

# Games, graphs, and machines

m > ma > mat > math > mathematics

↙ soon.

---

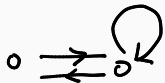
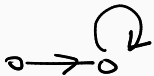
August 13, 2025

# Acyclic graphs

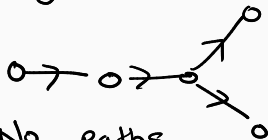
We say that  $G$  is *acyclic* if it has no (directed) cycle.

Suppose  $G$  is acyclic and has 100 vertices. What can you say about  $A^{100}$ ?

Not acyclic



acyclic

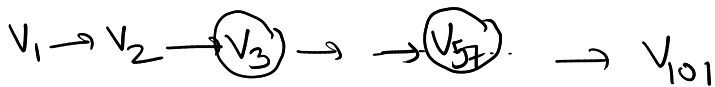


$$A^{100} = 0$$



No paths  
of length 100

Path of length 100 travels through 101 vertices



Must be a repeat (only have 100 vs)



cycle!

$G$  has no cycles and  $n$  vertices

$$\Rightarrow A^n = 0.$$

Suppose  $G$  has 100 vertices and

$$A^{100} = 0.$$

YES

Does this mean  $G$  is acyclic?

If we have a cycle (of any size)

$\Rightarrow$  can create a path of any length.

## Longest path

Let  $G$  be a graph with adjacency matrix  $A$ .

Using  $A$ , how will you find the length of the longest possible path in  $G$ ?

Largest  $n$  s.t.  $A^n \neq 0$ .

# The problem

What is the longest chain of words in the prefix relation?

m → ma → mat → math → mathematics

m → me → met → mete → meteor → meteorite → meteorites

Longer?

# The prefix relation and its graph

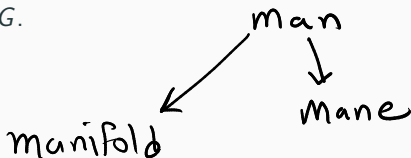
Let us restrict to words beginning with “m”. Let  $W$  be the set of all words beginning with “m”. Consider the graph  $G$  with vertices  $W$  and edges

$$w_1 \rightarrow w_2$$

me  $\rightarrow$  met

if  $w_1$  is a prefix of  $w_2$  and  $w_1 \neq w_2$ .

We want to find the longest path in  $G$ .



# The adjacency matrix

We first order the words.

Here is a list of all words beginning with “m”.

# The adjacency matrix

We first order the words.

Here is a list of all words beginning with “m”.

words

```
['m', 0  
'ma', 1  
'mañana', 2  
'mac', 3  
'macabre',  
'macadam',  
'macadamia',  
'macadamias',  
'macadamize',  
'macadamized',  
'macadamizes',  
'macadamizing',
```

# The adjacency matrix

```
N = len(words)
A = matrix(N,N, sparse=True)
    # the zero matrix

for i in range(0,N):
    for j in range(0,N):
        if (i != j) and words[j].startswith(words[i]):
            A[i,j] = 1
            # change the i,j entry to 1 if i-th word is a
            # prefix of j-th word.
```

*i* → *j*

# Powers of $A$

```
A.is_zero()
```

```
False
```

# Powers of $A$

```
A.is_zero()
```

```
False
```

```
A2 = A*A
```

```
A2.is_zero()
```

```
False
```

# Powers of $A$

```
A.is_zero()
```

```
False
```

```
A2 = A*A
```

```
A2.is_zero()
```

```
False
```

```
A3 = A2*A
```

```
A3.is_zero()
```

```
False
```

# Powers of A

```
A.is_zero()
```

```
False
```

```
A2 = A*A
```

```
A2.is_zero()
```

```
False
```

```
A3 = A2*A
```

```
A3.is_zero()
```

```
False
```

```
A4 = A3*A
```

```
A4.is_zero()
```

```
False
```

# Powers of A

```
A.is_zero()
```

```
False
```

```
A2 = A*A
```

```
A2.is_zero()
```

```
False
```

```
A3 = A2*A
```

```
A3.is_zero()
```

```
False
```

```
A4 = A3*A
```

```
A4.is_zero()
```

```
False
```

```
A5 = A4*A
```

```
A5.is_zero()
```

```
False
```

# Powers of A

A.is\_zero()

False

A2 = A\*A

A2.is\_zero()

False

A3 = A2\*A

A3.is\_zero()

False

A4 = A3\*A

A4.is\_zero()

False

A5 = A4\*A

A5.is\_zero()

False

A6 = A5\*A

A6.is\_zero()

False

# Powers of A

A.is\_zero()

False

A2 = A\*A

A2.is\_zero()

False

A3 = A2\*A

A3.is\_zero()

False

A4 = A3\*A

A4.is\_zero()

False

A5 = A4\*A

A5.is\_zero()

False

A6 = A5\*A

A6.is\_zero()

False

A7 = A6\*A

A7.is\_zero()

False

# Powers of A

A.is\_zero()

False

A2 = A\*A

A2.is\_zero()

False

A3 = A2\*A

A3.is\_zero()

False

A4 = A3\*A

A4.is\_zero()

False

A5 = A4\*A

A5.is\_zero()

False

A6 = A5\*A

A6.is\_zero()

False

A7 = A6\*A

A7.is\_zero()

False

A8 = A7\*A

A8.is\_zero()

False

# Powers of A

A.is\_zero()

False

A2 = A\*A

A2.is\_zero()

False

A3 = A2\*A

A3.is\_zero()

False

A4 = A3\*A

A4.is\_zero()

False

A5 = A4\*A

A5.is\_zero()

False

A6 = A5\*A

A6.is\_zero()

False

A7 = A6\*A

A7.is\_zero()

False

A8 = A7\*A

A8.is\_zero()

False

A9 = A8\*A

A9.is\_zero()

True

# The longest path

How do we actually find the path?

# The longest path

How do we actually find the path?

```
print(A8.nonzero_positions())
```

```
[(0, 981), (0, 2076), (0, 2199)]
```

# The longest paths

```
print(words[0])  
print(words[2199])
```

d

m → mi → min → mini → minim

m

minimalists

↓  
minimal ← minima  
↓  
minimalist  
↓  
minimalists

# The longest paths

```
print(words[0])  
print(words[2076])
```

```
m  
millionairesses
```

# The longest paths

```
print(words[0])  
print(words[981])
```

```
m  
materialistically
```

# What if?

We had not excluded self-loops?

```
N = len(words)
A = matrix(N,N, sparse=True)
    # the zero matrix

for i in range(0,N):
    for j in range(0,N):
        if (i != j) and words[j].startswith(words[i]):
            A[i,j] = 1
            # change the i,j entry to 1 if i-th word is a
            # prefix of j-th word.
```

## What if?

We considered the graph of the Hasse diagram instead of the whole relation? (Only join immediate successors).

## Further questions

1. How to efficiently compute  $A^k$ ?
2. How fast do the entries of  $A^k$  grow as  $k$  grows?