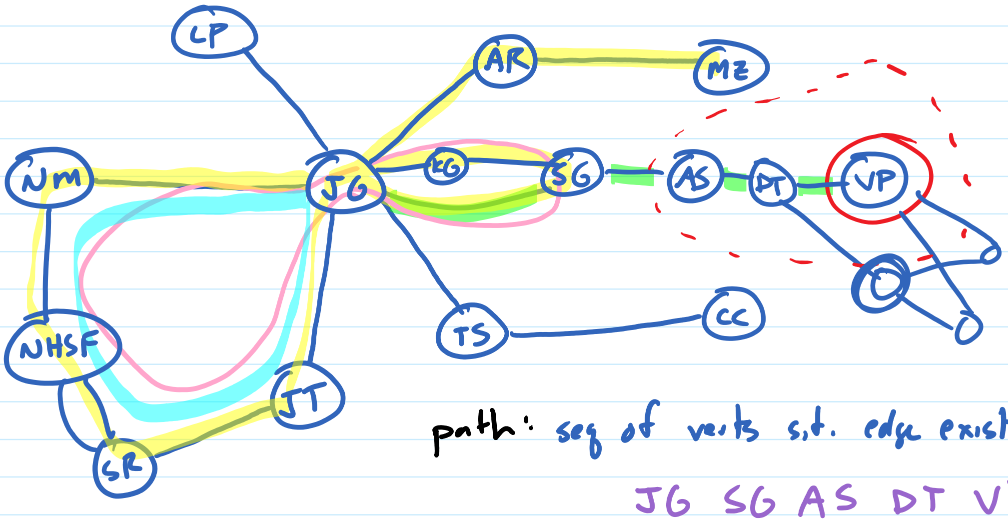
Graphs

↳ representation of things and relationships between them

people (vertices)

relationships (edges)



path: seq of verts s.t. edge exists between adj verts

JG SG AS DT VP

simple graph: at most one copy of edges

no self-loops

simple path: no repeated vertices

cycle : starts/ends at same place

JG NM NHSF SR JT JG

simple cycle: no repeats except beginning/end

path :

simple path:

cycle :

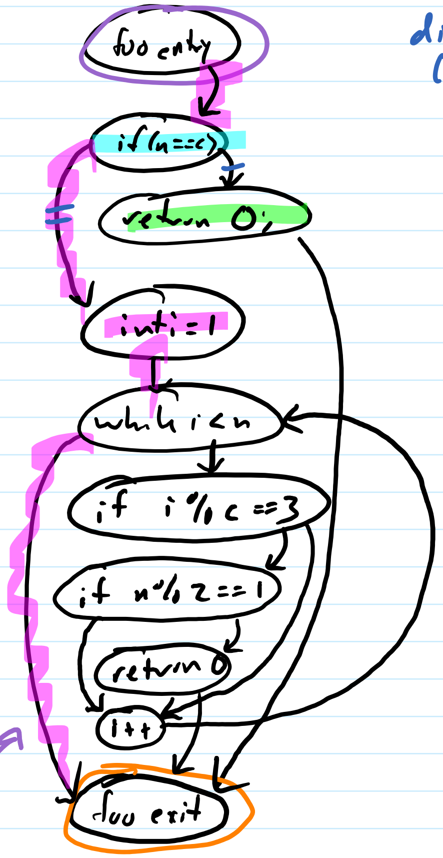
simple cycle:

Flow Control Graph

```
int foo(int n, int c)
{
    if (n == c)
    {
        return 0;
    }
    int i = 1;
    while (i < n)
    {
        if (i % c == 3)
        {
            if (n % 2 == 1)
            {
                return 0;
            }
        }
        i++;
    }
}
```

vertices: lines of code

edges: u → v if v can immediately follow u

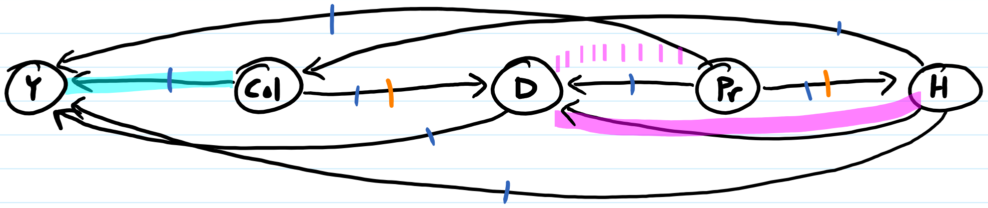


is there a path from beginning to end that doesn't go through return? yes

directed graph (direction is meaningful

can have edge in one dir or other or both or none)

Feedback Arc Set



2 upsets for ranking Y Col D Pr H

vertices teams

0 upsets for Y D Col H P

edges u → v if u lost to v

Columbia lost to Yale

Feedback Arc Set: what is min num edges you need to remove to make graph acyclic (no cycles)

find ranking of teams that minimizes number of upsets

↓ team lower in ranking beat team higher

is there a cycle? [easy]

if not, find ordering so all edges go in same dir [also easy]

if so, find ordering to minimize # of wrong-way edges [hard]

very fast growing

brute force: for each ordering n! possible orderings
count # of wrong-way edges
keep track of ordering giving min-so-far

$\} \; O(n^2 n!)$

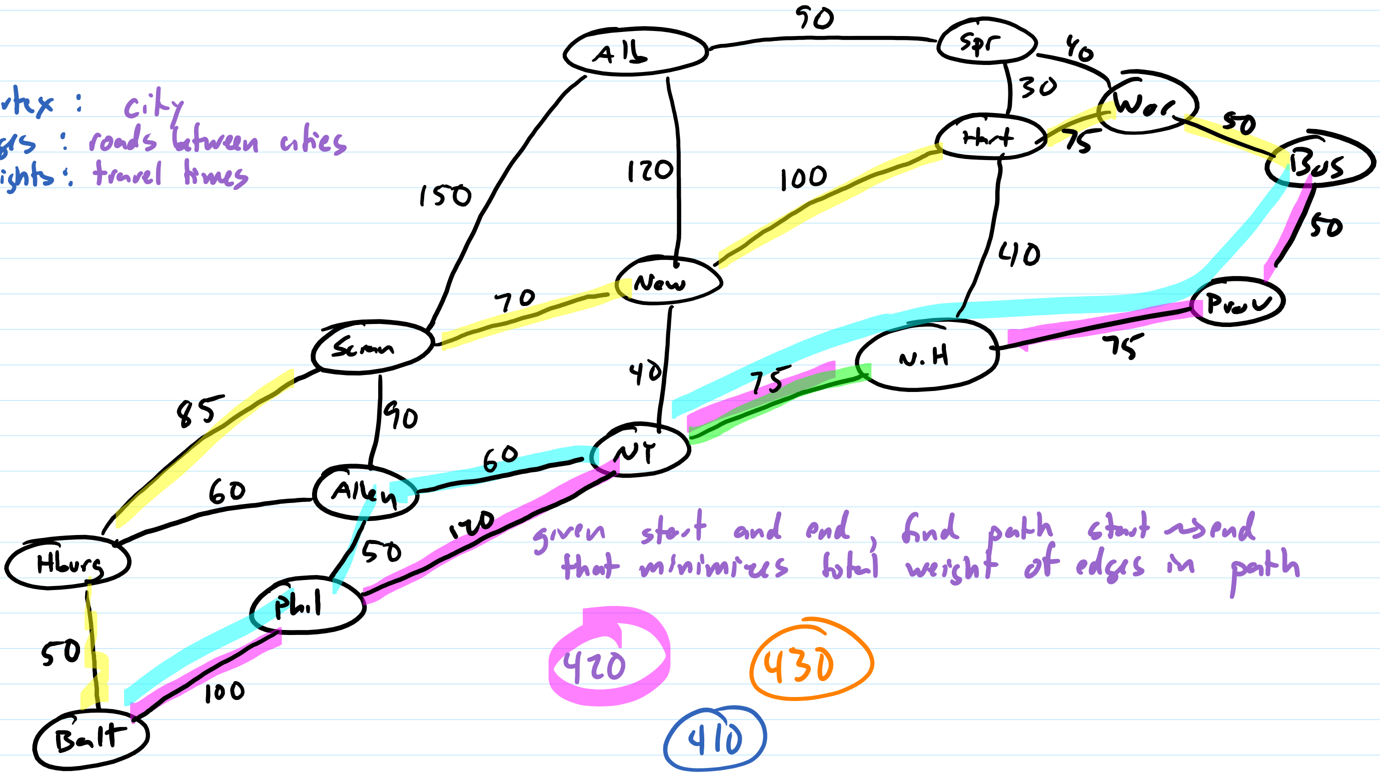
NP-complete (like TSP) : no one has done much better than inefficient brute force
no one has proved it is impossible to get poly-time algorithm

Weighted Graph

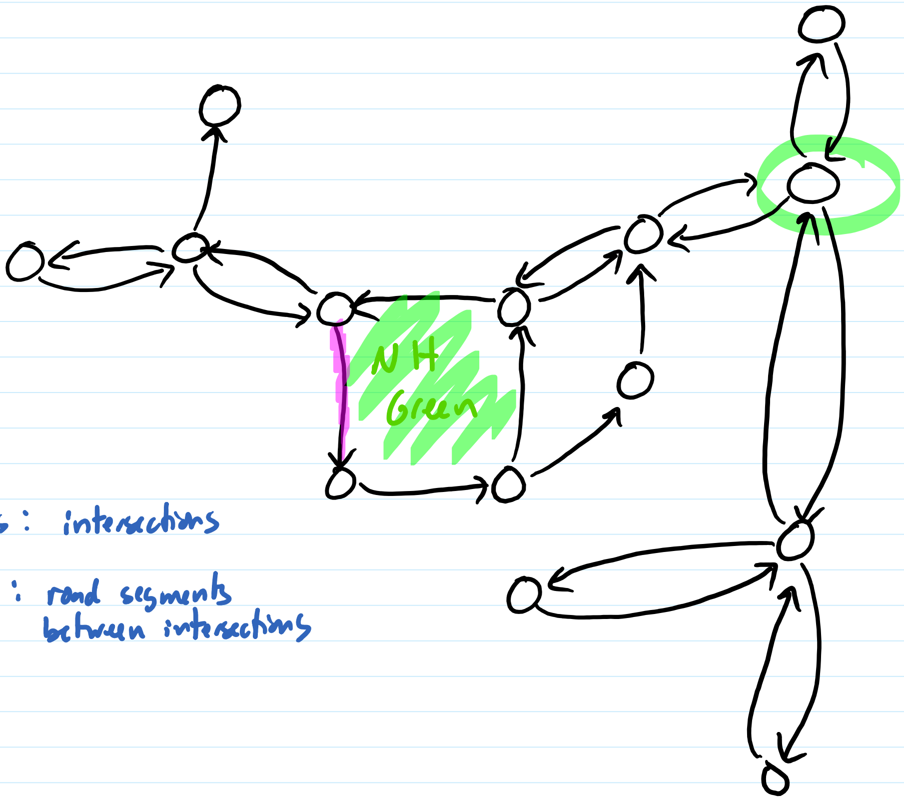
↓

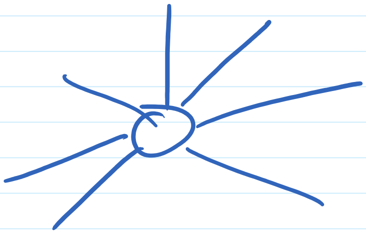
each edge labelled with weight

vertex : city
edges : roads between cities
weights : travel times



given start and end, find path start → end that minimizes total weight of edges in path

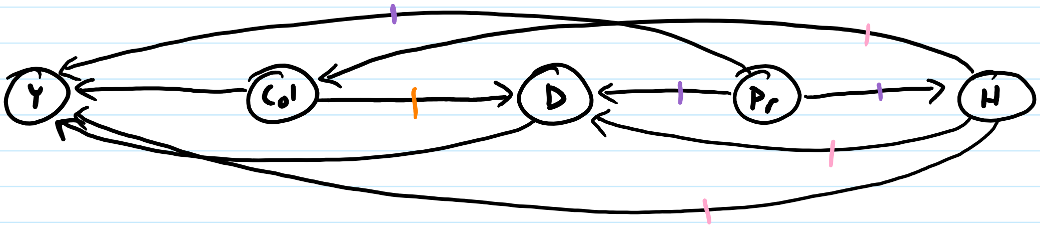




vertices : intersections

edges : road segments
between intersections

Graph Representation



Adjacency Matrix

to

		Y	Col	D	Pr	H
from	Y	F	F	F	F	F
	Col	T	F	T	F	F
	D	T	F	F	F	F
	Pr	T	F	T	F	T
	H	T	T	T	F	F

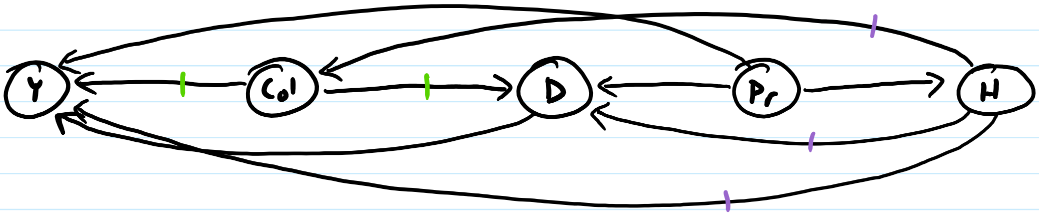
map vertex labels → indices

key	values
Y	0
Col	1
D	2
Pr	3
H	4

expected O(1) time to get indices

O(1) to get element at resulting location

total of expected O(1) to determine if edge exists



Adjacency List

outdegree				
0	0	Y	:	
2	1	Col	:	Y D
1	2	D	:	Y
3	3	Pr	:	Y D H
3	4	H	:	Y Col D

9 = # edges in graph

to determine if edge exists

O(1) expected — translate label to index

O(1) — get list at resulting index

O(n) worst — search list for to vertex

O(n) worst-case

has-edge (Pr, Col)

Adj Set : use hash table for set representation

O(1) expected

O(1)

O(1) expected

O(1) expected

for each edge

adj matrix :
```
for each row r         ] O(n^2) iterations total
  for each col c       ]
    if adj[r][c] == T        } O(1) (assuming O(1) for process edge)
      process edge (r,c)     }
```
$O(n^2)$ total

adj list :
```
for each list u                  O(n) iterations
  [for each v on adj[u]          worst case O(n)
    . process edge (u,v)         O(1)
```

$$\sum_{i=1}^{n} 1 + \sum_{j=1}^{outdegree(i)} c$$

$$\sum_{i=1}^{n}(1 + c \cdot outdegree(i)) = \sum_{i=1}^{n} 1 + c \cdot \sum_{i=1}^{n} outdegree(i)$$

$$= n + \boxed{m}$$ # edges

$O(n^2)$ worst case

$O(n+m)$

$n-1 \leq m \leq n(n-1)$