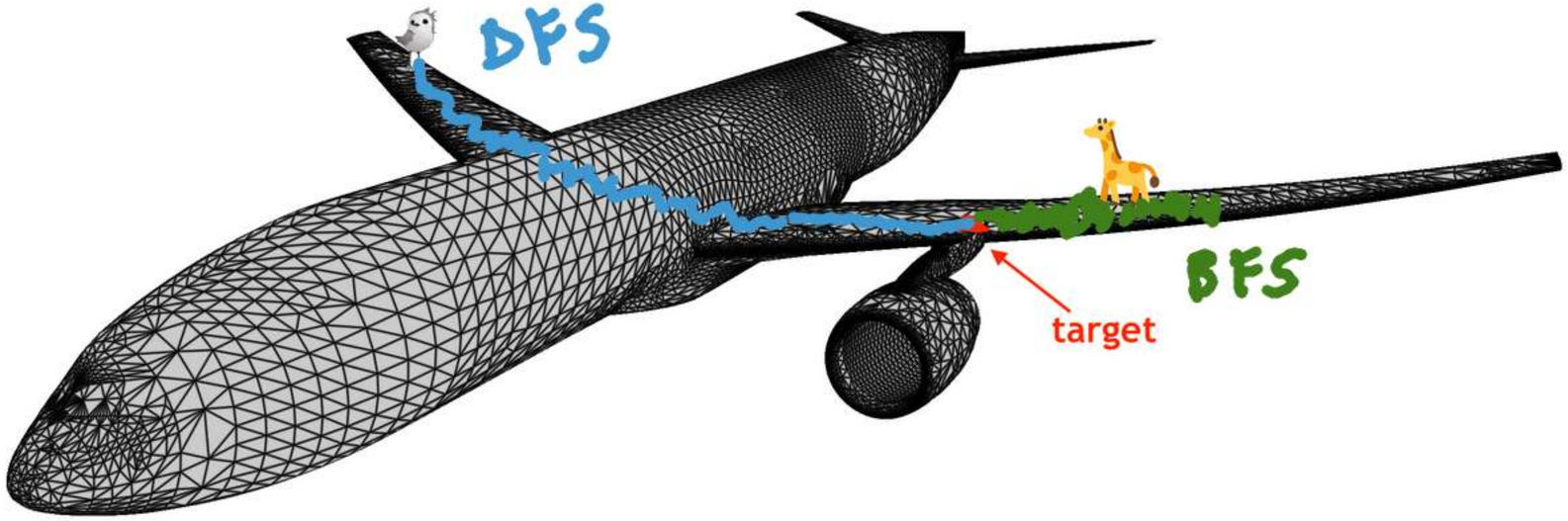
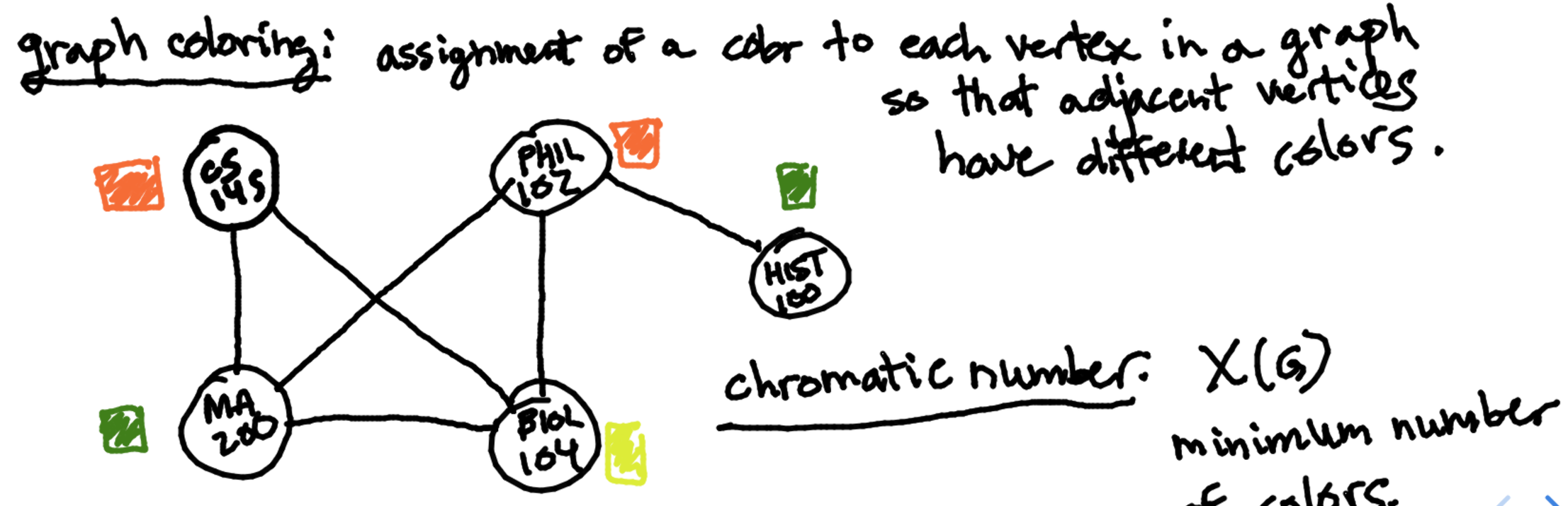
How I use DFS and BFS.

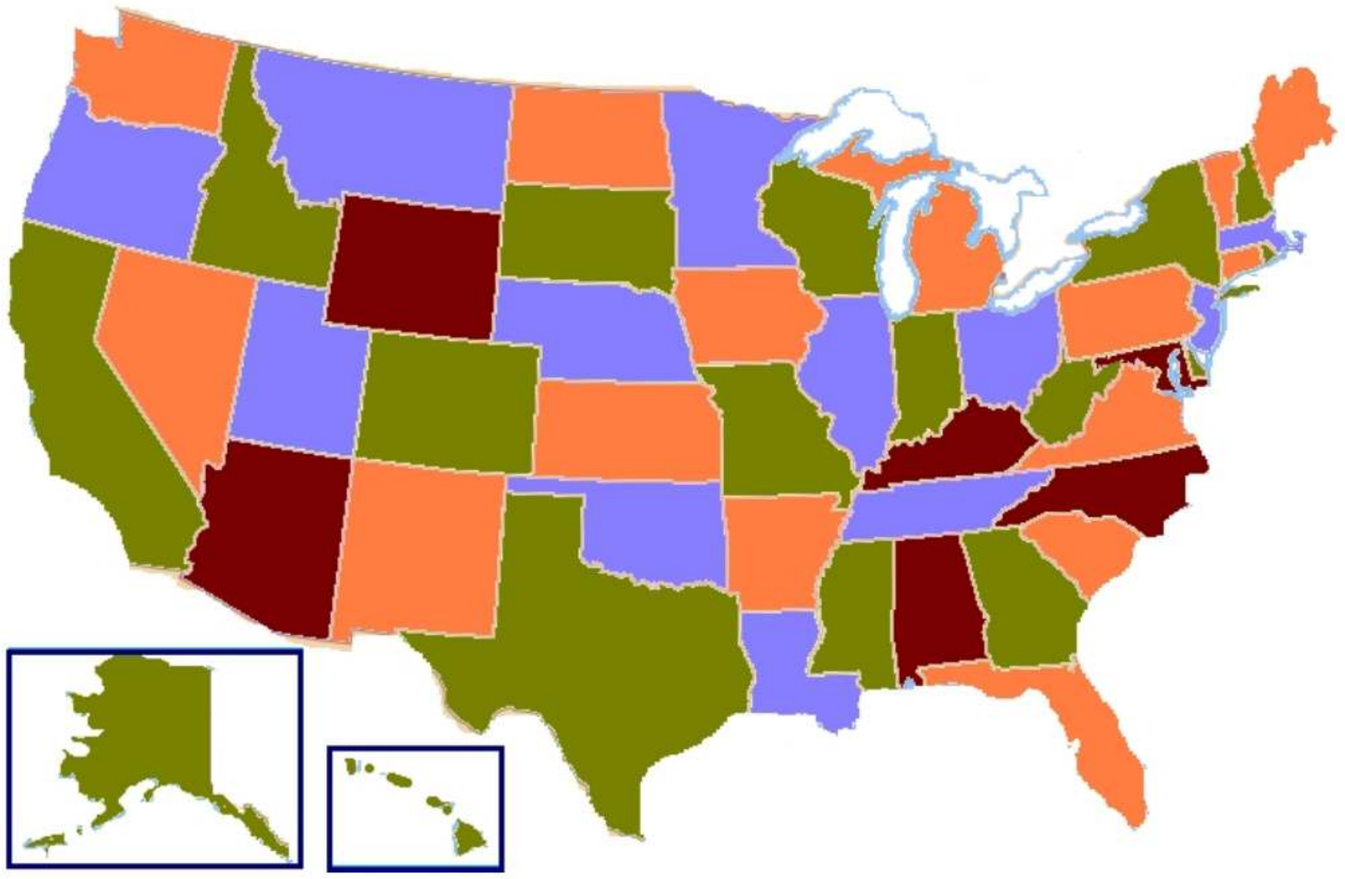


Let's interpret this as a graph:

	CSCI 145	MATH 200	PHIL 102	BIOL 104	HIST 100
CSCI 145		X		X	
MATH 200	X		X	X	
PHIL 102		X		X	X
BIOL 104	X	X	X		
HIST 100			X		



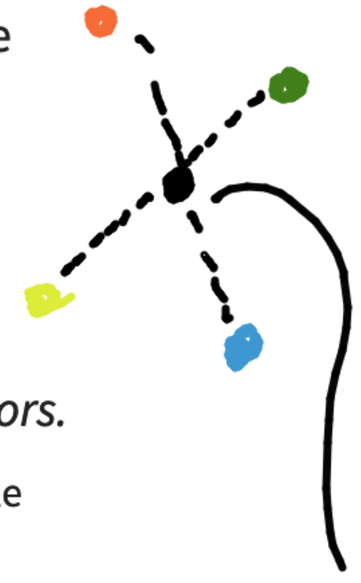
Also useful for coloring maps!



four color theorem

Bounding the chromatic number.

Theorem: A graph $G = (V, E)$ with $n = |V|$ vertices and a maximum degree of k can be colored with $k + 1$ colors.



Proof: We use a proof by induction on the number of vertices n. Let the induction hypothesis be the predicate $p(n)$:
A graph with n vertices and maximum degree k can be colored with $k + 1$ colors.

- **Base case:** For n=1, we have 0 edges, so the maximum degree is 0. This single vertex can be assigned a single color, which is the maximum degree (0) + 1.

- **Inductive step:** Assume $p(n)$ is true. We need to show a graph with n+1 vertices and maximum degree k can be colored with k+1 colors.

  Consider a graph G with n+1 vertices and maximum degree k. Remove a vertex v from G to get a n-vertex graph G'. The maximum degree of G' is also k which can be colored with $k + 1$ colors by our assumption.

  Let's add v back in. Since the degree of v will be at most k, suppose (in the worst case), all vertices connected to v have a different color, which means there are k colors we cannot pick from. Therefore, pick the $(k + 1)$-th color for v, which shows that G' can be colored with $k + 1$ colors.

v

pick the $(k+1)^{th}$ color

Therefore, by induction on n, a graph with maximum degree k can be colored with $k + 1$ colors.

Tips for writing pseudocode

1. Use **set notation** as much as possible (for example, $v \in V$, $e \in E$).
2. Determine which **data structures** you need: lists/arrays [], sets {}, maps/dictionaries {}.
3. Determine if you need any auxiliary functions (like for lists, sets, or graph helpers).
4. Use `if`, `else`, `else if` for conditionals.
5. Use `for`, `while` for loops (for example, `for` $i = 1 \to n$ or `for` $v \in V$).
6. Use comments (#) to clarify things!

depthFirstSearch(G)

input: connected graph $G = (V_G, E_G)$
output: spanning tree T

```
1  u ← arbitrary vertex in V_G
2  T ← ({u}, ∅)
3  visit(u, G, T)
```

visit(u, G, T)

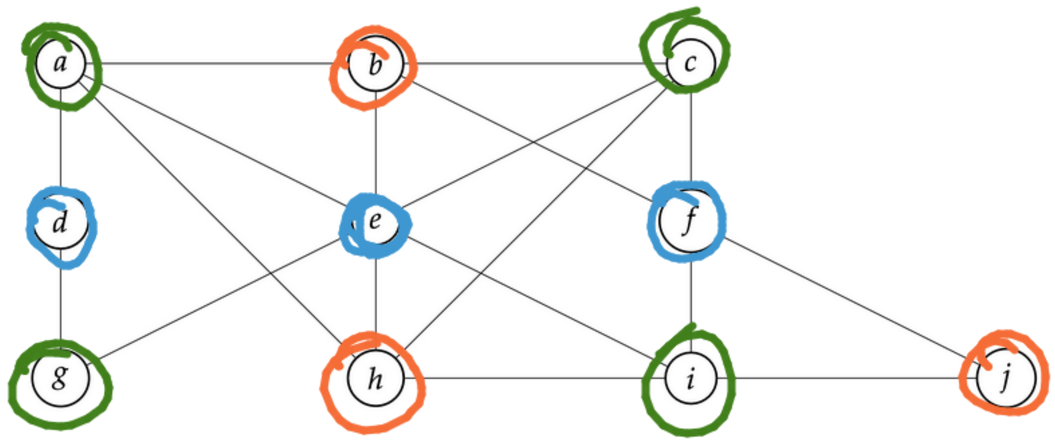
input: starting vertex u, connected graph $G = (V_G, E_G)$,
current spanning tree $T = (V_T, E_T)$
output: updated spanning tree $T = (V_T, E_T)$

```
1  for v ∈ neighbors(u, G)
2      if v ∈ V_T
3          continue
4      E_T ← append({u, v})
5      V_T ← append(v)
6      visit(v, G, T)
```

A graph coloring algorithm: (from *DMA 5th Ed. by Rosen*)

- Re-order vertices in order of decreasing degree: $\deg(v_1) \geq \deg(v_2) \geq \cdots \geq \deg(v_n)$.
- Assign color 1 to v_1 (the vertex with the highest degree), as well as to the next vertex in the list not adjacent to v_1 (if possible). Keep assigning color 1 to all vertices in this list not adjacent to a vertex that already has color 1.
- Now assign color 2 to the first vertex in the list that is not colored, and proceed down the list as in the last step (this time, coloring vertices that are not adjacent to a vertex with color 2).
- Keep doing this, introducing new colors, until all vertices have been colored.

1. Apply the steps above to color the graph below.
2. Translate the steps above into `pseudocode`.



Possible Python code for our graph coloring algorithm.

```
# initialize color of all vertices
colors = dict()
for u in vertices:
    colors[u] = 0  # initialize to no color

color = 1 # first color
count = 0 # count number of processed vertices
while count < len(graph.vertices):
    for u in graph.vertices: # find vertex with max deg
        if colors[u] == 0: # not processed yet
            colors[u] = color # try current color!
            for v in vertices: # check if allowed
                if (v in graph.neighbors[u] \
                  and colors[v] == color):
                    colors[u] = 0 # nope, reset color
                    break
            if colors[u] == color:
                count += 1 # this color was okay!
    color += 1 # go to next color
```

Possible **pseudocode** for our graph coloring algorithm.

```
graph_coloring(G)

    input: G = (V, E) # vertices V are ordered by decreasing degree
    output: C    # colors of the vertices V
 1  for  i = 1 → n
 2      C[i] = 0 # initialize colors to 0 (unassigned)
 3  count ← 0 # no vertices colored yet
 4  color ← 1 # try the first color
 5  while count < n
 6      for  i = 1 → n
 7          if C[i] == 0 # vertex is not yet colored
 8              C[i] = color # assume we can use this color, until we find out otherwise
 9              for  j = 1 → n
10                  if {v_i, v_j} ∈ E and C[j] == color
11                      C[i] = 0 # we cannot use this color, reassign to 0
12                      break
13              if C[i] == color
14                  count = count + 1 # this coloring worked!
15      color = color + 1 # go to the next color
```