Data Structures and Algorithms

(ESO207)

Lecture 22

Graphs

- Notations and terminologies
- Data structures for graphs
- A few algorithmic problems in graphs

Why Graphs ??

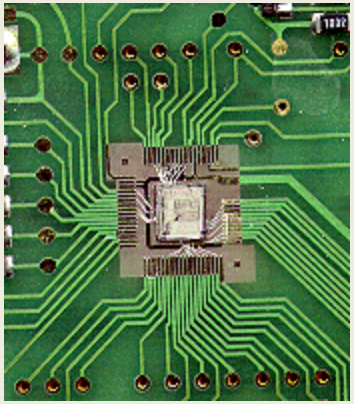
Finding **shortest route** between cities

Given a network of **roads** connecting various cities,

compute the <u>shortest route</u> between any two **cities**.

Just imagine how you would solve/approach this problem.

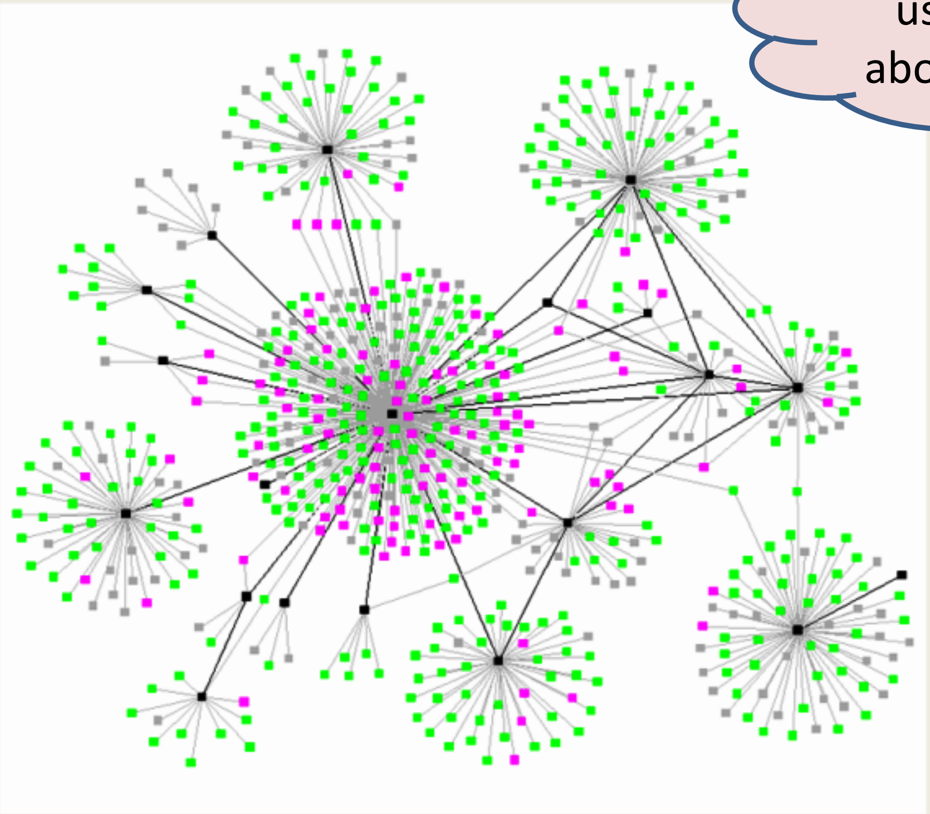
Embedding
an integrated circuit on mother board



How to embed **ports** of various ICs on a plane and make **connections** among them so that

- No two connections **intersect** each other
- The **total length** of all the connections is **minimal**

A social network or world wide web (WWW)



Can we make some useful observations about such networks ?

diameter

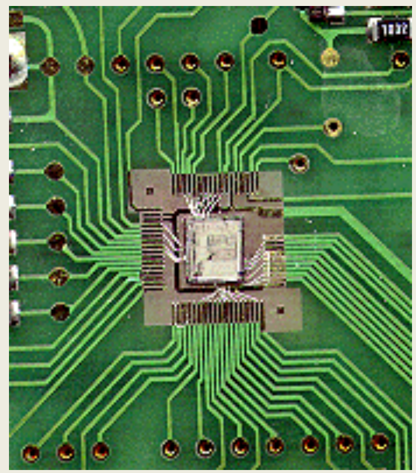
degree distribution

Do you know about the "6 degree of separation principle" of the world ?

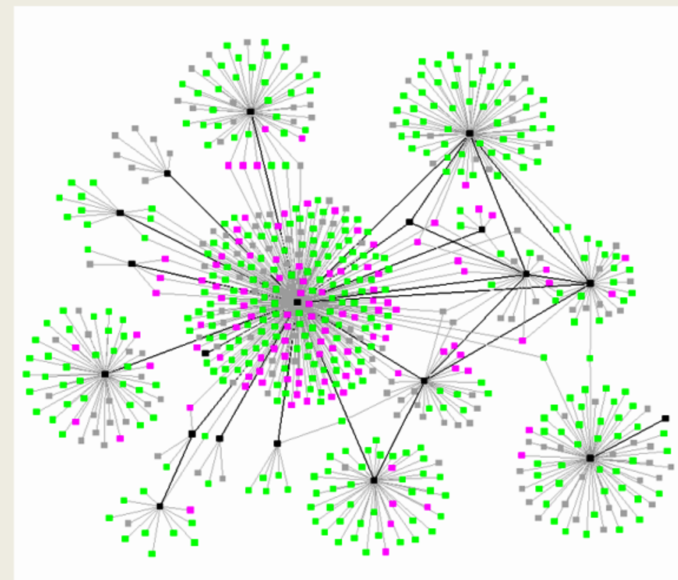
Visit the site https://en.wikipedia.org/wiki/Six_degrees_of_separation

How will you model these problems ?

I

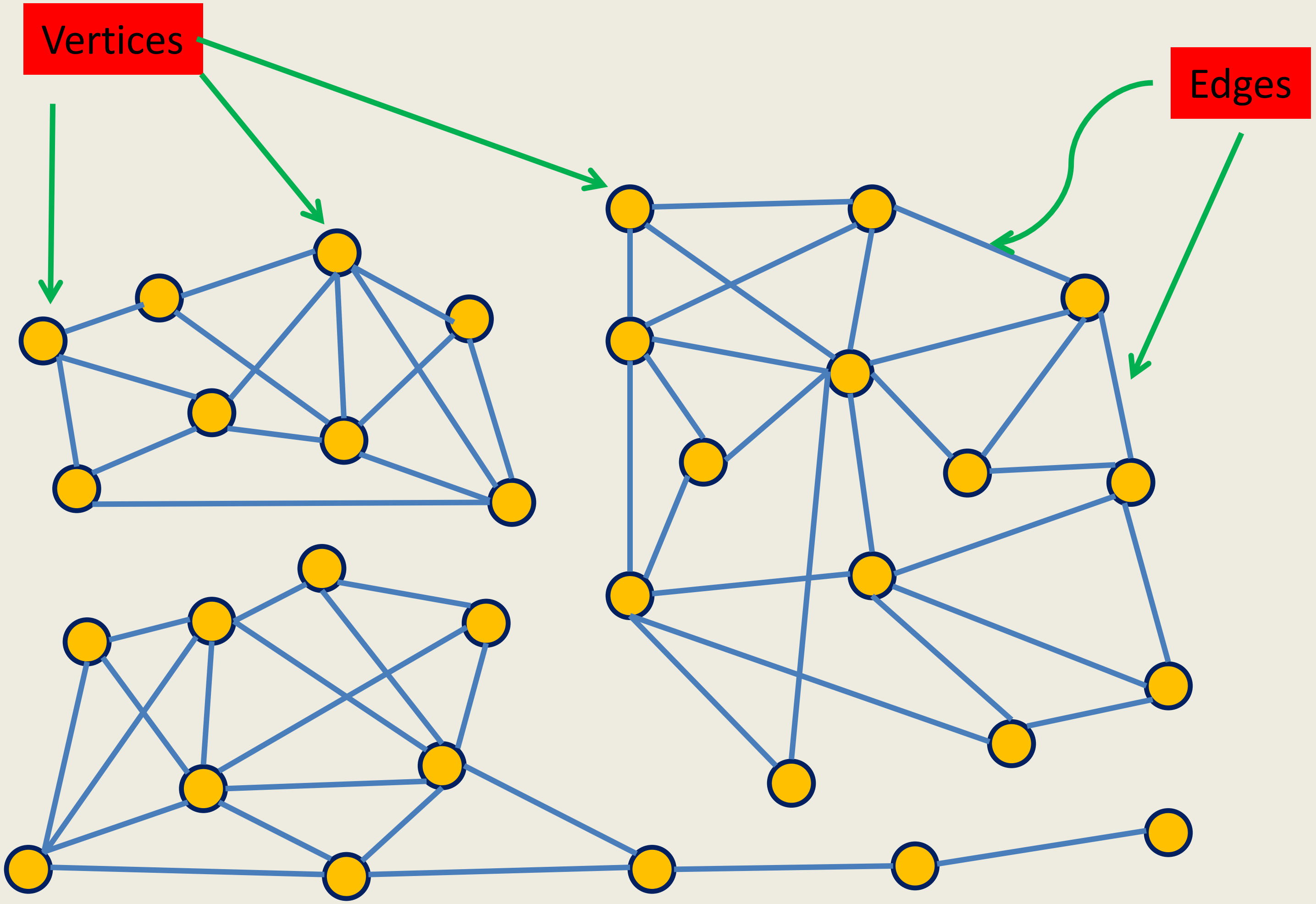


II



III

Graph



Graph

Definitions, notations, and terminologies

Graph

A graph G is defined by two sets

- V : set of vertices
- E : set of edges

Is there any relation between E and V ?

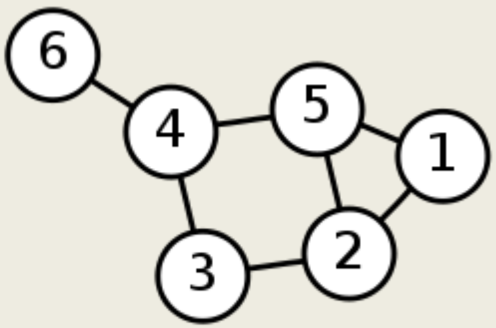
$E \subseteq (V \times V)$

Notation:

- A graph G consisting of vertices V and edges E is denoted by (V,E)

Types of graphs

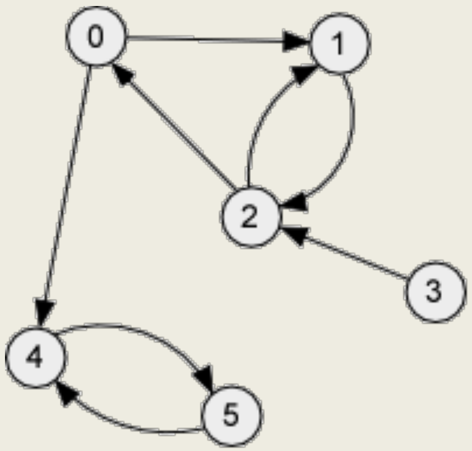
Undirected Graph



V= {1,2,3,4,5,6}

E= {(1,2), (1,5),
(2,5), (2,3),
(3,4),
(4,5), (4,6)}

Directed Graph



V= {0,1,2,3,4,5}

E= { (0,1), (0,4),
(1,2),
(2,0), (2,1),
(3,2),
(4,5),
(5,4) }

Notations

Notations:

- $n = |V|$
- $m = |E|$

Note: For directed graphs, $m \leq \boxed{n(n-1)}$

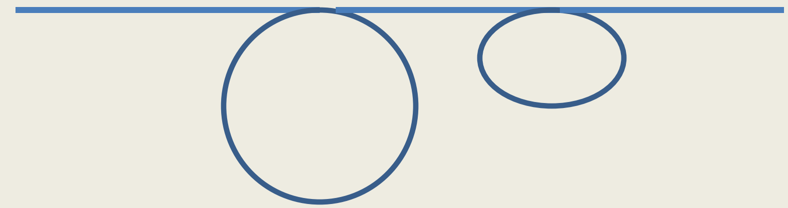
For undirected graphs, $m \leq \boxed{n(n-1)/2}$

Walks, paths, and cycles

Walk:

A sequence $<v_0, v_1, \ldots, v_k>$ of vertices is said to be a **walk** from x to y

- $x = v_0$
- $y = v_k$
- For each $i<k$, $(v_i, v_{i+1}) \in E$

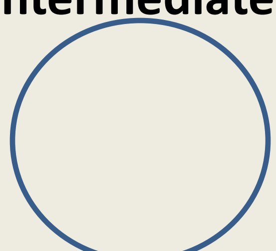


Path:

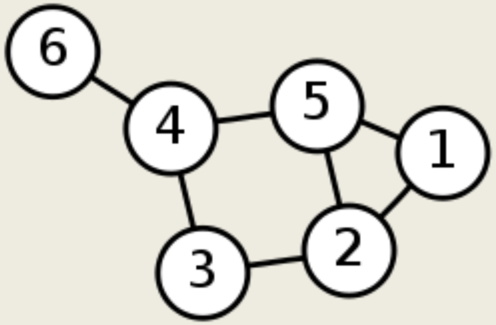
A walk $<v_0, v_1, \ldots, v_k>$ on which <u>no vertex appears twice</u>.

Cycle:

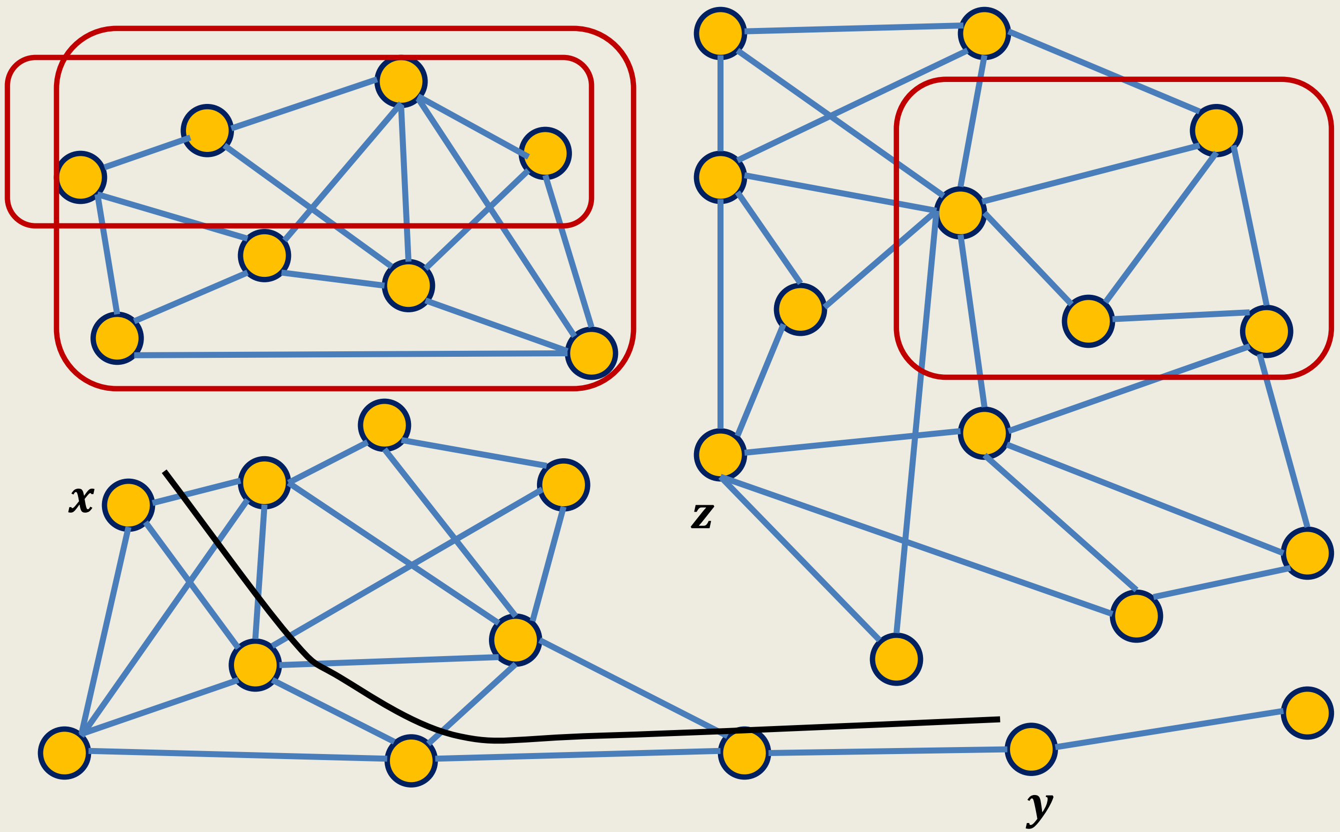
A walk $<v_0, v_1, \ldots, v_k>$ where no **intermediate** vertex gets repeated and $v_0 = v_k$



Examples



- **<1,5,4>** is a **walk** from **1** to **4.**
- <**1,3,2,5**> is **not** a **walk.**
- <**1,2,5,2,3,4,5,4,6**> is a **walk** from **1** to **6.**
- <**1,2,5,4,6**> is a **path** from **1** to **6.**
- <**2,3,4,5,2**> is a **cycle.**



two vertices are said to be ***connected*** if there is a **path** between them

Connected component:

A **maximal** subset of connected vertices

You can not add any more vertex to the subset and still keep it connected.

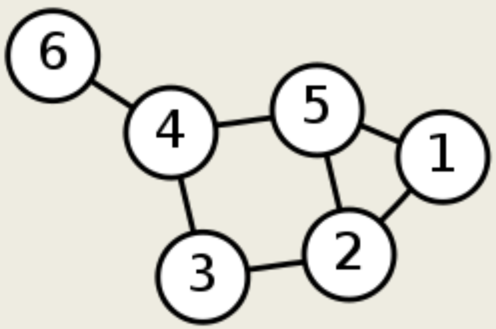
Data Structures for Graphs

Vertices are always numbered

$\mathbf{1, \dots, n}$ **Or** $\mathbf{0, \dots, n-1}$

Link based data structure for graph

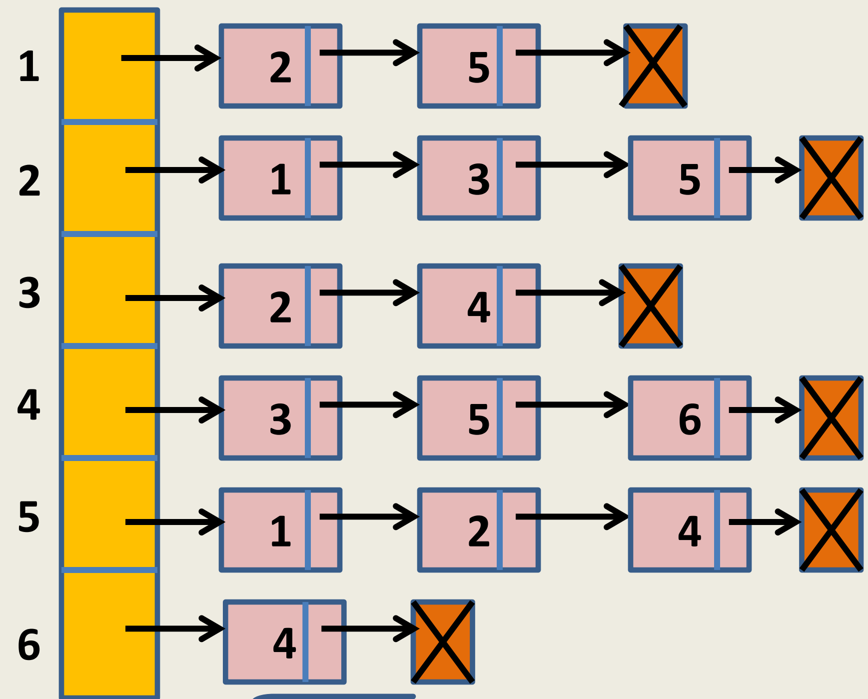
Undirected Graph



V= {1,2,3,4,5,6}

E= {(1,2), (1,5),
(2,5), (2,3),
(3,4),
(4,5), (4,6)}

Adjacency Lists



Size = O($n + m$)

Link based data structure for graph

Advantage of Adjacency Lists :

- Space efficient
- Computing all the neighbors of a vertex in optimal time.

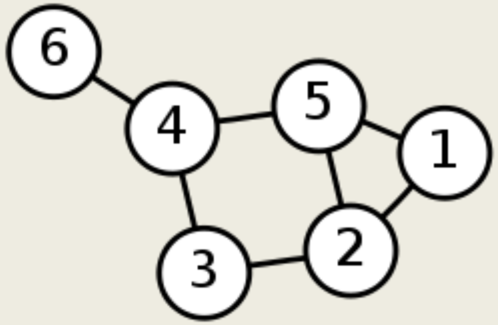
Disadvantage of Adjacency Lists :

- How to determine if there is an edge from ***x*** to ***y*** ?

(O(n) time in the worst case).

Array based data structure for graph

Undirected Graph



V= {1,2,3,4,5,6}

E= {(1,2), (1,5),
(2,5), (2,3),
(3,4),
(4,5), (4,6)}

Adjacency Matrix

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

Size = $O(n^2)$

Array based data structure for graph

Advantage of Adjacency Matrix :

- Determining whether there is an edge from ***x*** to ***y*** for any two vertices ***x*** and ***y***. in **O**(1) time

Disadvantage of Adjacency Matrix :

- Computing all neighbors of a given vertex ***x*** in **O**(n) time
- It takes **O**(n^2) space.

Which data structure is commonly used for storing graphs ?

Adjacency lists

Reasons:

- Graphs in real life are sparse ($\boldsymbol{m} \ll \boldsymbol{n^2}$).
- Most algorithms require processing neighbors of each vertex.

➔Adjacency matrix will enforce **O**($\boldsymbol{n^2}$) bound on time complexity for such algorithm.

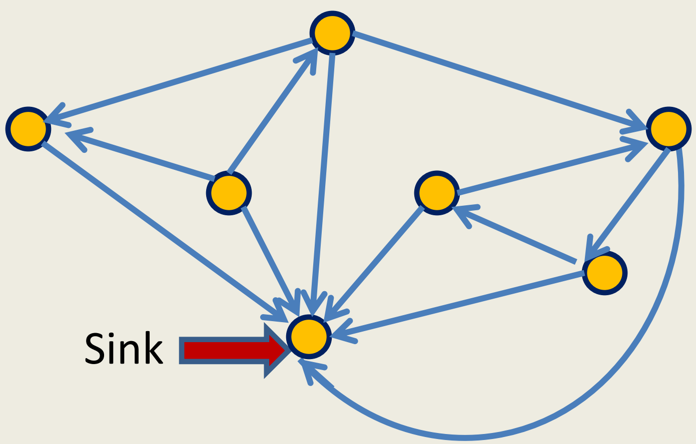
But, there are a few **exceptions**

An interesting problem

(Finding a sink)

A vertex ***x*** in a given directed graph is said to be a **sink** if

- There is no edge **emanating** from (leaving) ***x***
- Every other vertex has an edge **into** ***x***.



Given a directed graph ***G***=(***V***,***E***) in an **adjacency matrix** representation, design an **O**(n) time algorithm to determine if there is any **sink** in ***G***.

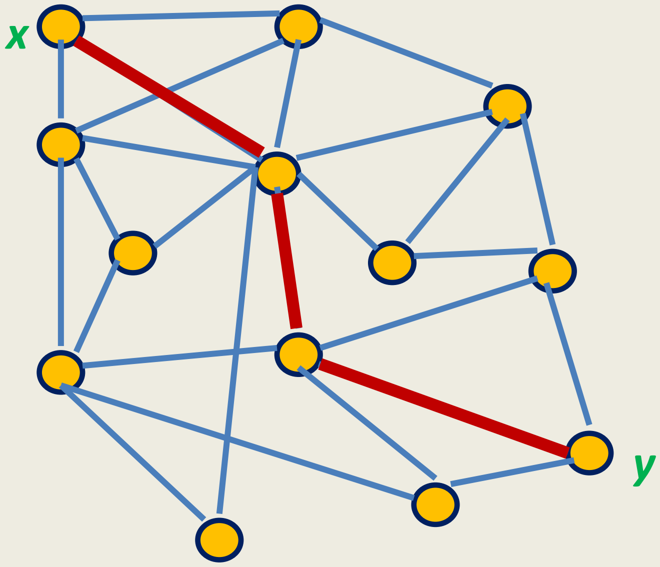
Graph traversal

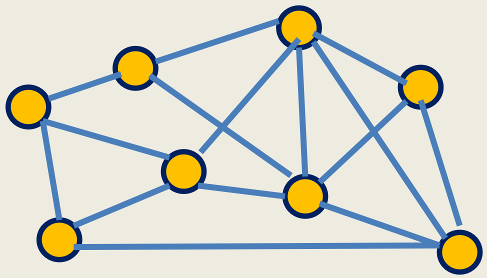
Topic for the next class

Graph traversal

Definition:

A vertex ***y*** is said to be reachable from ***x*** if there is a **path** from ***x*** to ***y***.





Graph traversal from vertex *x*: Starting from a given vertex ***x***, the aim is to visit all vertices which are reachable from ***x***.

Non-triviality of graph traversal

- **Avoiding loop**:
  How to avoid visiting a vertex multiple times ?
  (keeping track of vertices already visited)

- **Finite number of steps** :
  The traversal **must stop** in finite number of steps.

- **Completeness** :
  We must visit **all** vertices reachable from the start vertex ***x***.

Spend some time on these points and see how can you design such a traversal algorithm

A sample of Graph algorithmic Problems

- Are two vertices x and y connected ?
- Find all connected components in a graph.
- Is there is a cycle in a graph ?
- Compute a path of shortest length between two vertices ?
- Is there is a cycle passing through all vertices ?