1. **Greedy graph coloring.** Let $G = (V, E)$ be a graph with n nodes. We want to give a proper coloring $c : V \to \mathbb{N}$ of the vertices. Consider the following naive greedy algorithm.

   1. Pick an arbitrary vertex.
   2. Give it the smallest possible value.

   This algorithm has an upper bound of $\Delta + 1$ colors.

   (a) (2p.) Design a new greedy algorithm such that the number of colors is bounded by $\max_{1 \leq i \leq n} \min(d_i + 1, i)$ where d_i is the degree of the node v_i that gets colored in iteration i. Prove the correctness of the algorithm.
   Hint: Think about ordering the nodes.

The improved greedy algorithm is as follows:

(1) Order the vertices according to their degrees in non-increasing order. In other words, after sorting the vertices, we have: $\deg(v_1) \geq \deg(v_2) \geq ... \geq \deg(v_n)$

(2) Pick each vertex subsequently after they have been sorted

(3) Give it the smallest possible value in c

Prove of the upper bound of the improved greedy algorithm:

For each $v_i, v_2, ..., v_{i-1}$, we observe that the vertex v_i has at most $\min(\deg(v_i), i-1)$ neighbors among the previous sorted vertices $v_1, v_2, ..., v_{i-1}$. Because the upper bound of the chromatic number is $\chi(G) \leq \Delta(G) + 1$ and $\Delta(G)$ is determined by the vertex with the most neighbors

⇨ $\Delta(G) = \max_{1 \leq i \leq n} \deg(v_i) = \max_{1 \leq i \leq n} \min(\deg(v_i), i-1)$

Plugging into the formula, we have:
$\chi(G) \leq \max_{1 \leq i \leq n} \min(\deg(v_i), i-1) + 1 = \max_{1 \leq i \leq n} \min(\deg(v_i) + 1, i)$ (proven)

(b) (2p.) An interval graph is a graph that corresponds to a family of intervals $\{I_u\}_{u \in V}$ of $[0, 1]$ such that $\{u, v\} \in E \Leftrightarrow I_u \cap I_v \neq \emptyset$. Design an optimal graph coloring algorithm for interval graphs using a greedy approach and prove that it is correct. You can assume that the intervals corresponding to the interval graph are given as input.
Hint: Think about ordering the nodes again.

Greedy algorithm for interval graphs

(1) Sort the intervals in decreasing order according to their starting time:
$start(v_1) \leq start(v_2) \leq ... \leq start(v_n)$
(2) for i in range(1, n):
Assign to v_i the smallest color that has not been assigned to the previous starting vertices that intersect the duration of v_i (works like the naive greedy algorithm in part (a))

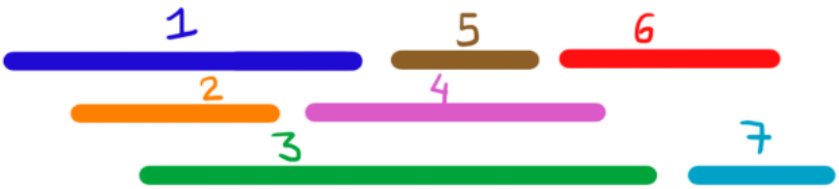
Proof of the optimality

Let k be the chromatic number at the end of the algorithm. At the moment when the color k is used, we know that all previous vertices use colors from 1 to k - 1. In the interval, the algorithm only allows a vertex to occupy a color from its start to finish time, which indicates that the set of vertices in the schedule contains k mutually intersecting vertices. This means that k is a lower bound of the chromatic number and since our algorithm uses exactly k colors, it uses the least number of colors

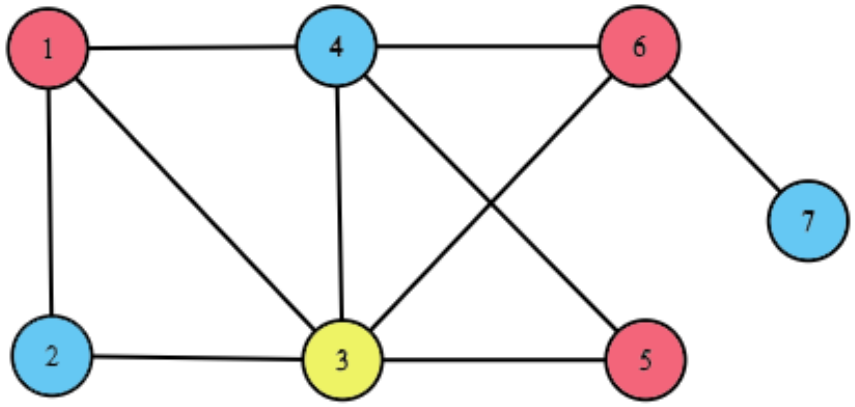
⇨ This algorithm is optimum for interval graphs

(c) (1p.) Show the first greedy algorithm with an arbitrary ordering is not optimum, not even for interval graphs.

Suppose that the durations in the schedule are given as follows:



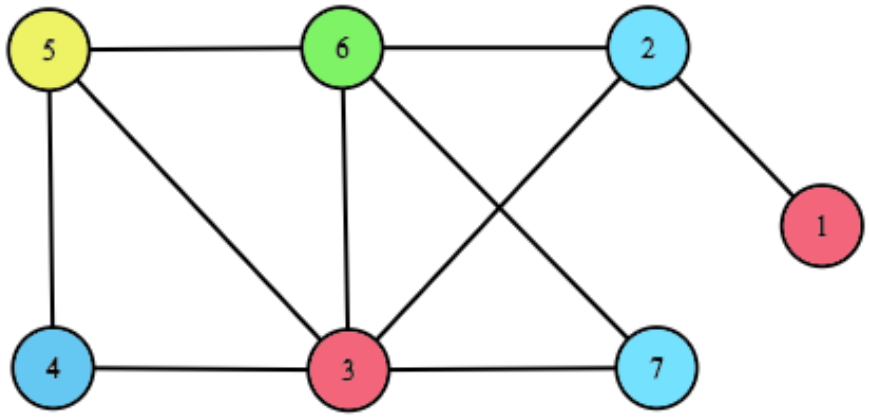
The numbers are given according to their starting time. From the interval graph, we sort the vertices based on their starting time. Then we proceed to color like the first greedy algorithm:



First, vertex 1 starts first so it is colored red. At vertex 2, it intersects vertex 1 so it needs blue. For vertex 3, it intersects both vertex 1 and 2 so it needs yellow. For vertex 4, it only intersects vertex 1 and 3 so the smallest color is red again. This works until we color all of the graph.

⇨ This interval graph needs only 3 colors

However, if we only use the first greedy algorithm without sorting, this color ordering yields 4 colors, which is not optimum for this graph => First greedy algorithm is not optimum for interval graphs



2. **Individual exercise: Greedy coloring of bipartite graphs.** A greedy algorithm for graph coloring of bipartite graphs uses the *color-degree* of each node i.e. the number of already colored neighbors. The algorithm is the following:

   1. The color-degree of each node is initialized to 0.
   2. Choose a node v with maximum color-degree and give it the smallest possible color.
   3. Update the color-degree of its neighbors.

   (a) (2p.) Show that this algorithm is optimum for bipartite graphs.
   Hint: you can use the fact that a graph is bipartite iff it has no odd cycles.

A bipartite graph is a graph consists of two disjoint sets of vertices L (left) and R (right), such that there is no edges that connect vertices in the same set. In other words, an edge in a bipartite graph only connects one vertex from L and one from R. In order to create a cycle, first we have go from a vertex in L to one in R, then from R to another vertex in L. Next, we go from L to another vertex in R and finally return back to the original vertex in L, taking 4 edges. The cycle can only be lengthened by alternating between the two groups in a pair of edges. Therefore, the bipartite can only have even length cycles. An even-length cycle can be colored with only 2 colors by alternation. If it is not a cycle, then it is a simple line with even length which also requires 2 colors. Therefore, for every bipartite graph, $\chi\left(G_{bipartite}\right)=2$, where one color only colors vertices in L and another only colors vertices in R.

Correctness of the algorithm: We can use the induction proof to verify that the algorithm returns the optimum, which is 2 colors.

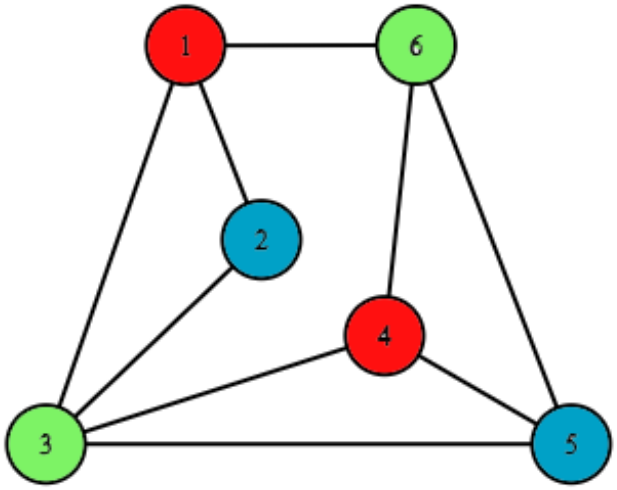
1. Base case:
   From the start, we know that all color-degree is 0. Suppose we pick one vertex from L and color it with color #1. When we go to R, we update the color-degree of all vertices connected to the one in L to 1, because there is only one colored neighbor. We color one vertex from R with degree 1 with the a new color #2. Now we have one vertex from L with color #1 and one vertex from R with color #2
2. Induction property:
   At step number k, we assume that all previously colored vertices in L are exclusively colored #1 and all vertices in R colored #2 => The base case is correct
3. Induction step
   At step number k + 1, let the vertex in consideration be v_{k+1} and we need to color it. Due to the greedy algorithm, we know that the vertex v_{k+1} has at least one colored neighbor. As we know, the neighbors of a vertex only belong to a different group than the current vertex. According to the induction property, all vertices are colored uniformly

⇨ All neighbors of vertex v_{k+1} are colored with one color, so it means the vertex v_{k+1} only needs to use the other alternating color. As we know, this alternating color is the same as the one used for the neighbors of neighbors of vertex v_{k+1}, so this implies that vertex v_{k+1} is colored the same as its current group it belongs to.
At stage k + 1, the induction property still holds, concluding the proof

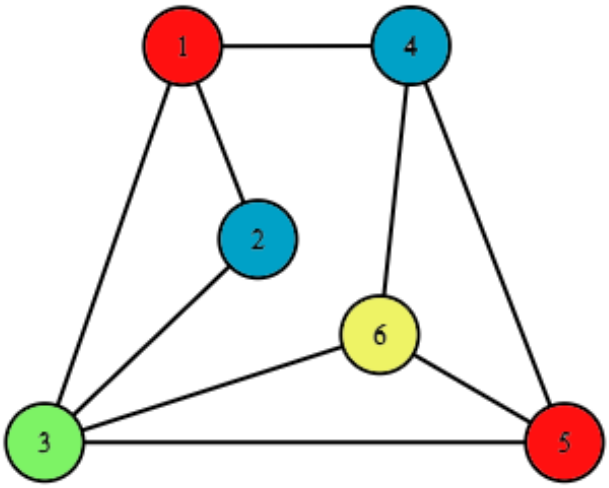
Therefore, this algorithm is optimum for bipartite graph

(b) (2p.) Show that the algorithm does not necessarily output an optimum coloring for general graphs.

In this graph, if we colors by this order with the first naive greedy algorithm, we will only need 3 colors, which is the chromatic number of this graph



However, if we use the greedy algorithm for the Bipartite, the graph can be colored as follows:



First, all nodes color-degree are 0. We color the first vertex as red. The color degree of vertices 2, 3, 4 are updated as 1. We randomly choose vertex 2 and color it blue. Now color degree of vertex 3 is 2, which is highest, so we proceed to color it as green and update the color degree of vertex 5 and 6 as 1. Now all vertices 4,5,6 have equal color degree of 1, so we randomly pick 4 and color it as blue. Now the color degree of 5 and 6 are 2. We randomly pick vertex 5 and colors it as red. Finally, due to unfortunate coloring configuration, the last vertex 6 has to resort to a fourth color which is yellow

⇨ Greedy algorithm for the bipartite graph is not optimum for general graphs

(c) (1p.) Analyze the runtime of the algorithm.

Consider a graph $G = (V, E)$, where $|V| = n$, and $|E| = m$. Since we need to visit each vertex once to visit it, and in each visit, we need to update the color degree. This updating requires a lower bound of O(m). In total, the running time of the bipartite coloring algorithm is O(n + m)