Week 12 — Sheet 11

Algorithms and Data Structures

11.12.2023 — Georg Hasebe

THANK YOU!❤️

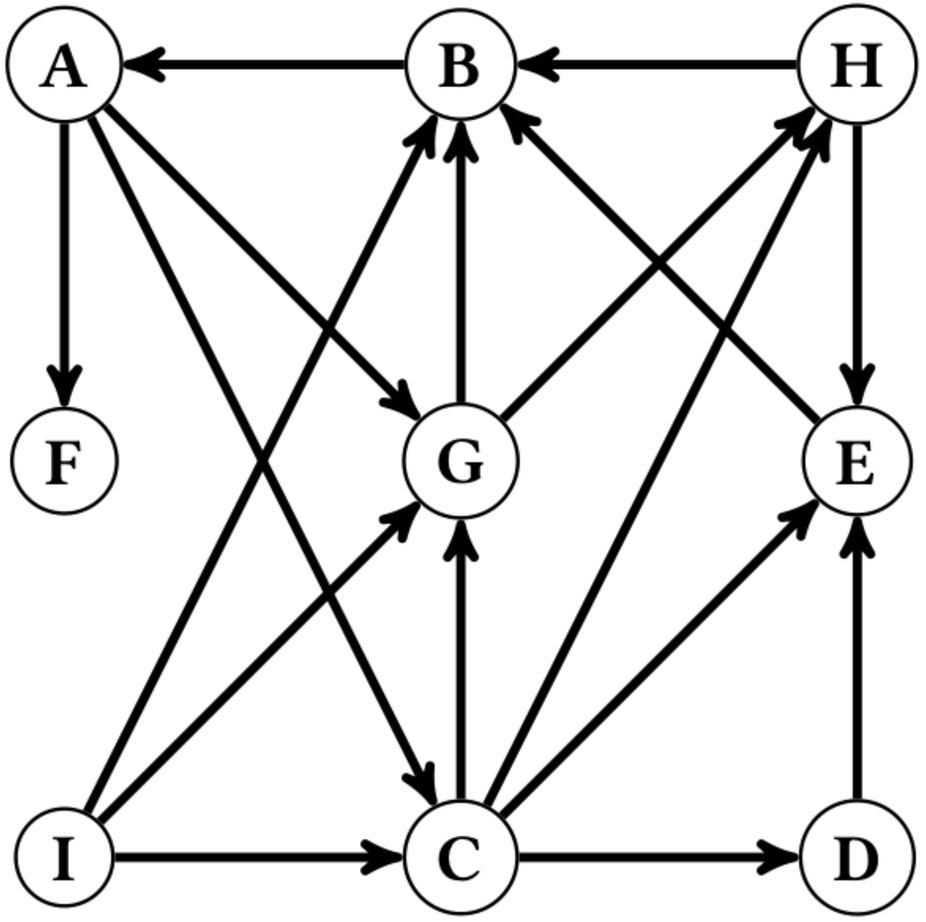
(won VIS Teaching Award!)

Exercise Sheet 11

Exercise 11.1

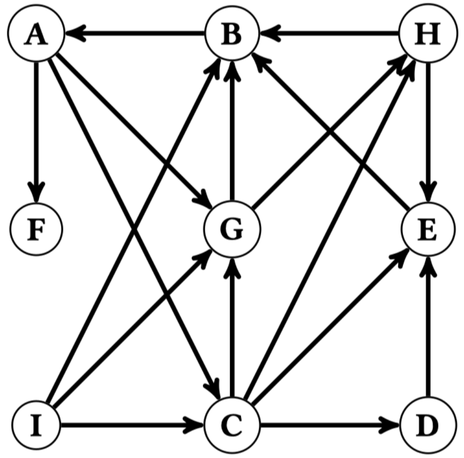
Exercise 11.1 *Breath-first search* **(1 point)**.

Execute a breadth-first search (Breitensuche) on the following graph[1] starting from vertex A. Use the algorithm presented in the lecture. When processing the neighbors of a vertex, process them in alphabetical order.

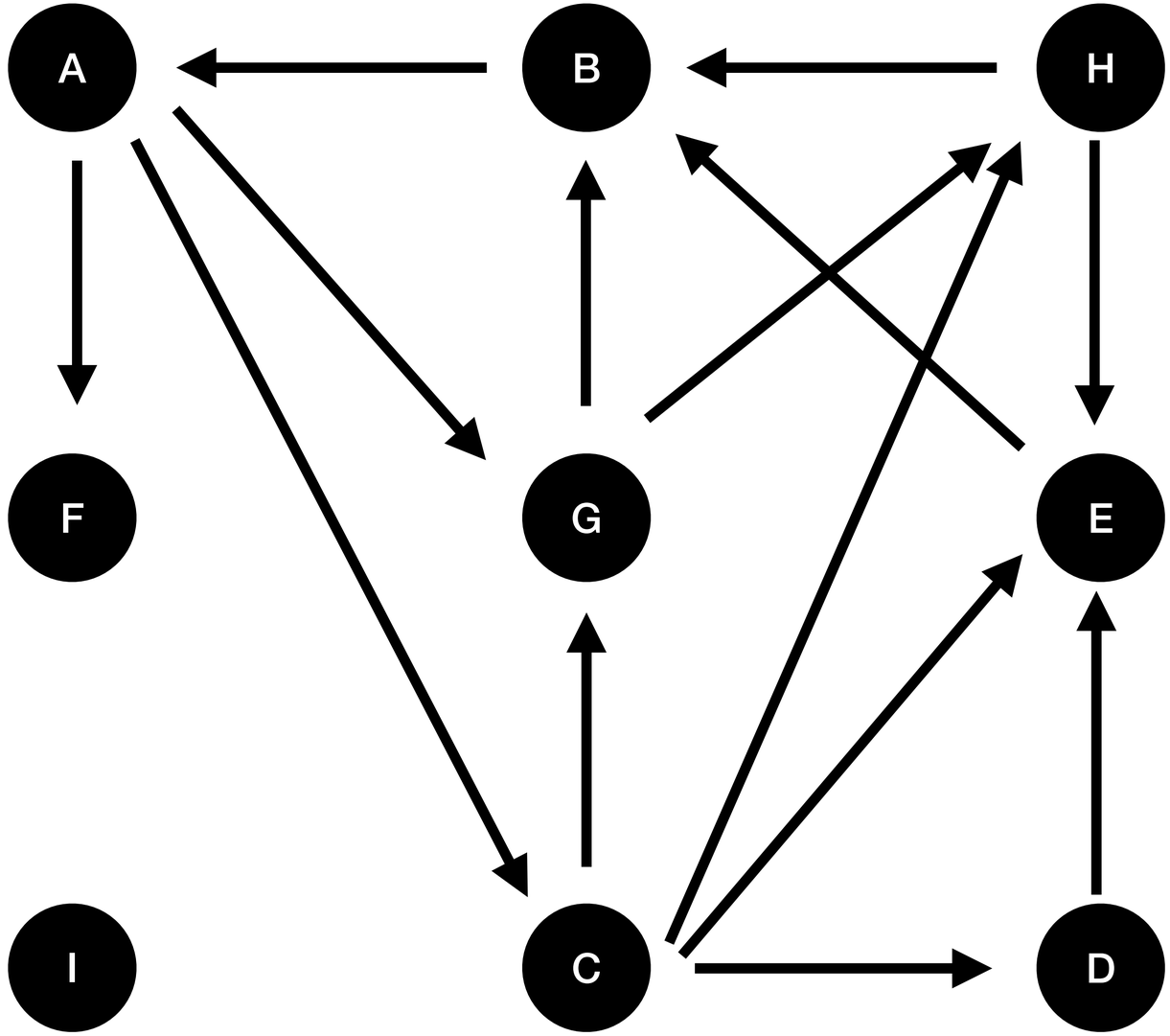


(a) For each vertex, give its enter- and leave-number.

Q:	A	C	F	G	D	E	H	B	



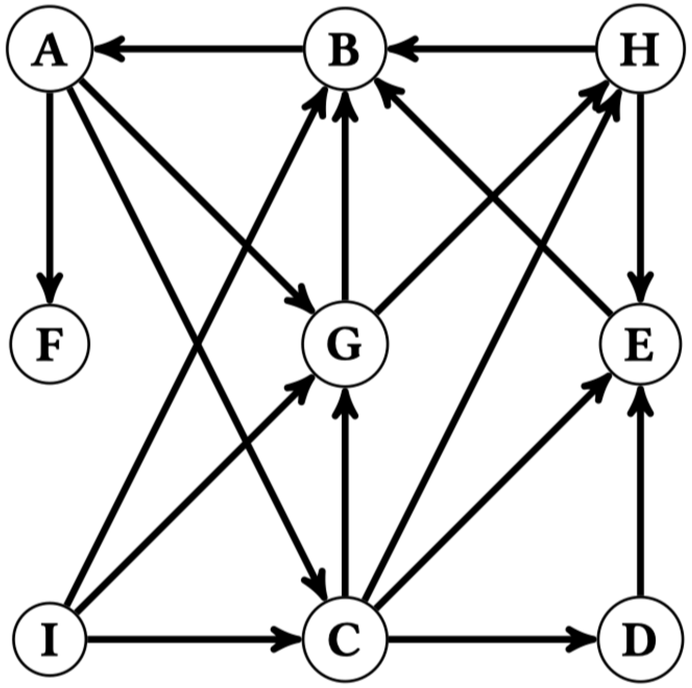
	enter	leave
A	0	1
B	11	15
C	2	5
D	6	12
E	7	13
F	3	9
G	4	10
H	8	14
I		



(b) Give the distances from A to any vertex.

$d(A, A) = 0$, $d(A, B) = 2$, $d(A, C) = 1$, $d(A, D) = 2$, $d(A, E) = 2$, $d(A, F) = 1$, $d(A, G) = 1$, $d(A, H) = 2$.

The distance $d(A, I)$ has not been assigned since I cannot be reached from A.

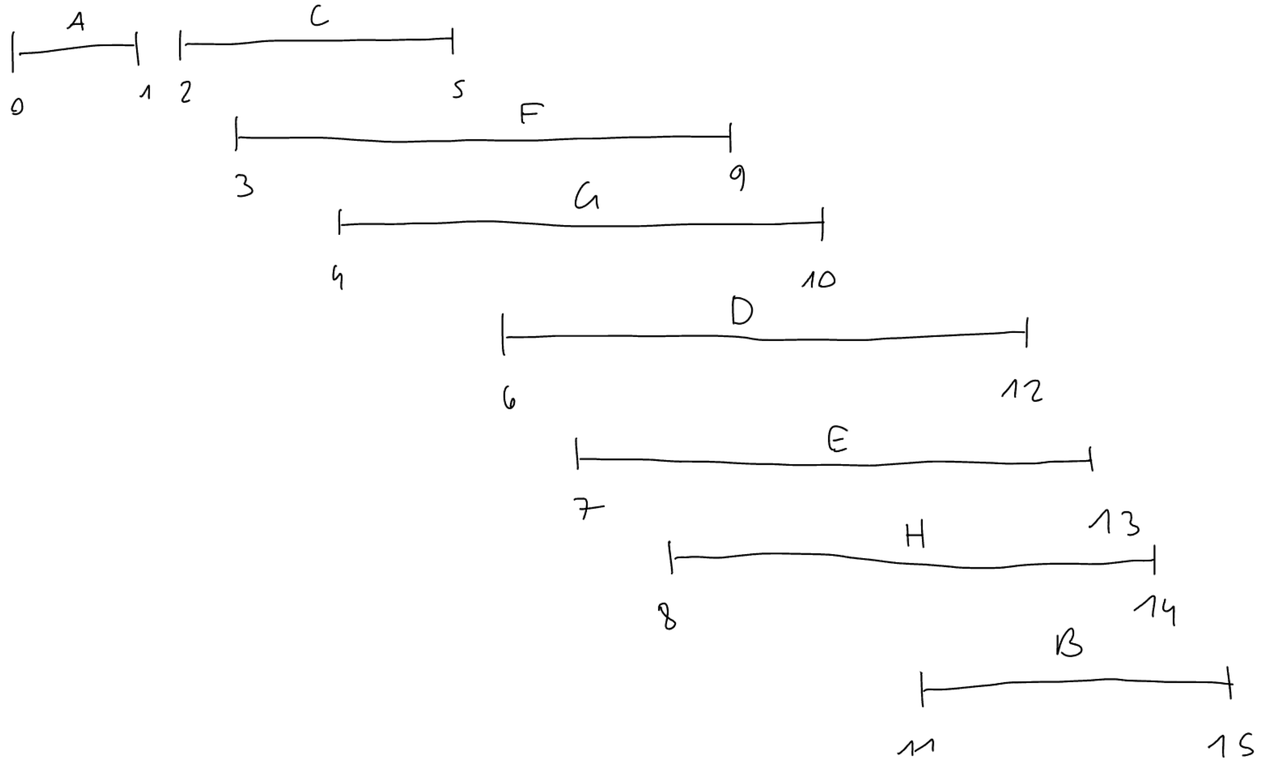


(c) Give the ordering of the vertices that results from sorting them by enter-number (or equivalently by leave-number).

	enter	leave
A	0	1
B	11	15
C	2	5
D	6	12
E	7	13
F	3	9
G	4	10
H	8	14
I		

(A,C,F,G,D,E,H,B)

(d) Draw a scale from 1 to 18 and mark for every vertex v the interval I_v we get from the above execution of breadth-first search. Is it possible that for some vertices $v \neq w$ we have $I_v \subseteq I_w$ (for a general graph G)? Why/Why not?



Let v and w be two distinct vertices of G. We have that $I_v \subseteq I_w$ is equivalent to enter$[v] <$ enter$[w]$ and leave$[w] <$ leave$[v]$ (note that the inequalities are strict since $v \neq w$). We know from the lecture that if enter$[v] <$ enter$[w]$, then also leave$[v] <$ leave$[w]$ (the enter- and leave- order are the same). Hence, we cannot have $I_v \subseteq I_w$ for any two distinct vertices v and w.

Exercise 11.4 *Driving from Zurich to Geneva* **(1 point).**

Bob is currently in Zurich and wants to visit his friend that lives in Geneva. He wants to travel there by car and wants to use only highways.

His goal is to get to Geneva as cheap as possible.

He has a map of the cities in Europe and which ones are connected by highways (in both directions).

For each highway connecting two cities he knows how much fuel he will need for this part (depending on the length, condition of the road, speed limit, etc.) and how much this will cost him.

This cost might be different depending on the direction in which he travels.

Furthermore, for some connections between two cities, he has the option to take a passenger with him that will pay him a certain amount of money. Again this might be different depending on the direction he travels.

Exercise 11.4 *Driving from Zurich to Geneva* **(1 point).**

We assume that this option is only availabe to him between cities directly connected by a highway and that the passengers want to travel the direct road and would not agree to making a detour.

Also Bob has a small car, so he can only take at most one passenger with him.

It is possible that he gains more money from this than he has to pay for the fuel between two given cities but we assume that he has no way to gain an infinite amount of money, i.e. there is no round-trip from any city that earns him money.

Exercise 11.4 *Driving from Zurich to Geneva* **(1 point).**

- as cheap as possible — Shortest path
- in both directions — ~~Undirected~~
- how much this will cost him — Weighted Graph; Edge cost
- different depending on the direction in which he travels — Directed!
- pay him a certain amount of money — Gain money; lowers edge cost (lower? Negative!)
- different depending on the direction he travels
- directly connected by a highway; want to travel the direct road; no detour — Only over one edge
- at most one passenger
- It is possible that he gains more money; no way to gain an infinite amount of money; no round-trip from any city that earns him money — No negative cycles

Exercise 11.4 *Driving from Zurich to Geneva* **(1 point).** Subtask (b)

Now we change the problem slightly. Bob got a list from his friend of certain highways that are in a bad condition.

To not damage his car, he decided that he want to use at most one of these highways.

Again, model the problem as a graph problem such that you can directly apply one of the algorithms in the lecture, without modifications to the algorithm:

Exercise 11.4 *Driving from Zurich to Geneva* **(1 point).** Subtask (b)

- highways that are in a bad condition
- to use at most one of these highways

Some edges change in some sense

Only use one single bad edge

Theory Recap

Kruskal’s Algorithm

Kruskal's Algorithm

Minimum Spanning Tree

- Create a forest (a set of trees) initially consisting of a separate single-vertex tree for each vertex in the input graph.
- Sort the graph edges by weight. (increasing order)
- Loop through the edges of the graph, in ascending sorted order by their weight (from cheapest to most expensive). For each edge:
  - Test whether adding the edge to the current forest would create a cycle (let's say we have e = {u,v}; if u,v are already in the same tree than there exists a undirected path from u to v; adding e to the tree would create a cycle)
  - If not, add the edge to the forest, combining two trees into a single tree

Kruskal's Algorithm

Pseudocode

```
Kruskal(G):

F ← ∅        (sichere Kanten)

for uv ∈ E, aufsteigend sortiert

    if u, v in verschiedenen ZHKs von F

        F ← F ∪ {uv}
```

Kruskal's Algorithm

Pseudocode (Wikipedia)

MAKE-SET, FIND-SET, UNION? How?

```
algorithm Kruskal(G) is
    F:= ∅
    for each v in G.V do
        MAKE-SET(v)
    for each {u, v} in G.E ordered by weight({u, v}), increasing do
        if FIND-SET(u) ≠ FIND-SET(v) then
            F := F ∪ { {u, v} }
            UNION(FIND-SET(u), FIND-SET(v))
    return F
```

Kruskal's Algorithm

union-find data structure (disjoint-set data structure)

We want: **MAKE-SET**, **FIND-SET** and **UNION**.

MAKE-SET: creates a set for every $v \in V$. This can be thought of as initially having $|V|$ ZHK.

FIND-SET: finds the sets (ZHK) of u, v from some edge $e = \{u, v\}$. This is important to not create cycles (refer to first Kruskal's algorithm slide).

UNION: in case we add some edge $e = \{u, v\}$ we want to merge the sets (ZHK) of u, v together.

Kruskal's Algorithm

union-find data structure (disjoint-set data structure)

A few implementation details:

- **MAKE-SET** is only used once in the beginning.
- How to implement a set (there are different ways of doing so)
  - A set usually has a rank that indicates the size of the set. This is important for runtime optimizations (add the smaller set to the bigger set is faster than adding the bigger set to the smaller one)
  - A node from a some set usually has a parent. If the parent of x is x (i.e. itself) it can be thought of the root of the set or the representant of a set (ZHK). This is used in order to know which vertex belongs to which set (ZHK).

Kruskal's Algorithm

union-find data structure (disjoint-set data structure)

A few implementation details:

- **FIND-SET** tries to find the set some $v \in V$ belongs to. This can be implemented in a few ways (path compression/halving/splitting see **Wikipedia**)

- **UNION** merges two sets. The representant of the bigger set (i.e. the set with more nodes, meaning bigger size/rank) becomes the representant of the smaller set.

Kruskal's Algorithm

union-find data structure (disjoint-set data structure)

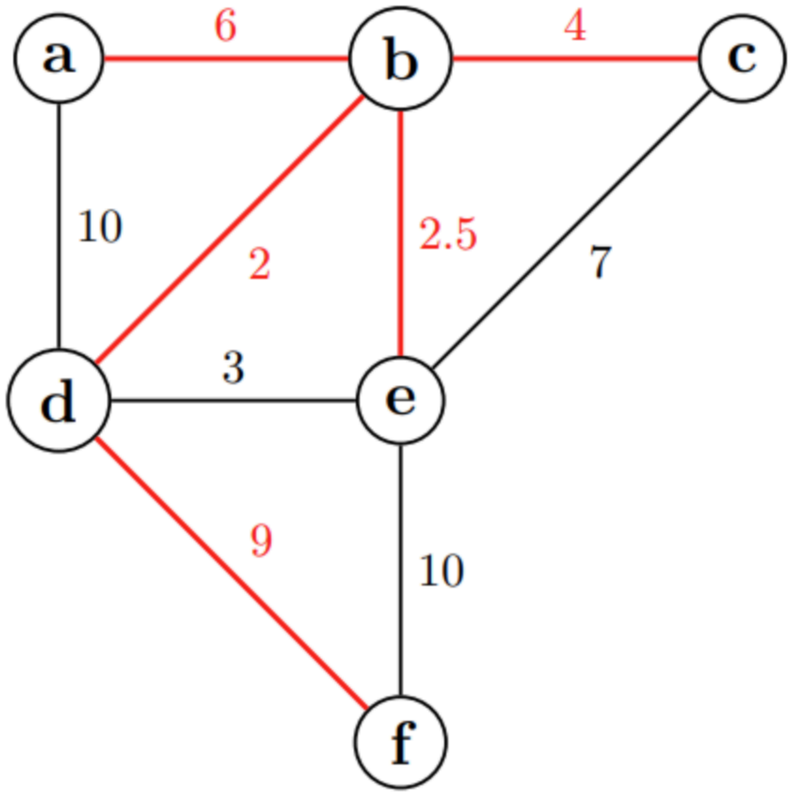
A few implementation details:

- Union-Find data structures can be a bit confusing and it really helps to try to visualize it.
  - Sets become trees.
  - The idea of parents can be visualized as Parent pointer trees.
  - Go through examples!

Kruskal's Algorithm

Useful for exam

e) *Minimum Spanning Tree:* Consider the following graph:



From FS23 https://exams.vis.ethz.ch/exams/b78ly1wf.pdf