CSCI 311

Algorithms and Data Structures

CSU Chico

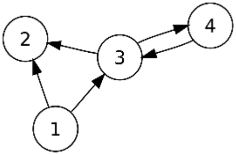
Week 11.01

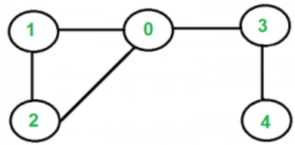
Today

- Graphs
  - Notation
  - Types of graphs
  - Representations

- Breadth-first search (BFS)
- Depth-first search (DFS)
- Edge classification

Graphs
Notation



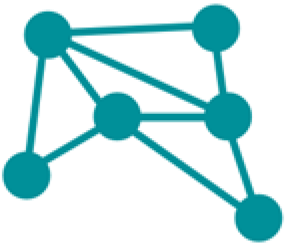


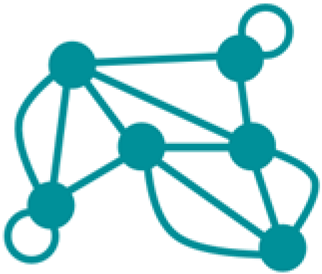
- $G = (V, E)$
  - V - a set of nodes or vertices, often labelled $1, \ldots, n$
  - E - a set of edges
- Edges can be undirected, $\{u, v\}$, or directed, (u, v)
- Vertices and edges can have attributes
  - $v.id$, $v.cost$, $v.dist$ where $v \in V$
  - $(u, v).weight$, $\{u, v\}.dist$ where $(u, v) \in E$ or $\{u, v\} \in E$
- Run time often measured as a function of $|V|$ and $|E|$

Graphs

Types

- G is a **simple** graph if
  - Edges are undirected and unweighted
  - There are no multi-edges (no same edge)
  - There are no self loops (no edge starts and ends at the same vertex)



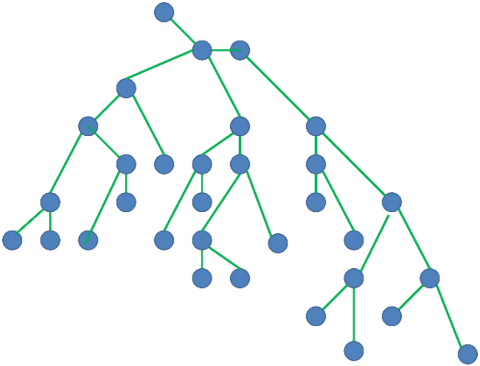


Graphs

Types

- G is a **tree** if

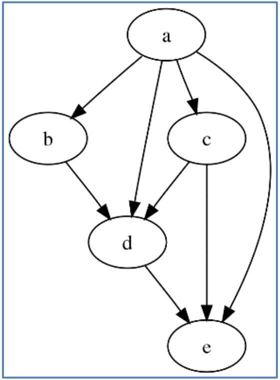
  For every pair $u, v \in V$ is connected by exactly one path

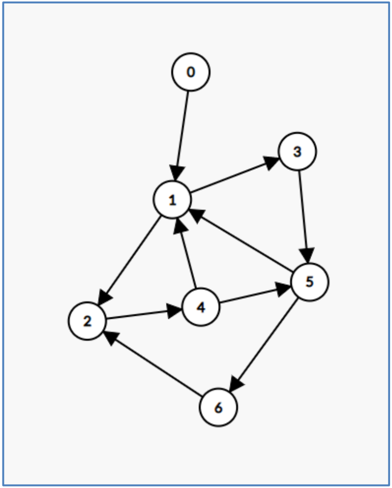


Graphs

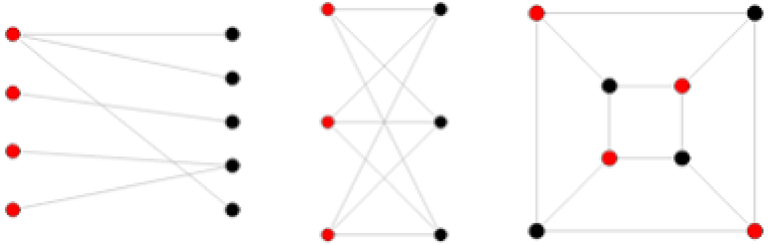
Types

- G is a **directed acyclic graph (DAG)** if directed but with no cycles



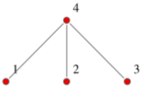


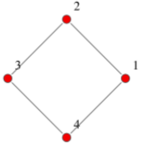
- G is a **bipartite graph** if
  - Two disjoint sets of vertices U and V
  - For all edges $\{u, v\} \in E$, u and v are in different vertex sets

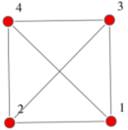


Graphs

Representations





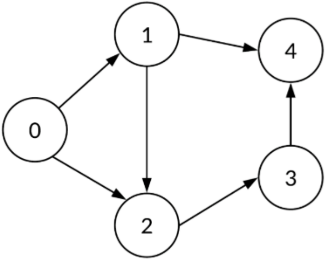


$$\begin{pmatrix} 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{pmatrix} \quad \begin{pmatrix} 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 \end{pmatrix} \quad \begin{pmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{pmatrix}$$

- Two sets V and E
- Adjacency matrix

  A is an $n \times n$ matrix

  $\mathbf{A}[i][j] = 1$ if $(i, j) \in E$ and 0 otherwise

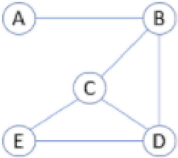


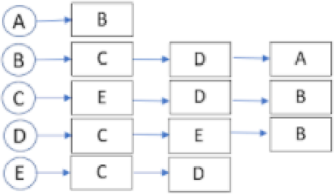
Adjacency Matrix

	0	1	2	3	4
0	0	1	1	0	0
1	0	0	1	0	1
2	0	0	0	1	0
3	0	0	0	0	1
4	0	0	0	0	0

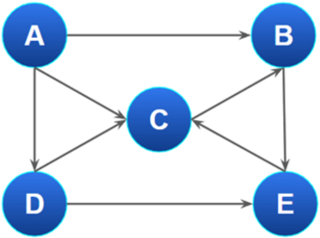
Graphs

Representations





- Two sets V and E
- Adjacency list
  - A is a list of n lists
  - $j \in A[i]$ if and only if $(i, j) \in E$



A	B, C, D
B	E
C	B
D	C, E
E	C

Breadth-First Search

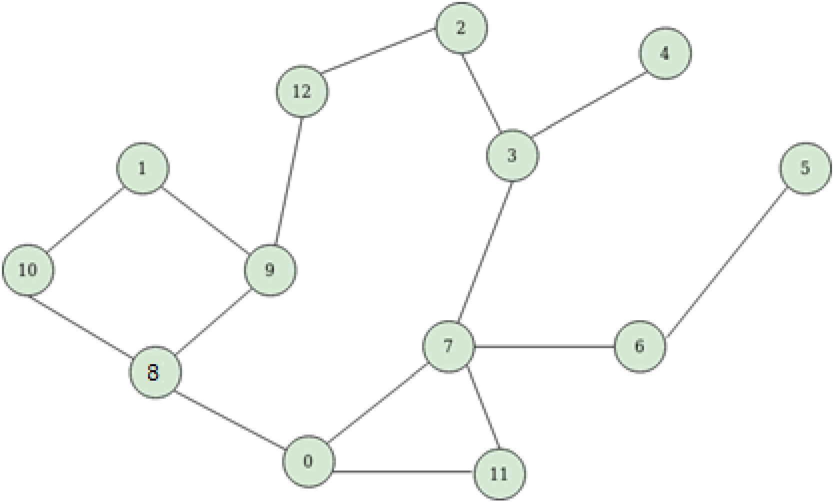
Intuition

Given a graph $G = (V, E)$ and a source node $s \in V$

1. Add s to the (initially empty) *work* queue
2. Mark s as visited
3. While this queue is not empty
   1. Remove the first node
   2. Mark this node as visited
   3. Add all of its unvisited neighbors to the queue

Breadth-First Search

Example



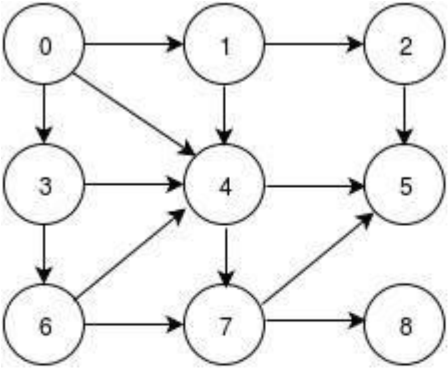
Breadth-First Search

Pseudocode (also calculating the distances)

```
function BFS(G, s)
  for u in V
    u.dist = ∞
    u.visited = false
    u.predecessor = null
  s.dist = 0
  s.visited = true
  Q = queue()
  Q.push(s)
  while length(Q) > 0
    u = Q.pop()
    for v in neighbors(G, u)
      if v.visited == false
        v.dist = u.dist +1
        v.visited = true v.
        predecessor = u Q.
        push(v)
```

Breadth-First Search

Example



Breath-First Search

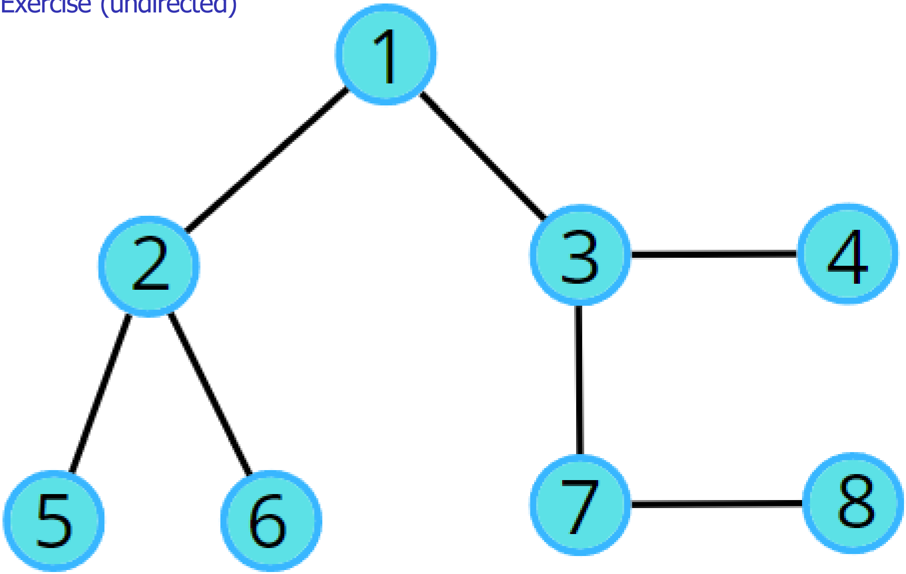
Keep Track of Distance

```
Node{
  dist
  visited
  predecessor
}
```

```
function BFS(G, s)
  for u in V
    u.dist = ∞
    u.visited = false
    u.predecessor = null
  s.dist = 0
  s.visited = true
  Q = queue()
  Q.push(s)
  while length(Q) > 0
    u = Q.pop()
    for v in neighbors(G, u)
      if v.visited == false
        v.dist = u.dist +1
        v.visited = true v.
        predecessor = u
        Q.push(v)
```

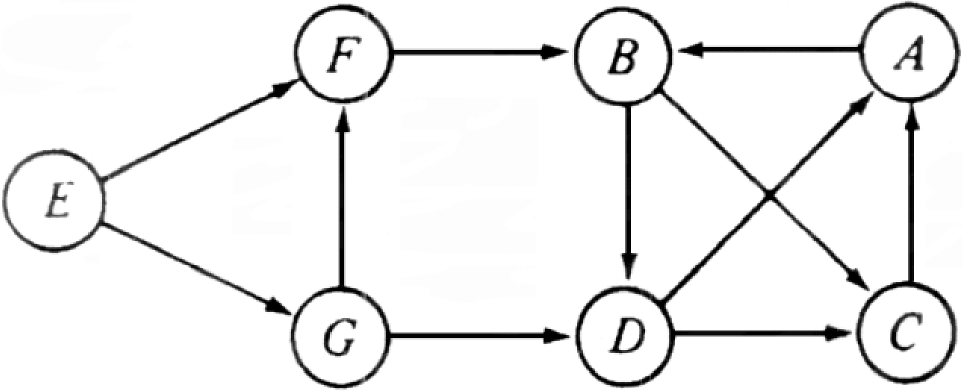
BFS-Undirected-Keep-Distance

Exercise (undirected)

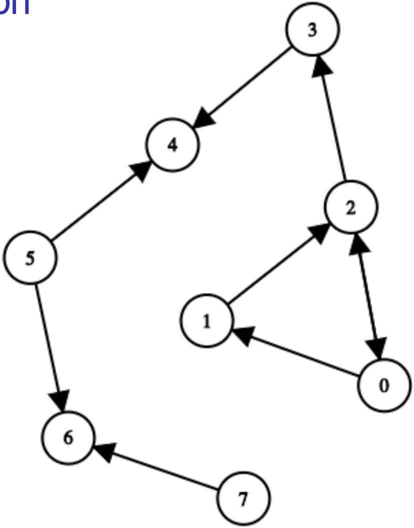


BFS-Directed-Keep-Distance

Exercise (directed)

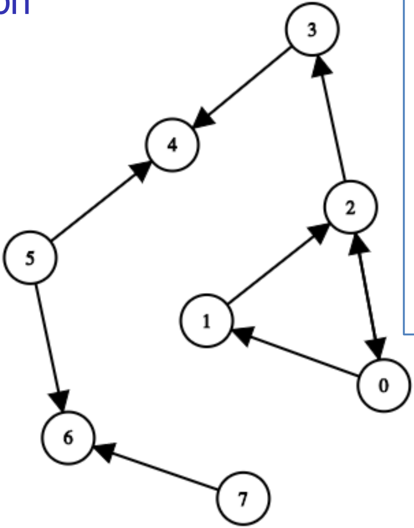


BFS on Disconnected Graph




```
function BFS(G, s)
  for u in V
    u.dist = ∞
    u.visited = false
    u.predecessor = null
  s.dist = 0
  s.visited = true
  Q = queue()
  Q.push(s)
  while length(Q) > 0
    u = Q.pop()
    for v in neighbors(G, u)
      if v.visited == false
        v.dist = u.dist +1
        v.visited = true v.
        predecessor = u Q.
        push(v)
```

BFSN on Disconnected Graph




```
Function BFSN(G)
  for u in V
    u.dist = ∞
    u.visited = false
    u.predecessor = null
  While any s in V is not visited
   s.dist = 0
   s.visited = true
   Q = queue()
   Q.push(s)
   while length(Q) > 0
    u = Q.pop()
    for v in neighbors(G, u)
      if v.visited == false
        v.dist = u.dist+1
        v.visited = true
        v.predecessor = u Q.push(v)
```

Depth-First Search (DFSVisit)

Intuition

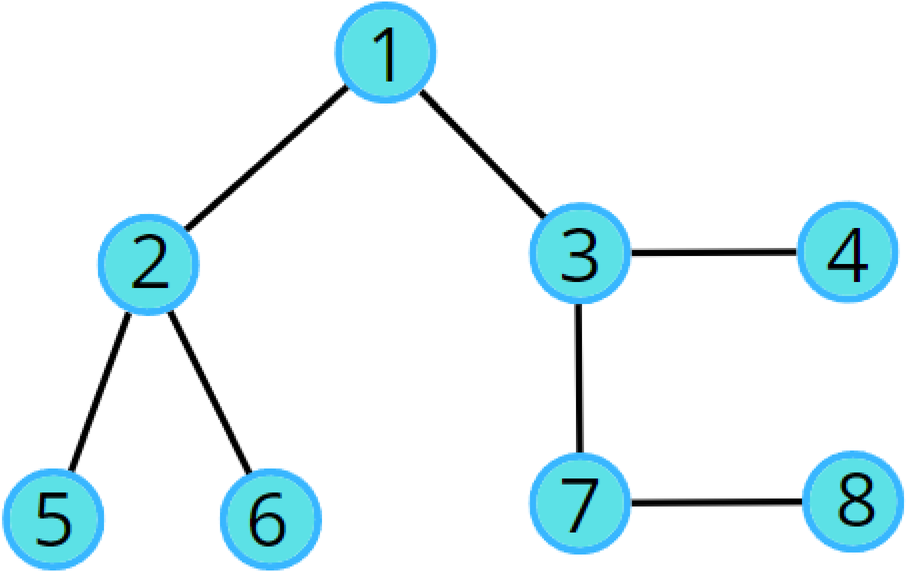
Given a graph $G = (V, E)$

1. Pick a source node s
2. Mark s as visited
3. for all neighbors u of s
    1. If u has not been visited, run depth first search on G with u as the source
4. Repeat until all vertices have been visited

Lab 7 Challenge DFS Discover & Finish Times

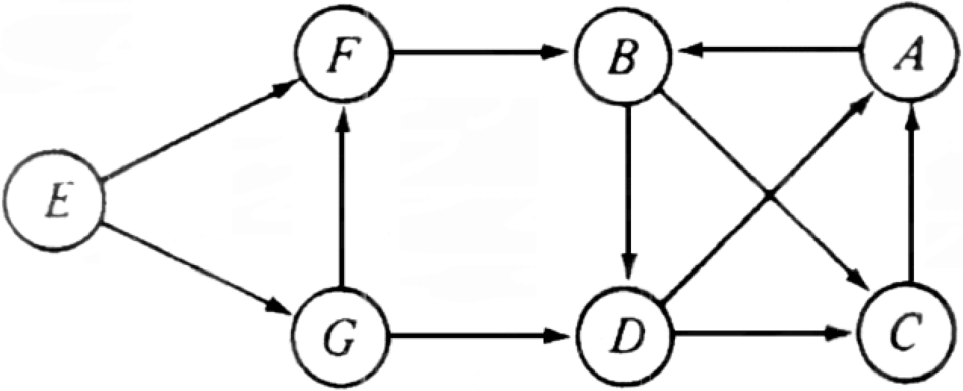
DFS-Undirected

Example (Discovery & Finish Time)

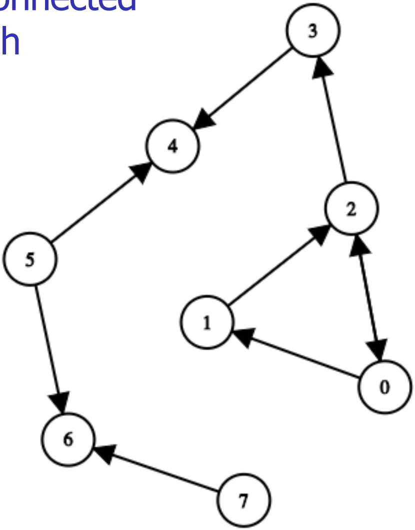


DFS-Directed

Example (Discovery & Finish Time)



DFS on Disconnected Graph




```
function DFS(G)
  for u in V
    u.visited = false
    u.predecessor = null
    u.discovered = -1
    u.finished = -1
  time = 0
  for u in V
    if u.visited == false
      time = DFSVisit(G, u, time)
```

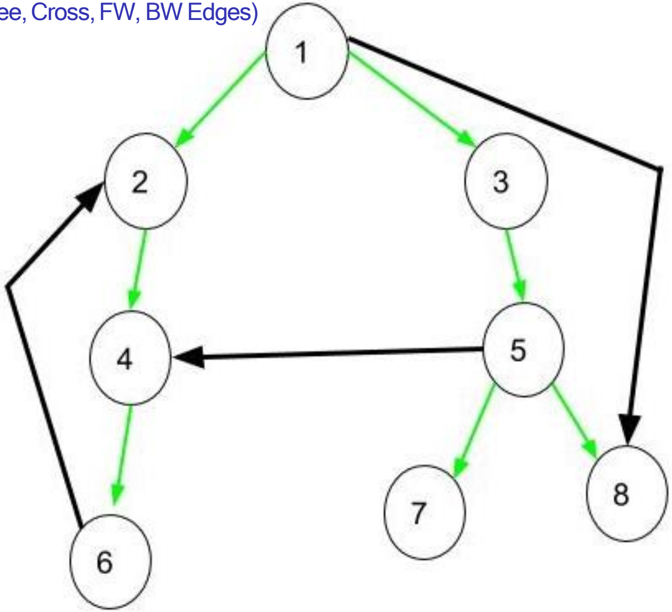
```
function DFSVisit(G, u, time)
  time = time + 1
  u.discovered = time
  u.visited = true
  for v in neighbors(G, u)
    if v.visited = false
      v.predecessor = u
      time = DFSVisit(G, v, time)
  time = time + 1
  u.finished = time
  return time
```

Classification of Edges (Directed)

- Let $G = (V, E)$, $s \in V$, and T_s be a D(B)FS tree
- Tree edge
  - $(u, v) \in E$ is a tree edge if $\{u, v\}$ is in T_s
  - v was visited following $(u, v) \in E$
- Back edge
  - $(u, v) \in E$ is a back edge if it is not a tree edge and v is an ancestor of u in T_s
  - u is visited as a result of the call `DFS(G, v)`
- Forward edge
  - $(u, v) \in E$ is a forward edge if it is not a tree edge and v is a descendent of u in T_s
  - v is visited as a result of the call `DFS(G, u)`
- Cross edge
  - All other edges

DFS Tree & Edges

Example (Tree, Cross, FW, BW Edges)

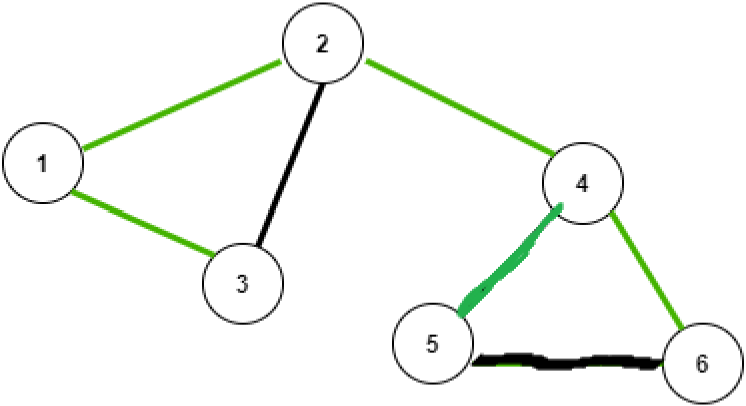


Classification of Edges (Undirected)

- Let $G = (V, E)$, $s \in V$, and T_s be a B(D)FS
- tree Tree edge
  - $(u, v) \in E$ is a tree edge if $\{u, v\}$ is in T_s
  - v was visited following $(u, v) \in E$
- Cross edge
  - All other edges

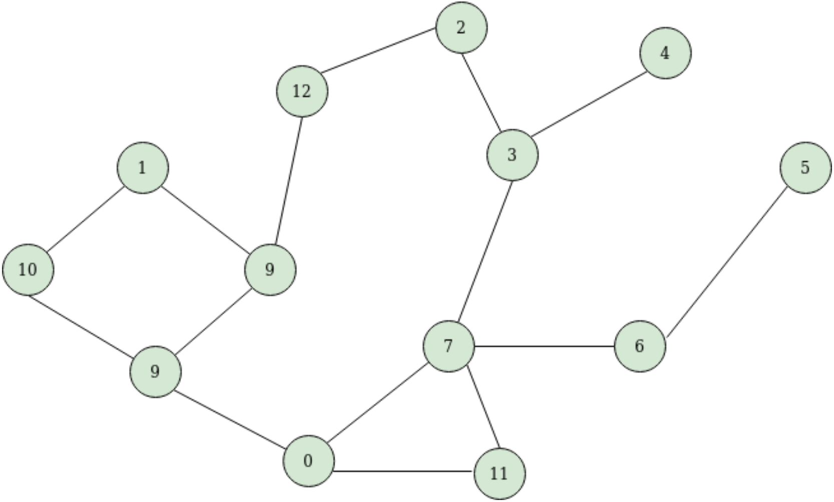
BFS Tree & Edges

Example (Tree & Cross Edges)



Depth-First Search (DFSVisit)

Example (Undirected) - Classification of Edges



Depth-First Search (DFSVisit)

Example (Directed) - Classification of Edges