THINK FAST

Competitive Programming

From Problem 2 Solution in O(1)

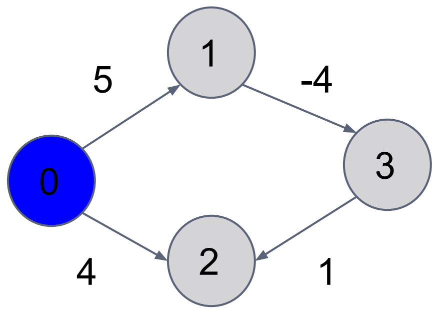
Graph Theory

Bellman-Ford Algorithm

Mostafa Saad Ibrahim
PhD Student @ Simon Fraser University

Recall: Dijkstra

- Solves Single-source shortest-paths (SSSP) problem
  - From one source s, find Shortest Path to all other nodes
- Dijkstra
  - Greedy + nonnegative weighted graph
  - 1st step: Pick non visited node with minimum cost

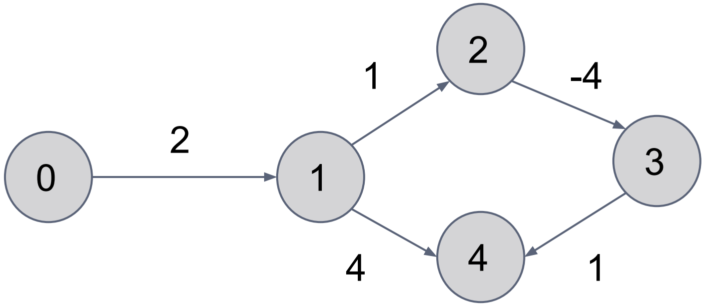


- Dijkstra pick shortest(0, 2) = [0, 2] = 4, **WRONG**
- shortest(0, 2) = [0, 1, 3, 2] = 5-4+1 = 2

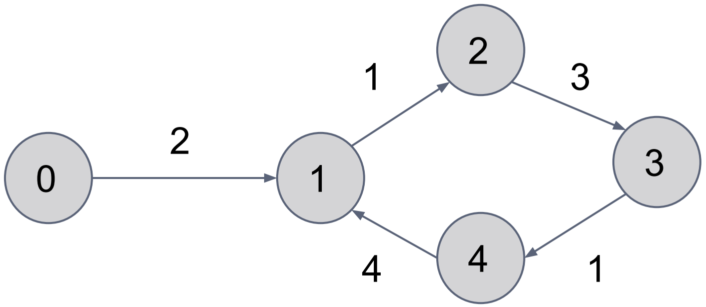
Bellman-Ford Algorithm

- Solves SSSP, but graph can have negative weights
- Why we may need -ve weights?
  - Money transactions: -10$ = money you have to pay
  - Games: -5 = lost 5 points for moving between states
  - Some algorithms, need a -ve weight SSSP due to its nature (e.g. Max Flow)
- If source can reach -ve cycle?
  - All nodes affected by the cycle has no path from src

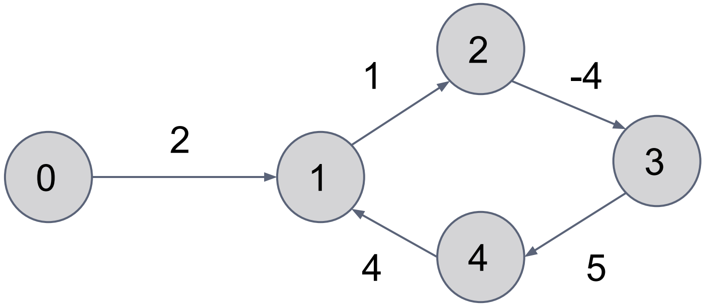
Cycles



No cycles.
Bellman works

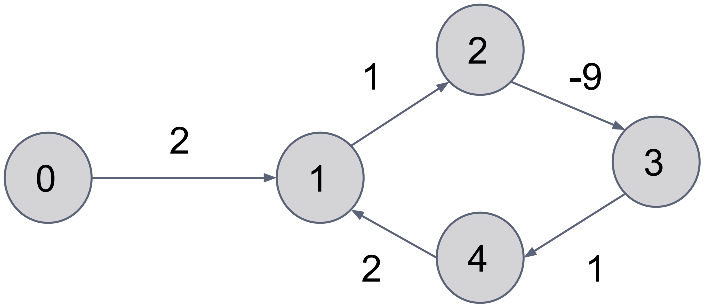


Positive cycle...reachable from 0
Bellman works

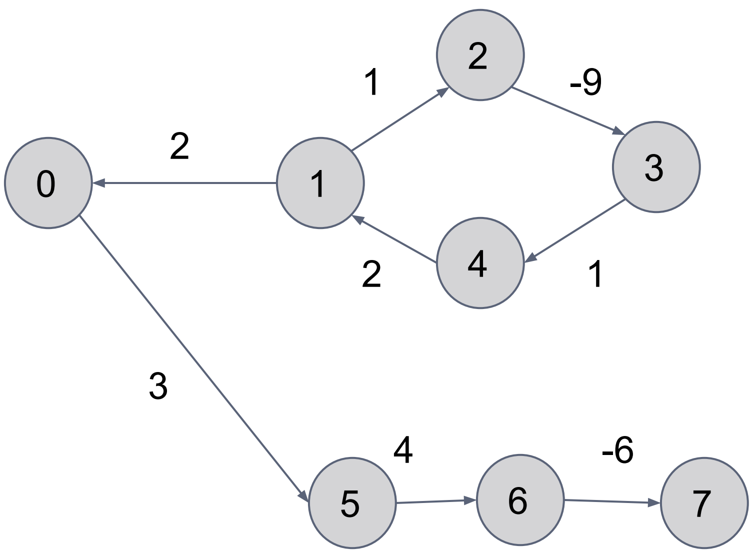


Positive cycle...reachable from 0
Bellman works

Cycles



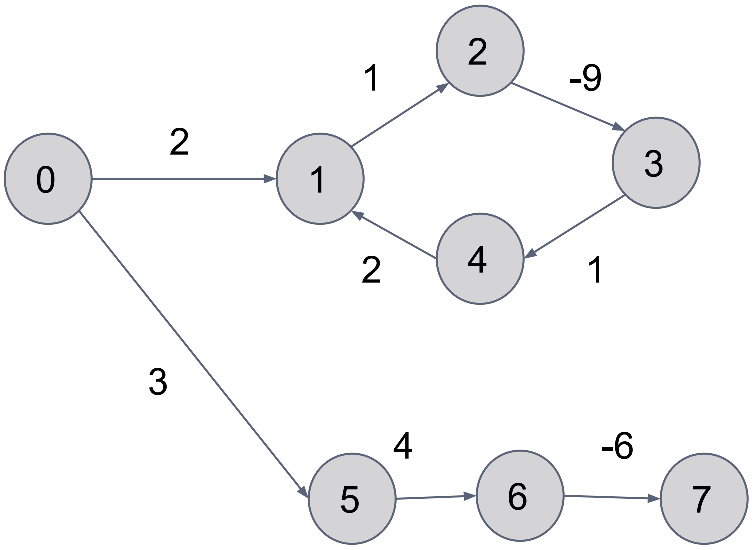
Negative cycle of cost -5...reachable from 0
NO algorithms can work



Negative cycle of cost -5...**NOT reachable** from 0
Bellman works
Cost for (1, 2, 3, 4) = OO

If added ege (7, 3) = -1, then -ve cycle is reachable

Cycles

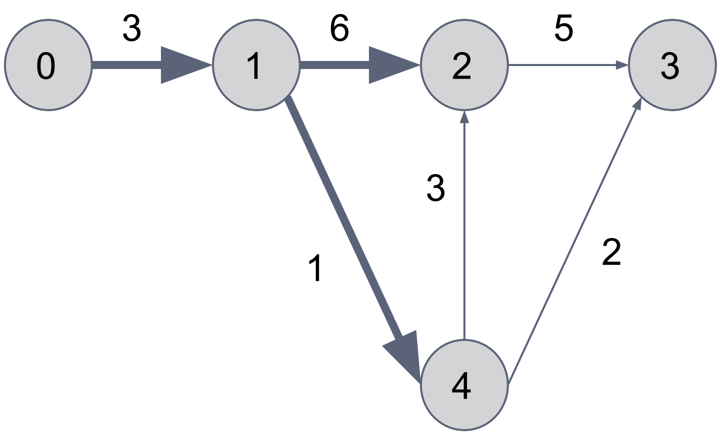


Negative cycle of cost -5...**reachable** from 0
Bellman has no path to (1, 2, 3, 4)
But has to (5, 6, 7)

Bellman-Ford Algorithm

- Fact: Simple path is **at most n – 1** edges
- There are 2 popular ways to outline bellman
- Think in bellman as contrast to Dijsktra
  - Relax ALL edges n-1 times (vs outgoing edges of node)
  - See Introduction to Algorithms book
  - Advantage: Minimize needed background to explain
- As dynamic programming solution
  - FindPath(from, at most edges) recurrence
  - See Algorithm Design book
  - This is simpler idea, easier to prove, but more tricky to get the algoithm optimized

Bellman-Ford Algorithm



What are all possible shortest paths from 0 with at most 2 edges?

{0, 1} = 3
{0, 1, 2} = 9
{0, 1, 4} = 4

Facts:

Expansions can be at most n-1 times

To expand for k edges, you need k-1 edges results

SP(S, X) uses 1 <= X <= n-1 edges

We can think of that recursively or iteratively

For each reachable node, **expand** it with every possible one edge:

{0, 1, 2} + (2, 3, 5) => {0, 1, 2, 3} = 14

{0, 1, 4} + (4, 3, 2) => {0, 1, 4, 3} = 6

{0, 1, 4} + (4, 2, 3) => {0, 1, 4, 2} = 7
which is better than {0, 1, 2} = 9

Bellman-Ford Algorithm: Rec

```
const int MAX = 1000;
int cost[MAX][MAX];
int from, n;

// source is globally defined: from
// Find shortest path from-to using at most max_edges
int bellman_rec(int to, int max_edges)
{
    if (max_edges == 1)
        return cost[from][to];

    // Actual path is not max_edges edges..use fewer edges
    int ans = bellman_rec(to, max_edges-1);

    // Find shortest path to node i + expand path with edge (i, to)
    for (int i = 0; i < n; ++i) if(i != to)
    {
        int total_cost = bellman_rec(i, max_edges-1) + cost[i][to];
        ans = min(ans, total_cost);
    }
    return ans;
}
```

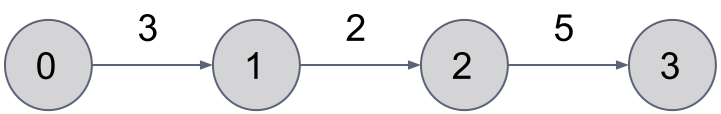
Bellman-Ford Algorithm: improvements

- Order: O(n^3) time and O(N^2) memory
- Switch to Adjacency list,
  - The node N^2 is replaced with M
  - O(NM) time and O(N^2) memory
- Write code using table method
  - Using rolling table technique in DP
  - Now O(N) memory
- Or you can directly prove next code as it is
  - Use the idea of edge expansion iteratively

Bellman-Ford Algorithm: iterative

```
struct edge {
    int from, to, w;

    edge(int from, int to, int w) :
        from(from), to(to), w(w) {
    }
};

void Bellman(vector<edge> & edgeList, int n, int from)
{
    vector<int> dist(n, OO);
    dist[ from ] = 0;

    for (int max_edges = 0; max_edges < n-1; ++max_edges)
    {
        // iterate on each node, iterate on its edges = iterate on all edges
        for (int j = 0; j < sz(edgeList); ++j)
        {
            edge ne = edgeList[j];

            // Can reach to with 1 more edge?
            if (dist[ne.to] > dist[ne.from] + ne.w)
                dist[ne.to] = dist[ne.from] + ne.w;
        }
    }
}
```

Bellman-Ford Tracing



Assume edges order:
(0, 1) = 3
(1, 2) = 2
(2, 3) = 5

Distance arr:
dist[0] = 0
dist[1] = OO
dist[2] = OO
dist[3] = OO

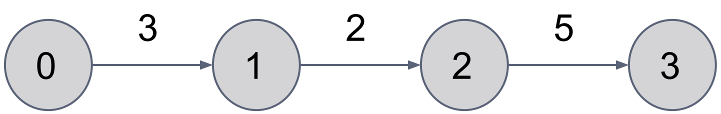
```
edge ne = edgeList[j];

// Can reach to with 1 more edge?
if (dist[ne.to] > dist[ne.from] + ne.w)
    dist[ne.to] = dist[ne.from] + ne.w;
```

max_edge = 0, j = 0
ne = {0, 1, 3}

dist[1] > dist[0] + 3
OO > 0 + 3 => YES

Relax using this info
dist[1] = 3

Bellman-Ford Tracing



Assume edges order:
(0, 1) = 3
(1, 2) = 2
(2, 3) = 5

Distance arr:
dist[0] = 0
dist[1] = 3
dist[2] = OO
dist[3] = OO

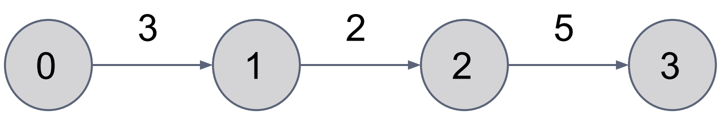
```
edge ne = edgeList[j];

// Can reach to with 1 more edge?
if (dist[ne.to] > dist[ne.from] + ne.w)
    dist[ne.to] = dist[ne.from] + ne.w;
```

max_edge = 0, j = 1
ne = {1, 2, 2}

dist[2] > dist[1] + 2
OO > 3 + 2 => YES

Relax using this info
dist[2] = 5

Bellman-Ford Tracing



Assume edges order:
(0, 1) = 3
(1, 2) = 2
(2, 3) = 5

Distance arr:
dist[0] = 0
dist[1] = 3
dist[2] = 5
dist[3] = OO

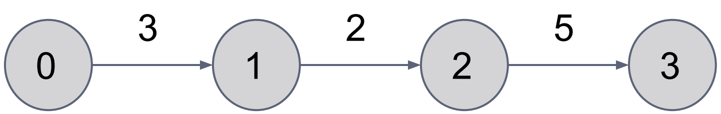
```
edge ne = edgeList[j];

// Can reach to with 1 more edge?
if (dist[ne.to] > dist[ne.from] + ne.w)
    dist[ne.to] = dist[ne.from] + ne.w;
```

max_edge = 0, j = 2
ne = {2, 3, 5}

dist[3] > dist[2] + 5
OO > 5 + 5 => YES

Relax using this info
dist[3] = 10

Bellman-Ford Tracing



Assume edges order:
(0, 1) = 3
(1, 2) = 2
(2, 3) = 5

Distance arr:
dist[0] = 0
dist[1] = 3
dist[2] = 5
dist[3] = 10

```
edge ne = edgeList[j];

// Can reach to with 1 more edge?
if (dist[ne.to] > dist[ne.from] + ne.w)
    dist[ne.to] = dist[ne.from] + ne.w;
```

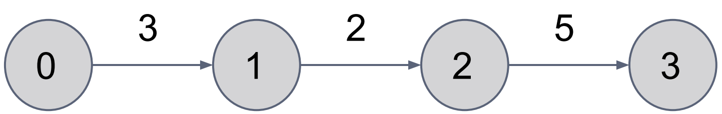
max_edge = 1, j = 0
ne = {0, 1, 3}

dist[1] > dist[0] + 3
3 > 3 + 0 => NO

And every next iteration will be zero

let's try different edges ordering

Bellman-Ford Tracing



Assume edges order:
(1, 2) = 2
(0, 1) = 3
(2, 3) = 5

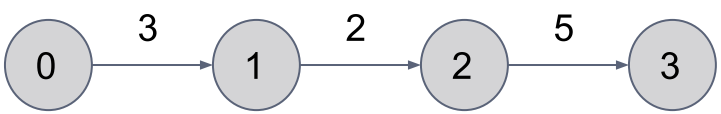
Distance arr:
dist[0] = 0
dist[1] = OO
dist[2] = OO
dist[3] = OO

```
edge ne = edgeList[j];

// Can reach to with 1 more edge?
if (dist[ne.to] > dist[ne.from] + ne.w)
    dist[ne.to] = dist[ne.from] + ne.w;
```

max_edge = 0, j = 0
ne = {1, 2, 2}

dist[2] > dist[1] + 2
OO > OO + 3 => NO

Bellman-Ford Tracing



Assume edges order:
(1, 2) = 2
(0, 1) = 3
(2, 3) = 5

Distance arr:
dist[0] = 0
dist[1] = OO
dist[2] = OO
dist[3] = OO

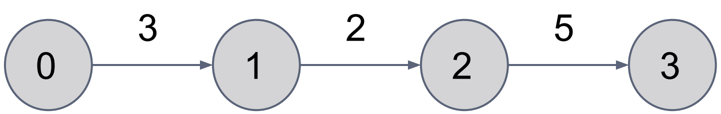
```
edge ne = edgeList[j];

// Can reach to with 1 more edge?
if (dist[ne.to] > dist[ne.from] + ne.w)
    dist[ne.to] = dist[ne.from] + ne.w;
```

max_edge = 0, j = 1
ne = {0, 1, 3}

dist[1] > dist[0] + 3
OO > 0 + 3 => YES

Relax using this info
dist[1] = 3

Bellman-Ford Tracing



Assume edges order:
(1, 2) = 2
(0, 1) = 3
(2, 3) = 5

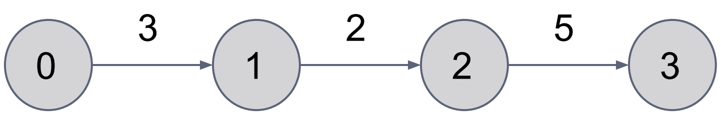
Distance arr:
dist[0] = 0
dist[1] = 3
dist[2] = OO
dist[3] = OO

```
edge ne = edgeList[j];

// Can reach to with 1 more edge?
if (dist[ne.to] > dist[ne.from] + ne.w)
    dist[ne.to] = dist[ne.from] + ne.w;
```

max_edge = 0, j = 2
ne = {2, 3, 5}

dist[3] > dist[2] + 5
OO > OO + 5 => NO

Bellman-Ford Tracing



Assume edges order:
(1, 2) = 2
(0, 1) = 3
(2, 3) = 5

Distance arr:
dist[0] = 0
dist[1] = 3
dist[2] = OO
dist[3] = OO

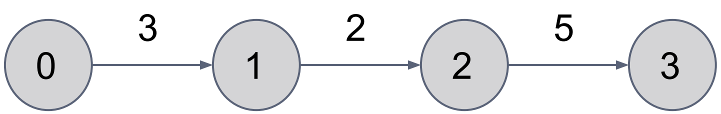
```
edge ne = edgeList[j];

// Can reach to with 1 more edge?
if (dist[ne.to] > dist[ne.from] + ne.w)
    dist[ne.to] = dist[ne.from] + ne.w;
```

max_edge = 1, j = 0
ne = {1, 2, 2}

dist[2] > dist[1] + 2
OO > 3 + 2 => Yes

Relax using this info
dist[2] = 5

Bellman-Ford Tracing



Assume edges order:
(1, 2) = 2
(0, 1) = 3
(2, 3) = 5

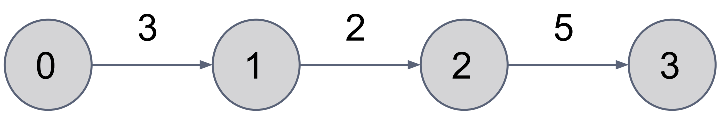
Distance arr:
dist[0] = 0
dist[1] = 3
dist[2] = 5
dist[3] = OO

```
edge ne = edgeList[j];

// Can reach to with 1 more edge?
if (dist[ne.to] > dist[ne.from] + ne.w)
    dist[ne.to] = dist[ne.from] + ne.w;
```

max_edge = 1, j = 1
ne = {0, 1, 3}

dist[1] > dist[0] + 3
3 > 0 + 3 => No

Bellman-Ford Tracing



Assume edges order:
(1, 2) = 2
(0, 1) = 3
(2, 3) = 5

Distance arr:
dist[0] = 0
dist[1] = 3
dist[2] = 5
dist[3] = OO

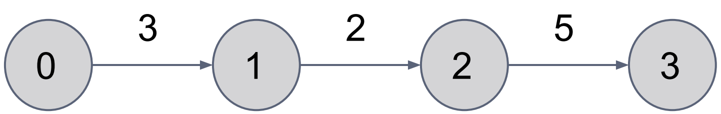
```
edge ne = edgeList[j];

// Can reach to with 1 more edge?
if (dist[ne.to] > dist[ne.from] + ne.w)
    dist[ne.to] = dist[ne.from] + ne.w;
```

max_edge = 1, j = 2
ne = {2, 3, 5}

dist[3] > dist[2] + 5
OO > 5 + 5 => YES

Relax using this info
dist[3] = 10

Bellman-Ford Tracing



Assume edges order:
(2, 3) = 5
(1, 2) = 2
(0, 1) = 3

Distance arr:
dist[0] = 0
dist[1] = OO
dist[2] = OO
dist[3] = OO

```
edge ne = edgeList[j];

// Can reach to with 1 more edge?
if (dist[ne.to] > dist[ne.from] + ne.w)
    dist[ne.to] = dist[ne.from] + ne.w;
```

After max_edge = 0

Distance arr:
dist[0] = 0
dist[1] = 3
dist[2] = OO
dist[3] = OO

After max_edge = 1

Distance arr:
dist[0] = 0
dist[1] = 3
dist[2] = 5
dist[3] = OO

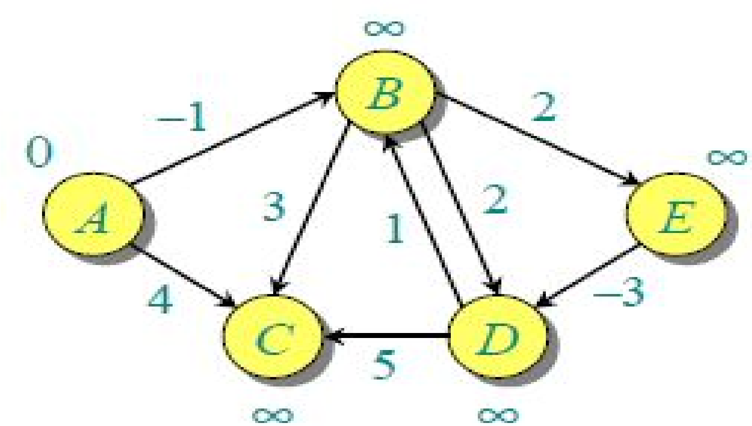
After max_edge = 2

Distance arr:
dist[0] = 0
dist[1] = 3
dist[2] = 5
dist[3] = 10

Bellman-Ford Algorithm: behaviour

- Bellman-ford is **pull-based** algorithm
  - It can only make use of neighbour info
  - E.g. when edges were totally reversed, it only made use of first edge 0-1
  - In 2nd iteration, it could only use 1-2
  - In 3rd iteration, it used 2-3
- So it expands knowledge based on its dist[]
- In worst case, n-1 is enough for any path
- In ith iteration, Shortest Paths of at most i-edges are found

Bellman-Ford Tracing



A	*B*	*C*	*D*	*E*
0	∞	∞	∞	∞

Assume edges order

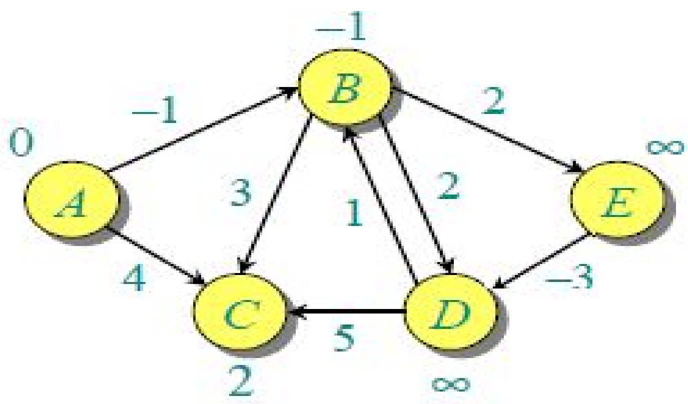
(B,E)
(D,B)
(B,D)
(A,B)
(A,C)
(D,C)
(B,C)
(E,D).

Pull-Based?

Only A is reachable => Either edge A-B or A-C will be first relaxation!

Src: http://www.geeksforgeeks.org/dynamic-programming-set-23-bellman-ford-algorithm/

Bellman-Ford Tracing



A	B	C	D	E
0	∞	∞	∞	∞
0	−1	∞	∞	∞
0	−1	4	∞	∞
0	−1	2	∞	∞

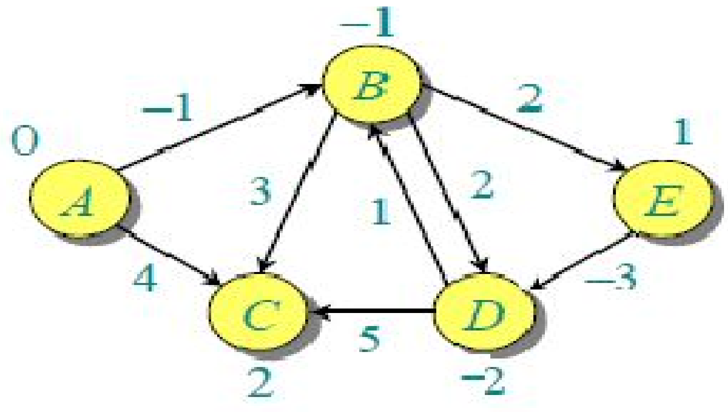
Assume edges order

(B,E)
(D,B)
(B,D)
(A,B)
(A,C)
(D,C)
(B,C)
(E,D).

Pull-Based?
A is reachable => Active edges {A-B, A-C}
B is reachable => Active edges {B-C, B-D, B-E}
C is reachable => Active edges {}

Src: http://www.geeksforgeeks.org/dynamic-programming-set-23-bellman-ford-algorithm/

Bellman-Ford Tracing



A	B	C	D	E
0	∞	∞	∞	∞
0	−1	∞	∞	∞
0	−1	4	∞	∞
0	−1	2	∞	∞
0	−1	2	∞	1
0	−1	2	1	1
0	−1	2	−2	1

Assume edges order

(B,E)
(D,B)
(B,D)
(A,B)
(A,C)
(D,C)
(B,C)
(E,D).

Src: http://www.geeksforgeeks.org/dynamic-programming-set-23-bellman-ford-algorithm/

Bellman-Ford Algorithm: improvement

- Assume N = 1000. In 50th step, the internal loop if condition is not activated
- Does it worth iterating more? No
- Improvement: If an iteration has no update, next won't have update..just break

Bellman-Ford Algorithm: improvement

```
void Bellman(vector<edge> & edgeList, int n, int from)
{
    vector<int> dist(n, OO);
    dist[ from ] = 0;

    for (int max_edges = 0, r = 0; max_edges < n-1; ++max_edges, r = 0)
    {
        // iterate on each node, iterate on its edges = iterate on all edges
        for (int j = 0; j < sz(edgeList); ++j)
        {
            edge ne = edgeList[j];

            // Can reach to with 1 more edge?
            if (dist[ne.to] > dist[ne.from] + ne.w)
                dist[ne.to] = dist[ne.from] + ne.w, r = 1;
        }
        if(!r)      // condition is not accessed!
            break;
    }
}
```

Bellman-Ford Algorithm: cycle detection

- So far we compute shortest path
- What if there is -ve cycle, how to detec?
- Simple trick
  - A path is at most n-1 edges
  - Can’t be relaxed more
  - If in n-th iteration a path is relaxed, this path has n edges
  - So not simple path
  - Then -ve cycle

Bellman-Ford Algorithm: cycle detection

```
bool Bellman(vector<edge> & edgeList, int n, int from)
{
    vector<int> dist(n, OO);
    dist[ from ] = 0;

    for (int max_edges = 0, r = 0; max_edges < n; ++max_edges, r = 0)
    {
        // iterate on each node, iterate on its edges = iterate on all edges
        for (int j = 0; j < sz(edgeList); ++j)
        {
            edge ne = edgeList[j];

            // Can reach to with 1 more edge?
            if (dist[ne.to] > dist[ne.from] + ne.w)
            {
                dist[ne.to] = dist[ne.from] + ne.w, r = 1;

                if (max_edges == n-1)
                    return true;    // -ve cycle
            }
        }
        if(!r)      // condition is not accessed!
            break;
    }
    return false;   // no -ve cycle
}
```

Bellman-Ford Algorithm: get path

```
bool Bellman(vector<edge> & edgeList, int n, int from)
{
    vector<int> dist(n, OO);
    vector<int> prev(n, -1);
    dist[ from ] = 0;

    for (int max_edges = 0, r = 0; max_edges < n; ++max_edges, r = 0)
    {
        // iterate on each node, iterate on its edges = iterate on all edges
        for (int j = 0; j < sz(edgeList); ++j)
        {
            edge ne = edgeList[j];

            // Can reach to with 1 more edge?
            if (dist[ne.to] > dist[ne.from] + ne.w)
            {
                dist[ne.to] = dist[ne.from] + ne.w, prev[ ne.to ] = ne.from, r = 1;

                if (max_edges == n-1)
                    return true;    // -ve cycle
            }
        }
        if(!r)      // condition is not accessed!
            break;
    }
    // backtrack on prev to get a path
    // See attached code to video :)
    return false;   // no -ve cycle
}
```

Bellman-Ford Algorithm: More

- We can know all nodes affected by -ve cycle
  - After bellman finishes, saves its distance array
  - Run bellman on updated array (not sure if 1 iter enough)
  - Compare with new dist arr, Different values = Node Cycle
- Find a cycle
  - Start from any affected node, say node A
  - it is either in the cycle...or cycle reach it
  - Go back (prev array), n steps
  - Now, you must end at cycle..say node B
  - Go back again, till you see B again..this a cycle
- Find positive cycle? Multiple graph with -1

تم بحمد الله

علمكم الله ما ينفعكم

ونفعكم بما تعلمتم

وزادكم علماً