Single-Source Shortest Path Problem

I/p: Directed weighted graph G and a source vertex s.

O/p: For each $v \in V(G)$ Compute, length of a shortest s-v path in G.

"length of path": Sum of edge weights of the path.

Recall Dijkstra's Algorithm

- Computes the shortest path from the origin s to every other node v in the graph. [Greedy Algorithm]

- The idea is to maintain a set S with the property that the shortest path from s to each node in S is known.

  We start with $S = \{s\}$

- At our greedy step, we consider the minimum cost edge leaving the set S,

  that is $\min_{\substack{u \in S \\ v \notin S}} \{w_{uv} + d(u)\}$

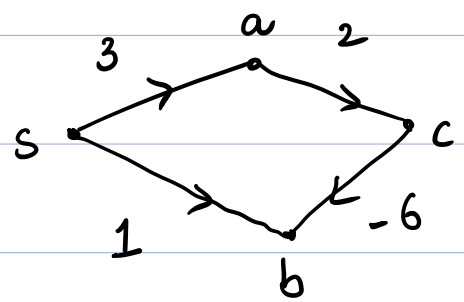
- If uv is the edge on which this minimum obtained then we add v to S.

- Dijkstra's algorithm runs in O(mlogn) using heaps.

Cons

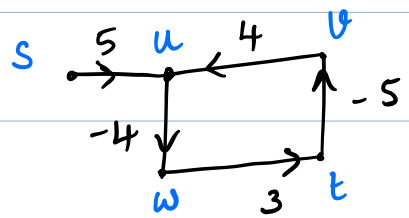
- No negative cost edges } Bellman-ford algorithm.

Dijkstra's greedy approach will not work if in case of negative edge costs/weights



Basic terminology:

- A cycle C in a weighted graph G is called negative cycle if $\sum_{e \in C} w_e < 0$.



C : $uwtv$ is a negative cycle

$\sum_{e \in C} w_e = 4 - 4 + 3 - 5$

$= -2 < 0$

Negative-weight cycles

If the graph contains a negative-weight cycle reachable from s, then shortest-path weights are not well defined.

No path from s to a vertex on the cycle can be a shortest path, because we can always find a path with lower weight by following the proposed shortest path and then traversing the negative-weight cycle.

Question:

Can a Shortest Path contain a cycle?

↓

Positive/negative weight cycle

Single-Source Shortest Paths in directed acyclic graphs (DAG)

As there are no cycles in DAG Shortest Paths are well defined even if there are negative-weight edges.

The idea is to use topological ordering of the DAG.

See the next page for full details.

Algorithm

DAG-SSP(s):

topologically sort the vertices of G.

For each vertex $u \in V(G)$

If $u = s$

$d(u) = 0$

else

$d(u) = \infty$

For each vertex u, taken in topological order

For each vertex v adjacent to u

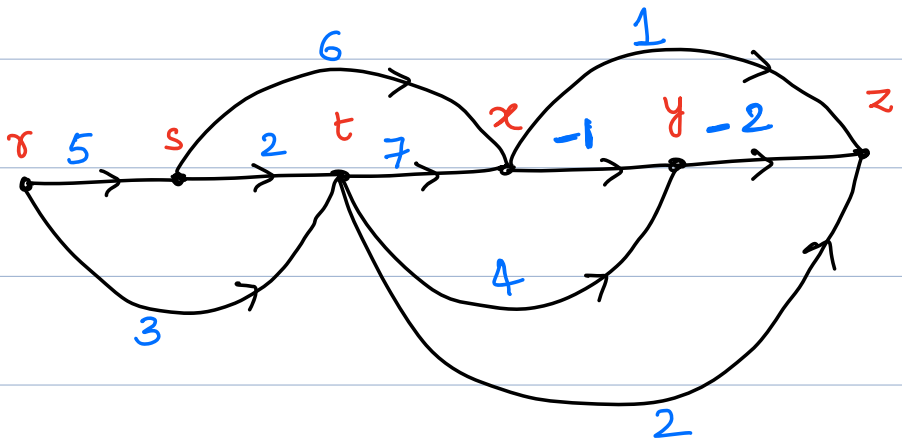
if $d(v) > d(u) + w_{uv}$

$d(v) = d(u) + w_{uv}$

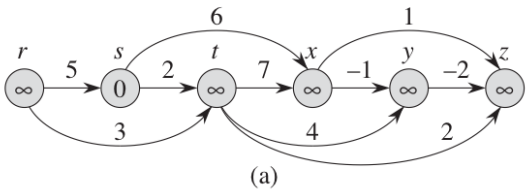
Running time: $\Theta(V+E)$

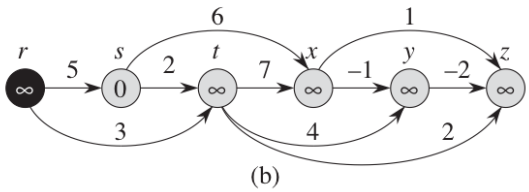
I/p: Adjacency list representation.

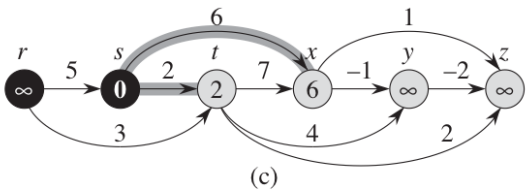
Example:

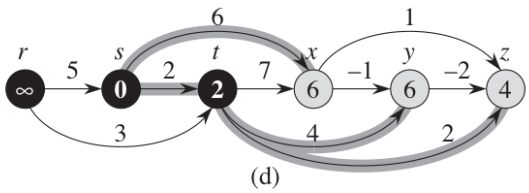


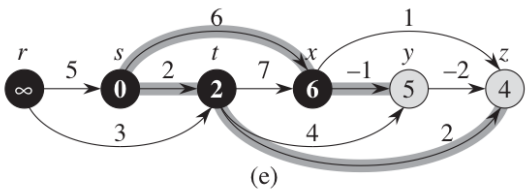
Execution of the algorithm on the above graph shown in the next page.

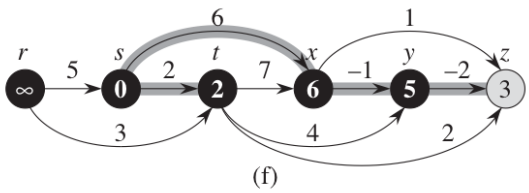

r s t x y z
∞ 0 ∞ ∞ ∞ ∞
5 2 7 −1 −2
6 1
3 4 2
(a)

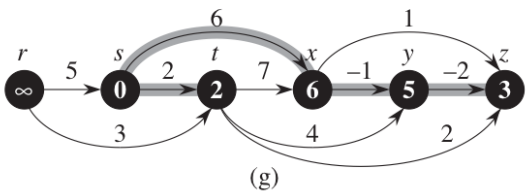

r s t x y z
∞ 0 ∞ ∞ ∞ ∞
5 2 7 −1 −2
6 1
3 4 2
(b)


r s t x y z
∞ 0 2 6 ∞ ∞
5 2 7 −1 −2
6 1
3 4 2
(c)


r s t x y z
∞ 0 2 6 6 4
5 2 7 −1 −2
6 1
3 4 2
(d)


r s t x y z
∞ 0 2 6 5 4
5 2 7 −1 −2
6 1
3 4 2
(e)

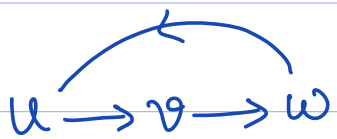

r s t x y z
∞ 0 2 6 5 3
5 2 7 −1 −2
6 1
3 4 2
(f)


r s t x y z
∞ 0 2 6 5 3
5 2 7 −1 −2
6 1
3 4 2
(g)

Q: Does the above idea work if G is NOT a DAG?

Ans: NO

Suppose the input graph contains a directed cycle



To Compute $d(w)$ we need $d(v)$

" " $d(v)$ " " $d(u)$

" " $d(u)$ " " $d(w)$

We get struck in an infinite loop!

(We use Bellman-Ford algorithm in this case)

See next page for details.

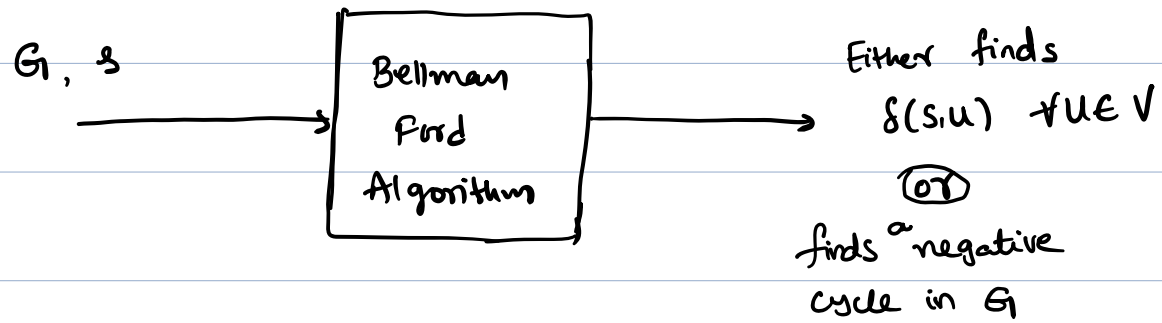
The Bellman-Ford algorithm

Ref: [KT-book]

The Algorithm Solves the Single-Source Shortest Paths Problem in which edge weights may be negative.

(*) If there is a negative-weight cycle that is reachable from the source, the algorithm indicates that no solution exists.

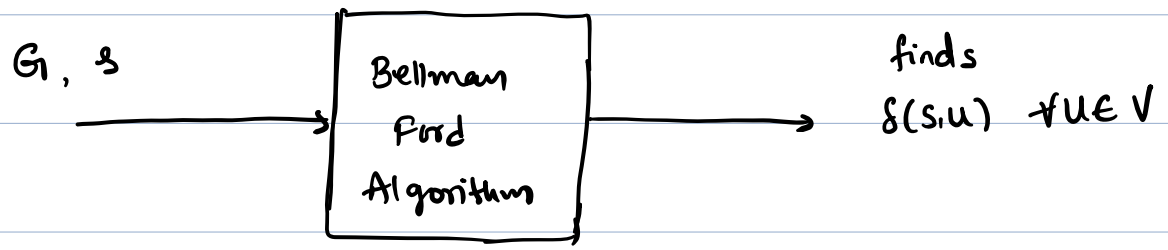
If there is no such cycle, the algorithm Produces the Shortest Paths and their weights.



For convenience,

Assume that G has no negative cycles reachable from the source.

[Reason is given at the end of the algorithm]



if G has negative cycles

⊛ Computing shortest cycle free s-t Path is NP-Complete

[more on this next week]

or

Algo-Π

True|False: Suppose G has no negative cycles, then for every v, there is a shortest s-v path with at most $n-1$ edges.

A Dynamic Programming approach

Note: Input graph G may have negative edge weights but no negative cycles reachable from source.

Lemma 1: If G has no negative cycles, then there is a shortest path from s to t which has at most $n-1$ edges.

Proof: if the shortest path P from s to t has at least n edges then a vertex v appears more than once in P. Since each cycle has non-negative cost, we could remove the portion of P blw consecutive visits to v, resulting in a path of no greater cost and fewer edges.

Every Prefix of a Shortest Path is itself a Shortest Path.

Suppose the Shortest Paths from s to t is

$$s \to v_1 \to v_2 \dots \to v_m \to t \quad \text{then}$$

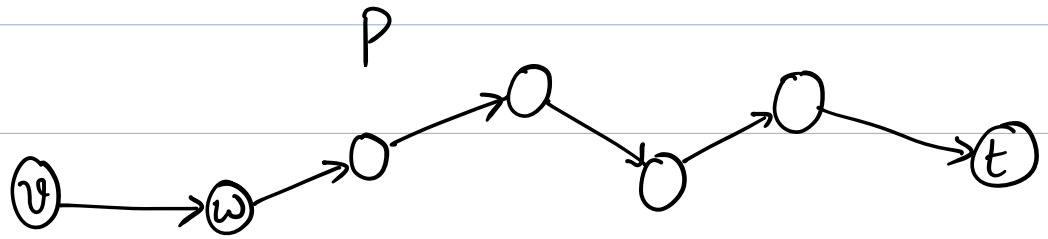
Every Prefix

$s \to v_1 \to \dots \to v_r$ is a shortest Path to v_r

$r \in \{1, \dots m\}$

Slightly easier version of the Problem

Given s & t find Shortest s to t Path.

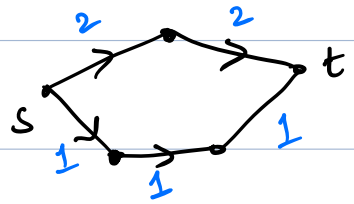


The minimum cost Path P from v to t using at most i edges.

Let OPT [i, v] denote the minimum cost v-t Path using atmost i edges.

From lemma 1, our original Problem is to compute OPT [n-1, s]

Example



$OPT[2, s] = 4$

$OPT[3, s] = 3.$

(3) [Optimal Substructure]

How to express $OPT[i,v]$ using Smaller Problems.

Let P be an optimal Path representing $OPT[i,v]$

① If P uses atmost $i-1$ edges then

$$OPT[i,v] = OPT[i-1,v]$$

② If P uses i edges, and the first edge is vw then

$$OPT[i,v] = w_{vw} + OPT[i-1, w]$$

∴ we get, for $i>0$,

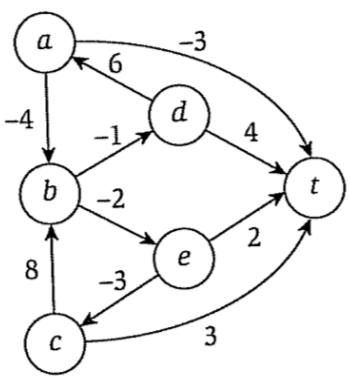
$$OPT[i,v] = \min\left\{ OPT[i-1,v],\ \min_{w \in V} \{OPT[i-1,w] + w_{vw}\} \right\} \quad \text{— ①}$$

using this reccurence, we get the following DP algorithm to find $OPT[n-1, s]$

Finds the minimum cost of an s-t Path

```
Shortest-Path(G, s, t)
  n = number of nodes in G
  Array M[0...n-1, V]
  Define M[0, t] = 0 and M[0, v] = ∞ for all other v ∈ V
  For i = 1, ..., n-1
    For v ∈ V in any order
      Compute M[i, v] using the recurrence (●●●●)
    Endfor
  Endfor
  Return M[n-1, s]
```

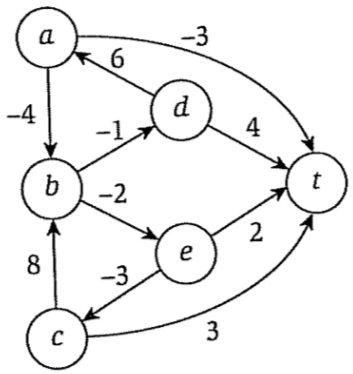
here we use M instead OPT



(a)

	0	1	2	3	4	5
t	0	0	0	0	0	0
a	∞	-3	-3			
b	∞	∞	0			
c	∞	3	3			
d	∞	4	3			
e	∞	2	0			

Example: Find a shortest path from each node to t.

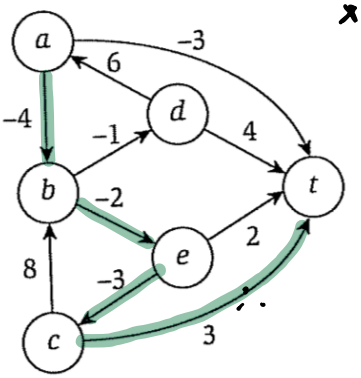


(a)

$M[i,v]$

	0	1	2	3	4	5
t	0	0	0	0	0	0
a	∞	-3	-3	-4	-6	-6
b	∞	∞	0	-2	-2	-2
c	∞	3	3	3	3	3
d	∞	4	3	3	2	0
e	∞	2	0	0	0	0

Table M



(a)

$a \rightsquigarrow t$

cost = $-4-2-3+3 = -6$

* A single row in the table corresponds to the shortest path from a particular node to t, as we allow the path to use an increasing number of edges.

* The last column indicates the cost of the shortest path from each node to t

Running time.

Table M has n^2 entries and each entry can take $O(n)$ time to complete, as there are at most n nodes WEV we have to consider.

$\therefore$ Algorithm runs in $O(n^3)$ time computes the minimum cost of an S-t path in any graph that has no negative cycles.

Negative cycles:

If the input graph G has a negative cycle, then we want our algorithm to report this fact.

(See the next page for details)

Checking For a Negative Cycle

Theorem:

If there is no negative cycles in G_1

$\Leftrightarrow$

$OPT(n,v) = OPT(n-1,v)$ for all v.

Proof: Forward direction :- follows from the Correctness of Bell-man ford. (lemma 1)

Reverse direction:

Assume $OPT[n-1,v] = OPT[n,v]$ for all $v \in V$

let $d(v) = OPT[n-1,v] = OPT[n,v]$

we know

$$OPT[n-1,v] = \min\{OPT[n-1,v], \min_{u \in E} \{OPT[n-1,u] + w_{vu}\}\}$$

($OPT[n-1,v]$ → $d(v)$; $OPT[n-1,u]$ → $d(u)$)

$$d(v) \leq d(u) + w_{vu}$$

$$w_{vu} \geq d(v) - d(u)$$

let $C = v_1 v_2 .. v_k v_1$ be an arbitrary cycle

Claim: $w(C) \geq 0$

$$w(C) = \sum_{i=1}^{k-1} w_{v_i v_{i+1}} + w_{v_k v_1}$$

$$\geq \sum_{i=1}^{k-1} d(v_i) - d(v_{i+1}) + d(v_k) - d(v_1)$$

$$\geq 0$$

With the above theorem,

Given an arbitrary graph G, we can modify Bellman-Ford algorithm to detect negative cycle by running it for one additional iteration.

Ref:
Coreman 24.1

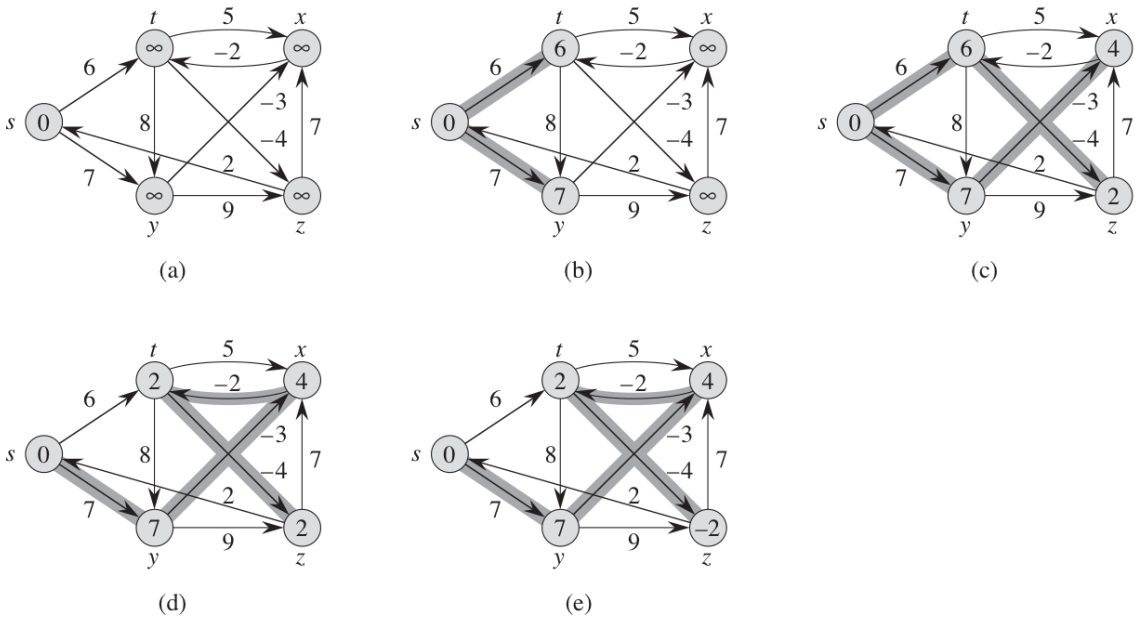
BELLMAN-FORD(G, w, s)

```
d(s) = 0
For each v ∈ V(G) - {s}
    d(v) = ∞
    v.π = NIL
For i = 1 to n-1
    For each edge uv ∈ E(G)
        if d(v) > d(u) + w_uv
            d(v) = d(u) + w_uv
            v.π = u
For each uv ∈ E(G)                    } checking for negative cycle
    if d(v) > d(u) + w_uv
        return FALSE
return TRUE
```

Running time : $O(VE)$

Example: Each pass we process the edges in the order

$tx, ty, tz, xt, yx, yz, zx, zs, st, sy.$



Algorithm returns TRUE for the above example.