Shortest Path Problem

Input: $G = (V, E)$, $w : E \to \mathbb{R}$, $s, t \in V$, $|V| = n$, $|E| = m$, directed

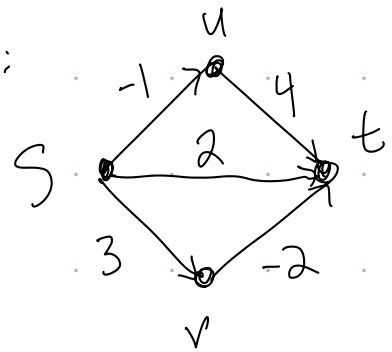
Output: Path P from s to t in G

e.g. $P = ((s,u), (u,v), \cdots (r,t))$ ← sequence of connected edges

such that $L(P) = \sum_{e \in P} w(e)$ is minimized

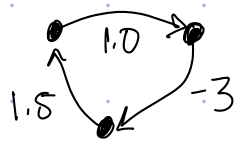
Example:

G:



Shortest Path: $((s,v), (v,t))$

Negative Cycle:



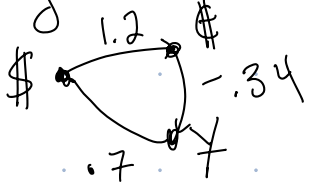
Not allowed

Various Approaches

- Greedy → Dijkstra's → edges have positive wt.
- Brute Force → Breadth-First Search → edges all have weight 1
- Dynamic → Bellman-Ford → edges can have negative wt
  → can work with global or distributed G
  → fails if must avoid neg. cycles

Applications:

- Data Routing in Networks:
- Bartering
- Arbitrage: Finding inefficiencies in financial markets



(Use variant to find a negative cycle.)

Stakeholders? Winners? Losers?
What do you want to do in warm weather?

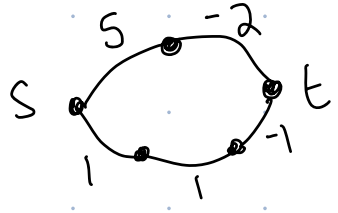
Bartering

Arbitrage

Defining Subproblems

Clever idea: $P_{u,i}$ = shortest path from s to u that uses at most i edges

What is $L(P_{t,1})$, $L(P_{t,2})$, $L(P_{t,3})$ for the following graph



A) ∞, 3, 1 B) ∞, 2, 3 C) 0, 2, 3 D) 0, 3, 1

Last choice a strategy makes? Vertex before t.

Designing a Dynamic Prog. Alg.

- Recurrence relation for $P_{u,i}$ = shortest path from s to u that uses at most i edges

$$P_{u,i} = \begin{cases} \underbrace{P_{v,i-1} + (v,u)}_{\text{append}} & \text{if shortest path with } i \text{ edges goes through } v \text{ prior to } u \\ P_{w,i-1} + (w,u) & \text{if shortest path with } i \text{ edges goes through } w \text{ prior to } u \\ \vdots & \\ P_{u,i-1} & \text{if shortest path to } u \text{ uses less than } i \text{ edges} \end{cases}$$

- Recurrence relation for objective function ($w(u,u) = 0$, $w(u,v) = \infty$ if no edge)

$$L(P_{u,i}) = \begin{cases} \min_{v \in V} \{ L(P_{v,i-1}) + w(v,u) \} \\ \text{Base case: } \begin{array}{l} L(P_{u,0}) = \infty \quad \text{if } u \neq s \\ L(P_{s,0}) = 0 \end{array} \end{cases}$$

Helpful notation $\rightarrow \min_{u \in S} \{ f(u) \}$

- Write Pseudocode to fill in A

Input: $n \times n$ array w where $w[u,v]$ = weight of edge (u,v), $s, t \in [n]$.
$w[u,v] = \infty$ if no edge. $w[u,u] = 0 \ \forall u \in [n]$

Output: ?x? array A s.t. $A[u,i] = L(P_{u,i})$
$n \times n$ ← n vertices; i goes from 0 length to $n-1$ length

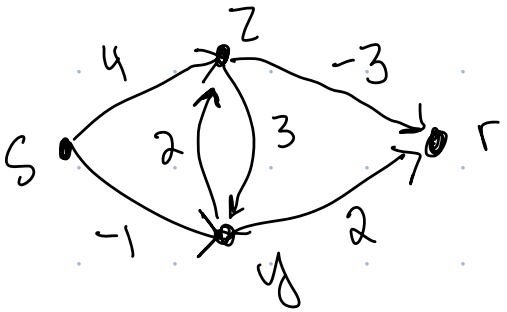
Initialize $A \leftarrow n \times n$ array filled with ∞'s.

$A[s,0] \leftarrow 0$
For $i \leftarrow 1$ to $n-1$:
  For $v \leftarrow 1$ to n:
    For $u \leftarrow 1$ to n:
      If $A[v,i] > A[u,i-1] + w[u,v]$:
        $A[v,i] \leftarrow A[u,i-1] + w[u,v]$

} Iterate over elements of A

} Fill out $A[v,i]$ by taking minimum over all options

Use your code to create A for the graph



What is the Runtime of Bellman-Ford?

A) $O(n)$ B) $O(n^2)$ C) $O(n^3)$ D) $O(nm)$ (n = # vertices, m = # edges)

Recall

Recurrence relation for objective function ($w(u,u) = 0$, $w(u,v) = \infty$ if no edge)

$$L(P_{u,i}) = \begin{cases} \min_{v \in V} \{ L(P_{v,i-1}) + w(v,u) \} \\ \text{Base case: } \begin{array}{l} L(P_{u,0}) = \infty \text{ if } u \neq s \\ L(P_{s,0}) = 0 \end{array} \end{cases}$$

Don't want to check every $v \in V$. Only want to check $v : (v,u) \in E$.

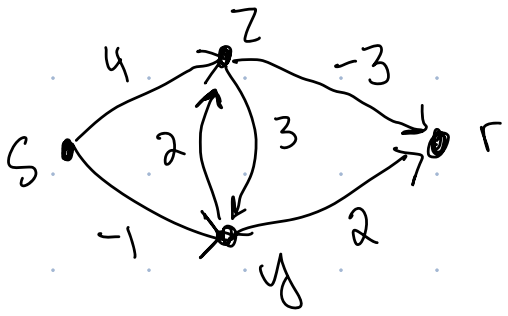
Reverse Adjacency List

$W[s] = \phi$

$W[z] = ((s,4),(y,2))$

$W[r] = ((z,-3),(y,2))$

$\vdots$



e.g.

$W[z,1] = (s,4)$ → learn that edge from s to z with weight 4.

Initialize $A \leftarrow n \times n$ array filled with ∞'s.

$A[s,0] \leftarrow 0$

For $i \leftarrow 1$ to $n-1$: } Iterate over elts of A

  For v 1 to n: }

    For u: $(u,v) \in E$:

      If $A[v,i] > A[u,i-1] + w[u,v]$:

        $A[v,i] \leftarrow A[u,i-1] + w[u,v]$

can do efficiently with reverse adj matrix

What is the Runtime of Bellman-Ford with Reverse Adj Matrix?

A) $O(n)$ B) $O(n^2)$ C) $O(n^3)$ **D)** $O(nm)$

(n = # vertices, m = # edges)

Worst

Best

In worst case, $m = O(n^2)$,
but in best case $m = O(n)$

Initialize $A \leftarrow n \times n$ array filled with ∞'s.

$A[s,0] \leftarrow 0$

For $i \leftarrow 1$ to $n-1$: $\leftarrow O(n)$

  For $v \leftarrow 1$ to n:

    For u: $(u,v) \in E$:

      If $A[v,i] > A[u,i-1] + w[u,v]$:

        $A[v,i] \leftarrow A[u,i-1] + w[u,v]$

Get all edges pointing at v_1
Get all edges pointing at v_2
⋮
} each edge appears once!

$O(m)$

Distributed Bellman-Ford

Vertex v can calculate $A[v,i]$ if gets passed $A[u,i-1]$ info from neighbor u: $(u,v) \in E$.
Then it passes $A[v,i]$ onto its neighbors.

Useful for network