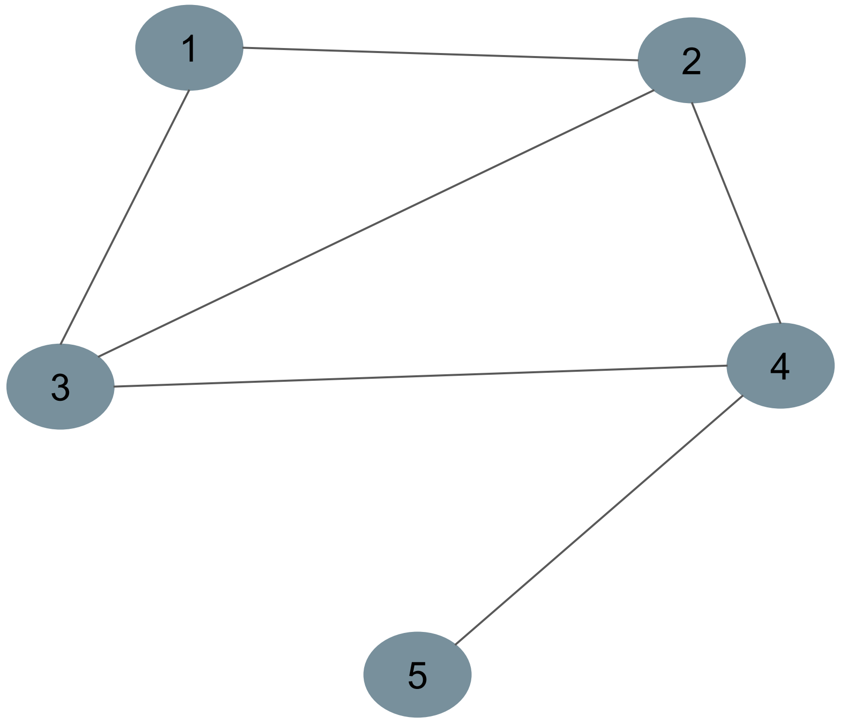
Let’s analyze the code by using “instance1.col”

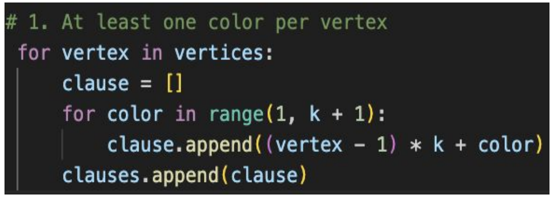


```
c FILE: instance1.col
c DESCRIPTION: A simple graph
p edge 5 6
e 1 2
e 1 3
e 2 3
e 2 4
e 3 4
e 4 5
```

1. dimacs_to_graph function read the file and creates:

```
edges = [(1, 2), (1, 3), (2, 3), (2, 4), (3, 4), (4, 5)]
vertices = [1, 2, 3, 4, 5]
```

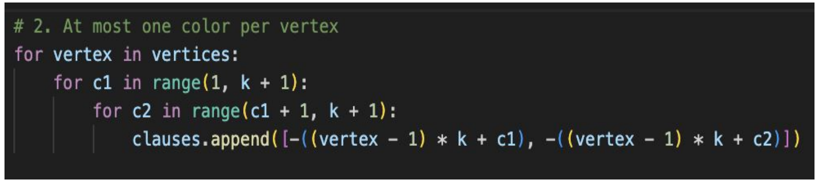
2. `create_CNF(vertices, edges, k):` Function creates clauses for SAT solver. There are 3 for loop in this function. Let’s assume k=2



```
# 1. At least one color per vertex
 for vertex in vertices:
    clause = []
    for color in range(1, k + 1):
        clause.append((vertex - 1) * k + color)
    clauses.append(clause)
```

For each vertex, the inner loop iterates over the range of colors `[1, 2]` for `k=2`. The resulting clause for each vertex represents that the vertex must be assigned at least one color. Let's compute the clauses:

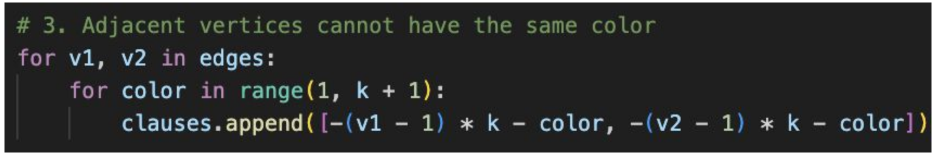
- For vertex `1`, the clause will be `[1, 2]`.
- For vertex `2`, the clause will be `[3, 4]`.
- For vertex `3`, the clause will be `[5, 6]`.
- For vertex `4`, the clause will be `[7, 8]`.
- For vertex `5`, the clause will be `[9, 10]`.



```
# 2. At most one color per vertex
for vertex in vertices:
    for c1 in range(1, k + 1):
        for c2 in range(c1 + 1, k + 1):
            clauses.append([-((vertex - 1) * k + c1), -((vertex - 1) * k + c2)])
```

For each vertex, the double for loop generates clauses to ensure that no vertex gets more than one color. Let's compute the clauses:

- For vertex 1, the clauses will be [-1, -2].
- For vertex 2, the clauses will be [-3, -4].
- For vertex 3, the clauses will be [-5, -6].
- For vertex 4, the clauses will be [-7, -8].
- For vertex 5, the clauses will be [-9, -10].



```python
# 3. Adjacent vertices cannot have the same color
for v1, v2 in edges:
    for color in range(1, k + 1):
        clauses.append([-(v1 - 1) * k - color, -(v2 - 1) * k - color])
```

For each edge, the for loop generates clauses to ensure that adjacent vertices cannot have the same color. Let's compute the clauses:

- For edge `(1, 2)`, the clauses will be `[-1, -3]`, `[-2, -4]`.
- For edge `(1, 3)`, the clauses will be `[-1, -5]`, `[-2, -6]`.
- For edge `(2, 3)`, the clauses will be `[-3, -5]`, `[-4, -6]`.
- For edge `(2, 4)`, the clauses will be `[-3, -7]`, `[-4, -8]`.
- For edge `(3, 4)`, the clauses will be `[-5, -7]`, `[-6, -8]`.
- For edge `(4, 5)`, the clauses will be `[-7, -9]`, `[-8, -10]`.

3. `solution = pycosat.solve(clauses)`

Then we check for solution for created clauses.

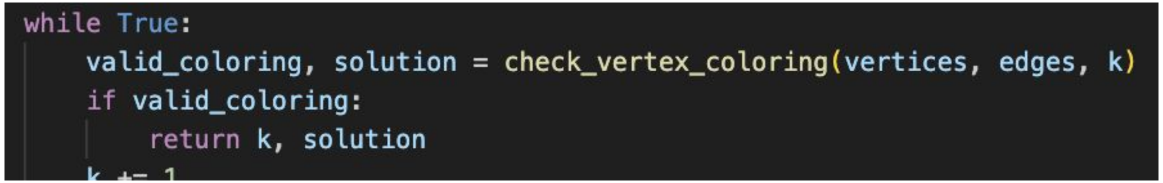
The function takes a list of clauses as input. Each clause is represented as a list of integers, where each integer represents a literal. Positive integers represent variables that are true, and negative integers represent variables that are false.

I use the `pycosat` library in this problem because it provides an efficient SAT solver implementation. Here are some benefits of using `pycosat` for this problem:

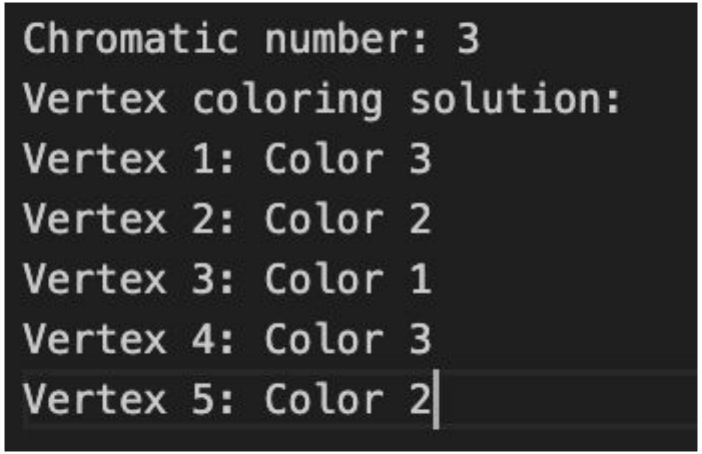
`pycosat` is a Python wrapper for the SAT solver, which is known for its efficiency. SAT (Boolean satisfiability) solvers are optimized for solving problems expressed in Conjunctive Normal Form (CNF), making them well-suited for problems like vertex coloring, which can be formulated as SAT problems.It is capable of handling large problem instances efficiently. For vertex coloring problems, as the size of the graph (number of vertices and edges) increases, `pycosat` can still provide solutions in reasonable time.It provides a simple interface for defining and solving SAT problems in Python. I

This problem Unsatisfiable with given k = 2. So let’s compute chromatic number function (Part 2)

This function starts k from 1 and increase one by one for each iteration to check whether the solution is exists or not



```
while True:
    valid_coloring, solution = check_vertex_coloring(vertices, edges, k)
    if valid_coloring:
        return k, solution
    k += 1
```



```
Chromatic number: 3
Vertex coloring solution:
Vertex 1: Color 3
Vertex 2: Color 2
Vertex 3: Color 1
Vertex 4: Color 3
Vertex 5: Color 2
```