Algorithmics

Sebastian Iwanowski
FH Wedel

4. Graph algorithms
4.2 Shortest paths

Algorithmics 4

SSSP: Single Source Shortest Path

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

Remark: For the problem to find the shortest 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 from s to v (∞ 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: Mathematical induction by number of iterations.

Algorithmics 4

SSSP: Single Source Shortest Path

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

Theorem of correctness (to be proven):

For each node v ∈ **`Done`** holds:

label (v) = $d_s(v)$.

where label (v) is the length of a path found from s to v
and $d_s(v)$ is the length of a shortest path from s to v

Lemma Subpath:

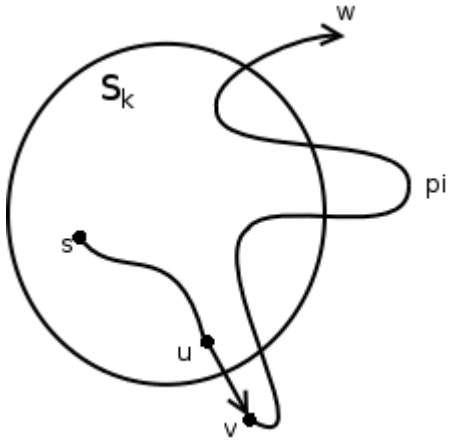
For each node u on the shortest path from s to v holds:
The subpath from s to u is the shortest path from s to u.

Proof by contraposition:
Let u be on any path from s to v and the subpath from s to u be not the shortest path from s to u.
Then the path from s to v is not the shortest path from s to v.
(Details: Exercise)

Algorithmics 4

SSSP: Single Source Shortest Path

Proof of the theorem of correctness by mathematical induction over iteration cycle i, which shifts v to `Done`:



Base case for i = 0 clear (Exercise!)

Inductive step ≤ i -> i+1:

(*) Let w be the node, which is shifted to **`Done`** in the (i+1). cycle

Assume, label (w) ≠ $d_s(w)$ (will be used to contradict (*)).

Since label (w) = ∞ or label (w) = length of some path to w, the following holds: $d_s(w)$ < label (w).
Let (u,v) be the first edge on the shortest path to w leaving **`Done`**, i.e. u ∈ **`Done`** and v ∉ **`Done`**.

shortest path some path

Then: $d_s(v) = d_s(u) + c(u,v) = \text{label}(u) + c(u,v) \geq \text{label}(v)$ => $d_s(v) = \text{label}(v)$

cf. Lemma Subpath Ind.ass. ass. above

has been set to minimum of path via u (since u ∈ **`Done`**) and previous value

This holds: label(v) = $d_s(v) \leq d_s(w) <$ label (w).
Since v,w ∉ **`Done`**, the following holds: v is shifted to **`Done`** before w => contradiction to (*)

Thus: label (w) = $d_s(w)$ q.e.d.

Algorithmics 4

SSSP: Single Source Shortest Path

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

Remark: For the problem to find the shortest 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) (by direct inspection of the nodes)

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

Remark: Using the special structure **`Fibonacci Heap`** with corresponding methods for decreasing keys, a very careful analysis using amortised time arguments over all loops shows that the worst case is O(m + n log n) only. This is an improvement for dense graphs only.

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: $O(n(m+n)\log n)$ (or only $O(n(m + n \log n)$

for arbitrary graphs: $O(n^3 \log n)$ (or even $O(n^3)$)
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}.

Time complexity: $O(n^3)$ (with simple implementation)

Other advantage of FW to Dijkstra:
FW works also for **negative** weights
(but no negative cycles).

```
1:  for i = 1,...,n do
2:    for j = 1,...,n do
3:      d_ij^(0) = { c(i,j):  falls (i,j) ∈ E
                   { ∞:      sonst
4:    end for
5:  end for
6:  for k = 1,...,n do
7:    for i = 1,...,n do
8:      for j = 1,...,n do
9:        d_ij^(k) = min(d_ij^(k-1), d_ik^(k-1) + d_kj^(k-1))
10:     end for
11:   end for
12: end for
```

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 stores in position (i,j) the length $d_{ij}^{(k)}$.
In particular, A^{n-1} stores 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.

Standard matrix multiplication: 2 matrices may be multiplied with $O(n^3)$ number operations.

Conclusion for APSP: $O(n^3 \log n)$ number operations (worse than Floyd-Warshall!)

References for a deeper insight:

Cormen, ch. 25.1 (relation to matrix multiplication)

Algorithmics 4

APSP: All Pairs Shortest Path

Find the shortest paths between all pairs of nodes

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

Note that $\log 7 \approx 2{,}81$

Conclusion for APSP? Time complexity $O(n^{\log 7} \log n)$ **?**

Unfortunately, no!

Strassen‘s algorithm needs inverse functions to the additive operation.
This does not hold for the minimum operation needed in the transform from APSP to matrix multiplication.

References for a deeper insight:

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