Agenda: 0) Review of graph definitions

1) Tree properties

2) Hand Shake Lemma

3) Graph representations

4) Depth first search

5) Breadth first search

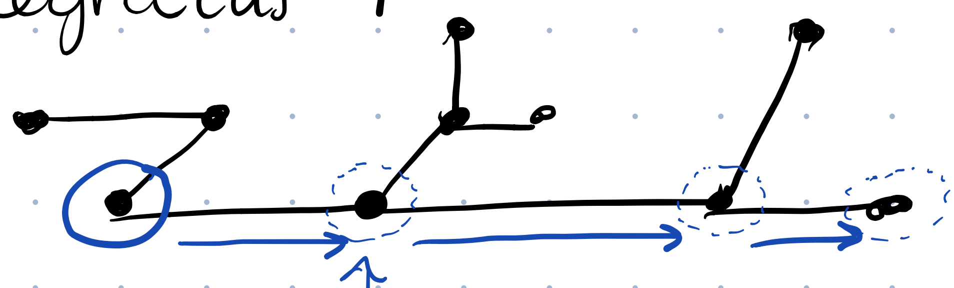
Review

- Graph — undirected / directed
- Vertex
- Edge
- Degree
- (simple) path
- reachable
- connected components
- (simple) cycle
- complete graph
- subgraph
- complement
- tree — unrooted / rooted
- forest
- Bipartite graph

Additional tree properties

(contained in the point below, not covering)

- Any tree has at least one vertex u with $degree(u) = 1$



pick a new edge since either we only have one edge

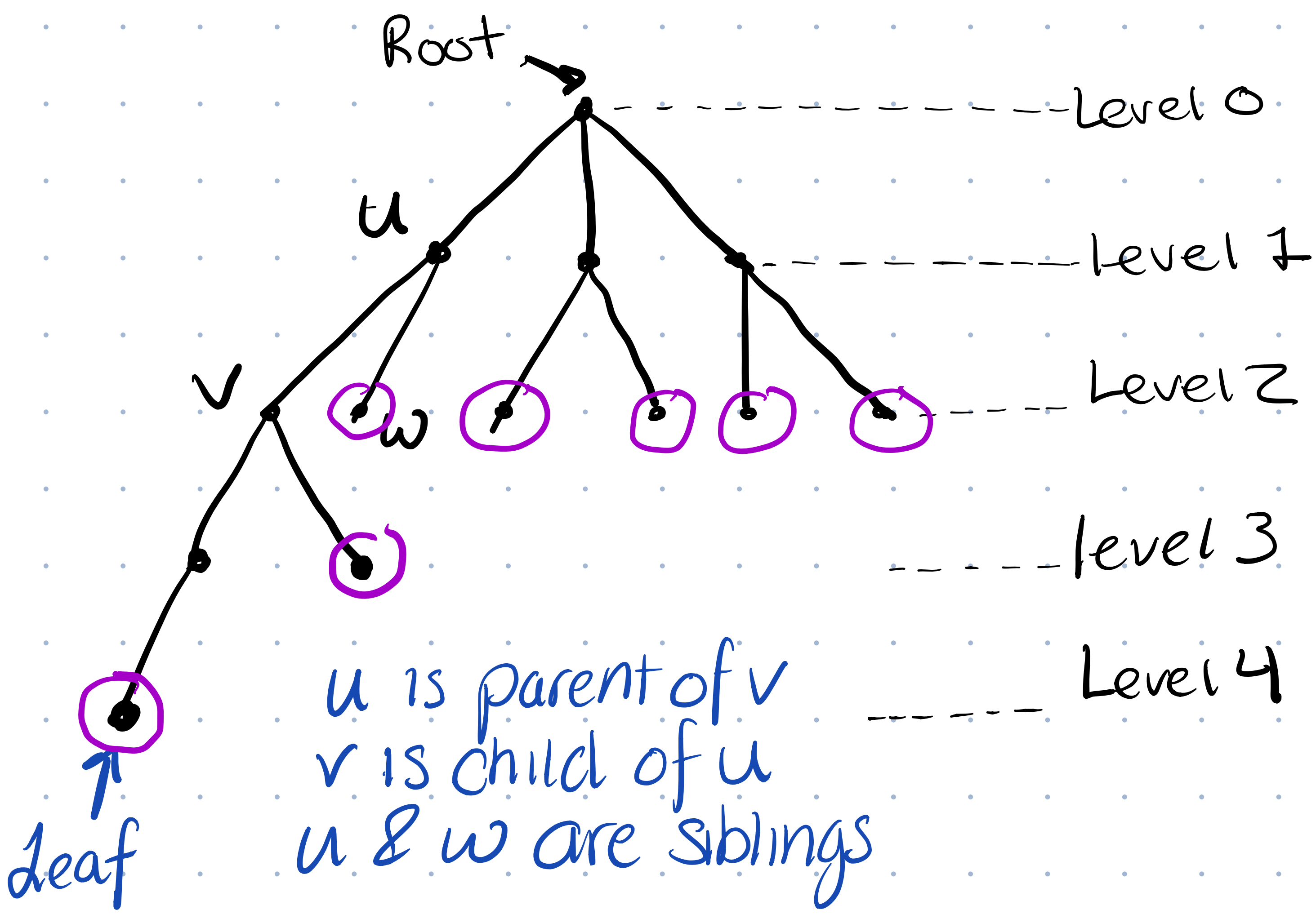
-or-

degree > 1 and it must exist.

won't return to the same vertex b/c we have no cycles in a tree

(starting here)

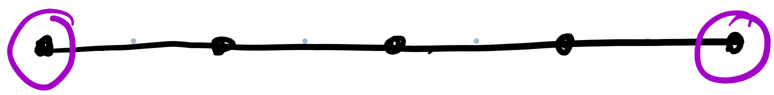
- Rooted trees have a few more definitions:



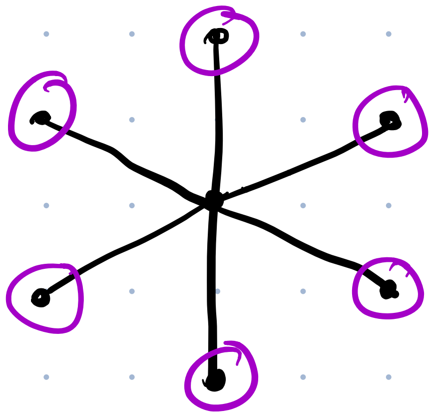
u is parent of v
v is child of u
u & w are siblings

Vertices with degree = 1 are leaves
(no children)

- Trees must have between 2 and $|V|-1$ leaves



2 leaves



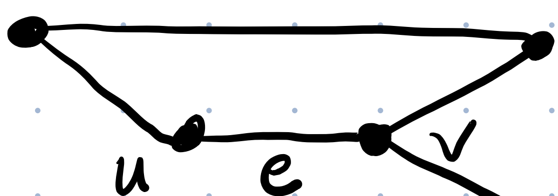
$|V|-1$ leaves

Handshake Lemma

Let G be any ~~undirected~~ directed graph:

$$\sum_{v \in V} \text{degree}(v) = 2|E|$$

$$\sum_{v \in V} \text{in-degree}(v) + \text{out-degree}(v) = 2|E|$$

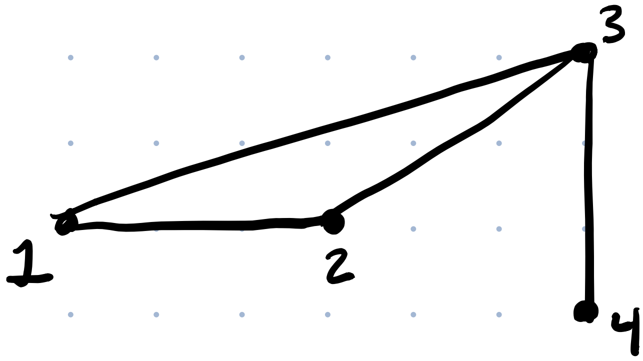


Pick some arbitrary edge e, contributes one degree count to u & one to v
$\therefore$ e contributes 2 to the total degree of G, and since e was chosen arbitrarily, this w.l.o.g can apply to any edge.

Corrollary: The total degree of a graph is even

Graph Representation

- Adjacency matrix:



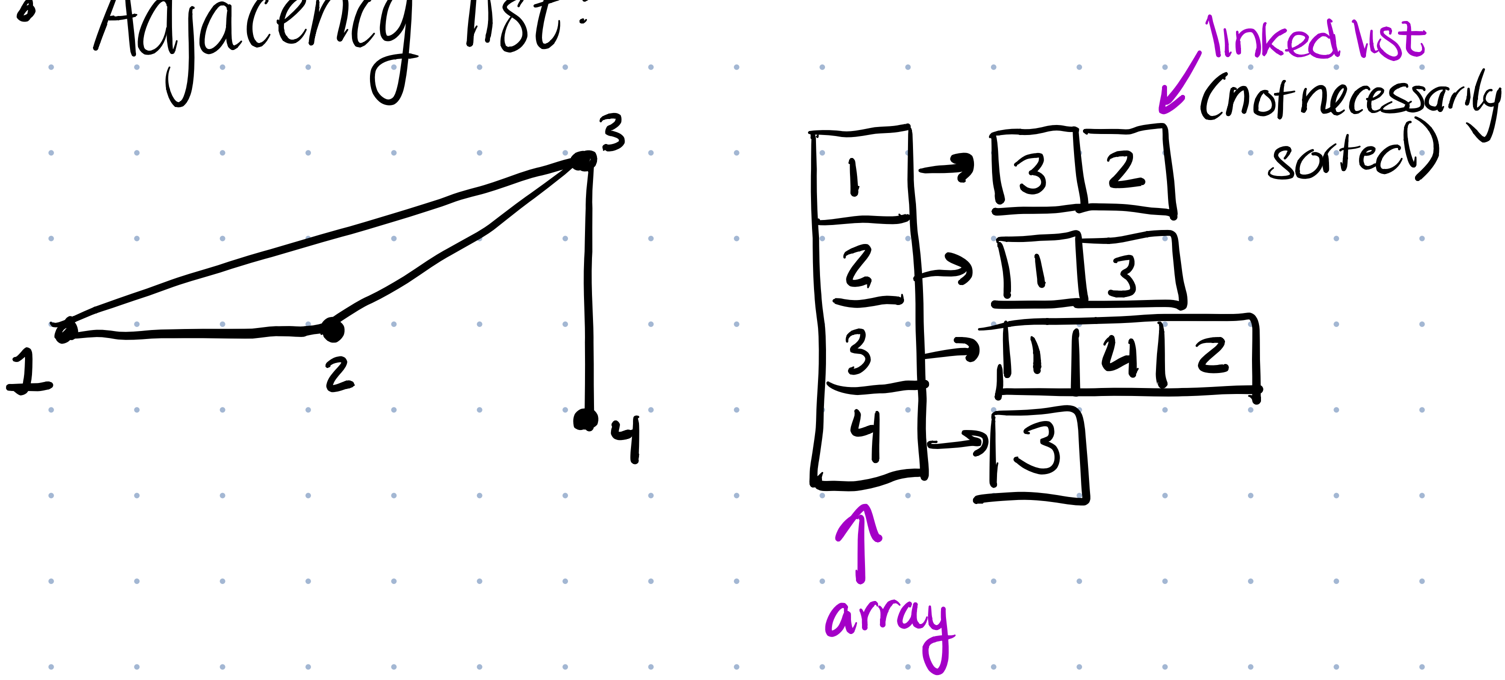
$$\begin{array}{c c} & \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{array}{c} 1 \\ 2 \\ 3 \\ 4 \end{array} & \begin{bmatrix} 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix} \end{array}$$

Note symmetry around diagonal

$$a_{ij} = \begin{cases} 1 & \text{iff edge exists between vertex } i \,\&\, j \\ 0 & \text{otherwise} \end{cases}$$

$O(1)$ How do we check if an edge exists between v_1 & v_3 check entry a_{13} or a_{31} in the matrix. To find all adjacent vertices, scan through the row. $O(|V|)$

- Adjacency list:



Find an edge → scan through array
Find all edges → return linked array

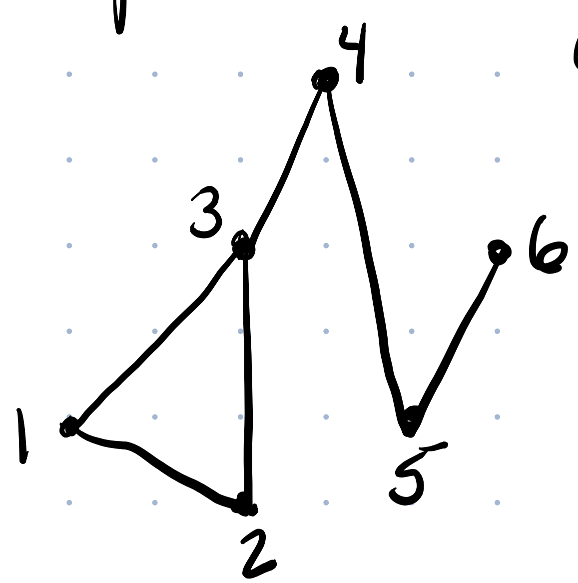
- When to use one or another?

Matrix: - Dense (graph is almost complete)
- static
- looking up specific edges, not searching through all adjacent vertices

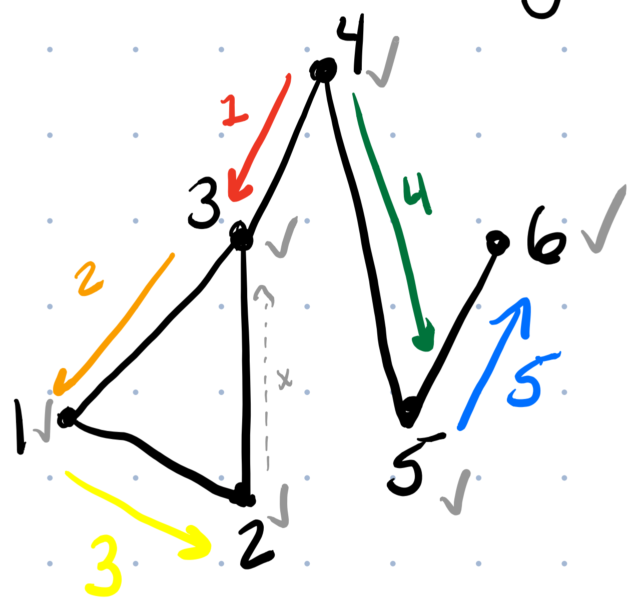
List: - Sparse (graph complement is mostly complete)
- updating
- searching through all adjacent vertices

~~Searching~~ Traversal Algorithms in Graphs

- Depth - First Search: Main idea, traverse as far as you can until either can go no further or you would visit a node already visited. Then return to the previous node & repeat.

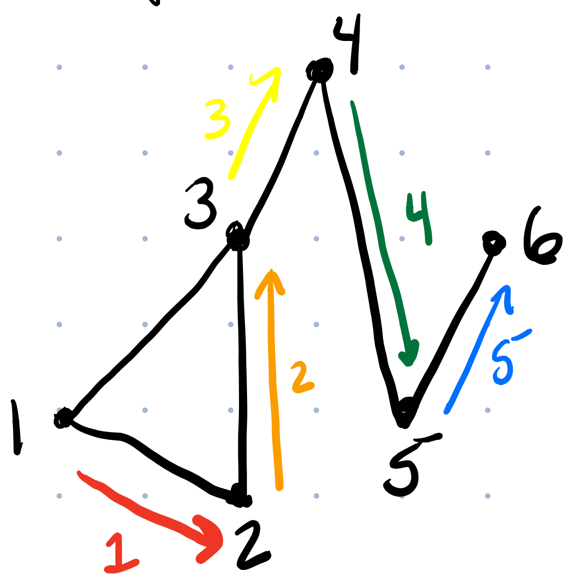


Example starting at 4.



4, 3, 1, 2, 5, 6

Example at 1:



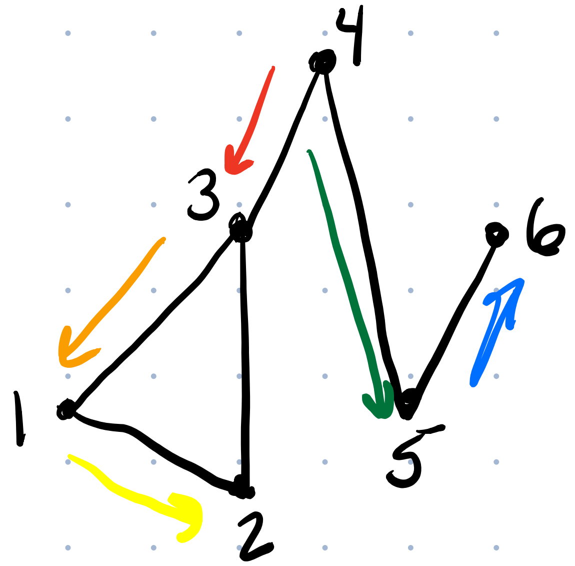
1, 2, 3, 4, 5, 6

In more depth:

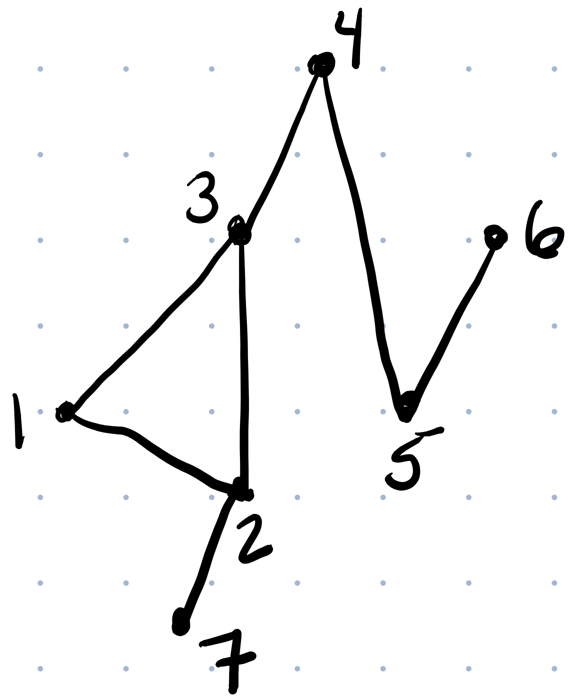
1	2	3	4	5	6
X	X	X	X	X	X

Visited

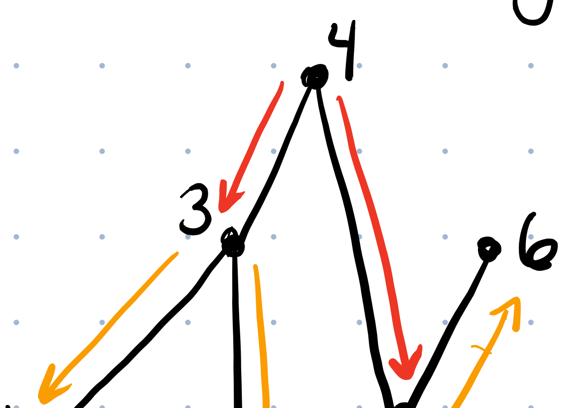
```
DFS(Vertex v)
    Visited[v] = true
    for adjacent u to v:
        if visted[u] = false
            DFS(u)
```



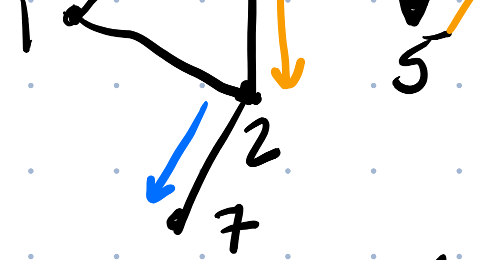
• Breadth - First Search: Main idea, traverse all your children & then traverse the chidren of the first then the second, etc ...



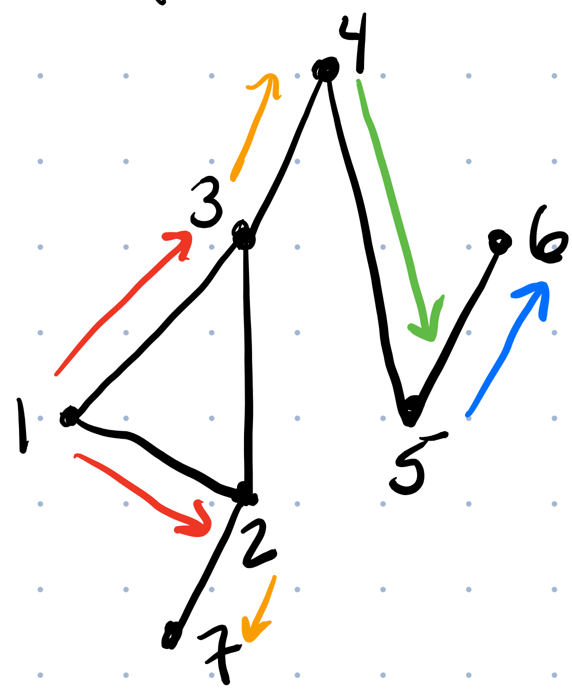
Example starting at 4.



4, 3, 5, 1, 2, 6, 7



Example at 1:



1, 2, 3, 7, 4, 5, 6

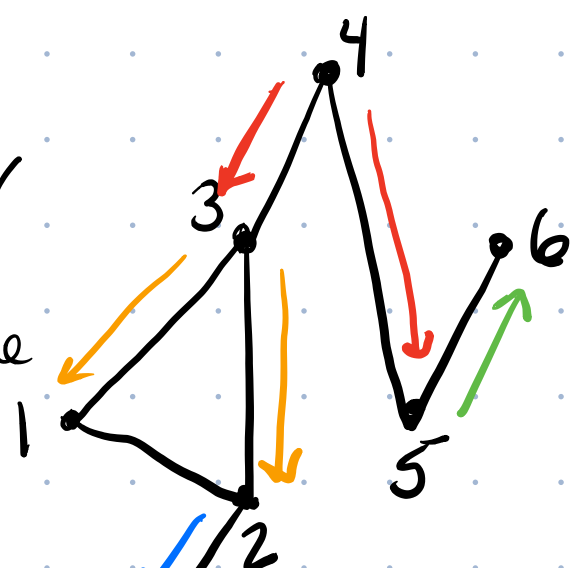
In more depth:

1	2	3	4	5	6	7
x	x	x	X	x	x	x

Visited

~~4~~
~~3~~
~~5~~
X
~~2~~
~~6~~
~~7~~

```
while queue ! empty
  v top item in queue
  for all u adjacent to v
    if u not visited
      visited[u] = true
      add to u to queue
```



Queue