University of Colorado **Boulder**

Department of Computer Science

CSCI 2824: Discrete Structures

Chris Ketelsen

Graphs
The Coloring Problem

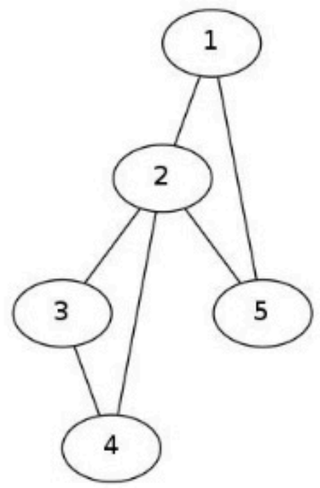
Graph Coloring

The Graph Coloring problem is an important problem in CS

We'll look at two specific applications:

- Scheduling Tasks
- Saving telecommunications companies bazillions of $'s

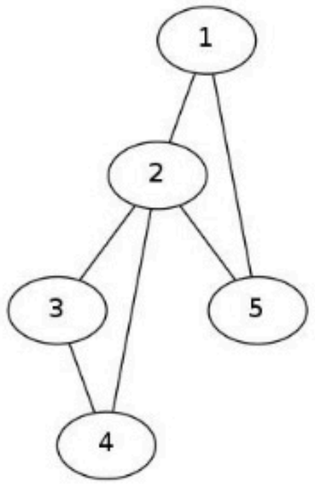
The goal of Graph Coloring is to assign labels (colors) to vertices in the graph so that no vertices of the same color share an edge.



Cycles

Before we can solve the GCP, we need a few more tools

Def: Let $G = (V, E)$ be a graph (directed, or undirected). A **cycle** of a graph is a sequence of alternating vertices and edges that start and end on the same vertex but don't repeat any other vertices. The number of edges traversed is called the **length** of the cycle.



Example: $1 \to 2 \to 5 \to 1$ or $2 \to 3 \to 4 \to 2$

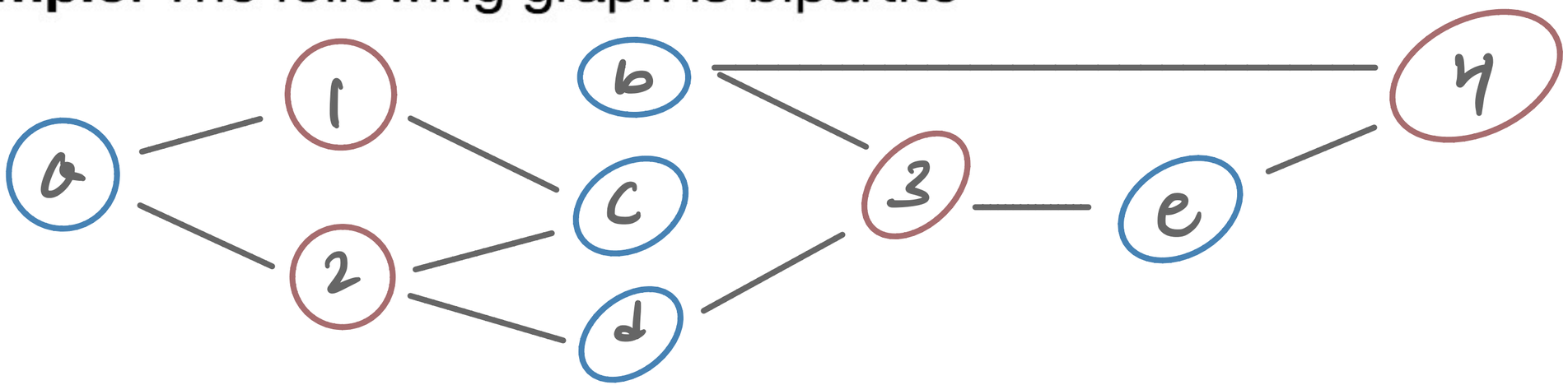
Bipartite Graphs

We'll start our discussion on the Coloring problem by looking at a special type of graph that can be colored using only two colors

Def: A $G = (V, E)$ is called **bipartite** iff the vertex set V can be partitioned into two disjoint sets V_1 and V_2 such that

- $V_1 \cup V_2 = V$ and $V_1 \cap V_2 = \emptyset$
- Any edge in E goes from a vertex V_1 to V_2, or vice-versa.

Example: The following graph is bipartite



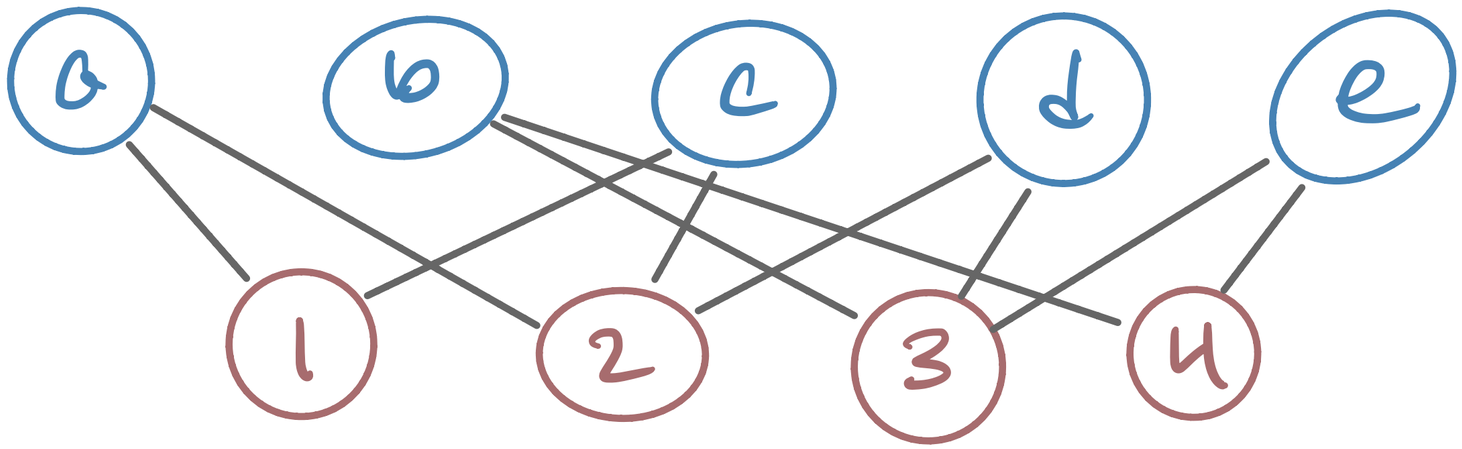
Bipartite Graphs

We'll start our discussion on the Coloring problem by looking at a special type of graph that can be colored using only two colors

Def: A $G = (V, E)$ is called **bipartite** iff the vertex set V can be partitioned into two disjoint sets V_1 and V_2 such that

- $V_1 \cup V_2 = V$ and $V_1 \cap V_2 = \emptyset$
- Any edge in E goes from a vertex V_1 to V_2, or vice-versa.

Example: It might be slightly easier to see if we rearrange a bit

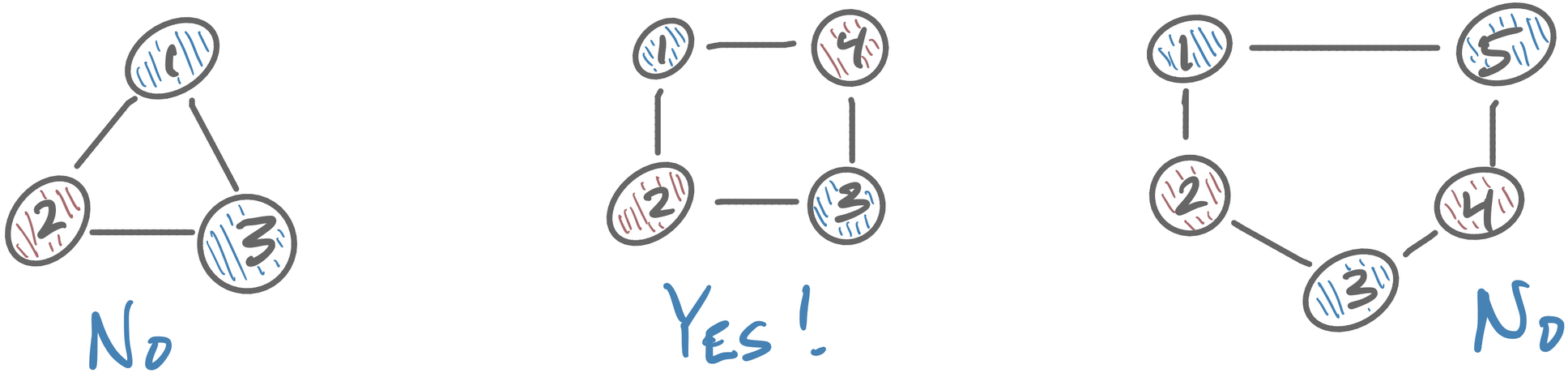


Bipartite Graphs and Cycles

We now state the key result involving bipartite graphs

Theorem: A graph G is bipartite *iif* it has **no odd length cycles**

Proof of One Direction: Let's start with just a single cycle



Notice that if the cycle has an even length it's bipartite, and if it has an odd length it is not bipartite. Now, picture putting an odd cycle into any graph. That particular cycle in the graph cannot be bipartite (two-colored) thus the entire graph cannot be bipartite

General Coloring Problems

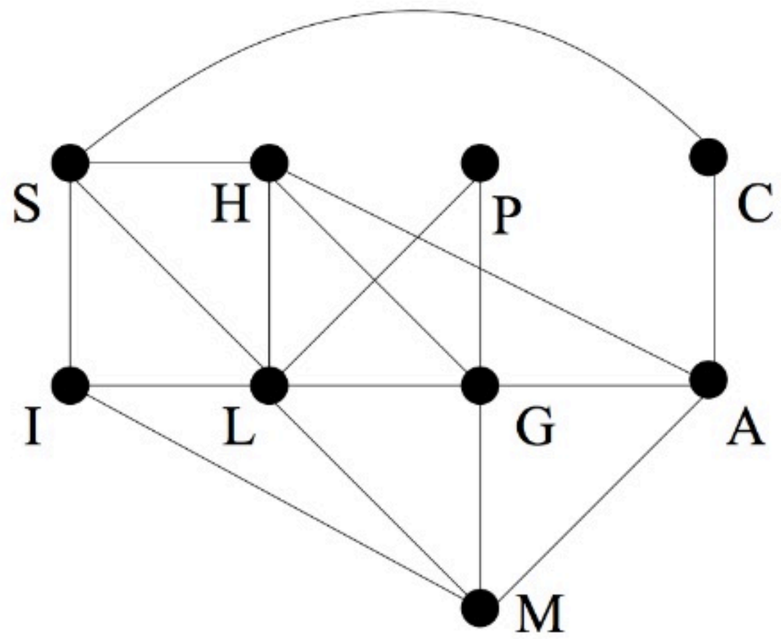
Example: Suppose you are responsible for scheduling final exams at a university. You want to make sure that any two courses with a common student are scheduled at different times to avoid a conflict.

Class	*A*	*C*	*G*	*H*	*I*	*L*	*M*	*P*	*S*
Astronomy		*X*	*X*	*X*			*X*		
Chemistry	*X*								*X*
Greek	*X*			*X*		*X*	*X*	*X*	
History	*X*		*X*			*X*			*X*
Italian						*X*	*X*		*X*
Latin			*X*	*X*	*X*		*X*	*X*	*X*
Music	*X*		*X*		*X*	*X*			
Philosophy			*X*			*X*			
Spanish		*X*		*X*	*X*	*X*			

General Coloring Problems

Example: Suppose you are responsible for scheduling final exams at a university. You want to make sure that any two courses with a common student are scheduled at different times to avoid a conflict.

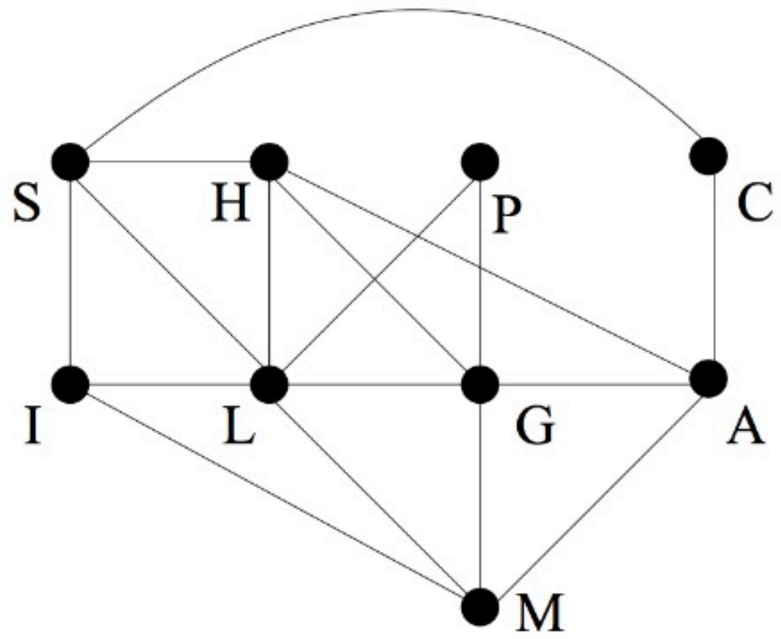
We can encode the information in a graph by assigning each course to a vertex, and connecting two vertices by an edge if they have a common student



General Coloring Problems

Example: Suppose you are responsible for scheduling final exams at a university. You want to make sure that any two courses with a common student are scheduled at different times to avoid a conflict.

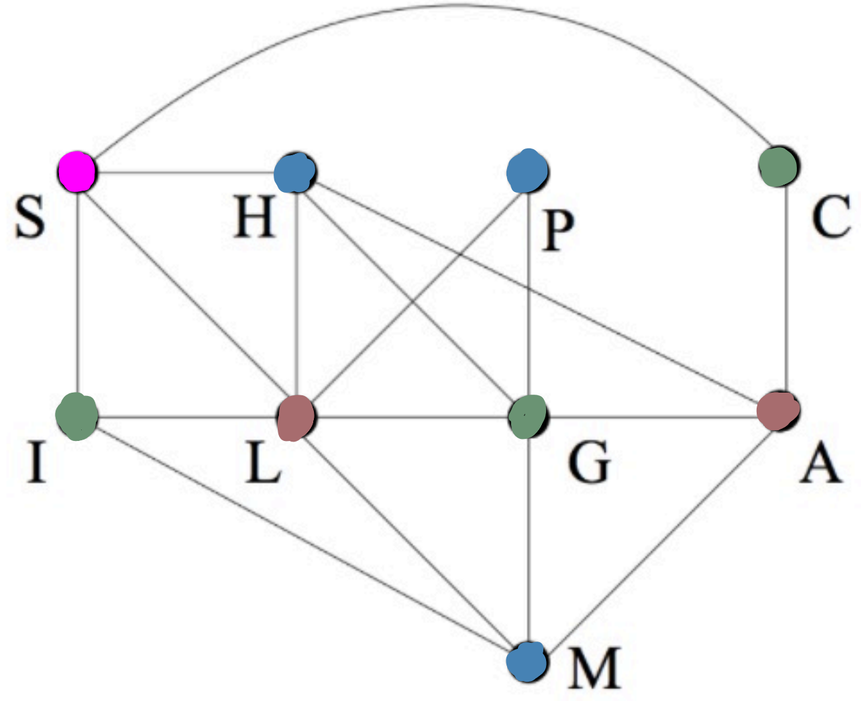
Think about finding a coloring of the graph, such that no adjacent vertices have the same color. Then associating a final exam time-slot with each color solves the problem.



General Coloring Problems

Example: Suppose you are responsible for scheduling final exams at a university. You want to make sure that any two courses with a common student are scheduled at different times to avoid a conflict.

One such coloring of this graph is as follows. Notice that it requires four different colors.



General Coloring Problems

So, this raises some **questions**:

- Can I always find a coloring of a graph?
- Are there certain colorings that are better than others?

General Coloring Problems

So, this raises some **questions**:

- Can I always find a coloring of a graph?
- Are there certain colorings that are better than others?

Answers:

- Yes! If the graph has n vertices then use n colors!
- Yeahhh, but the scheduling problem motivates us choosing as few colors as possible.

Def: The **chromatic number** of a graph, $\chi(G)$, is the minimum number of colors required to color a graph

A Greedy Coloring Algorithm

Now we describe a Greedy graph coloring algorithm, that while not guaranteed to find the minimum coloring of a graph, it *is* guaranteed to find *a* coloring that works.

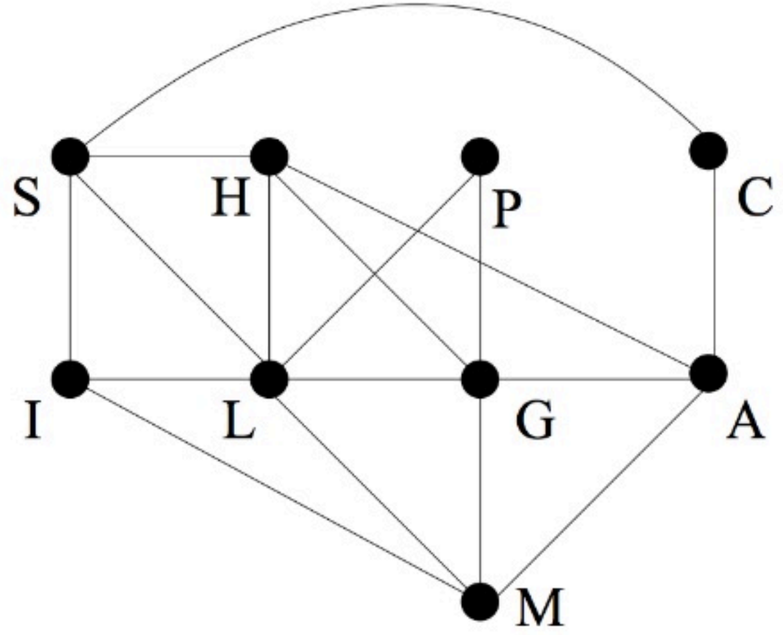
The procedure requires us to number the colors consecutively

1. Order the vertices in some (arbitrary) way
2. Color the first vertex with color 1
3. Pick the next uncolored vertex v. Color it with the lowest-numbered color that has not been used on any vertices adjacent to v. If you run out of colors, add a new color.
4. Repeat Steps 2-3 until all vertices are colored

A Greedy Coloring Algorithm

Example: Use the Greedy Coloring Algorithm to color the Exam schedule graph

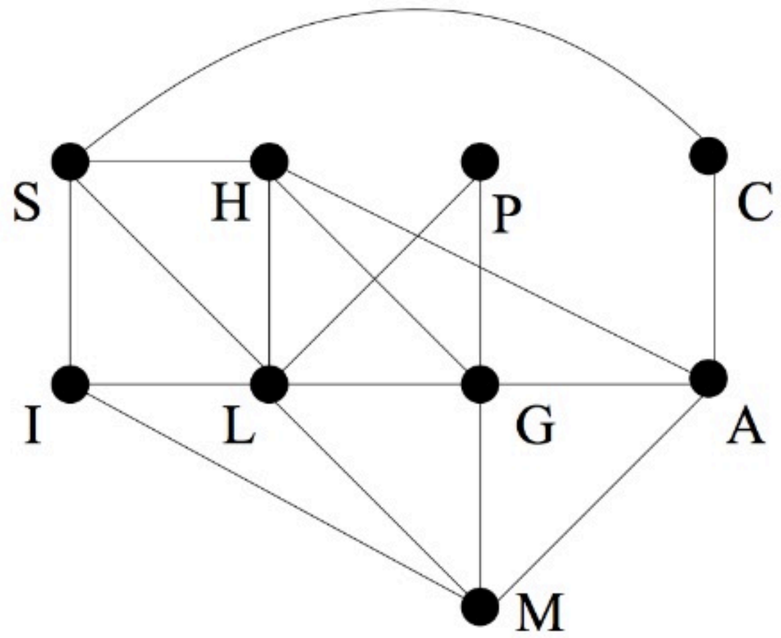
Let's order the vertices as $G, L, H, P, M, A, I, S, C$ and start with colors Green, Red, Blue (in that order)



A Greedy Coloring Algorithm

Example: Use the Greedy Coloring Algorithm to color the Exam schedule graph

Let's this time order the vertices as $A, I, P, M, S, C, H, L, G$



A Greedy Coloring Algorithm

OK, so we got two very different colorings based on the initial orderings of the vertices

This often happens with Greedy algorithms. In practice, you can run the algorithm a few times for different orderings and choose the one that provides the minimal coloring

Is there any way to tell how well we can do?

A Greedy Coloring Algorithm

OK, so we got two very different colorings based on the initial orderings of the vertices

This often happens with Greedy algorithms. In practice, you can run the algorithm a few times for different orderings and choose the one that provides the minimal coloring

Is there any way to tell how well we can do?

We can't (easily) say what the minimal coloring will be

But we can say what the worst case coloring will be

A Greedy Coloring Algorithm

Greedy Coloring Theorem: If d is the largest degree of any vertex in graph G, then G has a coloring with $d + 1$ or fewer colors, i.e. the chromatic number of G is at most $d + 1$.

Proof Sketch: Suppose that d is the largest degree in the graph

$\Rightarrow$ All vertices have d of fewer connections

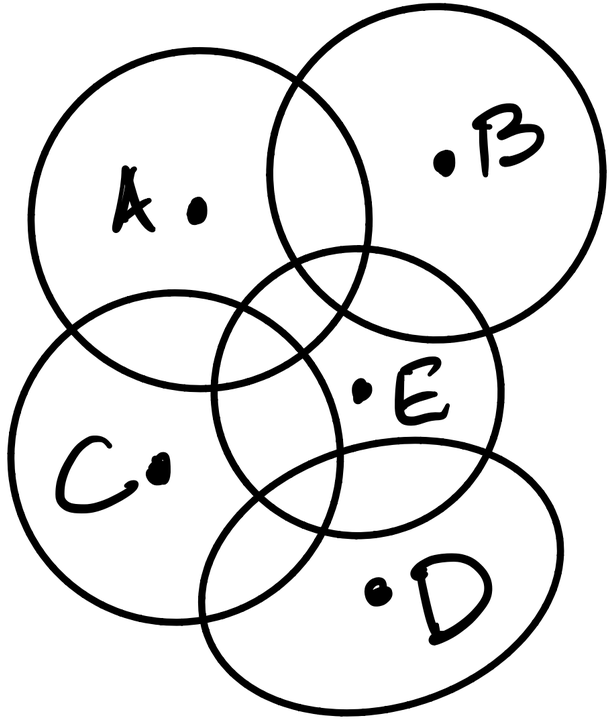
$\Rightarrow$ When we color a new vertex, it is connected to at most d other vertices, which may already be colored

$\Rightarrow$ Then there at most d colors we must avoid using

$\Rightarrow$ In the worst case we need $d + 1$ colors

Choosing Signal Tower Frequencies

Example: Radio Towers that have overlapping coverage must be assigned different frequencies to avoid interference with each other. But companies have to pay the federal government to use a particular frequency (and it's not cheap). The question is, given a coverage map for several Radio Towers, how many frequencies do you need to lease from the government?



Choosing Signal Tower Frequencies

Example: Radio Towers that have overlapping coverage must be assigned different frequencies to avoid interference with each other. But companies have to pay the federal government to use a particular frequency (and it's not cheap). The question is, given a coverage map for several Radio Towers, how many frequencies do you need to lease from the government?