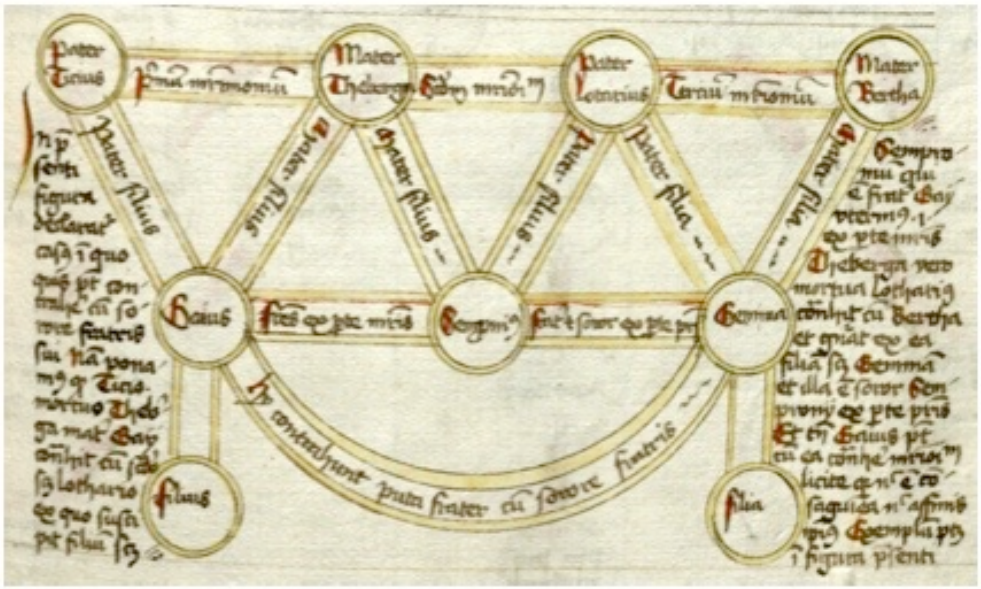
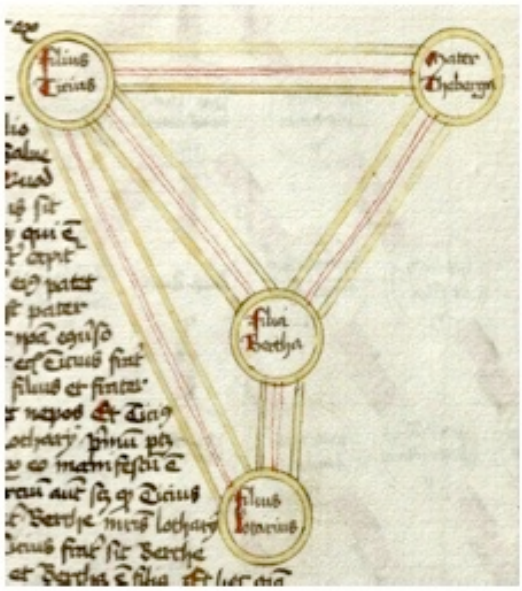
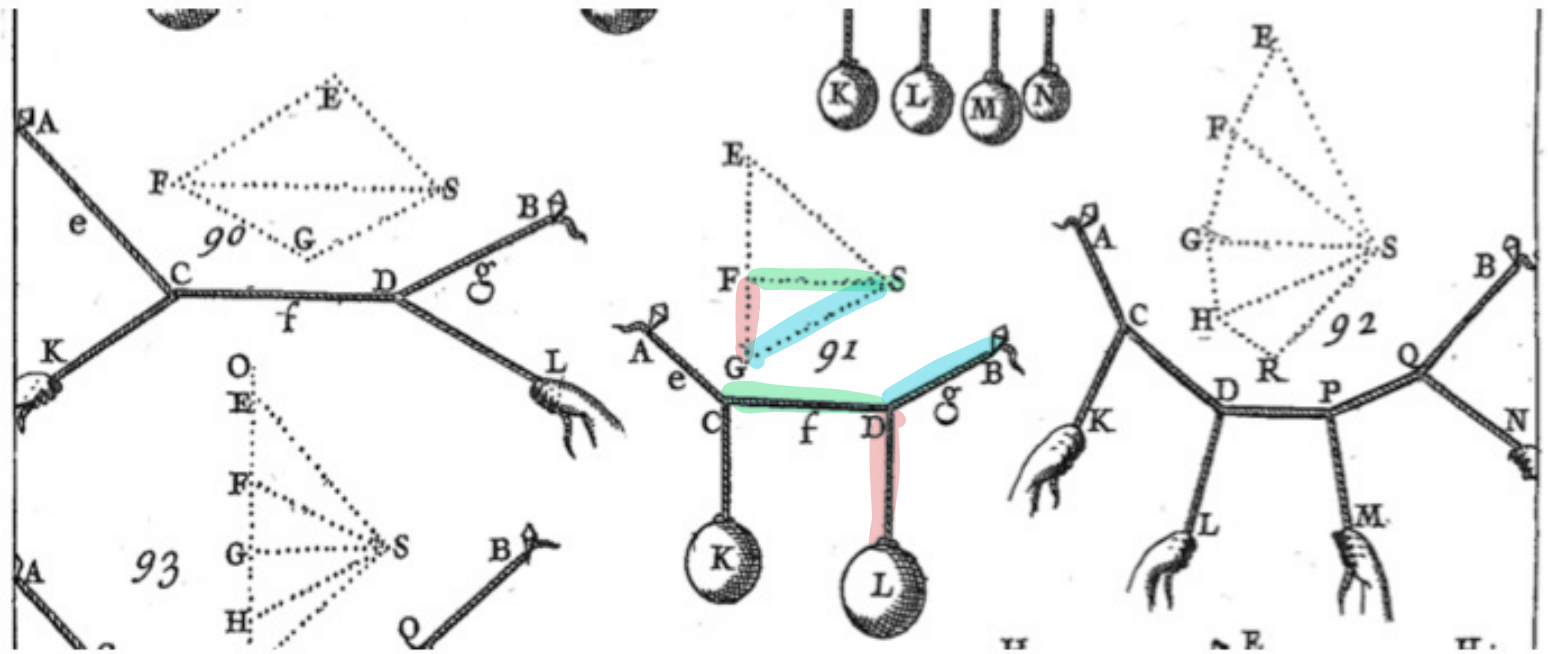
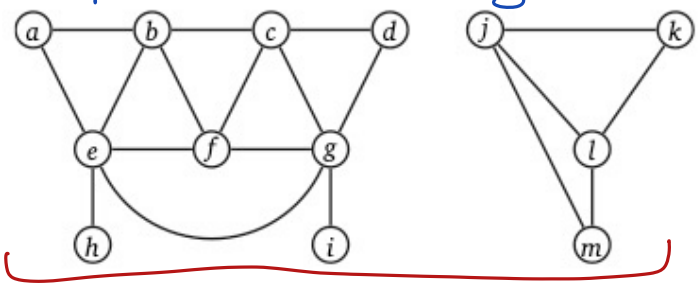
SARMATARUM.
AMAXOBII SARMAT
Tricornio
Monte aureo
Margum
Viminatio
Stanecli
argentaria
MESIA
Salluntio
Sanderua
Halata
Bersumno
Sinna
Epitauro
Aesinum
Viciniuum
Batua
Scobre
Lissum
adpicaria
Creueni
Gabuleo
Pistum
Dyrratio
Hapsum
apollonia
aulona
Pasthum
Balentium
Lupia
ydrunte
Castra Minerue
Balentium
Vihintum
Veretum
Manduris
Hercium
Scammun
Tarento
Mesochoro
Grumento
Heraclea
Semnum
Turis
Petelia
Crontona
Lacenium
Annibali
Cosilianum
Crater
Consentia
Temsa
Herulos
aque ange
tanno
Blanda
Laumium
Cerelis
Clampeia
Temsa
annicia
Vibona Balentia
Tauriana
Scilatio
Scyle
Leucopetra
Regio
Tanno
Strangile
Thermis
Cephaledo
Halesa
Calacte
agatinno
Tindareo
Messana
tauromenio
Simetus
Enna
agurio
Centuripa
aethna
Syracusis
aquas labodes
agrigento
Calusiana
Hible
agris
Port. Tragecyinus
Sabrata
Pontos
assaria
ad cypsaria Taberna
ad amonem
Pisida Municipio
Rufini Taberna
Tipasa
Haribus
Putea Pallene
fl. Ausere
BAGIGETULI
dugarmi
Ausere
Putea
Lamnine
Veri
fl. Girin


Pater Tetus
Mater Thebergis
Pater Lotharius
Mater Bertha
Tetus
Gemma
filius
filia

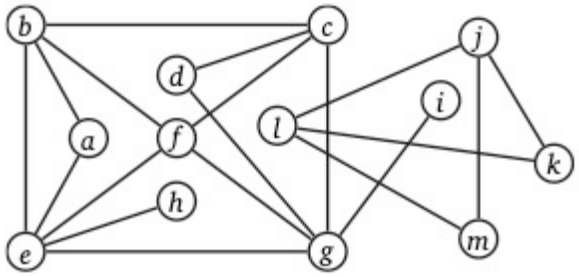

filius Tetus
Mater Thebergis
filia Bertha
filius Lotharius

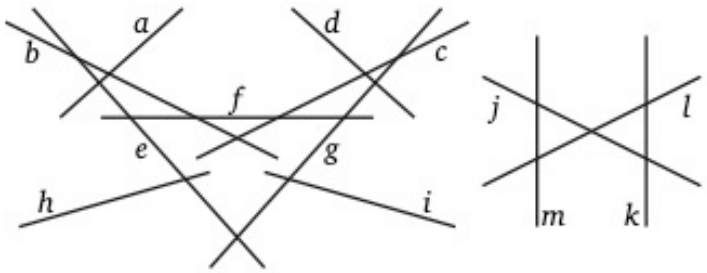

90
91
92
93

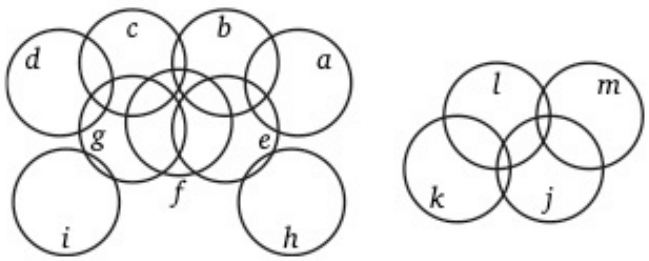
planar embeddings



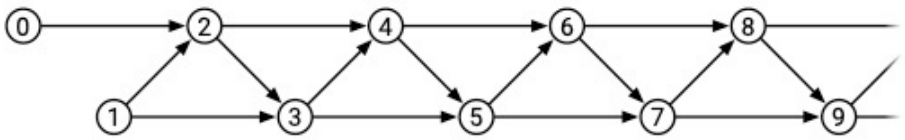
13 vertices
edges



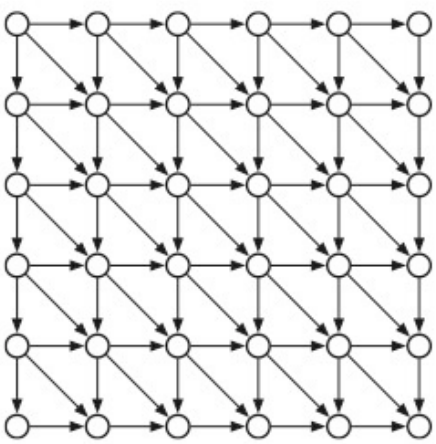




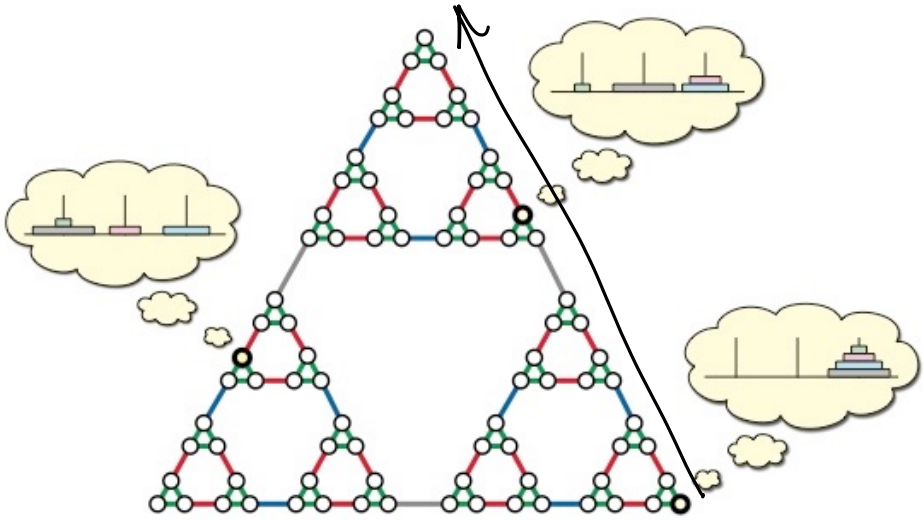
$$F_n = \begin{cases} 0 & \text{if } n = 0, \\ 1 & \text{if } n = 1, \\ F_{n-1} + F_{n-2} & \text{otherwise,} \end{cases}$$

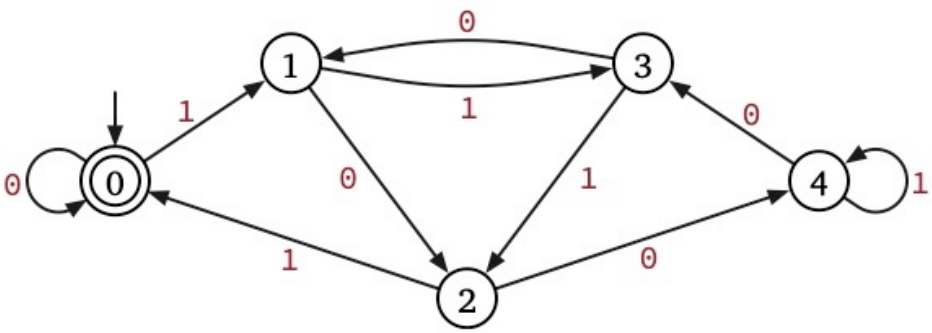


$$Edit(i,j) = \begin{cases} i & \text{if } j = 0 \\ j & \text{if } i = 0 \\ \min \begin{cases} Edit(i-1,j)+1 \\ Edit(i,j-1)+1 \\ Edit(i-1,j-1)+[A[i] \neq B[j]] \end{cases} & \text{otherwise} \end{cases}$$



directed acyclic graphs





NFA → DFA conversion
incremental subset

Vertices?

Edges?

Problem?

Algo?

Time?

Graph = (V, E)

V = vertices
ANY nonempty
Finite set

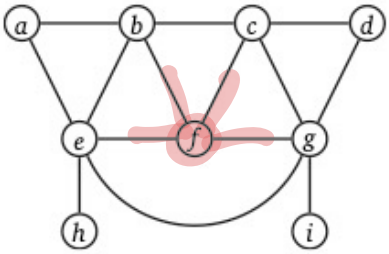
E = edges
= pairs of vertices

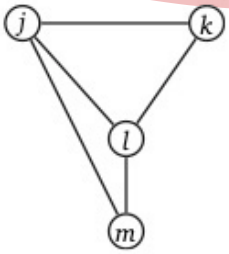
$\{u,v\} = uv$ undirected

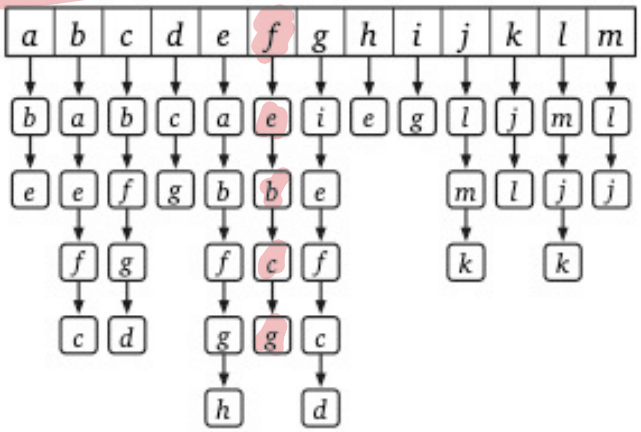
$(u,v) = u \to v$ directed

DATA STRUCTURES

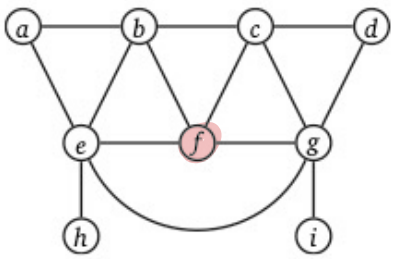
"adjacency list"

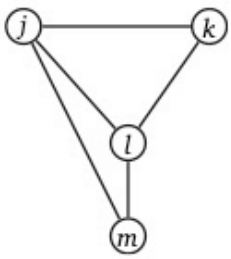






adjacency matrix





	a	b	c	d	e	f	g	h	i	j	k	l	m
a	0	1	0	0	1	0	0	0	0	0	0	0	0
b	1	0	1	0	1	1	0	0	0	0	0	0	0
c	0	1	0	1	0	1	1	0	0	0	0	0	0
d	0	0	1	0	0	0	1	0	0	0	0	0	0
e	1	1	0	0	0	1	1	1	0	0	0	0	0
f	0	1	1	0	1	0	1	0	0	0	0	0	0
g	0	0	1	1	1	1	0	0	1	0	0	0	0
h	0	0	0	0	1	0	0	0	0	0	0	0	0
i	0	0	0	0	0	0	1	0	0	0	0	0	0
j	0	0	0	0	0	0	0	0	0	0	1	1	1
k	0	0	0	0	0	0	0	0	0	1	0	1	0
l	0	0	0	0	0	0	0	0	0	1	1	0	1
m	0	0	0	0	0	0	0	0	0	1	0	1	0

Reachability

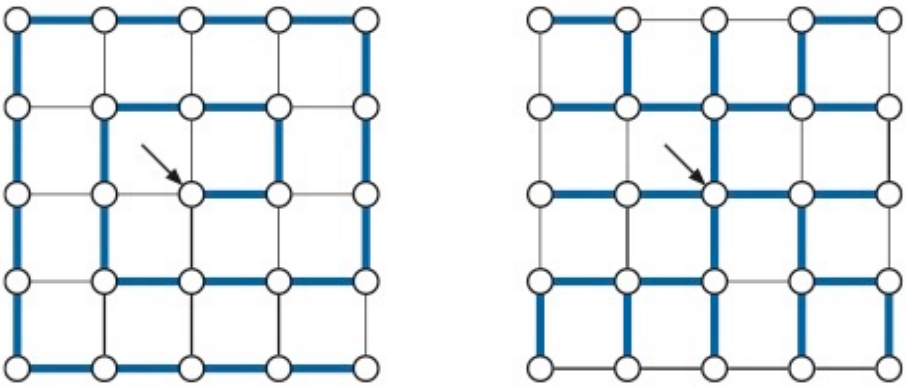
v can reach w ⇔ there is a path from v to w.

```
RecursiveDFS(v):
  if v is unmarked
      mark v
      for each edge vw
          RecursiveDFS(w)
```

```
IterativeDFS(s):
  Push(s)
  while the stack is not empty
      v ← Pop
      if v is unmarked
          mark v
          for each edge vw
              Push(w)
```

```
WhateverFirstSearch(s):
  put s into the bag
  while the bag is not empty
      take v from the bag
      if v is unmarked
          mark v
          for each edge vw
              put w into the bag
```

```
WhateverFirstSearch(s):
  put (∅, s) in bag
  while the bag is not empty
      take (p, v) from the bag                (⋆)
      if v is unmarked
          mark v
          parent(v) ← p
          for each edge vw                    (†)
              put (v, w) into the bag         (⋆⋆)
```



```
WFSALL(G):
  for all vertices v
      unmark v
  for all vertices v
      if v is unmarked
          WHATEVERFIRSTSEARCH(v)
```

```
COUNTCOMPONENTS(G):
  count ← 0
  for all vertices v
      unmark v
  for all vertices v
      if v is unmarked
          count ← count + 1
          WHATEVERFIRSTSEARCH(v)
  return count
```

```
COUNTANDLABEL(G):
  count ← 0
  for all vertices v
      unmark v
  for all vertices v
      if v is unmarked
          count ← count + 1
          LABELONE(v, count)
  return count
```

```
⟨⟨Label one component⟩⟩
LABELONE(v, count):
  while the bag is not empty
      take v from the bag
      if v is unmarked
          mark v
          comp(v) ← count
          for each edge vw
              put w into the bag
```