COMP 4030/6030 - Spring 2017
Final Exam

Name: Andrew Hayes

Note: Do not write on the backs; only front pages are digitized and graded.

1. (*25 points*) Write a Python procedure that prints out all sets consisting of maximally N numbers (N is a global variable). For example, with N=3, the call Set(-1) will print out

```
{}
{1}
{2}
{3}
{1,2}
{1,3}
{2,3}
{1,2,3}
```

The order of the sets does not have to be exactly like this, but there must be 8 of them (for N=3). Of course, your program must work for any value of N larger than 0. Complete the partially written Python code below.

```
# Assume N is a global variable.  Solution will be a boolean list.
Solution = [-1] * N

def Set(i):
    if i==N-1:
        # print out the set
        for x in Solution:
            if Solution[x] == True:
                print(x+1)
    else:
        # complete this part
        Solution[i+1] = True
        Set(i+1)
        Solution[i+1] = False
        Set(i+1)
Set(-1)
```

25

2. (*25 points*) Write a Python procedure that prints out all permutations consisting of N numbers (N is a global variable). For example, with N=3, the call Permutation(-1) will print out

```
[0,1,2]        [0,1,2,3] [0,2,3,1]
[0,2,1]        [0,2,1,3]
[1,0,2]        [0,1,3,2]                N!
[1,2,0]        [0,3,1,2]
[2,0,1]        [0,3,2,1]
[2,1,0]
```

The order of the permutations does not have to be exactly like this, but there must be 6 of them (for N=3). Of course, your program must work for any value of N larger than 0. Complete the partially written Python code below.

```
# Assume N is a global variable.  Solution.
Solution = [-1] * N

def Permutation(i):
    if i==N-1:
        print(Solution)
    else:
        # complete this part
        for x in Solution:
            Solution[x]=x

3       Permutation(i+1)

                        ¯\_('.')_/¯

Permutation(-1)
```

3. (*25 points*) Given a graph G, use backtracking to print out all sets of nodes that do not have any mutual edges. (Imagine in a social network, the problem is to list of sets of non-friends; i.e. such a set is a group of people who do not mutually know each other). Each such set must have more than one node.

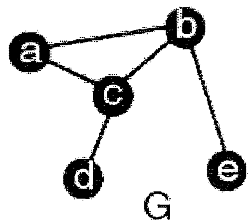

Figure 1: Five sets of non-friends are: { a, d }, { a, e }, { b, d }, { c, e }, { a, d, e }

```
Solution = [-1] * len(G.Nodes)    # Assume the graph G is a global variable.

def non_friends(i):
    if promising(i):  # complete the definition of promising below
        if i==N-1:
            # print out the set of non-friends
            if [i[(a,b)]] not in G:
               print Solution[i]
```

15

```
        else:
            # complete this part
            Solution[i+1]=True
            non_friends(i+1)
            Solution[i+1] = false
            non_friends(i+1)
# return True if the partial set represented by Solution[0], ..., Solution[i] is a set of non-friends.
# Hint: to check if there is an edge between node a and node b, use this code: (a,b) in G.
def promising(i):
    if the nodes in i are not (a,b) in G:
        return True

    return False
```

4. (*15 points*) Modify the last problem to find the largest set of non-friends. For example, in the graph shown in the previous problem, $\{a, d, e\}$ is the largest set of non-friends. (If there're more than one largest sets, anyone will do).

```
Solution = [-1] * len(G.Nodes)
largest = []

def largest_non_friends(i):
    # complete this
      if promising (i) >= largest :
        largest == promising (i)
```

```
def promising(i):
    # complete this
```

5. (*25 points*) In this problem, we want to know if it is possible to fit items perfectly into a bag. It is similar to a few problems we have studied. The output is True or False. The technique is dynamic programming.

The input is a list of weights $w_1, , w_n$ and a size C. The output is True if it is possible to fit some of these items perfectly to a bag with capacity C. Each item can be used only once.

Example, given weights 3, 5, 10, 2 and C = 7, the output is True (because 5+2=7). If C = 9, the answer is False.

Previously, we were interested in listing all valid packings. In this problem, we are only interested in whether it is possible to pack. Solve this problem by completing the partially written code below.

8

```
# Packable(i, c) returns True if it is possible to pack perfectly items
# 0, 1, ..., i with c being the capacity of the bag.
def Packable(i, c):
    if c == 0:
        return True
    if i < 0:
        return False

    # Solution of the problem is based on the analysis that there are only two possibilities:
    # (1) item being part of the solution,
    # (2) item i not being part of the solution.

    # Initialize these two options.
    i_selected = False
    i_not_selected = False


    # Fill in your code here to compute i_not_selected
      if c-W[i]:
         i_not_selected = True


    # Fill in your code here to compute i_selected
      if c-W[i] >= 0:
         i_selected = True



    # Fill in your code here to return the answer deciding which possible is the correct one.
       if i_selected == True:
          Packable(i+1, c-i)
      if i_not_selected == True:
         Packable(i+1, c)

# This code will solve the problem we are interested in
W = get_weights()    # this is a list of N weights
C = get_capacity()
print( Packable(len(W)-1, C) )
```