Shortest Paths

One common problem is finding the shortest path between two given vertices.

Recall that in a weighted graph, the length of a path is the sum of the weights of each of the edges in that path.

Applications

One application is circuit design: the time it takes for a change in input to affect an output depends on the shortest path.

Given the graph below, suppose we wish to find the shortest path from vertex 1 to vertex 13.
**Shortest Path**

- After some consideration, we may determine that the shortest path is as follows, with length 14.

![Diagram of shortest path](image)

- Other paths exist, but they are longer.

**Negative Cycles**

- Clearly, if we have negative vertices, it may be possible to end up in a cycle whereby each pass through the cycle decreases the total length.
- Thus, a shortest length would be undefined for such a graph.
- Consider the shortest path from vertex 1 to 4...
- We will only consider non-negative weights.

**Shortest Path Example**

- Given:
  - Weighted Directed graph \( G = (V, E) \).
  - Source \( s \), destination \( t \).
  - Find shortest directed path from \( s \) to \( t \).

![Diagram of shortest path example](image)

- Cost of path from 2 to 3 to 5 to 7: \[ 9 + 23 + 2 + 16 = 48 \]

**Discussion Items**

- How many possible paths are there from \( s \) to \( t \)?
- Can we safely ignore cycles? If so, how?
- Any suggestions on how to reduce the set of possibilities?
- Can we determine a lower bound on the complexity like we did for comparison sorting?
**Key Observation**

- A key observation is that if the shortest path contains the node $v$, then:
  - It will only contain $v$ once, as any cycles will only add to the length.
  - The path from $s$ to $v$ must be the shortest path to $v$ from $s$.
  - The path from $v$ to $t$ must be the shortest path to $t$ from $v$.
- Thus, if we can determine the shortest path to all other vertices that are incident to the target vertex we can easily compute the shortest path.
- Implies a set of sub-problems on the graph with the target vertex removed.

**Dijkstra’s Algorithm**

- Works when all of the weights are positive.
- Provides the shortest paths from a source to all other vertices in the graph.
  - Can be terminated early once the shortest path to $t$ is found if desired.

**Shortest Path**

- Consider the following graph with positive weights and cycles.

**Dijkstra’s Algorithm**

- A first attempt at solving this problem might require an array of Boolean values, all initially false, that indicate whether we have found a path from the source.

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>$F$</td>
<td>$F$</td>
<td>$F$</td>
<td>$F$</td>
<td>$F$</td>
<td>$F$</td>
<td>$F$</td>
<td>$F$</td>
<td>$F$</td>
<td>$F$</td>
</tr>
</tbody>
</table>
Dijkstra’s Algorithm

• Graphically, we will denote this with check boxes next to each of the vertices (initially unchecked)

![Graph with check boxes next to vertices]

Dijkstra’s Algorithm

• We will work bottom up.
  – Note that if the starting vertex has any adjacent edges, then there will be one vertex that is the shortest distance from the starting vertex. This is the shortest reachable vertex of the graph.
• We will then try to extend any *existing* paths to new vertices.
• Initially, we will start with the path of length 0 – this is the trivial path from vertex 1 to itself

Dijkstra’s Algorithm

• If we now extend this path, we should consider the paths
  – (1, 2) length 4
  – (1, 4) length 1
  – (1, 5) length 8

The *shortest* path so far is (1, 4) which is of length 1.

Dijkstra’s Algorithm

• Thus, if we now examine vertex 4, we may deduce that there exist the following paths:
  – (1, 4, 5) length 12
  – (1, 4, 7) length 10
  – (1, 4, 8) length 9

![Graph with check boxes next to vertices]
Dijkstra’s Algorithm

• We need to remember that the length of that path from node 1 to node 4 is 1
• Thus, we need to store the length of a path that goes through node 4:
  – 5 of length 12
  – 7 of length 10
  – 8 of length 9

Dijkstra’s Algorithm

• We now know that:
  – There exist paths from vertex 1 to vertices {2, 4, 5, 7, 8}.
  – We know that the shortest path from vertex 1 to vertex 4 is of length 1.
  – We know that the shortest path to the other vertices {2, 5, 7, 8} is at most the length listed in the table to the right.

### Vertex Length

<table>
<thead>
<tr>
<th>Vertex</th>
<th>Length</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>8</td>
</tr>
<tr>
<td>7</td>
<td>10</td>
</tr>
<tr>
<td>8</td>
<td>9</td>
</tr>
</tbody>
</table>

Dijkstra’s Algorithm

• There cannot exist a shorter path to either of the vertices 1 or 4, since the distances can only increase at each iteration.
• We consider these vertices to be visited

Dijkstra’s Algorithm

• We have already discovered that there is a path of length 8 to vertex 5 with the path (1, 5).
• Thus, we can safely ignore this longer path.

If you only knew this information and nothing else about the graph, what is the possible lengths from vertex 1 to vertex 2? What about to vertex 7?

<table>
<thead>
<tr>
<th>Vertex</th>
<th>Length</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>8</td>
</tr>
<tr>
<td>7</td>
<td>10</td>
</tr>
<tr>
<td>8</td>
<td>9</td>
</tr>
</tbody>
</table>
Relaxation

Maintaining this shortest discovered distance \( d[v] \) is called relaxation:

\[
\text{Relax}(u, v, w) \{
\text{if } (d[v] > d[u] + w) \text{ then } \\
\quad d[v] = d[u] + w;
\}
\]

Dijkstra’s Algorithm

- In Dijkstra’s algorithm, we always take the next unvisited vertex which has the current shortest path from the starting vertex in the table.
- This is vertex 2

Dijkstra’s Algorithm

- We can try to update the shortest paths to vertices 3 and 6 (both of length 5) however:
  - there already exists a path of length 8 < 10 to vertex 5 \((10 = 4 + 6)\)
  - we already know the shortest path to 4 is 1

Dijkstra’s Algorithm

- To keep track of those vertices to which no path has reached, we can assign those vertices an initial distance of either
  - infinity \((\infty)\),
  - a number larger than any possible path, or
  - a negative number
- For demonstration purposes, we will use \(\infty\)
Dijkstra’s Algorithm

- As well as finding the length of the shortest path, we’d like to find the corresponding shortest path.
- Each time we update the shortest distance to a particular vertex, we will keep track of the predecessor used to reach this vertex on the shortest path.

We will store a table of pointers, each initially 0.
- This table will be updated each time a distance is updated.

Dijkstra’s Algorithm

Graphically, we will display the reference to the preceding vertex by a red arrow:
- if the distance to a vertex is $\infty$, there will be no preceding vertex
- otherwise, there will be exactly one preceding vertex

Thus, for our initialization:
- we set the current distance to the initial vertex as 0
- for all other vertices, we set the current distance to $\infty$
- all vertices are initially marked as unvisited
- set the previous pointer for all vertices to null
Dijkstra’s Algorithm

• Thus, we iterate:
  – find an unvisited vertex which has the shortest distance to it
  – mark it as visited
  – for each unvisited vertex which is adjacent to the current vertex:
    • add the distance to the current vertex to the weight of the connecting edge
    • if this is less than the current distance to that vertex, update the distance and set the parent vertex of the adjacent vertex to be the current vertex

• Halting condition:
  – we successfully halt when the vertex we are visiting is the target vertex
  – if at some point, all remaining unvisited vertices have distance \( \infty \), then no path from the starting vertex to the end vertex exists

• Note: We do not halt just because we have updated the distance to the end vertex, we have to visit the target vertex.

Example

• Consider the graph:
  – the distances are appropriately initialized
  – all vertices are marked as being unvisited

• Visit vertex 1 and update its neighbours, marking it as visited
  – the shortest paths to 2, 4, and 5 are updated
Example

- The next vertex we visit is vertex 4
  - vertex 5 \(1 + 11 \geq 8\) don’t update
  - vertex 7 \(1 + 9 < \infty\) update
  - vertex 8 \(1 + 8 < \infty\) update

Example

- Next, visit vertex 2
  - vertex 3 \(4 + 1 < \infty\) update
  - vertex 4 already visited
  - vertex 5 \(4 + 6 \geq 8\) don’t update
  - vertex 6 \(4 + 1 < \infty\) update

Example

- Next, we have a choice of either 3 or 6
- We will choose to visit 3
  - vertex 5 \(5 + 2 < 8\) update
  - vertex 6 \(5 + 5 \geq 5\) don’t update

Example

- We then visit 6
  - vertex 8 \(5 + 7 \geq 9\) don’t update
  - vertex 9 \(5 + 8 < \infty\) update
Example

• Next, we finally visit vertex 5:
  – vertices 4 and 6 have already been visited
  – vertex 7 \(7 + 1 < 10\) update
  – vertex 8 \(7 + 1 < 9\) update
  – vertex 9 \(7 + 8 \geq 13\) don’t update

Example

• Given a choice between vertices 7 and 8, we choose vertex 7
  – vertices 5 has already been visited
  – vertex 8 \(8 + 2 \geq 8\) don’t update

Example

• Next, we visit vertex 8:
  – vertex 9 \(8 + 3 < 13\) update

Example

• Finally, we visit the end vertex
  • Therefore, the shortest path from 1 to 9 has length 11
Example

• We can find the shortest path by working back from the final vertex:
  – 9, 8, 5, 3, 2, 1
• Thus, the shortest path is (1, 2, 3, 5, 8, 9)

Example

• In the example, we visited all vertices in the graph before we finished
• This is not always the case, consider the next example

Example

• Find the shortest path from 1 to 4:
  – the shortest path is found after only three vertices are visited
  – we terminated the algorithm as soon as we reached vertex 4
  – we only have useful information about 1, 3, 4
  – we don’t have the shortest path to vertex 2

Dijkstra’s algorithm

\[
\begin{align*}
    &d[s] \leftarrow 0 \\
    &\text{for each } v \in V - \{s\} \quad \text{do } d[v] \leftarrow \infty \\
    &\quad S \leftarrow \emptyset \\
    &Q \leftarrow V \quad \triangleright Q \text{ is a priority queue maintaining } V - S \\
    &\text{while } Q \neq \emptyset \quad \text{do } u \leftarrow \text{Extract-Min}(Q) \\
    &\quad S \leftarrow S \cup \{u\} \\
    &\quad \text{for each } v \in \text{Adj}[u] \quad \text{do if } d[v] > d[u] + w(u, v) \\
    &\quad \quad \text{then } d[v] \leftarrow d[u] + w(u, v) \\
    &\quad \quad p[v] \leftarrow u
\end{align*}
\]
Dijkstra’s algorithm

\[ d[s] \leftarrow 0 \]
\[ \text{for each } v \in V - \{s\} \]
\[ \text{do } d[v] \leftarrow \infty \]
\[ S \leftarrow \emptyset \]
\[ Q \leftarrow V \quad \triangleright Q \text{ is a priority queue maintaining } V - S \]
\[ \text{while } Q \neq \emptyset \]
\[ \text{do } u \leftarrow \text{EXTRACT-MIN}(Q) \]
\[ S \leftarrow S \cup \{u\} \]
\[ \text{for each } v \in \text{Adj}[u] \]
\[ \text{do if } d[v] > d[u] + w(u, v) \]
\[ \text{then } d[v] \leftarrow d[u] + w(u, v) \]
\[ p[v] \leftarrow u \]
\[ \text{Implicit } \text{DECREASE-KEY} \]

Example of Dijkstra’s algorithm

Graph with nonnegative edge weights:

Example of Dijkstra’s algorithm

Example of Dijkstra’s algorithm

Example of Dijkstra’s algorithm

Initialize:

\[ Q: \quad A \quad B \quad C \quad D \quad E \]
\[ 0 \quad \infty \quad \infty \quad \infty \quad \infty \]
\[ S: \quad \{\} \]

“A” \leftarrow \text{EXTRACT-MIN}(Q):

\[ Q: \quad A \quad B \quad C \quad D \quad E \]
\[ 0 \quad \infty \quad \infty \quad \infty \quad \infty \]
\[ S: \quad \{A\} \]
Example of Dijkstra's algorithm

Relax all edges leaving A:

```
Q: A B C D E
   0 10 3 11 5
```

```
S: { A }
```

```
C ← EXTRACT-MIN(Q):
```

```
Q: A B C D E
   10 3 2 7 5
```

```
S: { A, C }
```

Example of Dijkstra’s algorithm

Relax all edges leaving C:

```
Q: A B C D E
   0 10 3 11 5
```

```
S: { A, C }
```

```
E ← EXTRACT-MIN(Q):
```

```
Q: A B C D E
   10 3 2 7 5
```

```
S: { A, C, E }
```
Example of Dijkstra’s algorithm

Relax all edges leaving $E$:

“$B$” $\leftarrow$ \texttt{EXTRACT-MIN($Q$)}:

“$D$” $\leftarrow$ \texttt{EXTRACT-MIN($Q$)}:
Summary

- Given a weighted directed graph, we can find the shortest distance between two vertices by:
  - starting with a trivial path containing the initial vertex
  - growing this path by always going to the next vertex which has the shortest current path

Bellman-Ford Algorithm

BellmanFord()

for each v ∈ V
  d[v] = ∞;
  d[s] = 0;
  for i=1 to |V|-1
    for each edge (u,v) ∈ E
      Relax(u,v, w(u,v));
    end
  end
  for each edge (u,v) ∈ E
    if (d[v] > d[u] + w(u,v))
      return "no solution";
  end
end
Relax(u,v,w): if (d[v] > d[u]+w) then d[v]=d[u]+w

DAG Shortest Paths

- Bellman-Ford takes O(VE) time.
- For finding shortest paths in a DAG, we can do much better by using a topological sort.
- If we process vertices in topological order, we are guaranteed to never relax a vertex unless the adjacent edge is already finalized. Thus: just one pass. O(V+E)

DAG-Shortest-Paths(G, w, s)

1. topologically sort the vertices of G
2. INITIALIZE-SINGLE-SOURCE(G, s)
3. for each vertex u, taken in topologically sorted order
   4. do for each vertex v ∈ Adj[u]
      5. do Relax(u, v, w)

Practice

<table>
<thead>
<tr>
<th>Node</th>
<th>Included</th>
<th>Distance</th>
<th>Path</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>t</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>B</td>
<td>t</td>
<td>4</td>
<td>A</td>
</tr>
<tr>
<td>C</td>
<td>t</td>
<td>2</td>
<td>A</td>
</tr>
<tr>
<td>D</td>
<td>t</td>
<td>5</td>
<td>A</td>
</tr>
<tr>
<td>E</td>
<td>t</td>
<td>10</td>
<td>B</td>
</tr>
<tr>
<td>F</td>
<td>t</td>
<td>8</td>
<td>B</td>
</tr>
</tbody>
</table>

Given the shortest path tree for node A for this graph using Dijkstra's shortest path algorithm. Show your work with the 3 arrays given and draw the resultant shortest path tree with edge weights included.
Usage Notes

- These slides are made publicly available on the web for anyone to use
- If you choose to use them, or a part thereof, for a course at another institution, I ask only three things:
  - that you inform me that you are using the slides,
  - that you acknowledge my work, and
  - that you alert me of any mistakes which I made or changes which you make, and allow me the option of incorporating such changes (with an acknowledgment) in my set of slides

Sincerely,
Douglas Wilhelm Harder, MMath
dwharder@alumni.uwaterloo.ca