Repetition – overview (1/2)

Experimental analysis of algorithms

- Why experimental analysis?
- Methods to analyse an algorithm: ratio-test and **power-test**
- What is the difference between the two methods?

Priority Queues

- Definition of a priority queue
- Heap-representation of a priority queue
  - binary heap, binomial heap, **fibonacci heap**
- What are PQs useful for in algorithms?
- Performance-differences between the three heaps

Repetition – overview (2/2)

Graph-Algorithms

- Basic algorithms
  - Depth-first search
  - Breadth-first search
- Path-finding algorithm
  - Dijkstra
  - Bellman-Ford
- Spanning-Tree algorithms
  - Kruskal
  - Prim

What you should prepare for Thursday

- Apply the depth-first and the breadth-first algorithm to a graph
- Apply Dijkstras algorithm to a weighted graph. Pre-Conditions
- Explain Prims algorithm for Minimum-Cost-Spanning trees and apply it on paper
- For what are priority queues used in Dijkstras and Prims algorithms?

Experimental analysis

What are important steps to prepare, implement and run a performance experiment?

1. Define the problem exactly. What do you want to find out?
2. Decide and define what you want to measure?
3. Generate test data that fit to the problem.
4. Implement and run the tests.
5. Analyse and visualize the results of the tests.

The power test

Explain the main idea of the power test.

What do we know about the runtime-function T(n) of the algorithm?

The polynomial runtime-function T(n) to analyse is **unknown**!

Give the general form of the runtime-function:

$$T(n) = bn^c$$

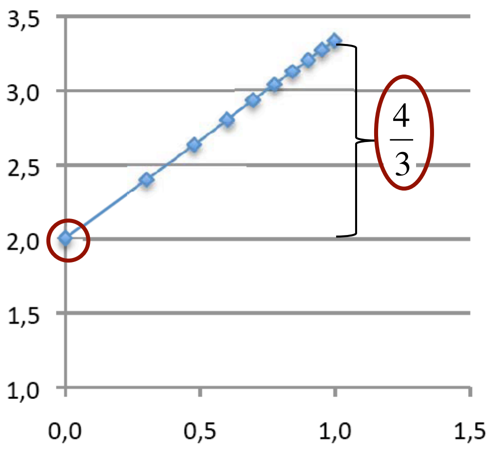
What is n? What applies to n?

The parameter is the **size** of the algorithm's input data: **N ⟶ ∞**

Calculation of T(n)

1. **Gather** the values (n, T(n)), N ⟶ ∞, in a table
2. **Transform** the values to (log(n), log(T(n)). Why?
3. **Analyse** the log-dataset with **linear regression**.
4. What can you find out with linear regression analysis?
   - A linear function $\log_a T(n) = c*\log_a n + b$
5. Exponentiate the linear function with the log-basis a
   - $T(n) = a^b * n^c$
6. Display the results.

Results of the Power-Tests



Find:

$$y = t(n) = bn^c$$

Find the exponent by log-log-transformation

$$y' = cx' + b$$

b = 2 (y-intersection)

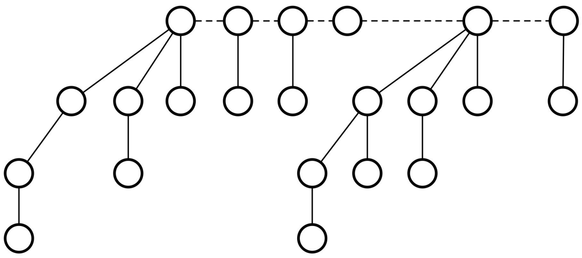
c = 4/3 (slope)

so: y'=(4/3)x'+2

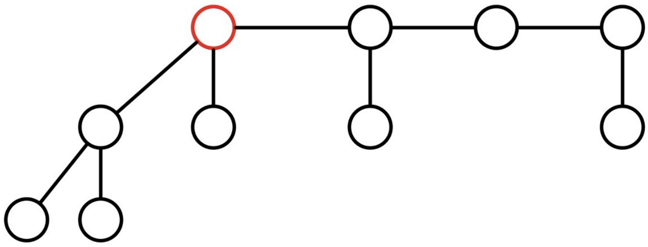
It follows:

$$t(n) = 10^2 \cdot n^{4/3}$$

Fibonacci-Heap

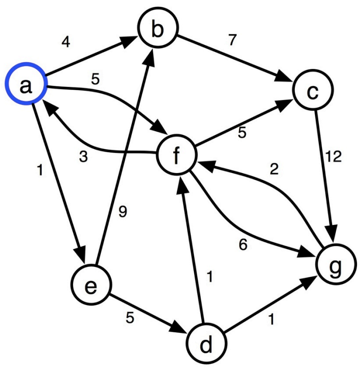


DeleteMin

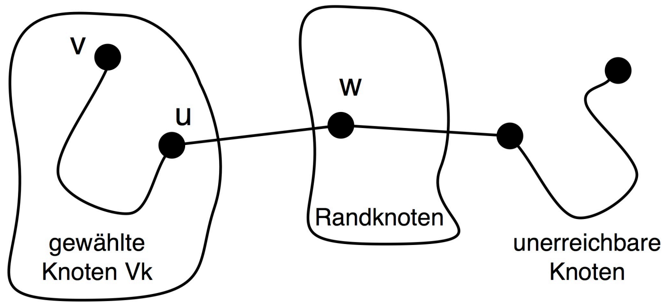


Dijkstra – 1/4

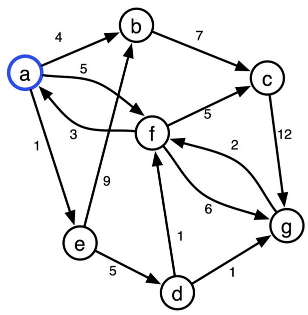
Run Dijkstra's algorithm on the graph below starting at node a. Your solution should be diagramatic, exactly like the example in the lecture.



Dijkstra – 2/4



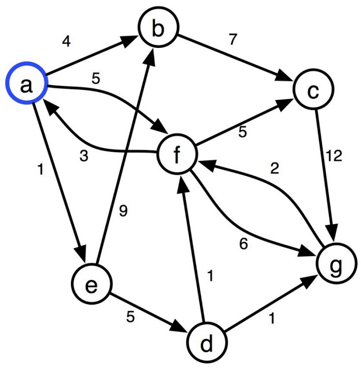
Dijkstra – 3/4

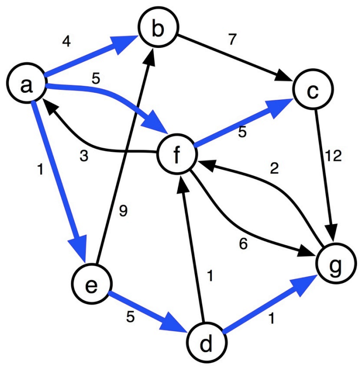


	a	b	c	d	e	f	g
	0	4	∞	∞	1	5	∞

Dijkstra – 4/4

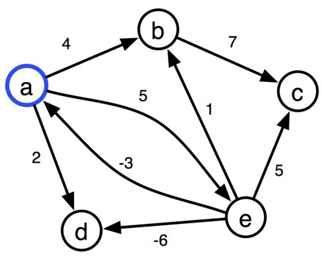
Draw the spanning tree of the shortest paths starting with node a.





Bellman-Ford

Run the Bellman-Ford algorithm on the graph below. Source is a. You must apply the relaxation procedure on the edges in the following order: (b,c), (e,a), (e,b), (e,c), (e,d), (a,b), (a,d), (a,e). Draw figures for each iteration of the algorithm.



Bellman-Ford Algorithm

```
1    d[s] ← 0
2    for each v ∈ V – {s}
3       do d[v] ← ∞
4    for i ← 1 to |V| -1  do
5       for each edge (u, v) ∈ E  do
6          if d[v] > d[u] + w(u, v)  then
7             d[v] ← d[u] + w(u, v)
8    π[v] ← u
9    for each edge (u, v) ∈ E
10      do if d[v] > d[u] + w(u, v)
11         then report that a negtive-weight cycle exists.
     At the end: d[v] = δ(s, v). Zeit: O(VE)
```

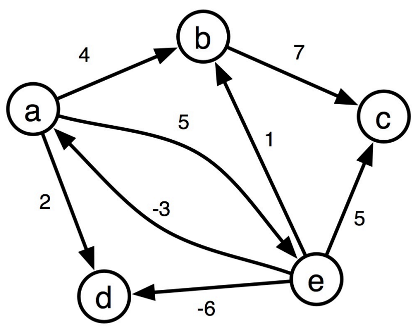
Rows 1–3: Initialize the nodes

k-th loop of row 04 – 08 computes the distance of the shortest path d[v] with at most *k* edges

Rows 6–7: Relaxe the nodes

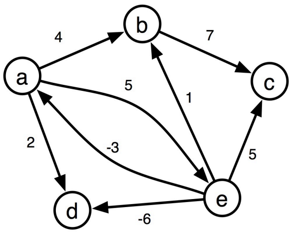
Exercise

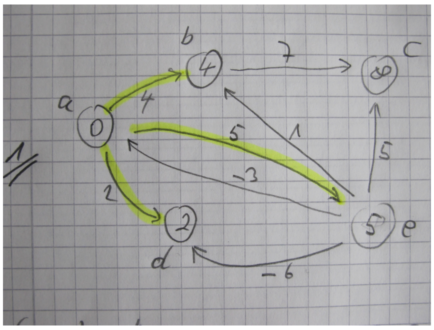
Run the Bellman-Ford algorithm on the graph below. Source is node a. You must apply the relaxation procedure on the edges in the following order: (b,c), (e,a), (e,b), (e,c), (e,d), (a,b), (a,d), (a,e). Draw figures for each iteration of the algorithm.



Exercise – 1/6

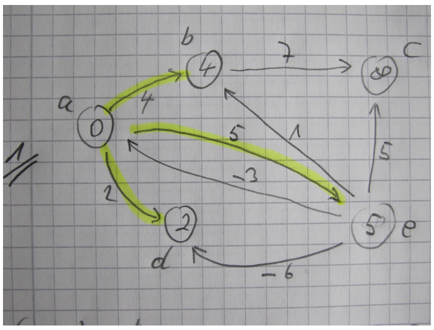
Relax in this order: (b,c), (e,a), (e,b), (e,c), (e,d), (a,b), (a,d), (a,e)

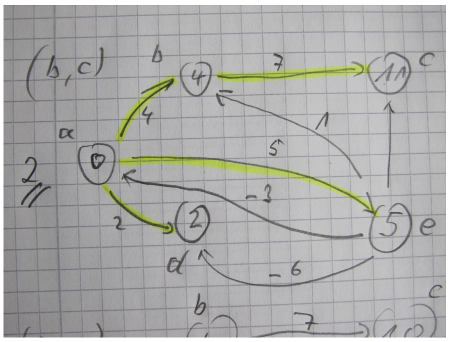




Exercise – 2/6

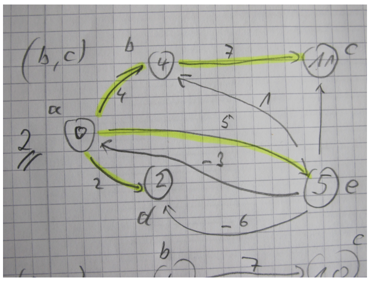
Relax in this order: (b,c), (e,a), (e,b), (e,c), (e,d), (a,b), (a,d), (a,e)

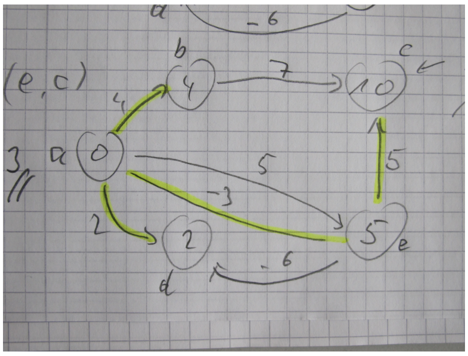




Exercise – 3/6

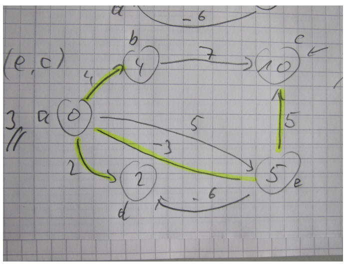
Relax in this order: (b,c), (e,a), (e,b), (e,c), (e,d), (a,b), (a,d), (a,e)



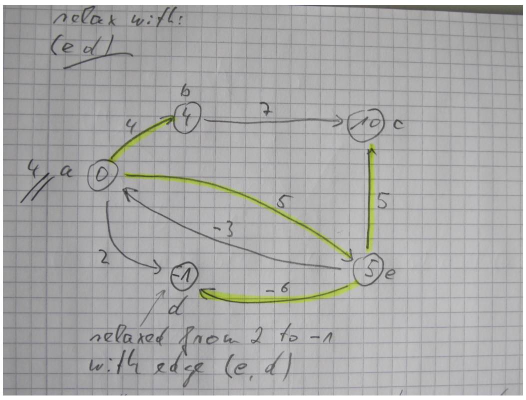


Exercise – 5/6

Relax in this order: (b,c), (e,a), (e,b), (e,c), (e,d), (a,b), (a,d), (a,e)

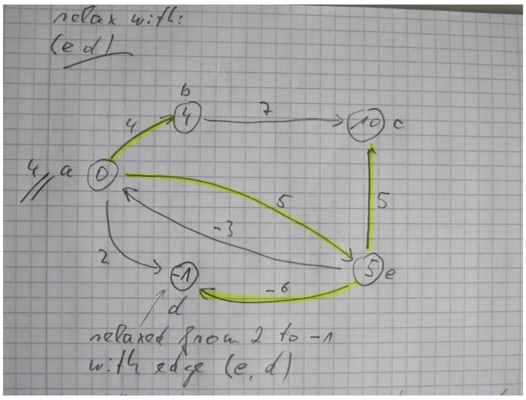


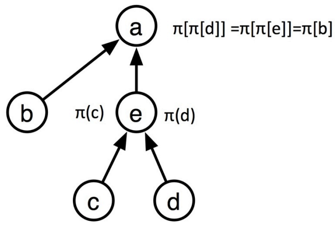
the marker (e to a) is wrong, the edge
(a to e) should be marked



Exercise – 6/6

Draw the spanning tree of the shortest paths





Dijkstra vs. Bellman-Ford

Why can Bellman-Ford operate with negative edges and Dijkstra not?

Bellman-Ford relaxes all nodes many times. The algorithm improves the shortests paths again and agian until all relax-steps are applied.

Dijkstra never relaxes a node. He is greedy and holds a shortest path as soon as it is found.