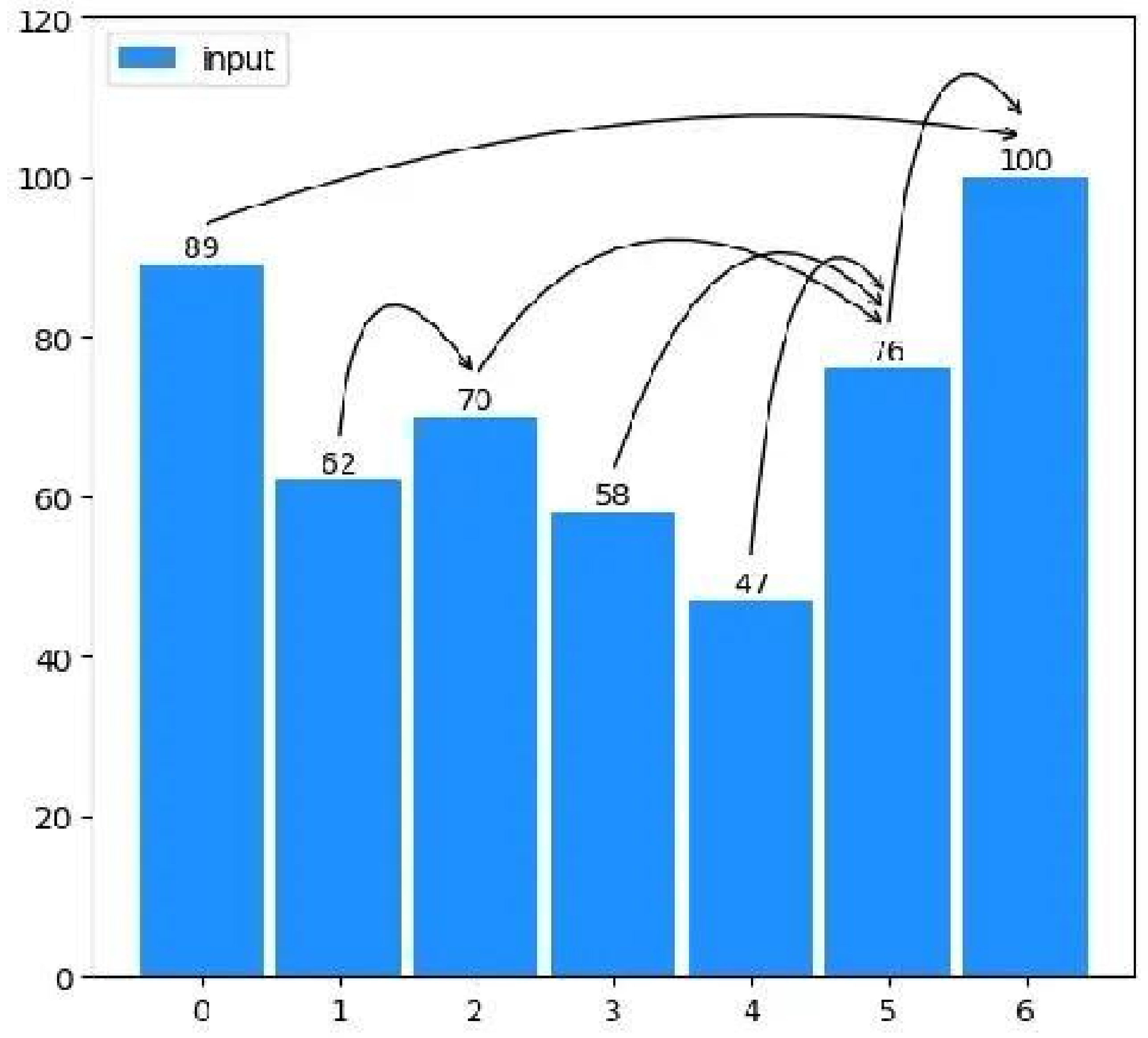
Tutorium #11

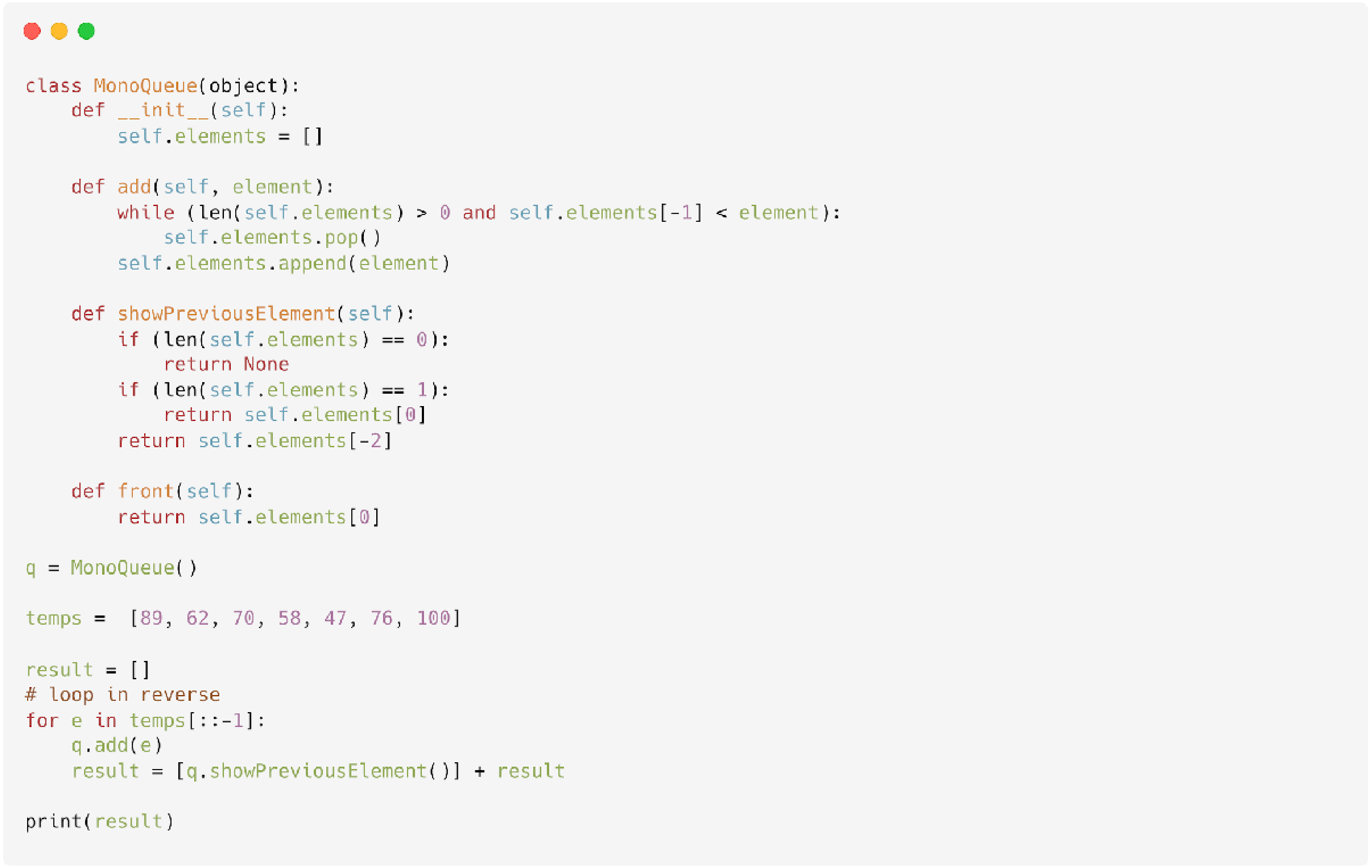
27/01/2023

Sample problem

Let's have a look at the Daily Temperatures problem, in it we are asked to find the next day with bigger temperature for each day. To solve it we are going to use decreasing MQ.

As an example let's take an array `[89, 62, 70, 58, 47, 76, 100]`. Below you can find an overview of the nearest biggest elements (to the right) for each element of the array. By doing so, we can pretty much see the result.





```python
class MonoQueue(object):
    def __init__(self):
        self.elements = []

    def add(self, element):
        while (len(self.elements) > 0 and self.elements[-1] < element):
            self.elements.pop()
        self.elements.append(element)

    def showPreviousElement(self):
        if (len(self.elements) == 0):
            return None
        if (len(self.elements) == 1):
            return self.elements[0]
        return self.elements[-2]

    def front(self):
        return self.elements[0]

q = MonoQueue()

temps =  [89, 62, 70, 58, 47, 76, 100]

result = []
# loop in reverse
for e in temps[::-1]:
    q.add(e)
    result = [q.showPreviousElement()] + result

print(result)
```

Quellen

https://1e9.medium.com/monotonic-queue-notes-980a019d5793

https://medium.com/algorithms-and-leetcode/monotonic-queue-explained-with-leetcode-problems-7db7c530c1d6

https://leetcode.com/problems/sliding-window-maximum/solutions/65885/this-is-a-typical-monotonic-queue-problem/?orderBy=most_relevant

Parallel Sorting

Quicksort main idea:

pivot p, smaller elements S, larger elements L

sorted: $\Longrightarrow$ quicksort(S) + p + quicksort(L)

Given $[10, 8, 3, 7, 4, 5]$ pivot

S 0 0 1 0 1

L 1 1 0 1 0

$$\Big[\underbrace{{}_1\ S\ {}_2}\ \Big|\ p\ \Big|\ \underbrace{L\ {}_4\ {}_5\ {}_6}\Big]$$

prefix sum

Comparison of parallel sorting algorithms
[https://arxiv.org/pdf/1511.03404.pdf]

Parallel Sorting Algorithms
[https://www.dcc.fc.up.pt/~ricroc/aulas/1516/cp/apontamentos/slides_sorting.pdf]

COMP526 (Fall 2022) 5-4 §5.4 Parallel sorting
[https://www.youtube.com/watch?v=IaPxlnBXN5E&ab_channel=SebastianWild%28Lectures%29]

Parallel quicksort algorithms
[https://www.uio.no/
studier%2Femner%2Fmatnat%2Fifi%2FINF3380%2Fv10%2Fundervisningsmateriale%2Finf338
0-week12.pdf%2F]

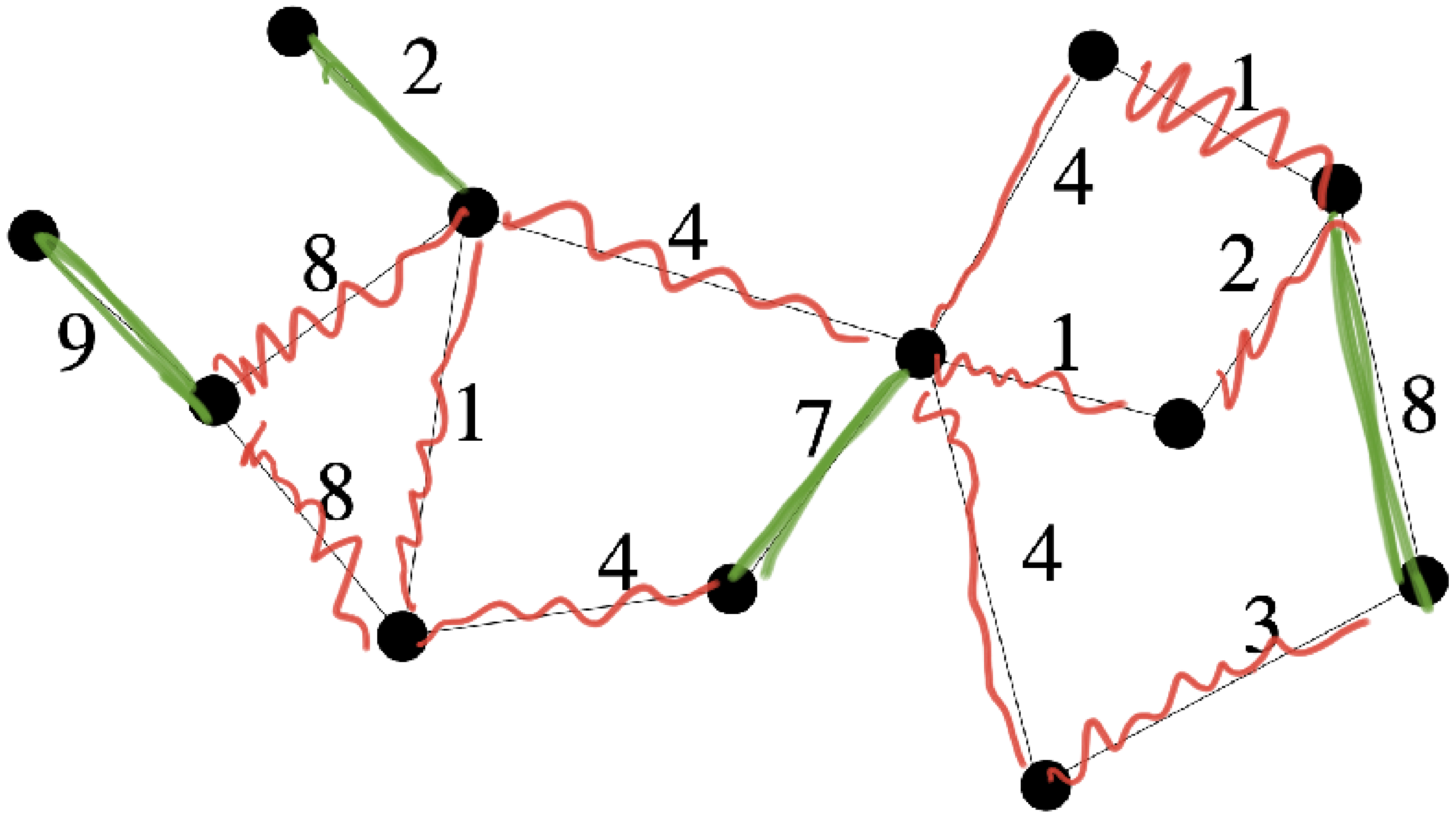
Problem 1 (8 points) Given is a graph $G = (V, E)$ with a cost function for the edges $\omega : E \to \mathbf{R}_{>0}$.

Definition: A matching $M \subseteq E$ is a set of edges that do not share any common nodes, i.e., the graph (V, M) has maximum degree one. A matching is *maximal* iff no edge can be added to the matching. A *maximum* weight matching $\mathcal{M}$ has the largest possible weight among all matchings, i.e. $\sum_{e \in \mathcal{M}} \omega(e) \geq \sum_{e \in M} \omega(e)$ for all possible matchings M.

Given is the following **greedy algorithm:**

```
$\mathcal{M}_{\text{Greedy}} := \emptyset$
**while** $E \neq \emptyset$ **do**
    take an edge $\{v, w\} \in E$ with *highest* weight;
    add $\{v, w\}$ to $\mathcal{M}_{\text{Greedy}}$
    remove all edges incident to $v$ or $w$ from $E$
```

Tasks:



1. (2 points) Give the result of the algorithm on the following graph by marking edges that will be in the final set $\mathcal{M}_{\text{Greedy}}$. The weight function ω is indicated by numbers next to the edges.
2. (2 points) Show that when the algorithm terminated, $\mathcal{M}_{\text{greedy}}$ is a matching.
3. (4 points) Show that the algorithm can be implemented in $O(m \log n)$ time.
4. (4 points) Show that $\omega(\mathcal{M}_{\text{Greedy}}) \geq \frac{1}{2}\omega(\mathcal{M}_{\text{Opt}})$ for non-negative ω where $\mathcal{M}_{\text{Opt}}$ is a maximum weight matching.

2. When the algorithm starts there are no edges in $\mathcal{M}_{\text{Greedy}}$. Therefore it is obviously an (empty) matching. After we add an edge, we remove all edges that are incident to one of the matched nodes. Therefore we can guarantee that when we add an edge to the matching, none of the adjacent vertices are already matched in $\mathcal{M}_{\text{Greedy}}$. Thus $\mathcal{M}_{\text{Greedy}}$ is still a matching. This is true for every edge added to $\mathcal{M}_{\text{Greedy}}$.

why

3. We can sort all edges by weight in $\boxed{O(m \log m) = O(m \log n).}$ As a matching can only have up to $\frac{n}{2}$ edges, we only take $O(n)$ edges and add them to $\mathcal{M}_{\text{Greedy}}$. In the process of the algorithm we remove $O(m)$ edges. The total runtime of the algorithm is therefore $O(m \log n)$.

4. For every edge in $\mathcal{M}_{\text{Greedy}}$, $e = (u, v)$, let $\mathcal{M}_e \subseteq \mathcal{M}_{OPT}$ be the set of all edges in $\mathcal{M}_{OPT}$ which have common vertices with e. As the maximum degree in a matching is 1, $|\mathcal{M}_e| \leq 2$, as both u and v can only be matched once each. As these edges in $\mathcal{M}_e$ were not chosen before e in the greedy matching, we know that their weight is equal or smaller than $\omega(e)$. Therefore the sum of their weights is $\leq 2 \cdot \omega(e)$. We now show that $\mathcal{M}_{OPT} = \bigcup\limits_{e \in \mathcal{M}_{\text{Greedy}}} \mathcal{M}_e$. Assume that an edge in $\mathcal{M}_{OPT}$ is in no $\mathcal{M}_e$. The algorithm would have added it to $\mathcal{M}_{\text{Greedy}}$, which results in a contradiction to the fact that it is not in $\mathcal{M}_e$ for any matched edge e. We get:

   $\omega(\mathcal{M}_{OPT}) \leq \sum_{e \in \mathcal{M}_{\text{Greedy}}} \omega(\mathcal{M}_e) \leq 2 \cdot \omega(\mathcal{M}_{\text{Greedy}})$,

   which shows the desired approximation rate of 2.

Problem 2 (8 points)

Let $G = (V, E)$ be a connected undirected graph with more than one vertex. The maximum cut problem tries to find sets A and B with $A \cup B = V$, $A \cap B = \emptyset$ such that the number of edges that run between A and B is maximum. The value of the maximum cut is denoted $\mathcal{M}$ within this task.

We define the following algorithm. Our algorithm starts with a partition of the node set into two sets A and B such that A and B are both nonempty, e.g. assign the first half to A and the second half of the vertices to B. The gain of the vertex v in A is defined as $g(v) := |N(v) \cap A| - |N(v) \cap B|$ and if the vertex is in B then $g(v) := |N(v) \cap B| - |N(v) \cap A|$ where $N(v)$ denotes the set of neighbors of v. The algorithm then proceeds in rounds. In each round the algorithm looks at every vertex and moves a vertex from its set to the opposite set if it has positive gain. The algorithm stops when no positive gain vertex is left.

1. Explain the intuition of the gain function.
2. Explain why this is a greedy algorithm.
3. Show that the algorithm terminates after at most $\mathcal{M}$ rounds.
4. Show that the running time of the algorithm is $O(\mathcal{M}(n + m))$
5. Show that when the algorithm terminates, A and B are nonempty.
6. Show that the algorithm is a 1/2-approximation algorithm.

2.1 The gain function measures the total increase in cut size in case a given node v is moved to the other block. If the gain is negative, its absolute value measures the total decrease in cut size if v is moved to the other block

2.2 The algorithm only makes decisions which strictly increase the total cut size. This makes it a greedy algorithm. local

2.3 The algorithm starts with a total cut greater than or equal to 0 and increases it by at least 1 after each round. Hence, the algorithm can have at most $\mathcal{M}$ rounds.

2.4 In the last answer, we showed that the algorithm can have at most $\mathcal{M}$ rounds. On the other hand, one round requires us to look all all edges of each node to compute the gains, hence a round costs $n + m$. Summing up, the overall running time of the algorithm is $O(\mathcal{M}(n + m))$.

So you don't recompute the gain, but update it

$\forall v \in V(G)$: we update the gain of $N_G(v)$

$\sum_i \deg(v) = 2m \Rightarrow O(m)$

2.5 Since each set starts with $\frac{n}{2}$ nodes and there are at least 2 vertices in G, then no set starts empty. Hence a set could only become empty after at least one movement of nodes by the algorithm. By the definition of cut, the total cut size becomes 0 if one of the two sets becomes empty. Since the cut size $\underline{\text{increases at least by 1 after}}$ each movement of nodes, hence no set can become empty by simply applying rounds of the provided algorithm.

2.6 Let z be the total edge cut after the algorithm stops. When the algorithm stops, no vertex with positive gain remains. As a consequence:

$$z \geq \frac{1}{2} \sum_{v \in A} |N(v)|; \qquad z \geq \frac{1}{2} \sum_{v \in B} |N(v)|$$

(since otherwise swap vertices and get a better cut)

Summing up both inequalities above, we have:

$$2z \geq \underbrace{\frac{1}{2} \sum_{v \in V} |N(v)| = m}_{} \geq \underset{\substack{\uparrow \\ \text{max. opt. value}}}{\mathcal{M}} \Rightarrow z \geq \frac{1}{2}\mathcal{M}$$

$\sum \deg = 2m$