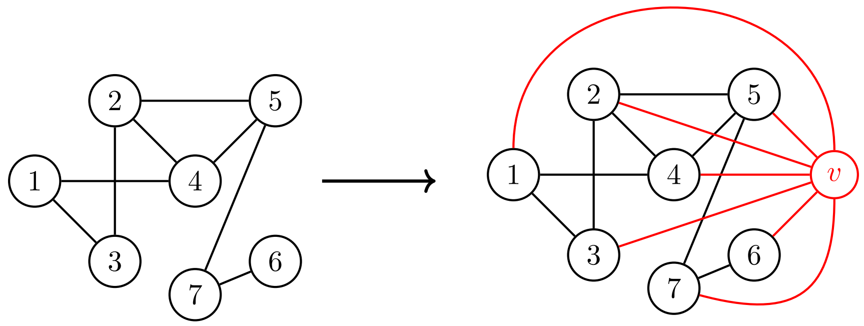
1 k-COL vs (k+1)-COL

<u>Statement:</u> *For every integer $k \geq 1$, give a direct proof (one that does not use any other results like Cook-Levin theorem) that k-COL $\leq^p$ $(k+1)$-COL*

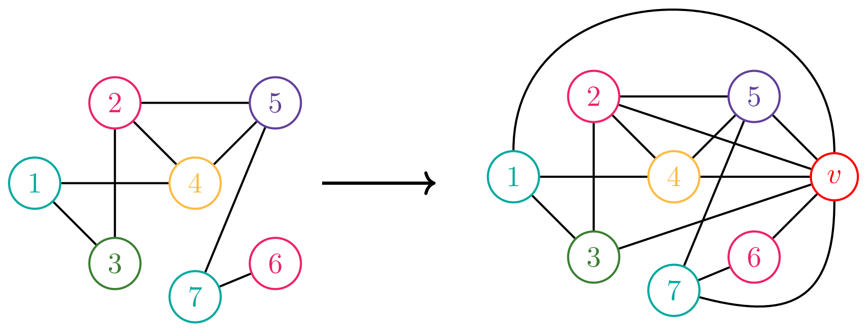
To show that k-COL is polynomial time reductible to the $(k+1)$-coloring problem, we need to find a polynomial time algorithm that takes an instance of k-COL as input and outputs an instance of $(k+1)$-COL. A valid k-coloring of one of theses instances has to be valid if and only if the other instance is valid.

Let G, a graph with n vertices, be an instance of k-COL. We construct an instance of $(k+1)$-COL. To do so, we add a new vertex v to the graph and we connect it to all vertices of G. Let G' be the resulting graph, with $n+1$ vertices.



The example above illustrates the construction of a graph by this algorithm. This construction takes $O(n)$ time because we have to connect the new vertex to each vertex of G, thus we have to travel along the entire list of vertices of G, which takes a time n

Let's now assume that the graph G has a valid k-coloring. We then color the vertex v with a new color which is not used in the k-coloring of G. We can claim that v has no neighbours with the same color since v is connected to every vertex of G, and has no color used in G. Knowing that G has a valid k-coloring, we can conclude that G' has a valid $(k+1)$-coloring.



Homework 1
Exercises around the k-coloring problem

Computational Complexity (CC-MIRI)

Duc DANG VU

Teacher: Albert ATSERIAS

UPC-FIB 2023

In the example above, v has no neighbours of the same color, thus the second graph has a valid $(k+1)$-coloring.

Let's now assume that G' has a valid $(k+1)$-coloring. We can remove v and its edges from G', and obtain a new graph G with a valid k-coloring. This follows because no two adjacent vertices in G' have the same color, and v has a unique color in G' (v is connected to every other vertex so it must have a unique color).

The construction of G' in polynomial time and the fact that we showed the equivalence between the k-coloring of G and the $(k+1)$-coloring of G' prove that $\underline{\boldsymbol{k}\textbf{-COL} \leq^{\boldsymbol{p}} (\boldsymbol{k}+\mathbf{1})\textbf{-COL}}$.

2 3-COL

Statement: *Look up a proof that 3-COL is NP-complete in the literature and reproduce it here in your own words. State your reference. For full credit, try to find the oldest reference for this.*

We will take as a reference the proof provided by Larry Stockmayer in March 1973. This paper, named *Planar 3-colorability is polynomial complete*, is based on Karp's work on graphs. Here is the URL of this paper: `https://dl.acm.org/doi/pdf/10.1145/1008293.1008294`.

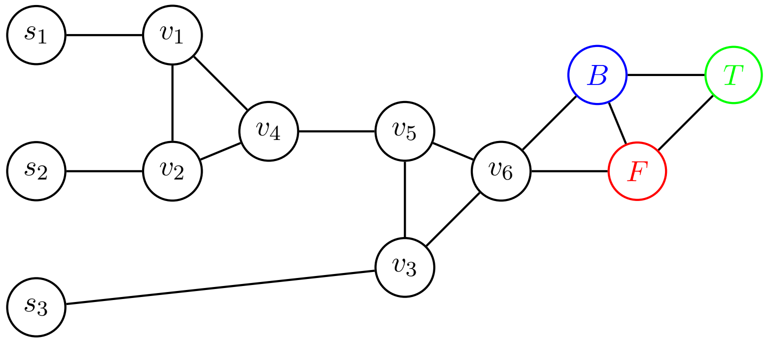
To show that 3-COL is NP-complete, we first need to show that it is NP. This is the case, because in order to verify that a graph $G = (V, E)$ with n vertices satisfies 3-COL, we have to go through the list of all vertices and check wether or not their neighbors are colored with a different color than the vertex studied. This check takes $O(n^2)$ time, thus 3-COL is NP.

Now we need to show that 3-COL is NP-hard. To do so, we have to give a polynomial-time reduction from 3-SAT to 3-COL. In other words, given an instance Φ of 3-SAT, we need to construct a graph G that satisfies 3-COL if and only if Φ is satisfiable.

Let Φ be a 3-SAT instance (a 3-CNF formula), with C_1, C_2, ... , C_m the clauses of Φ. Let x_1, x_2, ... , x_n be the variables of Φ.
The first step to build the graph G is to contruct a triangle where we assign values T, F and B to each vertex. These values correspond to "True", "False" and "Base". They correspond to the three colors of the graph. Then, for each variable x_i, we create a triangle (B, u_i, $\bar{u_i}$). These triangles will represent each value of x_i and its negation $\bar{x_i}$ (True for one and False for the other one). We see that if u_i is colored True, then $\bar{u_i}$ has to be colored False and vice-versa, which is what we are looking for. This first step should look like this:

These two colored structures illustrate the two properties: they have a valid 3-coloring and the output v_6 correspond to the truth assignment of a claus with three literals.

The final step of the construction of G consists in linking all structures S (there are as many as clauses) to the vertices labelled B and F of the first triangle:



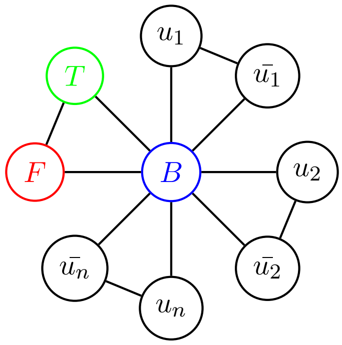
We will have m structures S linked to the base vertex. For each structure, we will link the vertices u_i (defined above) corresponding to each literals of one claus as entries of these structures.

Now that we have constructed the graph G, we now have to prove that the 3-SAT instance Φ is satisfiable if and only if the graph G is 3-colorable.

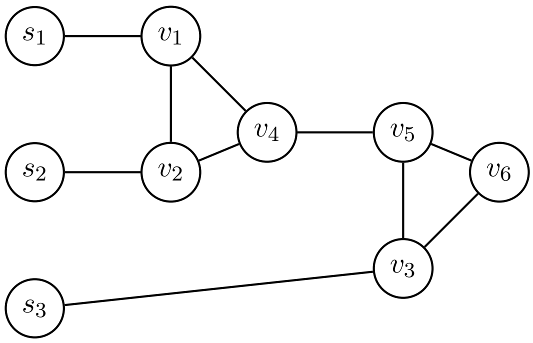
Let's assume that Φ is satisfiable. Let $(y_1, y_2, y_3, \dots , y_n)$ be the satisfying assignment. If y_i is True, then we color the corresponding vertex u_i with T and the vertex $\bar{u}_i$ with F. If y_i is False, then we will do the opposite. This coloring is valid knowing that u_i and $\bar{u}_i$ are adjacent to the Base vertex. Moreover, all output vertices of each structures (the vertex v_6) can be colored as T. Indeed, Φ is satisfiable, that means that for all structures S, at least one entry vertex s_i is colored T, thus the output vertex v_6 can be colored T by the second property. Knowing that each ouput vertex is connected to a B vertex and a F vertex, we can claim that G has a valid 3-coloring.

Let's now assume that G is 3-colorable. We construct the satisfying assignment of Φ by setting each literal x_i to True if the corresponding vertex u_i is colored with T, and vice-versa. We suppose that Φ is not satisfiable. That means that at least one claus has to be False, thus all literals of such a claus have to be False. Therefore, the structure S associated to such a claus has an output vertex v_6 colored with F, by the first property and knowing that all three entries are colored with F. Thus, a vertex colored with F would be adjacent to another vertex colored with F by construction: this is impossible because G is 3-colorable. Therefore, Φ is satisfiable.

The construction of the graph G takes a polynomial time, and we just gave a reduction from



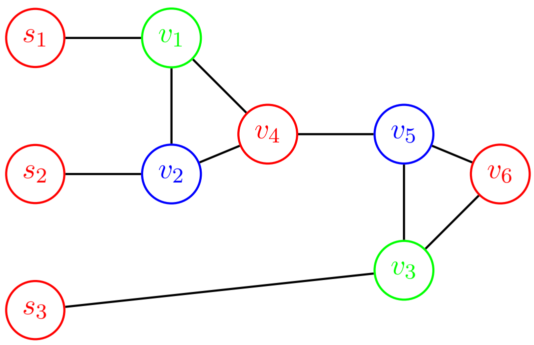
The second step consists in building a structure that will correspond to each clause of Φ. In order to do so, Larry Stockmayer constructed the following structure, that we will call S for the rest of the proof:

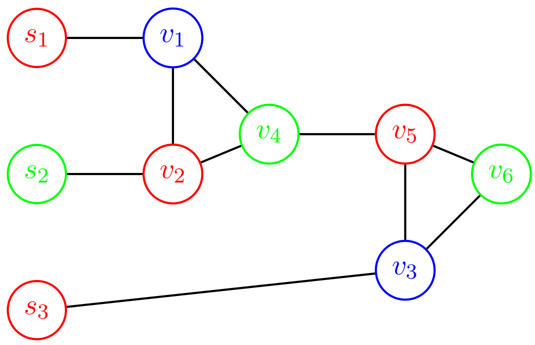


This structure S contains six vertices denoted by v_i and three entries s_i, which correspond to the three literals of a claus. In order to have a valid 3-coloring, S has to verify these 2 properties:

- If s_1, s_2 and s_3 have the same color c, then v_6 has to be colored with the color c. In other words, if the three clauses are F (respectively T), then v_6 has to be F (respectively T). This is what we expect from a claus in a 3-CNF formula.

- If s_1, s_2 and s_3 have colors where at least one of them is different of the two others, then there exist a valid 3-coloring of S where v_6 is colored with T. In other words, if at least one literal is assigned with T, then we can assign v_6 with T, which is what we expect from a claus in a 3-CNF formula.





3-SAT. We can now claim that 3-COL is NP-hard. Knowing that 3-COL is also NP, we can conclude that **3-COL is NP-complete.**

3 Self-reducibility

Statement: Show that, for every integer $k \geq 1$, if the decision version k-COL of the k-coloring problem can be solved in polynomial time, then its search version SEARCH-k-COL can also be solved in polynomial time, which means that there is a polynomial time algorithm that given a graph G outputs a valid k-coloring of it in the form of a list of pairs `(vertex, color)`. Clearly state the main idea of your algorithm at the beginning of your answer.

The idea to solve this problem is to explore each vertex v of a given graph G and to assign to each one a color c not used by one of its neighbours. Then, we check with the decision algorithm if G can have a valid k-coloring, knowing that the vertex v is colored with color c. If this is not the case, we color it with another color not used by one of its neighbours. By repeating this process with all vertices, we will find a valid k-coloring of the graph.

But how can we assign a color to each vertex so that the decision algorithm can understand which color we decide to assign at each step? The idea is to construct a new graph G' by adding to G k vertices, which will correspond to the k colors of k-COL. We will denote this new graph C (as "colors") such as: $C = (V_C, E_C) = (\{c_1, c_2, \ldots, c_k\}, E_C)$. We easily see that if we connect a vertex v to every c_i except one named c_j, then v has to be colored j. This trick will allow us to check with the decision algorithm if a given coloring of a vertex can lead to a valid coloring of the graph.
Note that in order for this idea to work, we have to make the subgraph C complete. In other words, we have to make every pair of vertices in V_C connected by an edge. This ensure that every c_i must have a different color. Otherwise, every c_i could have the same color, which is not what we want.

Now that we explained this idea, we now construct an algorithm to solve SEARCH-k-COL using the decison algorithm k-COL:
Given a graph G and an integer $k \geq 1$:

1. Run k-COL on G. If it outputs `NO`, output "`The graph cannot be k-colored`"
2. If it outputs `YES`, add a complete graph C to G with k vertices, denoted c_1, c_2, ... , c_k. The resulting graph is $G' = (V_{G'}, E_{G'})$.
3. Initialize a list `coloring` that will contain the valid k-coloring of G and a list `colored` that will contain the vertices of G already colored.