COMP2521 24T1
Graphs (VI)
Dijkstra’s Algorithm

Kevin Luxa
cs2521@cse.unsw.edu.au

shortest path
dijkstra’s algorithm
In a weighted graph...

A path is a sequence of edges connected end-to-end

\((v_0, v_1, w_1), (v_1, v_2, w_2), \ldots, (v_{m-1}, v_m, w_m)\)

The \textbf{cost} of a path is the sum of edge weights along the path

The \textbf{shortest path} between two vertices \textit{s} and \textit{t} is the path from \textit{s} to \textit{t} with minimum cost
Variations on shortest path problem:

- **Source-target** shortest path
  - Shortest path from source vertex \( s \) to target vertex \( t \)

- **Single-source** shortest path
  - Shortest path from source vertex \( s \) to all other vertices

- **All-pairs** shortest path
  - Shortest path between all pairs of source and target vertices
In a weighted graph, a path with more edges may be “shorter” than a path with fewer edges.
Dijkstra’s Algorithm

Invented by Dutch computer scientist Edsger W. Dijkstra in 1956
Dijkstra’s algorithm
is used to find the shortest path
in a weighted graph with non-negative weights
Data structures used in Dijkstra’s algorithm:

- **Distance array** \((\text{dist})\)
  - To keep track of shortest currently known distance to each vertex

- **Predecessor array** \((\text{pred})\)
  - Same purpose as in BFS/DFS
  - To keep track of the predecessor of each vertex on the shortest currently known path to that vertex
  - Used to construct the shortest path

- **Set of vertices**
  - Stores unexplored vertices
Algorithm

Edge relaxation

Pseudocode

Example

Path Finding

Vertex Set

Analysis

Other

Algorithms

Appendix

1. Create and initialise data structures
   - Create distance array, initialised to infinity
     - In C, can use INT_MAX (from <limits.h>)
   - Create predecessor array, initialised to -1
   - Initialise set of vertices to contain all vertices

2. Set distance of source vertex (s) to 0

3. While set of vertices is not empty:
   1. Remove vertex from vertex set with smallest distance in distance array
      - Let this vertex be \( v \)
   2. **Explore** \( v \) - that is, for each edge \( v \rightarrow w \):
      - Check if using this edge gives a shorter path to \( w \)
      - If so, update \( w \)'s distance and predecessor - this is called **edge relaxation**
During Dijkstra’s algorithm, the dist and pred arrays:
  • contain data about the shortest path discovered so far
  • need to be updated if a shorter path to some vertex is found
    • this is done via *edge relaxation*
Suppose we are considering edge \((v, w, \text{weight})\).
Suppose we are considering edge \((v, w, \text{weight})\).

We have the following data:

- \(\text{dist}[v]\) - length of shortest known path from \(s\) to \(v\)
- \(\text{dist}[w]\) - length of shortest known path from \(s\) to \(w\) (which may be \(\infty\))
Suppose we are considering edge \((v, w, \text{weight})\).

We have the following data:

- \(\text{dist}[v]\) - length of shortest known path from \(s\) to \(v\)
- \(\text{dist}[w]\) - length of shortest known path from \(s\) to \(w\) (which may be \(\infty\))

In edge relaxation, we take the shortest known path from \(s\) to \(v\) and extend it using edge \((v, w, \text{weight})\) to create a new path from \(s\) to \(w\).
Now we have two paths from $s$ to $w$:

- Shortest known path
- New path via $v$

If the new path is shorter, then we update $\text{dist}[w]$ and $\text{pred}[w]$.

```python
if \text{dist}[v] + \text{weight} < \text{dist}[w]:
    \text{dist}[w] = \text{dist}[v] + \text{weight}
    \text{pred}[w] = v
```
Before relaxation along \((u, w, 7)\)

- **Before:**
  - \(s\) to \(u\): \(\text{dist}[u] = 5\)
  - \(s\) to \(w\): \(\text{dist}[w] = \infty\)

- **After Relaxation:**
  - \(s\) to \(u\): \(\text{dist}[u] = 5\)
  - \(s\) to \(w\): \(\text{dist}[w] = 12\)

**Table Representation:**

<table>
<thead>
<tr>
<th></th>
<th>(s)</th>
<th>(u)</th>
<th>(w)</th>
<th>(w)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>dist</strong></td>
<td>... 5</td>
<td>...</td>
<td>(\infty)</td>
<td>...</td>
</tr>
<tr>
<td><strong>pred</strong></td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>-1</td>
</tr>
</tbody>
</table>
Before relaxation along \((u, w, 7)\)

- **Before** relaxation:
  - Distances: \(\text{dist}[u] = 5\), \(\text{dist}[w] = \infty\)
  - Previous vertex: \(\text{pred}[u] = -1\)

After relaxation along \((u, w, 7)\)

- **After** relaxation:
  - Distances: \(\text{dist}[u] = 5\), \(\text{dist}[w] = 12\)
  - Previous vertex: \(\text{pred}[w] = u\)
Before relaxation along \((v, w, 3)\)

\[
\begin{align*}
\text{dist}[v] &= 8 \\
\text{dist}[w] &= 12
\end{align*}
\]

\[
\begin{array}{c|c|c|c}
\text{dist} & [u] & [v] & [w] \\
\hline
\text{pred} & \ldots & \ldots & \ldots & u
\end{array}
\]
Before relaxation along \((v, w, 3)\)

\[
\begin{align*}
\text{dist}[w] &= 12 \\
\text{dist}[v] &= 8 \\
3 \\
\end{align*}
\]

\[
\begin{array}{c|c|c|c}
\text{dist} & [u] & [v] & [w] \\
\hline
\text{pred} & \ldots & \ldots & u \\
\end{array}
\]

After relaxation along \((v, w, 3)\)

\[
\begin{align*}
\text{dist}[w] &= 11 \\
\text{dist}[v] &= 8 \\
3 \\
\end{align*}
\]

\[
\begin{array}{c|c|c|c}
\text{dist} & [u] & [v] & [w] \\
\hline
\text{pred} & \ldots & \ldots & v \\
\end{array}
\]
dijkstraSSSP($G, \text{src}$):

\textbf{Input:} graph $G$, source vertex $\text{src}$

create dist array, initialised to $\infty$
create pred array, initialised to $-1$
create vSet containing all vertices of $G$

$\text{dist}[\text{src}] = 0$

\textbf{while} vSet is not empty:

\hspace{1em} find vertex $v$ in vSet such that dist[$v$] is minimal
\hspace{1em} remove $v$ from vSet
\hspace{1em} \textbf{for each} edge $(v, w, \text{weight})$ in $G$:
\hspace{2em} relax along $(v, w \text{ weight})$
Dijkstra’s algorithm starting at 0

The diagram shows a graph with vertices connected by edges and weights on the edges. The algorithm starts at vertex 0 and finds the shortest path to other vertices.
Initialisation

while vSet is not empty:
    find vertex \( v \) in vSet such that \( \text{dist}[v] \) is minimal
    and remove it from vSet

for each edge \((v, w, \text{weight})\) in \( G \):
    relax along \((v, w, \text{weight})\)
After first iteration \((v = 0)\)

\[
\begin{align*}
\text{while } & \text{vSet is not empty:} \\
& \text{find vertex } v \text{ in vSet such that } \text{dist}[v] \text{ is minimal} \\
& \text{and remove it from vSet} \\
\text{for each } & \text{edge } (v, w, \text{weight}) \text{ in } G: \\
& \text{relax along } (v, w, \text{weight})
\end{align*}
\]

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>dist</td>
<td>0</td>
<td>14</td>
<td>9</td>
<td>7</td>
<td>(\infty)</td>
<td>(\infty)</td>
</tr>
<tr>
<td>pred</td>
<td>-1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>-1</td>
<td>-1</td>
</tr>
</tbody>
</table>
After second iteration \((v = 3)\)

```
while vSet is not empty:
    find vertex \(v\) in vSet such that \(dist[v]\) is minimal
    and remove it from vSet

for each edge \((v, w, weight)\) in \(G\):
    relax along \((v, w, weight)\)
```

<table>
<thead>
<tr>
<th></th>
<th>[0]</th>
<th>[1]</th>
<th>[2]</th>
<th>[3]</th>
<th>[4]</th>
<th>[5]</th>
</tr>
</thead>
<tbody>
<tr>
<td>dist</td>
<td>0</td>
<td>14</td>
<td>9</td>
<td>7</td>
<td>∞</td>
<td>22</td>
</tr>
<tr>
<td>pred</td>
<td>-1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>-1</td>
<td>3</td>
</tr>
</tbody>
</table>
After third iteration ($v = 2$)

```plaintext
while vSet is not empty:
    find vertex $v$ in vSet such that
    dist[$v$] is minimal
    and remove it from vSet

for each edge $(v, w, weight)$ in $G$:
    relax along $(v, w, weight)$
```

```
<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>dist</td>
<td>0</td>
<td>13</td>
<td>9</td>
<td>7</td>
<td>∞</td>
<td>12</td>
</tr>
<tr>
<td>pred</td>
<td>-1</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>-1</td>
<td>2</td>
</tr>
</tbody>
</table>
```
After fourth iteration \( (v = 5) \)

while vSet is not empty:
  find vertex \( v \) in vSet such that dist\[v\] is minimal
  and remove it from vSet

for each edge \( (v, w, \text{weight}) \) in \( G \):
  relax along \( (v, w, \text{weight}) \)

<table>
<thead>
<tr>
<th></th>
<th>[0]</th>
<th>[1]</th>
<th>[2]</th>
<th>[3]</th>
<th>[4]</th>
<th>[5]</th>
</tr>
</thead>
<tbody>
<tr>
<td>dist</td>
<td>0</td>
<td>13</td>
<td>9</td>
<td>7</td>
<td>20</td>
<td>12</td>
</tr>
<tr>
<td>pred</td>
<td>-1</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>5</td>
<td>2</td>
</tr>
</tbody>
</table>
Example

After fifth iteration ($v = 1$)

```
while vSet is not empty:
    find vertex $v$ in vSet such that dist[$v$] is minimal
    and remove it from vSet

for each edge ($v$, $w$, weight) in $G$:
    relax along ($v$, $w$, weight)
```

```
<table>
<thead>
<tr>
<th>v</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>dist</td>
<td>0</td>
<td>13</td>
<td>9</td>
<td>7</td>
<td>18</td>
<td>12</td>
</tr>
<tr>
<td>pred</td>
<td>-1</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>2</td>
</tr>
</tbody>
</table>
```
After sixth iteration \((v = 4)\)

\[\begin{array}{ccccccc}
0 & 13 & 9 & 7 & 18 & 12 \\
\end{array}\]

while vSet is not empty:
find vertex \(v\) in vSet such that dist\([v]\) is minimal
and remove it from vSet

for each edge \((v, w, weight)\) in \(G\):
relax along \((v, w, weight)\)
while vSet is not empty:
   find vertex $v$ in vSet such that $\text{dist}[v]$ is minimal
   and remove it from vSet

for each edge $(v, w, \text{weight})$ in $G$:
   relax along $(v, w, \text{weight})$
The shortest path from the source vertex to any other vertex can be constructed by tracing backwards through the predecessor array (like for BFS).
Example: Shortest path from 0 to 4

0 → 2 → 1 → 4

Graph:

- Vertices: 0, 1, 2, 3, 4, 5
- Edges with weights:
  - (0, 1) with weight 9
  - (1, 2) with weight 4
  - (1, 3) with weight 10
  - (2, 5) with weight 8
  - (2, 4) with weight 14
  - (3, 4) with weight 7

Pred table:

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>pred</td>
<td>-1</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>2</td>
</tr>
</tbody>
</table>
Example: Shortest path from 0 to 4

0 \rightarrow 2 \rightarrow 1 \rightarrow 4

[0] [1] [2] [3] [4] [5]
pred: -1 2 0 0 1 2
Example: Shortest path from 0 to 4

0 —→ 2 —→ 1 —→ 4
Example: Shortest path from 0 to 4

0 → 2 → 1 → 4
Example: Shortest path from 0 to 4

0 → 2 → 1 → 4

Diagram:

```
0 - 2 - 1 - 4
     |    |
     14  4
     |
     9
3 ---- 5
      |
      7
      |
    10
```

Table:

<table>
<thead>
<tr>
<th></th>
<th>[0]</th>
<th>[1]</th>
<th>[2]</th>
<th>[3]</th>
<th>[4]</th>
<th>[5]</th>
</tr>
</thead>
<tbody>
<tr>
<td>pred</td>
<td>-1</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>2</td>
</tr>
</tbody>
</table>
Example: Shortest path from 0 to 4

0 → 2 → 1 → 4
Example: Shortest path from 0 to 4

0 → 2 → 1 → 4
How to find shortest path between two other vertices (neither of which are the source vertex)?

Generally, you will need to rerun Dijkstra’s algorithm from one of these vertices.
The vSet can be implemented in different ways:

1. Visited array
2. Explicit array/list of vertices
3. Priority queue
Visited array implementation:

- Similar to visited array in BFS/DFS
- Array of \( V \) booleans, initialised to false
- After exploring vertex \( v \), set visited\([v]\) to true
- At the start of each iteration, find vertex \( v \) such that visited\([v]\) is false and dist\([v]\) is minimal \(\Rightarrow O(V)\)
Array/list of vertices implementation:

- Store all vertices in an array/linked list
- After exploring vertex $v$, remove $v$ from array/linked list
- At the start of each iteration, find vertex in array/list such that $\text{dist}[v]$ is minimal $\Rightarrow O(V)$
Priority queue implementation:

- A priority queue is an ADT...
  - where each item has a priority
  - with two main operations:
    - **Insert**: insert item with priority
    - **Delete**: remove item with highest priority
- Use priority queue to store vertices, use *distance* to vertex as priority (smaller distance = higher priority)
- A good priority queue implementation has $O(\log n)$ insert and delete

Priority queues will be discussed in Week 9.
Proof by induction.

Aim is to prove that before and after each iteration:

1. For all explored nodes $s$, $\text{dist}[s]$ is shortest distance from source to $s$
2. For all unexplored nodes $t$, $\text{dist}[t]$ is shortest distance from source to $t$
   via explored nodes only

Ultimately, all nodes are explored, so by 1:

- For all nodes $v$, $\text{dist}[v]$ is the shortest distance from source to $v$
Base case:

- Start of first iteration
  - 1 holds, as there are no explored nodes
  - 2 holds, because
    - dist[source] = 0
    - For all other nodes \( t \), \( \text{dist}[t] = \infty \)
Induction step:

- Assume that 1 and 2 hold at the start of an iteration

![Graph Diagram]

- explored
- unexplored

- source node
Induction step:

- Assume that 1 and 2 hold at the start of an iteration
- Let $s$ be an unexplored node with minimum distance
Induction step:

- Assume that 1 and 2 hold at the start of an iteration
- Let $s$ be an unexplored node with minimum distance
- We claim that $\text{dist}[s]$ is the shortest distance from source to $s$
Induction step:

• Assume that 1 and 2 hold at the start of an iteration
• Let \( s \) be an unexplored node with minimum distance
• We claim that dist[\( s \)] is the shortest distance from source to \( s \)
  • If there is a shorter path to \( s \) via explored nodes only, then dist[\( s \)] would have been updated when exploring the predecessor of \( s \) on that path

\[\begin{align*}
&\text{explored} \\
&\text{unexplored}
\end{align*}\]

\( SRC \quad S \quad S \)
Induction step:

• Assume that 1 and 2 hold at the start of an iteration
• Let $s$ be an unexplored node with minimum distance
• We claim that $\text{dist}[s]$ is the shortest distance from source to $s$
  • If there is a shorter path to $s$ via explored nodes only, then $\text{dist}[s]$ would have been updated when exploring the predecessor of $s$ on that path
  • If there is a shorter path to $s$ via an unexplored node $u$, then $\text{dist}[u] < \text{dist}[s]$, which is a contradiction, since $s$ has minimum distance out of all unexplored nodes
Induction step (continued):

- $\text{dist}[s]$ is the shortest distance from source to $s$
Induction step (continued):

- $\text{dist}[s]$ is the shortest distance from source to $s$
- After exploring $s$:
Induction step (continued):

- \( \text{dist}[s] \) is the shortest distance from source to \( s \)
- After exploring \( s \):
  - 1 still holds for \( s \), since \( \text{dist}[s] \) is not updated while exploring \( s \)
  - Same for all other explored nodes

![Graph diagram](image-url)
Induction step (continued):

- $\text{dist}[s]$ is the shortest distance from source to $s$
- After exploring $s$:
  - 1 still holds for $s$, since $\text{dist}[s]$ is not updated while exploring $s$
  - Same for all other explored nodes
  - 2 still holds for all unexplored nodes $t$, since:

```
1. If there is a shorter path to $t$ via $s$ then we would have updated $\text{dist}[t]$ while exploring $s$
2. Otherwise, we would not have updated $\text{dist}[t]$ and it would remain as it is
```
Induction step (continued):

- dist[s] is the shortest distance from source to s

After exploring s:

1. still holds for s, since dist[s] is not updated while exploring s
   - Same for all other explored nodes
2. still holds for all unexplored nodes t, since:
   - If there is a shorter path to t via s then we would have updated dist[t] while exploring s

![Graph diagram](image-url)
Induction step (continued):

- dist\[s\] is the shortest distance from source to s
- After exploring s:
  - 1 still holds for s, since dist\[s\] is not updated while exploring s
    - Same for all other explored nodes
  - 2 still holds for all unexplored nodes t, since:
    - If there is a shorter path to t via s then we would have updated dist\[t\] while exploring s
    - Otherwise, we would not have updated dist\[t\] and it would remain as it is
Analysis:

- Each edge is considered once ⇒ $O(E)$
  - Undirected edges are considered once in each direction
- Outer loop has $V$ iterations
- Every iteration, algorithm must find vertex $v$ in vSet with minimum distance - time complexity depends on vSet implementation
  - Boolean array ⇒ $O(V)$ per iteration
    ⇒ overall cost = $O(E + V^2) = O(V^2)$
  - Array/list of vertices ⇒ $O(V)$ per iteration
    ⇒ overall cost = $O(E + V^2) = O(V^2)$
  - Priority queue ⇒ $O(\log V)$ per iteration
    ⇒ overall cost = $O(E + V \log V)$
Other Shortest Path Algorithms

- Floyd-Warshall Algorithm
  - All-pairs shortest path
  - Works for graphs with negative weights
- Bellman-Ford Algorithm
  - Single-source shortest path
  - Works for graphs with negative weights
  - Can detect negative cycles
https://forms.office.com/r/5c0fb4tvMb
Appendix
Dijkstra’s Algorithm

Example

Initialisation

while vSet is not empty:
    find vertex v in vSet such that dist[v] is minimal
    and remove it from vSet

for each edge (v, w, weight) in G:
    relax along (v, w, weight)
Dijkstra's Algorithm

Example

while vSet is not empty:
    find vertex \(v\) in vSet such that
dist[\(v\)] is minimal
    and remove it from vSet

for each edge \((v, w, \text{weight})\) in \(G\):
    relax along \((v, w, \text{weight})\)

0

1

4

5

2

3

\[
\begin{array}{cccccc}
0 & \infty & \infty & \infty & \infty & \infty & \infty \\
\text{pred} & -1 & -1 & -1 & -1 & -1 & -1 \\
\end{array}
\]
Dijkstra’s Algorithm

Example

while vSet is not empty:
    find vertex \( v \) in vSet such that \( \text{dist}[v] \) is minimal
    and remove it from vSet

for each edge \((v, w, \text{weight})\) in \( G \):
    relax along \((v, w, \text{weight})\)

Explore 0

$$\begin{array}{ccccccc}
\text{dist} & 0 & \infty & \infty & \infty & \infty & \infty \\
\text{pred} & -1 & -1 & -1 & -1 & -1 & -1 \\
\end{array}$$
Dijkstra’s Algorithm

Example

while vSet is not empty:
  find vertex v in vSet such that dist[v] is minimal
  and remove it from vSet

for each edge (v, w, weight) in G:
  relax along (v, w, weight)
Dijkstra’s Algorithm

Example

while vSet is not empty:
  find vertex \( v \) in vSet such that \( \text{dist}[v] \) is minimal
  and remove it from vSet

for each edge \( (v, w, \text{weight}) \) in \( G \):
  relax along \( (v, w, \text{weight}) \)

\[
\begin{array}{l}
\text{dist} \\
\hline
0 & \infty & \infty & \infty & \infty & \infty & \infty \\
\text{pred} \\
-1 & -1 & -1 & -1 & -1 & -1 & -1
\end{array}
\]
Dijkstra’s Algorithm

Example

```
while vSet is not empty:
    find vertex v in vSet such that dist[v] is minimal
    and remove it from vSet

for each edge (v, w, weight) in G:
    relax along (v, w, weight)
```

```
<table>
<thead>
<tr>
<th></th>
<th>[0]</th>
<th>[1]</th>
<th>[2]</th>
<th>[3]</th>
<th>[4]</th>
<th>[5]</th>
</tr>
</thead>
<tbody>
<tr>
<td>dist</td>
<td>0</td>
<td>∞</td>
<td>∞</td>
<td>∞</td>
<td>∞</td>
<td>∞</td>
</tr>
<tr>
<td>pred</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
</tr>
</tbody>
</table>
```
Dijkstra’s Algorithm
Example

while vSet is not empty:
    find vertex \( v \) in vSet such that dist[\( v \)] is minimal
    and remove it from vSet

for each edge (\( v, w \), weight) in \( G \):
    relax along (\( v, w \), weight)

Relax along (0, 1, 14)
\[ \text{dist}[0] + 14 = 14 < \text{dist}[1] \]
while vSet is not empty:
    find vertex $v$ in vSet such that $\text{dist}[v]$ is minimal
    and remove it from vSet

for each edge $(v, w, \text{weight})$ in $G$:
    relax along $(v, w, \text{weight})$

Relax along $(0, 1, 14)$
$\text{dist}[0] + 14 = 14 < \text{dist}[1]$

while vSet is not empty:
    find vertex $v$ in vSet such that $\text{dist}[v]$ is minimal
    and remove it from vSet

for each edge $(v, w, \text{weight})$ in $G$:
    relax along $(v, w, \text{weight})$

Relax along $(0, 1, 14)$
$\text{dist}[0] + 14 = 14 < \text{dist}[1]$

while vSet is not empty:
    find vertex $v$ in vSet such that $\text{dist}[v]$ is minimal
    and remove it from vSet

for each edge $(v, w, \text{weight})$ in $G$:
    relax along $(v, w, \text{weight})$
Dijkstra’s Algorithm

Example

while vSet is not empty:
    find vertex \( v \) in vSet such that \( \text{dist}[v] \) is minimal
    and remove it from vSet

for each edge \((v, w, \text{weight})\) in \( G \):
    relax along \((v, w, \text{weight})\)

\[
\begin{array}{ccccccc}
\text{dist} & 0 & 14 & \infty & \infty & \infty & \infty \\
\text{pred} & -1 & 0 & -1 & -1 & -1 & -1 \\
\end{array}
\]
Dijkstra’s Algorithm

Example

Relax along (0, 2, 9)

\[ \text{dist}[0] + 9 \]

```plaintext
while vSet is not empty:
    find vertex \( v \) in vSet such that
    \( \text{dist}[v] \) is minimal
    and remove it from vSet

for each edge \((v, w, \text{weight})\) in \( G \):
    relax along \((v, w, \text{weight})\)
```

<table>
<thead>
<tr>
<th></th>
<th>[0]</th>
<th>[1]</th>
<th>[2]</th>
<th>[3]</th>
<th>[4]</th>
<th>[5]</th>
</tr>
</thead>
<tbody>
<tr>
<td>dist</td>
<td>0</td>
<td>14</td>
<td>(\infty)</td>
<td>(\infty)</td>
<td>(\infty)</td>
<td>(\infty)</td>
</tr>
<tr>
<td>pred</td>
<td>-1</td>
<td>0</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
</tr>
</tbody>
</table>
Dijkstra’s Algorithm

Example

while vSet is not empty:
    find vertex \( v \) in vSet such that \( \text{dist}[v] \) is minimal
    and remove it from vSet

for each edge \( (v, w, \text{weight}) \) in \( G \):
    relax along \( (v, w, \text{weight}) \)

Relax along \( (0, 2, 9) \)
\[ \text{dist}[0] + 9 = 9 \]
### Dijkstra’s Algorithm

**Example**

Relax along (0, 2, 9)
\[ \text{dist}[0] + 9 = 9 < \text{dist}[2] \]

**Algorithm**

```
while vSet is not empty:
    find vertex \( v \) in vSet such that \( \text{dist}[v] \) is minimal and remove it from vSet

for each edge \((v, w, \text{weight})\) in \( G \):
    relax along \((v, w, \text{weight})\)
```

<table>
<thead>
<tr>
<th></th>
<th>[0]</th>
<th>[1]</th>
<th>[2]</th>
<th>[3]</th>
<th>[4]</th>
<th>[5]</th>
</tr>
</thead>
<tbody>
<tr>
<td>dist</td>
<td>0</td>
<td>14</td>
<td>∞</td>
<td>∞</td>
<td>∞</td>
<td>∞</td>
</tr>
<tr>
<td>pred</td>
<td>-1</td>
<td>0</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
</tr>
</tbody>
</table>
Dijkstra’s Algorithm

Example

while vSet is not empty:
  find vertex v in vSet such that dist[v] is minimal
  and remove it from vSet

for each edge (v, w, weight) in G:
  relax along (v, w, weight)

Relax along (0, 2, 9)
dist[0] + 9 = 9 < dist[2]

0
1
4
1

2
1
5
9
4
5

3
7
10
8

5
15

1

while vSet is not empty:

find vertex v in vSet such that

dist[v] is minimal

and remove it from vSet

for each edge (v, w, weight) in G:

relax along (v, w, weight)

\[
\begin{array}{ccccccc}
\hline 
0 & 14 & 9 & \infty & \infty & \infty & \infty \\
\text{pred} & -1 & 0 & 0 & -1 & -1 & -1 \\
\end{array}
\]
Dijkstra’s Algorithm

Example

```
while vSet is not empty:
    find vertex v in vSet such that dist[v] is minimal and remove it from vSet

for each edge (v, w, weight) in G:
    relax along (v, w, weight)
```

```
<table>
<thead>
<tr>
<th></th>
<th>[0]</th>
<th>[1]</th>
<th>[2]</th>
<th>[3]</th>
<th>[4]</th>
<th>[5]</th>
</tr>
</thead>
<tbody>
<tr>
<td>dist</td>
<td>0</td>
<td>14</td>
<td>9</td>
<td>∞</td>
<td>∞</td>
<td>∞</td>
</tr>
<tr>
<td>pred</td>
<td>-1</td>
<td>0</td>
<td>0</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
</tr>
</tbody>
</table>
```
Dijkstra’s Algorithm

Example

Relax along \((0, 3, 7)\)
\[\text{dist}[0] + 7\]

\[
\text{while } \text{vSet is not empty:}
\]
find vertex \(v\) in vSet such that \(\text{dist}[v]\) is minimal
and remove it from vSet

\[
\text{for each edge } (v, w, \text{weight}) \text{ in } G:
\]
relax along \((v, w, \text{weight})\)

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>dist</td>
<td>0</td>
<td>14</td>
<td>9</td>
<td>∞</td>
<td>∞</td>
<td>∞</td>
</tr>
<tr>
<td>pred</td>
<td>-1</td>
<td>0</td>
<td>0</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
</tr>
</tbody>
</table>
Dijkstra’s Algorithm

Example

while vSet is not empty:
    find vertex \( v \) in vSet such that \( \text{dist}[v] \) is minimal
    and remove it from vSet

for each edge \((v, w, \text{weight})\) in \( G \):
    relax along \((v, w, \text{weight})\)

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>dist</td>
<td>0</td>
<td>14</td>
<td>9</td>
<td>∞</td>
<td>∞</td>
<td>∞</td>
</tr>
<tr>
<td>pred</td>
<td>-1</td>
<td>0</td>
<td>0</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
</tr>
</tbody>
</table>
Dijkstra’s Algorithm
Example

while vSet is not empty:
    find vertex \( v \) in vSet such that \( \text{dist}[v] \) is minimal
    and remove it from vSet

for each edge \((v, w, \text{weight})\) in \( G \):
    relax along \((v, w, \text{weight})\)

Relax along \((0, 3, 7)\)
\(\text{dist}[0] + 7 = 7 < \text{dist}[3]\)
Dijkstra’s Algorithm

Example

while vSet is not empty:
    find vertex \( v \) in vSet such that dist[\( v \)] is minimal
    and remove it from vSet

for each edge \( (v, w, \text{weight}) \) in \( G \):
    relax along \( (v, w, \text{weight}) \)

### Relax along \( (0, 3, 7) \)

\[ \text{dist}[0] + 7 = 7 < \text{dist}[3] \]
while vSet is not empty:
    find vertex v in vSet such that dist[v] is minimal
    and remove it from vSet

for each edge (v, w, weight) in G:
    relax along (v, w, weight)
Remove 3 from vSet

while vSet is not empty:
    find vertex \( v \) in vSet such that \( \text{dist}[v] \) is minimal
    and remove it from vSet

for each edge \( (v, w, \text{weight}) \) in \( G \):
    relax along \( (v, w, \text{weight}) \)

<table>
<thead>
<tr>
<th>[0]</th>
<th>[1]</th>
<th>[2]</th>
<th>[3]</th>
<th>[4]</th>
<th>[5]</th>
</tr>
</thead>
<tbody>
<tr>
<td>dist</td>
<td>0</td>
<td>14</td>
<td>9</td>
<td>7</td>
<td>( \infty )</td>
</tr>
<tr>
<td>pred</td>
<td>-1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>-1</td>
</tr>
</tbody>
</table>
while vSet is not empty:
  find vertex \( v \) in vSet such that \( \text{dist}[v] \) is minimal
  and remove it from vSet

for each edge \((v, w, \text{weight})\) in \( G \):
  relax along \((v, w, \text{weight})\)
Dijkstra’s Algorithm

Example

No need to consider (3, 0, 7)
(0 has already been explored)

\[
\begin{align*}
\text{while } & \text{vSet is not empty:} \\
& \text{find vertex } v \text{ in vSet such that } \text{dist}[v] \text{ is minimal} \\
& \text{and remove it from vSet}
\end{align*}
\]

\[
\text{for each edge } (v, w, \text{weight}) \text{ in } G: \\
\text{relax along } (v, w, \text{weight})
\]

\[
\begin{array}{c|c|c|c|c|c|c}
\hline
\text{dist} & 0 & 14 & 9 & 7 & \infty & \infty \\
\text{pred} & -1 & 0 & 0 & 0 & -1 & -1 \\
\end{array}
\]
Dijkstra’s Algorithm

Example

Relax along (3, 2, 10)

while vSet is not empty:
    find vertex v in vSet such that dist[v] is minimal
    and remove it from vSet

for each edge (v, w, weight) in G:
    relax along (v, w, weight)
Dijkstra’s Algorithm
Example

while vSet is not empty:
    find vertex $v$ in vSet such that $dist[v]$ is minimal
    and remove it from vSet
for each edge $(v, w, weight)$ in $G$:
    relax along $(v, w, weight)$

Relax along $(3, 2, 10)$
$dist[3] + 10$

```
<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>dist</td>
<td>0</td>
<td>14</td>
<td>9</td>
<td>7</td>
<td>∞</td>
<td>∞</td>
</tr>
<tr>
<td>pred</td>
<td>-1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>-1</td>
<td>-1</td>
</tr>
</tbody>
</table>
```

Algorithm
Pseudocode
Example
Path Finding
Vertex Set
Analysis
Other Algorithms
Appendix
Example
Dijkstra’s Algorithm

Example

while vSet is not empty:
    find vertex v in vSet such that dist[v] is minimal
    and remove it from vSet

for each edge (v, w, weight) in G:
    relax along (v, w, weight)

Relax along (3, 2, 10)

\[
\text{dist}[3] + 10 = 17
\]
Dijkstra’s Algorithm

Example

```
while vSet is not empty:
    find vertex v in vSet such that dist[v] is minimal
    and remove it from vSet

for each edge (v, w, weight) in G:
    relax along (v, w, weight)
```

```
<table>
<thead>
<tr>
<th></th>
<th>[0]</th>
<th>[1]</th>
<th>[2]</th>
<th>[3]</th>
<th>[4]</th>
<th>[5]</th>
</tr>
</thead>
<tbody>
<tr>
<td>dist</td>
<td>0</td>
<td>14</td>
<td>9</td>
<td>7</td>
<td>∞</td>
<td>∞</td>
</tr>
<tr>
<td>pred</td>
<td>-1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>-1</td>
<td>-1</td>
</tr>
</tbody>
</table>
```
Dijkstra’s Algorithm

Example

while vSet is not empty:
    find vertex v in vSet such that dist[v] is minimal and remove it from vSet

for each edge (v, w, weight) in G:
    relax along (v, w, weight)

Relax along (3, 2, 10)
**Dijkstra’s Algorithm**

**Example**

Relax along (3, 5, 15)

while vSet is not empty:
    find vertex \( v \) in vSet such that \( \text{dist}[v] \) is minimal
    and remove it from vSet

for each edge \( (v, w, \text{weight}) \) in \( G \):
    relax along \( (v, w, \text{weight}) \)

---

**Algorithm**

**Pseudocode**

**Example**

Path Finding

Vertex Set

Analysis

Other Algorithms

Appendix

Example

---

**At the start:**

- \( \text{dist}[0] = 0 \)
- \( \text{pred}[0] = -1 \)

**The vertex set \( vSet \) is:** \( \{0, 1, 2, 3, 4, 5\} \)

**Initialization:**

- Remove 0 from \( vSet \)
- Explore 0
  - Relax along \( (0, 1, 14) \)
    - \( \text{dist}[0] + 14 = 14 < \text{dist}[1] \)
  - Relax along \( (0, 2, 9) \)
    - \( \text{dist}[0] + 9 = 9 < \text{dist}[2] \)
  - Relax along \( (0, 3, 7) \)
    - \( \text{dist}[0] + 7 = 7 < \text{dist}[3] \)

**Done with exploring 0:**

- Remove 3 from \( vSet \)
- Explore 3
  - No need to consider \( (3, 0, 7) \)
    - \( 0 \) has already been explored
  - Relax along \( (3, 2, 10) \)
    - \( \text{dist}[3] + 10 = 17 \geq \text{dist}[2] \)
  - Relax along \( (3, 5, 15) \)
    - \( \text{dist}[3] + 15 = 22 < \text{dist}[5] \)

**Done with exploring 3:**

- Remove 2 from \( vSet \)
- Explore 2
  - No need to consider \( (2, 0, 9) \)
    - \( 0 \) has already been explored
  - Relax along \( (2, 1, 4) \)
    - \( \text{dist}[2] + 4 = 13 < \text{dist}[1] \)
  - No need to consider \( (2, 3, 10) \)
    - \( 3 \) has already been explored
  - Relax along \( (2, 5, 3) \)
    - \( \text{dist}[2] + 3 = 12 < \text{dist}[5] \)

**Done with exploring 2:**

- Remove 5 from \( vSet \)
- Explore 5
  - No need to consider \( (5, 2, 3) \)
    - \( 2 \) has already been explored
  - No need to consider \( (5, 3, 15) \)
    - \( 3 \) has already been explored
  - Relax along \( (5, 4, 8) \)
    - \( \text{dist}[5] + 8 = 20 < \text{dist}[4] \)

**Done with exploring 5:**

- Remove 1 from \( vSet \)
- Explore 1
  - No need to consider \( (1, 0, 14) \)
    - \( 0 \) has already been explored
  - No need to consider \( (1, 2, 4) \)
    - \( 2 \) has already been explored
  - Relax along \( (1, 4, 5) \)
    - \( \text{dist}[1] + 5 = 18 < \text{dist}[4] \)

**Done with exploring 1:**

- Remove 4 from \( vSet \)
- Explore 4
  - No need to consider \( (4, 1, 5) \)
    - \( 1 \) has already been explored
  - No need to consider \( (4, 5, 8) \)
    - \( 5 \) has already been explored

**Done with exploring 4:**

**Finished**
Dijkstra’s Algorithm

**Example**

Relax along (3, 5, 15)

\[ \text{dist}[3] + 15 \]

**Pseudocode**

while vSet is not empty:
  find vertex \( v \) in vSet such that \( \text{dist}[v] \) is minimal
  and remove it from vSet

for each edge \((v, w, \text{weight})\) in \( G \):
  relax along \((v, w, \text{weight})\)

**Example**

<table>
<thead>
<tr>
<th></th>
<th>[0]</th>
<th>[1]</th>
<th>[2]</th>
<th>[3]</th>
<th>[4]</th>
<th>[5]</th>
</tr>
</thead>
<tbody>
<tr>
<td>dist</td>
<td>0</td>
<td>14</td>
<td>9</td>
<td>7</td>
<td>∞</td>
<td>∞</td>
</tr>
<tr>
<td>pred</td>
<td>-1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>-1</td>
<td>-1</td>
</tr>
</tbody>
</table>
Dijkstra’s Algorithm

Example

Relax along (3, 5, 15)
\[
\text{dist}[3] + 15 = 22
\]

while vSet is not empty:
  find vertex \( v \) in vSet such that \( \text{dist}[v] \) is minimal
  and remove it from vSet

for each edge \((v, w, \text{weight})\) in \( G \):
  relax along \((v, w, \text{weight})\)
**Dijkstra’s Algorithm**

Example

```
while vSet is not empty:
    find vertex v in vSet such that dist[v] is minimal
    and remove it from vSet

for each edge (v, w, weight) in G:
    relax along (v, w, weight)
```

![Graph](image)

Relax along (3, 5, 15)


<table>
<thead>
<tr>
<th></th>
<th>[0]</th>
<th>[1]</th>
<th>[2]</th>
<th>[3]</th>
<th>[4]</th>
<th>[5]</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>dist</strong></td>
<td>0</td>
<td>14</td>
<td>9</td>
<td>7</td>
<td>∞</td>
<td>∞</td>
</tr>
<tr>
<td><strong>pred</strong></td>
<td>-1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>-1</td>
<td>-1</td>
</tr>
</tbody>
</table>
Dijkstra’s Algorithm

Example

\[ \text{Relax along (3, 5, 15)} \]
\[ \text{dist[3] + 15 = 22 < dist[5]} \]

\[ \text{while vSet is not empty:} \]
\[ \text{find vertex } v \text{ in vSet such that dist}[v] \text{ is minimal} \]
\[ \text{and remove it from vSet} \]

\[ \text{for each edge (v, w, weight) in } G: \]
\[ \text{relax along (v, w, weight)} \]

\[
\begin{array}{ccccccc}
\text{dist} & 0 & 14 & 9 & 7 & \infty & 22 \\
\text{pred} & -1 & 0 & 0 & 0 & -1 & 3 \\
\end{array}
\]
Dijkstra’s Algorithm

Done with exploring 3

while vSet is not empty:
    find vertex \( v \) in vSet such that \( \text{dist}[v] \) is minimal
    and remove it from vSet

for each edge \( (v, w, \text{weight}) \) in \( G \):
    relax along \( (v, w, \text{weight}) \)

```
while vSet is not empty:
    find vertex \( v \) in vSet such that \( \text{dist}[v] \) is minimal
    and remove it from vSet

for each edge \( (v, w, \text{weight}) \) in \( G \):
    relax along \( (v, w, \text{weight}) \)
```
Dijkstra’s Algorithm

Remove 2 from vSet

while vSet is not empty:
  find vertex $v$ in vSet such that $\text{dist}[v]$ is minimal
  and remove it from vSet

for each edge $(v, w, \text{weight})$ in $G$:
  relax along $(v, w, \text{weight})$

<table>
<thead>
<tr>
<th></th>
<th>[0]</th>
<th>[1]</th>
<th>[2]</th>
<th>[3]</th>
<th>[4]</th>
<th>[5]</th>
</tr>
</thead>
<tbody>
<tr>
<td>dist</td>
<td>0</td>
<td>14</td>
<td>9</td>
<td>7</td>
<td>$\infty$</td>
<td>22</td>
</tr>
<tr>
<td>pred</td>
<td>-1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>-1</td>
<td>3</td>
</tr>
</tbody>
</table>

Graph:
- Vertices: 0, 1, 2, 3, 4, 5
- Edges:
  - (0, 1, 14)
  - (0, 2, 9)
  - (0, 3, 7)
  - (2, 1, 4)
  - (2, 3, 10)
  - (2, 5, 3)
  - (3, 0, 7)
  - (3, 2, 10)
  - (3, 5, 15)
  - (4, 1, 5)
  - (4, 5, 8)

Initialisation:
- $\text{dist}[0] = 0$
- $\text{dist}[1] = \infty$
- $\text{dist}[2] = \infty$
- $\text{dist}[3] = \infty$
- $\text{dist}[4] = \infty$
- $\text{dist}[5] = \infty$
- $\text{pred}[0] = -1$
- $\text{pred}[1] = -1$
- $\text{pred}[2] = -1$
- $\text{pred}[3] = -1$
- $\text{pred}[4] = -1$
- $\text{pred}[5] = -1$

Remove 0 from vSet (0)
- Explore 0 (0)
- Relax along (0, 1, 14): $\text{dist}[0] + 14 = 14 < \text{dist}[1]$
- Relax along (0, 2, 9): $\text{dist}[0] + 9 = 9 < \text{dist}[2]$
- Relax along (0, 3, 7): $\text{dist}[0] + 7 = 7 < \text{dist}[3]$

Remove 3 from vSet (3)
- Explore 3 (3)
- No need to consider (3, 0, 7) (0 has already been explored)
- Relax along (3, 2, 10): $\text{dist}[3] + 10 = 17 \not< \text{dist}[2]$
- Relax along (3, 5, 15): $\text{dist}[3] + 15 = 22 < \text{dist}[5]$

Remove 2 from vSet (2)
- Explore 2 (2)
- No need to consider (2, 0, 9) (0 has already been explored)
- Relax along (2, 1, 4): $\text{dist}[2] + 4 = 13 < \text{dist}[1]$
- No need to consider (2, 3, 10) (3 has already been explored)
- Relax along (2, 5, 3): $\text{dist}[2] + 3 = 12 < \text{dist}[5]$

Remove 5 from vSet (5)
- Explore 5 (5)
- No need to consider (5, 2, 3) (2 has already been explored)
- No need to consider (5, 3, 15) (3 has already been explored)
- Relax along (5, 4, 8): $\text{dist}[5] + 8 = 20 < \text{dist}[4]$

Remove 1 from vSet (1)
- Explore 1 (1)
- No need to consider (1, 0, 14) (0 has already been explored)
- No need to consider (1, 2, 4) (2 has already been explored)
- Relax along (1, 4, 5): $\text{dist}[1] + 5 = 18 < \text{dist}[4]$

Remove 4 from vSet (4)
- Explore 4 (4)
- No need to consider (4, 1, 5) (1 has already been explored)
- No need to consider (4, 5, 8) (5 has already been explored)

Done.
Dijkstra's Algorithm

Example

Explore 2

while vSet is not empty:
    find vertex v in vSet such that dist[v] is minimal
    and remove it from vSet

for each edge (v, w, weight) in G:
    relax along (v, w, weight)

Algorithm
Pseudocode
Example
Path Finding
Vertex Set
Analysis
Other
Algorithms
Appendix
Example
Dijkstra’s Algorithm
Example

No need to consider (2, 0, 9)
(0 has already been explored)

while vSet is not empty:
  find vertex \( v \) in vSet such that \( \text{dist}[v] \) is minimal
  and remove it from vSet

for each edge \((v, w, \text{weight})\) in \( G \):
  relax along \((v, w, \text{weight})\)

\[
\begin{array}{c|c|c|c|c|c|c}
0 & 14 & 9 & 7 & \infty & 22 \\
\hline
dist & -1 & 0 & 0 & 0 & -1 & 3 \\
\end{array}
\]
Dijkstra’s Algorithm
Example

Relax along (2, 1, 4)

while vSet is not empty:
    find vertex v in vSet such that dist[v] is minimal and remove it from vSet

for each edge (v, w, weight) in G:
    relax along (v, w, weight)
Dijkstra's Algorithm

Example

while vSet is not empty:
  find vertex \( v \) in vSet such that \( \text{dist}[v] \) is minimal
  and remove it from vSet

for each edge \((v, w, \text{weight})\) in \( G \):
  relax along \((v, w, \text{weight})\)

\[
\begin{array}{ccccccc}
\hline
0 & 14 & 9 & 7 & \infty & 22 \\
\end{array}
\]

\[
\begin{array}{ccccccc}
\text{pred} & -1 & 0 & 0 & 0 & -1 & 3 \\
\end{array}
\]
Dijkstra’s Algorithm

Example

while vSet is not empty:
    find vertex \( v \) in vSet such that \( \text{dist}[v] \) is minimal
    and remove it from vSet

for each edge \( (v, w, \text{weight}) \) in \( G \):
    relax along \( (v, w, \text{weight}) \)

Relax along \((2, 1, 4)\)
\[
\text{dist}[2] + 4 = 13
\]
Relax along (2, 1, 4)
\[ \text{dist}[2] + 4 = 13 < \text{dist}[1] \]

\[
\text{while vSet is not empty:} \\
\text{find vertex } v \text{ in vSet such that } \text{dist}[v] \text{ is minimal} \\
\text{and remove it from vSet}
\]

\[
\text{for each edge } (v, w, \text{weight}) \text{ in } G: \\
\text{relax along } (v, w, \text{weight})
\]

\[
\begin{array}{cccccc}
\hline
0 & 14 & 9 & 7 & \infty & 22 \\
\end{array}
\]

\[
\begin{array}{cccccc}
\text{pred} & -1 & 0 & 0 & 0 & -1 & 3 \\
\end{array}
\]
Dijkstra’s Algorithm

Example

```
while vSet is not empty:
    find vertex v in vSet such that dist[v] is minimal
    and remove it from vSet

for each edge (v, w, weight) in G:
    relax along (v, w, weight)
```

```
<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>dist</td>
<td>0</td>
<td>13</td>
<td>9</td>
<td>7</td>
<td>∞</td>
<td>22</td>
</tr>
<tr>
<td>pred</td>
<td>-1</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>-1</td>
<td>3</td>
</tr>
</tbody>
</table>
```
Dijkstra’s Algorithm

Example

No need to consider (2, 3, 10) (3 has already been explored)

while vSet is not empty:
    find vertex v in vSet such that 
    dist[v] is minimal 
    and remove it from vSet

for each edge (v, w, weight) in G:
    relax along (v, w, weight)

<table>
<thead>
<tr>
<th></th>
<th>[0]</th>
<th>[1]</th>
<th>[2]</th>
<th>[3]</th>
<th>[4]</th>
<th>[5]</th>
</tr>
</thead>
<tbody>
<tr>
<td>dist</td>
<td>0</td>
<td>13</td>
<td>9</td>
<td>7</td>
<td>∞</td>
<td>22</td>
</tr>
<tr>
<td>pred</td>
<td>-1</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>-1</td>
<td>3</td>
</tr>
</tbody>
</table>
Dijkstra’s Algorithm

Example

```
while vSet is not empty:
    find vertex v in vSet such that dist[v] is minimal
    and remove it from vSet

for each edge (v, w, weight) in G:
    relax along (v, w, weight)
```

- **Initialisation**
  - Remove 0 from vSet
  - Explore 0
  - Relax along (0, 1, 14)
    - \( \text{dist}[0] + 14 = 14 < \text{dist}[1] \)
    - \( \text{dist}[0] + 14 = 14 < \text{dist}[1] \)
    - \( \text{dist}[0] + 14 = 14 < \text{dist}[1] \)

- **Remove 3 from vSet**
  - Explore 3
    - No need to consider (3, 0, 7) (0 has already been explored)
    - Relax along (3, 2, 10)
      - \( \text{dist}[3] + 10 = 17 \)
        - \( \not< \text{dist}[2] \)
        - \( \not< \text{dist}[2] \)
        - \( \not< \text{dist}[2] \)
    - Relax along (3, 5, 15)
      - \( \text{dist}[3] + 15 = 22 < \text{dist}[5] \)
        - \( \text{dist}[3] + 15 = 22 < \text{dist}[5] \)
        - \( \text{dist}[3] + 15 = 22 < \text{dist}[5] \)

- **Remove 2 from vSet**
  - Explore 2
    - No need to consider (2, 0, 9) (0 has already been explored)
    - Relax along (2, 1, 4)
      - \( \text{dist}[2] + 4 = 13 < \text{dist}[1] \)
        - \( \text{dist}[2] + 4 = 13 < \text{dist}[1] \)
        - \( \text{dist}[2] + 4 = 13 < \text{dist}[1] \)
    - No need to consider (2, 3, 10) (3 has already been explored)
    - Relax along (2, 5, 3)
      - \( \text{dist}[2] + 3 = 12 < \text{dist}[5] \)
        - \( \text{dist}[2] + 3 = 12 < \text{dist}[5] \)
        - \( \text{dist}[2] + 3 = 12 < \text{dist}[5] \)

- **Remove 5 from vSet**
  - Explore 5
    - No need to consider (5, 2, 3) (2 has already been explored)
    - No need to consider (5, 3, 15) (3 has already been explored)
    - Relax along (5, 4, 8)
      - \( \text{dist}[5] + 8 = 20 < \text{dist}[4] \)
        - \( \text{dist}[5] + 8 = 20 < \text{dist}[4] \)
        - \( \text{dist}[5] + 8 = 20 < \text{dist}[4] \)

- **Remove 1 from vSet**
  - Explore 1
    - No need to consider (1, 0, 14) (0 has already been explored)
    - No need to consider (1, 2, 4) (2 has already been explored)
    - Relax along (1, 4, 5)
      - \( \text{dist}[1] + 5 = 18 < \text{dist}[4] \)
        - \( \text{dist}[1] + 5 = 18 < \text{dist}[4] \)
        - \( \text{dist}[1] + 5 = 18 < \text{dist}[4] \)

- **Remove 4 from vSet**
  - Explore 4
    - No need to consider (4, 1, 5) (1 has already been explored)
    - No need to consider (4, 5, 8) (5 has already been explored)

- **Done**

---

**Example**

```
<table>
<thead>
<tr>
<th></th>
<th>[0]</th>
<th>[1]</th>
<th>[2]</th>
<th>[3]</th>
<th>[4]</th>
<th>[5]</th>
</tr>
</thead>
<tbody>
<tr>
<td>dist</td>
<td>0</td>
<td>13</td>
<td>9</td>
<td>7</td>
<td>∞</td>
<td>22</td>
</tr>
<tr>
<td>pred</td>
<td>-1</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>-1</td>
<td>3</td>
</tr>
</tbody>
</table>
```
Dijkstra’s Algorithm

Example

while vSet is not empty:
    find vertex v in vSet such that dist[v] is minimal
    and remove it from vSet

for each edge (v, w, weight) in G:
    relax along (v, w, weight)

Relax along (2, 5, 3)
dist[2] + 3

while vSet is not empty:
    find vertex v in vSet such that dist[v] is minimal
    and remove it from vSet

for each edge (v, w, weight) in G:
    relax along (v, w, weight)

```
0  13  9  7  ∞  22
-1  2  0  0 -1  3
```
Dijkstra's Algorithm

Example

while vSet is not empty:
    find vertex v in vSet such that
    dist[v] is minimal
    and remove it from vSet

for each edge (v, w, weight) in G:
    relax along (v, w, weight)

Relax along (2, 5, 3)

dist[2] + 3 = 12

\[
\begin{array}{c|c|c|c|c|c}
0 & 13 & 9 & 7 & \infty & 22 \\
\hline
\text{dist} & -1 & 2 & 0 & 0 & -1 & 3 \\
\end{array}
\]
Dijkstra’s Algorithm

Example

while vSet is not empty:
    find vertex $v$ in vSet such that $dist[v]$ is minimal
    and remove it from vSet

for each edge $(v, w, \text{weight})$ in $G$:
    relax along $(v, w, \text{weight})$

Relax along $(2, 5, 3)$

Algorithm
Pseudocode
Example
Path Finding
Vertex Set
Analysis
Other Algorithms
Appendix
Example
Dijkstra’s Algorithm

Example

while vSet is not empty:
    find vertex v in vSet such that dist[v] is minimal
    and remove it from vSet

for each edge (v, w, weight) in G:
    relax along (v, w, weight)

Relax along (2, 5, 3)
\[ \text{dist}[2] + 3 = 12 < \text{dist}[5] \]
Dijkstra’s Algorithm

Example

while vSet is not empty:
    find vertex \( v \) in vSet such that \( \text{dist}[v] \) is minimal
    and remove it from vSet

for each edge \((v, w, \text{weight})\) in \( G \):
    relax along \((v, w, \text{weight})\)

```
while vSet is not empty:
    find vertex \( v \) in vSet such that \( \text{dist}[v] \) is minimal
    and remove it from vSet

for each edge \((v, w, \text{weight})\) in \( G \):
    relax along \((v, w, \text{weight})\)
```
Dijkstra’s Algorithm

Example

Remove 5 from vSet

while vSet is not empty:
    find vertex \( v \) in vSet such that \( \text{dist}[v] \) is minimal
    and remove it from vSet

for each edge \((v, w, \text{weight})\) in \( G \):
    relax along \((v, w, \text{weight})\)

```
[0] [1] [2] [3] [4] [5]
dist  0  13  9  7 \(\infty\) 12
pred  -1  2  0  0 -1  2
```
Dijkstra’s Algorithm

Example

Explore 5

while vSet is not empty:
    find vertex v in vSet such that dist[v] is minimal and remove it from vSet

for each edge (v, w, weight) in G:
    relax along (v, w, weight)

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>dist</td>
<td>0</td>
<td>13</td>
<td>9</td>
<td>7</td>
<td>∞</td>
<td>12</td>
</tr>
<tr>
<td>pred</td>
<td>-1</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>-1</td>
<td>2</td>
</tr>
</tbody>
</table>
Dijkstra’s Algorithm
Example

No need to consider (5, 2, 3)
(2 has already been explored)

while vSet is not empty:
    find vertex v in vSet such that
dist[v] is minimal
    and remove it from vSet

for each edge (v, w, weight) in G:
    relax along (v, w, weight)

<table>
<thead>
<tr>
<th></th>
<th>[0]</th>
<th>[1]</th>
<th>[2]</th>
<th>[3]</th>
<th>[4]</th>
<th>[5]</th>
</tr>
</thead>
<tbody>
<tr>
<td>dist</td>
<td>0</td>
<td>13</td>
<td>9</td>
<td>7</td>
<td></td>
<td>∞</td>
</tr>
<tr>
<td>pred</td>
<td>-1</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>-1</td>
<td>2</td>
</tr>
</tbody>
</table>
Dijkstra’s Algorithm

Example

No need to consider (5, 3, 15) (3 has already been explored)

\[
\begin{align*}
\text{while } & \text{vSet is not empty:} \\
& \text{find vertex } v \text{ in vSet such that } \text{dist}[v] \text{ is minimal} \\
& \text{and remove it from vSet} \\
\text{for each } & \text{edge } (v, w, \text{weight}) \text{ in } G: \\
& \text{relax along } (v, w, \text{weight})
\end{align*}
\]

<table>
<thead>
<tr>
<th></th>
<th>[0]</th>
<th>[1]</th>
<th>[2]</th>
<th>[3]</th>
<th>[4]</th>
<th>[5]</th>
</tr>
</thead>
<tbody>
<tr>
<td>dist</td>
<td>0</td>
<td>13</td>
<td>9</td>
<td>7</td>
<td>∞</td>
<td>12</td>
</tr>
<tr>
<td>pred</td>
<td>-1</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>-1</td>
<td>2</td>
</tr>
</tbody>
</table>
Dijkstra’s Algorithm

Example

Relax along (5, 4, 8)

while vSet is not empty:
    find vertex v in vSet such that dist[v] is minimal
    and remove it from vSet

for each edge (v, w, weight) in G:
    relax along (v, w, weight)

\[
\begin{array}{cccccc}
\hline
0 & 13 & 9 & 7 & \infty & 12 \\
\text{pred} & -1 & 2 & 0 & 0 & -1 & 2 \\
\end{array}
\]
Dijkstra’s Algorithm

Example

```
while vSet is not empty:
    find vertex v in vSet such that dist[v] is minimal and remove it from vSet
    for each edge (v, w, weight) in G:
        relax along (v, w, weight)
```

```
<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>dist</td>
<td>0</td>
<td>13</td>
<td>9</td>
<td>7</td>
<td>∞</td>
<td>12</td>
</tr>
<tr>
<td>pred</td>
<td>-1</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>-1</td>
<td>2</td>
</tr>
</tbody>
</table>
```
Relax along (5, 4, 8)
\[ \text{dist}[5] + 8 = 20 \]

\textbf{while} vSet is not empty:
\begin{itemize}
  \item find vertex \textit{v} in vSet such that dist[\textit{v}] is minimal and remove it from vSet
  \item for each edge \((v, w, \text{weight})\) in \(G\):
    \begin{itemize}
      \item relax along \((v, w, \text{weight})\)
    \end{itemize}
\end{itemize}
Dijkstra’s Algorithm

Example

while vSet is not empty:
  find vertex v in vSet such that dist[v] is minimal and remove it from vSet

for each edge (v, w, weight) in G:
  relax along (v, w, weight)

Relax along (5, 4, 8)
Dijkstra’s Algorithm Example

Relax along (5, 4, 8)
\[\text{dist}[5] + 8 = 20 < \text{dist}[4]\]

\[
\begin{array}{ccccccc}
0 & 1 & 2 & 3 & 4 & 5 \\
\hline
\text{dist} & 0 & 13 & 9 & 7 & 20 & 12 \\
\text{pred} & -1 & 2 & 0 & 0 & 5 & 2 \\
\end{array}
\]

while vSet is not empty:
  find vertex \( v \) in vSet such that \( \text{dist}[v] \) is minimal
  and remove it from vSet

for each edge \((v, w, \text{weight})\) in \( G \):
  relax along \((v, w, \text{weight})\)
Dijkstra’s Algorithm

Example

while vSet is not empty:
    find vertex v in vSet such that dist[v] is minimal
    and remove it from vSet

for each edge (v, w, weight) in G:
    relax along (v, w, weight)

0
1
2
3
4
5

14 5 8
4 3 10
7
15

while vSet is not empty:
    find vertex v in vSet such that dist[v] is minimal
    and remove it from vSet

for each edge (v, w, weight) in G:
    relax along (v, w, weight)

<table>
<thead>
<tr>
<th></th>
<th>[0]</th>
<th>[1]</th>
<th>[2]</th>
<th>[3]</th>
<th>[4]</th>
<th>[5]</th>
</tr>
</thead>
<tbody>
<tr>
<td>dist</td>
<td>0</td>
<td>13</td>
<td>9</td>
<td>7</td>
<td>20</td>
<td>12</td>
</tr>
<tr>
<td>pred</td>
<td>-1</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>5</td>
<td>2</td>
</tr>
</tbody>
</table>
Dijkstra’s Algorithm

Remove 1 from vSet

\[\text{while } \text{vSet is not empty:}
\]
\[\text{find vertex } v \text{ in vSet such that dist}[v] \text{ is minimal}
\]
\[\text{and remove it from vSet}
\]

\[\text{for each edge } (v, w, \text{weight}) \text{ in } G:\]
\[\text{relax along } (v, w, \text{weight})\]
Dijkstra’s Algorithm
Example

while vSet is not empty:
    find vertex \( v \) in vSet such that \( \text{dist}[v] \) is minimal
    and remove it from vSet

for each edge \((v, w, \text{weight})\) in \( G \):
    relax along \((v, w, \text{weight})\)

Explore 1

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>dist</td>
<td>0</td>
<td>13</td>
<td>9</td>
<td>7</td>
<td>20</td>
<td>12</td>
</tr>
<tr>
<td>pred</td>
<td>-1</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>5</td>
<td>2</td>
</tr>
</tbody>
</table>
Dijkstra’s Algorithm
Example

No need to consider (1, 0, 14)
(0 has already been explored)

while vSet is not empty:
    find vertex $v$ in vSet such that
    $\text{dist}[v]$ is minimal
    and remove it from vSet

for each edge $(v, w, \text{weight})$ in $G$:
    relax along $(v, w, \text{weight})
Dijkstra’s Algorithm

Example

No need to consider (1, 2, 4) (2 has already been explored)

```
while vSet is not empty:
    find vertex v in vSet such that dist[v] is minimal
    and remove it from vSet

for each edge (v, w, weight) in G:
    relax along (v, w, weight)
```

```
<table>
<thead>
<tr>
<th></th>
<th>[0]</th>
<th>[1]</th>
<th>[2]</th>
<th>[3]</th>
<th>[4]</th>
<th>[5]</th>
</tr>
</thead>
<tbody>
<tr>
<td>dist</td>
<td>0</td>
<td>13</td>
<td>9</td>
<td>7</td>
<td>20</td>
<td>12</td>
</tr>
<tr>
<td>pred</td>
<td>-1</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>5</td>
<td>2</td>
</tr>
</tbody>
</table>
```
Dijkstra’s Algorithm

Example

Relax along (1, 4, 5)

while vSet is not empty:
  find vertex \( v \) in vSet such that dist[\( v \)] is minimal
  and remove it from vSet

for each edge \((v, w, \text{weight})\) in \( G \):
  relax along \((v, w, \text{weight})\)

\[
\begin{array}{cccccc}
\text{dist} & 0 & 13 & 9 & 7 & 20 & 12 \\
\text{pred} & -1 & 2 & 0 & 0 & 5 & 2 \\
\end{array}
\]
Dijkstra’s Algorithm

Example

Relax along \((1, 4, 5)\)
\[\text{dist}[1] + 5\]

while vSet is not empty:
  find vertex \(v\) in vSet such that \(\text{dist}[v]\) is minimal
  and remove it from vSet

for each edge \((v, w, \text{weight})\) in \(G\):
  relax along \((v, w, \text{weight})\)

\[
\begin{array}{ccccccc}
\text{dist} & 0 & 13 & 9 & 7 & 20 & 12 \\
\text{pred} & -1 & 2 & 0 & 0 & 5 & 2 \\
\end{array}
\]
Dijkstra’s Algorithm Example

**Relax along (1, 4, 5)**

\[
\text{dist}[1] + 5 = 18
\]

**Example**

```
while vSet is not empty:
    find vertex \( v \) in vSet such that \( \text{dist}[v] \) is minimal
    and remove it from vSet

for each edge \((v, w, \text{weight})\) in \( G \):
    relax along \((v, w, \text{weight})\)
```

**Table**

<table>
<thead>
<tr>
<th></th>
<th>[0]</th>
<th>[1]</th>
<th>[2]</th>
<th>[3]</th>
<th>[4]</th>
<th>[5]</th>
</tr>
</thead>
<tbody>
<tr>
<td>(\text{dist})</td>
<td>0</td>
<td>13</td>
<td>9</td>
<td>7</td>
<td>20</td>
<td>12</td>
</tr>
<tr>
<td>(\text{pred})</td>
<td>-1</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>5</td>
<td>2</td>
</tr>
</tbody>
</table>
Dijkstra’s Algorithm

Example

Relax along (1, 4, 5)

\[ \text{dist}[1] + 5 = 18 < \text{dist}[4] \]

\[ \text{while vSet is not empty:} \]
\[ \quad \text{find vertex } v \text{ in vSet such that} \]
\[ \quad \quad \text{dist}[v] \text{ is minimal} \]
\[ \quad \quad \text{and remove it from vSet} \]

\[ \text{for each edge } (v, w, \text{weight}) \text{ in } G: \]
\[ \quad \text{relax along } (v, w, \text{weight}) \]
Dijkstra’s Algorithm

Example

Relax along (1, 4, 5)
\( \text{dist}[1] + 5 = 18 < \text{dist}[4] \)

while vSet is not empty:
find vertex \( v \) in vSet such that
\( \text{dist}[v] \) is minimal
and remove it from vSet

for each edge \( (v, w, \text{weight}) \) in \( G \):
relax along \( (v, w, \text{weight}) \)
Dijkstra’s Algorithm

Example

Done with exploring 1

while vSet is not empty:
find vertex \( v \) in vSet such that \( \text{dist}[v] \) is minimal
and remove it from vSet

for each edge \( (v, w, \text{weight}) \) in \( G \):
relax along \( (v, w, \text{weight}) \)

\[
\begin{array}{c|c|c|c|c|c|c}
\text{dist} & 0 & 13 & 9 & 7 & 18 & 12 \\
\text{pred} & -1 & 2 & 0 & 0 & 1 & 2 \\
\end{array}
\]

Algorithm
Pseudocode
Example
Path Finding
Vertex Set
Analysis
Other Algorithms
Appendix
Example
Dijkstra’s Algorithm

Remove 4 from vSet

```
while vSet is not empty:
    find vertex v in vSet such that dist[v] is minimal
    and remove it from vSet

for each edge (v, w, weight) in G:
    relax along (v, w, weight)
```

```
0 13 9 7 18 12
-1 2 0 0 1 2
```
Dijkstra’s Algorithm

```
while vSet is not empty:
  find vertex v in vSet such that dist[v] is minimal and remove it from vSet
  for each edge (v, w, weight) in G:
    relax along (v, w, weight)
```

Example

```
while vSet is not empty:
  find vertex v in vSet such that dist[v] is minimal
  and remove it from vSet
  for each edge (v, w, weight) in G:
    relax along (v, w, weight)
```

```
<table>
<thead>
<tr>
<th></th>
<th>[0]</th>
<th>[1]</th>
<th>[2]</th>
<th>[3]</th>
<th>[4]</th>
<th>[5]</th>
</tr>
</thead>
<tbody>
<tr>
<td>dist</td>
<td>0</td>
<td>13</td>
<td>9</td>
<td>7</td>
<td>18</td>
<td>12</td>
</tr>
<tr>
<td>pred</td>
<td>-1</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>2</td>
</tr>
</tbody>
</table>
```
Dijkstra’s Algorithm

Example

No need to consider (4, 1, 5)
(1 has already been explored)

```
while vSet is not empty:
    find vertex v in vSet such that dist[v] is minimal
    and remove it from vSet

for each edge (v, w, weight) in G:
    relax along (v, w, weight)
```

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>dist</td>
<td>0</td>
<td>13</td>
<td>9</td>
<td>7</td>
<td>18</td>
<td>12</td>
</tr>
<tr>
<td>pred</td>
<td>-1</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>2</td>
</tr>
</tbody>
</table>
Dijkstra's Algorithm

Example

No need to consider (4, 5, 8)
(5 has already been explored)

while vSet is not empty:
  find vertex v in vSet such that
dist[v] is minimal
  and remove it from vSet

for each edge (v, w, weight) in G:
  relax along (v, w, weight)
Dijkstra’s Algorithm

Example

while vSet is not empty:
    find vertex \( v \) in vSet such that \( \text{dist}[v] \) is minimal
    and remove it from vSet

for each edge \((v, w, \text{weight})\) in \( G \):
    relax along \((v, w, \text{weight})\)

Done with exploring 4

```
while vSet is not empty:
    find vertex \( v \) in vSet such that \( \text{dist}[v] \) is minimal
    and remove it from vSet

for each edge \((v, w, \text{weight})\) in \( G \):
    relax along \((v, w, \text{weight})\)
```
Dijkstra’s Algorithm

Example

while vSet is not empty:
    find vertex \( v \) in vSet such that dist[\( v \)] is minimal
    and remove it from vSet

for each edge \((v, w, weight)\) in \( G \):
    relax along \((v, w, weight)\)

\[\begin{array}{ccccccc}
\text{dist} & 0 & 13 & 9 & 7 & 18 & 12 \\
\text{pred} & -1 & 2 & 0 & 0 & 1 & 2 \\
\end{array}\]