The dictionary ADT

CS 146 - Spring 2017

Review question

- backtracking problem

Announcement

- Exam 1 scheduled for T 3/14 (4 more lectures)

- covers hw 1 to 4

Study recommendations

- master concepts/skills listed in homework descriptions

- start with a self-evaluation of this list,

- redo class examples, homework problems, additional practice problems under exam conditions

- but don’t memorize, understand

Today

- the dictionary ADT
- applications
- graphs
- caching and memoization

Dictionary ADT

- used to maintain a set of **entries or (key, value) pairs**
- insert(entry)
- delete(entry)
- search(key)
- values have distinct keys

The dictionary in Java

- `Map` interface
- implemented with `TreeMap`, `HashMap`
- ex: `Map<String, Integer> m = new TreeMap<>();`

Dictionaries are very useful

- databases (for storing actual data)
  - word → definition
  - word → pages containing that word
  - username → account
- compilers and interpreters: names → variables

Dictionaries are very useful

- network routers
  IP address → wire
- operating system
  virtual memory use
  page tables
  virtual address →
  physical addresses

Dictionaries are very useful

- key value stores for caching large amounts of web-data (html webpage, css, images): url → file

- any caching technique

Dictionaries in this class

- memoization
  input→output

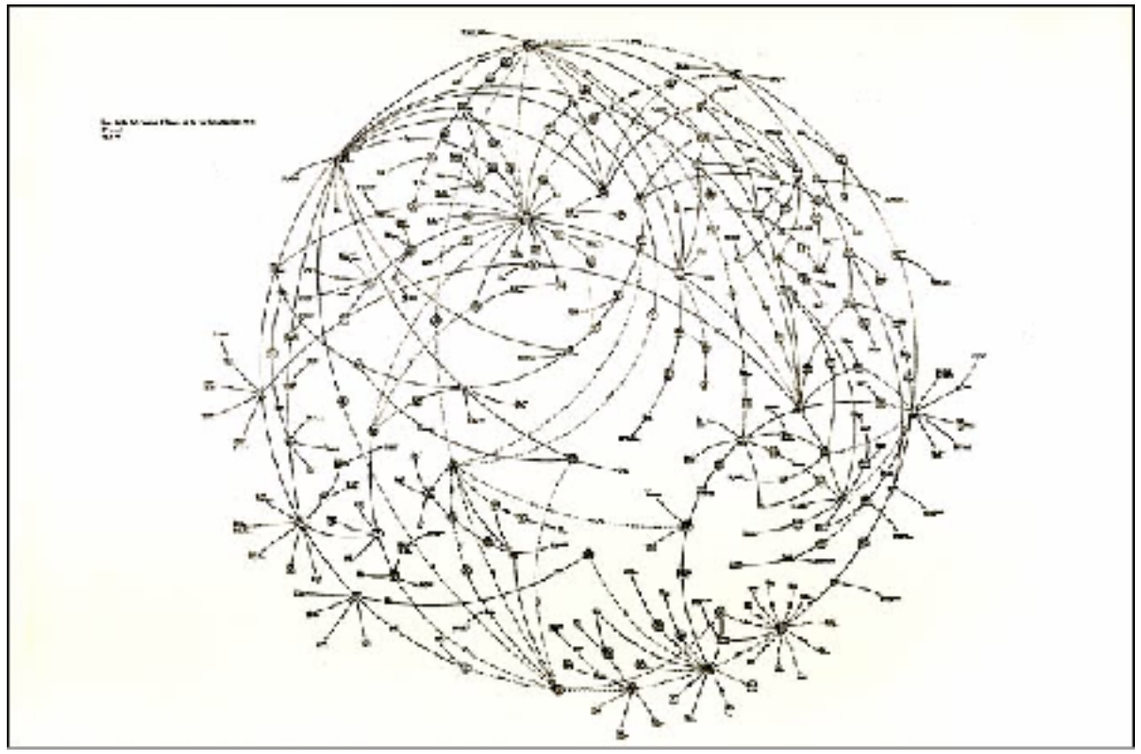
- graphs
  node→list of neighbors

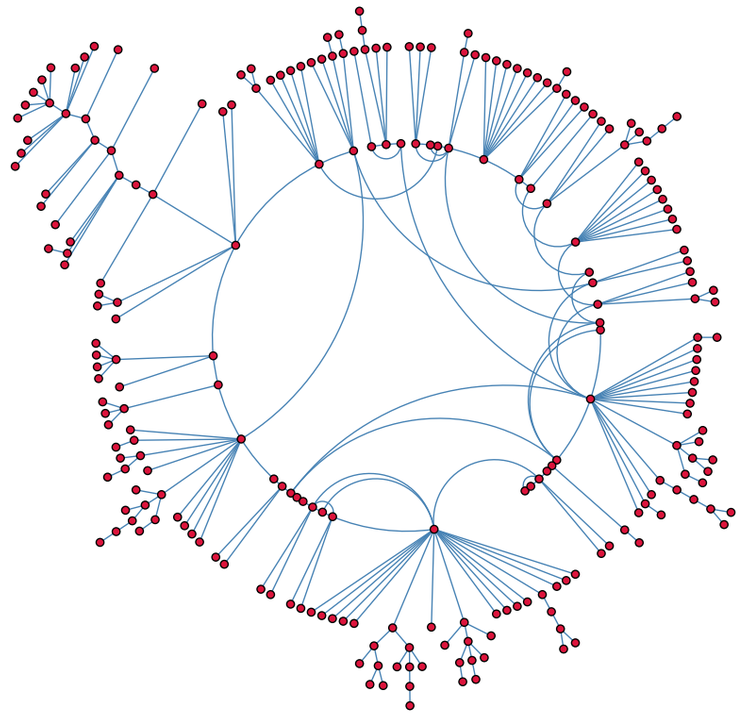
Graphs

Informally Graphs

A graph consists of a collection of entities together with a binary “relation”.

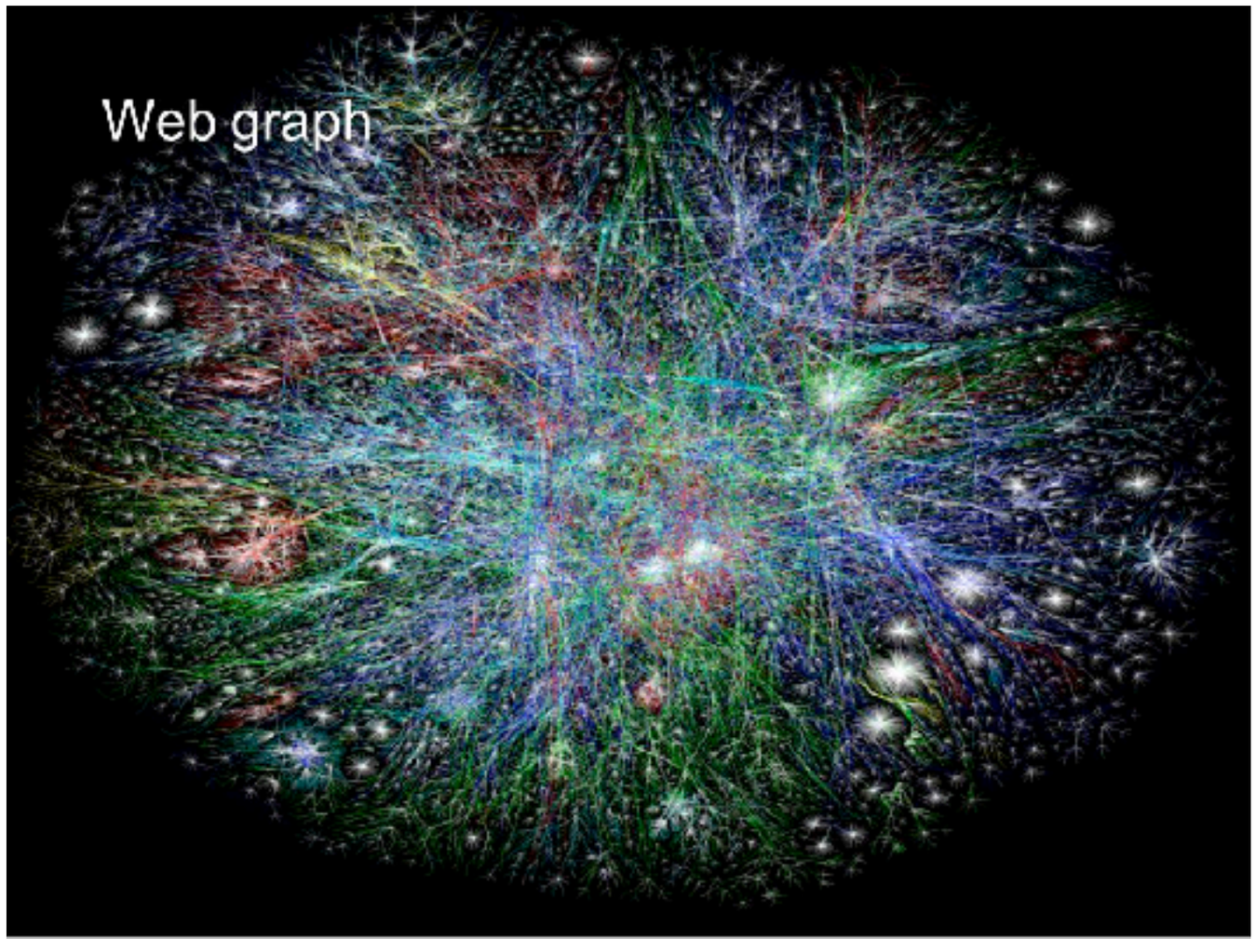
Social Interaction Graphs

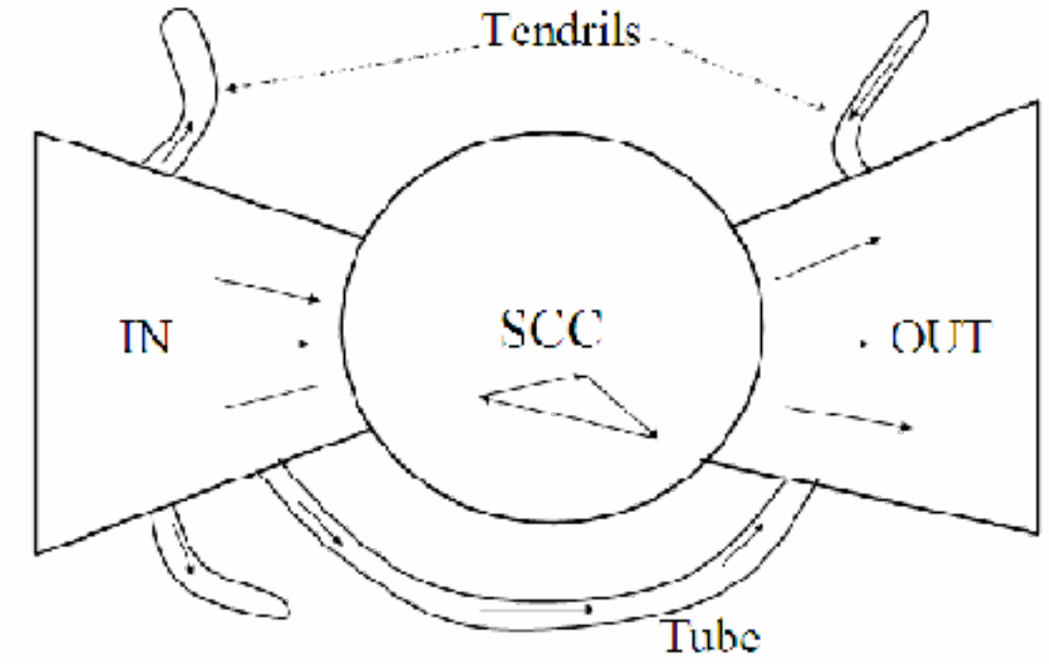




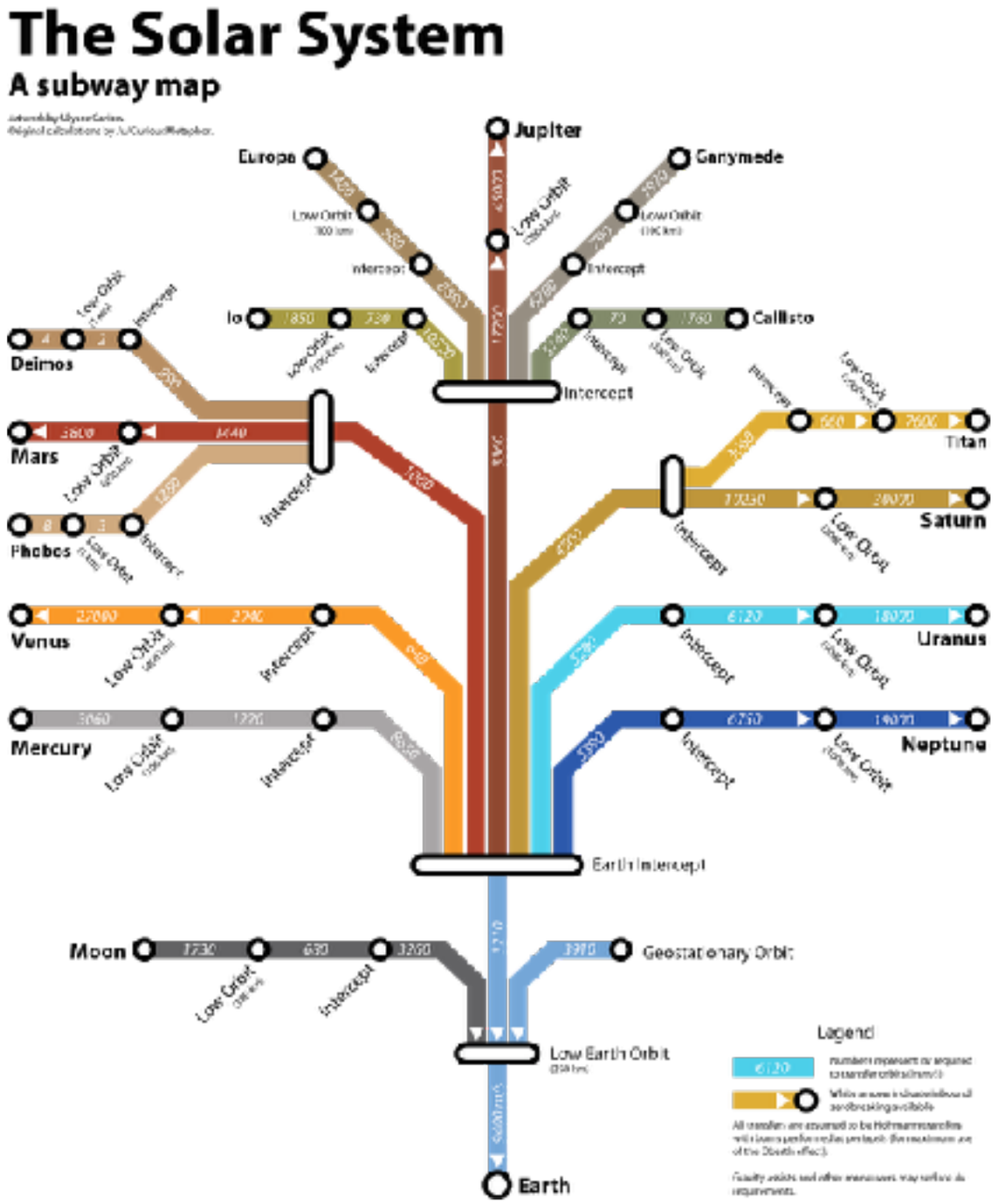
Communication Networks

Webgraph

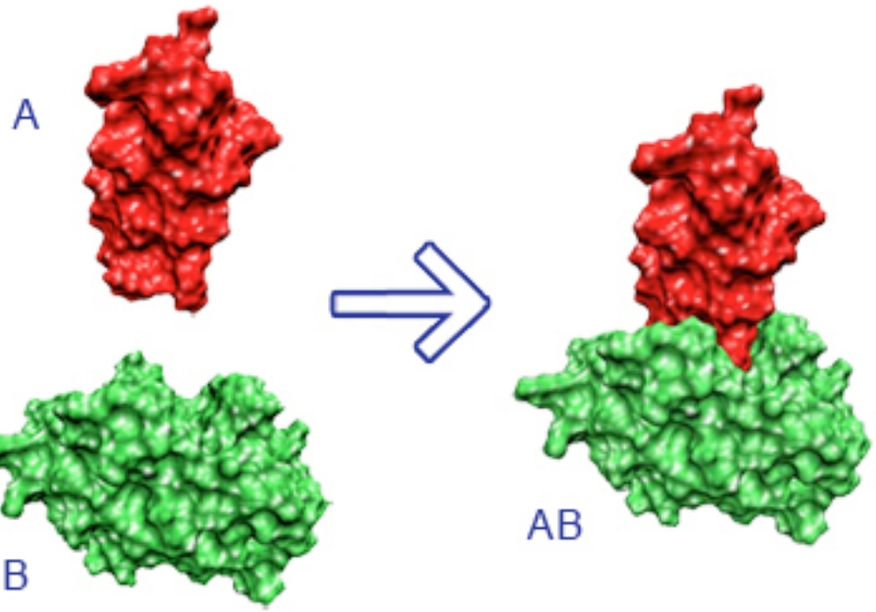


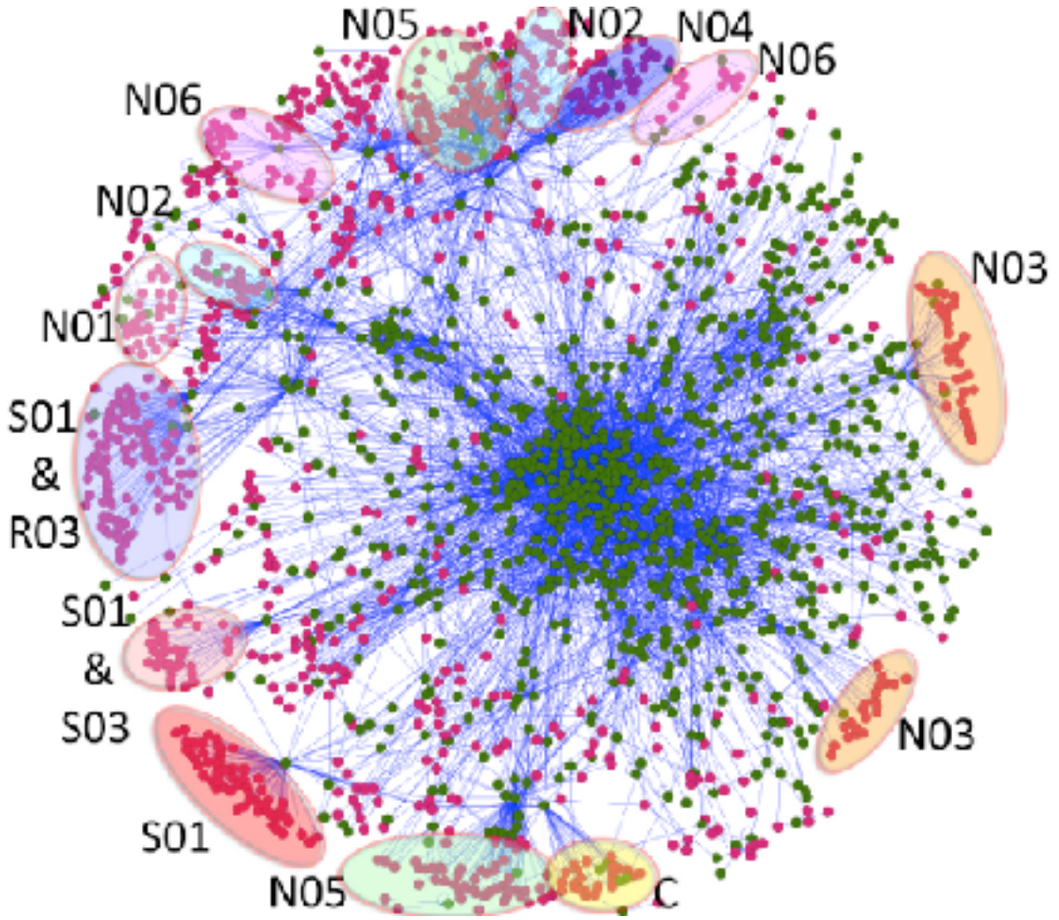


Transportation Networks

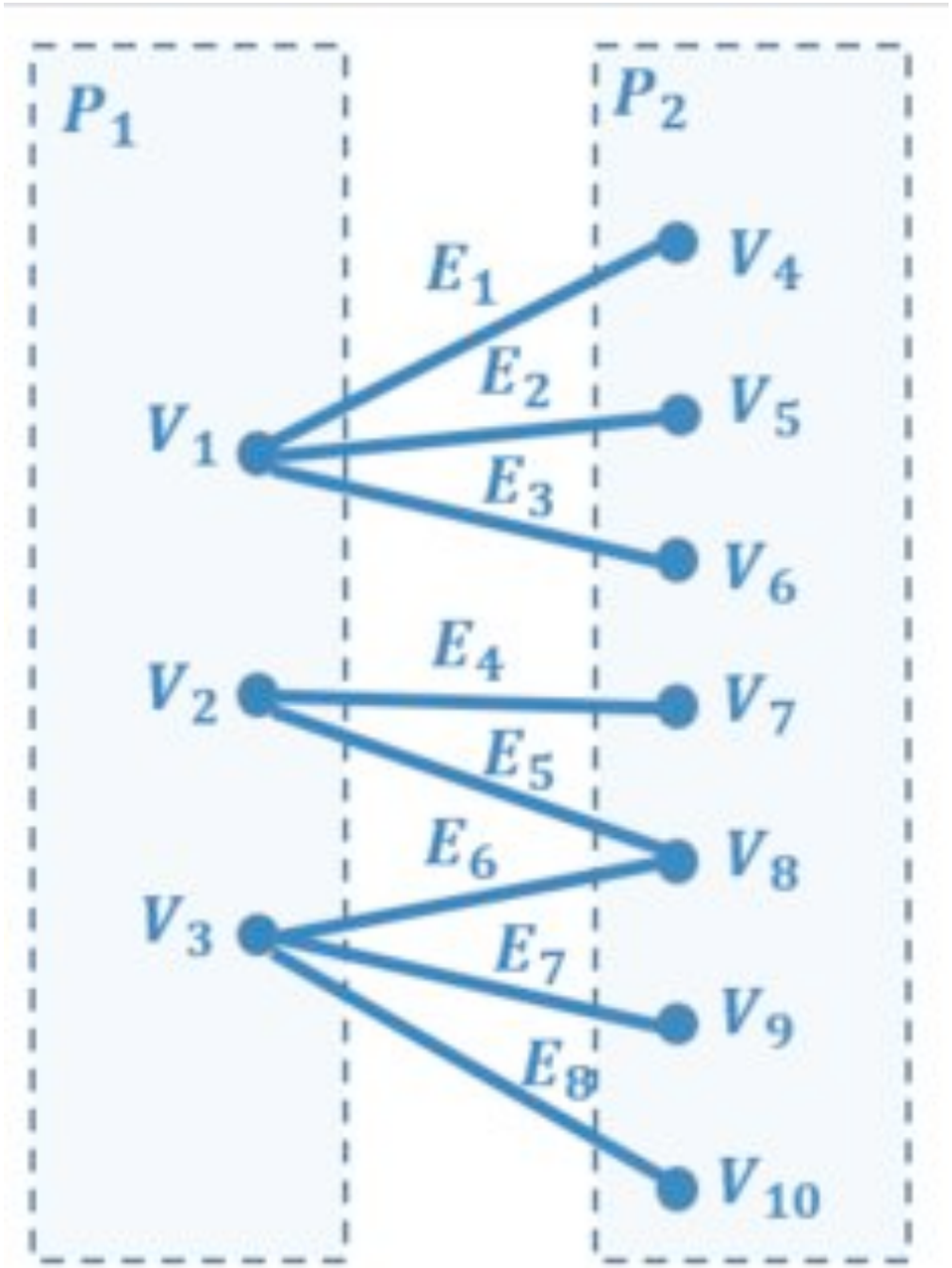


Protein-protein interaction networks

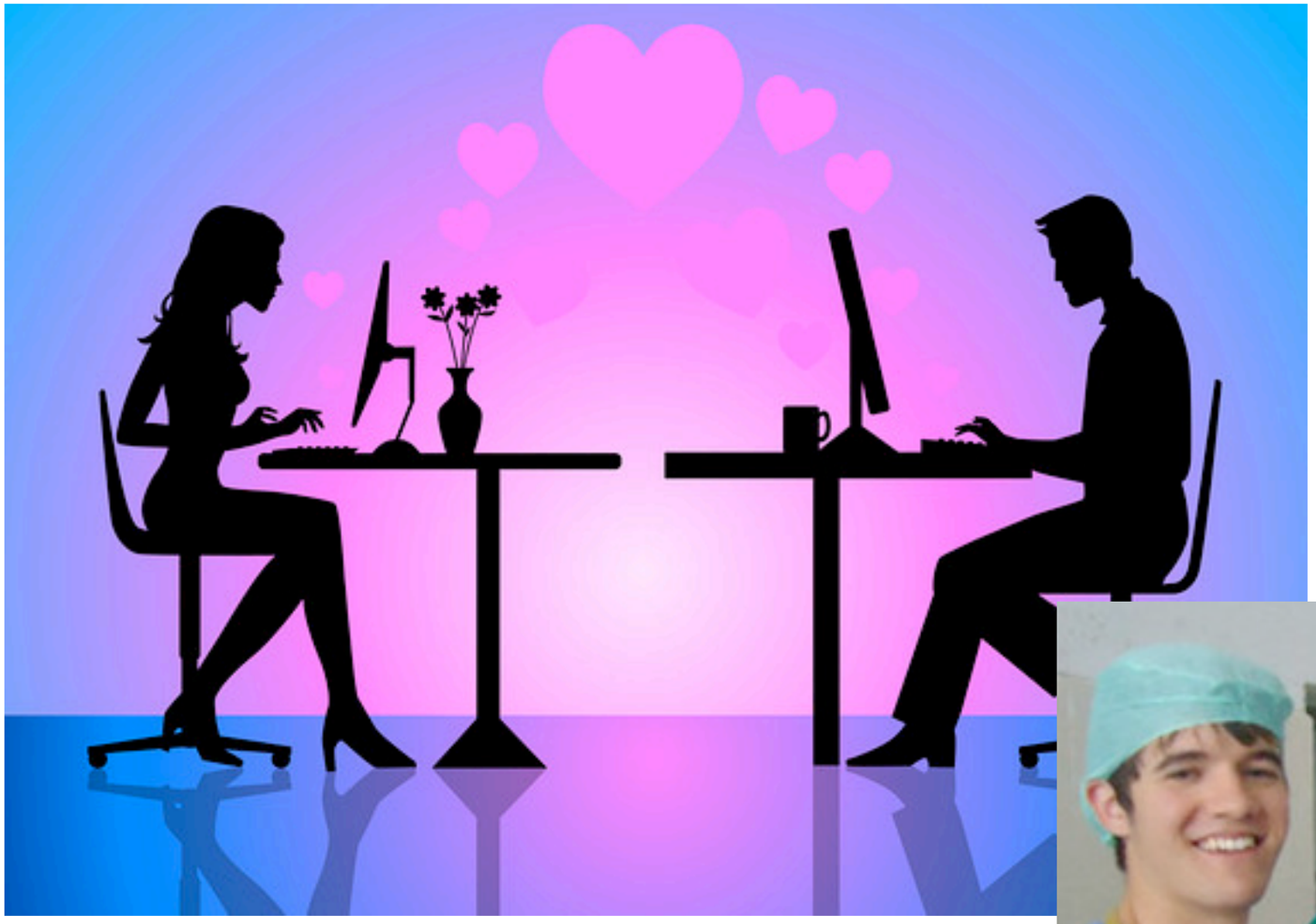




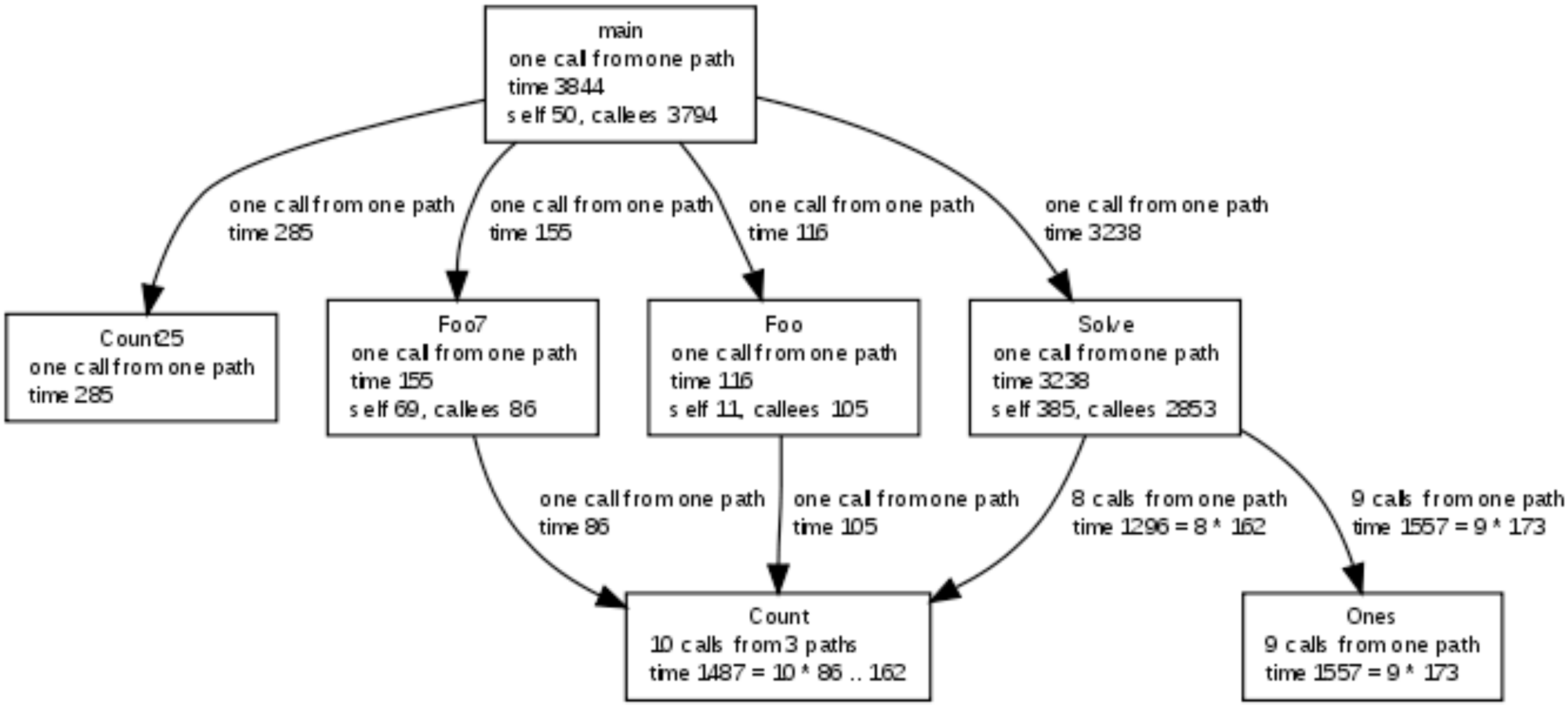
Advertising



Stable matching

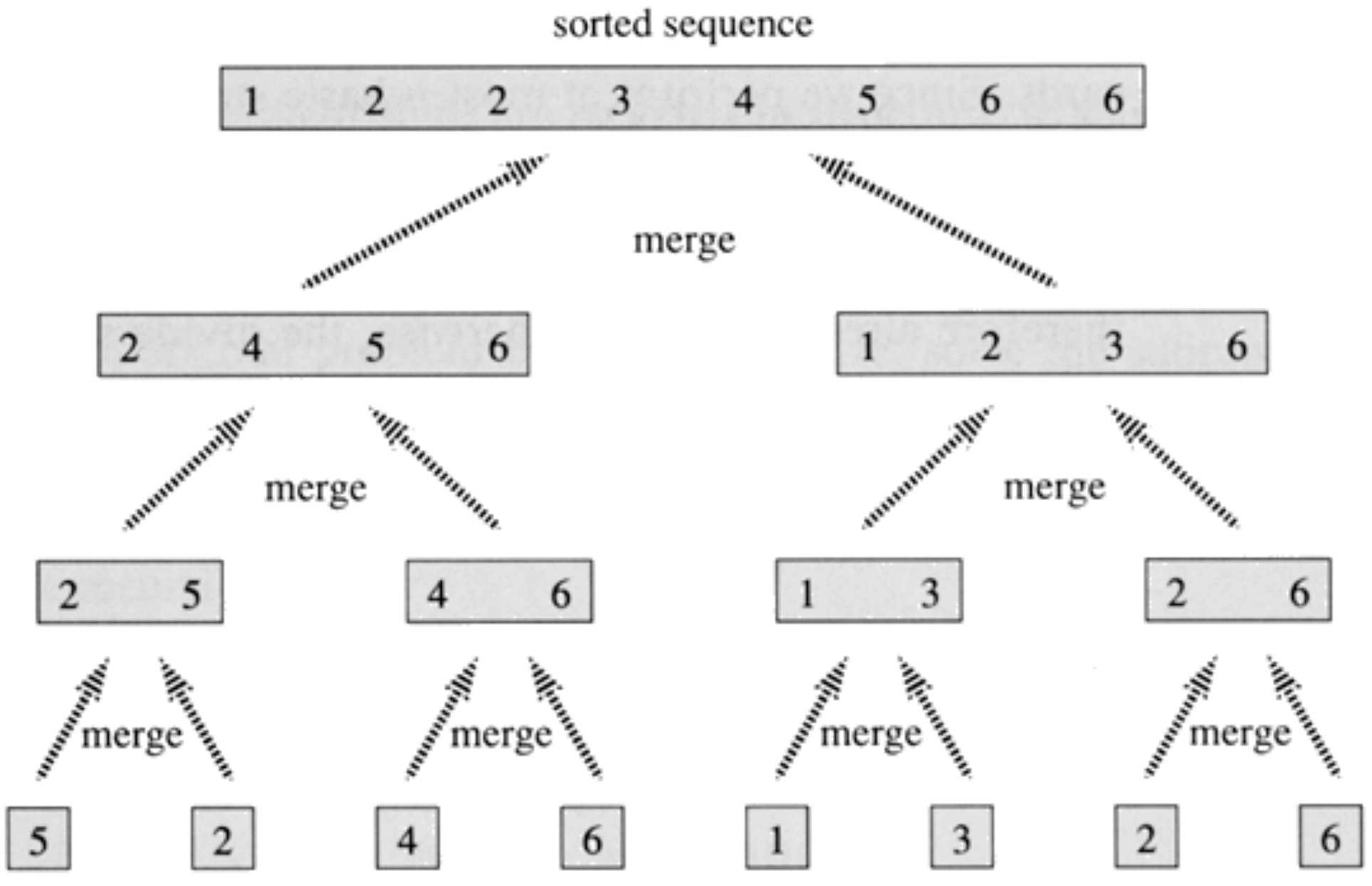


Call graph

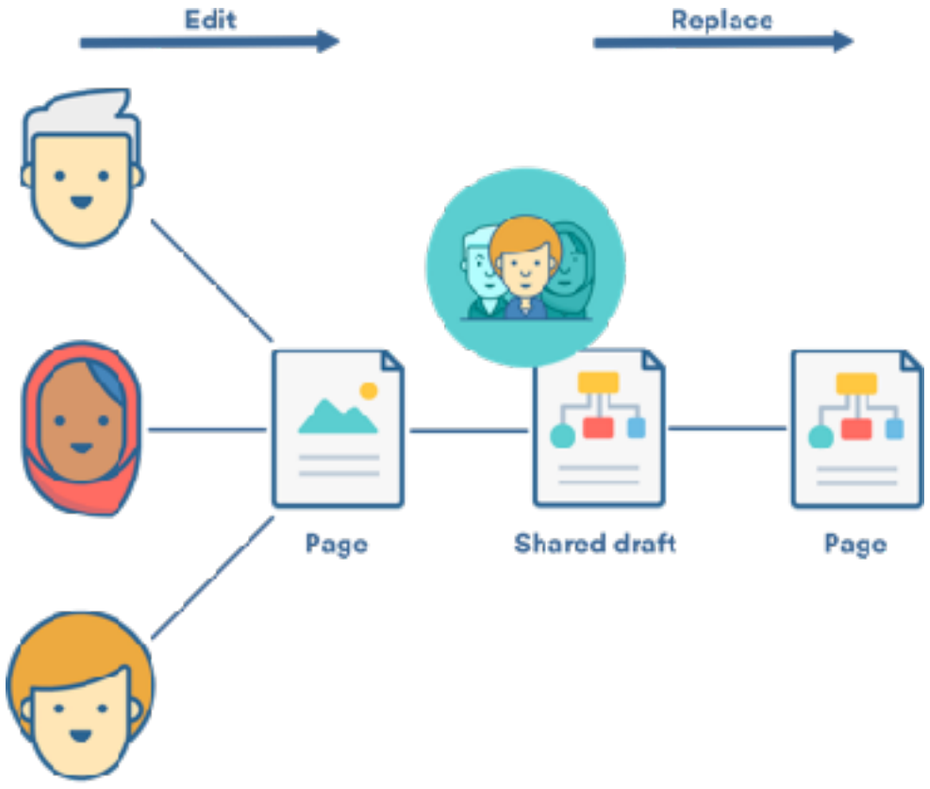


Call graph of main
time 3844

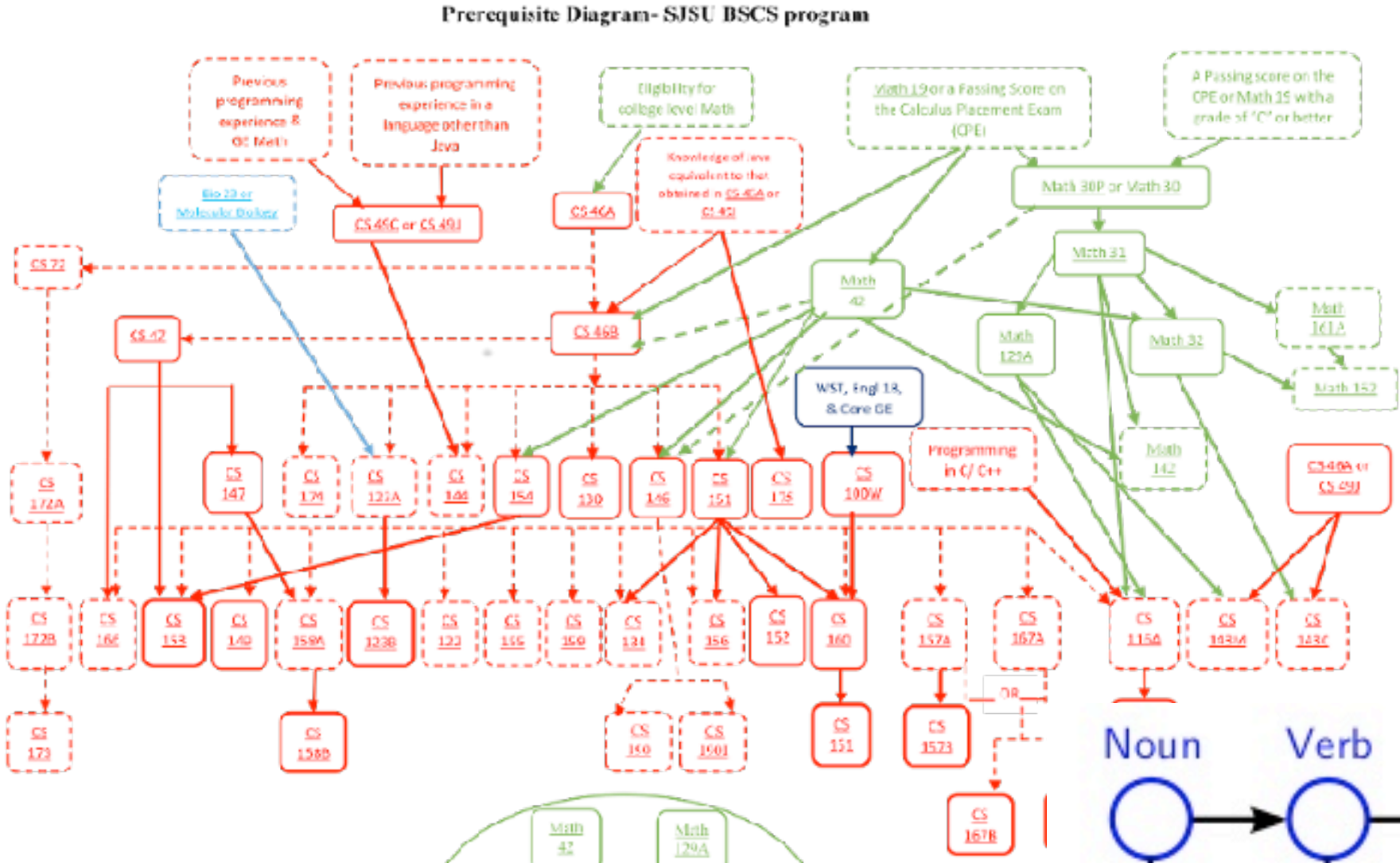
Recursion tree



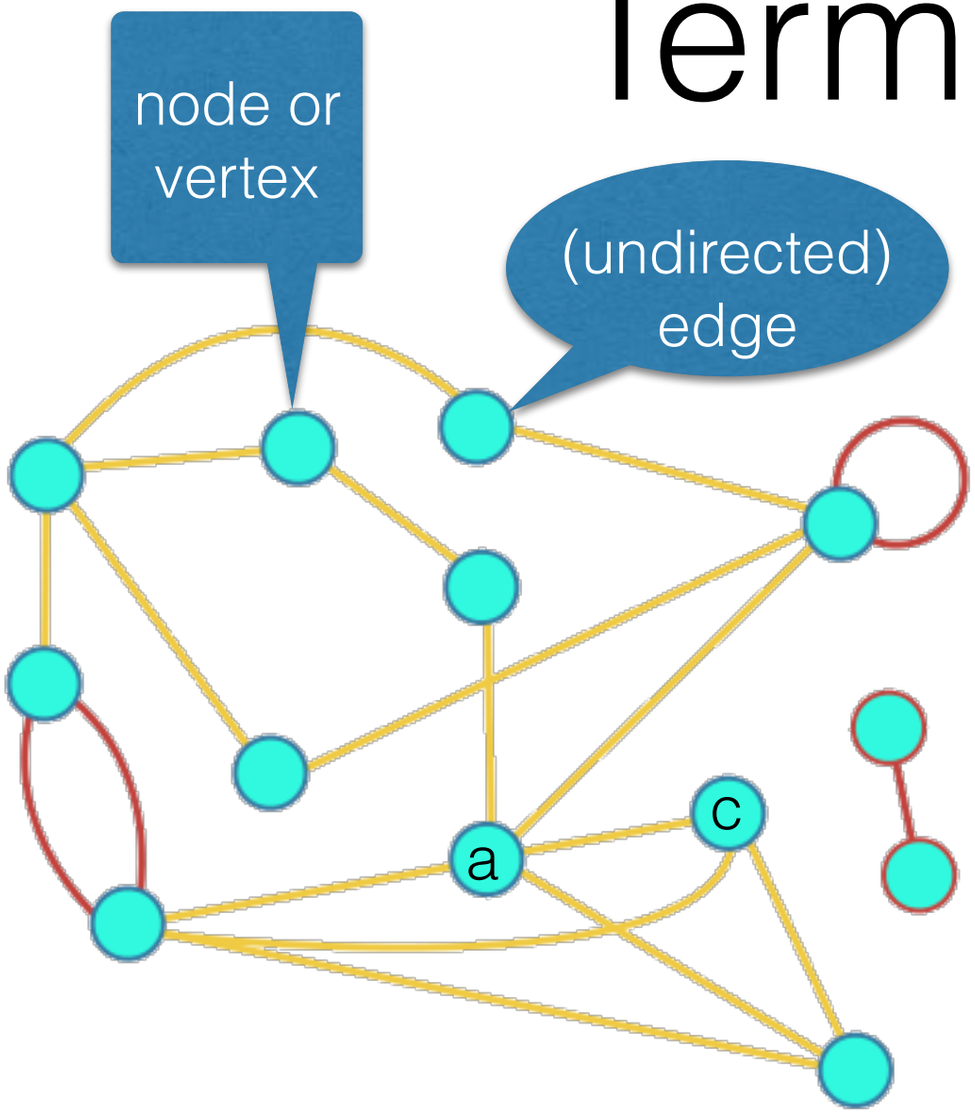
Version control



Dependencies and inference

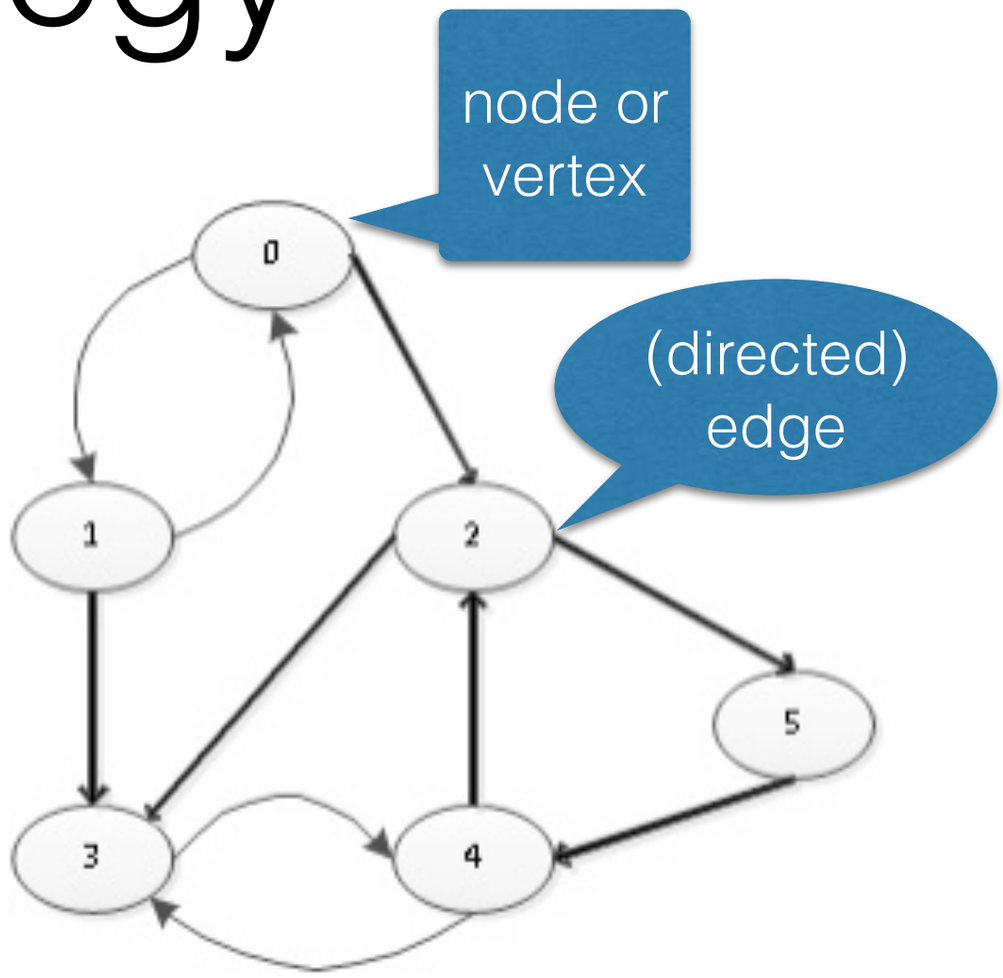


Terminology



undirected

a is a neighbor of c
c is a neighbor of a

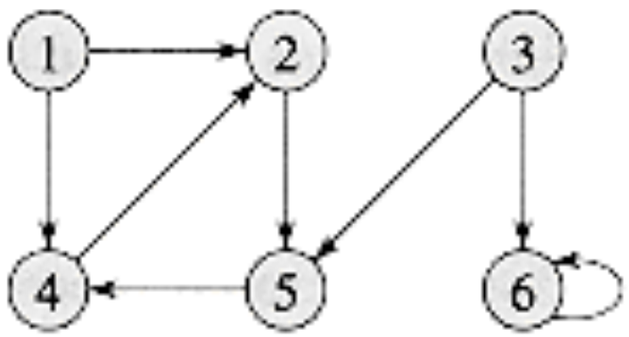


directed

3 is a neighbor of 2
2 is not a neighbor of 3

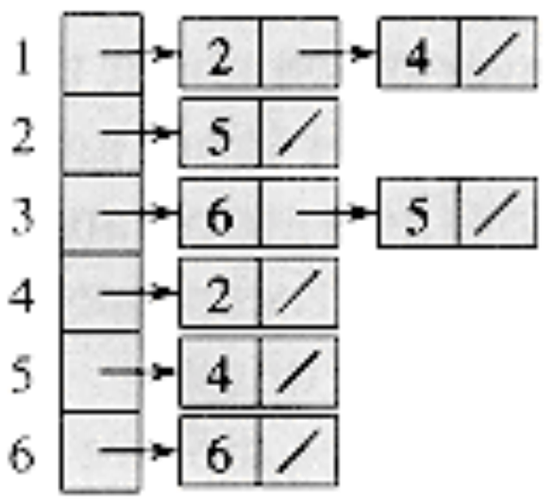
2→3

Computer representation of graphs



(a)

directed graph



(b)

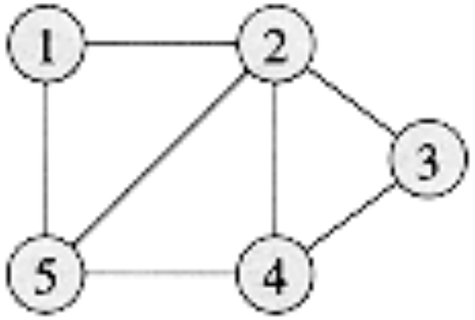
adjacency list

	1	2	3	4	5	6
1	0	1	0	1	0	0
2	0	0	0	0	1	0
3	0	0	0	0	1	1
4	0	1	0	0	0	0
5	0	0	0	1	0	0
6	0	0	0	0	0	1

(c)

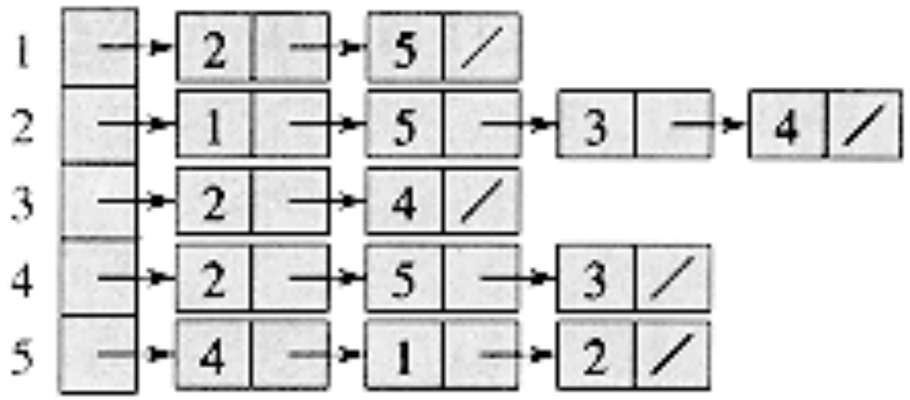
adjacency matrix

Computer representation of graphs



(a)

undirected graph



(b)

adjacency list

	1	2	3	4	5
1	0	1	0	0	1
2	1	0	1	1	1
3	0	1	0	1	0
4	0	1	1	0	1
5	1	1	0	1	0

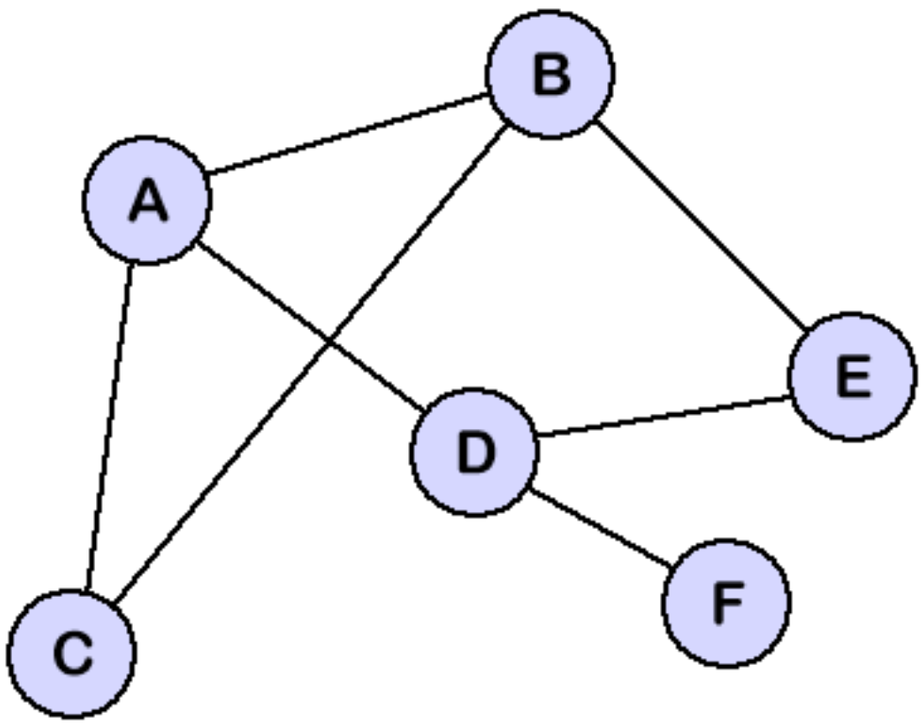
(c)

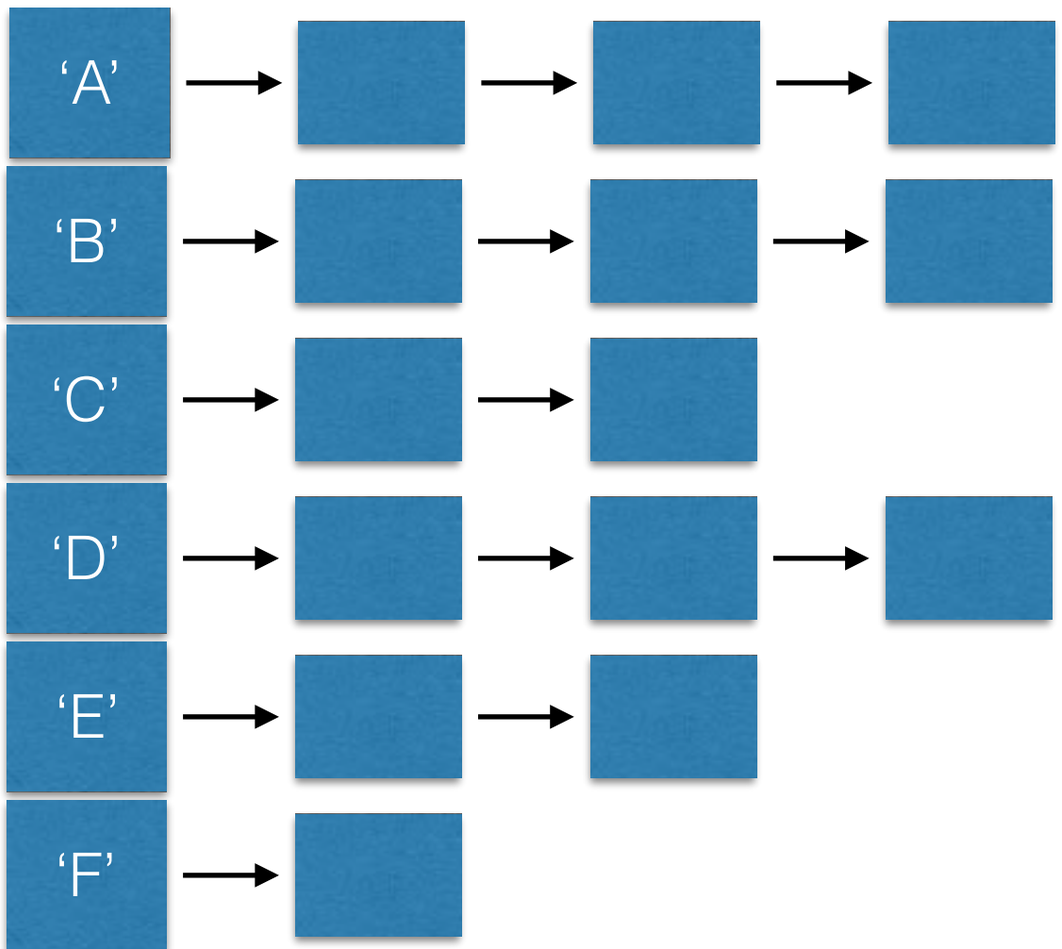
adjacency matrix

Dictionary representation

- same as adjacency list, but does not require indexing the vertices

- vertices can be easily inserted and deleted





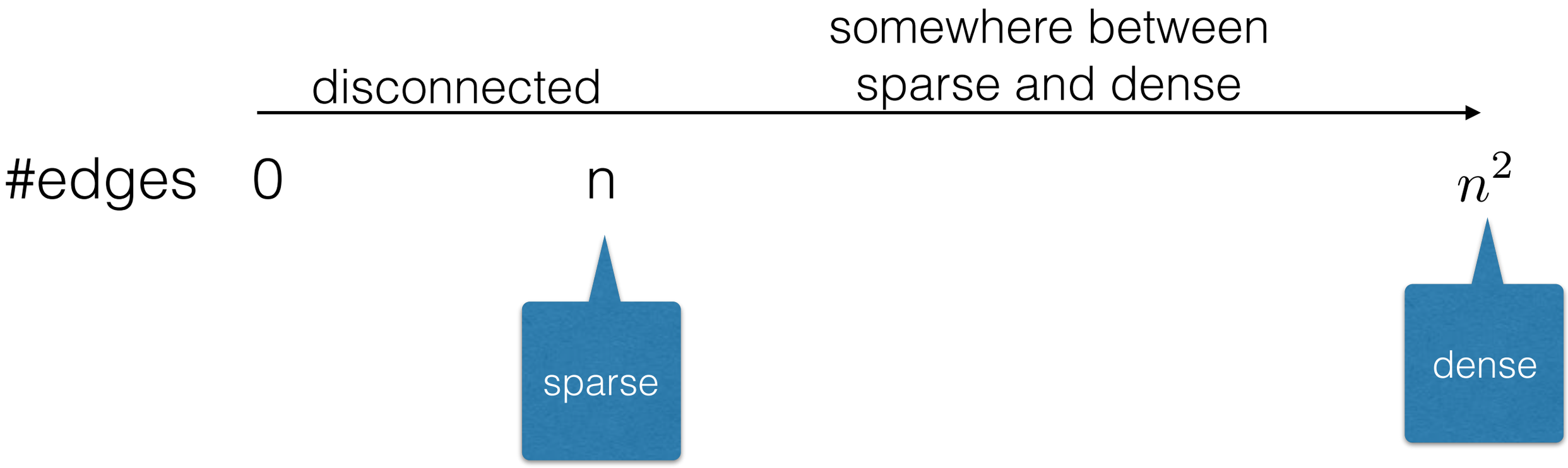
Which representation is better?

- depends on the **types of queries** one needs to do
  - is v a neighbor of w?
  - get all the neighbors of v
- depends on the **density** of the graph

Graph density

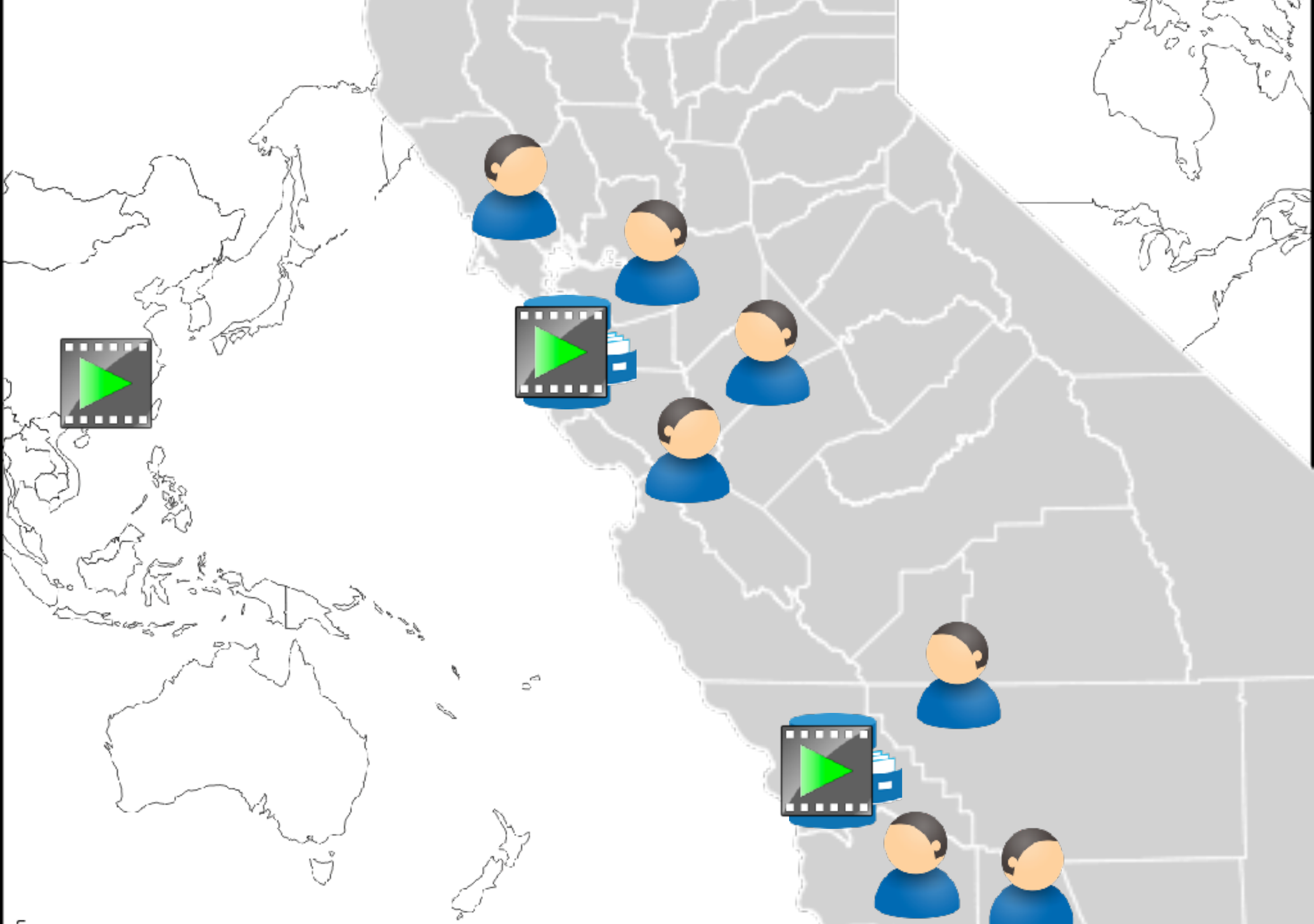
given a graph on n vertices

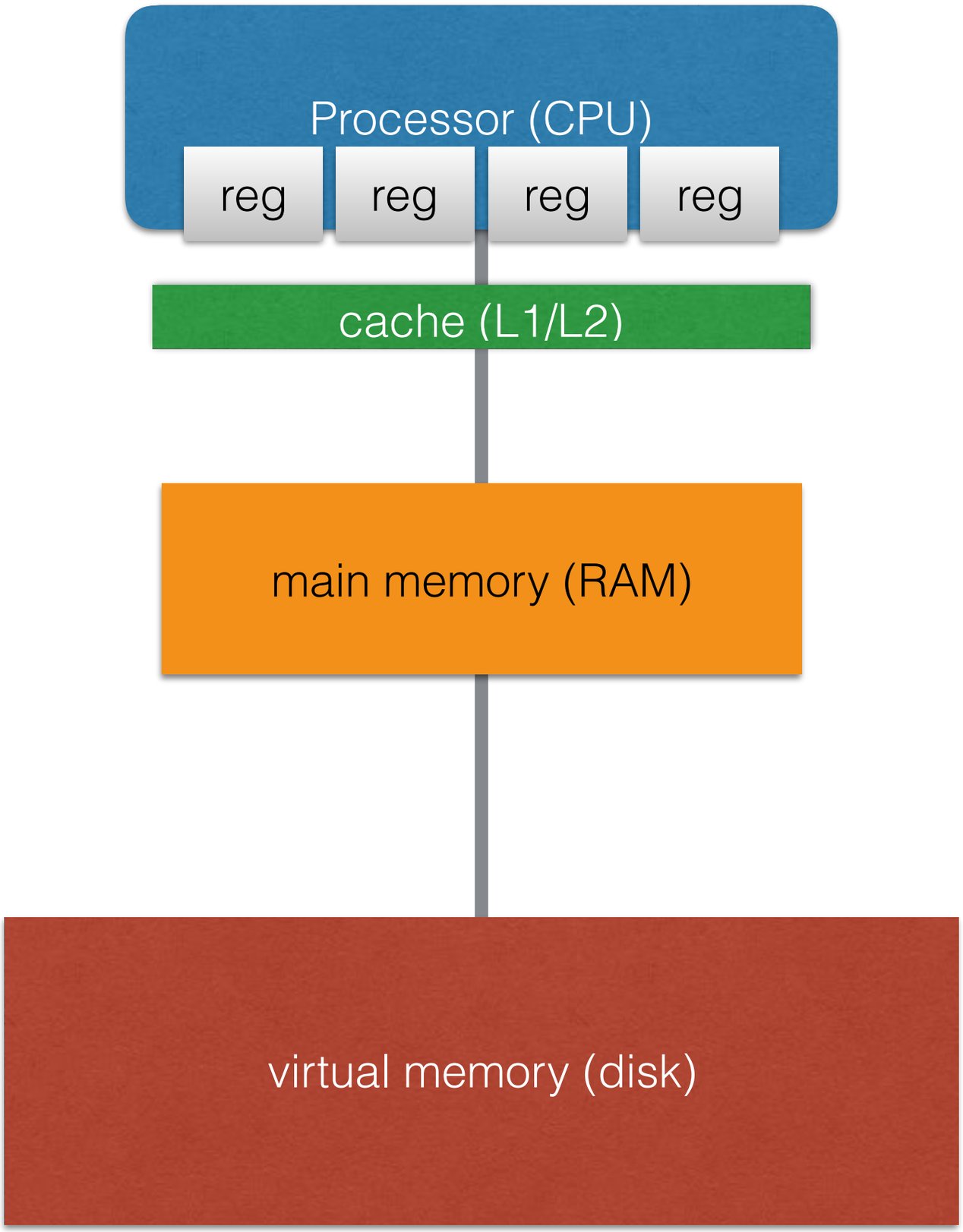
edges is at most …..

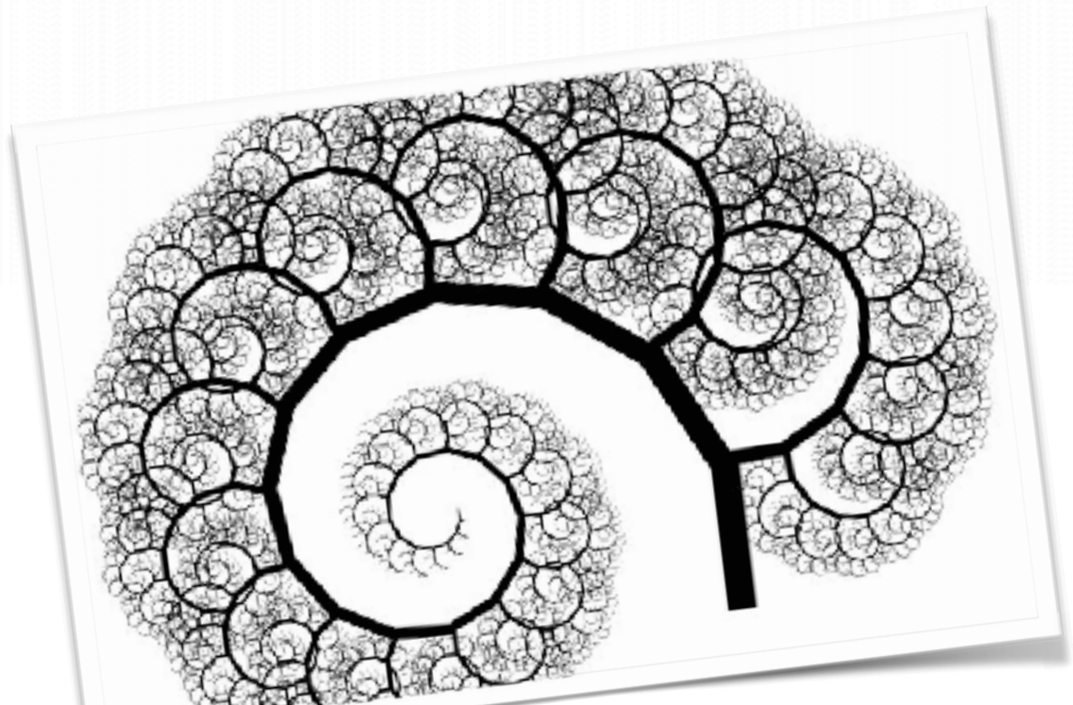


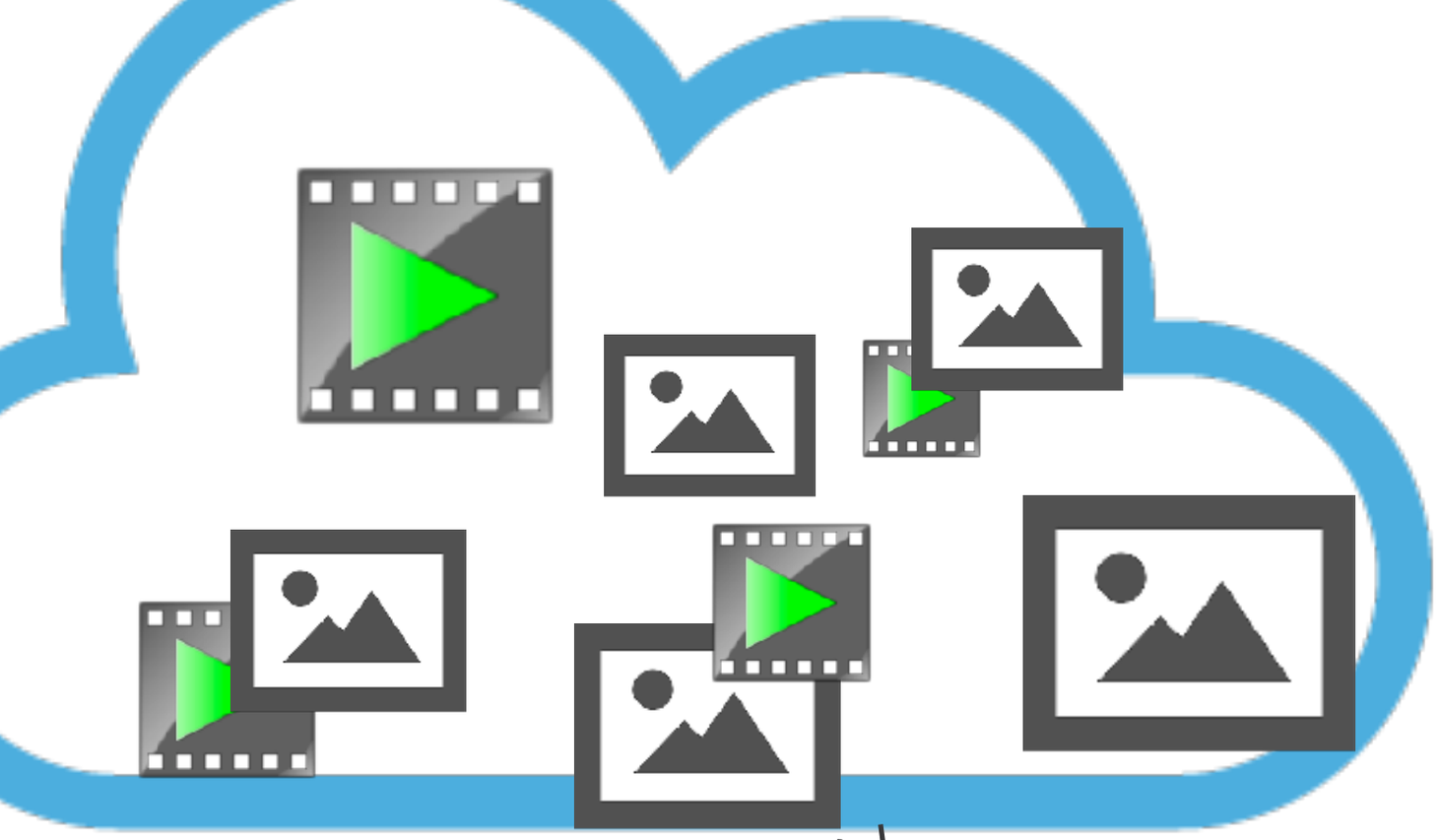
Caching and memoization

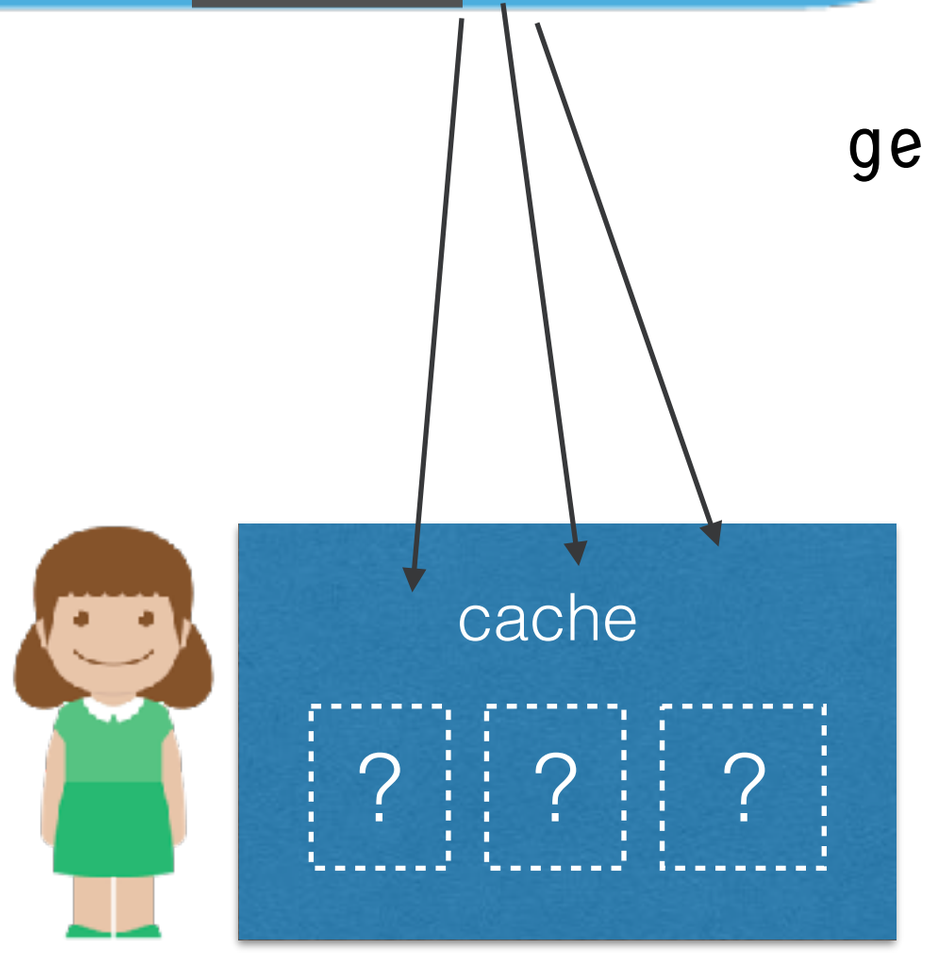
2000 km


2000 km
www.worldmaps.com


Processor (CPU)
reg
reg
reg
reg
cache (L1/L2)
main memory (RAM)
virtual memory (disk)







Template caching logic

```
get(key):
    if key in cache:
        value = cache.get(key)
    else:
        value = origin.get(key)
        cache.put(key, value)
    return value
```

foo(22)

foo(100)

foo(10)

foo(30)

foo(8)

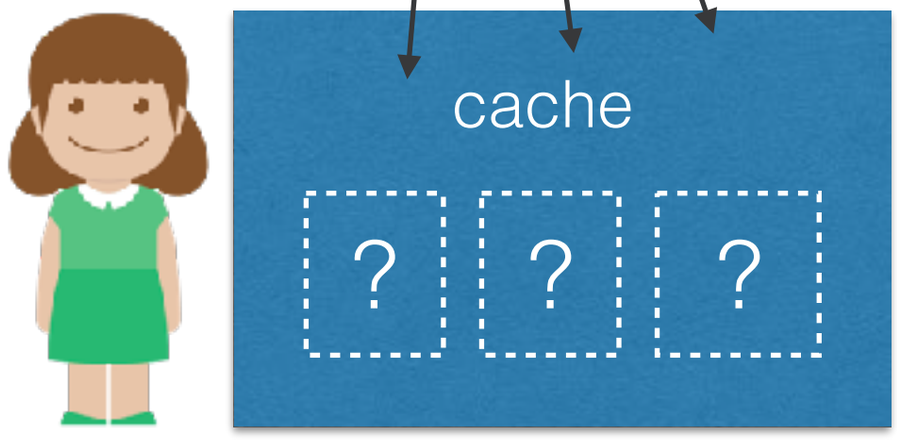
foo(11)

foo(7)

computing
from scratch

Memoization is caching

```
get-foo(input):
    if input in cache:
        output = cache.get(input)
    else:
        output = foo(input)
        cache.put(input, output)
    return output
```



Longest increasing subsequence

Given a sequence of numbers

for example: 2, 4, 3, 5, 1, 7, 6, 9, 8

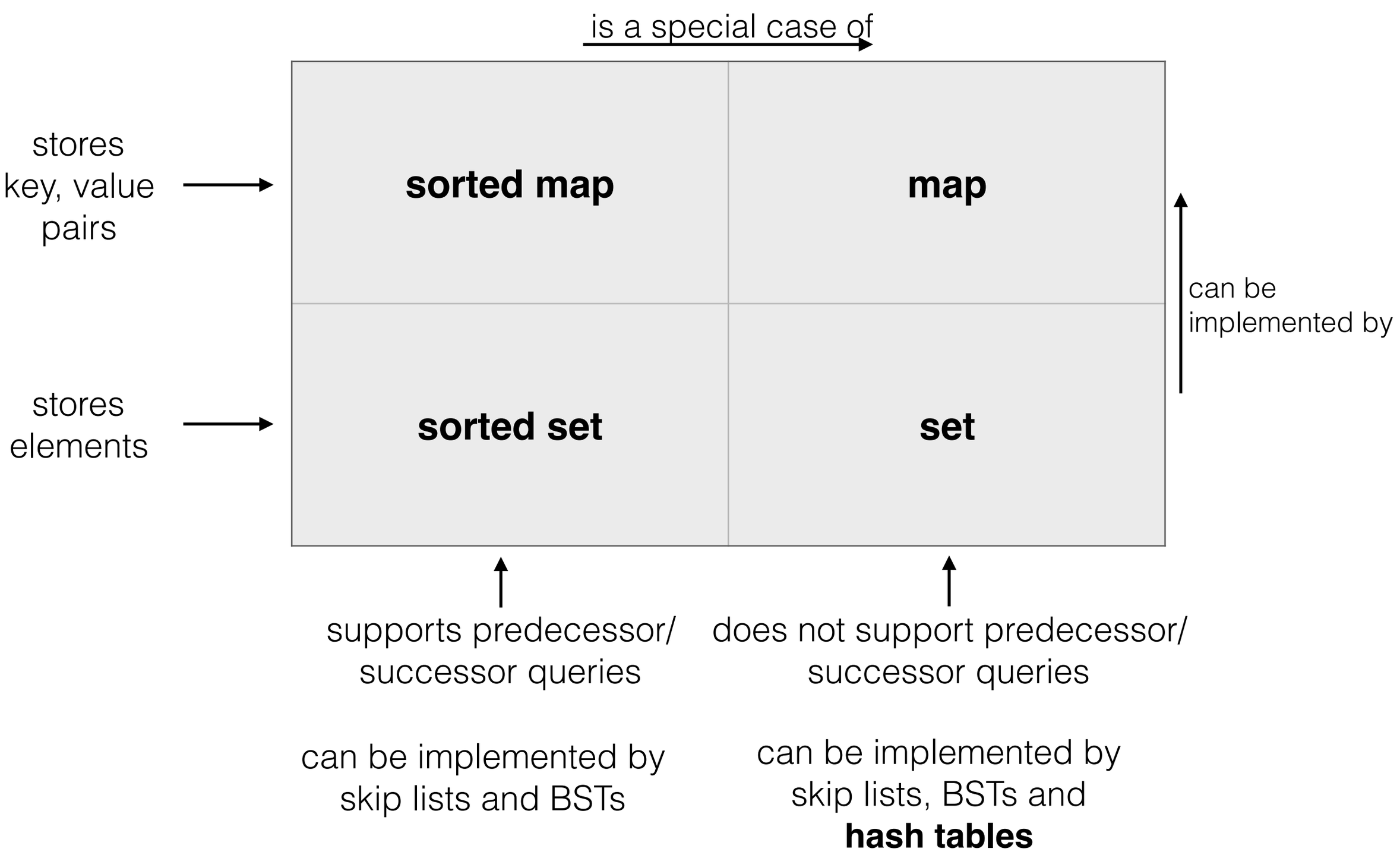
What is the length of the longest increasing subsequence?

strictly

going left to right, possibly skipping some

find an algorithm

Search ADTs


is a special case of
stores key, value pairs
sorted map
map
can be implemented by
stores elements
sorted set
set
supports predecessor/ successor queries
does not support predecessor/ successor queries
can be implemented by skip lists and BSTs
can be implemented by skip lists, BSTs and hash tables

Recap: dictionaries

- dictionaries are very useful
- graphs
  - 3 ways to represent them (including dictionary)
  - which is better depends on graph density and query type
- memoization: speeding up algorithms by caching intermediate values