

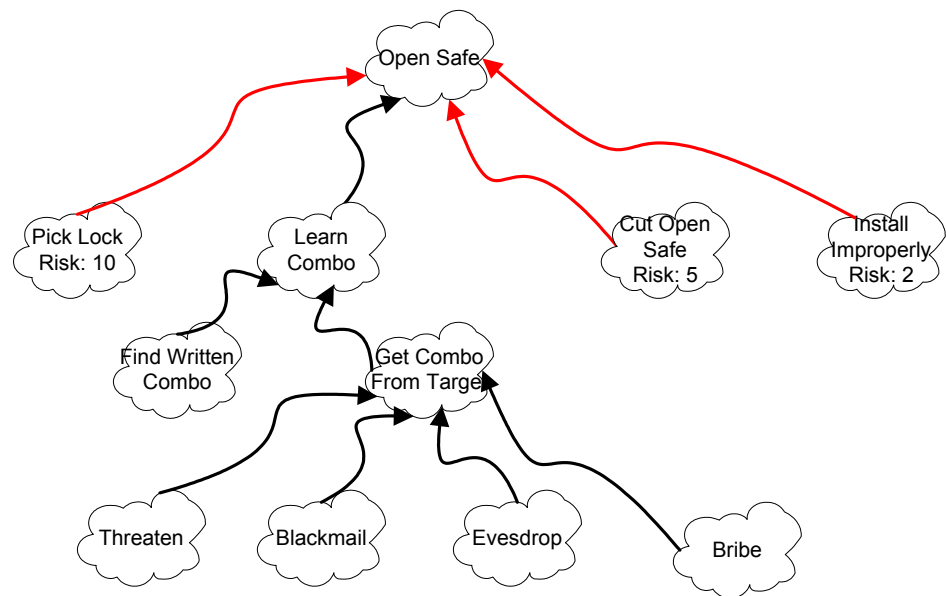
# Thinking with Graphs – a Simple Visual Introduction

## **Motivation:**

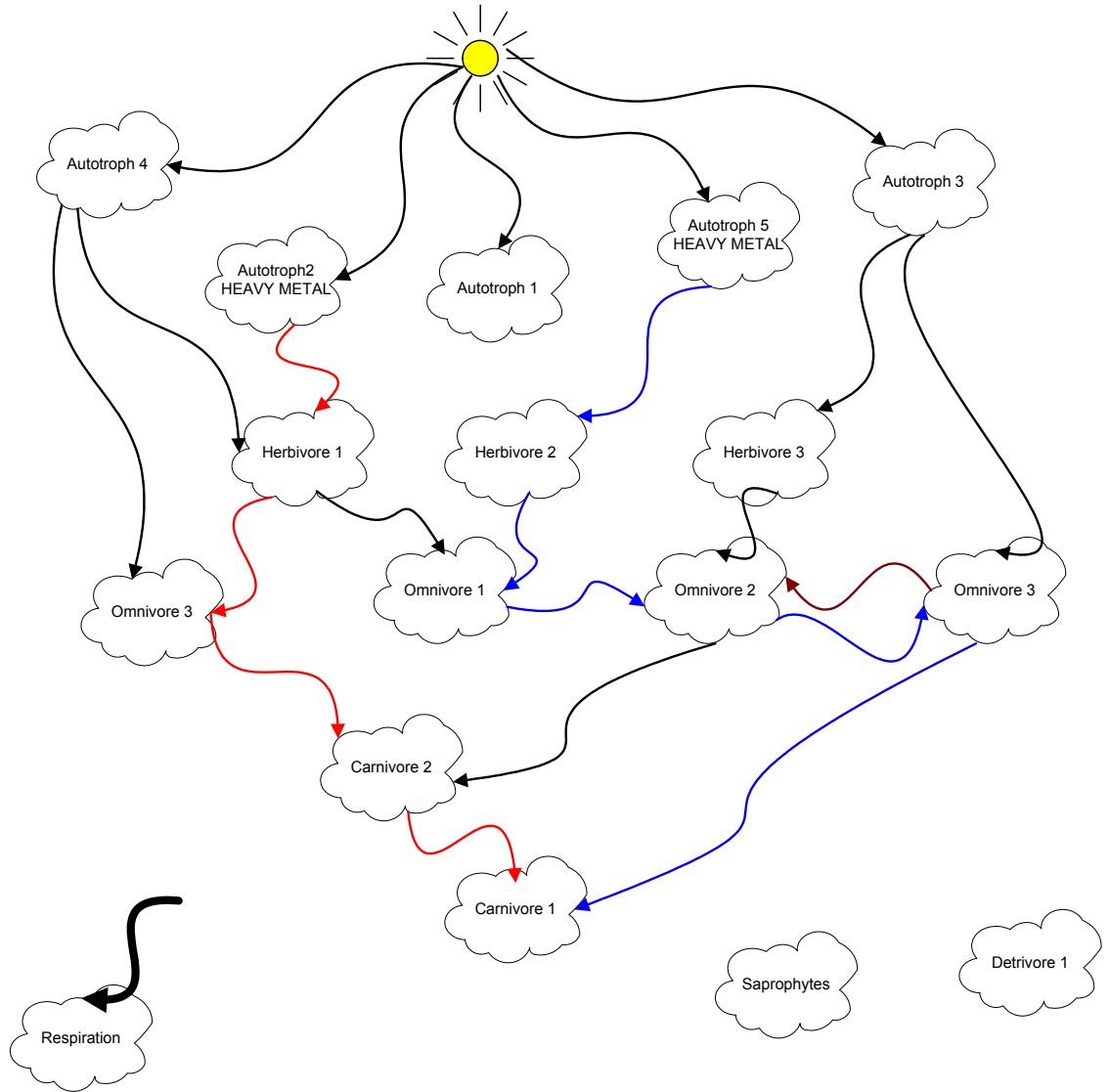
Many interesting problems around “searching” through a data set can be put in a form where they represent paths through a graph. Since we are good at visualizing such problems, graphs provide a nice metaphor for thinking about “searching” through data, otherwise known as “querying”. This short note, introduces the notion of “thinking with graphs” in a non-technical fashion as an introduction to the key metaphor in the XAYA project, which is building a Pythonic mini-language for “thinking with data”, based on the “data as graphs” metaphor ... i.e. “Thinking with Graphs”.

### **Example 1: What are the different ways to attack a safe?**

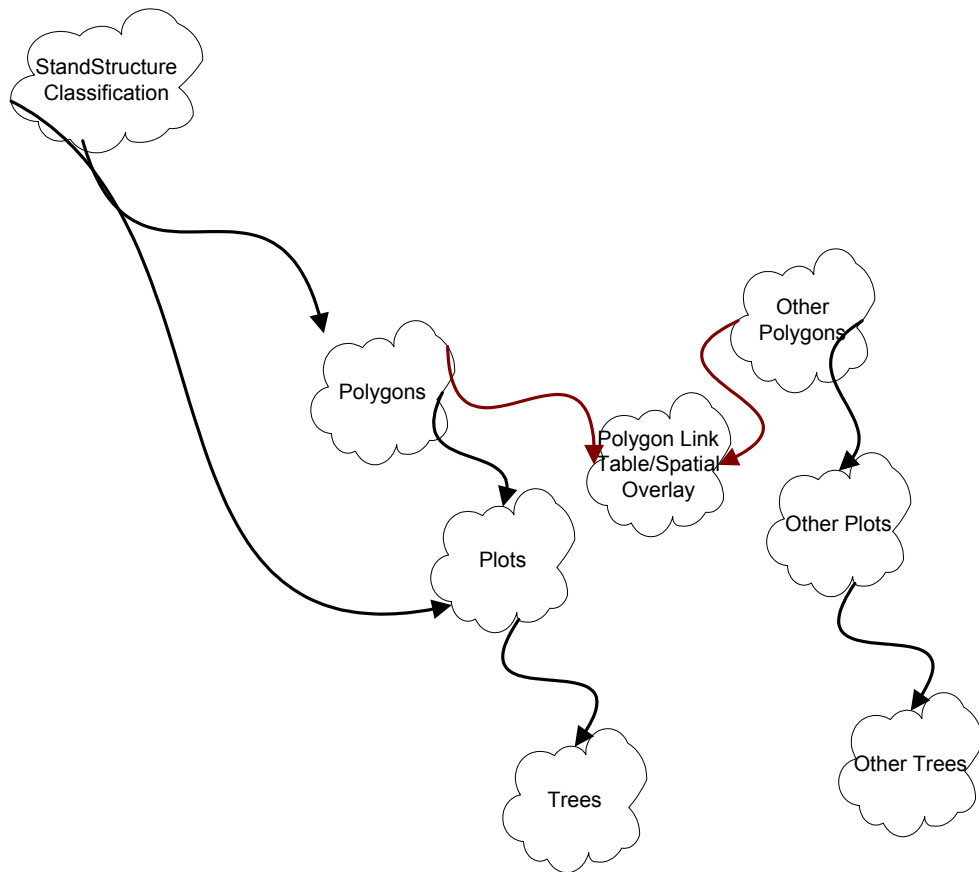
“Attack Tree” modified from Secrets and Lies by Bruce Schneier



**Example 2: A hypothetical ecosystem. What predator/prey paths contribute to “poisoning” the top of the food chain (the carnivores) with heavy metals.**



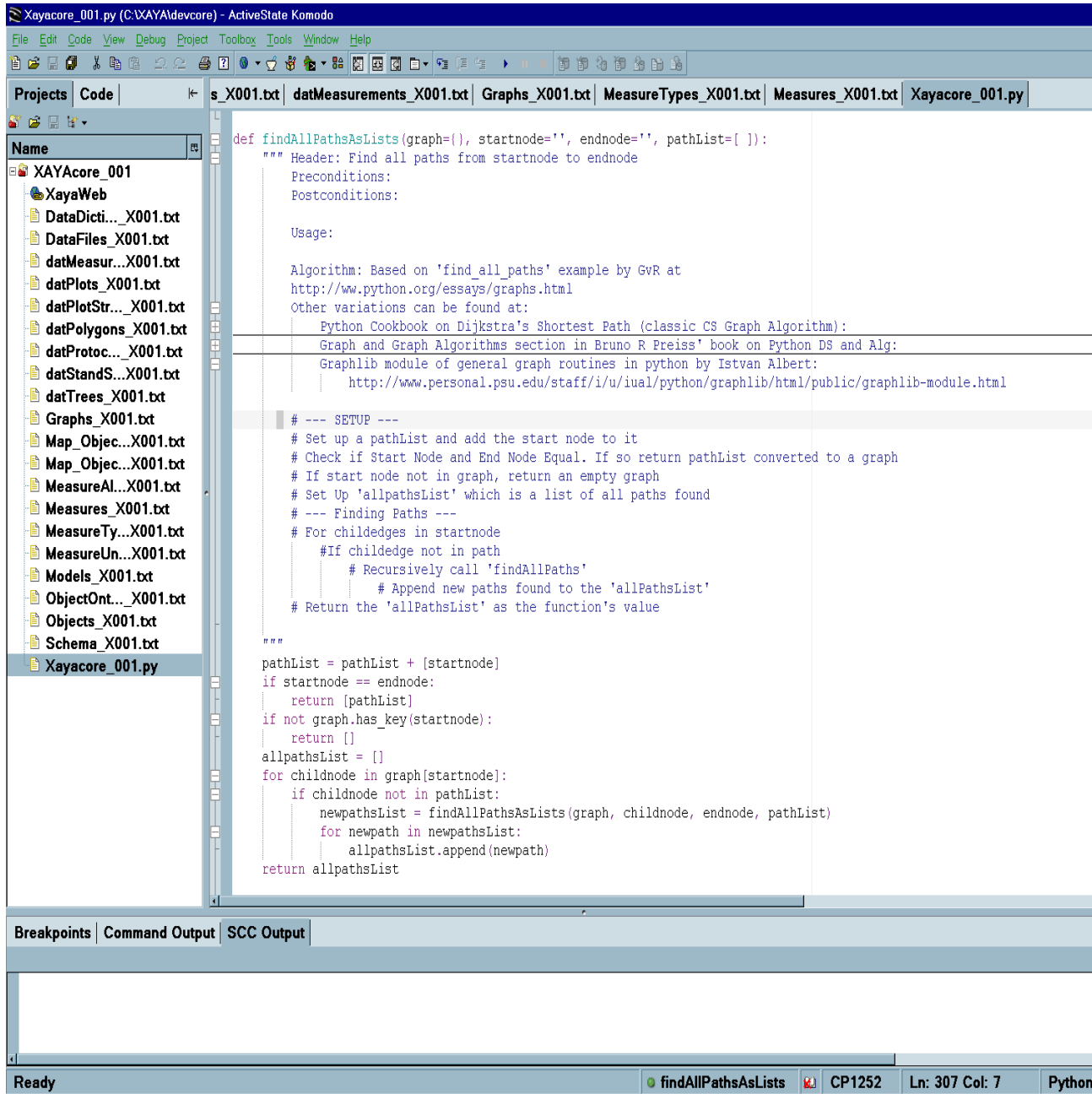
**Example 3: How can we consolidate information from two forestry databases (one beginning with “Stand Structure” and the other beginning with “Other Polygons”), which have similar kinds of information on “Forest Business Objects” (StandStructures, Polygons, Plots, Trees)?**



## Code and Usage Example:

Lets Focus on the common task in all these examples, “finding a path through a graph”.

Code below is adapted from “Python Patterns – Implementing Graphs” by GvR. Here is the annotated code and a usage example. (if two small too read, zoom in).



```
def findAllPathsAsLists(graph={}, startnode='', endnode='', pathList=[ ]):
    """ Header: Find all paths from startnode to endnode
        Preconditions:
        Postconditions:

        Usage:

        Algorithm: Based on 'find all_paths' example by GvR at
        http://ww.python.org/essays/graphs.html
        Other variations can be found at:
        Python Cookbook on Dijkstra's Shortest Path (Classic CS Graph Algorithm):
        Graph and Graph Algorithms section in Bruno R Preiss' book on Python DS and Alg:
        Graphlib module of general graph routines in python by Istvan Albert:
        http://www.personal.psu.edu/staff/i/u/ial/python/graphlib/html/public/graphlib-module.html

        # --- SETUP ---
        # Set up a pathList and add the start node to it
        # Check if Start Node and End Node Equal. If so return pathList converted to a graph
        # If start node not in graph, return an empty graph
        # Set Up 'allpathsList' which is a list of all paths found
        # --- Finding Paths ---
        # For childedges in startnode
        # If childedge not in path
        #     Recursively call 'findAllPaths'
        #     Append new paths found to the 'allPathsList'
        # Return the 'allPathsList' as the function's value

    """
    pathList = pathList + [startnode]
    if startnode == endnode:
        return [pathList]
    if not graph.has_key(startnode):
        return []
    allpathsList = []
    for childnode in graph[startnode]:
        if childnode not in pathList:
            newpathsList = findAllPathsAsLists(graph, childnode, endnode, pathList)
            for newpath in newpathsList:
                allpathsList.append(newpath)
    return allpathsList
```



```

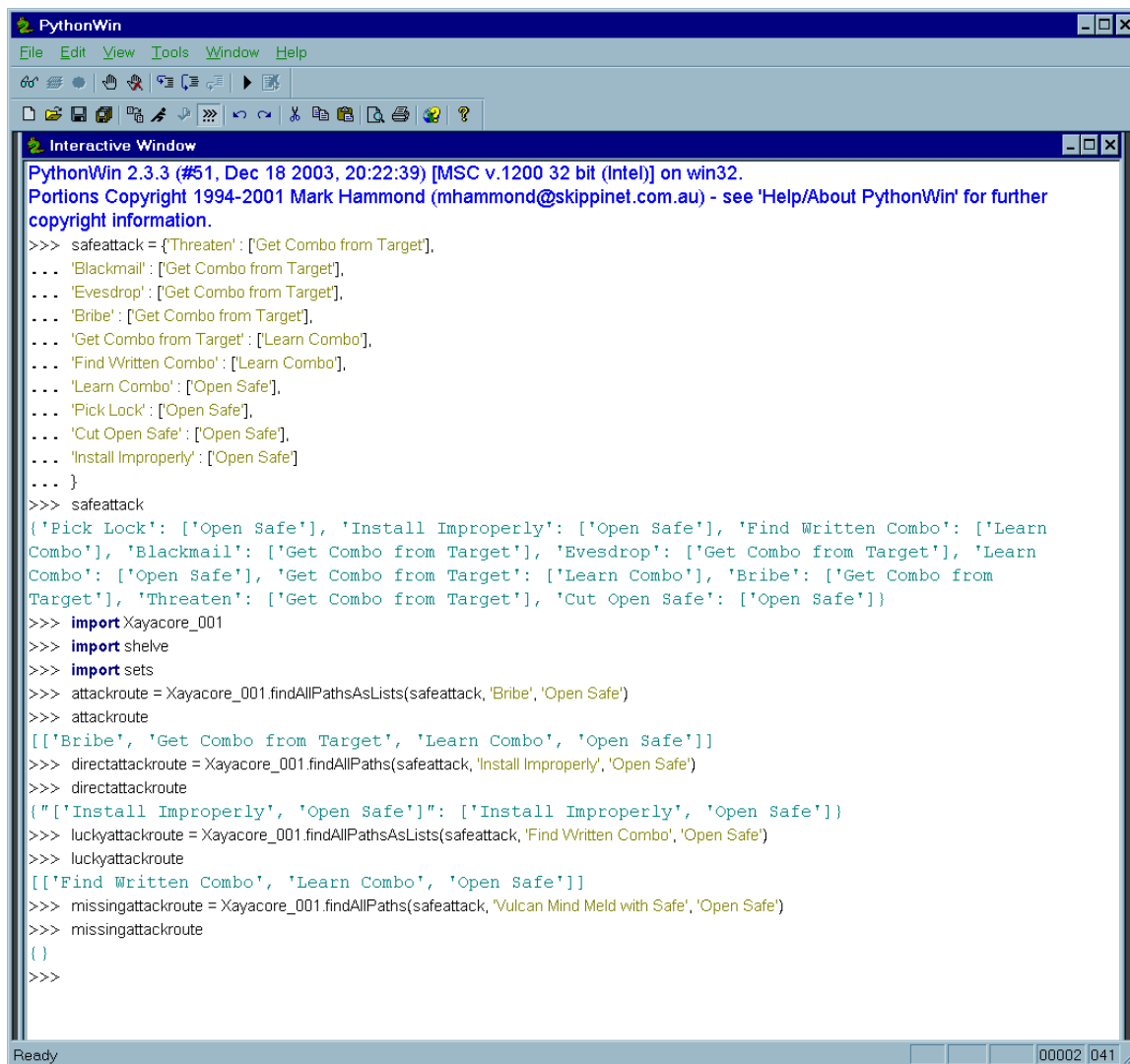
PythonWin
File Edit View Tools Window Help
Interactive Window
copyright information.
>>> import Xayacore_001
>>> import shelve
>>> graph = shelve.open('XAYA/devcore/shelfForestInventoryDBSchema')
>>> graph
{'StandStructure': ['Polygon', 'Plot', 'RolledUpStandTable', 'RolledUpGMStandTable'], 'Tree':
['TreeMeasurements', 'GrowthModelData'], 'Polygon': ['Plot'], 'PlotMeasurements': ['StandTable',
'GrowthModelStandTable'], 'Plot': ['Tree', 'CurrentStandTable', 'PlotSummary', 'PlotMeasurements']}
>>> inventorypath = Xayacore_001.findAllPaths(graph, 'StandStructure', 'GrowthModelData')
>>> inventorypath
[["StandStructure", "Plot", "Tree", "GrowthModelData"]: ["StandStructure", "Plot", "Tree",
"GrowthModelData"], ["StandStructure", "Polygon", "Plot", "Tree", "GrowthModelData"]:]
>>> inventorypathlists = Xayacore_001.findAllPathsAsLists(graph, 'StandStructure', 'GrowthModelData')
>>> inventorypathlists
[["StandStructure", "Polygon", "Plot", "Tree", "GrowthModelData"], ["StandStructure", "Plot",
"Tree", "GrowthModelData"]]
>>> cyclegraph = shelve.open('XAYA/devcore/shelfCycleGraph')
>>> cyclegraph
{'a': ['b', 'e'], 'c': ['d'], 'b': ['c'], 'e': ['f'], 'd': ['a'], 'g': ['h', 'd'], 'f': ['a'],
'h': ['c']}
>>> acycle = Xayacore_001.findAllPaths(cyclegraph, 'a', 'f')
>>> acycle
{"a": ["a", "e", "f"]}: ["a", "e", "f"]}
>>> # Lets do an example of finding roots
>>> # A root has no parents -- so it is those keys, that are not in the valuelist for a graph
>>> nodelist = []
>>> for key in graph:
...     for value in graph[key]:
...         nodelist.append(value)
...
>>> nodelist
['TreeMeasurements', 'GrowthModelData', 'Plot', 'StandTable', 'GrowthModelStandTable', 'Polygon',
'Plot', 'RolledUpStandTable', 'RolledUpGMStandTable', 'Tree', 'CurrentStandTable', 'PlotSummary',
'PlotMeasurements']
>>> import sets
>>> forestrynodeset = sets.Set(nodelist)
>>> for key in graph:
...     if key not in forestrynodeset:
...         print key
...
StandStructure
>>> |
Ready 00036 005

```

This example shows usage of the “findAllPaths” function, and how easy it would be to “find the roots” of a graph. Rather than “finding roots” in a graph, how would you go about finding leaf nodes (hint, a leaf node, has no child nodes, so it would not appear in keys for a dictionary representing the graph)?



Here's a second example, now utilizing the "Safe Attack" scenario:



```
PythonWin
File Edit View Tools Window Help
Interactive Window
PythonWin 2.3.3 (#51, Dec 18 2003, 20:22:39) [MSC v.1200 32 bit (Intel)] on win32.
Portions Copyright 1994-2001 Mark Hammond (mhammond@skippinet.com.au) - see 'Help/About PythonWin' for further
copyright information.
>>> safeattack = {'Threaten': ['Get Combo from Target'],
... 'Blackmail': ['Get Combo from Target'],
... 'Evesdrop': ['Get Combo from Target'],
... 'Bribe': ['Get Combo from Target'],
... 'Get Combo from Target': ['Learn Combo'],
... 'Find Written Combo': ['Learn Combo'],
... 'Learn Combo': ['Open Safe'],
... 'Pick Lock': ['Open Safe'],
... 'Cut Open Safe': ['Open Safe'],
... 'Install Improperly': ['Open Safe']
... }
>>> safeattack
{'Pick Lock': ['Open Safe'], 'Install Improperly': ['Open Safe'], 'Find Written Combo': ['Learn
Combo'], 'Blackmail': ['Get Combo from Target'], 'Evesdrop': ['Get Combo from Target'], 'Learn
Combo': ['Open Safe'], 'Get Combo from Target': ['Learn Combo'], 'Bribe': ['Get Combo from
Target'], 'Threaten': ['Get Combo from Target'], 'Cut Open Safe': ['Open Safe']}
>>> import Xayacore_001
>>> import shelve
>>> import sets
>>> attackroute = Xayacore_001.findAllPathsAsLists(safeattack, 'Bribe', 'Open Safe')
>>> attackroute
[['Bribe', 'Get Combo from Target', 'Learn Combo', 'Open Safe']]
>>> directattackroute = Xayacore_001.findAllPaths(safeattack, 'Install Improperly', 'Open Safe')
>>> directattackroute
{(['Install Improperly', 'Open Safe']): ['Install Improperly', 'Open Safe']}
>>> luckyattackroute = Xayacore_001.findAllPathsAsLists(safeattack, 'Find Written Combo', 'Open Safe')
>>> luckyattackroute
[['Find Written Combo', 'Learn Combo', 'Open Safe']]
>>> missingattackroute = Xayacore_001.findAllPaths(safeattack, 'Vulcan Mind Meld with Safe', 'Open Safe')
>>> missingattackroute
{}
>>>
```

This example uses a modified version of the "Attack Tree" on pg 321 (fig 21.2) of Bruce Schneier's "Secrets and Lies". I have "reversed the direction" of the tree he used, so that "Open Safe" rather than being the root, is the terminal node. And the roots are every beginning point towards attacking a safe.

PS: Don't try this at home. Jail sentence may follow.

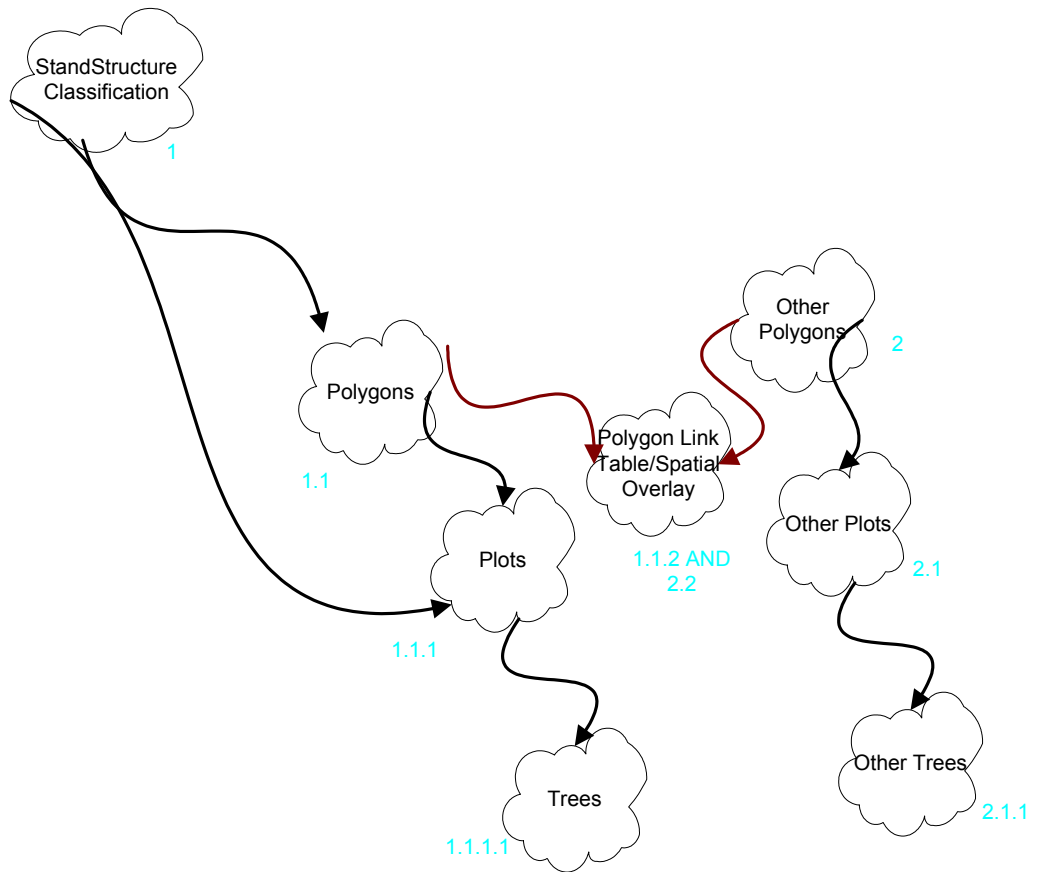


## Try at Home Exercise:

Let us generalize “findAllPaths” so that it no longer depends on knowing the “start” and ‘end-nodes”. How could we do this???? (An Algorithm please)

What tests would we write to test the algorithm????

A Hint: Here is a way of visualizing the problem as an indexing scheme exercise:



Obviously, the “index” for each object, also supplies it’s path back to the root. Objects with multiple roots have multiple index values. Using this information, can you develop code whose input is: (i) a graph, (ii) an unordered list of nodes (say those “objects” the user is primarily interested in) that works out the path, if it exists, through the user selected objects. If you try the exercise, and want to share it, please send me a note at [mishtu@harmeny.com](mailto:mishtu@harmeny.com) and I’ll post it to the XAYA website.