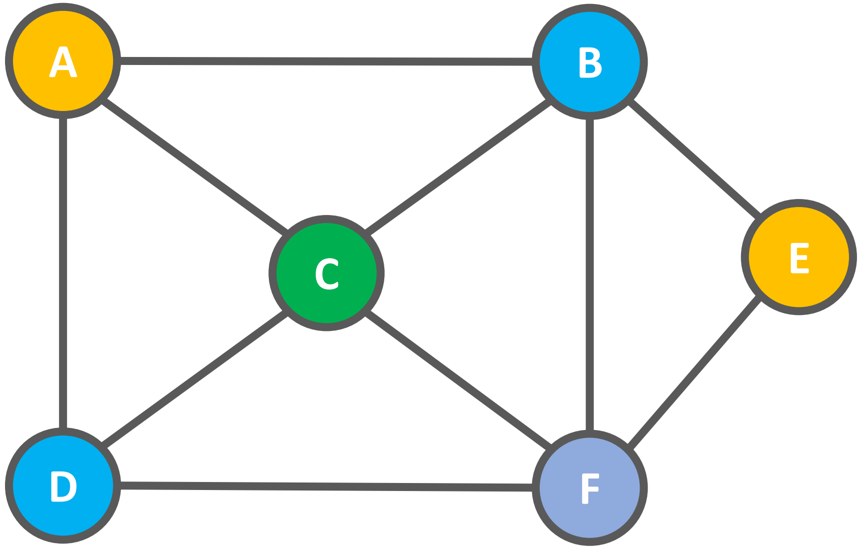
Coloring Problem

(Algorithmic Problems)

Coloring Problem

- the problem is that we have to color the vertices of a **G(V, E)** graph such that no two adjacent vertices share the same color
- we use integer representation for the given colors
- the smallest number of colors needed to color a graph **G(V, E)** is called its **chromatic number**
- there may be more than one solution - for example we can color a graph with **4** vertices in **12** ways with **3** colors
- there are several applications of this problem

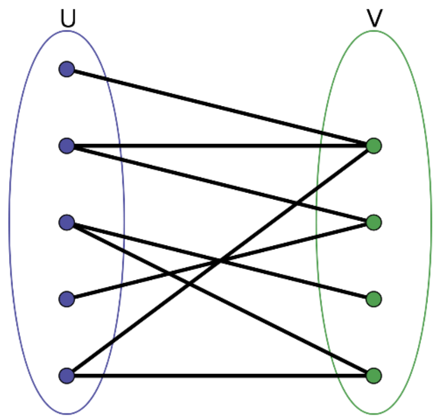
Coloring Problem



Coloring Problem

- the problem is that we have to color the vertices of a **G(V, E)** graph such that no two adjacent vertices share the same color
- it is again an **NP-complete** problem
- there is an exponential number of possible states
- if we have k colors and want to assign colors to vertices in a **G(V, E)** graph then the running time is **$O(k^V)$**
- so with brute-force approach we can achieve **exponential running** time complexity

Applications - Coloring Problem



1.) BIPARTITE GRAPHS

- determining if a graph can be colored with **2** colors is equivalent to determining whether or not the graph is bipartite
- this problem is computable in **O(N)** linear time using **breadth-first search**
- **bipartite graph**: a graph whose vertices can be divided into two disjoint sets **U** and **V** (**U** and **V** are independent sets) such that every edge connects a vertex in **U** to one in **V**

Applications - Coloring Problem

2.) SCHEDULING

- the aim is to make an **exam schedule** for a university
- we have different subjects and different students enrolled on every subject - many subjects would have common students
- **how do we schedule the exam so that no two exams with a common student are scheduled at the same time?**
- **how many minimum time slots are needed to schedule all exams?**

Applications - Coloring Problem

2.) SCHEDULING

- this problem can be represented as a **G(V, E)** graph where every vertex is a subject and an edge between two vertices means there is a common student
- so this is a **G(V, E)** graph **coloring problem** where minimum number of time slots is equal to the **k chromatic number** of the graph

Applications - Coloring Problem

3.) RADIO FREQUENCY ASSIGNMENT

- when frequencies are assigned to towers - frequencies assigned to all towers at the same location must be different because of the interference
- **how to assign frequencies with this constraint? what is the minimum number of frequencies needed?**
- this problem is also an instance of graph coloring problem where every tower represents a vertex
- an edge between two towers represents that they are in range of each other

Applications - Coloring Problem

4.) REGISTER ALLOCATION (COMPUTERS)

- **compiler optimization** relies heavily on the coloring problem
- register allocation is the process of assigning a large number of target program variables onto a small number of **CPU** registers

Applications - Coloring Problem

5.) MAP COLORING

- we want to construct a **map of countries** or states where adjacent countries or states can not be assigned the same color

- this is one of the widely known algorithmic problems

- four colors are enough to color any map – this is the so-called **four color theorem**

Algorithms - Coloring Problem

There are several **approaches** to solve this problem:

1.) greedy approach – it finds a solution but not necessarily the best one possible (it may use more colors)

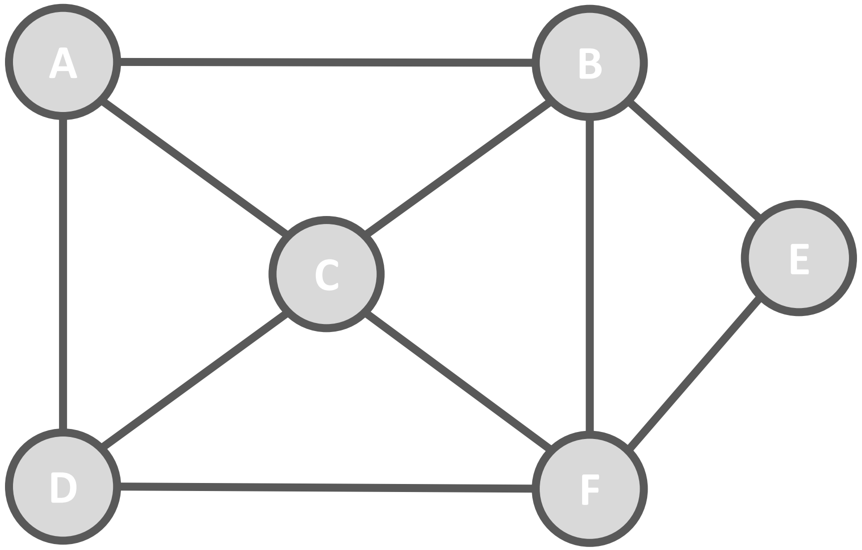
2.) backtracking – it can discard and reject multiple bad states within a single iteration (or recursive function call)

3.) Powell-Welsh algorithm - relies on sorting the nodes based on the degree (number of edges)

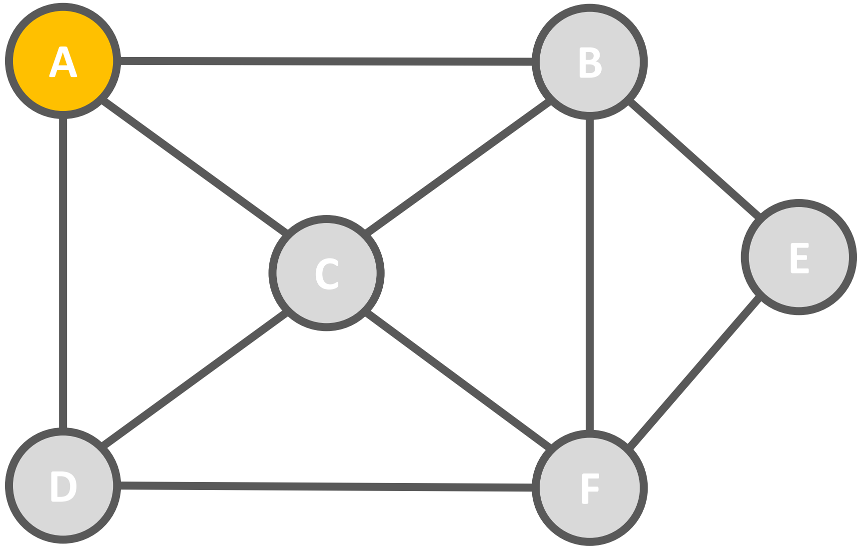
Coloring Problem

- we assign **colors** one by one to different vertices starting from the first vertex (for example with index **0**)
- before assigning a color: we check for safety by considering already assigned colors to the adjacent vertices
- if we find a color assignment which is feasible: we mark the color assignment as part of solution
- if we do not a find color due to clashes: we **BACKTRACK**

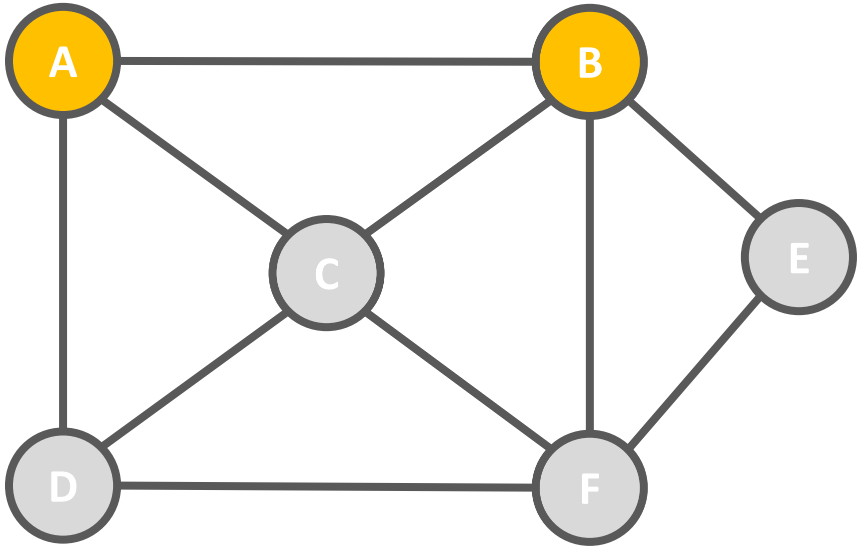
Coloring Problem



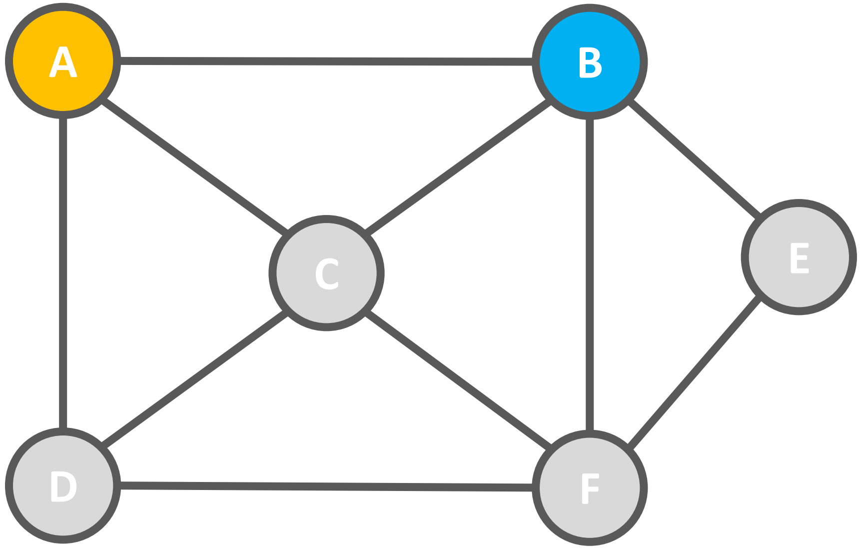
Coloring Problem



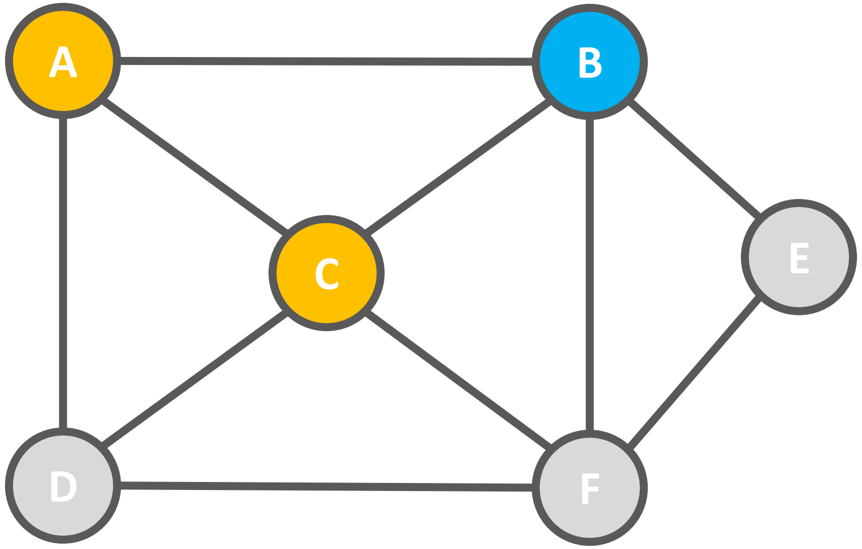
Coloring Problem



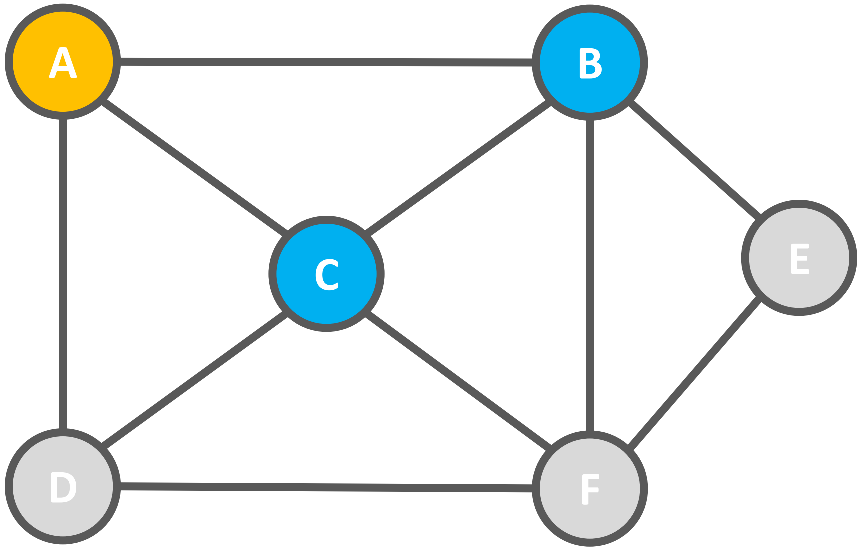
Coloring Problem



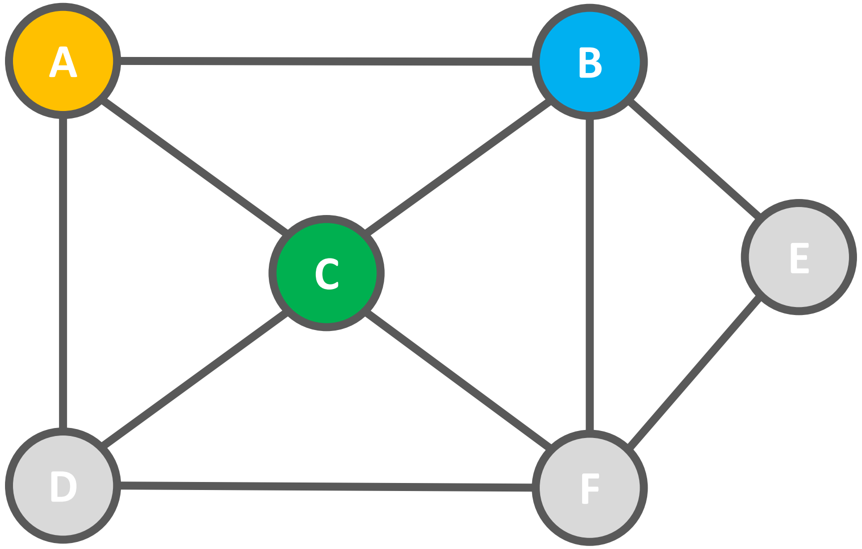
Coloring Problem



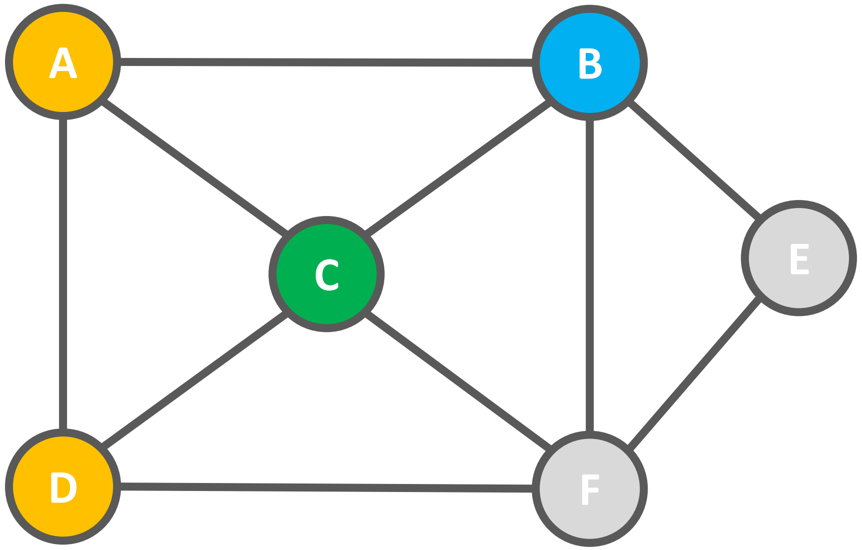
Coloring Problem



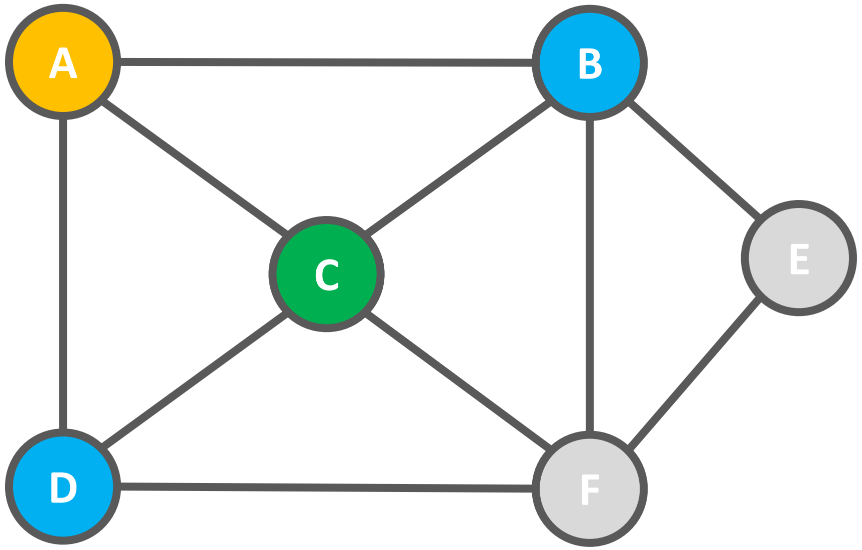
Coloring Problem



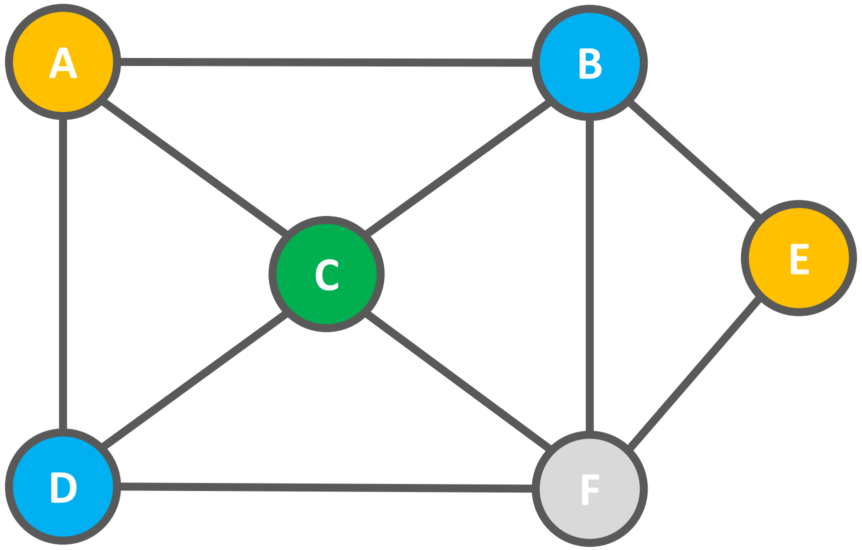
Coloring Problem



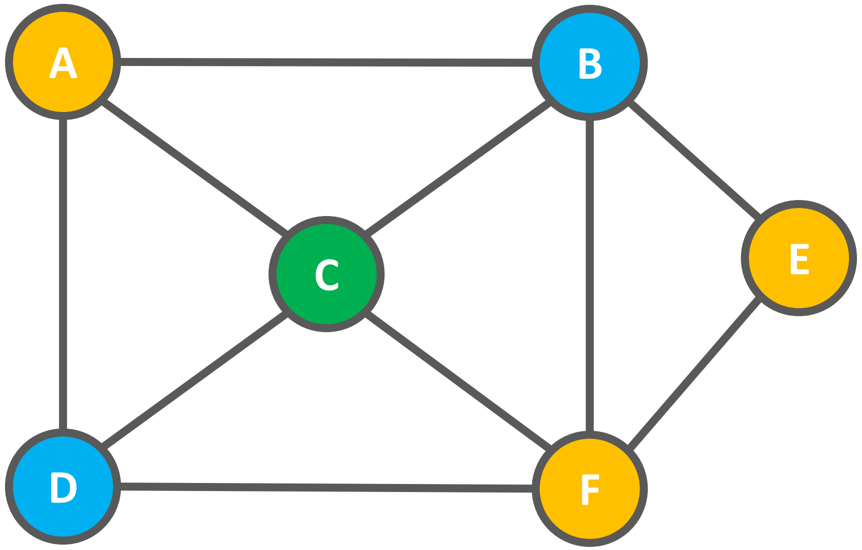
Coloring Problem



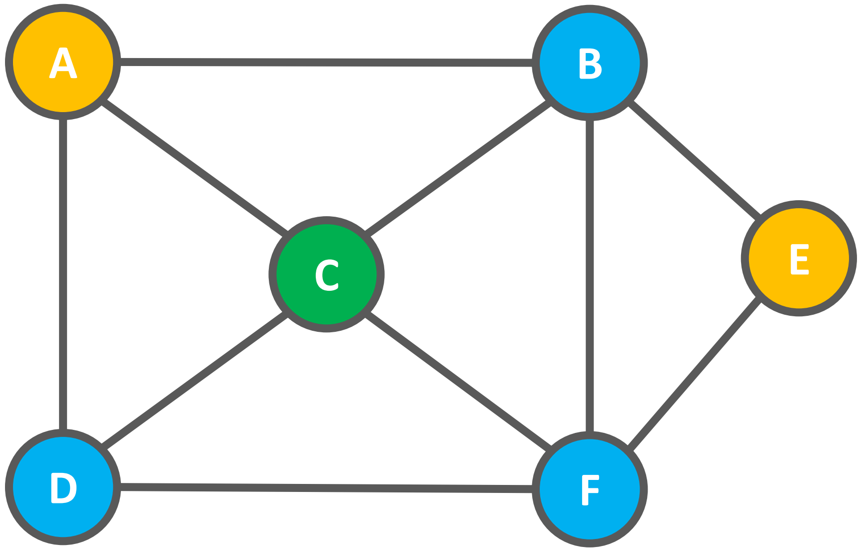
Coloring Problem



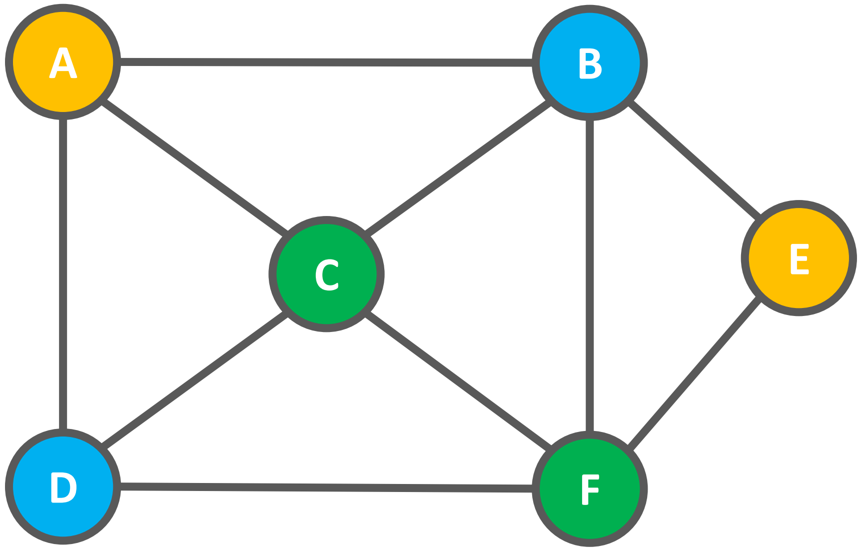
Coloring Problem



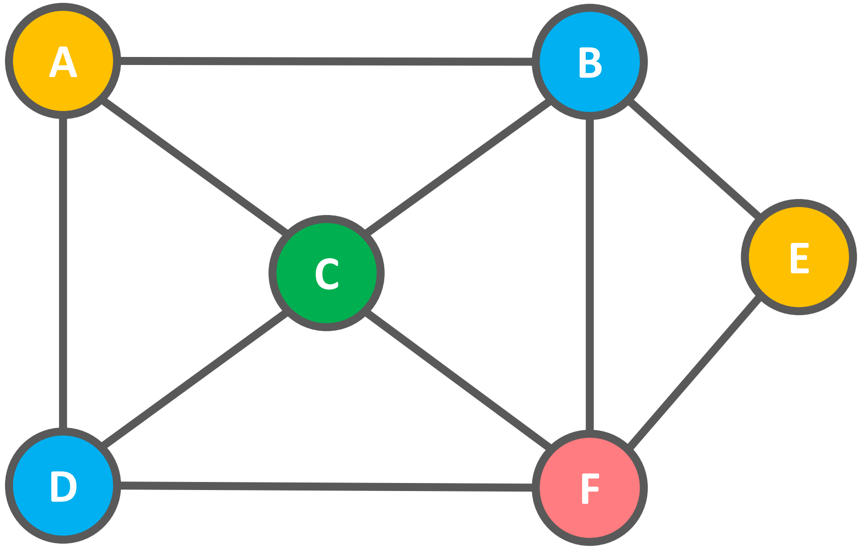
Coloring Problem



Coloring Problem



Coloring Problem

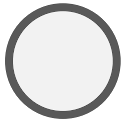


Coloring Problem

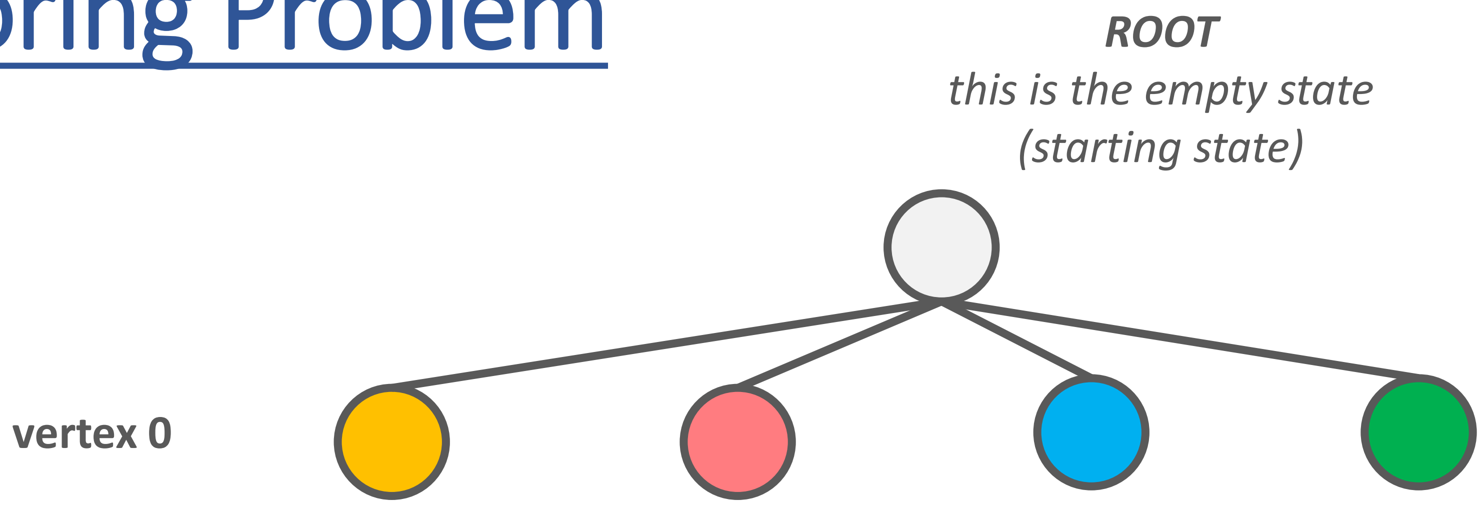
ROOT

this is the empty state

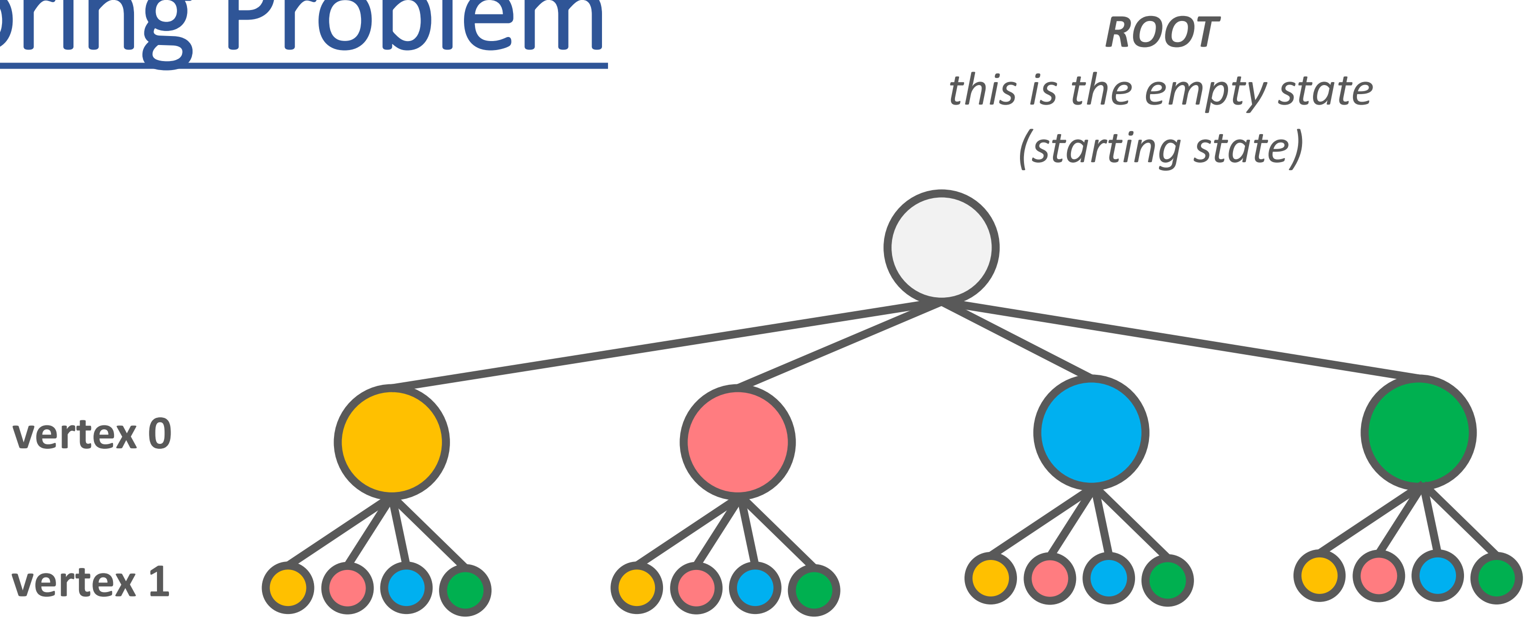
(starting state)



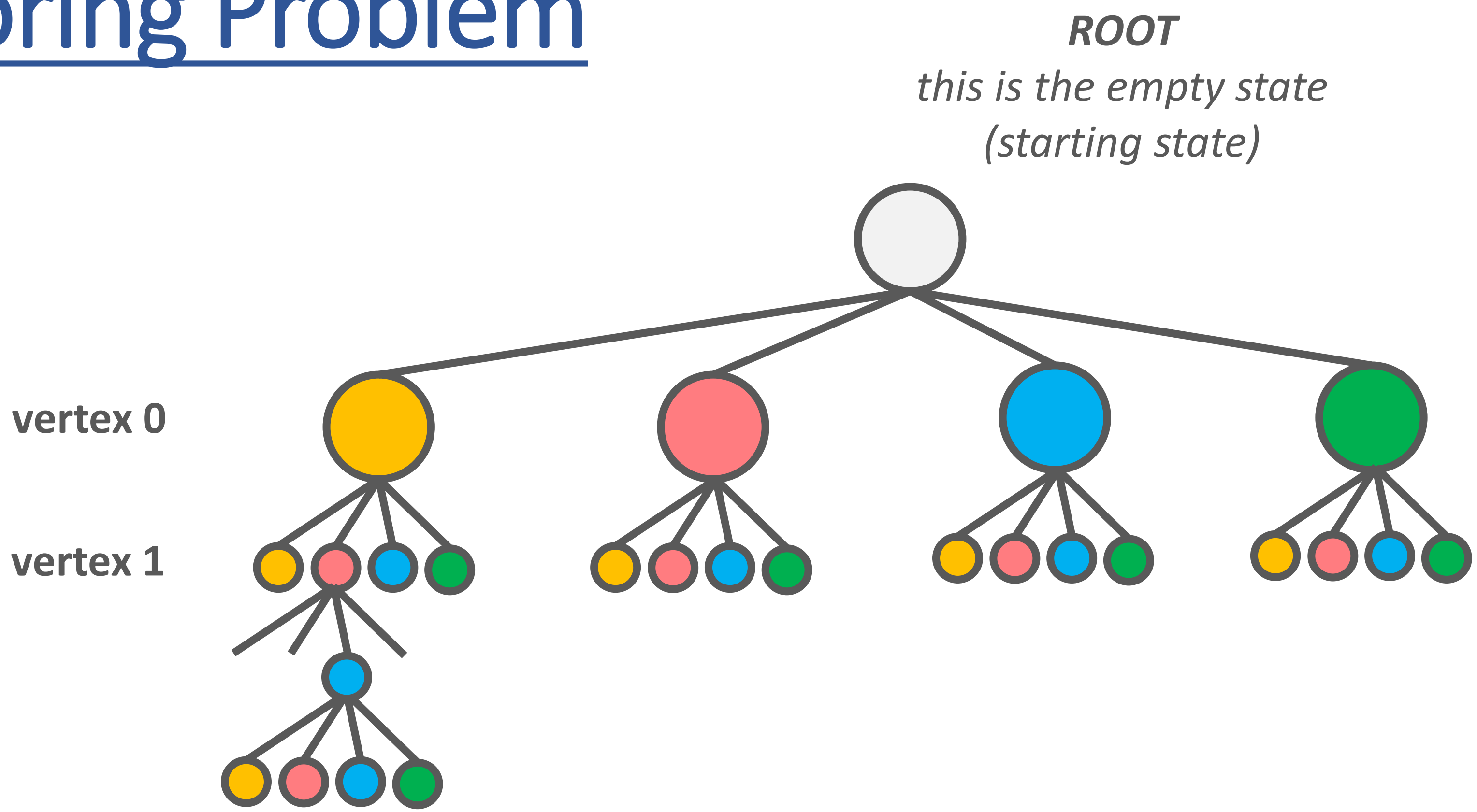
Coloring Problem



Coloring Problem



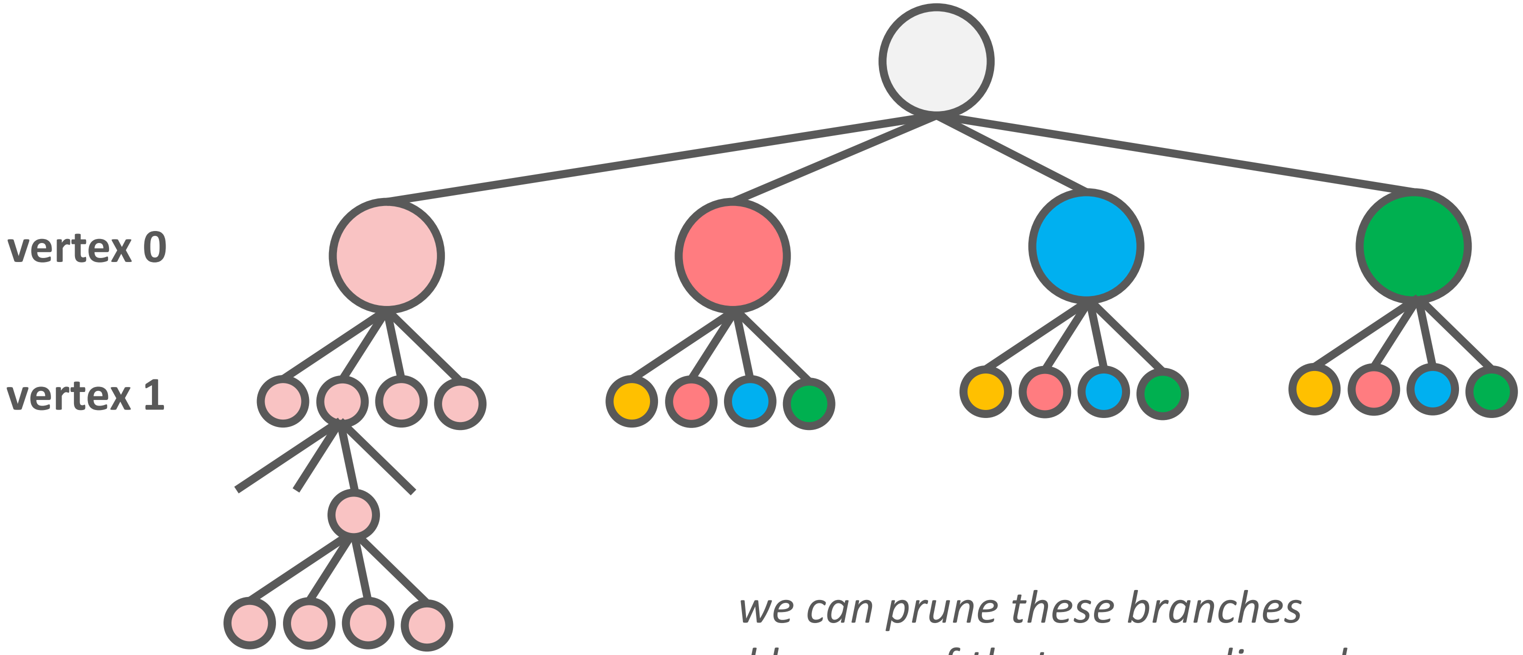
Coloring Problem



Coloring Problem

ROOT

this is the empty state
(starting state)



we can prune these branches
and beause of that we can discard
mutiple bad states (subtrees) in the same iteration
PRUNING BOOSTS THE RUNNING TIME OF BACKTRACKING