All pairs Shortest path

Robin Jadoul

OLYMPIADE BELGE D'INFORMATIQUE
BELGISCHE INFORMATICA-OLYMPIADE

March 13, 2016

Table of Contents

Floyd-Warshall

All pairs Shortest path

- Adjacency matrix

Floyd-Warshall

All pairs Shortest path

- Adjacency matrix
- For every node k:

Floyd-Warshall

All pairs Shortest path

- Adjacency matrix
- For every node k:
-   For every pair of nodes (u, v):

Floyd-Warshall

All pairs Shortest path

- Adjacency matrix
- For every node k:
- For every pair of nodes (u, v):
- Can the distance between these pairs be improved by going to k first?

Floyd-Warshall

All pairs Shortest path

- Adjacency matrix
- For every node k:
- For every pair of nodes (u, v):
- Can the distance between these pairs be improved by going to k first?
- $g[u][v] = min(g[u][v], g[u][k] + g[k][v])$

Floyd-Warshall

All pairs Shortest path

- Adjacency matrix
- For every node k:
-   For every pair of nodes (u, v):
-     Can the distance between these pairs be improved by going to k first?
-     $g[u][v] = min(g[u][v], g[u][k] + g[k][v])$
- Very easy implementation (4 lines)

Floyd-Warshall

All pairs Shortest path

- Adjacency matrix
- For every node k:
-   For every pair of nodes (u, v):
-     Can the distance between these pairs be improved by going to k first?
-     $g[u][v] = min(g[u][v], g[u][k] + g[k][v])$
- Very easy implementation (4 lines)
- $O(|V|^3) \Rightarrow$ feasible for up to approx. 400 nodes

Floyd-Warshall

All pairs Shortest path

- Reconstructing the path:

Floyd-Warshall

All pairs Shortest path

- Reconstructing the path:
- Use another $|V| \times |V|$ matrix *parent*

Floyd-Warshall

All pairs Shortest path

- Reconstructing the path:
- Use another $|V| \times |V|$ matrix *parent*
- $parent[u][v] =$ The last node before v on the shortest path from u to v

Floyd-Warshall

All pairs Shortest path

- Reconstructing the path:
- Use another $|V| \times |V|$ matrix *parent*
- $parent[u][v] =$ The last node before v on the shortest path from u to v
- Update when improving a path ($parent[u][v] = parent[k][v]$)

Floyd-Warshall

All pairs Shortest path

- Reconstructing the path:
- Use another $|V| \times |V|$ matrix *parent*
- $parent[u][v] =$ The last node before v on the shortest path from u to v
- Update when improving a path ($parent[u][v] = parent[k][v]$)
- Traverse the *parent chain* upwards

Floyd-Warshall

Alternative

- Repeated execution of Dijkstra's algorithm

Floyd-Warshall

Alternative

- Repeated execution of Dijkstra's algorithm
- $O(|V| \times |E| \times \log |V|)$

Floyd-Warshall

Alternative

- Repeated execution of Dijkstra's algorithm
- $O(|V| \times |E| \times \log |V|)$
- Better on sparse graphs ($|E| << |V|^2$)

Table of Contents

All pairs Shortest path

Applications

- Compute the transitive closure
  $close[u][v] = close[u][v] | (close[u][k] \& close[k][v])$

All pairs Shortest path

Applications

- Compute the transitive closure
  $close[u][v] = close[u][v] | (close[u][k] \& close[k][v])$
- Minimax of the graph: The path with minimal highest cost along the path
  $g[u][v] = min(g[u][v], max(g[u][k], g[k][v]))$

All pairs Shortest path

Applications

- Compute the transitive closure
  $close[u][v] = close[u][v] | (close[u][k] \& close[k][v])$
- Minimax of the graph: The path with minimal highest cost along the path
  $g[u][v] = min(g[u][v], max(g[u][k], g[k][v]))$
- Maximin of the graph: The path with maximal lowest cost along the path
  $g[u][v] = max(g[u][v], min(g[u][k], g[k][v]))$

All pairs Shortest path

Applications

- Compute the transitive closure
  $close[u][v] = close[u][v] | (close[u][k] \& close[k][v])$
- Minimax of the graph: The path with minimal highest cost along the path
  $g[u][v] = min(g[u][v], max(g[u][k], g[k][v]))$
- Maximin of the graph: The path with maximal lowest cost along the path
  $g[u][v] = max(g[u][v], min(g[u][k], g[k][v]))$
- Safest path: the path with highest survival probability (weights are probs along the edge)
  $g[u][v] = max(g[u][v], g[u][k] * g[k][v])$

All pairs Shortest path

Applications

- Compute the transitive closure
  $close[u][v] = close[u][v] | (close[u][k] \& close[k][v])$
- Minimax of the graph: The path with minimal highest cost along the path
  $g[u][v] = min(g[u][v], max(g[u][k], g[k][v]))$
- Maximin of the graph: The path with maximal lowest cost along the path
  $g[u][v] = max(g[u][v], min(g[u][k], g[k][v]))$
- Safest path: the path with highest survival probability (weights are probs along the edge)
  $g[u][v] = max(g[u][v], g[u][k] * g[k][v])$
- Most dangerous path: the path with lowest survival probability (weights are probs along the edge)
  $g[u][v] = min(g[u][v], g[u][k] * g[k][v])$

All pairs Shortest path

Applications

- Detecting negative weight cycles

All pairs Shortest path

Applications

- Detecting negative weight cycles
- Run the normal *Floyd-Warshall* algorithm

All pairs Shortest path

Applications

- Detecting negative weight cycles
- Run the normal *Floyd-Warshall* algorithm
- If any element on the diagonal becomes negative $\Rightarrow$ negative weight cycle found