Graphs

$G = (\overset{n}{V}, \overset{m}{E})$ $\quad E \subseteq V \times V$

Adjacency matrix $\quad n \times n$ boolean matrix

$A[i][j] = 1$ iff $(i,j) \in E$

$V = \{1, 2, \ldots, n\}$

Adjacency list

$i \to \text{Neighbours}(i)$

Reachability

Can I reach $\overset{\text{target}}{t}$ from $\overset{\text{source}}{s}$?

Path from s to t

Scan level by level

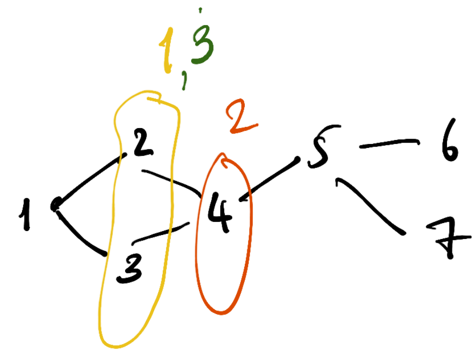
- Vertices reachable in 1 step
- " " " 2 steps

⋮

- Vertices reachable in $n-1$ steps

Breadth first search

explore neighbours of i



Remember which vertices have already been visited

Maintain visited[1..n], $\{0,1\}$ array

Assume $s = 1$

visited[1] = 1

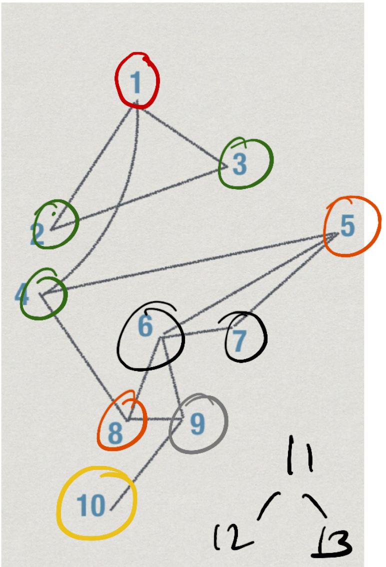
visited[i] = 0 for $i > 1$

explore(j)

  for each edge $(j,k) \in E$

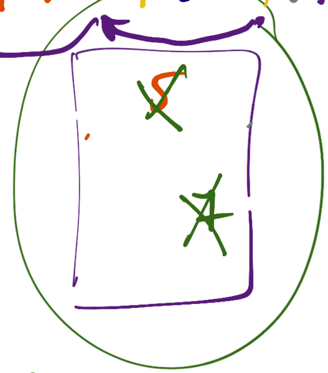
    if not visited[k]

      visited[k] = 1 ← explore(k)



	1	2	3	4	5	6	7	8	9	10
visited	1	1	1	1	1	1	1	1	1	1
parent	-	1	1	1	4	5	5	4	8	9

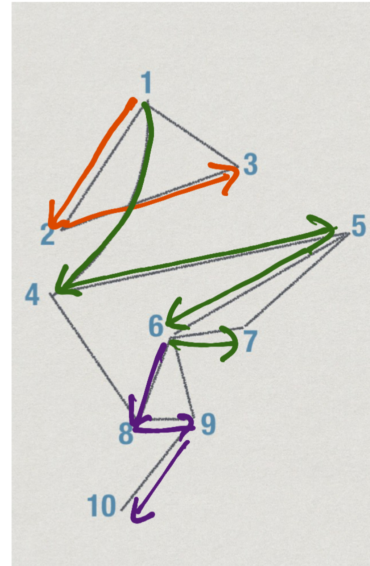
to-explore



Queue

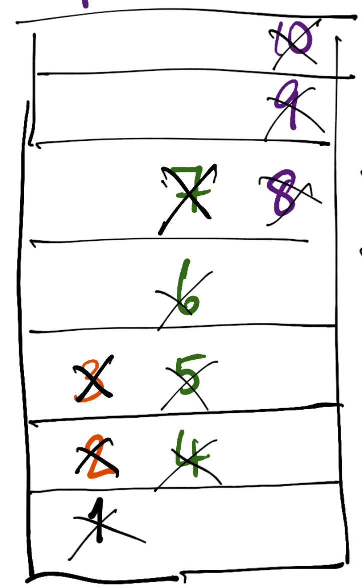
1 step, 2 step, 3 st.

Q 1 2 3 4 5 8 6 7 9 10



Depth first search (dfs)

Stack of pending exploration



visited[i]
parent[i]

Breadth first search

Adj Matrix: $O(n^2)$
Adj list: $O(n+m)$
LINEAR

```
function BFS(i) // BFS starting from vertex i

  //Initialization
  for j = 1..n {visited[j] = 0; parent[j] = -1}
  Q = []

  //Start the exploration at i
  visited[i] = 1; append(Q,i)

  //Explore each vertex in Q
  while Q is not empty
     j = extract_head(Q)
     for each (j,k) in E
        if visited[k] == 0
           visited[k] = 1; parent[k] = j; append(Q,k);
```

$O(n)$

$\sum_j \text{degree}(j) = 2m$

$O(\text{degree}(j))$ adj list

$O(n)$ adj matrix

n times

Depth first search

$O(n^2)$ Adj mat

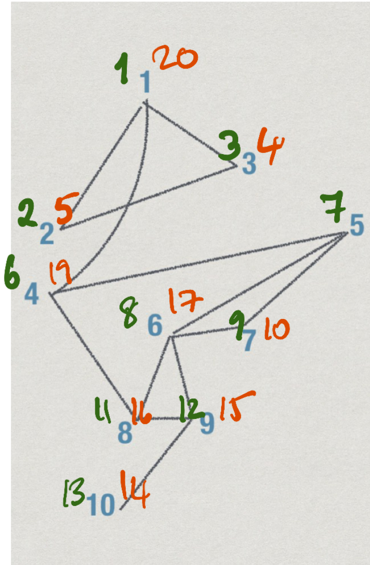
$O(n+m)$ Adj list

$O(n)$

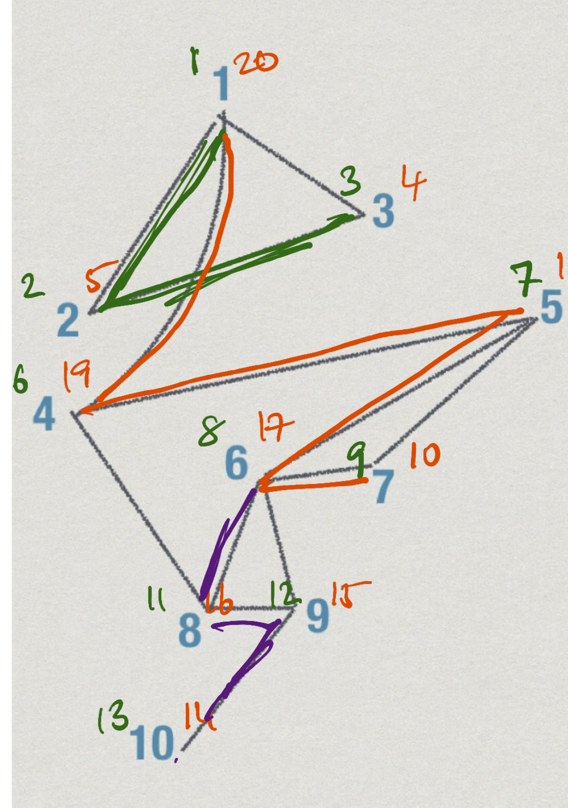
```
//Initialization
  for j = 1..n {visited[j] = 0; parent[j] = -1}

function DFS(i) // DFS starting from vertex i

  //Mark i as visited
  visited[i] = 1

  //Explore each neighbour of i recursively
  for each (i,j) in E
    if visited[j] == 0
      parent[j] = i
      DFS(j)
```

$O(n)$ Adj matrix

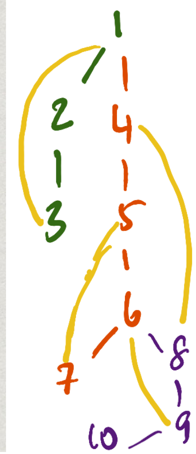
$O(\text{degree}(i))$ Adj list



Augment DFS with info about the order of visiting vertices



Explored edges = tree



Depth first search

```
//Initialization
   for j = 1..n {visited[j] = 0; parent[j] = -1}
   count = 0

function DFS(i) // DFS starting from vertex i

   //Mark i as visited
   visited[i] = 1; pre[i] = count; count++

   //Explore each neighbours of i recursively
   for each (i,j) in E
      if visited[j] == 0
         parent[j] = i
         DFS(j)
         post[i] = count; count++
```

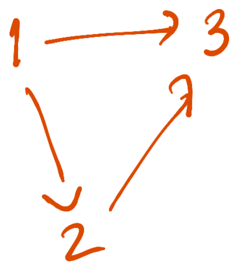
Tree

Connected graph without cycles

n vertices — n-1 edges

Unique path from any u to any v

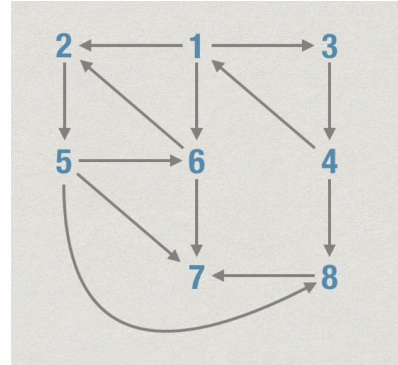
Directed graph

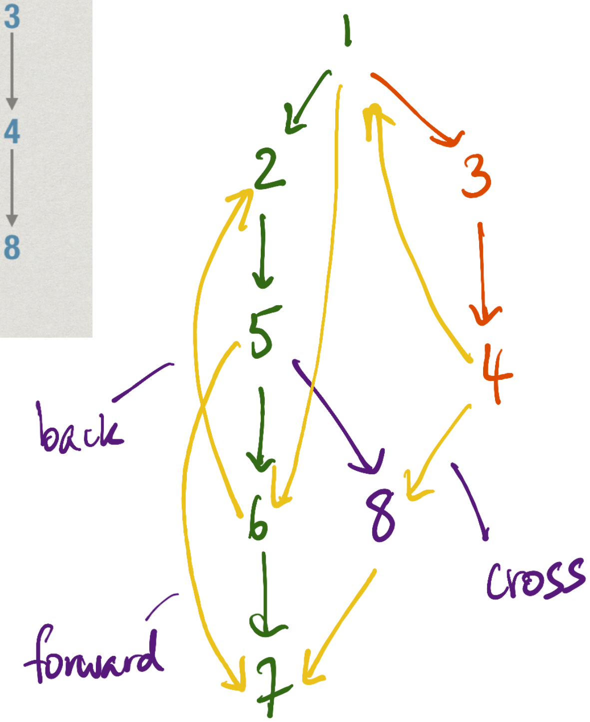


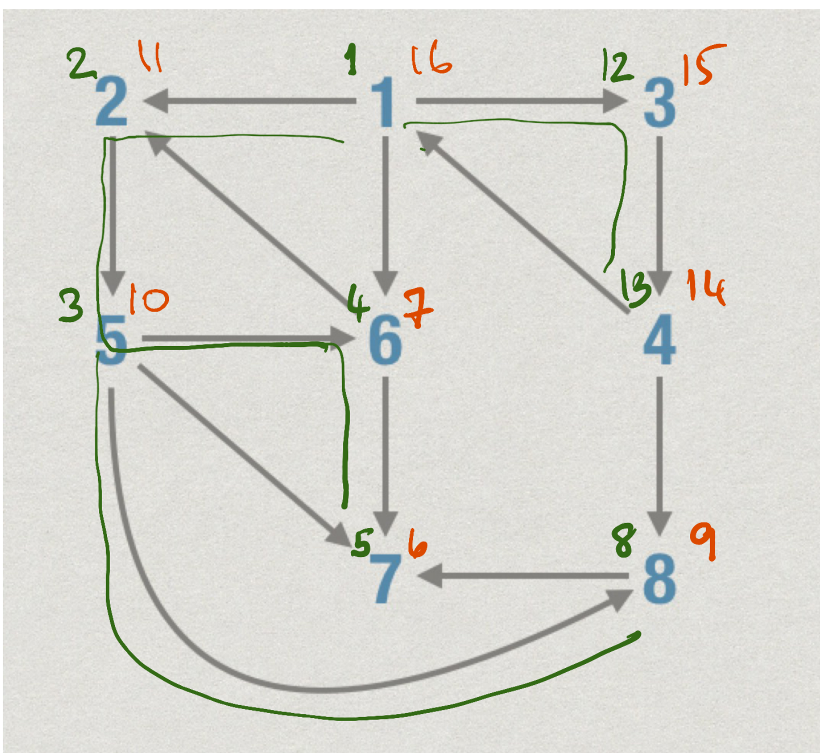
BFS(2)

	1	2	3
Visited	0	1	~~0~~ 1

~~2~~ ~~3~~

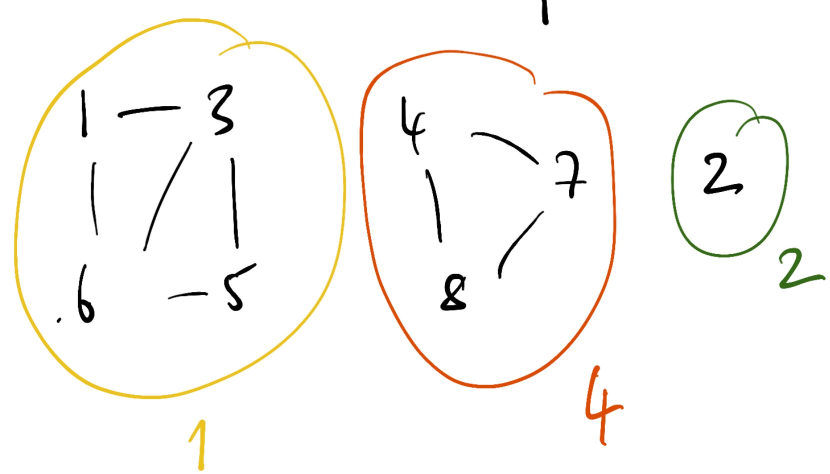






Applications of BFS & DFS

Undirected graph - Find Connected Components



Directed graph

Strongly Connected Component

Clever algo using DFS numbering