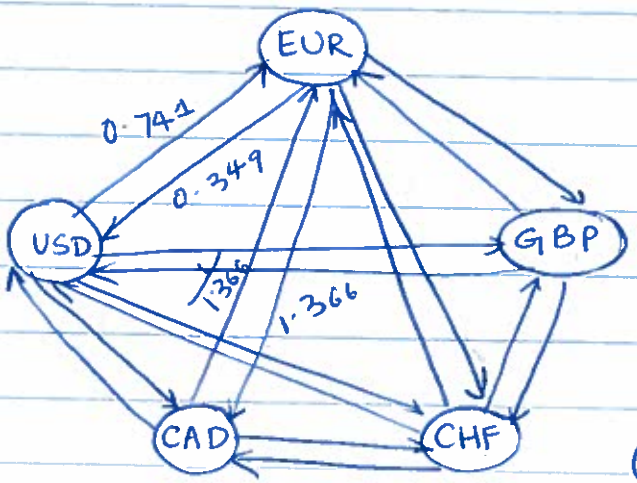
11/13/18

Dynamic Programming

→ Bellman-Ford Algorithm:

- Shortest paths with negative edge weights.
- Directed, weighted graph.
  - ↳ could be negative
- Want to find shortest s-t path.



Need to find the maximum rate

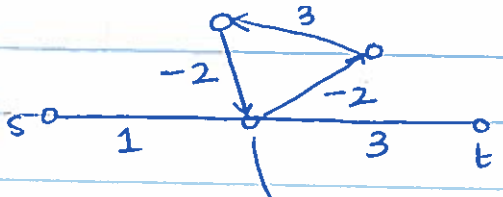
Say USD → EUR → C

0.741 × 1.366

↓

can find max with longest path (take log → and add changes to

- Negative Weight Cycle



net negative weight cycle

If ∃ s-t path with negative weight cycle then length of shortest path = -∞ (keep going over and over again through cycle)

Otherwise, shortest path is simple

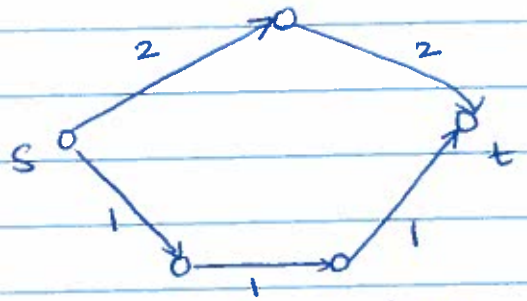
↳ does not repeat a node.

→ OPT(....) = { OPT(.....)

What are some of the natural subproblems

Subproblems: → Used in Bellman-Ford

- Restrict number of edges in a path.



Shortest path using ≤2 edges = 4

" " " ≤3 edges = 3

restrict #edges ↑ → subpath

BELLMAN-FORD ALGORITHM

- OPT(i,v) = length of the shortest v-t path using ≤ i edges.

What we want → OPT(large, s)

= n = total #edges

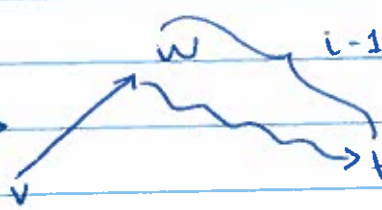
↓

n-1

why n-1?

- Because it is a simple path so visit each node only once.

Shortest v-t path with i edges ⇒



→ subproblem.

Set up recursion

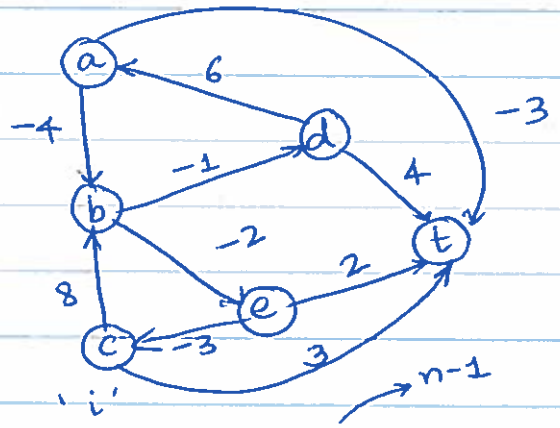
$$OPT(i,v) = \begin{cases} \min_{w \text{ such that there is edge } (v,w)} \{ \text{weight}(v,w) + OPT(i-1,w) \} \\ OPT(i-1,v) \end{cases}$$

↓ Choice to not take an extra edge.

Base Cases:

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

→ Example



OPT

'v'	0	1	2	3	4	5
t	0	0	0			
a	∞	-3	-3			
b	∞	∞	0			
c	∞	3				
d	∞	4				
e	∞	2				

→

$OPT(2,b) = \min\{\min\{-1 + OPT(\ldots$ (3, 4)

$-2 + OPT(1,\ldots$ (2)

$OPT(1,b)\}$ (0, ∞)

$= 0$

→ Time Complexity

Need to fill n^2 matrix → how much time is required per cell?

↳ Might be difficult to see, so look at work per column.

		If $m>n$	if $n>m$
Work per column:	$O(n+m)$	$O(m)$	$O(n)$
# of columns :	$O(n)$	$O(n)$	$O(m)$
Total Work :	$O(n^2+mn)$	$O(mn)$	$O(mn)$

↓ most common case.

~~If $m>n$~~

- V.S. Dijkstra's algo: → Better complexity, i.e. faster but can't handle negative edges.
  - Heap: $O(m \log n)$
  - L-list: $O(n^2+m)$

→ Memory usage of this algorithm:

Adjacency list : $O(n+m)$

OPT table : $O(n^2)$

Overall : $O(n^2+m)$

space complexity

? Can we improve on this?

- Only previous column needed to compute next column so we can store only that.

  we get $O(n+m) + O(n^2) = O(n^2+m)$

  $O(n)$ $O(n+m)$

- Negative weight cycle → arbitrage. (Example of where we can make money currency exchange)

  ↓

  Can we find this?