SE284: Introduction to Graph Algorithms

Katerina Taškova

Email: katerina.taskova@auckland.ac.nz
Room: 303S.483

Outline

The Graph Abstract Data Type

Graph Traversals and Applications

Weighted Digraphs and Optimization Problems

- Single-source shortest path problem
  - Dijkstra's algorithm
  - Bellman-Ford algorithm
- All-pairs shortest path problem
  - Floyd's algorithm
- Minimum spanning tree problem
  - Prim's algorithm
  - Kruskal's algorithm
- Hard problems

Dijkstra's algorithm and Priority-First Search, Bellman-Ford algorithm

Lecture Notes 31, Textbook 6.3

Acknowledgment for slide content: Michael Dinneen, Simone Linz

Dijkstra's algorithm, Priority-First Search version

```
1:  function DIJKSTRA2(weighted digraph (G, c); node s ∈ V(G))
2:      priority queue Q
3:      array colour[0..n − 1], dist[0..n − 1]
4:      for u ∈ V(G) do
5:          colour[u] ← WHITE
6:      colour[s] ← GREY
7:      Q.insert(s, 0)
8:      while not Q.isEmpty() do
9:          u ← Q.peek()   t1 ← Q.getKey(u)
10:         for each x adjacent to u do
11:             t2 ← t1 + c(u, x)
12:             if colour[x] = WHITE then
13:                 colour[x] ← GREY
14:                 Q.insert(x, t2)
15:             else if colour[x] = GREY and Q.getKey(x) > t2 then
16:                 Q.decreaseKey(x, t2)
17:         Q.delete()
18:         colour[u] ← BLACK
19:         dist[u] ← t1
20:     return dist
```

- n delete-min and (at most) m decrease-key operations.
- Dijkstra2 using a binary heap runs in time $O((n + m) \log n)$.

Bellman-Ford algorithm

```
1:  function BellmanFord(weighted digraph (G, c); node s ∈ V(G))
2:      array dist[0..n − 1]
3:      for u ∈ V(G) do
4:          dist[u] ← ∞
5:      dist[s] ← 0
6:      for i from 0 to n − 1 do
7:          for x ∈ V(G) do
8:              for v ∈ V(G) do
9:                  dist[v] ← min(dist[v], dist[x] + c(x, v))
10:     return dist
```

Bellman-Ford algorithm

Fact

Suppose that G contains no negative weight cycles reachable from starting node s. In the i^{th} iteration of the outer **for** *loop, dist[v] contains the minimum weight of a path from s to each node v whose level is at most i (i.e. number of arcs from s to v is at most i).*

Proof.

The update formula is such that dist values never increase.

We use induction on i.

Base case. When $i = 0$ the statement is true because of our initialization.

Induction hypothesis. Suppose the statement is true for i.

Bellman-Ford algorithm

Fact

Suppose that G contains no negative weight cycles reachable from starting node s. In the i^{th} iteration of the outer **for** *loop, dist[v] contains the minimum weight of a path from s to each node v whose level is at most i (i.e. number of arcs from s to v is at most i).*

Proof.

Show that statement is true for $i+1$**.** Let v be a node at level $i+1$, and let γ $(= s, x_1, x_2, \ldots x_k, v)$ be a minimum weight path from s to v.

- Since there are no negative weight cycles, γ has $i+1$ arcs.
- If x_k is the last node of γ before v, and γ_1 the subpath to x_k, then by the inductive hypothesis we have $dist[x_k] \leq |\gamma_1|$.
- Thus by the update formula we have $dist[v] \leq dist[x_k] + c(x_k, v) \leq |\gamma_1| + c(x_k, v) \leq |\gamma|$ as required.

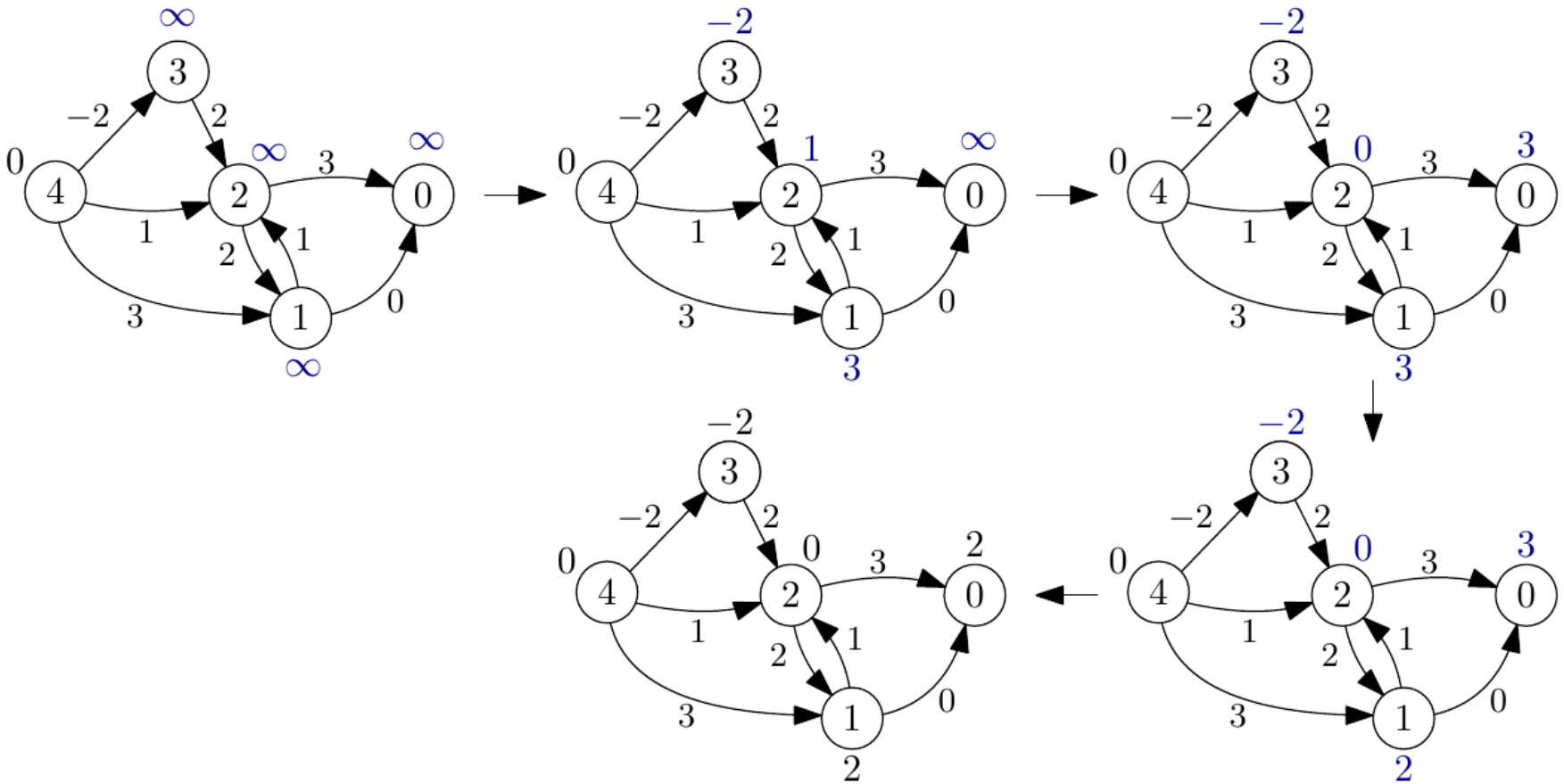
Comments on Bellman-Ford algorithm

Fact

- *This (non-greedy) algorithm handles negative weight arcs but not negative weight cycles.*
- *Runs slower than Dijkstra's algorithm since considers all nodes at "level" 0, 1, . . . , $n - 1$, in turn.*
- *Runs in time $\Theta(nm)$ with adjacency list, since the two inner-most* ***for*** *loops can be replaced with:* ***for*** *$(x, v) \in E(V)$.*
- *Can be modified to detect negative weight cycle.*

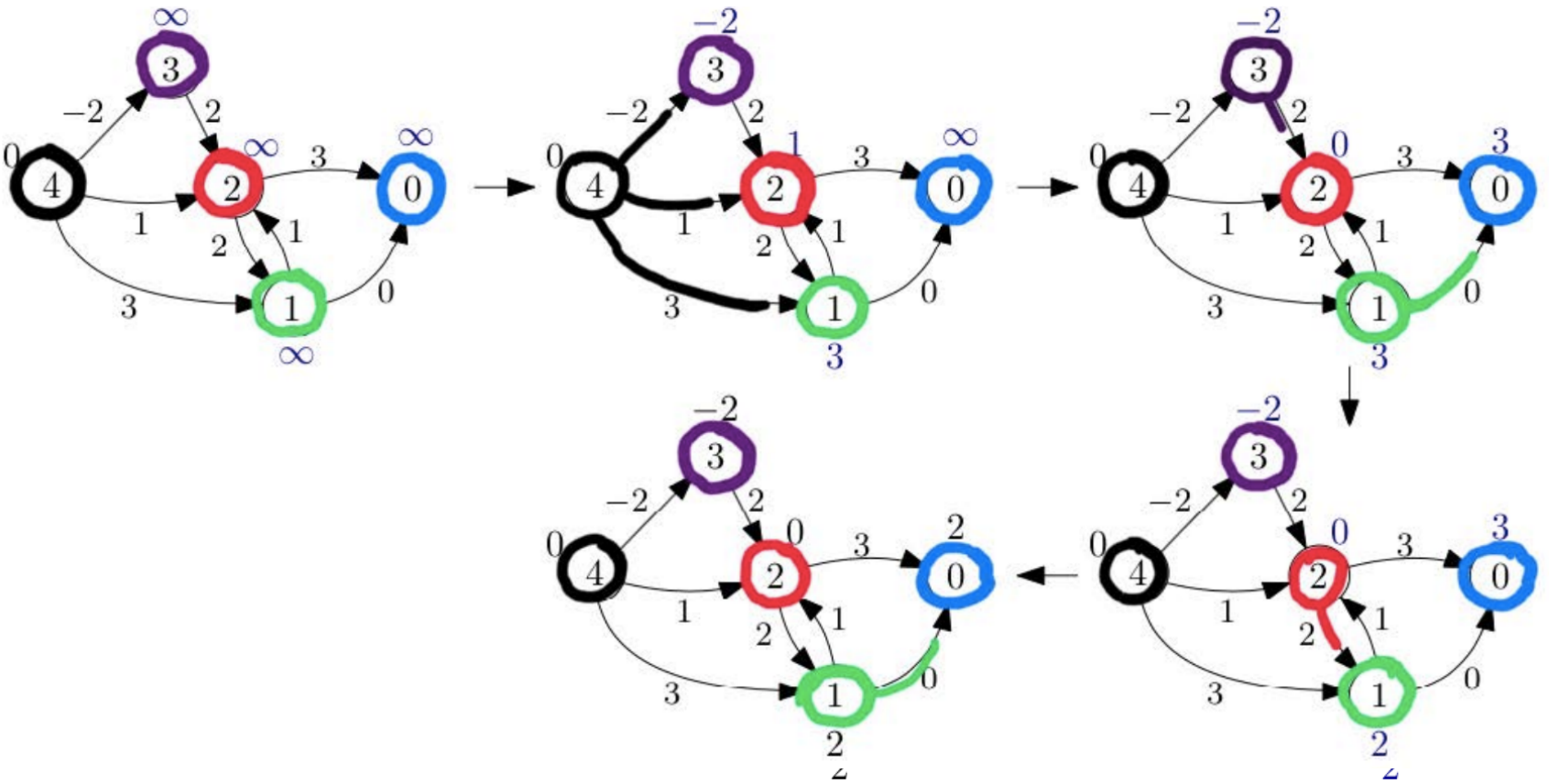
Bellman-Ford algorithm – example

Example 31.1. An application of Bellman–Ford algorithm with starting node 4 when the nodes are processed in the order from 0 to 4.



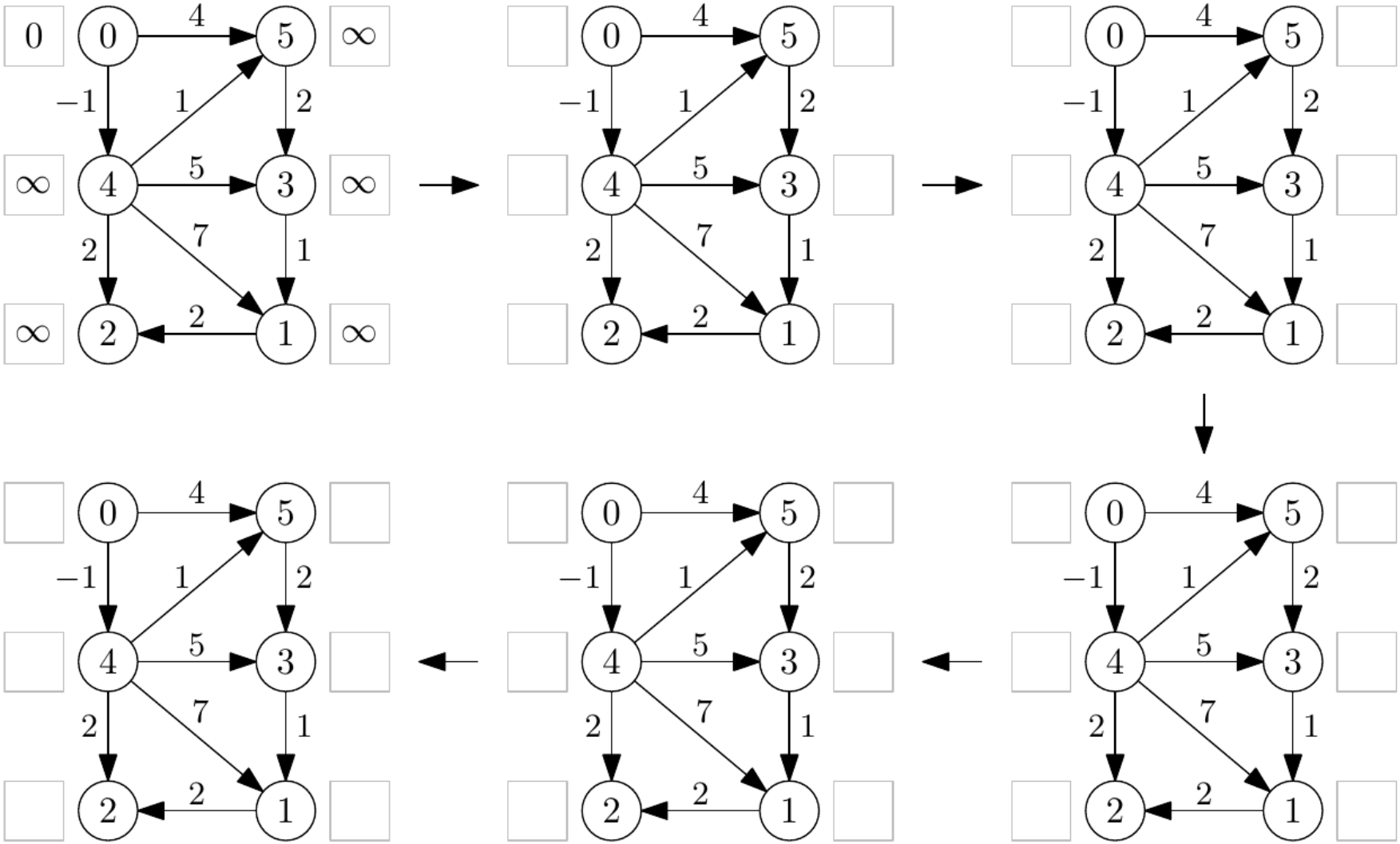
Bellman-Ford algorithm – example

Example 31.1. An application of Bellman–Ford algorithm with starting node 4 when the nodes are processed in the order from 0 to 4.



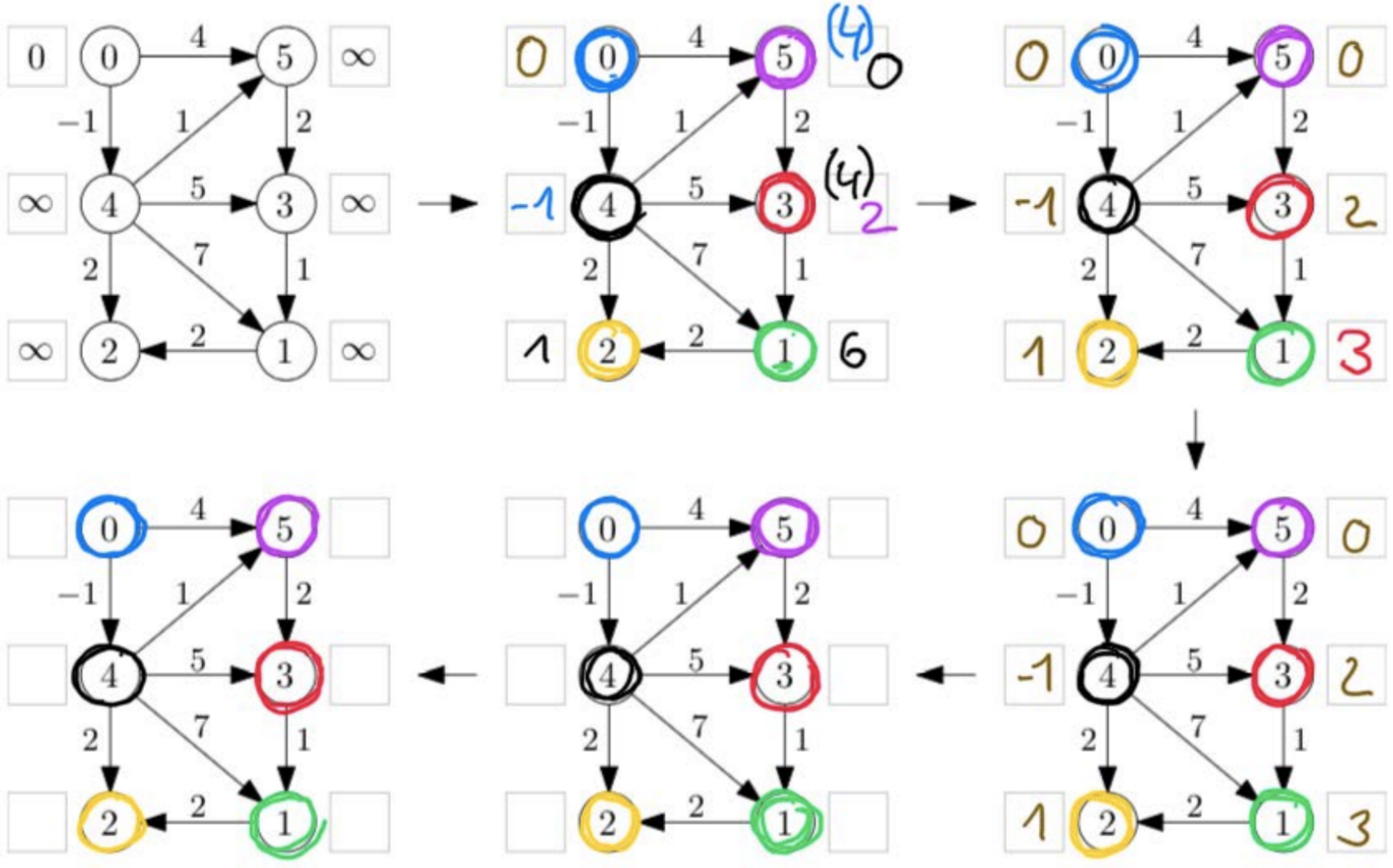
Bellman-Ford algorithm – example

Example 31.2. Execute the Bellman–Ford algorithm on the graph below with starting vertex 0. Process nodes in the order from 0 to 5.

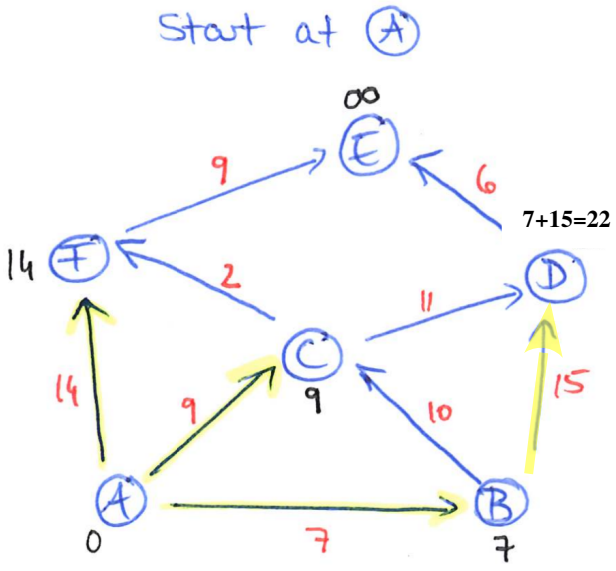


Bellman-Ford algorithm – example

Example 31.2. Execute the Bellman–Ford algorithm on the graph below with starting vertex 0. Process nodes in the order from 0 to 5.



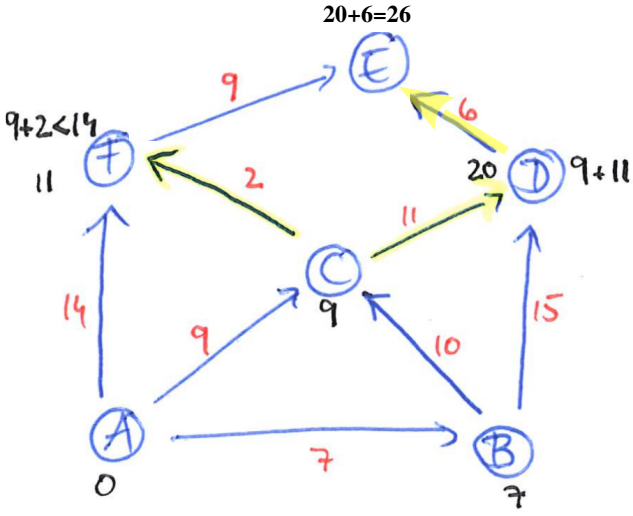
Bellman-Ford algorithm – example



i = 1, changes in dist[v]:

x = A & v = B,C,D

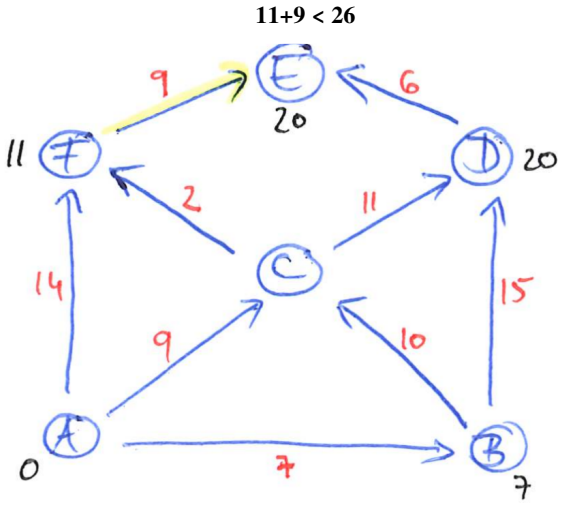
x = B & v = D



i = 1, changes in dist[v]:

x = C & v = D, F

x = D & v = E



i = 1, changes in dist[v]:

x = F & v= E

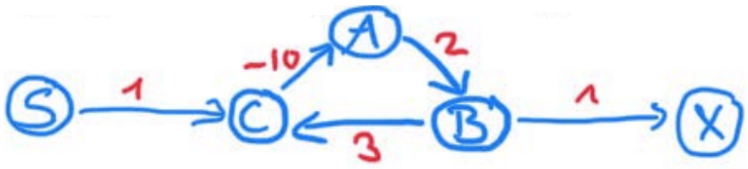
dist[v] stays the same in i=2,3,...n-1

Bellman-Ford algorithm – example

Example 30.4. Explain why the SSSP problem makes no sense if we allow digraphs with cycles of negative total weight.

Bellman-Ford algorithm – example

Example 30.4. Explain why the SSSP problem makes no sense if we allow digraphs with cycles of negative total weight.



Path from S to X

S-C-A-B-X $1+(-10)+2+1=-6$

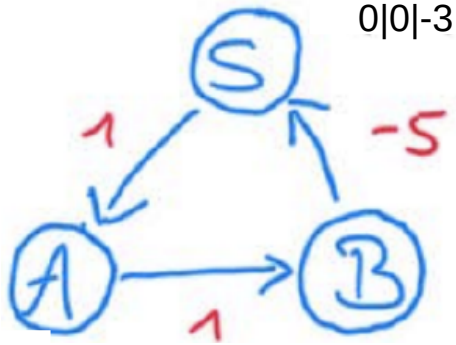
S-C-A-B-C-A-B-X $1+(-10)+2+3+(-10)+2+1=-11$

Bellman-Ford algorithm – example

Example 30.5. Suppose the input to the Bellman–Ford algorithm is a digraph with a negative weight cycle. How could the algorithm detect this, so it can exit gracefully with an error message?

Bellman-Ford algorithm – example

Example 30.5. Suppose the input to the Bellman–Ford algorithm is a digraph with a negative weight cycle. How could the algorithm detect this, so it can exit gracefully with an error message?



0|0|-3|-3|

∞|1|1|-2|

∞|∞|2|2|

dist[v] at i=0| i=1 | i=2 | i=3|

Bellman-Ford algorithm – example

Example 30.5. Suppose the input to the Bellman–Ford algorithm is a digraph with a negative weight cycle. How could the algorithm detect this, so it can exit gracefully with an error message?

Run outer-for loop for one more iteration (i.e. total number of iterations is n*). If dist[v] changes for some vertex v in the last iteration, then the graph has a negative weight cycle

* see pseudocode

Thank you!