Plan

- Administrative Information
- Discussion of Assignment 8
- Theory Recap
- Exam Task
- Peer-Grading (ex. 9.4)

Administrative Information

New groups for the last few weeks

- Can also be found in the polybox
- ❗ Groups 10 and 11 are groups of 3
- If any problems occur let me know

Siro Aebersold	saebersold@student.ethz.ch	1
Yathusan Anpalagan	yanpalagan@student.ethz.ch	13
Benedikt Baumgarten	bbaumgarten@student.ethz.ch	3
Beatrice Bold	bebold@student.ethz.ch	9
Yilmaz Bozkan	ybozkan@student.ethz.ch	1
Thomas Bundi	tbundi@student.ethz.ch	14
Cara Cerny	ccerny@student.ethz.ch	7
Adrian Efford Fernandez	aefford@student.ethz.ch	13
Ehad Evgin	eevgin@student.ethz.ch	6
Franco Fierro Brandes	ffierro@student.ethz.ch	3
Fabio Filipponi	ffilipponi@student.ethz.ch	15
Cyril Gremaud	cgremaud@student.ethz.ch	11
Julian Isser	jisser@student.ethz.ch	2
Maximilian Käfer	mkaefer@student.ethz.ch	10
Ashley Kellenberger	akellenberge@student.ethz.ch	2
Alex Lanz	lanzal@student.ethz.ch	9
Ruilai Li	ruilli@student.ethz.ch	8
Anna Limpaset	alimpaset@student.ethz.ch	5
Hafiz Ullah Mija Khel	hmijakhel@student.ethz.ch	12
Yannick Niederer	yniederer@student.ethz.ch	15
Nikol Nikolova	nnikolova@student.ethz.ch	7
Jannis Piekarek	jpiekarek@student.ethz.ch	5
Kiyan Rassouli	krassouli@student.ethz.ch	12
Lars Schell	laschell@student.ethz.ch	10
Camil Schmid	camschmid@student.ethz.ch	14
Dominik Schwaiger	dschwaiger@student.ethz.ch	4
Raoul Sidler	sidlerr@student.ethz.ch	4
Jonathan Soemer	jsoemer@student.ethz.ch	6
Kristof Szadeczky-Kardoss	kszadeczky@student.ethz.ch	10
Vinzenz Wolf	wolfvi@student.ethz.ch	11
Helena Yan	helyan@student.ethz.ch	8
Guohong Zhang	zhanggu@student.ethz.ch	11

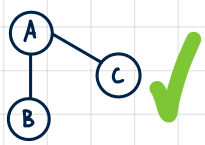
Updated graph cheatsheet in the Polybox

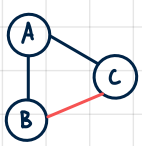
VIS Teaching Awards

- Vote for your favorite TA (doesn't have to be me ☺)
- Explain your choice (!)
  - ↳ Winner isn't necessarily the one with the most votes

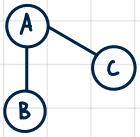
Discussion Assignment 8

+ excellent solutions overall !!
+ great proofs in 8.5
- drawing "no edge" as an edge with a different color

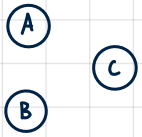




- three vertices are not pairwise adjacent ≠ three vertices are pairwise not adjacent



not pairwise adjacent ✓
pairwise not adjacent ✗



not pairwise adjacent ✓
pairwise not adjacent ✓

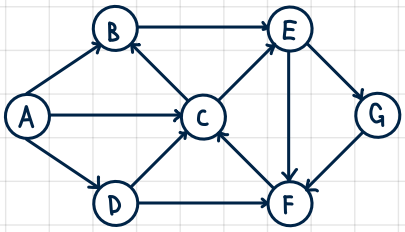
Breadth-First-Search (BFS)

- One of the most important algorithms (just like DFS)
  - ↳ very wide range of use (you will see some use cases in the next semester)
  - ↳ learn to implement this by heart, you will be using it a lot
- Runtime $O(|V|+|E|)$

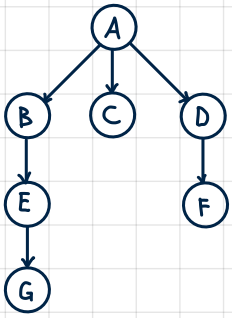
Algorithm 5 BFS(s)

```
1: Q ← {s}                                          ▷ Q ist eine Queue
2: enter[s] ← 0;   T ← 1
3: while Q ≠ ∅ do
4:     u ← dequeue(Q)
5:     leave[u] ← T;   T ← T + 1
6:     for (u, v) ∈ E, enter[v] nicht zugewiesen do
7:         enqueue(Q, v)
8:         enter[v] ← T;   T ← T + 1
```

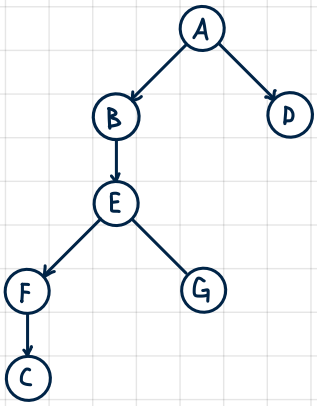
Example



BFS-Tree



DFS-Tree (for comparison)



Dijkstra's Algorithm

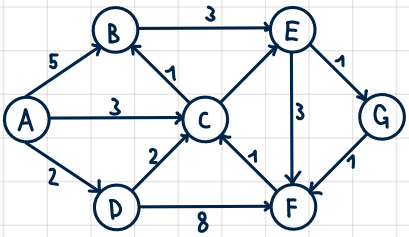
- First shortest path algorithm
- Very famous
- Only works if graph only has non-negative edge-weights
- Rather tricky to implement fast (atleast in Java)

Algorithm 6 Dijkstra(s)

```
1: d[s] ← 0;   d[v] ← ∞ ∀v ∈ V \ {s}
2: S ← ∅
3: H ← make-heap(V); decrease-key(H, s, 0)
4: while S ≠ V do
5:     v* ← extract-min(H)
6:     S ← S ∪ {v*}
7:     for (v*, v) ∈ E, v ∉ S do
8:         d[v] ← min{d[v], d[v*] + c(v*, v)}
9:         decrease-key(H, v, d[v])
```

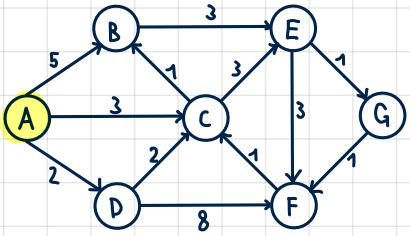
Example

= final
= unchanged (this iteration)
= changed (this iteration)



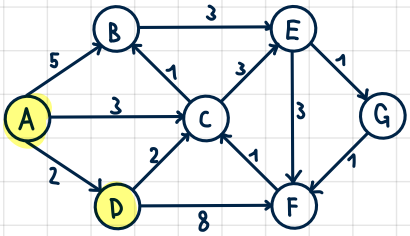
S = {}

v:	A	B	C	D	E	F	G
d[v]	0	∞	∞	∞	∞	∞	∞



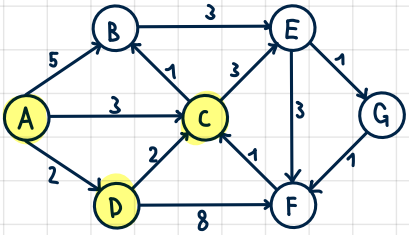
S = {A}

v:	A	B	C	D	E	F	G
d[v]	0	5	3	2	∞	∞	∞



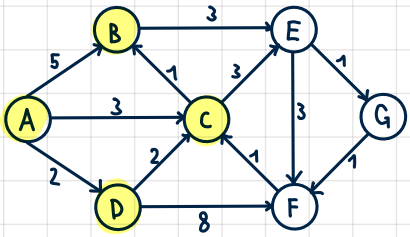
S = {A, D}

v:	A	B	C	D	E	F	G
d[v]	0	5	3	2	∞	10	∞



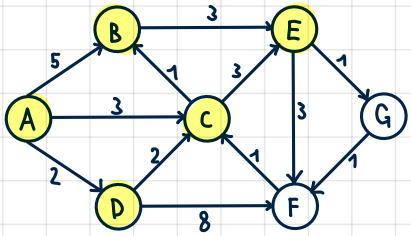
S = {A, D, C}

v:	A	B	C	D	E	F	G
d[v]	0	4	3	2	6	10	∞



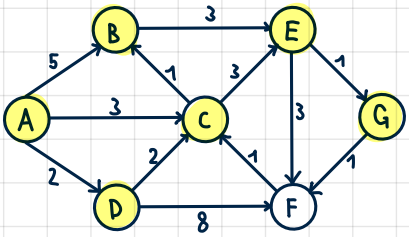
S = {A, D, C, B}

v:	A	B	C	D	E	F	G
d[v]	0	4	3	2	6	10	∞



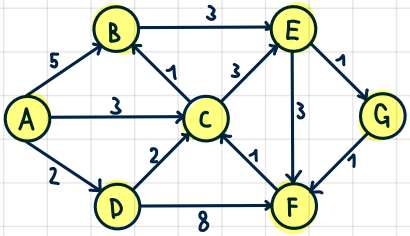
S = {A, D, C, B, E}

v:	A	B	C	D	E	F	G
d[v]	0	4	3	2	6	9	7



S = {A, D, C, B, E, G}

v:	A	B	C	D	E	F	G
d[v]	0	4	3	2	6	8	7



S = {A, D, C, B, E, G, F}

v:	A	B	C	D	E	F	G
d[v]	0	4	3	2	6	8	7

FS19, T2

Template for the solution in Polybox (QR-Code)

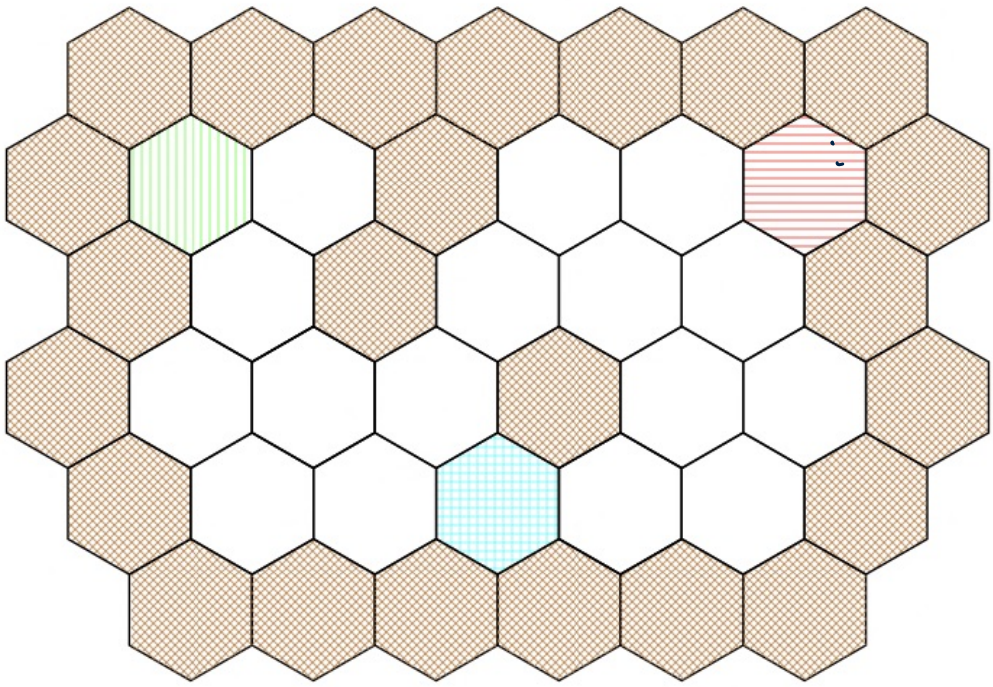
Theory Task T2.

/ 13 P

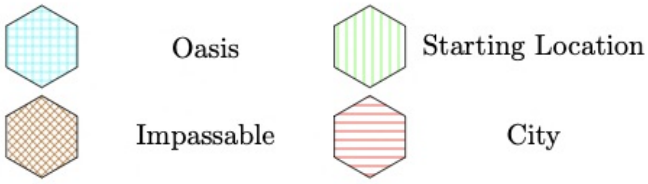
You are alone and stranded in a desert trying to reach the only city. The desert can be divided into hexagons and is surrounded by impassable cliffs. The desert terrain varies, and hexagons take differing amounts of time to traverse. There are oases in the desert.

- In this problem, you move from the center of one hexagon to the center of an adjacent hexagon.
- The amount of time spent traveling from the center of one hexagon to another is equal to half the amount of time needed to cross the first hexagon plus half the amount of time needed to cross the second.

Example:



Legend:



a) Your goal is to minimize the amount of time it takes to get from the starting location to the city. Model the problem with a graph in such a way that an algorithm from the lecture computes a solution to the problem. Describe precisely the set of vertices and edges, and describe what the vertices and edges represent. Name an, as efficient as possible, algorithm from the lecture to solve this problem, and state the running time of the algorithm in terms of the number of hexagons, N.

a.) Definition of the graph (if possible in words and not formal):

- We model the problem as an undirected graph with edge weights.
- The vertices correspond to the hexagons.
- There is an edge between two hexagons iff they are adjacent.
- The weight of the edge is half the time needed to cross one hexagon plus half the time needed to cross the other.

Number of vertices and edges (in Θ-Notation in N, as concisely as possible):

- We have N vertices.
- Each vertex has degree ≤ 6 and the graph is connected.
  - ↳ $N-1 \leq |E| \leq \frac{6N}{2}$, thus we have $\Theta(N)$ vertices and $\Theta(N)$ edges.

Algorithm (as efficient as possible):

- Since all edge weights are non-negative we can use Dijkstra's Algorithm to compute a shortest path from the starting location to the city.

Running time (as precisely as possible in Θ-Notation in terms of N). Justify your answer.

- The (worst-case) running time of Dijkstras Algorithm is $\Theta((|V|+|E|)\cdot \log|V|)$.
- We have $|V| = \Theta(N)$, $|E| = \Theta(N)$
  - ↳ Our running time is $\Theta(2N \cdot \log N) = \Theta(N \cdot \log N)$

b) We now modify the problem, adding the following rules:

- You begin a full water bottle that you can drink from D times, and you can refill it at each *oasis*.
- When you arrive at a hexagon that isn't an oasis or the city, you must drink water from your water bottle.
- If your water bottle is empty, you become dehydrated.

Your goal is to find the fastest route from the starting location to the city, without becoming dehydrated.

Model the modified problem with a graph in such a way that an algorithm from the lecture computes a solution to the problem. Describe precisely the set of vertices and edges, and describe what the vertices and edges represent. Name an, as efficient as possible, algorithm from the lecture to solve this problem, and state the running time of the algorithm in terms of the number of hexagons, N, and the size of your water bottle, D.

b.) Definition of the graph (if possible in words and not formal):

- We model the problem as an directed graph with edge weights.
- The vertices are pairs of hexagons and bottle states.
- A bottle state is a number from 1 to D, corresponding to the number of times the bottle can be drank from before being empty.
- There is an edge between two vertices if the corresponding hexagons are adjacent and going from the first hexagon with the first bottle state to the second hexagon results in the second bottle state. When reaching an oasis we refill the bottle.
- The weight of the edge is half the time needed to cross one hexagon plus half the time needed to cross the other.

Number of vertices and edges (in Θ-Notation in N, as concisely as possible):

- We have $\Theta(N \cdot D)$ vertices and $\Theta(N \cdot D)$ edges. (follows from (a))

Algorithm (as efficient as possible):

- Since all edge weights are still non-negative we can use Dijkstra's Algorithm to compute a shortest path from the starting location with bottle state D to any of the vertices corresponding to the city hexagon.

Running time (as precisely as possible in Θ-Notation in terms of N). Justify your answer.

- The (worst-case) running time of Dijkstras Algorithm is $\Theta((|V|+|E|) \cdot \log|V|)$.
- We have $|V| = \Theta(N \cdot D)$, $|E| = \Theta(N \cdot D)$
  - ↳ Our running time is $\Theta(2 \cdot N \cdot D \cdot \log(N \cdot D)) = \Theta(N \cdot D \cdot \log(N \cdot D))$