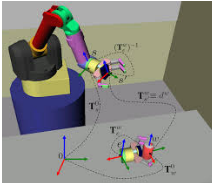
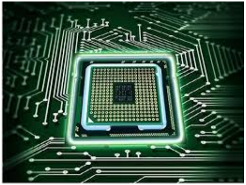
Shortest Path (Fall 2019)

Myriad of applications

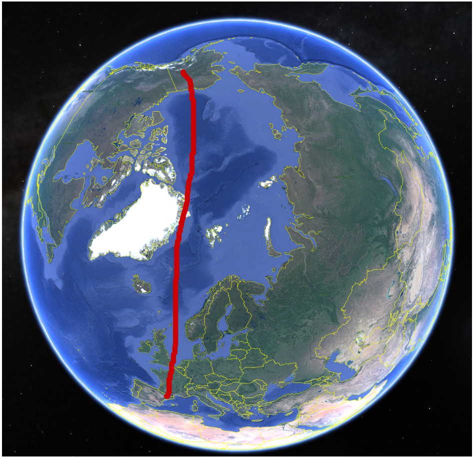
- Finding shortest distances between 2 locations (Google maps, etc.)
- Internet router protocols: OSPF (Open Shortest Path First) is used to find the shortest path to interchange packages between servers (IP)
- Traffic information systems
- Routing in VSLI
- etc ...





Shortest distance between two points not always follow human intuition

It may depend on many more constrains than the pure geometric ones.

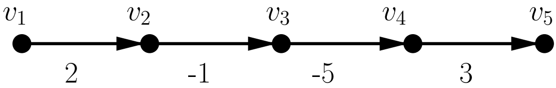


Shortest path problems in direct weighted graphs

Given a digraph $G = (V, \vec{E})$ with edge's weights $w : \vec{E} \to \mathbb{R}$, a path $p = \{v_0, \dots, v_k\}$ is a sequence of consecutive edges, where $(v_i, v_{i+1}) \in \vec{E}$ define $w(p) = \sum_{i=0}^{k-1} w(v_i, v_{i+1})$.

The shortest path between u and v as

$$\delta(u, v) = \min_p \{w(p) | u \rightsquigarrow^p v\}$$

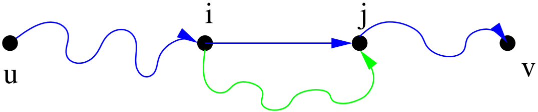


$\delta(v_1, v_5) = -1$

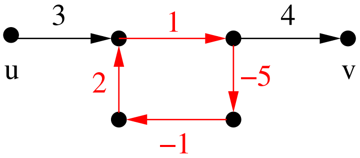
If G is undirected, we can consider every edge as doubly directed.
Unweighted, every edge of weight $=1$.

Optimal substructure of shortest path

Given $G = (V, \vec{E}), w : \vec{E} \to \mathbb{R}$, for any shortest path $p : u \rightsquigarrow v$ and any i, j vertices in p, the sub-path $p' = i \rightsquigarrow j$ in p has the shortest distance $\delta(i, j)$.



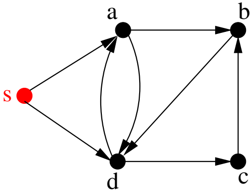
Negative cycles



$\delta(u, v) = -\infty$

Taxonomy of shortest path problems

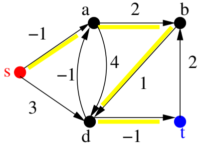
- Single source shortest path (SSSP): Given $G = (V, \vec{E}), w : \vec{E} \to \mathbb{R}$ and $s \in V$, compute $\delta(s, v), \forall v \in V - \{s\}$.
  In the graph below we want to compute $(s, a), (s, b), (s, c), (s, d)$
- All paths shortest paths (APSP): Given $= (V, \vec{E}), w : \vec{E} \to \mathbb{R}$ compute $\delta(u, v)$ for every pair $(u, v) \in V \times V$.
  In the graph below we want to compute $(s, a), (a, s), \ldots (d, b), (b, d), (d, c), (c, d)$

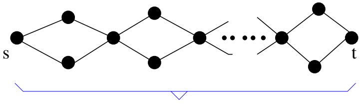


Single source shortest path

Let us consider the particular case of having a source s and a sink t Assume that $w : e \to \mathbb{R}^+$

Brute-force(G, W, s, t)
for all simple $p : s \to t$ **do**
  compute $w(p)$
end for
Compare all p
return the p with smallest $w(p)$





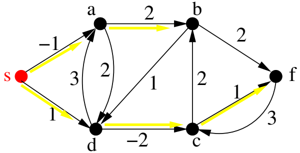
The number of paths could be $O(2^n)$

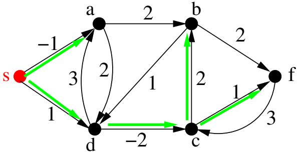
Shortest Path Tree

SSSP algorithms have the property that at termination the resulting paths form a shortest path tree.

Given $G = (V, \vec{E})$ with edge weights w_e and a distinguished $s \in V$, a shortest path tree is a directed sub-tree $T_s = (V', \vec{E}')$ of G, s.t.

- T_s is rooted at s,
- V' is the set of vertices in G reachable from s,
- $\forall v \in V'$ the path $s \rightsquigarrow v$ in T_s is the shortest path $\delta(s, v)$.





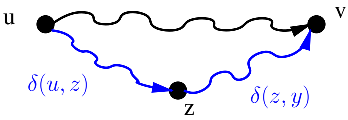
Triangle Inequality

Recall that $\delta(u, v)$ is shortest distance from $u \to v$

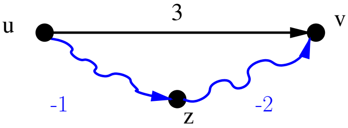
Given $G = (V, \vec{E}), W$, if $u, v, z \in V$, notice the shortest path $u \rightsquigarrow v$ is $\leq$ any other path between u and v. Therefore.

Theorem

For all $u, v, z \in V$ $\delta(u, v) \leq \delta(u, z) + \delta(z, v)$.



Want minimum $\delta(u, v)$



Notice, in this case $\delta(u, v) = -3$

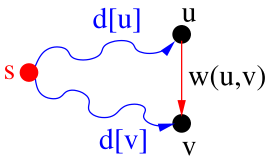
Basic technique for SSSP: Relaxation

Given $G = (V, \vec{E}), W$. $\forall v \in V$ we maintain a SP-estimate $d[v]$, which is an UB on $\delta(s, v)$.

Initially, start with $d[v] = +\infty, \forall v \in V - \{s\}$ and $d[s] = 0$. Repeatedly improve estimates toward the goal $d[v] = \delta(s, v)$.

For $(u, v) \in \vec{E}$,

Relax$(u, v, w(u, v))$
if $d[v] > d[u] + w(u, v)$
then
  $d[v] = d[u] + w(u, v)$
end if



Generic Relaxation algorithm

Relaxation(G, W, s)
for all $v \in V - \{s\}$ **do**
  $d[v] = +\infty$
end for
$d[s] = 0$
while $\exists(u, v)$ with $d[v] > d[u] + w(u, v)$ **do**
  Relax$(u, v, w(u, v))$
end while

How do we implement this function?

Can we replace the condition $d[v] > d[u] + w(u, v)$ by $d[v] \geq d[u] + w(u, v)$?

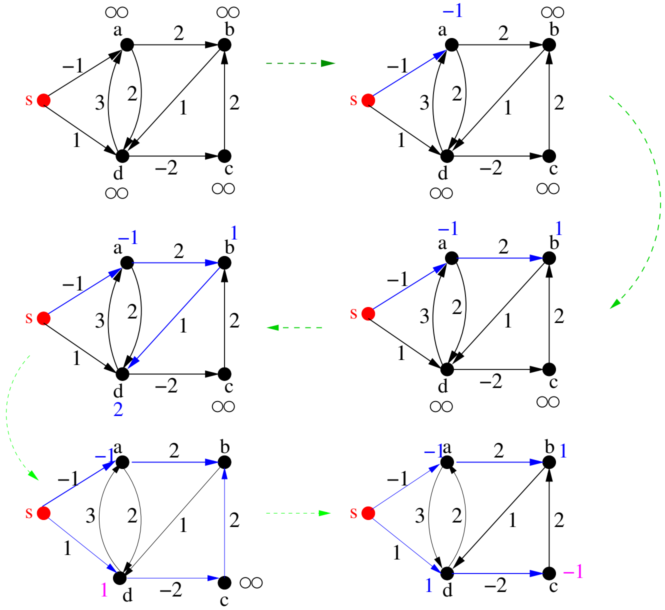
Generic Relaxation algorithm

Relaxation(G, W, s)
for all $v \in V - \{s\}$ **do**
  $d[v] = +\infty$
end for
$d[s] = 0$
while $\exists(u, v)$ with $d[v] > d[u] + w(u, v)$ **do**
  Relax$(u, v, w(u, v))$
end while

Lemma
For all $v \in V$*,* **Relaxation**(G, W, s) *maintains the invariant that* $d[v] \geq \delta(s, v)$*.*

Proof (Induction)
I.H. when applying **Relax**$(u, v, w(u, v))$ we get $d[u] \geq \delta(s, u)$
By the triangle ineq. $\delta(s, v) \leq \delta(s, u) + \delta(u, v) \leq d[u] + w(u, v)$.
Therefore, letting $\delta(u, v) = d[u] + w(u, v)$ is not a problem. $\square$

Generic Relaxation algorithm



Recall: Dijkstra SSSP

E.W.Dijkstra, *"A note on two problems in connexion with graphs"*. Num. Mathematik 1, (1959)

- Greedy algorithm.
- Relax edges in an increasing ball around s.
- Uses a priority queue Q
- Dijkstra does not work with negative weights

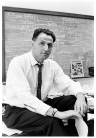
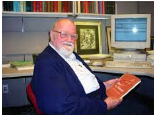
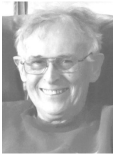
Dijkstra(G, W, s)
Initialize SP-estimates on V
$S = \emptyset$, $Q = \{V\}$
while $Q \neq \emptyset$ **do**
  $u =$EXT-MIN(Q)
  $S = S \cup \{u\}$
  for all $v \in Adj[u]$ **do**
    Relax$(u, v, w(u, v))$
  end for
end while

Dijkstra is the fastest SSSP algorithm, if $w \geq 0$.

Q implementation	Worst-time complexity
Array	$O(n^2)$
Heap	$O(m \lg n)$
Fibonacci heap	$O(m + n \lg n)$

Bellman-Ford-Moore-Shimbel SSSP

R. Bellman (1958)
L. Ford (1956)
E. Moore (1957)
A. Shimbel (1955)
(Shimbel matrices)

- The algorithm BFMS is used for G with negative weights, but without negative cycles.
- Given $G, w, s \in V(G)$,with n vertices and m edges, the BFMS algorithm does $n-1$ iterations:
- Each iteration i does a relaxation on all edges than can be reached from s in at most i-steps, the remaining ones are set to ∞

$$\underbrace{(e_1, e_2, \ldots, e_n)}_{i=1}, \underbrace{(e_1, e_2, \ldots, e_n)}_{i=2}, \cdots\cdots \underbrace{(e_1, e_2, \ldots, e_n)}_{i=n-1}$$

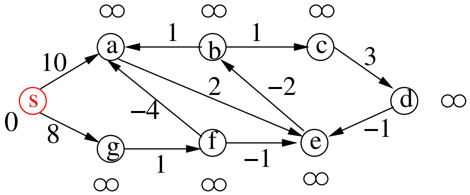
BFMS Algorithm

Recall that given a graph G, $|V| = n, |E| = m$, and a set of edges' weights w with a source vertex $v \in V$. Recall $\pi[v] = u$ points to the u used to compute $d[v]$.

BFMS (G, w, s)
Initialize $\forall v \neq s, d[v] = \infty$, $\pi[v] = v$
Initialize $d[s] = 0$
for $i = 1$ to $n - 1$ **do**
  for every $(u, v) \in E$ **do**
    Relax$(u, v, w(u, v))$
  end for
end for
for every $(u, v) \in E$ **do**
  if $d[v] > d[u] + w(u, v)$ **then**
    return Negative-weight cycle
  end if
end for

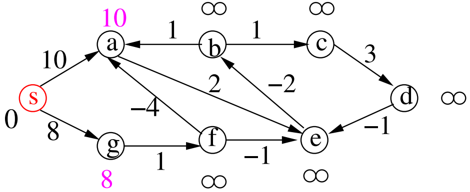
Relax$(u, v, w(u, v))$
if $d[v] > d[u] + w(u, v)$ **then**
  $d[v] = d[u] + w(u, v)$
  $\pi[v] = u$
end if

BFMS Algorithm: Example



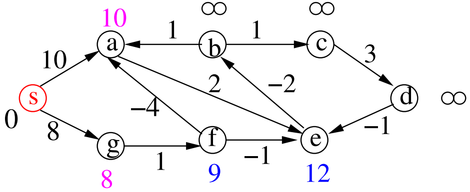
Node				*i*				
	0	1	2	3	4	5	6	7
s	0	0	0	0	0	0	0	0
a	∞	∞	∞	∞	∞	∞	∞	∞
b	∞	∞	∞	∞	∞	∞	∞	∞
c	∞	∞	∞	∞	∞	∞	∞	∞
d	∞	∞	∞	∞	∞	∞	∞	∞
e	∞	∞	∞	∞	∞	∞	∞	∞
f	∞	∞	∞	∞	∞	∞	∞	∞
g	∞	∞	∞	∞	∞	∞	∞	∞

BFMS Algorithm: Example



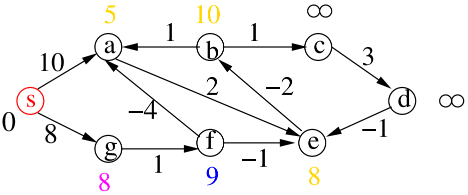
Node				*i*				
	0	1	2	3	4	5	6	7
s	0	0	0	0	0	0	0	0
a	∞	10	10	10	10	10	10	10
b	∞	∞	∞	∞	∞	∞	∞	∞
c	∞	∞	∞	∞	∞	∞	∞	∞
d	∞	∞	∞	∞	∞	∞	∞	∞
e	∞	∞	∞	∞	∞	∞	∞	∞
f	∞	∞	∞	∞	∞	∞	∞	∞
g	∞	8	8	8	8	8	8	8

BFMS Algorithm: Example



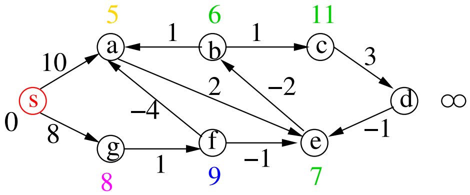
Node				*i*				
	0	1	2	3	4	5	6	7
s	0	0	0	0	0	0	0	0
a	∞	10	10	10	10	10	10	10
b	∞	∞	∞	∞	∞	∞	∞	∞
c	∞	∞	∞	∞	∞	∞	∞	∞
d	∞	∞	∞	∞	∞	∞	∞	∞
e	∞	∞	12	12	12	12	12	12
f	∞	∞	9	9	9	9	9	9
g	∞	8	8	8	8	8	8	8

BFMS Algorithm: Example



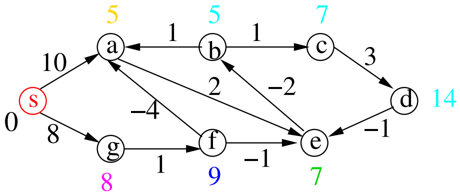
Node				*i*				
	0	1	2	3	4	5	6	7
s	0	0	0	0	0	0	0	0
a	∞	10	10	5	5	5	5	5
b	∞	∞	∞	10	10	10	10	10
c	∞	∞	∞	∞	∞	∞	∞	∞
d	∞	∞	∞	∞	∞	∞	∞	∞
e	∞	∞	12	8	8	8	8	8
f	∞	∞	9	9	9	9	9	9
g	∞	8	8	8	8	8	8	8

BFMS Algorithm: Example



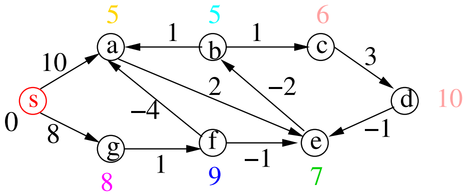
Node				i				
	0	1	2	3	4	5	6	7
s	0	0	0	0	0	0	0	0
a	∞	10	10	5	5	5	5	5
b	∞	∞	∞	10	6	10	10	10
c	∞	∞	∞	∞	11	11	11	11
d	∞	∞	∞	∞	∞	∞	∞	∞
e	∞	∞	12	8	7	7	7	7
f	∞	∞	9	9	9	9	9	9
g	∞	8	8	8	8	8	8	8

BFMS Algorithm: Example



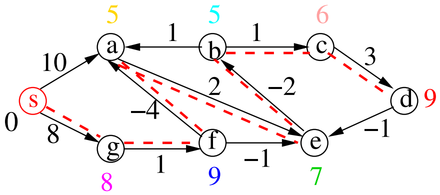
Node				*i*				
	0	1	2	3	4	5	6	7
s	0	0	0	0	0	0	0	0
a	∞	10	10	5	5	5	5	5
b	∞	∞	∞	10	6	5	5	5
c	∞	∞	∞	∞	11	7	7	7
d	∞	∞	∞	∞	∞	14	14	14
e	∞	∞	12	8	7	7	7	7
f	∞	∞	9	9	9	9	9	9
g	∞	8	8	8	8	8	8	8

BFMS Algorithm: Example



Node				*i*				
	0	1	2	3	4	5	6	7
s	0	0	0	0	0	0	0	0
a	∞	10	10	5	5	5	5	5
b	∞	∞	∞	10	6	5	5	5
c	∞	∞	∞	∞	11	7	6	6
d	∞	∞	∞	∞	∞	14	10	14
e	∞	∞	12	8	7	7	7	7
f	∞	∞	9	9	9	9	9	9
g	∞	8	8	8	8	8	8	8

BFMS Algorithm: Example



Node				i				
	0	1	2	3	4	5	6	7
s	0	0	0	0	0	0	0	0
a	∞	10	10	5	5	5	5	5
b	∞	∞	∞	10	6	5	5	5
c	∞	∞	∞	∞	11	7	6	6
d	∞	∞	∞	∞	∞	14	10	9
e	∞	∞	12	8	7	7	7	7
f	∞	∞	9	9	9	9	9	9
g	∞	8	8	8	8	8	8	8

Complexity BFMS

BFM (G, w, s)
Initialize $\forall v \neq s, d[v] = \infty,\ \pi[v] = u$
Initialize $d[s] = 0$
for $i = 1$ to $n - 1$ **do**
  for every $(u, v) \in E$ **do**
    Relax$(u, v, w(u, v))$
  end for
end for
for every $(u, v) \in E$ **do**
  if $d[v] > d[u] + w(u, v)$ **then**
    return Negative-weight cycle
  end if
end for

O(n x (n−1))

O(1) O(m) O(nm)

O(m)

Complexity T(n)=O(nm)

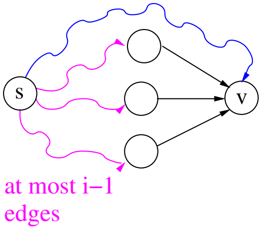
Correctness of BFMS

Lemma

In the BFMS-algorithm, after the ith. iteration we have that $d[v] \leq$ the weight of every path $s \rightsquigarrow v$ using at most i edges, $\forall v \in V$.

Proof (Induction on i)
Before the ith iteration, $d[v] \leq \min\{w(p)\}$ over all paths p with at most $i-1$ edges.
The relaxation only decreases $d[v]$

The ith iteration considers all paths with $\leq i$ edges when relaxing the edges to v. $\square$



Correctness of BFMS

Theorem

If G, w has no negative weight cycles, then at the end of the BFMS-algorithm $d[v] = \delta(s, v)$.

Proof

- Without negative-weight cycles, shortest paths are always simple.
- Every simple path has at most n vertices and $n - 1$ edges.
- By the previous lemma, the $n - 1$ iterations yield $d[v] \leq \delta(s, v)$.
- By the invariance of the relaxation algorithm $d[v] \geq \delta(s, v)$. $\square$

Correctness of BFMS

Theorem

BFMS will report negative-weight cycles if there exists in G.

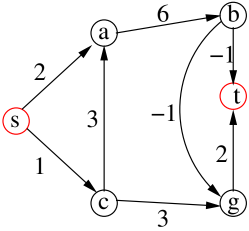
Proof

- Without negative-weight cycles in G, the previous theorem implies $d[v] = \delta(s, v)$, and by triangle inequality $d[v] \leq \delta(s, u) + w(u, v)$, so BFMS won't report a negative cycle if it doesn't exists.
- If there is a negative-weight cycle, then one of its edges can't be relaxed, so BFMS will report correctly. □

Shortest path in a direct acyclic graphs (dags).

Min-cost paths in DAG
INPUT: Edge weighted dag $G = (V, E, w)$,
$|V| = n, |E| = m, w : E \to \mathbb{R}$ together with given $s, t \in V$.
QUESTION: Find a path $P : s \to t$ of minimum total weight.

Notice given a dag $G = (V, E), W$ we wish to find a path P from s to t s.t. $\min_P \sum_{(ij)\in P} w_{ij}$.

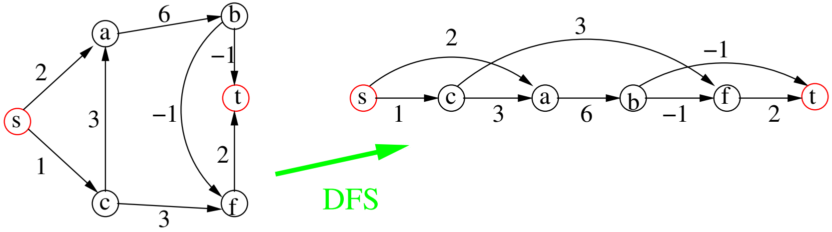


Arranging dag's into a line

A topological sort of a dag $G = (V, E)$ is a linear ordering of its vertices such that if G contains (u, v), then u appears before v in the order.

Topological sort is the easier problem of the graph layout problems.

If a digraph has cycles, there is not linear ordering.



Arranging dag's into a line

Arrange the dag in topological order, so that all edges go left to right.

This can be done in $\Theta(n + m)$ using DFS.

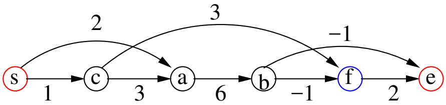
Topological Order $G = (V, E)$
for all $v \in V - \{s\}$ **do**
  Call DFS(G) to compute "preference order"
  Insert v in the front of a listed list L
end for
return layout L

Note DFS takes $\Theta(n + m)$ and the construction of L takes $\Theta(n)$

Shortest path in a direct acyclic graphs (dags).

We want to find shorter distance from s to v. Let $d(v)$ = distance $s \to v$

$d(f) = \min\{d(b)+2, d(c)+3\}$
The schema is based on the topological linearity of G.



Shortest distance in dag G
Initialize $d(s) := 0$ and $\forall v \in V - \{s\}$, $d(v) := \infty$
for all $v \in V - \{s\}$ in linearized order **do**
$d(v) := \min_{(u,v)\in E}\{d(u) + w_{uv}\}$
end for

Complexity? $T(n) = O(n + m)$

All pairs shortest paths: APSP

Given $G = (V, E)$, $|V| = n, |E| = m$ and a weight $w : E \to \mathbb{R}$ we want to determine $\forall u, v \in V$, $\delta(u, v)$.

We assume we can have $w < 0$ but G does not contain negative cycles.

Naive idea: We apply $O(n)$ times BFMS or Dijkstra (if there are not negative weights)

Repetition of BFMS: $O(n^2 m)$

Repetition of Dijkstra: $O(nm \lg n)$ (if Q is implemented by a heap)

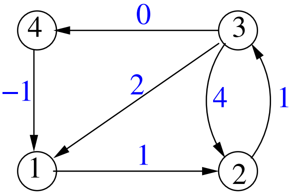
All pairs shortest paths: APSP

- Unlike in the SSSP algorithm that assumed adjacency-list representation of G, for the APSP algorithm we consider the adjacency matrix representation of G.
- For convenience $V = \{1, 2, \ldots n\}$. The $n \times n$ adjacency matrix $W = (w(i,j))$ of G, w:

$$w_{ij} = \begin{cases} 0 & \text{if } i = j \\ w_{ij} & \text{if } (i,j) \in E \\ \infty & \text{if } i \neq j \text{ and } (i,j) \notin E \end{cases}$$

All pairs shortest paths: APSP

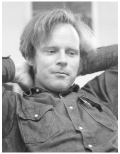
- The input is a $n \times n$ adjacency matrix $W = (w_{ij})$



$$W = \begin{pmatrix} 0 & 1 & \infty & \infty \\ \infty & 0 & 1 & \infty \\ 2 & 4 & 0 & 0 \\ -1 & \infty & \infty & 0 \end{pmatrix}$$

- The output is a $n \times n$ matrix $D = (d_{ij})$, where $d_{ij} = \delta(i, j)$
- For the implementation we also need to compute the set of predecessors matrix Π^k

Bernard-Floyd-Warshall Algorithm

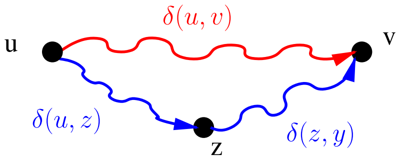
R. Bernard: *Transitivité et connexité* C.R.Aca. Sci. 1959
R. Floyd: *Algorithm 97: Shortest Path*. CACM 1962
S. Warshall: *A theorem on Boolean matrices*. JACM, 1962

The BFW Algorithm used dynamic programming to compare all possible paths between each pair of vertices in G.

The algorithm work in $O(n^3)$ and the number of edges could be $O(n^2)$.

Optimal substructure of APSP

Recall: Triangle inequality
$\delta(u,v) \leq \delta(u,z) + \delta(z,v)$.



- Let $p = p_1, \underbrace{p_2, \ldots, p_{r-1}}_{\text{intermediate v.}}, p_r$ and
- Let $d_{ij}^{(k)}$ be the shortest $i \rightsquigarrow j$ s.t. the intermediate vertices are in $\{1, \ldots, k\}$.
- So if $k = 0$, then $d_{ij}^{(0)} = w_{ij}$.

The recurrence

Let p a shortest path $i \rightsquigarrow j$ with value $d_{ij}^{(k)}$

- If k is not an intermediate vertex of p, then $d_{ij}^{(k)} = d_{ij}^{(k-1)}$
- If k is an intermediate vertex of p, then
  $p = \underbrace{(i, \ldots, k)}_{p_1} \cup \underbrace{(k, \ldots, j)}_{p_2}$
- By triangle inequality p_1 is a shortest path $i \rightsquigarrow k$ and p_2 is a shortest path $k \rightsquigarrow j$.

Therefore $d_{ij}^{(k)} = \begin{cases} w_{ij} & \text{if } k = 0 \\ \min\{d_{ij}^{(k-1)}, d_{ik}^{(k-1)} + d_{kj}^{(k-1)}\} & \text{if } k \geq 1 \end{cases}$

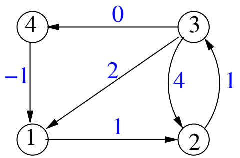
Bottom-up BFW-algorithm

Given $G = (V, E), w : E \to \mathbb{Z}$ without negative cycles, the following DP algo. computes $d_{ij}^{(n)}$, $\forall i, j \in V$:

BFW $W = (w_{ij})$
for $k = 1$ to n **do**
  for $i = 1$ to n **do**
    for $j = 1$ to n **do**
      $d_{ij}^{(k)} = \min\{d_{ij}^{(k-1)}, d_{ik}^{(k-1)} + d_{kj}^{(k-1)}\}$
    end for
  end for
end for
return $d^{(n)}$

- Time complexity: $T(n) = O(n^3), S(n) = O(n^3)$ but $S(n)$ can be lowered to $O(n^2)$ How?
- Correctness follows from the recurrence argument.

Example



$$D^{(0)} = \begin{pmatrix} 0 & 1 & \infty & \infty \\ \infty & 0 & 1 & \infty \\ 2 & 4 & 0 & 0 \\ -1 & \infty & \infty & 0 \end{pmatrix} \quad D^{(1)} = \begin{pmatrix} 0 & 1 & \infty & \infty \\ \infty & 0 & 1 & \infty \\ 2 & 3 & 0 & 0 \\ -1 & 0 & \infty & 0 \end{pmatrix}$$

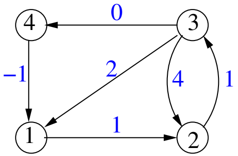
$$D^{(2)} = \begin{pmatrix} 0 & 1 & 2 & \infty \\ \infty & 0 & 1 & \infty \\ 2 & 3 & 0 & 0 \\ -1 & 0 & 1 & 0 \end{pmatrix} \quad D^{(3)} = \begin{pmatrix} 0 & 1 & 2 & 2 \\ 3 & 0 & 1 & 1 \\ 2 & 3 & 0 & 0 \\ -1 & 0 & 1 & 0 \end{pmatrix} \quad D^{(4)} = \begin{pmatrix} 0 & 1 & 2 & 2 \\ 0 & 0 & 1 & 1 \\ -1 & 0 & 0 & 0 \\ -1 & 0 & 1 & 0 \end{pmatrix}$$

For instance, $d^2_{3,2} = 3 \to 1 \to 2$ (using vertex 2)
$d^4_{3,1} = 3 \to 4 \to 1$ (using all vertices)

Example

$$D^{(0)} = \begin{pmatrix} 0 & 1 & \infty & \infty \\ \infty & 0 & 1 & \infty \\ 2 & 4 & 0 & 0 \\ -1 & \infty & \infty & 0 \end{pmatrix} \qquad \Pi^{(0)} = \begin{pmatrix} \text{NIL} & 1 & \text{NIL} & \text{NIL} \\ \text{NIL} & \text{NIL} & 2 & \text{NIL} \\ 3 & 3 & \text{NIL} & 3 \\ 4 & \text{NIL} & \text{NIL} & \text{NIL} \end{pmatrix}$$

$$D^{(1)} = \begin{pmatrix} 0 & 1 & \infty & \infty \\ \infty & 0 & 1 & \infty \\ 2 & 3 & 0 & 0 \\ -1 & 0 & \infty & 0 \end{pmatrix} \qquad \Pi^{(1)} = \begin{pmatrix} \text{NIL} & 1 & \text{NIL} & \text{NIL} \\ \text{NIL} & \text{NIL} & 2 & \text{NIL} \\ 3 & 1 & \text{NIL} & 3 \\ 4 & 1 & \text{NIL} & \text{NIL} \end{pmatrix}$$



$$D^{(2)} = \begin{pmatrix} 0 & 1 & 2 & \infty \\ \infty & 0 & 1 & \infty \\ 2 & 3 & 0 & 0 \\ -1 & 0 & 1 & 0 \end{pmatrix} \qquad \Pi^{(2)} = \begin{pmatrix} \text{NIL} & 1 & \text{NIL} & \text{NIL} \\ \text{NIL} & \text{NIL} & 1 & \text{NIL} \\ 3 & 1 & \text{NIL} & 3 \\ 4 & 1 & \text{NIL} & \text{NIL} \end{pmatrix}$$

$$D^{(3)} = \begin{pmatrix} 0 & 1 & 2 & 2 \\ 3 & 0 & 1 & 1 \\ 2 & 3 & 0 & 0 \\ -1 & 0 & 1 & 0 \end{pmatrix} \qquad \Pi^{(3)} = \begin{pmatrix} \text{NIL} & 1 & 2 & 3 \\ 3 & \text{NIL} & 2 & 3 \\ 3 & 1 & \text{NIL} & 3 \\ 4 & 1 & 2 & \text{NIL} \end{pmatrix}$$

Constructing the shortest path

- We want to construct the matrix $\Pi = (\pi_{ij})$, where $\pi_{ij} =$ predecessor of j in shortest $i \rightsquigarrow j$,
- we define a sequence of matrices $\Pi^{(0)}, \ldots, \Pi^{(n)}$ s.t. $\Pi^{(k)} = (\pi_{ij}^{(k)})$, i.e. the matrix of last predecessors in the shortest path $i \rightsquigarrow j$, which uses only vertices in $\{1, \ldots, k\}$.
- If $k = 0$: $\pi_{ij}^{(k)} = \begin{cases} NIL & \text{if } i = j \text{ or } w_{ij} = \infty, \\ i & \text{if } i \neq j \text{ and } w_{ij} \neq \infty. \end{cases}$
- For $k \geq 1$ we get the recurrence:

$$\pi_{ij}^{(k)} = \begin{cases} \pi_{ij}^{(k-1)} & \text{if } d_{ij}^{(k-1)} \leq d_{ik}^{(k-1)} + d_{kj}^{(k-1)}, \\ \pi_{kj}^{(k-1)} & \text{otherwise.} \end{cases}$$

BFW with paths

BFW W
$d^{(0)} = W$
for $k = 1$ to n **do**
  for $i = 1$ to n **do**
    for $j = 1$ to n **do**
      if $d_{ij}^{(k)} \leq d_{ij}^{(k-1)}, d_{ik}^{(k-1)} + d_{kj}^{(k-1)}$ **then**
        $d_{ij}^{(k)} = d_{ij}^{(k-1)}$
        $\Pi_{ij}^{(k)} = \Pi_{ij}^{(k-1)}$
      else
        $d_{ij}^{(k)} = d_{ik}^{(k-1)} + d_{kj}^{(k-1)}$
        $\Pi_{ij}^{(k)} = \Pi_{kj}^{(k-1)}$
      end if
    end for
  end for
end for
return $d^{(n)}$

Complexity: $T(n) = O(n^3)$

Conclusions

SSSP

	Dijkstra	BFMS
$w \geq 0$	$O(m \lg n)$	$O(nm)$
$w \in \mathbb{R}$	NO	$O(nm)$

APSP

	Dijkstra	BFMS	BFW
$w \geq 0$	$O(nm \lg n)$	$O(n^2 m)$	$O(n^3)$
$w \in \mathbb{R}$	NO	$O(n^2 m)$	$O(n^3)$

Conclusions: Remarks for APSP algorithms

- Note that for sparse graphs with $m = O(n)$, Dijkstra is the most efficient: $O(n^2 \lg n)$, while for dense graphs with $m = O(n^2)$, BFW is the best complexity.
- There exists an algorithm for the APSP problem by D. Johnson (1978) that works in $O(n^2 \lg n)$ for sparse graphs with negative edges. It uses Dijkstra and BFMS as functions.
- For graphs that are undirected and without weights, there is an algorithm by R.Seidel that works in $O(n^\omega \lg n)$, where n^ω is the complexity of multiplying two $n \times n$ matrices, which of as today is $\omega \sim 2.3$.
- For further reading on shortest paths, see chapters 24 and 25 of CLRS Introduction to Algorithms.