Constraint satisfaction problems - Local search for CSPs

- CSP is a problem when there are some contrains in problem
- **Variables:**
- **Domain:** range of values can be assigned to variables
- **Constraints:** allowable combinations of values for subsets of variables
- **Use:** Very useful in job scheduling, sudoku, permutation and math problem solver
- For example: who teaches what class, Transportation scheduling, Factory scheduling, etc.

python-constraint library

- Python libraries for constraint satisfaction problem: https://pypi.org/project/python-constraint/
- These library require variable and constraint

Pythagorean triples (Just for reference - learn to use python constraint solving library)

Find (x, y, z) such that,

- $x^2 + y^2 = z^2$
- $x < y < z$
- $\max\{x, y, z\} < 30$

In [1]:
```
import constraint
problem = constraint.Problem()

problem.addVariable('x', range(30))
problem.addVariable('y', range(30))
problem.addVariable('z', range(30))

def pythagorean_constraint(x, y, z):
    if(x<y and y<z):
        num_list = [x,y,z]
        if(max(num_list) <30 and min(num_list)>0):
            if((x ** 2) + (y ** 2) == (z ** 2)):
                return True
    return False

problem.addConstraint(pythagorean_constraint, ['x','y', 'z'])
solutions = problem.getSolutions()
print("Total possible solutions: ", len(solutions))
print("All possible solutoins: ", solutions)
```

```
Total possible solutions:  10
All possible solutoins:  [{'x': 20, 'y': 21, 'z': 29}, {'x': 15, 'y': 20, 'z':
25}, {'x': 12, 'y': 16, 'z': 20}, {'x': 10, 'y': 24, 'z': 26}, {'x': 9, 'y': 1
2, 'z': 15}, {'x': 8, 'y': 15, 'z': 17}, {'x': 7, 'y': 24, 'z': 25}, {'x': 6,
'y': 8, 'z': 10}, {'x': 5, 'y': 12, 'z': 13}, {'x': 3, 'y': 4, 'z': 5}]
```

Problem: Map coloring problem (10 marks)

- Variables: All provinces - ON, QC, NS, NB, MB, BC, PE, SK, AB, NL
- Domains: colors {red, green, blue}
- Constraints: Adjacent regions(provinces) must have different colors

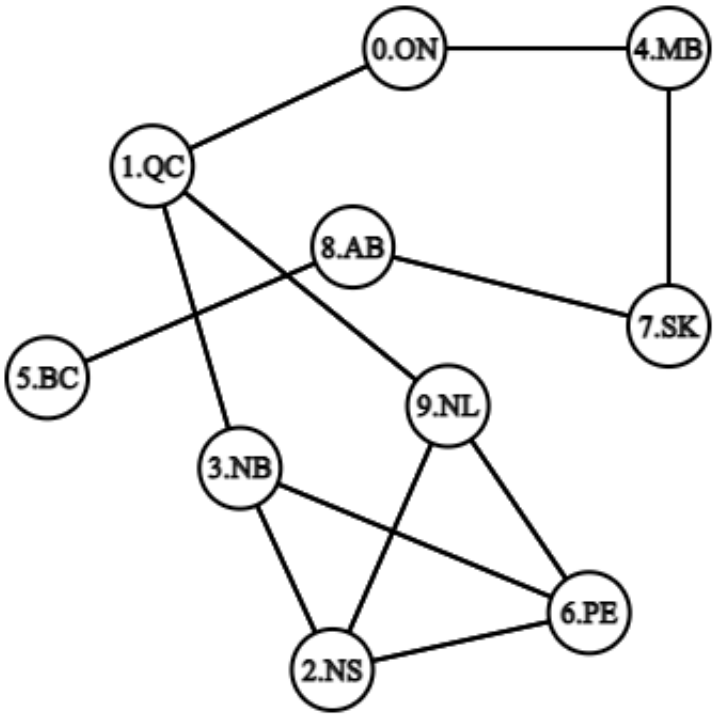
- Fill in the blanks
- Remove "pass" keyword and write appropritate code
- Use comments to get insight of question

Map of Canada:

Reference: https://en.wikipedia.org/wiki/Canada

Solution:

- Step 1: Form a graph of provinces with edges connected to adjacent provinces



- step 2: Create a graph representation of all provinces and their adjacent provinces
- step 3: write a code for CSP
- step 4: print possible color sequences if possible, else return - "not possible"

In [10]:

```
# This functino check if selected color for a province satisfy the constraint
def check_constraints(graph, state_index, color, c, total_states):
    for i in range(total_states):
        if(graph[state_index][i] == 1 and color[i] == c): #Q1: check if graph
            return False #constraint not satisified returning false
    return True #constraints satisfied returning true

# Recusrsive function
def apply_color(graph, total_color, color, total_states, state_index):
    if state_index == total_states: #end condition for recursive call
        return True

    #iterating for all colors
    for c in range(1, total_color + 1):
        if(check_constraints(graph, state_index, color, c, total_states) == T
            color[state_index] = c
            if(apply_color(graph, total_color,color, total_states, state_inde
                return True
            color[state_index] = 0


# main function to solve the problem
def csp_problem(graph, total_states, total_color):
    color = [0]*total_states #Q3: create list with all zeroes of size total_s
    if(apply_color(graph, total_color, color,total_states,  0) == None):
        print("solution doesn't exist")
        return False   #Q4: print that solution does not exist and return Fal

    print("Solution exists and states can be colored with:")
    for c in color:
        print(c, end = " ")#Q5: print color in one line - apply required form
        #instead of numbers you can endode them to print red, green, blue or
    return True

################################################################################
#code execution starts from here
total_states = 10  #total_states of canada

#creating graph which represent neighbour provinces set - check graph image i
# do not touch this - this has been already created for you
graph = [[0, 1, 0, 0, 1, 0, 0, 0, 0, 0],[1, 0, 0, 1, 0, 0, 0, 0, 0, 1],[0, 0,
 [1, 0, 0, 0, 0, 0, 0, 1, 0, 0],[0, 0, 0, 0, 0, 0, 0, 0, 1, 0],[0, 0, 1, 1, 0
 [0, 0, 0, 0, 0, 1, 0, 1, 0, 0],[0, 1, 1, 0, 0, 0, 1, 0, 0, 0]]

# feel free to change total_color and check if we can color all provinces of
# selected colors by satisfying neighbouring constrains
total_color = 3

csp_problem(graph, total_states, total_color) # calling function from here

# Expected answer: 1 2 1 3 2 1 2 1 2 3 (not necessary to be the same)
```

```
Solution exists and states can be colored with:
1 2 1 3 2 1 2 1 2 3
```

Out[10]: True

In [ ]: