Graph Basics

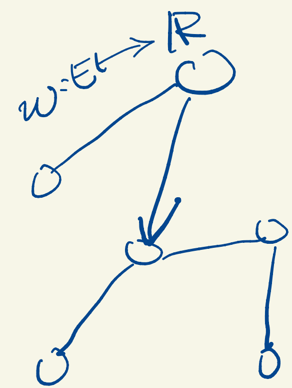
Xiangyu Guo.

(I) What is a graph?

Def: $G = (V, E)$: a collection of nodes (vertices) and edges connecting vertices.

Exmp:

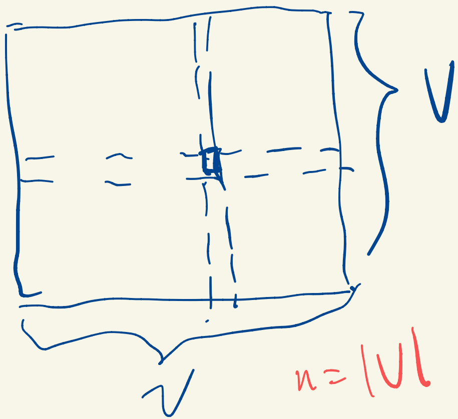
- Transportation network.
  - e.g.: $V = \{\text{cities}\}$, $E = \{\text{highways}\}$
- weight $w(i,j)$ = distance / length of roads between city i and j
- direction: one-way road.



(II) Representation

① Adjacency matrix

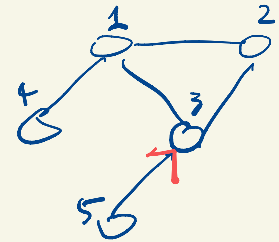
$G = (V, E)$

$A =$ 

$n = |V|$

$$A[i,j] \begin{cases} = 1 & \Longleftrightarrow (i,j) \in E \\ = w(i,j) & \Longleftrightarrow (i,j) \in E \;\&\; w(i,j) \end{cases}$$

Exmp:



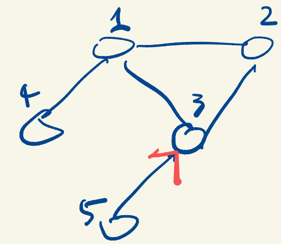
$$A = \begin{bmatrix} 0 & 1 & 1 & 1 & 0 \\ 1 & 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 & \cancel{1} \to 0 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \end{bmatrix}$$

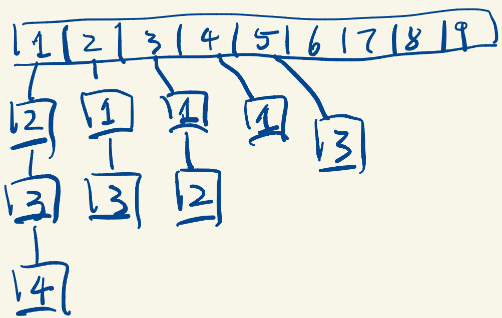
degree d_v:

\# of edges touching v.

② Adjacency List.

$G = (V, E)$. V is set of vertices



V: 

$\longrightarrow O(n)$

length of list of vertex $v = d_v$

$\therefore$ Total size/length of lists $= \sum_v d_v \leq 2|E| = 2m$

E.g. Assume every edge is undirected.

(i,j) appears exactly twice in all lists

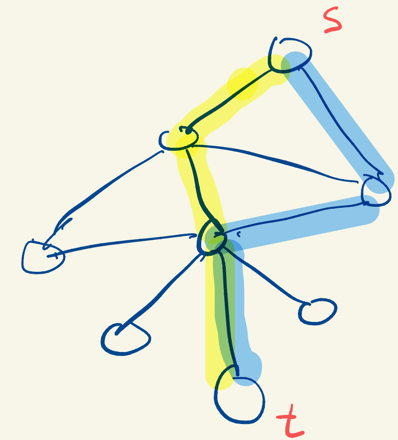
▷ Complexity

- $G = (V, E)$, $|V| = n$, $|E| = m$ (assume $0 \le m \le \frac{n(n-1)}{2}$)
- $d_v :=$ number of neighbors of v.

	Adj Matrix	Adj List
space usage	$O(n^2)$	$O(m+n)$
time to check if $(u,v) \in E$	$O(1)$	$O(d_v) = O(m)$
time to list all neighbors of v	$O(n)$	$O(d_v)$

Terminologies

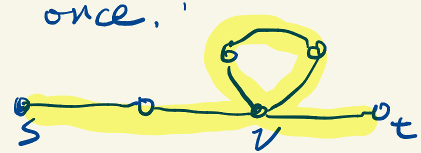
① Path : (s,t)-path. : sequence of edges.



$(s, v_0), (v_0, v_1), \cdots, (v_N, t)$

- simple path: every vertex appears in the path at most once.

counter example :

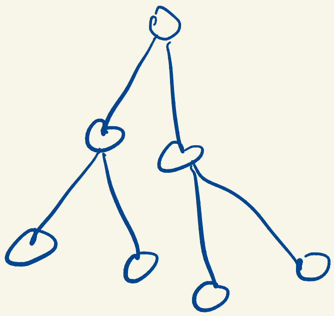


② Cycle : seq of edges with same beginning & ending.

- simple cycle : every vertex (except for the beginning one) appears ≤ 1 time in the seq.

③ Tree: a type of graph

- connected: $\exists$ path between every pair of vertices.
- no cycles.



(II) Connectivity Problem

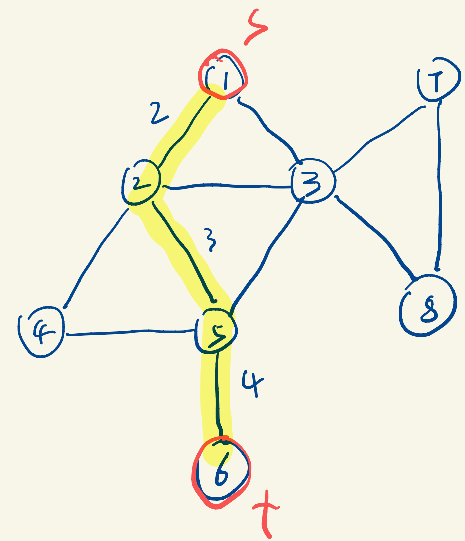
Problem:

Input: graph $G = (V, E)$

Two vertices $s, t \in V$

Output: "Yes" if $\exists$ (s,t)-path

"No" if otherwise.

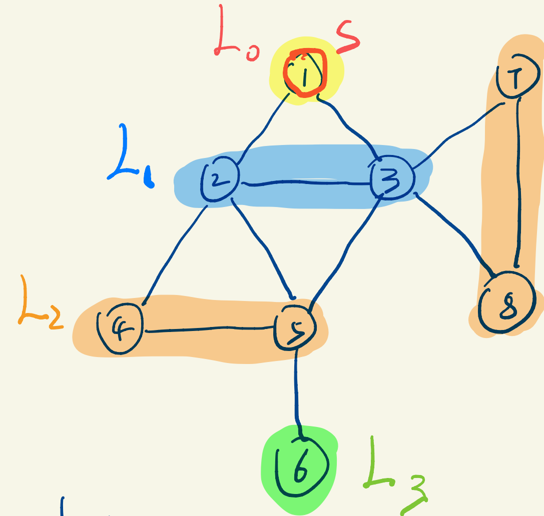


- Variants.
  - Shortest path problem: when edges have weight (distance)
  - Connected components:

▷ Breadth-First Search (BFS)

Input: $G = (V, E)$, $s \in V$

Output: All $t \in V$ s.t. s & t are connected.



Strategy:

Explore the graph level by level.

$L_0 = \{s\}$

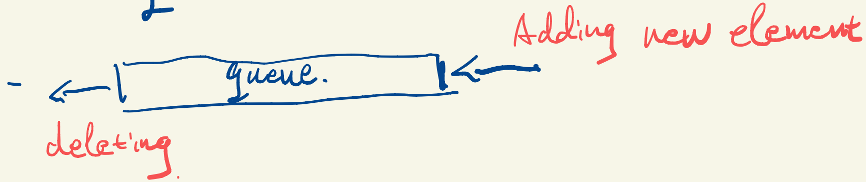
$L_1 = \{$all vertices that's not in L_0 but have an edge to $L_0\}$

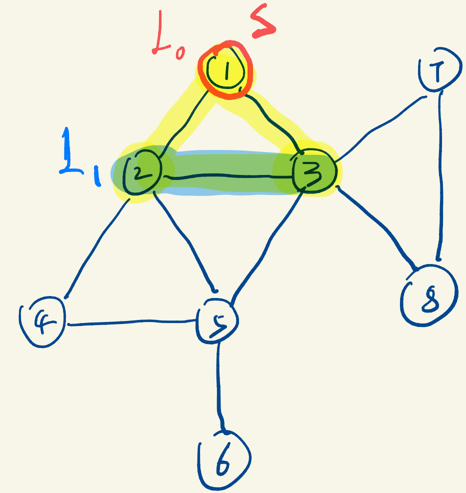
⋮

$L_n = \{$all vertices that's not in $L_0 \cup L_1 \cup \cdots \cup L_{n-1}$, but have an edge to $L_0 \cup L_1 \cup \cdots \cup L_{n-1}\}$.

▷ Implementing BFS

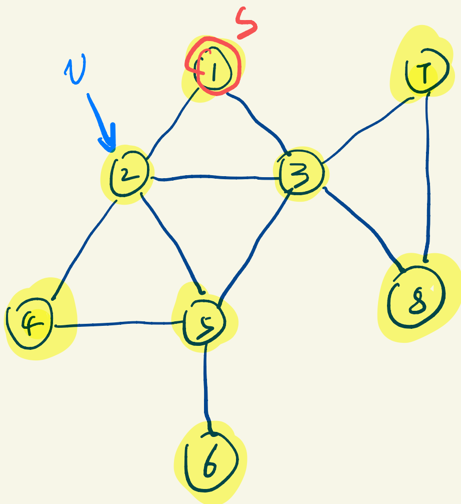
- Remember vertices already visited:
  - visited[v] = 1
- Use queue to traverse visited vertices



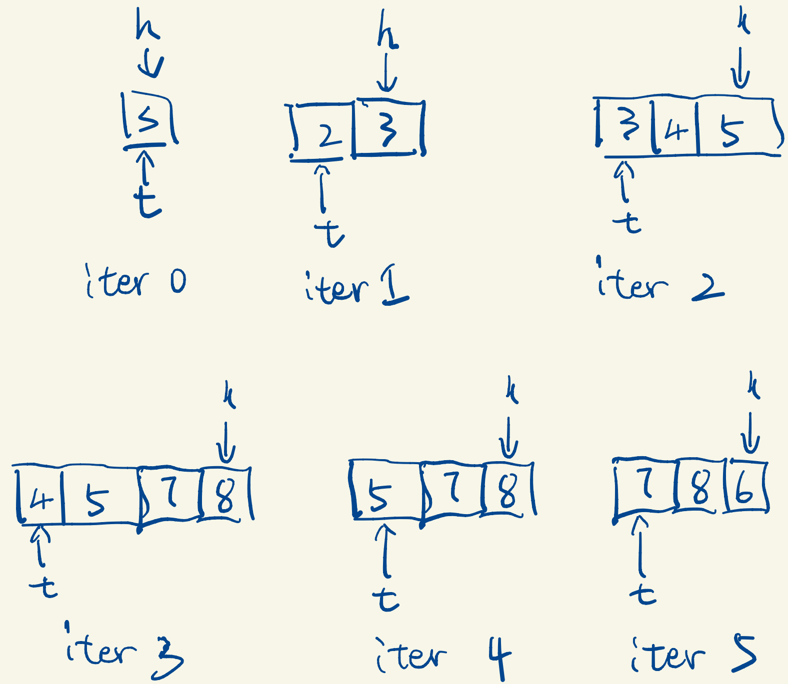


BFS(s)

```
head ← 1, tail ← 1, queue[1] ← s
mark s as "visited" and all other vertices as "unvisited"
while head ≥ tail
    v ← queue[tail], tail ← tail + 1
    for all neighbours u of v
        if u is "unvisited" then
            head ← head + 1, queue[head] = u
            mark u as "visited"
```



Queue:



Running time:

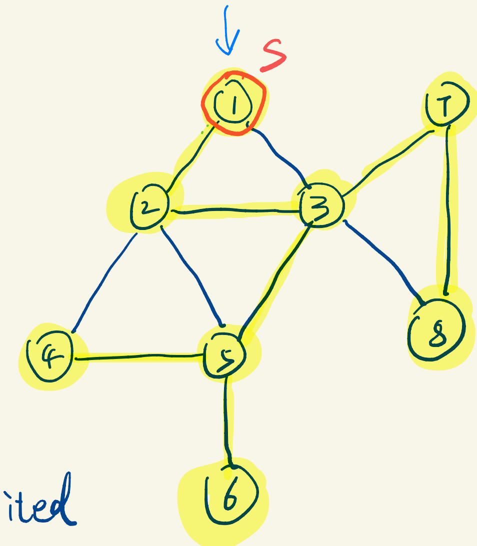
$$O(n+2m) = O(n+m)$$

$m = |E|$, $n = |V|$

▷ Depth-First Search (DFS)

Strategy:

Try to go as deep as you can. If reach a dead end, the backtrak.



▷ Let's say we're visiting vertex v

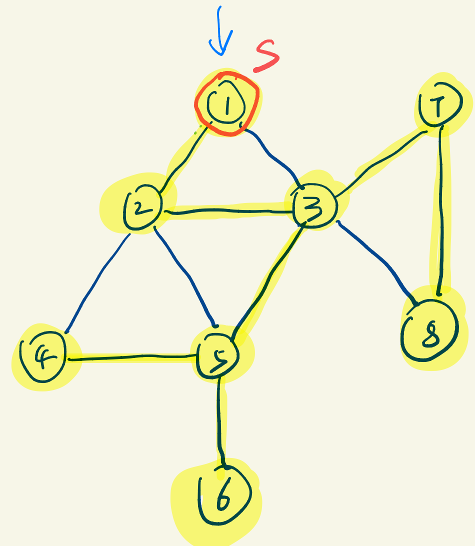
- "can go deeper": if $\exists$ nbr u of v is unvisited
- "dead end": if all its nbrs are visited
- "back track": find the vertex explored before v.

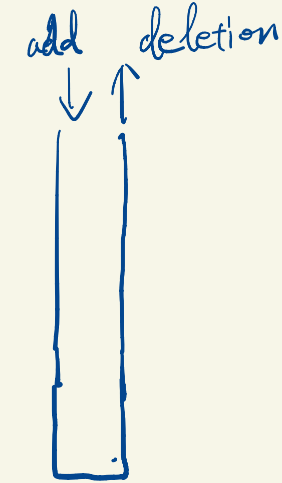
▷ Implementing DFS

s: starting vertex

DFS(s)
  mark all vertices as unvisited.
  recursive-DFS(s)

recursive-DFS(v)
  mark v as visited.
  for each nbr u of v:
      if u is unvisited
          recursive-DFS(u)





Running time:

$O(n+m)$

DFS-iterative(G, v)

    S ← empty stack
    S.push(v)
    while S is not empty:
        u = S.pop()
        if u is not visited:
            mark u as visited
            for all nbrs w of u:
                S.push(w)

u = S.pop() } → 2m times

mark u as visited } → n times

for all nbrs w of u: S.push(w) } → d_u times for each vertex u

$$\leq \sum_u d_u = 2|E| = 2m$$

$$\Rightarrow O(n+4m) = O(n+m)$$