COMP 4030/6030 -- Assignment 9

**Due: Tuesday 12/05/2017**

**This is a coding assignment. Mail your work to the TA:**

* The name your solution file should be the same as your UID, plus a .py extension. For example, if your UID is jsmith (i.e. your email is [jsmith@memphis.edu)](mailto:jsmith@memphis.edu)), then your solution file should be **jsmith.py**.
* In the file, put your full name, COMP 4030 or COMP 6030, and Assignment 9.
* Send your solution to the TA (Quang Tran, [qmtran@memphis.edu)](mailto:qmtran@memphis.edu)) with the subject line “**COMP 4030 Assignment 9**”.

1. (30 points) This Python function (pal) is slow. **Rewrite pal using a cache to speed it up**.

def pal(seq):

if len(seq) <= 1:

return len(seq)

if seq[0] == seq[-1]:

return 2 + pal(seq[1:-1])

else:

return max(pal(seq[1:]), pal(seq[:-1]))

**Time and compare the raw running times of pal and your cached version on random strings of different lengths**. The function rand\_str, defined below, can be used to generate a random string. For example, rand\_str(35, ‘acgt’) will generate a random string consisting of either a or c or g or t.

import random

def rand\_str(n, chars):

return ''.join([ random.choice(chars) for i in range(n) ])

1. (30 points) Let’s say you want to select participants on a social network for a game show. One condition you want is to minimize collusion among the participants. Because of that, the participants must not know each other. We can model this as a *graph* problem. Given an undirected graph (representing a social network), the goal of this problem is to enumerate through (or print out) all sets of *complete* *strangers* in this graph. A set of *complete* *strangers* in a graph is a set of vertices that have no edges among them. Recall that this backtracking function below prints out all sets:

def sets(i, solution):

if i==len(solution)-1:

print(solution)

else:

for item in [False, True]:

solution[i+1] = item

sets(i+1, solution)

Modify it so that when you find a *complete* solution check to make sure that it is a group strangers. In other words, your backtracking function will print out all sets of strangers in a graph.

Hint:

* You will want to pass a graph as an input parameter too.
* Prune your search space with a *promising* function in a similar way as in the NQueen problem.

1. (30 points) Write a new backtracking function that finds the largest group of complete strangers in a graph. Hint: have a global *largest\_group* variable to keep of track of the largest group of complete strangers.
2. (10 points) Speed up your backtracking function that finds the largest group of strangers using *pruning*. Do this by defining a *promising* function that checks a current partial solution (internal node of the search tree) in such a way that if the current solution cannot possibly be a group of strangers, then do not explore that subtree.

Can you speed up the pruning even more by comparing the current solution to the current best solution?