Algorithmik

Sebastian Iwanowski
FH Wedel

4. Graph algorithms
4.2 Shortest paths

SSSP: Single Source Shortest Path

Find the shortest paths from a source s to all other nodes

Remark: For the problem to find the shortes path between two given nodes there is no better algorithm known than those for SSSP, and those have not been proved being optimal even for SSSP.

Algorithm of Dijkstra for SSSP: (for graphs G with nonnegative edge costs only)

- Initialize the node set **Done** by s;
  Initialize the node set **Undone** by all other nodes of graph G;
  For all nodes v of the graph G:
  Let label (v) := length of edge between v and s (∞ if no edge is existing, 0 if v = s);
- While **Undone** is not empty:
  Search and delete the node v from **Undone** with minimal label;
  Insert v into **Done**;
  Update all neighbors n of v that are in **Undone**:
  If label (n) > label (v) + length of edge between v and n:
  Replace label (n) by that number;
  Let v be the predecessor of n.

Theorem: The labels of nodes v in **Done** are always the shortest path length from s to v and the shortest path is the shortest path from s to the predecessor of v followed by the edge from the predecessor to v.

Proof: Complete induction by number of iterations.

SSSP: Single Source Shortest Path

Find the shortest paths from a source s to all other nodes

Remark: For the problem to find the shortes path between two given nodes there is no better algorithm known than those for SSSP, and those have not been proved being optimal even for SSSP.

Algorithm of Dijkstra for SSSP: (for graphs G with nonnegative edge costs only)

Organize the edge costs in a heap.

Time complexity: O((m+n)log n)

for arbitrary graphs: $O(n^2 \log n)$
for graphs with a constant number of neighbors per node: O(n log n)

References:

Skript Alt 4.4.1 (p. 79-81),
Cormen, ch. 24 (much more detailed: SSSP)

Algorithmics 4

APSP: All Pairs Shortest Path

Find the shortest paths between all pairs of nodes

Trivial solution: Apply Dijkstra iteratively for all nodes as sources

Time complexity: O(n(m+n)log n))

for arbitrary graphs: $O(n^3 \log n)$

for graphs with a constant number of neighbors per node: $O(n^2 \log n)$

Algorithm of Floyd-Warshall:

Let V = {1,...n}.
$d_{ij}^{(k)}$ is the length of the shortest path between i and j using in between at most nodes from {1,...k}.

for $i = 1, \dots, n$ **do**
  for $j = 1, \dots, n$ **do**
    $d_{ij}^{(0)} = \begin{cases} c(i,j): & \text{falls } (i,j) \in E \\ \infty: & \text{sonst} \end{cases}$
  end for
end for
for $k = 1, \dots, n$ **do**
  for $i = 1, \dots, n$ **do**
    for $j = 1, \dots, n$ **do**
      $d_{ij}^{(k)} = \min(d_{ij}^{(k-1)}, d_{ik}^{(k-1)} + d_{kj}^{(k-1)})$
    end for
  end for
end for

Time complexity: $O(n^3)$

References:

Skript Alt 4.4.2, 4.4.3 (p. 81-83),
Cormen, ch. 25.2 (Floyd-Warshall)

Algorithmics 4

APSP: All Pairs Shortest Path

Find the shortest paths between all pairs of nodes

Relation to matrix multiplication:

Let V = {1,...n}.
$d_{ij}^{(k)}$ is the length of the shortest path between i und j using at most k edges.
Note: This definition is different from Floyd-Warshall's!

Theorem: Let A be the adjacency matrix.
Define the operation min instead of addition and the operation + instead of multiplication.

Then A^k holds in position (i,j) the length $d_{ij}^{(k)}$.
In particular, A^{n-1} holds in position (i,j) the length of the shortest path from i to j.

Quadratic potentiation: A^{n-1} may be computed with O(log n) matrix multiplications.

Strassens's algorithm: Two nxn-matrices may be multiplied with $O(n^{\log 7})$ operations.

Conclusion for APSP: Time complexity $O(n^{\log 7} \log n)$ Note that $\log 7 \approx 2{,}81$

References for a deeper insight:

Cormen, ch. 25.1 (relation to matrix multiplication), ch. 28.2 (Strassen's algorithm)