# ARTIFICIAL INTELLIGENCE (CSE401)

Lab File

by

**Yash Garg**
A2305218708
7CSE 12Y

to

**Ms. Sangeeta Rani**

**Amity School of Engineering and Technology**
**Amity University, Uttar Pradesh**

# INDEX

| S. No. | Name of Experiment | Date of Assignment | Date of Submission | Remarks |
|---|---|---|---|---|
| 1. | Implementing BFS and DFS | 21/07/2021 | 21/07/2021 | |
| 2. | 8 Puzzle Game (BFS) | 04/08/2021 | 04/08/2021 | |
| 3. | 8 Puzzle Game (A*) | 11/08/2021 | 11/08/2021 | |
| 4. | 8 puzzle using A* algorithm | 25/08/2021 | 25/08/2021 | |
| 5. | Graph colouring problem | 08/09/2021 | 08/09/2021 | |
| 6. | Knapsack Problem | 29/09/2021 | 29/09/2021 | |
| 7. | Preprocessing in NLP | 06/10/2021 | 06/10/2021 | |

# Lab 1: Implementing BFS and DFS

In [1]:

```
from time import time
import sys
import matplotlib.pyplot as plt
```

In [2]:

```
graphOne = {
  0 : [1, 2],
  1 : [2, 3],
  2 : [3],
  3 : []
}
```

In [3]:

```
visited = []
queue = []

def bfs(visited, graph, node):
  visited.append(node)
  queue.append(node)

  while queue:
    s = queue.pop(0)
    print(s)

    for neighbour in graph[s]:
      if neighbour not in visited:
        visited.append(neighbour)
        queue.append(neighbour)

visitedNodes = []

def DFS(visitedNodes, graph, currentNode):
    if currentNode not in visitedNodes:
        print(currentNode)
        visitedNodes.append(currentNode)

        for neighbour in graph[currentNode]:
            DFS(visitedNodes, graph, neighbour)
```

In [4]:

```
start = int(input("\nStarting node for BFS: "))

t0 = time()
bfs(visited, graphOne, start)
t1 = time()

dfstime = t1-t0
print("BFS timed at {:3f} seconds".format(t1-t0))
print("BFS took {} bytes".format(sys.getsizeof(visited) + sys.getsizeof(queue)))
```

```
Starting node for BFS: 1
1
2
3
BFS timed at 0.000789 seconds
BFS took 144 bytes
```

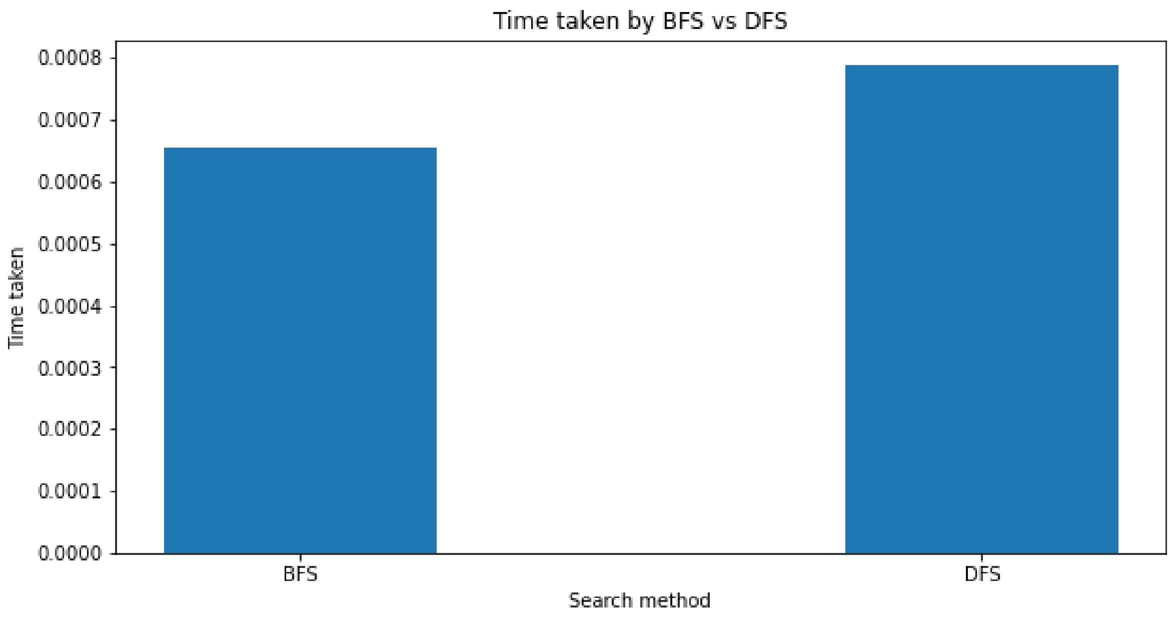
In [5]:

```
start = int(input("\nStarting node for DFS: "))

t0 = time()
DFS(visitedNodes, graphOne, start)
t1 = time()

bfstime = t1-t0
print("DFS timed at {:.3f} seconds".format(t1-t0))
print("DFS took {} bytes".format(sys.getsizeof(visitedNodes)))
```

```
Starting node for DFS: 1
1
2
3
DFS timed at 0.001 seconds
DFS took 88 bytes
```

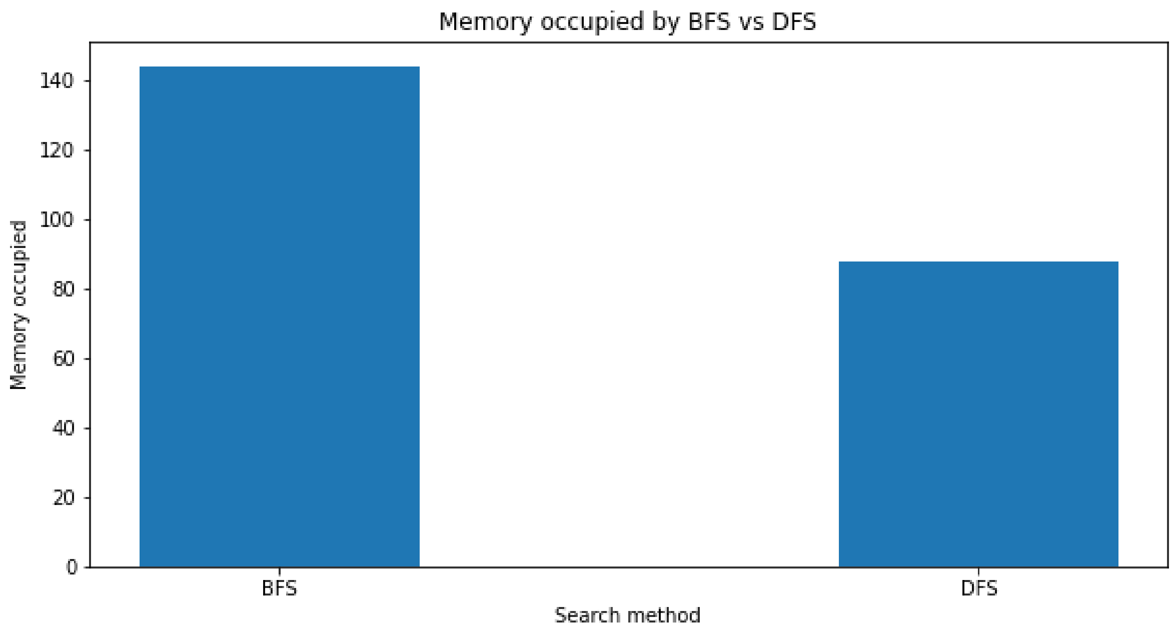
In [6]:

```
data = {'BFS': bfstime, 'DFS': dfstime}
method = list(data.keys())
values = list(data.values())

fig = plt.figure(figsize = (10, 5))

plt.bar(method, values, width = 0.4)
plt.xlabel("Search method")
plt.ylabel("Time taken")
plt.title("Time taken by BFS vs DFS")
plt.show()
```



In [7]:

```
data = {'BFS': (sys.getsizeof(visited) + sys.getsizeof(queue)), 'DFS': (sys.getsize
method = list(data.keys())
values = list(data.values())

fig = plt.figure(figsize = (10, 5))

plt.bar(method, values, width = 0.4)
plt.xlabel("Search method")
plt.ylabel("Memory occupied")
plt.title("Memory occupied by BFS vs DFS")
plt.show()
```



## Lab 2: 8 Puzzle Single Player Game (BFS)

In [1]:

```
#Import the necessary libraries
from time import time
from queue import Queue
```

In [2]:

```
#Creating a class Puzzle
class Puzzle:
    #Setting the goal state of 8-puzzle
    goal_state=[1,2,3,8,0,4,7,6,5]
    num_of_instances=0
    #constructor to initialize the class members
    def __init__(self,state,parent,action):
        self.parent=parent
        self.state=state
        self.action=action
        #TODO: incrementing the number of instance by 1
        Puzzle.num_of_instances+= 1

    #function used to display a state of 8-puzzle
    def __str__(self):
        return str(self.state[0:3])+'\n'+str(self.state[3:6])+'\n'+str(self.state[6

    #method to compare the current state with the goal state
    def goal_test(self):
        #TODO: include a condition to compare the current state with the goal state
        if self.state == Puzzle.goal_state:
            return True
        return False

    #static method to find the legal action based on the current board position
    @staticmethod
    def find_legal_actions(i,j):
        legal_action = ['U', 'D', 'L', 'R']
        if i == 0:
            # if row is 0 in board then up is disable
            legal_action.remove('U')
        elif i == 2:
            legal_action.remove('D')
        if j == 0:
            legal_action.remove('L')
        elif j == 2:
            legal_action.remove('R')

        return legal_action

    #method to generate the child of the current state of the board
    def generate_child(self):
        #TODO: create an empty list
        children=[]
        x = self.state.index(0)
        i = int(x / 3)
        j = int(x % 3)
        #TODO: call the method to find the legal actions based on i and j values
        legal_actions = Puzzle.find_legal_actions(i,j)

        #TODO:Iterate over all legal actions
        for action in legal_actions:
            new_state = self.state.copy()
            #if the legal action is UP
            if action=='U':
                #Swapping between current index of 0 with its up element on the boa
                new_state[x], new_state[x-3] = new_state[x-3], new_state[x]
            elif action=='D':
                #TODO: Swapping between current index of 0 with its down element on
                new_state[x], new_state[x+3] = new_state[x+3], new_state[x]
            elif action=='L':
                #TODO: Swapping between the current index of 0 with its left elemen
                new_state[x], new_state[x-1] = new_state[x-1], new_state[x]
            elif action=='R':
                #TODO: Swapping between the current index of 0 with its right eleme
                new_state[x], new_state[x+1] = new_state[x+1], new_state[x]
            children.append(Puzzle(new_state,self,action))
        #TODO: return the children
        return children
    #method to find the solution
    def find_solution(self):
        solution = []
        solution.append(self.action)
        path = self
        while path.parent != None:
            path = path.parent
            solution.append(path.action)
        solution = solution[:-1]
        solution.reverse()
        return solution
```

In [3]:

```
#method for breadth first search
#TODO: pass the initial_state as parameter to the breadth_first_search method
def breadth_first_search(initial_state):
    start_node = Puzzle(initial_state, None, None)
    print("Initial state:")
    print(start_node)
    if start_node.goal_test():
        return start_node.find_solution()
    q = Queue()
    #TODO: put start_node into the Queue
    q.put(start_node)
    #TODO: create an empty list of explored nodes
    explored=[]
    #TODO: Iterate the queue until empty. Use the empty() method of Queue
    while not(q.empty()):
        #TODO: get the current node of a queue. Use the get() method of Queue
        node=q.get()
        #TODO: Append the state of node in the explored list as node.state
        explored.append(node.state)
        #TODO: call the generate_child method to generate the child nodes of curren
        children=Puzzle.generate_child(node)
        #TODO: Iterate over each child node in children
        for child in children:
            if child.state not in explored:
                if child.goal_test():
                    return child.find_solution()
                q.put(child)
    return
```

In [4]:

```python
#Start executing the 8-puzzle with setting up the initial state
#Here we have considered 3 initial state intitalized using state variable
state=[[1, 3, 4,
        8, 6, 2,
        7, 0, 5],

       [2, 8, 1,
        0, 4, 3,
        7, 6, 5],

       [2, 8, 1,
        4, 6, 3,
        0, 7, 5]]
#Iterate over number of initial_state
for i in range(0,3):
    #TODO: Initialize the num_of_instances to zero
    Puzzle.num_of_instances=0
    #Set t0 to current time
    t0=time()
    bfs=breadth_first_search(state[i])
    #Get the time t1 after executing the breadth_first_search method
    t1=time()-t0
    print('BFS:', bfs)
    print('space:',Puzzle.num_of_instances)
    print('time:',t1)
    print()
print('------------------------------------------')
```

```
Initial state:
[1, 3, 4]
[8, 6, 2]
[7, 0, 5]
BFS: ['U', 'R', 'U', 'L', 'D']
space: 66
time: 0.0020859241485595703

Initial state:
[2, 8, 1]
[0, 4, 3]
[7, 6, 5]
BFS: ['U', 'R', 'R', 'D', 'L', 'L', 'U', 'R', 'D']
space: 591
time: 0.019245624542236328

Initial state:
[2, 8, 1]
[4, 6, 3]
[0, 7, 5]
BFS: ['R', 'U', 'L', 'U', 'R', 'R', 'D', 'L', 'L', 'U', 'R', 'D']
space: 2956
time: 0.06805133819580078

------------------------------------------
```

# Lab 3: 8 Puzzle Single Player Game (A* Algorithm)

In [1]:

```
from time import time
from queue import PriorityQueue
import math
```

In [2]:

```
class Puzzle:
    goal_state=[1,2,3,8,0,4,7,6,5]
    heuristic=None
    evaluation_function=None
    needs_hueristic=False
    num_of_instances=0
    def __init__(self,state,parent,action,path_cost,needs_hueristic=False):
        self.parent=parent
        self.state=state
        self.action=action
        if parent:
            self.path_cost = parent.path_cost + path_cost
        else:
            self.path_cost = path_cost
        if needs_hueristic:
            self.needs_hueristic=True
            self.generate_heuristic()
            self.evaluation_function=self.heuristic+self.path_cost
        Puzzle.num_of_instances+=1

    def __str__(self):
        return str(self.state[0:3])+'\n'+str(self.state[3:6])+'\n'+str(self.state[6

    def generate_heuristic(self):
        self.heuristic=0
        for num in range(1,9):
            distance=abs(self.state.index(num) - self.goal_state.index(num))
            i=int(distance/3)
            j=int(distance%3)
            self.heuristic=self.heuristic+i+j

    def goal_test(self):
        if self.state == self.goal_state:
            return True
        return False

    @staticmethod
    def find_legal_actions(i,j):
        legal_action = ['U', 'D', 'L', 'R']
        if i == 0:  # up is disable
            legal_action.remove('U')
        elif i == 2:  # down is disable
            legal_action.remove('D')
        if j == 0:
            legal_action.remove('L')
        elif j == 2:
            legal_action.remove('R')
        return legal_action

    def generate_child(self):
        children=[]
        x = self.state.index(0)
        i = int(x / 3)
        j = int(x % 3)
        legal_actions=self.find_legal_actions(i,j)

        for action in legal_actions:
            new_state = self.state.copy()
            if action == 'U':
                new_state[x], new_state[x-3] = new_state[x-3], new_state[x]
            elif action == 'D':
                new_state[x], new_state[x+3] = new_state[x+3], new_state[x]
            elif action == 'L':
                new_state[x], new_state[x-1] = new_state[x-1], new_state[x]
            elif action == 'R':
                new_state[x], new_state[x+1] = new_state[x+1], new_state[x]
            children.append(Puzzle(new_state,self,action,1,self.needs_hueristic))
        return children

    def find_solution(self):
        solution = []
        solution.append(self.action)
        path = self
        while path.parent != None:
            path = path.parent
            solution.append(path.action)
        solution = solution[:-1]
        solution.reverse()
        return solution
```

In [3]:

```
def Astar_search(initial_state):
    count=0
    explored=[]
    start_node=Puzzle(initial_state,None,None,0,True)
    q = PriorityQueue()
    q.put((start_node.evaluation_function,count,start_node))

    while not q.empty():
        node=q.get()
        node=node[2]
        explored.append(node.state)
        if node.goal_test():
            return node.find_solution()

        children=node.generate_child()
        for child in children:
            if child.state not in explored:
                count += 1
                q.put((child.evaluation_function,count,child))
    return
```

In [4]:

```python
#Start executing the 8-puzzle with setting up the initial state
#Here we have considered 3 initial state intitalized using state variable
state=[[1, 3, 4,
        8, 6, 2,
        7, 0, 5],

       [2, 8, 1,
        0, 4, 3,
        7, 6, 5],

       [2, 8, 1,
        4, 6, 3,
        0, 7, 5]]

for i in range(0,3):
    Puzzle.num_of_instances = 0
    t0 = time()
    astar = Astar_search(state[i])
    t1 = time() - t0
    print('A*:',astar)
    print('space:', Puzzle.num_of_instances)
    print('time:', t1)
    print()
print('------------------------------------------')
```

```
A*: ['U', 'R', 'U', 'L', 'D']
space: 16
time: 0.0009305477142333984

A*: ['U', 'R', 'R', 'D', 'L', 'L', 'U', 'R', 'D']
space: 42
time: 0.0019366741180419922

A*: ['R', 'U', 'L', 'U', 'R', 'R', 'D', 'L', 'L', 'U', 'R', 'D']
space: 95
time: 0.004247426986694336

------------------------------------------
```

# Lab 4: Water Jug Problem Using BFS & DFS

In [1]:

```
import collections
```

In [2]:

```
#This method return a key value for a given node.
#Node is a list of two integers representing current state of the jugs
def get_index(node):
    return pow(7, node[0]) * pow(5, node[1])
```

In [3]:

```
#This method accepts an input for asking the choice for type of searching required .
#Method return True for BFS, False otherwise
def get_search_type():
    s = input("Enter 'b' for BFS, 'd' for DFS: ")
    #TODO:convert the input into lowercase using lower() method
    s = s[0].lower()

    #TODO: Accept the input again if given input is not as 'b' for BFS, 'd' for DFS
    while s != 'b' and s != 'd':
        s = input("The input is not valid! Enter 'b' for BFS, 'd' for DFS: ")
        s = s[0].lower()
    #TODO: Return True for BFS option selected
    return s == 'b'
```

In [4]:

```
#This method accept volumes of the jugs as an input from the user.
#Returns a list of two integeres representing volumes of the jugs.
def get_jugs():
    print("Receiving the volume of the jugs...")
    #TODO: Create an empty list
    jugs = []

    temp = int(input("Enter first jug volume (>1): "))
    while temp < 1:
        temp = int(input("Enter a valid amount (>1): "))
    #TODO: Append the input quantity of jug into jugs list
    jugs.append(temp)

    temp = int(input("Enter second jug volume (>1): "))
    while temp < 1:
        temp = int(input("Enter a valid amount (>1): "))

    #TODO: Append the input quantity of jug into jugs list
    jugs.append(temp)

    #TODO: Return the list
    return jugs
```

In [5]:

```
#This method accepts the desired amount of water as an input from the user whereas
#the parameter jugs is a list of two integers representing volumes of the jugs
#Returns the desired amount of water as goal
def get_goal(jugs):

    print("Receiving the desired amount of the water...")

    #TODO: Find the maximum capacity of jugs using max()
    max_amount = max(jugs)
    s = "Enter the desired amount of water (1 - {0}): ".format(max_amount)
    goal_amount = int(input(s))
    #TODO: Accept the input again from the user if the bound of goal_amount is outs
    while goal_amount not in range (1, max_amount):
        goal_amount = int(input("Enter a valid amount (1 - {0}): ".format(max_amoun

    #TODO:Return the goal amount of water
    return goal_amount
```

In [6]:

```
#This method checks whether the given path matches the goal node.
#The path parameter is a list of nodes representing the path to be checked
#The goal_amount parameter is an integer representing the desired amount of water
def is_goal(path, goal_amount):

    print("Checking if the goal is achieved...")

    #TODO: Return the status of the latest path matches with the goal_amount of ano
    return path[-1][0] == goal_amount or path[-1][1] == goal_amount
```

In [7]:

```
#This method validates whether the given node is already visited.
#The parameter node is a list of two integers representing current state of the jug
#The parameter check_dict is   a dictionary storing visited nodes
def been_there(node, check_dict):

    print("Checking if {0} is visited before...".format(node))

    #TODO: Return True whether a given node already exisiting in a dictionary, othe
    return check_dict.get(node[0]) != None
```

In [8]:

```
#This method returns the list of all possible transitions
#The parameter jugs is a list of two integers representing volumes of the jugs
#The parameter path is a list of nodes represeting the current path
#The parameter check_dict is a dictionary storing visited nodes
def next_transitions(jugs, path, check_dict):

    print("Finding next transitions and checking for the loops...")

    #TODO: create an empty list
    result = []
    next_nodes = []
    node = []

    a_max = jugs[0]
    b_max = jugs[1]

    #TODO: initial amount in the first jug using path parameter
    a = path[-1][0]
    #TODO: initial amount in the second jug using path parameter
    b = path[-1][1]

    #Operation Used in Water Jug problem
    # 1. fill in the first jug
    node.append(a_max)
    node.append(b)
    if not been_there(node, check_dict):
        next_nodes.append(node)
    node = []

    # 2. fill in the second jug
    #TODO: Append with the initial amount of water in first jug
    node.append(a)
    #TODO: Append with the max amount of water in second jug
    node.append(b_max)
    #TODO: Check if node is not visited then append the node in next_nodes. Use bee
    if not been_there(node, check_dict):
        next_nodes.append(node)
    node = []

    # 3. second jug to first jug
    node.append(min(a_max, a + b))
    node.append(b - (node[0] - a))  # b - ( a' - a)
    if not been_there(node, check_dict):
        next_nodes.append(node)
    node = []

    # 4. first jug to second jug
    #TODO: Append the minimum between the a+b and b_max
    node.append(min(a+b, b_max))
    node.insert(0, a - (node[0] - b))
    if not been_there(node, check_dict):
        next_nodes.append(node)
    node = []

    # 5. empty first jug
    #TODO: Append 0 to empty first jug
    node.append(0)
    #TODO:Append b amount for second jug
    node.append(b)
    if not been_there(node, check_dict):
        next_nodes.append(node)
    node = []

    # 6. empty second jug
    #TODO:Append a amount for first jug
    node.append(a)
    #TODO: Append 0 to empty second jug
    node.append(0)
    if not been_there(node, check_dict):
        next_nodes.append(node)

    # create a list of next paths
    for i in range(0, len(next_nodes)):
        temp = list(path)
        #TODO: Append the ith index of next_nodes to temp
        temp.append(next_nodes[i])
        result.append(temp)

    if len(next_nodes) == 0:
        print("No more unvisited nodes...\nBacktracking...")
    else:
        print("Possible transitions: ")
        for nnode in next_nodes:
            print(nnode)
    #TODO: return result
    return result
```

In [9]:

```
# This method returns a string explaining the transition from old state/node to new
# The parameter old is a list representing old state/node
# The parameter new is a list representing new state/node
# The parameter jugs is a list of two integers representing volumes of the jugs

def transition(old, new, jugs):

    #TODO: Get the amount of water from old state/node for first Jug
    a = old[0]
    #TODO: Get the amount of water from old state/node for second Jug
    b = old[1]
    #TODO: Get the amount of water from new state/node for first Jug
    a_prime = new[0]
    #TODO: Get the amount of water from new state/node for second Jug
    b_prime = new[1]
    #TODO: Get the amount of water from jugs representing volume for first Jug
    a_max = jugs[0]
    #TODO: Get the amount of water from jugs representing volume for second Jug
    b_max = jugs[1]

    if a > a_prime:
        if b == b_prime:
            return "Clear {0}-liter jug:\t\t\t".format(a_max)
        else:
            return "Pour {0}-liter jug into {1}-liter jug:\t".format(a_max, b_max)
    else:
        if b > b_prime:
            if a == a_prime:
                return "Clear {0}-liter jug:\t\t\t".format(b_max)
            else:
                return "Pour {0}-liter jug into {1}-liter jug:\t".format(b_max, a_m
        else:
            if a == a_prime:
                return "Fill {0}-liter jug:\t\t\t".format(b_max)
            else:
                return "Fill {0}-liter jug:\t\t\t".format(a_max)
```

In [10]:

```
#This method prints the goal path
#The path is a list of nodes representing the goal path
#The jugs is a list of two integers representing volumes of the jugs

def print_path(path, jugs):

    print("Starting from:\t\t\t\t", path[0])
    for i in  range(0, len(path) - 1):
        print(i+1,":", transition(path[i], path[i+1], jugs), path[i+1])
```

In [11]:

```
#This method searches for a path between starting node and goal node
# The parameter starting_node is a list of list of two integers representing initia
#The parameter jugs a list of two integers representing volumes of the jugs
#The parameter goal_amount is an integer represting the desired amount
#The parameter check_dict is  a dictionary storing visited nodes
#The parameter is_breadth is implements BFS, if True; DFS otherwise
def search(starting_node, jugs, goal_amount, check_dict, is_breadth):

    if is_breadth:
        print("Implementing BFS...")
    else:
        print("Implementing DFS...")

    goal = []
    #TODO: SET accomplished to be False
    accomplished = False

    #TODO: Call a deque() using collections
    q = collections.deque()
    q.appendleft(starting_node)

    while len(q) != 0:
        path = q.popleft()
        check_dict[get_index(path[-1])] = True
        if len(path) >= 2:
            print(transition(path[-2], path[-1], jugs), path[-1])
        if is_goal(path, goal_amount):
            #TODO: Set accomplished to be True
            accomplished = True
            goal = path
            break

        #TODO: Call next_transitions method for generating the further nodes
        next_moves = next_transitions(jugs, path, check_dict)
        #TODO: Iterate over the next_moves list
        for i in next_moves:
            if is_breadth:
                q.append(i)
            else:
                q.appendleft(i)

    if accomplished:
        print("The goal is achieved\nPrinting the sequence of the moves...\n")
        print_path(goal, jugs)
    else:
        print("Problem cannot be solved.")
```

In [12]:

```
if __name__ == '__main__':
    starting_node = [[0, 0]]
    #TODO: Call the get_jugs() method
    jugs = get_jugs()
    #TODO: Call the get_goal() method
    goal_amount = get_goal(jugs)
    #TODO: Create an empty dictionary
    check_dict = {}
    #TODO: call the get_search_type() method
    is_breadth = get_search_type()
    #TODO: Call the search method with the required parameters
    search(starting_node, jugs, goal_amount, check_dict, is_breadth)
```

```
Receiving the volume of the jugs...
Enter first jug volume (>1): 2
Enter second jug volume (>1): 3
Receiving the desired amount of the water...
Enter the desired amount of water (1 - 3): 2
Enter 'b' for BFS, 'd' for DFS: b
Implementing BFS...
Checking if the goal is achieved...
Finding next transitions and checking for the loops...
Checking if [2, 0] is visited before...
Checking if [0, 3] is visited before...
Checking if [0, 0] is visited before...
Checking if [0, 0] is visited before...
Checking if [0, 0] is visited before...
Checking if [0, 0] is visited before...
Possible transitions:
[2, 0]
[0, 3]
[0, 0]
[0, 0]
[0, 0]
[0, 0]
Fill 2-liter jug:                       [2, 0]
Checking if the goal is achieved...
The goal is achieved
Printing the sequence of the moves...

Starting from:                          [0, 0]
1 : Fill 2-liter jug:                   [2, 0]
```

# Lab 5: Graph Coloring Problem

In [1]:

```
from typing import Generic, TypeVar, Dict, List, Optional
from abc import ABC, abstractmethod
```

In [2]:

```
V = TypeVar('V') # variable type
D = TypeVar('D') # domain type
```

In [3]:

```
# Base class for all constraints
class Constraint(Generic[V, D], ABC):
    # The variables that the constraint is between
    def __init__(self, variables: List[V]) -> None:
        self.variables = variables

    # Must be overridden by subclasses
    @abstractmethod
    def satisfied(self, assignment: Dict[V, D]) -> bool:
        ...
```

In [4]:

```
# A constraint satisfaction problem consists of variables of type V
# that have ranges of values known as domains of type D and constraints
# that determine whether a particular variable's domain selection is valid
class CSP(Generic[V, D]):
    def __init__(self, variables: List[V], domains: Dict[V, List[D]]) -> None:
        self.variables: List[V] = variables # variables to be constrained
        self.domains: Dict[V, List[D]] = domains # domain of each variable
        self.constraints: Dict[V, List[Constraint[V, D]]] = {}
        for variable in self.variables:
            self.constraints[variable] = []
            if variable not in self.domains:
                raise LookupError("Every variable should have a domain assigned to 

    def add_constraint(self, constraint: Constraint[V, D]) -> None:
        for variable in constraint.variables:
            if variable not in self.variables:
                raise LookupError("Variable in constraint not in CSP")
            else:
                self.constraints[variable].append(constraint)

    # Check if the value assignment is consistent by checking all constraints
    # for the given variable against it
    def consistent(self, variable: V, assignment: Dict[V, D]) -> bool:
        for constraint in self.constraints[variable]:
            if not constraint.satisfied(assignment):
                return False
        return True

    def backtracking_search(self, assignment: Dict[V, D] = {}) -> Optional[Dict[V, 
        # assignment is complete if every variable is assigned (our base case)
        if len(assignment) == len(self.variables):
            return assignment

        # get all variables in the CSP but not in the assignment
        unassigned: List[V] = [v for v in self.variables if v not in assignment]

        # get the every possible domain value of the first unassigned variable
        first: V = unassigned[0]
        for value in self.domains[first]:
            local_assignment = assignment.copy()
            local_assignment[first] = value
            # if we're still consistent, we recurse (continue)
            if self.consistent(first, local_assignment):
                result: Optional[Dict[V, D]] = self.backtracking_search(local_assig
                # if we didn't find the result, we will end up backtracking
                if result is not None:
                    return result
        return None
```

In [5]:

```python
class MapColoringConstraint(Constraint[str, str]):
    def __init__(self, place1: str, place2: str) -> None:
        super().__init__([place1, place2])
        self.place1: str = place1
        self.place2: str = place2

    def satisfied(self, assignment: Dict[str, str]) -> bool:
        # If either place is not in the assignment then it is not
        # yet possible for their colors to be conflicting
        if self.place1 not in assignment or self.place2 not in assignment:
            return True
        # check the color assigned to place1 is not the same as the
        # color assigned to place2
        return assignment[self.place1] != assignment[self.place2]
```

In [6]:

```python
if __name__ == "__main__":
    variables: List[str] = ["BOX_1", "BOX_2", "BOX_4",
                            "BOX_3", "BOX_5", "BOX_6", "BOX_7"]
    domains: Dict[str, List[str]] = {}
    for variable in variables:
        domains[variable] = ["red", "green", "blue"]
    csp: CSP[str, str] = CSP(variables, domains)
    csp.add_constraint(MapColoringConstraint("BOX_1", "BOX_2"))
    csp.add_constraint(MapColoringConstraint("BOX_1", "BOX_4"))
    csp.add_constraint(MapColoringConstraint("BOX_4", "BOX_2"))
    csp.add_constraint(MapColoringConstraint("BOX_3", "BOX_2"))
    csp.add_constraint(MapColoringConstraint("BOX_3", "BOX_4"))
    csp.add_constraint(MapColoringConstraint("BOX_3", "BOX_5"))
    csp.add_constraint(MapColoringConstraint("BOX_5", "BOX_4"))
    csp.add_constraint(MapColoringConstraint("BOX_6", "BOX_4"))
    csp.add_constraint(MapColoringConstraint("BOX_6", "BOX_5"))
    csp.add_constraint(MapColoringConstraint("BOX_6", "BOX_7"))
    csp.add_constraint(MapColoringConstraint("BOX_7", "BOX_1"))
    solution: Optional[Dict[str, str]] = csp.backtracking_search()
    if solution is None:
        print("No solution found!")
    else:
        print(solution)
```

```
{'BOX_1': 'red', 'BOX_2': 'green', 'BOX_4': 'blue', 'BOX_3': 'red', 'B
OX_5': 'green', 'BOX_6': 'red', 'BOX_7': 'green'}
```

In [7]:

```python
class SendMoreMoneyConstraint(Constraint[str, int]):
    def __init__(self, letters: List[str]) -> None:
        super().__init__(letters)
        self.letters: List[str] = letters

    def satisfied(self, assignment: Dict[str, int]) -> bool:
        # if there are duplicate values then it's not a solution
        if len(set(assignment.values())) < len(assignment):
            return False

        # if all variables have been assigned, check if it adds correctly
        if len(assignment) == len(self.letters):
            s: int = assignment["S"]
            e: int = assignment["E"]
            n: int = assignment["N"]
            d: int = assignment["D"]
            m: int = assignment["M"]
            o: int = assignment["O"]
            r: int = assignment["R"]
            y: int = assignment["Y"]
            send: int = s * 1000 + e * 100 + n * 10 + d
            more: int = m * 1000 + o * 100 + r * 10 + e
            money: int = m * 10000 + o * 1000 + n * 100 + e * 10 + y
            return send + more == money
        return True # no conflict
```

In [8]:

```python
if __name__ == "__main__":
    letters: List[str] = ["S", "E", "N", "D", "M", "O", "R", "Y"]
    possible_digits: Dict[str, List[int]] = {}
    for letter in letters:
        possible_digits[letter] = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
    possible_digits["M"] = [1]  # so we don't get answers starting with a 0
    csp: CSP[str, int] = CSP(letters, possible_digits)
    csp.add_constraint(SendMoreMoneyConstraint(letters))
    solution: Optional[Dict[str, int]] = csp.backtracking_search()
    if solution is None:
        print("No solution found!")
    else:
        print(solution)
```

```
{'S': 9, 'E': 5, 'N': 6, 'D': 7, 'M': 1, 'O': 0, 'R': 8, 'Y': 2}
```

# Lab 6: Implement a Knapsack problem using Brute Force Method and Dynamic Programming

In [1]:

```
from itertools import product
from collections import namedtuple
try:
    from itertools import izip
except ImportError:
    izip = zip
```

In [2]:

```
Reward = namedtuple('Reward', 'name value weight volume')

bagpack =   Reward('bagpack',  0, 25.0, 0.25)

items = [Reward('laptop', 3000,  0.3, 0.025),
         Reward('printer',   1800,  0.2, 0.015),
         Reward('headphone',    2500,  2.0, 0.002)]
```

In [3]:

```
def tot_value(items_count):
    """
    Given the count of each item in the sack return -1 if they can't be carried or 

    (also return the negative of the weight and the volume so taking the max of a s
    values will minimise the weight if values tie, and minimise the volume if value
    """
    global items, bagpack
    weight = sum(n * item.weight for n, item in izip(items_count, items))
    volume = sum(n * item.volume for n, item in izip(items_count, items))
    if weight <= bagpack.weight and volume <= bagpack.volume:
        return sum(n * item.value for n, item in izip(items_count, items)), -weight
    else:
        return -1, 0, 0
```

In [4]:

```
def knapsack():
    global items, bagpack
    # find max of any one item
    max1 = [min(int(bagpack.weight // item.weight), int(bagpack.volume // item.volu

    # Try all combinations of reward items from 0 up to max1
    return max(product(*[range(n + 1) for n in max1]), key=tot_value)
```

In [5]:

```
max_items = knapsack()
maxvalue, max_weight, max_volume = tot_value(max_items)
max_weight = -max_weight
max_volume = -max_volume

print("The maximum value achievable (by exhaustive search) is %g." % maxvalue)
item_names = ", ".join(item.name for item in items)
print("  The number of %s items to achieve this is: %s, respectively." % (item_name
print("  The weight to carry is %.3g, and the volume used is %.3g." % (max_weight,
```

```
The maximum value achievable (by exhaustive search) is 54500.
  The number of laptop, printer, headphone items to achieve this is:
(9, 0, 11), respectively.
  The weight to carry is 24.7, and the volume used is 0.247.
```

## Implement a Knapsack problem using Dynamic Programming. Compare the execution time of brute-force and dynamic programming algorithms

Instead, use a technique known as dynamic programming, which is similar in concept to memoization. Instead of solving a problem outright with a brute-force approach, in dynamic programming one solves subproblems that make up the larger problem, stores those results, and utilizes those stored results to solve the larger problem. As long as the capacity of the knapsack is considered in discrete steps, the problem can be solved with dynamic programming.

In [6]:

```
def knapSack(W, wt, val):
    n=len(val)
    table = [[0 for x in range(W + 1)] for x in range(n + 1)]

    for i in range(n + 1):
        for j in range(W + 1):
            if i == 0 or j == 0:
                table[i][j] = 0
            elif wt[i-1] <= j:
                table[i][j] = max(val[i-1] + table[i-1][j-wt[i-1]],  table[i-1][j])
            else:
                table[i][j] = table[i-1][j]

    return table[n][W]
```

In [7]:

```
val = [50,100,150,200]
wt = [8,16,32,40]
W = 64

print(knapSack(W, wt, val))
```

```
350
```

# Lab 7: Preprocessing Techniques in NLP Using NLTK package

In [1]:

```
#Import the necessary libraries
import nltk                                    # Python library for NLP
from nltk.corpus import twitter_samples        # sample Twitter dataset from NLTK
import matplotlib.pyplot as plt                # library for visualization
import random                                  # pseudo-random number generator
import numpy as np
```

In [2]:

```
# downloads sample twitter dataset. execute the line below if running on a local mac
nltk.download('twitter_samples')
```

```
[nltk_data] Downloading package twitter_samples to
[nltk_data]     /home/yash/nltk_data...
[nltk_data]   Package twitter_samples is already up-to-date!
```

Out[2]:

```
True
```

We can load the text fields of the positive and negative tweets by using the module's `strings()` method like this:

In [3]:

```
# select the set of positive and negative tweets
all_positive_tweets = twitter_samples.strings('positive_tweets.json')
all_negative_tweets = twitter_samples.strings('negative_tweets.json')
```

Next, we'll print a report with the number of positive and negative tweets. It is also essential to know the data structure of the datasets

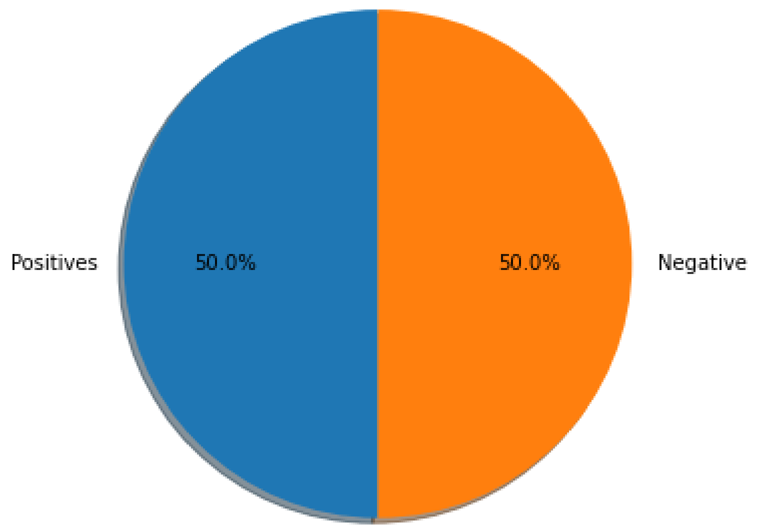
In [4]:

```
#TODO: Print the size of positive and negative tweets
print('Number of positive tweets: ', len(all_positive_tweets))
print('Number of negative tweets: ', len(all_negative_tweets))

#TODO:Print the type of positive and negative tweets, using type()
print('\nThe type of all_positive_tweets is: ', type(all_positive_tweets))
print('The type of a tweet entry is: ', type(all_negative_tweets))
```

```
Number of positive tweets:  5000
Number of negative tweets:  5000

The type of all_positive_tweets is:  <class 'list'>
The type of a tweet entry is:  <class 'list'>
```

In [5]:

```
#PLOT the positive and negative tweets in a pie-chart
# Declare a figure with a custom size
fig = plt.figure(figsize=(5, 5))

# labels for the two classes
labels = 'Positives', 'Negative'

# Sizes for each slide
sizes = [len(all_positive_tweets), len(all_negative_tweets)]

# Declare pie chart, where the slices will be ordered and plotted counter-clockwise
plt.pie(sizes, labels=labels, autopct='%1.1f%%',
        shadow=True, startangle=90)

# Equal aspect ratio ensures that pie is drawn as a circle.
plt.axis('equal')

# Display the chart
plt.show()
```



## Looking at raw texts

Before anything else, we can print a couple of tweets from the dataset to see how they look. Understanding the data is responsible for 80% of the success or failure in data science projects. We can use this time to observe aspects we'd like to consider when preprocessing our data.

Below, you will print one random positive and one random negative tweet. We have added a color mark at the beginning of the string to further distinguish the two.

In [6]:

```
#TODO: Display a random tweet from positive and negative tweet.
#The size of positive and negative tweets are 5000 each.
#Generate a random number between 0 and 5000 using random.randint()
# print positive in greeen
print('\033[92m' + all_positive_tweets[420])

# print negative in red
print('\033[91m' + all_negative_tweets[69])
```

```
@applewriter yep, you *are* an expert on bisexuality :) Good luck!
@IzzyTailford that's true, I just want it sooooooooner :(
```

## Preprocess raw text for Sentiment analysis

In [7]:

```
# Our selected sample. Complex enough to exemplify each step
tweet = all_positive_tweets[2277]
print(tweet)
```

```
My beautiful sunflowers on a sunny Friday morning off :) #sunflowers #
favourites #happy #Friday off… https://t.co/3tfYom0N1i (https://t.co/3
tfYom0N1i)
```

In [8]:

```
# download the stopwords from NLTK
nltk.download('stopwords')
```

```
[nltk_data] Downloading package stopwords to /home/yash/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
```

Out[8]:

```
True
```

In [9]:

```
#Import the necessary libraries
import re                                  # library for regular expression operati
import string                              # for string operations

from nltk.corpus import stopwords          # module for stop words that come with N
from nltk.stem import PorterStemmer        # module for stemming
from nltk.tokenize import TweetTokenizer   # module for tokenizing strings
```

### Remove hyperlinks, Twitter marks and styles

Since we have a Twitter dataset, we'd like to remove some substrings commonly used on the platform like the hashtag, retweet marks, and hyperlinks. We'll use the re (https://docs.python.org/3/library/re.html) library to perform regular expression operations on our tweet. We'll define our search pattern and use the `sub()` method to remove matches by substituting with an empty character (i.e. `''` )

In [10]:

```
print('\033[92m' + tweet)
print('\033[94m')

# remove old style retweet text "RT"
tweet2 = re.sub(r'^RT[\s]+', '', tweet)

# remove hyperlinks
tweet2 = re.sub(r'https?:\/\/.*[\r\n]*', '', tweet2)

# remove hashtags
# only removing the hash # sign from the word
tweet2 = re.sub(r'#', '', tweet2)

print(tweet2)
```

```
My beautiful sunflowers on a sunny Friday morning off :) #sunflowers #
favourites #happy #Friday off… https://t.co/3tfYom0N1i (https://t.co/3
tfYom0N1i)

My beautiful sunflowers on a sunny Friday morning off :) sunflowers fa
vourites happy Friday off…
```

## Tokenize the string

To tokenize means to split the strings into individual words without blanks or tabs. In this same step, we will also convert each word in the string to lower case. The tokenize (https://www.nltk.org/api/nltk.tokenize.html#module-nltk.tokenize.casual) module from NLTK allows us to do these easily:

In [11]:

```
print()
print('\033[92m' + tweet2)
print('\033[94m')

# instantiate tokenizer class
tokenizer = TweetTokenizer(preserve_case=False, strip_handles=True,
                               reduce_len=True)

# tokenize tweets
tweet_tokens = tokenizer.tokenize(tweet2)

print()
print('Tokenized string:')
print(tweet_tokens)
```

```

My beautiful sunflowers on a sunny Friday morning off :) sunflowers fa
vourites happy Friday off…


Tokenized string:
['my', 'beautiful', 'sunflowers', 'on', 'a', 'sunny', 'friday', 'morni
ng', 'off', ':)', 'sunflowers', 'favourites', 'happy', 'friday', 'of
f', '…']
```

## Remove stop words and punctuations

The next step is to remove stop words and punctuation. Stop words are words that don't add significant meaning to the text. You'll see the list provided by NLTK when you run the cells below.

In [12]:

```
#Import the english stop words list from NLTK
stopwords_english = stopwords.words('english')

print('Stop words\n')
print(stopwords_english)

print('\nPunctuation\n')
print(string.punctuation)
```

```
Stop words

['i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselves', 'you',
"you're", "you've", "you'll", "you'd", 'your', 'yours', 'yourself', 'y
ourselves', 'he', 'him', 'his', 'himself', 'she', "she's", 'her', 'her
s', 'herself', 'it', "it's", 'its', 'itself', 'they', 'them', 'their',
'theirs', 'themselves', 'what', 'which', 'who', 'whom', 'this', 'tha
t', "that'll", 'these', 'those', 'am', 'is', 'are', 'was', 'were', 'b
e', 'been', 'being', 'have', 'has', 'had', 'having', 'do', 'does', 'di
d', 'doing', 'a', 'an', 'the', 'and', 'but', 'if', 'or', 'because', 'a
s', 'until', 'while', 'of', 'at', 'by', 'for', 'with', 'about', 'again
st', 'between', 'into', 'through', 'during', 'before', 'after', 'abov
e', 'below', 'to', 'from', 'up', 'down', 'in', 'out', 'on', 'off', 'ov
er', 'under', 'again', 'further', 'then', 'once', 'here', 'there', 'wh
en', 'where', 'why', 'how', 'all', 'any', 'both', 'each', 'few', 'mor
e', 'most', 'other', 'some', 'such', 'no', 'nor', 'not', 'only', 'ow
n', 'same', 'so', 'than', 'too', 'very', 's', 't', 'can', 'will', 'jus
t', 'don', "don't", 'should', "should've", 'now', 'd', 'll', 'm', 'o',
're', 've', 'y', 'ain', 'aren', "aren't", 'couldn', "couldn't", 'did
n', "didn't", 'doesn', "doesn't", 'hadn', "hadn't", 'hasn', "hasn't",
'haven', "haven't", 'isn', "isn't", 'ma', 'mightn', "mightn't", 'must
n', "mustn't", 'needn', "needn't", 'shan', "shan't", 'shouldn', "shoul
dn't", 'wasn', "wasn't", 'weren', "weren't", 'won', "won't", 'wouldn',
"wouldn't"]

Punctuation

!"#$%&'()*+,-./:;<=>?@[\]^_`{|}~
```

In [13]:

```
print()
print('\033[92m')
print(tweet_tokens)
print('\033[94m')

#TODO: Create the empty list to store the clean tweets after removing stopwords and
tweets_clean = []

#TODO: Remove stopwords and punctuation from the tweet_tokens
for word in tweet_tokens: # Go through every word in your tokens list
    if (word not in stopwords_english and  # remove stopwords
        word not in string.punctuation):  # remove punctuation
        #TODO: Append the clean word in the tweets_clean list
        tweets_clean.append(word)

print('removed stop words and punctuation:')
print(tweets_clean)
```

```

['my', 'beautiful', 'sunflowers', 'on', 'a', 'sunny', 'friday', 'morni
ng', 'off', ':)', 'sunflowers', 'favourites', 'happy', 'friday', 'of
f', '…']

removed stop words and punctuation:
['beautiful', 'sunflowers', 'sunny', 'friday', 'morning', ':)', 'sunfl
owers', 'favourites', 'happy', 'friday', '…']
```

## Stemming

Stemming is the process of converting a word to its most general form, or stem. This helps in reducing the size of our vocabulary.

Consider the words:

- **learn**
- **learn**ing
- **learn**ed
- **learn**t

All these words are stemmed from its common root **learn**. However, in some cases, the stemming process produces words that are not correct spellings of the root word. For example, **happi** and **sunni**. That's because it chooses the most common stem for related words. For example, we can look at the set of words that comprises the different forms of happy:

- **happ**y
- **happi**ness
- **happi**er

We can see that the prefix **happi** is more commonly used. We cannot choose **happ** because it is the stem of unrelated words like **happen**.

NLTK has different modules for stemming and we will be using the PorterStemmer (https://www.nltk.org/api/nltk.stem.html#module-nltk.stem.porter) module which uses the Porter Stemming Algorithm (https://tartarus.org/martin/PorterStemmer/). Let's see how we can use it in the cell below.

In [14]:

```
print()
print('\033[92m')
print(tweets_clean)
print('\033[94m')

# Instantiate stemming class
stemmer = PorterStemmer()

#TODO: Create an empty list to store the stems
tweets_stem = []

#TODO: Itearate over the tweets_clean fot stemming
for word in tweets_clean:
    #TODO:call the stem function for stemming the word
    stem_word = stemmer.stem(word)               # stemming word
    #TODO:Append the stem_word in tweets_stem list
    tweets_stem.append(stem_word)

print('stemmed words:')
print(tweets_stem)
```

```
['beautiful', 'sunflowers', 'sunny', 'friday', 'morning', ':)', 'sunfl
owers', 'favourites', 'happy', 'friday', '…']

stemmed words:
['beauti', 'sunflow', 'sunni', 'friday', 'morn', ':)', 'sunflow', 'fav
ourit', 'happi', 'friday', '…']
```

## process_tweet()

As shown above, preprocessing consists of multiple steps before you arrive at the final list of words. We will not ask you to replicate these however. You will use the function `process_tweet(tweet)` available below.

To obtain the same result as in the previous code cells, you will only need to call the function `process_tweet()`. Let's do that in the next cell.

In [15]:

```
def process_tweet(tweet):
    """Process tweet function.
    Input:
        tweet: a string containing a tweet
    Output:
        tweets_clean: a list of words containing the processed tweet

    """
    stemmer = PorterStemmer()
    stopwords_english = stopwords.words('english')
    # remove stock market tickers like $GE
    tweet = re.sub(r'\$\w*', '', tweet)
    # remove old style retweet text "RT"
    tweet = re.sub(r'^RT[\s]+', '', tweet)
    # remove hyperlinks
    tweet = re.sub(r'https?:\/\/.*[\r\n]*', '', tweet)
    # remove hashtags
    # only removing the hash # sign from the word
    tweet = re.sub(r'#', '', tweet)
    # tokenize tweets
    tokenizer = TweetTokenizer(preserve_case=False, strip_handles=True,
                               reduce_len=True)
    tweet_tokens = tokenizer.tokenize(tweet)

    tweets_clean = []
    for word in tweet_tokens:
        if (word not in stopwords_english and  # remove stopwords
                word not in string.punctuation):  # remove punctuation
            # tweets_clean.append(word)
            stem_word = stemmer.stem(word)  # stemming word
            tweets_clean.append(stem_word)

    return tweets_clean
```

In [16]:

```
# choose the same tweet
tweet = all_positive_tweets[2277]

print()
print('\033[92m')
print(tweet)
print('\033[94m')

#TODO: call the process_tweet function
tweets_stem = process_tweet(tweet)

print('preprocessed tweet:')
print(tweets_stem) # Print the result
```

```
My beautiful sunflowers on a sunny Friday morning off :) #sunflowers #
favourites #happy #Friday off… https://t.co/3tfYom0N1i (https://t.co/3
tfYom0N1i)

preprocessed tweet:
['beauti', 'sunflow', 'sunni', 'friday', 'morn', ':)', 'sunflow', 'fav
ourit', 'happi', 'friday', '…']
```

## Building and Visualizing word frequencies

In this lab, we will focus on the `build_freqs()` helper function and visualizing a dataset fed into it. In our goal of tweet sentiment analysis, this function will build a dictionary where we can lookup how many times a word appears in the lists of positive or negative tweets. This will be very helpful when extracting the features of the dataset in the week's programming assignment. Let's see how this function is implemented under the hood in this notebook.

In [17]:

```
#TODO: Concatenate the lists, 1st part is the positive tweets followed by the negat
tweets = all_positive_tweets + all_negative_tweets

# let's see how many tweets we have
print("Number of tweets: ", len(tweets))
```

```
Number of tweets:  10000
```

In [18]:

```
# make a numpy array representing labels of the tweets
labels = np.append(np.ones((len(all_positive_tweets))), np.zeros((len(all_negative_
```

### Dictionaries

In Python, a dictionary is a mutable and indexed collection. It stores items as key-value pairs and uses [hash tables (https://en.wikipedia.org/wiki/Hash_table)](https://en.wikipedia.org/wiki/Hash_table) underneath to allow practically constant time lookups. In NLP, dictionaries are essential because it enables fast retrieval of items or containment checks even with thousands of entries in the collection.

## Definition

A dictionary in Python is declared using curly brackets. Look at the next example:

In [19]:

```
dictionary = {'key1': 1, 'key2': 2}
```

The former line defines a dictionary with two entries. Keys and values can be almost any type (with a few restriction on keys (https://docs.python.org/3/tutorial/datastructures.html#dictionaries)), and in this case, we used strings. We can also use floats, integers, tuples, etc.

## Adding or editing entries

New entries can be inserted into dictionaries using square brackets. If the dictionary already contains the specified key, its value is overwritten.

In [20]:

```
# Add a new entry
dictionary['key3'] = -5

# Overwrite the value of key1
dictionary['key1'] = 0

print(dictionary)
```

```
{'key1': 0, 'key2': 2, 'key3': -5}
```

## Accessing values and lookup keys

Performing dictionary lookups and retrieval are common tasks in NLP. There are two ways to do this:

- Using square bracket notation: This form is allowed if the lookup key is in the dictionary. It produces an error otherwise.
- Using the get() (https://docs.python.org/3/library/stdtypes.html#dict.get) method: This allows us to set a default value if the dictionary key does not exist.

Let us see these in action:

In [21]:

```
# Square bracket lookup when the key exist
print(dictionary['key2'])
```

```
2
```

However, if the key is missing, the operation produce an erro

In [22]:

```
# The output of this line is intended to produce a KeyError
print(dictionary['key8'])
```

```
----------------------------------------------------------------------
-----
KeyError                                  Traceback (most recent call
 last)
<ipython-input-22-8d63520997fb> in <module>
      1 # The output of this line is intended to produce a KeyError
----> 2 print(dictionary['key8'])

KeyError: 'key8'
```

When using a square bracket lookup, it is common to use an if-else block to check for containment first (with the keyword `in`) before getting the item. On the other hand, you can use the `.get()` method if you want to set a default value when the key is not found. Let's compare these in the cells below:

## This prints a value

if 'key1' in dictionary: print("item found: ", dictionary['key1']) else: print('key1 is not defined')

## Same as what you get with get

print("item found: ", dictionary.get('key1', -1))

In [23]:

```
# This prints a message because the key is not found
if 'key7' in dictionary:
    print(dictionary['key7'])
else:
    print('key does not exist!')

# This prints -1 because the key is not found and we set the default to -1
print(dictionary.get('key7', -1))
```

```
key does not exist!
-1
```

### Word frequency dictionary

In [24]:

```
def build_freqs(tweets, ys):
    """Build frequencies.
    Input:
        tweets: a list of tweets
        ys: an m x 1 array with the sentiment label of each tweet
            (either 0 or 1)
    Output:
        freqs: a dictionary mapping each (word, sentiment) pair to its
        frequency
    """
    # Convert np array to list since zip needs an iterable.
    # The squeeze is necessary or the list ends up with one element.
    # Also note that this is just a NOP if ys is already a list.
    yslist = np.squeeze(ys).tolist()

    # Start with an empty dictionary and populate it by looping over all tweets
    # and over all processed words in each tweet.
    #TODO:Create the empty dictionary
    freqs = {}
    for y, tweet in zip(yslist, tweets):
        #TODO: Iterate over all the words returned by calling the process_tweet() fu
        for word in process_tweet(tweet):
            pair = (word, y)
            #TODO: If pair matches in the dictionary, then increment the count of co
            if pair in freqs:
                freqs[pair] += 1
            else:
                #TODO: If pair does not matches in the dictionary, then set the coun
                freqs[pair] = 1

    #TODO: Return the dictionary
    return freqs
```

In [25]:

```python
#TODO: Call the build_freqs function to create frequency dictionary based on tweets
freqs = build_freqs(tweets, labels)

#TODO: Display the data type of freqs
print(f'type(freqs) = {freqs}')

#TODO: Display the length of the dictionary
print(f'len(freqs) = {freqs}')
```

```
type(freqs) = {('followfriday', 1.0): 25, ('top', 1.0): 32, ('enga
g', 1.0): 7, ('member', 1.0): 16, ('commun', 1.0): 33, ('week', 1.
0): 83, (':)', 1.0): 3568, ('hey', 1.0): 76, ('jame', 1.0): 7, ('od
d', 1.0): 2, (':/', 1.0): 5, ('pleas', 1.0): 97, ('call', 1.0): 37,
('contact', 1.0): 7, ('centr', 1.0): 2, ('02392441234', 1.0): 1, ('a
bl', 1.0): 8, ('assist', 1.0): 1, ('mani', 1.0): 33, ('thank', 1.0):
620, ('listen', 1.0): 16, ('last', 1.0): 47, ('night', 1.0): 68, ('b
leed', 1.0): 2, ('amaz', 1.0): 51, ('track', 1.0): 5, ('scotland',
1.0): 2, ('congrat', 1.0): 21, ('yeaaah', 1.0): 1, ('yipppi', 1.0):
1, ('accnt', 1.0): 2, ('verifi', 1.0): 2, ('rqst', 1.0): 1, ('succee
d', 1.0): 1, ('got', 1.0): 69, ('blue', 1.0): 9, ('tick', 1.0): 1,
('mark', 1.0): 1, ('fb', 1.0): 6, ('profil', 1.0): 2, ('15', 1.0):
5, ('day', 1.0): 246, ('one', 1.0): 129, ('irresist', 1.0): 2, ('fli
pkartfashionfriday', 1.0): 17, ('like', 1.0): 233, ('keep', 1.0): 6
8, ('love', 1.0): 400, ('custom', 1.0): 4, ('wait', 1.0): 70, ('lon
g', 1.0): 36, ('hope', 1.0): 141, ('enjoy', 1.0): 75, ('happi', 1.
0): 211, ('friday', 1.0): 116, ('lwwf', 1.0): 1, ('second', 1.0): 1
0, ('thought', 1.0): 29, ('’', 1.0): 21, ('enough', 1.0): 18, ('tim
e', 1.0): 127, ('dd', 1.0): 1, ('new', 1.0): 143, ('short', 1.0): 7,
```

In [26]:

```python
#TODO: prinf all the key-value pair of frequency dictionary
freqs
```

Out[26]:

```
{('followfriday', 1.0): 25,
 ('top', 1.0): 32,
 ('engag', 1.0): 7,
 ('member', 1.0): 16,
 ('commun', 1.0): 33,
 ('week', 1.0): 83,
 (':)', 1.0): 3568,
 ('hey', 1.0): 76,
 ('jame', 1.0): 7,
 ('odd', 1.0): 2,
 (':/', 1.0): 5,
 ('pleas', 1.0): 97,
 ('call', 1.0): 37,
 ('contact', 1.0): 7,
 ('centr', 1.0): 2,
 ('02392441234', 1.0): 1,
 ('abl', 1.0): 8,
 ('assist', 1.0): 1,
```

## Table of word counts

In [27]:

```python
# select some words to appear in the report. we will assume that each word is uniqu
keys = ['happi', 'merri', 'nice', 'good', 'bad', 'sad', 'mad', 'best', 'pretti',
        '❤', ':)', ':(', '😒', '😬', '😄', '😍', '♛',
        'song', 'idea', 'power', 'play', 'magnific']



#TODO: Create the empty list as list representing our table of word counts, where
#each element consist of a sublist with this pattern: [<word>, <positive_count>, <n
data = []

#TODO:Iterate over each word in keys
for word in keys:

    # initialize positive and negative counts
    pos = 0
    neg = 0

    # retrieve number of positive counts
    if (word, 1) in freqs:
        pos = freqs[(word, 1)]

    # retrieve number of negative counts
    if (word, 0) in freqs:
        neg = freqs[(word, 0)]

    # append the word counts to the table
    data.append([word, pos, neg])

data
```

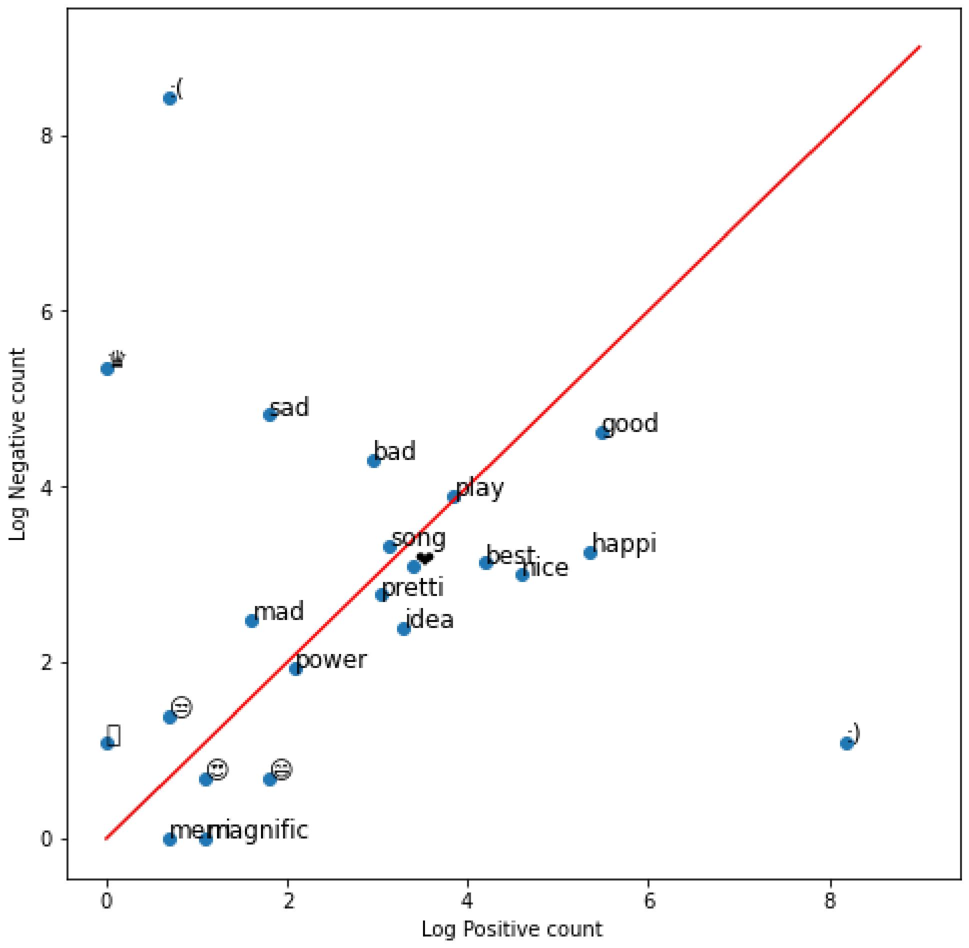
Out[27]:

```
[['happi', 211, 25],
 ['merri', 1, 0],
 ['nice', 98, 19],
 ['good', 238, 101],
 ['bad', 18, 73],
 ['sad', 5, 123],
 ['mad', 4, 11],
 ['best', 65, 22],
 ['pretti', 20, 15],
 ['❤', 29, 21],
 [':)', 3568, 2],
 [':(', 1, 4571],
 ['😒', 1, 3],
 ['😬', 0, 2],
 ['😄', 5, 1],
 ['😍', 2, 1],
 ['♛', 0, 210],
 ['song', 22, 27],
 ['idea', 26, 10],
 ['power', 7, 6],
 ['play', 46, 48],
 ['magnific', 2, 0]]
```

In [28]:

```
fig, ax = plt.subplots(figsize = (8, 8))

# convert positive raw counts to logarithmic scale. we add 1 to avoid log(0)
x = np.log([x[1] + 1 for x in data])

# do the same for the negative counts
y = np.log([x[2] + 1 for x in data])

# Plot a dot for each pair of words
ax.scatter(x, y)

# assign axis labels
plt.xlabel("Log Positive count")
plt.ylabel("Log Negative count")

# Add the word as the label at the same position as you added the points just befor
for i in range(0, len(data)):
    ax.annotate(data[i][0], (x[i], y[i]), fontsize=12)

ax.plot([0, 9], [0, 9], color = 'red') # Plot the red line that divides the 2 areas
plt.show()
```

```
/home/yash/anaconda3/lib/python3.8/site-packages/matplotlib/backends/b
ackend_agg.py:238: RuntimeWarning: Glyph 128556 missing from current f
ont.
  font.set_text(s, 0.0, flags=flags)
/home/yash/anaconda3/lib/python3.8/site-packages/matplotlib/backends/b
ackend_agg.py:201: RuntimeWarning: Glyph 128556 missing from current f
ont.
  font.set_text(s, 0, flags=flags)
```



:(
♛
sad
good
bad
play
song
happi
❤
best
nice
pretti
mad
idea
power
:)
merri
magnific
Log Negative count
Log Positive count