CS 577 - All Pair Shortest Paths

Manolis Vlatakis

Department of Computer Sciences
University of Wisconsin – Madison

Fall 2024

WISCONSIN
UNIVERSITY OF WISCONSIN–MADISON

DIJKSTRA VS BELLMAN-FORD

- **Dijkstra Algorithm** is faster but does not work with negative weights.
    - Assumes that distances do not decrease along the shortest path.
- **Bellman-Ford Algorithm** works with negative weights.
    - Distances can decrease along the shortest path.
    - The "last" vertex can have a shorter distance from the start.

QUESTIONS - EXERCISES

- Negative weights → add a large number → positive weights → Dijkstra's algorithm?
- Does BFS compute shortest paths when edges have unit lengths?
- When there are non-negative weights, can a shortest path tree and a shortest path graph have no common edge?
- **Bottleneck Shortest Paths:**
  - Path cost $c(p) = \max_{e \epsilon p} w(e)$
  - Shortest paths with bottleneck cost
  - Modified Dijkstra solves Bottleneck Shortest Paths (even with negative weights):

$$D[u] = \min\{D[v], \max(D[u], w(v, u))\}$$

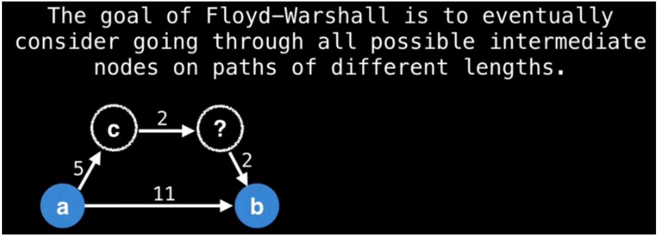
FLOYD-WARSHALL

ALL-PAIRS SHORTEST PATHS

- Compute distance $d(v,u)$ and shortest path $v-u$ path for all pairs $(v,u) \in V \times V$.
- Algorithm for shortest path from a single source for each $s \in V$:
    - Negative weights: Bellman-Ford, time $\Theta(n^2 m)$.
    - Non-negative weights: Dijkstra, time $\Theta(nm + n^2 \log n)$.
- For negative weights: Floyd-Warshall in time $\Theta(n^3)$.
- Solution Representation:
    - Distances: matrix $D[1..n][1..n]$
    - Shortest paths: predecessor matrices.

FLOYD-WARSHALL ALGORITHM

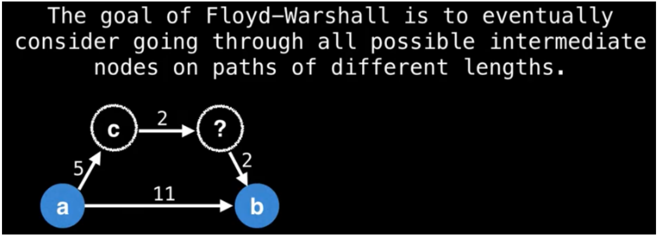
INTUITION



The goal of Floyd–Warshall is to eventually consider going through all possible intermediate nodes on paths of different lengths.

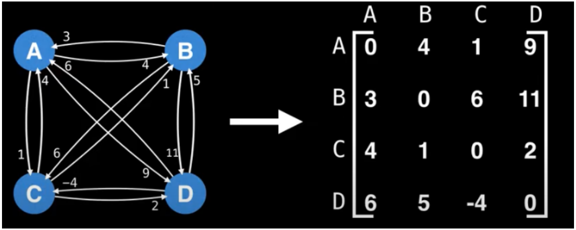
FLOYD-WARSHALL ALGORITHM

INTUITION



The goal of Floyd-Warshall is to eventually consider going through all possible intermediate nodes on paths of different lengths.

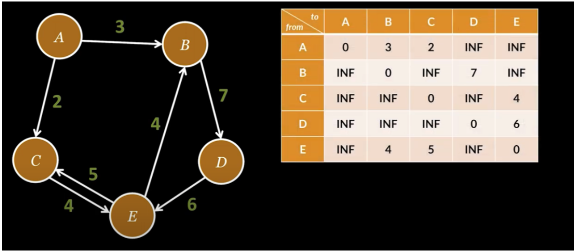
We need quick access to neighbors



$$\begin{array}{c|cccc} & A & B & C & D \\ A & 0 & 4 & 1 & 9 \\ B & 3 & 0 & 6 & 11 \\ C & 4 & 1 & 0 & 2 \\ D & 6 & 5 & -4 & 0 \end{array}$$

FLOYD-WARSHALL ALGORITHM

INITIALIZATION



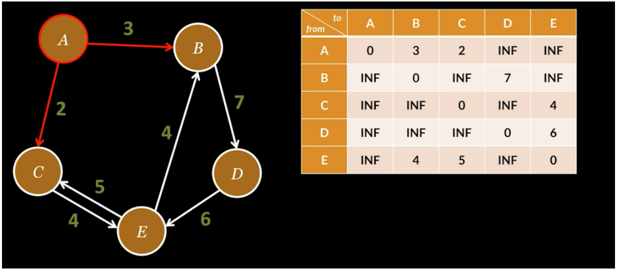
to from	A	B	C	D	E
A	0	3	2	INF	INF
B	INF	0	INF	7	INF
C	INF	INF	0	INF	4
D	INF	INF	INF	0	6
E	INF	4	5	INF	0

FLOYD-WARSHALL ALGORITHM

INTUITION

In words: We check for every pair (v_i, v_j) what is the optimal path?

We request one of the $v_1, \ldots, v_k$ to be an intermediate node.

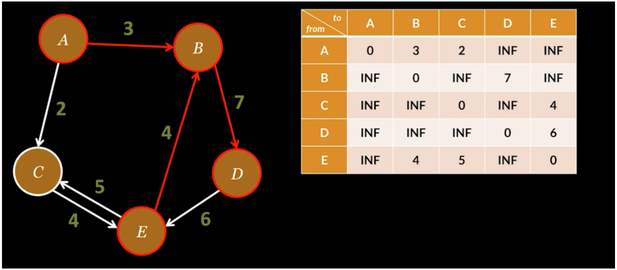


to / from	A	B	C	D	E
A	0	3	2	INF	INF
B	INF	0	INF	7	INF
C	INF	INF	0	INF	4
D	INF	INF	INF	0	6
E	INF	4	5	INF	0

FLOYD-WARSHALL ALGORITHM

INTUITION

In words: We check for every pair (v_i, v_j) what is the optimal path?
We request one of the $v_1, \ldots, v_k$ to be an intermediate node.



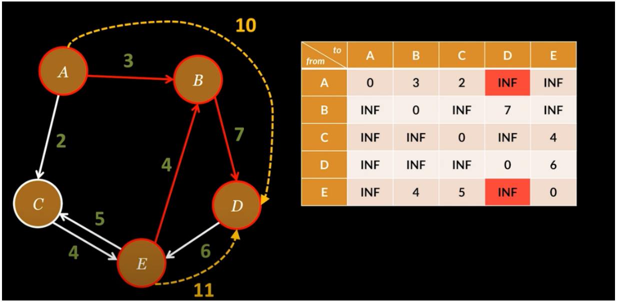
to from	A	B	C	D	E
A	0	3	2	INF	INF
B	INF	0	INF	7	INF
C	INF	INF	0	INF	4
D	INF	INF	INF	0	6
E	INF	4	5	INF	0

FLOYD-WARSHALL ALGORITHM

INTUITION

In words: We check for every pair (v_i, v_j) what is the optimal path?

We request one of the $v_1, \ldots, v_k$ to be an intermediate node.

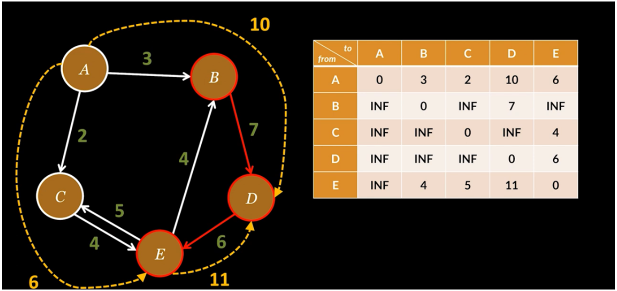


to from	A	B	C	D	E
A	0	3	2	INF	INF
B	INF	0	INF	7	INF
C	INF	INF	0	INF	4
D	INF	INF	INF	0	6
E	INF	4	5	INF	0

FLOYD-WARSHALL ALGORITHM

INTUITION

In words: We check for every pair (v_i, v_j) what is the optimal path?
We request one of the $v_1, \ldots, v_k$ to be an intermediate node.



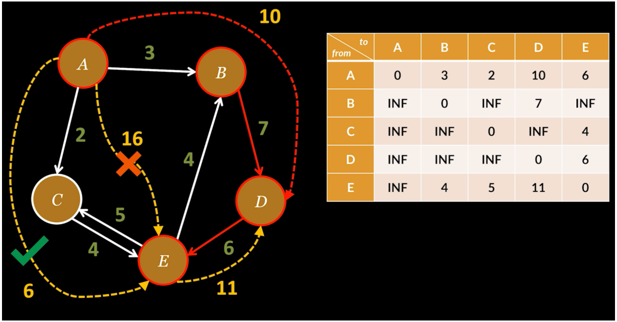
to from	A	B	C	D	E
A	0	3	2	10	6
B	INF	0	INF	7	INF
C	INF	INF	0	INF	4
D	INF	INF	INF	0	6
E	INF	4	5	11	0

FLOYD-WARSHALL ALGORITHM

INTUITION

In words: We check for every pair (v_i, v_j) what is the optimal path?

We request one of the $v_1, \ldots, v_k$ to be an intermediate node.



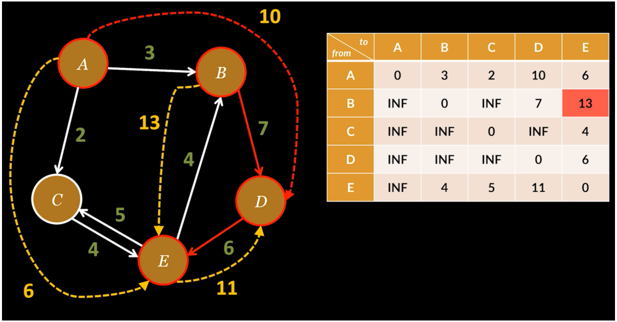
to / from	A	B	C	D	E
A	0	3	2	10	6
B	INF	0	INF	7	INF
C	INF	INF	0	INF	4
D	INF	INF	INF	0	6
E	INF	4	5	11	0

FLOYD-WARSHALL ALGORITHM

INTUITION

In words: We check for every pair (v_i, v_j) what is the optimal path?

We request one of the $v_1, \dots, v_k$ to be an intermediate node.



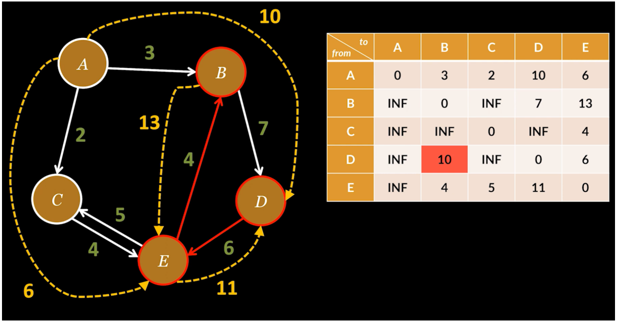
from \ to	A	B	C	D	E
A	0	3	2	10	6
B	INF	0	INF	7	13
C	INF	INF	0	INF	4
D	INF	INF	INF	0	6
E	INF	4	5	11	0

FLOYD-WARSHALL ALGORITHM

INTUITION

In words: We check for every pair (v_i, v_j) what is the optimal path?

We request one of the $v_1, \dots, v_k$ to be an intermediate node.



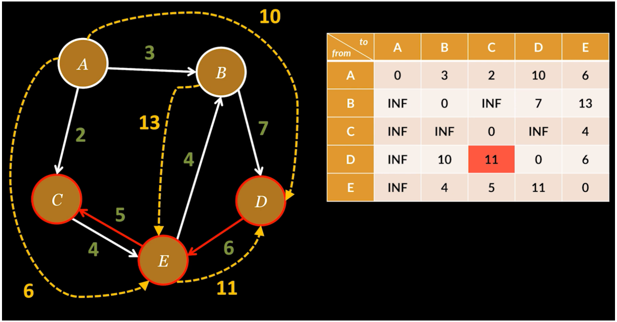
to from	A	B	C	D	E
A	0	3	2	10	6
B	INF	0	INF	7	13
C	INF	INF	0	INF	4
D	INF	10	INF	0	6
E	INF	4	5	11	0

FLOYD-WARSHALL ALGORITHM

INTUITION

In words: We check for every pair (v_i, v_j) what is the optimal path?

We request one of the $v_1, \ldots, v_k$ to be an intermediate node.



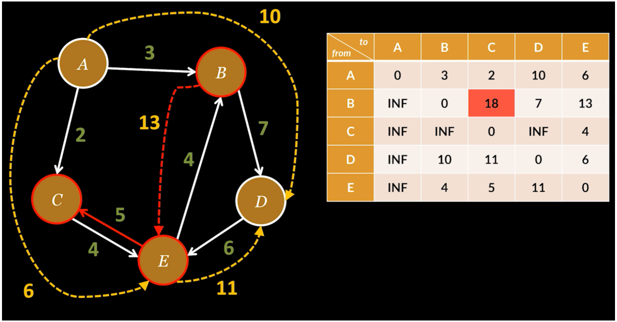
to / from	A	B	C	D	E
A	0	3	2	10	6
B	INF	0	INF	7	13
C	INF	INF	0	INF	4
D	INF	10	11	0	6
E	INF	4	5	11	0

FLOYD-WARSHALL ALGORITHM

INTUITION

In words: We check for every pair (v_i, v_j) what is the optimal path?

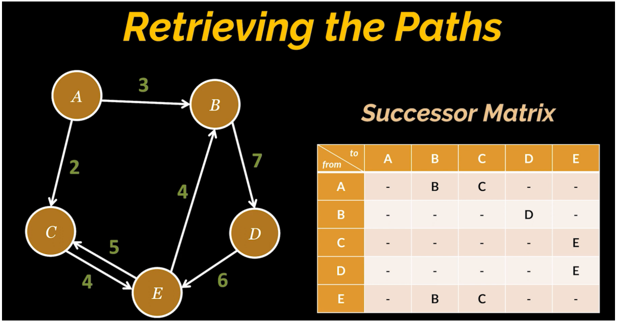
We request one of the $v_1, \ldots, v_k$ to be an intermediate node.



to / from	A	B	C	D	E
A	0	3	2	10	6
B	INF	0	18	7	13
C	INF	INF	0	INF	4
D	INF	10	11	0	6
E	INF	4	5	11	0

FLOYD-WARSHALL ALGORITHM

COMPUTE FULL PATH



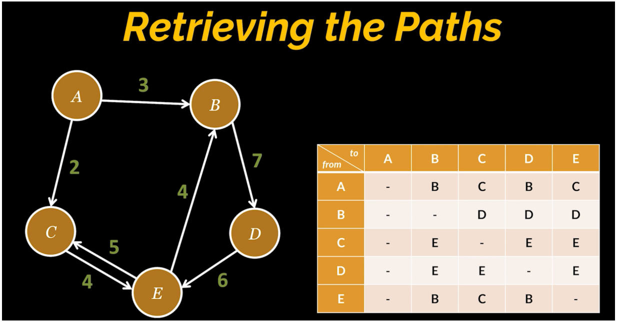
Retrieving the Paths

Successor Matrix

to from	A	B	C	D	E
A	-	B	C	-	-
B	-	-	-	D	-
C	-	-	-	-	E
D	-	-	-	-	E
E	-	B	C	-	-

FLOYD-WARSHALL ALGORITHM

COMPUTE FULL PATH

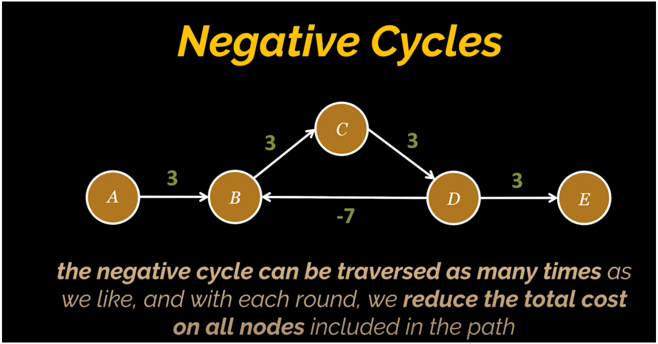


Retrieving the Paths

to / from	A	B	C	D	E
A	-	B	C	B	C
B	-	-	D	D	D
C	-	E	-	E	E
D	-	E	E	-	E
E	-	B	C	B	-

FLOYD-WARSHALL ALGORITHM

NEGATIVE CYCLES

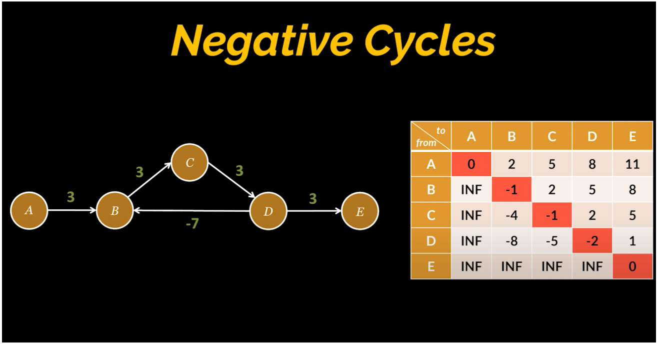


Negative Cycles

the negative cycle can be traversed as many times *as we like, and with each round, we* ***reduce the total cost on all nodes*** *included in the path*

FLOYD-WARSHALL ALGORITHM

NEGATIVE CYCLES

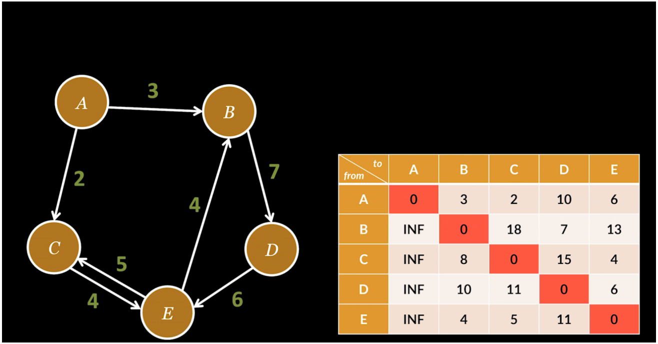


Negative Cycles

to / from	A	B	C	D	E
A	0	2	5	8	11
B	INF	-1	2	5	8
C	INF	-4	-1	2	5
D	INF	-8	-5	-2	1
E	INF	INF	INF	INF	0

FLOYD-WARSHALL ALGORITHM

NEGATIVE CYCLES



from \ to	A	B	C	D	E
A	0	3	2	10	6
B	INF	0	18	7	13
C	INF	8	0	15	4
D	INF	10	11	0	6
E	INF	4	5	11	0

FLOYD-WARSHALL ALGORITHM

IMPLEMENTATION

- Consider a graph $G(V, E, w)$ with edge weights.
- Graph representation with adjacency matrix:

$$w(v_i, v_j) = \begin{cases} 0 & \text{if } v_i = v_j \\ w(v_i, v_j) & \text{if } (v_i, v_j) \in E \\ \infty & \text{otherwise} \end{cases}$$

- Compute distance $d(v_i, v_j)$ from $d(v_i, v_k), d(v_k, v_j)$ for all $k \in V \setminus \{v_i, v_j\}$:

$$d(v_i, v_j) = \min\{w(v_i, v_j), d(v_i, v_k) + d(v_k, v_j)\}$$

In words: We check for every pair (v_i, v_j) what is the optimal path?

We request one of the $v_1, \dots, v_k$ to be an intermediate node.

- Negative cycle: $d(v_i, v_j) \to d(v_i, v_k)$
- Dynamic Programming: Compute all distances systematically bottom-up.

JOHNSON'S ALGORITHM

FASTER THAN FLOYD-WARSHALL

IS IT POSSIBLE?

If all weights are positive, then we can run $n = |V|$ times Dijkstra's algorithm!

$$\text{RUN-TIME}(\text{APSPs} \mid n\text{-Dijkstra}) = n \times O(m + n \log n) = O(mn + n^2 \log n)$$

- The maximum number of edges is $m = \binom{n}{2} = \Theta(n^2)$.

$$\text{RUN-TIME}(\text{APSPs} \mid n\text{-Dijkstra}) = O(mn) = O(n^3)$$

FASTER THAN FLOYD-WARSHALL

IS IT POSSIBLE?

If all weights are positive, then we can run $n = |V|$ times Dijkstra's algorithm!

$$\textsc{Run-Time}(\text{APSPs} \mid n\text{-Dijkstra}) = n \times O(m + n \log n) = O(mn + n^2 \log n)$$

- The maximum number of edges is $m = \binom{n}{2} = \Theta(n^2)$.

$$\textsc{Run-Time}(\text{APSPs} \mid n\text{-Dijkstra}) = O(mn) = O(n^3)$$

- If the number of edges is $m = o(n^2)$, then:

$$\textsc{Run-Time}(\text{APSPs} \mid n\text{-Dijkstra}) = O(mn) = o(n^3)$$

FASTER THAN FLOYD-WARSHALL

IS IT POSSIBLE?

🤔What can we do for the case of negatively-weighted graphs?

FASTER THAN FLOYD-WARSHALL

IS IT POSSIBLE?

🤔What can we do for the case of negatively-weighted graphs?

- Check the existence of negative cycles in $O(mn)$ time.
  🤔**How?**

FASTER THAN FLOYD-WARSHALL

IS IT POSSIBLE?

🤔What can we do for the case of negatively-weighted graphs?

- Check the existence of negative cycles in $O(mn)$ time.
  🤔**How?** Using the Bellman-Ford algorithm.

Faster than Floyd-Warshall

Is it possible?

🤔What can we do for the case of negatively-weighted graphs?

- Check the existence of negative cycles in $O(mn)$ time. 🤔**How?** Using the Bellman-Ford algorithm.
- 🤔Can we adjust the weight of every edge to run Dijkstra's algorithm n times?

FASTER THAN FLOYD-WARSHALL

IS IT POSSIBLE?

🤔What can we do for the case of negatively-weighted graphs?

- Check the existence of negative cycles in $O(mn)$ time.
  🤔**How?** Using the Bellman-Ford algorithm.
- 🤔Can we adjust the weight of every edge to run Dijkstra's algorithm n times? No.

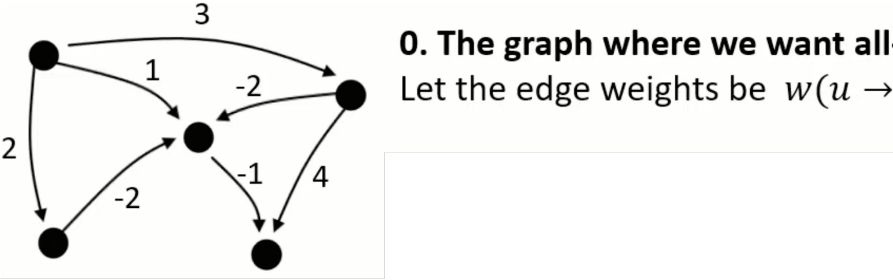
Faster than Floyd-Warshall

Is it possible?

🤔What can we do for the case of negatively-weighted graphs?

- Check the existence of negative cycles in $O(mn)$ time. 🤔**How?** Using the Bellman-Ford algorithm.
- 🤔Can we adjust the weight of every edge to run Dijkstra's algorithm n times? No.
- 🤔Any extra ideas??

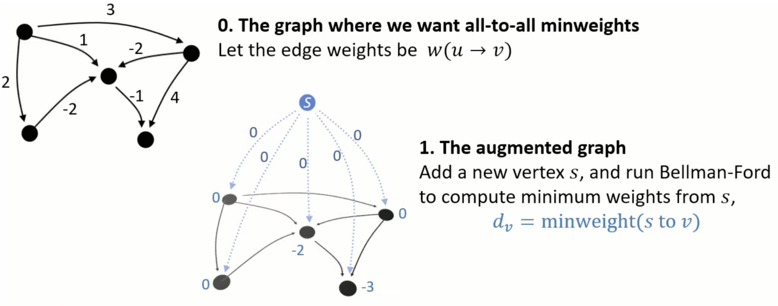
JOHNSON'S ALGORITHM



0. The graph where we want all

Let the edge weights be $w(u \to$

JOHNSON'S ALGORITHM



0. The graph where we want all-to-all minweights

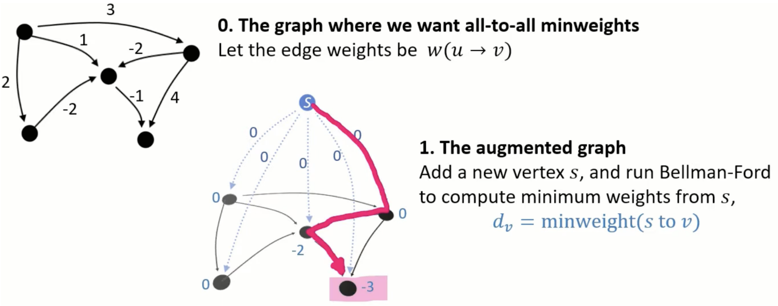
Let the edge weights be $w(u \to v)$

1. The augmented graph

Add a new vertex s, and run Bellman-Ford to compute minimum weights from s,

$$d_v = \text{minweight}(s \text{ to } v)$$

JOHNSON'S ALGORITHM



0. The graph where we want all-to-all minweights

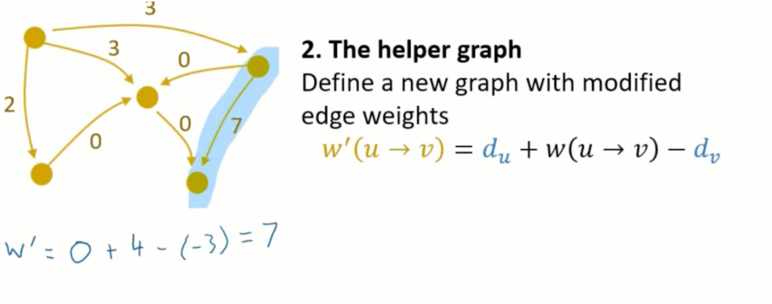
Let the edge weights be $w(u \to v)$

1. The augmented graph

Add a new vertex s, and run Bellman-Ford to compute minimum weights from s,

$$d_v = \text{minweight}(s \text{ to } v)$$

JOHNSON'S ALGORITHM



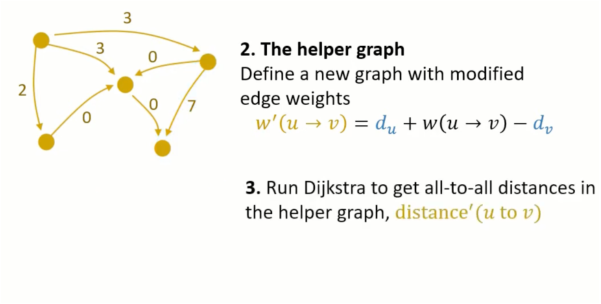
2. The helper graph

Define a new graph with modified edge weights

$$w'(u \to v) = d_u + w(u \to v) - d_v$$

$w' = 0 + 4 - (-3) = 7$

JOHNSON'S ALGORITHM

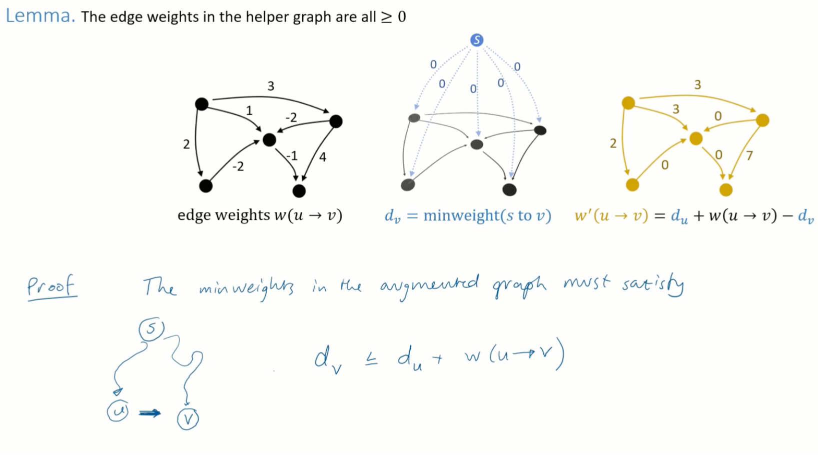


2. The helper graph
Define a new graph with modified edge weights

$$w'(u \to v) = d_u + w(u \to v) - d_v$$

3. Run Dijkstra to get all-to-all distances in the helper graph, $\text{distance}'(u \text{ to } v)$

JOHNSON'S ALGORITHM



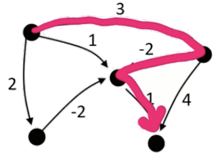
Lemma. The edge weights in the helper graph are all ≥ 0

edge weights $w(u \to v)$ $d_v = \text{minweight}(s \text{ to } v)$ $w'(u \to v) = d_u + w(u \to v) - d_v$

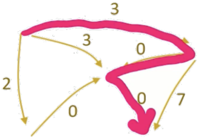
Proof The minweights in the augmented graph must satisfy

$$d_v \leq d_u + w(u \to v)$$

Johnson’s Algorithm



edge weights $w(u \to v)$



$w'(u \to v) = d_u + w(u \to v) - d_v$

JOHNSON'S ALGORITHM

- All-pairs shortest paths for all vertex pairs in sparse graphs with negative weights:
  - Transform negative weights to non-negative without changing shortest paths.
- Algorithm for graph $G(V, E, w)$:
  - Add a new vertex s connected to each $u \in V$ with zero-weight edges.
  - Bellman-Ford for G' with s as the source.
  - If no negative cycle, compute new (non-negative) weights:

$$\hat{w}(u, v) = w(v, u) + h(v) - h(u)$$

  - Use Dijkstra on $G(V, E, \hat{w})$ for each starting vertex u.

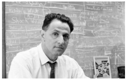
SUMMARY

- Single-source shortest paths from an initial vertex s:
  - Negative weights: Bellman-Ford in time $\Theta(nm)$.
  - DAGs with negative weights in time $\Theta(m+n)$.
  - Non-negative weights: Dijkstra in time $\Theta(m+n\log n)$.
- All-pairs shortest paths:
  - Negative weights: Floyd-Warshall in time $\Theta(n^3)$.
  - Sparse graphs with (possibly) negative weights $m=o(n^2)$:
    - n applications of Dijkstra in time $\Theta(nm+n^2\log n)$.
    - With negative weights, Johnson's algorithm for non-negative transformation.

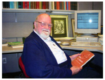
APPENDIX

REFERENCES

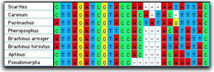
Image Sources I



https://medium.com/neurosapiens/2-dynamic-programming-9177012dcdd



https://angelberh7.wordpress.com/2014/10/08/biografia-de-lester-randolph-ford-jr/



Scarites	C	T	T	A	G	A	T	C	G	T	A	C	C	A	A	-	-	-	A	A	T	A	T	T	A	C
Carenum	C	T	T	A	G	A	T	C	G	T	A	C	C	A	C	A	-	T	A	C	-	T	T	T	A	C
Pasimachus	A	T	T	A	G	A	T	C	G	T	A	C	C	A	C	T	A	T	A	A	G	T	T	T	A	C
Pheropsophus	C	T	T	A	G	A	T	C	G	T	T	C	C	A	C	-	-	-	A	C	A	T	A	T	A	C
Brachinus armiger	A	T	T	A	G	A	T	C	G	T	A	C	C	A	C	-	-	-	A	T	A	T	A	T	T	C
Brachinus hirsutus	A	T	T	A	G	A	T	C	G	T	A	C	C	A	C	-	-	-	A	T	A	T	A	T	A	C
Aptinus	C	T	T	A	G	A	T	C	G	T	A	C	C	A	C	-	-	-	A	C	A	A	T	T	A	C
Pseudomorpha	C	T	T	A	G	A	T	C	G	T	A	C	C	-	-	-	-	-	A	C	A	A	A	T	A	C

http://www.sequence-alignment.com/

https://medium.com/koderunners/genetic-algorithm-part-3-knapsack-problem-b59035

WISCONSIN
UNIVERSITY OF WISCONSIN–MADISON

https://brand.wisc.edu/web/logos/

IMAGE SOURCES II

https://www.pngfind.com/mpng/mTJmbx_spongebob-squarepants-png-image-spongebob-cartoo

https://www.pngfind.com/mpng/xhJRmT_cheshire-cat-vintage-drawing-alice-in-wonderland