Paths in Graphs: Bellman-Ford Algorithm

Michael Levin

Department of Computer Science and Engineering
University of California, San Diego

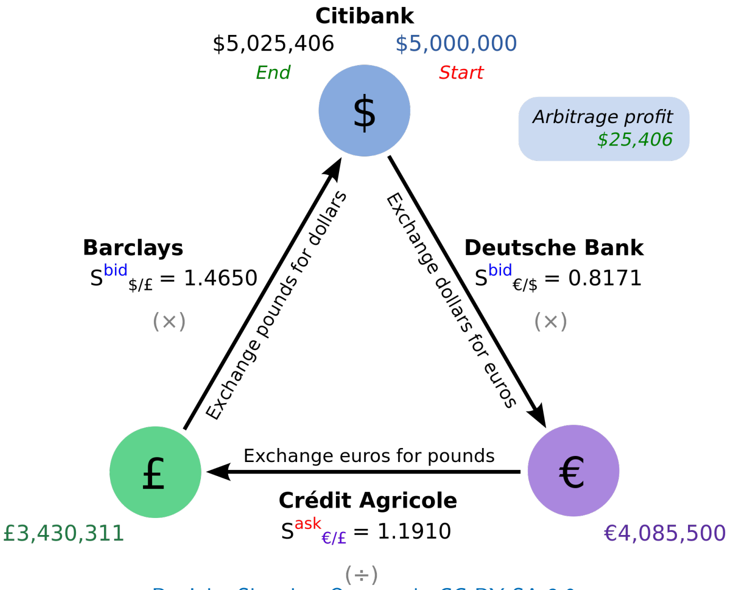
Graph Algorithms
Algorithms and Data Structures

Outline

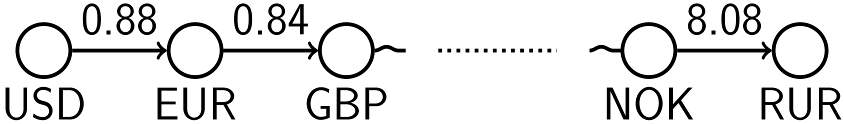
Currency Exchange

You can convert some currencies into some others with given exchange rates. What is the maximum amount in Russian rubles you can get from 1000 US dollars using unlimited number of currency conversions? Is it possible to get as many Russian rubles as you want? Is it possible to get as many US dollars as you want?

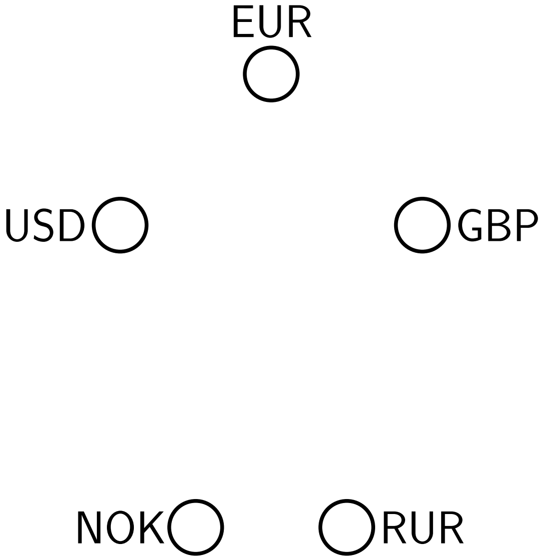
Arbitrage

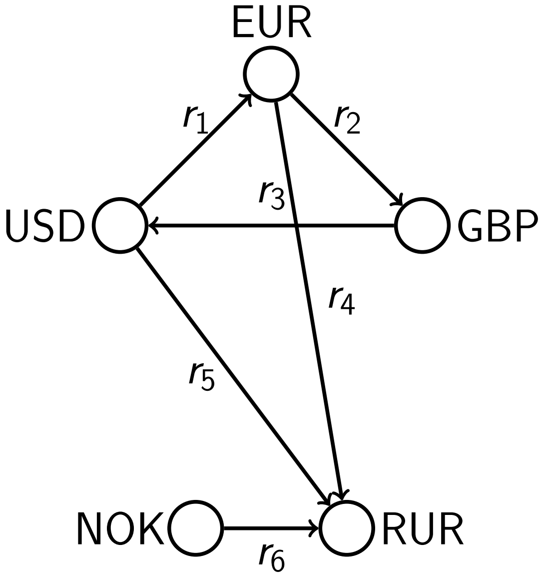


Citibank
$5,025,406
$5,000,000
End
Start
$
Arbitrage profit
$25,406
Exchange pounds for dollars
Exchange dollars for euros
Barclays
S^bid $/£ = 1.4650
(×)
Deutsche Bank
S^bid €/$ = 0.8171
(×)
£
Exchange euros for pounds
€
Crédit Agricole
S^ask €/£ = 1.1910
£3,430,311
€4,085,500
(÷)



$$1 \text{ USD} \rightarrow 0.88 \cdot 0.84 \cdot \ldots \cdot 8.08 \text{ RUR}$$


EUR
USD
GBP
NOK
RUR


EUR
USD
GBP
NOK
RUR
r_1
r_2
r_3
r_4
r_5
r_6

Maximum product over paths

Input: Currency exchange graph with weighted directed edges e_i between some pairs of currencies with weights r_{e_i} corresponding to the exchange rate.

Output: Maximize $\prod_{j=1}^{k} r_{e_j} = r_{e_1} r_{e_2} \dots r_{e_k}$ over paths $(e_1, e_2, \dots, e_k)$ from USD to RUR in the graph.

Outline

1. Currency Exchange
2. Reduction to Shortest Paths
3. Bellman–Ford algorithm
4. Proof of Correctness
5. Negative Cycles
6. Infinite Arbitrage

Maximum product over paths

Input: Currency exchange graph with weighted directed edges e_i between some pairs of currencies with weights r_{e_i} corresponding to the exchange rate.

Output: Maximize $\prod_{j=1}^{k} r_{e_j} = r_{e_1} r_{e_2} \dots r_{e_k}$ over paths $(e_1, e_2, \dots, e_k)$ from USD to RUR in the graph.

Reduction to shortest paths

Use two standard approaches:

- Replace product with sum by taking logarithms of weights

Reduction to shortest paths

Use two standard approaches:

- Replace product with sum by taking logarithms of weights
- Negate weights to solve minimization problem instead of maximization problem

Taking the Logarithm

$$xy = 2^{\log_2(x)} 2^{\log_2(y)} = 2^{\log_2(x) + \log_2(y)}$$

Taking the Logarithm

$$xy = 2^{\log_2(x)} 2^{\log_2(y)} = 2^{\log_2(x) + \log_2(y)}$$

$$xy \to \max \Leftrightarrow \log_2(x) + \log_2(y) \to \max$$

Taking the Logarithm

$$xy = 2^{\log_2(x)} 2^{\log_2(y)} = 2^{\log_2(x) + \log_2(y)}$$

$$xy \to \max \Leftrightarrow \log_2(x) + \log_2(y) \to \max$$

$4 \times 1 \times \frac{1}{2} = 2 = 2^1$

$\log_2(4) + \log_2(1) + \log_2(\frac{1}{2}) = 2 + 0 + (-1) = 1$

Taking the Logarithm

$$xy = 2^{\log_2(x)} 2^{\log_2(y)} = 2^{\log_2(x)+\log_2(y)}$$

$$xy \to \max \Leftrightarrow \log_2(x) + \log_2(y) \to \max$$

$4 \times 1 \times \frac{1}{2} = 2 = 2^1$

$\log_2(4)+\log_2(1)+\log_2(\frac{1}{2}) = 2+0+(-1) = 1$

$$\prod_{j=1}^{k} r_{e_j} \to \max \Leftrightarrow \sum_{j=1}^{k} \log(r_{e_j}) \to \max$$

Negation

$$\sum_{j=1}^{k} \log(r_{e_j}) \to \max \Leftrightarrow -\sum_{j=1}^{k} \log(r_{e_j}) \to \min$$

Negation

$$\sum_{j=1}^{k} \log(r_{e_j}) \to \max \Leftrightarrow -\sum_{j=1}^{k} \log(r_{e_j}) \to \min$$

$$\sum_{j=1}^{k} \log(r_{e_j}) \to \max \Leftrightarrow \sum_{j=1}^{k} (-\log(r_{e_j})) \to \min$$

Reduction

Finally: replace edge weights r_{e_i} by $(-\log(r_{e_i}))$ and find the shortest path between USD and RUR in the graph.

Solved?

- Create currency exchange graph with weights r_{e_i} corresponding to exchange rates

Solved?

- Create currency exchange graph with weights r_{e_i} corresponding to exchange rates
- Replace $r_{e_i} \to (-\log(r_{e_i}))$

Solved?

- Create currency exchange graph with weights r_{e_i} corresponding to exchange rates
- Replace $r_{e_i} \to (-\log(r_{e_i}))$
- Find the shortest path from USD to RUR by Dijkstra's algorithm

Solved?

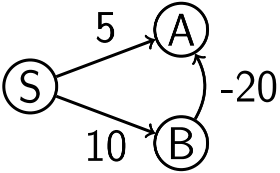
- Create currency exchange graph with weights r_{e_i} corresponding to exchange rates
- Replace $r_{e_i} \to (-\log(r_{e_i}))$
- Find the shortest path from USD to RUR by Dijkstra's algorithm
- Do the exchanges corresponding to the shortest path

Where Dijkstra's algorithm goes wrong?

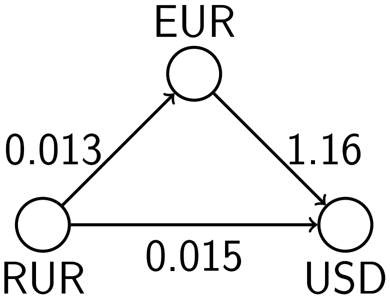
- Dijkstra's algorithm relies on the fact that a shortest path from s to t goes only through vertices that are closer to s than t.

Where Dijkstra's algorithm goes wrong?

- Dijkstra's algorithm relies on the fact that a shortest path from s to t goes only through vertices that are closer to s than t.
- This is no longer the case for graphs with negative edges:

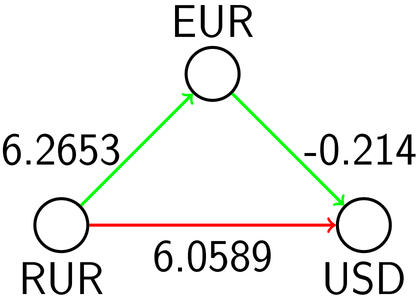


Currency exchange example



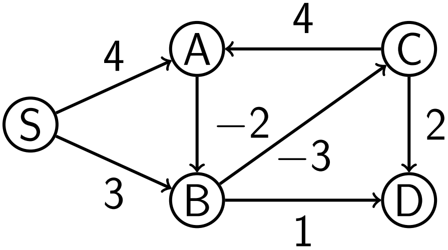
$$0.013 \times 1.16 = 0.01508 > 0.015$$

Currency exchange example

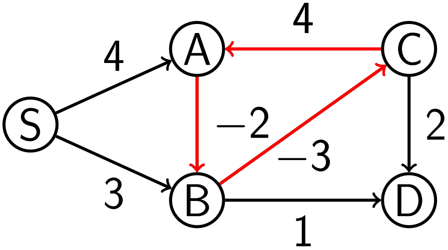


$$0.013 \times 1.16 = 0.01508 > 0.015$$

Negative weight cycles

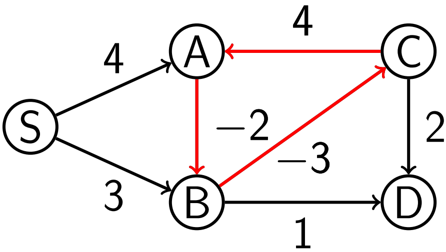


Negative weight cycles



$$d(S, A) = d(S, B) = d(S, C) = d(S, D) = -\infty$$

Negative weight cycles



$d(S, A) = d(S, B) = d(S, C) = d(S, D) = -\infty$

In currency exchange, a negative cycle can make you a billionaire!

Outline

Naive algorithm

- Remember naive algorithm from the previous lesson?

Naive algorithm

- Remember naive algorithm from the previous lesson?
- Relax edges while dist changes

Naive algorithm

- Remember naive algorithm from the previous lesson?
- Relax edges while dist changes
- Turns out it works even for negative edge weights!

$\texttt{Relax}((u, v) \in E)$

$\texttt{if}\;\; \mathit{dist}[v] > \mathit{dist}[u] + w(u, v)\texttt{:}$

$\quad \mathit{dist}[v] \leftarrow \mathit{dist}[u] + w(u, v)$

$\quad \mathit{prev}[v] \leftarrow u$

Bellman–Ford algorithm

```
BellmanFord(G, S)
```

$\{$`no negative weight cycles in` $G\}$

`for all` $u \in V$:

  `dist`$[u] \leftarrow \infty$

  `prev`$[u] \leftarrow$ `nil`

$dist[S] \leftarrow 0$

`repeat` $|V| - 1$ `times`:

  `for all` $(u, v) \in E$:

    `Relax`(u, v)

Bellman–Ford algorithm

```
BellmanFord(G, S)

{no negative weight cycles in G}
for all u ∈ V:
  dist[u] ← ∞
  prev[u] ← nil
dist[S] ← 0
repeat |V| − 1 times:
  for all (u, v) ∈ E:
    Relax(u, v)
```

Bellman–Ford algorithm

```
BellmanFord(G, S)

{no negative weight cycles in G}
for all u ∈ V:
   dist[u] ← ∞
   prev[u] ← nil
dist[S] ← 0
repeat |V| − 1 times:
   for all (u, v) ∈ E:
      Relax(u, v)
```

Bellman–Ford algorithm

BellmanFord(G, S)

```
{no negative weight cycles in G}
for all u ∈ V:
    dist[u] ← ∞
    prev[u] ← nil
dist[S] ← 0
repeat |V| − 1 times:
    for all (u, v) ∈ E:
        Relax(u, v)
```

Bellman–Ford algorithm

BellmanFord(G, S)

```
{no negative weight cycles in G}
for all u ∈ V:
    dist[u] ← ∞
    prev[u] ← nil
dist[S] ← 0
repeat |V| − 1 times:
    for all (u, v) ∈ E:
        Relax(u, v)
```

Bellman–Ford algorithm

```
BellmanFord(G, S)

{no negative weight cycles in G}
for all u ∈ V:
    dist[u] ← ∞
    prev[u] ← nil
dist[S] ← 0
repeat |V| − 1 times:
    for all (u, v) ∈ E:
        Relax(u, v)
```

Bellman–Ford algorithm

$\texttt{BellmanFord}(G, S)$

```
{no negative weight cycles in G}
for all u ∈ V:
    dist[u] ← ∞
    prev[u] ← nil
dist[S] ← 0
repeat |V| − 1 times:
    for all (u, v) ∈ E:
        Relax(u, v)
```

Bellman–Ford algorithm

BellmanFord(G, S)

```
{no negative weight cycles in G}
for all u ∈ V:
    dist[u] ← ∞
    prev[u] ← nil
dist[S] ← 0
repeat |V| − 1 times:
    for all (u, v) ∈ E:
        Relax(u, v)
```

Running Time

Lemma

The running time of Bellman–Ford algorithm is $O(|V||E|)$.

Proof

- Initialize dist — $O(|V|)$

Running Time

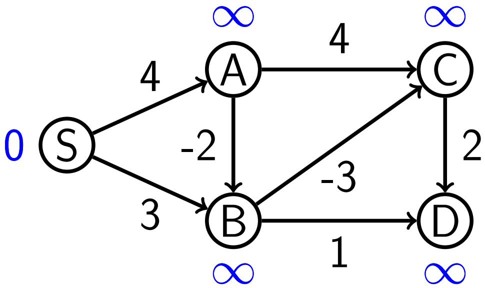
Lemma

The running time of Bellman–Ford algorithm is $O(|V||E|)$.

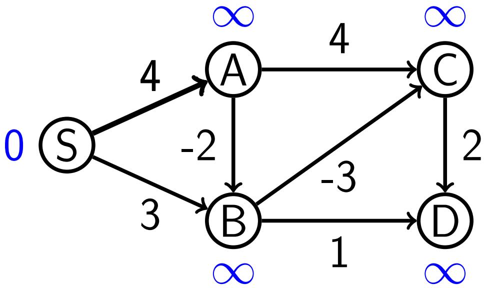
Proof

- Initialize dist — $O(|V|)$
- $|V| - 1$ iterations, each $O(|E|)$ — $O(|V||E|)$ $\square$

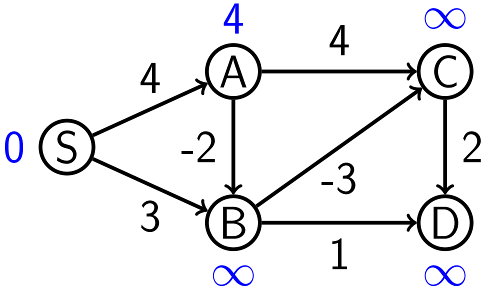
Example



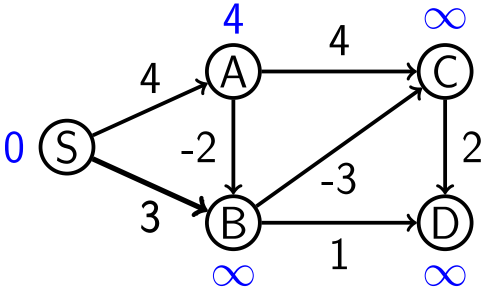
Example



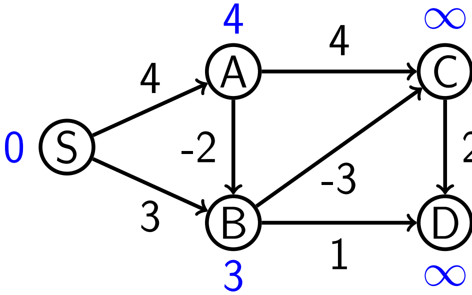
Example



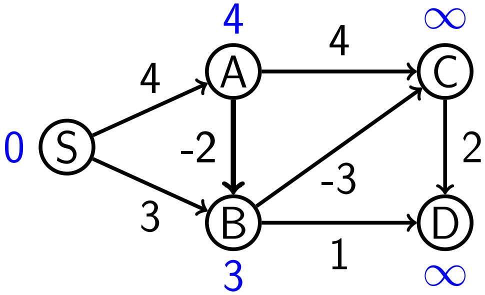
Example



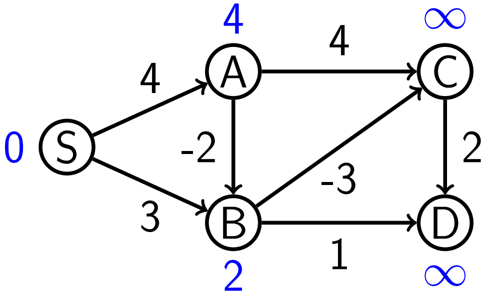
Example



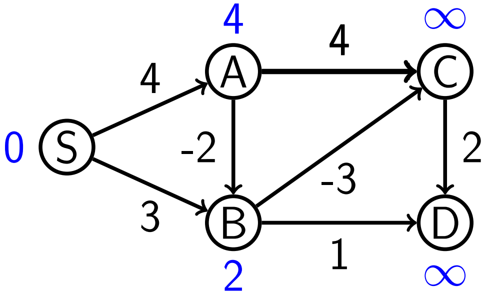
Example



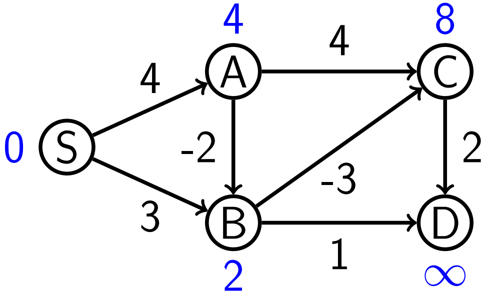
Example



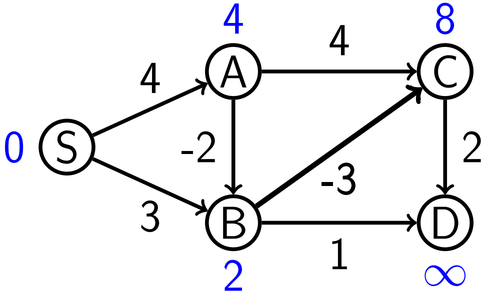
Example



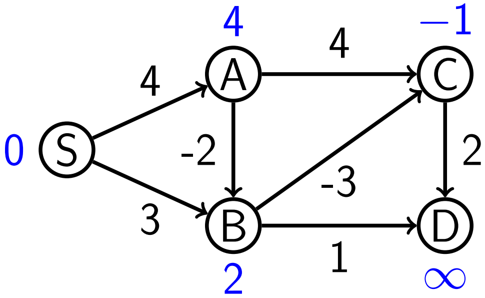
Example



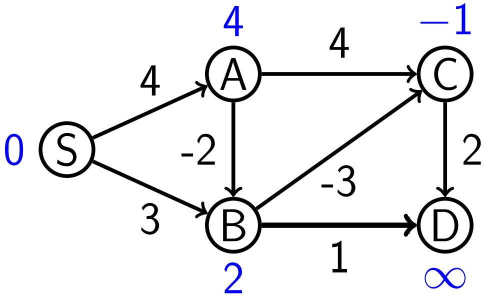
Example



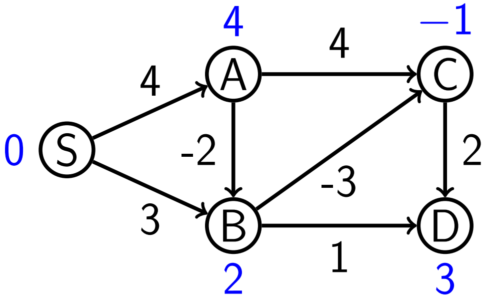
Example



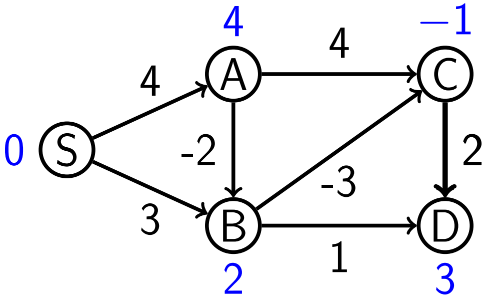
Example



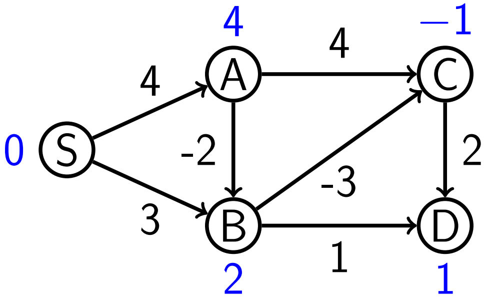
Example



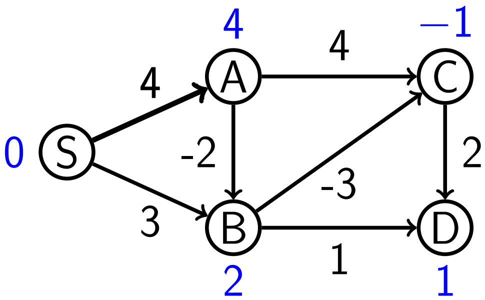
Example



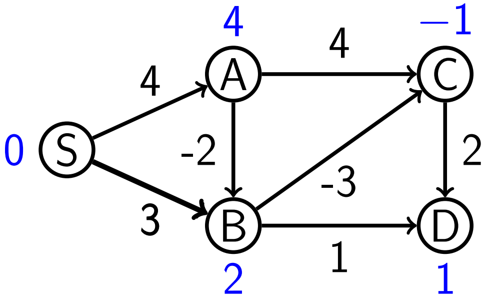
Example



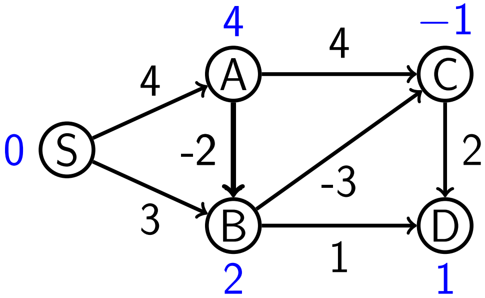
Example



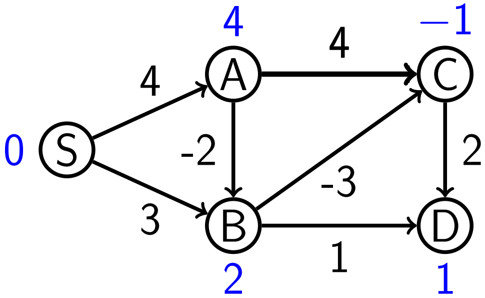
Example



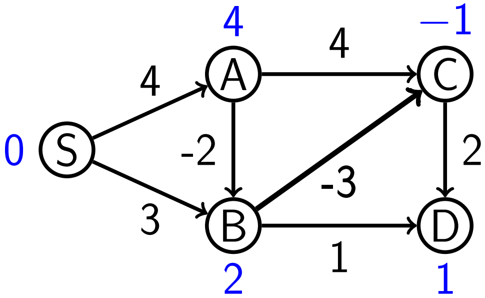
Example



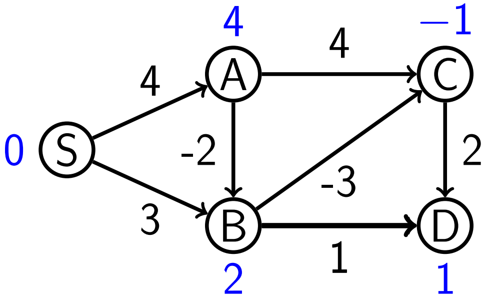
Example



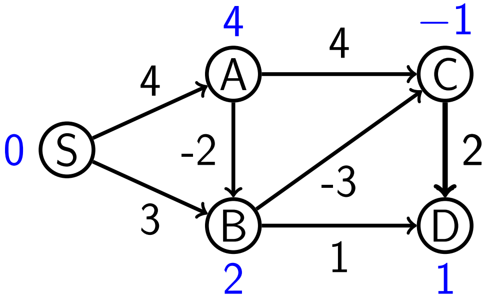
Example



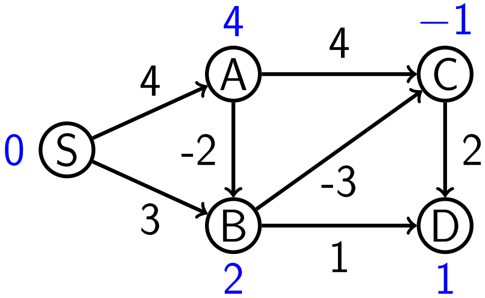
Example



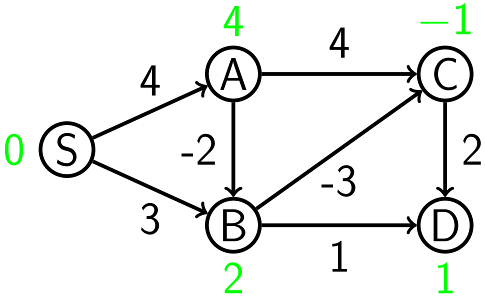
Example



Example



Example



Outline

Lemma

After k iterations of relaxations, for any node u, $\text{dist}[u]$ is the smallest length of a path from S to u that contains at most k edges.

Proof

- Use mathematical induction

Proof

- Use mathematical induction
- Base: after 0 iterations, all dist-values are ∞, but for dist$[S] = 0$, which is correct.

Proof

- Use mathematical induction
- Base: after 0 iterations, all dist-values are ∞, but for $\text{dist}[S] = 0$, which is correct.
- Induction: proved for $k \rightarrow$ prove for $k + 1$

Proof

- Before $k+1$-th iteration, $\mathrm{dist}[u]$ is the smallest length of a path from S to u containing at most k edges

Proof

- Before $k+1$-th iteration, $\mathrm{dist}[u]$ is the smallest length of a path from S to u containing at most k edges
- Each path from S to u goes through one of the incoming edges (v, u)

Proof

- Before $k+1$-th iteration, dist$[u]$ is the smallest length of a path from S to u containing at most k edges
- Each path from S to u goes through one of the incoming edges (v, u)
- Relaxing by (v, u) is comparing it with the smallest length of a path from S to u through v containing at most $k+1$ edge $\square$

Corollary

In a graph without negative weight cycles, Bellman–Ford algorithm correctly finds all distances from the starting node S.

Proof

- Any path with at least V edges contains a cycle

Proof

- Any path with at least V edges contains a cycle
- This cycle can be removed without making the path longer

Proof

- Any path with at least V edges contains a cycle
- This cycle can be removed without making the path longer
- Shortest path contains at most $V - 1$ edges and will be found after $V - 1$ iterations

Corollary

If there is no negative weight cycle reachable from S such that u is reachable from this negative weight cycle, Bellman–Ford algorithm correctly finds $dist[u] = d(S, u)$.

Outline

Negative weight cycles

Lemma

A graph G contains a negative weight cycle if and only if $|V|$-th (additional) iteration of `BellmanFord`(G, S) updates some `dist`-value.

Proof

$\Leftarrow$ If there are no negative cycles, then all shortest paths from S contain at most $|V| - 1$ edges, so no `dist`-value can be updated on $|V|$-th iteration.

Proof

$\Rightarrow$ There's a negative weight cycle, say $a \rightarrow b \rightarrow c \rightarrow a$, but no relaxations on $|V|$-th iteration.

Proof

$\Rightarrow$ There's a negative weight cycle, say $a \to b \to c \to a$, but no relaxations on $|V|$-th iteration.

$$\texttt{dist}[b] \leq \texttt{dist}[a] + w(a, b)$$
$$\texttt{dist}[c] \leq \texttt{dist}[b] + w(b, c)$$
$$\texttt{dist}[a] \leq \texttt{dist}[c] + w(c, a)$$

Proof

$$\Rightarrow \texttt{dist}[b] + \texttt{dist}[c] + \texttt{dist}[a] \leq \texttt{dist}[a] + w(a, b) + \texttt{dist}[b] + w(b, c) + \texttt{dist}[c] + w(c, a)$$

Proof

$\Rightarrow$ $\texttt{dist}[b] + \texttt{dist}[c] + \texttt{dist}[a] \le \texttt{dist}[a] + w(a, b) + \texttt{dist}[b] + w(b, c) + \texttt{dist}[c] + w(c, a)$

$w(a, b) + w(b, c) + w(c, a) \ge 0$ — a contradiction as $a \to b \to c \to a$ is a negative cycle. $\square$

Finding Negative Cycle

Algorithm:

- Run $|V|$ iterations of Bellman–Ford algorithm, save node v relaxed on the last iteration

Finding Negative Cycle

Algorithm:

- Run $|V|$ iterations of Bellman–Ford algorithm, save node v relaxed on the last iteration
- v is reachable from a negative cycle

Finding Negative Cycle

Algorithm:

- Run $|V|$ iterations of Bellman–Ford algorithm, save node v relaxed on the last iteration
- v is reachable from a negative cycle
- Start from $x \leftarrow v$, follow the link $x \leftarrow \text{prev}[x]$ for $|V|$ times — will be definitely on the cycle

Finding Negative Cycle

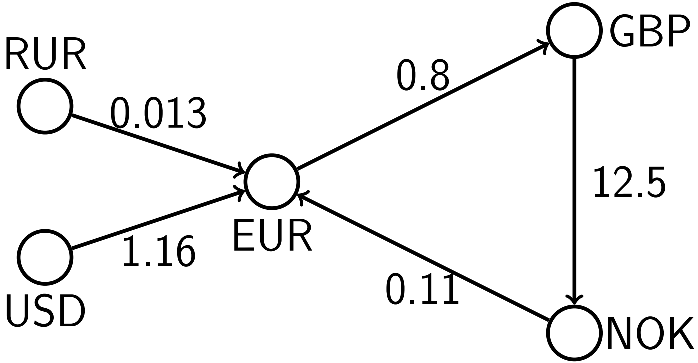
Algorithm:

- Run $|V|$ iterations of Bellman–Ford algorithm, save node v relaxed on the last iteration
- v is reachable from a negative cycle
- Start from $x \leftarrow v$, follow the link $x \leftarrow \text{prev}[x]$ for $|V|$ times — will be definitely on the cycle
- Save $y \leftarrow x$ and go $x \leftarrow \text{prev}[x]$ until $x = y$ again

Is it possible to get as many rubles as you want from 1000 USD?

Is it possible to get as many rubles as you want from 1000 USD?

Not always, even if there is a negative cycle



Cannot exchange USD into rubles via (negative) cycle EUR $\rightarrow$ GBP $\rightarrow$ NOK.

Outline

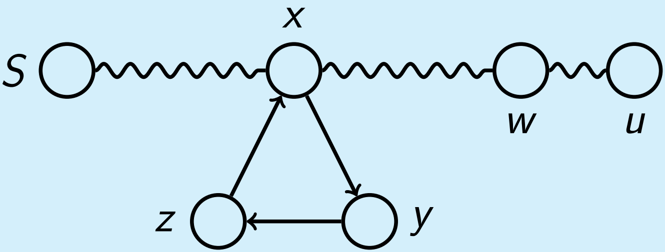
Detect Infinite Arbitrage

Lemma

It is possible to get any amount of currency u from currency S if and only if u is reachable from some node w for which dist$[w]$ decreased on iteration V of Bellman-Ford.

Proof

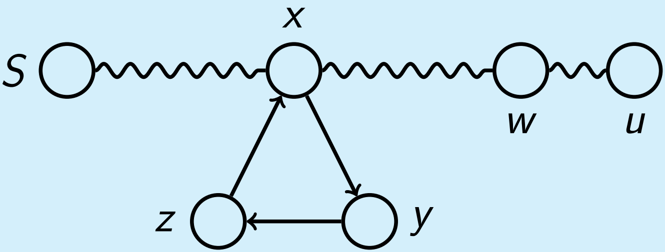
($\Leftarrow$)



- dist[w] decreased on iteration $V \Rightarrow w$ is reachable from a negative weight cycle

Proof

$(\Leftarrow)$



- dist$[w]$ decreased on iteration $V \Rightarrow w$ is reachable from a negative weight cycle
- w is reachable $\Rightarrow u$ is also reachable $\Rightarrow$ infinite arbitrage $\square$

Proof

($\Rightarrow$)

- Let L be the length of the shortest path to u with at most $V-1$ edges

Proof

($\Rightarrow$)

- Let L be the length of the shortest path to u with at most $V-1$ edges
- After $V-1$ iterations, dist[u] is equal to L

Proof

$(\Rightarrow)$

- Let L be the length of the shortest path to u with at most $V - 1$ edges
- After $V - 1$ iterations, dist[u] is equal to L
- Infinite arbitrage to $u \Rightarrow$ there exists a path shorter than L

Proof

($\Rightarrow$)

- Let L be the length of the shortest path to u with at most $V - 1$ edges
- After $V - 1$ iterations, dist[u] is equal to L
- Infinite arbitrage to $u \Rightarrow$ there exists a path shorter than L
- Thus dist[u] will be decreased on some iteration $k \geq V$

Proof

($\Rightarrow$ continued)

- If edge (x, y) was not relaxed and dist$[x]$ did not decrease on i-th iteration, then edge (x, y) will not be relaxed on $i + 1$-st iteration

Proof

($\Rightarrow$ continued)

- If edge (x, y) was not relaxed and dist$[x]$ did not decrease on i-th iteration, then edge (x, y) will not be relaxed on $i + 1$-st iteration
- Only nodes reachable from those relaxed on previous iterations can be relaxed

Proof

($\Rightarrow$ continued)

- If edge (x, y) was not relaxed and dist[x] did not decrease on i-th iteration, then edge (x, y) will not be relaxed on $i + 1$-st iteration
- Only nodes reachable from those relaxed on previous iterations can be relaxed
- dist[u] is decreased on iteration $k \geq V \Rightarrow u$ is reachable from some node relaxed on V-th iteration $\square$

Detect Infinite Arbitrage

- Do $|V|$ iterations of Bellman–Ford, save all nodes relaxed on V-th iteration — set A

Detect Infinite Arbitrage

- Do $|V|$ iterations of Bellman–Ford, save all nodes relaxed on V-th iteration — set A
- Put all nodes from A in queue Q

Detect Infinite Arbitrage

- Do $|V|$ iterations of Bellman–Ford, save all nodes relaxed on V-th iteration — set A
- Put all nodes from A in queue Q
- Do breadth-first search with queue Q and find all nodes reachable from A

Detect Infinite Arbitrage

- Do $|V|$ iterations of Bellman–Ford, save all nodes relaxed on V-th iteration — set A
- Put all nodes from A in queue Q
- Do breadth-first search with queue Q and find all nodes reachable from A
- All those nodes and only those can have infinite arbitrage

Reconstruct Infinite Arbitrage

- During Breadth-First Search, remember the parent of each visited node

Reconstruct Infinite Arbitrage

- During Breadth-First Search, remember the parent of each visited node
- Reconstruct the path to u from some node w relaxed on iteration V

Reconstruct Infinite Arbitrage

- During Breadth-First Search, remember the parent of each visited node
- Reconstruct the path to u from some node w relaxed on iteration V
- Go back from w to find negative cycle from which w is reachable

Reconstruct Infinite Arbitrage

- During Breadth-First Search, remember the parent of each visited node
- Reconstruct the path to u from some node w relaxed on iteration V
- Go back from w to find negative cycle from which w is reachable
- Use this negative cycle to achieve infinite arbitrage from S to u

Conclusion

- Can implement best possible exchange rate
- Can determine whether infinite arbitrage is possible
- Can implement infinite arbitrage
- Can find shortest paths in graphs with negative edge weights
- Can detect and find negative cycles in graphs