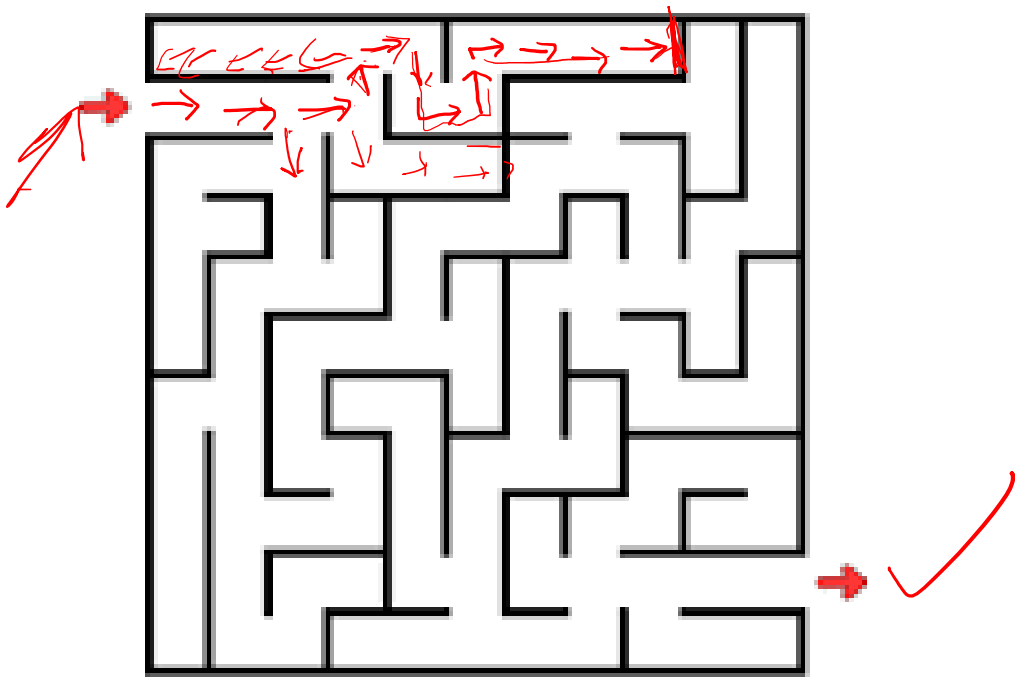
Backtracking Strategy

Backtracking

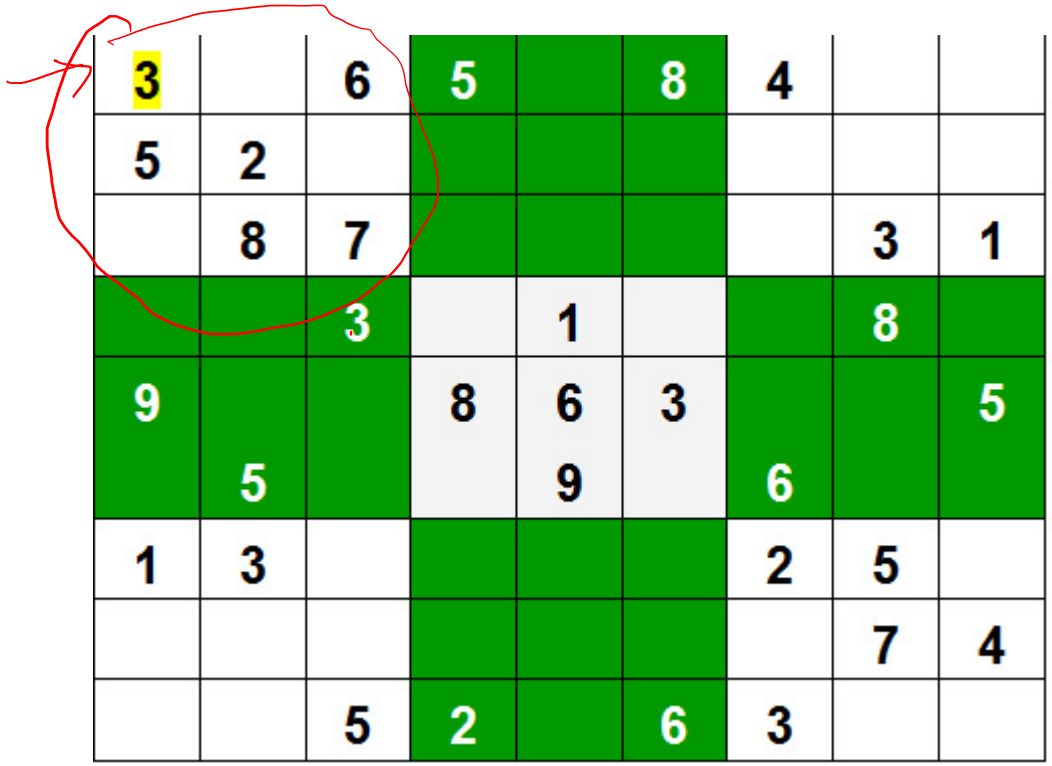
- considers searching every possible combination in order to solve a computational problem.
- A *refined brute force* technique for systematically searching for a solution.
- When a decision is executed, it leads to another set of decisions.
- When there is an impasse, one backtracks to previous decision and selects next possible decision.
- Common examples, playing mazes and Sudoku.

When deadend reached, backtrack to previous decision,
and take a new decision

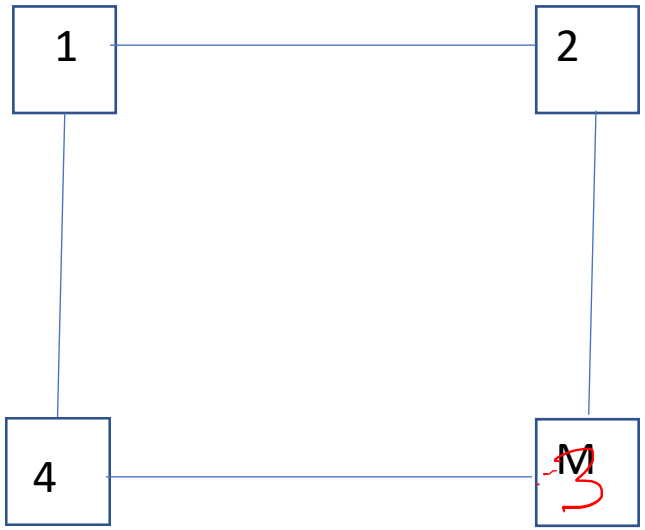


- Digits are filled one by one.
- When there is a clash, we cannot move to next position.
- we backtrack and choose a different digit at that position to move on.

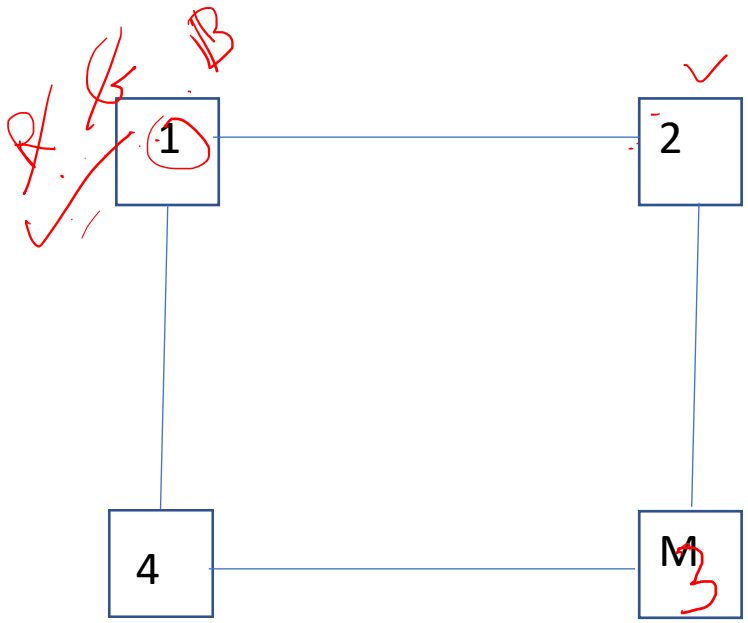
-



1. Graph Coloring


1
2
4
M

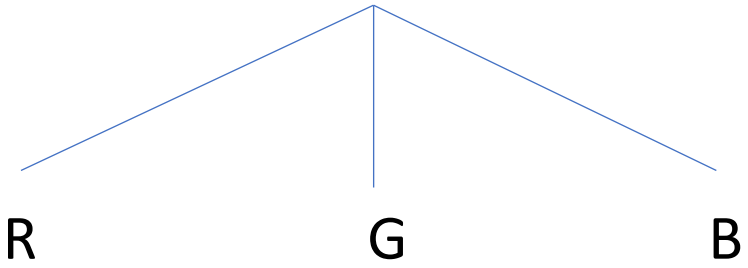
Let us say we want to color this graph, given M colors R,G, B.



In how many ways it can be done, with the condition that vertices on same edge will not have same color.

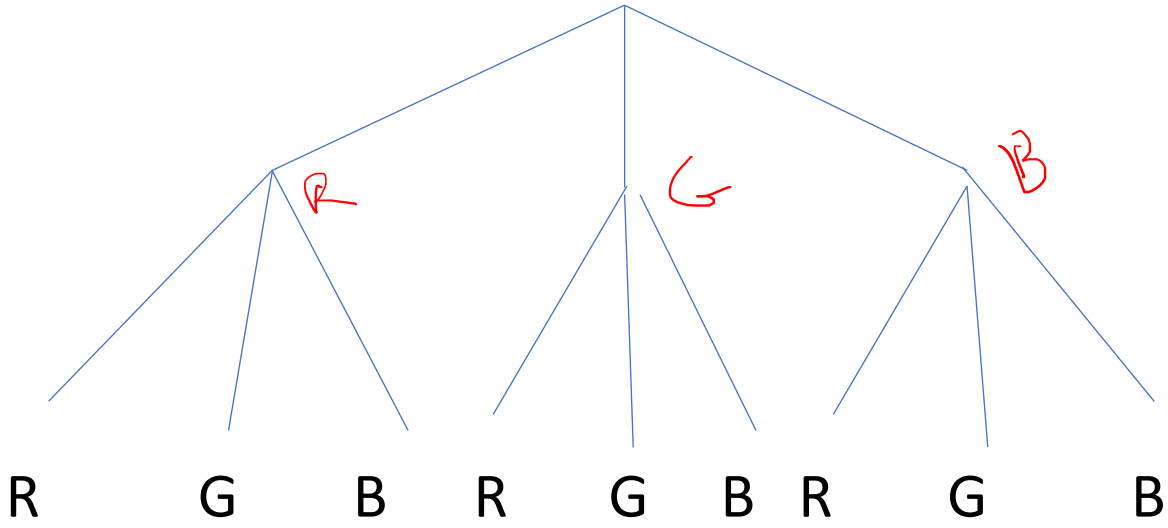
Let us first create a state-space without imposing any conditions

Vertex 1 can take any of the M colors.

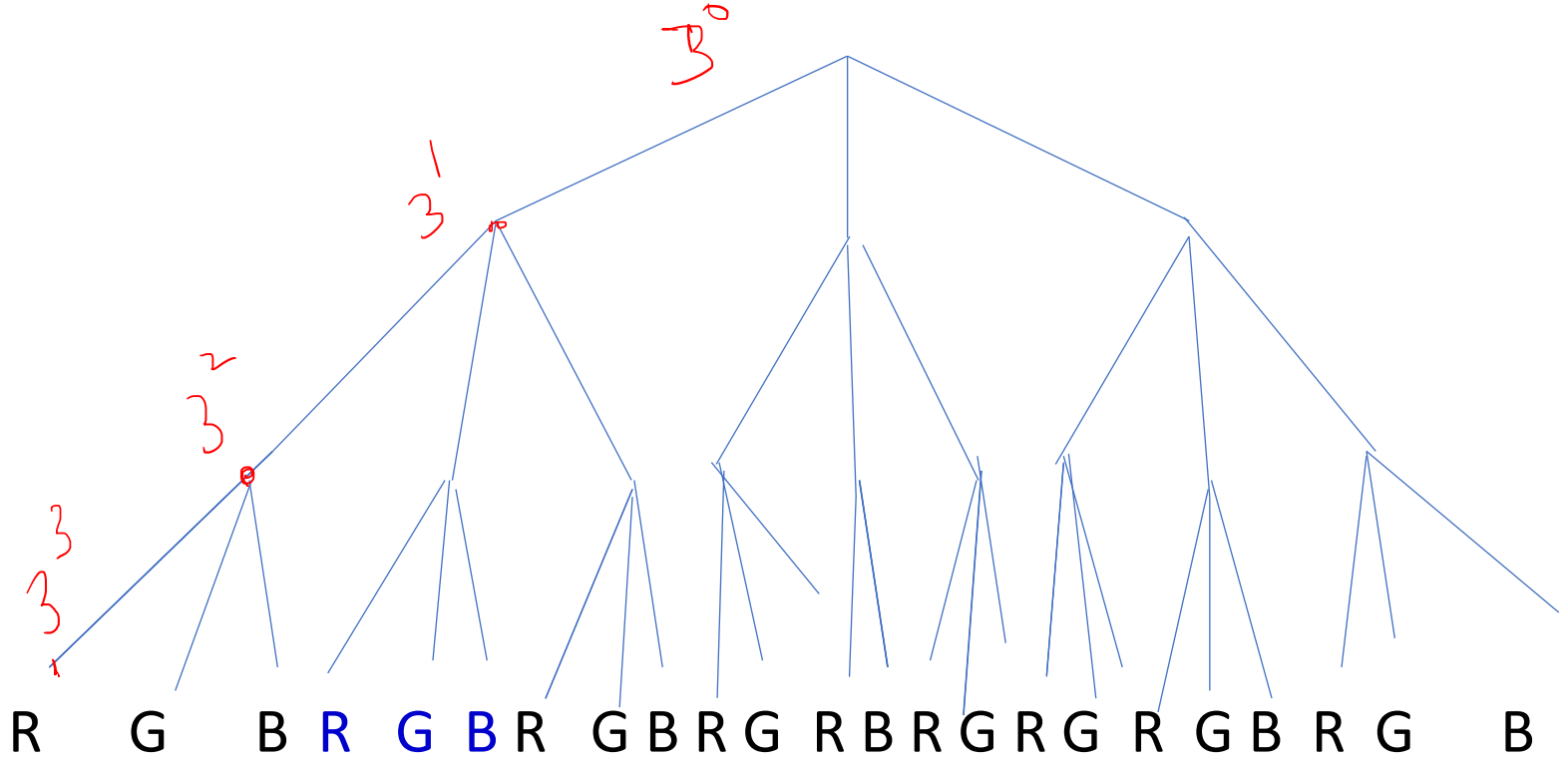


R

For each selection of R, G, B at vertex 1, Vertex 2 can also be given M different colors ,



For each selection of color at Vertex 2, M different colors can be assigned at vertex M

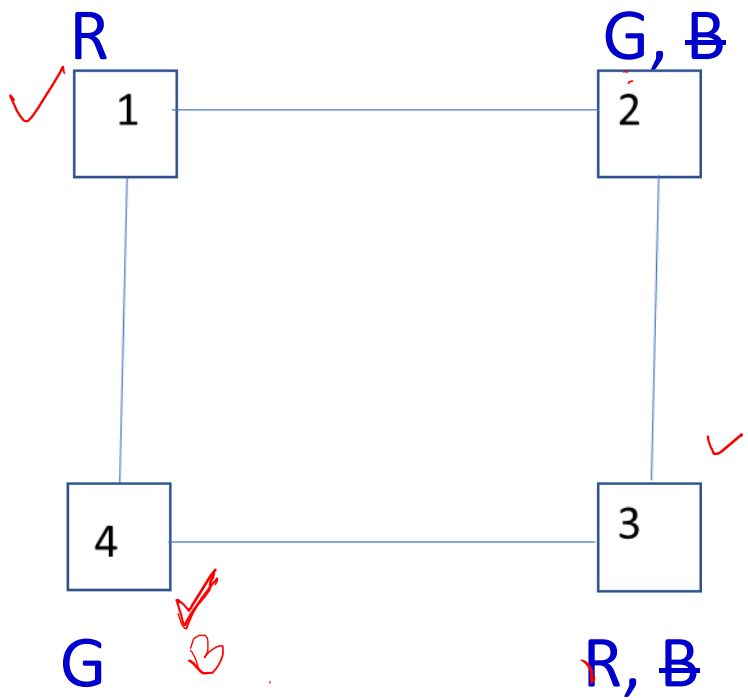


This process can be continued for vertex 4. Let us just compute how many assignment are possible:

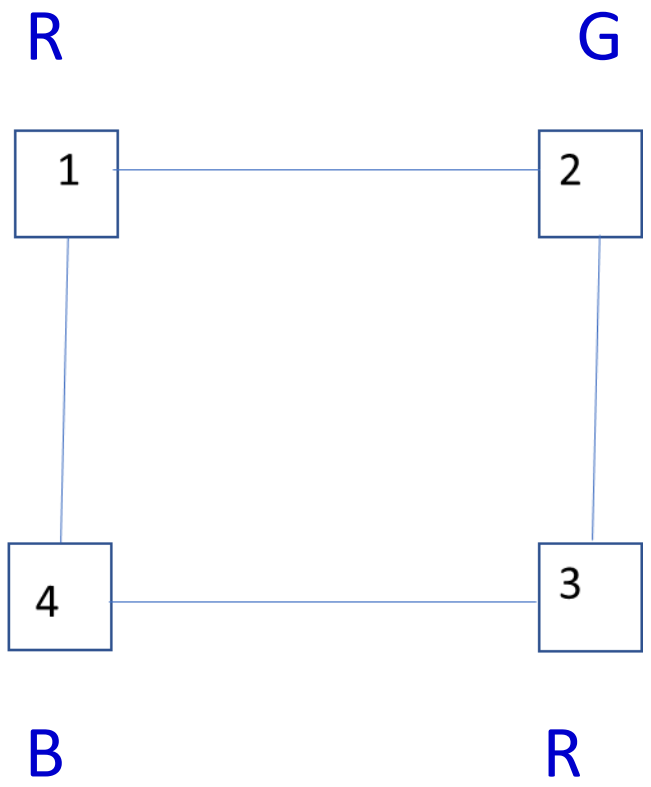
$M = 3$

$1 + M + M^2 + M^3 + M^4 = (M^{4+1}) / (M - 1)$, where M is no. of colors, and 4 is no. of vertices

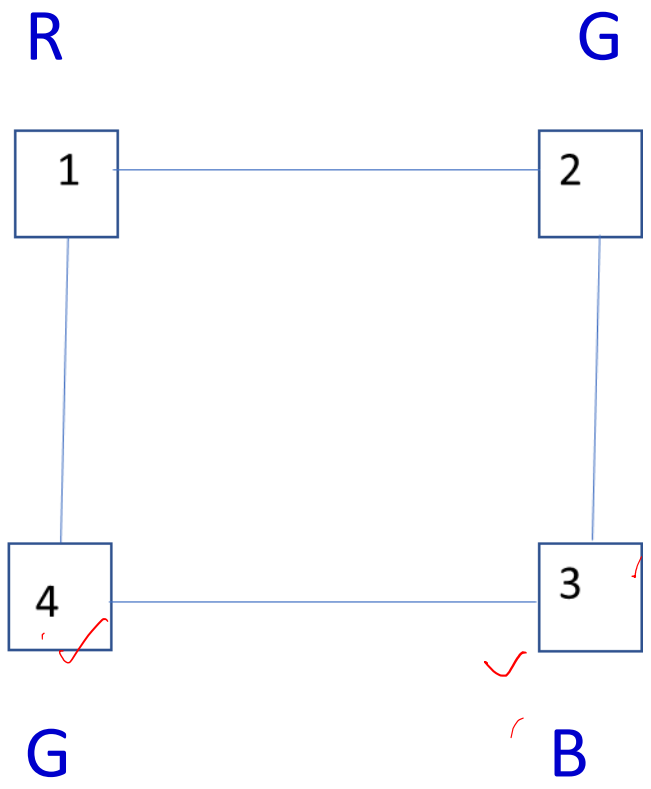
For n vertices and c colors $O(c^n)$ EXPONENTIAL PROBLEM



- Let us now use backtracking to color the vertices such that no two adjacent colors are same.
- This is something like DFS
- Start by R at 1
- 2 can have G or B
- Let us assign G to 2
- Vertex 3 can have R or B
- Let us assign R
- Vertex 4 can be assigned G
- One solution: R G R G



- R G R G
- To find another solution, Let us now backtrack from vertex 4
- Vertex 4 can also be assigned B
- R G R B is another solution



- Go back to vertex 3
- Can it get color B?
- Yes, R G B for vertices 1, 2, 3
- then for vertex 4, we can have G
- Next solution R G B G
- Another possible solutionR B R G
- Now number of possible combinations will be less than earlier, as many states get killed by bounding function (Constraint)

- Abdul Bari
  https://www.youtube.com/watch?v=052VkKhIaQ4

- One of the application of graph coloring is MAP COLORING.

50 States, 50 colors ???

Why Map Coloring Problem became important?

Printing map on paper required many cycles

We don’t want 50 cycles

How to solve the Map Coloring Problem?

Convert it to a graph coloring problem

Assign 1 vertex to each state

Color the Map of the United States of America

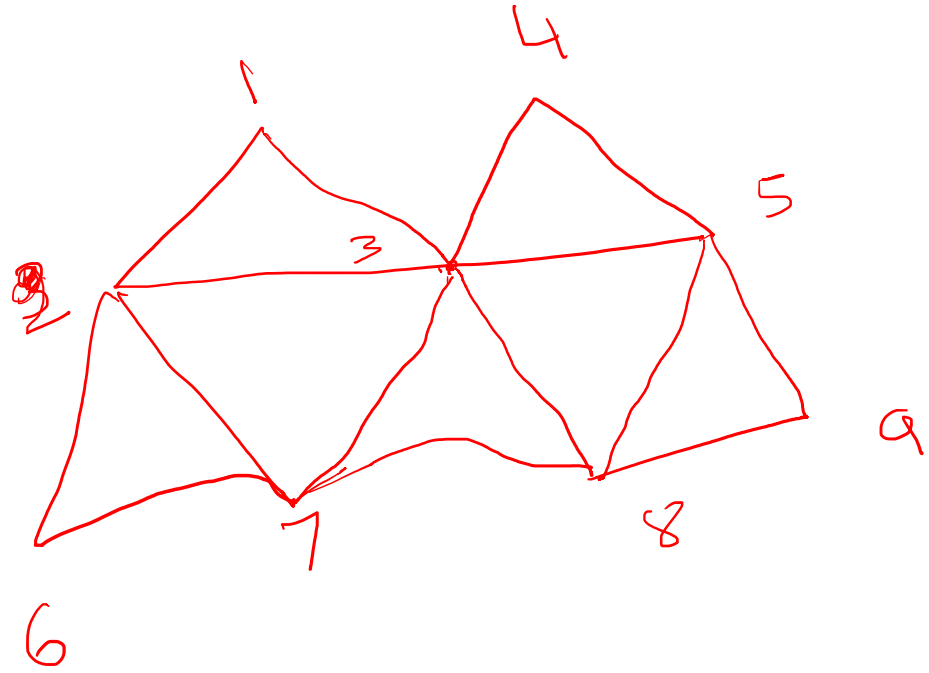
Form graph vertices

Name: _____ Date: _____

Color the Map of the United States of America

Draw the edges

Graph coloring

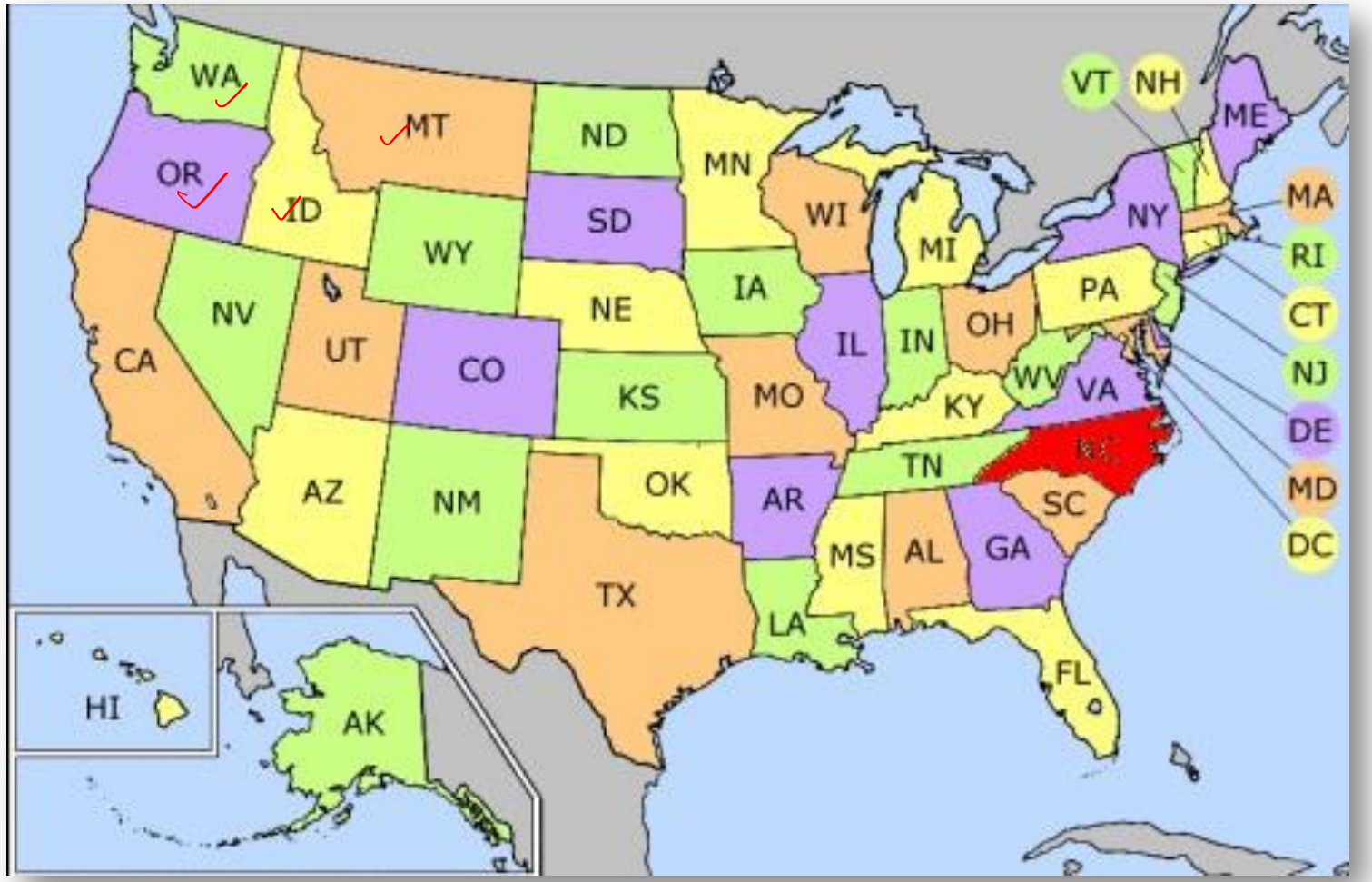


- Assign a different color

to each vertex

- 50 states, 50 colors
- Make the problem interesting
- How many minimum colors are needed to color the complete map/graph

US Map Coloring: with 5-Colors



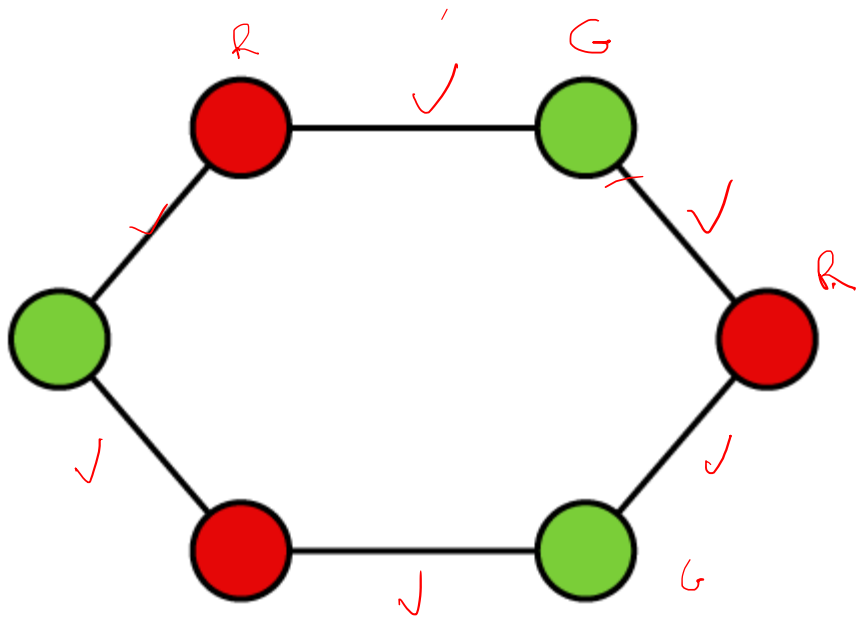
Graph Coloring Problem

- Procedure of assignment of colors to each vertex such that adjacent vertices do not get same color.
- Objective: to minimize number of colors while coloring a graph.
- Smallest number of colors required to color a graph G is called its chromatic number.

- **Cycle Graph:**
- A graph is a cycle graph, if it contains 'n' edges and 'n' vertices (n >= M), which form a cycle of length 'n'.
- **Chromatic number:**

1. The chromatic number in a cycle graph will be 2 if the number of vertices in that graph is even.
2. The chromatic number in a cycle graph will be M if the number of vertices in that graph is odd.

- Here the chromatic number is 2 (even vertices)



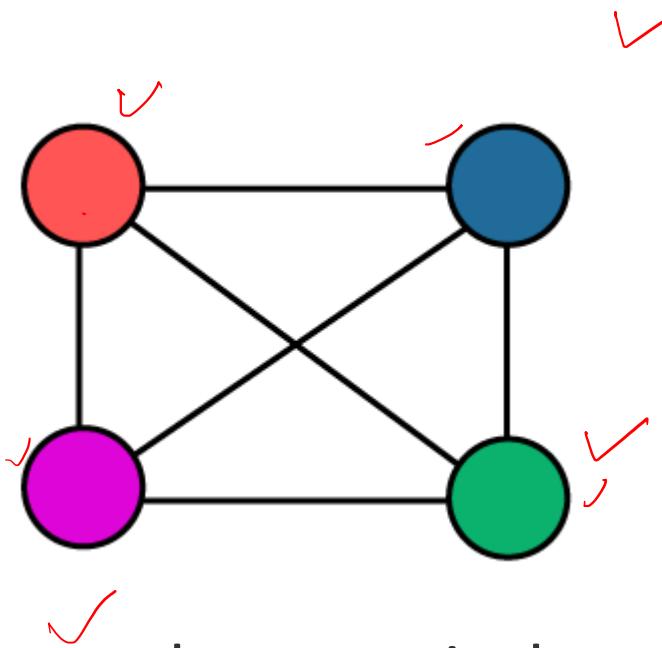
6 vertices

6 edges

- **Complete Graph**
- Every vertex is connected with every other vertex. Every vertex will be colored with a different color.

- **Chromatic Number of complete graph**
- In a complete graph, the chromatic number will be equal to the number of vertices in that graph.

- In this complete graph, the chromatic number will be 4



- Let us now look at another practical example of graph coloring

Meeting scheduling
can be viewed as
graph coloring problem

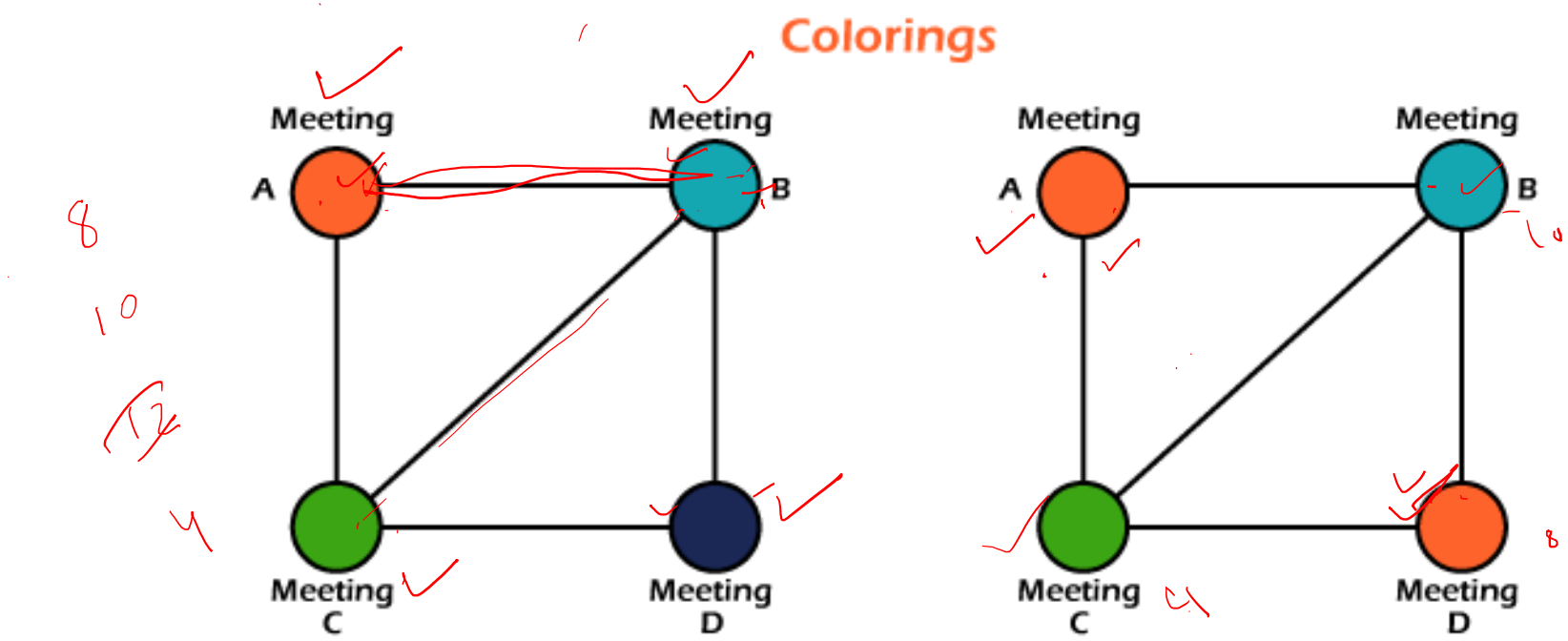
Scheduling Meetings

- Schedule meetings using as few time slots as possible.
- If a person needs to be present at two different meetings, then the manager needs to use different time schedules for those meetings.

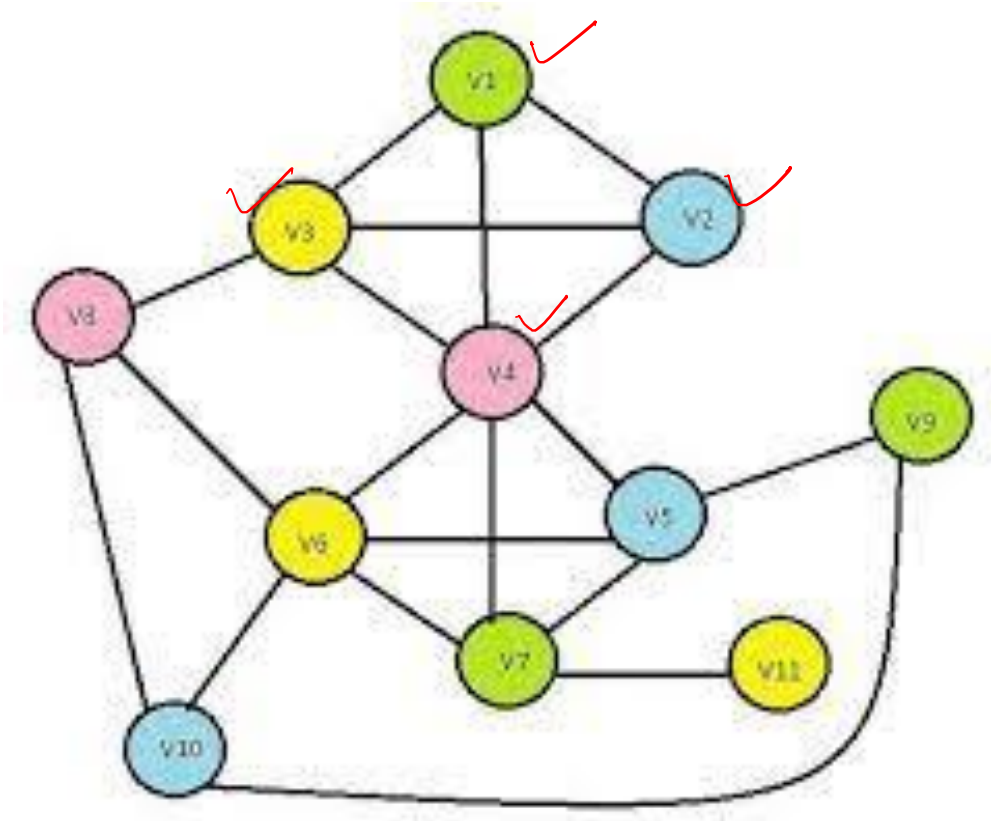
- Let a vertex denote a meeting.
- If a person has two meetings to attend, then the meetings will be connected with the help of an edge.

- Different colors used to represent different time slots.
- Vertices on an edge cannot be assigned same color. (To ensure that a person who needs to attend the two meetings, does not get the same time slot).

- 4 meetings do not need 4 time-slots, 3 will do. Why?

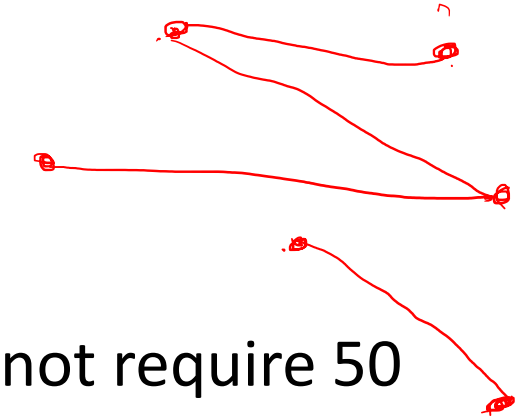


- 11 meetings can be conducted in 4 time-slots.



Scheduling Exams

Exam Scheduling as graph coloring



- If there are 50 exams to be conducted at an institute, it does not require 50 days to schedule them.
- There are many exams in which there are no common students.
- Such exams can be held in parallel (at same time)
- The exam scheduling problem can also be thought of as a graph coloring problem

Exam Scheduling example

We want to schedule exams for following CS courses with following course numbers:

1007, M1M7, M157, M20M, M261, 4115, 4118, 4156

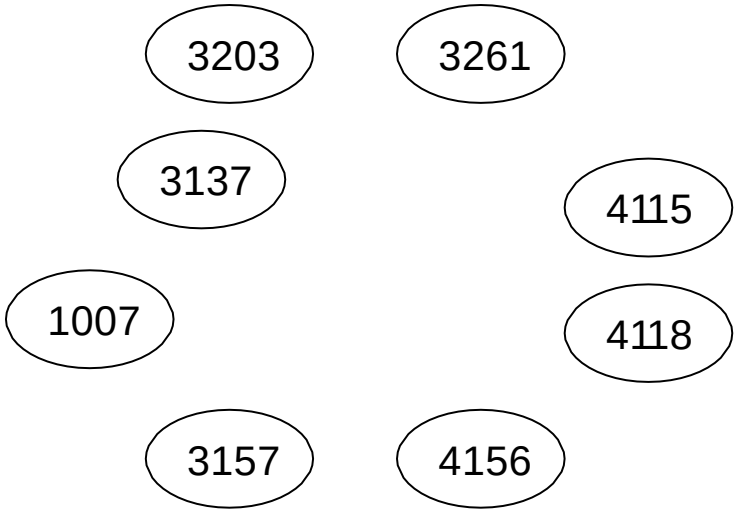
No students in common taking the following pairs of courses:

1007-M1M7, 1007-M157, 1007-4156, 1007-M20M
1007-M261, 1007-4115, 1007-4118,
M1M7-M261, M1M7-M157, M1M7-4156, M1M7-4118
M1M7-4115, M157-4156,
M261-4115 , M20M-M261, M20M-4115,

How many exam slots are necessary to schedule exams?

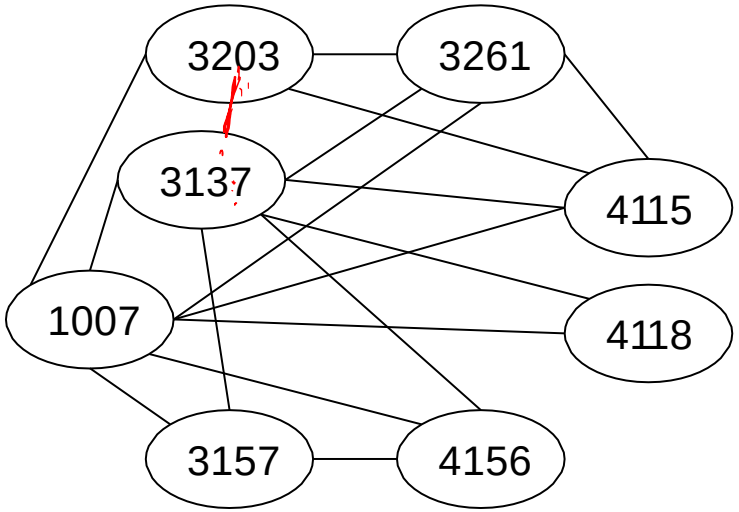
Graph Coloring and Scheduling

- Convert problem into a graph coloring problem.
- Represent Courses as vertices.
- Connect two vertices with an edge, if the 2 courses have a student in common.



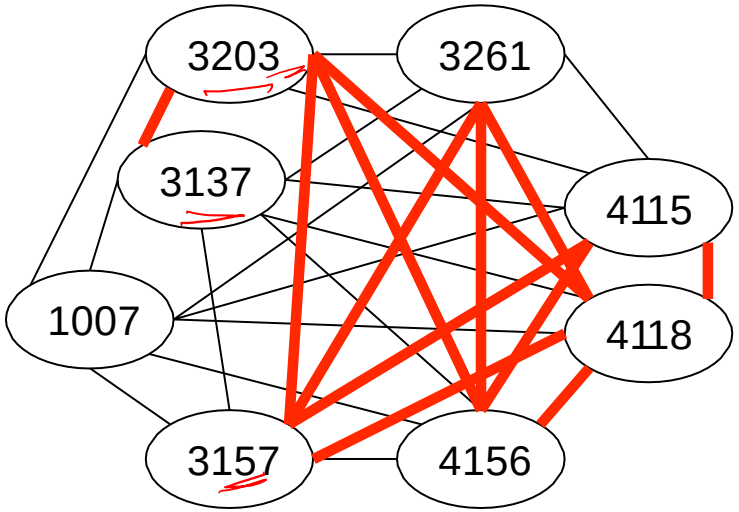
Put complementary edges first

One way: First put edges down where students mutually excluded…



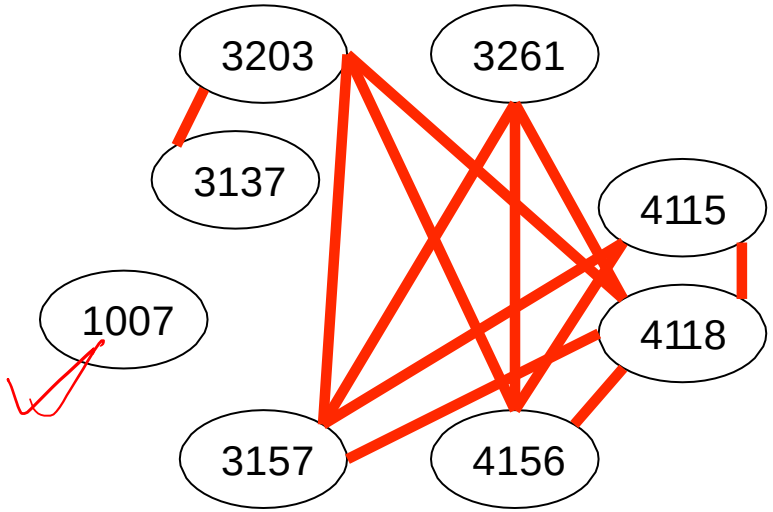
Then put proper edges

…and then consider its complementary graph:

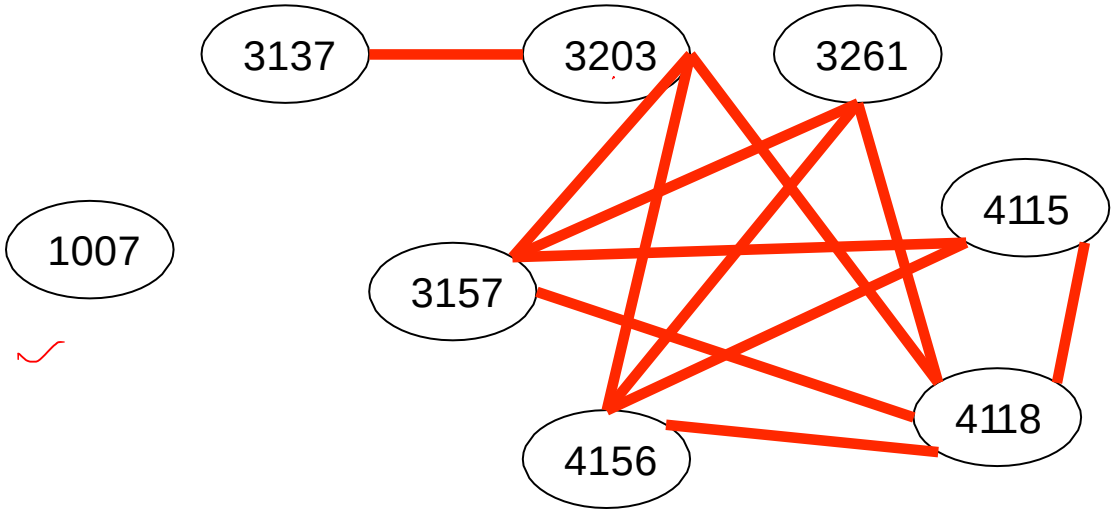


Common students shown on Graph

The graph now shows common students in courses

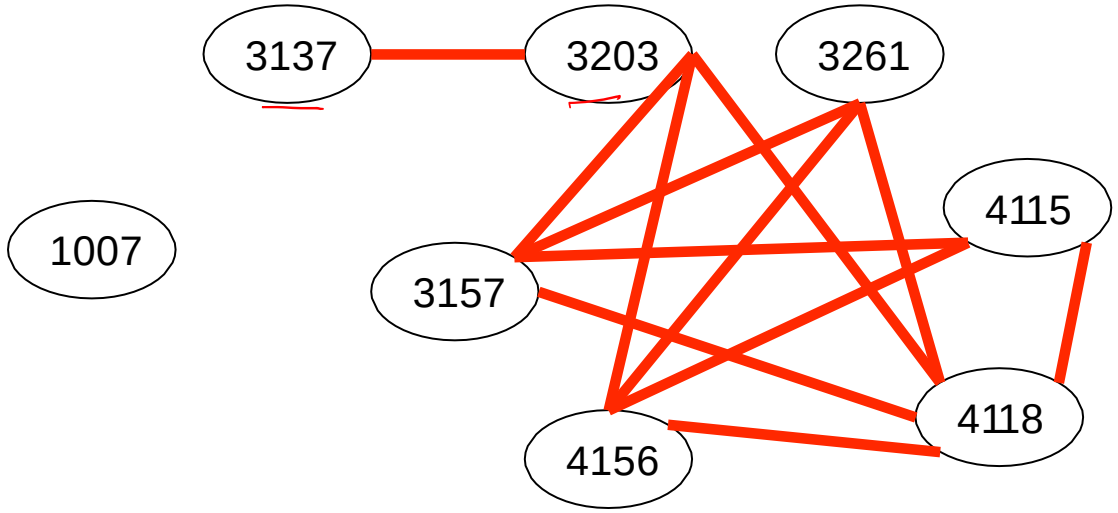


Redraw the graph for convenience:



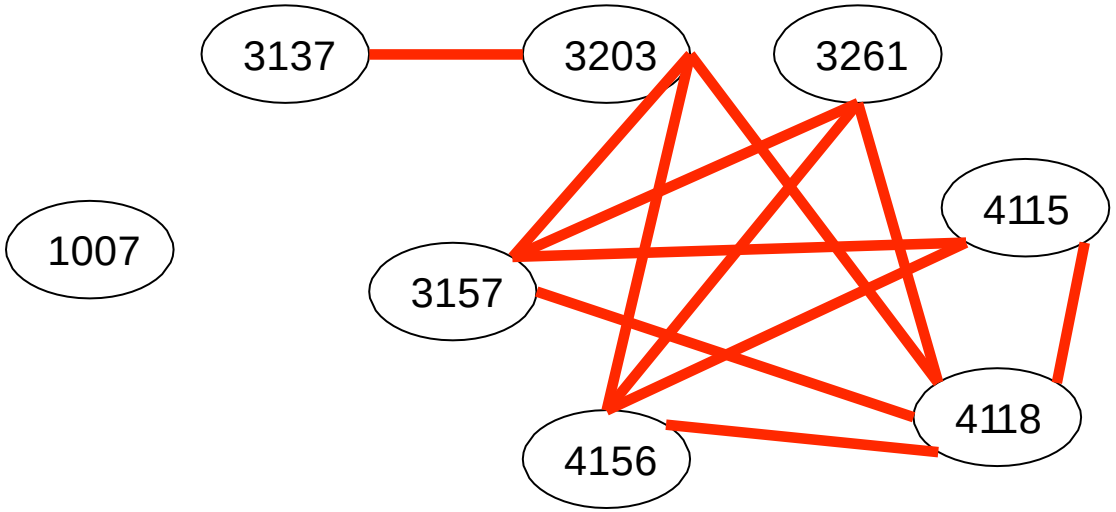
Can one use simply 1 color ?

The graph is obviously not 1-colorable because there exist edges.



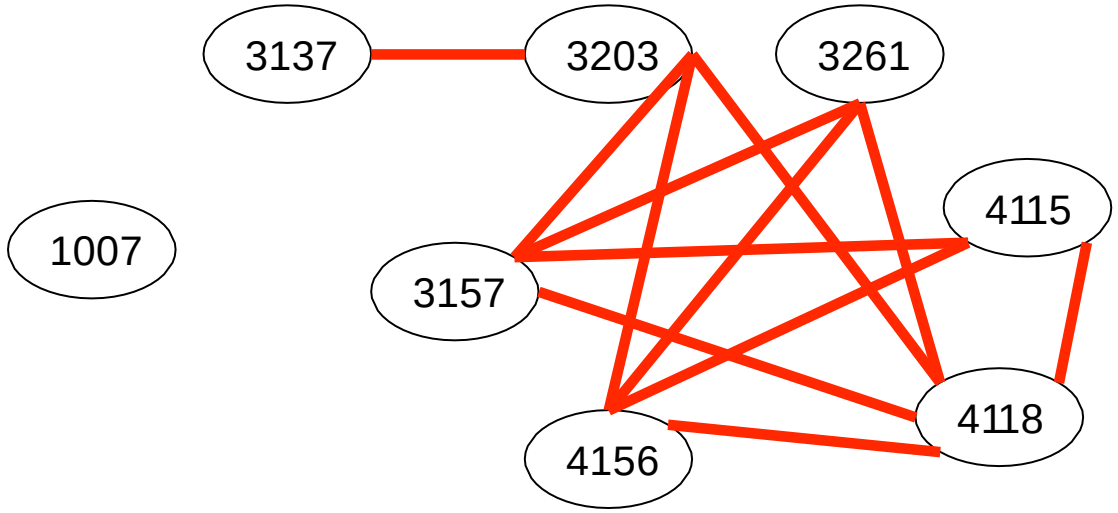
Can one manage with 2 colors ?

The graph is not 2-colorable because there exist triangles.



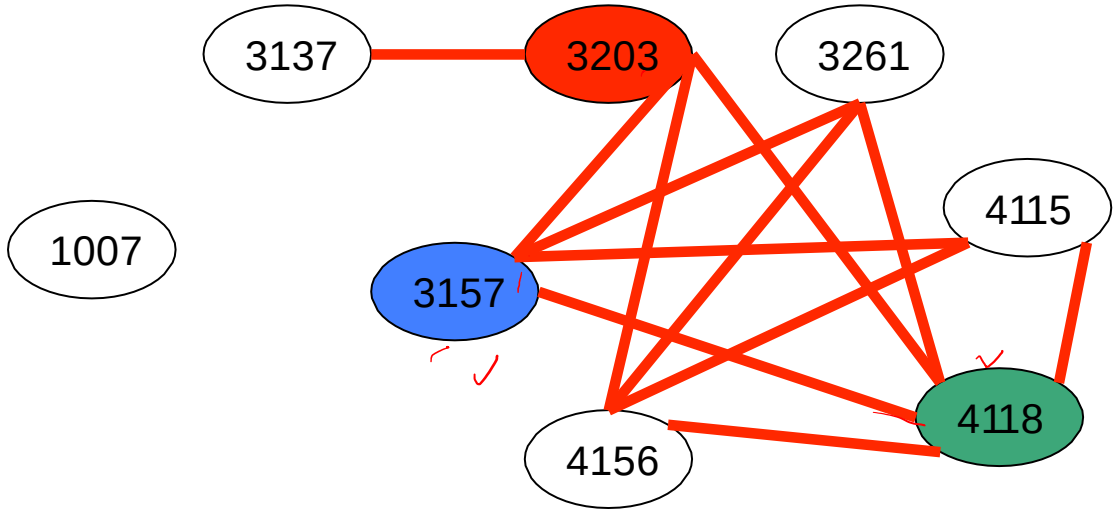
Try with 3 colors….

Is it 3-colorable? Try to color with Red, Green, Blue.



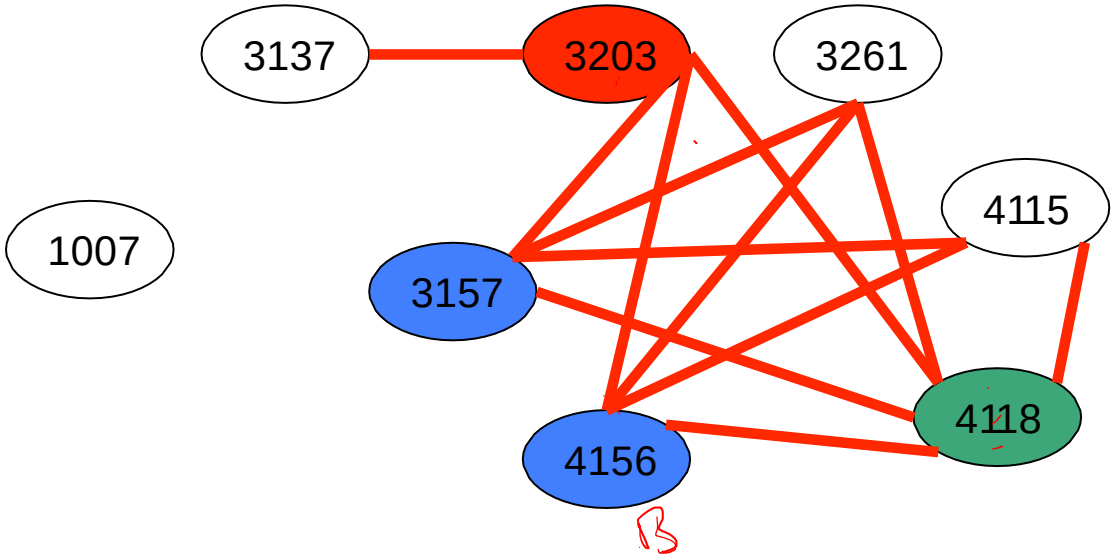
Graph Coloring

Pick a triangle and color the vertices with 3 colors
3203-Red, 3157-Blue and 4118-Green.



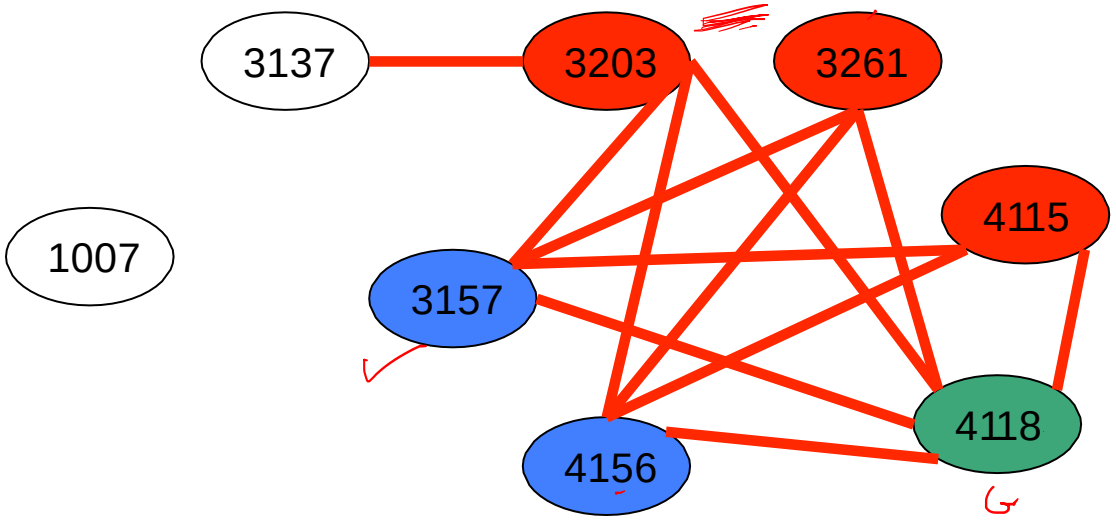
Graph Coloring

So 4156 must be Blue:



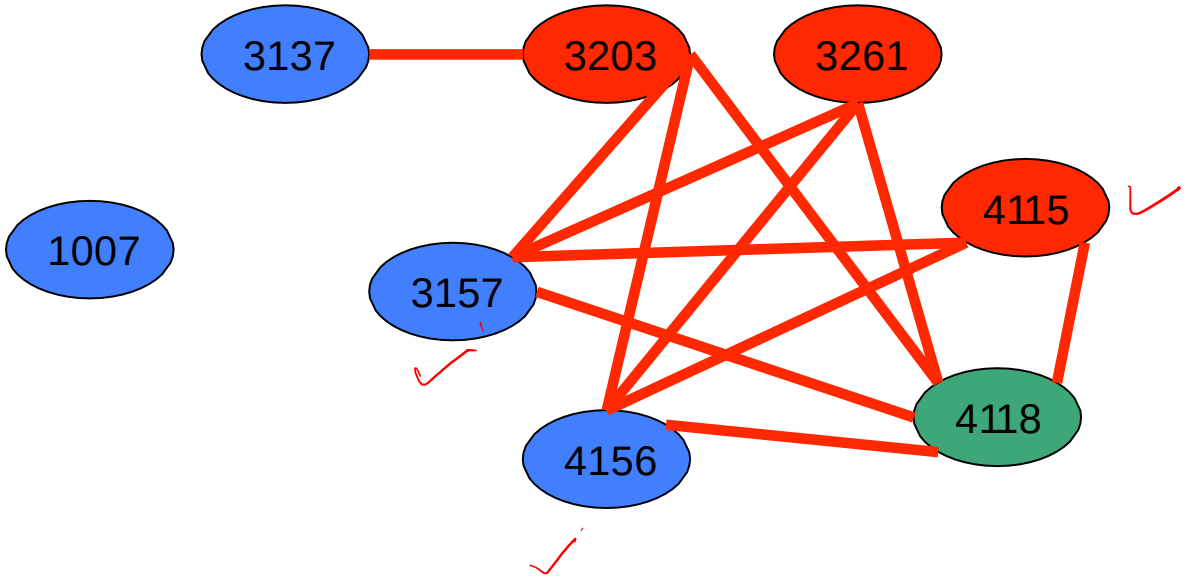
Graph Coloring

So 3261 and 4115 must be Red.



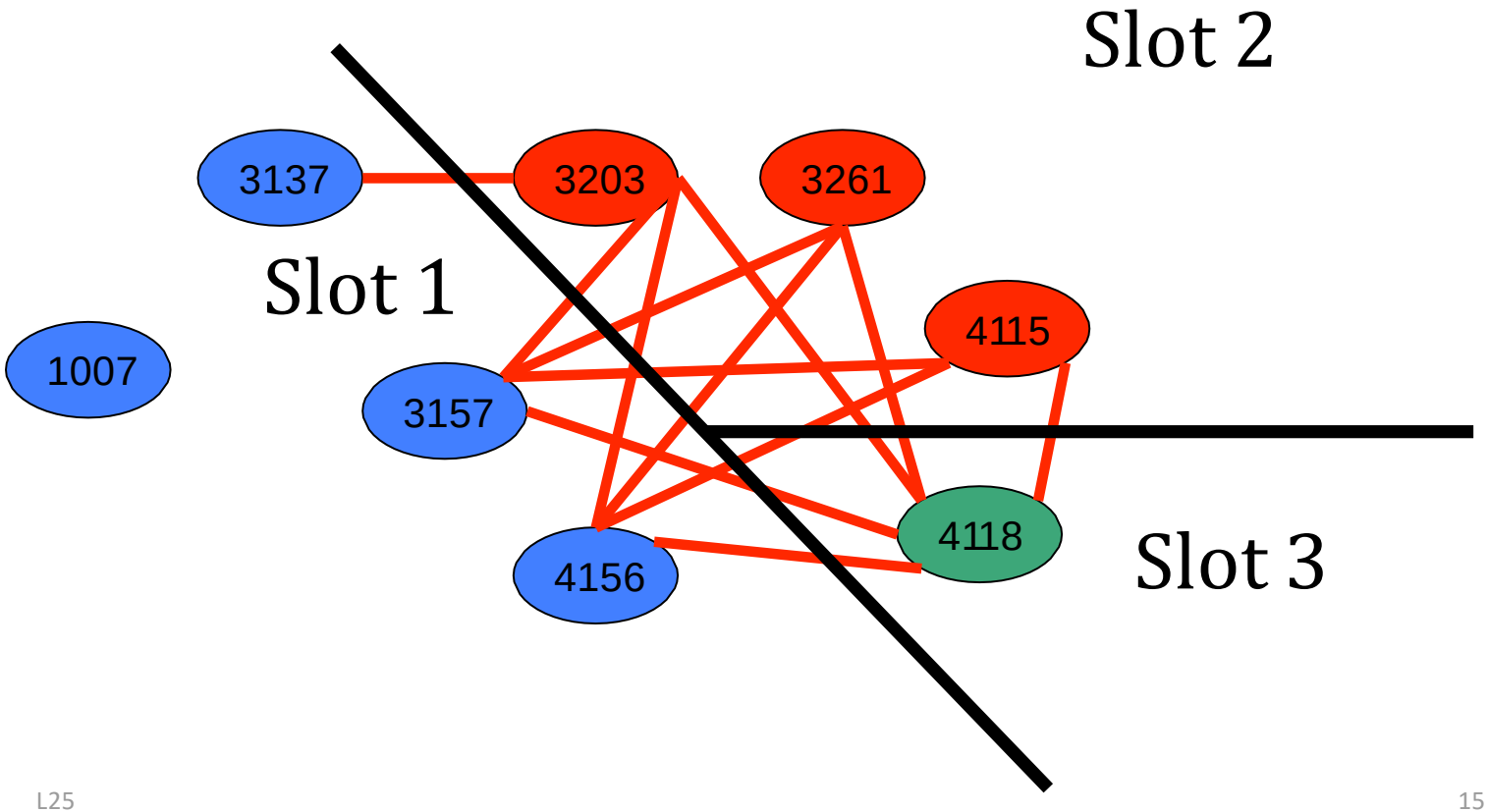
Graph Coloring

3137 and 1007 easy to color – pick Blue.



Graph completely colored

3 Colors are sufficient. Therefore we need 3 exam slots:



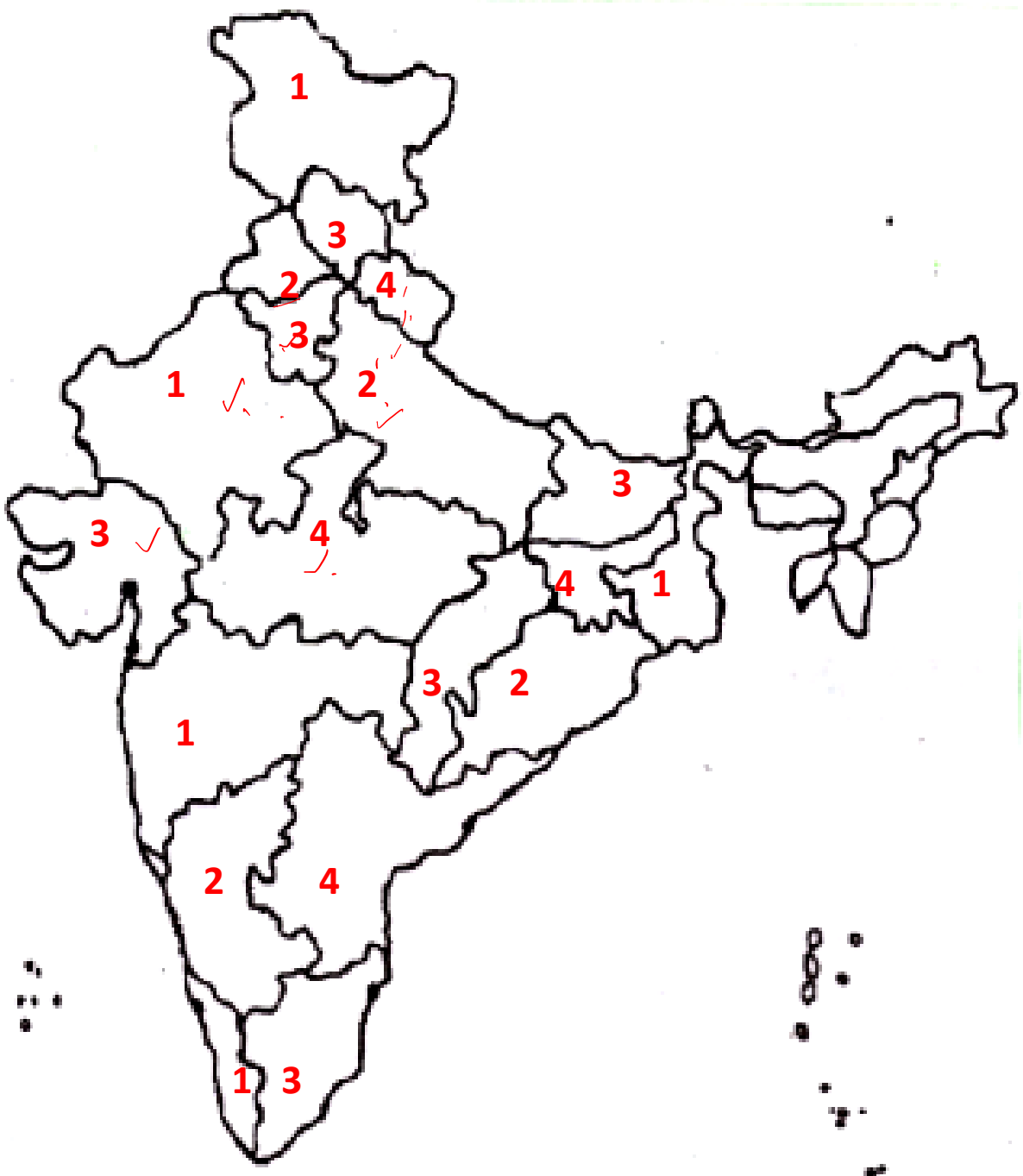
- Exams can be held for all the courses using just 3 time slots.

- In general, how many colors needed to color a geographical map?

US Map Coloring: with 5-Colors

US Map Coloring: with 4-Colors

- Any planar map can be colored with 4 colors



India Map with 4 colors

- Graph Coloring algorithm

graph_coloring(i)

```
                                        // Node i
if color_okay(i)
    if( i == n)                         //all vertices colored
        print color [ 1 2  . . . . n]
    else
        j = 1
        while ( j < = M) do             // do for all M colors
            color(i+1) = j              // Assign j to node i+1
            graph_coloring(j+1)
            j = j + 1
end
```

color_okay(i)

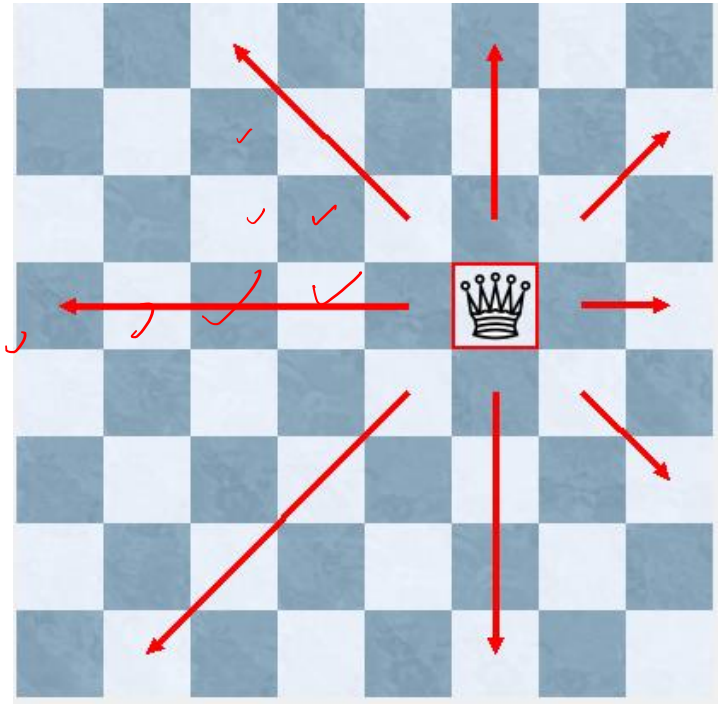
- flag = true
- for j = 1 to i – 1 do
- if (i is neighbor of vertex j) then // from Adjancency matrix
- if color(i) == color (j) then
- flag = false
- return (flag)

Complexity of graph coloring

- It is basically the number of nodes in the state space.
- $1 + M + M^2 + M^3 + \ldots + M^n = (M^{n+1} - 1) / (M - 1)$, where M is no. of colors, and n is no. of vertices
- = **$O(M^n)$** EXPONENTIAL PROBLEM

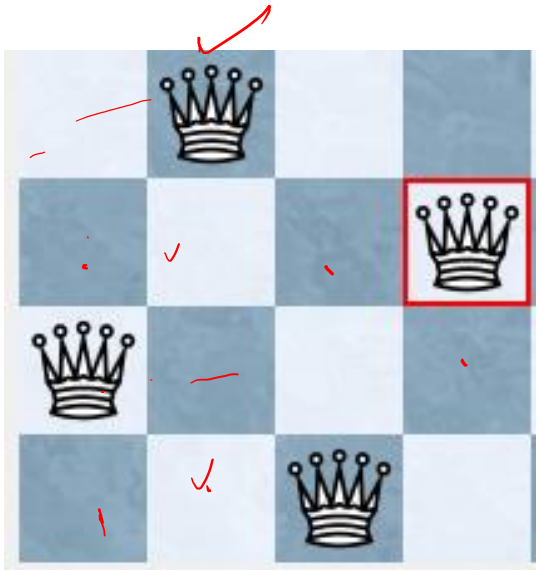
2. n-Queens Problem

Movement of a Queen on a Chessboard



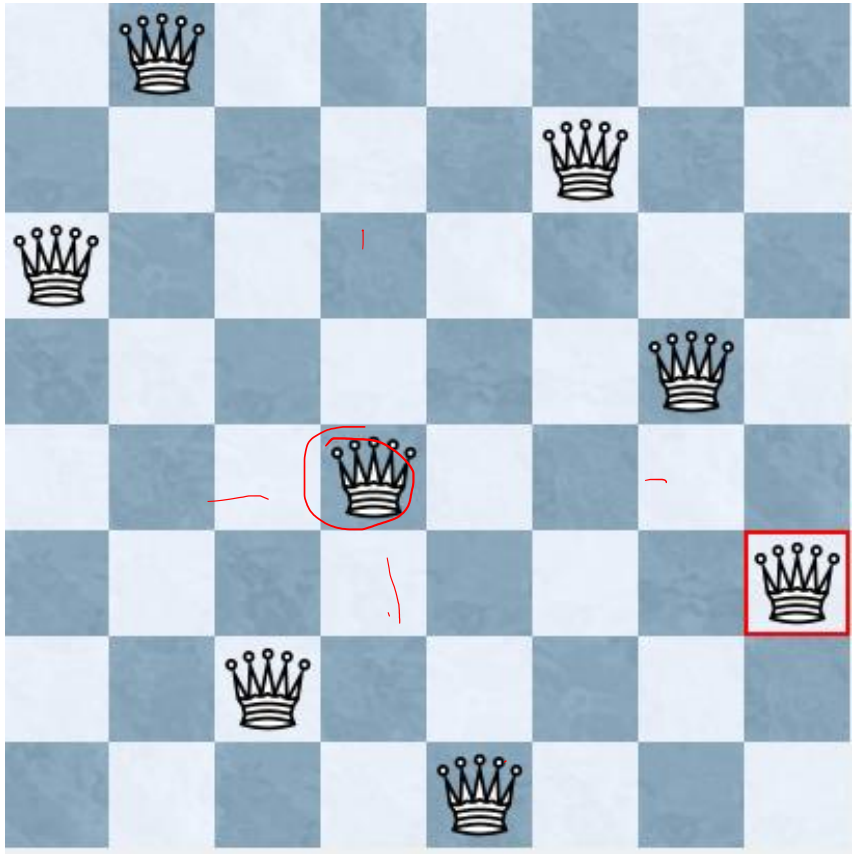
4-Queens Problem

Place *4* queens on a 4x4 chessboard such that no two queens attack each other.



8-Queens Problem

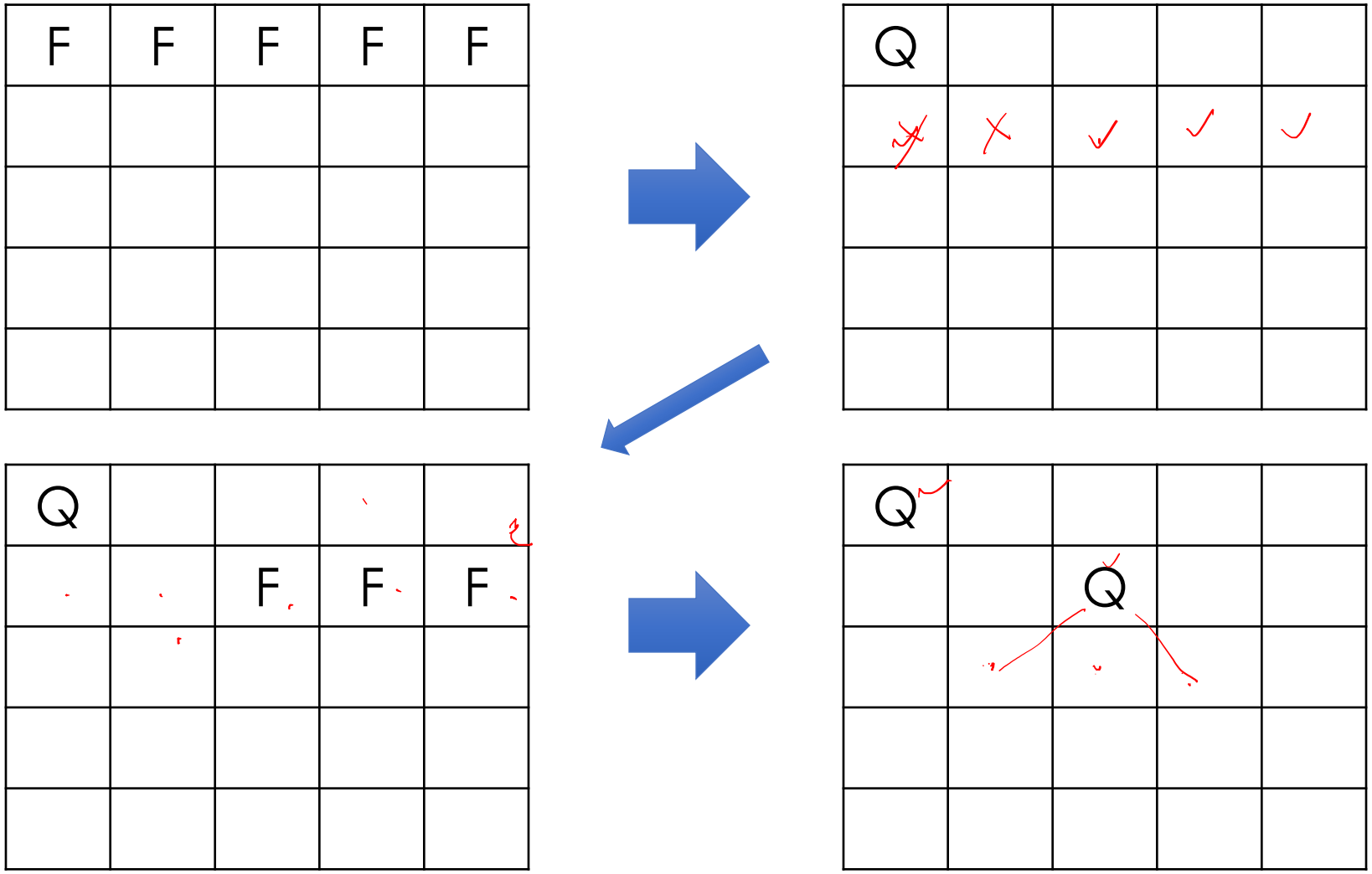
Place *8* queens on a *8x8* chessboard such that no two queens attack each other.



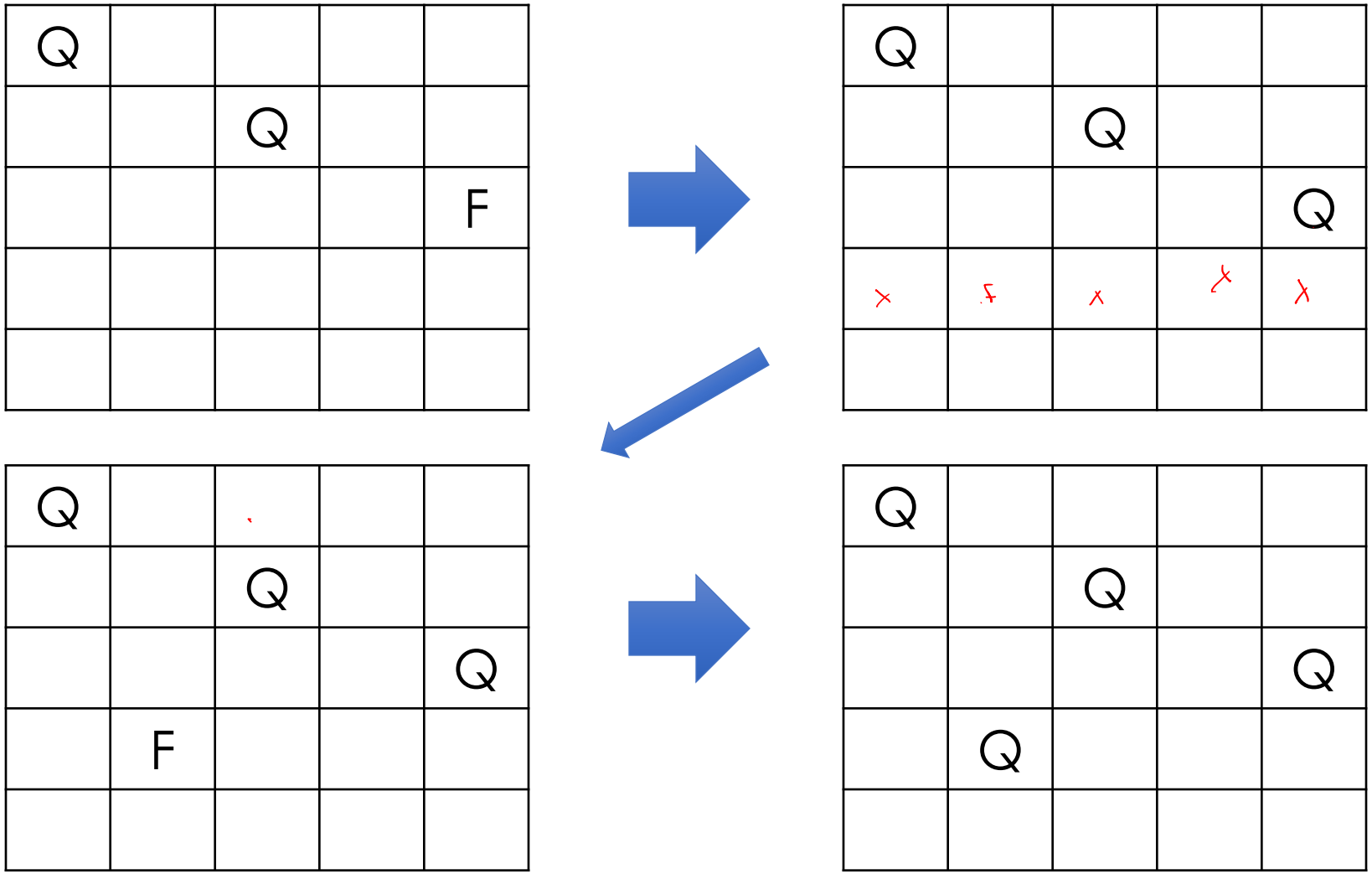
Basic Algorithm

1) Generate a list of free cells on the first row.
2) Place a queen on a free cell
3) Generate a list of free cells on the next row.
   - If no free cells, backtrack (choose another free cell in previous row)
4) Place a queen on next free cell & proceed with step 3 for next row
   - If no next row, proceed to step 5
5) At this point, you have filled all rows, so count/store as a solution

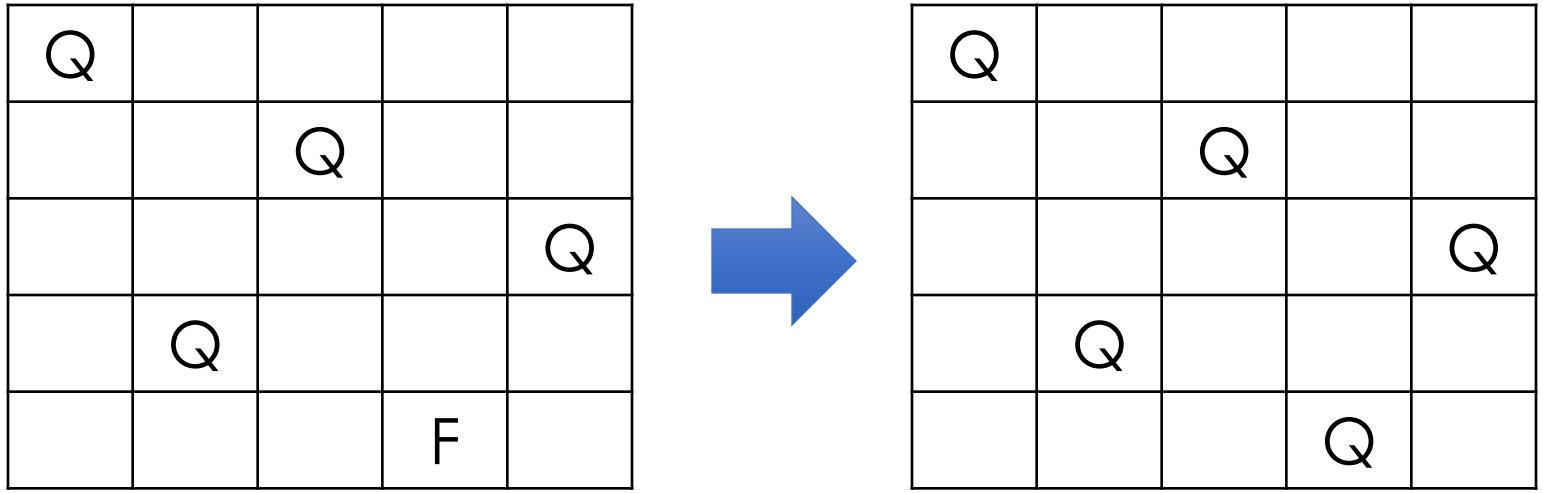
Solving the 5-Queens Problem



Solving the 5-Queens Problem



Solving the 5-Queens Problem



DONE

- The 4-Queens problem can be started with first queen in top most left position.
- The final solution involves lot of backtracking steps.

n-Queens algorithm (Column wise fill)

- 1. Start from column 1, check each row, place Queen in Free position
- 2. Move to next column, place next Queen.
- M. If Queen is safe, check if all queens placed on board
- If yes, print solution
- else, remove last queen and backtrack.
- if no solution found, backtrack still further.
- 4. If at the end solution not found, report error.

queen(i)

```
// Place Queen  i   in col(i)
Begin
    if Okay(i) then          (Okay is a bounding function)
        if ( i = = n) print col[1]. . . . . .col[n]
    else
        for j = 1 to n do              // all  columns j
            col[i + 1 ]  = j
            queen[ i + 1];
end
```

Okay (i)

```
//   feasibility of placing the queen
flag = true
for k = 1 to i – 1   do
     if ( col[i] == col[k]) then                  //rows are same
          if (( col [i] – col[k]  |=  |i – k |
          or (col[i]==col[k] then                  // attacking diagonal
                flag = false;
return (flag)
```

Complexity of n-Queen

- We present a rough scheme for working out number of operations
- First queen has n possibilities
- second one has n – 1 possibilities
- third one has n – 2 possibilities

- Roughly speaking it works out to **O(n!)**