EE360C ALGORITHMS - FALL 2018 - November 13 - Morning

Bellman-Ford Algorithm

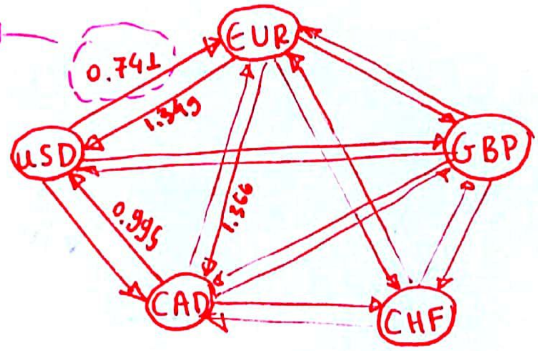
Shortest Paths with negative edge weights

We consider directed (although the same can be applied to undirected graphs as well), weighted graph.

could be negative

Want to find shortest s-t path.

you get 0.741 € for 1 $



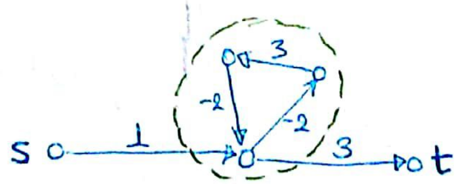
e.g. If you change 1 dollar to euro, and then to CAD, you will get $1 \cdot 0.741 \cdot 1.366$

if you then change back to usd, you will get

$1 \cdot 0.741 \cdot 1.366 \cdot 0.995$

$= 1.007144\ldots$

Negative Weight Cycles



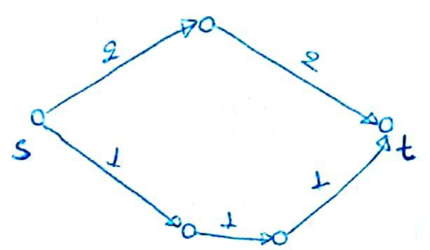
negative weight cycle

If $\exists$ s-t path with negative weight cycle, then length of shortest path $= -\infty$.

Otherwise, shortest path is simple. (does not repeat a node)

Subproblems: Reduce the number of edges in a path.

e.g.



- shortest path using ≤ 2 edges has length 4.
- shortest path using ≤ 3 edges has length 3.

$$\boxed{OPT(i,v) = \text{length of the shortest } v,t \text{ path using at most } i \text{ edges.}}$$

What we want: $OPT(\underset{\sim}{\text{large}}, s)$

$n-1 =$ ~~m = total number of edges in graph~~

because we are looking for simple paths, and a simple path in an n-node graph can have length at most $n-1$

i.e. # of edges

shortest v-t path using i edges:

$$OPT(i,v) = \min \left\{ \min_{\substack{w \text{ s.t.} \\ \exists \text{ edge } (v,w)}} \{ weight(v,w) + OPT(i-1,w) \} \; , \; OPT(i-1,v) \right\}$$

Base Cases:

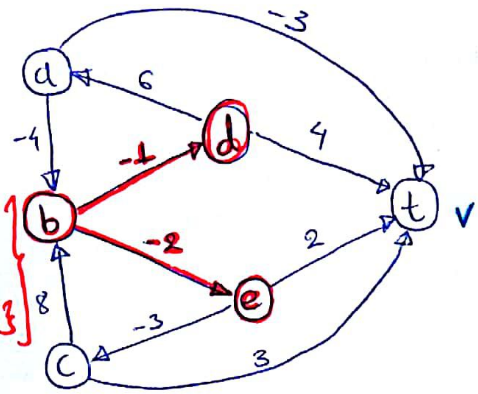
$$OPT(i,v) = \begin{cases} 0 & \text{if } i=0 \text{ \& } v=t \\ \infty & \text{if } i=0 \text{ \& } v \neq t \end{cases}$$

Example:

$$OPT(2,b) = \min\begin{cases} OPT(1,b), \\ \min\{-1+OPT(1,d),\ -2+OPT(1,e)\} \end{cases}$$

$$= \min\begin{cases} \infty, \\ \min\{3,0\} \end{cases}$$

$$= \min\{\infty, 0\} = 0.$$



OPT

v \ i	0	1	2	3	4	5 ($n-1$)
t	0	0	0			
a	∞	-3	-3			
b	∞	∞	0			
c	∞	3				
d	∞	4				
e	∞	2				

Work per column: $O(n+m)$

\# of columns: $O(n)$ ← $\boxed{\text{if } m>n}$

Total Work: $O(n^2 + m \cdot n)$

Work per column: $O(n)$

\# of columns: $O(m)$ ← $\boxed{\text{if } n>m}$

Total Work: $O(m \cdot n)$

vs. Dijkstra's Algorithm

HEAP: $O(n \log n)$

L-LIST: $O(n^2 + m)$

Dijkstra is "cheaper" but cannot handle negative edge weights!

Memory Usage:

Adjacency List: $O(n+m)$

OPT Table: $\cancel{O(n^2)}$ $O(n)$

Overall: $\cancel{O(n^2+m)}$ $O(n+m)$

Note that to calculate OPT(i,v), we need info only from "line" i-1 ↓ i.e. row