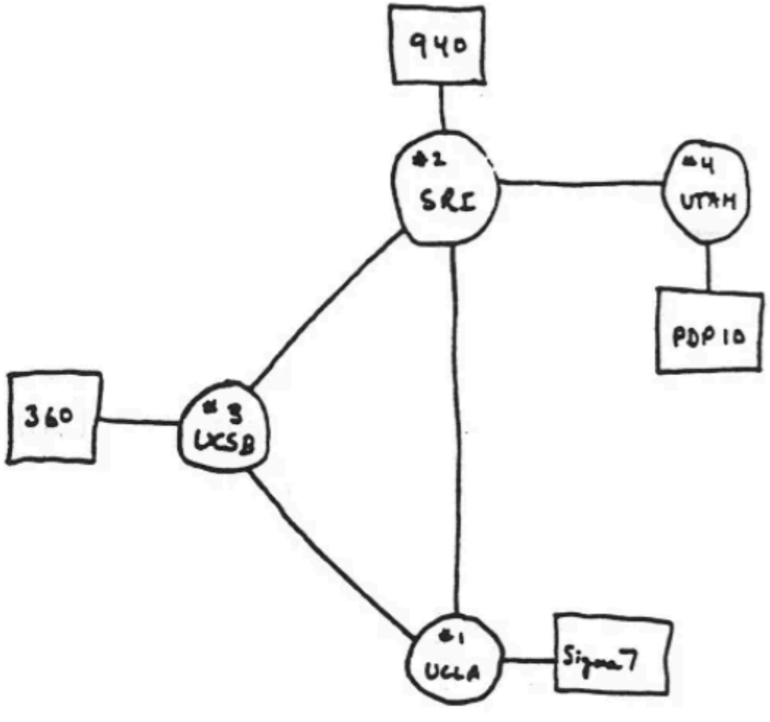
GRAPHS



THE ARPA NETWORK

DEC 1969

4 NODES

*The IBM 360, the IMP, and the workstations were all located in North Hall.
https://jeweledplatypus.org/news/text/ucsbnet.html

Graphs: terminology

$G = (V, E)$

Path: Sequence of vertices that are adjacent to each other

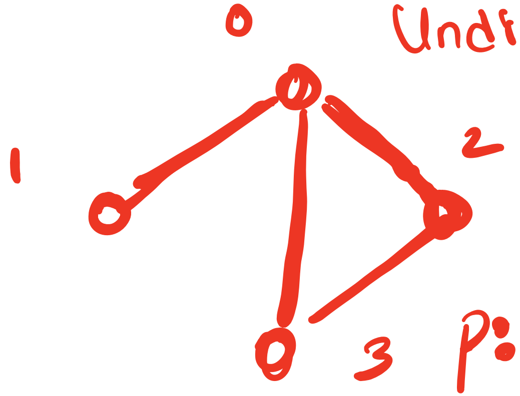
V: Vertices - collection of objects

E: Edges - connections between vertices

directed (i, j) - order pair

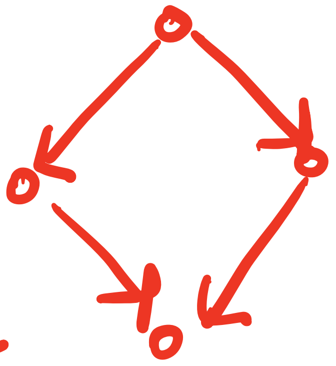
undirected $\{i, j\}$ - unorder pair

Undirected

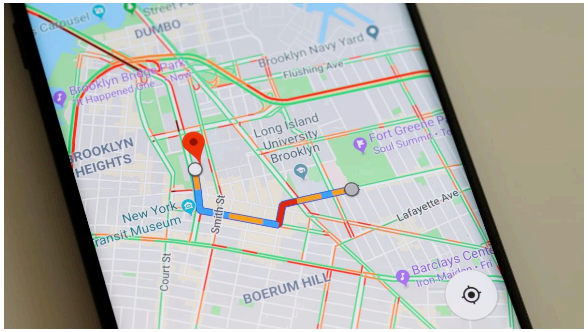


p: 1, 0, 3, 2

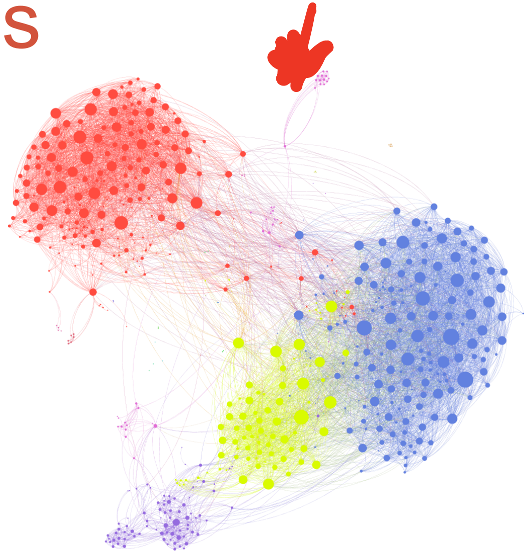
Directed



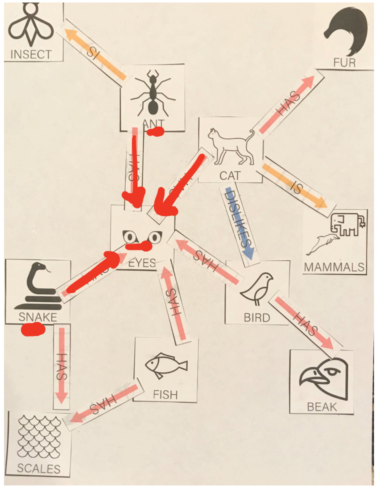
Graph applications



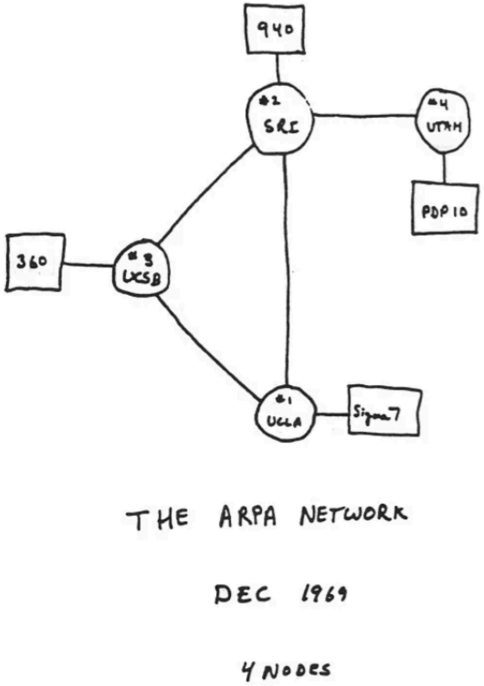
Road networks



Social networks



Semantic networks



Computer networks*

Remember: If your problem maps to a well-known graph problem, it usually means you can solve it blazingly fast!

How do we discuss running time on graphs?

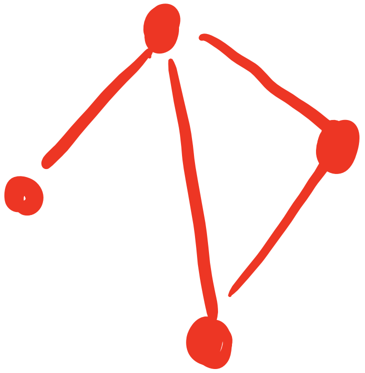
$G = (V, E)$

Size of a graph: expressed as

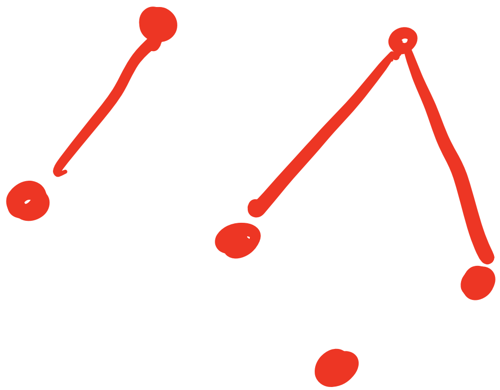
No. of edges, denote as m or $|E|$

· No. of vertices, denote as n or $|V|$

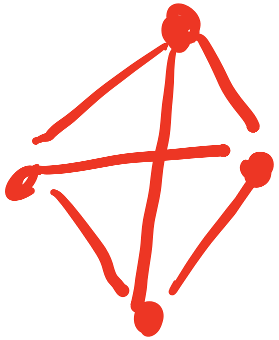
Connected



Disconnected



Fully connected



Concept Question

What is minimum and maximum number of edges in a connected graph with n vertices?

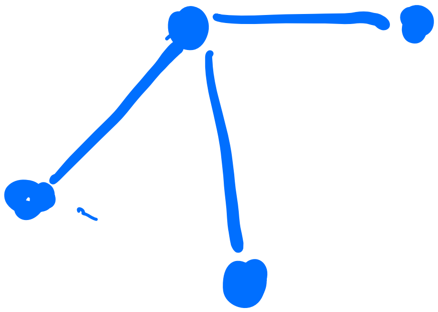
(no parallel edges

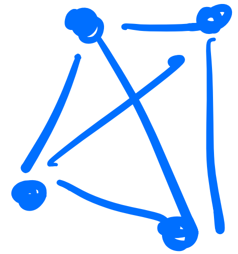
A. 0 and n

B. (n - 1) and n (n - 1) / 2

C. (n - 1) and n^2

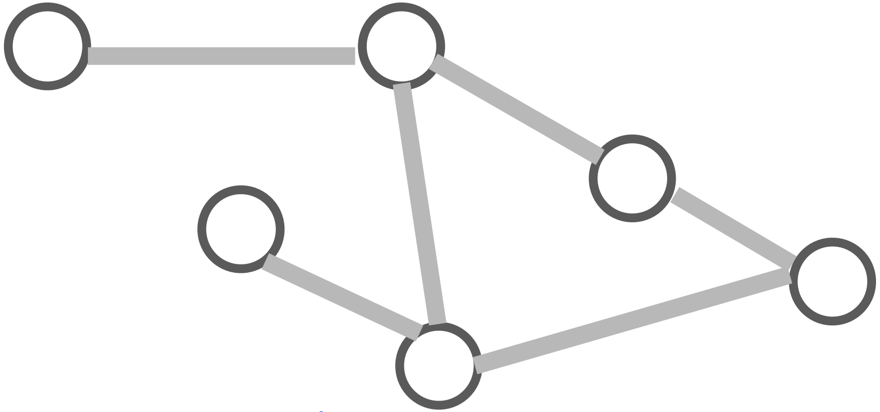
D. (n - 1) and 2^n



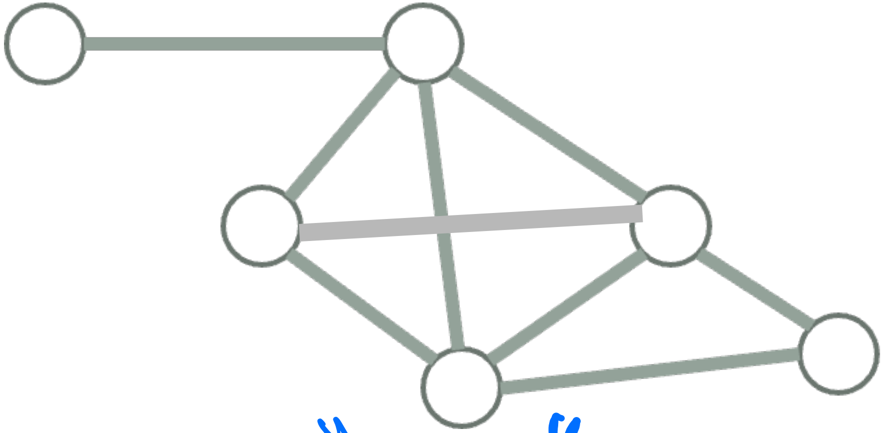


n choose 2 $\rightsquigarrow \binom{n}{2} = \frac{n!}{(n-2)!\,2!}$

Sparse vs. Dense Graphs



"sparse" connected



"dense"

n^2

m

A dense graph is one where |E| is "close to" $|V|^2$.

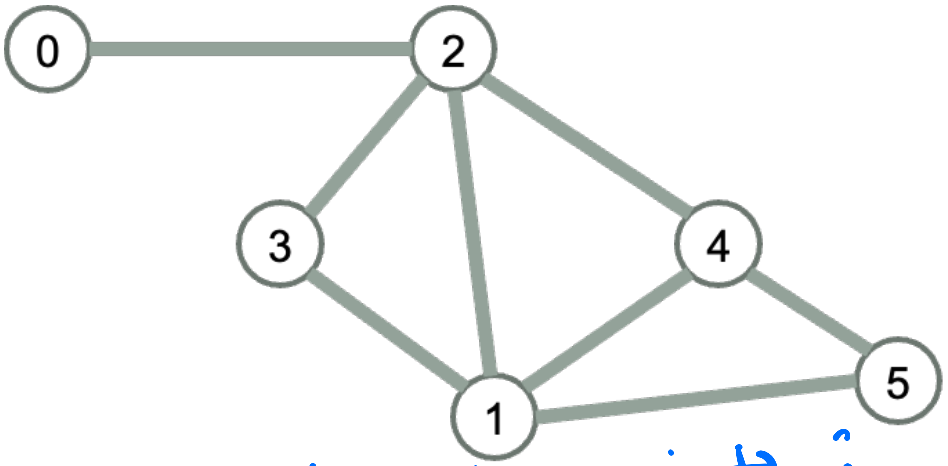
A sparse graph is one where |E| is "closer to" |V|.

m n

Adjacency Matrix Representation of a Graph

Represent the graph by a n x n binary valued adjacency matrix, A

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

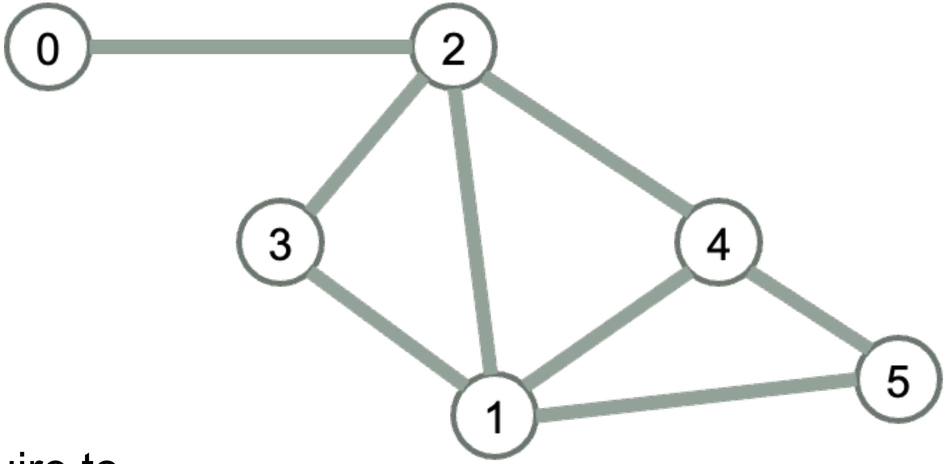


$A[i,j] = 1$, if there is an edge from i to j

$= 0$, otherwise

Adjacency Matrix

Represent the graph by a n x n binary valued adjacency matrix, A
A[i, j] = 1, if there is an edge from i to j

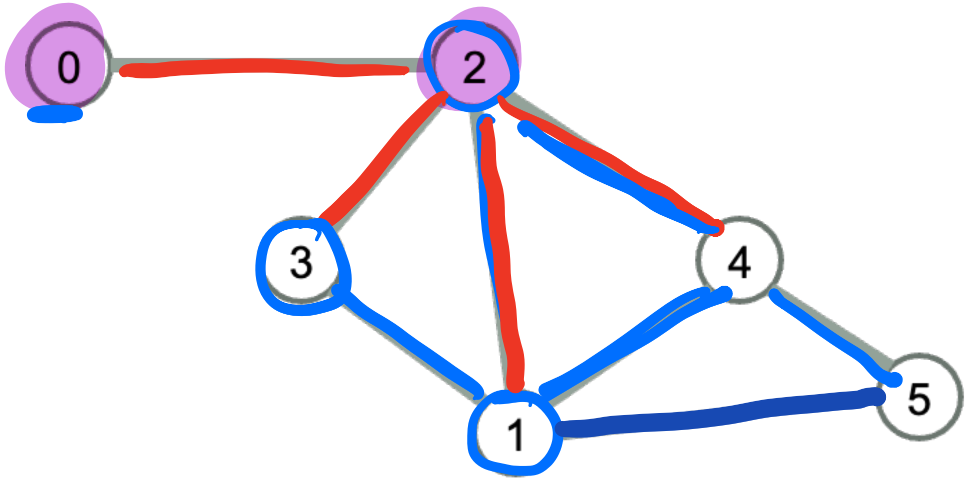


How much space does an adjacency matrix require to represent a graph with n vertices and m edges?

A. O(n)
B. O(m)
C. O(n + m)
D. O(n^2)
E. O(mn)

Adjacency List Representation of a Graph

Vertices and edges stored as lists
Each vertex points to all its edges



0 : 2
1 : 2, 3, 4, 5
2 : 0, 1, 3, 4
3 : 2, 1
4 : 1, 2, 5
5 : 1, 4

How much space does an adjacency list require to represent a graph with n vertices and m edges?

A. O(n)
B. O(m)
C. O(n + m)
D. O(n^2)
E. O(m.n)

Assume each vertex is identified by an integer index

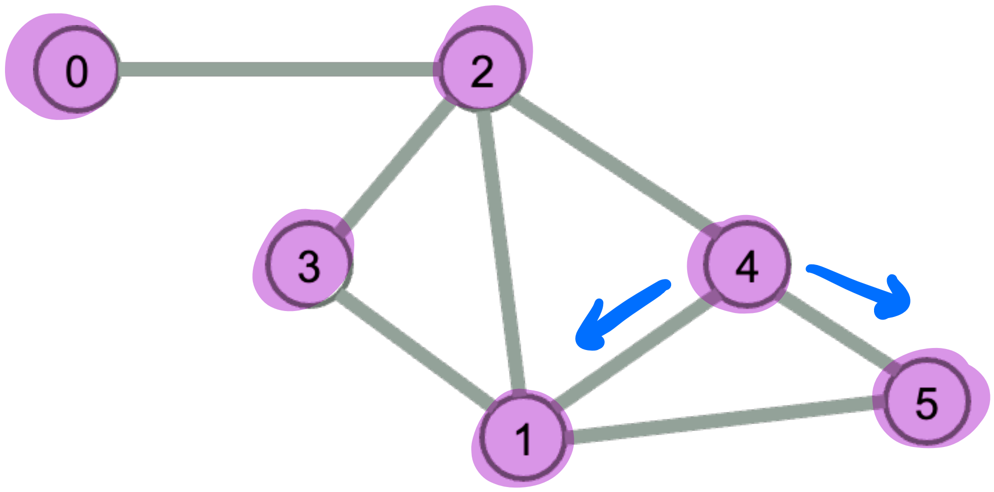
```
class graph{

  private:
  _________ adjlist;
};
```

Choose the ADT to represent the adjacency list

```
A. vector<int>
B. vector<vector<int>>
C. list<vector<int>>
D. vector<list<int>>
E. set<list<int>>
```

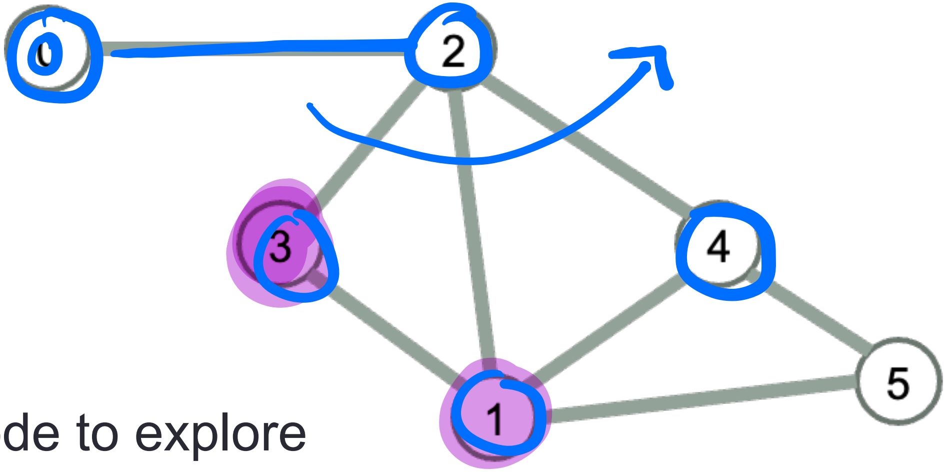
Graph search: general approach

Starting with a source node
- find everything that can be explored
- don't explore anything twice



Graph search: breadth first (BFS)

Explore all the nodes reachable from a given node before moving on to the next node to explore



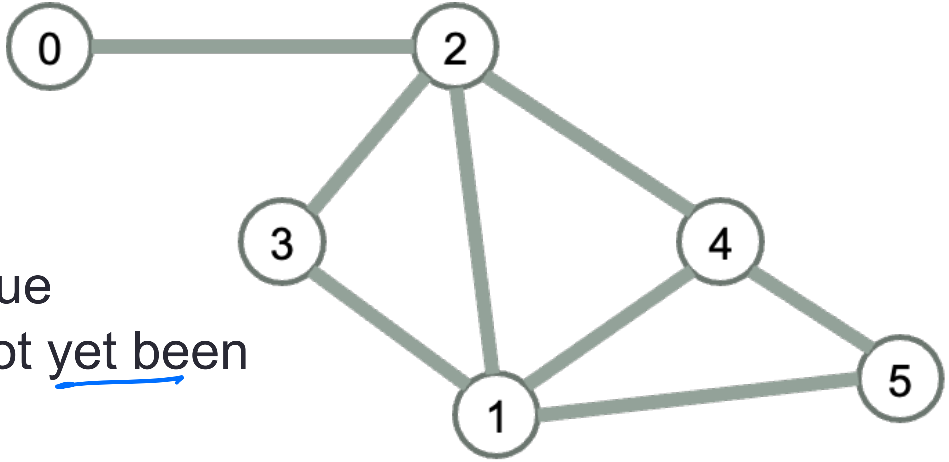
0, 2, 1, 3, 4, 5

Assume BFS chooses the lower number node to explore first, in what order does BFS visit the nodes in this graph

A. 0, 1, 2, 3, 4, 5
B. 0, 1, 3, 2, 4, 5
C. 0, 2, 3, 1, 4, 5
D. 0, 2, 1, 3, 4, 5
E. Something else

BFS Traverse: Sketch of Algorithm

Start at source *s*;
Mark *s* as visited
push *s* into a queue
while the queue is not empty:
- pop the vertex *u* from the front of the queue
- for each of *u*'s adjacent nodes that has not yet been visited (*v*):
    - Mark *v* as visited
    - Push *v* in the queue



Questions:

- What difference(s) do you observe with the BSF we covered for trees?
- What data do you need to keep track of for each node?

Implement the graph ADT given in your handout

```
class graph{
  public:
     graph(int n = 0) { // n is the number of vertices
        adjList = vector <list ;<int > (n);
     }
     void addEdge(int from, int to);
     bool hasEdge(int i, int j) const;
     vector<bool> bfs(int source) const;
    // performs a breadth first search starting from the source and returns a vector with vertices that were visited set to true
     bool isValidPath(const vector<int> & path) const; // returns true if the input path exists in the graph
     bool isReachable(int source, int dest) const;// returns true if a path exists from the source to the destination
  private:
     vector< list <int>> ; adj list ;
};
```

Link to hand out: https://bit.ly/CS24F23GraphsHandout