Shortest paths with negative edge weights

Abhiram Ranade

March 8, 2016

Shortest paths with negative edge weights

Shortest paths with negative edge weights

Input: Graph G with positive or negative edge weights. Nodes s, t.

Shortest paths with negative edge weights

Input: Graph G with positive or negative edge weights. Nodes s, t.
Output: Least weight path from s to t.

Shortest paths with negative edge weights

Input: Graph G with positive or negative edge weights. Nodes s, t.
Output: Least weight path from s to t.

Dijkstra does not work:

Shortest paths with negative edge weights

Input: Graph G with positive or negative edge weights. Nodes s, t.
Output: Least weight path from s to t.

Dijkstra does not work: $w_{st} = 5$, $w_{su} = 6$, $w_{ut} = -3$.

Shortest paths with negative edge weights

Input: Graph G with positive or negative edge weights. Nodes s, t.
Output: Least weight path from s to t.

Dijkstra does not work: $w_{st} = 5$, $w_{su} = 6$, $w_{ut} = -3$.

Negative cycle: Directed cycle in G with total weight < 0.

Shortest paths with negative edge weights

Input: Graph G with positive or negative edge weights. Nodes s, t.
Output: Least weight path from s to t.

Dijkstra does not work: $w_{st} = 5$, $w_{su} = 6$, $w_{ut} = -3$.

Negative cycle: Directed cycle in G with total weight < 0.

Thm: Graph has a negative cycle $\Rightarrow$ shortest paths may not exist.

Shortest paths with negative edge weights

Input: Graph G with positive or negative edge weights. Nodes s, t.
Output: Least weight path from s to t.

Dijkstra does not work: $w_{st} = 5$, $w_{su} = 6$, $w_{ut} = -3$.

Negative cycle: Directed cycle in G with total weight < 0.

Thm: Graph has a negative cycle $\Rightarrow$ shortest paths may not exist.
Proof: Start at s, go to the negative cycle, traverse it several times, then to to t. Path weight can be made as small as we want.

Shortest paths with negative edge weights

Input: Graph G with positive or negative edge weights. Nodes s, t.
Output: Least weight path from s to t.

Dijkstra does not work: $w_{st} = 5$, $w_{su} = 6$, $w_{ut} = -3$.

Negative cycle: Directed cycle in G with total weight < 0.

Thm: Graph has a negative cycle $\Rightarrow$ shortest paths may not exist.
Proof: Start at s, go to the negative cycle, traverse it several times, then to to t. Path weight can be made as small as we want.

Thm: No negative cycles $\Rightarrow$ s, t shortest paths must exist.

Shortest paths with negative edge weights

Input: Graph G with positive or negative edge weights. Nodes s, t.
Output: Least weight path from s to t.

Dijkstra does not work: $w_{st} = 5$, $w_{su} = 6$, $w_{ut} = -3$.

Negative cycle: Directed cycle in G with total weight < 0.

Thm: Graph has a negative cycle $\Rightarrow$ shortest paths may not exist.
Proof: Start at s, go to the negative cycle, traverse it several times, then to to t. Path weight can be made as small as we want.

Thm: No negative cycles $\Rightarrow$ s, t shortest paths must exist.
Proof: Non simple path will not be shortest because we can remove the cycle and reduce the weight.

Shortest paths with negative edge weights

Input: Graph G with positive or negative edge weights. Nodes s, t.
Output: Least weight path from s to t.

Dijkstra does not work: $w_{st} = 5$, $w_{su} = 6$, $w_{ut} = -3$.

Negative cycle: Directed cycle in G with total weight < 0.

Thm: Graph has a negative cycle $\Rightarrow$ shortest paths may not exist.
Proof: Start at s, go to the negative cycle, traverse it several times, then to to t. Path weight can be made as small as we want.

Thm: No negative cycles $\Rightarrow$ s, t shortest paths must exist.
Proof: Non simple path will not be shortest because we can remove the cycle and reduce the weight.
Shortest path must have length at most $n - 1$.

Shortest paths with negative edge weights

Input: Graph G with positive or negative edge weights. Nodes s, t.
Output: Least weight path from s to t.

Dijkstra does not work: $w_{st} = 5$, $w_{su} = 6$, $w_{ut} = -3$.

Negative cycle: Directed cycle in G with total weight < 0.

Thm: Graph has a negative cycle $\Rightarrow$ shortest paths may not exist.
Proof: Start at s, go to the negative cycle, traverse it several times, then to to t. Path weight can be made as small as we want.

Thm: No negative cycles $\Rightarrow$ s, t shortest paths must exist.
Proof: Non simple path will not be shortest because we can remove the cycle and reduce the weight.
Shortest path must have length at most $n - 1$.

n = number of vertices, m = number of edges

Shortest paths with negative edge weights

Input: Graph G with positive or negative edge weights. Nodes s, t.
Output: Least weight path from s to t.

Dijkstra does not work: $w_{st} = 5$, $w_{su} = 6$, $w_{ut} = -3$.

Negative cycle: Directed cycle in G with total weight < 0.

Thm: Graph has a negative cycle $\Rightarrow$ shortest paths may not exist.
Proof: Start at s, go to the negative cycle, traverse it several times, then to to t. Path weight can be made as small as we want.

Thm: No negative cycles $\Rightarrow$ s, t shortest paths must exist.
Proof: Non simple path will not be shortest because we can remove the cycle and reduce the weight.
Shortest path must have length at most $n - 1$.

n = number of vertices, m = number of edges

Only finite number of paths to consider.

An attempt at a brute force recursive algorithm

An attempt at a brute force recursive algorithm

Decisions that we make in designing an algorithm:

What is the first node in the path?

An attempt at a brute force recursive algorithm

Decisions that we make in designing an algorithm:

What is the first node in the path? some out-neighbour of s

An attempt at a brute force recursive algorithm

Decisions that we make in designing an algorithm:

What is the first node in the path? some out-neighbour of s

What is the next ...

An attempt at a brute force recursive algorithm

Decisions that we make in designing an algorithm:

What is the first node in the path? some out-neighbour of s

What is the next ...

Optimal substructure: Path from s to t = edge from s to some neighbour si, || a shortest path from si to t.

An attempt at a brute force recursive algorithm

Decisions that we make in designing an algorithm:

What is the first node in the path? some out-neighbour of s

What is the next ...

Optimal substructure: Path from s to t = edge from s to some neighbour si, || a shortest path from si to t.

```
sPath(s,t){
 if s = t return {}
 for each out-neighbour si of s
   Pi = s || sPath(si,t)
 end for
 return the shortest among Pi
}
```

An attempt at a brute force recursive algorithm

Decisions that we make in designing an algorithm:

What is the first node in the path? some out-neighbour of s

What is the next ...

Optimal substructure: Path from s to t = edge from s to some neighbour si, || a shortest path from si to t.

```
sPath(s,t){
 if s = t return {}
 for each out-neighbour si of s
   Pi = s || sPath(si,t)
 end for
 return the shortest among Pi
}
```

Not proper recursion: Instances on which recursive calls are made are not "simpler" than original instance.

Make the recursion work!

Make the recursion work!

```
sPath(i,v){  // shortest v-t path of at most i edges
  if i = 0 and v = t then return null
  if i = 0 and v != t then return "invalid"

  P0 = sPath(i-1,v)  // if length(shortest path) < i
  for each out-neighbour vj of v
    Pi = v || sPath(i-1,vj)
  end for

  return shortest valid path among Pi.
}
```

Make the recursion work!

```
sPath(i,v){  // shortest v-t path of at most i edges
  if i = 0 and v = t then return null
  if i = 0 and v != t then return "invalid"

  P0 = sPath(i-1,v)  // if length(shortest path) < i
  for each out-neighbour vj of v
    Pi = v || sPath(i-1,vj)
  end for

  return shortest valid path among Pi.
}
```

$d(i, v) =$ length of shortest $v - t$ path of at most i edges.

Make the recursion work!

```
sPath(i,v){  // shortest v-t path of at most i edges
  if i = 0 and v = t then return null
  if i = 0 and v != t then return "invalid"

  P0 = sPath(i-1,v)  // if length(shortest path) < i
  for each out-neighbour vj of v
    Pi = v || sPath(i-1,vj)
  end for

  return shortest valid path among Pi.
}
```

$d(i, v) =$ length of shortest $v - t$ path of at most i edges.
$= 0$ if $i = 0$ and $v = t$

Make the recursion work!

```
sPath(i,v){  // shortest v-t path of at most i edges
  if i = 0 and v = t then return null
  if i = 0 and v != t then return "invalid"

  P0 = sPath(i-1,v)  // if length(shortest path) < i
  for each out-neighbour vj of v
    Pi = v || sPath(i-1,vj)
  end for

  return shortest valid path among Pi.
}
```

$d(i, v) =$ length of shortest $v - t$ path of at most i edges.

$= 0$ if $i = 0$ and $v = t$

$= \infty$ if $i = 0$ and $v \neq t$

Make the recursion work!

```
sPath(i,v){  // shortest v-t path of at most i edges
  if i = 0 and v = t then return null
  if i = 0 and v != t then return "invalid"

  P0 = sPath(i-1,v)  // if length(shortest path) < i
  for each out-neighbour vj of v
    Pi = v || sPath(i-1,vj)
  end for

  return shortest valid path among Pi.
}
```

$d(i, v) =$ length of shortest $v - t$ path of at most i edges.

$= 0$ if $i = 0$ and $v = t$

$= \infty$ if $i = 0$ and $v \neq t$

$= \min(d(i-1, v), \min_{u|v \sim u}(w_{vu} + d(i-1, u)))$ otherwise

The table

The table

$d(i, v) =$ length of shortest $v - t$ path of at most i edges.

The table

$d(i, v) =$ length of shortest $v - t$ path of at most i edges.
$= 0$ if $i = 0$ and $v = t$

The table

$d(i, v) =$ length of shortest $v - t$ path of at most i edges.

$= 0$ if $i = 0$ and $v = t$

$= \infty$ if $i = 0$ and $v \neq t$

The table

$d(i, v) =$ length of shortest $v - t$ path of at most i edges.

$= 0$ if $i = 0$ and $v = t$

$= \infty$ if $i = 0$ and $v \neq t$

$= \min(d(i-1, v), \min_{u|v \sim u}(w_{vu} + d(i-1, u)))$ otherwise

The table

$d(i, v) =$ length of shortest $v - t$ path of at most i edges.
$= 0$ if $i = 0$ and $v = t$
$= \infty$ if $i = 0$ and $v \neq t$
$= \min(d(i-1, v), \min_{u|v \sim u}(w_{vu} + d(i-1, u)))$ otherwise

i need be at most $n - 1$. So we need a table $T[1..n-1, 1..n]$

The table

$d(i, v)$ = length of shortest $v - t$ path of at most i edges.

$= 0$ if $i = 0$ and $v = t$

$= \infty$ if $i = 0$ and $v \neq t$

$= \min(d(i-1, v), \min_{u|v\sim u}(w_{vu} + d(i-1, u)))$ otherwise

i need be at most $n-1$. So we need a table $T[1..n-1, 1..n]$

$T[0, t] = 0$

The table

$d(i, v)$ = length of shortest $v - t$ path of at most i edges.

$= 0$ if $i = 0$ and $v = t$

$= \infty$ if $i = 0$ and $v \neq t$

$= \min(d(i-1, v), \min_{u|v\sim u}(w_{vu} + d(i-1, u)))$ otherwise

i need be at most $n - 1$. So we need a table $T[1..n-1, 1..n]$

$T[0, t] = 0$

$T[0, v] = \infty$ if $v \neq t$

The table

$d(i, v)$ = length of shortest $v - t$ path of at most i edges.

$= 0$ if $i = 0$ and $v = t$

$= \infty$ if $i = 0$ and $v \neq t$

$= \min(d(i-1, v), \min_{u | v \sim u}(w_{vu} + d(i-1, u)))$ otherwise

i need be at most $n - 1$. So we need a table $T[1..n-1, 1..n]$

$T[0, t] = 0$
$T[0, v] = \infty$ if $v \neq t$
for $i = 1..n-1$

The table

$d(i, v)$ = length of shortest $v - t$ path of at most i edges.
$= 0$ if $i = 0$ and $v = t$
$= \infty$ if $i = 0$ and $v \neq t$
$= \min(d(i-1, v), \min_{u|v\sim u}(w_{vu} + d(i-1, u)))$ otherwise

i need be at most $n - 1$. So we need a table $T[1..n-1, 1..n]$

$T[0, t] = 0$
$T[0, v] = \infty$ if $v \neq t$
for $i = 1..n - 1$
    for $v = 1..n$ except t

The table

$d(i, v)$ = length of shortest $v - t$ path of at most i edges.

$$
\begin{aligned}
&= 0 && \text{if } i = 0 \text{ and } v = t \\
&= \infty && \text{if } i = 0 \text{ and } v \neq t \\
&= \min(d(i-1, v), \min_{u|v\sim u}(w_{vu} + d(i-1, u))) && \text{otherwise}
\end{aligned}
$$

i need be at most $n - 1$. So we need a table $T[1..n-1, 1..n]$

$T[0, t] = 0$
$T[0, v] = \infty$ if $v \neq t$
for $i = 1..n-1$
    for $v = 1..n$ except t
        $T[i, v] = \min(T[i-1, v], \min_{u|v\sim u}(w_{vu} + T[i-1, u]))$

The table

$d(i, v) =$ length of shortest $v - t$ path of at most i edges.
$= 0$ if $i = 0$ and $v = t$
$= \infty$ if $i = 0$ and $v \neq t$
$= \min(d(i-1, v), \min_{u|v \sim u}(w_{vu} + d(i-1, u)))$ otherwise

i need be at most $n - 1$. So we need a table $T[1..n-1, 1..n]$

$T[0, t] = 0$
$T[0, v] = \infty$ if $v \neq t$
for $i = 1..n-1$
    for $v = 1..n$ except t
        $T[i, v] = \min(T[i-1, v], \min_{u|v \sim u}(w_{vu} + T[i-1, u]))$
    end for

The table

$d(i, v) =$ length of shortest $v - t$ path of at most i edges.
$= 0$ if $i = 0$ and $v = t$
$= \infty$ if $i = 0$ and $v \neq t$
$= \min(d(i-1, v), \min_{u|v \sim u}(w_{vu} + d(i-1, u)))$ otherwise

i need be at most $n - 1$. So we need a table $T[1..n-1, 1..n]$

$T[0, t] = 0$
$T[0, v] = \infty$ if $v \neq t$
for $i = 1..n-1$
    for $v = 1..n$ except t
        $T[i, v] = \min(T[i-1, v], \min_{u|v \sim u}(w_{vu} + T[i-1, u]))$
    end for
end for

The table

$d(i, v) =$ length of shortest $v - t$ path of at most i edges.

$= 0$ if $i = 0$ and $v = t$

$= \infty$ if $i = 0$ and $v \neq t$

$= \min(d(i-1, v), \min_{u|v \sim u}(w_{vu} + d(i-1, u)))$ otherwise

i need be at most $n - 1$. So we need a table $T[1..n-1, 1..n]$

$T[0, t] = 0$
$T[0, v] = \infty$ if $v \neq t$
for $i = 1..n-1$
    for $v = 1..n$ except t
        $T[i, v] = \min(T[i-1, v], \min_{u|v \sim u}(w_{vu} + T[i-1, u]))$
    end for
end for

Innermost loop time $= \sum_v outdegree(v) = O(m)$.

The table

$d(i, v) =$ length of shortest $v - t$ path of at most i edges.

$$
\begin{aligned}
&= 0 && \text{if } i = 0 \text{ and } v = t \\
&= \infty && \text{if } i = 0 \text{ and } v \neq t \\
&= \min(d(i-1, v), \min_{u|v\sim u}(w_{vu} + d(i-1, u))) && \text{otherwise}
\end{aligned}
$$

i need be at most $n - 1$. So we need a table $T[1..n-1, 1..n]$

$T[0, t] = 0$
$T[0, v] = \infty$ if $v \neq t$
for $i = 1..n - 1$
  for $v = 1..n$ except t
    $T[i, v] = \min(T[i-1, v], \min_{u|v\sim u}(w_{vu} + T[i-1, u]))$
  end for
end for

Innermost loop time $= \sum_v outdegree(v) = O(m)$.
Total time $= O(mn)$.