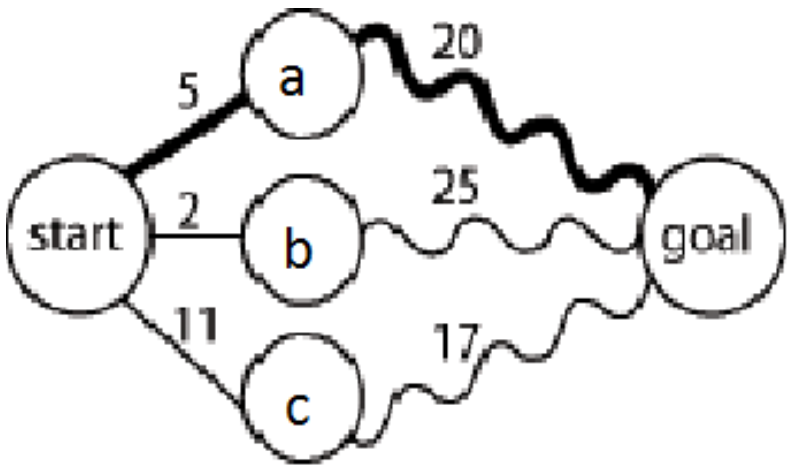

THE UNIVERSITY OF
AUCKLAND
Te Whare Wānanga o Tāmaki Makaurau
NEW ZEALAND

Algorithms and Data Structures

Lecture 20 Shortest Paths, Revisited

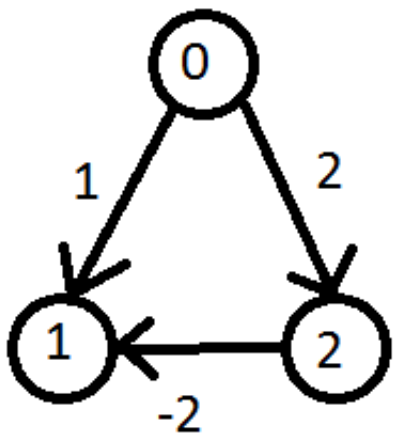
Jiamou Liu
The University of Auckland



Shortest Path with Potentially Negative Edges

Recap: Being Greedy is Risky

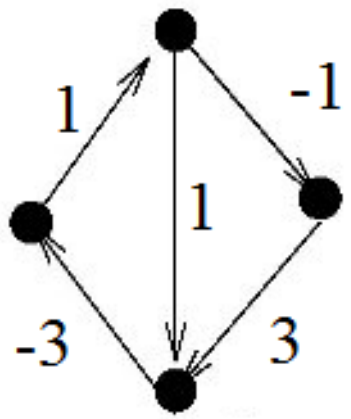
Dijkstra's algorithm does not work for weighted graph where the weights could be negative.



Greedy choice does not work here.

Shortest Path with Potentially Negative Edges

Observation



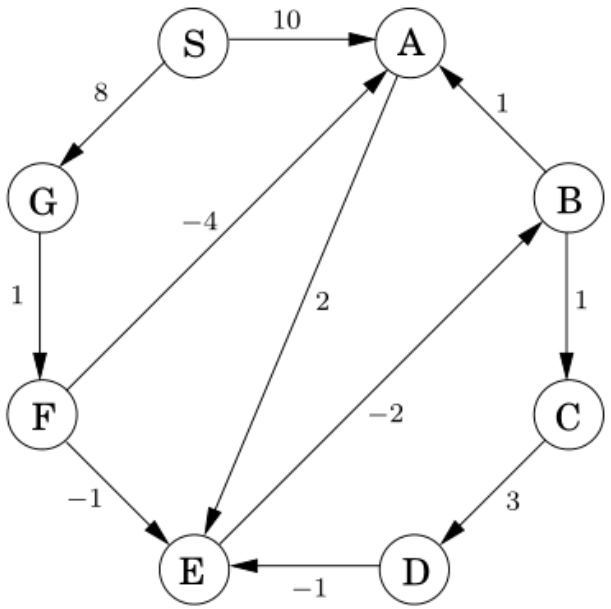
- If there is a "negative cycle" (a cycle whose total edge weight is negative), then the problem does not make sense.
- For simplicity let's first assume there is no negative cycle.

Paths with Bounded Length

Notation

Let $d_k(u)$ denote the length of the shortest path from S to u that uses at most k edges.

Example:



$d_0(A) = \infty, d_1(A) = 10, d_2(A) = 10, d_3(A) = 3$
$d_0(E) = d_1(E) = \infty, d_2(E) = 12, d_3(E) = 8, d_4(E) = 7$

Paths with Bounded Length

Fact

Suppose G does not contain a negative cycle. Then the distance from S to u is $d_{n-1}(u)$ for every $u \in V$.

Paths with Bounded Length

Fact

Suppose G does not contain a negative cycle. Then the distance from S to u is $d_{n-1}(u)$ for every $u \in V$.

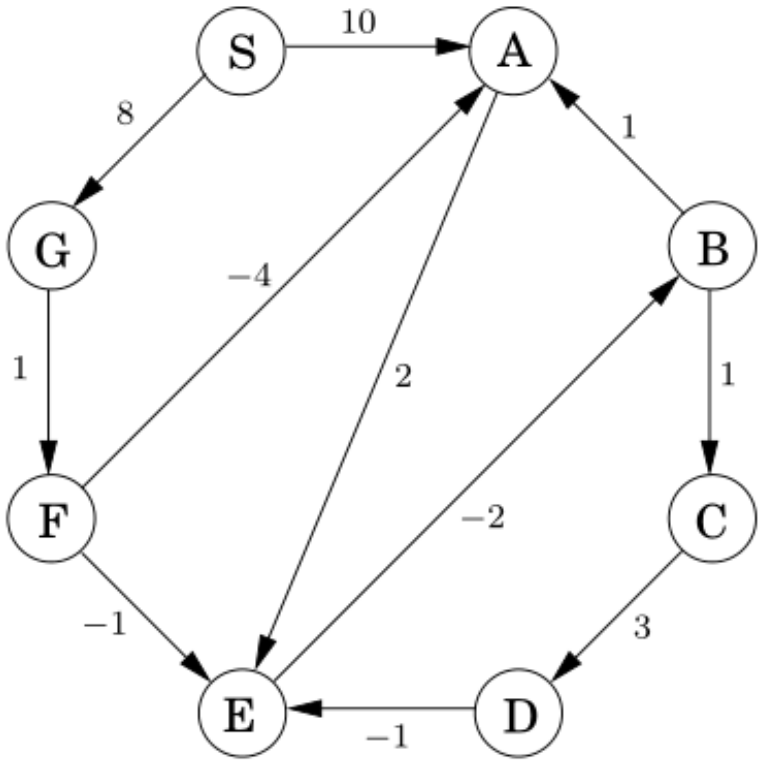
Proof. Take a path from S to u of length $> n - 1$.

- This path must contain $> n$ nodes.
- This means that some node appears more than once, i.e., a cycle exists in the path.
- But then we can reduce the length by removing this cycle.

Therefore this path cannot be the shortest path from S to u. □

Computing Distances

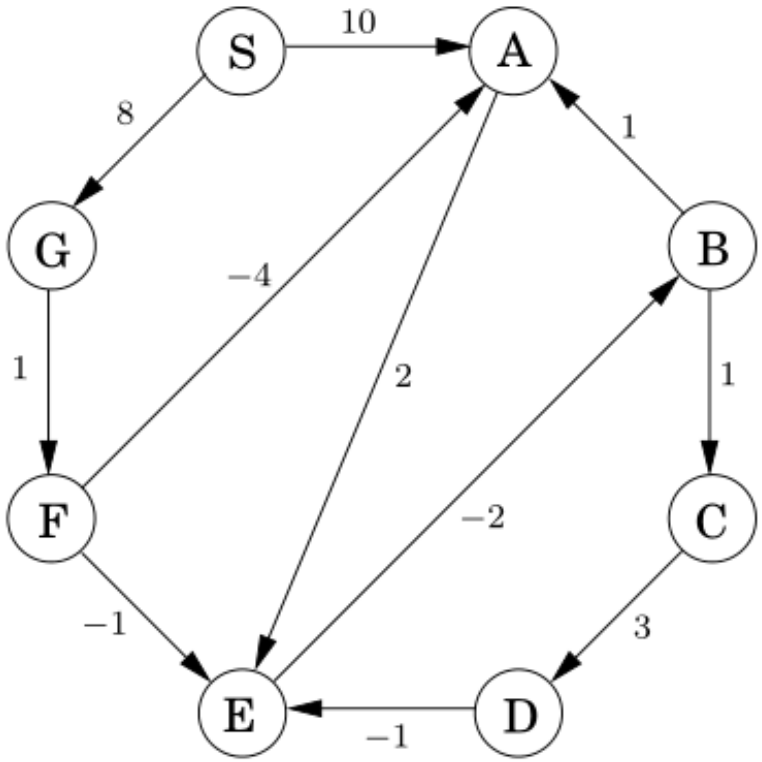
Computing distances is reduced to computing $d_{n-1}(u)$.



	k							
Node	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

Computing Distances

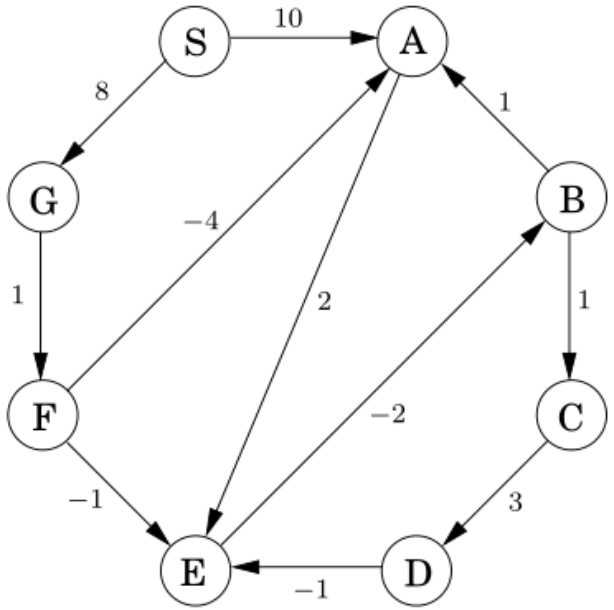
Computing distances is reduced to computing $d_{n-1}(u)$.



distances

	k							
Node	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

Computing Distances

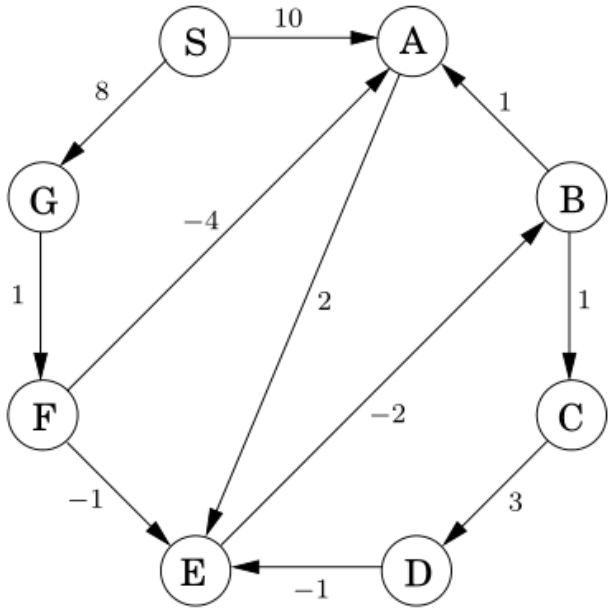


Node	0
S	0
A	∞
B	∞
C	∞
D	∞
E	∞
F	∞
G	∞

Computing $d_0(u)$

- $d_0(s) = 0$
- $d_0(u) = \infty$ for every $u \neq s$

Computing Distances

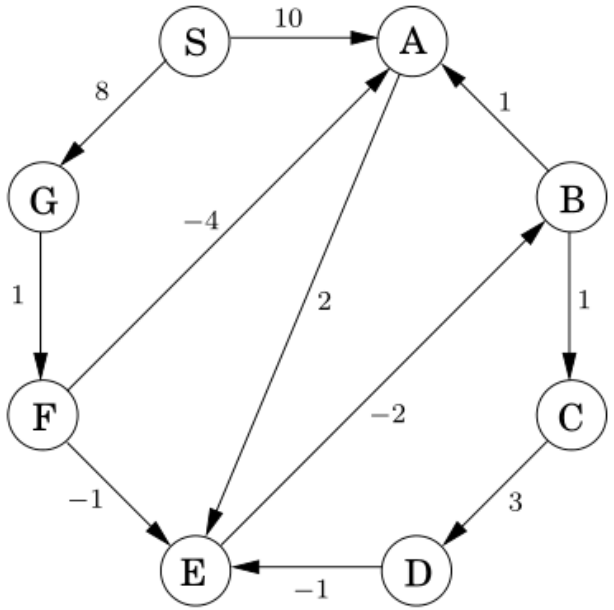


Node	0	1
S	0	0
A	∞	10
B	∞	∞
C	∞	∞
D	∞	∞
E	∞	∞
F	∞	∞
G	∞	8

Computing $d_1(u)$

- $d_1(s) = 0$
- $d_1(u) = w(s, u)$ for every out-neighbour u of s
- $d_1(u) = \infty$ for all other nodes

Computing Distances

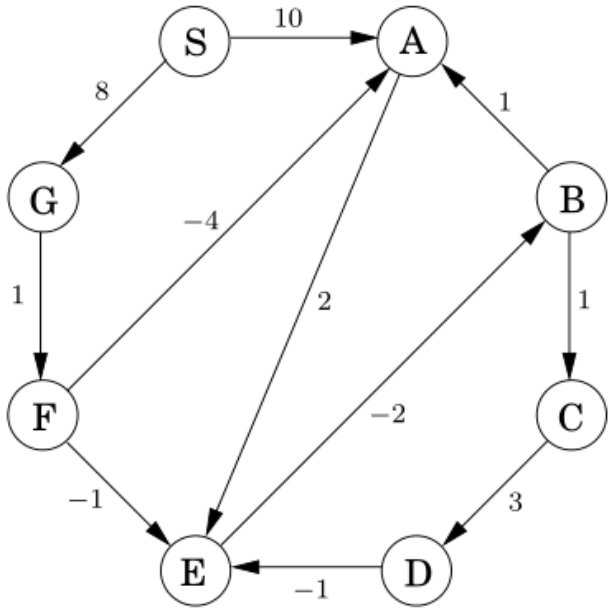


Node	0	1	2
S	0	0	0
A	∞	10	10
B	∞	∞	∞
C	∞	∞	∞
D	∞	∞	∞
E	∞	∞	12
F	∞	∞	9
G	∞	8	8

Computing $d_2(u)$

How can we tell that $d_2(E) = 12$?
$d_1(A) = 10$ and $w(A, E) = 2$
So $d_2(E) = d_1(A) + w(A, E)$

Computing Distances

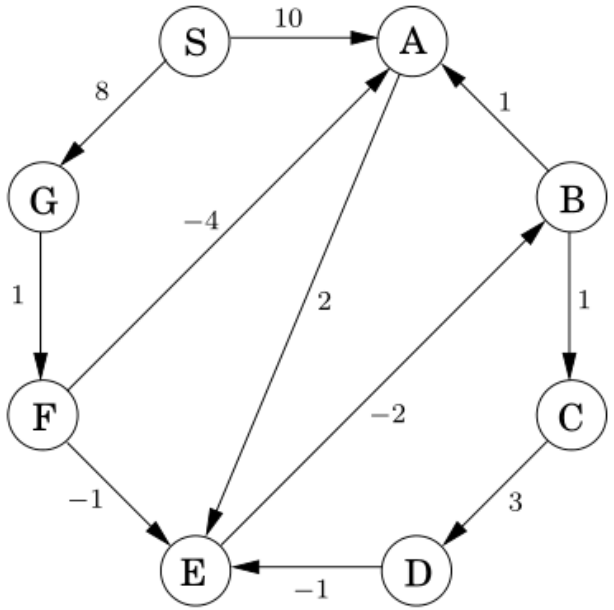


Node	0	1	2	3
S	0	0	0	0
A	∞	10	10	5
B	∞	∞	∞	10
C	∞	∞	∞	∞
D	∞	∞	∞	∞
E	∞	∞	12	8
F	∞	∞	9	9
G	∞	8	8	8

Computing $d_3(u)$

How can we tell that $d_3(A) = 5$?
$d_2(F) = 9$ and $w(F, A) = -4$
So $d_3(A) = d_2(F) + w(F, A)$

Computing Distances

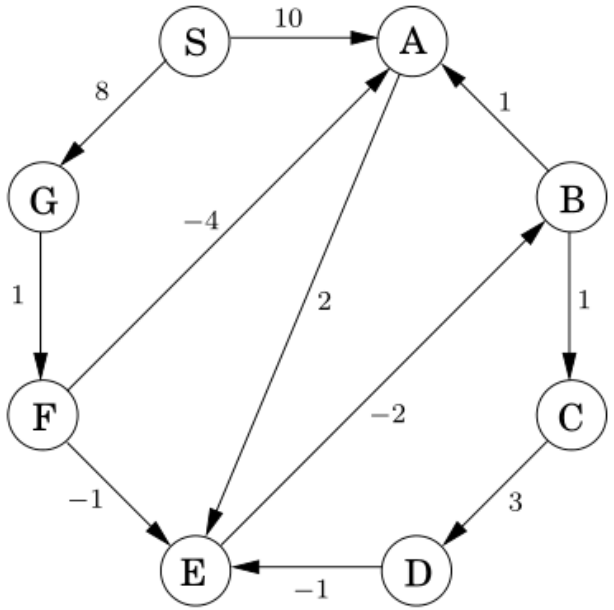


	k							
Node	0	1	2	3	4	5	6	7
S	0	0	0	0	0	0	0	0
A	∞	10	10	5	5	5	5	5
B	∞	∞	∞	10	6	5	5	5
C	∞	∞	∞	∞	11	7	6	6
D	∞	∞	∞	∞	∞	14	10	9
E	∞	∞	12	8	7	7	7	7
F	∞	∞	9	9	9	9	9	9
G	∞	8	8	8	8	8	8	8

Computing $d_3(u)$

How can we tell that $d_3(B) = 10$?
$d_2(E) = 12$ and $w(E, B) = -2$
So $d_3(B) = d_2(E) + w(E, B)$

Computing Distances

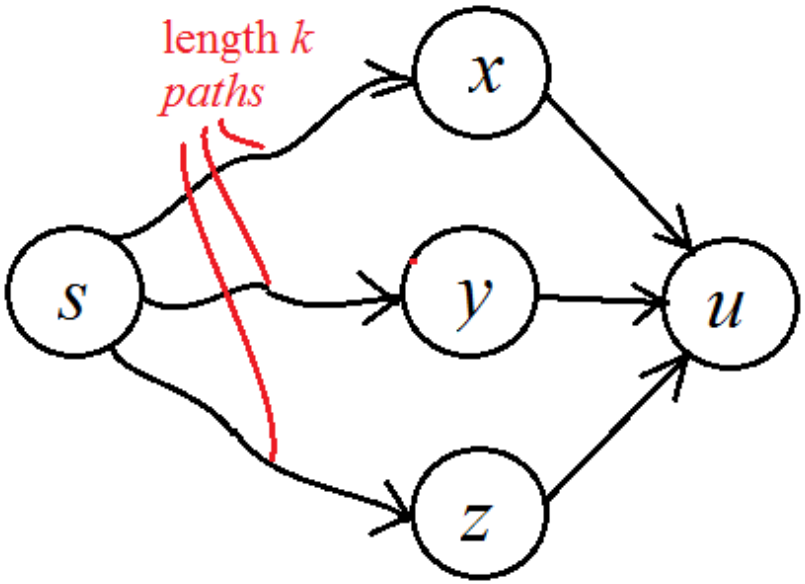


	k							
Node	0	1	2	3	4	5	6	7
S	0	0	0	0	0	0	0	0
A	∞	10	10	5	5	5	5	5
B	∞	∞	∞	10	6	5	5	5
C	∞	∞	∞	∞	11	7	6	6
D	∞	∞	∞	∞	∞	14	10	9
E	∞	∞	12	8	7	7	7	7
F	∞	∞	9	9	9	9	9	9
G	∞	8	8	8	8	8	8	8

Computing $d_3(u)$

How can we tell that $d_3(E) = 8$?
$d_2(F) = 9$ and $w(F, E) = -1$; $d_2(A) = 10$ and $w(A, F) = 2$
So $d_3(B) = d_2(F) + w(F, E)$

Computing Distances



Computing $d_{k+1}(u)$

$d_{k+1}(u)$ is the smallest among

$$d_k(u), d_k(x) + w(x,u), d_k(y) + w(y,u), d_k(z) + w(z,u)$$

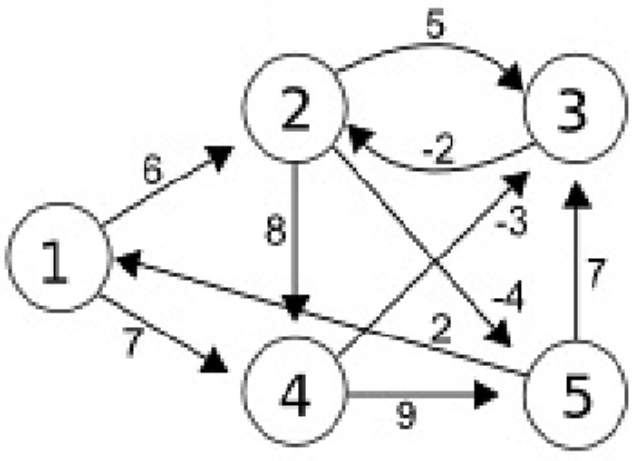
$$\Rightarrow d_{k+1}(u) = \min\{d_k(u), \min\{d_k(v) + w(v,u) \mid (v,u) \in E\}\}$$

Bellman-Ford Algorithm

Bellman-Ford(G, s)

INPUT: A graph G (without negative cycle) and starting node s
OUTPUT: $d(u)$ for every node u denoting distance from s to u
$d(s) \leftarrow 0$, $d(v) \leftarrow \infty$ for all other v
for $i = 1$ to $n - 1$ **do** :
    for $u \in V$ **do**
        $d'(u) \leftarrow d(u)$
        for $(v, u) \in E$ **do**
            $d'(u) \leftarrow \min\{d'(u), d(v) + w(v, u)\}$
    Replace d by d'

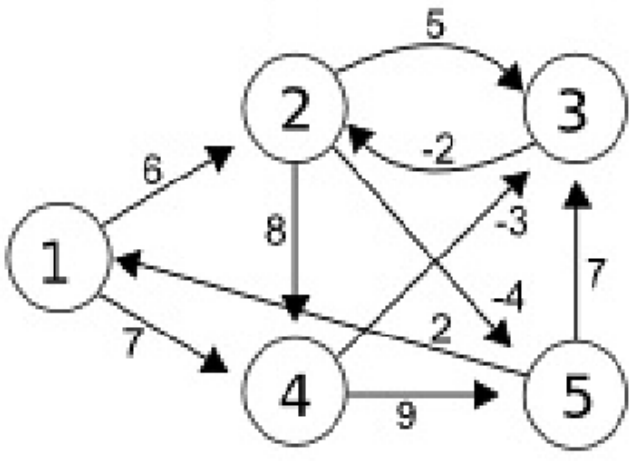
Bellman-Ford Algorithm



Iteration 1:

u	1	2	3	4	5
$d(u)$	0	∞	∞	∞	∞
$d'(u)$	0	6	∞	7	∞

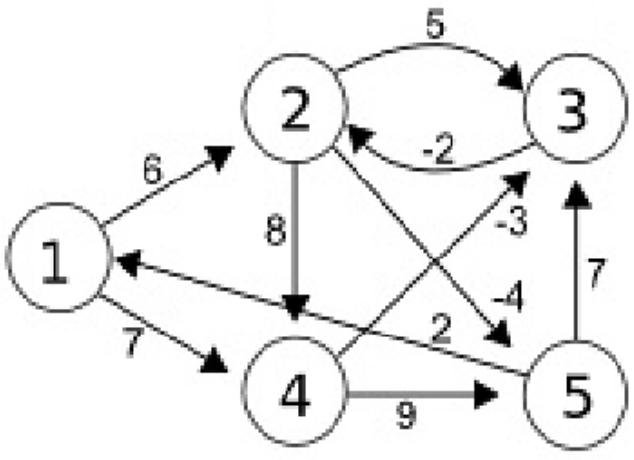
Bellman-Ford Algorithm



Iteration 1:

u	1	2	3	4	5
$d(u)$	0	6	∞	7	∞
$d'(u)$	0	6	∞	7	∞

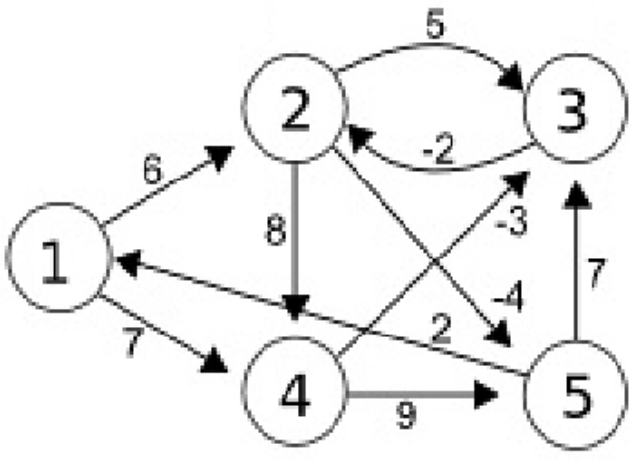
Bellman-Ford Algorithm



Iteration 2:

u	1	2	3	4	5
$d(u)$	0	6	∞	7	∞
$d'(u)$	0	6	4	7	2

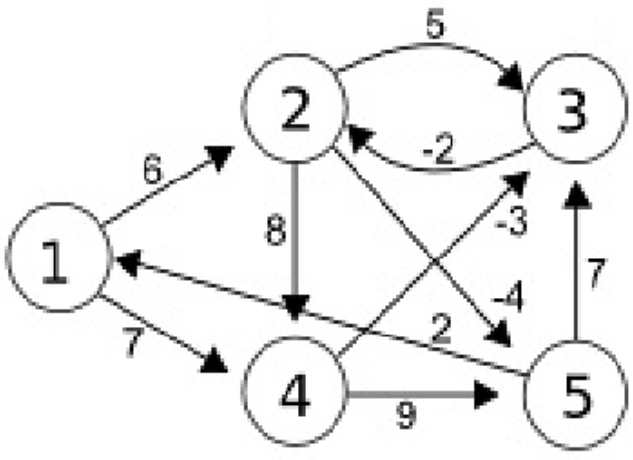
Bellman-Ford Algorithm



Iteration 2:

u	1	2	3	4	5
$d(u)$	0	6	4	7	2
$d'(u)$	0	6	4	7	2

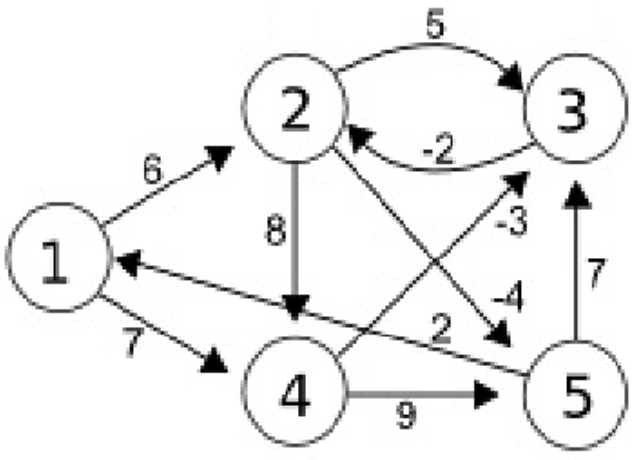
Bellman-Ford Algorithm



Iteration 3:

u	1	2	3	4	5
$d(u)$	0	6	4	7	2
$d'(u)$	0	2	4	7	−2

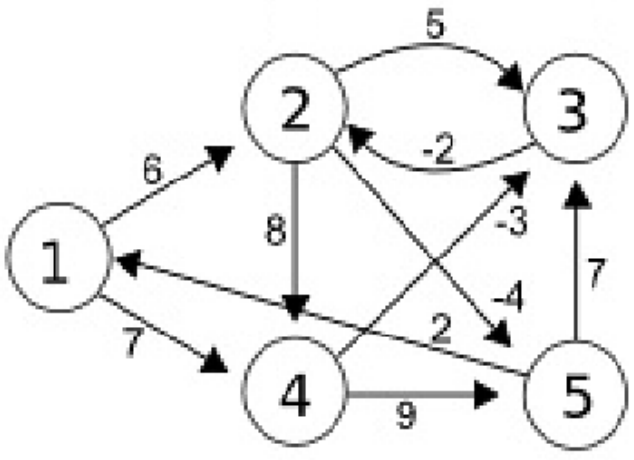
Bellman-Ford Algorithm



Iteration 3:

u	1	2	3	4	5
$d(u)$	0	2	4	7	−2
$d'(u)$	0	2	4	7	−2

Bellman-Ford Algorithm



Iteration 4:

u	1	2	3	4	5
$d(u)$	0	2	4	7	−2
$d'(u)$	0	2	4	7	−2

Bellman-Ford Algorithm: Correctness

Fact 1.

Suppose G has no negative cycle. After running Bellman-Ford(G, s), $d(u)$ is the distance from s to u for every $u \in V$.

Bellman-Ford Algorithm: Correctness

Fact 1.

Suppose G has no negative cycle. After running Bellman-Ford(G, s), $d(u)$ is the distance from s to u for every $u \in V$.

Proof. We prove the following loop invariant by induction on i:

After i rounds of the outmost for-loop, $d(u) = d_i(u)$.

Base case: $i = 0$. Before the loop, the statement is clear.

Inductive step: Suppose after i rounds, $d(u) = d_i(u)$ for every $u \in V$.

- Note $d_{i+1}(u) = \min\{d_i(u), \min\{d_i(v) + w(v, u) \mid (v, u) \in E\}\}$.
- Therefore after the $(i + 1)$th round, $d(u) = d_{i+1}(u)$.

□

Bellman-Ford Algorithm: Complexity

Fact 2.

Suppose G has no negative cycle. Bellman-Ford(G, s) computes the shortest distance from s to all other nodes in the graph G in time $\Theta(mn)$.

Proof.

- There are $n - 1$ iterations
- At each iteration, we process every incoming edge for every node.

  This means that we examine every edge in the graph (exactly once).

Therefore the total running time is $(n - 1) \times m$. □

All-Pair Shortest Path

Dijkstra's and Bell-Ford algorithm both solves Single-Source Shortest Path Problem.

All-Pair Shortest Path Problem

Compute the shortest distance between any pair of nodes in G.

All-Pair Shortest Path

Dijkstra's and Bell-Ford algorithm both solves Single-Source Shortest Path Problem.

All-Pair Shortest Path Problem

Compute the shortest distance between any pair of nodes in G.

Solution: Run the single-source algorithm n times, each time from a different node!
$\Rightarrow O(n^2 m)$ time

All-Pair Shortest Path

Dijkstra's and Bell-Ford algorithm both solves Single-Source Shortest Path Problem.

All-Pair Shortest Path Problem

Compute the shortest distance between any pair of nodes in G.

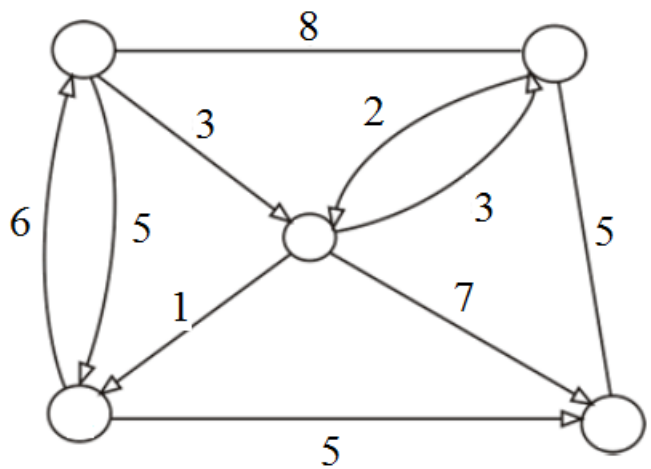
Solution: Run the single-source algorithm n times, each time from a different node!
$\Rightarrow O(n^2 m)$ time

Note:

- m could be as large as $\Theta(n^2)$.
- So Bellman-Ford algorithm would runs in $O(n^4)$.
- We would like a faster algorithm for this problem.

Floyd-Warshall Algorithm

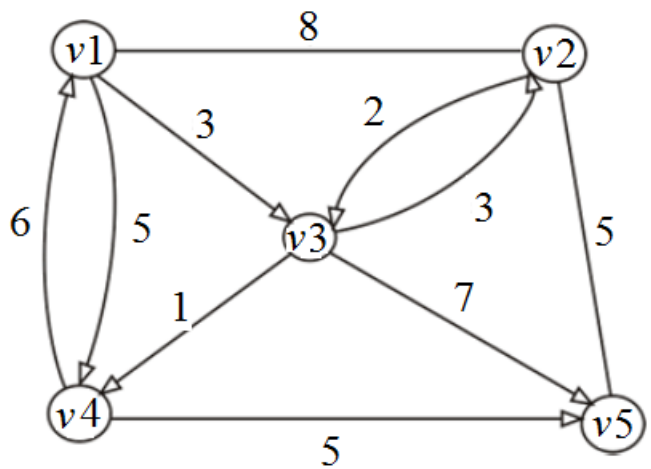
Floyd-Warshall algorithm



Floyd-Warshall Algorithm

Floyd-Warshall algorithm

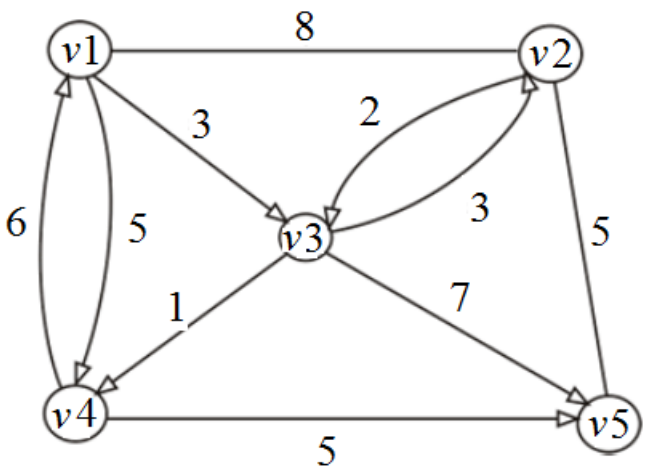
- Label all nodes using $v_1, v_2, \ldots, v_n$.



Floyd-Warshall Algorithm

Floyd-Warshall algorithm

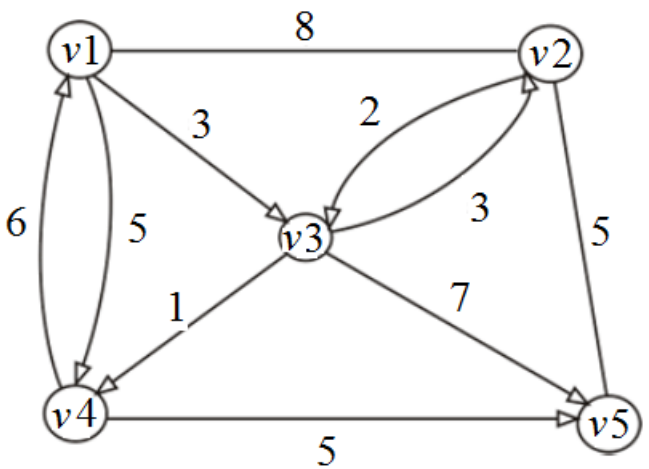
- Label all nodes using $v_1, v_2, \ldots, v_n$.
- Define $f_k(i, j)$ as the length of the shortest path between v_i, v_j that uses only $v_1, \ldots, v_k$ as intermediate nodes.



Floyd-Warshall Algorithm

Floyd-Warshall algorithm

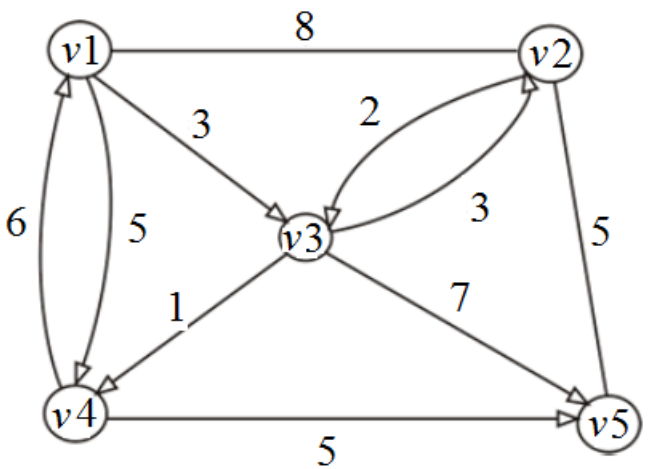
- Label all nodes using $v_1, v_2, \ldots, v_n$.
- Define $f_k(i, j)$ as the length of the shortest path between v_i, v_j that uses only $v_1, \ldots, v_k$ as intermediate nodes.
- **Fact**: $f_n(i, j)$ is the distance from v_i to v_j.



Floyd-Warshall Algorithm

Floyd-Warshall algorithm

- Label all nodes using $v_1, v_2, \ldots, v_n$.
- Define $f_k(i, j)$ as the length of the shortest path between v_i, v_j that uses only $v_1, \ldots, v_k$ as intermediate nodes.
- **Fact**: $f_n(i, j)$ is the distance from v_i to v_j.



Example: Going from 1 to 5:
Able to pass through v_1, v_2: $f_2(1, 5) = 13$
Able to pass through v_1, v_2, v_3: $f_3(1, 5) = 10$
Able to pass through v_1, v_2, v_3, v_4: $f_4(1, 5) = 9$

Floyd-Warshall Algorithm

Suppose G does not contain a negative cycle.

Computing $f_0(i, j)$

$f_0(i, j) = w(v_i, v_j)$

Floyd-Warshall Algorithm

Suppose G does not contain a negative cycle.

Computing $f_0(i, j)$

$f_0(i, j) = w(v_i, v_j)$

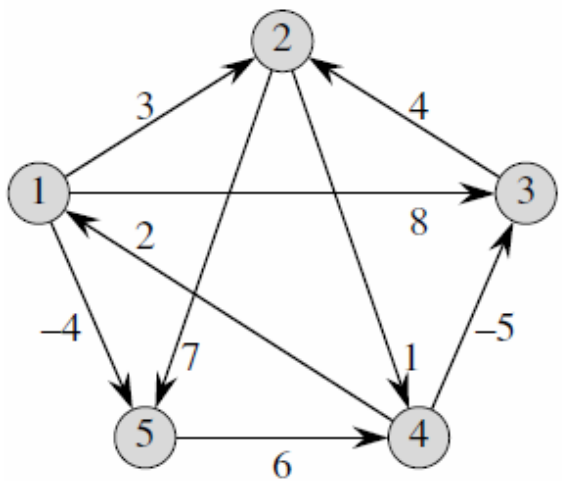
Computing $f_{k+1}(i, j)$

- Optimal Substructure: Suppose $u \rightsquigarrow v_{k+1} \rightsquigarrow v$ is a shortest path from u, v that only uses $v_1, \ldots, v_{k+1}$ as intermediate nodes, then $u \rightsquigarrow v_{k+1}$ and $v_{k+1} \rightsquigarrow v$ are shortest paths from u to v_{k+1} and from v_{k+1} to v using only $v_1, \ldots, v_k$ as intermediate nodes.
- Therefore we have
$$f_{k+1}(i, j) = \min\{f_k(i, j), f_k(i, k+1) + f_k(k+1, v)\}$$

Floyd-Warshall Algorithm

Floyd-Warshall(G)

INPUT: A graph G (without negative cycle)
OUTPUT: $f(i, j)$ for any i, j denoting distance from v_i to v_j
Create 2-dim arrays f, f'
$f(i, j) \leftarrow w(v_i, v_j)$ for all i, j
for $k = 1..n$ **do**
    for $i = 1$ to n **do**
        for $j = 1$ to n **do**
            $f'(i, j) \leftarrow \min\{f(i, j), f(i, k) + f(k, j)\}$
    Replace f by f'



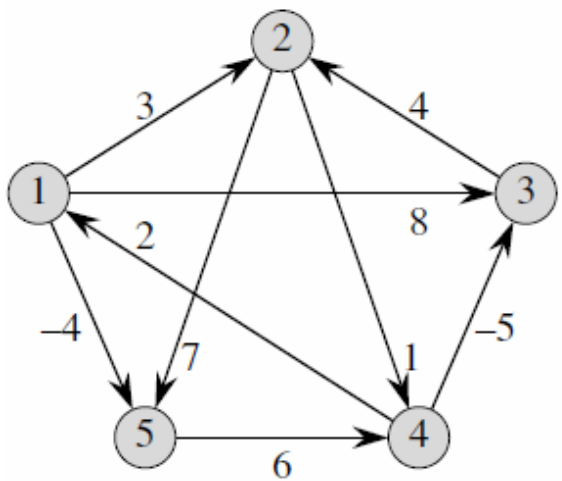
Initial Stage

$$f = \begin{pmatrix} 0 & 3 & 8 & \infty & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & \infty & \infty \\ 2 & \infty & -5 & 0 & \infty \\ \infty & \infty & \infty & 6 & 0 \end{pmatrix}$$

Floyd-Warshall Algorithm

Floyd-Warshall(G)

INPUT: A graph G (without negative cycle)
OUTPUT: $f(i, j)$ for any i, j denoting distance from v_i to v_j
Create 2-dim arrays f, f'
$f(i, j) \leftarrow w(v_i, v_j)$ for all i, j
for $k = 1..n$ **do**
    for $i = 1$ to n **do**
        for $j = 1$ to n **do**
            $f'(i, j) \leftarrow \min\{f(i, j), f(i, k) + f(k, j)\}$
    Replace f by f'



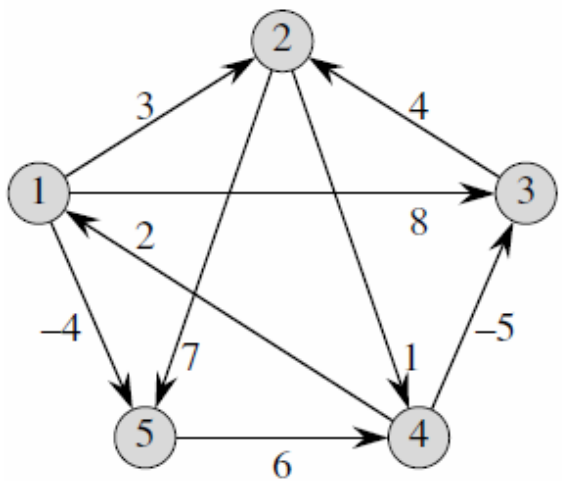
Stage 1

$$f = \begin{pmatrix} 0 & 3 & 8 & \infty & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & \infty & \infty \\ 2 & 5 & -5 & 0 & -2 \\ \infty & \infty & \infty & 6 & 0 \end{pmatrix}$$

Floyd-Warshall Algorithm

Floyd-Warshall(G)

INPUT: A graph G (without negative cycle)
OUTPUT: $f(i, j)$ for any i, j denoting distance from v_i to v_j
Create 2-dim arrays f, f'
$f(i, j) \leftarrow w(v_i, v_j)$ for all i, j
for $k = 1..n$ **do**
    for $i = 1$ to n **do**
        for $j = 1$ to n **do**
            $f'(i, j) \leftarrow \min\{f(i, j), f(i, k) + f(k, j)\}$
    Replace f by f'



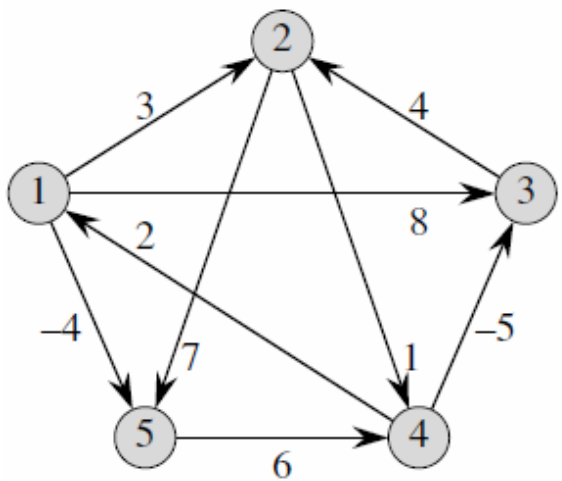
Stage 2

$$f = \begin{pmatrix} 0 & 3 & 8 & 4 & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & 5 & 11 \\ 2 & 5 & -5 & 0 & -2 \\ \infty & \infty & \infty & 6 & 0 \end{pmatrix}$$

Floyd-Warshall Algorithm

Floyd-Warshall(G)

INPUT: A graph G (without negative cycle)
OUTPUT: $f(i, j)$ for any i, j denoting distance from v_i to v_j
Create 2-dim arrays f, f'
$f(i, j) \leftarrow w(v_i, v_j)$ for all i, j
for $k = 1..n$ **do**
    for $i = 1$ to n **do**
        for $j = 1$ to n **do**
            $f'(i, j) \leftarrow \min\{f(i, j), f(i, k) + f(k, j)\}$
    Replace f by f'



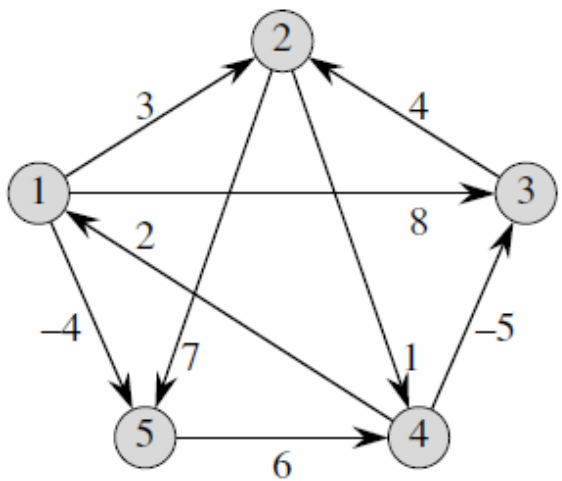
Stage 3

$$f = \begin{pmatrix} 0 & 3 & 8 & 4 & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & 5 & 11 \\ 2 & -1 & -5 & 0 & -2 \\ \infty & \infty & \infty & 6 & 0 \end{pmatrix}$$

Floyd-Warshall Algorithm

Floyd-Warshall(G)

INPUT: A graph G (without negative cycle)
OUTPUT: $f(i, j)$ for any i, j denoting distance from v_i to v_j
Create 2-dim arrays f, f'
$f(i, j) \leftarrow w(v_i, v_j)$ for all i, j
for $k = 1..n$ **do**
    for $i = 1$ to n **do**
        for $j = 1$ to n **do**
            $f'(i, j) \leftarrow \min\{f(i, j), f(i, k) + f(k, j)\}$
    Replace f by f'



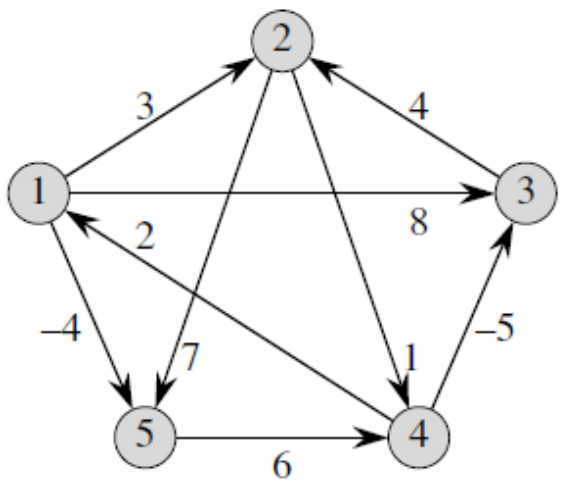
Stage 4

$$f = \begin{pmatrix} 0 & 3 & -1 & 4 & -4 \\ 3 & 0 & -4 & 1 & -1 \\ 7 & 4 & 0 & 5 & 3 \\ 2 & -1 & -5 & 0 & -2 \\ 8 & 5 & 1 & 6 & 0 \end{pmatrix}$$

Floyd-Warshall Algorithm

Floyd-Warshall(G)

INPUT: A graph G (without negative cycle)
OUTPUT: $f(i, j)$ for any i, j denoting distance from v_i to v_j
Create 2-dim arrays f, f'
$f(i, j) \leftarrow w(v_i, v_j)$ for all i, j
for $k = 1..n$ **do**
    for $i = 1$ to n **do**
        for $j = 1$ to n **do**
            $f'(i, j) \leftarrow \min\{f(i, j), f(i, k) + f(k, j)\}$
    Replace f by f'



Stage 5

$$f = \begin{pmatrix} 0 & 1 & -3 & 2 & -4 \\ 3 & 0 & -4 & 1 & -1 \\ 7 & 4 & 0 & 5 & 3 \\ 2 & -1 & -5 & 0 & -2 \\ 8 & 5 & 1 & 6 & 0 \end{pmatrix}$$

Floyd-Warshall Algorithm

Time Complexity : $O(n^3)$

Floyd-Warshall Algorithm

Time Complexity : $O(n^3)$

Further Questions

- How can we modify Floyd-Warshall Algorithm to give us the shortest paths, rather than the distances?
- What if the graph has a negative cycle? Can Floyd-Warshall Algorithm detect it?

Richard Bellman
``Dynamic Programming'' 1957

Robert Floyd
``Assigning Meaning to Programs''
1967

Shortest Paths, A Summary

SUMMARY

- Single-Source Positive Weights: Dijkstra's algorithm

  Complexity: Depends on the priority queue implementation

  - List $O(n^2)$
  - Binary /Binomial Heap $O((n+m)\log n)$
  - Fibonacci Heap $O(m + n\log n)$

- Single-Source Positive/Negative Weights: Bellman-Ford algorithm

  Complexity: $O(nm)$

- All-Pair: Floyd-Warshall algorithm

  Complexity: $O(n^3)$


THE UNIVERSITY OF
AUCKLAND
Te Whare Wānanga o Tāmaki Makaurau
NEW ZEALAND