USACO Silver Important Ideas:

DFS:

Type of graph traversal to visit all vertices.

Ex: generate an order to go through vertices so that each vertex is visited exactly once.

Goes through possible paths, going forward to new vertices if it can.

If a dead end is reached, mark as a dead end, then retrace and look for more vertices.

Make a Boolean array to store whether a node has been visited or not

**Naturally recursive (better version) – Recursive Preorder DFS**

visited = [False] \* G.size()

def dfs(G, curr):

visit(curr)

visited[curr] = True

for n in G.neighbors(curr):

if not visited[n]:

dfs(G, n)

**Iterative Method: using a list to keep track of nodes to visit (more generalizable for graph traversal)**

visited = [False] \* G.size()

def dfs(G, v):

arr = [v]

while len(stack) > 0:

v = stack.pop()

if not visited[v]:

visit(v)

visited[v] = true

for n in G.neighbors(v):

if not visited[n]:

arr.append(n)

preorder DFS traversal: order that new vertices are visited

postorder DFS traversal: order of points that we reach a dead end on. Modify by moving visit to the end of recursive DFS after the for loop.

Connected components: DFS on unvisited vertices to do this.

Topological sort: use a reversed DFS in a directed acyclic graph (don’t worry about arrow direction)

**Greedy Algorithm:**

Selects the most optimal choice at each step. Will need a value function to determine which choice is most optimal.

THINK THROUGH HOW TO DEFINE ALGORITHM, be wary of when it could fail.

**Binary Search:**

Start with a sorted list. Look at middle element. If not equal to target, look at greater than or less than. Then look at the half of the data left. Repeat this until we either find the value or the remaining data is empty set, which we return impossible or -1.

L and R are the indices marking the start and end of the range that we are looking at.

Use this formula to prevent sum overflowing: middle = L + floor((R – L)/2)) = L + (R – L)//2

L = 0, R = N – 1

While L <= R:

Mid = L + (R – L)//2

If arr[mid] == target:

Return mid

If a[mid[ < target:

L = mid + 1

Else:

R = mid – 1

Return -1

Searching for greater than equal:

Same thing, except initialize ans to -1, if a[mid] >= target: and = mid.

Rotated/shifted array:

Find smallest element after rotation.

123456 => 456123

Compare all elements with a[-1].

Then you get an array of bools, and you can run binary search on them.

If a list increases then decreases, go through in binary search, bool for whether a thing is greater than the previous element.