Paths in Graphs: Currency Exchange

Michael Levin

Higher School of Economics

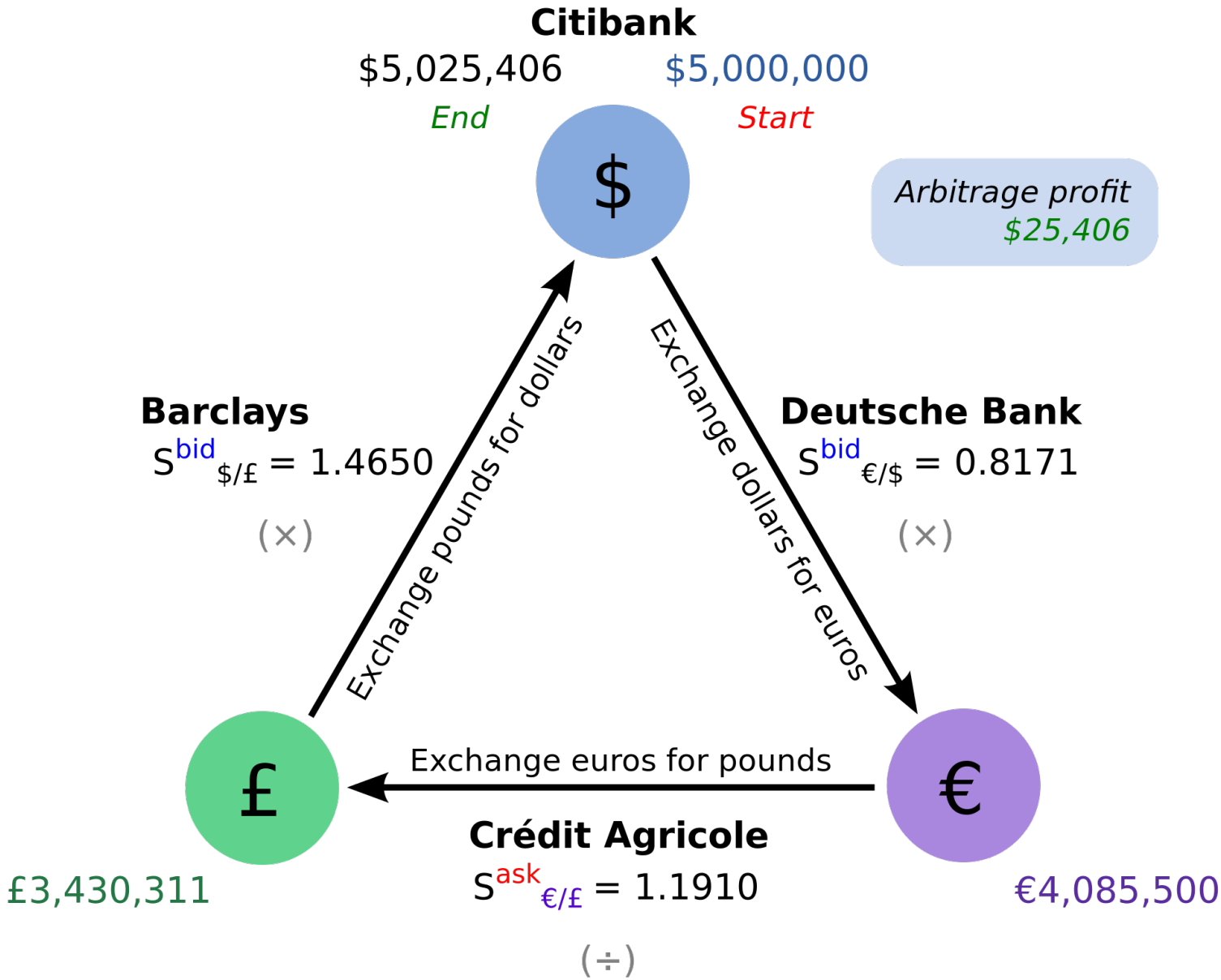
Graph Algorithms
Data Structures and Algorithms

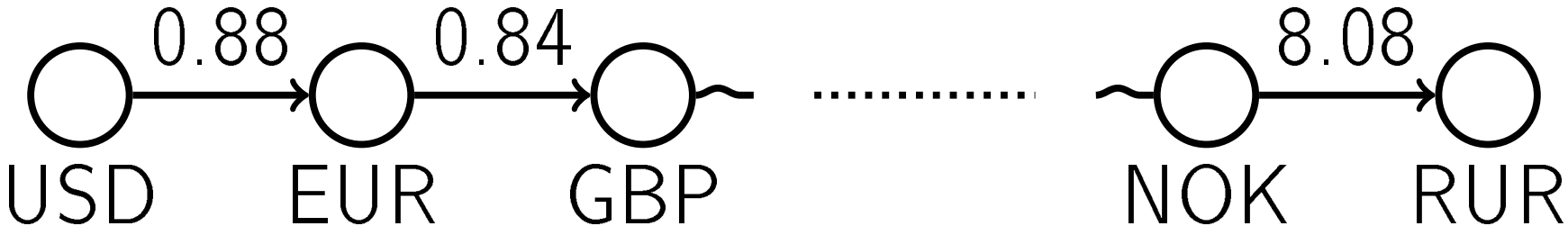
Outline

Currency Exchange

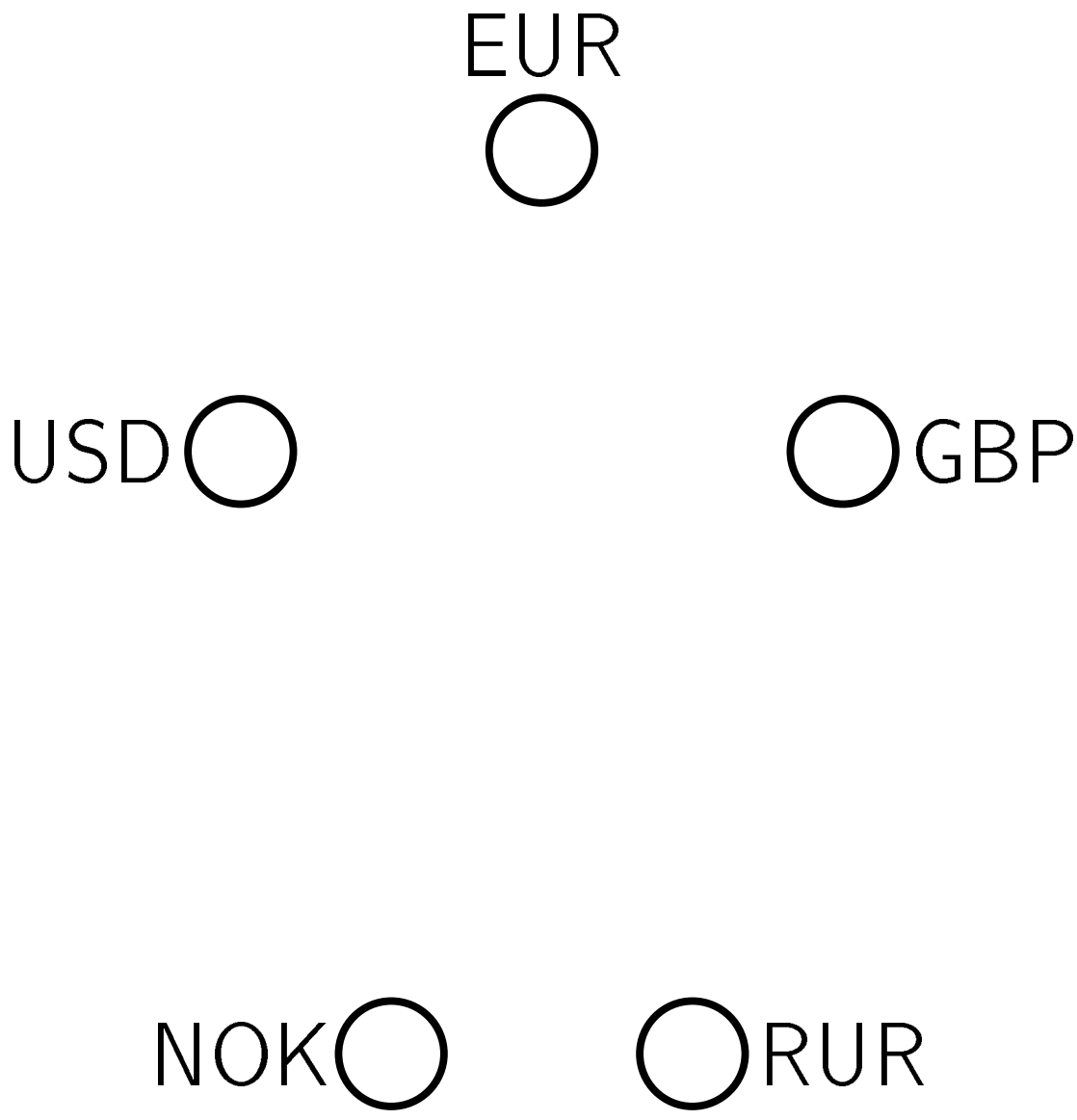
You can convert some currencies into some others with given exchange rates. What is the maximum amount in Russian rubles you can get from 1000 US dollars using unlimited number of currency conversions? Is it possible to get as many Russian rubles as you want? Is it possible to get as many US dollars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

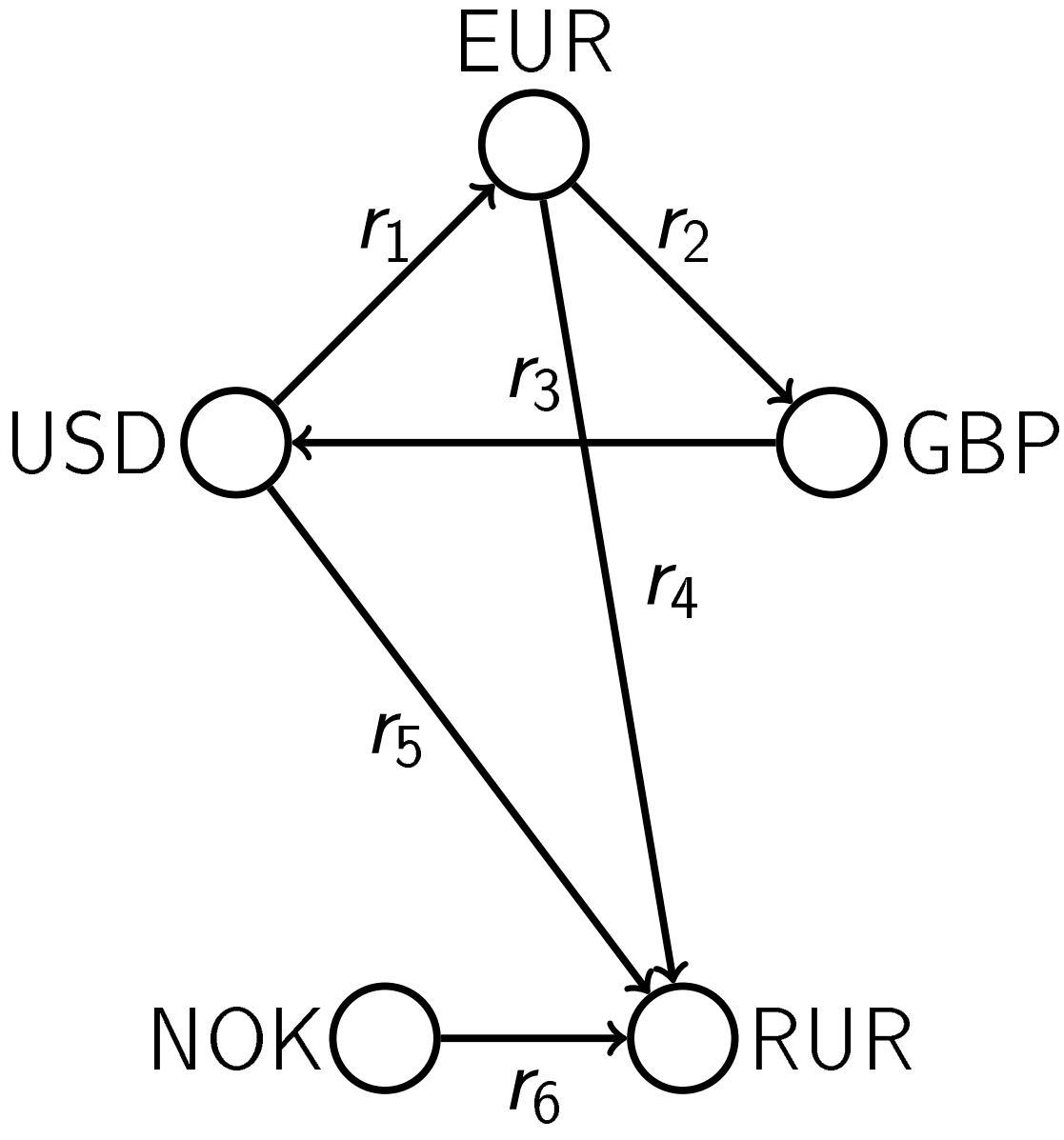
Arbitrage





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


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.

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 instead of maximization

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

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

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

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

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

Taking the Logarithm

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

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

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

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

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

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 $r_{e_i} \rightarrow (-\log(r_{e_i}))$

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?

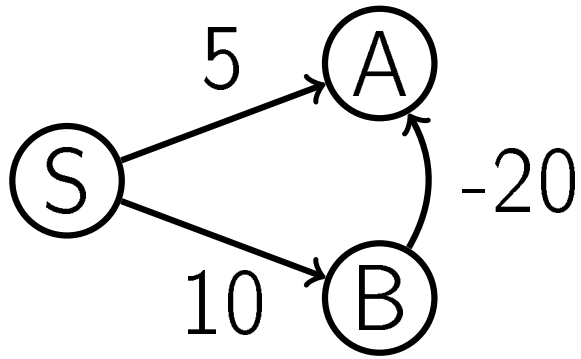
- Create currency exchange graph with weights r_{e_i} corresponding to exchange rates
- Replace $r_{e_i} \rightarrow (-\log(r_{e_i}))$
- Find the shortest path from USD to RUR by Dijkstra's algorithm
- Do the exchanges corresponding to the shortest path

Where Dijkstra's algorithm goes wrong?

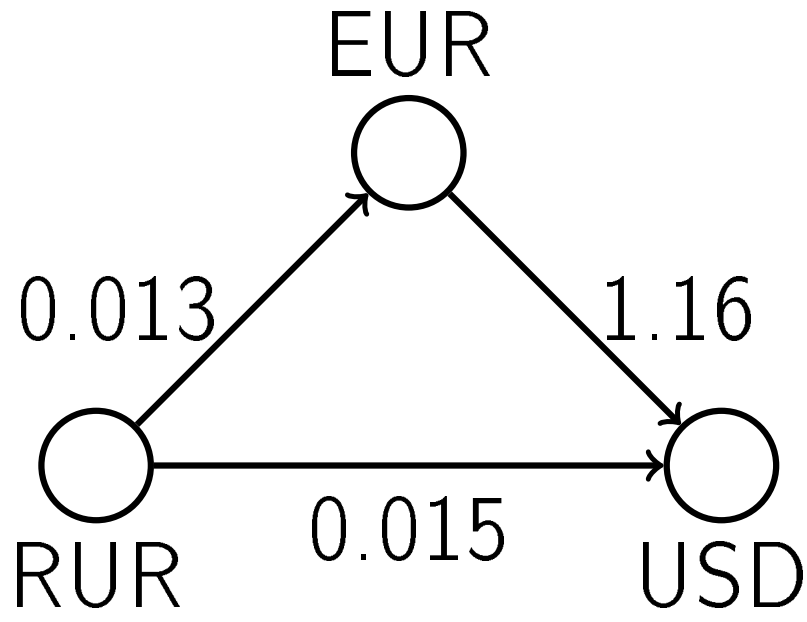
- Dijkstra's algorithm relies on the fact that a shortest path from s to t goes only through vertices that are closer to s.

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.
- This is no longer the case for graphs with negative edges:

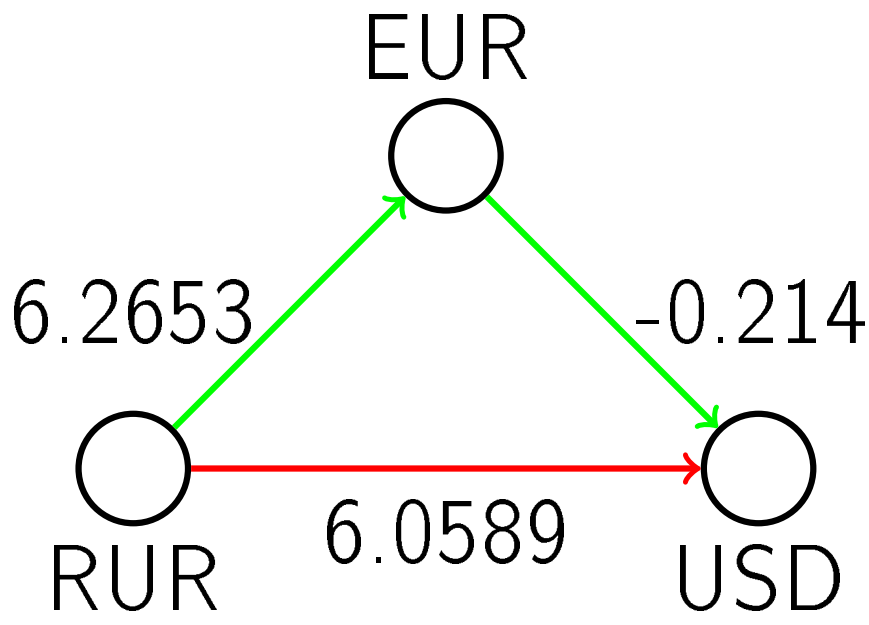


Currency exchange example



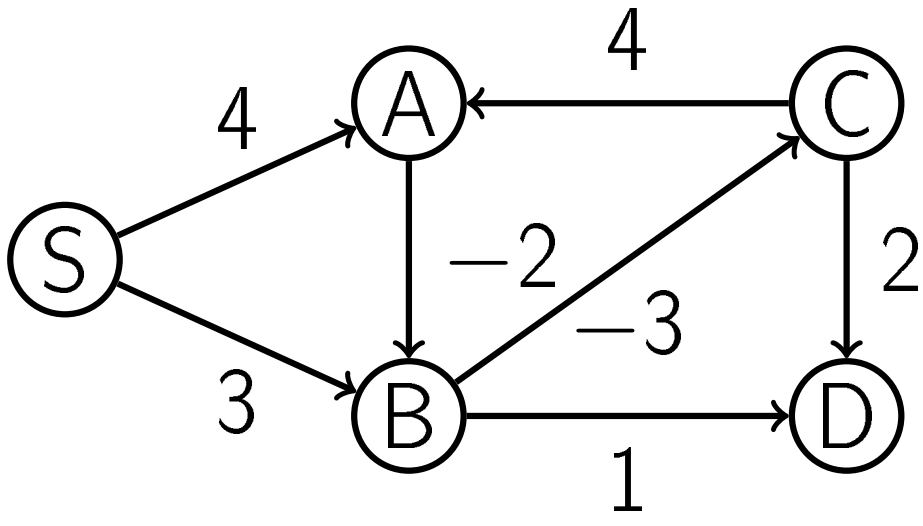
$$0.013 \times 1.16 = 0.01508 > 0.015$$

Currency exchange example

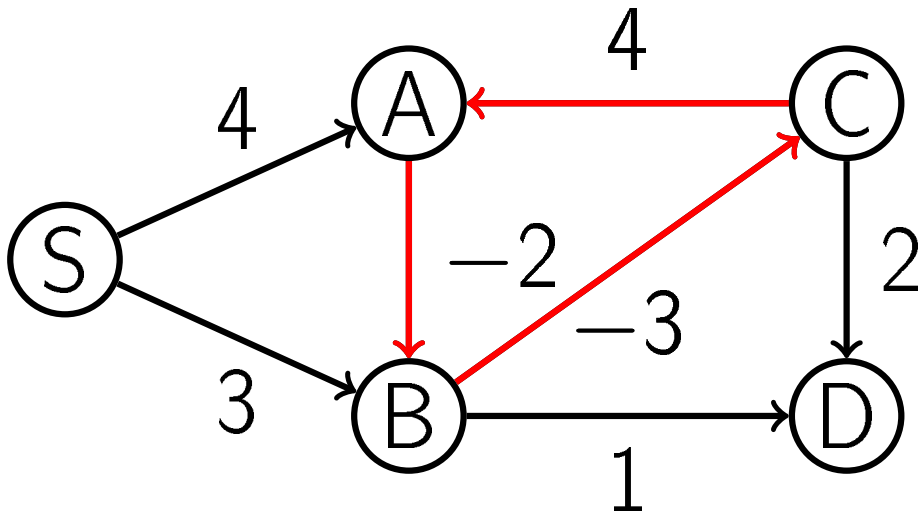


$$0.013 \times 1.16 = 0.01508 > 0.015$$

Negative weight cycles

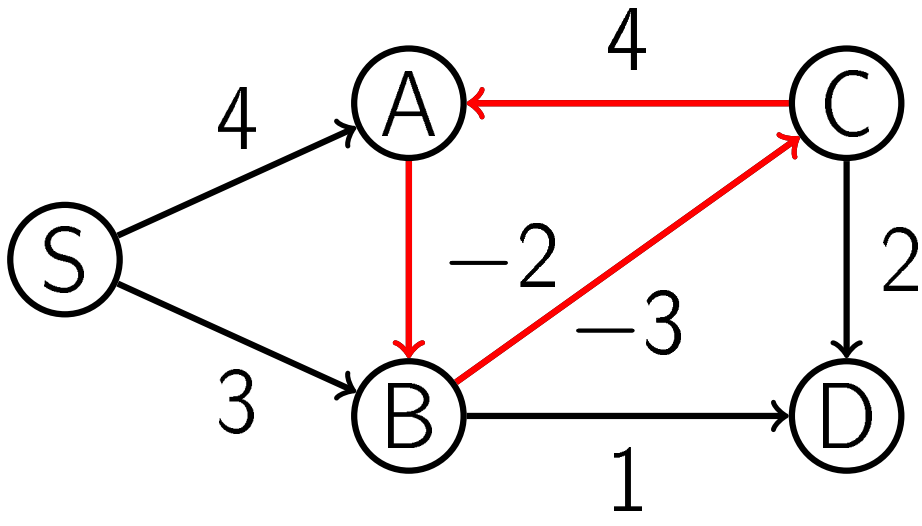


Negative weight cycles



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

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!

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

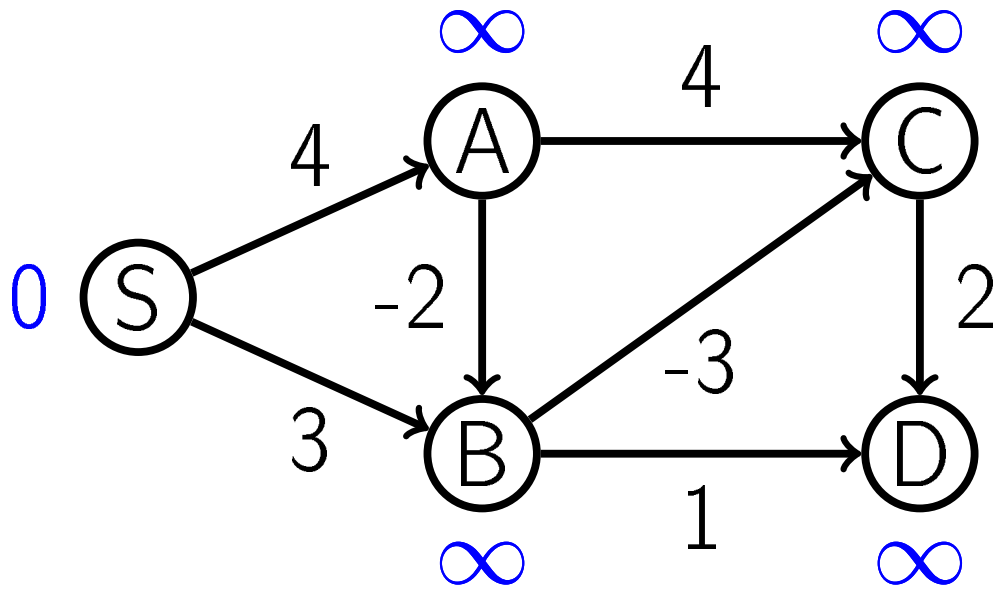
Lemma

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

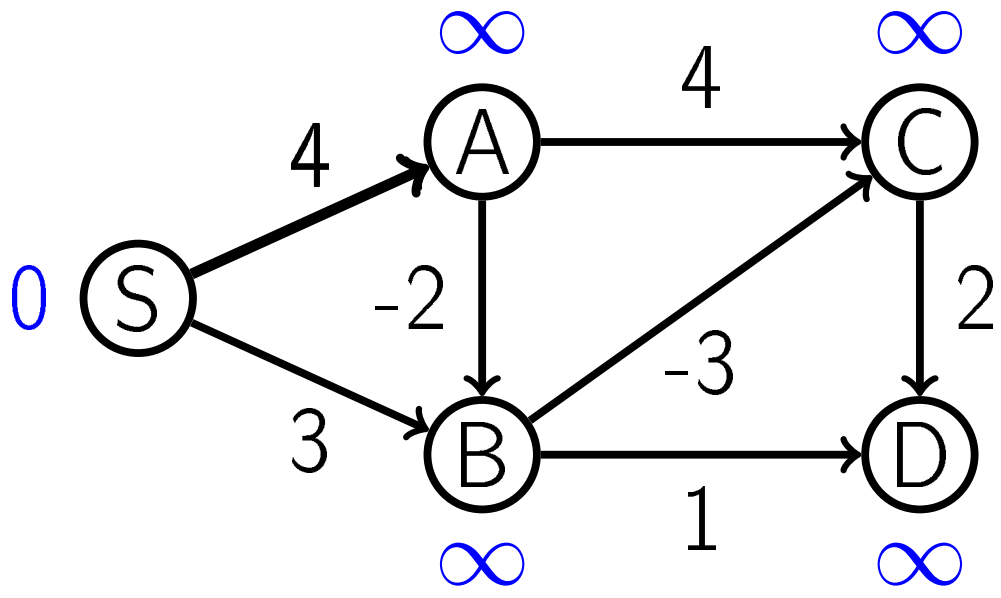
Proof

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

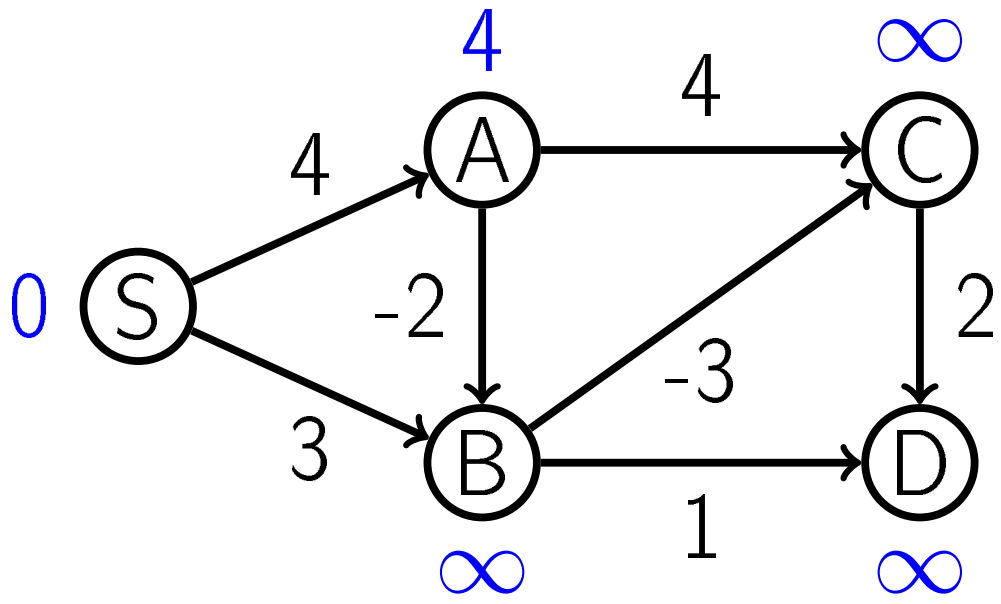
Example



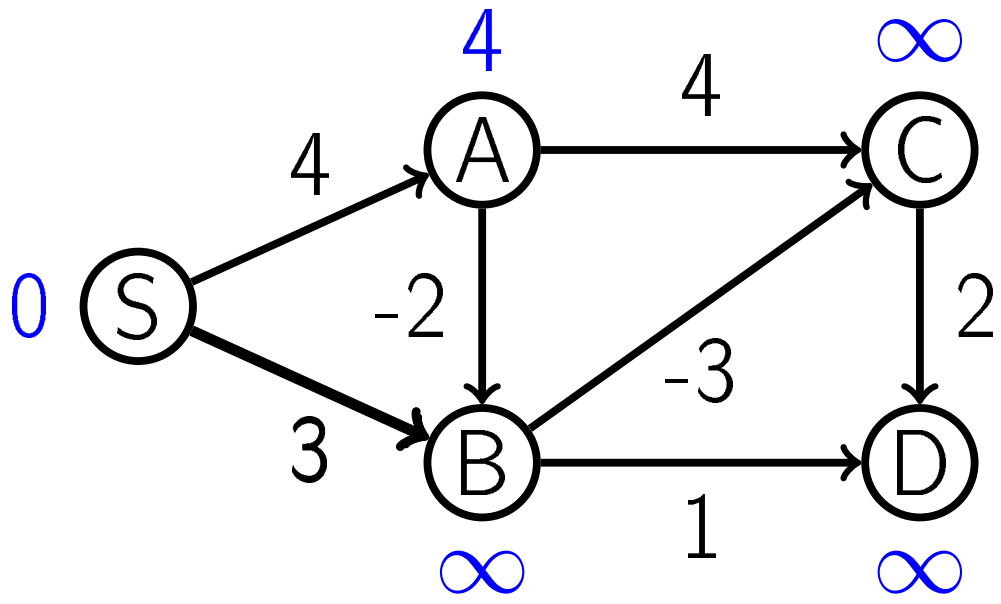
Example



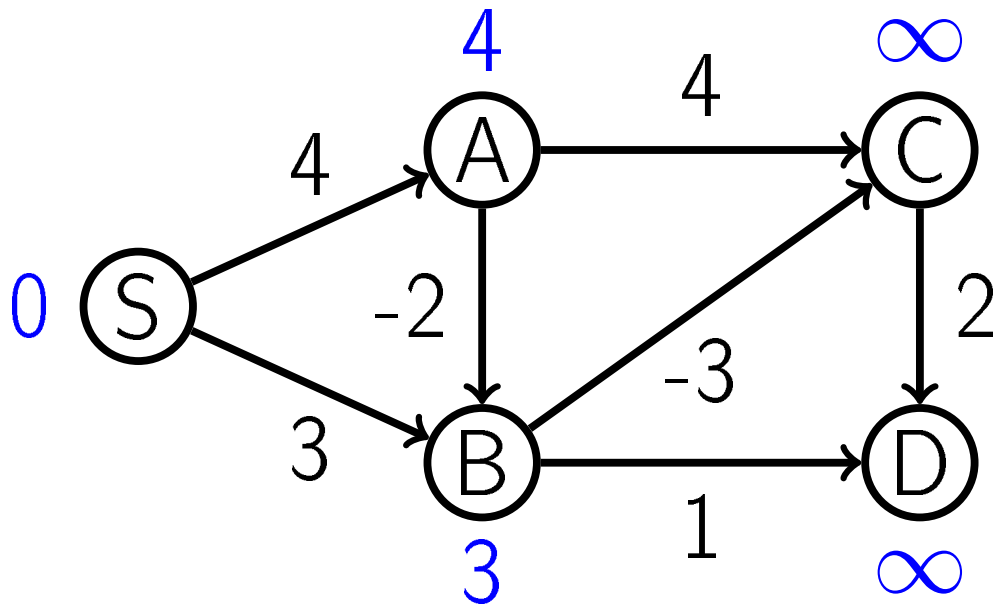
Example



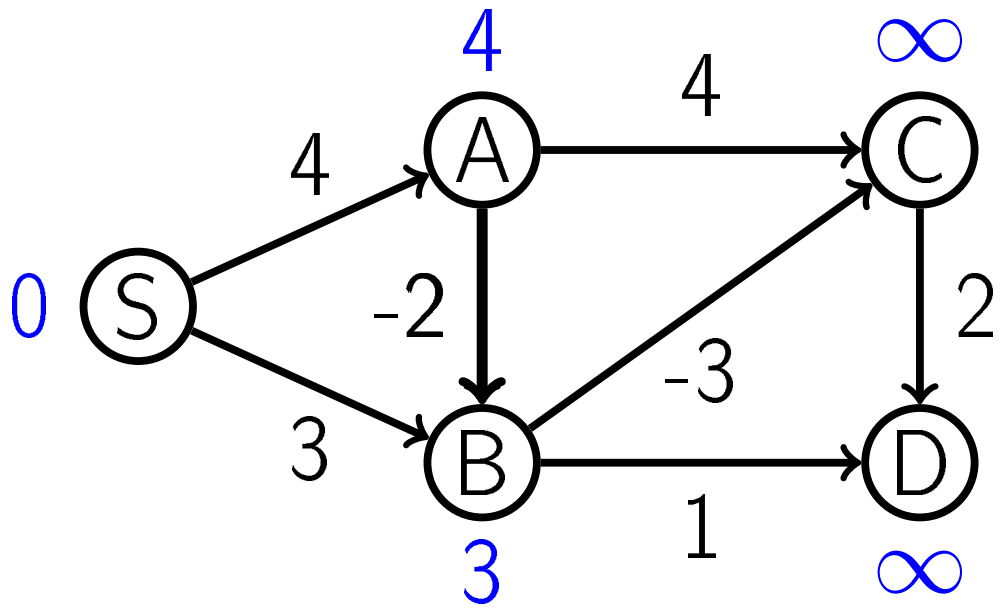
Example



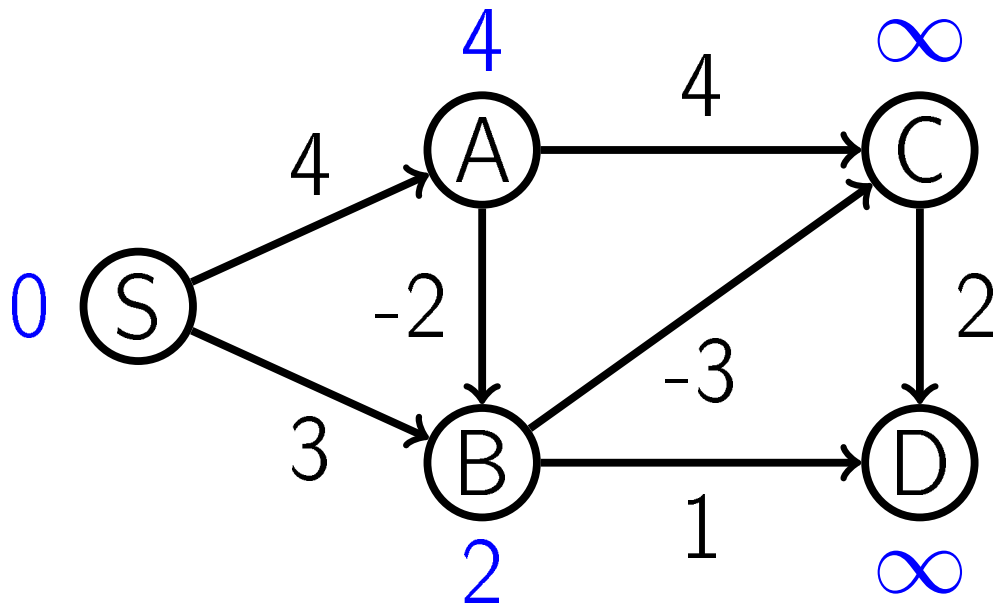
Example



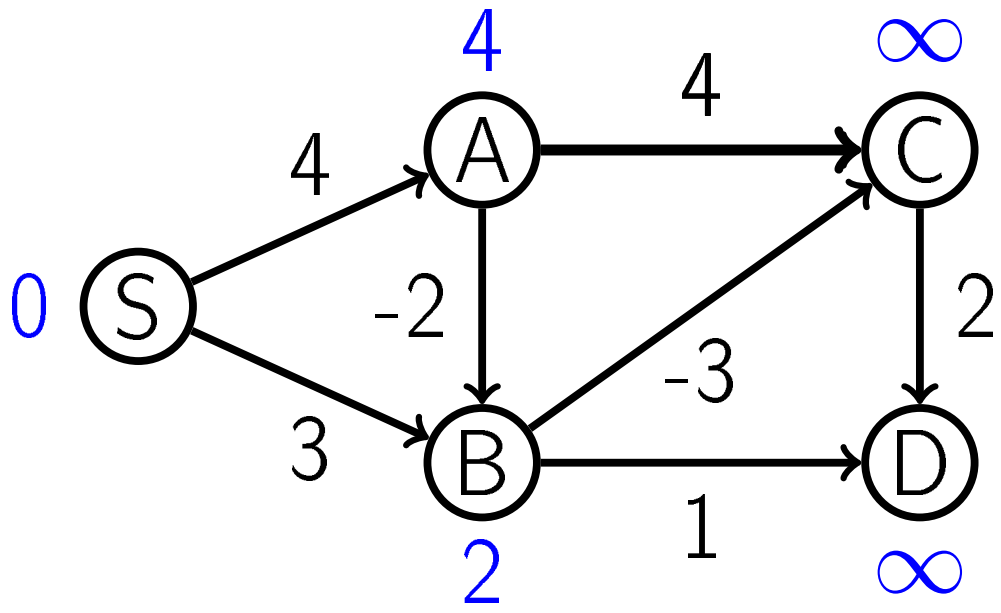
Example



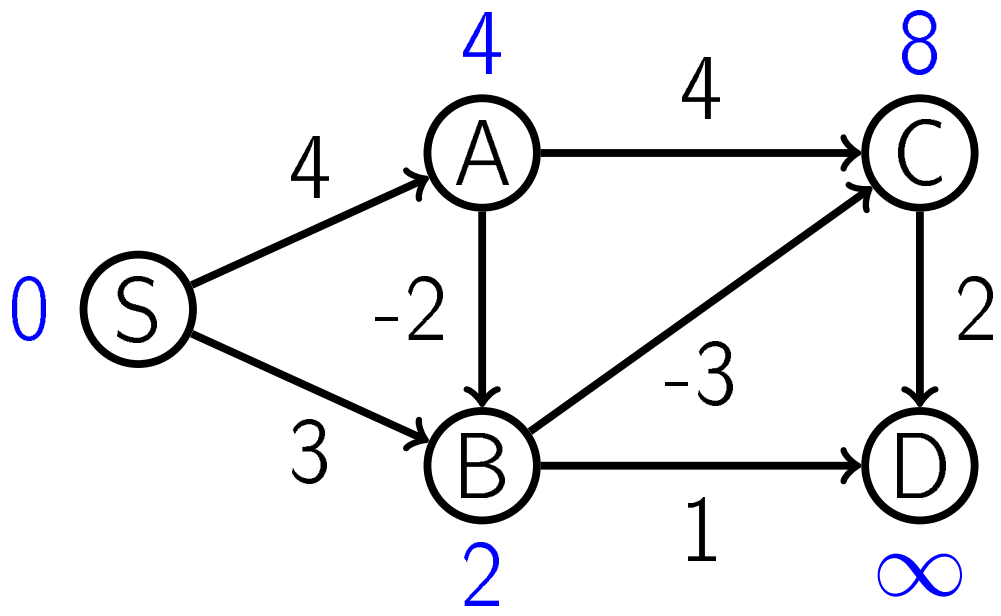
Example



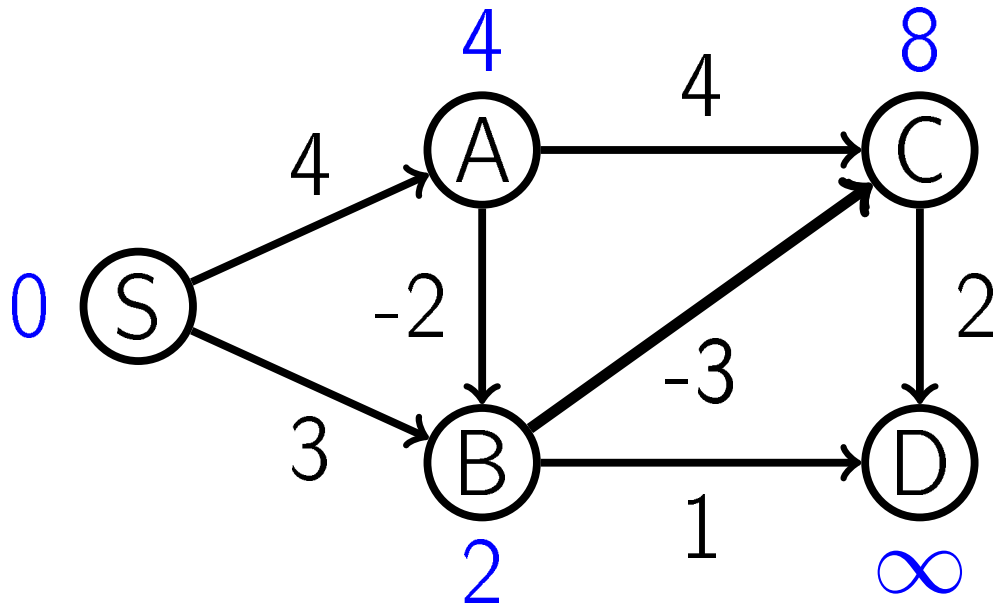
Example



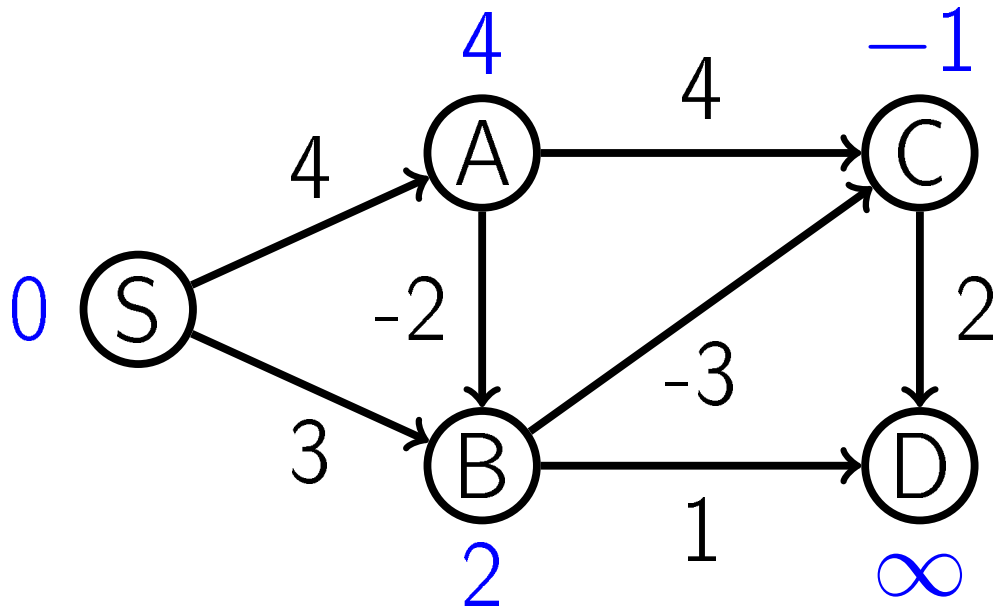
Example



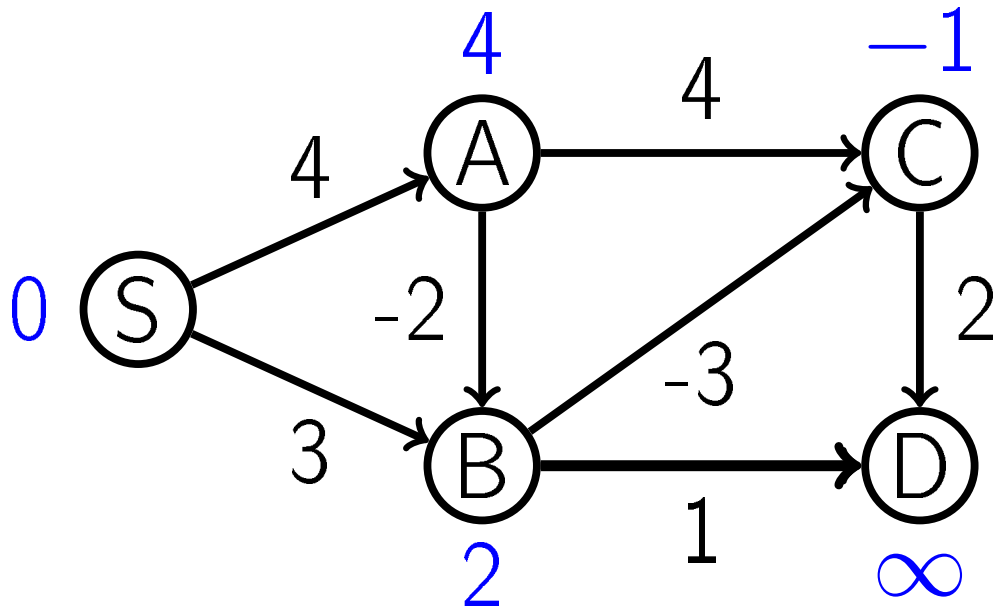
Example



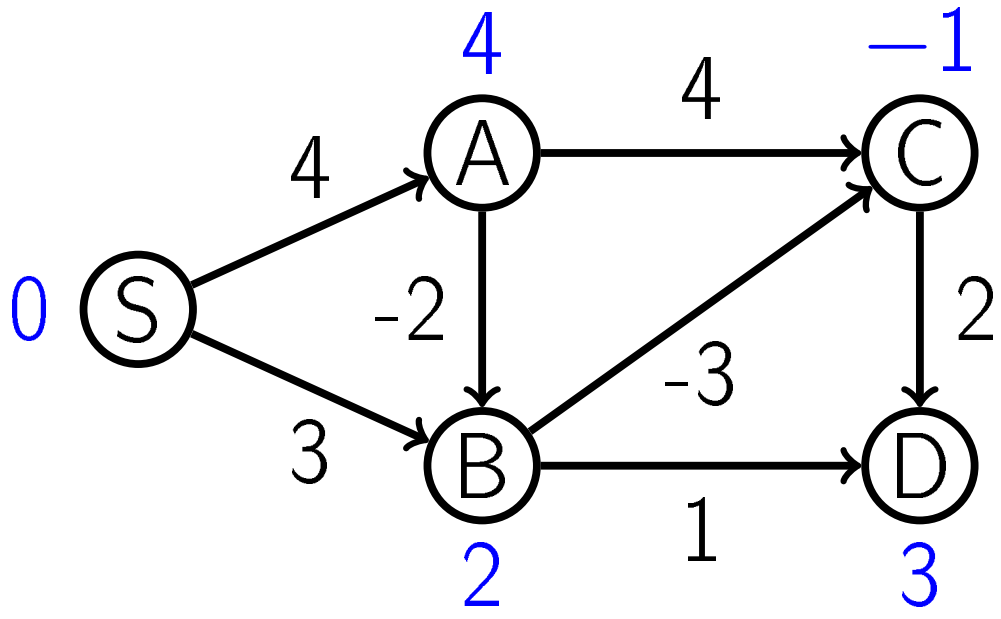
Example



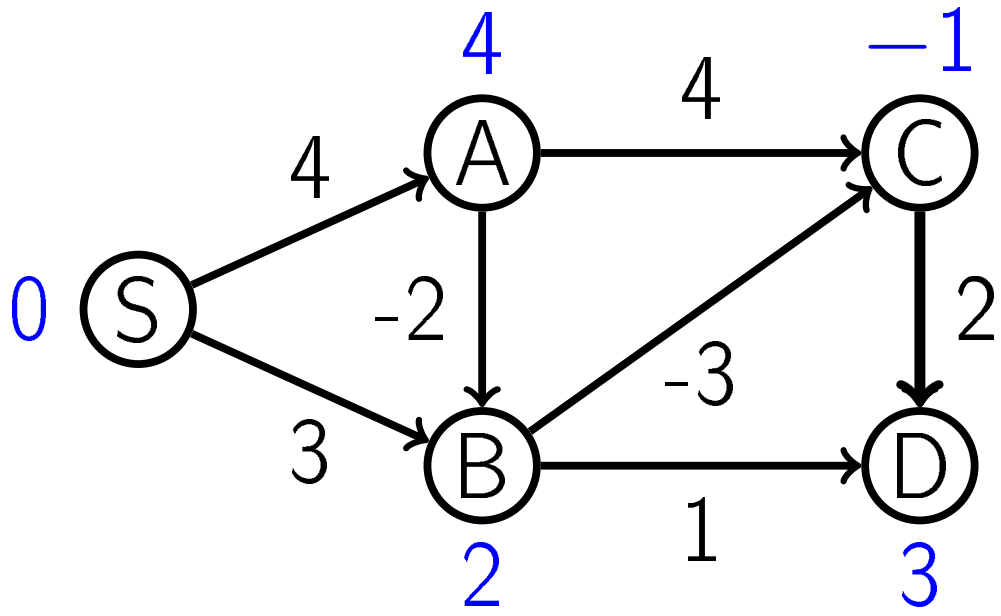
Example



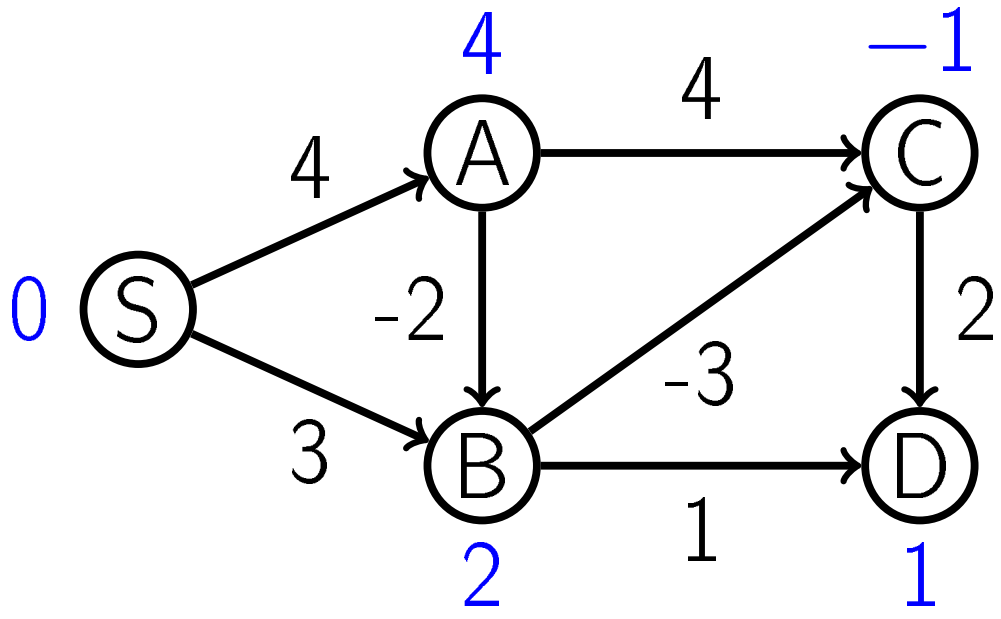
Example



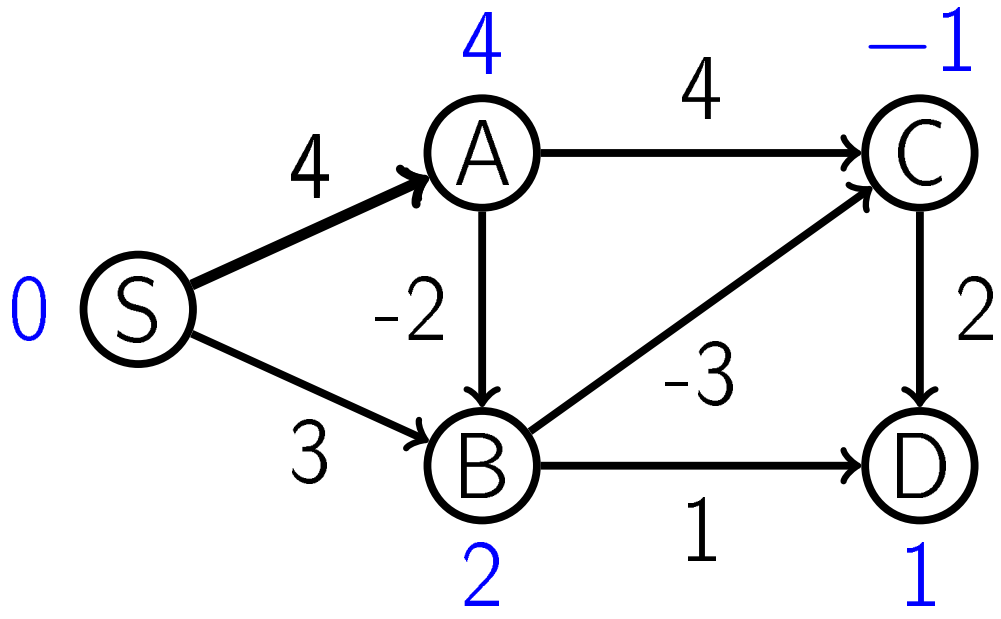
Example



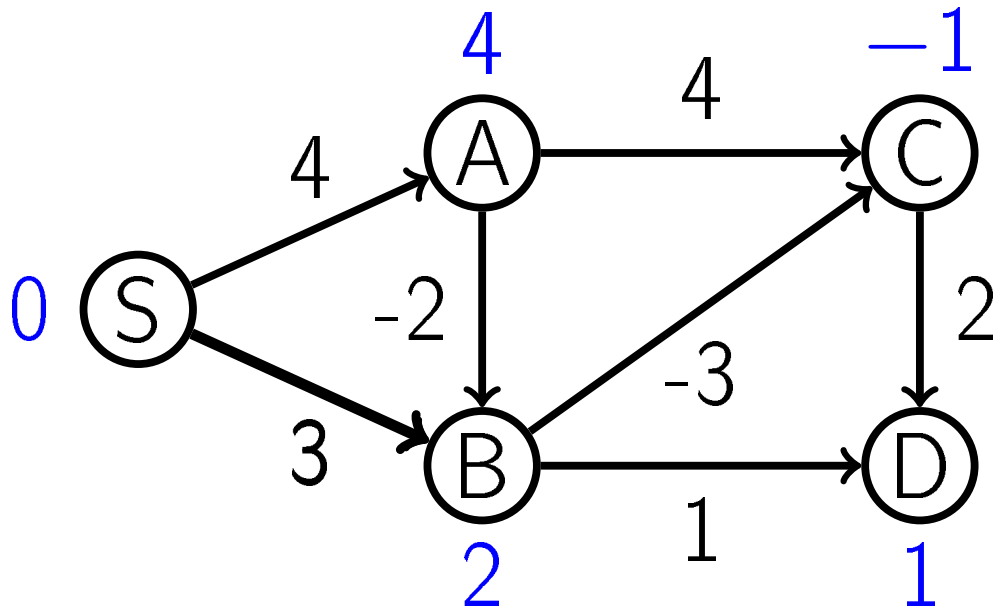
Example



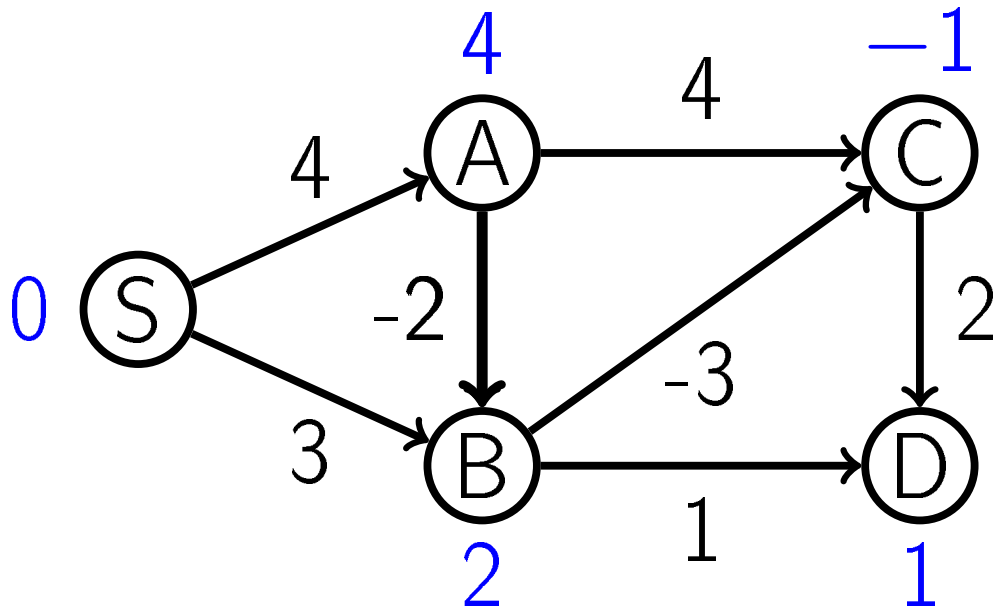
Example



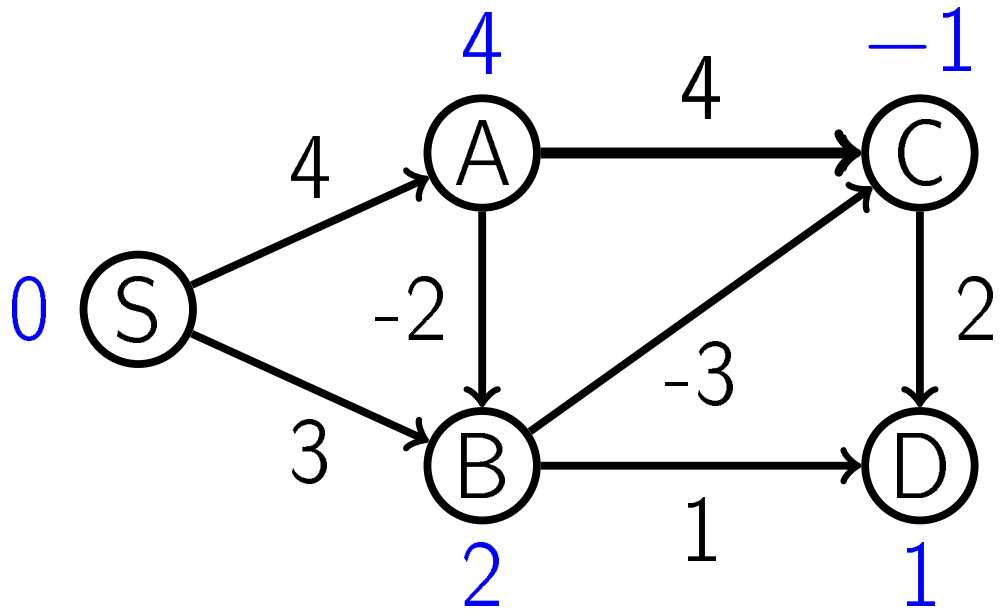
Example



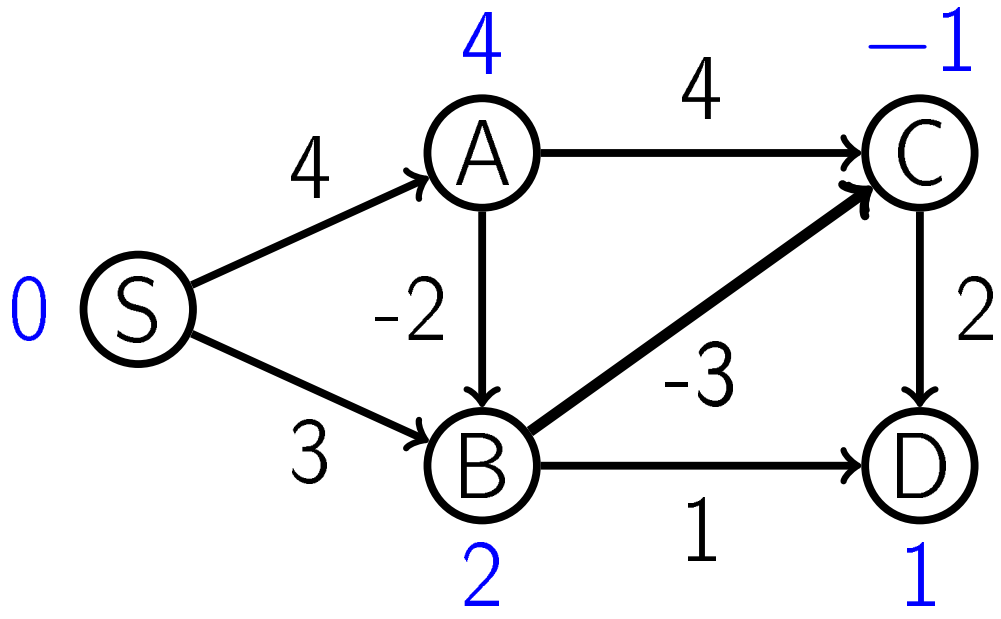
Example



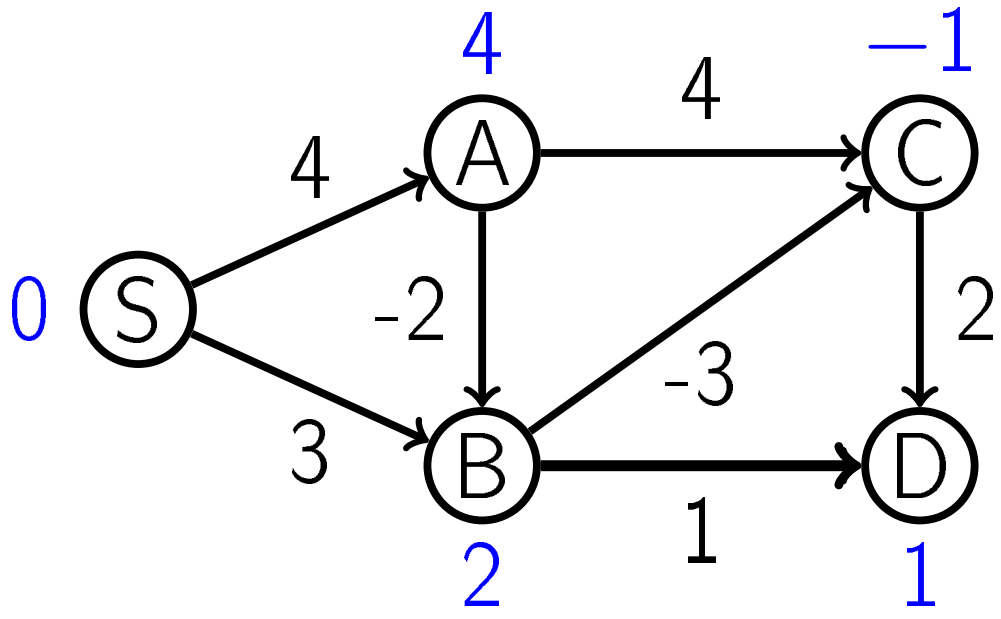
Example



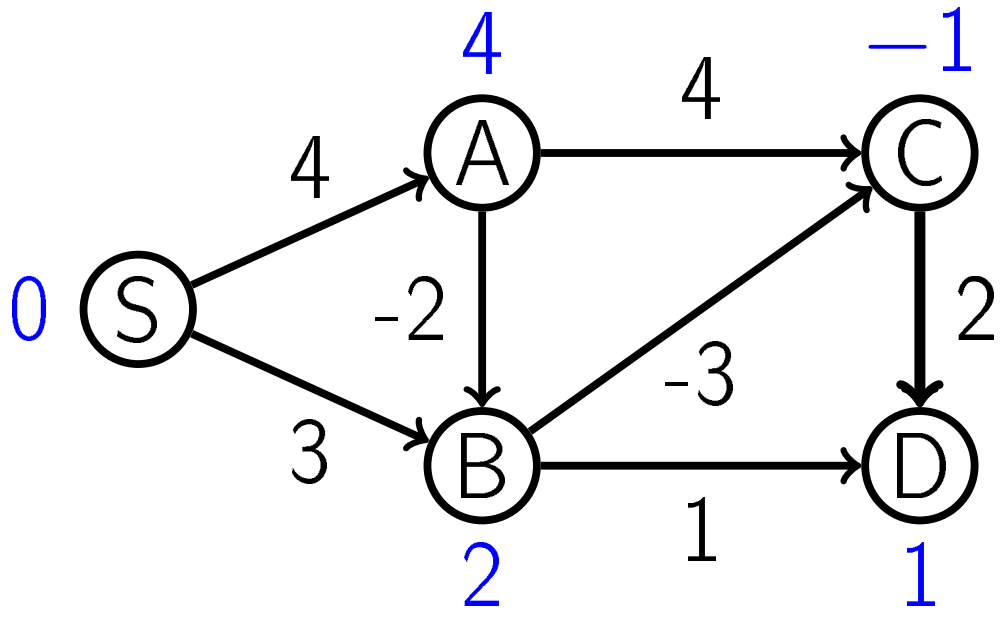
Example



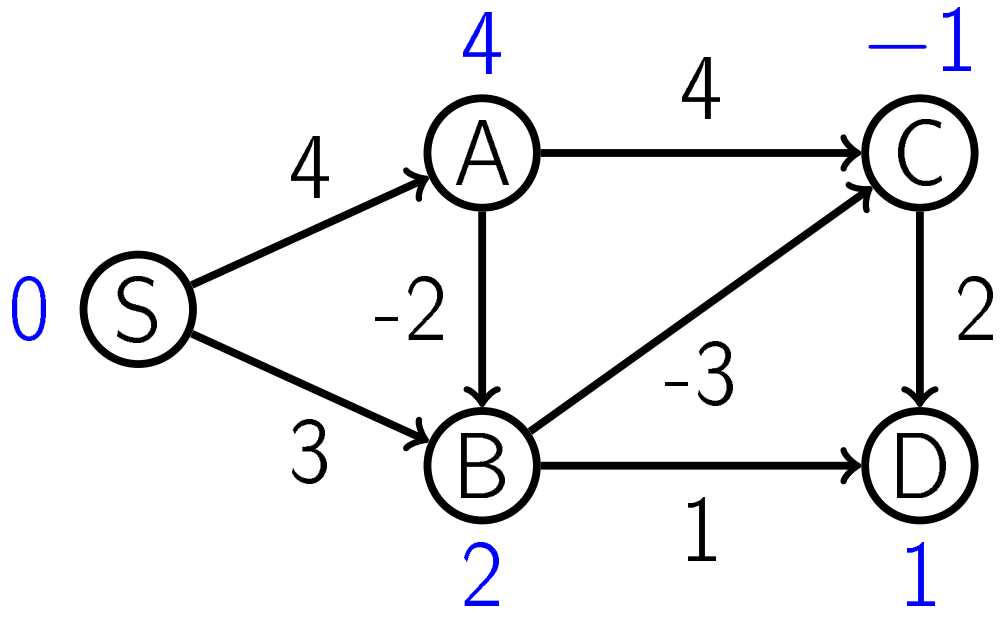
Example



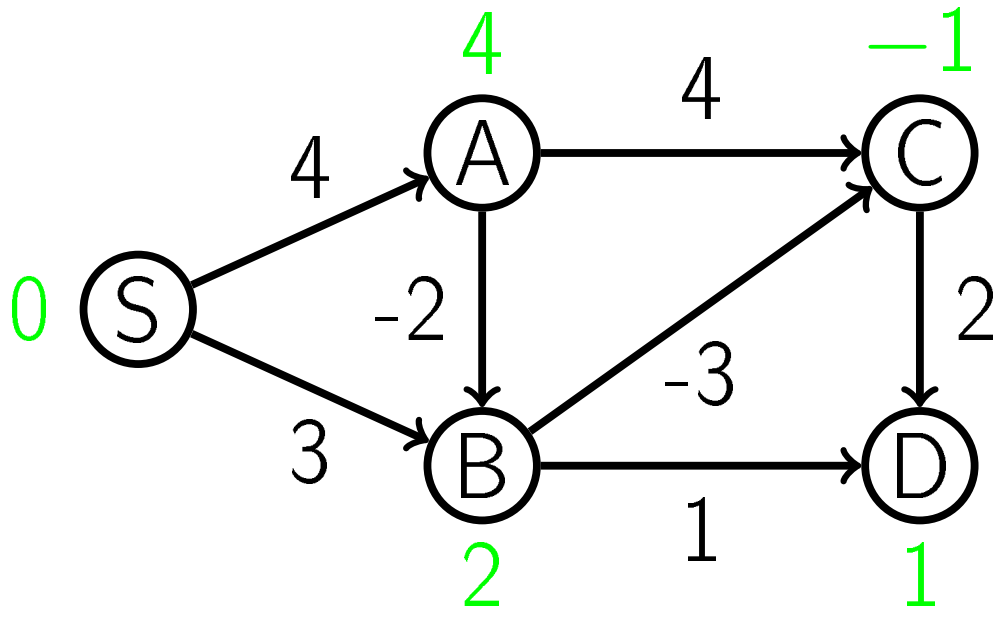
Example



Example



Example



Outline

Lemma

After k iterations of relaxations, for any node u, 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 dist$[S] = 0$, which is correct.
- Induction: proved for $k \rightarrow$ prove for $k + 1$

Proof

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

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)

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

Corollary

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

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 (any path with $\geq |V|$ edges contains a cycle, it is non-negative, so it can be removed from the shortest path), so no `dist`-value can be updated 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.

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

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

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

$w(a, b) + w(b, c) + w(c, a) \geq 0$ — a contradiction.

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

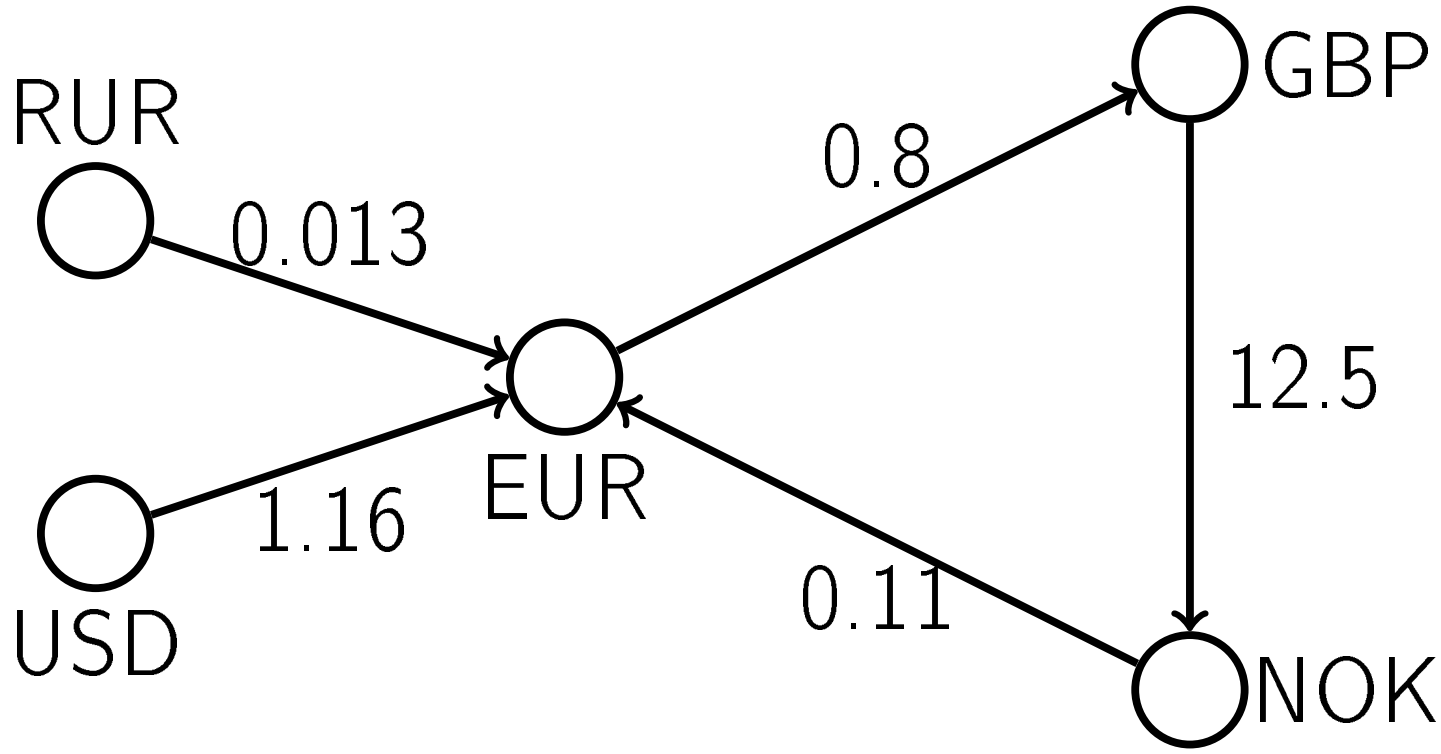
Algorithm:

- 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.

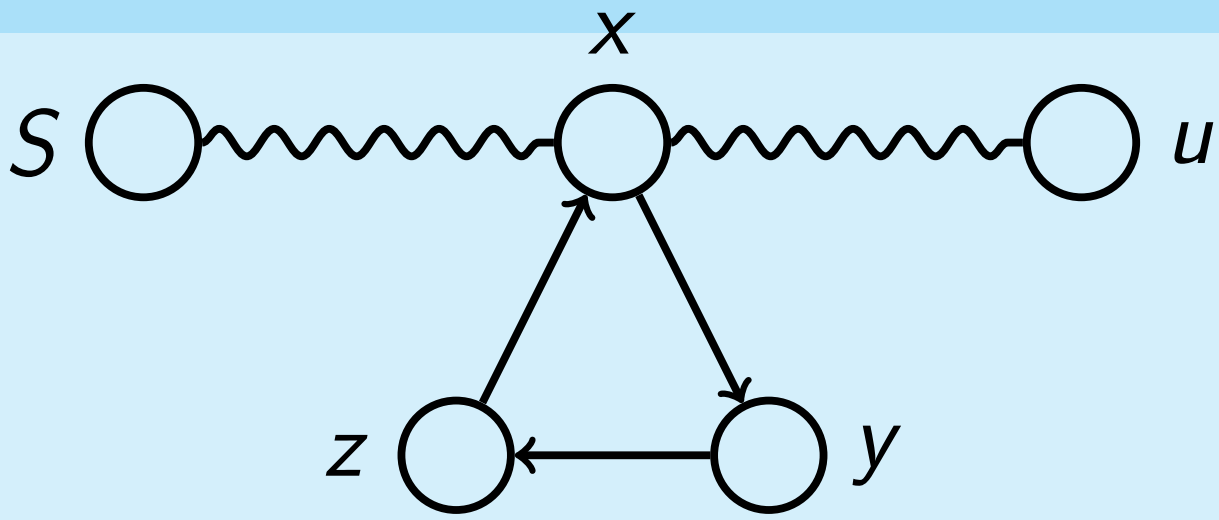
Detect Infinite Arbitrage

Lemma

If it is possible to get as much as possible of currency u from currency S, then dist[u] will be relaxed between $|V|$-th and $2|V|$-th iteration of Bellman–Ford.

Detect Infinite Arbitrage

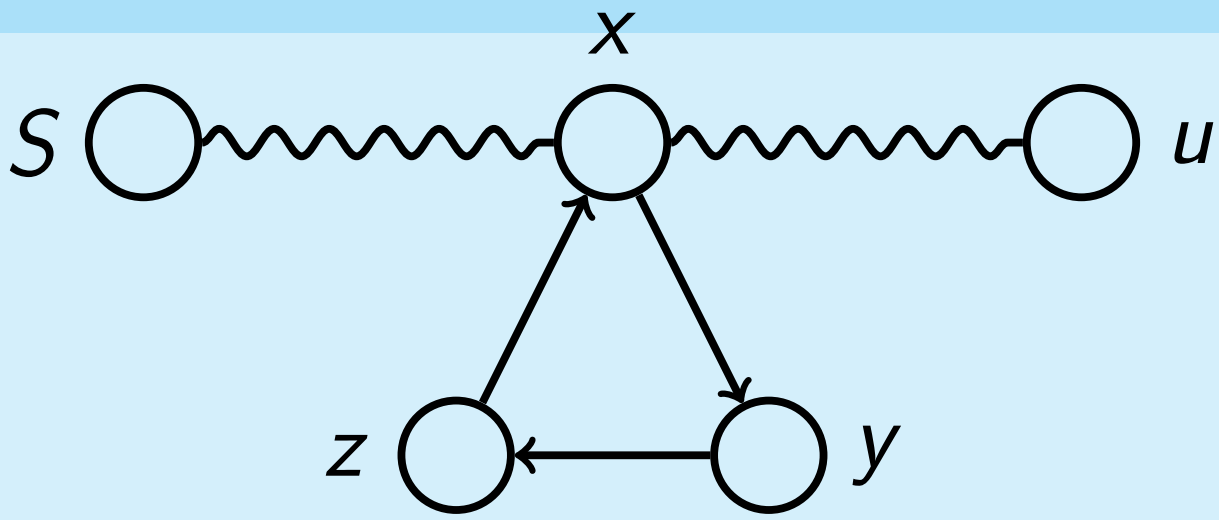
Proof



- $L(S - u) \leq |V| - 1$

Detect Infinite Arbitrage

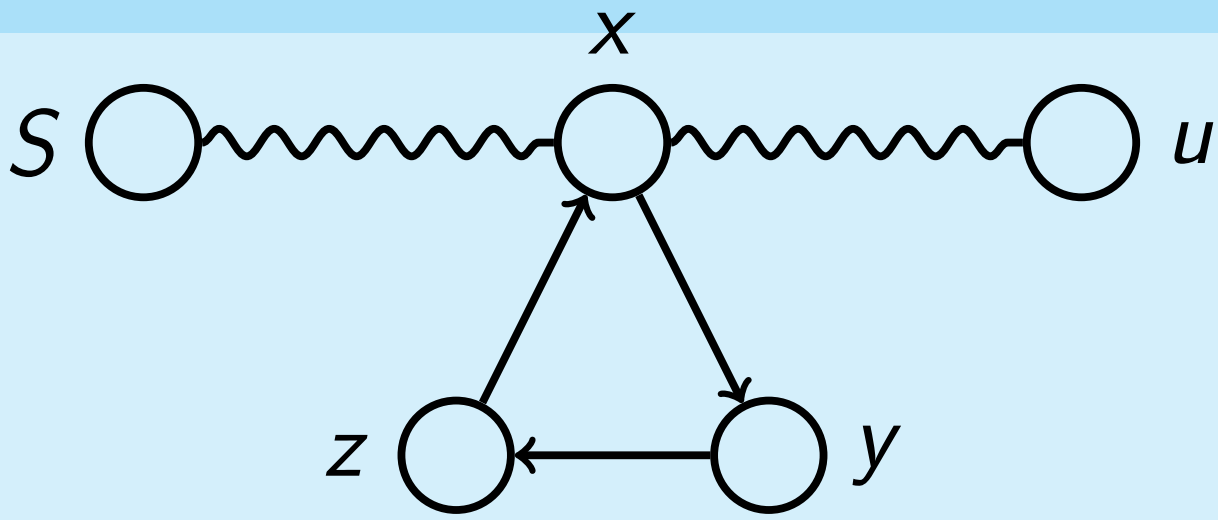
Proof



- $L(S - u) \leq |V| - 1$
- $L(x - y - z - x) \leq |V|$

Detect Infinite Arbitrage

Proof



- $L(S-u) \leq |V|-1$
- $L(x-y-z-x) \leq |V|$
- $|V| \leq L(S-x-\ldots-x-u) \leq 2|V|$ □

Detect Infinite Arbitrage

- Do $2|V|$ iterations of Bellman–Ford, save prev separately after $|V| - 1$-st iteration to be able to find paths from S

Detect Infinite Arbitrage

- Do $2|V|$ iterations of Bellman–Ford, save prev separately after $|V| - 1$-st iteration to be able to find paths from S
- If dist$[u]$ changed during iterations $|V| - 2|V|$, infinite arbitrage from S to u is possible, otherwise it is not

Detect Infinite Arbitrage

- Do $2|V|$ iterations of Bellman–Ford, save prev separately after $|V| - 1$-st iteration to be able to find paths from S
- If dist$[u]$ changed during iterations $|V| - 2|V|$, infinite arbitrage from S to u is possible, otherwise it is not
- If infinite arbitrage is possible, find it using the algorithm for finding a negative cycle and the saved prev from iteration $|V| - 1$.

Conclusion

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