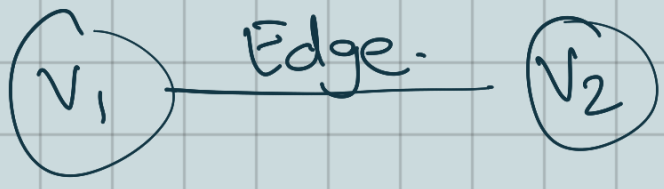
→ Vertices model objects
→ Edges model relationship b/w objects



→ Graphs are everywhere:

- → Social Network
- → Collaboration Network
- → Transportation Network
- → Computer Netwo
- → Connectomics
- → Genomics
- → Financial transa 'ons.
- → Etc.

→ Edges can be directed
Relationships can go one way or both ways.

→ Edges can we weighted
Denotes strength, distance. etc.

→ Edges and nodes can have metadata

Computation on Graphs

(1) Social Network queries

Q. Find all your friends who went to the same
Q. Finding common friends with someone.
Q. Social network recommending people whom you might know.

(2) Finding good clusters.

Q. Finding people wil similar interests.
Q. Detecting fraudulent websites
Q. Document clustering.
Q. Unsupervised learning.

→ Finding groups of vertices that are "well-connected" internally and "poorly-connected" externally.

(3) Subgraph finding

→ Finding or counting specific subgraphs inside a graph
→ Finding recurrent subgraph

Q. Node importance in social Network
Q. Function in biological networks.

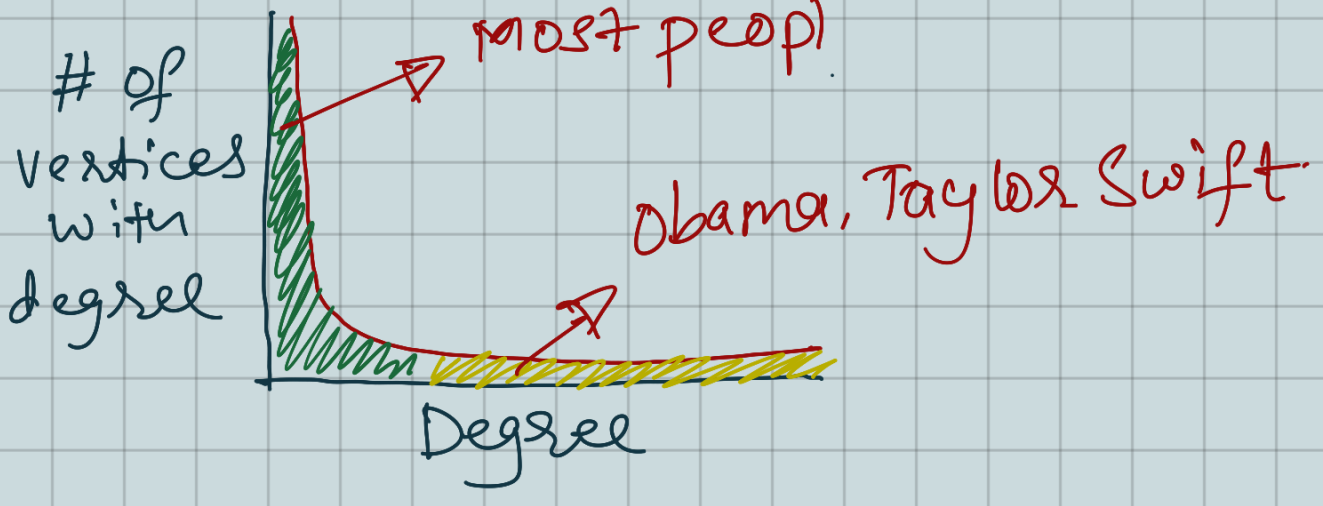
Properties of Real-world graphs

→ They are big.

Twitter: 41M vertices, 1.5B edges
63 GB
Web graph: 3.5B vertices, 128B edges
540 GB

→ Sparse ($m = cn$ for a small constant c)

→ Degrees can be highly skewed



→ studies have shown that many real-world graphs have a power-law distribution.

$$\#\text{vertices with deg. } d \approx a \times d^{-p}$$

$$(2 < p < 3)$$

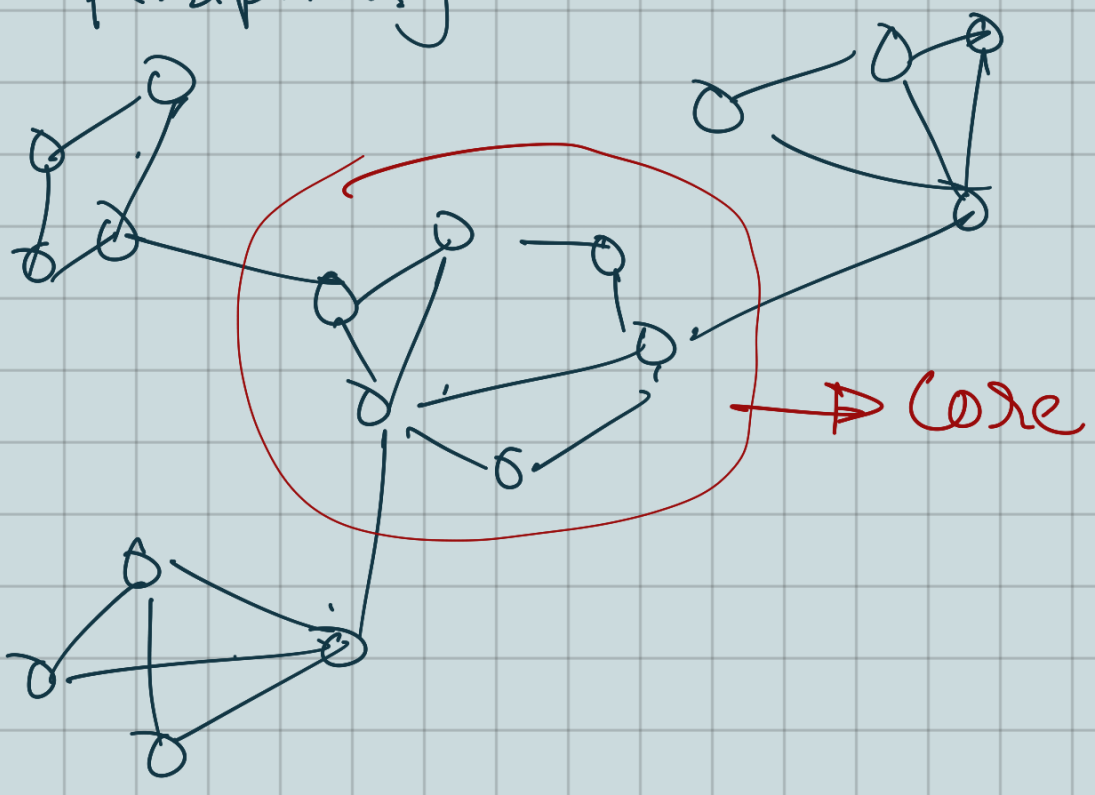
Small world phenomenon

⇒ Also, known as "six degrees of separation".

Core-periphery structure:

⇒ High-status nodes linked in a dense core.

⇒ Low-status nodes are on sparse "periphery"



Graph Representations

→ Adjacency matrix.

Vertices are labeled from 0 to n-1

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

→ 1 if edge exists,
0 otherwise

→ Space $O(n^2)$

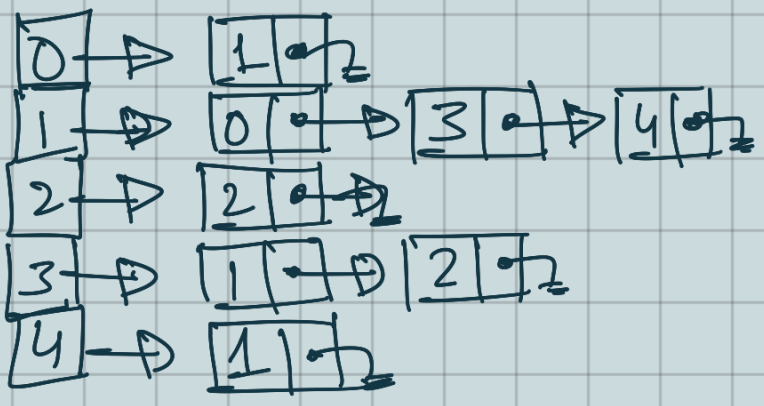
→ Edge list

→ Space $O(m)$

(0, 1)
(1, 0)
(1, 3)
(1, 4)
(2, 3)
(3, 1)
(3, 2)
(4, 1)

→ Adjacency list.

Array of pointers (one / vertex)
Each vertex has an onordered list of its edges



→ Space O(m+n)
→ Can substitute linked lists with arrays for better cache performance.

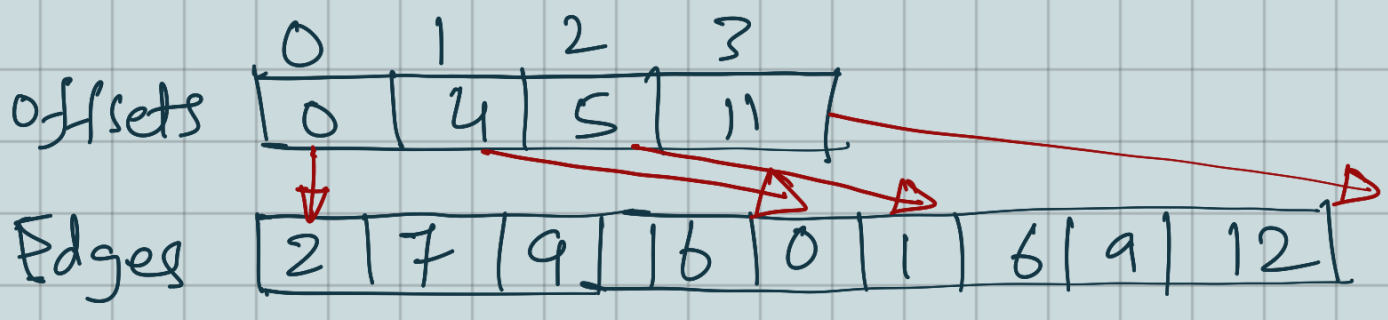
→ Trade off: more expensive to update graph.

→ Compressed Sparse rows

Two arrays : Offsets , Edges.

Offset[i] stores the offset of where vertex i's edges start in Edges.

Vertex IDs



Q. How do we know the degree of the vertex?

→ Degree$_i$ = Offset[i+1] - offset[i];

→ Space Usage : O(m+n)

→ Can also store value on the edges with an additional array or interleaved with Edges.

Trade offs in Graph Representation.

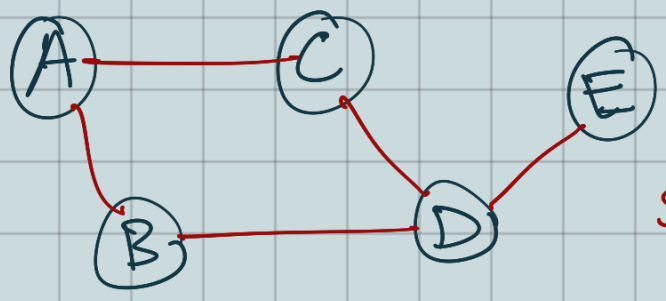
⇒ What is the cost of different operations?

	Adj. matrix	Edge list	Adj. List	Compressed Sparse rows
Storage cost / scan	$O(n^2)$	$O(m)$	$O(m+n)$	$O(m+n)$
Add edge	$O(1)$	$O(1)$	$O(1)$	$O(m+n)$
Delete edge	$O(1)$	$O(m)$	$O(\deg(v))$	$O(m+n)$
Get neighbors	$O(n)$	$O(m)$	$O(\deg(v))$	$O(\deg(v))$
Is neighbor	$O(1)$	$O(m)$	$O(\deg(v))$	$O(\deg(v))$

→ There are variants / combinations of there representations.

Breadth-first Search

→ Given a source vertex s, visit the vertices in order of distance from s.



→ Possible outputs:

Vertices in order they were visited:
- D, B, C, E, A

Distance from each vertex to s

A	B	C	D	E
2	1	1	0	1

→ BFS requires $O(n+m)$ work for n vertices & m edges

Ligra:

⇒ A lightweight interface for graph algorithms that is particularly well suited for graph traversal problems.

⇒ Compare-and-Swap (CAS)

⇒ An atomic instruction that takes three arguments

- a memory location (loc)
- an old value (oldV)
- a new value (newV)

⇒ if the value stored at loc is equal to the oldV, it store newV at loc and returns true

⇒ Else Returns false

$G = (V, E)$

VertexSubset $U \subseteq V$
: a subset of vertices

Ligra Interface:-

① SizE (U : vertexSubset) : N
Returns |U|

② EdgeMap (G : graph,
U : vertexSubset,
F : (V × V → bool,
C : (V → bool) :
vertexSubset

→ For an unweighted graph G = (V, E) EdgeMap applies the function F to all edges with source vertex in U and target vertex satisfying C

$$E_a = \{(u,v) \in E \mid u \in U \wedge C(v) = true\}$$

$$Out = \{v \mid (u,v) \in E_a \wedge F(u,v) = true\}$$

③ VertexMap (U: VertexSubset
F: vertex → bool):

VertexSubset

Applies F to every vertex in U.

It returns a vertexSubset.

Out = {u ∈ U | F(u) = true}.

★ For both EdgeMap & VertexMap the function F can run in parallel.

★ VertexSubset has two representations:

→ set of integers if sparse
→ bit array of |V| if dense.

BFS in Ligra:-

Parents = {-1, ... -1}

```
Procedure UPDATE(s,d)
    return (CAS(&Parents[d],-1,s))

Procedure COND(i)
    return (Parents[i] == -1)

Procedure BFS(G,r) → r is the root
  Parents[r] = r
  Frontier = {r}

  while (SIZE(FRONTIER) ≠ 0) do
    Frontier = EdgeMap(G, Frontier,
                       UPDATE,
                       COND)
```