

Games, graphs, and machines

m > ma > mat > math > mathematics

August 16, 2024

The problem

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

$m \rightarrow ma \rightarrow mat \rightarrow math \rightarrow mathematics$

$m \rightarrow me \rightarrow met \rightarrow mete \rightarrow meteor \rightarrow meteorite \rightarrow meteorites$

Longer?

The problem

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

$m \rightarrow ma \rightarrow mat \rightarrow math \rightarrow mathematics$

$m \rightarrow me \rightarrow met \rightarrow mete \rightarrow meteor \rightarrow meteorite \rightarrow meteorites$

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

$$w_1 \rightarrow w_2$$

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

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

$$w_1 \rightarrow w_2$$

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.

We make a list of all words beginning with “m”.

The adjacency matrix

We first order the words.

We make a list of all words beginning with “m”.

words

```
['m',  
 'ma',  
 'mañana',  
 'mac',  
 '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.
```


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])
```

```
m  
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?