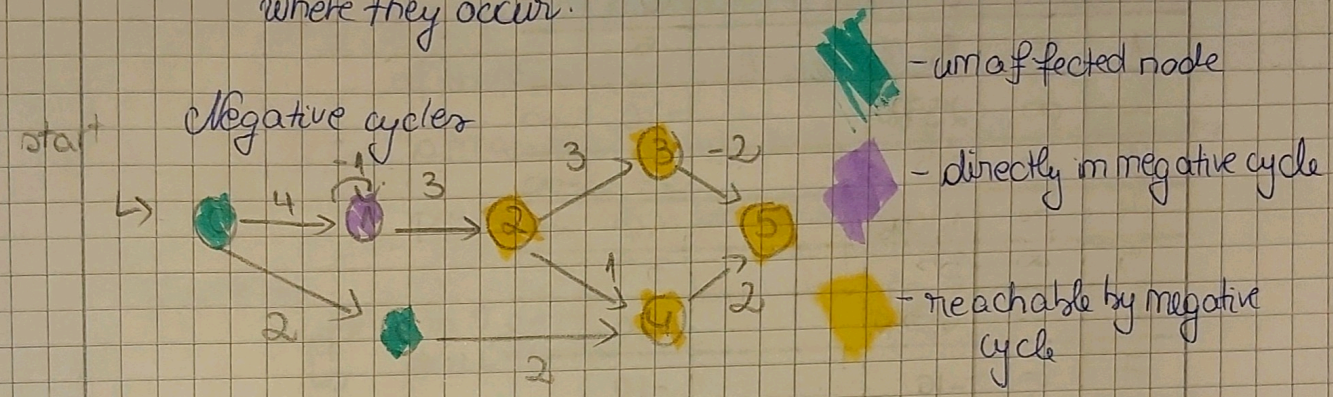
# BELLMAN-FORD ALGORITHM

- Single Source Shortest Path algorithm
- Complexity: $O(EV)$
  $O((E+V)\log(V))$ when using a binary heap priority queue
  - Dijkstra
- useful when the graph has negative edge weights
- can be used to detect negative cycles and determine where they occur.




## Bellman-Ford Algorithm Steps

$E$ - edge number, $V$ - vertices number, $S$ - id of the starting node

$D$ - array of size $V$ that tracks the best distance from $S$ to each node

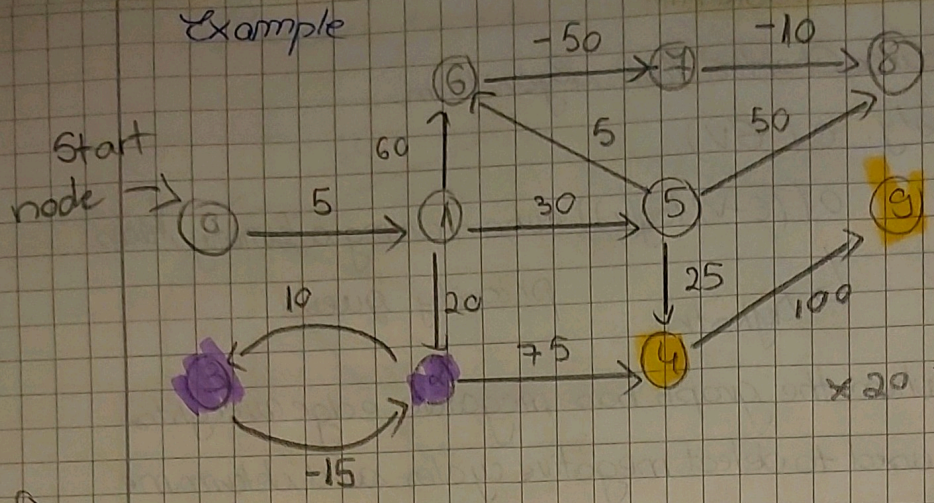
1.) Set every entry in $D$ to $+\infty$

2.) $D[S] = 0$

3.) Relax each edge $V-1$ times

```
for (i = 0; i < V-1; i++):
    for edge in graph.edges: //Relax Edge
        if (D.[edge.from] + edge.cost < D[edge.to])
            D[edge.to] = D[edge.from] + edge.cost
// Repeat to find nodes caught in a negative cycle
    for (i = 0; i < V-1; i++):
        for edge in graph.edges:
            if (D[edge.from] + edge.cost < D[edge.to])
                D[edge.to] = -∞
```

Example




| | | | | | |
|---|---|---|---|---|---|
| 0 | ∞ | 0 | ... | 0 | |
| 1 | ∞ | 5 | | 5 | |
| 2 | ∞ | 25 ~~20~~ | | -20 | -∞ |
| 3 | ∞ | 35 | | -5 | -∞ |
| 4 | ∞ | 100 ~~60~~ | | 60 | -∞ |
| 5 | ∞ | 35 | | 35 | |
| 6 | ∞ | 65 ~~40~~ | | 40 | |
| 7 | ∞ | -10 | | -10 | |
| 8 | ∞ | ~~-50~~ 20 | | -20 | |
| 9 | ∞ | 200 | | 160 | -∞ |

1st iteration to "relax" edges

↑ now detecting any negative cycles

```
function BellmanFord(list vertices, list edges, vertex source)
    distance[], predecessor[]
// Step 1. Initialize the graph
    for each vertex v in vertices:
        if v is source then distance[v] := 0
        else distance[v] := inf
        predecessor[v] = null
// Step 2. Relax edges repeatedly
    for i from 1 to size(vertices)-1:
        for each edge (u,v) with weight w in edges:
            if distance[u]+w < distance[v]:
                distance[v] := distance[u]+w
                predecessor[v] := u
```

// Step 3 : check for negative weight cycles

for each edge (u,v) with weight w in edges:

if distance[u]+w < distance[v]:

error "Graph contains a negative weight cycle"

return distance[], predecessor[]

Improvement

Input: s,t - 2 vertices

Output: dist: a map that associates to each accesible vertex the cost of the minimum cost walk from s to t

prev: a map that maps each accesible vertex to its predecessor on a path from s to t

Algorithm:

```
for x in V do:
    dist[x] = ∞

dist[s] = 0
changed = true; i = 0
while changed && i < V-1 do:
    changed = false
    for (x,y) in E do
        if dist[y] > dist[x] + cost(x,y) then
            dist[y] = dist[x] + cost(x,y)
            prev[y] = x
            changed = true
    i = i + 1
```

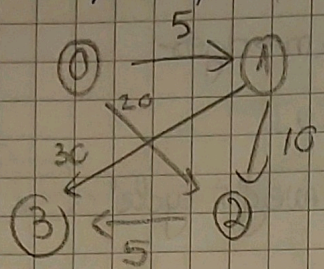
+ identify the negative cycles

BELLMAN'S OPTIMALITY PRINCIPLE

$d(x,z) <= d(x,y) + d(y,z)$

if $x, \dots, y, \dots, z$ is an optimal walk from x to z, then $x, \dots, y$ is an optimal part from x to y

Example of manual execution




D=0, k=3

| | changed | edge (x,y) | dist dict | prev dict |
|---|---|---|---|---|
| | | | 0 1 2 3 | |
| init | true | | 0 ∞ ∞ ∞ | |
| iteration 1 | false | | 0 1 2 3 | 0 1 2 3 |
| | true | (0,1) | 0 5 ∞ ∞ | 0 |
| | true | (0,2) | 0 5 20 ∞ | 0 0 |
| | true | (1,2) | 0 5 15 ∞ | 0 1 |
| | true | (1,3) | 0 5 15 35 | 0 1 1 |
| | true | (2,3) | 0 5 15 20 | 0 1 2 |
| iteration 2 | false | (0,1) | 0 5 15 20 | |
| | | (0,2) | 0 5 15 20 | |
| | | (1,2) | 0 5 15 20 | |
| | | (1,3) | 0 5 15 20 | |
| | | (2,3) | 0 5 15 20 | |
| | stop | | | |

The minimum cost walk from 0 to 3 is:

$0 \Rightarrow 1 \Rightarrow 2 \Rightarrow 3$ and has the cost of 20

## SHORTEST PATHS AND MATRIX MULTIPLICATION

Assumption: negative edge weights may be present, but no negative weight cycles

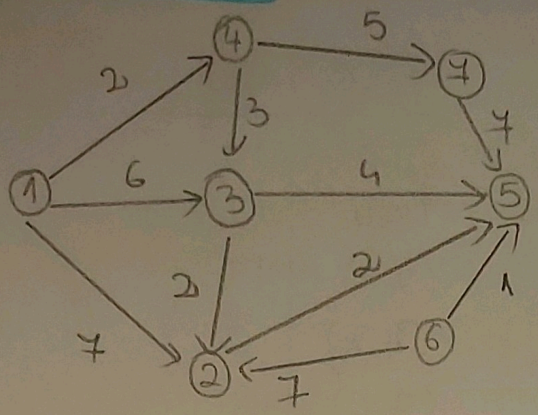
Shortest path from $v_i$ to $v_j$: $p_{ij}^m$ s.t. $|p_{ij}^m| \leq m$

i.e path $p_{ij}^m$ has at most m edges

$i = j \Rightarrow |p_{ii}| = 0$ & $w(p_{ii}) = 0$

$i \neq j \Rightarrow$ decompose path $p_{ij}^m$ into $p_{ik}^{m-1}$ & $v_k \rightarrow v_j$

# BELLMAN-FORD




if dist[y] > dist[x] + cost(x,y)

dist[y] = dist[x] + cost(x,y)

prev[y] = x

changed = true

| | changed | edge (x,y) | dist: dictionary | prev: dictionary |
|---|---|---|---|---|
| | | | 1 2 3 4 5 6 7 | |
| init | | | 0 ∞ ∞ ∞ ∞ ∞ ∞ | |
| it. 1 | false | | 1 2 3 4 5 6 7 | 1 2 3 4 5 6 7 |
| | true | (1,2) | 0 7 ∞ ∞ ∞ ∞ ∞ | 1 |
| | true | (1,3) | 0 7 6 ∞ ∞ ∞ ∞ | 1 1 |
| | true | (1,4) | 0 7 6 2 ∞ ∞ ∞ | 1 1 1 |
| | true | (4,3) | 0 7 5 2 ∞ ∞ ∞ | 1 4 1 |
| | — | (3,2) | 0 7 5 2 ∞ ∞ ∞ | 1 4 1 |
| | true | (4,7) | 0 7 5 2 ∞ ∞ 7 | 1 4 1 4 |
| | true | (3,5) | 0 7 5 2 9 ∞ 7 | 1 4 1 3 4 |
| | — | (7,5) | 0 7 5 2 9 ∞ 7 | 1 4 1 3 4 |
| | — | (2,5) | 0 7 5 2 9 ∞ 7 | 1 4 1 3 4 |
| | — | (2,6) | | |
| | — | (6,5) | | |
| it. 2 | false | | 1 2 3 4 5 6 7 | 1 2 3 4 5 6 7 |
| | false | (1,2) | 0 7 5 2 9 ∞ 7 | 1 4 1 3 - 4 |
| | false | (1,3) | 0 7 5 2 9 ∞ 7 | 1 4 1 3 - 4 |
| | false | (1,4) | 0 7 5 2 9 ∞ 7 | 1 4 1 3 - 4 |
| | false | (4,3) | | |
| | false | (3,2) | | |
| | false | (4,7) | | |
| | false | (3,5) | | |
| | false | (7,5) | | |
| | false | (2,5) | | |

Minimum cost walks

1→2: 1→2

1→3: 1→4→3

1→4: 1→4

1→5: 1→4→3→5

1→6: ∄

1→7: 1→4→7

Minimum cost walk starting from vertex 1 of length exactly 2.

∞ 8 5 ∞ 9 ∞ 7

1→1: 0

1→2: 1→3→2, cost 8

1→3: 1→4→3, cost 5

1→4: ∄

1→5: 1→2→5 : cost 9

1→6: ∄ , 1→7: 1→4→7 : cost 7