Lecture - 7 Graphs Algorithms

CS313E - Elements of Software Design

Kia Teymourian

05/11/2022

Agenda

Graphs

Graph data are present in many applications, for example the following applications are dealing with graph data:

- Web Crawling and Web search
- Social Network, e.g. Friends-Of-Friend network (Goal Community detection)
- Computer Networks.
- Reference Counting in Memory Garbage Collection (based on graphs between allocated memories)

Graph Representations

A graph has two ingredients, Vertices and Edges.

- ▷ ***V* = A Set Vertices** (Vertex singular)
- ▷ ***E* = A Set of Edges**, each edge is a vertex pair (v, w).

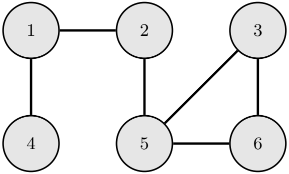


Figure: Representations of an undirected graph G with 6 vertices and 6 edges.

Directed and Undirected Graphs

- **Directed Graph** has edges that are ordered pair of vertices.
- **Undirected Graph** has edges that are unordered pair of vertices.

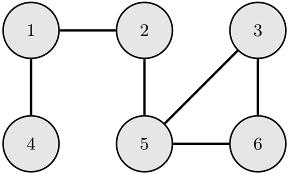


Figure: An Example Undirected Graph G with 6 vertices and 6 edges.

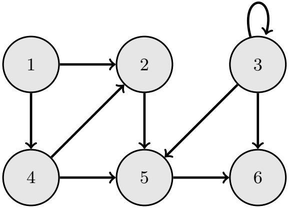
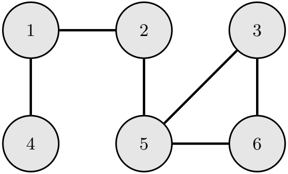


Figure: An example directed graph G with 6 vertices and 9 edges

Adjacency-matrix Representation of a Graph



$$
\begin{array}{c|cccccc|}
 & 1 & 2 & 3 & 4 & 5 & 6 \\
\hline
1 & 0 & 1 & 0 & 1 & 0 & 0 \\
2 & 1 & 0 & 0 & 0 & 1 & 0 \\
3 & 0 & 0 & 0 & 0 & 1 & 1 \\
4 & 1 & 0 & 0 & 0 & 0 & 0 \\
5 & 0 & 1 & 1 & 0 & 0 & 1 \\
6 & 0 & 0 & 1 & 0 & 1 & 0 \\
\hline
\end{array}
$$

Figure: Representations of an undirected graph G with 6 vertices and 6 edges using adjacency-matrix representation of G.

- One disadvantage of adjacency-matrix representation is that it requires a large memory space to store it.
- If the graph has $|V|$ number of vertices, it requires $\Theta(n^2)$ memory space for storage.

Adjacency-list representation

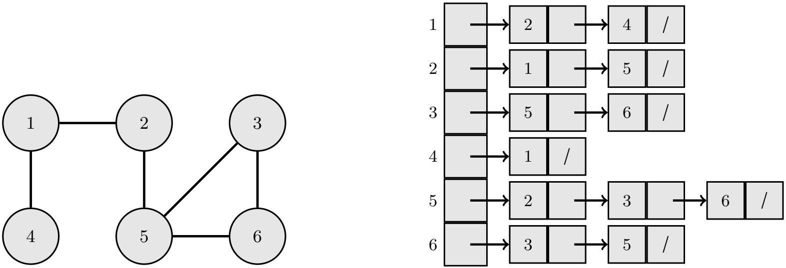
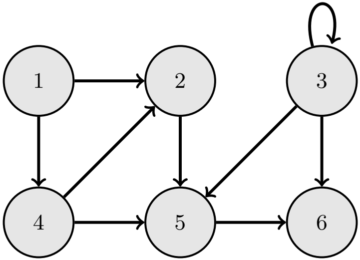


Figure: Representations of an undirected graph G with 6 vertices and 6 edges using an adjacency-list representation of G.

- The required memory space for adjacency list storage is $\Theta(V+E)$, the total sum of the number of edges and vertices.
- In python programing, we can store it adjacency lists in a simple dictionary of list/set values.
- Vertex can be any hashable object in python for example integer or tuple. One advantage is that we can store multiple graphs on the same vertices set.

Adjacency-Matrix representation of the directed Graph G.

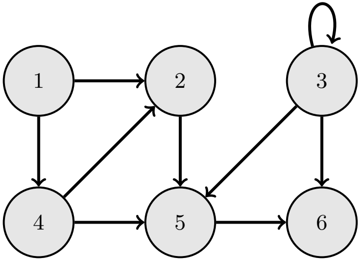


	1	2	3	4	5	6
1	0	1	0	1	0	0
2	0	0	0	0	1	0
3	0	0	1	0	1	1
4	0	1	0	0	1	0
5	0	0	0	0	0	1
6	0	0	0	0	0	0

Figure: The adjacency-matrix representation of the directed Graph G.

- In the adjacency matrix, we start from rows of the matrix, matrix values equal to 1 define that there is a an edge between the vertices.

Adjacency-List representation of the directed Graph G.



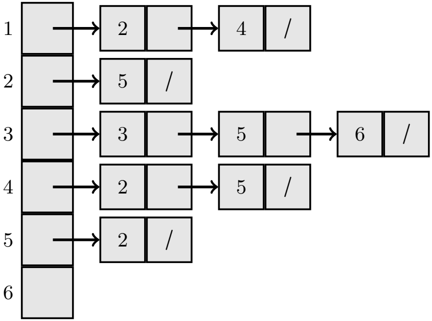


Figure: An adjacency-list representation of the directed Graph G.

- ▷ In the adjacency list the order defines the direction of the edges.

Graph Search Problem

Sometimes we have applications that require to explored the entire graph.
It means to ...

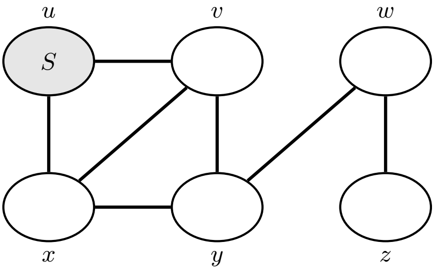
- find a path from a start vertex S to a desired destination vertex.
- visit all vertices or edges of a graph , or visit only a subset that can be reached from start S.

Breadth-First Search (BFS)

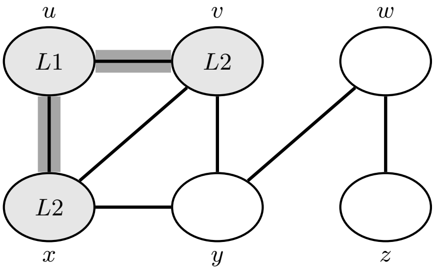
The goal of the Breath-First-Search (BFS) Algorithm is to explore the entire graph level-by-level from a start vertex S

- Start point is vertex $\boldsymbol{S}$
- Start $\boldsymbol{level} = \{\boldsymbol{S}\}$ initialization of the level set.
- Next, find out which other vertices can be reached from the start.
  $\boldsymbol{level_i} = \{$**All reachable edges with one step**$\}$
- Build the next level by using all outgoing edges, and ignoring visited vertices from previous levels.

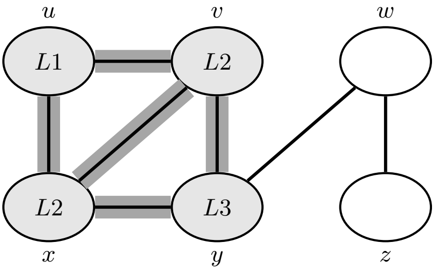
Example - Breadth-First Search (BFS) - Start Level 1



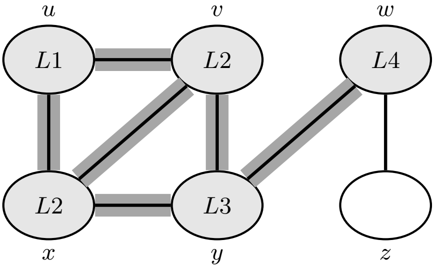
Example - Breadth-First Search (BFS) - Level 2



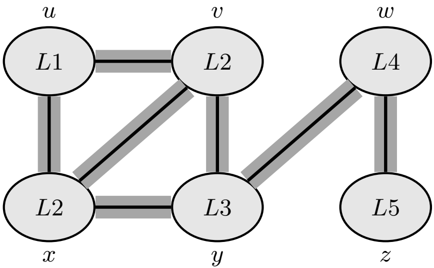
Example - Breadth-First Search (BFS) - Level 3



Example - Breadth-First Search (BFS) - Level 4



Example - Breadth-First Search (BFS) - Level 5



```
from collections import defaultdict
# Function to print a BFS of graph
def BFS(s, adj):
    i = 1  # set the start level to 1
    level = defaultdict(list) # A dict for levels of our
        visits
    # A queue for BFS
    frontier = []

    # Mark the source node as
    frontier.append(s)
    level[s] = 1

    while frontier:
    # Get the frontier and print it.
    s = frontier.pop(0)
    print(s, end = " ")

    # Get all adjacent vertices of the
    # If it is not been visited, has no levels
    for i in adj[s]:
        if i not in level:
        frontier.append(i)
        level[i] = i
    i += 1 # increment the level up
    print("\nLevels are:", dict(level), "\n")
```

Listing 1: BSF in Python

BSF Python RUN

```
########################################
####    RUN
########################################

graph = defaultdict(list)

graph[0].append(1) # 0 --> 1
graph[0].append(2) # 0 --> 2
graph[2].append(3) # 2 --> 3
graph[1].append(3) # 1 --> 3
graph[3].append(4) # 3 --> 4

BFS(1, graph)

1 3 4
Levels are: {1: 1, 3: 3, 4: 4}
```

Listing 2: Run of BSF

Algorithm 1 BFS(G,s)

```
for each vertex u ∈ G.V − {s} do
    u.color = WHITE
    u.d = ∞
    u.π = NIL
end for
s.color = GRAY, s.d = 0, s.π = NIL
Q = ∅
ENQUEUE(Q, s)
while Q ≠ ∅ do
    u = DEQUEUE(Q)
    for each v ∈ G.Adj[u] do
        if v.color == WHITE then
            v.color = GRAY
            v.d = u.d + 1
            v.π = u
            ENQUEUE(Q, v)
        end if
    end for
    u.color = BLACK
end while
```

Analysis of BFS

- Each vertex the set $\boldsymbol{V}$ only one time, and as next for the frontier only once. Because we ignore the visited vertices in each level. Base case is $V = S$ the start position.
- This means that we loop through the adjacency list $adj[V]$ only once.
  $\boldsymbol{time = \sum_{v \in V} |adj[v]|}$ will be $|E|$ number of edges for the directed graph and $2|E|$ for undirected graphs
  This results in $O(E)$ time
- We write $\boldsymbol{O(V+E)}$ to also include list of vertices unreachable from v (not assigned levels).
- **This will be in total LINEAR TIME.**

Shortest Path

The shortest path means that we are looking to find for every vertex v, the fewest edges to get from an start vertex s to v

$$\boldsymbol{shortestPath(s,v)} = \begin{cases} \boldsymbol{level[v]} & \textbf{if } \boldsymbol{v} \textbf{ assigned level} \\ \boldsymbol{\infty} & \textbf{if there is no path} \end{cases}$$

The $\boldsymbol{level[v]}$ provides the number of edges to get from a start vertex s to a vertex v.

To find the shortest path, the path is:

- we need to take v and
- $parent[v]$ parent of v and
- $parent[parent[v]]$ parent of parent of v and
- and so on, until start vertex s is reached or nothing reaching (No paths exists)

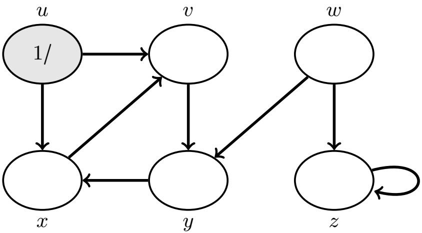
Depth-First Search (DFS)

- The goal is to explore a Graph G.
- For example to find a path from start vertex S to a destination vertex v.
- The depth-First Search algorithm execution is similar to exploring a maze, you would follow up a path until you get stuck and then go back to find a new path.

We can summarize the steps as follows:

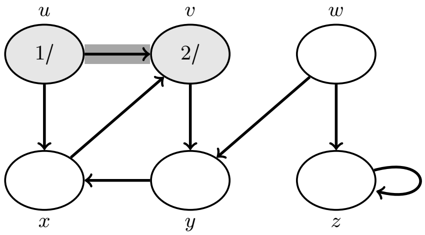
1. Follow up a path until there is no more paths to follow.
2. Backtrack along breadcrumbs until we are back to a point that we have unexplored neighbor
3. Recursive explore
4. Mark the vertices to not repeat a vertex.

Example - Depth-First Search (DFS)



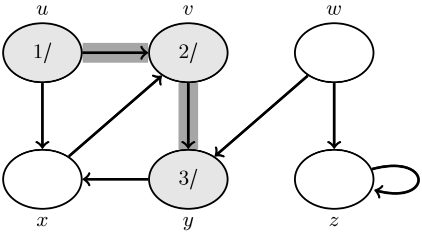
- As edges are explored by the algorithm, they are shown as either shaded (if they are tree edges) or dashed (otherwise).
- Nontree edges are labeled B, C, or F according to whether they are back, cross, or forward edges.
- Timestamps within vertices indicate discovery time/finishing times

Example - Depth-First Search (DFS)



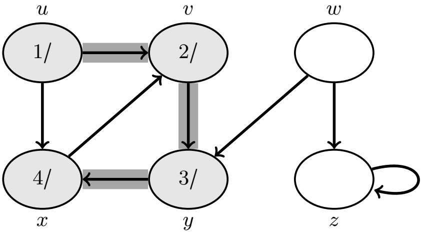
- As edges are explored by the algorithm, they are shown as either shaded (if they are tree edges) or dashed (otherwise).
- Nontree edges are labeled B, C, or F according to whether they are back, cross, or forward edges.
- Timestamps within vertices indicate discovery time/finishing times

Example - Depth-First Search (DFS)



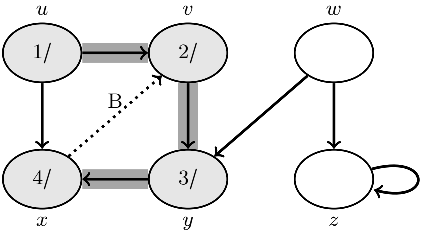
- As edges are explored by the algorithm, they are shown as either shaded (if they are tree edges) or dashed (otherwise).
- Nontree edges are labeled B, C, or F according to whether they are back, cross, or forward edges.
- Timestamps within vertices indicate discovery time/finishing times

Example - Depth-First Search (DFS)



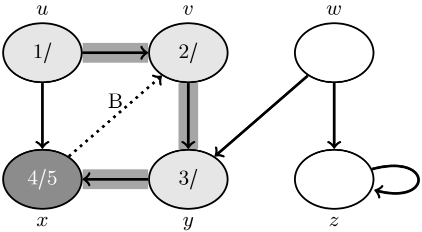
- As edges are explored by the algorithm, they are shown as either shaded (if they are tree edges) or dashed (otherwise).
- Nontree edges are labeled B, C, or F according to whether they are back, cross, or forward edges.
- Timestamps within vertices indicate discovery time/finishing times

Example - Depth-First Search (DFS)



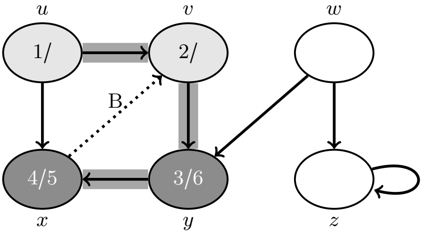
- As edges are explored by the algorithm, they are shown as either shaded (if they are tree edges) or dashed (otherwise).
- Nontree edges are labeled B, C, or F according to whether they are back, cross, or forward edges.
- Timestamps within vertices indicate discovery time/finishing times

Example - Depth-First Search (DFS)



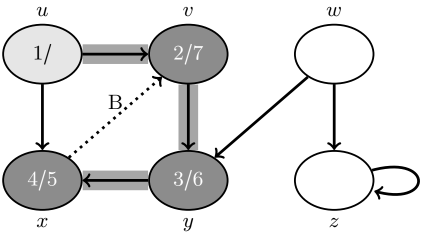
- As edges are explored by the algorithm, they are shown as either shaded (if they are tree edges) or dashed (otherwise).
- Nontree edges are labeled B, C, or F according to whether they are back, cross, or forward edges.
- Timestamps within vertices indicate discovery time/finishing times

Example - Depth-First Search (DFS)



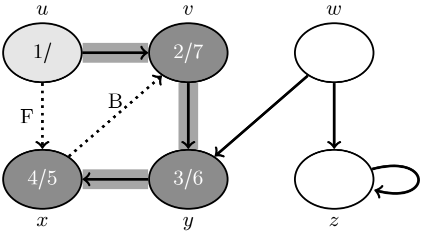
- As edges are explored by the algorithm, they are shown as either shaded (if they are tree edges) or dashed (otherwise).
- Nontree edges are labeled B, C, or F according to whether they are back, cross, or forward edges.
- Timestamps within vertices indicate discovery time/finishing times

Example - Depth-First Search (DFS)



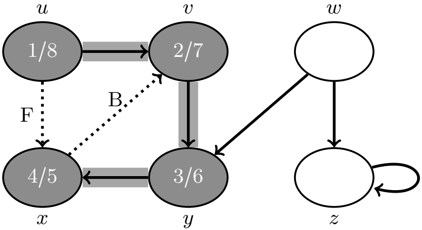
- As edges are explored by the algorithm, they are shown as either shaded (if they are tree edges) or dashed (otherwise).
- Nontree edges are labeled B, C, or F according to whether they are back, cross, or forward edges.
- Timestamps within vertices indicate discovery time/finishing times

Example - Depth-First Search (DFS)



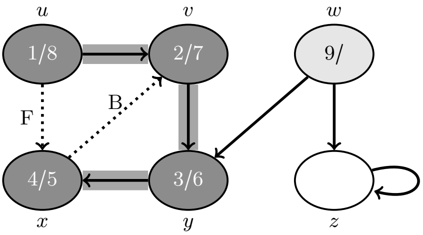
- As edges are explored by the algorithm, they are shown as either shaded (if they are tree edges) or dashed (otherwise).
- Nontree edges are labeled B, C, or F according to whether they are back, cross, or forward edges.
- Timestamps within vertices indicate discovery time/finishing times

Example - Depth-First Search (DFS)



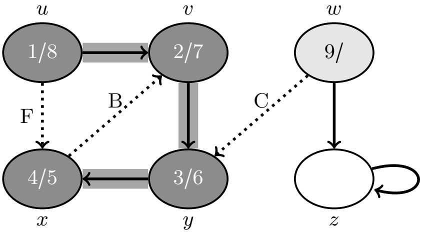
- As edges are explored by the algorithm, they are shown as either shaded (if they are tree edges) or dashed (otherwise).
- Nontree edges are labeled B, C, or F according to whether they are back, cross, or forward edges.
- Timestamps within vertices indicate discovery time/finishing times

Example - Depth-First Search (DFS)



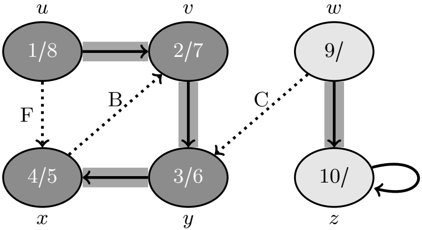
- As edges are explored by the algorithm, they are shown as either shaded (if they are tree edges) or dashed (otherwise).
- Nontree edges are labeled B, C, or F according to whether they are back, cross, or forward edges.
- Timestamps within vertices indicate discovery time/finishing times

Example - Depth-First Search (DFS)



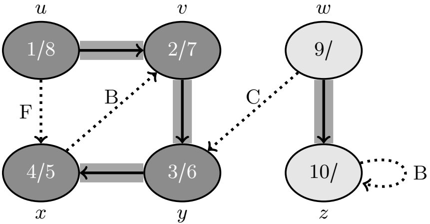
- As edges are explored by the algorithm, they are shown as either shaded (if they are tree edges) or dashed (otherwise).
- Nontree edges are labeled B, C, or F according to whether they are back, cross, or forward edges.
- Timestamps within vertices indicate discovery time/finishing times

Example - Depth-First Search (DFS)



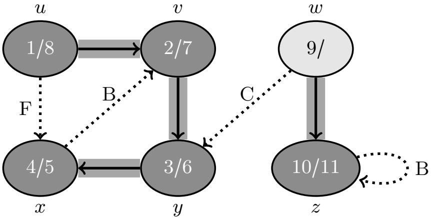
- As edges are explored by the algorithm, they are shown as either shaded (if they are tree edges) or dashed (otherwise).
- Nontree edges are labeled B, C, or F according to whether they are back, cross, or forward edges.
- Timestamps within vertices indicate discovery time/finishing times

Example - Depth-First Search (DFS)



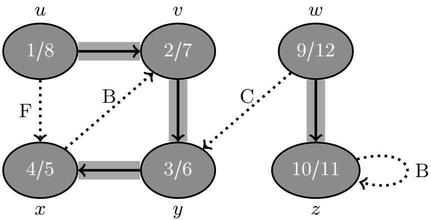
- As edges are explored by the algorithm, they are shown as either shaded (if they are tree edges) or dashed (otherwise).
- Nontree edges are labeled B, C, or F according to whether they are back, cross, or forward edges.
- Timestamps within vertices indicate discovery time/finishing times

Example - Depth-First Search (DFS)



- As edges are explored by the algorithm, they are shown as either shaded (if they are tree edges) or dashed (otherwise).
- Nontree edges are labeled B, C, or F according to whether they are back, cross, or forward edges.
- Timestamps within vertices indicate discovery time/finishing times

Example - Depth-First Search (DFS)



- As edges are explored by the algorithm, they are shown as either shaded (if they are tree edges) or dashed (otherwise).
- Nontree edges are labeled B, C, or F according to whether they are back, cross, or forward edges.
- Timestamps within vertices indicate discovery time/finishing times

DFS

Algorithm 2 DFS

```
1: for each vertex u ∈ G.V do
2:     u.color = WHITE
3:     u.π = NIL
4: end for
5: time = 0
6: for each vertex u ∈ G.V do
7:     if u.color == WHITE then
8:         DFS-VISIT(G,u)
9:     end if
10: end for
```

- Lines 1–3: Paint all vertices white and initialize their π attributes to NIL.
- Line 5: Reset the global time counter.
- Lines 6–10: Check each vertex in V, when white, visit it using DFS-VISIT().

DFS-VISIT(G, u)

DFS-VISIT(G, u) visits the vertex u. Vertex u becomes the root of a new tree in the depth-first forest.

Algorithm 3 DFS-VISIT(G, u)

$time = time + 1$ ▷ increments the global variable time
$u.d = time$ ▷ records the new value of time as the discovery time u:d
$u.color = GRAY$ ▷ paints u gray
for each $v \in G.Adj[u]$ **do** ▷ Explore edge (u, v)
    if $v.color == WHITE$ **then** ▷ Examine each vertex v adjacent to u
        $v.\pi = u$
        $DFS-VISIT(G, v)$ ▷ Recursively visit v if it is white
    end if
end for
$u.color = BLACK$ ▷ blacken u; it is finished
$time = time + 1$
$u.f = time$

Every vertex u has been assigned a discovery time $u.d$ and a finishing time $u.f$.

Analysis of DFS

- We call DFS-VISIT with a vertex S only once because $parent[s]$ is set. This results that the time in DFS-VISIT is $\boldsymbol{\sum_{s \in V} |Adj[s]| = O(E)}$
- DFS outer loop adds just another linear time into it $O(V)$ which results that total run time be $\boldsymbol{O(V + E)}$ **Linear Time**.

Simulation Links

- Depth-First Search (DFS)
  `https://www.cs.usfca.edu/~galles/visualization/DFS.html`
- `https://visualgo.net/en/dfsbfs?slide=1`

Readings from CLRS Book (Introduction to Algorithms, 3rd Edition)

Chapter 22 Graphs

- Section 22.1 Representations of graphs
- Section 22.2 Breadth-first search
- Section 22.3 Depth-first search