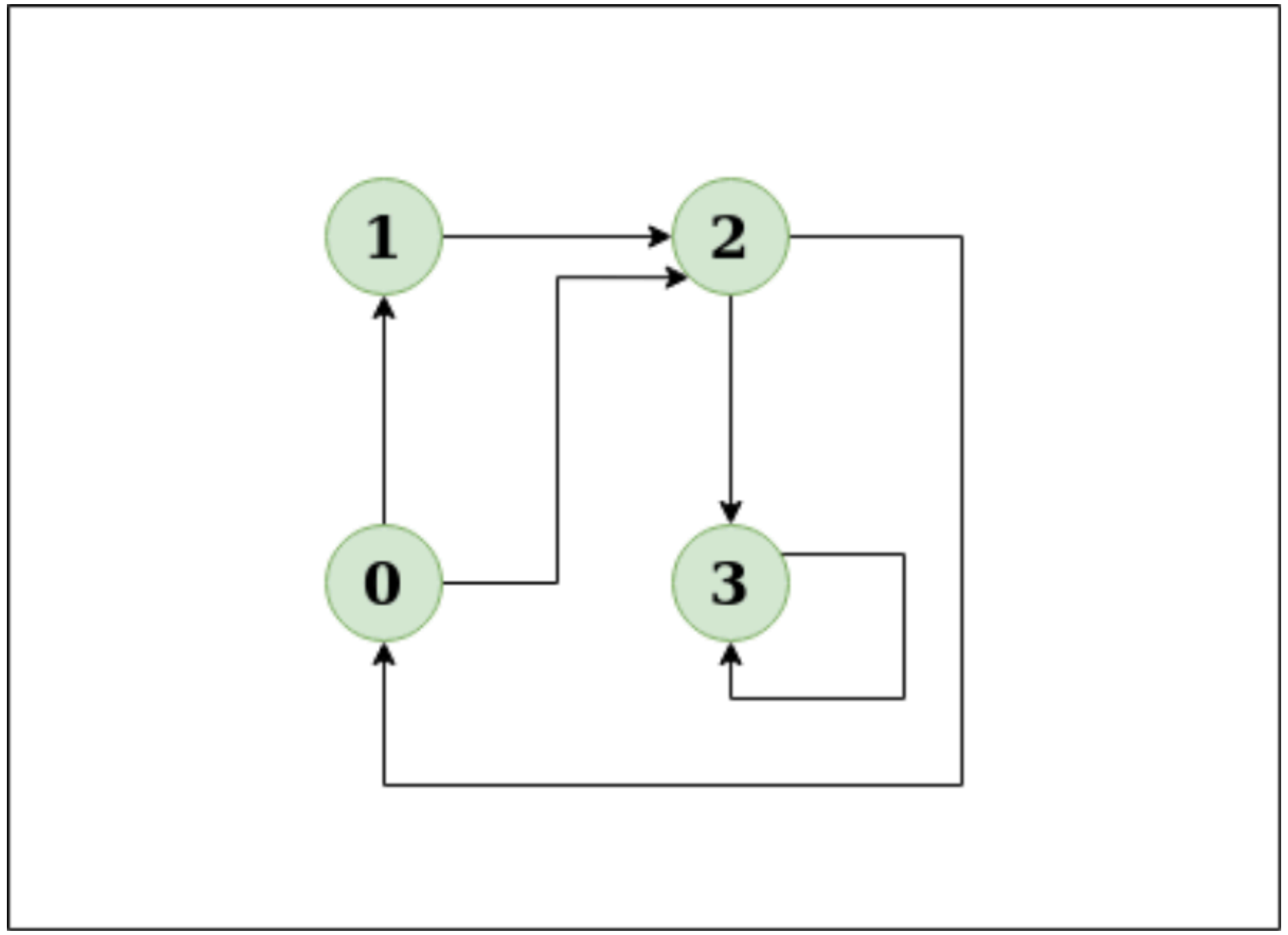
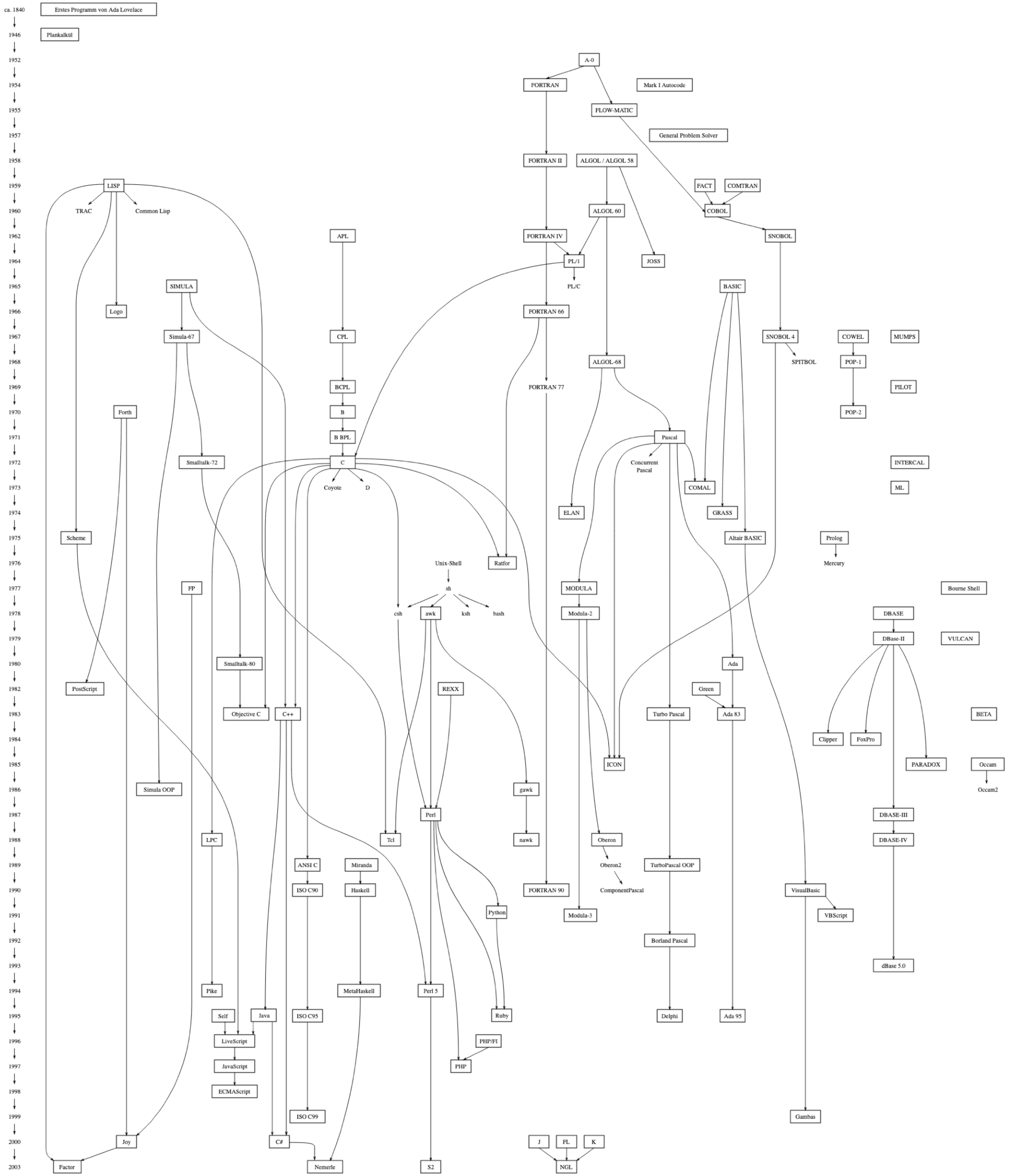
Does a graph have a cycle?

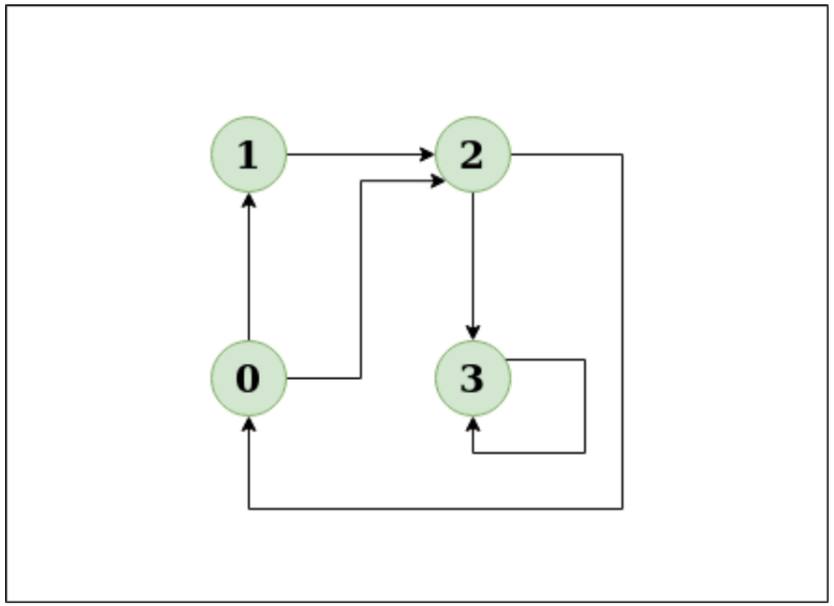
Mar 25


1
2
0
3


ca. 1840
1946
1952
1954
1955
1957
1958
1959
1960
1962
1964
1965
1966
1967
1968
1969
1970
1971
1972
1973
1974
1975
1976
1977
1978
1979
1980
1982
1983
1984
1985
1986
1987
1988
1989
1990
1991
1992
1993
1994
1995
1996
1997
1998
1999
2000
2003
Erstes Programm von Ada Lovelace
Plankalkül
A-0
FORTRAN
Mark I Autocode
FLOW-MATIC
General Problem Solver
FORTRAN II
ALGOL / ALGOL 58
LISP
FACT
COMTRAN
TRAC
Common Lisp
ALGOL 60
COBOL
APL
FORTRAN IV
SNOBOL
PL/1
JOSS
SIMULA
PL/C
BASIC
Logo
FORTRAN 66
Simula-67
CPL
SNOBOL 4
COWEL
MUMPS
ALGOL-68
SPITBOL
POP-1
BCPL
FORTRAN 77
PILOT
Forth
B
POP-2
B BPL
Pascal
Smalltalk-72
C
Concurrent Pascal
INTERCAL
Coyote
D
COMAL
ML
ELAN
GRASS
Scheme
Altair BASIC
Prolog
Unix-Shell
Ratfor
Mercury
FP
sh
MODULA
Bourne Shell
csh
awk
ksh
bash
Modula-2
DBASE
DBase-II
VULCAN
Smalltalk-80
Ada
PostScript
REXX
Green
Objective C
C++
Turbo Pascal
Ada 83
BETA
Clipper
FoxPro
ICON
PARADOX
Occam
Simula OOP
gawk
Occam2
Perl
DBASE-III
LPC
Tcl
nawk
Oberon
DBASE-IV
ANSI C
Miranda
Oberon2
TurboPascal OOP
ISO C90
Haskell
FORTRAN 90
ComponentPascal
VisualBasic
Python
Modula-3
VBScript
Borland Pascal
dBase 5.0
Pike
MetaHaskell
Perl 5
Self
Java
ISO C95
Ruby
Delphi
Ada 95
LiveScript
PHP/FI
JavaScript
PHP
ECMAScript
ISO C99
Gambas
Joy
C#
J
FL
K
Factor
Nemerle
S2
NGL

Representing a graph

- Use an adjacency list




```
Node:
    id: an integer
    links: an array of integers

Graph:
    g: an array of nodes
```

Position	Node
0	{id: 0, links: [1,2]}
1	{id: 1, links:[2]}
2	{id: 2, links:[0,3]}
3	{id: 3: links[3]}

Depth First Search

On an acyclic graph

- Recursion
- No base case!
- Time / space complexity

- Will this explore whole graph?
- How do we handle possibility of cycles?

```
Given: g – a graph
       n – an integer
           (of some node in graph)

DFS(g, n):
    for link in g[n].links:
        DFS(g, link)
```

Coloring

Extending Node

- white: unvisited
- grey: being processed
- black: visited

- Start by setting color of every node to white

```
Node:
    id: an integer
    links: an array of integers
    color: white, black, or grey

Graph:
    g: an array of nodes
```

DFS: with coloring

- Is this enough?
  - Detect a cycle if it exists?
  - Stops if there is a cycle?

```
Given: g – a graph
       n – an integer
           (of some node in graph)
Return -- true if there is a cycle

DFS(g, n):
    g[n].color=grey
    for link in g[n].links:
        if g[link].color==grey
           return true
        if DFS(g, link)==true
           return true
    g[n].color=black
    return false
```

Final Algorithm

```
for i in 0..nodes.count
    g[i].color=white
for i in 0..nodes.count
    if g[i].color==white
       if DFSc(g,i)
           return true
return false
```

```
Given: g - a graph
       n - an integer
           (of some node in graph)
Return -- true if there is a cycle

DFSc(g, n):
    g[n].color=grey
    for link in g[n].links:
         if g[link].color==grey
            return true
         if DFSc(g, link)==true
            return true
    g[n].color=black
    return false
```

Aliases

assignment_num	count	graded
2	12	18
3	11	13
4	9	17
5	9	16
6	6	15
20	1	
21	1	

Graph Representations

Name			
Implementation (Java)			
O(space)			

Topological sort

- Realistically do not need, just need to find nodes with in-degree of zero
  - Complexity of finding this?

- Can we use my cycle colorer to determine path from Washington to DC

Finding Cycles
and Topological Sort

Topo Sort Algorithm

```
Count in-degree of all nodes
Add all indegree=0 to Q
While Q
  remove n from Q
  for each e (edge from n)
    reduce indegree of node e.too
    if indegree of e.too == 0
      add e.too to Q
```

- for a graph G is topo sort unique

- For cycle finding do I need to do a full Topological Sort?
- if not, what do I need?

Graph Representations

Name			
Implementation (Java)			
O(space)			
Cost to find indegree: of each node of a given node			
Cost to find outdegree: of each node of a given node			

Final Algorithm With Topo Step 1

```
for i in 0..nodes.count
    g[i].color=white
for n in nodes with indegree==0
    if g[i].color==white
       if DFSc(g,i)
           return true
return false
```

```
Given: g - a graph
       n - an integer
           (of some node in graph
Return -- true if there is a cycl

DFSc(g, n):
    g[n].color=grey
    for link in g[n].links:
         if g[link].color==grey
            return true
         if DFSc(g, link)==true
            return true
    g[n].color=black
    return false
```

Is Topo (step 1) worth doing?

- Time complexity??

- Likely real-world time??

Use Coloring for maze walking?

"Death of a salesman" problem is, at core, a maze

- Can I?
- Algorithm
  - change from -->
- Benefits?
- Costs?

```
Given: g – a graph
       n – an integer
           (of some node in graph)
Return -- true if there is a cycle

DFSc(g, n):
    g[n].color=grey
    for link in g[n].links:
        if g[link].color==grey
            return true
        if DFSc(g, link)==true
            return true
    g[n].color=black
    return false
```