 + 9 = 18; \quad Q = \{f_9, g_{18}\}\]
Dijkstra’s Algorithm: PFS Version:

Completing the \texttt{while}-loop for $u = f$

$\text{dist}[f] \leftarrow t_1 = 9$; $Q = \{g_{18}\}$

$v \in V | \begin{array}{cccccccc}
    a & b & c & d & e & f & g \\
    \text{key}_v & 0 & 2 & 3 & 3 & 4 & 9 & 18 \\
    \text{dist}[v] & 0 & 2 & 3 & 3 & 4 & 9 & \_ \\
\end{array}$
Dijkstra’s Algorithm: PFS Version:

Steps 24 – 25

Completing the while-loop for $u = g$

$\text{dist}[g] \leftarrow t_1 = 18$; no adjacent vertices for $g$; empty $Q = \{\}$

<table>
<thead>
<tr>
<th>$v \in V$</th>
<th>$a$</th>
<th>$b$</th>
<th>$c$</th>
<th>$d$</th>
<th>$e$</th>
<th>$f$</th>
<th>$g$</th>
</tr>
</thead>
<tbody>
<tr>
<td>key$_v$</td>
<td>0</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>4</td>
<td>9</td>
<td>18</td>
</tr>
<tr>
<td>$\text{dist}[v]$</td>
<td>0</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>4</td>
<td>9</td>
<td>18</td>
</tr>
</tbody>
</table>
SSSP: Bellman-Ford Algorithm

**algorithm** Bellman-Ford( weighted digraph \((G, c)\); node \(s\) )

array \(dist[n] = \{\infty, \infty, \ldots\}\)

\(dist[s] \leftarrow 0\)

for \(i\) from 0 to \(n - 1\) do

    for \(x \in V(G)\) do

        for \(v \in V(G)\) do

            \(dist[v] \leftarrow \min(dist[v], dist[x] + c(x, v))\)

        end for

    end for

end for

return \(dist\)

end

Time complexity – \(\Theta(n^3)\); unlike the Dijkstra’s algorithm, it handles negative weight arcs (but no negative weight cycles making the SSSP senseless).
SSSP: Bellman-Ford Algorithm (Alternative Form)

**Algorithm** Bellman-Ford( weighted digraph \((G, c)\); node \(s\) )

array \(dist[n] = \{\infty, \infty, \ldots\}\)

\(dist[s] \leftarrow 0\)

for \(i\) from 0 to \(n - 1\) do

for \((x, v) \in E(G)\) do

\(dist[v] \leftarrow \min(dist[v], dist[x] + c(x, v))\)

end for

end for

return \(dist\)

end

Replacing the two nested for-loops by the nodes \(x, v \in V(G)\) with a single for-loop by the arcs \((x, v) \in E(G)\).

Time complexity: \(\Theta(mn)\) using adjacency lists vs. \(\Theta(n^3)\) using an adjacency matrix.
Bellman-Ford Algorithm

Slower than Dijkstra’s algorithm when all arcs are nonnegative.

Basic idea as in Dijkstra’s: to find the single-source shortest paths (SSSP) under progressively relaxing restrictions.

- Dijkstra’s: one node a time based on their current distance estimate.
- Bellman-Ford: all nodes at “level” 0, 1, . . . , n − 1 in turn.
  - Level of a node v – the minimum possible number of arcs in a minimum weight path to that node from the source s.

**Theorem 6.9**

If a graph $G$ contains no negative weight cycles, then after the $i^{th}$ iteration of the outer for-loop, the element $dist[v]$ contains the minimum weight of a path to $v$ for all nodes $v$ with level at most $i$. 
Proving Why Bellman-Ford Algorithm Works

Just as for Dijkstra’s, the update ensures \( dist[v] \) never increases.

Induction by the level \( i \) of the nodes:

- **Base case**: \( i = 0 \); the result is true due to initialisation:
  \[
  dist[s] = 0; \quad dist[v] = \infty; \quad v \in V \setminus s.
  \]

- **Induction hypothesis**: \( dist[v]; \quad v \in V \), are true for \( i - 1 \).

- **Induction step** for a node \( v \) at level \( i \):
  
  - Due to no negative weight cycles, a min-weight \( s \)-to-\( v \) path, \( \gamma \), has \( i \) arcs.
  
  - If \( y \) is the last node before \( v \) and \( \gamma_1 \) the subpath to \( y \), then \( dist[y] \leq |\gamma_1| \) by the induction hypothesis.

  - Thus by the update rule:
    \[
    dist[v] \leq dist[y] + c(y, v) \leq |\gamma_1| + c(y, v) \leq |\gamma|
    \]

    as required at level \( i \).
Illustrating Bellman-Ford Algorithm

<table>
<thead>
<tr>
<th>$i$</th>
<th>$\text{dist}[x]$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$a$</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
</tr>
</tbody>
</table>
Illustrating Bellman-Ford Algorithm

The Bellman-Ford algorithm is a dynamic programming algorithm that computes the shortest paths from a single source node to all other nodes in a weighted graph, even if the graph contains negative weight edges. The algorithm works by iteratively relaxing the edges of the graph until the shortest paths are found.

In the example shown, we have a weighted directed graph with five nodes labeled a, b, c, d, and e. The table on the right side of the image shows the distance matrix for each iteration of the algorithm:

<table>
<thead>
<tr>
<th>i</th>
<th>dist[x]</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>a  b  c  d  e</td>
</tr>
<tr>
<td>0</td>
<td>0  ∞  ∞  ∞  ∞</td>
</tr>
<tr>
<td>1</td>
<td>0  3  -1  ∞  ∞</td>
</tr>
<tr>
<td>2</td>
<td>0  0  -1  3  5</td>
</tr>
<tr>
<td>3</td>
<td>0  0  -1  2  0</td>
</tr>
<tr>
<td>4</td>
<td>0  0  -1  2  -1</td>
</tr>
</tbody>
</table>

The algorithm starts with an initial distance of 0 for the source node and ∞ for all other nodes. It then iteratively updates the distances by relaxing each edge in the graph. The process is repeated until no further updates are possible, indicating that the shortest paths have been found.

The diagram on the left visually represents the graph, with arrows indicating the direction of the edges and weights on the edges. The nodes are labeled with their respective values, and the edges are labeled with their weights.
Illustrating Bellman-Ford Algorithm

\[ \begin{array}{c|cccccc} \hline i & a & b & c & d & e \\ \hline 0 & 0 & \infty & \infty & \infty & \infty \\ 1 & 0 & 3 & -1 & \infty & \infty \\ 2 & 0 & 0 & -1 & 3 & 5 \\ 3 & 0 & 0 & -1 & 2 & 0 \\ 4 & 0 & 0 & -1 & 2 & -1 \\ \hline \end{array} \]
Illustrating Bellman-Ford Algorithm

```
0

3  3
b
2

2

a

1

2

-2

c

-1

4

d

-3

2

-1

e

6

-3

-1
```

<table>
<thead>
<tr>
<th>i</th>
<th>dist[x]</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>a  b  c  d  e</td>
</tr>
<tr>
<td>0</td>
<td>0  ∞  ∞  ∞  ∞</td>
</tr>
<tr>
<td>1</td>
<td>0  3  -1  ∞  ∞</td>
</tr>
<tr>
<td>2</td>
<td>0  0  -1  3  5</td>
</tr>
<tr>
<td>3</td>
<td>0  0  -1  2  0</td>
</tr>
<tr>
<td>4</td>
<td>0  0  -1  2  -1</td>
</tr>
</tbody>
</table>
Illustrating Bellman-Ford Algorithm

\[\begin{array}{|c|c|c|c|c|c|}
\hline
i & a & b & c & d & e \\
\hline
0 & 0 & \infty & \infty & \infty & \infty \\
1 & 0 & 3 & -1 & \infty & \infty \\
2 & 0 & 0 & -1 & 3 & 5 \\
3 & 0 & 0 & -1 & 2 & 0 \\
4 & 0 & 0 & -1 & 2 & -1 \\
\hline
\end{array}\]
### Illustrating Bellman-Ford Algorithm (Alternative Form)

<table>
<thead>
<tr>
<th>Arc ((x, v)):</th>
<th>a,b</th>
<th>a,c</th>
<th>b,a</th>
<th>b,d</th>
<th>c,b</th>
<th>c,d</th>
<th>c,e</th>
<th>d,b</th>
<th>d,c</th>
<th>d,e</th>
</tr>
</thead>
<tbody>
<tr>
<td>(c(x, v)):</td>
<td>3</td>
<td>−1</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>4</td>
<td>6</td>
<td>−2</td>
<td>2</td>
<td>−3</td>
</tr>
</tbody>
</table>

#### Iteration \(i = 0\)

<table>
<thead>
<tr>
<th>(x, v)</th>
<th>Distance (d[v] \leftarrow \min{d[v], d[x] + c(x, v)})</th>
<th>a</th>
<th>b</th>
<th>c</th>
<th>d</th>
<th>e</th>
</tr>
</thead>
<tbody>
<tr>
<td>a, b</td>
<td>(d[b] \leftarrow \min{\infty, 0 + 3}) = 3</td>
<td>0</td>
<td>3</td>
<td>(\infty)</td>
<td>(\infty)</td>
<td>(\infty)</td>
</tr>
<tr>
<td>a, c</td>
<td>(d[c] \leftarrow \min{\infty, 0 - 1}) = −1</td>
<td>0</td>
<td>3</td>
<td>−1</td>
<td>(\infty)</td>
<td>(\infty)</td>
</tr>
<tr>
<td>b, a</td>
<td>(d[a] \leftarrow \min{0, 3 + 2}) = 0</td>
<td>0</td>
<td>3</td>
<td>−1</td>
<td>(\infty)</td>
<td>(\infty)</td>
</tr>
<tr>
<td>b, d</td>
<td>(d[d] \leftarrow \min{\infty, 3 + 2}) = 5</td>
<td>0</td>
<td>3</td>
<td>−1</td>
<td>5</td>
<td>(\infty)</td>
</tr>
<tr>
<td>c, b</td>
<td>(d[b] \leftarrow \min{3, -1 + 1}) = 0</td>
<td>0</td>
<td>0</td>
<td>−1</td>
<td>5</td>
<td>(\infty)</td>
</tr>
<tr>
<td>c, d</td>
<td>(d[d] \leftarrow \min{5, -1 + 4}) = 3</td>
<td>0</td>
<td>0</td>
<td>−1</td>
<td>3</td>
<td>(\infty)</td>
</tr>
<tr>
<td>c, e</td>
<td>(d[e] \leftarrow \min{\infty, -1 + 6}) = 5</td>
<td>0</td>
<td>0</td>
<td>−1</td>
<td>3</td>
<td>5</td>
</tr>
<tr>
<td>d, b</td>
<td>(d[b] \leftarrow \min{0, 3 - 2}) = 0</td>
<td>0</td>
<td>0</td>
<td>−1</td>
<td>3</td>
<td>5</td>
</tr>
<tr>
<td>d, c</td>
<td>(d[c] \leftarrow \min{-1, 3 + 2}) = −1</td>
<td>0</td>
<td>0</td>
<td>−1</td>
<td>3</td>
<td>5</td>
</tr>
<tr>
<td>d, e</td>
<td>(d[e] \leftarrow \min{5, 3 - 3}) = 0</td>
<td>0</td>
<td>0</td>
<td>−1</td>
<td>3</td>
<td>0</td>
</tr>
</tbody>
</table>
### Illustrating Bellman-Ford Algorithm (Alternative Form)

<table>
<thead>
<tr>
<th>Arc ((x,v)):</th>
<th>a,b</th>
<th>a,c</th>
<th>b,a</th>
<th>b,d</th>
<th>c,b</th>
<th>c,d</th>
<th>c,e</th>
<th>d,b</th>
<th>d,c</th>
<th>d,e</th>
</tr>
</thead>
<tbody>
<tr>
<td>(c(x,v)):</td>
<td>3</td>
<td>-1</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>4</td>
<td>6</td>
<td>-2</td>
<td>2</td>
<td>-3</td>
</tr>
</tbody>
</table>

**Iteration \(i = 1\)**

<table>
<thead>
<tr>
<th>(x,v)</th>
<th>Distance (d[v] \leftarrow \min{d[v], d[x] + c(x,v)})</th>
<th>a</th>
<th>b</th>
<th>c</th>
<th>d</th>
<th>e</th>
</tr>
</thead>
<tbody>
<tr>
<td>a, b</td>
<td>(d[b] \leftarrow \min{0, 0 + 3} = 0)</td>
<td>0</td>
<td>0</td>
<td>-1</td>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>a, c</td>
<td>(d[c] \leftarrow \min{-1, 0 - 1} = -1)</td>
<td>0</td>
<td>0</td>
<td>-1</td>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>b, a</td>
<td>(d[a] \leftarrow \min{0, 0 + 2} = 0)</td>
<td>0</td>
<td>0</td>
<td>3</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>b, d</td>
<td>(d[d] \leftarrow \min{3, 0 + 2} = 2)</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>c, b</td>
<td>(d[b] \leftarrow \min{0, -1 + 1} = 0)</td>
<td>0</td>
<td>0</td>
<td>-1</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>c, d</td>
<td>(d[d] \leftarrow \min{2, -1 + 4} = 2)</td>
<td>0</td>
<td>0</td>
<td>-1</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>c, e</td>
<td>(d[e] \leftarrow \min{0, -1 + 6} = 0)</td>
<td>0</td>
<td>0</td>
<td>-1</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>d, b</td>
<td>(d[b] \leftarrow \min{0, 2 - 2} = 0)</td>
<td>0</td>
<td>0</td>
<td>-1</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>d, c</td>
<td>(d[c] \leftarrow \min{-1, 2 + 2} = -1)</td>
<td>0</td>
<td>0</td>
<td>-1</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>d, e</td>
<td>(d[e] \leftarrow \min{0, 2 - 3} = -1)</td>
<td>0</td>
<td>0</td>
<td>-1</td>
<td>2</td>
<td>-1</td>
</tr>
</tbody>
</table>
Illustrating Bellman-Ford Algorithm (Alternative Form)

<table>
<thead>
<tr>
<th>Arc $(x, v)$:</th>
<th>a,b</th>
<th>a,c</th>
<th>b,a</th>
<th>b,d</th>
<th>c,b</th>
<th>c,d</th>
<th>c,e</th>
<th>d,b</th>
<th>d,c</th>
<th>d,e</th>
</tr>
</thead>
<tbody>
<tr>
<td>$c(x, v)$:</td>
<td>3</td>
<td>-1</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>4</td>
<td>6</td>
<td>-2</td>
<td>2</td>
<td>-3</td>
</tr>
</tbody>
</table>

Iteration $i = 2..4$

<table>
<thead>
<tr>
<th>$x, v$</th>
<th>Distance $d[v] \leftarrow \min{d[v], d[x] + c(x, v)}$</th>
<th>a</th>
<th>b</th>
<th>c</th>
<th>d</th>
<th>e</th>
</tr>
</thead>
<tbody>
<tr>
<td>a, b</td>
<td>$d[b] \leftarrow \min{0, 0 + 3} = 0$</td>
<td>0</td>
<td>0</td>
<td>-1</td>
<td>2</td>
<td>-1</td>
</tr>
<tr>
<td>a, c</td>
<td>$d[c] \leftarrow \min{-1, 0 - 1} = -1$</td>
<td>0</td>
<td>0</td>
<td>-1</td>
<td>2</td>
<td>-1</td>
</tr>
<tr>
<td>b, a</td>
<td>$d[a] \leftarrow \min{0, 0 + 2} = 0$</td>
<td>0</td>
<td>0</td>
<td>-1</td>
<td>2</td>
<td>-1</td>
</tr>
<tr>
<td>b, d</td>
<td>$d[d] \leftarrow \min{2, 0 + 2} = 2$</td>
<td>0</td>
<td>0</td>
<td>-1</td>
<td>2</td>
<td>-1</td>
</tr>
<tr>
<td>c, b</td>
<td>$d[b] \leftarrow \min{0, -1 + 1} = 0$</td>
<td>0</td>
<td>0</td>
<td>-1</td>
<td>2</td>
<td>-1</td>
</tr>
<tr>
<td>c, d</td>
<td>$d[d] \leftarrow \min{2, -1 + 4} = 2$</td>
<td>0</td>
<td>0</td>
<td>-1</td>
<td>2</td>
<td>-1</td>
</tr>
<tr>
<td>c, e</td>
<td>$d[e] \leftarrow \min{-1, -1 + 6} = -1$</td>
<td>0</td>
<td>0</td>
<td>-1</td>
<td>2</td>
<td>-1</td>
</tr>
<tr>
<td>d, b</td>
<td>$d[b] \leftarrow \min{0, 3 - 2} = 0$</td>
<td>0</td>
<td>0</td>
<td>-1</td>
<td>2</td>
<td>-1</td>
</tr>
<tr>
<td>d, c</td>
<td>$d[c] \leftarrow \min{-1, 3 + 2} = -1$</td>
<td>0</td>
<td>0</td>
<td>-1</td>
<td>2</td>
<td>-1</td>
</tr>
<tr>
<td>d, e</td>
<td>$d[e] \leftarrow \min{-1, 3 - 3} = -1$</td>
<td>0</td>
<td>0</td>
<td>-1</td>
<td>2</td>
<td>-1</td>
</tr>
</tbody>
</table>
Comments on Bellman-Ford Algorithm

- This (non-greedy) algorithm handles negative weight arcs, but not negative weight cycles.

- Running time with the two innermost nested for-loops: \( O(n^3) \).
  - Runs slower than the Dijkstra’s algorithm since considers all nodes at “level” \( i = 0, 1, \ldots, n - 1 \), in turn.

- The alternative form where the two inner-most for-loops are replaced with: \( \text{for } (u, v) \in E(V) \) runs in time \( O(nm) \).
  - The outer for-loop (by \( i \)) in this case can be terminated after no distance changes during the iteration (e.g., after \( i = 2 \) in the example on Slide 39).

- Bellman-Ford algorithm can be modified to detect negative weight cycle (see Textbook, Exercise 6.3.4)
All Pairs Shortest Path (APSP) Problem

Given a weighted digraph \((G, c)\), determine for each pair of nodes \(u, v \in V(G)\) (the length of) a minimum weight path from \(u\) to \(v\).

Convenient output: a distance matrix \(D = [D[u, v]]_{u,v \in V(G)}\)

- Time complexity \(\Theta(nA_{n,m})\) of computing the matrix \(D\) by finding the single-source shortest paths (SSSP) from each node as the source in turn.
  - \(A_{n} = |V(G)|, m = |E(G)|\) – the complexity of the SSSP algorithm.

- The APSP complexity \(\Theta(n^3)\) for the adjacency matrix version of the Dijkstra’s SSSP algorithm: \(A_{n,m} = n^2\).

- The APSP complexity \(\Theta(n^2m)\) for the Bellman-Ford SSSP algorithm: \(A_{n,m} = mn\).
Floyd’s algorithm – one of the known simpler algorithms for computing the distance matrix (three nested for-loops; $\Theta(n^3)$ time complexity):

1. Number all nodes (say, from 0 to $n - 1$).
2. At each step $k$, maintain the matrix of shortest distances from node $i$ to node $j$, not passing through nodes higher than $k$.
3. Update the matrix at each step to see whether the node $k$ shortens the current best distance.

An alternative to running the SSSP algorithm from each node.

- Better than the Dijkstra’s algorithm for dense graphs, probably not for sparse ones.
- Unlike the Dijkstra’s algorithm, can handle negative costs.
- Based on Warshall’s algorithm (just tells whether there is a path from node $i$ to node $j$, not concerned with length).
**Floyd’s Algorithm**

```plaintext
algorithm Floyd( weighted digraph \((G, c)\) )

Initialisation: for \(u, v \in V(G)\) do \(D[u, v] \leftarrow c(u, v)\) end for

for \(x \in V(G)\) do
  for \(u \in V(G)\) do
    for \(v \in V(G)\) do
      \(D[u, v] \leftarrow \min\{D[u, v], D[u, x] + D[x, v]\}\)
    end for
  end for
end for
```

This algorithm is based on **dynamic programming** principles.

At the bottom of the outer for-\(x\)-loop, \(D[u, v]\) for each \(u, v \in V(G)\) is the length of the shortest path from \(u\) to \(v\) passing through intermediate nodes \(x\) having been seen in that loop.
Illustrating Floyd’s Algorithm

Adjacency/cost matrix $c[u, v]$
Illustrating Floyd’s Algorithm: \( x = 0 \)

\[
D_0[u, v] = \min\{\infty, 2c_{[1,0]} - 1c_{[0,1]}\} = 1
\]
Illustrating Floyd’s Algorithm: $x = 1$

Distance matrix $D_1[u, v]$

- $D_1[0, 3] = \min\{\infty, 3D_0[0, 1] + 2D_0[1, 3]\} = 5$
- $D_1[2, 3] = \min\{4, 1D_0[2, 1] + 2D_0[1, 3]\} = 3$
- $D_1[3, 2] = \min\{2, -2D_0[3, 1] + 1D_0[1, 2]\} = -1$
Illustrating Floyd’s Algorithm: $x = 2$

Distance matrix $D_2[u, v]$

$D_2[0, 1] = \min\{3, -1D_{1}[0, 2] + 1D_{1}[2, 1]\} = 0$
$D_2[0, 3] = \min\{5, -1D_{1}[0, 2] + 3D_{1}[2, 3]\} = 2$
$D_2[0, 4] = \min\{\infty, -1D_{1}[0, 2] + 6D_{1}[2, 4]\} = 5$
$D_2[1, 4] = \min\{\infty, 1D_{1}[1, 2] + 6D_{1}[2, 4]\} = 7$
Illustrating Floyd’s Algorithm: $x = 3$

Distance matrix $D_3[u, v]$

- $D_3[0, 4] = \min\{5, 2D_2[0, 3] - 3D_2[3, 4]\} = -1$
- $D_3[1, 4] = \min\{7, 2D_1[1, 3] - 3D_1[3, 4]\} = -1$
- $D_3[2, 4] = \min\{6, 3D_1[2, 3] - 3D_1[3, 4]\} = 0$
Illustrating Floyd’s Algorithm: $x = 4$

**Final distance matrix** $D \equiv D_4[u, v]$
Proving Why Floyd’s Algorithm Works

Theorem 6.12: At the bottom of the outer for-loop, for all nodes \( u \) and \( v \), \( D[u, v] \) contains the minimum length of all paths from \( u \) to \( v \) that are restricted to using only intermediate nodes that have been seen in the outer for-loop.

When algorithm terminates, all nodes have been seen and \( D[u, v] \) is the length of the shortest \( u \)-to-\( v \) path.

Notation: \( S_k \) – the set of nodes seen after \( k \) passes through this loop; \( S_k \)-path – one with all intermediate nodes in \( S_k \); \( D_k \) – the corresponding value of \( D \).

Induction on the outer for-loop:

- **Base case**: \( k = 0 \); \( S_0 = \emptyset \), and the result holds.
- **Induction hypothesis**: It holds after \( k \geq 0 \) times through the loop.
- **Inductive step**: To show that \( D_{k+1}[u, v] \) after \( k + 1 \) passes through this loop is the minimum length of an \( u \)-to-\( v \) \( S_{k+1} \)-path.
Proving Why Floyd’s Algorithm Works

**Inductive step:**
Suppose that $x$ is the last node seen in the loop, so $S_{k+1} = S_k \cup \{x\}$.

- Fix an arbitrary pair of nodes $u, v \in V(G)$ and let $L$ be the min-length of an $u$-to-$v$ $S_{k+1}$-path, so that obviously $L \leq D_{k+1}[u, v]$.

- To show that also $D_{k+1}[u, v] \leq L$, choose an $u$-to-$v$ $S_{k+1}$-path $\gamma$ of length $L$. If $x \notin \gamma$, the result follows from the induction hypothesis.

- If $x \in \gamma$, let $\gamma_1$ and $\gamma_2$ be, respectively, the $u$-to-$x$ and $x$-to-$v$ subpaths. Then $\gamma_1$ and $\gamma_2$ are $S_k$-paths and by the inductive hypothesis,

\[
L \geq |\gamma_1| + |\gamma_2| \geq D_k[u, x] + D_k[x, v] \geq D_{k+1}[u, v]
\]

Non-negativity of the weights is not used in the proof, and Floyd’s algorithm works for negative weights (but negative weight cycles should not be present).
Floyd’s Algorithm: Example 2

Computing all-pairs shortest paths
Floyd’s Algorithm: Example 2

**Initialisation**

\[
[D[u, v]]_{u,v \in V(G)} \leftarrow \begin{bmatrix}
0 & 2 & 3 & 3 & \infty & \infty & \infty \\
2 & 0 & 4 & \infty & 3 & \infty & \infty \\
3 & 4 & 0 & 5 & 1 & 6 & \infty \\
3 & \infty & 5 & 0 & \infty & 7 & \infty \\
\infty & 3 & 1 & \infty & 0 & 8 & \infty \\
\infty & \infty & 6 & 7 & 8 & 0 & 9 \\
\infty & \infty & \infty & \infty & \infty & \infty & 9 & 0 \\
\end{bmatrix}
\]

for \(x \in V = \{a, b, c, d, e, f, g\}\) do

for \(u \in V = \{a, b, c, d, e, f, g\}\) do

for \(v \in V = \{a, b, c, d, e, f, g\}\) do

\[D[u, v] \leftarrow \min \{D[u, v], D[u, x] + D[x, v]\}\]

end for

end for

end for
Floyd’s Algorithm: Example 2

\[ D[u, v] \leftarrow \min \{D[u, v], D[u, a] + D[a, v]\}; \]
\((u, v) \in V^2\)

E.g.,
\[ D[b, d] \leftarrow \min\{D[b, d], D[b, a] + D[a, d]\} \]
\[ = \min\{\infty, 2 + 3\} = 5 \]
Floyd’s Algorithm: Example 2

\[ D[u,v] \leftarrow \min \{D[u,v], D[u,b] + D[b,v]\}; \quad (u,v) \in V^2 \]

E.g.,

\[ D[a,e] \leftarrow \min\{D[a,e], D[a,b] + D[b,e]\} \]
\[ = \min\{\infty, 2 + 3\} = 5 \]
Floyd’s Algorithm: Example 2

E.g.,

\[ D[a, f] \leftarrow \min \{D[a, f], D[a, c] + D[c, f]\} \]

\[ = \min\{\infty, 3 + 6\} = 9 \]
Floyd’s Algorithm: Example 2

\[ D[u, v] \leftarrow \min \{ D[u, v], D[u, d] + D[d, v] \} \quad (u, v) \in V^2 \]

E.g.,

\[ D[a, f] \leftarrow \min \{ D[a, f], D[a, d] + D[d, f] \} \]
\[ = \min \{ 9, 3 + 7 \} = 9 \]
Floyd’s Algorithm: Example 2

\[x \leftarrow e\]

\[
\begin{bmatrix}
    a & b & c & d & e & f & g \\
    a & 0 & 2 & 3 & 3 & 4 & 9 & \infty \\
    b & 2 & 0 & 4 & 5 & 3 & 10 & \infty \\
    c & 3 & 4 & 0 & 5 & 1 & 6 & \infty \\
    d & 3 & 5 & 5 & 0 & 8 & 7 & \infty \\
    e & 4 & 3 & 1 & 8 & 0 & 7 & \infty \\
    f & 9 & 10 & 6 & 7 & 7 & 0 & 9 \\
    g & \infty & \infty & \infty & \infty & \infty & \infty & 9 & 0
\end{bmatrix}
\]

\[D[u, v] \leftarrow \min \{D[u, v], D[u, e] + D[e, v]\}; \quad (u, v) \in V^2\]

E.g.,

\[D[b, f] \leftarrow \min\{D[b, f], D[b, e] + D[e, f]\}\]

\[= \min\{9, 3 + 7\} = 9\]
Floyd’s Algorithm: Example 2

\[
D[u, v] \leftarrow \min \{D[u, v], D[u, f] + D[f, v]\} ;
\]
\[(u, v) \in V^2\]

E.g.,

\[
D[a, g] \leftarrow \min\{D[a, g], D[a, f] + D[f, g]\}
\]
\[= \min\{\infty, 9 + 9\} = 18\]
Computing Actual Shortest Paths

• In addition to knowing the shortest distances, the shortest paths are often to be reconstructed.
• The Floyd’s algorithm can be enhanced to compute also the **predecessor matrix** \( \Pi = [\pi_{i,j}]_{i,j=1,1}^{n,n} \) where vertex \( \pi_{i,j} \) precedes vertex \( j \) on a shortest path from vertex \( i \); \( 1 \leq i, j \leq n \).

Compute a sequence \( \Pi^{(0)}, \Pi^{(1)}, \ldots, \Pi^{(n)} \),

where vertex \( \pi_{i,j}^{(k)} \) precedes the vertex \( j \) on a shortest path from vertex \( i \) with all intermediate vertices in \( V_{(k)} = \{1, 2, \ldots, k\} \).

For case of no intermediate vertices:

\[
\pi_{i,j}^{(0)} = \begin{cases} 
\text{NIL} & \text{if } i = j \text{ or } c[i,j] = \infty \\
 i & \text{if } i \neq j \text{ and } c[i,j] < \infty 
\end{cases}
\]
Computing Actual Shortest Paths

- In addition to knowing the shortest distances, the shortest paths are often to be reconstructed.
- The Floyd’s algorithm can be enhanced to compute also the **predecessor matrix** $\Pi = [\pi_{ij}]_{i,j=1}^{n}$ where vertex $\pi_{i,j}$ precedes vertex $j$ on a shortest path from vertex $i$; $1 \leq i, j \leq n$.

Compute a sequence $\Pi^{(0)}, \Pi^{(1)}, \ldots \Pi^{(n)}$, where vertex $\pi_{i,j}^{(k)}$ precedes the vertex $j$ on a shortest path from vertex $i$ with all intermediate vertices in $V_{(k)} = \{1, 2, \ldots, k\}$.

For case of no intermediate vertices:

$$\pi_{i,j}^{(0)} = \begin{cases} 
\text{NIL} & \text{if } i = j \text{ or } c[i, j] = \infty \\
i & \text{if } i \neq j \text{ and } c[i, j] < \infty 
\end{cases}$$
Floyd’s Algorithm with Predecessors

**Algorithm** \( \text{FloydPred}(\text{weighted digraph } (G, c)) \)

\[
D \leftarrow c \\
\Pi \leftarrow \Pi^{(0)}
\]

Create initial distance matrix from weights.

Initialize predecessors from \( c \) as in Slide 60.

\[
\text{for } k \text{ from } 1 \text{ to } n \text{ do} \\
\quad \text{for } i \text{ from } 1 \text{ to } n \text{ do} \\
\quad \quad \text{for } j \text{ from } 1 \text{ to } n \text{ do} \\
\quad \quad \quad \text{if } D[i, j] > D[i, k] + D[k, j] \text{ then} \\
\quad \quad \quad \quad D[i, j] \leftarrow D[i, k] + D[k, j]; \quad \Pi[i, j] \leftarrow \Pi[k, j] \\
\quad \quad \quad \text{end if} \\
\quad \quad \text{end for} \\
\quad \text{end for} \\
\text{end for}
\]
Illustrating Floyd’s Algorithm with Predecessors

\[
D^{(0)} = \begin{bmatrix}
1 & 2 & 3 & 4 & 5 \\
1 & 0 & 3 & 8 & \infty \\
2 & \infty & 0 & \infty & 1 \\
3 & \infty & 4 & 0 & \infty \\
4 & 2 & \infty & -5 & 0 \\
5 & \infty & \infty & \infty & 6 & 0
\end{bmatrix}
\]

\[
\Pi^{(0)} = \begin{bmatrix}
1 & 2 & 3 & 4 & 5 \\
1 & \text{NIL} & 1 & 1 & \text{NIL} \\
2 & \text{NIL} & \text{NIL} & \text{NIL} & 2 \\
3 & \text{NIL} & 3 & \text{NIL} & \text{NIL} \\
4 & 4 & \text{NIL} & 4 & \text{NIL} \\
5 & \text{NIL} & \text{NIL} & \text{NIL} & 5
\end{bmatrix}
\]
Illustrating Floyd’s Algorithm with Predecessors: $k = 1$

\[
D^{(1)} = \begin{bmatrix}
1 & 2 & 3 & 4 & 5 \\
1 & 0 & 3 & 8 & \infty & -4 \\
2 & \infty & 0 & \infty & 1 & 7 \\
3 & \infty & 4 & 0 & \infty & \infty \\
4 & 2 & 5 & -5 & 0 & -2 \\
5 & \infty & \infty & \infty & 6 & 0 \\
\end{bmatrix}
\]

\[
\Pi^{(1)} = \begin{bmatrix}
1 & 2 & 3 & 4 & 5 \\
1 & \text{NIL} & 1 & 1 & \text{NIL} & 1 \\
2 & \text{NIL} & \text{NIL} & \text{NIL} & 2 & 2 \\
3 & \text{NIL} & 3 & \text{NIL} & \text{NIL} & \text{NIL} \\
4 & 4 & 1 & 4 & \text{NIL} & 1 \\
5 & \text{NIL} & \text{NIL} & \text{NIL} & 5 & \text{NIL} \\
\end{bmatrix}
\]
Illustrating Floyd’s Algorithm with Predecessors: $k = 2$

![Graph Illustration](image)

$D^{(2)}$:

\[
D^{(2)} = \begin{bmatrix}
1 & 2 & 3 & 4 & 5 \\
1 & 0 & 3 & 8 & 4 & -4 \\
2 & \infty & 0 & \infty & 1 & 7 \\
3 & \infty & 4 & 0 & 5 & 11 \\
4 & 2 & 5 & -5 & 0 & -2 \\
5 & \infty & \infty & \infty & 6 & 0
\end{bmatrix}
\]

$\Pi^{(2)}$:

\[
\Pi^{(2)} = \begin{bmatrix}
1 & 2 & 3 & 4 & 5 \\
1 & NIL & 1 & 1 & 2 & 1 \\
2 & NIL & NIL & NIL & 2 & 2 \\
3 & NIL & 3 & NIL & 2 & 2 \\
4 & 4 & 1 & 4 & NIL & 1 \\
5 & NIL & NIL & NIL & 5 & NIL
\end{bmatrix}
\]
Illustrating Floyd’s Algorithm with Predecessors: $k = 3$

D$(3) = $

\begin{bmatrix}
1 & 2 & 3 & 4 & 5 \\
1 & 0 & 3 & 8 & 4 & -4 \\
2 & \infty & 0 & \infty & 1 & 7 \\
3 & \infty & 4 & 0 & 5 & 11 \\
4 & 2 & -1 & -5 & 0 & -2 \\
5 & \infty & \infty & \infty & 6 & 0 \\
\end{bmatrix}

\Pi^{(3)} =

\begin{bmatrix}
1 & 2 & 3 & 4 & 5 \\
1 & NIL & 1 & 1 & 2 & 1 \\
2 & NIL & NIL & NIL & 2 & 2 \\
3 & NIL & 3 & NIL & 2 & 2 \\
4 & 4 & 3 & 4 & NIL & 1 \\
5 & NIL & NIL & NIL & 5 & NIL \\
\end{bmatrix}
Illustrating Floyd’s Algorithm with Predecessors: $k = 4$

$D^{(4)} = \begin{bmatrix}
1 & 2 & 3 & 4 & 5 \\
1 & 0 & 3 & -1 & 4 & -4 \\
2 & 3 & 0 & -4 & 1 & -1 \\
3 & 7 & 4 & 0 & 5 & 3 \\
4 & 2 & -1 & -5 & 0 & -2 \\
5 & 8 & 5 & 1 & 6 & 0
\end{bmatrix}$

$\Pi^{(4)} = \begin{bmatrix}
1 & 2 & 3 & 4 & 5 \\
1 & NIL & 1 & 4 & 2 & 1 \\
2 & 4 & NIL & 4 & 2 & 1 \\
3 & 4 & 3 & NIL & 2 & 1 \\
4 & 4 & 3 & 4 & NIL & 1 \\
5 & 4 & 3 & 4 & 5 & NIL
\end{bmatrix}$
Illustrating Floyd’s Algorithm with Predecessors: \( k = 5 \)

\[
D^{(5)} = \begin{bmatrix}
1 & 2 & 3 & 4 & 5 \\
1 & 0 & -3 & 2 & -4 \\
2 & 3 & 0 & -4 & 1 & -1 \\
3 & 7 & 4 & 0 & 5 & 3 \\
4 & 2 & -1 & -5 & 0 & -2 \\
5 & 8 & 5 & 1 & 6 & 0
\end{bmatrix}
\]

\[
\Pi^{(5)} = \begin{bmatrix}
1 & 2 & 3 & 4 & 5 \\
1 & NIL & 3 & 4 & 5 & 1 \\
2 & 4 & NIL & 4 & 2 & 1 \\
3 & 4 & 3 & NIL & 2 & 1 \\
4 & 4 & 3 & 4 & NIL & 1 \\
5 & 4 & 3 & 4 & 5 & NIL
\end{bmatrix}
\]
Getting Shortest Paths from $\Pi$ Matrix

The recursive algorithm using the predecessor matrix $\Pi = \Pi^{(n)}$ to print the shortest path between vertices $i$ and $j$:

**algorithm** PrintPath($\Pi$, $i$, $j$)

if $i = j$ then print $i$
else
    if $\pi_{i,j} = \text{NIL}$ then print “no path from $i$ to $j$”
    else
        PrintPath($\Pi$, $i$, $\pi_{i,j}$)
        print $j$
    end if
end if
Illustrating PrintPath Algorithm

\[ \Pi^{(5)} = \begin{bmatrix}
1 & 2 & 3 & 4 & 5 \\
NIL & 3 & 4 & 5 & 1 \\
2 & 4 & NIL & 4 & 2 & 1 \\
3 & 4 & 3 & NIL & 2 & 1 \\
4 & 4 & 3 & 4 & NIL & 1 \\
5 & 4 & 3 & 4 & 5 & NIL
\end{bmatrix} \]

PrintPath(\( \Pi^{(5)}, 5, 3 \))
→ PrintPath(\( \Pi^{(5)}, 5, \pi_{5,3} = 4 \))
→ PrintPath(\( \Pi^{(5)}, 5, \pi_{5,4} = 5 \))
   print 5
   print 4
   print 3

PrintPath(\( \Pi^{(5)}, 1, 2 \))
→ PrintPath(\( \Pi^{(5)}, 1, \pi_{1,2} = 3 \))
→ PrintPath(\( \Pi^{(5)}, 1, \pi_{1,3} = 4 \))
→ PrintPath(\( \Pi^{(5)}, 1, \pi_{1,4} = 5 \))
→ PrintPath(\( \Pi^{(5)}, 1, \pi_{1,5} = 1 \))
   print 1
   print 5
   print 4
   print 3
   print 2