list of sets

```
a, b, c, d, e, f, g, h = range(8) # generates a list of incremental integers,
                                  # i.e. [0, 1, 2, 3, 4, 5, 6, 7]

N = [
    {b, c, d, e, f}, # a
    {c, e}, # b
    {d}, # c
    {e}, # d
    {f}, # e
    {c, g, h}, # f
    {f, h}, # g
    {f, g} # h
]

NOTE: > 2.7 supports set literals {1, 2, 3} and set([1, 2, 3])
```

list of lists

```
a, b, c, d, e, f, g, h = range(8)
N = [
    [b, c, d, e, f], # a
    [c, e], # b
    [d], # c
    [e], # d
    [f], # e
    [c, g, h], # f
    [f, h], # g
    [f, g] # h
]
```

adjacency dict with edge weights

```
a, b, c, d, e, f, g, h = range(8)
N = [
    {b:2, c:1, d:3, e:9, f:4}, # a
    {c:4, e:3}, # b
    {d:8}, # c
    {e:7}, # d
    {f:5}, # e
    {c:2, g:2, h:2}, # f
    {f:1, h:6}, # g
    {f:9, g:8} # h
]
```

dict with adjacency sets

```
N={
    'a': set('bcdef'),
    'b': set('ce'),
    'c': set('d'),
    'd': set('e'),
    'e': set('f'),
    'f': set('cgh'),
    'g': set('fh'),
    'h': set('fg')
}
```

adjacency matrix using nested lists

```
a, b, c, d, e, f, g, h = range(8)
# a b c d e f g h
N = [
    [0,1,1,1,1,1,0,0], # a
    [0,0,1,0,1,0,0,0], # b
    [0,0,0,1,0,0,0,0], # c
    [0,0,0,0,1,0,0,0], # d
    [0,0,0,0,0,1,0,0], # e
    [0,0,1,0,0,0,1,1], # f
    [0,0,0,0,0,1,0,1], # g
    [0,0,0,0,0,1,1,0]  # h
]
```

a weight matrix using infinity for missing edges

```
a, b, c, d, e, f, g, h = range(8)
_ = float('inf')
# a b c d e f g h
W = [
    [0,2,1,3,9,4,_,_], # a
    [_,0,4,_,3,_,_,_], # b
    [_,_,0,8,_,_,_,_], # c
    [_,_,_,0,7,_,_,_], # d
    [_,_,_,_,0,5,_,_], # e
    [_,_,2,_,_,0,2,2], # f
    [_,_,_,_,_,1,0,6], # g
    [_,_,_,_,_,9,8,0]  # h
]
```

additional representations

- not limited to just adjacency lists and adjacency matrices
  - edge lists or edge sets using pairs (or an "Edge" class) to represent edges as node pairs
  - also incidence matrices for multigraphs
  - interval graphs
  - see Spinrad

trees

list of lists

```
T =
[
    ["a", "b"],
    ["c"],
    ["d", ["e", "f"]]
]
```

binary tree class

```
class Tree:
    def __init__(self, left, right):
        self.left = left self.right = right
```

multiway tree

```
class Tree:
    def __init__(self, kids, next=None):
        self.kids = self.val = kids
        self.next = next
```