Data Structures
Graph Fundamentals

CS 225
November 10, 2023

Brad Solomon & G Carl Evans

Department of Computer Science

Plagiarism is not ok

Learning Objectives

Finish discussing MinHash Sketches

Define graph vocabulary

Discuss graph implementation and storage strategies

Cardinality Sketch

Given any dataset and a SUHA hash function, we can **estimate the number of unique items** by tracking the **k-th minimum hash value**.

Hash

0.253	0.839	0.327	0.655	0.491

Applied Cardinalities

Cardinalities

$$|A|$$

$$|B|$$

$$|A \cup B|$$

$$|A \cap B|$$

Set similarities

$$O = \frac{|A \cap B|}{min(|A|, |B|)}$$

$$J = \frac{|A \cap B|}{|A \cup B|}$$

Real-world Meaning

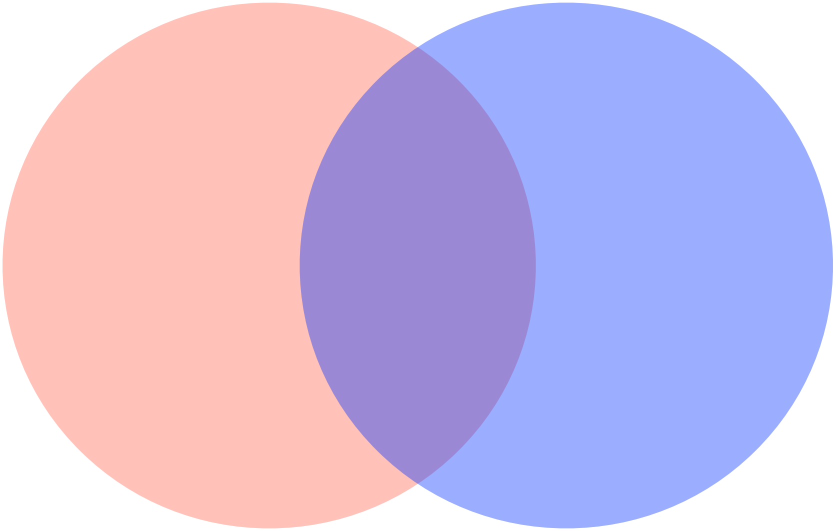
```
AGGCCACAGTGTATTATGACTG
||||||||||| ||||||||||
AGGCCACAGTGAGTTATGACTG

AAAAAAAAAAAGATGT-AAGTA
|||||||||||||||| |||||
AAAAAAAAAAAGATGTAAAGTA

GAGG--TCAGATTCACAGCCAC
||||  ||||||||||||||||
GAGGGGTCAGATTCACAGCCAC
```

Set Similarity Review

To measure **similarity** of A & B, we need both a measure of how similar the sets are but also the total size of both sets.



$$J = \frac{|A \cap B|}{|A \cup B|}$$

J is the ***Jaccard coefficient***

MinHash Sketch

Claim: Under SUHA, set similarity can be estimated by sketch similarity!

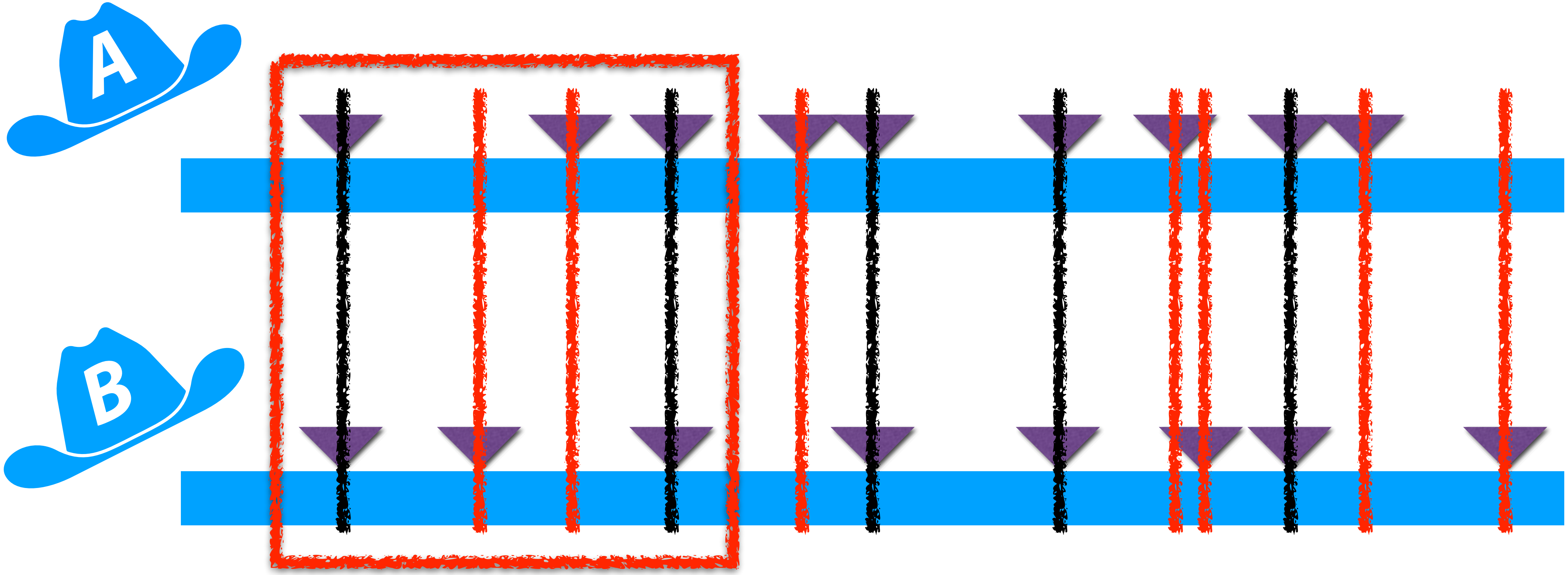


Image inspired by: Ondov B, Starrett G, Sappington A, Kostic A, Koren S, Buck CB, Phillippy AM. **Mash Screen: high-throughput sequence containment estimation for genome discovery**. *Genome Biol* 20, 232 (2019)

MinHash Jaccard Estimation

Let’s assume we have sets A and B sampled uniformly from [0, 100].

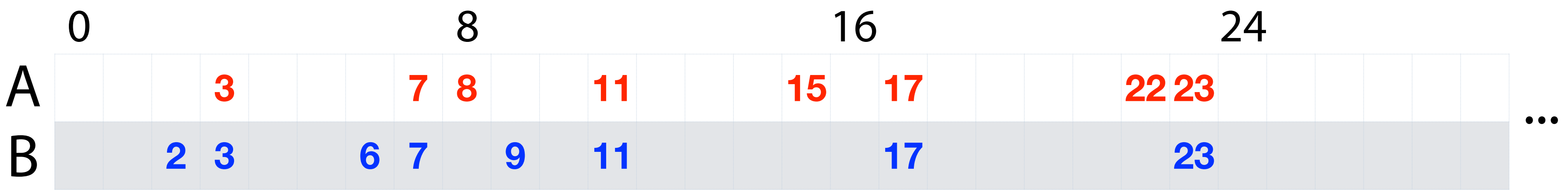
Instead of storing A & B, we store the bottom-8 **MinHash**

Sketch A

3	15
7	17
8	22
11	23

Sketch B

2	9
3	11
6	17
7	23



MinHash Jaccard Estimation

We dont *know* $|A \cup B|$, but we can estimate it!

Sketch A

3	15
7	17
8	22
11	23

$\cup$

Sketch B

2	9
3	11
6	17
7	23

$=$

Sketch of $|A \cup B|$

2	8
3	9
6	11
7	15

	0								8								16								24						
A				3				7	8			11				15		17					22	23							…
B			2	3			6	7		9		11						17						23							

MinHash Cardinality Estimate

We can estimate the cardinality of the actual sets using our sketches.

Sketch of
$|A \cup B|$

2	8
3	9
6	11
7	15

Our sets sampled from [0, 100].

$$\frac{15}{100} = \frac{8}{N+1}$$

MinHash Indirect Jaccard Estimation

$$\frac{|A| \cap |B|}{|A| \cup |B|} = \frac{|A| + |B| - |A \cup B|}{|A \cup B|}$$

$k = 8$ MinHash sketches

Our sets sampled from [0, 100]

Sketch A

3	15
7	17
8	22
11	23

Sketch B

2	9
3	11
6	17
7	23

Sketch of $|A \cup B|$

2	8
3	9
6	11
7	15

$$= \frac{(800/23-1) + (800/23-1) - (800/15-1)}{800/15-1}$$

$$= \frac{34.782 + 34.782 - 53.333 - 1}{53.333 - 1} \approx 0.29$$

MinHash Direct Jaccard Estimate

We can also estimate cardinality directly using our sketches!

Sketch A

3	15
7	17
8	22
11	23

Sketch B

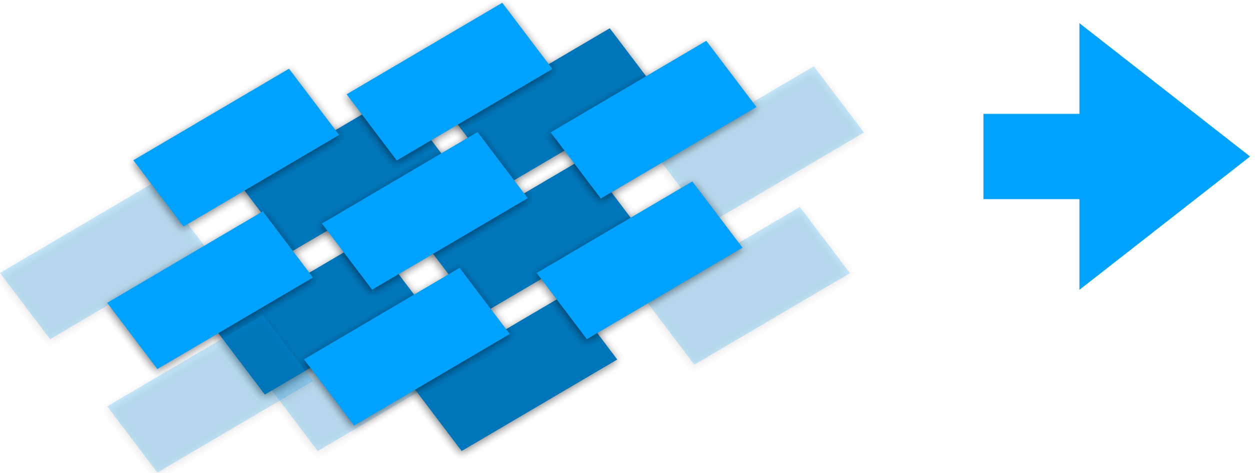
2	9
3	11
6	17
7	23

Intersection

Union

MinHash Sketch

We can convert any hashable dataset into a **MinHash sketch**



{1 , 3 , 5}

We lose our original dataset, but we can still estimate two things:

1.

2.

Alternative MinHash Sketch Approaches

The **easiest** version of MinHash uses k hashes. How might this work?

1) Sequence decomposed into **kmers**

2) Multiple hash functions (**Γ**) map kmers to values.

3) The smallest values for each hash function is chosen

4) The Jaccard similarity can be estimated by the overlap in the **Min**imum **Hash**es (**MinHash)**

S_1 : CATGGACCGACCAG

CAT GAC GAC
ATG ACC ACC
TGG CCG CCA
GGA CGA CAG

GCAGTACCGATCGT : S_2

GTA CGA CGT
AGT CCG TCG
CAG ACC ATC
GCA TAC GAT

Γ_1	Γ_2	Γ_3	Γ_4	
19	14	57	36	CAT
14	57	36	19	ATG
58	37	16	15	TGG
40	23	2	61	GGA
33	28	11	54	GAC
5	48	47	26	ACC
22	1	60	43	CCG
24	7	50	45	CGA
33	28	11	54	GAC
5	48	47	26	ACC
20	3	62	41	CCA
18	13	56	39	CAG

	Γ_1	Γ_2	Γ_3	Γ_4
GCA	36	19	14	57
CAG	18	13	56	39
AGT	11	54	33	28
GTA	44	27	6	49
TAC	49	44	27	6
ACC	5	48	47	26
CCG	22	1	60	43
CGA	24	7	50	45
GAT	35	30	9	52
ATC	13	56	39	18
TCG	54	33	28	11
CGT	27	6	49	44

[5, 1, 2, 15]
Sketch (S_1)

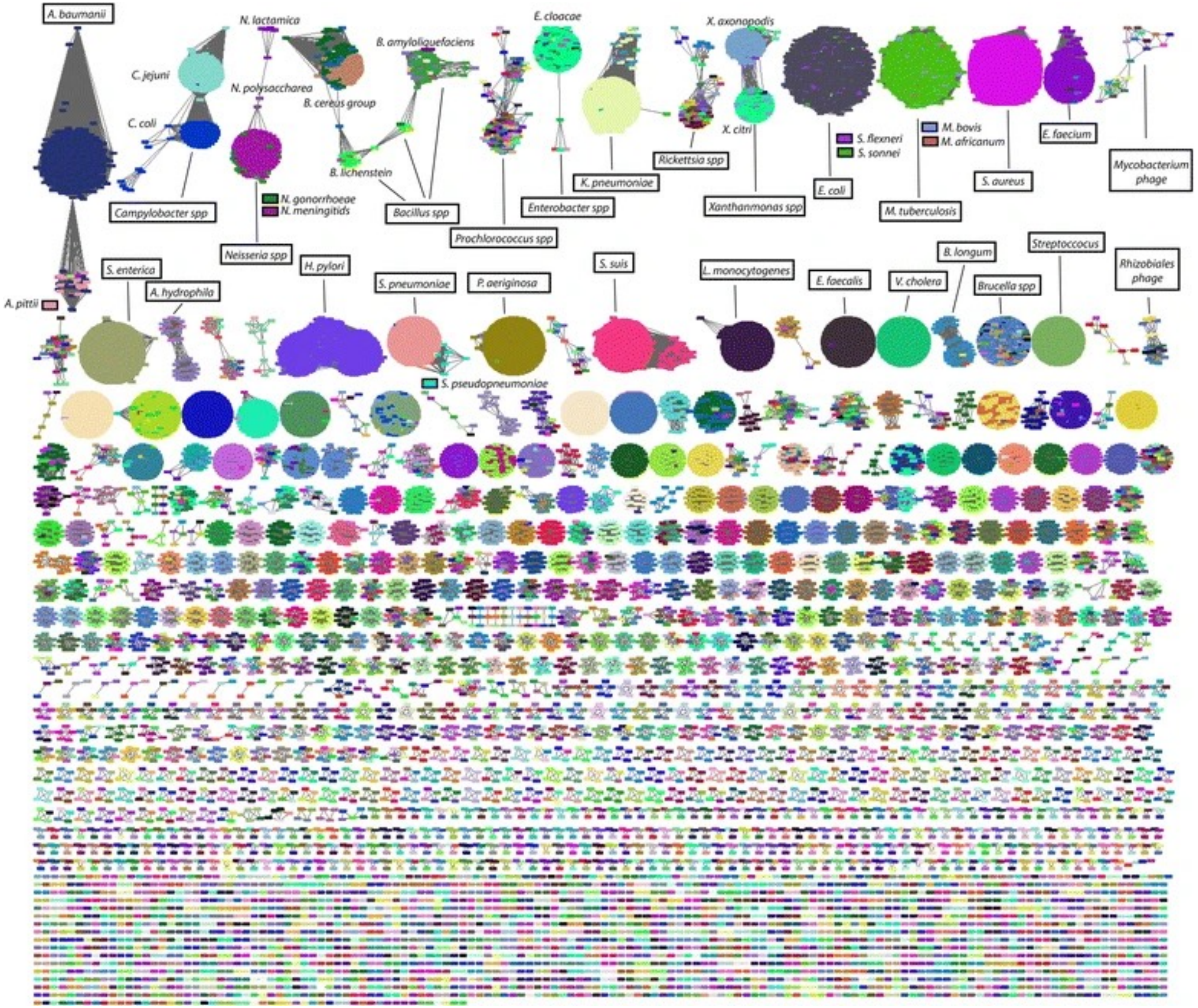
[5, 1, 6, 6]
Sketch (S_2)

$J(S_1, S_2) \approx 2/4 = 0.5$

```
S1 : CATGGACCGACCAG
       |  |||||  |
S2 : GCAGTACCGATCGT
```

Assembling large genomes with single-molecule sequencing and locality-sensitive hashing
Berlin et al (2015) *Nature Biotechnology*

MinHash in practice



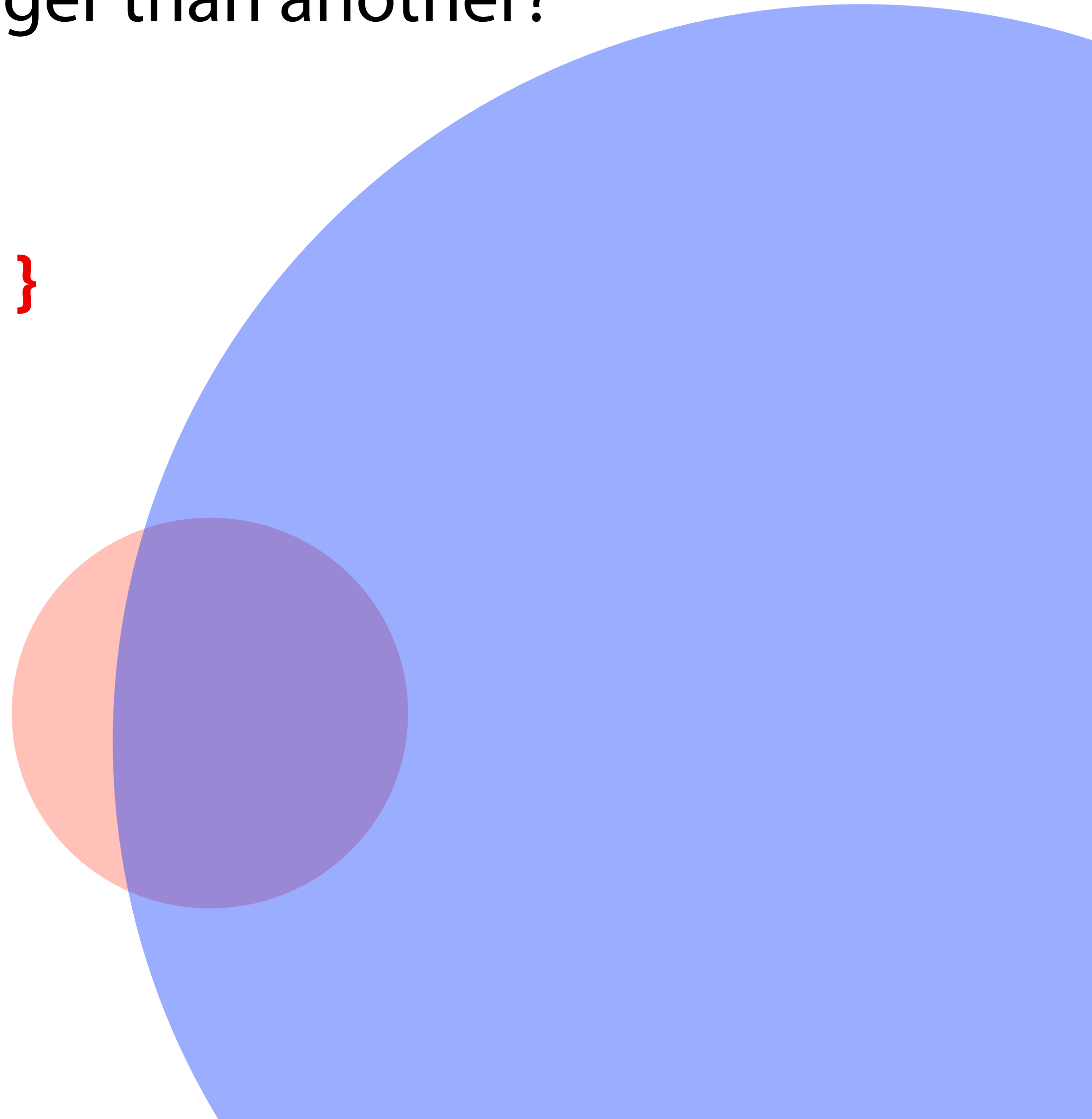
Mash: fast genome and metagenome distance estimation using MinHash
Ondov et al (2016) *Genome Biology*

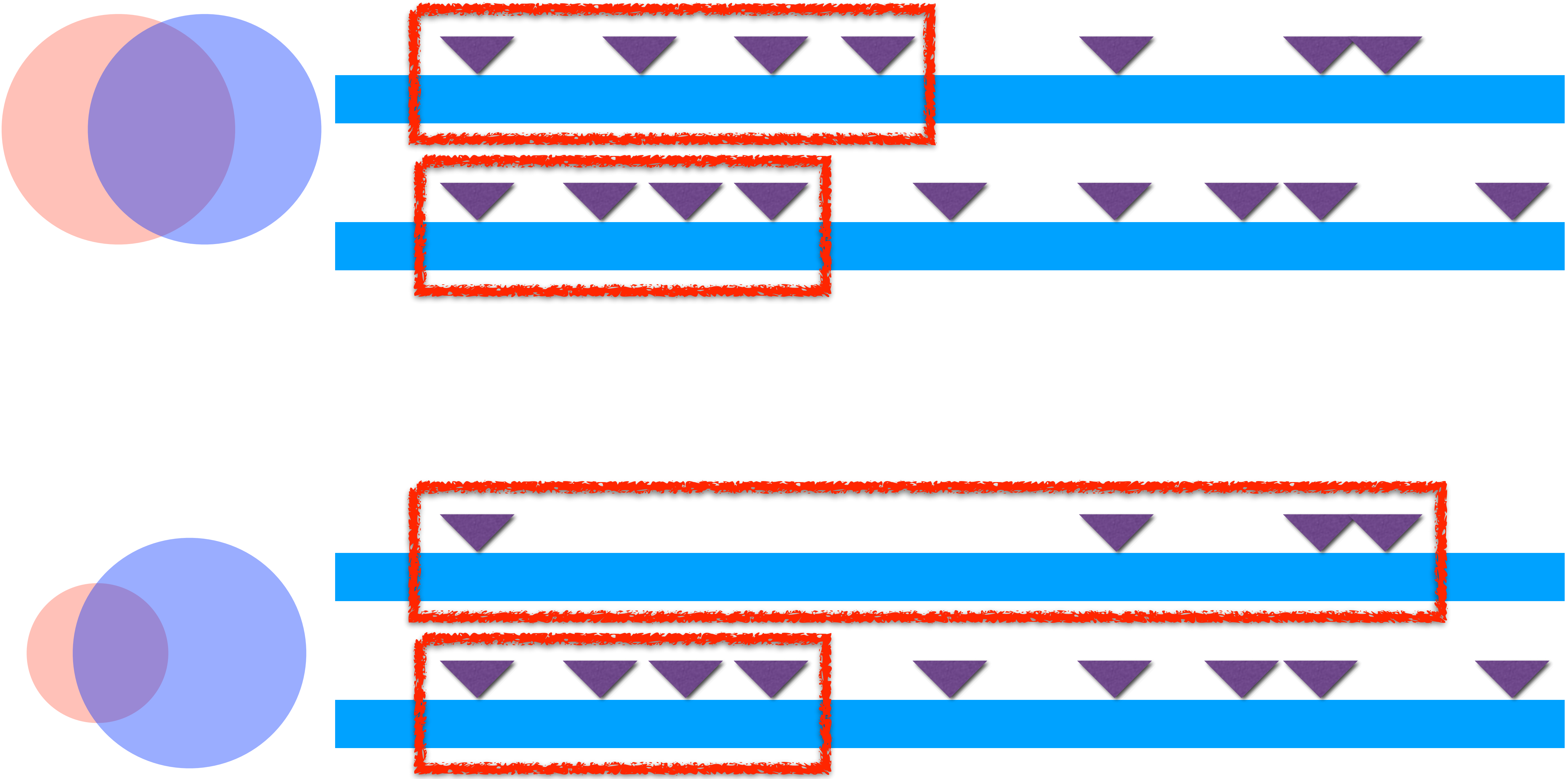
Alternative MinHash Sketch Approaches

What if I have a dataset which is **much** larger than another?

$S_1 = \{ 1, 3, 40, 59, 82, 101 \}$

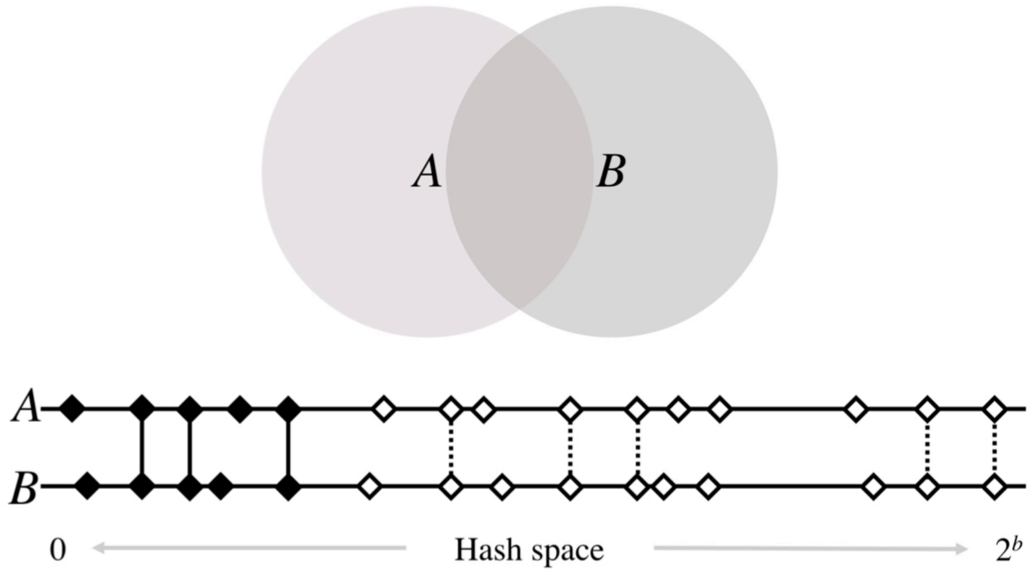
$S_2 = \{ 1, 2, 3, 4, 5, 6, 7, \ldots 59, 82, 101, \ldots \}$

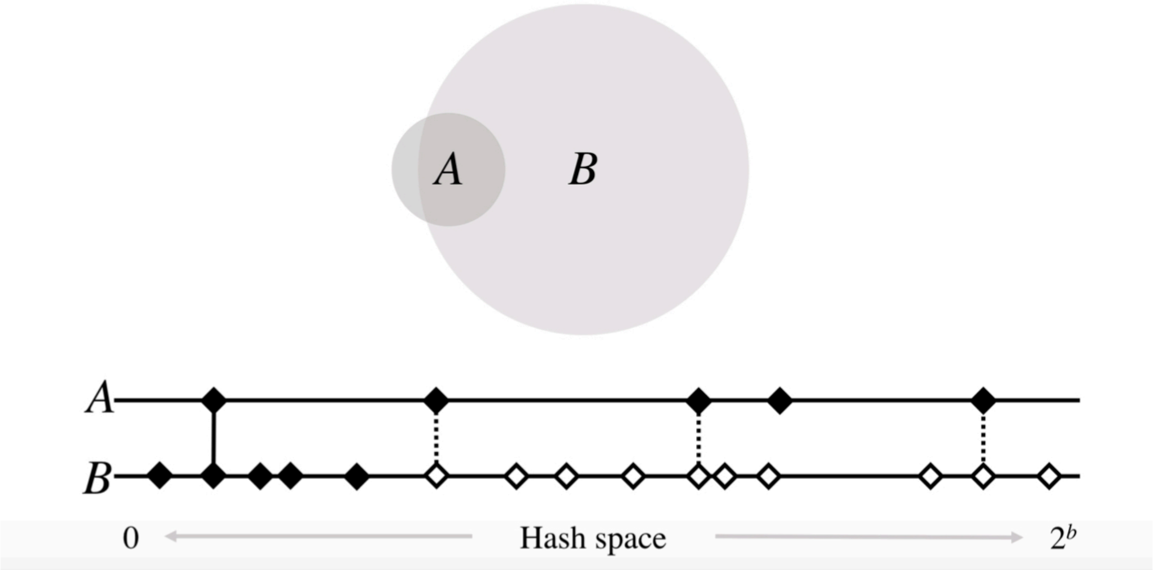




Alternative MinHash sketches

Bottom-k minhash has low accuracy if the cardinality of sets are skewed

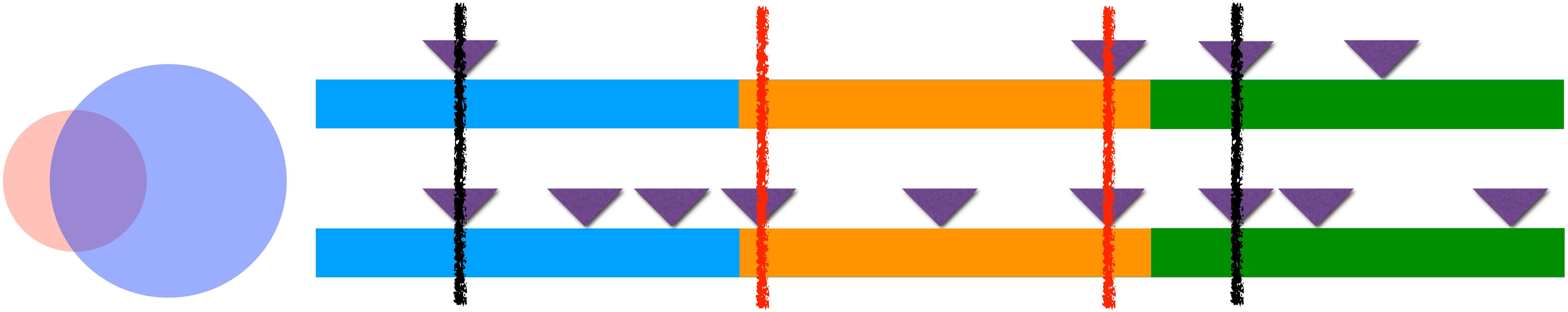




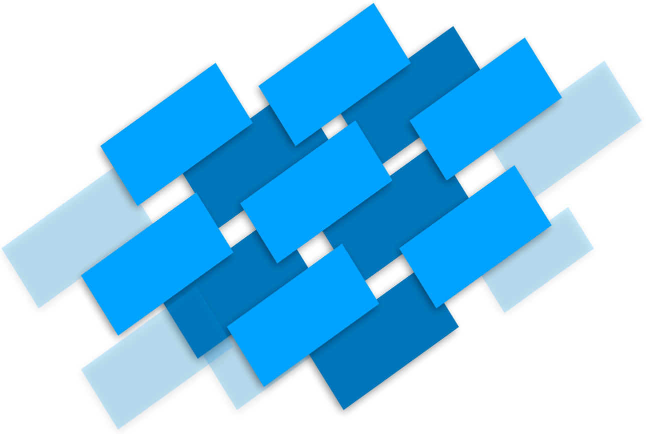
Ondov, Brian D., Gabriel J. Starrett, Anna Sappington, Aleksandra Kostic, Sergey Koren, Christopher B. Buck, and Adam M. Phillippy. **Mash Screen: High-throughput sequence containment estimation for genome discovery**. *Genome biology* 20.1 (2019): 1-13.

Alternative MinHash Sketch Approaches

If there is a large cardinality difference, **use k-partitions!**



K-Partition Minhash



Hash

1010110101
0001111010
1101101011
1011010110
0101100000
0010001101

Partition

00
01111010
10001101

01
01100000

10
10110101
11010110

11
01101011

K-Partition Minhash

Hint: What bitwise operator(s) will allow me to do this?

`1010110101`	Partition →	`10`
`0001111010`		`00`

What information do I need to do this in general?

MP_Sketching: A MinHash experiment

Using legitimate hashes, write MinHash sketch three ways:

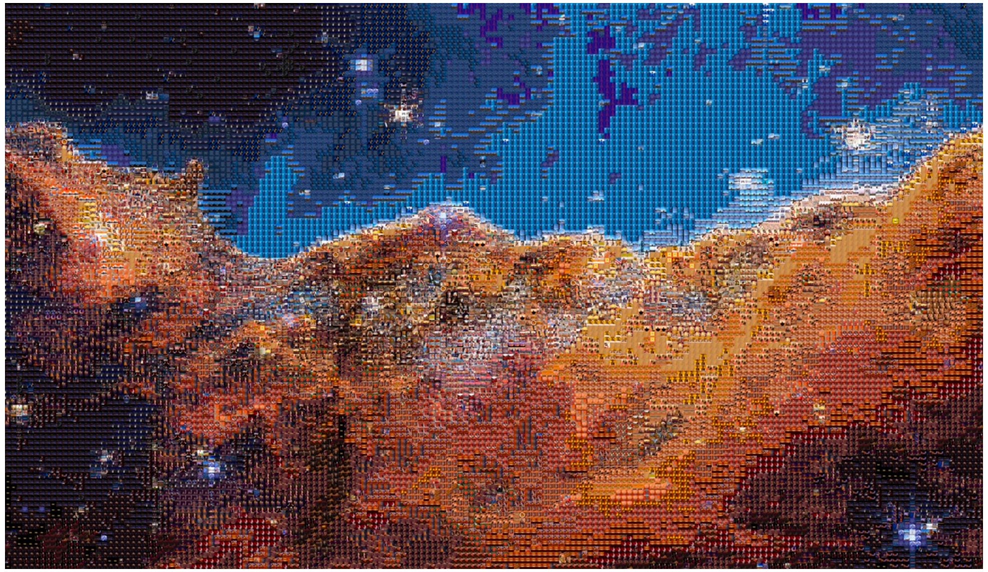
```
std::vector<uint64_t> khash_minhash(std::vector<int> inList, std::vector<hashFunction> hv);
```

```
std::vector<uint64_t> kminhash(std::vector<int> inList, unsigned k, hashFunction h);
```

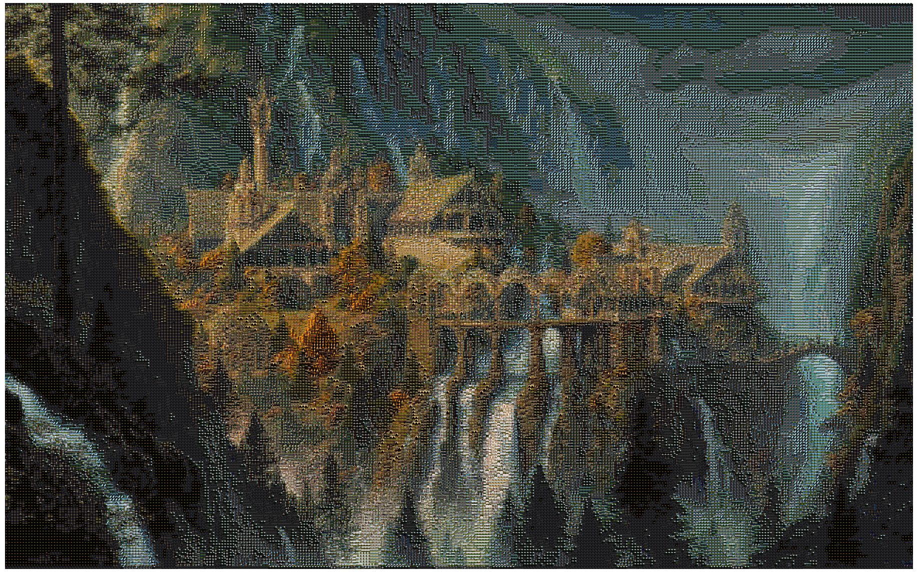
```
std::vector<uint64_t> kpartition_minhash(std::vector<int> inList, int part_bits, hashFunction h);
```

MP_Sketching: A MinHash experiment

Use MinHash sketches to estimate PNG similarity



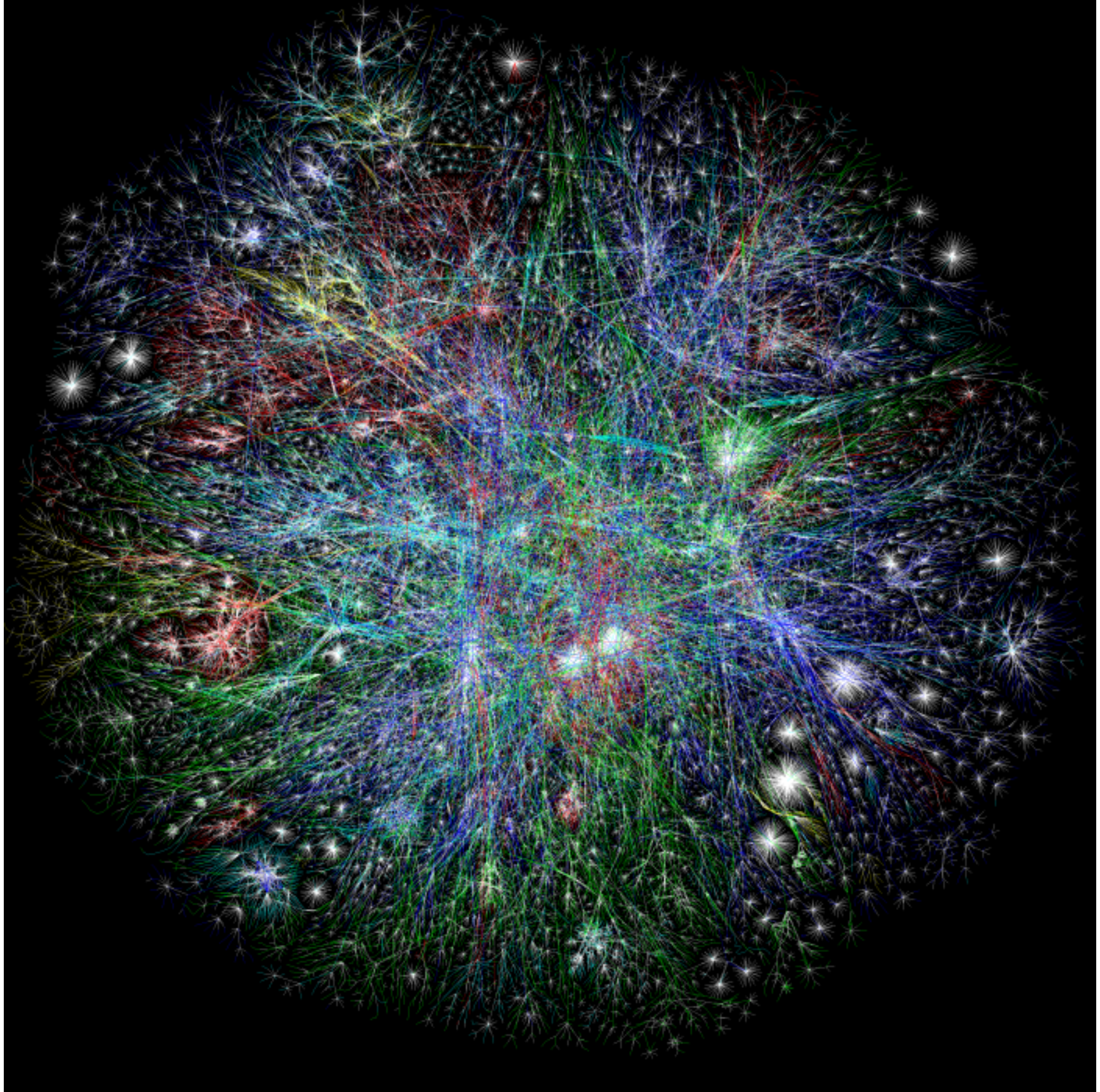
Mosaics (Discord: Bose)



Mosaics (Discord: LightningStorm)

MP_Sketching: A MinHash experiment

Build a weighted graph of every possible pairwise comparison!

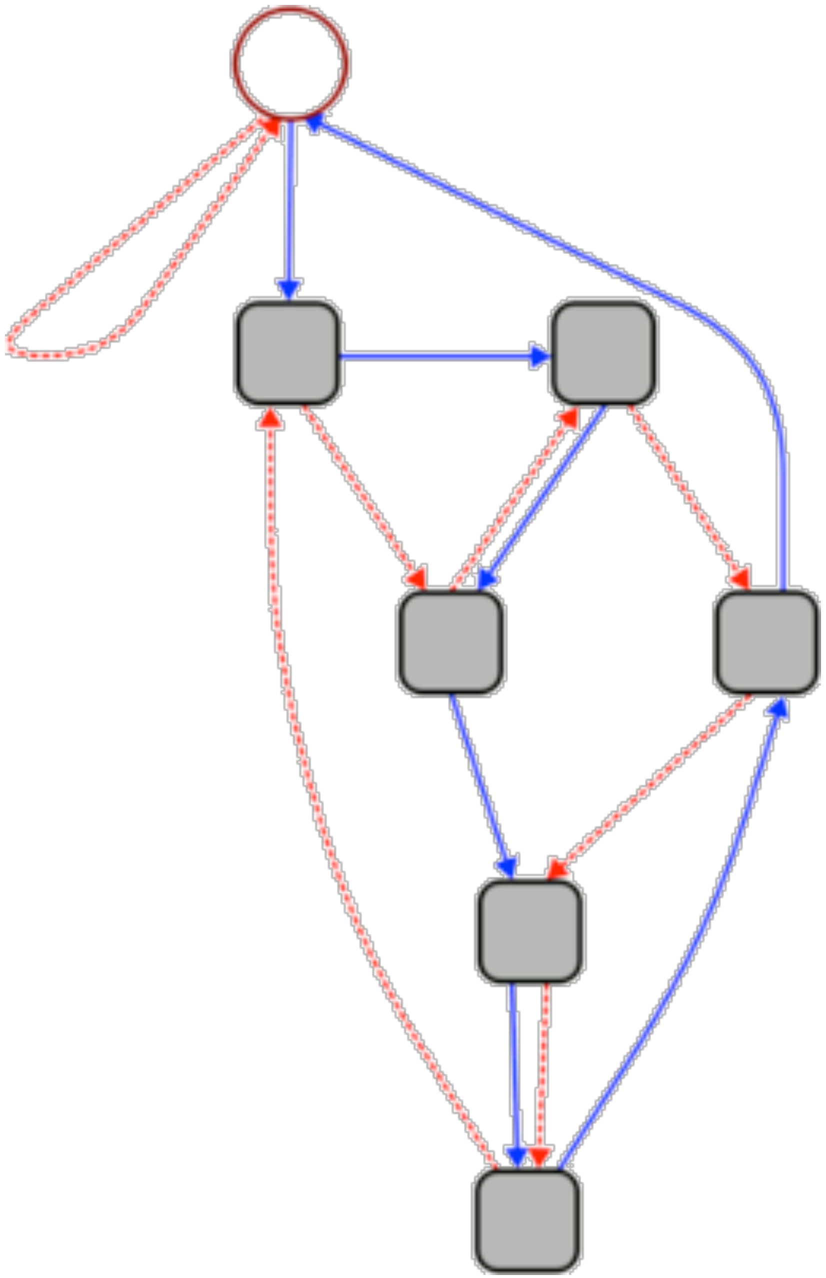


Nodes: Routers and servers

Edges: Connections

The Internet 2003

The OPTE Project (2003)



This graph can be used to quickly calculate whether a given number is divisible by 7.

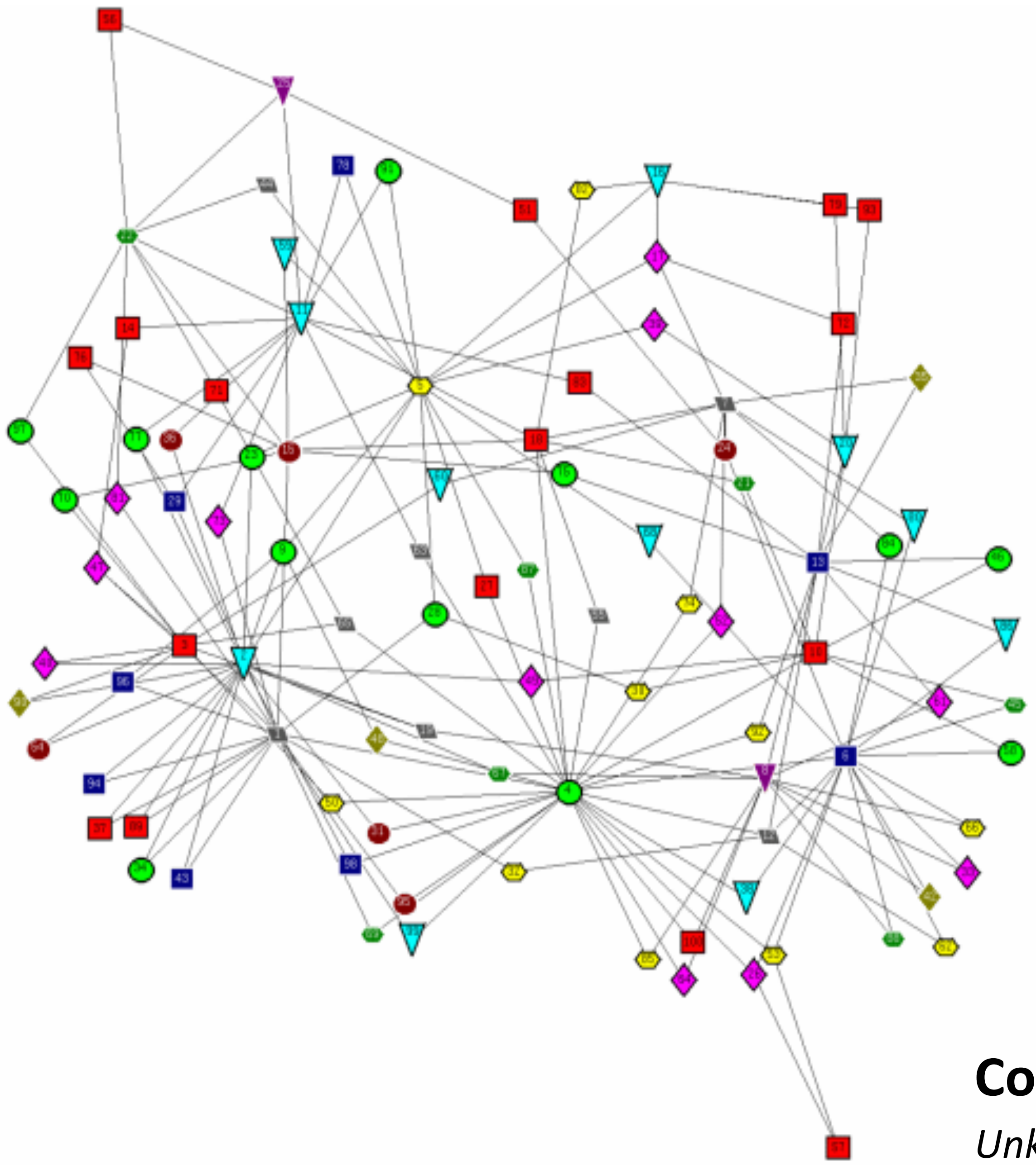
1. Start at the circle node at the top.
2. For each digit **d** in the given number, follow **d** blue (solid) edges in succession. As you move from one digit to the next, follow **1** red (dashed) edge.
3. If you end up back at the circle node, your number is divisible by 7.

3703

“Rule of 7”

Unknown Source

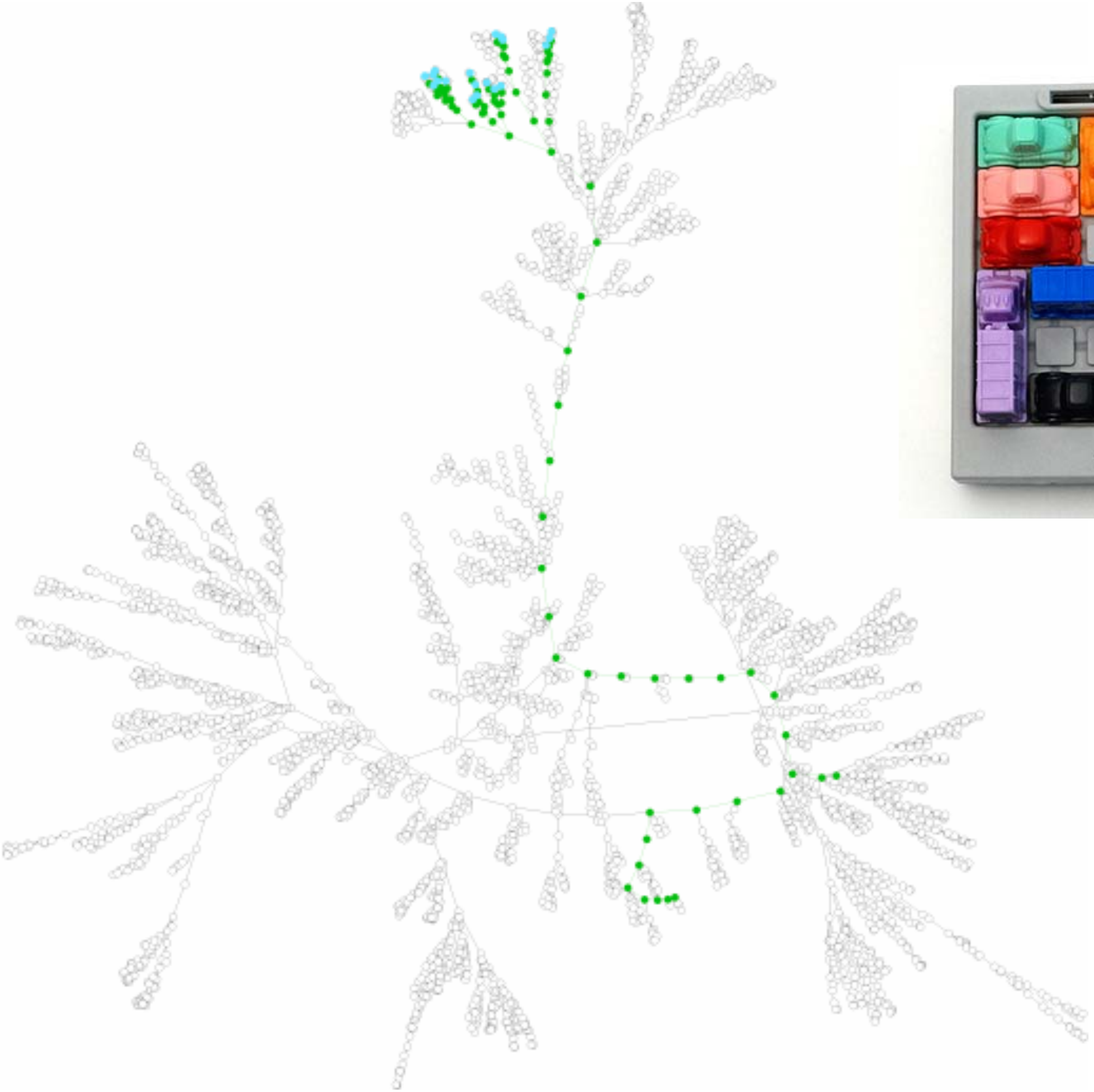
Presented by Cinda Heeren, 2016



Conflict-Free Final Exam Scheduling Graph

Unknown Source

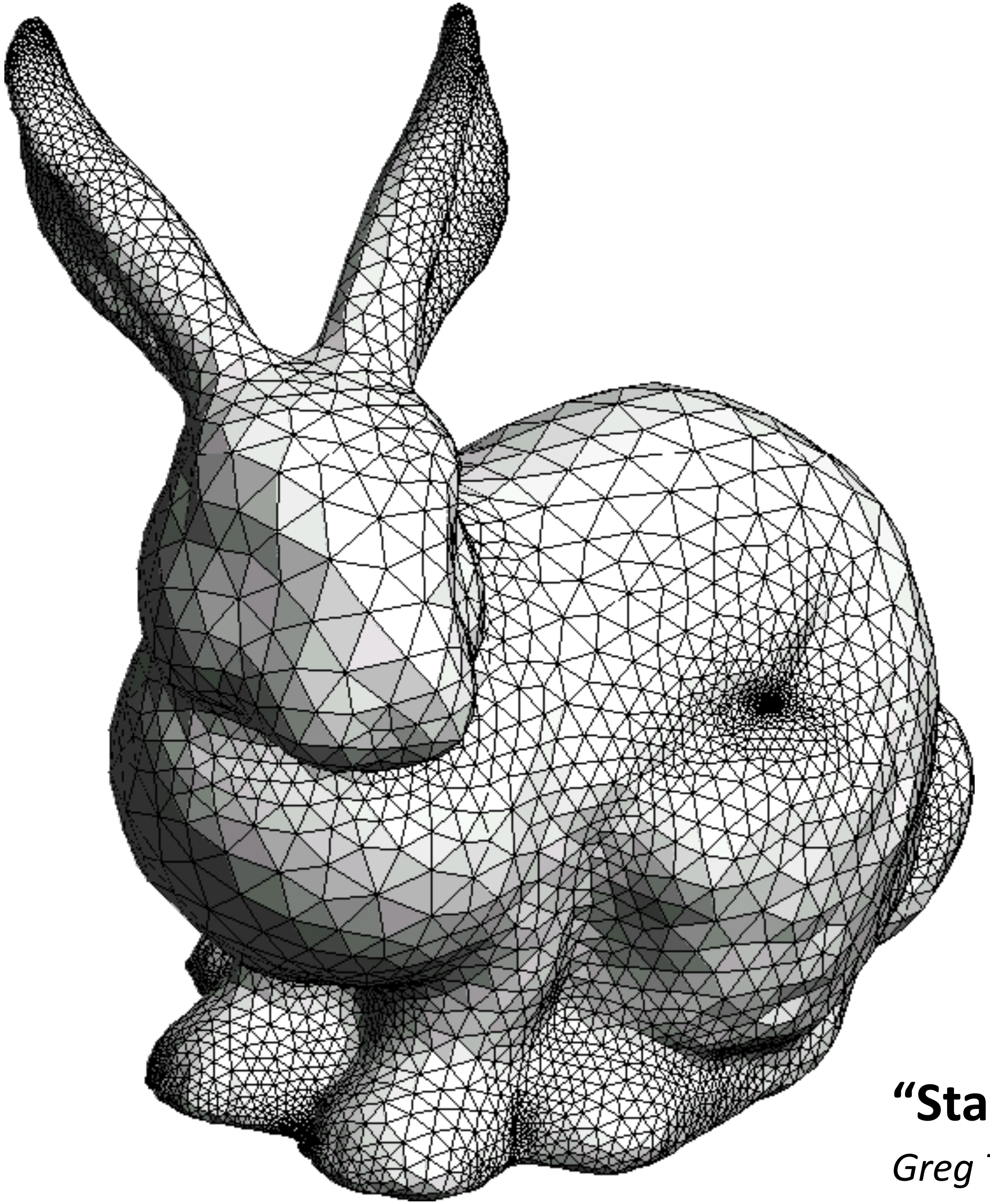
Presented by Cinda Heeren, 2016



“Rush Hour” Solution

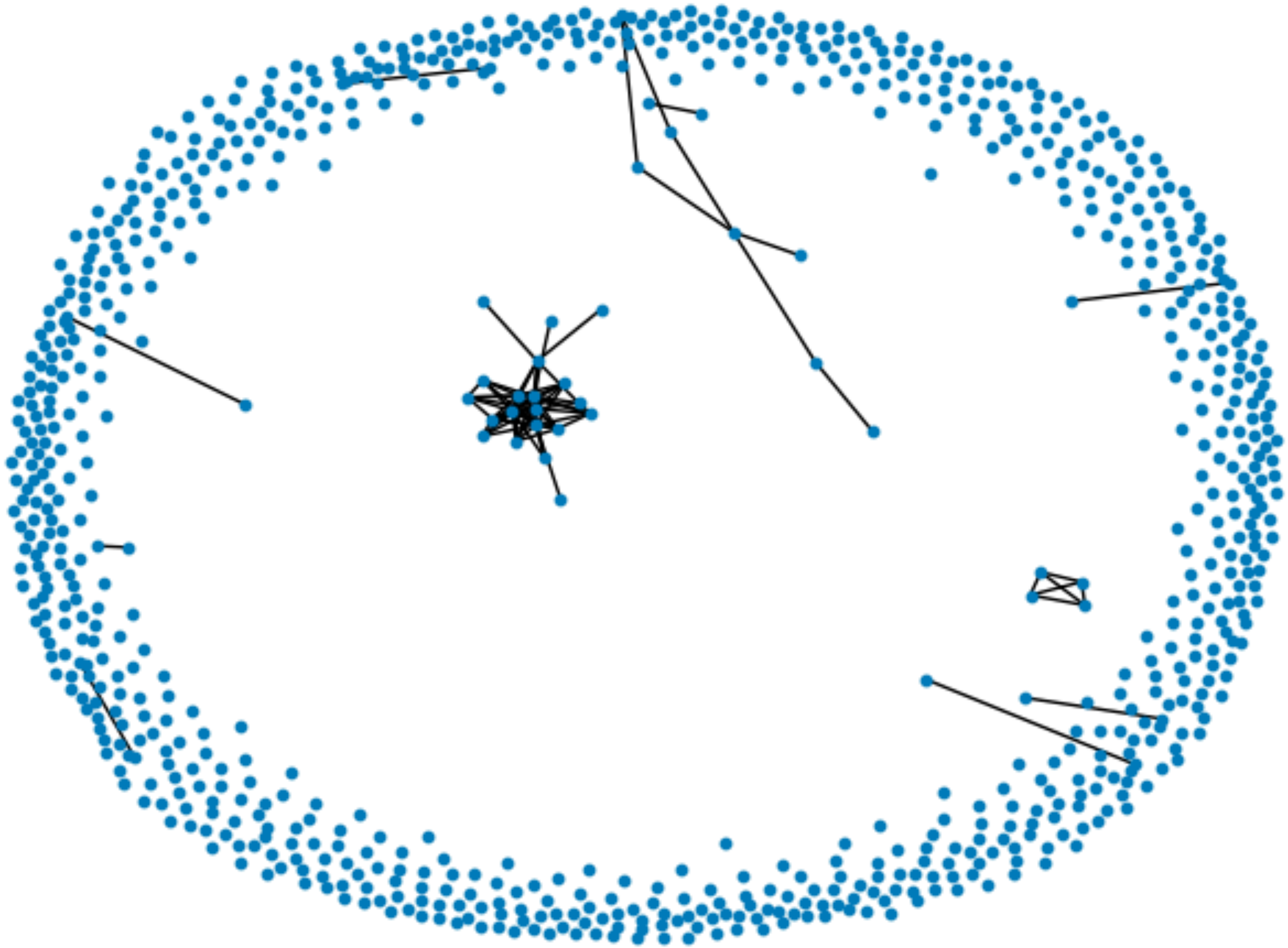
Unknown Source

Presented by Cinda Heeren, 2016

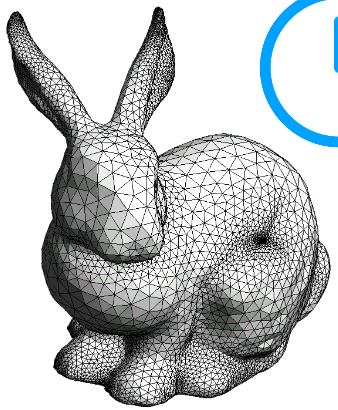


“Stanford Bunny”

Greg Turk and Mark Levoy (1994)

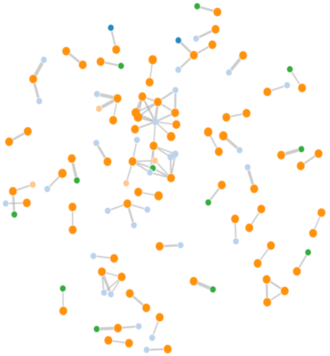


Graphs



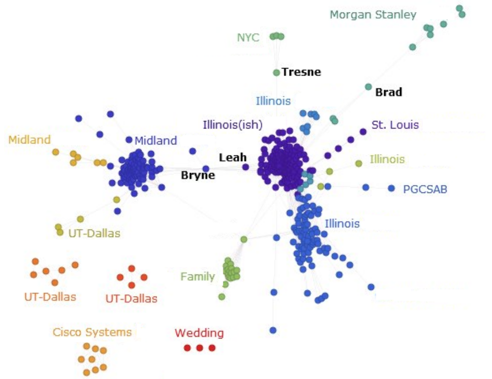
To study all of these structures:

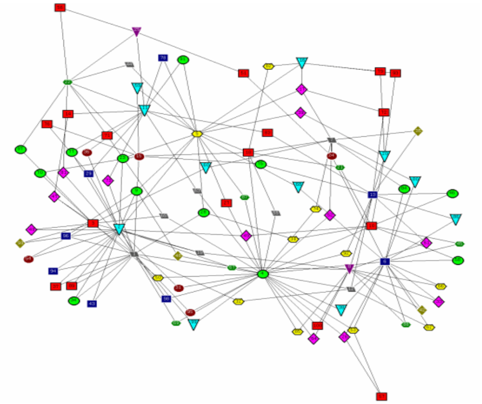
1. A common vocabulary
2. Graph implementations
3. Graph traversals
4. Graph algorithms

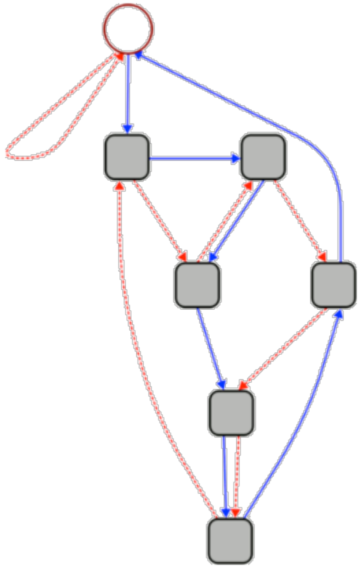


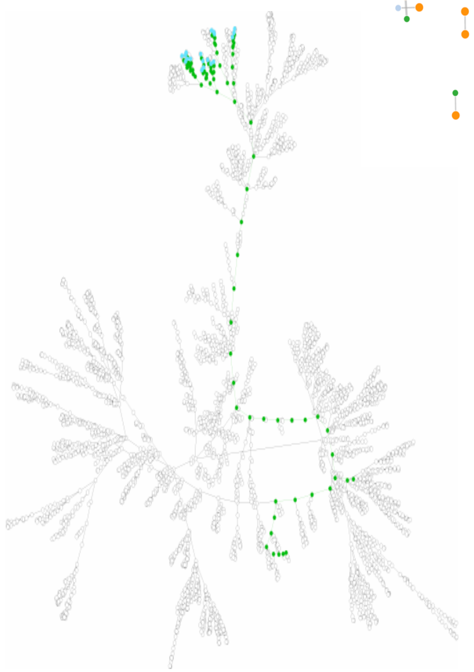

HAMLET


TROILUS AND CRESSIDA





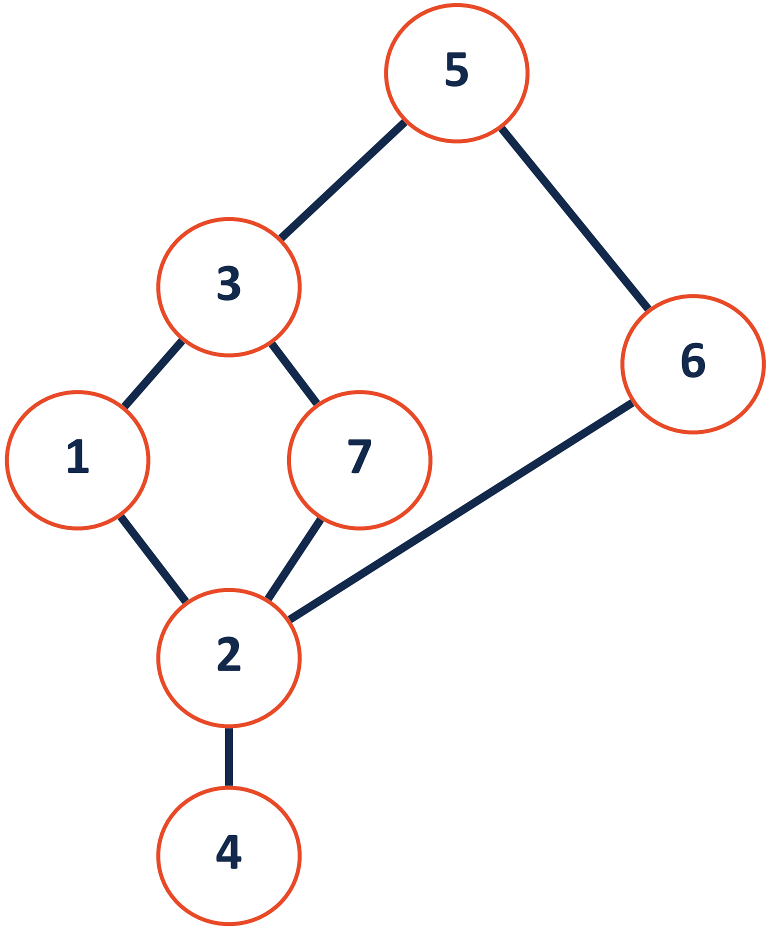




Graph Vocabulary

$G = (V, E)$

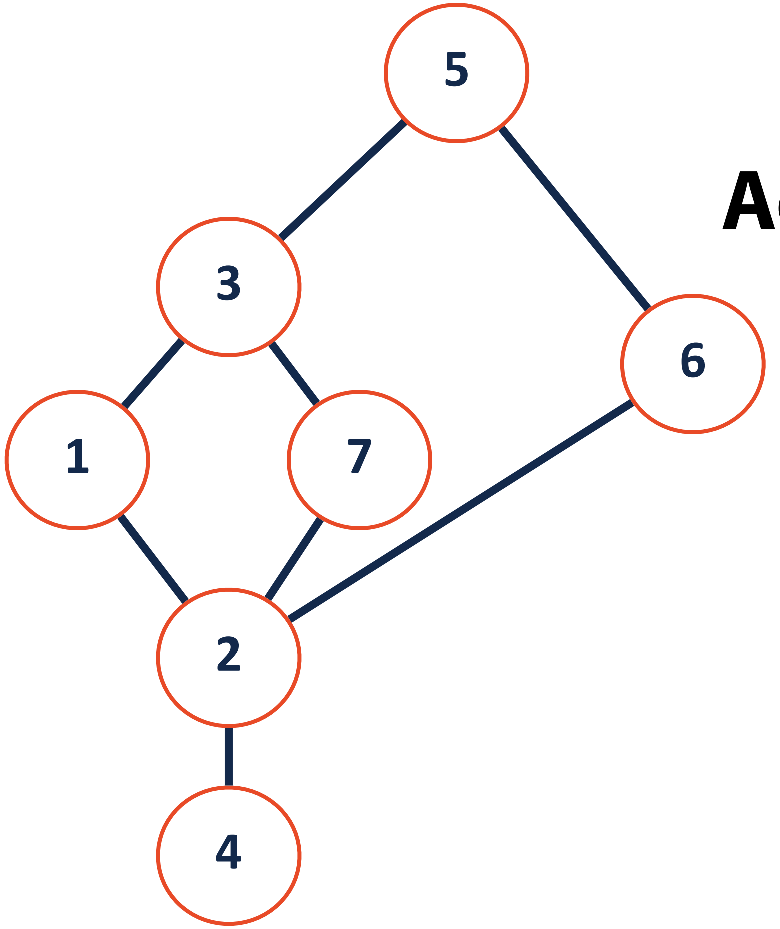
A **graph** is a data structure containing a set of vertices and a set of edges



Vertex:

Edges:

Graph Vocabulary



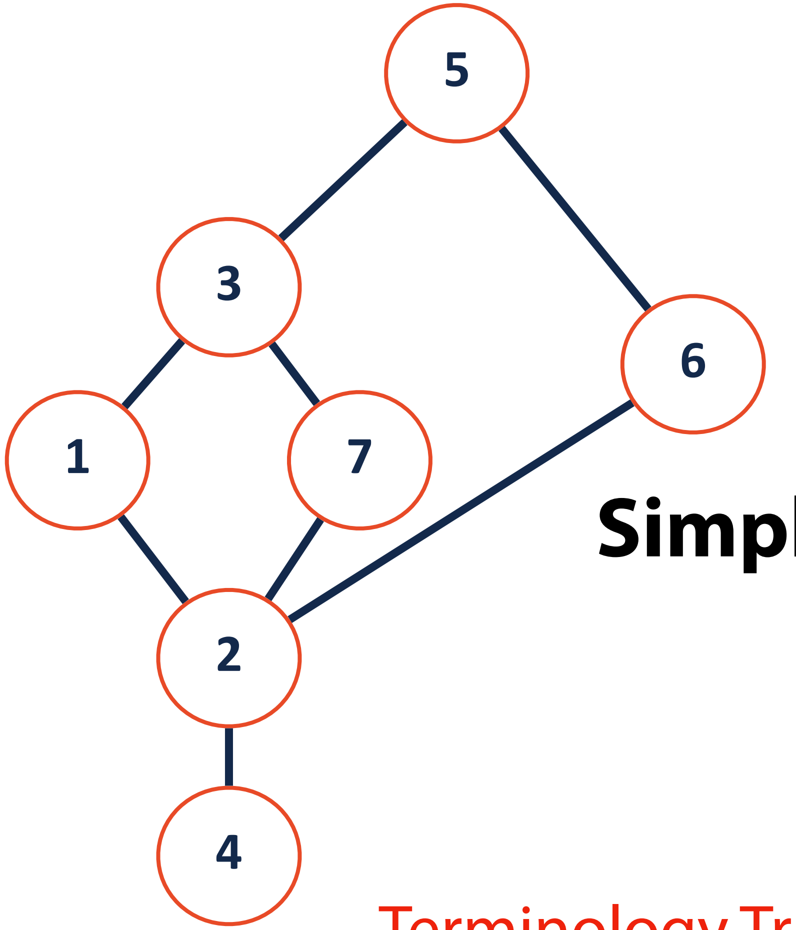
Degree: # of edges touching a vertex

Adjacency: Two vertices are adjacent if they are connected by an edge

Path: A sequence of vertices (or edges) between two nodes

Graph Vocabulary

A graph has **no root** and **may contain cycles**

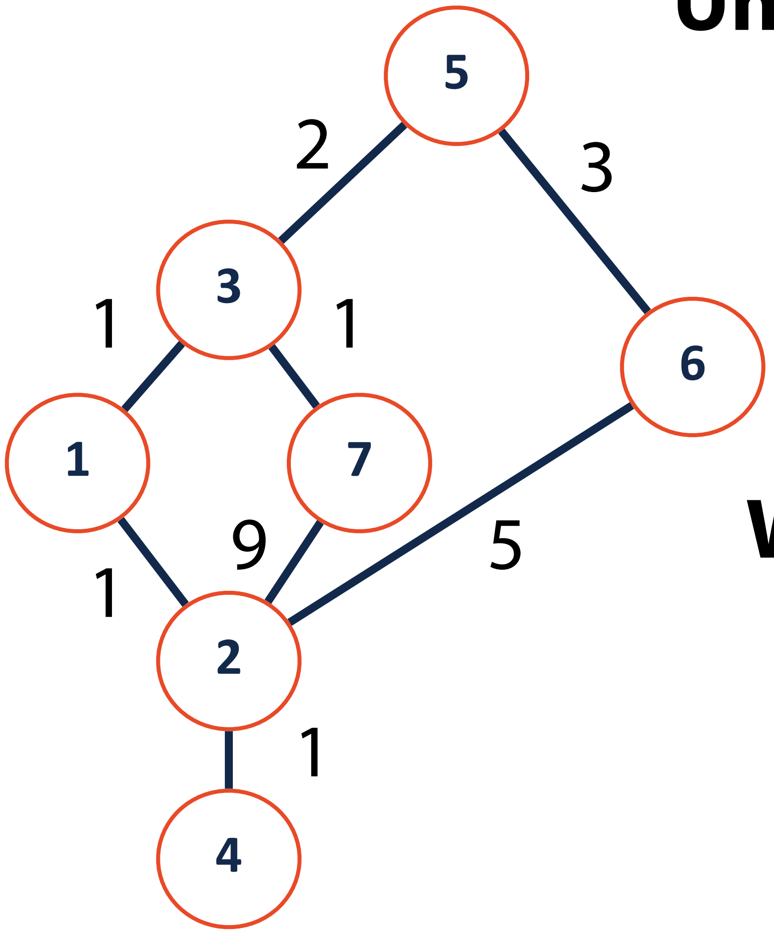


Cycle: A path from a node to itself

Simple Graph: No self-loops or multi-edges

Terminology Trivia: Every tree is a graph but not every graph is a tree

Graph Vocabulary



Directed: Edges are one way connections

Undirected: Traversable in either direction

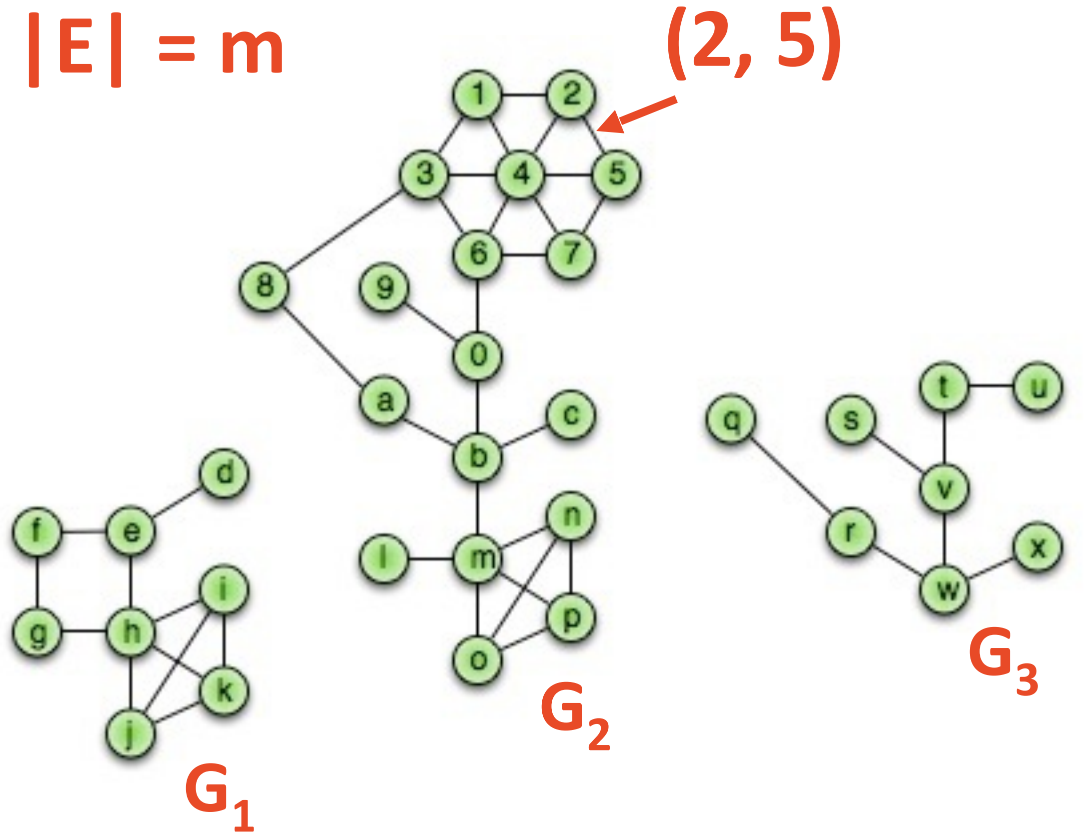
Weighted: A value associated with an edge

Graph Vocabulary

$G = (V, E)$

$|V| = n$

$|E| = m$



Subgraph(G):

$G' = (V', E')$:

$V' \in V$, $E' \in E$, and

$(u, v) \in E' \rightarrow u \in V'$, $v \in V'$

Complete subgraph(G)

Connected subgraph(G)

Connected component(G)

Acyclic subgraph(G)

Spanning tree(G)

Running times are often reported by **n**, the number of vertices, but often depend on **m**, the number of edges.

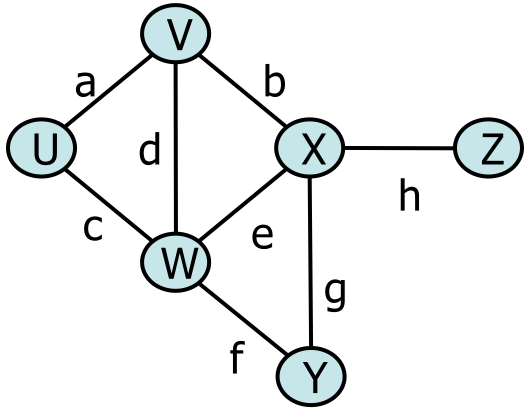
How many edges? **Minimum edges:**

Not Connected:

Connected*:

Maximum edges:

Simple:

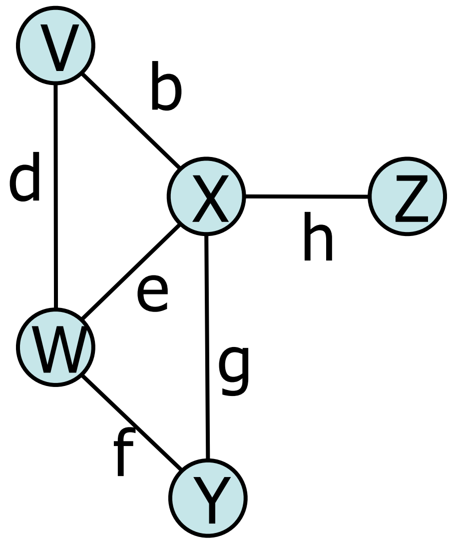


$$\sum_{v \in V} deg(v) =$$

Graph ADT

Data:

- Vertices
- Edges
- Some data structure maintaining the structure between vertices and edges.



Functions:

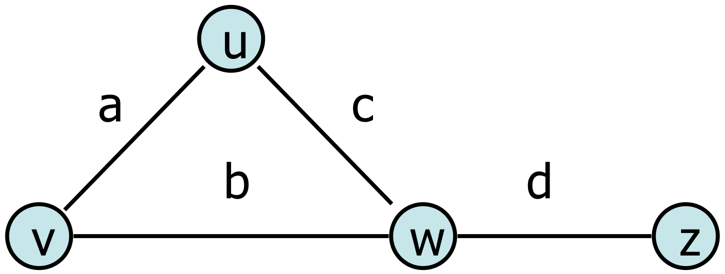
- insertVertex(K key);
- insertEdge(Vertex v1, Vertex v2, K key);

- removeVertex(Vertex v);
- removeEdge(Vertex v1, Vertex v2);

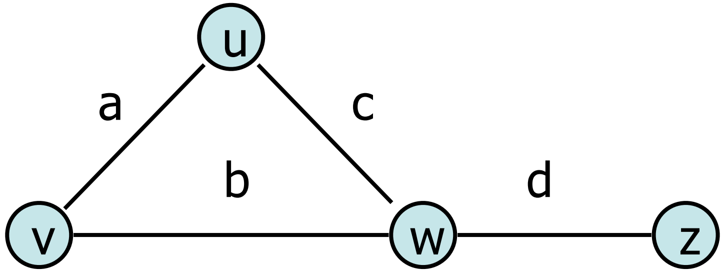
- incidentEdges(Vertex v);
- areAdjacent(Vertex v1, Vertex v2);

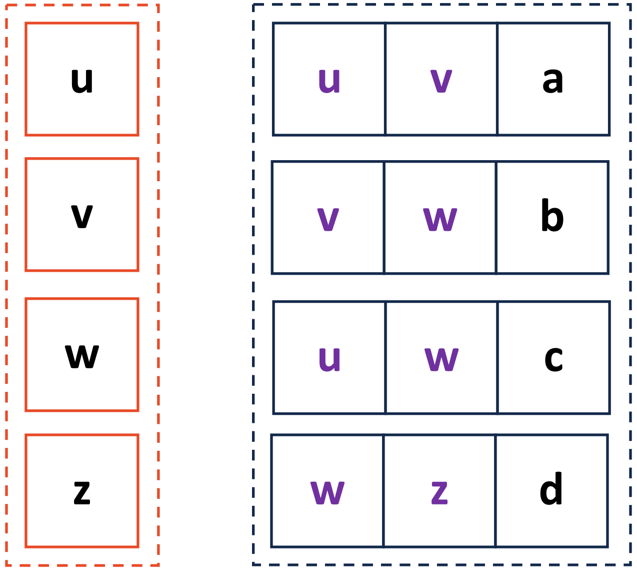
- origin(Edge e);
- destination(Edge e);

Graph Implementation Idea



Graph Implementation: Edge List

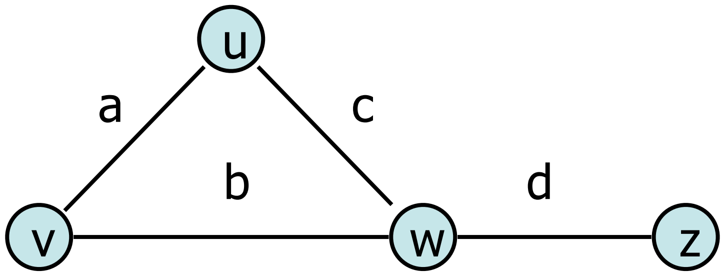


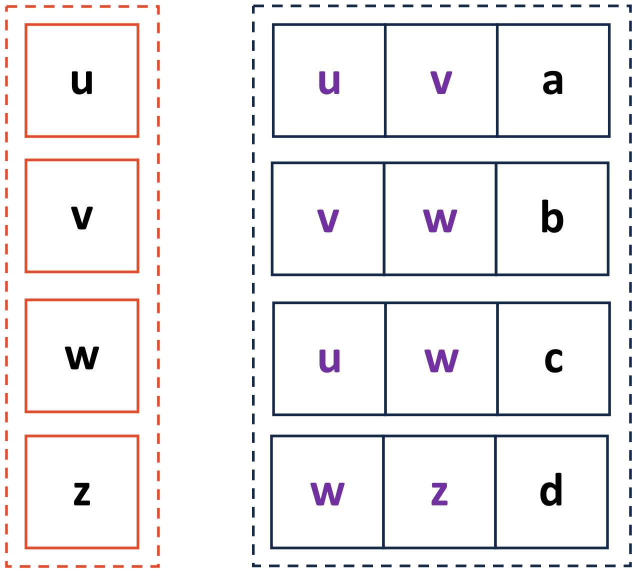


Vertex Collection:

Edge Collection:

Graph Implementation: Edge List

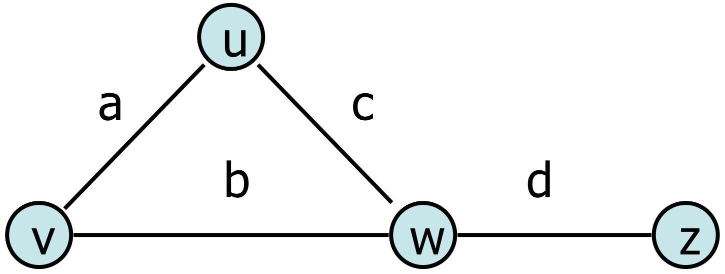


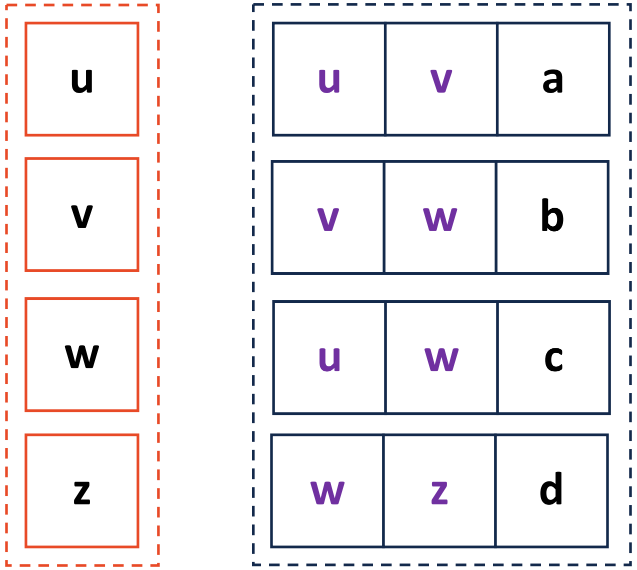


insertVertex(K key):

removeVertex(Vertex v):

Graph Implementation: Edge List



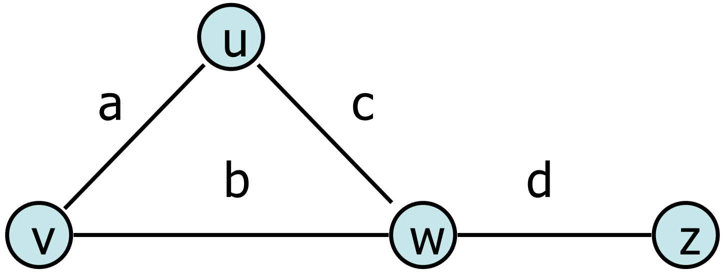


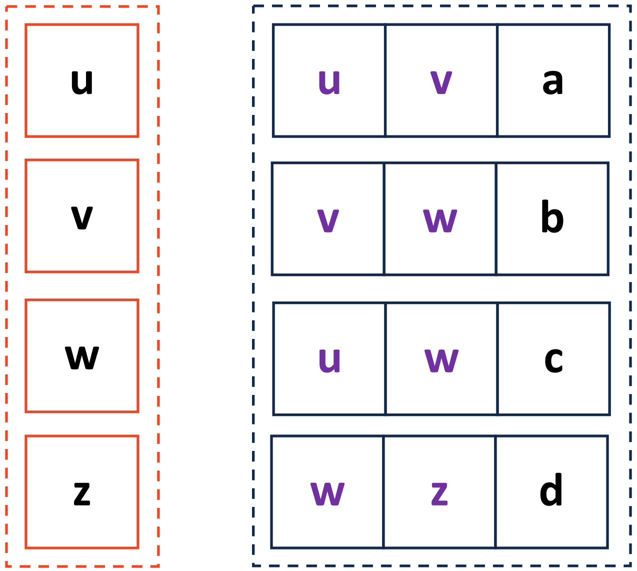
incidentEdges(Vertex v):

areAdjacent(Vertex v1, Vertex v2):

```
G.incidentEdges(v1).contains(v2)
```

Graph Implementation: Edge List





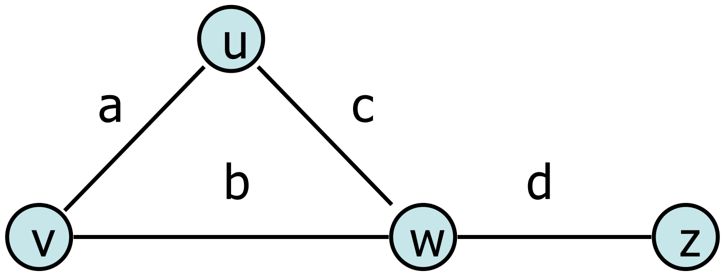
insertEdge(Vertex v1, Vertex v2, K key):

Graph Implementation: Edge List

Pros:

Cons:

Graph Implementation: Adjacency Matrix



insertVertex(K key);
removeVertex(Vertex v);
areAdjacent(Vertex v1, Vertex v2);
incidentEdges(Vertex v);

u	
v	
w	
z	

	u	v	w	z
u				
v				
w				
z				